PERFORMANCE EVALUATION OF A LOW COST PROCESSOR WITH WIRELESS CONNECTIVITY

A Thesis presented to the faculty of the Graduate School

University of Missouri-Columbia

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

---

by

JIN JUNG D. YOO

Dr. Harry W. Tyrer, Thesis Supervisor

JULY 2005

The undersigned, appointed by the Dean of the Graduate School,

have examined the thesis entitled.

PERFORMANCE EVALUATION OF A LOW COST PROCESSOR WITH WIRELESS CONNECTIVITY

Presented by Jin Jung D. Yoo

A candidate for the degree of Master of Science

And hereby certify that in their opinion it is worthy of acceptance.

Harry W. Tyrer, Professor, Dept. of Electrical and Computer Engineering

Dominic K. Ho, Associate Professor, Dept. of Electrical and Computer Engineering

Frank Z. Feng, Professor, Dept. of Mechanical and Aerospace Engineering

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to Dr. Harry W. Tyrer, my advisor, for his invaluable guidance and encouragement in the completion of this study.

A special thanks to Jang Tai Sung, Han Kyung Min, Kim Jay Hun and Chong Su Han for allowing me to use their equipment. I am also grateful to Yoon Sea Min and Chong Jay Kyo who proofread my writing in this thesis, and to Kim Chang Soo and Yun Jung Woo for helping me with the use of software and data analysis

My greatest debt is to my parents. Special appreciation must be expressed to them for their patience, encouragement and financial support throughout my study in the United States.

LIST OF TABLES

LIST OF ILLUSTRATIONS

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

This study presents the performance characteristics of a low cost processor acting as nodes in a sensor network. The study has four sections: The first section covers building the sensor, nodes, and the PC server. There are 3 main parts in a sensor node: wireless, communication, and processor. We selected each part based on cost, performance, and availability. Secondly, we developed the protocols (Ethernet and Round Robin) and methods for measuring the performance of the sensor systems. In the third section we measured the performance of the system's transmission time and packet collisions. We used two sensor systems with different protocols. Finally, we measured the performance of the antenna to transmit through different environments such as steel and wooden doors, unobstructed openings, walls, and floors.

1.1 Literature Review

An application for the system we developed can be found in ad-hoc networks (wireless sensor technology), swarm intelligent systems, and elder care monitoring systems. The wireless sensor technology is low-cost, low-power, small in size, and also communicates for short distances. These tiny sensor nodes, which consist of sensing,

data processing, and communicating components, leverage the idea of sensor networks based on collaborative effort of a number of nodes [1, 2].

The best known wireless sensor system is the Motes hardware, which was designed by UC Berkeley [4] and produced by Crossbow [5]. The Motes hardware system focuses on ad-hoc peer-to-peer networks instead of on-board or central processing for large amounts of data [4].

When designing the protocols for our system, we compared different protocols used in wireless sensor systems, such as LEACH [16], Directed Diffusion [17], PEGASIS [18], SPIN [19] and GEAR [20] protocols. These protocols have of design goals of minimization of energy and/or maximization of life time [21, 22]. We developed a lightweight protocol for our system with the focus on a low cost processor and a low data rate from the antenna.

Swarm intelligent systems exhibit the collective behavior of unsophisticated agents interacting locally with their environment to cause coherent, functional, global patterns to emerge. Swarm intelligence provides a basis to explore collective or distributed problem solving without centralized control or the provision of a global model [23]. Each element of the swarm is a simple processor with communication capability like our sensor nodes.

Monitoring elders with medical problems has been done with wireless sensors. The sensors are shaped like a ring to fit the finger and use Micro Electro Mechanical Systems (MEMS) to monitor ~~a~~ patients. Low power systems can reduce power consumption by 14% by using a low power algorithm [6] and some systems provide security and safety issues in the medical environment [8] Additionally, we can use a wireless sensor shaped as a shoe to measure gait parameters outside of a traditional laboratory setting [7].

1.2 Hardware Background

The hardware of the node consists of the: sensing unit, processing unit, transceiver, and power unit [1]. The sensing unit has two parts: the sensor and the analog to digital converter.

The processing unit is a low cost processor rather than a high cost processor. Although a high cost processor has a higher processing power, we want a low cost processor for lower power usage, cost effectiveness, and reduced area of the hardware. Table 1.1 displays the cost range of different processors. The processor selected has a cost of less then 50$, power usage of less then 1mA at 5V, and area of 2”×2”×1”.

Processor family	Rabbit 3000	Intel IXP465	Arm7TDM	BasicX-24	Basic stamp 2
Cost($)	15.00	142.00	250.00	49.95	49.00

Table 1.1 Cost of Processors

There are many low cost processors such as Rabbit [11], Intel family [12], Arm [13], BasicX [14], and the Basic stamp 2 [3]. The processor that meets our requirements of low cost and availability was the Basic stamp 2 processor also called the Board of Education. The Intel family and the Arm were too high in cost, while the BasicX and the Rabbit had a problem with availability and processing power.

The processing unit also has two parts: the processor and storage. Having selected the Basic Stamp 2 as the processor, we selected a transceiver that was compatible with the on board software. Storage can be added as needed by adding Random Access Memory (RAM).

CHAPTER 2: METHODS: HARDWARE, PROTOCOLS AND EXPERIMENTS

This chapter has three parts: a description of the hardware components which made up the system in this study, a description of the protocols used in the system, and a description of the simulation to evaluate the protocols and the antenna's performance. The first part (2.1 Hardware), describes the hardware components used in the sensor and the PC server. Additionally, it explains the specifications and capabilities of the hardware components in the system. The second part (2.2 Communication protocols), describes two protocols (Ethernet and Round Robin) and how one or more sensors send packets to the PC sever. The third part (2.3 Simulation methods for performance of wireless Sensor systems), describes the measurement of the performance of the system. We measured the performance of the packet transmission time and collisions with different protocols. We also measured the performance of this wireless sensor to send packets through different obstructions i.e. doors, walls, and large distances.

2.1 Hardware

The system consists of two parts, the sensor and the PC server. The sensor-system gathers data and sends it to the PC server. The PC server receives and stores data in the database, and also controls the communication of the system. Both parts have three common components: Basic Stamp 2(BS2) [3], the Board of Education [3], and the 433.92MHz Transceiver antenna [15].

The BASIC Stamp 2(Figure 2.1) is a microcontroller with 24-pins. This component is a single board computer since it has its very own processor, memory, clock, and interface (via 16 I/O pins). Table 2.1 has the specification of the BASIC Stamp 2 microcontroller [3].

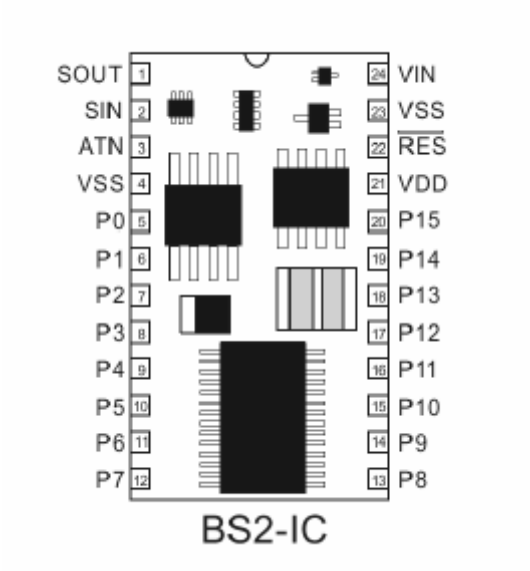


Figure 2.1 BASIC Stamp 2-pin map

Processor speed	20MHz
Program Execution speed	About 4,000 instructions / Sec
RAM size	32 Bytes (6 I/O, 26 variable)
EEPROM (program) size	2K Bytes, about 500 instructions
I/O pins	16 + 2 Dedicated serial
Voltage requirements	5 -15 vdc
Current draw at 5V	3mA Run / 50uA Sleep
PBASIC commands	42 commands
Size	1.2"*0.6"*0.4"

Table 2.1 BASIC Stamp 2 microcontroller specification

The Board of Education (Figure 2.2) is a development platform for BASIC Stamp 2. It uses a 9-volt battery as a power supply. The BASIC Stamp 2 [3] is placed ~~on~~ in the microprocessor slot on the board. We can connect components to I/O pins P0 - P15 (Figure 2.1). In addition, Vdd and Vss are available to the breadboard connections brought adjacent to 5.1 x 3.5 cm (2" x 1 3/8") breadboard area for development and test devices. All connections available for the BASIC Stamp 2 are on the Board of Education [3].

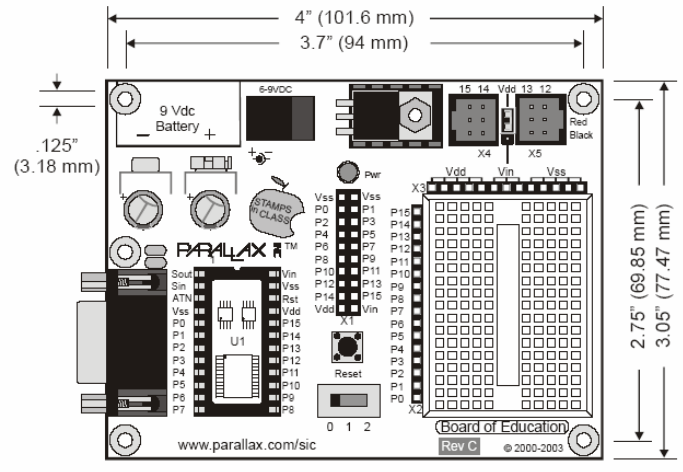


Figure 2.2 Board of Education

The 433.92MHz Transceiver antenna [15] transmits raw serial data (using the serout command) and receives raw serial data (using the serin command). The specifications indicate that the typical maximum range is 150-feet using 600 to 2400 baud rate (1200 baud rate is recommended). Our work shows this is an optimistic transmitting distance.

Figure 2.3 433.92MHz Transceiver antenna

2.1.1 Components of the sensor system

We assumed that the input to the sensor system was the analog voltage value. The sensor sends the data to the PC server using a 433.92MHz Transceiver antenna. We used a SHARP GP2D02 [10] as a distance sensor (Figure 2.4). When enabled, this sensor provides a measure of the distance to the subject, and reports it as a serial byte-value that ranges from a minimum of 10 cm (~4") to a maximum of 80 cm (~30").

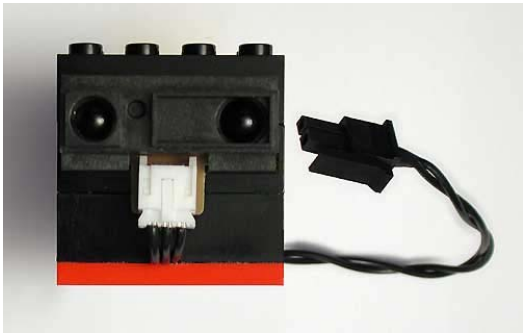


Figure 2.4 SHARP GP2D02 distance sensor

For the distance sensor, we simulated the input of data (0 V to 5 V) from a sensor using a 1K potentiometer (2.5 Figure) connected to the ADC0831 (Figure 2.6). Then, the ADC0831 [9] converted the analog voltage to 8bits of serial data (256 levels scaling 0 to 5v). Upon asserting the pin 1 of ADC0831, the data was sent to the Board of Education (pin 6 the D0 on Figure 2.6). Note that asserting is /CS that is CS = 0 V.

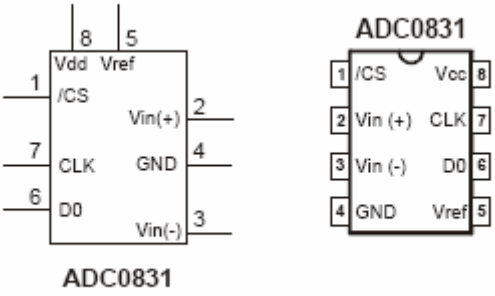


Figure 2.5 1K potentiometer

Figure 2.6 Circuit Symbol and Pin Map

Figure 2.7 is the diagram of the system that shows the connection of the sensor. Figure 2.8 is the actual sensor system.

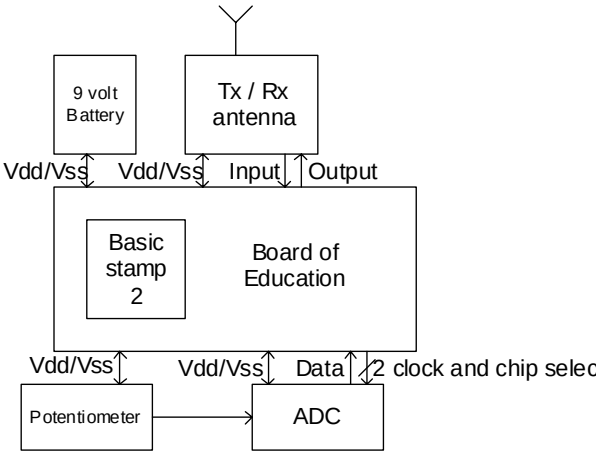


Figure 2.7 diagram of the sensor system

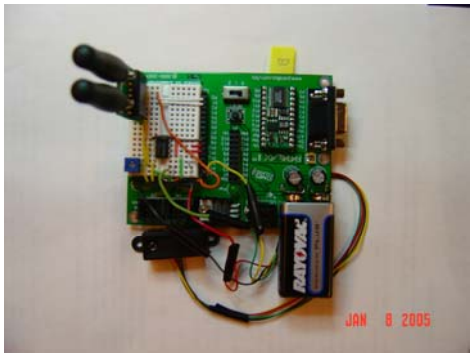


Figure 2.8 sensor system

2.1.2 Components of the PC server

We used a notebook computer as a PC server. It receives data from the sensor, identifies the sender, and controls the data traffic produced by sensors. For performance analysis, we also added an external clock. The clock is Pocket Watch B [3] (Figure 2.9). It is a real time clock module with a serial interface that adds an accurate time base to the Board of Education. It keeps track of seconds, minutes, hours, days, months, and years. For our analysis, we only need seconds and minutes. Baud rates are selectable at 2400, 4800, and 9600. The size is 2.94 w x 2.54 t x 0.64 d cm (5-pin SIP package).

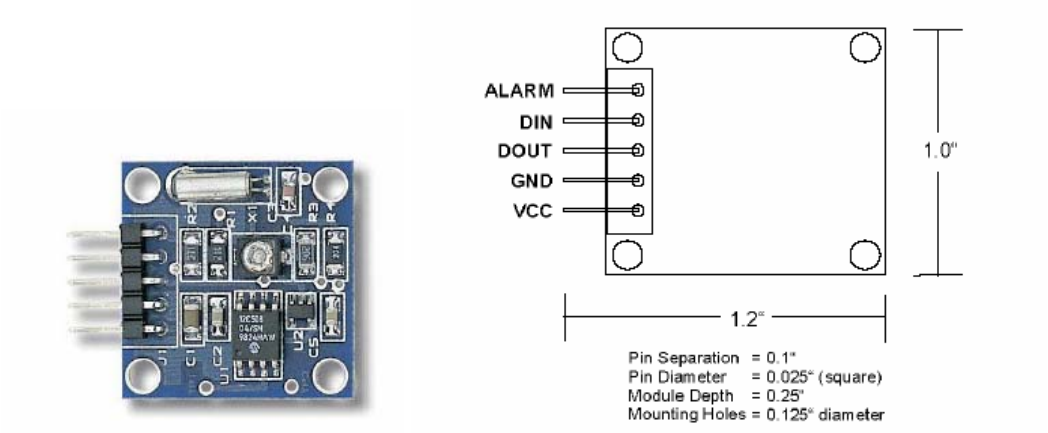


Figure 2.9 Pocket Watch B and pin map

Figure 2.10 is the diagram of the system. The connection to the PC and database (the hard drive on the PC) is done by a serial commutation via RS-232 cable.

