S3C2800

32-BIT RISC

MICROPROCESSOR

USER'S MANUAL

Revision 1.1

SAMSUNG

ELECTRONICS

Important Notice

The information in this publication has been carefully checked and is believed to be entirely accurate at the time of publication. Samsung assumes no responsibility, however, for possible errors or omissions, or for any consequences resulting from the use of the information contained herein.

Samsung reserves the right to make changes in its products or product specifications with the intent to improve function or design at any time and without notice and is not required to update this documentation to reflect such changes.

This publication does not convey to a purchaser of semiconductor devices described herein any license under the patent rights of Samsung or others.

Samsung makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Samsung assume any liability arising out of the application or use of any product or circuit and specifically disclaims any and all liability, including without limitation any consequential or incidental damages.

"Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by the customer's technical experts.

Samsung products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, for other applications intended to support or sustain life, or for any other application in which the failure of the Samsung product could create a situation where personal injury or death may occur.

Should the Buyer purchase or use a Samsung product for any such unintended or unauthorized application, the Buyer shall indemnify and hold Samsung and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expenses, and reasonable attorney fees arising out of, either directly or indirectly, any claim of personal injury or death that may be associated with such unintended or unauthorized use, even if such claim alleges that Samsung was negligent regarding the design or manufacture of said product.

S3C2800 32-Bit RISC Microprocessor
User's Manual, Revision 1.1
Publication Number: 21.1-S3-C2800-092002

Samsung Electronics' microcontroller business has been awarded full ISO-9001 certification (BSI Certificate No. FM24653). All semiconductor products are designed and manufactured in accordance with the highest quality standards and objectives.

Samsung Electronics Co., Ltd.
San #24 Nongseo-Ri, Giheung- Eup
Yongin-City, Gyeonggi-Do, Korea
C.P.O. Box #37, Suwon 449-900

TEL: (82)-(031)-209-1934
FAX: (82)-(031)-209-1899

Home Page: http://www.samsungsemi.com

Printed in the Republic of Korea

Table of Contents

Chapter 1 Product Overview

Chapter 2 Programmer's Model

Table of Contents (Continued)

Chapter 3 Instruction Set

Table of Contents (Continued)

Chapter 3 Instruction Set (Continued)

Table of Contents (Continued)

Chapter 3 Instruction Set (Continued)

Table of Contents (Continued)

Chapter 3 Instruction Set (Continued)

Table of Contents (Continued)

Chapter 4 Caches, Write Buffer, and Physical Address Tag(PA TAG) RAM

Chapter 5 Memory Management Unit

Table of Contents (Continued)

Chapter 6 Clock & Power Management

Chapter 7 Memory Controller

Table of Contents (Continued)

Chapter 8 DMA

Chapter 9 GPIO ports

Chapter 10 Timer

Table of Contents (Continued)

Chapter 11 UART

Chapter 12 Interrupt Controller

Table of Contents (Continued)

Chapter 13 Remote Control Signal Receive

Chapter 14 RTC (Real Time Clock)

Table of Contents (Continued)

Chapter 15 Watchdog Timer

Chapter 16 IIC-Bus Interface

Table of Contents (Continued)

Chapter 17 PCI-Bus Interface

Table of Contents (Continued)

Chapter 18 Electrical Data

Chapter 19 Mechanical Data

List of Figures

List of Figures (Continued)

List of Figures (Continued)

List of Figures (Continued)

List of Figures (Continued)

List of Tables

List of Tables (Continued)

List of Tables (Continued)

1 PRODUCT OVERVIEW

INTRODUCTION

SAMSUNG's S3C2800 32-bit RISC microprocessor is designed to provide a cost-effective and high-performance micro-controller solution for general applications. The S3C2800 features the following integrated on-chip support to help design a system a low cost: 16KB I/D caches, 2-ch UART with handshake, 4-ch DMA, memory controller, 3-ch timer, GPIO (General-Purpose Input/Output) ports, RTC (Real Time Clock), 2-ch IIC-BUS interface, and a built-in PLL for system clock.

Based on ARM920T core, the S3C2800 is developed using 0.18 um CMOS standard cells and a memory compiler. Its simple, elegant, and fully static low-power design is particularly suitable for both cost-sensitive and power-sensitive applications. The 32-bit ARM920T RISC processor core (220Mips @200MHz), designed by Advanced RISC Machines, Ltd., provides architectural enhancements such as the Thumb de-compressor, a 32-bit hardware multiplier, and an on-chip ICE debug support. Also, the S3C2800 features the Harvard BUS architecture for efficient data/instruction transfers.

By integrating various common system peripherals, the S3C2800 minimizes the overall system cost and eliminates the need to configure additional components. The integrated on-chip functions are summarized as follows :

- PCI BUS interface (32-bit, up to 66MHz).
- 1.8V static ARM920T CPU core with 16KB I/D (Instruction/Data) cache. (Harvard bus architecture up to 200MHz).
- External memory controller. (FP/EDO/SDRAM control, Chip select logic).
- 4-ch general DMAs with external request pins.
- 2-ch UART with handshake (IRDA1.0, 16-byte FIFO), Modem Interface.
- 2-ch multi-master IIC-BUS controller.
- 3-ch 16-bit timer.
- 16-bit Watchdog timer.
- 44 general-purpose GPIO ports including 8 external interrupt source.
- Power management: Normal, Slow, and Idle modes.
- RTC with calendar function.
- On-chip PLL clock generator.

FEATURES

Architecture

- ARM920T CPU core supports the Thumb instruction, ARM instruction, and core debug
- Enhanced multiplier, JTAG, and the embedded ICE
- Support boundary scans
- Memory Management Unit (support virtual memory)
- Internal AMBA bus architecture (AMBA 2.0, AHB/APB)
- Maximum CPU clock frequency of 200MHz@1.8V

Memory Controller

- Little-/Big-endian support for external memory.
- Address space: 32Mbytes per each bank (Total 256Mbyte)
- Supports programmable 8/16/32-bit data bus width for each memory bank
- Fixed bank start address for all (static memory and dynamic memory banks)
- 8 memory banks
  – 4 memory banks for static memory (ROM, SRAM, FLASH etc)
  – 4 memory banks for dynamic memory (Fast Page, EDO, and Synchronous DRAM)
- Fully programmable access cycles for all static memory banks
- Supports external wait signal to extend the bus cycle
- Supports self-refresh mode in DRAM/SDRAM.
- Supports asymmetric/symmetric address of DRAM

I/D (Instruction/Data) Cache Memory

- 64-way set-associative ICache (16KB) and DCache (16KB)
- 8 words per line with one valid bit and 2 dirty bits per line
- Pseudo-random or round-robin replacement algorithm
- Write-through and Write-back cache operation.
- The write buffer can hold 16 words of data and 4 addresses
- Low voltage cache for reduced power consumption

Clock & Power Manager

- The on-chip PLL generates the necessary clock for the operation of MCU at maximum of 200MHz@1.8V
- Input frequency range: (Fin) = 6MHz – 10MHz.
- Output frequency range: (FCLK) = 20MHz – 200MHz
- Clock can be selectively provided to each function block by software
- Power Down Mode: NORMAL, SLOW, and IDLE mode

  – NORMAL mode: Normal operating mode
  – SLOW mode: Low frequency clock without PLL
  – IDLE mode: Clock to CPU is disabled

PCI Bus Interface

- Embedded PCI Host Bridge
- 32-bit data bus at 66MHz

FEATURES (Continued)

Interrupt Controller

- 34 Interrupt sources.
  (3 for Timers, 6 for UART, 8 for External interrupts, 4 for DMA, 2 for RTC, 2 for IIC, 2 for RCSR (Remote Control Signal Receiver), and 7 for PCI))
- Software polling Interrupt mode
- Selectable level- or edge-triggered external interrupts source
- Programmable IRQ/FIQ for each interrupt request
- Supports FIQ (Fast Interrupt Request) for very urgent interrupt request

Timer

- 3-ch 16-bit Timer with DMA-based or interrupt-based operation

Watchdog Timer

- 16-bit Watchdog Timer

RCSR (Remote Control Signal Receiver)

- 8-step FIFO
- FIFO interrupt is generated on full (8) step overflow

RTC (Real Time Clock)

- Full clock feature: sec, min, hour, date, day, week, month, and year
- 32.768 kHz input clock
- Alarm interrupt
- Time tick interrupt

GPIO (General-Purpose Input/Output) Ports

- 8 external interrupt ports
- 44 multiplexed input/output ports.

UART

- 2-channel UART with DMA-based or interrupt based operation
- Supports 5-bit, 6-bit, 7-bit, or 8-bit serial data transmit/receive
- Supports hardware handshaking during transmit/receive operation
- Programmable baud rates (up to 230.4Kbps).
- Supports IrDA 1.0 (up to 115.2Kbps)
- Loop back mode for testing
- Program accessible 16-byte FIFO (2x16 byte FIFO for transmit/receive data)

DMA Controller

- 4-channel general-purpose Direct Memory Access controller without CPU intervention.
- Support memory to memory, memory to I/O and I/O to I/O DMA operations of the following 6 types:

  Software, 3 internal function blocks (UART0, UART1, Timer), and 2 External requests
- Burst transfer mode to enhance the transfer rate on the FPDRAM, EDODRAM and SDRAM

IIC-BUS Interface

- 2-ch Multi-Master IIC-Bus with interrupt-based operation
- Serial, 8-bit oriented, bi-directional data transfers at up to 100 Kbit/s in the standard mode or up to 400 Kbit/s in the fast mode

Operating Voltage Range

- Core: 1.8 V 0.1 V/+0.15 V
- I/O: 3.3 V ± 0.3 V

Operating Frequency

- Up to 200 MHz.

Package

- 208-pin LQFP

BLOCK DIAGRAM

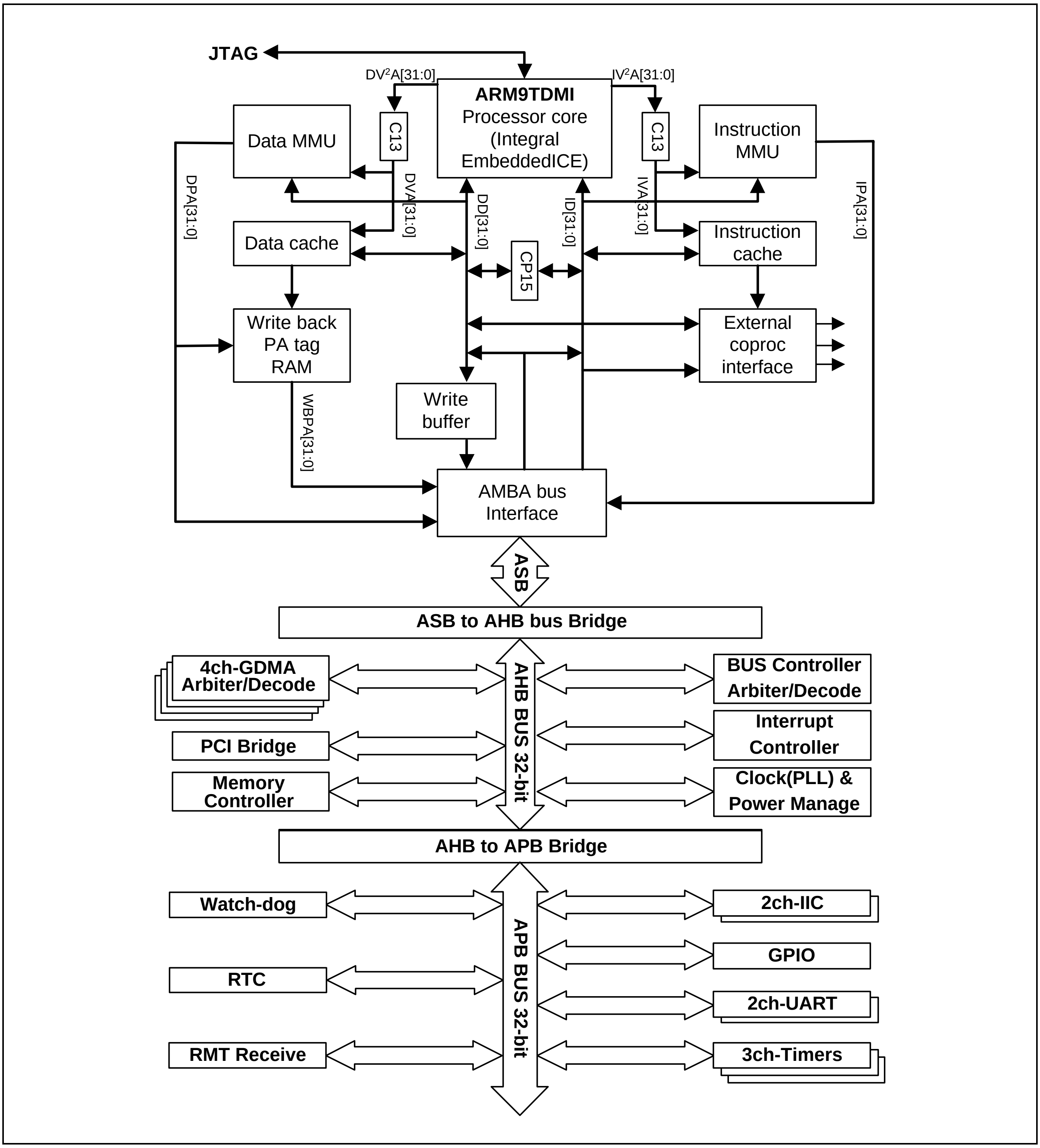


Figure 1-1. S3C2800 Block Diagram

OVERVIEW OF THE ARM920T

The ARM920T is a member of the ARM9 Thumb family of general-purpose microprocessors. The ARM920T is targeted for embedded control applications where high-performance, small die size, and low-power are all important. The ARM920T supports both the 32-bit ARM and 16-bit Thumb instruction sets, allowing the user to trade off between high-performance and high code density. The ARM920T supports the ARM debug architecture and includes logic to assist in both hardware and software. The ARM920T also includes support for coprocessors.

The ARM920T is a Harvard cache architecture processor. The instruction and data caches are of 16KB size with an 8-word line length. The ARM920T implements an enhanced ARM Architecture V4 MMU to provide translation and permission checks for instruction and data accesses.

The processor core within ARM920T is an ARM9TDMI, implemented using a five-stage pipeline consisting of fetch, decode, execute, memory and write stages, and can be provided as a stand-alone core which can be embedded into more complex devices.

The ARM920T interface to the rest of the system is via unified address and data buses. This interface is compatible with the *Advanced Microcontroller Bus Architecture* (AMBA) bus scheme. For coprocessor support, the instruction and data buses are exported along with simple handshaking signals. The ARM920T also has a *TrackingICE* mode, which allows an approach similar to a conventional ICE mode of operation.

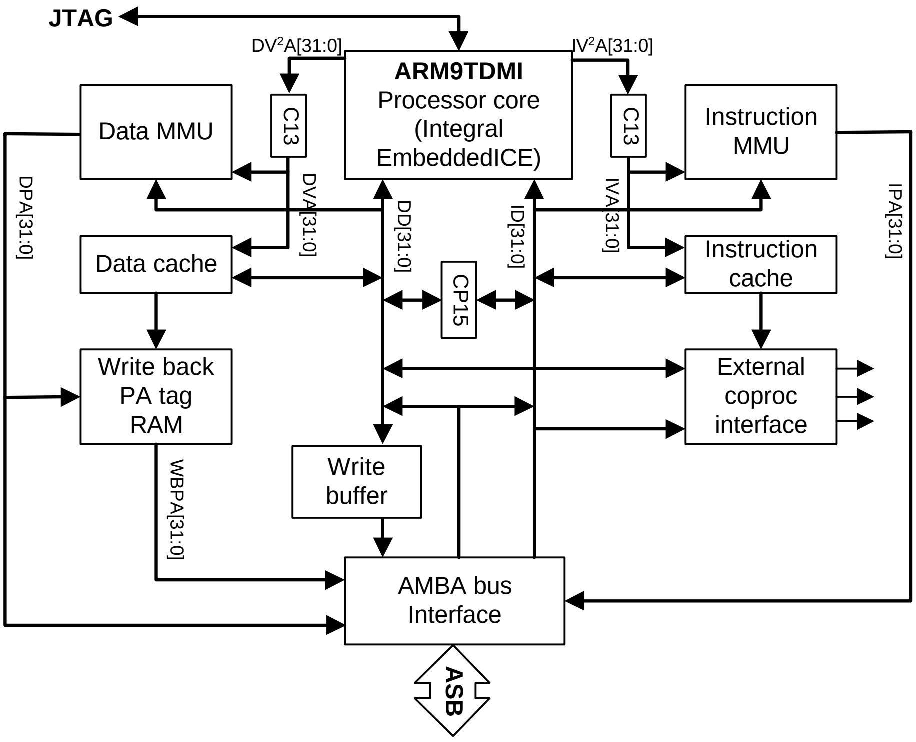


Figure 1-2. ARM920T Functional Block Diagram

PIN DIAGRAM (208-LQFP)

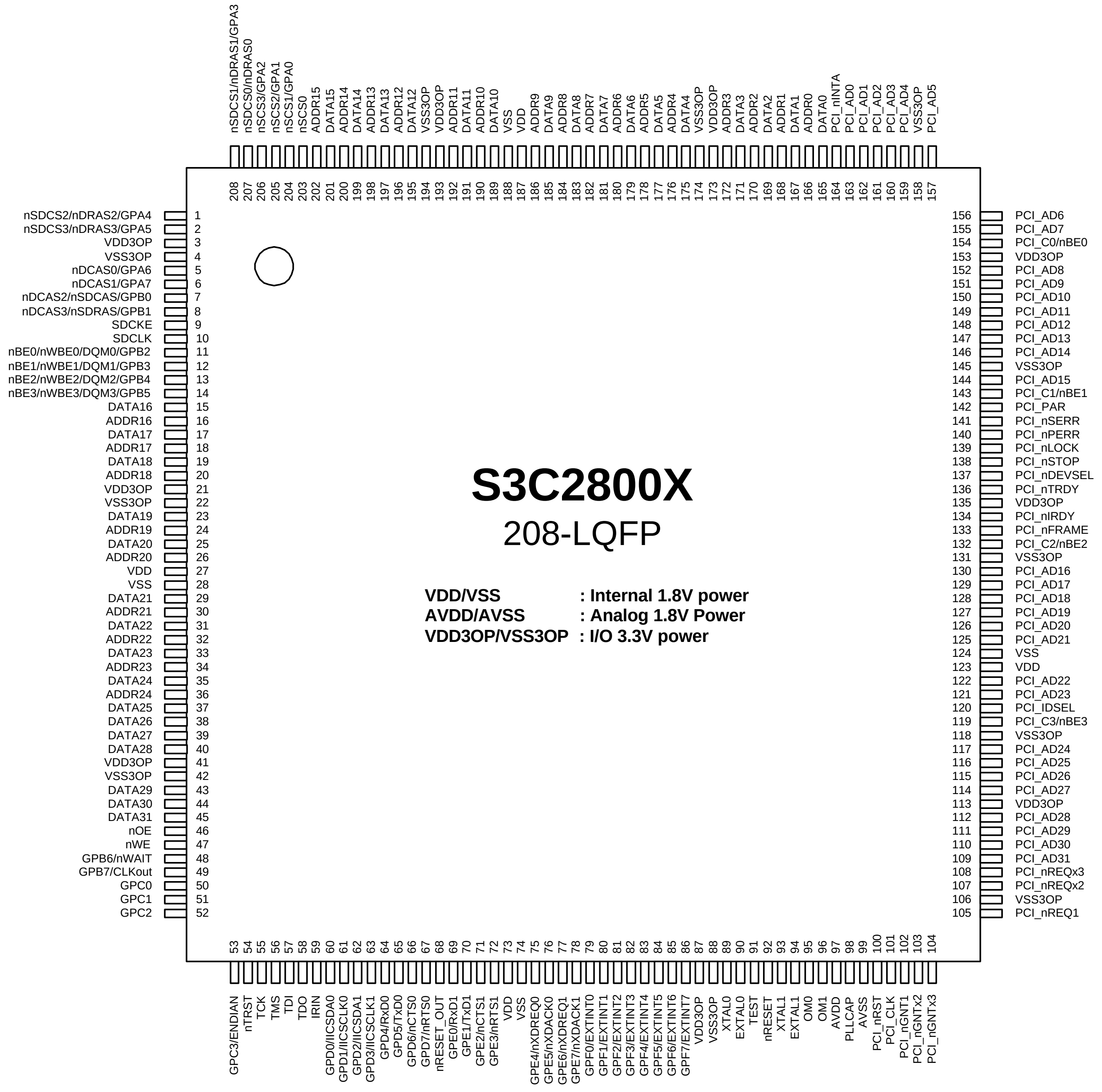


Figure 1-3. S3C2800 Pin Assignment (208-LQFP)

PIN ASSIGNMENTS

Table 1-1. Pin Assignment Description

I/O Type	Descriptions
vdd1ih, vss3I vdd1ih_pci, vss3I_pci	1.8V power/ground for internal logic
vdd1t_abb, vss1t_abb	1.8V power/ground for analog circuitry
vdd3op, vss3op vdd3op_pci, vss3op_pci	3.3V power/ground for external interface logic
poar50_abb	1.8V analog output (A capacitor is connected between the pin and analog ground)
phsoscm16	Oscillator cell width enable and feedback resistor (6 M – 40 MHz)
phsosck17	Oscillator cell width enable and feedback resistor (– 100 kHz)
Phis	3.3V interface LVCMOS schmitt trigger level input buffers
Phisu	3.3V interface LVCMOS schmitt trigger level input buffers with 100 KΩ pull-up resistor.
phob8	3.3V LVCMOS normal output buffers, Io = 8 mA
phob8sm	3.3V LVCMOS normal output buffers with medium slew-rate, Io = 8 mA
phot8	3.3V LVCMOS tri-state output buffers, Io = 8 mA
phob12	3.3V LVCMOS normal output buffers, Io = 12 mA
phbsud4	3.3V open-drain bi-directional buffers with 100 KΩ pull-up resistor. Io=4mA
phbsu50cd4sm	3.3V bi-directional pad, LVCMOS schmitt trigger, open-drain, 50 KΩ pull-up resistor with control, tri-state, Io = 4 mA
phbsu50ct8sm	3.3V bi-directional pad, LVCMOS schmitt trigger, 50 KΩ pull-up resistor with control, tri-state, Io = 8 mA
phbsu50ct12sm	3.3V bi-directional pad, LVCMOS schmitt trigger, 50 KΩ pull-up resistor with control, tri-state, Io = 12 mA
ptipci	3.3V input buffer
ptopci	3.3V output buffer with tri-state
ptbpci	3.3V bi-directional buffer with input and tri-state output
ptbdpci	3.3V bi-directional buffer with input and open-drain output, tri-state

NOTES:

1. ENDIAN value is latched only at the rising edge of nRESET: when nRESET is Low, the ENDIAN (GPC3) pin operates in input mode; nRESET becomes High, the ENDIAN pin will automatically switch to output mode.
2. IICSDA, IICSCLK, PCI_nSERR, and PCI_nINTA pins are of open-drain type.
3. AI/AO means analog input/output.

Table 1-2. 208-Pin LQFP Pin Assignment

Pin #	Pin Name	Default Function	I/O State @Initial	I/O TYPE
1	nSDCS2/nDRAS2/GPA4	nSDCS2	O/IO	phbsu50ct8sm
2	nSDCS3/nDRAS3/GPA5	nSDCS3	O/IO	
3	VDD3OP	VDD3OP	P	vdd3op
4	VSS3OP	VSS3OP	P	vss3op
5	nDCAS0/GPA6	nDCAS0	O/IO	phbsu50ct8sm
6	nDCAS1/GPA7	nDCAS1	O/IO	
7	nDCAS2/nSDCAS/GPB0	nSDCAS	O/IO	
8	nDCAS3/nSDRAS/GPB1	nSDRAS	O/IO	
9	SDCKE	SDCKE	O	phob8
10	SDCLK	SDCLK	O	phob12
11	nBE0/nWBE0/DQM0/GPB2	DQM0	O/IO	phbsu50ct8sm
12	nBE1/nWBE1/DQM1/GPB3	DQM1	O/IO	
13	nBE2/nWBE2/DQM2/GPB4	DQM2	O/IO	
14	nBE3/nWBE3/DQM3/GPB5	DQM3	O/IO	
15	DATA16	DATA16	I/O	phbsu50ct12sm
16	ADDR16	ADDR16	O	phot8
17	DATA17	DATA17	I/O	phbsu50ct12sm
18	ADDR17	ADDR17	O	phot8
19	DATA18	DATA18	I/O	phbsu50ct12sm
20	ADDR18	ADDR18	O	phot8
21	VDD3OP	VDD3OP	P	vdd3op
22	VSS3OP	VSS3OP	P	vss3op
23	DATA19	DATA19	I/O	phbsu50ct12sm
24	ADDR19	ADDR19	O	phot8
25	DATA20	DATA20	I/O	phbsu50ct12sm
26	ADDR20	ADDR20	O	phot8
27	VDD	VDD	P	vdd1ih
28	VSS	VSS	P	vss3i
29	DATA21	DATA21	I/O	phbsu50ct12sm
30	ADDR21	ADDR21	O	phot8
31	DATA22	DATA22	I/O	phbsu50ct12sm
32	ADDR22	ADDR22	O	phot8
33	DATA23	DATA23	I/O	phbsu50ct12sm
34	ADDR23	ADDR23	O	phot8
35	DATA24	DATA24	I/O	phbsu50ct12sm

Table 1-2. 208-Pin LQFP Pin Assignment (Continued)

Pin #	Pin Name	Default Function	I/O State @Initial	I/O TYPE
36	ADDR24	ADDR24	O	phot8
37	DATA25	DATA25	I/O	phbsu50ct12sm
38	DATA26	DATA26	I/O	
39	DATA27	DATA27	I/O	
40	DATA28	DATA28	I/O	
41	VDD3OP	VDD3OP	P	vdd3op
42	VSS3OP	VSS3OP	P	vss3op
43	DATA29	DATA29	I/O	phbsu50ct12sm
44	DATA30	DATA30	I/O	
45	DATA31	DATA31	I/O	
46	nOE	nOE	O	phob8sm
47	nWE	nWE	O	
48	GPB6/nWAIT	GPB6	IO	phbsu50ct8sm
49	GPB7/CLKout	GPB7	IO	
50	GPC0	GPC0	IO	
51	GPC1	GPC1	IO	
52	GPC2	GPC2	IO	
53	GPC3/ENDIAN	ENDIAN	I(1)	
54	nTRST	nTRST	I	phis
55	TCK	TCK	I	phis
56	TMS	TMS	I	phis
57	TDI	TDI	I	phis
58	TDO	TDO	O	phot8
59	IRIN	IRIN	I	phis
60	GPD0/IICSDA0	GPD0	IO(2)	phbsu50cd4sm
61	GPD1/IICSCLK0	GPD1	IO(2)	
62	GPD2/IICSDA1	GPD2	IO(2)	
63	GPD3/IICSCLK1	GPD3	IO(2)	
64	GPD4/RxD0	GPD4	IO	phbsu50ct8sm
65	GPD5/TxD0	GPD5	IO	
66	GPD6/nCTS0	GPD6	IO	
67	GPD7/nRTS0	GPD7	IO	
68	nRESET_OUT	nRESET_OUT	O	phob8
69	GPE0/RxD1	GPE0	IO	phbsu50ct8sm
70	GPE1/TxD1	GPE1	IO	

Table 1-2. 208-Pin LQFP Pin Assignment (Continued)

Pin #	Pin Name	Default Function	I/O State @Initial	I/O TYPE
71	GPE2/nCTS1	GPE2	IO	phbsu50ct8sm
72	GPE3/nRTS1	GPE3	IO	
73	VDD	VDD	P	vdd1ih
74	VSS	VSS	P	vss3i
75	GPE4/nXDREQ0	GPE4	IO	phbsu50ct8sm
76	GPE5/nXDACK0	GPE5	IO	
77	GPE6/nXDREQ1	GPE6	IO	
78	GPE7/nXDACK1	GPE7	IO	
79	GPF0/EXTINT0	GPF0	IO	
80	GPF1/EXTINT1	GPF1	IO	
81	GPF2/EXTINT2	GPF2	IO	
82	GPF3/EXTINT3	GPF3	IO	
83	GPF4/EXTINT4	GPF4	IO	
84	GPF5/EXTINT5	GPF5	IO	
85	GPF6/EXTINT6	GPF6	IO	
86	GPF7/EXTINT7	GPF7	IO	
87	VDD3OP	VDD3OP	P	vdd3op
88	VSS3OP	VSS3OP	P	vss3op
89	XTAL0	XTAL0	AI(3)	phsoscm16
90	EXTAL0	EXTAL0	AO(3)	
91	TEST	TEST	I	phis
92	nRESET	nRESET	I	phisu
93	XTAL1	XTAL1	I	phsosck17
94	EXTAL1	EXTAL1	O	
95	OM0	OM0	I(1)	phis
96	OM1	OM1	I(1)	
97	AVDD	AVDD	P	vdd1t_abb
98	PLLCAP	PLLCAP	AO(3)	poar50_abb
99	AVSS	AVSS	P	vss1t_abb/vbb1_abb
100	PCI_nRST	PCI_nRST	I	ptipci
101	PCI_CLK	PCI_CLK	I	
102	PCI_nGNT1	PCI_nGNT1	IO	ptbpci
103	PCI_nGNTx2	PCI_nGNTx2	O	ptopci
104	PCI_nGNTx3	PCI_nGNTx3	O	
105	PCI_nREQ1	PCI_nREQ1	IO	ptbpci

Table 1-2. 208-Pin LQFP Pin Assignment (Continued)

Pin #	Pin Name	Default Function	I/O State @Initial	I/O TYPE
106	VSS3OP	VSS3OP	P	vss3op_pci
107	PCI_nREQx2	PCI_nREQx2	I	ptipci
108	PCI_nREQx3	PCI_nREQx3	I	
109	PCI_AD31	PCI_AD31	I/O	ptbpci
110	PCI_AD30	PCI_AD30	I/O	
111	PCI_AD29	PCI_AD29	I/O	
112	PCI_AD28	PCI_AD28	I/O	
113	VDD3OP	VDD3OP	P	vdd3op_pci
114	PCI_AD27	PCI_AD27	I/O	ptbpci
115	PCI_AD26	PCI_AD26	I/O	
116	PCI_AD25	PCI_AD25	I/O	
117	PCI_AD24	PCI_AD24	I/O	
118	VSS3OP	VSS3OP	P	vss3op_pci
119	PCI_C3/nBE3	PCI_C3/nBE3	I/O	ptbpci
120	PCI_IDSEL	PCI_IDSEL	I	ptipci
121	PCI_AD23	PCI_AD23	I/O	ptbpci
122	PCI_AD22	PCI_AD22	I/O	
123	VDD	VDD	P	vdd1ih_pci
124	VSS	VSS	P	vss3i_pci
125	PCI_AD21	PCI_AD21	I/O	ptb_pci
126	PCI_AD20	PCI_AD20	I/O	
127	PCI_AD19	PCI_AD19	I/O	
128	PCI_AD18	PCI_AD18	I/O	
129	PCI_AD17	PCI_AD17	I/O	
130	PCI_AD16	PCI_AD16	I/O	
131	VSS3OP	VSS3OP	P	vss3op_pci
132	PCI_C2/nBE2	PCI_C2/nBE2	I/O	ptbpci
133	PCI_nFRAME	PCI_nFRAME	I/O	
134	PCI_nIRDY	PCI_nIRDY	I/O	
135	VDD3OP	VDD3OP	P	vdd3op_pci
136	PCI_nTRDY	PCI_nTRDY	I/O	ptbpci
137	PCI_nDEVSEL	PCI_nDEVSEL	I/O	
138	PCI_nSTOP	PCI_nSTOP	I/O	
139	PCI_nLOCK	PCI_nLOCK	I	ptipci
140	PCI_nPERR	PCI_nPERR	I/O	ptbpci

Table 1-2. 208-Pin LQFP Pin Assignment (Continued)

Pin #	Pin Name	Default Function	I/O State @Initial	I/O TYPE
141	PCI_nSERR	PCI_nSERR	I/O(2)	ptbdpci
142	PCI_PAR	PCI_PAR	I/O	ptbpci
143	PCI_C1/nBE1	PCI_C1/nBE1	I/O	
144	PCI_AD15	PCI_AD15	I/O	
145	VSS3OP	VSS3OP	P	vss3op_pci
146	PCI_AD14	PCI_AD14	I/O	ptbpci
147	PCI_AD13	PCI_AD13	I/O	
148	PCI_AD12	PCI_AD12	I/O	
149	PCI_AD11	PCI_AD11	I/O	
150	PCI_AD10	PCI_AD10	I/O	
151	PCI_AD9	PCI_AD9	I/O	
152	PCI_AD8	PCI_AD8	I/O	
153	VDD3OP	VDD3OP	P	vdd3op_pci
154	PCI_C0/nBE0	PCI_C0/nBE0	I/O	ptbpci
155	PCI_AD7	PCI_AD7	I/O	
156	PCI_AD6	PCI_AD6	I/O	
157	PCI_AD5	PCI_AD5	I/O	
158	VSS3OP	VSS3OP	P	vss3op_pci
159	PCI_AD4	PCI_AD4	I/O	ptbpci
160	PCI_AD3	PCI_AD3	I/O	
161	PCI_AD2	PCI_AD2	I/O	
162	PCI_AD1	PCI_AD1	I/O	
163	PCI_AD0	PCI_AD0	I/O	
164	PCI_nINTA	PCI_nINTA	I/O(2)	phbsud4
165	DATA0	DATA0	I/O	phbsu50ct12sm
166	ADDR0	ADDR0	O	phot8
167	DATA1	DATA1	I/O	phbsu50ct12sm
168	ADDR1	ADDR1	O	phot8
169	DATA2	DATA2	I/O	phbsu50ct12sm
170	ADDR2	ADDR2	O	phot8
171	DATA3	DATA3	I/O	phbsu50ct12sm
172	ADDR3	ADDR3	O	phot8
173	VDD3OP	VDD3OP	P	vdd3op
174	VSS3OP	VSS3OP	P	vss3op
175	DATA4	DATA4	I/O	phbsu50ct12sm

Table 1-2. 208-Pin LQFP Pin Assignment (Continued)

Pin #	Pin Name	Default Function	I/O State @Initial	I/O TYPE
176	ADDR4	ADDR4	O	phot8
177	DATA5	DATA5	I/O	phbsu50ct12sm
178	ADDR5	ADDR5	O	phot8
179	DATA6	DATA6	I/O	phbsu50ct12sm
180	ADDR6	ADDR6	O	phot8
181	DATA7	DATA7	I/O	phbsu50ct12sm
182	ADDR7	ADDR7	O	phot8
183	DATA8	DATA8	I/O	phbsu50ct12sm
184	ADDR8	ADDR8	O	phot8
185	DATA9	DATA9	I/O	phbsu50ct12sm
186	ADDR9	ADDR9	O	phot8
187	VDD	VDD	P	vdd1ih
188	VSS	VSS	P	vss3i
189	DATA10	DATA10	I/O	phbsu50ct12sm
190	ADDR10	ADDR10	O	phot8
191	DATA11	DATA11	I/O	phbsu50ct12sm
192	ADDR11	ADDR11	O	phot8
193	VDD3OP	VDD3OP	P	vdd3op
194	VSS3OP	VSS3OP	P	vss3op
195	DATA12	DATA12	I/O	phbsu50ct12sm
196	ADDR12	ADDR12	O	phot8
197	DATA13	DATA13	I/O	phbsu50ct12sm
198	ADDR13	ADDR13	O	phot8
199	DATA14	DATA14	I/O	phbsu50ct12sm
200	ADDR14	ADDR14	O	phot8
201	DATA15	DATA15	I/O	phbsu50ct12sm
202	ADDR15	ADDR15	O	phot8
203	nSCS0	nSCS0	O	phob8sm
204	nSCS1/GPA0	nSCS1	O/IO	phbsu50ct8sm
205	nSCS2/GPA1	nSCS2	O/IO	phbsu50ct8sm
206	nSCS3/GPA2	nSCS3	O/IO	phbsu50ct8sm
207	nSDCS0/nDRAS0	nSDCS0	O	phob8sm
208	nSDCS1/nDRAS1/GPA3	nSDCS1	O/IO	phbsu50ct8sm

SIGNAL DESCRIPTIONS

Table 1-3. S3C2800 Signal Descriptions

Signal	I/O	Description
BUS CONTROLLER		
OM[1:0]	I	OM [1:0] is used to determines the bus width of static memory bank0 (boot ROM). The pull-up/down resistor determines the logic level. 00 = 8-bit 01 = 16-bit 10 = 32-bit 11 = Not used
ADDR[24:0]	O	ADDR [24:0] (Address Bus) outputs the memory address of the corresponding bank.
DATA[31:0]	IO	DATA [31:0] (Data Bus) inputs data during memory read and outputs data during memory write. The bus width is programmable among 8/16/32-bit.
nSCS[3:0]	O	nSCS[3:0] (Static memory bank Select) are activated when the address of a static memory is within the address region of each bank. The number of access cycles and the bank size can be programmed.
nWE	O	nWE (Write Enable) indicates that the current bus cycle is a write cycle.
nWBE[3:0]	O	Write Byte Enable.
nBE[3:0]	O	16-bit SRAM Byte Enable.
nWAIT	I	Request to prolong a current bus cycle. As long as nWAIT is Low, the current bus cycle can't be completed.
nOE	O	nOE (Output Enable) indicates that the current bus cycle is a read cycle.
ENDIAN	I	It determines whether or not the data type is Little-endian or Big-endian. The pull-up/down resistor determines the logic level during the reset cycle. ENDIAN value is latched only at the rising edge of nRESET: when nRESET is Low, the ENDIAN (GPC3) pin operates in input mode; nRESET becomes High, the ENDIAN pin will automatically switch to output mode. 0 = Little-endian 1 = Big-endian
DRAM/SDRAM		
nDRAS[3:0]	O	Row Address Strobe.
nDCAS[3:0]	O	Column Address Strobe.
nSDRAS	O	SDRAM Row Address Strobe.
nSDCAS	O	SDRAM Column Address Strobe.
nSDCS[3:0]	O	SDRAM Chip Select.
DQM[3:0]	O	SDRAM Data Mask.
SDCLK	O	SDRAM Clock (SDCLK = HCLK).
SDCKE	O	SDRAM Clock Enable.
INTERRUPT CONTROL UNIT		
EXTINT[7:0]	I	External Interrupt request.
DMA		
nXDREQ[1:0]	I	External DMA request.
nXDACK[1:0]	O	External DMA acknowledge.

Table 1-3. S3C2800 Signal Descriptions (Continued)

Signal	I/O	Description
UART		
RxD[1:0]	I	UART receives data input.
TxD[1:0]	O	UART transmits data output.
nCTS[1:0]	I	UART clear to send input signal.
nRTS[1:0]	O	UART request to send output signal.
IIC-BUS		
IICSDA[1:0]	IO	IIC-bus data.
IICSCL[1:0]	IO	IIC-bus clock.
Remote Control Signal Input Interrupt		
IRIN	I	Remote controller signal receive interrupt
GENERAL-PURPOSE I/O PORTs		
GPx[7:0] x 5 GPC[3:0]	IO	General-purpose input/output ports (GPA[7:0], GPB[7:0], GPC[3:0], GPD[7:0] , GPE[7:0], GPF[7:0])
RESET & CLOCK		
nRESET	ST	nRESET suspends any operation in progress and places S3C2800 into a known reset state. For a reset, nRESET must be held to low level for at least 4 CPUCLK after the processor power is stabilized.
nRESET_OUT	O	The nRESET_OUT pin is asserted during hardware reset(POR,nRESET), software reset and watchdog reset.
XTAL0	AI	Crystal Input for internal OSC circuit for system clock. If it isn't used, XTAL0 has to be high level.
EXTAL0	AO	Crystal output for internal OSC circuit for system clock. It is the inverted output of XTAL0. If it isn't used, it has to be a floating pin.
PLLCAP	AI	Loop filter capacitor for system clocks PLL. (1uF)
XTAL1	AI	32 KHz crystal input for RTC.
EXTAL1	AO	32 KHz crystal output for RTC. It is the inverted output of XTAL1.
JTAG TEST LOGIC		
nTRST	I	nTRST(TAP Controller Reset) resets the TAP controller at start. If debugger is used, A 10K pull-up resistor has to be connected. If debugger(black ICE) isn't used, nTRST pin has to be low level or low active pulse.
TMS	I	TMS (TAP Controller Mode Select) controls the sequence of the TAP controller's states. A 10K pull-up resistor has to be connected to TMS pin.
TCK	I	TCK (TAP Controller Clock) provides the clock input for the JTAG logic. A 10K pull-up resistor has to be connected to TCK pin.
TDI	I	TDI (TAP Controller Data Input) is the serial input for test instructions and data. A 10K pull-up resistor has to be connected to TDI pin.
TDO	O	TDO (TAP Controller Data Output) is the serial output for test instructions and data.

Table 1-3. S3C2800 Signal Descriptions (Continued)

Signal	I/O	Description
POWER		
VDD	P	S3C2800 core logic V_{DD} (1.8 V).
VSS	P	S3C2800 core logic V_{SS}.
AVDD	P	S3C2800 Analog logic (PLL loop filter) V_{DD}(1.8V).
AVSS	P	S3C2800 Analog logic (PLL loop filter) V_{SS}.
VDD3OP	P	S3C2800 GPIO port V_{DD} (3.3 V).
VSS3OP	P	S3C2800 GPIO port V_{SS}.
PCI-BUS		
PCI_AD[31:0]	I/O	PCI Address/Data Bus. Multiplexed address and data bus.
PCI_C[3:0]/ nBE[3:0]	I/O	PCI C (bus command) or Byte enables.
PCI_PAR	I/O	PCI-parity. Parity is even across PCI_AD[31:0] and PCI_C[3:0]/nBE[3:0]. PCI_PAR is valid one cycle after either an address or data phase. The PCI device that drives PCI_AD[31:0] is responsible for driving PCI_PAR on the next PCI bus clock.
PCI_nFRAME	I/O	PCI_nFRAME is driven by the current PCI bus master to indicate beginning and duration of a PCI access.
PCI_nTRDY	I/O	The target of the current PCI transaction drives PCI_nTRDY. Assertion of PCI_nTRDY indicates that the PCI target is ready to transfer data.
PCI_nIRDY	I/O	The current PCI bus master drives PCI_nIRDY. Assertion of PC_nIRDY indicates that the PCI initiator is ready to transfer data.
PCI_nSTOP	I/O	The target of the current PCI transaction may assert PCI_nSTOP to indicate to the requesting PCI master that it wants to end the current transaction.
PCI_nDEVSEL	I/O	The target of the current PCI transaction drives PCI_nDEVSEL. A PCI target asserts PCI_nDEVSEL when it decodes an address and command encoding, and claims the transaction.
PCI_IDSEL	I	PCI_IDSEL is used during configuration cycles to select the PCI slave interface for configuration.
PCI_nPERR	I/O	PCI_nPERR is used for reporting data parity errors on PCI transactions. PCI_nPERR is driven active by the device receiving PCI_AD[31:0], PCI_C[3:0]/nBE[3:0], and PCI_PARITY, two PCI clocks following the data in which bad parity is detected.
PCI_nSERR	I/O	PCI_nSERR is used for reporting address parity errors or catastrophic failures detected by a PCI target.
PCI_nLOCK		PCI_nLOCK indicates an atomic operation to a bridge that may require multiple transactions to complet. When PCI_nLOCK is asserted, non-exclusive transactions may proceed to a bridge that is not currently locked. A grant to start a transaction on PCI does not guarantee a control of PCI_nLOCK. Locked transactions may be initiated only by the host bridges.
PCI_nREQ1	I/O	When internal arbiter is used, PCI_nREQ1 is input mode. Or when external arbiter is used, PCI_nREQ1 is output mode.

Table 1-3. S3C2800 Signal Descriptions (Continued)

Signal	I/O	Description
PCI_nREQx[3:2]	I	PCI_nREQx[3:2] input when internal arbiter is used. Request indicates to the arbiter that this agent desires use of the bus. This is a point-to-point signal. Every master has its own PCI_nREQx, which must be tri-stated, while PCI_nRST is asserted.
PCI_nGNT1	I/O	When internal arbiter is used, PCI_nGNT1 is output mode. Or when external arbiter is used, PCI_nGNT1 is input mode.
PCI_nGNTx[3:2]	O	PCI_nGNTx[3:2] output when internal arbiter is used. Grant indicates to the agent that access to the bus has been granted. This is a point-to-point signal. Every master has its own PCI_nGNTx, which must be ignored while PCI-nRST is asserted.
PCI_CLK	I	PCI_CLK is used as the asynchronous PCI clock.
PCI_nRST	O	PCI specific reset
PCI_nINTA	O	PCI interrupt.

S3C2800 SPECIAL FUNCTION REGISTERS

Table 1-4. S3C2800 Special Function Registers

Register Name	Address	R/W	Description	Reset Value
CLOCK & POWER MANAGEMENT				
PLLCON	0x1000 0000	R/W	PLL configuration register	Undefined
CLKCON	0x1000 0004	R/W	Clock control register	0x0000 17FC
CLKSLOW	0x1000 0008	R/W	Slow clock control register	0x0000 0000
LOCKTIME	0x1000 000C	R/W	PLL lock time count register	0x0000 0FFF
SWRCON	0x1000 0010	W	Software reset control register	0x0000 0000
RSTSR	0x1000 0014	R/W	Reset status register	0x0000 0001
Reserved	0x1000 0018 – 0x1000 FFFF		Reserved	
MEMORY CONTROLLER				
ENDIAN	0x1001 0000	R	Endian status register	Undefined
SMBCON0	0x1001 0004	R/W	Bank 0 control register for static memory	0x0000 00A2
SMBCON1	0x1001 0008	R/W	Bank 1 control register for static memory	0x0000 00A2
SMBCON2	0x1001 000C	R/W	Bank 2 control register for static memory	0x0000 00A2
SMBCON3	0x1001 0010	R/W	Bank 3 control register for static memory	0x0000 00A2
REFRESH	0x1001 0014	R/W	DRAM/SDRAM refresh control register	0x00A4 0000
DMTMCON	0x1001 0018	R/W	Timing control for dynamic memory	0x0002 0D50
MRSR	0x1001 001C	R/W	Mode register set register for SDRAM	0x0000 0030
Reserved	0x1001 0020 – 0x1001 FFFF		Reserved	
INTERRUPT CONTROLLER				
SRCPND	0x1002 0000	R/W	Indicates the interrupt request status.	0x0000 0000
INTMOD	0x1002 0004	R/W	Interrupt mode register	0x0000 0000
INTMSK	0x1002 0008	R/W	Determines which interrupt source is masked	0x0000 0000
IRQPND	0x1002 000C	R	IRQ interrupt service pending register	0x0000 0000
FIQPND	0x1002 0010	R	FIQ interrupt service pending register	0x0000 0000
Reserved	0x1002 0014 – 0x1002 FFFF		Reserved	
DMA CONTROLLER				
DISRC0	0x1003 0000	R/W	DMA 0 source initial register	0x0000 0000
DIDST0	0x1003 0004	R/W	DMA 0 destination initial register	0x0000 0000
DCON0	0x1003 0008	R/W	DMA 0 control register	0x0000 0000
DSTAT0	0x1003 000C	R	DMA 0 status register	0x0000 0000
DCSRC0	0x1003 0010	R	DMA 0 current source register	0x0000 0000
DCDST0	0x1003 0014	R	DMA 0 current destination register	0x0000 0000

Table 1-4. S3C2800 Special Function Registers (Continued)

Register Name	Address	R/W	Description	Reset Value
DMA CONTROLLER (Continued)				
DMASKTRIG0	0x1003 0018	R/W	DMA 0 mask trigger register	0x0000 0000
Reserved	0x1003 001C – 0x1003 FFFF		Reserved	
DISRC1	0x1004 0000	R/W	DMA 1 source initial register	0x0000 0000
DIDST1	0x1004 0004	R/W	DMA 1 destination initial register	0x0000 0000
DCON1	0x1004 0008	R/W	DMA 1 control register	0x0100 0000
DSTAT1	0x1004 000C	R	DMA 1 status register	0x0000 0000
DCSRC1	0x1004 0010	R	DMA 1 current source register	0x0000 0000
DCDST1	0x1004 0014	R	DMA 1 current destination register	0x0000 0000
DMASKTRIG1	0x1004 0018	R/W	DMA 0 mask trigger register	0x0000 0000
Reserved	0x1004 001C – 0x1004 FFFF		Reserved	
DISRC2	0x1005 0000	R/W	DMA 2 source initial register	0x0000 0000
DIDST2	0x1005 0004	R/W	DMA 2 destination initial register	0x0000 0000
DCON2	0x1005 0008	R/W	DMA 2 control register	0x0200 0000
DSTAT2	0x1005 000C	R	DMA 2 status register	0x0000 0000
DCSRC2	0x1005 0010	R	DMA 2 current source register	0x0000 0000
DCDST2	0x1005 0014	R	DMA 2 current destination register	0x0000 0000
DMASKTRIG2	0x1005 0018	R/W	DMA 0 mask trigger register	0x0000 0000
Reserved	0x1005 001C – 0x1005 FFFF		Reserved	
DISRC3	0x1006 0000	R/W	DMA 3 source initial register	0x0000 0000
DIDST3	0x1006 0004	R/W	DMA 3 destination initial register	0x0000 0000
DCON3	0x1006 0008	R/W	DMA 3 control register	0x0300 0000
DSTAT3	0x1006 000C	R	DMA 3 status register	0x0000 0000
DCSRC3	0x1006 0010	R	DMA 3 current source register	0x0000 0000
DCDST3	0x1006 0014	R	DMA 3 current destination register	0x0000 0000
DMASKTRIG3	0x1006 0018	R/W	DMA 0 mask trigger register	0x0000 0000
Reserved	0x1006 001C – 0x1006 FFFF		Reserved	
PCI-BUS (Configuration register)				
CIVDIDR	0x1008 0000	R	PCI vendor ID and device ID register	0x2800 144D
PCISCR	0x1008 0004	R/WC	PCI status and command register	0x02B0 0000
PCICRIDR	0x1008 0008	R/W	PCI class code and revision ID register	0x0D80 0001
PCIGCONR	0x1008 000C	R/W	PCI general control register	0x0000 0000

Table 1-4. S3C2800 Special Function Registers (Continued)

Register Name	Address	R/W	Description	Reset Value
PCI-BUS(Configuration register - Continued)				
PCIBAR0	0x1008 0010	R/W	Memory bar 0 size and location (of fast decode)	0x0000 0008
PCIBAR1	0x1008 0014	R/W	Memory bar 1 size and location (of medium decode)	0x0000 0008
PCIBAR2	0x1008 0018	R/W	I/O bar 2 size and location (of medium decode)	0x0000 0001
PCISSVIDR	0x1008 002C	R/W	PCI subsystem and subsystem vendor ID register	0x2800 144D
PCICPR	0x1008 0034	R	PCI capability pointer register	0x0000 00DC
PCIMISCR	0x1008 003C	R/W	PCI miscellaneous register	0x0000 0000
PCITOR	0x1008 0040	R/W	PCI target ready and retry timeout register	0x0000 8080
Reserved	0x1008 00E4 – 0x1008 00FF		Reserved	
PCI-BUS (BIF Special Function Register)				
PCICON	0x1008 0100	R/W	PCI control and status register	0x0000 0010
PCISET	0x1008 0104	R/W	PCI command, read count and DAC address register	0x0000 0000
PCIINTEN	0x1008 0108	R/W	PCI interrupt enable register	0x0000 0000
PCIINTST	0x1008 010C	R/WC	PCI interrupt statue and pending register	0x0000 0000
PCIINTAD	0X1008 0110	R	PCI interrupted address register	0x0000 0000
PCIBATAPM	0x1008 0114	R/W	PCI base address translation register from AHB to PCI of memory cycle.	0x0000 0000
PCIBATAPI	0x1008 0118	R/W	PCI base address translation register from AHB to PCI of I/O cycle.	0x0000 0000
PCIBELAP	0x1008 0128	R/W	PCI INTA# assert control register in adaptor mode	0x0000 0000
PCIBATPA0	0x1008 0140	R/W	PCI base address translation from PCI to AHB of memory address bar 0.	0x0000 0000
PCIBAM0	0x1008 0144	R/W	PCI base address mask register of memory address bar 0.	0xFFFF 0000
PCIBAM1	0x1008 014C	R/W	PCI base address mask register of memory address bar 1.	0xFFFF FE00
PCIBATPA2	0x1008 0150	R/W	PCI base address translation register from PCI to AHB bus of I/O address bar 2.	0x1008 0100
PCIBAM2	0x1008 0154	R	PCI base address mask register of I/O address bar 2.	0xFFFF FF00
Reserved	0x1008 0158 – 0x100F FFFF			

Table 1-4. S3C2800 Special Function Registers (Continued)

Register Name	Address	R/W	Description	Reset Value
GPIO PORT				
PCONA	0x1010 0000	R/W	Configures the pins of port A	0x0000 FFFF
PDATA	0x1010 0004	R/W	The data register for port A	Undefined
PUPA	0x1010 0008	R/W	Pull-up resistor control register for port A	0x0000 0000
PCONB	0x1010 000C	R/W	Configures the pins of port B	0x0000 0FFF
PDATB	0x1010 0010	R/W	The data register for port B	Undefined
Reserved	0x1010 0014			
PCONC	0x1010 0018	R/W	Configures the pins of port C	0x0000 0000
PDATC	0x1010 001C	R/W	The data register for port C	Undefined
PUPC	0x1010 0020	R/W	Pull-up resistor control register for port C	0x0000 0000
PCOND	0x1010 0024	R/W	Configures the pins of port D	0x0000 0000
PDATD	0x1010 0028	R/W	The data register for port D	Undefined
PUPD	0x1010 002C	R/W	Pull-up resistor control register for port D	0x0000 0000
PCONE	0x1010 0030	R/W	Configures the pins of port E	0x0000 0000
PDATE	0x1010 0034	R/W	The data register for port E	Undefined
PUPE	0x1010 0038	R/W	Pull-up resistor control register for port E	0x0000 0000
PCONF	0x1010 003C	R/W	Configures the pins of port F	0x0000 0000
PDATF	0x1010 0040	R/W	The data register for port F	Undefined
PUPF	0x1010 0044	R/W	Pull-up resistor control register for port F	0x0000 0000
EXTINTR	0x1010 0048	R/W	External Interrupt control register	0x0000 0000
SPUCR	0x1010004C	R/W	Special pull-up resistor control register for DATA port	0x0000 0000
Reserved	0x1010 0050 – 0x1010 FFFF		Reserved	
REMOTE CONTROL SIGNAL RECEIVER				
RRCR	0x1011 0000	R/W	Remote control signal receiver control register	0x0000 0010
FIFOD	0x1011 0004	R	FIFO Data register	Undefined
Reserved	0x1011 0008 – 0x1011 FFFF		Reserved	
WATCHDOG TIMER				
WTPSCLR	0x1012 0000	R/W	Watchdog timer prescaler value register	0x0000 0080
WTCON	0x1012 0004	R/W	Watchdog timer control register	0x0000 0000
WTCNT	0x1012 0008	R	Watchdog timer count register	0x0000 0000
Reserved	0x1012 000C – 0x1012 FFFF		Reserved	

Table 1-4. S3C2800 Special Function Registers (Continued)

Register Name	Address	R/W	Description	Reset Value
TIMER				
TMCON0	0x1013 0000	R/W	Timer 0 control register	0x0000 0000
TMDATA0	0x1013 0004	R/W	Timer 0 data register	0x0080 FFFF
TMCNT0	0x1013 0008	R	Timer 0 count register	0x0000 FFFF
TMDMASEL	0x1013 000C	R/W	DMA or Interrupt mode selecton register	0x0000 0000
Reserved	0x1013 0010 – 0x1013 FFFF		Reserved	
TMCON1	0x1014 0000	R/W	Timer 1 control register	0x0000 0000
TMDATA1	0x1014 0004	R/W	Timer 1 data register	0x0080 FFFF
TMCNT1	0x1014 0008	R	Timer 1 count register	0x0000 FFFF
Reserved	0x1014 000C – 0x1014 FFFF		Reserved	
TMCON2	0x1015 0000	R/W	Timer 2 control register	0x0000 0000
TMDATA2	0x1015 0004	R/W	Timer 2 data register	0x0080 FFFF
TMCNT2	0x1015 0008	R	Timer 2 count register	0x0000 FFFF
Reserved	0x1015 000C – 0x1015 FFFF		Reserved	
RTC				
RTCCON	0x1016 0000	R/W	RTC control register	0x0000 0000
RTCALM	0x1016 0004	R/W	RTC alarm control register	0x0000 0000
ALMSEC	0x1016 0008	R/W	Alarm second data register	0x0000 0000
ALMMIN	0x1016 000C	R/W	Alarm minute data register	0x0000 0000
ALMHOUR	0x1016 0010	R/W	Alarm hour data register	0x0000 0000
ALMDAY	0x1016 0014	R/W	Alarm day data register	0x0000 0001
ALMMON	0x1016 0018	R/W	Alarm month data register	0x0000 0001
ALMYEAR	0x1016 001C	R/W	Alarm hour data register	0x0000 0000
BCDSEC	0x1016 0020	R/W	BCD second register	Undefined
BCDMIN	0x1016 0024	R/W	BCD minute register	Undefined
BCDHOUR	0x1016 0028	R/W	BCD hour register	Undefined
BCDDAY	0x1016 002C	R/W	BCD day register	Undefined
BCDDATE	0x1016 0030	R/W	BCD date register	Undefined
BCDMON	0x1016 0034	R/W	BCD month register	Undefined
BCDYEAR	0x1016 0038	R/W	BCD year register	Undefined
Reserved	0x1016 003C		Reserved	

Table 1-4. S3C2800 Special Function Registers(Continued)

Register Name	Address	R/W	Description	Reset Value
RTC (Continued)				
TICNT	0x1016 0040	R/W	Tick time count register	0x0000 0000
RTCRST	0x1016 0044	R/W	RTC round reset register	-0x0000 0000
Reserved	0x1016 0044 – 0x1016 FFFF		Reserved	
UART				
ULCON0	0x1017 0000	R/W	UART 0 line control register	0x0000 0000
UCON0	0x1017 0004	R/W	UART 0 control register	0x0000 0000
UFCON0	0x1017 0008	R/W	UART 0 FIFO control register	0x0000 0000
UMCON0	0x1017 000C	R/W	UART 0 modem control register	0x0000 0000
UTRSTAT0	0x1017 0010	R	UART 0 Tx/Rx status register	0x0000 0006
UERSTAT0	0x1017 0014	R	UART 0 Rx error status register	0x0000 0000
UFSTAT0	0x1017 0018	R	UART 0 FIFO status register	0x0000 0000
UMSTAT0	0x1017 001C	R	UART 0 modem status register	0x0000 0000
UTXH0	0x1017 0020(L) 0x1017 0023(B)	W (by byte)	UART 0 transmit holding register	Undefined
URXH0	0x1017 0024(L) 0x1017 0027(B)	R (by byte)	UART 0 receive buffer register	Undefined
UBRDIV0	0x1017 0028	R/W	UART 0 baud rate divisior register	0x0000 001A
Reserved	0x1017 002C – 0x1017 FFFF		Reserved	
ULCON1	0x1018 0000	R/W	UART 1 line control register	0x0000 0000
UCON1	0x1018 0004	R/W	UART 1 control register	0x0000 0000
UFCON1	0x1018 0008	R/W	UART 1 FIFO control register	0x0000 0000
UMCON1	0x1018 000C	R/W	UART 1 modem control register	0x0000 0000
UTRSTAT1	0x1018 0010	R	UART 1 Tx/Rx status register	0x0000 0006
UERSTAT1	0x1018 0014	R	UART 1 Rx error status register	0x0000 0000
UFSTAT1	0x1018 0018	R	UART 1 FIFO status register	0x0000 0000
UMSTAT1	0x1018 001C	R	UART 1 modem status register	0x0000 0000
UTXH1	0x1018 0020(L) 0x1018 0023(B)	W (by byte)	UART 1 transmit holding register	Undefined
URXH1	0x1018 0024(L) 0x1018 0027(B)	R (by byte)	UART 1 receive buffer register	Undefined
UBRDIV1	0x1018 0028	R/W	UART 1 baud rate divisior register	0x0000 001A
Reserved	0x1018 002C – 0x1018 FFFF		Reserved	

Table 1-4. S3C2800 Special Function Registers(Continued)

Register Name	Address	R/W	Description	Reset Value
IIC-BUS				
IICCON0	0x1019 0000	R/W	IIC-Bus 0 control register	0x0000 0020
IICSTAT0	0x1019 0004	R/W	IIC-Bus 0 control/status register	0x0000 0000
IICADD0	0x1019 0008	R/W	IIC-Bus 0 address register	Undefined
IICDS0	0x1019 000C	R/W	IIC-Bus 0 transmit/receive data shift register	Undefined
Reserved	0x1019 0010 – 0x1019 FFFF		Reserved	
IICCON1	0x101A 0000	R/W	IIC-Bus 1 control register	0x0000 0020
IICSTAT1	0x101A 0004	R/W	IIC-Bus 1 control/status register	0x0000 0000
IICADD1	0x101A 0008	R/W	IIC-Bus 1 address register	Undefined
IICDS1	0x101A 000C	R/W	IIC-Bus 1 transmit/receive data shift register	Undefined
Reserved	0x101A 0010 – 0x101F FFFF		Reserved	
Special Register : 0x1000 0000 ~ 0x101F FFFF (Total 2MByte)				

IMPORTANT NOTES ABOUT S3C2800 SPECIAL REGISTERS

1. (L) indicates that Little-endian address has to be used: (B) indicates that Big-endian address has to be used.
2. All registers must be accessed in word unit for both Little-/Big-endian modes. That is, the special function registers have to be accessed using either LDR/STR or int pointer (int *) type: LDRB/STRB (byte access), char pointer (char *), LDRH/STRH (half-word access) or short int pointer (short int *) type must not be used for the special function register access except for UART registers (UTXHn/URXHn).
3. It's very important that the UART registers(UTXHn/URXHn) are read/written by the specified access unit and the address. Also, the endian mode used must be considered carefully for the register access.

2 PROGRAMMER'S MODEL

ABOUT THE PROGRAMMER'S MODEL

ARM920T incorporates the ARM9TDMI Integer Core, which implements the ARMv4T Architecture. It executes the ARM and Thumb instruction sets, and includes EmbeddedICE JTAG software debug features.

The programmer’s model of ARM920T is the programmer’s model of ARM9TDMI extended in the following ways:

- The system control coprocessor (CP15), which is integrated within ARM920T, provides additional registers that are used to configure and control the caches, MMU, protection system, and clocking mode of ARM920T.
- The MMU page tables which reside in main memory describe the virtual to physical address mapping, access permissions, and cache and write buffer configuration. These are created by the operating system software and accessed automatically by the ARM920T MMU hardware whenever an access causes a TLB miss.

THE ARM9TDMI PROGRAMMERS MODEL

The ARM9TDMI processor core implements ARM Architecture v4T, and so executes the ARM 32-bit instruction set and the compressed Thumb 16-bit instruction set.

The ARM v4T architecture specifies a small number of implementation options. The options selected in the ARM9TDMI implementation are listed in the table below. For comparison, the options selected for the ARM7TDMI implementation are also shown.

Table 2-1. ARM9TDMI Implementation option

Processor Core	ARM Architecture	Data Abort Model	Values stored by direct STR, STRT, STM of PC
ARM7TDMI	v4T	Base updated	Address of Inst + 12
ARM9TDMI	v4T	Base restored	Address of Inst + 12

The ARM9TDMI is code compatible with the ARM7TDMI, with two exceptions:

- The ARM9TDMI implements the Base Restored Data Abort model, which significantly simplifies the software data abort handler.
- The ARM9TDMI fully implements the instruction set extension space added to the ARM(32-bit) instruction set in Architecture v4 and v4T.

These differences are explained in more detail below.

Data abort model

The ARM9TDMI implements the Base Restored Data Abort Model, which differs from the Base updated data abort model implemented by ARM7TDMI.

The difference in the Data Abort Model affects only a very small section of operating system code, the data abort handler. It does not affect user code. With the Base Restored Data Abort Model, when a data abort exception occurs during the execution of a memory access instruction, the base register is always restored by the processor hardware to the value the register contained before the instruction was executed. This removes the need for the data abort handler to ‘unwind’ any base register update which may have been specified by the aborted instruction.

The Base Restored Data Abort Model significantly simplifies the software data abort handler.

Instruction set extension spaces

All ARM processors implement the undefined instruction space as one of the entry mechanisms for the Undefined Instruction Exception. That is, ARM instructions with opcode[27:25]=0b011 and opcode[4]=1 are UNDEFINED on all ARM processors including the ARM9TDMI and ARM7TDMI

ARM Architecture v4 and v4T also introduced a number of instruction set extension space to the ARM instruction set. These are:

— arithmetic instruction extension space

— control instruction extension space

— coprocessor instruction extension space

— load/store instruction extension space

Instructions in these space are UNDEFINED (they cause an Undefined Instruction Exception). The ARM9TDMI fully implements all the instruction set extension spaces defined in ARM Architecture v4T as UNDEFINED instructions, allowing emulation of future instruction set additions.

THE ARM920T PROGRAMMERS MODEL

The ARM920T implements uses a five-stage pipeline design. These five stages are:

— instruction fetch (F)

— instruction decode (D)

— execute (E)

— data memory access (M)

— register write (W)

ARM implementations are fully interlocked, so that software will function identically across different implementations without concern for pipeline effects. Interlock do affect instruction times. For example, the following sequence suffers a single cycle penalty due to a load-use interlock on register r0:

```
LDR     R0,[R7]
ADD     R5,R0,R1
```

PROCESSOR OPERATING STATES

From the programmer's point of view, the ARM920T can be in one of two states:

- ARM state which executes 32-bit, word-aligned ARM instructions.
- *THUMB state* which can execute 16-bit, halfword-aligned THUMB instructions. In this state, the PC uses bit 1 to select between alternate halfwords.

NOTE

Transition between these two states does not affect the processor mode or the contents of the registers.

SWITCHING STATE

Entering THUMB State

Entry into THUMB state can be achieved by executing a BX instruction with the state bit (bit 0) set in the operand register. Transition to THUMB state will also occur automatically on return from an exception (IRQ, FIQ, UNDEF, ABORT, SWI etc.), if the exception was entered with the processor in THUMB state.

Entering ARM State

Entry into ARM state happens:

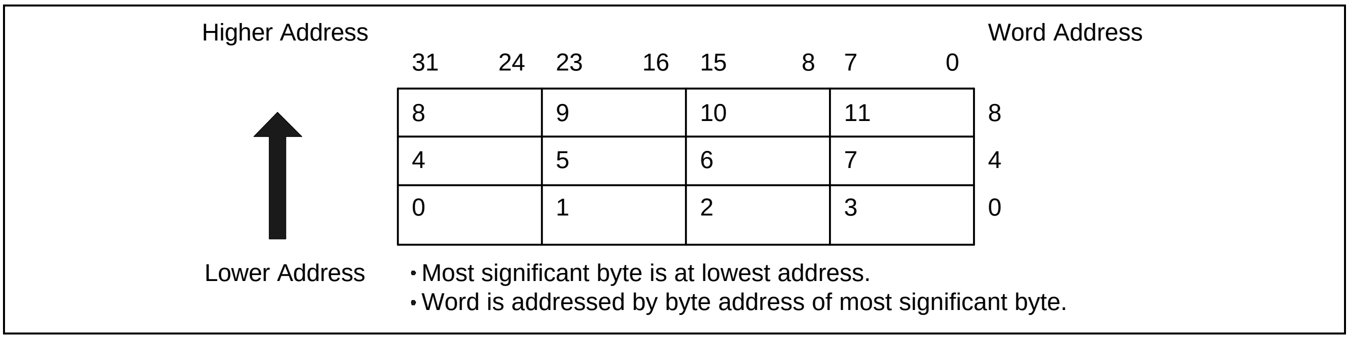
- On execution of the BX instruction with the state bit clear in the operand register.
- On the processor taking an exception (IRQ, FIQ, RESET, UNDEF, ABORT, SWI etc.). In this case, the PC is placed in the exception mode's link register, and execution commences at the exception's vector address.

MEMORY FORMATS

ARM920T views memory as a linear collection of bytes numbered upwards from zero. Bytes 0 to 3 hold the first stored word, bytes 4 to 7 the second and so on. ARM920T can treat words in memory as being stored either in Big-Endian or Little-Endian format.

Big-Endian Format

In Big-Endian format, the most significant byte of a word is stored at the lowest numbered byte and the least significant byte at the highest numbered byte. Byte 0 of the memory system is therefore connected to data lines 31 through 24.



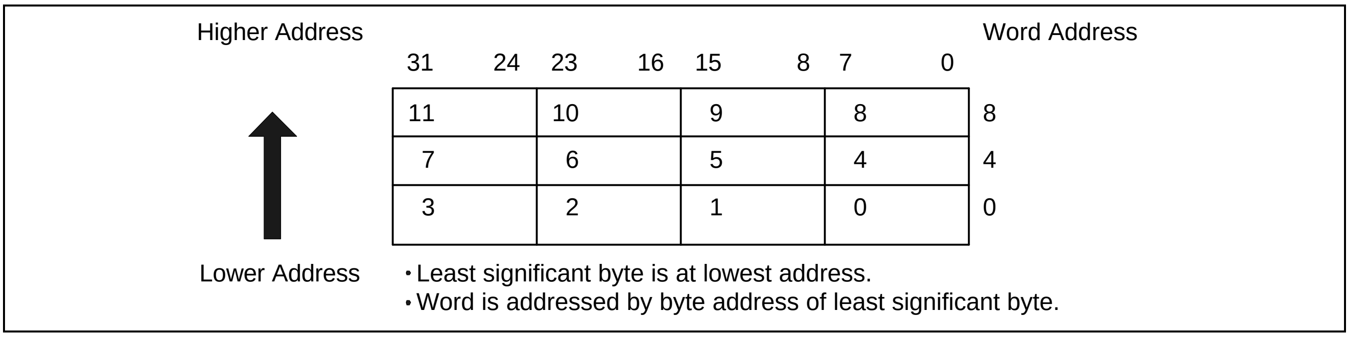
31 24	23 16	15 8	7 0	Word Address
8	9	10	11	8
4	5	6	7	4
0	1	2	3	0

- Most significant byte is at lowest address.
- Word is addressed by byte address of most significant byte.

Figure 2-1. Big-Endian Addresses of Bytes within Words

Little-Endian Format

In Little-Endian format, the lowest numbered byte in a word is considered the word's least significant byte, and the highest numbered byte the most significant. Byte 0 of the memory system is therefore connected to data lines 7 through 0.



31 24	23 16	15 8	7 0	Word Address
11	10	9	8	8
7	6	5	4	4
3	2	1	0	0

- Least significant byte is at lowest address.
- Word is addressed by byte address of least significant byte.

Figure 2-2. Little-Endian Addresses of Bytes whthin Words

INSTRUCTION LENGTH

Instructions are either 32 bits long (in ARM state) or 16 bits long (in THUMB state).

DATA TYPES

ARM920T supports byte (8-bit), halfword (16-bit) and word (32-bit) data types. Words must be aligned to four-byte boundaries and half words to two-byte boundaries.

OPERATING MODES

ARM920T supports seven modes of operation:

- User (usr): The normal ARM program execution state
- FIQ (fiq): Designed to support a data transfer or channel process
- IRQ (irq): Used for general-purpose interrupt handling
- Supervisor (svc): Protected mode for the operating system
- Abort mode (abt): Entered after a data or instruction prefetch abort
- System (sys): A privileged user mode for the operating system
- Undefined (und): Entered when an undefined instruction is executed

Mode changes may be made under software control, or may be brought about by external interrupts or exception processing. Most application programs will execute in User mode. The non-user modes' known as privileged modes-are entered in order to service interrupts or exceptions, or to access protected resources.

REGISTERS

ARM9TDMI has a total of 37 registers - 31 general-purpose 32-bit registers and six status registers - but these cannot all be seen at once. The processor state and operating mode dictate which registers are available to the programmer.

The ARM State Register Set

In ARM state, 16 general registers and one or two status registers are visible at any one time. In privileged (non-User) modes, mode-specific banked registers are switched in. Figure 2-3 shows which registers are available in each mode: the banked registers are marked with a shaded triangle.

The ARM state register set contains 16 directly accessible registers: R0 to R15. All of these except R15 are general-purpose, and may be used to hold either data or address values. In addition to these, there is a seventeenth register used to store status information.

Register 14 is used as the subroutine link register. This receives a copy of R15 when a Branch and Link (BL) instruction is executed. At all other times it may be treated as a general-purpose register. The corresponding banked registers R14_svc, R14_irq, R14_fiq, R14_abt and R14_und are similarly used to hold the return values of R15 when interrupts and exceptions arise, or when Branch and Link instructions are executed within interrupt or exception routines.

Register 15 holds the Program Counter (PC). In ARM state, bits [1:0] of R15 are zero and bits [31:2] contain the PC. In THUMB state, bit [0] is zero and bits [31:1] contain the PC.

Register 16 is the CPSR (Current Program Status Register). This contains condition code flags and the current mode bits.

FIQ mode has seven banked registers mapped to R8-14 (R8_fiq-R14_fiq). In ARM state, many FIQ handlers do not need to save any registers. User, IRQ, Supervisor, Abort and Undefined each have two banked registers mapped to R13 and R14, allowing each of these modes to have a private stack pointer and link registers

ARM State General Registers and Program Counter

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	◣ R8_fiq	R8	R8	R8	R8
R9	◣ R9_fiq	R9	R9	R9	R9
R10	◣ R10_fiq	R10	R10	R10	R10
R11	◣ R11_fiq	R11	R11	R11	R11
R12	◣ R12_fiq	R12	R12	R12	R12
R13	◣ R13_fiq	◣ R13_svc	◣ R13_abt	◣ R13_irq	◣ R13_und
R14	◣ R14_fiq	◣ R14_svc	◣ R14_abt	◣ R14_irq	◣ R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

ARM State Program Status Registers

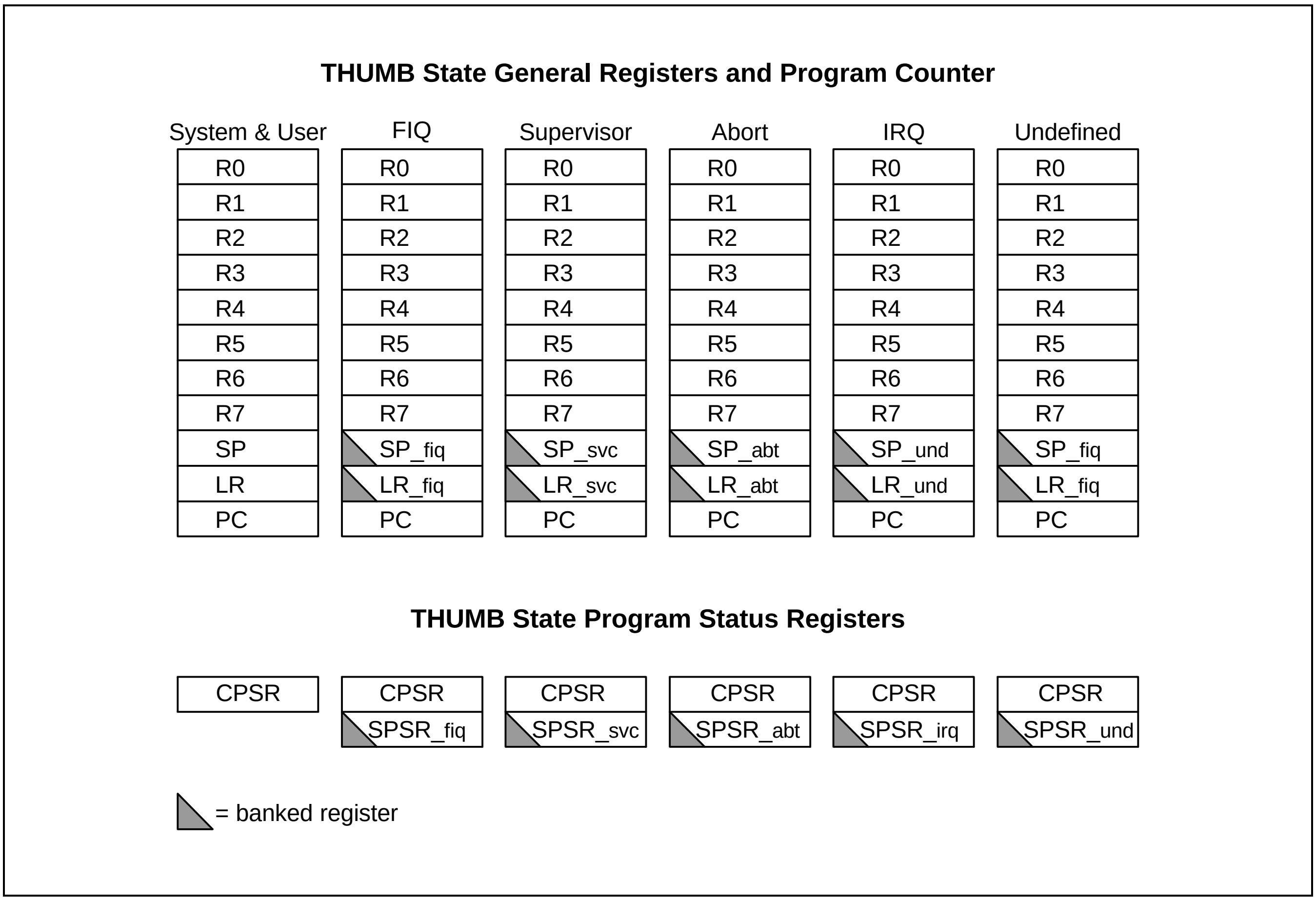
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	◣ SPSR_fiq	◣ SPSR_svc	◣ SPSR_abt	◣ SPSR_irq	◣ SPSR_und

◣ = banked register

Figure 2-3. Register Organization in ARM State

The THUMB State Register Set

The THUMB state register set is a subset of the ARM state set. The programmer has direct access to eight general registers, R0-R7, as well as the Program Counter (PC), a stack pointer register (SP), a link register (LR), and the CPSR. There are banked Stack Pointers, Link Registers and Saved Process Status Registers (SPSRs) for each privileged mode. This is shown in Figure 2-4.



THUMB State General Registers and Program Counter

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
SP	◣ SP_fiq	◣ SP_svc	◣ SP_abt	◣ SP_und	◣ SP_fiq
LR	◣ LR_fiq	◣ LR_svc	◣ LR_abt	◣ LR_und	◣ LR_fiq
PC	PC	PC	PC	PC	PC

THUMB State Program Status Registers

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	◣ SPSR_fiq	◣ SPSR_svc	◣ SPSR_abt	◣ SPSR_irq	◣ SPSR_und

◣ = banked register

Figure 2-4. Register Organization in THUMB state

The Relationship Between ARM and THUMB State Registers

The THUMB state registers relate to the ARM state registers in the following way:

- THUMB state R0-R7 and ARM state R0-R7 are identical
- THUMB state CPSR and SPSRs and ARM state CPSR and SPSRs are identical
- THUMB state SP maps onto ARM state R13
- THUMB state LR maps onto ARM state R14
- The THUMB state Program Counter maps onto the ARM state Program Counter (R15)

This relationship is shown in Figure 2-5.

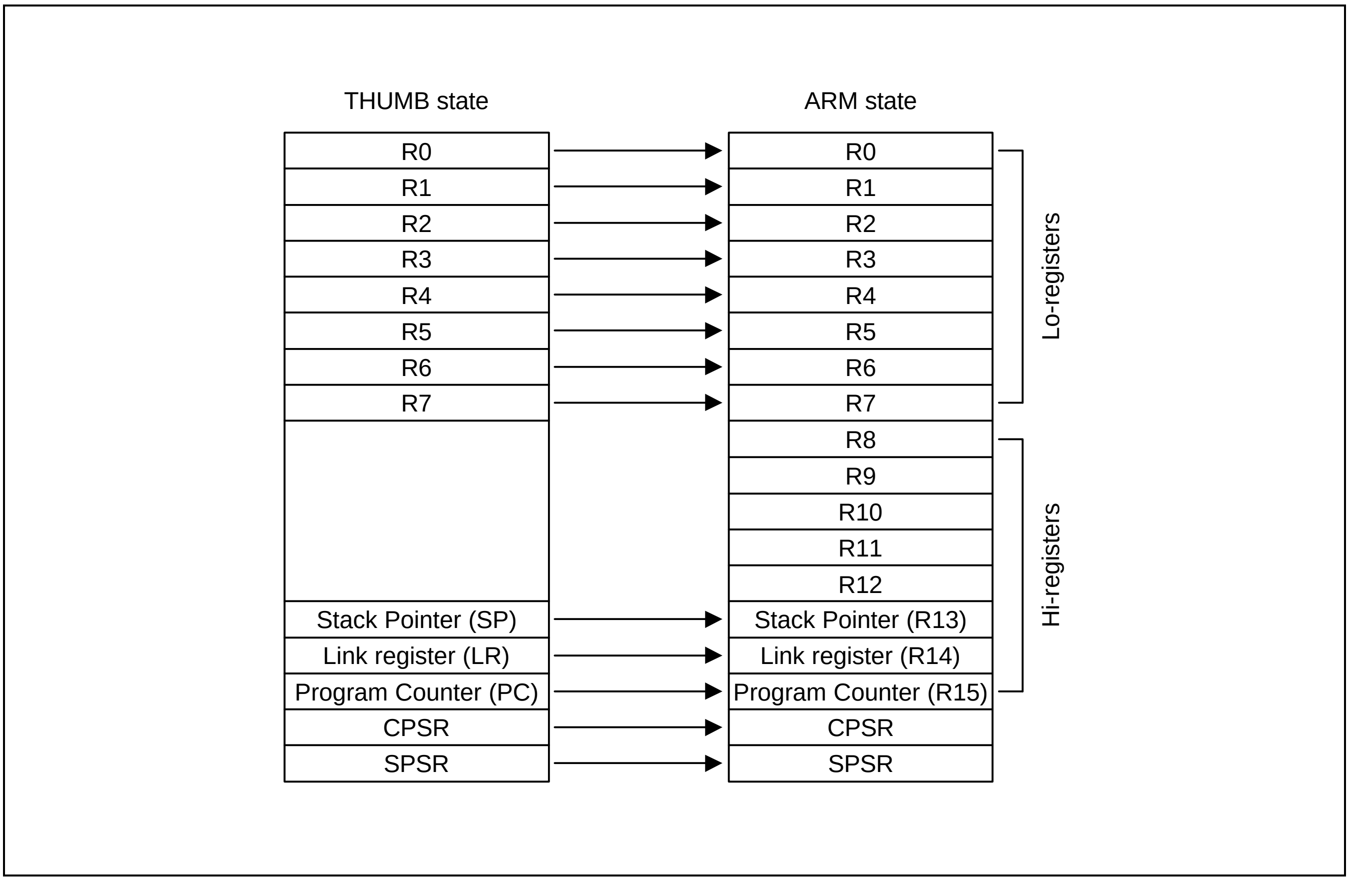


Figure 2-5. Mapping of THUMB State Registers onto ARM State Registers

Accessing Hi-Registers in THUMB State

In THUMB state, registers R8-R15 (the Hi registers) are not part of the standard register set. However, the assembly language programmer has limited access to them, and can use them for fast temporary storage.

A value may be transferred from a register in the range R0-R7 (a Lo register) to a Hi register, and from a Hi register to a Lo register, using special variants of the MOV instruction. Hi register values can also be compared against or added to Lo register values with the CMP and ADD instructions. For more information, refer to Figure 3-38 in Chapter 3 *Instruction set*.

THE PROGRAM STATUS REGISTERS

The ARM920T contains a Current Program Status Register (CPSR), plus five Saved Program Status Registers (SPSRs) for use by exception handlers. These register's functions are:

- Hold information about the most recently performed ALU operation
- Control the enabling and disabling of interrupts
- Set the processor operating mode

The arrangement of bits is shown in Figure 2-6.

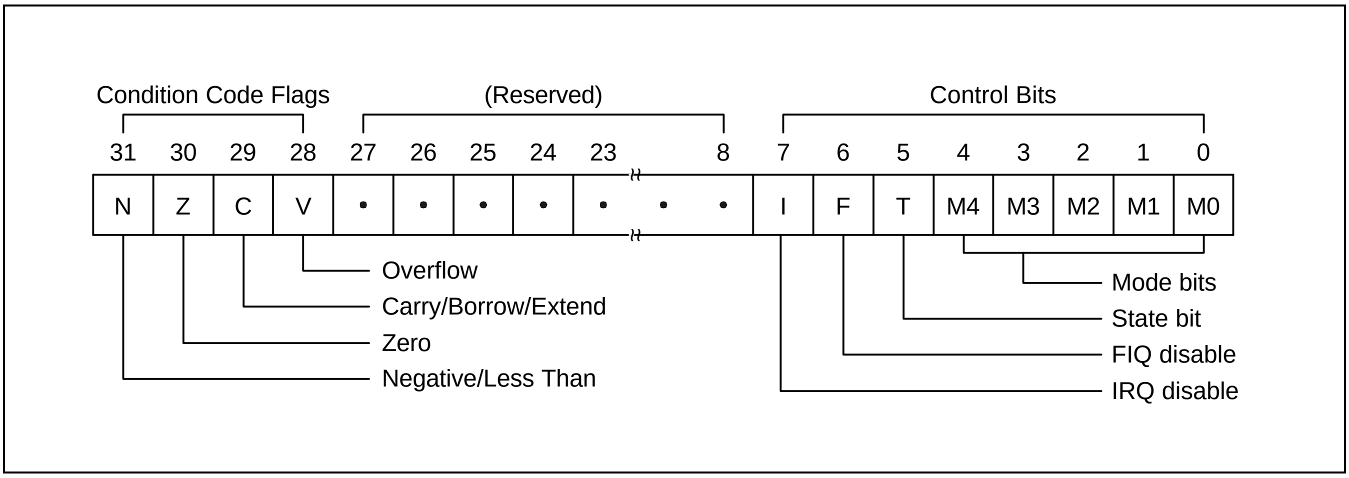


Figure 2-6. Program Status Register Format

The Condition Code Flags

The N, Z, C and V bits are the condition code flags. These may be changed as a result of arithmetic and logical operations, and may be tested to determine whether an instruction should be executed.

In ARM state, all instructions may be executed conditionally: see Table 3-5 in Chapter 3 *Instruction set* for details.

In THUMB state, only the Branch instruction is capable of conditional execution: see Figure 3-50 in Chapter 3 *Instruction set* for details.

The Control Bits

The bottom 8 bits of a PSR (incorporating I, F, T and M[4:0]) are known collectively as the control bits. These will be changed when an exception arises. If the processor is operating in a privileged mode, they can also be manipulated by software.

The T bit This reflects the operating state. When this bit is set, the processor is executing in THUMB state, otherwise it is executing in ARM state. This is reflected on the **TBIT** external signal.

Note that the software must never change the state of the **TBIT** in the CPSR. If this happens, the processor will enter an unpredictable state.

Interrupt disable bits The I and F bits are the interrupt disable bits. When set, these disable the IRQ and FIQ interrupts respectively.

The mode bits The M4, M3, M2, M1 and M0 bits (M[4:0]) are the mode bits. These determine the processor's operating mode, as shown in Table 2-2. Not all combinations of the mode bits define a valid processor mode. Only those explicitly described shall be used. The user should be aware that if any illegal value is programmed into the mode bits, M[4:0], then the processor will enter an unrecoverable state. If this occurs, reset should be applied.

Reserved bits The remaining bits in the PSRs are reserved. When changing a PSR's flag or control bits, you must ensure that these unused bits are not altered. Also, your program should not rely on them containing specific values, since in future processors they may read as one or zero.

Table 2-2. PSR Mode Bit Values

M[4:0]	Mode	Visible THUMB state registers	Visible ARM state registers
10000	User	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR
10001	FIQ	R7..R0, LR_fiq, SP_fiq PC, CPSR, SPSR_fiq	R7..R0, R14_fiq..R8_fiq, PC, CPSR, SPSR_fiq
10010	IRQ	R7..R0, LR_irq, SP_irq PC, CPSR, SPSR_irq	R12..R0, R14_irq, R13_irq, PC, CPSR, SPSR_irq
10011	Supervisor	R7..R0, LR_svc, SP_svc, PC, CPSR, SPSR_svc	R12..R0, R14_svc, R13_svc, PC, CPSR, SPSR_svc
10111	Abort	R7..R0, LR_abt, SP_abt, PC, CPSR, SPSR_abt	R12..R0, R14_abt, R13_abt, PC, CPSR, SPSR_abt
11011	Undefined	R7..R0 LR_und, SP_und, PC, CPSR, SPSR_und	R12..R0, R14_und, R13_und, PC, CPSR
11111	System	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR

Reserved bits The remaining bits in the PSR's are reserved. When changing a PSR's flag or control bits, you must ensure that these unused bits are not altered. Also, your program should not rely on them containing specific values, since in future processors they may read as one or zero.

EXCEPTIONS

Exceptions arise whenever the normal flow of a program has to be halted temporarily, for example to service an interrupt from a peripheral. Before an exception can be handled, the current processor state must be preserved so that the original program can resume when the handler routine has finished.

It is possible for several exceptions to arise at the same time. If this happens, they are dealt with in a fixed order. See *Exception Priorities*.

Action on Entering an Exception

When handling an exception, the ARM920T:

1. Preserves the address of the next instruction in the appropriate Link Register. If the exception has been entered from ARM state, then the address of the next instruction is copied into the Link Register (that is, current PC + 4 or PC + 8 depending on the exception. See Table 2-3 for details). If the exception has been entered from THUMB state, then the value written into the Link Register is the current PC offset by a value such that the program resumes from the correct place on return from the exception. This means that the exception handler need not determine which state the exception was entered from. For example, in the case of SWI, MOVS PC, R14_svc will always return to the next instruction regardless of whether the SWI was executed in ARM or THUMB state.
2. Copies the CPSR into the appropriate SPSR
3. Forces the CPSR mode bits to a value which depends on the exception
4. Forces the PC to fetch the next instruction from the relevant exception vector

It may also set the interrupt disable flags to prevent otherwise unmanageable nestings of exceptions.

If the processor is in THUMB state when an exception occurs, it will automatically switch into ARM state when the PC is loaded with the exception vector address.

Action on Leaving an Exception

On completion, the exception handler:

1. Moves the Link Register, minus an offset where appropriate, to the PC. (The offset will vary depending on the type of exception.)
2. Copies the SPSR back to the CPSR
3. Clears the interrupt disable flags, if they were set on entry

NOTE

An explicit switch back to THUMB state is never needed, since restoring the CPSR from the SPSR automatically sets the T bit to the value it held immediately prior to the exception.

Exception Entry/Exit Summary

Table 2-3 summarizes the PC value preserved in the relevant R14 on exception entry, and the recommended instruction for exiting the exception handler.

Table 2-3. Exception Entry/Exit

	Return Instruction	Previous State		Notes
		ARM R14_x	THUMB R14_x	
BL	MOV PC, R14	PC + 4	PC + 2	1
SWI	MOVS PC, R14_svc	PC + 4	PC + 2	1
UDEF	MOVS PC, R14_und	PC + 4	PC + 2	1
FIQ	SUBS PC, R14_fiq, #4	PC + 4	PC + 4	2
IRQ	SUBS PC, R14_irq, #4	PC + 4	PC + 4	2
PABT	SUBS PC, R14_abt, #4	PC + 4	PC + 4	1
DABT	SUBS PC, R14_abt, #8	PC + 8	PC + 8	3
RESET	NA	-	-	4

NOTES:

1. Where PC is the address of the BL/SWI/Undefined Instruction fetch which had the prefetch abort.
2. Where PC is the address of the instruction which did not get executed since the FIQ or IRQ took priority.
3. Where PC is the address of the Load or Store instruction which generated the data abort.
4. The value saved in R14_svc upon reset is unpredictable.

FIQ

The FIQ (Fast Interrupt Request) exception is designed to support a data transfer or channel process, and in ARM state has sufficient private registers to remove the need for register saving (thus minimising the overhead of context switching).

FIQ is externally generated by taking the **nFIQ** input LOW. This input can except either synchronous or asynchronous transitions, depending on the state of the **ISYNC** input signal. When **ISYNC** is LOW, **nFIQ** and **nIRQ** are considered asynchronous, and a cycle delay for synchronization is incurred before the interrupt can affect the processor flow.

Irrespective of whether the exception was entered from ARM or Thumb state, a FIQ handler should leave the interrupt by executing

```
            SUBS        PC,R14_fiq,#4
```

FIQ may be disabled by setting the CPSR's F flag (but note that this is not possible from User mode). If the F flag is clear, ARM920T checks for a LOW level on the output of the FIQ synchroniser at the end of each instruction.

IRQ

The IRQ (Interrupt Request) exception is a normal interrupt caused by a LOW level on the **nIRQ** input. IRQ has a lower priority than FIQ and is masked out when a FIQ sequence is entered. It may be disabled at any time by setting the I bit in the CPSR, though this can only be done from a privileged (non-User) mode.

Irrespective of whether the exception was entered from ARM or Thumb state, an IRQ handler should return from the interrupt by executing

```
SUBS        PC,R14_irq,#4
```

Abort

An abort indicates that the current memory access cannot be completed. It can be signaled by the external **ABORT** input. ARM920T checks for the abort exception during memory access cycles.

There are two types of abort:

- *Prefetch abort:* occurs during an instruction prefetch.
- *Data abort:* occurs during a data access.

If a prefetch abort occurs, the prefetched instruction is marked as invalid, but the exception will not be taken until the instruction reaches the head of the pipeline. If the instruction is not executed - for example because a branch occurs while it is in the pipeline - the abort does not take place.

If a data abort occurs, the action taken depends on the instruction type:

- Single data transfer instructions (LDR, STR) write back modified base registers: the Abort handler must be aware of this.
- The swap instruction (SWP) is aborted as though it had not been executed.
- Block data transfer instructions (LDM, STM) complete. If write-back is set, the base is updated. If the instruction would have overwritten the base with data (ie it has the base in the transfer list), the overwriting is prevented. All register overwriting is prevented after an abort is indicated, which means in particular that R15 (always the last register to be transferred) is preserved in an aborted LDM instruction.

The abort mechanism allows the implementation of a demand paged virtual memory system. In such a system the processor is allowed to generate arbitrary addresses. When the data at an address is unavailable, the Memory Management Unit (MMU) signals an abort. The abort handler must then work out the cause of the abort, make the requested data available, and retry the aborted instruction. The application program needs no knowledge of the amount of memory available to it, nor is its state in any way affected by the abort.

After fixing the reason for the abort, the handler should execute the following irrespective of the state (ARM or Thumb):

```
SUBS        PC,R14_abt,#4           ;   for a prefetch abort, or
SUBS        PC,R14_abt,#8           ;   for a data abort
```

This restores both the PC and the CPSR, and retries the aborted instruction.

Software Interrupt

The software interrupt instruction (SWI) is used for entering Supervisor mode, usually to request a particular supervisor function. A SWI handler should return by executing the following irrespective of the state (ARM or Thumb):

```
MOV         PC,R14_svc
```

This restores the PC and CPSR, and returns to the instruction following the SWI.

NOTE

nFIQ, nIRQ, ISYNC, LOCK, BIGEND, and ABORT pins exist only in the ARM920T CPU core.

Undefined Instruction

When ARM920T comes across an instruction which it cannot handle, it takes the undefined instruction trap. This mechanism may be used to extend either the THUMB or ARM instruction set by software emulation.

After emulating the failed instruction, the trap handler should execute the following irrespective of the state (ARM or Thumb):

```
MOVS        PC,R14_und
```

This restores the CPSR and returns to the instruction following the undefined instruction.

Exception Vectors

The following table shows the exception vector addresses.

Table 2-4. Exception Vectors

Address	Exception	Mode in Entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software Interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	Reserved	Reserved
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

Exception Priorities

When multiple exceptions arise at the same time, a fixed priority system determines the order in which they are handled:

Highest priority:

1. Reset
2. Data abort
3. FIQ
4. IRQ
5. Prefetch abort

Lowest priority:

6. Undefined Instruction, Software interrupt.

Not All Exceptions Can Occur at Once:

Undefined Instruction and Software Interrupt are mutually exclusive, since they each correspond to particular (non-overlapping) decodings of the current instruction.

If a data abort occurs at the same time as a FIQ, and FIQs are enabled (ie the CPSR's F flag is clear), ARM920T enters the data abort handler and then immediately proceeds to the FIQ vector. A normal return from FIQ will cause the data abort handler to resume execution. Placing data abort at a higher priority than FIQ is necessary to ensure that the transfer error does not escape detection. The time for this exception entry should be added to worst-case FIQ latency calculations.

INTERRUPT LATENCIES

The worst case latency for FIQ, assuming that it is enabled, consists of the longest time the request can take to pass through the synchroniser (*Tsyncmax* if asynchronous), plus the time for the longest instruction to complete (*Tldm*, the longest instruction is an LDM which loads all the registers including the PC), plus the time for the data abort entry (*Texc*), plus the time for FIQ entry (*Tfiq*). At the end of this time ARM920T will be executing the instruction at 0x1C.

Tsyncmax is 3 processor cycles, *Tldm* is 20 cycles, *Texc* is 3 cycles, and *Tfiq* is 2 cycles. The total time is therefore 28 processor cycles. This is just over 1.4 microseconds in a system which uses a continuous 20 MHz processor clock. The maximum IRQ latency calculation is similar, but must allow for the fact that FIQ has higher priority and could delay entry into the IRQ handling routine for an arbitrary length of time. The minimum latency for FIQ or IRQ consists of the shortest time the request can take through the synchroniser (*Tsyncmin*) plus *Tfiq*. This is 4 processor cycles.

RESET

When the **nRESET** signal goes LOW, ARM920T abandons the executing instruction and then continues to fetch instructions from incrementing word addresses.

When **nRESET** goes HIGH again, ARM920T:

1. Overwrites R14_svc and SPSR_svc by copying the current values of the PC and CPSR into them. The value of the saved PC and SPSR is not defined.
2. Forces M[4:0] to 10011 (Supervisor mode), sets the **I** and F bits in the CPSR, and clears the CPSR's T bit.
3. Forces the PC to fetch the next instruction from address 0x00.
4. Execution resumes in ARM state.

ARM920T SYSTEM CONTROL COPROCESSOR (CP15) REGISTER MAP SUMMARY

CP15 defines 16 registers. The register map for CP15 is shown in Table 2-5.

Table 2-5. CP15 Abbreviations

Register	Reads	Writes
0	ID code [1]	Unpredictable
0	Cache type [1]	Unpredictable
1	Control	Control
2	Translation table base	Translation table base
3	Domain access control	Domain access control
4	Unpredictable	Unpredictable
5	Fault status [2]	Fault status [2]
6	Fault address	Fault address
7	Unpredictable	Cache operations
8	Unpredictable	TLB operations
9	Cache lock down [2]	Cache lock down [2]
10	TLB lock down [2]	TLB lock down [2]
11	Unpredictable	Unpredictable
12	Unpredictable	Unpredictable
13	PCSE PID	PCSE PID
14	Unpredictable	Unpredictable
15	Test configuration	Test configuration

NOTES:

1. Register location 0 provides access to more than one register. The register accessed depends on the value of the OPCODE_2 field. See the register description for details.
2. Separate registers for instruction and data. See the register description for details.

ADDRESS IN ARM920T

Three distinct types of address exit in an ARM920T system

- Virtual address (VA).
- Modified Virtual Address (MVA).
- Physical Address (PA).

Below is an example of the address manipulation when the ARM9TDMI requests an instruction (see Figure 2-16)

1. The *Instruction VA* (IVA) is issued by ARM9TDMI.
2. This is translated by the ProcID to the instruction MVA (IMVA). It is the IMVA that the Instruction Cache (ICache) and MMU see.
3. If the protection check carried out by the IMMU on the IMVA does not abort, and the IMVA tag is in the ICache, the instruction data is returned to the ARM9TDMI.
4. If the ICache misses (the IMVA tag is not in the ICache), then the IMMU performs a translation to produce the *instruction PA* (IPA). This address is given to the AMBA bus interface to perform an external access.

Table 2-6. Address Type in ARM920T

Domain	ARM9TDMI	Caches and TLBs	AMBA bus
Address	Virtual (VA)	Modified Virtual (MVA)	Physical (PA)

ACCESSING CP15 REGISTERS

The terms and abbreviations shown in Table 2-7 are used throughout this section.

Table 2-7. CP15 Abbreviations

Term	Abbreviation	Description
unpredictable	UNP	For reads, the data returned when reading from this location is unpredictable; It can have any value. For writes, writing to this location causes unpredictable behavior, or an unpredictable change in device configuration.
should be zero	SBZ	When writing to this location, all bits of this field should be 0.

In all cases, reading from, or writing any data values to any CP15 registers, including those fields specified as *unpredictable or should be zero*, does not cause any permanent damage.

All CP15 register bits that are defined and contain state, are set to zero by **BnRES** except the V bit in register 1, which takes the value of macrocell input **VINITHI** when **BnRES** is asserted.

You can only access CP15 registers with MRC and MCR instructions in a privileged mode. The instruction bit pattern of the MCR and MRC instructions is shown in Figure 2-7. The assembler for these instructions is:

```
MCR/MRC{cond} P15,opcode_1,Rd,CRn,CRm,opcode_2
```

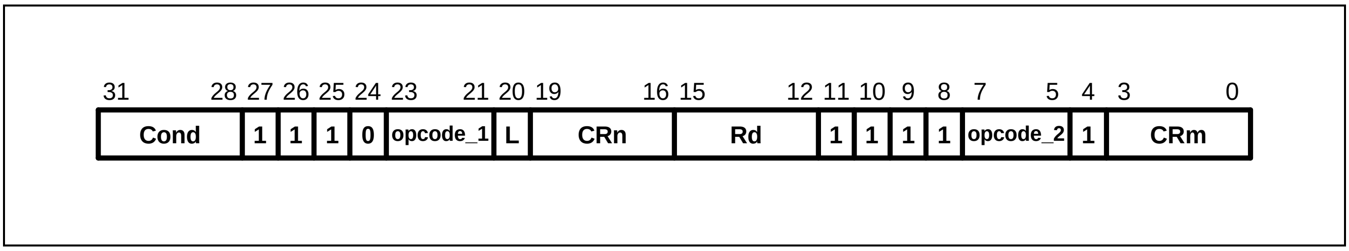


Figure 2-7. CP15 MRC and MCR Bit Pattern

Instructions CDP, LDC and STC, together with unprivileged MRC and MCR instructions to CP15, cause the undefined instruction trap to be taken. The CRn field of MRC and MCR instructions specifies the coprocessor register to access. The CRm field and opcode_2 field specify a particular action when addressing registers. The L bit distinguishes between an MRC (L=1) and an MCR (L=0).

NOTE

Attempting to read from a non-readable register, or to write to a non-writable register cause unpredictable results. The opcode_1, opcode_2 and CRm fields should be zero, except when the values specified are used to select the desired operations, in all instructions that access CP15. Using other values results in unpredictable behavior.

REGISTER 0: ID CODE REGISTER

This is a read-only register that returns a 32-bit device ID code.

You can access the ID code register by reading CP15 register 0 with the opcode_2 field set to any value other than 1 (the CRm field should be zero when reading). For example:

```
        MRC     p15, 0, Rd, c0, c0, 0                   ; returns ID register
```

The contents of the ID code are shown in Table 2-8.

Table 2-8. Register 0, ID Code

Register Bits	Function	Value
31:24	Implementer	0x41
23:20	Specification revision	0x1
19:16	Architecture (ARMv4T)	0x2
15:4	Part number	0x920
3:0	Layout revision	Revision

REGISTER 0, CACHE TYPE REGISTER

This is a read-only register that contains information about the size and architecture of the caches, allowing operating systems to establish how to perform such operations as cache cleaning and lockdown. All ARMv4T and later cached processors contain this register, allowing RTOS vendors to produce future-proof versions of their operating systems.

You can access the cache type register by reading CP15 register 0 with the opcode_2 field set to 1. For example:

```
        MRC     p15, 0, Rd, c0, c0, 1                   ; returns cache details
```

The format of the cache type register is shown in Figure 2-8.

31	30	29	28 25	24	23 12	11 0
0	0	0	ctype	S	Dsize	Isize

ctype
The ctype field dtermines the cache type.

S bit
Specifies whether the cache is a unified cache or separate instruction and data caches.

Dsize
Specifies the size, line length, and associativity of the data cache.

Isize
Specifies the size, line length, and associativity of the instruction cache.

Figure 2-8. Cache Type Register Format

The Dsize and Isize fields in the cache type register have the same format. This is shown in Figure 2-9.

11	10	9	8 7 6	5 4 3	2	1 0
0	0	0	size	assoc	M	len
23	22	21	20 19 18	17 16 15	14	13 12

size The size field determines the cache size in conjunction with the M bit.

assoc The assoc field determines the cache associativity in conjunction with the M bit.

M bit The multiplier bit. Determines the cache size and cache associativity values in conjuctioin with the size and assoc fields.

len The len field determines the line length of the cache.

Figure 2-9. Dsize and Isize Field Format

The register values for the ARM920T cache type register are listed in Table 2-9.

Table 2-9. Cache Type Register Format

Function		Register Bits	Value
Reserved		[31:29]	0b000
ctype		[28:25]	0b0110
S		[24]	0b1 = Harvard cache
Dsize	Reserved	[23:21]	0b000
	size	[20:18]	0b101 = 16KB
	assoc	[17:15]	0b110 = 64-way
	M	[14]	0b0
	len	[13:12]	0b10 = 8 words per line (32 bytes)
Isize	Reserved	[11:9]	0b000
	size	[8:6]	0b101 = 16KB
	assoc	[5:3]	0b110 = 64-way
	M	[2]	0b0
	len	[1:0]	0b10 = 8 words per line (32 bytes)

Bits [28:25] indicate which major cache class the implementation falls into. 0x6 means that the cache provides:

— cache-clean-step operation

— cache-flush-step operation

— lockdown facilities

The size of the cache is determined by the size field and the M bit. The M bit is 0 for the data and instruction caches. Bits [20:18] for the *Data Cache* (DCache) and bits [8:6] for the *Instruction Cache* (ICache) are the size field. Table 2-10 shows the cache size encoding.

Table 2-10. Cache Size Encoding (M=0)

Size Field	Cache Size
0b000	512B
0b001	1KB
0b010	2KB
0b011	4KB
0b100	8KB
0b101	16KB
0b110	32KB
0b111	64KB

The associativity of the cache is determined by the assoc field and the M bit. The M bit is 0 for the data and instruction caches. Bits [17:15] for the DCache and bits [5:3] for the ICache are the assoc field. Table 2-11 shows the cache associativity encoding.

Table 2-11. Cache Associativity Encoding (M=0)

assoc Field	Associativity
0b000	Direct mapped
0b001	2-way
0b010	4-way
0b011	8-way
0b100	16-way
0b101	32-way
0b110	64-way
0b111	128-way

The line length of the cache is determined by the len field. Bits [13:12] for the DCache and bits [1:0] for the ICache are the len field. Table 2-12 shows the line length encoding.

Table 2-12. Line Length Encoding

len Field	Cache Line Length
00	2 words (8 bytes)
01	4 words (16 bytes)
10	8 words (32 bytes)
11	16 words (64 bytes)

REGISTER 1, CONTROL REGISTER

This register contains the control bits of the ARM920T. All reserved bits must either be written 0 or 1, as indicated, or written using read-modify-write. The reserved bits have an unpredictable value when read. Use the following instructions to read and write this register:

```
MRC     p15, 0, Rd, c1, c0, 0                ; read control register
MCR     p15, 0, Rd, c1, c0, 0                ; write control register
```

All defined control bits are set to 0 on reset, except the V bit. The V bit is set to 0 at reset if the **VINTHI** pin is LOW, or 1 if the **VINTHI** pin is HIGH, The functions of the control bits are shown in Table 2-13.

Table 2-13. Control register 1 bit functions

Register bits	Name	Function	Value
[31]	iA bit	Asynchronous clock select	See table 2-14.
[30]	nF bit	notFastBus select	See table 2-14.

[29:15]	–	Reserved	Read = Unpredictable.
			Write = Should be zero.

[14]	RR bit	Round robin replacement	0 = Random replacement. 1 = Round-robin replacement.
[13]	V bit	Base location of exception registers	0 = Low address = 0x0000 0000. 1 = High address = 0xFFFF 0000.
[12]	I bit	ICache enable	0 = Icache disabled. 1 = Icache enabled.

[11:10]	–	Reserved	Read = 00.
			Write = 00.

[9]	R bit	ROM protection	This bit modifies the MMU protection system. See domain access control in chapter 5 Memory Management Unit).
[8]	S bit	System protection	This bit modifies the MMU protection system. See domain access control in chapter 5 Memory Management Unit).
[7]	B bit	Endiannes	0 = Little-endian operation. 1 = Big-endian operation.

[6:3]	–	Reserved	Read = 1111.
			Write = 1111.

[2]	C bit	DCache enable	0 = DCache disabled. 1 = DCache enabled.
[1]	A bit	Alignment fault enable	Data address alignment fault checking. 0 = Fault checking disabled. 1 = Fault checking enabled.
[0]	M bit	MMU enable	0 = MMU disabled. 1 = MMU enabled.

Register 1 bits [31:30] select the clocking mode of the ARM920T, as shown in Table 2-14.

Table 2-14. Clocking Modes

Clocking mode	iA	nF
FastBus mode	0	0
Synchronous	0	1
Reserved	1	0
Asynchronous	1	1

Enabling the MMU

You must take care with the address mapping of the code sequence used to enable the MMU (see *Enabling the MMU* in chapter 5 *Memory Management Unit*).

See *Enabling and disabling the ICache* in chapter 4 *Cache, Write Buffer, and Physical Address TAG (PA TAG) RAM* and *Enabling and disabling the DCache and write buffer* in chapter 4 *Cache, Write Buffer, and Physical Address TAG (PA TAG) RAM* for the restrictions and the effects fo having caches enabled with the MMU disabled.

REGISTER 2, TRANSLATION TABLE BASE (TTB) REGISTER

This is the *Translation Table Base* (TTB) register, for the currently active first-level translation table. The contents of register 2 are shown in Table 2-15.

Reading from register 2 returns the pointer to the first-level translation table in bits [31:14]. Writing to register 2 updates the pointer to the first-level translation table from bits [31:14] of the written value.

Bits[13:0] should be zero when written, and are unpredictable when read.

Table 2-15. Register 2, Translation Table Base

Register bits	Function
[31:14]	Pointer to first-level translation table base. Read/write.

[13:0]	Reserved:
	Read = Unpredictable.
	Write = Should be zero.

You can use the following instructions to access the TLB:

```
MRC     p15, 0, Rd, c2, c0, 0                 ; read TTB register
MCR     p15, 0, Rd, c2, c0, 0                 ; write TTB register
```

REGISTER 3, DOMAIN ACCESS CONTROL REGISTER

Register 3 is the read and write domain access control register, consisting of 16 2-bit fields. Each of these 2-bit fields defines the access permissions for the domains shown in Table 2-16.

Table 2-16. Register 3, Domain Access Control

Register Bits	Domain
[31:30]	D15
[29:28]	D14
[27:26]	D13
[25:24]	D12
[23:22]	D11
[21:20]	D10
[19:18]	D9
[17:16]	D8
[15:14]	D7
[13:12]	D6
[11:10]	D5
[9:8]	D4
[7:6]	D3
[5:4]	D2
[3:2]	D1
[1:0]	D0

The encoding of the two bit domain access permission field is given in *Domain access control* in chapter 5 *Memory Management Unit*. You can use the following instructions to access the domain access control register:

```
MRC     p15, 0, Rd, c3, c0, 0            ; read domain 15:0 access permissions
MCR     p15, 0, Rd, c3, c0, 0            ; write domain 15:0 access permissions
```

REGISTER 4, RESERVED

You must not access (read or write) this register because it causes unpredictable behavior.

REGISTER 5, FAULT STATUS REGISTER

Register 5 is the *fault status register* (FSR). The FSR contains the source of the last data fault, indicating the domain and type of access being attempted when the Data Abort occurred.

Table 2-17. Register 5, Fault status register

Register bits	Description

[31:9]	Read = Unpredictable
	Write = Should be zero

[8]	Read = 0
	Write = Should be zero

[7:4]	Domain being accessed when fault occurred (D15 – D0)
[3:0]	Fault type

The fault type encoding is shown in *Fault address and fault status registers* on chapter 5 *Memory Management Unit*.

The data FSR is defined in ARMv4T. Additionally, a pipelined prefetch FSR is available, for debug purpose only. The pipeline matches that of the ARM9TDMI.

You can use the following instructions to access the data and prefetch FSR:

```
MRC     p15, 0, Rd, c5, c0, 0                 ; read data FSR value
MCR     p15, 0, Rd, c5, c0, 0                 ; write data FSR value
MRC     p15, 0, Rd, c5, c0, 1                 ; read prefetch FSR value
MCR     p15, 0, Rd, c5, c0, 1                 ; write prefetch FSR value
```

The ability to write to the FSR is useful for a debugger to restore the value of the FSR. You must write to the register using the read-modify-write method. Birs [31:8] should be zero.

REGISTER 6, FAULT ADDRESS REGISTER

Register 6 is the *fault address register* (FAR). This contains the MVA of the access being attempted when the last fault occurred. The FAR is only updated for data faults, not for prefetch faults. (You can fined the address for a prefetch fault in R14.)

