Chapter 10

10.1. LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY prob10_1 IS
  GENERIC ( modulus : INTEGER := 8 ) ;
  PORT ( Resetn : IN STD_LOGIC ;
         Clock, E, L : IN STD_LOGIC ;
         R : IN INTEGER RANGE 0 TO modulus—1 ;
         Q : OUT INTEGER RANGE 0 TO modulus—1 ) ;
END prob10_1 ;

ARCHITECTURE Behavior OF prob10_1 IS
  SIGNAL Count : INTEGER RANGE 0 TO modulus—1 ;
BEGIN
  PROCESS ( Resetn, Clock )
  BEGIN
    IF Resetn = '0' THEN
      Count <= 0 ;
    ELSIF (Clock'EVENT AND Clock = '1') THEN
      IF E = '1' THEN
        IF L = '1' THEN
          Count <= R ;
        ELSE
          Count <= Count+1 ;
        END IF;
      END IF ;
    END IF ;
    END IF ;
  END PROCESS ;
  Q <= Count ;
END Behavior ;

10.2. (a) A modified ASM chart that has only Moore-type outputs in state S2 is given below.
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.components.shiftrne;

ENTITY prob10_2 IS
  PORT ( Clock, Resetn : IN STD_LOGIC;
      LA, s : IN STD_LOGIC;
      Data : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      B : BUFFER INTEGER RANGE 0 to 8;
      Done : OUT STD_LOGIC);
END prob10_2;

ARCHITECTURE Behavior OF prob10_2 IS
  TYPE STATE_TYPE IS (S1, S2, S3, S4);
  SIGNAL y : STATE_TYPE;
  SIGNAL A : STD_LOGIC_VECTOR(7 DOWNTO 0);
  SIGNAL z, EA, LB, EB, low : STD_LOGIC;
BEGIN
  FSM_transitions: PROCESS (Resetn, Clock)
  BEGIN
    IF Resetn = '0' THEN
      y <= S1;
    ELSIF (Clock'EVENT AND Clock='1') THEN
      CASE y IS
        WHEN S1 =>
          IF s = '0' THEN y <= S1; ELSE y <= S2; END IF;
        WHEN S2 | S3 =>
          IF z = '0' THEN
            IF A(0) = '0' THEN y <= S2;
            ELSE y <= S3;
            END IF;
          ELSE
            y <= S4;
          END IF;
        WHEN S4 =>
          IF s = '1' THEN y <= S4; ELSE y <= S1; END IF;
      END CASE;
    END IF;
  END PROCESS;
  FSM_outputs: PROCESS (y, s, LA)
  BEGIN
    EA <= '0'; LB <= '0'; EB <= '0'; Done <= '0';
    CASE y IS
      WHEN S1 =>
        LB <= '1'; EB <= '1';
        IF s = '0' AND LA = '1' THEN EA <= '1'; ELSE EA <= '0'; END IF;
      WHEN S2 =>
        EA <= '1';
  ... cont'
WHEN S3 =>
    EA <= '1' ; EB <= '1' ;
WHEN S4 =>
    Done <= '1' ;
END CASE ;
END PROCESS ;
- - The datapath circuit is described below
upcount: PROCESS ( Resetn, Clock )
BEGIN
    IF Resetn = '0' THEN
        B <= 0 ;
    ELSIF (Clock'EVENT AND Clock = '1') THEN
        IF EB = '1' THEN
            IF LB = '1' THEN B <= 0 ;
            ELSE B <= B + 1 ;
            END IF ;
        END IF ;
    END IF ;
END PROCESS ;
low <= '0' ;
ShiftA: shiftrne GENERIC MAP ( N => 8 ) PORT MAP ( Data, LA, EA, low, Clock, A ) ;
z <= '1' WHEN A = "00000000" ELSE '0' ;
END Behavior ;

10.3. (a)
(c) The ASM chart for the control circuit is shown below. Note that we assume the EB signal is controlled by external logic.
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE work.components.all;

ENTITY prob10_3 IS
  GENERIC ( N : INTEGER := 8 ; NN : INTEGER := 16 ) ;
  PORT ( Clock : IN STD_LOGIC ;
         Resetn : IN STD_LOGIC ;
         LA, LB, s : IN STD_LOGIC ;
         DataA : IN STD_LOGIC VECTOR (N-1 DOWNTO 0) ;
         DataB : IN STD_LOGIC VECTOR (N-1 DOWNTO 0) ;
         P : BUFFER STD_LOGIC VECTOR (NN-1 DOWNTO 0 ) ;
         Done : OUT STD_LOGIC ) ;
END prob10_3 ;
ARCHITECTURE Behavior OF prob10_3 IS

TYPE State_type IS ( S1, S2, S3 ) ;
SIGNAL y : State_type ;
SIGNAL Psel, EA, EP, Zero : STD_LOGIC ;
SIGNAL B, N_Zeros : STD_LOGIC_VECTOR(N - 1 DOWNTO 0) ;
SIGNAL A, Ain, DataP, Sum : STD_LOGIC_VECTOR(NN - 1 DOWNTO 0) ;
SIGNAL ZeroI, C : INTEGER RANGE 0 TO N - 1 ;
SIGNAL EC, LC, Bc, z : STD_LOGIC ;

BEGIN
FSM_transitions: PROCESS ( Resetn, Clock )
BEGIN
IF Resetn = '0' THEN
  y <= S1 ;
ELSIF (Clock'EVENT AND Clock='1') THEN
  CASE y IS
    WHEN S1 =>
      IF s = '0' THEN y <= S1 ; ELSE y <= S2 ; END IF ;
    WHEN S2 =>
      IF z = '0' THEN y <= S2 ; ELSE y <= S3 ; END IF ;
    WHEN S3 =>
      IF s = '1' THEN y <= S3 ; ELSE y <= S1 ; END IF ;
  END CASE ;
END IF ;
END PROCESS ;

FSM_outputs: PROCESS ( y, s , LA, Bc )
BEGIN
  EP <= '0' ; EA <= '0' ; Done <= '0' ; Psel <= '0' ;
  EC <= '0' ; LC <= '0' ;
  CASE y IS
    WHEN S1 =>
      EP <= '1' ; EC <= '1' ; LC <= '1' ;
      IF s = '0' AND LA = '1' THEN EA <= '1' ; ELSE EA <= '0' ; END IF ;
    WHEN S2 =>
      Psel <= '1' ; EA <= '1' ; EC <= '1' ;
      IF Bc = '1' THEN EP <= '1' ; ELSE EP <= '0' ; END IF ;
    WHEN S3 =>
      Done <= '1' ;
  END CASE ;
END PROCESS ;

-- datapath
Zero <= '0' ;
N_Zeros <= (OTHERS => '0') ;
Ain <= N_Zeros & DataA ;

ShiftA: shiftlne GENERIC MAP ( N => NN )
  PORT MAP ( Ain, LA, EA, Zero, Clock, A ) ;
RegB: regne GENERIC MAP ( N => N )
  PORT MAP ( DataB, Resetn, LB, Clock, B ) ;

... con't
Sum <= A + P;
DataP <= Sum WHEN Psel='1' ELSE (OTHERS => '0');
RegP: regne GENERIC MAP ( N => NN )
    PORT MAP ( DataP, Resetn, EP, Clock, P );
ZeroI <= 0;
-- Multiplexer to Select bit C from B
Bc <= B(C);
upcounter: prob10_1
    GENERIC MAP ( modulus => N )
    PORT MAP ( Resetn, Clock, EC, LC, ZeroI, C );
-- check if we have gone through all n bits of B
z <= '1' WHEN (C = (N-1)) ELSE '0';
END Behavior;

