THE MOBILE ARDUINO UNO

ERIC MOSER

11/10/11

Embedded Systems

EE 2920-001

Milwaukee School of Engineering

Electrical Engineering Department

Table of Contents:

Table of Figures

Introduction

There were four main objectives to this lab. First of all, I needed to look into the Sony remotes and how they output a square wave signal of information. Secondly, I had to figure out how to receive the IR signal data and store it for use in my program. The next step to the lab was to figure out with calculations, how to decode my stored IR signal into more useable data. Finally, the last step of this lab was to come up with my own implementation of concepts we have learned over this term with the pressing of different buttons on my remote.

Theoretical Concepts

Sony Protocol:

First of all, the Sony remotes output an Inferred Square wave or IR signal. This signal is 13bits long, starting with the first bit called the start bit. This start bit is designated by a 2.4ms high signal and it is used to tell any electronic readers that "Hey I'm sending data." The next 7 bits are called the command bits and will tell the IR sensor what button was pressed. This button could be anything from channel up to volume control and even the number pad values. The last 5 bits in the signal are called the address bits and will tell the sensor what device it is sending to. This device could be the TV, DVD player or other Sony products. It is also important to note that this output is serial communication therefore, 1 bit is sent out at a time. Logic "0" in Sony Protocol is designated by a 1.2ms high followed by a 0.6ms low. Logic "1" in Sony Protocol is then a 0.6ms high follow by a 0.6ms low. Also stated above, the start bit consists of a 2.4ms high followed by a 0.6ms low. Overall, this Sony Protocol is quite simple to understand and can be read by the input capture unit on our Atmega 328p microcontrollers.

DC Motors:

For the optional or "Impress Me" section of this lab I choose to mobilize my microcontroller with the use of two DC motors and wheels. The DC motors work similar to the servo motors that we worked with in lab 8 were as a PWM signal caused movement of the motor. However, unlike the servo motors, a 100% duty cycle would cause the DC motor to keep spinning. However, due to the current needed to drive the DC motors and the fact that the 20mA from our microcontrollers, a motor driver chip was needed. This motor driver chip acted as a current amplifier giving the motors all the current that they would need. Also due to the fact that our motors had gears in them, it was necessary to slowly speed up and then slowly speed down to switch directions. I used a L 293D motor driver chip from the tec. Center.

Battery System:

As to the issue of current draw described above in the DC motor I had to use the wall adapter to help give my motors the power they needed. As a note, the wall adapter wasn't fully needed, but it did increase the speed of the motors. After mounting wheels on my motors, and wanting to get some range from the outlet, I looked into creating a battery pack for my system. I took the 4 AA battery holder from my EE100 kit to hold my batteries. I then picked up the proper barrel connector from RadioShack. After sotering the two together, I got my system mobile. It's also noted that is runs slower than the USB and the wall connector.

Solution

Configuring T/C:

There were many aspects to the final solving of this lab. I first set Timer Counter1 on CTC mode with a prescale of 1 and a top of 65535 (2^16). I then configured the input compare turned on the input compare interrupt and turned on global interrupts. I choose Timer Counter 1 because when it is set the way I have it, it will overflow every 4ms leaving me with more than enough time to collect my data.

Collecting the Signal:

I wrote all my collection data in the input compare interrupt. I created my program so that the first time that the interrupt was triggered on a falling edge, it would set TCNT1 equal to 0 and set the Input Capture Interrupt to rising edge trigger. Therefore, the second time the ISR was triggered, I was able to store my ICR1 value into an array and then set the IRS back to falling edge trigger. This method results in an ICR1 value that is exactly the value I am trying to measure and ensures that TCNT never overflowed. The start bit is the longest bit at 2.4ms and my T/C1 overflows at 4ms so I have time to spare. I then set a small if statement to refresh the system after all the bits were stored so that it would be ready for the next button input.

```
ISR(TIMER1_CAPT_vect)  FIGURE1
{
        if(check==0)
        {
                TCNT1=0;//setting tcnt back down
                TCCR1B|=(1<<ICES1); //raising edge
                check=1;
        }
        else if(check==1)
        {
                bits[i]=ICR1;//storing in array
                TCCR1B&=(~(1<<ICES1));//falling edge
                check=0;
                i++; //varrible for array
```

```
        }
        if(i==13)//refresh system
        {
                i=0;
                count=1;
        }
}
```