You can use the following instructions to access the FAR:

```
MRC     p15, 0, Rd, c6, c0, 0                 ; read FAR data
MCR     p15, 0, Rd, c6, c0, 0                 ; write FAR data
```

The ability to write to the FAR is provide to allow a debugger to restore a previous state.

REGISTER 7, CACHE OPERATIONS REGISTER

Register 7 is a write-only register used to manage the ICache and Dcache.

The cache operations provided by register 7 are described in Table 2-18.

Table 2-18. Register 7, Function Descriptions

Functions	Description
Invalidate cache	Invalidates all cache data, including any dirty data.(NOTE) Use with caution.
Invalidate single entry using MVA	Invalidates a single cache line, discarding any dirty data. (NOTE) Use with caution.
Clean D single entry using either index or MVA	Writes the specified cache line to main memory, if the line is marked valid and dirty, and marks the line as not dirty. [a] The valid bit is unchanged.
Clean and invalidate D entry using either index or MVA	Writes the specified cache line to main memory, if the line is marked valid and dirty. (NOTE) The line is marked not valid.
Prefetch cache line	Performs an ICache lookup of the specified MVA. If the cache misses, and the region is cacheable, a line fill is performed.

NOTE: Dirty data is data that has been modified in the cache but not yet written to main memory.

The function of each cache operation is selected by the opcode_2 and CRm fields in the MCR instruction used to write CP15 register 7. Writing other opcode_2 or CRm values is unpredictable.

Reading from CP15 register 7 is unpredictable.

Table 2-19 shows instructions that you can use to perform cache operations with register 7.

Table 2-19. Register 7, Cache Operations

Function	Data	Instruction
Invalidate ICache and DCaches	SBZ	MCR p15, 0, Rd, c7, c7, 0
Invalidate ICache	SBZ	MCR p15, 0, Rd, c7, c5, 0
Invalidate I single entry (using MVA)	MVA format	MCR p15, 0, Rd, c7, c5, 1
Prefetch ICache line (using MVA)	MVA format	MCR p15, 0, Rd, c7, c13, 1
Invalidate DCache	SBZ	MCR p15, 0, Rd, c7, c6, 0
Invalidate DCache single entry (using MVA)	MVA format	MCR p15, 0, Rd, c7, c6, 1
Clean DCache single entry (using MVA)	MVA format	MCR p15, 0, Rd, c7, c10, 1
Clean and invalidate DCache entry (using MVA)	MVA format	MCR p15, 0, Rd, c7, c14, 1
Clean DCache single entry (using index)	Index format	MCR p15, 0, Rd, c7, c10, 2
Clean and invalidate DCache entry (using index)	Index format	MCR p15, 0, Rd, c7, c14, 2
Drain write buffer (1)	SBZ	MCR p15, 0, Rd, c7, c10, 4
Wait for interrupt (2)	SBZ	MCR p15, 0, Rd, c7, c0, 4

NOTES:
1. Stops execution until the write buffer has drained.
2. Stops execution in a LOW power state until an interrupt occurs.

The operations that you can carry out on a single cache line identify the line using the data passed in the MCR instruction. The data is interrupted using one of the formats shown in Figure 2-10 or Figure 2-11.

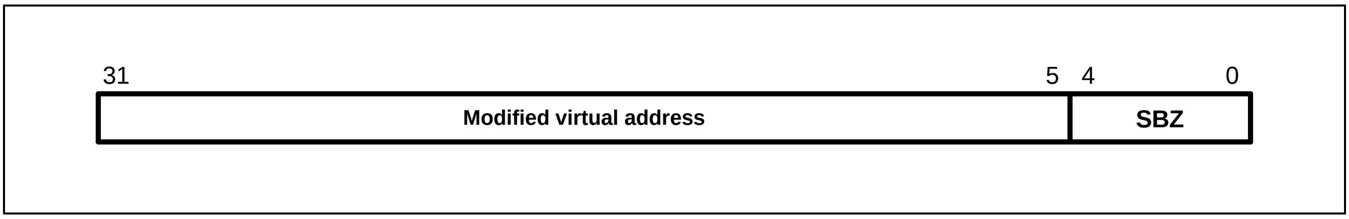


Figure 2-10. Register 7 MVA Format

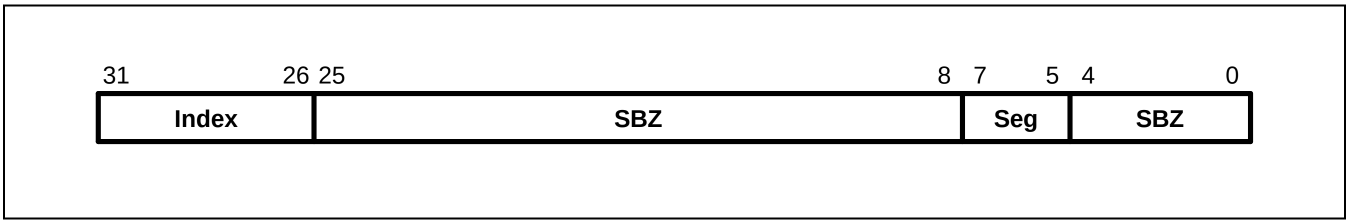


Figure 2-11. Register 7 Index Format

The use of register 7 is described in Chapter 4 *Caches, Write Buffer, and Physical Address TAG (PA TAG) RAM*.

REGISTER 8, TLB OPERATIONS REGISTER

Register 8 is a write-only register used to manage the *Translation Lookaside Buffers* (TLBs), the instruction TLB, and the data TLB.

Five TLB operations are defined and you can select the function to be performed with the opcode_2 and CRm fields in the MCR instruction used to write CP15 register 8. Writing other opcode_2 or CRm values is unpredictable. Reading from CP15 register 8 is unpredictable.

Table 2-20 shows instructions that you can use to perform TLB operations using register 8.

Table 2-20. Register 8, TLB Operations

Function	Data	Instruction
Invalidate TLB(s)	SBZ	MCR p15,0,Rd,c8,c7,0
Invalidate I TLB	SBZ	MCR p15,0,Rd,c8,c5,0
Invalidate I TLB single entry (using MVA)	MVA format	MCR p15,0,Rd,c8,c5,1
Invalidate D TLB	SBZ	MCR p15,0,Rd,c8,c6,0
Invalidate D TLB single entry (using MVA)	MVA format	MCR p15,0,Rd,c8,c6,1

NOTE

These functions invalidate all the unpreserved entries in the TLB. Invalidate TLB single entry functions invalidate any TLB entry corresponding to the MVA given in Rd, regardless of its preserved state. See *Register 10, TLB lockdown register*.

Figure 2-12 shows the MVA format used for operations on single entry TLB lines using register 8.

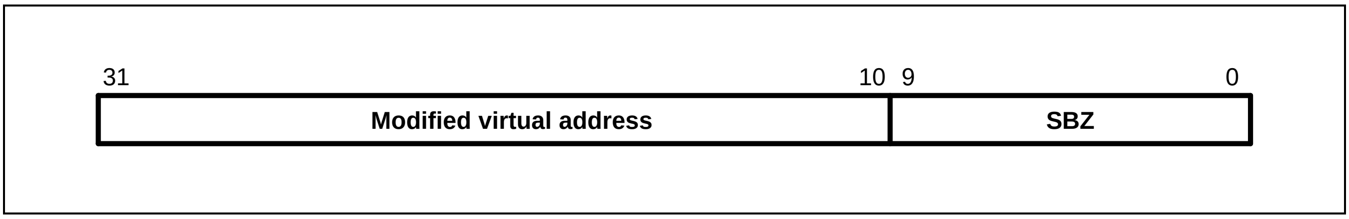


Figure 2-12. Register 8 MVA Format

REGISTER 9, CACHE LOCKDOWN REGISTER

Register 9 is the cache lockdown register. The cache lockdown register is 0x0 on reset. The cache lockdown register allows software to control which cache line in the ICache or DCache respectively is loaded for a linefill and to prevent lines in the ICache or DCache from being evicted during a linefill, locking them into the cache.

There is a register for each of the ICache and DCache. The value of opcode_2 determines which cache register to access:

opcode_2 = 0x0 accesses the DCache register

opcode_2 = 0x1 accesses the ICache register.

The Opcode_1 and CRm fields should be zero.

Reading CP15 register 9 returns the value of the cache lockdown register, which is the base pointer for all cache segments.

NOTE

Only bits [31:26] are returned. Bits [25:0] are unpredictable.

Writing CP15 register 9 updates the cache lockdown register, both the base and the current victim pointer for all cache segments. Bits [25:0] should be zero.

The victim counter specifies the cache line to be used as the victim for the next linefill. This is incremented using either a random or round-robin replacement policy, determined by the state of the RR bit in register 1. The victim counter generates values in the range (base to 63). This locks lines with index values in the range (0 to base-1). If base = 0, there are no locked lines.

Writing to CP15 register 9 updates the base pointer and the current victim pointer. The next linefill uses, and then increments, the victim pointer. The victim pointer continues incrementing on linefills, and wraps around to the base pointer. For example, setting the base pointer to 0x3 prevents the victim pointer from selecting entries 0x0 to 0x2, locking them into the cache. shows how you can load a cache line into ICache line 0 and lock it down.

Example 2-1: Load a cache line into ICache line 0 and lock it down

MCR to CP15 register 9, opcode_2 = 0x1, Victim=Base=0x0

MCR I prefetch. Assuming the ICache misses, a linefill occurs to line 0.

MCR to CP15 register 9, opcode_2 = 0x1, Victim=Base=0x1

More ICache linefills now occur into lines 1-63.

Example 2-2 shows how you can load a cache line into DCache line 0 and lock it down.

Example 2-2: Load a cache line into DCache line 0 and lock it down

MCR to CP15 register 9, opcode_2 = 0x0, Victim=Base=0x0

Data load (LDR/LDM). Assuming the DCache misses, a linefill occurs to line 0.

MCR to CP15 register 9, opcode_2 = 0x0, Victim=Base=0x1

More DCache linefills now occur into lines 1-63.

NOTE

Writing CP15 register 9, with the CRm field set to b0001, updates the current victim pointer only for the specified segment. Bits [31:26] specify the victim. Bits [7:5] specify the segment (for a 16KB cache). All other bits should be zero. This encoding is intended for debug use. You are not recommended to use this encoding.

Figure 2-13 shows the format of bits in register 9.

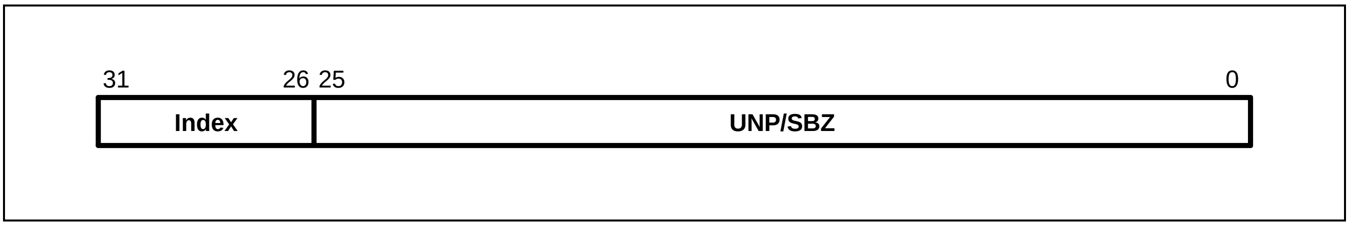


Figure 2-13. Register 9

Table 2-21 shows the instructions you can use to access the cache lockdown register.

Table 2-21. Register 9, Accessing the Cache Lockdown

Function	Data	Instruction
Read DCache lockdown base	Base	MRC p15,0,Rd,c9,c0,0
Write DCache victim and lockdown base	Victim=Base	MCR p15,0,Rd,c9,c0,0
Read ICache lockdown baseBase	Base	MRC p15,0,Rd,c9,c0,1
Write ICache victim and lockdown base	Victim=Base	MCR p15,0,Rd,c9,c0,1

REGISTER 10, TLB LOCKDOWN REGISTER

Register 10 is the TLB lockdown register. The TLB lockdown register is 0x0 on reset. There is a TLB lockdown register for each of the TLBs, the value of opcode_2 determines which TLB register to access:

opcode_2 = 0x0 accesses the D TLB register

opcode_2 = 0x1 accesses the I TLB register.

Reading CP15 register 10 returns the value of the TLB lockdown counter base register, the current victim number, and the preserve bit (P bit). Bits [19:1] are unpredictable when read.

Writing CP15 register 10 updates the TLB lockdown counter base register, the current victim pointer, and the state of the preserve bit. Bits [19:1] should be zero when written.

Table 2-22 shows the instructions you can use to access the TLB lockdown register.

Table 2-22. Register 10, Accessing the TLB Lockdown

Function	Data	Instruction
Read D TLB lockdown	TLB lockdown	MRC p15,0,Rd,c10,c0,0
Write D TLB lockdown	TLB lockdown	MCR p15,0,Rd,c10,c0,0
Read I TLB lockdown	TLB lockdown	MRC p15,0,Rd,c10,c0,1
Write I TLB lockdown	TLB lockdown	MCR p15,0,Rd,c10,c0,1

Figure 2-14 shows the format of bits in register 10.

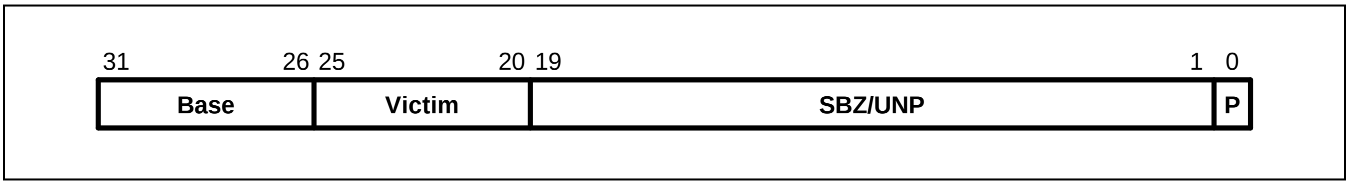


Figure 2-14. Register 10

The entries in the TLBs are replaced using a round-robin replacement policy. This is implemented using a victim counter that counts from entry 0 up to 63, and then wraps back round to the base value and continues counting, wrapping around to the base value from 63 each time.

There are two mechanisms available for ensuring entries are not removed from the TLB:

Locking an entry down prevents it from being selected for overwriting during a table walk. You can do this by programming the base value to which the victim counter reloads. For example, if the bottom 3 entries (0-2) are to be locked down, you must program the base counter to 3.

You can preserve an entry during an Invalidate All instruction. You can do this by ensuring the P bit is set when the entry is loaded into the TLB. Examples that show how you can load a single entry into the I and D TLBs at location 0, make it immune to Invalidate All, and lock it down are shown in Example 2-3 and Example 2-4.

Example 2-3: Load a single entry into I TLB location 0, make it immune to Invalidate All and lock it down

MCR to CP15 register 10, opcode_2 = 0x1, Base Value = 0, Current Victim = 0, P = 1

MCR I prefetch. Assuming an I TLB miss occurs, then entry 0 is loaded.

MCR to CP15 register 10, opcode_2 = 0x1, Base Value = 1, Current Victim = 1, P = 0

Example 2-4: Load a single entry into D TLB location 0, make it immune to Invalidate All and lock it down

MCR to CP15 register 10, opcode_2 = 0x0, Base Value = 0, Current Victim = 0, P = 1

Data load (LDR/LDM) or store (STR/STM). Assuming a D TLB miss occurs, then entry 0 is loaded.

MCR to CP15 register 10, opcode_2 = 0x0, Base Value = 1, Current Victim = 1, P = 0

REGISTERS 11, 12, AND 14, RESERVED

Accessing (reading or writing) any of these registers causes unpredictable behavior.

REGISTER 13, FCSE PID REGISTER

Register 13 is the *Fast Context Switch Extension* (FCSE) *Process Identifier* (PID) register. The FCSE PID register is 0x0 on reset.

Reading from CP15 register 13 returns the value of the FCSE PID. Writing CP15 register 13 updates the FCSE PID to the value in bits [31:25]. Bits [24:0] should be zero.

Register 13 bit assignments are shown in Figure 2-15.

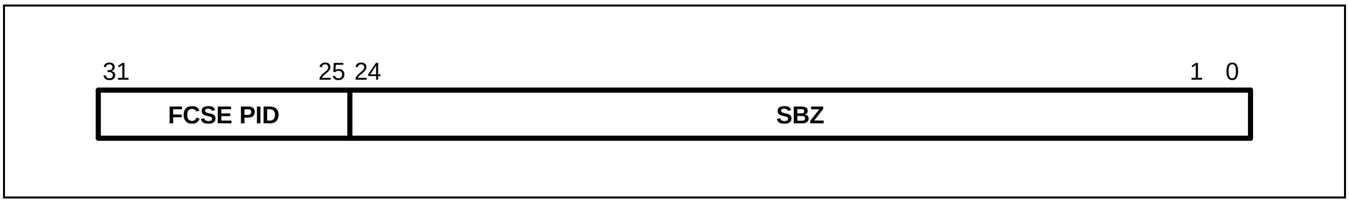


Figure 2-15. Register 13

You can access register 13 using the following instructions:

```
MRC     p15, 0, Rd, c13, c0, 0          ; read FCSE PID
MCR     p15, 0, Rd, c13, c0, 0          ; write FCSE PID
```

Using the FCSE process identifier (FCSE PID)

Addresses issued by the ARM9TDMI core in the range 0 to 32MB are translated by CP15 register 13, the FCSE PID register. Address A becomes A + (FCSE_PID x 32MB). It is this translated address that is seen by both the caches and MMU. Addresses above 32MB undergo no translation. This is shown in Figure 2-16.

The FCSE_PID is a 7-bit field, enabling 128 x 32MB processes to be mapped.

NOTE

If FCSE_PID is zero, as it is on reset, then there is a flat mapping between the ARM9TDMI and the caches and MMU.

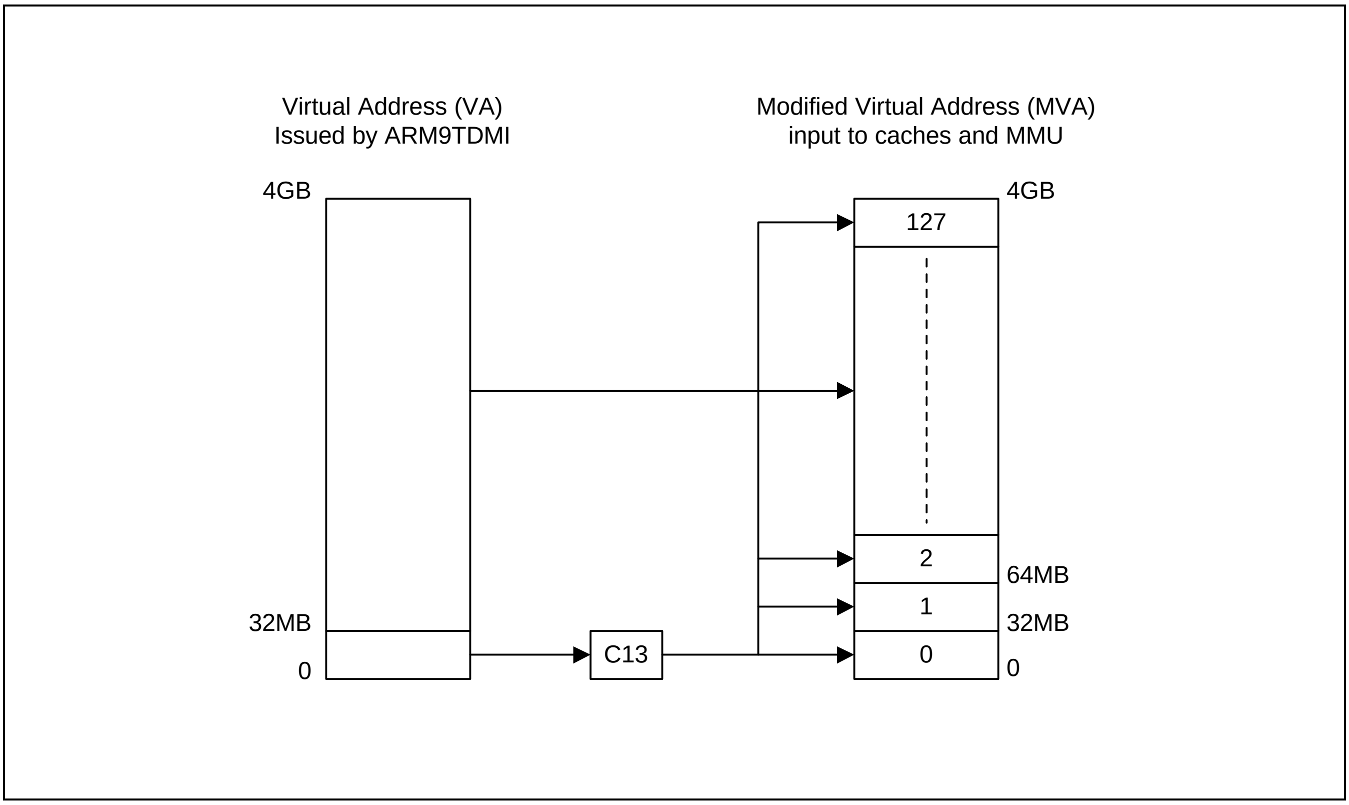


Figure 2-16. Address Mapping using CP15 Register 13

Changing the FCSE PID, performing a fast context switch

To do a fast context switch, write to CP15 register 13. The contents of the caches and TLBs do not have to be flushed after a fast context switch because they still hold valid address tags. The two instructions after the MCR to write the FCSE_PID are fetched with the old FCSE_PID value:

```
{FCSE_PID = 0}
MOV r0, #1:SHL:25                ; Fetched with FCSE_PID = 0
MCR p15,0,r0,c13,c0,0            ; Fetched with FCSE_PID = 0
A1                               ; Fetched with FCSE_PID = 0
A2                               ; Fetched with FCSE_PID = 0
A3                               ; Fetched with FCSE_PID = 1
```

REGISTER 15, TEST CONFIGURATION REGISTER

Register 15 is used for test purposes. Accessing (reading or writing) this register causes the ARM920T to have unpredictable behavior.

3 INSTRUCTIONS SET

INSTRUCTION SET SUMMAY

This chapter describes the ARM instruction set and the THUMB instruction set in the ARM9TDMI core.

The ARM920T implements uses a five-stage pipeline design. These five stages are:

— instruction fetch (F)

— instruction decode (D)

— execute (E)

— data memory access (M)

— register write (W)

ARM implementations are fully interlocked, so that software will function identically across different implementations without concern for pipeline effects. Interlock do affect instruction times. For example, the following sequence suffers a single cycle penalty due to a load-use interlock on register r0:

```
LDR        R0,[R7]
ADD        R5,R0,R1
```

ABOUT THE INSTRUCTION CYCLE SUMMARY

All signals quoted in this chapter are ARM9TDMI signals, and are internal to the ARM920T. In all cases it is assumed that all accesses are from cached regions of memory.

If an instruction causes an external access, either when prefetching instructions or when accessing data, the instruction takes more cycles to complete execution. The additional number of cycles is dependent on the system implementation.

INSTRUCTION CYCLE TIMES

Table 3-1 shows a key to the symbols used in tables in this section

Table 3-1. Symbol Used in Tables

Symbol	Meaning
B	The number of busy-wait states during coprocessor accesses
M	Is in the range 0 to 3, depending on early termination
N	The number of words transferred in an LDM/STM/LDC/STC
C	Coprocessor register transfer (C-cycle)
I	Internal cycle (I-cycle)
N	Non-sequential cycle (N-cycle)
S	Sequential cycle (S-cycle)

Table 3-2 summarize the ARM920T instruction cycle counts and bus activity when executing the ARM instruction set.

Table 3-2. Instruction Cycle Bus Times

Instruction	Cycle	Instruction Bus	Data Bus	Comment
Data Op	1	1S	1I	Normal case
Data Op	2	1S+1I	2I	With register controlled shift
LDR	1	1S	1N	Normal case, not loading PC
LDR	2	1S+1I	1N+1I	Not loading PC and following instruction uses loaded word (1 cycle load-use interlock)
LDR	3	1S+2I	1N+2I	Loaded byte, halfword, or unaligned word used by following instruction (2 cycle load-use interlock)
LDR	5	2S+2I+1N	1N+4I	PC is destination register
STR	1	1S	1N	All cases
LDM	2	1S+1I	1S+1I	Loading 1 Register, not the PC
LDM	N	1S+(n-1)I	1N+(n-1)S	Loading n registers, n > 1, not loading the PC
LDM	n+4	2S+1N+(n+1)I	1N+(n-1)S+4I	Loading n registers including the PC, n > 0
STM	2	1S+1I	1N+1I	Storing 1 Register
STM	N	1S+(n-1)I	1N+(n-1)S	Storing n registers, n > 1
SWP	2	1S+1I	2N	Normal case
SWP	3	1S+2I	2N+1I	Loaded byte used by following instruction
B, BL, BX	3	2S+1N	3I	All cases
SWI, Undefined	3	2S+1N	3I	All cases
CDP	b+1	1S+bI	(1+b)I	All cases
LDC, STC	b+n	1S+(b+n-1)I	bI+1N+(n-1)S	All cases
MCR	b+1	1S+bI	bI+1C	All cases
MRC	b+1	1S+bI	bI+1C	Normal case
MRC	b+2	1S+(b+1)I	(b+I)I+1C	Following instruction uses transferred data
MUL, MLA	2+m	1S+(1+m)I	(2+m)I	All cases
SMULL,UMULL, SMLAL,UMLAL	3+m	1S+(2+m)I	(3+m)I	All cases

Table 3-3 shows the instruction cycle times from the perspective of the data bus.

Table 3-3. Data Bus Instruction Times

Instruction	Cycle time
LDR	1N
STR	1N
LDM, STM	1N+(n-1)S
SWP	1N+1S
LDC, STC	1N+(n-1)S
MCR, MRC	1C

MULTIPLIER CYCLE COUNTS

The number of cycles that a multiply instruction takes to complete depends on which instruction, and on the value of the multiplier-operand. The multiplier-operand is the contents of the register specified by bits [11:8] of the ARM multiply instructions, or bits [2:0] of the Thumb multiply instructions.

For ARM MUL, MLA, SMULL, SMLAL, and Thumb MUL, m is:

1. if bits [31:8] of the multiplier operand are all 0 or all 1
2. if bits [31:16] of the multiplier operand are all 0 or all 1
3. if bits [31:24] of the multiplier operand are all 0 or all 1
4. otherwise.

For ARM UMULL, UMLAL, m is:

1. if bits [31:8] of the multiplier operand are all 0
2. if bits [31:16] of the multiplier operand are all 0
3. if bits [31:24] of the multiplier operand are all 0
4. otherwise.

INTERLOCKS

Pipeline interlocks occur when the data required for an instruction is not available due to the incomplete execution of an earlier instruction. When an interlock occurs, instruction fetches stop on the instruction memory interface of the ARM920T. Four examples are given in.

- Example 3-1
- Example 3-2
- Example 3-3
- Example 3-4

Example 3-1: Single load interlock

In this example, the following code sequence is executed:

```
LDR     R0, [R1]
ADD     R2, R0, R1
```

The ADD instruction cannot start until the data is returned from the load. The ADD instruction therefore, has to delay entering the Execute stage of the pipeline by one cycle. The behavior on the instruction memory interface is shown in Figure 3-1.

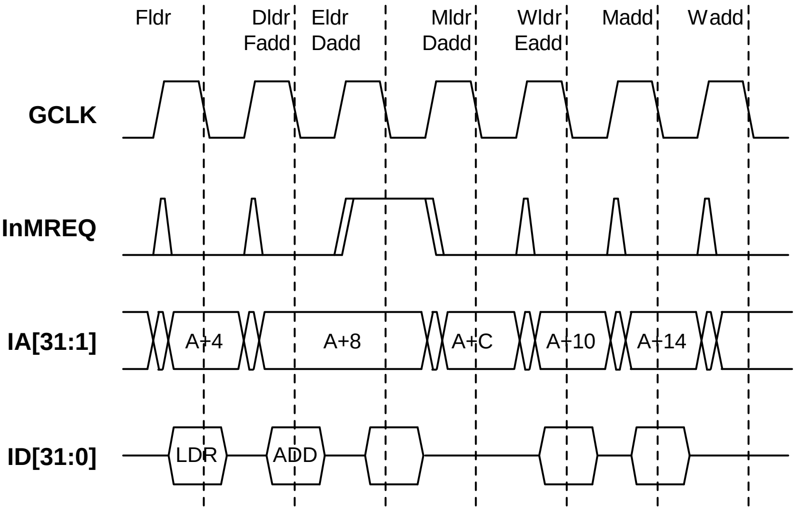


Figure 3-1. Single load interlock timing

Example 3-2: Two cycle load interlock

In this example, the following code sequence is executed:

```
LDRB    R0, [R1, #1]
ADD     R2, R0, R1
```

Now, because a rotation must occur on the loaded data, there is a second interlock cycle. The behavior on the instruction memory interface is shown in Figure 3-2.

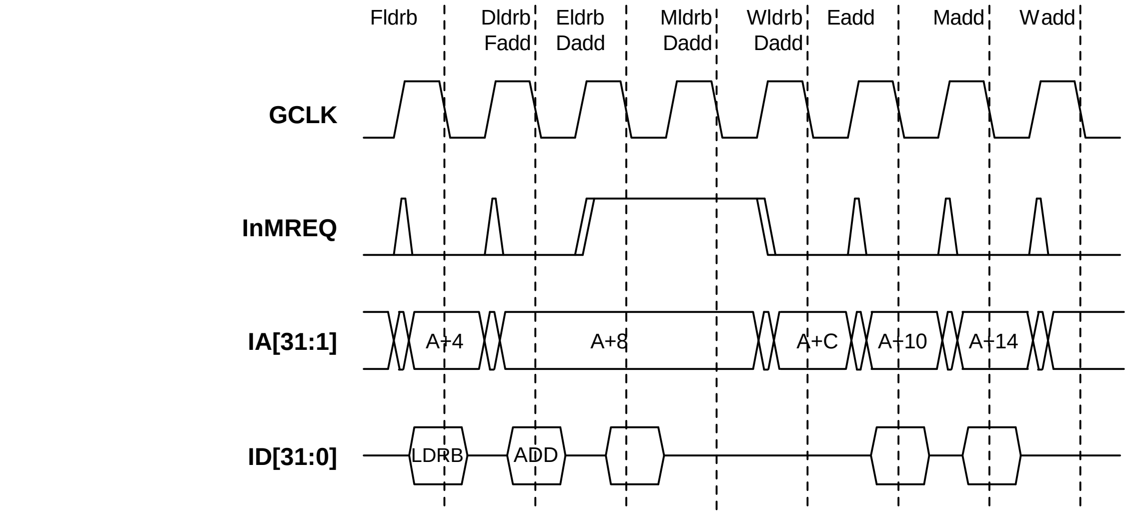


Figure 3-2. Tow cycle load interlock

Example 3-3: ldm interlock

In this example, the following code sequence is executed:

```
LDM     R12, {R1–R3}
ADD     R2, R2, R1
```

The LDM takes three cycle to execute in the Memory stage of the pipeline. The ADD is therefore delayed until the LDM begins its final memory fetch. The behavior on the instruction memory interface is shown in Figure 3-3.

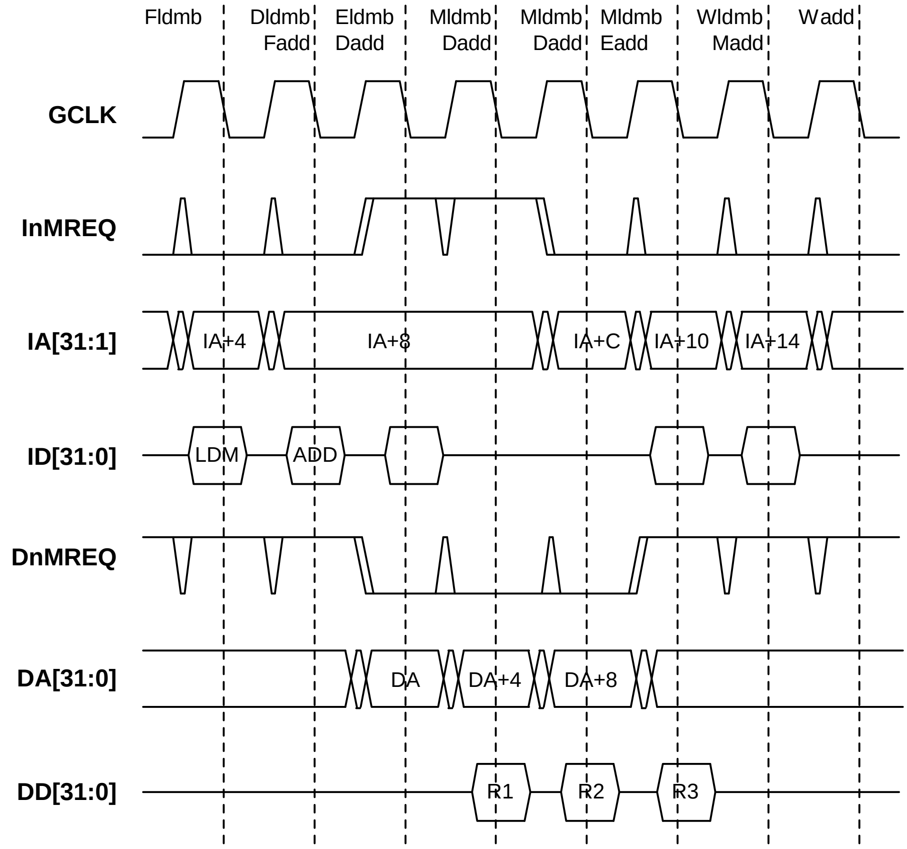


Figure 3-3. LDM interlock

Example 3-4: LDM Dependent Interlock

In this example, the following code sequence is executed:

```
LDM     R12, {R1–R3}
ADD     R4, R3, R1
```

The code is the same code as in example 3-3, but in this instance the ADD instruction uses R3. Due to the nature of load multiples, the lowest register specified is transferred first, and the highest specified register last. Because the ADD is dependent on R3, there must be another cycle of interlock while R3 is loaded. The behavior on the instruction and data memory interface is shown in Figure 3-4.

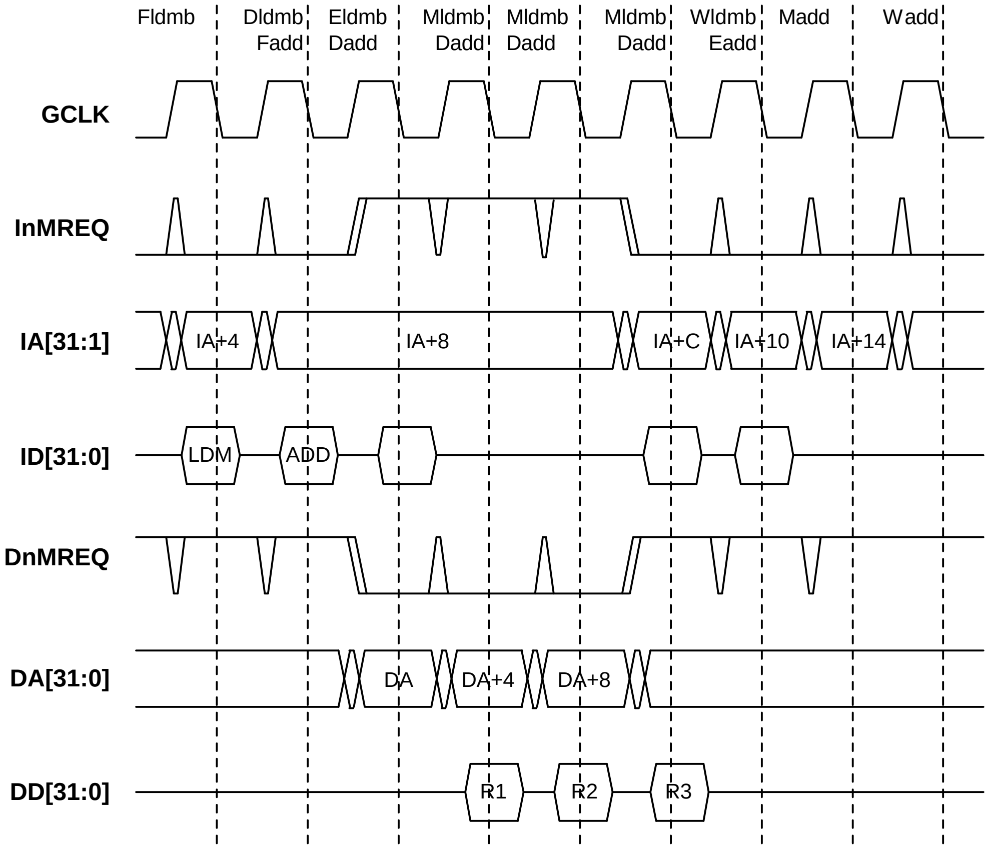
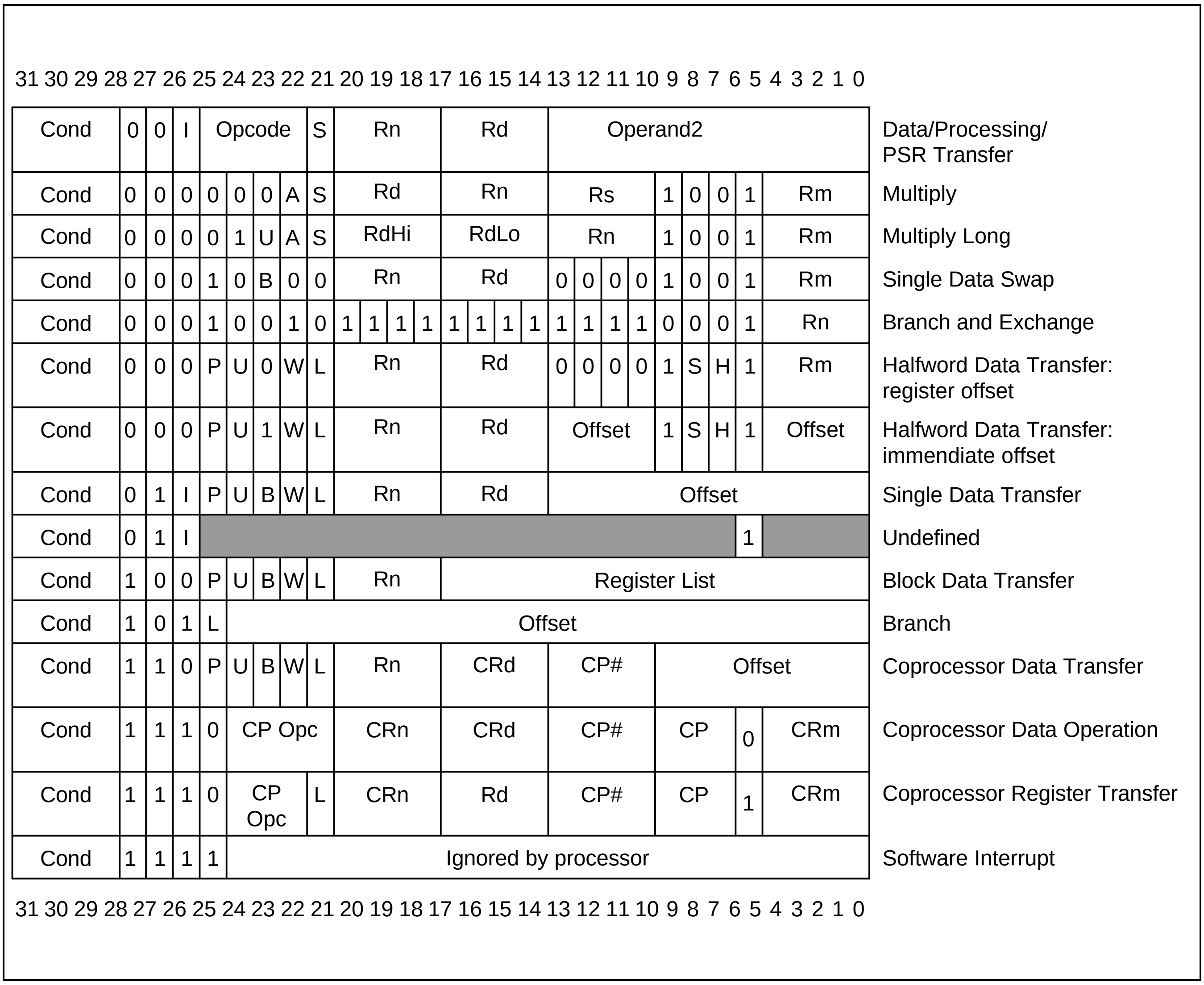


Figure 3-4. LDM dependent interlock

FORMAT SUMMARY

The ARM instruction set formats are shown below.



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Cond				0	0	I	Opcode				S	Rn				Rd				Operand2												Data/Processing/ PSR Transfer
Cond				0	0	0	0	0	0	A	S	Rd				Rn				Rs				1	0	0	1	Rm				Multiply
Cond				0	0	0	0	1	U	A	S	RdHi				RdLo				Rn				1	0	0	1	Rm				Multiply Long
Cond				0	0	0	1	0	B	0	0	Rn				Rd				0	0	0	0	1	0	0	1	Rm				Single Data Swap
Cond				0	0	0	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	Rn				Branch and Exchange
Cond				0	0	0	P	U	0	W	L	Rn				Rd				0	0	0	0	1	S	H	1	Rm				Halfword Data Transfer: register offset
Cond				0	0	0	P	U	1	W	L	Rn				Rd				Offset				1	S	H	1	Offset				Halfword Data Transfer: immendiate offset
Cond				0	1	I	P	U	B	W	L	Rn				Rd				Offset												Single Data Transfer
Cond				0	1	I	▨																				1	▨				Undefined
Cond				1	0	0	P	U	B	W	L	Rn				Register List																Block Data Transfer
Cond				1	0	1	L	Offset																								Branch
Cond				1	1	0	P	U	B	W	L	Rn				CRd				CP#				Offset								Coprocessor Data Transfer
Cond				1	1	1	0	CP Opc				CRn				CRd				CP#				CP			0	CRm				Coprocessor Data Operation
Cond				1	1	1	0	CP Opc			L	CRn				Rd				CP#				CP			1	CRm				Coprocessor Register Transfer
Cond				1	1	1	1	Ignored by processor																								Software Interrupt
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Figure 3-5. ARM Instruction Set Format

NOTE

Some instruction codes are not defined but they do not cause the Undefined instruction trap to be taken, for instance a Multiply instruction with bit 6 changed to a 1. These instructions should not be used, as their action may change in future ARM implementations.

INSTRUCTION SUMMARY

Table 3-4. The ARM Instruction Set

Mnemonic	Instruction	Action
ADC	Add with carry	Rd: = Rn + Op2 + Carry
ADD	Add	Rd: = Rn + Op2
AND	ANDo	Rd: = Rn AND Op2
B	Branch	R15: = address
BIC	Bit Clear	Rd: = Rn AND NOT Op2
BL	Branch with Link	R14: = R15, R15: = address
BX	Branch and Exchange	R15: = Rn, T bit: = Rn[0]
CDP	Coprocessor Data Processing	(Coprocessor-specific)
CMN	Compare Negative	CPSR flags: = Rn + Op2
CMP	Compare	CPSR flags: = Rn - Op2
EOR	Exclusive OR	Rd: = (Rn AND NOT Op2) OR (Op2 AND NOT Rn)
LDC	Load coprocessor from memory	Coprocessor load
LDM	Load multiple registers	Stack manipulation (Pop)
LDR	Load register from memory	Rd: = (address)
MCR	Move CPU register to coprocessor register	cRn: = rRn {<op>cRm}
MLA	Multiply Accumulate	Rd: = (Rm × Rs) + Rn
MOV	Move register or constant	Rd: = Op2
MRC	Move from coprocessor register to CPU register	Rn: = cRn {<op>cRm}
MRS	Move PSR status/flags to register	Rn: = PSR
MSR	Move register to PSR status/flags	PSR: = Rm
MUL	Multiply	Rd: = Rm × Rs
MVN	Move negative register	Rd: = 0 × FFFFFFFF EOR Op2
ORR	OR	Rd: = Rn OR Op2
RSB	Reverse Subtract	Rd: = Op2 - Rn
RSC	Reverse Subtract with Carry	Rd: = Op2 - Rn - 1 + Carry
SBC	Subtract with Carry	Rd: = Rn - Op2 - 1 + Carry
STC	Store coprocessor register to memory	address: = CRn
STM	Store Multiple	Stack manipulation (Push)
STR	Store register to memory	<address>: = Rd
SUB	Subtract	Rd: = Rn - Op2
SWI	Software Interrupt	OS call
SWP	Swap register with memory	Rd: = [Rn], [Rn] := Rm
TEQ	Test bitwise equality	CPSR flags: = Rn EOR Op2
TST	Test bits	CPSR flags: = Rn AND Op2

THE CONDITION FIELD

In ARM state, all instructions are conditionally executed according to the state of the CPSR condition codes and the instruction's condition field. This field (bits 31:28) determines the circumstances under which an instruction is to be executed. If the state of the C, N, Z and V flags meets the conditions encoded by the field, the instruction is executed, otherwise, it is ignored.

There are sixteen possible conditions, each represented by a two-character suffix that can be appended to the instruction's mnemonic. For example, a Branch (B in assembly language) becomes BEQ for "Branch if Equal", which means the Branch will only be taken if the Z flag is set.

In practice, fifteen different conditions may be used: they are listed in Table 3-5. The sixteenth (1111) is reserved, and must not be used.

In the absence of a suffix, the condition field of most instructions is set to "Always" (suffix AL). This means the instruction will always be executed regardless of the CPSR condition codes.

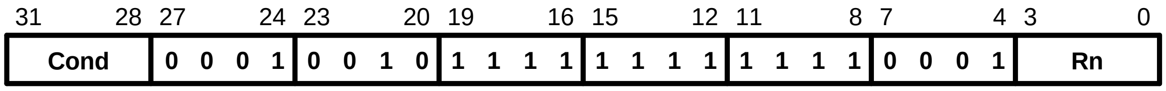
Table 3-5. Condition Code Summary

Code	Suffix	Flags	Meaning
0000	EQ	Z set	equal
0001	NE	Z clear	not equal
0010	CS	C set	unsigned higher or same
0011	CC	C clear	unsigned lower
0100	MI	N set	negative
0101	PL	N clear	positive or zero
0110	VS	V set	overflow
0111	VC	V clear	no overflow
1000	HI	C set and Z clear	unsigned higher
1001	LS	C clear or Z set	unsigned lower or same
1010	GE	N equals V	greater or equal
1011	LT	N not equal to V	less than
1100	GT	Z clear AND (N equals V)	greater than
1101	LE	Z set OR (N not equal to V)	less than or equal
1110	AL	(ignored)	always

BRANCH AND EXCHANGE (BX)

This instruction is executed only if the condition is true. The various conditions are defined in Table 3-5.

This instruction performs a branch by copying the contents of a general register, Rn, into the program counter, PC. The branch causes a pipeline flush and refill from the address specified by Rn. This instruction also permits the instruction set to be exchanged. When the instruction is executed, the value of Rn[0] determines whether the instruction stream would be decoded as ARM or THUMB instructions.



31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
Cond	**0 0 0 1**	**0 0 1 0**	**1 1 1 1**	**1 1 1 1**	**1 1 1 1**	**0 0 0 1**	**Rn**

[3:0] Operand Register
If bit0 of Rn = 1, subsequent instructions decoded as THUMB instructions
If bit0 of Rn =0, subsequent instructions decoded as ARM instructions

[31:28] Condition Field

Figure 3-6. Branch and Exchange Instructions

INSTRUCTION CYCLE TIMES

The BX instruction takes 2S + 1N cycles to execute, where S and N are defined as sequential (S-cycle) and non-sequencial (N-cycle), respectively.

ASSEMBLER SYNTAX

BX - branch and exchange.
BX {cond} Rn

{cond}	; Two character condition mnemonic. See Table 3-5.
Rn	; is an expression evaluating to a valid register number.

USING R15 AS AN OPERAND

If R15 is used as an operand, its behaviour is undefined.

Examples

```
ADR         R0, Into_THUMB + 1          ;  Generate branch target address
                                        ;  and set bit 0 high - hence
                                        ;  arrive in THUMB state.
BX          R0                          ;  Branch and change to THUMB
                                        ;  state.
CODE16                                  ;  Assemble subsequent code as
Into_THUMB                              ;  THUMB instructions
•
•
•
ADR R5, Back_to_ARM                     ;  Generate branch target to word aligned address
                                        ;  - hence bit 0 is low and so change back to ARM state.
BX R5                                   ;  Branch and change back to ARM state.
•
•
•
ALIGN                                   ;  Align words
CODE32                                  ;  Assemble subsequent codes as ARM instructions
Back_to_ARM
```

BRANCH AND BRANCH WITH LINK (B, BL)

The instruction is executed only if the condition is true. The various conditions are defined Table 3-5. The instruction encoding is shown in Figure 3-7, below.

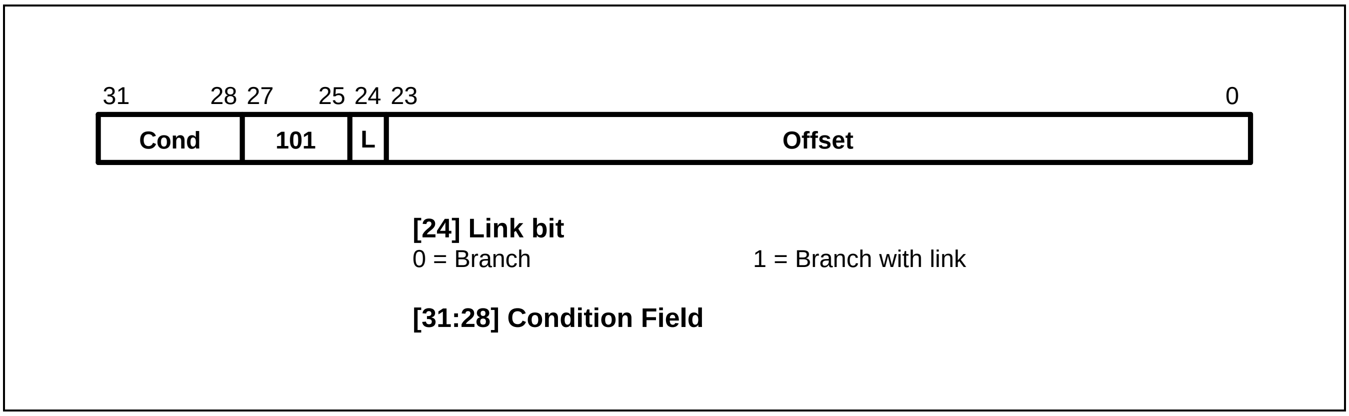


Figure 3-7. Branch Instructions

Branch instructions contain a signed 2's complement 24-bit offset. This is shifted left two bits, sign extended to 32 bits, and added to the PC. The instruction can therefore specify a branch of +/- 32Mbytes. The branch offset must take account of the prefetch operation, which causes the PC to be 2 words (8 bytes) ahead of the current instruction.

Branches beyond +/- 32Mbytes must use an offset or absolute destination which was previously loaded into a register. In this case, the PC should be manually saved in R14 if a Branch with Link type operation is required.

THE LINK BIT

Branch with Link (BL) writes the old PC into the link register (R14) of the current bank. The PC value written into R14 is adjusted to allow for the prefetch, and it contains the address of the instruction following the branch and link instruction. Note that the CPSR is not saved with the PC and R14[1:0] are always cleared.

To return from a routine called by Branch with Link use MOV PC,R14 if the link register is still valid or LDM Rn!,{..PC} if the link register has been saved onto a stack pointed to by Rn.

INSTRUCTION CYCLE TIMES

Branch and Branch with Link instructions take 2S + 1N incremental cycles, where S and N are defined as sequential (S-cycle) and internal (I-cycle).

ASSEMBLER SYNTAX

Items in {} are optional. Items in <> must be present.

B{L}{cond} <expression>

{L}
be

Used to request the Branch with Link form of the instruction. If absent, R14 will not

affected by the instruction.

{cond}

A two-character mnemonic as shown in Table 3-5. If absent, AL (ALways) will be used.

<expression>

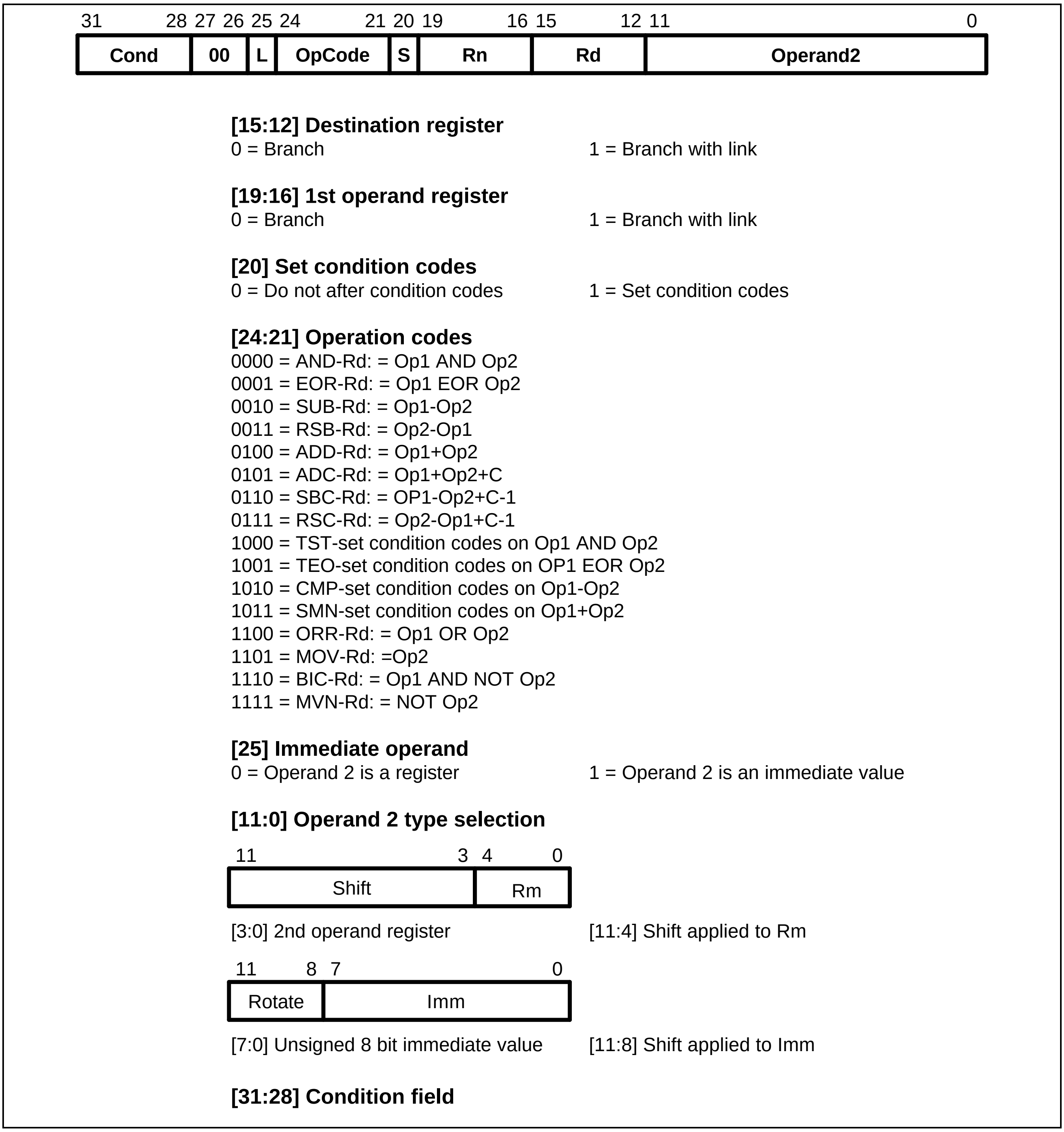
The destination. The assembler calculates the offset.

Examples

```
here    BAL     here              ; Assembles to 0xEAFFFFFE (note effect of PC offset).
        B       there             ; Always condition used as default.
        CMP     R1,#0             ; Compare R1 with zero and branch to fred
                                  ; if R1 was zero, otherwise continue.
        BEQ     fred              ; Continue to next instruction.
        BL      sub+ROM           ; Call subroutine at computed address.
        ADDS    R1,#1             ; Add 1 to register 1, setting CPSR flags
                                  ; on the result, then call subroutine if
        BLCC    sub               ; the C flag is clear, which will be the
                                  ; case unless R1 is held 0xFFFFFFFF.
```

DATA PROCESSING

The data processing instruction is executed only if the condition is true. The conditions are defined in Table 3-5. The instruction encoding is shown in Figure 3-8.



[15:12] Destination register
0 = Branch 1 = Branch with link

[19:16] 1st operand register
0 = Branch 1 = Branch with link

[20] Set condition codes
0 = Do not after condition codes 1 = Set condition codes

[24:21] Operation codes
0000 = AND-Rd: = Op1 AND Op2
0001 = EOR-Rd: = Op1 EOR Op2
0010 = SUB-Rd: = Op1-Op2
0011 = RSB-Rd: = Op2-Op1
0100 = ADD-Rd: = Op1+Op2
0101 = ADC-Rd: = Op1+Op2+C
0110 = SBC-Rd: = OP1-Op2+C-1
0111 = RSC-Rd: = Op2-Op1+C-1
1000 = TST-set condition codes on Op1 AND Op2
1001 = TEO-set condition codes on OP1 EOR Op2
1010 = CMP-set condition codes on Op1-Op2
1011 = SMN-set condition codes on Op1+Op2
1100 = ORR-Rd: = Op1 OR Op2
1101 = MOV-Rd: =Op2
1110 = BIC-Rd: = Op1 AND NOT Op2
1111 = MVN-Rd: = NOT Op2

[25] Immediate operand
0 = Operand 2 is a register 1 = Operand 2 is an immediate value

[11:0] Operand 2 type selection

[3:0] 2nd operand register [11:4] Shift applied to Rm

[7:0] Unsigned 8 bit immediate value [11:8] Shift applied to Imm

[31:28] Condition field

Figure 3-8. Data Processing Instructions

The instruction produces a result by performing a specified arithmetic or logical operation on one or two operands. The first operand is always a register (Rn).

The second operand may be a shifted register (Rm) or a rotated 8-bit immediate value (Imm) according to the value of the I bit in the instruction. The condition codes in the CPSR may be maintained or updated as a result of this instruction, according to the value of the S bit in the instruction.

Certain operations (TST, TEQ, CMP, CMN) do not write the result to Rd. They are used only to perform tests and to set the condition codes on the result and always have the S bit set. The instructions and their effects are listed in Table 3-6.

CPSR FLAGS

The data processing operations may be classified as logical or arithmetic. The logical operations (AND, EOR, TST, TEQ, ORR, MOV, BIC, MVN) perform the logical action on all corresponding bits of the operand or operands to produce the result. If the S bit is set (and Rd is not R15, see below), the V flag in the CPSR will be unaffected, the C flag will be set to the carry out from the barrel shifter (or maintained when the shift operation is LSL #0), the Z flag will be set if and only if the result is all zeros, and the N flag will be set to the logical value of bit 31 of the result.

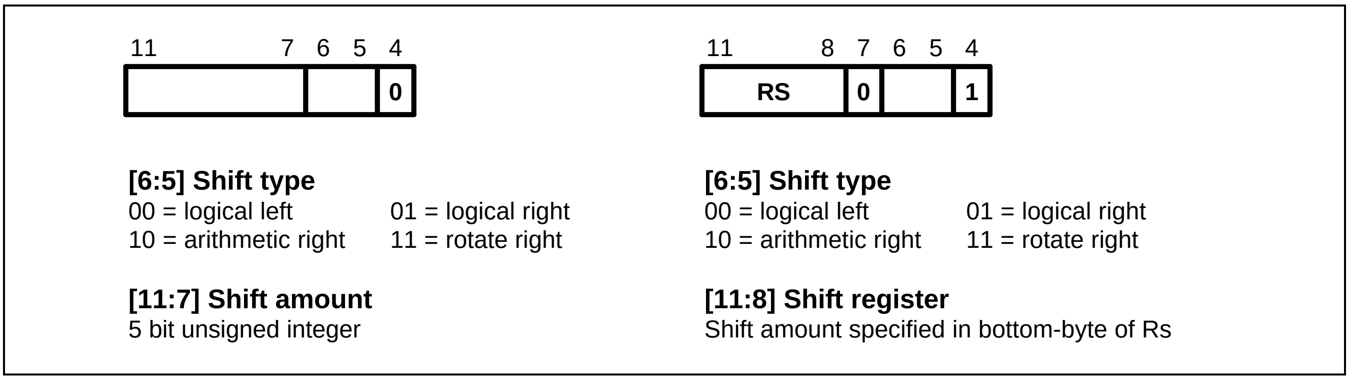
Table 3-6. ARM Data Processing Instructions

Assembler Mnemonic	OP Code	Action
AND	0000	Operand1 AND operand2
EOR	0001	Operand1 EOR operand2
WUB	0010	Operand1 - operand2
RSB	0011	Operand2 operand1
ADD	0100	Operand1 + operand2
ADC	0101	Operand1 + operand2 + carry
SBC	0110	Operand1 - operand2 + carry - 1
RSC	0111	Operand2 - operand1 + carry - 1
TST	1000	As AND, but result is not written
TEQ	1001	As EOR, but result is not written
CMP	1010	As SUB, but result is not written
CMN	1011	As ADD, but result is not written
ORR	1100	Operand1 OR operand2
MOV	1101	Operand2 (operand1 is ignored)
BIC	1110	Operand1 AND NOT operand2 (Bit clear)
MVN	1111	NOT operand2 (operand1 is ignored)

The arithmetic operations (SUB, RSB, ADD, ADC, SBC, RSC, CMP, CMN) treat each operand as a 32-bit integer (either unsigned or 2's complement signed, the two are equivalent). When the S bit is set (and Rd is not R15), the V flag in the CPSR will be set if an overflow occurs in bit 31 of the result; this may be ignored if the operands were considered unsigned, but there will be a warning of a possible error if the operands were 2's complement signed. The C flag will be set to the carry out of bit 31 of the ALU, the Z flag will be set if and only if the result was zero, and the N flag will be set to the value of bit 31 of the result (indicating a negative result if the operands are considered to be 2's complement signed).

SHIFTS

When the second operand is specified to be a shifted register, the operation of the barrel shifter is controlled by the Shift field in the instruction. This field indicates the type of shift to be performed (logical left or right, arithmetic right or rotate right). The amount by which the register should be shifted may be contained in an immediate field in the instruction, or in the bottom byte of another register (other than R15). The encoding for the different shift types is shown in Figure 3-9.



[6:5] Shift type
00 = logical left   01 = logical right
10 = arithmetic right   11 = rotate right

[11:7] Shift amount
5 bit unsigned integer

[6:5] Shift type
00 = logical left   01 = logical right
10 = arithmetic right   11 = rotate right

[11:8] Shift register
Shift amount specified in bottom-byte of Rs

Figure 3-9. ARM Shift Operations

Instruction specified shift amount

When the shift amount is specified in the instruction, it is contained in a 5-bit field which may take any value from 0 to 31. A logical shift left (LSL) takes the contents of Rm and moves each bit by the specified amount to a more significant position. The least significant bits of the result are filled with zeros, and the high bits of Rm which do not map into the result are discarded. The least significant bit discarded becomes the shifter carry output which may be latched into the C bit of the CPSR when the ALU operation is in the logical class (see above). For an example, the effect of LSL #5 is shown in Figure 3-10.

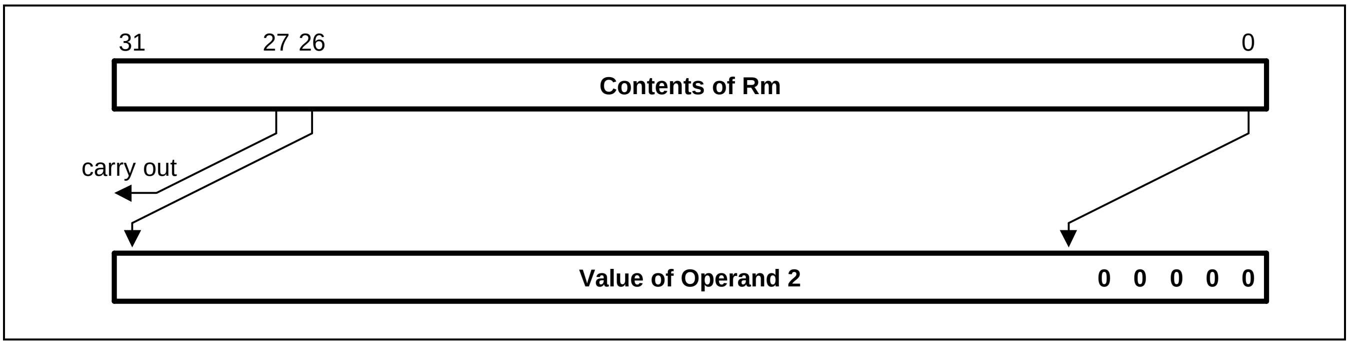


Figure 3-10. Logical Shift Left

NOTE

LSL #0 is a special case, where the shifter carry out is the old value of the CPSR C flag. The contents of Rm are used directly as the second operand. A logical shift right (LSR) is similar to this except that the contents of Rm are moved to less significant positions in the result. LSR #5 has the effect shown in Figure 3-11.

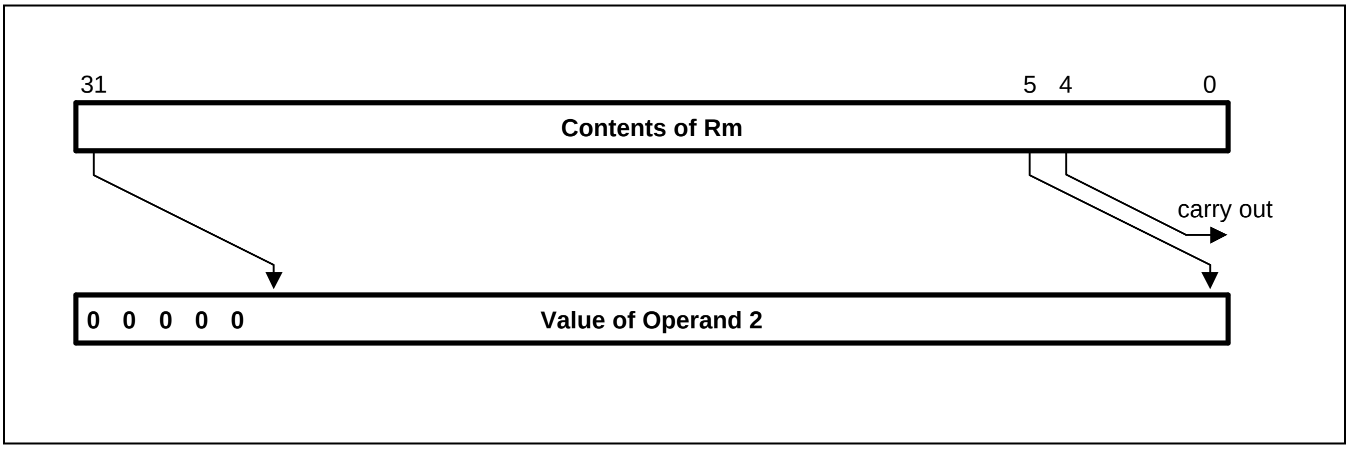


Figure 3-11. Logical Shift Right

The form of the shift field expected to correspond to LSR #0 is used to encode LSR #32, which has a zero result with bit 31 of Rm as the carry output. Logical shift right zero is redundant as it is the same as logical shift left zero, so the assembler will convert LSR #0 (and ASR #0 and ROR #0) into LSL #0, and allow LSR #32 to be specified.

An arithmetic shift right (ASR) is similar to logical shift right, except that the high bits are filled with bit 31 of Rm instead of zeros. This contain the sign in 2's complement notation. For an example, ASR #5 is shown in Figure 3-12.

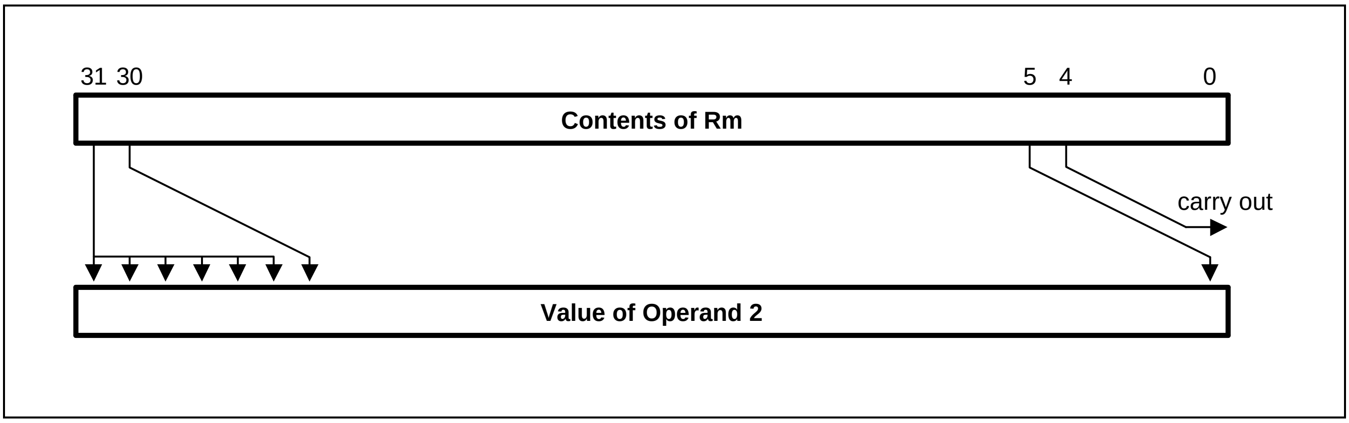


Figure 3-12. Arithmetic Shift Right

The form of the shift field expected to give ASR #0 is used to encode ASR #32. Bit 31 of Rm is again used as the carry output, and each bit of operand 2 is also equal to bit 31 of Rm. The result is therefore all ones or all zeros, according to the value of bit 31 of Rm.

Rotate right (ROR) operations reuse the bits which "overshoot" in a logical shift right operation by reintroducing them at the high end of the result, in place of the zeros used to fill the high end in logical right operations. For an example, ROR #5 is shown in Figure 3-13.

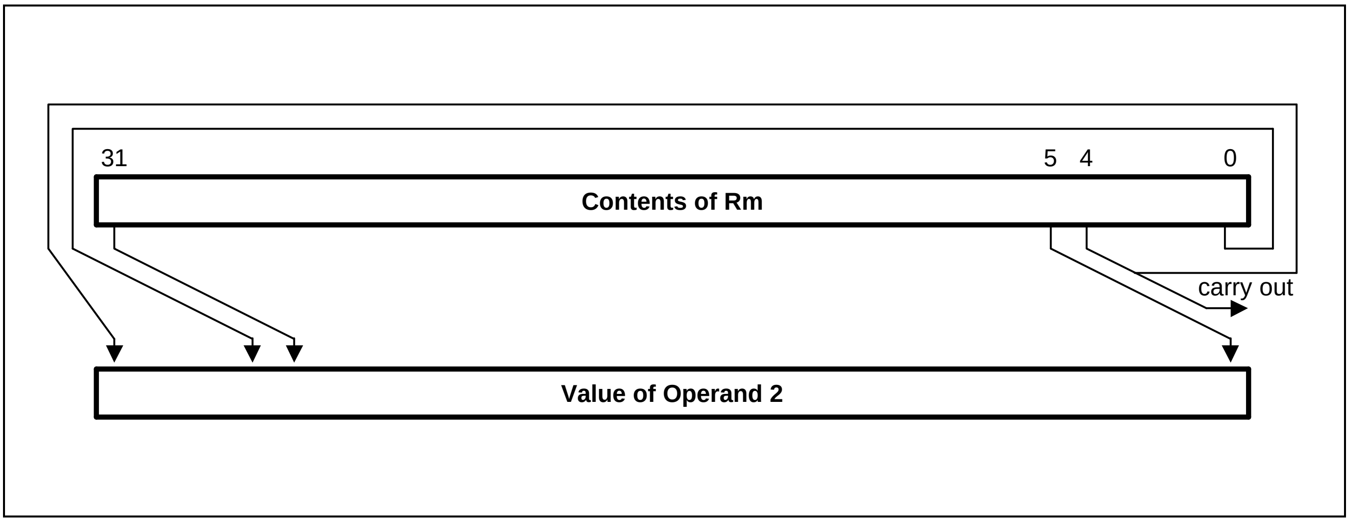


Figure 3-13. Rotate Right

The form of the shift field expected to give ROR #0 is used to encode a special function of the barrel shifter, rotate right extended (RRX). It enables rotating right by one bit position of the 33-bit quantity formed by appending the CPSR C flag to the most significant end of the contents of Rm as shown in Figure 3-14.

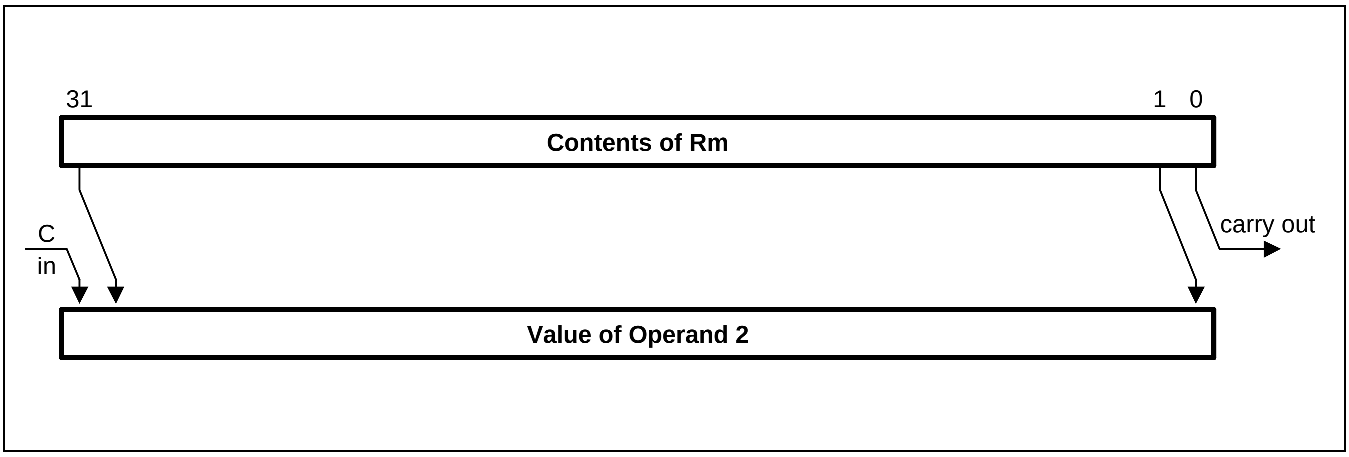


Figure 3-14. Rotate Right Extended

Register specified shift amount

Only the least significant byte of the contents of Rs is used to determine the shift amount. Rs can be any general register other than R15.

If this byte is zero, the unchanged contents of Rm will be used as the second operand, and the old value of the CPSR C flag will be passed on as the shifter carry output.

If the byte has a value between 1 and 31, the shifted result will exactly match that of an instruction specified shift with the same value and shift operation.

If the value in the byte is 32 or more, the result will be a logical extension of the shift described above:

1. LSL by 32 has result zero, carry out equal to bit 0 of Rm.
2. LSL by more than 32 has result zero, carry out zero.
3. LSR by 32 has result zero, carry out equal to bit 31 of Rm.
4. LSR by more than 32 has result zero, carry out zero.
5. ASR by 32 or more has result filled with and carry out equal to bit 31 of Rm.
6. ROR by 32 has result equal to Rm, carry out equal to bit 31 of Rm.
7. ROR by n where n is greater than 32 will give the same result and carry out as ROR by n-32; therefore repeatedly subtract 32 from n until the amount is in the range 1 to 32 and see above.

NOTE

The zero in bit 7 of an instruction with a register controlled shift is compulsory; a one in this bit will cause the instruction to be a multiply or undefined instruction.

IMMEDIATE OPERAND ROTATES

The immediate operand rotate field is a 4-bit unsigned integer which specifies a shift operation on the 8-bit immediate value. This value is zero extended to 32 bits, subject to a rotate right by twice the value in the rotate field. This enables many common constants to be generated, for example, all powers of 2.

WRITING TO R15

When Rd is a register other than R15, the condition code flags in the CPSR may be updated from the ALU flags as described above.

When Rd is R15 and the S flag in the instruction is not set the result of the operation is placed in R15 and the CPSR is unaffected.

When Rd is R15 and the S flag is set the result of the operation is placed in R15 and the SPSR corresponding to the current mode is moved to the CPSR. This allows state changes which atomically restore both PC and CPSR. This form of instruction should not be used in User mode.

USING R15 AS AN OPERAND

If R15 (the PC) is used as an operand in a data processing instruction, the register is used directly.

The PC value will be the address of the instruction, plus 8 or 12 bytes due to instruction prefetching. If the shift amount is specified in the instruction, the PC will be 8 bytes ahead. If a register is used to specify the shift amount the PC will be 12 bytes ahead.

TEQ, TST, CMP AND CMN OPCODES

NOTE

> TEQ, TST, CMP and CMN do not write the result of their operation but do set flags in the CPSR. An assembler should always set the S flag for these instructions even if this is not specified in the mnemonic.

The TEQP form of the TEQ instruction used in earlier ARM processors must not be used: the PSR transfer operations should be used instead.

The TEQP in the ARM9TDMI moves SPSR_<mode> to the CPSR if the processor is in a privileged mode and does nothing in User mode.

INSTRUCTION CYCLE TIMES

Data Processing instructions vary in the number of incremental cycles as follows:

Table 3-7. Incremental Cycle Times

Processing Type	Cycles
Normal data processing	1S
Data processing with register specified shift	1S + 1I
Data processing with PC written	2S + 1N
Data processing with register specified shift and PC written	2S + 1N +1I

NOTE: S, N, and I mean sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively.

ASSEMBLER SYNTAX

- MOV,MVN (single operand instructions).
  <opcode>{cond}{S} Rd,<Op2>
- CMP,CMN,TEQ,TST (instructions which do not produce a result).
  <opcode>{cond} Rn,<Op2>
- AND,EOR,SUB,RSB,ADD,ADC,SBC,RSC,ORR,BIC
  <opcode>{cond}{S} Rd,Rn,<Op2>

where:

<Op2> Rm{,<shift>} or,<#expression>

{cond} A two-character condition mnemonic. See Table 3-5.

{S} Set condition codes if S is present (implied for CMP, CMN, TEQ, TST).

Rd, Rn and Rm Expressions evaluating to a register number.

<#expression> If this is used, the assembler will attempt to generate a shifted immediate 8-bit field to match the expression. If this is impossible, it will give an error.

<shift> <Shiftname> <register> or <shiftname> #expression, or RRX (rotate right one bit with extend).

<shiftname>s ASL, LSL, LSR, ASR, ROR. (ASL is a synonym for LSL, they assemble to the same code.)

Examples

```
ADDEQ     R2,R4,R5              ;  If the Z flag is set, make R2:=R4+R5
TEQS      R4,#3                 ;  Test R4 for equality with 3.
                                ;  (The S is in fact redundant as the
                                ;  assembler inserts it automatically.)
SUB       R4,R5,R7,LSR R2       ;  Logical right shift R7 by the number in
                                ;  the bottom byte of R2, subtract result
                                ;  from R5, and put the answer into R4.
MOV       PC,R14                ;  Return from subroutine.
MOVS      PC,R14                ;  Return from exception and restore CPSR
                                ;  from SPSR_mode.
```

PSR TRANSFER (MRS, MSR)

The instruction is executed only if the condition is true. The various conditions are defined in Table 3-5.

The MRS and MSR instructions are formed from a subset of the Data Processing operations and are implemented using the TEQ, TST, CMN, and CMP instructions without the S flag set. The encoding is shown in Figure 3-15.

These instructions allow access to the CPSR and SPSR registers. The MRS instruction allows the contents of the CPSR or SPSR_<mode> to be moved to a general register. The MSR instruction allows the contents of a general register to be moved to the CPSR or SPSR_<mode> register.

The MSR instruction also allows an immediate value or register contents to be transferred to the condition code flags (N,Z,C and V) of CPSR or SPSR_<mode> without affecting the control bits. In this case, the top four bits of the specified register contents or 32-bit immediate values are written to the top four bits of the relevant PSR.

OPERAND RESTRICTIONS

- In user mode, the control bits of the CPSR are protected from a change, so only the condition code flags of the CPSR can be changed. In other (privileged) modes, the entire CPSR can be changed.
- Note that the software must never change the state of the T bit in the CPSR, otherwise the processor would enter an unpredictable state.
- The SPSR register which is accessed depends on the mode at the time of execution. For example, only SPSR_fiq is accessible when the processor is in FIQ mode.
- You must not specify R15 as the source or destination register.
- Also, do not attempt to access an SPSR in User mode, since no such register exists.

MRS (transfer PSR contents to a register)

31 28	27 23	22	21 16	15 12	11 0
Cond	00010	Ps	001111	Rd	000000000000

[15:21] Destination Register

[19:16] Source PSR
0 = CPSR 1 = SPSR_<current mode>

[31:28] Condition Field

MRS (transfer register contents to PSR)

31 28	27 23	22	21 12	11 4	3 0
Cond	00010	Pd	101001111	00000000	Rm

[3:0] Source Register

[22] Destination PSR
0 = CPSR 1 = SPSR_<current mode>

[31:28] Condition Field

MRS (transfer register contents or immediate value to PSR flag bits only)

31 28	27 26	25	24 23	22	21 12	11 0
Cond	00	I	10	Pd	101001111	Source operand

[22] Destination PSR
0 = CPSR 1 = SPSR_<current mode>

[25] Immediate Operand
0 = Source operand is a register
1 = SPSR_<current mode>

[11:0] Source Operand

11 4	3 0
00000000	Rm

[3:0] Source Register
[11:4] Source operand is an immediate value

11 8	7 0
Rotate	Imm

[7:0] Unsigned 8 bit immediate value
[11:8] Shift applied to Imm

[31:28] Condition Field

Figure 3-15. PSR Transfer

RESERVED BITS

Only twelve bits of the PSR are defined in ARM9TDMI (N,Z,C,V,I,F, T & M[4:0]); the remaining bits are reserved for use in future versions of the processor. Refer to Figure 2-6 in Cahpter 2 *Programmer's model* for a full description of the PSR bits.

To ensure the maximum compatibility between ARM9TDMI programs and future processors, the following rules should be observed:

- The reserved bits should be preserved when changing the value in a PSR.
- Programs should not rely on specific values from the reserved bits when checking the PSR status, since they may read as one or zero in future processors.

A read-modify-write strategy should therefore be used when altering the control bits of any PSR register; this involves transferring the appropriate PSR register to a general register using the MRS instruction, changing only the relevant bits and then transferring the modified value back to the PSR register using the MSR instruction.

Examples

The following sequence performs a mode change:

```
MRS     R0,CPSR              ;  Take a copy of the CPSR.
BIC     R0,R0,#0x1F          ;  Clear the mode bits.
ORR     R0,R0,#new_mode      ;  Select new mode
MSR     CPSR,R0              ;  Write back the modified CPSR.
```

When the aim is simply to change the condition code flags in a PSR, a value can be written directly to the flag bits without disturbing the control bits. The following instruction sets the N,Z,C and V flags:

```
MSR     CPSR_flg,#0xF0000000 ;  Set all the flags regardless of their previous state
                             ;  (does not affect any control bits).
```

No attempt should be made to write an 8-bit immediate value into the whole PSR since such an operation cannot preserve the reserved bits.

INSTRUCTION CYCLE TIMES

PSR transfers take 1S incremental cycles, where S is defined as Sequential (S-cycle).

ASSEMBLY SYNTAX

- MRS - transfer PSR contents to a register
  MRS{cond} Rd,<psr>
- MSR - transfer register contents to PSR
  MSR{cond} <psr>,Rm
- MSR - transfer register contents to PSR flag bits only
  MSR{cond} <psrf>,Rm

The most significant four bits of the register contents are written to the N,Z,C & V flags, respectively.

- MSR - transfer immediate value to PSR flag bits only
  MSR{cond} <psrf>,<#expression>

The expression should symbolize a 32-bit value of which the most significant four bits are written to the N,Z,C and V flags respectively.

Key:

{cond} Two-character condition mnemonic. See Table 3-5..

Rd and Rm Expressions evaluating to a register number other than R15

<psr> CPSR, CPSR_all, SPSR or SPSR_all. (CPSR and CPSR_all are synonyms ous with SPSR and SPSR_all)

<psrf> CPSR_flg or SPSR_flg

<#expression> Where this is used, the assembler will attempt to generate a shifted immediate 8-bit
field

to match the expression. If this is impossible, it will give an error.

Examples

In User mode, the instructions behave as follows:

```
MSR     CPSR_all,Rm             ;  CPSR[31:28] <- Rm[31:28]
MSR     CPSR_flg,Rm             ;  CPSR[31:28] <- Rm[31:28]
MSR     CPSR_flg,#0xA0000000    ;  CPSR[31:28] <- 0xA (set N,C; clear Z,V)
MRS     Rd,CPSR                 ;  Rd[31:0] <- CPSR[31:0]
```

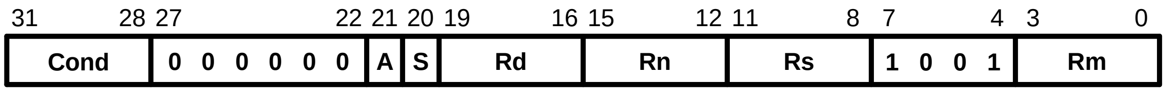
In privileged modes, the instructions behave as follows:

```
MSR     CPSR_all,Rm             ;  CPSR[31:0]  <- Rm[31:0]
MSR     CPSR_flg,Rm             ;  CPSR[31:28] <- Rm[31:28]
MSR     CPSR_flg,#0x50000000    ;  CPSR[31:28] <- 0x5 (set Z,V; clear N,C)
MSR     SPSR_all,Rm             ;  SPSR_<mode>[31:0]<- Rm[31:0]
MSR     SPSR_flg,Rm             ;  SPSR_<mode>[31:28] <- Rm[31:28]
MSR     SPSR_flg,#0xC0000000    ;  SPSR_<mode>[31:28] <- 0xC (set N,Z; clear C,V)
MRS     Rd,SPSR                 ;  Rd[31:0] <- SPSR_<mode>[31:0]
```

MULTIPLY AND MULTIPLY-ACCUMULATE (MUL, MLA)

The instruction is executed only if the condition is true. The various conditions are defined in Table 3-5. The instruction encoding is shown in Figure 3-16.

The multiply and multiply-accumulate instructions use an 8-bit Booth's algorithm to perform integer multiplication.



[15:12][11:8][3:0] Operand Registers
[19:16] Destination Register

[20] Set Condition Code
0 = Do not after condition codes
1 = Set condition codes

[21] Accumulate
0 = Multiply only
1 = Multiply and accumulate

[31:28] Condition Field

Figure 3-16. Multiply Instructions

The multiply form of the instruction gives Rd:=Rm*Rs. Rn is ignored, and should be set to zero for compatibility with possible future upgrades to the instruction set.The multiply-accumulate form gives Rd:=Rm*Rs+Rn, which can save an explicit ADD instruction in some circumstances. Both forms of the instruction work on operands which may be considered as signed (2's complement) or unsigned integers.

The results of a signed multiply and of an unsigned multiply of 32-bit operands differ only in the upper 32 bits the low 32 bits of the signed and unsigned results are identical. As these instructions only produce the low 32 bits of a multiply, they can be used for both signed and unsigned multiplies.

For example, consider the multiplication of the operands:

Operand A	Operand B	Result
0xFFFFFFF6	0x0000001	0xFFFFFF38

If the Operands Are Interpreted as Signed

Operand A has the value -10, operand B has the value 20, and the result is -200 which is correctly represented as 0xFFFFFF38.

If the Operands Are Interpreted as Unsigned

Operand A has the value 4294967286, operand B has the value 20 and the result is 85899345720, which is represented as 0x13FFFFFF38, so the least significant 32 bits are 0xFFFFFF38.

Operand Restrictions

The destination register Rd must not be the same as the operand register Rm. R15 must not be used as an operand or as the destination register.

All other register combinations will give correct results, and Rd, Rn and Rs may use the same register when required.

CPSR FLAGS

Setting the CPSR flags is optional, and is controlled by the S bit in the instruction. The N (Negative) and Z (Zero) flags are set correctly on the result (N is made equal to bit 31 of the result, and Z is set if and only if the result is zero). The C (Carry) flag is set to a meaningless value and the V (oVerflow) flag is unaffected.

INSTRUCTION CYCLE TIMES

MUL takes 1S + m**I** and MLA 1S + (m+1)**I** cycles to execute, where S and **I** are defined as sequential (S-cycle) and internal (**I**-cycle), respectively.

m The number of 8-bit multiplier array cycles is required to complete the multiply, which is controlled by the value of the multiplier operand specified by Rs. Available values are as follows:

1 If bits [32:8] of the multiplier operand are all zero or all one.

2 If bits [32:16] of the multiplier operand are all zero or all one.

3 If bits [32:24] of the multiplier operand are all zero or all one.

4 In all other cases.

ASSEMBLER SYNTAX

```
MUL{cond}{S} Rd,Rm,Rs
     MLA{cond}{S} Rd,Rm,Rs,Rn
```

{cond} Two-character condition mnemonic. See Table 3-5..

{S} Set condition codes if S is present

Rd, Rm, Rs and Rn Expressions evaluating to a register number other than R15.

Examples

```
MUL       R1,R2,R3          ;  R1: = R2*R3
MLAEQS    R1,R2,R3,R4       ;  Conditionally R1: = R2*R3+R4, Setting condition codes.
```

MULTIPLY LONG AND MULTIPLY-ACCUMULATE LONG (MULL, MLAL)

The instruction is executed only if the condition is true. The various conditions are defined in Table 3-5. The instruction encoding is shown in Figure 3-17.

The multiply long instructions perform integer multiplication on two 32-bit operands and produce 64-bit results. Signed and unsigned multiplication each with optional accumulate give rise to four variations.

31 28	27 23	22	21	20	19 16	15 12	11 8	7 4	3 0
Cond	0 0 0 0 1	U	A	S	RdHi	RdLo	Rs	1 0 0 1	Rm

[11:8][3:0] Operand Registers
[19:16][15:12] Source Destination Registers

[20] Set Condition Code
0 = Do not alter condition codes
1 = Set condition codes

[21] Accumulate
0 = Multiply only
1 = Multiply and accumulate

[22] Unsigned
0 = Unsigned
1 = Signed

[31:28] Condition Field

Figure 3-17. Multiply Long Instructions

The multiply forms (UMULL and SMULL) take two 32-bit numbers and multiply them to produce a 64-bit result of the form RdHi,RdLo := Rm * Rs. The lower 32 bits of the 64-bit result are written to RdLo, the upper 32 bits of the result are written to RdHi.

The multiply-accumulate forms (UMLAL and SMLAL) take two 32-bit numbers, multiply them and add a 64-bit number to produce a 64-bit result of the form RdHi,RdLo := Rm * Rs + RdHi,RdLo. The lower 32 bits of the 64-bit number to add is read from RdLo. The upper 32 bits of the 64-bit number to add is read from RdHi. The lower 32 bits of the 64-bit result are written to RdLo. The upper 32 bits of the 64-bit result are written to RdHi.

The UMULL and UMLAL instructions treat all of their operands as unsigned binary numbers and write an unsigned 64-bit result. The SMULL and SMLAL instructions treat all of their operands as 2's-complement signed numbers and write a 2's-complement signed 64-bit result.

OPERAND RESTRICTIONS

- R15 must not be used as an operand or as a destination register.
- RdHi, RdLo, and Rm must all specify different registers.

CPSR FLAGS

Setting the CPSR flags is optional, and is controlled by the S bit in the instruction. The N and Z flags are set correctly on the result (N is equal to bit 63 of the result, Z is set if and only if all 64 bits of the result are zero). Both the C and V flags are set to meaningless values.

INSTRUCTION CYCLE TIMES

MULL takes 1S + (m+1)I and MLAL 1S + (m+2)I cycles to execute, where *m* is the number of 8-bit multiplier array cycles required to complete the multiply, which is controlled by the value of the multiplier operand specified by Rs.

Available values are as follows:

For Signed INSTRUCTIONS SMULL, SMLAL:

- If bits [31:8] of the multiplier operand are all zero or all one.
- If bits [31:16] of the multiplier operand are all zero or all one.
- If bits [31:24] of the multiplier operand are all zero or all one.
- In all other cases.

For Unsigned Instructions UMULL, UMLAL:

- If bits [31:8] of the multiplier operand are all zero.
- If bits [31:16] of the multiplier operand are all zero.
- If bits [31:24] of the multiplier operand are all zero.
- In all other cases.

S and I are defined as sequential (S-cycle) and internal (I-cycle), respectively.

ASSEMBLER SYNTAX

Table 3-8. Assembler Syntax Descriptions

Mnemonic	Description	Purpose
UMULL{cond}{S} RdLo,RdHi,Rm,Rs	Unsigned Multiply Long	32 x 32 = 64
UMLAL{cond}{S} RdLo,RdHi,Rm,Rs	Unsigned Multiply & Accumulate Long	32 x 32 + 64 = 64
SMULL{cond}{S} RdLo,RdHi,Rm,Rs	Signed Multiply Long	32 x 32 = 64
SMLAL{cond}{S} RdLo,RdHi,Rm,Rs	Signed Multiply & Accumulate Long	32 x 32 + 64 = 64

where:

{cond}	Two-character condition mnemonic. See Table 3-5.
{S}	Set condition codes if S is present
RdLo, RdHi, Rm, Rs	Expressions evaluating to a register number other than R15.

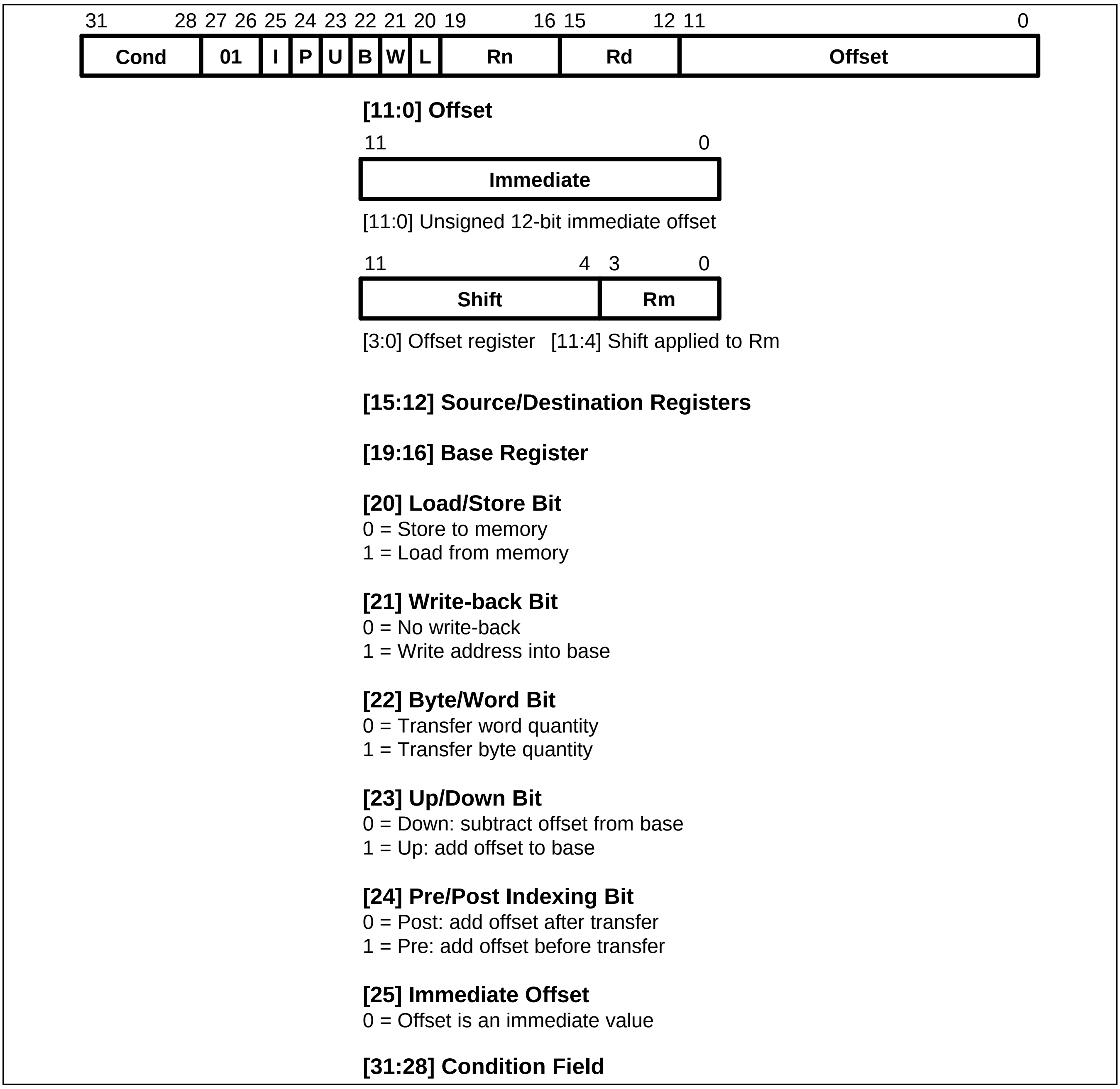
Examples

```
        UMULL     R1,R4,R2,R3          ;  R4,R1: = R2*R3
        UMLALS    R1,R5,R2,R3          ;  R5,R1: = R2*R3+R5,R1 also setting condition codes
```

SINGLE DATA TRANSFER (LDR, STR)

The instruction is executed only if the condition is true. The various conditions are defined in Table 3-5. The instruction encoding is shown in Figure 3-18. The single data transfer instructions are used to load or store single bytes or words of data. The memory address used in the transfer is calculated by adding an offset to or subtracting an offset from a base register.

The result of this calculation may be written back into the base register if auto-indexing is required.



[11:0] Offset

[11:0] Unsigned 12-bit immediate offset

[3:0] Offset register [11:4] Shift applied to Rm

[15:12] Source/Destination Registers

[19:16] Base Register

[20] Load/Store Bit
0 = Store to memory
1 = Load from memory

[21] Write-back Bit
0 = No write-back
1 = Write address into base

[22] Byte/Word Bit
0 = Transfer word quantity
1 = Transfer byte quantity

[23] Up/Down Bit
0 = Down: subtract offset from base
1 = Up: add offset to base

[24] Pre/Post Indexing Bit
0 = Post: add offset after transfer
1 = Pre: add offset before transfer

[25] Immediate Offset
0 = Offset is an immediate value

[31:28] Condition Field

Figure 3-18. Single Data Transfer Instructions

OFFSETS AND AUTO-INDEXING

The offset from the base may be either a 12-bit unsigned binary immediate value in the instruction, or a second register (possibly shifted in some way). The offset may be added to (U=1) or subtracted from (U=0) the base register Rn. The offset modification may be performed either before (pre-indexed, P=1) or after (post-indexed, P=0) the base is used as the transfer address.

The W bit gives optional auto increment and decrement addressing modes. The modified base value may be written back into the base (W=1), or the old base value may be kept (W=0). In the case of post-indexed addressing, the write back bit is redundant and is always set to zero, since the old base value can be retained by setting the offset to zero. Therefore post-indexed data transfers always write back the modified base. The only use of the W bit in a post-indexed data transfer is in privileged mode code, where setting the W bit forces non-privileged mode for the transfer, allowing the operating system to generate a user address in a system where the memory management hardware makes suitable use of this hardware.

SHIFTED REGISTER OFFSET

The 8 shift control bits are described in the data processing instructions section. However, the register specified shift amounts are not available in this instruction class. See Figure 3-6.

BYTES AND WORDS

This instruction class may be used to transfer a byte (B=1) or a word (B=0) between an ARM9TDMI register and memory.

The action of LDR(B) and STR(B) instructions is influenced by the **BIGEND** control signal of ARM9TDMI core. The two possible configurations are described below.

Little-Endian Configuration

A byte load (LDRB) expects the data on data bus inputs 7 through 0 if the supplied address is on a word boundary, on data bus inputs 15 through 8 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bits of the destination register, and the remaining bits of the register are filled with zeros. Please see Figure 2-2 in Chapter 2 *Programmer's model*.

A byte store (STRB) repeats the bottom 8 bits of the source register four times across data bus outputs 31 through 0. The external memory system should activate the appropriate byte subsystem to store the data.

A word load (LDR) will normally use a word aligned address. However, an address offset from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 0 to 7. This means that half-words accessed at offsets 0 and 2 from the word boundary will be correctly loaded into bits 0 through 15 of the register. Two shift operations are then required to clear or to sign extend the upper 16 bits.

A word store (STR) should generate a word aligned address. The word presented to the data bus is not affected if the address is not word aligned. That is, bit 31 of the register being stored always appears on data bus output 31.

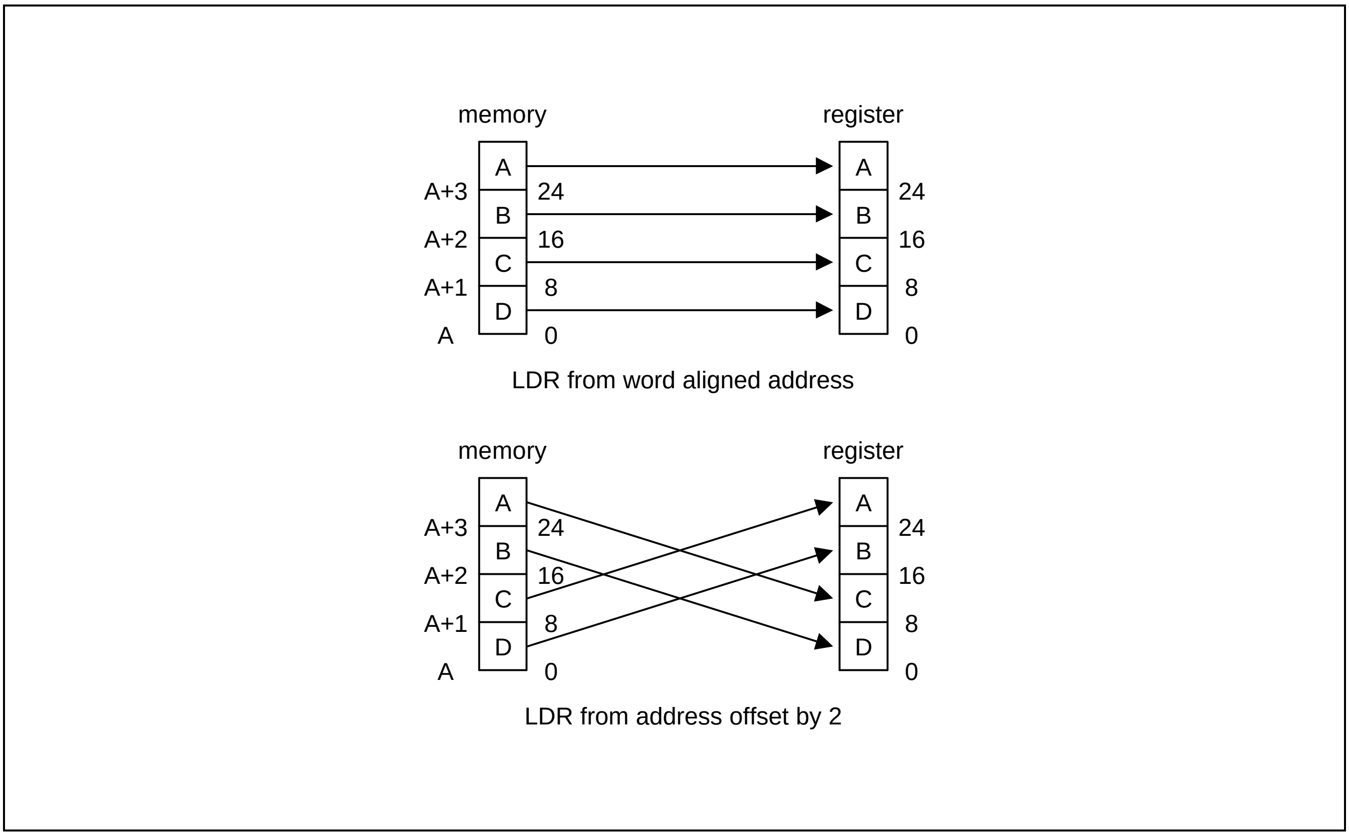


Figure 3-19. Little-Endian Offset Addressing

Big-Endian Configuration

A byte load (LDRB) expects the data on data bus inputs 31 through 24 if the supplied address is on a word boundary, on data bus inputs 23 through 16 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bits of the destination register and the remaining bits of the register are filled with zeros. Please see Figure 2-1 in Chapter 2 *Programmer's model*.

A byte store (STRB) repeats the bottom 8 bits of the source register four times across data bus outputs 31 through 0. The external memory system should activate the appropriate byte subsystem to store the data.

A word load (LDR) should generate a word aligned address. An address offset of 0 or 2 from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 31 through 24. This means that half-words accessed at these offsets will be correctly loaded into bits 16 through 31 of the register. A shift operation is then required to move (and optionally sign extend) the data into the bottom 16 bits. An address offset of 1 or 3 from a word boundary will cause the data to be rotated into the register so that the addressed byte occupies bits 15 through 8.

A word store (STR) should generate a word aligned address. The word presented to the data bus is not affected if the address is not word aligned. That is, bit 31 of the register being stored always appears on data bus output 31.

USE OF R15

Write-back must not be specified if R15 is specified as the base register (Rn). When using R15 as the base register you must remember that it contains an address 8 bytes on from the address of the current instruction.

R15 must not be specified as the register offset (Rm).

When R15 is the source register (Rd) of a register store (STR) instruction, the stored value will be address of the instruction plus 12.

RESTRICTION ON THE USE OF BASE REGISTER

When configured for late aborts, the following example code is difficult to unwind as the base register, Rn, gets updated before the abort handler starts. Sometimes it may be impossible to calculate the initial value.

After an abort, the following example code is difficult to unwind as the base register, Rn, gets updated before the abort handler starts. Sometimes it may be impossible to calculate the initial value.

Example

```
            LDR          R0,[R1],R1
```

Therefore a post-indexed LDR or STR where Rm is the same register as Rn should not be used.

DATA ABORTS

A transfer to or from a legal address may cause problems for a memory management system. For instance, in a system which uses virtual memory, the required data may be absent from main memory. The memory manager can signal a problem by taking the processor ABORT input HIGH whereupon the Data Abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continued.

INSTRUCTION CYCLE TIMES

Normal LDR instructions take 1S + 1N + 1I and LDR PC take 2S + 2N +1I incremental cycles, where S,N and I are defined as sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively. STR instructions take 2N incremental cycles to execute.

ASSEMBLER SYNTAX

where:

LDR Load from memory into a register

STR Store from a register into memory

{cond} Two-character condition mnemonic. See Table 3-5.

{B} If B is present, then byte transfer, otherwise word transfer

{T} If T is present, the W bit will be set in a post-indexed instruction, forcing non-privileged mode for the transfer cycle. T is not allowed when a pre-indexed addressing mode is specified or implied.

Rd An expression evaluating to a valid register number.

Rn and Rm Expressions evaluating to a register number. If Rn is R15 then the assembler will subtract 8 from the offset value to allow for ARM7TDMI pipelining. In this case, base write-back should not be specified.

<Address>can be:

1 An expression which generates an address:
The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated.

2 A pre-indexed addressing specification:
[Rn] offset of zero
[Rn,<#expression>]{!} offset of <expression> bytes
[Rn,{+/-}Rm{,<shift>}]{!} offset of +/- contents of index register, shifted by <shift>

3 A post-indexed addressing specification:
[Rn],<#expression> offset of <expression> bytes
[Rn],{+/-}Rm{,<shift>} offset of +/- contents of index register, shifted as by <shift>.

<shift> General shift operation (see data processing instructions) but you cannot specify the shift amount by a register.

{!} Writes back the base register (set the W bit) if ! is present.

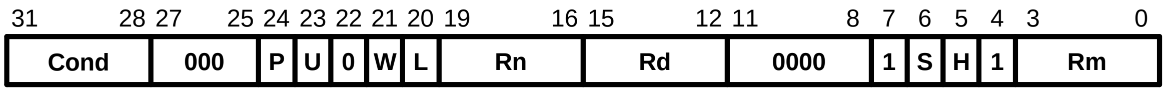
Examples

```
        STR         R1,[R2,R4]!             ;   Store R1 at R2+R4 (both of which are registers)
                                            ;   and write back address to R2.
        STR         R1,[R2],R4              ;   Store R1 at R2 and write back R2+R4 to R2.
        LDR         R1,[R2,#16]             ;   Load R1 from contents of R2+16, but don't write back.
        LDR         R1,[R2,R3,LSL#2]        ;   Load R1 from contents of R2+R3*4.
        LDREQB      R1,[R6,#5]              ;   Conditionally load byte at R6+5 into
                                            ;   R1 bits 0 to 7, filling bits 8 to 31 with zeros.
        STR         R1,PLACE                ;   Generate PC relative offset to address PLACE.
        PLACE
```

HALFWORD AND SIGNED DATA TRANSFER (LDRH/STRH/LDRSB/LDRSH)

The instruction is executed only if the condition is true. The various conditions are defined in Table 3-5. The instruction encoding is shown in Figure 3-20.

These instructions are used to load or store half-words of data and also load sign-extended bytes or half-words of data. The memory address used in the transfer is calculated by adding an offset to or subtracting an offset from a base register. The result of this calculation may be written back into the base register if auto-indexing is required.



[3:0] Offset Register

[6][5] S H
0 0 = SWP instruction
0 1 = Unsigned halfword
1 1 = Signed byte
1 1 = Signed halfword

[15:12] Source/Destination Register

[19:16] Base Register

[20] Load/Store
0 = Store to memory
1 = Load from memory

[21] Write-back
0 = No write-back
1 = Write address into base

[23] Up/Down
0 = Down: subtract offset from base
1 = Up: add offset to base

[24] Pre/Post Indexing
0 = Post: add/subtract offset after transfer
1 = Pre: add/subtract offset bofore transfer

[31:28] Condition Field

Figure 3-20. Halfword and Signed Data Transfer with Register Offset

31 28	27 25	24	23	22	21	20	19 16	15 12	11 8	7	6	5	4	3 0
Cond	000	P	U	1	W	L	Rn	Rd	Offset	1	S	H	1	Offset

[3:0] Immediate Offset (Low Nibble)

[6][5] S H
0 0 = SWP instruction
0 1 = Unsigned halfword
1 1 = Signed byte
1 1 = Signed halfword

[11:8] Immediate Offset (High Nibble)

[15:12] Source/Destination Register

[19:16] Base Register

[20] Load/Store
0 = Store to memory
1 = Load from memory

[21] Write-back
0 = No write-back
1 = Write address into base

[23] Up/Down
0 = Down: subtract offset from base
1 = Up: add offset to base

[24] Pre/Post Indexing
0 = Post: add/subtract offset after transfer
1 = Pre: add/subtract offset bofore transfer

[31:28] Condition Field

Figure 3-21. Halfword and Signed Data Transfer with Immediate Offset and Auto-Indexing

OFFSETS AND AUTO-INDEXING

The offset from the base may be either an 8-bit unsigned binary immediate value in the instruction, or a second register. The 8-bit offset is formed by concatenating bits 11 to 8 and bits 3 to 0 of the instruction word, so that bit 11 becomes the MSB and bit 0 becomes the LSB. The offset may be added to (U=1) or subtracted from (U=0) the base register Rn. The offset modification may be performed either before (pre-indexed, P=1) or after (post-indexed, P=0) the base register is used as the transfer address.

The W bit gives optional auto-increment and decrement addressing modes. The modified base value may be written back into the base (W=1), or the old base may be kept (W=0). In the case of post-indexed addressing, the write back bit is redundant and is always set to zero, since the old base value can be retained if necessary by setting the offset to zero. Therefore post-indexed data transfers always write back the modified base. The write-back bit should not be set high (W=1) when post-indexed addressing is selected.

HALFWORD LOAD AND STORES

Setting S=0 and H=1 may be used to transfer unsigned Half-words between an ARM9TDMI register and memory.

The action of LDRH and STRH instructions is influenced by the BIGEND control signal. The two possible configurations are described in the section below.

SIGNED BYTE AND HALFWORD LOADS

The S bit controls the loading of sign-extended data. When S=1, the H bit selects between Bytes (H=0) and Half-words (H=1). The L bit should not be set low (Store) when Signed (S=1) operations have been selected.

The LDRSB instruction loads the selected Byte into bits 7 to 0 of the destination register and bits 31 to 8 of the destination register are set to the value of bit 7, the sign bit.

The LDRSH instruction loads the selected Half-word into bits 15 to 0 of the destination register and bits 31 to 16 of the destination register are set to the value of bit 15, the sign bit.

The action of the LDRSB and LDRSH instructions is influenced by the BIGEND control signal. The two possible configurations are described in the following section.