10.4. LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
USE work.components.all ;
ENTITY prob10_4 IS
    GENERIC ( N : INTEGER := 8 ) ;
    PORT ( Clock, Resetn : IN STD_LOGIC ;
          DataA, DataB : IN STD_LOGIC VECTOR(N-1 DOWNTO 0) ;
          LA, EB, s : IN STD_LOGIC ;
          Q, R : OUT STD_LOGIC VECTOR(N-1 DOWNTO 0) ;
          Done : OUT STD_LOGIC ) ;
END prob10_4 ;
ARCHITECTURE Behavior OF prob10_4 IS
    TYPE State_type IS (S1, S2, S3, S4) ;
    SIGNAL y : State_type ;
    SIGNAL EC, LC, ER, LR, EA, EQ, LQ, Rsel, low, Cout, z : STD_LOGIC ;
    SIGNAL RminusB, AR, Rmux : STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
    SIGNAL BR, RR, QR, Zero : STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
    SIGNAL RminusBwCarry : STD_LOGIC_VECTOR(N DOWNTO 0) ;
    SIGNAL CR : INTEGER RANGE 0 TO N-1 ;
BEGIN
    FSM_transitions: PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            y <= S1 ;
        ELSIF (Clock'EVENT AND Clock='1') THEN
            CASE y IS
                WHEN S1 =>
                    IF s = '0' THEN y <= S1 ; ELSE y <= S2 ; END IF ;
                WHEN S2 =>
                    y <= S3 ;
                WHEN others =>
                    y <= S4 ;
            END CASE ;
        END IF ;
    END PROCESS ;
END Behavior ;
WHEN S3 =>
    IF z = '1' THEN y <= S4 ; ELSE y <= S2 ; END IF ;
WHEN S4 =>
    IF s = '1' THEN y <= S4 ; ELSE y <= S1 ; END IF ;
END CASE ;
END IF ;
END PROCESS ;
FSM_outputs: PROCESS ( y, s, Cout, z, LA )
BEGIN
    Rsel <= '0' ; LR <= '0' ; ER <= '0' ; LC <= '0' ; EC <= '0' ;
    EA <= '0' ; EQ <= '0' ; Done <= '0' ;
    CASE y IS
        WHEN S1 =>
            LR <= '1' ; ER <= '1' ; LC <= '1' ; EC <= '1' ;
            IF s = '0' AND LA='1' THEN EA <= '1' ; ELSE EA <= '0' ; END IF ;
        WHEN S2 =>
            ER <= '1' ; EA <= '1' ;
        WHEN S3 =>
            EQ <= '1' ; Rsel <= '1' ; EC <= '1' ;
            IF Cout = '1' THEN ER <= '1' ; LR <= '1' ; ELSE LR <= '0' ; ER <= '0' ; END IF ;
        WHEN S4 =>
            Done <= '1' ;
    END CASE ;
END PROCESS ;
SHLR: shiftline GENERIC MAP ( N => N )
    PORT MAP ( Rmux, LR, ER, AR(N-1), Clock, RR ) ;
    low <= '0' ;
SHLA: shiftline GENERIC MAP ( N => N )
    PORT MAP ( DataA, LA, EA, low, Clock, AR ) ;
    LQ <= '0' ;
    Zero <= (OTHERS => '0') ;
SHLQ: shiftline GENERIC MAP ( N => N )
    PORT MAP ( Zero, LQ, EQ, Cout, Clock, QR ) ;
REGB: regne GENERIC MAP ( N => N )
    PORT MAP ( DataB, Resetn, EB, Clock, BR ) ;
    RminusBwCarry <= ("0" & RR) + ("0" & (NOT BR)) + 1 ;
    RminusB <= RminusBwCarry(N-1 DOWNTO 0) ;
    Cout <= RminusBwCarry(N) ;
    Rmux <= RminusB WHEN Rsel='1' ELSE Zero ;
    - - downcnt component can be used instead
CNT_C: PROCESS
BEGIN
    WAIT UNTIL Clock'EVENT AND Clock='1' ;
    IF EC='1' THEN
        IF LC='1' THEN CR <= N-1 ; ELSE CR <= CR-1 ; END IF ;
    END IF ;
END PROCESS ;
z <= '1' WHEN CR=0 ELSE '0' ;
R <= RR ;
Q <= QR ;
END Behavior ;
10-9
10.5. (a)

```
Q ← 0

S1

Load A
Load B

```

10-10
(c) \[
\text{Reset}
\]

![](image)

(d) \[
\text{LIBRARY ieee ;}
\]
\[
\text{USE ieee.std_logic_1164.all ;}
\]
\[
\text{USE ieee.std_logic_unsigned.all ;}
\]
\[
\text{USE work.components.all ;}
\]

\[
\text{ENTITY prob10_5 IS}
\]
\[
\text{GENERIC ( N : INTEGER := 8 ; P2N : INTEGER := 256 ) ;}
\]
\[
\text{PORT ( Clock, Resetn, s IN STD_LOGIC ;}
\]
\[
\text{LoadA, LoadB : IN STD_LOGIC ;}
\]
\[
\text{DataA, DataB : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;}
\]
\[
\text{Q : OUT INTEGER RANGE 0 TO P2N-1 ;}
\]
\[
\text{R : BUFFER STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;}
\]
\[
\text{Done : OUT STD_LOGIC ;}
\]
\[
\text{END prob10_5 ;}
\]

\[
\text{ARCHITECTURE Behavior OF prob10_5 IS}
\]
\[
\text{TYPE State_type IS ( S1, S2, S3 ) ;}
\]
\[
\text{Signal y : State_type ;}
\]
\[
\text{SIGNAL Rsel, ER, ERint, LQ, EQ : STD_LOGIC ;}
\]
\[
\text{SIGNAL Rmux, RminusB, B : STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;}
\]
\[
\text{SIGNAL RminusBwCarry : STD_LOGIC_VECTOR(N DOWNTO 0) ;}
\]
\[
\text{SIGNAL Zero : INTEGER RANGE 0 TO P2N-1 ;}
\]
\[
\text{SIGNAL Cout : STD_LOGIC ;}
\]
\[
\text{BEGIN}
\]
\[
\ldots \text{con't}
\]
FSM transitions: PROCESS ( Resetn, Clock )
BEGIN
  IF Resetn = '0' THEN
    y <= S1 ;
  ELSIF (Clock'EVENT AND Clock='1') THEN
    CASE y IS
      WHEN S1 =>
        IF s = '0' THEN y <= S1 ; ELSE y <= S2 ; END IF ;
      WHEN S2 =>
        IF Cout='1' THEN y <= S2 ; ELSE y <= S3 ; END IF ;
      WHEN S3 =>
        IF s = '1' THEN y <= S3 ; ELSE y <= S1 ; END IF ;
    END CASE ;
  END IF ;
END PROCESS ;
FSM_outputs: PROCESS ( y, Cout )
BEGIN
  EQ <= '0'; LQ <= '0'; ER <= '0'; Rsel <= '0'; Done <= '0';
  CASE y IS
    WHEN S1 =>
      LQ <= '1'; EQ <= '1';
    WHEN S2 =>
      Rsel <= '1';
      IF Cout='1' THEN EQ <= '1'; ER <= '1'; ELSE EQ <= '0'; END IF ;
    WHEN S3 =>
      Done <= '1';
  END CASE ;
END PROCESS ;
- - Data Path Definition
Rmux <= DataA WHEN Rsel='0' ELSE RminusB ;
ERint <= ER OR LoadA ;
RegR: regne GENERIC MAP ( N => N )
  PORT MAP ( Rmux, Resetn, ERint, Clock, R ) ;
RegB: regne GENERIC MAP ( N => N )
  PORT MAP ( DataB, Resetn, LoadB, Clock, B ) ;
Zero <= 0 ;
CntQ: prob10_1 GENERIC MAP ( modulus=> P2N )
  PORT MAP ( Resetn, Clock, EQ, LQ, Zero, Q ) ;
RminusBwCarry <= ("0" & R ) + ("0" & ( NOT B )) + 1 ;
RminusB <= RminusBwCarry( N–1 DOWNTO 0 ) ;
Cout <= RminusBwCarry( N ) ;
END Behavior ;

