5-2022

Analog & Digital Metronome

Andrew Deering
Trinity University, adeerin1@trinity.edu

Paul Kim
Trinity University, pkim1@trinity.edu

Gabriel Ogden
Trinity University, gogden@trinity.edu

TRINITY UNIVERSITY

Analog & Digital Metronome

ENGR-4367

Instructor: Dr. Kevin Nickels
Group: P1
PLEDGED: A. Deering, P. Kim, G. Ogden

5/16/2022

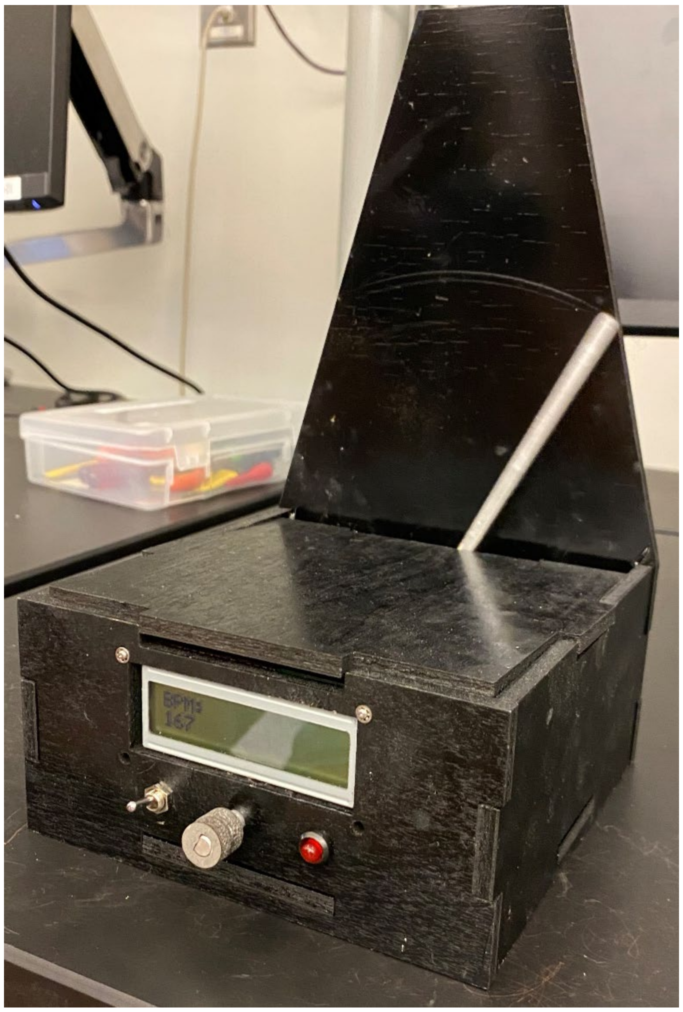

Table of Contents

1 Design Summary

The metronome is a tabletop device designed to keep a consistent rhythm for musicians to follow. This specific metronome combines the visuals of a traditional analog metronome with the ease of use and functionality of its digital counterpart. Like a conventional digital metronome, the user can control the beats per minute (BPM) using the control knob on the front of the device and read the current BPM value on the front mounted display. The user can observe each beat with any combination of the three separate outputs from the device. Firstly, a buzzer is used to generate a brief but loud audio tone at each beat. There is also an LED indicator mounted to the front panel of the device which flashes along with the set BPM. Finally, a silver arm near the back of the metronome oscillates back and forth in sync with the BPM mimicking traditional analog metronomes. The entire device is powered by an internal battery which can be turned on and off using a front panel mounted switch.

