CG2271 Real Time Operating Systems

Tutorial 3

You are only allowed to write bare-metal programs for this tutorial.

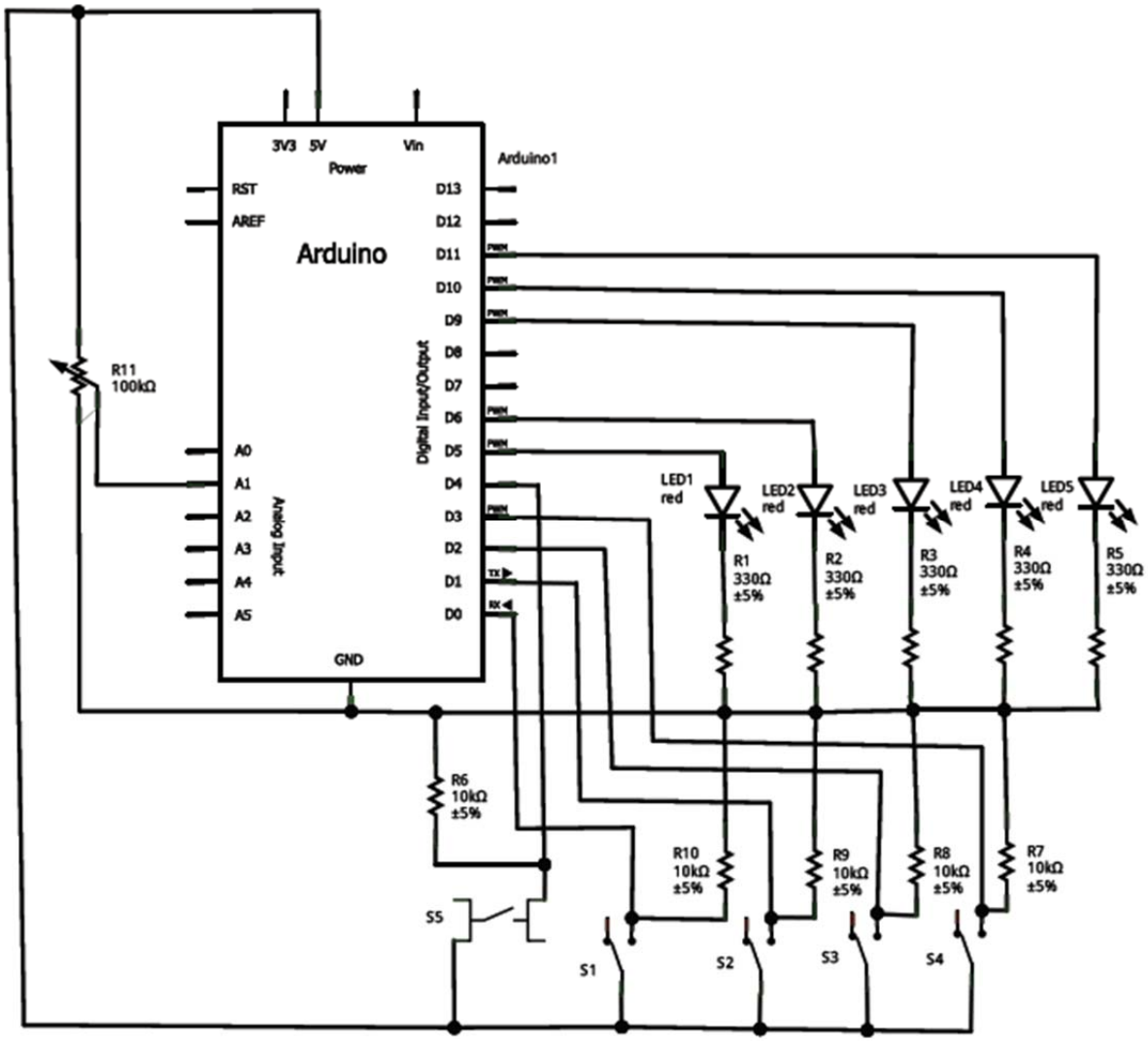
Question 1

We are all familiar with the binary number system, and in this question we will use the Arduino Uno has a (very expensive) binary to binary-coded-decimal (BCD) converter. The mapping between binary bits b3-b0 and BCD B4-B0 is shown in the table below:

Dec	b3	b2	b1	b0		B4	B3	B2	B1	B0
0	0	0	0	0		0	0	0	0	0
1	0	0	0	1		0	0	0	0	1
2	0	0	1	0		0	0	0	1	0
3	0	0	1	1		0	0	0	1	1
4	0	1	0	0		0	0	1	0	0
5	0	1	0	1		0	0	1	0	1
6	0	1	1	0		0	0	1	1	0
7	0	1	1	1		0	0	1	1	1
8	1	0	0	0		0	1	0	0	0
9	1	0	0	1		0	1	0	0	1
10	1	0	1	0		1	0	0	0	0
11	1	0	1	1		1	0	0	0	1
12	1	1	0	0		1	0	0	1	0
13	1	1	0	1		1	0	0	1	1
14	1	1	1	0		1	0	1	0	0
15	1	1	1	1		1	0	1	0	1

a. Design a circuit for the Arduino Uno with four switches (not push-buttons) connected to digital pins 0, 1, 2 and 3, and five LEDs connected to digital pins 5, 6, 9, 10 and 11. In addition there is a push-button switch connected to digital pin 4. You can use Fritzing (http://fritzing.org) to draw your circuits, in which case use the push-button for the switch at pin 4, and the toggle switches for pins 0-3.

Circuit for Q1 AND Q2:



b. Write a program that, when the push-button is depressed, causes the Uno to read in the binary patterns on the four switches, and outputs the BCD equivalent on the LEDs. There is no need to do debouncing for any of the switches.

```
#include <avr/io.h>
// Note: The following is needed for _delay_ms to work properly. This is the system
// clock frequency. Must be declared BEFORE including delay.h

#define F_CPU   16000000
#include <util/delay.h>

// Reads in the inputs and returns as an integer. Assumes pin3=MSB.
// Arduino pins 3-0 = PD3-PD0
int read_input()
{
        return PIND & 0xF;
}

// Converts from binary to BCD. Essentially checks if in is more than 9 (1001). If so
// it sets MSB to 1, and subtracts 10 from the number. So for example:
// 0b1010 (decimal 10) gives you msb=1 and out=0 (0b1 0000), 0b1011 (decimal 13)
// gives you msb=1 and out=3 (0b1 0011) etc.
void convert(int in, int *msb_on, int *out)
{
        if(in>9)
        {
                *msb_on=1;
                *out=in-10;
        }
        else
        {
                *msb_on=0;
                *out=in;
        }
}
```

```
// Outputs the BCD. Assumed that MSB is at digital output 11 (D11). So our BCD sequence
// is read (from left to right) D11 D10 D9 D6 D5, with D5 being LSB.
// The pins are D11 (PB3) D10 (PB2) D9(PB1) D6(PD6) and D5(PD5).
// Yes I know it's a royal PITA to keep looking up that table. Sorry folks.
// B4 = MSB from convert, B3toB0=out from convert
void output(int B4, int B3toB0)
{
        // Switch off LEDs at PD6 and PD5.
        PORTD &=0b10011111;

        // Switch off LEDs at PB3 PB2 PB1
        PORTB &=0b11110001;

        // LSB is at PD6 and PD5. We can just mask off all the rest of the bits
        // in B3toB0 then shift left by 6 bits.
        PORTD |= ((B3toB0 & 0b11) << 6);

        // Now we mask off B1 and B0, then shift right by 1 bit to light PB2 and PB1.
        // So if B3toB2 is 1011, then &0b1100 gives us 1000. If we don't shift right
        // We will end up lighting up PB3 and PB2 instead.
        PORTB |= ((B3toB2 & 0b1100) >> 1);

        // Finally light up PB3 with B4.
        PORTB |= (B4 << 3);
}

int main()
{
        // Remember that we have a push-button at D4, which is PD4.
        // Initialize PD4-0 as input and PD6-5 as output
        DDRD &= 0b1110 0000;
        DDRD |=  0b0110 0000;

        int B4=0, B3toB0=0, input;

        input=read_input();
        convert(input, &B4, &B3toB0);
```

```
        while(1)
        {
                output(B4, B3toB0);

                // Test push button
                if(PIND & 0b0001 0000)
                {
                        input=read_input();
                        convert(input, &B4, &B3toB0);
                }
        }
}
```

