ROBOCUP SYSTEMS ENGINEERING PROJECT 2002

A Design Project Report

Presented to the Engineering Division of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering (Electrical)

by

Liang-Yu (Tom) Chi

Wajih Effendi

Michael Jordan

Sin-Man Ko

Wei-Feng Li

Evan Malone

Shahab Najmi

Michael Schwaller

Project Advisor: Prof. Raffaello D'Andrea

Degree Date: May 2002

Abstract

Master of Electrical Engineering Program
Cornell University
Design Project Report

Project Title: **ROBOCUP SYSTEMS ENGINEERING PROJECT 2002**

Author: Liang-Yu (Tom) Chi , Michael Jordan, Sin-Man Ko,
Wei-Feng Li, Evan Malone, Shahab Najmi,
Michael Schwaller

Abstract:

The international RoboCup competition entails the construction of fully autonomous, fast-moving robots which work together as a team to compete against similar teams in a robotic soccer match. These robots function within the larger context of a system combining artificial intelligence, computer vision, and numerous electrical and mechanical subsystems.

The electrical engineering team designed and built the individual electrical subsystems (wireless, microcontroller design, motion control, dribbling, and kicking) as well as smoothly integrated them into the larger system. Our system is a significant improvement over that of 2001 as we have revamped the firmware architecture and made our dribbling, kicking, IR sensing, and wireless systems more robust. We have also improved debugging capabilities, added in-circuit programmability, improved configurability, and modularized the board design.

The result is a well-integrated electrical system which allows the realization of the mechanical design goals and enhances the functionality of the artificial intelligence and control. Our robot is faster, more robust, more flexible and easier to service than in previous years.

Report Approved by

Project Advisor: ____Date: ______________

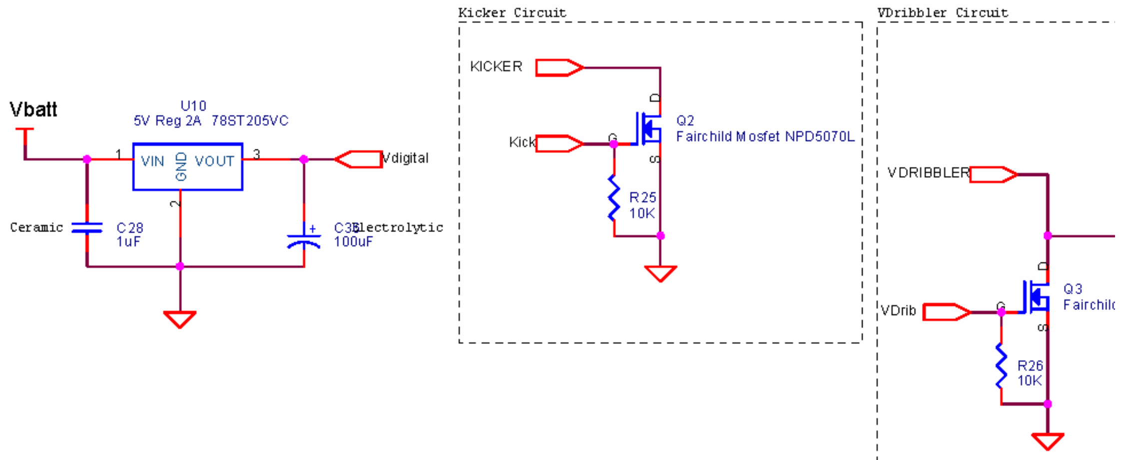
Table of Contents

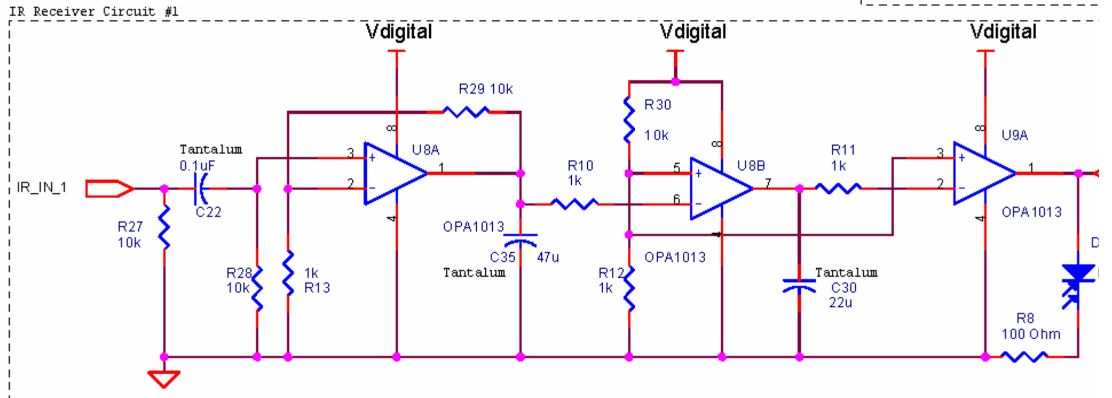
Table of Contents 3
1. Individual Contributions 14
2. Executive Summary 16
3. System Overview 17
4. Systems Engineering Design Process 19
4.1. Process Overview 19
4.1.1. Overview 19
4.1.2. Group Meetings and Minutes 20
4.1.3. Individual Laboratory Notebooks 20
4.1.4. Intranet/Electronic Mail 21
4.1.5. Group Electronic Documentation 21
4.1.6. Datasheets 21
4.1.7. Code Versioning System / CVS 21
4.1.8. Parts List 22
4.1.9. Board Issues List 22
4.1.10. Battery Logging 23
4.1.11. Conclusion / Process Benefits 23
4.2. System Engineering Product Cycle 23
4.3. Customer Needs 24
4.4. Requirements Gathering 24
4.5. Special Task Forces 24
4.5.1. Design for Testability 24
4.5.2. Design for Usability 26
4.5.3. Design for Maintainability 26
4.5.3.1. Batteries 27
4.5.3.2. Location of connectors 27
4.5.3.3. Types of connectors 27
4.5.3.4. Wireless module selection 28
4.5.3.5. Design for Robustness 28
4.5.3.5.1. Enlarging PCB trace width 28
4.5.3.5.2. Observing current ratings 29
4.5.3.5.3. Inter-board connection 29
4.5.3.5.4. System reset problems 29
4.5.3.5.5. Wireless communications flooded 30
4.6. Preliminary Design 30
4.7. Preliminary Testing 30
4.8. Manufacturing 30
5. Digital System 33
5.1. Main Microcontroller 33
5.1.1. Overview 33
5.1.1.1. Microcontroller comparison 34
5.1.1.2. Selection criteria: 35
5.1.1.3. Microcontroller features: 36
5.1.2. System Features 37

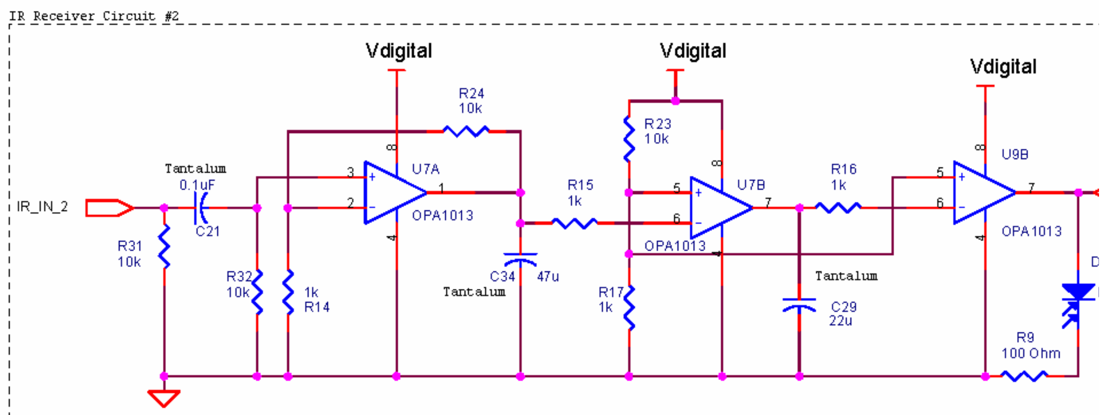
5.1.2.1. Interrupt based implementation 37
5.1.2.2. Easy to Debug 38
5.1.2.3. Fast / More reliable system....38
5.1.3. Main Microcontroller pin allocation 38
5.1.4. Function description: 39
5.1.4.1. Initialization 40
5.1.4.2. Receiving wireless packet 42
5.1.4.3. Parsing/processing packet 43
5.1.4.4. Kick....45
5.1.4.5. Horizontal dribbler 47
5.1.4.6. Check battery voltage 49
5.1.4.7. Buzzer function 49
5.1.4.8. Robot mode switch 49
5.1.4.9. LCD Display 49
5.1.4.10. Wheel Velocities 50
5.2. Wireless Microcontroller 51
5.3. Motion Microcontroller 52
5.3.1. Overview 52
5.3.2. Wheel Velocity Transformations 52
5.3.3. Wheel Velocity Calculations 54
5.3.4. Motion Control Variables: Tradeoffs....55
5.3.5. Architecture....55
5.3.6. Motion Microcontroller pin allocation....56
5.3.7. Rise Time and Steady State Error 57
5.3.8. Slow Speed Add-ons 57
5.4. FPGA 58
5.4.1. Overview 58
5.4.2. Selection of the FPGA 58
5.4.3. Features of EPM7128S 58
5.4.4. Detailed Implementation....59
5.4.5. Special considerations:....60
5.4.5.1. HDL (Hardware Description Language)....60
5.4.5.2. Pin Assignments....60
5.4.5.3. Dedicated Pin Connections 61
5.4.5.4. Clock Frequency Consideration and others 61
5.5. Ultrasonic 61
5.5.1. Transmitter....62
5.5.2. Receiver 62
6. Analog System....63
6.1. Batteries 63
6.1.1. Overview 63
6.1.2. Battery Selection....64
6.1.2.1. Current Estimate 64
6.1.2.2. Power Consumption....64
6.1.2.3. Final Battery Selection....65
6.1.3. Battery Specifications and characteristics: 67

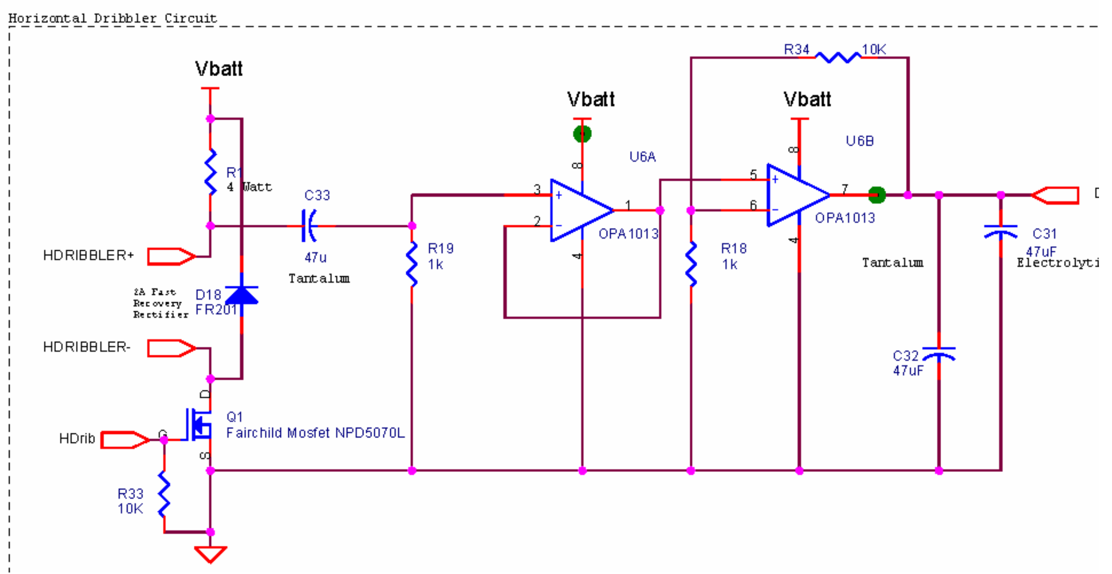
6.1.4. Fuses and Voltage Indicator: 69
6.1.5. Kicker Battery....70
6.2. 5-Volt Integrated Switching Voltage Regulator 70
6.3. Kicker....73
6.3.1. Variable Speed Kicking 73
6.3.2. Testing of Kicker 73
6.4. IR Circuit....74
6.4.1. Overview 74
6.4.2. IR Transmitter....74
6.4.3. IR Receiver 74
6.4.4. Possible Improvements 75
6.5. Dribbling System....76
6.6. H-Bridges....77
7. Wireless System....79
7.1. Overview 79
7.2. Requirements....80
7.3. Design to Meet Requirements....81
7.4. Preliminary Research....83
7.4.1. Technologies Considered....83
7.4.2. Wireless Modules Considered 83
7.4.2.1. Radio Packet Controllers....84
7.4.2.2. TX2/RX2....85
Notes 86
7.4.2.3. TX3/RX3....86
7.4.2.4. SE200 87
7.4.2.5. Aerocomm....88
7.4.2.6. Wireless Mountain - Unilink....90
7.5. Developing Functional Prototypes....91
7.6. Functional Design 92
7.6.1. Overview 92
7.6.2. Packet Structure....93
7.6.3. Hardware Design....94
7.6.4. Software Implementation 96
7.7. Testing....97
7.7.1. Overview 97
7.7.2. Procedure 98
7.7.3. Latency Test 99
7.7.4. Maximum Bit Rate Test 100
7.7.5. Bit Error Test....101
7.8. On-Robot Hardware Design....101
7.8.1. Hardware Layout for Wireless RX Board....101
7.8.2. Antennas....104
7.8.2.1. Antennas Types....104
7.8.2.2. Connector Types 105

7.8.2.3. Mounting Considerations to the Wireless Board 106
7.8.2.4. Optimization 106
7.8.3. Interference 106
7.8.4. Mounting Considerations 108
7.9. Off-Robot Hardware Design 108
7.9.1. Hardware Layout for TX Wireless Board 108
7.10. Conclusion 110
8. Goalie Systems 111
8.1. Overview 111
8.2. Goalie Systems 111
8.2.1. Bi-directional Solenoid 111
8.2.2. Horizontal Dribbler Motor 112
8.2.3. Chip-Kick 112
8.2.4. Motion Control 112
8.3. Customized Packet Structure 113
9. System Use 114
9.1. Microcontroller 114
9.1.1. Setting Robot ID 114
9.1.2. Test Modes 114
9.2. Programming 114
9.2.1. Overview 114
9.2.2. Watchdog timers 114
9.2.3. Programming the Bootloader onto the PIC16F877 115
9.2.4. Programming the PIC16F877 116
9.2.5. Programming the FPGA 117
9.2.6. Setting Wireless Module ID 118
9.3. System Debugging 119
9.3.1. Inter-microcontroller Communication 119
9.3.1.1. Parallel vs. Serial Communication 119
9.3.1.2. Serial Communication via PIC16F877 119
9.3.1.3. Overview of inter-microcontroller Communication 120
9.3.2. Debug Ports 122
10. Appendix 123
A. NETWORKING 123
B. SCHEMATICS 131

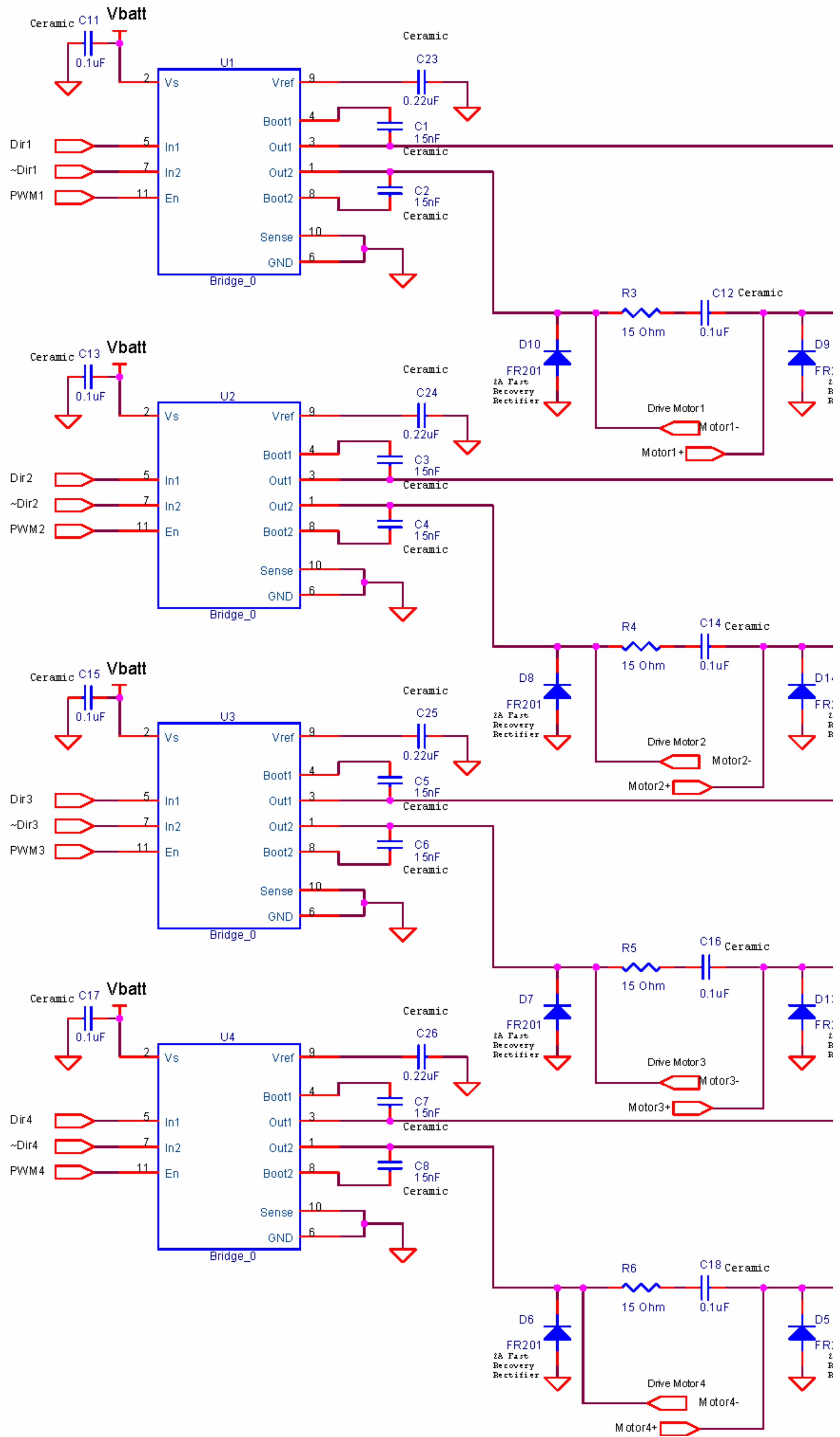

Vbatt
U10
5V Reg 2A 78ST205VC
VIN
GND
VOUT
Vdigital
Ceramic
C28
1uF
Electrolytic
100uF
Kicker Circuit
KICKER
Kick
Q2
Fairchild Mosfet NPD5070L
R25
10K
VDribbler Circuit
VDRIBBLER
VDrib
Q3
R26
10K


IR Receiver Circuit #1
Vdigital
Vdigital
Vdigital
R29 10k
R30
10k
IR_IN_1
Tantalum
0.1uF
C22
U8A
U8B
U9A
R10
1k
R11
1k
R27
10k
R28
10k
1k
R13
OPA1013
C35
47u
Tantalum
R12
1k
OPA1013
Tantalum
C30
22u
OPA1013
R8
100 Ohm


IR Receiver Circuit #2
Vdigital
Vdigital
Vdigital
R24
10k
R23
10k
IR_IN_2
Tantalum
0.1uF
C21
U7A
U7B
U9B
R15
1k
R16
1k
R31
10k
R32
10k
1k
R14
OPA1013
C34
47u
Tantalum
OPA1013
R17
1k
Tantalum
C29
22u
OPA1013
R9
100 Ohm


Horizontal Dribbler Circuit
Vbatt
Vbatt
Vbatt
R34
10K
U6A
U6B
4 Watt
C33
47u
Tantalum
OPA1013
OPA1013
HDRIBBLER+
R19
1k
R18
1k
Tantalum
C31
47uF
2A Fast
Recovery
Rectifier
D18
FR201
HDRIBBLER-
C32
47uF
Q1
Fairchild Mosfet NPD5070L
HDrib
R33
10K

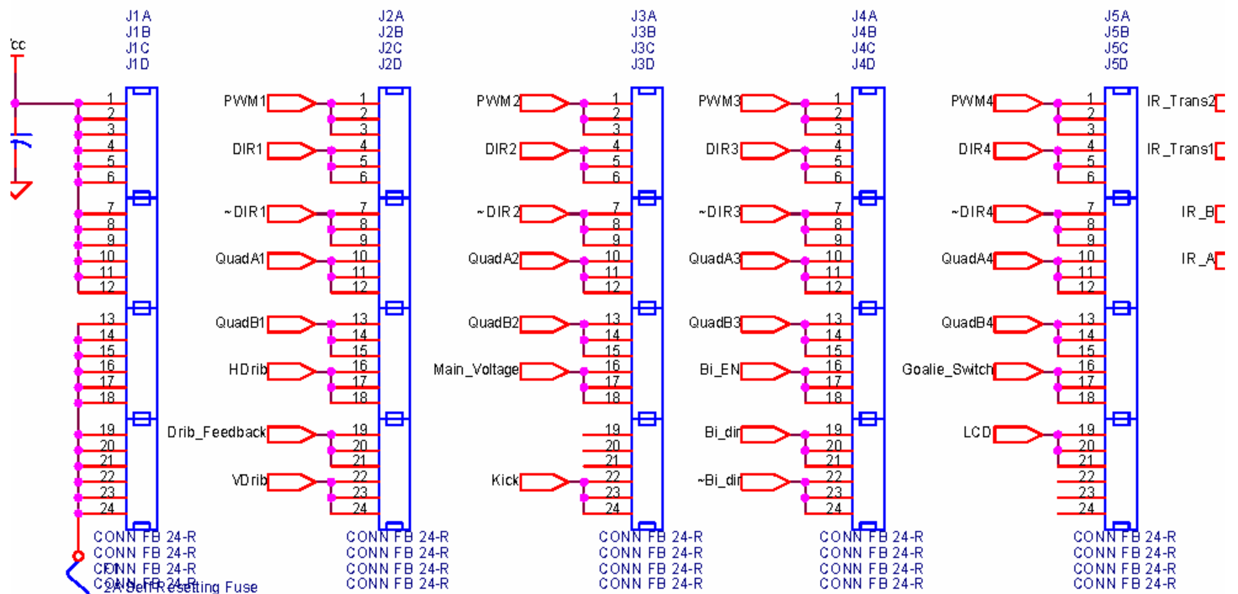
....131



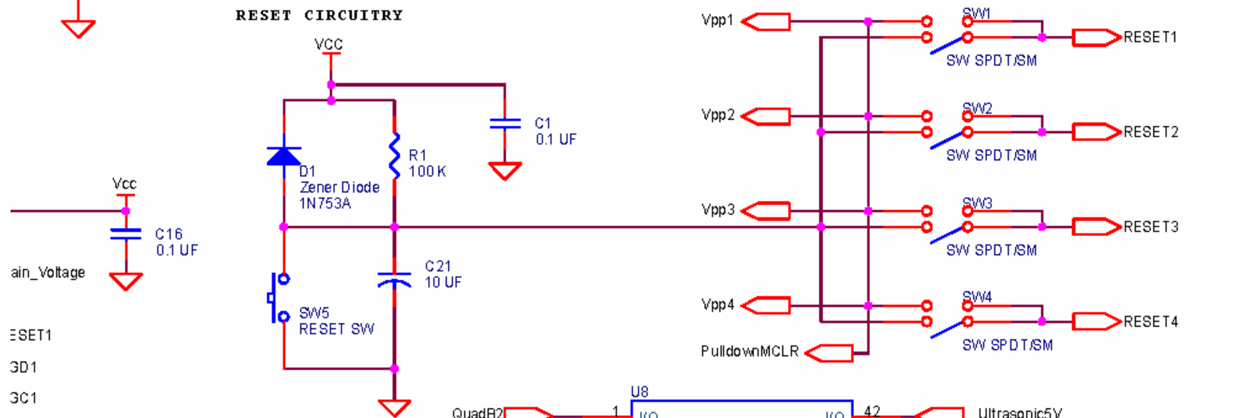
2
Vbatt
Ceramic
C11
0.1uF
U1
Vs
Vref
Boot1
In1
Out1
In2
Out2
En
Boot2
Sense
GND
Bridge_0
Dir1
~Dir1
PWM1
C23
0.22uF
C1
15nF
C2
R3
15 Ohm
C12
0.1uF
D10
FR201
2A Fast Recovery Rectifier
D9
Drive Motor1
Motor1-
Motor1+
C13
U2
Dir2
~Dir2
PWM2
C24
C3
C4
R4
C14
D8
Drive Motor2
Motor2-
Motor2+
C15
U3
Dir3
~Dir3
PWM3
C25
C5
C6
R5
C16
D7
Drive Motor3
Motor3-
Motor3+
C17
U4
Dir4
~Dir4
PWM4
C26
C7
C8
R6
C18
D6
D5
Drive Motor4
Motor4-
Motor4+

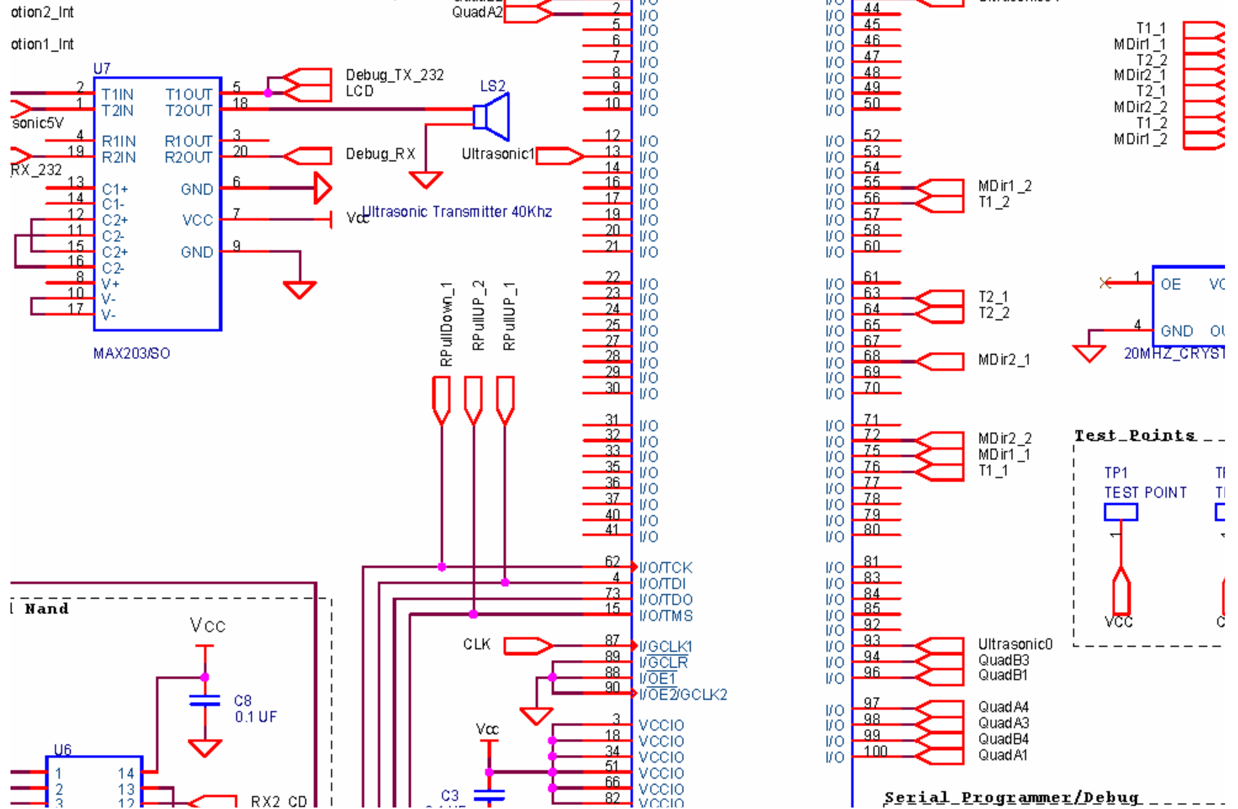
....132

onnectors from/to Digital



RESET CIRCUITRY





....134
C. BOARD LAYOUT 136
D. MICROCONTROLLER: REASONS FOR CHOSING PIC16F877..............146
E. MEETING MINUTES TEMPLATE 147
F. FPGA CIRCUIT BLOCK DIAGRAM / PINOUT DIAGRAM......................148
G. PHOTO GALLERY..151
H. MAIN MICROCONTROLLER FIRMWARE...156
I. MOTION CONTROL FIRMWARE..171
J. SYSTEM REQUIREMENTS 175

Acknowledgements

The 2002 RoboCup EE team would like to acknowledge all those who made this project possible. We would like to thank Prof. Raffaello D'Andrea for his precise input and constant guidance throughout the course of the project. We would also like to thank Jin-Woo Lee for constructive and instructive advice in all technical matters. We are grateful to Evan Malone for his numerous insights and tireless devotion to the team. To Tama's, for giving our robots direction (literally), and finally, to Wajih Effendi who lead the team with endless enthusiasm and energy.

1. Individual Contributions

Liang-Yu (Tom) Chi

Tom Chi designed and implemented the new IR, ultrasonic, variable dribbling, and variable kicking circuits. Specifically he improved the IR robustness, took the ultrasonic system and variable dribbling from concept to completion, and discovered a simple solution to variable kicking. He also wrote the new proportional-control motion code as well as designing the hardware necessary to drive the motors. During first semester, he worked with Katherine Ko to develop the Robocup networking API and perform network timing tests.

Michael Jordan

Michael Jordan's main responsibility was the wireless communications system for the RoboCup system. He was the head of wireless sub-group, which worked on increasing the reliability of the wireless communications system. To this end, Michael Jordan worked on researching alternative wireless modules as well as designing, implementing, and laying out the circuitry to support these modules. He also developed, implemented, and applied a system of tests and test metrics to evaluate the performance of the wireless modules and their associated design parameters. He is now working on incorporating forward error control and error detection onto the wireless system. In addition to his work on the wireless communication, he provides supporting roles to the other team members in areas such as system design, debugging, and maintenance.

Sinman Katherine Ko

Ko, Sinman Katherine is in the electrical engineering team of Robocup. She was responsible for developing a robust wireless system and efficient networking code for the autonomous robots. For networking, she developed server/multi-clients application for effective communication between the vision with other systems, for instance the compiler and simulator. Another task was to design a robust wireless system that would reduce frequency interference, bit error rate and send packet within the 16.6ms time constraint, performed trade-off analysis and testing of different wireless module. She also was participating in overall analog and digital circuit layout testing.

Wei Feng Li

Wei Feng Li was responsible for integrating the Altera FPGA into the system. He designed, debugged and verified the circuits on the FPGA, which are parts of both the motion control and ultrasonic systems. To achieve this he analyzed the characteristics of the motor encoder. He also helped to verify the circuit layout for the final design

circuits, and to investigate potential applications of the Atmel microcontroller in future design.

Evan Malone

Evan Malone is the project systems engineer. He participated in the identification of requirements for, and the conception, design, and manufacture of the 2002 electrical systems. His primary focus is the packaging of the systems to maximize modularity, robustness, and maintainability, without sacrificing performance. Through the dedicated cooperative effort of the electrical and mechanical engineering groups, and the advice of the 2001, these goals have been greatly exceeded.

Shahab Najmi

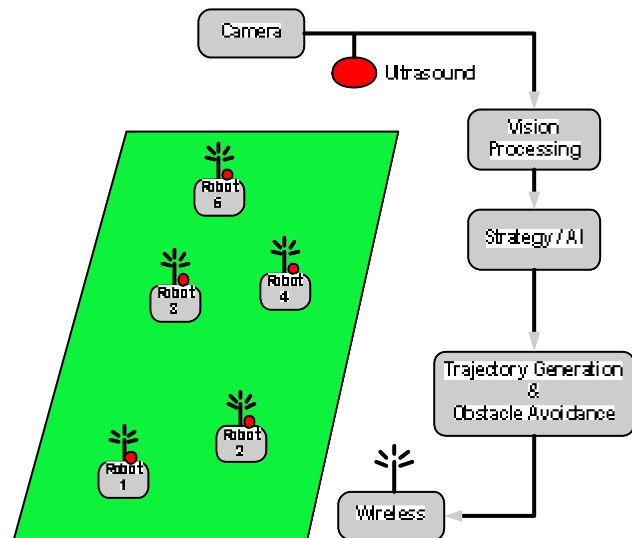
Shahab Ahmed Najmi took the lead on microcontroller selection, and was responsible for developing the main microcontroller code. The main microcontroller is the processing and control hub of the robot. It takes input from the wireless microcontroller, collecting, parsing and converting packets into wheel velocities and subsystem control instructions. Once parsed, the ma in microcontroller forwards these commands and performs necessary control for the variable speed kicking, dribbling, and motion microcontrollers. In addition to these duties, it checks battery level and writes key information to the debug LCD. Shahab also worked on the design of robot support systems and actively participated in the system debugging at all stages.

Michael Schwaller

Michael Schwaller contributed to four main areas on the 2002 RoboCup system: the systems engineering and integration effort, the wireless communication subsystem, process and documentation, and overall system design and construction. The systems engineering process included special focus on designing for testability, usability, maintainability, and robustness. Michael contributed to the wireless module evaluation, selection, and implementation to produce a low-latency, high quality broadcast link to the team of robots. Continued process and documentation improvement improved the quality of the design team's efforts.

2. Executive Summary

The international RoboCup Competition pits teams from around the world against each other in fast-paced robotic soccer matches. Despite the competition format, there is a larger research motivation behind RoboCup: to develop a team of fully-autonomous robots who can defeat the human World Cup champions by 2050.



Cornell has historically been an important contributor to RoboCup, introducing a number of key innovations over the years which have significantly raised the level of play. Among these are the introduction of dribblers, omni-directional drive and controlled passing and motion. These innovations have been rewarded by 1st place finishes in 1999 and 2000, and a strong 3rd place finish in 2001.

The 2001 system offered improvements, and introduced CNC machining methods, but it also had significant drawbacks: overall the robots were slower than much of the competition and several subsystems including wireless, IR, and motion were less robust than desired. Consequently our electrical engineering team has directed energies toward improving the robustness of these systems: supporting multiple carrier frequencies on the wireless system, making IR resistant to ambient light, and replacing unreliable PID control with microcontroller-based proportional control. Beyond improving robustness, we have also supported a number of improvements and innovations. We now help realize a faster drive system, have added tightly controlled kicking and dribbling, and provided ultrasonic talkback to the intelligence and control system.

Throughout the project, our team has maintained a strong systems focus. This focus has greatly aided the integration of complex electrical subsystems into the larger context of mechanical design and artificial intelligence. This year's boards are more cleanly packaged and have been more thoroughly tested and reviewed. Systems concepts have manifested themselves in our work via adding in-circuit programmability, improving configurability, extending debugging capabilities, and modularizing board design. It has also simplified our trade-off analysis and allowed us to accomplish our goals within a compressed timeframe.

The result is a well integrated electrical system which allows the realization of the mechanical design goals and enhances the functionality of the artificial intelligence and control. Our robot is faster, more robust, more flexible and easier to service than in previous years.

3. System Overview

This is the fourth year the Cornell RoboCup team has participated in the RoboCup Small-Size League competition. The responsibilities of the electrical engineering group within the RoboCup project are to design and improve the electrical systems for the new team of robots, maintain the electrical system on the old robots, and to assist the RoboFlag team in their electrical requirements.

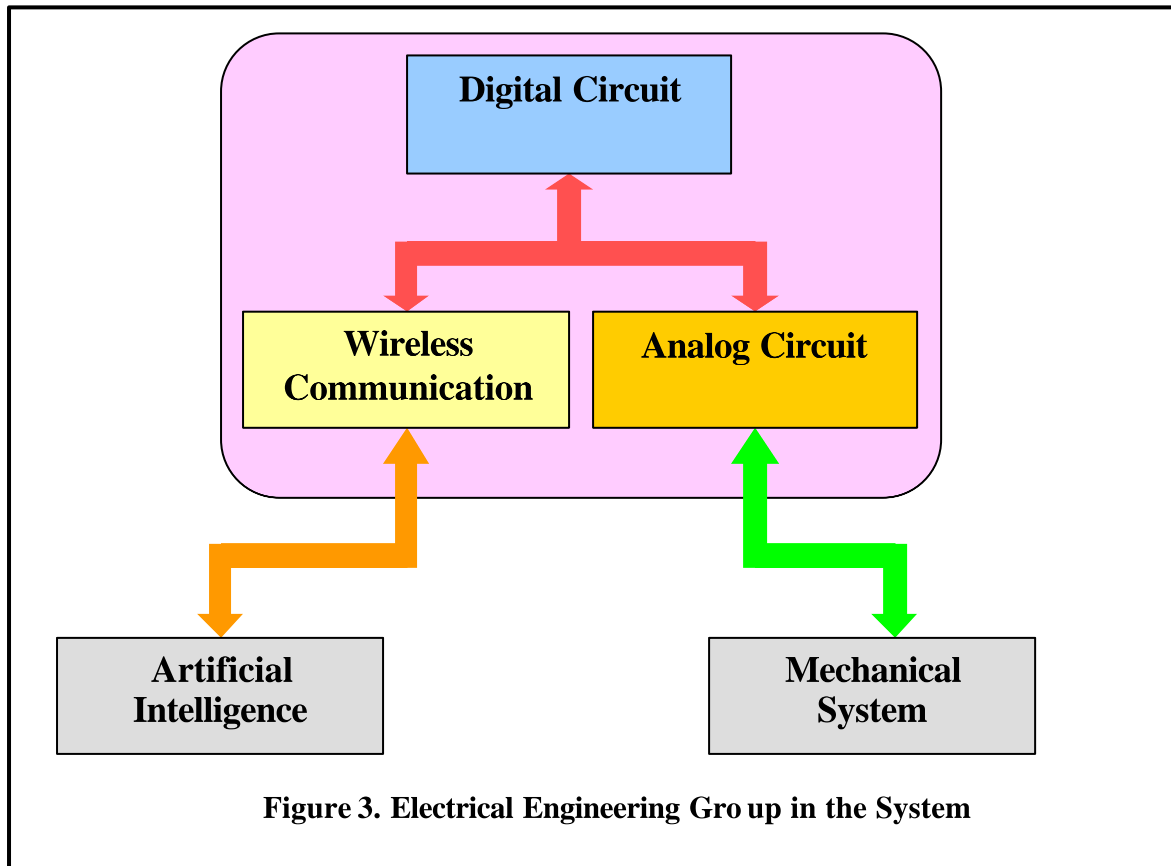


Figure 3. Electrical Engineering Gro up in the System

The goal of the electrical engineering team is to build an efficient, robust and reliable electrical system that will help the Cornell RoboCup Team to reclaim the championship this year. As Figure 1 suggests, we divide the project into three main platforms: digital, analog and wireless systems. The wireless component receives the data transmitted from the AI computer. The digital component processes the data from the wireless system and controls the different functions of the robot. The analog component distributes the digital supply voltage and provides power to the kicking, dribbling and motion control actuators.

Last year, the electrical engineering team focused on improving robot performance of with most of the hardware unchanged. This year, we take a different approach. We decided to take a leading role by innovating rather than merely supporting the efforts of

the Computer Science (CS) and Mechanical Engineering (ME) groups. We researched and worked on several areas like new microcontrollers, more sophisticated and flexible control of actuators, more robust sensing, modularity, testability, maintainability, robustness, and usability. We also took the lead in promoting designs and processes aimed at satisfying project, rather than individual group goals.

This year's electrical engineering team has begun to apply the systems engineering approach to the design process. Focusing on well-documented requirements up-front compiled from prior years' experience and from the insight of current project stakeholders ensures a more methodical, goal-oriented design approach. Through the use of special task forces we are also able to focus on robust designs that place a special emphasis on maintainability, usability, and testability. By explicitly considering systems integration issues early in the design phase and seeking inter-disciplinary team input leads to a tighter integration of the electrical subsystem into the final product. In the presence of this well-documented process we reduce uncertainty and promote success of the electrical system while enhancing knowledge retention for future years.

Here are the highlights of the major improvements we made from robots from previous generations. A new wireless communications system between the control system and the robots is implemented. The new system supports three different types of modules operating on different carrier frequencies for broadcasting data to the robots. In addition, an optional, ultrasonic return path has been implemented to add flexibility. The main control system on the robot has been upgraded from a polling-based to a completely interrupt-driven system, thereby increasing robustness and reducing latency. A new high-performance four-wheel omni-locomotion system conceived by the mechanical group has been made feasible by the introduction of flexible, software configurable feedback control circuitry, and more efficient power electronics. Other performance improvements include variable-energy kicking, torque-feedback controlled dribbling for better ball control. A great deal of effort was expended on making the electrical systems easily used, maintained, debugged, and more robust through the addition of a debug port, error display, intra-board communication eavesdropping, modular motherboard/daughterboard configuration, and by designing to counteract anticipated failures.

4. Systems Engineering Design Process

4.1. Process Overview

4.1.1. Overview

The discipline of systems engineering operates to integrate all of the functional subgroups involved in a project over the whole product cycle. System engineers make sure the product matches the marketplace, define the components to enable the technical engineers to design and build them, determine most of the design choices affecting system cost and performance, ensure that the components will integrate successfully and perform together as required, provided that their specifications are free of error. At the project planning phase, a project manager creates an initial systems engineering management plan for the project, defining tasks, resources, milestones, costs and schedule at each milestone. At the same time a core technical process is applied to create a behavioral and structural model of the system. Trade-off and risk analysis are performed to search first for feasible, then for optimal solutions, given the available resources and constraints. During the project implementation phase, an iterative of sequential build-and-test process is carried out, and development is monitored with the aim of identifying any major errors that arise during the development process. The product is finally validated by testing compliance with the required performance measures. The whole systems engineering process proceeds in an iterative, controlled manner to ensure that the system meets the design criteria within the available time and budget. Written documentation is kept throughout the cycle to maintain corporate memory and permit learning from prior cycles.

RoboCup is a complex, systems engineering project, with many components being redesigned and constructed from scratch by an inexperienced team of students subject to performance, schedule, resource and budgetary constraints. Simultaneously satisfying these constraints and while proceeding through the technological development of the system is a very challenging information management problem. It is essential that the various technical subgroups in the project share a common vision of the system on which they are working, and that the specific technical design requirements that they are attempting to satisfy result from a rational decomposition of the requirements for the fully integrated system. Furthermore, everyone participating in the project has technical development responsibilities, and little time to spend on inessential overhead tasks. For these reasons, a pragmatic and methodical approach to project information management is necessary. Several of the M.Eng. degree candidates participating in RoboCup are involved in the Cornell University Systems Engineering option, and have attempted to experiment with and apply a variety of systems engineering techniques and tools to RoboCup. The result is a significant deviation from the idealized presentation of systems engineering provided above. Over the course of the project, useful methods have evolved and expanded, while others have proven too burdensome or ineffective, and have died away. A discussion of the various methods evaluated follows.

4.1.2. Group Meetings and Minutes

The EE team holds a functional group meeting every week. Group members would update and report their progress on the project over the previous week. The meetings also allows group members to voice their problems and concerns for the whole team for open discussion. This interactive communication process allows group members to exchange information, expectations and feedback as well as discuss about planning, upcoming schedules and deadlines and priorities.

Minutes were initially instituted as a means for progress and task responsibility tracking, as well as a decision process history for ourselves or future teams to use to identify the origin of team decision process and system design failures. The example form in the appendix is typical of the style used for recording meeting minutes. Minutes were kept fairly reliably for the first ¾ of the project duration, but few people found them useful, and most found them burdensome to record and post. A more streamlined minutes design was suggested and employed by the EE group (See appendix), but even with the reduced overhead of this design, few people found them worth keeping. In the very intense final quarter of the project duration, minutes have been *de facto* eliminated.

4.1.3. Individual Laboratory Notebooks

Each EE team member has his/her own individual laboratory notebook. The function of the notebook is for idea retention. It is advisable to transform the little ideas into the notebook. .

Nearly the entire first ½ of the project duration was spent in construction of 10 units of the 2001 design robots for use in an exhibition match of 11 vs. 11 robot soccer at the upcoming competition. While manufacturing these robots, it was discovered that last year's documentation was erroneous and incomplete in many details, and in ways which led to significant wasted time for this year's team in reconstructing lost knowledge and correcting errors. It was gradually realized that because the RoboCup project extends into the summer, beyond submission of grades and graduation and the associated motivation for diligence, most of the work performed during the summer did not make its way into any form of document, and was lost. It is also all too common that from time to time, members may come up with some unique and interesting ideas, but then lose them for want of a very simple means of storing them. Another advantage is that they are very handy for recording test results. From these considerations emerged the concept of providing individual laboratory notebooks to each team member. It is hoped that by placing few restrictions on their use, other than that content be legible, dated, and given a topic heading, members would find these notebooks to be a low impedance repository for ideas, observations, and test results that might prove useful to us or to future teams.

4.1.4. Intranet/Electronic Mail

This year's team was fortunate (for the initiative of an EE team member) to receive sponsorship from Intranets.com to have a team intranet for posting information, storing documents, posting calendars, and numerous other functions. The intranet was an extremely useful tool for some and not that useful for others. One major use of the intranet was the storage and distribution of documents for the team to access. It possesses a number of tools for event calendars with email reminders, polls, discussions, etc. Unfortunately, though, the calendar tool is not very sophisticated, and could not handle multiple schedules. An expanded bulletin board area on the site would have been valuable, as well. More could have been made of the tool, but members and leaders both were somewhat reluctant to spend the time required to keep the information on the site up to date. In addition to the intranet site, email list-serves were created for each of the subgroups to enable easy communication among team members and among functional subgroups. These tools enabled the entire team to keep each other abreast of major accomplishments by delivering information in a timely manner to the entire team.

4.1.5. Group Electronic Documentation

The team felt that electronic documentation was also very important as a way to keep track of important data that was easy to lose in paper format. Electronic documentation included:

- Datasheets
- Code Versioning System / CVS
- Parts List
- Board Issues List
- Battery logging

4.1.6. Datasheets

Electronic copies of most of the data sheets available online were stored in a directory for easy access. The ease of accessibility allowed quick information retrieval during both design and debugging phases. A copy of all of the datasheets used by the 2002 electrical engineering team can be found in the Appendix

4.1.7. Code Versioning System / CVS

A late addition to the suite of electronic tools used was a code versioning system. The original method of backup entailed copies of versions of the code accompanied by a

running history of modifications in a README file. The system was replaced by the use of the free CVS package also used by the artificial intelligence and control group. This software package allowed for much more comprehensive version control and comparisons of different states of the software development. CVS will prove to be an invaluable tool as we approach the summer months of development.

4.1.8. Parts List

A parts list spreadsheet was developed to satisfy multiple needs during the project. Major categories in the spreadsheet included the following:

- Part Description
- Component Designator (name on schematic)
- Footprint (Layout)
- Description
- Value
- Manufacturer
- Manufacturer Part Number
- Digikey (Distributor) Part Number
- Use (specific circuit in which part is used)
- Quantity per Board
- Comments
- Bin Number
- Total in Stock

The spreadsheet allowed us to keep track of the inventory of parts still in stock, allowing us to order enough components, and Bin Number corresponded to the exact location of the parts. Digikey part numbers along with quantities needed per board made it simple to order more parts as necessary. Component designators provided the link between part numbers and the actual location of them on the physical board. Lastly, the footprint was invaluable while laying out the board. All of this information was in one convenient location and found multiple uses throughout our development process. One minor setback was getting everyone to continue to keep the document updated. To improve this process we recommend using change tracking, using a database system (MS Access, for instance), or keeping the document in CVS to keep mistaken changes from entering into the system.

4.1.9. Board Issues List

A board issues list was started for each board after our first design was sent out for fabrication. As errors were discovered, they were posted to a document with a status and comments. This document served invaluable as a checklist when performing revisions on the board to ensure that the team did not miss errors.

4.1.10. Battery Logging

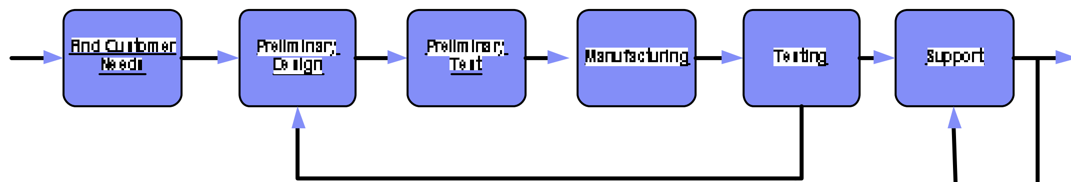
Batteries were labeled with a designator and then matched to a battery log to record the voltage measured after charging. This record of charging history allows us to keep track of the battery performance over its lifetime. Battery charging instructions were written and training for other team members was initiated to ensure proper maintenance habits.

4.1.11. Conclusion / Process Benefits

Time and again, we discovered that the sense of urgency of an impending deadline caused us to rush through preparations in the naïve hope of being lucky – getting a successful result – this despite our stated goal of pursuing a methodical systems engineering approach. This never works. We are building a complex system, and there is no substitute for careful forethought, constant scrutiny of each others work, constant vigilance against the typical and inexplicable human failures. Despite even the most sophisticated and patient methods, errors will still occur, but careful design, and a habit of diligence make the scale of the errors, and finding them far less difficult.

