EE/CS 120B: Introduction to Embedded Systems
University of California, Riverside
Winter 2016

Laboratory Exercise 6

This laboratory exercise will introduce the concepts of Analog-to-Digital Conversion (ADC) and Pulse-Width Modulation (PWM) and how they can be used and implemented on a microcontroller.

Introduction to Analog-to-Digital Conversion (ADC)

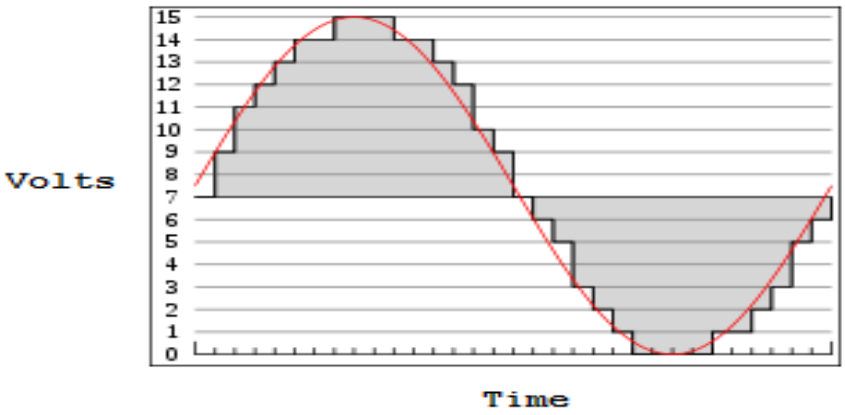
A programmer often wants to read input from the environment, such as sound from a microphone, levels from a light sensor, temperature from a thermometer, etc. Sensors often convert environmental input into an analog signal output. The amplitude, and perhaps frequency, of the analog signal output usually indicates the intensity of the sensory input.

A microcontroller can react to stimulus detected by the sensor by interpreting the analog output signal's amplitude or frequency. For example, a microphone that converts sound waves into an analog signal can be connected to a microcontroller that enables a light if the sound is too loud (perhaps as an anti-burglar device).

An analog signal must first be converted into a digital value before a microcontroller can use it. Most microcontrollers provide analog-to-digital converters (ADCs) for this purpose. In this lab, you will utilize the ADC on a microcontroller to interface with a light sensor.

How ADC Works

Consider a sine wave analog signal like the red line in the image below. This particular sine wave has a DC bias of 7.5 Volts and an amplitude of +/- 7.5 Volts, creating a signal that oscillates between 0 and 15 Volts. An analog-to-digital converter **discretizes** the analog signal, yielding a digital number that represents the amplitude of the signal. The image below shows how the size wave might be discretized into 16 levels using a 4-bit ADC. Note that the discretization of the signal results in 'steps' being formed when the signal is traced, (an effect usually called **aliasing**).



The ADC on the ATmega1284 is 10 bits, thus offering 1024 individual possible levels that an analog signal can be represented as a digital value. The ATmega1284 uses the following equation to calculate the converted digital value of the signal amplitude, which is stored in the register ADC:

$$\mathbf{ADC = (V_{IN} * 1024) / V_{REF}}$$

- The input voltage V_{IN} is the analog signal from a sensor.
- The reference voltage V_{REF} is used to scale the result.
- Higher values of ADC correspond to voltage levels closer to V_{REF}.