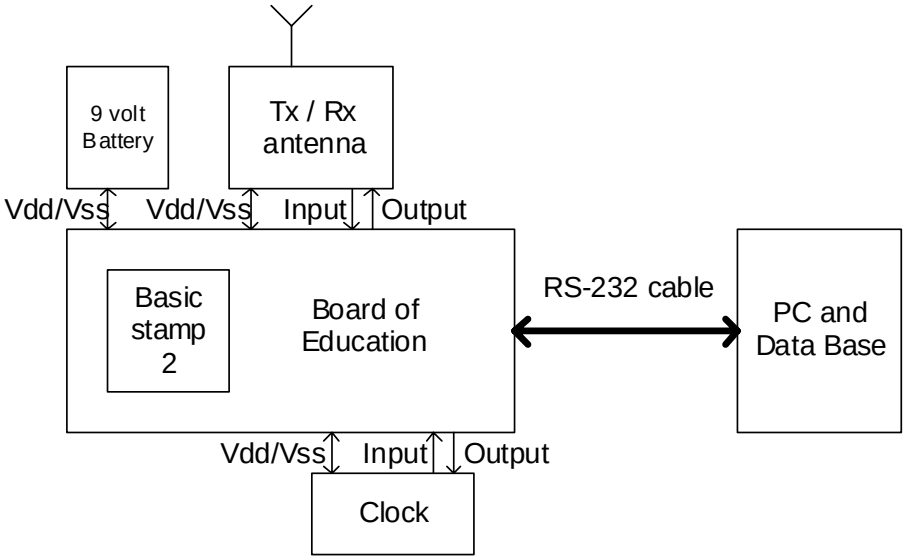


Figure 2.10 diagram of the PC server

2.2 Communication protocols

We considered two communication protocols: Ethernet and Round Robin. Ethernet is the routine protocol of the wireless communication system. Round Robin protocol is a modification of the Ethernet protocol. This protocol reduces packet loss due to collisions. Both protocols are programmed to each stamp using PBASIC.

Before we start discussing protocols, we need to define a collision event and packet formats. A collision event is when two sensors send data at the same time resulting in unrecognizable packets in the PC server (Figure 2.11).

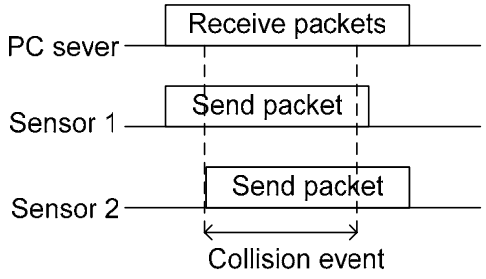


Figure 2.11 collision event

Using the embedded basic package we created several different packet formats. The packet formats are 1header 1character, 1header 2character, 1header 4character, 1header 8character, and 1header 16character (Figure 2.12).

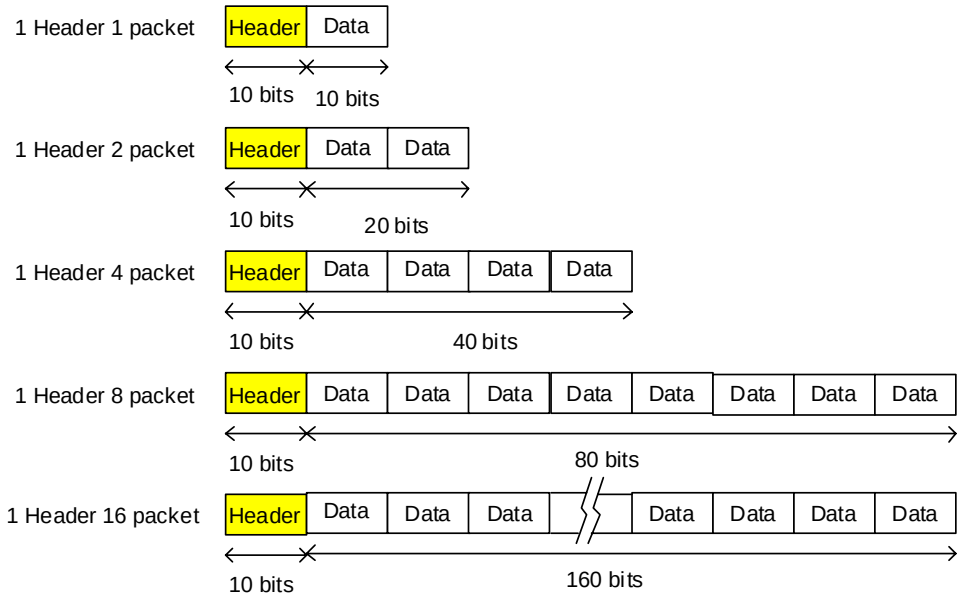


Figure 2.12 Packet formats

The sensor identification (I.D.) is the header on Figure 2.12. We used sensor I.D.'s, A and B.

2.2.1 Ethernet protocol

The Ethernet protocol operates as follows: 1. the PC server listens for a Sensor system to send a data request; 2. a data request event occurs when the subject triggers the proximity sensor; 3. then the sensor system sends a data request to the PC server, which returns an acknowledgement signal to the sensor; and 4. the sensor system then starts to send data to the PC server. Ethernet and Round Robin protocols differ in their handling of collisions. The Ethernet protocol discards data and the Round Robin protocol gets

data by a sending data requests in a preset order of all Sensor systems.

The two hardware systems (i.e. Sensor system and PC server) need to be programmed to be coherent. In order to make the systems coherent, we need to select the baud rate, packet formats, and communication timing (i.e. if one system sends wireless data, the other system is ready to receive that data) to work as one system. Sending and receiving wireless data is done using SERIN and SEROUT commands along with the corresponding baud rate. The commands are as follows:

For receiving data:

SERIN in_pin_number, RF_baud, timeout, subroutine, [in_Data]

For sending data:

SEROUT out_pin_number, RF_baud, timeout, [out_Data]

where RF_baud is the baud rate for the antenna and needs to be coherent with each other, the in_pin_number and out_pin_number are the respective pins that are connected to the input and output of the antenna, the timeout is an optional command for

jumping a subroutine when there is no input for a set period of time, and [in_Data] and [out_Data] are the variables that store the data.

The pseudo code for Ethernet protocol is explained in Table 2.2 for both the sensor system and PC server. At the beginning of the protocol, the sensor system first checks to see if a subject is in proximity of the sensor system. This is done by doing a subroutine Read1 (see Table 2.3, distances value), which gets distance data from the GP2D02 sensor and repeats this process two times. This process allows us to check to see if there is a change in distance between the first time and the second time (ES 1 on the sensor system). If there is a change in distance, then the subject is in proximity of the sensor system, which will let the sensor system follow through to the next step, ES 3, or go back to the first part of the program, ES 1 (ES 2 on the sensor system). This portion of the program is as follows:

```
GOSUB read1 ; call measurement routine
 v1 = 5 * val / 255 ; store distance in voltage
 GOSUB read1 ; call measurement routine
 v2 = 5 * val / 255 l ; store distance in voltage
 C= V1-V2 ; Get the difference of the distance
 IF C=0 THEN main ; in there is no chance then go to the start if not then next step
```

The next step is to check to see if the other systems are sending packets. This is done by receiving data from the antenna. If the other systems are sending packets then the sensor system pauses and checks again, or it goes to the next step (ES 3 on the sensor system). If no other systems are sending packets, it becomes safe to send data packets (ES 4 on the sensor system). Then the sensor system sends out a sensor I.D to the PC server to declare which sensor system is sending data (ES 5 on the sensor system). When the sensor system receives an ack (ack is for acknowledgment) (ES 6 on the sensor system) from the PC server, the sensor system sends out data packets to the PC server in the desired Packet format (i.e.: for 1header 4packets repeating step 4 times). Note that each data packet simulates the input of data (0 V to 5 V) from a sensor using a 1K potentiometer (Figure 2.5) connected to the ADC0831 (Figure 2.6) with a serial communication. The subroutine is send_data (Table 2.3).

The PC server first listens for the sensor I.D. The sensor I.D. then enables the PC server to identify which sensor system is sending data (EP 1 on the PC server). If it is a sensor I.D. then the PC server will start to record which sensor is sending data on the PC database (using SEROUT commanded at 9600 baud and the PC Server runs on StampDAQ [3]) or it will go back to the first step EP 1 (EP 2 on the PC server). Then the PC server will send out an ack (EP 3 on the PC server) to the sensor system. The

sensor system will then send data to the PC server, which will start to receive and record (on the PC database) data in the desired packet formats, and then return to the first process after receiving all the packets (EP 4 and EP 5 on the PC server). This is called the subroutine main_x.

Sensor System	PC server
(ES1) The distance sensor checks if a subject is in the proximity→ (A on Figure 2.13) (ES 2) IF there is a subject then DO(ES 3) ELSE GOTO(ES 1) (ES 3) The TX/RX checks if other systems are sending packets → (B on Figure 2.13) (ES 4) IF the TX/RX does receive input PAUSE and GOTO (ES 3) ELSE DO (5) (ES 5) The TX/RX OUTPUT sensor I.D. to the Sever PC → (C on Figure 2.13) (ES 6) The DB OUTPUT ack to sensor system → (D on Figure 2.13) (ES 7) The sensor system OUTPUT data packets → (E on Figure 2.13)	(EP1) The TX/RX checks if sensor is sending sensor I.D. → (H on Figure 2.13) (EP 2) IF not do (EP 1) if yes do(EP 3) (EP 3) Send ack to sensor → (J on Figure 2.13) (EP 4) Receive data → (K on Figure 2.13) (EP 5) End of data packet IF yes GOTO (EP 1) ELSE (EP 4)

Table2.2 Pseudo code for Ethernet protocol

Subroutines	
Name	Source

Read1 [3]	LOW cl ; turn on detector for reading rl: IF IN15 = 0 THEN rl ; wait for input high SHIFTIN dt, cl,MSBPOST,[val\8] HIGH cl ; turn detector off PAUSE 1 ; let detector reset
send_data [10]	HIGH CS LOW CS ; select chip LOW CLK PULSOUT CLK, 210 ; The pulse for sending data SHIFTIN DataOutput,CLK,MSBPOST,[adcBits\8]; Shift in data v = 5 * adcBits / 255; scale the data in voltage
main_x	SEROUT 0,RF_baud,10,["a"] ;send out sensor I.D. SERIN 1,RF_baud,100,main,[STR y\1] ; receive data GOSUB get_data ; store data to PC GOTO main ; go back to main
get_data :	SERIN 1,25389,[DEC1 v] ; send data to PC

Table 2.3 Subroutines code for Ethernet protocol

2.2.2 Ethernet protocol operation

The flow chart of the Ethernet protocol is shown on Figure 2.13. The sensor system sends data to the PC server when there is a change in the data from the distance sensor (see 2.1.1 Components of the sensor). Label (A) on Figure 2.13 shows the sensor system survey of the proximity of the subjects. When a subject appears, then the sensor (B) checks to see if other sensor systems are sending packets. If the other sensor systems are sending packets, the sensor system pauses for one second (G) and then continues to check for other packets (B). If no other sensor system is sending data, then the sensor sends the sensor I.D (C) and then receives an ack (D) packet. If the sensor system does

not receive an ack, it pauses again (G). If an ack is received, the sensor system will start to send data packets (E). After sending all the data packets, it returns to the beginning (A). We programmed the sensor system to send as many data packets as desired. (i.e.: for 1header 4packets repeating step F four times)

The PC server polls until it gets a sensor I.D from the sensor system (H). If the PC server receives the sensor I.D. (K), then the PC server goes to step (I). If the data in the sensor ID is unknown, then is goes back to (H). After we stored the sensor I.D. on the database, we sent out an ack signal (K). After the sensor system gets the ack, the PC server receives and stores the data packets. After receiving all the data packets, it then goes to the starting point (H). Appendix A contains the program for this protocol.

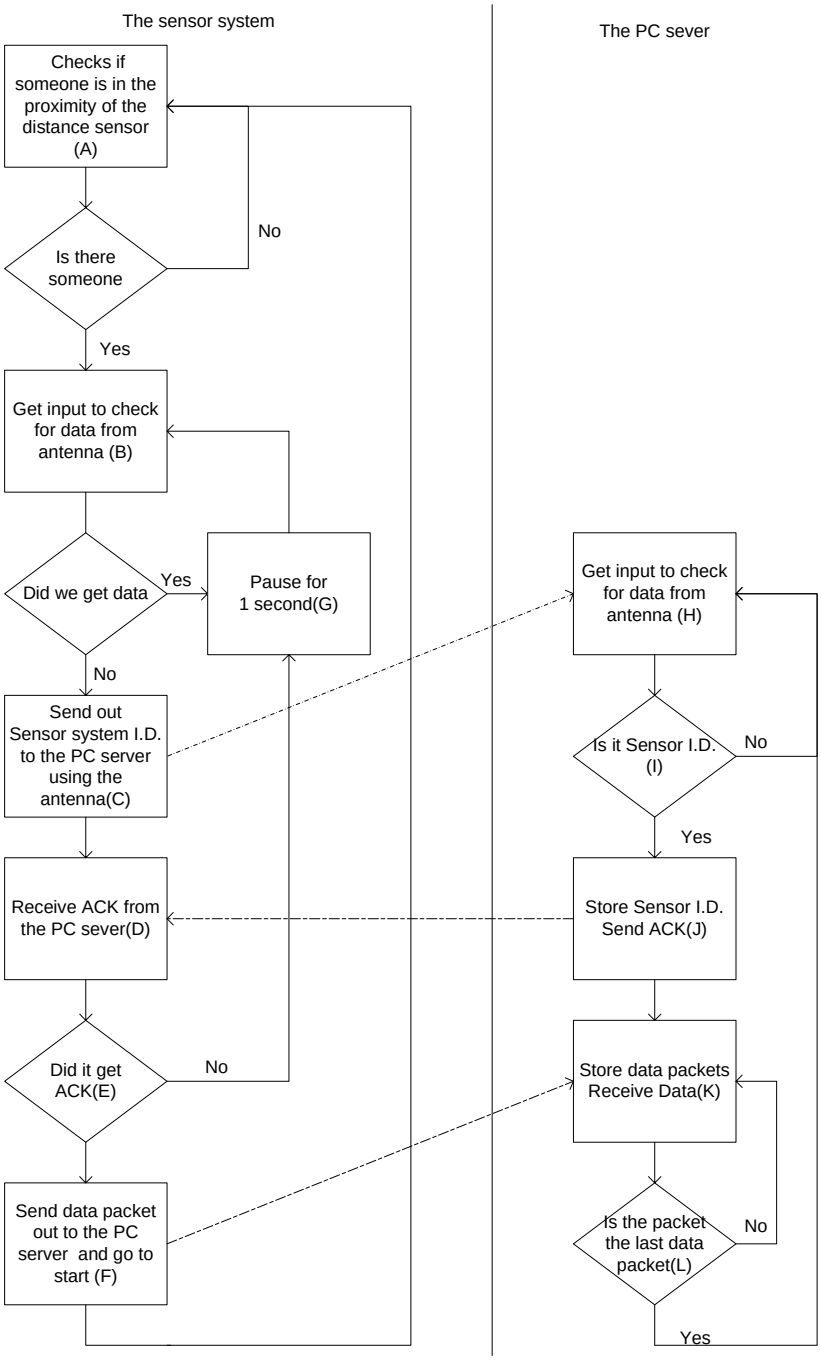


Figure 2.13 Ethernet protocol flow chart

The dotted line in Figure 2.13 shows the case of success in sending data packets. This case shows the case of when there are no collisions. In the case of a collision, both

sensor system and PC server will go back to the starting point.