4.2. System Engineering Product Cycle

The systems engineering product cycle that the electrical engineering team adopted can be seen below. Initially, previous years' documentation was read to define paths already taken and areas that could use improvement. Project stakeholders or customers are then interviewed to determine new requirements. Customers included the project advisor and other team members. The requirements were then used to drive the design. Preliminary design consisted mainly of breadboarding circuits followed by preliminary testing. After having confidence in the design, the designs were sent for manufacturing and then more testing. After iterations of this process, our final product emerged. This end result then needed support through competition. The need for support is then passed on to future teams.



4.3. Customer Needs

Customer needs were gathered from previous years' documentation, key veteran experts, current project stakeholders, and video recordings of prior matches. The information gathered was then synthesized into requirements that can be found in the Appendix.

4.4. Requirements Gathering

An essential component of systems engineering (arguably of any task) is the clear statement of goals. To this end, in the first ½ of the project, an effort was made to capture the global requirements for the RoboCup system, and to decompose them into technical requirements that could be used as design specifications by each of the technical subgroups working on the system. Interviews were conducted with all technical groups, with project leadership, with all veteran members, and a close inspection of prior year's documentation, prior year's systems designs, and the work of other teams (on video) was made. A set of documents were generated which stated the primary requirements that this year's system was/is believed to need to meet to attain success. These were decomposed through additional interviews with technical subgroups into more field-specific requirements.

4.5. Special Task Forces

As the design process proceeded, a conscious attempt was made to ensure that all technical subgroups were sharing the same vision of the fully integrated system, and that they were making design decisions first and foremost on the basis of their implications for the system. As systems become more complex, it becomes more essential to give careful consideration to matters beyond performance, most notably to the fact that everything must work before performance can matter. This required some special EE and interdisciplinary task forces to consider special systems design issues. The issues can be broken down into several categories

- **Design for Testability**
- **Design for Usability**
- **Design for Maintainability**
- **Design for Robustness**
- **Design for Manufacture**

4.5.1. Design for Testability

A large effort was put into making this year's robots extremely easy to debug and test, as this was one of the main complaints of last year's system after having built and

debugged eleven of the 2001 robots. A team was formed with the sole intent of making the subsystems and other portions of the hardware and software systems easy to debug. A large list of numerous possible enhancements were considered including the following:

- Adding jumpers to sections of circuits in order to physically remove parts of the system.
- Aggregating wire harnesses so that there could be one point of connection from the robot to the electronics, allowing for a test harness to be built with monitoring points available on the harness.
- Test points in the circuit allowing convenient probing of important signals such as power, ground, 5V regulation, and other key signals.
- On-board display to show battery health, robot ID, wireless signal strength, and robot operational mode.
- Register values that show the "health" of certain part of the system. This "health" could be as simple as a "heartbeat" signal to ensure that the circuit we are viewing is still active and operating as expected.
- A serial port for communications for debugging the micro-controller.
- A graphical user interface (GUI) for the wireless transmitter in order to:
    - Test wireless transmissions
    - Gather data on other parts of the system
    - Re-use the application that can be used as a diagnostic tool plugged into the serial port.
- Power on tests or self-tests to test each subsystem and other features.
- Selectable test modes to isolate and test:
    - Dribbling
    - Kicking
    - Motion Control
    - Wireless System including Latency, Bandwidth, and Bit Error Rate

After further consideration, analysis, or testing, some of the above were accepted, and some rejected. A summary of the outcomes and the reasons for them follow:

- Jumpers were not implemented, however, as a fortuitous side effect of our only feasible concept for multi-processor in-circuit programming (using SPDT switches to hold all but the micro to be programmed in reset), we are able to shut down individual microcontrollers.
- An aggregating wire harness has become a motherboard with socketed daughterboards.
- Testpoints were designed in, included in a first revision of the boards, then later removed to speed the routing process on a later revision under time pressure. In the end, the hindrance to the routing process turned out to be an incorrectly sized "DRC routing box" - the inexperience of the OrCad operator. A deadline precluded replacing them in the design.

- A backlit LCD is supported in the current design, and correct operation has been verified, but the device has yet to find its place in the very tightly packaged robot. A few possible locations still exist.
- The register/heartbeat code may still be included if code memory permits.
- The debugging serial port is implemented in hardware, but is not yet supported in software. It remains to be seen whether sufficient code memory space remains to include this functionality.
- At present, it appears that we lack sufficient time prior to competition to develop a more sophisticated interface to the wireless transmitter.
- Power-on self-tests will also depend on the availability of code memory.
- Several selectable test modes have been implemented, but their sophistication and extent are limited.

4.5.2. Design for Usability

Another focus of the 2002 EE team was the usability of the robot. In previous years, the human-robot interface was limited to a bank of dip switches to set robot modes, a small LED to display robot number, and a reset switch. Connectors for hardware such as motors and sensors found themselves on all sides of the board and occasionally running from one side of the robot to the other. Not many of the connections were labeled causing even more confusion to the user. With respect to the board, the system was made up of a digital and analog board, where the analog board found itself underneath the digital board. This placement made it difficult to gain access to the analog board. The boards were individually fastened to each other and to the chassis via four screws making board replacement an involved task. Furthermore, a mechanical engineer attempting to access the kicking solenoid would find removal of the boards a difficult and annoying task.

In 2002, the main design changes that made large differences in system usability were in the area of the human-robot interface and the physical layout and connection of the electrical boards to the robot chassis. The LED has been replaced with a small back-lit liquid crystal display (LCD) that enables us to convey multiple pieces of information to the user. Extra LEDs on both the analog and digital board convey information on different subsystem status.

4.5.3. Design for Maintainability

Maintainability may seem as an unimportant area to focus design energy on, but you will quickly learn the importance after using any system for a while. The main focus of the maintainability team was on following items:

- Batteries
- Location of connectors
- Types of connectors
- Wireless module selection

4.5.3.1. Batteries

Battery removal on the 2001 robots was very difficult and required access to the bottom of the robot and removal of two small screws holding battery plates against the chassis. Batteries were in two packs that were connected by a wire harness that needed to be disconnected to remove the batteries. This system worked well except that it was difficult to quickly remove the batteries from the robot and was extremely annoying with the frequency of battery replacement. Secondly, the robot was not protected by its carbon fiber hat when the user would turn the robot upside down, placing a lot of pressure on the circuit boards and allowing for the possibility of harming circuit components by either force or static discharge. In conjunction with the mechanical engineers, the batteries were designed in three always connected packs that could be inserted from the sides of the robot through hinged doors. This solution made battery replacement much easier and decreased the size because of removal of connector complexity. Screws were replaced by pins that could quickly be pulled in and out of the battery door hinges.

4.5.3.2. Location of connectors

Connectors are a point of constant abuse, especially in the 2001 robot system where device connected from all sides of the robot to the bottom board with very short wires. The removal of the electrical boards required constant disconnection of the devices, which in turn suffered a lot of stress since the connectors were small and difficult to disconnect. The 2002 system was designed to require a minimal amount of connectors to attach the devices to the actual robot. The new system employs a motherboard/daughterboard design where the analog and digital boards plug into a motherboard that contains all of the necessary connectors. In order to remove either of the main boards, no connectors must be disconnected. Secondly, all connectors are placed near their respective device, removing excess cabling that would otherwise clutter up the robot and make ma intenance difficult.

4.5.3.3. Types of connectors

The types of connectors were also studied in detail for there maintainability. Connectors that were very difficult to connect and disconnect eventually lead to lots of frustration and stress on the system as users try to pull them apart. Connectors were chosen that would allow simple connection, such as the new battery connectors. The battery connectors are larger with accentuated clips that allow ease of use.

4.5.3.4. Wireless module selection

Wireless module selection was never an issue in the past, since there has always been only one type of receiver on board. However, this year's team felt that redundancy was necessary in some form to be able to ensure a dependable wireless communications link. Because of this redundancy, the robot finds itself capable of using 3 different modules. To make the system more maintainable, a special circuit was designed using a DIP switch bank and some tri-state buffers to handle the intelligent switching of module control. This system is described in more detail in the wireless section of the document. With this small amount of design, we have been able to create an easily maintainable system, allowing users to quickly change wireless modules and control code with only a few simple steps.

4.5.3.5. Design for Robustness

Designing for robustness is a critical component to the design process, especially for a team of robots that will face hands-off competition. Many robots don't have as precise control or obstacle avoidance as our system does. Therefore, we find our robots getting pushed and knocked around. Our drive systems are also powerful enough to cause components to experience stresses. Quick motions of the drive system and constant abuse can degrade the performance of our system. The main areas of focus here were the following items:

- Enlarging PCB trace width
- Observing current ratings
- Inter-board connection
- System reset problems
- Wireless communications flooded

4.5.3.5.1. Enlarging PCB trace width

The current 2001 system had larger printed circuit board (PCB) layout traces for digital and high current analog signals. We studied the entire system to determine the magnitudes of the current drains that we would find in our system. One of the conclusions was that the traces should be increased to handle larger amounts of current for longer periods of time. The result has the most impact on the drive and kicking subsystems. If the drive motors get jammed and the controller continues to force the motor in a certain direction, the motors will begin to drain a lot of current. This current drain could potentially lead to a burnt trace on the board rendering it almost unusable. It is especially difficult to repair a board having undergone this stress and be able to guarantee its future performance. For these reasons, we enlarged the traces to anywhere from 0.035" to 0.050" width on all devices that required large amounts of current. To be frank, the actual widths used were not selected via any analytical method. Traces were enlarged in proportion to their estimated worst case currents, and to the expected length of traces after routing. Please refer to the board layouts in the Appendix for a closer look at these improvements.

4.5.3.5.2. Observing current ratings

Along the same lines as increasing trace width is examining the components that accompany those traces. Most of the inter-board connectors and device connectors were changed due to the current flowing through them. The average current was measured in most cases and then a worst case scenario was employed to raise the current ratings to levels that would not be seen by the robot. This over specification of the current guaranteed or at least minimized the probability of a component failure in the system. Specific examples of this detail include the larger battery connectors and the Futurebus connectors used for inter-board communication. Different types of connectors were used for digital and analog systems. Furthermore, multiple pins were used in order to increase the amount of current capable of flowing through a connector when the worst-case current ratings were not met.

4.5.3.5.3. Inter-board connection

The current design consists of a motherboard and two daughterboards (analog and digital boards). Because of the stress that the boards feel from being mounted perpendicular to the ground, we needed to ensure the robustness of the design. Through the use of specially designed bus connectors, compact sizing, stabilizing bars we were able to accomplish this mounting and maintain a robust system. The Futurebus connectors that we used had very long pins and deep sockets to enable a solid contact. The connectors had a tight fit that was still simple to disconnect. The mounting of the connectors to the actual daughterboards was also very sturdy. The boards were designed more in the shape of a rectangle than a square also partly due to space requirements. The compact design coupled with solid bus connectors gave us most of the stability that we required. As added security the implementation of two rigid post that will cradle the edges of the board perpendicular to the ground will also strengthen our mounting solution.

4.5.3.5.4. System reset problems

System reset and boot-up sequence are also very important to the success of the robot. The 2001 system robots used PID chips as the basis for their motion control. These PID chips would occasionally reset causing the whole system to need a reset. The time while the reset occurs could be a crucial moment in the game. Secondly, we experienced problems with our current distributed computing system stemming from the timing of starting up different subsystems of the robot. This issue was easily fixed through well thought out startup sequences of subsystems such as wireless, motion control, and main robot control. Through careful timing we can now guarantee that the robot will be in a controlled state at all times.

4.5.3.5.5. Wireless communications flooded

One last area of work that could be a possible weakness in the robustness of our system is the wireless communications. Although wireless communications can be difficult to maintain a constant connection, there are also other factors that are under the users control. If for some reason the artificial intelligence floods the wireless system by sending two packets in succession, our system needs to be designed to handle this problem and not freeze up. The current 2002 robots have been designed to only send the first packet that has been received and decoded and not flush out its memory to start the process again. More details on this design can be found in the wireless section.

4.6. Preliminary Design

Preliminary designs were drawn to meet the requirements gathered. Several designs were discussed in group meetings to gauge their feasibility before additional work was done.

4.7. Preliminary Testing

Each design was constructed and thoroughly tested on breadboard. Several quality measures were used to test the circuit, including noise tolerance, resistance to physical shock, robustness given variability in component values, and possible failure modes given improper supply voltages. Circuits also underwent peer review by group members for design verification.

4.8. Manufacturing

Manufacturing is an extremely difficult and tricky process for newcomers because of the attention to detail necessary to arrive at a perfect design. Our new board designs had a higher cost than previous years and therefore, should have elicited more care on our part. Because of delaying prototyping and testing, we shipped designs that weren' t fully tested, which is an absolute guarantee for failure. Designs were sent out at the last minute after sleepless nights when attention to detail diminishes.

The above-mentioned behaviors led to the release of a design that was manufactured incorrectly with only ourselves to blame. Our motherboard was missing an internal routing layer, which we thought rendered our initial boards useless. We also found mistakes on our digital board and continued to submit the wrong files, illustrating our lack of revision control, leaving us with errors that we already knew existed. The morale of the team was extremely low, as we had spent large amounts of money on

unusable boards. The situation was made even worse by the fact that we had many surface mount components with extremely small pitch sizes.

With large amounts of teamwork, determination, and many sleepless nights the team was able to hand-populate the small components including a 100 pin Altera FPGA. Modifications on the digital board were made and connections for an entire layer were made on the motherboard. After many creative soldering and rework strategies, the team managed to successfully get the incorrect boards to function correctly. Photos of the salvaged boards can be seen in the Appendix.

In the end, a second revision was sent out and manufactured without any other errors. Recommendations and lessons learned are described below:

- Make a majority if not everyone check and "sign off" designs to ensure that fresh eyes get a chance to evaluate others' work.
- Do not focus resources on areas that do not have a high probability of failure such as the auto-routing feature of Orcad Layout.
- Make sure there are redundant experts in the use of Orcad Layout and make sure that the routing window is large and not too small. Small routing windows lead to long routing times and a reduced quality of routing. The routing box size can be changed by adjusting the "route grid spacing" in the system settings box. See the Orcad Layout help files for more information.
- Follow our main strategies to checking the board

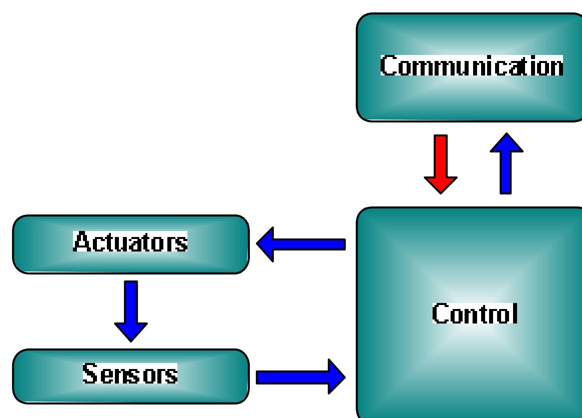
    - CHECK THE DESIGN! The majority of errors were purely design errors that slipped through testing, were mistakenly entered into schematic capture, never breadboarded to ensure proper design, etc. Check the schematic against a fully operational breadboarded design.
    - Check modifications against the Board Issues List mentioned in the Group Electronic Documentation section.
    - Check all parts for power and ground connectivity.
    - Check part pinouts between schematic and data sheets.
    - Check all part footprints versus datasheets, and model the board by sticking parts through a real-size printout of the layout that is laid on top of a Styrofoam base.
    - Ensure that all traces originate from a pad or via and end at a pad or via. Only a couple of errors originated from the use of auto-routing. One trace had a small break in it and a couple of traces ran too close to board vias.

- Board Rework Strategies !!!

    - Surface Mount Components – SMT
        - Use "clay" under body of SMT parts to hold them in place while you solder them to the board. Secondly, if you are resoldering and the pads are dirty, this will allow you to line up the leads without a problem.

        - Cut solder into "chips" with an knife into pieces that are almost invisible. Touch the soldering iron to the chip of solder and then to the pin to solder.
        - Use fine gauge wire to "thread" under a chip's lead then heat up with a soldering iron and gently lift the lead. This helps when you need to solder something to a pin while disconnecting it from the circuit.
        - Bend axial lead resistors and solder them to pads if you don't have any surface mount resistors.
        - If the SMT package size is too small, take a chip carrier and solder wires to the carrier and then carefully solder the wires to the board or vias that connect to the footprint.
        - It is possible to solder to SMT pins, just don't hold the soldering iron on them too long. It is preferred to put solder on a wire and then heat it up against the lead.
        - To solder resistors, capacitors, and LEDs, place a small amount of solder on the pads initially and then place the part on there and carefully heat up the pads individually.
    - Use shrink tubing whenever possible to eliminate possible contacts and shorts.
    - Ensure no cold joints by soldering carefully and correctly.
    - Check part numbers and orientation before soldering.
    - It is possible to solder on to exposed traces!
    - Cut traces using a sharp knife and "scratch away" layer above trace, then take piece of trace out.
    - Check package types vs. footprints on circuits for component interference.
    - The silkscreen is important for population and debug.

Overall, be careful and check all of your work twice. Errors will occur, but they can be fixed and learned from to lead you to a successful outcome.



5. Digital System

5.1. Main Microcontroller

5.1.1. Overview

The main microcontroller is responsible for regulating all on-board operations. The following diagram depicts the sub-systems managed and information flow between the microcontroller and the sub-systems.

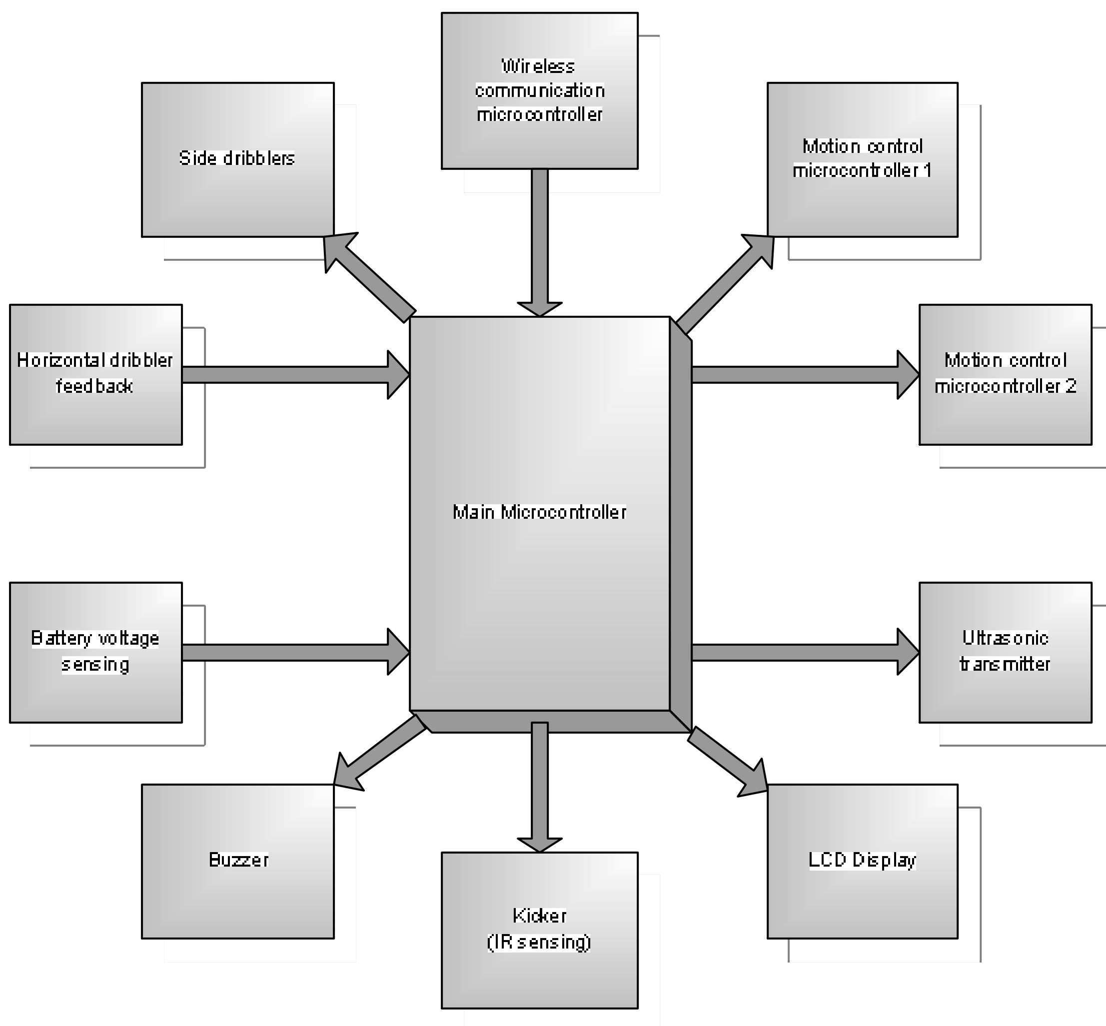


Figure 5.1.1: Interaction with different sub-systems

Since many sub-systems depend on the main microcontroller, the choice of controller was of pivotal importance. Numerous criteria were considered before choosing the final microcontroller; the key criteria are discussed in later sections.

5.1.1.1. Microcontroller comparison

The 2001 system used a PIC17C766 microcontroller. Despite many useful features, it had one serious drawback. The memory of the microcontroller was not electrically erasable; rather it had to be UV erased for approximately 30 minutes before reprogramming. This led us to investigating other microcontroller options.

Many microcontrollers were investigated in view of our system requirements. Some of the microcontrollers that were short-listed are shown in the table below.

Features		Microchip	Microchip (Flash)	Microchip (Flash)	Atmel (Flash)
		PIC17C766	PIC16F877	PIC16F876	AT91M55800A
Maximum Frequency		33 MHz	20Mhz	20 MHz	33 MHz
Operating Voltage Range		3.0 – 5.5V	2-5.5V	2-5.5V	2.7 – 3.6V
Program Memory	RAM	8K	8K	8K	8K
	FLASH	-	-	-	-
Timers		4	3	3	3
Capture inputs		4	2	2	-
PWM outputs		3	2	2	12
USART/SCI		2	1 HR*	1	2
A/D Channels (10 bits)		16	8	5	8
Watchdog Timer		Yes	Yes	Yes	Yes
External Interrupts		Yes	Yes	Yes	Yes
I/O Pins		66	34	22	147

Table 5.1.1: Comparison of features of some microcontrollers

* there is one hardware SCI/USART but all digital I/O pins can be programmed to be USART pins.

It must be kept in mind that the features given in the table above were not the only criteria used in our decision. Several important considerations that tilted the balance in favor of PIC16F877 are given in the next section.

5.1.1.2. Selection criteria:

From table 5.1.1, it is obvious that all the microcontrollers listed have many useful features. While the number of I/O pins is limited in PIC16F877 as compared to the some of the other microcontrollers available in the market, it has many other attractive features which led to it being chosen. Some of these features are discussed below.

1. The microcontroller can be programmed through flash memory in approximately 30 seconds. This is a great improvement to the PIC17CXX series controller used last year which had to be erased in a UV Eraser for almost 30 minutes before reprogramming.

2. The same microcontroller is being used to implement wireless communication packet handling and proportional motion control. This simplifies our work because there is only one development environment/programming setup to learn.

3. The microcontroller must support enough interrupts to implement an interrupt based system. Previous years used a polling system where all operations ran sequentially in an infinite loop that polled for change. Interrupt based systems are more responsive and robust, and so we needed to have a device capable of handling all the interrupts. The PIC16F877 has 13 interrupts and thus proved ideal.

4. It provides a high speeds for most basic operations, and is reasonably fast for multiply and ADC conversion.

Based on our expectation of a higher pin count than last year (where 58 I/O pins were used) we decided initially to go with the Atmel AT91M55800. However, there were some problems encountered. It proved difficult to program, had an unreliable simulator and a inadequate debugger. A significant reduction in pin count requirements (due to switching from PIDs to p-control, and from offloading wireless to a PIC) allowed us to reconsider our microcontroller choice, and we switched from the AT91M55800 to the PIC16F877. A detailed discussion of the change is given in the Appendix C.

5.1.1.3. Microcontroller features:

Some of the salient features of the PIC16F877 are given below.

Architecture:

- High performance 8-bit RISC CPU
- Only 35 single word instructions to learn
- All instructions execute in a single cycle except branches which take two
- Operating speed: DC - 20 MHz clock input
- DC - 200 ns instruction cycle

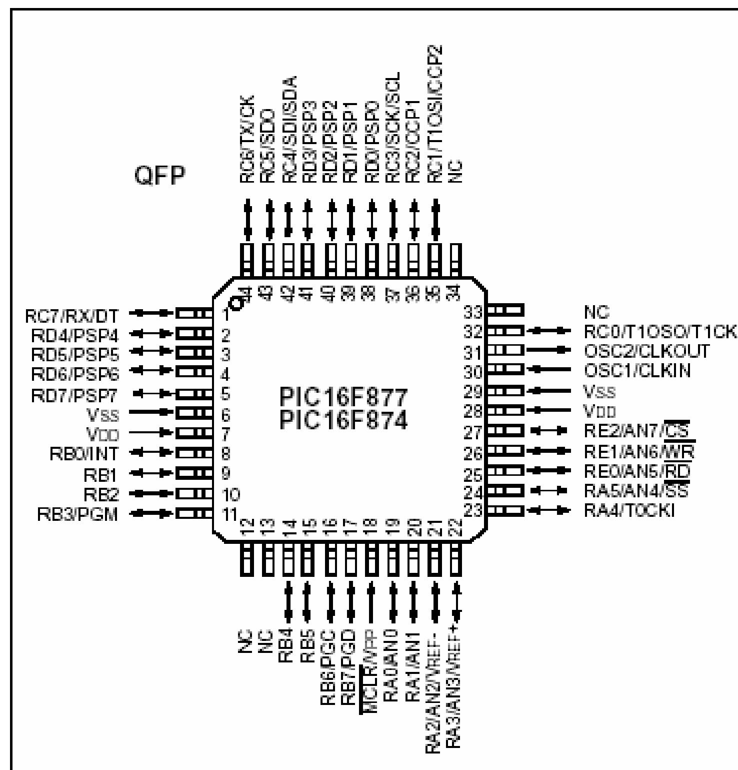


Figure 5.1.1.3: The pin diagram of the PIC16F877 (QFP package type)

Memory:

- Up to 8K x 14 words of FLASH Program Memory
- Up to 368 x 8 bytes of Data Memory (RAM)
- Up to 256 x 8 bytes of EEPROM Data Memory

Compatibility:

- Pinout compatible to the PIC16C73B/74B/76/77
- Interrupt capability (up to 14 sources)

Miscellaneous:

- Eight level deep hardware stack
- Direct, indirect and relative addressing modes
- In-Circuit Serial Programming. (ICSP) via two pins
- Single 5V In-Circuit Serial Programming capability
- In-Circuit Debugging via two pins
- Processor read/write access to program memory

Electrical Characteristics:

- Wide operating voltage range: 2.0V to 5.5V
- High Sink/Source Current: 25 mA
- Commercial, Industrial and Extended temperature ranges
- Low-power consumption:
  - < 0.6 mA typical @ 3V, 4 MHz
  - 20 µA typical @ 3V, 32 kHz
  - < 1 µA typical standby current

5.1.2. System Features

5.1.2.1. Interrupt based implementation

A main improvement to robot microcontroller functionality came from our shift from a polling based system to a completely interrupt based system. There are several advantages of using an interrupt based system as opposed to a polling/looping system:

1. Reduced latency
   The system is designed to optimize the performance and to reduce the delays inherent to a polling based system. All time-critical tasks are implemented in interrupt service routines. Many of the internal time consuming functions have also been implemented such that there is no polling thus eliminating unnecessary internal latency. This essentially reduces response time to all time-critical functions (e.g. wireless packet available, IR sensor beam broken, etc.)

2. Improved reliability
   The interrupt based system improves the reliability because the microprocessor core ensures that incoming interrupts are serviced even if they appear at the

same time. Thus there is little chance of critical functions being missed or delayed.

5.1.2.2. Easy to Debug

The system can easily be debugged by "listening" to the inter-microcontroller communication. All the microcontrollers communicate serially to each other, so we can view data passed between them on the serial port using HyperTerminal (a general serial port communication program available in Windows).

This provides an easy way to isolate points of failure.

5.1.2.3. Fast / More reliable system

The 2002 system with many watchdog codes to cater for the different contingencies and with its interrupt based nature should not only be more responsive but also much more reliable than the previous systems. We have eliminated the PID controllers from previous years that were known to reset the system on many occasions.

Thus the major emphasis in designing the system has been on reducing the latencies and on making the system more robust.

5.1.3. Main Microcontroller pin allocation

The pin allocation table shows the name of the pins as well as the functions associated with each pin in the microcontroller.

Pin #	Pin Name	Functions implemented
1	RC7RX/DT	Wireless IN
2	RD4/PSP4	Motion Control PIC1
3	RD5/PSP5	Motion Control PIC2
4	RD6/PSP6	Speaker Out
5	RD7/PSP7	LCD Serial
6	Vss	Vss
7	Vdd	Vdd
8	RB0/INT	IR (Interrupt)
9	RB1	Kicker
10	RB2	Ultrasonic 1
11	RB3/PGM	PGM
12	NC	*No connection*
13	NC	*No connection*

14	RB4	Motion Control Interrupt1
15	RB5	Motion Control Interrupt2
16	RB6/PGC	PGC
17	RB7/PGD	PGD
18	MCLR/VPP	Reset
19 20	RA0/AN0 RA1/AN1	Kicker voltage in Battery voltage in
21	RA2/AN2/Vref-	Gnd
22	RA3/AN3/Vref+	Vdd
23	RA4/TOCKl	Ultrasonic 0
24	RA5/AN4/SS	Dribbler feedback
25	RE0/RD/AN5	Input for goalie
26	RE1/WR/AN6	
27	RE2/CS/AN7	
28	Vdd	Vdd
29	Vss	Gnd
30	OSC1/CLKIN	Clk
31	OSC2/CLKOUT	Clkout
32	RC0/T10SO/T1CKl	Debug RX
33	NC	*No connection*
34	NC	*No connection*
35	RC1/T1OSl/CCP2	Dribbler PWM
36	RC2/CCP1	Goalie Enable/Side dribbler
37	RC3/SCK/SCL	Goalie direction
38	RD0/PSP0	DIP0
39	RD1/PSP1	DIP1
40	RD2/PSP2	DIP2
41	RD3/PSP3	DIP3
42	RC4/SDI/SDA	Motion RX1
43	RC5/SDO	Motion RX2
44	RC6TX/CK	Wireless Out

5.1.4. Function description:

The main microcontroller program can be divided into the following parts.

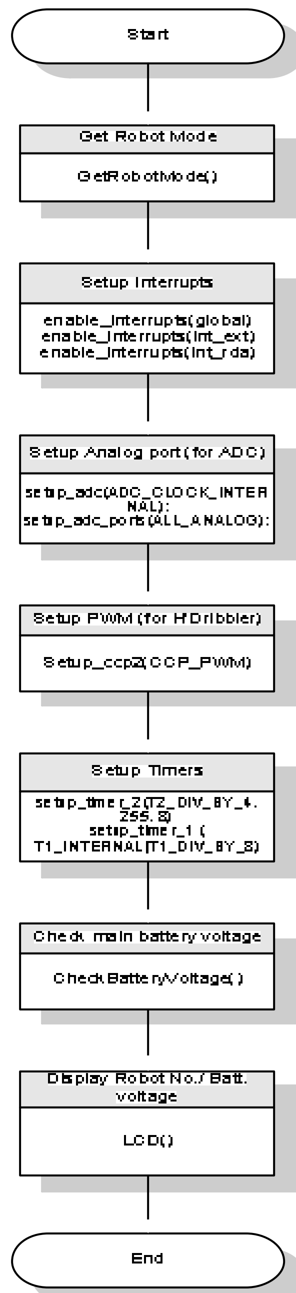
- Initialization
- Receiving wireless packet
- Parsing/processing packet
- Kick
- Horizontal dribbler control
- LCD display
- Calculation of wheel velocities
- Sending motion vectors
- Battery voltage check
- ‘Beep’ enable

All of these parts of the code are described in this section.

5.1.4.1. Initialization

Function name:	void Initialize(void);
Input/output:	None
Function(s) called:	None
Description:	This function is used to initialize all the ports, setup counters, interrupts and check if the battery voltage is up to acceptable level for operation of the robot. This function is also responsible for checking the ID of the robot as set by the ID DIP switch on the robot. Another thing that this function implements is that it prints the value of the battery level and the ID of the robot on the debug LCD screen on the robot. This function is called after either a power up or after a reset.

The following diagram lists all the sequence of events that take place in this function.


Start
Get Robot Mode
GetRobotMode()
Setup Interrupts
enable_interrupts(global)
enable_interrupts(int_ext)
enable_interrupts(int_rda)
Setup Analog port (for ADC)
setup_adc(ADC_CLOCK_INTERNAL);
setup_adc_ports(ALL_ANALOG);
Setup PWM (for H'Dribbler)
Setup_ccp2(CCP_PWM)
Setup Timers
setup_timer_2(T2_DIV_BY_4, 255, 8)
setup_timer_1 (T1_INTERNAL|T1_DIV_BY_8)
Check main battery voltage
CheckBatteryVoltage()
Display Robot No./ Batt. voltage
LCD()
End

5.1.4.2. Receiving wireless packet

Function name: void GetWirelessPacket(void);
Input/output: None
Function(s) called: AfterWirelessPacket( );
Description: This function is called whenever there is data on the RX line of the main microcontroller (from U2). The purpose of the function is to get the 5 byte packet for the robot depending upon the robot number set by the robot ID DIP switch. It also reads the rest of the packets (that do not belong to the particular robot) to clear the buffer. When the packet is read AfterWirelessData( ) is called to process the received packet.

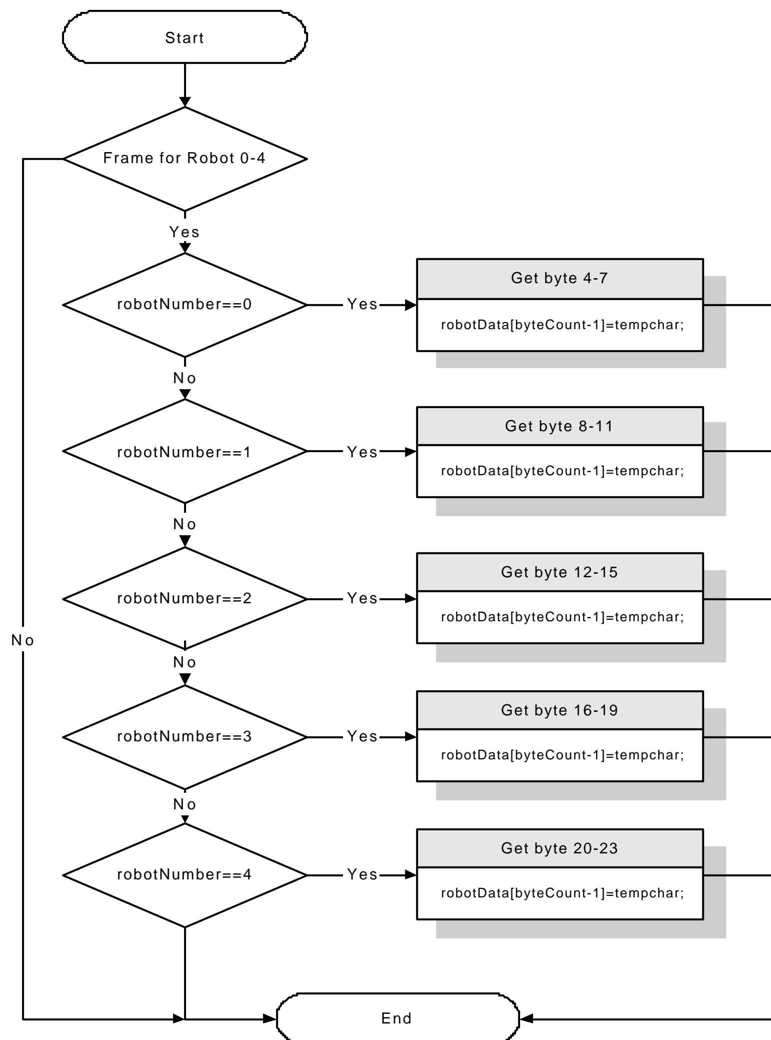
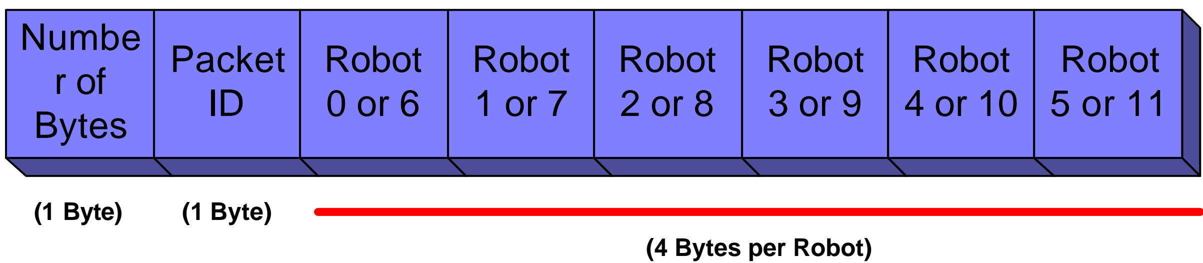


Figure 5.1.4.2: Receiving a wireless packet

5.1.4.3. Parsing/processing packet

Function name: void AfterWirelessPacket(void);
Input/output: None
Function(s) called: Wheel1( );
Wheel2( );
Wheel3( );
Wheel4( );
Kick( );
ControlHDribbler( );
Description: This function is the main control function that performs many operations when the wireless packet is received. It first parses the packet into different commands. Then it converts robot motion vectors into Vx, Vy (m/s) and Vtheta (rad/s) and then into wheel velocities V1, V2, V3, V4. Once the wheel velocities are calculated, it sends them onto motion control microcontrollers (U3, U4). Similarly actions are taken on the kicker and horizontal dribbler depending upon whether AI has activated/deactivated them and if activated, what speeds are indicated.

The wireless packet sent by the AI is in the format shown in the following figure.



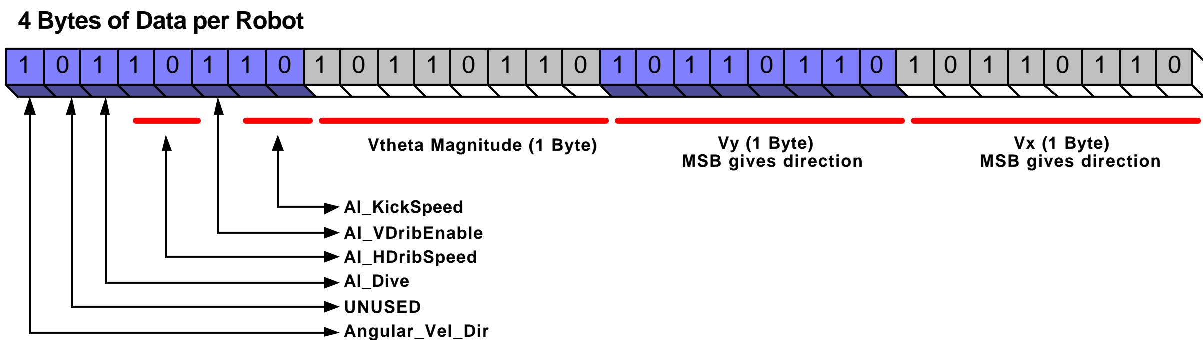
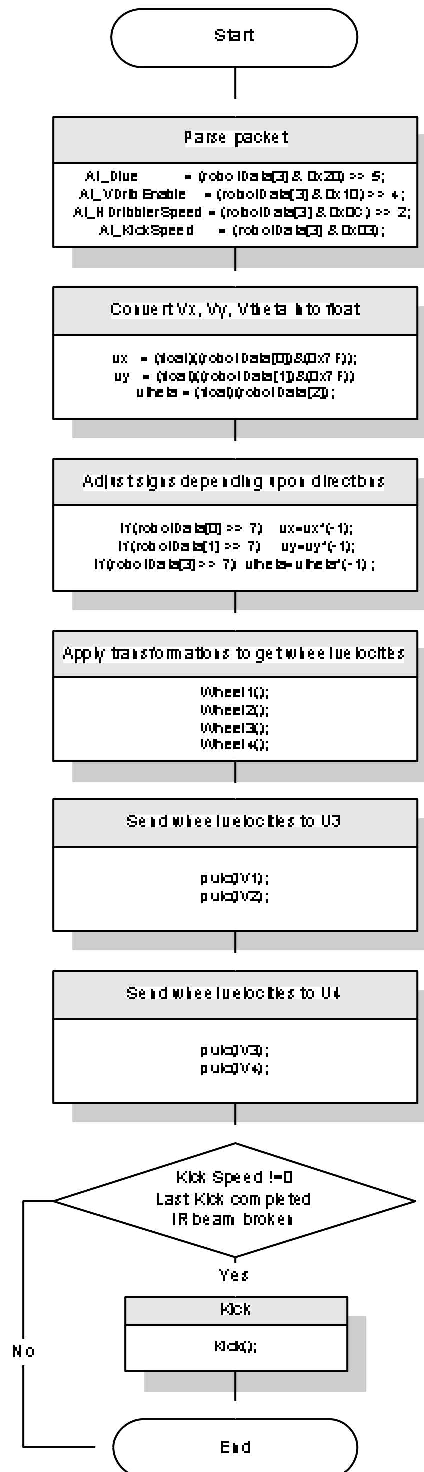


Figure 5.1.4.3.1: Contents of the wireless packet

The AfterWirelessPacket() function is depicted in the figure below.



5.1.4.4. Kick

The kick is interrupt driven and would activate immediately on the external IR interrupt if the AI has enabled the kick in the wireless packet. However there is more than one possible scenario when the kicker system would be activated. The schematic below shows two of the possible scenarios when the kicker would be activated.

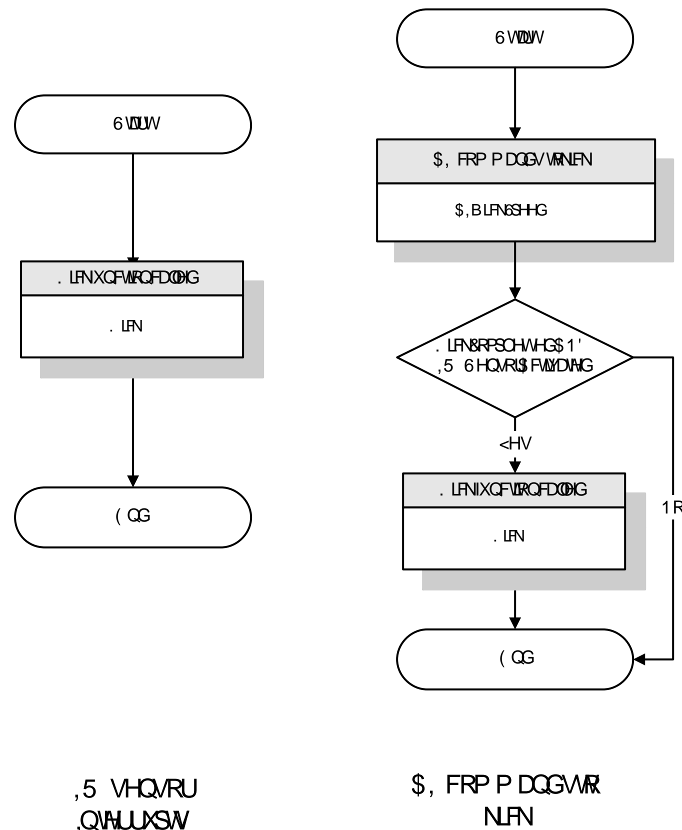


Figure 5.1.4.4.1: Possible events that trigger a kick

1. IR Sensor activated:
   In case there is a interrupt generated by the IR sensor (ball detected) the function Kick( ) is called immediately. This function is discussed in detail below. This function makes sure that the robot never kicks unless it has been instructed by the AI to kick and if the previous kick has not been completed (which includes some time to retract the solenoid).

2. AI commands to kick:
   When the AI in a wireless packet tells the microcontroller to kick the ball, the controller checks whether the previous kick has been completed and the IR sensor has been activated. In case both of these conditions are satisfied, the Kick() function would be called.

Kick( ) function

Function name:	void Kick(void);
Input/output:	None
Function(s) called:	None
Timer(s) Set:	Timer1. Interrupt on overflow
Description:	*Details below.*

The Kick( ) starts a two stage process that does the following

1. It ensures that the kicker would not be re-activated if the kicker is already activated. This is necessary since the time required to get a powerful kick is more than many times the time between the packets and thus it is possible that the AI commands the kicker to kick again even when the previous kick has not been completed.

2. Does not hold up the processor while the kicker is activated since for a strong kick can take up to hundreds of milliseconds as opposed to the time between consecutive frames that is approximately 16ms.

3. It also ensures that there is some minimum time maintained between two consecutive kicks so that the system does not get taxed by consecutive kicking.

The Kick() function checks whether the AI has enabled the kick and if so, is the kick speed. Another thing that it checks is whether the last kick has been completed or is some part of the kicking system is still not ready for a new kick. When the above conditions are satisfied, the function activates the kick and starts a timer with the kick activation time set according to the AI kick speed. The larger the time the kicker is activated the stronger the kick (up to a certain limit).

Timer 1 (based on counter 1 which is a 16 bit counter) is set to a particular value. On overflow the kicker is de-activated and the kicker is allowed to retract for a set time. Each count is incremented every 1.6 us ((1/20,000,000)*4*8). Thus for 2 ms time we need 1250 counts (2ms/1.6us) before the counter overflows and the timer interrupt is triggered.

A flow diagram of the kick function is shown in the figure below.

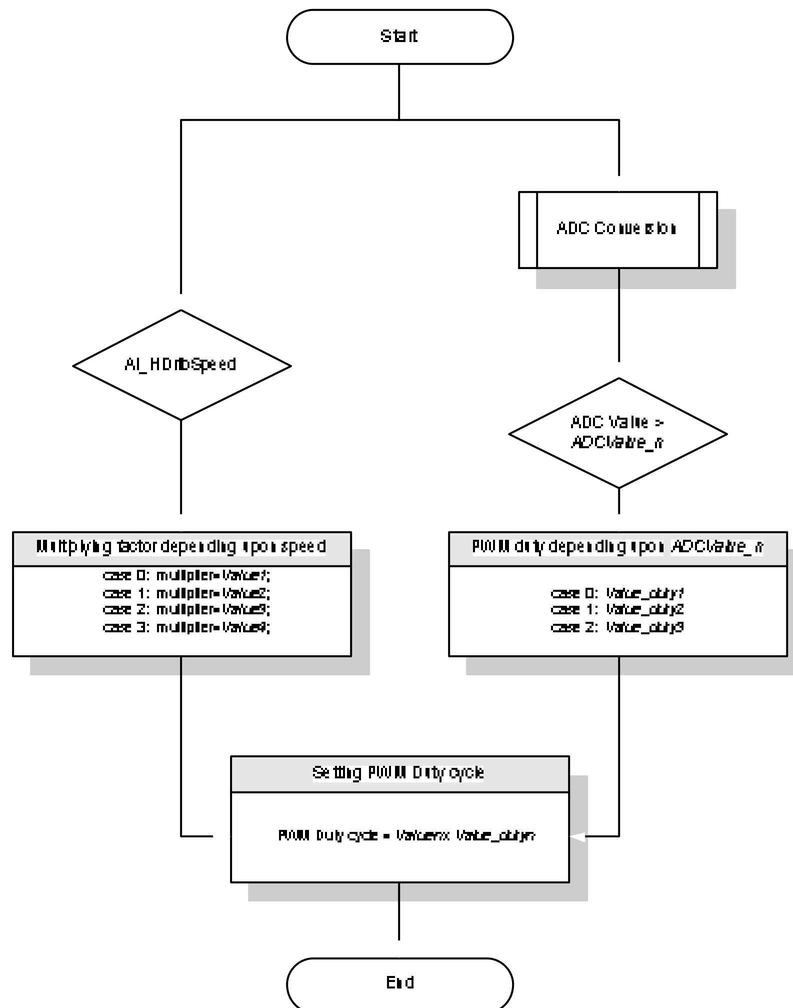
Figure 5.1.4.4.2: Complete kick sequence

5.1.4.5. Horizontal dribbler

Function name:	Void ControlHDribbler(int);
Input/output:	AI_HDribblerSpeed/None
Function(s) called:	None
Timer(s) Set:	Timer2 for generating a PWM output to the analog circuit
Description:	*Details below.*

The main function of the horizontal dribbler is to keep possession of the ball when the ball is kicked towards a robot or the robot is running with the ball.

There are two levels of control for the dribbler. A coarse control is defined by the AI which essentially limits the speed of the dribbler and a fine control is done on the basis of the current feedback circuit. The current measured using the current feedback circuit (described in detail in Section 6.5) is proportional to the torque being applied to the ball. This sensor provides feedback to the microcontroller which is used to change the speed of the dribbler to ensure that the ball is not lost in wake of the different loading conditions on the ball. The load on the ball would be different, e.g., if the robot is running back with the ball as opposed to running forward. This dual feedback system ensures that the ball is not lost in any case no matter what is the type of load on the ball. The dribbler speed is adjusted every time a new wireless packet is received (approximately every 16 ms).



5.1.4.6. Check battery voltage

Function name:	Void CheckBatteryVoltage(void);
Input/output:	None
Function(s) called:	Buzzer
Description:	This function checks the battery voltage every time the robot is either reset or switched on. The value of the battery level is shown as a percentage of the maximum battery level on the LCD display. If the battery level is lower than a certain set threshold then a buzzer sounds twice to warn that the battery level is low.

