The Robot control using the wireless communication and the serial communication

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

JONG HOON AHNN

Project Advisor: Professor Mark Campbell

Degree Date: May 2007

Abstract

Master of Electrical Engineering Program

Cornell University

Design Project Report

Project Title: Robot control using the wireless communication and the serial communication

Author: Jong Hoon Ahnn

Abstract:

This project outlines the strategy adopted for establishing two kinds of communications; one for wireless communication between a mobile Robot and a remote Base Station, another for serial communication between a remote Base Station and a GUI Application, PC. TekBot is a low cost mobile Robot built by Oregon State University which in its current version requires wired communication. Our aim is to be able to command and control the Robot wirelessly by the GUI Application. This will be a useful addition for NASA's curriculum development for master's degree programs.

The principle task of this project was to program the AVR microcontroller interfaced to a radio packet controller module (operating at a frequency of 433 MHz) which would enable us to wirelessly control the Robot. The communication protocols dealing with transmission and reception of data and wireless control of the TekBot have been successfully implemented. These details are discussed in this report.

Report Approved by

Project Advisor:____Date:_____________

Executive Summary

The NASA Robotics Alliance Cadets Program is being designed to be implemented at a very low start-up cost, and to make this goal obtainable, the program is being developed using low to no cost components from already developed, well-tested and robust engineering testbeds.

In this project we are trying to establish both wireless communication between the mobile Robot and the remote Base Station, and serial communication between the remote Base Station and the GUI Application. The Base Station requires the serial communication with the GUI Application and also needs to be hardwired with the radio packet controller, FRPC2 for wireless control. Our aim is to be able to command and control the Robot wirelessly by the GUI Application.

The main task of this project is two parts: (1) to program the AVR microcontroller on both the Base Station and the Robot interfaced to the radio packet controller module which would enable us to wirelessly control the Robot; (2) to program the GUI Application which would enable us to serially control the Base Station.

Theoretical system limitation for the packet transmission is evaluated and analyzed. We tested packet stress to the wireless module while varying the number of Robots and the payload data. The wireless parts were evaluated with CRC error checking.

As a result, we achieved control both wireless communication between the mobile Robot and the remote Base Station, and serial communication between the remote Base Station and the GUI Application. This level of completely was successfully tested on groups at up to four Robots. Hence the wireless communication and the serial communication were successful in the downlink.

Contents

1. Introduction…………………………………………………………………….4
2. Project requirements……………………………………………………………..7
3. Radio Packet Controller Module ………………………………………………...8
Design Requirements………………………………………………………8
Fast Radio Packet Controller (FRPC) …………………………………….10
Signals……………………………………………………………………..13
Functional Description…………………………………………………….13
Operating States…………………………………………………………...15
Port Operations and Initializations………………………………………..16
4. ATmega 128.2 Microcontroller Board………………………………………..18
Introduction………………………………………………………………..18
ISP Connections…………………………………………………………..19
Hardware Description……………………………………………………..19
Port Description and Initialization………………………………………...21
5. Design Protocols for Wireless Communication………………………………22
Protocol Overview………………………………………………………...22
Network Description……………………………………………………...23
Packet Format……………………………………………………………..23
6. Evaluation
Palyload Estimation……………………………………………………….26
Error Rate of The packet Transmission…………………………………...27
7. Results ……………………..…………………………………………………….28
8. Future of this Project……………………………………………………………..28
9. References………………………………………………………………………..29

10. Appendix A- Registers…………………………………………………………...29
11. Appendix B- Code……………………………………………………………….37

1. Introduction

The NASA Robotics Alliance Cadets Program is a far-reaching, innovative project aimed at creating a new highly integrated and interactive college undergraduate level curriculum centered around Robotics and focusing on the content of at least the first two years of Mechanical Engineering, Electrical Engineering and Computer Science. This project is being co-led by Mark Leon, NASA AMES Director of Education and David Schneider of Cornell University and is supported by David Lavery, NASA Program Executive of Planetary and Solar Exploration. The program is being designed to be implemented at a very low start-up cost, and to make this goal obtainable, the program is being developed using low to no cost components from already developed, well-tested and robust engineering testbeds. The Microcontroller board to be used is the Oregon State University TekBots platforms, whose base kit is approximately $100.

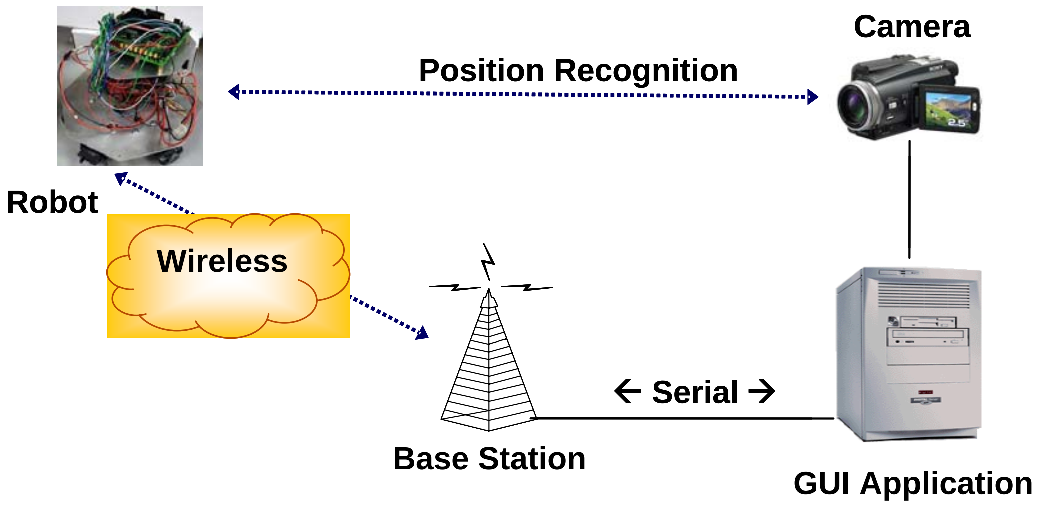


Fig. 1.1 The system structure

In this project we are trying to establish both wireless communication between the mobile Robot and the remote Base Station, and serial communication between the remote Base Station and the GUI Application shown in Fig 1.1.

The Base Station requires serial communication with the GUI Application and also needs to be hardwired with the radio packet controller, FRPC2 for wireless control. Our aim is to be able to command and control the Robot wirelessly by the GUI Application shown in Fig. 1.4.

The main task of this project is two parts: (1) to program the AVR microcontroller on both the Base Station shown in Fig. 1.3 and the Robot in Fig. 1.2 interfaced to the radio packet controller module which would enable us to wirelessly control the Robot; (2) to program the GUI Application which would enable us to serially control the Base Station.

