Gene Shokes, gshokes@ufl.edu, 281-1543 Joshua Rogers, fatman31@ufl.edu, 850-554-6583

EEL 4914 Electrical Engineering Design

Project Design Report:
Equine Dental Camera

Abstract:

Equine Dentistry is normally provided in a rustic setting such as a stable or barn where horses are lightly tranquilized and restrained in the standing position as dental work is provide. Restraints do not impede the over excited horse from flailing and causing damage in the immediate area. The possible violent reaction of the horse coupled with the rustic features of location prevents the Veterinarian Technician from placing expensive computer and electronic equipment in close proximity to the horse. However, there is an ever growing need to document, by digital photography, the dental disposition before and after service. Thus, this project consists of building an equine dental camera in the likeness of a gun with a mirror mounted to the top of the body and a camera on the end for inspecting the horse's teeth. Non native lighting will be provided by microprocessor driven pulse wave modulated LED's varied in intensity by user input into the microprocessor. The images from the camera will be displayed on a color LCD screen mounted on the back of the gun and will also be wirelessly transmitted to a distant laptop via RF transmitters. There will be streaming video, as well as an option to capture screen shots.

Table of Contents

*Appendices only included in soft copy.

Features:

The equine dental camera provides the equestrian dental industry with a new product that incorporates inspecting and documenting dental procedures. Throughout the semester, there were many features that were discussed and wanted to be implemented. In the end, not all features could be realized. The following features were incorporated in the product:

- CMOS camera.
- Adjustable brightness for the camera's LED's.
- Wireless transmission of a picture.

The Competition:

There are not any products available for the equine industry, but there is for the human dentistry field. Our product should be less than this and built with the equine industry in mind. Figure 1 shows the human type.

#AIC888 / #AIC900
AdvanceCAM Intraoral Camera (wireless)

- camera weights only 2 ounces
- high resolution, auto focus
- view image in full or quad screen
- 3 hours of operation before recharging
- four independent wireless channels
- high, medium and low power indicator (LED)
- 8"L x 1'W x 1.25" H

#AIC900
Wireless Transmitter (included)

- 2.4 GHz, 8.5V charger
- LED channel indicator
- 2.0" x 1.0"

$1,650 To order, please phone us for an individual consultation to determine which combination of components best fits your needs

Figure 1

Technical Concepts:

There are several technical objectives that must be overcome to produce a competitive product into the equine dentistry industry. The unit has size constraints, durability issues, minimum operating time, and user functionality.

First, the unit must be small enough to fit into the horse's mouth. It must be able to reach to the back of the mouth and produce quality video of the teeth. The unit has to be thick enough to incorporate the mirror and camera as well. To achieve this, the unit will need to be between 0.75 and 1.5 inches thick. The end will also be tapered to allow it to reach to the rear of the mouth.

The next challenge will be making the unit durable enough to be used in the rustic surroundings of a barn or out in the fields. The unit will have to be waterproof since it will be used in the mouth of the horse. To achieve this goal, the mirror and camera housing will be sealed and the unit will be built as thick as possible. The thickness will help to make the unit more rigid and durable.

The third challenge will be maintaining a minimum operating time. The unit will need to be able to last for the entire time it takes to examine the horse. To overcome this problem, the equine dental camera will be implementing a 12V lead acid battery power supply. The unit will have to have a voltage regulator to step down the voltage for the microprocessor though.

The final challenge is functionality. The user will need to be able to operate the unit with one hand. All controls and functions will have to be readily available at one's finger tips. To accomplish this, the push buttons for power and capturing images will be implemented by way of a two trigger design. The LED brightness buttons will be on the rear, underneath the LCD screen.

Components:

To be able to provide the features listed above, the following parts in table 1 will be required:

Component	Options	Advantages
Camera	CMOS	Digital output
	CCD	Analog output
RF Transmitter/ Receiver	Nordic	Up to 2Mbps
	Xbee	Easy to interface
Push Buttons	NO	Would not source current
	NC	Cheaper
LED's	White	Cheap and available
	Color	Would not white out camera

FPGA	128 pin	Cheaper
	256 pin	More processing power
Microprocessor	Atmel	More built in functions
	Pic	Smaller
Color LCD	Accelevision	NTSC compatible
	PSP screen	Larger screen and clearer

Table 1

The camera that has been chosen was a CMOS and has a VGA resolution of 640x480. There are cameras available with higher resolutions, however for this design the video being captured is close to the camera lens and VGA will be plenty of resolution. The cost is also cheaper for less resolution.