(\textsuperscript{e}) This implementation of a divider is less efficient in the worst case when compared to the other implementations shown. The efficient algorithms presented are able to perform division in $n$ cycles for $n$-bit inputs. However, the method of repeated subtraction takes $2^n$ cycles for the worst case, which is when dividing by 1. On the other hand, if the two numbers $A$ and $B$ are close in size, then repeated subtraction is an efficient approach.
10.6. State S3 is responsible for loading the operands into the divider, while state S4 starts the division operation. These states can be combined into a single state. We can use the \( z \) flag to indicate the first time that we've entered the new combined state. When \( z = 1 \) a mealy output is produced which loads the operands and decrements the counter. Thus, the \( z \) flag changes to a 0. The combined state now produces a mealy output which starts the division, on the condition that \( z = 0 \). This control circuit ASM chart is shown below.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY prob10_7 IS
  PORT ( Clock, Resetn, s, z, zz : IN STD_LOGIC ;
        EC, LC, Ssel, ES, LA, EB, Div, Done : OUT STD_LOGIC ) ;
END prob10_7 ;

... con’t
```
ARCHITECTURE Behavior OF prob10_7 IS
  TYPE STATE_TYPE IS (S1, S2, S3, S4);
  SIGNAL y : STATE_TYPE;
BEGIN
  FSM_transitions: PROCESS ( Resetn, Clock, s, z, zz )
  BEGIN
    IF Resetn = '0' THEN
      y <= S1;
    ELSIF (Clock'EVENT AND Clock='1') THEN
      CASE y IS
        WHEN S1 =>
          IF s = '0' THEN y <= S1 ; ELSE y <= S2 ; END IF ;
        WHEN S2 =>
          IF z = '0' THEN y <= S2 ; ELSE y <= S3 ; END IF ;
        WHEN S3 =>
          IF z = '1' THEN
            y <= S3 ;
          ELSE
            IF zz = '0' THEN y <= S3 ; ELSE y <= S4 ; END IF ;
          END IF ;
        WHEN S4 =>
          IF s = '1' THEN y <= S4 ; ELSE y <= S1 ; END IF ;
      END CASE ;
    END IF ;
  END PROCESS ;
  FSM_outputs: PROCESS ( y, z )
  BEGIN
    LA <= '0' ; EB <= '0' ; EC <= '0' ; LC <= '0' ;
    ES <= '0' ; Ssel <= '0' ; Div <= '0' ; Done <= '0' ;
    CASE y IS
      WHEN S1 =>
        EC <= '1' ; LC <= '1' ; ES <= '1' ;
      WHEN S2 =>
        ES <= '1' ; Ssel <= '1' ;
      WHEN S3 =>
        IF z = '0' THEN EC <= '1' ; ELSE EC <= '0' ; END IF ;
      WHEN S4 =>
        IF z = '0' THEN Div <= '1' ; LA <= '0' ; EB <= '0' ; EC <= '0' ;
        ELSE Div <= '0' ; LA <= '1' ; EB <= '1' ; EC <= '1' ;
      END IF ;
    END CASE ;
  END PROCESS ;
END Behavior ;

10.8. The states S2 and S3 can be merged into a single state by performing the assignment \( C_j = C_i + 1 \). The circuit would require an adder to increment \( C_i \) by 1 and the outputs of this adder would be loaded in parallel into the counter \( C_j \). If instead of using counters to implement \( C_i \) and \( C_j \) we used shift registers, then the effect of producing \( C_i + 1 \) could be efficiently implemented by wiring \( C_i \) to the parallel-load data inputs on \( C_j \) such that the bits are shifted by one position.
10.9. (a) The part of the datapath circuit that needs to be modified is shown below. The rest of the datapath is the same as the circuit shown in Figure 10.37.

(b) LIBRARY ieee;
    USE ieee.std_logic_1164.all;
    USE ieee.std_logic_unsigned.all;
    USE work.components.all;

ENTITY prob10_9 IS
    GENERIC ( N : INTEGER := 4 );
    PORT ( Clock, Resetn : IN   STD_LOGIC ;
        s, WrInit, Rd : IN   STD_LOGIC ;
        DataIn  : IN   STD_LOGIC_VECTOR(N–1 DOWNTO 0);
        RAdd    : IN   INTEGER RANGE 0 TO 3 ;
        DataOut : BUFFER STD_LOGIC_VECTOR(N–1 DOWNTO 0);
        Done    : BUFFER STD_LOGIC );
END prob10_9 ;

... con’t
ARCHITECTURE Behavior OF prob10 IS
  TYPE STATE_TYPE IS ( S1, S2, S3, S4, S5, S6, S7, S8, S9 ) ;
  SIGNAL y : STATE_TYPE ;
  SIGNAL Rin : STD_LOGIC_VECTOR(0 TO 3) ; -- reg write controls
  TYPE RegArray IS ARRAY(0 TO 3) OF STD_LOGIC_VECTOR(N−1 DOWNTO 0) ;
  SIGNAL R : RegArray ; -- reg outputs
  SIGNAL RData, ABMux : STD_LOGIC_VECTOR(N−1 DOWNTO 0) ; -- reg inputs
  SIGNAL Int, Csel, Wr, BltA : STD_LOGIC ;
  -- SIGNAL CMux, IMux : INTEGER RANGE 0 TO 3 ;
  SIGNAL Ain, Bin, Aout, Bout : STD_LOGIC ;
  SIGNAL LI, LJ, EI, EJ, zi, zj : STD_LOGIC ;
  SIGNAL A, B, ABData : STD_LOGIC_VECTOR(N−1 DOWNTO 0) ;
  -- Redefined Signals
  SIGNAL C, Cj, Cmux : STD_LOGIC_VECTOR(0 TO 3);
  -- New Signals
  SIGNAL low : STD_LOGIC ;
  SIGNAL Rout : STD_LOGIC_VECTOR(0 TO 3);
  SIGNAL Addr0 : STD_LOGIC_VECTOR(0 TO 3); -- Parallel Load for Ci
  SIGNAL ExtAdd : STD_LOGIC_VECTOR(0 TO 3);
BEGIN
  FSM_transitions: PROCESS ( Resetn, Clock )
  BEGIN
    IF Resetn = '0' THEN
      y <= S1 ;
    ELSIF (Clock'EVENT AND Clock = '1') THEN
      CASE y IS
        WHEN S1 => IF S = '0' THEN y <= S1 ; ELSE y <= S2 ; END IF ;
        WHEN S2 => y <= S3 ;
        WHEN S3 => y <= S4 ;
        WHEN S4 => y <= S5 ;
        WHEN S5 => IF BltA = '1' THEN y <= S6 ; ELSE y <= S8 ; END IF ;
        WHEN S6 => y <= S7 ;
        WHEN S7 => y <= S8 ;
        WHEN S8 =>
          IF zj = '0' THEN y <= S4 ;
          ELSIF zi = '0' THEN y <= S2 ;
          ELSE y <= S9 ;
          END IF ;
        WHEN S9 =>
          IF s = '1' THEN y <= S9 ; ELSE y <= S1 ; END IF ;
      END CASE ;
    END IF ;
  END PROCESS ;
  Int <= '0' WHEN y = S1 ELSE '1' ;
  Done <= '1' WHEN y = S9 ELSE '0' ;
  FSM_outputs: PROCESS ( y, zi, zj )
  BEGIN
    LI <= '0' ; LJ <= '0' ; El <= '0' ; Ej <= '0' ; Csel <= '0' ; Wr <= '0' ;
    Ain <= '0' ; Bin <= '0' ; Aout <= '0' ; Bout <= '0' ;
    ... con't
CASE y IS
  WHEN S1 => LI <= '1'; EI <= '1';
  WHEN S2 => Ain <= '1'; LJ <= '1'; EJ <= '1';
  WHEN S3 => EJ <= '1';
  WHEN S4 => Bin <= '1'; Csel <= '1';
  WHEN S5 => -- no outputs asserted in this state
  WHEN S6 => Csel <= '1'; Wr <= '1'; Aout <= '1';
  WHEN S7 => Wr <= '1'; Bout <= '1';
  WHEN S8 => Ain <= '1';
    IF zj = '0' THEN EJ <= '1';
    ELSE EJ <= '0';
    IF zi = '0' THEN EI <= '1';
    ELSE EI <= '0';
    END IF;
  END IF;
  WHEN S9 => -- Done is assigned 1 by conditional signal assignment
END CASE;
END PROCESS;
-- define the datapath circuit
GenReg: FOR i IN 0 TO 3 GENERATE
  Reg: regne GENERIC MAP (N => N)
    PORT MAP (RData, Resetn, Rin(i), Clock, R(i));
END GENERATE;
RegA: regne GENERIC MAP (N => N)
  PORT MAP (ABData, Resetn, Ain, Clock, A);
RegB: regne GENERIC MAP (N => N)
  PORT MAP (ABData, Resetn, Bin, Clock, B);
BltA <= '1' WHEN B <= A ELSE '0';
ABMux <= A WHEN Bout='0' ELSE B;
RData <= ABMux WHEN WrInit='0' ELSE DataIn;
Addr0 <= "1000"; low <= '0';
OuterLoop: shiftrne GENERIC MAP (N => 4)
  PORT MAP (Addr0, LI, EI, low, Clock, Ci);
InnerLoop: shiftrne GENERIC MAP (N => 4)
  PORT MAP (Ci, LJ, EJ, low, Clock, Cj);
CMux <= Ci WHEN Csel= '0' ELSE Cj;
Rout <= Cmux WHEN Int='1' ELSE ExtAdd;
-- Note that the 4-to-2 encoder that drives IMux is implicitly implemented below
WITH Rout SELECT
  ABData <= R(0) WHEN "1000",
             R(1) WHEN "0100",
             R(2) WHEN "0010",
             R(3) WHEN OTHERS;
Rin <= Rout WHEN (WrInit OR Wr) = '1' ELSE "0000";
zi <= Cj(2); zj <= Cj(3);
DataOut <= (OTHERS => 'Z') WHEN Rd = '0' ELSE ABData;
WITH Radd SELECT
  ExtAdd <= "1000" WHEN 0,
            "0100" WHEN 1,
            "0010" WHEN 2,
            "0001" WHEN OTHERS;
END Behavior;
The major drawback of using shift-registers instead of counters is that the number of flip-flops is increased. Each counter uses $\log_2 n$ flip-flops while each shift register contains $n$ flip-flops. However, the shift-register requires no combinational logic to perform tests such as whether the count value $k - 2$ has been reached — in the shift register we directly access bit $k - 2$ of the register to perform this test. It should also be possible to clock the datapath at a higher maximum clock frequency when using shift-registers, because they are simpler than counters.

10.11. The VHDL code below shows the changes needed in the datapath so that an SRAM block can be used instead of registers. The SRAM block is clocked on the negative edge of the Clock signal, hence changes in the outputs produced by the other datapath elements must be stable before the negative edge; the clock period must be long enough to accommodate this constraint.

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
LIBRARY lpm;
USE lpm.lpm_components.all;
USE work.components.regne;
USE work.components.upcount;

ENTITY prob10_11 IS
  GENERIC ( N: INTEGER := 4 ) ;
  PORT ( Clock, Resetn : IN STD_LOGIC;
  s, WrInit, Rd : IN STD_LOGIC;
  DataIn : IN STD_LOGIC VECTOR(N/1 DOWNTO 0);
  RAdd : IN INTEGER RANGE 0 TO 3;
  DataOut : BUFFER STD_LOGIC VECTOR(N/1 DOWNTO 0);
  Done : BUFFER STD_LOGIC ) ;
END prob10_11 ;

ARCHITECTURE Behavior OF prob10_11 IS
  TYPE STATE_TYPE IS ( S1, S2, S3, S4, S5, S6, S7, S8, S9 ) ;
  SIGNAL y : STATE_TYPE ;
  SIGNAL Ci, Cj : INTEGER RANGE 0 TO 3 ; -- counters Ci & Cj output
  SIGNAL RData, ABMux : STD_LOGIC VECTOR(N-1 DOWNTO 0) ;
  SIGNAL Int, Csel, Wr, BltA : STD_LOGIC ;
  SIGNAL CMux, IMux : INTEGER RANGE 0 TO 3 ; -- address mux outputs
  SIGNAL Ain, Bin, Aout, Bout : STD_LOGIC ;
  SIGNAL LI, LJ, EI, EJ, zi, zj, WE, NClock : STD_LOGIC ;
  SIGNAL Zero : INTEGER RANGE 3 DOWNTO 0 ; -- parallel data for Ci = 0
  SIGNAL A, B, ABData : STD_LOGIC VECTOR(N-1 DOWNTO 0) ;
  SIGNAL MemAdd : STD_LOGIC VECTOR(1 DOWNTO 0) ; -- SRAM address holder
BEGIN
  FSM_transitions: PROCESS ( Resetn, Clock )
  BEGIN
    ... code not shown. See Figure 10.40
  END PROCESS ;
  ...
  END ;

10-18
FSM_outputs: PROCESS ( y, s, BltA, zi, zj )
BEGIN
   . . . code not shown. See Figure 10.40
END PROCESS;
Int <= '0' WHEN y = S1 ELSE '1';
Zero <= 0;
RegA: regne GENERIC MAP ( N => N )
   PORT MAP ( ABData, Resetn, Ain, Clock, A );
RegB: regne GENERIC MAP ( N => N )
   PORT MAP ( ABData, Resetn, Bin, Clock, B );
BltA <= '1' WHEN B < A ELSE '0';
ABMux <= A WHEN Bout = '0' ELSE B;
RDAta <= ABMux WHEN WrInit = '0' ELSE DataIn;
OuterLoop: upcount GENERIC MAP ( modulus => 4 )
   PORT MAP ( Resetn, Clock, EI, LI, Zero, Ci );
InnerLoop: upcount GENERIC MAP ( modulus => 4 )
   PORT MAP ( Resetn, Clock, EI, LI, Ci, Cj );
CMux <= Ci WHEN Csel = '0' ELSE Cj;
IMux <= Cmux WHEN Int = '1' ELSE Add;
MemAdd <= CONV_STD_LOGIC_VECTOR(IMux, 2);
WE <= WrInit OR Wr;
NClock <= NOT Clock;

SRAM_block : LPM_RAM_DQ
   GENERIC MAP ( LPM_WIDTH => 4, LPM_WIDTHAD => 2,
                 LPM_ADDRESS_CONTROL => "REGISTERED",
                 LPM_INDATA => "REGISTERED",
                 LPM_OUTDATA => "UNREGISTERED" )
   PORT MAP ( address => MemAdd, we => WE, q => ABData,
               inclock => NClock, data => RData );
Zi <= '1' WHEN Ci = 2 ELSE '0';
Zj <= '1' WHEN Cj = 3 ELSE '0';
DataOut <= (OTHERS => 'Z') WHEN Rd = '0' ELSE ABData;
END Behavior;

10.12. Pseudo-code that represents the $\log_2$ operation is

