The University of Queensland
School of Information Technology and Electrical Engineering

Design of DC Motor Controllers for a Humanoid Robot

Written by
JARAD HEATH STIRZAKER
Bachelor of Electrical Engineering
October 19th 2001

25 Laurier St
Annerley QLD 4103
19th October 2001

Head of School
Information Technology and Electrical Engineering
University of Queensland
St. Lucia QLD 4072

Dear Professor Kaplan

In accordance with the requirements of the degree of Bachelor of Engineering (Honours) in the division of Electrical Engineering, I present the following thesis entitled "Design of Joint Controllers for a Humanoid Robot". This thesis project was completed under the supervision of Dr Gordon Wyeth.

I declare that all work submitted in this thesis is my own, except where acknowledged by references and footnotes. This work, to the best of my knowledge, has not been previously submitted for a degree at The University of Queensland or any other institution.

Yours faithfully

Jarad Stirzaker

Design of an Autonomous Humanoid Robot Paper

The following paper, Design of an Autonomous Humanoid Robot, by Wyeth, Kee, Wagstaff et al. was accepted at the Australian Conference on Robotics and Automation, Sydney 2001.

Design of an Autonomous Humanoid Robot

Gordon Wyeth, Damien Kee, Mark Wagstaff, Nathaniel Brewer,
Jared Stirzaker, Timothy Cartwright, Bartek Bebel
School of Computer Science and Electrical Engineering
University of Queensland
St. Lucia, Queensland, 4072
Australia

Abstract

This paper describes the design of an autonomous humanoid robot. The robot itself is currently under construction, however the process of designing the robot has revealed much about the considerations for creating a robot with humanoid shape. The mechanical design is a complete CAD solids model, with specific motors and transmission systems selected. The electronic design of a distributed control system is also complete, along with the electronics for power and sensor processing. A high fidelity graphical simulator has been developed, providing important early feedback on critical design decisions.

1 Introduction

There are several reasons to build a robot with humanoid form. It has been argued that to build a machine with human like intelligence, it must be embodied in a human like body. Others argue that for humans to interact naturally with a robot, it will be easier for the humans if that robot has humanoid form. A third, and perhaps more concrete, reason for building a humanoid robot is to develop a machine that interacts naturally with human spaces. The architectural constraints on our working and living environments are based on the form and dimensions of the human body. Consider the design of stairs, cupboards and chairs; the dimensions of doorways, corridors and benches. A robot that lives and works with humans in an unmodified environment must have a form that can function with everyday objects. The only form that is guaranteed to work in all cases is the form of humanoid.

1.1 The *GuRoo* Project

The *GuRoo* project in the University of Queensland Robotics Laboratory aims to design and build a 1.2m tall robot with human proportions that is capable of balancing, walking, turning, crouching, and standing from a prostrate position. The target mass for the robot is 30 kg, including on-board power and computation. The robot will have active, monocular, colour vision and vision processing.

The intended challenge task for the robot is to play a game of soccer with or against human players or other humanoid robots. To complete this challenge, the robot must be able to move freely on its two legs. It requires a vision sense that can detect the objects in a soccer game, such as the ball, the players from both teams, the goals and the boundaries. It must also be able to manipulate and kick a ball with its feet, and be robust enough to deal with legal challenges from human players. Clearly, the robot must operate in a completely autonomous fashion without support harnesses or wiring tethers.

These goals are yet to be realised for the GuRoo project. Currently the robot exists as a complete mechanical CAD model (see Figure 1), a complete electronic model and a high fidelity dynamic simulation. The dynamic simulation has been programmed to crouch, jump and balance. The progress to this stage has revealed much about the design considerations for a humanoid robot.

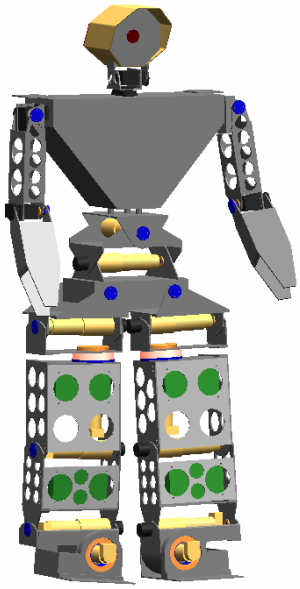


Figure 1: Full CAD model of the GuRoo humanoid robot.

1.2 Paper Overview

This section has described the motivation for building a humanoid robot, and the specific challenge that has been set for the GuRoo project. The subsequent section will look at other humanoid robot projects, including bipedal walking robots.

The rest of the paper describes the mechanical, electronic and software design of the GuRoo robot. In

particular, the paper will detail the mechanical model of the robot and a comparison to the human form, the motors and sensors, the complete electronic design, a full dynamic software simulation of the robot, the software architecture of the robot, and results for balancing and crouching in simulation.

2 Prior Art

2.1 Bipedal Walking Robots

Research into bipedal walking robots can be split into two categories: *active* and *passive*. The passive or un-powered category (for example, McGeer's passive dynamic walker [McGeer, 1990]) is of interest as it illustrates that walking is fundamentally a dynamic problem. Passive walkers do not require actuators, sensors, or computers in order to make them move, but walk down gentle slopes generating motion by the hardware geometry. The passive walkers also illustrate the walking can be performed with very little power input.

Active walkers can further be split into two categories; those that employ the natural dynamics of specialised actuators, and those that are fully power operated. Raibert [Raibert, 1986] and later Pratt [Pratt, 1998] have shown some impressive feats of walking and gymnastic ability in robots that have the capacity for energy storage in the actuator. These robots have been shown to have robust and stable performance from relatively simple control mechanisms.

The alternate approach is to control the joints through pre-specified trajectories to a known "good" gait pattern (for example, [Golden, 1990]). This is a simple approach, but lacks robustness to disturbances. This approach becomes more complex when additional layers are added to provide adjustments to the gait for disturbance. Controlling a fully powered biped in a manner that depends on the dynamic model is complicated by the complex dynamic equations for the robot's motion. Yamaguchi et al. [Yamaguchi, 1998] moved a dynamic torso with significant mass through 2 DOF to keep the Zero Moment Point (ZMP) within the polygon of the support foot. This approach contributed to successful control of the robot, but produces an awkward gait.

2.2 Bipedal Walking Humanoid Robots

There are few examples of autonomous biped walkers that resemble the structure of a human. The Honda company biped robots, P2 and P3 are two of the few examples of such robots [Hirai, 1998]. P3 can walk on level ground, walk up and down stairs, turn, balance, and push objects. The robot is completely electrically and mechanically autonomous. The Sony SDR-3X robot is another example with similar capabilities, although details of the design are yet to be published.

3 Mechanics

The mechanical design of the humanoid requires careful and complex tradeoffs between form, function, power, weight, cost and manufacturability. For example, in terms of form, the robot should conform to the proportions of a 1.2m tall human. However, retaining the exact proportions compromises the design in terms of the selection of actuation and mechanical power transmission systems. Affordable motors that conform to the dimensional restrictions have insufficient power for the robot to walk or crouch. This section describes the final mechanical design and how the balance between conflicting design requirements has been achieved.

3.1 Proportions

The target proportions for the robot are based on biomechanical data of the human form. Figure 2 shows the proportions of the frontal plane dimensions of a 50th percentile male based on data from a United States survey [Dempster, 1965]. The dimensions shown in millimetres indicate the appropriate sizes of anatomical features when scaled to a total height of 1200 mm against the comparable dimensions on GuRoo.

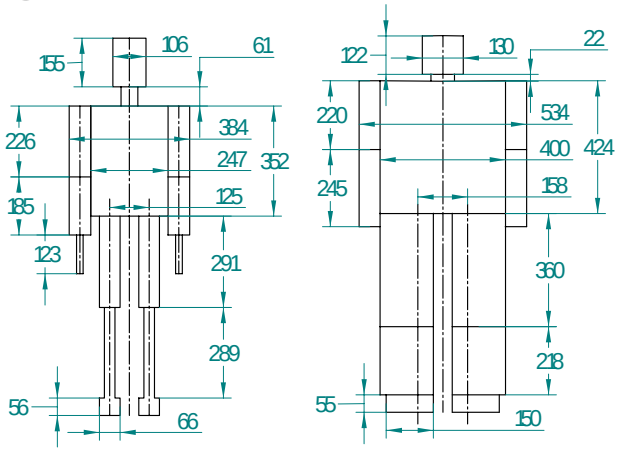


Figure 2: The proportions of typical human anatomy compared to the matching proportions of GuRoo's anatomy. The dimensions indicate the sizes for a human scaled to 1.2m in height.

By comparison, GuRoo is somewhat thickset in the legs, as was dictated by the form of the chosen actuators (see Section 3.3). The spacing between the hips and ankles has been retained, rather than placing the hips and ankles along the frontal centreline of each leg. Our simulation studies showed that the required torques around the roll axes of the hips and ankles becomes excessive if the hips and ankles are spaced too far apart (see Section 5.3).

The body and upper leg of GuRoo are somewhat longer than the counterparts in the human model. This is due to the chain of actuators required for three degrees of freedom in the waist and hips respectively (see Section 3.2). Consequently, the lower leg and the neck and head are shorter to compensate. The overall effect is still convincingly human-like in shape.

The changes in volume required to house the actuators, as well as the mass of the actuators themselves have an effect on the mass distribution. Table 1 shows the mass distribution of GuRoo compared to that of a human. The most notable exception is that the shin and foot are much heavier in GuRoo than the human counterpart, due to the mass of the powerful actuators required in the ankle. The arms are significantly lighter than the human

counterpart, as they are significantly inferior in power and do not have hands. GuRoo's mass distribution is closer to the human distribution than either MIT's active bipedal walker [Paluska, 2000], or McGeer's passive dynamic bipedal walker.

Table 1: Comparison of GuRoo mass distribution with human mass distribution, and with the mass distribution of MIT's M2 bipedal walker and McGeer's passive dynamic walker.

Body Component	GuRoo mass (kg)	GuRoo	Human	M2	PDW
Head and Upper torso	7.3	24%	31%	0%	0%
Abdomen and Hips	9.1	30%	27%	51%	50%
Thigh	5.8	19%	20%	22%	30%
Shin and Foot	6.4	21%	12%	27%	20%
Arm	1.9	6%	10%	0%	0%
Total	30.5				

The other notable point from Table 1 is the total mass of the robot. A 1.2 m tall human would typically be a child approaching his or her 7th birthday, with a 50th percentile mass of 23 kg. A child with mass of 30.5 kg at the same age would be in 97th percentile, indicating that GuRoo is somewhat overweight.

3.2 Architecture

The extent to which human joint function can be replicated is another key factor in robot design. Figure 3 shows the degrees of freedom contained in each joint area of the robot. In the cases where there are multiple degrees of freedom (for example, the hip) the joints are implemented sequentially through short links rather than as spherical joints. Other key differences to the human form are the lack of a continuous flexible spine, and the lack of a yaw axis in the ankle. Another point to note is that the roll and pitch axes of the ankle are orthogonal, whereas the human ankle has an angle of about 64° between the roll and pitch axes.

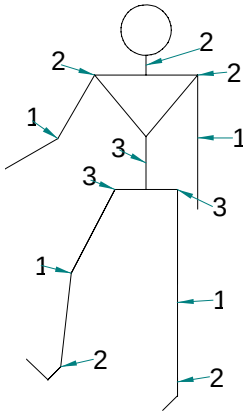


Figure 3: The location of the joints in GuRoo, indicating the degrees of freedom in each joint.

3.3 Motor Choice

The key element in driving the mechanical design has been the choice of actuator. The robot has 23 joints in total. The legs and abdomen contain 15 joints that are required to produce significant mechanical power, most generally with large torques and relatively low speeds. The other 8 joints drive the head and neck assembly, and the arms. The torque and speed requirements are significantly less. Factors of cost, weight and availability limited the choice of actuators to rotary DC motors

The 15 high power joints all use the same motor-gearbox combination. The motor is a Maxon RE 36 wound for a nominal voltage of 32V. This motor can provide 88.5 mNm of torque continuously, with a matching current consumption of 1.99 A. The motor has a maximum permissible speed of 8200 RPM. The gearbox has a reduction of 156, with an efficiency of 72%. The maximum continuous generated output torque is 10 Nm, with a maximum output speed of 51 RPM, or 5.3 rad/s. The thermal limits of the motor permit intermittent output torque of up to 19Nm. Each motor is fitted with an optical encoder for position and velocity feedback. The total mass of the motor/gearbox/encoder unit is 0.85 kg.

The 8 low power joints are Hi-Tec RC servo motors model HS705-MG. These motors have an integrated gearbox and have rated output torque to 1.4 Nm, at speeds of 5.2 rad/s. These also have potentiometer feedback and built-in control and power electronics. They require 6V power, and a pulse width modulated signal to indicate desired position. The mass of each unit is 0.125 kg.

4 Electronics

A distributed control network controls the robot, with a central computing hub that sets the goals for the robot, processes the sensor information, and provides coordination targets for the joints. The joints have their own control processors that act in groups to maintain global stability, while also operating individually to provide local motor control. The distributed system is connected by a CAN network. In addition, the robot requires various sensor amplifiers and power conversion circuits.

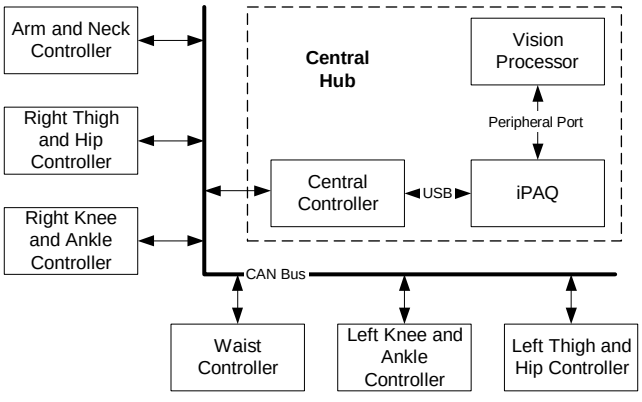


Figure 4: Block diagram of the distributed control system.

4.1 Computing

4.1.1 Central Hub

The central control of the robot derives from a hub of three heterogeneous microprocessors that provide coordination between joints, integrate sensor information, and process the vision input. This hub also provides communication to the outside world through user interfaces and communication peripherals.

The primary component of the central controller is an iPAQ pocket pc from Compaq. The iPAQ features a 208 MHz StrongARM microcontroller, 32 Mb of RAM and a 320 x 240 colour screen. The screen is touch sensitive allowing stylus input of text and graphics. The iPAQ has 16 Mb of Flash ROM to store the operating system. The iPAQ in the GuRoo operates with Windows CE. As well as the touch screen interface, the iPAQ is equipped with a speaker and microphone, a joypad, and four push-buttons. It has an infra-red interface for external communication.

The second component of the central hub is a TMS320F243 microcontroller that acts as an adapter and filter for the robot's internal CAN network (see Section 4.1.3). The microcontroller communicates with the robot's distributed control system through the CAN network, and to the iPAQ through the iPAQ's USB serial communication port. The microcontroller also manages the power supply (see Section 4.2.3) providing centralised control of the robot power supply in the event of system failure. This microcontroller is the same device used in the joint controllers (see Section 4.1.2).

The final component of the central is the vision processing board. This board has been developed for the ViperRoos robot soccer team [Chang, 2001] and features a 200 MHz Hitachi Super-H SH4 microcontroller, an FPGA-based programmable camera and bus adapter, 16 Mb of RAM, 8 Mb of flash ROM, and 512 kb of fast SRAM for video caching. The board interfaces to the 100 pin parallel peripheral bus on the iPAQ to provide real time visual display on the iPAQ's colour screen. The vision input comes from a custom digital CMOS camera, based around the OV7620 camera chip from OmniVision, which can provide 640 x 480 images at up to 25 fps. The camera can provide data in YUV or RGB formats, and can be programmed to only send data from selected areas of the sense region.

4.1.2 Joint Controllers

The TMS320F24x series is a 32 bit DSP designed for motor control. The availability of the Control Area Network (CAN) module in this series, along with bootloader programmable internal Flash memory makes the device particularly attractive for this application. Furthermore the device features 8k words of internal flash memory, 8 PWM channels with deadband generation, quadrature input circuitry, an 8 channel 10 bit analog to digital converter with a conversion time of 800ns, a power drive protection external interrupt, and a 50ns instruction time. The TMS320F241 from Texas Instruments operates at 20MHz, and can read the A/D converter, calculating a PID control law, current limit, and generate the required PWM output, in under 10 μs [Wyeth, 2001]. In this application, we use the TMS320F243, which has an external bus that is used for attaching additional sensor interfaces. Five controller boards control the 15 high power motors, each board controlling three motors. A sixth controller board controls the eight RC servo motors.

4.1.3 Internal Network

The CAN bus is a highly reliable standard developed by Robert Bosch GmbH for use in the automotive environment. It is a multi-master system, with sophisticated error checking and arbitration, so that any high priority message will always get through first without corruption by other messages. All data contained in each packet (up to eight bytes) is also checked with a Cyclic Redundancy Check (CRC) error-checking scheme that can correct up to five random errors, and will be automatically retransmitted if not correct. The network operates at up to 1 Mbit/sec.

4.2 Power

4.2.1 Drive Power Electronics

The drive power electronics is based on a switch mode power stage, requiring only a single supply rail and having an efficiency over 90%. This efficiency results in several advantages such as small size, lower cost power devices and less heatsinking. The H-Bridge channels are driven from separate PWM outputs of the DSP, allowing the deadband features of the PWM peripheral to be used, along with the immediate (<12ns) shutdown of these pins in the event of a fault which triggers the Power Drive Protect Interrupt (PDPInt) pin on the DSP.

A integrated solution was chosen for this design – the SGS-Thomson L6203. This device uses low on-resistance and fast switching MOSFETs, to give maximum efficiency and best control. The voltage limit of the devices is 48V, and the total continuous RMS current limit is 4A. This is a good match to the chosen motors and batteries. The total on-resistance of the power devices is 0.3Ω. The cost of the device is low, compared to a discrete solution, and the volume and mass of the electronics is minimised by the choice of an integrated solution.

4.2.2 Battery Packs

The power for the 15 high power motors is provided by 4 x 1.5Ah 42V NiCd packs. These packs are effectively paralleled to a common bus (see Section 4.2.3). The packs are chosen to give 20 minutes of continuous operation. The power for the 8 low power motors is derived from a single 3Ah 7.2 V NiCd battery pack. The power for the control electronics is derived from a second single 3Ah 7.2V NiCd pack. The voltage from this pack is distributed to the various boards that require power where it is regulated locally.

4.2.3 Power Regulation

Connecting NiCd batteries in parallel can be extremely hazardous to the life of the batteries. Uneven charging and discharging characteristics between packs can lead to uneven load sharing and high current circulation between packs. The power from each pack is controlled through switch mode buck converters to provide even current sharing between packs, providing a voltage bus at marginally below the lowest battery voltage.

4.3 Sensing

4.3.1 Joint Sensing

Current sensing is performed in the high power joints by a 0.01Ω resistance in the ground leg of the H-Bridge. The voltage from these sense resistors is amplified by differential amplifiers and measured by the ADC. Current is also checked against a screwdriver adjustable hard limit that is used to trigger the Power Drive Protect interrupt. The position feedback from the encoders on the high power joints provides a count on every edge of both quadrature channels. This provides 2000 counts per motor revolution from the 500 count encoder wheels. In addition, each DSP can measure the bus voltage, and the temperatures of the MOSFETs and motors.

4.3.2 Motion Sensing

In addition to the sensing in each joint, and of course the visual feedback, the robot features 2 x 2-axis accelerometers to provide information about the torso's dynamic behaviour and the relationship to the vertical gravity force. While it is impossible to resolve the motion components of the body's acceleration from the effects of gravity, these sensors may be able to provide information with regard to disturbances while walking – playing a similar role to the human middle ear.

Provision has also been made for the contact switches in the feet and in the joints. These switches may prove useful for determining when contact is made with the ground, or initialising joints at robot start up.

5 Software

The software consists of four main entities: the global movement generation code, the local motor control, the low-level code of the robot, and the simulator. The software is organised to provide a standard interface to both the low-level code on the robot and the simulator. This means that the software developed in simulation can be simply re-compiled to operate on the real robot. Consequently, the robot needs a number of standard interface calls that are used for both the robot and the simulator. Figure 5 shows modularisation of the software, and the common interfaces.

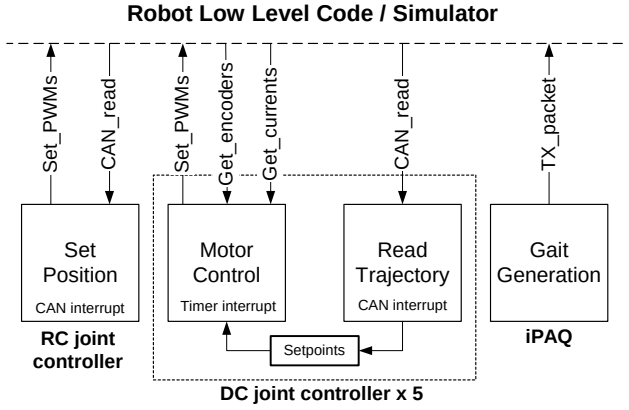


Figure 5: Block diagram of common software modules and the interface used to both the real robot and the simulator.

5.1 Simulator

At present, all evaluations of the robot have taken place in a high fidelity dynamic simulator. The simulator is based on the *DynaMechs* project [McMillan, 1995]. DynaMechs is an object-oriented, open source code library that provides full dynamic simulation for tree-structured robots having a star topology. The algorithms are capable of simulating fixed and mobile bases. The library is based on efficient recursive algorithms for the dynamic calculations, and provides graphical display of the robot in an OpenGL environment.

The simulator uses the DynaMechs package as the core, with additions to simulate specific features of the robot such as the DC motors and motor drives, the RC servos, the sensors, the heterogeneous processing environment and the CAN network. These additions provide an identical interface between the dynamic graphical simulation and the controller and gait generation code. The parameters for the simulator are derived from the CAD models and the data sheets from known components. These parameters include the modified Denavit-Hartenberg parameters that describe the robot topology, the tensor matrices of the links and the various motor and gearbox characteristics associated with each joint. The surface data from the CAD model is also imported to the simulator for the graphical display.

The simulator uses an integration step size of 500μs and updates the graphical display every 5ms of simulated time. When running on 1.5 GHz Pentium 4 under Windows 2000, the simulation updates all 23 joints at a very useable 40% of real time speed.

5.2 Joint Controller Software

For the high power DC motor joints, the simulator provides the programmer with readings from the encoders and the current sensors, based on the velocities and torques from the dynamic equations. In the case of the RC servos, the simulator updates the position of the joints based on a PD model with a limited slew rate. The programmer must supply the simulator with PWM values for the motors to provide the control. The simulator provides fake interrupts to simulate the real events that are the basis of the control software.

There are two types of joint controller boards used in the robot – five controller boards control the fifteen high power motors and one controller controls the eight low power motors. The controller software for the low power motors is a single interrupt routine that is triggered by the arrival of a CAN packet addressed to the controller's mailbox. The routine reads the CAN mailbox for the change in position sent by the gait generation routine. The PWM duty cycle that controls the position of the RC servos is varied accordingly.