2.2.3 Round Robin protocol

The Round Robin protocol is similar to the Ethernet protocol. The difference between the two is the handling of the collision event. In the case of the Round Robin protocol, when a collision event occurs, the server PC will send out a Round Robin signal to trigger a Round Robin event on the sensor systems. When the Round Robin event is triggered the sensor will send data one by one in a pre-specified order.

The Round Robin protocol is explained in Table 2.4 for both sensor system and PC server. First, the sensor system checks to see if there is a Round Robin signal (RS 1 on the Sensor system). If this occurs, the program will go to step RS 9, which entails waiting until it receives the sensor system's I.D. from the PC server, and then to step RS 4. However, if this does not occur, it will move to the next step, RS 3. The next steps from RS 2 to RS 8 are the same as steps ES 1 to ES 7 on Table 2.1 of the sensor system.

The PC server first searches for an input to the antenna (RP 1 on the PC server). If it is a sensor I.D. then the PC server uses the same program execution as the Ethernet protocol (EP 1 to EP 5 on Table 2.1). If it is not a sensor I.D., the PC server sends out a Round Robin signal. Then, the PC server executes a Round Robin subroutine which

sends out a sensor I.D. (RP 6 on the PC server) in a pre-set order and receives data in the same execution as the Ethernet protocol (EP 1 to EP 5 on Table 2.1, RP 7 on the PC server). After doing the Round Robin, the program returns to the first step (RP 8 on the PC server).

Sensor System	PC server
(RS 1) The TX/RX checks if we need to do a round robin IF yes them GOTO(RS 9) ELSE GOTO(RS 2) → (A on Figure 2.15) (RS 2) The distance sensor checks if a subject is in the proximity→ (B on Figure 2.15) (RS 3) IF there is a subject then DO(RS 4) ELSE GOTO(RS 2) (RS 4) The TX/RX checks if other systems are sending packets → (C on Figure 2.15) (RS 5) IF the TX/RX does receive input PAUSE and GOTO (RS 4) ELSE DO (RS 6) (RS 6) The TX/RX OUTPUT sensor I.D. to the Sever PC → (D on Figure 2.15) (RS 7) The DB OUTPUT ack to sensor system → (E on Figure 2.15) (RS 8) The sensor system OUTPUT data packets → (F on Figure 2.15)	(RP 1) The TX/RX checks if sensor is sending sensor I.D. → (I on Figure 2.15) (RP 2) IF not do (RP 6) if yes do(RP 3) (RP 3) Send ack to sensor → (J on Figure 2.15) (RP 4) Receive data → (K on Figure 2.15) (RP 5) End of data packet IF yes GOTO (RP 1) ELSE (RP 4)
Subroutine for Round Robin	

(RS 9) The TX/RX INPUT keep polling packets until the Sensor System gets the Sensor System's I.D. then(RS 4)	(RP 6) Send packets request in pre-specified order(3 on fig2.3) → (K on Figure 2.15) (RP 7) DO (PS 2)~(PS 5) (RP 8) Is this the last sensor(in the round robin order) yes GOTO (RP 1) ELSE go to the next sensor(in the round robin order) and GOTO(RP 6)

Table2.4 Pseudo code for Round Robin protocol

Subroutines	
Name	Source
RR	RR: SEROUT 0,25389,10,["e"] ;send outRound Robin signal GOSUB maina GOSUB mainb ; do round robin GOTO main ; go back to start

Table2.5 Subroutines code for Round Robin protocol

2.2.4 Round Robin protocol operation

Figure 2.14 shows the process timing diagram in a collision event. The Round Robin protocol responds to the simultaneous data request in an event (1) on Figure 2.14, which results in a collision event (2). The PC server then sends out a Round Robin signal (3). When the Round Robin event is triggered, the sensor systems send data one after another in a pre-specified order (4 and 5).

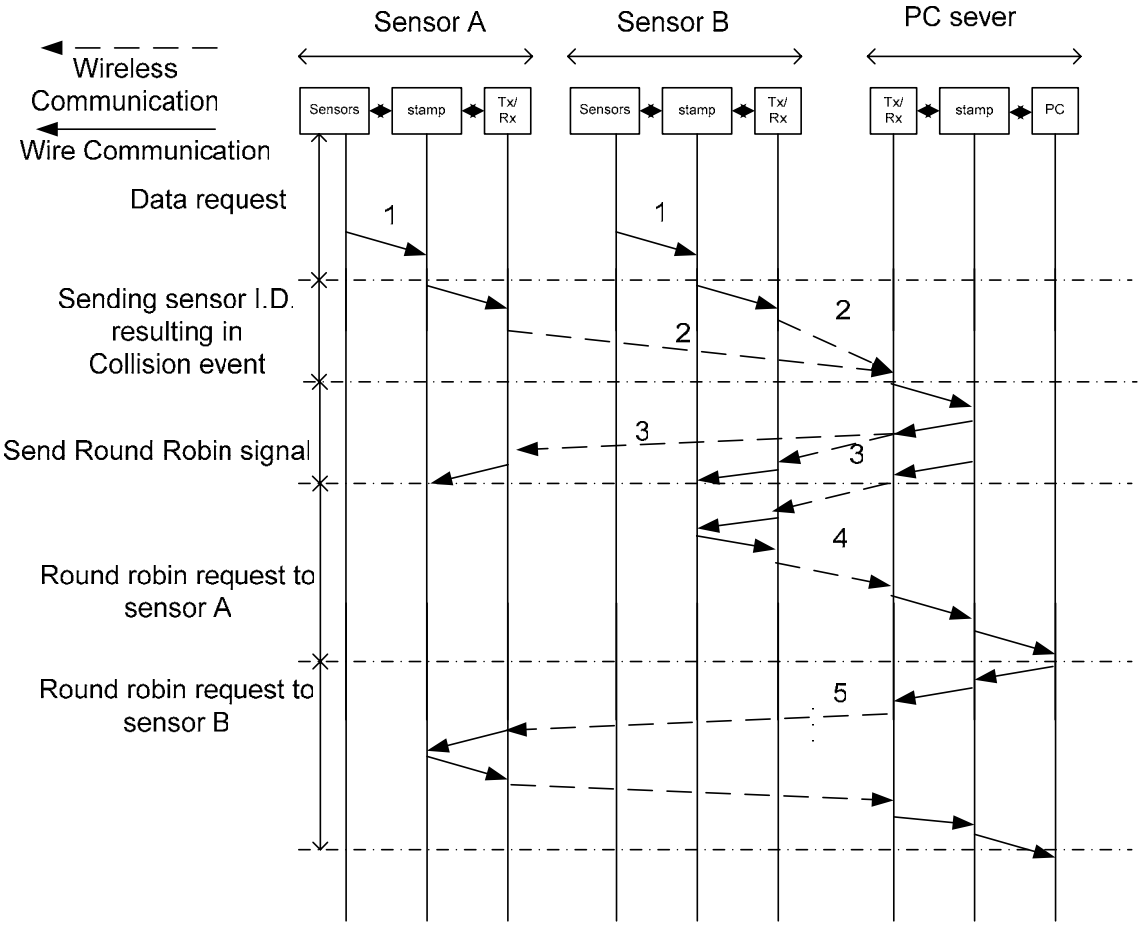


Figure 2.14 Round Robin protocol's timing diagram of collision event subroutine

The flow chart of the Round Robin protocol is on Figure 2.15. The sensor system first checks if there is a Round Robin signal (A) on Figure 2.15. The Round Robin signal is then sent by the PC server when a collision event occurs and the PC server commands the sensor system to send data in a pre-set order. If there is a Round Robin signal, the sensor system waits (G) for its turn to send data until it gets its Sensor I.D. After doing

this step, the rest of the steps are the same as in the Ethernet sensor system flow chart steps (A) to (F) on the Figure 2.13 Ethernet protocol flow chart.

The PC server also has the same steps as the Ethernet PC server, with the exception of when a collision event occurs (J). When there is a collision, the PC server will send a Round Robin signal and do a data request in the designated order. Steps (J), (K), and (L) on the Ethernet protocol (Figure 2.14) correspond to steps (K), (L) and (M) on the Round Robin protocol (Figure 2.15). Appendix A contains the program for this protocol.

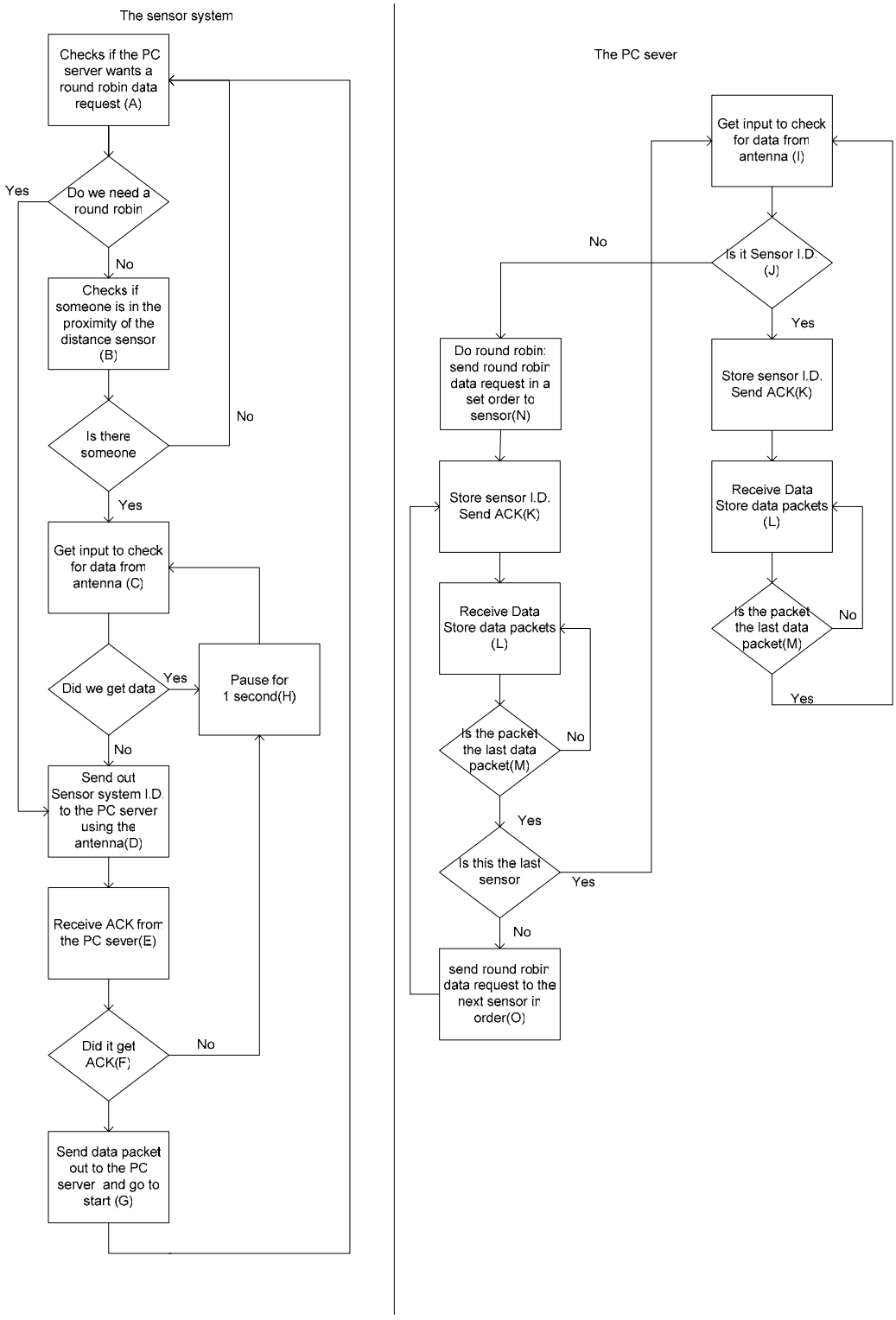


Figure 2.15 Round Robin protocol flow chart

2.3 Experimental Methods

We performed two sets of experiments to determine system performance. The first was to evaluate the performance of two protocols, Ethernet and Round Robin. Here we measured the transmission time to send a set number of packets and the number of collisions (packets lost due to interference between packets). The second is signal loss in the wireless Sensor system (2.3.2). We measure transition through different obstacles and distances. We counted packet loss due to lower signal strength in passing through obstacles such as cement walls, wooden doors, steel doors, etc. We measure the number of packets lost vs. the number of obstacles.

2.3.1 Experiment for different methods of protocol

Here we wanted to measure collisions and transmission time of the two protocols. In summary we performed the following. Before we measured the transmission time of the two protocols, we determined the need to find the program overhead and bad data. Thus, in the first section (2.3.1.1 Method for Best case for using protocols) we performed two tests. Test 1 determined the overhead due to program execution. Test 2 determined the transmission time of the total system. Next we looked at the best case, that is when the system sends packets with the shortest time possible and without

collisions. This will set the lower bound of time to complete sending data. This will help us to get an insight for the transmission time for both protocols.

2.3.1.1 Method for Best case for using protocols

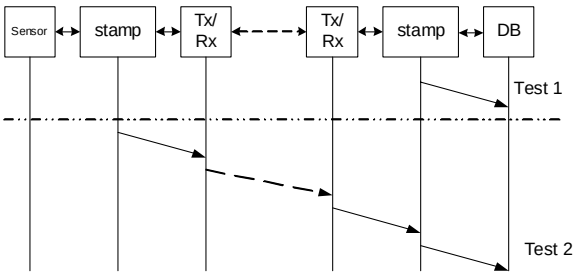


Figure 2.16 Timing diagrams of Test 1 and Test 2

The test 1 (Figure 2.16) experiment was to measure the speed of sending data with the use of a cable. The hardware setup was a stamp board to a RS-232 serial cable and then to the database on the PC. We used the "stampDAQ [3]" software to store the data in the database. We used the SEROUT command with FOR loops (See appendix on programs) and sent 20000 packets of 10 bit data at 9600 baud rate.

The test 2 (Figure 2.16) experiment was to measure the speed of sending data with the use of antennas. The hardware setup was stamp board(Tx) to antenna(Tx) to antenna(Rx) to stamp board(Rx) to serial cable and then to the database. We used the

"stampDAQ" software to store the data in the database. We sent 512 packets of 10 bit data at 1200 baud rate. Then, we compared the measured transmission time to the programmed transmission time.