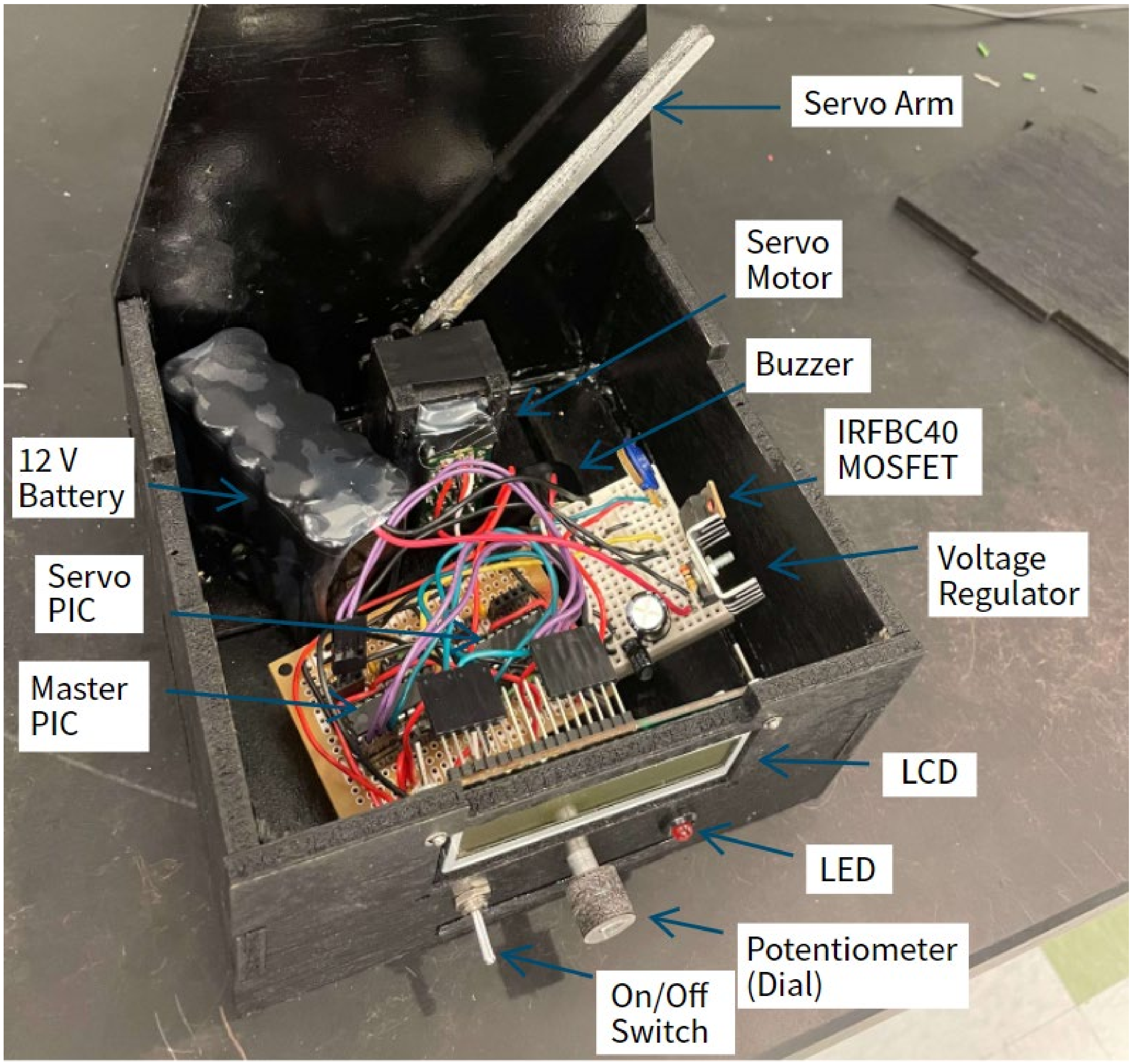

Figure 1: Guts of the device

2 System Details

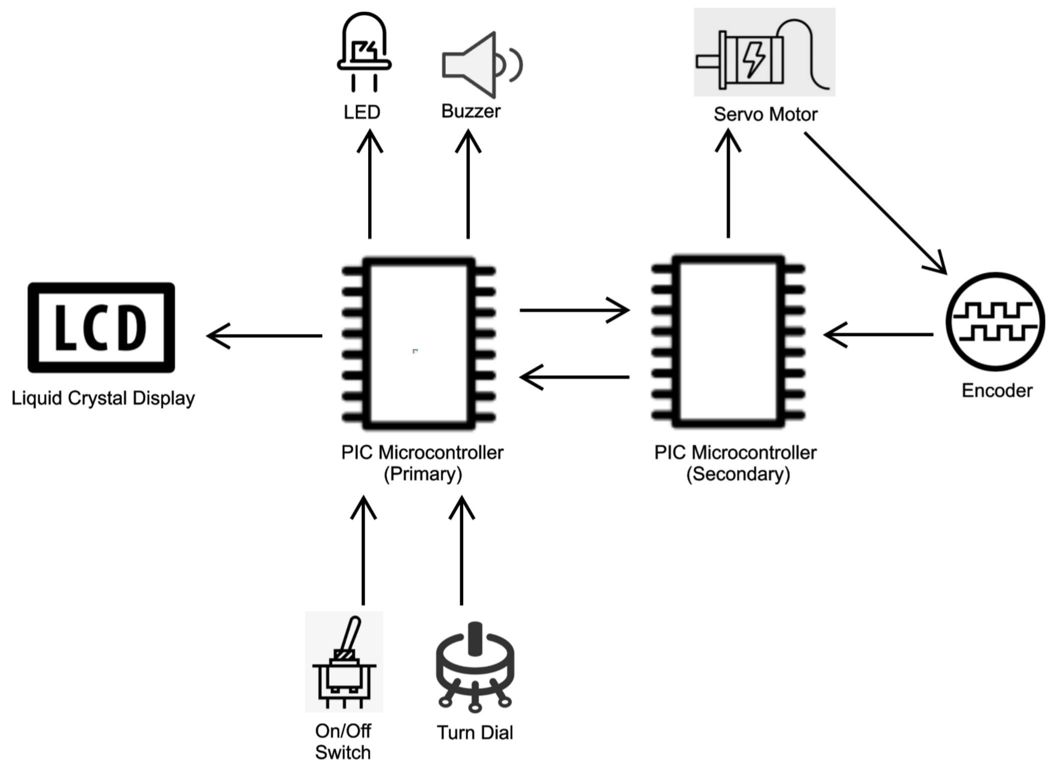

Figure 2: System functional diagram

2.1 Logic, Processing, and Control

The information processing and system control is executed by two PIC16F88 microcontrollers setup in a primary/secondary configuration as shown in Fig. 2. The primary (master) PIC handles the overall system flow shown in Fig. 3 and directly controls the display, audio, and user input systems of the metronome. The secondary (servo) PIC controls the actuator and sensor systems. Figure 4 shows a detailed depiction of the program logic for each PIC as well as the communication between them.

While there are several communication protocols that could be used to link the PICs such as serial, I2C, or SPI, even a slight delay could desynchronize the PICs due to the time-sensitivity of the metronome. Instead, the metronome inter-PIC communication system utilizes three direct, one-way digital connections for immediate information transfer. ‘Handshake’ and ‘Reset’ are directed from the master PIC to the servo PIC, while ‘Zero’ is directed from servo to master. The PICs are synchronized with a short (~20 milliseconds) procedure during the metronome setup, after which they can send bits back and forth without introducing any time delays or interrupting the system flow.

The master PIC runs on a timer interrupt system which increments a variable (ticks) every 16.384 milliseconds. This timing variable is compared dynamically to the desired BPM based on the time between beats (milliseconds), which is calculated with Eq. 1. When the time elapsed in the program reaches the time between beats, the