5.1.4.7. Buzzer function

Function name:	Void Buzzer(int);
Input/output:	Number of times to beep
Function(s) called:	None
Description:	This function makes the buzzer beep. The number of times that the buzzer would beep depends upon the input argument to this function.

5.1.4.8. Robot mode switch

Function name:	Void GetRobotMode(void);
Input/output:	None
Function(s) called:	None
Description:	This function is used to read the value for the robot ID set using the ID DIP switch on the robot. Many operations of the robot are dependent upon its ID like going into test modes, setting robot's number for a game, etc.

5.1.4.9. LCD Display

Function name:	Void LCD (void);
Input/output:	None
Function(s) called:	None
Description:	This function can be used to write useful robot and debug information on the LCD screen. Currently the battery level and the robot ID are displayed on the screen.

5.1.4.10. Wheel Velocities

Function name:	void Wheel*n*(void);
Input/output:	None
Function(s) called:	None
Description:	There are four functions that use the information in the packet for Vx, Vy and Vtheta and convert them using the geometry of the robot into the appropriate wheel velocities for the four wheels of the robot.

This function also encodes the values of the wheel speed in m/s into a 7 bit number for the magnitude and one bit for the direction of the velocity.

See section 5.3.2 for details on the transformations used in calculating the wheel velocities.

5.2. Wireless Microcontroller

For this microcontroller we are currently using the legacy system. The code will be rewritten for the upcoming competition. Refer to RoboCup 2001 for documentation on its current configuration.

5.3. Motion Microcontroller

5.3.1. Overview

The motion control system is comprised of the hardware and firmware that interprets motion vectors sent from AI and transforms them into the physical motion of the robot. The lifecycle of these vectors are as follows:

1) The vectors for each robot are calculated by AI based our current strategy.
2) The motion vectors for all robots are transmitted wirelessly in a 25byte frame, intermingled with non-motion instructions for all the robots.
3) The wireless system reads off the vectors and forwards what is applicable to the main micro.
4) Inside the main micro, the global motion of the robot: Vx, Vy, and Vtheta is converted to 4 individual wheel vectors.
5) The wheel vectors are sent to the wheel-control micros, which update their desired speeds and use proportional control to maintain those speeds.

5.3.2. Wheel Velocity Transformations

Figure 5.3.1 shows a schematic view of the robot wheel orientations. The transformation used by the main microcontroller to convert the values from the AI to the wheel velocities (W1, W2, W3 and W4) are given by the equation below.

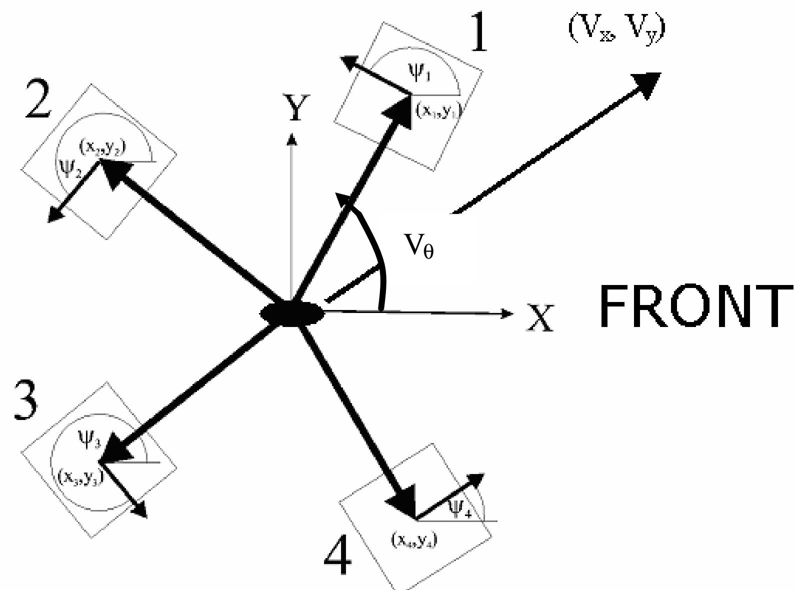


Figure 5.3.1: Drive Geometry

The above diagram represents the robot wheel contact points relative to the (arbitrary) geometrical center of the robot, and wheel drive direction angles relative to the forward (X) direction of the robot. From this, it is relatively easy to obtain the coordinate transformation (see below) between the three-component vector velocity of the geometric center of the robot, represented by $\begin{bmatrix} V_x \\ V_y \\ V_\theta \end{bmatrix}$, and the 4 wheel velocities, $\begin{bmatrix} W_1 \\ W_2 \\ W_3 \\ W_4 \end{bmatrix}$.

$$\begin{bmatrix} W_1 \\ W_2 \\ W_3 \\ W_4 \end{bmatrix} = \begin{bmatrix} Cos(\psi_1) & Sin(\psi_1) & -y_1 Cos(\psi_1) + x_1 Sin(\psi_1) \\ Cos(\psi_2) & Sin(\psi_2) & -y_2 Cos(\psi_2) + x_2 Sin(\psi_2) \\ Cos(\psi_3) & Sin(\psi_3) & -y_3 Cos(\psi_3) + x_3 Sin(\psi_3) \\ Cos(\psi_4) & Sin(\psi_4) & -y_4 Cos(\psi_4) + x_4 Sin(\psi_4) \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_\theta \end{bmatrix}$$

The angles between the wheel positive drive direction and the forward direction on the robot (see diagram above) are listed below, and are calculated from the design in ProEngineer. These have been measured as well, and to the crude limits of the measurement method (protractor), the angles as manufactured agree with the design.

Symbol	Angle Degrees	Angle Radians
ψ_1	145	2.531
ψ_2	225	3.927
ψ_3	315	5.498
ψ_4	35	0.611

The actual values of the x and y locations of the wheel contact points are listed below, and generated from the midlines of the wheels (midway between the inner and outer beads on the omni wheels) in the ProEngineer mechanical CAD files.

Wheel	X (m)	Y(m)
1	145	2.531
2	225	3.927
3	315	5.498
4	35	0.611

In order to permit the AI/Strategy components of the system to easily work with heterogeneous teams comprised of robots of different designs/vintages, the wheel velocity coordinate transformations take place within the robot, rather than on the AI side as was the case in prior years.

It must be born in mind that the total velocity of each wheel contact point is actually the sum of the driven and passive velocity vectors. The four passive velocities can scale as necessary to match the combined constraint of the four driven velocities. It must also be born in mind that with only 3 degrees of freedom, but 4 wheels to control, the not every combination of wheel velocities equates to a conflict-free net motion of the robot - lag or velocity errors in the control of the motors, and geometric errors in manufacturing, measuring or calculating the coordinate transformation can also result in fighting between drive wheels. For more complete discussion of the consequences and conditions for fighting, see the section below on Geometric Tolerances.

5.3.3. Wheel Velocity Calculations

The following calculation is used to relate target counts to speed inside of the motion controller. The full mapping between these speeds is in the document *motocalc.xls*.

Wheel Radius:	2cm
Wheel Circumference	4 * pi = 12.566 cm
Counts / Rotation:	2048
Gear Ratio:	9
Frames / sec:	600

Target Speed:	2 cm / sec
# Rot / sec @ Target:	2 cm / 12.566 = 0.159
Counts / sec @ Target:	0.159 * 2048 * 9 = 2934
Counts / frame:	2934 / 600 = 4.9

The counts/frame is the value which is used by the microcontroller to do velocity control. Currently, 4.9 is approximated as 5 in the interest of simplifying the arithmetic being demanded of the microcontroller (uses integer multiply as opposed to floating pt. multiply).

2 cm/sec was chosen arbitrarily as a good target for the lower bound of robot speed control. The upper bound is around 2.5 m/s. Neither target is a rigid boundary, rather they were chosen because they cover the assumed operating range of the robot. The upper bound was arrived at by calculating the maximum speed which could be achieved given the max acceleration of the robot and the available space on the field. Given this range, the following, simple mapping was used to convert incoming wheel vectors:

wheel vector: [8 bits]: [sign] [7bit magnitude]

7 bit magnitude range:	0-127
Speed range:	0-2.5m/s

Mapping :: magnitude * 2 = speed in cm/sec.
This provides a range of 0-2.54 m/s

5.3.4. Motion Control Variables: Tradeoffs

The calculations above have been done assuming a frame rate of 600 Hz. The system can also be run at 300 or 150 Hz. In the code this can be done simply by setting the GLOBAL_RATE at the tope of the code. The desired counts/frame and proportional feedback strength K will be set accordingly. Setting the GLOBAL_RATE will adjust how often the p-controller will control. The rule of thumb for dynamic system control is that the control rate should be at least 10 times as fast as the fastest dynamics of the system. Estimating that the fastest robot dynamics of import happen around 10Hz, control rates of 100Hz or faster should be acceptable. That said, the full ramifications of the various control speeds have not been examined in detail. Ostensibly, faster control rates will have more problems at slow speeds since they will collect fewer counts given their short frames. However, faster control rates might provide an advantage because they offer the possibility of implementing traction control.

To increase the drive strength, the BASE_K can be set. Setting BASE_K too high can have an adverse effect at low speeds. For example, if the desired wheel speed is zero and the robot is slightly perturbed in one direction, a high gain will overcompensate and run the wheel hard enough so that it registers counts in the other direction. The next push overcompensates and sends it in the original direction. Oscillations can result. Too low a K can also cause trouble since at slow speeds, where the difference between the desired count and the actual count is small even when the robot is stationary, the value written to the PWMs may not be enough to overcome stall current.

5.3.5. Architecture

Each motion control micro controls 2 wheels. U3 controls the wheels on the left side of the robot, while U4 controls the right. The motion core control code consists of 2 ISRs. One is interrupted by new data on its hardware RX: this routine reads data off the RX line and sets the desired wheel speeds by reading a sign and magnitude off of each byte. The second ISR runs the tight proportional-control loop: this is where adjustments are made to the duty cycle of the motor controlling PWMs so as to match a desired velocity.

In order to do velocity control, one must know what their current velocity and direction are. This is achieved by interpreting the A_{out} and B_{out} encoder signals from each wheel. The number of counts on A_{out} and B_{out} can be mapped directly to wheels speeds, and their relative phase offset indicates the direction of the motor. In order to get the finest count resolution possible, we used the FPGA to create a quadrature encoder signal: simply put it creates a small pulse for each transition in A or B. Direction was collected by using a flip flop that read the value of A as pos edged clocked on B. The quadrature count and direction are then fed into the motion control micro counters, where the counted value can be compared to the desired count, and the appropriate changes can be applied.

5.3.6. Motion Microcontroller pin allocation

Pin #	Pin Name	Functions implemented
1	RC7RX/DT	Motion Vector IN
2	RD4/PSP4	
3	RD5/PSP5	
4	RD6/PSP6	
5	RD7/PSP7	
6	Vss	Vss
7	Vdd	Vdd
8	RB0/INT	
9	RB1	~DIR2
10	RB2	DIR2: Motor 2 Direction
11	RB3/PGM	PGM
12	NC	*No connection*
13	NC	*No connection*
14	RB4	~DIR1
15	RB5	DIR1 : Motor 1 Direction
16	RB6/PGC	PGC
17	RB7/PGD	PGD
18	MCLR/VPP	Reset
19	RA0/AN0	
20	RA1/AN1	
21	RA2/AN2/Vref-	
22	RA3/AN3/Vref+	
23	RA4/TOCKl	Counter : Motor 2
24	RA5/AN4/SS	Dribbler feedback
25	RE0/RD/AN5	Input for goalie
26	RE1/WR/AN6	
27	RE2/CS/AN7	
28	Vdd	Vdd
29	Vss	Gnd
30	OSC1/CLKIN	Clk
31	OSC2/CLKOUT	Clkout
32	RC0/T10SO/T1CKl	Counter: Motor 1
33	NC	*No connection*
34	NC	*No connection*
35	RC1/T1OSl/CCP2	Motor 2 PWM
36	RC2/CCP1	Motor 1 PWM
37	RC3/SCK/SCL	DIR1_IN: Read Direction for Motor1
38	RD0/PSP0	
39	RD1/PSP1	
40	RD2/PSP2	
41	RD3/PSP3	
42	RC4/SDI/SDA	DIR2_IN: Read Direction for Motor2
43	RC5/SDO	ISR debug signal
44	RC6TX/CK	Debugging Output

5.3.7. Rise Time and Steady State Error

From bench tests we have determined the rise-time for a single wheel to accelerate to 1m/s to be approximately 110ms. The Steady State Error is 1.5% using a frame rate of 300 Hz and a K=8. These are in agreement with early simulations which predicted a 100ms rise time.

The experiment was conducted by having a motion micro output a digital high when it first started to control its wheel, and then outputting a low when the desired count was within 2% of the actual count. There are some caveats to this experiment. It was literally conducted on a lab table where the oscilloscope probe had been crudely connected to the output bit. In order to conduct the tests more accurately, a longer tether must be built to relay the output to the scope. This will be necessary if we are to allow the robot to run on the field during the test. Doing so is important because the field material has different frictional properties which may affect robot acceleration. An easier but less accurate means of conducting the test would be to allow the vision system to track how fast the robot is moving.

The test can also be extended by having the output bit flip at several points along the acceleration curve. In other words the output would flip when it had reached 15, 30, ... 90% of its goal. This would fill out the curve and may also give us some indication of what wheel slipping looks like electrically. Also the test could be run with a number of values for K with the hopes of finding an optimal one.

5.3.8. Slow Speed Add-ons

Our current tests have shown that having the robot move at slow speeds is difficult. With lower gain, the robot simply doesn't move. Using a high gain, it is possible to move, but the motion is sporadic, and the wheels are especially sensitive to "jittering." So far, there doesn't appear to be a sweet spot which avoids both problems.

In order to handle slow speeds there are numerous possibilities, here are some:

a. Use a form of integral control.
b. Have a K which changes according to desired wheel speed.
c. Operate at a lower frame rate.

The problem with all of these is that they may introduce instabilities or oscillations at junctures between behaviors (the point where a lower K meets a higher, or the speed above which integral control is abandoned). These problems can not be definitively ruled out without proper calculation and simulation.

5.4. FPGA

5.4.1. Overview

The Field Programmable Gate Array (FPGA) plays an important role in two systems on the robots: the motion control system and the ultrasonic system.

In the motion control system, the FPGA provides an interface between the motor encoder and the motion control microcontrollers (PIC16F877). It takes the two motor encoder signals and provides a quadrature signal which gives more resolution to our motion count. It also keeps track of motor direction.

In the ultrasonic system, the FPGA is used to generate the 40kHz pulse which is necessary to run the ultrasonic transmitter.

5.4.2. Selection of the FPGA

Altera has been a generous sponsor from the past, and this year was no exception. They donated development software, design laboratory kits, and the FPGAs used in final production. We compared a number of FPGAs in the MAX 7000 family to choose one that satisfied our requirements.

One major factor we considered was in-system programmability since the FPGAs are to be programmed on board. MAX 7000S devices are in-system programmable via an industry-standard 4-pin Joint Test Action Group (JTAG) interface. The MAX 7000S architecture internally generates the high programming voltage required to program, making it programmable with a 0-5V input.

While most of the FPGAs in the MAX7000S family meet our needs, our final board design uses the EPM7128STC100-7 FPGA because it is small and surface mount. With a capacity of 2500 gates, it leaves us room for future additions. For our actual implementation and functional testing, we used the EPM 7128SLC84-7 which comes with the Altera Design Laboratory Package.

5.4.3. Features of EPM7128S

- High-performance, EEPROM-based programmable logic devices
- 5.0-V in-system programmability (ISP) through the built-in IEEE Std. 1149.1 Joint Test Action Group (JTAG) interface
- High logic density offering 2,500 usable gates
- 5-ns pin-to-pin logic delays supporting up to 175.4-MHz clock frequencies
- 3.3-V or 5.0-V operation MultiVolt I/O, allowing devices to interface with 3.3-V or 5.0-V devices

Feature	EPM7032S	EPM7064S	EPM7128S	EPM7160S

Usable gates	600	1,250	2,500	3,200
Max. User I/O pins	36	68	100	104
T_{PD}(ns)	5	5	6	6
T_{SU}(ns)	2.9	2.9	3.4	3.4
T_{FSU} (ns)	2.5	2.5	2.5	2.5
T_{CO1}(ns)	3.2	3.2	4	3.9
T_{CNT}(MHz)	175.4	175.4	147.1	149.3

Table 5.4.3: MAX 7000S Device Features

The number of usable gates determines how complicated our circuit can be. The EPM7128S family has 2,500 usable gates. As a point of reference, our final design utilizes 36% of all the usable gates. This covers four 6-bit counters to generate quadrature encoding, one 8-bit counter for ultrasonic frequency generation, and miscellaneous logic to support enable functionality. The maximum number User I/O pins for the EPM7128S is 100 and we used 22 in our final design. The bottom five rows following rows describe setup time for the clock and inputs. These numbers are all on the order of nanoseconds and are easily good enough for our design. Our final circuit runs at 20MHz.

5.4.4. Detailed Implementation

In the motion control system, the FPGA takes two input signals from each motor encoder. These are encA and encB. These two signals are square pulses and are 90 degree out of phase from each other. We can derive both direction and velocity information from these two signals. Figure 5.4.4.1 below is a sample of the possible input and its corresponding output.

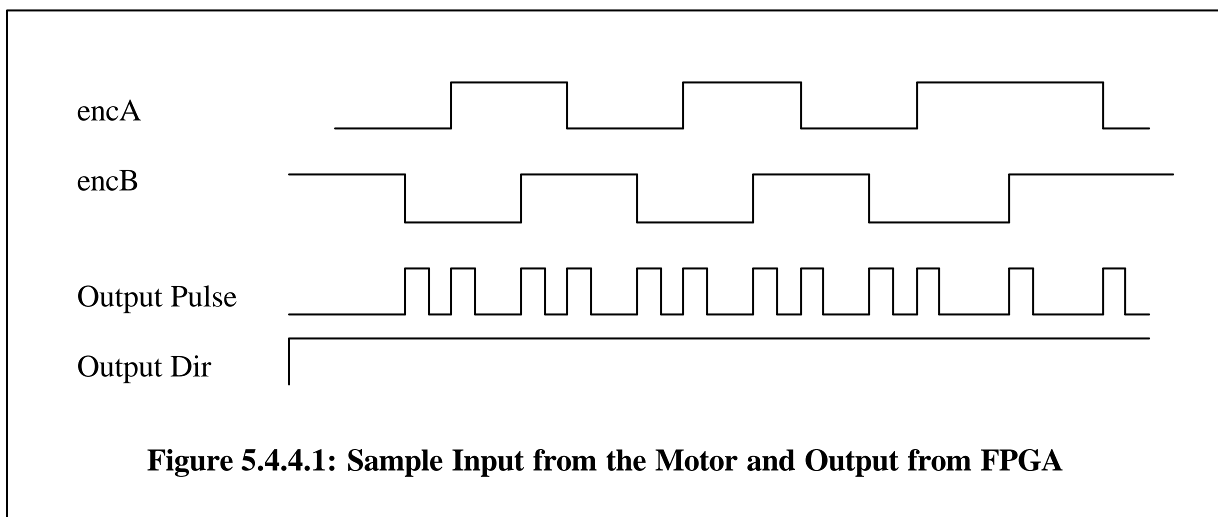


Figure 5.4.4.1: Sample Input from the Motor and Output from FPGA

The second function the FPGA serves is in the ultrasonic system. The FPGA gets an enable signal from the main microcontroller and outputs a 40kHz signal as long as the enable is high. This system is used in the scenario when AI has asked the robot for this

ball status, and the robot has the ball. The ultrasonic transmitter is sensitive to the signal frequency. The signal it transmits will is greatly attenuated if the signal is off from 40kHz. Figure 5.4.4.2 below is a sample of input from the ball detector and the output of the FPGA

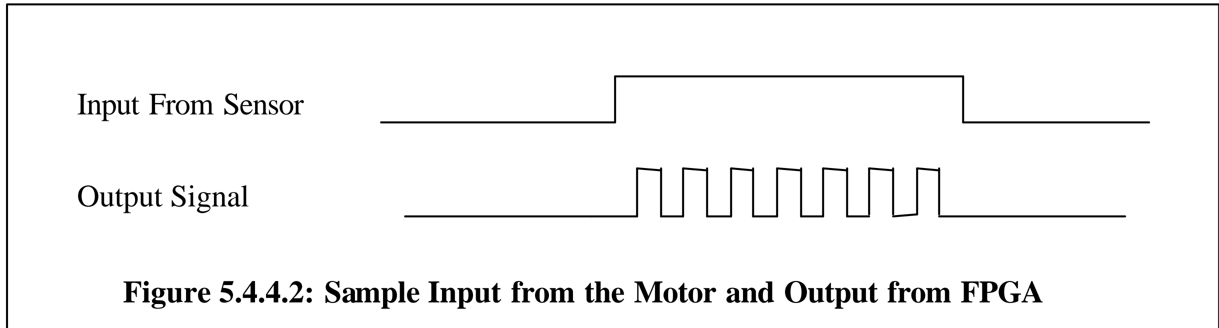


Figure 5.4.4.2: Sample Input from the Motor and Output from FPGA

We have first tried to implement the circuit using Verilog, however, some unstable behavior is observed. We eventually used schematic to implement the circuits. The actually circuits are attached in the appendix.

5.4.5. Special considerations:

While the circuit implemented might look simple, there are several things we have to take into special consideration. These considerations are discussed below:

5.4.5.1. HDL (Hardware Description Language)

When using HDL or Verilog to design circuits, the FPGA can only take one module as the final design. When we started with the FPGA, we initially used Verilog to design our circuit, since it is easier to understand and maintain (it is similar to C code). However, during our design, we experienced some unstable behavior. Our design functioned correctly with the addition of some meaningless print statements, but it does not work correctly without the junk code. We suspect this has to do with how the FPGA understand a sequential circuit written in Verilog, but we never found the root of the problem. It should also be noted that our simulator results were always correct during these test, despite the fact that the actual circuit was nonfunctional. To finally get our design to work we used the schematic view to draw our gates and counters directly. The result is very stable but slightly harder to extend.

5.4.5.2. Pin Assignments

One advantage of using the Altera FPGA is that the programmer lets you assign signals to different I/O pins. This can be helpful when routing since pin locations can be changed to ease routing. However, when routing is not a problem (this year it wasn't an issue), it is always a good idea to follow the suggested optimal layout that the

MAX+plus II compiler gives. Once the final design/FPGA functionality is set, the final pins can be noted off the auto-generated pin layout.

5.4.5.3. Dedicated Pin Connections

In addition to I/O pins and CLK pins there are also a number of pins which need to be connected to ground or VDD in order for the FPGA to work properly. Table 5.4.5.3 lists these pins. Besides these pin assignments, small value capacitors (0.1 uF) should be connected the power pins so that the FPGA can suppress noise in the power supply.

Dedicated Pin	84-Pin PLCC	100-Pin TQFP
INPUT/GCLK1	83	87
INPUT/GCLRn	1	89
INPUT/OE1	84	88
INPUT/OE2/GCLK2	2	90
TDI(3)	14	4
TMS(3)	23	15
TCK(3)	62	62
TDO(3)	71	73
GNDINT	42,82	38,86
GNDIO	7,19,32,47,59,72	11,26,43,59,74,95
VCCINT(5.0V only)	3,43	39,91
VCCIO(33.v or 5.0V)	13,26,38,53,66,78	3,18,34,51,66,82
Total User I/O Pins	68	84

Table 5.4.5.3 Dedicated Pins for the FPGA

5.4.5.4. Clock Frequency Consideration and others

Since the circuits for both the motor control and ultrasonic use the global clock as a counter reference, it is very important to modify the design if the global clock frequency changes. For example, sometimes designs that work on the development board will not work on the actual robot circuit boards because they have different clock frequencies. This is doubly important for the ultrasonic transmitter because it is sensitive to minor frequency variations.

5.5. Ultrasonic

The ultrasonic system allows the robots to give feedback to the AI. Primarily it is used to allow a robot to say that it has possession of the ball. This is useful in situations

where the ball is occluded since the vision system can not verify possession if it can not see the ball. The system could also be used to send other data as well, but only at very limited bandwidth (globally BW: 240 bps, max). Though our system does not currently do this, the robots could use these bits to indicate whether they are being pushed illegally. Despite the bandwidth limitations, we have more bandwidth than we have data to communicate. This is mostly a function of not having many status sensors to report.

5.5.1. Transmitter

The design of the transmitter involves four parts: physical transmitter, max203, FGPA, and main micro. At the output it is just the ultrasonic transmitter attached to the output of an RS203. The 203 serves as an amplifier, generating a +/- 12V signal from the 0-5V signal it receives at its input. The input to the 203 comes from the FGPA and is a 0-5V square wave at 40kHz which is switched on and off by the main micro. The enable pin is a standard micro I/O pin, so it is the envelope that is applied to the 40kHz pulse is completely determined in firmware.

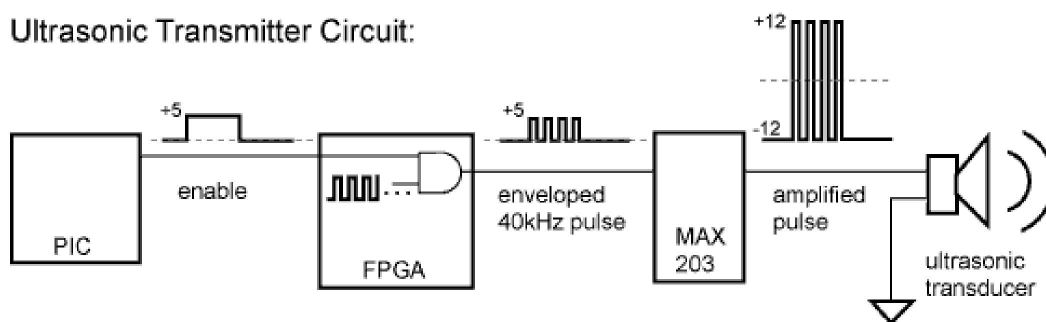


Figure 5.6.1: Ultrasonic Transmitter Circuit

The robots all transmit at the same frequency (40kHz), so it is necessary to do simple TDMA to distinguish which robot is which. AI will chose which robot it would like to hear during a given frame, and only that robot will be allowed to speak during the allotted time slot. This eliminates the addressing/decoding complexity in the transmitted signal; the receiver need only look for the presence or lack of a signal.

5.5.2. Receiver

There is only one receiver in the circuit which handles the transmissions for all the robots on the field. Its main goal is convert the relatively weak (2-5mV) 40kHz pulses into a coherent binary verdict. The receiver design consists of op-amp based amplifiers, signal-conditioning filters, all of which bring the signal to a level which can be interpreted by a microcontroller. The microcontroller makes the call on whether the

signal exists or not, and manages feeding that data back into AI. The micro can also be reconfigured to measure pulse width of the incoming signal, and convert that pulse width into sensible data.

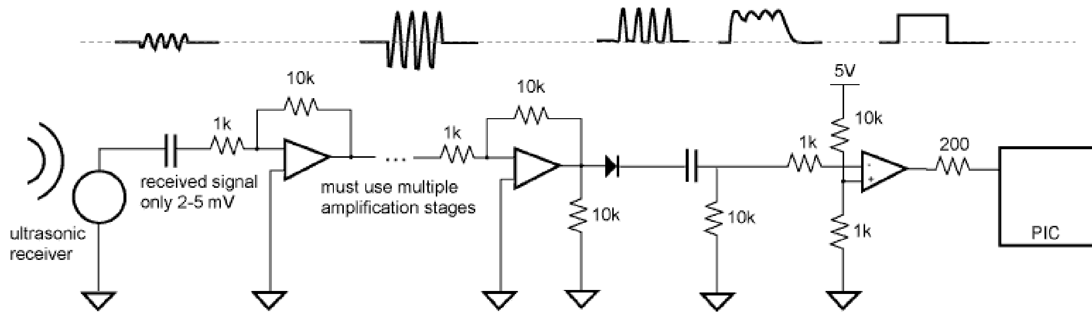


Figure 5.6.2.1: Ultrasonic Receiver Circuit

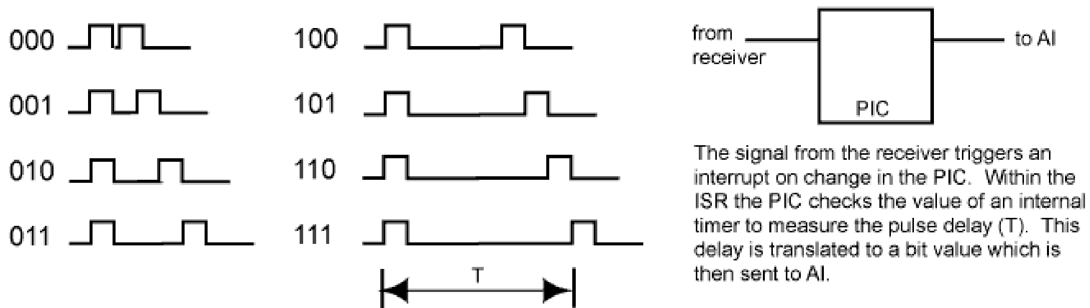


Figure 5.6.2.2: Data Encoding with Ultrasonic pulses

6. Analog System

6.1. Batteries

6.1.1. Overview

Changes to the robot this year may demand more performance from the batteries than in previous years. The most obvious change is that the robots will be driving four wheels instead of three. Because of this, we must re-evaluate our power supply system to make sure it meets our requirements.

6.1.2. Battery Selection

6.1.2.1. Current Estimate

To select the appropriate battery packs, we have to consider the current and the power the robots consume. With the exception of the kicker, in schematic and layout, the driving motors, the horizontal dribbler, the vertical dribblers, and the 5V regulator have their input voltages tied at the same potential.

At startup, each driving motor requires 5.4 Amps, and the 5V regulator that supports the digital and analog circuitry requires approximately 0.3 Amp input. In addition, assuming that the horizontal dribbler will also be turned on at startup, the stall current for the horizontal motor is 5.2 Amp, for the vertical motor is 3.3Amp. Assuming that the kicker is only used in spurts and not a factor in initial power consumption, the battery needs to give supply a total of 33.64 Amp at the input voltage at startup. Table 6.1.2.1 gives a break down of the current requirement when the robots start up and when it is at normal playing mode. Note that these numbers are calculated under the assumption that the robots are running with 13V power supply. At normal operation the kicker and dribblers are also assumed to be off. Please see the section 6.1.2.2 for more detail on these assumptions. The total start up current might seem large. This is because we have assumed worst case for all the components that might be on at start up, while in real life it might not be the case.

Device	Start Up(Am)	Normal Operation(Am)
Motor (x4)	5.4	.25
Horizontal Dribbler	5.2	0
Vertical Dribbler (x2)	3.3	0
5V Regulator	.14	.14
Misc. (IR sensor etc)	0.1	.1
Total	33.64	1.24

Table 6.1.2.1: Calculated Current at Start Up and Normal Operation

6.1.2.2. Power Consumption

In order to estimate how long the batteries will last, we have estimated the power consumption of the robot in Table 6.1.2.2.

Device		Units	Resist (Ohm)	Op. Volt (Volt)	Current (A)	Power per unit(W)	
						Max	Avg
Motor	Max (5%)	4	2.4	13	5.4	70	6.6
	Norm (90%)	4		13	.25		
Kicker		1	1.52	13	8.6	112	.37
Horizontal Dribbler		1	2.5	13	5.2	67.6	6.7
Vertical Dribbler		2	2.4	8	3.3	26.4	2.64

5 Volt Regulator	1		13	.14	1.82	1.82
Other Misc	N/A		13	.1	1.3	1.3
Total Power					515	39.33

Table 6.1.2.2: Power Consumption By the Robot

Max. Power	=	70 x 4 + 112 + 67.6 + 26.4 x 2 + 1.82 + 1.3
	=	515.52 W
Avg. Power	=	7.7 x 4 + .37 + 6.7 + 2.64 x 2 + 1.82 + 1.3
	=	42.33 W
Avg. Current	=	Motor Cur. + Kicker Cur + H_Drib Cur. + V_Drib Cur + Misc
	=	4 x (5.4 x 10% + .25 x 90%) + (.373 / 1.52) + (6.7 / 2.5) + 2 x (2.64 / 2.4) + .14 + .1
	=	4 x (.27 + .24) + .245 + 2.68 + 2 x 1.1 + .24
	=	7.405 A

The above calculation is based on the following assumptions:

Motor:
5% of the time, the motor is running at maximum power (i.e. stalling, cold start)
95% of the time, the motor is running smoothly and only consuming 250mA

Kicker:
40 Kicks a game (20 minutes)
100ms per kick
Avg Power = (0.1 sec * 112W * 40)/ (20 x 60) = 0.373 W

Dribblers:
10% of on when playing

5V Voltage Regulator
Outputs .3Amp to the digital board
85% efficiency

The above maximum power 515W is considered to be the absolute maximum power that can be consumed by the robot.
On average, the robot will only consume 39.33W with current 7.4 A.

6.1.2.3. Final Battery Selection

Last year, the team used Panasonic HHR210A NiMH for their design as the same as the year before. During the first semester, we manufactured more 2001 robots in preparation for an 11 on 11 robot demo. When we order more batteries from Panasonic for the 11-robot demo, we also ask for advice in batteries selection. The manager at Panasonic advised that the batteries we used in previous years are not designed to handle the high

current drain in motor driven applications. He also suggested 4/5 SC (HHR200SCP) or SC (HHR-300SCP) size NiMH cells would be better suited for our robots. We research into the two models suggested. We eventually decided to use the HHR-300SCP because it offers 3050mAH capacity, which is 50% more capacity than the HHHR-200SCP's 2000mAH. Our initial configuration is to put 9 cells in series as in Figure 6.1.2, note that the center packs are different configuration because they are behind the kicker.

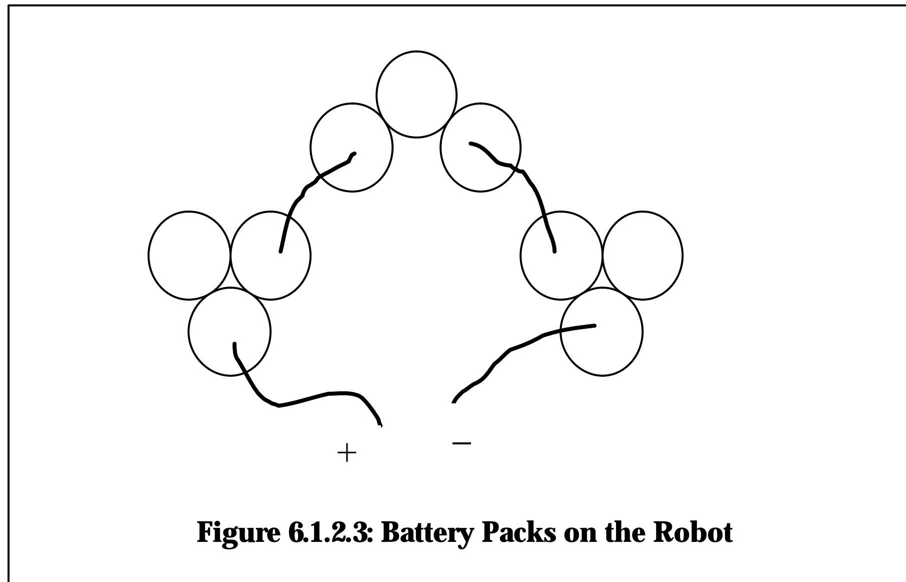


Figure 6.1.2.3: Battery Packs on the Robot

We used the above configuration in our first robot. The result shows that the robots are running smoothly with the battery packs and they fit perfectly into the mechanical system.

Another concern in selecting batteries is whether our current battery chargers could recharge them. In the lab there are eleven Maha MH-C777 designed to charge NiCad or NiMH battery packs with the voltage ranging from 4.8 to 12 Volts. They worked fine. Our final decision on batteries is to use the following model with the configuration in figure 6.1.2.

Panasonic Nickel Metal Hydride
HHR-300SCP
BATTERY NIMH 3000 MAH 1.2V

One concern is how long the battery packs will last. Here we will provide a rough estimate. After fully charged, the battery packs can provide up to above 12V. Given the average power consumption (39.33W) listed in Table 6.1.2.2, and the average current 7.4A, we expect the battery packs to run for more than 20 minutes until it drops to its nominal voltage (10V). The estimate is based on the discharge graph shown at Figure 6.1.3.3. When discharging with 10A, it takes 7 minutes for the voltage to drop to 1.1V. We will pay attention to how long the batteries actually run during the practice games before the competition.

6.1.3. Battery Specifications and characteristics:

	Diameter	23mm
	Height	43mm
	Approximate Weight	57g
	Nominal Voltage	1.2V
	Average Discharge Capacity	3050mAh
	Rated(Min.) Discharge Capacity	2800mAh
	Internal Impedance	4mOhm
	Standard Charge	300mAx16hrs.
	Rapid Charge	3000mAx1.2hrs.
Ambient Temperature	Standard Charge	0°C to 45°C
	Rapid Charge	10°C to 40°C
	Discharge	-10°C to 65°C
	Storage <1 year	-20°C to 35°C
	Storage <3 months	-20°C to 45°C

Table 6.1.3: Specifications of the HHR-300SCP

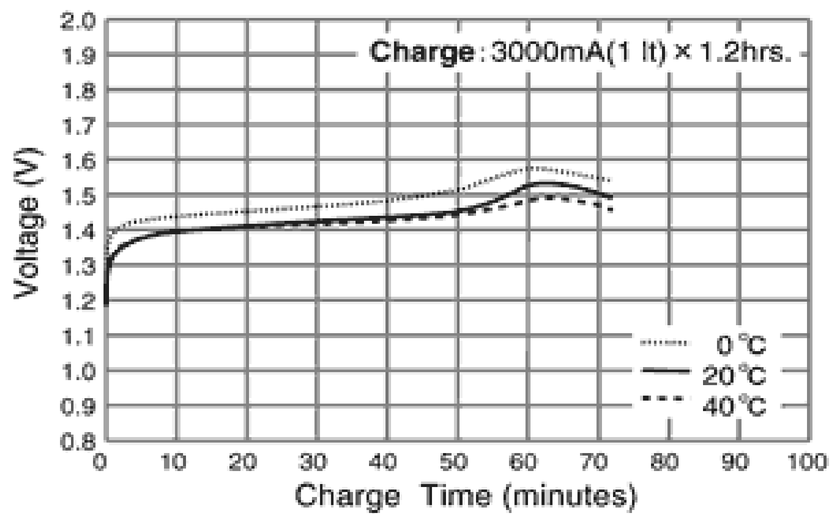


Figure 6.1.3.1: Typical Charge Characteristics

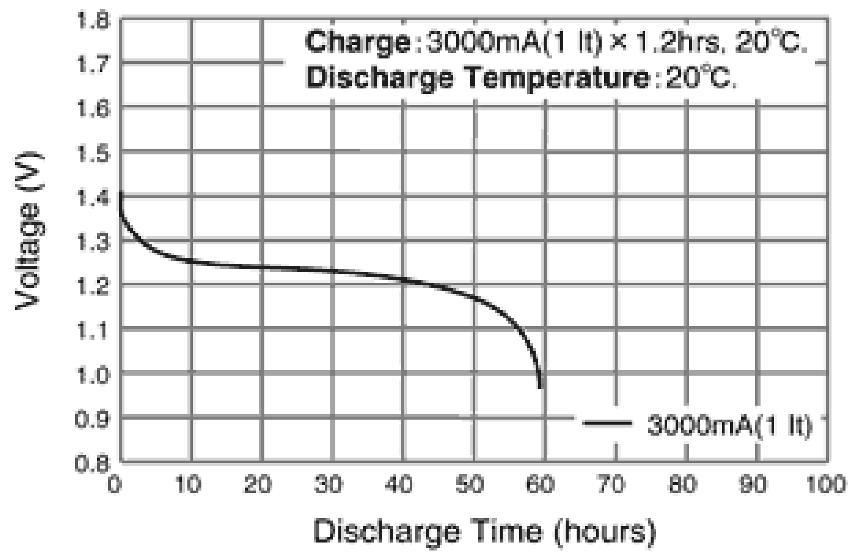


Figure 6.1.3.2: Typical Charge Characteristics at Low Current

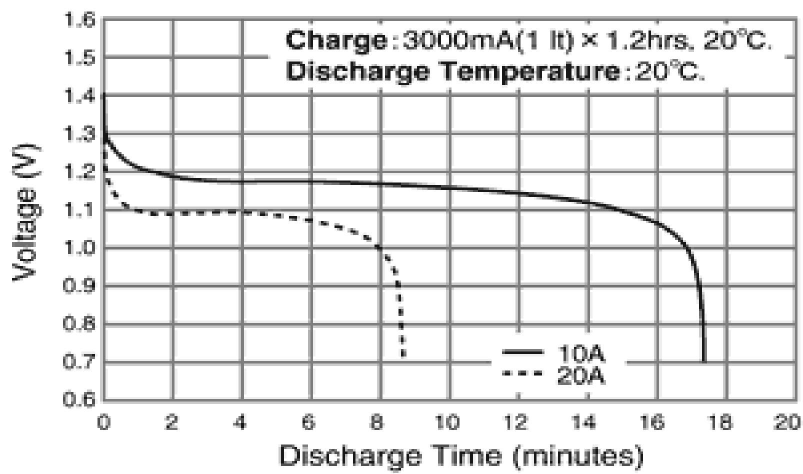


Figure 6.1.3.3: Typical Charge Characteristics at High Current

6.1.4. Fuses and Voltage Indicator:

To protect the circuits, we use fuses on both the motherboard and the digital board. On the digital board, we use Fairchild Mosfet NPD5070L. This is a 2A self-reset fuse which will reset itself when the robot resets. On the motherboard, we used F992-ND 15A fuse.

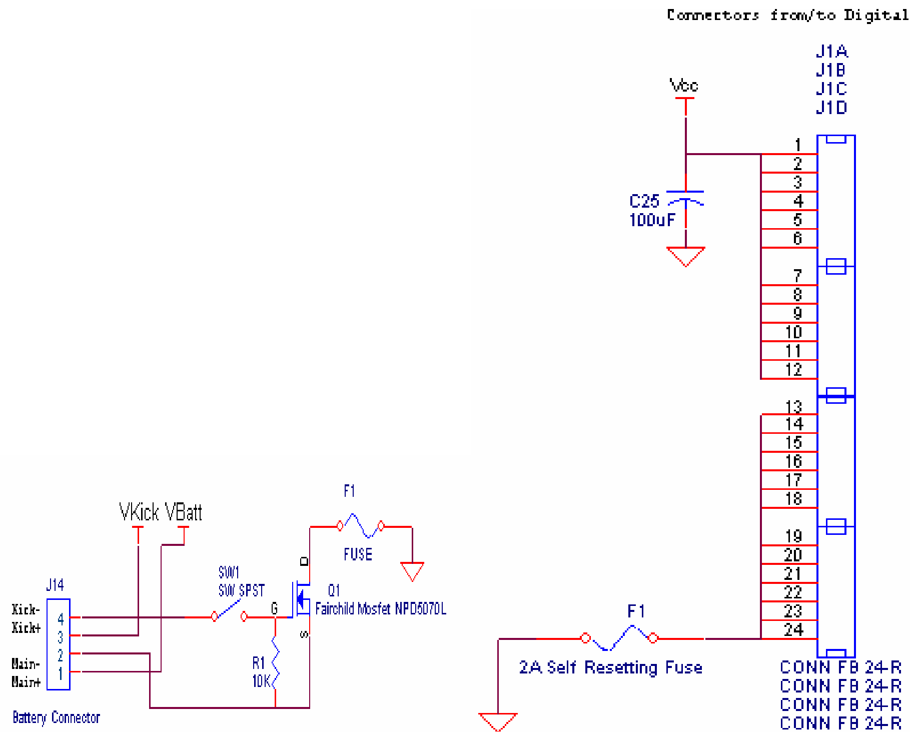


Figure 6.1.3.3: Fuses in the Schematic

The voltage indicator is essentially a voltage divider. When the voltage supply drops below certain value, the voltage indicator will trigger the beeper to give sound, warning batteries low. In our case, the indicator will trigger a beep when power drops under about 10.5V. The threshold voltage could also be modified easily in software. Introducing the voltage indicator enables the robots to report problems instead of us trying to look for it.

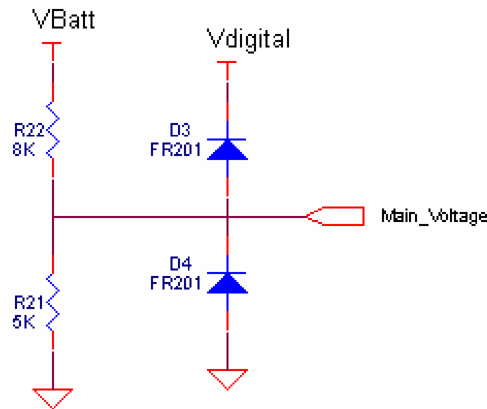


Figure 6.1.3.3: Voltage Indicator Circuit

6.1.5. Kicker Battery

To get best performance for the kicker, we should use battery packs with least internal resistance. This year, the mechanical engineering team picked the Sanyo Nickel Cadmium Battery because of its low internal resistance and small size. The model number is N-250AAA. The 6 batteries are connected in series to provide 7.2V nominal voltage to the kicker for kicks in different speeds.

6.2. 5-Volt Integrated Switching Voltage Regulator

In 2000 and 2001, the same basic 5 volt regulator circuitry was employed to generate the 5 V supply required for the digital electronics and various outputs and signals, despite a change in the main battery pack voltage from a nominal 9.6V to 12V. This consisted of a step-down switching regulator circuit built around a National Semiconductor LM2675 Switching Regulator Controller Chip, using the design recommended in the data sheet for this chip.

In the course of maintaining old and constructing new units of the 2001 design robots, the EE team discovered several undocumented irregularities in the regulator circuit. Firstly, several of the old 2001 robots had single or multiple inductors in their circuit, with a variety of resultant inductances, typically less than that specified in the schematic (33µH). Rumor had been circulating from veteran team members of some overheating and loss of regulation with the 2001 design, and it seemed that during the summer, after the documentation deadline, some experiments and modifications had been made to the circuit. When tested in breadboard (see appended document), the overheating problem could not be replicated. However, given rumor of instability, coupled with the undocumented changes, and the fact that the regulator circuit design was based on the 2000 year nominal supply of 9.6V, it was decided to review the circuit design and

external component selection. Typical for these types of chips from reputable manufacturers, the LM2675 data sheet includes a comprehensive design guide and recommended circuits.

For a 12V nominal input voltage and about 0.5 A output , a 47µH or 68µH inductor is recommended. This was tested satisfactorily in breadboard and was populated on the new 2001 design boards. We also observed some instability on some of the new 2001 boards. Although it was difficult to reproduce consistently, the regulator output would sometimes droop to 2-3 V when a drive motor on the robot was loaded or stalled. Monitoring the input voltage to the regulator did not reveal any problem, as input voltage never fell below 7 V, even briefly. Despite tracing the circuit, and a full week devoted to debugging, the cause of this behavior was not identified. It is most likely related to a design error since the routing software and the outsourced manufacturing of boards are unlikely sources of error.

After this nightmare, and the realization that switching regulator design is not trivial, it was suggested that a more fully self-contained, COTS (commercial off the shelf) regulator be investigated to resolve the regulation issue on the 2001 boards. A team member had had some experience with appropriate products marketed by PowerTrends, now Texas Instruments Plug-in Power Solutions. A web search of TI's plug-in power website led to the selection of the 78ST205VC, 2A, 5V integrated switching regulator (ISR). This is a simple 3 pin device (input, ground, and output), about 1" tall by 1" wide, by 0.25" thick, including heat sink and inductor, with one external filter capacitor required. Please see the datasheet in the appendix for details. These were retrofit into all of the new, and several of the old 2001 design boards, bypassing the existing circuit, since when there have been no problems with regulation. The tradeoff is a slight reduction in efficiency, from a nominal 90% to about 85%, probably due to the use of a tiny (given the 2A output) surface-mount inductor on the ISR, vs. the substantial through-hole inductors used in the previous circuit.

The solid performance of these products led to the decision to integrate them into the 2002 design.

A brief estimate of the current demand that the 5 V supplied components (directly and indirectly) of the 2002 design follows, and justified the use of the 2A version of the component as opposed to the 3A. Note that the following is based upon datasheet nominal values and schematic calculations.

Component	Description	Quantity	Current/unit	Total
PIC16F877	Microcontroller	4	15mA	60mA
EPM7128S	FPGA	1	50mA	50mA

MAX203	RS232 Driver	1	8mA	8mA
LM555	Timer	1	3mA	3mA
OPA1013	OpAmp	2	negligible	negligible
ECS200	Clock	1	25mA	25mA
LCD0821	LCD	1	50mA	50mA
MM74HC132	Schmitt NAND	1	negligible	negligible
MM74HC126	3-State Buffer	1	negligible	negligible
EFR-TQB40K5	Ultrasonic Tx	1	7mA	7mA
Radiometrix RPC	Wireless Module	1	34mA	34mA
SML-LX1206	LED Indicator	5	23mA	115mA
			TOTAL	352mA

This is in reasonable agreement with a measurement made by patching an ammeter in series with the 78ST205VC on a set of prototype 2002 boards, which came to 250mA for a fully operating board, including interprocessor serial communications and driving the digital inputs of several analog power devices, but lacking the LCD. Including the estimated LCD requirement, this would come to 300mA. Interestingly, the calculated value in the table above turns out to be a moderate overestimate - the datasheet specifications of several of the components were generously estimated.

According to the datasheet for the 78ST205VC, we can expect ~83% conversion efficiency from the component at 12V input, and ~300mA output.

The following schematic depicts the relevant circuit, on the 2002 analog board design.