Question 2

a. Modify your circuit from Q1 by adding a potentiometer to analog pin 1.

   See Q1a for answer.

b. Add in code for your program in Q1 to read in the potentiometer, using interrupt driven ADC. Store the read value in a variable called "adc_in". The potentiometer is read every time the push-button at pin 4 is pressed.

```
#include <avr/io.h>
// Note: The following is needed for _delay_ms to work properly. This is the system
// clock frequency. Must be declared BEFORE including delay.h

#define F_CPU   16000000
#include <util/delay.h>
#include <avr/interrupt.h>

int loval, highval, adc_in;
// Declare the ISR
ISR(ADC_vect)
{
        loval=ADCL;
        hival=ADCH;
        adc_in=(hival << 8) + ADCL;

        // Re-start conversion by setting ADSC (bit 6) of ADCSRA to 1.
        ADCSRA |= 0b0100 0000;
}

// Init ADC. We will use a sampling frequency of 125 kHz, giving us a prescalar of 128.
```

void init_adc()
{
PRR &=0b11111110; // Power on ADC
// ADCSRA set to:

ADEN	ADSC	ADATE	ADIF	ADIE	ADSP2	ADSP1	ADSP0
1	0	0	0	1	1	1	1

ADCSRA=0b10001111;

// Set up ADMUX so that we use VCC (REFS1-0 = 01) and convert channel 1 (001)

REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0
0	1	0	0	0	0	0	1

ADMUX=0b01000001;
}

```
// Reads in the inputs and returns as an integer. Assumes pin3=MSB.
// Arduino pins 0-3 = PD0-PD3
int read_input()
{
	return PIND & 0xF;
}

// Converts from binary to BCD. Essentially checks if in is more than 9 (1001). If so
// it sets MSB to 1, and subtracts 10 from the number. So for example:
// 0b1010 (decimal 10) gives you msb=1 and out=0 (0b1 0000), 0b1011 (decimal 13)
// gives you msb=1 and out=3 (0b1 0011) etc.
void convert(int in, int *msb_on, int *out)
{
	if(in>9)
	{
		*msb_on=1;
		*out=in-10;
	}
	else
	{
		*msb_on=0;
		*out=in;
	}
}
```

```
// Outputs the BCD. Assumed that MSB is at digital output 11 (D11). So our BCD sequence
// is read (from left to right) D11 D10 D9 D6 D5, with D5 being LSB.
// The pins are D11 (PB3) D10 (PB2) D9(PB1) D6(PD6) and D5(PD5).
// Yes I know it's a royal PITA to keep looking up that table. Sorry folks.
// B4 = MSB from convert, B3toB0=out from convert
void output(int B4, int B3toB0)
{
        // Switch off LEDs at PD6 and PD5.
        PORTD &=0b10011111;

        // Switch off LEDs at PB3 PB2 PB1
        PORTB &=0b11110001;

        // LSB is at PD6 and PD5. We can just mask off all the rest of the bits
        // in B3toB0 then shift left by 6 bits.
        PORTD |= ((B3toB0 & 0b11) << 6);

        // Now we mask off B1 and B0, then shift right by 1 bit to light PB2 and PB1.
        // So if B3toB2 is 1011, then &0b1100 gives us 1000. If we don't shift right
        // We will end up lighting up PB3 and PB2 instead.

        PORTB |= ((B3toB2 & 0b1100) >> 1);
        // Finally light up PB3 with B4.
        PORTB |= (B4 << 3);
}

int main()
{
        // Remember that we have a push-button at D4, which is PD4.
        // Initialize PD0-4 as input and PD5-6 as output
        DDRD &= 0b1110 0000;
        DDRD |=  0b0110 0000;

        // Initialize the ADC
        init_adc();

        int B4=0, B3toB0=0, input;

        input=read_input();
        convert(input, &B4, &B3toB0);

        // Start converting the pot.
        ADCSRA |= 0b01000000;
        sei();
```

```
        while(1)
        {
                output(B4, B3toB0);

                // Test push button
                if(PIND & 0b0001 0000)
                {
                        input=read_input();
                        convert(input, &B4, &B3toB0);
                }
        }
}
```