The best case experiment was to measure the speed of sending data with the use of antennas and the Ethernet protocol. The hardware setup is the same as test 2 sending serial output of 201 packets of 10 bit data at 1200 baud rate. Similarly, we compared the measured transmission time to the programmed transmission time. Appendix B contains the program for this protocol.

2.3.1.2 Method of using protocols experiment

The setting for this simulation is that two sensor systems send 100 data strings to the PC server. The data strings will be sent at baud rates of 300, 600, 1200, 2400, and 4800, with different packets contents of 1header 1character, 1header 2character, 1header 4character, 1header 8character, and 1header 16character. We repeat each experiment ten times to get the mean and variance of packets lost and transmission time. We do this for both Ethernet and Round Robin protocols.

The server system for these experiments sent out programmed data. The sensor systems program was modified so that data request events will occur randomly (See

section 2.2 Communication methods and protocols for both protocols). This is done so that the data request events are not periodic (Figure 2.18). Furthermore, we can change the frequency of data request events to simulate the worst case; that is, both sensor systems will send a burst of packets resulting in a high number of collisions. The PC server that receives the packets from the sensor system will count lost packets and transmission time. The PC server program did not need modifications.

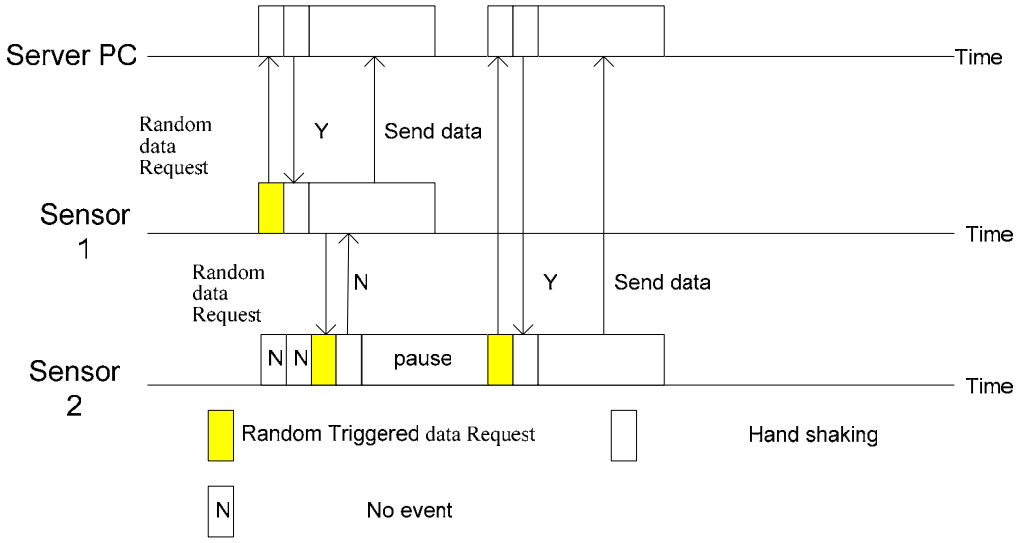


Figure 2.17 Random events in Experiments of protocols

Figure 2.17 shows an example of two sensor systems sending data and the PC sever receiving the data using the Ethernet protocol. Sensor 1 is the first to send data since Sensor 2 has randomly executed No event and Sensor 1 to data Request. We can see that Sensor 2 tries to send data but detects that another sensor is sending. It then pauses then checks again. Since there is no data being sent, Sensor 2 sends data.

Appendix B contains the program for this protocol.

2.3.2 Methods of measurement for performance of the sensor system in different environments

We measured the performance of the system transmission in an environment with obstacles such as walls, corners, doors, floors, and distance. The setup of the system will be the same as 2.3.1.1 Method for Best case with different obstacles (Figure 2.18). The sensor system sends 100 packets using the Ethernet protocol at 1200 baud rate, and the received packets were counted. Each transmission was repeated ten times. Experimental conditions produced by the use of different obstacles between the PC server and the sensor system, caused the signal to deteriorate, which caused the number of packets lost. Measuring distance was done at Stankowski Field at the University of Missouri-Columbia (UMC) and in lab rooms in the Engineering Building West at UMC. This produced a wireless environment for signal reduction. Lost packets were again counted.

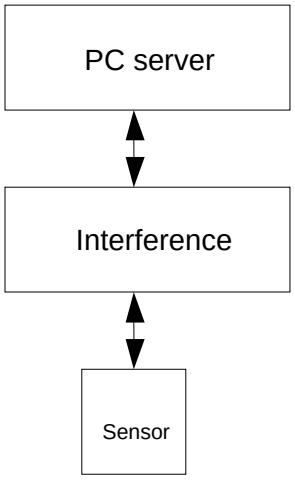


Figure 2.18 Setup of performance of the antenna

CHAPTER 3: RESULTS

In this chapter we will present the results of the experiments on the total system (sensor system and PC). This chapter is divided into two different parts. The first part describes the effectiveness of the protocols. To do this, we measured transmission time, time to complete sending a set amount of data, and collisions, number of data lost due to collision of packets. We describe the acquisition of the data in 3.1 Performance of protocols. The second part measures system errors, which is data loss due to interference of the wireless signal (3.2 Performance of the sensor system).

3.1 Performance of protocols

We first will look at overheads in transmission time. The reason for this is that the speed of the programmed baud rate does not match the measured data rate. Once the measurements and calculations are done we can figure the lowest limit to find the best protocol. The measurements and calculations are presented in 3.1.1 Best Case since it is the best transmission time this protocol can do. Section 3.1.2 Ethernet protocol and 3.1.3 Round Robin protocol will show the measurement of transmission time and collisions vs. baud rate. Section 3.1.4 Evaluation of Performance of protocols will give

us insight into the trade-off between transmission time and collisions with different protocols.

3.1.1 Best case

This section tests the performance of the system of serial and wireless communication, which is the fastest transmission time of data without packet loss. In section 3.1.1.1 Test 1 we will describe the performance of serial communication. In 3.1.1.2 Test 2 we will describe the measurement of the overheads and the transmission time of the total wireless system. In section 3.1.1.3 we simulate the best case and, identify the lower bound of the system (least delay of the system) using the Ethernet protocol.

3.1.1.1 Test 1

Experimental Test 1 was to find out the actual data rate of the stamp board compared to the declared rate on the Basic stamp2. This times the serout function on the stamp, which is used to output data to the PC. The PC receives data on the program excelDAQ. The experimental setup will be the stamp board connected to the PC via a RS-232 cable (see Figure 2.16). We will always use a baud rate of 9600 for line communication. For this experiment we will be sending 20,000 data characters with 10

bits in each character. The result (Table 3.1) shows that the measured delay consists of calculated time to send and to execute, all with in 0. 4% (0.2/56) error.

Measured time to completion(Sec)	Calculated time		Time difference (Measured time - Calculated time)
	Time to send	Execution time	
56	30.8	25	0.2

Table 3.1 Test 1

We calculate the time to send data (column 2) from the stamp board to the PC database by dividing the bits sent with the baud rate. We calculate the execution time (column 3), which is the time for the Stamp Board to process the serial command. This is calculated in appendix C.1.

Time difference (column 4) in Table 3.1 is obtained by measured time to completion (column 1 - Calculated time), gives a measure of error. The execution time in the program in Stamp significantly increases the overall transmission time, and causes the effective throughput rate to decrease.

3.1.1.2 Test 2

Experimental Test 2 determines data rate of the transmit/receive system. We examine the measured data rate vs. declared baud rate. The experimental setup will be stamp board to antenna (Tx) to antenna (Rx) to stamp board and then to the PC (see Figure 2.16).

To do this we will be sending 512 character data with 10 bits in each character. The result is displayed in Table 3.2. The ‘Calculated Transmission` (column 2) is the time for the sensor system to send data to the PC server (see appendix C.2.1). We measured ‘Time to Completion` (column 5) and ‘Bad Data` (column 4). Then we calculated ‘Program Overhead` (column 3) which is instruction execution time (see appendix C.2.2). The overheads consist of program overheads and bad data. Finally, we determine the ‘Time Difference` (column 6) between the measured delay and calculated delay by:

Time Difference =

[Time to Completion – { Calculated Transmission + Program Overhead + Bad Data }].

Baud rate	Calculated Transmission (Sec)(2)	Over Head		Time to Completion (5)	Time Difference (6) [(5) - { (2) + (3) + (4) }]
		Program Overhead(Sec)(3)	Bad Data (Sec)(4)		
300	51.2	16.8	4.6	74	1.4
600	25.6	16.8	2.3	46	1.3
1200	12.8	16.8	1.2	32	1.2
2400	6.4	16.8	0.6	25	1.2
4800	3.2	16.8	0.3	21	0.7

Table3.2 Test 2

In Figure 3.1 we compared the measured data rate vs. programmed baud rate. Where the programmed baud rate is set by the software on the Stamp Board, and the measured data rate is an experimental data rate and is calculated as follows:

measured data rate = total bits sent / (calculated transmission + time difference)

where the total bits sent is 5120 bits (512 characters ×10bits).

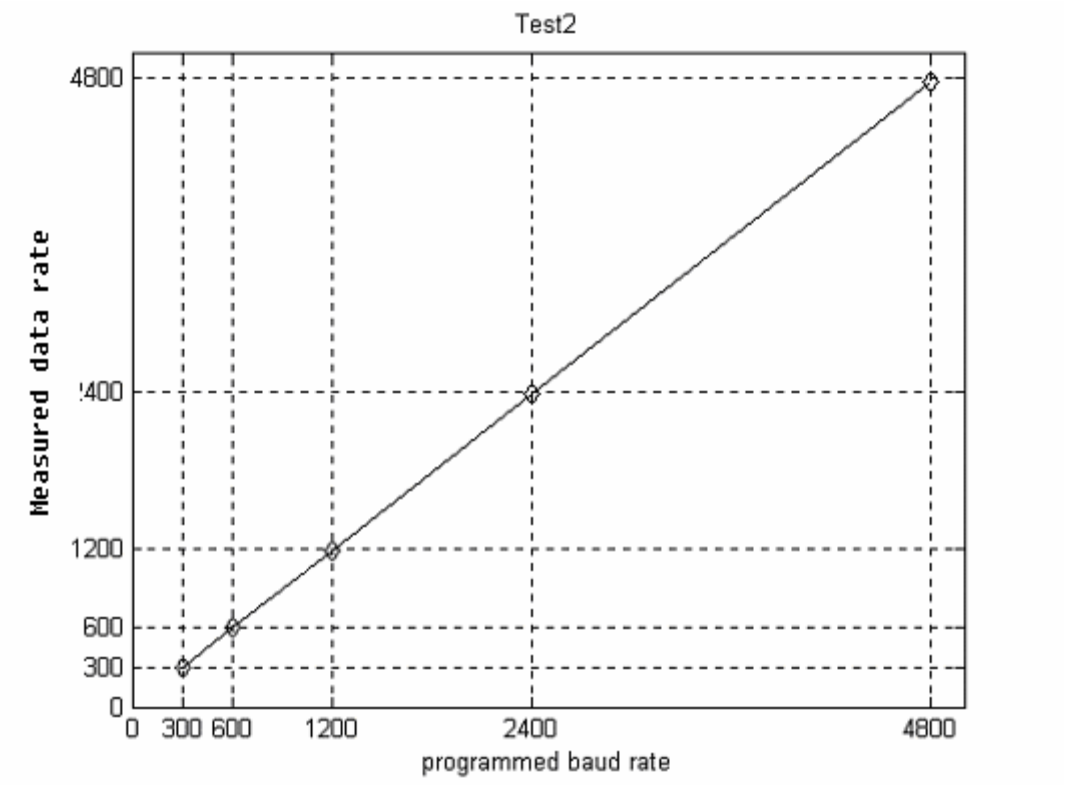


Figure3.1 Slope of Test2

We calculated and displayed the slopes in Table 3.3. We expect the slope to be 1 indication equality between measured data rate and calculated baud rates. We can see that the system shows almost no delay error due to overheads consistent with column 6.

Baud rate	300	600	1200	2400	4800
Slope	0.9967	1.0000	0.9958	0.9979	0.9938

Table 3.3 Slope of test

3.1.1.3 Best case using protocol

In this section we will examine the measured data rate vs. programmed baud rate (i.e. declared baud rate) using protocols. Since the best case has no collision events, both protocols will have the same transmission time.

To do this we will be sending 201 character data with 10 bits in each character with 5 transmissions. The result is shown in Table 3.4; the 'Calculated Transmission` (column 4) is the time for the sensor system to send data to the PC server and is calculated in appendix C.3.1. The 'Program Overhead` is the time for the processor to execute the protocol (column 3), and is calculated in appendix C.3.2. We measured the 'Time to Completion` in (column 2) on the PC server.

Baud rate	Time to Completion (2)	Over Head : Program Overhead(3)	Calculated Transmission (4)	Time Difference (5) [(2) - { (3) + (4) }]
300	49	10.8	33.5	4.7
600	29	10.8	16.8	1.4
1200	20	10.8	8.4	1
2400	16	10.8	4.2	1
4800	13	10.8	2.1	0.1

Table3.4 Best case using protocol

Figure 3.2 is the measured data rate vs. programmed baud rate, where the programmed baud rate is the baud rate declared on the Stamp Board and the measured

data rate is an experimental data rate and is calculated the same as test 2 with Total bits sent being 2010 bits (201 characters × 10bits).

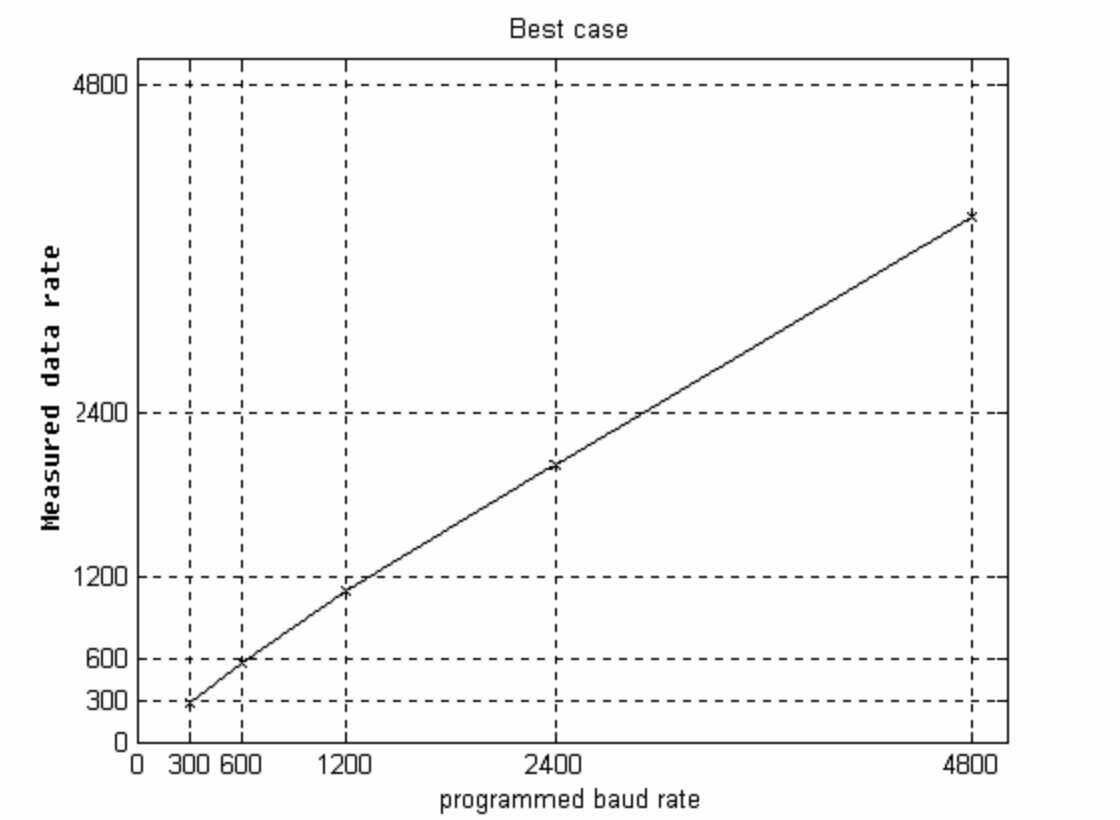


Figure 3.2 Best case using protocol

Baud rate	300	600	1200	2400	4800
Slope	0.9734	0.9517	0.9143	0.8421	0.8000

Table 3.5 Slope of Best case using protocol

The slopes are in Table 3.5. Since the overhead is fixed, the faster transmission time will have greater error (and have lower slope).