The RF transmitter and receiver pair chosen was the Nordic. There are models available with a larger range, however the distance between the individual and the laptop should not be greater than 50 feet.

The push buttons are generic buttons used for turning power on and off and another button for activating a still shot. There are many different types, but the easy choice is the free one that will not source current.

The LED's incorporated into the camera housing only need to illuminate the local region where the camera is being used. The LED system needs to be dark enough that it doesn't white out the camera feed. There are many different options for LED's, but the white ones available in lab have the right brightness.

Two Atmel microprocessors were used in the project. The microprocessors will be used to monitor the brightness and capture buttons and adjust the pulse width modulation driving the LED's. They were also used to transmit a signal to the laptop to take a screen shot when the button is pressed.

The FPGA initially chosen for the project was the Cyclone II EP2C20F256C6 which is a dash 6 speed, 256 pin ball grid mount FPGA with 18752 LE's, 152 user I/O's, 239620 bits of memory, and 4 embedded PLL's. However, due to time constraints the UF 4712 board was used to implement the project.

Architecture:

Figure 2 shows the initial design.

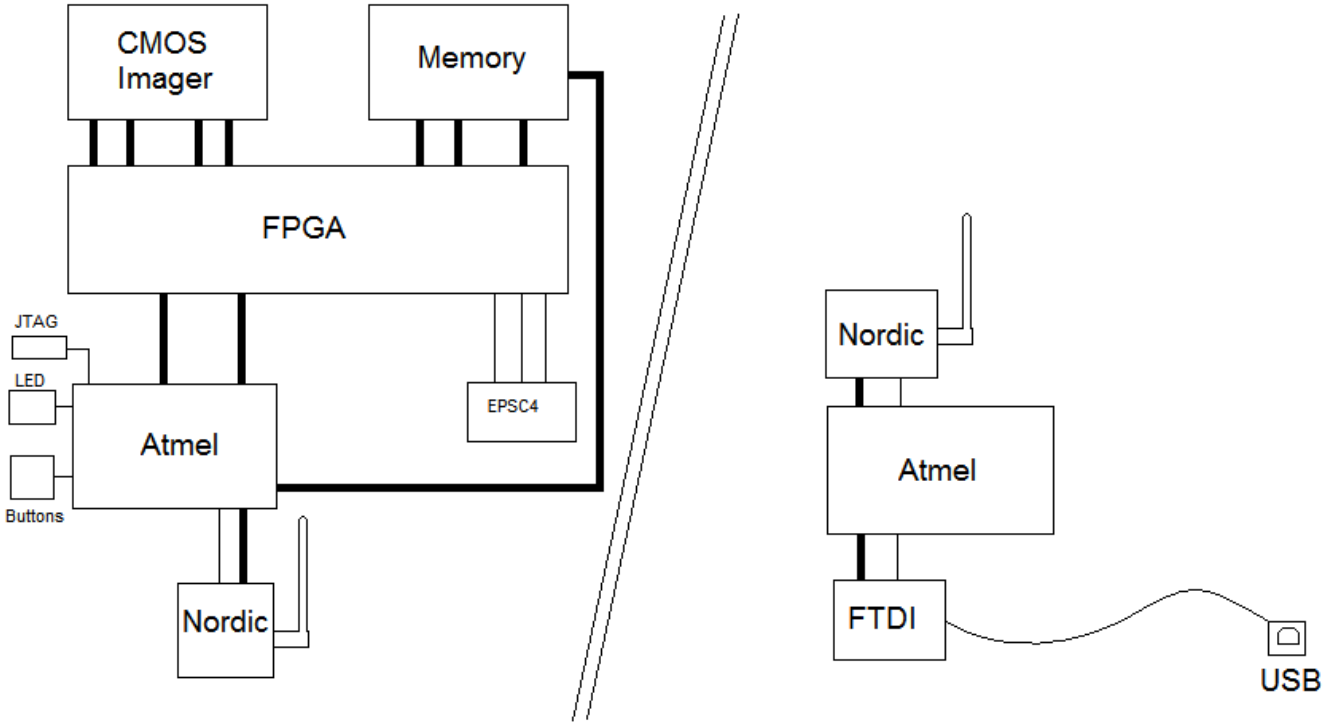

Figure 2

The initial design incorporates two Atmel processors and a Cyclone 2 FPGA. The FPGA interfaces with the CMOS image chip and memory, and the Atmel processors interface with the memory, FTDI chip, and the Nordic chip.