```
- - assume that $K \geq 1$
L = 0;
while ($K > 1$) do
   K = K / 2;
   L = L + 1;
end while;
- - L now has the largest value such that $2^L < K$
```

An ASM chart that corresponds to the pseudo-code is
From the ASM chart, a shift-register is needed to divide $K$ by 2, and a counter is needed for $L$. An appropriate datapath circuit is

An ASM chart for the control circuit is
Complete VHDL code for this circuit is shown below.

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE work.components.all;

ENTITY prob10_12 IS
  PORT ( Clock, Resetn, LK, s : IN STD_LOGIC ;
         Data : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
         L   : BUFFER INTEGER RANGE 0 TO 7 ;
         Done : OUT STD_LOGIC );
END prob10_12 ;

ARCHITECTURE Behavior OF prob10_12 IS
  TYPE State_type IS ( S1, S2, S3 ) ;
  SIGNAL y : State_type ;
  SIGNAL K : STD_LOGIC_VECTOR(7 DOWNTO 0) ;
  SIGNAL Kgt1, EK, LL, EL, low : STD_LOGIC ;
  SIGNAL Zero : INTEGER RANGE 0 TO 7 ;
BEGIN
  FSM_transitions: PROCESS ( Resetn, Clock )
  BEGIN
    IF Resetn = '0' THEN
      y <= S1 ;
    ELSIF ( Clock'EVENT AND Clock = '1' ) THEN
      ... con't
CASE y IS
    WHEN S1 =>
        IF s = '0' THEN y <= S1; ELSE y <= S2; END IF;
    WHEN S2 =>
        IF Kgt1 = '1' THEN y <= S2; ELSE y <= S3; END IF;
    WHEN S3 =>
        IF s = '1' THEN y <= S3; ELSE y <= S1; END IF;
END CASE;
END IF;
END PROCESS;

FSM outputs: PROCESS ( y, LK, Kgt1 )
BEGIN
    EL <= '0'; LL <= '0'; EK <= '0'; Done <= '0';
    CASE y IS
        WHEN S1 =>
            EL <= '1'; LL <= '1';
            IF LK = '1' THEN EK <= '1'; ELSE EK <= '0'; END IF;
        WHEN S2 =>
            EK <= '1';
            IF Kgt1 = '1' THEN EL <= '1'; ELSE EL <= '0'; END IF;
        WHEN S3 =>
            Done <= '1';
        END CASE;
    END PROCESS;

low <= '0';
ShiftK: shiftrne GENERIC MAP ( N => 8 )
    PORT MAP ( Data, LK, EK, low, Clock, K );
Zero <= 0;
CntL: upcount GENERIC MAP ( modulus => 8 )
    PORT MAP ( Resetn, Clock, EL, LL, Zero, L );
Kgt1 <= '1' WHEN K > 1 ELSE '0';
END Behavior;

10.13. LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE work.components.all;

entity prob10_13 is
    GENERIC ( N : INTEGER := 8 );
    PORT ( Clock, Resetn : IN STD_LOGIC;
        Data : IN STD_LOGIC_VECTOR(N−1 DOWNTO 0);
        RAdd : IN INTEGER RANGE 0 TO 3;
        s, ER : IN STD_LOGIC;
        M : OUT STD_LOGIC_VECTOR(N−1 DOWNTO 0);
        Done : OUT STD_LOGIC );
END prob10_13;

...con’t
ARCHITECTURE Behavior OF prob10_13 IS

TYPE State_type IS ( S1, S2, S3, S4, S5 ) ;
SIGNAL y : State_type ;
SIGNAL Dec_RAdd, Rin : STD_LOGIC_VECTOR(0 TO 3) ;
TYPE RegArray IS
  ARRAY(0 TO 3) OF STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
SIGNAL R : RegArray ;
SIGNAL Ri, SR, Sin : STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
SIGNAL Sum : STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
SIGNAL ES, Ssel, EC, LC, z : STD_LOGIC ;
SIGNAL C : INTEGER RANGE 0 TO 3 ;
SIGNAL LA, EB, zz, Div : STD_LOGIC ;
BEGIN
  FSM_transitions: PROCESS ( Resetn, Clock )
BEGIN
    IF Resetn = '0' THEN
      y <= S1 ;
    ELSIF (Clock'EVENT AND Clock = '1') THEN
      CASE y IS
        WHEN S1 => IF s = '0' THEN y <= S1 ; ELSE y <= S2 ; END IF ;
        WHEN S2 => IF z = '1' THEN y <= S3 ; ELSE y <= S2 ; END IF ;
        WHEN S3 => y <= S4 ;
        WHEN S4 => IF zz='0' THEN y <= S4 ; ELSE y <= S5 ; END IF ;
        WHEN S5 => IF s = '1' THEN y <= S5 ; ELSE y <= S1 ; END IF ;
      END CASE ;
    END IF ;
  END PROCESS ;
  FSM_outputs: PROCESS ( y, z )
BEGIN
    EC <= '0' ; LC <= '0' ; Ssel <= '0' ; ES <= '0' ; Done <= '0' ;
    LA <= '0' ; EB <= '0' ; Div <= '0' ;
    CASE y IS
      WHEN S1 =>
        EC <= '1' ; LC <= '1' ; ES <= '1' ;
      WHEN S2 =>
        ES <= '1' ; Ssel <= '1' ;
        IF z='0' THEN EC <= '1' ; ELSE EC <= '0' ; END IF ;
      WHEN S3 =>
        LA <= '1' ; EB <= '1' ;
      WHEN S4 =>
        Div <= '1' ; Div <= '1' ; (not really used in this cct)
      WHEN S5 =>
        Div <= '1' ; Done <= '1' ;
    END CASE ;
  END PROCESS ;
  WITH RAdd SELECT
    Dec_RAdd <= "1000" WHEN 0,
               "0100" WHEN 1,
               "0010" WHEN 2,
               "0001" WHEN OTHERS ;
  Rin <= Dec_RAdd WHEN ER = '1' ELSE "0000" ;
END FSM_outputs ;
END BEGIN ;

END prob10_13 ;
10.14. From Figures 10.26 and 10.27, we can see that the divider subcircuit does not use its adder while in state $S_1$. Since the control circuit for the divider stays in $S_1$ while $s = 0$, it is possible to lend the adder to another circuit while we are in $S_1$ and $s = 0$. The Figure below shows the changes needed in the divider circuit: a multiplexer is added to each data input on the adder. The multiplexer select line is driven by the divider’s $s$ input — this signal is called $Div$ in the figure, because $Div$ is the signal in the Mean circuit that drives the $s$ input on the divider subcircuit. When $Div = 1$ the adder is provided with the normal data used in the division operation. But when $Div = 0$ the adder is provided with the external data inputs named $Op_1$ and $Op_2$, which come from the Mean circuit. Note that the $C_{in}$ input on the adder is controlled by $Div$. This feature is needed because the divider uses its adder to perform subtraction.

VHDL code for the modified divider circuit is shown below.
ENTITY divider IS
  GENERIC ( N : INTEGER := 8 ) ;
  PORT ( Clock, Resetn : IN STD_LOGIC ;
        s, LA, EB : IN STD_LOGIC ;
        DataA, DataB : IN STD_LOGIC_VECTOR(N−1 DOWNTO 0) ;
        R, Q : BUFFER STD_LOGIC_VECTOR(N−1 DOWNTO 0) ;
        Op1, Op2 : IN STD_LOGIC_VECTOR(N−1 DOWNTO 0) ; -- new ports
        Result : OUT STD_LOGIC_VECTOR(N−1 DOWNTO 0) ; -- new port
        Done : OUT STD_LOGIC ) ;
END divider ;
ARCHITECTURE Behavior OF divider IS
  TYPE STATE_TYPE IS ( S1, S2, S3 ) ;
  SIGNAL y : STATE_TYPE ;
  SIGNAL Zero, Cout, z : STD_LOGIC ;
  SIGNAL EA, Rsel, LR, ER, ER0, LC, EC, R0 : STD_LOGIC ;
  SIGNAL A, B, DataR : STD_LOGIC_VECTOR(N−1 DOWNTO 0) ;
  SIGNAL Sum : STD_LOGIC_VECTOR(N DOWNTO 0) ; -- adder outputs
  SIGNAL Count : INTEGER RANGE 0 TO N−1 ;
  SIGNAL AddIn1 : STD_LOGIC_VECTOR(N DOWNTO 0) ; -- see problem 10.14
  SIGNAL AddIn2 : STD_LOGIC_VECTOR(N−1 DOWNTO 0) ;
BEGIN
  FSM transitions: PROCESS ( Resetn, Clock )
  BEGIN
    ... code not shown. See Figure 10.28
  END PROCESS ;
  FSM outputs: PROCESS ( s, y, Cout, z )
  BEGIN
    ... code not shown. See Figure 10.28
  END PROCESS ;
  Zero <= '0' ;
  RegB: regne GENERIC MAP ( N => N )
  PORT MAP ( DataB, Resetn, EB, Clock, B ) ;
  ShiftR: shiftlne GENERIC MAP ( N => N )
  PORT MAP ( DataR, LR, ER, R0, Clock, R ) ;
  FF R0: muxdff PORT MAP ( Zero, A(N−1), ER0, Clock, R0 ) ;
  ShiftA: shiftlne GENERIC MAP ( N => N )
  PORT MAP ( DataA, LA, EA, Cout, Clock, A ) ;
  Q <= A ;
  Counter: downcnt GENERIC MAP ( modulus => N )
  PORT MAP ( Clock, EC, LC, Count ) ;
  z <= '1' WHEN Count = 0 ELSE '0' ;
  ... con't
Code for the modified Mean circuit is shown below.

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE work.components.all;

ENTITY prob10_15 IS
    GENERIC ( N : INTEGER := 8 ) ;
    PORT ( Clock, Resetn : IN STD_LOGIC ;
            Data : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
            RAdd : IN INTEGER RANGE 0 TO 3 ;
            s, ER : IN STD_LOGIC ;
            M : BUFFER STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