The variable adc_in will be used in the next tutorial to control the brightness of the LEDs.

Question 3

Modify your program in Q2 so that the program reads the potentiometer at analog input 1 and the switches at pins 0 to 3 every 5 ms instead of each time the push-button is pressed.

NOTE TO TUTORS: More credit is given if you set the ISR to go off every 5 ms instead of every 1 ms and counting to 5. Please inform the students of this.

We need to figure out the prescalar P and count value V. We use the following table, assuming a 16 MHz clock.

Prescalar	Resolution	V (5000 us)
1	0.0625 us	80000
8	0.5 us	10000
64	4 us	1250
256	16 us	312.5
1024	64 us	78.125

Only usable values are P=1024 and V=78. This gives us an interrupt every 4.99 ms.

```
#include <avr/io.h>
// Note: The following is needed for _delay_ms to work properly. This is the system
// clock frequency. Must be declared BEFORE including delay.h

#define F_CPU   16000000
#include <util/delay.h>
#include <avr/interrupt.h>

int loval, highval, adc_in;

// Prototype for read_input and convert
int read_input();
void convert(int, int*, int*);

// input, B4, B3toB0 are all now global.
int input=0, B4=0, B3toB0=0;

// Declare the Timer ISR. Read in the input and convert.
ISR(TIMER0_COMPA_vect)
{
        input=read_input();
        convert(input, &B4, &B3toB0);
}
// Set up the timer
void init_timer()
{
        // Initialze TCNT0
        TCNT0=0;
        OCR0A=78;

        // Disconnect OC0A and set CTC in TCCR0A
```

COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00
0	0	0	0	0	0	1	0

```
        TCCR0A=0b00000010;

        // Enable the output compare interrupt OCIE0A in TIMSK0
```

-	-	-	-	-	OCIE0B	OCIE0A	TOIE0
0	0	0	0	0	0	1	0

```
        TIMSK0|=0b10;
}


// Declare the ISR
```

```
ISR(ADC_vect)
{
        loval=ADCL;
        hival=ADCH;
        adc_in=(hival << 8) + ADCL;

        // Re-start conversion by setting ADSC (bit 6) of ADCSRA to 1.
        ADCSRA |= 0b0100 0000;
}

// Init ADC. We will use a sampling frequency of 125 kHz, giving us a prescalar of 128.
void init_adc()
{
        PRR &=0b11111110; // Power on ADC
        // ADCSRA set to:
```

ADEN	ADSC	ADATE	ADIF	ADIE	ADSP2	ADSP1	ADSP0
1	0	0	0	1	1	1	1

```
        ADCSRA=0b10001111;
        // Set up ADMUX so that we use VCC (REFS1-0 = 01) and convert channel 1 (001)
```