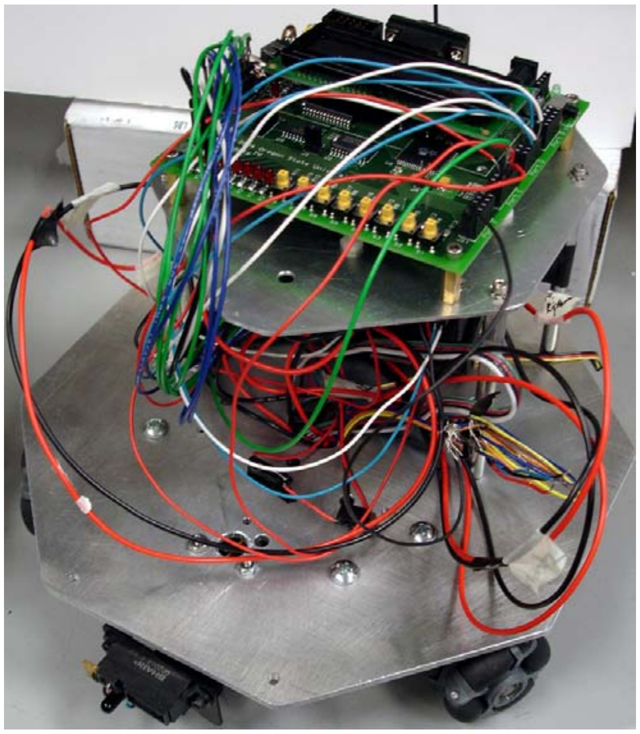


Fig. 1.2 The Robot

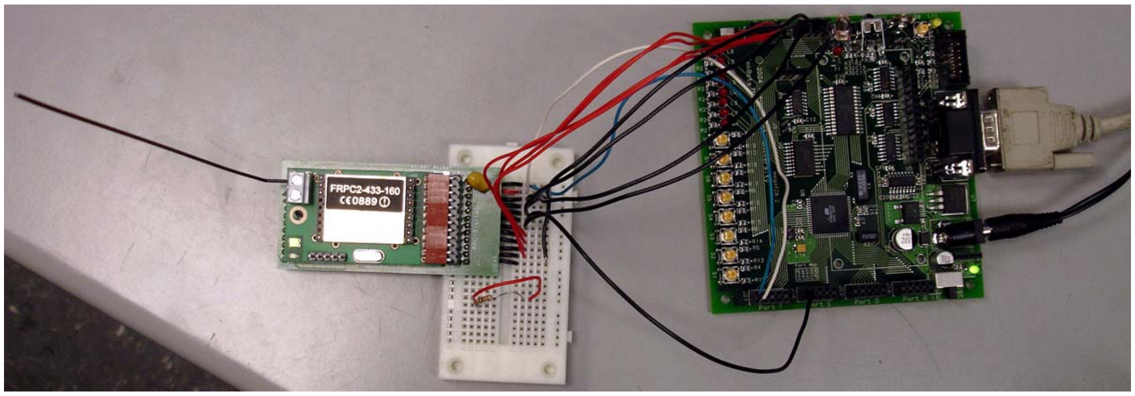
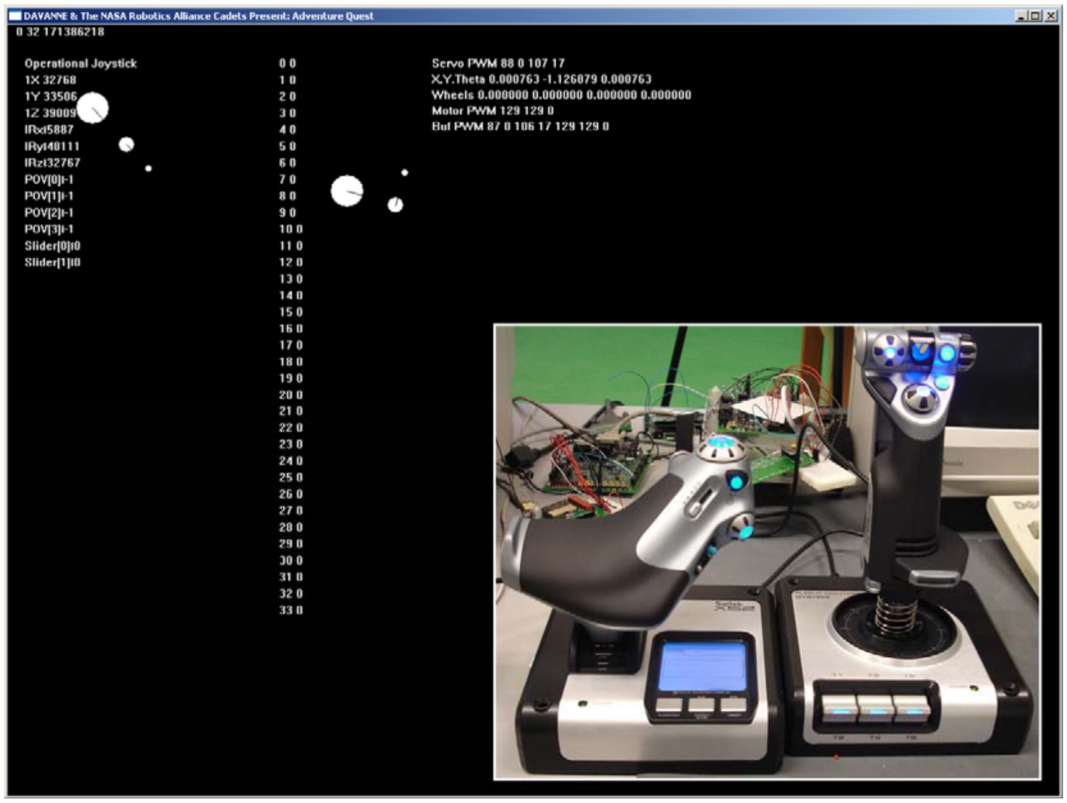


Fig. 1.3 The Base Station

The GUI Application is operated by a joystick with two components as shown in Fig. 1.4. By moving the joystick, the GUI Application commands movement such as go forward, go backward, left turn, right turn as well as the operation of modular, optional tools.



```
DAVANNE & The NASA Robotics Alliance Cadets Present: Adventure Quest
0 32 171386218

Operational Joystick        0 0          Servo PWM 88 0 107 17
1X 32768                    1 0          X.Y.Theta 0.000763 -1.126879 0.000763
1Y 33506                    2 0          Wheels 0.000000 0.000000 0.000000 0.000000
1Z 39009                    3 0          Motor PWM 129 129 0
IRxl5887                    4 0          Buf PWM 87 0 106 17 129 129 0
IRyl48111                   5 0
IRzl32767                   6 0
POV[0]l-1                   7 0
POV[1]l-1                   8 0
POV[2]l-1                   9 0
POV[3]l-1                   10 0
Slider[0]l0                 11 0
Slider[1]l0                 12 0
                            13 0
                            14 0
                            15 0
                            16 0
                            17 0
                            18 0
                            19 0
                            20 0
                            21 0
                            22 0
                            23 0
                            24 0
                            25 0
                            26 0
                            27 0
                            28 0
                            29 0
                            30 0
                            31 0
                            32 0
                            33 0
```

Fig. 1.4 The GUI Application

The data transmission from the GUI Application to the Robot would be implemented and the reverse transmission from the Robot to the GUI Application will remain for the future work. However, possible enhancements will be discussed in the end of the report.

2. Project Requirements

Objectives:

- Benchmark the wireless modules.
- Learn the working and functioning of the Tekbot Microcontroller board.
- Interface the Tekbot microcontroller board with a radio packet controller module.
- Interface the Tekbot microcontroller board with a serial port.
- Build a transmitter consisting of the Tekbot microcontroller board interfaced with a FRPC2 radio packet controller.
- Program the microcontroller so as to wirelessly control the Cadets Robot.
- Program a multithreaded GUI Application so as to serially communicate with the Base Station.
- Establish the structure of packets for the wireless and serial transmission.
- Solder a tiny26 Microcontroller board
- Develop efficient code along with adequate internal and external documentation.
- Measure the performance and error rate of the packet transmission.

3. Radio Packet Controller Module

The Radiometrix RPC transceiver has been chosen by the RoboCup team as the communication link for several previous years. Though it has proven to be an adequate system, during the competition, many teams used the same RPC unit and were sharing the same frequencies in their Robots causing interference and drop-outs, especially when teams were testing their own Robots. Based on the RoboCup team's trouble, it was decided to develop a metric to find better alternatives. This entire metric and its results are mentioned in Section 3.1.

3.1 Design Requirements

The previous system using the RPC is susceptible to interferences caused by the sharing of same frequencies. Hence we need to find a new system that is not only preventing such occurrences but also reliable during the uncertain conditions. The system should also have error-checking capabilities.

With regard to Performance, since our Robotic system is real-time, there is a need for a low latency system, which is affected by the delay in processing and transmission. Latency in processing is defined to be the time taken to process the signal and high transmission rate is defined as the time taken for actual transmission.

With regard to Ease of Implementation, the new system should be easy to integrate into the current Robotic unit so that no major redesigning of the boards etc is needed. It should also easy for the team to pick up and learn without needed a lot of expertise since student projects are time-constrained. Its implantation should not delay the development of other parts of the project.

With regard to Cost, we need to consider the cost of new modules and new development kits. It has been suggested by past teams that full-duplexity in the communication link is desired. The advantage of having a full-duplex system is that information can be transmitted and received simultaneously from the Robot to the GUI Application through the Base Station, which might be useful in Robots' control.

However, through benchmarking other wireless modules shown in Table 3.1, we decided that first we adopt the Fast RPC (FRPC2) whose performance is better than RPC. The advantage choosing the FRPC2 is the low cost, better performance, error-checking available, and ease of implementation. Later on, we plan to upgrade the wireless module to a system even which will be better than FRPC2.

The chip with the FRPC2 is the BiM which is more reliable, easy to establish a communication link and it has lower latency. In addition, the BiM is easy to change Frequency without code compilation. In general, the BiM outperforms high-throughput. Other consideration to choose the wireless module is that it should be individually addressable, and have high resistance to interference.