The gun side contains all of the components on the left side of figure 2. The Atmel continuously runs a 10% duty cycle PWM outputting to a mosfet that drives the LED's. There are three interrupts: LED brighter, LED darker, and capture. The LED brighter interrupt increases the duty cycle of the PWM to increase the power driven to the LED's. The LED darker interrupt decreases the duty cycle of the PWM to decrease the amount of power used to drive the LED's. The capture interrupt tells the FPGA to store the current frame into memory. From the memory, the Atmel receives the data that the FPGA addresses on the memory address bus. The data is then sent to the Nordic chip to be wirelessly transmitted to the receiver station.

The receiver station continuously keeps the USB bus out of low power sleep mode. This allows the Nordic to have plenty of power to receive the file. When the Nordic

receives data, it interrupts the Atmel processor and the data is loaded from the Nordic chip to the Atmel where it is sent to the FTDI chip to be loaded into a computer.

Figure 3 shows what the final product is envisioned to look like.

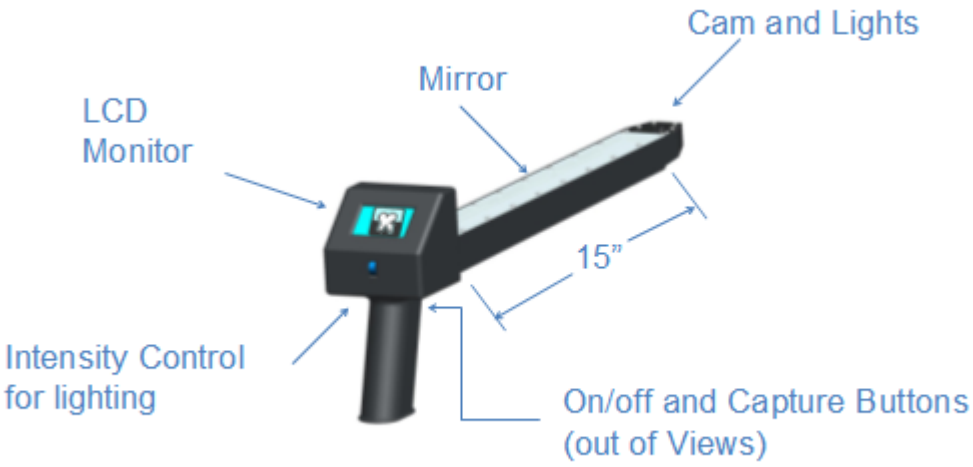

Figure 3

Flow Charts:

Figure 4 shows the flow chart for the Atmel processor on the gun side.