The control loop for the high power controllers has two interrupt routines. As for the low power controller, an interrupt is executed upon receipt of trajectory data in the CAN mailbox. The data is used to set the velocity setpoints for the motor control routine. There is also a periodic interrupt every 500 μs to run the motor control software. The motor control routine compares the error between velocity setpoint and the encoder reading and generates a PWM value for the motor based on a Proportional-Integral control law. The routine also checks

the motor current against the current limits, and adjusts the PWM value to prevent over-current situations.

5.3 Motion Generation Software

To this point, the software for motion generation has been used to test the designed geometries and chosen motors in the simulator. The software uses only local joint feedback; it does not use feedback from the joint sensors in a global sense or use the motion sensors to modify the motion to maintain balance. The tests are run without current limiting in the local control loop to evaluate worst-case performance.

The first test motion is a crouch with a return to the standing position. This test has been designed to evaluate the required torques in the pitch joints of hip, knee and ankle. The worst-case results for the knee joint are shown in Figure 6. The second test motion is a lean to balance over one leg, designed to evaluate the required torques in the roll joints of hip and ankle. The joints are driven according to the following equations. The worst-case results for the ankle are shown in Figure 7. In both of these worst cases, the current consumption only briefly exceeds the continuous current rating, and the motor stays within thermal limits.

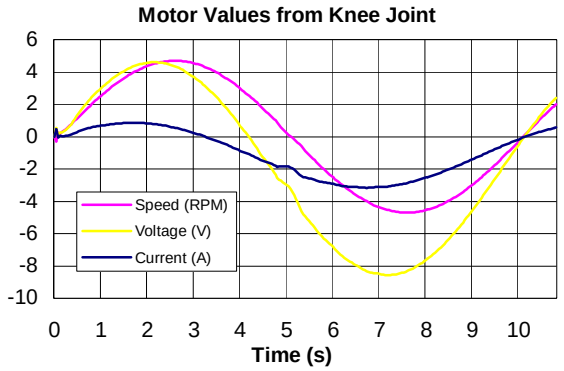


Figure 6: Simulation results for knee motor during a squatting movement. The movement cycle time is 10 seconds.

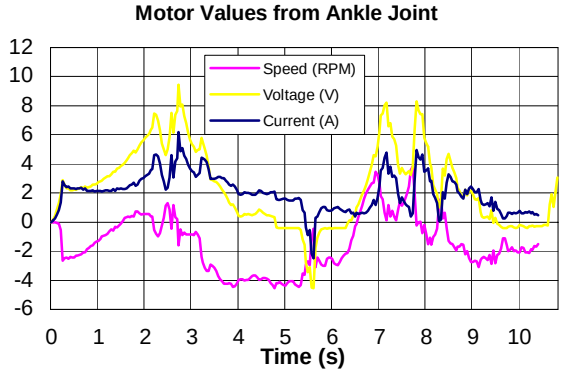


Figure 7: Simulation results for ankle motor during a balancing movement. The movement cycle time is 10 seconds.

6 Conclusions

This paper has illustrated the design of a practical, affordable, autonomous, humanoid robot. The robot is well proportioned in relation to the human form, with most of the major degrees of freedom of the human body implemented. The robot design has a distributed control design with processors dedicated to each of the key roles around the robot. Investigations of the CAD design using a high fidelity simulation have shown that robot is capable of crouching and balancing.

[**Note for reviewers**: This project involves a large team who intend to have the real robot constructed and walking by September. The final paper will have further results, and the conference presentation is likely to feature a video, and possibly the robot itself.]

References

[Chang, 2001] M. Chang, B. Browning and G. Wyeth. ViperRoos 2000. RoboCup-2000: Robot Soccer World Cup IV. Lecture Notes in Artificial Intelligence 2019. Springer Verlag, Berlin, 2001.

[Dempster, 1965] W.T.Dempster and G.Gaughran. Properties of body segments based on size and weight. American Journal of Anatomy,1965.

[Golden, 1990] J. A. Golden and Y. F. Zheng. Gait Synthesis For The SD-2 Biped Robot To Climb Stairs. International Journal of Robotics and Automation 5(4). Pages 149-159, 1990.

[Hirai, 1998] K. Hirai, M. Hirose, Y. Haikawa, and Takenaka. The Development of Honda Humanoid Robot. IEEE Conference on Robotics and Automation, 1998.

[Hodgins, 1995] J. K. Hodgins, W. L. Wooten, D. C. Brogan, and J. F. O'Brien. Animating Human Athletics. In Computer Graphics, (Siggraph 1995).

[McMillan, 1995] S. McMillan, Computational Dynamics for Robotic Systems on Land and Underwater, PhD Thesis, Ohio State University, 1995.

[McGeer, 1990] T. McGeer. Passive Dynamic Walking. International Journal of Robotics Research, 9(2):62-82, 1990.

[Paluska, 2000] D.J. Paluska, Design of a Humanoid Biped for Walking Research, Masters Thesis, MIT, 2000.

[Pratt, 1998] J. Pratt and G. Pratt. Intuitive Control of a Planar Bipedal Walking Robot. IEEE Conference on Robotics and Automation, 1998.

[Raibert, 1986] M. H. Raibert. Legged Robots that Balance. MIT Press, Cambridge, MA, 1986.

[Wyeth, 2001] Wyeth G.F., Kennedy J. and Lillywhite J. (2000) Distributed Digital Control of a Robot Arm, Proceedings of the Australian Conference on Robotics and Automation (ACRA 2000), August 30 - September 1, Melbourne.

[Yamaguchi, 1998] J. Yamaguchi, S. Inoue, D. Nishino, and A. Takanishi. Development of a Bipedal Humanoid Robot Having Antagonistic Driven Joints and Three DOF Trunk. Proceedings of the 1998 IEEE/RSJ Conference.

Acknowledgements

I would like to thank the following people:

Firstly, thanks to Dr Gordon Wyeth, my supervisor, for his guidance and help throughout the year.

Tim Cartwright for his concurrent work on the joint controllers for the servo board and also use of the CAN Network.

The remainder of the GuRoo humanoid team for their support, help and understanding of different aspects of the humanoid project.

My siblings for enduring the late nights and lack of holidays.

Abstract

This thesis illustrates the design, construction and testing of DC motor controllers for a humanoid robot. The humanoid is to have 23 degrees of freedom, from the ankle to the neck. To control each joint throughout the body reliably and quickly is a major part of this thesis.

The design implements a recent Digital Signal Processing (DSP) chip from Texas Instruments. This chip has been designed with motor control in mind, with a Controller Area Network (CAN) interface for networking, Pulse Width Modulation (PWM) outputs and Quadrature Decoding (QDEC). This forms a powerful-networked controller base for the robot. Each controller controls three DC motors and is placed physically close to the joints it is controlling. This keeps wires short and noise to a minimum. Each controller is networked together through CAN for central control from a Compaq iPaQ. This design process is shown, with device selection and subsequent implementation.

Limited software for this DSP is shown in later chapters. Emphasis is on the control of the DC motors found in the lower half of the robot. To date, only testing software has been written for the controller boards. A simple control loop has been written and this implements a proportional control loop. The abilities of each controller are shown with possible future improvements. Overall the hardware works to its intended design, although not all elements have been tested. There has also been limited success with software design.

Table of Contents

List of Figures and Illustrations

Chapter 1 - Introduction

1.1 Introduction

The aim of this thesis is to outline the design and eventual creation of DC motor controllers for a humanoid robot. The robot (named GuRoo) has 23 degrees of freedom ranging from ankle joints to neck. This task hasn't to date been undertaken at The University of Queensland and so this design is to be created from the ground up. With only a biped walker previously constructed within The University of Queensland, this new project is a very challenging one.

This robot has been designed on a 7-year-old child. The robot is in proportion to a human and represents a human as closely as possible. This is for the eventual interaction with humans, and possibility of a human vs. robot soccer league by 2050. It is hoped this robot will walk at $0.1ms^{-1}$, track a soccer ball, walk to it and kick it into a goal. The aim was to have this completed for the robot soccer championships in Seattle in early August 2001, but due to circumstances this was unfortunately not possible.

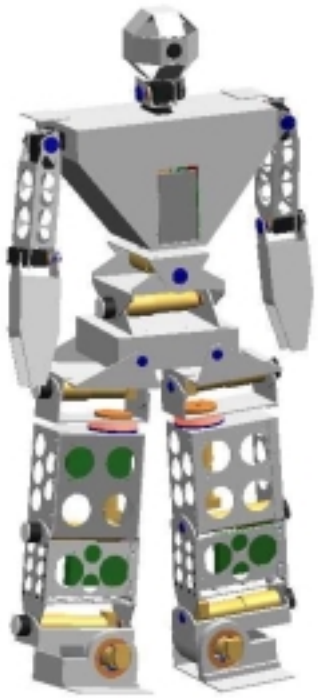


Figure 1.1: Model of the Humanoid

The joint controllers presented within this thesis each control three DC motors. Each of the controllers communicate with one another through a CAN (Controller Area Network) protocol. Each controller has a Texas Instruments TMS320F243 DSP (Digital Signal Processor) micro controller. The controllers are connected to a centrally located Main Processing Unit, which has been chosen as the Compaq iPaQ (PDA with colour screen). The iPaQ has a StrongArm

206Mhz processor, with 16MB of ROM for the operating system and 32MB of RAM, which should be sufficient processing power and memory for this application.

1.2 The Humanoid Thesis Team

With this new design in mind, it was required for the group of twelve to actively work together to find a solution to this mammoth task. There were four members involved with the control and power systems, three involved with the vision system, two writing the walking software and three doing the mechanical design. The twelve members of the team are shown below in table 1.1, with each of their responsibilities: -

Table 1.1 – Team Responsibilities

Name	Responsibility
Jarad Stirzaker (Thesis Author)	Joint Controllers
Tim Cartwright	Joint Controllers
Bartek Bebel	USB to CAN Network Interface
Nathaniel Brewer	Power System
Shane Hosking	iPaQ to Vision Hardware
Andrew Blower	Vision Hardware
David Prasser	Vision Software
Andrew Smith	Walking Software
Emanuel Zelniker	Walking Software
Damien Kee	Driver System Design and Implementation
Mark Wagstaff	Mechanical Design
Anthony Hunter	Mechanical Design

This thesis aims only to show the design of the DC motor controllers of the humanoid. Timothy Cartwright and Jarad Stirzaker (thesis author) have worked on the joint controller project concurrently. This thesis contains emphasis on the hardware for the DC motor boards (Boards 1-5 explained in *Chapter 3 - Specifications*) and controller software for control algorithms. Cartwright, 2001 [3] has shown the remainder of the controller thesis, including the DC servo board (Board 6) and the CAN interface. To gather an overview of the whole project it would be required to read all the theses relating to the humanoid (see Appendix A).

1.3 Thesis Goal

The aim of this thesis is to show the control of 15 separate joints of a humanoid robot. Each joint should have closed loop control with open loop control by higher-level software. The controllers were to be produced under budget and also to a size and weight restriction. The size of each controller is important to fit within the limited size of the robot chassis.

Due to the size and budget restrictions, it will be required that one micro controller controls more than one motor. There will be a limited amount of current and torque that can be supplied to the motors, due to these restrictions. These are just a few of the decisions and sacrifices that were required for the project.

This thesis will show the design, implementation, coding and testing of each of the DC motor controller boards. It will detail the selection of the microcontroller used, along with all the other components needed following this initial selection. The thesis will then show the design of the PCB with selected components. Finally a description of the low level software and finally the testing of the controllers as a whole will be included.

PI (Proportional plus Integral) for improved steady state error, is hoped to be complete by the end of the year. This will provide a framework for further work and subsequent control theses of the humanoid. This control system will later be improved with the adaptation of a PID (Proportional plus Integral plus Derivative) controller, for improved steady-state error and transient response. These control techniques are shown in Section *2.4 Motor Control*.

It is anticipated that these controller boards will be tested in a working chassis of the robot, even if only the legs of the robot can be created by the time the Thesis Expo is held on October 30th. However, at the time of writing this thesis, the chassis wasn't built. Simple testing had been done on the controller boards, with advanced control not having been implemented to date.

1.4 Chapter Outline

Chapter 2 contains a review of the work that has been already done in this area of robotics. Also discussed is the DSP controller, CAN protocol and basic controller theory.

Chapter 3 describes the basis of the problem with which needs to be solved and how it will be achieved. Contained within this chapter is a block diagram of the hardware which is used to create the hardware itself, there is also a brief description of the hardware and software solutions.

Chapter 4 provides a detailed review of the hardware design process for each of the required blocks of the controllers. Included is the component selection process and circuit design, PCB design and creation.

Chapter 5 outlines the low level software that is used to control each of the controllers contained within the robot. This software demonstrates the intended closed loop control algorithms on each of the five controller boards, as well as currently tested software.

Chapter 6 discusses the testing of the project to date. This chapter shows that since the software hadn't been completed at the time of writing this thesis, what had been is demonstrated. It also details any alternatives and other complications that were met during the project.

Chapter 7 concludes the thesis with overall remarks on the project. Also shown is the scope for future work, areas that can be improved, and perhaps other simpler solutions that were found.

Chapter 2 – The Past and the Present

Before the task of creating controllers for the motors of each joint was undertaken, it is first required to look into the past areas of work and current technology to aid in the creation of the design. The aim is to have a freestanding one metre tall robot, with on-board batteries and control for tracking, walking, balancing and kicking. Shown will also be some current and past projects taken via various companies and universities.

2.1 Current and Past Projects

Several projects are currently ongoing in this field of robotics. These include several projects from Honda, Sony and Shadow Robot Project. These robots however are a league ahead of our intended design due to their far superior budget, assigned workload to the project as well as the fact that these are run by multi-million dollar companies. The University of Queensland is hoping to rival these large companies by producing a robot that can function to the same degree on an extremely low budget and utilising university students. Work may eventually be conjoined with other universities such the University of New South Wales and Melbourne University.

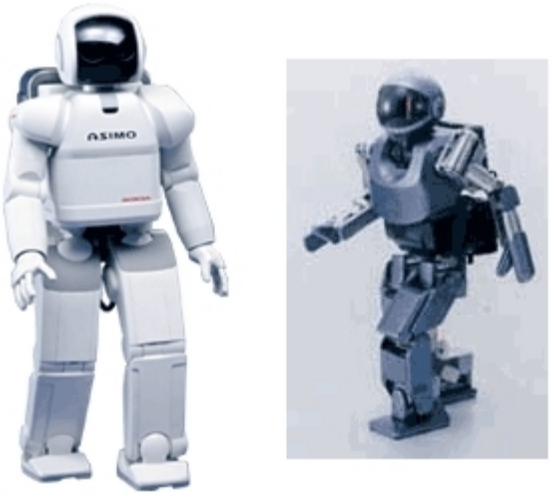


Figure 2.1: Sony and Honda Humanoids

The Korean Institute of Science and Technology (KIST) has developed a humanoid robot CENTAUR that has two arms and four legs based on a fictional centaur. Similar to the current project, all parts including motor controllers, sensors, voice recognition systems and batteries are on board. It has 36 joints with 37 degrees of freedom. Obviously it has a different mechanical design, but the controllers are similar to those intended for the humanoid.

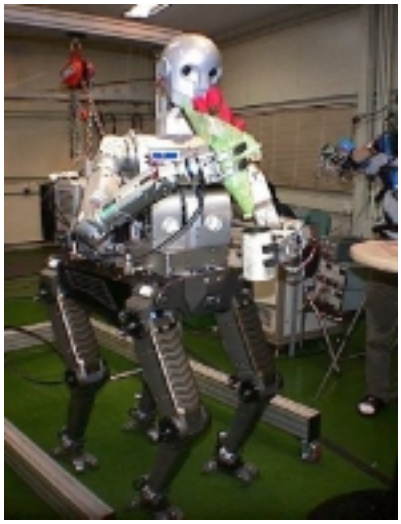


Figure 2.2: The CENTAUR

Kim et al, 1999 [5] has shown that each of the controllers is operated with a Texas Instruments TMS320F240 DSP chip, which is the same generation, but different model to that used in the humanoid. PWM, ADC and encoders are used in these controller boards, with SPI used for the communication. These boards are also small enough to be embedded into the robot itself, allowing free wireless motion. This is very similar to the humanoid boards, with set requirements of size and also on board peripherals. It is shown that the control of several motors per board is achievable with hardware very similar to that found in the humanoid.

A MVME162 CPU is the highest level controller and acts in a similar manner to the iPaQ (discussed in chapter 3). This drives the lower level controllers with position or velocity commands for each motor. However, there is also a higher-level controller, in the way of the user, from which commands are sent to the CENTAUR through use of wireless Ethernet (UDP/IP). This is not within the humanoid design, as it is a self contained and self-controlled unit, making it a more advanced and adaptable system.

Kennedy, 1999 [4] has also provided a framework that will form a good basis with which to extend the design. This thesis outlines the control of a PUMA 560 industrial arm. The PUMA arm has six degrees of freedom, each controlled with a separate controller board and networked together with a CAN network. Each of the motors is a high power motor, with currents up to 9A possible (with 4A continuous current). The

continuous current obtainable by each motor is equivalent to the motors for the humanoid, however, the semi-discrete drive circuitry allows for 9A non-repetitive currents.

Figure 2.3: PUMA Arm

Each of the controller boards presented in subsequent chapters control three motors each. Some difficulties in safety features arise due to this, but the savings in cost and size are necessary for the final design requirements. The principals for control and the use of the CAN network by Kennedy, make it an attractive basis for comparison. This initial design can be used and extended to fulfil the specifications outlined in Chapter 3.

Each of the motor controllers is operated with the Texas Instruments TMS320F241 DSP. Similar to the CENTAUR project and the humanoid utilising the same generation of DSP chip but a different model. The Texas Instruments TMS320F243 has been used in the humanoid controllers, however being the same generation and sibling models, makes this thesis a great starting point.

To date the hardware has been the focus of this thesis. The hardware is based on Kennedy's thesis, with a similar sized board produced. The principles used within this thesis have presented a great basis for design. Some implementations extended from Kennedy's thesis have been the serial programming of the boards, the PCB layout and also the split plane four-layer board.

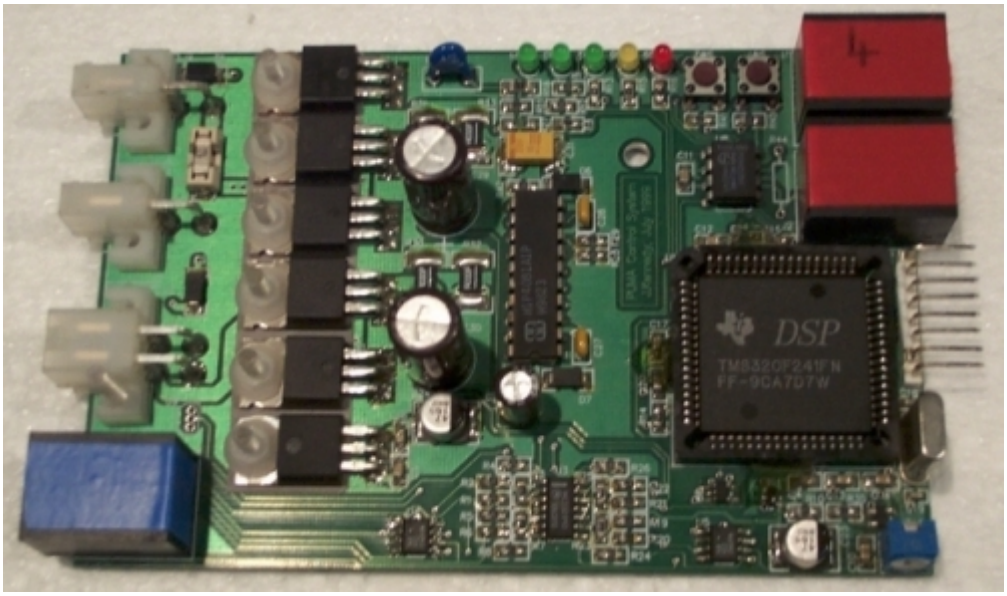


Figure 2.4: Kennedy's Controller Board

Due to time constraints the software has been reviewed, however, this hasn't been fully implemented yet for the humanoid. Kennedy's software has been written in assembler, whereas the humanoid's is written in 'C'. Despite different coding, the same principles and processes are still required to obtain full control of each motor.

The hardware and software design included with this thesis has been very useful in the creation of the humanoid controller design. With quite a bit of information on this project, this will be a very useful thesis for the creation, design and implementation of the humanoid controllers.

2.2 CAN Network

Similar to Kennedy, 1999 [4], the CAN (Controller Area Network) will be used to control several controller boards contained within the body of the humanoid. The use of CAN is widely documented, with the main use for this protocol in the car industry. Bosch GmbH [1] developed this protocol for use in the automotive industry. It allows many nodes on a network to communicate to other nodes, over a two-wire bus, with high reliability and bandwidth. Particularly useful in very noisy environments were reliability is essential.

The system incorporates a sophisticated arbitration scheme to dispose of message collisions within the network. CAN is a carrier sense, collision detection network protocol (CSMA/CD) which makes it quite useful when there is a lot of traffic on the network. A data packet can hold a maximum of eight bytes of data, and has an 11-bit arbitration field used to determine priority and thus through filtering the required node receives the message. This allows for up to 2048 priorities or nodes to be present on the network. It can allow up to 536 million nodes using extended CAN that has a 29-bit arbitration field.

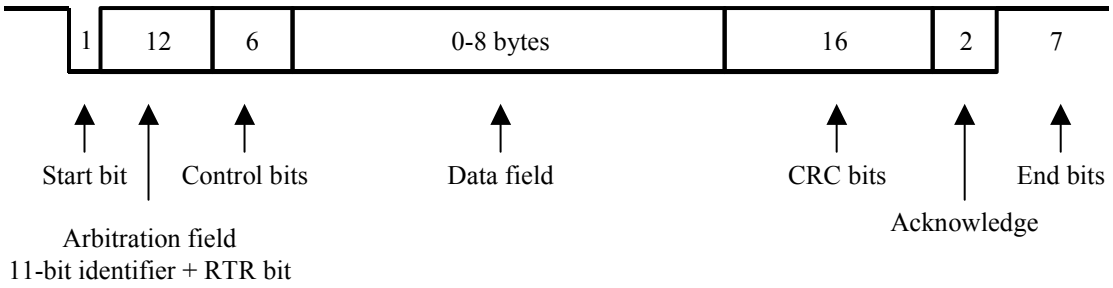


Figure 2.5: CAN Data Frame

The CAN network can operate at 1Mbit/s for short networks up to 30m. The error handling of the protocol can allow up to five random errors and burst errors of less than 15-bits in length to be detected.

If two nodes are transmitting at the same time, the message with higher priority (lower arbitration field value) will continue transmission while the other node will terminate

before sending another bit. Obviously a node senses the network, and will not start transmission until the network is free. The network operates on two states, dominant and recessive. For dominant, the CAN-H and CAN-L lines are 5V and 0V respectively, and on recessive are both 2.5V. If node A transmits a dominant bit, and node B a recessive, node A will read a dominant bit and B will receive an error.