One of the reasons that we choose FRPC2 is that the overhead cost in significant speed and low complexity outweighs the advantage of having a full-duplex wireless channel. The entire metric is mentioned in Section 3.1.

company	**component**	**feature**	**Baud Rate**	**Frequency band**	**operating range**
Radiometrix	**RPC-418-40**	IC+**BiM**, UK version	40kbit/s (half duplex)	433 MHz	30m ~ 120m
Radiometrix	**RPC-433-40**	IC+**BiM**, Euro version	40kbit/s (half duplex)	433 MHz	30m ~ 120m
Radiometrix	**FRPC2-433-160**	IC+**BiM2**-433-160	160kbit/s (half duplex)	433 MHz	30m ~ 200m
Parallax	**RF Transceiver Package**	RF Tranceiver	1200 ~ 19.2kbit/s	433 MHz	30m ~ 200m
ABACOM TECHNOLOGIES	**SILRX-433-A TXM-433-A**	UHF FM Transceiver	5 kbps (ver.A), 10kbps (ver.F)	433 MHz	50m ~ 200m
RCS	**SE200-A1**	UHF FM Transceiver	up to 10kbit/s (half duplex)	433.92MHz ISM-band	152.4m
AEROCOMM	**LX2400S-3A**	Frequency-Hopping Spread-Spectrum	144kbit/s	2.402 ~ 2.479GHz	30.48m ~ 152.4m
AEROCOMM	**LX2400S-10**	Frequency-Hopping Spread-	144kbit/s	2.402 ~ 2.479GHz	30.48m ~ 152.4m

		Spectrum			
AEROCOMM	**LX2400S-150**	Frequency-Hopping Spread-Spectrum	144kbit/s	2.402 ~ 2.479GHz	30.48m ~ 152.4m
Talisman Electronics	**BlueWAVE RS232 DCE**	Bluetooth Specification v1.1	244byte/s ~ 1.38Mbbyte/s (full duplex)	2.4GHz	10m ~ 100m
Broadcom	**BCM 4306 chipset**	IEEE 802.11b/g	1-54 Mbyte/s (full duplex)	4.900 to 5.850 GHz and 2.300 to 2.500 GHz	18m ~ 36m (a) / 27m ~ 45m (b/g)
Atheros	**AR5002X 802**	IEEE 802.11a/b/g	1-54 Mbyte/s (full duplex)	4.900 to 5.850 GHz and 2.300 to 2.500 GHz	18m ~ 36m (a) / 27m ~ 45m (b/g)
Maxstream	**XStream RS-232/RS-485**	RF Modems	9600bit/s or 19.2 kbit/s (full duplex)	900 MHz or 2.4 GHz	18m ~ 36m (a) / 27m ~ 45m (b/g)

Table 3.1 Benchmark wireless modules

company	**component**	**feature**	**Baud Rate**	**Frequency band**	**operating range**
Neteon.Net	**LS100W**	Serial(RS232) to IEEE 802.11**b**	up to 115Kbps (Full duplex)	2400 ~ 2485MHz	27m ~ 45m
Data Hunter	**O.E.M. WLAN 802.11b " Mini-b" Serial**	Serial(RS232) to IEEE 802.11**b**	9600 to 115 kbps (Full duplex)	2.4 ~ 2.4835 GHz	30m ~ 300m
Data Hunter	**802.11"g" Wireless RS232**	Serial(RS232) to IEEE 802.11**b/g**	up to 54Mbps (Full duplex)	2.4 ~ 2.4835 GHz	27m ~ 45m
Parani	**ESD100 /110**	Serial(RS232) to Bluetooth v.1.2 Class I	1200 ~ 230 Kbps (Full duplex)	2.4GHz	**100m~1000m**
Parani	**ESD200 /210**	Serial(RS232) to Bluetooth v.1.2 Class II	1200 ~ 230 Kbps (Full duplex)	2.4GHz	30m ~300m

Table 3.2 Benchmark alternative wireless modules

Moreover, as a future upgrade the Bluetooth, IEEE 802.11a/b/g and Wi-Fi will be considered as shown in Table 3.2 and Table 3.3.

Protocol	**Frequency**	**Max Through-put**	**Actual Through-put**	**Signal Range**	**Latency**
Bluetooth	2.4 GHz	1.2 Mbps full-duplex	680 kbps	10-100 m	< 2 ms
802.11a	5 GHz	53 Mbps full-duplex	21 Mbps	18-36 m	< 2.5 ms
802.11b	2.4 GHz	11 Mbps full-duplex	4 Mbps	27 ~ 45 m	< 5 ms
802.11g	2.4 GHz	54 Mbps full-duplex	17 Mbps	27 ~ 45 m	< 2.5 ms

Table 3.3 Wireless Standard

3.2 Fast Radio Packet Controller (FRPC2)

The Fast Radio Packet Controller (FRPC2) module shown in Fig. 3.2.1 is an intelligent transceiver which enables a radio network/link to be simply implemented between a number of digital devices. The module combines a UHF radio transceiver and a 160kbit/s packet controller.

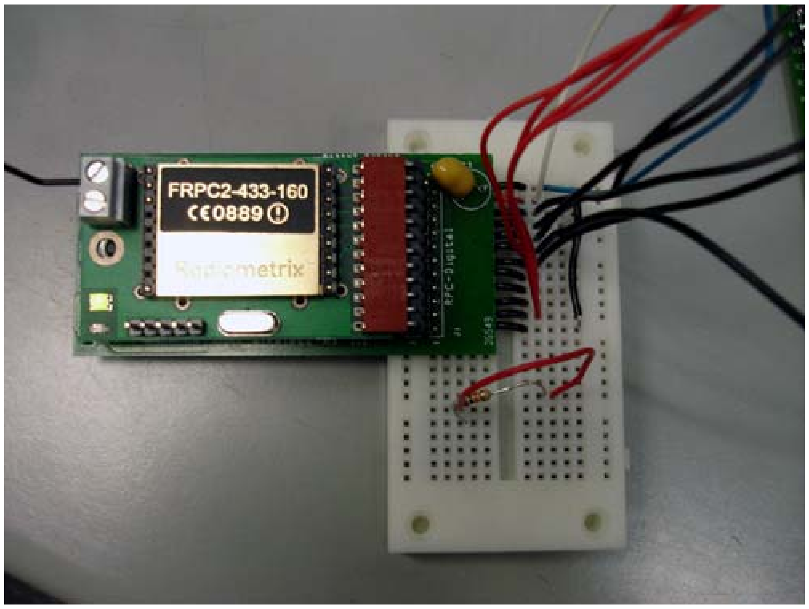


Fig. 3.2.1 FRPC2-433-160

The FRPC2 is a self-contained plug-on radio port which requires only a simple antenna, 5V supply and a byte-wide I/O port on a microcontroller (or bi-directional PC port). The module provides all the RF circuits and processor intensive low level packet formatting and packet recovery functions required to inter-connect a number of Microcontrollers in a radio network. More details can be found in Section 3.3 and 3.4.

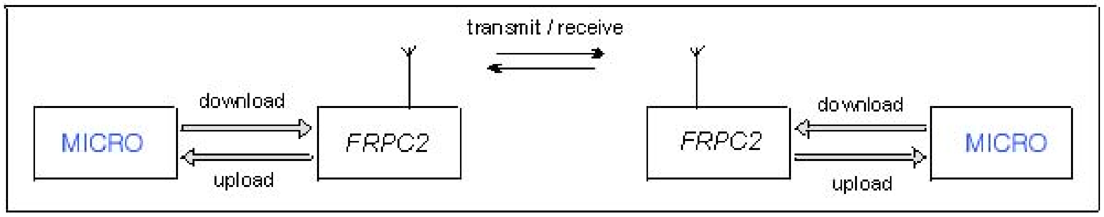


Fig. 3.2.2 FRPC2 + Micro µController

The flow of data transmission begins with a data packet of 1 to 60 bytes downloaded by a Microcontroller into the FRPC2's packet buffer. This is then transmitted by the FRPC2's transceiver and will appear in the receive buffer of all the FRPC2's within radio range shown in Fig. 3.2.2. The FRPC2’s block diagram is also shown in Fig. 3.2.3.