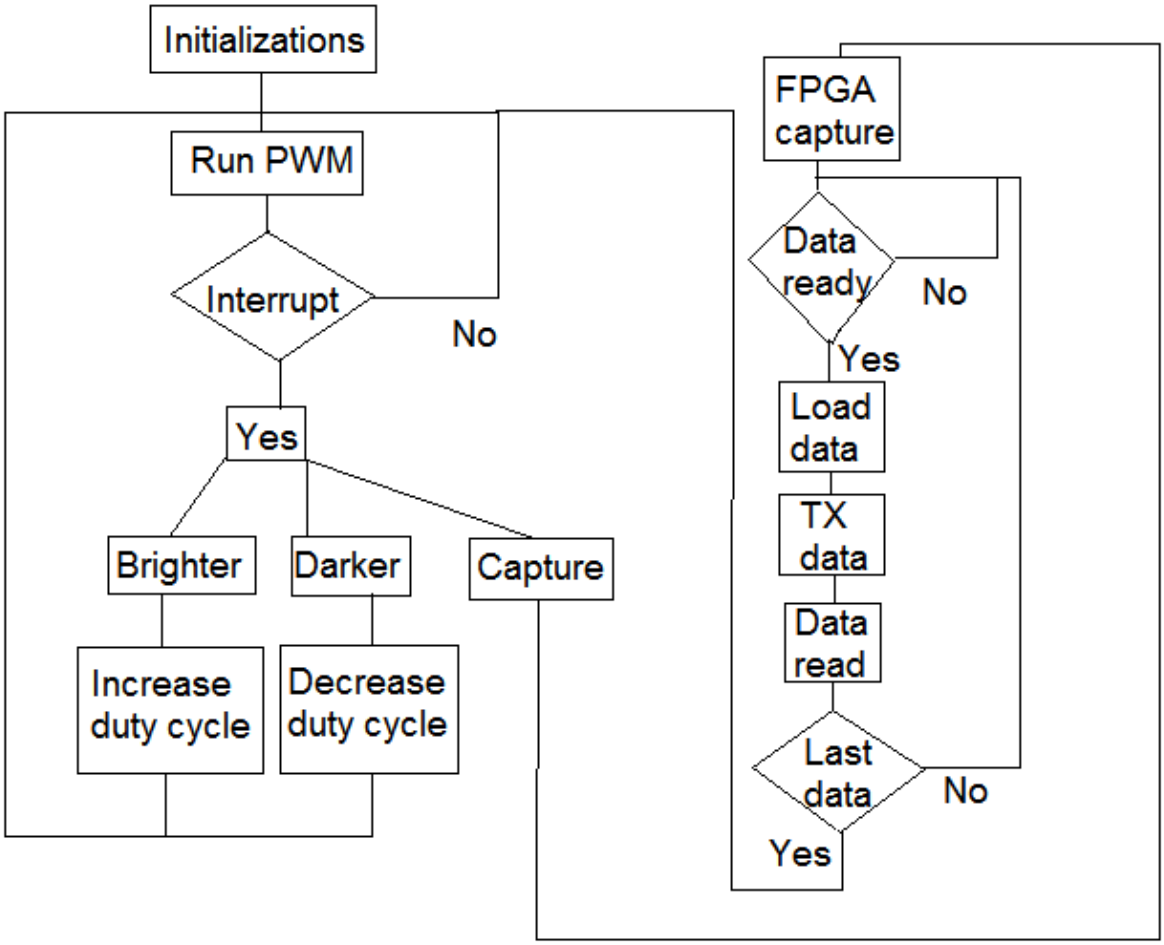

Figure 4

Figure 5 shows the flow chart for the Atmel on the receiver side.

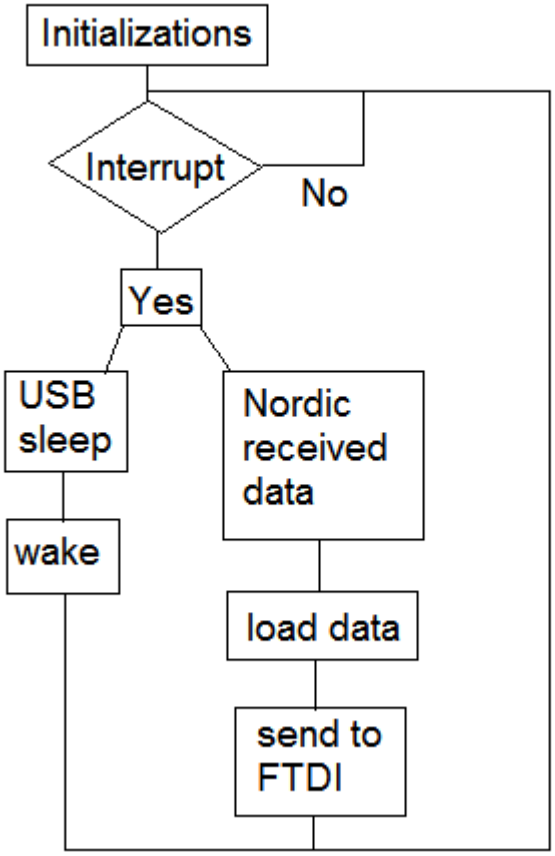

Figure 5

Figure 6 shows the flow chart for the FPGA.