REFS1	REFS0	ADLAR	-	MUX3	MUX2	MUX1	MUX0
0	1	0	0	0	0	0	1

```
        ADMUX=0b01000001;
}
// Reads in the inputs and returns as an integer. Assumes pin3=MSB.
// Arduino pins 0-3 = PD0-PD3
int read_input()
{
        return PIND & 0xF;
}
```

```
// Converts from binary to BCD. Essentially checks if in is more than 9 (1001). If so
// it sets MSB to 1, and subtracts 10 from the number. So for example:
// 0b1010 (decimal 10) gives you msb=1 and out=0 (0b1 0000), 0b1011 (decimal 13)
// gives you msb=1 and out=3 (0b1 0011) etc.
void convert(int in, int *msb_on, int *out)
{
	if(in>9)
	{
		*msb_on=1;
		*out=in-10;
	}
	else
	{
		*msb_on=0;
		*out=in;
	}
}

// Outputs the BCD. Assumed that MSB is at digital output 11 (D11). So our BCD sequence
// is read (from left to right) D11 D10 D9 D6 D5, with D5 being LSB.
// The pins are D11 (PB3) D10 (PB2) D9(PB1) D6(PD6) and D5(PD5).
// Yes I know it's a royal PITA to keep looking up that table. Sorry folks.
// B4 = MSB from convert, B3toB0=out from convert
void output(int B4, int B3toB0)
{
	// LSB is at PD6 and PD5. We can just mask off all the rest of the bits
	// in B3toB0 then shift left by 6 bits.
	PORTD |= ((B3toB0 & 0b11) << 6);

	// Now we mask off B1 and B0, then shift right by 1 bit to light PB2 and PB1.
	// So if B3toB2 is 1011, then &0b1100 gives us 1000. If we don't shift right
	// We will end up lighting up PB3 and PB2 instead.
	PORTB |= ((B3toB2 & 0b1100) >> 1);

	// Finally light up PB3 with B4.
	PORTB |= (B4 << 3);
}
```

```
int main()
{
        // We no longer use PD4 for the push-button so don't set its direction.
        // Initialize PD0-3 as input and PD5-6 as output
        DDRD &= 0b1111 0000;
        DDRD |=  0b0110 0000;

        // Initialize the ADC
        init_adc();
        init_timer();

        // Note: input, B4, B3toB0 all moved to global. We do the FIRST conversion only.
        input=read_input();
        convert(input, &B4, &B3toB0);

        sei();

        // Start the timer, with prescalar of 128 (0b101) in TCCR0B
```

FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00
0	0	0	0	0	1	0	1

```
        TCCR0B=0b00000101;

        // Start converting the pot.
        ADCSRA |= 0b01000000;
        // Conversion is now done by the timer. We just output the converted value.
        while(1)
          output(B4, B3toB0);

}
```

Question 4

Supposed we are given a system where the PWM encoding is to be read from/written to a port with the following specification:

Port Data Rate:	512 bps
PWM Encoding Lenth:	16 bits

a. What is the length of the PWM cycle, in milliseconds?

16/512 = 0.03125 seconds. =31.25 millseconds.

b. Given that the analog voltage range is ±5v, what is the resolution of the A/D and D/A converters, in volts per bit?

There is a 10-volt rage from -5v to 5v, so the resolution is 10/16 = 0.625 volts.

c. Suggest the encoding for -5v, 0v and 5v. Give the encoding for -3.5v, 2.2v, 4.5v. Is it possible to get exact encodings for these voltages? Why or why not?

0000 0000 0000 0000 = -5V 0000 0000 1111 1111 = 0 V 1111 1111 1111 1111 = 5V

Each bit represents 0.625 volts. -3.5v = (-5v + 1.5v) # of 1 bits = 1.5/0.625 □ 3 bits So encoding ofr -3.5v is 0000 0000 0000 0111 (Actual value is -3.125v)

2.2v = (-5v + 7.2v) # of 1 bits =7.2/0.625 □ 12 bits So encoding is 0000 1111 1111 1111 (actual value is 2.5v)

4.5v = -5v+9.5v # of 1 bits = 9.5/0.625 □ 16 bits So encoding is 1111 1111 1111 1111 (actual value is 5v)

Note that the encodings are not exact. This is normal.