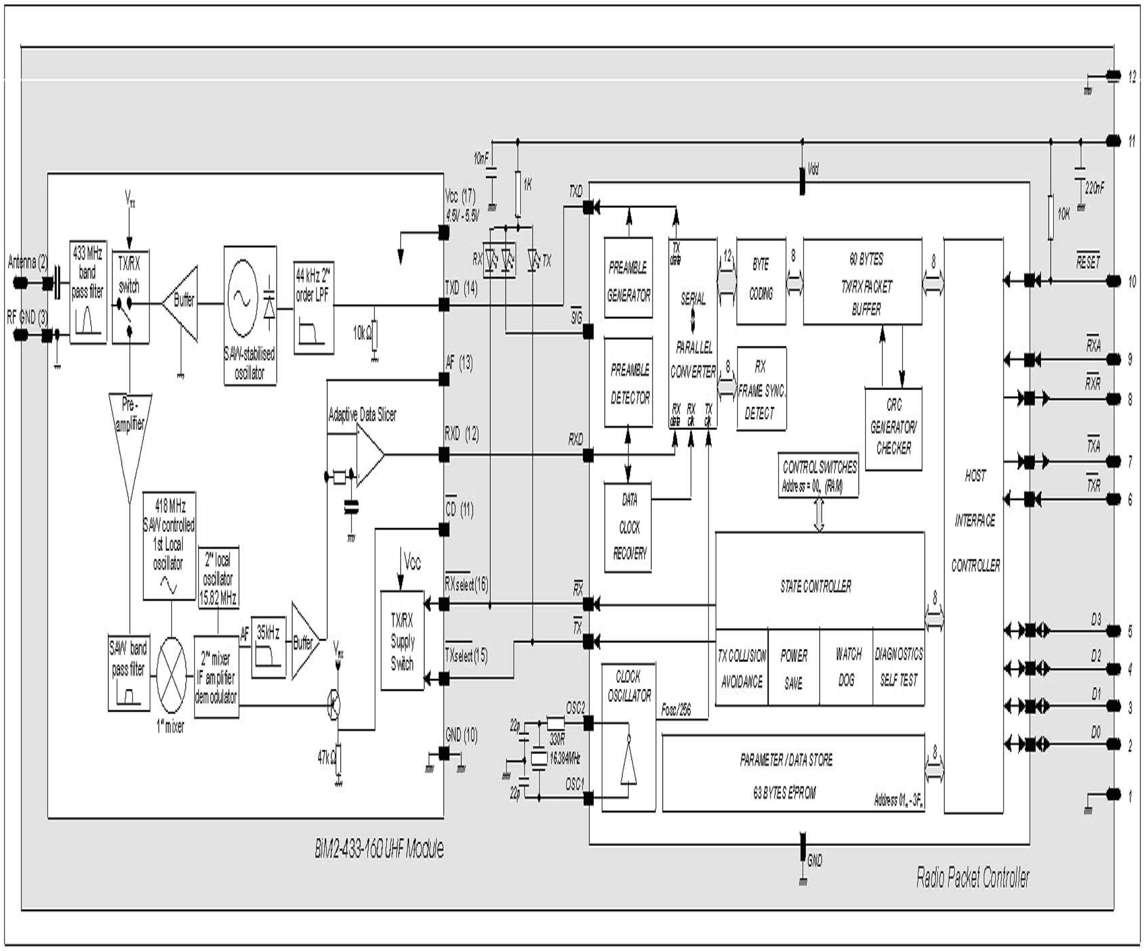


Fig. 3.2.3 FRPC2-433-160 block diagram

A data packet received by the FRPC2's transceiver is then decoded using the FRPC2’s internally provided function, stored in a packet buffer and the Microcontroller receives a signal that a valid packet is waiting to be uploaded. The output of pins is shown in Fig. 3.2.4 and will be used to help describe this process in future detail in the next Section 3.3.

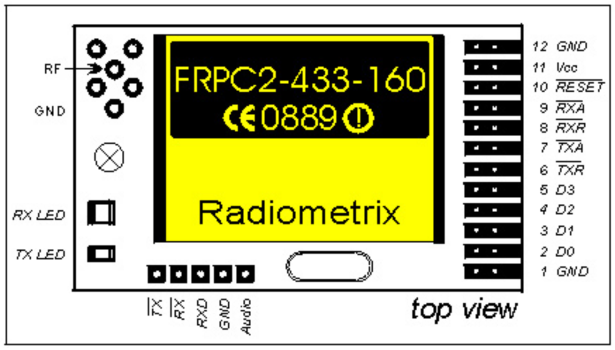


Fig. 3.2.4 The FRPC2's pin-out

3.3 SIGNALS

In order to set up the signal ports, it is recommended that the FRPC2 be assigned to a single byte wide bi-directional I/O port on the Microcontroller board. The 8 lines that must be connected consisted 4 data lines D0 to D4 and 4 handshake lines called TXR, TXA, RXA, RXR. The table 3.3.1 below gives the description of the pins in Fig. 3.2.4 and their use will be discussed in Section 3.5.

Important points are: (1) the 4 Handshake lines are active low; (2) the 4 Data lines true data; (3) logic levels are 5 volt CMOS; (4) input pins have a weak pull-up internally.

The port must be such that the four data lines can be direction controlled without affecting the other four handshake lines which are used in the handshake between FRPC2 modules.

In short, the pin TXR signals a data transfer request from the Microcontroller to the FRPC2. The TXA pin accepts the handshake from the Microcontroller.

Similarly, the RXR pin signals for a transfer request at data from the FRPC2 to the Microcontroller. And the RXA pin is used by the Microcontroller to accept the handshake from the FRPC2. D0 to D3 indicates four bit bi-direction data bus which has tri-state between packet transfers, driven on receipt for accept signal until packet transfer is complete shown in Table 3.3.1.

Pin name	Pin no.	Pin function	I/O	Description
TXR	6	TX Request	I/P	Data transfer request from Microcontroller to FRPC2
TXA	7	TX Accept	O/P	Data accept handshake back to Microcontroller

RXR	8	RX Request	O/P	Data transfer request from FRPC2 to Microcontroller
RXA	9	RX Accept	I/P	Data accept handshake back to FRPC2
D0	2	Data 0 (4)	Bi-dir	4 bit bi-directional data bus. Tri-state between packet transfers, driven on receipt for Accept signal until packet transfer is complete.
D1	3	Data 1 (5)	Bi-dir	
D2	4	Data 2 (6)	Bi-dir	
D3	5	Data 3 (7)	Bi-dir	

Table 3.3.1 The pins description

3.4 FUNCTIONAL DESCRIPTION

The whole packet structure of the FRPC2 is detailed in Fig. 3.4.1. On receipt of a packet downloaded by the Microcontroller, the FRPC2 will append to the packet: its own preamble, start byte and an error check code. The packet is then coded for security and mark; space balanced and transmitted as a 160kbit/s synchronous stream. One of four methods of collision avoidance (Listen Before TX) may be user selected, which will be discussed in details in Section 3.5.3.

When not in transmit mode, the FRPC2 continuously searches the radio noise for valid preamble. On detection of a preamble, the FRPC2 synchronizes to the in-coming data stream, decodes the data and validates the check sum. The Microcontroller is then signaled that a valid packet is waiting to be unloaded. The format of the packet is entirely of the users' determination except the 1st byte (the Control Byte shown in Fig. 3.6.1) which must specify the packet type (control or data) and the packet size. A valid received packet is presented back to the Microcontroller in exactly the same form as it is discussed in Section 5.3.

To preserve versatility, the FRPC2 does not generate routing information (i.e. source/ destination addresses) nor does it handshake packets. These network specific functions are left for the Microcontroller to perform.

Additional features of the FRPC2 include extensive diagnostic/debug functions for evaluation and debugging of the radio and Microcontroller driver software, a built in self test function and a sleep mode / wake-up mechanism which may be programmed to reduce the average current to less than 100μA. The operating parameters are fully programmable by the Microcontroller and held in EEPROM, the Microcontroller may also use the EEPROM as a general purpose non-volatile store for addresses, routing information etc. However, in this project these features are not currently used.

Initialization of packet transmission is shown in Source Code 3.4.1 which initializes the ports that FRPC2 use.

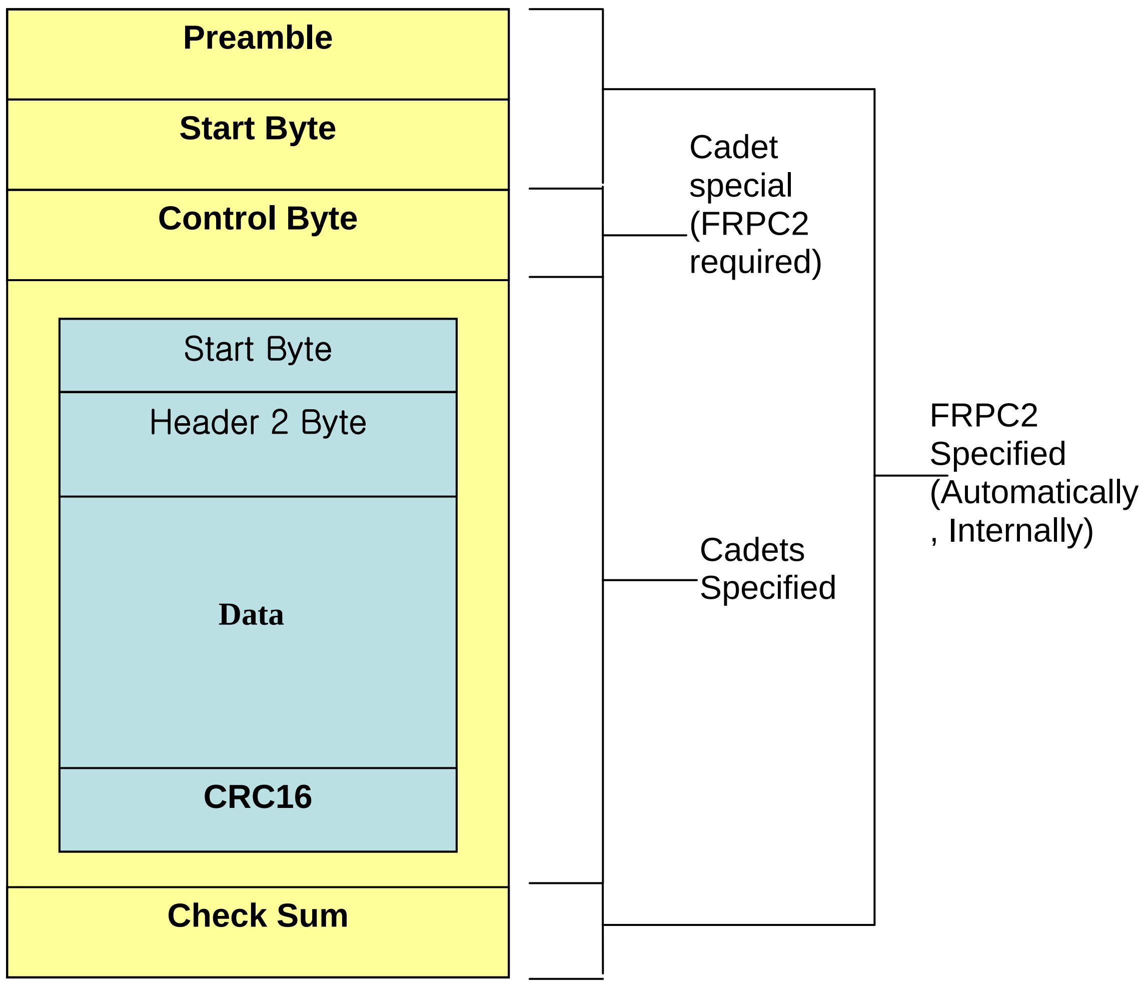


Fig. 3.4.1 FRPC2's packet

```
void packet_init(void)
{

   UCSR0B |= _BV(RXCIE0) | _BV(RXEN0) | _BV(TXEN0);
   UCSR0C |= _BV(UCSZ01) | _BV(UCSZ00);
   UBRR0L = 51;
   UBRR0H = 0;

   UDR0 = 'S';
}
```