The result of the conversion, stored in ADC, can be read at any time by the C program. A programmer simply writes code like:

```
unsigned short x = ADC;  // Value of ADC register now stored in variable x.
```

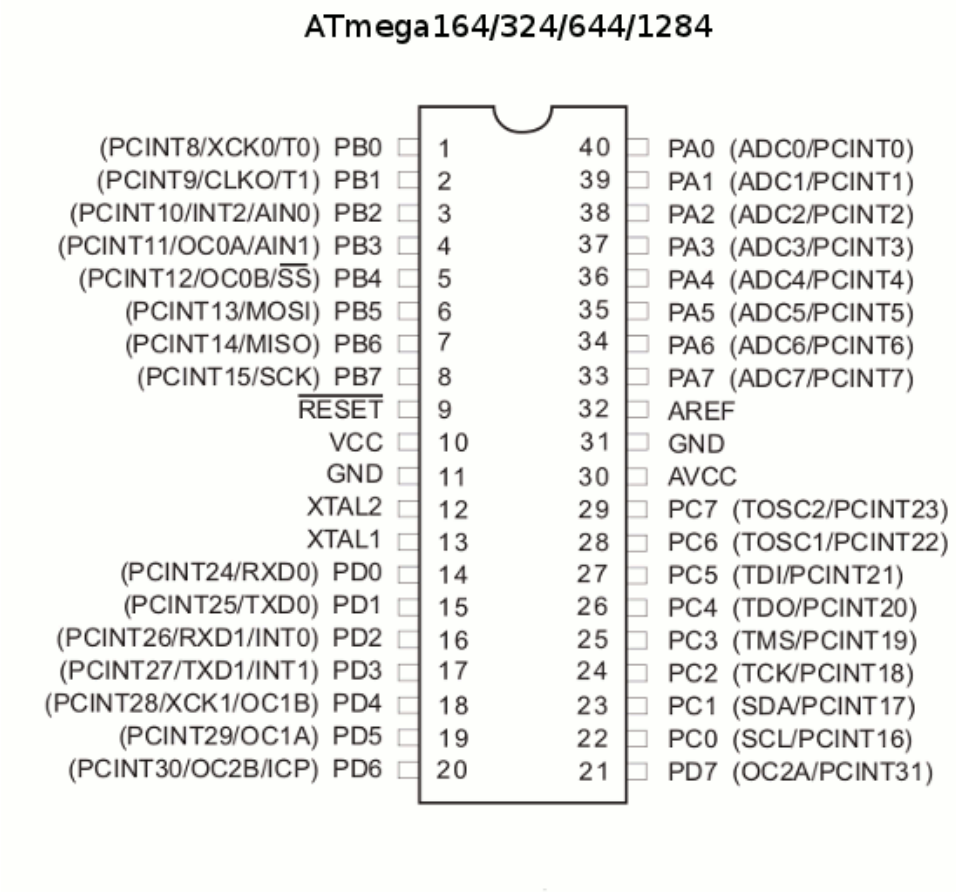
Initialization

ADC will only work with the ATmega1284 if specific flags are set within the microcontroller. The function below sets those flags. Copy and paste the function to the start of your C program to initialize the A/D converter. Make sure to call the function within main, but before the while loop.

```
void ADC_init() {
ADCSRA |= (1 << ADEN) | (1 << ADSC) | (1 << ADATE);
   // ADEN:   setting this bit enables analog-to-digital conversion.
   // ADSC:   setting this bit starts the first conversion.
   // ADATE:  setting this bit enables auto-triggering. Since we are
   //         in Free Running Mode, a new conversion will trigger
   //         whenever the previous conversion completes.
}
```

After the above function is called, analog to digital conversion will work as follows.