ENDIANNESS AND BYTE/HALFWORD SELECTION

Little-Endian Configuration

A signed byte load (LDRSB) expects data on data bus inputs 7 through to 0 if the supplied address is on a word boundary, on data bus inputs 15 through to 8 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bits of the destination register, and the remaining bits of the register are filled with the sign bit, bit 7 of the byte. Please see Figure 2-2 in Chapter 2 *Programmer's model*.

A halfword load (LDRSH or LDRH) expects data on data bus inputs 15 through to 0 if the supplied address is on a word boundary and on data bus inputs 31 through to 16 if it is a halfword boundary, (A[1]=1).The supplied address should always be on a halfword boundary. If bit 0 of the supplied address is HIGH, the ARM9TDMI will load an unpredictable value. The selected halfword is placed in the bottom 16 bits of the destination register. For unsigned half-words (LDRH), the top 16 bits of the register are filled with zeros and for signed half-words (LDRSH), the top 16 bits are filled with the sign bit, bit 15 of the halfword.

A halfword store (STRH) repeats the bottom 16 bits of the source register twice across the data bus outputs 31 through to 0. The external memory system should activate the appropriate halfword subsystem to store the data. Note that the address must be halfword aligned. If bit 0 of the address is HIGH, this will cause unpredictable behaviour.

Big-Endian Configuration

A signed byte load (LDRSB) expects data on data bus inputs 31 through to 24 if the supplied address is on a word boundary, on data bus inputs 23 through to 16 if it is a word address plus one byte, and so on. The selected byte is placed in the bottom 8 bits of the destination register, and the remaining bits of the register are filled with the sign bit, bit 7 of the byte. Please see Figure 2-1 in Chapter 2 *Programmer's model*.

A halfword load (LDRSH or LDRH) expects data on data bus inputs 31 through to 16 if the supplied address is on a word boundary and on data bus inputs 15 through to 0 if it is a halfword boundary, (A[1]=1). The supplied address should always be on a halfword boundary. If bit 0 of the supplied address is HIGH, the ARM9TDMI will load an unpredictable value. The selected halfword is placed in the bottom 16 bits of the destination register. For unsigned half-words (LDRH), the top 16 bits of the register are filled with zeros and for signed half-words (LDRSH), the top 16 bits are filled with the sign bit, bit 15 of the halfword.

A halfword store (STRH) repeats the bottom 16 bits of the source register twice across the data bus outputs 31 through to 0. The external memory system should activate the appropriate halfword subsystem to store the data. Note that the address must be halfword aligned. If bit 0 of the address is HIGH, this will cause unpredictable behaviour.

USE OF R15

Write-back should not be specified if R15 is specified as the base register (Rn). When using R15 as the base register, you must remember that it contains an address 8 bytes on from the address of the current instruction.

R15 should not be specified as the register offset (Rm).

When R15 is the source register (Rd) of a Half-word store (STRH) instruction, the stored address will be address of the instruction plus 12.

DATA ABORTS

A transfer to or from a legal address may cause problems for a memory management system. For instance, in a system which uses virtual memory, the required data may be absent from the main memory. The memory manager can signal a problem by taking the processor ABORT input HIGH whereupon the Data Abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continues.

INSTRUCTION CYCLE TIMES

Normal LDR(H,SH,SB) instructions take 1S + 1N + 1I. LDR(H,SH,SB) PC take 2S + 2N + 1I incremental cycles. S,N and I are defined as sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively. STRH instructions take 2N incremental cycles to execute.

ASSEMBLER SYNTAX

<LDR|STR>{cond}<H|SH|SB> Rd,<address>

LDR	Load from memory into a register
STR	Store from a register into memory
{cond}	Two-character condition mnemonic. See Table 3-5..
H	Transfer halfword quantity
SB	Load sign-extended byte (Only valid for LDR)
SH	Load sign-extended halfword (Only valid for LDR)
Rd	An expression evaluating to a valid register number.

<address> can be:

1 An expression which generates an address:
The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated.

2 A pre-indexed addressing specification:

[Rn]	offset of zero
[Rn,<#expression>]{!}	offset of <expression> bytes
[Rn,{+/-}Rm]{!}	offset of +/- contents of index register

3 A post-indexed addressing specification:

[Rn],<#expression>	offset of <expression> bytes
[Rn],{+/-}Rm	offset of +/- contents of index register.

4 Rn and Rm are expressions evaluating to a register number. If Rn is R15 then the assembler will subtract 8 from the offset value to allow for ARM9TDMI pipelining. In this case, base write-back should not be specified.

{!} Writes back the base register (set the W bit) if ! is present.

Examples

```
LDRH      R1,[R2,-R3]!                ;  Load R1 from the contents of the halfword address
                                      ;  contained in R2-R3 (both of which are registers)
                                      ;  and write back address to R2
STRH      R3,[R4,#14]                 ;  Store the halfword in R3 at R14+14 but don't write back.
LDRSB     R8,[R2],#-223               ;  Load R8 with the sign-extended contents of the byte
                                      ;  address contained in R2 and write back R2-223 to R2.
LDRNESH   R11,[R0]                    ;  Conditionally load R11 with the sign-extended contents
                                      ;  of the halfword address contained in R0.
HERE                                  ;  Generate PC relative offset to address FRED.
STRH      R5, [PC,#(FRED-HERE-8)];   Store the halfword in R5 at address FRED
FRED
```

BLOCK DATA TRANSFER (LDM, STM)

The instruction is executed only if the condition is true. The various conditions are defined in Table 3-5. The instruction encoding is shown in Figure 3-22.

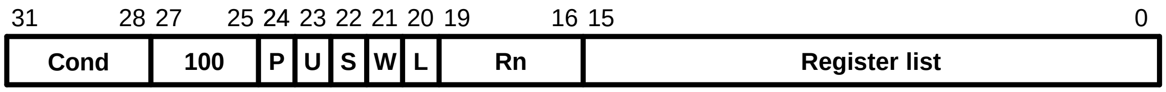
Block data transfer instructions are used to load (LDM) or store (STM) any subset of the currently visible registers. They support all possible stacking modes, maintaining full or empty stacks which can grow up or down memory, and they are very efficient instructions for saving or restoring context, or for moving large blocks of data around main memory.

THE REGISTER LIST

The instruction can cause the transfer of any registers in the current bank (and non-user mode programs can also transfer to and from the user bank, see below). The register list is a 16-bit field in the instruction, with each bit corresponding to a register. A 1 in bit 0 of the register field will cause R0 to be transferred, a 0 will cause it not to be transferred; similarly, bit 1 controls the transfer of R1, and so on.

Any subset of the registers, or all the registers, may be specified. The only restriction is that the register list should not be empty.

Whenever R15 is stored to memory, the stored value is the address of the STM instruction plus 12.



[19:16] Base Register

[20] Load/Store Bit
0 = Store to memory
1 = Load from memory

[21] Write-back Bit
0 = No write-back
1 = Write address into base

[22] PSR & Force User Bit
0 = Do not load PSR or user mode
1 = Load PSR or force user mode

[23] Up/Down Bit
0 = Down: subtract offset from base
1 = Up: add offset to base

[24] Pre/Post Indexing Bit
0 = Post: add offset after transfer
1 = Pre: add offset bofore transfer

[31:28] Condition Field

Figure 3-22. Block Data Transfer Instructions

ADDRESSING MODES

The transfer addresses are determined by the contents of the base register (Rn), the pre/post bit (P) and the up/down bit (U). The registers are transferred in the order lowest to highest, so R15 (if in the list) will always be transferred last. The lowest register also gets transferred to/from the lowest memory address. By way of illustration, consider the transfer of R1, R5, and R7 in the case where Rn=0x1000 and write back of the modified base is required (W=1). Figure 3-23 ~ 3-26 shows the sequence of register transfers, the addresses used, and the value of Rn after the instruction has completed.

In all cases, had write back of the modified base not been required (W=0), Rn would have retained its initial value of 0x1000 unless it was also in the transfer list of a load multiple register instruction, when it would have been overwritten with the loaded value.

ADDRESS ALIGNMENT

The address should normally be a word aligned quantity and non-word aligned addresses do not affect the instruction. However, the bottom 2 bits of the address will appear on **A[1:0]** and might be interpreted by the memory system.

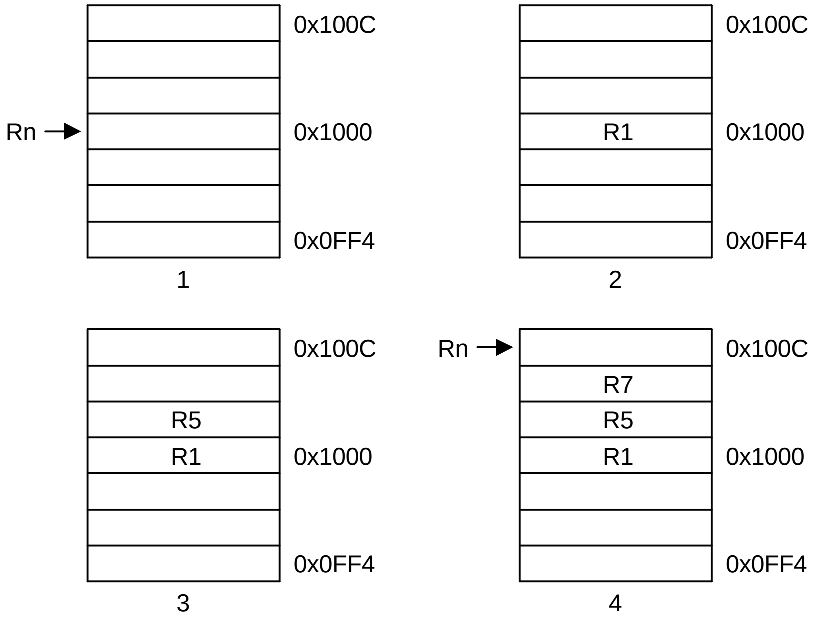


Figure 3-23. Post-Increment Addressing

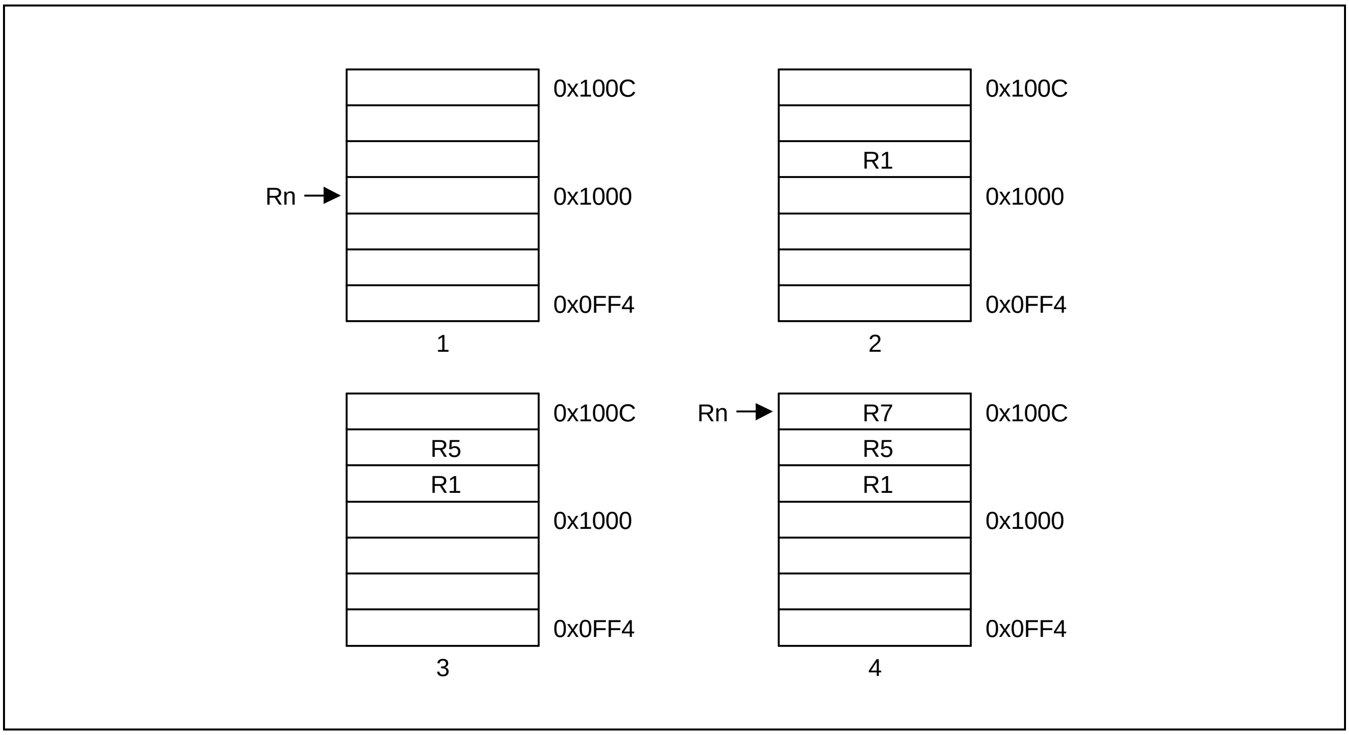


Figure 3-24. Pre-Increment Addressing

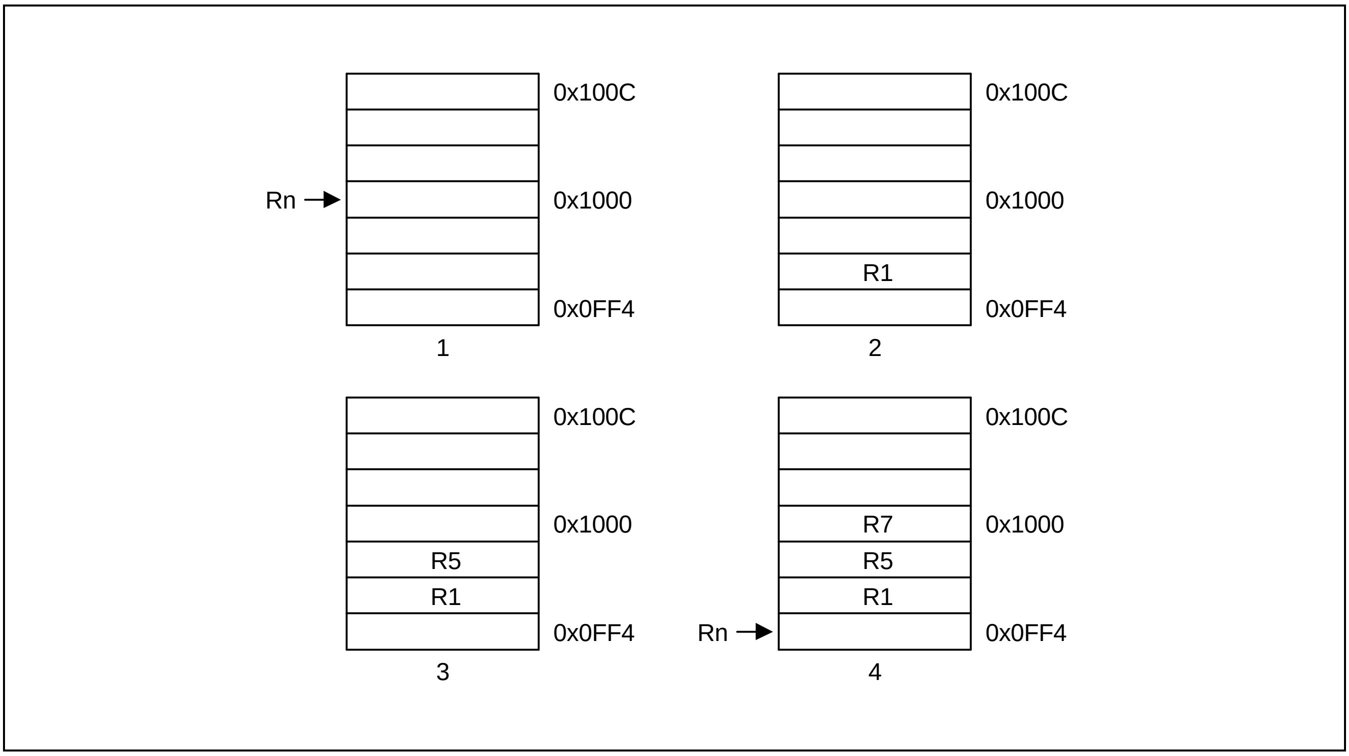


Figure 3-25. Post-Decrement Addressing

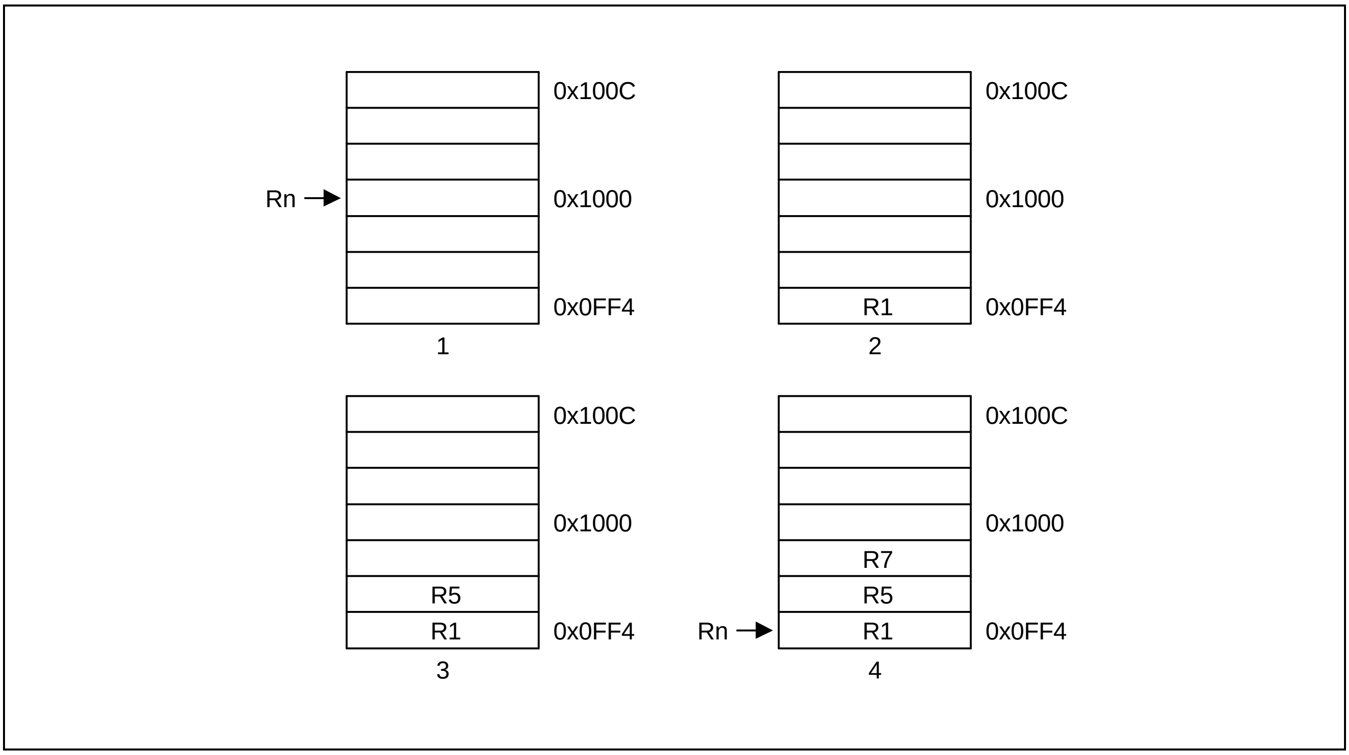


Figure 3-26. Pre-Decrement Addressing

USE OF THE S BIT

When the S bit is set in an LDM/STM instruction, its meaning depends on whether or not R15 is in the transfer list and on the type of instruction. The S bit should only be set if the instruction is to execute in a privileged mode.

LDM with R15 in Transfer List and S Bit Set (Mode Changes)

If the instruction is an LDM, SPSR_<mode> is transferred to CPSR at the same time as R15 is loaded.

STM with R15 in Transfer List and S Bit Set (User Bank Transfer)

The registers transferred are taken from the User bank rather than the bank corresponding to the current mode. This is useful for saving the user state on process switches. Base write-back should not be used when this mechanism is employed.

R15 not in List and S Bit Set (User Bank Transfer)

For both LDM and STM instructions, the User bank registers are transferred rather than the register bank corresponding to the current mode. This is useful for saving the user state on process switches. Base write-back should not be used when this mechanism is employed.

When the instruction is an LDM, care must be taken not to read from a banked register during the following cycle (inserting a dummy instruction such as MOV R0, R0 after the LDM will ensure safety).

USE OF R15 AS THE BASE

R15 should not be used as the base register in any LDM or STM instruction.

INCLUSION OF THE BASE IN THE REGISTER LIST

When write-back is specified, the base is written back at the end of the second cycle of the instruction. During an STM, the first register is written out at the start of the second cycle. An STM which includes storing the base, with the base as the first register to be stored, will therefore store the unchanged value, whereas with the base second or later in the transfer order, will store the modified value. An LDM will always overwrite the updated base if the base is in the list.

DATA ABORTS

Some legal addresses may be unacceptable to a memory management system, and the memory manager can indicate a problem with an address by taking the **ABORT** signal HIGH. This can happen on any transfer during a multiple register load or store, and must be recoverable if ARM9TDMI is to be used in a virtual memory system.

Abort during STM Instructions

If the abort occurs during a store multiple instruction, ARM9TDMI takes little action until the instruction completes, whereupon it enters the data abort trap. The memory manager is responsible for preventing erroneous writes to the memory. The only change to the internal state of the processor will be the modification of the base register if write-back was specified, and this must be reversed by software (and the cause of the abort resolved) before the instruction may be retried.

Aborts during LDM Instructions

When ARM9TDMI detects a data abort during a load multiple instruction, it modifies the operation of the instruction to ensure that recovery is possible.

- Overwriting of registers stops when the abort happens. The aborting load will not take place but earlier ones may have overwritten registers. The PC is always the last register to be written and so will always be preserved.
- The base register is restored, to its modified value if write-back was requested. This ensures recoverability in the case where the base register is also in the transfer list, and may have been overwritten before the abort occurred.

The data abort trap is taken when the load multiple has completed, and the system software must undo any base modification (and resolve the cause of the abort) before restarting the instruction.

INSTRUCTION CYCLE TIMES

Normal LDM instructions take nS + 1N + 1I and LDM PC takes (n+1)S + 2N + 1I incremental cycles, where S, N, and I are defined as sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively.
STM instructions take (n-1)S + 2N incremental cycles to execute, where *n* is the number of words transferred.

ASSEMBLER SYNTAX

<LDM|STM>{cond}<FD|ED|FA|EA|IA|IB|DA|DB> Rn{!},<Rlist>{^}

where:

{cond}	Two-character condition mnemonic. See Table 3-5.
Rn	An expression evaluating to a valid register number
<Rlist>	A list of registers and register ranges enclosed in {} (e.g. {R0,R2-R7,R10}).
{!}	If present requests write-back (W=1), otherwise W=0.
{^}	If present set S bit to load the CPSR along with the PC, or force transfer of user bank when in privileged mode.

Addressing Mode Names

There are different assembler mnemonics for each of the addressing modes, depending on whether the instruction is being used to support stacks or for other purposes. The equivalence between the names and the values of the bits in the instruction are shown in the following table 3-9.

Table 3-9. Addressing Mode Names

Name	Stack	Other	L bit	P bit	U bit
Pre-Increment Load	LDMED	LDMIB	1	1	1
Post-Increment Load	LDMFD	LDMIA	1	0	1
Pre-Increment Load	LDMEA	LDMDB	1	1	0
Post-Increment Load	LDMFA	LDMDA	1	0	0
Pre-Increment Load	STMFA	STMIB	0	1	1
Post-Increment Load	STMEA	STMIA	0	0	1
Pre-Increment Load	STMFD	STMDB	0	1	0
Post-Increment Load	STMED	STMDA	0	0	0

FD, ED, FA, and EA define pre/post indexing and the up/down bit by referencing to the form of the stack required. The F and E refer to a "full" or "empty" stack, i.e. whether a pre-index has to be done (full) before storing to the stack. The A and D refer to whether the stack is ascending or descending. If ascending, an STM will go up and an LDM down, if descending, vice versa.

IA, IB, DA, and DB allow control when an LDM/STM are not being used for stacks and simply mean Increment After, Increment Before, Decrement After, and Decrement Before.

Examples

```
          LDMFD     SP!,{R0,R1,R2}          ;  Unstack 3 registers.
          STMIA     R0,{R0-R15}             ;  Save all registers.
          LDMFD     SP!,{R15}               ;  R15 ← (SP), CPSR unchanged.
          LDMFD     SP!,{R15}^              ;  R15 ← (SP), CPSR ← SPSR_mode
                                            ;  (allowed only in privileged modes).
          STMFD     R13,{R0-R14}^           ;  Save user mode regs on stack
                                            ;  (allowed only in privileged modes).
```

These instructions may be used to save state on subroutine entry, and restore it efficiently on return to the calling routine:

```
          STMED     SP!,{R0-R3,R14}         ;  Save R0 to R3 to use as workspace
                                            ;  and R14 for returning.
          BL        somewhere               ;  This nested call will overwrite R14
          LDMED     SP!,{R0-R3,R15}         ;  Restore workspace and return.
```

SINGLE DATA SWAP (SWP)

31 28	27 23	22	21 20	19 16	15 12	11 8	7 4	3 0
Cond	00010	B	00	Rn	Rd	0000	1001	Rm

[3:0] Source Register

[15:12] Destination Register

[19:16] Base Register

[22] Byte/Word Bit
0 = Swap word quantity
1 = Swap word quantity

[31:28] Condition Field

Figure 3-27. Swap Instruction

The instruction is executed only if the condition is true. The various conditions are defined in Table 3-5. The instruction encoding is shown in Figure 3-27.

The data swap instruction is used to swap a byte or word quantity between a register and external memory. This instruction is implemented as a memory read followed by a memory write which are “locked” together (the processor cannot be interrupted until both operations are completed, and the memory manager is warned to treat them as inseparable). This class of instruction is particularly useful for implementing software semaphores.

The swap address is determined by the contents of the base register (Rn). The processor first reads the contents of the swap address. Then it writes the contents of the source register (Rm) to the swap address, and stores the old memory contents in the destination register (Rd). The same register may be specified as both the source and destination.

The **LOCK** output goes HIGH for the duration of the read and write operations to signal to the external memory manager that they are locked together, and should be allowed to complete without interruption. This is important in multi-processor systems where the swap instruction is the only indivisible instruction which may be used to implement semaphores; control of the memory must not be removed from a processor while it is performing a locked operation.

BYTES AND WORDS

This instruction class may be used to swap an byte (B=1) or a word (B=0) between an ARM9TDMI register and memory. The SWP instruction is implemented as a LDR followed by an STR and the action of these is as described in the section on single data transfers. In particular, the description of Big and Little-Endian configuration applies to the SWP instruction.

USE OF R15

Do not use R15 as an operand (Rd, Rn or Rs) in an SWP instruction.

DATA ABORTS

If the address used for the swap is unacceptable to a memory management system, the memory manager can flag the problem by driving ABORT HIGH. This can happen on either the read or the write cycle (or both), and in either case, the Data Abort trap will be taken. It is up to the system software to resolve the cause of the problem, then the instruction can be restarted and the original program continues.

INSTRUCTION CYCLE TIMES

Swap instructions take 1S + 2N +1I incremental cycles to execute, where S, N, and I are defined as sequential (S-cycle), non-sequential, and internal (I-cycle), respectively.

ASSEMBLER SYNTAX

<SWP>{cond}{B} Rd,Rm,[Rn]

{cond}	Two-character condition mnemonic. See Table 3-5.
{B}	If B is present then byte transfer, otherwise word transfer
Rd,Rm,Rn	Expressions evaluating to valid register numbers

Examples

```
        SWP      R0,R1,[R2]          ;  Load R0 with the word addressed by R2, and
                                     ;  store R1 at R2.
        SWPB     R2,R3,[R4]          ;  Load R2 with the byte addressed by R4, and
                                     ;  store bits 0 to 7 of R3 at R4.
        SWPEQ    R0,R0,[R1]          ;  Conditionally swap the contents of the
                                     ;  word addressed by R1 with R0.
```

SOFTWARE INTERRUPT (SWI)

The instruction is executed only if the condition is true. The various conditions are defined in Table 3-5. The instruction encoding is shown in Figure 3-28 below.

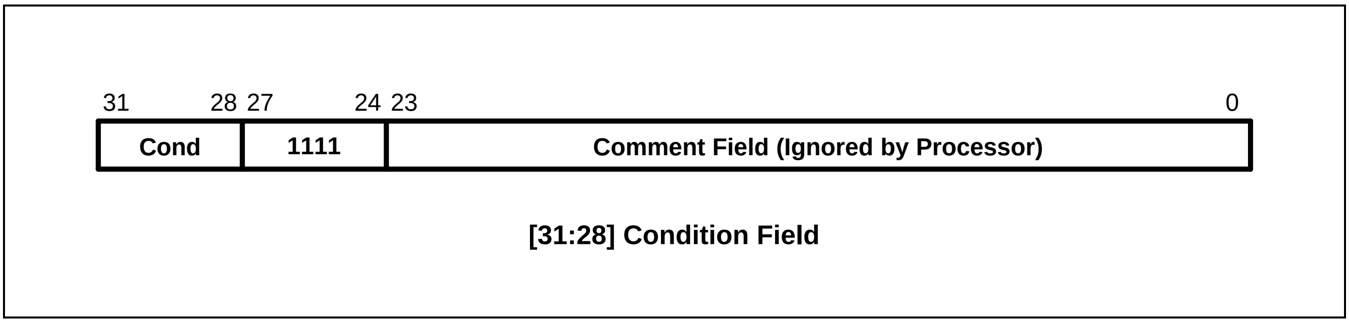


Figure 3-28. Software Interrupt Instruction

The software interrupt instruction is used to enter Supervisor mode in a controlled manner. The instruction causes the software interrupt trap to be taken, which effects the mode change. The PC is then forced to a fixed value (0x08) and the CPSR is saved in SPSR_svc. If the SWI vector address is suitably protected (by external memory management hardware) from modification by the user, a fully protected operating system may be constructed.

RETURN FROM THE SUPERVISOR

The PC is saved in R14_svc upon entering the software interrupt trap, with the PC adjusted to point to the word after the SWI instruction. MOVS PC,R14_svc will return to the calling program and restore the CPSR.

Note that the link mechanism is not re-entrant, so if the supervisor code wishes to use software interrupts within itself, it must first save a copy of the return address and SPSR.

COMMENT FIELD

The bottom 24 bits of the instruction are ignored by the processor, and may be used to communicate information to the supervisor code. For instance, the supervisor may look at this field and use it to index into an array of entry points for routines which perform the various supervisor functions.

INSTRUCTION CYCLE TIMES

Software interrupt instructions take 2S + 1N incremental cycles to execute, where S and N are defined as sequential (S-cycle) and non-sequential (N-cycle).

ASSEMBLER SYNTAX

SWI{cond} <expression>

{cond} Two-character condition mnemonic, Table 3-5.

<expression> Evaluated and placed in the comment field (which is ignored by ARM9TDMI).

Examples

```
        SWI      ReadC               ;  Get next character from read stream.
        SWI      WriteI+"k”          ;  Output a "k" to the write stream.
        SWINE    0                   ;  Conditionally call supervisor with 0 in comment field.
```

Supervisor code

The previous examples assume that suitable supervisor code exists, for instance:

```
        0x08 B Supervisor            ;  SWI entry point
        EntryTable                   ;  Addresses of supervisor routines
        DCD ZeroRtn
        DCD ReadCRtn
        DCD WriteIRtn
        • • •
        Zero     EQU  0
ReadC   EQU  256
WriteI  EQU  512

        Supervisor                   ;  SWI has routine required in bits 8-23 and data (if any) in
                                     ;  bits 0–7. Assume R13_svc points to a suitable stack
        STMFD    R13,{R0-R2,R14}     ;  Save work registers and return address.
        LDR      R0,[R14,#-4]        ;  Get SWI instruction.
        BIC      R0,R0,#0xFF000000   ;  Clear top 8 bits.
        MOV      R1,R0,LSR#8         ;  Get routine offset.
        ADR      R2,EntryTable       ;  Get start address of entry table.
        LDR      R15,[R2,R1,LSL#2]   ;  Branch to appropriate routine.
        WriteIRtn                    ;  Enter with character in R0 bits 0-7.
        • • •
        LDMFD    R13,{R0-R2,R15}^    ;  Restore workspace and return,
                                     ;  restoring processor mode and flags.
```

COPROCESSOR DATA OPERATIONS (CDP)

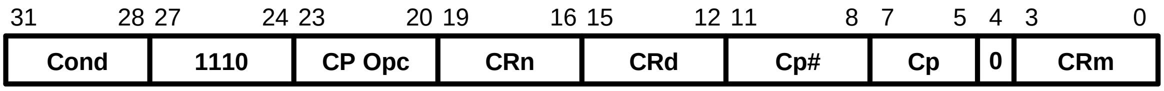
The instruction is executed only if the condition is true. The various conditions are defined in Table 3-5. The instruction encoding is shown in Figure 3-29.

This class of instruction is used to tell a coprocessor to perform some internal operation. No result is communicated back to ARM9TDMI, and it will not wait for the operation to complete. The coprocessor could contain a queue of such instructions awaiting execution, and their execution can overlap other activity, allowing the coprocessor and ARM9TDMI to perform independent tasks in parallel.

COPROCESSOR INSTRUCTIONS

The S3C2800, unlike some other ARM-based processors, does not have an external coprocessor interface. It does not have an on-chip coprocessor also.

So then all coprocessor instructions will cause the undefinded instruction trap to be taken on the S3C2800. These coprocessor instructions can be emulated by the undefined trap handler. Even though an external coprocessor can not be connected to the S3C2800, the coprocessor instructions are still described here in full for completeness. (Remember that any external coprocessor described in this section is a software emulation.)



[3:0] Coprocessor operand register

[7:5] Coprocessor information

[11:8] Coprocessor number

[15:12] Coprocessor destination register

[19:16] Coprocessor operand register

[23:20] Coprocessor operation code

[31:28] Condition Field

Figure 3-29. Coprocessor Data Operation Instruction

THE COPROCESSOR FIELDS

Only bit 4 and bits 24 to 31 are significant to ARM9TDMI. The remaining bits are used by coprocessors. The above field names are used by convention, and particular coprocessors may redefine the use of all fields as appropriate except CP#. The CP# field is used to contain an identifying number (in the range 0 to 15) for each coprocessor, and a coprocessor will ignore any instruction which does not contain its number in the CP# field.

The conventional interpretation of the instruction is that the coprocessor should perform an operation specified in the CP Opc field (and possibly in the CP field) on the contents of CRn and CRm, and place the result in CRd.

INSTRUCTION CYCLE TIMES

Coprocessor data operations take 1S + bI incremental cycles to execute, where *b* is the number of cycles spent in the coprocessor busy-wait loop.

S and **I** are defined as sequential (S-cycle) and internal (**I**-cycle).

ASSEMBLER SYNTAX

CDP{cond} p#,<expression1>,cd,cn,cm{,<expression2>}

{cond}	Two-character condition mnemonic. See Table 3-5.
p#	The unique number of the required coprocessor
<expression1>	Evaluated to a constant and placed in the CP Opc field
cd, cn and cm	Evaluate to the valid coprocessor register numbers CRd, CRn and CRm respectively
<expression2>	Where present is evaluated to a constant and placed in the CP field

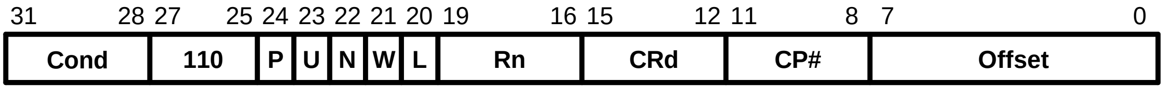
Examples

```
        CDP      p1,10,c1,c2,c3      ;  Request coproc 1 to do operation 10
                                     ;  on CR2 and CR3, and put the result in CR1.
        CDPEQ    p2,5,c1,c2,c3,2     ;  If Z flag is set, request coproc 2 to do operation 5 (type 2)
                                     ;  on CR2 and CR3, and put the result in CR1.
```

COPROCESSOR DATA TRANSFERS (LDC, STC)

The instruction is executed only if the condition is true. The various conditions are defined in Table 3-5. The instruction encoding is shown in Figure 3-30.

This class of instruction is used to load (LDC) or store (STC) a subset of a coprocessor's registers directly to memory. ARM9TDMI is responsible for supplying the memory address, and the coprocessor supplies or accepts the data and controls the number of words transferred.



[7:0] Unsigned 8 Bit Immediate Offset

[11:8] Coprocessor Number

[15:12] Coprocessor Source/Destination Register

[19:16] Base Register

[20] Load/Store Bit
0 = Store to memory
1 = Load from memory

[21] Write-back Bit
0 = No write-back
1 = Write address into base

[22] Transfer Length

[23] Up/Down Bit
0 = Down: subtract offset from base
1 = Up: add offset to base

[24] Pre/Post Indexing Bit
0 = Post: add offset after transfer
1 = Pre: add offset before transfer

[31:28] Condition Field

Figure 3-30. Coprocessor Data Transfer Instructions

THE COPROCESSOR FIELDS

The CP# field is used to identify the coprocessor which is required to supply or accept the data, and a coprocessor will respond only if its number matches the contents of this field.

The CRd field and the N bit contain information for the coprocessor which may be interpreted in different ways by different coprocessors, but by convention CRd is the register to be transferred (or the first register where more than one is to be transferred), and the N bit is used to choose one of two transfer length options. For instance N=0 could select the transfer of a single register, and N=1 could select the transfer of all the registers for context switching.

ADDRESSING MODES

ARM9TDMI is responsible for providing the address used by the memory system for the transfer, and the addressing modes available are a subset of those used in single data transfer instructions. Note, however, that the immediate offsets are 8 bits wide and specify word offsets for coprocessor data transfers, whereas they are 12 bits wide and specify byte offsets for single data transfers.

The 8-bit unsigned immediate offset is shifted left 2 bits and either added to (U=1) or subtracted from (U=0) the base register (Rn); this calculation may be performed either before (P=1) or after (P=0) the base is used as the transfer address. The modified base value may be overwritten back into the base register (if W=1), or the old value of the base may be preserved (W=0). Note that post-indexed addressing modes require explicit setting of the W bit, unlike LDR and STR, which always write-back when post-indexed.

The value of the base register, modified by the offset in a pre-indexed instruction, is used as the address for the transfer of the first word. The second word (if more than one is transferred) will go to or come from an address one word (4 bytes) higher than the first transfer, and the address will be incremented by one word for each subsequent transfer.

ADDRESS ALIGNMENT

The base address should normally be a word aligned quantity. The bottom 2 bits of the address will appear on **A[1:0]** and might be interpreted by the memory system.

USE OF R15

If Rn is R15, the value used will be the address of the instruction plus 8 bytes. Base write-back to R15 must not be specified.

DATA ABORTS

If the address is legal but the memory manager generates an abort, the data trap will be taken. The write-back of the modified base will take place, but all other processor state will be preserved. The coprocessor is partly responsible for ensuring that the data transfer can be restarted after the cause of the abort is resolved, and must ensure that any subsequent action it undertakes can be repeated when the instruction is retried.

INSTRUCTION CYCLE TIMES

Coprocessor data transfer instructions take (n-1)S + 2N + bI incremental cycles to execute, where:

n	The number of words transferred.
b	The number of cycles spent in the coprocessor busy-wait loop.

S, N, and I are defined as sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle), respectively.

ASSEMBLER SYNTAX

<LDC|STC>{cond}{L} p#,cd,<Address>

LDC Load from memory to coprocessor

STC Store from coprocessor to memory

{L} When present, perform long transfer (N=1), otherwise perform short transfer (N=0)

{cond} Two character condition mnemonic. See Table 3-5..

p# The unique number of the required coprocessor

cd An expression evaluating to a valid coprocessor register number that is placed in the CRd field

<Address> can be:

1 An expression which generates an address:
The assembler will attempt to generate an instruction using the PC as a base and a corrected immediate offset to address the location given by evaluating the expression. This will be a PC relative, pre-indexed address. If the address is out of range, an error will be generated

2 A pre-indexed addressing specification:
[Rn] offset of zero
[Rn,<#expression>]{!} offset of <expression> bytes

3 A post-indexed addressing specification:
[Rn],<#expression offset of <expression> bytes
{!} write back the base register (set the W bit) if ! is present
Rn is an expression evaluating to a valid ARM9TDMI register number.

NOTE

If Rn is R15, the assembler will subtract 8 from the offset value to allow for ARM9TDMI pipelining.

Examples

```
LDC      p1,c2,table          ;  Load c2 of coproc 1 from address
                              ;  table, using a PC relative address.
STCEQL   p2,c3,[R5,#24]!      ;  Conditionally store c3 of coproc 2
                              ;  into an address 24 bytes up from R5,
                              ;  write this address back to R5, and use
                              ;  long transfer option (probably to store multiple words).
```

NOTE

Although the address offset is expressed in bytes, the instruction offset field is in words. The assembler will adjust the offset appropriately.

COPROCESSOR REGISTER TRANSFERS (MRC, MCR)

The instruction is executed only if the condition is true. The various conditions are defined in Table 3-5. The instruction encoding is shown in Figure 3-31.

This class of instruction is used to communicate information directly between ARM9TDMI and a coprocessor. An example of a coprocessor to ARM9TDMI register transfer (MRC) instruction would be a FIX of a floating point value held in a coprocessor, where the floating point number is converted into a 32-bit integer within the coprocessor, and the result is then transferred to ARM9TDMI register. A FLOAT of a 32-bit value in ARM9TDMI register into a floating point value within the coprocessor illustrates the use of ARM9TDMI register to coprocessor transfer (MCR).

An important use of this instruction is to communicate control information directly from the coprocessor into the ARM9TDMI CPSR flags. As an example, the result of a comparison of two floating point values within a coprocessor can be moved to the CPSR to control the subsequent flow of execution.

31 28	27 24	23 21	20	19 16	15 12	11 8	7 5	4	3 0
Cond	1110	CP Opc	L	CRn	Rd	CP#	CP	1	CRm

[3:0] Coprocessor Operand Register

[7:5] Coprocessor Information

[11:8] Coprocessor Number

[15:12] ARM Source/Destination Register

[19:16] Coprocessor Source/Destination Register

[20] Load/Store Bit
0 = Store to coprocessor
1 = Load from coprocessor

[21] Coprocessor Operation Mode

[31:28] Condition Field

Figure 3-31. Coprocessor Register Transfer Instructions

THE COPROCESSOR FIELDS

The CP# field is used, as for all coprocessor instructions, to specify which coprocessor is being called upon.

The CP Opc, CRn, CP, and CRm fields are used only by the coprocessor, and the interpretation presented here is derived from convention only. Other interpretations are allowed where the coprocessor functionality is incompatible with this one. The conventional interpretation is that the CP Opc and CP fields specify the operation the coprocessor is required to perform, CRn is the coprocessor register which is the source or destination of the transferred information, and CRm is a second coprocessor register which may be involved in some way which depends on the particular operation specified.

TRANSFERS TO R15

When a coprocessor register transfer to ARM9TDMI has R15 as the destination, bits 31, 30, 29, and 28 of the transferred word are copied into the N, Z, C, and V flags, respectively. The other bits of the transferred word are ignored, and the PC and other CPSR bits are unaffected by the transfer.

TRANSFERS FROM R15

A coprocessor register transfer from ARM9TDMI with R15 as the source register will store the PC+12.

INSTRUCTION CYCLE TIMES

MRC instructions take 1S + (b+1)I +1C incremental cycles to execute, where S, I and C are defined as sequential (S-cycle), internal (I-cycle), and coprocessor register transfer (C-cycle), respectively. MCR instructions take 1S + bI +1C incremental cycles to execute, where *b* is the number of cycles spent in the coprocessor busy-wait loop.

ASSEMBLER SYNTAX

<MCR|MRC>{cond} p#,<expression1>,Rd,cn,cm{,<expression2>}

MRC	Move from coprocessor to ARM9TDMI register (L=1)
MCR	Move from ARM9TDMI register to coprocessor (L=0)
{cond}	Two-character condition mnemonic. See Table 3-5
p#	The unique number of the required coprocessor
<expression1>	Evaluated to a constant and placed in the CP Opc field
Rd	An expression evaluating to a valid ARM9TDMI register number
cn and cm	Expressions evaluating to the valid coprocessor register numbers CRn and CRm respectively
<expression2>	Where present is evaluated to a constant and placed in the CP field

Examples

```
        MRC      p2,5,R3,c5,c6          ;  Request coproc 2 to perform operation 5
                                        ;  on c5 and c6, and transfer the (single
                                        ;  32-bit word) result back to R3.
        MCR      p6,0,R4,c5,c6          ;  Request coproc 6 to perform operation 0
                                        ;  on R4 and place the result in c6.
        MRCEQ    p3,9,R3,c5,c6,2        ;  Conditionally request coproc 3 to
                                        ;  perform operation 9 (type 2) on c5 and
                                        ;  c6, and transfer the result back to R3.
```

UNDEFINED INSTRUCTION

The instruction is executed only if the condition is true. The various conditions are defined in Table 3-5. The instruction format is shown in Figure 3-32.

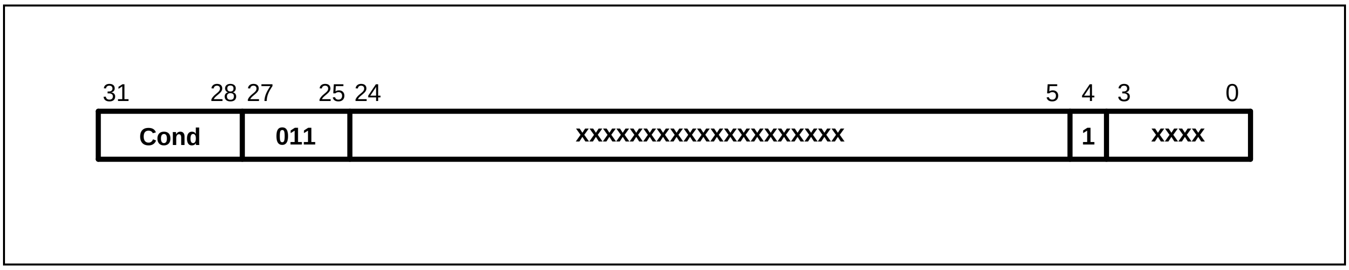


Figure 3-32. Undefined Instruction

If the condition is true, the undefined instruction trap will be taken.

Note that the undefined instruction mechanism involves offering this instruction to any coprocessors which may be present, and all coprocessors must refuse to accept it by driving **CPA** and **CPB** HIGH.

INSTRUCTION CYCLE TIMES

This instruction takes 2S + 1I + 1N cycles, where S, N, and I are defined as sequential (S-cycle), non-sequential (N-cycle), and internal (I-cycle).

ASSEMBLER SYNTAX

The assembler has no mnemonics for generating this instruction. If it is adopted in the future for some specified use, suitable mnemonics will be added to the assembler. Until then, this instruction must not be used.

INSTRUCTION SET EXAMPLES

The following examples show ways in which the basic ARM9TDMI instructions can combine to give efficient codes. None of these methods saves a great deal of execution time (although they may save some), mostly they just save codes.

USING THE CONDITIONAL INSTRUCTIONS

Using Conditional instruction for Logical OR

```
CMP      Rn,#p              ; If Rn=p OR Rm=q THEN GOTO Label.
BEQ      Label
CMP      Rm,#q
BEQ      Label
```

This can be replaced by

```
CMP      Rn,#p
CMPNE    Rm,#q              ; If condition not satisfied try other test.
BEQ      Label
```

Absolute Value

```
TEQ      Rn,#0              ; Test sign
RSBMI    Rn,Rn,#0           ; and 2's complement if necessary.
```

Multiplication by 4, 5, or 6 (Run Time)

```
MOV      Rc,Ra,LSL#2        ; Multiply by 4,
CMP      Rb,#5              ; Test value,
ADDCS    Rc,Rc,Ra           ; Complete multiply by 5,
ADDHI    Rc,Rc,Ra           ; Complete multiply by 6.
```

Combining Discrete and Range Tests

```
TEQ      Rc,#127            ; Discrete test,
CMPNE    Rc,#” ”-1          ; Range test
MOVLS    Rc,#”.”            ; IF Rc<= "" OR Rc=ASCII(127)
                            ; THEN Rc:= "."
```

Division and Remainder

A number of divide routines for specific applications are provided in source form as part of the ANSI C library provided with the ARM Cross Development Toolkit, available from your supplier. A short general purpose divide routine is as follows.

```
                                                ; Enter with numbers in Ra and Rb.
          MOV       Rcnt,#1                     ; Bit to control the division.
Div1      CMP       Rb,#0x80000000              ; Move Rb until greater than Ra.
          CMPCC     Rb,Ra
          MOVCC     Rb,Rb,ASL#1
          MOVCC     Rcnt,Rcnt,ASL#1
          BCC       Div1
          MOV       Rc,#0
Div2      CMP       Ra,Rb                       ; Test for possible subtraction.
          SUBCS     Ra,Ra,Rb                    ; Subtract if ok,
          ADDCS     Rc,Rc,Rcnt                  ; Put relevant bit into result
          MOVS      Rcnt,Rcnt,LSR#1             ; Shift control bit
          MOVNE     Rb,Rb,LSR#1                 ; Halve unless finished.
          BNE       Div2                        ; Divide result in Rc, remainder in Ra.
```

Overflow Detection in the ARM9TDMI

1. Overflow in unsigned multiply with a 32-bit result

```
          UMULL     Rd,Rt,Rm,Rn                 ; 3 to 6 cycles
          TEQ       Rt,#0                       ; +1 cycle and a register
          BNE       overflow
```

2. Overflow in signed multiply with a 32-bit result

```
          SMULL     Rd,Rt,Rm,Rn                 ; 3 to 6 cycles
          TEQ       Rt,Rd ASR#31                ; +1 cycle and a register
          BNE       overflow
```

3. Overflow in unsigned multiply accumulate with a 32-bit result

```
          UMLAL     Rd,Rt,Rm,Rn                 ; 4 to 7 cycles
          TEQ       Rt,#0                       ; +1 cycle and a register
          BNE       overflow
```

4. Overflow in signed multiply accumulate with a 32-bit result

```
          SMLAL     Rd,Rt,Rm,Rn                 ; 4 to 7 cycles
          TEQ       Rt,Rd, ASR#31               ; +1 cycle and a register
          BNE       overflow
```

5. Overflow in unsigned multiply accumulate with a 64-bit result

```
UMULL     Rl,Rh,Rm,Rn         ;  3 to 6 cycles
ADDS      Rl,Rl,Ra1           ;  Lower accumulate
ADC       Rh,Rh,Ra2           ;  Upper accumulate
BCS       overflow            ;  1 cycle and 2 registers
```

6. Overflow in signed multiply accumulate with a 64-bit result

```
SMULL     Rl,Rh,Rm,Rn         ;  3 to 6 cycles
ADDS      Rl,Rl,Ra1           ;  Lower accumulate
ADC       Rh,Rh,Ra2           ;  Upper accumulate
BVS       overflow            ;  1 cycle and 2 registers
```

NOTE

Overflow checking is not applicable to unsigned and signed multiplies with a 64-bit result, since an overflow does not occur in such calculations.

PSEUDO-RANDOM BINARY SEQUENCE GENERATOR

It is often necessary to generate (pseudo-) random numbers and the most efficient algorithms are based on shift generators with exclusive-OR feedback rather like a cyclic redundancy check generator. Unfortunately the sequence of a 32-bit generator needs more than one feedback tap to be maximal length (i.e. 2^32-1 cycles before repetition), so this example uses a 33-bit register with taps at bits 33 and 20. The basic algorithm is newbit: = bit 33 or bit 20, shift left the 33-bit number and put in newbit at the bottom; this operation is performed for all the newbits needed (i.e. 32-bits). The entire operation can be done in 5 S cycles:

```
                              ;  Enter with seed in Ra (32-bits),
                              ;  Rb (1 bit in Rb lsb), uses Rc.
TST       Rb,Rb,LSR#1         ;  Top bit into carry
MOVS      Rc,Ra,RRX           ;  33-bit rotate right
ADC       Rb,Rb,Rb            ;  Carry into lsb of Rb
EOR       Rc,Rc,Ra,LSL#12     ;  (involved!)
EOR       Ra,Rc,Rc,LSR#20     ;  (similarly involved!) new seed in Ra, Rb as before
```

MULTIPLICATION BY CONSTANT USING THE BARREL SHIFTER

Multiplication by 2^n (1,2,4,8,16,32..)

```
MOV         Ra, Rb, LSL #n
```

Multiplication by 2^n+1 (3,5,9,17..)

```
ADD         Ra,Ra,Ra,LSL #n
```

Multiplication by 2^n-1 (3,7,15..)

```
RSB         Ra,Ra,Ra,LSL #n
```

Multiplication by 6

```
ADD         Ra,Ra,Ra,LSL #1         ;  Multiply by 3
MOV         Ra,Ra,LSL#1             ;  and then by 2
```

Multiply by 10 and add in extra number

```
ADD         Ra,Ra,Ra,LSL#2          ;  Multiply by 5
ADD         Ra,Rc,Ra,LSL#1          ;  Multiply by 2 and add in next digit
```

General recursive method for Rb := Ra*C, C a constant:

1. If C even, say C = 2^n*D, D odd:

```
D=1:        MOV   Rb,Ra,LSL #n
D<>1:       {Rb := Ra*D}
MOV         Rb,Rb,LSL #n
```

2. If C MOD 4 = 1, say C = 2^n*D+1, D odd, n>1:

```
D=1:        ADD   Rb,Ra,Ra,LSL #n
D<>1:       {Rb := Ra*D}
ADD         Rb,Ra,Rb,LSL #n
```

3. If C MOD 4 = 3, say C = 2^n*D-1, D odd, n>1:

```
D=1:        RSB   Rb,Ra,Ra,LSL #n
D<>1:       {Rb := Ra*D}
RSB         Rb,Ra,Rb,LSL #n
```

This is not quite optimal, but close. An example of its non-optimality is multiply by 45 which is done by:

```
RSB         Rb,Ra,Ra,LSL#2          ;  Multiply by 3
RSB         Rb,Ra,Rb,LSL#2          ;  Multiply by 4*3-1 = 11
ADD         Rb,Ra,Rb,LSL# 2         ;  Multiply by 4*11+1 = 45
```

rather than by:

```
ADD         Rb,Ra,Ra,LSL#3          ;  Multiply by 9
ADD         Rb,Rb,Rb,LSL#2          ;  Multiply by 5*9 = 45
```

LOADING A WORD FROM AN UNKNOWN ALIGNMENT

```
                                      ;   Enter with address in Ra (32 bits) uses
                                      ;   Rb, Rc result in Rd. Note d must be less than c e.g. 0,1
        BIC        Rb,Ra,#3           ;   Get word aligned address
        LDMIA      Rb,{Rd,Rc}         ;   Get 64 bits containing answer
        AND        Rb,Ra,#3           ;   Correction factor in bytes
        MOVS       Rb,Rb,LSL#3        ;    ...now in bits and test if aligned
        MOVNE      Rd,Rd,LSR Rb       ;   Produce bottom of result word (if not aligned)
        RSBNE      Rb,Rb,#32          ;   Get other shift amount
        ORRNE      Rd,Rd,Rc,LSL Rb    ;   Combine two halves to get result
```

THUMB INSTRUCTION SET FORMAT

The thumb instruction sets are 16-bit versions of ARM instruction sets (32-bit format). The ARM instructions are reduced to 16-bit versions, Thumb instructions, at the cost of versatile functions of the ARM instruction sets. The thumb instructions are decompressed to the ARM instructions by the Thumb decompressor inside the ARM9TDMI core. As the Thumb instructions are compressed ARM instructions, the Thumb instructions have the 16-bit format instructions and have some restrictions. The restrictions by 16-bit format is fully notified for using the Thumb instructions.

FORMAT SUMMARY

The THUMB instruction set formats are shown in the following figure.

<table>
<tr><td></td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td></td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td colspan="2">Op</td><td colspan="5">Offset5</td><td colspan="3">Rs</td><td colspan="3">Rd</td><td>Move Shifted register</td></tr>
<tr><td>2</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>I</td><td>Op</td><td colspan="3">Rn/offset3</td><td colspan="3">Rs</td><td colspan="3">Rd</td><td>Add/subtract</td></tr>
<tr><td>3</td><td>0</td><td>0</td><td>1</td><td colspan="2">Op</td><td colspan="3">Rd</td><td colspan="8">Offset8</td><td>Move/compare/add/
subtract immediate</td></tr>
<tr><td>4</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="4">Op</td><td colspan="3">Rs</td><td colspan="3">Rd</td><td>ALU operations</td></tr>
<tr><td>5</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td colspan="2">Op</td><td>H1</td><td>H2</td><td colspan="3">Rs/Hs</td><td colspan="3">Rd/Hd</td><td>Hi regiter operations
/branch exchange</td></tr>
<tr><td>6</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td colspan="3">Rd</td><td colspan="8">Word8</td><td>PC-relative load</td></tr>
<tr><td>7</td><td>0</td><td>1</td><td>0</td><td>1</td><td>L</td><td>B</td><td>0</td><td colspan="3">Ro</td><td colspan="3">Rb</td><td colspan="3">Rd</td><td>Load/store with register
offset</td></tr>
<tr><td>8</td><td>0</td><td>1</td><td>0</td><td>1</td><td>H</td><td>S</td><td>1</td><td colspan="3">Ro</td><td colspan="3">Rb</td><td colspan="3">Rd</td><td>Load/store sign-extended
byte/halfword</td></tr>
<tr><td>9</td><td>0</td><td>1</td><td>1</td><td>B</td><td>L</td><td colspan="5">Offset5</td><td colspan="3">Rb</td><td colspan="3">Rd</td><td>Load/store with immediate
offset</td></tr>
<tr><td>10</td><td>1</td><td>0</td><td>0</td><td>0</td><td>L</td><td colspan="5">Offset5</td><td colspan="3">Rb</td><td colspan="3">Rd</td><td>Load/store halfword</td></tr>
<tr><td>11</td><td>1</td><td>0</td><td>0</td><td>1</td><td>L</td><td colspan="3">Rd</td><td colspan="8">Word8</td><td>SP-relative load/store</td></tr>
<tr><td>12</td><td>1</td><td>0</td><td>1</td><td>0</td><td>SP</td><td colspan="3">Rd</td><td colspan="8">Word8</td><td>Load address</td></tr>
<tr><td>13</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>S</td><td colspan="7">SWord7</td><td>Add offset to stack pointer</td></tr>
<tr><td>14</td><td>1</td><td>0</td><td>1</td><td>1</td><td>L</td><td>1</td><td>0</td><td>R</td><td colspan="8">Rlist</td><td>Push/pop register</td></tr>
<tr><td>15</td><td>1</td><td>1</td><td>0</td><td>0</td><td>L</td><td colspan="3">Rb</td><td colspan="8">Rlist</td><td>Multiple load/store</td></tr>
<tr><td>16</td><td>1</td><td>1</td><td>0</td><td>1</td><td colspan="4">Cond</td><td colspan="8">Softset8</td><td>Conditional branch</td></tr>
<tr><td>17</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td><td colspan="8">Value8</td><td>Software interrupt</td></tr>
<tr><td>18</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td colspan="11">Offset11</td><td>Unconditional branch</td></tr>
<tr><td>19</td><td>1</td><td>1</td><td>1</td><td>1</td><td>H</td><td colspan="11">Offset</td><td>Long branch with link</td></tr>
<tr><td></td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td><td></td></tr>
</table>

Figure 3-33. THUMB Instruction Set Formats

OPCODE SUMMARY

The following table summarizes the THUMB instruction set. For further information about a particular instruction please refer to the sections listed in the right-most column.

Table 3-10. THUMB Instruction Set Opcodes

Mnemonic	Instruction	Lo-Register Operand	Hi-Register Operand	Condition Codes Set
ADC	Add with Carry	4	–	4
ADD	Add	4	–	4 (1)
AND	AND	4	–	4
ASR	Arithmetic Shift Right	4	–	4
B	Unconditional branch	4	–	–
Bxx	Conditional branch	4	–	–
BIC	Bit Clear	4	–	4
BL	Branch and Link	–	–	–
BX	Branch and Exchange	4	4	–
CMN	Compare Negative	4	–	4
CMP	Compare	4	4	4
EOR	EOR	4	–	4
LDMIA	Load multiple	4	–	–
LDR	Load word	4	–	–
LDRB	Load byte	4	–	–
LDRH	Load halfword	4	–	–
LSL	Logical Shift Left	4	–	4
LDSB	Load sign-extended byte	4	–	–
LDSH	Load sign-extended halfword	4	–	–
LSR	Logical Shift Right	4	–	4
MOV	Move register	4	4	4 (2)
MUL	Multiply	4	–	4
MVN	Move Negative register	4	–	4
ADC	Add with Carry	4	–	4
ADD	Add	4	–	4 (1)
AND	AND	4	–	4
ASR	Arithmetic Shift Right	4	–	4
B	Unconditional branch	4	–	–

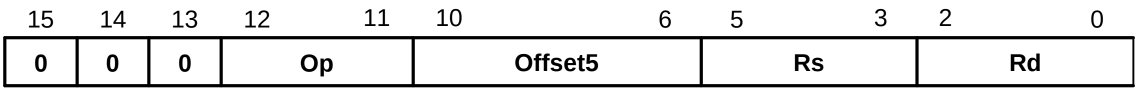
Table 3-10. THUMB Instruction Set Opcodes (Continued)

Mnemonic	Instruction	Lo-Register Operand	Hi-Register Operand	Condition Codes Set
Bxx	Conditional branch	4	–	–
BIC	Bit Clear	4	–	4
BL	Branch and Link	–	–	–
BX	Branch and Exchange	4	4	–
CMN	Compare Negative	4	–	4
CMP	Compare	4	4	4
EOR	EOR	4	–	4
LDMIA	Load multiple	4	–	–

NOTES:

1. The condition codes are unaffected by the format 5, 12 and 13 versions of this instruction.
2. The condition codes are unaffected by the format 5 version of this instruction.

FORMAT 1: MOVE SHIFTED



[2:0] Destination Register

[5:3] Source Register

[10:6] Immediate Vale

[12:11] Opcode
0 = LSL
1 = LSR
2 = ASR

Figure 3-34. Format 1

OPERATION

These instructions move a shifted value between Lo registers. The THUMB assembler syntax is shown in Table 3-11.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-11. Summary of Format 1 Instructions

OP	THUMB Assembler	ARM Equipment	Action
00	LSL Rd, Rs, #Offset5	MOVS Rd, Rs, LSL #Offset5	Shift Rs left by a 5-bit immediate value and store the result in Rd.
01	LSR Rd, Rs, #Offset5	MOVS Rd, Rs, LSR #Offset5	Perform logical shift right on Rs by a 5-bit immediate value and store the result in Rd.
10	ASR Rd, Rs, #Offset5	MOVS Rd, Rs, ASR #Offset5	Perform arithmetic shift right on Rs by a 5-bit immediate value and store the result in Rd.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-11. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

```
        LSR        R2, R5, #27          ;  Logical shift right the contents
                                        ;  of R5 by 27 and store the result in R2.
                                        ;  Set condition codes on the result.
```

FORMAT 2: ADD/SUBTRACT

15	14	13	12	11	10	9	8 6	5 3	2 0
0	**0**	**0**	**1**	**1**	**I**	**Op**	**Rn/Offset3**	**Rs**	**Rd**

[2:0] Destination Register

[5:3] Source Register

[8:6] Register/Immediate Vale

[9] Opcode
0 = ADD
1 = SUB

[10] Immediate Flag
0 = Register operand
1 = Immediate oerand

Figure 3-35. Format 2

OPERATION

These instructions allow the contents of a Lo register or a 3-bit immediate value to be added to or subtracted from a Lo register. The THUMB assembler syntax is shown in Table 3-12.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-12. Summary of Format 2 Instructions

OP	I	THUMB Assembler	ARM Equipment	Action
0	0	ADD Rd, Rs, Rn	ADDS Rd, Rs, Rn	Add contents of Rn to contents of Rs. Place result in Rd.
0	1	ADD Rd, Rs, #Offset3	ADDS Rd, Rs, #Offset3	Add 3-bit immediate value to contents of Rs. Place result in Rd.
1	0	SUB Rd, Rs, Rn	SUBS Rd, Rs, Rn	Subtract contents of Rn from contents of Rs. Place result in Rd.
1	1	SUB Rd, Rs, #Offset3	SUBS Rd, Rs, #Offset3	Subtract 3-bit immediate value from contents of Rs. Place result in Rd.

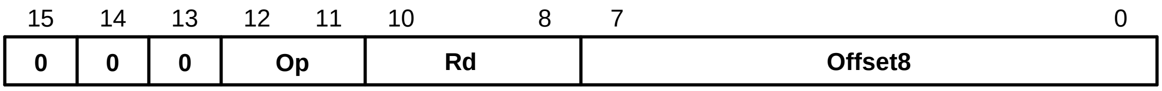
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-12. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

```
ADD      R0, R3, R4          ;  R0 := R3 + R4 and set condition codes on the result.
SUB      R6, R2, #6          ;  R6 := R2 - 6 and set condition codes.
```

FORMAT 3: MOVE/COMPARE/ADD/SUBTRACT IMMEDIATE



[7:0] Immediate Vale

[10:8] Source/Destination Register

[12:11] Opcode
0 = MOV
1 = CMP
2 = ADD
3 = SUB

Figure 3-36. Format 3

OPERATION

The instructions in this group perform operations between a Lo register and an 8-bit immediate value. The THUMB assembler syntax is shown in Table 3-13.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-13. Summary of Format 3 Instructions

OP	THUMB Assembler	ARM Equipment	Action
00	MOV Rd, #Offset8	MOVS Rd, #Offset8	Move 8-bit immediate value into Rd.
01	CMP Rd, #Offset8	CMP Rd, #Offset8	Compare contents of Rd with 8-bit immediate value.
10	ADD Rd, #Offset8	ADDS Rd, Rd, #Offset8	Add 8-bit immediate value to contents of Rd and place the result in Rd.
11	SUB Rd, #Offset8	SUBS Rd, Rd, #Offset8	Subtract 8-bit immediate value from contents of Rd and place the result in Rd.

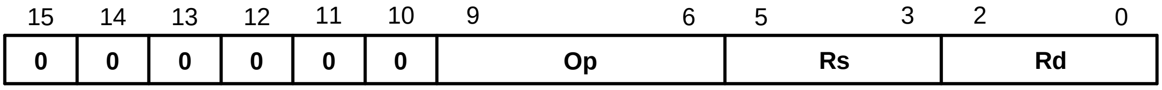
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-13. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

```
MOV     R0, #128              ; R0 := 128 and set condition codes
CMP     R2, #62               ; Set condition codes on R2 - 62
ADD     R1, #255              ; R1 := R1 + 255 and set condition codes
SUB     R6, #145              ; R6 := R6 - 145 and set condition codes
```

FORMAT 4: ALU OPERATIONS



[2:0] Source/Destination Register

[5:3] Source Register 2

[9:6] Opcode

Figure 3-37. Format 4

OPERATION

The following instructions perform ALU operations on a Lo register pair.

NOTE

All instructions in this group set the CPSR condition codes.

Table 3-14. Summary of Format 4 Instructions

OP	THUMB Assembler	ARM Equipment	Action
0000	AND Rd, Rs	ANDS Rd, Rd, Rs	Rd:= Rd AND Rs
0001	EOR Rd, Rs	EORS Rd, Rd, Rs	Rd:= Rd EOR Rs
0010	LSL Rd, Rs	MOVS Rd, Rd, LSL Rs	Rd := Rd << Rs
0011	LSR Rd, Rs	MOVS Rd, Rd, LSR Rs	Rd := Rd >> Rs
0100	ASR Rd, Rs	MOVS Rd, Rd, ASR Rs	Rd := Rd ASR Rs
0101	ADC Rd, Rs	ADCS Rd, Rd, Rs	Rd := Rd + Rs + C-bit
0110	SBC Rd, Rs	SBCS Rd, Rd, Rs	Rd := Rd - Rs - NOT C-bit
0111	ROR Rd, Rs	MOVS Rd, Rd, ROR Rs	Rd := Rd ROR Rs
1000	TST Rd, Rs	TST Rd, Rs	Set condition codes on Rd AND Rs
1001	NEG Rd, Rs	RSBS Rd, Rs, #0	Rd = - Rs
1010	CMP Rd, Rs	CMP Rd, Rs	Set condition codes on Rd - Rs
1011	CMN Rd, Rs	CMN Rd, Rs	Set condition codes on Rd + Rs
1100	ORR Rd, Rs	ORRS Rd, Rd, Rs	Rd := Rd OR Rs
1101	MUL Rd, Rs	MULS Rd, Rs, Rd	Rd := Rs * Rd
1110	BIC Rd, Rs	BICS Rd, Rd, Rs	Rd := Rd AND NOT Rs
1111	MVN Rd, Rs	MVNS Rd, Rs	Rd := NOT Rs

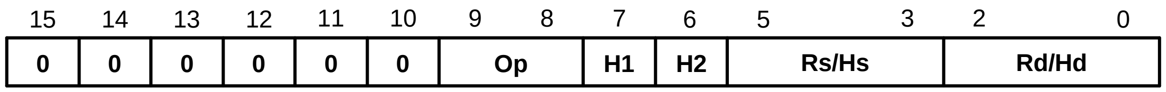
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-14. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

```
EOR      R3, R4              ;  R3 := R3 EOR R4 and set condition codes
ROR      R1, R0              ;  Rotate Right R1 by the value in R0, store
                             ;  the result in R1 and set condition codes
NEG      R5, R3              ;  Subtract the contents of R3 from zero,
                             ;  Store the result in R5. Set condition codes ie R5 = - R3
CMP      R2, R6              ;  Set the condition codes on the result of R2 - R6
MUL      R0, R7              ;  R0 := R7 * R0 and set condition codes
```

FORMAT 5: HI-REGISTER OPERATIONS/BRANCH EXCHANGE



15	14	13	12	11	10	9 8	7	6	5 3	2 0
0	0	0	0	0	0	Op	H1	H2	Rs/Hs	Rd/Hd

[2:0] Destination Register

[5:3] Source Register

[6] Hi Operand Flag 2

[7] Hi Operand Flag 1

[9:8] Opcode

Figure 3-38. Format 5

OPERATION

There are four sets of instructions in this group. The first three allow ADD, CMP and MOV operations to be performed between Lo and Hi registers, or a pair of Hi registers. The fourth, BX, allows a Branch to be performed which may also be used to switch processor state. The THUMB assembler syntax is shown in Table 3-15.

NOTE

In this group only CMP (Op = 01) sets the CPSR condition codes.

The action of H1= 0, H2 = 0 for Op = 00 (ADD), Op =01 (CMP) and Op = 10 (MOV) is undefined, and should not be used.

Table 3-15. Summary of Format 5 Instructions

Op	H1	H2	THUMB assembler	ARM equivalent	Action
00	0	1	ADD Rd, Hs	ADD Rd, Rd, Hs	Add a register in the range 8-15 to a register in the range 0-7.
00	1	0	ADD Hd, Rs	ADD Hd, Hd, Rs	Add a register in the range 0-7 to a register in the range 8-15.
00	1	1	ADD Hd, Hs	ADD Hd, Hd, Hs	Add two registers in the range 8-15
01	0	1	CMP Rd, Hs	CMP Rd, Hs	Compare a register in the range 0-7 with a register in the range 8-15. Set the condition code flags on the result.
01	1	0	CMP Hd, Rs	CMP Hd, Rs	Compare a register in the range 8-15 with a register in the range 0-7. Set the condition code flags on the result.

Table 3-15. Summary of Format 5 Instructions (Continued)

Op	H1	H2	THUMB assembler	ARM equivalent	Action
01	1	1	CMP Hd, Hs	CMP Hd, Hs	Compare two registers in the range 8-15. Set the condition code flags on the result.
10	0	1	MOV Rd, Hs	MOV Rd, Hs	Move a value from a register in the range 8-15 to a register in the range 0-7.
10	1	0	MOV Hd, Rs	MOV Hd, Rs	Move a value from a register in the range 0-7 to a register in the range 8-15.
10	1	1	MOV Hd, Hs	MOV Hd, Hs	Move a value between two registers in the range 8-15.
11	0	0	BX Rs	BX Rs	Perform branch (plus optional state change) to address in a register in the range 0-7.
11	0	1	BX Hs	BX Hs	Perform branch (plus optional state change) to address in a register in the range 8-15.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-15. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

THE BX INSTRUCTION

BX performs a Branch to a routine whose start address is specified in a Lo or Hi register.

Bit 0 of the address determines the processor state on entry to the routine:

Bit 0 = 0 Causes the processor to enter ARM state.

Bit 0 = 1 Causes the processor to enter THUMB state.

NOTE

The action of H1 = 1 for this instruction is undefined, and should not be used.

Examples

Hi-Register Operations

```
          ADD        PC, R5                   ;  PC := PC + R5 but don't set the condition codes.
          CMP        R4, R12                  ;  Set the condition codes on the result of R4 - R12.
          MOV        R15, R14                 ;  Move R14 (LR) into R15 (PC)
                                              ;  but don't set the condition codes,
                                              ;  eg. return from subroutine.
```

Branch and Exchange

```
                                              ;  Switch from THUMB to ARM state.
          ADR        R1,outofTHUMB            ;  Load address of outofTHUMB into R1.
          MOV        R11,R1
          BX         R11                      ;  Transfer the contents of R11 into the PC.
                                              ;  Bit 0 of R11 determines whether
                                              ;  ARM or THUMB state is entered, ie. ARM state here.
          •
          •
          ALIGN
          CODE32
          outofTHUMB                          ;  Now processing ARM instructions...
```

USING R15 AS AN OPERAND

If R15 is used as an operand, the value will be the address of the instruction + 4 with bit 0 cleared. Executing a BX PC in THUMB state from a non-word aligned address will result in unpredictable execution.

FORMAT 6: PC-RELATIVE LOAD

15	14	13	12	11	10 8	7 0
0	**0**	**0**	**0**	**0**	**Rd**	**Word 8**

[7:0] Immediate Value

[10:8] Destination Register

Figure 3-39. Format 6

OPERATION

This instruction loads a word from an address specified as a 10-bit immediate offset from the PC. The THUMB assembler syntax is shown below.

Table 3-16. Summary of PC-Relative Load Instruction

THUMB assembler	ARM equivalent	Action
LDR Rd, [PC, #Imm]	LDR Rd, [R15, #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the PC. Load the word from the resulting address into Rd.

NOTE: The value specified by #Imm is a full 10-bit address, but must always be word-aligned (ie with bits 1:0 set to 0), since the assembler places #Imm >> 2 in field Word 8. The value of the PC will be 4 bytes greater than the address of this instruction, but bit 1 of the PC is forced to 0 to ensure it is word aligned.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

```
          LDR R3,[PC,#844]                  ;  Load into R3 the word found at the
                                            ;  address formed by adding 844 to PC.
                                            ;  bit[1] of PC is forced to zero.
                                            ;  Note that the THUMB opcode will contain
                                            ;  211 as the Word8 value.
```

FORMAT 7: LOAD/STORE WITH REGISTER OFFSET

15	14	13	12	11	10	9	8 6	5 3	2 0
0	**1**	**0**	**1**	**L**	**B**	**0**	**Ro**	**Rb**	**Rd**

[2:0] Source/Destination Register

[5:3] Base Register

[8:6] Offset Register

[10] Byte/Word Flag
0 = Transfer word quantity
1 = Transfer byte quantity

[11] Load/Store Flag
0 = Store to memory
1 = Load from memory

Figure 3-40. Format 7

OPERATION

These instructions transfer byte or word values between registers and memory. Memory addresses are pre-indexed using an offset register in the range 0-7. The THUMB assembler syntax is shown in Table 3-17.

Table 3-17. Summary of Format 7 Instructions

L	B	THUMB assembler	ARM equivalent	Action
0	0	STR Rd, [Rb, Ro]	STR Rd, [Rb, Ro]	Pre-indexed word store: Calculate the target address by adding together the value in Rb and the value in Ro. Store the contents of Rd at the address.
0	1	STRB Rd, [Rb, Ro]	STRB Rd, [Rb, Ro]	Pre-indexed byte store: Calculate the target address by adding together the value in Rb and the value in Ro. Store the byte value in Rd at the resulting address.
1	0	LDR Rd, [Rb, Ro]	LDR Rd, [Rb, Ro]	Pre-indexed word load: Calculate the source address by adding together the value in Rb and the value in Ro. Load the contents of the address into Rd.
1	1	LDRB Rd, [Rb, Ro]	LDRB Rd, [Rb, Ro]	Pre-indexed byte load: Calculate the source address by adding together the value in Rb and the value in Ro. Load the byte value at the resulting address.

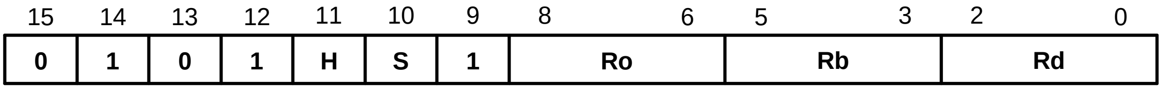
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-17. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

```
        STR     R3, [R2,R6]         ;  Store word in R3 at the address
                                    ;  formed by adding R6 to R2.
        LDRB    R2, [R0,R7]         ;  Load into R2 the byte found at
                                    ;  the address formed by adding R7 to R0.
```

FORMAT 8: LOAD/STORE SIGN-EXTENDED BYTE/HALFWORD



[2:0] Destination Register

[5:3] Base Register

[8:6] Offset Register

[10] Sign-Extended Flag
0 = Operand not sing-extended
1 = Operand sing-extended

[11] H Flag

Figure 3-41. Format 8

OPERATION

These instructions load optionally sign-extended bytes or halfwords, and store halfwords. The THUMB assembler syntax is shown below.

Table 3-18. Summary of format 8 instructions

S	H	THUMB assembler	ARM equivalent	Action
0	0	STRH Rd, [Rb, Ro]	STRH Rd, [Rb, Ro]	Store halfword: Add Ro to base address in Rb. Store bits 0-15 of Rd at the resulting address.
0	1	LDRH Rd, [Rb, Ro]	LDRH Rd, [Rb, Ro]	Load halfword: Add Ro to base address in Rb. Load bits 0-15 of Rd from the resulting address, and set bits 16-31 of Rd to 0.
1	0	LDSB Rd, [Rb, Ro]	LDRSB Rd, [Rb, Ro]	Load sign-extended byte: Add Ro to base address in Rb. Load bits 0-7 of Rd from the resulting address, and set bits 8-31 of Rd to bit 7.
1	1	LDSH Rd, [Rb, Ro]	LDRSH Rd, [Rb, Ro]	Load sign-extended halfword: Add Ro to base address in Rb. Load bits 0-15 of Rd from the resulting address, and set bits 16-31 of Rd to bit 15.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-18. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

```
STRH    R4, [R3, R0]        ;  Store the lower 16 bits of R4 at the
                            ;  address formed by adding R0 to R3.
LDSB    R2, [R7, R1]        ;  Load into R2 the sign extended byte
                            ;  found at the address formed by adding R1 to R7.
LDSH    R3, [R4, R2]        ;  Load into R3 the sign extended halfword
                            ;  found at the address formed by adding R2 to R4.
```

FORMAT 9: LOAD/STORE WITH IMMEDIATE OFFSET

15	14	13	12	11	10 6	5 3	2 0
0	**1**	**1**	**B**	**L**	**Offset5**	**Rb**	**Rd**

[2:0] Source/Destination Register

[5:3] Base Register

[10:6] Offset Register

[11] Load/Store Flag
0 = Store to memory
1 = Load from memory

[12] Byte/Word Flad
0 = Transfer word quantity
1 = Transfer byte quantity

Figure 3-42. Format 9

OPERATION

These instructions transfer byte or word values between registers and memory using an immediate 5 or 7-bit offset. The THUMB assembler syntax is shown in Table 3-19.

Table 3-19. Summary of Format 9 Instructions

L	B	THUMB assembler	ARM equivalent	Action
0	0	STR Rd, [Rb, #Imm]	STR Rd, [Rb, #Imm]	Calculate the target address by adding together the value in Rb and Imm. Store the contents of Rd at the address.
1	0	LDR Rd, [Rb, #Imm]	LDR Rd, [Rb, #Imm]	Calculate the source address by adding together the value in Rb and Imm. Load Rd from the address.
0	1	STRB Rd, [Rb, #Imm]	STRB Rd, [Rb, #Imm]	Calculate the target address by adding together the value in Rb and Imm. Store the byte value in Rd at the address.
1	1	LDRB Rd, [Rb, #Imm]	LDRB Rd, [Rb, #Imm]	Calculate source address by adding together the value in Rb and Imm. Load the byte value at the address into Rd.

NOTE: For word accesses (B = 0), the value specified by #Imm is a full 7-bit address, but must be word-aligned (ie with bits 1:0 set to 0), since the assembler places #Imm >> 2 in the Offset5 field.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-19. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

```
LDR     R2, [R5,#116]     ; Load into R2 the word found at the
                          ; address formed by adding 116 to R5.
                          ; Note that the THUMB opcode will
                          ; contain 29 as the Offset5 value.
STRB    R1, [R0,#13]      ; Store the lower 8 bits of R1 at the
                          ; address formed by adding 13 to R0.
                          ; Note that the THUMB opcode will
                          ; contain 13 as the Offset5 value.
```

FORMAT 10: LOAD/STORE HALFWORD

15	14	13	12	11	10 6	5 3	2 0
0	**1**	**0**	**0**	**L**	**Offset5**	**Rb**	**Rd**

[2:0] Source/Destination Register

[5:3] Base Register

[10:6] Immediate Value

[11] Load/Store Flag
0 = Store to memory
1 = Load from memory

Figure 3-43. Format 10

OPERATION

These instructions transfer halfword values between a Lo register and memory. Addresses are pre-indexed, using a 6-bit immediate value. The THUMB assembler syntax is shown in Table 3-20.

Table 3-20. Halfword Data Transfer Instructions

L	THUMB assembler	ARM equivalent	Action
0	STRH Rd, [Rb, #Imm]	STRH Rd, [Rb, #Imm]	Add #Imm to base address in Rb and store bits 0–15 of Rd at the resulting address.
1	LDRH Rd, [Rb, #Imm]	LDRH Rd, [Rb, #Imm]	Add #Imm to base address in Rb. Load bits 0-15 from the resulting address into Rd and set bits 16-31 to zero.

NOTE: #Imm is a full 6-bit address but must be halfword-aligned (ie with bit 0 set to 0) since the assembler places #Imm >> 1 in the Offset5 field.

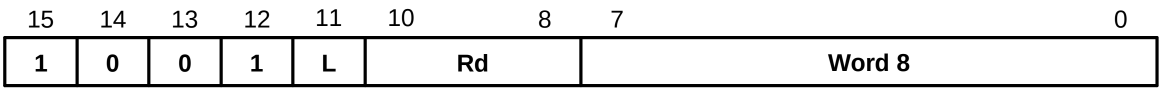
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-20. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

```
        STRH      R6, [R1, #56]        ;  Store the lower 16 bits of R4 at the address formed by
                                       ;  adding 56 R1. Note that the THUMB opcode will contain
                                       ;  28 as the Offset5 value.
        LDRH      R4, [R7, #4]         ;  Load into R4 the halfword found at the address formed by
                                       ;  adding 4 to R7. Note that the THUMB opcode will contain
                                       ;  2 as the Offset5 value.
```

FORMAT 11: SP-RELATIVE LOAD/STORE



[7:0] Immediate Value

[10:8] Destination Register

[11] Load/Store Bit
0 = Store to memory
1 = Load from memory

Figure 3-44. Format 11

OPERATION

The instructions in this group perform an SP-relative load or store. The THUMB assembler syntax is shown in the following table.

Table 3-21. SP-Relative Load/Store Instructions

L	THUMB assembler	ARM equivalent	Action
0	STR Rd, [SP, #Imm]	STR Rd, [R13 #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the SP (R7). Store the contents of Rd at the resulting address.
1	LDR Rd, [SP, #Imm]	LDR Rd, [R13 #Imm]	Add unsigned offset (255 words, 1020 bytes) in Imm to the current value of the SP (R7). Load the word from the resulting address into Rd.

NOTE: The offset supplied in #Imm is a full 10-bit address, but must always be word-aligned (ie bits 1:0 set to 0), since the assembler places #Imm >> 2 in the Word8 field.

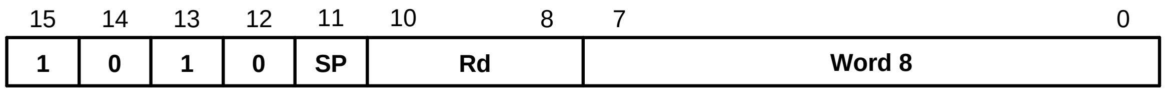
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-21. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

```
          STR       R4, [SP,#492]       ;  Store the contents of R4 at the address
                                        ;  formed by adding 492 to SP (R13).
                                        ;  Note that the THUMB opcode will contain
                                        ;  123 as the Word8 value.
```

FORMAT 12: LOAD ADDRESS



[7:0] 8-bit Unsigned Constant

[10:8] Destination Register

[11] Source
0 = PC
1 = SP

Figure 3-45. Format 12

OPERATION

These instructions calculate an address by adding an 10-bit constant to either the PC or the SP, and load the resulting address into a register. The THUMB assembler syntax is shown in the following table.

Table 3-22. Load Address

SP	THUMB assembler	ARM equivalent	Action
0	ADD Rd, PC, #Imm	ADD Rd, R15, #Imm	Add #Imm to the current value of the program counter (PC) and load the result into Rd.
1	ADD Rd, SP, #Imm	ADD Rd, R13, #Imm	Add #Imm to the current value of the stack pointer (SP) and load the result into Rd.

NOTE: The value specified by #Imm is a full 10-bit value, but this must be word-aligned (ie with bits 1:0 set to 0) since the assembler places #Imm >> 2 in field Word 8.

Where the PC is used as the source register (SP = 0), bit 1 of the PC is always read as 0. The value of the PC will be 4 bytes greater than the address of the instruction before bit 1 is forced to 0.

The CPSR condition codes are unaffected by these instructions.

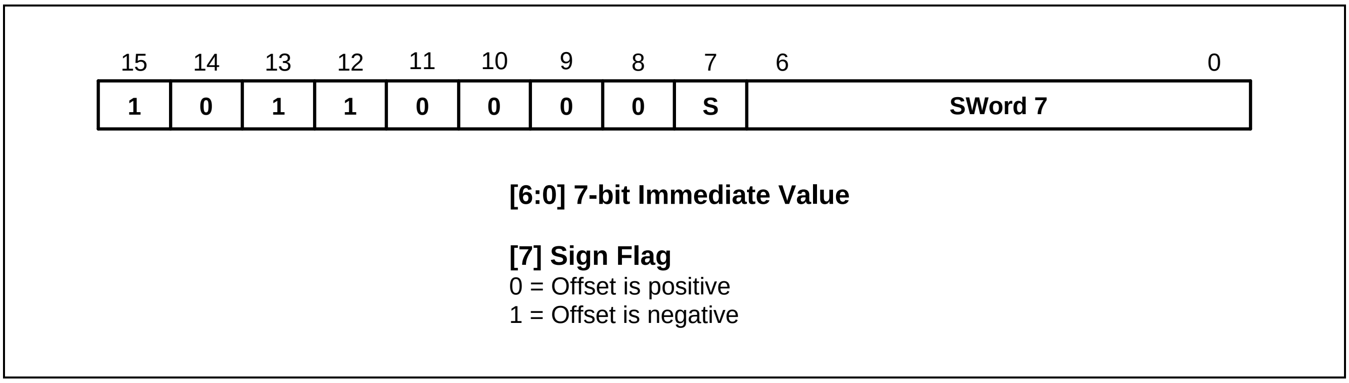
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-22. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

```
ADD      R2, PC, #572         ;  R2 := PC + 572, but don't set the
                              ;  condition codes. bit[1] of PC is forced to zero.
                              ;  Note that the THUMB opcode will
                              ;  contain 143 as the Word8 value.
ADD      R6, SP, #212         ;  R6 := SP (R13) + 212, but don't
                              ;  set the condition codes.
                              ;  Note that the THUMB opcode will
                              ;  contain 53 as the Word 8 value.
```

FORMAT 13: ADD OFFSET TO STACK POINTER



[6:0] 7-bit Immediate Value

[7] Sign Flag
0 = Offset is positive
1 = Offset is negative

Figure 3-46. Format 13

OPERATION

This instruction adds a 9-bit signed constant to the stack pointer. The following table shows the THUMB assembler syntax.

Table 3-23. The ADD SP Instruction

S	THUMB assembler	ARM equivalent	Action
0	ADD SP, #Imm	ADD R13, R13, #Imm	Add #Imm to the stack pointer (SP).
1	ADD SP, # -Imm	SUB R13, R13, #Imm	Add #-Imm to the stack pointer (SP).

NOTE: The offset specified by #Imm can be up to -/+ 508, but must be word-aligned (ie with bits 1:0 set to 0) since the assembler converts #Imm to an 8-bit sign + magnitude number before placing it in field SWord7.

The condition codes are not set by this instruction.

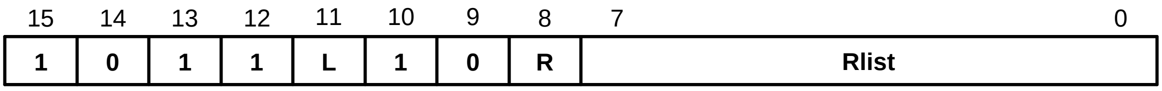
INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-23. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

```
ADD     SP, #268        ; SP (R13) := SP + 268, but don't set the condition codes.
                        ; Note that the THUMB opcode will
                        ; contain 67 as the Word7 value and S=0.
ADD     SP, #-104       ; SP (R13) := SP - 104, but don't set the condition codes.
                        ; Note that the THUMB opcode will contain
                        ; 26 as the Word7 value and S=1.
```

FORMAT 14: PUSH/POP REGISTERS



15	14	13	12	11	10	9	8	7 – 0
1	0	1	1	L	1	0	R	Rlist

[7:0] Register List

[8] PC/LR Bit
0 = Do not store LR/Load PC
1 = Store LR/Load PC

[11] Load/Store Bit
0 = Store to memory
1 = Load from memory

Figure 3-47. Format 14

OPERATION

The instructions in this group allow registers 0-7 and optionally LR to be pushed onto the stack, and registers 0-7 and optionally PC to be popped off the stack. The THUMB assembler syntax is shown in Table 3-24.

NOTE

The stack is always assumed to be Full Descending.

Table 3-24. PUSH and POP Instructions

L	R	THUMB assembler	ARM equivalent	Action
0	0	PUSH { Rlist }	STMDB R13!, { Rlist }	Push the registers specified by Rlist onto the stack. Update the stack pointer.
0	1	PUSH { Rlist, LR }	STMDB R13!, { Rlist, R14 }	Push the Link Register and the registers specified by Rlist (if any) onto the stack. Update the stack pointer.
1	0	POP { Rlist }	LDMIA R13!, { Rlist }	Pop values off the stack into the registers specified by Rlist. Update the stack pointer.
1	1	POP { Rlist, PC }	LDMIA R13!, {Rlist, R15}	Pop values off the stack and load into the registers specified by Rlist. Pop the PC off the stack. Update the stack pointer.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-24. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

```
        PUSH     {R0-R4,LR}          ;  Store R0,R1,R2,R3,R4 and R14 (LR) at
                                     ;  the stack pointed to by R13 (SP) and update R13.
                                     ;  Useful at start of a sub-routine to
                                     ;  save workspace and return address.
        POP      {R2,R6,PC}          ;  Load R2,R6 and R15 (PC) from the stack
                                     ;  pointed to by R13 (SP) and update R13.
                                     ;  Useful to restore workspace and return from sub-routine.
```

FORMAT 15: MULTIPLE LOAD/STORE

15	14	13	12	11	10 8	7 0
1	**1**	**0**	**0**	**L**	**RB**	**Rlist**

[7:0] Register List

[10:8] Base Register

[11] Load/Store Bit
0 = Store to memory
1 = Load from memory

Figure 3-48. Format 15

OPERATION

These instructions allow multiple loading and storing of Lo registers. The THUMB assembler syntax is shown in the following table.

Table 3-25. The Multiple Load/Store Instructions

L	THUMB assembler	ARM equivalent	Action
0	STMIA Rb!, { Rlist }	STMIA Rb!, { Rlist }	Store the registers specified by Rlist, starting at the base address in Rb. Write back the new base address.
1	LDMIA Rb!, { Rlist }	LDMIA Rb!, { Rlist }	Load the registers specified by Rlist, starting at the base address in Rb. Write back the new base address.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-25. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

```
STMIA    R0!, {R3-R7}           ;  Store the contents of registers R3-R7
                                ;  starting at the address specified in
                                ;  R0, incrementing the addresses for each word.
                                ;  Write back the updated value of R0.
```

FORMAT 16: CONDITIONAL BRANCH

15	14	13	12	11 8	7 0
1	1	0	1	Cond	SOffset 8

[7:0] 8-bit Signed Immediate

[11:8] Condition

Figure 3-49. Format 16

OPERATION

The instructions in this group all perform a conditional Branch depending on the state of the CPSR condition codes. The branch offset must take account of the prefetch operation, which causes the PC to be 1 word (4 bytes) ahead of the current instruction.

The THUMB assembler syntax is shown in the following table.

Table 3-26. The Conditional Branch Instructions

Cond	THUMB assembler	ARM equivalent	Action
0000	BEQ label	BEQ label	Branch if Z set (equal)
0001	BNE label	BNE label	Branch if Z clear (not equal)
0010	BCS label	BCS label	Branch if C set (unsigned higher or same)
0011	BCC label	BCC label	Branch if C clear (unsigned lower)
0100	BMI label	BMI label	Branch if N set (negative)
0101	BPL label	BPL label	Branch if N clear (positive or zero)
0110	BVS label	BVS label	Branch if V set (overflow)
0111	BVC label	BVC label	Branch if V clear (no overflow)
1000	BHI label	BHI label	Branch if C set and Z clear (unsigned higher)
1001	BLS label	BLS label	Branch if C clear or Z set (unsigned lower or same)
1010	BGE label	BGE label	Branch if N set and V set, or N clear and V clear (greater or equal)
1011	BLT label	BLT label	Branch if N set and V clear, or N clear and V set (less than)
1100	BGT label	BGT label	Branch if Z clear, and either N set and V set or N clear and V clear (greater than)
1101	BLE label	BLE label	Branch if Z set, or N set and V clear, or N clear and V set (less than or equal)

NOTES:

1. While label specifies a full 9-bit two's complement address, this must always be halfword-aligned (ie with bit 0 set to 0) since the assembler actually places label >> 1 in field SOffset8.
2. Cond = 1110 is undefined, and should not be used.
   Cond = 1111 creates the SWI instruction: see .

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-26. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

```
            CMP R0, #45                     ;   Branch to 'over' if R0 > 45.
            BGT over                        ;   Note that the THUMB opcode will contain
            •                               ;   the number of halfwords to offset.
            •
over        •                               ;   Must be halfword aligned.
```

FORMAT 17: SOFTWARE INTERRUPT

15	14	13	12	11	10	9	8	7 … 0
1	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**Value 8**

[7:0] Comment Field

Figure 3-50. Format 17

OPERATION

The SWI instruction performs a software interrupt. On taking the SWI, the processor switches into ARM state and enters Supervisor (SVC) mode.

The THUMB assembler syntax for this instruction is shown below.

Table 3-27. The SWI Instruction

THUMB assembler	ARM equivalent	Action
SWI Value 8	SWI Value 8	Perform Software Interrupt: Move the address of the next instruction into LR, move CPSR to SPSR, load the SWI vector address (0x8) into the PC. Switch to ARM state and enter SVC mode.

NOTE: Value8 is used solely by the SWI handler; it is ignored by the processor.

INSTRUCTION CYCLE TIMES

All instructions in this format have an equivalent ARM instruction as shown in Table 3-27. The instruction cycle times for the THUMB instruction are identical to that of the equivalent ARM instruction.

Examples

```
        SWI 18                          ;  Take the software interrupt exception.
                                        ;  Enter Supervisor mode with 18 as the
                                        ;  requested SWI number.
```

FORMAT 18: UNCONDITIONAL BRANCH

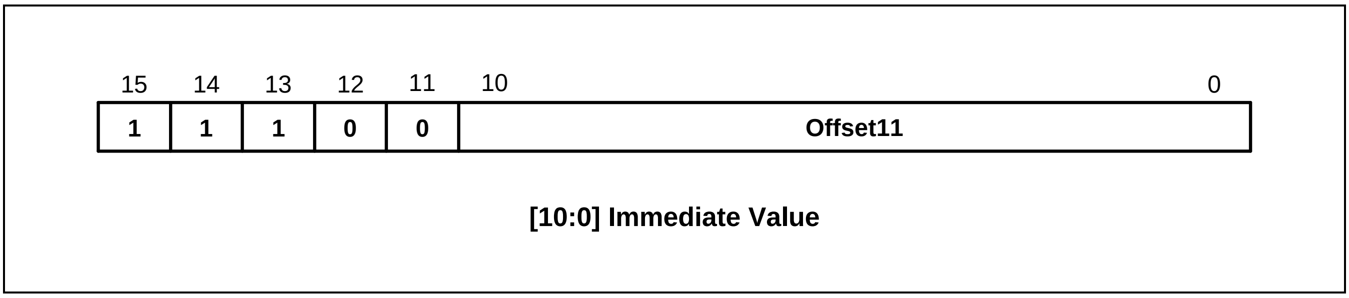


Figure 3-51. Format 18

OPERATION

This instruction performs a PC-relative Branch. The THUMB assembler syntax is shown below. The branch offset must take account of the prefetch operation, which causes the PC to be 1 word (4 bytes) ahead of the current instruction.

Table 3-28. Summary of Branch Instruction

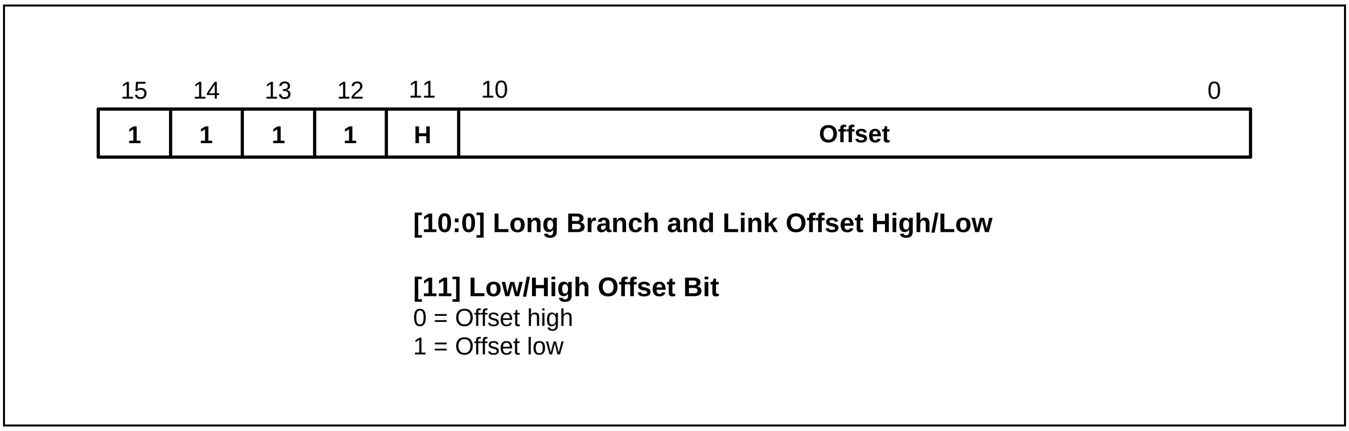
THUMB assembler	ARM equivalent	Action
B label	BAL label (halfword offset)	Branch PC relative +/- Offset11 << 1, where label is PC +/- 2048 bytes.

NOTE: The address specified by label is a full 12-bit two’s complement address,
but must always be halfword aligned (ie bit 0 set to 0), since the assembler places label >> 1 in the Offset11 field.

Examples

```
here      B here                      ;  Branch onto itself. Assembles to 0xE7FE.
                                      ;  (Note effect of PC offset).
          B jimmy                     ;  Branch to 'jimmy'.
          •                           ;  Note that the THUMB opcode will contain the number of
          •
          •                           ;  halfwords to offset.
jimmy     •                           ;  Must be halfword aligned.
```

FORMAT 19: LONG BRANCH WITH LINK



15	14	13	12	11	10 … 0
1	1	1	1	H	Offset

[10:0] Long Branch and Link Offset High/Low

[11] Low/High Offset Bit
0 = Offset high
1 = Offset low

Figure 3-52. Format 19

OPERATION

This format specifies a long branch with link.

The assembler splits the 23-bit two’s complement half-word offset specifed by the label into two 11-bit halves, ignoring bit 0 (which must be 0), and creates two THUMB instructions.

Instruction 1 (H = 0)

In the first instruction the Offset field contains the upper 11 bits of the target address. This is shifted left by 12 bits and added to the current PC address. The resulting address is placed in LR.

Instruction 2 (H =1)

In the second instruction the Offset field contains an 11-bit representation lower half of the target address. This is shifted left by 1 bit and added to LR. LR, which now contains the full 23-bit address, is placed in PC, the address of the instruction following the BL is placed in LR and bit 0 of LR is set.

The branch offset must take account of the prefetch operation, which causes the PC to be 1 word (4 bytes) ahead of the current instruction

INSTRUCTION CYCLE TIMES

This instruction format does not have an equivalent ARM instruction.

Table 3-29. The BL Instruction

L	THUMB assembler	ARM equivalent	Action
0	BL label	none	LR := PC + OffsetHigh << 12

1			temp := next instruction address
			PC := LR + OffsetLow << 1
			LR := temp | 1

Examples

```
        BL faraway              ;  Unconditionally Branch to 'faraway'
next    •                       ;  and place following instruction
        •                       ;  address, ie ’next’, in R14,the Link
                                ;  register and set bit 0 of LR high.
                                ;  Note that the THUMB opcodes will
                                ;  contain the number of halfwords to offset.
faraway •                       ;  Must be Half-word aligned.
        •
```

INSTRUCTION SET EXAMPLES

The following examples show ways in which the THUMB instructions may be used to generate small and efficient code. Each example also shows the ARM equivalent so these may be compared.

MULTIPLICATION BY A CONSTANT USING SHIFTS AND ADDS

The following shows code to multiply by various constants using 1, 2 or 3 Thumb instructions alongside the ARM equivalents. For other constants it is generally better to use the built-in MUL instruction rather than using a sequence of 4 or more instructions.

```
        Thumb       ARM
```

1. Multiplication by 2^n (1,2,4,8,...)

```
        LSL         Ra, Rb, LSL #n          ;   MOV Ra, Rb, LSL #n
```

2. Multiplication by 2^n+1 (3,5,9,17,...)

```
        LSL         Rt, Rb, #n              ;   ADD Ra, Rb, Rb, LSL #n
        ADD         Ra, Rt, Rb
```

3. Multiplication by 2^n-1 (3,7,15,...)

```
        LSL         Rt, Rb, #n              ;   RSB Ra, Rb, Rb, LSL #n
        SUB         Ra, Rt, Rb
```

4. Multiplication by -2^n (-2, -4, -8, ...)

```
        LSL          Ra, Rb, #n             ;   MOV Ra, Rb, LSL #n
        MVN          Ra, Ra                 ;   RSB Ra, Ra, #0
```

5. Multiplication by -2^n-1 (-3, -7, -15, ...)

```
        LSL         Rt, Rb, #n              ;   SUB Ra, Rb, Rb, LSL #n
        SUB         Ra, Rb, Rt
```

Multiplication by any C = {2^n+1, 2^n-1, -2^n or -2^n-1} * 2^n
Effectively this is any of the multiplications in 2 to 5 followed by a final shift. This allows the following additional constants to be multiplied. 6, 10, 12, 14, 18, 20, 24, 28, 30, 34, 36, 40, 48, 56, 60, 62 .....

```
        (2..5)                              ;   (2..5)
        LSL          Ra, Ra, #n             ;   MOV Ra, Ra, LSL #n
```

GENERAL PURPOSE SIGNED DIVIDE

This example shows a general purpose signed divide and remainder routine in both Thumb and ARM code.

Thumb code

```
;signed_divide                                  ;   Signed divide of R1 by R0: returns quotient in R0,
                                                ;   remainder in R1

;Get abs value of R0 into R3
            ASR         R2, R0, #31             ;   Get 0 or -1 in R2 depending on sign of R0
            EOR         R0, R2                  ;   EOR with -1 (0×FFFFFFFF) if negative
            SUB         R3, R0, R2              ;   and ADD 1 (SUB -1) to get abs value

;SUB always sets flag so go & report division by 0 if necessary
            BEQ         divide_by_zero

;Get abs value of R1 by xoring with 0xFFFFFFFF and adding 1 if negative
            ASR         R0, R1, #31             ;   Get 0 or -1 in R3 depending on sign of R1
            EOR         R1, R0                  ;   EOR with -1 (0×FFFFFFFF) if negative
            SUB         R1, R0                  ;   and ADD 1 (SUB -1) to get abs value

;Save signs (0 or -1 in R0 & R2) for later use in determining ; sign of quotient & remainder.
            PUSH         {R0, R2}

;Justification, shift 1 bit at a time until divisor (R0 value) ; is just <= than dividend (R1 value). To do this shift
dividend ; right by 1 and stop as soon as shifted value becomes >.
            LSR         R0, R1, #1
            MOV         R2, R3
            B           %FT0
just_l      LSL         R2, #1
0           CMP         R2, R0
            BLS         just_l
            MOV         R0, #0                  ;   Set accumulator to 0
            B           %FT0                    ;   Branch into division loop

div_l       LSR         R2, #1
0           CMP         R1, R2                  ;   Test subtract
            BCC         %FT0
            SUB         R1, R2                  ;   If successful do a real subtract
0           ADC         R0, R0                  ;   Shift result and add 1 if subtract succeeded

            CMP         R2, R3                  ;   Terminate when R2 == R3 (ie we have just
            BNE         div_l                   ;   tested subtracting the 'ones' value).
```

Now fixup the signs of the quotient (R0) and remainder (R1)

```
                POP         {R2, R3}                ;   Get dividend/divisor signs back
                EOR         R3, R2                  ;   Result sign
                EOR         R0, R3                  ;   Negate if result sign = – 1
                SUB         R0, R3
                EOR         R1, R2                  ;   Negate remainder if dividend sign = – 1
                SUB         R1, R2
                MOV         pc, lr
```

ARM Code

```
signed_divide                                       ;   Effectively zero a4 as top bit will be shifted out later
                ANDS        a4, a1, #&80000000
                RSBMI       a1, a1, #0
                EORS        ip, a4, a2, ASR #32
;ip bit 31 = sign of result
;ip bit 30 = sign of a2
                RSBCS        a2, a2, #0

;Central part is identical code to udiv (without MOV a4, #0 which comes for free as part of signed entry sequence)
                MOVS        a3, a1
                BEQ         divide_by_zero

just_l                                              ;   Justification stage shifts 1 bit at a time
                CMP         a3, a2, LSR #1
                MOVLS       a3, a3, LSL #1          ;   NB: LSL #1 is always OK if LS succeeds
                BLO         s_loop

div_l
                CMP         a2, a3
                ADC         a4, a4, a4
                SUBCS       a2, a2, a3
                TEQ         a3, a1
                MOVNE       a3, a3, LSR #1
                BNE         s_loop2
                MOV         a1, a4
                MOVS        ip, ip, ASL #1
                RSBCS       a1, a1, #0
                RSBMI       a2, a2, #0
                MOV         pc, lr
```

DIVISION BY A CONSTANT

Division by a constant can often be performed by a short fixed sequence of shifts, adds and subtracts.

Here is an example of a divide by 10 routine based on the algorithm in the ARM Cookbook in both Thumb and ARM code.

Thumb Code

```
udiv10                                  ; Take argument in a1 returns quotient in a1,
                                        ; remainder in a2
          MOV       a2, a1
          LSR       a3, a1, #2
          SUB       a1, a3
          LSR       a3, a1, #4
          ADD       a1, a3
          LSR       a3, a1, #8
          ADD       a1, a3
          LSR       a3, a1, #16
          ADD       a1, a3
          LSR       a1, #3
          ASL       a3, a1, #2
          ADD       a3, a1
          ASL       a3, #1
          SUB       a2, a3
          CMP       a2, #10
          BLT       %FT0
          ADD        a1, #1
          SUB       a2, #10
0
          MOV        pc, lr
```

ARM Code

```
udiv10                                  ; Take argument in a1 returns quotient in a1,
                                        ; remainder in a2
          SUB       a2, a1, #10
          SUB       a1, a1, a1, lsr #2
          ADD       a1, a1, a1, lsr #4
          ADD       a1, a1, a1, lsr #8
          ADD       a1, a1, a1, lsr #16
          MOV       a1, a1, lsr #3
          ADD       a3, a1, a1, asl #2
          SUBS      a2, a2, a3, asl #1
          ADDPL     a1, a1, #1
          ADDMI     a2, a2, #10
          MOV       pc, lr
```

NOTES

4 CACHES, WRITE BUFFER, and PHYSICAL ADDRESS TAG(PA TAG) RAM

ABOUT THE CACHES AND WRITE BUFFER

The ARM920T level-one memory system includes an *instruction cache* (ICache), a *data cache* (DCache), a write buffer and a *Physical Address* (PA) TAG RAM to reduce the effect of main memory bandwidth and latency on performance.

The ARM920T implements separate 16KB instruction and 16KB data caches (ICache and DCache).

The caches have the following features:

- Virtually-addressed 64-way associative cache.
- words per line (32 bytes per line) with one valid bit and two dirty bits per line, allowing half-line write-backs.
- Write-through and write-back cache operation (write-back caches are also known as copy back caches), selected per memory region by the C and B bits in the MMU translation tables (for data cache only).
- Pseudo-random or round-robin replacement, selectable via RR bit in CP15 register 1.
- Low-power CAM-RAM implementation.
- Caches independently lockable with granularity of 1/64th of cache, which is 64 words (256 bytes).
- For compatibility with Microsoft WindowsCE, and to reduce interrupt latency, the physical address corresponding to each data cache entry is stored in the PA TAG RAM for use during cache line write-backs, in addition to the VA (virtual address) TAG stored in the cache CAMs. This means that the MMU is not involved in cache write-back operations, removing the possibility of TLB misses related to the write-back address.
- Cache maintenance operations to provide efficient cleaning of the entire data cache, and to provide efficient cleaning and invalidation of small regions of virtual memory. The latter allows ICache coherency to be efficiently maintained when small code changes occur, for example self-modifying code and changes to exception vectors.

The write buffer:

- has a 16-word data buffer
- has a 4-address address buffer
- can be drained under software control, using a CP15 MCR instruction (see *Drain write buffer on page 4-17*).

The ARM920T can be drained under software control and the ARM920T put into a low-power state until an interrupt occurs, using a CP15 MCR instruction (see *Wait for interrupt on page 4-17*).

INSTRUCTION CACHE (ICache)

The ARM920T includes a 16KB instruction cache. The ICache has 512 lines of 32 bytes (8 words), arranged as a 64-way set-associative cache and uses MVAs (modified virtual addresses), translated by CP15 register 13 (see *Address translation on page 5-4*), from the ARM9TDMI core.

The ICache implements allocate-on-read-miss. Random or round-robin replacement can be selected under software control using the RR bit (CP15 register 1, bit 14). Random replacement is selected at reset.

Instructions can also be locked in the ICache so that they cannot be overwritten by a linefill. This operates with a granularity of $^{1}/_{64}$th of the cache, which is 64 words (256 bytes).

All instruction accesses are subject to MMU permission and translation checks. Instruction fetches that are aborted by the MMU do not cause linefills or instruction fetches to appear on the AMBA ASB interface

NOTE

For clarity, the I bit (bit 12 in CP15 register 1) is called the *Icr bit* throughout the following text. The C bit from the MMU translation table descriptor corresponding to the address being accessed is called the *Ctt bit*.

ICACHE ORGANIZATION

The ICache is organized as eight segments, each containing 64 lines, and each line cantaning eight words. The position of the line within the segment is a number from 0 to 63. This is called the *index*. A line in the cache can be uniquely identified by its segment and index. The index is independent of the MVA. The segment is selected by bits [7:5] of the MVA.

Bits [4:2] of the MVA specify the word within a cache line that is accessed. For halfword operations. bit [1] of the MVA sepcifies the halfword that is accessed within the word. For byte operations, bits [1:0] specify the byte within the word that is accessed.

Bits [31:8] of the MVA of each cache line are called the TAG. The MVA TAG is stored in the cache, along with the 8-words of data, when the line is loaded by a linefill.

Cache lookups compare bits [31:8] of the MVA of the access with the stored TAG to determine whether the access is a hit or miss. The cache is therfore said to be virtually addresssed. The logical model of the 16KB ICache is shown in Figure 4-1.