3.1.2 Performance of Ethernet protocol

To explain the performance of Ethernet protocol we will look at collisions and transmission time for the system to send a set number of packets. In Figure 3.3 we compare the transmission time of the Ethernet protocol to the best case (See section 3.1.1.3 Best case using protocol) at different baud rates from 300 to 4800. Both protocols sent 200 packets at the shown baud rate at the antenna. Ethernet protocol was used to send 100 packets from 2 sensor systems (the total of 200 packets to the PC) in contrast to the best case which sent 200 packets from one sensor system to the server PC.

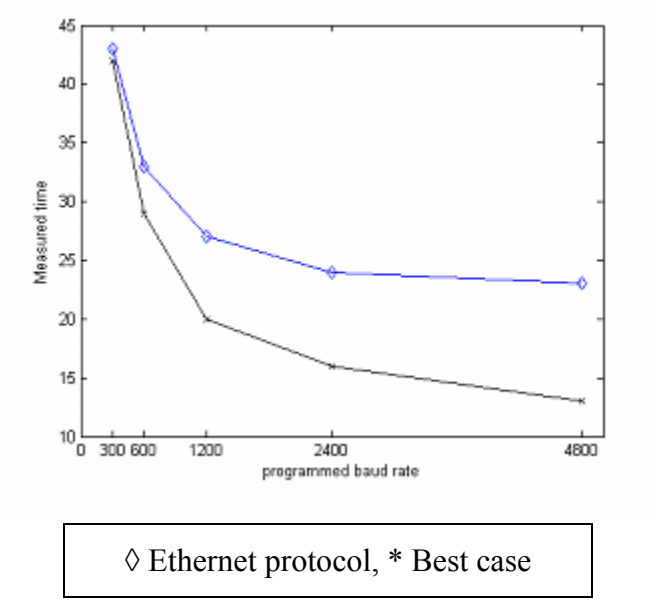


Figure 3.3 Ethernet protocol vs. Best case

We can see that the Ethernet protocol has a larger transmission time compared to the best case as the baud rate gets higher. This is due to the fact that the Ethernet waits for the sensor system or server PC to send data. This is also due to collisions.

We show in Figure 3.4 and Figure 3.5 the number of collisions and transmission of the system under normal conditions (that is without use of Round Robin protocol). For this experiment we sent 100 packets from 2 sensor systems at baud rates of 300, 600, 1200, 2400, 4800. Each curve is caused by the different packet contents of 1header 1character, 1header 2character, 1header 4character, 1header 8character, 1header 16character. We repeat this ten times. Figure 3.4 shows number of collisions and Figure 3.5 is the transmission time, the error bars being the upper and lower standard deviation.

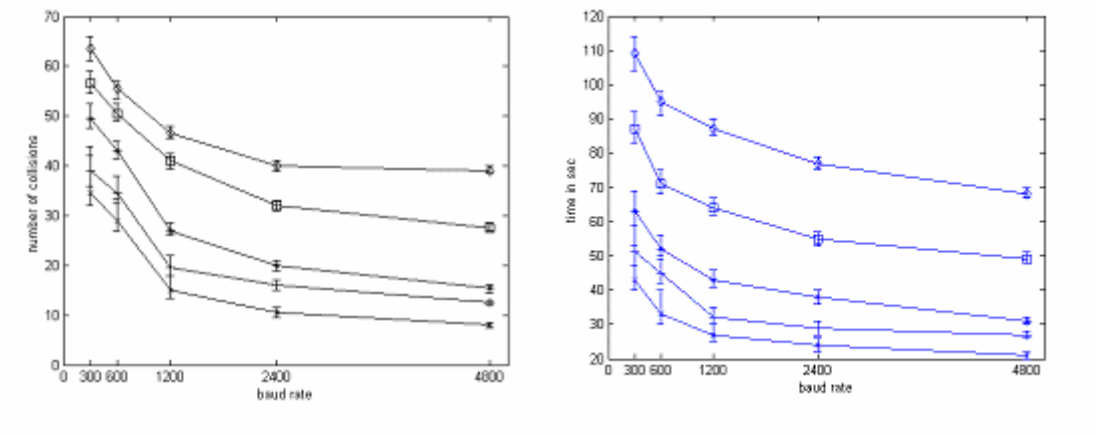


Figure 3.4 Collisions(Ethernet)

Figure 3.5 Transmission Time(Ethernet)

◊ 1 header 16 character
□1 header 8 character
* 1 header 4 character
- 1 header 2 character
×1 header 1 character

As the baud rate goes up the transmission time and collisions go down. This is due to the fact that the Ethernet protocol pauses fewer times and the length of time to send packets is shortened. The standard deviation decreases for the same reason.

3.1.3 Performance of Round Robin protocol

Similarly the performance of the Round Robin protocol will look at collisions and transmission time when the system sends a set number of packets. In Figure 3.6 we compare the transmission time of the Round Robin protocol with the best case. For Round Robin and Ethernet protocol we sent 100 packets from 2 sensor systems (the total of 200 packets to the PC), the best case sent 200 packets from one sensor system, to the server PC.

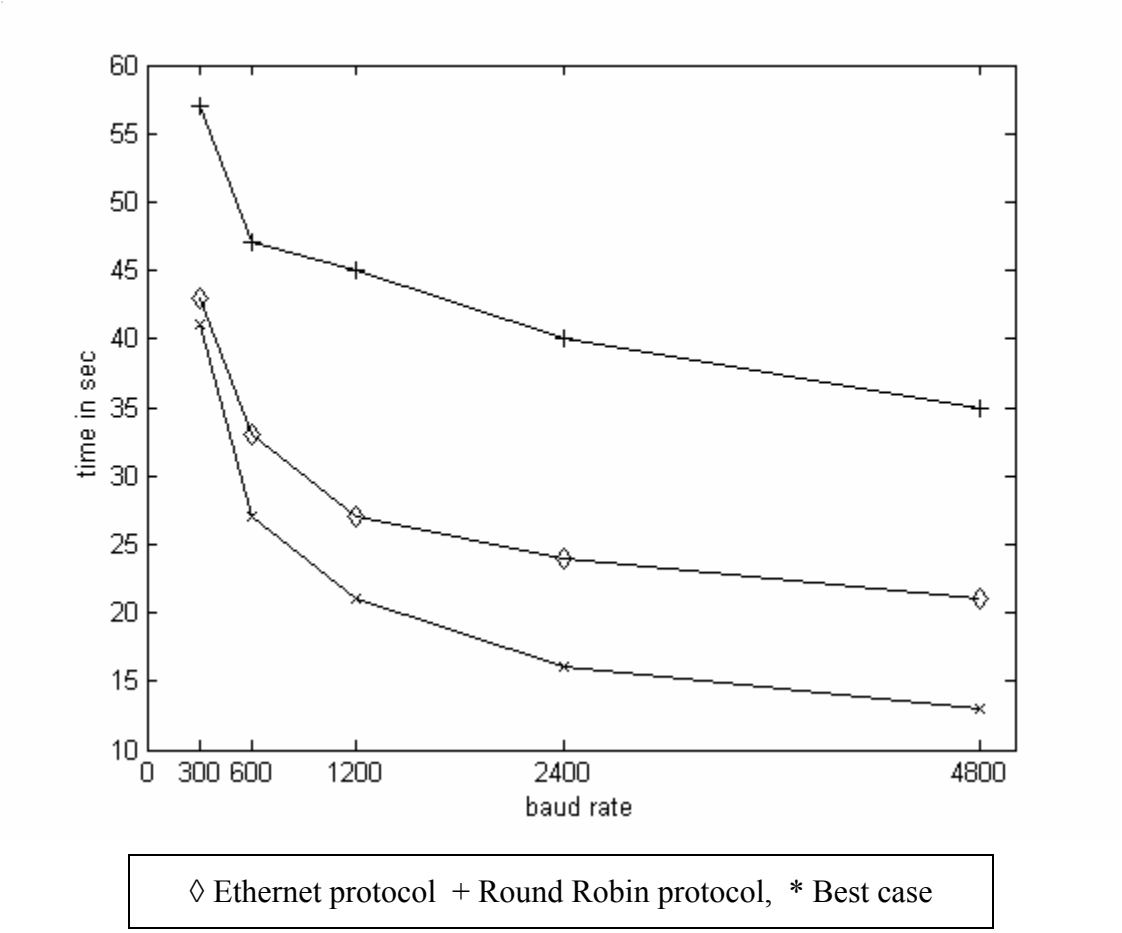
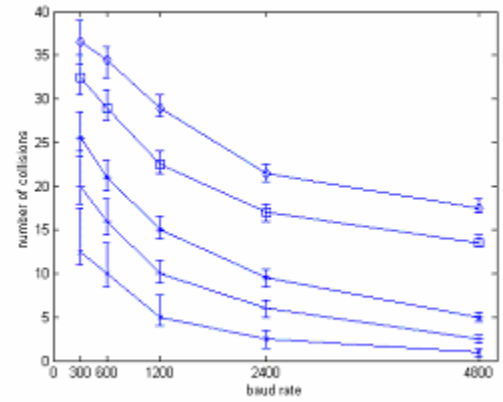


Figure 3.6 Best case vs. Round Robin protocol

We can see that the Round Robin protocol has a larger transmission time compared to the best case and the Ethernet protocol as the baud rate gets higher. As before both the Ethernet protocol, and Round Robin protocol pause when another sensor system or server PC sends data. The Round Robin protocol's handing of collision results in greater transmission time compared to Ethernet protocol.

Figure 3.7 and Figure 3.8 show the result of round robin operation. Similar to before we send 100 packets from 2 sensor systems at baud rates of 300, 600, 1200, 2400, and 4800. Each curve of transmission time and collision come from the packets of 1header 1character, 1header 2character, 1header 4character, 1header 8character, and 1header 16character and repeated ten times. Figure 2.7 shows the number of collisions and Figure 3.8 shows the transmission time with error bars being the upper and lower standard deviation.



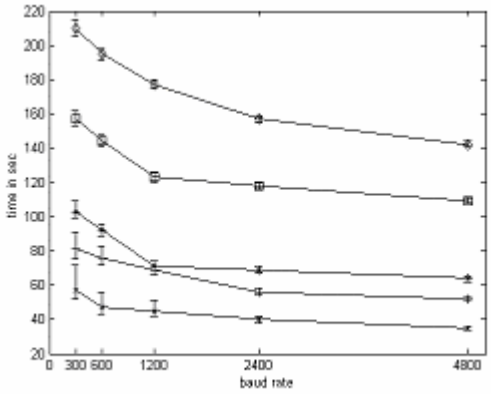


Figure 3.7 Collisions(Round Robin) **Figure 3.8** Transmission Time(Round Robin)

◊ 1 header 16 character
□1 header 8 character
* 1 header 4 character
- 1 header 2 character
×1 header 1 character

Similarly, the transmission time and collisions curves have a hierocracy due to the packet size. Furthermore, compared to Ethernet, the decrease in collisions and increase in transmission time with increase in baud rate due to pause time. Note the decrease in standard deviation. Furthermore the system acts as an Ethernet except when a collision occurs, in which case the system evokes the round robin protocol.

3.1.4 Evaluation of protocols Performance

The Performance Rate of the protocols for the system calculates the total number of characters received over transmission time. We calculate it as follows

Performance_rate = data characters received / transmission time

We want the Performance_rate as high as possible, that is we want to receive the highest number in the least possible time. Figure 3.9 and Figure 3.10 gives a new perspective of the performance of both protocols; we plot using performance rate and the corresponding baud rate.

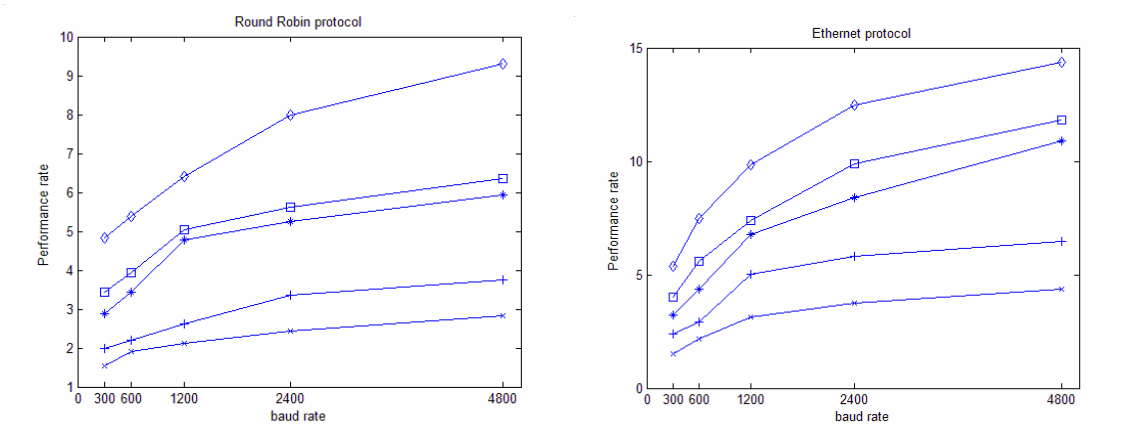


Figure 3.9 performance of Round Robin **Figure 3.10** performance of Ethernet

◊ 1 header 16 character
□1 header 8 character
* 1 header 4 character
- 1 header 2 character
×1 header 1 character

Note that the Round Robin protocol out performs the Ethernet protocol as the baud rate increases up to 50%.

3.2 Performance of the sensor system in different environments

We measured the performance of the system in the environment in which the antenna will be used. Walls, corners, doors and floors all interfere with transmission, and of course the signal fades with distance. These factors change the outcome of the performance of the antenna by losing signal strength. This manifests greater sensitivity to noise and interference.