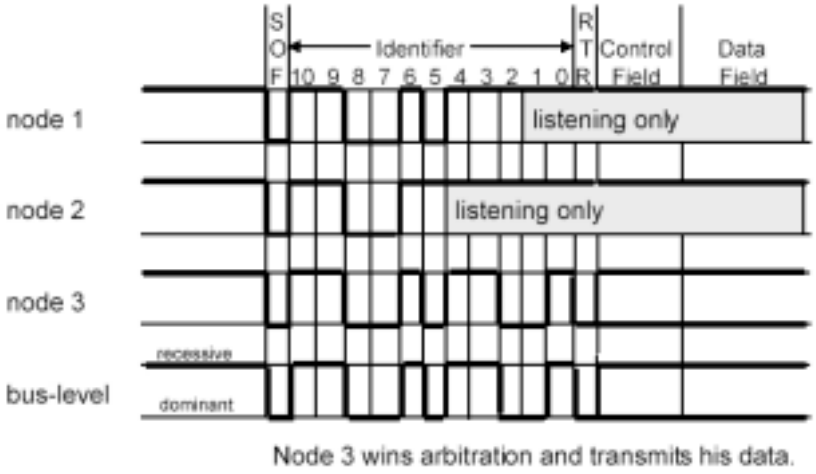


Figure 2.6: CAN Arbitration Diagram

This protocol can be seen in Figure 2.6 above. Nodes 1,2 and 3 will start transmission of there messages, at the same time. Each transmitter checks the arbitration field until the intended transmitted bit doesn't match the bus bit. So at bit 5, node 2 is transmitting a recessive bit, and will read back an error since the other nodes are transmitting a dominant. At this point node 2 terminates transmission immediately and will wait for the bus to be free before trying to retransmit. Transmission will continue until bit 3 at which point node 1 loses arbitration, and hence node 3 has the highest priority and transmits its data. The resultant bus state will be the same as that on node 3.

Although we will only be using seven nodes on the network (six controller boards and the iPaQ), there is plenty of room for expansion. Also, these extra arbitration fields will be used for such things as a global shutdown command, as well as control commands and sensor information.

2.3 Micro Controllers

Traditional control of motors in industry and robotics has relied on analog components. A simple PID (Proportional plus Integral plus Derivative) controller can be made from a few resistors, capacitors and an op-amp. This can provide virtually infinite resolution and continuous processing of the signal. However, analog controllers suffer from component ageing and temperature drift.

There has been a major swing to digital controllers, comprising of a micro controller or microprocessor. These controllers require additional A/D and D/A converters (if not on-board) since analog signals are required for this control. The micro controller doesn't suffer from component ageing or temperature drift, but has reduced resolution and due to conversions, creates a phase delay in the system. Due to their programmability, they can be easily upgraded and contain far more advanced control. In the end, digital controllers are far better than their analog counterparts.

A major task of this thesis was the selection and subsequent implementation of the micro controller into each controller. However, to just simply select the right micro controller for the job sounds easy enough, but when cost, sourcing, memory and required peripherals come into play, this becomes a very difficult task.

There are many micro controllers that could be used for this kind of application. Companies like Texas Instruments, Motorola, Analog Devices and Atmel make such devices. All are fairly close in their features but would often differ in the on-board peripherals that they have, and also the size of their memory. Peripherals available on micro controllers can vary from just a simple I/O port to containing a variety of peripherals. Obviously with cost comes added features and ease of use.

These peripherals include high speed (20MHz), in-circuit programming, multiple channel A/D converters, quadrature decoders, multiple PWM channels, external interrupts and networking capabilities (such as the CAN interface). The micro controller that is required for the application is very specific, and hence the requirement

for peripherals is very specific. There are thousands of micro controllers that could be used, but it is the cost, size and speed that determine the ideal one.

Fortunately for us, these micro controllers have made the control algorithms originally done with analog circuitry relatively easy. They are also relatively easily expandable with changes to the software. And because these can be networked together, they become very powerful in the control of multi-node robots, and other electrical equipment.

2.4 Motor Control

Before we can utilise the micro controllers discussed in the previous section, it is first required to understand the basics of motor control. Utilising this knowledge the software can then be written to fully reflect these control principles. There are a varying number of ways in which control can be implemented, with different systems and also using analog and/or digital circuitry.

The systems to be used and the type of circuitry to be used will depend on what accuracies and compensating requirements are needed for the particular application. Obviously the control of an elevator will have different requirements to that of a delicate surgeon point.

2.4.1 Open and Closed Loop Control

When a system is simply driven by a controller, with no respect to its current state and its output isn't compared to its input, this is known as an open-loop control system. The input to the system is in such a manner that it is converted to what can be utilised by the output of the system (these could be voltages, currents, angles etc).

An example of this is a drill, used to screw in a screw. In essence, the drill will keep driving the screw until either the thread of the screw is destroyed or the drill itself is destroyed. The input to the system is your finger on the trigger and the output is the

drive torque of the motor. The drill (ignoring automatic clutch) doesn't consider disturbances such as the completion of screwing the screw in.

However, if the visual confirmation of the current screw position were included this would be considered a closed loop system, as feedback is now included. When the screw reaches its required position the feedback of visual confirmation will prevent it screwing any further. The input to the system will remain constant (the desired position), but in the closed loop system an error between the current output and intended output is used to feedback the system, to drive it to the intended output. Such feedback could be a pot, sensor, variable resistor etc. A voltage is often used here as this can easily be subtracted from the input voltage to create the error voltage.

Closed loop control is thus formed when the system incorporates devices that will monitor the difference between intended output and actual input. This difference can be used to correct the output, and this loop is continually run until the intended and actual outputs match. This could be the intended position of the screw height with the actual position. Closed loop control is utilised with motor control, as continuous feedback is required to drive the motor, and thus this will be discussed in following sections.

2.4.2 Basic Controller Topologies

The closed loop control system will be focussed on, as this will be required for the humanoid control system, Figure 2.7 below illustrates a simple closed loop system.

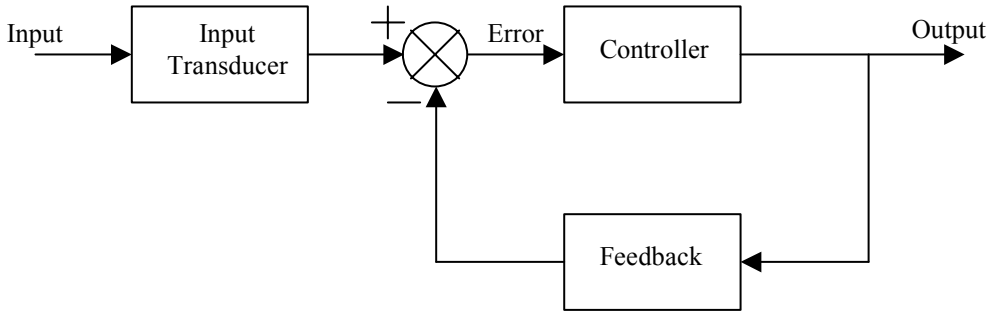


Figure 2.7: Closed Loop Block Diagram

The control loop operates by comparing the output of the system with the input. An error is obtained and this is subtracted from the desired input to form an adjusted error input to the compensator. This loop continues indefinitely matching the input to the output, even after they are the same. The rate at which the output matches the input and the overall error can be changed depending on the application.

Transient response, steady-state response and also stability play a major role in the selection of which topology to use and the components used to create this. Nise, 2000 [7] states, "In the case of an elevator, a slow transient response make passengers impatient, whereas an excessively rapid response makes them uncomfortable." Steady-state response concerns with the final steady state accuracy of the system. Stability is concerned with whether the system will finally reach a constant value, or it will indefinitely oscillate.

There are many ways in which this control loop compensator can be designed. Ignoring the simplest form of feedback, the unity feedback system, three topologies are discussed here. This includes the PD, PI and PID control compensators. Where 'P' is proportional, 'I' is integral and 'D' is derivative. Each of these is added together in each different system to form the error value.

A PI (Proportional plus Integral) controller is used to improve steady-state error. By placing an open-loop pole at the origin (poles and s plane discussed in Nise, 2000 [7]), this can be achieved. However, it is also required to place a zero close to this pole, but not at the origin. This can be exactly implemented with active components or approximated to the lag compensator with passive components. The transfer function for this compensator is as follows:

$$G_{PI}(s) = \frac{K(s + z_c)}{s} \qquad \textbf{(Eqn. 1)}$$

The PD (Proportional plus Derivative) controller is used to improve transient response, namely settling time. To do this, a zero is placed at some point on the s-plane where only a small adjustment to the gain is required. This is usually placed at some point so

as to get the desired closed loop pole locations. If passive components are used, this can be approximated as a lead compensator. The transfer function is as follows:

$$G_{PD}(s) = K(s + z_c) \qquad \textbf{(Eqn. 2)}$$

Finally, the PID (Proportional plus Integral plus Derivative) controller can be used to improve both steady-state error and transient response. This is achieved by combining both the PI and PD controllers with their independent properties. When approximated with passive components a lag-lead compensator is formed. The transfer function is as follows:

$$G_{PID}(s) = K\frac{(s + z_{lag})(s + z_{lead})}{s} \qquad \textbf{(Eqn. 3)}$$

It is intended to use the PID controller for control of each of the motors as this allows adjustment of both transient response and steady state error. This control will be shown in chapter 5, however it is required to have certain hardware to be able to achieve this control in the first place.

Chapter 3 – Specifications

As already outlined the aim of this thesis is to show the control of 15 separate joints of a small-scale humanoid robot. Cartwright, 2001 [3] covers the remaining eight degrees of freedom required by the humanoid. This sounds simple enough, however there were a few constraints that limited the choice of components and circuitry.

3.1 Hardware Specifications

Given requirements from the both the university and also the project supervisor, the following restrictions were specified: -

- Limited budget of $3000
- Board size restriction of 170mm x 100mm x 40mm
- Reliable and fast network to link individual controllers
- Fast computation power of micro controller
- Range of peripherals including ADC and PWM

Considering Kennedy, 1999 [4] controls one motor per board, and given the restrictions above, mainly size and budget, each of these controllers is required to drive more than one motor. Component selection will be such as to reduce the size of the boards to a minimum. Also to be considered is the power and current required by the hardware.

It is required that the controller has several peripherals to allow the connection of many subsystems to the controller core. Some of these include motor drivers, quadrature decoders, temperature and foot sensors and also communications. For this application it will also be required to have in circuit programming. Programming through a network would ultimately be desirable, thus the boards wouldn't need to be removed from the robot itself.

For the safety of the robot and also the supervisors of its operation, it will be required to have many safety features. Such safety features include a master off switch, motor current sensing, limit switches and also temperature sensing. The watchdog timer will be a safety feature in software, and will reset on phantom and incorrect interrupts. If a problem does arise, then with the use of the CAN network a broadcast emergency message can be sent and all motors shut down, and the central processor notified of this condition.

It is hoped to have simple PI (Proportional plus Integral) control of each motor. A velocity command will be transmitted to each controller board every 2ms. The software is thus required to manipulate this velocity command into a useable PWM duty cycle that can then be applied to the driver and hence motor. The quadrature decoders can then be used to feedback the actual velocity of each motor.

The software is to operate with use of the interrupts available on the Texas Instruments TMS320F243 device. The device runs a "main" empty loop until an interrupt from one of the several devices is obtained. At this point the interrupt is called, the required software is run and then control is passed back to the "main" loop. The control loop will run at 500Hz, with the ADC system being utilised constantly for up to date information for current sensing and temperature readings.

3.2 Controller Location

There are 15 degrees of freedom that are managed by the DC motor controller boards, from the back joints to the ankles. The degrees of freedom contained within this thesis are distributed as follows: -

- 2 ankle freedoms (pitch and roll)
- 1 knee freedom (pitch)
- 3 hip freedoms (pitch, roll and yaw)
- 3 waist freedoms (pitch, roll and yaw)

The number of freedoms is easily seen here in Figure 3.1. These clearly match up with the distribution listing above. The remaining eight degrees of freedom are also shown here.

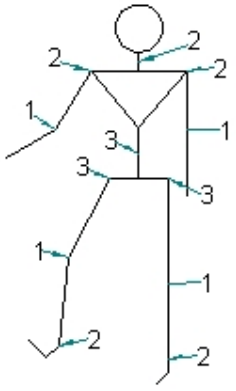


Figure 3.1: Joint Locations

Each of the DC controller board will control three motors. Since there are 15 degrees to be controlled, five boards will be required for this task. As can be seen in Figure 3.2 below, boards 1 & 2 control the two ankle freedoms and a knee freedom. Boards 3 & 4 control the three hip freedoms of each upper leg and board 5 controls the three waist freedoms. Each of these 15 motors operates with the same motor and gearing.

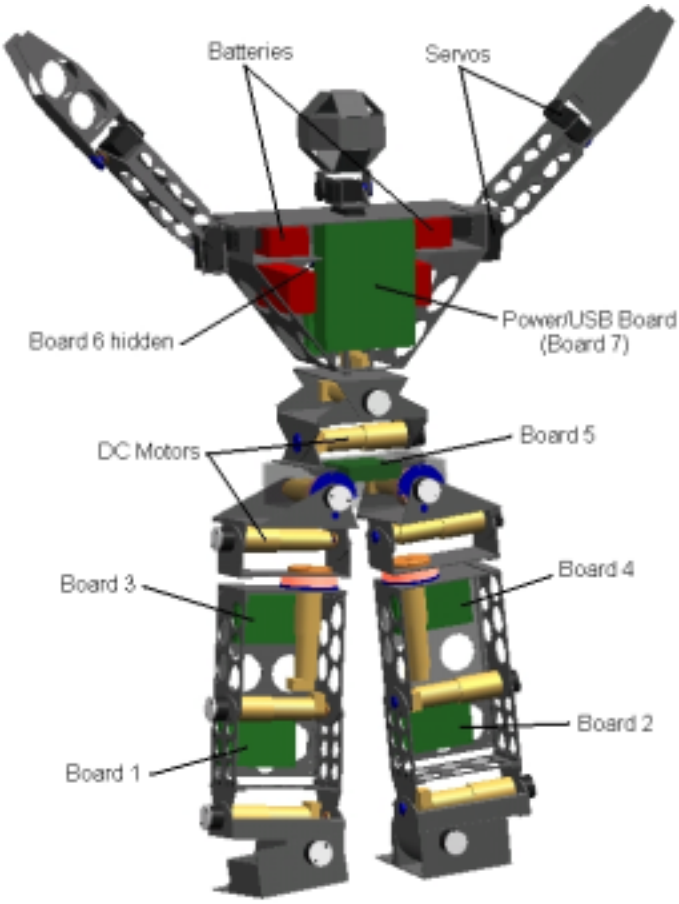


Figure 3.2: Intended Board Locations

In addition to controlling three motors, each board will monitor the temperature of each motor. They will also monitor current to the motors and limit switches for calibration and limitation of each joint. In addition boards 1 & 2 will read foot sensor values and send these back to the iPaQ. Board 5 will monitor the gyroscopes and accelerometers for closed loop control by the iPaQ. These features are intended for the final product, but may not be implemented this year.

3.3 iPaQ to CAN Network

As stated already, overall control of the humanoid is through the CAN network, and these messages originate from the centrally located Compaq iPaQ. This is a PDA (Personal Digital Assistant) and can be used to facilitate a variety of tasks. The iPaQ actually operates a hand-held version of Windows or Linux. It has a StrongArm 206Mhz processor, with 16MB of ROM for the operating system and 32MB of RAM that should be sufficient processing power and memory for this application. It also has a colour touch screen and speaker that may be utilised at a later date.

Figure 3.3: Compaq iPaQ

The CAN network is connected to this device through the USB (Universal Serial Bus), that can operate at up 12Mbit/s. However, the CAN network can only operate at a maximum of 1Mbit/s and these two protocols are not directly compatible. It is thus required to bridge these with a seventh board. This seventh board will be used for the USB to CAN operations as well as the power control required for the humanoid.

To keep design of this seventh board simple, the Texas Instruments TMS320F243 micro controller was used again. It is intended to initially have open-loop control from the iPaQ, with closed-loop control to be implemented later. Control commands are forwarded from the iPaQ to Board 7 via USB, which then converts this to the CAN protocol and forwards them onto the network for the intended board. This same scheme but in reverse is used to return the sensor information to the iPaQ for later closed-loop control.

The implementation of the USB to CAN bridge is described in Bebel's thesis, "*Design and Implementation of a USB-to-CAN Bridge for the GuRoo Project*" found in Appendix A.

Chapter 4 – Hardware Design

As shown in the previous chapter, each of the individual systems will be created to work together to control several motors simultaneously. A design will be required to meet all of the design criteria outlined in Section *3.1 Hardware Specifications*. It was first required to choose a micro controller from which the subsequent systems could be added.

4.1 Block Diagram

Given the specifications outlined, the hardware can be represented with the block diagram in Figure 4.1. It shows each of the required hardware blocks for complete control of each of the DC motors. Each of these systems will be described in subsequent sections of this thesis.

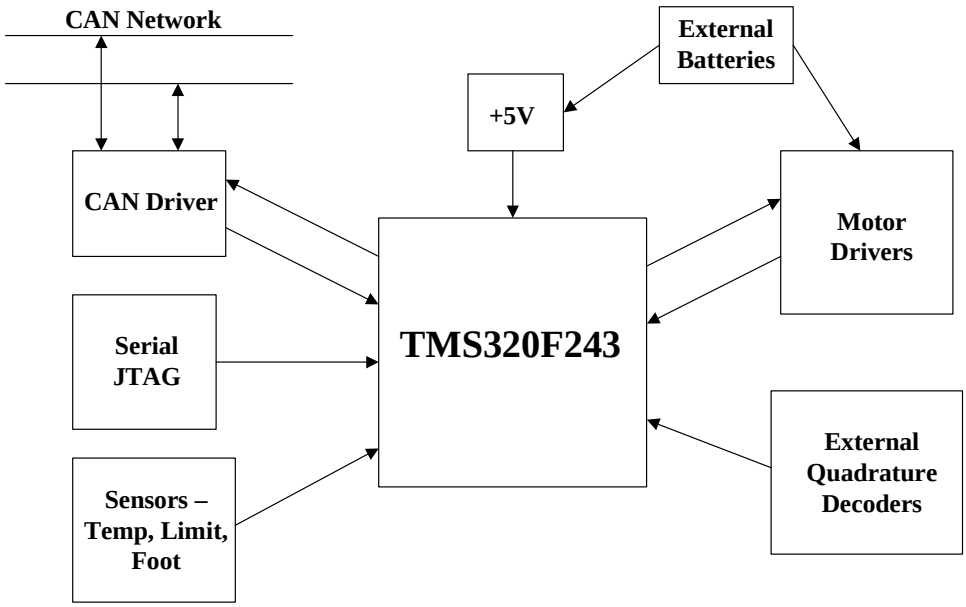


Figure 4.1: Hardware Block Diagram

Each arrow represents an interaction between blocks, be this a signal or actual data. The direction of the arrow represents the direction in which this interaction will flow. So for the case of the External Quadrature Decoders, they will be sending their data to the TMS320F243.

4.2 Micro Controller

As seen in Figure 4.1, there are several considerations that need to be taken when choosing a micro controller for this application. The requirements for our micro controller are as follows: -

- Sufficient internal memory (Flash or EEPROM)
- CAN Network interface
- Multiple PWM Outputs
- Quadrature Decoders
- Multiple ADC Inputs
- External Interrupts
- In Circuit Programming
- Speed and DSP

There are quite a few micro controllers that have some of these attributes but it was quite difficult to find one with all of them. Due to accuracies and peripherals, an 8-bit processor would not be sufficient for the task. A high-end processor wouldn't be acceptable either with their high cost and lack of peripherals. From this either a 16-bit or 32-bit controller would be required. A couple of micro controllers were found that could do these tasks. These included the Motorola 68HC12 and 68376, Texas Instruments TMS320F241 and TMS320F243.

The Motorola 68376 was the ideal solution for this application, however, this chip had a lead-time of six weeks, which at the time seemed unreasonable (early May for July completion), and they also required external memory. There was also a budget problem in that it was required to purchase 24 chips at a total cost of $1000. Since only seven chips were required, and the Texas Instruments chips could be sampled, the TMS320F243 was our next best candidate. The Texas Instruments chips were very suitable as they were designed specifically for motor controllers.

The TMS320F243 was required over the TMS320F241, as it was needed to have three quadrature decoders per board. Since both chips only had one, it was required to have external decoders, and only the TMS320F243 had an external data port. A normal port couldn't be used since these were taken with PWM, ADC and the required control signals.

In the end the chip was very well suited to the application as it contained 8k of internal flash memory, eight (8) 10-bit multiplexed fast ADC input channels and eight (8) PWM output channels with adjustable dead band. Also onboard was the required CAN interface, 20MHz operating frequency, one (1) quadrature decoder and external data and address ports. An external Power Drive Protection Interrupt (PDPINT) was also present and could be utilised for safety features.

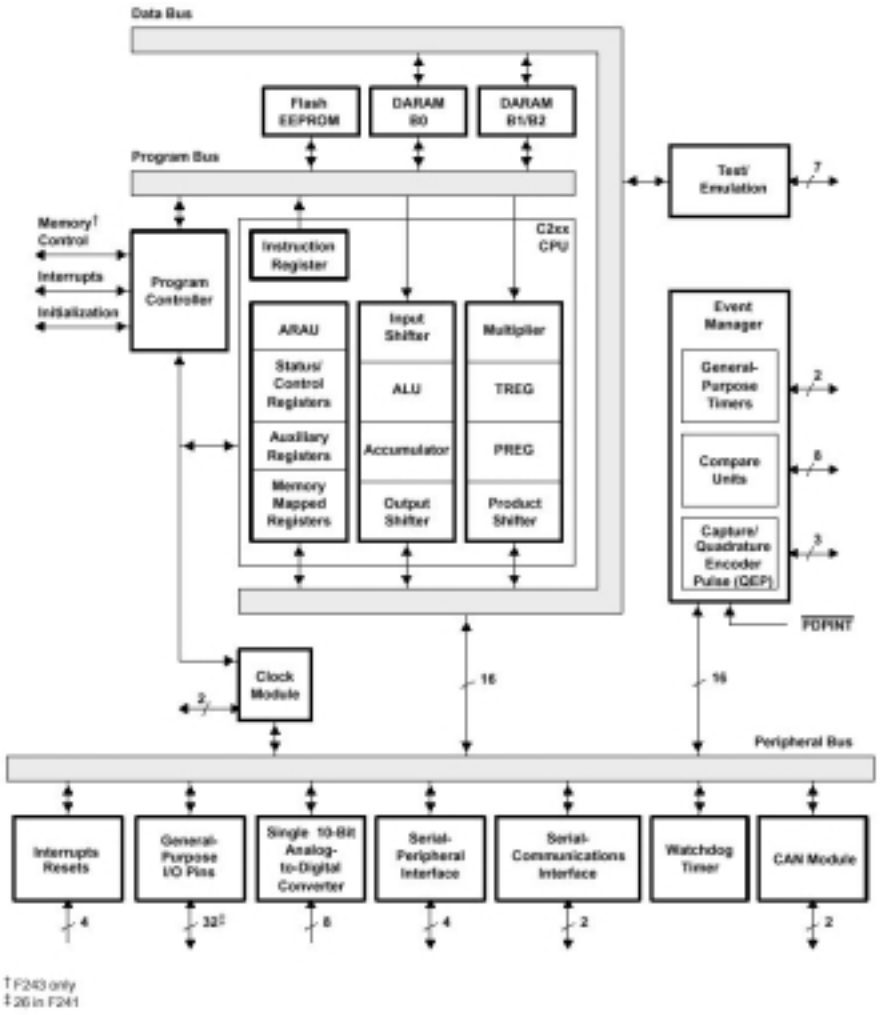


Figure 4.2: TMS320F243 Internal Block Diagram

A 5MHz crystal is used which generates a 20MHz internal clock signal with the internal phase locked loop (PLL) circuitry. The 20MHz clock is also output on the CLKOUT pin and can be used by external circuitry for clocking purposes. For correct operation the Maxim MAX811 reset controller was used. It generates a reset low pulse guaranteed for 140ms on VCC dropping below 4.63V. It could also be manually reset using the external reset button RSW. This device helps prevent the corruption of the internal flash memory and aims to guarantee a stable supply.