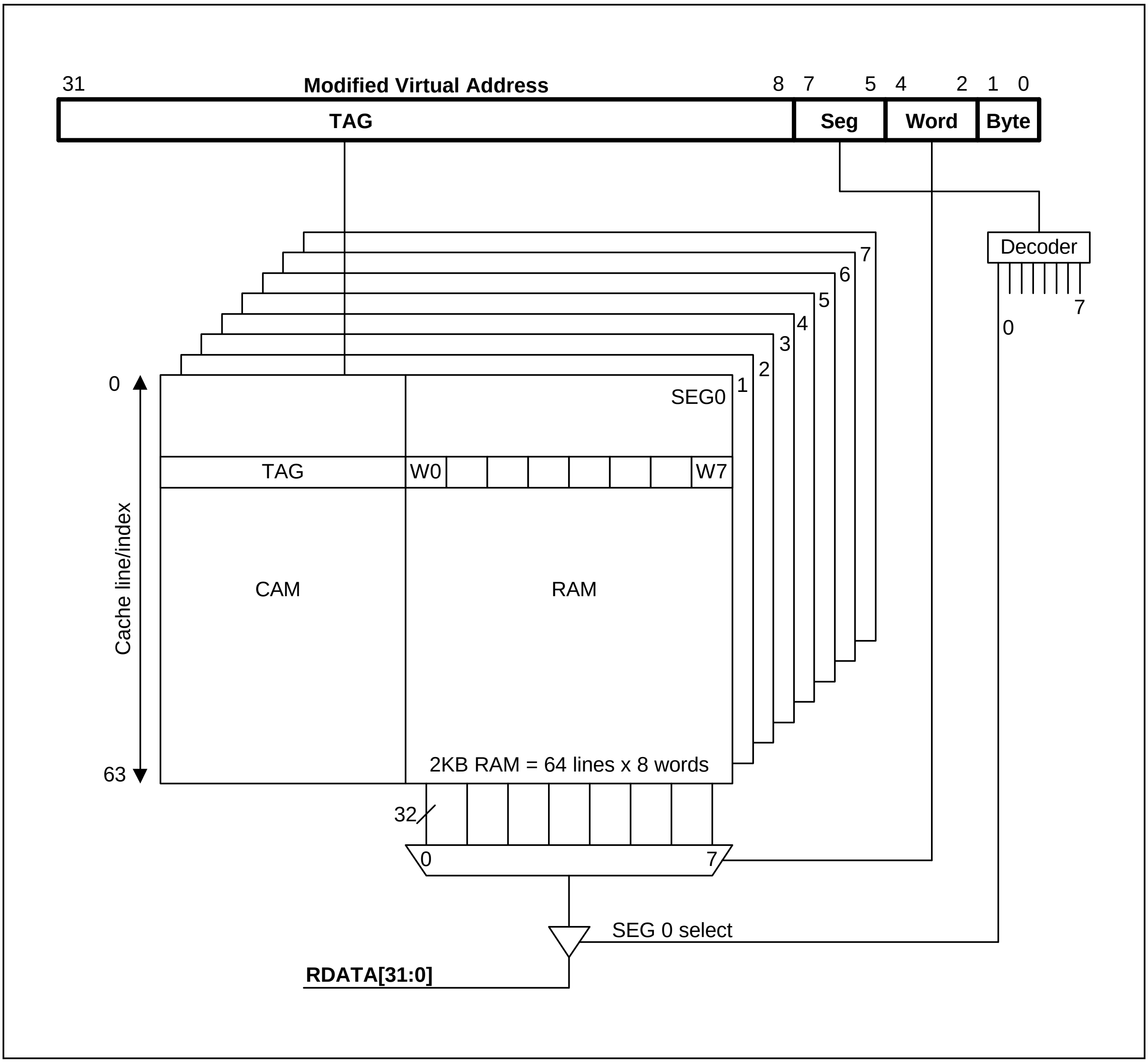


Figure 4-1. Addressing the 16KB ICache

ENABLING AND DISABLING THE ICACHE

On reset, the ICache entries are all invalidated and the ICache is disabled.

You can enable the ICache by writing 1 to the Icr bit, and disable it by writing 0 to the Icr bit.

When the ICache is disabled, the cache contents are ignored and all instruction fetchs appear on the AMBA ASB interface as separate nonsequential accesses.

The ICache is usually used with the MMU enabled, in this case the Ctt in the relevant MMU translation table descriptor indicates whether an area of memory is cacheable.

If the cache is enabled after having been enabled, all cache contents are ignored. All instruction fetches appear on the AMBA ASB interface as separate nonsequential accesses and the cache is not updated. If the cache is subsequently re-enabled its contents are unchanged. If the contents are no longer coherent with main memory, you must invalidate the ICache before you re-enable it (see *Register 7, cache operations register on page 2-28*).

If the cache is enabled with the MMU disabled, all instruction fetches are treated as cachable. No protection checks are made, and the physical address is flat-mapped to the modified virtual address.

You can enable the MMU and ICache simultaneously by writing a 1 to the M bit, and a 1 to the Icr bit in CP15 register 1, with a single MCR instruction.

NOTE

ARM920T implements a nonsequential access on the AMBA ASB interface as an A-TRAN cycle followed by an S-TRAN cycle. It does not produce N-TRAN cycles. A linefill appears as an A-TRAN cycle followed by an S-TRAN cycle.

ICACHE OPERATION

If the ICache is disabled, each instruction fetch results in a separate nonsequential memory access on the AMBA ASB interface, giving very low bus and memory performance. Therefore, you must enable the ICache as soon as possible after reset.

If the ICache is enabled, an ICache lookup is performed for each instruction fetch regardless of the setting of the Ctt bit in the relevant MMU translation table descriptor.

- If the required instruction is found in the cache, the lookup is called a cache hit. If the instruction fetch is a cache hit and Ctt=1, indicating a cacheable region of memory, then the instruction is returned from the cache to the ARM9TDMI CPU core.
- If the required instruction is not found in the cache, the lookup is called a cache miss. If it is a cache miss and Ctt=1, then an eight-word linefill is performed, possibly replacing another entry. The entry to be replaced, (called the victim), is chosen from the entries that are not locked, using either a random or round-robin replacement policy. If Ctt=0, indicating a non-cacheable region of memory, then a single nonsequential memory access appears on the AMBA ASB interface.

NOTE

If Ctt=0, indicating a non-cacheable region of memory, then the cache lookup results in a cache miss. The only way that it can result in a cache hit is if software has changed the value of the Ctt bit in the MMU translation table descriptor without invalidating the cache contents. This is a programming error. The behavior in this case is architecturally unpredictable and varies between implementations.

ICACHE REPLACEMENT ALGORITHM

The ICache and DCache replacement algorithm is selected by the RR bit in the CP15 control register (CP15 register 1, bit 14). Random replacement is selected at reset. Setting the RR bit to 1 selects round-robin replacement means that entries are replaced sequentially in each cache segment.

ICACHE LOCKDOWN

You can lock instructions into the ICache, causing the ICache to gurantee a hit, and provide optimum and predictable excution time.

If you enable the ICache, an ICache lookup is performed for each instruction fetch. If the ICache misses and the Ctt=1 than an eight-word linefill is performed. The entry to be replaced is selected by the victim pointer. You can lock instructions into the ICache by controlling the victim pointer, and forcing prefetches to the ICache.

Yoy lock instructions in the ICache by first ensuring the code to be locked is not already in the cache. You can ensure this by invalidating either the whole ICache or specific lines:

```
MCR     p15, 0, Rd, c7, c5, 0 ; Invalidate ICache
MCR     p15, 0, Rd, c7, c5, 1 ; Invalidate ICache line using MVA
```

You can use a short software routine to load the instructions into the ICache. The software routine must either be noncacheable, or already in the ICache but not in an ICache line about to be overwritten. You must enable the MMU to ensure that any TLB misses that occur while loading the instructions cause a page table walk.

The software routine operates by writing to CP15 register 9 to force the victim pointer to a specific ICache line and by using the prefetch ICache line operation to force the ICache to perform a lookup. This misses, assuming the code has been invalidated, and an 8-word linefill is performed loading the cache line into the entry specified by the victim pointer. When all the instructions have been loaded, They are then locked by writing to CP15 register 9 to set the victim pointer base to be one higher than the last entry written. All further linefills now occur in the range victim base to 63.

An example ICache lockdown routine is shown in Example 4-1. The example assumes that the number of cache lines to be loaded is not known. The address does not have to be cache line or word-aligned but this is recommended to ensure future compatibility.

NOTE

The prefetch ICache Line operation uses MVA formet, because address aliasing is not performed on the address in Rd. It is advisable for the associated TLB entry to be locked into the TLB to avoid page table walks during execution of the locked code.

Example 4-1 Icache Lockdown Routine

```
              ADRL       r0,start_address           ; address pointer
              ADRL       r1,end_address
              MOV        r2,#lockdown_base<<26      ; victim pointer
              MCR        p15,0,r2,c9,c0,1           ; write ICache victim and lockdown base

loop
              MCR        p15,0,r0,c7,c13,1           ; Prefetch ICache line
              ADD        r0,r0,#32                   ; increment address pointer to next
                                                     ; ICache line

;; do we need to increment the victim pointer?
;; test for segment 0, and if so, increment the victim pointer
;; and write the ICache victim and lockdown base.

              AND        r3,r0,#0xE0                 ; extract the segment bits from the
                                                     ; address
              CMP        r3,#0x0                     ; test for segment 0
              ADDEQ      r2,r2,#0x1<<26              ; if segment 0, increment victim pointer
              MCREQ      p15,0,r2,c9,c0,1            ; and write ICache victim and lockdown
                                                     ; base

;; have we linefilled enough code?
;; test for the address pointer being less than or equal to the
;; end_address and if so, loop and perform another linefill

              CMP        r0,r1                       ; test for less than or equal to
                                                     ; end_address
              BLE        loop                        ; if not, loop

;; have we exited with r3 pointing to segment 0?
;; if so, the ICache victim and lockdown base has already been set to one
;; higher than the last entry written.
;; if not, increment the victim pointer and write the ICache victim and
;; lockdown base.

              CMP        r3,#0x0                     ; test for segments 1 to 7
              ADDNE      r2,r2,#0x1<<26              ; if address is segment 1 to 7,
              MCRNE      p15,0,r2,c9,c0,1            ; write ICache victim and lockdown base
```

DCACHE AND WRITE BUFFER

The ARM920T includes a 16KB DCache (Data Cache) and a write buffer to reduce the effect of main memory bandwidth and latency on data access performance. The DCache has 512 lines of 32 bytes (8-words), arranged as a 64-way set-associative cache and uses MVAs (Modified Virtual Addresses) translated by CP15 register 13 (see *Address translation on page 5-4*) from the ARM9TDMI CPU core. The write buffer can hold up to 16 words of data and four separate addresses. The operation of the DCache and the write buffer are closely connected.

The DCache supports write-through and write-back memory regions, controlled by the C and B bits in each section and page descriptor within the MMU translation tables. For clarity, these bits are called Ctt and Btt in the following text. For details see *DCache and write buffer operation on page 4-8*.

Each DCache line has two dirty bits, one for the first four words of the line, the other for the last four words, and a single virtual TAG address and valid bit for the entire 8-word line. The physical address from which each line is loaded is stored in the PA TAG RAM and is used when writing modified lines back to memory.

When a store hits in the DCache, if the memory region is write-back, the associated dirty bit is set marking the appropriate half-line as being modified. If the cache line is replaced due to a linefill, or if the line is the target of a DCache clean operation, the dirty bits are used to decide whether the whole, half, or none of the line is written back to memory. The line is written back to the same physical address from which it was loaded, regardless of any changes to the MMU translation tables.

The DCache implements allocate-on-read-miss. Random or round-robin replacement can be selected under software control by the RR bit (CP15 register 1, bit 14). Random replacement is selected at reset. A linefill always loads a complete 8-word line.

Data can also be locked in the DCache so that it cannot be overwritten by a linefill. This operates with a granularity of $^{1}/_{64}$th of the cache, which is 64 words (256 bytes).

All data accesses are subject to MMU permission and translation checks. Data accesses that are aborted by the MMU do not cause linefills or data accesses to appear on the AMBA ASB interface.

For clarity, the C bit (bit 2 in CP15 register 1) is called the Ccr bit throughout the following text.

ENABLING AND DISABLING THE DCACHE AND WRITE BUFFER

On reset, the DCache entries are invalidated and the DCache is disabled, and the write buffer contents are discarded.

There is no explicit write buffer enable bit implemented in ARM920T. The write buffer is used in the following ways:

- You can enable the DCache by writing 1 to the Ccr bit, and disabled it by writing 0 to the Ccr bit.
- You must only enable the DCache when the MMU is enabled. This is because the MMU translation tables define the cache and write buffer configuration for each memory region.
- If the DCache is disabled after having been enabled, the cache contents are ignored and all data accesses appear on the AMBA ASB (Advanced System Bus) interface as separate nonsequential accesses and the cache is not updated. If the cache is subsequently re-enabled its contents are unchanged. Depending on the software system design, you might have to clean the cache after it is disabled, and invalidate it before you re-enable it. See *Cache coherence on page 4-14*.
- You can enable or disable the MMU and DCache simultaneously with a single MCR that changes the M and C bit in the control register (CP15 register 1).

DCACHE AND WRITE BUFFER OPERATION

The DCache and write buffer configuration of each memory region is controlled by the Ctt and Btt bits in each section and page descriptor in the MMU translation tables. You can modify the configuration using the DCache enable bit in the CP15 control register. This is called Ccr.

If the DCache is enabled, a DCache lookup is performed for each data access initiated by the ARM9TDMI CPU core, regardless of the value of the Ctt bit in the relevant MMU translation table descriptor. If the required data is not found, the loockup is called a *cache miss*. In this context a data access means any type of load (read), store (write), or swap instruction, including LDR, LDRB, LDRH, LDM, LDC, STR, STRB, STRH, STC, SWP and SWPB.

Accesses appear on the AMBA ASB interface in program order but the ARM9TDMI CPU core can continue executing at full speed, reading instructions and data from the caches, and writing to the DCache and write buffer, while buffered writes are being written to memory through the AMBA ASB interface.

Table4-1 describes the DCache and write buffer behavior for each type of memory configuration. *Ctt AND Ccr* means the bitwise Boolean AND of Ctt with Ccr.

Table 4-1. DCache and Write Buffer Configuration

Ctt and Ccr	Btt	Data cache, write buffer and memory access behavior
0 [1]	0	Noncached, nonbuffered (NCNB). Reads and writes are not cached. They always perform accesses on the AMBA ASB interface. Writes are not buffered. The CPU halts until the write is completed on the AMBA ASB interface. Reads and writes can be externally aborted. Cache hits never occur under normal operation. [2]
0	1	Noncached, buffered (NCB). Reads and writes are not cached, and always perform accesses on the AMBA ASB interface. Writes are placed in the write buffer and appear on the AMBA ASB interface. The CPU continues execution as soon as the write is placed in the write buffer. Reads can be externally aborted. Writes can not be externally aborted. Cache hits never occur under normal operation. [2]
1	0	Cached, write-through mode (WT). Reads that hit in the cache read the data from the cache and do not perform an access on the AMBA ASB interface. Reads that miss in the cache cause a linefill. Writes that hit in the cache update the cache. All writes are placed in the write buffer and appear on the AMBA ASB interface. The CPU continues execution as soon as the write is placed in the write buffer. Reads and writes cannot be externally aborted.
1	1	Cached write-back mode (WB). Reads that hit in the cache read the data from the cache and do not perform an AMBA ASB interface access. Reads that miss in the cache cause a linefill. Writes that hit in the cache update the cache and mark the appropriate half of the cache line as dirty, and do not cause an AMBA ASB interface access. Writes that miss in the cache are placed in the write buffer and appear on the AMBA ASB interface. The CPU continues execution as soon as the write is placed in the write buffer. Cache write-backs are buffered. Reads, Writes and write-backs cannot be externally aborted.

NOTES:

1. If the control register C bit (Ccr) is zero, it disables all lookups in the cache, while if the translation table descriptor C bit (Ctt) is zero, it only stops new data being loaded into the cache. With Ccr = 1 and Ctt = 0 the cache is still searched on every access to check whether the cache contains an entry for the data.
2. It is an operating system software error if a cache hit occurs when reading from, or writing to, a region of memory marked as NCNB or NCB. The only way this can occur is if the operating system changes the value of the C and B bits in a page table descriptor, while the cache contains data from the area of virtual memory controlled by that descriptor. The cache and memory system behavior resulting from changing the page table descriptor in this way is unpredictable. If the operating system has to change the C and B bits of a page table descriptor, it must ensure that the caches do not contain any data controlled by that descriptor. In some circumstances, the operating system might have to clean and flush the caches to ensure this.

A linefill performs an 8-word burst read from the AMBA ASB interface and places it as a new entry in the cache, possibly replacing another line at the same location within the cache. The location that is replaced, called the victim, is chosen from the entries that are not locked using either a random or round-robin replacement policy. If the cache line being replaced is marked as dirty, indicating that it has been modified and that main memory has not been updated to reflect the change, a cache writeback occurs.

Depending on whether one or both halves of the cache line are dirty, the write-back performs a 4 or 8-word sequential burst write access on the AMBA ASB interface. The CPU can then continue while the write-back data is written to memory over the AMBA ASB interface.

Load multiple (LDM) instructions accessing NCNB or NCB regions perform sequential bursts on the AMBA ASB interface. Store multiple (STM) instructions accessing NCNB regions also perform sequential bursts on the AMBA ASB interface.

The sequential burst is split into two bursts if it crosses a 1KB boundary. This is because the smallest MMU protection and mapping size is 1KB, so the memory regions on each size of the 1KB boundary can have different properties.

This means that sequential access generated by ARM920T do not cross a 1KB boundary, this can be exploited to simplify memory interface design. For example, a simple page-mode DRAM controller can perform a page-mode access for each sequential access, provided the DRAM page size is 1KB or larger.

See also *Cache coherence on page 4-14*.

DCACHE ORGANIZATION

The DCache is organized as eight segments, each containing 64 lines, and each line containing eight words. The position of the line within the segment is a number from 0 to 63. This is called the index. A line in the cache can be uniquely identified by its segment and index. The index is independent of the MVA. The segment is selected by bits [7:5] of the MVA.

Bits [4:2] of the MVA specify which word within a cache line is accessed. For halfword operations, bit [1] of the MVA specifies which halfword is accessed within the word. For byte operations, bits [1:0] specify which byte within the word is accessed.

Bits [31:8] of the MVA of each cache line are called the TAG. The MVA TAG is stored in the cache, along with the eight words of data, when the line is loaded by a linefill.

Cache lookups compare bits [31:8] of the MVA of the access with the stored TAG to determine whether the access is a hit or miss. The cache is therefore said to be virtually addressed.

The DCache logical model is the same as for the ICache. *See Figure 4-1 Addressing the 16KB ICache*.

DCACHE REPLACEMENT ALGORITHM

The DCache and ICache replacement algorithm is selected by the RR bit in the CP15 Control register (CP15 register 1, bit 14). Random replacement is selected at reset. Setting the RR bit to 1 selects round-robin replacement means that entries are replaced sequentially in each segment.

SWAP INSTRUCTIONS

Swap instruction (SWP or SWPB) behavior is dependent on whether the memory region is cacheable or noncacheable.

Swap instructions to cacheable regions of memory are useful for implementing semaphores or other synchronization primitives in multithreaded uniprocessor software systems.

Swap instructions to noncacheable memory regions are useful for synchronization between two bus masters in a multi-master bus system. This can be two processors, or one processor and a DMA controller.

When a swap instruction accesses a cacheable region of memory (write-through or write-back), the DCache and write buffer behavior will is the same as having a load followed by a store according to the normal rules described. The **BLOK** pin is not asserted during the execution of the instruction. It is guaranteed that no interrupt can occur between the load and store portions of the swap.

When a swap instruction accesses a noncacheable (NCB or NCNB) region of memory, the write buffer is drained, and a single word or byte is read from the AMBA ASB interface. The write portion of the swap is then treated as nonbufferable, regardless of the value of Btt, and the processor is stalled until the write is completed on the AMBA ASB interface. The **BLOK** pin is asserted to indicate that you can treat the read and write as an atomic operation on the bus.

Like all other data accesses, a swap to a noncacheable region that hits in the cache indicates a programming error.

DCACHE LOCKDOWN

You can lock data into the DCache, causing the DCache to guarantee a hit, and provide optimum and predictable execution time.

If you enable the DCache, a DCache lookup is performed for each load. If the DCache misses and the Ctt=1 then an eight-word linefill is performed. The entry to be replaced is selected by the victim pointer. You can lock data into the DCache by controlling the victim pointer, and forcing loads to the DCache.

You lock data in the DCache by first ensuring the data to be locked is not already in the cache. You can ensure this by cleaning and invalidating eighter the whole DCache or specific lines. Example 4-2 shows DCache invalidate and clean operations that you can perform to do this.

Example 4-2 Dcache Invalidate and Clean Operations

```
MCR p15, 0, Rd, c7, c6, 0        ; Invalidate DCache
MCR p15, 0, Rd, c7, c6, 1        ; Invalidate DCache single entry using MVA
MCR p15, 0, Rd, c7, c10, 1       ; Clean DCache single entry using MVA
MCR p15, 0, Rd, c7, c14, 1       ; Clean and Invalidate DCache single entry using MVA
MCR p15, 0, Rd, c7, c10, 2       ; Clean DCache single entry using Index
MCR p15, 0, Rd, c7, c14, 2       ; Clean and Invalidate DCache single entry using Index
```

Yoy can then use a short software routine to load the data into the DCache. You can locate the software routine in a cachable region of memory providing it does not contain any loads or stores. You must enable the MMU.

The software routine operates by writing to CP15 register 9 to force the victim pointer to a specific DCache line and by using an LDR or LDM to force the DCache to perform a lookup. This misses, assuming the data was previously invalidated, and an eight-word linefill is performed loading the cache line into the entry specified by the victim pointer. When all the data has been loaded, it is then locked by writing to CP15 register 9 to set the victim pointer base to be one higher than the last entry written. All further linefills now occur in the range victim base to 63.

An example DCache lockdown routine is shown in Example 4-3. The example assumes that the number of cache lines to be loaded in not known. The address does not have cache line or word-aligned, although it is preferable for future compatibility.

NOTE

The LDR or LDM uses VA format, because address aliasing is performed on the address.
It is advisable for the associated TLB entry to be locked into the TLB to avoid page table walks during accesses of the locked data.

Example 4-3 Dcache Lockdown Routine

```
ADRL        r0,start_address                      ; address pointer
            ADRL        r1,end_address
            MOV         r2,#lockdown_base<<26     ; victim pointer
            MCR         p15,0,r2,c9,c0,0          ; write DCache victim and lockdown
                                                  ; base

loop

            LDR         r3,[r0],#32               ; load DCache line, increment to next
                                                  ; DCache line

;; do we need to increment the victim pointer?
;; test for segment 0, and if so, increment the victim pointer and
;; write the ICache victim and lockdown base.

            AND         r3,r0,#0xE0                ; extract the segment bits from the
                                                   ; address
            CMP         r3,#0x0                              ; test for segment 0
            ADDEQ       r2,r2,#0x1<<26             ; if segment 0, increment victim pointer
            MCREQ       p15,0,r2,c9,c0,0           ; and write DCache victim and lockdown
                                                   ; base

;; have we linefilled enough code?
;; test for the address pointer being less than or equal to the end_address
;; and if so, loop and perform another linefill

            CMP         r0,r1                      ; test for less than or equal to
                                                   ; end_address,
            BLE         loop                       ; if not, loop

;; have we exited with r3 pointing to segment 0?
;; if so, the ICache victim and lockdown base has already been set to one
;; higher than the last entry written.
;; if not, increment the victim pointer and write the ICache victim and
;; lockdown base.

            CMP         r3,#0x0                    ; test for segments 1 to 7
            ADDNE       r2,r2,#0x1<<26             ; if address is segment 1 to 7,
            MCRNE       p15,0,r2,c9,c0,0           ; write DCache victim and lockdown
                                                   ; base
```

CACHE COHERENCE

The ICache and DCache contain copies of information normally held in main memory. If these copies of memory information get out of step with each other because one is updated and the others is not updated, they are said to have become incoherent. If the DCache contains a line that has been modified by a store or swap instruction, and the main memory has not been updated, the cache line is said to be dirty. Clean operations force the cache to write dirty lines back to main memory. The ICache then has to be made coherent with a changed area of memory after any changes to the instructions that appear at an MVA, and before the new instructions are executed.

On the ARM920T, software is responsible for maintaining coherence between main memory, the ICache, and the DCache.

Register 7, *cache operations register on page 2-28* describes facilities for invalidating the entire ICache or individual ICache lines, and for cleaning and/or invalidating DCache lines, or for invalidating the entire DCache.

To clean the entire DCache efficiently, software must loop though each cache entry using the *clean D single entry (using index)* operation or the *clean and invalidate D entry (using index)* operation. You must perform this using a two-level nested loop going though each index value for each segment. See *DCache organization on page 4-10*.

Example 4-4 shows an example loop for two alternative DCache cleaning operation.

Example 4-4 Dcache Cleaning Loop

```
for seg = 0 to 7
        for index = 0 to 63
                Rd = {seg,index}

                MCR p15,0,Rd,c7,c10,2       ; Clean DCache single
                                            ; entry (using index)

                                        or

                MCR p15,0,Rd,c7,c14,2       ; Clean and Invalidate
                                            ; DCache single entry
                                            ; (using index)

        next index
next seg
```

DCache, ICache, and memory coherence is generally achieved by:

- cleaning the DCache to ensure memory is up to date with all changes
- invalidating the ICache to ensure that the ICache is forced to re-fetch instructions from memory.

Software can minimize the performance penalties of cleaning and invalidating caches by:

- Cleaning only small portions of the cache when only a small area of memory has to be made coherent, for example, when updating an exception vector entry. Use Clean DCache single entry (using MVA) or Clean and Invalidate DCache single entry (using MVA).
- Invalidating only small portions of the ICache when only a small number of instructions are modified, for example, when updating an exception vector entry. Use *Invalidate ICache single entry (using MVA)*.
- Not invalidating the ICache in situations where it is known that the modified area of memory cannot be in the cache, for example, when mapping a new page into the currently running process.

Situations that necessitate cache cleaning and invalidating include:

- Writing instructions to a cacheable area of memory using STR or STM instructions, for example:

  – self-modifying code

  – JIT compilation

  – copying code from another location

  – downloading code using the EmbeddedICE JTAG debug features

  – updating an exception vector entry.
- Another bus master, such as a DMA controller, modifying a cacheable area main memory.
- Turning the MMU on or off
- Changing the virtual-to-physical mappings or Ctt, or Btt, or protection information, in the MMU page tables. The DCache must be cleaned, and both caches invalidated, before the cache and write buffer configuration of an area of memory is changed by modifying Ctt or Btt in the MMU translation table descriptor. This is not necessary if it is known that the caches cannot contain any entries from the area of memory whose translation table descriptor is being modified.
- Turning the ICache or DCache on, if its contents are no longer coherent.

Changing the FCSE PID in CP15 register 13 does not change the contents of the cache or memory, and does not affect the mapping between cache entries and physical memory locations. It only changes the mapping between ARM9TDMI addresses and cache entries. This means that changing the FCSE PID does not lead to any coherency issues. No cache cleaning or cache invalidation is required when the process FCSE PID is changed.

The software design must also consider that the pipelined design of the ARM9TDMI core means that it fetches three instructions ahead of the current execution point. So, for example, the three instructions following an MCR that invalidates the ICache, have already been read from the ICache before it is invalidated.

CACHE CLEANING WHEN LOCKDOWN IS IN USE

The *clean DCache single entry (using index)* and *clean and invalidate DCache entry (using index)* operations can leave the victim pointer set to the index value used by the operation. In some circumstances, if DCache locking is in use, this can leave the victim pointer in the locked region, leading to locked data being evicted from the cache. You can move the victim pointer outside the locked region by implementing the cache loop, enclosed by the reading and writing of the base and victim pointer:

```
MRC p15, 0, Rd, c9, c0, 0          ; Read D Cache Base into Rd
Index Clean or Index Clean and Invalidate loops
MCR p15, 0, Rd, c9, c0, 0          ; Write D Cache Base and Victim from Rd
```

Clean DCache single entry (using MVA) and *clean and invalidate DCache entry (using MVA)* operations do not move the victim pointer, so you do not have to reposition the victim pointer after using these operations.

IMPLEMENTATION NOTES

This section describes the behavior of the ARM920T implementation in areas that are architecturally unpredictable. For portability to other ARM implementations, software must not depend on this behavior.

A read from a noncacheable (NCB or NCNB) region that unexpectedly hits in the cache still reads the required data from the AMBA ASB interface. The contents of the cache are ignored, and the cache contents are not modified. This includes the read portion of a swap (`SWP` or `SWPB`) instruction.

A write to a noncacheable (NCB or NCNB) region that unexpectedly hits in the cache updates the cache and still cause an access on the AMBA ASB interface. This includes the write portion of a swap instruction.

There are two test interfaces to both the DCache and ICache:

- debug interface
- AMBA test interface.

PHYSICAL ADDRESS TAG RAM

The ARM920T implements a *Physical Address* (PA) TAG RAM in order to perform write-backs from the DCache.

A write-back occurs when dirty data, that is about to be overwritten by linefill data, comes from a memory region that is marked as a write-back region. This data is written back to main memory to maintain memory coherency.

NOTE

Dirty data is data that has been modified in the cache, but not updated in main memory.

When a line is written into the data cache, the PA TAG is written into the PA TAG RAM. If this line has to be written back to main memory, the PA TAG RAM is read and the physical address is used by the AMBA ASB interface to perform the write-back.

The PA TAG RAM array for a 16KB DCache comprises eight segments x 64 rows per segment x 26 bits per row. There are two test interfaces to the PA TAG RAM.

DRAIN WRITE BUFFER

You can drain the write buffer under software control, so thar further instructions are not excuted until the write buffer is drained, using the following methods:

- store to nonbufferable memory
- load from noncachable memory
- MCR drain write buffer:

```
MCR          p15, 0, Rd, c7, c10, 4
```

The write buffer is also drained before forming the following less controllable activities, which you must consider as implementation-defined:

- fetch from noncachable memory
- DCache linefill
- ICache linefill.

WAIT FOR INTERRUPT

You can place the ARM920T into a low power state by executing the CP15 MCR wait for interrupt:

```
MCR          p15, 0, Rd, c7, c0, 4
```

Execution of this MCR causes the write buffer to drain and the ARM920T is put into a state where it will resume execution of code after either an interrupt or a debug request. When the interrupt occurs the MCR instruction completes and the **FIQ** or **IRQ** handler is entered as normal. The return link in R14_fiq or R14_irq contains the address of the MCR instruction plus8, so that the normal instruction used for interrupt returns to the instruction following the MCR:

```
SUBS         pc, r14, #4
```

NOTES

5 MEMORY MANAGEMENT UNIT

ABOUT THE MMU

ARM920T implements an enhanced ARM architecture v4 MMU to provide translation and access permission checks for the instruction and data address ports of the ARM9TDMI. The MMU is controlled from a single set of two-level page tables stored in main memory, that are enabled by the M bit in CP15 register 1, providing a single address translation and protection scheme. You can independently lock and flush the instruction and data TLBs in the MMU.

The MMU features are:

- standard ARMv4 MMU mapping sizes, domains, and access protection scheme
- mapping sizes are 1MB (sections), 64KB (large pages), 4KB (small pages), and 1KB (tiny pages)
- access permissions for sections
- access permissions for large pages and small pages can be specified separately for each quarter of the page (these quarters are called subpages)
- domains implemented in hardware
- entry instruction TLB and 64 entry data TLB
- hardware page table walks
- round-robin replacement algorithm (also called cyclic)
- invalidate whole TLB, using CP15 register 8
- invalidate TLB entry, selected by MVA, using CP15 register 8
- independent lockdown of instruction TLB and data TLB, using CP15 register 10.

ACCESS PERMISSIONS AND DOMAINS

For large and small pages, access permissions are defined for each subpage (1KB for small pages, 16KB for large pages). Sections and tiny pages have a single set of access permissions.

All regions of memory have an associated domain. A domain is the primary access control mechanism for a region of memory. It defines the conditions necessary for an access to proceed. The domain determines if:

- the access permissions are used to qualify the access
- the access is unconditionally allowed to proceed
- the access is unconditionally aborted.

In the latter two cases, the access permission attributes are ignored.
There are 16 domains. These are configured using the domain access control register.

TRANSLATED ENTRIES

Each TLB caches 64 translated entries. During CPU memory accesses, the TLB provides the protection information to the access control logic.

If the TLB contains a translated entry for the MVA, the access control logic determines if access is permitted:

- if access is permitted and an off-chip access is required, the MMU outputs the appropriate physical address corresponding to the MVA
- if access is permitted and an off-chip access is not required, the cache services the access
- if access is not permitted, the MMU signals the CPU core to abort.

If a TLB misses (it does not contain an entry for the VA) the translation table walk hardware is invoked to retrieve the translation information from a translation table in physical memory. When retrieved, the translation information is written into the TLB, possibly overwriting an existing value.

The entry to be written is chosen by cycling sequentially through the TLB locations. To enable use of TLB locking features, you can specify the location to write using CP15 register 10, TLB lockdown.

When the MMU is turned off, as happens on reset, no address mapping occurs and all regions are marked as noncachable and nonbufferable. See *About the caches and write buffer on page 4-1*.

MMU PROGRAM ACCESSIBLE REGISTERS

Table 5-1 lists the CP15 registers that are used in conjunction with page table descriptors stored in memory to determine the operation of the MMU.

Table 5-1. CP15 Register Functions

Register	Number	Bits	Register description
Control register	1	M, A, S, R	Contains bits to enable the MMU (M bit), enable data address alignment checks (A bit), and to control the access protection scheme (S bit and R bit).
Translation table base register	2	[31:14]	Holds the physical address of the base of the translation table maintained in main memory. This base address must be on a 16KB boundary and is common to both TLBs.
Domain access control register	3	[31:0]	Comprises 16 2-bit fields. Each field defines the access control attributes for one of 16 domains (D15-D0).
Fault status register	5 (I and D)	[7:0]	Indicates the cause of a Data or Prefetch Abort, and the domain number of the aborted access, when an abort occurs. Bits 7:4 specify which of the 16 domains (D15-D0) was being accessed when a fault occurred. Bits 3:0 indicate the type of access being attempted. The value of all other bits is unpredictable. The encoding of these bits is shown in Table 5-9 on page 5-19.
Fault address register	6 (D)	[31:0]	Holds the VA associated with the access that caused the Data Abort. See Table 5-9 on page 5-19 for details of the address stored for each type of fault. You can use ARM9TDMI register 14 to determine the VA associated with a Prefetch Abort.
TLB operations register	8	[31:0]	You can write to this register to make the MMU perform TLB maintenance operations. These are either invalidating all the (unpreserved) entries in the TLB, or invalidating a specific entry.
TLB lockdown register	10 (I and D)	[31:20] and 0	Allows specific page table entries to be locked into the TLB and the TLB victim index to be read or written: – opcode 2 = 0x0 accesses the D TLB lockdown register – opcode 2 = 0x1 accesses the I TLB lockdown register. Locking entries in the TLB guarantees that accesses to the locked page or section can proceed without incurring the time penalty of a TLB miss. This allows the execution latency for time-critical pieces of code such as interrupt handlers to be minimized.

All the CP15 MMU registers, except register 8, contain state. You can read them using MRC instructions, and write them using MCR instructions. Registers 5 and 6 are also written by the MMU during a Data Abort. Writing to Register 8 causes the MMU to perform a TLB operation, to manipulate TLB entries. This register cannot be read. The Instruction TLB (I TLB) and Data TLB (D TLB) both have a copy of register 10. The opcode_2 field in the CP15 instruction is used to determine the one accessed.

CP15 is described in Chapter 2 *Programmer's Model*, with details of register formats and the coprocessor instructions you can use to access them.

ADDRESS TRANSLATION

The MMU translates VAs generated by the CPU core, and by CP15 register 13, into physical addresses to access external memory. It also derives and checks the access permission, using a TLB.

The MMU table walking hardware is used to add entries to the TLB. The translation information, that comprises both the address translation data and the access permission data, resides in a translation table located in physical memory. The MMU provides the logic for you to traverse this translation table and load entries into the TLB.

There are one or two stages in the hardware table walking, and permission checking, process. The number of stages depends on whether the address is marked as a section-mapped access or a page-mapped access.

There are three sizes of page-mapped accesses and one size of section-mapped access. The page-mapped accesses are for:

- large pages
- small pages
- tiny pages.

The translation process always starts out in the same way, with a level one fetch. A section-mapped access requires only a level one fetch, but a page-mapped access requires a subsequent level two fetch.

TRANSLATION TABLE BASE

The hardware translation process is initiated when the TLB does not contain a translation for the requested MVA. The Translation Table Base (TTB) register points to the base address of a table in physical memory that contains section or page descriptors, or both. The 14 low-order bits of the TTB register are set to zero on a read, and the table must reside on a 16KB boundary. Figure 5-1 shows the format of the TTB register.

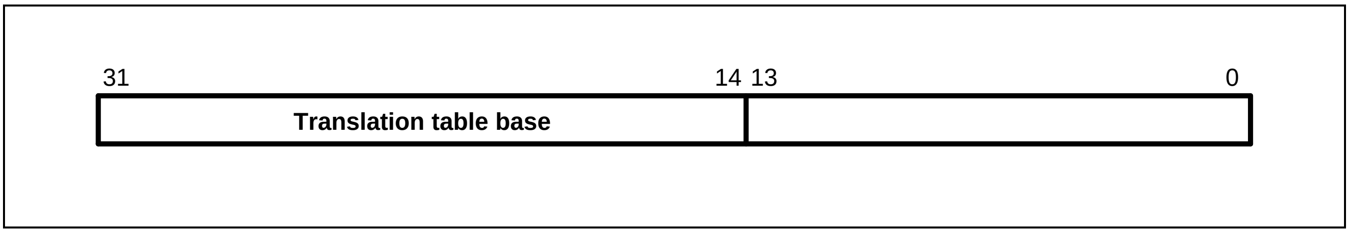


Figure 5-1. Translation Table Base Register

The translation table has up to 4096 x 32-bit entries, each describing 1MB of virtual memory. This allows up to 4GB of virtual memory to be addressed. Figure 5-2 shows the table walk process.

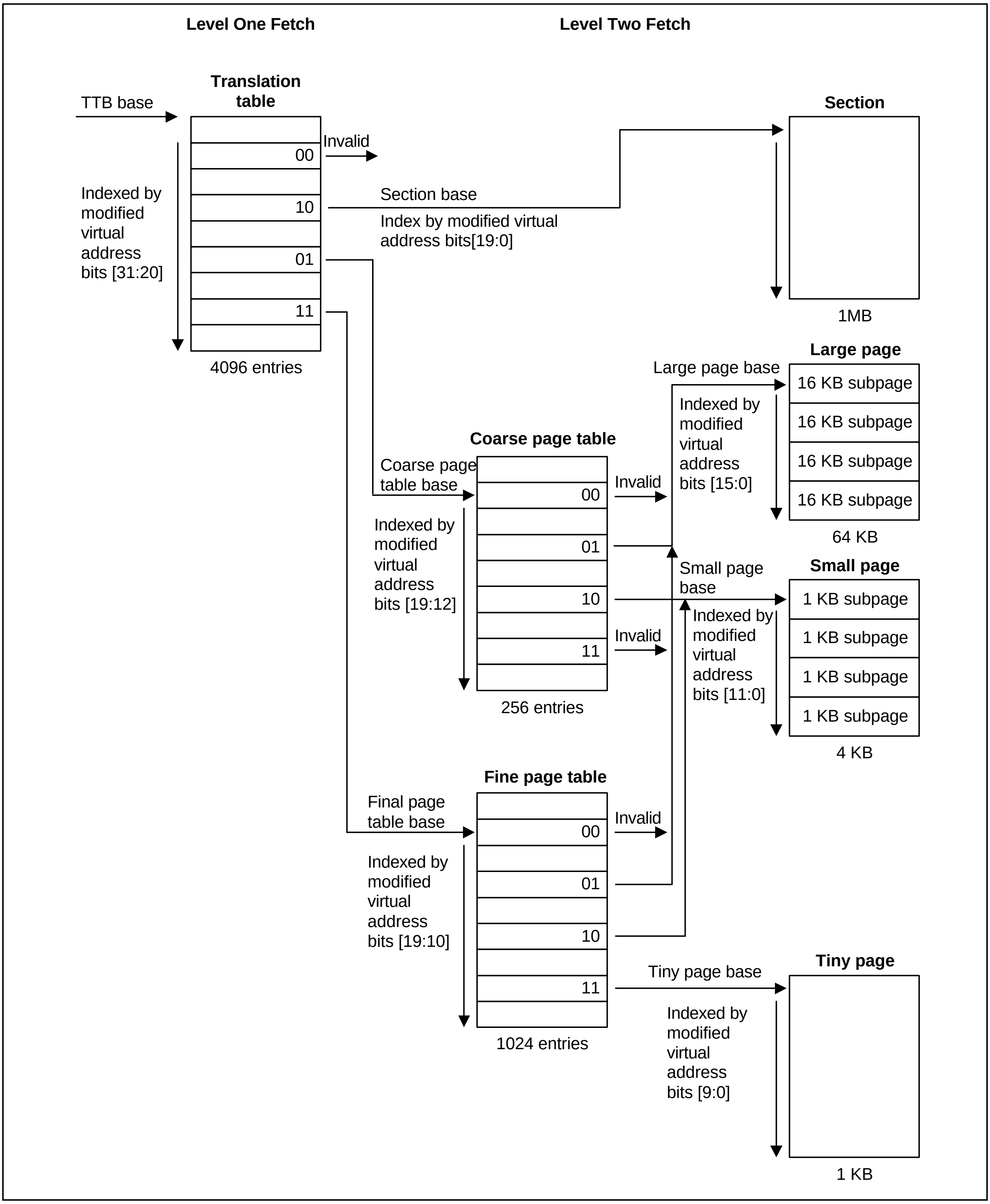


Figure 5-2. Translating Page Tables

LEVEL ONE FETCH

Bits [31:14] of the TTB register are concatenated with bits [31:20] of the MVA to produce a 30-bit address as shown in Figure 5-3.

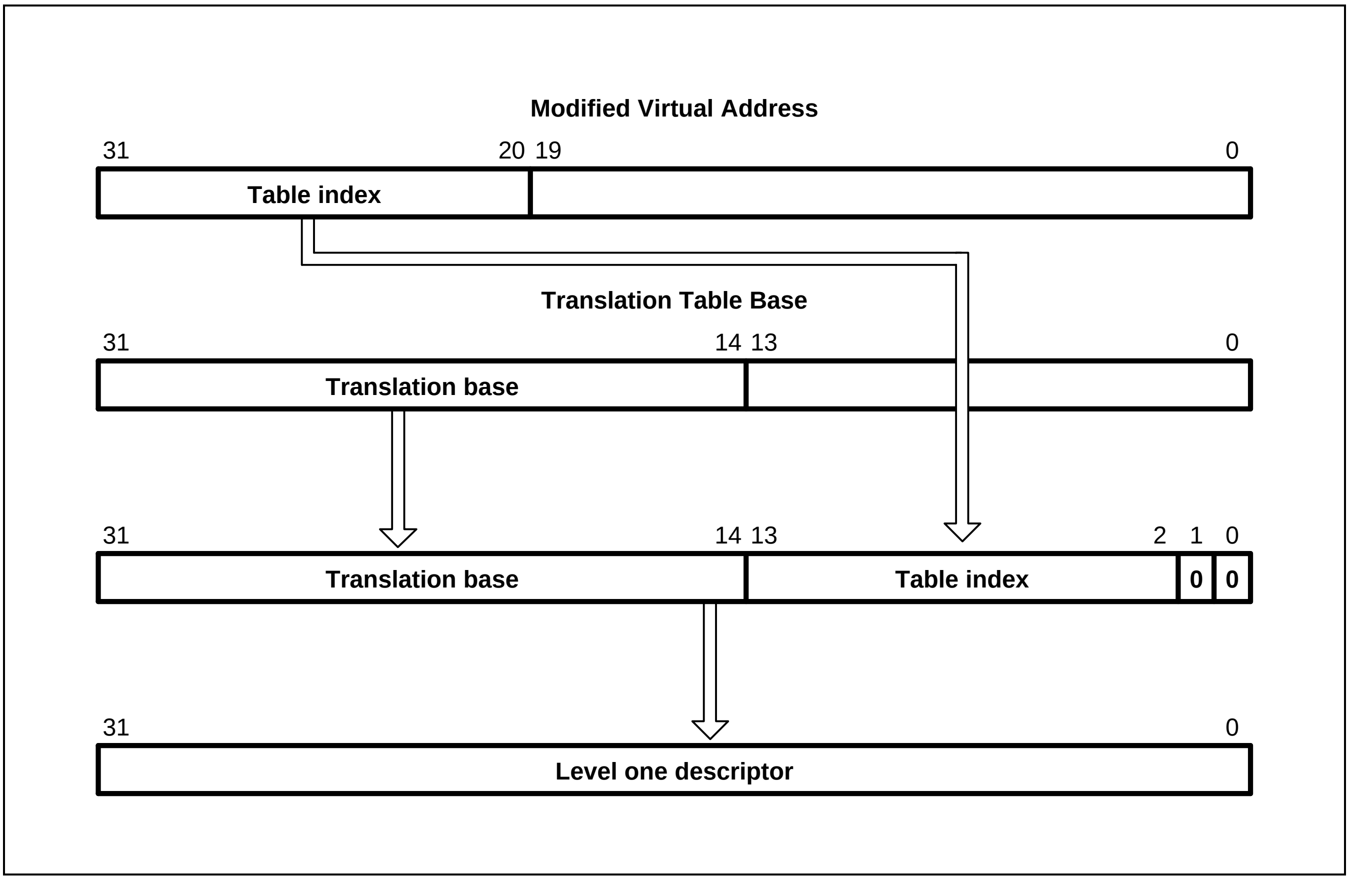


Figure 5-3. Accessing Translation Table Level One Descriptors

This address selects a 4-byte translation table entry. This is a level one descriptor for either a section or a page table.

LEVEL ONE DESCRIPTOR

The level one descriptor returned is either a section descriptor, a coarse page table descriptor, or a fine page table descriptor, or is invalid. Figure 5-4 shows the format of a level one descriptor.

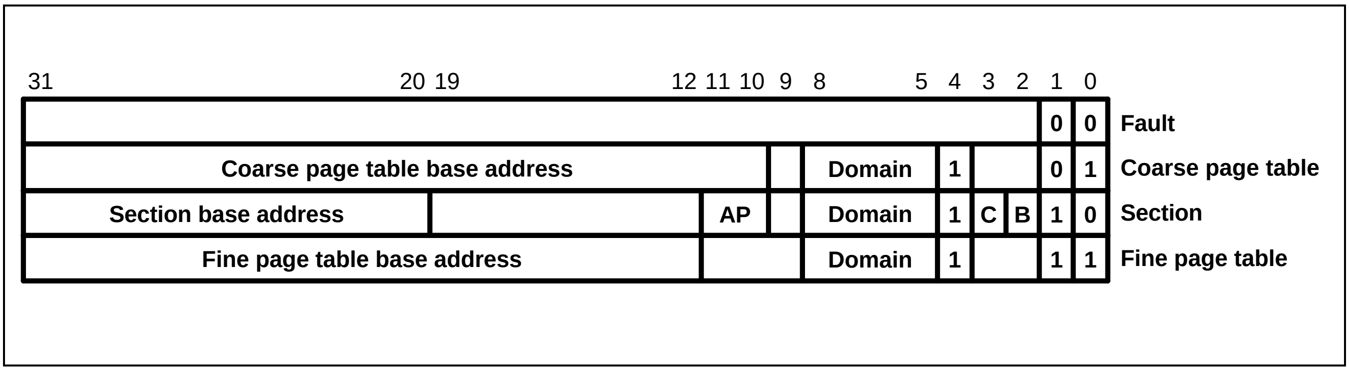


Figure 5-4. Level One Descriptor

A section descriptor provides the base address of a 1MB block of memory.

The page table descriptors provide the base address of a page table that contains level two descriptors. There are two sizes of page table:

- coarse page tables have 256 entries, splitting the 1MB that the table describes into 4KB blocks
- fine page tables have 1024 entries, splitting the 1MB that the table describes into 1KB blocks.

Level one descriptor bit assignments are shown in Table 5-2.

Table 5-2. Level One Descriptor Bits

Bits			Description
Section	Coarse	Fine	
[31:20]	[31:10]	[31:12]	These bits form the corresponding bits of the physical address
[19:12]	–	–	Should be zero
[11:10]	–	–	Access permission bits. *Domain access control on page 5-20* and *Fault checking sequence on page 5-21* show how to interpret the access permission bits
[9]	[9]	[11:9]	Should be zero
[8:5]	[8:5]	[8:5]	Domain control bits
[4]	[4]	[4]	Must be 1
[3:2]	–	–	These bits, C and B, indicate whether the area of memory mapped by this page is treated as write-back cacheable, write-through cacheable, noncached buffered, or noncached nonbuffered
–	[3:2]	[3:2]	Should be zero
[1:0]	[1:0]	[1:0]	These bits indicate the page size and validity and are interpreted as shown in Table 5-3

The two least significant bits of the level one descriptor indicate the descriptor type as shown in Table 5-3

Table 5-3. Interpreting Level One Descriptor Bits [1:0]

Value	Meaning	Description
0 0	Invalid	Generates a section translation fault
0 1	Coarse page table	Indicates that this is a coarse page table descriptor
1 0	Section	Indicates that this is a section descriptor
1 1	Fine page table	Indicates that this is a fine page table descriptor

SECTION DESCRIPTOR

A section descriptor provides the base address of a 1MB block of memory. Figure 5-5 shows the format of a section descriptor.

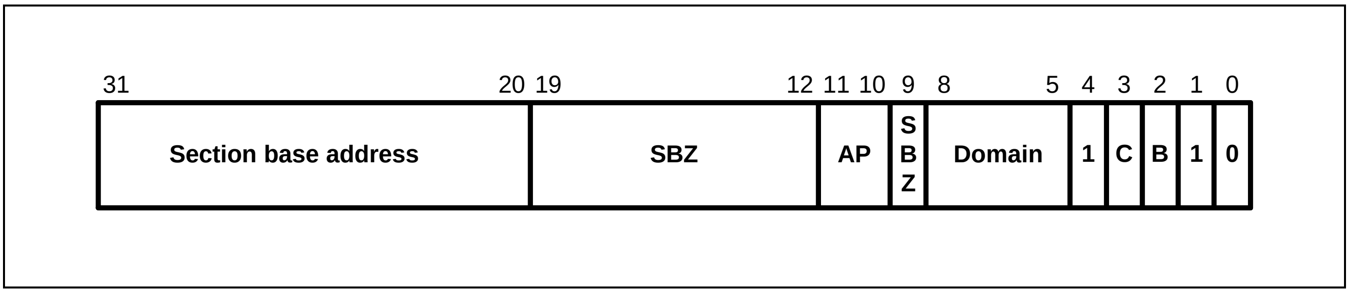


Figure 5-5. Section Descriptor

Section descriptor bit assignments are described in Table 5-4

Table 5-4. Section Descriptor Bits

Bits	Description
[31:20]	Form the corresponding bits of the physical address for a section
[19:12]	Always written as 0
[11:10]	(AP) Specify the access permissions for this section
[9]	Always written as 0
[8:5]	Specify one of the 16 possible domains (held in the domain access control register) that contain the primary access controls
[4]	Should be written as 1, for backward compatibility
[3:2]	These bits (C and B) indicate whether the area of memory mapped by this section is treated as write-back cacheable, write-through cacheable, noncached buffered, or noncached nonbuffered
[1:0]	These bits must be 10 to indicate a section descriptor

COARSE PAGE TABLE DESCRIPTOR

A coarse page table descriptor provides the base address of a page table that contains level two descriptors for either large page or small page accesses. Coarse page tables have 256 entries, splitting the 1MB that the table describes into 4KB blocks. Figure 5-6 shows the format of a coarse page table descriptor.

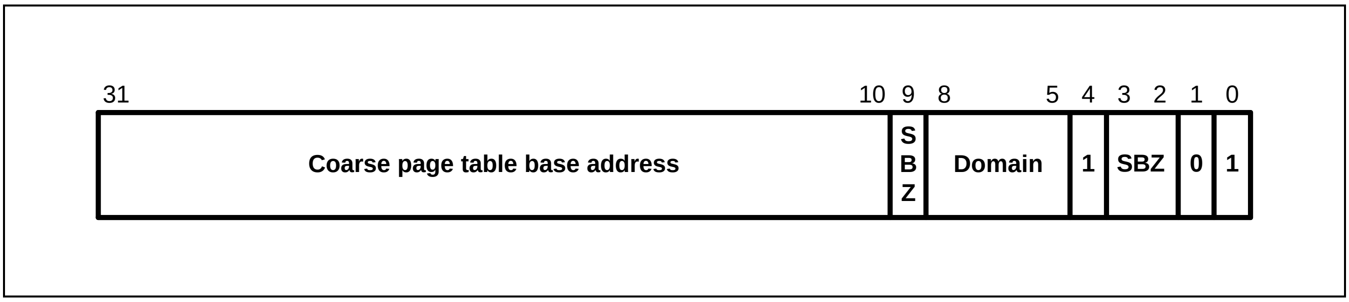


Figure 5-6. Coarse Page Table Descriptor

NOTE

If a coarse page table descriptor is returned from the level one fetch, a level two fetch is initiated.

Coarse page table descriptor bit assignments are described in Table 5-5.

Table 5-5. Coarse Page Table Descriptor Bits

Bits	Description
[31:10]	These bits form the base for referencing the level two descriptor (the coarse page table index for the entry is derived from the MVA)
[9]	Always written as 0
[8:5]	These bits specify one of the 16 possible domains (held in the domain access control registers) that contain the primary access controls
[4]	Always written as 1
[3:2]	Always written as 0
[1:0]	These bits must be 01 to indicate a coarse page table descriptor

FINE PAGE TABLE DESCRIPTOR

A fine page table descriptor provides the base address of a page table that contains level two descriptors for large page, small page, or tiny page accesses. Fine page tables have 1024 entries, splitting the 1MB that the table describes into 1KB blocks. Figure 5-7 shows the format of a fine page table descriptor.

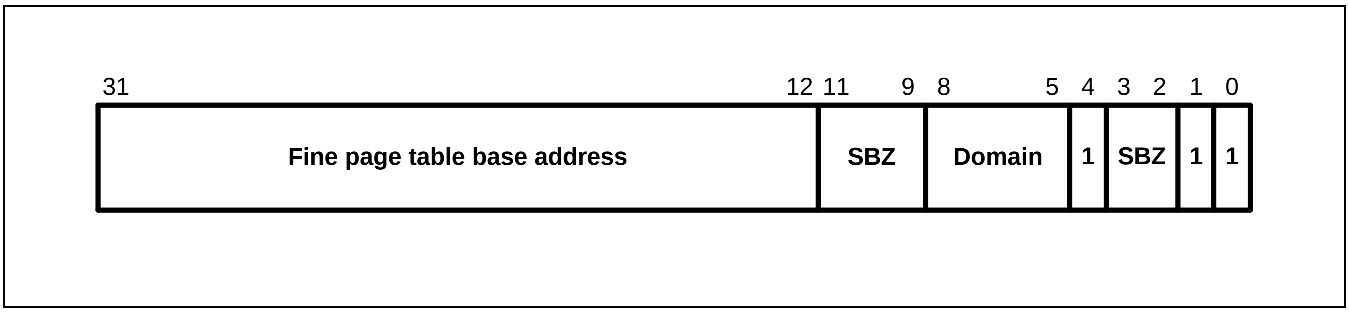


Figure 5-7. Fine Page Table Descriptor

NOTE

If a fine page table descriptor is returned from the level one fetch, a level two fetch is initiated.

Fine page table descriptor bit assignments are described in Table 5-6.

Table 5-6. Fine Page Table Descriptor Bits

Bits	Description
[31:12]	These bits form the base for referencing the level two descriptor (the fine page table index for the entry is derived from the MVA)
[11:9]	Always written as 0
[8:5]	These bits specify one of the 16 possible domains (held in the domain access control registers) that contain the primary access controls
[4]	Always written as 1
[3:2]	Always written as 0
[1:0]	These bits must be 11 to indicate a fine page table descriptor

TRANSLATING SECTION REFERENCES

Figure 5-8 shows the complete section translation sequence.

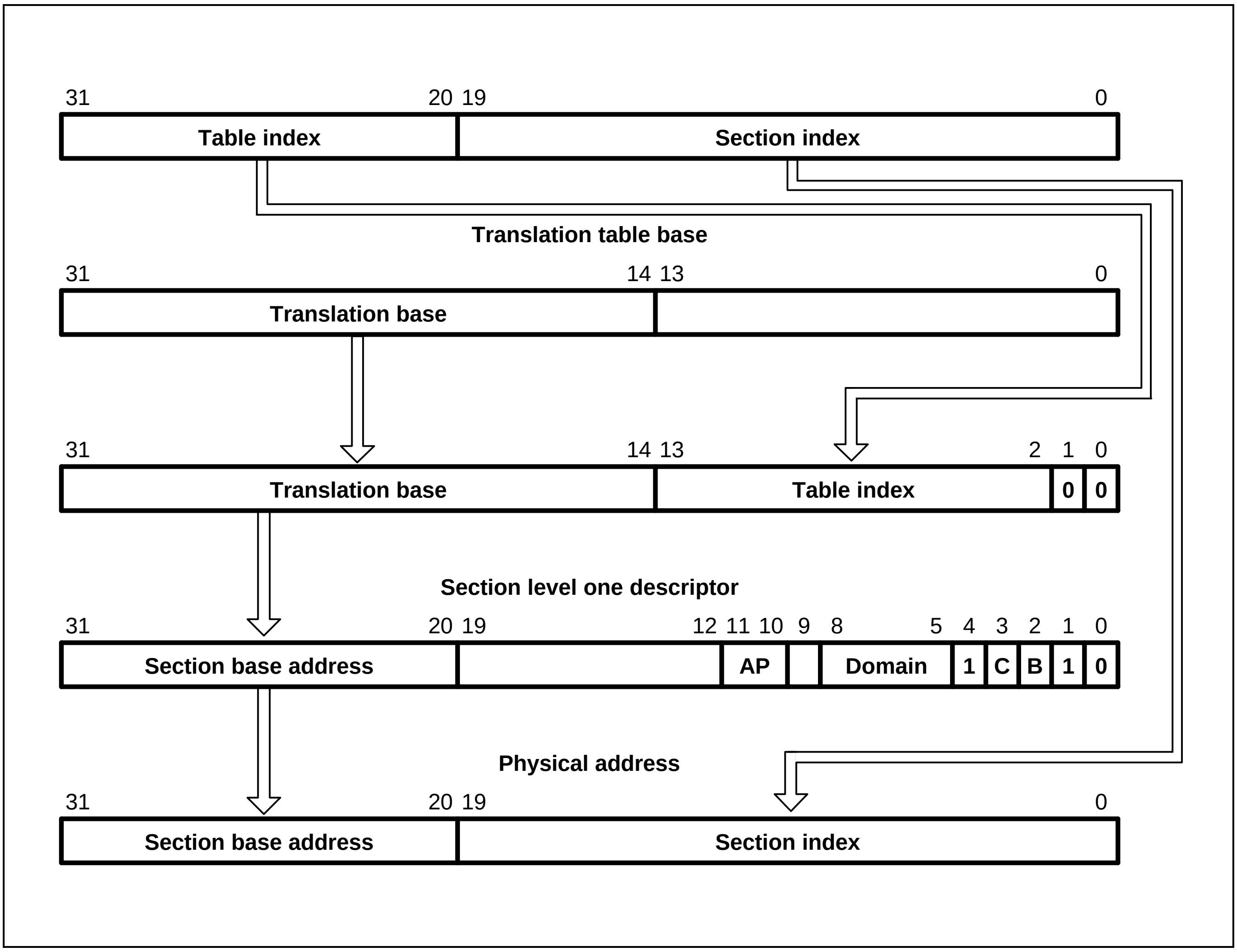


Figure 5-8. Section Translation

NOTE

You must check access permissions contained in the level one descriptor before generating the physical address.

LEVEL TWO DESCRIPTOR

If the level one fetch returns either a coarse page table descriptor or a fine page table descriptor, this provides the base address of the page table to be used. The page table is then accessed and a level two descriptor is returned. Figure 5-9 shows the format of level two descriptors.

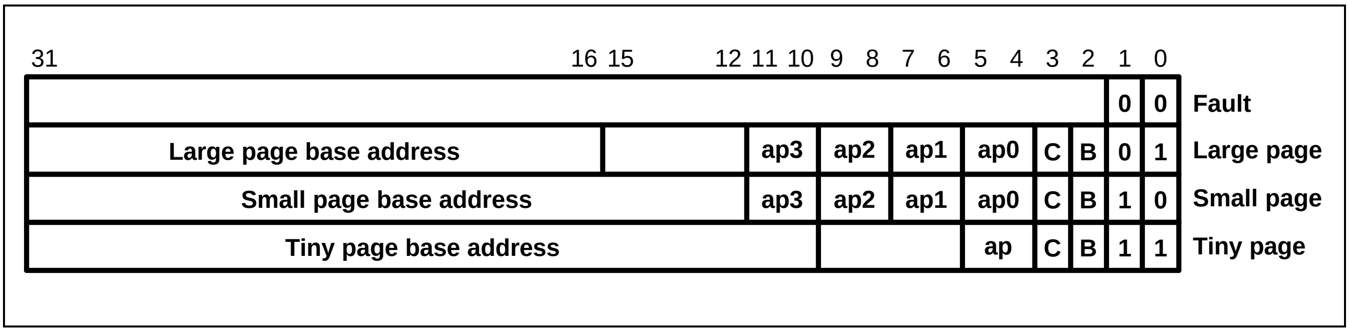


Figure 5-9. Level Two Descriptor

A level two descriptor defines a tiny, a small, or a large page descriptor, or is invalid:

- a large page descriptor provides the base address of a 64KB block of memory
- a small page descriptor provides the base address of a 4KB block of memory
- a tiny page descriptor provides the base address of a 1KB block of memory.

Coarse page tables provide base addresses for either small or large pages. Large page descriptors must be repeated in 16 consecutive entries. Small page descriptors must be repeated in each consecutive entry.

Fine page tables provide base addresses for large, small, or tiny pages. Large page descriptors must be repeated in 64 consecutive entries. Small page descriptors must be repeated in four consecutive entries and tiny page descriptors must be repeated in each consecutive entry.

Level two descriptor bit assignments are described in Table 5-7.

Table 5-7. Level Two Descriptor Bits

Bits			Description
Large	Small	Tiny	
[31:16]	[31:12]	[31:10]	These bits form the corresponding bits of the physical address
[15:12]	-	[9:6]	Should be zero
[11:4]	[11:4]	[5:4]	Access permission bits. *Domain access control* and *Fault checking sequence* show how to interpret the access permission bits
[3:2]	[3:2]	[3:2]	These bits, C and B, indicate whether the area of memory mapped by this page is treated as write-back cacheable, write-through cacheable, noncached buffered, or noncached nonbuffered
[1:0]	[1:0]	[1:0]	These bits indicate the page size and validity and are interpreted as shown in Table 5-8

The two least significant bits of the level two descriptor indicate the descriptor type as shown in Table 5-8.

Table 5-8. Interpreting Page Table Entry Bits [1:0]

Value	Meaning	Description
0 0	Invalid	Generates a page translation fault
0 1	Large page	Indicates that this is a 64KB page
1 0	Small page	Indicates that this is a 4KB page
1 1	Tiny page	Indicates that this is a 1KB page

NOTE

Tiny pages do not support subpage permissions and therefore only have one set of access permission bits.

TRANSLATING LARGE PAGE REFERENCES

Figure 5-10 shows the complete translation sequence for a 64KB large page.

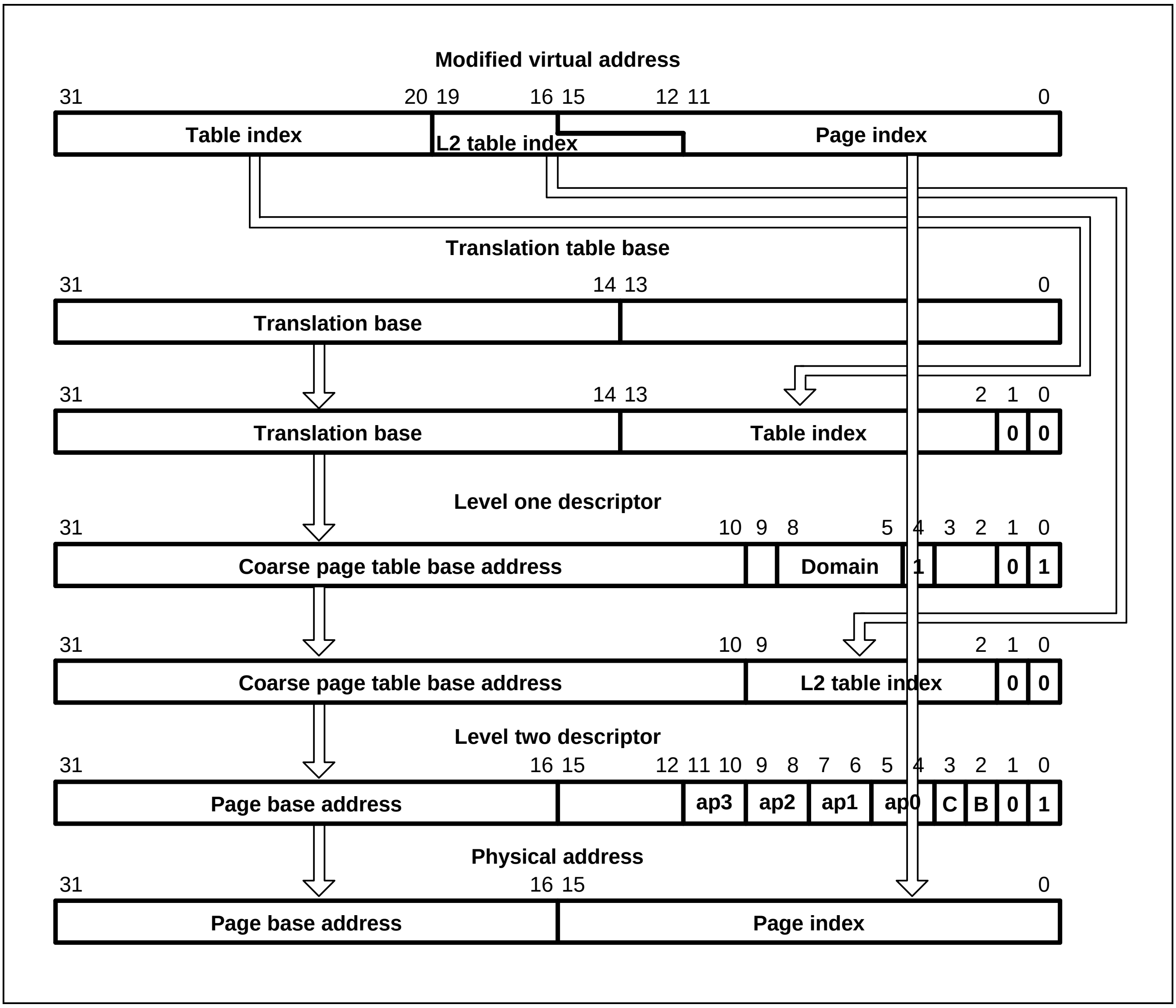


Figure 5-10. Large Page Translation from a Coarse Page Table

Because the upper four bits of the page index and low-order four bits of the coarse page table index overlap, each coarse page table entry for a large page must be duplicated 16 times (in consecutive memory locations) in the coarse page table.

If a large page descriptor is included in a fine page table, the high-order six bits of the page index and low-order six bits of the fine page table index overlap. Each fine page table entry for a large page must therefore be duplicated 64 times.

TRANSLATING SMALL PAGE REFERENCES

Figure 5-11 shows the complete translation sequence for a 4KB small page.

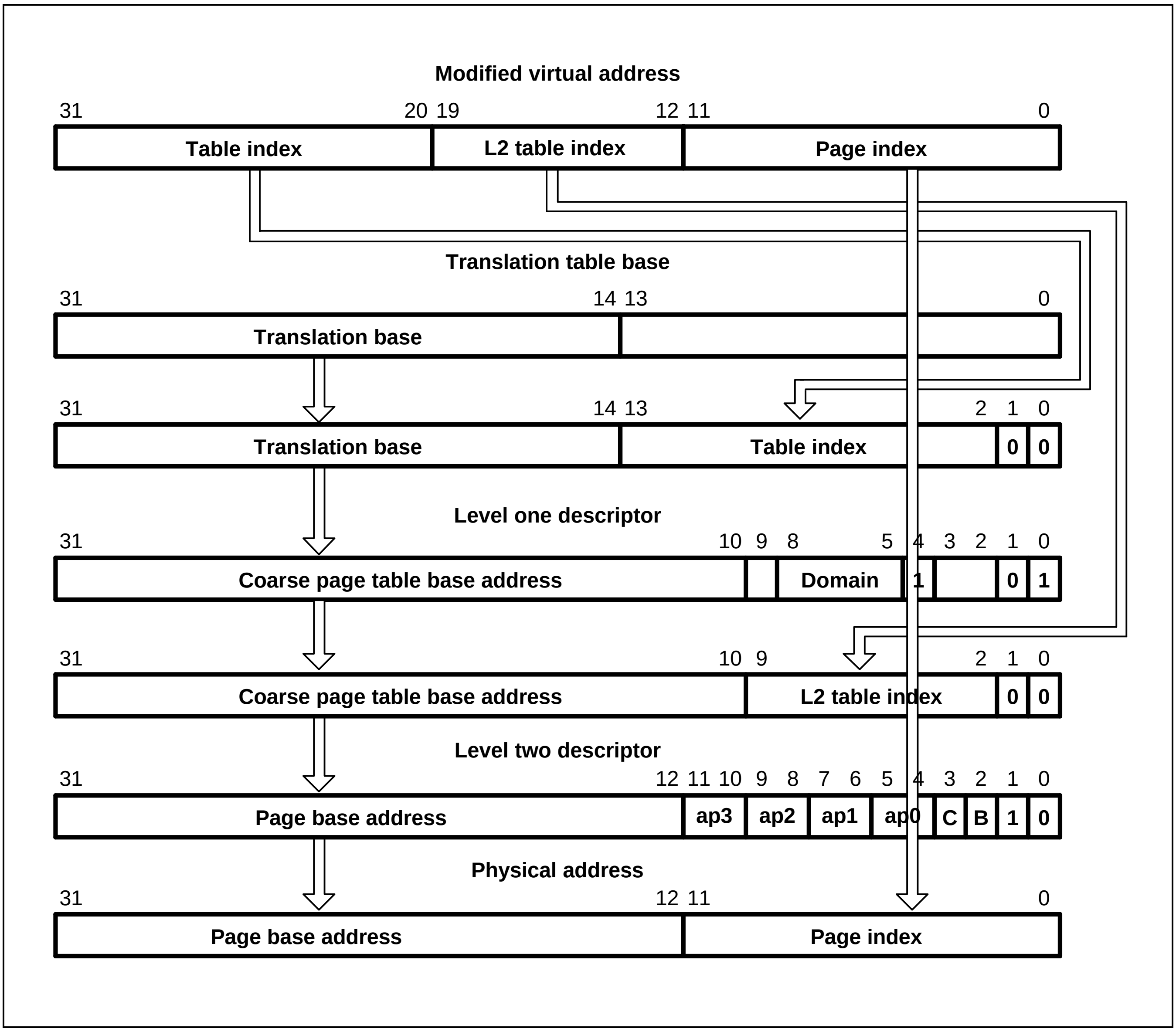


Figure 5-11. Small Page Translation from a Coarse Page Table

If a small page descriptor is included in a fine page table, the upper two bits of the page index and low-order two bits of the fine page table index overlap. Each fine page table entry for a small page must therefore be duplicated four times.

TRANSLATING TINY PAGE REFERENCES

Figure 5-12 shows the complete translation sequence for a 1KB tiny page.

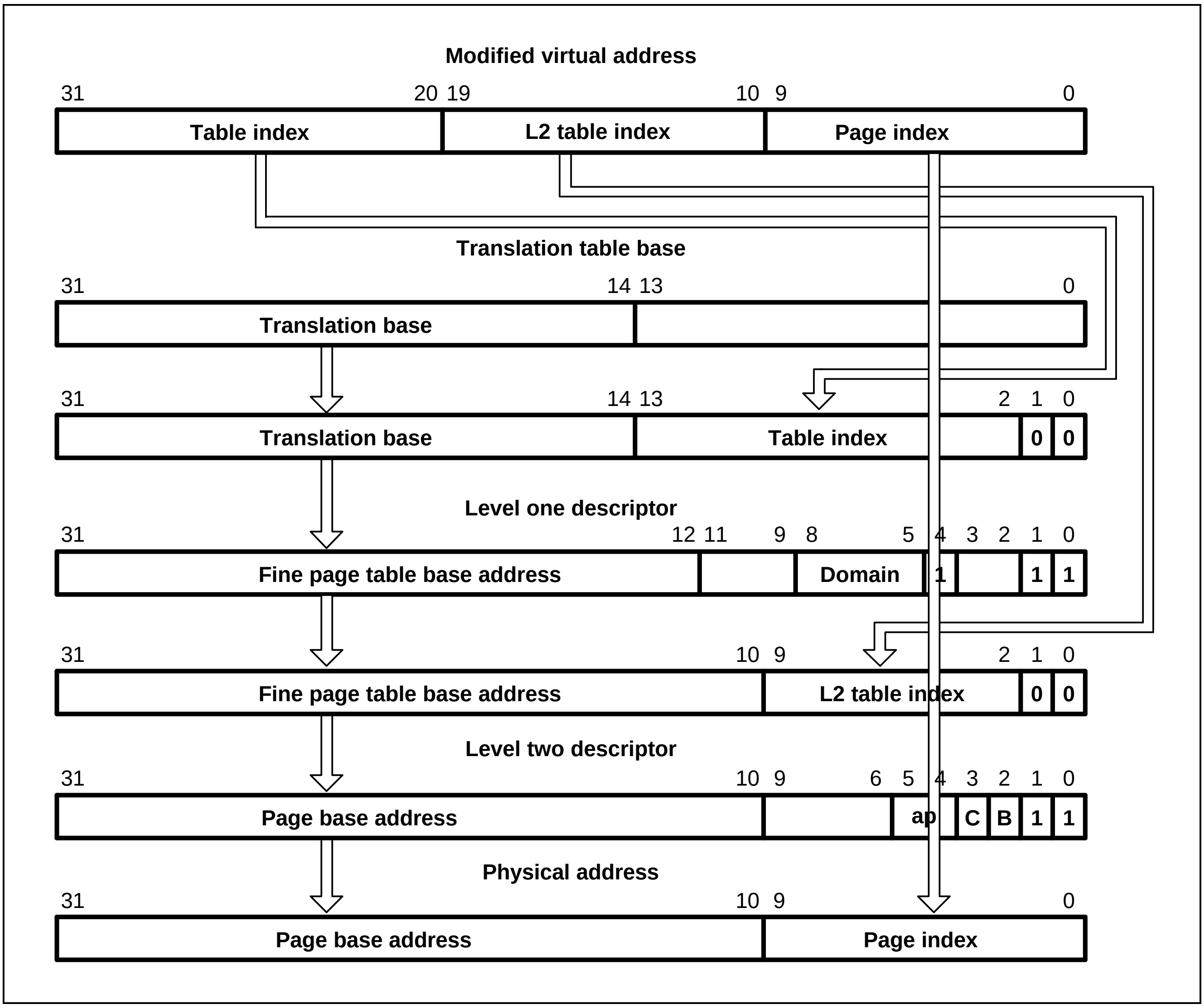


Figure 5-12. Tiny Page Translation from a Fine Page Table

Page translation involves one additional step beyond that of a section translation. The level one descriptor is the fine page table descriptor and this is used to point to the level one descriptor.

NOTE

The domain specified in the level one description and access permissions specified in the level one description together determine whether the access has permissions to proceed. See section *Domain access control* on page 5-20 for details.

SUBPAGES

You can define access permissions for subpages of small and large pages. If, during a page walk, a small or large page has a non-identical subpage permission, only the subpage being accessed is written into the TLB. For example, a 16KB (large page) subpage entry is written into the TLB if the subpage permission differs, and a 64KB entry is put in the TLB if the subpage permissions are identical.

When you use subpage permissions, and the page entry then has to be invalidated, you must invalidate all four subpages separately.

MMU FAULTS AND CPU ABORTS

The MMU generates an abort on the following types of faults:

- alignment faults (data accesses only)
- translation faults
- domain faults
- permission faults.

In addition, an external abort can be raised by the external system. This can happen only for access types that have the core synchronized to the external system:

- noncachable loads
- nonbufferable writes.

Alignment fault checking is enabled by the A bit in CP15 register 1. Alignment fault checking is not affected by whether or not the MMU is enabled. Translation, domain, and permission faults are only generated when the MMU is enabled.

The access control mechanisms of the MMU detect the conditions that produce these faults. If a fault is detected as a result of a memory access, the MMU aborts the access and signals the fault condition to the CPU core. The MMU retains status and address information about faults generated by the data accesses in the fault status register and fault address register (see *Fault address and fault status registers on page 5-19*). The MMU does not retain status about faults generated by instruction fetches.

An access violation for a given memory access inhibits any corresponding external access, with an abort returned to the CPU core.

FAULT ADDRESS AND FAULT STATUS REGISTERS

On a Data Abort, the MMU places an encoded 4-bit value, FS[3:0], along with the 4-bit encoded domain number, in the data FSR. Similarly, on a Prefetch Abort, in the prefetch FSR, intended for debug purposes only. In addition, the MVA associated with the Data Abort is latched into the FAR. If an access violation simultaneously generates more than one source of abort, they are encoded in the priority given in Table 5-9. The FAR is not updated by faults caused by instruction prefetches.

FAULT STATUS

Table 5-9 describes the various access permissions and controls supported by the data MMU and details how these are interpreted to generate faults.

Table 5-9. Priority Encoding of Fault Status

Priority	Source	Size	Status	Domain	FAR
Highest	Alignment	–	b00x1	Invalid	MVA of access causing abort
	Translation	Section Page	b0101 b0111	Invalid Valid	MVA of access causing abort
	Domain	Section Page	b1001 b1011	Valid Valid	MVA of access causing abort
	Permission	Section Page	b1101 b1111	Valid Valid	MVA of access causing abort
Lowest	External abort on noncachable nonbufferable access or noncachable bufferable read	Section Page	b1000 b1010	Valid Valid	MVA of access causing abort

NOTE

For data FSR only. Alignment faults can write either b0001 or b0011 into FS [3:0]. Invalid values in domains 3:0 can occur because the fault is raised before a valid domain field has been read from a page table descriptor. Any abort masked by the priority encoding can be regenerated by fixing the primary abort and restarting the instruction.
For instruction FSR only. The same priority applies as for the data FSR, except that alignment faults cannot occur, and external aborts apply only to noncachable reads.

DOMAIN ACCESS CONTROL

MMU accesses are primarily controlled through the use of domains. There are 16 domains and each has a 2-bit field to define access to it. Two types of user are supported, clients and managers. The domains are defined in the domain access control register. Figure 5-13 shows how the 32 bits of the register are allocated to define the 16 2-bit domains.

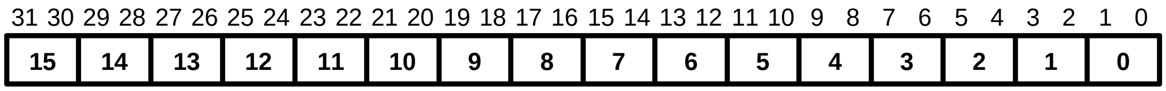


Figure 5-13 Domain Access Control Register Format

Table 5-10 defines how the bits within each domain are interpreted to specify the access permissions.

Table 5-10. Interpreting Access Control Bits in Domain Access Control Register

Value	Meaning	Description
0 0	No access	Any access generates a domain fault.
0 1	Client	Accesses are checked against the access permission bits in the section or page descriptor.
1 0	Reserved	Reserved. Currently behaves like the no access mode.
1 1	Manager	Accesses are not checked against the access permission bits so a permission fault cannot be generated.

Table 5-11 shows how to interpret the *Access Permission* (AP) bits and how their interpretation is dependent on the S and R bits (control register bits 8 and 9).

Table 5-11. Interpreting Access Permission (AP) Bits

AP	S	R	Supervisor Permissions	User Permissions	Description
00	0	0	No access	No access	Any access generates a permission fault
00	1	0	Read-only	No access	Only Supervisor read permitted
00	0	1	Read-only	Read-only	Any write generates a permission fault
00	1	1	Reserved	–	–
01	x	x	Read/write	No access	Access allowed only in Supervisor mode
10	x	x	Read/write	Read-only	Writes in User mode cause permission fault
11	x	x	Read/write	Read/write	All access types permitted in both modes
xx	1	1	Reserved	–	–

FAULT CHECKING SEQUENCE

The sequence the MMU uses to check for access faults is different for sections and pages. The sequence for both types of access is shown in Figure 5-14.

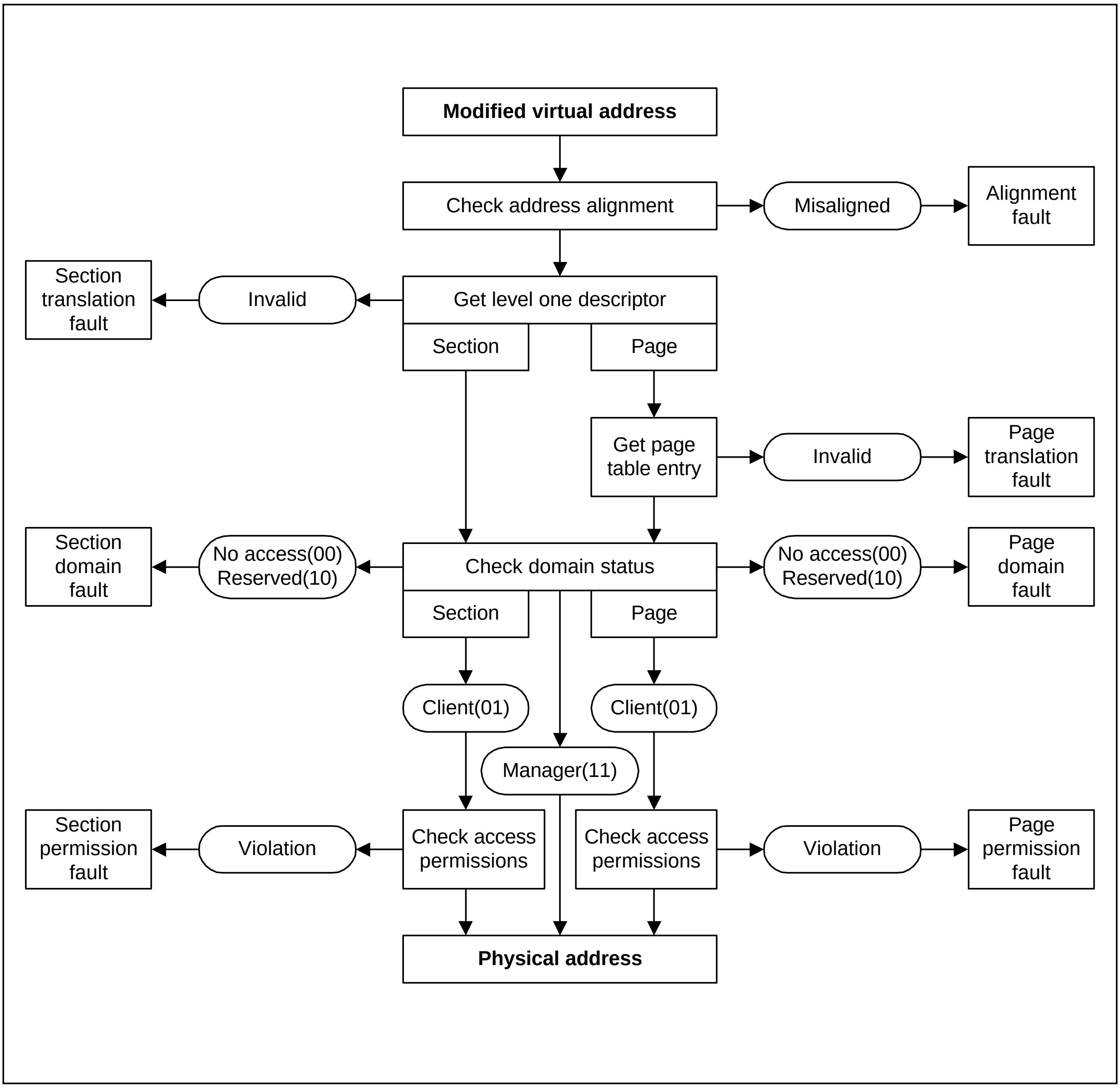


Figure 5-14. Sequence for Checking Faults

The conditions that generate each of the faults are described in:

- *Alignment fault*
- *Translation fault*
- *Domain fault*
- *Permission fault on page 5-23*

Alignment Fault

If alignment fault is enabled (A bit in CP15 register 1 set), the MMU generates an alignment fault on any data word access, if the address is not word-aligned, or on any halfword access, if the address is not halfword-aligned, irrespective of whether the MMU is enabled or not. An alignment fault is not generated on any instruction fetch, nor on any byte access.

NOTE

If the access generates an alignment fault, the access sequence aborts without reference to more permission checks.

Translation Fault

There are two types of translation fault:

Section A section translation fault is generated if the level one descriptor is marked as invalid. This happens if bits [1:0] of the descriptor are both 0.

Page A page translation fault is generated if the level one descriptor is marked as invalid. This happens if bits [1:0] of the descriptor are both 0.

Domain Fault

There are two types of domain fault:

Section The level one descriptor holds the 4-bit domain field, which selects one of the 16 2-bit domains in the domain access control register. The two bits of the specified domain are then checked for access permissions as described in Table 5-11. The domain is checked when the level one descriptor is returned.

Page The level one descriptor holds the 4-bit domain field, which selects one of the 16 2-bit domains in the domain access control register. The two bits of the specified domain are then checked for access permissions as described in Table 5-11. The domain is checked when the level one descriptor is returned.

If the specified access is either no access (00) or reserved (10) then either a section domain fault or page domain fault occurs.

Permission Fault

If the 2-bit domain field returns 01 (client) then access permissions are checked as follows:

Section If the level one descriptor defines a section-mapped access, the AP bits of the descriptor define whether or not the access is allowed, according to Table 5-11. Their interpretation is dependent on the setting of the S and R bits (control register bits 8 and 9). If the access is not allowed, a section permission fault is generated.

Large page or small page If the level one descriptor defines a page-mapped access and the level two descriptor is for a large or small page, four access permission fields (ap3-ap0) are specified, each corresponding to one quarter of the page. For small pages ap3 is selected by the top 1KB of the page and ap0 is selected by the bottom 1KB of the page. For large pages, ap3 is selected by the top 16KB of the page and ap0 is selected by the bottom 16KB of the page. The selected AP bits are then interpreted in exactly the same way as for a section (see Table 5-11). The only difference is that the fault generated is a page permission fault.

Tiny page If the level one descriptor defines a page-mapped access and the level two descriptor is for a tiny page, the AP bits of the level one descriptor define whether or not the access is allowed in the same way as for a section. The fault generated is a page permission fault.

EXTERNAL ABORTS

In addition to the MMU-generated aborts, the ARM920T can be externally aborted by the AMBA bus. This can be used to flag an error on an external memory access. However, not all accesses can be aborted in this way and the *Bus Interface Unit* (BIU) ignores external aborts that cannot be handled.

The following accesses can be aborted:

- noncached reads
- unbuffered writes
- read-lock-write sequence, to noncachable memory.

In the case of a read-lock-write (SWP) sequence, if the read aborts the write is always attempted.

INTERACTION OF THE MMU AND CACHES

The MMU is enabled and disabled using bit 0 of the CP15 control register as described in:

- Enabling the MMU
- Disabling the MMU.

Enabling the MMU

To enable the MMU:

1. Program the TTB and domain access control registers.
2. Program level 1 and level 2 page tables as required.
3. Enable the MMU by setting bit 0 in the control register.

You must take care if the translated address differs from the untranslated address because several instructions following the enabling of the MMU might have been prefetched with the MMU off (using physical = VA - flat translation).

In this case, enabling the MMU can be considered as a branch with delayed execution. A similar situation occurs when the MMU is disabled. Consider the following code sequence:

```
MRC          p15, 0, R1, c1, C0, 0: Read control rejection
ORR          R1, #0x1
MCR          p15,0,R1,C1, C0,0                           ; Enable MMUS
Fetch Flat
Fetch Flat
Fetch Translated
```

You can enable the ICache and DCache simultaneously with the MMU using a single MCR instruction.

Disabling the MMU

To disable the MMU, clear bit 0 in the control register. The data cache must be disabled prior to, or at the same time as, the MMU is disabled by clearing bit 2 of the control register. See Enabling the MMU regarding prefetch effects.

NOTE

If the MMU is enabled, then disabled and subsequently re-enabled, the contents of the TLBs are preserved. If these are now invalid, you must invalidate the TLBs before re-enabling the MMU. See *Register 8, TLB operations register on page 2-30*.

NOTES

6 CLOCK & POWER MANAGEMENT

OVERVIEW

The clock & power management unit consists of clock control, power control and reset control.

The clock control logic in S3C2800 generates various system clock signals: FCLK for CPU, HCLK for the AHB bus peripherals and PCLK for the APB bus peripherals. The clock control logic allows bypassing of PLL for slow clock and connection/disconnection of the clock to each peripheral block by software, which results in power reduction.

Also, S3C2800 has the power control logic to support various power management schemes for optimal power consumption for a given application. The power management provides three power down modes: NORMAL mode, SLOW mode, and IDLE mode.

In NORMAL mode clock is supplied to CPU as well as all peripherals in S3C2800. The power consumption will be a maximum when all peripherals are turned on. Also, user is allowed to control supply of the clock to peripherals by software. For example, if user does not need timer and DMA, user can disconnect the clock to timer and DMA to reduce the power consumption.

The SLOW mode is a non-PLL mode. Only difference to NORMAL mode is that the SLOW mode uses the external clock as a master clock in S3C2800 rather than the internal PLL clock. In this case, the power consumed by PLL itself is eliminated, and the power consumption will depend on the frequency of the external clock.

The IDLE mode disconnects the clock to CPU core while maintaining the clock to all peripherals. By using this IDLE mode, we can further reduce the power consumption by the CPU core. The wake-up from IDLE mode is done by an interrupt request to CPU.

The reset controller in S3C2800 consists of three reset types: hardware reset, software reset and watchdog reset. These types of reset are described in detail on page 6-9 *Reset Controller*.

FEATURE

- Input frequency range : 6MHz – 10MHz.
- Output frequency range : 20MHz – 200MHz.
- Programmable frequency divider
- Power management : Normal, Slow, and Idle.
- Reset controller : Hardware, Software, and Watchdog reset.

FUNCTION DESCRIPTION

CLOCK GENERATION

Figure 6-1 shows a block diagram of the clock generator. An external crystal clock is connected to the oscillation amplifier, and the PLL (Phase-Locked-Loop) converts the low input frequency into a high-frequency clock required by S3C2800. The clock generator block also has a built-in logic to stabilize the clock frequency after each system reset since the clock takes time before stabilized.

MAXIMUM BUS FREQUENCIES

Table 6-1 lists the maximum operating frequencies for the S3C2800. When selecting strap settings, make sure that the bus divider ratios do not result in the bus frequencies that exceed these maximums.

Table 6-1. Maximum Bus Frequencies

Internal Bus	Maximum Frequency	Module on the Internal Bus	Symbol
CPU	200MHz	CPU Core, I/D cache, R/W Buffer, MMU	FCLK
AHB	100MHz	DMA,Interrupt,Clock & Power, PCI, Memory controller	HCLK
APB	50MHz	IIC, GPIO, UART, Timer, Remote Signal Receive, RTC, Watchdog timer.	PCLK

Table 6-2. Examples of Maximum Bus Frequency

CPU Frequency(FCLK)	AHB Frequency(HCLK)	APB Frequency(PCLK)
200MHz	100MHz (= FCLK/2)	50MHz (= HCLK/2)
150MHz	75MHz (= FCLK/2)	37.5MHz (= HCLK/2)
100MHz	100MHz (= FCLK)	50MHz (= HCLK/2)
50MHz	50MHz (= FCLK)	50MHz (= HCLK)

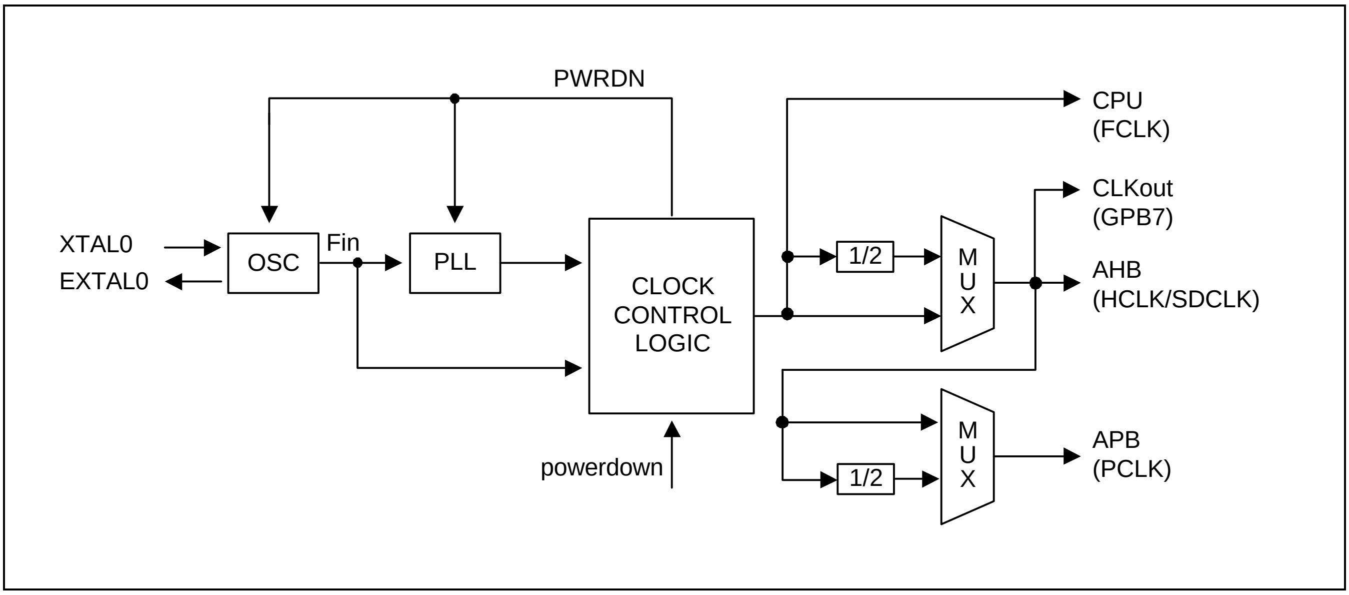


Figure 6-1. Clock Generator Block Diagram

NOTE: Until PLLCON register is configured for desired clock frequency by user, OSC clock (Fin) is supplied to the system.

PLL (PHASE LOCKED LOOP)

The PLL in the clock generator synchronizes the output signal with the input reference signal in terms of frequency as well as phase. The PLL is composed of the following basic blocks (Figure 6-2 shows the PLL block diagram): (a) the VCO(Voltage Controlled Oscillator) to generate the output frequency proportional to the input DC voltage, (b) the divider P to divide the reference frequency by p, (c) the divider M to divide the VCO output frequency by m which is input to PFD(Phase Frequency Detector), (d) the divider S to divide the VCO output frequency by s which is Fpllo(the output frequency from PLL block), (e) a phase detector, (f) a charge pump, and (g) a loop filter. The output clock frequency Fpllo is related to the reference input clock frequency Fin by the following equation:

Fpllo = (m * Fin) / (p * 2^s)
m = M (the value for divider M)+ 8, p = P(the value for divider P) + 2

The following sections describe the PLL operation that includes the phase detector, the charge pump, the VCO (Voltage controlled oscillator), and the loop filter.

Phase Detector

The phase detector monitors the phase difference between the Fref (the reference frequency as shown in Fig. 6-2) and Fvco (the output frequency from VCO and Divider M block), and generates a control signal(tracking signal) when it detects the difference between reference frequency and output frequency.

Charge Pump

The charge pump converts the phase detector control signal into a proportional charge across the external filter that drives the VCO.

Loop Filter

The control signal that the phase detector generates for the charge pump, may generate large excursions(ripples) each time the VCO output is compared to the system clock. To avoid overloading of the VCO, a low pass filter samples and filters the high-frequency components out of the control signal. The filter typically is a single-pole RC filter, consisting of a resistor and a capacitor.

A recommended capacitance in an external loop filter(Capacitance as shown in Figure 6-2) is 1µF.

Voltage Controlled Oscillator (VCO)

The output voltage from the loop filter drives the VCO, causing its oscillation frequency to increase or decrease linearly as a function of variations in average voltage. When the VCO output matches the system clock in terms of frequency as well as phase, the phase detector stops sending a control signal to the charge pump, which, in turn, stabilizes the input voltage to the loop filter. The VCO frequency then remains constant, and the PLL remains locked onto the system clock.

Usual Condition for PLL & Clock Generator

For proper operation of PLL, the following component value is recommended:

Loop filter capacitance	1 uF
External X-tal frequency	6 – 10 MHz

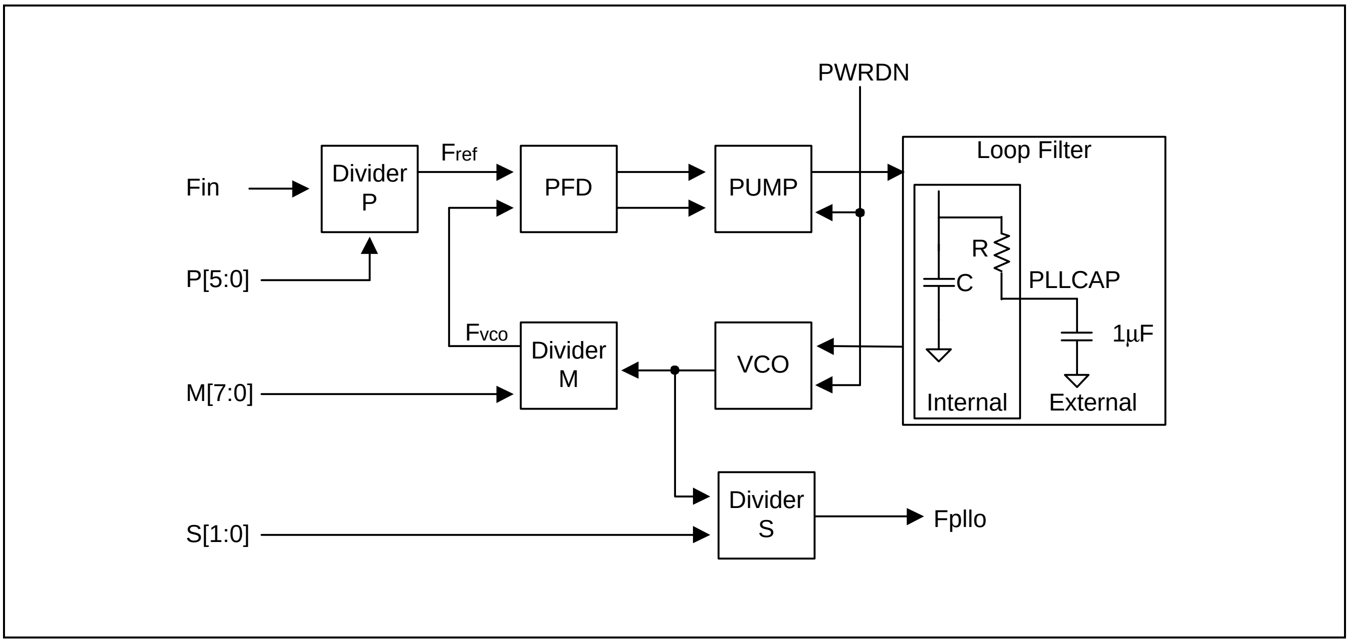


Figure 6-2. PLL (Phase-Locked Loop) Block Diagram

CLOCK CONTROL LOGIC

The clock control logic determines the clock source to be used, i.e., the PLL clock (Fpllo) or the direct OSC (Fin) clock. When PLL is configured to new frequency value, the clock control logic disables the FCLK until PLL output is stabilized using the PLL locking time. The clock control logic is also activated when the power-on reset and wake-up from power-down mode.

PLL Lock Time

The lock-time is the minimum time required for PLL output is stabilized. The lock time should be a minimum of 200µs. After reset, the lock time is inserted automatically by internal logic with lock time count register. The lock time is calculated as follows;

t_lock(the PLL lock time by hardware logic) = (1/ Fin) x n, (n = LOCKTIME register value)

Power-On Reset

Figure 6-3 shows the clock behavior during the power-on reset sequence. The crystal oscillator begins oscillation within several milliseconds. When nRESET is released after the stabilization of OSC (Fin) clock, the PLL starts to operate according to the default PLL configuration. However, PLL is commonly known to be unstable after power-on reset, so Fin fed directly to FCLK instead of the Fpllo (PLL output) before the software newly configures the PLLCON register.

The PLL begins the lockup sequence again toward the new frequency only after the software configures the PLL with a new frequency. FCLK can be configured to be PLL output (Fpllo) immediately after lock time.

NOTE

The internal power-on reset circuit in S3C2800 is designed to be activated when the 1.8V core voltage reaches a certain voltage level from Low 0V. Due to the filtering capacitance of power circuitry in the CPU board, there will be a case when the current in the capacitors will not be fully discharged if the power is turned on right after power-off. If this happens, the internal power-on reset circuit will not operate properly. Therefore, it is recommended to use a RESET IC on nRESET pin of S3C2800 when designing hardware.

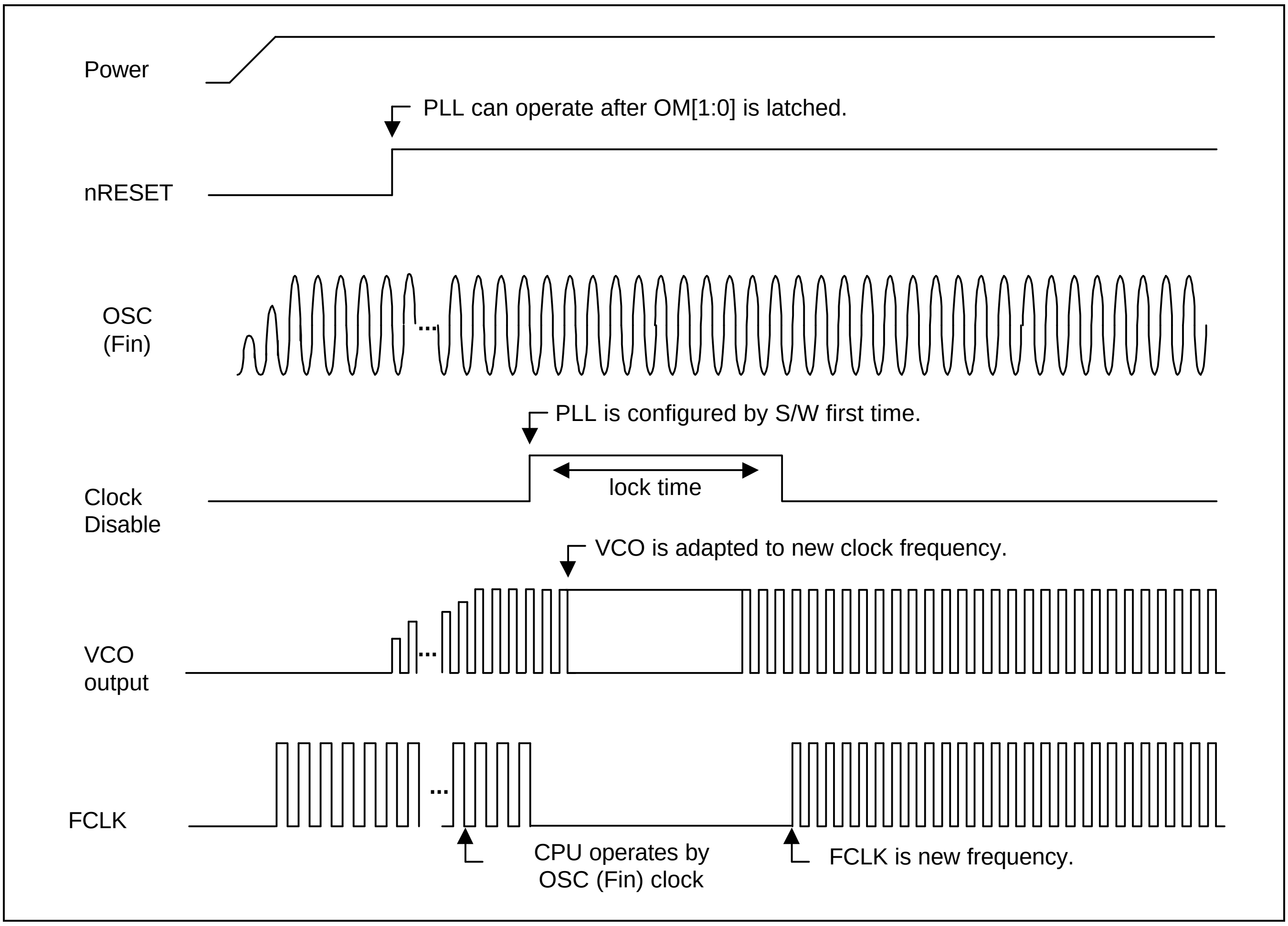

Power
PLL can operate after OM[1:0] is latched.
nRESET
OSC
(Fin)
PLL is configured by S/W first time.
Clock
Disable
lock time
VCO is adapted to new clock frequency.
VCO
output
FCLK
CPU operates by
OSC (Fin) clock
FCLK is new frequency.

Figure 6-3. Power-On Reset Sequence

Change PLL Settings in Normal Operation Mode

During the operation of S3C2800 in NORMAL mode, if users want to change the frequency by modifying PMS value, the PLL lock time is automatically inserted. During the lock time, the clock will not be supplied to internal blocks in S3C2800. The timing diagram is as follow.

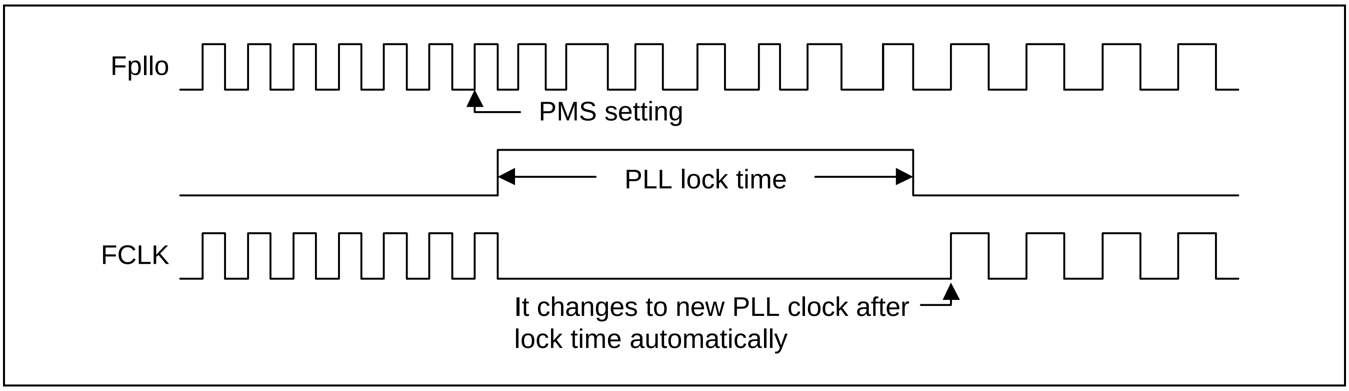


Figure 6-4. Timing Diagram of Clock Change in NORMAL Mode

POWER MANAGEMENT

The power management block allows the control of the system clocks by software, for the reduction of power consumption in S3C2800. These schemes are related to PLL, the clock control logic (FCLK, HCLK, PCLK), the wake-up signal.

S3C2800 has three power-down modes. The following section describes each power managing mode. The transition between the modes isn't allowed freely. For transitions among the modes, please refer to Figure 6-6.

NORMAL Mode

In NORMAL mode, All peripherals(UART, DMA, Timer, and so on) and the basic blocks(CPU core, bus controller, memory controller, interrupt controller, and power management block) may operate fully. But, the clock to each peripheral, except the basic blocks, can be stopped selectively by software to reduce power consumption.

IDLE Mode

In IDLE mode, the clock to CPU core is stopped except the bus controller, the memory controller, the interrupt controller, and the power management block. If CLKCON[2] is set to 1, S3C2800 enters into IDLE mode after some delay(Up to when the power control logic receives ACK signal from the CPU wrapper). To exit IDLE mode, EXTINT[7:0], RTC alarm interrupt, or the other interrupts should be activated. (If users want to use EXTINT[7:0], GPIO block has to be turned on before the activation).

INPORTANT NOTE

In order to use IDLE mode, the clock mode of ARM920T must be set for Asynchronous mode. If users want to use either FastBus or Synchronous mode, ARM920T clock mode must be switched to Asynchronous mode before entering IDLE mode, and then switched back to the previous mode (FastBus or Synchronous mode) after the wake-up operation. In other words, make sure that ARM920T is operating in Asynchronous mode prior to entering IDLE mode. The detailed information on ARM920T clock mode can be found in *ARM920T Technical Reference Manual*. For more information on clock mode setting, refer to *Table 2-13 and Table2-14* on pages 2-24 to 2-25.

SLOW Mode (non-PLL Mode)

In SLOW mode, the power consumption is decreased due to the slow system clock, and the power consumed by PLL itself is also eliminated. The Fout is the frequency of divide_by_n of the inpue clock (Fin) without PLL. The divider ratio is determined by SLOW_VAL in the CLKSLOW control register.

In SLOW mode, the PLL will be turned off to reduce the PLL power consumption. When PLL is turned off in SLOW mode and users change the power mode from SLOW mode to NORMAL mode, the PLL needs clock stabilization time (PLL lock time). This PLL stabilization time is automatically inserted by the internal logic with lock time count register. The PLL stability time will take 200µs after PLL is turned on. During PLL lock time, the FCLK is SLOW clock.

Users can change the frequency by enabling SLOW mode bit in CLKSLOW register. The SLOW clock is generated during SLOW mode. The timing diagram is as follows.

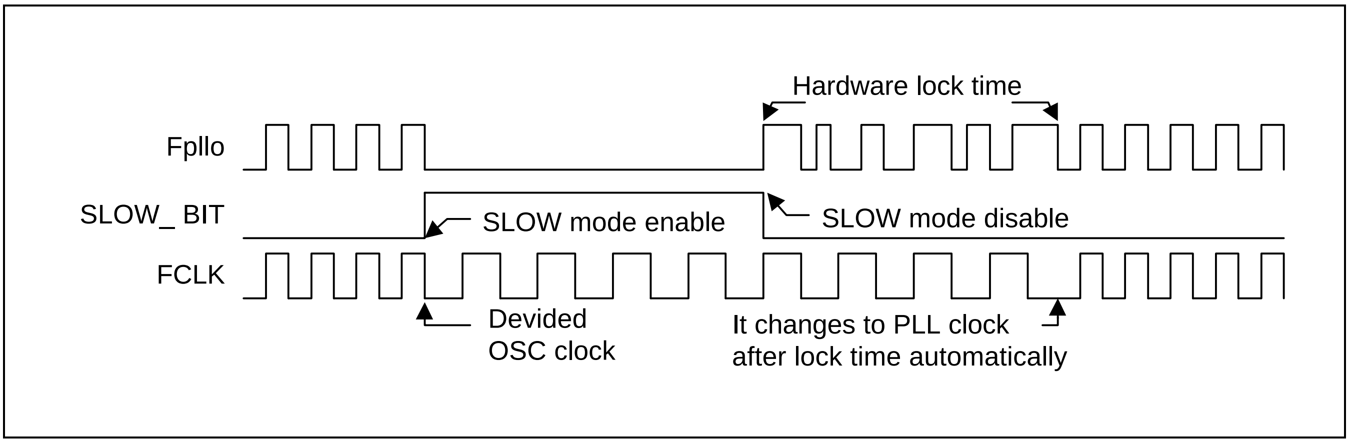


Figure 6-5. The Timing Diagram in Slow Mode

Power Management State Machine

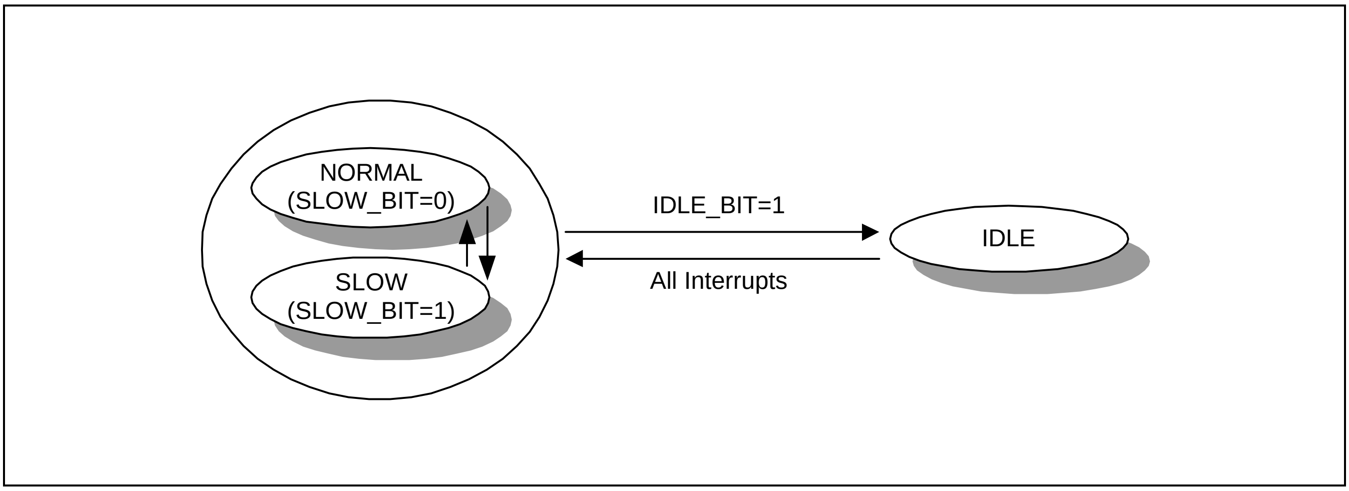


Figure 6-6. Power Management State Machine

RESET CONTROLLER

The reset controller manages the various reset sources in the S3C2800. For a programmer, two reset control registers are provided: one used to invoke software reset and one to read the status showing why the processor is reset after the reset sequence. After booting from the reset, software can examine the reset status register (RSTSR) to determine which types of reset has caused the reset condition.