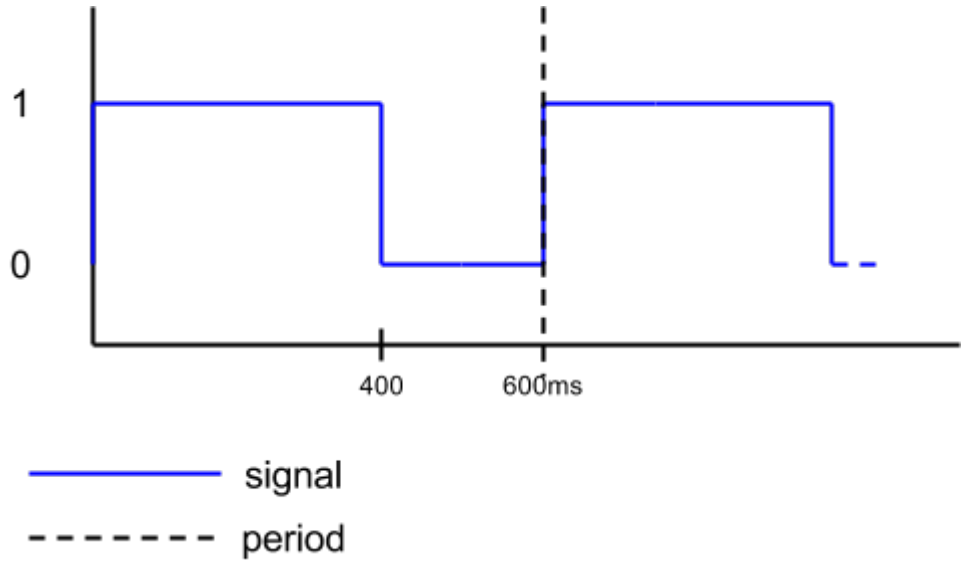
- V_{in} is connected to PA0.
- **IMPORTANT:** The AREF (V_{ref}) pin is connected directly to the +5 Volt power supply. AREF is the pin located between PA7 and the ground pin. A schematic of the ATmega1284 is given below.



Introduction to Pulse-Width Modulation (PWM)

A pulse width modulator (PWM) is a programmable component that generates pulses to achieve a specified period and duty cycle. Such pulses have many uses, such as generating sounds of a particular frequency, controlling a motor with a specific average input voltage, communicating information, and more. Some key PWM definitions are:

- **Period (T):** The time for one cycle of a repeating pulse pattern. The signal below has a period of 600 ms.
- **Duty cycle:** The percentage of time that a signal is high during the period. The signal below has a duty cycle of 66.6% since the signal is high for 400 ms of the 600 ms period, and 400/600 = 66.6%



This lab uses pulse width modulation to create sounds of various frequencies.

How does the period influence sound from a speaker?

Sound is generated by pulsing a speaker's input voltage at a particular frequency: the higher the frequency, the higher the sound. The range of an average adult's hearing is between 20 Hz and 16,000 Hz. Hz is a measure of frequency, indicating the number of pulses per second. Frequency is the inverse of period: ***frequency = 1/period***.

How does a speaker work?

A speaker contains a magnetic component called a "diaphragm". When positive voltage is supplied to the speaker, the diaphragm is pulled inwards. When no voltage is supplied, the diaphragm is released and moves outwards. Pulsing the voltage causes the diaphragm to pulse at the same frequency, causing sound waves to be created at that frequency. A 50% duty cycle means voltage is supplied just as often as not, minimizing the delay between the signal's high and low values, maximizing the number of pulses per period. A 50% duty cycle produces the loudest possible tone. If the duty cycle is higher or lower, then less time is spent pulsing, resulting in a quieter tone.

Musical sounds

The frequencies of middle notes of a piano are listed below. Based off of these frequencies, the period can be calculated by inverting the frequency. The following frequencies, and calculated periods, will be used throughout the lab.

Musical Note	Frequency (Hz)
C_4	261.63
D_4	293.66
E_4	329.63
F_4	349.23
G_4	392.00
A_4	440.00
B_4	493.88
C_5	523.25

ATmega1284's PWM functionality

In previous labs, writing a synchSM that could pulse at different frequencies could get extensive and complicated really quick. These synchSMs can be greatly simplified by using the ATmega1284's built in PWM functionality.

The ATmega has multiple methods of implementing PWM. The method used in this lab uses a timer and a counter. A variable is set to a desired 16-bit value. The counter counts up to the value of the variable. When the counter equals the value of the variable, an interrupt flag is set, and the pin outputting the PWM signal is toggled.