LED and buzzer are turned on to indicate a quarter note at the current BPM. At this point, the master PIC also sets the 'Handshake' line high to communicate the quarter note with the servo PIC. The LED, buzzer, and handshake line are left high for a brief period to give the user time to process the various outputs. The LED, buzzer, and handshake line are then turned off until the time between beats is reached again, restarting the loop. If the BPM dial is adjusted at any point, the system will set the Reset line high to pause the servo PIC and wait until the BPM value is no longer changing before turning off Reset and executing the main loop with the new BPM parameters.

The servo PIC uses the hardware pulse-width-modulation (HPWM) integrated into the PIC to control the servo. Standard PWM could also have been used to send signals to the servo motor, however any implementation using PICBASIC Pro involves a time delay which could desynchronize the PICs. To generate the 20ms period PWM signal required by servo motors, the 8MHz internal oscillator was stepped down to 125kHz. When the handshake line is set high, the servo PIC switches the servo position setpoint by changing the HPWM duty cycle, thus oscillating the metronome arm at the desired BPM. The full PICBASIC Pro programs for the master and servo PICs can be found in sections A.2.1 and A.2.2, respectively.

$$Milliseconds\ per\ beat = \frac{60000}{BPM} \tag{1}$$

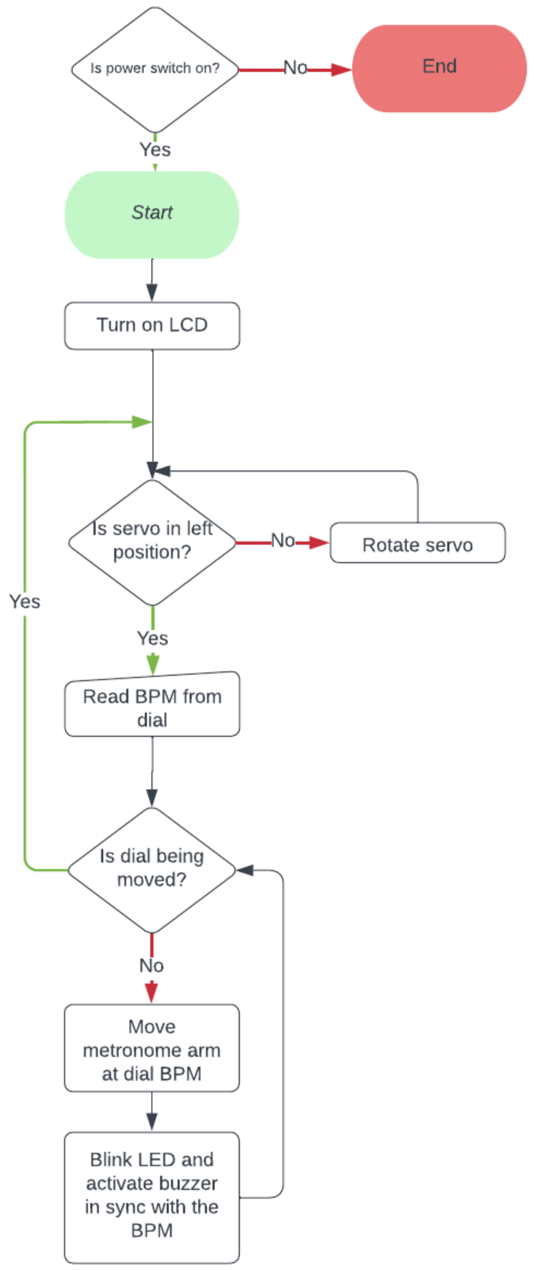
Is power switch on?
No
End
Yes
Start
Turn on LCD
Is servo in left position?
No
Rotate servo
Yes
Read BPM from dial
Yes
Is dial being moved?
No
Move metronome arm at dial BPM
Blink LED and activate buzzer in sync with the BPM