Three types of reset in the S3C2800 are described below:

Hardware Reset

Hardware reset is invoked when the nRESET pin is asserted, and all units in the S3C2800 are initialized to a known state. Hardware reset is intended to be used for power-up only. Because the memory controller receives a full reset, all dynamic memory(DRAM/SDRAM) contents will be lost during hardware reset.

The nRESET_OUT pin is asserted during hardware reset.

Software Reset

Software reset is invoked when the software reset (SWR) bit in the SWRCON is set by software. After the SWR bit is set, the S3C2800 stays in reset state for 128 APB bus clocks (PCLK) and then is allowed to boot again.

The nRESET_OUT pin is asserted during software reset

Watchdog Reset

Watchdog reset is invoked when the watchdog enable bits in the WTCON[7:0] are set and the watchdog timer counter (WTCNT) overflows. The reset sequence of watchdog initiated reset is identical to software reset. When the WTCNT overflows, the S3C2800 stays in reset state for 128 APB bus clocks (PCLK) and then is allowed to boot again.

The nRESET_OUT pin is asserted during watchdog reset.

CLOCK AND POWER MANAGEMENT SPECIAL FUNCTION REGISTERS

PLL CONTROL REGISTER (PLLCON)

Fpllo = (m * Fin) / (p * 2^S)
m = (MDIV + 8), p = (PDIV + 2), s = SDIV

Table 6-3. Recommended Value of MDIV, PDIV, SDIV

Fin	6 MHz			8 MHz			10 MHz		
Fpllo	**MDIV**	**PDIV**	**SDIV**	**MDIV**	**PDIV**	**SDIV**	**MDIV**	**PDIV**	**SDIV**
200 MHz	0x5c	1	0	0x5c	2	0	0x5c	3	0
190 MHz	0x57	1	0	0x57	2	0	0x57	3	0
180 MHz	0x52	1	0	0x52	2	0	0x52	3	0
170 MHz	0x4d	1	0	0x4d	2	0	0x4d	3	0
160 MHz	0x48	1	0	0x48	2	0	0x48	3	0
150 MHz	0x43	1	0	0x43	2	0	0x43	3	0
140 MHz	0x3e	1	0	0x3e	2	0	0x3e	3	0
130 MHz	0x39	1	0	0x39	2	0	0x39	3	0
120 MHz	0x34	1	0	0x34	2	0	0x34	3	0
110 MHz	0x2f	1	0	0x2f	2	0	0x2f	3	0
100 MHz	0x5c	1	1	0x5c	2	1	0x5c	3	1
90 MHz	0x52	1	1	0x52	2	1	0x52	3	1
80 MHz	0x48	1	1	0x48	2	1	0x48	3	1
70 MHz	0x3e	1	1	0x3e	2	1	0x3e	3	1
60 MHz	0x34	1	1	0x34	2	1	0x34	3	1
50 MHz	0x2a	1	1	0x2a	2	1	0x2a	3	1
40 MHz	0x20	1	1	0x20	2	1	0x20	3	1
30 MHz	0x34	1	2	0x34	2	2	0x34	3	2
20 MHz	0x20	1	2	0x20	2	2	0x20	3	2

NOTE: This value may be calculated using PLLSET.EXE utility from Samsung. This PLL is not guaranteed that the PMS values are all zeros.

PLL Value Selection Guide (Must be)

1. (Fin/p) ≥ 2MHz , (p=PDIV + 2).
2. PDIV ≥ 1 ,(PDIV ≠ 0).
3. Fpllo * 2^S ≤ 300MHz,

Register	Address	R/W	Description	Reset Value
PLLCON	0x1000 0000	R/W	PLL configuration Register	Undefined

PLLCON	Bit	Description	Initial State
MDIV	[19:12]	Main divider control	Undefined
Reserved	[11:10]	Reserved	0
PDIV	[9:4]	Pre-divider control	Undefined
Reserved	[3:2]	Reserved	00
SDIV	[1:0]	Post divider control	Undefined

CLOCK CONTROL REGISTER (CLKCON)

Register	Address	R/W	Description	Reset Value
CLKCON	0x1000 0004	R/W	Clock control Register	0x0000 17FC

CLKCON	Bit	Description	Initial State
PCLK	[12]	APB clock division ratio from AHB clock 0 = HCLK 1 = HCLK/2	1
HCLK	[11]	AHB clock division ratio from CPU clock 0 = FCLK 1 = FCLK/2	0
PCI	[10]	Controls FCLK into PCI block 0 = Disable, 1 = Enable	1
IIC1	[9]	Controls PCLK into IIC0 block 0 = Disable, 1 = Enable	1
IIC0	[8]	Controls PCLK into IIC1 block 0 = Disable, 1 = Enable	1
RTC	[7]	Controls PCLK into RTC control block. Even if this bit is cleared to 0, RTC timer is alive. 0 = Disable, 1 = Enable	1
UART1	[6]	Controls PCLK into UART1 block 0 = Disable, 1 = Enable	1
UART0	[5]	Controls PCLK into UART0 block 0 = Disable, 1 = Enable	1
DMA2,3	[4]	Controls HCLK into DMA channel 2,3 block 0 = Disable, 1 = Enable (If DMA is turned off, the peripherals in the peripheral bus may not be accessed)	1
DMA0,1	[3]	Controls HCLK into DMA channel 0,1 block 0 = Disable, 1 = Enable (If DMA is turned off, the peripherals in the peripheral bus may not be accessed)	1
TIMER	[2]	Controls PCLK into TIMER block 0 = Disable, 1 = Enable	1
IDLE BIT	[1]	Enters IDLE mode. This bit cleared automatically by wake-up. 0 = Disable, 1 = Transition to IDLE mode	0
Reserved	[0]	Reserved	0

CLOCK SLOW CONTROL REGISTER (CLKSLOW)

Register	Address	R/W	Description	Reset Value
CLKSLOW	0x1000 0008	R/W	Slow clock control register	0x0000 0000

CLKSLOW	Bit	Description	Initial State

SLOW_BIT	[4]	Slow mode enable or Disable
		0 : Disable SLOW mode (NORMAL mode)
		FCLK = Fpllo (PLL output)
		1 : Enable SLOW mode (SLOW mode)
		FCLK = HCLK / (2 x SLOW_VAL), (SLOW_VAL > 0)
		FCLK = HCLK, (SLOW_VAL = 0)
			0x0

SLOW_VAL	[3:0]	The divider value for slow clock when SLOW_BIT is on.	0x0

LOCK TIME COUNT REGISTER (LOCKTIME)

Register	Address	R/W	Description	Reset Value
LOCKTIME	0x1000 000C	R/W	PLL lock time count register	0x0000 0FFF

LOCKTIME	Bit	Description	Initial State
LTIME CNT	[11:0]	PLL lock time count value	0xFFF

SOFTWARE RESET CONTROL REGISTER (SWRCON)

The software reset control register has a software reset bit, which when set, causes a reset of the S3C2800. The software-reset bit (SWR) is located within the least significant bit of the write-only software reset register (SWRCON). Writing a one to this bit causes all on-chip resources to reset but does not cause the PLL to go out of lock. The software reset bit is self-clearing. It is automatically cleared to zero after a few system clock cycles once it is set. Writing zero to the software reset bit has no effect. Care should be taken to restrict access to this register by programming MMU permissions.

The following table shows the SWRCON.

Register	Address	R/W	Description	Reset Value
SWRCON	0x1000 0010	W	Software reset control register	0x0000 0000

SWRCON	Bit	Description	Initial State
SWR	[0]	Software reset. 0 = Do not invoke a software reset of the chip. 1 = Invoke a software reset of the chip. This bit is self-clearing, and is automatically cleared several system clock cycles after it has been set.	0

RESET STATUS REGISTER (RSTSR)

To determine the last cause or causes of the reset, the CPU can refer to the reset status register (RSTSR). The S3C2800 has three sources of reset:

- Hardware reset
- Software reset
- Watchdog reset

Each RSTSR status bit is set by a different source of reset, and can be cleared by setting a one of the other reset status bits. Note that the hardware reset state of software and watchdog reset bit is zero.

The table below shows the status bits within RSTSR.

Register	Address	R/W	Description	Reset Value
RSTSR	0x1000 0014	R/W	Reset status register	0x0000 0001

RSTSR	Bit	Description	Initial State
WDR	[2]	Watchdog reset.(Read only) 0 = Watchdog reset has not occurred. 1 = Watchdog reset has occurred This bit is cleared automatically when one of the other reset status bit is set.	0
SWR	[1]	Software reset.(Read only) 0 = Software reset has not occurred. 1 = Software reset has occurred This bit is cleared automatically when one of the other reset status bit is set.	0
HWR	[0]	Hardware reset.(Read only) 0 = Hardware reset has not occurred. 1 = Hardware reset has occurred This bit is cleared automatically when one of the other reset status bit is set.	1

7 MEMORY CONTROLLER

OVERVIEW

The S3C2800 memory controller provides the necessary memory control signals for external memory access. Some of the main features include;

- Up to 100MHz interface (HCLK = SDCLK)
- Little-/Big-endian addressing mode support for external memory
- Up to 256MB of external memory space:
  - – Up to 128MB for Static Memory
  - – Up to 128MB for Dynamic Memory
- Programmable data width (8/16/32-bit) for all banks
- Total 8 memory banks:
  - – 4 memory banks for ROM, SRAM ,Flash
  - – 4 memory banks for FP/EDO/SDRAM etc
- Fixed memory bank start address
- Programmable access cycles for each static memory banks
- External wait for extending the bus cycles
- Supports self-refresh mode support for DRAM/SDRAM power-down mode
- Asymmetric or symmetric addressable DRAM support

MEMORY MAP

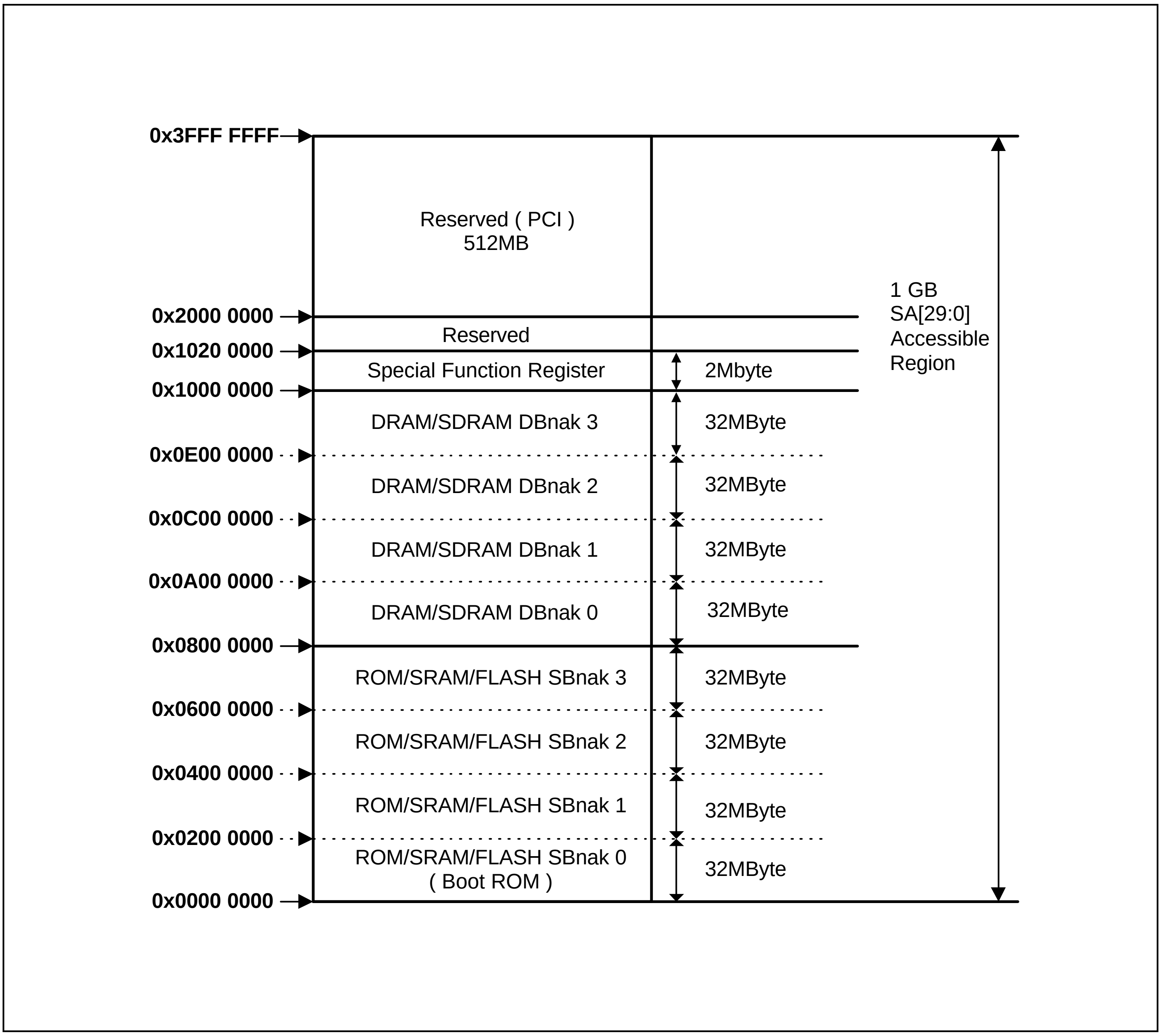


Figure 7-1. Memory Map after Reset

FUNCTION DESCRIPTION

LITTLE-/BIG-ENDIAN SELECTION (FOR EXTERNAL MEMORY ONLY)

While nRESET is Low, the ENDIAN (GPC3) pin determines the endian mode of the external memory. If the ENDIAN pin is connected to V_{SS} with pull-down resistor, the Little-endian mode is selected. If the pin is connected to V_{DD} with a pull-up resistor, the Big-endian mode is selected.

ENDIAN Input @Reset	ENDIAN Mode
0	Little-endian
1	Big-endian

BANK0 BUS WIDTH (FOR STATIC MEMORY)

The data bus width of boot ROM bank, BANK0, must be configured before the first access to ROM. The data bus width a 8-, 16-, and 32-bit is determined by logic level of OM[1:0] at reset.

OM1 (Operating Mode 1)	OM0 (Operating Mode 0)	Booting ROM Data width
0	0	8-bit
0	1	16-bit
1	0	32-bit
1	1	Not used

MEMORY ADDRESS PIN CONNECTIONS

MEMORY ADDR. PIN	S3C2800 ADDR. @ 8-bit DATA BUS	S3C2800 ADDR. @ 16-bit DATA BUS	S3C2800 ADDR. @ 32-bit DATA BUS
A0	ADDR0	ADDR1	ADDR2
A1	ADDR1	ADDR2	ADDR3
A2	ADDR2	ADDR3	ADDR4
A3	ADDR3	ADDR4	ADDR5
. . .	. . .	. . .	. . .

SDRAM BANK ADDRESS PIN CONNECTION

Table 7-1. SDRAM Bank Address Configuration (Example)

Bank Size	Bus Width	Base Component	Memory Configuration	Bank Address
2MB	x8	16Mb	(1Mb x 8 x 2B) x 1	ADDR[20]
	x16		(512Kb x 16 x 2B) x 1	
4MB	x8	16Mb	(2Mb x 4 x 2B) x 2	ADDR[21]
	x16		(1Mb x 8 x 2B) x 2	
	x32		(512Kb x 16 x 2B) x 2	
8MB	x16	16Mb	(2Mb x 4 x 2B) x 4	ADDR[A22]
	x32		(1Mb x 8 x 2B) x 4	
	x8	64Mb	(4Mb x 8 x 2B) x 1	ADDR[A22]
	x8		(2Mb x 8 x 4B) x 1	ADDR[22:21]
	x16		(2Mb x 16 x 2B) x 1	ADDR[22]
	x16		(1Mb x 16 x 4B) x 1	ADDR[22:21]
	x32		(512Kb x 32 x 4B) x 1	
16MB	x32	16Mb	(2Mb x 4 x 2B) x 8	ADDR[23]
	x8	64Mb	(8Mb x 4 x 2B) x 2	ADDR[23]
	x8		(4Mb x 4 x 4B) x 2	ADDR[23:22]
	x16		(4Mb x 8 x 2B) x 2	ADDR[23]
	x16		(2Mb x 8 x 4B) x 2	ADDR[23:22]
	x32		(2Mb x 16 x 2B) x 2	ADDR[23]
	x32		(1Mb x 16 x 4B) x 2	ADDR[23:22]
	x8	128Mb	(4Mb x 8 x 4B) x 1	ADDR[23:22]
	x16		(2Mb x 16 x 4B) x 1	
32MB	x16	64Mb	(8Mb x 4 x 2B) x 4	ADDR[24]
	x16		(4Mb x 4 x 4B) x 4	ADDR[24:23]
	x32		(4Mb x 8 x 2B) x 4	ADDR[24]
	x32		(2Mb x 8 x 4B) x 4	ADDR[24:23]
	x16	128Mb	(4Mb x 8 x 4B) x 2	ADDR[24:23]
	x32		(2Mb x 16 x 4B) x 2	
	x8	256Mb	(8Mb x 8 x 4B) x 1	ADDR[24:23]
	x16		(4Mb x 16 x 4B) x 1	

NOTE: MB=Mega Byte, Mb=Mega bits, Kb=Killo bits, B=Banks

ROM & SRAM MEMORY INTERFACE EXAMPLE

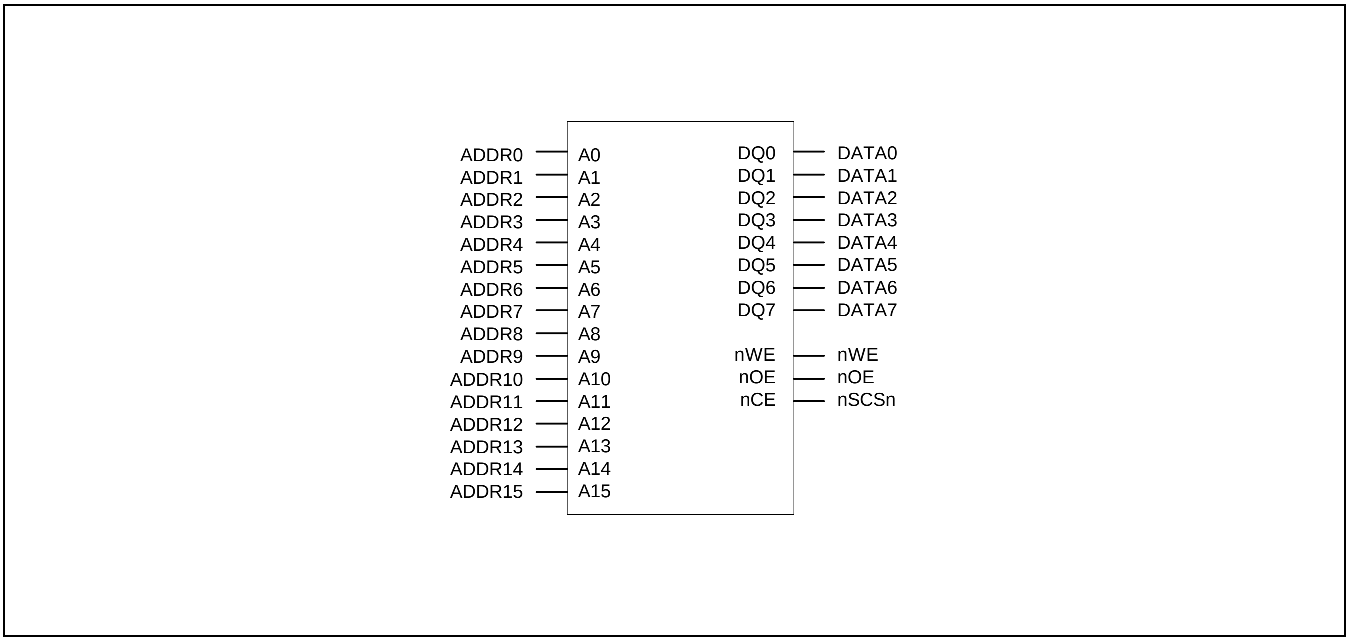


Figure 7-2. Memory Interface with 8bit ROM

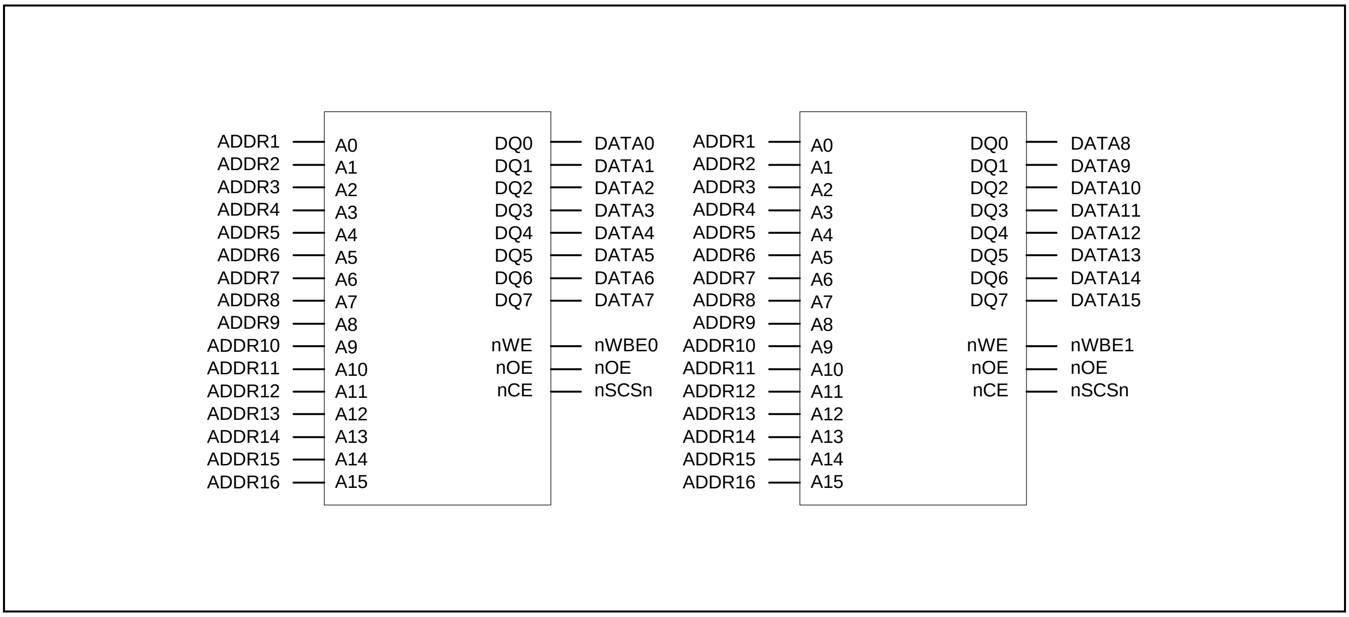


Figure 7-3. Memory Interface with 8bit ROM x 2

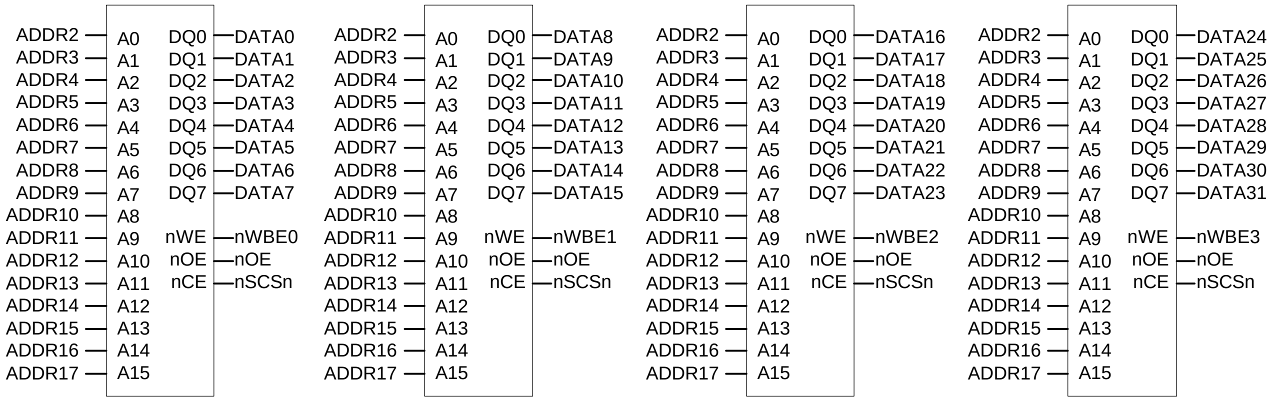


Figure 7-4. Memory Interface with 8bit ROM x 4

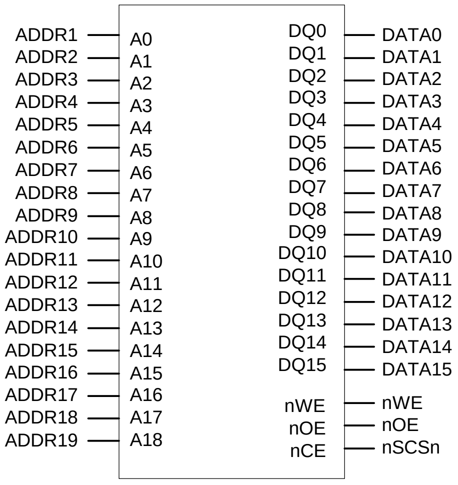


Figure 7-5. Memory Interface with 16bit ROM

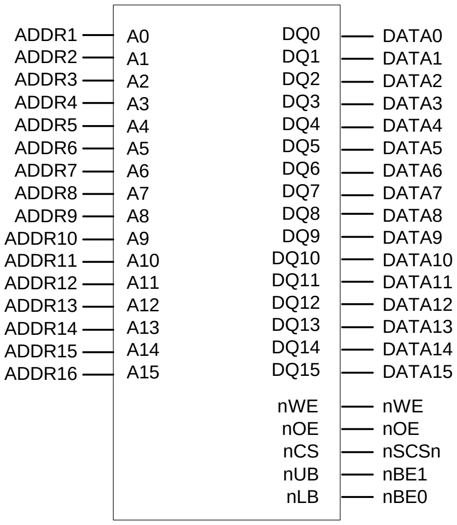


Figure 7-6. Memory Interface with 16bit SRAM

DRAM MEMORY INTERFACE EXAMPLE

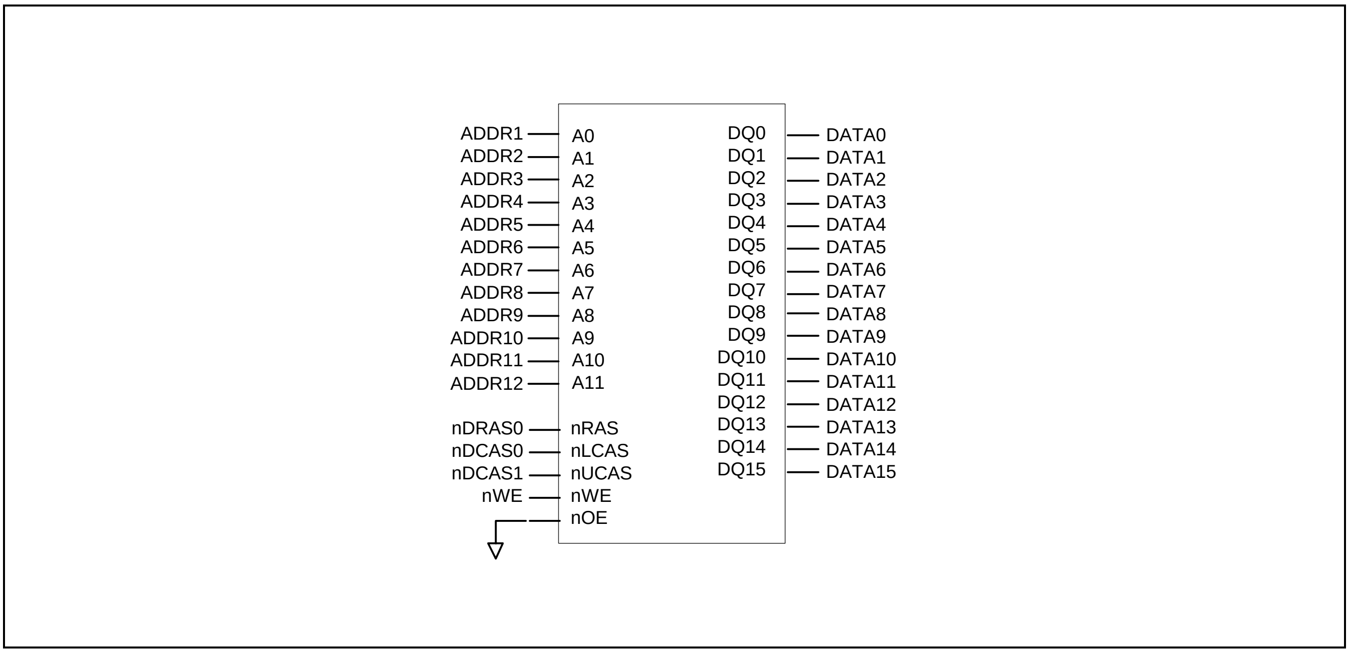


Figure 7-7. Memory Interface with 16bit DRAM

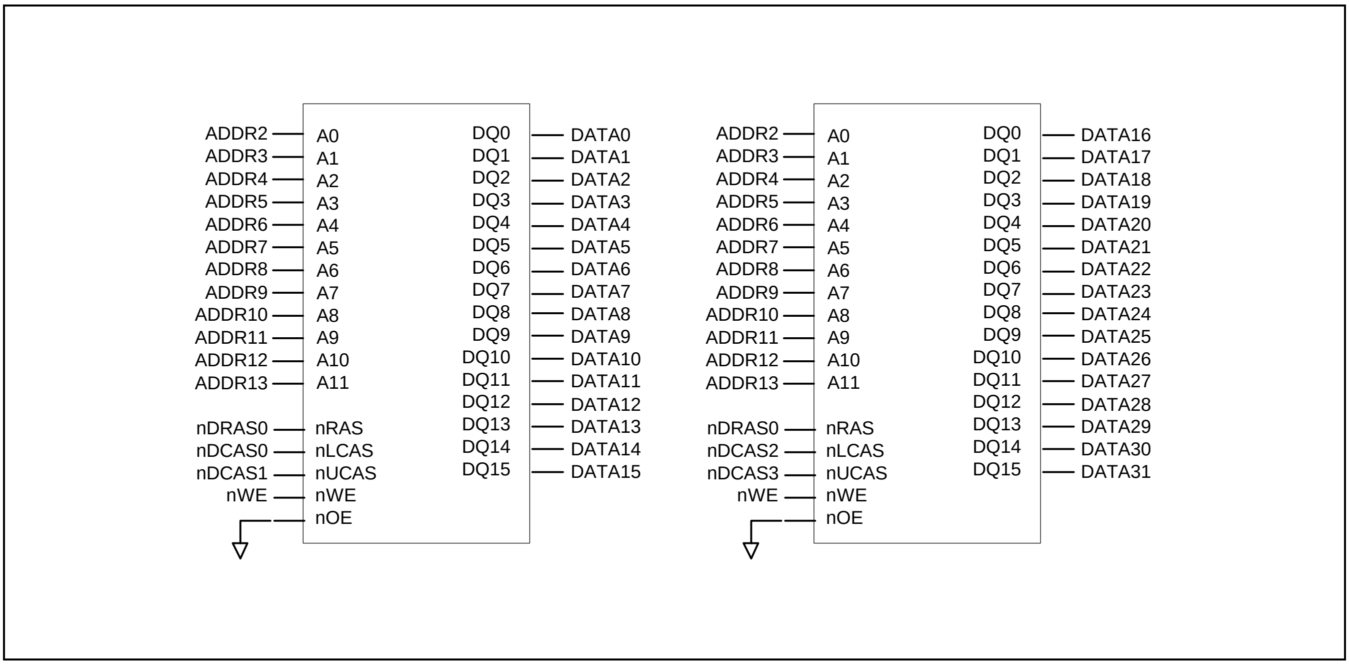


Figure 7-8. Memory Interface with 16bit DRAM x 2

SDRAM MEMORY INTERFACE EXAMPLE

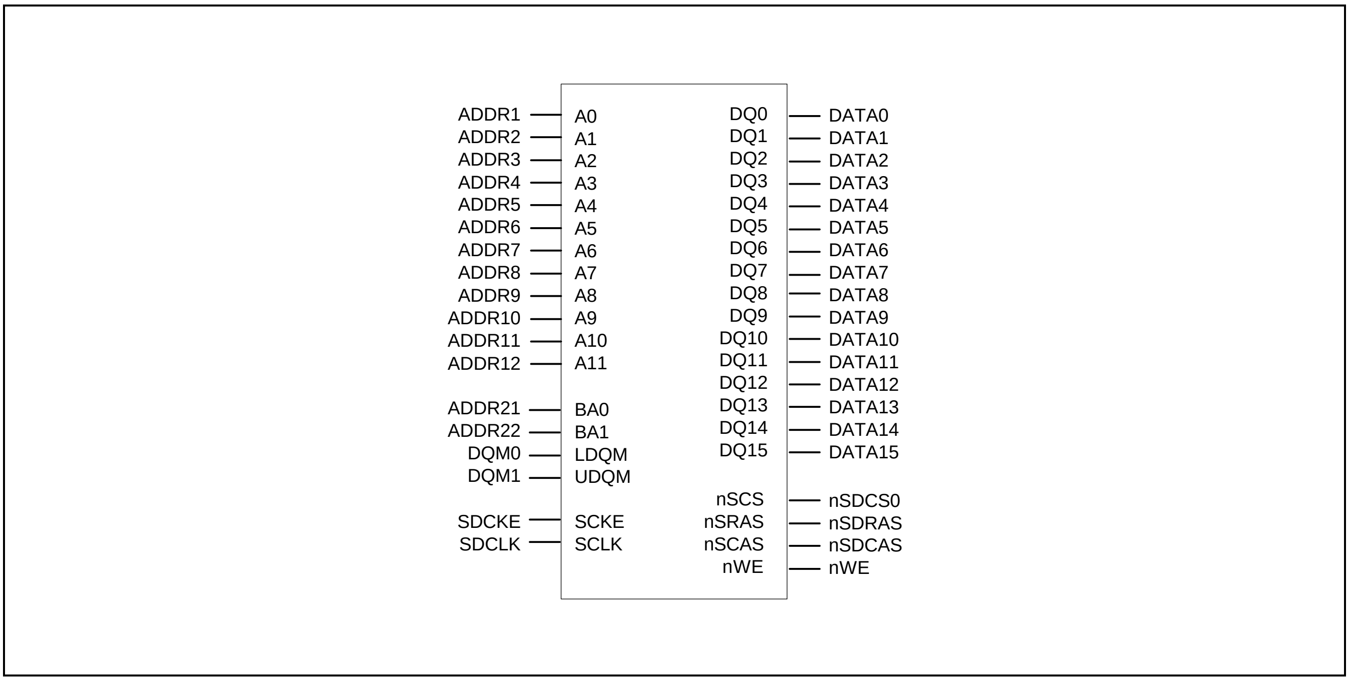


Figure 7-9. Memory Interface with 16bit SDRAM (1Mb x 16bit x 4banks)

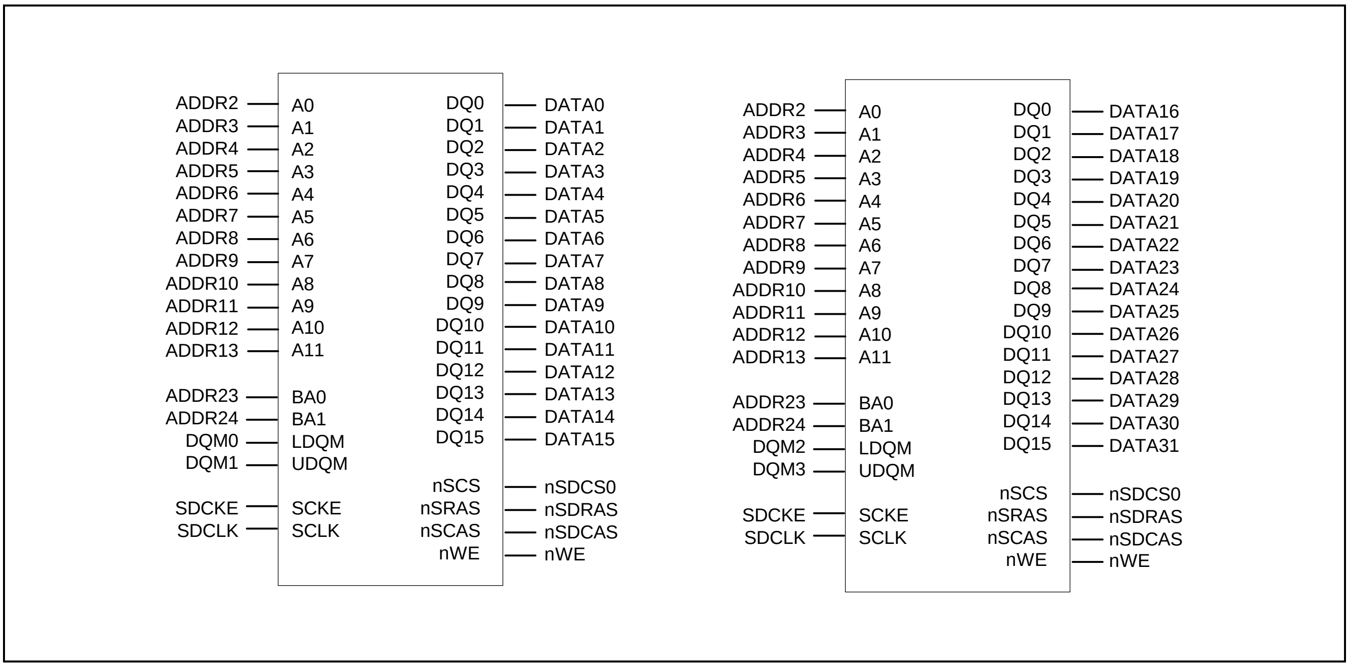


Figure 7-10. Memory Interface with 16bit SDRAM (2Mb x 16bit x 4banks x 2ea)

NOTE : Please refer to Table 7-1 the Bank Address configurations of SDRAM.

STATIC MEMORY TIMING DIAGRAM

READ TIMING FOR STATIC MEMORY

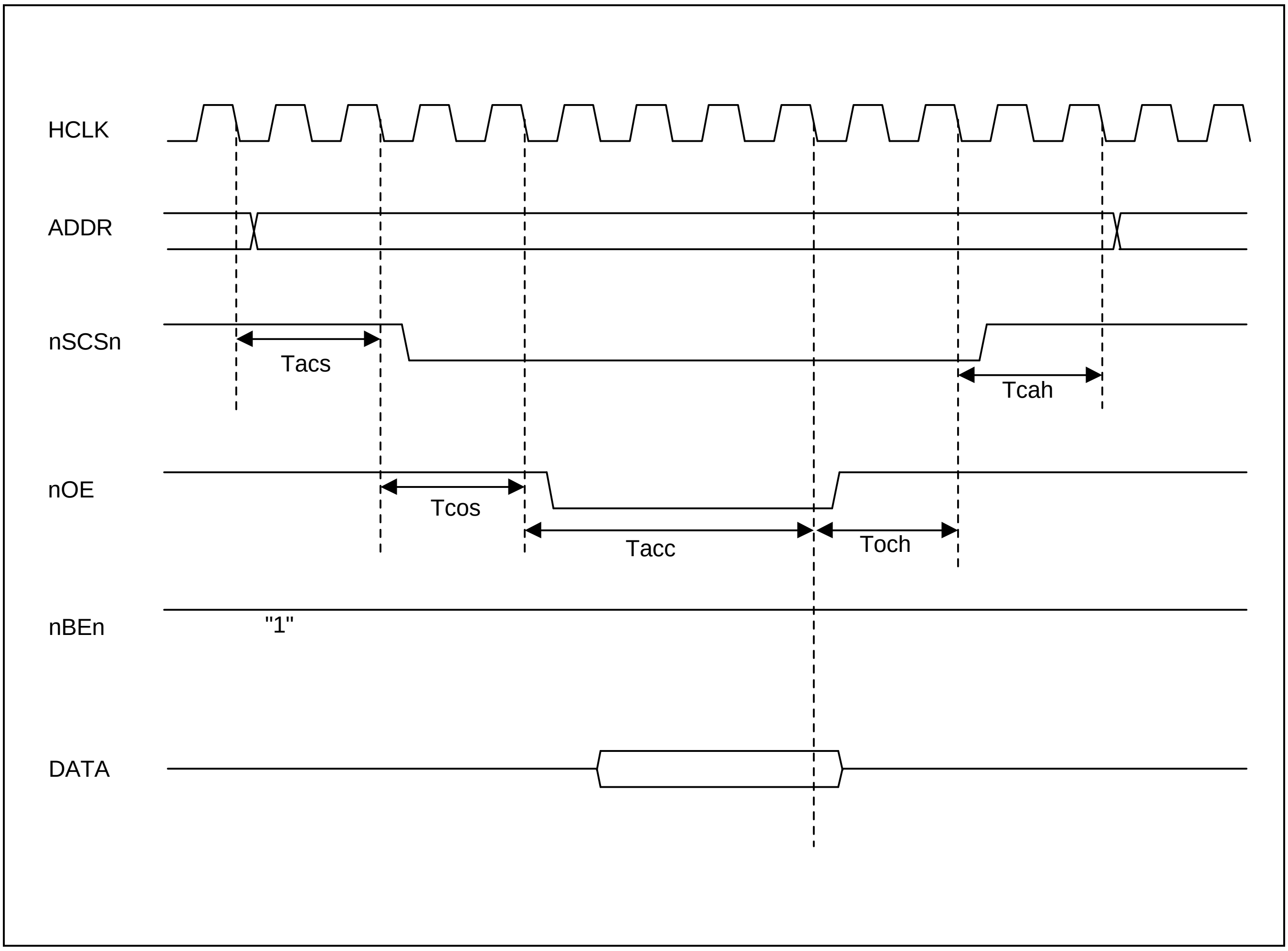


Figure 7-11. Static Memory READ Timing
(Tacs=2,Tcos=2, Tacc=4, Toch=2, Tcah=2,ST=0)

WRITE TIMING FOR STATIC MEMORY

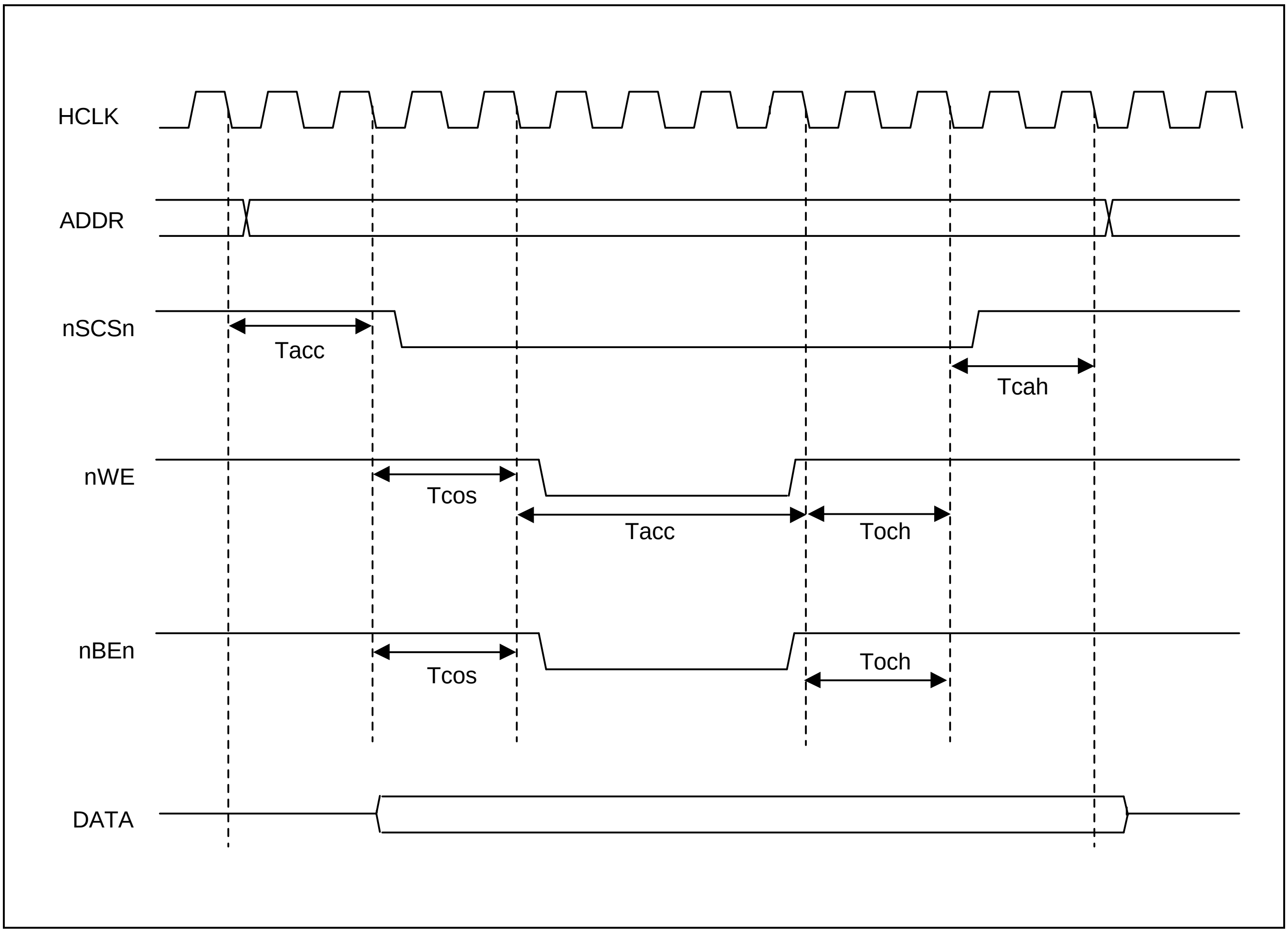


Figure 7-12. Static Memory WRITE Timing
(Tacs=2,Tcos=2,Tacc=4,Toch=2, Tcah=2, ST=0)

nWAIT PIN OPERATION

If the WAIT corresponding each memory bank is enabled, the nOE duration is prolonged by the external nWAIT pin while the memory bank is active.

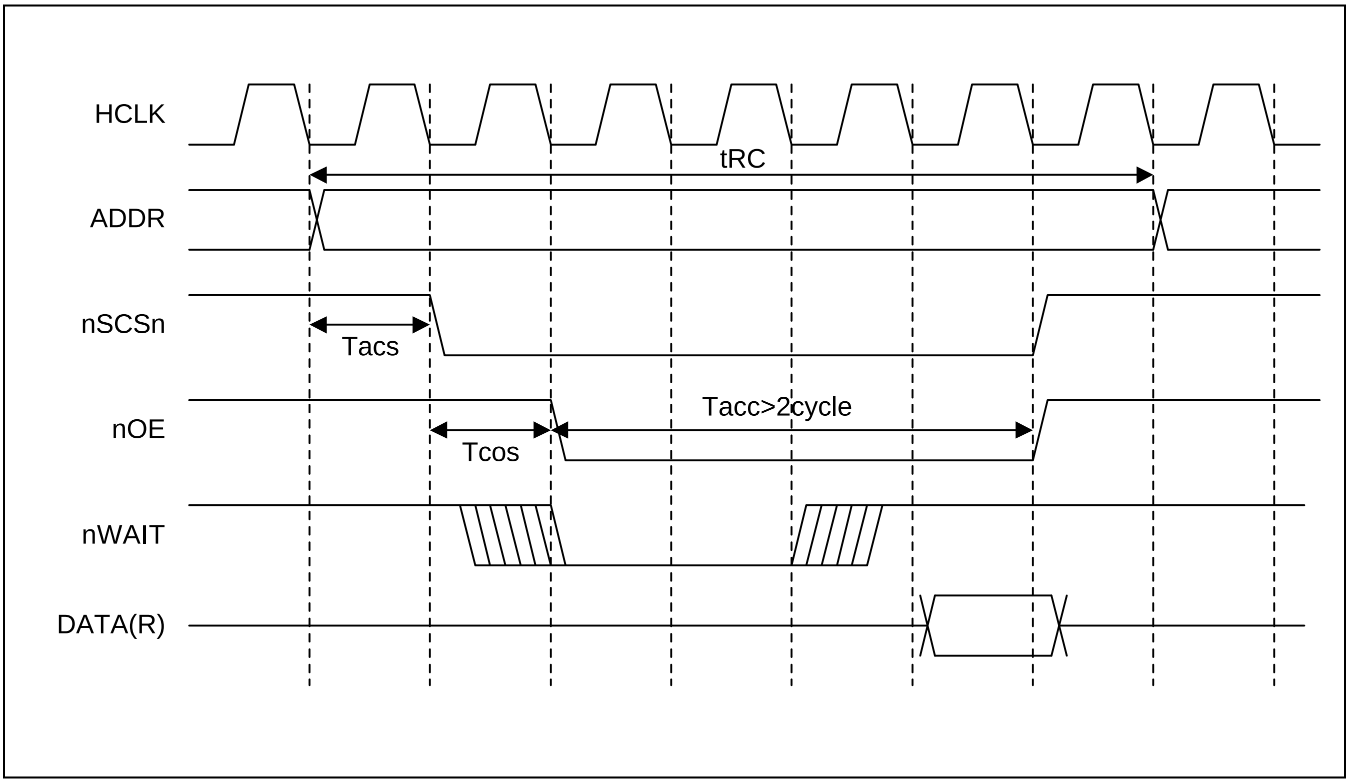


Figure 7-13. External nWAIT Timing Diagram

DYNAMIC MEMORY TIMING DIAGRAM

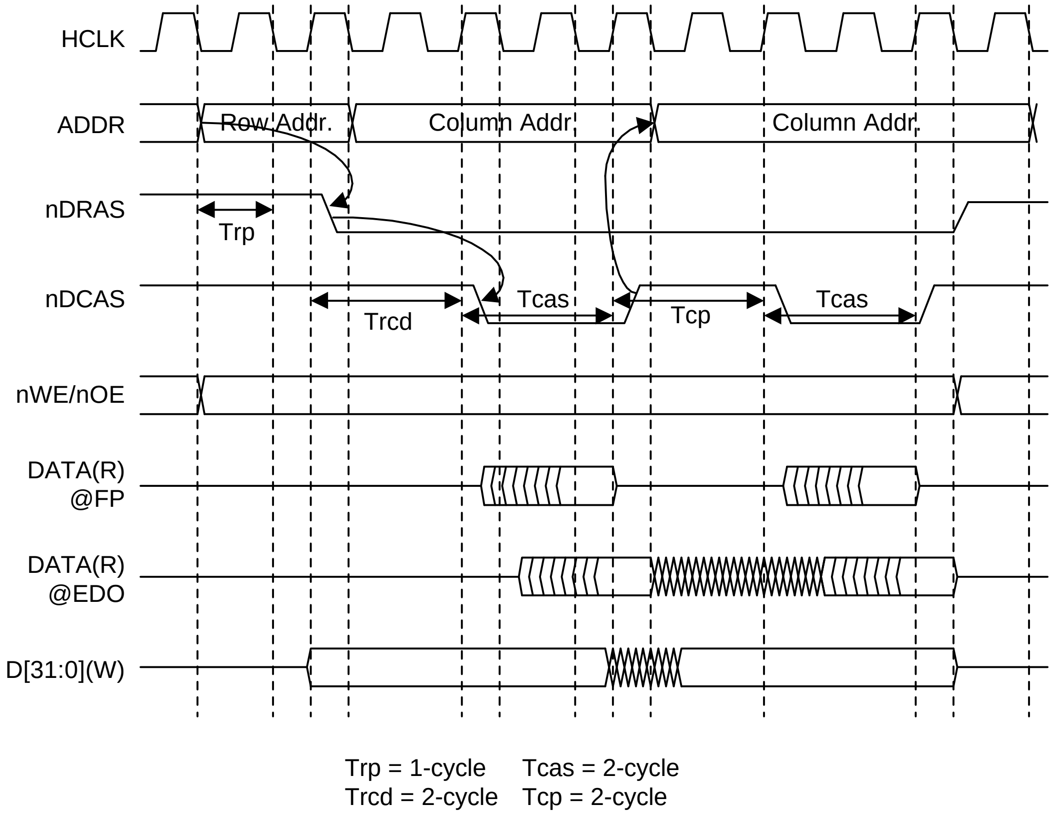


Figure 7-14. DRAM Timing Diagram

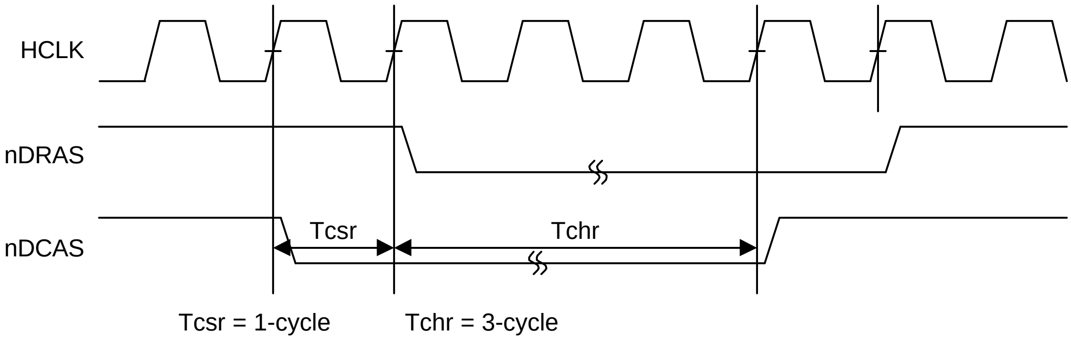


Figure 7-15. DRAM Refresh Timing Diagram

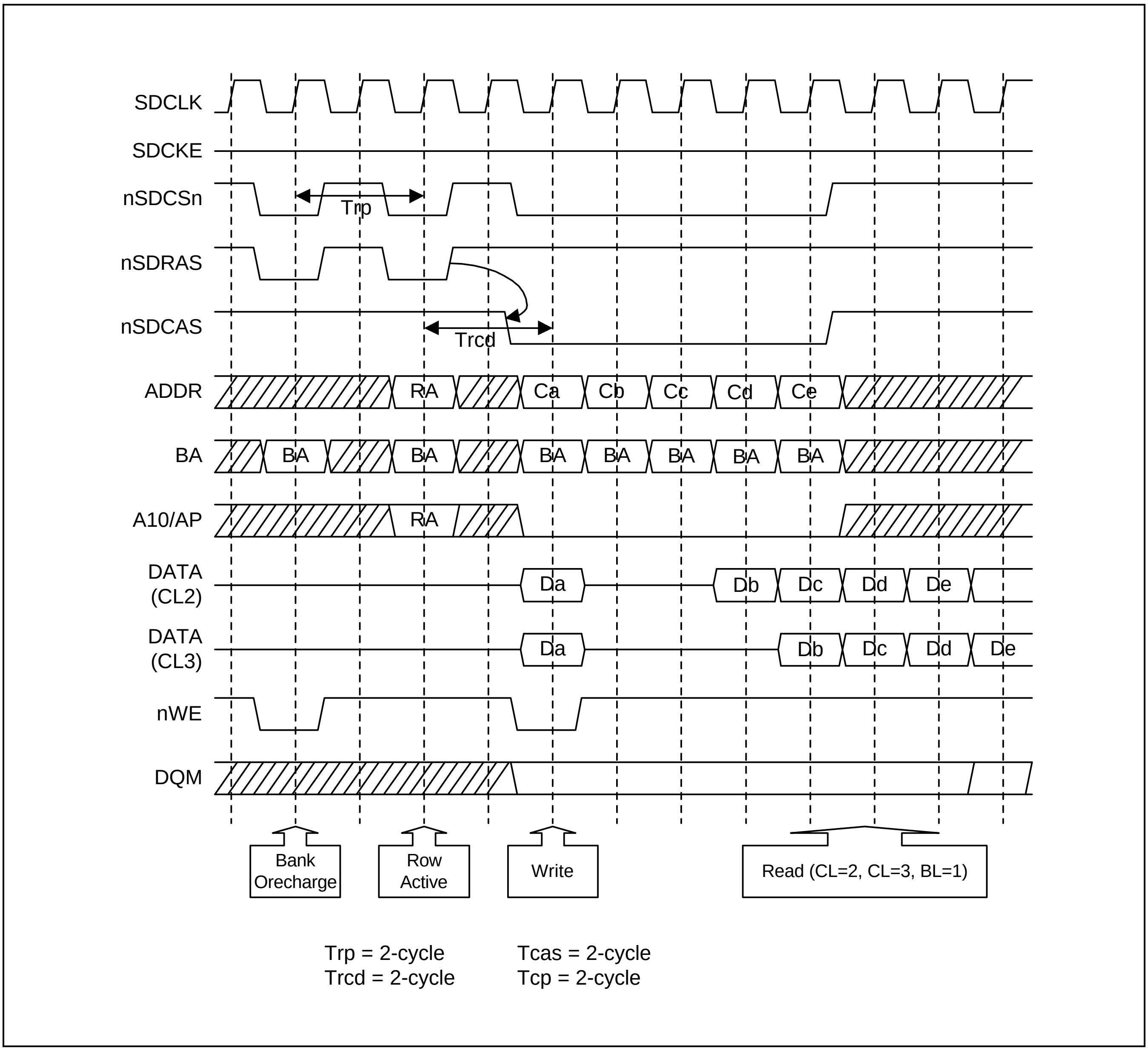


Figure 7-16. SDRAM Timing Diagram

MEMORY CONTROLLER SPECIAL FUNCION REGISTERS

ENDIAN STATUS REGISTER (ENDIAN)

Register	Address	R/W	Description	Reset Value
ENDIAN	0x1001 0000	R	Endian mode control	Undefined

ENDIAN	Bit	Description	Initial state
ENDIAN	[0]	Indicate endian mode (read only) 0 = Little-endian 1 = Big-endian The state is selected by ENDIAN (GPC3) pin	Undefined

STATIC MEMORY BANK CONTROL REGISTER (SMBCON0-SMBCON3)

These registers must be configured to enable static memory (ROM, SRAM, and Flash) for each bank. Boot ROM must be attached to bank 0 if installed.

Register	Address	R/W	Description	Reset Value
SMBCON0	0x1001 0004	R/W	Bank 0 control register for static memory	0x0000 00A2
SMBCON1	0x1001 0008	R/W	Bank 1 control register for static memory	0x0000 00A2
SMBCON2	0x1001 000C	R/W	Bank 2 control register for static memory	0x0000 00A2
SMBCON3	0x1001 0010	R/W	Bank 3 control register for static memory	0x0000 00A2

SMBCONn	Bit	Description	Initial State
WS	[20]	Determines WAIT status 0 = WAIT disable 1 = WAIT enable	0
ST	[19]	Determines SRAM for using UB/LB 0 = Not used UB/LB 1 = Using UB/LB	0
Reserved	[18:14]	Reserved	0x0
Tacs	[13:12]	Address set-up time before nSCSn 00 = 0 clock 01 = 1 clock 10 = 2 clocks 11 = 4 clocks	00
Tcos	[11:10]	Chip selection set-up time before nOE 00 = 0 clock 01 = 1 clock 10 = 2 clocks 11 = 4 clocks	00
Toch	[9:8]	Chip selection hold on time after nOE 00 = 0 clock 01 = 1 clock 10 = 2 clocks 11 = 4 clocks	00
Tacc	[7:4]	Access cycle 0000 = 1 clock 0001 = 2 clocks 0010 = 3 clocks 0011 = 4 clocks 0100 = 6 clocks 0101 = 7 clocks 0110 = 8 clocks 0111 = 9 clocks 1000 = 10 clocks 1001 = 12 clocks 1010 = 14 clocks 1011,11xx = Not used	1010
Tcah	[3:2]	Address holding time after nSCSn 00 = 0 clock 01 = 1 clock 10 = 2 clocks 11 = 4 clocks	00
SDW	[1:0]	Static memory data bus width 00 = 8-bit 01 = 16-bit 10 = 32-bit 11 = Not used **NOTE:** Bank0 = Read only (in SMBCON0) (The states are selected by OM[1:0] pins)	10

DYNAMIC MEMORY REFRESH CONTROL REGISTER (REFRESH)

Register	Address	R/W	Description	Reset Value
REFRESH	0x1001 0014	R/W	DRAM/SDRAM refresh control register for bank0 – bank3.	0x00A4 0000

REFRESH	Bit	Description	Initial state
REFEN	[23]	DRAM/SDRAM refresh enable 0 = Disable 1 = Enable (self or CBR/auto refresh)	1
REFMD	[22]	DRAM/SDRAM refresh mode 0 = CBR/Auto Refresh 1 = Self Refresh In self-refresh time, the DRAM/SDRAM control signals are driven to appropriate level.	0
Trp	[21:20]	DRAM/SDRAM RAS pre-charge time DRAM : 00 =1.5 clocks 01 =2.5 clocks 10 = 3.5 clocks 11 =4.5 clocks SDRAM : 00 = 2 clocks 01 = 3 clocks 1 0 = 4 clocks 11 = Not used	10
Reserved	[19]	Reserved	0
Trc	[18:16]	SDRAM RC minimum time 000 = 4 clocks 001 = 5 clocks 010 = 6 clocks 011 = 7 clocks 100 = 8 clocks 1xx = Not used	100
Reserved	[15:14]	Reserved	00
Tchr	[13:12]	CAS hold time (DRAM) 00 = 1 clock 01 = 2 clocks 10 = 3 clocks 11 = 4 clocks	00
Reserved	[11]	Reserved	
Refresh Counter	[10:0]	DRAM/SDRAM refresh count value. Refresh period = $(2^{11}$-refresh_count+1)/HCLK Ex) If refresh period is 15.6 us and HCLK is 100MHz, the refresh count is as follows; refresh count = $2^{11} + 1 – 100x15.6 = 489$ (=0x1e9)	0x000

DYNAMIC MEMORY TIMING CONTROL REGISTER (DMTMCON)

Register	Address	R/W	Description	Reset Value
DMTMCON	0x1001 0018	R/W	Timing control for dynamic memory (bank0–bank3)	0x0002 0D50

DMTMCON	Bit	Description	Initial state
DW	[17:16]	Dynamic memory bank (0–3) data bus width 00 = 8-bit 01 = 16-bit 10 = 32-bit 11 = Not used	10
Reserved	[15:12]	Reserved	0x0
MT	[11:10]	Dynamic memory type 00 = Reserved 01 = FP DRAM 10 = EDO DRAM 11 = SDRAM	11
Trcd (DRAM)	[9:8]	DRAM RAS to CAS delay 00 = 1 clock 01 = 2 clocks 10 = 3 clocks 11 = 4 clocks	01
Tcas (DRAM)	[7:6]	DRAM CAS pulse width 00 = 1 clock 01 = 2 clocks 10 = 3 clocks 1x = Not used	01
Tcp (DRAM)	[5:4]	DRAM CAS pre-charge 00 = 1 clock 01 = 2 clocks 10 = 3 clocks 1x = Not used	01
CAN/SCAN DRAM/SDRAM	[3:2]	DRAM/SDRAM column address number DRAM : 00 = 8-bit 01 = 9-bit 10 = 10-bit 11 = 11-bit SDRAM : 00 = 8-bit 01 = 9-bit 10 = 10-bit 11 = Not Used	00
Trcd (SDRAM)	[1:0]	SDRAM RAS to CAS Delay 00 = 2 clocks 01 = 3 clocks 1x = 4 clocks	00

SDRAM MODE REGISTER SET REGISTER (MRSR)

Register	Address	R/W	Description	Reset Value
MRSR	0x1001 001C	R/W	Mode register set register for SDRAM bank 0 – bank 3.	0x0000 0030

MRSR	Bit	Description	Initial state
WBL	[9]	Write burst length 0 = Burst (Fixed)	0
TM	[8:7]	Test mode 00 = Mode Register Set (Fixed)	00
CL	[6:4]	CAS latency 000 = 1 clock 001 = Not Used 010 = 2 clocks 011 = 3 clocks 1xx = Not used	011
BT	[3]	Burst type 0 = Sequential (Fixed)	0
BL	[2:0]	Burst length 000 = 1 (Fixed)	000

NOTE: MRSR register must not be reconfigured while the code is running on SDRAM.

Examples Programming of Memory Configuration

```
            ldr         r0,=SMRDATA
            ldr         r1,=SMBCON0            ;SMBCON0 Address
            add         r2, r0, #28            ;End address of SMRDATA (7*4)

0           ldr         r3, [r0], #4
            str         r3, [r1], #4
            cmp         r2, r0
            bne         %B0

SMRDATA DATA

DCD         0x000100a1  ; SMBCON0 , Bus width=16-bit,Access time=14clocks
DCD         0x000100a2  ; SMBCON1 , Bus width=32-bit,Access time=14clocks
DCD         0x000100a2  ; SMBCON2 , Bus width=32-bit,Access time=14clocks
DCD         0x000100a2  ; SMBCON3 , Bus width=32-bit,Access time=14clocks
DCD         0x008301e9  ; REFRESH , Refresh=Enable,CBR/Auto refresh,Counter=0x1e9
DCD         0x00020d55  ; DMTMCON , Bus width=32-bit,Memory=SDRAM,Column addr=9-bit
DCD         0x00000030  ; MRSR    , CAS latency=3clocks
```

NOTES

8 DMA

OVERVIEW

S3C2800 supports four-channel DMA controller that is located between the system bus (AHB) and the peripheral bus(APB). Each channel of DMA controller can perform data transfers between devices in the system bus and/or peripheral bus with no restrictions. In other words, each channel can handle the following data transfers:

1. both source and destination are in the system bus (AHB)
   (ex, Memory to Memory transfer)
2. source is in the system bus (AHB) while destination is in the peripheral bus (APB)
   (ex, memory to an I/O device transfer)
3. source is in the peripheral bus (APB) while destination is in the system bus (AHB)
   (ex, I/O device to memory transfer)
4. both source and destination are in the peripheral bus (APB)
   (ex, I/O device to I/O device transfer)

The main advantage of DMA is that it allows the data transfer the data without CPU intervention. The DMA operation can be requested by either software or hardware including external DMA source.

DMA OPERATION

The details of DMA operation can be explained based-on the following 3-state FSM (finite state machine):

State-1. As an initial state, it waits for the DMA request. If it comes, go to state-2. At this state, DMA ACK and INT REQ are high.

State-2. In this state, DMA ACK becomes low and current transfer counter (CURR_TC) is loaded from the DCON[19:0] register. Note that DMA ACK becomes low and remains low until it becomes high later.

State-3. In this state, the atomic operation of DMA handled by sub-FSM is initiated. The sub-FSM reads the data from the source address and then writes it to destination address. In this operation, data size(byte, half word, or word) and transfer size (single or burst) are considered. This operation is repeated until the current transfer counter(CURR_TCreaches 0 in the whole service mode, while performed only once in a single service mode. The main FSM (this FSM) counts down the CURR_TC when the sub-FSM finishes each atomic operation. In addition, this main FSM asserts the INT REQ signal when CURR_TC becomes 0 and the interrupt setting of DCON[[28] register is set to 1. In addition, the DMA ACK becomes high if one of the following conditions are met.

1) CURR_TC becomes 0 in the whole service mode, or
2) atomic operation finishes in the single service mode.

Note that in the single service mode, these three states of main FSM are performed once, then stops, and waits for another DMA REQ. And if another DMA REQ occurs, all the three states are repeated. Therefore, DMA ACK is asserted and then de-asserted for each atomic transfer. In contrast, in the whole service mode, main FSM waits at state-3 until CURR_TC becomes 0. Therefore, DMA ACK is asserted during all the transfers and then de-asserted when TC reaches 0.

However, INT REQ is asserted only if CURR_TC becomes 0 regardless of the service mode (single service mode or whole service mode).

Figure 8-1 shows the internal diagram of a DMA block. The DMA acts as a bridge, which provides the interface layer between AHB and APB. The main role of DMA is to transfer the data between external memory and internal peripherals such as UART, Timer, etc, which are attached to APB. The Timer can also request DMA operation with a specified time interval. Usually, CPU or other master device should access to external memory through memory controller, which is attached to AHB. Please remind that the DMA is also a kind of master device. To transfer the data from memory(peripheral devices) to peripheral devices(memory) attached to APB(AHB), we should use the memory controller attached to AHB. Because the DMA is in the Bridge, which is an interface layer between AHB and APB, it can transfer the data between two devices, which are attached to AHB as well as APB.

The DMA also provides a temporary buffer which allows multiple transfers to enhance the bus utilization as well as transfer speed. Specifically, S3C2800 has a 4-word FIFO-type buffer to support the 4-word burst transfer during DMA operation. For example, the DMA operation between memories can be done by a 4-word burst write followed by a 4-word burst read.

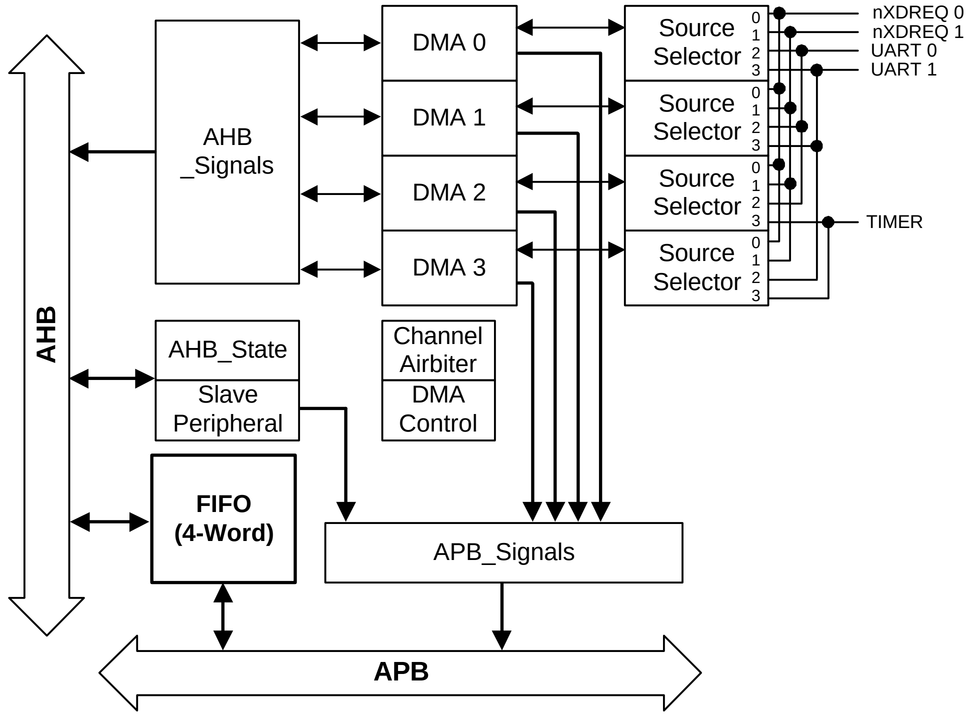


Figure 8-1. DMA Controller Block Diagram

DMA REQUEST/ACKNOWLEDGE PROTOCOL

There are two types of DMA request/acknowledge protocol. Each type defines how DMA request and acknowledge signals are related to these protocol.

Basic DMA Timing

The DMA service means paired reads and writes cycles during DMA operation, which is one DMA operation. The Figure 8-2 shows the basic timing in the DMA operation.

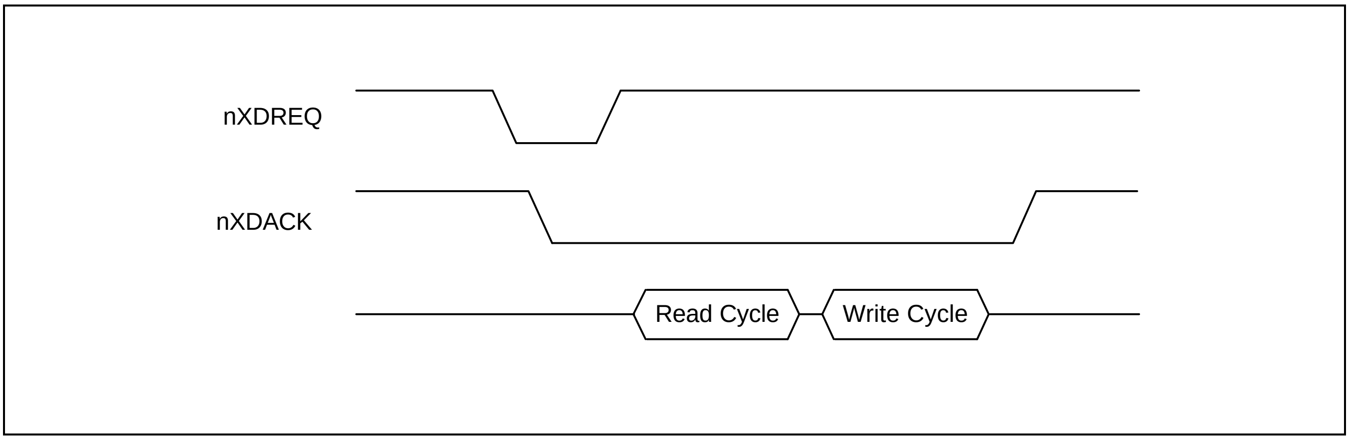


Figure 8-2. Basic DMA Timing Diagram

Demand/Handshake Mode Comparison – Related to the Protocol between nXDREQ and nXDACK

These are two different modes related to the protocol between nXDREQ and nXDACK. Fig. 8-3 shows the differences between these two modes i.e., Demand and Handshake modes.

At the end of one transfer(Single/Burst transfer), DMA checks the state of double-synched nXDREQ.

Demand mode

— If nXDREQ remains asserted, the next transfer starts immediately. Otherwise it waits for nXDREQ to be asserted.

Handshake mode

— If nXDREQ is deasserted, DMA deasserts nXDACK. Otherwise it waits until nXDREQ is deasserted. Caution : nXDREQ has to be asserted(low) only after the deassertion(high) of nXDACK.

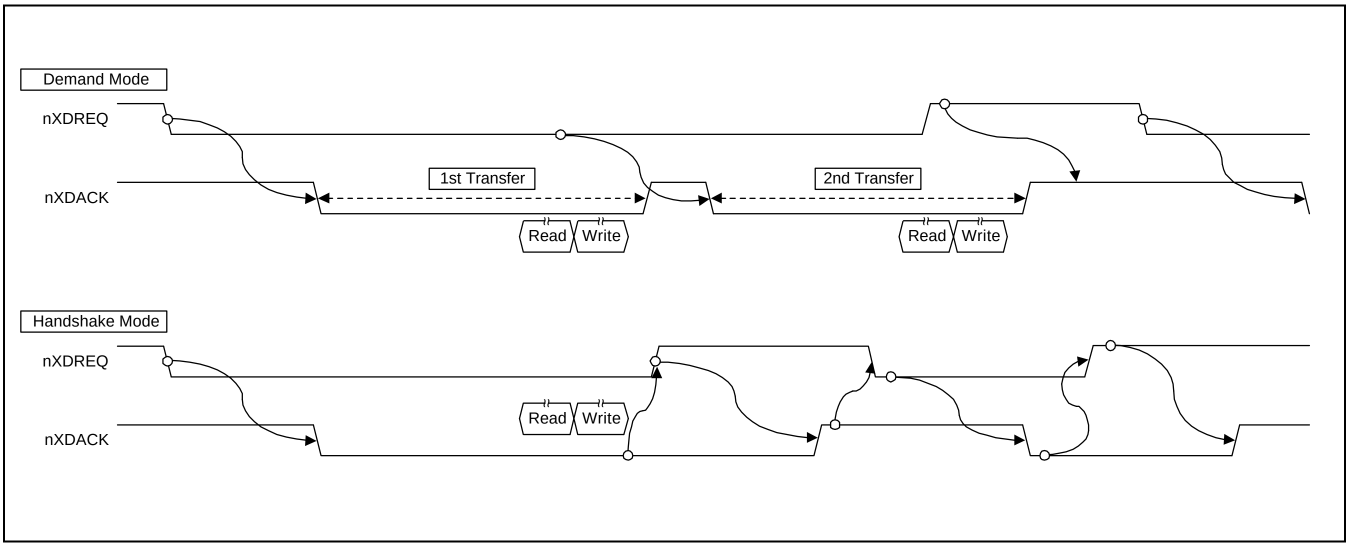


Figure 8-3. Demand/Handshake Mode Comparison

SERVICE MODE

Single service Mode

The single service mode means that there is one DMA acknowledge cycles indicating DMA read and write cycle..

When the DMA request signal goes low, the bus controller indicates the bus allocation for the DMA operation by lowering the DMA acknowledge signal if there is no higher priority bus request except this DMA request. During the first low level period of the DMA acknowledge signal, a DMA read cycle will be initiated. After the DMA read cycle, the next DMA write cycle follows.

The DMA ACK signal is de-asserted when the atomic operation (i.e., read followed by write operation) is finished. The INT REQ signal is asserted only if current transfer counter (CURR_TC) becomes 0.

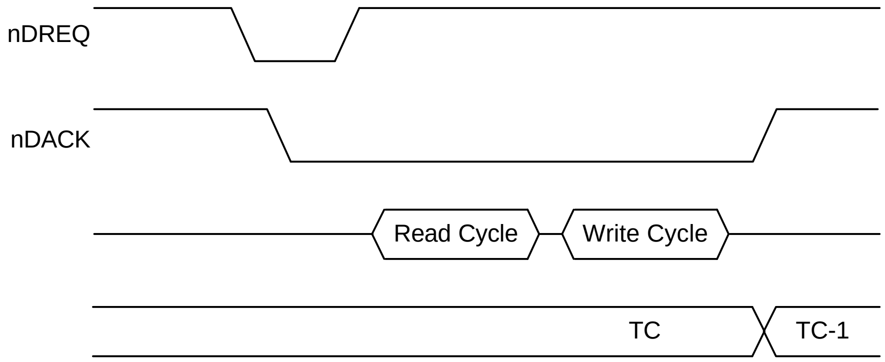


Figure 8-4. Single Transfer in Single Service Mode

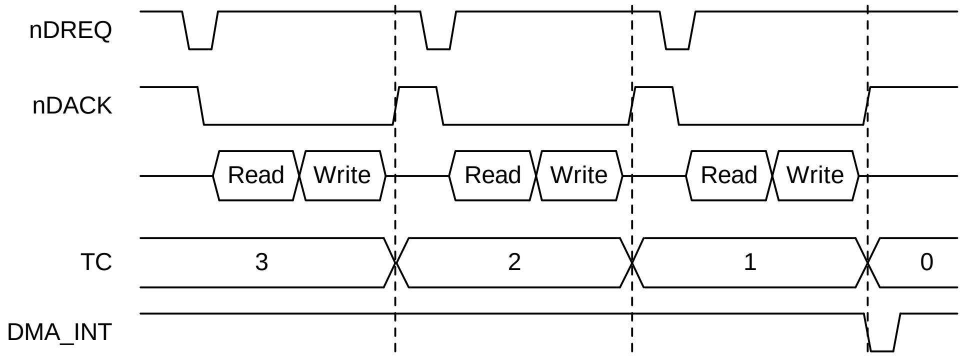


Figure 8-5. Sequential Transfer in Single Service Mode

Whole Service Mode

The whole service mode means that the specified number of DMA operations, i.e., number of DMA operations based on transfer count, will be initiated by a single activation of DMA request, and will proceed without further activation of DMA requests. The below figure shows how the whole service mode proceeds. The nDACK signal remains active until the end of whole DMA operations.

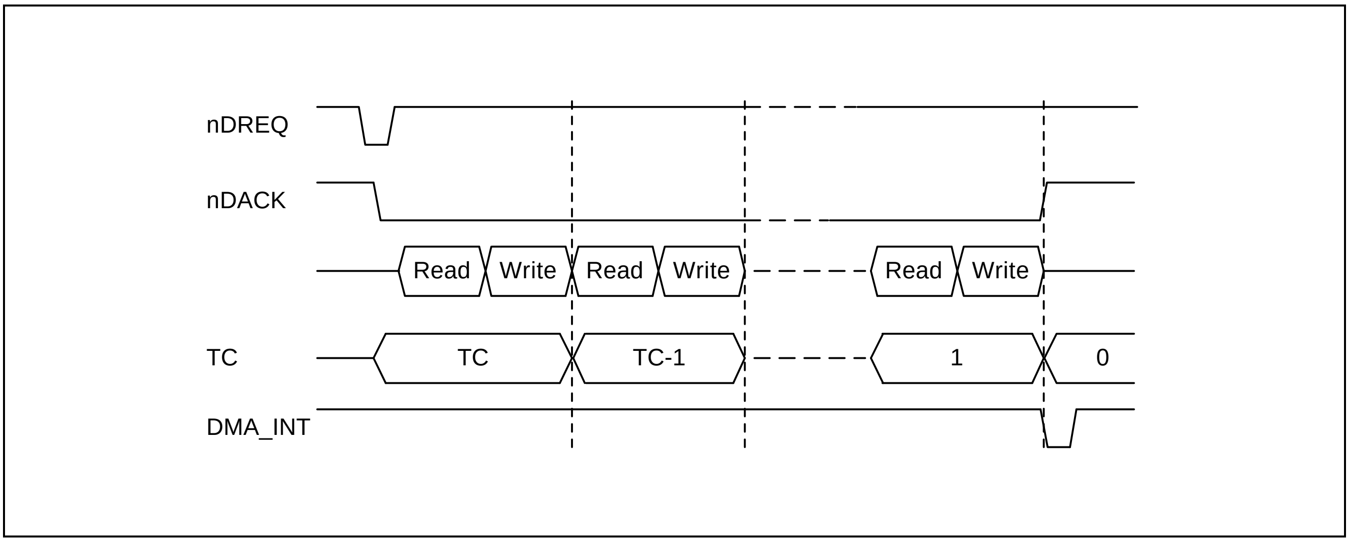


Figure 8-6. Whole Service Mode

DMA TRANSFER SIZE

There are tow types of DMA transfer size (Single transfer size, Burst transfer size). Unlike DMA request/acknowledge protocol, the DMA transfer size defines the number of read/write per unit transfer as shown in the following table.

DMA Transfer Size	Read/Write
Single transfer	1 unit read, then 1 unit write
Burst transfer	4 unit burst read, then 4 unit burst write

NOTE: unit is Byte, Half Word, or Word.

Single Transfer Size

The single transfer mode means that the paired DMA read/write cycle is performed per each DMA request as shown in Figure 8-4.

Burst (4-unit) Transfer Size

The burst (4 unit) transfer mode means that the successive 4-unit DMA read cycle is performed followed by the successive 4-unit DMA write cycles.

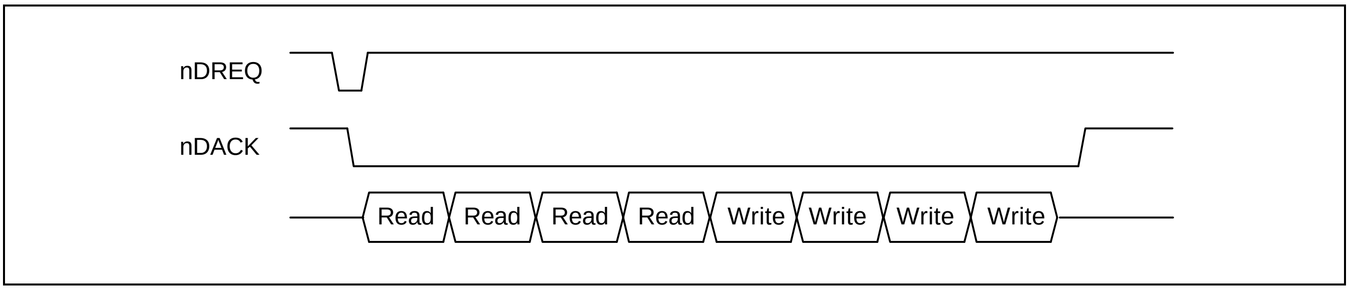


Figure 8-7. Burst (4-unit) Transfer Size

DMA REQUEST SOURCE SELECTION

Register DCONn[23] selects either software or hardware DMA request mode: if hardware request mode is set, one of the source in Table 8-1 can be selected by register DCONn[25:24].
Note that if software request mode is selected, the selected hardware DMA source does not have any meaning. The hardware DMA request source for each channel are as follows.

Table 8-1. Hardware DMA Sources

	Source 0	Source 1	Source 2	Source 3
Ch-0	nXDREQ0	nXDREQ1	UART0	UART1
Ch-1	nXDREQ0	nXDREQ1	UART0	UART1
Ch-2	nXDREQ0	nXDREQ1	UART0	TIMER
Ch-3	nXDREQ0	nXDREQ1	UART1	TIMER

NOTE: The PCI memory space can be accessed through the DMA channel 0 in S/W trigger mode. However, the burst transfer size cannot be used.

The software trigger can be done by writing the SW-TRIG field as 1 in DMASKTRIGn register, i.e., the start of DMA. Before the start of DMA, we should configure DMA-related parameter, for example source address, destination address, transfer count and so on. Based-on these configuration, the DMA operation will start when we write the SW_TRIG field as 1. In this case of software trigger, the DMA operations will continue as long as the bust mastership is allocated to DMA master and as long as the DMA current transfer count (CURR_TC) reaches to zero, i.e., the completion of DMA operation.

In DMA, there are five hardware request sources: nXDREQ0, nXDREQ1, UART0, UART,1and TIMER. The DMA can also be initiated by software. The sources of DMA operation are selected by writing the HWSRCSEL(bit[25:24]) field in DCONn register.

Here nXDREQ0 and nXDREQ1 represent two external source(External Devices).

DMA SPECIAL FUNCTION REGISTERS

There are seven control registers for each DMA channel. (Since there are four channels, the total number of control register is 28.)—four of them control the DMA transfer, and other three show the status of DMA controller. The details of those registers are given below.

DMA INITIAL SOURCE REGISTER (DISRCn)

Register	Address	R/W	Description	Reset Value
DISRC0	0x1003 0000	R/W	DMA 0 Initial source register	0x0000 0000
DISRC1	0x1004 0000	R/W	DMA 1 Initial source register	0x0000 0000
DISRC2	0x1005 0000	R/W	DMA 2 Initial source register	0x0000 0000
DISRC3	0x1006 0000	R/W	DMA 3 Initial source register	0x0000 0000

DISRCn	Bit	Description	Initial State
S_LOC	[31]	Select the location of source. 0 = The source is in the system bus (AHB). 1 = The source is in the peripheral bus(APB).	0
S_INC	[30]	Select the source address increment. 0 = Increment 1 = Fixed(Not changed) If it is 0, the address is increased by its data size after each transfer in burst and single transfer mode. If it is 1, the address is not changed after the transfer (In the burst mode, address is increased during the burst transfer, but the address is recovered to its first value after the transfer).	0
S_ADDR	[29:0]	These bits are the base address (start address) of source data to transfer. This value will be loaded into CURR_SRC only if the CURR_TC is 0 and the DMA ACK is asserted.	0x0000 0000

DMA INITIAL DESTINATION REGISTER (DIDSTn)

Register	Address	R/W	Description	Reset Value
DIDST0	0x1003 0004	R/W	DMA 0 Initial destination register	0x0000 0000
DIDST1	0x1004 0004	R/W	DMA 1 Initial destination register	0x0000 0000
DIDST2	0x1005 0004	R/W	DMA 2 Initial destination register	0x0000 0000
DIDST3	0x1006 0004	R/W	DMA 3 Initial destination register	0x0000 0000

DIDSTn	Bit	Description	Initial State
D_LOC	[31]	Select the location of destination. 0 = The destination is in the system bus (AHB). 1 = The destination is in the peripheral bus(APB).	0
D_INC	[30]	Select the source address increment 0 = Increment 1 = Fixed(Not changed) If it is 0, the address is increased by its data size after each transfer in burst and single transfer mode If it is 1, the address is not changed after the transfer (In the burst mode, address is increased during the burst transfer, but the address is recovered to its first value after the transfer).	0
D_ADDR	[29:0]	These bits are the base address (start address) of destination for the transfer. This value will be loaded into CURR_SRC only if the CURR_TC is 0 and the DMA ACK is asserted.	0x0000 0000

DMA CONTROL REGISTER (DCONn)

Register	Address	R/W	Description	Reset Value
DCON0	0x1003 0008	R/W	DMA 0 control register	0x0000 0000
DCON1	0x1004 0008	R/W	DMA 1 control register	0x0100 0000
DCON2	0x1005 0008	R/W	DMA 2 control register	0x0200 0000
DCON3	0x1006 0008	R/W	DMA 3 control register	0x0300 0000

DCONn	Bit	Description	Initial State
DMD_HS	[30]	Select one between demand mode and handshake mode. 0 = demand mode is selected. 1 = handshake mode is selected. In both modes, DMA controller starts its transfer and asserts DACK for a given asserted DREQ. The difference between two modes is whether it waits de-asserted DREQ or not. In handshake mode, DMA controller waits fot the de-asserted DREQ before starting a new transfer. if it sees the de-asserted DREQ, it de-asserts DACK and waits for another asserted DREQ. In contrast, in the demand mode, DMA controller does not wait until the DREQ is de-asserted. It just de-asserted DACK and then starts another transfer if DREQ is asserted. *We recommend using handshake mode for external DMA request sources to prevent unintended starts of new transfers.*	0
SYNC	[29]	Select DREQ/DACK synchronization. 0 = DREQ and DACK are synchronized to PCLK. 1 = DREQ and DACK are synchronized to HCLK. Therefore, devices attached to AHB system bus, this bit has to 1, while those devides attached to APB system should be set to 0. For the devices attached to external system, user should select this bit depending on whether the external system is synchronized with AHB system or APB system.	0
INT	[28]	Enable/Disable the Interrupt. 0 = Interrupt is disabled . 1 = Interrupt is enabled Interrupt request is generated when all the transfer is done (i.e., CURR_TC becomes 0)..	0
TSZ	[27]	Selected the transfer size of an atomic transfer (i.e., transfer performed at each time DMA owns the bus before releasing the bus). 0 = a single transfer is performed. 1 = a burst transfer of length four is performed.	0

DCONn	Bit	Description	Initial State
SERVMODE	[26]	Select the service mode between single service mode and whole service mode. 0 = Single service mode is selected in which after each atomic transfer (single or burst of length four) DMA stops and waits for another DMA request. 1 = Whole service mode is selected in which one request gets atomic transfers to be repeated until the transfer count reaches to 0. In this mode, additional request is not required.	0
HWSRCSEL	[25:24]	Select DMA request source for each DMA. Refer to Table 8-2 This bits control the 4:1 MUX to select the DMA request source of each DMA. These bits have meanings if and only if hardware request mode is selected by DCONn[23].	Refer to Table 8-2
SWHW_SEL	[23]	Select the DMA source between software and hardware request mode. 0 = S/W request mode is selected and DMA is triggered by setting SW_TRIG bit of DMASKTRIGn control register. 1 = Hardware request is selected. DMA source selected by bit[25:24] is used to trigger the DMA operation. **NOTE:** The PCI memory space can be accessed through the DMA channel 0 in software trigger mode. However, the burst transfer size cannot be used.	0
Auto_MASK_ ON_OFF	[22]	Set the auto mask of the MASK_ON_OFF bit in DMASKTRIG register. 0 = Auto mask off is not performed when a current value of transfer count becomes 0 (i.e., all the required transfers are performed). 1 = DMA channel(DMA REQ) is turned off when a current value of transfer count becomes 0. The channel on/off bit (DMASKTRIGn[1]) is set to 0 (DREQ off) to prevent unintended further start of new DAM operation.	0
DSZ	[21:20]	Data size to be transferred. 00 = Byte 01 = Half word 1x = Word	00
TC	[19:0]	Initial transfer count (or transfer beat) Note that the actual number of bytes that are transferred is computed by the following equation: DSZ x TSZ x TC, where DSZ, TSZ, and TC represent data size (bit[21:20]), transfer size (bit[27]), and initial transfer count, respectively. This value will be loaded into CURR_TC only if the CURR_TC is 0 and the DMA ACK is 1.	0x0 0000

Table 8-2. DMA Source selection

Register	Bit	Description	Initial State
DCON0 (DMA 0)	[25:24]	Selection of hardware DMA request for DMA channel 0 00 = nXDREQ0 01 = nXDREQ1 10 = UART0 11 = UART1	00
DCON1 (DMA 1)		Selection of hardware DMA request for DMA channel 1 00 = nXDREQ0 01 = nXDREQ1 10 = UART0 11 = UART1	01
DCON2 (DMA 2)		Selection of hardware DMA request for DMA channel 2 00 = nXDREQ0 01 = nXDREQ1 10 = UART0 11 = TIMER	10
DCON3 (DMA 3)		Selection of hardware DMA request for DMA channel 3 00 = nXDREQ0 01 = nXDREQ1 10 = UART1 11 = TIMER	11

NOTE: A DMA cannot use a DMA request source used by another DMA channel.
Ex) If the DMA channel 0 has been selected to UART0, DMA1,DMA2 and DMA3 cannot use UART0 as the request source.

DMA STATUS REGISTER (DSTATn)

Register	Address	R/W	Description	Reset Value
DSTAT0	0x1003 000C	R	DMA 0 status register	0x0000 0000
DSTAT1	0x1004 000C	R	DMA 1 status register	0x0000 0000
DSTAT2	0x1005 000C	R	DMA 2 status register	0x0000 0000
DSTAT3	0x1006 000C	R	DMA 3 status register	0x0000 0000

DSTATn	Bit	Description	Initial State
STAT	[20]	Status of the DMA controller. 0 = It indicates that DMA controller is ready for another DMA request. 1 = It indicates that DMA controller is busy for transfers.	0
CURR_TC	[19:0]	Current value of transfer counts. Note that transfer count is initially set to the value of DCONn[19:0] register and decreased by one at the end of every atomic transfer.	0x00 0000

DMA CURRENT SOURCE REGISTER (DCSRCn)

Register	Address	R/W	Description	Reset Value
DCSRC0	0x1003 0010	R	DMA 0 current source register	0x0000 0000
DCSRC1	0x1004 0010	R	DMA 1 current source register	0x0000 0000
DCSRC2	0x1005 0010	R	DMA 2 current source register	0x0000 0000
DCSRC3	0x1006 0010	R	DMA 3 current source register	0x0000 0000

DCSRCn	Bit	Description	Initial State
CURR_SRC	[29:0]	Current source address for DMAn	0x0000 0000

DMA CURRENT DESTINATION REGISTER (DCDSTn)

Register	Address	R/W	Description	Reset Value
DCDST0	0x1003 0014	R	DMA 0 current destination register	0x0000 0000
DCDST1	0x1004 0014	R	DMA 1 current destination register	0x0000 0000
DCDST2	0x1005 0014	R	DMA 2 current destination register	0x0000 0000
DCDST3	0x1006 0014	R	DMA 3 current destination register	0x0000 0000

DCDSTn	Bit	Description	Initial State
CURR_DST	[29:0]	Current destination address for DMAn	0x0000 0000

DMA MASK TRIGGER REGISTER (DMASKTRIGn)

Register	Address	R/W	Description	Reset Value
DMASKTRIG0	0x1003 0018	R/W	DMA 0 mask trigger register	0x0000 0000
DMASKTRIG1	0x1004 0018	R/W	DMA 0 mask trigger register	0x0000 0000
DMASKTRIG2	0x1005 0018	R/W	DMA 0 mask trigger register	0x0000 0000
DMASKTRIG3	0x1006 0018	R/W	DMA 0 mask trigger register	0x0000 0000

DMASKTRIGn	Bit	Description	Initial State
STOP	[2]	Stop the DMA operation. 1 = DMA stops as soon as the current atomic transfer ends. If there is no current running atomic transfer, DMA stops immediately. The CURR_TC will be 0. **NOTE**: Due to possible current atomic transfer, "stop" may take several cycles. The finish of "stopping" operation (i.e., actual stop time) can be detected by waiting until the channel on/off bt(bit[1]) is set to off. This stop is "actual stop".	0
MASK_ ON_OFF	[1]	DMA mask on/off bit (This bit is depending on the DCONn[22] register). 0 = Masked (DMA channel is turned off). The DMA request to this channel is ignored, the DMA controller is masked and not handled. **NOTE**: User can't write "0" (ignored). 1 = DMA channel is turned on and the DMA request is handled. This bit is automatically set to 0 if we set the DCONn[22] bit to 1 and/or DMASKTRIGn[2] bit to 1 (stop). Note that when DCON[22] bit is 1 , this bit becomes 0 when CURR_TC reaches 0. If the STOP bit is 1, this bit becomes 0 as soon as the current atomic transfer finishes.	0
SW_TRIG	[0]	Trigger the DMA channel in software request mode. 1 = it requests a DMA operation to this controller. However, note that for this trigger to have effects software request mode has to be selected (DCONn[23]) and MASK_ON_OFF bit has to be set to 1. When DMA operation starts, this bit is cleared automatically.	0

NOTE: You can freely change the values of DISRC register, DIDST registers, and TC field of DCON register. Those changes take effect only after the finish of current transfer (i.e., when CURR_TC becomes 0). On the other hand, any change made to other registers and/or fields takes immediate effect. Therefore, be careful in changing those registers and fields.

9 GPIO PORTS

OVERVIEW

S3C2800 has 44 multi-functional GPIO (general-purpose input/output) port pins organized into six port groups:

- Five 8-bit input/output ports(A,B,D,E,F).
- One 4-bit input/output ports(C).

Each port can be easily configured by software to meet various system configuration and design requirements. These multi-functional pins need to be properly configured before their use. If a multiplexed pin is not used as a dedicated functional pin, this pin can be configured as GPIO ports.

The initial pin states, before pin configurations, are configured elegantly to avoid some problems.

Table 9-1. Port Configuration Overview

Port A	Selectable Pin functions	
	Function 1	**Function 2**
GPA0	Input/output	nSCS1
GPA1	Input/output	nSCS2
GPA2	Input/output	nSCS3
GPA3	Input/output	nSDCS1/nDRAS1
GPA4	Input/output	nSDCS2/nDRAS2
GPA5	Input/output	nSDCS3/nDRAS3
GPA6	Input/output	nDCAS0
GPA7	Input/output	nDCAS1

Port B	Selectable Pin functions	
	Function 1	**Function 2**
GPB0	Input/output	nDCAS2/nSDCAS
GPB1	Input/output	nDCAS3/nSDRAS
GPB2	Input/output	nBE0/nWBE0/DQM0
GPB3	Input/output	nBE1/nWBE1/DQM1
GPB4	Input/output	nBE2/nWBE2/DQM2
GPB5	Input/output	nBE3/nWBE3/DQM3
GPB6	Input/output	nWAIT
GPB7	Input/output	CLKout

Port C	Selectable Pin functions	
	Function 1	**Function 2**
GPC0	Input/output	-
GPC1	Input/output	-
GPC2	Input/output	-
GPC3	ENDIAN	output only

NOTE: ENDIAN is used only when nRESET is Low.
Endian value is latched only at the rising edge of nRESET: when nRESET is Low, the ENDIAN(GPC3) pin operates in input mode; nRESET becomes High, the ENDIAN pin will automatically switch to output mode.

Table 9-1. Port Configuration Overview (Continued)

Port D	Selectable Pin functions	
	Function 1	Function 2
GPD0	Input/output	IICSDA0
GPD1	Input/output	IICSCLK0
GPD2	Input/output	IICSDA1
GPD3	Input/output	IICSCLK1
GPD4	Input/output	RxD0
GPD5	Input/output	TxD0
GPD6	Input/output	nCTS0
GPD7	Input/output	nRTS0

Port E	Selectable Pin functions	
	Function 1	Function 2
GPE0	Input/output	RxD1
GPE1	Input/output	TxD1
GPE2	Input/output	nCTS1
GPE3	Input/output	nRTS1
GPE4	Input/output	nXDREQ0
GPE5	Input/output	nXDACK0
GPE6	Input/output	nXDREQ1
GPE7	Input/output	nXDACK1

Port E	Selectable Pin functions	
	Function 1	Function 2
GPF0	Input/output	EXTINT0
GPF1	Input/output	EXTINT1
GPF2	Input/output	EXTINT2
GPF3	Input/output	EXTINT3
GPF4	Input/output	EXTINT4
GPF5	Input/output	EXTINT5
GPF6	Input/output	EXTINT6
GPF7	Input/output	EXTINT7

NOTES:

1. The underlined function name is selected just after a reset.
2. IICSDAn and IICSCLKn pins are open-drain pin. So, this pin needs pull-up resistors when used as output port(GPD[3:0]).

PORT CONTROL DESCRIPTIONS

Port Configuration Register (PCONA – PCONF)

In S3C2800, most pins are multiplexed, and the PCONn (port control register) determines which function is used for each pin.

If GPF0 – GPF7 is used for the wakeup signal in power down mode, these ports must be configured for interrupt mode.

Port Data Register (PDATA – PDATF)

If Ports are configured as output ports, data can be written to the corresponding bit of PDATn. If Ports are configured as input ports, the data can be read from the corresponding bit of PDATn.

Port Pull-Up Register (PUPA, PUPC-PUPF)

The port pull-up register controls the pull-up resistor enable/disable of each port group except port B. When the corresponding bit is 0, the pull-up resistor of the pin is enabled. When 1, the pull-up resistor is disabled.

When the port pull-up resistors are enabled, they are effective only for input mode. In output mode, the pull-up resistors are ineffective even if they are enabled.

External Interrupt Control Register

The 8 external interrupts support various trigger mode: the trigger mode can be configured as low-level trigger, high-level trigger, falling-edge trigger, rising-edge trigger, and both edge trigger.

Because each external interrupt pin has an integrated digital noise filter, the interrupt controller can recognize the request signal that lasts longer than 3 clocks.

GPIO PORT SPECIAL FUNCTION REGISTERS

PORT A CONTROL REGISTERS (PCONA, PDATA, PUPA)

Register	Address	R/W	Description	Reset Value
PCONA	0x1010 0000	R/W	Configures the pins of port A	0x0000 FFFF
PDATA	0x1010 0004	R/W	The data register for port A	Undefined
PUPA	0x1010 0008	R/W	Pull-up resistor control register for port A	0x0000 0000

PCONA	Bit	Description
GPA7	[15:14]	00 = Input 01 = Output 1x = nDCAS1
GPA6	[13:12]	00 = Input 01 = Output 1x = nDCAS0
GPA5	[11:10]	00 = Input 01 = Output 1x = nSDCS3/nDRAS3
GPA4	[9:8]	00 = Input 01 = Output 1x = nSDCS2/nDRAS2
GPA3	[7:6]	00 = Input 01 = Output 1x = nSDCS1/nDRAS1
GPA2	[5:4]	00 = Input 01 = Output 1x = nSCS3
GPA1	[3:2]	00 = Input 01 = Output 1x = nSCS2
GPA0	[1:0]	00 = Input 01 = Output 1x = nSCS1

PDATA	Bit	Description
GPA[7:0]	[7:0]	When the port is configured as output port, the pin state is the same as the corresponding bit. When the port is configured as functional pin, the undefined value will be read.

PUPA	Bit	Description
GPA[7:0]	[7:0]	0: the pull-up resistor of the corresponding port pin is enabled. 1: the pull-up resistor is disabled.

PORT B CONTROL REGISTERS (PCONB, PDATB)

Register	Address	R/W	Description	Reset Value
PCONB	0x1010 000C	R/W	Configures the pins of port B	0x0000 0FFF
PDATB	0x1010 0010	R/W	The data register for port B	Undefined

PCONB	Bit	Description
GPB7	[15:14]	00 = Input 01 = Output 1x = AHBCLK out
GPB6	[13:12]	00 = Input 01 = Output 1x = nWAIT
GPB5	[11:10]	00 = Input 01 = Output 1x = nBE3/nWBE3/DQM3
GPB4	[9:8]	00 = Input 01 = Output 1x = nBE2/nWBE2/DQM2
GPB3	[7:6]	00 = Input 01 = Output 1x = nBE1/nWBE1/DQM1
GPB2	[5:4]	00 = Input 01 = Output 1x = nBE0/nWBE0/DQM0
GPB1	[3:2]	00 = Input 01 = Output 1x = nDCAS3/nSDRAS
GPB0	[1:0]	00 = Input 01 = Output 1x = nDCAS2/nSDCAS

PDATB	Bit	Description
GPB[7:0]	[7:0]	When the port is configured as output port, the pin state is the same as the corresponding bit. When the port is configured as functional pin, the undefined value will be read.

NOTE: Port B does not have built-in pull-up resistors.

PORT C CONTROL REGISTERS (PCONC, PDATC, PUPC)

Register	Address	R/W	Description	Reset Value
PCONC	0x1010 0018	R/W	Configures the pins of port C	0x0000 0000
PDATC	0x1010 001C	R/W	The data register for port C	Undefined
PUPC	0x1010 0020	R/W	Pull-up resistor control register for port C	0x0000 0000

PCONC	Bit	Description
GPC3	[7:6]	Read only. 0x = Input (ENDIAN) 1x = Output ENDIAN is used only when nRESET is Low. Endian value is latched only at the rising edge of nRESET: when nRESET is Low, the ENDIAN(GPC3) pin operates in input mode; nRESET becomes High, the ENDIAN pin will automatically switch to output mode.
GPC2	[5:4]	0x = Input 1x = Output
GPC1	[3:2]	0x = Input 1x = Output
GPC0	[1:0]	0x = Input 1x = Output

PDATC	Bit	Description
GPC[3:0]	[3:0]	When the port is configured as output port, the pin state is the same as the corresponding bit. When the port is configured as functional pin, the undefined value will be read.

PUPC	Bit	Description
GPC[2:0]	[2:0]	0 : the pull-up resistor of the corresponding port pin is enabled. 1 : the pull-up resistor is disabled. GPC3 does not have programmale pull-up resistor.

PORT D CONTROL REGISTERS (PCOND, PDATD, PUPD)

Register	Address	R/W	Description	Reset Value
PCOND	0x1010 0024	R/W	Configures the pins of port D	0x0000 0000
PDATD	0x1010 0028	R/W	The data register for port D	Undefined
PUPD	0x1010 002C	R/W	Pull-up resistor control register for port D	0x0000 0000

PCOND	Bit	Description
GPD7	[15:14]	00 = Input 01 = Output 1x = nRTS0
GPD6	[13:12]	00 = Input 01 = Output 1x = nCTS0
GPD5	[11:10]	00 = Input 01 = Output 1x = TxD0
GPD4	[9:8]	00 = Input 01 = Output 1x = RxD0
GPD3	[7:6]	00 = Input 01 = Output 1x = IICSCLK1
GPD2	[5:4]	00 = Input 01 = Output 1x = IICSDA1
GPD1	[3:2]	00 = Input 01 = Output 1x = IICSCLK0
GPD0	[1:0]	00 = Input 01 = Output 1x = IICSDA0

PDATD	Bit	Description
GPD[7:0]	[7:0]	When the port is configured as output port, the pin state is the same as the corresponding bit. When the port is configured as functional pin, the undefined value will be read.

PUPD	Bit	Description
GPD[7:0]	[7:0]	0: the pull-up resistor of the corresponding port pin is enabled. 1: the pull-up resistor is disabled.

PORT E CONTROL REGISTERS (PCONE, PDATE, PUPE)

Register	Address	R/W	Description	Reset Value
PCONE	0x1010 0030	R/W	Configures the pins of port E	0x0000 0000
PDATE	0x1010 0034	R/W	The data register for port E	Undefined
PUPE	0x1010 0038	R/W	Pull-up resistor control register for port E	0x0000 0000

PCONE	Bit	Description
GPE7	[15:14]	00 = Input 01 = Output 1x = nXDACK1
GPE6	[13:12]	00 = Input 01 = Output 1x = nXDREQ1
GPE5	[11:10]	00 = Input 01 = Output 1x = nXDACK0
GPE4	[9:8]	00 = Input 01 = Output 1x = nXDREQ0
GPE3	[7:6]	00 = Input 01 = Output 1x = nRTS1
GPE2	[5:4]	00 = Input 01 = Output 1x = nCTS1
GPE1	[3:2]	00 = Input 01 = Output 1x = TxD1
GPE0	[1:0]	00 = Input 01 = Output 1x = RxD1

PDATE	Bit	Description
GPE[7:0]	[7:0]	When the port is configured as an output port, the pin state is the same as the corresponding bit. When the port is configured as a functional pin, the undefined value will be read.

PUPE	Bit	Description
GPE[7:0]	[7:0]	0: the pull-up resistor of the corresponding port pin is enabled. 1: the pull-up resistor is disabled.

PORT F CONTROL REGISTERS (PCONF, PDATF, PUPF)

If GPF0 - GPF7 are used for wake-up interrupt in the IDLE mode, the ports have to be set in external interrupt mode before entering IDLE mode.

Register	Address	R/W	Description	Reset Value
PCONF	0x1010 003C	R/W	Configures the pins of port F	0x0000 0000
PDATF	0x1010 0040	R/W	The data register for port F	Undefined
PUPF	0x1010 0044	R/W	Pull-up resistor control register for port F	0x0000 0000

PCONF	Bit	Description
GPF7	[15:14]	00 = Input 01 = Output 1x = EXTINT7
GPF6	[13:12]	00 = Input 01 = Output 1x = EXTINT6
GPF5	[11:10]	00 = Input 01 = Output 1x = EXTINT5
GPF4	[9:8]	00 = Input 01 = Output 1x = EXTINT4
GPF3	[7:6]	00 = Input 01 = Output 1x = EXTINT3
GPF2	[5:4]	00 = Input 01 = Output 1x = EXTINT2
GPF1	[3:2]	00 = Input 01 = Output 1x = EXTINT1
GPF0	[1:0]	00 = Input 01 = Output 1x = EXTINT0

PDATF	Bit	Description
GPF[7:0]	[7:0]	When the port is configured as an input port, the corresponding bit is the pin state. When the port is configured as output port, the pin state is the same as the corresponding bit. When the port is configured as a functional pin, the undefined value will be read.

PUPF	Bit	Description
GPF[7:0]	[7:0]	0: the pull-up resistor of the corresponding port pin is enabled. 1: the pull-up resistor is disabled.

EXTINTR (EXTERNAL INTERRUPT CONTROL REGISTER)

The EXTINTR register selects the trigger types among various level or edge trigger mode of the external interrupt.

Register	Address	R/W	Description	Reset Value
EXTINTR	0x1010 0048	R/W	External Interrupt control register	0x0000 0000

EXTINTR	Bit	Description
EXTINT7	[30:28]	Trigger mode of the EXTINT7. 000 = Low level interrupt 001 = High level interrupt 01x = Falling edge triggered 10x = Rising edge triggered 11x = Both edge triggered
EXTINT6	[26:24]	Trigger mode of the EXTINT6. 000 = Low level interrupt 001 = High level interrupt 01x = Falling edge triggered 10x = Rising edge triggered 11x = Both edge triggered
EXTINT5	[22:20]	Trigger mode of the EXTINT5. 000 = Low level interrupt 001 = High level interrupt 01x = Falling edge triggered 10x = Rising edge triggered 11x = Both edge triggered
EXTINT4	[18:16]	Trigger mode of the EXTINT4. 000 = Low level interrupt 001 = High level interrupt 01x = Falling edge triggered 10x = Rising edge triggered 11x = Both edge triggered
EXTINT3	[14:12]	Trigger mode of the EXTINT3. 000 = Low level interrupt 001 = High level interrupt 01x = Falling edge triggered 10x = Rising edge triggered 11x = Both edge triggered
EXTINT2	[10:8]	Trigger mode of the EXTINT2. 000 = Low level interrupt 001 = High level interrupt 01x = Falling edge triggered 10x = Rising edge triggered 11x = Both edge triggered
EXTINT1	[6:4]	Trigger mode of the EXTINT1. 000 = Low level interrupt 001 = High level interrupt 01x = Falling edge triggered 10x = Rising edge triggered 11x = Both edge triggered
EXTINT0	[2:0]	Trigger mode of the EXTINT0. 000 = Low level interrupt 001 = High level interrupt 01x = Falling edge triggered 10x = Rising edge triggered 11x = Both edge triggered

NOTES:

1. Because each external interrupt pins has a digital filter, the interrupt controller can recognize a request signal that is longer than 3 clocks.
2. If users want to change the trigger mode in the external interrupt mode, users are first required to switch the corresponding pin to input mode and then change the trigger mode.

SPECIAL PULL-UP RESISTOR CONTROL REGISTER(SPUCR)

DATA[31:0] pin pull-up resistor can be controlled by SPUCR register.