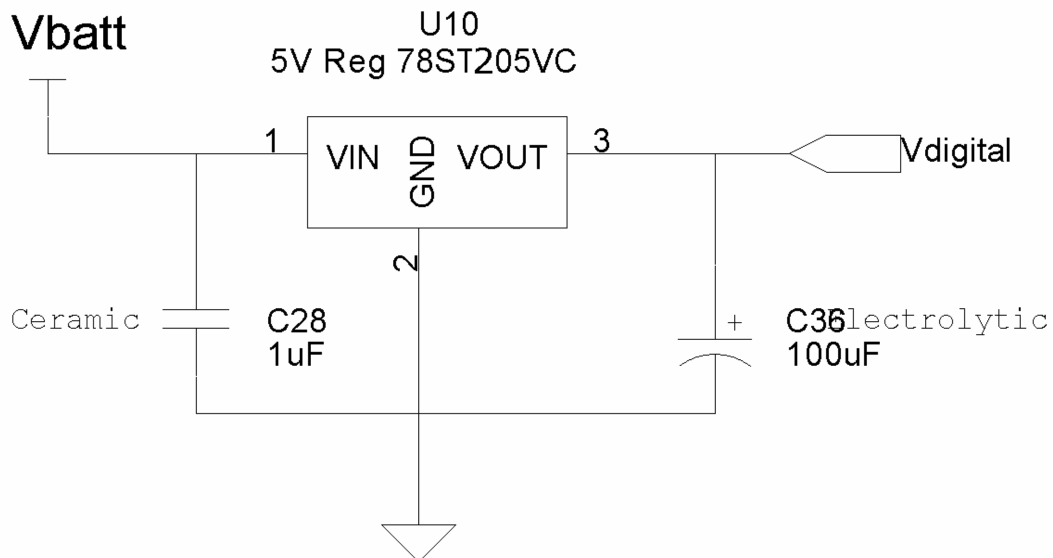


Figure 6.2: Voltage regulator circuit

The ceramic input filter capacitor is considered optional by the manufacturer, but was included to help guarantee clean power. Both the ceramic and the electrolytic capacitors were specified to tolerate 50V because the presence of rapidly switched 12V signals on the analog board might lea d to substantial spikes at the input.

The physical placement of the parts on the 2002 analog board - see the photos of the boards in the Appendix - is in retrospect, less than optimal, although its layout position was selected with the intention of minimizing trace lengths to reduce noise on the 5V supply. The pins on the 78ST205VC are rather brittle, and there are no structural mounting pins. We located the part too close to the edge of the board, and it is in jeopardy of being bumped by anyone removing/inserting the motor1 connector, or the dribbler connectors from the mother board. We have yet to subject the boards to the full rigors of daily use, but we are hoping that the emphasis on the overall maintainability, reliability, and modularity of the robot will minimize the need remove these connectors.

6.3. Kicker

6.3.1. Variable Speed Kicking

This is a very simple system. The kicker is activated using a simple MOSFET switch, which, when activated, provides the solenoid (being held at V_{kick}) a path to the ground. The voltage across the solenoid is equal to the battery voltage plus a nominal 7.2V from a six cell Sanyo KR-N-250 AAA NiCad external battery pack (a total of at least 18 V). They have 250mA-h capacity. The strength of the kick is controlled by limiting how much time the switch is open with typical values being on the order of 50ms. In the main microcontroller code this is done by setting a timer when the switch is first activated, and closing the switch when the timer interrupts on overflow.

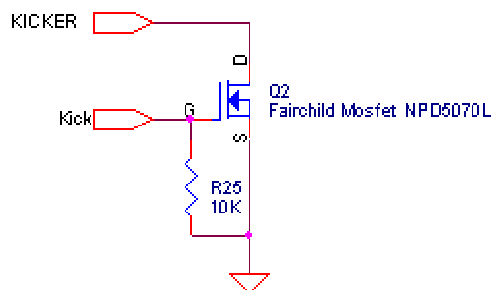


Figure 6.3.1: Kicker Circuit

6.3.2. Testing of Kicker

Note that when testing the kicker circuit, batteries must be used to power the robot. A benchtop supply cannot source a sufficient current pulse for the circuit

to operate. The result will be the appearance that no power is being delivered to the kicker.

6.4. IR Circuit

6.4.1. Overview

In 2001 the IR system would sometimes malfunction in the presence of bright light. Bright light prevented the system from detecting a broken beam because the receiver would capture enough ambient infrared lux to register as "unbroken". The primary goal of the 2002 design was to address this problem.

This year's IR detector is designed to be less sensitive to misalignment and bright overhead lights. It does this by sending a pulsed signal to its receiver. The received signal goes through a high-pass filter to remove DC before further processing. The DC filtering allows the overall room brightness to be subtracted out, leaving the pulsed signal to determine whether the beam is blocked.

6.4.2. IR Transmitter

The transmitter is a 555 timer chip running an infrared LED at 240 Hertz. The 555 is a standard timer/oscillator whose oscillating frequency can be set by a resistor capacitor pair. We ran ours at a low frequency of 240 Hz because the receiver's received power drops off at high frequency as does the amplification response of the OPA1013 op-amps (which do the signal preparation). To adjust the transmitted signal strength, there is a current limiting resistor on the output leg.

The infrared LED used this year is brighter and shines in a wider angular field than the one used in 2001. Both of these factors contribute to improved resistance to misalignment. We've also added the hardware necessary to support two IR sensors. This was done as a backup measure so more complex beam geometries could be supported. Preliminary test, however, seem to indicate that one beam will be effective. A special connector was made to link the IR receiver-In signals of both detectors because without a signal going into both the unconnected IR receiver will report that its beam is broken.

Another thing to note is that the total current consumption for two IR transmitters is around 100mA. This is a non-trivial current for the digital board to support, especially since the IR transmitters are always on. Despite this, it was calculated that the demand can be met by the digital board without adverse effects.

6.4.3. IR Receiver

The new design receives a pulsed IR signal instead of a steady beam. When the beam is picked up by the receiver the DC bias can be removed (this subtracts out ambient light

levels) and the pulsed AC component can be processed. The rest of the receiver circuit does simple signal processing to detect the strength of the AC component and generate a digital low if the beam is broken. The first element of the processing is an amplifier which makes the signal easier to work with. A passive filter averages out the amplified signal so that it is approximately flat and can be compared to a level of 0.5V in a comparator. With excellent filtering only one comparator would be needed, but in our current setup there are situations where one comparator does not give a steady digital output (sometimes the filtered signal will drop below the comparator threshold). This problem was addressed by adding another filtering capacitor and running the filtered output into a second comparator. This setup gives good results which are resistant to both ambient light and mechanical shock.

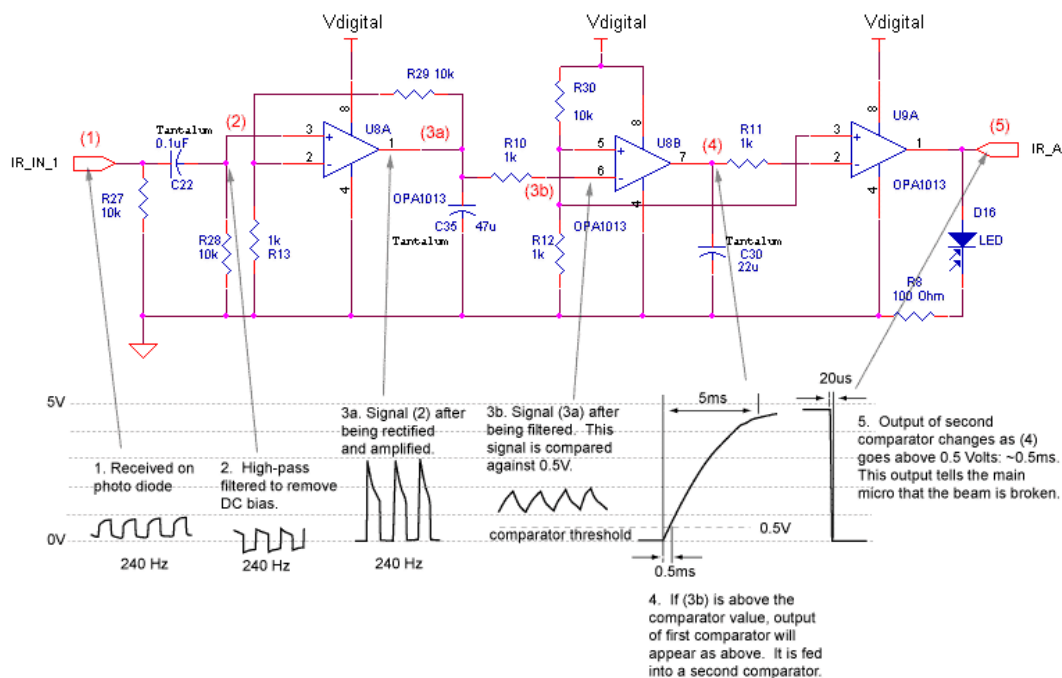


Figure 6.4.3: IR Receiver

6.4.4. Possible Improvements

In the future the system could be tuned to require two op-amps instead of three. This could be done by combining filtering into the amplification stage so that a single comparator could make the call. The circuit could also be improved by upping the frequency of the IR pulse (perhaps to 5kHz). Two factors currently limit this: the photo-pin diode (a.k.a. receiver), and the op-amp. Both are less responsive at higher frequencies (op-amp amplification reduced, received voltage over diode lowered) but with different op-amps we could raise the frequency and thereby reduce the capacitance (and size) of our filtering elements. This has the added benefit of slightly improving the response time (smaller filtering elements means faster responses to changes). If

implemented as described above, the IR system could be compacted to take less than half its current board area while maintaining the same functionality.

6.5. Dribbling System

Variable dribbling speeds can be helpful in improving ball control, especially when turning. If a ball is being spun quickly, its angular momentum can keep it from turning smoothly with the robot. If the robot turns suddenly the ball may be lost due to excessive spin on the ball. If the ball is dribbled slower just prior to turning, this problem can be largely avoided.

We implemented a variable speed dribbler by creating a current sensing circuit which tells the main micro how much current is being used to dribble the ball. Current is directly proportional to torque being applied to the ball and torque is the physical property we would like to control. Currently the dribbler has a primitive control. It has some discrete values of dribbling speed and switches from one of these values to the next whenever the current consumption gets beyond certain set limit. It speeds up the dribbler if the current is too low, and slows down if it is too high. In our final version it will be a simple p-controller with a more continuous set of speeds values for a more accurate control. This will be implemented in summer of 2002.

The current sensing circuit consists of a low value resistor (0.5 Ohm), over which the voltage is measured and converted to a signal that is readable for the microcontroller A/D (0-5V).

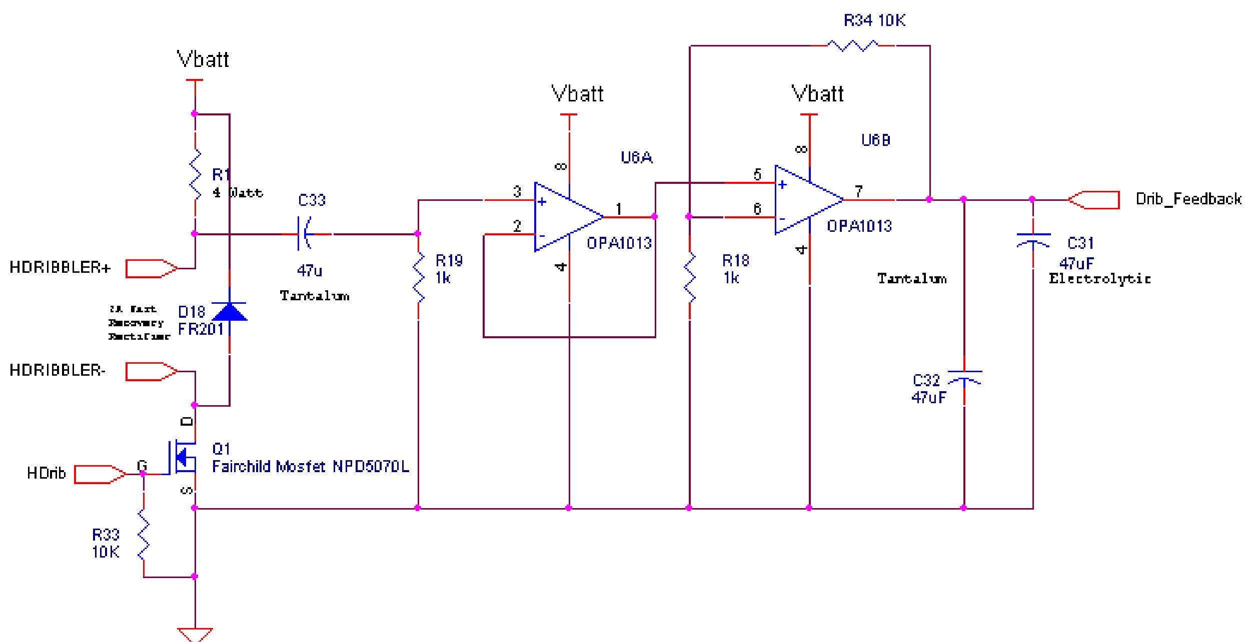


Figure 6.5: Horizontal Dribbler circuit

This circuit converts the current (AC) across the resistor R1 as a DC voltage within a usable range for the microcontroller to read using one of its Analog-to-Digital converter ports. The microcontroller uses this value to control the speed of the dribbler motor. The microcontroller, in turn, generates a PWM signal with adjusted duty cycle on HDrib (gate of Q1) to change the current consumption (or torque of the motor).

C33 is used as a high pass filter to isolate any DC offset from the motor. The first Op-Amp is a unity gain feedback stage to provide electrical isolation between the input and the output stages. The second Op-Amp provides the gain to convert the voltage values to ones that are in the readable range of the microcontroller (0-5V). C32 and C31 are used to further filter the signal and remove any ripple in it (i.e., low pass filtering). When the dribbler is run with a PWM at 1kHz, the output signal appears well filtered and its level well correlated with dribbling speed.

6.6. H-Bridges

While the PWM signals from the motion micros determine how hard the motors should be driven, it is the H-bridges which actually drive them. We used the L6203, which can handle up to 5A and in testing demonstrated a lower transistor turn-on voltage. This is important because the lower the turn-on voltage across the H-bridge, the more voltage the motor actually sees, and consequently the easier it is to accelerate. This drop can be measured by probing the voltage over the H-bridge while the motor is being stalled (held still while power is being applied). Performing this test we found the 2001 H-bridges yielded a drop of 3V, (a significant fraction of the nominal 10.8V supply), while the L6203's by comparison yielded a 1.8V drop (a 15% improvement over 2001's H-bridges).

Our implementation of the H-bridge is a slight adaptation of the implementation discussed on page 12 of the L6203 datasheet. The salient features are as follows:

- Pins 1 and 3 are the output terminals. The voltage drop over them is what supplies power to the motor. They each have a bootstrapping capacitor attached.
- Between pin 1 and 3 there is also a 15 Ohm resistor/capacitor pair. These components greatly reduced transient noise spikes. The diodes also helps reduce the spikes.
- Pins 5 and 7 determine the direction of the motor. They take in digital signals and 7 should be the negation of 5. In our design we used 2 pins from the motion micros to supply DIR and ~DIR. This was chosen over the option of using one pin and a NOT gate because conserving board space was more important than saving microcontroller pins.
- Pin 11 is the PWM input. The duty cycle determines how hard the motor will be driven

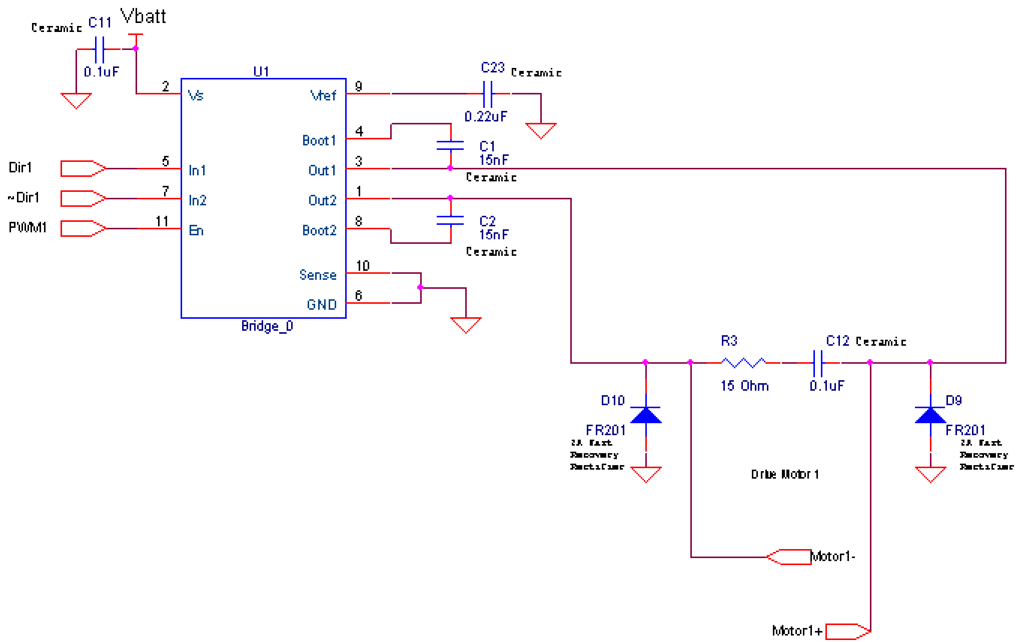


Figure 6.6 H-bridge Circuit

Formal tests were not run on the thermal characteristics of the H-bridges. They do however get significantly hot if the motors are stalled for more than 10 seconds. Stress testing may reveal whether hot H-bridges lead to any adverse effects. At this point, however, this testing is still in the pipe.

7. Wireless System

7.1. Overview

One of the major changes made by the EE team was the redesign of the wireless communications system between the intelligence and control system on the sideline and the robots on the field. The design requirements of this system were two fold.

Our primary goal was to improve the robustness of the wireless system so that we would not be plagued with the intermittent problems that former years' wireless systems faced during competition. We satisfied this requirement in several ways. First, instead of relying on only one type of wireless module, our system now supports three different types of modules operating on different carrier frequencies: 418 MHz, 433 MHz, 869 MHz, and 915 MHz bands. Thus if a certain module is not working well at competition, we merely have to swap a different module onto the robot. Secondly, while we are still offering backwards compatibility with the Radiometrix RPC used in previous years, we are moving to the Radiometrix TX/RX modules that allow us increased controllability. This control allows us to implement methods of Error Detection and Forward Error Correction onto the wireless channel.

Our secondary goal was to move from a simplex system where only the intelligence and control system could talk to the robots, to a full-duplex wireless system where information could flow in both directions. This means that the robots can now send local information back to the intelligence and control system on the PC. The 2002 Robots will feature ultrasonic transmitters which offer the ability to send one bit of information, back to the intelligence and control system indicating when that particular robot has the ball.

Our team has also researched and designed a multi-frequency, high bandwidth, low-latency wireless return path using the SE200 modules. This design is ready to be implemented onto next year's robots so that we can send more than one bit of local information back to the intelligence and control system. For example we can send information regarding the BER rate of the forward wireless channel so that we know if we need to change to a different wireless module, battery voltage status so that we can tell which robots need to have their batteries recharged, status on how the various subsystems of the robot are performing, or local position information via an optical position tracking system.

In Section 7.2 we detail all of the different Wireless Modules that we researched for use on the 2002 RoboCup System. We detail the specifications, advantages, disadvantages, current status, and application for use of that particular module as it applies to the RoboCup System. We also include corporate references and references to documentation where applicable. Then in Section 7.7 we detail our wireless test setup

and tests that were performed to evaluate and compare these modules. Section 7.3 details the results of our tests as they apply to the 2002 RoboCup system.

7.2. Requirements

The requirements for this year' s wireless system were relatively straightforward. These requirements were decomposed a little bit further to give us more specific requirements.

R1. The system shall be more robust than last year and more fault tolerant.
- R1.A The system shall be able to use multiple frequencies.
- R1.B The system shall be able to handle the overflow of the receiver.
- R1.C The system shall be more immune to interference than last year's system.

R2. The system shall be compatible with last year's system.
- R2.A The system shall have a latency no greater than ~16ms.
- R2.B The system shall be able to use the same packet structure as 2001.

R3. The system shall have a return communication path for extremely low bandwidth communication (1 bit of information for a robot to acknowledge if it has possession of the ball or not).

(Note: Decomposed requirements are listed beneath the major headings.)

7.3. Design to Meet Requirements

2001 System

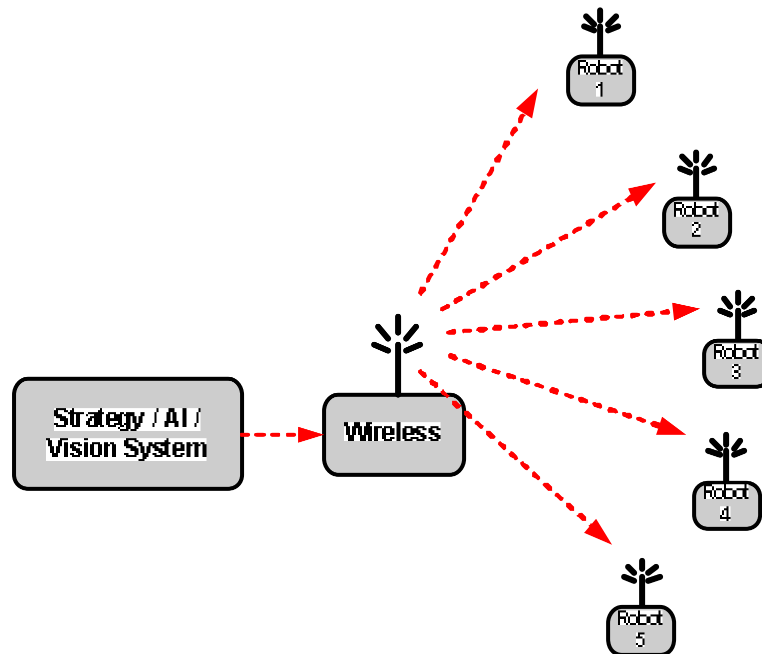


Figure 7.3.1: 2001 Simplex Wireless System

The 2001 wireless system used the Radio Packet Controller (RPC) as its only available wireless module with two available frequency ranges. The system performed broadcast communications from the artificial intelligence computer to the robots.

2002 Communications System Design

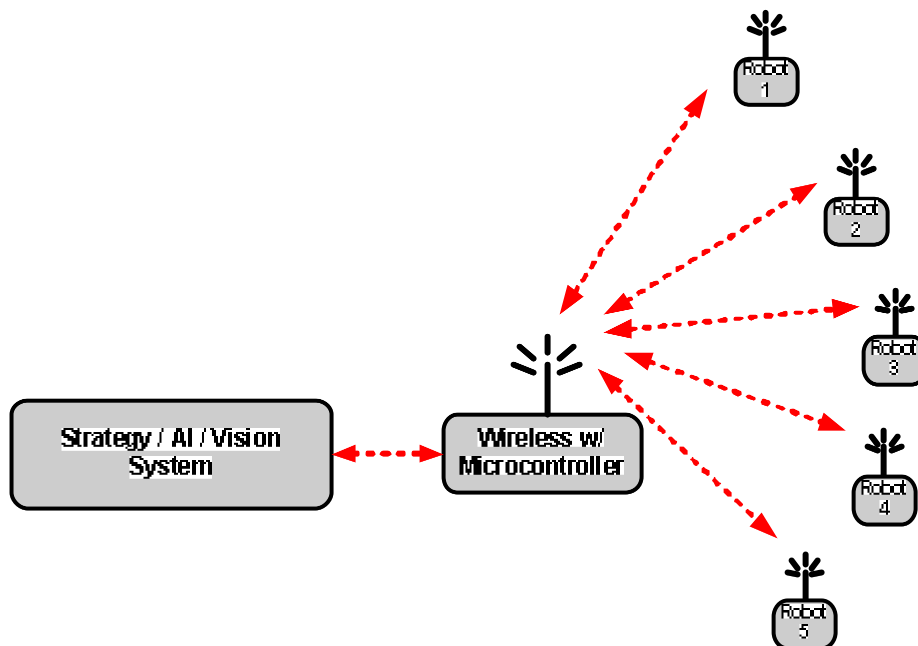


Figure 7.3.2 2002 Full Duplex System

The main difference between the 2001 and 2002 design is that we insert a microcontroller between the communication paths on the two end-to-end communications. The advantages of including this microcontroller into new design are:

(1) The AI system sends a packet of data every 16.6ms. Instead having a processor on the PC spend all of its time negotiating with the wireless module, it can instead send the packet to the microcontroller who will negotiate the transfer with the wireless module, thus freeing up the PC processor.

(2) Different types of wireless modules can be interchanged into the system, while remaining transparent to the AI system. Thus if we need to change the wireless module during competition due to interference or malfunction, these settings need only be configured by a DIP switch at the microcontroller rather than reconfiguring code on the AI system

Our 2002 requirements led to the formation of our preliminary design. Because of space limitations, multiple wireless modules could not be feasibly mounted on the robot simultaneously. Therefore, the concept of a single wireless module board that would adapt a number of modules onto the wireless board was conceptualized. The different modules could cover the different frequency ranges and possibly different communication schemes. The system would have not only a broadcast to the robots, but also a return path to the artificial intelligence computer to convey information such as ball possession. Overflow of the receiver could be handled in software on the wireless

microcontroller's control software. The same packet structure implies that 26 bytes of information need to be sent in 16.6ms creating a minimum data rate of 15,700bps (26 bytes * 10 bits/byte/16.6ms). We note that we need to count 10 bits per byte since there is a start and stop bit associated with each 8-bit byte of data. Lastly, immunity to interference can be increased by layout of the module and antenna choice.

7.4. Preliminary Research

7.4.1. Technologies Considered

Numerous technologies were considered on the path to settling on a couple of feasible solutions. A listing of the technologies considered follows.

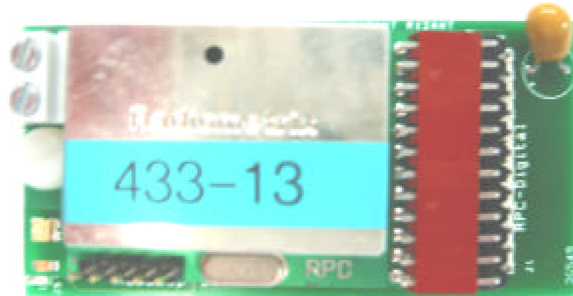
- Radio frequency
  - 418 MHz / 433 MHz
  - 869 MHz / 914 MHz
- Ultrasonic
- Spread Spectrum
  - 802.11b – Note: Since this technology was not used in the final 2002 Robot Design, due to time constraints we will be adding documentation over the summer.
  - FHSS (Frequency Hopping Spread Spectrum)
- Infra-red – Note: Since this technology was not used in the final 2002 Robot Design, due to time constraints we will be adding documentation over the summer.

7.4.2. Wireless Modules Considered

Numerous wireless modules were researched in our quest to find the best modules for our application. We performed a great deal of research on the following modules.

- Radiometrix
  - TX2 / RX2
  - TX3 / RX3
  - RPC
- Wireless Mountain
  - Unilink
- Aerocomm
  - AC1524C-3
- Conrad
  - SE200

7.4.2.1. Radio Packet Controllers



Specifications

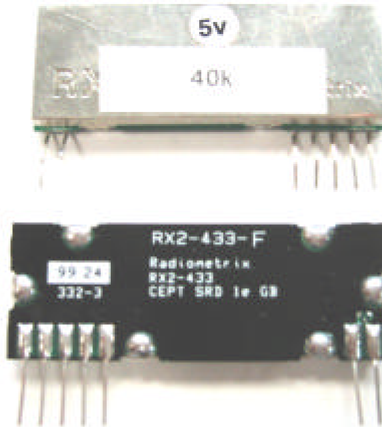
RPC Module	
Module Type Carrier Frequency	Transceiver 418 MHz, 433 MHz
Data Rate	64 kbps
Setup Time	~5ms
Manufacturer	Radiometrix, UK tech_support@radiometrix.com www.radiometrix.com
Distributor	Attn: Danny Lemos Lemos International 1305 Post Road Suite 108 Fairfield, CT 06430 1 (866) 345-3667 sales@lemosint.com www.lemosint.com
Interface	parallel (1-byte wide)
Data Rates	40 kbps half-duplex
Frequency Band	418 MHz / 433 MHz
Performance	30m in-building range / 120m open ground
Dimensions\Weight\Volume	54 x 32 x 16
Power	5V @ <20mA
TX Latency	13.8ms

Advantages / Disadvantages

The advantage of the RPC is that it is a totally integrated unit with a PIC microcontroller on board. However, it takes in parallel data which requires a lot of I/O ports on our

microcontroller. With its large setup time, timing can become an issue with an increase of data. Previous years' RoboCup teams have also experienced interference issues using this module.

7.4.2.2. TX2/RX2



Specifications

TX2/RX2 Modules	
Module Type	Transmitter/Receiver
Carrier Frequency	418 MHz, 433 MHz
Data Rate	160 kbps
Setup Time	~ 1 ms
Manufacturer	Radiometrix, UK tech_support@radiometrix.com www.radiometrix.com
Distributor	Attn: Danny Lemos Lemos International 1305 Post Road Suite 108 Fairfield, CT 06430 1 (866) 345-3667 sales@lemosint.com www.lemosint.com

Advantages / Disadvantages

The TX2/RX2 modules are separate transmitter and receiver modules with a carrier frequency of 418 MHz and 433 MHz. The module is extremely small in size and in contrast to the RPC module takes in digital data serially. A separate microcontroller is needed to format the data and balance the packets in order to send them through the

module. However, this also means that we gain more control over encoding and decoding schemes as well as rate of data transfer.

Notes

The RX2 module used a carrier detect signal to internally detect when to capture the incoming signal. A 22 µF and a 0.033 µF capacitor were used to filter the incoming signal to enable a more stable output. Manchester encoding was used to guarantee that an equal number of ones and zeros are transmitted over a period of several milliseconds. A 50% bandwidth overhead goes into the Manchester encoding, but the TX2/RX2 module's maximum data rate is well above twice our required data rate; we will also be exploring the viability of alternate encoding schemes over the summer.

7.4.2.3. TX3/RX3

Specifications

TX3/RX3 Modules	
Module Type	Transmitter/Receiver
Carrier Frequency	869 MHz, 914 MHz
Data Rate	64 kbps
Setup Time	~ 1 ms
Manufacturer	Radiometrix, UK tech_support@radiometrix.com www.radiometrix.com
Distributor	Attn: Danny Lemos Lemos International 1305 Post Road Suite 108 Fairfield, CT 06430 1 (866) 345-3667 sales@lemosint.com

	www.lemosint.com

Advantages / Disadvantages

The advantage of using this module is that it operates in an alternate frequency range than the 418/433 MHz used by most other RoboCup teams in competition. Like the TX2/RX2 the modules are extremely small in size and in contrast to the RPC module take in digital data serially. A separate microcontroller is needed to format the data and balance the packets in order to send them through the module. However, this also means that we gain more control over encoding and decoding schemes as well as rate of data transfer.

Notes

The module takes voltages in the range 0-3 V which is different from the RX2 and RPC from Radiometrix. The data sheet recommends placing a 100 kΩ resistor on the line to bring the voltage down from 5 V to the 3 V range. A 22 μF capacitor was used to filter the input, enabling a more stable output.

Specifications	
Interface	Serial
Data Rates Frequency Band	50kbps 914 MHz
Performance	30m in-building / 120m open ground
Dimensions\Weight\Volume	12 x 32 x 3.8mm
Power	2.7V-12V @ 9.5mA

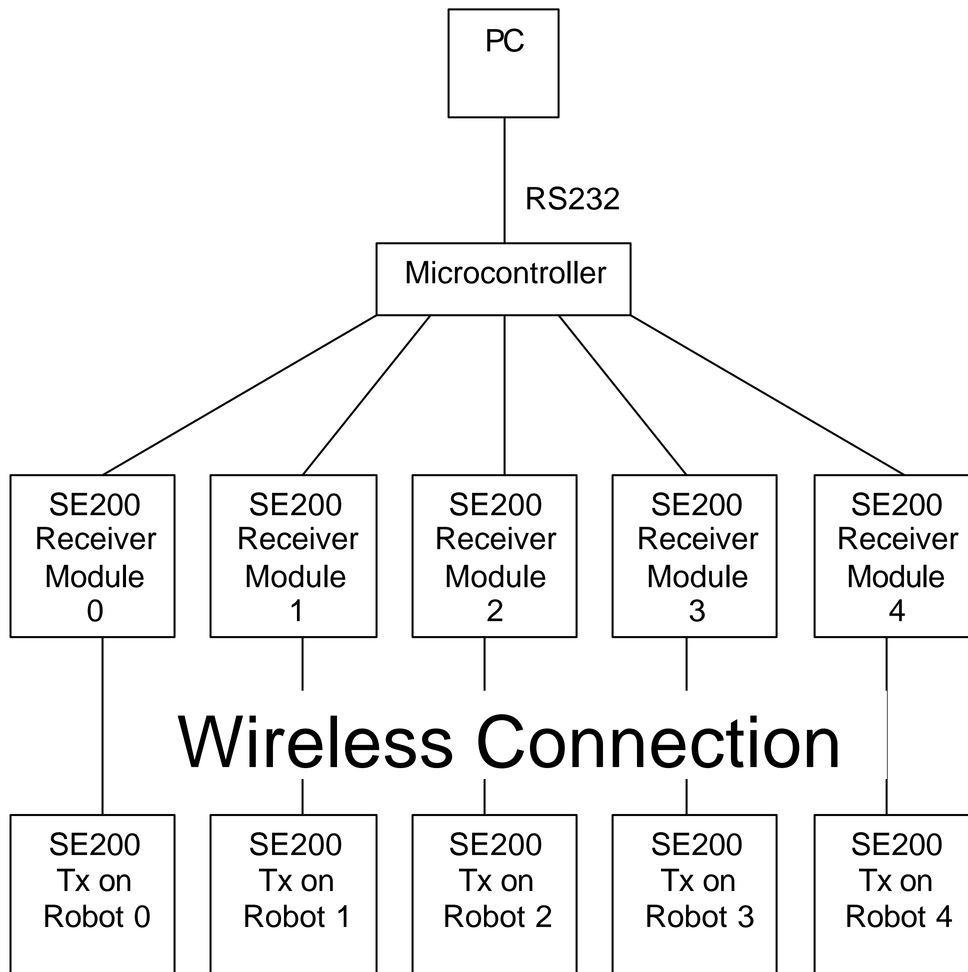
References:
http://www.radiometrix.com/html/products/tx3.html

7.4.2.4. SE200

Our testing and analysis has shown that the SE200 can be used to support a multi-frequency, high bandwidth, low latency return channel. A block diagram representation of this system appears below.

While this reverse channel system offers many advantages, it was not used on the 2002 Robots due preliminary concerns regard about the extra microcontroller on the robot that it required would not fit on the Digital Board using the Motherboard PCB board layout architecture that we had planned to use. Instead of moving to a different board layout or trying to adapt our preliminary design for the Motherboard layout to get all this to fit, the decision was made to forego this multi-frequency, high bandwidth, low latency return channel. Our work remains ready for implementation on the 2003 robots.

Note that over the summer we will be adding a great deal to this section: hardware schematics and explanations behind them, software to get the SE200's operational, and explanations and preliminary test results of the SE200, so that the research that we have already done can be readily picked up by next year's team. Unfortunately, due to time constraints we did not have time to add all of this information to this document.



7.4.2.5. **Aerocomm**

Specifications

Interface	20-pin mini connector
Serial Communications	Up to 115.2 Kbps
Frequency Band	2.402 – 2.478 GHz, FHSS
Performance	Up to 100ft (30m) inside, 1,000ft (305m) line of sight
RF Channels	10
Dimensions: Weight:	2.65" x 1.65" x 0.2" (6.7x4.2x0.5 cm) <.7 ounces (<20 gm)
Power	100mA (Tx); 80mA (RX) Consumption 3mW output power
Voltage	5V

Advantages

The main advantage of this product is that fact that it utilizes Frequency Hopping Spread Spectrum (FHSS) technology to handle interference problems with other RF devices. Additionally this product provides us with a high bandwidth communications systems with a serial communications interface.

Disadvantages

We note that the advantages of FHSS technology does come at the cost of increased overhead and latency to the system. After speaking with a technical representative for Aerocomm we were able to ascertain that while the normal specification of time to packetize and send data is about 16ms (very close to the 1/60s limitation of our system), the Aerocomm devices do allow us to define and program our own communications protocol within the onboard EEPROM. The technical representative said that by removing some of the features of the default protocol, we can get our latency down to the 5 – 7ms range, especially due to the fact that we only have about 27 bytes of data to send. There is a 1ms overheard due to frequency hopping, and we can define our own parameters for channel number, addressing mode, and error control. Since retransmission protocols do not help us with such a real time system, we can simply eliminate the error control overhead and we should be able to get the device to operate within the latency requirements of our system.

We would also like to note that last year's RoboCup team also tested a FHSS product from Aerocomm for the 2000-2001 RoboCup system. After reading through their Final Design Document, we note that the main problems dealt with the hardware and software setup of the system (page 62 – 64 of the 2001 Electrical Engineering Final Design Document). They also mention that the maximum bandwidth that they could get out of the system was 800bps. This number seems extremely low for a system that is supposed to run at a maximum of 115.2Kbps. While the means of testing as well as the associated calculations for this 800bps calculation are not given, we propose that they were measuring effective bandwidth after the all of the control overhead was introduced. If we were to use an Aerocomm device we would be defining our own communications protocol so that this overhead would be much lower for our system.

Furthermore we are using the 1524 family of Aerocomm devices which is their low-latency product, rather than the LX2400S-10 which was tested last year.

References

Company	Aerocomm
Product	AC1524C-3
Address	10981 Eicher Drive Lenexa, Kansas 66219
Phone	1.800.492.2320
Webpage	www.aerocomm.com

7.4.2.6. **Wireless Mountain - Unilink**

Specifications	
Interface	RS-232 (Serial)
Serial Communications	300 to 19.2KBPS
Frequency Band	902-928 MHz ISM Band
Performance	125 meters, up to 400 feet, line of sight
RF Channels	Factory configured, up to 254 user assignable addresses per channel
Dimensions: Weight: Volume:	1.3" x 2.0" x 0.5" (3.3x5.1x1.3 cm) 2.0 ounces (60 gm) 1.3 cubic inches (21cc)
Power	Powered via RS-232 line

*Advantages***:**

Being a self contained, "black box" unit with only a serial connector as an interface, which makes this unit very easy to use and set up. The associated PC software allows the user to configure the device settings such as addressing and error control by plugging the device into the PC. (Note that since we get new data every 1/60th of a second we do not have time for retransmission of data and hence it is our desire to decrease the error control functionality as much as possible in order to cut down on the latency of the system.)

Disadvantages:

We note that the RoboRoos team from Australia used the Wireless UniLink device for the 2001-2002 competition. It was noted by Professor D' Andrea and Michael Babish that they believed that the RoboRoos experienced some problems with interference when using this product. If necessary we will contact them after we get samples of this product and evaluate its performance.

References:

Company	Wireless Mountain
Product	UniLink
Address	560 Higuera Street San Luis Obispo, CA 93014
Phone	805.596.0960
Webpage	www.wirelessmountain.com
Contact Personal (conversational) Info	Matt Steenwyk Mechanical Engineer, graduated and got degree from USAF, likes to joke about MechE's vs. EE's.

7.5. Developing Functional Prototypes

After selecting our preliminary modules for consideration we next needed to figure out how to get the modules working and begin performing some preliminary tests to make sure that the modules would work with our system. Since previous years' teams had already been using the Radiometrix RPC, we knew that this module would suit our purposes. We thus focused on the TX2/RX2, TX3/RX3, and the SE200.

Since all three of these devices accepted a digital serial input on the transmitter end and output a digital serial output at the receiver end, we initially thought that it would not require too much effort to get the modules working: we could just connect the transmitter and receivers to the respective microcontroller USART connections. Unfortunately, while the modules are interfaced serially, there are quite a number of steps required both in hardware and software needed to get the modules operational. This process was made increasingly difficult by the fact that the manufacture's documentation and application notes were sparse and unclear. Through many hard hours of consultation with the manufacturer mixed with some trial and error, we were able to arrive at the proper hardware and software configurations needed to get the modules working. We note a couple of key findings:

- All three modules require an approximately 50:50 mark to space ratio. This means that on a time average on the order of several milliseconds there need to be an equal number of 1's and 0's received by the receiver module. This ratio can be accomplished by encoding schemes such as Manchester Encoding, which we will be discussed in Section 7.6.4.
- The receiver module needs to be resynchronized at the onset of each packet. Inter-packet noise lacks the 50:50 mark to space ratio and serves to desynchronize the module. Thus at the onset of each packet we need to resynchronize the module with 1[ms] to 3[ms] of alternating marks and spaces, which correspond to sending 0xAA or 0x55 across the transmitter. (The exact timing specifications vary from module to module and are detailed in Section 7.7.3.)
- The USART also needs to be resynchronized at the onset of each packet. Prior to module synchronization data into the USART from the receiver has

been the noise on the wireless channel. Thus after module synchronization the USART needs to be synchronized such that the start of the packet data stream corresponds to the start of byte, i.e. the data stream is byte aligned to that on the USART. This is done by sending one to two byes of mark, 0xFF. Recall that during normal operation of the USART, the channel remains high when no data is being sent and then the line drops low for the start bit of a byte. Thus the effect of sending OxFF is to keep the channel high long enough allow the USART to synchronize with the end and start bits of the OxFF bytes.

7.6. Functional Design

7.6.1. Overview

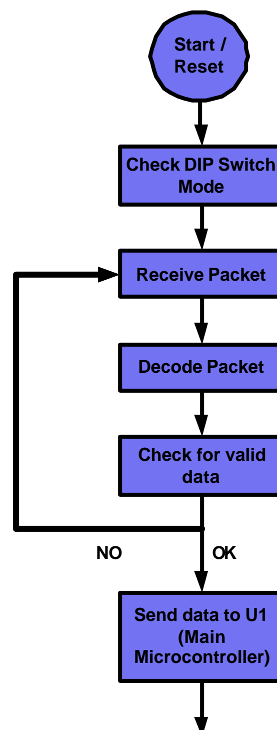


Figure 7.6.1.1

On startup or system reset, the wireless microcontroller begins by reading in the wireless dip switch value (SW7), which is in the range from 0-3. Depending on the number read from the dip switch, different pieces of software are run to handle the different timings required by the different modules. The microcontroller then enters its normal operational state of waiting for a packet of the correct length. Once a packet is received, the data is decoded and then checked for validity. After the validity test,

correct data is sent to the main microcontroller; otherwise the wireless microcontroller waits for the next packet and continues in this operational state.
Currently, all of the data that is decoded is sent to the main microcontroller on the robot, instead of parsing out only the data necessary for that particular robot. Our initial prototype had only one way communication, so this was our initial software implementation. This implementation satisfies the necessary data rate. For future improvements, we have modified the hardware to handle two way communication so that the main microcontroller, U1, which can send the wireless microcontroller the robot's ID number. Using this ID number, U2 will be able to send only the necessary data to the robot's main microcontroller. Because the modules that we are using offer a significantly high data rate, this is currently a low priority, however this functionality will need to be implemented in software if we move to an 11v11 system.

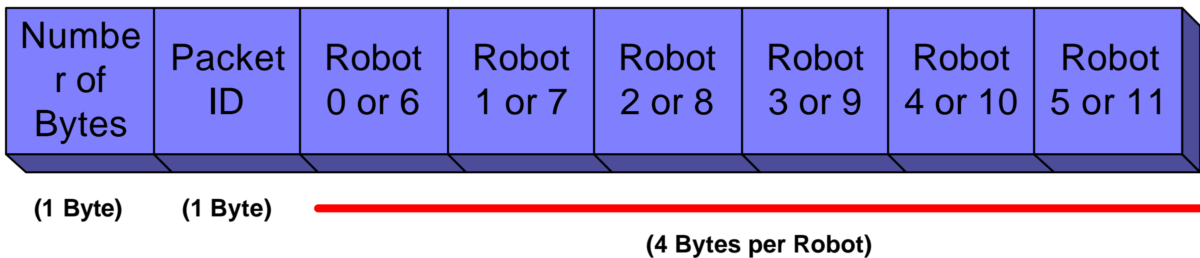
7.6.2. **Packet Structure**

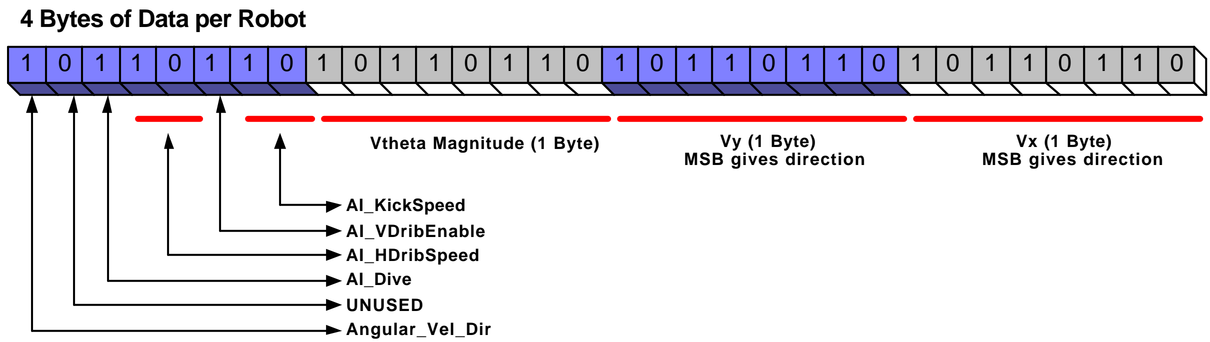
The figure below shows the packet structure for the 2002 Robot System. This packet structure is byte-identical to the packet structure developed for the 2001 11v11 system and is considered to be the standard packet structure moving forward.

Referring to the diagram we see that the first byte of data refers to the number of bytes in the packet. Thus this packet should always read as 0d26. This byte is necessary for operation of the Radiometrix RPC module, and remains part of the packet structure for all modules.

The second packet is the Packet ID. This number reads as '0' if the packet pertains to Robots 0 to 5, or '1' if the packet pertains to Robots 7 to 11.

The rest of the packet consists of 6 sets of 4 bytes, each corresponding to a particular robot. The individual bits are described in the diagram below.





7.6.3. **Hardware Design**

Figure 7.6.3.1 shows the hardware overview for the RX2 and RX3 systems on the 2002 robot. (The RPC connects to the U2 microcontroller via a parallel interface and so is not shown here.)

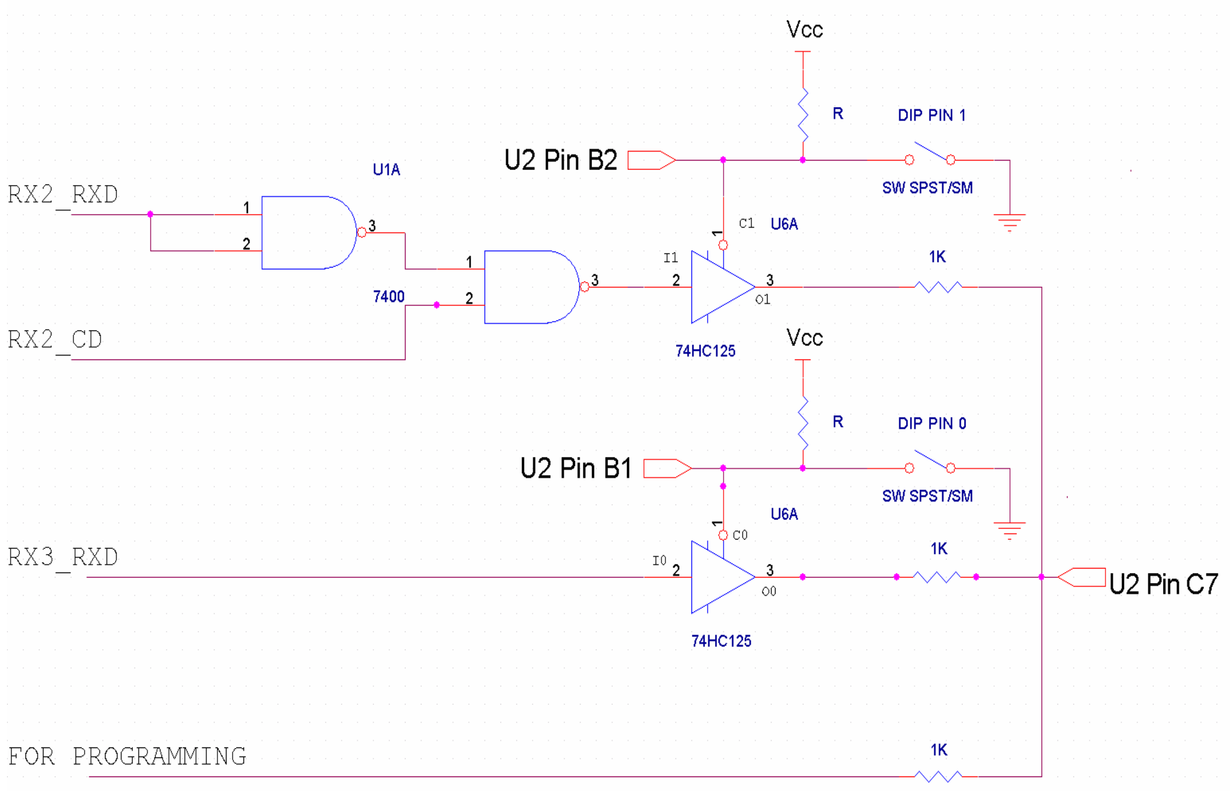


Figure 7.6.3.1: Schematic Diagram of RX2 and RX3 Interfaces to U2 Microcontroller

RX2_RXD, the digital output from the RX2 module is connected to both inputs of a NAND gate such that the output is the logical NOT of RX2_RXD, i.e. ~(RX2_RXD). This output and RX2_CD, the RX2 carrier detect signal are the inputs to a second NAND gate, such that the resulting output, I1, is a logical 0 if and only if the received digital data RX2_RX2 is a logical 0 while the carrier detect line is asserted. The truth table for this output is:

RX2_RXD	RX2_CD	Output I1
0	0	1
0	1	0
1	0	1
1	1	1

Table 7.6.3.1: RX2 Hardware Logic Structure

Why would we want our digital data from the RX2 module to be defined in this manner? Note that we would like our system to output a digital 0 when RX2_RXD is low AND RX2_RX2 is asserted and that we would like our system to output a digital 1 when RX2_RXD AND RX2_RXD are both asserted high. Next recall that when data is not being sent across the USART, the line should remain asserted high. Note this desired effect is produced when RX2_CD line is low. Our digital logic thus produces our desired output for all states of RX2_RXD and RX2_CD.