The functions below set up TCCR0A, TCCR0B and OCR0A for the timer/counter 3. Copy and paste this code to the top of the project. For more information on what these function do, refer to page 140 of the ATmega1284's datasheet.

```
//    0.954 hz is lowest frequency possible with this
//    function, based on settings in PWM_on(). Passing 0 as
//    the frequency stop the speaker from generating sound
void set_PWM(double frequency) {

      // Keeps track of the currently set frequency
      // Will only update the registers when the frequency
      // changes, plays music uninterrupted.
      static double current_frequency;

      if (frequency != current_frequency) {

            //stops timer/counter
            if (!frequency) { TCCR0B &= 0x08; }
            // resumes/continues timer/counter
            else { TCCR0B |= 0x03; }

            // prevents OCR3A from overflowing, using
            // prescaler 64; 0.954 is smallest frequency that
            // will not result in overflow
            if (frequency < 0.954) { OCR0A = 0xFFFF; }

            // prevents OCR0A from underflowing, using
            // prescaler 64; 31250 is largest frequency that
            // will not result in underflow
            else if (frequency > 31250) { OCR0A = 0x0000; }

            // set OCR3A based on desired frequency
            else { OCR0A = (short)(8000000 /
                           (128 * frequency)) - 1; }

            // reset counter and update the current frequency
            TCNT0 = 0;
            current_frequency = frequency;
      }
}

void PWM_on() {
      TCCR0A = (1 << COM0A0);
      // COM3A0: Toggle PB3 on compare match between counter
      // and OCR0A
      TCCR0B = (1 << WGM02) | (1 << CS01) | (1 << CS00);
      // WGM02: When counter (TCNT0) matches OCR0A, reset
      // counter
      // CS01 & CS30: Set a prescaler of 64
      set_PWM(0);
}
```

```
void PWM_off() {
      TCCR0A = 0x00;
      TCCR0B = 0x00;
}
```

NOTE: Removing the "static double current_frequency" and the "if (current_frequency != frequency)" guard will update the frequency every time it is called, even if the frequency passed in is the same as the last time set_PWM() was called.

Function Descriptions:

PWM_on(): Enables the ATmega1284's PWM functionality.

**PWM_off(): ** Disables the ATmega1284's PWM functionality.

**set_PWM(double frequency): ** Sets the frequency output on OC0A (Output Compare pin). OC0A is pin PB3 on your microcontroller. The function uses the passed in frequency to determine what the value of OCR0A should be so the correct frequency will be output on PB3. The equation below shows how that output frequency is calculated, and how the equation in set_PWM was derived.

$$f_{OC2} = f_{Clock} / (2 * \text{prescaler} * (OCR0A + 1))$$

IMPORTANT: PB3 MUST BE USED AS YOUR SPEAKER OUTPUT PIN because the PWM connects to that pin. Make sure to set PB3 as output (DDRB = xxxx 1xxx;).

NOTE: There are multiple pins on the ATMega1284 that can be used for PWM. If you are using a pin other PB3 you will need to set up a different timer. The ATMega 1284 datasheet can help with this endeavor.