When CPU does not require accesses to external memory for data (e.g., all required data resides in the cache), the data bus DATA[31:0] becomes Hi-Z state and unnecessary power consumption will result. In this case, enabling pull-up resistor of DATA[31:0] helps the reduction of power consumption. In particular, it is recommended to enable the pull-up resistor for a great power reduction in IDLE mode when no data access is necessary, However, the pull-up resistor should be turned off for normal data access operation in other cases.

Register	Address	R/W	Description	Reset Value
SPUCR	0x1010 004C	R/W	Special pull-up resistor control register for DATA ports.	0x0000 0000

PCONG	Bit	Description
SPUCR1	[1]	0 = DATA[31:16] port pull-up resistor is enabled 1 = DATA[31:16] port pull-up resistor is disabled
SPUCR0	[0]	0 = DATA[15:0] port pull-up resistor is enabled 1 = DATA[15:0] port pull-up resistor is disabled

10 TIMER

OVERVIEW

The S3C2800 has three 16-bit timers, Timers 0, 1, 2, that operate in interval mode, interrupt-based or DMA-based mode.

Each timer is composed of an 8-bit prescaler, a clock-divider (1/4, 1/8, 1/16, 1/32), and a 16-bit down-counter. The programmable 8-bit prescaler divides the PCLK signal depending on the prescaler value in TMDATAn register and the clock divider further divides the PCLK to generate the timer clock.

Each timer has its own 16-bit down-counter that is driven by the timer clock. When the down-counter occurs underflow, the timer interrupt request is generated to inform the CPU that the timer has expired. When the timer counter occurs underflow, the value of corresponding TMDATAn is automatically reloaded into the down-counter to continue the next iteration. However, if the timer is stopped, by clearing the count enable bit of TMCONn while the timer is running, the value of TMDATAn will not be reloaded into the counter.

FEATURE

- Three 16-bit timers with DMA-based or interrupt-based operation
- Three 8-bit prescalers and three 5-bit dividers
- Auto reload operation
- Maximum frequency source is 50MHz (PCLK)

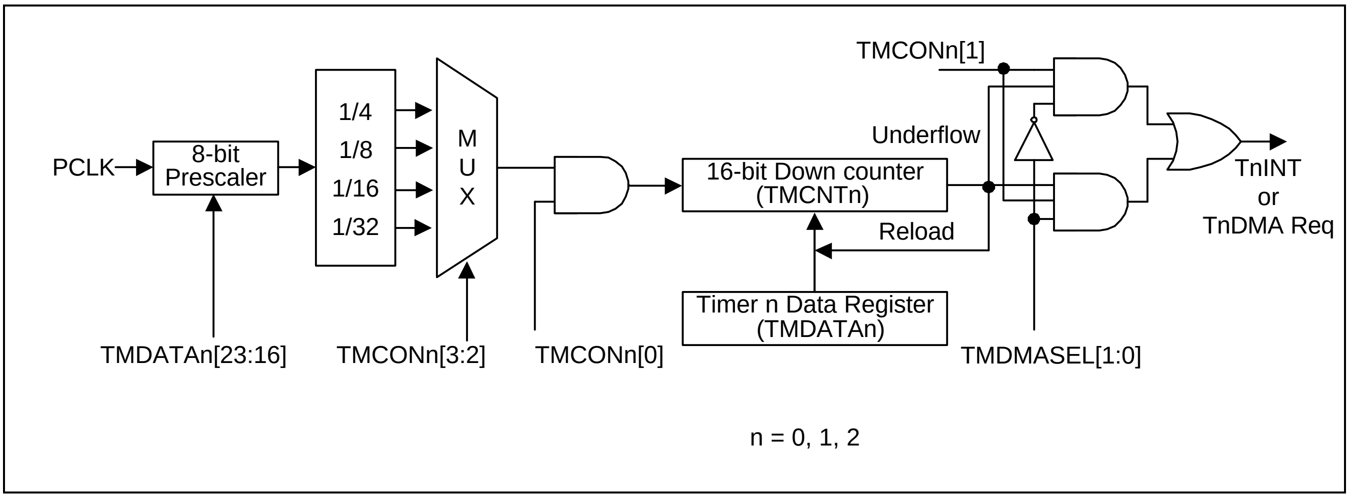


Figure 10-1. 16-bit Timer Block Diagram

16-BIT TIMER OPERATION

PRESCALER & DIVIDER

Examples of timer clock frequency for various combination of prescaler and divider are given in Table 10-1.

Table 10-1. Example for Interval Timing

Divider settings	Minimum resolution (prescaler = 1)	Maximum resolution (prescaler = 255)	Maximum interval (TCNTBn = 65535)
1/4 (PCLK = 50 MHz)	0.16 us (6.25 MHz)	20.48 us (48.83 KHz)	1.342 sec
1/8 (PCLK = 50 MHz)	0.32 us (3.125 MHz)	40.96 us (24.42 KHz)	2.684 sec
1/16 (PCLK = 50 MHz)	0.64 us (1.563 MHz)	81.92 us (12.21 KHz)	5.368 sec
1/32 (PCLK = 50 MHz)	1.28 us (0.782 MHz)	163.84 us (6.11 KHz)	10.736 sec

DMA REQUEST MODE

The timer can generate DMA request at a specified time interval. The timer keeps DMA request signal low until the timer receives the ACK signal. When receives the ACK signal, it makes the request signal inactive. In order to use the timer in DMA mode, DMA mode bits (TMCONn register) must be set. Only one of the timer can operate in DMA mode at a time, and others will operate in interrupt mode. If a timer is configured for DMA request mode, the timer does not generate an interrupt request.

DMA MODE CONFIGURATION AND DMA/INTERRUPT OPERATION

DMA mode	DMA request	Timer0 INT	Timer1 INT	Timer2 INT
00	No select	ON	ON	ON
01	Timer0	OFF	ON	ON
10	Timer1	ON	OFF	ON
11	Timer2	ON	ON	OFF

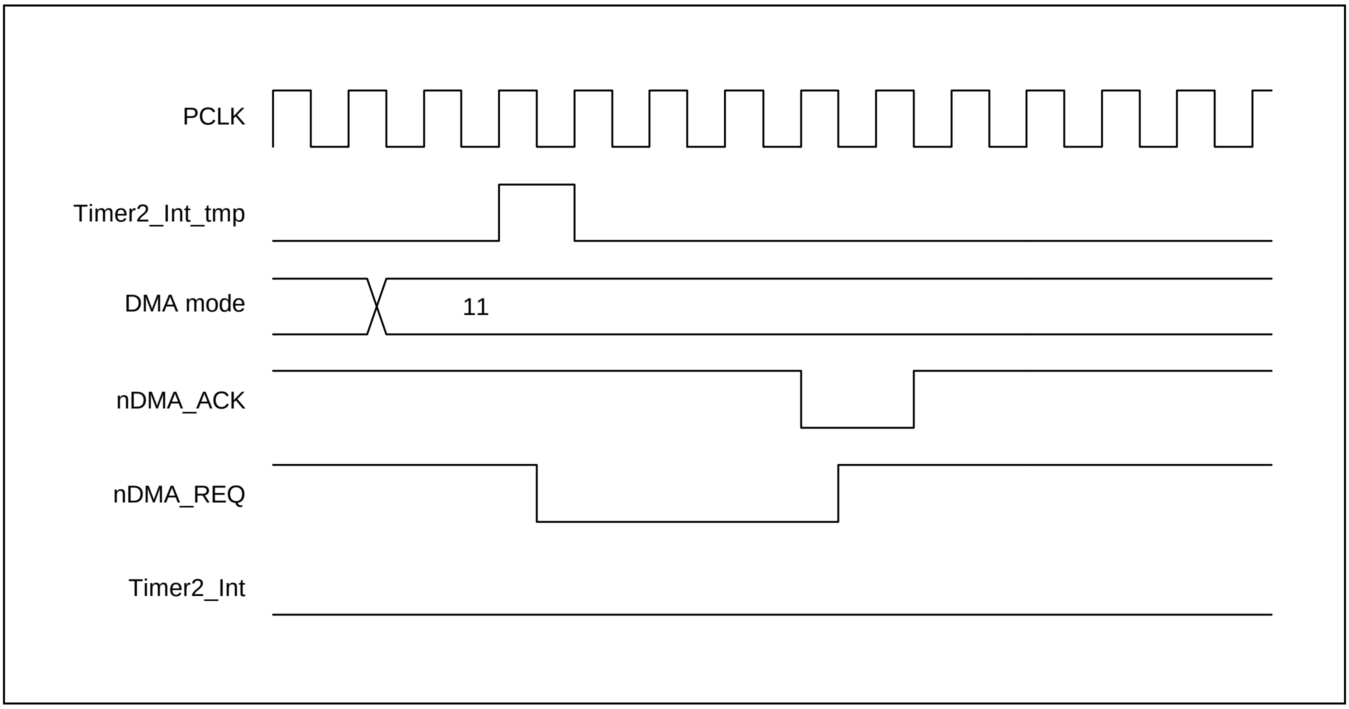


Figure 10-2. The Timer2 DMA mode operation

TIMER SPECIAL FUNCTION REGISTERS

TIMER DMA SELECTION REGISTER (TMDMASEL)

Register	Address	R/W	Description	Reset Value
TMDMASEL	0x1013 000C	R/W	DMA or Interrupt mode selecton register	0x0000 0000

TMDMASEL	Bit	Description	Initial State
DMA mode	[1:0]	Select DMA request channel 00 = No select(All interrupt) 01 = Timer0 10 = Timer1 11 = Timer2	0

TIMER CONTROL REGISTER (TMCONn)

Timer input clock Frequency = APBCLK / {prescaler value+1} / {divider value}
{prescaler value} = 1–255 (TMDATAn[23:16])
{divider value} = 4, 8, 16, 32

Register	Address	R/W	Description	Reset Value
TMCON0	0x1013 0000	R/W	Timer 0 control register	0x0000 0800
TMCON1	0x1014 0000	R/W	Timer 1 control register	0x0000 0800
TMCON2	0x1015 0000	R/W	Timer 2 control register	0x0000 0800

TMCONn	Bit	Description	Initial State
MUX	[3:2]	Select MUX input 00 = 1/4 01 = 1/8 10 = 1/16 11 = 1/32	00
Interrupt/DMA Enable	[1]	Interrupt or DMA Enable 0 = Disable 1 = Enable	0
Count Enable	[0]	Timer down counter run or stop 0 = Stop 1 = Run This bit enables or disables timer. When the bit is set to “0”, the 16-bit down counter is clear to “0x0000”, and then it is stops. When set to “1”, the timer down-counter starts counting after reloading the timer data and pre-scaler value. The down-counter decrements itself by one on accepting every timer clock.	0

NOTE: When the Timer data or prescaler value need updating, first stop the timer counter.

TIMER DATA REGISTER (TMDATAn)

The timer data registers, TMDATA0, TMDATA1, and TMDATA2, contain a value that specifies the time-out period for each timer. The formula for calculating time-out period is (Timer data + 1) cycles.

Register	Address	R/W	Description	Reset Value
TMDATA0	0x1013 0004	R/W	Timer 0 data register	0x0080 FFFF
TMDATA1	0x1014 0004	R/W	Timer 1 data register	0x0080 FFFF
TMDATA2	0x1015 0004	R/W	Timer 2 data register	0x0080 FFFF

TMDATAn	Bit	Description	Initial State
Pre-scaler	[23:16]	These 8 bits determine prescaler value (1 – 255) 0 = not supported.	0x80
Timer data value	[15:0]	Timer data value This field specifies the time-out period of the corresponding timer. The time-out period is calculated as (Timer data + 1) cycles. Therefore, a maximum time-out period of 65,536 cycles is possible (when the timer data value is 0xffff). The minimum time-out period (2 cycles) is obtained when the timer data register field has the value 0x0001h.	0x0000 FFFF

TIMER COUNT REGISTER (TMCNTn)

The timer count registers, TMCNT0, TMCNT1, and TMCNT2, are 16-bit counters for timer 0, 1, and 2, respectively. The value of timer count registers decrease on each time clock.

Register	Address	R/W	Description	Reset Value
TMCNT0	0x1013 0008	R	Timer 0 count register	0x0000 FFFF
TMCNT1	0x1014 0008	R	Timer 1 count register	0x0000 FFFF
TMCNT2	0x1015 0008	R	Timer 2 count register	0x0000 FFFF

TMCNTn	Bit	Description	Initial State
Timer count value	[15:0]	Timer down-counter This field keeps the actual timer count value. When the timer is enabled by setting TMCONn[0] = 1, timer data value in TMDATAn register is loaded into this field and the down-counting process begins. An interrupt is generated (if enabled) and this filed is automatically re-loaded with timer data value when the underflow occurs.	0x0000 FFFF

NOTES

11 UART

OVERVIEW

The S3C2800 UART (Universal Asynchronous Receiver and Transmitter) unit provides two independent serial GPIO ports, each of which can operate in interrupt-based or DMA-based mode. The UART can generate an interrupt or DMA request to transfer data between CPU and UART. It supports bit rates of up to 230.4Kbps. Each UART channel contains two 16-byte FIFOs for receive and transmit.

The UART supports programmable baud-rates, infra-red (IR) transmit/receive, one or two stop bit insertion, 5-bit, 6-bit, 7-bit or 8-bit data and parity checking.

Each UART is composed of a baud-rate generator, transmitter, receiver and control unit, as shown in Figure11-1. The baud-rate generator is clocked by PCLK.

FEATURE

- RxD0,TxD0,RxD1,TxD1 with DMA-based or interrupt-based operation
- UART 0 with IrDA 1.0 and 16-byte FIFO
- UART 1 with IrDA 1.0 and 16-byte FIFO
- Supports transmit/receive handshake

Table 11-1. Maximum Baud-Rate for Each Input Clock

PCLK	Baud-rate	PCLK	Baud-rate	PCLK	Baud-rate	PCLK	Baud-rate
50.0MHz	115.2Kbps	37.5MHz	230.4Kbps	25.0MHz	57.6Kbps	12.5MHz	38.4Kbps
47.5MHz	230.4Kbps	35.0MHz	115.2Kbps	22.5MHz	230.4Kbps	10.0MHz	57.6Kbps
45.0MHz	230.4Kbps	32.5MHz	230.4Kbps	20.0MHz	115.2Kbps	7.5MHz	230.4Kbps
42.5MHz	115.2Kbps	30.0MHz	230.4Kbps	17.5MHz	57.6Kbps	5.0MHz	38.4Kbps
40.0MHz	230.4Kbps	27.5MHz	115.2Kbps	15.0MHz	230.4Kbps		

BLOCK DIAGRAM

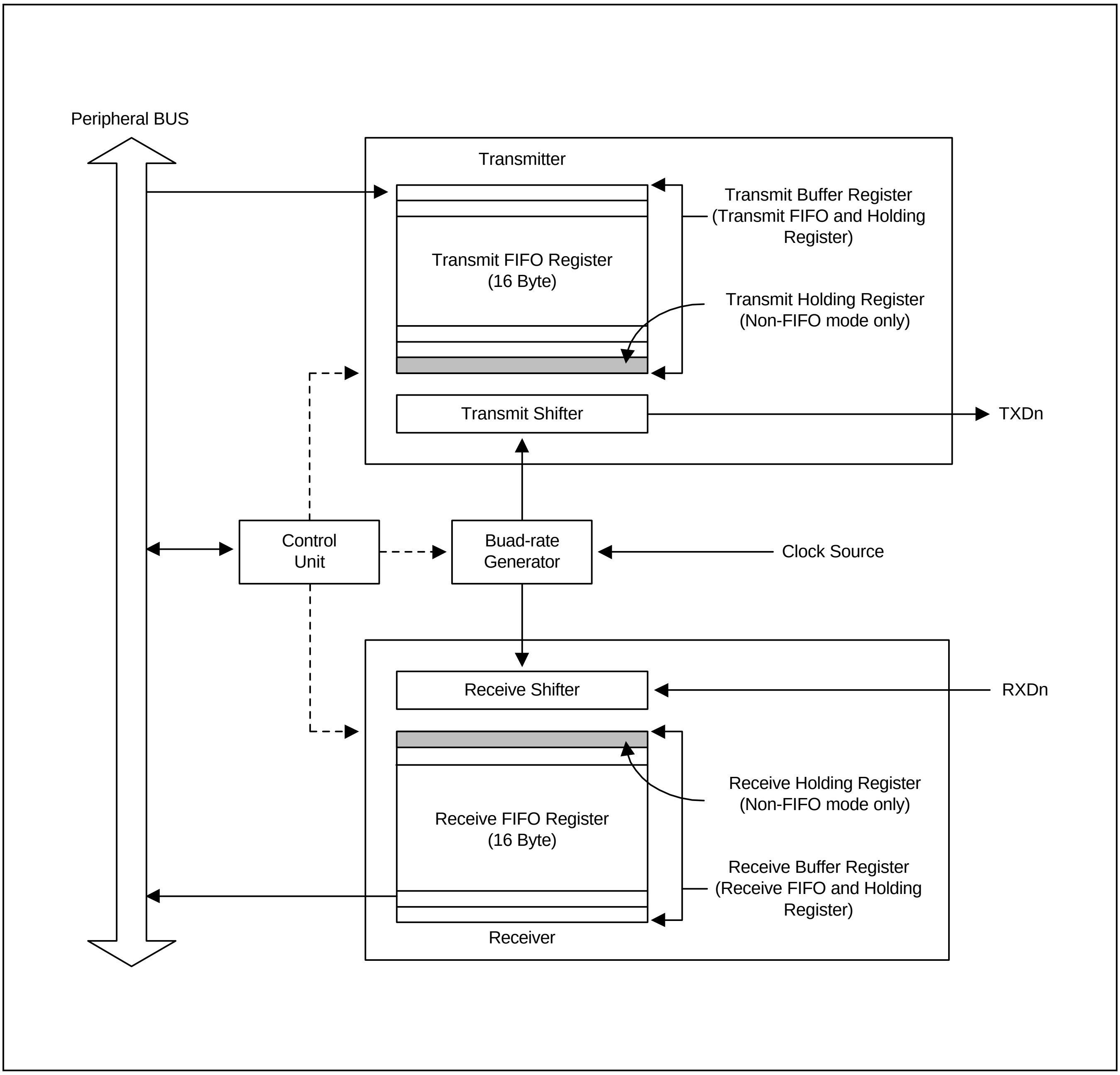


Figure 11-1. UART Block Diagram (with FIFO)

UART OPERATION

The following sections describe the UART operations such as data transmission, data reception, interrupt generation, baud-rate generation, loopback mode, infra-red mode, and auto flow control.

DATA TRANSMISSION

The data frame for transmission is programmable. It consists of a start bit, 5 to 8 data bits, an optional parity bit and 1 to 2 stop bits, which can be specified by the line control register (ULCONn). The transmitter can also produce the break condition. The break condition forces the serial output to logic 0 state for one frame transmission time. This block transmit break signal after the present transmissive word transmits perfectly. After the break signal transmission, it continuously transmits data into the Tx FIFO (Tx holding register in the case of Non-FIFO mode).

DATA RECEPTION

Like the transmission, the data frame for reception is also programmable. It consists of a start bit, 5 to 8 data bits, an optional parity bit and 1 to 2 stop bits in the line control register (ULCONn). The receiver can detect overrun error, parity error, frame error and break condition, each of which can set an error flag.

— The overrun error indicates that new data has overwritten the old data before the old data has been read.

— The parity error indicates that the receiver has detected an unexpected parity condition.

— The frame error indicates that the received data does not have a valid stop bit.

— The break condition indicates that the RxDn input is held in the logic 0 state for the duration longer than one frame transmission time.

Receive time-out condition occurs when it does not receive data during the 3 word time frame (This interval follows the setting of Word Length bit) and the Rx FIFO is not empty in the FIFO mode.

AUTO FLOW CONTROL(AFC)

S3C2800's UART supports auto flow control with nRTS and nCTS signals for UART to UART connection. On the other hand, when connecting UART to a modem, it is recommended to disable auto flow control bit in UMCONn register and control the signal of nRTS by software.

In AFC, nRTS is controlled by condition of the receiver and operation of transmitter is controlled by the nCTS signal. The UART's transmitter transfers the data in FIFO only when nCTS signal is active(In AFC, nCTS means that the other UART's FIFO is ready to receive data). Before the UART receives data, nRTS has to be activated when its receive FIFO has a spare more than 2-byte and has to be inactivated when its receive FIFO has a spare under 1-byte(In AFC, nRTS means that its own receive FIFO is ready to receive data).

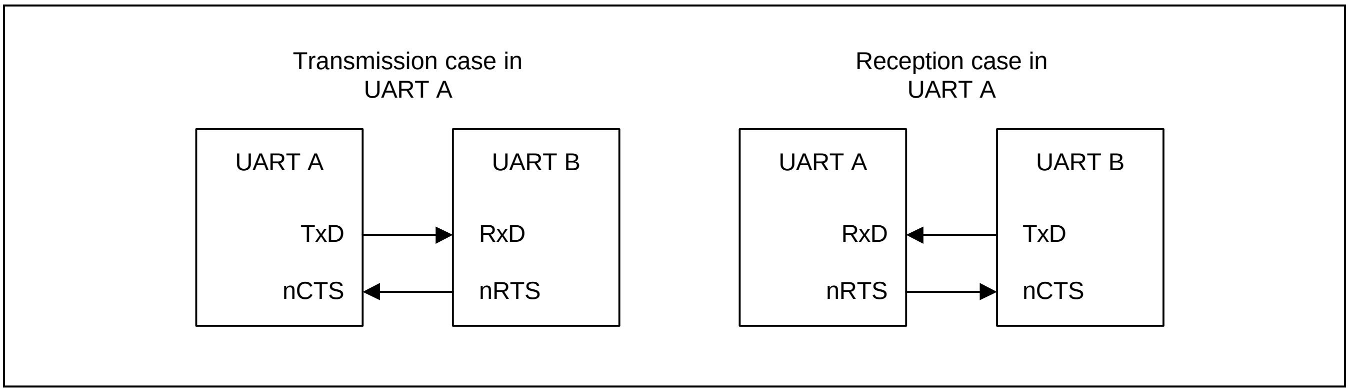


Figure 11-2. UART AFC interface

NON AUTO-FLOW CONTROL (CONTROLLING NRTS AND NCTS BY S/W)

Rx operation

1. Select receive mode (Interrupt or DMA mode)
2. Check the value of Rx FIFO count in UFSTATn register. If the value is less than 15, users have to set the value of UMCONn[0] to '1'(activate nRTS), and if it is equal or larger than 15 users have to set the value to '0'(inactivate nRTS).
3. Repeat item 2.

Tx operation

1. Select transmit mode (Interrupt or DMA mode)
2. Check the value of UMSTATn[0]. If the value is '1'(nCTS is activated), users write the data to Tx FIFO register.

RS-232C INTERFACE

If users connect to modem interface (not a null modem), nRTS, nCTS, nDSR, nDTR, DCD and nRI signals are needed. In this case, users have to control these signals using GPIO ports by software because the AFC does not support the RS-232C interface.

INTERRUPT/DMA REQUEST GENERATION

Each UART has seven status(Tx/Rx/Error) signals: overrun error, parity error, frame error, break, receive buffer data ready, transmit buffer empty, and transmit shifter empty, all of which are indicated by the corresponding UART status register (UERSTATn /UTRSTATn).

The overrun error, parity error, frame error and break condition are referred to as the receive error status, each of which can cause the receive error status interrupt request, if the Rx error status interrupt enable bit (UCONn[6]) is set. When a Rx error status interrupt occurs, the types of Rx error can be identified by reading UERSTSTn.

When interrupt request mode is selected:

In the FIFO mode, and Rx/Tx interrupt is generated when the number of received/transmitted data reaches Rx/Tx FIFO trigger level, respectively. On the other hand, and interrupt is generated on data receive in the Rx buffer for Non-FIFO Rx mode and when Tx buffer becomes empty after transmitting the data for Non-FIFO Tx mode.

When DMA request mode is selected:

If the Rx/Tx mode in the control registers (UCONn) are selected as the DMA request mode, then DMA request is generated instead of Rx or Tx interrupt in the situation mentioned above.

Table 11-2. Interrupts In Connection With FIFO

Type	FIFO Mode	Non-FIFO Mode
Rx interrupt	Each time receive data reaches the trigger level of receive FIFO, the Rx interrupt will be generated. When the number of data in FIFO does not reaches Rx FIFO trigger level and does not receive data during 3 word time (This interval follows the setting of word length bit), the Rx interrupt will be generated (receive time out).	Each time receive data becomes full, the receive holding register, generates an interrupt.
Tx interrupt	Each time transmit data reaches the trigger level of transmit FIFO (Tx FIFO trigger level), the Tx interrupt will be generated.	Each time transmit data become empty, the transmit holding register generates an interrupt.
Error interrupt	Frame error, parity error, and break signal are detected, and will generate an error interrupt. When it gets to the top of the receive FIFO without reading out data in it, the error interrupt will be generated (overrun error).	All errors generate an error interrupt immediately. However if another error occurs at the same time, only one interrupt is generated.

UART ERROR STATUS FIFO

UART has the status FIFO associated with the Rx FIFO register. The status FIFO indicates which data, among FIFO registers, is received with an error. The error interrupt will be issued only when the data, which has an error, is ready to read out. To clear the status FIFO, the URXHn, with an error and UERSTATn must be read out.

For example, it is assumed that the UART FIFO receives A, B, C, D, E characters sequentially and the frame error occurs while receiving the 'B' , and the parity error occurs while receiving 'D'.

Although the actual UART error occurred, the error interrupt will not be generated because the character, which was received with an error, has not been read yet. The error interrupt will occur when the character is read out.

Time	Sequence flow	Error interrupt	Note
#0	When no character is read out	–	
#1	After A is read out	The frame error(in B) interrupt occurs	The 'B' has to be read out
#2	After B is read out	–	
#3	After C is read out	The parity error(in D) interrupt occurs	The 'D' has to be read out
#4	After D is read out	–	
#5	After E is read out	–	

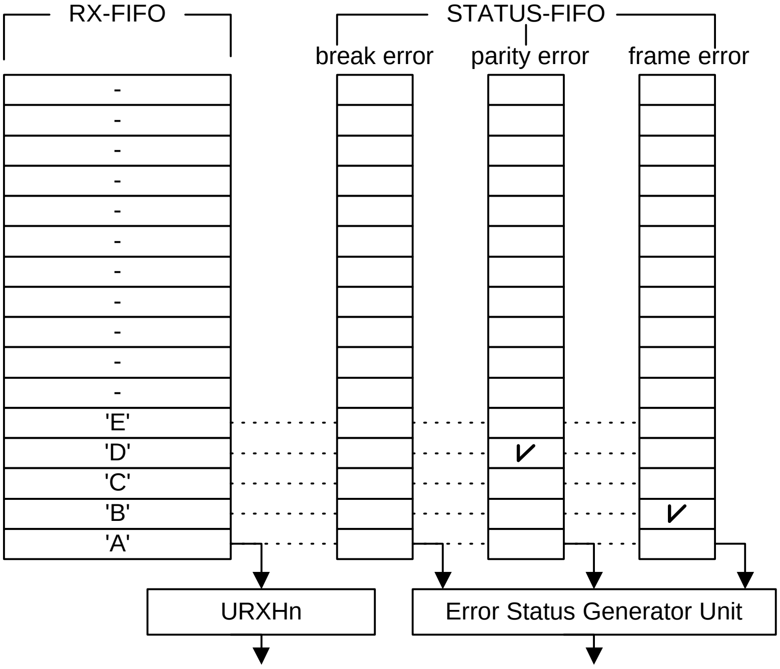


Figure 11-3. The Case that UART Receives 5 Characters Including 2 Errors

BAUD-RATE GENERATION

Each UART's baud-rate generator provides the serial clock for transmitter and receiver. The source clock for the baud-rate generator can be selected with the S3C2800's internal APB bus clock (PCLK). The baud-rate clock is generated by dividing the source clock by 16 and a 16-bit divisor specified by the UART baud-rate divisor register (UBRDIVn). The UBRDIVn can be determined as follows:

UBRDIVn = (round_off)(PCLK/(bps x 16)) -1

where the divisor should be from 1 to (2^{16}-1). For example, if the baud-rate is 230400 bps and PCLK is 50 MHz , UBRDIVn is:

UBRDIVn = (int)(50000000/(230400 x 16)+0.5) -1
= (int)(13.6+0.5) -1
= 14 -1 = 13

LOOP-BACK MODE

The S3C2800 UART provides a test mode referred to as the loopback mode, to aid in isolating faults in the communication link. In this mode, the transmitted data is immediately received. This feature allows the processor to verify the internal transmit and to receive the data path of each SIO channel. This mode can be selected by setting the loopback-bit in the UART control register (UCONn).

BREAK CONDITION

The break is defined as a continuous low level signal for one frame transmission time on the transmit data output.

IR (INFRA-RED) MODE

The S3C2800 UART block supports infra-red (IR) transmission and reception, which can be selected by setting the infra-red-mode bit in the UART line control register (ULCONn). The implementation of the mode is shown in Figure 11-4.

In IR transmit mode, the transmit period is pulsed at a rate of 3/16, the normal serial transmit rate (when the transmit data bit is zero); In IR receive mode, the receiver must detect the 3/16 pulsed period to recognize a zero value (refer to the frame timing diagrams shown in Figure 11-6 and 11-7).

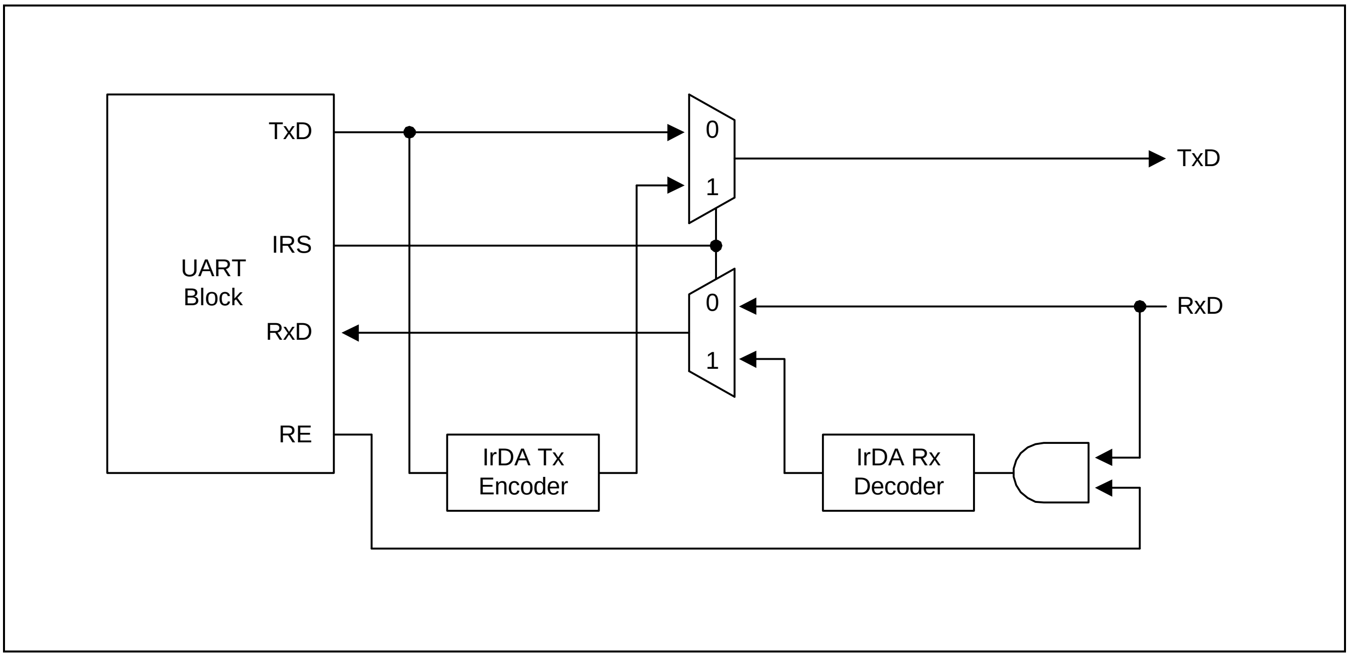


Figure 11-4. IrDA Function Block Diagram

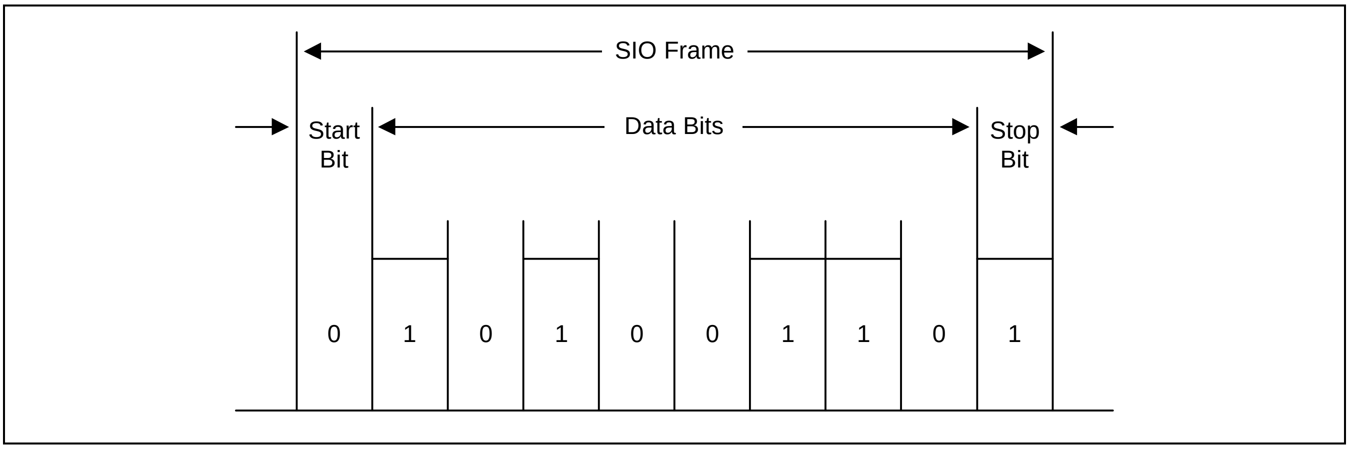


Figure 11-5. Serial I/O Frame Timing Diagram (Normal UART)

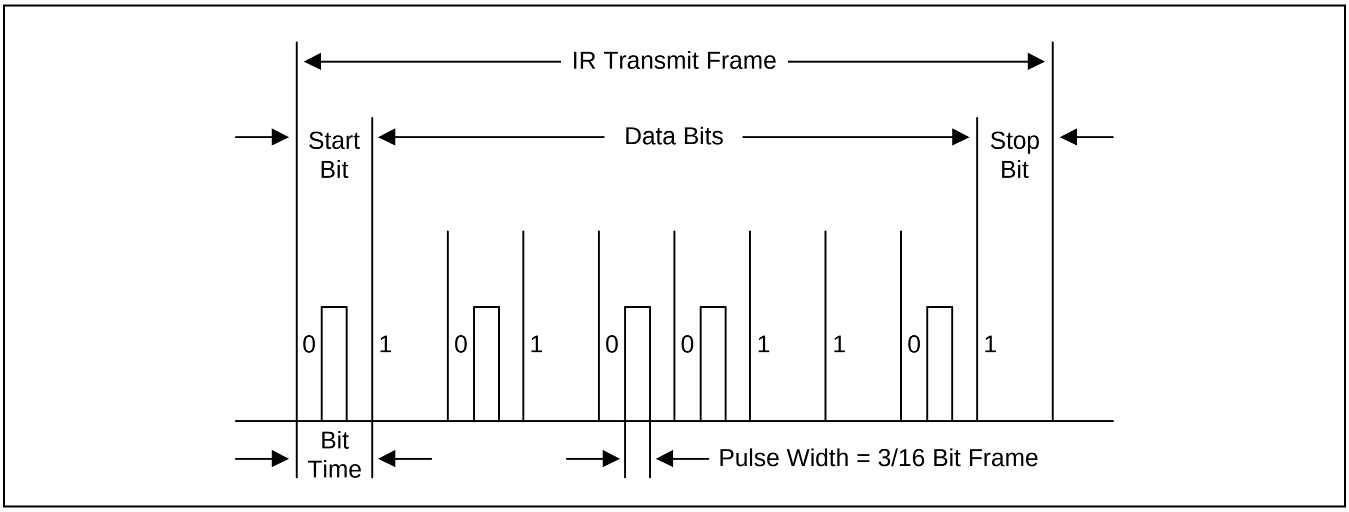


Figure 11-6. Infra-Red Transmit Mode Frame Timing Diagram

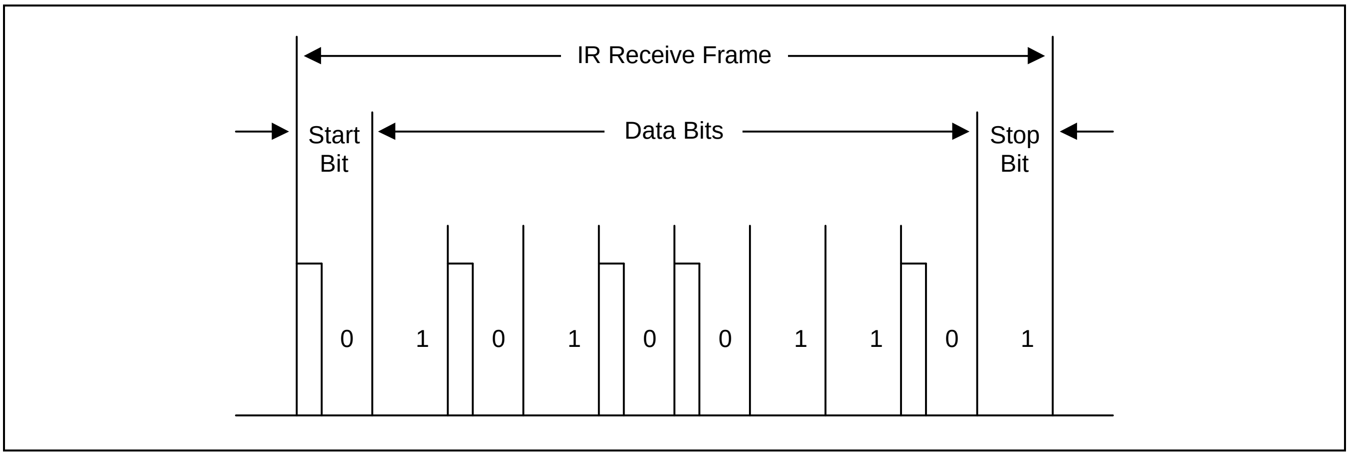


Figure 11-7. Infra-Red Receive Mode Frame Timing Diagram

UART SPECIAL FUNCTION REGISTERS

UART LINE CONTROL REGISTER (ULCONn)

There are two UART line control registers, ULCON0 and ULCON1 in the UART block.

Register	Address	R/W	Description	Reset Value
ULCON0	0x1017 0000	R/W	UART 0 line control register	0x0000 0000
ULCON1	0x1018 0000	R/W	UART 1 line control register	0x0000 0000

ULCONn	Bit	Description	Initial State
Infra-Red Mode	[7]	The Infra-Red mode determines whether or not to use the Infra-Red mode. 0 = Normal mode operation 1 = Infra-Red Tx/Rx mode	0
Parity Mode	[6:4]	The parity mode specifies how parity generation and checking are to be performed during UART transmit and receive operation. 0xx = No parity 100 = Odd parity 101 = Even parity 110 = Parity forced/checked as 1 111 = Parity forced/checked as 0	000
Reserved	[3]	Reserved	0
Number of stop bit	[2]	The number of stop bits specifies how many stop bits are to be used to signal end-of-frame. 0 = One stop bit per frame 1 = Two stop bit per frame	0
Word length	[1:0]	The word length indicates the number of data bits to be transmitted or received per frame. 00 = 5-bits 01 = 6-bits 10 = 7-bits 11 = 8-bits	00

UART CONTROL REGISTER (UCONn)

There are two UART control registers, UCON0 and UCON1 in the UART block.

Register	Address	R/W	Description	Reset Value
UCON0	0x1017 0004	R/W	UART 0 control register	0x0000 0000
UCON1	0x1018 0004	R/W	UART 1 control register	0x0000 0000

UCONn	Bit	Description	Initial State
Tx interrupt type	[9]	Interrupt request type 0 = Pulse (Interrupt is requested as soon as Tx buffer becomes empty in Non-FIFO mode or Tx FIFO reaches trigger level in FIFO mode) 1 = Level (Interrupt is requested while Tx buffer is empty in Non-FIFO mode or Tx FIFO has been being in trigger level in FIFO mode)	0
Rx interrupt type	[8]	Interrupt request type 0 = Pulse (Interrupt is requested the instant Rx buffer receives the data in Non-FIFO mode or reaches Rx FIFO trigger level in FIFO mode) 1 = Level (Interrupt is requested while Rx buffer is receiving data in Non-FIFO mode or reaches Rx FIFO trigger level in FIFO mode)	0
Rx time out interrupt	[7]	Enable/Disable Rx time out interrupt when UART FIFO is enabled. 0 = Disable 1 = Enable	0
Rx error status interrupt enable	[6]	This bit enables the UART to generate an interrupt if an exception, such as a break, frame error, parity error, or overrun error occurs during a receive operation. 0 = Do not generate receive error status interrupt 1 = Generate receive error status interrupt	0
Loop-back mode	[5]	Setting loop-back bit to 1 causes the UART to enter loop-back mode. This mode is provided for test purposes only. 0 = Normal operation 1 = Loop-back mode	0
Send break signal	[4]	Setting this bit causes the UART to send a break during 1 frame time. This bit is auto-cleared after sending break signal. 0 = Normal transmit 1 = Send break signal	0
Transmit mode	[3:2]	These two bits determine which function is currently able to write Tx data to the UART transmit holding register. 00 = Disable 01 = Interrupt request 1x = DMA request	00
Receive mode	[1:0]	These two bits determine which function is currently able to read data from UART receive buffer register. 00 = Disable, 01 = Interrupt request 1x = DMA request	00

UART FIFO CONTROL REGISTER (UFCONn)

There are two UART FIFO control registers, UFCON0 and UFCON1 in the UART block.

Register	Address	R/W	Description	Reset Value
UFCON0	0x1017 0008	R/W	UART 0 FIFO control register	0x0000 0000
UFCON1	0x1018 0008	R/W	UART 1 FIFO control register	0x0000 0000

UFCONn	Bit	Description	Initial State
Tx FIFO trigger level	[7:6]	These two bits determine trigger level of transmit FIFO. 00 = Empty 01 = 4-byte 10 = 8-byte 11 = 12-byte	00
Rx FIFO trigger level	[5:4]	These two bits determine trigger level of receive FIFO. 00 = 4-byte 01 = 8-byte 10 = 12-byte 11 = 16-byte	00
Reserved	[3]		0
Tx FIFO reset	[2]	This bit is auto-cleared after resetting FIFO 0 = Normal 1 = Tx FIFO reset	0
Rx FIFO reset	[1]	This bit is auto-cleared after resetting FIFO 0 = Normal 1 = Rx FIFO reset	0
FIFO enable	[0]	0 = FIFO disable 1 = FIFO mode	0

NOTE: When the UART does not get to FIFO trigger level and does not receive data during 3 word time in DMA receive mode with FIFO, the Rx interrupt will be generated(receive time out), and users should have to check the FIFO status and read out the rest.

UART MODEM CONTROL REGISTER (UMCONn)

There are two UART MODEM control registers, UMCON0 and UMCON1 in the UART block.

Register	Address	R/W	Description	Reset Value
UMCON0	0x1017 000C	R/W	UART 0 Modem control register	0x0000 0000
UMCON1	0x1018 000C	R/W	UART 1 Modem control register	0x0000 0000

UMCONn	Bit	Description	Initial State
AFC(Auto Flow Control)	[1]	0 = Disable 1 = Enable	0
Request to send	[0]	If AFC bit is enabled, this value will be ignored. In this case the S3C2400 will control nRTS automatically. If AFC bit is disabled, nRTS must be controlled by S/W. 0 = 'H' level(Inactivate nRTS) 1 = 'L' level(Activate nRTS)	0

UART TX/RX STATUS REGISTER (UTRSTATn)

There are two UART Tx/Rx status registers, UTRSTAT0 and UTRSTAT1 in the UART block.

Register	Address	R/W	Description	Reset Value
UTRSTAT0	0x1017 0010	R	UART 0 Tx/Rx status register	0x0000 0006
UTRSTAT1	0x1018 0010	R	UART 1 Tx/Rx status register	0x0000 0006

UTRSTATn	Bit	Description	Initial State
Transmit shifter empty	[2]	This bit is automatically set to 1 when the transmit buffer register has no valid data to transmit and the transmit shift register is empty. 0 = Not empty 1 = Transmit holding and shift register empty	1
Transmit FIFO empty/ Transmit buffer empty	[1]	This bit is automatically set to 1 when the transmit buffer register is empty. 0 =The buffer register is not empty 1 = Empty (In Non-FIFO mode, Interrupt or DMA is requested. In FIFO mode, Interrupt or DMA is requested, when Tx FIFO trigger level is set to 00 (Empty)) If the UART uses the FIFO, users should check Tx FIFO count bits and Tx FIFO full bit in the UFSTAT register instead of this bit.	1
Receive FIFO data ready/ Receive buffer data ready	[0]	This bit is automatically set to 1 whenever the receive buffer register contains valid data, received over the RXDn port. 0 = Empty 1 = The buffer register has a received data (In Non-FIFO mode, Interrupt or DMA is requested) If the UART uses the FIFO, users should check Rx FIFO Count bits and Rx FIFO Full bit in the UFSTAT register instead of this bit.	0

UART ERROR STATUS REGISTER (UERSTATn)

There are two UART Rx error status registers, UERSTAT0 and UERSTAT1 in the UART block.

Register	Address	R/W	Description	Reset Value
UERSTAT0	0x1017 0014	R	UART 0 Rx error status register	0x0000 0000
UERSTAT1	0x1018 0014	R	UART 1 Rx error status register	0x0000 0000

UERSTATn	Bit	Description	Initial State
Break detect	[3]	This bit is automatically set to 1 to indicate that a break signal has been received. 0 = No break receive 1 = Break receive(Interrupt is requested)	0
Frame error	[2]	This bit is automatically set to 1 whenever a frame error occurs during receive operation. 0 = No frame error during receive 1 = Frame error(Interrupt is requested)	0
Parity error	[1]	This bit is automatically set to 1 whenever a parity error occurs during receive operation. 0 = No parity error during receive 1 = Parity error(Interrupt is requested)	0
Overrun error	[0]	This bit is automatically set to 1 whenever an overrun error occurs during receive operation. 0 = No overrun error during receive 1 = Overrun error(Interrupt is requested)	0

NOTE: These bits (UERSATn[3:0]) are automatically cleared to 0 when you read the UART error status register.

UART FIFO STATUS REGISTER (UFSTATn)

There are two UART FIFO status registers, UFSTAT0 and UFSTAT1 in the UART block.

Register	Address	R/W	Description	Reset Value
UFSTAT0	0x1017 0018	R	UART 0 FIFO status register	0x0000 0000
UFSTAT1	0x1018 0018	R	UART 1 FIFO status register	0x0000 0000

UFSTATn	Bit	Description	Initial State
Tx FIFO full	[9]	This bit is automatically set to 1 whenever transmit FIFO is full during transmit operation 0 = 0-byte ≤ Tx FIFO data ≤ 15-byte 1 = Full	0
Rx FIFO full	[8]	This bit is automatically set to 1 whenever receive FIFO is full during receive operation 0 = 0-byte ≤ Rx FIFO data ≤ 15-byte 1 = Full	0
Tx FIFO count	[7:4]	Number of data in Tx FIFO	0
Rx FIFO count	[3:0]	Number of data in Rx FIFO	0

UART MODEM STATUS REGISTER (UMSTATn)

There are two UART modem status register, UMSTAT0 and UMSTAT1 in the UART block.

Register	Address	R/W	Description	Reset Value
UMSTAT0	0x1017 001C	R	UART 0 Modem status register	0x0000 0000
UMSTAT1	0x1018 001C	R	UART 1 Modem status register	0x0000 0000

UMSTATn	Bit	Description	Initial State
Delta CTS	[1]	This bit indicates that the nCTS input to S3C2800 has changed state since the last time it was read by CPU. (Refer to Fig. 11-8) 0 = Has not changed 1 = Has changed	0
Clear to send	[0]	0 = CTS signal is not activated(nCTS pin is high) 1 = CTS signal is activated(nCTS pin is low)	0

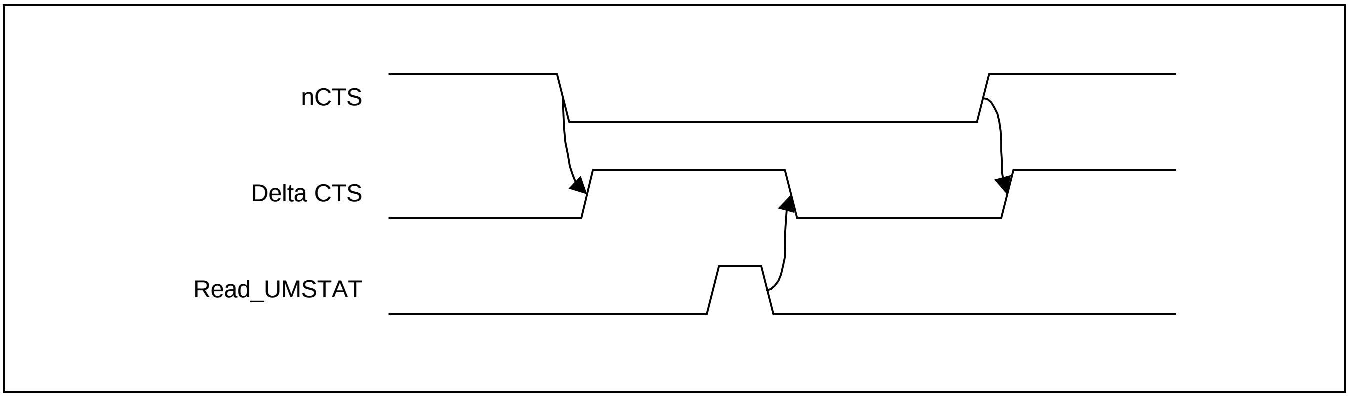


Figure 11-8. nCTS and Delta CTS Timing Diagram

UART TRANSMIT BUFFER REGISTER (HOLDING REGISTER & FIFO REGISTER) (UTXHn)

UTXHn has an 8-bit data for transmission data.

Register	Address	R/W	Description	Reset Value
UTXH0	0x1017 0020(L) 0x1017 0023(B)	W (by byte)	UART 0 transmit buffer register	Undefined
UTXH1	0x1018 0020(L) 0x1018 0023(B)	W (by byte)	UART 1 transmit buffer register	Undefined

UTXHn	Bit	Description	Initial State
TXDATAn	[7:0]	Transmit data for UARTn	Undefined

NOTES:

1. (L) Indicates Little-endian mode
2. (B) Indicates Big-endian mode

UART RECEIVE BUFFER REGISTER (HOLDING REGISTER & FIFO REGISTER) (URXHn)

URXHn has an 8-bit data for received data.

Register	Address	R/W	Description	Reset Value
URXH0	0x1017 0024(L) 0x1017 0027(B)	R (by byte)	UART 0 receive buffer register	Undefined
URXH1	0x1018 0024(L) 0x1018 0027(B)	R (by byte)	UART 1 receive buffer register	Undefined

URXHn	Bit	Description	Initial State
RXDATAn	[7:0]	Receive data for UARTn	Undefined

NOTE: When an overrun error occurs, the URXHn must be read. If not, the next received data will also make an overrun error, even though the overrun bit of USTATn had been cleared.

1. (L) Indicates Little-endian mode
2. (B) Indicates Big-endian mode

UART BAUD RATE DIVISION REGISTER (UBRDIVn)

The value stored in the baud-rate divisor register, UBRDIV, is used to determine the serial Tx/Rx clock rate (baud rate) as follows:

UBRDIVn = (round_off)(PCLK / (bps x 16)) -1

where the divisor should be from 1 to (2^{16}-1).
For example1, if the baud-rate is 230.4Kbps and PCLK is 25MHz , UBRDIVn is:

UBRDIVn = (int)(25000000 / (230400 x 16)+0.5) -1
= (int)(6.8+0.5) -1
= 7 -1 = 6 (=0x6)

For example2, if the baud-rate is 115.2Kbps and PCLK is 25MHz , UBRDIVn is:

UBRDIVn = (int)(25000000 / (115200 x 16)+0.5) -1
= (int)(13.6+0.5) -1
= 14 -1 = 13 (=0xD)

Register	Address	R/W	Description	Reset Value
UBRDIV0	0x1017 0028	R/W	UART 0 baud-rate divisior register 0	0x0000 000D
UBRDIV1	0x1018 0028	R/W	UART 1 baud-rate divisior register 1	0x0000 000D

UBRDIVn	Bit	Description	Initial State
UBRDIV	[15:0]	Baud rate division value UBRDIVn >0	0x000D

12 INTERRUPT CONTROLLER

OVERVIEW

The interrupt controller in S3C2800 manages the interrupt request from 28 sources. These interrupt sources can be organized in two groups: the internal peripheral (such as the DMA, UART, PCI, IIC, etc.) and the external interrupt request pins (such as EXTINT[7:0], IRIN, etc.). Note that the UART error and PCI interrupts are 'OR'ed together to from and interrupt sources.

The role of the interrupt controller is to ask for the FIQ or IRQ interrupt request to the ARM920T core after the arbitration process when there are multiple interrupt requests from internal peripherals and external interrupt request pins.

The arbitration process is performed by Interrupt mode (IRQ or FIQ) and interrupt mask logic and the result is written to the interrupt pending register (FIQ/IRQ) that can be read by the users in the interrupt service routine..

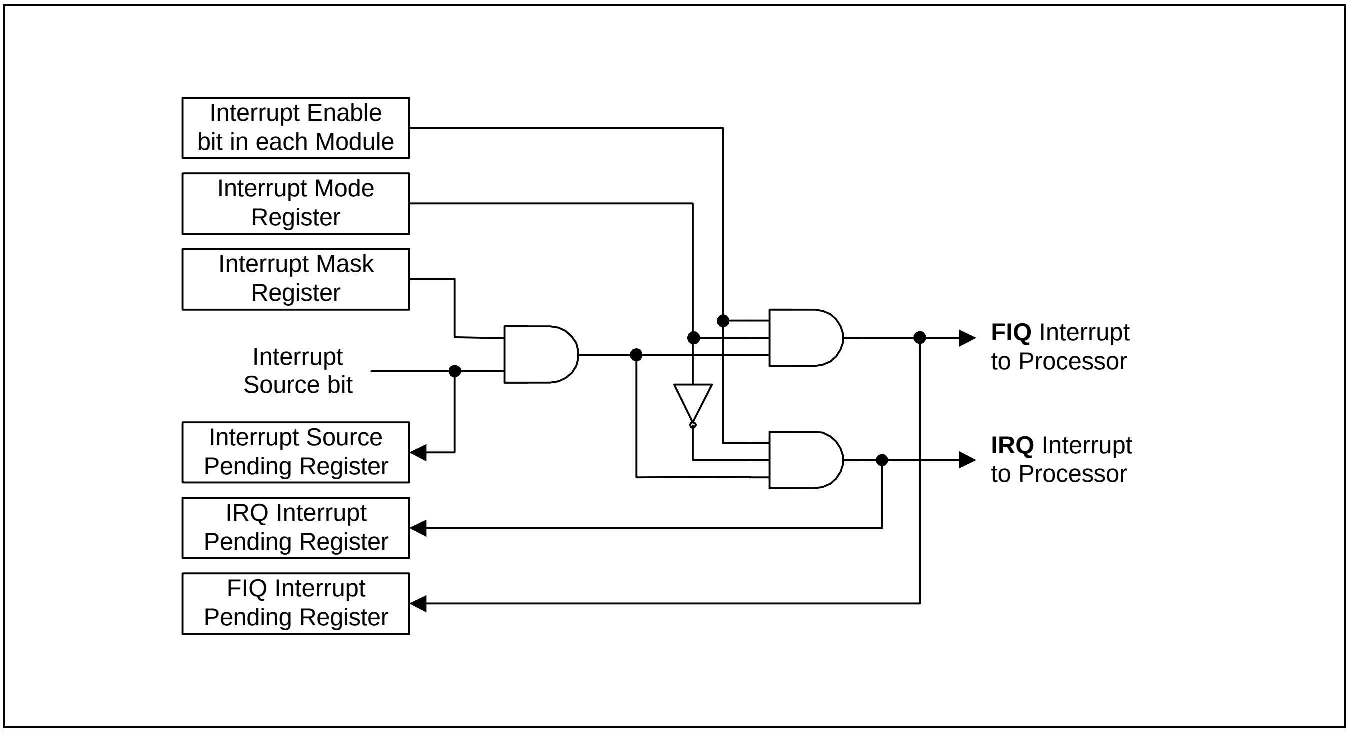

Interrupt Enable bit in each Module
Interrupt Mode Register
Interrupt Mask Register
Interrupt Source bit
Interrupt Source Pending Register
IRQ Interrupt Pending Register
FIQ Interrupt Pending Register
FIQ Interrupt to Processor
IRQ Interrupt to Processor

Figure 12-1. Interrupt Controller Block Diagram

INTERRUPT CONTROLLER OPERATION

F-BIT AND I-BIT OF PSR (PROGRAM STATUS REGISTER)

If the F-bit of PSR (program status register in ARM920T CPU) is set to 1, the CPU does not accept the FIQ (fast interrupt request) from the interrupt controller. If I-bit of PSR (program status register in ARM920T CPU) is set to 1, the CPU does not accept the IRQ (interrupt request) from the interrupt controller. So, to enable the interrupt reception, the F-bit or I-bit of PSR has to be cleared to 0 and also the corresponding bit of INTMSK has to be set to 1.

INTERRUPT MODE

ARM920T has 2 types of interrupt mode, FIQ and IRQ. All the interrupt sources determine the mode of interrupt to be used at interrupt request.

INTERRUPT SOURCES

Interrupt controller supports 28 interrupt sources as shown in table 12-1. User can find out the interrupt source in the interrupt service routine by reading the IRQPND & FIQPND register.

Table 12-1. Interrupt Source & Corresponding Bit

Corresponding bit	Sources	Descriptions
[28]	INT_RTC	RTC alarm interrupt
[27]	INT_TICK	RTC Time tick interrupt
[26]	INT_FULL	Remocon data FIFO full interrupt
[25]	INT_RMT	Remote control signal input interrupt
[24]	INT_UERR1	UART 1 error Interrupt
[23]	INT_UERR0	UART 0 error Interrupt
[22]	INT_TxD1	UART1 transmit interrupt
[21]	INT_TxD0	UART0 transmit interrupt
[20]	INT_ RxD1	UART 1 receive interrupt
[19]	INT_RxD0	UART 0 receive interrupt
[18]	INT_IIC1	IIC 1 interrupt
[17]	INT_ IIC0	IIC 0 interrupt
[16]	INT_TIMER2	Timer 2 interrupt
[15]	INT_TIMER1	Timer 1 interrupt
[14]	INT_TIMER0	Timer 0 interrupt
[13]	INT_DMA3	General DMA 3 interrupt
[12]	INT_DMA2	General DMA 2 interrupt
[11]	INT_DMA1	General DMA 1 interrupt
[10]	INT_DMA0	General DMA 0 interrupt
[9]	Reserved	Reserved
[8]	INT_PCI	PCI interrupt (7 PCI interrupt source in PCI block)
[7]	EXTINT7	External interrupt 7
[6]	EXTINT6	External interrupt 6
[5]	EXTINT5	External interrupt 5
[4]	EXTINT4	External interrupt 4
[3]	EXTINT3	External interrupt 3
[2]	EXTINT2	External interrupt 2
[1]	EXTINT1	External interrupt 1
[0]	EXTINT0	External interrupt 0

INTERRUPT CONTROLLER SEPCIAL FUNCTION REGISTERS

There are five control registers in the interrupt controller: source pending register **(SRCPND)**, interrupt mode register **(INTMOD)**, mask register **(INTMSK)**, and interrupt pending registers (**IRQPND**,**FIQPND**).

All the interrupt requests from the interrupt sources are first registered in the source pending register. They are divided into two groups based on the interrupt mode register, i.e., one FIQ request and the remaining IRQ requests. Masked interrupt source will not be set in the interrupt pending registers (IRQPND,FIQPND). The details of each control registers are as follows.

INTERRUPT SOURCE PENDING REGISTER (SRCPND)

SRCPND register is composed of 28 bits, each of which is related to an interrupt source (Refer to Table 12-1). Each bit is set to 1 if the corresponding interrupt source generates the interrupt request and waits for the interrupt to be serviced. Note that each bit of SRCPND register is automatically set by the interrupt sources regardless of the masking bits in the INTMSK register.

In the interrupt service routine for a specific interrupt source, the corresponding bit of SRCPND register has to be cleared to get the interrupt request from the same source correctly. If you return from the ISR without clearing the bit, interrupt controller operates as if another interrupt request comes in from the same source. In other words, if a specific bit of SRCPND register is set to 1, it is always considered as a valid interrupt request waiting to be serviced.

You can clear the specific bit of SRCPND register as follows: In the interrupt service routine for IRQ or FIQ, write 1 to SRCPND register. And then it automatically clears the interrupt pending registers (IRQPND,FIQPND).

Register	Address	R/W	Description	Reset Value
SRCPND	0x1002 0000	R/W	Indicates the interrupt request status.	0x0000 0000

SRCPND	Bit	Description	Initial State
EIN_xx (Refer to Table 12-1)	[28:0]	Indicates the interrupt request status 0 = Not requested, 1 = Requested Bit [9] = Reserved	0x0000 0000

INTERRUPT MODE REGISTER (INTMOD)

This register is composed of 28 bits, each of which is related to an interrupt source (Refer to Table 12-1). If a specific bit is set to 1, the corresponding interrupt is processed in the FIQ (fast interrupt) mode. Otherwise, it is processed in the IRQ mode (normal interrupt).

Note that at most only one interrupt source can be serviced in the FIQ mode in the interrupt controller. (You should use the FIQ mode only for the urgent interrupt.) Thus, only one bit of INTMOD can be set to 1 at most.

Register	Address	R/W	Description	Reset Value
INTMOD	0x1002 0004	R/W	Interrupt mode Register	0x0000 0000

INTMOD	Bit	Description	Initial State
INT_xx (Refer to Table 12-1)	[28:0]	Interrupt mode Register 0 = IRQ mode 1 = FIQ mode Bit [9] = Reserved	0x0000 0000

INTERRUPT MASK REGISTER (INTMSK)

Each of the 28 bits in the interrupt mask register is related to an interrupt source (Refer to Table 12-1). If you clear a specific bit to 0, the interrupt request from the corresponding interrupt source is not serviced by the CPU. (Note that even in such a case, the corresponding bit of SRCPND register is set to 1). If the mask bit is 0, the interrupt request can be serviced.

Register	Address	R/W	Description	Reset Value
INTMSK	0x1002 0008	R/W	Determines which interrupt source is masked. The masked interrupt source will not be serviced.	0x0000 0000

INTMSK	Bit	Description	Initial State
INT_xx (Refer to Table 12-1)	[28:0]	Determines which interrupt source is masked. The masked interrupt source will not be serviced. 0 = Masked 1 = Service available Bit [9] = Reserved	0x0000 0000

IRQ/FIQ INTERRUPT PENDING REGISTER (IRQPND/FIQPND)

The IRQPND and the FIQPND contain one flag per interrupt (28 total) that indicates an interrupt request has been made by a unit. Inside the interrupt service routine, the IRQPND and FIQPND are read to determine the interrupt source.

Bits within the IRQPND and FIQPND are read only. Once an interrupt has been serviced, the handler clears the pending bit in the interrupt service routine by writing 1 to the necessary bit in the source pending register (SRCPND). Clearing the interrupt source pending bit in the interrupt service routine automatically clears the corresponding bit in the IRQPND and FIQPND register.

This is a read-only register.

Register	Address	R/W	Description	Reset Value
IRQPND	0x1002 000C	R	IRQ interrupt service pending register	0x0000 0000
FIQPND	0x1002 0010	R	FIQ interrupt service pending register	0x0000 0000

IRQPND/FIQPND	Bit	Description	Initial State
INT_xx (Refer to Table 12-1)	[28:0]	IRQ/FIQ interrupt service pending register 0 = not requested 1 = requested now Bit [9] = Reserved	0x0000 0000

IMPORTANT NOTE

To clear the IRQPND/FIQPND, the following two rules has to be obeyed.

1. The pending bit in source pending register (SRCPND) should have to clear by writing 1.
2. And then it is cleared the pending bit in interrupt pending register (IRQPND/FIQPND) automatically.

13 REMOTE CONTROL SIGNAL RECEIVER

OVERVIEW

The S3C2800 is capable of capturing and storing up to 8 remote control signals in 8-bit resolution. The remote control signal receiver can detect the rising edge, falling edge or rising/falling edge of the remote control signal and captures the width of the pulse edges. The minimum pulse width of the remote control signal should be greater than (1/32768)*5 sec; otherwise, the remote control signal receiver may not capture the pulse width. Figure 3-1 shows the components in this receiver block.

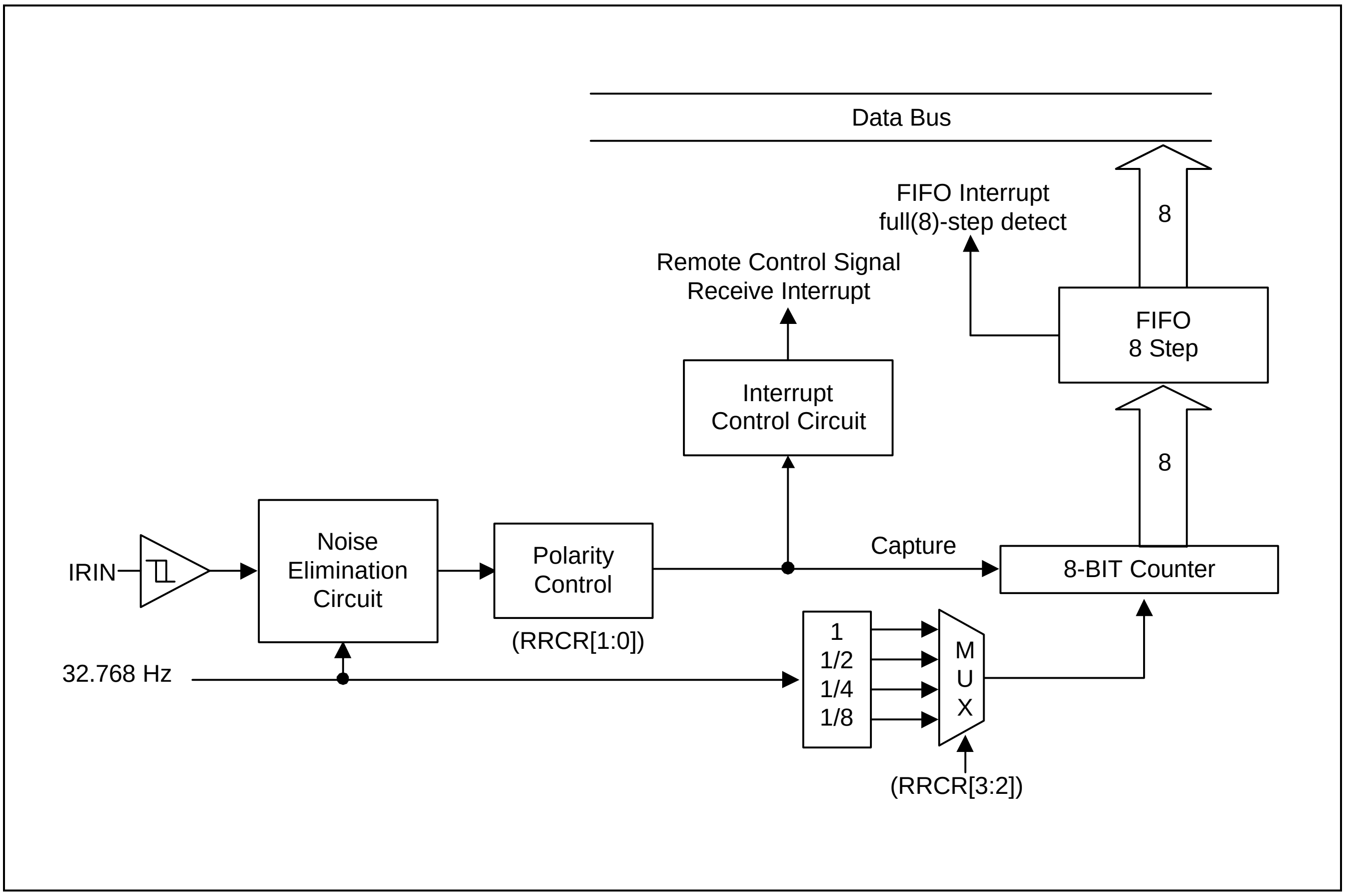


Figure 13-1. Remote Control Signal Receiver Block Diagram

REMOTE CONTROL SIGNAL RECEIVER BLOCK OPERATION

When the remote control signal is input to the IRIN pin, the current 8-bit counter value is written to FIFO at the signal’s edge (Falling or Rising) and the counter is cleared to 0 and then the counter starts to count again until the next edge of the signal is detected. Refer to Figure 13-2 for timing information of the counter value capture process.

The S3C2800 remote control signal receiver block has eight 8-bit FIFO's–the FIFO full status flag (RRCR[5]) will be set when all the FIFO's become full. The FIFO's will not be overwritten after the FIFO's have been filled even if a remote control signal is detected. Thus, in order to prevent the loss of incoming remote control signal, users are recommended to read the FIFO data out appropriate to ensure that the FIFO's does not continue to be full.

NOTE: The very first capture counter data in FIFO is meaningless because the first data is written to the FIFO at the first edge of the incoming remote control signal. Therefore, care must be taken when programming the remote receiver block.

NOISE FILTER

The S3C2800 remote control signal receiver block has a built-in 5-step noise filter, which operates 32768Hz, for prevention of errors from noise. Therefore, the minimum pulse width of the remote control signal must be greater than that of the filtering pulse of the noise filter. That is, a pulse width smaller than this is considered as noise and is ignored. The filtering pulse width of the noise filter is ((1/32768)*5)sec(=152.6us).

8-BIT COUTER SAMPLING CLOCK

The 8-bit counter sampling clock can be selected by bit [3:2] in the RRCR register. This clock must be set such that the error is minimum and overflow of the counter does not occur in consideration of the pulse width of the remote signal. If the pulse width of the remote control signal is small but the sampling time is set too large, the error will increases; on the other hand, if the pulse width is large but the sampling time is set too small, an overflow occurs.

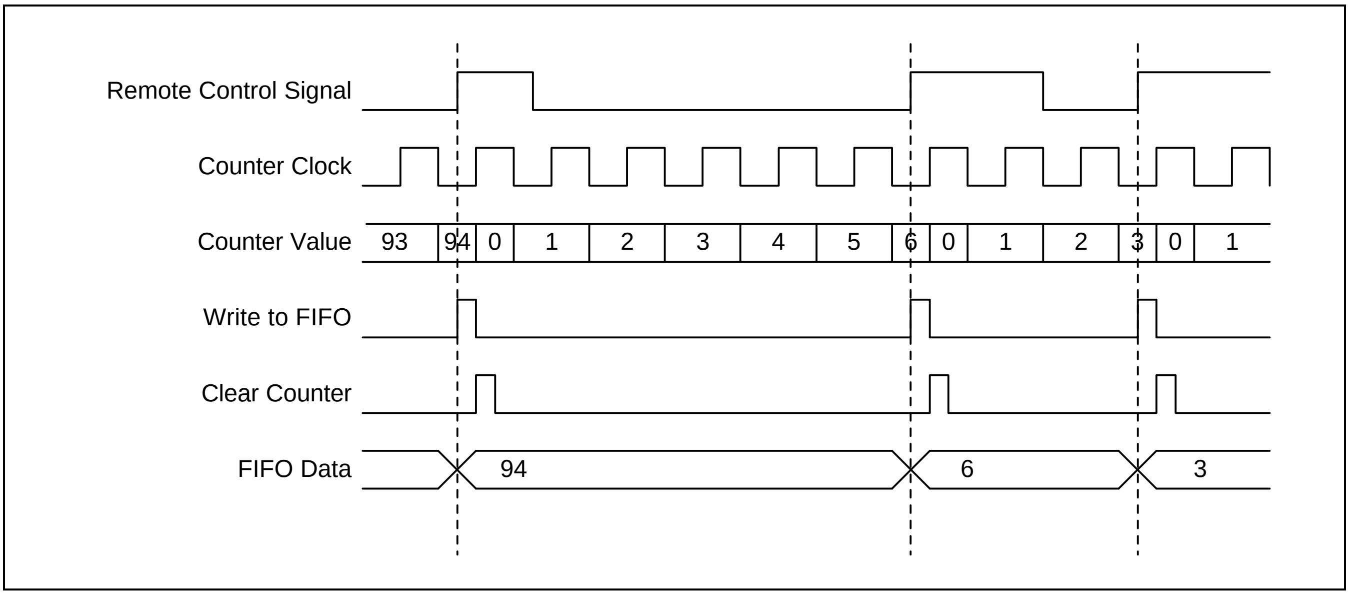


Figure 13-2. Remote Control Signal Receiver Operation Timing

REMOTE CONTROL SIGNAL RECEIVER SPECIAL FUNCTION REGISTERS

REMOTE CONTROL SIGNAL RECEIVER CONTROL REGISTER (RRCR)

Register	Address	R/W	Description	Reset Value
RRCR	0x1011 0000	R/W	Remote signal receiver control register	0x0000 0010

RRCR	Bit	Description	Initial State
Counter overflow status flag	[8]	Counter overflow status flag 0 = Not overflow 1 = Overflow	0
FIFO full (8)-step detect interrupt	[7]	Enable FIFO full (8)-step detect interrupt 0 = Disable the FIFO full interrupt 1 = Enable the FIFO full interrupt	0
Remote control signal receive Interrupt	[6]	Enable remote control signal receive interrupt 0 = Disable the interrupt 1 = Enable the interrupt	0
FIFO full status flag	[5]	FIFO full (8) status flag (Read only) 0 = Not full 1 = Full	0
FIFO empty status flag	[4]	FIFO empty status flag (Read only) 0 = Not empty 1 = Empty	1
Counter clock selection	[3:2]	8-bit counter clock selection 00 = 32,768 Hz/1 01 = 32,768 Hz/2 10 = 32,768 Hz/4 11 = 32,768 Hz/8	00
Polarity control flag	[1:0]	Polarity control flag for remocon input interrupt 0x = Rising edge mode 10: Falling edge mode 11: Rising & Falling edge mode	00

NOTE: If all FIFOs are full, the next input data does not go into FIFO.

FIFO DATA REGISTER (FIFOD)

Register	Address	R/W	Description	Reset Value
FIFOD	0x1011 0004	R	FIFO Data register	Undefined

RRCR	Bit	Description	Initial State
FIFO Data	[7:0]	FIFO Data (8-bit)	Undifined

NOTES

14 RTC (REAL TIME CLOCK)

OVERVIEW

The RTC (Real Time Clock) unit provides the clock data to CPU in BCD (Binary Coded Decimal) format. The clock data include second, minute, hour, date, day, month, and year. The RTC unit also supports an alarm function and works with an external 32.768 kHz crystal.

FEATURE

- BCD number: second, minute, hour, date, day, month, year
- Leap year generator
- Alarm interrupt
- Year 2000 problem is removed.
- Supports millisecond tick time interrupt for RTOS kernel time tick.
- Round reset function

REAL TIME CLOCK OPERATION

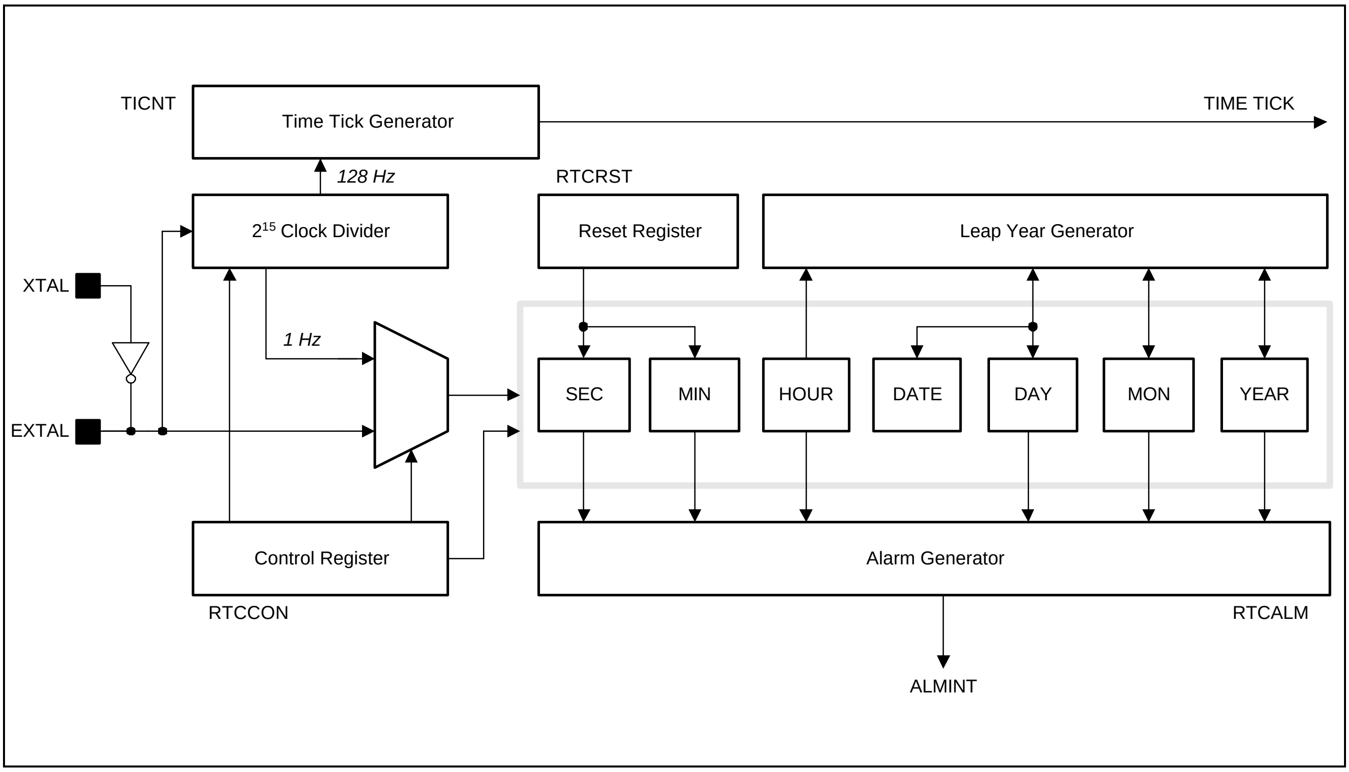


Figure 14-1. Real Time Clock Block Diagram

LEAP YEAR GENERATOR

Based on data from BCDDAY, BCDMON, and BCDYEAR this block determines whether the last date of each month is 28, 29, 30, or 31. This block considers the leap year to determine the last date of each month. Since an 8-bit counter can only represent 2 BCD digits, it cannot be determined whether 00 year is a leap year or not. For example, it cannot distinguish between 1900 and 2000. Please note 1900 is not leap year while 2000 is leap year. To solve this problem, the RTC block in S3C2800 has a hard-wired logic to support the leap year in 2000. Therefore, two digits of 00 in S3C2800 represents the year 2000, not 1900.

READ/WRITE REGISTERS

Bit 0 of the RTCCON register must be set to high in order to write the BCD register in RTC block. When the CPU loads the data in BCDSEC, BCDMIN, BCDHOUR, BCDDAY, BCDDATE, BCDMON, and BCDYEAR of the RTC block, a one second deviation may exist because multiple registers are being loaded. For example, assuming the current RTC data of 2059 (Year), 12 (Month), 31(Date), 23 (Hour) and 59 (Minute), there is no problem if CPU loads the BCDSEC register and the value ranges from 1 to 59 (Second). However, if the loaded value is 0 sec., the year, month, date, hour, and minute may have been changed to 2060 (Year), 1 (Month), 1 (Date), 0 (Hour) and 0 (Minute) because of the one second deviation. In this case, values of BCDYEAR to BCDSEC must be re-loaded for correct value.

ALARM FUNCTION

The RTC can generates an alarm signal at a specified time when the alarm interrupt (ALMINT) is activated. The RTC alarm register, RTCALM, determines the alarm enable/disable and the condition of the alarm time setting.

TICK TIME INTERRUPT

The RTC tick time is used for interrupt request. The TICNT register has an interrupt enable bit and the count value for the interrupt. When the count value reaches '0', the tick time interrupt occurs at the following period.

Period = (n+1) / 128 second
n: Tick time count value (1–127)

This RTC time tick may be used for RTOS(real time operating system) kernel time tick. If time tick is generated by RTC time tick, the time related function of RTOS will always synchronized with real time.

ROUND RESET FUNCTION

The round boundary (30, 40, or 50 sec) of the second carry generation can be selected through RTC round reset register, RTCRST[1:0]. The second value is rounded to zero when the round reset bit (RTCRST[2]) is set. For example, when the current time is 23:37:47 and the round boundary is selected to be 40 sec, the round reset changes the current time to 23:38:00.

32.768KHZ X-TAL CONNECTION EXAMPLE

The Figure 17-2 is an example circuit of the RTC unit oscillation at 32.768 kHz.

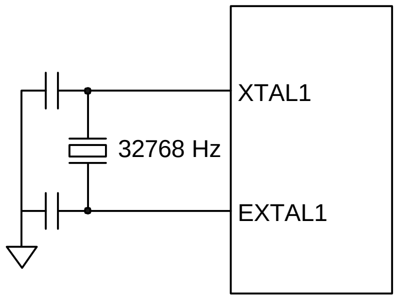


Figure 14-2. Main Oscillator Circuit Examples

REAL TIME CLOCK SPECIAL FUNCTION REGISTERS

REAL TIME CLOCK CONTROL REGISTER (RTCCON)

The RTCCON register is composed of RTCEN, which controls the read/write enable of the BCD registers, CLKSEL, CNTSEL, and CLKRST.

RTCEN allows the control of read/write operation between the CPU and the RTC, so it should be set to 1 in an RTC control routine to enable data read/write after a system reset. Also before power off, the RTCEN bit should be cleared to 0 to prevent inadvertent writing into RTC registers.

Register	Address	R/W	Description	Reset Value
RTCCON	0x1016 0000	R/W	RTC control register	0x0000 0000

RTCCON	Bit	Description	Initial State
CLKRST	[3]	RTC clock count stop 0 = Run 1 = Stop	0
CNTSEL	[2]	BCD count select 0 = Merge BCD counters 1 = Reserved (Separate BCD counters)	0
CLKSEL	[1]	BCD clock select 0 = XTAL 1/2^{15} divided clock 1 = Reserved (XTAL clock)	0
RTCEN	[0]	RTC control enable 0 = Disable 1 = Enable **NOTE:** Only BCD time count and read operation can be performed.	0

RTC ALARM CONTROL REGISTER (RTCALM)

RTCALM register determines the alarm enable and the alarm time. Note that the RTCALM register generates the alarm signal through ALMINT.

Register	Address	R/W	Description	Reset Value
RTCALM	0x1016 0004	R/W	RTC alarm control register	0x0000 0000

RTCALM	Bit	Description	Initial State
ALMEN	[6]	Alarm global enable 0 = Disable, 1 = Enable	0
YEAREN	[5]	Year alarm enable 0 = Disable, 1 = Enable	0
MONREN	[4]	Month alarm enable 0 = Disable, 1 = Enable	0
DAYEN	[3]	Day alarm enable 0 = Disable, 1 = Enable	0
HOUREN	[2]	Hour alarm enable 0 = Disable, 1 = Enable	0
MINEN	[1]	Minute alarm enable 0 = Disable, 1 = Enable	0
SECEN	[0]	Second alarm enable 0 = Disable, 1 = Enable	0

ALARM SECOND DATA REGISTER (ALMSEC)

Register	Address	R/W	Description	Reset Value
ALMSEC	0x1016 0008	R/W	Alarm second data register	0x0000 0000

ALMSEC	Bit	Description	Initial State
SECDATA	[6:4]	BCD value for alarm second from 0 to 5	000
	[3:0]	from 0 to 9	0000

ALARM MIN DATA REGISTER (ALMMIN)

Register	Address	R/W	Description	Reset Value
ALMMIN	0x1016 000C	R/W	Alarm minute data register	0x0000 0000

ALMMIN	Bit	Description	Initial State
MINDATA	[6:4]	BCD value for alarm minute from 0 to 5	000
	[3:0]	from 0 to 9	0000

ALARM HOUR DATA REGISTER (ALMHOUR)

Register	Address	R/W	Description	Reset Value
ALMHOUR	0x1016 0010	R/W	Alarm hour data register	0x0000 0000

ALMHOUR	Bit	Description	Initial State
HOURDATA	[5:4]	BCD value for alarm hour from 0 to 2	00
	[3:0]	from 0 to 9	0000

ALARM DAY DATA REGISTER (ALMDAY)

Register	Address	R/W	Description	Reset Value
ALMDAY	0x1016 0014	R/W	Alarm day data register	0x0000 0001

ALMDAY	Bit	Description	Initial State
DAYDATA	[5:4]	BCD value for alarm day, from 0 to 28, 29, 30, 31 from 0 to 3	00
	[3:0]	from 0 to 9	0001

ALARM MON DATA REGISTER (ALMMON)

Register	Address	R/W	Description	Reset Value
ALMMON	0x1016 0018	R/W	Alarm month data register	0x0000 0001

ALMMON	Bit	Description	Initial State
MONDATA	[4]	BCD value for alarm month from 0 to 1	0
	[3:0]	from 0 to 9	0001

ALARM YEAR DATA REGISTER (ALMYEAR)

Register	Address	R/W	Description	Reset Value
ALMYEAR	0x1016 001C	R/W	Alarm hour data register	0x0000 0000

ALMYEAR	Bit	Description	Initial State
YEARDATA	[7:0]	BCD value for year from 00 to 99	0x00

BCD SECOND REGISTER (BCDSEC)

Register	Address	R/W	Description	Reset Value
BCDSEC	0x1016 0020	R/W	BCD second register	Undefined

BCDSEC	Bit	Description	Initial State
SECDATA	[6:4]	BCD value for second from 0 to 5	Undefined
	[3:0]	from 0 to 9	Undefined

BCD MIN REGISTER (BCDMIN)

Register	Address	R/W	Description	Reset Value
BCDMIN	0x1016 0024	R/W	BCD minute register	Undefined

BCDMIN	Bit	Description	Initial State
MINDATA	[6:4]	BCD value for minute from 0 to 5	Undefined
	[3:0]	from 0 to 9	Undefined

BCD HOUR REGISTER (BCDHOUR)

Register	Address	R/W	Description	Reset Value
BCDHOUR	0x1016 0028	R/W	BCD hour register	Undefined

BCDHOUR	Bit	Description	Initial State
HOURDATA	[5:4]	BCD value for hour from 0 to 2	Undefined
	[3:0]	from 0 to 9	Undefined

BCD DAY REGISTER (BCDDAY)

Register	Address	R/W	Description	Reset Value
BCDDAY	0x1016 002C	R/W	BCD day register	Undefined

BCDDAY	Bit	Description	Initial State
DAYDATA	[5:4]	BCD value for day from 0 to 3	Undefined
	[3:0]	from 0 to 9	Undefined

BCD DATE REGISTER (BCDDATE)

Register	Address	R/W	Description	Reset Value
BCDDATE	0x1016 0030	R/W	BCD date register	Undefined

BCDDATE	Bit	Description	Initial State
DATEDATA	[2:0]	BCD value for date from 1 to 7	Undefined

BCD MON REGISTER (BCDMON)

Register	Address	R/W	Description	Reset Value
BCDMON	0x1016 0034	R/W	BCD month register	Undefined

BCDMON	Bit	Description	Initial State
MONDATA	[4]	BCD value for month from 0 to 1	Undefined
	[3:0]	from 0 to 9	Undefined

BCD YEAR REGISTER (BCDYEAR)

Register	Address	R/W	Description	Reset Value
BCDYEAR	0x1016 0038	R/W	BCD year register	Undefined

BCDYEAR	Bit	Description	Initial State
YEARDATA	[7:0]	BCD value for year from 00 to 99	Undefined

TICK TIME COUNT REGISTER (TICNT)

Register	Address	R/W	Description	Reset Value
TICNT	0x1016 0040	R/W	Tick time count register	0x0000 0000

TICNT	Bit	Description	Initial State
Tick INT Enable	[7]	Tick time interrupt enable 0 = disable 1 = enable	0
Tick Time Count	[6:0]	Tick time count value. (1–127)	000000

RTC ROUND RESET REGISTER (RTCRST)

This register allows control of the round reset function. The result of round reset function depends on the time that the round second reset enable bit is set. If the round reset is enabled before the round boundary selected by SECCR, only second reset occurs; if enabled past the round boundary, both second carry and reset occur.

For example, assume that the round boundary is selected to be 40 sec, (1) if the round reset is enabled at 23:37:35, the time changes to 23:37:00. (2) if enabled at 23:37:45, it changes to 23:38:00.

Register	Address	R/W	Description	Reset Value
RTCRST	0x1016 0044	R/W	RTC round reset Register	0x0000 0000

RTCRST	Bit	Description	Initial State
SRSTEN	[2]	Round second reset enable 0 = Disable, 1 = Enable When this bit is set , it automatically will be cleared.	0-
SECCR	[1:0]	Round boundary for second carry generation. 00 = over than 30 sec 01 = over than 40 sec 1x = over than 50 sec	00

NOTES

15 WATCHDOG TIMER

OVERVIEW

The S3C2800 watchdog timer is used to resume the controller operation when it had been disturbed by malfunctions such as noise or system errors. The watchdog timer generates the reset signal for 128 PCLK cycles.

CONSIDERATION OF DEBUGGING ENVIRONMENT

When S3C2800 is in debug mode using Embedded ICE, the watchdog timer must not operate. The watchdog timer can determine whether or not the current mode is the debug mode from the CPU core signal (DBGACK signal). Once the DBGACK signal is asserted, the reset output of the watchdog timer isn't activated when the watchdog timer expires.

WATCHDOG TIMER OPERATION

The functional block diagram of the watchdog timer is shown in Figure 15-1. The watchdog timer uses internal APB bus clock (PCLK) as its only source clock. To generate the corresponding watchdog timer clock, the PCLK frequency is prescaled first, and the resulting frequency is divided again.

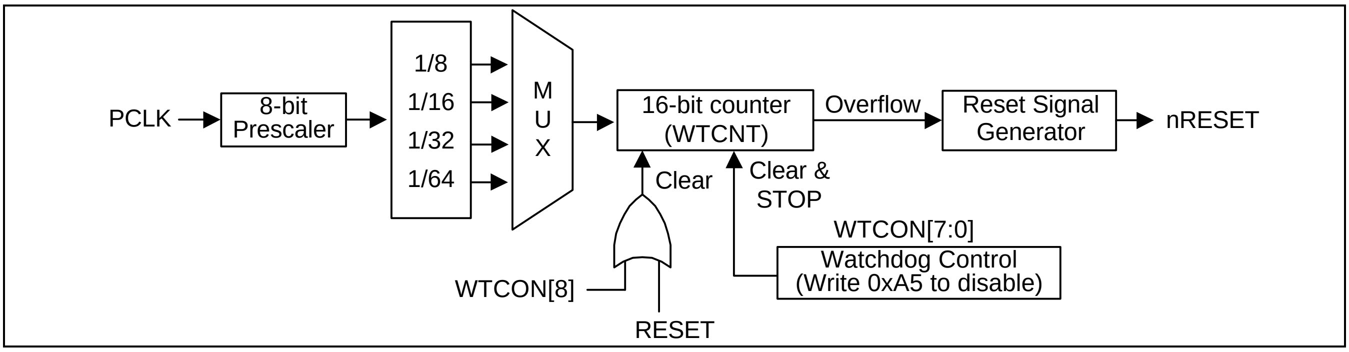


Figure 15-1. Watchdog Timer Block Diagram

The prescaler value and the frequency division factor are specified in the watchdog timer control register, WTCON. The valid prescaler values range from 1 to 2^8-1. The frequency division factor can be selected among 8, 16, 32, or 64. Use the following equation to calculate the watchdog timer reset interval time.

$$t\_watchdog = (1/(PCLK / (Prescaler\ value + 1)/Division\ factor)) * 2^{16}\ (16\text{-bit counter})$$

Table 15-1. An Example of Watchdog Interval Time

Divider settings	Minimum resolution (prescaler = 1)	Maximum resolution (prescaler = 255)	maximum interval (WTCNT = 65535)
1/8 (PCLK = 50 MHz)	0.32 us (3.125 MHz)	40.96 us (24.42 kHz)	2.684 sec
1/16 (PCLK = 50 MHz)	0.64 us (1.563 MHz)	81.92 us (12.21 kHz)	5.368 sec
1/32 (PCLK = 50 MHz)	1.28 us (0.782 MHz)	163.84 us (6.11 kHz)	10.736 sec
1/64 (PCLK = 50 MHz)	2.56 us (0.391 MHz)	327.68 us (3.06 kHz)	21.472 sec
1/8 (PCLK = 37.5 MHz)	0.42 us (2.344 MHz)	54.61 us (18.31 kHz)	3.579 sec
1/16 (PCLK = 37.5 MHz)	0.84 us (1.172 MHz)	109.22 us (9.16 kHz)	7.158 sec
1/32 (PCLK = 37.5 MHz)	1.71 us (0.586 MHz)	218.44 us (4.58 kHz)	14.316 sec
1/64 (PCLK = 37.5 MHz)	3.41 us (0.293 MHz)	436.9 us (2.28 kHz)	28.63 sec

WATCHDOG TIMER SPECIAL FUNCTION REGISTERS

WATCHDOG TIMER PRESCALER VALUE REGISTER (WTPSCLR)

The valid prescaler values range from 1 to 2^8-1

Register	Address	R/W	Description	Reset Value
WTPSCLR	0x1012 0000	R/W	Watchdog timer prescaler value Register	0x0000 0080

WTPSCLR	Bit	Description	Initial State
Pre-Scaler	[7:0]	8-bit pre-scaler value (1 – 255) 0 = Not supported	0x80

NOTE: The Watchdog timer must be “Disabled” before changing the prescaler value. After the prescaler value is set, the Watchdog timer can be “Enabled”.

WATCHDOG TIMER CONTROL REGISTER (WTCON)

Using the Watchdog Timer Control register, WTCON, you can enable/disable the watchdog timer, clear the watchdog timer counter , select the clock signal from 4 different sources.
The watchdog timer is used to resume the S3C2800 restart on mal-function after power-on; if controller restart is not desired, the watchdog timer should be disabled.

Register	Address	R/W	Description	Reset Value
WTCON	0x1012 0004	R/W	Watchdog timer control Register	0x0000 0000

WTCON	Bit	Description	Initial State
Clock select	[11:10]	This two bits determines the clock division factor. 00 = 1/8 01 = 1/16 10 = 1/32 11 = 1/64	00
Reserved	[9]	Reserved	0
Watchdog timer counter clear	[8]	Clear to watchdog timer count value 0 = No effect 1 = Clear to count value When this bit is set, it clears the watchdog timer counter (WTCNT) and clears itself too.	0
Watchdog timer Enable	[7:0]	This bit enables or disables the Watch-dog timer output for reset signal. 1010 0101b = Disable the reset function of the watchdog timer. The 16-bit counter is clear to 0x0, and then it is stop. Other Value = Assert reset signal of the S3C2800 at watchdog time out. The 16-bit counter starts counting from 0x0 again after re-load the prescaler value.	0x00

WATCHDOG TIMER COUNTER REGISTER (WTCNT)

The watchdog timer counter register, WTCNT, contains the current count values of the watchdog timer during normal operation.

Register	Address	R/W	Description	Reset Value
WTCNT	0x1012 0008	R	Watchdog timer counter Register	0x0000 0000

WTCNT	Bit	Description	Initial State
Count value	[15:0]	The current count value of the watchdog timer counter	0x0000

16 IIC-BUS INTERFACE

OVERVIEW

The S3C2800 RISC microprocessor supports 2-channel multi-master IIC-bus serial interface. A dedicated bi-directional serial data line (IICSDAn) and a serial clock line (IICSCLKn) carry information between bus masters and peripheral devices connected to the IIC-bus.

In multi-master IIC-bus mode, multiple S3C2800 RISC microprocessor can receive or transmit serial data to or from slave devices. The master S3C2800, which can initiate a data transfer over the IIC-bus, is responsible for termination of the transfer. Standard bus arbitration procedure is employed in S3C2800 IIC_bus.

To control multi-master IIC-bus operations, values must be written to the following registers:

- Multi-master IIC-bus control register, IICCONn
- Multi-master IIC-bus control/status register, IICSTATn
- Multi-master IIC-bus Tx/Rx data shift register, IICDSn
- Multi-master IIC-bus address register, IICADDn

When the IIC-bus is free, the SDA and SCL lines should be both at high level. A high-to-low transition of SDA can initiate a start condition. A low-to-high transition of SDA can initiate a stop condition while SCL remains steady at high level.

The start and stop conditions can always be generated by the master devices. A 7-bit address value in the first data byte, which is put onto the bus after the start condition has been initiated, determines the slave device which the bus master device has selected. The 8th bit determines the direction of the transfer (read or write).

Every data byte put onto the SDA line should total eight bits. The number of bytes that can be sent or received for each bus transfer is unlimited. Data is always sent from most-significant bit (MSB) first, and every byte should be immediately followed by an acknowledge (ACK) bit.

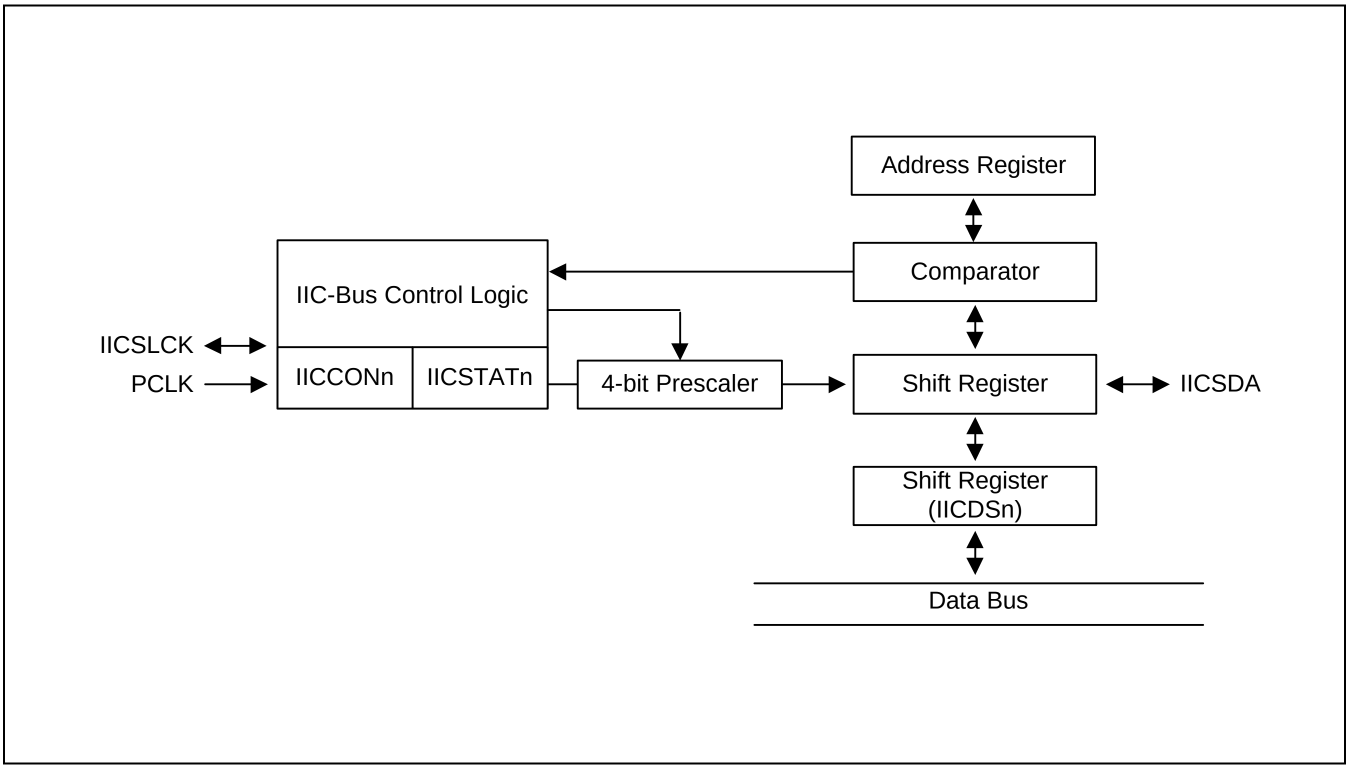


Figure 16-1. IIC-Bus Block Diagram

THE IIC-BUS OPERATION

THE IIC-BUS INTERFACE

The S3C2800 IIC-bus interface supports four operation modes:

- Master transmitter mode
- Master receive mode
- Slave transmitter mode
- Slave receive mode

Functional relationships among these operating modes are described below.

START AND STOP CONDITIONS

When the IIC-bus interface is in inactive, it is usually in slave mode. In other word, the interface should be in slave mode before detecting a start condition on the SDA line. (A Start condition is initiated with a high-to-low transition of the SDA line while the clock signal of SCL is high) When the interface state is changed to the master mode, a data transfer on the SDA line can be initiated and SCL (IIC clock) is generated.

A start condition initiates a one-byte serial data over the SDA line, and a stop condition terminate the data transfer. A stop condition is a low-to-high transition of the SDA line while SCL is High. start and stop conditions are always generated by the master. The IIC-bus is busy when a start condition is generated. A few clocks after the stop condition, the IIC-bus becomes free again.

When a master initiates a start condition, it should send a slave address to notify the slave device. The one byte of address field consist of a 7-bit address and a 1-bit transfer direction indicator (that is, write or read).
If bit 8 is 0, it indicates a write operation (transmit operation). If bit 8 is 1, it indicates a request for data read (receive operation).

The master will finish the transfer operation by transmitting a stop condition. If the master wants to continue the data transmission on the bus, it should generate another start condition as well as a slave address. In this way, the read and write operation can be performed in various format.

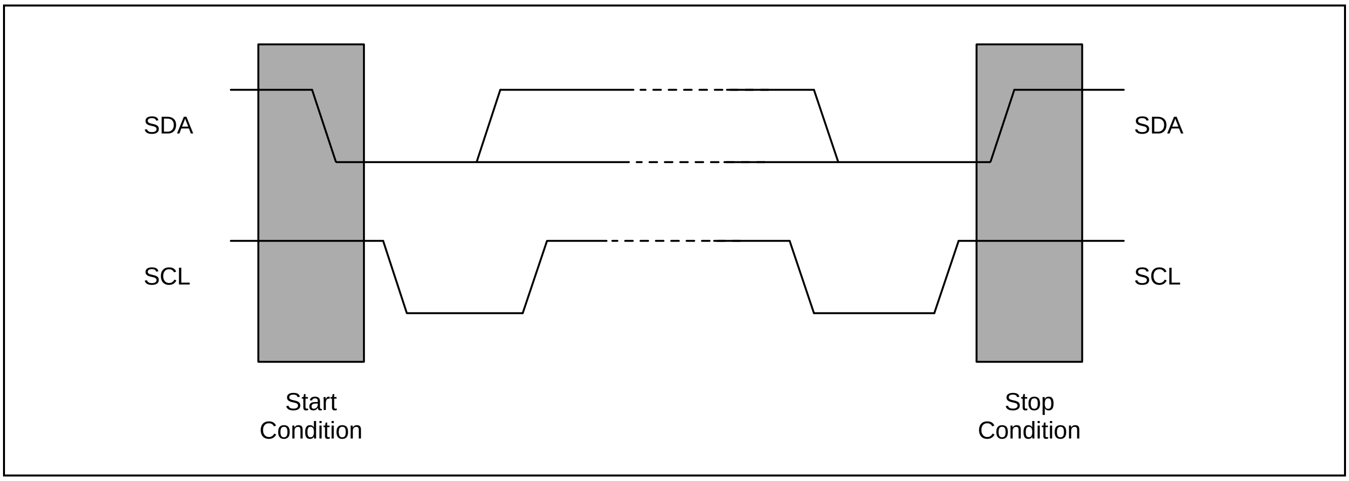
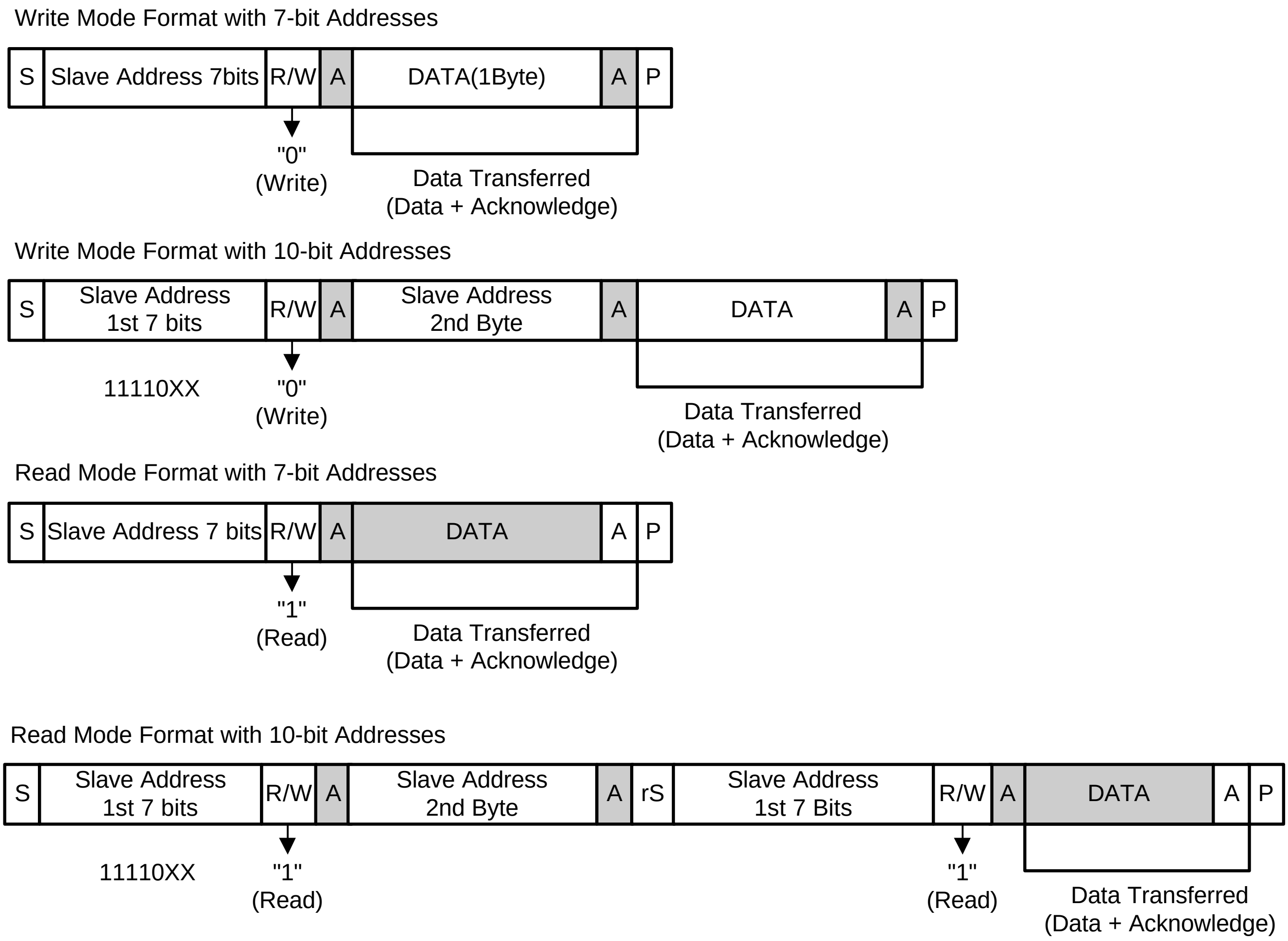


Figure 16-2. Start and Stop Condition

DATA TRANSFER FORMAT

Every byte placed on the SDA line should be eight bits in length. The number of bytes which can be transmitted per transfer is unlimited. The first byte following a start condition should have the address field. The address field can be transmitted by the master when the IIC-bus is operating in master mode. Each byte should be followed by an acknowledgement (ACK) bit. The MSB bit of the serial data and addresses are always sent first.



NOTES:

1. S: Start, rS: Repeat Start, P: Stop, A: Acknowledge
2. □ : From Master to Slave, ▨ : from Slave to Master

Figure 16-3. IIC-Bus Interface Data Format

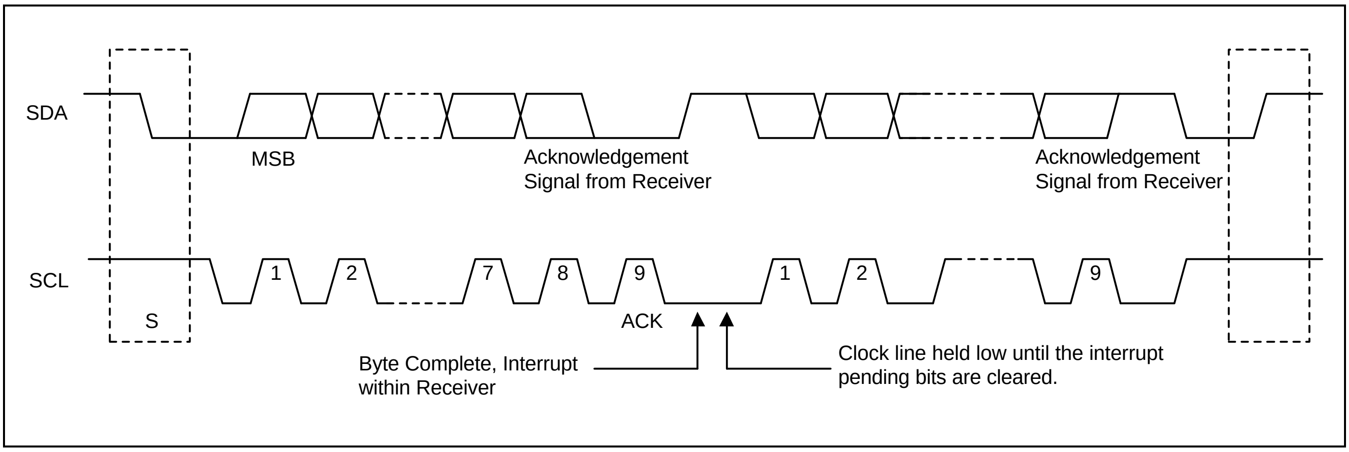

SDA
SCL
S
MSB
1
2
7
8
9
ACK
1
2
9
Acknowledgement Signal from Receiver
Acknowledgement Signal from Receiver
Byte Complete, Interrupt within Receiver
Clock line held low until the interrupt pending bits are cleared.

Figure 16-4. Data Transfer on the IIC-Bus

ACK SIGNAL TRANSMISSION

To finish a one-byte transfer operation completely, the receiver should send an ACK bit to the transmitter. The ACK pulse should occur at the ninth clock of the SCL line. Eight clocks are required for the one-byte data transfer. The master should generate the clock pulse required to transmit the ACK bit.

The transmitter should release the SDA line by making the SDA line High when the ACK clock pulse is received. The receiver should also drive the SDA line Low during the ACK clock pulse so that the SDA is Low during the High period of the ninth SCL pulse.

The ACK bit transmit function can be enabled or disabled by software (IICSTATn). However, the ACK pulse on the ninth clock of SCL is required to complete a one-byte data transfer operation.