We next put this I1 signal through a tri-state buffer before it goes to the U2 Pin C7 hardware RX interrupt line. DIP PIN1 of the Wireless DIP controls the C1 control line into this tri-state buffer. This controls whether the output of the tri-state buffer should be the input I1, or if it should be put into a high Z state. The DIP switch allows the output to be kept in the high Z state for all modes except when RX2 is the selected module for use in the 2002 Robot system. The truth table for this tri-state buffer can be represented by:

I1	C1	O1
X	L	Z
0	1	0
1	1	H

Table 7.6.3.2 Tri-sate Buffer Truth Table

Since the RX3 module lacks a carrier detect signal, the RX3_RXD digital output from the RX3 module is input directly into a tri-state buffer which performs that

same functionality as that for the RX2 module. Here DIP PIN0 of the Wireless DIP controls the C0 control line into this tri-state buffer. Thus for RX3 we have:

RX3_RXD	C0	O0
X	L	Z
0	1	0
1	1	H

Table 7.6.3.3 RX3 Hardware Logic Structure

The outputs O0 and O1 of the two tri-state buffers, along with the U2 programming line are all connected to Pin C7 of U2. Each line has associated with it a 1k resistor to prevent against driver contention from the other lines.

As detailed in Section 9.2.5, the Wireless DIP configurations for the different Wireless modes are:

DIP P0	DIP P1	O0	O1	State
0	0	Z	Z	Programming Mode RPC Mode
0	1	Z	I1	RX2 Mode
1	0	I0	Z	RX3 Mode
1	1	-	-	Undefined (Resistors prevent contention.)

Table 7.6.3.4: DIP Configurations for Wireless Module States

7.6.4. Software Implementation

Recall the functional diagram from Figure 7.6.1.1 and the discussion in Section 7.5. The transmitter first sends the required setup information of 1 to 3ms of alternating mark and space, followed by two bytes of all marks. This is followed by a Manchester Encoded version of the data from AI.

The main data manipulation in the wireless microcontroller is the Manchester decoding. This encoding scheme can be described as replacing a *0 → 01* and a *1 → 10.* Manchester encoding works well for our application because it guarantees a balanced packet of equal numbers of ones and zeros, which gives us the best receiver sensitivity. For our purposes, we performed decoding as a lookup table.

(Note: More efficient encoding schemes will be investigated in the summer We will also be adding more information to this section regarding the actual software code that is being used. Unfortunately, due to time constraints we did not have time to add all of this information to this document.)

Decimal Value	Binary Value (Index)	Encoded Value
0	0000	01010101
1	0001	01010110
2	0010	01011001
3	0011	01011010
4	0100	01100101
5	0101	01100110
6	0110	01101001
7	0111	01101010
8	1000	10010101
9	1001	10010110
10	1010	10011001
11	1011	10011010
12	1100	10100101
13	1101	10100110
14	1110	10101001
15	1111	10101010

Figure 7.6.4

7.7. Testing

7.7.1. Overview

Preliminary testing of the Wireless modules needed to be performed before the 2002 Robots were completed, or even designed, to ensure that the chosen wireless modules would work with the system. In order to do this we needed to develop a test system that would model the 2002 RoboCup Wireless system. We detail the design and operation of this system in Section 7.7.2.

We originally planned to use this Wireless Test System to perform three different tests: a latency test, a maximum bit rate test, and a bit-error test. We define these tests in greater detail in Sections 7.7.3 though 7.7.5 respectively.

At this point we have only tested the modules using our latency test. During the latency testing phase of the project, we discovered that the TX2/RX2, TX3/RX3, and SE200

modules could not consistently output stable data. We traced this problem to the fact that in our test setup we did not have a way to effectively mount the antennas onto the modules. We found that moving the antennas just slightly would dramatically change the module's ability to read valid data. This issue did not affect our ability to perform the latency test since we could measure the latency during periods the antennas were aligned correctly.

While we could still use our Wireless Test Setup to perform latency tests on these different modules, their inability to reliably accept valid data meant that we could not perform the maximum bit rate or bit-error tests until after we could mount the modules onto a more secure platform. For this reason we chose to delay performing these tests until after we designed the wireless PCB boards that these modules would be mounted on when integrated as part of the 2002 RoboCup System. Since our support for the RPC in this year's design meant that we could get a 2002 System operational using merely the RPC, priorities and human resources were shifted away from continuing these tests and toward other parts of the RoboCup system. These tests will be performed over the summer, and their results used to determine optimal data encoding and antenna configuration.

7.7.2. Procedure

Before we could begin testing, we first needed to design a Wireless test system that would simulate the actual 2002 RoboCup system. As shown in Figure 7.7.2.1, we used four microcontrollers to simulate this RoboCup system.

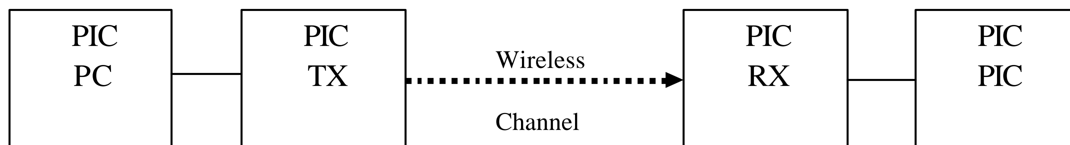


Figure 7.7.2.1 Four microcontrollers representing the wireless system.

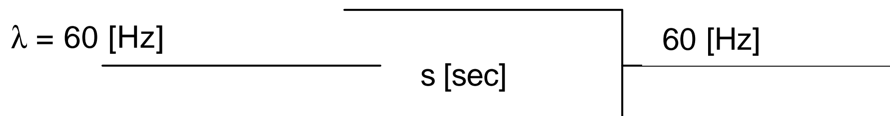
Referring to Figure 7.7.2.1, we see that the first microcontroller, PIC PC, represents the PC which is going to be sending the data to the PIC TX module every 1/60th of a second. PIC TX represents the microcontroller which will be taking the incoming data from the PC, encoding that data, and then sending it out across the wireless channel. PIC RX represents the microcontroller U2 on the robot which detects incoming data on the wireless channel, decodes this data, and then sends it to microcontroller U1 on the robot.

PIC PIC represents microcontroller U1 which receives the packet data and then parses it into the individual packet components.

In terms of our testing metrics, we note that they apply only to the actual wireless channel itself, and hence only effect parts PIC TX and PIC RX of our Wireless Testing Model. We use PIC PC and PIC PIC to emulate the parts of the system that PIC TX and PIC RX need to communicate with. To do this we connect the change on interrupt PIN B4 of PIC PC to a function generator. On this interrupt we have PIC PC send a packet of data to PIC TX. Thus we see that we can adjust the rate at which PIC PC sends data, and hence the arrival rate into PIC TX by adjusting that frequency of the square wave 0 to 5V frequency generator.

7.7.3. Latency Test

In terms of latency, what we would really like to measure is the latency as defined as the total time required from the moment the first bit of information enters part of the wireless system until the moment that the last bit of information leaves that part of the wireless system. Our main concern is that any one stage of the RoboCup system is less than 1/60th of a second. Why is this? With our system defined by the input frame rate of 60 fps, this means that not one subsystem can have a processing time of greater than 1/60th of a second. To see this let us model a subsystem as a simple queuing system:



With the arrival rate into any subsystem fixed at 60 Hz, we see that in order to keep the departure rate of our system fixed at 60 Hz, we need to keep our system stable. This occurs only when the arrival rate multiplied by the service time is less than or equal to 1. Thus the service time of each subsystem must be less than 1/60th of a second.

Thus, referring to Figure 7.7.2.1, we see that the service times for the PIC TX, wireless channel, PIC RX must all be less than 1/60th of a second. We note that the service time on the wireless channel is minimal since it corresponds only to the propagation of radio waves in air. We thus need only to measure the latency associated with PIC TX and PIC RX. We define T1 to be the time from the receivable of the first bit of data to the transmission of the last bit of data on PIC TX. Similarly we define T3 as the time from the receiving of the first bit of data to the transmitting of the last bit of data in PIC RX. We measured these values by setting a port pin high at the onset of data being received and then setting it low again at the end of the data being transmitted.

Device	Data Rate (bps)	T1 (ms)	T3 (ms)	Notes

TX2/RX2 40kbps	38,400	19.0	15.4	Initialization is 12 bytes of 0xaa or approximately 3ms Using Manchester Encoding
TX2/RX2 40kbps	38,400	17.6	15.4	Initialization is 6 bytes of 0xaa Using Manchester Encoding
TX2/RX2 40kbps	38,400	17.84	15.4	Initialization is 7 bytes of 0xaa Using Manchester Encoding
TX2/RX2 160kbps	38,400	17.84	15.4	Initialization is 7 bytes of 0xaa Using Manchester Encoding
TX2/RX2 160kbps	57,600	14.2	11.12	Initialization is 16 bytes of 0xaa Using Manchester Encoding
TX3/RX3 64kbps	38,400	18.10	15.4	Initialization is 7 bytes of 0xaa followed by two start bytes Using Manchester Encoding
TX3/RX3 64kbps	57,600	< 16	< 16	Measured as less than 16 ms, but actual value not recorded. Using Manchester Encoding

Table 7.7.3.1: Latency tests for modules in various configurations

In Table 7.7.3.1, "Device" refers to the specific device tested. The kbps rating is listed as part of the specification since the modules can be purchased in different configurations based on baud rate. "Data Rate" refers to the data rate that U2 sends data to the wireless module, and thus the rate at which the wireless module sends data across the wireless channel.

Referring to Table 7.7.3.1, it is clear that T1, the time required from the first bit being received by PIC TX (U2) to the last bit being sent by PIC TX (U2) is the timing limiting factor in our system. This makes sense because since it also sends the channel synchronization information it has the largest amount of data to send.

We also see that the required data rate to maintain the 1/60th second service time falls between 38,400 and 57,600 bps. Since both the TX2/RX2 and TX3/RX3 modules specification supports data rates above this requirement both modules satisfy our latency requirement.

7.7.4. Maximum Bit Rate Test

We would also like to determine what the maximum amount of data is, so that we can send over a 1/60th of a second period. We currently are sending 26 data bytes per frame. We note that these robots may also be used in RoboFlag, an 11 versus 11 match, or other situations that require more than five robots. In these situations, we need to send two different sets of packets, one for Robots 0 to 6, and another for Robots 7 – 12. Sending this data can be accomplished by either decreasing the frames per second or increasing the data rate of the system. Our current implementation for RoboFlag and 11 versus 11 is to lower the frames per second to 30fps, but with the latter more desirable, we use our maximum bit rate test to determine its feasibility. This test will be performed over the summer.

7.7.5. **Bit Error Test**

We would also like to measure the bit-error rate of the wireless devices. We plan to do this by having the transmitter send "known" data to the receiver. The microcontroller on the receiving end will then compare the data received to the data expected, and keep track of when the bits are not in agreement. Since BER tends to be very small, we will run this test for 30 minutes to see how many errors have occurred during that time period, where 30 minutes is an arbitrarily large number and is also on the order of the length of a RoboCup match.

We plan to conduct this BER test in two different environments. The first test will be in a relatively RF noise-free environment. The second test will be in a relatively RF noisy environment. We will simulate the RF noisy environment by having many RF transmitters transmitting, as well as, at Evan Malone's suggestion, having a Dremel turned on to generate broadband RF noise.

We would also like to note that we can use this BER test to evaluate the need for error-checking and error-correction in our system. While our "real-time" RoboCup system makes Error Detection / Automatic Retransmission Request (ED/ARQ) techniques infeasible, Error Detection / Forward Error Control (ED/FEC) techniques could conceivably be used. By sending redundant data, we could detect that an error has occurred, and then correct it, or possibly ignore the data sent for that cycle. We are cognizant of the fact that ED/FEC techniques will increase the amount of data in our system as well as increase the processing time, so our plan is to first measure the actual BER and then determine if these schemes are necessary. This testing will be performed over the summer and used to find our optimal choice of antenna and software configurations for the different modules.

7.8. *On-Robot Hardware Design*

7.8.1. **Hardware Layout for Wireless RX Board**

The RX Wireless board serves as a mounting board for the three different modules that can be used in the 2002 Robot System. This board serves as the interface between the connections on the module and those on the digital board. They also contain filter capacitors and other signaling components. Even though only one module will be mounted to the RX Wireless board, to reduce design complexity we designed one board to support all three modules, rather than three independent boards. This design choice also provides lower production costs since we can send out around 50 of the same type of board and increased

maintainability since one general board can be used for three different module types.

The circuit diagram for the RPC is shown in Figure 7.8.1.1. Here we see that the signals are connected directly between the RPC header which connects to the RPC module and the RPC to Wireless To Digital Board Header which connects to the Digital Board. Pin 11 of the RPC header is the connected to the ground via a 22uF filtering capacitor.

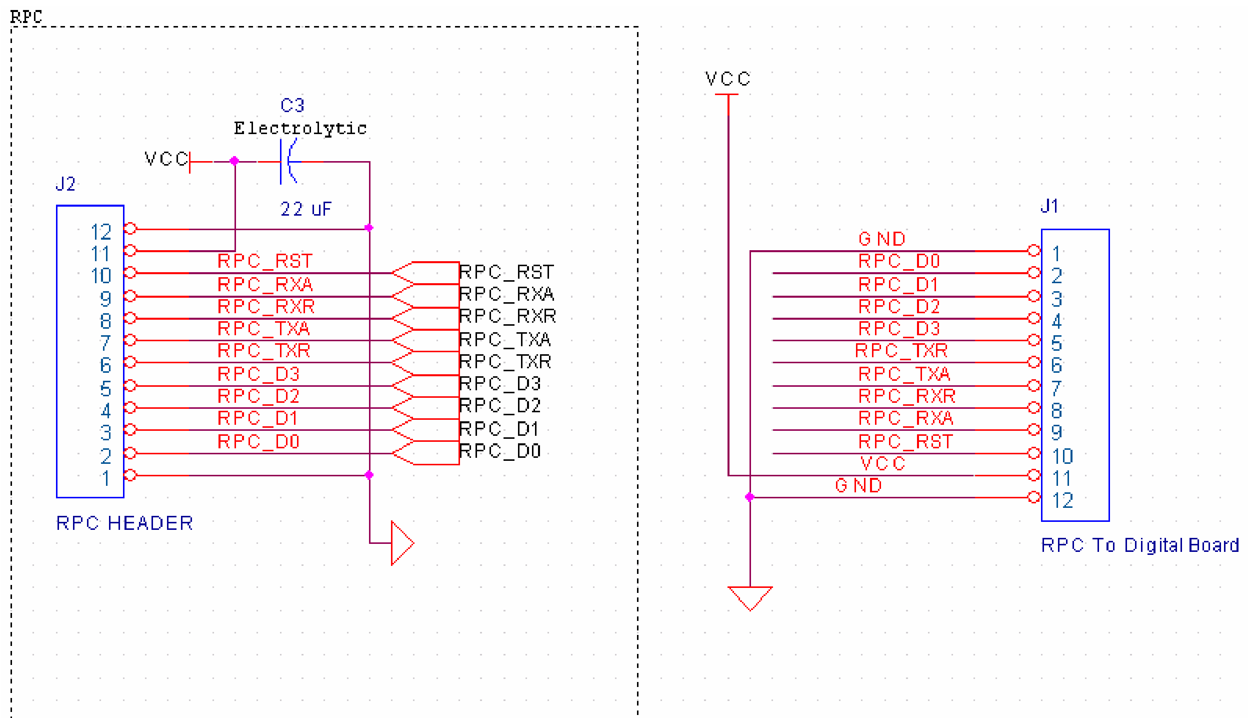


Figure 7.8.1.1 Circuit design for the RPC.

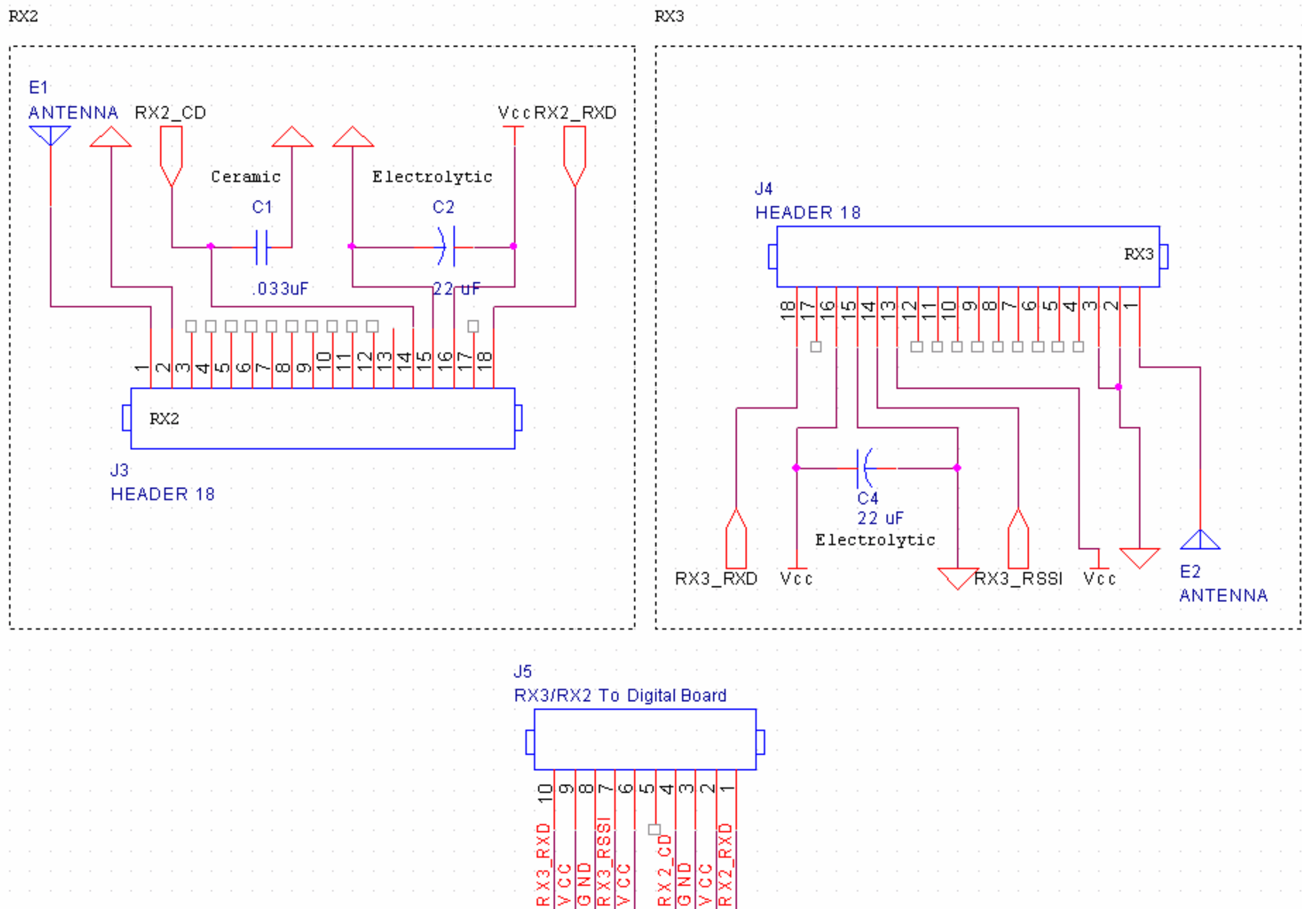


Figure 7.8.1.2: Circuit design for the RX2 and RX3 modules.

In a similar fashion to the RPC, signals from RX2 and RX3 are routed from the headers to the RX3/RX2 to Digital Board Header. Again 22uF filtering capacitors are used between power and ground. Additionally the RX2 module contains a .033uF low-pass filtering capacitor between RX2_CD and ground. We experimented with different values for this low-pass filter, and found .033uF to be an optimal value to eliminate noise on the CD line while not rejecting too high of a frequency that the actual CD signal would not be detected.

7.8.2. **Antennas**

7.8.2.1. **Antennas Types**

The two main categories of antennas that exist are integral and external antennas. External antennas have the best performance as one would one guess. However, the integral antenna types provide more compact solutions and are the preferred solution for portable applications. Four main types of integral antennas were considered:

- ¼ wave whip
- Base-loaded whip
- Helical
- Loop

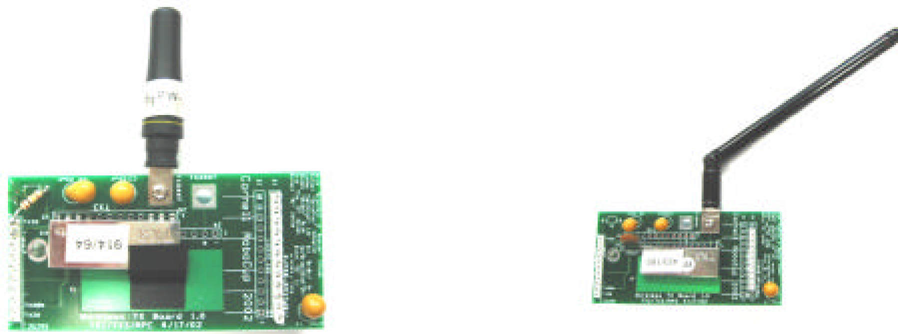
Quarter wave whip antennas are the length of ¼ of the desired wavelength. They can be made of wire, PCB traces, or a combination of the two. The length in millimeters is equal to 71250 divided by the frequency in MHz. A variation on this antenna is the base-loaded whip antenna which uses a coil at the base to shorten the length of the wire. The helical antenna is like a spring with reception characteristics being determined by the length, number of turns, and turn spacing. Lastly, the loop antenna looks as its name implies, a loop.

Radiometrix had a good comparison of different factors between the different types, which helped us in choosing our antenna types. The information is summarized below.

	¼ wave whip	Base-loaded whip	Helical	Loop
Performance	+++	++	++	+
Ease of design set-up	+++	++	++	+
Size	+	+++	+++	++
Immunity to interference	++	+	+	+++

Because of its physical characteristics, the loop antenna was eliminated very quickly. We felt that the helical antenna might not be the best choice, because it wouldn't really extend outside of the robot's enclosure, reducing its reception capability. The helical also doesn't have great immunity to interference. We concentrated on whip antenna types, with a greater emphasis on ¼ wave types because of their performance, immunity to interference and ease of design, since we are first-time wireless users.

Below are the antennas that we are currently considering for our wireless modules.



7.8.2.2. **Connector Types**

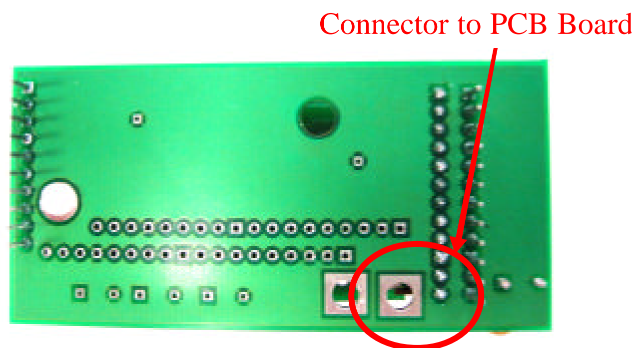


Figure 7.8.2.2.1

Many types of connectors exist to connect antennas to the system. We focused on two types of connectors, namely BNC and flat metal plates that connect via a screw. The easiest type of connector to mount to the wireless board was the flat metal plate as can be seen in the picture above. This type of connector appeared to be the most compact and be a standard connection type on the more compact antennas. They appeared to be designed to screw directly onto a metal pad on a PCB board.

7.8.2.3. **Mounting Considerations to the Wireless Board**

Mounting considerations to the wireless board were very critical as any oversight can cause degradation in the wireless performance. Major considerations included placement near microprocessors that emit radio interference at many different frequencies or other noisy components, the trace length to the antenna connection, and the contact to the actual PCB board. We decided to locate the wireless board as far away from the FPGA and four microprocessors as possible near the top of the board to have the antenna located above the two daughter boards. The use of a ground plane was also employed as an added way to reduce the amount of spurious noise that would effect the wireless module. Trace length from the wireless module to the antenna was minimized as much as possible. It should be in an order of less than 1/10 of the carrier wavelength; otherwise an impedance match strip line should be used. We created a large pad on the wireless board where we were able to connect the antenna with a screw and ensure a good quality contact. Other system considerations were how the vision system would handle the antenna showing through the top of the robot's hat. After some testing, the vision sub-group concluded that the antenna would not interfere with the vision system's recognition of the robots color identification markers.

7.8.2.4. **Optimization**

More extensive testing on antenna choices will occur over the summer because our breadboarded test setups were not very robust. It was decided that initial tests would be performed to determine feasibility. After the system's boards have been fabricated, extensive testing will generate much more reliable data.

7.8.3. **Interference**

Interference can be a major problem in the design of a wireless system. Radiometrix's application notes gave us a lot of information on potential sources of interference that we needed to be aware of in building our system. In previous years, many failures occurred with the RPC where the receiver just went dead for a short period of time. These phenomena could have easily been explained by interference from neighboring computer equipment.

Some possible sources of failure include:

- Computers and other digital electronics can produce broadband noise and weak clock harmonics to 1GHz and above. It is worth noting that even EMC-approved equipment could still be legally radiating spurious signals that are 40-50dB above our example receiver's noise threshold.

- An extremely common and particularly difficult variation on the above is interference from digital electronics within the product in which the receiver is used. Since the interfering source is usually within 5 to 20cm of the receiving antenna and is always present, it masks all incoming signals below a certain level. The result is that the receiver is permanently "deaf".
- Microwave ovens and industrial heaters - multiple unstable 2.4GHz carriers.
- Switch mode power supplies - harmonics up to 100MHz and above.
- Amateur radio transmissions on 433 MHz and other low power radio systems in the local area.

Because of what we read about positioning wireless modules, we had to be very careful with our 4 microcontrollers and our FPGA. We also had to be especially careful to keep the modules as far away from the drive motors as possible. By mounting the board at the top edge, we also gave the antenna the highest probability of having good reception. Another consideration was to employ a ground plane to isolate spurious noise from the wireless board. With all of these strategies in mind, we chose our current location of the wireless board along with the use of a ground plane on the wireless board.

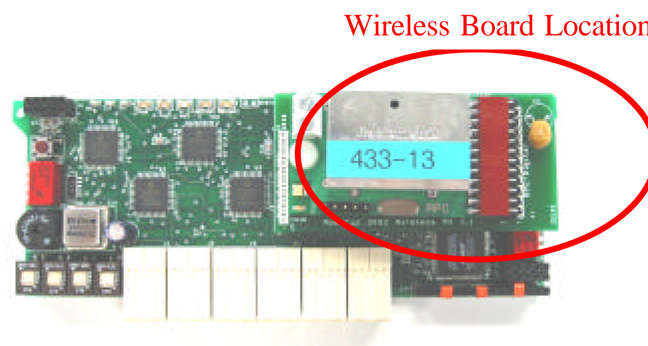


Figure 7.8.3.1

Some methods mentioned to overcome the interference were discussed. One such method involves increasing the output transmitter power above the receiver's detection threshold in order to achieve a more reliable link. Another method is to use multiple receivers so that if one is in a null zone or receives data with errors, the other can compensate for this problem. Furthermore, having multiple transmitters spatially separated and sending the same data with a slight time offset affords similar benefits. These strategies were difficult to employ especially with our space constraints and particular application. However, these are strategies that should not be forgotten if we decide to move into a larger league.

Information provided in Radiometrix application note entitled "99% is Good Enough" at http://www.radiometrix.com/html/products/apnt6.html

7.8.4. **Mounting Considerations**

A separate wireless board was chosen over mounting modules directly on to the digital board. This mounting method allows a more stable mounting of the module with a shorter path to the attached antenna. The goal with mounting an antenna is to keep the trace as small as possible. It was also argued that the separate board would be able to isolate the module from some of the noise in the system. Secondly, modules should be mounted on an area of ground plane to reduce noise and allow for better reception. Modules are connected to the digital board via two headers and a nylon screw to hold the module in place.

7.9. Off-Robot Hardware Design

7.9.1. **Hardware Layout for TX Wireless Board**

Similar to the RX Wireless board, the TX Wireless board also serves as a mounting board for the three different modules that can be used in the 2002 Robot System. Again, this board serves as the interface between the connections on the module and those on the digital board. It also contains filter capacitors and other signaling components. Just as with the RX board, even though only one module will be mounted to the TX Wireless board to reduce design complexity, we designed one board to support all three modules instead of three independent boards. This design choice also provides lower production costs since we can send out around 50 of the same type of board and increased maintainability since one general board can be used for three different module types.

The circuit diagram for the RPC is shown in Figure 7.9.1.1. Here we see that the signals are connected directly between the RPC header which connects to the RPC module and the RPC to Wireless To Digital Board Header which connects to the Digital Board. Pin 11, RPC_ON, of the RPC header is then connected via a 22uF filtering capacitor to ground. While Pin 11 of the RPC is Vcc, we connect is to the Transmitter board via the signaling pin RPC_ON. The Transmitter board will have all three modules connected simultaneously, but we will only want one of them powered at a time. Thus when we desire to use the RPC, RPC_ON will be high, and when we are using a different module RPC_ON will be low. Similarly as shown in Figure 7.9.1.2 the TX2 module has TX2_ON and the TX3 module has TX3_ON.

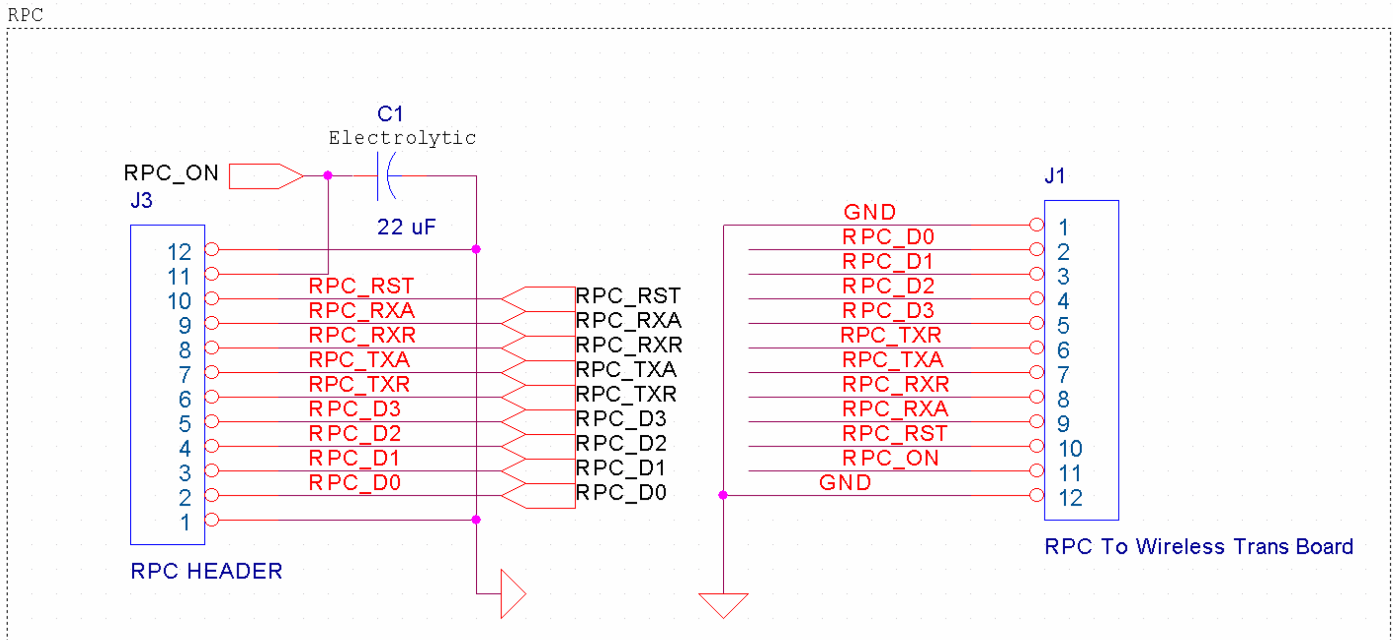


Figure 7.9.1.1: Schematic diagram for the RPC module on the Wireless Transmitter Board

The TX2 module also has a .033uF low-pass filtering capacitor on the TX2_TXD line. We experimented with different values for this low-pass filter, and found .033uF to be an optimal value to smooth the digital input into the TX2 module without degrading the input pulse too much that it could no longer be readable by the RX2 receiver. The TX3 module has a 100k resistor in series along the TX3_TXD line. As per the Radiometrix documentation this is required to lower the input signal from a 0V – 5V signal to a 0V – 3V signal. As with the RPC, both modules also have 22uF filtering capacitors between power and ground.

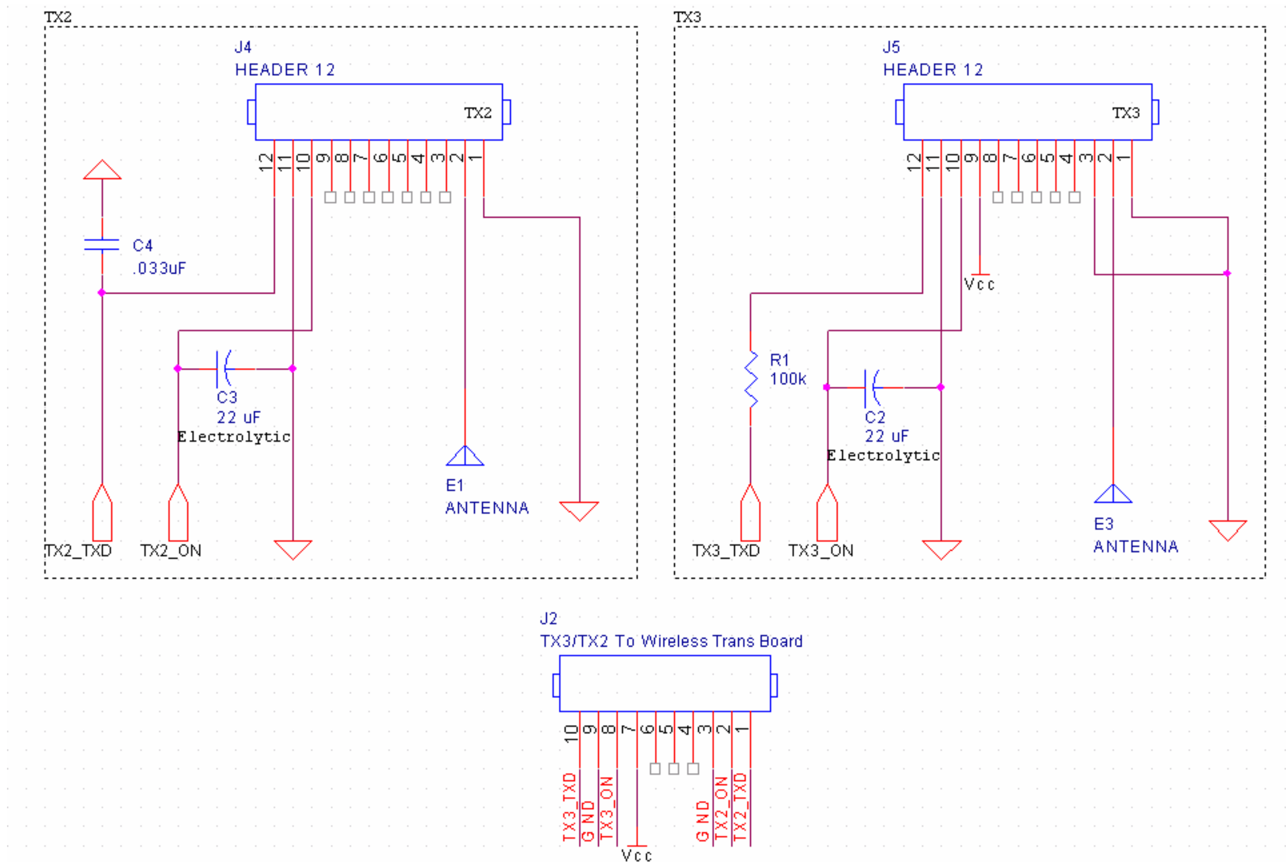


Figure 7.9.1.2: Schematic diagram for the TX modules on the Wireless Transmitter Board

7.10. Conclusion

Overall, our recommendation to next year's team is to leave a lot of extra time in their schedule, realizing that wireless is an art. Limit the number of module choices to a reasonable amount and seek help whenever you stumble into a wall. Many countless hours can be spent debugging wireless modules. One of the largest surprises to a newcomer to the wireless area is how great an impact a small tuned filter, consisting of a capacitor, can have on the performance of the module. In most cases, the filtering capacitor that was used caused the module to work extremely well compared to basically not functioning well at all. Also, it appears that antennas are very important to the modules getting clear reception.

.

8. Goalie Systems

8.1. Overview

The mechanical realization of the goalie started out very early with the rest of the robot but when it came to building robots, the goalie was put on the hold with the intension that that goalie would be constructed over the summer if time allows. For this reason, the mechanical engineers were not able to give us a final list of requirements. Therefore the electronics for the goalie was designed with the maximum flexibility possible; our electronics can be used to control any of the multiple configurations that themechanical engineers were considering.

The leading mechanical design requires controlling the following actuators: a bi-directional solenoid, a horizontal dribbler motor (uni-directional), and a "cocking " chip-kick motor. The "cocking" chip-kick motor also requires a switch that is used to determine when the motor is in the "cocked" position.

8.2. Goalie Systems

8.2.1. Bi-directional Solenoid

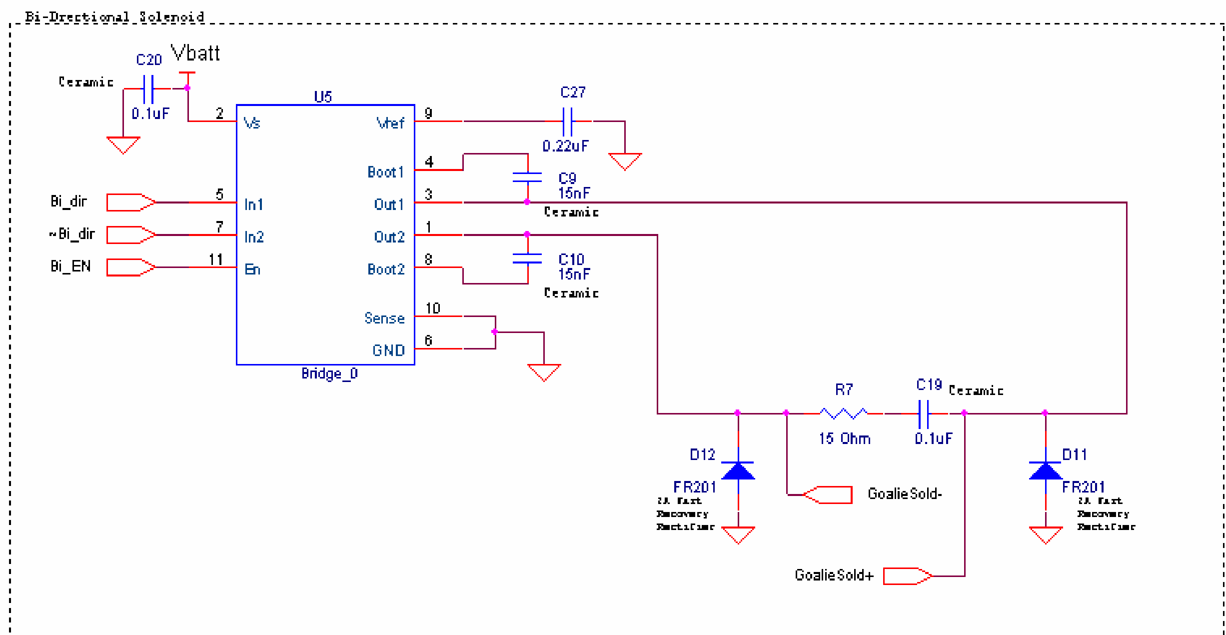


Figure 8.2.1: Bi-directional Solenoid Schematic

The Bi-directional solenoid is controlled via an H-bridge. The circuit is exactly the same as the motor control circuit (Figure 6.6). The three signals that are used to control the solenoid are EN, DIR, and ~DIR. When EN is high, the motor is turned on, while DIR and ~DIR are used to control the direction of the motor. When DIR = 0 and ~DIR = 1 the motor rotates in one direction, and when DIR = 1 and ~DIR= 0 the motor rotates in the other direction.

8.2.2. Horizontal Dribbler Motor

The goalie's horizontal motor will be controlled with the normal robot's horizontal motor control circuitry (see Figure 6.5)

8.2.3. Chip-Kick

The chip-kick uses a motor that cocks the chip-kick and then the same motor releases the cocked chip-kick. To reduce circuitry, the chip-kick motor will be powered through the existing vertical dribbling circuit, because the goalie robot will not have vertical dribblers.

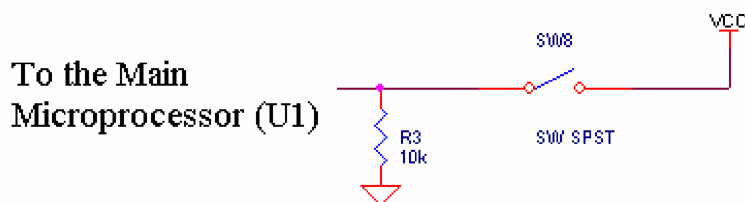


Figure 8.2.3

The main microcontroller (U1) will turn on and off the chip-kick motor. A switch will be mounted next to the motor to determine when the motor needs to be stopped (Figure 8.2.3). This switch is connected to U1 with a pull-down resistor. When the switch is closed, the line is pulled high through the switch.

8.2.4. Motion Control

The goalie's motion control will be implemented exactly like the field player's motion control with the following exceptions: motion control is that the goalie may have 3 wheels instead of 4 and the goalie will definitely have a different wheel geometry than

the normal field player. This geometry will change the code geometry transformation code in U1.

Currently, the goalie's geometry has not been determined, so the new geometry transformation cannot be determined.

8.3. Customized Packet Structure

To implement the goalie's functionality, the data packet structure being sent from artificial intelligence will have to be changed. The first three bytes, Vx, Vy, and Vtheta will not need to change. The last byte will need to change to enable chip-kick control instead of vertical dribbler control.

To keep track of which player is the goalie, we will set-up the robots so that robot number 0 is always the goalie.

TBD

9. System Use

9.1. Microcontroller

9.1.1. Setting Robot ID

Use the red 4 bit DIP switch to set the ID for the robot. The first 5 numbers (0-4) are currently for setting the robot numbers. The rest of them have till not been defined. However we plan to implement different test modes for IDs 10-15 (A-F).

9.1.2. Test Modes

The test modes for the robot have to be defined in the summer of 2002. The following test modes are planned for this year's robots.

1. Wireless test mode
2. Horizontal dribbler test mode
3. Motion control test mode
4. Kick test mode

All the test modes would be used to test some sub-system of the robot.

9.2. Programming

9.2.1. Overview

This section details the procedure of events for programming the four 16F877 microcontrollers on the 2002 Digital Board. This is a two step procedure. The first time that the microcontrollers are programmed, we need to load them with the Bootloader program. This program allows successive re-programming of the microcontrollers to be done via in-circuit serial programming. In order to do this we need to connect the microcontrollers, one at a time, up to the PICSTART Plus Development programmer via the programming interface module that we designed.

Once we have the bootloader program loaded onto the microcontroller, we can program them via the serial/USART port using the PIC downloader.exe program.

9.2.2. Watchdog timers

There are watchdog timers in all the microcontrollers on the robot. The reason for using the watchdog timers is that the microcontrollers are reset if they do not receive any serial data for a particular amount of time.

Resetting the system makes sure that the system re-initializes as soon as possible.

NOTE: Setup timer 0 before setting up the watchdog timer.

9.2.3. Programming the Bootloader onto the PIC16F877

1. The 20 Mhz clock must be removed from the board during programming of the microcontrollers.
2. Position the switches, SW1 to SW4 to the proper states. The switch for the microcontroller to be programmed should be in the on/high/1 state, while the other three switches should be in the off/low/0 state. Here "high" refers to the top of the board, i.e. "high" is the top of the switch if you are oriented so that you can read the text on the switch. Note when we refer to the switch being "on", this refers to the microcontroller being "off".
3. Connect the 2002 Digital Board to the PICSTART Plus Development Programmer via the constructed interface module. This module connects the MCU programming pins (J7 to J10, where J7 corresponds to micro 1 and J10 corresponds to micro 4) to the PICSTART programmer. The 5V Vdigital power needs to be supplied (i.e. the robot should be turned on) to the 2002 Digital Board. Ground should also be shared between the PICSTART programmer and the 2002 Digital Board.
4. Open "bootldr.pjt" in MPLAB IDE. Make sure that the PIC16F877 processor is selected. Also double check that lines 66 – 69 of the code read:

   ```
   list p=16f877                 ; <<< set type of microcontroller
                                 ; <<< set same microcontroller in the project
   #define FOSC D'20000000'      ; <<< set quartz frequency [Hz], max. 20 MHz
   #define BAUD D'57600'         ; <<< set baud rate [bit/sec]
   ```

   Line 66 defines the chip, PIC16F877, that we are using. Line 68 defines the 20 Mhz clock rate that we are using for our microcontroller. Line 69 defines the baud rate, 57.6 kbps, that we will be using to program the microcontrollers by in-circuit serial programming via the PICdownloader.exe program. Note: This 57.6 kbps rate does not reflect the rate of all USART communications speeds, only the speed at which the microcontroller will be programmed via ICSP.
5. Go to menu Project -> Make Project. Verify that there are no errors or warnings. Build results should read, "Build completed successfully."
6. Go to menu PICSTART Plus -> Enable Programmer
   If the MPLAB IDE program crashes at this point, it may prove helpful to log off or restart your computer and try again.
7. In the configuration bits window, select the desired configuration bits. We are currently using:

Configuration Bit	**Setting**	**Reason**
Oscillator	HS	A 20 Mhz clock is a high speed clock.
Watchdog Timer	On	This will allow the processor to be restarted on a software lockup. To

		work properly this must be implemented in software with the setup_wdt() and restart_wdt().
Power Up Timer	On	We calculated the Vdd rise time to be approximately 200 usec which is faster than 0.05V/msec. MCLR changes at 0.8*Vdd or 4V. Since our external reset circuitry reaches 0.8*Vdd after 872ms, this power up timer should have no effect. However, since we desire Brown Out Detect, we leave this timer enabled.
Code Protect	Off	Disabled to allow ISCP.
Brown Out Detect	On	Prevent abnormal operation on low supply voltage.
Low Voltage Program	Disabled	Not necessary when using PICSTART Plus programmer.
Date EE Protect	Off	Disabled to allow ISCP.
Flash Program Write Enabled	Enabled	Allows us to write to Flash memory via the PICdownloader.exe program.

8. Go to menu PICSTART Plus -> Program/Verify
   You may receive a message informing you that the firmware for the PICSTART programmer is out of date. Ignore this message and click OK.
9. Click on the "Program" window in the Program/Verify window. (Note: We do not really need to program all of the microcontroller' s memory. Instead we can program from 0000 to 0008, and then again from 1F28 to 1FFF. This is done by programming the microcontroller twice. During the first programming, type "0" into the "Start Address" field and "8" into the "End Address" field, and then hit "Program". During the second programming, type "1F28" into the "Start Address" field and "1FFF" into the "End Address" field and hit "Program". Note that depending on whether the microcontroller has previously been write protected, this option may or may not be possible.
10. Verify that the microcontroller has been programmed successfully. This can be checked by the word "success" or "failure" in the Program/Verify window. The microcontroller has now been programmed with the bootloader successfully.

9.2.4. Programming the PIC16F877

1. Compile your C program in CCS C. Compilation will produce a file called yourfile.hex.
2. Open PIC downloader.exe and use the "search" button to locate yourfilex.hex.
3. Set the port to the serial port you are using, and select 56000 as the baud rate.
4. On the 2002 Digital Board connect the GND, RX, and TX lines from the DB-9 cable to the desired destination microcontroller' s programming pins on J9.

5. Position the switches, SW1 to SW4 to the proper states. The switch for the microcontroller to be programmed should be in the off/low/0 state, while the other three switches should be in the on/high/1 state. Here “high” refers to the top of the board, i.e. “high” is the top of the switch if you are oriented so that you can read the text on the switch.
6. Click on the “Write” button in the PIC downloader.exe application. You should see “Searching for the bootloader” message in the Info bar. If you get an “Open Port Error” message, make sure that no other applications are currently using the selected serial port. HyperTerminal and MPLAB IDE are two common programs to do this.
7. Press and then depress the Reset Switch, SW5, on the 2002 Digital Board. You should see a “Writing, please wait!” message in the info bar.
8. The message “All ok” will appear If the microcontroller was programmed successfully. If the microcontroller was not programmed successfully, press the “Cancel” button, and the repeat steps 6 and 7.