Reading the Signal:

I first checked if the first part of the array bits[0] or the start bit was equal to around 38400 or 2.4ms. If the start bit checked out, the rest of my program would then run. I then set this first part of the array to 0 just so I controlled what it was. I then wrote a for statement to convert my array to binary values in a second array based on how big the ICR1 value was. Next, I wrote another for loop, but this time it acted as a shift on the LCD display to print out each value of the binary array (LSB first). I also implemented a default case that would display that nothing has occurred and printing a notification on to LCD. The next step was to convert my binary array to a single value that I could use to exactly determine the button press for my motor function calls. I decided to convert my array to a decimal number. I calculated what the value of each spot in the binary value would be and add them all up to find my decimal value. This part of my code consisted of 13 if statements, which actually looks quite clean. I then printed this decimal value to the LCD under the line for the binary value of the ICR1. Depending on the decimal number, I printed the corresponding button designation on the LCD to the right of my decimal number.

```
        ////TRANSFER SYSTEM TO BINARRY//   FIGURE2
        if(startbit==1)//is start bit is good
        {
                for(int b=1; b<13; b++)
                {
                        if(bits[b]>19000)
                        {
                                result[b]=1;
                        }
                        if(bits[b]<12000)
                        {
                                result[b]=0;
                        }
                        }

//CONVERTING ARRAY TO DECIMAL//    FIGURE3
        if(count==1)
        {
        number=0;
        if(result[0]==1){number=number+1;}
        if(result[1]==1){number=number+2;}
        if(result[2]==1){number=number+4;}
        if(result[3]==1){number=number+8;}
        if(result[4]==1){number=number+16;}
        if(result[5]==1){number=number+32;}
        if(result[6]==1){number=number+64;}
        if(result[7]==1){number=number+128;}
        if(result[8]==1){number=number+256;}
        if(result[9]==1){number=number+512;}
        if(result[10]==1){number=number+1024;}
        if(result[11]==1){number=number+2048;}
```

```
if(result[12]==1){number=number+4096;}
count=0;
        }
```

Motor Solution:

I decided to make my microcontroller mobile with the use of two DC motors. I used Timer Counter 2 and outputted a fast PWM signal out to the enables of my motor driver chip. I set ports for my “in”s of my motor driver chip to control movement and polarity. I set ComA and ComB to outputs on and the corresponding ports to outputs. I used 5 buttons to control what my motors did. First with the forward button, I wrote both my OCR2A and B to ramp up to 155 or 100% duty at 20% duty increments. I also added some LEDs to this forward action. I had two red blinking leds and one bright white headlight led. Next, when the back button was pressed, the polarity of the motor was changed and the duty was once again ramped up. However, it got a little more difficult when I wrote my code for turning. I wrote one Com output to a 100% duty on ramp up and then turned it off and turned the second Com output to a smaller 80% duty. I switched between these on/offs fast enough that it is hard to tell anyone is off for a period of time. I repeated this process for turning in the other direction, just switching while Com got more speed. With each turn mode, I set that sides led to blink, simulating a directional. I also made the headlight led flash when the motors where in reverse. Looking ahead to possible crashes, I wrote the middle button of the directional pad to shut down all motor and led systems to save it at the last minute.