Figure 3: Metronome system flow chart

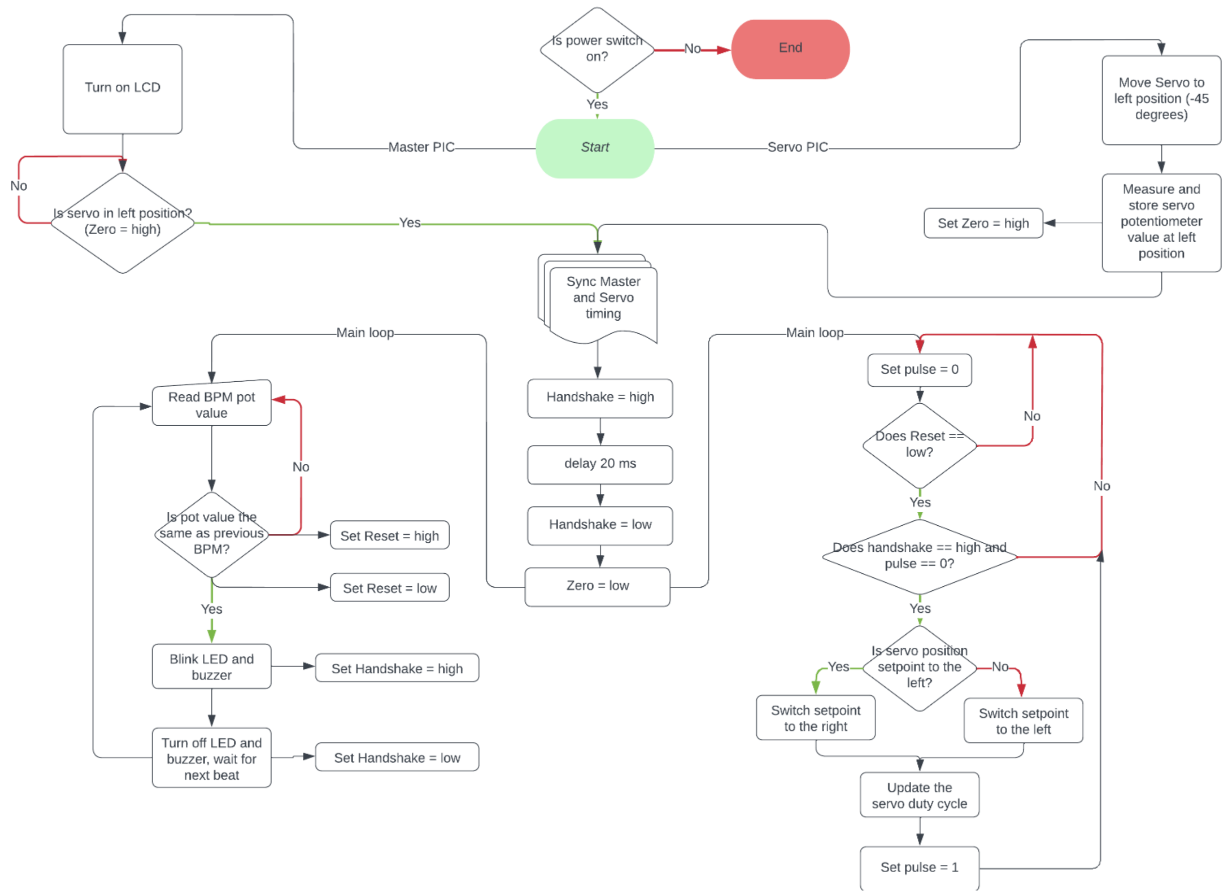
Is power switch on?
No
End
Yes
Start
Master PIC
Servo PIC
Turn on LCD
No
Is servo in left position? (Zero = high)
Yes
Move Servo to left position (-45 degrees)
Measure and store servo potentiometer value at left position
Set Zero = high
Sync Master and Servo timing
Main loop
Main loop
Read BPM pot value
No
Is pot value the same as previous BPM?
Set Reset = high
Set Reset = low
Yes
Blink LED and buzzer
Set Handshake = high
Turn off LED and buzzer, wait for next beat
Set Handshake = low
Handshake = high
delay 20 ms
Handshake = low
Zero = low
Set pulse = 0
Does Reset == low?
No
Yes
No
Does handshake == high and pulse == 0?
Yes
Is servo position setpoint to the left?
Yes
No
Switch setpoint to the right
Switch setpoint to the left
Update the servo duty cycle
Set pulse = 1

Figure 4: Software Flowchart depicting the program logic for the master PIC, the servo PIC, and the communication between them.