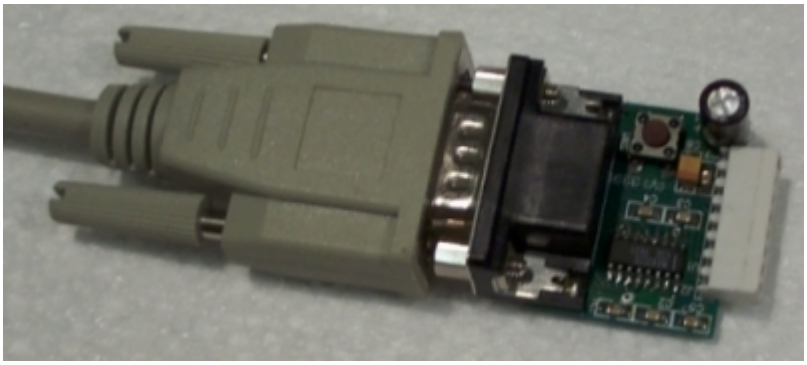


Figure 4.3: Serial Programmer

An external serial boot loader is used to program the device, this was designed and used by Kennedy, 1999 [4]. To program the TMS320F243, both the V_{CCP} (Flash programming voltage) and BIO (Branch control input) pins are asserted high. With the connection of serial boot loader to the 8-pin header SRL, the device automatically enters programming mode. If the internal flash is not correctly written (due to some error) and corrupts the internal boot loader, this code will need to be restored with the use of the JTAG adapter. This connects directly to the 14-pin header JTAG. The use of these programming features is outlined in Appendix E.

It should be noted, due to the PCB design, the serial programmer needs to be connected in reverse to the orientation of the board being programmed. In other words, the serial programmer will be upside down in comparison to the controller board.

4.3 Motor Driver Electronics

It was considered essential to use the driver that could take advantage of the on board PWM output channels on the TMS chip. By varying the duty cycle of the PWM signal the speed and direction of drive of the motor could be altered. A switch-mode power amplifier would be required to drive the motors, such a device in a H-bridge configuration was ideal for this application.

However, there were several options for this H-bridge that could be taken and all were considered. The two main options were the use of a completely integrated package containing drivers and switching devices. Or a semi-discrete solution could be utilised for better efficiency and more output power. However, due to our budget restrictions and also the limited PCB space, it was essential to lose some of the efficiency for the smaller size. The space saving was about 80% compared to a semi discrete solution. This size restriction was the main factor in choice of driver circuitry. If there had been no size restriction then the semi-discrete option would have been more suitable.

The integrated package needed to include four power MOSFET's to switch the circuit. The driver was required to operate at 40-42V due to the battery supply voltage and also power transfer. The driver chip had to be able to handle this as well as give a decent output current to be able to drive the motor under load. The higher the output power and hence higher output current capability, the better the driver. However, due to the complete integrated package, heat dissipation does become a problem and hence heat sinks will need to be considered.

The integrated package needs to be a full H-bridge or half H-bridge and be able to handle PWM frequencies up to 100kHz. The SGS-Thompson L6203 DMOS Full Bridge Driver was a very suitable component for this requirement. This device can be supplied with up to 48V and can handle up to 100kHz, both required by our application. It can drive 5A peak and 4A continuous RMS current. The on-resistance of the MOSFET's is also fairly small at 0.3Ω. The device has an enable pin as well as a dead time protection with a minimal 100ns dead band created between transitions. This prevents the simultaneous conduction of both arms of the H-bridge, which will result in

a short between the power rail and ground (minus the small on-resistance of the MOSFET's).

The package of this device that has been used is the 11-pin multiwatt package. This allows the greatest heat dissipation and allows easy connection to a heat sink. Considering the placement of the each of the boards, the actual frame of the robot itself has been considered as a heat sink.

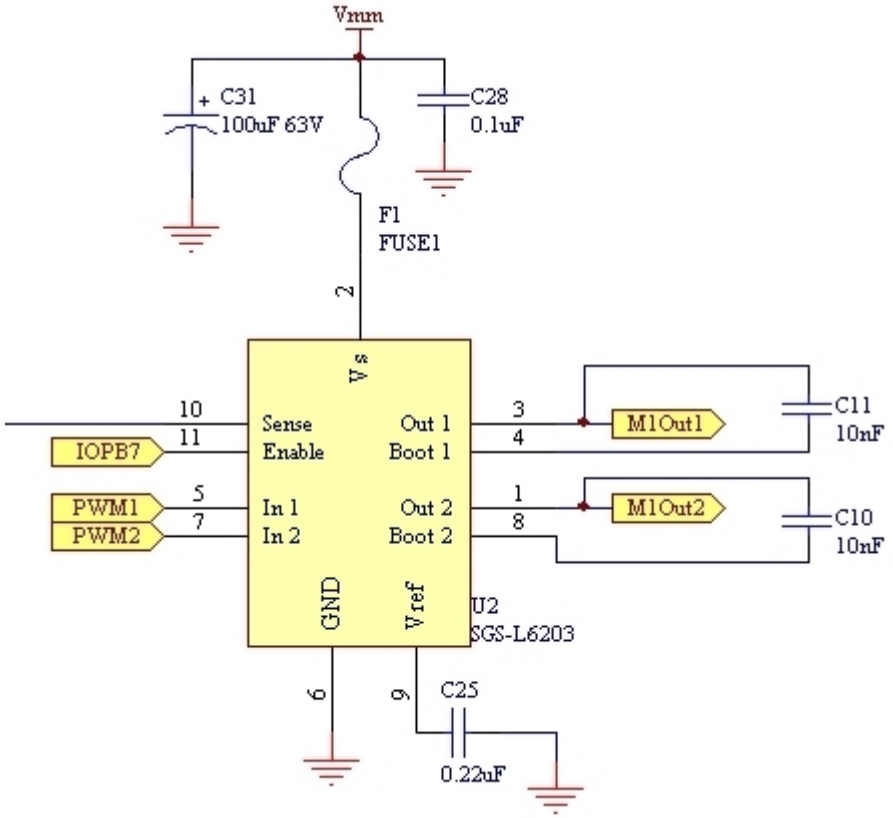


Figure 4.4: L6203 Driver Circuitry

As can be seen in Figure 4.4, the driver circuitry is fairly small, with two bootstrap capacitors that are used to drive the internal MOSFET's to their required gate voltage. These capacitors are used to drive the upper transistors since a voltage greater than the supply is required. There is also a reference capacitor used. A fuse is used with each driver, rated at 5A that is the maximum peak current the L6203 can handle. Each of the three motors is connected through connectors MP1-MP3. Each of these connectors can handle up to 7A, and are easily connected and disconnected.

4.3.1 H-Bridge Driving Techniques

With the driving circuit mentioned above, it was thus required to use the bipolar switching method to drive each of the three motors per board. There is also the unipolar option, but this was difficult and costly to implement with our hardware. The unipolar scheme (Figure 4.5 – adapted from Kennedy, 1999 [4]) works by simply pulsing the motor in the direction that is required. So if the motor was to drive at half speed then a 50% duty cycle would be used in the particular direction required, with the other direction remaining at zero.

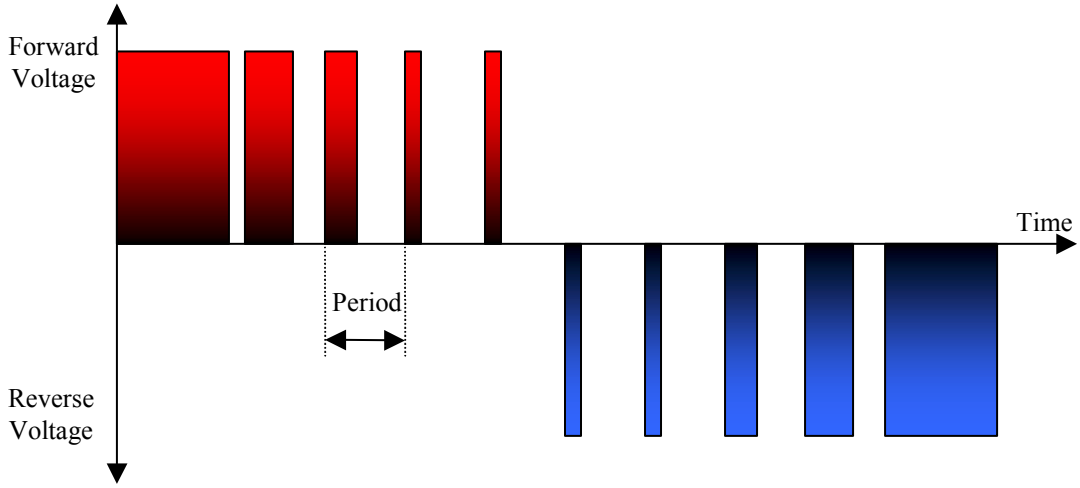


Figure 4.5: Unipolar Switching Method

To keep the motor stationary, there is a 0% duty cycle applied to both sides of the motor. However, since there is no braking system on our motors and it can be required to keep the motors stationary and keep the robot in a rigid position, it is required to have power to keep it there. Due to this, a bipolar switching (Figure 4.6 – adapted from Kennedy, 1999 [4]) method is adopted. This is also due to the limitations of the number of PWM outputs on the TMS chip. To keep the motors stationary, a 50% duty cycle is applied to both sides of the motor, and thus they will cancel each other and keep the motor stationary.

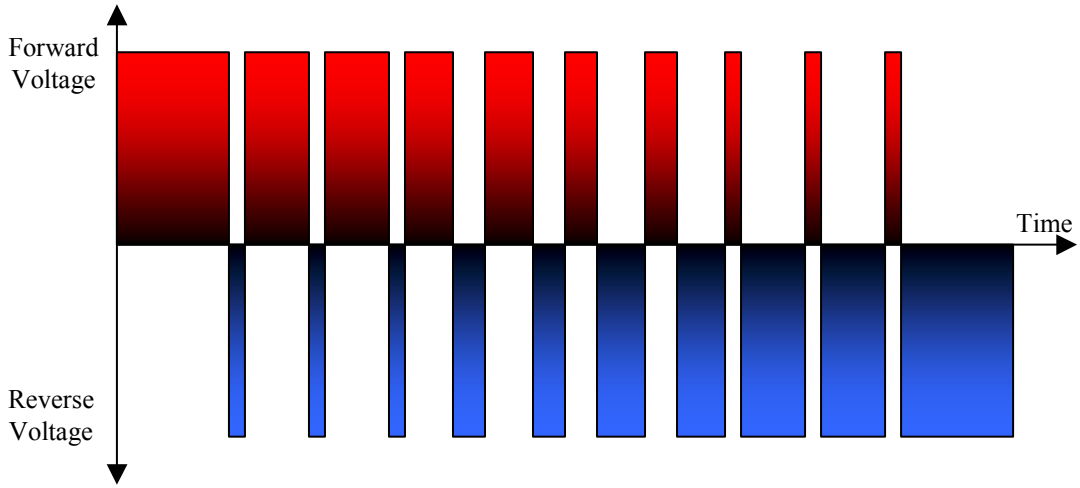


Figure 4.6: Bipolar Switching Method

More power is used with the bipolar system, and there are also large voltage swings, since the voltage will be switched from 42V to –42V and vice versa. These can result in current spikes, which will affect the ability of the current sense resistors. Operating at high frequencies helps reduces these effects.

The bipolar method requires that both opposing MOSFET's be turned on and off simultaneously (left high and right low on and off together). Where as with unipolar operation the low of the respective side can remain on, while the opposite high MOSFET can be switched on and off. Obviously, as can be seen here, since an integrated package is used, individual control of each MOSFET is not possible. The internal schematic of the L6203 (Figure 4.7) is shown here with each of the drive MOSFET's.

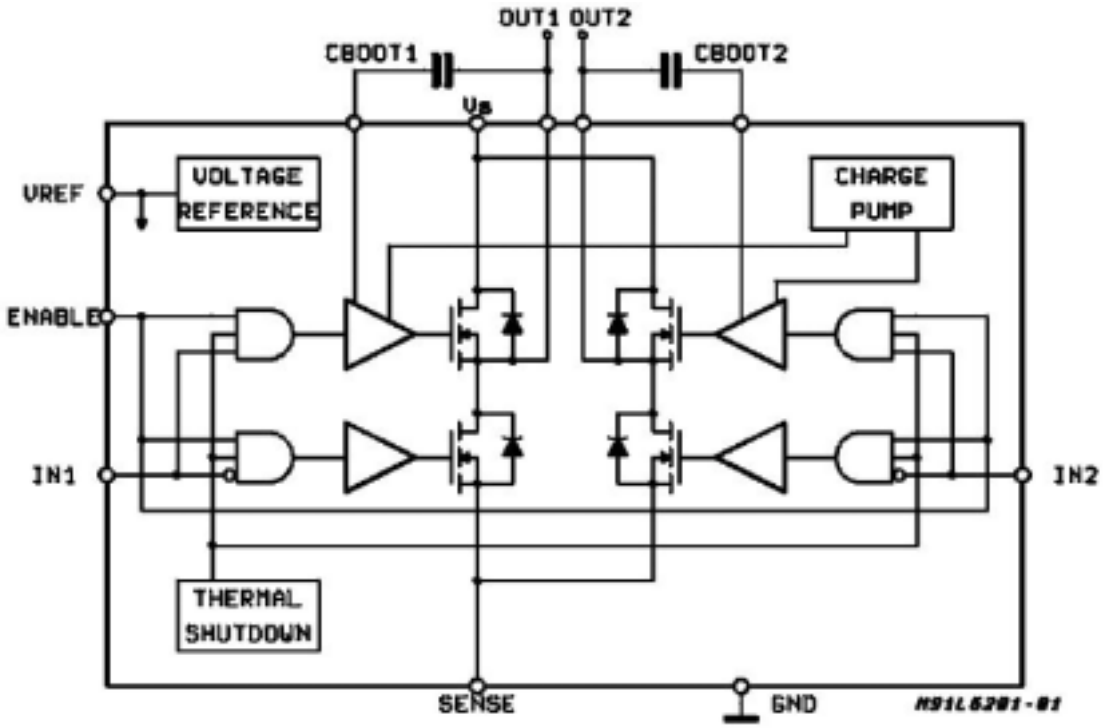


Figure 4.7: Internal L6203 Diagram

Although the TMS320F243 states it has 8 PWM output channels, there are really only five independent channels that can be used. There are three pairs of dependent PWM outputs that have dedicated circuitry, with another two outputs that can operate off the timer compare circuitry. Since there are only five outputs, and six are required for three motors, the use of the dependent outputs is required and thus bipolar switching method is used.

4.4 External Quadrature Decoders

Since there are three motors driven by each controller, position or velocity will need to be known for each, such that an error can be calculated. To do this, there are three quadrature decoders that are required to form closed loop control. This velocity control is obtained through the three-channel square wave quadrature encoders found on each motor. Two channels supply the quadrature signals with the third having a single revolution pulse. Due to lack of peripherals for this, the third channel was not used, or really required.

The encoder will produce two square waves that are out of phase by 90°. The speed of the encoder and thus motor is determined by the frequency of the square waves, and direction is determined by which channel is leading. These square waves are produced by reading the pulses from an infrared LED shooting through evenly distributed slots on a disk on the drive shaft of the motor. Each edge of the signal from both channels is recorded, so for 500 slots, there are actually 2000 counts recorded. This increases the accuracy but will create 4 times the number of overflows of the relevant registers.

Each of the motors and respective gearing forms a gear ratio of 156:1. So if there are 2000 counts/rev and 156:1 gear ratio, this results in 312,000 counts in the register for one revolution of the drive shaft. Using a 16-bit register, there will be 4.76 overflows per revolution. This can be taken care of in software, however there is no need for a complete revolution of any of the joints. The knee will only move a maximum of 51°, and the hip 56°. 312,000 counts per revolution will give an accuracy of 0.0012°.

There weren't many external chips found that could do this job cheaply. Most controllers have one on-board quadrature decoder, if at all. The Agilent HCTL-2016 is such a device for our application, it reads in the two channels and converts these into a 16-bit number. However, there is only an 8-bit data port on the device and thus a control signal for the high and low bytes is required for this.

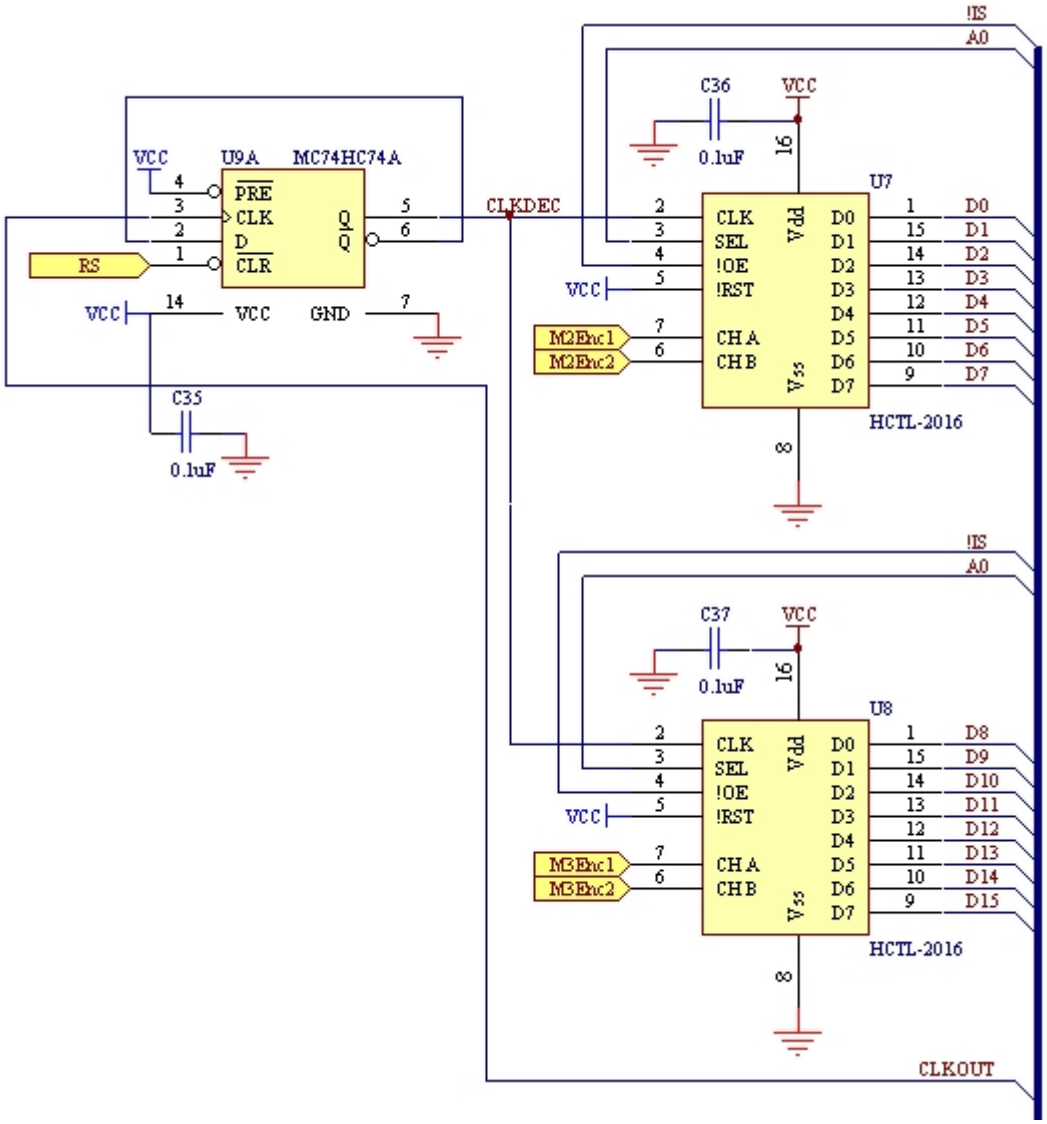


Figure 4.8: External Quadrature Decoders

The address and data port of the TMS320F243 has been used especially for this purpose, as this was the reason for this device over the TMS320F241. The read address of this port is actually a control signal to set which byte is returned on the read. When the TMS320F243 reads from the data port, the !IS pin is asserted (low logic value) to indicate a read from I/O space. This value remains low throughout the read, and thus indicates to the HCTL-2016 to hold the current quadrature value. As soon as this pin is unasserted, the HCTL-2016 returns the output pins to a high-impedance state.

Since the operating frequency of these devices is only a maximum of 14MHz, the clock out signal from the TMS is required to be reduced from its current value of 20MHz. The simplest method was to implement a T flip-flop by feeding back the inverter output of a D flip-flop (U9A). By doing this, the clock has been reduced to 10MHz for clocking the HCTL-2016. This is a suitable frequency for our application. The On Semi 74HC74A D Flip Flop has been used, and it fulfils the requirements easily.

Due to the time delays required for the operation of the HCTL-2016, one software wait state is required so that the data is guaranteed to be stable on the read. The timing diagrams for TMS320F243 data port and the HCTL-2016 is shown in Appendix F. It can be seen that if there is no software wait state, then the data will not be ready when the TMS chip buffers the value. The data will not be available on the HCTL-2016 for a maximum of 65ns, but the data port is expecting this after 30ns, which will not be possible without the presence of at least one wait state.

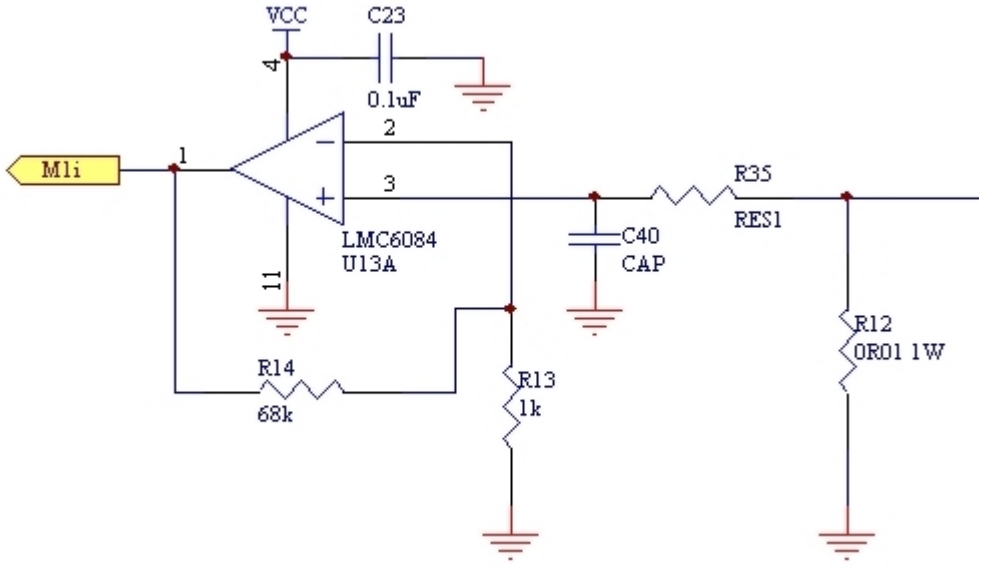
Each of the encoders is connected with connectors J4-J6. They are 4-pin connectors that have a low number of connection repeats, since it was hoped that these weren't the limitation to the system. So testing this is difficult, but in the final controller these will be very reliable connectors. Each of the signals is pulled high with a 3.3kΩ resistor. This will verify that the quadrature decoders aren't counting any phantom signals, and helps in pulling the signal high, for correct operation.