Calculations:

1. Time to overflow =top(n/fcpu)
    a. Top=2^16 = 65535
    b. N=1 and fcpu=16000000Hz
    c. T overflow=4.1ms
2. ICR value of “start bit”
    a. ICR=16000000*time
    b. ICR2=16000000*.0024
    c. IRC2=38,400
3. ICR value of “1”
    a. ICR=16000000*time
    b. ICR2=16000000*.0012
    c. IRC2=19200
4. ICR value of “0”
    a. ICR=16000000*time
    b. ICR2=16000000*.06

    c. IRC2=9600
5. Motor Fast PWM at 100Hz
    a. F=(fcpu/(N(top=1))
    b. Top=(fcpu/NF)-1
    c. N=1024 Fcpu=16000000Hz
    d. Top=155
    e. F=100.16Hz =OK for my use
6. Binary to Decimal
    a. Normal binary to decimal calculations

Program Code

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <inttypes.h>
#include <MSOE/delay.c>
#include <MSOE/bit.c>
#include <MSOE/lcd.c>
//NOTE: math library also added//

//GLOBAL VARRIBLES
int check=0;
uint16_t bits[13];
int i=0;
int count=0;

int main (void )
{
	//SETTING UP T/C 2 FOR MOTORS
	DDRC|=(1<<PORTC0)|(1<<PORTC1);//motor control bit
	DDRC|=(1<<PORTC2)|(1<<PORTC3);//motor control bit
	DDRD|=(1<<PORTD3);//pwm output digital 3
	DDRB|=(1<<PORTB3); //digital 11
	TCCR2A=(1<<COM2B1)|(1<<COM2A1)|(1<<WGM21)|(1<<WGM20);//fast pwm mode and
clear on compare
	TCCR2B=(1<<CS22)|(1<<CS21)|(1<<CS20)|(1<<WGM22);//prescale 1024

	//SETTING UP T/C 1 FOR IC
	DDRB|=(~(1<<PORTB0));//input channel
	TCCR1A=(1<<COM1A1);//ctc mode and clear on compare
	TCCR1B=(1<<WGM12)|(1<<CS10);//Prescale 1
	OCR1A=65535;//top   falling edge
	TIMSK1|=(1<<ICIE1);//input compare interrupt
	sei();//turning on global interrupts
	lcd_init();lcd_clear();lcd_home();//LCD enable
	int startbit=0;
	uint16_t result[13];
	int motorfwd(void);//function prototype
	int number=0;

	//LEDS
	DDRC|=(1<<PORTC4)|(1<<PORTC5); //analog 4 and 5
	DDRD|=(1<<PORTD2);//digital 2


	//function prototypes
	int allstop(void);
	int motorsfwd(void);
	int motorsback(void);
	int leftturn (void);
	int rightturn (void);

while(1)
{
	lcd_home();
	//START BIT CHECK//
```

```
if(bits[0]>38400)
{
	startbit=1;
	result[0]=0;//setting start bit
}
////TRANSFER SYSTEM TO BINARRY//
if(startbit==1)//is start bit is good
{
	for(int b=1; b<13; b++)
	{
		if(bits[b]>19000)
		{
			result[b]=1;
		}
		if(bits[b]<12000)
		{
			result[b]=0;
		}
	}
	//PRINTING BINARY TO LCD//
	for(int c=1; c<13; c++)
	{
		lcd_goto_xy(c-1,0);//shifting bits
		lcd_print_uint16(result[c]);//printing each bit
	}
}
//DEFAULT CASE//
else
{
	lcd_printf("PRESS ME");
}
//CONVERTING ARRAY TO DECIMAL//
if(count==1)
{
number=0;
if(result[0]==1){number=number+1;}
if(result[1]==1){number=number+2;}
if(result[2]==1){number=number+4;}
if(result[3]==1){number=number+8;}
if(result[4]==1){number=number+16;}
if(result[5]==1){number=number+32;}
if(result[6]==1){number=number+64;}
if(result[7]==1){number=number+128;}
if(result[8]==1){number=number+256;}
if(result[9]==1){number=number+512;}
if(result[10]==1){number=number+1024;}
if(result[11]==1){number=number+2048;}
if(result[12]==1){number=number+4096;}
count=0;
}
//PRINTING DECIMAL NUMBER TO LCD//
lcd_goto_xy(0,1);
lcd_print_uint16(number);