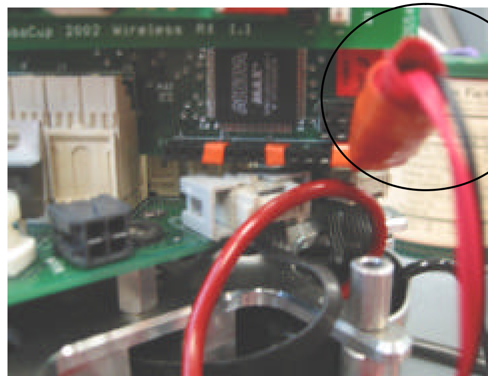


Figure 9.2.3: Programming cable and programming port

9.2.5. Programming the FPGA

Here are the detailed programming procedures (note: anything in quotations marks are the actually words you will see in the software):

1. Open the MAX+plus II software
2. Open the schematic file. You should always open the high level module if there is one. (“File”→ “Open”)
3. A new window “Open” will pop up for you to choose the file. Go to the directory where you store the files and pick the schematic file which ends with

.gdf. (Hint: Since all of our implementations are in schematics, clicking the "Graphic Editor Files" option under "Show in Files List" will make selection much easier). Always choose the files with name High or Main, since these are the top level schematics.

4. Before compiling, we need to make sure of the following two things:
    - We have assigned the correct device ("Assign"→"Device"). You should see EPM7128STC100-7 highlighted. If it is not, you first have to unselect the "Show Only Fastest Speed Grades" option, and then select EPM7128STC100-7 from the list.
    - We have assigned the input/output signals to the desired pins. ("Assign"→"Pin/Location/Chip"). A new "Pin/Location/Chip" window will pop up. All the assigned pins are listed under "Existing Pin/Location/Chip Assignments:" window. The pin assignments are shouldn' t be changed unless you have changed the design.
5. Set the current schematic to current project. ("File"→"Project"→"Set Project to Current File")
6. Compile the design ("MAX+plus II" → "Compiler"). A new "compiler" window will pop up and click "Start".
7. After the circuit is successfully complied, program the FPGA (""MAX+plus II" → "Programmer"). In the pop up "Programmer" window, one should verify the "File" and the "Device" to be correct. Click "Program" button. Sometimes, it might give you a warning about the pin assignment. It is all right.

The FPGA now should be programmed.
A common problem and their solution:
When programming the FPGA, an error message might pop up. "ByteBlaster is not present– check power and cables). You should first check if the cables are connected correctly. Second, leave the "Programmer" window on the screen, go to top menu choose "Options"→"Hardware Setup". Make sure the "Hardware Type" is "ByteBlaster(MV)" and "Parallel Port" is "LPT1:". In some cases, you have to reinstall ByteBlaster software. Please see the Altera handbook for detail about how to do that.

9.2.6. Setting Wireless Module ID

The Piano DIP Switch SW7 is used to set which wireless module is going to interface with microcontroller U2. The setting of this DIP designates both in hardware and software the choice of wireless module to be used. Referring to Figure 3.3.4.3.1, in hardware, the desired input line is set open, and the other line(s) are set to high Z. The output of these DIP switches also connects to pins B1 and B2 on U2. These pins are read during the initialization code of U2 to set the proper version of software to use for the wireless module being used.

Table 9.2.5 outlines the proper DIP settings for the wireless modules and programming mode. Refer also to Section 3.5.5.1.1 for a more detailed explanation of interrelationship of the Wireless Piano DIP Switch to the rest of the On-Robot Communication Receiver Hardware System.

Figure 9.2.5 Hardware Configuration of Wireless DIP

DIP P0	DIP P1	O0	O1	State
0	0	Z	Z	Programming Mode RPC Mode
0	1	Z	I1	RX2 Mode
1	0	I0	Z	RX3 Mode
1	1	-	-	Undefined (Resistors prevent contention.)

Table 9.2.5: DIP Configurations for Wireless Module States

9.3. System Debugging

There are many features included in this year's design to allow for easy debugging of the system. Some of these features are discussed below.

9.3.1. Inter-microcontroller Communication

9.3.1.1. Parallel vs. Serial Communication

Communication between the microprocessors is an essential design consideration when designing a multi-processor system. We looked into serial and parallel methods of communication. After careful consideration, we decided to implement the serial method of communication because of the following reasons:

1. Serial Communication only requires 2 physical lines of connections between microprocessors, saving I/O pins for other uses.
2. Serial Communication is easy to implement with the PIC16F877.
3. Serial Communication allows us to achieve a high data rate (up to 115,200 Kbps in hardware or software).

9.3.1.2. Serial Communication via PIC16F877

The PIC16F877 supports one hardware TX/RX pair and any other pair of pins can be a software TX/RX. The hardware RX institutes a 3 byte buffer which makes sure that no data is missed. The hardware RX also handles everything that is required to interrupt on an incoming byte of data. The hardware TX implements the transfer of the data in

hardware, meaning that the microprocessor is free to do other tasks while the data is being sent.

Implementing the software code to transmit serial data is easy. First, the serial port must be specified using the follow ing line:

```
#use rs232(baud=57600, xmit=PIN_C4, rcv=PIN_C5)
```

This line implements the hardware TX/RX pair which is Pin C4 and C5. The same line will also implement a software TX/RX pair; for example:

```
#use rs232(baud=57600, xmit=PIN_D4, rcv=PIN_C4, parity=n)
```

This line implements a software TX/RX pair on pins D4 and C4.

There are two basic commands to send data via the serial TX:

```
putc('A');
printf("Data goes here \n\r");
```

There is one basic command to get data via the serial RX:

```
getc();
```

Note: getc() is a blocking function, i.e. the microprocessor will wait until it receives data. This can cause the microprocessor to freeze and must be addressed in the code by using a timed getc (see the U1.c code).

9.3.1.3. Overview of inter-microcontroller Communication

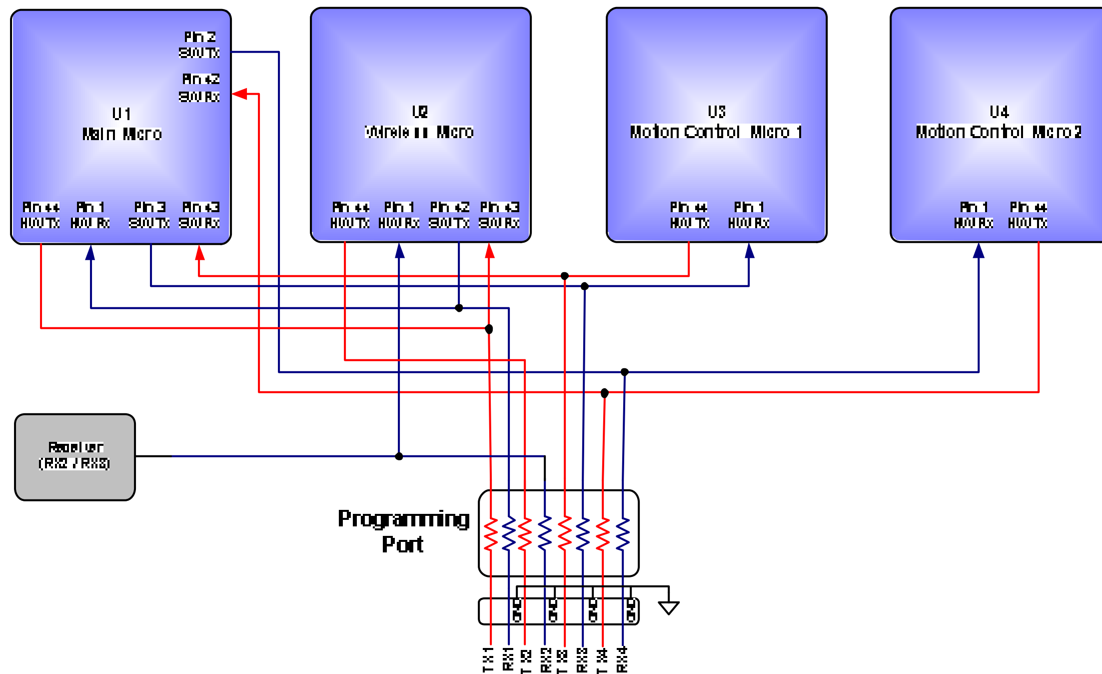


Figure 9.3.1.3– Diagram of the Inter-Microprocessor Communication

Refer to the diagram above. A few things to note:

1. Most of the communication is handled on the hardware TX/RX pairs. The only exceptions to this are the output of U2 to U1 and the outputs of U1 to U3 and U4.

The output of U2 is not on a hardware TX/RX pair because the receiver is using the hardware TX/RX. The receiver needs to interrupt the RX line, which when using a software RX requires implementation in software, while the hardware TX already takes care of interrupting on the RX line.

The outputs of U1 to U3 and U4 are not on hardware RX/TX pairs because the input from U2 uses the hardware TX/RX. The input from U2 needs to interrupt the RX line which as explained above requires implementation in software when using a software RX.

2. There are no resistors that protect against driver-contention between microprocessors. We looked into this issue and determined that it is not necessary to protect against driver contention because each pin of the microprocessor can source 25mA and each pin can also sink 25mA.

3. There are resistors between the serial programming header and the microprocessors that protect against driver-contention. This is required because an external source may source more than the 25mA that the microprocessor can sink.

9.3.2. Debug Ports

There are four ports that connect to the transmit/receive pins of all the four microcontrollers on the digital board. There are two purposes of the debug ports:

1. To program the microcontrollers. For a detailed treatment of this topic see section 9.2.
2. To eavesdrop on the inter-microcontroller communication.

By using a dedicated cable to connect to the serial port of the microcontroller we can get information about the operation of the robot as the following table shows:

Port 1: Provides the output of the main microcontroller. Normally the output of the microcontroller does not give any data on its TX line unless there is some debugging code outputting some debug information.

Port 2: Wireless packet data is sent periodically to the main micro on the TX line of the wireless micro. Thus if we connect the cable to the port 2, we can see what the actual packets being sent to the main microcontroller. However in HyperTerminal the data is displayed in ASCII which complicates translation into robot vectors we expect.

Port 3: The motion control microcontroller is programmed to send the values of the desired counts its been given and the actual value of the counts from its motor encoders. This gives an insight into both the operation of the p-controller, and the whether the correct values have been forwarded to it from the main microcontroller.

Port 4: Port 4 is connected to the output of the second motion control microcontroller. Thus the behavior of this port is exactly the same as the other motion control microcontroller (Port 3).

10. Appendix

A. NETWORKING

Network Communication

1 Relevant Files

Server.cpp	: Server code
Server.h	
Client.cpp	: Client code
Client.h	
Socket.cpp	: Core networking routines
Socket.h	
CommType.h	: node type definition

2 Architecture:

This year's system is a TCP/IP-based server-client architecture based on Winsock (the standard Window's network library). In our setup, nodes have three possible designations: JUST_SEND, JUST_RECEIVE, and SEND_RECEIVE. JUST_SEND is used for server nodes, JUST_RECEIVE, for clients, and SEND_RECIEVE designates a node which can act as both.

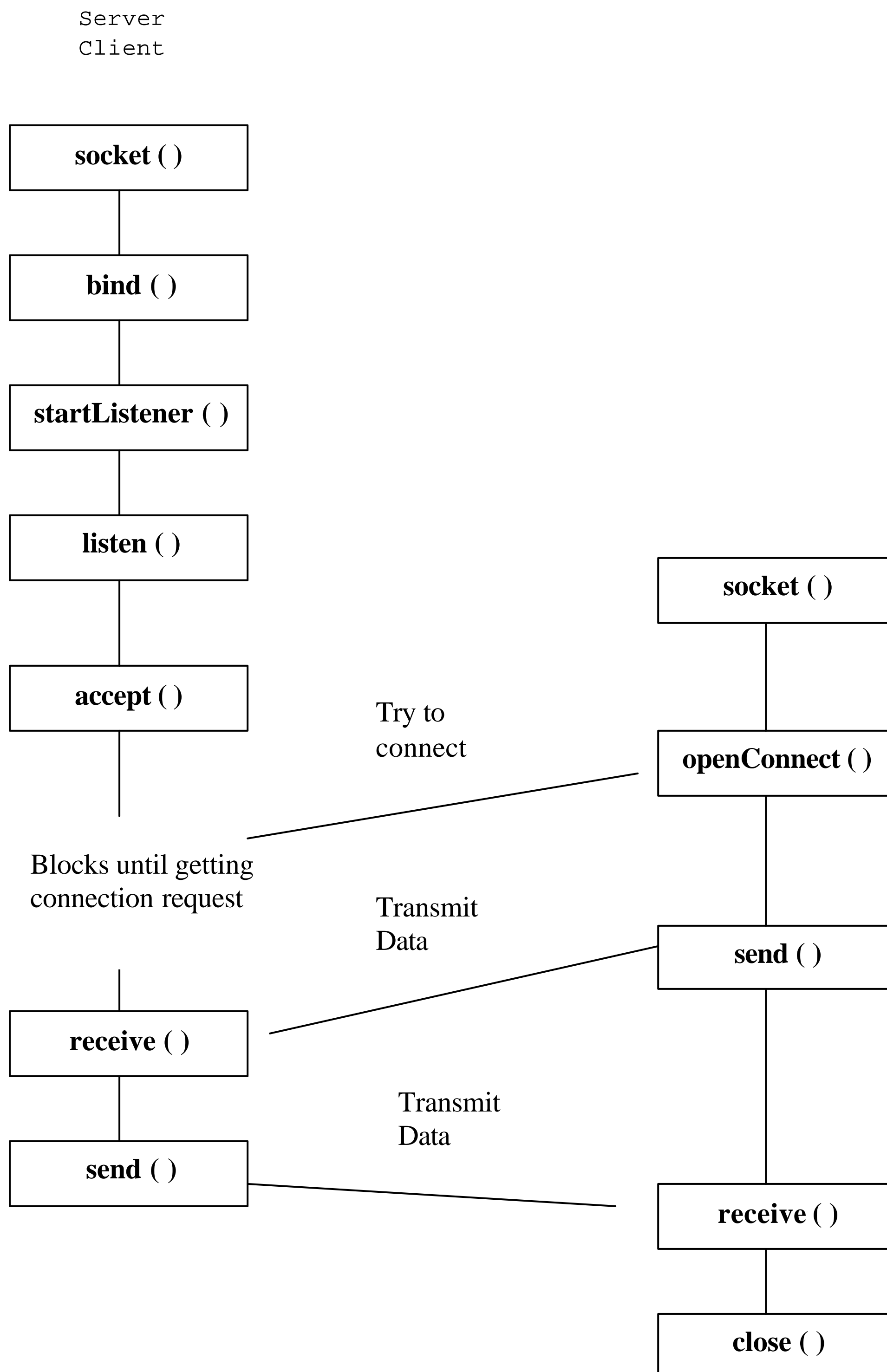
3 Server Side

To instantiate a server, one must specify the type, the address and port number the server is to be established with. Once instantiated, the server sets up a listening socket on the specified port and creates a data structure for managing connections. The server then starts a listener running on a separate thread so it is free to perform other tasks while also monitoring any connection requests.

When a client requests a connection from the server, a handshake is initiated (standard TCP/IP) to confirm the connection.

4 Client Side

In order to establish a connection with the server, the client is first instantiated using the appropriate IP address and port number and the connection() method is called. The client type and server type must match for a connection to open successfully. E.g. if the server type and/or client type is not recognized or not an allowable configuration (senseless to have two JUST_SEND entities connect to each other), the connection will not be created. If the connection succeeds, the client and server can communicate using the send() and receive() methods.


Server
Client
socket ( )
bind ( )
startListener ( )
listen ( )
accept ( )
Blocks until getting
connection request
receive ( )
send ( )
socket ( )
openConnect ( )
send ( )
receive ( )
close ( )
Try to
connect
Transmit
Data
Transmit
Data

6 The detail structure of each class is defined as follow:
CommType is a structure that defines the communication mode the client or server can take on. These modes include: duplex mode defined as SEND_RECEIVE, send-only mode defined as JUST_SEND, and receive-only mode defined as JUST_RECEIVE.

Class Socket: Contains methods to set up a socket connection; open/close connection; send and receive data over the connection. It is uses the Winsock library and provides the basic calls which are used by both client and server classes.
In order to use this class, user should add ws2_32.lib to the link, this can be done by:
In Visual C++: Go to Project->Settings. Click on the Link tab and within Object/library modules, add the statement: "ws2_32.lib"

Method Details:
Socket();
Default constructor. Creates a new Socket object

bool openConnection(char* hostname, int port);
The method opens a connection when given a hostname and port number.

Input parameters:
hostname: The hostname of the server
port: port number of the server

Return value:
returns 1 if successful, 0 if failed

bool openListener(char* hostname, int port);
This method opens a listening socket using the given hostname and port number; it then binds the new socket and starts listening for an incoming connection. Listeners are used to listen for incoming connection requests.

Input parameters:
hostname: The hostname of the server
port: port number of the server

Return value:
returns true if successful, false if failed

bool isOpen();
This method checks if a connection is currently open

Return value:
Return true if connected, false if not connected

void close();
This method closes a connection

bool sendData(char* message, int size);

This method sends data over a socket connection. It returns true if successful and returns false if:
1) the size of data sent is not the same as the size of actual data
2) the connection is already closed

Input parameter:
message: pointer to the message that would like to send
size: size of the message

Return value:
Return true if successful, false if failed

bool receiveData(char* message, int size);

This method is used to receive data over a socket connection. It returns true if the data is received correctly, and false if:
1) the size of data received is not the same as the size of actual data
2) the connection is already closed

Input parameter:
message: pointer to buffer to the message that would like to receive
size: size of the message

Return value:
Return true if success, false if fail

Socket* acceptConnection();

This method accepts an incoming connection request

Return value:
Return a newly connected socket

Class Server: This class includes everything needed to create and use a Server. Methods include setting up a connection; listening and accepting incoming socket connections; sending and receiving data from different client types. The methods are built from calls to the socket class.

Method Details:

Server (CommType serverType, int MaxSend, int MaxReceive);

Constructor of the server class. It takes in the type of server. Possible types are defined in CommType.h and are as follows:
JUST_SEND: server that only sends data (e.g. Vision)
JUST_RECEIVE: server that only receives data
SEND_RECEIVE: server that both sends and receives data

Servers accept a certain max number of clients that can send and/or receive data

Input parameter:
serverType: Type of the server
MaxSend: store the max number of clients of type JUST_RECEIVE and SEND_RECEIVE
MaxReceive: store the max number of clients of type JUST_SEND and SEND_RECEIVE

bool startListener(char* hostname, int port);

This method generates a thread to listen for connection requests. A connection will be accepted if given the client satisfies both of the following conditions:

1) The server and client types are compatible
2) There are still available slots for the incoming client type.

It also checks if there is already a connection set up for the client. If there is, the server ignores the new connection request and returns false.

Input parameter:
Hostname: hostname of the server would like to connect to
Post: port number of the server client would like to connect to

Return value:
Return true if successful, false if failed

void listen();

This method is called by startListener() and it accepts connections from the listener socket and assigns them to the proper client index by calling the appropriate private methods –addSender(); addReceiver() and addSenderReceiver():

1) For JUST_SEND client type, calls addSender()
2) For JUST_RECEIVE client type, calls addReceiver()
3) For SEND_RECEIVE client type, calls addSenderReceiver()

void sendAll(void* message, int size);

This method sends data to all receiving clients, which includes JUST_RECEIVE and SEND_RECEIVE

Input parameter:
message: pointer to buffer to the message that would like to receive
size: size of the message

bool send(int clientIndex, void* message, int size);

This method sends data to a particular client who has the given clientIndex value

Input parameter:
clientIndex: use to identity the client. It is the index of the private variable - sendIndex[] array

message: pointer to buffer to the message that would like to send
size: size of the message
Return value:
Return true if success, false if failed

bool receive(int clientIndex,void* message,int size);
This method receives data from a particular client who has the given clientIndex value

Input parameter:
clientIndex: use to identity the client. It is the index of the private variable - receiveIndex[] array
message: pointer to buffer to the message that would like to receive
size: size of the message

Return value:
Return true if successful, false if failed

void setSendBuffer(int bufSize)
This method set the size of the sender's buffer. The buffersize for Robocup is not much of an issue since the typical amount of data which is sent in a packet is small, and considerably smaller than the default kernel buffersize of 8192.

Input parameter:
bufSize: size of the buffer

void setReceiveBuffer(int bufSize)
This method set the size of the receiver's buffer
Input parameter:
bufSize: size of the buffer

Class Client: This class includes everything needed to create and use a Client. Methods include setting up a connection; checking connection status; sending and receiving data. The methods are built from calls to the socket class.

Method details:
Client(CommType clientType):
The client constructor uses a default host address of 127.0.0.1 (loopback/localhost address) and port number 4545.

Input parameters:
clientType: defines the client's communication mode by either SEND_RECEIVE, JUST_SEND or JUST_RECEIVE.

Client(CommType clientType, const char* hostname, int port):
This is a constructor to create Client with user-specified communication type and user-defined server address and port number.

Input parameter:
clientType: defines the client' s communication mode by SEND_RECEIVE, JUST_SEND and JUST_RECEIVE
hostname: defines the address to connect to
port: defines the port number to connect to

bool openConnection()
This method opens a connection to communicate with the server having address defined as "Client->hostname" and port number as "Client->port".

Return Value:
Returns true if connection is opened successfully and false if the client type is rejected by the server or connection fails.

bool openConnection(const char* hostname, int port)
This method opens a connection to communicate with the specified server.
Input parameter:
hostname: defines the server' s address
port: defines the server's port number
Return Value:
Returns true if successful and false if open connection fails.

bool isOpen()
This method checks if the client has opened a connection with a server.

Return Value:
Returns true if successful and false if open connection fails.

void close()
This method closes a client connection.
Input parameter: None
Return Value: None

bool send(void* message, int size)
This method sends a message of the specified size to the server. It also performs a check to ensure server is not in JUST_SEND mode.

Input parameter:
message: pointer to the message to be sent
size: length of the message
Return Value:
Return true if data is sent and false if the server is defined to be in the send-only mode.

bool receive(void* message, int size)

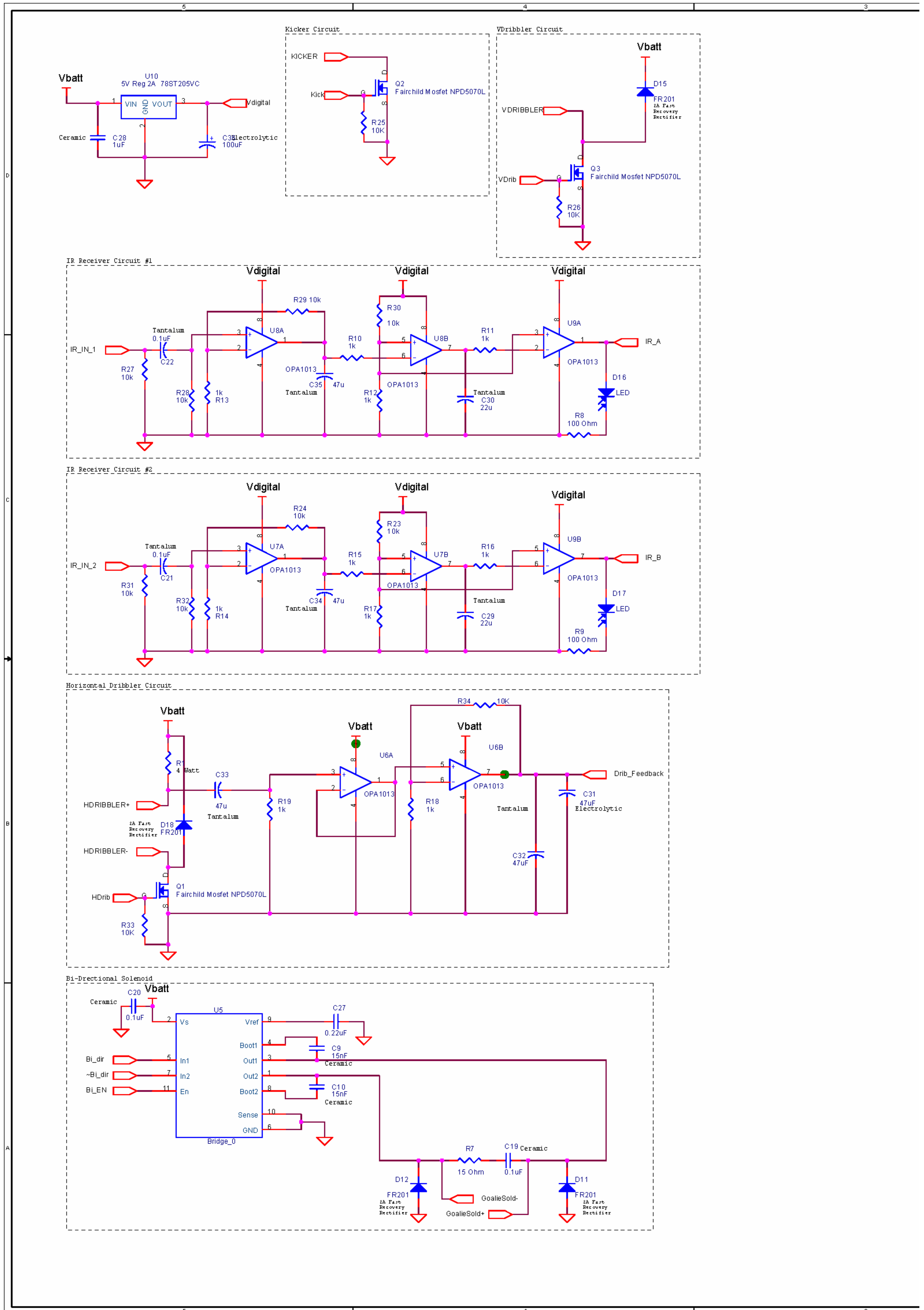
This method receives a message of the specified size from the server. It also performs a check to ensure the server is not in the JUST_RECEIVE mode.
Input parameter:
message: pointer to the message to be received.
size: length of the message
Return Value:
Return true if data is received and false if the server is defined to be in the JUST_RECEIVE mode.

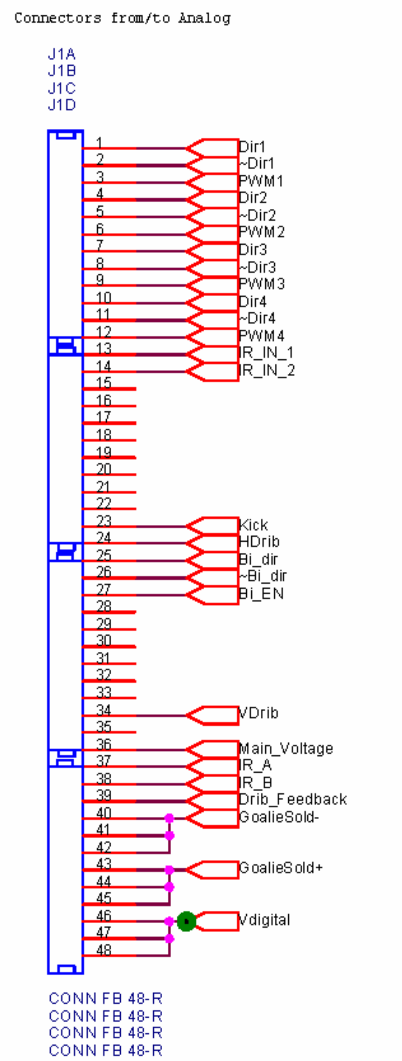
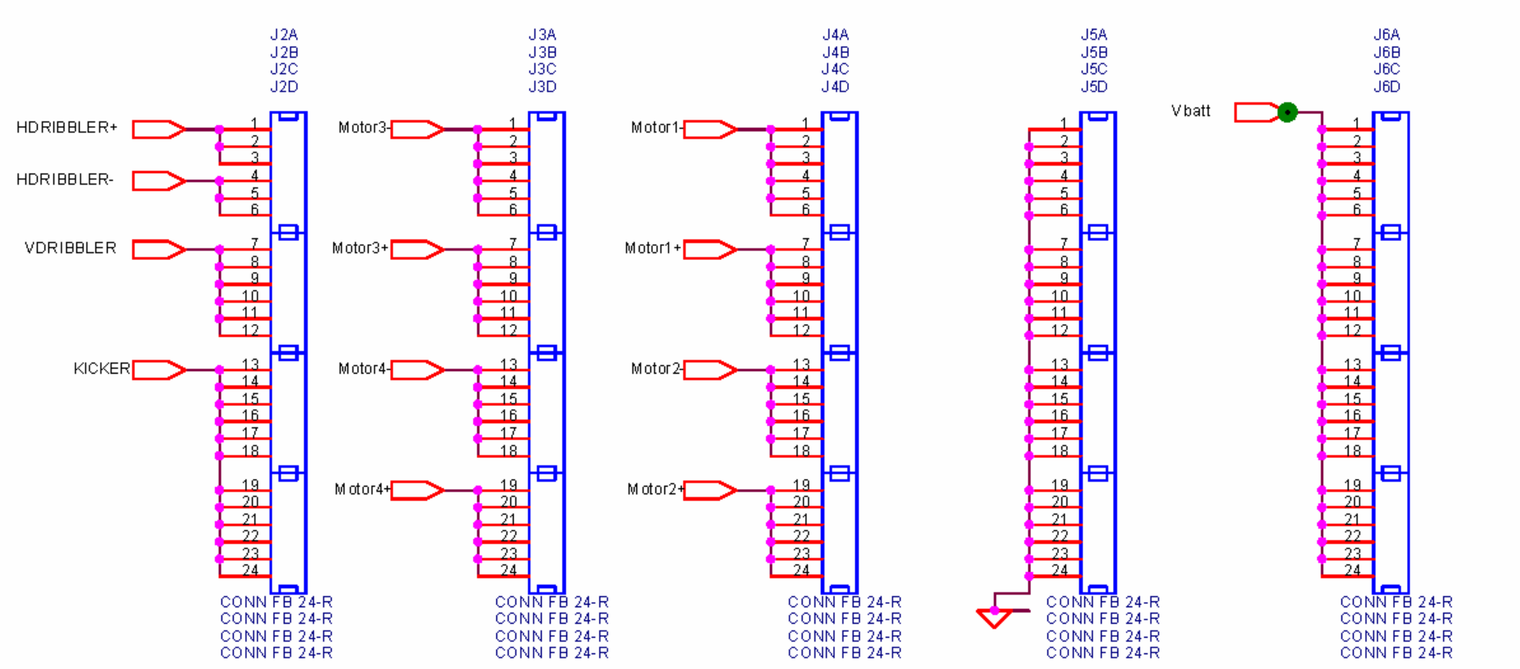
7 Constraints

Networking is one of many pieces whose combined time must be under 16.6ms so that our system is able to run at 60Hz. Because our system is currently running on a 100Mb/s local Ethernet there negligible timing delays once our packets get to the wire. It is important to have the Nagle algorithm switched off, since the Nagle algorithm clusters packets and prevents them from being sent immediately (it does so as to avoid the "silly window" syndrome which is more of a problem for networks with higher overall utilization). Due to the real-time qualities of our domain, it is critical that we send packets as soon as possible, thus the Nagle algorithm should stay off.