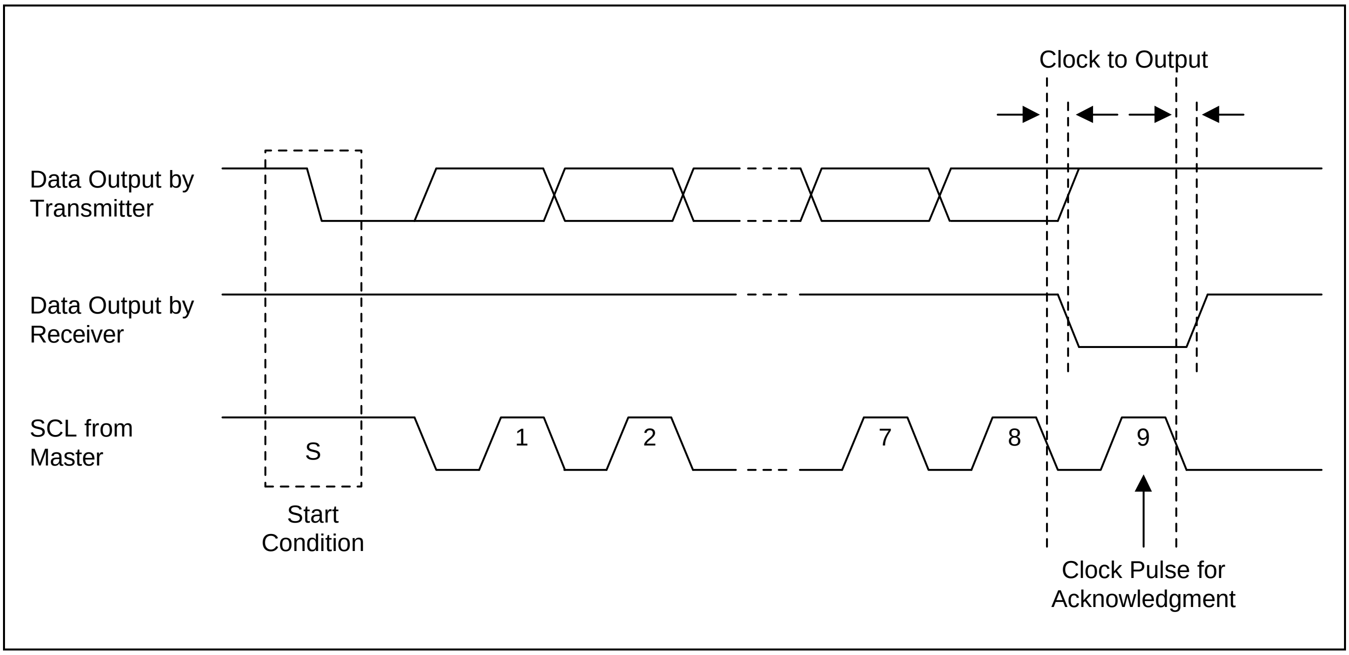


Figure 16-5. Acknowledge on the IIC-Bus

READ-WRITE OPERATION

In transmitter mode, after the data is transferred, the IIC-bus interface will wait until IICDSn (IIC-bus Data Shift Register) is written with a new data. Until the new data is written, the SCL line will be held low. After the new data is written to IICDSn register, the SCL line will be released. The S3C2800 should hold the interrupt to identify the completion of current data transfer. After the CPU receives the interrupt request, it should write a new data into IICDSn, again.

In receive mode, after the data is received, the IIC-bus interface will wait until IICDSn register is read. Until the new data is read out, the SCL line will be held low. After the new data is read out from IICDSn register, the SCL line will be released. The S3C2800 should hold the interrupt to identify the completion of the new data reception. After the CPU receives the interrupt request, it should read the data from IICDS.

BUS ARBITRATION PROCEDURES

Arbitration takes place on the SDA line to prevent the contention on the bus between two masters. If a master with a SDA high level detects another master with a SDA active low level, it will not initiate a data transfer because the current level on the bus dose not correspond to its own. The arbitration procedure will be extended until the SDA line will be High.

However when the masters simultaneously lower the SDA line, each master should evaluate whether or not the mastership is allocated to itself. For the purpose of evaluation, each master should detect the address bits. While each master generates the slaver address, it should also detect the address bit on the SDA line because the lowering of SDA line is stronger than maintaining High on the line. For example, one master generates a Low as first address bit, while the other master is maintaining High. In this case, both masters will detect Low on the bus because Low is stronger than High even if first master is trying to maintain High on the line. When this happens, Low (as the first bit of address) -generating master will get the mastership and High (as the first bit of address) - generating master should withdraw the mastership. If both masters generate Low as the first bit of address, there should be an arbitration for second address bit, again. This arbitration will continue to the end of last address bit.

ABORT CONDITIONS

If a slave receiver cannot acknowledge the confirmation of the slave address, it should hold the level of the SDA line High. In this case, the master should generate a stop condition and to abort the transfer.

If a master receiver is involved in the aborted transfer, it should signal the end of the slave transmit operation by canceling the generation of an ACK after the last data byte received from the slave. The slave transmitter should then release the SDA to allow a master to generate a Stop condition.

CONFIGURING THE IIC-BUS

To control the frequency of the serial clock (SCL), the 4-bit prescaler value can be programmed in the IICCONn register. The IIC-bus interface address is stored in the IIC-bus address register, IICADDn. (By default, the IIC-bus interface address is an unknown value.)

FLOWCHARTS OF THE OPERATIONS IN EACH MODE

The following steps must be executed before any IIC Tx/Rx operations.

1) Write own slave address on IICADDn register if needed.

2) Set IICCONn Register.
   a) Enable interrupt
   b) Define SCL period

3) Set IICSTATn to enable Serial Output.

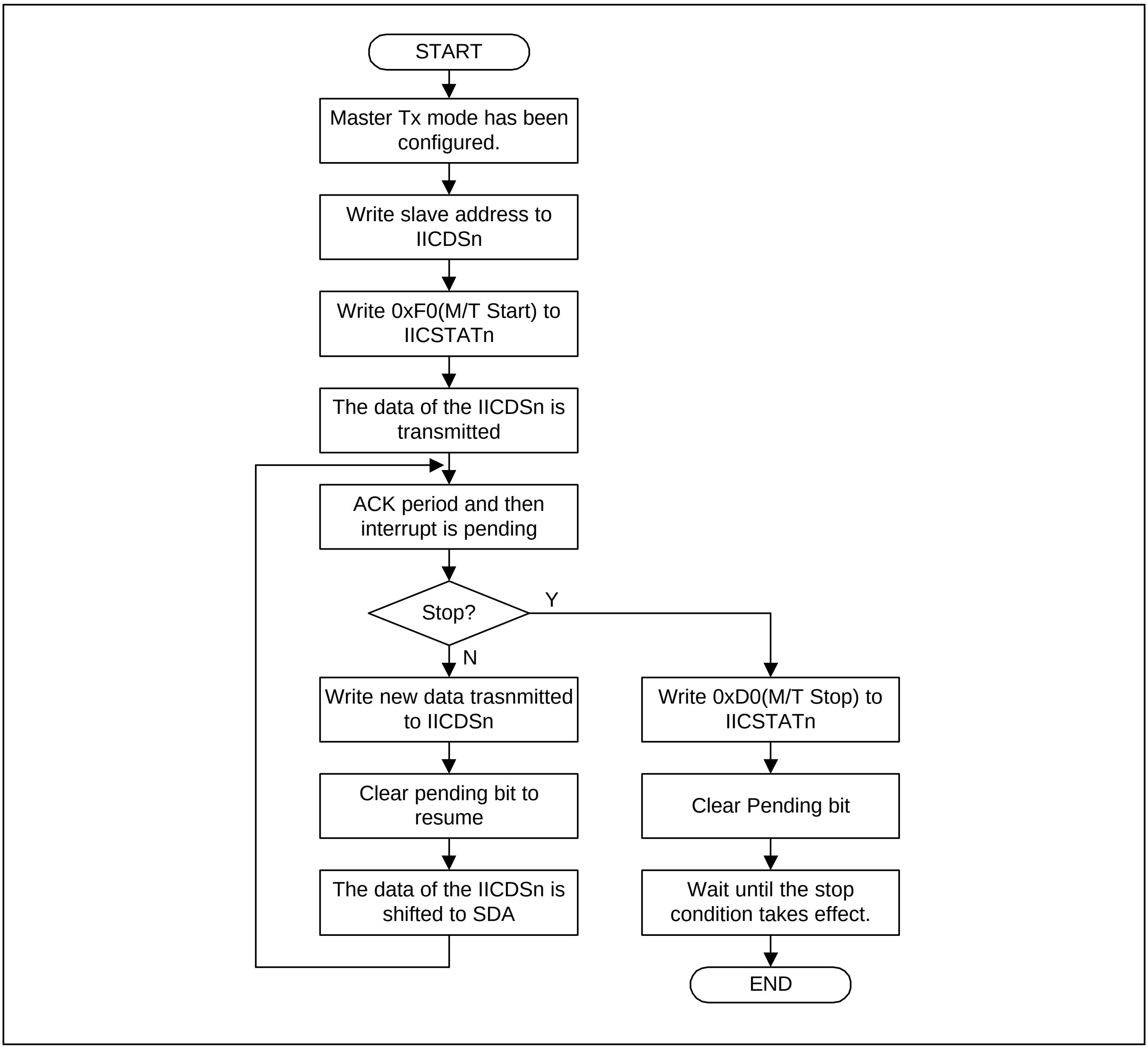

START
Master Tx mode has been configured.
Write slave address to IICDSn
Write 0xF0(M/T Start) to IICSTATn
The data of the IICDSn is transmitted
ACK period and then interrupt is pending
Stop?
Y
N
Write new data trasnmitted to IICDSn
Clear pending bit to resume
The data of the IICDSn is shifted to SDA
Write 0xD0(M/T Stop) to IICSTATn
Clear Pending bit
Wait until the stop condition takes effect.
END

Figure 16-6. Operations for Master/Transmitter Mode

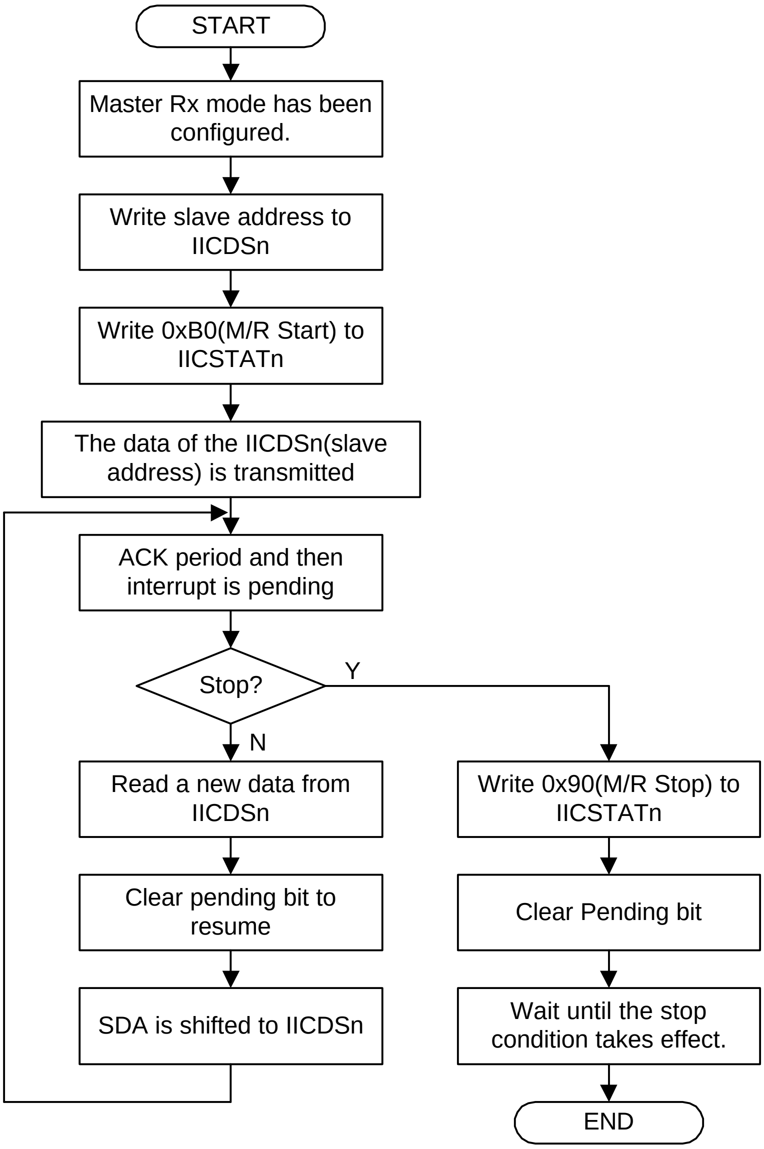

START
Master Rx mode has been configured.
Write slave address to IICDSn
Write 0xB0(M/R Start) to IICSTATn
The data of the IICDSn(slave address) is transmitted
ACK period and then interrupt is pending
Stop?
Y
N
Read a new data from IICDSn
Clear pending bit to resume
SDA is shifted to IICDSn
Write 0x90(M/R Stop) to IICSTATn
Clear Pending bit
Wait until the stop condition takes effect.
END

Figure 16-7. Operations for Master/Receiver Mode

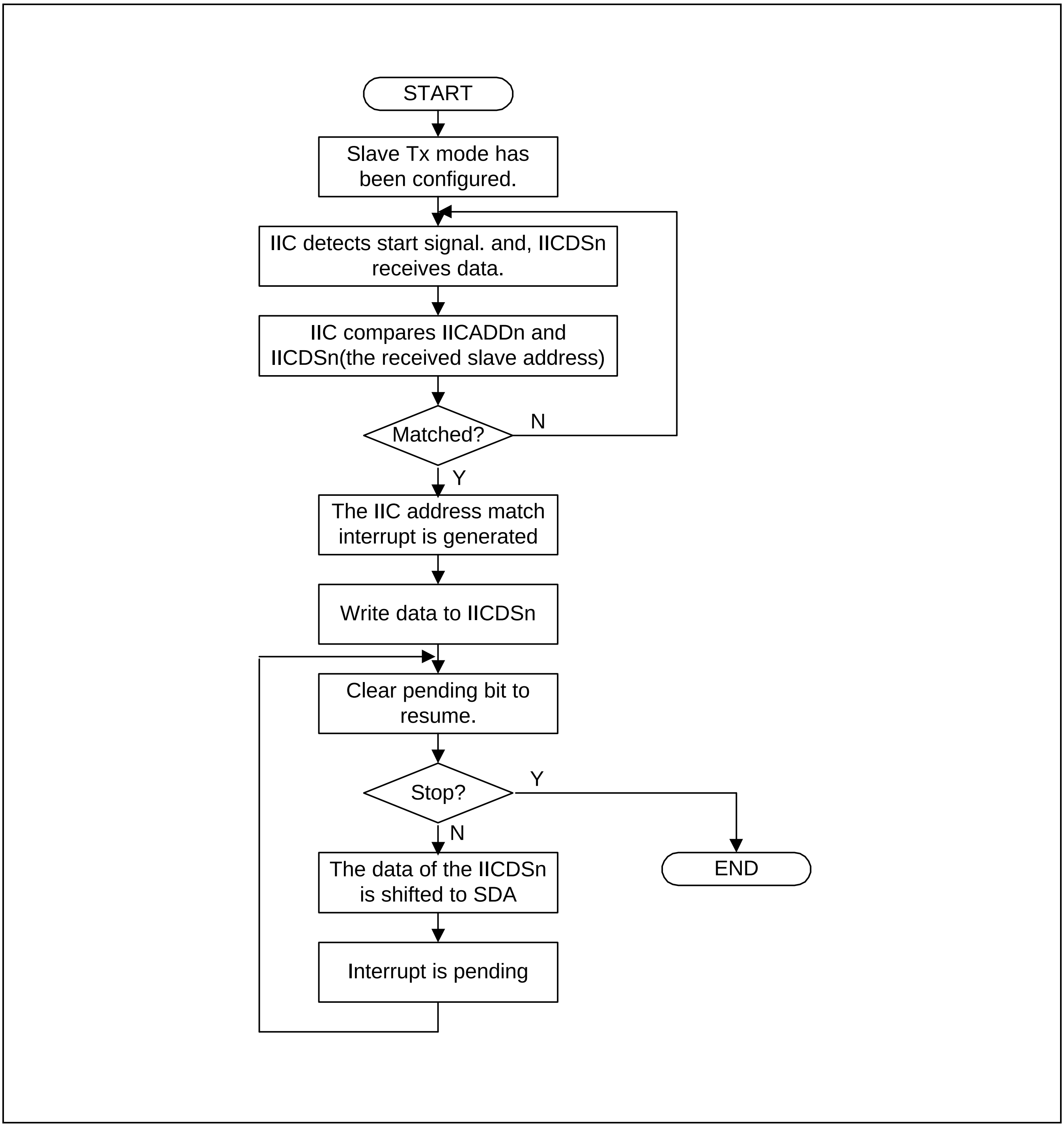

START
Slave Tx mode has been configured.
IIC detects start signal. and, IICDSn receives data.
IIC compares IICADDn and IICDSn(the received slave address)
Matched?
N
Y
The IIC address match interrupt is generated
Write data to IICDSn
Clear pending bit to resume.
Stop?
Y
N
END
The data of the IICDSn is shifted to SDA
Interrupt is pending

Figure 16-8. Operations for Slave/Transmitter Mode

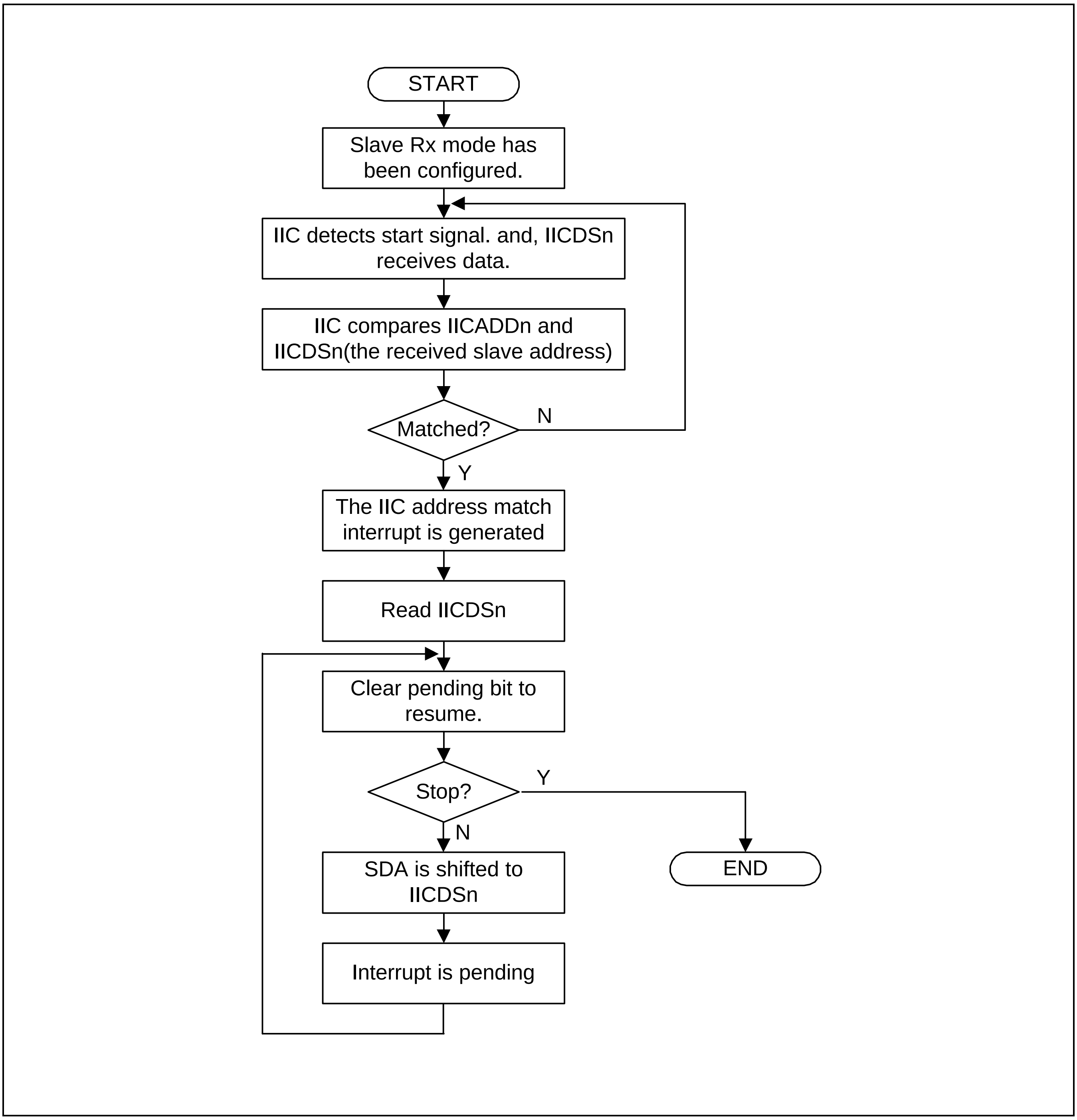

START
Slave Rx mode has been configured.
IIC detects start signal. and, IICDSn receives data.
IIC compares IICADDn and IICDSn(the received slave address)
Matched?
N
Y
The IIC address match interrupt is generated
Read IICDSn
Clear pending bit to resume.
Stop?
Y
N
SDA is shifted to IICDSn
END
Interrupt is pending

Figure 16-9. Operations for Slave/Receiver Mode

IIC-BUS INTERFACE SPECIAL FUNCTION REGISTERS

IIC-BUS CONTROL REGISTER (IICCONn)

Register	Address	R/W	Description	Reset Value
IICCON0	0x1019 0000	R/W	IIC-Bus 0 control register	0x0000 0020
IICCON1	0x101A 0000	R/W	IIC-Bus 1 control register	0x0000 0020

IICCONn	Bit	Description	Initial State
Acknowledge enable [1]	[6]	IIC-bus acknowledge enable bit 0=Disable ACK generation 1=Enable ACK generation In Tx mode, the IICSDA is free in the ack time In Rx mode, the IICSDA is Low in the ack time.	0
Tx clock source selection	[5]	Source clock of IIC-bus transmit clock prescaler selection bit 0= IICCLK = APBCLK /16 1= IICCLK = APBCLK /256	1
Tx/Rx Interrupt enable	[4]	IIC-Bus Tx/Rx interrupt enable/disable bit 0=Disable interrupt, 1=Enable interrupt	0
Transmit clock value [2]	[3:0]	IIC-Bus transmit clock prescaler IIC-Bus transmit clock frequency is determined by this 4-bit prescaler value, according to the following formula: Tx clock = IICCLK/(IICCON[3:0]+1)	0x0

NOTES:

1. Interfacing EEPROM, the ACK generation may be disabled before reading the last data in order to generate the STOP condition in Rx mode.
2. IICCLK is determined by IICCON[5].
   Tx clock can vary by SCL transition time.

Table 16-1. Example for Setting of the IICSCL

IIC_SCL (KHz)	APBCLK = 50MHz		APBCLK = 37.5MHz	
	IICCON[5]	IICCON[3:0]	IICCON[5]	IICCON[3:0]
100 (Real =73.2)	–	–	1=APBCLK/256	0x1=IICCLK/2
100 (Real = 97.7)	1=APBCLK/256	0x1=IICCLK/2	–	–
400 (Real = 390.6)	0=APBCLK/16	0x7=IICCLK/8	0=APBCLK/16	0x5=IICCLK/6

IIC-BUS CONTROL/STATUS REGISTER (IICSTATn)

Register	Address	R/W	Description	Reset Value
IICSTAT0	0x1019 0004	R/W	IIC-Bus 0 control/status register	0x0000 0000
IICSTAT1	0x101A 0004	R/W	IIC-Bus 1 control/status register	0x0000 0000

IICSTATn	Bit	Description	Initial State
Mode selection	[7:6]	IIC-bus master/slave Tx/Rx mode select bits: 00: Slave receive mode 01: Slave transmit mode 10: Master receive mode 11: Master transmit mode	0
Busy signal status/ START STOP condition	[5]	IIC-Bus busy signal status bit: 0 = read) IIC-bus not busy. (IIC always senses BUS start/stop condition.) write) IIC-bus STOP signal generation 1 = read) IIC-bus busy write) IIC-bus START signal generation. The data in IICDSn will be transferred automatically just after the start signal. Also, the delay to check the start condition is inserted automatically.	0
Serial output enable	[4]	IIC-bus data output enable/disable bit: 0=Disable Rx/Tx, 1=Enable Rx/Tx	0
Arbitration status flag	[3]	IIC-bus arbitration procedure status flag bit: 0 = Bus arbitration successful 1 = Bus arbitration failed during serial I/O	0
Address/Data field classification bit	[2]	IIC-bus Address/Data field classification bit 0 = When reset or START/STOP, or when the received data is in the data field. 1 = When received slave address matches to IICADDn register or general call.	0
Address zero status flag	[1]	IIC-bus address zero status flag bit: 0 = cleared when START/STOP condition was detected at the SDA/SCL line. 1 = Received slave address is 00000000b	0
Last-received bit status flag	[0]	IIC-bus last-received bit status flag bit 0 = Last-received bit is 0 (ACK was received) 1 = Last-receive bit is 1 (ACK was not received)	0

IIC-BUS ADDRESS REGISTER (IICADDn)

Register	Address	R/W	Description	Reset Value
IICADD0	0x1019 0008	R/W	IIC-Bus 0 address register	Undefined
IICADD1	0x101A 0008	R/W	IIC-Bus 1 address register	Undefined

IICADDn	Bit	Description	Initial State
Slave address	[7:0]	7-bit slave address, latched from the IIC-bus: When serial output enable=0 in the IICSTATn, IICADDn is write-enabled. The IICADDn value can be read any time, regardless of the current serial output enable bit (IICSTATn) setting. IICADDn is used only when the IIC mode is selected to slave receive/transmit mode. Slave address = [7:1] Not mapped = [0]	Undefined

IIC-BUS TRANSMIT/RECEIVE DATA SHIFT REGISTER (IICDSn)

Register	Address	R/W	Description	Reset Value
IICDS0	0x1019 000C	R/W	IIC-Bus 0 transmit/receive data shift register	Undefined
IICDS1	0x101A 000C	R/W	IIC-Bus 1 transmit/receive data shift register	Undefined

IICDSn	Bit	Description	Initial State
Data shift	[7:0]	8-bit data shift register for IIC-bus Tx/Rx operation: When serial output enable = 1 in the IICSTATn, IICDSn is write-enabled. The IICDSn value can be read any time, regardless of the current serial output enable bit (IICSTATn) setting NOTE: The bit[0] of the data, which is transferred just after start condition, is determined by the mode selection bit. If the mode selection bit is "receive", the bit will be 1 (read). If the mode selection bit is "transmit", the bit will be 0 (write).	Undefined

NOTES

17 PCI-BUS INTERFACE

OVERVIEW

The PCI Controller is divided into two blocks as shown in Figure 17-1: BIF(AHB Bus Interface) and PCI (PCI bus Interface). BIF block including BIF SFR (Special Function Register) operates in AHB clock domain and PCI block including PCI configuration registers in PCI clock domain.

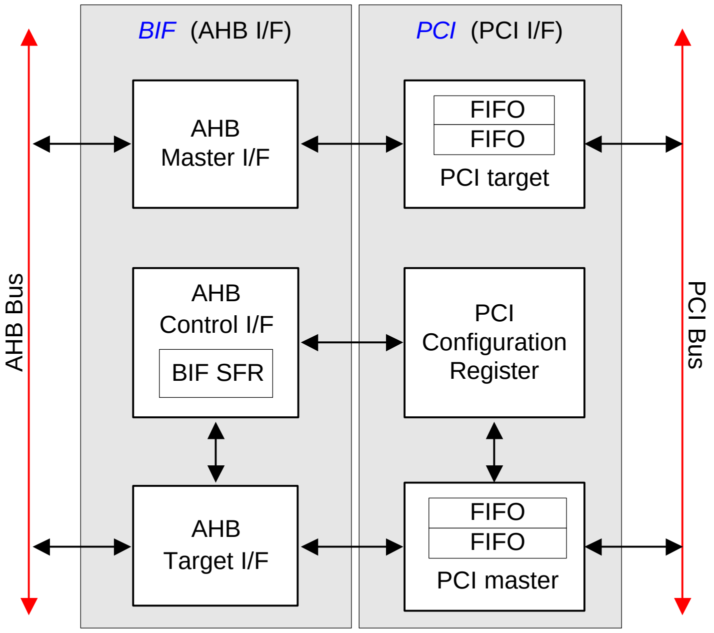


Figure 17-1. PCI Controller Block Diagram

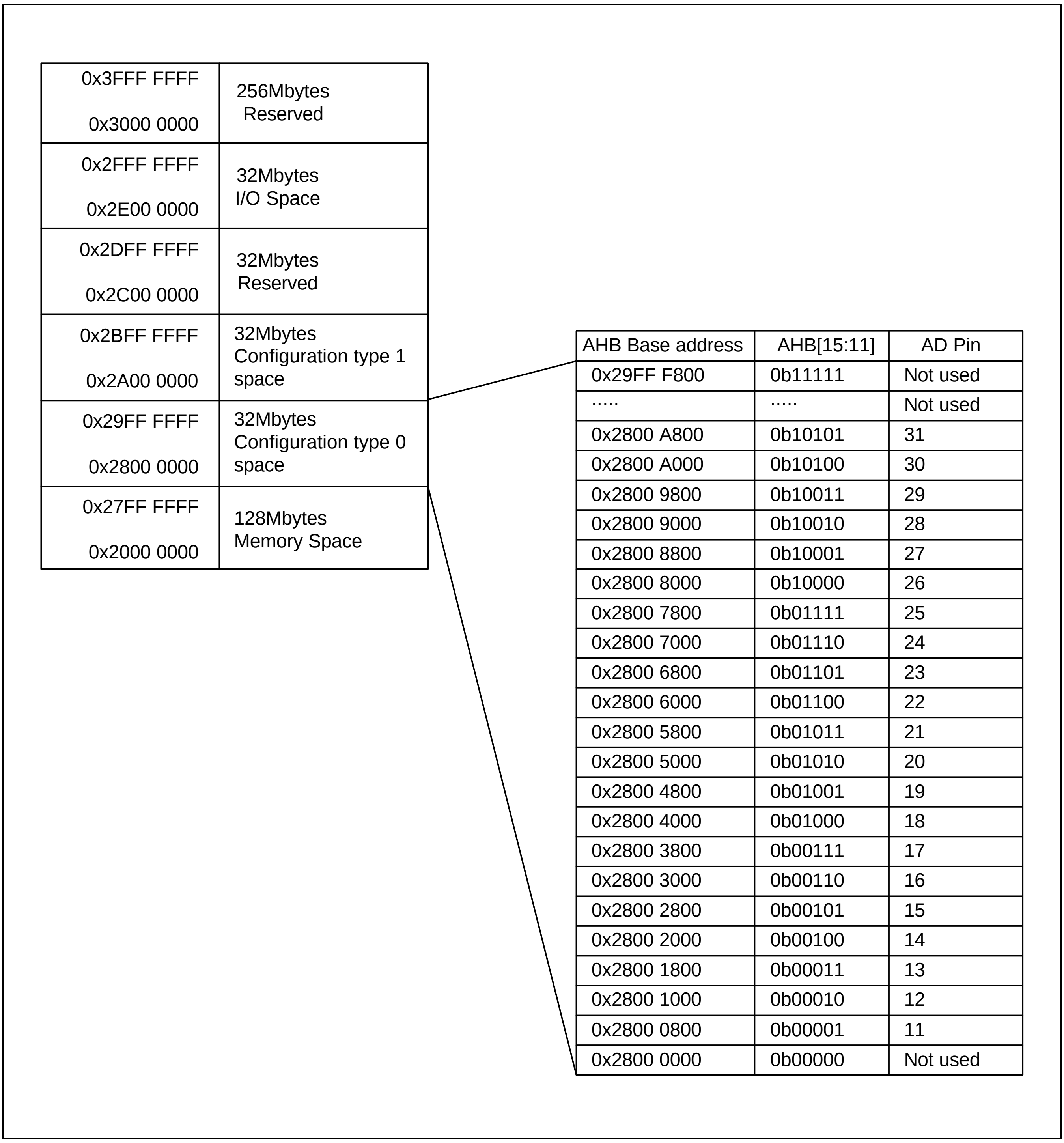
FEATURE

- PCI master and target device
- 32-bit bus width and supports 33/66 MHz at 3.3V
- PCI Local Bus Specification Rev. 2.2 compliant
- System clock can run asynchronously to PCI clock
- Supports three base address decoding
  1. Memory base address 0 : prefetchable, 16Bytes – 2GB programmable size
  2. Memory base address 1 : prefetchable, 16Bytes – 2GB programmable size
  3. I/O base address 2 : non-prefetchable, 256-byte fixed size
- Four independent 8-word deep FIFO
- Supports base address translation from PCI bus to AHB bus
- Supports noncontiguous byte enable transfer
- Supports target fast back-to-back cycle operation
- Supports target lock operation through PCI_nLOCK pin
- Supports PCI parity generation and check
- Supports 8-bit DAC (Dual Access Cycle) : 40-bit address space
- Little-endian type PCI interface

TERMINOLOGY OF REGISTER GROUPS

- PCI SFR (PCI Special Function Registers) : all PCI special function registers including BIF SFR and PCI configuration registers
- BIF SFR (BIF Special Function Registers) : Registers in BIF block cannot be accessed by configuration cycle
- PCI configuration registers : PCI configuration header (can be accessed by configuration cycle)

PCI ADDRESS SPACE DEFINE



0x3FFF FFFF 0x3000 0000	256Mbytes Reserved
0x2FFF FFFF 0x2E00 0000	32Mbytes I/O Space
0x2DFF FFFF 0x2C00 0000	32Mbytes Reserved
0x2BFF FFFF 0x2A00 0000	32Mbytes Configuration type 1 space
0x29FF FFFF 0x2800 0000	32Mbytes Configuration type 0 space
0x27FF FFFF 0x2000 0000	128Mbytes Memory Space

AHB Base address	AHB[15:11]	AD Pin
0x29FF F800	0b11111	Not used
…..	…..	Not used
0x2800 A800	0b10101	31
0x2800 A000	0b10100	30
0x2800 9800	0b10011	29
0x2800 9000	0b10010	28
0x2800 8800	0b10001	27
0x2800 8000	0b10000	26
0x2800 7800	0b01111	25
0x2800 7000	0b01110	24
0x2800 6800	0b01101	23
0x2800 6000	0b01100	22
0x2800 5800	0b01011	21
0x2800 5000	0b01010	20
0x2800 4800	0b01001	19
0x2800 4000	0b01000	18
0x2800 3800	0b00111	17
0x2800 3000	0b00110	16
0x2800 2800	0b00101	15
0x2800 2000	0b00100	14
0x2800 1800	0b00011	13
0x2800 1000	0b00010	12
0x2800 0800	0b00001	11
0x2800 0000	0b00000	Not used

Figure 17-2. PCI Address Space Define

ACCESS FROM AHB BUS OR ARM CPU

PCI controller is accessed by ARM CPU using the following address region. BIF SFR is accessed without wait states. But there is some latency in accessing PCI configuration registers. because PCI configuration registers are updated based on the PCI clock.

AHB Memory Map :

0x1008 0000 – 0x1008 0043 : PCI configuration registers

0x1008 0100 – 0x1008 0157 : BIF SFR (Special Function Registers)

ACCESS FROM PCI BUS OR OTHER PCI DEVICE

PCI configuration registers are directly accessed by other PCI master through PCI configuration cycle. Using configuration cycle, however, any other address space in PCI block cannot be accessed. Peripherals' SFR and external memory space must be accessed using memory or I/O cycle with proper address defined in base address bar of PCI configuration registers. PCI SFR (BIF SFR and PCI configuration registers) can be also accessed by this mechanism. (As a result, PCI configuration registers can be accessed using two methods. However, read/write properties are different for two methods.)

At reset, Memory base address bar 1 and I/O base address bar 2 are mapped to BIF SFR region, but base address bar 0, 1, or 2 can be re-programmed to map to any region on AHB bus.

Refer to PCI Functional Description for address translation.

BASE ADDRESS BAR

There are three base address bars in PCI configuration registers (10h~1Bh). These base address bars are used for direct access from PCI bus to AHB bus.

Table 17-1. PCI Base Address Bar Type

Address Bar	Memory base address bar 0	Memory base address bar 1	I/O base address bar 2
PCI Configuration Register Address Offset	10h	14h	18h
PCI Access Cycle	Memory	Memory	I/O
PCI Decoding Speed	Fast/Medium	Medium	Medium
Prefetchable Property	Yes	Yes	No
Size Property	Programmable	Programmable	Fixed
Range of Size	16Bytes – 2GB	16Bytes – 2GB	256 Bytes
BIF SFR defining Size	PCIBAM0	PCIBAM1	PCIBAM2 (R/O)
BFI SFR related Address Translation	PCIBATPA0	PCIBATPA1	PCIBATPA2
Size at Reset	64Kbytes	512 Bytes	256 Bytes
Address Translation (Mapping) at Reset	0x0000 0000 (Bank 0)	0x1008 0000 (PCI & BIF SFR)	0x1008 0100 (BIF SFR)

On reset, memory base address bar 1 and I/O base address bar 2 are mapped to BIF SFR region and memory base address bar 0 is mapped to other region (refer to PCI SFR reset value). However, the mapping can be changed by re-initializing (re-mapping) destination address (PCIBATPAn registers) using serial-EEPROM data or initialization program in flash memory. According to the PCI local bus spec. rev. 2.2, the size of each base address bar should remain fixed after host/PCI reads it.

Prefetchable property of base address bars is fixed (cannot be changed). In case of PCI write operation, PCI Controller will gather several data in FIFO and does a burst or single operation irrespective of prefetchable property. On the other hand, PCI read operation requires many more cycles on PCI bus than PCI write operation. If any master read from prefetchable regions, PCI Controller does burst read transfers (prefetching) repeatedly on AHB bus irrespective of master's read count. But if any master read from non-prefetchable region (indicated by I/O bar 2), PCI Controller does only single read transfers repeatedly on AHB bus for each PCI cycle. In case of PCI read operation, PCI Controller issues 'retry' to PCI bus until data is ready. Therefore, if mapping of prefetchable memory base address translation registers (PCIBATPA0, PCIBATPA1) includes non-prefetchable region such as SFR (for example, some peripheral's SFR is not prefetchable), some data can be lost because read operation may not complete before the next address is issued from master.

PCI base address registers (PCIBAR0,1,2) must be set to non-zero value because this registers do not support PCI address 0.

Refer to PCIBATPA0, PCIBATPA1, PCIBATPA2, PCIBAM0, PCIBAM1, and PCIBAM2 in BIF SFR for programming base address bar.

PCI BUS FUNCTIONAL DESCRIPTION

The following sections provide a functional description of the PCI BUS controller operations.

PCI BUS TRANSFERS

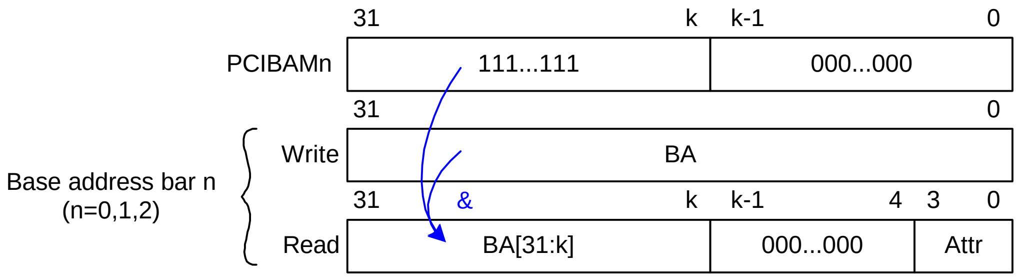
Refer to PCI Local Bus Specification Rev. 2.2 for information on PCI BUS transactions in detail.

SIZE OF BASE ADDRESS BAR SPACE

Base address bar 0, 1, and 2 registers are written through PCI bus from host/PCI bridge (host system software). Software can determine the size of required address space by finding the implemented bit position and write the start address of unique address space.

Since address space requirement can vary from applications to applications, the size of required address space can be specified by PCIBAMn (n=0,1,2) registers. These three registers must be set before read by system software and should remain unchanged once set.

The size of address space should be a power of 2 (2^k bytes) and aligned. According to PCI Local Bus Spec. Rev. 2.2, the size of address space can be from 16 bytes to 2GB. Memory base address bar 0 and 1 are programmable to this full range but I/O base address bar 2 is fixed to 256 bytes.



If any BA is written to base address bar, lower bits [k-1:4] is read as 0 by being masked by PCIBAMn. To determine the size of base address space, host system software write all 1s to base address bar and read it again. Host sytem software can get to know the size by finding k.

Size of Address Space = 2^k bytes
PCI Spec. Rev. 2.2: 4<=k<=31 (16Bytes<=size<=2GB)

Figure 17-3. Determining the Size of Base Address Bar Space

ADDRESS TRANSLATION BETWEEN PCI BUS AND AHB BUS

- When PCICON [4]="1", automatically translates from the AHB bus address to the PCI bus space address
- When PCICON [4]="0", all PCI address translation has to be done manually using PCISET register.

When PCICON [04]="1", the method of automatic translation from the AHB bus address to the PCI bus address:

Configuration Space (Type 0)

Configuration space type 0 (configuration address) is “0x2800” in the AHB address [31:16] and the device is selected by the device number (configuration address[15:11]). The function number (configuration address[10:8]) and register number (configuration address[7:2]) are transferred to the PCI address bus, as is.

Because the slave **IDSEL#** pin is connected to the host PCI bus address pin (AD11–) in a system with a PCI bus, one of the PCI bus address 11–31 is compatible to S3C2800 device number (configuration address[15:11]).

For example, the read function for the AHB address is 0x28006810:

AHB[31:16] = 0x2800; select configuration type 0 space

AHB[15:11] = 0b01101; PCI address bus pin 23 asserts to low. The device which has the slave device's **IDSEL#** pin connected to S3C2800 PCI address pin 23 is selected.

AHB[10:2] = 0x10; The register number 0x10 of the function number 0 is selected.

AHB[1:0] = Configuration type 0 address type.

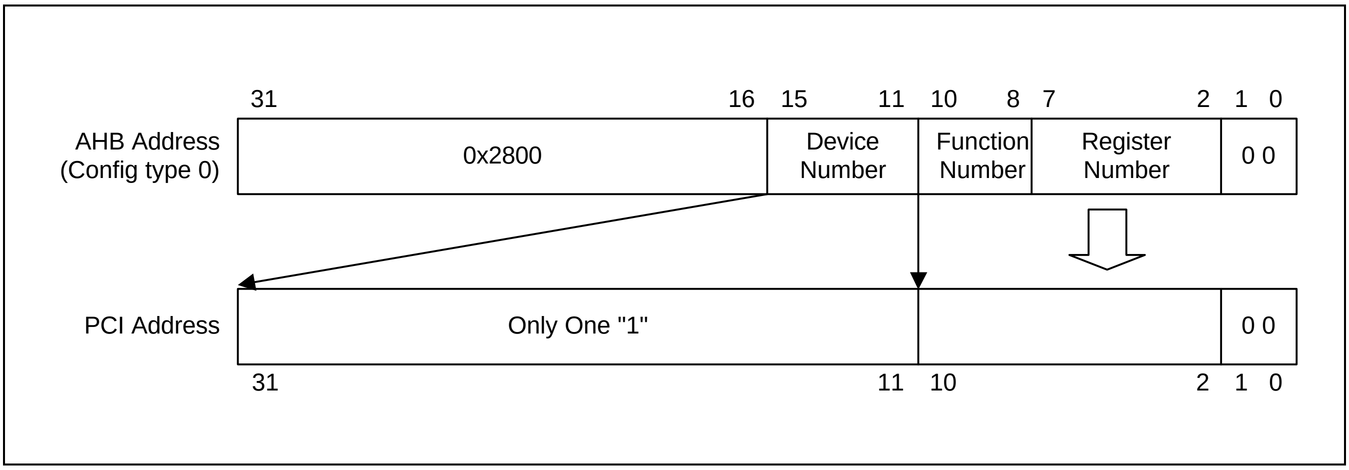


Figure 17-4. Host Bridge Translation for Type0 Configuration PCI Address from AHB Address

Configuration Space (Type 1)

The PCI address for configuration type 1 space is translated to AHB address 0x2a00_0000 – 0x2bff_ffff, as shown in Figure 17-5. Because AHB address[31:24] is translated to a reserved PCI address space, it is used as an index to indicate configuration type1 space and does not affect the PCI address.

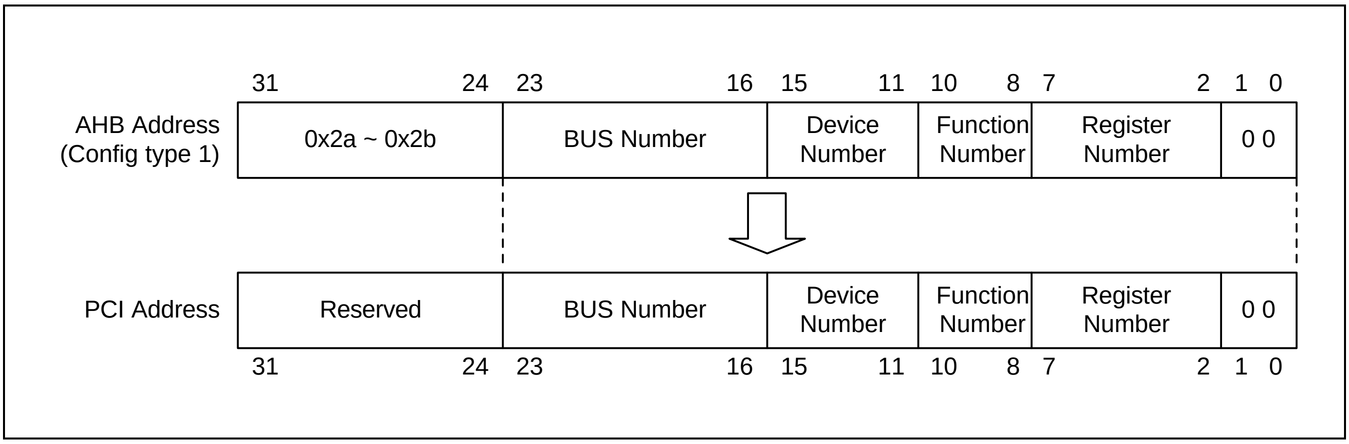


Figure 17-5. Host Bridge Translation for Type1 Configuration PCI Address from AHB Address

Memory space and IO space (Address translation From AHB bus to PCI bus; Master mode)

Unlike the configuration space address translation, the memory space address translation must use the BIF register PCIBATAPM. That is, when the address is read or written to the master as an AHB address 0x2000_0000 – 0x27ff_ffff , 5 bits in the AHB address[31:27] and the upper 5 bits in the PCIBATAPM register (PCIBATAPM[31:27]) are exchanged to generate the PCI address.

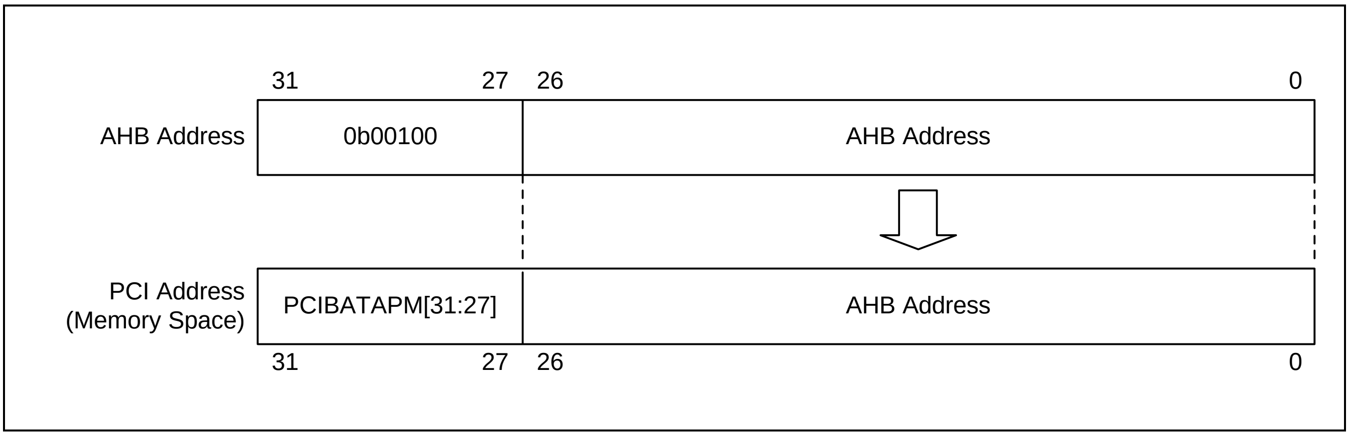


Figure 17-6. Host Bridge Translation for Memory Space PCI Address from AHB Address

Unlike the configuration space address translation, the IO space address translation must use the BIF register PCIBATAPI. That is, when the address is read or written to the Master as an AHB address 0x2000_0000 – 0x2fff_ffff, 7 bits in the AHB address[31:25] and the upper 7 bits in the PCIBATAPI register (PCIBATAPI[31:25]) are exchanged to generate the PCI address.

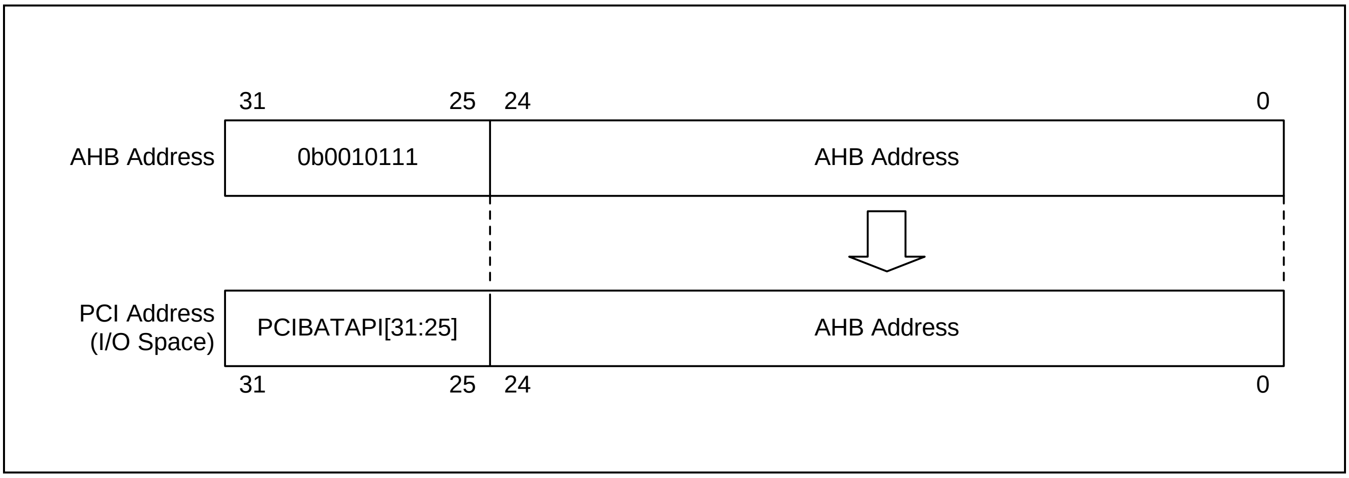


Figure 17-7. Host Bridge Translation for IO Space PCI Address from AHB Address

Memory Space and IO Space (Address Translation from PCI Bus to AHB Bus; Target Mode)

Address translation related PCI SFR for each base address bar include:

- Base address bar 0, 1, 2 of PCI configuration registers
- PCIBAM 0, 1, 2 of BIF SFR
- PCIBATPA 0, 1, 2 of BIF SFR

PCIBAMn registers are fixed value for the size of address space and effect the base address bar register and PCIBATPAn registers. For the first time, AHB address is generated only when PCI address is in one of base address bar space (“hit”). If PCI address is hit, AHB address is equal to PCI address which upper bits are replaced with PCIBATPAn as follows;

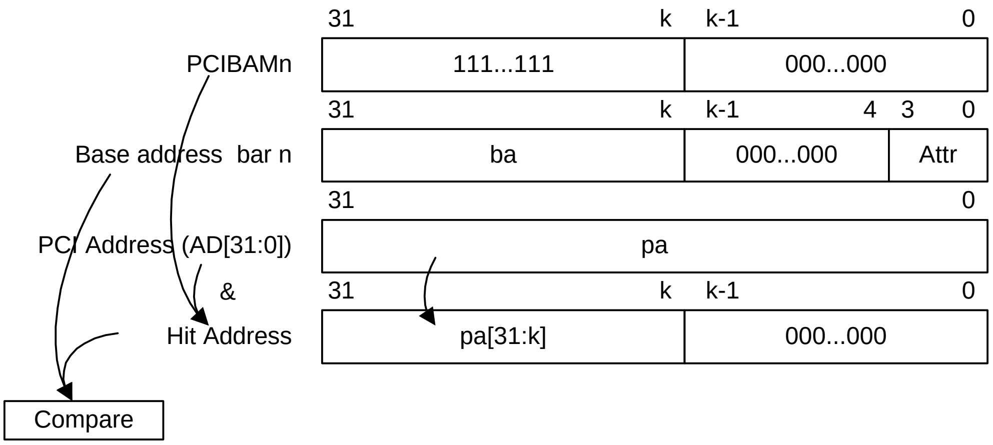
AHB address = ((PCIBATPAn & PCIBAMn) | ((PCI address) & –PCIBAMn))

On the other hand, to get the PCI address for specified AHB address, following formulas are used.

PCIBATPAn = (AHB address) & PCIBAMn

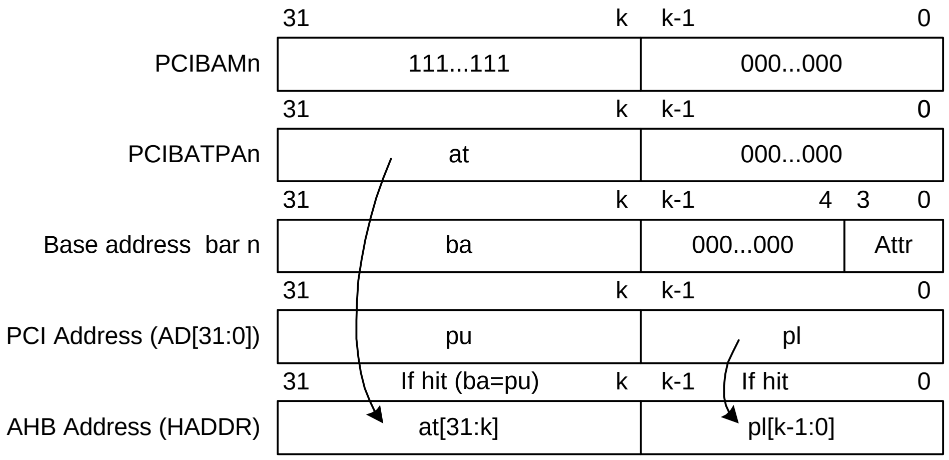
PCI address = ((Base address bar) & PCIBAMn) | ((AHB address) & –PCIBAMn))

To access specified address on AHB bus, PCIBATPAn of BIF SFR must be set to point to that address. And then that address can be accessed through PCI bus with PCI address which upper bits are replaced with base address bar n.



If base address bar n is equal to hit address (ba==pa[31:k]),
it indicates that PCI address is hit and PCI&CardBus Target Controller
assert DEVSEL# on PCI bus.

Figure 17-8. Comparison between PCI address and base address bar to check hit



AHB address are equal to PCI address that upper bits of PCI address
are replaced with PCIBATPAn. (k is defined by PCIBAMn)

Figure 17-9. Address Translation from PCI&CardBus Bus to AHB Bus

DATA TRANSFER BETWEEN PCI BUS AND AHB BUS (EXTERNAL MEMORY)

The data transfer between PCI bus and AHB bus (e.g., S3C2800 external memory) is always performed in Little_endian format regardless of the endian mode of S3C2800.

(1) When S3C2800 operates in Little-endian mode:

Both data transfers of CPU core to/from AHB bus and PCI to/from AHB bus are done in Little-endian format

(2) When S3C2800 operates in Big-endian mode:

Data transfer between CPU core and AHB bus is done in Big-endian format; On the other hand, PCI bus to/from AHB bus transfer is done in Little-endian format.

Figure 17-10 shows how data is transfered among PCI bus, AHB bus and external memory.

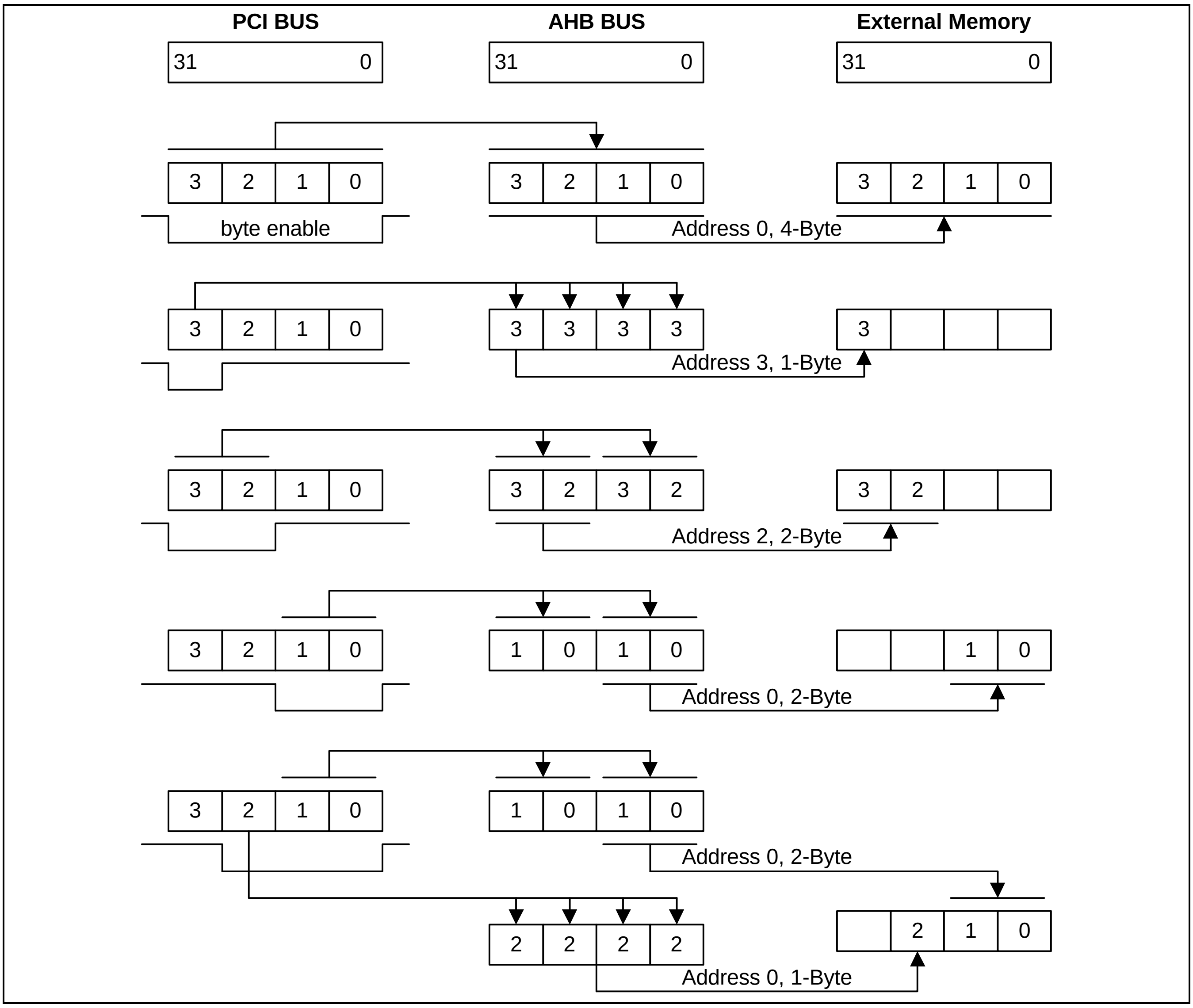


Figure 17-10. Data Transfer Between PCI Bus and AHB Bus

PCI INTERRUPT DESCRIPTION

There are two types of interrupt support for PCI controller an internal ARM CPU interrupt via interrupt controller and PCI interrupt through INTA# on PCI bus.

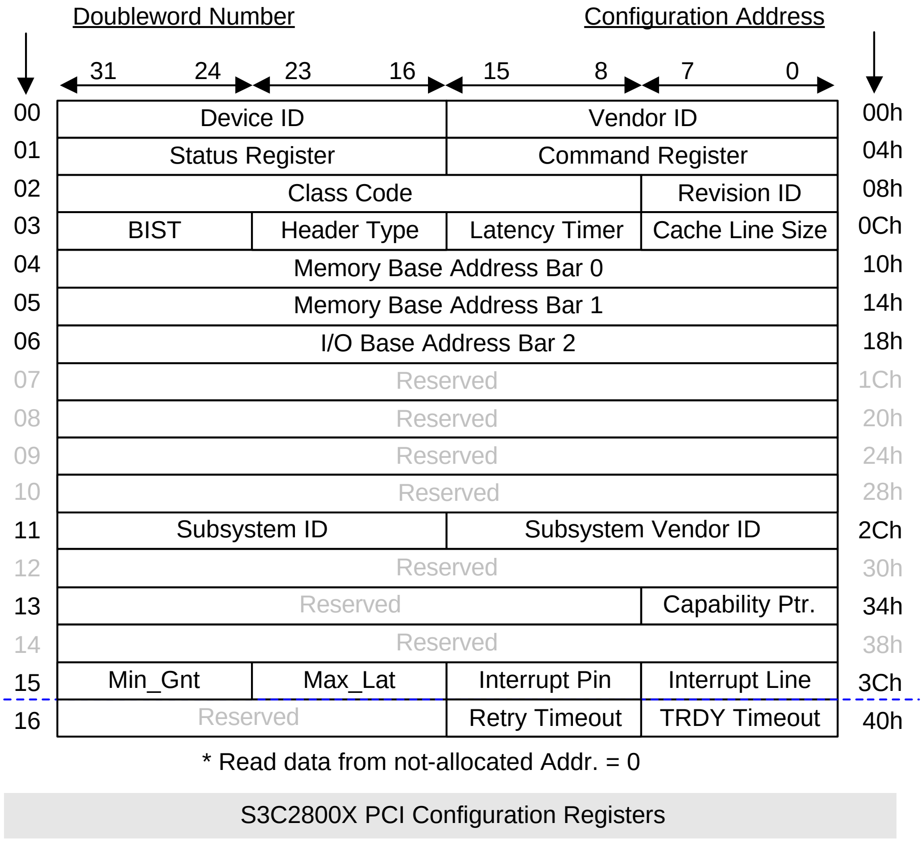
Internal ARM CPU Interrupt

The internal ARM CPU interrupt is controlled by PCIINTEN and PCIINTST register of BIF SFR. When each bit of PCIINTEN and PCIINTST is set, internal interrupt will be asserted to interrupt controller. To clear the interrupt pending in interrupt service routine, it can be cleared by writing 1 to corresponding bit of PCIINTST.

NOTE: To clear the PCI interrupt pending bit, the corresponding bit in the PCIINTST must be cleared first, and then the corresponding bit in the SRCPND register must be cleared.

PCI SPECIAL FUNCTION REGISTERS (PCI CONFIGURATION REGISTERS)

As explained above, PCI SFR (PCI Special Function Registers) has BIF SFR and PCI configuration registers.



Doubleword Number	31 24	23 16	15 8	7 0	Configuration Address
00	Device ID		Vendor ID		00h
01	Status Register		Command Register		04h
02	Class Code			Revision ID	08h
03	BIST	Header Type	Latency Timer	Cache Line Size	0Ch
04	Memory Base Address Bar 0				10h
05	Memory Base Address Bar 1				14h
06	I/O Base Address Bar 2				18h
07	Reserved				1Ch
08	Reserved				20h
09	Reserved				24h
10	Reserved				28h
11	Subsystem ID		Subsystem Vendor ID		2Ch
12	Reserved				30h
13	Reserved			Capability Ptr.	34h
14	Reserved				38h
15	Min_Gnt	Max_Lat	Interrupt Pin	Interrupt Line	3Ch
16	Reserved		Retry Timeout	TRDY Timeout	40h

* Read data from not-allocated Addr. = 0

S3C2800X PCI Configuration Registers

Figure 17-11. PCI Configuration Registers

All configuration registers are described in : "PCI Local Bus Specification Revision 2.2".

Table 17-2. PCI Configuration Registers Overview

Offset	Bits	Name	R/W		Description	Reset Value
			AHB	PCI		
0x00	[15:0]	Vendor ID	R/O	R/O	Chip vendor identification	0x144D
	[31:16]	Device ID	R/O	R/O	Chip device identification	0x2800
0x04	[15:0]	Command Register	R/W	R/O	Basic control register to perform PCI access	0x0000
	[31:16]	Status Register	R/WC	R/WC	PCI bus-related status register	0x02B0
0x08	[7:0]	Revision ID	R/W	R/O	Identifies the revision number of the device	0x01
	[31:8]	Class Code	R/W	R/O	Identifies the basic function of the device	0x0D8 0000
0x0C	[7:0]	Cache Line Size	R/W	R/W	System cache line size	0x00
	[15:8]	Latency Timer	R/W	R/W	Maximum clocks that master can own the bus	0x00
	[23:16]	Header Type	R/O	R/O	Indicates a single or multi-function	0x00
	[31:24]	BIST	R/O	R/O	Register for built-in self-test	0x00
0x10	[31:0]	Memory Base Address 0	R/W	R/O	Memory bar 0 size and location (of fast decode)	0x0000 0008
0x14	[31:0]	Memory Base Address 1	R/W	R/O	Memory bar 1 size and location (of medium decode)	0x0000 0008
0x18	[31:0]	I/O Base Address	R/W	R/O	I/O bar size and location (of medium decode)	0x0000 0001
0x1C–0x2B		Reserved			Reserved	
0x2C	[15:0]	Subsystem Vendor ID	R/W	R/O	Add-in card or subsystem vendor identification	0x144D
	[31:16]	Subsystem ID	R/W	R/O	Add-in card or subsystem identification	0x2800
0x30–0x33		Reserved			Reserved	
0x34	[7:0]	Capabilities Pointer	R/O	R/O	Additional set of linked list registers	0xDC
	[31:8]	Reserved			Reserved	
0x38–0x3B		Reserved			Reserved	
0x3C	[7:0]	Interrupt Line	R/W	R/O	Interrupt request line routing information (IRQn)	0x00
	[15:8]	Interrupt Pin	R/W	R/O	Interrupt request pin number (INTA#)	0x00
	[23:16]	Min_Gnt	R/W	R/O	Minimum time of how long master needs burst period	0x00
	[31:24]	Max_Lat	R/W	R/O	Maximum time of how often device needs to gain access	0x00
0x40	[7:0]	TRDY Timeout	R/W	R/W	Maximum time of master wait for TRDY#	0x80
	[16:8]	Retyr Timeout	R/W	R/W	Maximum number of master retry	0x80
	[31:17]	Reserved			Reserved	

NOTE: R/O = Read-only, R/W = Read and Write, R/WC = Read and Write 1 to clear.

PCI VENDOR ID & DEVICE ID REGISTER (PCIVDIDR)

This register identifies the manufacture of the device and the particular device.

Register	Address	R/W	Description	Reset Value
PCIVDIDR	0x1008 0000	R	PCI vendor ID and device ID register	0x2800 144D

PCIVDIDR	Bit	Description	Initial State
DEVID	[31:16]	PCI device identification (Read-only) This field identifies the particular device. This identifier is allocated by the vendor.	0x2800
VENID	[15:0]	PCI vendor identification (Read-only) This field identifies the manufacturer of the device.	0x144D

PCI STATUS & COMMAND REGISTER (PCISCR)

The Status register (PCISCR[31:16]) is used to record the status information of PCI bus related events. Reserved bits are read-only and return zero when read. Reads to this register behave normally. Writes are slightly different in that bits can be reset, but no set. A one bit is reset whenever the register is written, and the write data in the corresponding bit location is a 1.

The Command register (PCISCR[15:0]) provide coarse control over a device's ability to generate and respond to PCI cycles. When this register is set to 0, the device is logically disconnected from the PCI bus for all accesses except configuration accesses.

Register	Address	R/W	Description	Reset Value
PCISCR	0x1008 0004	R/W	PCI status and command register	0x02B0 0000

PCISCR	Bit	Description	Initial State
Detected Parity Error	[31]	Detected parity error status bit (Read or write-1-to-clear) This bit set by a device whenever it detects a parity error, regardless of the state of the parity error response bit (PCISCR[6]). This bit is required to be set by the device when any of the following conditions occurs: 1) The device's parity checking logic detects an error in a single address cycle or either address phase of a dual address cycle. 2) The device's parity checking logic detects a data parity error and the device is the target of a write transaction. 3)The device's parity checking logic detects a data parity error and the device is the master of a read transaction.	0
Signaled System Error	[30]	Signaled system error bit (Read or write-1-to-clear) This bit set whenever the device asserts **SERR**#.	0

PCISCR	Bit	Description	Initial State
Received Master Abort	[29]	Received master abort bit (Read or write-1-to-clear) This bit set by a master device whenever its transaction (except for Special Cycle) is terminated with Master-Abort.	0
Received Target Abort	[28]	Received target abort bit (Read or write-1-to-clear) This bit set by a master device whenever its transaction is terminated with Target-Abort.	0
Signaled Target Abort	[27]	Signaled target abort bit (Read or write-1-to-clear) This bit set by a target device whenever it terminates a transaction with Target-Abort.	0
DEVSEL timing	[26:25]	DEVSEL# response timing bits (Read or write-1-to-clear) These bits encode the timing of **DEVSEL**#. Three allowable timings for assertion of **DEVSEL**#. 00 = fast timing 01 = medium timing 10 = slow timing 11 = reserved It asserts DEVSEL# on the second clock after FRAME# is assert by a PCI master attempting to access memory. These bits are read-only and indicate the slowest time that a device asserts **DEVSEL#** for any bus command except Configuration Read and Configuration Write.	01
Master Data Parity Error	[24]	Master data parity error status bit (Read or write-1-to-clear) If the parity response bit (PCISCR[6] is cleared, the master not set this bit, even if the master detects a parity error or the target asserts PERR#. Targets never set this bit. This bit is only implemented by bus masters. It is set when three conditions are met: 1) the bus agent asserted **PERR#** itself (on a read) or observed **PERR#** asserted (on a write); 2) the agent setting the bit acted as the bus master for the operation in which the error occurred; 3) the Parity Error Response bit (PCISCR[6]) is set.	0
Fast Back-to-Back Capable	[23]	Fast back to back capable bit (Read or write-1-to-clear) This bit indicates whether or not the target is capable of accepting fast back-to-back transactions when the transactions are not to the same agent. 0 = Not capable the fast back-to-back transaction. 1 = Capable the fast back-to-back transaction.	1
Reserved	[22]	Reserved	0

PCISCR	Bit	Description	Initial State
66MHz Capable	[21]	This bit indicates whether or not this device is capable of running at 66 MHz. (Read or write-1-to-clear) 0 = Capable 33 MHz. 1 = capable 66 MHz.	1
Capabilities List	[20]	Capabilities list pointer bit (Read or write-1-to-clear) This bit indicates whether or not this device implements the pointer for a New Capabilities linked list at offset 34h. 0 = No available New Capabilities linked list. 1 = Available New Capabilities linked list.	1
Reserved	[19:10]	Reserved	
Fast Back-to-Back Enable	[9]	Fast back-to-back write enable bit This bit controls whether or not a master can do fast back-to-back transactions to different devices. Initialization software will set the bit if all targets are fast back-to-back capable. 0 = The master is allowed to generate fast back-to-back transactions to the same agent. 1 = The master is allowed to generate fast back-to-back transactions to different agents.	0
SERR# Enable	[8]	This bit is an enable bit for the **SERR#** driver. 0 = disables the **SERR#** driver. 1 = enables the **SERR#** driver. Address parity errors are reported only if this bit and parity error response bit (PCISCR[6]) are 1.	0
Stepping Control	[7]	This bit is used to control whether or not a device does address/data stepping. 0 = Devices that never do stepping. 1 = Devices that always do stepping. **NOTE**: S3C2800 does not support.	0

PCISCR	Bit	Description	Initial State
Parity Error Response	[6]	This bit controls the device's response to parity errors. 0 = The device sets its Detected Parity Error status bit (PCISCR[31]) when an error is detected, but does not assert **PERR#** and continues normal operation. 1 = The device must take its normal action when a parity error is detected.	0
VGA Palette Snoop	[5]	This bit controls how VGA compatible and graphics devices handle accesses to VGA palette registers. 0 = Disable palette snooping. (The device should treat palette write accesses like all other accesses.) 1 = Enable palette snooping. (The device does not respond to palette register writes and snoops the data.) **NOTE** : S3C2800 does not support.	0
Memory Write and Invalidate Enable	[4]	This is an enable bit for using the Memory Write and Invalidate command. 0 = Disable the command (Memory Write command is used). 1 = Enable the memory write and invalidate command. **NOTE** : S3C2800 does not support.	0
Special Cycle	[3]	Controls a device's action on Special Cycle operations. 0 = Disable Special Cycle operations. 1 = Enable Special Cycle operations. **NOTE** : S3C2800 does not support.	0
Bus Master	[2]	Controls a device's ability to act as a master on the PCI bus. 0 = Disables the device from generating PCI accesses. 1 = Enable the device to behave as a bus master.	0
Memory space	[1]	Controls a device's response to Memory Space accesses. 0 = Disable master to respond as a PCI memory target. 1 = Enable master to respond as a PCI memory target. If this bit is "0", S3C2800 Memory Space cannot be accessed by the other PCI device. S3C2800 memory space is a type that is set in the Memory Base Address Register PCIBAR0/1.	0
IO space	[0]	Controls a device's response to I/O Space accesses. 0 = Disable master to respond as a PCI I/O target. 1 = Enable master to respond as a PCI I/O target. If this bit is "0",S3C2800 IO space cannot be accessed by the other PCI device. S3C2800 IO space is a type that is set in the IO Base Address Register PCIBAR2.	0

PCI CLASS CODE & REVISION ID REGISTER (PCICRIDR)

The Class Code register (PCICRIDR[31:8]) is read-only and is used to identify the generic function of the device and, in some case, a specific register-level programming interface, The register is broken into three byte-size fields. The upper byte is a base class code. The middle byte is a sub-class code. The lower byte identifies a specific register-level programming interface.

Register	Address	R/W	Description	Reset Value
PCICRIDR	0x1008 0008	R/W	PCI class code and revision ID register	0x0D80 0001

PCICRIDR	Bit	Description	Initial State
BASCLS	[31:24]	Base class code bits These bits indicate that it broadly classifies the type of function the device performs.	0x0D
SUBCLS	[23:16]	Sub-class code bits These bits indicate that it identifies more specifically the function of device.	0x80
SRLPI	[15:8]	Specific register-level programming interface bits These bits indicate that it identifies a specific register-level programming interface so that device dependent software can interact with the device.	0x00
REVID	[7:0]	Revision ID bits These bits are used to specifies a device specific revision identifier. The value is chosen by the vendor. Zero is acceptable value.	0x01

PCI GENERAL CONTROL REGISTER (PCIGCONR)

Register	Address	R/W	Description	Reset Value
PCIGCONR	0x1008 000C	R/W	PCI general control register	0x0000 0000

PCIGCONR	Bit	Description	Initial State
SUPBIST	[31]	BIST (Built-in Self Test) capable bit (Read-only) 0 = Not BIST capable 1 = Support BIST **NOTE**: S3C2800 does not support.	0
STRBIST	[30]	BIST (Bulit-in Self Test) start bit (Read-only) Write a 1 to invoke BIST. Device resets the bit when BIST is complete. Software should fail the device if BIST is not complete after 2 seconds. **NOTE**: S3C2800 does not support..	0
–	[29:28]	Reserved	00
COMCODE	[27:24]	Completion code bits (Read-only) Device-specific failure codes can be encoded in the non-zero value. 0 = The device has passed its test. Non zero values = The device failed. **NOTE**: S3C2800 does not support.	0x0
HDTYPFUNC	[23]	Multi-function device select bit (Header Type) (Read-only) 0 = single function device 1 = multi function device	0
PREDHD	[22:16]	The layout of the 2nd part of predefined header bits. (Read-only) 0x00 = Type 00h configuration space header 0x01 = PCI-to-PCI bridges Other values = Reserved	0x00
LATTIME	[15:8]	Latency timer bits Maximum clocks that master can own the bus. These bits specifies the value of the latency timer for this PCI bus master in units of PCI bus clocks	0x00
CACHELSIZ	[7:0]	System Cache Line size bits These bits specifies the system cache size in units of DWORDs. These bits are used by master devices to determine whether to use Read, Read Line, or Read Multiple commands for accessing memory. A device may limit the number of cache line size that it supports. If an unsupported value is written to these bits, the device should behaves as if a value of 0 was written. **NOTE**: S3C2800 does not support.	0x00

PCI BASE ADDRESS REGISTERS (PCIBARn)

Register	Address	R/W	Description	Reset Value
PCIBAR0	0x1008 0010	R/W	Memory bar 0 size and location (of fast decode)	0x0000 0008
PCIBAR1	0x1008 0014	R/W	Memory bar 1 size and location (of medium decode)	0x0000 0008
PCIBAR2	0x1008 0018	R/W	I/O bar 2 size and location (of medium decode)	0x0000 0001

PCIBARn	Bit	Description	Initial State
BASEADDR	[31:4]	PCI base address bits	0x000 0000
ADPREFT	[3]	Prefetchable bit (In case of memory space) (Read-only) 0 = Non-prefetchable data 1 = Pre-fetchable data In case of I/O space (PCIBAR2), this bit is used to base address.	1 PCIBAR2 = 0
ADDRSPS	[2:1]	Base address space select bits (In case of memory space) (Read-only) 00 = Base register is 32 bits wide and can be mapped anywhere in the 32-bit memory space. 10 = Base register is 64 bits wide and can be mapped anywhere in the 64-bit memory space. (Not supported) Other values = Reserved In case of I/O space (PCIBAR2), Bit 2 is used to base address and Bit 1 is always 0.	00
ADMAPSEL	[0]	Address map select bit. (Read-only) 0 = Memory space indicator (PCIBAR0, PCIBAR1) 1 = I/O space indicator (PCIPAR2)	0 PCIBAR2 = 1

NOTE: PCI base address registers (PCIBAR0, 1, 2) must be set to non-zero value because this registers do not support PCI address 0.

PCI SUBSYSTEM & SUBSYSTEM VENDOR ID REGISTER (PCISSVIDR)

This register is used to uniquely identify the expansion board or subsystem where the PCI device resides.

Register	Address	R/W	Description	Reset Value
PCISSVIDR	0x1008 002C	R/W	PCI subsystem and subsystem vendor ID register	0x2800 144D

PCISUBSYSIDR	Bit	Description	Initial State
SUBSYSID	[31:16]	PCI subsystem ID bits. These bits are used to uniquely identify the subsystem where the PCI device resides.	0x2800
SUBSYSVENID	[15:0]	PCI subsystem vendor ID bits. These bits are used to identify the vendor of the expansion board or subsystem.	0x144D

PCI CAPABILITY POINTER REGISTER (PCICPR)

This register is used to point to a linked list of new capabilities implemented by this device. This register is only valid if the “Capabilities List” bit (PCISCR[20]) in the Status Register is set.

Register	Address	R/W	Description	Reset Value
PCICPR	0x1008 0034	R	PCI Capability pointer register	0x0000 00DC

CAPAPTR	Bit	Description	Initial State
–	[31:8]	Reserved	
CAPAPTR	[7:0]	Additional set of linked list bits (Read only) This bits are used to pointer to a linked of new capabilities implemented by this device. This bits is only valid if the “Capabilities List” bit (PCISCR[20]) in the Status Register is set.	0xDC

PCI MISCELLANEOUS REGISTER (PCIMISCR)

Register	Address	R/W	Description	Reset Value
PCIMISCR	0x1008 003C	R/W	PCI miscelaneous register	0x0000 0000

PCIMISCR	Bit	Description	Initial State
MAX_LAT	[31:24]	Maximum latency timer value bits These bits specify how often the device needs to gain access to PCI bus. The value specifies a period of time in units of 0.25µs (at 33MHz). Values of 0 indicate that the device has no major requirement for the settings of the latency timers. Values should be chosen assuming that the target does not insert any wait-states.	0x00
MIN_GNT	[23:16]	Minimum grant timer value bits These bits specify how long a burst period the device needs assuming a clock rate of 33MHz. The value specifies a period of time in units of 0.25µs. Values of 0 indicate that the device has no major requirement for the settings of the latency timers. Values should be chosen assuming that the target does not insert any wait-states.	0x00
INT_PIN	[15:8]	Interrupt pin select bits 0x00 = No use interrupt pin 0x01 = INTA# Other values = Reserved	0x00
INT_LINE	[7:0]	Interrupt line bits The value of these bits tells which input of the system interrupt controller the device’s interrupt pin is connected to. The device itself does not use this value, rather it is used by device drivers and operating systems. Device drivers and operating systems can use this information to determine priority and vector information. POST software will write the routing information into these bits as it initializes and configures the system.	0x00

PCI TARGET READY & RETRY TIMEOUT REGISTER (PCITOR)

Register	Address	R/W	Description	Reset Value
PCITOR	0x1008 0040	R/W	PCI target ready and retry timeout register	0x0000 8080

PCITOR	Bit	Description	Initial State
RETRYTOUT	[15:8]	Maximum number of master retry attempt The value specifies the number of times that the master performs retry operation.	0x80
TRDYTOUT	[7:0]	Maximum time of master wait for TRDY# The value specifies the number of PCI clocks the master waits for TRDY#.	0x80

BIF SPECIAL FUNCTION REGISTERS (BIFSFR)

These registers cannot be accessed by PCI configuration cycle, but can be accessed from AHB bus or through the base address bar from PCI bus.

Table 17-3. BIF Special Function Registers Overview

Offset	Bits	Name	R/W		Description	Reset Value
			AHB	PCI		
0x100	[15:0]	PCICON (Control)	R/W	R/W	PCI Control & Status Register (Control)	0x00B0
	[23:16]	Reserved				0x00
	[31:24]	PCICON (Status)	R/O	R/O	PCI Control & Status Register (Status)	0x00
0x104	[7:0]	PCISET (DAC)	R/W	R/W	PCI Command, Read Count & DAC Address Register (DAC)	0x00
	[15:8]	PCISET (RDC)	R/W	R/W	PCI Command, Read Count & DAC Address Register (RDC)	0x04
	[19:16]	PCISET (CMD)	R/W	R/W	PCI Command, Read Count & DAC Address Register (CMD)	0x0
0x108	[31:0]	PCIINTEN	R/W	R/W	PCI Interrupt Enable Register	0x0000 0000
0x10C	[31:0]	PCIINTST	R/WC	R/WC	PCI Interrupt Status Register	0x0000 0000
0x110	[31:0]	PCIINTAD	R/O	R/O	PCI Interrupted Address Register	0x0000 0000
0x114	[31:0]	PCIBATAPM	R/W	R/W	Base Address Translation from AHB to PCI of Memory Cycle	0x0000 0000
0x118	[31:0]	PCIBATAPI	R/W	R/W	Base Address Translation from AHB to PCI of I/O Cycle	0x0000 0000
0x128	[31:0]	PCIBELAP	R/W	R/W	PCI INTA# Assert control register in adaptor mode	0x0000 0000
0x140	[31:0]	PCIBATPA0	R/W	R/W	Base Address Translation from PCI to AHB of Memory Bar 0	0x0000 0000
0x144	[31:0]	PCIBAM0	R/W	R/W	PCI Base Address Mask Register of Memory Address Bar 0	0xFFFF 0000
0x148	[31:0]	PCIBATPA1	R/W	R/W	Base Address Translation from PCI to AHB of Memory Bar 1	0x1008 0000
0x14C	[31:0]	PCIBAM1	R/W	R/W	PCI Base Address Mask Register of Memory Address Bar 1	0xFFFF FE00
0x150	[31:0]	PCIBATPA2	R/W	R/W	Base Address Translation from PCI to AHB of I/O Bar 2	0x1008 0100
0x154	[31:0]	PCIBAM2	R/O	R/O	PCI Base Address Mask Register of I/O Address Bar 2	0xFFFF FF00

NOTE: R/O = Read-only, R/W = Read and Write, R/WC = Read and Write 1 to clear.

PCI CONTROL AND STATUS REGISTER (PCICON)

All PCI SFR including PCICON register can be accessed from AHB bus or PCI bus in byte, half-word or word size (and any byte enables from PCI bus).

Some bits are critical to PCI Controller's operation and should not be controlled by illegal software.

Register	Address	R/W	Description	Reset Value
PCICON	0x1008 0100	R/W	PCI control and status register	0x0000 0010

PCICON	Bit	Description	Initial State
INT	[31]	Internal interrupt status bit (Read-only) 0 = No interrupt 1 = Internal interrupt occured This bit indicates that one of the interrupts, enabled by the PCI interrupt enable register (PCIINTEN), has been generated. This bit is set or cleared by the PCI interrupt status and pending register (PCIINTST[10:0]). PCIINTST[31:30] does not affect this bit	0
Reserved	[30]	Reserved	0
MBS	[29]	PCI master interface busy (Read-only) 0 = Master interface idle 1 = Master interface busy Indicate that PCI master block of PCI controller is busy.	0
TBS	[28]	PCI target interface busy (Read-only) 0 = Target interface idle 1 = Target interface busy Indicate that the PCI target is busy.	0
CRS	[27]	ARM CPU reset status (Read-only) 0 = Normal operation 1 = Reset occured Indicate whether ARM CPU has been reset.	0
Reserved	[26:10]	Reserved	0x0000
RDY	[9]	System read ready 0 = Force PCI target to retry to PCI read command 1 = Enable PCI target read operation PCICON[8] bit of 0 forces PCI target to issue 'retry' for any transaction. On the other hand, PCICON[[9] bit of 0 forces PCI target to issue 'retry' only for memory (or I/O) read transaction. In general, this bit may be set with same value as PCICON[8] and can be cleared to suspend the response for PCI read transaction after PCICON[8]=1. This bit can be set or reset by ARM CPU.	0

PCICON	Bit	Description	Initial State
CFD	[8]	Configuration done 0 = Configuration registers aren't yet set and force PCI target to retry 1 = Initialization of configuration registers are done This bit is 0 at reset and PCI block does not accept any PCI transaction. PCI controller always issues 'retry' until this bit is set to 1. For normal transactions, this bit should be set to 1. This bit must be set to 1 by ARM CPU after all PCI SFR (PCI configuration registers and BIF SFR) are properly set. This bit can be set or reset by ARM CPU at any time.	0
Reserved	[7:5]	Reserved	000
ATS	[4]	Enable address translation to PCI 0 = Send address directly 1 = Enable address translation This bit controls whether or not to automatically translate the AHB address to the PCI address (In Master mode) and vice versa (In Target mode) according to the PCI command. This bit affects the configuration read/write, memory read/write and IO read/write commands ,but all other commands are not supported, so the user must manually translate the PCI address. However, must be set to 1 for dual access cycle. . S3C2800 allocated the AHB address 0x2000_0000 ~ 0x2fff_ffff for the PCI address and these allocated spaces were divided again into memory space (,0x2000_0000 ~ 0x27ff_ffff), configuration type0 space (0x2800_0000 ~ 0x29ff_ffff), configuration type1 space (0x2a00_0000 ~ 0x2bff_ffff), IO space (0x2e00_0000 ~ 0x2fff_ffff), and special space (0x2c00_0000 ~ 0x2dff_ffff). When PCICON [04]=”1”, the PCI addresses in the AHB bus are translated to corresponding spaces in the PCI bus address and loaded on the PCI bus. For example, if a read command loads 0x2300_0000 address on the AHB bus, this is recognized as an access to the PCI memory space, memory space command and address are generated in the PCI bus.	1
Reserved	[3:2]	Reserved	0
ARB	[1]	Internal PCI arbiter enable 0 = Disable arbiter (when used external arbiter) 1 = Enable PCI arbiter (When used internal arbiter)	0
HST	[0]	Host/PCI bridge mode 0 = Adaptor (agent) mode 1 = Host/PCI bridge mode	0

PCI COMMAND, READ COUNT & DAC ADDRESS REGISTER (PCISET)

Register	Address	R/W	Description	Reset Value
PCISET	0x1008 0104	R/W	PCI command, Read count and DAC address register	0x0000 0000

PCISET	Bit	Description	Initial State
CMD	[19:17]	PCI command when PCICON[4] = 0. If automatic address generation is disabled (PCICON[4]=0), PCI address must be generated manually. Bit 0 of the 4-bit PCI command indicates read (0)/write (1), so this register sets the command with the upper 3 bits (bit 3~bit1) and program processes the read/ write. The command supports the IO, memory and configuration read/write normally in both master and target modes. DAC (Dual Address Cycle) adds 8 bits in master mode to allow 40-bit address but is not supported in target mode. **NOTE:** S3C2800 PCI module does not support cache. Therefore cache-related commands are not supported.	0x0
Reserved	[15:8]	Reserved	0x00
DAC	[7:0]	PCI DAC upper 8 bit address PCI DAC (Dual Address Cycle) upper 8 bit [39:32] address. This bits should be all zero when normal SAC (Single Address Cycle). DAC function operates only when accessing the memory space. The configuration space and IO space are not affected.	0x00

PCI INTERRUPT ENABLE REGISTER (PCIINTEN)

This PCIINTEN register controls masking of the internal ARM CPU interrupt. If each mask bit is 1, interrupt is enabled, otherwise interrupt is disabled.

Register	Address	R/W	Description	Reset Value
PCIINTEN	0x1008 0108	R/W	PCI interrupt enable register	0x0000 0000

PCIINTEN	Bit	Description	Initial State
BAP	[31]	PCI INTA# assert operation enable in adaptor mode (PCICON[0]=0) 0=Disable interrupt 1=Enable interrupt	0
Reserved	[30:11]	Reserved	
INA	[10]	PCI INTA# interrupt enable in host/PCI bridge mode 0=Disable interrupt 1=Enable interrupt Interrupt enable of the PCI INTA# is asserted in host/PCI bridge mode.	0
SER	[9]	PCI SERR# interrupt enable in host/PCI bridge mode 0=Disable interrupt 1=Enable interrupt Interrupt enable of the PCI SERR# is asserted in host/PCI bridge mode.	0
Reserved	[8:5]	Reserved	0
TPE	[4]	PCI target parity error interrupt enable 0=Disable interrupt 1=Enable interrupt Internal (AHB) interrupt enable of the PCI target parity error detected.	0
MPE	[3]	PCI master parity error interrupt enable 0=Disable interrupt 1=Enable interrupt Internal (AHB) interrupt enable of the PCI master parity error detected.	0
MFE	[2]	PCI master fatal error interrupt enable 0=Disable interrupt 1=Enable interrupt Internal (AHB) interrupt enable of the PCI master fatal error (master abort / target abort) detected.	0
PRA	[1]	PCI reset asserted interrupt enable 0=Disable interrupt 1=Enable interrupt Internal (AHB) interrupt enable of PCI reset asserted	0
PRD	[0]	PCI reset de-asserted interrupt enable 0=Disable interrupt 1=Enable interrupt Internal (AHB) interrupt enable of PCI reset de-asserted	0

PCI INTERRUPT STATUS & PENDING REGISTER (PCIINTST)

Each PCIINTST register bit except [31:30] is masked by each PCIINTEN bit. If this masked value is not 0, internal interrupt signal can be generated and transferred to interrupt controller. Otherwise, no interrupt signal is generated. Each interrupt status (pending) bit is cleared by explicitly writing 1.

Register	Address	R/W	Description	Reset Value
PCIINTST	0x1008 010C	R/WC	PCI interrupt statue and pending register	0x0000 0000

PCIINTST	Bit	Description	Initial State
WDE	[31]	PCI master fatal error is occurred on write operation (Read-only) 0 = PCI master fatal error is not occurred 1 = PCI master fatal error is occurred	0
RDE	[30]	PCI master fatal error is occurred on read operation (Read-only) 0 = PCI master fatal error is not occurred 1 = PCI master fatal error is occurred	0
Reserved	[29:11]	Reserved	
INA	[10]	PCI INTA# is asserted in host/PCI bridge mode. (Read or write-1-to-clear) 0 = it's not active 1 = it is activated (if read), clear (if write)	0
SER	[9]	PCI SERR# is asserted in host/PCI bridge mode (Read or write-1-to-clear) 0 = it's not active 1 = it is activated (if read), clear (if write)	0
Reserved	[8:5]	Reserved	
TPE	[4]	PCI target detects parity error (Read or write-1-to-clear) 0 = it's not active 1 = it is activated (if read), clear (if write)	0
MPE	[3]	PCI master detects parity error (Read or write-1-to-clear) 0 = it's not active 1 = it is activated (if read), clear (if write)	0
MFE	[2]	PCI master detects fatal error (master abort / target abort) (Read or write-1-to-clear) 0 = it's not active 1 = it is activated (if read), clear (if write)	0
PRA	[1]	PCI reset is asserted (Read or write-1-to-clear) 0 = it's not active 1 = it is activated (if read), clear (if write)	0
PRD	[0]	PCI reset is deasserted (Read or write-1-to-clear) 0 = it's not active 1 = it is activated (if read), clear (if write)	0

NOTE: To clear the PCI interrupt pending bit, the corresponding bit in the PCIINTST must be cleared first, and then the corresponding bit in the SRCPND register must be cleared.

PCI INTERRUPTED ADDRESS REGISTER (PCIINTAD)

Register	Address	R/W	Description	Reset Value
PCIINTAD	0X1008 0110	R	PCI interrupted address register	0x0000 0000

PCIINTAD	Bit	Description	Initial State
ADR	[31:0]	Interrupted address when PCIINTST [2] or PCIINTST [3] is asserted. (Read-only) This address register is set to AHB address requested when PCI master detects master abort, target abort or parity error. And PCIINTST [30], PCIINTST [31] is also set.	0x0000 0000

PCI BASE ADDRESS TRANSLATION REGISTER FROM AHB TO PCI OF MEMORY CYCLE (PCIBATAPM)

Register	Address	R/W	Description	Reset Value
PCIBATAPM	0x1008 0114	R/W	PCI base address translation register from AHB to PCI of memory cycle.	0x0000 0000

PCIBATAPM	Bit	Description	Initial State
APM	[31:27]	PCI base address when memory space is accessed from AHB bus. AHB address of 00100b (MSB of 0x2000 0000) is replaced with these 5 bits for PCI memory address when PCICON[4] = 1.	00000b
Reserved	[26:0]	Reserved	0

PCI BASE ADDRESS TRANSLATION REGISTER FROM AHB TO PCI OF I/O CYCLE (PCIBATAPI)

Register	Address	R/W	Description	Reset Value
PCIBATAPI	0x1008 0118	R/W	PCI base address translation register from AHB to PCI of I/O cycle.	0x0000 0000

PCIBATAPI	Bit	Description	Initial State
API	[31:25]	PCI base address when I/O space is accessed from AHB bus. AHB address of 0010111b (MSB of 0x2E00 0000) is replaced with these 7 bits for PCI I/O address when PCICON [4] = 1.	0x00
Reserved	[24:0]	Reserved	0

PCI INTA# ASSERT CONTROL IN ADAPTOR MODE (PCIBELAP)

Register	Address	R/W	Description	Reset Value
PCIBELAP	0x1008 0128	R/W	PCI INT# Assert control register in adaptor mode	0x0000 0000

PCIBELAP	Bit	Description	Initial State
LAP	[31]	PCI INTA# assert in adaptor mode. 0 = INTA# NOT asserted 1 = INTA# asserted	0
Reserved	[30:0]	Reserved	0

PCI BASE ADDRESS TRANSLATION FROM PCI TO AHB OF MEMORY ADDRESS BAR 0 (PCIBATPA0)

This register can be changed after PCICON [8] = 1.

Register	Address	R/W	Description	Reset Value
PCIBATPA0	0x1008 0140	R/W	PCI base address translation from PCI to AHB of memory address bar 0.	0x0000 0000

PCIBATPA0	Bit	Description	Initial State
PA0	[31:0]	AHB base address when memory address bar 0 is hit. Only upper n bits defined in PCIBAM0 is used.	0x0000 0000

PCI BASE ADDRESS MASK REGISTER OF MEMORY ADDRESS BAR 0 (PCIBAM0)

When software writes all 1s to memory base address bar 0 in PCI configuration registers, the value read back is this PCIBAM0 value. PCI base address mask registers including this PCIBAM0 are used to define the base address decoding range.

This register must be kept to same value after PCICON [8] = 1.

Register	Address	R/W	Description	Reset Value
PCIBAM0	0x1008 0144	R/W	PCI base address mask register of memory address bar 0.	0xFFFF 0000

PCIBAM0	Bit	Description	Initial State
AM0	[31:0]	Base address mask of memory address bar 0. This bits are used for indicating the size of the address region. This mask register must be set before other PCI device access the configuration register and cannot be changed after PCICON [8] = 1. And this register value must be valid number (111...111000...000b), otherwise the results are unkown.	0xFFFF 0000

PCI BASE ADDRESS TRANSLATION FROM PCI TO AHB OF MEMORY ADDRESS BAR 1 (PCIBATPA1)

Register	Address	R/W	Description	Reset Value
PCIBATPA1	0x1008 0148	R/W	PCI base address translation register from PCI to AHB bus of memory address bar 1.	0x1008 0000

PCIBATPA1	Bit	Description	Initial State
PA1	[31:0]	AHB base address when memory address bar 1 is hit. Only upper n bits defined in PCIBAM1 is used.	0x1008 0000

PCI BASE ADDRESS MASK REGISTER OF MEMORY ADDRESS BAR 1 (PCIBAM1)

When software writes all 1s to memory base address bar 1 in PCI configuration registers, the value read back is this PCIBAM1 value.

Register	Address	R/W	Description	Reset Value
PCIBAM1	0x1008 014C	R/W	PCI base address mask register of memory address bar 1.	0xFFFF FE00

PCIBAM1	Bit	Description	Initial State
AM1	[31:0]	Base address mask of memory address bar 1. This bits are used for indicating the size of the address region. This mask register must be set before other PCI device access the configuration register and can not be changed after PCICON [8] = 1. And this register value must be valid number (111...111000...000b), otherwise it leads to unknown operation.	0xFFFF FE00

PCI BASE ADDRESS TRANSLATION FROM PCI TO AHB OF I/O ADDRESS BAR 2 (PCIBATPA2)

Register	Address	R/W	Description	Reset Value
PCIBATPA2	0x1008 0150	R/W	PCI base address translation register from PCI to AHB bus of I/O address bar 2.	0x1008 0100

PCIBATPA2	Bit	Description	Initial State
PA2	[31:8]	AHB base address when I/O address bar 2 is hit. Upper 24 bits are used for translation. All 24 bits are used.	0x10 0801
Reserved	[7:0]	Reserved (Always 0).	0x00

PCI BASE ADDRESS MASK REGISTER OF I/O ADDRESS BAR 2 (PCIBAM2)

When software writes all 1s to I/O base address bar 2 in PCI & CardBus configuration registers, the value read back is this PCIBAM2 value (FFFF_FF00h) indicating that 256 bytes of I/O address space are required.

Register	Address	R/W	Description	Reset Value
PCIBAM2	0x1008 0154	R	PCI base address mask register of I/O address bar 2.	0xFFFF FF00

PCIBAM2	Bit	Description	Initial State
AM2	[31:0]	Base address mask of I/O address bar 2. (Read-only) This bits are used for indicating the size of the address region. This mask register is read-only and hardwired to 0xFFF_FF00.	0xFFFF FF00

NOTES

18 ELECTRICAL DATA

ABSOLUTE MAXIMUM RATINGS

Table 18-1. Absolute Maximum Rating

Symbol	Parameter	Rating		Unit
V_{DD}	1.8V Core DC Supply Voltage	2.4		V
V_{DDP}	3.3V I/O DC Supply Voltage	3.8		V
V_{IN}	DC Input Voltage	3.3 V Input buffer	3.8	V
V_{OUT}	DC Output Voltage	3.3 V Output buffer	3.8	V
I_{latch}	Latch-up Current	± 200		mA
T_{STG}	Storage Temperature	– 65 to 150		^{o}C

RECOMMENDED OPERATING CONDITIONS

Table 18-2. Recommended Operating Conditions

Symbol	Parameters	Condition	Min	Type	Max	Unit
V_{DD}	1.8V Core DC Supply Voltage	Commercial	1.7	1.8	1.95	V
V_{DDP}	3.3V I/O DC Supply Voltage	Commercial	3.0	3.3	3.6	V
V_{IN}	DC Input Voltage	3.3V Input buffer	3.0	3.3	3.6	V
V_{OUT}	DC Output Voltage	3.3V Output buffer	3.0	3.3	3.6	V
T_{OPR}	Operating Temperature	Commercial	0		70	^{o}C
I_{DD}	Normal operating current (FCLK : HCLK : PCLK = 1: 1/2 : 1/4)					mA
	1.8V core supply current	FCLK = 200MHz, V_{DD} = 1.95V	–	210	300	
	3.3V I/O supply current	FCLK = 200MHz, V_{DDP} = 3.6V	–	75	110	
I_{DD1}	Idle mode current (FCLK : HCLK : PCLK = 1: 1/2 : 1/4)					mA
	1.8V core supply current	FCLK = 200MHz, V_{DD} = 1.95V	–	75	110	
	3.3V I/O supply current	FCLK = 200MHz, V_{DDP} = 3.6V	–	15	30	
I_{DD2}	Slow mode current (FCLK : HCLK : PCLK = 1: 1/2 : 1/2)					mA
	1.8V core supply current	FCLK = 6MHz, V_{DD} = 1.95V	–	15	30	
	3.3V I/O supply current	FCLK = 6MHz, V_{DDP} = 3.6V	–	5	10	

D.C. ELECTRICAL CHARACTERISTICS

Table 18-3. Normal I/O PAD D.C. Electrical Characteristics

(V_{DD} = 1.8 V -0.1 V/+0.15 V, V_{DDP} = 3.3 V ± 0.3 V, T_{OPR} = 0 to 70 °C)

Symbol	Parameters	Condition	Min	Type	Max	Unit
V_{IH}	High level input voltage					V
	LVCMOS interface		2.0			
V_{IL}	Low level input voltage					V
	LVCMOS interface				0.8	
VT	Switching threshold			1.4		V
VT+	Schmitt trigger, positive-going threshold	CMOS			2.0	V
VT-	Schmitt trigger, negative-going threshold	CMOS	0.8			
I_{IH}	High level input current					µA
	Input buffer	$V_{IN} = V_{DDP}$	-10		10	
I_{IL}	Low level input current					µA
	Input buffer	$V_{IN} = V_{SS}$	-10		10	
	Input buffer with pull-up		-120	-66	-20	
V_{OH}	High level output voltage					V
	Type B4	I_{OH} = -4 mA	2.4			
	Type B8	I_{OH} = -8 mA	2.4			
	Type B12	I_{OH} = -12 mA	2.4			
V_{OL}	Low level output voltage					V
	Type B4	I_{OL} = 4 mA			0.4	
	Type B8	I_{OL} = 8 mA			0.4	
	Type B12	I_{OL} = 12 mA			0.4	
C_{IN}	Input capacitance	Any Input and Bi-directional Buffers			4	pF
C_{OUT}	Output capacitance	Any Output Buffers			4	pF

Table 18-4. PCI I/O PAD D.C. Electrical Characteristics

(V_{DD} = 1.8 V -0.1 V/+0.15 V, V_{DDP} = 3.3 V ± 0.3 V, T_{OPR} = 0 to 70 °C)

Symbol	Parameters	Condition	Min	Type	Max	Unit
V_{IH}	High level input voltage		0.47VDDP		VDDP+0.5	V
V_{IL}	Low level input voltage		-0.5		0.33VDDP	V
I_I	Input Leakage Current		-10		10	μA
V_{OH}	High level output voltage	I_{OH} = -500 μA	0.9V_{DDP}			V
V_{OL}	Low level output voltage	I_{OL} = 1500 μA			0.1V_{DDP}	V

A.C. ELECTRICAL CHARACTERISTICS

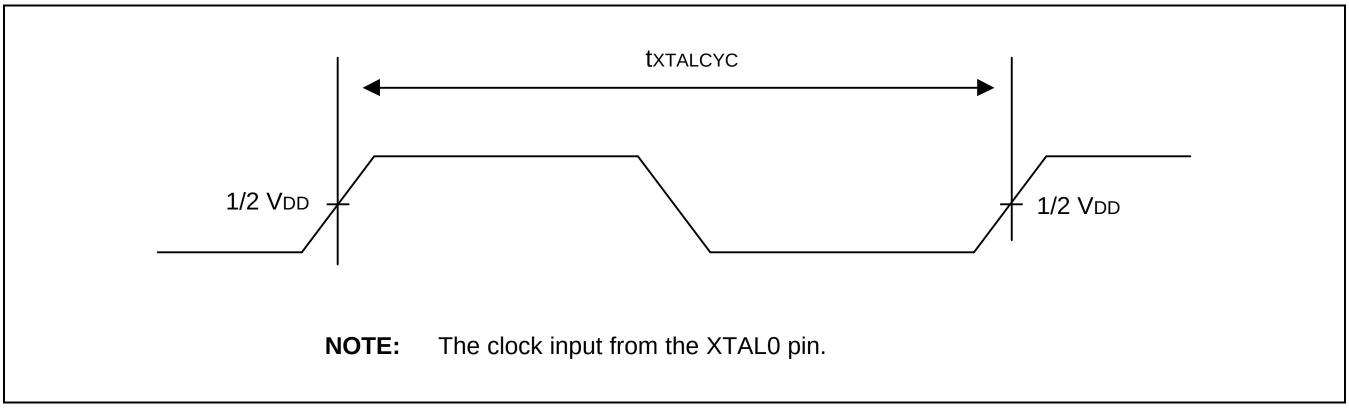


Figure 18-1. XTAL0 Clock Timing

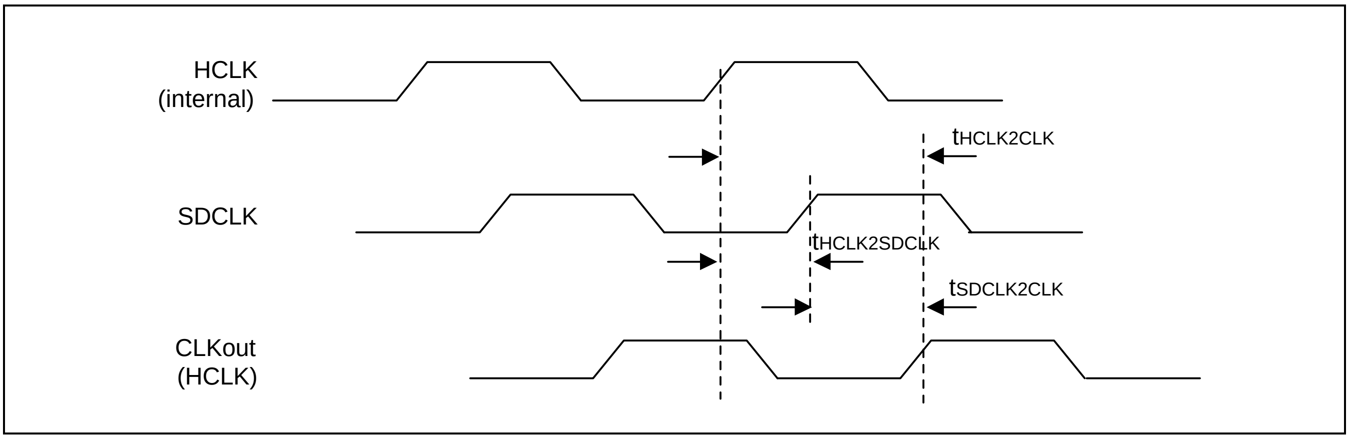


Figure 18-2. AHBCLK/CLKout/SDCLK Timing

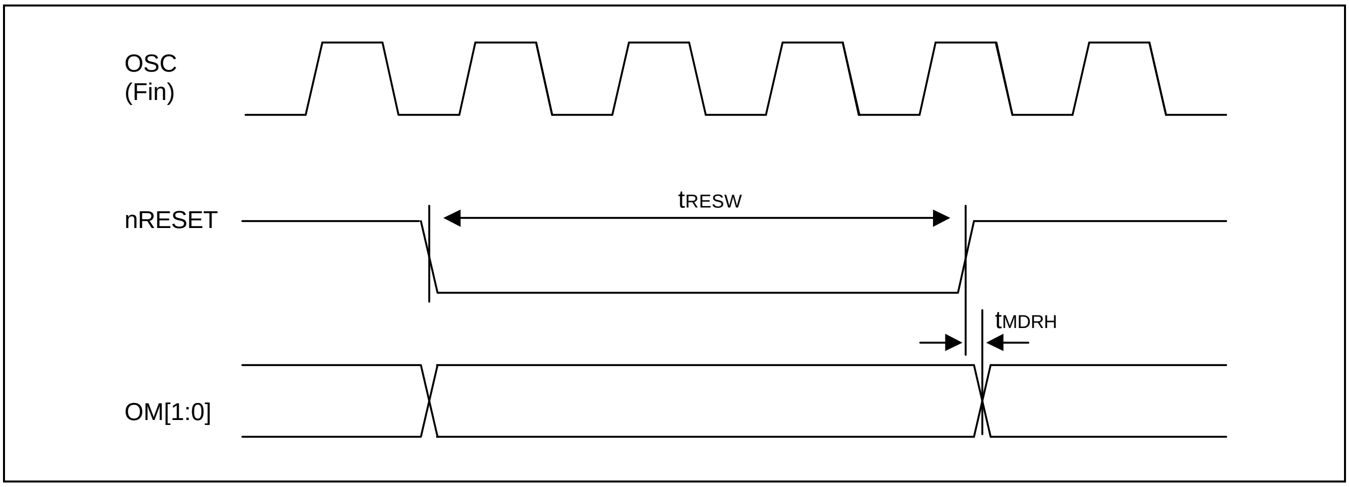


Figure 18-3. Manual Reset and OM[1:0] Input Timing

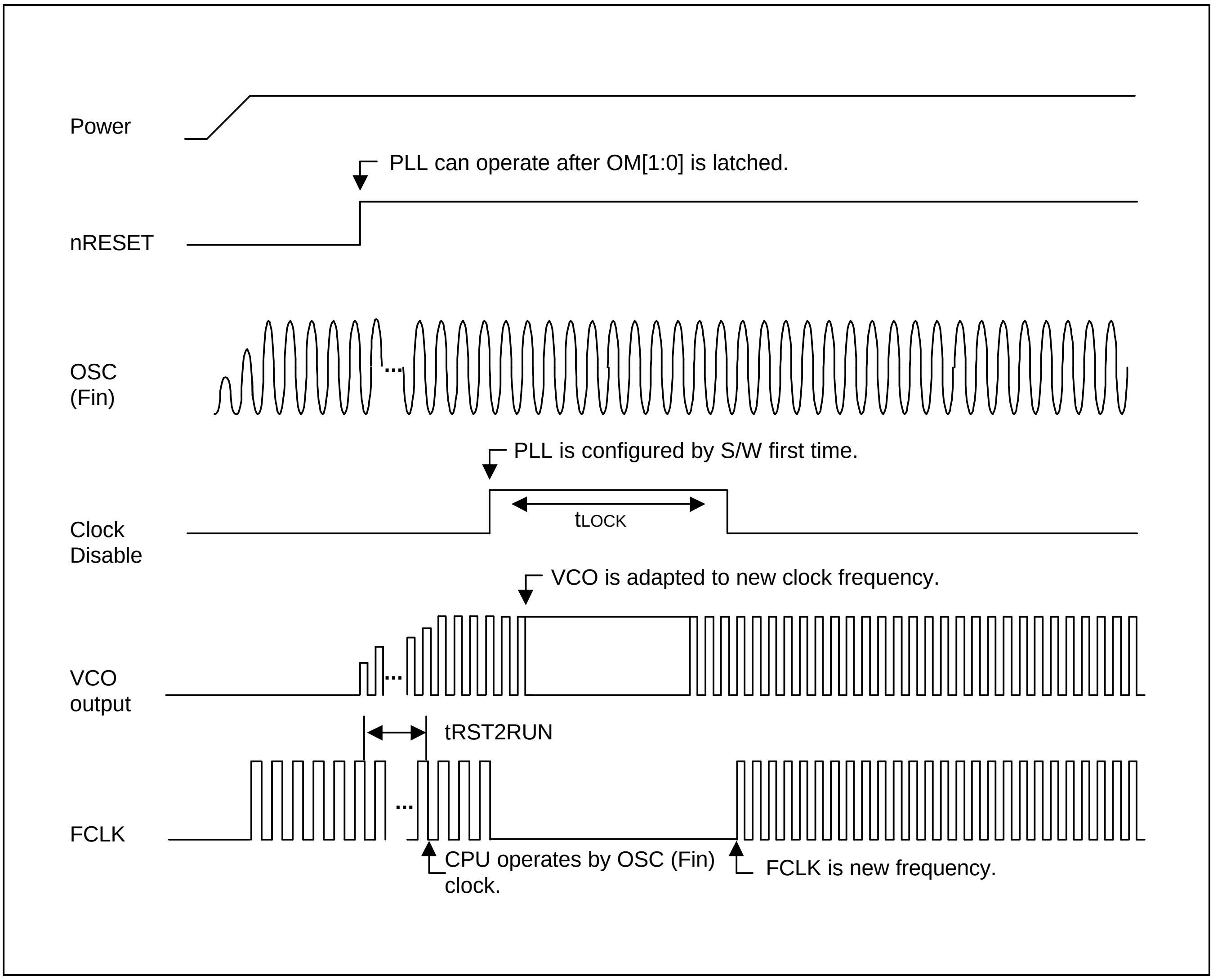

Power
PLL can operate after OM[1:0] is latched.
nRESET
OSC
(Fin)
PLL is configured by S/W first time.
tLOCK
Clock
Disable
VCO is adapted to new clock frequency.
VCO
output
tRST2RUN
FCLK
CPU operates by OSC (Fin)
clock.
FCLK is new frequency.