//BUTTON PRESSES TO LCD
lcd_goto_xy(10,1);
if(number==256){lcd_printf("1    ");}
```

```
	if(number==258){lcd_printf("2    ");}
	if(number==260){lcd_printf("3    ");}
	if(number==262){lcd_printf("4    ");}
	if(number==264){lcd_printf("5    ");}
	if(number==266){lcd_printf("6    ");}
	if(number==268){lcd_printf("7    ");}
	if(number==270){lcd_printf("8    ");}
	if(number==272){lcd_printf("9    ");}
	if(number==274){lcd_printf("0    ");}
	if(number==292){lcd_printf("vol+ ");}
	if(number==294){lcd_printf("vol-");}
	if(number==288){lcd_printf("Chan+");}
	if(number==290){lcd_printf("Chan-");}
	if(number==488){lcd_printf("up   ");}
	if(number==490){lcd_printf("down ");}
	if(number==360){lcd_printf("left ");}
	if(number==358){lcd_printf("right");}
	if(number==458){lcd_printf("STOP ");}

	//RUNNING THE MOTOR
	if(number==488)//forward
	{
		motorsfwd();
	}
	if(number==490)//backwards
		{
		motorsback();
		}
	if(number==458)//all stop
	{
		allstop();
	}
	if(number==358)//right
	{
		rightturn();
	}
	if(number==360)//left
	{
		leftturn();
	}
}//while
}//main

//INPUT CAPTURE INTERRUPT//
ISR(TIMER1_CAPT_vect)
{
		if(check==0)
		{
			TCNT1=0;//setting tcnt back down
			TCCR1B|=(1<<ICES1); //raising edge
			check=1;
		}
		else if(check==1)
		{
			bits[i]=ICR1;//storing in array
			TCCR1B&=(~(1<<ICES1));//falling edge
			check=0;
```

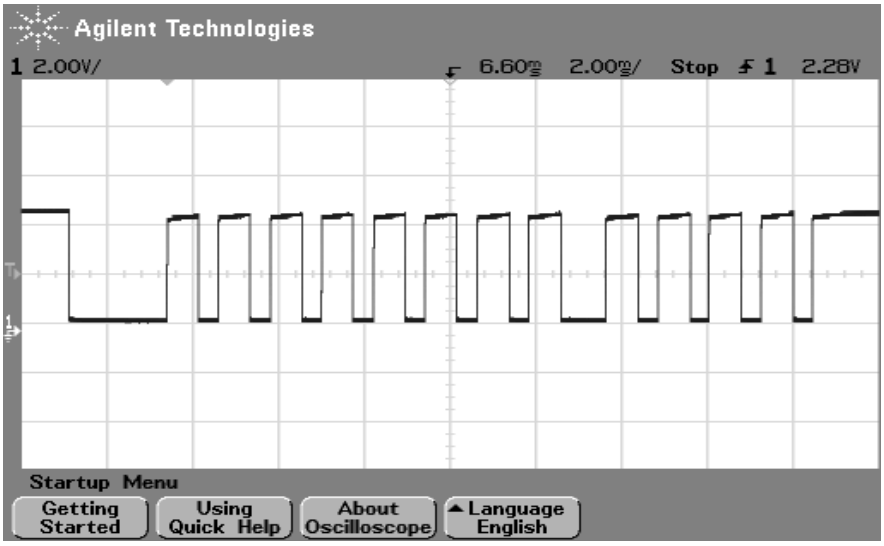
```
                  i++; //varrible for array
            }
            if(i==13)//refresh system
            {
                  i=0;
                  count=1;
            }
}