2.2 Output Display

The BPM is displayed using a liquid crystal display (LCD), which is connected to the master PIC. In the initialization phase, the LCD will display that the servo motor is calibrating. Once the motor is done calibrating, the LCD will display the current BPM value in decimal form and update accordingly when the BPM dial is turned. The LED is used to visually represent the BPM by blinking on each beat. Changing the BPM will increase or decrease the time between each blink.

2.3 Audio Output Device

The metronome audio output consists of a buzzer that will emit a brief tone at each beat, like the LED. The buzzer is driven by a pull-up MOSFET to generate loud tones, which can be seen in Fig. A-1.

2.4 Manual User Input

Two manual user inputs are used in this project: a switch and a potentiometer. A switch is used to turn the device on and off by connecting and disconnecting the battery from the rest of the circuit. A potentiometer is used as a dial to control the BPM based on the variable resistance. Both manual inputs are connected to the master PIC.

2.5 Actuators, Mechanisms, and Hardware

The metronome uses a servo motor as an actuator to swing an arm in time with the set BPM. The motor shaft position is determined by the duty cycle of the PWM signal from the master PIC. The arm rotates 45 degrees from vertical on either side for an overall 90 degree range of motion.

Servo motors require significant current to operate, so a 12-volt battery was selected as the metronome power source. A 5V voltage regulator steps down the batter supply voltage to the operating conditions of the main circuit.

2.6 Automatic Sensor

The servo motor used to move the metronome arm has a built-in potentiometer attached to the motor shaft which changes resistance based on the shaft angular position. The integrated control system for the servo uses this position measurement as feedback to achieve closed-loop control of the servo position. The analog potentiometer signal is accessible through the PCB underneath the servo base plate and is used to measure the angle of the metronome arm.

3 Design Evaluation

The LCD used to display the BPM and calibrating stage functioned reliably. However, we initially planned to use I2C communication to decrease the number of pins so only one microcontroller could be used. After considerable amount of research, implementing I2C communication for an LCD with a PIC was found to be difficult. Instead, serial communication was used between two PICs. An example of using the LCD with serial communication is given in the textbook. The LED did not take much research to implement but the LED worked without failure. The use of LED with the PIC was practiced during lab.

The buzzer initially worked at the correct BPM, but the sound was too quiet to hear well. Therefore, a MOSFET was used as a pull-up resistor to drive the buzzer at a constant five volts. This method was not discussed in class or lab and effortless research had to be done to create a functioning circuit.

The system used a switch and potentiometer for manual user input. The switch was used to turn the system on and off by connecting and disconnecting the battery to the rest of the circuit. Implementing the switch did not take much effort since the switch was set in series in between the battery and the voltage regulator. The potentiometer also did not take much effort to use as the potentiometer had pins that connected to voltage input, ground, and adjust. The adjust was set to the pin to control the BPM and another potentiometer was used to control the contrast of the LCD. However, both manual user inputs worked as intended.

The potentiometer used to measure the angular position of the servo shaft functioned reliably. Potentiometer sensors were covered in the textbook but not in lab activities and figuring out how to access the right node from the servo PCB required some research. However, implementing the analog voltage across the pot did not require extensive effort.

The actuator and hardware systems functioned reliably. The servo motor oscillated in sync with the rest of the metronome, and the power supply powered the system effectively. The manufactured parts of the device were completely laser cut from ¼” plywood. This was chosen because it was inexpensive, easy to iterate, and more than sturdy enough to be suitable for a metronome housing. The parts were designed to fit together simply like puzzle pieces and some glue and a coating of protective spray paint was used to aid in holding it together and increasing the durability and look of the product. Some effort was expended to create a clean aesthetic for final iteration of the hardware, though minimal research was required.

The logic and control systems for the metronome functioned reliably as designed. Timer interrupts were discussed in the textbook but not in any labs and required significant research to implement successfully. The frequency difference between the servo HPWM and the timer-interrupt system is the main reason for using two PICs for the control architecture. Consolidating to one PIC is possible but would have reduced the response time of our metronome to user input for a slight $0.50 cost reduction. Overall, the software development required significant research and effort but not enough to warrant the full 20-point rating.