***NOTE*:** If you use PB6 as the PWM pin, then in order to program your ATMega you will need to disconnect the speaker.

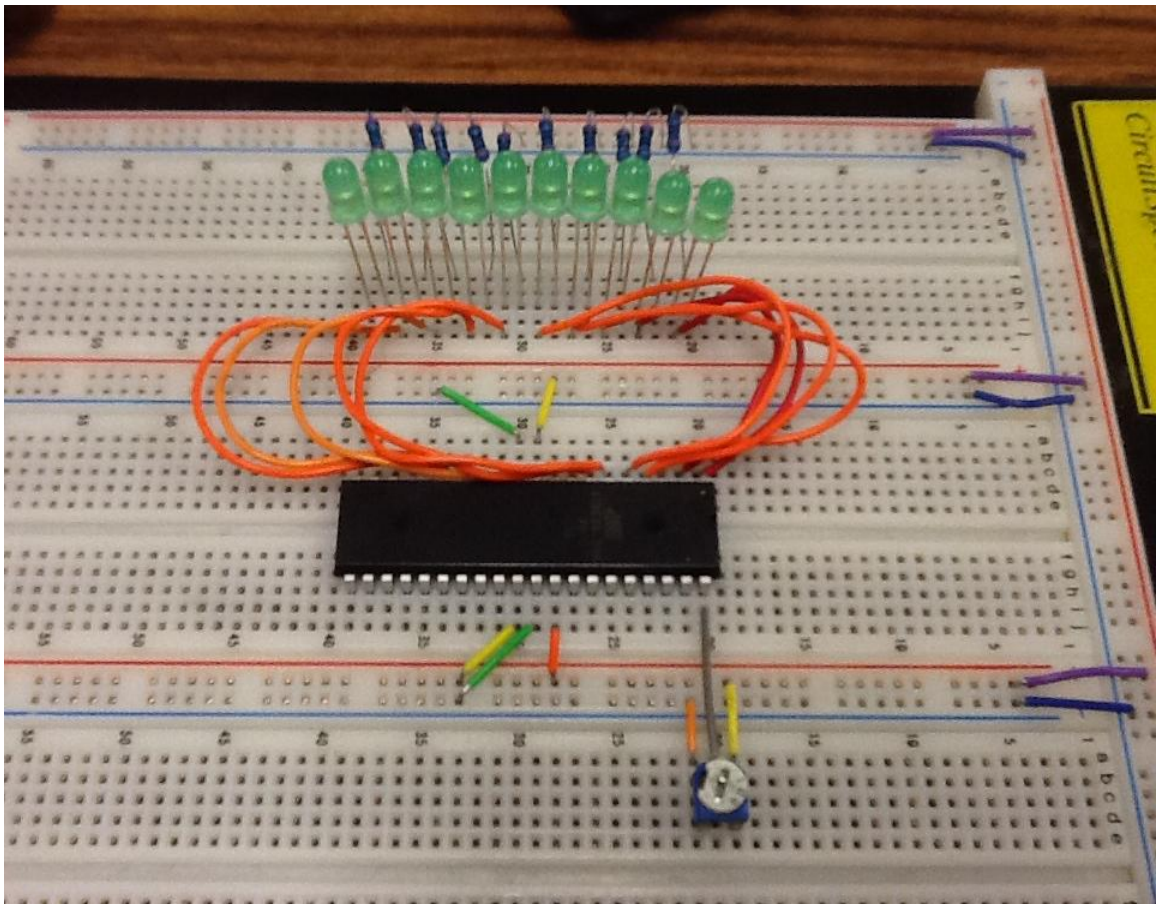
Prelab:

There are separate setups for the ADC and PWM portions of this laboratory exercise. Our recommendation is that one partner set up his/her breadboard for ADC, and the other for PWM.

Prelab (ADC):