As mentioned in section 2.3.2, with the method for this measurement, the sensor sent 100 packets using Ethernet protocol at 1200 baud rate and the received packets were counted. Each transmission was repeated ten times. A new transmission condition occurred due to change in experimental conditions, such as the distance and the number of walls, which obstructed the Sensor Systems sending data to the server PC.

Experiment 3.2.1 was done at Stankowski Field at UMC (distance experiment). Experiment 3.2.2 to 3.2.6 was done in the lab rooms at Engineering Building West at UMC (Transmission experiment).

3.2.1 Performance of the system to transmit according to distance in free space

We measured the variation in performance of the system due to distance. We sent 100 packets from the sensor to the PC, increasing the distance by 50 feet from 50 to 200 feet and counted received packets. Figure 3.11 is the count of errors and standard deviation with increasing distance of 50, 100, 150, and 200 feet. Note the manufactures recommended distance for this system is 150 feet.

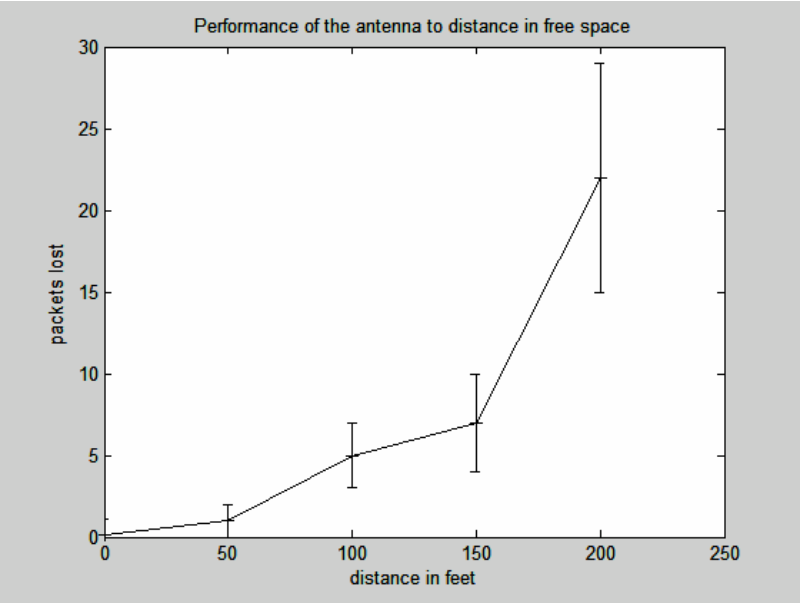


Figure 3.11 Error count of free space

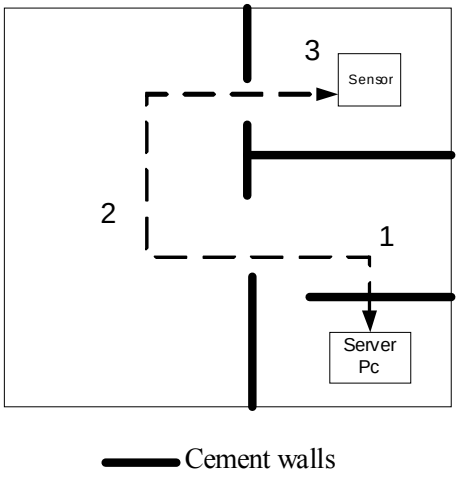
At 50 feet we lose 0.1% of the packets. Measurement for section 3.2.2 - 3.2.6 are done within 50 feet, so we will assume that there will be no error due to distance.

3.2.2 Performance of the antenna to transmit with interference due to walls with obstructed openings

We measured the performance of the system in communicating though an obstruction of cement walls with openings. Figure 3.12 shows the experimental setup for the sensor communicating with the PC though wall obstructions. We sent 100 packets from the sensor to the PC and counted the received packets from location 1, 2,

and 3 on Figure 3.12. Figure 3.13 shows the mean number of packets lost with the count of errors and standard deviation verses increasing numbers of obstruction items.

For each obstruction there is a sharp increase in the mean and variance. This means that as the obstruction increases the lost packet count increases and the transmission variability also increases.



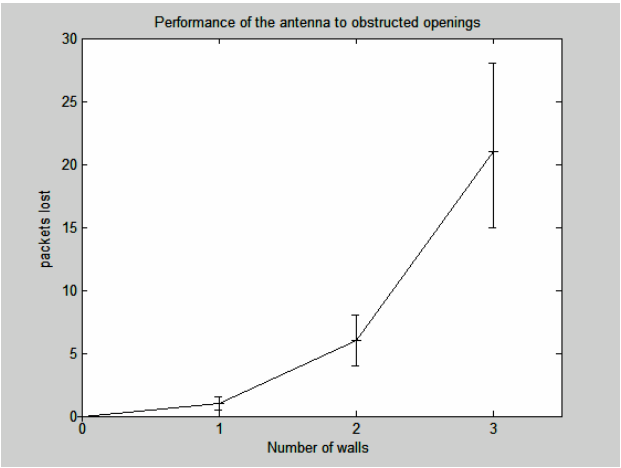
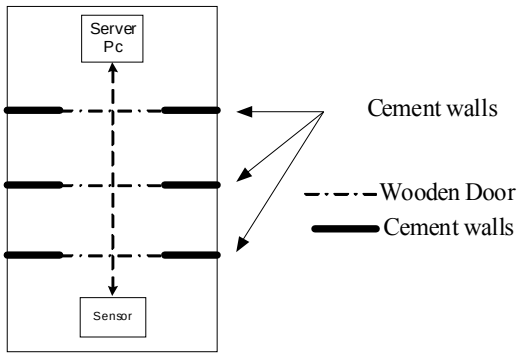


Figure 3.12 Diagram of obstructed opening **Figure 3.13** Error count of obstructed opening

3.2.3. Performance of the antenna to transmit with interference by wooden doors obstruction

Similarly, we measured system performance this time with obstruction by cement walls with wooden doors. Figure 3.14 shows the experimental setup for the sensor

communicating with the PC though wooden door obstructions. We sent 100 packets from the sensor to the PC and counted the received packets. We repeated the measurement with increasing number of doors, from 1 to 3, as shown in Figure 3.14. Figure 3.15 is the mean count of lost packets and variance with increasing numbers of obstruction item. Note that compared to the unobstructed walls of the previous section there is increase in packets lost.



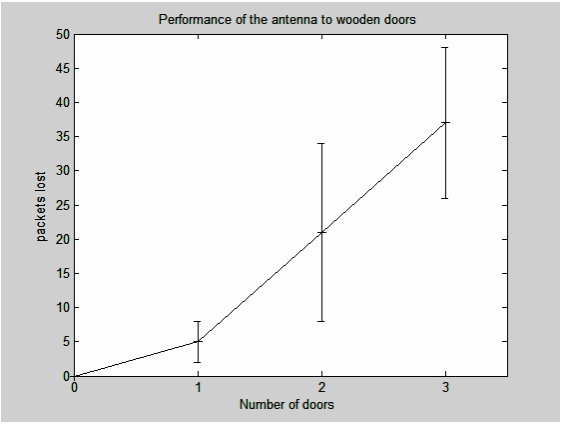
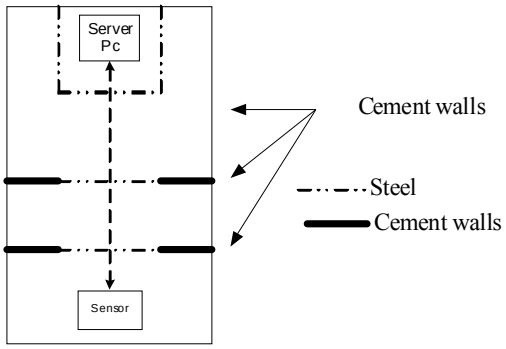


Figure 3.14 Diagram of wooden doors **Figure 3.15** Error count of wooden doors

3.2.4 Performance of the antenna to transmit with interference by steel door obstruction

Similarly, we measured system performance this time with obstruction by cement walls with steel doors. Figure 3.16 shows the experimental setup for the sensor

communicating with the PC though obstruction by cement walls. We sent 100 packets from the sensor to the PC and counted the received packets. We repeated the measurement with increasing number of doors, from 1 to 3, as shown in Figure 3.16. Figure 3.17 is the mean count of lost packets and standard deviation with increasing numbers of obstruction items. Note that compared to the wooden doors of the previous section there is a decrease in packets lost.



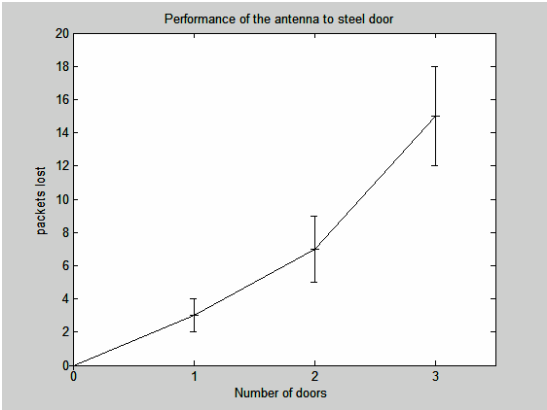


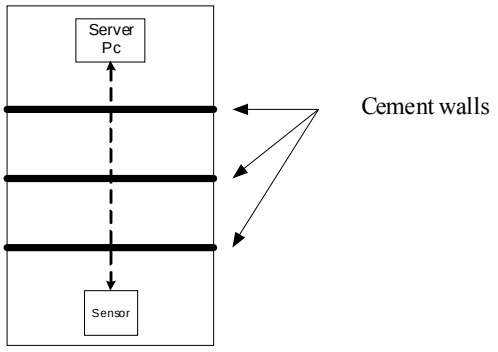
Figure 3.16 Diagram of steel door **Figure 3.17** Error count of steel door

3.2.5 Performance of the antenna to transmit with interference by cement walls obstruction

Similarly, we measured system performance this time with obstruction by cement walls. Figure 3.18 shows the experimental setup. The sensor communicates with the PC

through the obstructions shown. We sent 100 packets from the sensor to the PC and counted the received packets. We repeated the measurements with 1 to 3 walls, as shown in Figure 3.18. Figure 3.19 is the mean count of lost packets and the standard deviation of the count with increasing numbers of obstruction items.

In surveying the experimental outcome at the 3rd wall, on average we received only 13 packets, occasionally we received no packets (see large standard deviation). In wall 2 we lost half the packets, and in wall we received 90% of the packets.



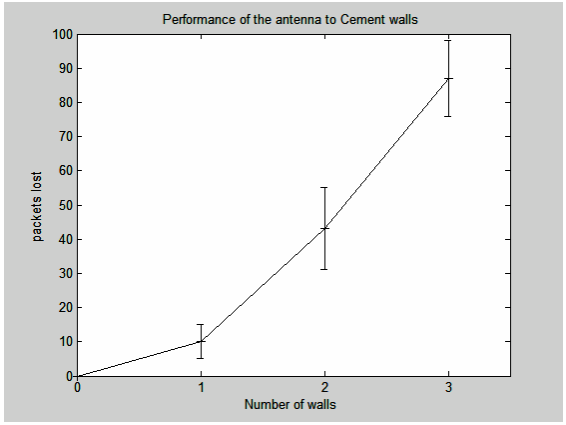


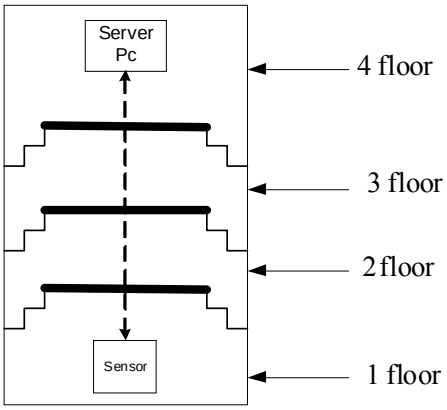
Figure 3.18 Diagram of cement walls **Figure 3.19** Error count of cement walls

3.2.6 Performance of the antenna to transmit with interference from different floor obstruction

Similarly, we measured system performance, this time with interference by different floors. Figure 3.20 shows the experimental setup for the sensor communicating

with the PC on different floors. We sent100 packets from the sensor to the PC and counted the received packets. We repeated the measurement with increasing number of floors from 1 to 3 as shown in Figure 3.20. Figure 3.21 is the mean count of lost packets and standard deviation of packets lost with increasing number of floors.

Note that similar to the cement walls of the previous section, there is high loss of packets. Again, in the 3rd floor, we lost all packets. This data represents the lowest performance of the system on all experiments we ran. The floors are cement and it appears that this material seriously compromised our signal transmission signal.



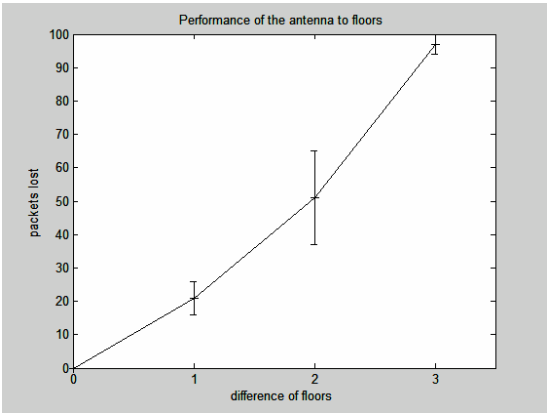


Figure 3.20 Diagram of floors **Figure 3.21** Error count of floors

CHAPTER 4: DISCUSSIONS

4.1 Hardware

In this paper we have constructed and tested a wireless sensor system with two sensors and a PC server. The sensor system gathers data and sends the data upon proximity interrupt event (an event when the system detects that a subject is near the sensor). The sensor system data passes though its parts: sensors (for gathering data), to processor (for controlling and storing data), and to antenna (for sending the data to the PC server).

The PC server receives the data sent by the sensor and stores the data. The PC server receives data though 2 parts: the antenna (for receiving data) to the processor (for controlling and storing data) and finally, to the database via serial communication (for sending the data to the and saving it on the PC). The data contains packets of 2 to 17 bytes. The first byte is a header, the remaining bytes are data.

In integrating the system, we first selected the processor for the sensor system and the PC server. The processor is a key factor in the outcome of the performance of the systems. The selection of the processor was done by comparing the factors, cost, and availability. From these factors, we decided on the Basic stamp2 processor. Then we selected the 433.92MHz transceiver antenna by its ability to send and received wireless

data from the BASIC stamp 2 processor. Other components such as ADC and proximity sensor were selected with cost and availability in mind.

4.2 Protocols

After we decided on hardware we implemented two protocols on the system. Since we have two sensors and a PC server we need a protocol to make the system work as one system. For this we designed two protocols: the Ethernet and the Round Robin protocols. Ethernet is the routine protocol of the wireless communication system. Round Robin protocol is a modification of the Ethernet protocol. The Round Robin protocol reduces packet loss caused by collision.

To program the protocols we had to resolve the fate of collisions during a wireless transmission. For the Ethernet protocol, we discarded the data upon collisions; while the Round Robin protocol was designed to get data from every sensor in a set order. We also decided on packet formats and baud rates to make the system coherent with each other. After we implemented the protocols, we considered two factors for deciding the performance of the protocols: transmission time and collisions. The results in 3.1.2 Ethernet protocol and 3.1.3 Round Robin protocol show that for transmission time, the Round Robin protocol is slower then the Ethernet protocol transmission time (50 - 75%),

but the Round Robin protocol has less collisions (0~95%).