Source Code 3.4.1 Initialization of Packet Transmission

3.5 OPERATING STATES

The FRPC2 has four normal operating states: (1) IDLE/SLEEP; (2) MICROCONTROLLER-TRANSFERS; (3) TRANSMIT; (4) RECEIVE. For our requirements we will not be operating the FRPC2's in the IDLE/SLEEP state. Since FRPC2 has half-duplex feature, a packet can't be sent and received simultaneously.

3.5.1 IDLE/SLEEP

The IDLE state is the quiescent/rest state of the FRPC2. In IDLE the FRPC2 enables the receiver and continuously searches the radio noise for message preamble. If the power saving modes have been enabled, the FRPC2 will pulse the receiver on, check for preamble and go back to SLEEP if nothing is found. The 'ON' time is 2.5ms, OFF time is programmable in the FRPC2's EEPROM and can vary between 22 ms and 181ms. The TX Request line from the Microcontroller is constantly monitored and will be acted upon if found active (low). A TX Request will immediately wake the FRPC2 up from SLEEP mode.

3.5.2 MICROCONTROLLER-TRANSFERS

If the Microcontroller sets the TX Request line low a data transfer from the

Microcontroller to the FRPC2 will be initiated. Similarly the FRPC2 will pull RX Request low when it requires transferring data to the Microcontroller.

The transfer protocol is fully asynchronous, i.e. the Microcontroller may service another interrupt and then continue with the FRPC2 transfer. It is desirable that all transfers are completed quickly since the radio transceiver is disabled until either the Microcontroller to the FRPC2 or the FRPC2 to the Microcontroller transfer is completed. Typically a Microcontroller can transfer a 60 byte packet to / from the FRPC2 in under 1ms.

3.5.3 TRANSMIT

When a data packet is received from the Microcontroller, the packet - preamble, frame sync byte and an error check sum are appended by the FRPC2's internal function. For mark such as space balance, the packet is then coded and transmitted. In 6ms of TX air time (60 byte data @ 160kb/s + 1.25ms preamble), a full 60 byte packet can be transmitted

Collision avoidance (Listen before transmit) functions can be enabled to prevent loss of packets. Data packets may be sent with either normal or extended preamble. Extended preamble is used if the remote FRPC2 is in power save mode. Extended preamble length can be changed in the EEPROM memory.

3.5.4 RECEIVE

On detection of preamble from the radio receiver, the FRPC2 will phase lock, decode and error check the incoming synchronous data stream. If it is successful, the data is then placed in a buffer and the RX Request line is pulled low to signal to the Microcontroller that a valid packet waits to be uploaded to the Microcontroller shown in Fig. 3.5.1.

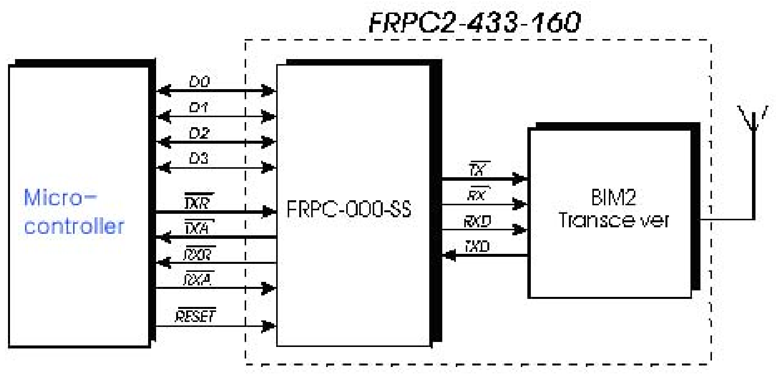


Fig. 3.5.1 Microcontroller to FRPC2 connection

4. ATMega128.2 Microcontroller Board

4.1 Introduction

The ATmega128.1 board was developed within the TekBots group at the Electrical and Computer Engineering department at Oregon State University. The mega128.1 board is intended to be used as a tool for learning assembly, C programming, and microcontroller architecture. Besides a tool for learning, it also provides a stable platform for the development of other projects requiring a microcontroller in Fig. 4.1.1.

Features of ATMega128.2 are: (1) ATmega128 AVR 8-Bit RISC microcontroller; (2) 32Kbyte external RAM (3) (ISP) in system programming via PC parallel port; (4) RS-232 port; (5) IR transmitter and receiver; (6) eight push buttons; (7) eight LEDs for general use.

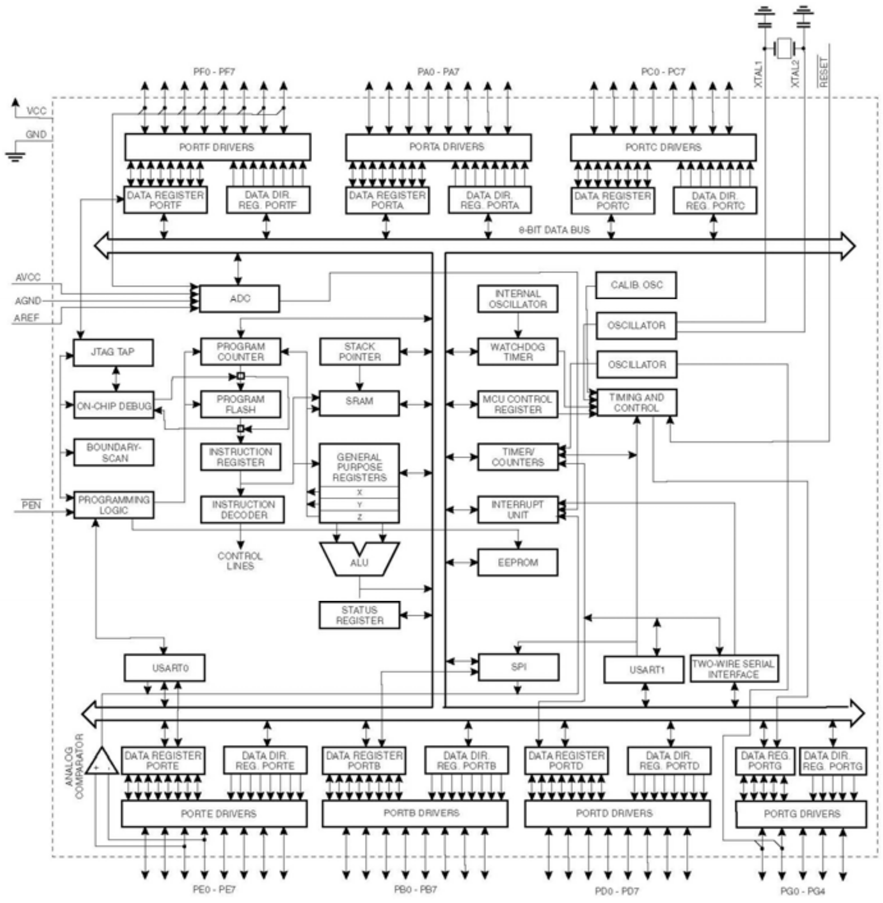


Fig. 4.1.1 Block diagram of ATMega128.2

4.2 ISP Connection

At the end of the 10 Pin ribbon cable, there are two female headers in Fig. 4.2.1 and Fig. 4.2.2. Both headers are notched to prevent them from being connected backwards. Connect the one end of the ribbon cable to J22 on the mega128.1 board, and the other end to J1 on the DB25 dongle. Note that J1 on the dongle board does not have a notched shroud but the notch is clearly indicated on the silkscreen. Lastly connect the DB25 dongle to the parallel port of the computer. In most cases this would be LPT1.