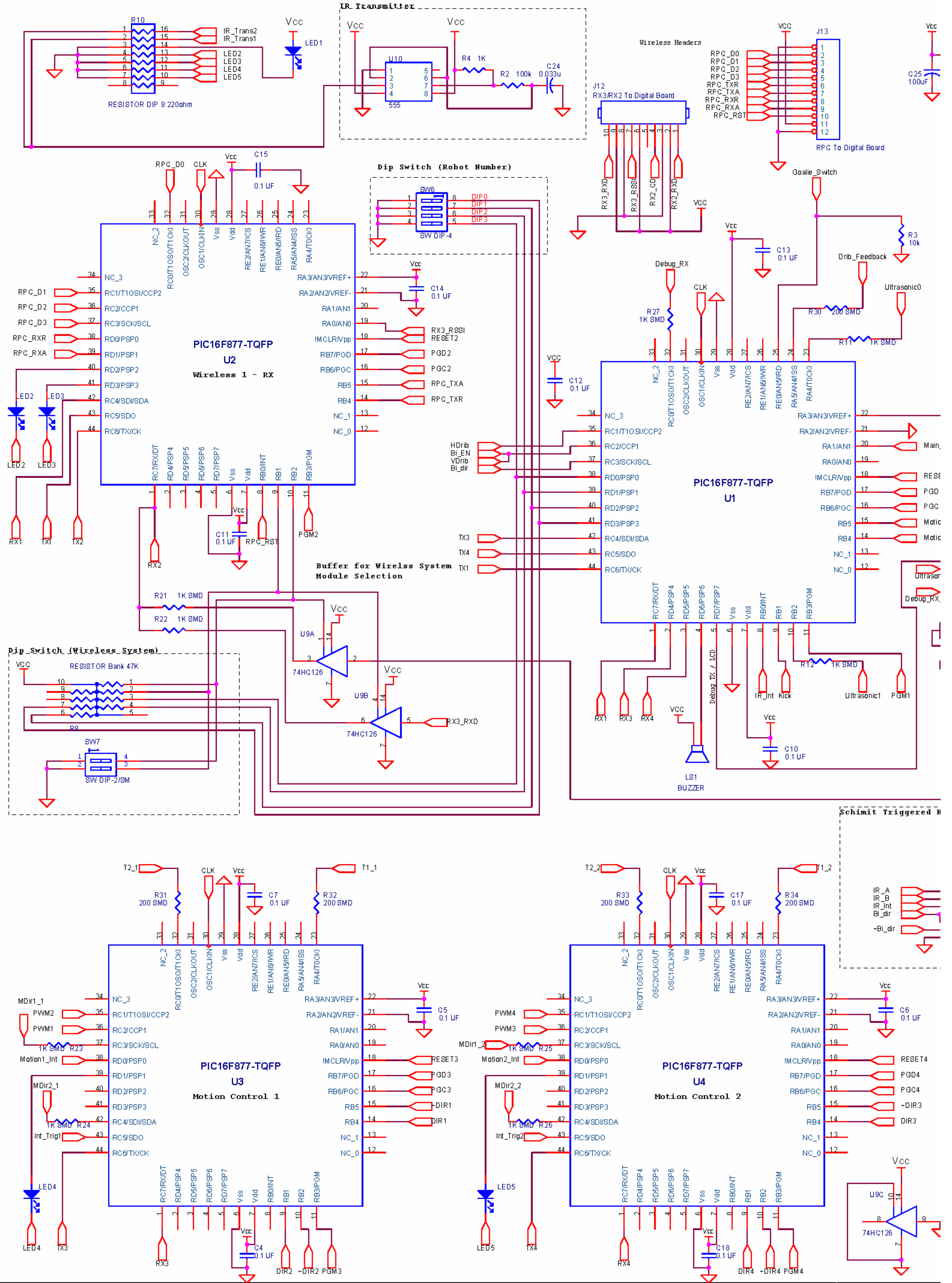
B. SCHEMATICS

ANALOG

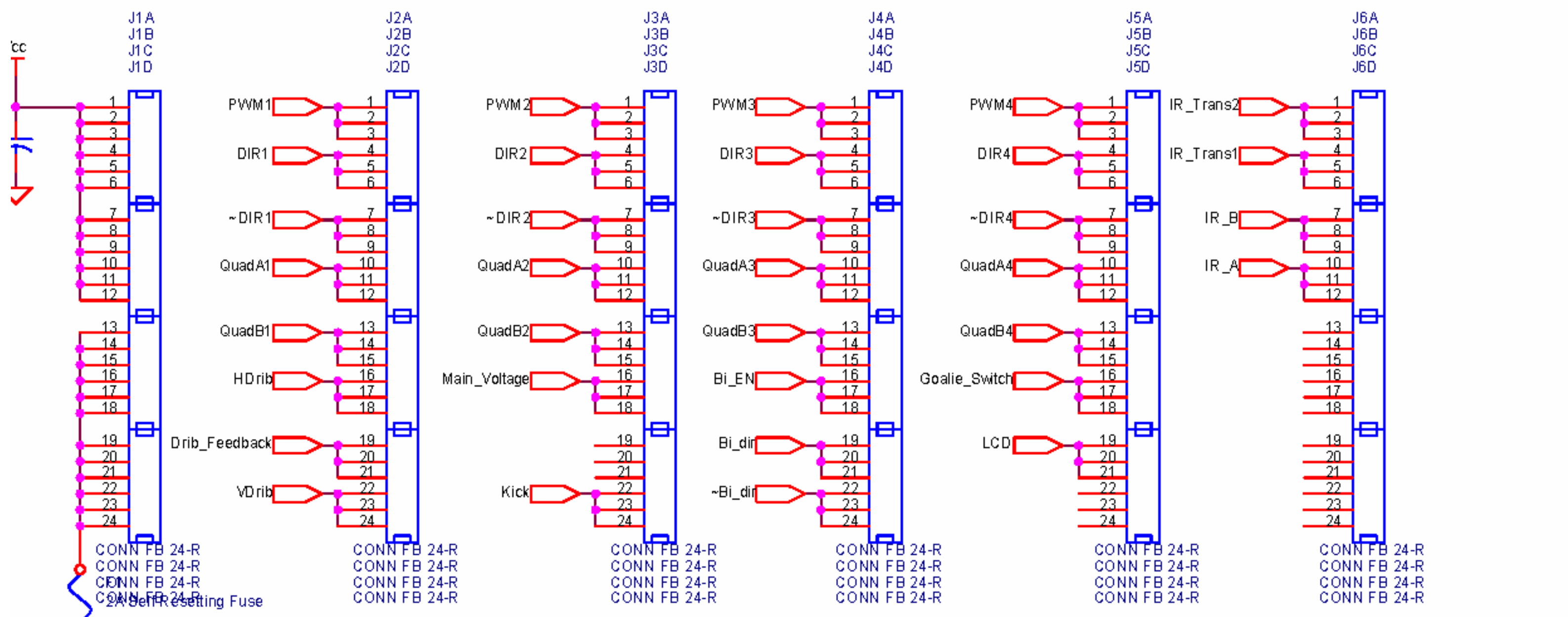


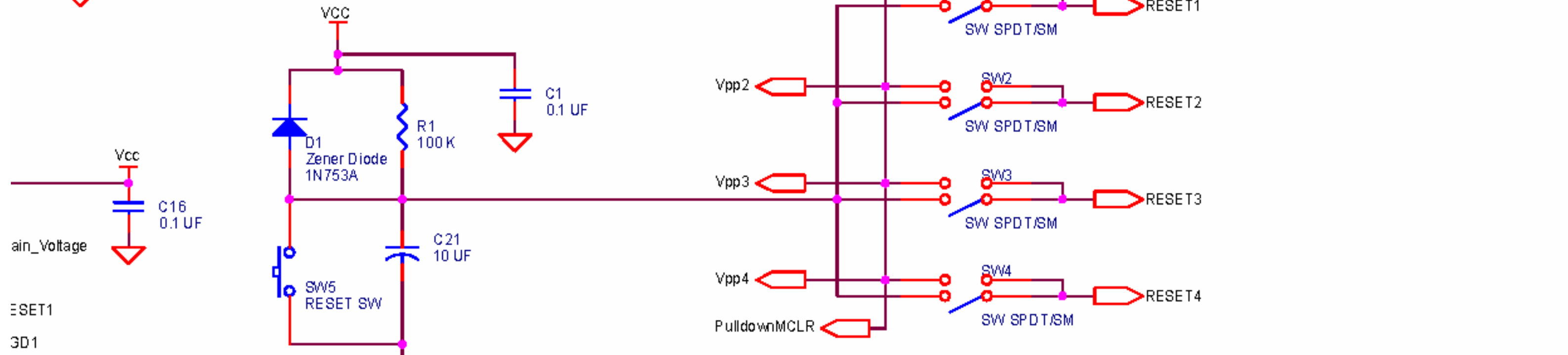



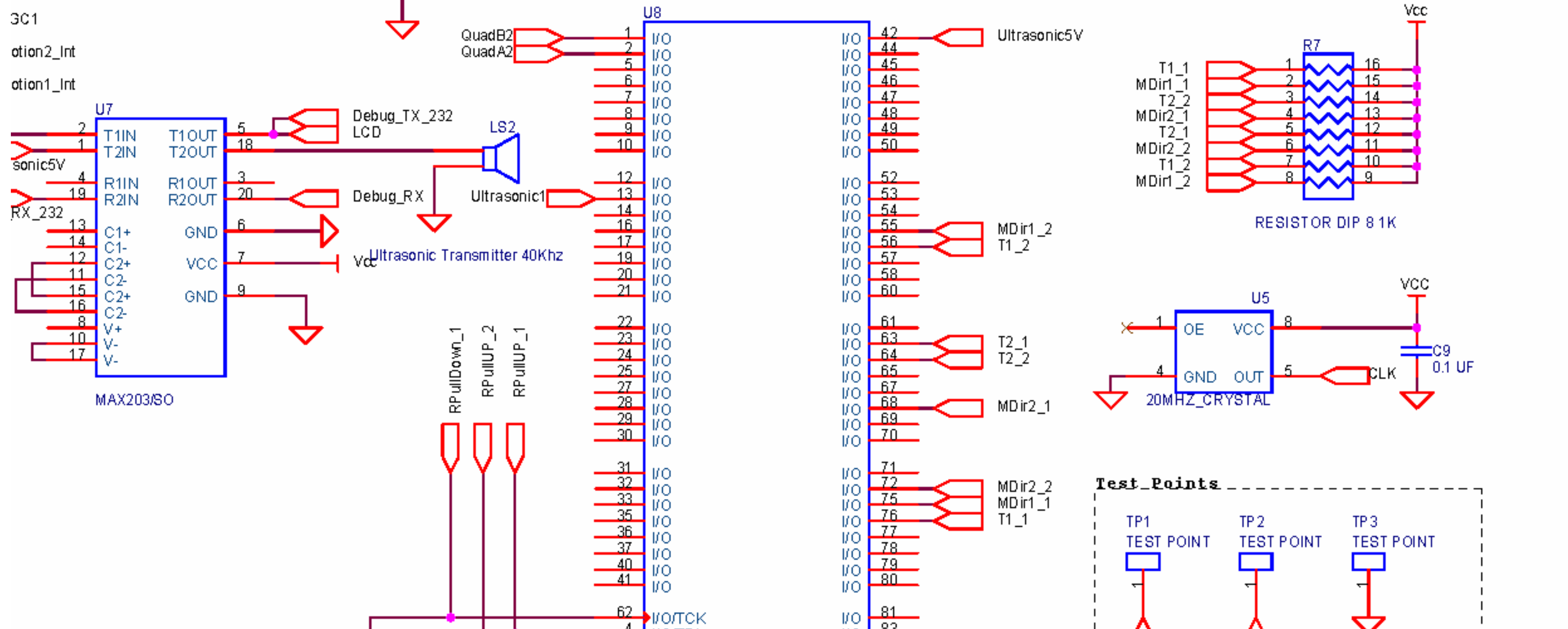
DIGITAL

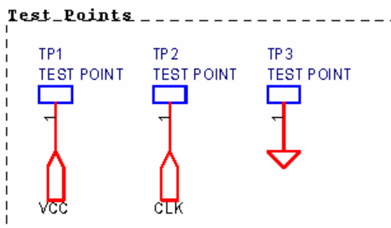


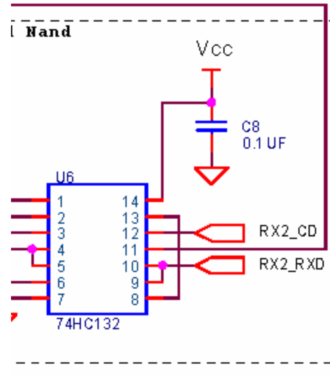
onnectors from/to Digital

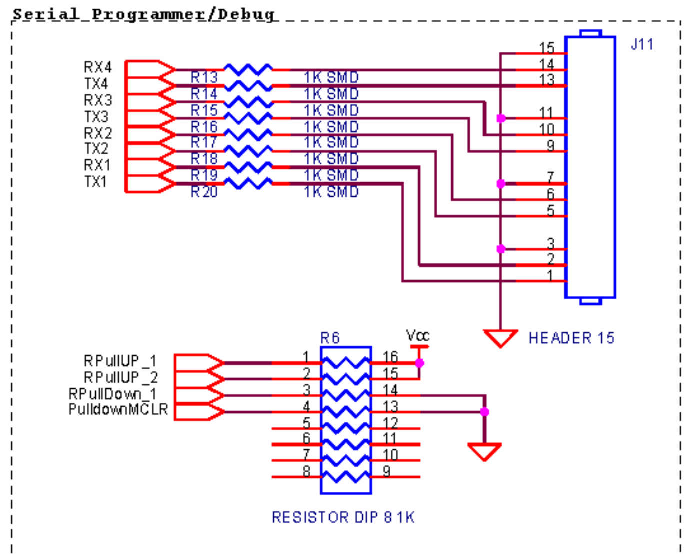












Vcc

J14

CONN RCPT 5x2

Altera FPGA Programming Port

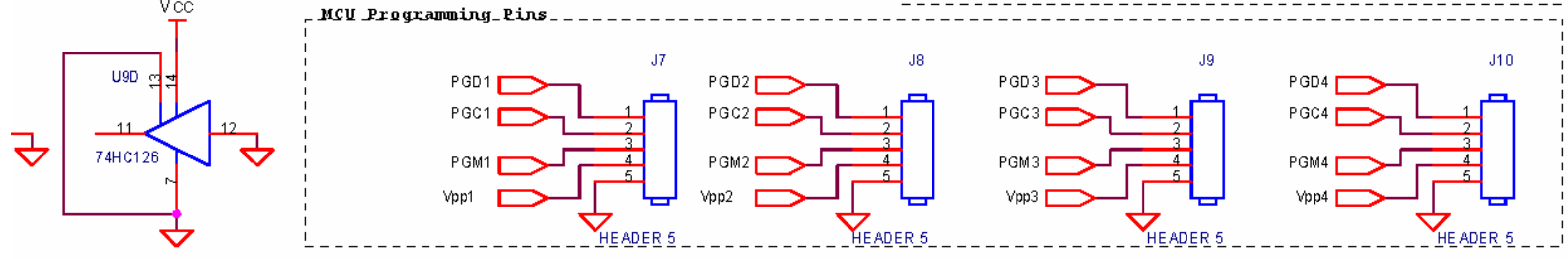
Debug Port

Debug_TX_232

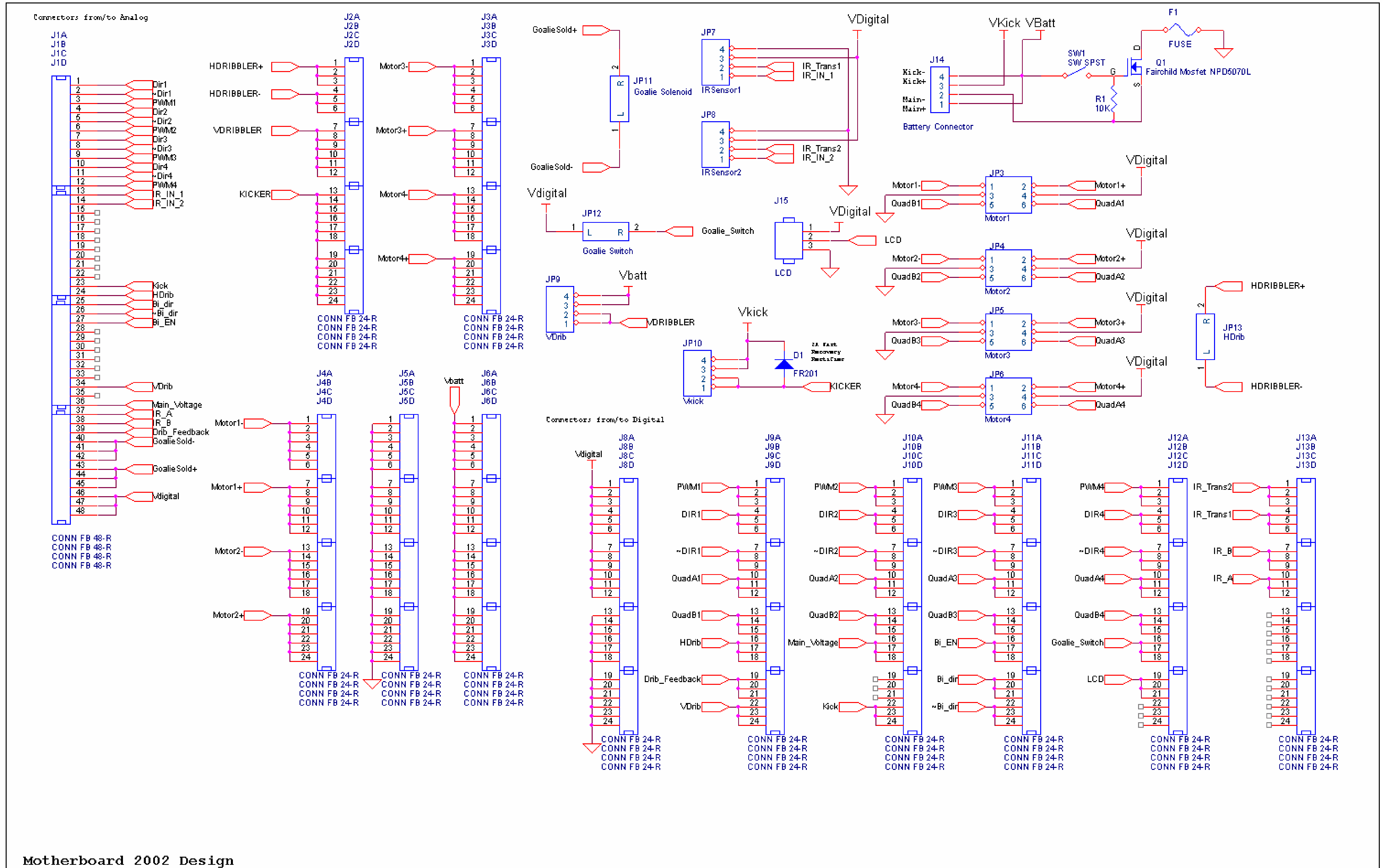
P1

Debug_RX_232

CONN RCPT 5x2

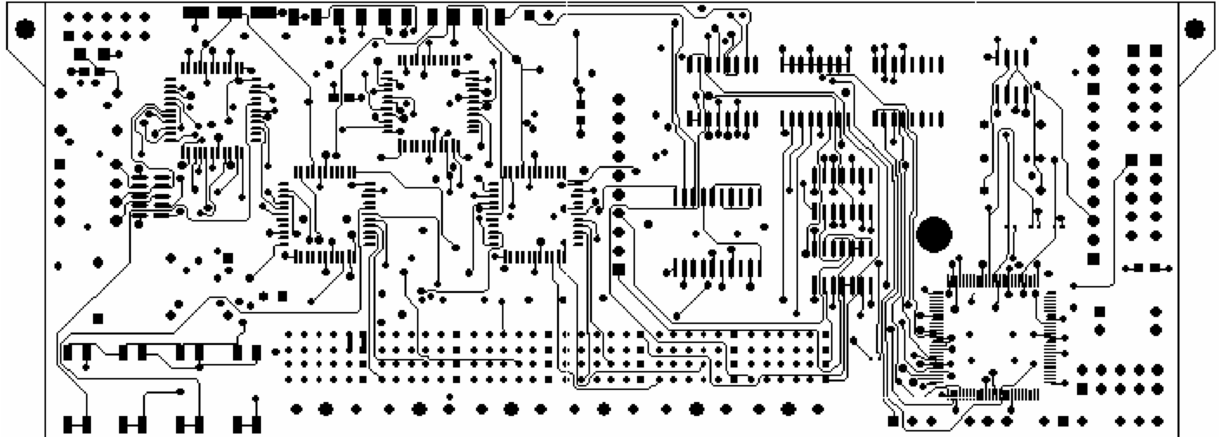


MOTHERBOARD

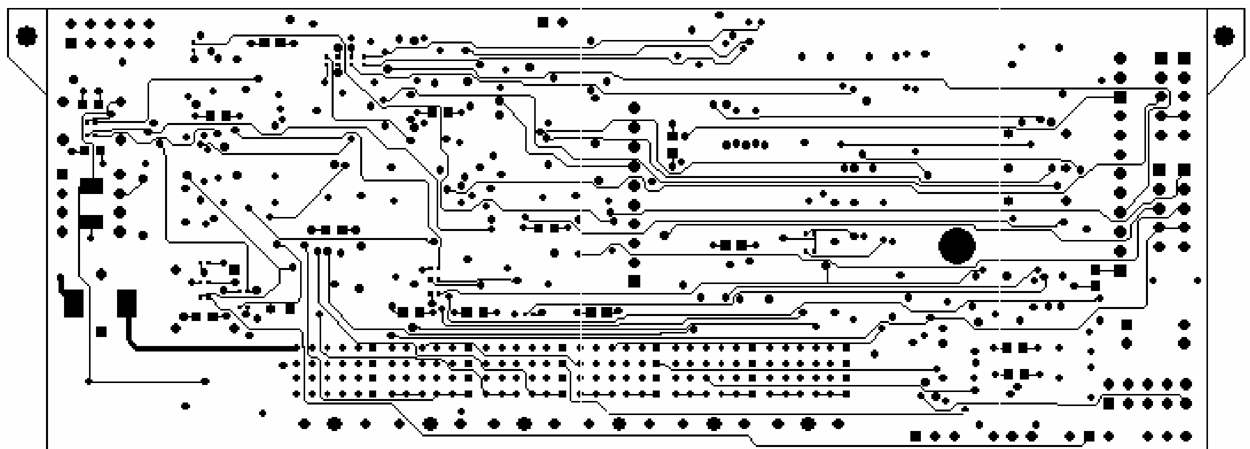


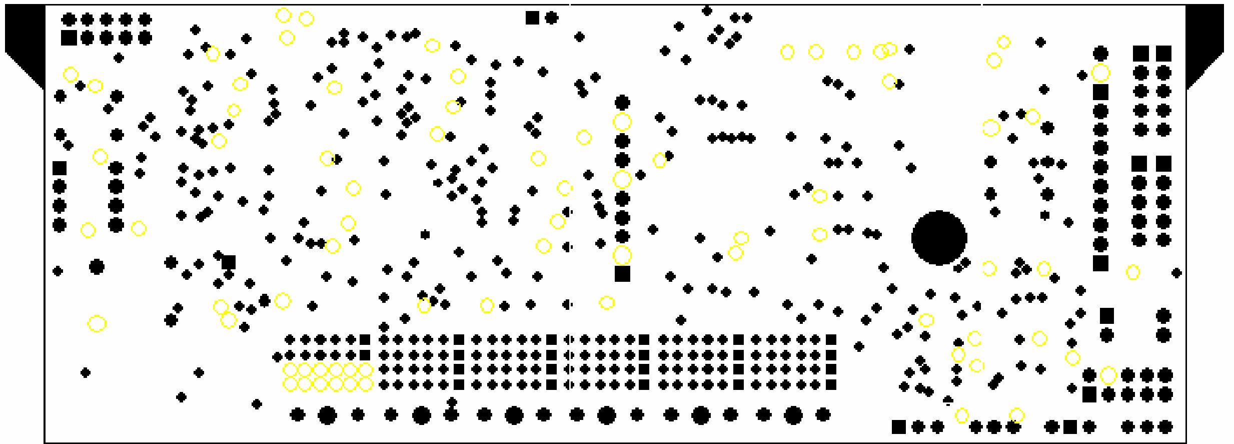
C. BOARD LAYOUT

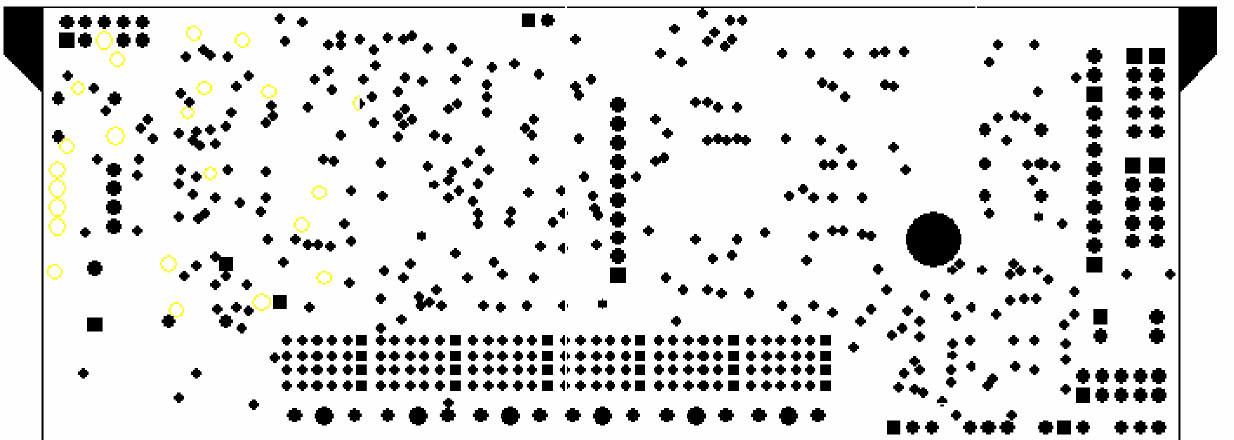
Layer 1 - TOP

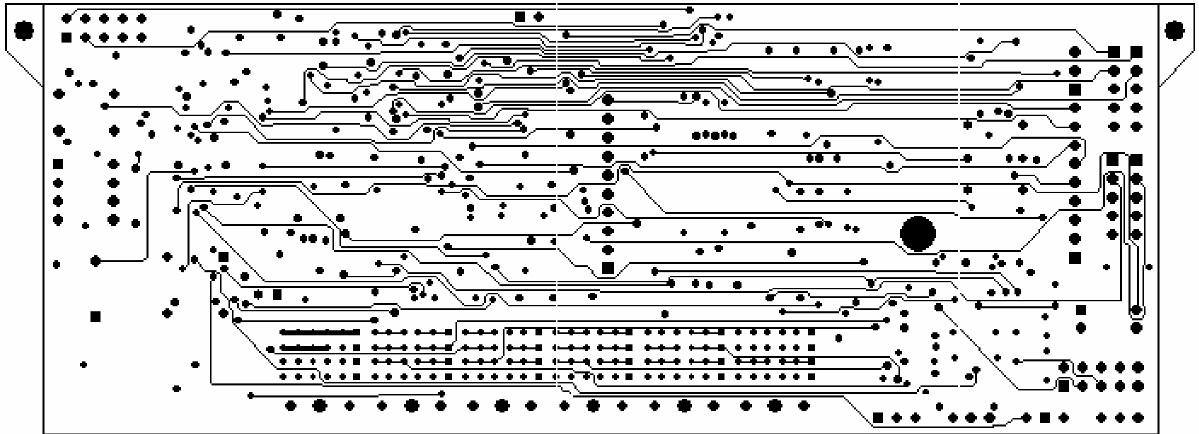


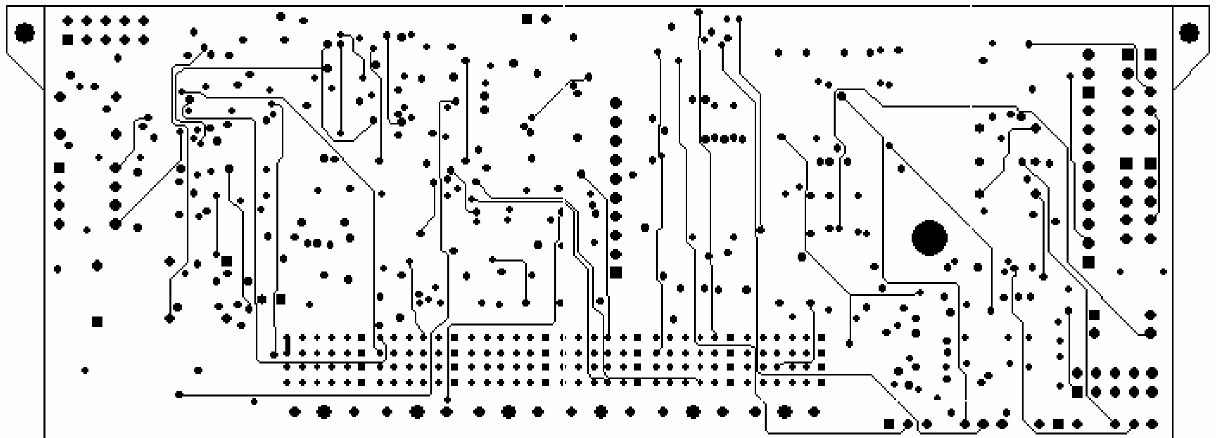
Layer 2 - Bottom

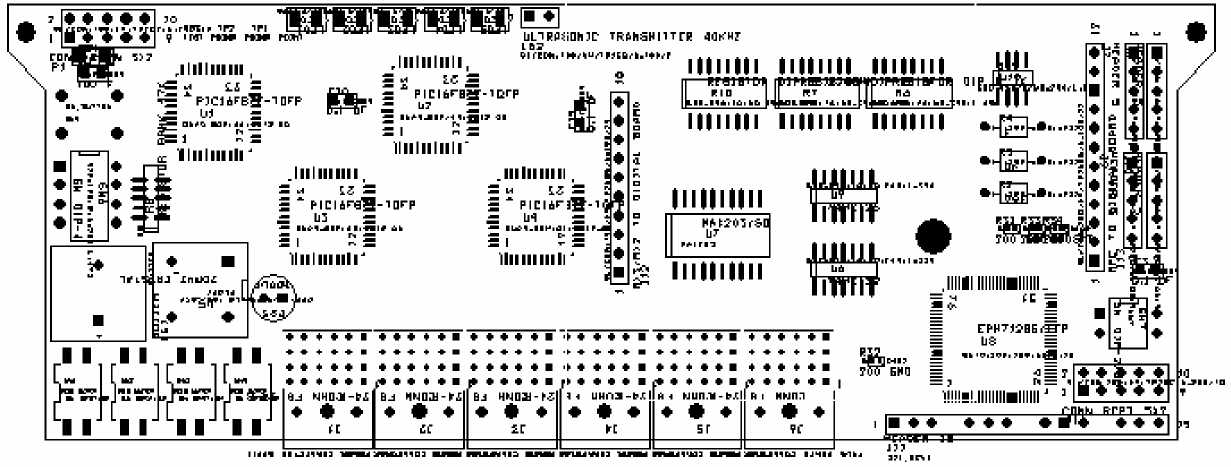


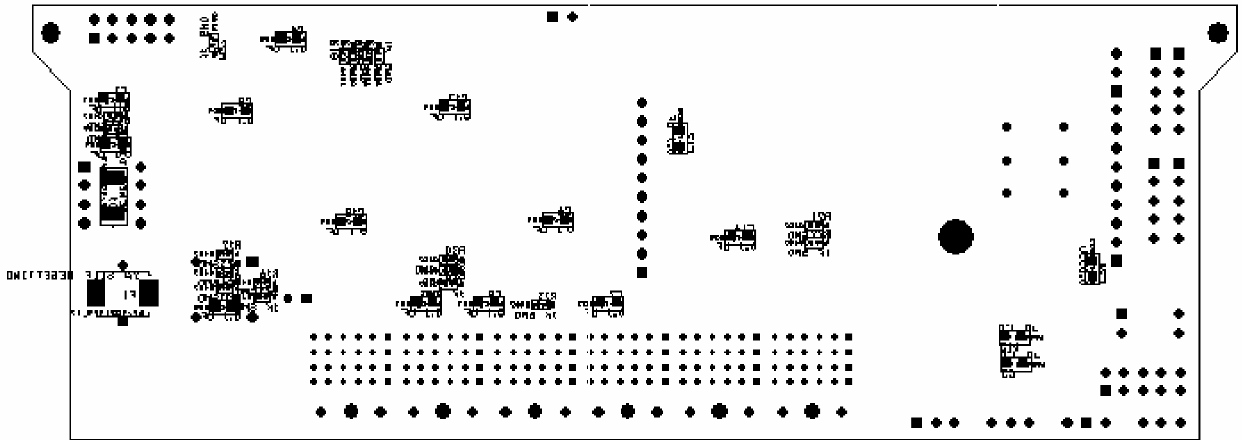

Layer 3 - Power


Layer 4 - GND

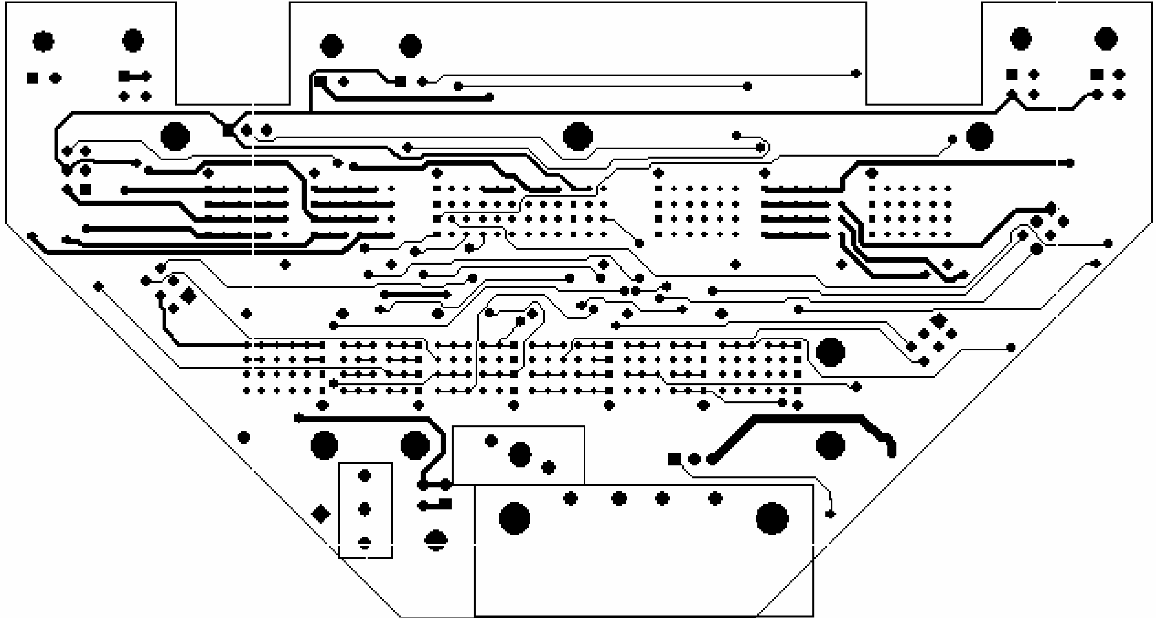

Layer 5 - INNER1


Layer 6 - INNER2

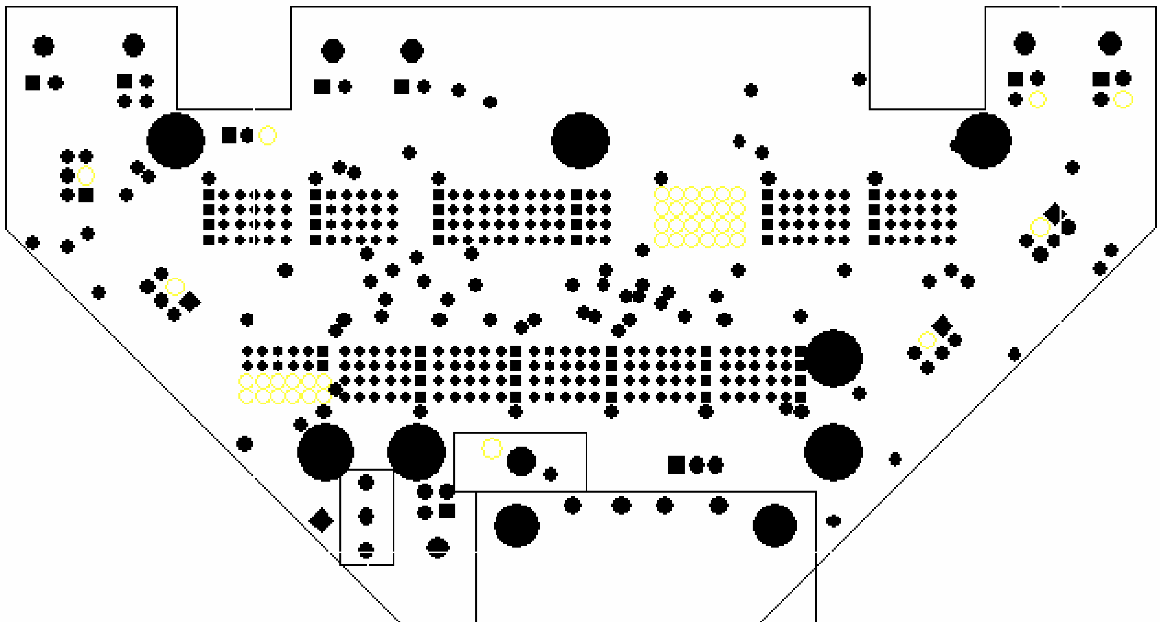

ULTRASONIC TRANSMITTER 40KHZ
MAX203/SO



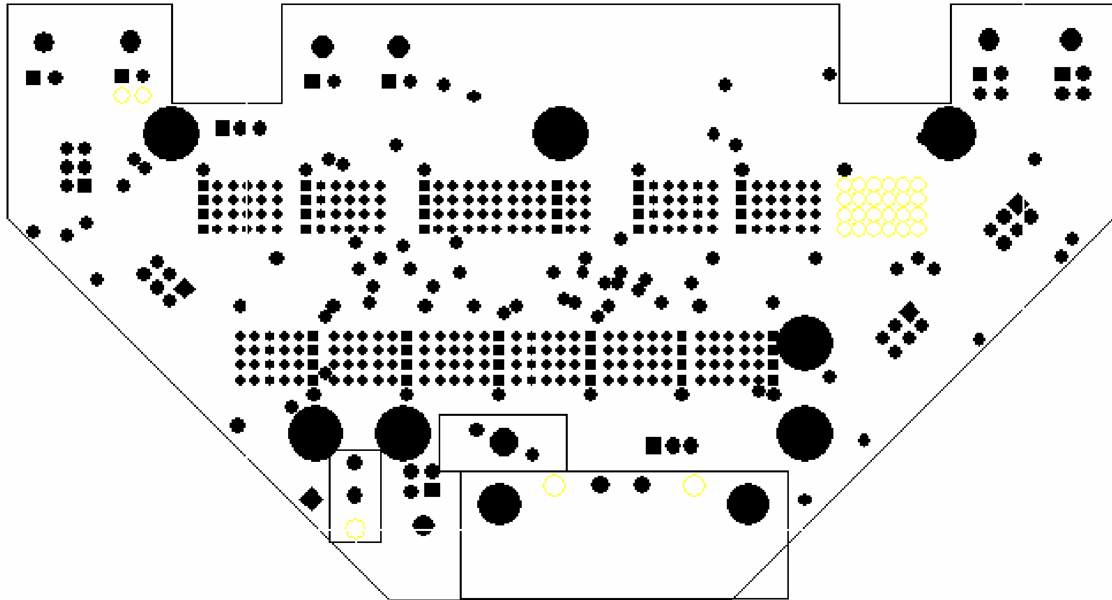
LAYER 1 -- TOP LAYER



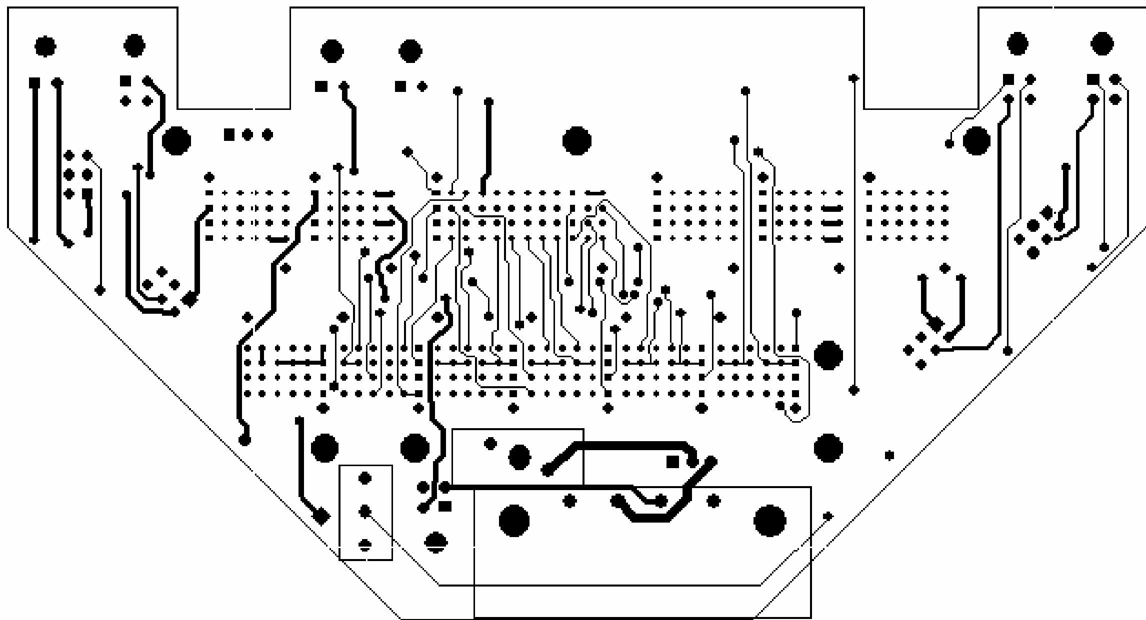
LAYER 2 -- GND LAYER

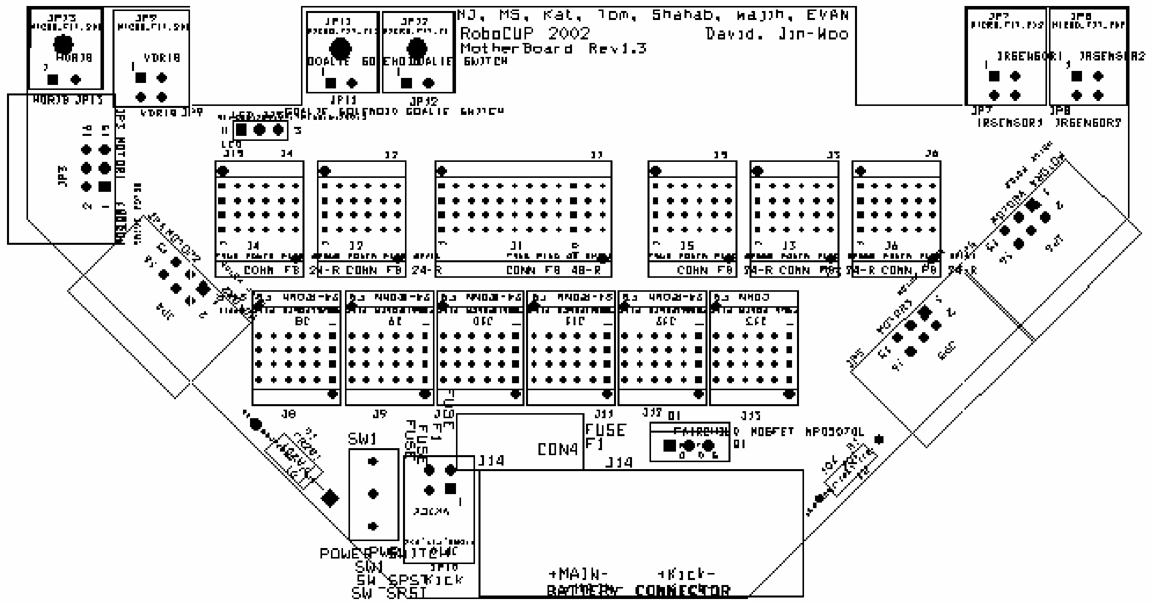


LAYER 3 -- PWR LAYER

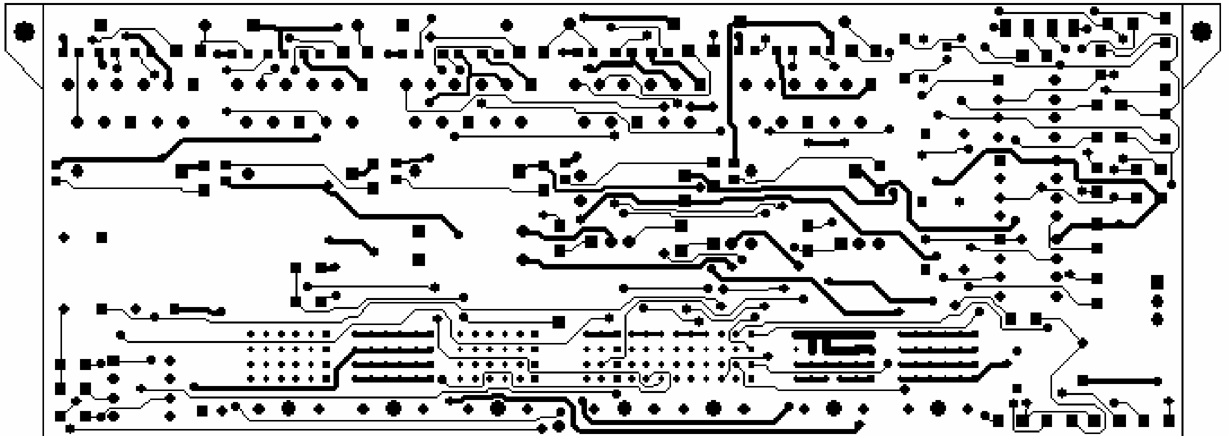


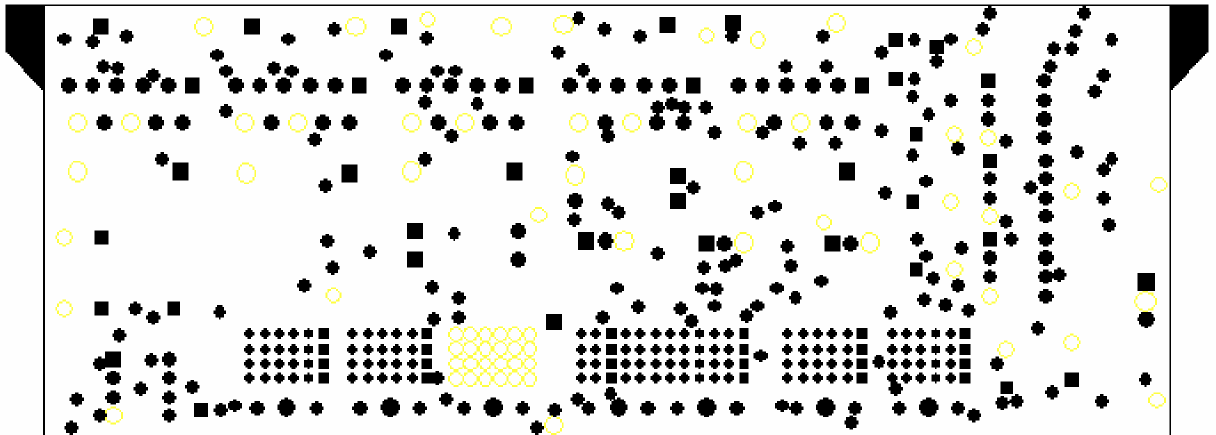
LAYER 4 -- BOTTOM LAYER

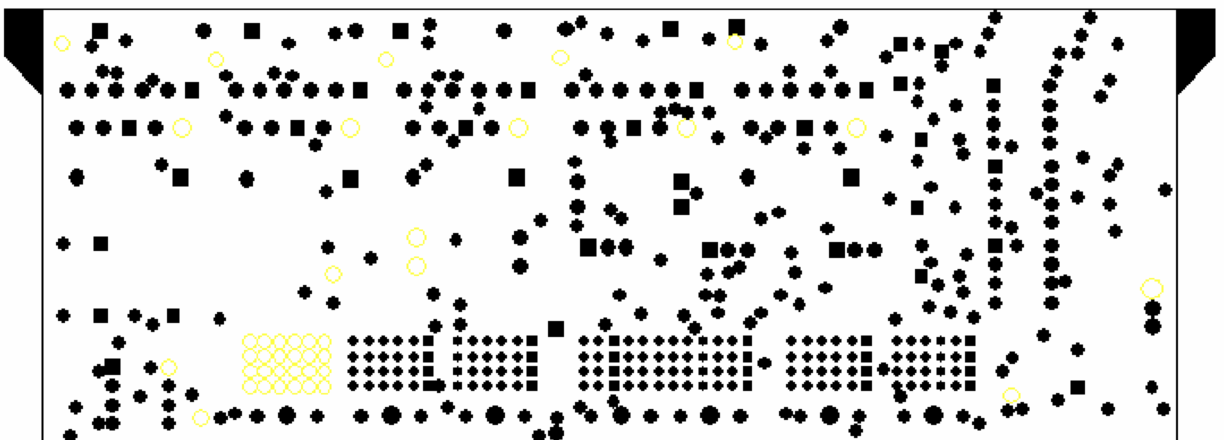




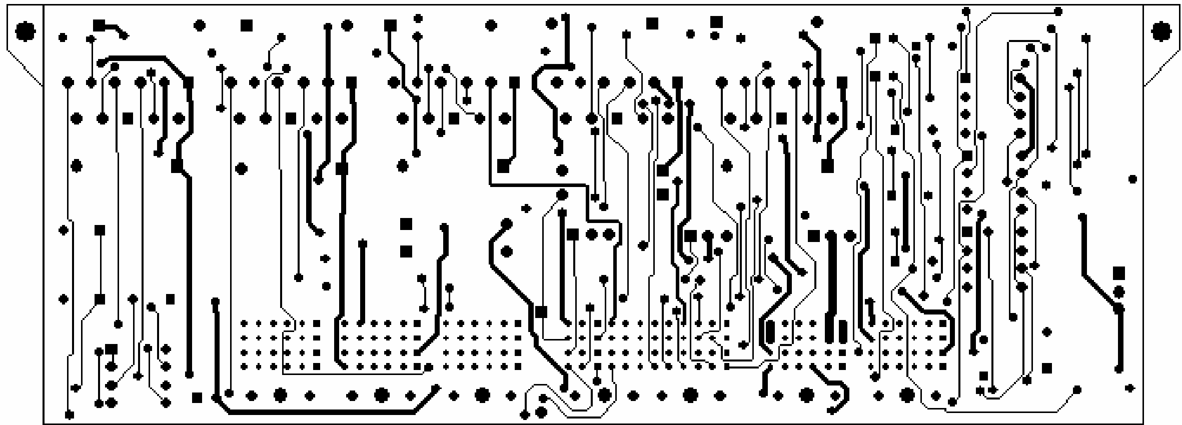
Layer 1 -- TOP

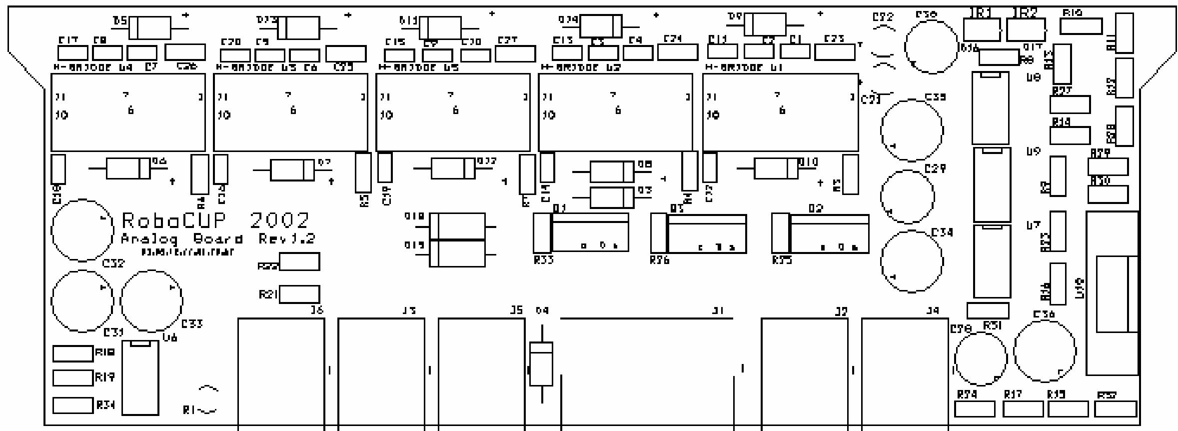


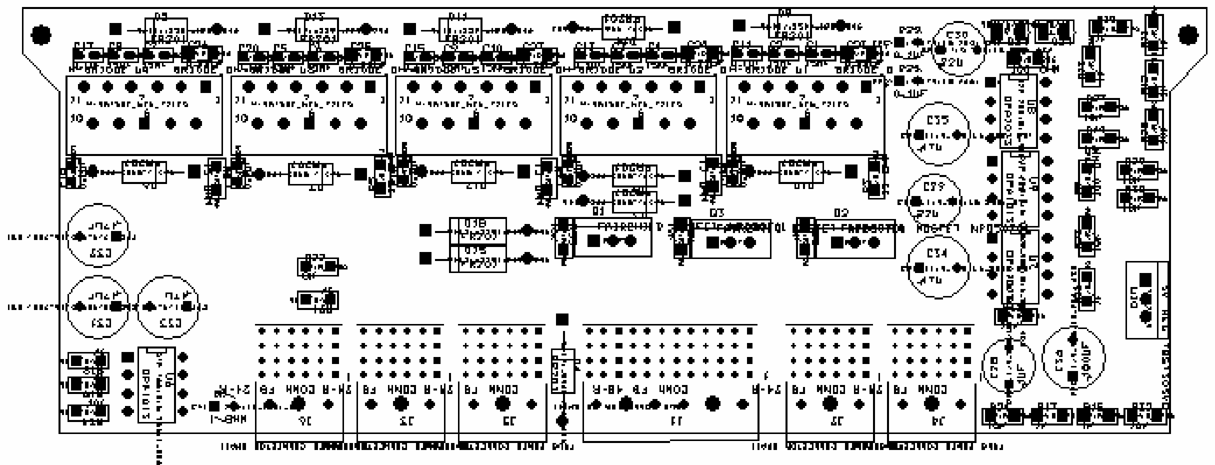

Layer 2 -- GND


Layer 3 -- POWER

Layer 4 -- Bottom

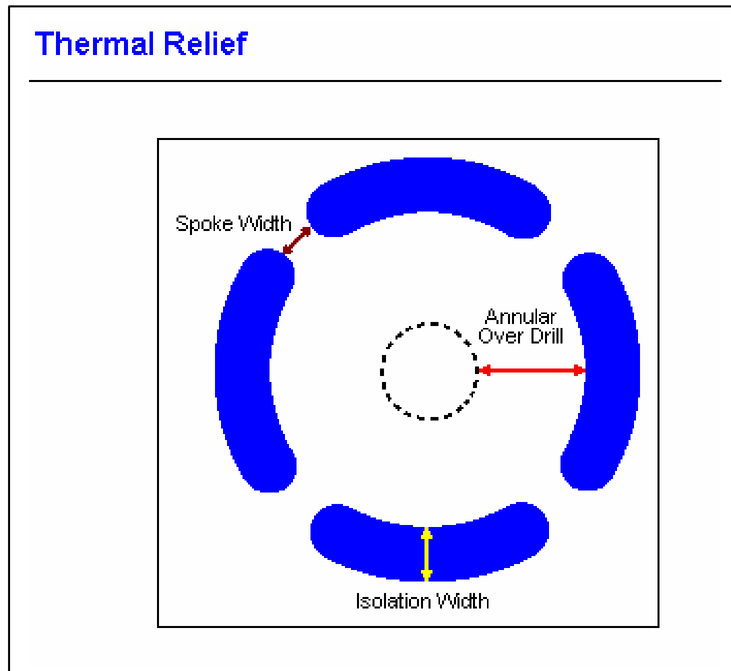






Explanation of Plane Layers:

The plane layers (PWR and GND) are shown as negatives, meaning that everywhere where there is ink there will not be copper and everywhere where this is no ink there will be copper. The light yellow circles on these layers are thermal reliefs. Thermal reliefs are placed around pins that are connected to the layer. They allow the pin to be connected to the rest of the layer while leaving enough space for expansion of the layer due to heat generated by current flowing to the pin.



D. MICROCONTROLLER: REASONS FOR CHOSING PIC16F877

Reasons for changing to the PIC16F877

Atmel microcontroller was chosen for its flexibility, processing power and the number of I/O pins. It had many attractive features that included the following.

- High-performance 32-bit RISC Architecture
- Fully-programmable External Bus Interface (EBI)
- 8 Chip Selects
- Software Programmable 8/16-bit External data bus
- 8-level Priority, Individually mask-able, Vectored Interrupt Controller
- 58 Programmable I/O Lines
- 6-channel 16-bit Timer/Counter
- 6 External Clock Inputs and 2 Multi-purpose I/O Pins per Channel
- 3 USARTs
- 8-channel 10-bit ADC
- 2-channel 10-bit DAC
- 8-channel Peripheral Data Controller for USARTs and SPIs
- IEEE 1149.1 JTAG Boundary-scan on all Digital Pins

However there were some reasons due to which it was abandoned for the favor of PIC microcontroller. These reasons are,

1. The supporting documentation for the Atmel microcontroller was not very good and there is no documentation for the libraries with the development board. There are many functions provided with the development board that can be used with the development board but the only way to understand how a function works is to see from the library code itself or to interpret from an example (some examples are also provided in the libraries).
2. The development environment used (Multi 2000) had some version conflicts with the library version.
3. The debugger had some bugs which caused it to behave randomly on many occasions.

E. MEETING MINUTES TEMPLATE

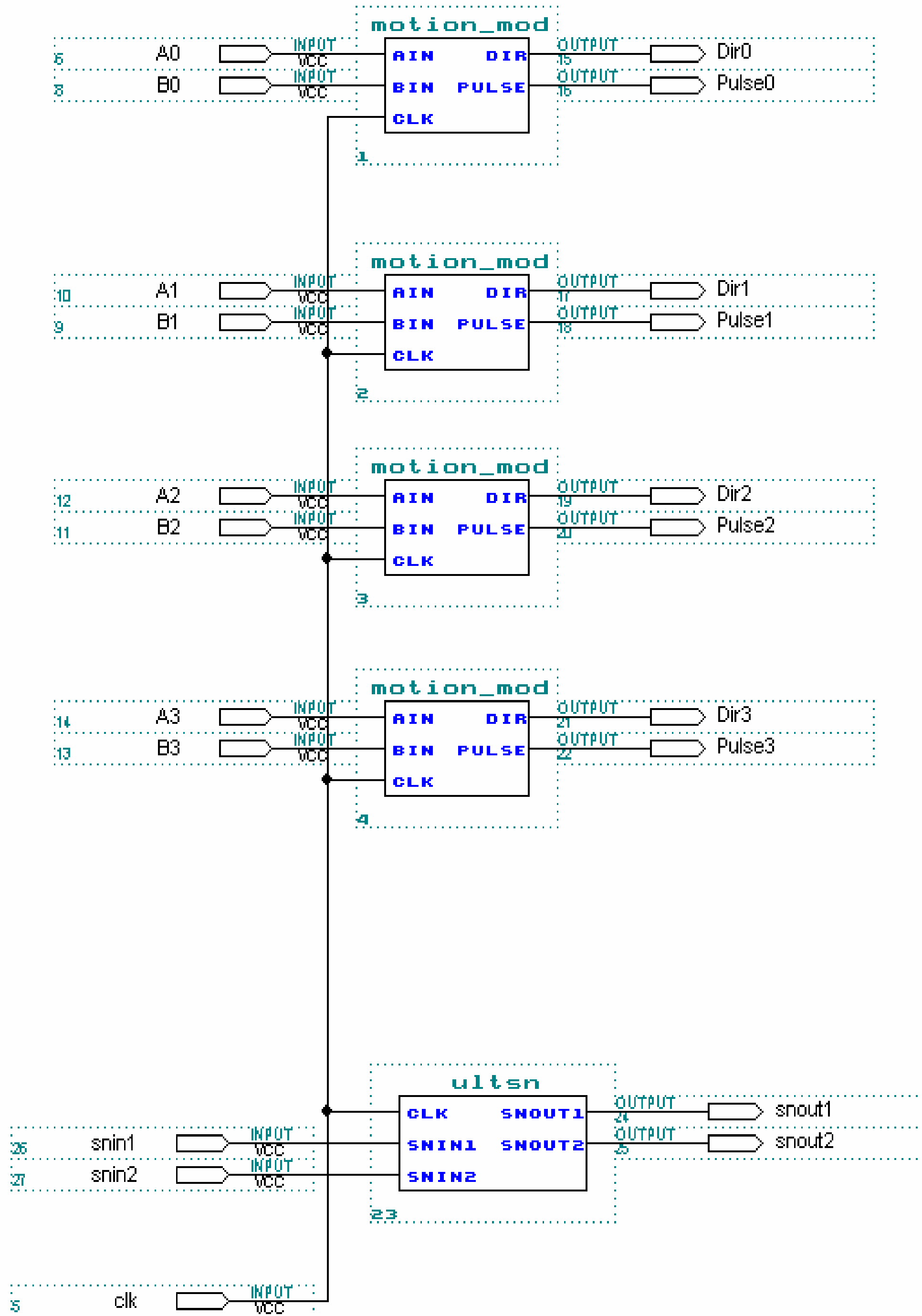
Minutes of the EE meeting on 11/12/01

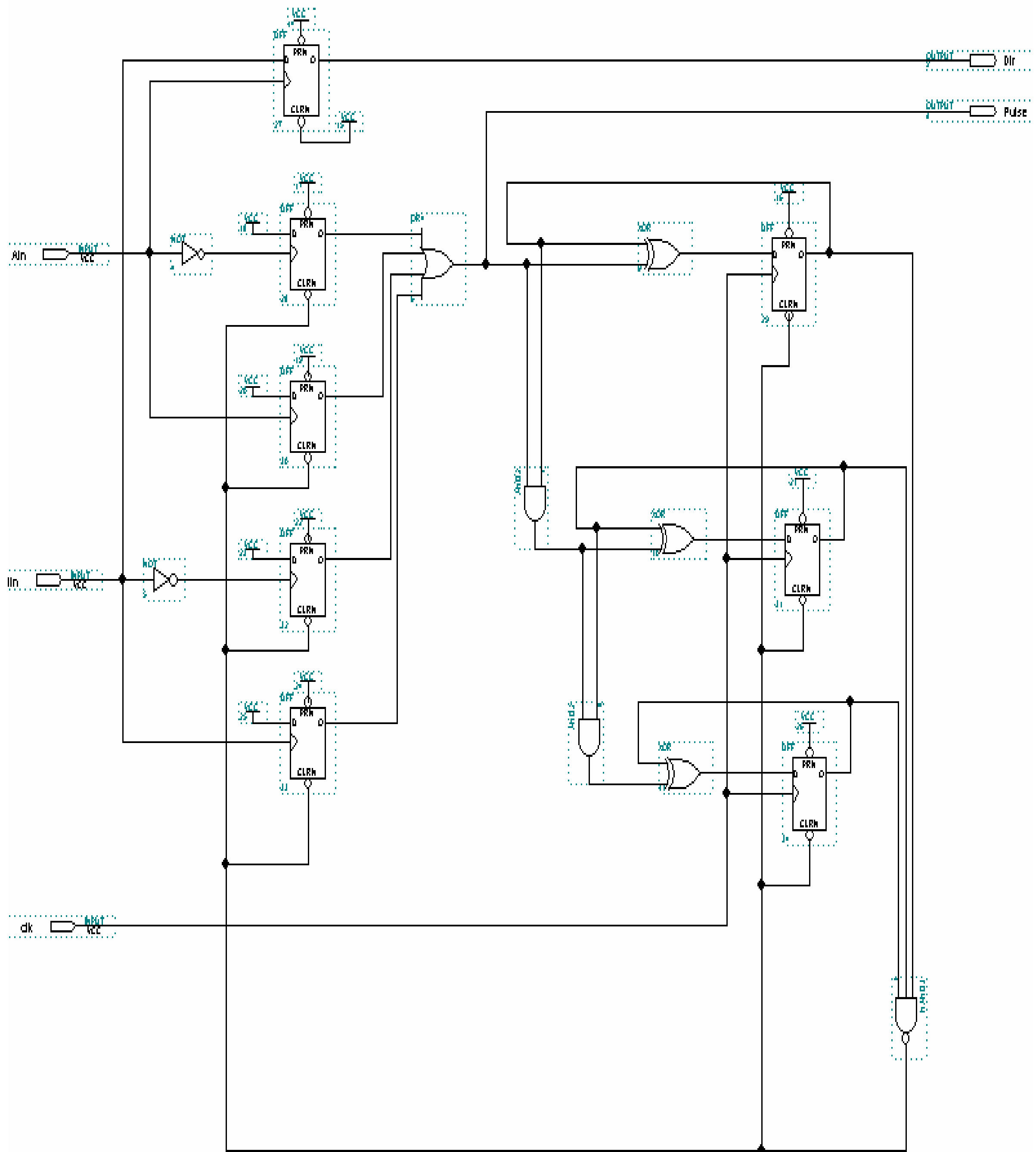
Written by: Shahab A. Najmi

S. No.	Issue	Priority (Critical, Major, Minor)	Responsibility	Due by	Task assigned on	Status and Notes
1	Soldering of 11 on 11 boards	M	Wajih	12-Nov		P – A local company has sponsored the soldering of all boards
2	Maximum current, voltage that is available to the MECH team	M	Wajih	19-Nov	12-Nov	-
3	Look into the motor control options	M	Wajih	15-Nov	12-Nov	-
4	Microcontroller type to be decided	C	Shahab	15-Nov	1-Nov	P – old code finally working!
5	Compiler for Black box/ wireless microcontroller to be ordered	M	MJ	12-Nov	8-Nov	DONE
6	Loop timing test	M	Shahab	17-Nov	1-Nov	
7	Frequency hopping. Comment – the latency seems to be too much for our application	Mi	MJ	17-Nov	1-Nov	
8	Write-up on Wireless	M	MJ	15-Nov	12-Nov	
9	Write-up on IR	M	MS	15-Nov	12-Nov	