4.5 Current Sensing

As shown in Section *4.3 Motor Driver Electronics*, the maximum continuous RMS current supplied by the L6203 is 4A with a peak of 5A. There are three limits that have been implemented through hardware to prevent over current on any of the motors. The first limit is the use of a software control of this current. The use of a 1W 0.01Ω resistor acts as the sense resistor, with the voltage as a representation of the current through the motor, from Ohm's law or V=IR.

This resistor can handle up to 1W of power due to the current's that will flow through it. So if this resistor had 5A through it, then there would be $P=I^2R$ W of power, which is $P = 5^2x0.01 = 0.25W$. For this same current, there will only be a voltage of $V = IR = 5x0.01 = 0.05V$ across the resistor. This voltage is too small for any use, so this is amplified to a voltage between 0V and 5V where 5V represents about 8A.

Operation is at 100kHz, and the signal is first passed through a low pass filter to filter out ripples that may form at this high frequency. This is an added feature, and possibly isn't essential for current sensing. It will smooth out the signal to be amplified and thus give a better value for the analog to digital converter.



Future 4.9: Current Sensing

This smoothed voltage is amplified with a gain of 69, through the non-inverting amplifier circuitry. The op-amp (U13A-C) forms the basis for this non-inverting amplifier. The gain for this is calculated through simple analysis as:

$$Gain = \frac{V_{out}}{V_{in}} = 1 + \frac{R_2}{R_1} \qquad \textbf{(Eqn. 4)}$$

So using values R_1=1kΩ and R_2=68kΩ, we achieve the required gain of 69. With this gain, we can detect a current of 7.25A, which is well above the peak 5A allowable by

the L6203 driver device. This allows for accurate amplification readings without saturation of the op-amp to its supply rail. The op-amp used is the National Semiconductors LMC6084, which has a very low offset voltage (≈ 1mV), since we will be operating with 0V to 50mV, this is essential for correct operation.

This amplified signal is passed into three of the ADC input channels. These can be monitored and if they reach over a certain voltage, say 2.75V for 4A for the driver, the driver duty cycle can be reduced to help decrease the current.

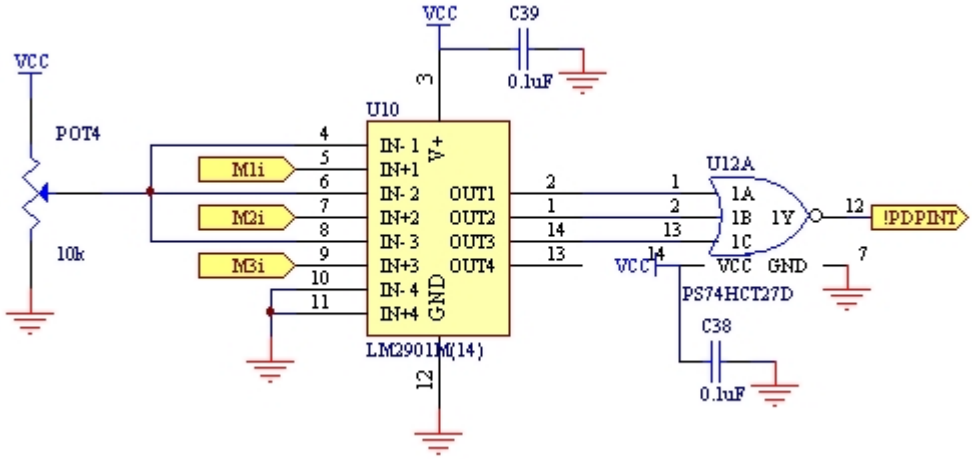


Figure 4.10: PDPINT Hardware

This amplified signal is also compared to a pot voltage and connected to the PDPINT pin, which is an external power drive protection interrupt. If this pin is asserted (low signal) from the NOR gate then all PWM outputs are disabled, this is the second protection device. These are disabled until the TMS320F243 chip is reset. A NOR gate is required since its output will remain high until one of the inputs becomes high, and hence over current. The comparator used is the National Semiconductors LM2901, which again has a small offset voltage, and low power consumption. The NOR gate used is the Philips 74HCT27D, which is high speed gate for our application.

The third protection device is simply the fuses that are placed on each driver chip. They are rated at 5A, however they can handle a non-continuous current spike. If these blow, then this circuitry will not work until the physical fuse is replaced, and will continue to blow until the problem is fixed.

4.6 Temperature and Foot Sensors

With the three safety devices shown in Section *4.5 Current Sensing*, temperature sensors have also be incorporated into the operation of these controllers. These sensors are to be placed on the motors themselves, so if for some reason, these motors overheat, but over current is not detected, these can be shut down on this error.

The commonly used National Semiconductors LM135 has been used for this application. These devices have precision temperature sensing at +10mV/K, and can operate between -55ºC to +150ºC (to +200ºC for a short period). This is a suitable operating range since the DC motors will melt at 125ºC. The TO-46 package has been used for easy connection to the DC motors. It has a maximum error of 5ºC, but a safety margin of 115ºC will be used, above which shutdown will occur, and hence this error won't be significant.

The sensor has been connected as the simple calibration circuit, as seen in Figure 4.11. A 10kΩ pot is used to calibrate the device by connecting it across the output to ground and using the adjust pin. Since the output slope is linear, calibration will cause the output to be calibrated at all temperatures, within its operating range.

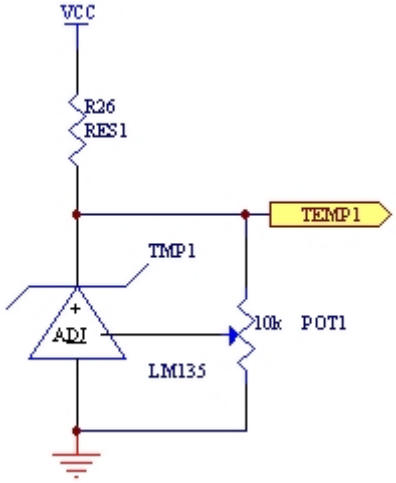


Figure 4.11: Temperature Sensor

Since there are only eight multiplexed ADC input channels on the TMS320F243, it has been required to multiplex the three temperature sensors. The reason for this is the three current sensors have used the other inputs and the four foot sensors (mentioned shortly), leave only one ADC input for temperature. With eventual closed loop walking, the foot sensors have a higher priority than the temperature considering that the current sensors should detect this earlier.

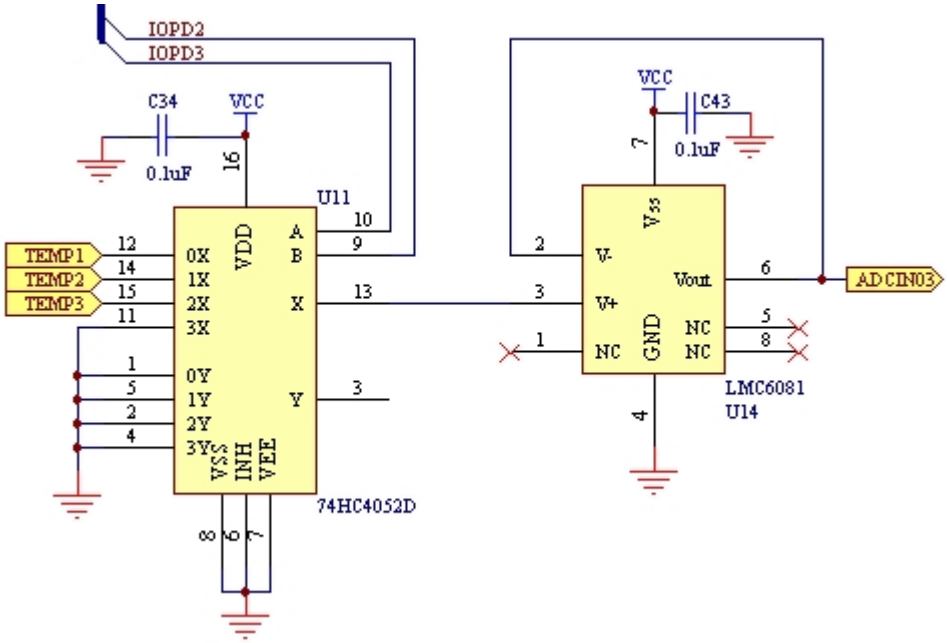


Figure 4.12: Analog Multiplexer and Buffer

To multiplex the three temperature signals, an external analog multiplexer has been used. The On Semi MC74HC4052D analog multiplexer has been used here. The device is a dual 4 input analog multiplexer with common select pins. Two pins of Port D control the select lines, and ADC channel 3 receives the multiplexed analog value.

The input source impedance to the ADC (from the analog multiplexer) on the TMS chip is to remain below 10Ω for conversions to remain within specifications. Although our device had the lowest that could be found at 240Ω, this wasn't enough for the TMS320F243. To do this, a voltage follower or buffer to reduce the resistance to an acceptable level follows the multiplexer. The National Semiconductors LMC6081 op-amp was used here which is the same as the LMC6084, just a single op-amp package, as only one op-amp is required here.

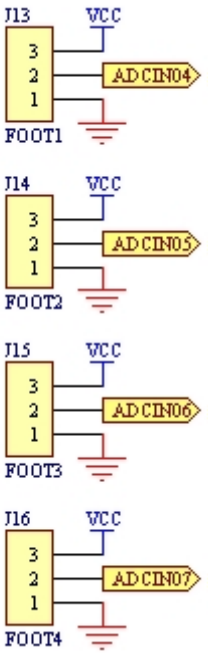


Figure 4.13: Foot sensors

The foot sensors connect directly into the ADC input channels 4-7 from connectors J13-J16. Each sensor is supplied with V_{CC} and GND, and the respective analog signal is returned from the sensor. These sensors are only used on Boards 1 and 2, since these are the only two with control of the feet. Boards 3-5 have this as redundant circuitry, but these could be used later for additional sensing.

Each foot has four sensors on the under side, so that the distributed pressure of each foot can be calculated. It is anticipated to have analog sensing here, however, a switch may only be possibly used due to the cost of these pressure sensors. The distribution of foot pressure will be used by the iPaQ for balance and walking calculations.

4.7 The Control Network

There are seven nodes on our network and it is required to have a high speed, reliable and robust network with which to transmit our control data. These nodes include the five DC motor controller boards, the servo board and the power/USB board. The network is linked to the iPaQ through its Universal Serial Bus (USB) connector, which can send data at up to 12Mbit/s.

As already shown, the CAN network protocol will be used. It is especially useful in this application due to its high speed and reliability. We will be using the CAN2.0A specification, which allows for up to 2048 nodes (11-bit arbitration field) to be on the network. Although the arbitration field is used for message filtering, and not as specific nodal addresses, we can set up the message filtering to act in this manner. The use of an emergency broadcast is set for the highest priority, to enable the shut down of the robot on any faults.

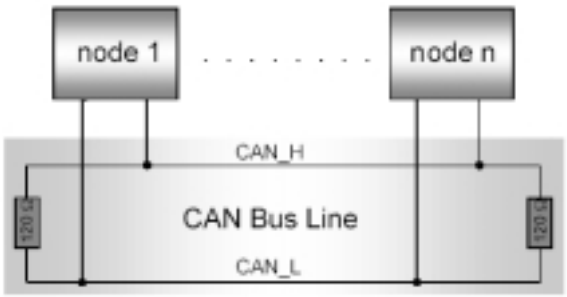


Figure 4.14: Simple CAN Network

The TMS320F243 has a CAN network interface which when combined with a CAN driver, produces a powerful controller networking tool. The CAN network is a bus network, with each node connecting to the CAN-H and CAN-L lines of the two wire network. It is terminated at either end with a 120Ω resistor, to minimise reflections at the ends of the bus. A jumper JP1 is supplied here, so that any node can be the terminating node, by simply connecting the jumper. This is useful for testing the boards and the network. Two connectors are used for the two-wire network, this allows the easy addition and removal of subsequent nodes. The Philips PCA82C250 CAN driver has been used in this application. It can operate at up to 1-Mbit/s, which is the limit of the CAN protocol.

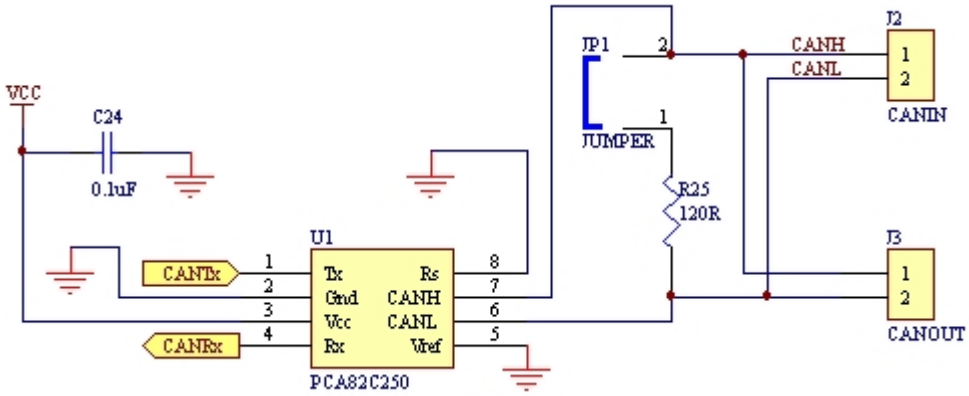


Figure 4.15: CAN Interface

This CAN network is described in more detail by Cartwright, 2001 [3]. The full coding for the CAN network is also shown within this thesis.

4.8 Miscellaneous Hardware

To supply the digital side of the circuitry, including controller, all logic devices, sensors etc are supplied with a simple 7805 step down voltage regulator. This device is only about 80% efficient, but will only use about 0.4W (200mA x 2V), which is quite insignificant, compared to power drawn through the motors. It is very simple and can provide significant current, up to 1A, for the digital circuitry. The National Semiconductor LMC2940C is used in this application. The unregulated +7V is

connected through J1. The regulated +5V is then supplied through the power plane to all of the required circuitry.

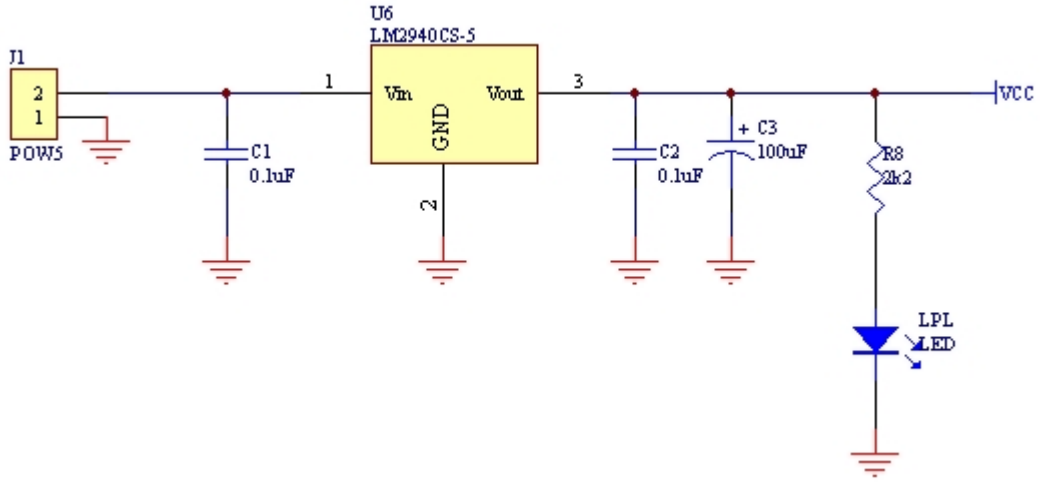


Figure 4.16: +5V Regulator Circuitry

The power circuitry is supplied with an unregulated power supply from the power board. This voltage can vary between 30-40V dependent on battery charge and load. This power is connected into connector POW42. However, this output voltage is variable and can be adjusted on the power board depending on requirements.

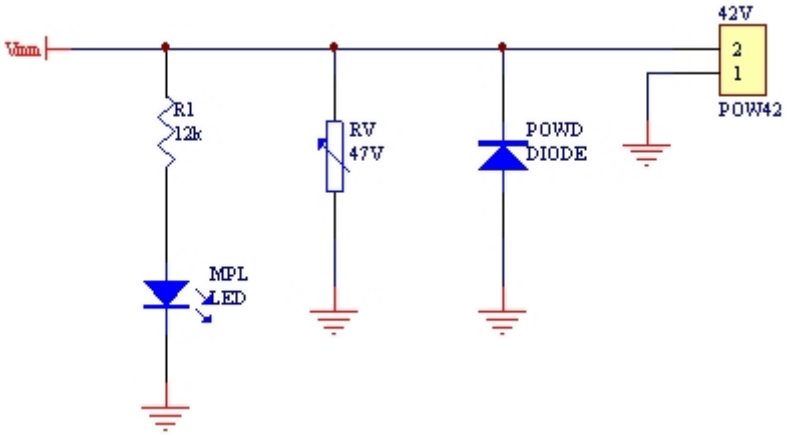


Figure 4.17: +42V Input Circuitry

There is a protection diode (POWD), however this will not work without a fuse directly next to it, to prevent reverse power connection. There is a varistor that will prevent over voltage by shorting at over voltage. For the best over current, reverse voltage and over voltage protection, a power fuse should be supplied. However, this is not really possible since there are three devices powered, and a 12A fuse would be useless if one driver was drawing 6A, and hence would blow up the driver but not the fuse. To fix

this, individual supply connectors for each driver and hence fuses would be required, but due to our limited space this was not possible.

To cover these problems there are few features on the power board that should be mentioned. There are 10A fuses and current sensing that protects each battery. There is also voltage sensing on each power output. Potentially short circuits can be detected with a large voltage decrease. For further information see Appendix A for Brewer's thesis on "Power System for a Humanoid".

Two power indicators have been added to allow easy detection of power-on on the boards, MPL and LPL. Three testing and indicator LED's have been included for troubleshooting TL1-3. Having three LED's allows for eight combinations of status that can be used for troubleshooting. A pushbutton SW1 has also be included for testing and perhaps future use connected to one of the external interrupts.

Considering the set up of the quadrature decoders for position/velocity control of each joint, there is no resetting of these external devices. This would have added hardware, which can be eliminated in software. Instead the power on value is used as the reference point and the limit switches used so that in calibration, the limits of each joint is tested and recorded. To do this, one joint at a time is moved until the limit switch triggers the interrupt, the values for this joint is then recorded. Each of the quadrature decoders is reset on power reset, but the initial physical position with reference to the joint is unknown to the controller.

This can also be used as a safety feature, so that if a motor over shoots its joint boundaries, the limit switch of the joint is hit, and causes an interrupt. In this situation, the quadrature decoders haven't measured this distance match with the limit value, and hence an error has occurred. At least that driver will be shutdown and perhaps the other drivers. An emergency broadcast can then be sent to warn other controllers of this condition and also the iPaQ.

There a few control pins that need to be set on the TMS320F243 for correct operation. These are seen on the schematic shown in Appendix B, they aren't all explained, but for more information see the TMS320F243 data sheet [10]. To use the external data bus, it is required to set ENA_144 (pin 18) high to allow use of this data bus. READY (pin 44) is pulled high through a resistor, this makes sure that on a read of the data bus, to the TMS chip the external device has data ready, and the use of wait states prevented a read straight away. MP/!MC is pulled low, to boot the controller from on-board memory rather than external memory (in use by quadrature decoders).

4.9 PCB Design and Construction

The schematic for this controller has been shown in great detail above, but to create a Printed Circuit Board (PCB) from this uses a bit of skill and creativity. Each of the five DC motor controllers is identical. This would reduce costs, since only a few designs could be on each PCB panel (each panel was shared with other projects), and also simplicity for design. Obviously five differently designed boards would take five times as long to create a PCB for. As shown above, some circuitry becomes redundant, the foot sensors are the one main example.

The servo controller board was a completely different design since servo motors were being controlled, and these only require a position input value, as apposed to a dual PWM signal. Due to this, the drive circuitry wasn't needed, just a simple buffer to supply the servos with the correct current. See Cartwright, 2001 [3] for the design of this board.

Figure 4.18: Servo Controller Board

To fulfil the second specification, i.e. board size restriction of 170mm x 100mm x 40mm, it was required to make the boards as small as physically possible. This would enable each of the boards to be

placed as required in Figure 3.2. Considering the size of the produce PCB to be 127mm x 88mm x 25mm, this met the required specification fairly easily, with possible reductions in size still possible. One of the methods employed to reduce PCB space was to use as many surface mount components as possible. This effectively reduces the space required to nearly a half, since there are no through-holes impeding other devices.

Layout of these devices and there position on the board was one major concern with the design of this board. It was required to have the board to interface to

- 2 x 2-pin power connectors (+5V and +42V)
- 3 x 2-pin motor connectors
- 3 x 4-pin quadrature encoder connectors
- 2 x 2-pin CAN connectors
- 3 x 3-pin temperature sensor connectors
- 4 x 3-pin foot sensor connectors (redundant on boards 3-5)
- 6 x 2-pin limit switch connectors
- 1 x 8-pin serial loader connector
- 1 x 14-pin JTAG connector

To add to this, it was required to have two power supplies within the board, one for the low current +5V digital circuitry and the other for the high switching current +42V analog circuitry. Due to the switching and high currents of the analog circuitry, noise is produced which can affect the operation of the digital circuitry. To help minimise this, the digital circuitry is located at one end of the board and the analog circuitry the other.

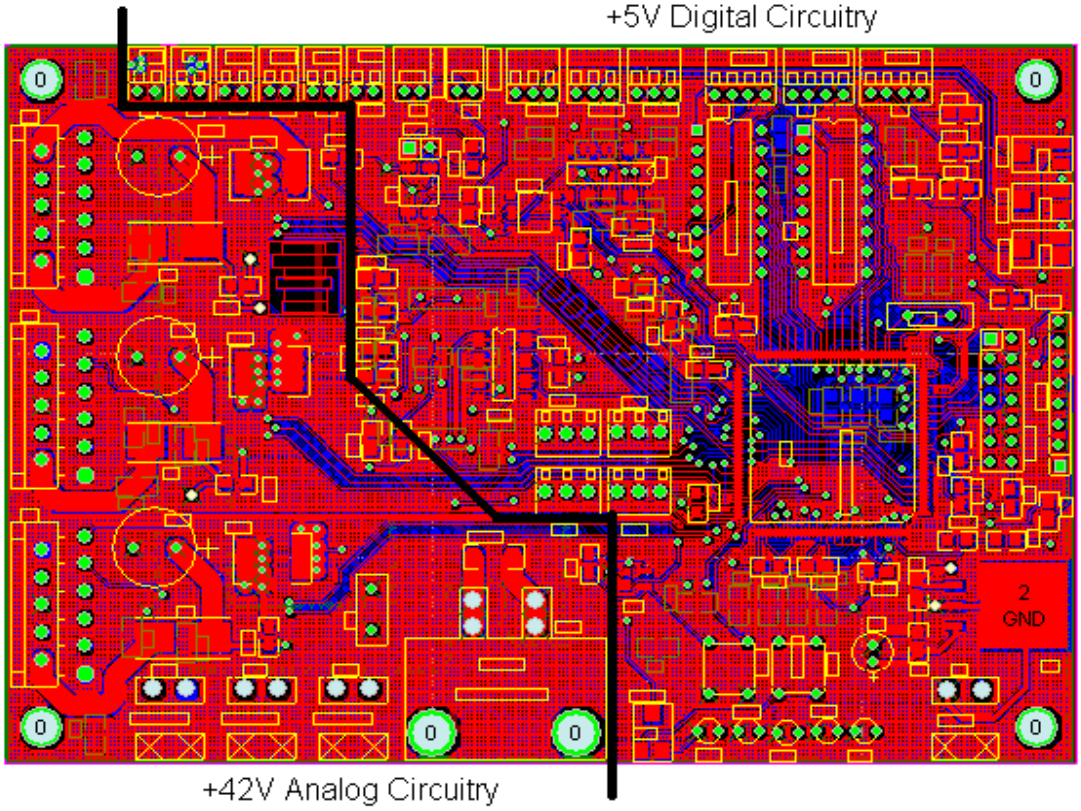


Figure 4.19: PCB Showing Split Planes

A four layer split plane PCB is used to further reduce noise and help minimise PCB space. There is top and bottom signal layers, a ground plane, and a split power plane. The ground plane is not split between analog and digital circuitry, as this would have been more difficult to implement and would lead to little noise improvement. The power plane has been split between the digital circuitry and the analog circuitry. +5V flows through the right ¾ of the board, containing the digital circuitry (as per Figure 4.19) and +42V through the left side. There are also a few large tracks through the power plane that are used to connect the drivers to the motors. This left room on the top and bottom layers for connections without as much current and hence less noise produced.