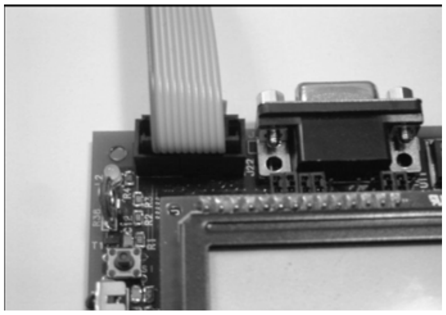


Fig. 4.2.1 ISP Connector J22

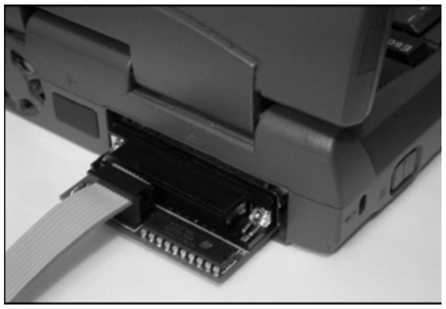


Fig. 4.2.2 DB25 Dongle

4.3 HARDWARE DESCRIPTION

The following diagram in Fig. 4.3.1 gives the complete hardware description of the microcontroller board with all the associated interfaces and connections. The hardware picture is shown in Fig. 4.3.2.

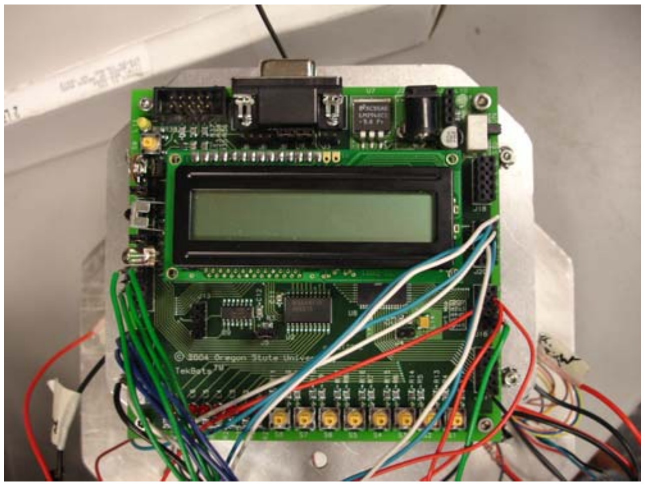


Fig. 4.3.1 Hardware picture

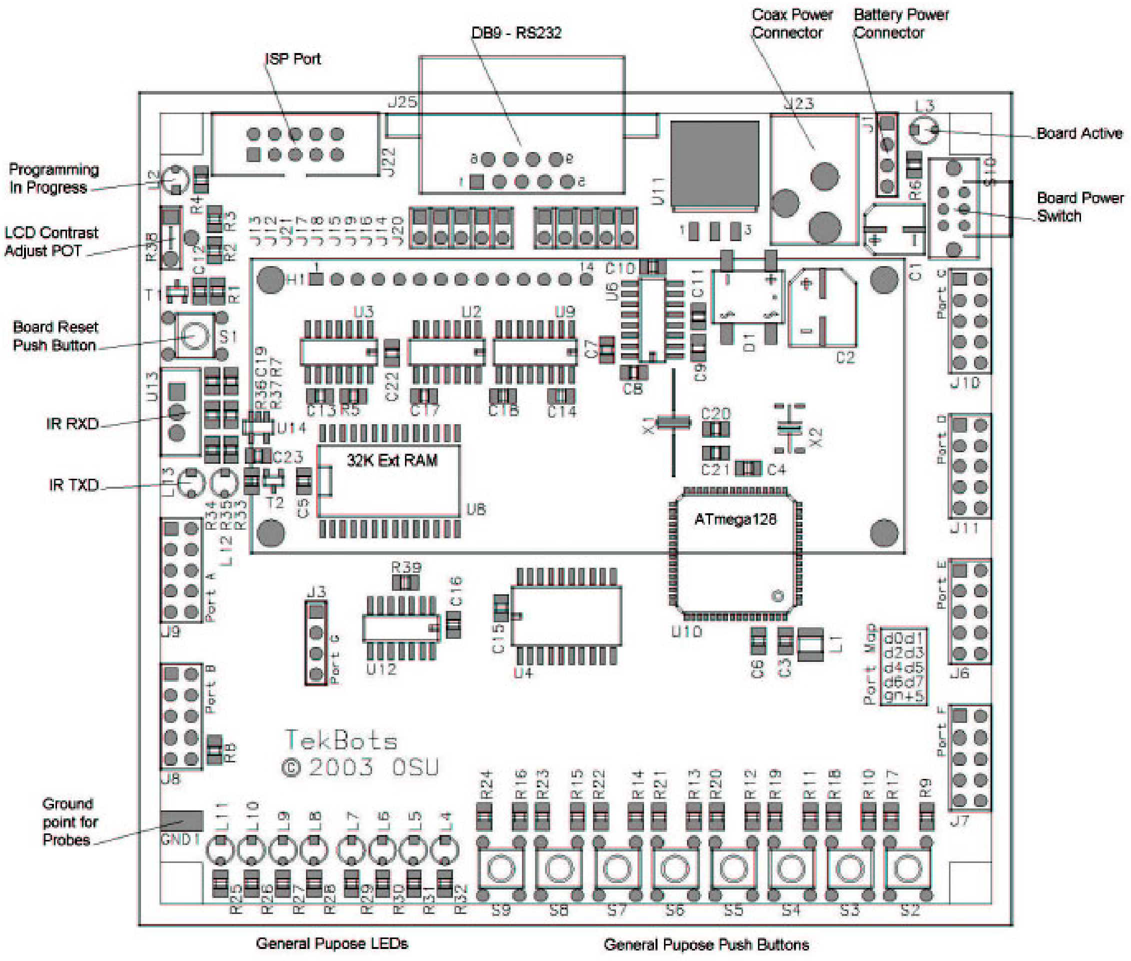


Fig. 4.3.2 Hardware diagram

4.3.1 General Purpose LEDs

There are 8 LEDs on the ATmega128.1 board, L4 through L11, that are connected to Port B and are driven directly by the microcontroller. These are high efficiency LEDs that only take 2mA to drive to full brightness. Thus always having them connected to Port B does not load the Port significantly.

4.3.2 General Purpose Switches

There are 8 push button switches on the mega128.1 board. These switches, S2 through S9, are connected to Port D. An external 10K resistor is used to pull the Port pin high. When the push button is pressed, the Port pin is pulled low through a 470Ω resistor. This was done to prevent directly grounding a pin that may be configured to force a logic high.

4.3.3 Indicator LEDs

There are three LEDs on the mega128.1 board that are dedicated to indicate that specific functions are active. L3 is a greed LED and indicates that there is power connected to the board and that switch S10 is in the “on” position. Yellow LED L2 indicates that the System Programming Port is active. L12 is a red LED and indicates that the IR is transmitting.

4.3.4 Reference # Color Function

Green on L3 indicates that the board is powered. Yellow on L2 indicates that ISP is in an active status. Red on L12 indicates that IR TXD is active.

4.3.5 Reset Push Button

Pushing push button S1 resets the ATmega128 microcontroller. This reset is OR’ed within the system programming chip reset. The ATmega128 reset is active low. The figure below shows the reset circuitry.

4.4 Port Description & Initialization

The below configurations are done according to the protocols which require you to configure certain bits as input/output as can be seen in the code.

As a default, the Cadets’ Robot setup uses PORTA, as the FRPC2 controlled port. This part is initialized to hex 19. Similarly PORTB is internally connected to the LED’s, and is initialized to hex FF to set as o/p. PORTC is a data port (pins 0 through 3 are connected to data pins D0-D3 of FRPC2). PORTD is internally connected to the switches, and it is initialized to hex 00 to set as i/p. PORTF is used to power the FRPC2, and it is initialized to hex FF (pins 6, and 7 are connected to GND and VCC of the FRPC2 respectively).