            Done : OUT STD_LOGIC ) ;
END prob10_15 ;

ARCHITECTURE Behavior OF prob10_15 IS
    COMPONENT divider
        GENERIC ( N : INTEGER := 8 ) ;
        PORT ( Clock, Resetn : IN STD_LOGIC ;
                s, LA, EB : IN STD_LOGIC ;
                DataA, DataB : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                R, Q : BUFFER STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                Op1, Op2 : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                Result : OUT STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                Done : OUT STD_LOGIC ) ;
    END COMPONENT ;
    TYPE State_type IS ( S1, S2, S3, S4, S5 ) ;
    SIGNAL y : State_type ;
    SIGNAL Dec_RAdd, Rin : STD_LOGIC_VECTOR(0 TO 3) ;
    TYPE RegArray IS
        ARRAY(0 TO 3) OF STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
    SIGNAL R : RegArray ;
    SIGNAL Ri, SR, Sin : STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
    SIGNAL Sum, Remainder, K : STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
    SIGNAL ES, Ssel, EC, LC, z : STD_LOGIC ;

    ... con't
SIGNAL C : INTEGER RANGE 0 TO 3;
SIGNAL LA, EB, zz, Div : STD_LOGIC;
BEGIN
  FSM_tran: PROCESS ( Resetn, Clock )
  BEGIN
    IF Resetn = '0' THEN
      y <= S1;
    ELSIF (Clock'EVENT AND Clock = '1') THEN
      CASE y IS
        WHEN S1 => IF s = '0' THEN y <= S1 ELSE y <= S2; END IF;
        WHEN S2 => IF z = '1' THEN y <= S3 ELSE y <= S2; END IF;
        WHEN S3 => y <= S4;
        WHEN S4 => IF zz='0' THEN y <= S4 ELSE y <= S5; END IF;
        WHEN S5 => IF s = '1' THEN y <= S5 ELSE y <= S1; END IF;
      END CASE;
    END IF;
  END PROCESS;
  FSM_outputs: PROCESS ( y, z )
  BEGIN
    EC <= '0'; LC <= '0'; Ssel <= '0'; ES <= '0'; Done <= '0';
    LA <= '0'; EB <= '0'; Div <= '0';
    CASE y IS
      WHEN S1 =>
        EC <= '1'; LC <= '1'; ES <= '1';
      WHEN S2 =>
        ES <= '1'; Ssel <= '1';
        IF z='0' THEN EC <= '1' ELSE EC <= '0'; END IF;
      WHEN S3 =>
        LA <= '1'; EB <= '1';
      WHEN S4 =>
        Div <= '1';
      WHEN S5 =>
        Div <= '1'; Done <= '1';
    END CASE;
  END PROCESS;
  -- 2-to-4 decoder
  WITH RAdd SELECT
    Dec_RAdd <= "1000" WHEN 0,
               "0100" WHEN 1,
               "0010" WHEN 2,
               "0001" WHEN OTHERS ;
  Rin <= Dec_RAdd WHEN ER = '1' ELSE "0000";
  GenReg: FOR i IN 0 TO 3 GENERATE
    Reg: regne GENERIC MAP ( N => N )
      PORT MAP ( Data, Resetn, Rin(i), Clock, R(i) );
  END GENERATE;

  ...con't
- - downcnt is defined in Figure 7.55
Count: downcnt GENERIC MAP ( modulus => 4 )
       PORT MAP ( Clock, EC, LC, C ) ;
z <= '1' WHEN C = 0 ELSE '0' ; - - NOR GATE
Sin <= Sum WHEN Ssel = '1' ELSE (OTHERS => '0') ;
RegS: regne GENERIC MAP ( N => N )
       PORT MAP ( Sin, Resetn, ES, Clock, SR ) ;
- - Mux Regs
WITH C SELECT
   Ri <= R(0) WHEN 0, R(1) WHEN 1, R(2) WHEN 2, R(3) WHEN OTHERS ;
- - Sum <= SR + Ri (see problem 10.13);
K <= "00000100" ;
DivideBy4: divider
       PORT MAP ( Clock => Clock, Resetn => Resetn, s => Div, LA => LA,
                   EB => EB, DataA => SR, DataB => K, R => Remainder, Q => M,
                   Op1 => SR, Op2 => Ri, Result => Sum, Done => zz ) ;
END Behavior ;

10.15. The modified pseudo-code is

for i = 0 to k - 2 do
   A = Ri ;
   for j = i + 1 to k - 1 do
      if Rj < A then
         Ri = Rj ;
         Rj = A ;
      end if ;
   end for ;
   A = Ri ;
end for ;

An ASM chart that corresponds to the pseudo-code is
From the ASM chart, we can see that the datapath circuit needs a multiplexer to allow the operation $R_i \leftarrow R_j$. An appropriate datapath is shown below.

An ASM chart for the control circuit is
ENTITY prob10_17 IS
  GENERIC ( N : INTEGER := 4 ; LOG2K : INTEGER := 2 );
  PORT ( Clock, Resetn : IN STD_LOGIC ;
  s, WrInit, Rd : IN STD_LOGIC ;
  DataIn : IN STD_LOGIC_Vector(N-1 DOWNTO 0) ;
  RAdd : IN INTEGER RANGE 0 TO 3 ;
  DataOut : BUFFER STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
  Done : BUFFER STD_LOGIC ) ;
END prob10_17 ;

ARCHITECTURE Behavior of prob10_17 IS
  TYPE State_type IS ( S1, S2, S3, S4, S5, S6, S7, S8 ) ;
  SIGNAL y : State_type ;
  SIGNAL Ci, Cj : INTEGER RANGE 0 to 3 ;
  SIGNAL Rin : STD_LOGIC_VECTOR(0 TO 3) ;
  TYPE RegArray IS
    ARRAY(0 TO 3) OF STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
  SIGNAL R : RegArray ;
  SIGNAL RData : STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
  SIGNAL Int, Csel, Wr : STD_LOGIC ;
  SIGNAL Cmux, Imux : INTEGER RANGE 0 TO 3 ;
  SIGNAL LI, LJ, EI, EJ, zi, zj, Ain, Rjout : STD_LOGIC ;
  SIGNAL Zero : INTEGER RANGE 0 TO 3 ;
  SIGNAL A, AData : STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
  SIGNAL ARjMux, Rj : STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
BEGIN
  FSM.transitions: PROCESS ( Resetn, Clock )
  BEGIN
    IF Resetn = '0' THEN
      y <= S1 ;
    ELSIF ( Clock'EVENT AND Clock = '1') THEN
      CASE y IS
        WHEN S1 => IF s = '0' THEN y <= S1 ; ELSE y <= S2 ; END IF ;
        WHEN S2 => y <= S3 ;
        WHEN S3 => y <= S4 ;
        WHEN S4 => IF RjtA = '1' THEN y <= S5 ; ELSE y <= S7 ; END IF ;
        WHEN S5 => y <= S6 ;
        WHEN S6 => y <= S7 ;
        WHEN S7 =>
          IF zj = '0' THEN y <= S4 ;
          ELSIF zi = '0' THEN y <= S2 ;
          ELSE y <= S8 ;
          END IF ;
    ... con't
WHEN S8 =>
  IF s = '1' THEN y <= S8 ; ELSE y <= S1 ; END IF ;
END CASE ;
END IF ;
END PROCESS ;
Int <= '0' WHEN y = S1 ELSE '1' ;
Done <= '1' WHEN y = S8 ELSE '0' ;
FSM_outputs: PROCESS ( y, zi, zj )
BEGIN
  LI <= '0' ; LJ <= '0' ; EI <= '0' ; EJ <= '0' ; Csel <= '0' ;
  Wr <= '0' ; Ain <= '0' ; RjOut <= '0' ;
CASE y IS
  WHEN S1 => LI <= '1' ; EI <= '1' ;
  WHEN S2 => Ain <= '1' ; LJ <= '1' ; EJ <= '1' ;
  WHEN S3 => EJ <= '1' ;
  WHEN S4 => - - nothing to do
  WHEN S5 => Wr <= '1' ; Rjout <= '1' ;
  WHEN S6 => Csel <= '1' ; Wr <= '1' ;
  WHEN S7 => Ain <= '1' ;
    IF zj = '0' THEN EJ <= '1' ;
    ELSE EJ <= '0' ;
    IF zi = '0' THEN EI <= '1' ;
    ELSE EI <= '0' ;
    END IF ;
  END IF ;
  WHEN S8 => --
END CASE ;
END PROCESS ;
Zero <= 0 ;
GenReg: FOR i IN 0 TO 3 GENERATE
  Reg: regne GENERIC MAP ( N => N )
    PORT MAP ( RData, Resetn, Rin(i), Clock, R(i) ) ;
END GENERATE ;
RegA: regne GENERIC MAP ( N => N )
  PORT MAP ( AData, Resetn, Ain, Clock, A ) ;
-- New DataPath
WITH Cj SELECT
  Rj <= R(0) WHEN 0 ,
    R(1) WHEN 1 ,
    R(2) WHEN 2 ,
    R(3) WHEN OTHERS :
ARjMux <= Rj WHEN Rjout='1' ELSE A ;
RData <= ARjMux WHEN WRInit = '0' ELSE DataIn ;
RjltA <= '1' WHEN Rj < A ELSE '0' ;
-- End of New DataPath
OuterLoop: upcount GENERIC MAP ( modulus => 4 )
  PORT MAP ( Resetn, Clock, EI, LI, Zero, Ci ) ;
InnerLoop: upcount GENERIC MAP ( modulus => 4 )
  PORT MAP ( Resetn, Clock, EJ, LJ, Ci, Cj ) ;

... con't
Cmux <= Ci WHEN Csel = '0' ELSE Cj;
Imux <= Cmux WHEN Int = '1' ELSE Radd;
WITH Imux SELECT
  AData <= R(0) WHEN 0,
          R(1) WHEN 1,
          R(2) WHEN 2,
          R(3) WHEN OTHERS;
RinDec: PROCESS (WrInit, Wr, Imux)
BEGIN
  IF (WrInit OR Wr) = '1' THEN
    CASE Imux IS
      WHEN 0 => Rin <= "1000";
      WHEN 1 => Rin <= "0100";
      WHEN 2 => Rin <= "0010";
      WHEN OTHERS => Rin <= "0001";
    END CASE;
  ELSE Rin <= "0000";
  END IF;
END PROCESS;
zi <= '1' WHEN Ci = 2 ELSE '0';
zj <= '1' WHEN Cj = 3 ELSE '0';
DataOut <= (OTHERS => 'Z') WHEN Rd='0' ELSE AData;
END Behavior;

10.18.