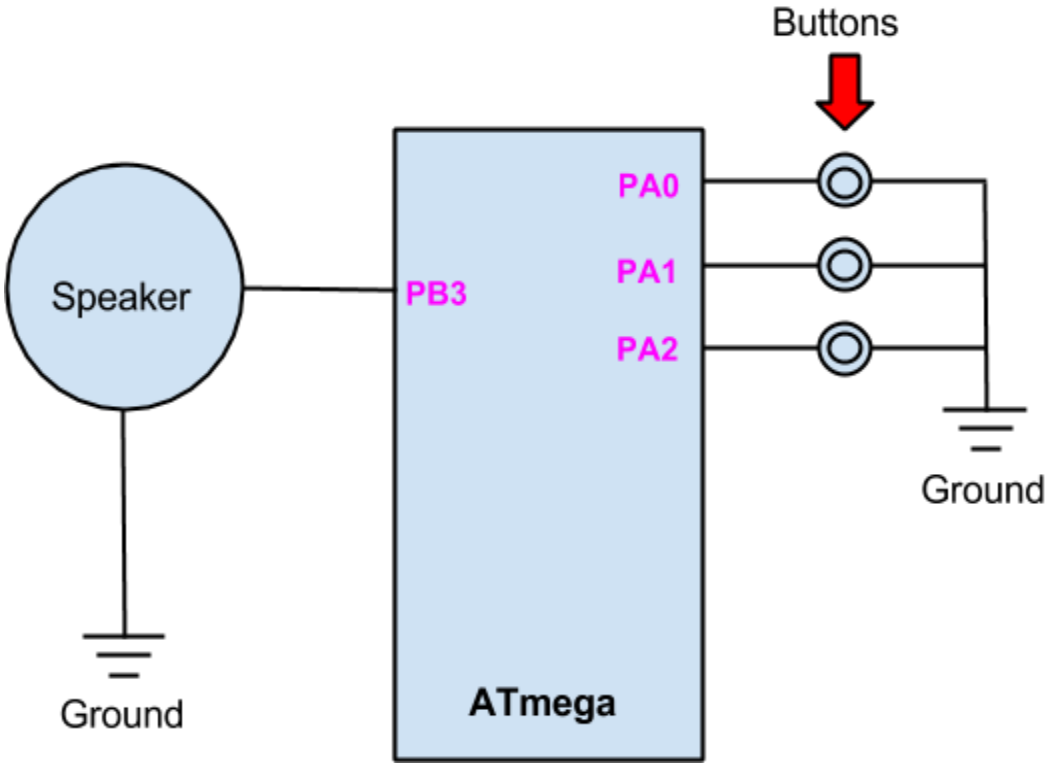
To support ADC, wire the board as shown below. The connections are as follows:

input: PA[0] connected to potentiometer
output: PB[7:0] connected to LEDs
PD[1:0] connected to LEDs



Prelab (PWM):

Come prepared with a board wired according to the image below. Placement of the buttons can vary, but the speaker must be connected to PB3.



ADC Part 1: Using the Potentiometer

Make sure your breadboard is wired according to the prelab. The potentiometer is used to adjust the voltage supplied to the microcontroller for ADC . Design a system that reads the 10-bit ADC result from the ADC register, and displays the result on a bank of 10 LEDs.

Hints:

- Assuming the breadboard has been wired according to the prelab photo, display the lower 8 bits on port B, and the upper 2 bits on port D.
- Use a "short" variable to hold the ADC result.
- Use bit shifting and casting to align the proper bits to the proper ports. For example:

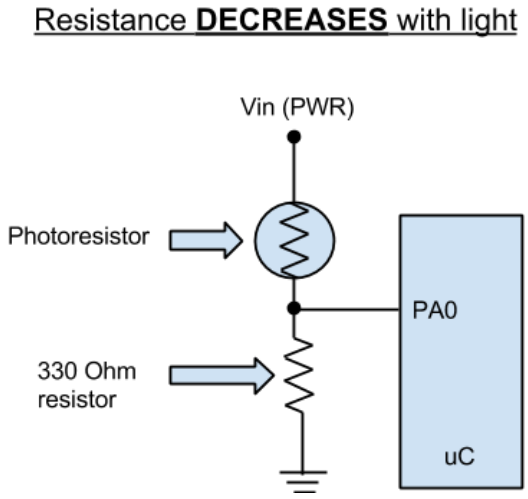
```
unsigned short my_short = 0xABCD;
unsigned char  my_char = (char)my_short; // my_char = 0xCD
my_char = (char)(my_short >> 4); // my_char = 0xBC
```

Video Demonstration: http://youtu.be/DB4fXVLT9r0

ADC Part 2: Using a Photoresistor

What is a Photoresistor?