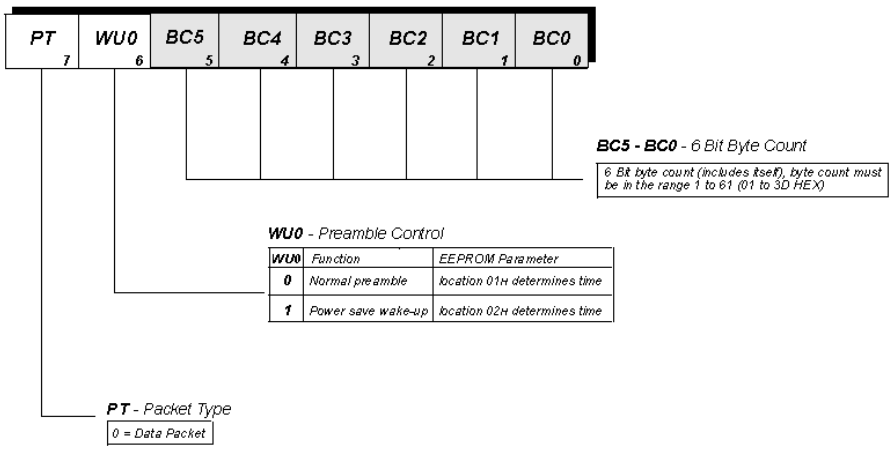

PT 7
WU0 6
BC5 5
BC4 4
BC3 3
BC2 2
BC1 1
BC0 0
BC5 - BC0 - 6 Bit Byte Count
6 Bit byte count (includes itself), byte count must be in the range 1 to 61 (01 to 3D HEX)
WU0 - Preamble Control
WU0 | Function | EEPROM Parameter
0 | Normal preamble | location 01H determines time
1 | Power save wake-up | location 02H determines time
PT - Packet Type
0 = Data Packet

Fig. 3.6.1 Control byte for data packet

The control byte shown in Fig. 3.6.1 basically defines the various modes of operation of the FRPC2. The control byte includes details such as the number of bytes to be transmitted using BC5-BC0. The control byte also details the type of data packet (PT) and the preamble control (WU0). In the Cadets system, the default was set to be bit seven was set to zero (PT=0,) since all are data packets and bits five and WH=0 to use the normal preamble. Bit 6 is also set to zero the Cadets data will never exceed x bytes. On top of the FRPC2 protocol, the Cadets system uses its own additional protocol structure consisting four bytes i.e. start byte, two packet header bytes and a stop byte. These additional four bytes are discussed in detail in Section 5.3.

5. Design Protocols for the wireless communication

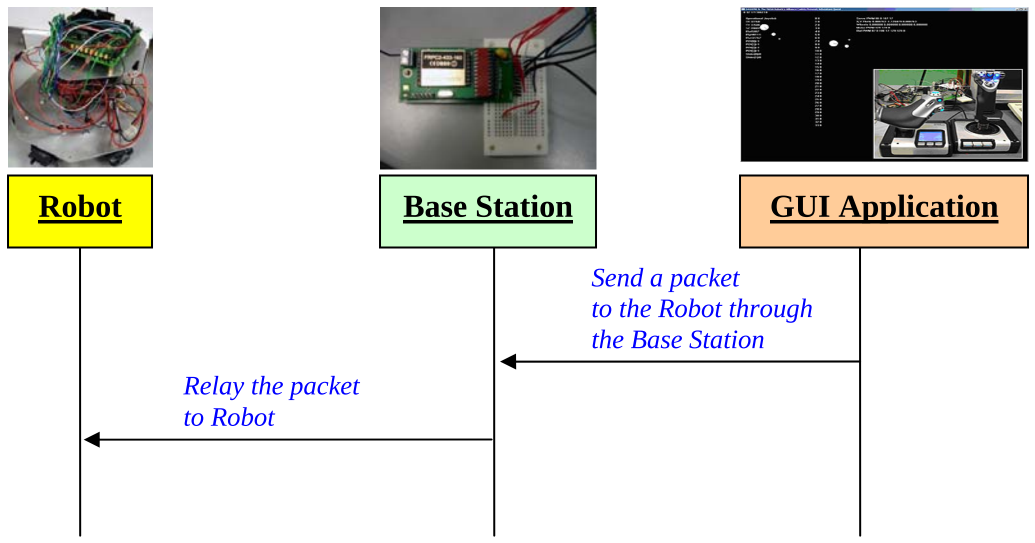

Robot
Base Station
GUI Application
Send a packet to the Robot through the Base Station
Relay the packet to Robot

Diagram 5.1 Time diagram of packet transmission in the system

In our system, we adopt the proposed protocol. The follow section provides details on this protocol.

5.1 Protocol Overview

The proposed protocol defines a simple packet syntax that is independent of the physical transport medium. The protocol is designed to support up to 15 uniquely addressed nodes where a node can be a Robot or a Base Station. Within the network, all nodes are peers, and therefore any node can communicate with any other node. In turn, individual packets can be labeled with one of the 16 FXN IDs shown in Section 5.3.

5.2 Network Description

The proposed protocol most closely fits into layer 2 (Data Link Layer) of the OSI (Open Systems Interconnection Basic Reference Model) model which is a layered, abstract description for communications and computer network protocol design, developed as part of the Open Systems Interconnection initiative shown in Diagram 5.2.1.

Although the proposed protocol can be used in a number of network topologies, it is optimized for use in a peer, asynchronous, bussed network. Every node in the network must have a unique, nonzero address. This allows every packet to carry both source and destination information. Therefore, by default, any node can send a packet to any other node. Other network configurations can be built on top of the proposed protocol (master, slave, synchronous, etc.), but they are not defined as a part of the current standard protocol.

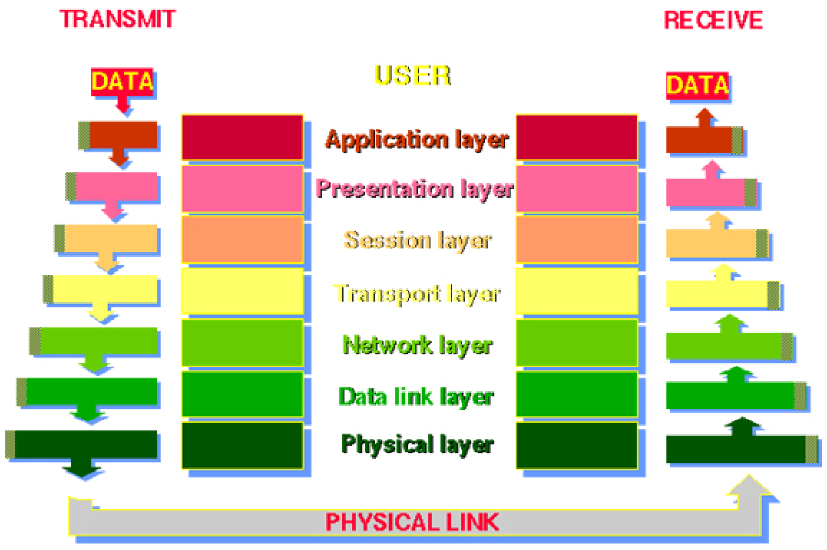


Diagram 5.2.1 OSI 7 Layers

5.3 Packet Format

In order to be as flexible as possible, the packet is byte aligned, with 8bit bytes. This makes implementation easy and efficient on a number of platforms, including the 8bit AVR microcontroller used as the main processing unit on the Robots. Byte alignment also makes the packet easily compatible with a number of physical transport layers (e.g. RS232).

A packet is made up of four sections: START, HEADER, DATA, and CHECKSUM, transmitted in that order. These sections are summarized in Table 5.3.1.

Packet Type	Bytes	Description		
START	1	Always 0x5A		
HEADER	1	DST	SRC	First byte holds destination address (DST) in upper nibble and

				source address (SRC) in lower nibble. SRC must never be 0. LEN must never be 0.
	1	FXN	LEN	Second byte holds function (FXN) in upper nibble and data length (LEN) in lower nibble. DST of 0 is general call.
DATA	LEN	Main data (D[1]...D[LEN]). Must be at least one byte of data. Organization of data is implementation dependent, but big-endian is recommended for multi-byte data.		
CHECKSUM	2	16bit cyclic redundancy check, MSB first (CRC1, CRC0). Polynomial used in $x^{16} + x^{15} + x^2 + 1$.		

Table 5.3.1 Packet structure

The START byte is used to synchronize the transmitter and the receiver. Since the START byte must always be of a known value, the receiver can know with a high degree of certainty that a packet is starting. Note, however, that the START byte value is not reserved, and that same value is allowed to appear within the HEADER, DATA, and/or CHECKSUM sections.

The HEADER section contains all important information about a packet. It contains four nibbles split between two bytes that collectively define the packet. The first byte contains source and destination address information (SRC, DST). The source address must always be the unique address of the originating node. Note that address 0 is reserved for future use and thus cannot be used as an originating address. The destination address can either be the address of the intended recipient or 0 for a general call. Every node is required to accept a general call broadcasting of a message from one node automatically. Broadcasting from one node forces all other nodes in range to switch to receive nodes. This is detailed in Section 3.2.

The second HEADER byte contains a function identifier (FXN) and the length of the payload data (LEN). The function identifier is used to specify what the payload data is, i.e. provide a label to act as a signal to the receiving node's Host code. The precise meaning of each function identifier is left up to the implementation. No standards have been established at this time.

The length parameter defines the number of payload bytes being carried in the packet. Note that a LEN of 0 is reserved for future use in general, and therefore every packet must have a minimum of 1 byte of payload. Since the length parameter is 1 nibble, the maximum payload length is 15 bytes.

The DATA section is the actual payload data. As noted, a packet can carry between 1 and 15 bytes of data. Any data values are acceptable, and the data can be structured in any manner desired. Note that although this protocol does not define any requirements for the data, it is recommended that multi-byte data (such as long integers) be transmitted big-endian where (1) big-endian means that the most significant byte of any multi-byte data field is stored at the lowest memory address, which is also the address of the larger field; (2) little-endian means that the least significant byte of any multi-byte data field is stored at the lowest memory address, which is also the address of the larger field.