```
10-34
```
10.19. (a) An ASM chart for the control circuit is shown below.

(b) LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE work.components.all;

ENTITY prob10_19 IS
GENERIC ( N : INTEGER := 8 ) ;
PORT ( Data : IN STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
   Resetn, w : IN STD_LOGIC ;
   Clock : IN STD_LOGIC ;
   RinExt : IN STD_LOGIC_VECTOR(1 TO 3);
   BusWires : BUFFER STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END prob10_19 ;

ARCHITECTURE Behavior OF prob10_19 IS
   TYPE State_type IS ( S1, S2, S3, S4 ) ;
   SIGNAL y : State_type ;
   SIGNAL RinCtrl, Rin, Rout : STD_LOGIC_VECTOR(1 TO 3) ;
   SIGNAL R1, R2, R3 : STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
   . . . con’t
BEGIN
FSM transitions: PROCESS (Resetn, Clock)
BEGIN
  IF Resetn = '0' THEN
    y <= S1;
  ELSIF (Clock'EVENT AND Clock = '1') THEN
    CASE y IS
      WHEN S1 => IF w = '0' THEN y <= S1; ELSE y <= S2; END IF;
      WHEN S2 => y <= S3;
      WHEN S3 => y <= S4;
      WHEN S4 => y <= S1;
    END CASE;
  END IF;
END PROCESS;
FSM outputs: PROCESS (y)
BEGIN
  RinCtrl <= "000" ; Rout <= "000";
  CASE y IS
    WHEN S1 =>
    WHEN S2 => Rout(1) <= '1' ; RinCtrl(3) <= '1';
    WHEN S3 => Rout(2) <= '1' ; RinCtrl(1) <= '1';
    WHEN S4 => Rout(3) <= '1' ; RinCtrl(2) <= '1';
  END CASE;
END PROCESS;
Reg1: regne GENERIC MAP (N => N)
  PORT MAP (BusWires, Resetn, Rin(1), Clock, R1);
Reg2: regne GENERIC MAP (N => N)
  PORT MAP (BusWires, Resetn, Rin(2), Clock, R2);
Reg3: regne GENERIC MAP (N => N)
  PORT MAP (BusWires, Resetn, Rin(3), Clock, R3);
Rin <= RinCtrl OR RinExt;
  -- Mux
  WITH Rout SELECT
    BusWires <= R1 WHEN "100",
               R2 WHEN "010",
               R3 WHEN "001",
               Data WHEN OTHERS;
END Behavior;

10.20. An ASM chart for the processor is shown below.
Load function register

T0

I = 0

I = 1

T1

R_X ← Data,Done

R_X ← R_Y,Done

A ← R_X

T2

G ← A + R_Y

G ← A - R_Y

R_X ← G,Done

T3

I = 0

I = 1

I = 2

Reset
10.21. (a) An ASM chart for the control circuit is shown below.