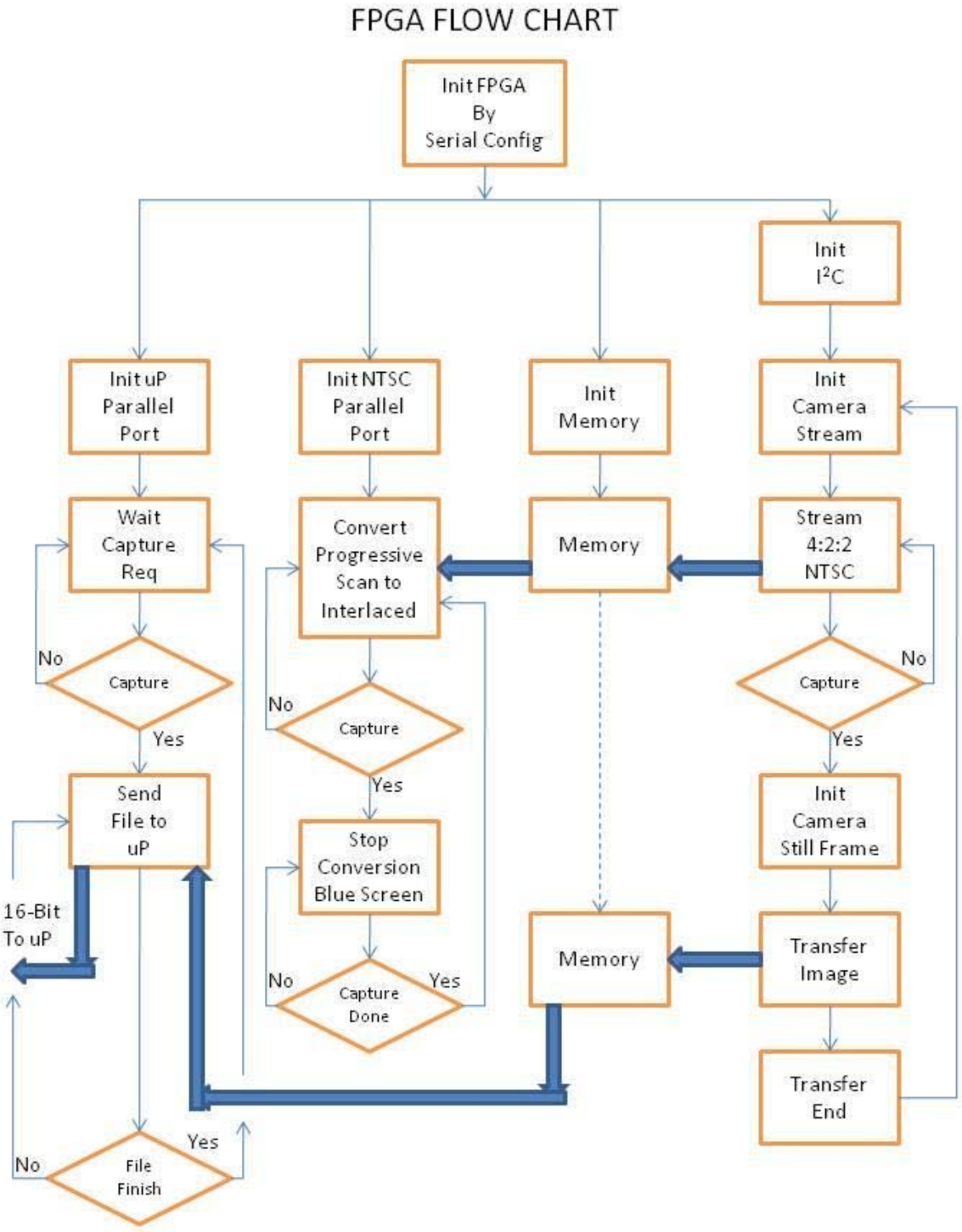

Figure 6

Labor Division:

Table 2 shows the tasks at hand and the individual who is responsible for that task.

Responsibility	Person	Percentage
Research	Josh	50%
	Gene	50%
Writing reports and presentations	Josh	50%
	Gene	50%
Design PCB's	Josh	25%
	Gene	75%
Interface with Nordic	Josh	
Interface with camera	Gene	
Interface with laptop	Josh	
Interface SRAM	Gene	
write C for Atmel processors	Josh	
write VHDL for FPGA	Gene	
design casing in ProE	Josh	25%
	Gene	75%
populate PCB's	Josh	60%
	Gene	40%
build gun	Josh	20%
	Gene	80%
build computer case	Josh	

Table 2

Bill of Materials:

Table 3 contains the cost of the materials used in the project.

Component	Cost	Use
Camera	$134	Capture video and still shots
RF Transmitter/ Receiver	$87	Wireless transmission of video
Push Buttons	Free	power and capture features
LED's	Free	Lighting for camera
Potentiometer	Free	To adjust LED brightness
Microprocessor	Free	Drive LED's and capture function
Color LCD	$75	View from camera
FPGA	$70	Integrate camera, memory, and microprocessor
Misc. resistors/caps/etc	$50	Bring all components together
TOTAL	$416	

Table 3

Timeline:

Table 3 shows the up to date Gantt chart that reflects how long each task was planned and how long it actually took.