Figure 18-4. Power-On Oscillation Setting Timing

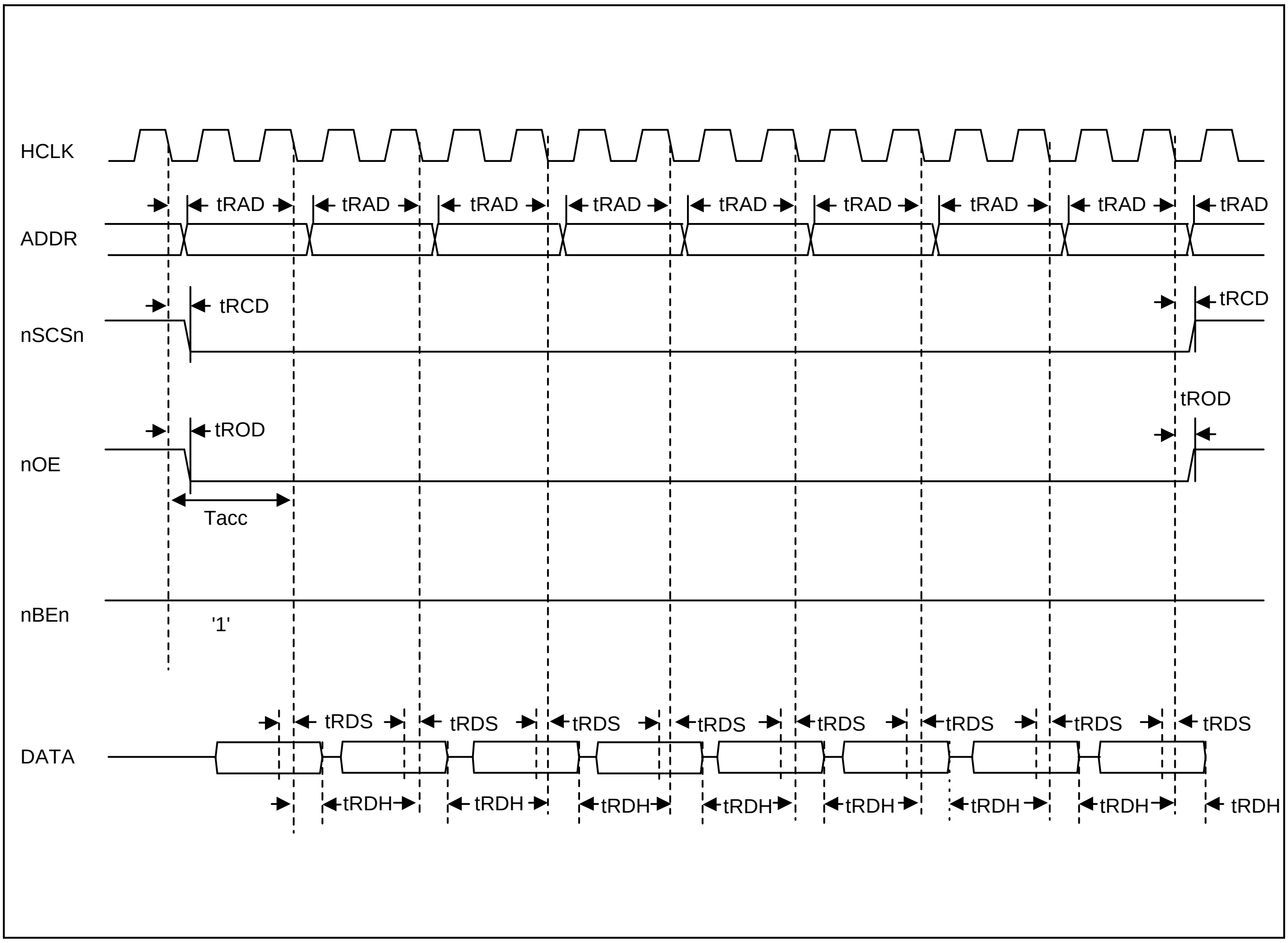


Figure 18-5. ROM/SRAM Burst READ Timing(I)
(Tacs=0, Tcos=0, Tacc=2, Toch=0, Tcah=0, ST=0, SDW=16bit)

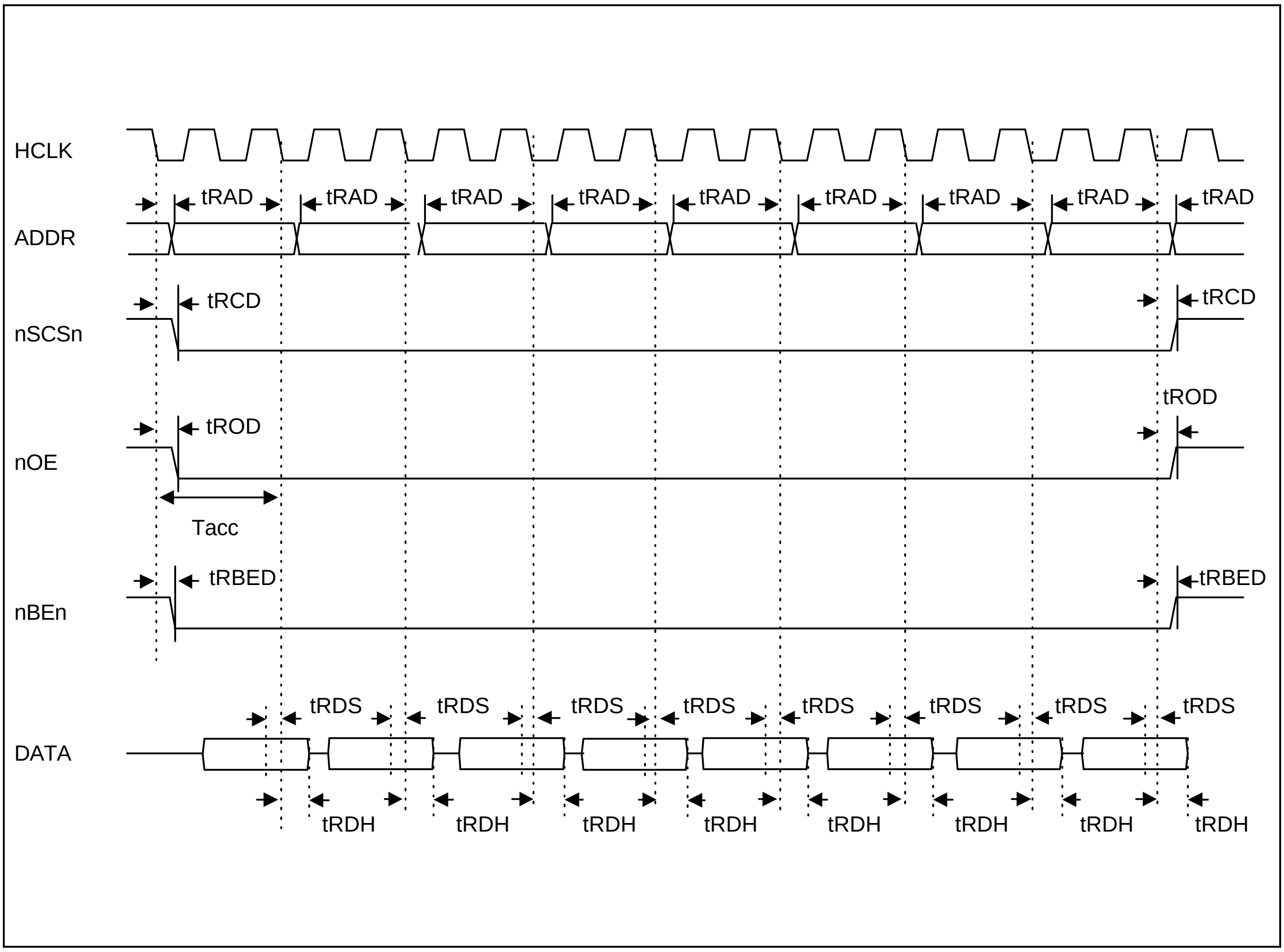


Figure 18-6. ROM/SRAM Burst READ Timing(II)
(Tacs=0, Tcos=0, Tacc=2, Toch=0, Tcah=0, ST=1, SDW=16bit)

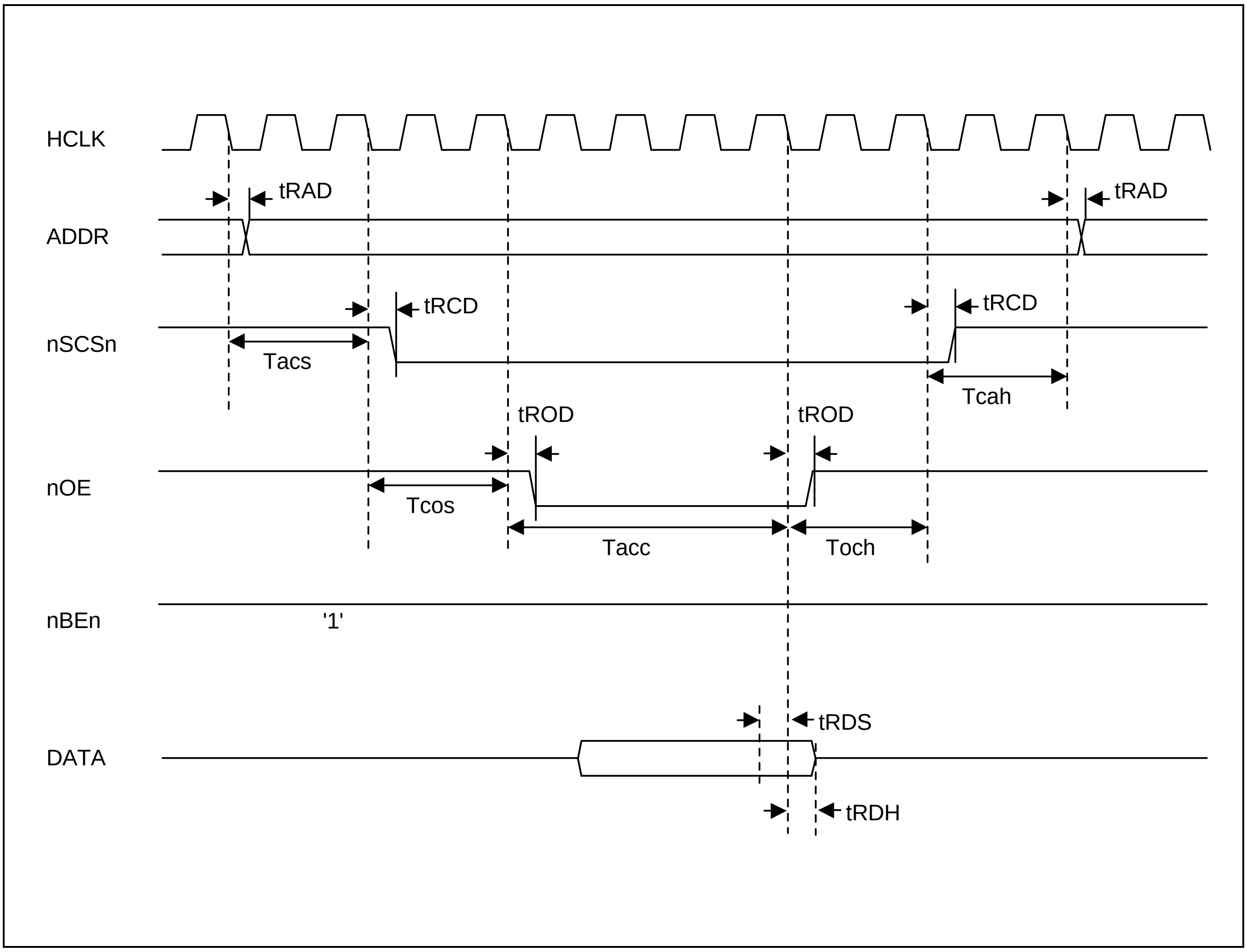


Figure 18-7. ROM/SRAM READ Timing (I)
(Tacs=2,Tcos=2, Tacc=4, Toch=2, Tcah=2,ST=0)

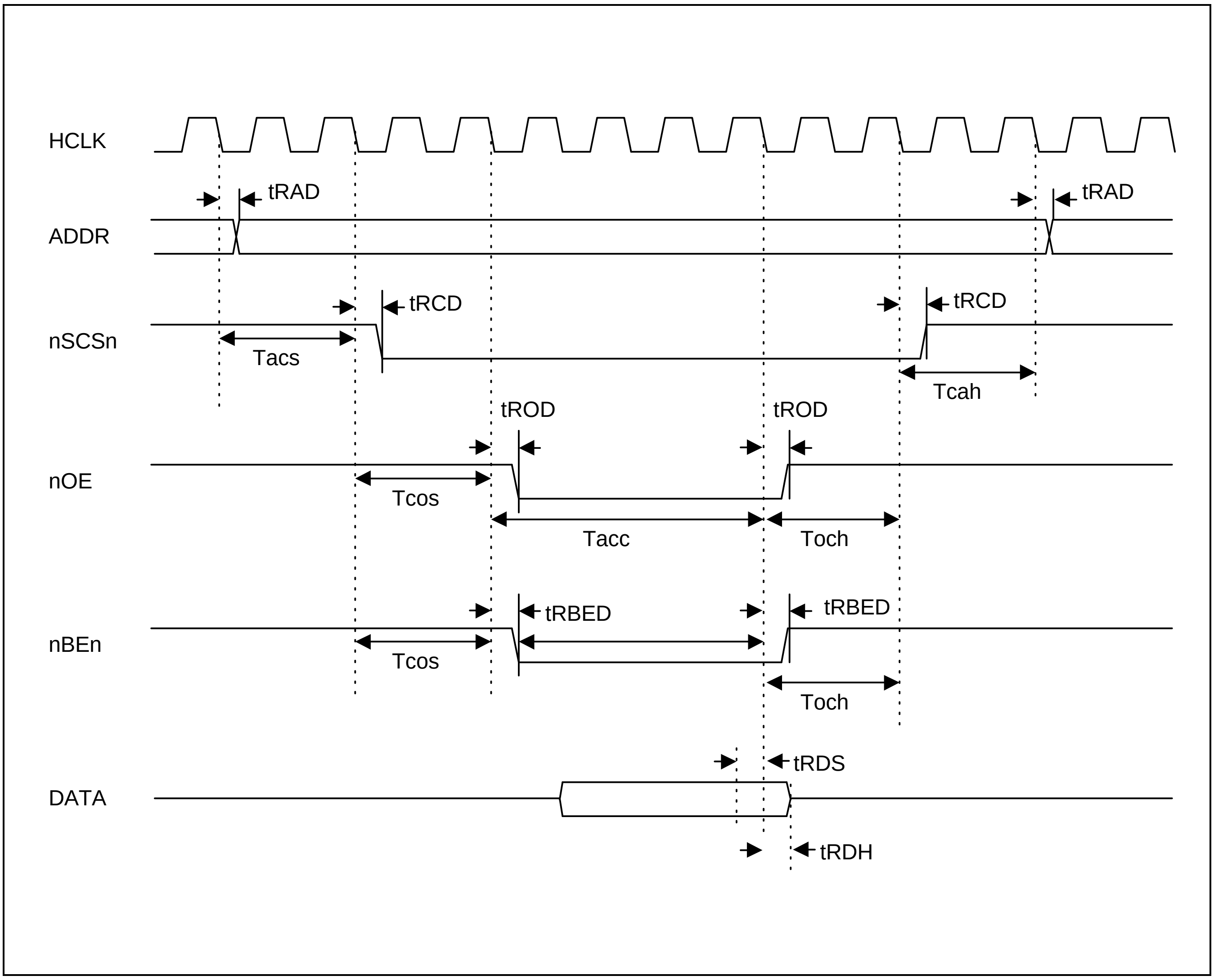


Figure 18-8. ROM/SRAM READ Timing (II)
(Tacs=2, Tcos=2, Tacc=4, Toch=2, Tcah=2, ST=1)

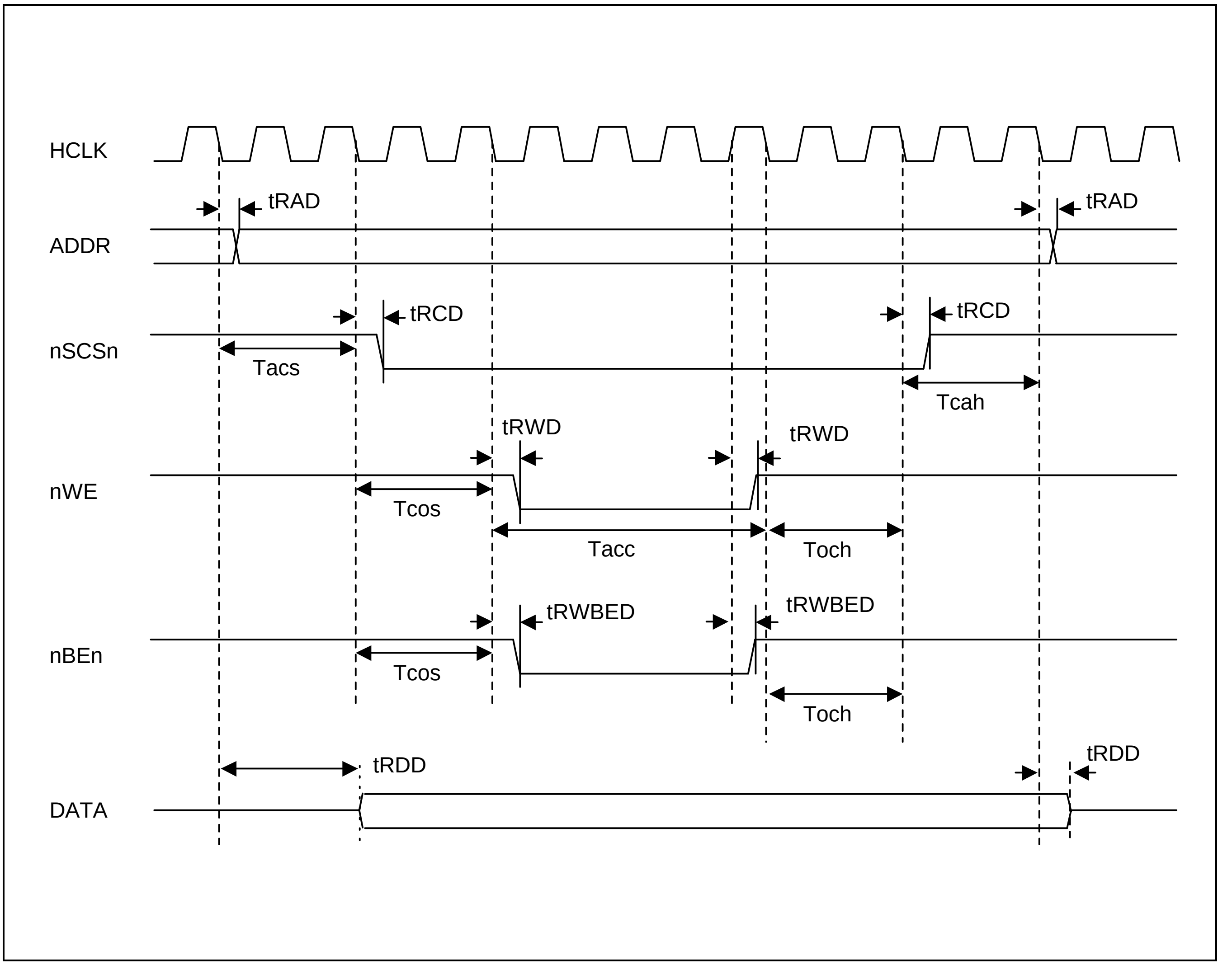


Figure 18-9. ROM/SRAM WRITE Timing (I)
(Tacs=2,Tcos=2,Tacc=4,Toch=2, Tcah=2, ST=0)

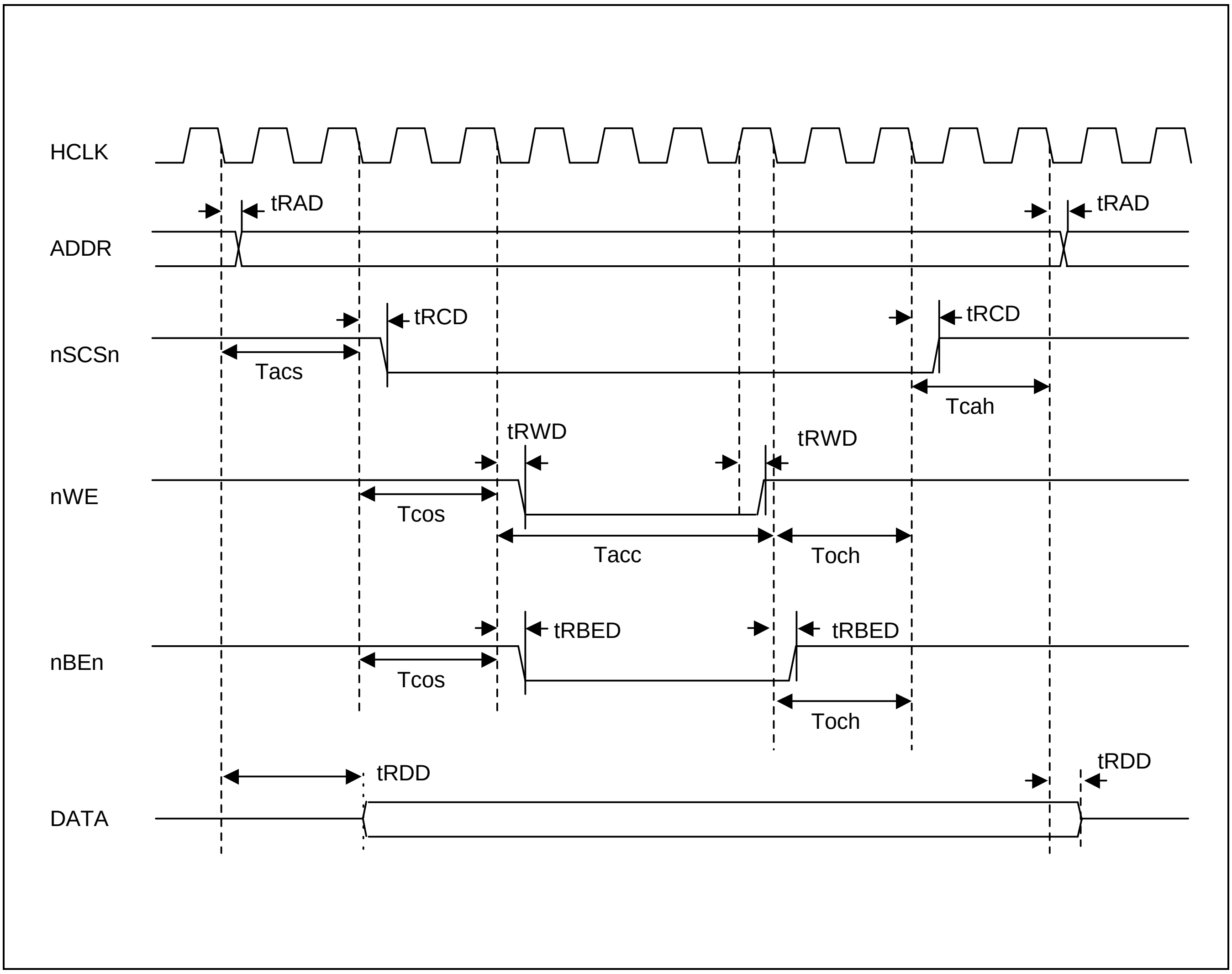


Figure 18-10. ROM/SRAM WRITE Timing (II)
(Tacs=2, Tcos=2, Tacc=4, Toch=2, Tcah=2, ST=1)

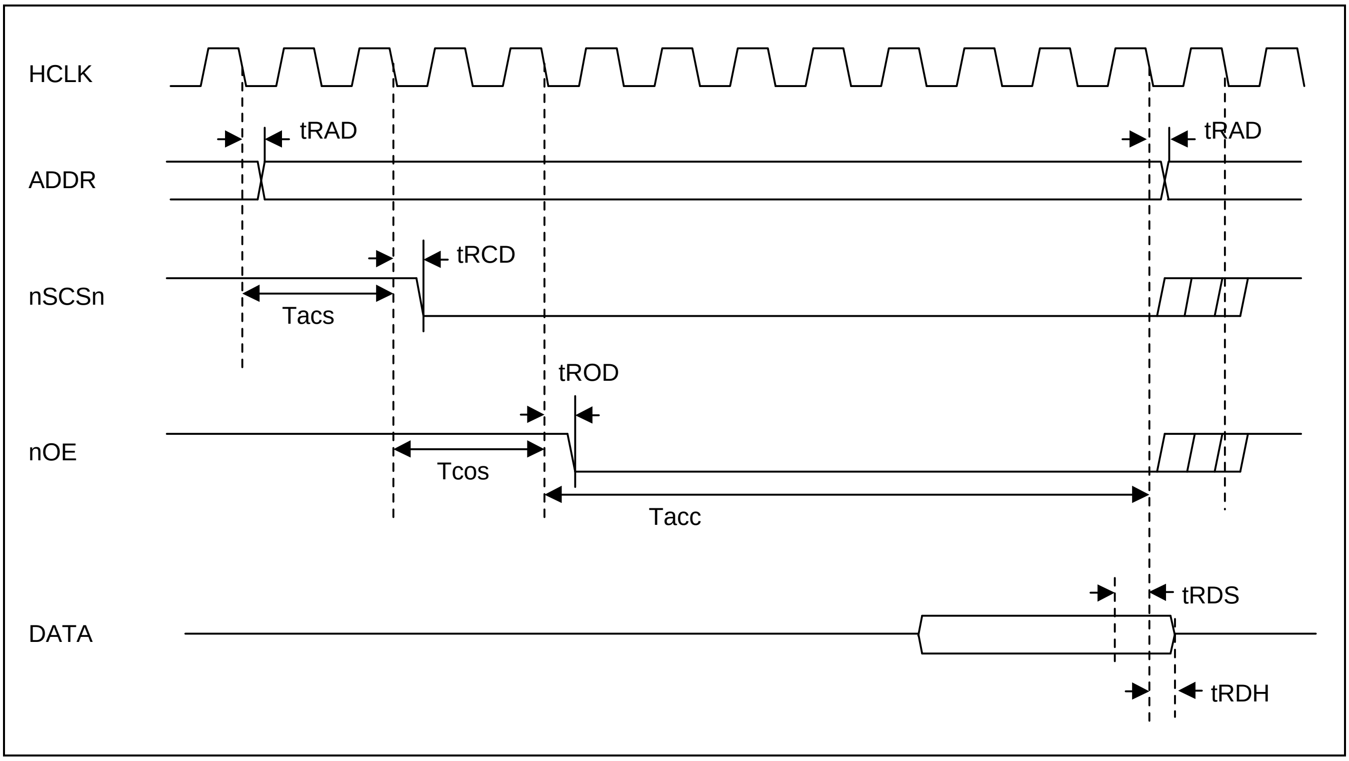
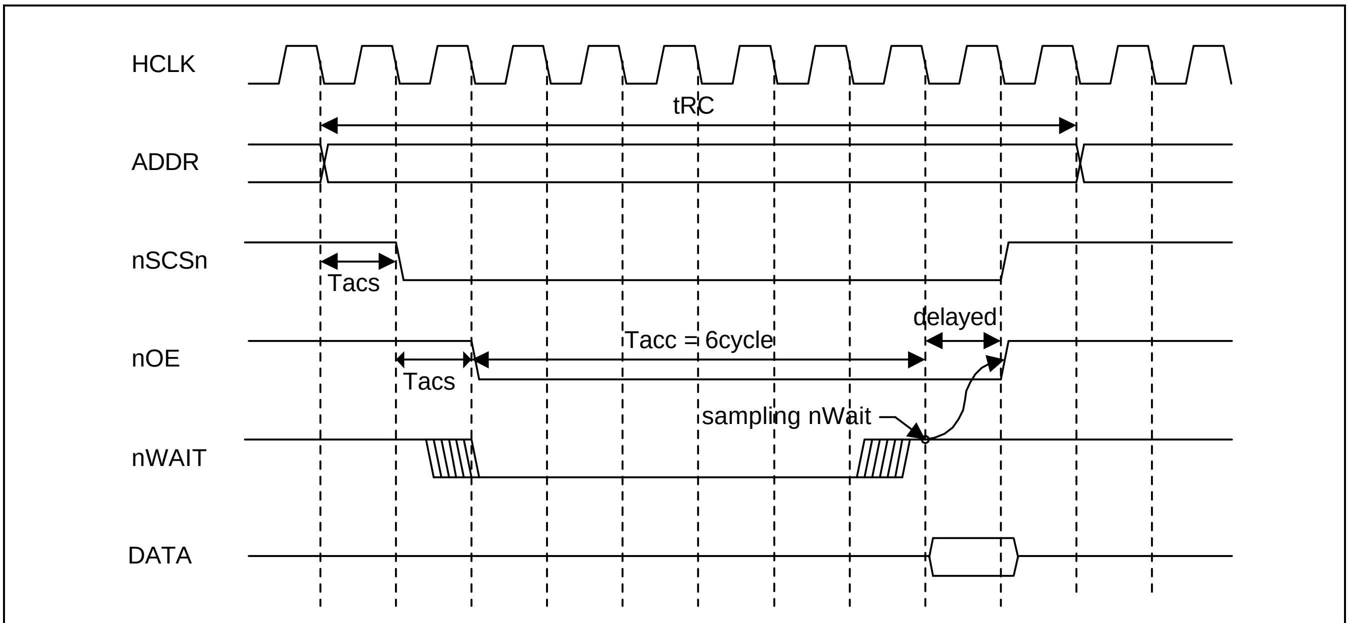


Figure 18-11. Masked-ROM Single READ Timing (Tacs=2, Tcos=2, Tacc=8)



NOTE : The status of nWait is checked at (Tacc-1) cycle.

Figure 18-12. External nWAIT READ Timing
(Tacs=0, Tcos=0, Tacc=6, Toch=0, Tcah=0, ST=0)

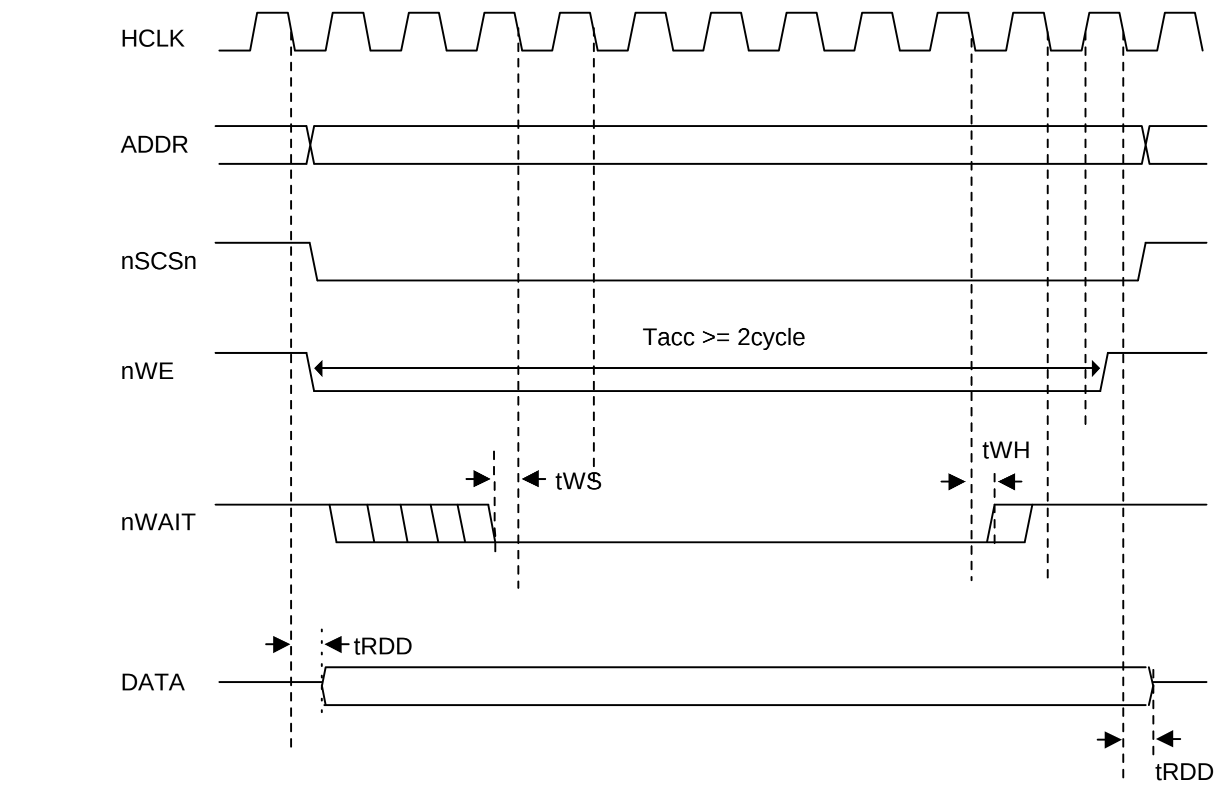


Figure 18-13. External nWAIT WRITE Timing
(Tacs=0, Tcos=0, Tacc=4, Toch=0, Tcah=0, ST=0)

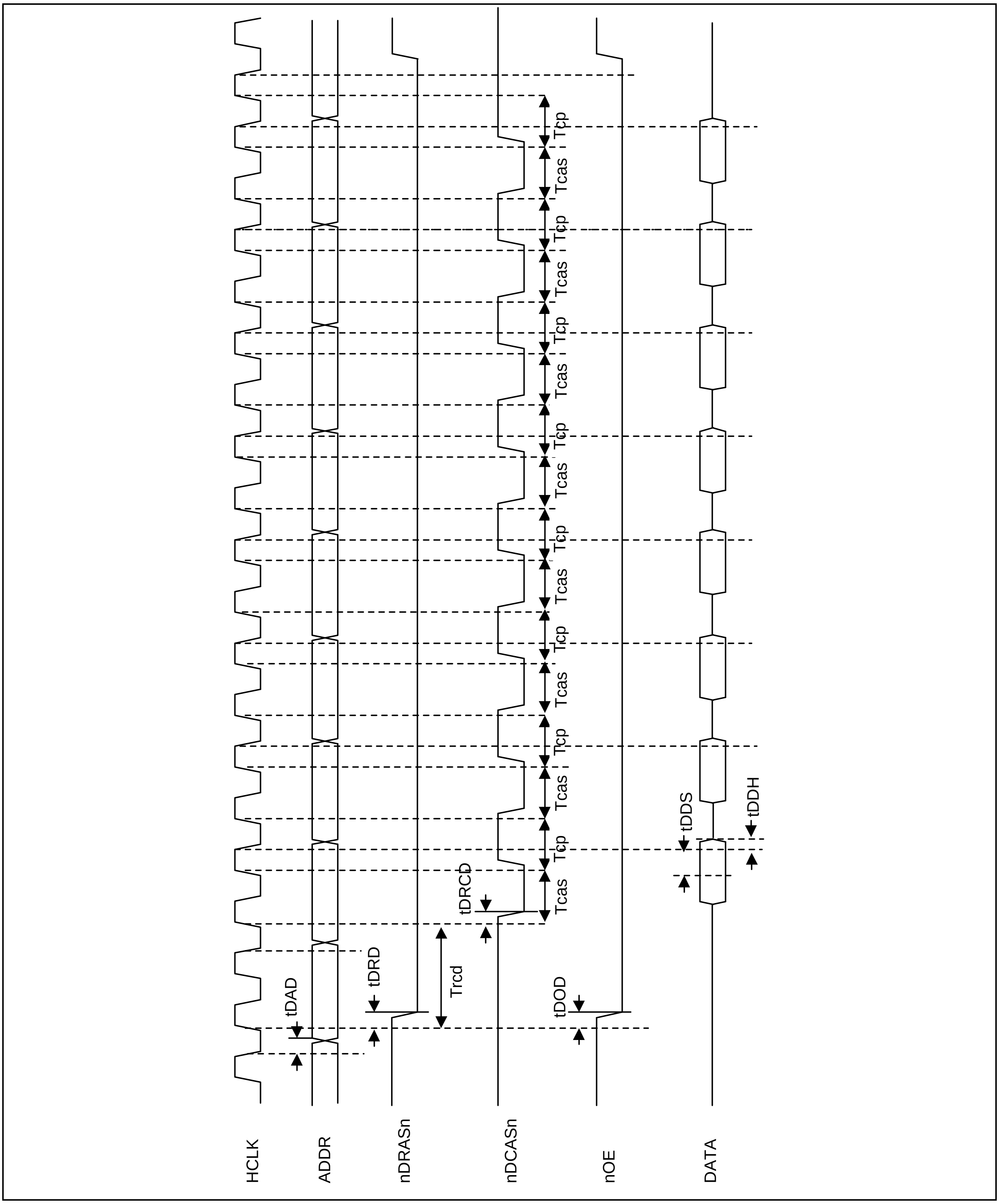


Figure 18-14. DRAM (EDO) Burst READ Timing (Trcd=2, Tcas=1, Tcp=1, Trp=3.5, MT=10, DW = 16bit)

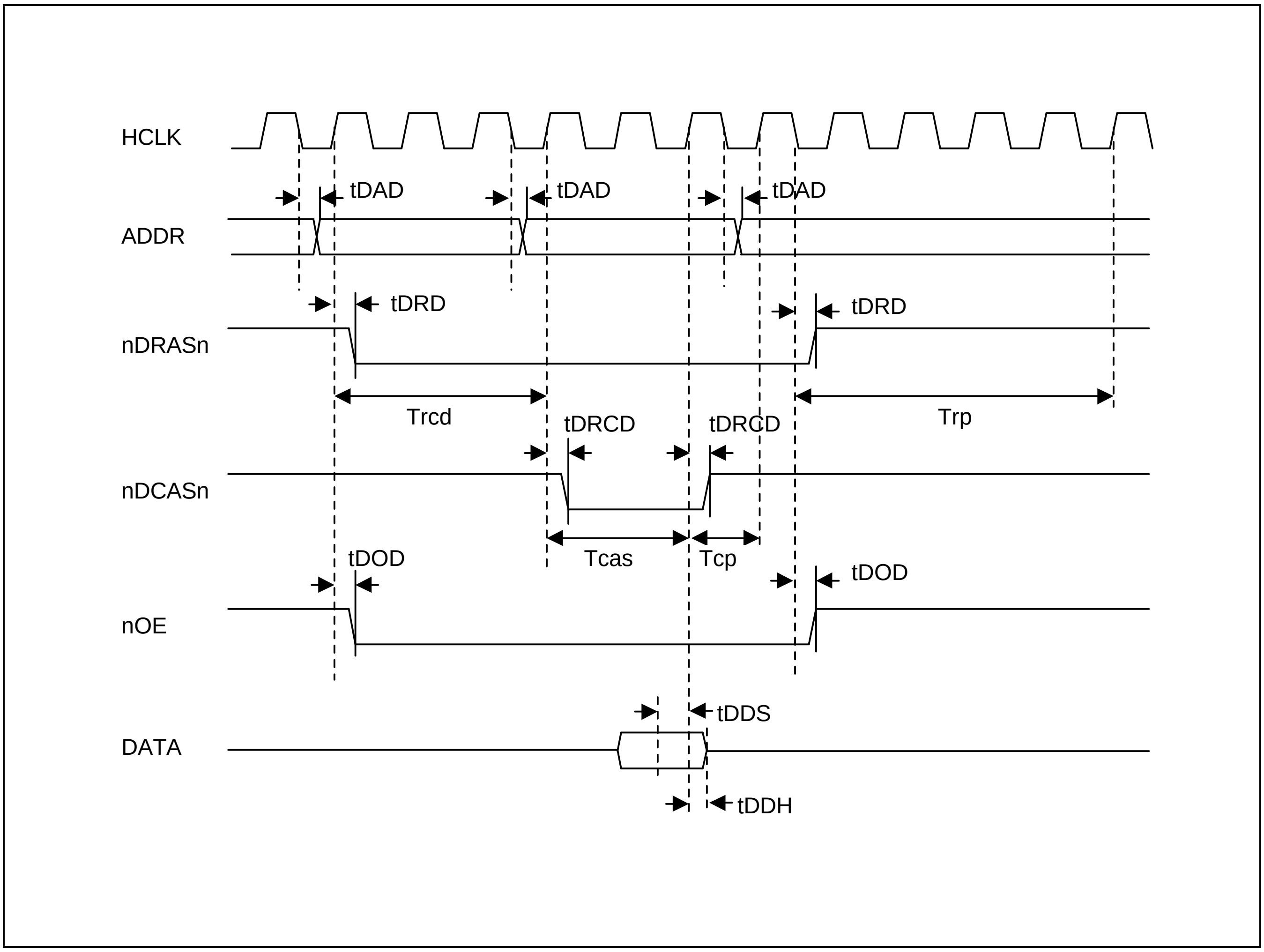


Figure 18-15. DRAM(FP) Single READ Timing (Trcd=3, Tcas=2, Tcp=1, Trp=4.5, MT=01)

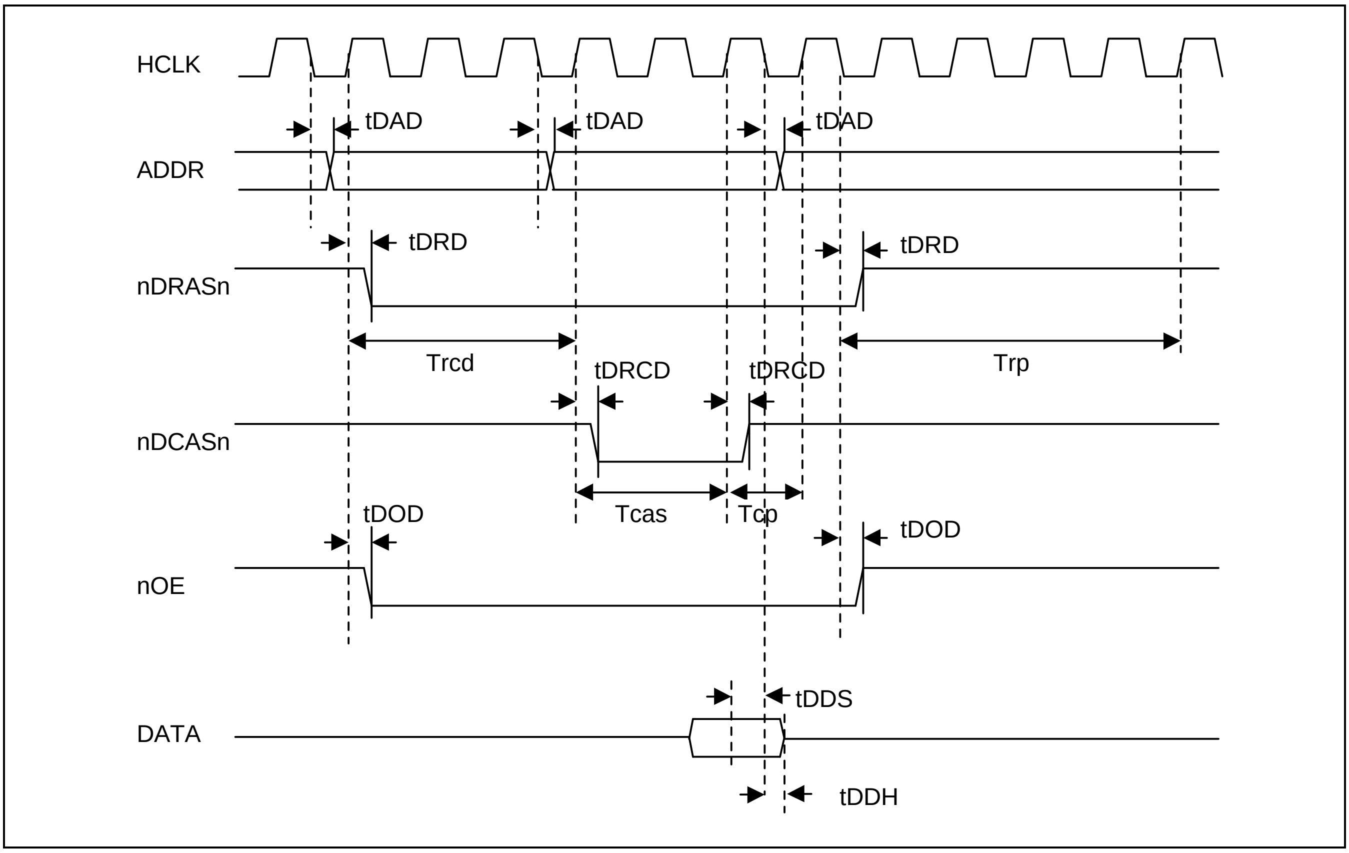


Figure 18-16. DRAM(EDO) Single READ Timing (Trcd=3, Tcas=2, Tcp=1, Trp=4.5, MT=10)

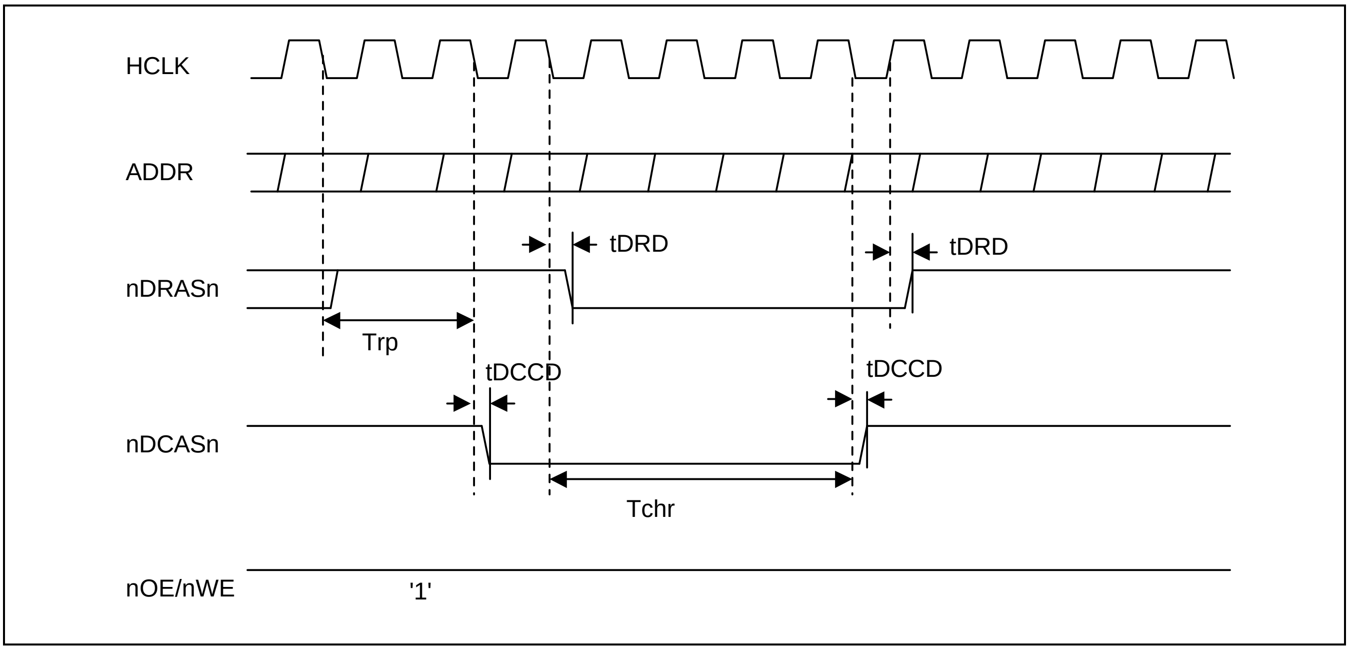


Figure 18-17. DRAM CBR Refresh Timing (Tchr=4)

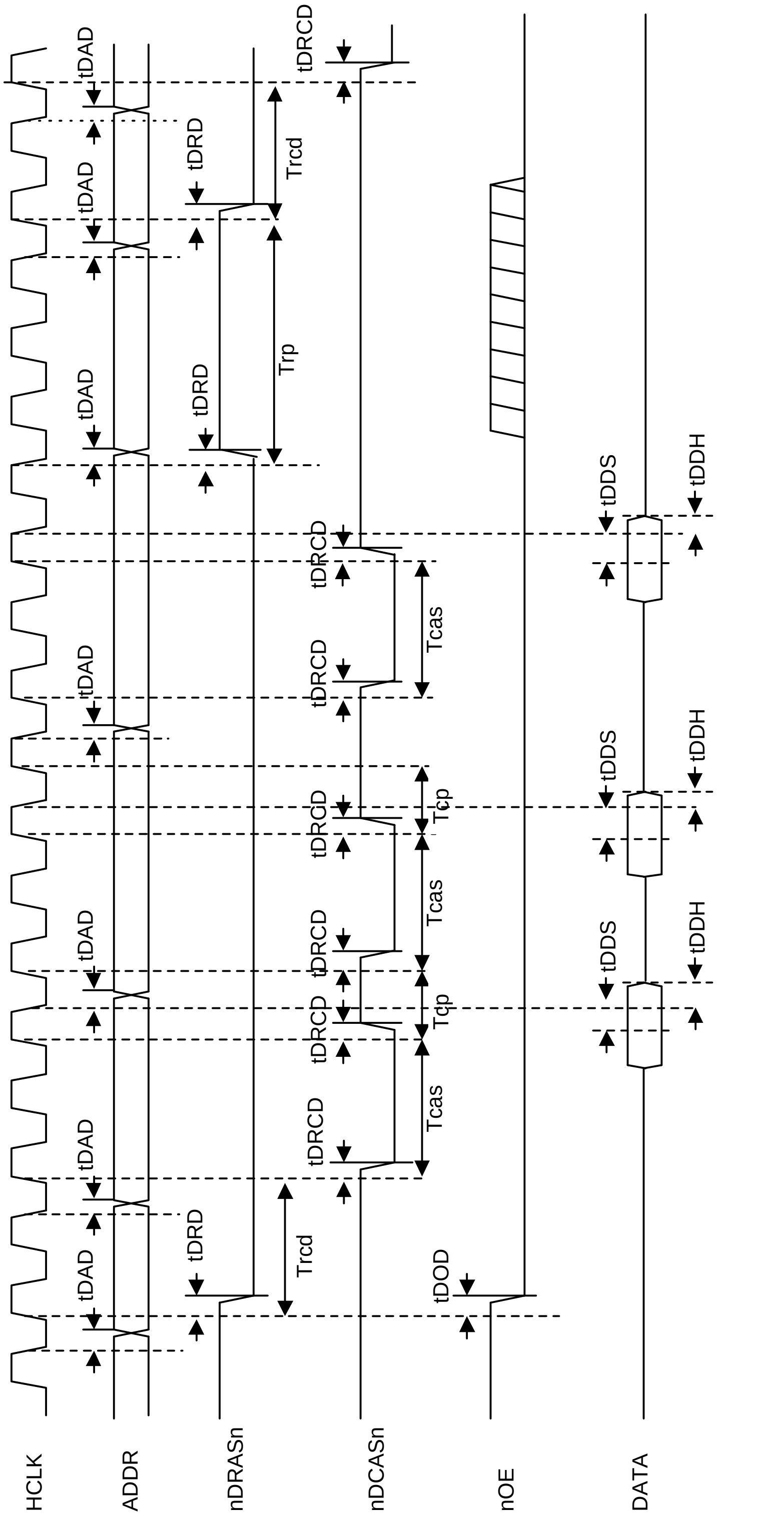


Figure 18-18. DRAM(EDO) Page Hit-Miss READ Timing (Trcd=2, Tcas=2, Tcp=1, Trp=3.5, MT=10)

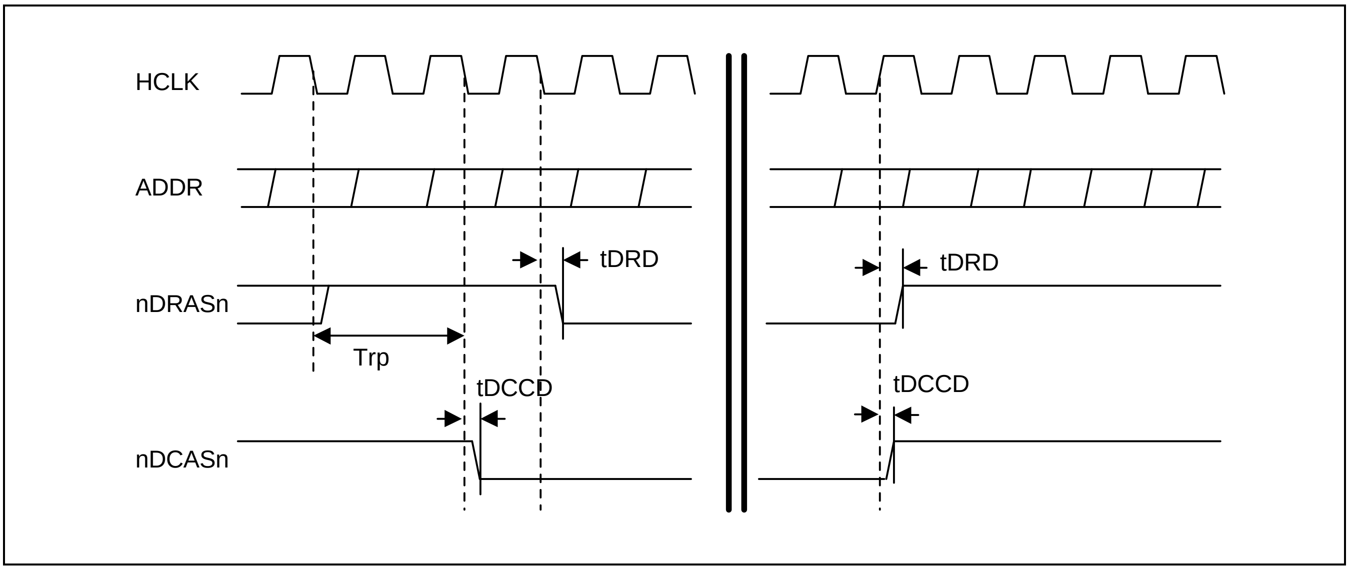


Figure 18-19. DRAM Self Refresh Timing

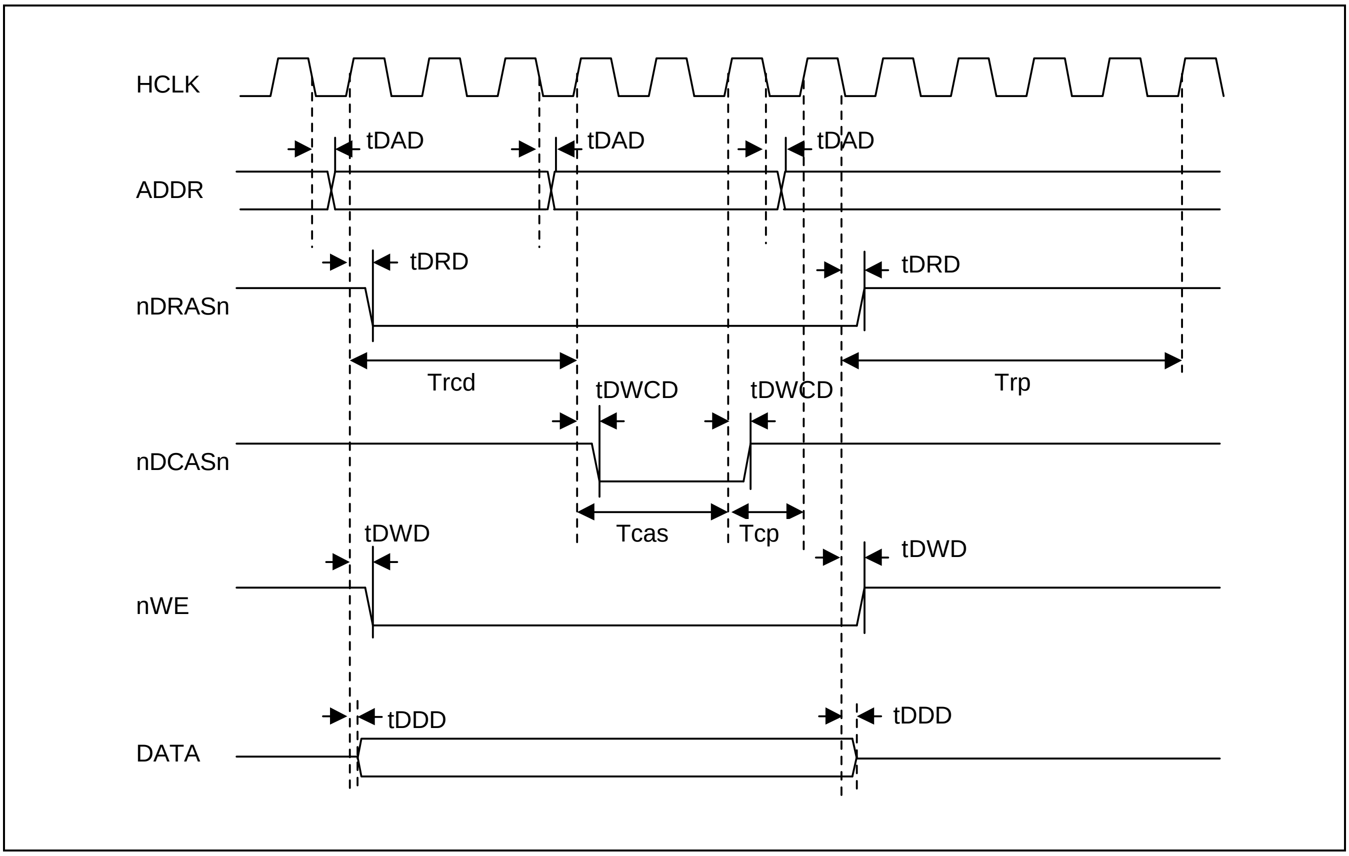


Figure 18-20. DRAM(FP/EDO) Single Write Timing
(Trcd=3, Tcas=2, Tcp=1, Trp=4.5, MT=01/10)

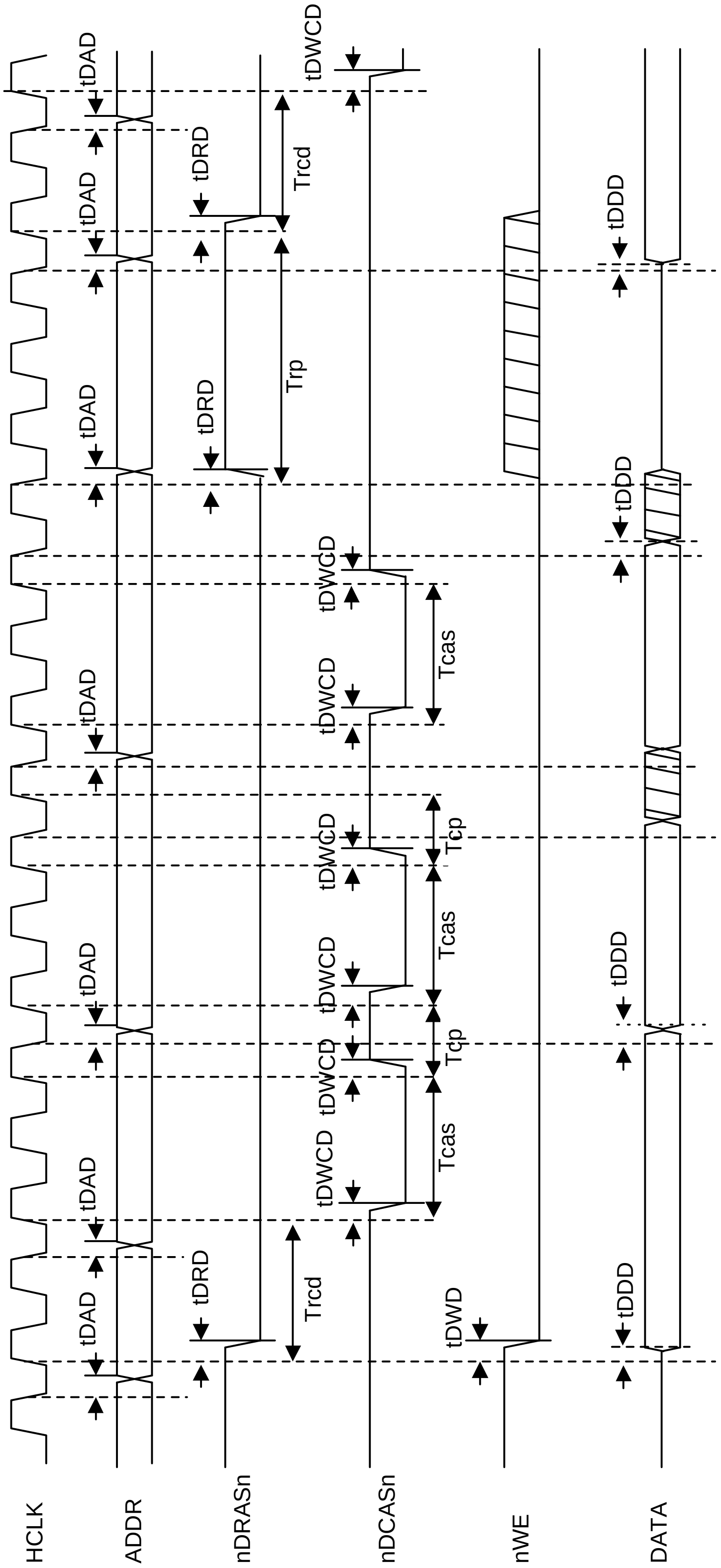


Figure 18-21. DRAM(FP/EDO) Page Hit-Miss Write Timing (Trcd=2, Tcas=2, Tcp=1, Trp=3.5, MT=01/10)

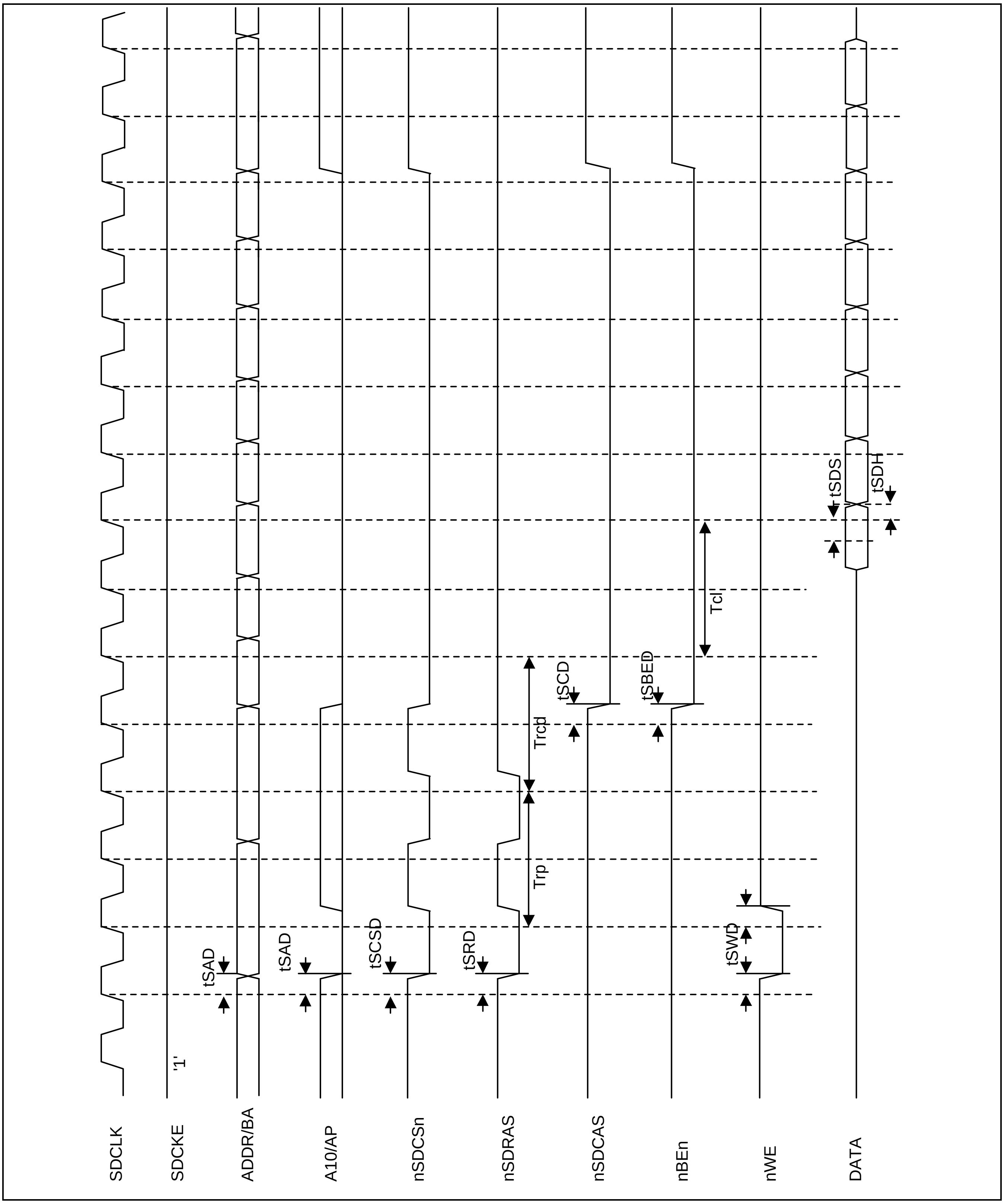


Figure 18-22. SDRAM Single Burst READ Timing (Trp=2, Trcd=2, Tcl=2, DW=16bit)

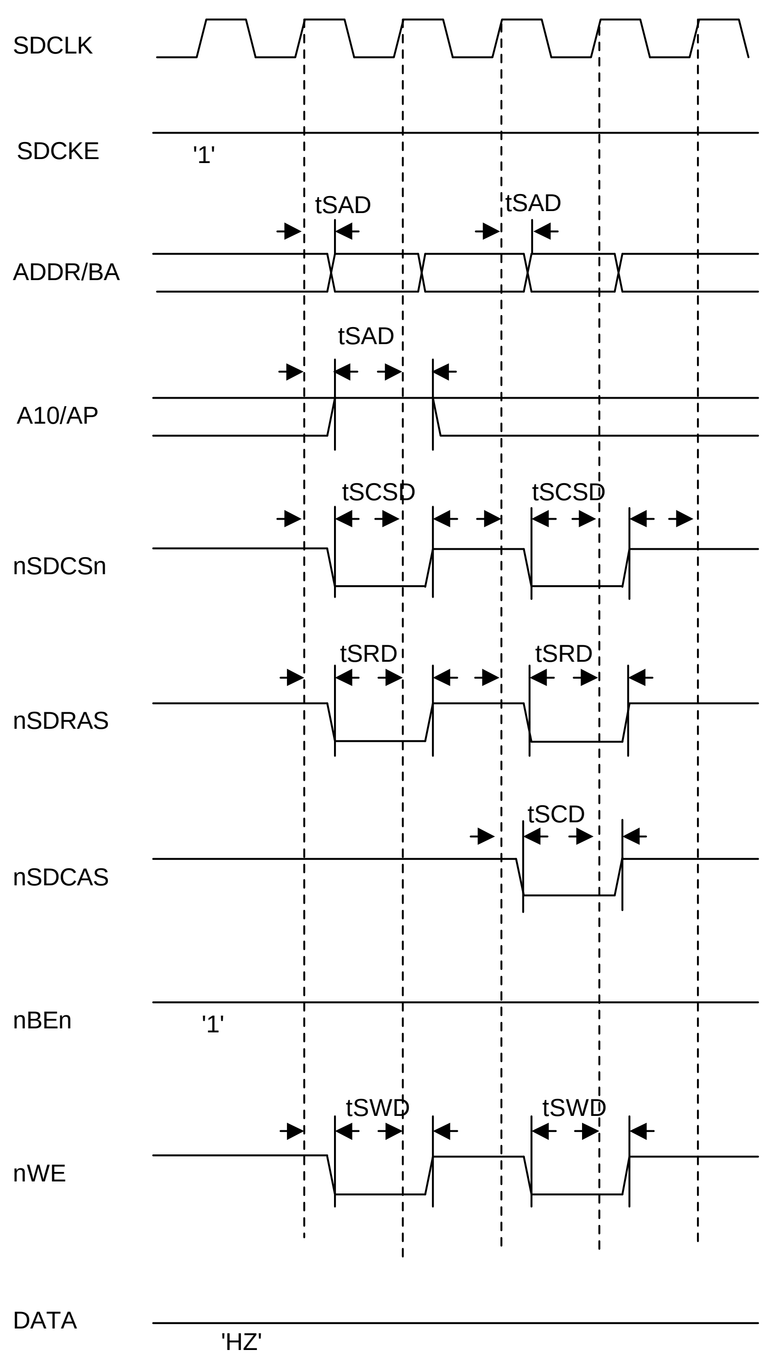


Figure 18-23. SDRAM MRS Timing

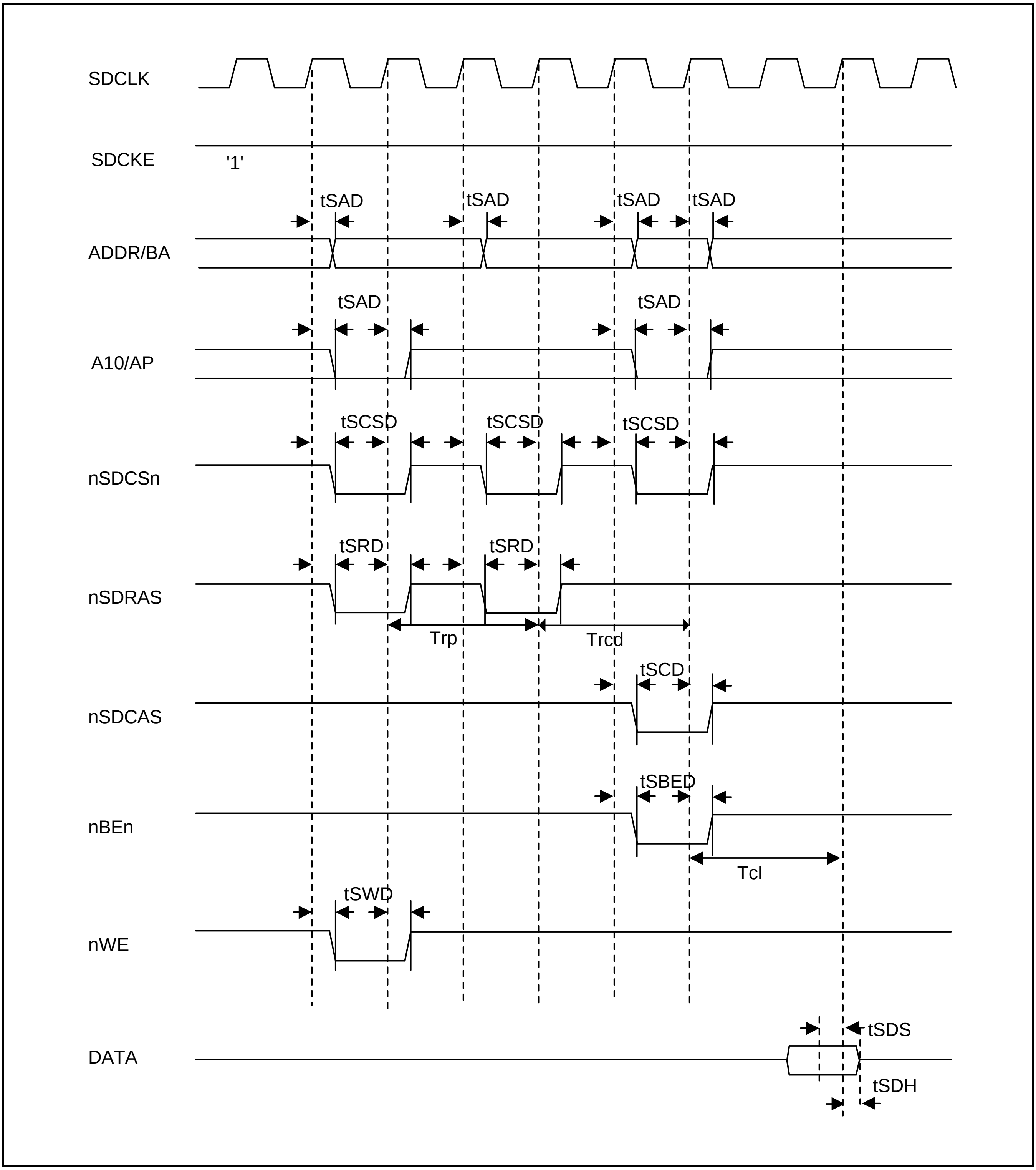


Figure 18-24. SDRAM Single READ Timing(I) (Trp=2, Trcd=2, Tcl=2)

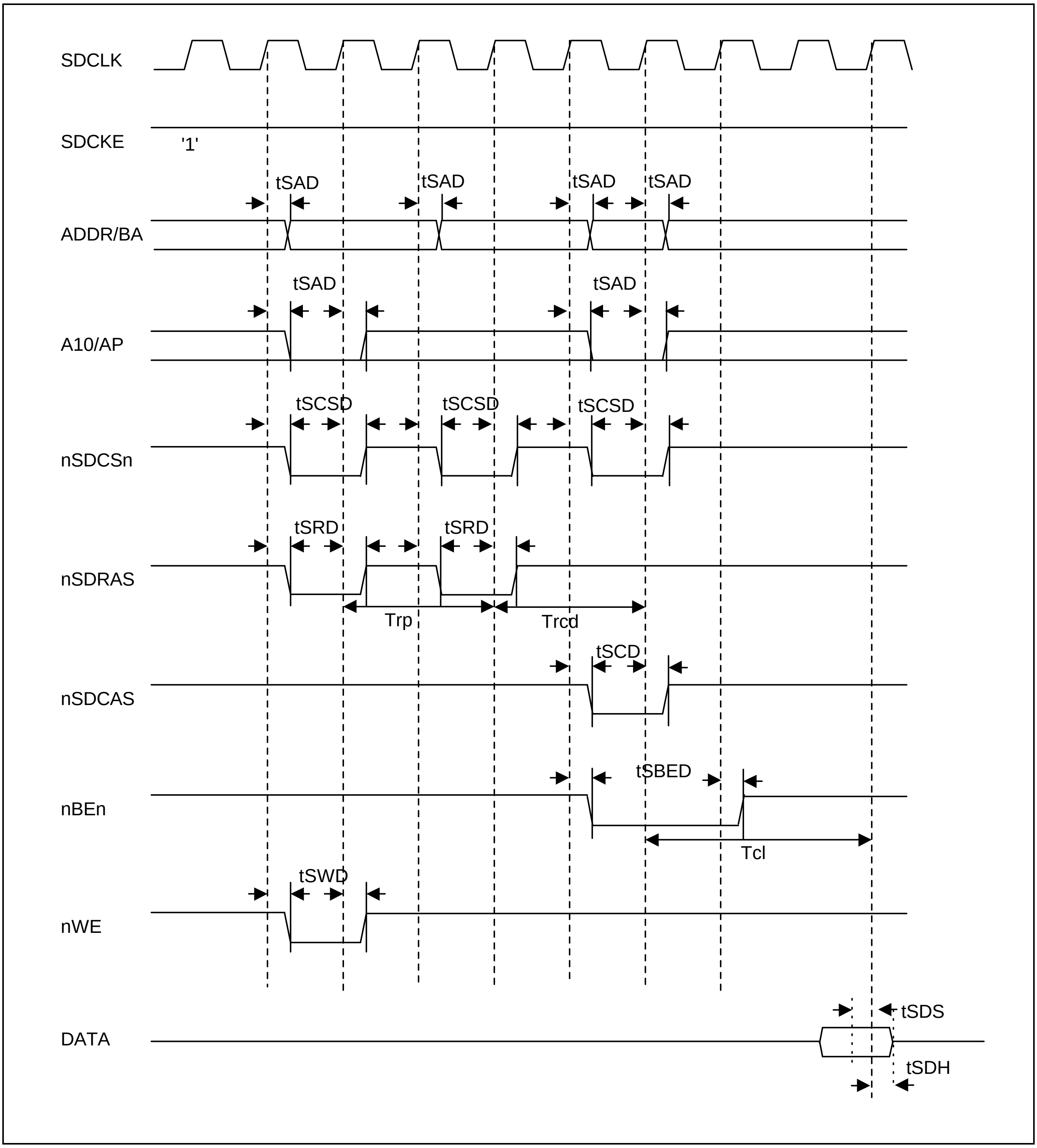
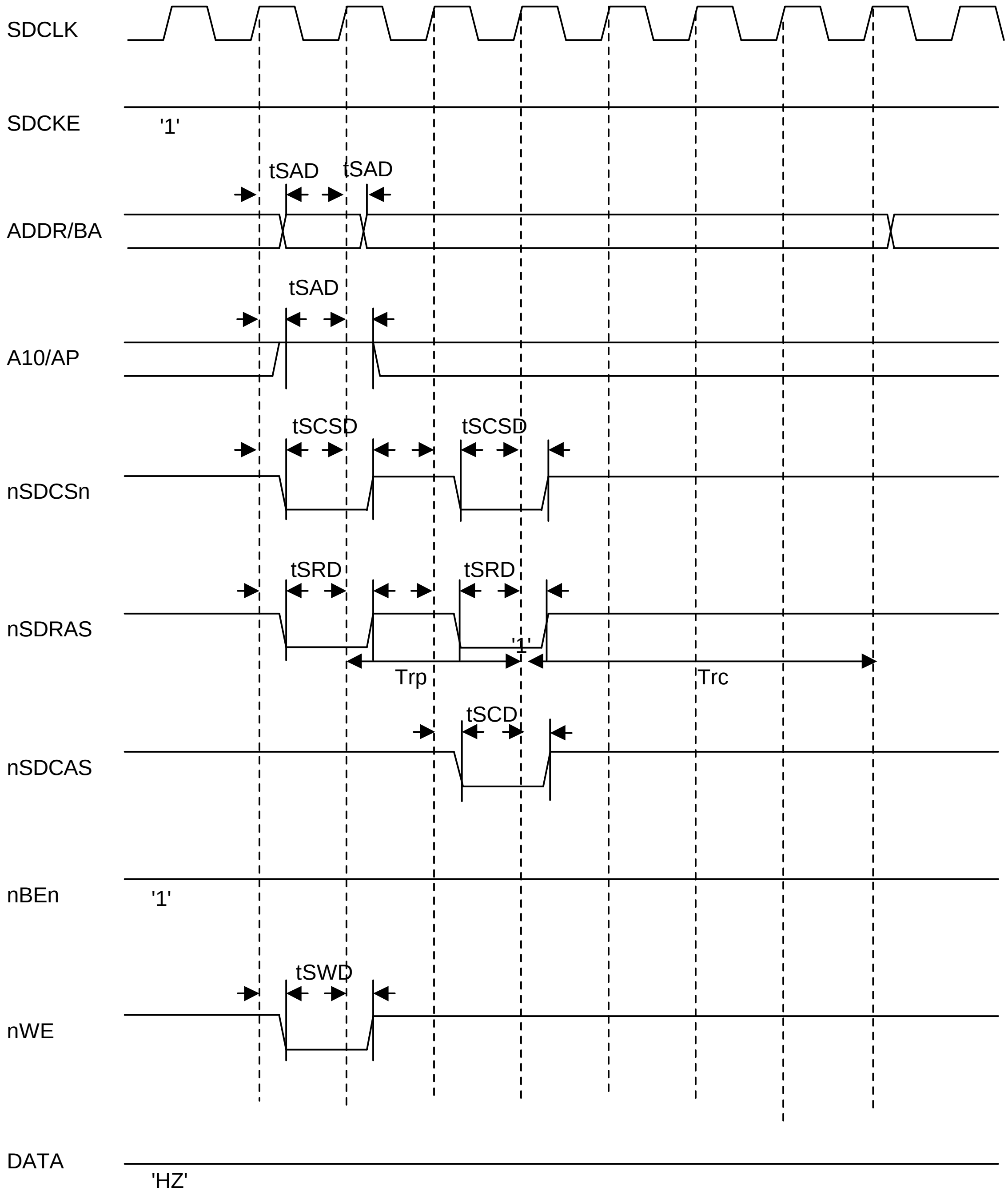


Figure 18-25. SDRAM Single READ Timing(II) (Trp=2, Trcd=2, Tcl=3)



NOTE: Before executing auto/self refresh command, all banks must be idle state.

Figure 18-26. SDRAM Auto Refresh Timing (Trp=2, Trc=4)

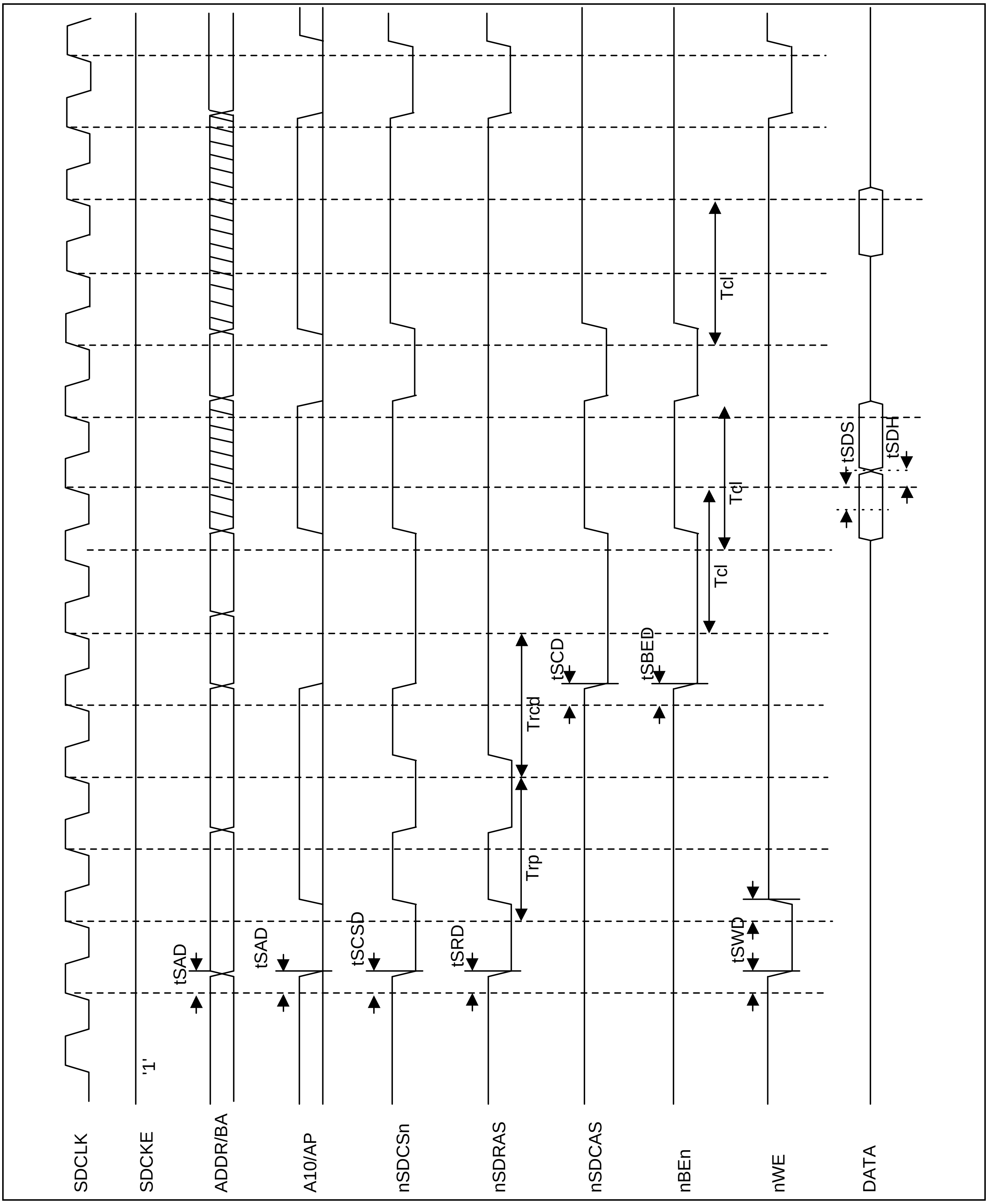
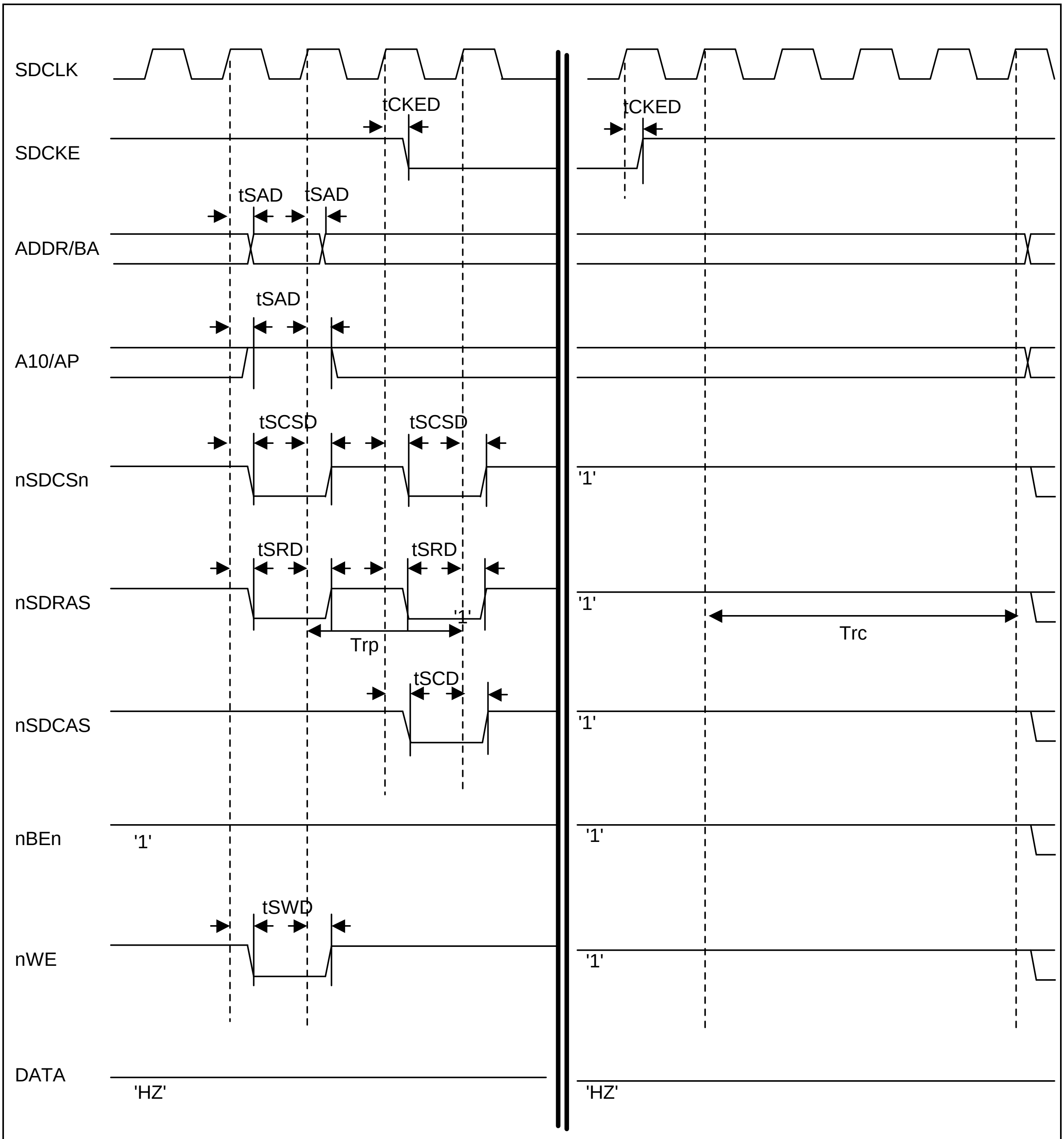


Figure 18-27. SDRAM Page Hit-Miss READ Timing (Trp=2, Trcd=2, Tcl=2)



NOTE: Before executing auto/self refresh command, all banks must be idle state.

Figure 18-28. SDRAM Self Refresh Timing (Trp=2, Trc=4)

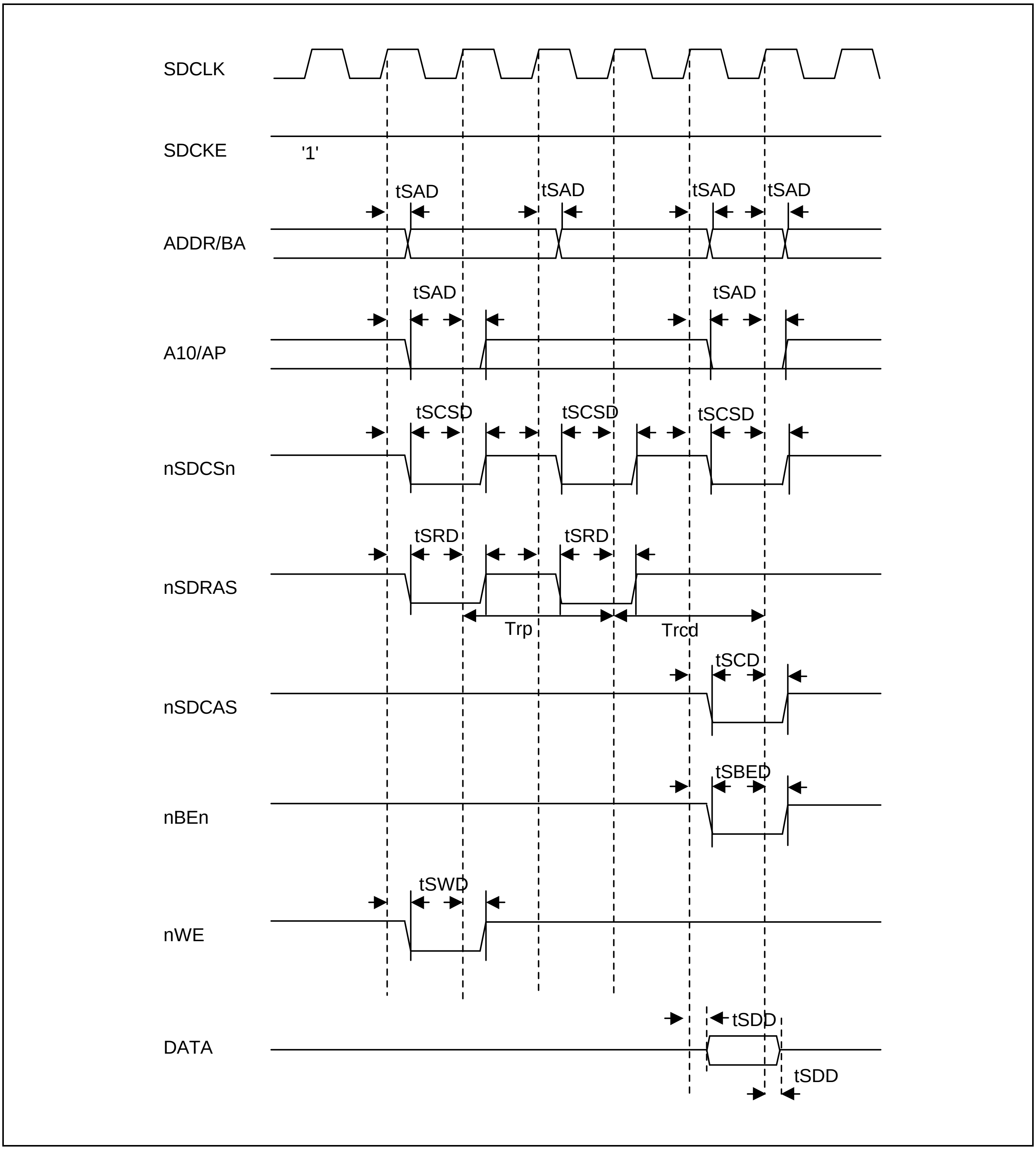


Figure 18-29. SDRAM Single Write Timing (Trp=2, Trcd=2)

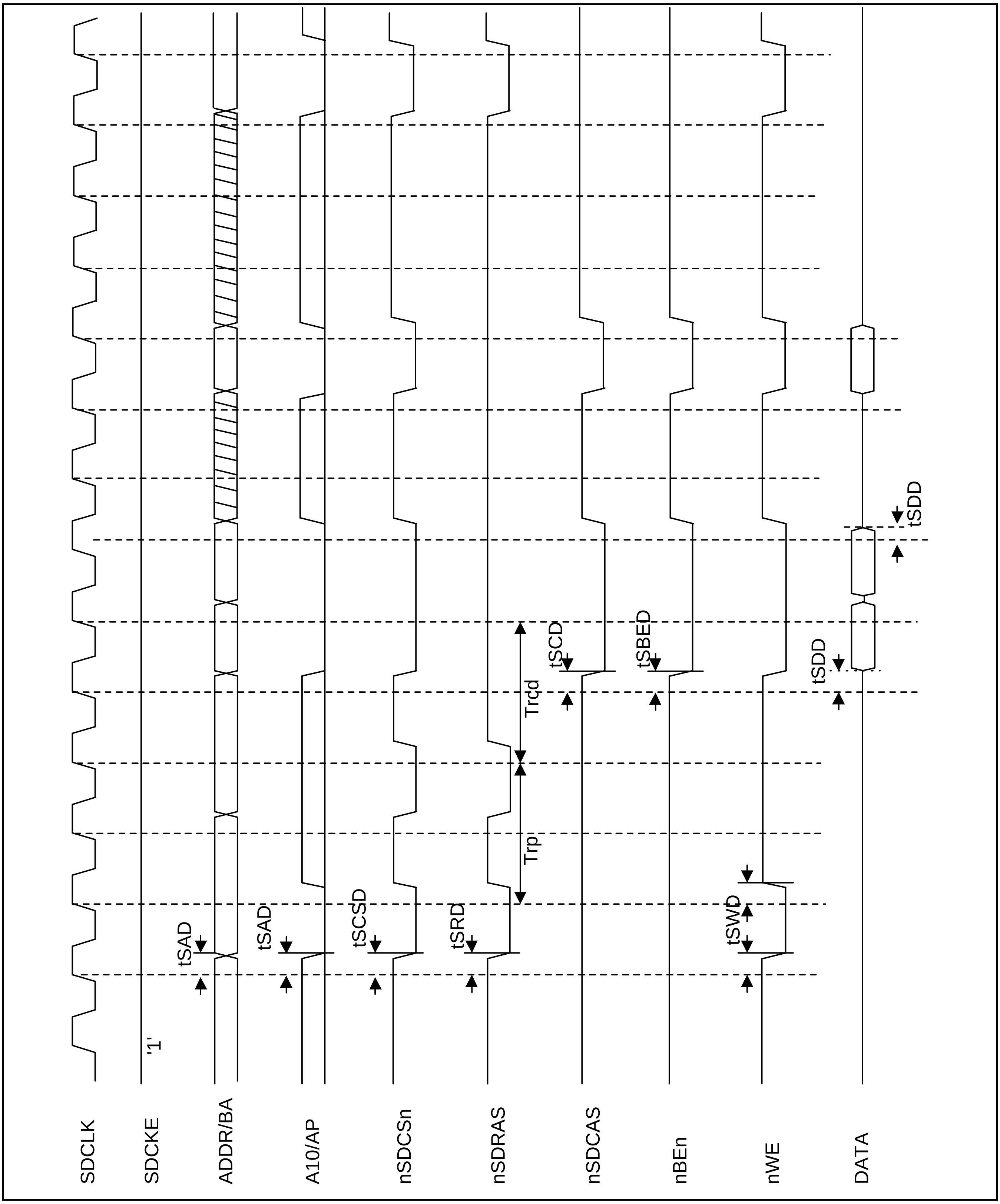


Figure 18-30. SDRAM Page Hit-Miss Write Timing (Trp=2, Trcd=2, Tcl=2)

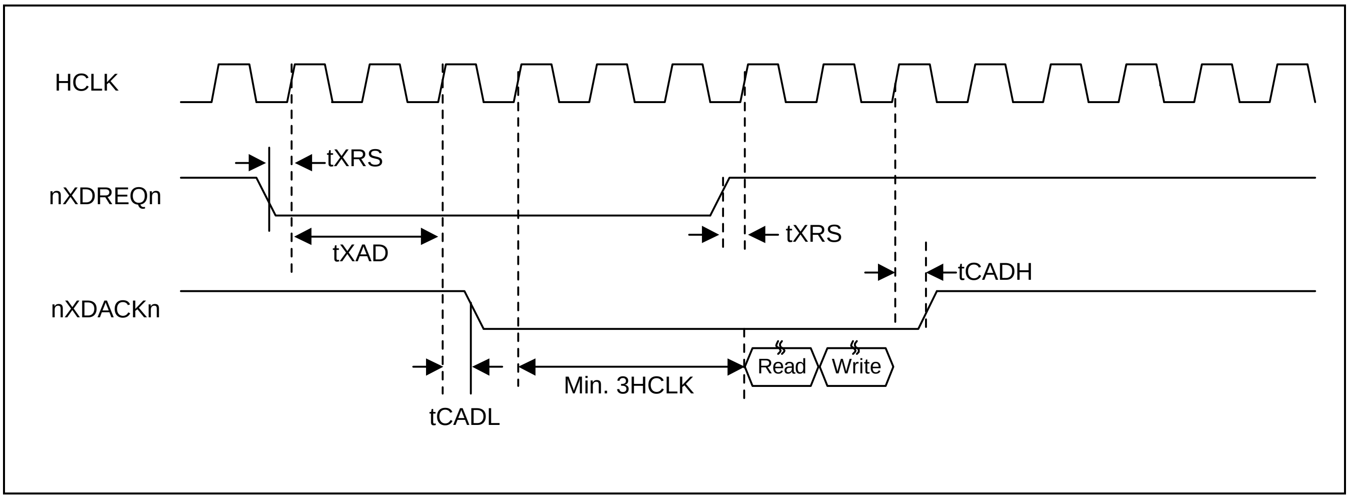


Figure 18-31. External DMA Timing (Handshake, Single transfer)

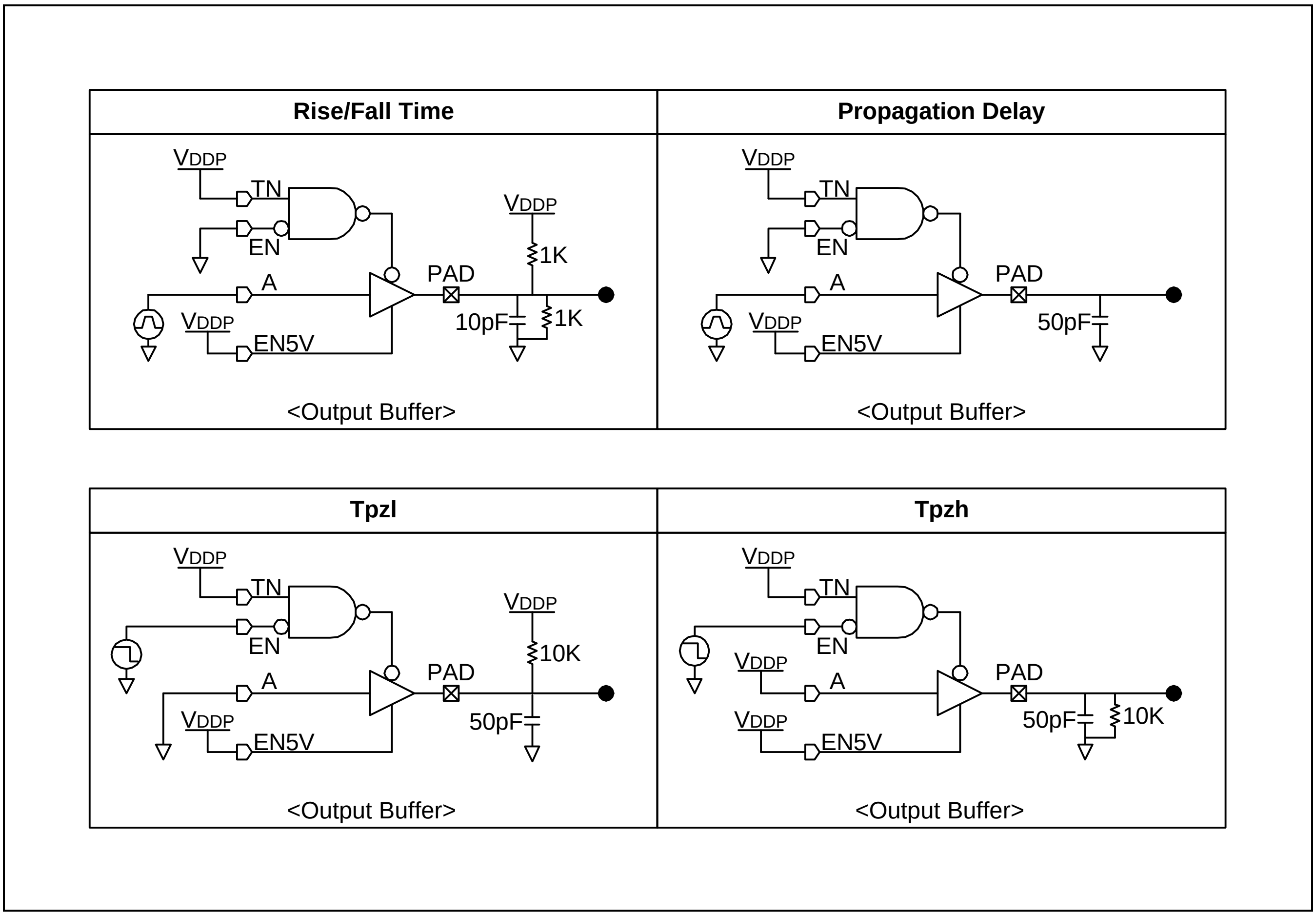


Figure 18-32. PCI output AC characteristics test circuits for 3.3V signaling

Table 18-5. Clock Timing Constants

(V_{DD} = 1.8 V -0.1 V/+0.15 V, V_{DDP} = 3.3 V ± 0.3 V, T_A = 0 to 70 °C, max/min = typ ± 30%)

Parameter	Symbol	Min	Typ	Max	Unit
Crystal clock input frequency	f_{XTAL}	6	–	10	MHz
Crystal clock input cycle time	$t_{XTALCYC}$	100	–	166.7	ns
AHBCLK(internal) to CLKout	$t_{HCLK2CLK}$	–	7.7	–	ns
AHBCLK(internal) to SDCLK	$t_{HCLK2SDCLK}$	–	2.8	–	ns
SDCLK to CLKout	$t_{SDCLK2CLK}$	–	4.9	–	ns
Mode reset hold time	T_{mDRH}	3.0	–	–	ns
Reset assert time after clock stabilization	t_{RESW}	4	–	–	OSC (Fin)
Power-on oscillation setting time	t_{LOCK}	–	–	4096	OSC (Fin)
the interval before CPU runs after nRESET is released.	$T_{RST2RUN}$	–	7	–	OSC (Fin)

Table 18-6. ROM/SRAM Bus Timing Constants

(V_{DD} = 1.8 V -0.1 V/+0.15 V, V_{DDP} = 3.3 V ± 0.3 V, T_A = 0 to 70 °C)

Parameter	Symbol	Min	Typ	Max	Unit
ROM/SRAM Address Delay	t_{RAD}	–	13	–	ns
ROM/SRAM Chip select Delay	t_{RCD}	–	8	–	ns
ROM/SRAM Output enable Delay	t_{ROD}	–	7	–	ns
ROM/SRAM read Data Setup time.	t_{RDS}	–	1	–	ns
ROM/SRAM read Data Hold time.	t_{RDH}	–	5	–	ns
ROM/SRAM Byte Enable Delay	t_{RBED}	–	8	–	ns
ROM/SRAM Write Byte Enable Delay	t_{RWBED}	–	8	–	ns
ROM/SRAM output Data Delay	t_{RDD}	–	8	–	ns
ROM/SRAM external Wait Setup time	t_{WS}	–	1	–	ns
ROM/SRAM external Wait Hold time	t_{WH}	–	5	–	ns
ROM/SRAM Write enable Delay	t_{RWD}	–	9	–	ns

Table 18-7. DRAM Bus Timing Constants

(V_{DD} = 1.8 V -0.1 V/+0.15 V, V_{DDP} = 3.3 V ± 0.3 V, T_A = 0 to 70 °C)

Parameter	Symbol	Min	Typ	Max	Unit
DRAM Address Delay	t_{DAD}	–	9	–	ns
DRAM Row active Delay	t_{DRD}	–	14	–	ns
DRAM Read Column active Delay	t_{DRCD}	–	15	–	ns
DRAM Output enable Delay	t_{DOD}	–	14	–	ns
DRAM read Data Setup time	t_{DDS}	–	1	–	ns
DRAM read Data Hold time	t_{DDH}	–	5	–	ns
DRAM Write Cas active Delay	t_{DWCD}	–	15	–	ns
DRAM Cbr Cas active Delay	t_{DCCD}	–	9	–	ns
DRAM Write enable Delay	t_{DWD}	–	15	–	ns
DRAM output Data Delay	t_{DDD}	–	16	–	ns

Table 18-8. SDRAM Bus Timing Constants

(V_{DD} = 1.8 V -0.1 V/+0.15 V, V_{DDP} = 3.3 V ± 0.3 V, T_A = 0 to 70 °C)

Parameter	Symbol	Min	Typ	Max	Unit
SDRAM Address Delay	t_{SAD}	–	5	–	ns
SDRAM Chip Select Delay	t_{SCSD}	–	4	–	ns
SDRAM Row active Delay	t_{SRD}	–	4	–	ns
SDRAM Column active Delay	t_{SCD}	–	4	–	ns
SDRAM Byte Enable Delay	t_{SBED}	–	4	–	ns
SDRAM Write enable Delay	t_{SWD}	–	4	–	ns
SDRAM read Data Setup time	t_{SDS}	–	4	–	ns
SDRAM read Data Hold time	t_{SDH}	–	0	–	ns
SDRAM output Data Delay	t_{SDD}	–	5	–	ns
SDRAM Clock Eable Delay	T_{cked}	–	5	–	ns

Table 18-9. DMA Controller Module Signal Timing Constants

(V_{DD} = 1.8 V -0.1 V/+0.15 V, V_{DDP} = 3.3 V ± 0.3 V, T_A = 0 to 70 °C)

Parameter	Symbol	Min	Typ.	Max	Unit
External Request Setup	t_{XRS}	–	9.3	–	ns
Access to Ack Delay when Low transition	t_{CADL}	–	6.8	–	Ns
Access to Ack Delay when High transition	t_{CADH}	–	6.6	–	Ns
External Request Delay	t_{XAD}	2	–	–	HCLK

Table 18-10. IIC BUS Controller Module Signal Timing

(V_{DD} = 1.8 V -0.1 V/+0.15 V, V_{DDP} = 3.3 V ± 0.3 V, T_A = 0 to 70 °C)

Parameter	Symbol	Min	Typ.	Max	Unit
SCL clock frequency	f_{SCL}	–	–	std. 100 fast 400	kHz
SCL high level pulse width	$t_{SCLHIGH}$	std. 4.0 fast 0.6	–	–	µs
SCL low level pulse width	t_{SCLLOW}	std. 4.7 fast 1.3	–	–	µs
Bus free time between STOP and START	t_{BUF}	std. 4.7 fast 1.3	–	–	µs
START hold time	t_{STARTS}	std. 4.0 fast 0.6	–	–	µs
SDA hold time	t_{SDAH}	std. 0 fast 0	–	std. - fast 0.9	µs
SDA setup time	t_{SDAS}	std. 250 fast 100	–	–	ns
STOP setup time	T_{stOPH}	std. 4.0 fast 0.6	–	–	µs

NOTE: Std. means Standard Mode and fast means Fast Mode.

Table 18-11. PCI BUS A.C. Electrical Characteristics

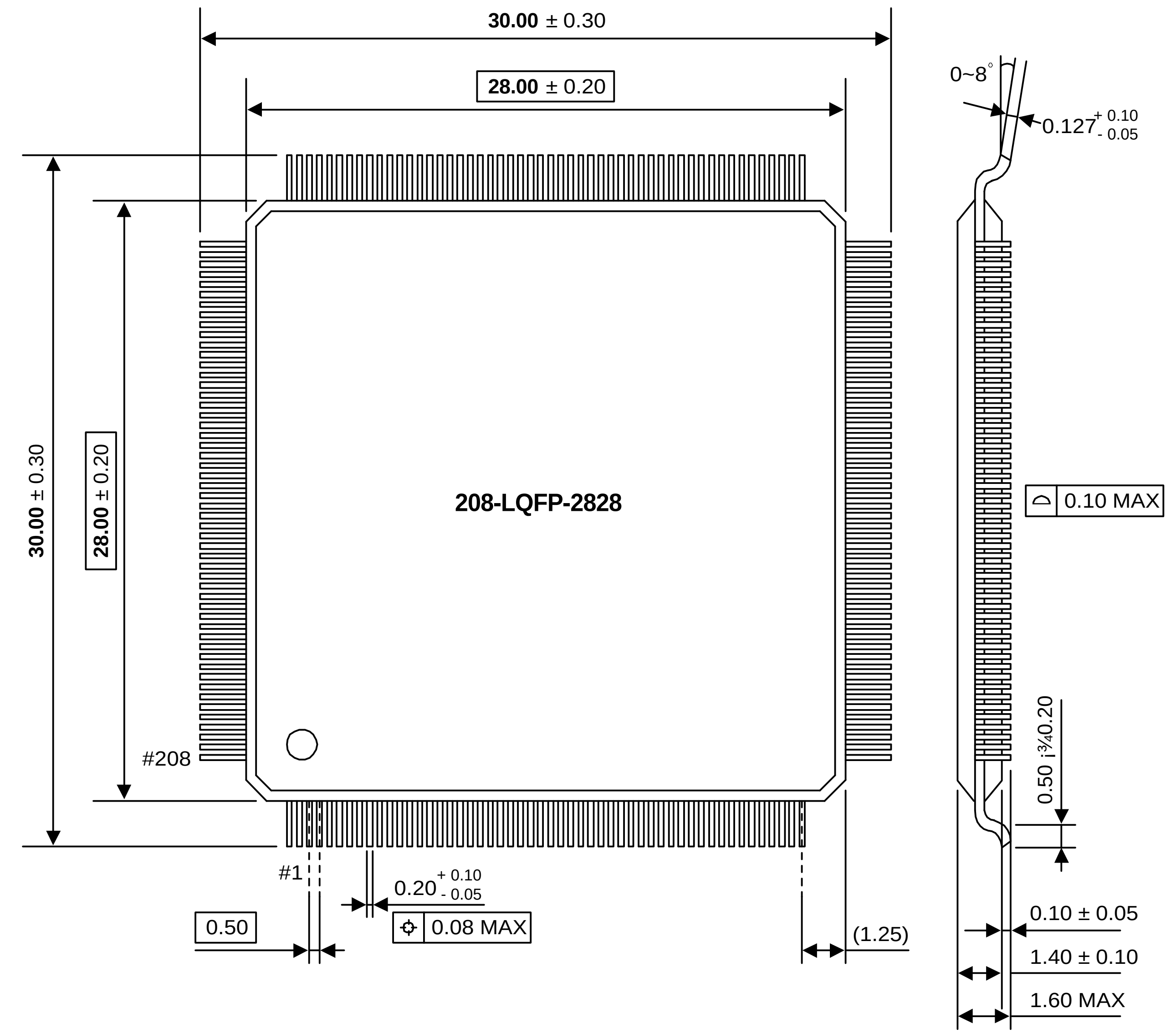
(V_{DD} = 1.8 V -0.1 V/+0.15 V, V_{DDP} = 3.3 V ± 0.3 V, T_A = 0 to 70 °C)

Parameter	Symbol	3.3V Signaling (33MHz or 66MHz)				Unit
		Condition	Min	Typ.	Max	
Switching Current High	I_{OH}(AC)	V_{OUT} = 0.3V_{DDP}	-12V_{DDP}			mA
		V_{OUT} = 0.7V_{DDP}			-32V_{DDP}	
		V_{OUT} = 0.9V_{DDP}	-1.71V_{DDP}			
Switching Current Low	I_{OL}(AC)	V_{OUT} = 0.6V_{DDP}	16V_{DDP}			mA
		V_{OUT} = 0.1V_{DDP}	2.67V_{DDP}			
		V_{OUT} = 0.18V_{DDP}			38V_{DDP}	
Low Clamp Current	I_{CL}	-3 < V_{IN} ≤ -1	-25+(V_{IN}+1) /0.015			mA
High Clamp Current	I_{CH}	V_{DDP}+1 ≤ V_{IN} < V_{DDP}+4	25+(V_{IN} -$V_{DDP_}$1) /0.015			mA
Output Rise Time	Tr	PAD(0.3V_{DDP}) to PAD(0.6V_{DDP}) (refer to Figure 18-32)	1.0		4.0	V/ns
Output Fall Time	Tf	PAD(0.6V_{DDP}) to PAD(0.3V_{DDP}) (refer to Figure 18-32)	1.0		4.0	V/ns
Output Falling Propagation delay	Tplh	A(0.5V_{DDP}) to PAD(0.4V_{DDP}) (refer to Figure 18-32)	1.3		4.0	ns
Output Rising Propagation delay	Tphl	A(0.5V_{DDP}) to PAD(0.4V_{DDP}) (refer to Figure 18-32)	1.3		4.0	ns
Output Enable time	Tpzl	EN(0.5V_{DDP}) to PAD(0.4V_{DDP}) (refer to Figure 18-32)	2.0		4.0	ns
Output Enable time	Tpzh	EN(0.5V_{DDP}) to PAD(0.4V_{DDP}) (refer to Figure 18-32)	2.0		4.3	ns
Input Rising Propagation delay	Tphl_in	PAD(0.4V_{DDP}) to Y(0.5V_{DDP}) CL = 1pF	0.4		0.8	ns
Input Falling Propagation delay	Tplh_in	PAD(0.4V_{DDP}) to Y(0.5V_{DDP}) CL = 1pF	0.4		0.8	ns

NOTES

19 MECHANICAL DATA

PACKAGE DIMENSIONS



NOTE: Dimensions are in millimeters.

Figure 19-1. 208-LQFP-2828 Package Dimensions

NOTES