4 Partial Parts List

Table 1:

Reference Number	Qty	Model Number	Part Name	Price (individual)	Price (total)	Vendor
1	1	S148	Servo Motor	$15.99	$15.99	Futaba
2	2	PIC16F88	PIC	$0.50	$1.00	Digi-key
3	1	1739	Buzzer	$0.95	$0.95	Digi-key
4	1	IRFBC40	MOSFET	$2.26	$2.26	Mouser

5 Lessons Learned

Time Expectations

If there was one constant throughout this design implementation process and implementation it was that everything can and will go wrong. Our time spent on the project was roughly five times what we expected. Whether it was having to wait a day to receive a part, getting stuck on code implementation, or just faulty hardware, there was always a delay. The entire process would have been much smoother if work, even small amounts, were started much earlier. This would help in gaining a better understanding on the true difficulty of certain project aspects that may be underestimated.

I2C with PIC

We initially decided to use an I2C backpack for our LCD display to reduce wire clutter and pin consumption based on experience from previous projects. There are several I2C LCD libraries that make interfacing and communicating with I2C components very simple *for Arduinos*. Even after over 30 hours of research and troubleshooting with PICBASIC Pro, we only managed to power the LCD. The documentation for I2C communication in the PICBASIC Pro manual is not helpful, and the example programs we found on various support resources were not easy to translate. If I2C communication is necessary, avoid PICBASIC Pro and use an Arduino to interface the component.

Debug systems

The I2C LCD was the first system we tried to implement, as LCDs make excellent debugging tools when a serial monitor is unavailable. However, we did not think to incorporate a standard LCD to debug the I2C communication which made troubleshooting tedious and ultimately ineffective. Even if an LCD isn’t part of the project design, incorporating one (or a serial monitor) early into project development can increase debug efficiency and subsequently save a lot of time.

Breadboard reliability

Up until the very end of the process, breadboards were being used for all of the circuit wiring. This was great for testing different configurations and components, but when the project was coming to a close, the heavy reliability on the breadboards led to a lot of issues. Shorts between heavily used pins and loose connections were the stem of many problems that used hours of time debugging just to find out it was a bad connection. This was eventually resolved with the use of soldered connections on a project board.

Time Synchronization

Synchronizing program timing to real time is crucial for a metronome. The PICBASIC Pro delay function is helpful for asynchronous timing functionality but cannot be used to keep time in parallel with the rest of the program. The first iterations of our metronome used the pause command to implement the time between beats and would quickly diverge from the correct BPM. Timer interrupts required a significant initial time investment to implement successfully but were very helpful and likely saved us a great deal of time trying to tune a pause-based timing system.

Part selection

Overall, the parts selected were appropriate in achieving the goals of the project. The dual pic system, while more difficult than controllers such as an Arduino (especially for LCD display), was sufficient in handling the logic of the system while costing a fraction of the other options. The servo is the only part where spending a little more could have gone a long way. The servo implemented was cheap and had plastic gears which caused it to be loud. A more expensive servo would limit the noise from the servo movement and greatly better the user experience.

6 References

[1] CLOCKX18. (n.d.), ME Labs. Accessed: May 16, 2022. [Online]. Available: https://melabs.com/samples/LABX18-16F88/CLOCKX18.htm

[2] HPWM_servo. (n.d), K. Nickels. Accessed: Date May 16, 2022. [Online]. Available: https://tlearn.trinity.edu/pluginfile.php/861101/mod_resource/content/3/hpwm_servo.pbp