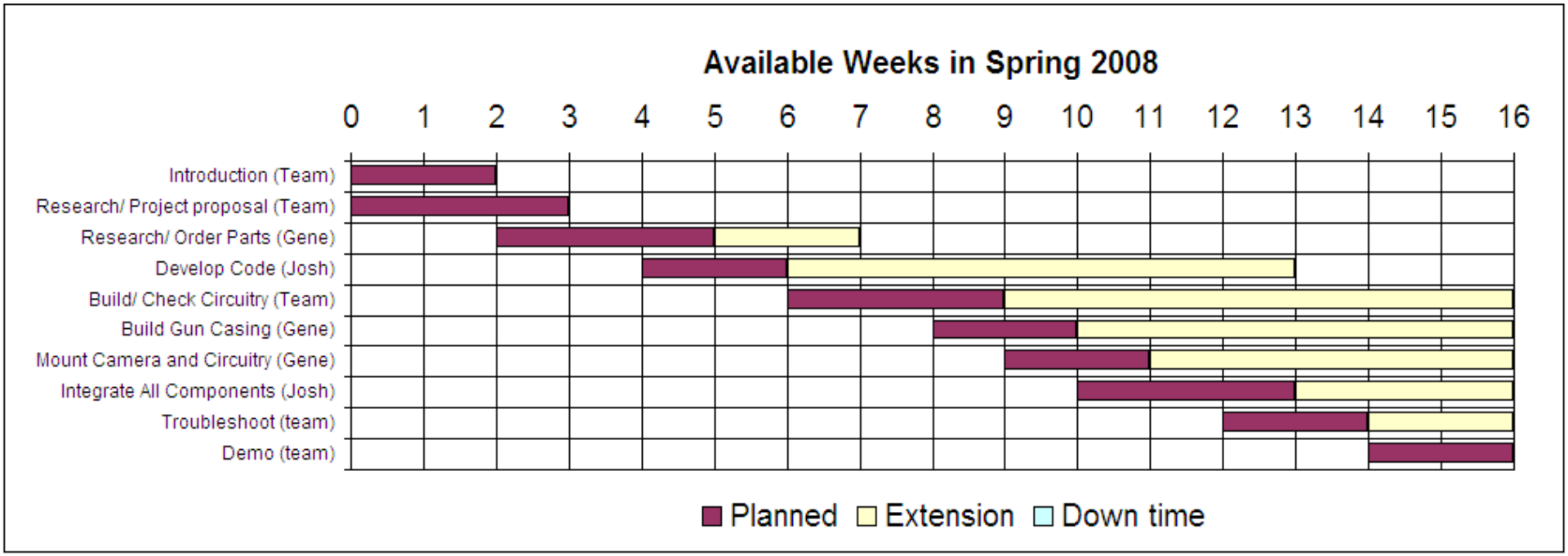

Table 4

Appendices:

Atmel Code for Gun Side:

```
//Joshua Rogers
//Senior Design Project Code
//Gun Side Final Software


#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <inttypes.h>
#include <stdio.h>


//Variables
unsigned int dataa;
unsigned int datac;
unsigned int g=0;
unsigned int h=0;
unsigned int i=0;
unsigned int t=0;
unsigned int y=0;
unsigned int x;

//Sub programs

void INITIAL(void)
{

DDRB |= 0x16;          //Portb.4 output for PWM and MOSI SCK for SPI and LED
DDRC &= ~(0xFF);       // Port C all inputs
DDRA &= ~(0xFF);       // PortA all inputs
DDRD &= ~(0x0F);       //Portd.0-3 all inputs interrupts
DDRD |= 0x30;          //PORTD.4-5 outputs to nordic
DDRE &= 0x00;          //Porte all inputs
DDRF |= 0x0F;          //Portf3-0 outputs
DDRG |=0x0F;           // PortG3-0 outputs


// Set up external interrupts
```

```
EIMSK &= ~(0xFF);
EICRA &= ~(0xFF);               //low level generates int
EIFR |= 0x07;
EIMSK |= 0x07;              //Int2-0 turned on
SREG |= 0x80;               //Set I-bit in SREG


//Set up PWM by timers and toggle PORTB 4 on interrupts

TCCR2 |= 0x01;    //turn timer2 on
TCCR1A &= 0x00;         //turn timer 1 on
TCCR1B &= 0x00;
TCCR1B |= 0x01;
OCR2 |= 0x10;       //output compare is how to adjust
OCR1AH |= 0x05; //output compare to 10KHz period
OCR1AL |= 0xBE;
TIMSK |=0x90;     //set interrupt for OC on timer 1 and 2 on

sei();  //global interrupt on


//Set up SPI

SPCR &= 0x00;
SPCR |= 0x50; //MSB first, SPI enable, 128 Master
SPSR &= ~(0xFF);          //f/4 for SPI
}

void wait(void)      //10us delay 160*16Mhz=10us
{
redo:
if(x < 150)   //less than 160 but Nordic will operate correctly
{
x=x+1;
goto redo;
}

x=0;
}

void debounce(void)       //debounce switches
```

```
{
i=0;
redo3:
if( i < 10)
{
h=0;
redo2:

if( h < 64000 )
{
redo1:
if(g < 64000)
{
g=g+1;
goto redo1;
}
g=0;
h=h+1;
goto redo2;
}
h=0;
i=i+1;
goto redo3;
}
i=0;
}


void SPITX( unsigned int reg, unsigned int dat)      //used for nordic setup
{
PORTD &= 0xDF;

SPDR = reg;
while(!(SPSR & (1<<SPIF)))

wait();

SPDR = dat;
while(!(SPSR & (1<<SPIF)))

wait();
```

```
PORTD |= 0x20;

}

void SPITX2( unsigned int reg2, char dat1, char dat2)      //used for data sending
{
PORTD &= 0xDF;

SPDR = reg2;
while(!(SPSR & (1<<SPIF)))

wait();

SPDR = dat1;
while(!(SPSR & (1<<SPIF)))

wait();


SPDR = dat2;
while(!(SPSR & (1<<SPIF)))

wait();

PORTD |= 0x20;

}

void NORDIC_INT_TX(void)  //initialize Nordic in TX mode
{
SPITX(0x21,0x00);       //disanle AA
SPITX(0x22,0x01);       //enable data pipe 0
SPITX(0x23,0x03);       //5 byte addressing
SPITX(0x24,0x00);       //retransmit nothing
SPITX(0x25,0x34);       //channel 50
SPITX(0x26,0x0E);       //2 meg high power
SPITX(0x31,0x02);       //pipe 0 2 byte pipe width
SPITX(0x20,0x72);       //no int and PU TX
}

void getdataandtx(void)
{
```

```
        retry:
        if( (PINE == 2) | (PINE ==3) )  //data ready signal
        {
        dataa = PINA;
        datac = PINC;
        SPITX2(0xa0,dataa,datac);       //load data to nordic
        PORTD|=0x10;                    //toggle CE to TX
        wait();
        PORTD &=~(0x10);
        }
        else
        {
        goto retry;
        }

}

void capture(void)
{
        PORTF |= 0x01;
        PORTF &= ~(0x01);
}

void read(void)
{
        PORTF |= 0x02;
        PORTF &= ~(0x02);
}

void fpgareset(void)
{
PORTF &= ~(0x04);
PORTF |= 0x04;
}

//MAIN

int main()    //initialize everything and wait for interrupts
{

INITIAL();
```

```
PORTF |= 0x04; //fpga

NORDIC_INT_TX();

debounce();


while(1)
        {
        wait();         //wait for interrupts
        }

}

//Interrupts

ISR(INT0_vect)    //Capture and send
{
        debounce();
        //1.capture to FPGA
        //2.wait for data ready
        //3.watch "last" and continue to send data and use data ready
        //and data request
        //4.send double of end file if half word occurs
        fpgareset();  //reset to FPGA
        capture();    //send to FPGA
        getdataandtx();
start:
        read();
        if ( (PINE == 1) | (PINE == 3))
        {
        getdataandtx();
        read();
        return(0);
        }
        getdataandtx();
        goto start;
}

ISR(INT1_vect)    //interrupts brighter (increase duty cycle)
{
        debounce();
```

```
	unsigned int o;
	o = OCR2;
	if ( o >= 231 )
	{
	OCR2 = 0xFF;
	return(0);
	}
	else
	{
		o=o+25;
		OCR2 = o;
	}
	return(0);

}

ISR(INT2_vect) //Darker (decrease duty cycle)
{

	debounce();
	unsigned int p;
	p = OCR2;
	if ( p <= 26 )
	{
	OCR2 = 0x01;
	return(0);
	}
	else
	{
		p=p-25;
		OCR2 = p;
	}
	return(0);
}

ISR(INT3_vect)
{

return(0);
}

//PWM for LEDS via timer interrupts. This controls the frequency
```

```
//to manufacturers recommended period.

ISR(TIMER1_COMPA_vect)
{
        PORTB |= 0x10;    //LED on
        TCNT1L = 0x00;
        TCNT2 = 0x00;
        TCNT1H = 0x00;
        return(0);
}

ISR(TIMER2_COMP_vect)
{

        PORTB &= ~(0x10);       //LED off
        return(0);
}
```