To implement this design on a conventional two-layer board would require large width tracks to carry up to 12A for the power circuitry, and 4A to each motor. This would produce significant noise on the board. These power planes have significantly reduced the noise and also made the layout of the board far simpler. This same principle was

used to connect drivers 2 & 3 (U3 and U4) to their respective motor connectors, to reduce noise and ease layout of the board. The TMS320F243 has been placed as physically as far as possible from the high switching current analog circuitry, so as to not inhibit operation due to noise.

Each of the IC's on the board has been decoupled using a small 0.1µF surface mount capacitor. These are placed as close as physically possible to the supply pin of each device. A large 100µF 63V capacitor has been connected close to each of the driver chips. This helps reduce noise and also helps supply the driver with the required current for high current switching, due to the bipolar switching method.

The connectors have been placed so as to be relative to the device with which they are connected. The board is placed within the leg, with the long edge horizontal (this orientation is seen in Figure 3.2), which allows space for all of the required connectors. The driver chips have been placed on the vertical sides, which can be mounted on the frame of the robot for heat dissipation. This may negate the need for heat sinks, this is yet to be tested.

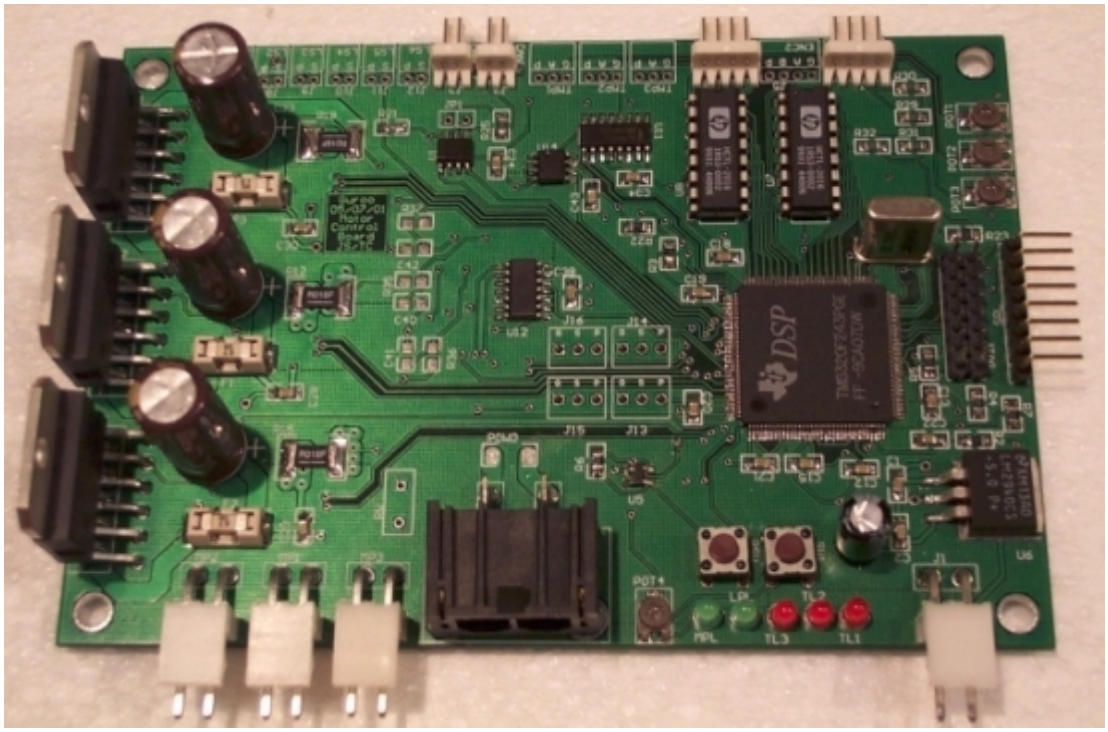


Figure 4.20: Populated DC Motor Controller Board

Each board drives motors both above and below the controller board. However this differs between boards and hence the three motor drivers and power connectors have been placed on the low edge of the board. All of the quadrature encoders, limit switches, CAN and temperature sensor connectors have been placed on the top edge (all edges are relative to Figure 4.20, with actual orientation to be selected). This helps keep noise from these signals as well as keeping all digital circuitry on one side of the board. Unfortunately there wasn't enough room to place the foot sensor connectors on an edge, so vertical connectors were placed in the middle of the board.

The serial connector has been placed on the right vertical edge of the board. This allows for easy programming whilst out of the robot and possible programming whilst in the robot. Nevertheless, with further software coding each board should be programmable through the CAN network from the iPaQ, which means software can easily be changed without the removal and subsequent replacing of each controller board.

Chapter 5 – Software Design

All of the software designed for this system is micro controller software. Higher-level control is taken care of by the iPaQ, and this is not discussed here. The micro controller software is to mainly drive the control algorithms of each motor required by the higher-level software. Each board is to control three motors with feedback from the quadrature decoders and other sensors. This simple feedback with higher-level control is shown in the figure below.

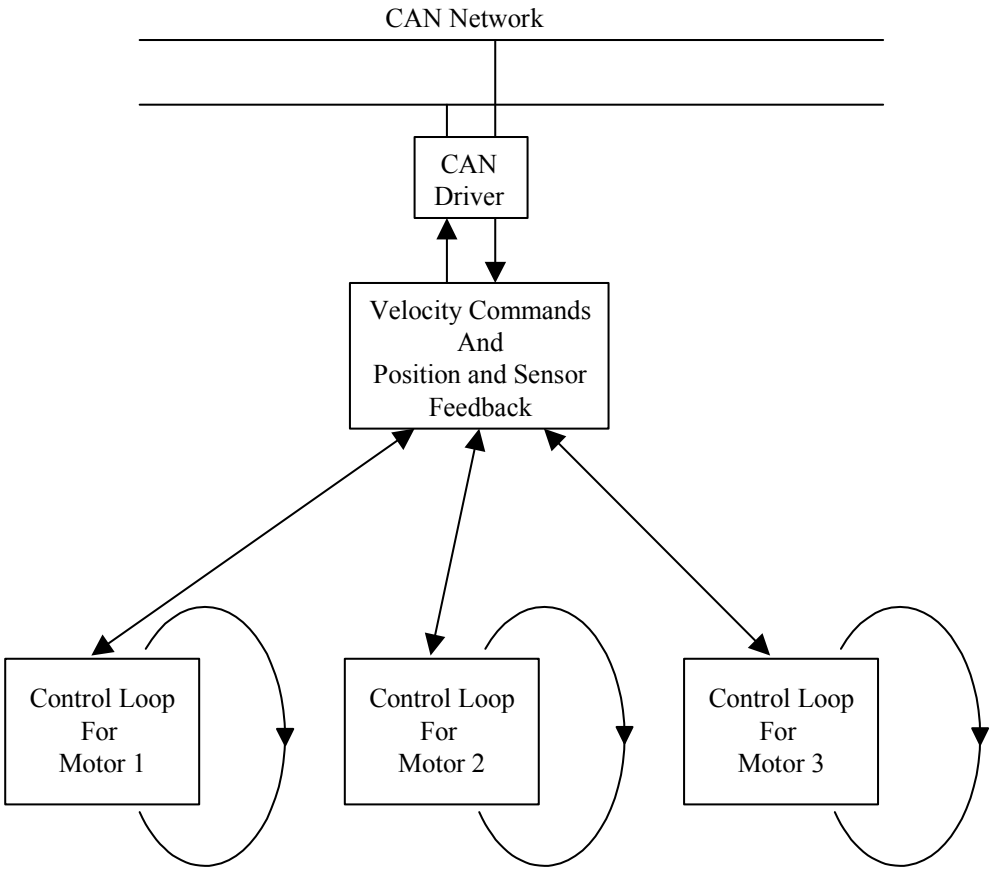


Figure 5.1: 3 Motor Control Loops

5.1 Software Overview

It has been decided that the higher-level control would operate at a frequency of 500Hz. However, low-level control will operate at 2kHz (four times the incoming velocity commands). Since the network is operating at 1-Mbit/s, half of the network bandwidth will be used for control commands, and the remainder for feedback, such as motor positions and other sensor values.

Control of each board will consider incoming velocity commands for each motor as well as the control of each motor on this request. Figure 5.1 shows a block diagram of the controlling algorithm. Control is interrupt driven, with each of the Interrupt Service Routines (ISR) shown. A lot of the execution time is spent inside the main loop, where nothing is happening, just waiting for an interrupt. An interrupt occurs when the required hardware "interrupts" execution and calls a vector to the ISR. Once an interrupt goes off, then the needed service routine is called. After execution has finished control is passed back to the main loop.

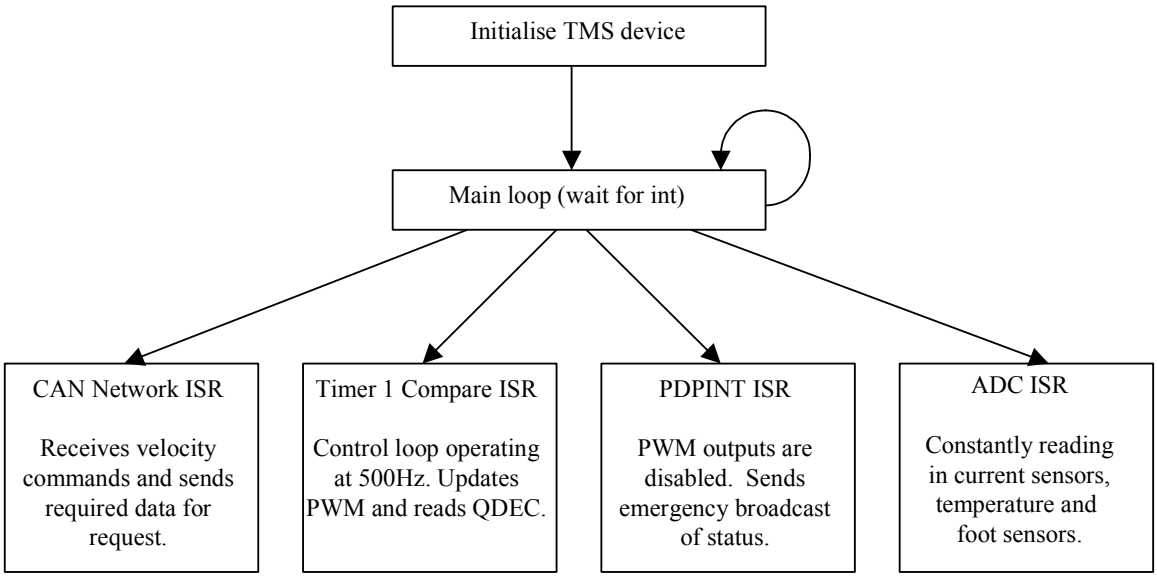


Figure 5.2: ISR Block Diagram

The software shown in subsequent sections is for the testing program Loop1.C as seen in Appendix D. A listing of the code is seen here, and this is the basis for the following software description. This program, controls one motor with the use of the internal

quadrature decoder as feedback. The encoder difference between loops is utilised and this is multiplied with a gain value to calculate the needed PWM value for the motor.

5.2 Board Initialisation

There are several peripherals that need to be initialised and set up for correct operation in this application. These include control of the PWM and quadrature decoders. This also includes the initialisation of several control registers within the TMS chip itself.

For ease of testing, the watchdog timer has been disabled. For the demonstration this will set up to reset the device after 6.55ms (the smallest allowable watchdog overflow period). Since the control loop will operate at 2kHz, this will allow enough time to do all of the control algorithms, and other code, and then kick the watchdog again. If this timer overflows then the TMS chip is reset. This stops the device from entering 'Phantom' interrupts that aren't recognised by the system.

After the timer (Timer 1) is enabled for PWM output, the period register is set to 200_{int}. This sets the PWM output frequency to 100kHz (20MHz/200), so there is an output pulse on all six outputs every 10μs. This frequency was used to keep the current drawn and ripple to a minimum. This frequency is the limit for the driver chips, and due to the size of the period register, the resolution of the output PWM is limited. Since bipolar switching is used, there are essentially 100 forward PWM values (50-100% duty cycle) and 100 reverse PWM values (0-50% duty cycle). So there is a resolution of 0.5% for the duty cycle or 7.6bits.

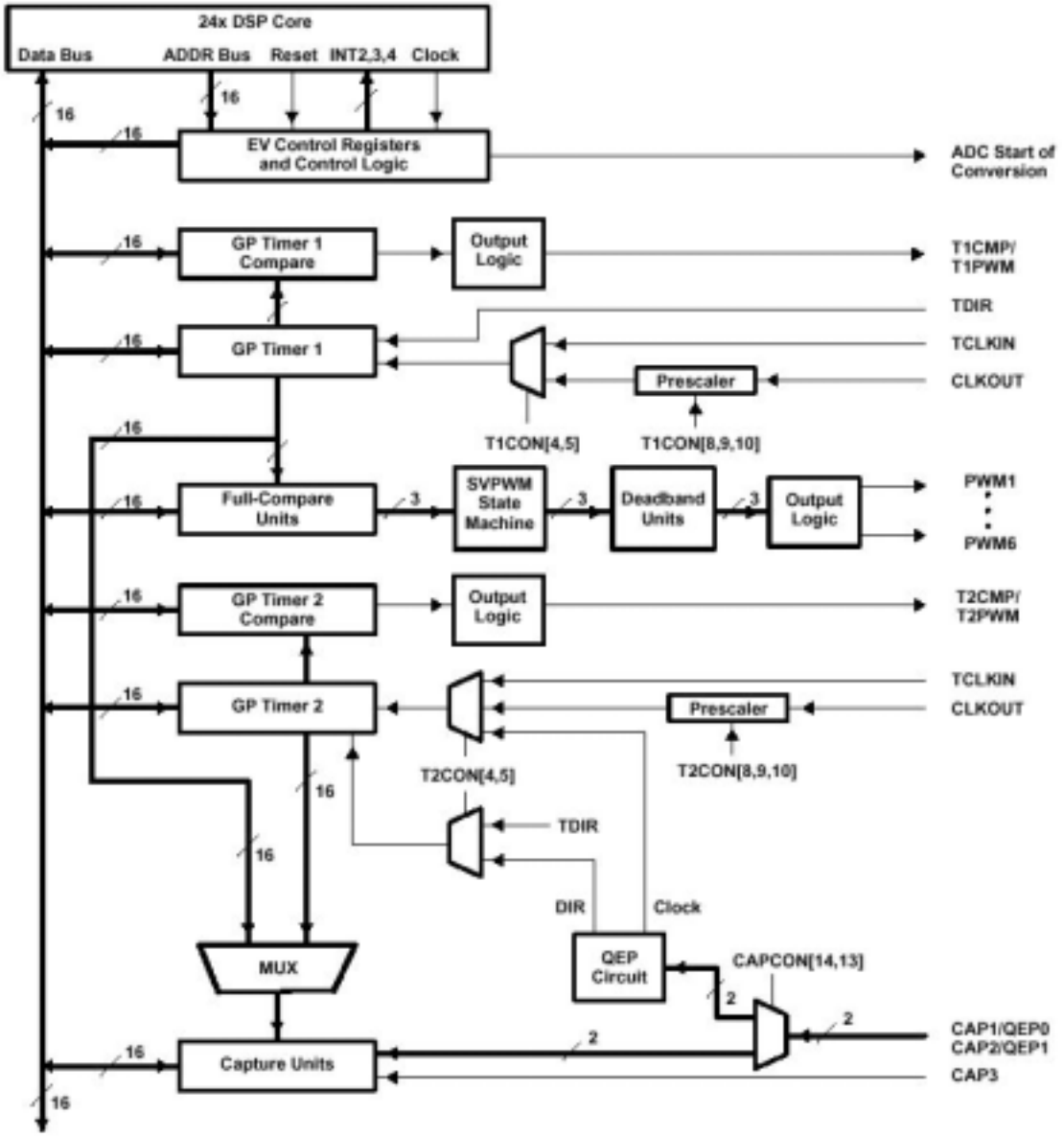


Figure 5.3: TMS320F243 Event Manager

Timer 1 is used to control the PWM outputs (Figure 5.3 above). On each compare of Timer 1 to the Timer 1 period register (T1PR) a PWM duty cycle is initiated. The dead band generation registers are also set up at this point, they are set for minimal dead band, since the L6203 driver chips have there own dead band generation.

It is also required to have the multiplexed I/O pins set to the correct inputs/outputs, since all special pins are multiplexed with an I/O port. However, there are six dedicated

pins on Port D that can be used specifically for I/O. All special hardware including CAN interface, PWM and QDEC all use multiplexed I/O ports.

Timer 2 is used to control the internal quadrature decoder. This is connected to CAP1 and CAP 2 (pins 121 and 123). Timer 2 period register (T2PR) is set to the maximum of $FFFF_{hex}$ to allow for minimal number of overflows. T2CNT register is used to show the current position of the motor.

5.3 Timer 1 Interrupt Service Routine

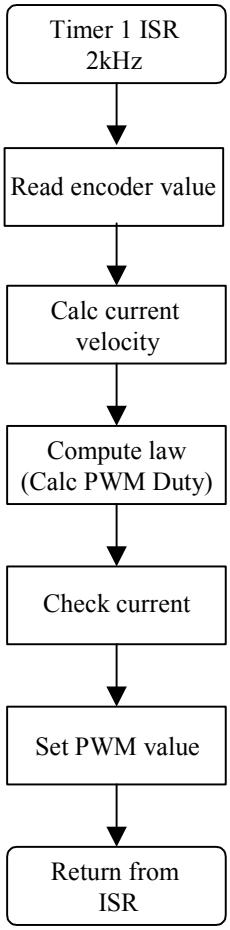


Figure 5.4: Timer 1 ISR

As can be seen in the Loop1.C in Appendix D, operation has initially used polling instead of interrupts. As explained previously, all code should be interrupt driven, with a lot of the time spent waiting for an interrupt to occur.

The intended Timer 1 ISR operates the control loop shown in Figure 5.4. This loop operates at 2kHz per motor. There is a new velocity command from the CAN network at a frequency of 500Hz, however, the controller loop operates at four times this. This loop will be run three times within the 2kHz slot due to the three motors per board.

The 16-bit encoder value is read in and this is used to calculate the current velocity due to the difference in encoder counts between loops. This is compared to the intended velocity command and the required PWM duty cycle is then calculated. The current is then checked to see if there is over-current and the PWM cycle is adjusted accordingly. The PWM value is then written to the required register and then control is passed back to the main loop.

5.4 Main Loop

As can be seen in the control section of the main loop code below, there is a delay created to calculate the difference between encoder counts on each cycle of the loop. This delay has been experimentally found to be the most effective for the encoder and motor with which the board is connected. T2cnt_old holds the previous value of the Timer 2 count register. This is subtracted from the current value to find the difference. This is scaled by –0.0488 and added with 100 to make it a value between 0_{int} and 200_{int}. This will then give a duty cycle between 0-100%. The scaling value has also been found experimentally and allows the motor to move significantly but not to draw too much current from the supply.

```
for (temp = 0; temp < 500000; temp++);

/* Try polling first, using delay above for difference */
t2cnt_sub = T2CNT - t2cnt_old;
cmpr1_val = (-0.0488 * t2cnt_sub) + 100; /* scale 0-100% duty */
CMPR1 = cmpr1_val;

t2cnt_old = T2CNT;
```

5.5 Intended CAN Interrupt Service Routine

As has been previously shown the CAN network is used to communicate between the iPaQ and each of the boards located throughout the body of the robot. This CAN network has been described in Section *4.7 The Control Network*. CAN2.0A has been used for this application. The arbitration field is 11-bits in length, and hence allows up to 2048 message identifiers, nodes and priority on the network. There are four different types of frames that can be sent over the network. These are Data Frames, Remote, Error and Overflow Frames. Data frames are used to transmit standard data, and the remote frame is used to request this data with the same identifier.

The TMS320F243 has many memory mapped registers that are dedicated to the operation of the CAN interface. There are six dedicated mailboxes for use with the

CAN network. Two transmit mailboxes, two receive and two configurable mailboxes. Each of these mailboxes is 8-bytes in length, and hence spans eight bytes in memory. A CAN data frame can contain from 44 to 108 bits. This includes the 8-byte data field as well as the 11-bit arbitration field and other control fields. The other fields are seen in Figure 2.5. These are taken care of in hardware, and will not be discussed. To send data, the data is inserted into the required mailbox, and required arbitration field is set, and the hardware will send off the message, taking care of errors etc.

A new CAN message will arrive at a frequency of 500Hz (or every 2ms). On complete reception of the new message the interrupt is asserted. This should coincide with the Timer 1 interrupt used for the control loop. This new velocity command is processed and used to adjust the PWM duty cycle of the required motor driver.

5.6 Other Intended Interrupts

There are two other interrupts that should be considered in the design of the controller. Although these weren't implemented for the submission of this thesis, they are intended to be completed for the Innovation Expo.

5.6.1 PDPINT ISR

As described in Section *4.5 Current Sensing*, the PDPINT pin is connected to a NOR gate which is driven by the current sensing of the motor drivers. When any of the three drivers exceed the allowable current this interrupt is set and the PWM outputs are automatically disabled.

Within this interrupt routine, it is also required to use the CAN network to broadcast an emergency message of the presence of over-current in one of the motors. This will be inspected by the iPaQ, and action can be taken from here. This may only require the already shutdown action of that controller board, or it may incur the shutdown of the whole system.

5.6.2 Analog to Digital ISR

As shown in Section *4.5 Current Sensing* and *4.6 Temperature and Foot Sensors*, all of the eight multiplexed ADC channels of the TMS320F243 have been utilised. Current sensing, foot sensors and also temperature sensors use these ADC channels. Since there are a total of 10 analog values to be read, it is required that the on-board ADC unit of the TMS320F243 be utilised at all times. This will produce the greatest benefits of the system, in both safety and sensing.