```
(a) An ASM chart for the control circuit is shown below.

10-38
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;
USE work.components.all;

ENTITY prob10_21 IS
  PORT ( Data : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
          Clock, Resetn, w : IN STD_LOGIC;
          F, Rx, Ry : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
          Done : OUT STD_LOGIC;
          BusWires : INOUT STD_LOGIC_VECTOR(7 DOWNTO 0) );
END prob10_21;

ARCHITECTURE Behavior OF prob10_21 IS
  TYPE State_Type IS ( T0, T1, T2, T3 );
  SIGNAL T : State_Type;
  SIGNAL Gin, Gout, Extern : STD_LOGIC;
  SIGNAL High, FRin, AddSub, Ain : STD_LOGIC;
  SIGNAL Rin, Rout, X, Y : STD_LOGIC_VECTOR(0 TO 3);
  SIGNAL I : STD_LOGIC_VECTOR(1 DOWNTO 0);
  SIGNAL R0, R1, R2, R3, G, A, Sum : STD_LOGIC_VECTOR(7 DOWNTO 0);
  SIGNAL Func, FuncReg, Sel : STD_LOGIC_VECTOR(1 TO 6);

BEGIN
  FSM_Transitions: PROCESS ( Clock, Resetn )
  BEGIN
    IF Resetn = '0' THEN
      T <= T0 ;
    ELSIF Clock'EVENT AND Clock = '1' THEN
      CASE T IS
        WHEN T0 =>
          IF w = '0' THEN T <= T0 ; ELSE T <= T1 ; END IF ;
        WHEN T1 =>
          IF I = "00" OR I = "01" THEN T <= T0 ; ELSE T <= T2 ; END IF ;
        WHEN T2 =>
          T <= T3 ;
        WHEN T3 =>
          T <= T0 ;
      END CASE ;
    END IF ;
  END PROCESS ;