4.3 Results

Before assessing the protocols, we determined the overheads in transmission time. This is shown in 3.1.1 Best case in Test 1(Table 3.1), which tests the transmission time between the BASIC Stamp2 and database. We found that sending data to the database required 25 seconds, execution time for BASIC Stamp2 was 56 seconds for completion time, which led to a 55% overhead. In test 2 we measured the transmission time between the sensor and the PC server at different baud rates and found that as the baud rate goes up the transmission time goes down (Table 3.2). Also, there is a fixed program execution overhead of 16.8 seconds. In Table 3.3 and Table 3.5 we showed the measured baud rate to the calculated baud rate. This was done for a single signal sensor and for multiple sensors, which shows compression of the slopes, that is, multiple sensors slow the system down.

The 3.1.2 Performance of Ethernet protocol showed the collision and transmission time in Figure 3.5 and Figure 3.6. In Section 3.1.2, as the baud rate goes up, the transmission time and collisions go down, because the Ethernet protocol pauses fewer times and the length of time to send packets is shortened. The variance decreases for the

same reason. Collision is important because of data survival, and we want to minimize transmission time, since a higher baud rate is needed to send large amounts of data in real time.

Similarly, in 3.1.3 Performance of Round Robin protocol, we measured collisions and transmission time for the system to send a set number of packets. The result also had similar characteristics. When the baud rate went up, the transmission time and collisions went down. The Round Robin protocol is slower than the Ethernet protocol since a collision evokes the round robin protocol. In Figure 3.7, we compared the transmission time of each protocol and found that the best case had the fastest time, followed by the Ethernet protocol and then the Round Robin protocol. This order is due to the program overhead and collisions.

To get a different prospective in section 3.1.4 Evaluation of protocols Performance, we showed that there is a trade-off between data survival rate and time of completion rate, with the Ethernet being faster and the Round Robin having a better survival rate. We calculated the performance rate to see which protocol receives as many characters as possible in the shortest time; in this case Round Robin protocol again out-performed the Ethernet protocol. (Figure 3.10 and Figure 3.11).

Finally in Section 3.2 we measured the performance of the wireless sensor in

environments in which the antenna will be used, walls, corners, doors and floors, which interfere with transmission. The number of packets lost increases in the following order: free space (3.2.1), steel door obstruction (3.2.4), unobstructed openings (3.2.2), wooden doors obstruction (3.2.3), walls obstruction (3.2.5), and floor obstruction (3.2.6).

In the free space experiment, we determined the distance that the signal traverses. The systems specifications for the antenna have a range of 150 feet our data show that we lose about 5% of the packets. In some applications this may be unacceptable. Also door obstructions, unobstructed openings, and wooden doors can be ignored, compared to walls, and floor obstructions which cause significant problems.

4.4 Summary

We evaluated the preference of two wireless sensors with the Basic stamp2 processor. The computation of over head was a significant part in transmission time. By using the Round Robin protocol instead of Ethernet protocol we decreased collision counts. Finally, the system operates relatively well in small distances and obstructions reduce the signal.

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

In building the system the focus was on the cost and availability of hardware and the performance of the system. We succeeded in connecting the system to work in a wireless environment using protocols of Ethernet and Round Robin. We identified three problems.

The first problem is overheads that slow down the system. The overheads occur because data transmissions require processor power to transmit from a single node to the PC server. The overheads increase the transmissions data rate and are due to the processor's ability to process data.

The second problem is losing data to collisions. This is due to a burst of data transmissions from two nodes which results in collisions. Also the protocol makes the node pause upon collisions which increases the transmission time. Furthermore, both protocols are designed to pause for one second if other nodes are sending data; this adds additional time to processing data. However, by pausing we avoid losing packets (that is avoid collisions).

Overheads, and collisions, need to be considered when there are two or more

nodes. We can see a trade-off between packet survival and transmission time using different protocols, there is a limited data rate that this system can effectively process. The Round Robin's performance rate is the best.

The third problem is the performance when transmitting through different obstacles; the nodes' ability to send data through different types of obstacles. Although the fewer packets are lost due to obstacles of this system, there can be a problem receiving data from unwanted nodes. In this case, obstacles that block connectivity can be efficient.

5.2 Future work

The overhead causes significant problems for the data rate of the system. To reduce the effect of the overhead, it would be interesting to see the effectiveness when using higher power processors. For the reduction of collisions and transmission time we should consider using a higher baud rate antenna which is an antenna with a higher bandwidth. Note that these changes to the system will lead to a higher cost in hardware and software development. Also, a study could be conducted on sending larger amounts of data such as images or speech data.

REFERENCES

[1] I.F. Akyildiz, W. Su*, Y. Sankarasubramaniam, E. Cayirci, "Wireless sensor networks: a survey, Computer Networks", p393-422, 2002

[2] C. Intanagonwiwat, R. Govindan, D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks", Proceedings of the ACM Mobi- Com'00, Boston, MA, p56–67,2000

[3] Parallax Inc. Found at: http://www.parallax.com

[4] Berkley Wireless Embedded Sestems Project Found at: http://webs.cs.berkeley.edu

[5] Crossbow Technology Inc. Found at: http://www.xbow.com

[6] Boo-Ho Yang, Sokwoo Rhee, and Haruhiko H. Asada, "A Twenty-Four Hour Tele-Nursing System Using a Ring Sensor", Proceedings of IEEE International Conference on Robotics & Automation, Leuven, Belgium, p387-392,1998

[7] Ari Y. Benbasat, Stacy J. Morris, and Joseph A. Paradiso, "A Wireless Modular Sensor Architecture and its Application in On-shoe Gait Analysis", Responsive Enviroments Group, p1088-1091

[8] Anu Bhargava and Mike Zoltowski, "Sensors and Wireless Communication for Medical Care", Proceedings of the 14th International Workshop on Database and Expert Systems Applications, 2003

[9] National Inc. Found at: http://cache.national.com/pf/AD/ADC0834.html

[10] Acroname Inc. Found at: http://www.acroname.com/robotics/parts/gp2d02.pdf

[11] Rabbit semiconductor Inc. Found at: http://www.rabbitsemiconductor.com/

[12] Intel Inc. Found at: http://www.intel.com

[13] Arm Inc. Found at: http://www.arm.com/

[14] NetMedia Inc. Found at: http://www.basicx.com/

[15] RF Digital Corporation Found at: http://www.rfdigital.com/item.htm?item=RFD27997

[16] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, " Energy efficient communication protocols for wireless microsensor networks", Proceedings of the 33rd International Conference on System Sciences(HICSS '00), January 2000

[17] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a saclable and robust communication paradigm for sensor networks", In proceeding of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '00), August 2000

[18] S. Lindset and C. S. Raghavendra, " PEGASIS: Power-efficient gathering in sensor information systems", In proceeding of the IEEE Aerospace Conference, April 2001

[19] W. Heinzelman, J. Kulik and H. Balakrishnan, "Adaptive protocols for information dissemination in wireless sensor networks", ", In proceeding of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '99), August 1999

[20] Y. Yu, R. Govindan, and D. Estrin, "Geographical and energy-aware routing: a recursive data dissemination protocol for wireless sensor networks", In proceeding of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '01), August 2001

[21] Ahmed A. Ahmed, Hongchi Shi, and Yi Shang, " A survey on network protocols for wireless sensor networks", IEEE, 20003

[22] MEMS and Nano technology exchange. Found at: http://www.memsnet.org/mems/what-is.html

[23] NASA Jet Propulsion Laboratory Communication Systems and Research Section. Found at : http://dsp.jpl.nasa.gov/members/payman/swarm/

APPENDIX

APPENDIX A

SOURCE CODE FOR ETHERNET PROTOCOL AT PC SERVER

```
'{$STAMP BS2}
'{$PBASIC 2.5}

sensor VAR Byte(6)
y VAR Byte
v VAR Byte(3)
counter VAR Byte(2)

main:
counter=0
SERIN 1,25389,[STR sensor\1]
IF sensor="a" THEN maina
IF sensor="b" THEN mainb
 SEROUT 0,25389,10,["e"] 'for error

GOTO main

maina:

 SEROUT 0,25389,10,["a"]
SERIN 1,25389,100,main,[STR y\1]
DEBUG "main_a=",STR sensor   ,CR

GOSUB get_data
GOTO main

mainb:
 SEROUT 0,25389,10,["b"]
SERIN 1,25389,100,main,[STR y\1]
DEBUG "main_b=",STR sensor   ,CR
get_data :
'SEROUT 0,25389,10,["r"]
```

```
SERIN 1,25389,[DEC1 v]
DEBUG "ans=",DEC1 v   ,CR
GOTO main


GOSUB get_data
GOTO mainb
```

SOURCE CODE FOR ETHERNET PROTOCOL AT SENSOR

```
'{$STAMP BS2}
'{$PBASIC 2.5}

'-----------------------------------------------------
' variable declarations
adcBits VAR Byte
v VAR Byte
counter VAR Byte
v1 VAR Byte
v2 VAR Byte
val VAR Byte
C VAR Byte
sensor VAR Byte
indata VAR Byte(6)
'-----------------------------------------------------
' constant declarations
cl CON 14 ' pin 14 is output (clock)
dt CON 15 ' pin 15 is the input (data)
CS PIN 9
CLK PIN 10
DataOutput PIN 11

INPUT dt ' make pin 15 the input
HIGH cl ' make pin 14 output and high
```

```
main:
  sensor="a"
  counter=0
  GOSUB chk_interrupt



'  PAUSE 100 ' stall so display readable
GOTO main

chk_interrupt:    'chk for movement
  GOSUB read1 ' call measurement routine
  v1 = 5 * val / 255
  GOSUB read1 ' call measurement routine
  v2 = 5 * val / 255
  C= V1-V2
  IF C=0 THEN main
   DEBUG CR, "int" , CR ' display the value
  SERIN 2,25389,10,send_data_int,[STR indata\1] 'chk to see if other is
sending data if so wait
GOTO wait_for_Tx

send_data_int:

  SEROUT 1,25389,10,["a"] 'first contact
  SERIN 2,25389,100,main,[STR indata\1]
   ' DEBUG "send_data_int",STR indata   ,CR
  SEROUT 1,25389,10,["y"]
GOTO send_data

wait_for_Tx :

    PAUSE 1000'wait to send
GOTO chk_interrupt
```

```
send_data:

HIGH CS
LOW CS
LOW CLK
PULSOUT CLK, 210
SHIFTIN DataOutput,CLK,MSBPOST,[adcBits\8]
v = 5 * adcBits / 255

  SEROUT 1,25389,10,[DEC1 v]
'DEBUG "8-bit binary value: "    , BIN8 adcBits
'DEBUG CR,CR,  "Decimal value: "  , DEC3 adcBits
DEBUG CR, "DVM Reading:     ",     DEC3 v,  "    Volts"    '  ¡à new
line
'PAUSE 1000
'DEBUG CR
GOTO main

read1:
  LOW cl ' turn on detector for reading
  rl:
  IF IN15 = 0 THEN rl ' wait for input high
  SHIFTIN dt, cl,MSBPOST,[val\8]
  HIGH cl ' turn detector off
  PAUSE 1 ' let detector reset
RETURN
```

SOURCE CODE FOR ROUND ROBIN PROTOCOL AT PC SERVER

```
'{$STAMP BS2}
'{$PBASIC 2.5}

sensor VAR Byte(6)
y VAR Byte
v VAR Byte(3)
```

```
counter VAR Byte(2)


main:
counter=0
SERIN 1,25389,[STR sensor\1]

IF sensor=127 THEN main
IF sensor="a" THEN maina
IF sensor="b" THEN mainb
DEBUG "start=",DEC sensor   ,CR

'PAUSE 10
GOTO rr


maina:
SEROUT 0,25389,10,["a"]
SERIN 1,25389,100,main,[STR y\1]
DEBUG "main_a=",STR sensor   ,CR
GOSUB get_data
GOTO main

mainb:
SEROUT 0,25389,10,["b"]
SERIN 1,25389,100,main,[STR y\1]
DEBUG "main_b=",STR sensor   ,CR
GOSUB get_data
GOTO mainb


get_data :
'SEROUT 0,25389,10,["r"]
SERIN 1,25389,[DEC1 v]
DEBUG "ans=",DEC1 v   ,CR
GOTO main
```

```
rr:
SEROUT 0,25389,10,["e"] 'for error
GOSUB maina
GOSUB mainb
GOTO main
```

SOURCE CODE FOR ROUN DROBIN PROTOCOL AT PC SENSOR

```
'{$STAMP BS2}
'{$PBASIC 2.5}

'-----------------------------------------------------
' variable declarations
adcBits VAR Byte
v VAR Byte
counter VAR Byte
v1 VAR Byte
v2 VAR Byte
val VAR Byte
C VAR Byte
sensor VAR Byte
indata VAR Byte(6)
'-----------------------------------------------------
' constant declarations
cl CON 14 ' pin 14 is output (clock)
dt CON 15 ' pin 15 is the input (data)
CS PIN 9
CLK PIN 10
DataOutput PIN 11

INPUT dt ' make pin 15 the input
HIGH cl ' make pin 14 output and high
```

```
main:
  GOSUB chk_collision
  sensor="a"
  counter=0
  'GOSUB chk_interrupt



'  PAUSE 100 ' stall so display readable
GOTO main

chk_collision:

  SERIN 2,25389,10,chk_interrupt,[STR indata\1]
  DEBUG CR,"collision_int",STR indata   ,CR
  IF indata="e" THEN send_data_int
  SEROUT 1,25389,10,["x"]  'endofdata

  GOTO main



chk_interrupt:    'chk for movement
  GOSUB read1 ' call measurement routine
  v1 = 5 * val / 255
  GOSUB read1 ' call measurement routine
  v2 = 5 * val / 255
  C= V1-V2
  IF C=0 THEN main
   DEBUG CR, "int" , CR ' display the value
  SERIN 2,25389,10,send_data_int,[STR indata\1] 'chk to see if other is
sending data if so wait
GOTO wait_for_Tx

send_data_int:

  SEROUT 1,25389,10,["a"] 'first contact
```

```
  SERIN 2,25389,100,main,[STR indata\1]
   ' DEBUG "send_data_int",STR indata   ,CR
  SEROUT 1,25389,10,["y"]
GOTO send_data

wait_for_Tx :

    PAUSE 1000'wait to send
GOTO chk_interrupt

send_data:

HIGH CS
LOW CS
LOW CLK
PULSOUT CLK, 210
SHIFTIN DataOutput,CLK,MSBPOST,[adcBits\8]
v = 5 * adcBits / 255

  SEROUT 1,25389,10,[DEC1 v]
'DEBUG "8-bit binary value: "    , BIN8 adcBits
'DEBUG CR,CR,  "Decimal value: "  , DEC3 adcBits
DEBUG CR, "DVM Reading:     ",     DEC3 v,  "    Volts"    '  ¡à new
line
'PAUSE 1000
'DEBUG CR
GOTO main

read1:
  LOW cl ' turn on detector for reading
  rl:
  IF IN15 = 0 THEN rl ' wait for input high
  SHIFTIN dt, cl,MSBPOST,[val\8]
  HIGH cl ' turn detector off
  PAUSE 1 ' let detector reset
RETURN
```