Atmel Code for Receiver Side:

```
//Joshua Rogers
//Senior Design Project Code
//Computer Side Software Demo


#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <inttypes.h>
#include <stdio.h>




void INTIAL(void)
{

DDRA |= 0xFF;
DDRB |= 0x06;             //Portb output for  MOSI SCK for SPI
DDRD &= ~(0x0F);          //Portd.0-3 all inputs interrupts
DDRE &= 0x00;
```

```
DDRE |= 0xC3;              //Porte.0,1,6,7 outputs

PORTE |= 0x40;

// Set up external interrupts

EIMSK &= ~(0xFF);
EICRA |= 0x03;             //low level generates int1,2,3 rising edge for 0
EIFR |= 0x0F;
EIMSK |= 0x0F;             //Int3-0 turned on
SREG |= 0x80;              //Set I-bit in SREG
sei();  //global interrupt on


//Set up SPI

SPCR &= 0x00;
SPCR |= 0x50;      //MSB first, SPI enable, 128 Master, Int enable
SPSR &= ~(0xFF);         //f/4 for SPI
}

void setcsn(void)
{
PORTE |= 0x02;
}

void resetcsn(void)
{
PORTE &= ~(0x02);
}

void setce(void)
{
PORTE |= 0x01;
}

void resetce(void)
{
PORTE &= ~(0x01);
}

unsigned int x;
```

```
void wait(void)      //16MHz*2*250=30us delay
{
redo:
if(x == 250)
{
goto here;
}
else
{
x=x+1;
goto redo;
}
here:
x=0;
}

void SPITX( unsigned int reg, unsigned int dat)
{
resetcsn();

SPDR = reg;
while(!(SPSR & (1<<SPIF)))

wait();

SPDR = dat;
while(!(SPSR & (1<<SPIF)))

wait();

setcsn();

}

char R;
char E;

void SPIRX2( unsigned int rex)
{

resetcsn();
```

```
SPDR=rex;
while(!(SPSR & (1<<SPIF)))

wait();

SPDR=0x00;
while(!(SPSR & (1<<SPIF)))

wait();

R = SPDR;

SPDR=0x00;
while(!(SPSR & (1<<SPIF)))

wait();

E = SPDR;

setcsn();
}



void FTDITX(void)
{
//while((PINE &= 0x20)==32)
//{wait();}
PORTA = R;
PORTE |= 0x80;
PORTE &= ~(0x80);
PORTA = E;
PORTE |= 0x80;
PORTE &= ~(0x80);
}


void NORDIC_INT_RX(void)
{

SPITX(0x21,0x00);         //disable AA
```

```
SPITX(0x22,0x01);       //Enable data pipe 0
SPITX(0x23,0x03);       //5 byte addressing
SPITX(0x24,0x00);       //re-transmit disabled
SPITX(0x25,0x34);       //channel 50
SPITX(0x26,0x08);       //2 mega per second
//SPITX(0x31,0x20);     //32byte wide pipe0
SPITX(0x31,0x02);       //1 byte info
SPITX(0x20,0x33);       //RX_DR int,PU,RX_en
setce();
}


unsigned int T=0;
unsigned int q=0;

int main()
{

INTIAL();

setcsn();

wait();

NORDIC_INT_RX();

chill:
wait();
goto chill;

}

ISR(INT0_vect)    //sleep int from FTDI
{
DDRE |= 0x10;
PORTE |= 0x10;
PORTE &= ~(0x10);
wait();
DDRE &= ~(0x10);

return(0);
}
```

```
ISR(INT1_vect)
{
return(0);
}

ISR(INT2_vect)
{
return(0);
}

ISR(INT3_vect)    //RX interrupt from Nordic
{

resetce();

SPIRX2(0x61);
FTDITX();   //sending data to FTDI
SPITX(0x27,0x40);

setce();

return(0);
}
```