  FSM_Outputs: PROCESS ( w, T, I )
  BEGIN
    FRin <= '0' ; Rin <= "0000" ; Rout <= "0000" ; Done <= '0' ;
    Gin <= '0' ; Gout <= '0' ; Extern <= '0' ;
    Ain <= '0' ; AddSub <= '0' ;
    ... con't
CASE T IS
  WHEN T0 =>
    IF w = '1' THEN FRin <= '1';
    ELSE FRin <= '0';
    END IF;
  WHEN T1 =>
    Ain <= '1'; - doesn’t matter if we load A when not needed
    IF I = "00" THEN
      Done <= '1'; Rin <= X; Rout <= "0000"; Extern <= '1';
      ELSIF I = "01" THEN
        Done <= '1'; Rin <= X; Rout <= Y; Extern <= '0';
        ELSE
          Done <= '0'; Rin <= "0000"; Rout <= X; Extern <= '0';
      END IF;
  WHEN T2 =>
    Gin <= '1'; Rout <= Y;
    IF I = "10" THEN AddSub <= '0';
    ELSE AddSub <= '1';
    END IF;
  WHEN T3 =>
    Gout <= '1'; Rin <= X; Done <= '1';
END CASE;
END PROCESS;
- - Datapath is the same as in Figure 7.75, without the T counter
  Func <= F & Rx & Ry;
  functionreg: regne GENERIC MAP ( N => 6 )
    PORT MAP ( Func, Resetn, FRin, Clock, FuncReg );
  High <= '1';
  I <= FuncReg(1 TO 2);
  decX: dec2to4 PORT MAP ( FuncReg(3 TO 4), High, X );
  decY: dec2to4 PORT MAP ( FuncReg(5 TO 6), High, Y );
- - Note: changed GENERIC parameter N in regne that sets # bits to 8
  reg0: regne PORT MAP ( BusWires, Resetn, Rin(0), Clock, R0 );
  reg1: regne PORT MAP ( BusWires, Resetn, Rin(1), Clock, R1 );
  reg2: regne PORT MAP ( BusWires, Resetn, Rin(2), Clock, R2 );
  reg3: regne PORT MAP ( BusWires, Resetn, Rin(3), Clock, R3 );
  regA: regne PORT MAP ( BusWires, Resetn, Ain, Clock, A );
  alu: WITH AddSub SELECT
    Sum <= A + BusWires WHEN '0',
    A – BusWires WHEN OTHERS;
  regG: regne PORT MAP ( Sum, Resetn, Gin, Clock, G );
  Sel <= Rout & Gout & Extern;
  WITH Sel SELECT
    BusWires <= R0 WHEN "100000",
    R1 WHEN "010000",
    R2 WHEN "001000",
    R3 WHEN "000100",
    G WHEN "000010",
    Data WHEN OTHERS;
END Behavior;
10.22. (a) An ASM chart for the traffic controller is shown below.

(b). The two counters, $C_1$ and $C_2$, each require clock enable and parallel-load inputs. Assuming that the clock enables signals are called $EC_1$ and $EC_2$ and the parallel-load inputs are called $LC_1$ and $LC_2$, an ASM chart for the control circuit is
ENTITY prob10_22 IS
    PORT ( Clock, Resetn : IN STD_LOGIC;
            G1, Y1, R1 : OUT STD_LOGIC;
            G2, Y2, R2 : OUT STD_LOGIC );
END prob10_22 ;

... con't
ARCHITECTURE Behavior OF prob10_22 IS
  TYPE State_type IS ( S1, S2, S3, S4, S5 ) :
  SIGNAL y : State_type :
  SIGNAL zC1, zC2, EC1, LC1, EC2, LC2 : STD_LOGIC :
  SIGNAL C1, C2, TICKS1, TICKS2 : STD_LOGIC_VECTOR(3 DOWNTO 0 ) :
BEGIN
  FSM_transition: PROCESS ( Clock, Resetn )
  BEGIN
    IF Resetn = '0' THEN
      y <= S1 ;
    ELSIF Clock'EVENT AND Clock = '1' THEN
      CASE y IS
        WHEN S1 => y <= S2 ;
        WHEN S2 => IF zC1 = '0' THEN y <= S2 ; ELSE y <= S3 ; END IF ;
        WHEN S3 => IF zC2 = '0' THEN y <= S3 ; ELSE y <= S4 ; END IF ;
        WHEN S4 => IF zC1 = '0' THEN y <= S4 ; ELSE y <= S5 ; END IF ;
        WHEN S5 => IF zC2 = '0' THEN y <= S5 ; ELSE y <= S2 ; END IF ;
      END CASE ;
    END IF ;
  END PROCESS ;
  FSM_outputs: PROCESS ( y, zC1, zC2 )
  BEGIN
    G1 <= '0' ; Y1 <= '0' ; R1 <= '0' ;
    G2 <= '0' ; Y2 <= '0' ;
    LC1 <= '0' ; EC1 <= '0' ; EC2 <= '0' ;
    CASE y IS
      WHEN S1 =>
        LC1 <= '1' ; EC1 <= '1' ;
      WHEN S2 =>
        EC1 <= '1' ; G1 <= '1' ; R2 <= '1' ;
        IF zC1 = '1' THEN LC2 <= '1' ; EC2 <= '1' ; ELSE LC2 <= '0' ; EC2 <= '0' ; END IF ;
      WHEN S3 =>
        EC2 <= '1' ; Y1 <= '1' ; R2 <= '1' ;
        IF zC2 = '1' THEN LC1 <= '1' ; EC1 <= '1' ; ELSE LC1 <= '0' ; EC1 <= '0' ; END IF ;
      WHEN S4 =>
        EC1 <= '1' ; R1 <= '1' ; G2 <= '1' ;
        IF zC1 = '1' THEN LC2 <= '1' ; EC2 <= '1' ; ELSE LC2 <= '0' ; EC2 <= '0' ; END IF ;
      WHEN S5 =>
        EC2 <= '1' ; R1 <= '1' ; Y2 <= '1' ;
        IF zC2 = '1' THEN LC1 <= '1' ; EC1 <= '1' ; ELSE LC1 <= '0' ; EC1 <= '0' ; END IF ;
    END CASE ;
  END PROCESS ;
  -- Setup Constants
  TICKS1 <= "0011" ; -- 4 TICKS for C1
  TICKS2 <= "0001" ; -- 2 TICKS for C2
  -- Check for zero count
  zC1 <= '1' WHEN C1 = 0 ELSE '0' ;
  zC2 <= '1' WHEN C2 = 0 ELSE '0' ;
  ...
... con't
- - Could use downcnt component instead
CNT_1: PROCESS
BEGIN
  WAIT UNTIL Clock’EVENT AND Clock = ’1’;
  IF EC1 = ’1’ THEN
    IF LC1 = ’1’ THEN C1 <= TICKS1 ; ELSE C1 <= C1 - 1; END IF;
  END IF;
END PROCESS;
CNT_2: PROCESS
BEGIN
  WAIT UNTIL Clock’EVENT AND Clock = ’1’;
  IF EC2 = ’1’ THEN
    IF LC2 = ’1’ THEN C2 <= TICKS2 ; ELSE C2 <= C2 - 1; END IF;
  END IF;
END PROCESS;
END Behavior;

10.23. The debounce circuit has three parts, as shown below. The Data signal from the switch has to be synchronized to the 102.4 KHz signal using two flip-flops. The synchronized signal called Sync is fed to an FSM. The FSM also uses the counter shown, which counts for 1024 cycles of the 102.4 KHz signal, providing a 10 msec delay.

An ASM chart for the FSM is given below. The FSM provides the z output, which is the debounced version of the Data signal.
10.24. (a) If we set $C_1 = 1$ pF, then $R_a = 0$ and $R_b = 1.43 \, k\Omega$

(b) If we set $C_1 = 1$ pF, then $R_a = 1.42 \, k\Omega$ and $R_b = 0.71 \, k\Omega$