A Appendix

A.1 Wiring Schematic

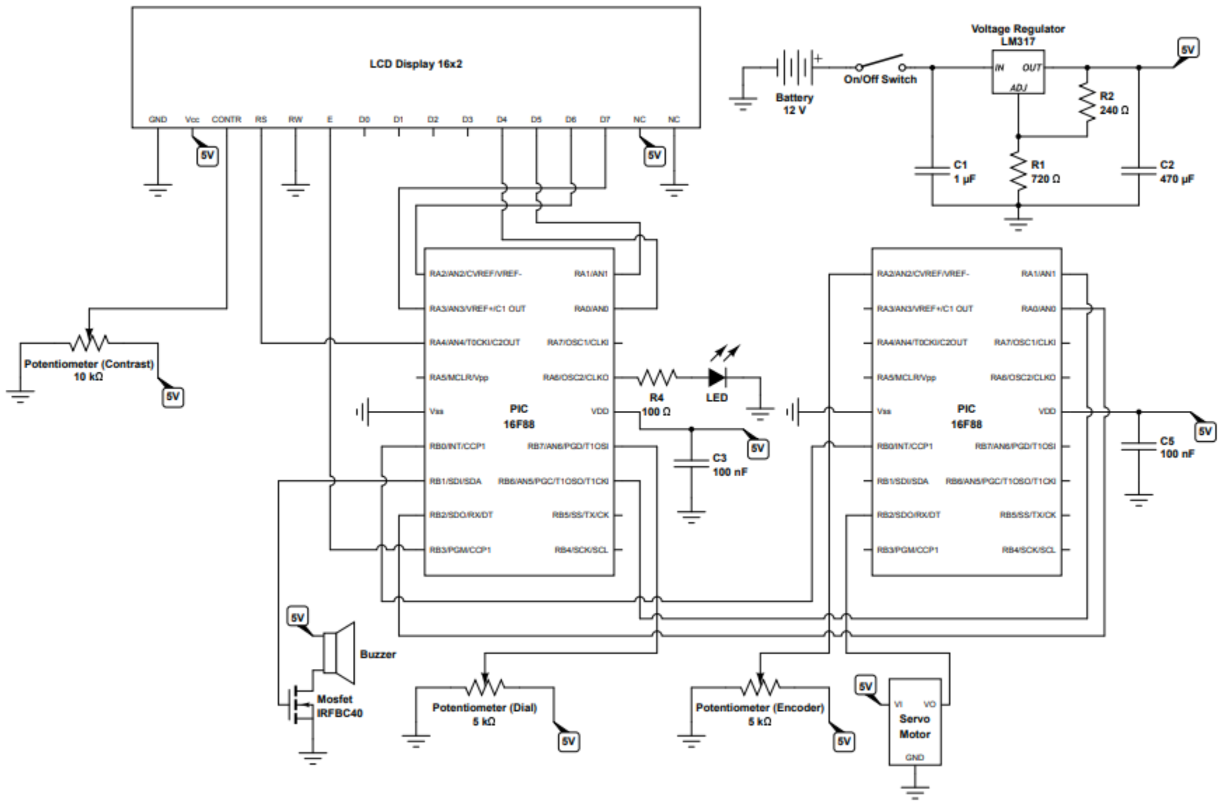

Figure A-1: Detailed wiring schematic, showing the connections for input and output devices.

A.2 PICBASIC Pro Code

The master PIC timer interrupt system is based on a clock program from ME Labs [1], and the servo PIC HPWM borrows from Dr. Nickel's demo [2].

A.2.1 Master PIC

```
'****************************************************************
'*  Name    : metronome_master_PIC.BAS                      *
'*  Author  : Andrew Deering                          *
'*  Date    : 5/4/2022                          *
'*  Version : 1.9                               *
'*  Notes   : Program to control the Master PIC for the         *
'*          : analog/digital metronome                    *
'*          : Timer interrupt based on MElabs 'CLOCKX18.pbp'    *
'****************************************************************
;----[16F88 Hardware Configuration]-----------------------------------------
#CONFIG
   __CONFIG _CONFIG1, _INTRC_IO & _PWRTE_ON & _MCLR_OFF & _LVP_OFF &
_WDT_OFF
#ENDCONFIG

;----[Oscillator Setup]-----------------------------------------------------
define OSC 8    '8MHz internal oscillator
OSCCON.4 = 1
OSCCON.5 = 1
OSCCON.6 = 1



;----[Initialize Hardware]---------------------------------------------------
ANSEL = 0 'turn off all A/D converters
ANSEL.6 = 1 'turn on RB7 analog input
```

```
TRISB.6 = 0  'set handshake pin to output

' Pin assignments
led          var PORTA.6   'LED connected to pin RA6
hand_shake    var PORTB.6   'Handshake output to servo PIC
reset        var portb.0   'Reset output to servo PIC
zero         var portb.2   'Zero input from servo PIC
buzzer       var portb.1   'Buzzer connected to pin RB1


;----[Variables]--------------------------------------------------------------
pot_value     var BYTE          'Analog POT value measure from BPM dial
BPM          var byte          'Metronome BPM variable
SPB          var word          'seconds per beat: 60000/BPM
ticks        var word          'timing variable (16.384 milliseconds per tick)

;----[Program Start]----------------------------------------------------------

low reset       'Resetting the communication lines to the servo PIC
low hand_shake
pause 500          ' Wait 500ms for LCD to start
LCDout $FE, 1      'clear lcd
LCDout $FE, $80, "Calibrating..."
pause 500       'wait for servo to reach reset position
do while (zero == 0)    'wait for servo PIC to send zero signal
loop

'send signal to sync program location with servo PIC
high hand_shake
pause 20
low hand_shake
```

```
do while (zero == 1) 'wait for time sync signal from servo PIC
loop

adcin 6, pot_value      'reading BPM dial on port RB7/AN6
BPM = pot_value         'storing measured BPM
spb = 60000/bpm         'caluclating the associated milliseconds per beat

'printing BPM value to LCD
LCDout $FE, 1
LCDout $FE, $80, "BPM: "
LCDout $FE, $c0, dec BPM

' Set TMR0 to interrupt every 16.384 milliseconds
OPTION_REG = %01010110 ' Set TMR0 configuration and enable PORTB pullups
INTCON = %10100000     ' Enable TMR0 interrupts

On Interrupt Goto tickint

ticks = 0 'reseting tick Count

;----[Main Program Loop]-------------------------------------------------------
main:
   while (1)       'loop forever
      if (pot_value > 2+BPM) or (pot_value < BPM-2)  then 'if dial changed
         low hand_shake
         adcin 6, pot_value   'remeasure dial BPM
         BPM = pot_value
         spb = 60000/BPm
      else  'normal metronome functionality
          adcin 6, pot_value     'measure dial value to compare to BPM variable
```

```
        endif
    wend
end



;----[Interrupt handlers]-------------------------------------------------------
' Interrupt routine to handle each timer tick
Disable         ' Disable interrupts during interrupt handler
tickint:      ' 61 ticks per second (16.384ms per tick)
   If ticks*163 < 200 Then   'extending the pulse time into the start of the next cycle
    high led
    high buzzer
    high hand_shake
   elseif ticks*163 < spb*10 then  'time between beats: low pulse
    low led
    low buzzer
    low hand_shake
   else   'ticks*163 = SPB: start high pulse
        high led
        high buzzer
        high hand_shake
        ticks = 0       'reset tick Count
   endif
   ticks = ticks + 1    ' increment tick counter between intrrupts
   intcon.2 = 0         'reset interrupt flag
resume main     'resume interrupts
```