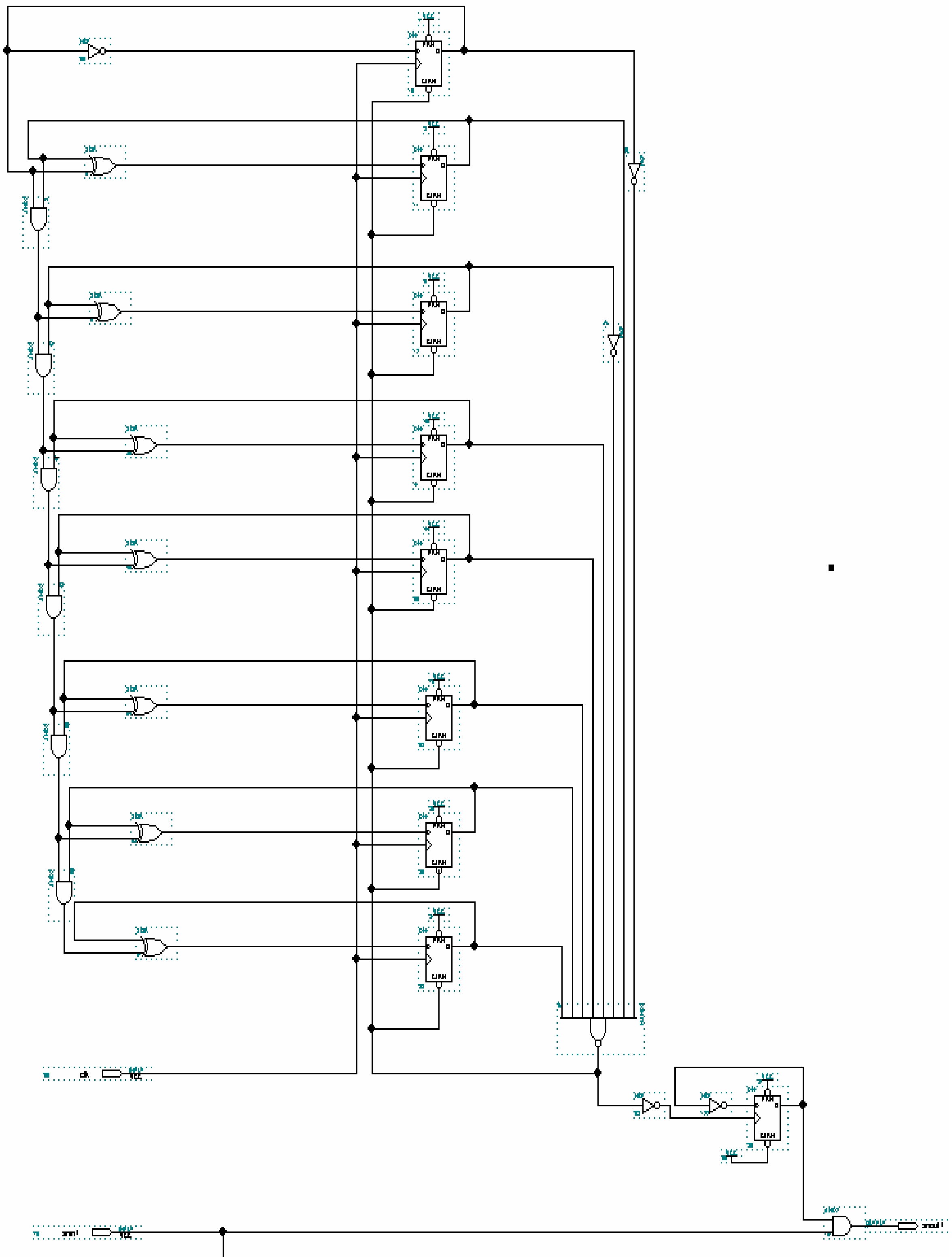
Comments:

F. FPGA CIRCUIT BLOCK DIAGRAM / PINOUT DIAGRAM



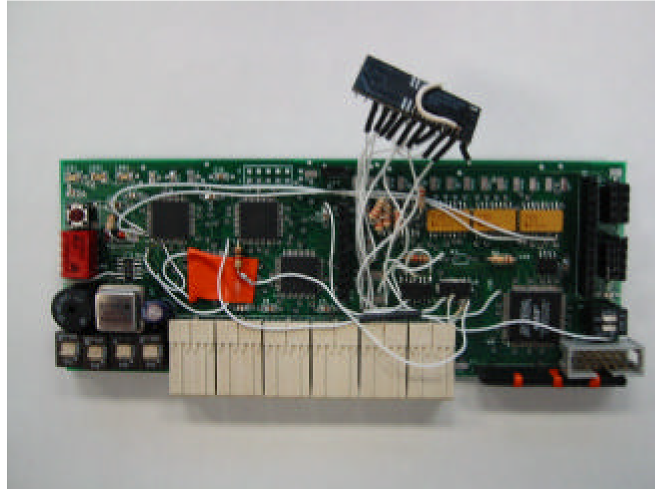


Dir
Pulse
AIn
BIn
clk

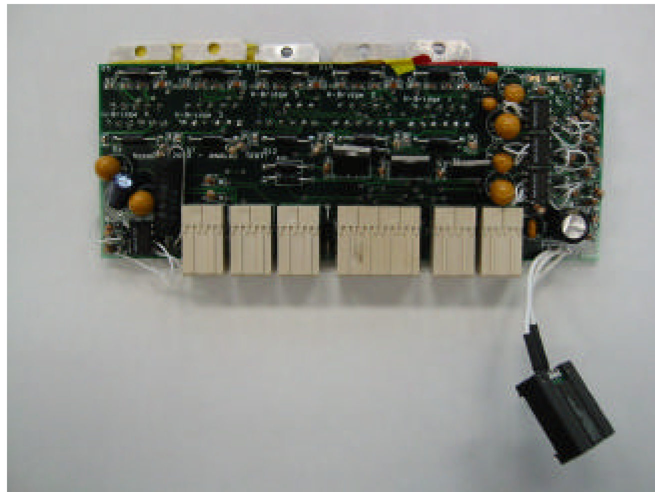


```
Device: EPM7128STC100-7

Device Options:
    Turbo Bit                                    = ON
    Security Bit                                 = OFF
    Enable JTAG Support                          = ON
    User Code                                    = ffff
    MultiVolt I/O                                = OFF

                           R             R R R   R R R R R
                           E             E E E   E E E E E
                           S V           S S S   S S S S S P
                         s E C           E E E V E E E E E u
                         n R C           R R R C R R R R R l
                     G   i V I G G G c G V V V C V V V V V s
             A B A A B N B n E N N N N l N E E E I E E E E E e
             0 3 2 3 0 D 2 2 D T D D D k D D D D O D D D D D 0
           ---------------------------------------------------_
          / 100  98  96  94  92  90  88  86  84  82  80  78  76   |_
         /     99  97  95  93  91  89  87  85  83  81  79  77       |
      B1 |  1                                                    75 | Dir0
      A1 |  2                                                    74 | GND
  VCCIO  |  3                                                    73 | #TDO
    #TDI |  4                                                    72 | Dir3
RESERVED |  5                                                    71 | RESERVED
RESERVED |  6                                                    70 | RESERVED
RESERVED |  7                                                    69 | RESERVED
RESERVED |  8                                                    68 | Dir1
RESERVED |  9                                                    67 | RESERVED
RESERVED | 10                                                    66 | VCCIO
     GND | 11                                                    65 | RESERVED
RESERVED | 12                                                    64 | Pulse3
   snin1 | 13                  EPM7128STC100-7                   63 | Pulse1
RESERVED | 14                                                    62 | #TCK
    #TMS | 15                                                    61 | RESERVED
RESERVED | 16                                                    60 | RESERVED
RESERVED | 17                                                    59 | GND
   VCCIO | 18                                                    58 | RESERVED
RESERVED | 19                                                    57 | RESERVED
RESERVED | 20                                                    56 | Pulse2
RESERVED | 21                                                    55 | Dir2
RESERVED | 22                                                    54 | RESERVED
RESERVED | 23                                                    53 | RESERVED
RESERVED | 24                                                    52 | RESERVED
RESERVED | 25                                                    51 | VCCIO
         |     27  29  31  33  35  37  39  41  43  45  47  49   _|
          \  26  28  30  32  34  36  38  40  42  44  46  48  50      |
           \---------------------------------------------------
             G R R R R R R R V R R R G V s R s G R R R R R R R
             N E E E E E E E C E E E N C n E n N E E E E E E E
             D S S S S S S S C S S S D C o S o D S S S S S S S
               E E E E E E E I E E E   I u E u   E E E E E E E
               R R R R R R R O R R R   N t R t   R R R R R R R
               V V V V V V V   V V V   T 2 V 1   V V V V V V V
               E E E E E E E   E E E       E     E E E E E E E
               D D D D D D D   D D D       D     D D D D D D D
```

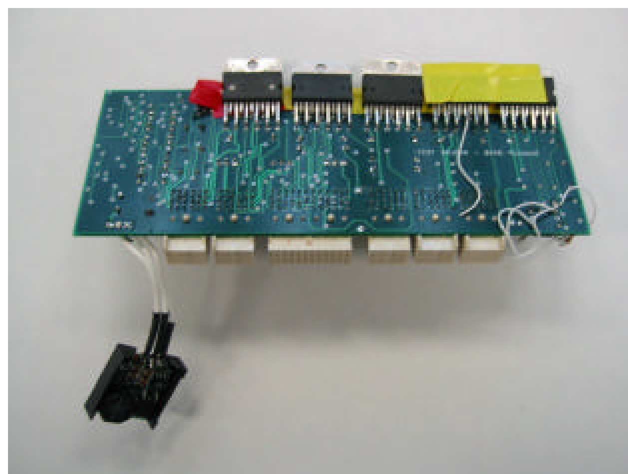
G. PHOTO GALLERY



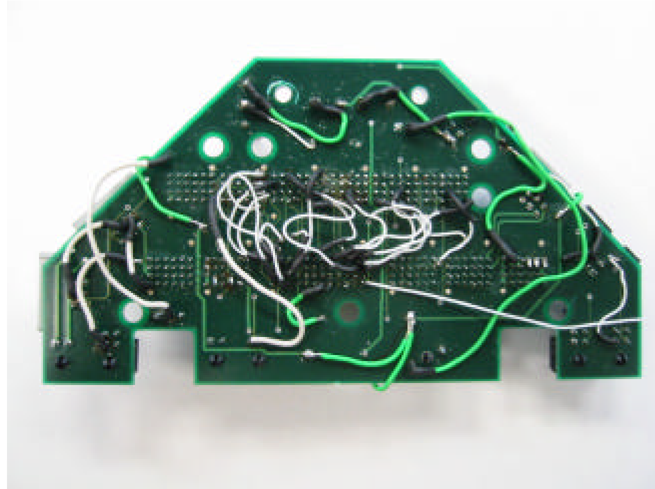
Reworked Digital Board



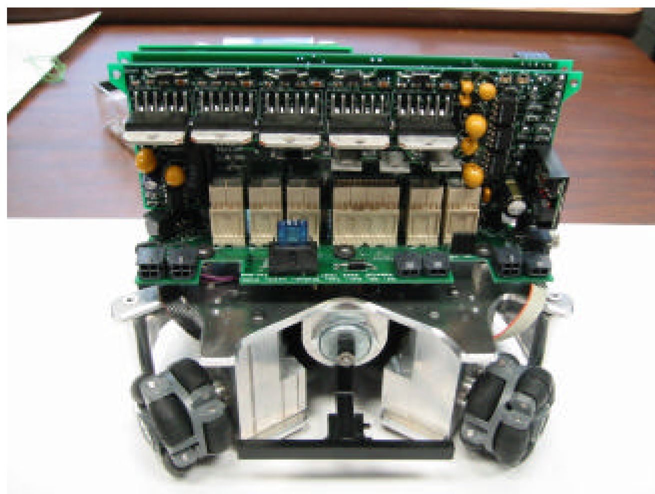
Reworked Analog Board



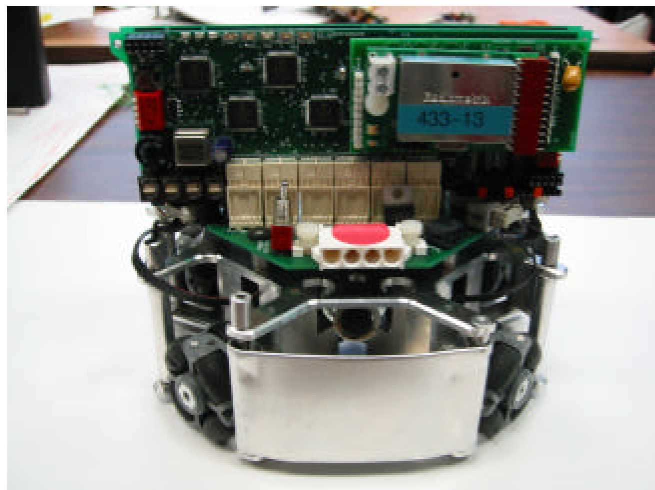
Reworked Analog Board (Rear View)



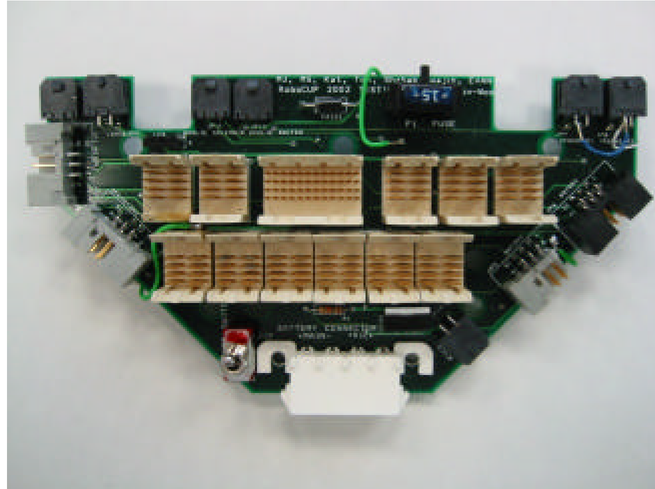
Reworked Motherboard (Rear View)



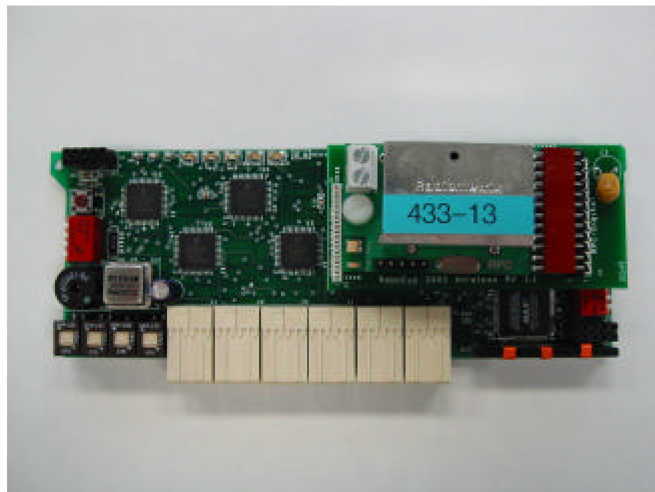
Second Revision (Analog Board)



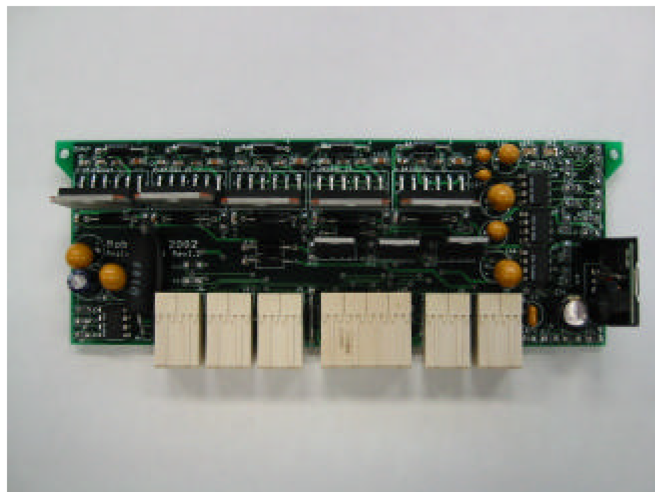
Second Revision (Digital Board)



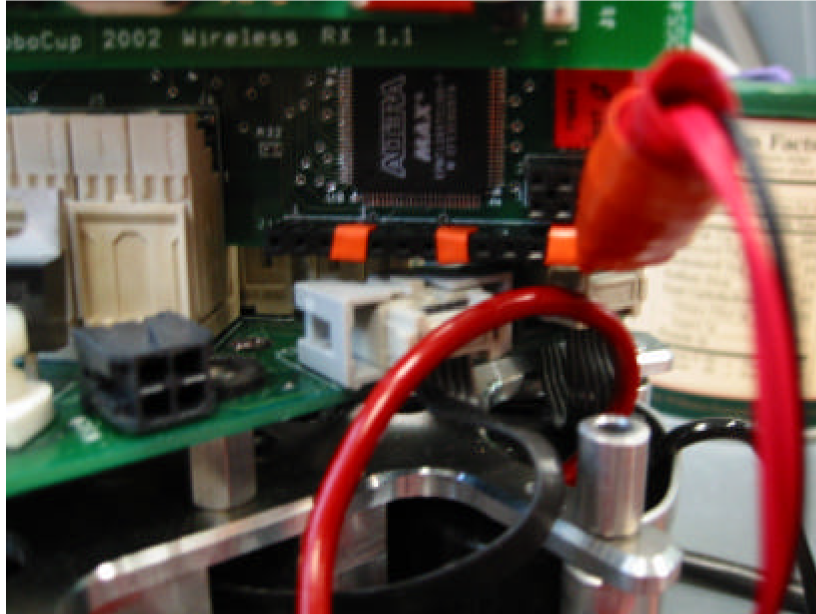
Second Revision (Motherboard)



Second Revision (Digital Board)



Second Revision (Analog Board)



Programming Port / Listening Port (Close-up)

H. MAIN MICROCONTROLLER FIRMWARE

```
//******************        CORNELL ROBOCUP 2002 CODE  (U1)       *************************//
//******************         By:  Shahab A. Najmi            ************************//
// and Wajih Effendi, Michael Jordan, Michael Schwaller, Katherine Ko, David Li, Evan
Malone//


#include <16F877.h>
#include <u1.h>
#use delay(clock=20000000)
#use rs232(baud=57600, xmit=PIN_C6, rcv=PIN_C7, parity=n)
#org 0x1F00,0x1FFF
void loader() {}
#define JUSTKICKED      0
#define RETRACTING     1
#define RETRACT2       2
#define KICKCOMPLETED   3


// ******************** Control switches for testing ********************************
//#define WHEEL
//#define CONTROL
// *********************************************************************************

void Buzzer(int);
void CheckKickerVoltage(void);
void CheckBatteryVoltage(void);
int GetRobotMode(void);
void Kick(void);
void KickerDown(void);
void ControlHDribbler(int);
void KickerIR_ISR(void);
void KickRetract(void);
void test(void);
void GetWirelessPacket(void);
void AfterWirelessPacket(void);
void Wheel1();
void Wheel2();
void Wheel3();
void Wheel4();
void Test_Motion(unsigned int8 v1, unsigned int8 v2, unsigned int8 v3, unsigned int8 v4);
void LCD(void);
byte bgetc(void);



void initialize(void)                              //Initializes the different ports, etc.
{
```

```
  robotNumber=GetRobotMode();          //Reads the robot mode set up by the Robot ID DIP
  kickStatus=KICKCOMPLETED;                          //Initialize the kicker
  byteCount=255;                             //Temporary variable to neglect the first bit in a
wireless frame
  current=0;
  AI_KickSpeed=1;                            //Initialize the kick speed to some value
  AI_HDribblerSpeed=1;                                //Initialize the H Dribbler speed
  enable_interrupts(GLOBAL);                 //Enable interrupts
  enable_interrupts(INT_EXT);                //Enable external interrupt
  enable_interrupts(INT_RDA);                //Enable interrupt on change on port A
  setup_adc(ADC_CLOCK_INTERNAL);             //Setup the ADC on internal clock
  setup_adc_ports(ALL_ANALOG);               //Setup analog port (for current sensing)
  setup_ccp2(CCP_PWM);                                //
  setup_timer_2(T2_DIV_BY_4, 255, 8);        //Setup timer 2 for horizontal dribbler PWM
  setup_timer_1 ( T1_INTERNAL | T1_DIV_BY_8);         //Timer setup for kicker
  CheckBatteryVoltage();                     //Checks the voltage of the battery an sounds
                                             //the buzzer if the voltage is too low
  LCD();                                     //Writes the value of the battery voltage on the
                                             //LCD
}


void main()
{

  initialize();                              //Initializes the different ports etc.

  do                                         //Infinite loop
  {



    if (kickStatus==RETRACTING)              //Checks if the kicker timer is out and it needs
to be retracted
    {
      set_timer1(10000);
      setup_timer_1(T1_INTERNAL | T1_DIV_BY_8);//Sets timer for retracting
      kickStatus=RETRACT2;
    }

  }  while (TRUE);
}

                                  /*  Added this to eliminate hanging inside of getc in ISR's*/

short timeout_error;

char timed_getc()                            //Gets a character. Times out if nothing is
received for a certain amount of time
{
  long timeout;
```

```
  timeout_error=FALSE;
  timeout = 0;
  while(!kbhit()&&(++timeout< 5000))
    delay_us(10);
  if (kbhit())
    return(getc());
  else
  {
    timeout_error=TRUE;
    return(0);
  }
}                                              //end function


#int_ext
void KickerIR_ISR(void)                        //This ISR calls the function kick()
{
  kick();
}



void Kick(void)                                //This function energizes the kicker for
                                               //different time depending upon the speed
                                               //needed
{

  unsigned int16 speedCount;
  long fineTune;

  switch (AI_KickSpeed)                        //Depending upon the speed from the AI the
counter is set
  {
    case 0:
      return;
    case 1:
      speedCount=40000;
      break;
    case 2:
      speedCount=30000;
      break;
    case 3:
      speedCount=0;
      break;
    default:
      speedCount=62411;
      break;
  }

  if (kickStatus==KICKCOMPLETED)               //Only starts if the previous kick has been
completed
  {
```

```
    output_high(PIN_B1);                      //Makes the kicker pin high
    AI_KickSpeed=0;                           //Makes the speed zero to avoid kicking again
    enable_interrupts(INT_TIMER1);            //Enables interrupt on timer overflow
    setup_timer_1 ( T1_INTERNAL | T1_DIV_BY_8); //Setup timer 1: 16 bit counter, increments
every 1.6us, total time 2ms
    set_timer1(speedCount);                   //Timer started
    kickStatus=JUSTKICKED;
  }
}                                             //End of kicker function



#int_timer1                                   //When timer 1 overflows this ISR is called
void KickerDown(void)
{
  int temp;

  if (kickStatus == JUSTKICKED)               //If the status is JUSTKICKED
  {
    output_low(PIN_B1);                       //Kicker down
    kickStatus=RETRACTING;
  }
  else if (kickStatus==RETRACT2)              //If status is RETRACT2
    kickStatus=KICKCOMPLETED;                 //finish the kicker routine

}


#int_rda                                      //Interrupt on change
void GetWirelessPacket()                      //Get the frame from the wireless
microcontroller
{
  tempchar = timed_getc();                    //Getting a byte of data


  // Just Robot 0 for now...fix later
  current++;
  //if(current<WIRELESSPOS[robotNumber]+1)
  if(current<5+1)
  {

    if(byteCount++==255)
    {
      return;
    }

    robotData[byteCount-1]=tempchar;
  }
  else if(current<25)
  {
    return;
  }
```

```
  else
  {
    current=0;
    byteCount=255;

    AfterWirelessPacket();                          //When the frame is completely received,
AfterWirelessPacket() is called
  }
}                                                   //This function currently only works for robot 0


void AfterWirelessPacket(void)
{

                                                              //Parsing the packet


  AI_Dive          = (robotData[3] & 0x20) >> 5;              //Dive bit
  AI_VDribEnable   = (robotData[3] & 0x10) >> 4;              //Side dribbler enable
  AI_HDribblerSpeed = (robotData[3] & 0x0C) >> 2;             //Horizontal dribbler speed
  AI_KickSpeed     = (robotData[3] & 0x03);                   //Kick speed



                                                              // parsing vx, vy, vtheta from
robotData
                                                              // converting to meters/second and
radians/second

                                                                /*
                                                                NOTE:
                                                                  vx   = MAX_VELOCITY * vx / 0x7D;

                                                                This should, in theory, be the
                                                                following:
                                                                  vx   = MAX_VELOCITY * vx / 0x7F;

                                                                However, empirically, 0x7D works
                                                                better than 0x7F.  We should find out
                                                                why!
                                                                */


  vx   = (float)((robotData[0])&(0x7F));                      //Converting the value received from
                                                              //the AI into float (first 7 bits)
  vx   = MAX_VELOCITY * vx / 0x7D;                            //Converting vx into m/s
  if (robotData[0] >> 7)                            //Changing the sign of the speed depending
upon the MSB
     vx=vx*(-1);


  vy   = (float)((robotData[1])&(0x7F));            //Converting vy
  vy   = MAX_VELOCITY * vy / 0x7D;
  if (robotData[1] >> 7)
```

```
    vy=vy*(-1);

   vtheta = (float)(robotData[2]);                    //Converting into float
   vtheta = MAX_ANGULAR_VELOCITY * vtheta / 0xFE;        //Converting into rad/s
   if (robotData[3] >> 7)
     vtheta=vtheta*(-1) ;


   Wheel1();                                          //Converting into wheel speed for
                                                      //wheel 1
   Wheel2();                                          //Converting into wheel speed for
                                                      //wheel 2
   Wheel3();                                          //Converting into wheel speed for
                                                      //wheel 3
   Wheel4();                                          //Converting into wheel speed for
                                                      //wheel 4

   #use rs232(baud=57600, xmit=PIN_D4, rcv=PIN_C4, parity=n)
   putc(iV1);                                         //Sending the motion micro 1 values
 for wheel 1
   putc(iV2);                                         //Sending the motion micro 1 values
 for wheel 2

   #use rs232(baud=57600, xmit=PIN_D5, rcv=PIN_C5, parity=n)
   putc(iV3);                                         //Sending the motion micro 1 values
 for wheel 3
   putc(iV4);                                         //Sending the motion micro 1 values
 for wheel 4

 if ((AI_KickSpeed !=0) & (kickStatus==KICKCOMPLETED))//Kick if AI commands and if last
                                                      //kick already completed
   {
     if ( input(PIN_B1) )
       kick();
   }


   ControlHDribbler(AI_HDribblerSpeed);

   if (AI_VDribEnable)
     output_high(PIN_C2);
   else
     output_low(PIN_C2);

   return;

 }

 void Wheel1()                                         //Conversion into wheel
                                                       //velocity for wheel 1
 {
```

```
  V  = cos_psi1 * vx;
  V += sin_psi1 * vy;
  V += vtheta_coeff * vtheta;                                   //Using the geometric
                                                                //conversion

  if (V < 0)
  {
    if (V < -MAX_WHEEL_VELOCITY) { V = -MAX_WHEEL_VELOCITY; }   //Truncates
                                                                //negative overflow
    iV1 = (int8) (0x7F * -V / MAX_WHEEL_VELOCITY);
    iV1 = iV1 | 0x80;
  }
  else {
    if (V > MAX_WHEEL_VELOCITY) { V = MAX_WHEEL_VELOCITY; }     //Truncates positive
                                                                //overflow
    iV1 = (int8) (0x7F * V / MAX_WHEEL_VELOCITY);
  }

}

void Wheel2()
{
  V  = cos_psi2 * vx;
  V += sin_psi2 * vy;
  V += vtheta_coeff * vtheta;

  if (V < 0) {
    if (V < -MAX_WHEEL_VELOCITY) { V = -MAX_WHEEL_VELOCITY; }   //Truncates
                                                                //negative overflow
    iV2 = (int8) (0x7F * -V / MAX_WHEEL_VELOCITY);
    iV2 = iV2 | 0x80;
  }
  else {
    if (V > MAX_WHEEL_VELOCITY) { V = MAX_WHEEL_VELOCITY; }     //Truncates positive
                                                                //overflow
    iV2 = (int8) (0x7F * V / MAX_WHEEL_VELOCITY);
  }

}

void Wheel3()
{
  V  = cos_psi3 * vx;
  V += sin_psi3 * vy;
  V += vtheta_coeff * vtheta;

  if (V < 0) {
    if (V < -MAX_WHEEL_VELOCITY) { V = -MAX_WHEEL_VELOCITY; }   //Truncates
                                                                //negative overflow
    iV3 = (int8) (0x7F * -V / MAX_WHEEL_VELOCITY);
    iV3 = iV3 | 0x80;
  }
```

```
  else {
    if (V > MAX_WHEEL_VELOCITY) { V = MAX_WHEEL_VELOCITY; }     //Truncates positive
                                                                //overflow
    iV3 = (int8) (0x7F * V / MAX_WHEEL_VELOCITY);
  }

}

void Wheel4()
{
  V  = cos_psi4 * vx;
  V += sin_psi4 * vy;
  V += vtheta_coeff * vtheta;

  if (V < 0) {
    if (V < -MAX_WHEEL_VELOCITY) { V = -MAX_WHEEL_VELOCITY; }    //truncates
                                                                 //negative overflow....
    iV4 = (int8) (0x7F * -V / MAX_WHEEL_VELOCITY);
    iV4 = iV4 | 0x80;
  }
  else {
    if (V > MAX_WHEEL_VELOCITY) { V = MAX_WHEEL_VELOCITY; }    //truncates positive
                                                               //overflow....
    iV4 = (int8) (0x7F * V / MAX_WHEEL_VELOCITY);
  }

}



void ControlHDribbler(int hDribblerSpeed)            //This function controls the
                                                     //speed of the dribbler
                                                     //depending upon
                                                     //the voltage from the current
                                                     //sensing circuitry
{
  signed int16 duty;
  int analogHDribblerValue;
  float multiplier;

  switch(AI_HDribblerSpeed)                          //Depending upon the speed
                                                     //from the AI, a base value of
                                                     //multiplier is set
  {
    case 0:
      multiplier=0;
      break;
    case 1:
      multiplier=.150;
      break;
    case 2:
      multiplier=.300;
```

```
      break;
    case 3:
      multiplier=1;
      break;
    default:
      multiplier=0;
      break;
  }

  set_adc_channel(4);
  delay_us(10);
  analogHDribblerValue = read_adc();                    //Reads the value from the
ADC input


  if ( (analogHDribblerValue > 170) || (analogHDribblerValue== -1) )

    duty = 520 * multiplier;
  else if (analogHDribblerValue > 85)

    duty = 700 * multiplier;
  else
    duty = 900 * multiplier;                    //Duty cycle set depending upon the
value of the current

  //setup_ccp2(CCP_PWM);
  //setup_timer_2(T2_DIV_BY_4, 127, 1);

  set_pwm2_duty(duty);
}                                               //End of the function

int8 GetRobotMode(void)                                   //Reads the robot ID
from the ID DIP switch
{

  modeNumber= (input_d() & 0b00001111);
  return(modeNumber);
}


void Buzzer(int timesOn)                        //Sounds the buzzer
depending upon the input argument to it
{

int i;

  for (i=0; i<timesOn; i++)
  {
    output_low(PIN_D6);        //Buzzer on
    delay_ms(500);
    output_high(PIN_D6);        //Buzzer off
    delay_ms(500);
```

```
  }
}


void CheckBatteryVoltage(void)                          //Measures the battery voltage and
sounds a buzzer if it is low
{
  int16 value;

  set_adc_channel(1);
  delay_us(100);
   value = read_adc();
  batteryLevel=value;

  if (value <=batteryThreshold)
    Buzzer(2);
}                                                              //End of the function


void LCD(void)
{

  float percentage;



  #use rs232(baud=19200, xmit=PIN_D7, rcv=PIN_C0, parity=n)   //Setting the pins to send
data to the LCD
     printf("            \n");
    printf("            \n");
    printf("\nBat %x  ",batteryLevel);
    printf("\nRNo %d       ",robotNumber);                     //Printing robot No and
battery level
}



void test(void)                                                //Sends pre-programmed
directions to the robot
{
  #ifdef WHEEL
  printf("IN\n\r");
    if (robotNumber == 0x0e)

    {
      Test_Motion(0x08, 0x08, 0x08, 0x08);
    }
    if (robotNumber == 0x0c)
    {
                                                //Data for Straight
      robotData[0] = 0x26;                                 //vx
      robotData[1] = 0x00;                                 //vy
```

```
        robotData[2] = 0x00;                                    //vtheta
        robotData[3] = 0x00;                                    // and MSB of this is sign bit
                                                                //for vtheta

        AfterWirelessPacket();
    }
    else if(robotNumber == 0x0a)
    {
                                                            //Data for Left
        robotData[0] = 0x00;                                    //vx
        robotData[1] = 0x26;                                    //vy
        robotData[2] = 0x00;                                    //vtheta
        robotData[3] = 0x00;                                    // and MSB of this is sign bit
                                                                //for vtheta

        AfterWirelessPacket();
    }
    else if(robotNumber == 0x08)
    {
                                                        //Data for Back
//      robotData[0] = -0x26;                                   //vx
        robotData[1] = 0x00;                                    //vy  //was 80
        robotData[2] = 0x00;                                    //vtheta
        robotData[3] = 0x00;                                    // and MSB of this is sign  bit
                                                                //for vtheta

        AfterWirelessPacket();
    }
    else if(robotNumber == 0x06)
    {
                                                            //Data for Right
        robotData[0] = 0x00;                                    //vx
//      robotData[1] = -0x26;                                   //vy  //ec
        robotData[2] = 0x00;                                    //vtheta
        robotData[3] = 0x00;                                    // and MSB of this is sign bit
for vtheta

        AfterWirelessPacket();
    }
    else if(robotNumber == 0x04)
    {
                                                            //Data for Rotate Right
        robotData[0] = 0x00;                                    //vx
        robotData[1] = 0x00;                                    //vy
//      robotData[2] = -0x26;                           //vtheta
        robotData[3] = 0x00;                                    // and MSB of this is sign bit
for vtheta

        AfterWirelessPacket();
    }
    else if(robotNumber == 0x02)
    {
```

```
    //Data for Rotate Left
    robotData[0] = 0x00;                              //vx
    robotData[1] = 0x00;                              //vy
    robotData[2] = 0x26;                              //vtheta
    robotData[3] = 0x00;                              // and MSB of this is sign bit
                                                      //for vtheta

    AfterWirelessPacket();
  }
#endif

#ifdef CONTROL

  if(robotNumber == 0x00)
  {
    //Test Kicker 00 - Kicker Speed 0 (off)
    robotData[0] = 0x00;
    robotData[1] = 0x00;
    robotData[2] = 0x00;
    robotData[3] = 0x00; // 2 LSBs are kicker Magnitude

    AfterWirelessPacket();
  }

   if(robotNumber == 0x02)
  {
    //Test Kicker 01 - Kicker Speed 1
    robotData[0] = 0x00;
    robotData[1] = 0x00;
    robotData[2] = 0x00;
    robotData[3] = 0b00000101;                        // 2 LSBs are kicker Magnitude

    AfterWirelessPacket();
  }
   if(robotNumber == 0x04)
  {
    //Test Kicker 02 - Kicker Speed 2
    robotData[0] = 0x00;
    robotData[1] = 0x00;
    robotData[2] = 0x00;
    robotData[3] = 0b00011010;  // 2 LSBs are kicker Magnitude

    AfterWirelessPacket();
  }
   if(robotNumber == 0x06)
  {
    //Test Kicker 03 - Kicker Speed 3
    robotData[0] = 0x00;
    robotData[1] = 0x00;
    robotData[2] = 0x00;
    robotData[3] = 0b00011111;  // 2 LSBs are kicker Magnitude
```

```
      AfterWirelessPacket();
    }
     if(robotNumber == 0x08)
    {
      //Test Kicker
      output_high(PIN_B1);
      delay_ms(200);

      output_low(PIN_B1);
      delay_ms(1000);
      output_low(PIN_B1);
    }
    #endif
}


void Test_Motion(unsigned int8 v1, unsigned int8 v2, unsigned int8 v3, unsigned int8 v4)
{
                                                                        //This function lets
you send 8bit wheel velocities to the
                                                                        //motion control

  iV1 = v1;
  iV2 = v2;
  iV3 = v3;
  iV4 = v4;

  #use rs232(baud=57600, xmit=PIN_D4, rcv=PIN_C4, parity=n)
  putc(iV1);
  putc(iV2);

  #use rs232(baud=57600, xmit=PIN_D5, rcv=PIN_C5, parity=n)
  putc(iV3);
  putc(iV4);
}
```

```
//********************************Header file for the U1 code *************************************//
int robotNumber;
int kickerBatteryThreshold=200;
int batteryThreshold=220;
int modeNumber=0;
int IR_Status;
int kickStatus=KICKCOMPLETED;
unsigned int8 robotData[4];
byte AI_Dive;
byte AI_VDribEnable;
byte AI_KickSpeed;
byte AI_HDribblerSpeed;
char tempchar;
int8 current;
int8 byteCount;
int16 batteryLevel;
#define bkbhit (next_in!=next_out)
#define BUFFER_SIZE 25
byte buffer[BUFFER_SIZE];
byte next_in = 0;
byte next_out = 0;


/* Start LUT Variables */

/*  Wheel Drive Direction Angles
Psi1 = 145 degrees
Psi2 = 225 degrees
Psi3 = 315 degrees
Psi4 = 35 degrees
*/

//Wheel geometry constants
FLOAT CONST cos_psi1 = -0.819152;
FLOAT CONST sin_psi1 = 0.573576;
FLOAT CONST cos_psi2 = -0.707107;
FLOAT CONST sin_psi2 = -0.707107;
FLOAT CONST cos_psi3 = 0.707107;
FLOAT CONST sin_psi3 = -0.707107;
FLOAT CONST cos_psi4 = 0.819152;
FLOAT CONST sin_psi4 = 0.573576;

//Vphi constant
//NOTE:  vtheta_coeff needs to be updated to reflect any
//robot geometry changes
FLOAT CONST vtheta_coeff = 0.0717;

/*  Wheel Positions */
FLOAT CONST x1 = 0.0372;
FLOAT CONST y1 = 0.0615;
FLOAT CONST x2 = -0.0525;
FLOAT CONST y2 = 0.0489;
```

```
FLOAT CONST x3 = -0.0525;
FLOAT CONST y3 = -0.0489;
FLOAT CONST x4 = 0.0372;
FLOAT CONST y4 = -0.0615;

//to convert wireless commands into meters/second and radians/second
FLOAT CONST MAX_WHEEL_VELOCITY   =  2.5;
FLOAT CONST MAX_VELOCITY         =  2.5;
FLOAT CONST MAX_ANGULAR_VELOCITY = 10.0;

//for calculating coordinate transformations
float vx;
float vy;
float vtheta;

//to hold the integer wheel velocity to send to motion control
unsigned int8 iV1;
unsigned int8 iV2;
unsigned int8 iV3;
unsigned int8 iV4;

float V;

/* End LUT Variables*/


/* Begin Wireless RX receive Variables */
  BYTE CONST WIRELESSPRE[6] = {0,1,5,9,13,17};    // end byte bytes before robotData
  BYTE CONST WIRELESSWAN[6] = {1,5,9,13,17,21};   // robotData
  BYTE CONST WIRELESSPOS[6] = {5,9,13,17,21,26};  // start bytes after robotData
/* End Wireless RX receive Variables */
```

I. MOTION CONTROL FIRMWARE

```
// RoboCup 2002 motion control code - Tom Chi
#include <16F877.h>
// #include <16F876.h>
#use delay(clock=20000000)
#use rs232(baud=57600, xmit=PIN_C6, rcv=PIN_C7, parity=n)
#org 0x1F00,0x1FFF
   void loader() {}

#define VECTOR_LENGTH 7            // 10 bits of control for each wheel
#define BASE_K 4                        // proportional control constant
#define J 2                        // integral control constant
#define BUFFER_SIZE 2              // Buffersize for Motion Vector
#define SSE_THRESHOLD 5            // allowable Stead State Error Threshold
#define GLOBAL_RATE 300            // (Hz) control loop rate

byte buffer[BUFFER_SIZE];          // input buffer to store motion vectors on the HW RX
byte next_in = 0;
byte next_out = 0;

int8 count_scale = 0;               // count scale that is used to set desired count
based on rate
int8 K = 0;
signed int16 timer0track = 0;

int1 dir1 = 1;                     // wheel direction: default forward = CCW
int1 dir2 = 1;                     // wheel direction: default forward = CCW
signed int16 pwm1val = 0;          // PWMval set in tight loop
signed int16 pwm2val = 0;          // PWMval set in tight loop

int8 dir1mem = 0;                   // 1 bit memory to filter direction errors
int8 dir2mem = 0;                   // 1 bit memory to filter direction errors

signed int16 cnt1 = 0;             // timer0 value
signed int16 cnt2 = 0;             // timer1 value
signed int16 cnt1des = 0;          // ideal count1 value
signed int16 cnt2des = 0;          // ideal count2 value
signed int16 cnt1int = 0;          // integral control count 1
signed int16 cnt2int = 0;          // integral control count 2

signed int16 dif1 = 0;             // count difference for motor 1
signed int16 dif2 = 0;             // count difference for motor 2

signed int16 temp1 = 0;            // intermediate calculation val
signed int16 temp2 = 0;            // intermediate calculation val
        int1 dir1Read = 0;         // observed wheel direction 1
        int1 dir2Read = 0;         // observed wheel direction 2

signed int16 cnt1db = 0;           // motor 1 DEBUG (DB) variables
signed int16 tmp1db = 0;
signed int16 dif1db = 0;
signed int16 pwm1db = 0;
        int1 dir1db = 0;
signed int16 cnt2db = 0;           // motor 2 DEBUG (DB) variables
signed int16 tmp2db = 0;
signed int16 dif2db = 0;
signed int16 pwm2db = 0;
        int1 dir2db = 0;

#define bkbhit (next_in!=next_out)

byte bgetc() {
   byte c;

   while(!bkbhit) ;
   c=buffer[next_out];
   next_out=(next_out+1) % BUFFER_SIZE;
   return(c);
}
```

```
short timeout_error;

char timed_getc() {
   long timeout;

   timeout_error=FALSE;
   timeout = 0;
   while(!kbhit()&&(++timeout< 5000))
      delay_us(10);
   if (kbhit())
      return(getc());
   else {
      timeout_error=TRUE;
      return(0);
   }
}

// ******************************************************************************
// INTERRUPT SERVICE ROUTINES //

#INT_RDA
serial_isr() {                           // Buffers motion vectors coming on Serial Rx
   int t;

   // read motion vectors
   buffer[0] = timed_getc();
   buffer[1] = timed_getc();

   //printf("%x ",buffer[0]);         //temp delete
   //printf("%x \r\n",buffer[1]);

   // Convert Vector into PWM values (can take up to 10 bit)
   cnt1des = (int16)(buffer[0] & 0x7F) * count_scale;  // approximate count scaling: see
spreadsheet: motorcalc.xls in actuality: 4.9
   dir1 = buffer[0] >> 7;

   cnt2des = (int16)(buffer[1] & 0x7F) * count_scale;
   dir2 = buffer[1] >> 7;

   // Start Counters
   set_timer0(0);
   set_timer1(0);

   dir1mem = dir1;
   dir2mem = dir2;
}

#int_rtcc
timer0_isr() {
   timer0track++;
   if (timer0track > 3)
      timer0track = 0;
}

#int_timer2
wheel_control_isr() {
//   output_bit(PIN_C5,0);                 // debug scope signal to mark beginning of
ISR
//______________________________________________________________________________ GET
FRAME INFO
   dir1Read = input(PIN_C3);             // get motor direction: motor1
   dir2Read = input(PIN_C4);             // get motor direction: motor2

   cnt1 = get_timer0() + (timer0track << 8);                  // read number of counts
(timer0)
   cnt2 = get_timer1();                  // read number of counts (timer1)
   cnt1 = cnt1 << 1;            // !!! HACK HACK HACK :: to get rid of overcounting.


   set_timer0(0);                        // reinitialize timer0
```

```
   timer0track = 0;                        // var needed to make timer0 (8bit counter)
count up to 1024
   set_timer1(0);                          // reinitialize timer1
//____________________________________________________________________________________
CALCULATE NEW PWMs

   if ((dir1Read!=dir1) && (dir1mem==dir1Read))
      cnt1 = - cnt1;   // if dir is opposite of desired: switch sign on Encoder Count

   if ((dir2Read != dir2) && (dir2mem==dir2Read))
      cnt2 = - cnt2;

   dir1mem=dir1Read;
   dir2mem=dir2Read;

   dif1 = cnt1des-cnt1;
   dif2 = cnt2des-cnt2;


   if (temp1 < 0)                         // MOTOR 1: if negative
      pwm1val = -temp1;
   else if (temp1 > 1023)                  // cap PWM value at 1023
      pwm1val = 1023;
   else                                    // else: regular case
      pwm1val = temp1;

   if (temp2 < 0)                          // MOTOR 2: if negative
      pwm2val = -temp2;
   else if (temp2 > 1023)                 // cap PWM value at 1023
      pwm2val = 1023;
   else                                   // else: regular case
      pwm2val = temp2;

//____________________________________________________________________________________ SET
DIRECTION

if (temp1 < 0) {                           // on overcount, flip dir
   if (dir1 == 1)
      dir1 = 0;
   else
      dir1 = 1;
}
if (temp2 < 0) {
   if (dir2 == 1)
      dir2 = 0;
   else
      dir2 = 1;
}

   if (dir1) {                            // set direction
      output_bit(PIN_B5,1);
      output_bit(PIN_B4,0);
   }
   else {
      output_bit(PIN_B5,0);
      output_bit(PIN_B4,1);
   }

   if (dir2) {
      output_bit(PIN_B2,1);
      output_bit(PIN_B1,0);
   }
   else {
      output_bit(PIN_B2,0);
      output_bit(PIN_B1,1);
   }

if (temp1 < 0) {                           // flip back
   if (dir1 == 1)
      dir1 = 0;
   else
```

```
      dir1 = 1;
}
if (temp2 < 0) {
   if (dir2 == 1)
      dir2 = 0;
   else
      dir2 = 1;
}

   set_pwm1_duty(pwm1val);          // wait till after direction set to set PWM
   set_pwm2_duty(pwm2val);

//_____________________________________________________________________________ DEBUG
SIGNALS
   dir2db = dir2Read;              // set debug variables
   cnt2db = cnt2;
   tmp2db = temp2;
   dif2db = dif2;
   pwm2db = pwm2val;

 //  output_bit(PIN_C5,1);          // scope signal to mark end of ISR
} // END wheel_control_isr()



// ************************************************************************************
// DEFINES //


void main() {
  enable_interrupts(GLOBAL);                // turn on interrupts
  enable_interrupts(INT_RDA);               // Hardware RX Interrupt
  enable_interrupts(INT_TIMER2);            // TIMER2 interrupt: triggers control loop
code
  enable_interrupts(int_rtcc);            // 8b count ... need to track to count up to
10 bits.
  setup_ccp1(CCP_PWM);                      // PWM1
  setup_ccp2(CCP_PWM);                      // PWM2
  setup_timer_0(RTCC_EXT_L_TO_H);           // counter 1
  setup_timer_1(T1_EXTERNAL);               // counter 2



if (GLOBAL_RATE == 600) {                   // sets timebase for control loop AND PWM
rate : 600 Hz control loop
  setup_timer_2(T2_DIV_BY_4, 255, 8);
  count_scale = 5;
  K = BASE_K << 2;
}
else if (GLOBAL_RATE == 300) {              // 300 Hz control loop
   setup_timer_2(T2_DIV_BY_4, 255, 16);
   count_scale = 10;
   K = BASE_K << 1;
}
else if (GLOBAL_RATE == 150) {              // 150 Hz control loop
   setup_timer_2(T2_DIV_BY_16, 255, 8);
   count_scale = 20;
   K = BASE_K;
}

  set_pwm1_duty(0);                         // init state: motor1 off
  set_pwm2_duty(0);                         // init state: motor2 off
  cnt1des = 0;
  cnt2des = 0;

//  printf("*** Starting Main Loop...\r\n");
  while(1) {

  }
}
```

J. SYSTEM REQUIREMENTS

CORNELL 2002 ROBOCUP 5 ROBOT SYSTEM

1. SCOPE
1.1. This source specification establishes the performance, design, development, and test requirements for a system of autonomous soccer-playing robots.

2. MISSION
2.1. The mission of the Cornell 2002 RoboCup 5 Robot System is to win the 2002 RoboCup F180 League Competition.

3. APPLICABLE DOCUMENTS
3.1. CORNELL 2002 ROBOCUP 5 ROBOT SYSTEM source specification (this document)
3.2. RC02Orient - Microsoft PowerPoint Presentation, Dr. R. D'Andrea
3.3. RoboCup Organization F180 (Small Size) League Rules for RoboCup 2002
3.4. Cornell RoboCup Documentation from 1999, 2000, 2001
3.5. Documentation of other RoboCup systems
3.6. Systems Engineering process references
3.6.1. Engineering Complex Systems, Oliver, Kelliher, Keegan

4. GENERAL REQUIREMENTS
4.1. The system shall comply with the 2002 RoboCup Organization F180 (Small Size) League Rules.
4.2. The system shall win robot soccer matches played against the Cornell 2001 RoboCup 5 Robot System, prior to the 2002 RoboCup Competition.
4.3. The system shall win robot soccer matches played against the Cornell 2001 RoboCup 11 Robot System, prior to the 2002 RoboCup Competition.
4.4. The system shall not experience spontaneous malfunction or breakdown during the course the RoboCup 2002 Competition.
4.4.1. The system shall be easily testable.
4.4.2. The system shall be easily maintainable.
4.5. The system shall not display any of the known malfunctions of the Cornell 2001 RoboCup 5 Robot System.
4.6. The system shall be safely and easily transported to the location of the 2002 RoboCup Competition in Japan.
4.6.1. The system shall comply with air transport regulations.
4.7. The system shall comply with all applicable laws at the competition venue (Fukuoka, Japan).
4.8. The system shall perform at least as well in the RoboCup 2002 venue as in the Cornell RoboCup Laboratory.
4.9. The system shall be tolerant of human errors.
4.10. The system behavior shall be easily adjustable during competition.
4.10.1. The system shall allow strategy changes at the competition.
4.11. Every subsystem shall be either redundant, or easily replaced.
4.12. The system shall, where practical, share hardware resources with the Cornell 2002 11 Robot System.