A photoresistor is a resistor whose resistance varies based on how much light the photoresistor detects. An additional resistor needs to be connected in parallel with the photoresistor. Without the additional resistor, results from ADC will be too small to observe. A 330 ohm resistor is chosen because it is common in the lab kits. Connect the photoresistor to the microcontroller according to the diagram below.



NOTE: With the photoresistor connected in this way, more light equals higher ADC values.

Exercise: Revise Part 1 by replacing the potentiometer with a photoresistor and 330 ohm resistor. Take note of the highest ADC value displayed when the photoresistor is exposed to light, and the lowest ADC value displayed when the photoresistor is deprived of all light. These values will be used for the remaining lab exercises.

Video Demonstration: **http://youtu.be/Zx2ulW9UJfc**

ADC Part 3: Turning on/off an LED Using a Photoresistor

Design a system where an LED is illuminated if enough light is detected from the photo resistor.

Criteria:

- If the result of the ADC is >= MAX/2, the LED is illuminated.
- If the result of the ADC is < MAX/2, the LED is turned off.

Notes:

- MAX is the highest ADC value observed from part 2 of the lab.

Video Demonstration: **http://youtu.be/Hx0FWGBI6-g**

ADC Part 4: Light Meter (Challenge)

Design a system, using a bank of eight LEDs, where the number of LEDs illuminated is a representation of how much light is detected. For example, when more light is detected, more LEDs are illuminated.

Criteria:

- The LEDs should illuminated in sequence from 0 to 7, based on the amount of light detected by the photoresistor.

Hints:

- Use the maximum ADC value observed from part 2 as the highest amount of light detectable by the photoresistor. Divide that number by eight to determine the thresholds for each LED.

Video Demonstration: **http://youtu.be/n9ejT-PNJTI**

PWM Part 1: Tone Selector

Using the ATmega1284's PWM functionality, design a system that uses three buttons to select one of three tones to be generated on the speaker. When a button is pressed, the tone mapped to it is generated on the speaker.

Criteria:

- Use the tones C_4, D_4, and E_4 from the table in the introduction section.
- When a button is pressed and held, the tone mapped to it is generated on the speaker.
- When more than one button is pressed simultaneously, the speaker remains silent.
- When no buttons are pressed, the speaker remains silent.

Video Demonstration: http://youtu.be/_w4BmDvA9mw

PWM Part 2: 8-note Scale

Using the ATmega1284's PWM functionality, design a system where the notes: C_4, D, E, F, G, A, B, and C_5, from the table at the top of the lab, can be generated on the speaker by scaling up or down the eight note scale. Three buttons are used to control the system. One button toggles sound on/off. The other two buttons scale up, or down, the eight note scale.

Criteria:

- The system should scale up/down one note per button press.
- When scaling down, the system should not scale below a C.
- When scaling up, the system should not scale above a C.

Hint:

- Breaking the system into multiple synchSMs could make this part easier.

Video Demonstration: http://youtu.be/BjPZhS_gGzU

PWM Part 3: Compose a Short Melody (Challenge)

Using the ATmega1284's built in PWM functionality, design a system where a short, five-second melody, is played when a button is pressed. **NOTE:** The melody must be somewhat complex (scaling from C to B is NOT complex).

Criteria:

- When the button is pressed, the melody should play until completion
- Pressing the button again in the middle of the melody should do nothing
- If the button is pressed and held, when the melody finishes, it should not repeat until the button is released and pressed again

Hint:

- One approach is to use three arrays. One array holds the sequence of notes for the melody. Another array holds the times that each note is held. The final array holds the down times between adjacent notes.

Video Demonstration: http://youtu.be/4fBw0aR3nvQ

Each student must submit their .c source files according to instructions in the lab submission guidelines. Post any questions or problems you encounter to the wiki and discussion boards on iLearn.