Lab 6

- Use the PWM mode to the CCP module on the PICDEM DEM 2 board to drive the piezo buzzer
- Use the functionality developed in lab 5 (button logic)
- Line 1 output
  - Frequency in Hz (20Hz – 800Hz), columns 1 – 3
  - Oscillator setting in kHz (250kHz – 16,000kHz), columns 5 – 10
  - Duty cycle (0 – 100), columns 12 – 14
- Line 2 output
  - Value stored in PRx (0 – 255), columns 1 – 3
  - Value stored in CCPRxL (0 – 255), columns 5 – 7
  - Value stored in CCP1CONbits.DC1B (00 – 11), columns 9 – 10 (binary)

Lab 6

- These may help you debug
  - Turn RB3 on when the right most button is depress, off when it is not
  - Default is potentiometer varies the range of the PMW period
    - RB2 on
    - RB1 off
  - First push of right most button has the potentiometer vary the range of the PWM duty cycle period
    - RB2 off
    - RB1 on
  - Second push of right most button returns to default mode

PWM

- Wikipedia states that for humans, the audible range is 20Hz – 20kHz
- Most of us should be able to hear 20Hz – 800Hz (my informal survey)
- Sounds above 100Hz seem to be somewhat annoying

Performing Computations

- Duty cycle = min duty cycle + ((max duty cycle – min duty cycle)/1,023) x potentiometer value
  - Min duty cycle = 0
  - Max duty cycle = 100
  - Potentiometer range is [0, 1023]
  - Potentiometer value = current value of analog to digital conversion for the potentiometer
  - Pay attention to floating point versus integer math
  - I used double to do the computation

Performing Computations

- Frequency = min frequency + ((max frequency – min frequency)/1,023) x potentiometer value
  - Min frequency = 20Hz
  - Max frequency = 800Hz (100 for testing)
  - Potentiometer range is [0, 1023]
  - Potentiometer value = current value of analog to digital conversion for the potentiometer
  - Pay attention to floating point versus integer math
  - I used double to do the computation
- Period = 1/frequency (0 – 255, 8 bit register)

Pulse width

- The pulse width is the period x (duty cycle/100)
  - Since period is an 8 bit integer, pulse width will also be an 8 bit integer (8 MSBs)
- Since we have 10 bits for the pulse width
  - The 2 LSBs can be computed by
    - Subtract integer version of pulse width from floating point version
    - Divide result by 0.25 [0, 4)
    - Convert to int (truncate)

Clock Frequency and Pre Scale

- You can use Timer2, Timer4, or Timer6
  - I use Timer6
- A pre scale of 16 works for our entire range
- We want to find the largest clock frequency that supports the range
  - The timer uses the system clock, so reducing the frequency of the timer will also require reducing the frequency of the system clock
    - At 250kHz, the processor is running at 1/64 of it's maximum speed (1/32 of what we typically use)
- The system clock range of 250kHz – 8MHz supports the frequency range we are interested in

System Clock

- Remember that the system clock is used for our pause function
  - Uses by LCD code
- So, running at 250kHz will slow everything down
- I used the 600kHz dedicated clock for the A/D conversion
- I made some minor changes to my version of pause to account for clock speed changes
- Your main loop will also run slower
  - This is why we are using RB1, RB2, and RB3 to see the mode and the button push has been capture
    - I poll the button, interrupts might work better

Calculating Clock Speed

- The maximum clock period is
  - 256 x (1/Fosc) x pre scale x 4
- The period is
  - 1/frequency
- Find the largest clock speed (Fosc) such that
  - Maximum clock period >= period
- The pre scale can be fixed at 16
- Fosc in [250kHz, 8MHz]

Alternate PWM

- For low frequency PWM, such as the 20Hz frequency sound
  - We could
    - Use use the CCP module in compare mode
      - Can use 16 bit register and high frequency system clock
    - Simulate the Timer2/PR2 pair with the 16 bit CCP register and Timer1
    - Put the pulse width period in CCP register
    - Set output pin high
    - When interrupt generated
      - set output pin low
      - Put (1 – duty cycle) x pulse width into CCP register
    - Use interrupts versus polling
    - Requires more logic than using CCP module in PWM mode

Formulas

- Period = (PRx + 1) x 4 x (1/Fosc) x (timer pre scale)
  - Solve for PRx and assign to Prx
- Pulse width = Period x (duty cycle/100)
- Pulse width = CCPRxL:CCPxCON<5:4> x (1/Fosc) x (timer pre scale)
  - Solve for CCPRxL
- This is equivalent to
  - Pulse width = CCPRxL x (1/Fosc) x 4 x (timer pre scale)
- Determine CCPxCONbits.DC1B
  - Subtract integer version of pulse width from floating point
  - Divide result by 0.25 [0, 4)
  - Convert to int

Lab 7 Hints

- Use Timer0 and external count to generate an interrupt every second
- You can get away with just using the 8 MSB of the A/D conversion (if left justified)
- Use Timer1 as a counter – external time source(connected to the optical interrupter)
- In ISR with count >= value associated with 1 sec
  - Computer RPS
  - Toggle output
  - Clear Timer1 register(s)
- Use CCP2
- CMCON0bits.CM = 0b111;
- OPTION_REG = 0b10000101;
- Those doing this lab should be done tonight

Lab 7 questions: what do CMCON0bits.CM = 0b111 and OPTION_REG = 0b10000101 do for us?

Lab 6 Hints

- You may want to initialize your duty cycle to 1% so that you start out making sound
- I use CCP1 and Timer6
- Keep track of current mode, last pot value, last duty cycle, last period
- If no change to pot or mode
  - Do nothing
- If pot changes or mode doesn't
  - Recompute appropriate values based on mode
- I start the next A/D conversion when updating values and the LCD

Lab 6 Hints

- If frequency changes
  - Compute period
  - Find max oscillator that works
  - Determine and set PRx
  - Determine 8 MSB and 2 LSB of pulse width
  - Set values
- If duty cycle changes
  - Determine 8 MSB and 2 LSB of pulse width
  - Set values

Lab 6 Hints

- Infinite loop
  - Wait for A/D to finish
  - Poll buttons
  - Start next A/D conversion
  - Compute values
    - For some reason I'm currently computing and update all values each time through my loop
  - Set registers
  - Update LCD