As current sensing is the most important sensor to be checked, this will be checked more regularly than the other sensors. Port D is also required to switch between the temperature sensors due to the external analog multiplexer. It would probably be advised to check the three current sensors, read in half of the foot and temperature sensors, check the three current sensors again, and finally the remaining sensors. This loop will continue indefinitely.

Chapter 6 – Project Performance and Testing

At the time of writing this thesis, the controller had been semi-constructed and initial testing software written. The main components required had been soldered to the PCB, tested and shown to be satisfactory. It was not possible to test some of the circuitry since not all components had arrived by the time this thesis was written.

All of the required circuitry was in place except for a few of the connectors required for the temperature and foot sensors and limit switches. Also the LM2901 quad comparator had not arrived despite attempts to get this device. This provides the basis for the PDPINT pin NOR gate circuitry, and thus in an emergency the ADC will be relied upon for shutdown. This shouldn't be a problem as full load will not be required on any of the joints, however, it is hoped that this device arrives soon, so this circuitry can be tested for the Innovation Expo.

6.1 Hardware Performance

It was originally hoped to have the humanoid ready for the soccer championships in Seattle in August 2001. However, due to problems building the chassis for the humanoid, programming problems of the controller boards and the project as a whole, this was not possible. This decision was not made until the beginning of June, and because of this the hardware design process had been accelerated somewhat.

The Printed Circuit Board (PCB) had been designed and constructed by early August. Due to this early construction it was hoped that the software would be completed by the end of second semester. Due to the cost of the boards it was only possible to have one PCB manufacture run and this required complete accuracy on the first design of the boards. This is also one of the reasons that the PCBs weren't designed and constructed earlier. If these PCBs weren't going to work it would be almost impossible to test the required components on either breadboard or other prototyping methodologies due to high switching currents.

Fortunately, the PCB was indeed designed and implemented correctly, and to date no problems have been encountered. The external quadrature decoders aren't working, however they have only been tested once with code that is unknown to be working. The major blocks of the circuitry that were required are working and fulfil the design specifications to which they have been tested.

6.2 Software Performance

As has already been discussed, the software hasn't been completed at the time of writing this thesis. It is hoped that a simple PI controller will be implemented for demonstration at the Innovation Expo. Simple implies that only the bare requirements for this control will be used, this includes the motor drivers and quadrature decoders. This will allow the motors to be driven and also feedback provided through the quadrature decoders. The over-current sensing features of the boards will be used, as this is a required safety option for the boards. Time permitting other features such as temperature sensors will be added to this controller.

All of the software shown in Appendix D was written in 'C'. 'C' was used since due to the code creation time required to write assembler, 'C' was a far better option. Assembler would produce far more efficient code to be run on the micro controllers, but since the control loops will only be operating at 2kHz, this isn't really a problem. If it was running at 5kHz-50kHz, this could have been a problem. At 2kHz, this allows 500µs for each control cycle to be executed. Each instruction is executed in 50ns, which will allow for 10,000 instructions per cycle. It should be noted that these are assembler instructions and some instructions require multiple instruction cycles. Considering this though, this is ample time to utilise the coding features of 'C'.

Once the programming problem was overcome (described in Section *6.4 Project Weaknesses or Problems*), late in second semester code generation begun, however it was too late for the intended PI controller to be written for this thesis. In the small amount of time used, several small programs had been written to test the individual

sections of the controller board. These include a flashing LED program, simple PWM drive, internal quadrature decoder use and also a simple feedback program utilising both the PWM drive and internal quadrature decoder. To date, the external quadrature decoders aren't operational, this is yet to be extensively tested.

Each of these simple programs ran well on the TMS chip, it was actually required to insert delays into the code to slow it down for testing purposes. Each of the programs is written in a modular form, which makes code generation in the future far simpler. In other words each program has a main loop, and all other code including initialisation is done in separate methods. So for the implementation of the PI controller, it will require copying the needed methods and inserting the new code for the PI control. One of the advantages of writing the software in 'C' is its adaptability, and this can be utilised here.

6.3 Overall System Performance

A simple feedback program has been demonstrated to work. This utilised one of the PWM output channels and the internal quadrature decoders. This was tested on a motor of one robot with the quadrature encoder of another robot. The actual motors to be controlled arrived two days before the submission of this thesis, and due to this an alternative was required for testing.

It was shown that when the encoder wheel was moved in one direction, the motor would drive in the opposite direction to overcome this disturbance. The encoder difference between loops is utilised and this is multiplied with a gain value to calculate the needed PWM value for the motor. The drivers could handle all voltages and currents unless limited by the power supply.

From this working, a simple PI control is very likely to be created in time for the demonstration. However, to control three motors per board it certainly will be required to have the external quadrature decoders working.

6.4 Project Weaknesses or Problems

One of the major setbacks with the software creation for this project was the programming of the devices. After the PCBs had been created, soldered and tested, there was a problem programming the boards. Programming serially (as shown in Appendix E) was a problem, and for some reason JTAG would not correct this problem. It appeared as though the jump vectors were being incorrectly loaded in memory, and all that was required was the serial boot loader to be programmed back, and then serial programming could be utilised from this point.

Since the boot loader for these devices resides in the internal flash memory, each time the device is programmed, the serial boot loaded is reloaded into memory. So if this were corrupted during programming, the device would not be programmable serially until the boot loader was reloaded. Fortunately the JTAG pod could be used to do this (this is shown in Appendix E).

Since most of the hardware software interaction was not completely tested, it is difficult to say where the problems in the system are. The hardware works to its desired specifications, and initial software also works to specifications. Unfortunately the external quadrature decoders aren't currently working, however it is hoped that they will for demonstration.

Chapter 7 – Future Work and Conclusions

This project was a successful hardware design project with limited software success. These limitations can be attributed to programming problems faced early in semester two. These were eventually overcome with a simple solution, which allowed for these devices to be programmed and tested to their intended design.

7.1 Future Work

Considering that the hardware and software wasn't completely integrated for this thesis, there is a vast field of work that can be extended to improve the humanoid controllers.

- It is anticipated that a simple PI controller will be implemented for the Innovation Expo. It is obvious that the completion of the software control system will be required for open loop walking by the humanoid.

- Experiments will need to be run to confirm that the drivers can support the selected DC motors chosen for the project. The expected demonstration of this system will involve control of the leg and hips of the robot. These won't be with the full load of the upper body and the requirement for balance. Under full load from the upper body and friction from the ground, the ability of the humanoid will be inhibited significantly.

- If larger currents will be required for the more strenuous joints such as the knee and hip, a semi discrete driving solution may be required. This will require more hardware and hence a new PCB. The size of PCBs may be a problem with this, and this will need to be considered.

- Extending the software to implement a full PID compensator will be necessary for the robot to be successful. This will allow the system to be optimised for

both transient response and also steady state error. This can also implement the use of temperature sensors, foot sensors and limit switches.

- Redesign of the hardware could be considered using another later micro controller. A suggested device is the original intended device being the Motorola 68376. This would remove the bottleneck of the external quadrature decoders.

- Programming of the micro controllers through the CAN network would be a very useful feature. It is then not necessary to remove the controller boards from the robot chassis and only one connection is required to program all of the devices. It could also be later possible to program the controller boards from the iPaQ.

7.2 Outcomes and Conclusions

The aim for this thesis was to outline the design and eventual creation of DC motor controllers for a humanoid robot. The hardware design has been shown in great detail, with some of the software also shown. The project has provided a platform for which full PID control can be implemented. The hardware was designed successfully and allows the control of 15 motors from five independent networked controller boards.

An extensive platform has been provided for all requirements of the humanoid. Temperature and foot sensors and also limit switches will provide useful feedback for closed loop control of the overall system from the iPaQ. All of this hardware is in place, only the software is required to utilise these systems.

It was hoped that a detailed description of the software would also be included, however this was not possible due to programming problems faced. A hardware basis has been provided and this will form an interesting control thesis for coming years.

Figure 7.1: Motor within Lower Leg Section

References

[1]
Bosch GmbH, R, *CAN Specification*, Ver 2.0, Stuttgart, 1991.

[2]
Bosch GmbH, R, *Bosch CAN Homepage*, http://www.can.bosch.com, (current October 18th, 2001)

[3]
Cartwright, T, *Design and Implementation of Small Scale Joint Controllers for a Humanoid Robot,* Undergraduate Thesis, Univ. of Queensland, Computer Science and Electrical Engineering, 2001.

[4]
Kennedy, J, *Design and Implementation of a Distributed Digital Control System in an Industrial Robot*, Undergraduate Thesis, Univ. of Queensland, Computer Science and Electrical Engineering, 1999.

[5]
Kim, M et al., "Development of a Humanoid Robot CENTAUR – Design, Human Interface, Planning and Control of its Upper-Body", *IEEE International Conference on Systems, Man and Cybernetics*, Vol 4, Tokyo, Japan, 1999, pp. 948-953.

[6]
Miller, G.H, *Microcomputer Engineering*, 2nd Ed, Prentice Hall, Upper Saddle River, New Jersey, 1999.

[7]
Nise, N.S, *Control Systems Engineering*, 3rd Ed, John Wiley & Sons Inc, New York, New York, 2000.

[8]
CAN in Automation (CiA), *Controller Area Network (CAN)*, http://www.can-cia.de, (current October 18th, 2001).

[9]
Agilent Technologies, *HCTL-2016 Datasheet*, http://www.agilent.com, (current October 18th, 2001) .

[10]
SGS Thompson Microelectronics, *L6203 Datasheet*, http://www.st.com, (current October 18th 2001).

[11]
Texas Instruments, *TMS320F243 Data Sheet,* Texas Instruments Literature SPRS06B, http://www.ti.com, 1999, (current October 18th 2001).

[12]
Philips, *PCA82C250 Data Sheet*, http://www.philips.com, (current October 18th 2001).

[13]
Protel International Ltd, http://www.protel.com, (current October 18th, 2001)

[14]
National Semiconductors, http://www.national.com (current October 18th, 2001).

[15]
On Semiconductor, http://www.onsemi.com (current October 18th, 2001).

[16]
Maxim Semiconductors, http://www.maxim-ic.com (current October 18th, 2001).

Appendix A – The Humanoid Team

Appendix B – Schematic Diagram

Appendix C – PCB Diagram

Appendix D – Software Listings

D.1 – Loop1.c

D.2 – Motor.c

D.3 – Interrupt Setup File – Vectors.asm

Appendix E – Programming the DC Motor Controllers

Appendix F – Timing Diagrams

F.1 – HCTL-2016 Timing Diagram and Timing Table

F.2 – TMS320F243 Timing Diagram

F.3 – TMS320F243 Timing Table

Appendix G – Integrated Semiconductor Datasheets

G.1 – TMS320F243 DSP Datasheet

G.2 – HCTL-2016 QDEC Datasheet

Appendix A – The Humanoid Team

Each of the following is an Undergraduate Thesis, Univ. of Queensland, Computer Science and Electrical Engineering, 2001.

Bebel, B, "*Design and Implementation of a USB-to-CAN Bridge for the GuRoo Project*".

Blower, A, "*Development of a Vision System for a Humanoid Robot*".

Brewer, N, "*Power System for a Humanoid*".

Cartwright, T, "*Design and Implementation of Joint Controllers for a Humanoid Robot*".

Hosking, S, "*High Speed Peripheral Interface*".

Hunter, A, "*Mechanical Design of a Humanoid"*.

Kee, D, "*Draft Systems for a Humanoid Robot*"

Prasser, D, "*Vision Software for a Humanoid Robot*"

Smith, A, "*Simulator Development and Gait Pattern Creation for a Humanoid Robot*".

Wagstaff, M, "*Mechanical Design and Internal Sensors for a Humanoid Robot*".

Zelniker, E, "*Joint Control for an Autonomous Humanoid Robot*".

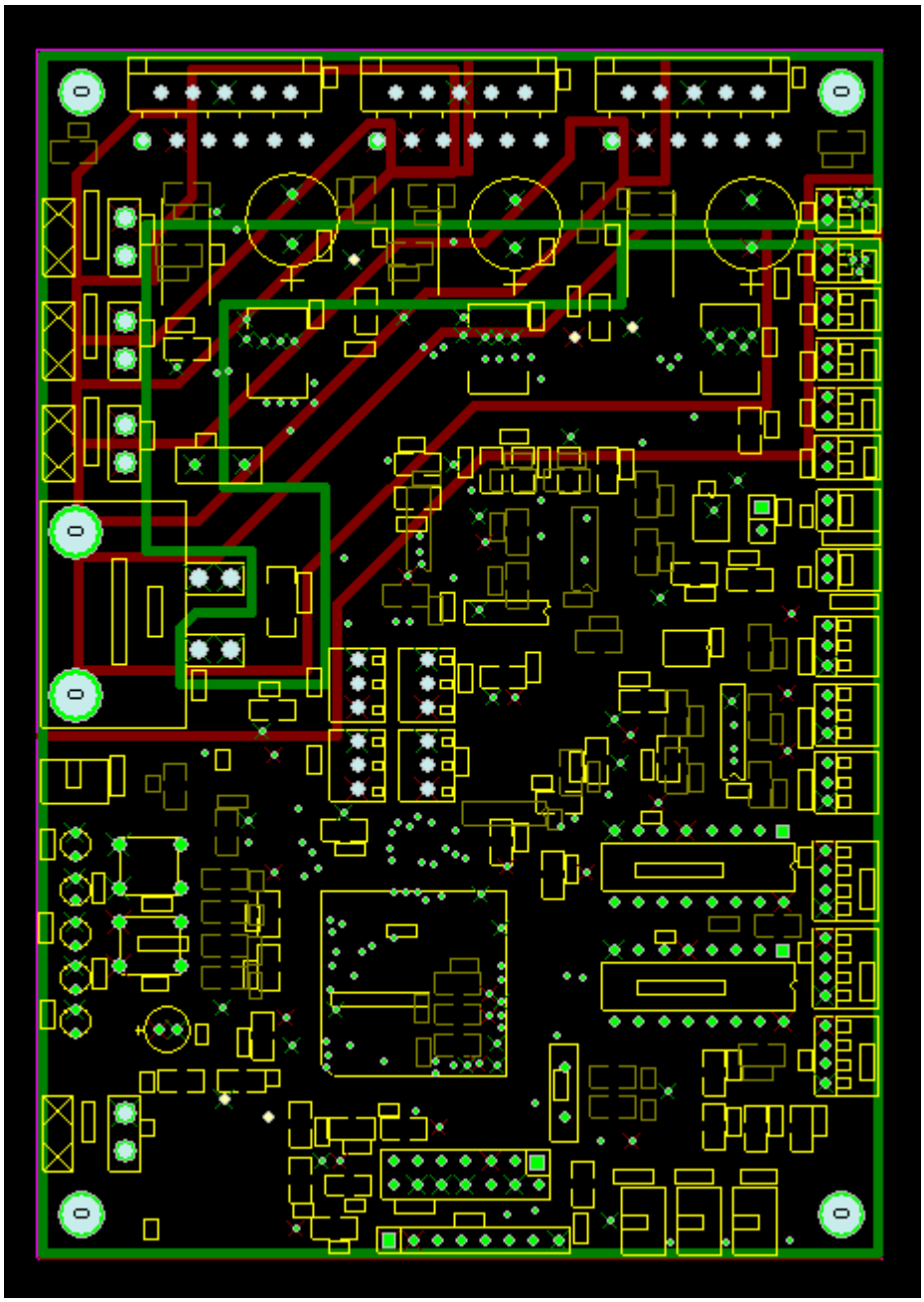
Appendix B – Schematic Diagram

Note: The 130R resistor is only for the terminating controller boards. These being boards 1 and 2 - the two boards controlling the ankles and knees.

Title	Joint Controller for 3 motors
Size	A2
Number	A1
Revision	E3
Date:	19-Oct-2001
Sheet	1 of 1
File:	H:\schematics\boards - jetad final.ddb
Drawn By:	

Appendix C – PCB Diagram

PCB showing internal planes



Appendix D – Software Listings

D.1 Loop1.c

```
/*
 * Jarad Stirzaker - 33696915
 * DC Motor Controllers for a Humanoid Robot
 * 26/09/2001
 *
 * This program uses some feedback from the QDEC to change the PWM duty cycle
 * of the motor. Since the QDEC is external to the motor, the program is used
 * to test the feedback abilities of the tms chip.
 */
#include "F24x.h"

#define PERIOD 200            /* for 100kHz PWM */
unsigned int cmpr1_val;
unsigned int cmpr2_val;
unsigned int cmpr3_val;

void pwm_setup(void);
void qdec_setup(void);

void main(void)
{
        long temp;
        int t2cnt_old;
        int t2cnt_sub;
        unsigned int compval;

        WDDISABLE;
        EVIMRA = 0x0000;
        INT_DISABLE;

        pwm_setup();
        qdec_setup();

        cmpr1_val = PERIOD / 2;              /* Set to initially 50% duty cycle */
        cmpr2_val = PERIOD / 2;
        cmpr3_val = PERIOD / 2;

        CMPR1 = cmpr1_val;
        CMPR2 = cmpr2_val;
        CMPR3 = cmpr3_val;

        /*
         * Data and Direction Control Registers 7-8
         * PxDATDIR   15-8 direction, 7-0 data, 0 - input, 1 - output
         */
        PDDATDIR = 0xFC00;            /* Clear Port D */
        PBDATDIR = 0xFFFF;            /* Enable L6203 chips */

        t2cnt_old = T2CNT;            /* setup previous value of T2CNT */

        while (1) {
                compval = T2CNT / 32;   /*  right shift MSB's to leds */
                PDDATDIR = (PDDATDIR & 0xFC1F) | compval;
                for (temp = 0; temp < 500000; temp++);

                /* Try polling first, using delay above for difference */
                t2cnt_sub = T2CNT - t2cnt_old;
                cmpr1_val = (-0.0488 * t2cnt_sub) + 100; /* scale between 0-100% duty */
                CMPR1 = cmpr1_val;

                t2cnt_old = T2CNT;
        }
}
```

```
void pwm_setup(void)
{
        /*
         * T1PR        Timer 1 Period Register - 20Mhz/T1PR        - freq
         * CMPRx       Compare Register x                          - duty
         * OCRA    7-4 I/O Mux Control Register A                  - set pwm pins
         * ACTR    8-37 Compare Action Control Register            - active high
         * DBTCON 8-41 Dead Band Timer Control Register            - dead band
         * COMCON 8-36 Compare Control Register                    - pwm
         * T1CON  8-28 GP Timer Control Register (individual)      - up, prescale
         * GPTCON 8-30 GP Timer Control Register                   - up, pol
         */

        /* Set period register for 100kHz = 20MHz/200 */
        T1PR = PERIOD;

        OCRA           = 0x0FD8;      /* 00001111 11011000 */

        ACTR           = 0x0999;      /* 00001001 10011001 */
        DBTCON  = 0x02E0;             /* 00000010 11100000 */

        CMPR1   = 0x0000;
        CMPR2   = 0x0000;
        CMPR3   = 0x0000;

        /*      +----------------- compare enabled
                |++--------------- reload compare register on underflow or period match
                |||+-------------- space vector not enabled
                ||||++------------ action control register reload on underflow
                ||||||+----------- compare output pins enabled
                |||||||+-++++++++- reserved
                |||||||| ||||||||
                10100010 00000000 */
        COMCON  = 0xA200;

        /*      ++---------------- emulation control = not affected
                || ++-------------- continuous up counting
                || ||+++---------- prescaler = 1
                || ||||| +-------- Use own TENABLE bit
                || ||||| |+------- enable T1
                || ||||| ||++----- internal clock as source
                || ||||| ||||++--- reload period on underflow or equal
                || ||||| ||||||+-- enable compare operation
                || ||||| |||||||+- not used in T1
                || ||||| ||||||||
                11010000 01000110 */
        T1CON   = 0xD046;

        /*       +---------------- read only counting status
                 |+--------------- read only counting status
                 ||  ++----------- no adc event with timer 2
                 ||  ||+-+-------- no adc event with timer 1
                 ||  ||| |+------- compare output enable - enable
                 ||  ||| ||  ++--- polarity timer 2 - active high
                 ||  ||| ||  ||++- polarity timer 1 - active high
                 ||  ||| ||  ||||
                00000000 01001010 */
        GPTCON   = 0x004A;
}
```

```
void qdec_setup(void)
{
        /* info   8-66
         * T2CNT  7-19
         * T2PR   7-15
         * T2CMPR 7-14
         * T2CON  8-28
         * CAPCON 8-59
         */

        T2CNT = 0x7FFF;
        T2PR = 0xFFFF;
        T2CMPR = 0xFFFF;

        /* ++---------------- not affected by emulation suspend
           || ++-------------- Directional Up/Down Count mode
           || ||+++---------- Clock Prescalar = /1
           || ||||| +-------- Use own TENABLE bit
           || ||||| |+------- TENABLE - Enable Timer
           || ||||| ||++----- Clock Source - QEP Circuit
           || ||||| ||||++--- Register reload when counter is 0
           || ||||| ||||||+-- TECMPR - enable timer compare
           || ||||| |||||||+- SELTPR1 - use own period register
           || ||||| ||||||||
           11011000 01110010 */
        T2CON = 0xD872;

        /* +---------------- Clear all capture registers
           |++-------------- Enable QEP Circuit (bits 9, 7-4 ignored - *)
           |||+------------- Disbale CAP3
           |||| +----------- GP Timer selection for cap3   = GP Timer 2
           |||| |+---------- GP Timer selection for cap1/2 = GP Timer 2 - *
           |||| ||+--------- CAP3TOADC = no event
           |||| ||| ++------ CAP1 Edge Detection = Detect both edges - *
           |||| ||| ||++---- CAP2 Edge Detection = Detect both edges - *
           |||| ||| ||||++-- CAP3 Edge Detection = none
           |||| ||| ||||||
           01100000 11110000 */
        CAPCON = 0x60F0;
}
```