APPENDIX B

SIIMULATION SOURCE CODE FOR TEST1

```
'{$STAMP BS2}
'{$PBASIC 2.0}

x VAR Byte
Y VAR Byte

OUTPUT 1 'Make pin 1 an output pin.
 INPUT 2 'Make pin 1 an output pin.
Baudmode CON 16468
C CON 1010

DEBUG CLS

FOR X=1 to 100
  FOR y=1 to 1000
     SEROUT 1,10,Baudmode,[C]
  END

END
```

SIIMULATION SOURCE CODE FOR TEST2 AT TX

```
'{$STAMP BS2}
'{$PBASIC 2.5}
'-----------------------------------------------------
' variable declarations

adcBits VAR Byte

counter VAR Byte
v1 VAR Byte
v2 VAR Byte
```

```
val VAR Byte
C VAR Byte
sensor VAR Byte
indata VAR Byte(6)
v VAR v1.BIT0
'-----------------------------------------------------
' constant declarations
cl CON 14 ' pin 14 is output (clock)
dt CON 15 ' pin 15 is the input (data)
RF_baud CON 17197
CS PIN 9
CLK PIN 10
DataOutput PIN 11
x VAR Byte
y VAR Byte
sPin   CON   16
Baud   CON   84
INPUT dt ' make pin 15 the input
HIGH cl ' make pin 14 output and high


v="1"
SEROUT sPin,Baud,["DATA,TIME,", ",", DEC1 v,CR]
FOR y=0 TO 1
  FOR x=0 TO 255
SEROUT 1,RF_baud,10,[DEC3 x]
  NEXT
NEXT
SEROUT sPin,Baud,["DATA,TIME,", ",", DEC1 v,CR]
```

SIIMULATION SOURCE CODE FOR TEST2 AT RX

```
' {$STAMP BS2}
' {$PBASIC 2.0}
```

```
TM CON 14
FM CON 15
sPin   CON   16
RF_baud CON 17197
Baud   CON   84
'Declare variables
ss VAR Byte 'seconds
mm VAR Byte 'minutes
hh VAR Byte 'hours
dd VAR Byte 'days
mo VAR Byte 'months
yl VAR Byte 'years low
yh VAR Byte 'years high
v VAR Byte 'seconds
x VAR Byte
y VAR Byte
HIGH FM 'ensure no spurious start bit
  PAUSE 10



SetTimeCommand:'set TO 0:00:00AM,

  SERIN 1,RF_baud,2000,telltime,[DEC1 v]    'when the first data comes
set clock
  SEROUT FM,Baud,[$55,$00,$00,$00,$00,$03,$06,$61]
DEBUG "Alarm: ",DEC2 hh,":",DEC2 mm,":",DEC2 ss," ",DEC2 mo,"/",DEC2
dd,"/19",DEC2 yl, CR
FOR y=0 TO 200
  FOR x=0 TO 200

   SERIN 1,RF_baud,2000,telltime,[DEC3 v]
   SEROUT sPin,Baud,[ DEC3 v,CR]
  NEXT
NEXT
```

```
telltime:
SEROUT FM,Baud,[$55,$02]
SERIN TM,Baud,5000,SetTimeCommand,[ss,mm]',hh,dd,mo,yl]
DEBUG "Alarm: ",DEC2 hh,":",DEC2 mm,":",DEC2 ss," ",DEC2 mo,"/",DEC2
dd,"/19",DEC2 yl, CR
    SEROUT sPin,Baud,["DATA,TIME,", ",", DEC3 x,CR]
        SEROUT sPin,Baud,["DATA,TIME,", ",", DEC3 y,CR]
PAUSE 5000
```

SIIMULATION SOURCE CODE FOR PC USING ETHERNET

```
'{$STAMP BS2}

'{$PBASIC 2.5}

sensor VAR Byte(6)
y VAR Byte
v VAR Byte(3)
counter VAR Byte(2)
car VAR Byte 'which sensor

Baud   CON   84
sPin   CON   16  'Serial Pin - P16, Programming port
RF_baud CON 19697


SEROUT sPin,Baud,[CR]   'Send a lone CR to ensure StampDAQ buffer is
ready
Configure:

SEROUT sPin,Baud,[CR,"LABEL,TIME, TX, REC_DATA",CR]  'Label 3 columns
with TIME, X, and SIN X

SEROUT sPin,Baud,["CLEARDATA",CR]
```

```
main:
counter=0
SERIN 1,RF_baud,[STR sensor\1]
IF sensor="a" THEN maina
IF sensor="b" THEN mainb
'car="e"
SEROUT sPin,Baud,["DATA,TIME,", STR sensor, ",", DEC1 v,CR]
PAUSE 10    '1 second wait before starting again
GOTO main




maina:


SEROUT 0,RF_baud,10,["a"]
SERIN 1,RF_baud,100,main,[STR y\1]
car=sensor

GOSUB get_data

GOTO main


 mainb:
 SEROUT 0,RF_baud,10,["b"]
SERIN 1,RF_baud,100,main,[STR y\1]

 car=sensor
GOSUB get_data

GOTO mainb
```

```
get_data :
'SEROUT 0,25389,10,["r"]
SERIN 1,RF_baud,[DEC1 v]
DEBUG "ans=",DEC3 v   ,CR
 '    SEROUT sPin,Baud,["DATA,TIME,", STR sensor, ",", DEC3 v,CR]
GOTO main
```

SIIMULATION SOURCE CODE FOR SENSOR USING ETHERNET

```
'{$STAMP BS2}
'{$PBASIC 2.5}
'-----------------------------------------------------
' variable declarations

adcBits VAR Byte

counter VAR Byte
v1 VAR Byte
v2 VAR Byte
val VAR Byte
C VAR Byte
sensor VAR Byte
indata VAR Byte(6)
v VAR v1.BIT0
'-----------------------------------------------------
' constant declarations
cl CON 14 ' pin 14 is output (clock)
dt CON 15 ' pin 15 is the input (data)
RF_baud CON 25389
CS PIN 9
CLK PIN 10
```

```
DataOutput PIN 11


INPUT dt ' make pin 15 the input
HIGH cl ' make pin 14 output and high



main:
  sensor="a"
  counter=0
  GOSUB chk_interrupt
GOTO main

chk_interrupt:    'chk for movement
RANDOM v1

  IF v=0 THEN main
SERIN 2,RF_baud,10,send_data_int,[STR indata\1] 'chk to see if other is
sending data if so wait
GOTO wait_for_Tx

send_data_int:

  SEROUT 1,RF_baud,10,["a"] 'first contact
  SERIN 2,RF_baud,100,main,[STR indata\1]
    DEBUG "send_data_int",STR indata   ,CR
  SEROUT 1,RF_baud,10,["y"]
GOTO send_data

wait_for_Tx :

    PAUSE 1000'wait to send
GOTO chk_interrupt

send_data:
```

```
HIGH CS
LOW CS
LOW CLK
PULSOUT CLK, 210
SHIFTIN DataOutput,CLK,MSBPOST,[adcBits\8]
v = 5 * adcBits / 255

  SEROUT 1,RF_baud,10,[DEC1 v]
DEBUG CR, "DVM Reading:     ",     DEC3 v,  "    Volts"    '  ¡à new
line
GOTO main

read1:
  LOW cl ' turn on detector for reading
  rl:
  IF IN15 = 0 THEN rl ' wait for input high
  SHIFTIN dt, cl,MSBPOST,[val\8]
  HIGH cl ' turn detector off
  PAUSE 1 ' let detector reset
RETURN
```

SIIMULATION SOURCE CODE FOR PC USING ROUNDROBIN

```
'{$STAMP BS2}
'{$PBASIC 2.5}

sensor VAR Byte(6)
y VAR Byte

v1 VAR Byte
v2 VAR Byte
v3 VAR Byte
v VAR Byte
v4 VAR Byte
```

```
v5 VAR Byte
v6 VAR Byte
v7 VAR Byte

va1 VAR Byte
va2 VAR Byte
va3 VAR Byte
va VAR Byte
va4 VAR Byte
va5 VAR Byte
va6 VAR Byte
va7 VAR Byte

counter VAR Byte(2)



Baud   CON   84
sPin   CON   16  'Serial Pin - P16, Programming port
RF_baud CON 17197

SEROUT sPin,Baud,[CR]   'Send a lone CR to ensure StampDAQ buffer is
ready
Configure:

SEROUT sPin,Baud,[CR,"LABEL,TIME, TX, REC_DATA",CR]  'Label 3 columns
with TIME, X, and SIN X

SEROUT sPin,Baud,["CLEARDATA",CR]

main:
counter=0
SERIN 1,RF_baud,[STR sensor\1]

IF sensor=127 THEN main
IF sensor="a" THEN maina
IF sensor="b" THEN mainb
```

```
SEROUT sPin,Baud,["DATA,TIME,", STR sensor, ",", DEC1 v,CR]


GOTO rr


maina:
SEROUT 0,RF_baud,10,["a"]
SERIN 1,RF_baud,100,main,[STR y\1]
'DEBUG "main_a=",STR sensor   ,CR
GOSUB get_data
GOTO main

mainb:
SEROUT 0,RF_baud,10,["b"]
SERIN 1,RF_baud,100,main,[STR y\1]
'DEBUG "main_b=",STR sensor   ,CR
GOSUB get_data
GOTO mainb


get_data :
'SEROUT 0,25389,10,["r"]
SERIN 1,RF_baud,[DEC1 v]
SERIN 1,RF_baud,[DEC1 v1]
SERIN 1,RF_baud,[DEC1 v2]
SERIN 1,RF_baud,[DEC1 v3]
'DEBUG "i"
'SERIN 1,RF_baud,[DEC1 v4]
'SERIN 1,RF_baud,[DEC1 v5]
'SERIN 1,RF_baud,[DEC1 v6]
'SERIN 1,RF_baud,[DEC1 v7]
'SERIN 1,RF_baud,[DEC1 va]
'SERIN 1,RF_baud,[DEC1 va1]
'SERIN 1,RF_baud,[DEC1 va2]
'SERIN 1,RF_baud,[DEC1 va3]
```

```
'SERIN 1,RF_baud,[DEC1 va4]
'SERIN 1,RF_baud,[DEC1 va5]
'SERIN 1,RF_baud,[DEC1 va6]
'SERIN 1,RF_baud,[DEC1 va7]



 SEROUT sPin,Baud,["DATA,TIME,", STR sensor, ",", DEC1 v, ",", DEC1
v1,",", DEC1 v2,",", DEC1 v3 ,CR]'",", DEC1 v4, ",", DEC1 v5,",", DEC1
v6,",", DEC1 v7, CR]
   ' SEROUT sPin,Baud,["DATA,TIME,", STR sensor, ",", DEC1 va, ",",
DEC1 va1,",", DEC1 va2,",", DEC1 va3, ",", DEC1 va4, ",", DEC1 va5,",",
DEC1 va6,",", DEC1 va7, CR]


GOTO main

rr:
SEROUT 0,25389,10,["e"] 'for error
GOSUB maina
GOSUB mainb
GOTO main
```

SIIMULATION SOURCE CODE FOR SENSOR USING ROUNDROBIN

```
'{$STAMP BS2}
'{$PBASIC 2.5}

'-----------------------------------------------------
' variable declarations
adcBits VAR Byte
v VAR Byte
counter VAR Byte
v1 VAR Byte
v2 VAR Byte
```

```
val VAR Byte
C VAR Byte
sensor VAR Byte
indata VAR Byte(6)
'-----------------------------------------------------
' constant declarations
cl CON 14 ' pin 14 is output (clock)
dt CON 15 ' pin 15 is the input (data)
CS PIN 9
CLK PIN 10
DataOutput PIN 11

INPUT dt ' make pin 15 the input
HIGH cl ' make pin 14 output and high



main:
  GOSUB chk_collision
  GOSUB chk_interrupt

  sensor="a"
  counter=0
GOTO main

chk_collision:

  SERIN 2,25389,10,chk_interrupt,[STR indata\1]
  DEBUG CR,"collision_int",STR indata   ,CR
  IF indata="e" THEN send_data_int
  SEROUT 1,25389,10,["x"]  'end of data

RETURN



chk_interrupt:    'chk for movement
  GOSUB read1 ' call measurement routine
```

```
  v1 = 5 * val / 255
  GOSUB read1 ' call measurement routine
  v2 = 5 * val / 255
  C= V1-V2
  IF C=0 THEN main
   'DEBUG CR, "int" , CR ' display the value
  SERIN 2,25389,10,send_data_int,[STR indata\1] 'chk to see if other is
sending data if so wait
GOTO wait_for_Tx

send_data_int:

  SEROUT 1,25389,10,["a"] 'first contact
  SERIN 2,25389,100,main,[STR indata\1]
   ' DEBUG "send_data_int",STR indata   ,CR
  SEROUT 1,25389,10,["y"]
GOTO send_data

wait_for_Tx :
  PAUSE 1000'wait to send
GOTO chk_interrupt

send_data:

HIGH CS
LOW CS
LOW CLK
PULSOUT CLK, 210
SHIFTIN DataOutput,CLK,MSBPOST,[adcBits\8]
v = 5 * adcBits / 255

SEROUT 1,25389,10,[DEC1 v]
DEBUG CR, "DVM Reading:     ",     DEC3 v,  "    Volts"    '  ¡à new
line
GOTO main
```

```
read1:
  LOW cl ' turn on detector for reading
  rl:
  IF IN15 = 0 THEN rl ' wait for input high
  SHIFTIN dt, cl,MSBPOST,[val\8]
  HIGH cl ' turn detector off
  PAUSE 1 ' let detector reset
RETURN
```

APPENDIX C

C.1 CACULATIONS FOR TEST1 (From table 3.1)

Time to send data = total bits sent / baud rate + pause time = 200000 / 9600 + 10 = 30.8 seconds (cable communication to PC)

Execution time = number of operations / CPU speed = 20000×5 / 4000 = 25 seconds

C.2 CACULATIONS FOR TEST2

C.2.1 Calculated Transmission (From table 3.2)

For every baud rate the calculation is done by

Time to send = total bits sent / baud rate = 512×3 / X seconds

Clock operation time = 1.2 seconds

Where X is the baud rate

C.2.2 Program Overhead (From table 3.2)

Instruction execution time = number of operations / CPU speed = 512×8 / 4000 = 1 seconds

Time to send = total bits sent / baud rate = 512×3 / 9600 = .2 seconds (internal communication)

Pause time = total pause for each bits sent + for reset = 512×10×3/1000+.2 = 15.6 seconds

Total = 16.8 seconds

C.3 CACULATIONS FOR BEST CASE

C.3.1Calculated Transmission (From table 3.4)

For every baud rate the calculation is done by

Time to send = total bits sent / baud rate = 512×10×3 / X seconds

Where X is the baud rate.

C.3.2 Program Overhead (From table 3.4)

Instruction execution time = number of operations / CPU speed = 201×14 / 4000 = .7 seconds

Time to send = total bits sent / baud rate = 201×1 / 4000 = .02 seconds (internal communication)

Pause time = total pause for each bits sent 201×10×5/1000 = 10.05 seconds

Total = 10.8 seconds