//MOTOR FUCTIONS//
int allstop(void)
{
      PORTC=0b00000000;
      PORTD=(~(1<<PORTD2));
      return 0;
}
int motorsfwd(void)
{
      TCCR2A|=(1<<COM2B1)|(1<<COM2A1);
      PORTC|=(1<<PORTC1)|(1<<PORTC2);//"in"s motor forward
      PORTC&=(~(1<<PORTC0))&(~(1<<PORTC3));//"in"s
      PORTC|=(1<<PORTC4)|(1<<PORTC5);
      PORTD|=(1<<PORTD2);//digital 2.
      delay_ms(250);
      PORTC&=(~(1<<PORTC4))&(~(1<<PORTC5));
      for(int a=0; a<155; a=a+31)
      {
            OCR2B=a;
            OCR2A=a;
            delay_ms(10);
      }
      return 0;
}
int motorsback(void)
{
      PORTD|=(1<<PORTD2);
      delay_ms(500);
      PORTD&=(~(1<<PORTD2));
      TCCR2A|=(1<<COM2B1)|(1<<COM2A1);
      PORTC|=(1<<PORTC0)|(1<<PORTC3);//"in"s motor forward
      PORTC&=(~(1<<PORTC1))&(~(1<<PORTC2));//"in"s
      for(int f=0; f<155; f=f+31)
      {
            OCR2B=f;
            OCR2A=f;
            delay_ms(10);
      }
      return 0;
}
int leftturn (void)
{
      TCCR2A&=(~(1<<COM2A1));
      TCCR2A|=(1<<COM2B1);
      PORTC|=(1<<PORTC1)|(1<<PORTC2);//"in"s motor forward
      PORTC&=(~(1<<PORTC0))&(~(1<<PORTC3));//"in"s
      for(int h=0; h<155; h=h+31)
      {
```

```
            OCR2B=155;
            OCR2A=h;
            PORTC|=(1<<PORTC4);
            delay_ms(10);
            PORTD|=(1<<PORTD2);
      }
      TCCR2A&=(~(1<<COM2B1));
      TCCR2A|=(1<<COM2A1);
      for(int g=0; g<93; g=g+31)
      {
            OCR2A=155;
            OCR2B=g;
            PORTC&=(~(1<<PORTC4));
            delay_ms(10);
      }
      return 0;
}
int rightturn (void)
{
      TCCR2A&=(~(1<<COM2B1));
      TCCR2A|=(1<<COM2A1);
      PORTC|=(1<<PORTC1)|(1<<PORTC2);//"in"s motor forward
      PORTC&=(~(1<<PORTC0))&(~(1<<PORTC3));//"in"s
      for(int i=0; i<93; i=i+31)
      {
            OCR2A=155;
            OCR2B=i;
            PORTC|=(1<<PORTC5);
            delay_ms(10);
            PORTD|=(1<<PORTD2);
      }
      TCCR2A&=(~(1<<COM2A1));
      TCCR2A|=(1<<COM2B1);
      for(int j=0; j<155; j=j+31)
      {
            OCR2A=j;
            OCR2B=155;
            PORTC&=(~(1<<PORTC5));
            delay_ms(10);
      }
      return 0;
}
```

Discussion

Example of Sony IR Signal (note signal is inverted) FIGURE 4



After testing my program, I noticed a few interesting things. First of all, if my system didn't receive a full signal, I would freeze up and spin in a circle until I pressed the reset button. Also, a few times my motors would act a little strange and ignore a button command. Near the start of this project, I had some random values in an array, so I zeroed it out and that seemed to do the trick. However, later down the road I took this zeroing out and it worked just fine.

Conclusion

Overall, this project was designed to figure out how to read an IR signal, but I went above and beyond this by making my own movable robot. I was able to successfully read and decode a Sony remote signal and then use my knowledge of the course material to design my own part to this lab. This project taught me how to decode an IR remote signal using the input capture unit and how to use DC motors. Even though this lab took many hours to complete, it was very enjoyable and I like programming even more due to it.