D.2 Motor.c

```
/*
 * Jarad Stirzaker - 33696915
 * Controllers for a humanoid robot
 * 26/09/2001
 *
 * This program simply uses the external interupt XINT1 to stop and reverse the
 * the motion of the motor, by slowing change the PWM duty cycle.
 */
#include "F24x.h"

unsigned long compval;

void pwm_setup(void);
void int_setup(void);
void c_int6(void);

void main(void)
{
        long temp;

        WDDISABLE;
        EVIMRA = 0x0000;

        INT_DISABLE;

        pwm_setup();
        int_setup();

        compval = 25;

        CMPR1 = compval;
        CMPR2 = 0x0040;
        CMPR3 = compval;

        /*
         * Data and Direction Control Registers 7-8
         * PxDATDIR   15-8 direction, 7-0 data
         * 0 - input, 1 - output
         */

        PBDATDIR = 0xFFFF;
        PDDATDIR = 0xFC00;

        while (1) {
                PDDATDIR = (PDDATDIR & 0xFF5F) | ~(PDDATDIR | 0xFF5F);
                for (temp = 0; temp < 100000; temp++);
        }
}


void pwm_setup(void)
{
        /*
         * T1PR        Timer 1 Period Register - 20Mhz/T1PR        - freq
         * CMPRx       Compare Register x                          - duty
         * OCRA    7-4 I/O Mux Control Register A                  - set pwm pins
         * ACTR    8-37 Compare Action Control Register            - active high
         * DBTCON 8-41 Dead Band Timer Control Register            - dead band
         * COMCON 8-36 Compare Control Register                    - pwm
         * T1CON  8-28 GP Timer Control Register (individual)      - up, prescale
         * GPTCON 8-30 GP Timer Control Register                   - up, pol
         */


        /* Set period register for 100kHz = 20MHz/200 */
        T1PR = 200;

        OCRA           = 0x0FC4;      /* 00001111 11000100 */
        ACTR           = 0x0999;      /* 00001001 10011001 */
```

```
        DBTCON  = 0x02E0;             /* 00000010 11100000 */

        CMPR1   = 0x0000;
        CMPR2   = 0x0000;
        CMPR3   = 0x0000;

        /*      +----------------- compare enabled
                |++--------------- reload compare register on underflow or period match
                |||+-------------- space vector not enabled
                ||||++------------ action control register reload on underflow
                ||||||+----------- compare output pins enabled
                |||||||+ ++++++++- reserved
                ||||||| ||||||||
                10100010 00000000 */
        COMCON  = 0xA200;

        /*      ++----------------- emulation control = not affected
                || ++------------- continuous up counting
                || ||+++---------- prescaler = 1
                || ||||| +-------- do not enable T2 as well
                || ||||| |+------- enable T1
                || ||||| ||++----- internal clock as source
                || ||||| ||||++--- reload period on underflow or equal
                || ||||| ||||||+-- enable compare operation
                || ||||| |||||||+- not used in T1
                || ||||| ||||||||
                11010000 01000110 */
        T1CON   = 0xD046;

        /*       +---------------- counting up
                 |+--------------- counting up
                 ||  ++----------- no adc event with timer 2
                 ||  ||+-+-------- no adc event with timer 1
                 ||  ||| |+------- compare output enable - enable
                 ||  ||| ||  ++--- polarity timer 2 - active high
                 ||  ||| ||  ||++- polarity timer 1 - active high
                 ||  ||| ||  ||||
                01100000 01001010 */
        GPTCON   = 0x604A;

}


void int_setup(void)
{
        /*
         * IMR     3-15       Interrupt Mask Register
         * IFR     3-13       Interrupt Flag Register
         * XINT1CR 6-5        XINT1 Control Register
         */

        IMR = 0x0020;         /* INT6 unmasked */
        XINT1CR = 0x0007;     /* XINT1 rising edge, low priority, enabled */
        INT_ENABLE;           /* Enable interupts */

}
```

```
void c_int6(void)
{
        /* initially cmpr1 is set to 25, want to change through 100 and up to 175
         * and vice versa if set at already 25 */
        long counter;

        if (compval == 25) {
                while (compval < 175) {
                        for (counter = 0; counter < 20000; counter++);
                        compval++;
                        CMPR1 = compval;
                }
        } else if (compval == 175) {
                while (compval > 25) {
                        for (counter = 0; counter < 20000; counter++);
                        compval--;
                        CMPR1 = compval;
                }
        }

        PDDATDIR = (PDDATDIR & 0xFFBF) | ~(PDDATDIR | 0xFFBF);

        XINT1CR = 0x8007;
        /* clear interrupt level 6 flag */
        IFR = 0x0020;
}
```

D.3 Interrupt Setup File – Vectors.asm

```
;-------------------------------------------------------------------------
; Vector address declarations
; for motor.c
;-------------------------------------------------------------------------
         .sect  ".redir"
         .ref   _c_int0
               B       _c_int0 ; redirects the reset vector to boot.obj

         .sect  ".vectors"
         .ref  _c_int6

RSVECT         B    1F00H     ; PM 0         Reset Vector          1
INT1           B    PHANTOM   ; PM 2         Int level 1           4
INT2           B    PHANTOM   ; PM 4         Int level 2           5
INT3           B    PHANTOM   ; PM 6         Int level 3           6
INT4           B    PHANTOM   ; PM 8         Int level 4           7
INT5           B    PHANTOM   ; PM A         Int level 5           8
INT6           B    _c_int6   ; PM C         Int level 6           9
RESERVED       B    PHANTOM   ; PM E         (Analysis Int)        10
SW_INT8        B    PHANTOM   ; PM 10        User S/W int          -
SW_INT9        B    PHANTOM   ; PM 12        User S/W int          -
SW_INT10       B    PHANTOM   ; PM 14        User S/W int          -
SW_INT11       B    PHANTOM   ; PM 16        User S/W int          -
SW_INT12       B    PHANTOM   ; PM 18        User S/W int          -
SW_INT13       B    PHANTOM   ; PM 1A        User S/W int          -
SW_INT14       B    PHANTOM   ; PM 1C        User S/W int          -
SW_INT15       B    PHANTOM   ; PM 1E        User S/W int          -
SW_INT16       B    PHANTOM   ; PM 20        User S/W int          -
TRAP           B    PHANTOM   ; PM 22        Trap vector           -
NMI            B    PHANTOM   ; PM 24        Non maskable Int      3
EMU_TRAP       B    PHANTOM   ; PM 26        Emulator Trap         2
SW_INT20       B    PHANTOM   ; PM 28        User S/W int          -
SW_INT21       B    PHANTOM   ; PM 2A        User S/W int          -
SW_INT22       B    PHANTOM   ; PM 2C        User S/W int          -
SW_INT23       B    PHANTOM   ; PM 2E        User S/W int          -

*-------------------------------------------------------------------------
* Phantom ISR - Just changes the led config and then holts until the
* processor is reset.
*-------------------------------------------------------------------------
PHANTOM:        B    PHANTOM

*-------------------------------------------------------------------------
* Program end
*-------------------------------------------------------------------------
               .end
```

Appendix E – Programming the DC Motor Controllers

See Kennedy, 1999 [4] for information not shown here.

The TI software tools should be installed on the computer with all files found in the root directory. From here a directory should be formed with the .C file for the code and also the *vectors.asm* file required for interrupts for this .C file.

e.g. *motor.c*

TI Tools	C:\DSPTOOLS
Code	C:\DSPTOOLS\MOTOR

Use the batch file *cl.bat* to run the C compiler and keep all created code within the directory above. This is compiled by running *cl motor*. This batch file is listed here.

```
dspcl -q -v2xx -k -s %1\*.c %1\*.asm -fr %1 -fs %1 -z F243.cmd -o %1\prog.out
               -m %1\prog.map
pause
F240_hex %1\prog.out
copy %1\prog.hex %1\%1.hex
```

Also, the file *F243.cmd* on the following page is required for the *cl.bat* to execute completely.

```
/*************************************************************************/
/* File Name:     f243.cmd                                               */
/* Target System: C24x Evaluation Board                                  */
/*                                                                       */
/* Description:   A basic linker command file for the 'F240 device.      */
/*                This file is used by the linker to determine where     */
/*                certain sections of code should reside in memory.      */
/*                                                                       */
/* Revision:      1.00                                                   */
/*************************************************************************/

/*-----------------------------------------------------------------------*/
/*  LINKER COMMAND FILE - MEMORY SPECIFICATION for the F240              */
/*-----------------------------------------------------------------------*/

-stack 100
-l rts2xx.lib

MEMORY
{
    PAGE 0 :   VECS   : origin =    0h , length =   040h   /* VECTORS */
               JUMP   : origin =   40h , length =    02h   /* REDIRECT */
               PROG   : origin =   42h , length = 01EC0h   /* PROGRAM */

    PAGE 1 :   MMRS   : origin =    0h , length =   060h   /* MMRS    */
               B2     : origin = 0060h , length =   020h   /* DARAM   */
               DARAM  : origin = 0200h , length =  0200h   /* DARAM   */
}

/*-----------------------------------------------------------------------*/
/* SECTIONS ALLOCATION                                                   */
/*-----------------------------------------------------------------------*/
SECTIONS
{
/* Vectors.asm, Interrupt vector table */
    .vectors > VECS      PAGE 0

/* Jump vector to boot.obj */
    .redir > JUMP        PAGE 0

/* C, Executable code and floating point constants */
    .text    > PROG      PAGE 0

/* C, Tables for explicity initialized global and static variables */
    .cinit   > PROG      PAGE 0

/* C, Jump tables for large switch statements */
    .switch  > PROG      PAGE 0

/* C, String literals, and global and static const variables that are
   explicitly initialized */
    .const   > PROG     PAGE 0

/* C, Global and static variables */
    .bss     > DARAM     PAGE 1

/* C, Software Stack */
    .stack   > DARAM     PAGE 1

/* C, Dynamic memory area for malloc functions */
    .sysmem  > DARAM     PAGE 1

/* Memory mapped registers */
    .mmrs    > MMRS      PAGE 1

/* Initialization data tables */
    .data    > DARAM     PAGE 1
}
```

If the boot loader code within memory is corrupted then this needs to be reloaded onto the device. To do this the following steps need to be followed as per the TI documentation.

- Switch off power
- Connect JTAG connector
- Connect BIO and VCCP high
- Turn on power
- Run *EMURST.exe* to reset the JTAG pod
- Run *BTEST.bat* to test communications with the JTAG pod

Don't go on until *BTEST.bat* completes successfully

- Run *BC0.bat* to clear the flash memory
- Run *BE0.bat* to erase the flash memory
- Run *PROG.bat* to reload the boot loader
- Turn off power

PROG.bat is listed here.

```
prg2xx -p 240 -m 0x0006 -w 6 src\c2xx_bpX.out sf_pe.out
```

Now the device should be serially programmable as per usual.

Appendix F – Timing Diagrams

F.1 – HCTL-2016 Timing Diagram and Timing Table

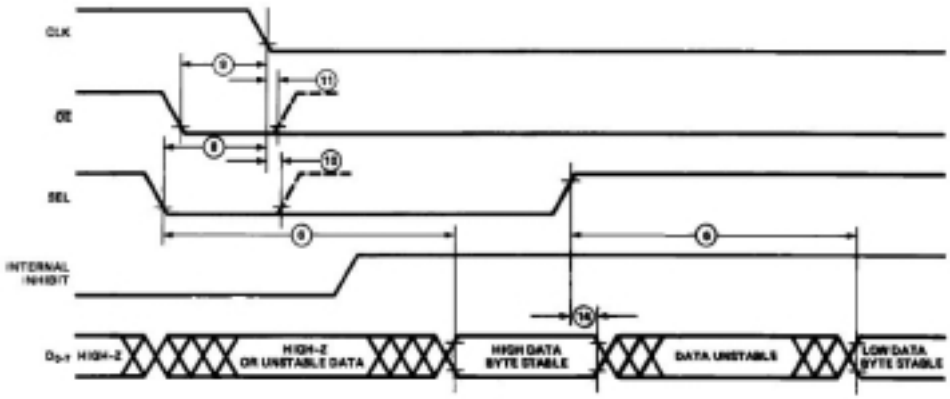


Figure 4. Bus Control Timing.

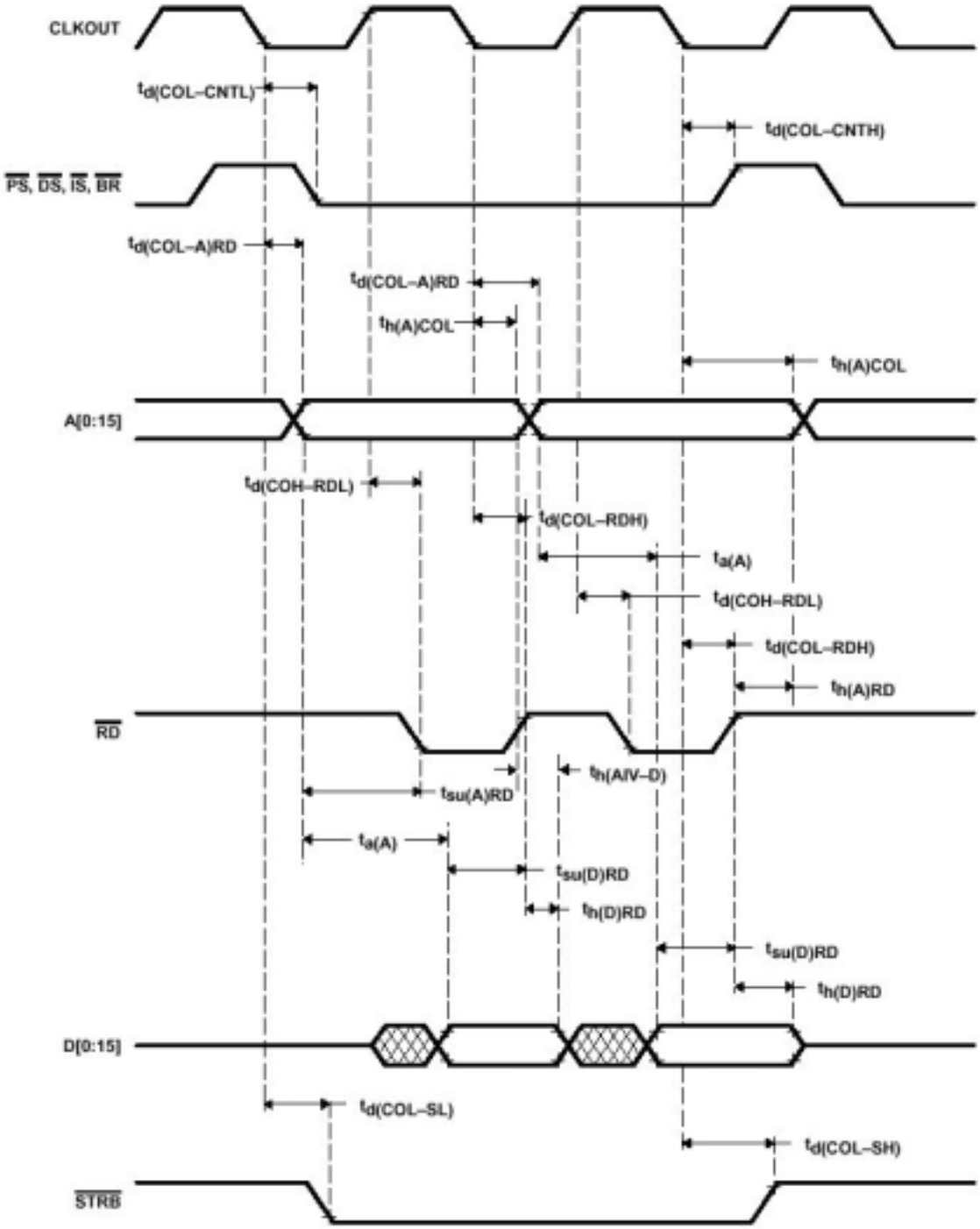
Switching Characteristics

Table 5. Switching Characteristics Min/Max specifications at V_{DD} = 5.0 ± 5%, T_A = -40 to + 85 °C.

	Symbol	Description	Min.	Max.	Units
1	t_{CLK}	Clock period	70		ns
2	t_{CHH}	Pulse width, clock high	28		ns
3	t_{CD}[1]	Delay time, rising edge of clock to valid, updated count information on D0-7		65	ns
4	t_{ODE}	Delay time, $\overline{OE}$ fall to valid data		65	ns
5	t_{OHZ}	Delay time, $\overline{OE}$ rise to Hi-Z state on D0-7		40	ns
6	t_{SDV}	Delay time, SEL valid to stable, selected data byte (delay to High Byte = delay to Low Byte)		65	ns
7	t_{CLH}	Pulse width, clock low	28		ns
8	t_{SS}[2]	Setup time, SEL before clock fall	20		ns
9	t_{OS}[2]	Setup time, $\overline{OE}$ before clock fall	20		ns
10	t_{SH}[2]	Hold time, SEL after clock fall	0		ns
11	t_{OH}[2]	Hold time, $\overline{OE}$ after clock fall	0		ns
12	t_{RST}	Pulse width, $\overline{RST}$ low	28		ns
13	t_{DCD}	Hold time, last position count stable on D0-7 after clock rise	10		ns
14	t_{DSD}	Hold time, last data byte stable after next SEL state change	5		ns
15	t_{DOD}	Hold time, data byte stable after $\overline{OE}$ rise	5		ns
16	t_{UDD}	Delay time, U/$\overline{D}$ valid after clock rise		45	ns
17	t_{CHD}	Delay time, CNT_{DCDR} or CNT_{CAS} high after clock rise		45	ns
18	t_{CLD}	Delay time, CNT_{DCDR} or CNT_{CAS} low after clock fall		45	ns
19	t_{UDH}	Hold time, U/$\overline{D}$ stable after clock rise	10		ns
20	t_{UDCS}	Setup time, U/$\overline{D}$ valid before CNT_{DCDR} or CNT_{CAS} rise	t_{CLK}-45		ns
21	t_{UDCH}	Hold time, U/$\overline{D}$ stable after CNT_{DCDR} or CNT_{CAS} rise	t_{CLK}-45		ns

Notes:
1. t_{CD} specification and waveform assume latch not inhibited.
2. t_{SS}, t_{OS}, t_{SH}, t_{OH} only pertain to proper operation of the inhibit logic. In other cases, such as 8 bit read operations, these setup and hold times do not need to be observed.

F.2 – TMS320F243 Timing Diagram



F.3 – TMS320F243 Timing Table

external memory interface *read* timings

switching characteristics over recommended operating conditions for an external memory interface read (see Figure 37)

PARAMETER		MIN	MAX	UNIT
$t_{d(COL-CNTL)}$	Delay time, CLKOUT low to control valid		3	ns
$t_{d(COL-CNTH)}$	Delay time, CLKOUT low to control inactive		3	ns
$t_{d(COL-A)RD}$	Delay time, CLKOUT low to address valid		5	ns
$t_{d(COH-RDL)}$	Delay time, CLKOUT high to $\overline{RD}$ strobe active		4	ns
$t_{d(COL-RDH)}$	Delay time, CLKOUT low to $\overline{RD}$ strobe inactive high	–4	0	ns
$t_{d(COL-SL)}$	Delay time, CLKOUT low to $\overline{STRB}$ strobe active low		3	ns
$t_{d(COL-SH)}$	Delay time, CLKOUT low to $\overline{STRB}$ strobe inactive high		3	ns
$t_{h(A)COL}$	Hold time, address valid after CLKOUT low	–4		ns
$t_{su(A)RD}$	Setup time, address valid before $\overline{RD}$ strobe active low	22		ns
$t_{h(A)RD}$	Hold time, address valid after $\overline{RD}$ strobe inactive high	–1		ns

timing requirements [H = $0.5t_{c(CO)}$] (see Figure 37)

		MIN	MAX	UNIT
$t_{a(A)}$	Access time, read data from address valid		2H–20	ns
$t_{su(D)RD}$	Setup time, read data before $\overline{RD}$ strobe inactive high	12		ns
$t_{h(D)RD}$	Hold time, read data after $\overline{RD}$ strobe inactive high	0		ns
$t_{h(A)V-D}$	Hold time, read data after address invalid	–3		ns

Appendix G – Integrated Semiconductor Datasheets

G.1 – TMS320F243 DSP Datasheet

TMS320F243, TMS320F241
DSP CONTROLLERS

SPRS064C – DECEMBER 1997 – REVISED SEPTEMBER 2000

- High-Performance Static CMOS Technology
- Includes the TMS320C2xx Core CPU
  - Object-Compatible With the TMS320C2xx
  - Source-Code-Compatible With TMS320C25
  - Upwardly Compatible With TMS320C5x™
  - 50-ns Instruction Cycle Time
- Commercial and Industrial Temperature Available
- Memory
  - 544 Words x 16 Bits of On-Chip Data/Program Dual-Access RAM (DARAM)
  - 8K Words x 16 Bits of Flash EEPROM
  - 224K Words x 16 Bits of Total Memory Address Reach (F243 only)
- External Memory Interface (F243 only)
- Event-Manager Module
  - Eight Compare/Pulse-Width Modulation (PWM) Channels
  - Two 16-Bit General-Purpose Timers With Four Modes, Including Continuous Up and Up/Down Counting
  - Three 16-Bit Full Compare Units With Deadband
  - Three Capture Units (Two With Quadrature Encoder-Pulse Interface Capability)
- Single 10-Bit Analog-to-Digital Converter (ADC) Module With 8 Multiplexed Input Channels
- Controller Area Network (CAN) Module
- 26 Individually Programmable, Multiplexed General-Purpose I/O (GPIO) Pins
- Six Dedicated GPIO Pins (F243 only)
- Phase-Locked-Loop (PLL)-Based Clock Module
- Watchdog (WD) Timer Module
- Serial Communications Interface (SCI) Module
- 16-Bit Serial Peripheral Interface (SPI) Module
- Five External Interrupts (Power Drive Protection, Reset, NMI, and Two Maskable Interrupts)
- Three Power-Down Modes for Low-Power Operation
- Scan-Based Emulation
- Development Tools Available:
  - Texas Instruments (TI) ANSI C Compiler, Assembler/Linker, and C-Source Debugger
  - Full Range of Emulation Products
    - Self-Emulation (XDS510™)
  - Third-Party Digital Motor Control and Fuzzy-Logic Development Support
- 144-Pin LQFP PGE Package (F243)
- 68-Pin PLCC FN Package (F241)
- 64-Pin QFP PG Package (F241)

description

The TMS320F243 and TMS320F241 devices are members of the 24x generation of digital signal processor (DSP) controllers based on the TMS320C2000™ platform of 16-bit fixed-point DSPs. The F243 is a superset of the F241. These two devices share similar core and peripherals with some exceptions. For example, the F241 does not have an external memory interface. This new family is optimized for digital motor/motion control applications. The DSP controllers combine the enhanced TMS320™ DSP family architectural design of the C2xx core CPU for low-cost, high-performance processing capabilities and several advanced peripherals optimized for motor/motion control applications. These peripherals include the event manager module, which provides general-purpose timers and PWM registers to generate PWM outputs, and a single,10-bit analog-to-digital converter (ADC), which can perform conversion within 1 µs.

TEXAS INSTRUMENTS
POST OFFICE BOX 1443 • HOUSTON, TEXAS 77251-1443

G.2 – HCTL-2016 QDEC Datasheet

Agilent Technologies
Innovating the HP Way

Quadrature Decoder/Counter Interface ICs

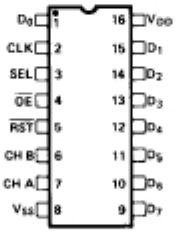
Technical Data

HCTL-2000
HCTL-2016
HCTL-2020

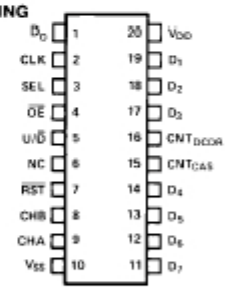
Features

- **Interfaces Encoder to Microprocessor**
- **14 MHz Clock Operation**
- **Full 4X Decode**
- **High Noise Immunity:** Schmitt Trigger Inputs Digital Noise Filter
- **12 or 16-Bit Binary Up/ Down Counter**
- **Latched Outputs**
- **8-Bit Tristate Interface**
- **8, 12, or 16-Bit Operating Modes**
- **Quadrature Decoder Output Signals, Up/Down and Count**
- **Cascade Output Signals, Up/ Down and Count**
- **Substantially Reduced System Software**

DIGITAL MOTION ENCODING



PINOUT A



PINOUT B

Applications

- **Interface Quadrature Incremental Encoders to Microprocessors**
- **Interface Digital Potentiometers to Digital Data Input Buses**

Description

The HCTL-2000, 2016, 2020 are CMOS ICs that perform the quadrature decoder, counter, and bus interface function. The HCTL-20XX family is designed to improve system performance

Devices

Part Number	Description	Package Drawing
HCTL-2000	12-bit counter. 14 MHz clock operation.	A
HCTL-2016	All features of the HCTL-2000. 16-bit counter.	A
HCTL-2020	All features of the HCTL-2016. Quadrature decoder output signals. Cascade output signals.	B