The CHECKSUM section is used for transmission error detection. It holds a 16bit CRC in two bytes. The first byte holds the most significant byte and the second byte the lease significant. The polynomial used is $x^{16} + x^{15} + x^2 + 1$. Note that the CRC encompasses every byte of the packet preceding the CHECKSUM section, including the START byte.

Source Code 5.3.1 details the packet structure.

```
pkt_s_tx[0] = PKT_START_BYTE;                  // start byte
pkt_s_tx[1] = (dest<<4) | PKT_SRC_ADDR;     // destination and source address
pkt_s_tx[2] = (fxn<<4) | (len & 0x0F);
for (i = 3; i < len+3; i++)
        pkt_s_tx[i] = data[i-3];      // real data
pkt_s_tx[i++] = 0;                         // Pad with 0 for CRC generation
pkt_s_tx[i++] = 0;                         // Ditto
pkt_s_tx_size = i;                        // size of a packet

crc = 0;
for (i = 0; i < pkt_s_tx_size; i++)
        crc = _crc16_update(crc, pkt_s_tx[i]);          //calculating CHECKSUM
pkt_s_tx[pkt_s_tx_size-2] = crc>>8;
pkt_s_tx[pkt_s_tx_size-1] = crc & 0xFF;
```

Source Code 5.3.1

6. Evaluation

6.1 Payload Estimation

Theoretical system limitation for the packet transmission in each case is shown in Fig. 6.1.1 and Fig. 6.1.2.

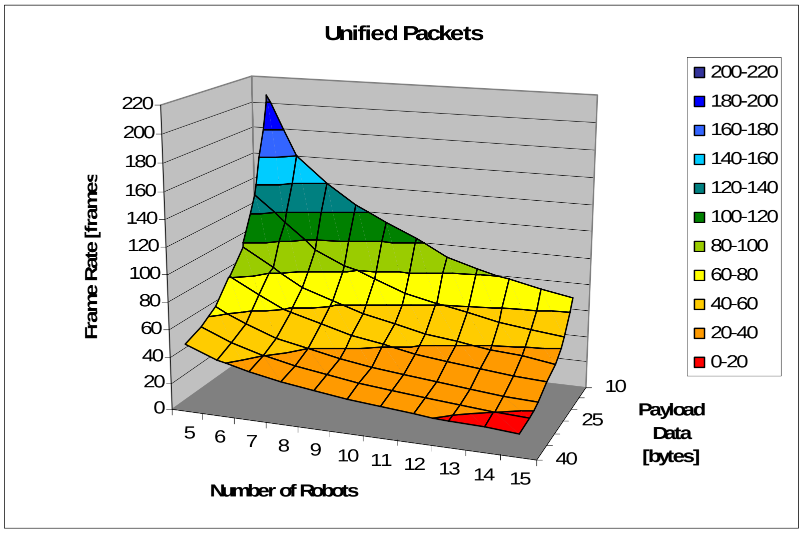


Fig. 6.1.1 Payload for unified packets

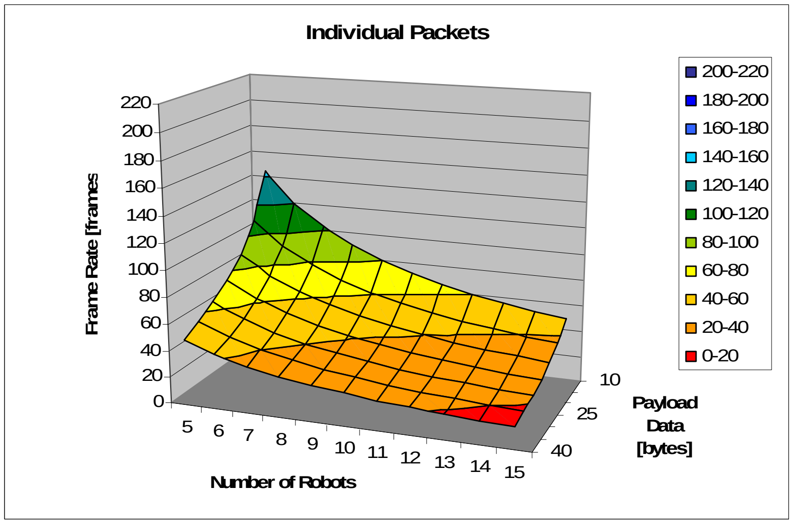


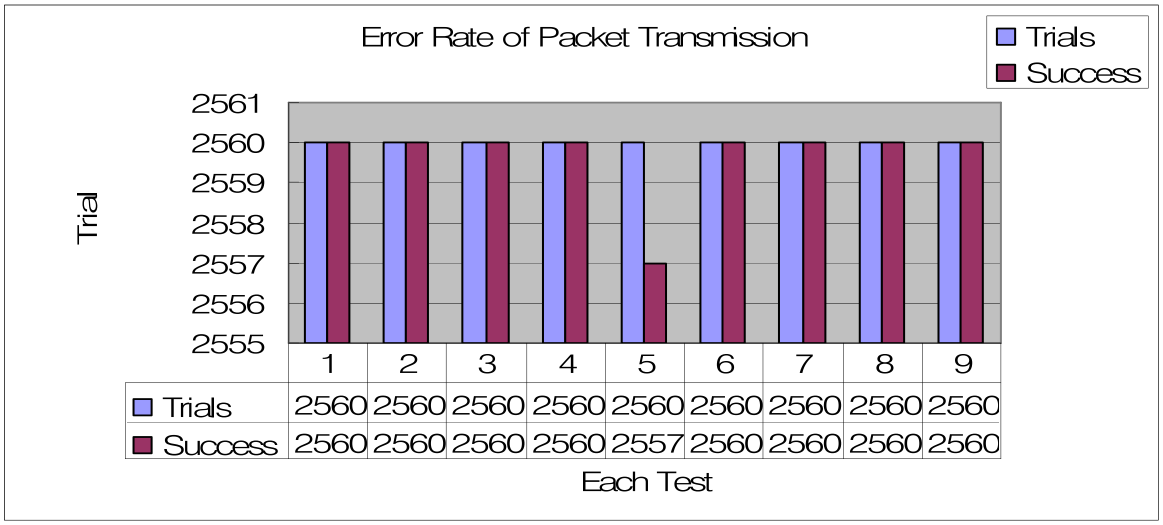
Fig. 6.1.2 Payload for individual packets

Packets/frame and bytes/frame are based on payload data [bytes] and number of the Robots shown in Fig. 6.1.1 and Fig. 6.1.2. As the number of Robots increases the frame rates decrease. As the data bytes of payload increase, the frame rates decrease as well. In both cases, how much higher the number of the frame rate is depends on the number of the Robots. With no doubt, the smaller the number of the Robots, the higher the frame rates. In the case that the payload per a Robot is less than 25 bytes, the system guarantees a frame rate of over 100.

6.2 Error Rate of the Packet Transmission

We tested packet stress to the wireless module while varying the number of Robots and the payload data. In our tests, payload was set to the 8 bytes in each packet transmission. X-axis indicates a number of Robots (up to 15 Robots). Y-axis indicates frame rates for each payload type.

The wireless parts were evaluated with CRC error checking. 23,040 packet transmissions were tested varying the distance from the Base Station to the Robots. The experiments ranged starting from 10 cm to 400 cm and in each test set the distance was increased by 50 cm. At each distance, for each test case, we sent a packet 256 times from the GUI application to the Robots. In all tests, however the payload was 8 bytes for each packet transmission. The packet transmission path was from the GUI Application to the Robots through the Base Station.



	1	2	3	4	5	6	7	8	9
Trials	2560	2560	2560	2560	2560	2560	2560	2560	2560
Success	2560	2560	2560	2560	2557	2560	2560	2560	2560

Table 6.2.1 Signal to Noise

As a result, the average of the error rate for every packet transmissions is 0.013020833 [%] which is very low. It is especially promising as the expected maximum distance in the intended application is 4 meters. In the case of the 200 cm distance, the error rate is

0.1171875 [%] which is abnormal in the data set. The problem might be caused by the FRPC2 antenna’s direction. No efforts were made to ensure a direct path between the Robots’ FRPC2 and the Base Station’s FRPC2.

7. Results

In this project, we achieved control both wireless communication between the mobile Robot and the remote Base Station, and serial communication between the remote Base Station and the GUI Application. The main task of this project was two parts: to program the AVR microcontroller on both the Base Station and the Robot interfaced to the radio packet controller module which would enable us to wirelessly control the Robot and to program the GUI Application which would enable us to serially control the Base Station. This level of completely was successfully tested on groups at up to four Robots. Hence the wireless communication and the serial communication were successful in the downlink.

8. Future of This Project

Implement uplink communication from the Robots to GUI Application through the Base Station.
Control up to 10 Robots from the GUI Application through the Base Station.
Use a secured wireless channel using encryption and decryption.
Consider larger bandwidth system should be onboard because video streaming service desired.

9. References

1. Oregon State Tekbots