A.2.2 Servo PIC

```
'****************************************************************
'*  Name    : metronome_servo_PIC.BAS                          *
'*  Author  : Andrew Deering                             *
'*  Date    : 5/4/2022                              *
'*  Version : 1.3                                   *
'*  Notes   : Program to control the servo PIC for            *
'*          : the analog/digital metronome                  *
'*          : Hardware PWM based on Dr. Nickels 'HPWM_servo.pbp'*
'**********************************************************
;----[16F88 Hardware Configuration]----------------------------------------
#CONFIG
   __CONFIG _CONFIG1, _INTRC_IO & _PWRTE_ON & _MCLR_OFF & _LVP_OFF
#ENDCONFIG

;----[Oscillator Setup]-----------------------------------------------------
DEFINE OSC 8        'stepping down 8 MHz internal clock to 125 kHz
OSCCON = %00010010 ' 125kHz
'pause command = seconds * 16
'pauseus = ms * 16

;----[Initialize Hardware]---------------------------------------------------
ansel = 0 ' Turn off the analog to digital converters.
ansel.2 = 1 'turn on porta2 ADC for the servo potentiometer
TRISB.0 = 0 ' Set PORTB.0 (CCP1) to output
TRISA.1 = 1  'set handshake line to input
CCP1CON = %00001100 ' Set CCP1 to PWM
T2CON = %00000101    ' Turn on Timer2, Prescale=4

;----[Pin Assignments]-------------------------------------------------------
hand_shake  var PORTA.1     'Handshake input from master PIC
reset       var portb.2     'Reset input from master PIC
zero        var porta.0     'Zero output to master PIC
```

```
;----[Variables]---------------------------------------------------------------
duty VAR byte  ' Duty cycle value (CCPR1L:CCP1CON<5:4>)
 ' Period =  20ms
 ' 0.5 ms = 16 = -180 deg
 ' 0.75 ms = 24 = -135 deg
 ' 1.0 ms = 32 = -90 deg
  ' 1.0 ms = 40 = -45 deg
 ' 1.5 ms = 48 = 0 deg
 ' 1.75 ms = 56 = 45 deg
 ' 2.0 ms = 64 = 90 deg
 ' 2.25 ms = 72 = 135 deg
 ' 2.5 ms = 80 = 180 deg

setPos var byte 'servo position setpoint
rightPos var byte 'servo position limit (right)
leftPos var byte 'servo position limit (left)
pulse var byte 'variable to determine wheteher to move the servo
position var byte 'variable to store the encoder value
leftpot var byte 'servo encoder value at leftpos

;----[Program Start]-----------------------------------------------------------
' Use formula to determine PR2 value for a 50Hz signal,
' 125kHz clock, and prescale=4. (125e3/(4*4*50))-1=155
PR2 = 155 ' Set PR2 to get 50Hz out

leftpos = 40 '-45 deg
rightpos = 56 '45 deg
setpos = leftpos

gosub setduty 'move servo to leftpos
pause 8 'wait 500 ms to stabilize
```

```
gosub readposition 'finding the left position encoder value
leftpot = position 'storing left position encoder value
pulse = 0
high zero 'telling master pic that servo is in position

'waiting for the main loop signal from master pic
do while (hand_shake == 0)
loop
do while (hand_shake == 1)
loop
low zero  'sending master pic time sync
setPos = leftpos 'set position left because main will switch position

;----[Main Program Loop]------------------------------------------------------
main:
    while (1)
       if reset == 0 then 'dial has not changed
         if hand_shake == 1 and  pulse == 0 then 'and servo is not in position
            gosub switchpos 'switch position
            gosub setduty
            pulse = 1
            low zero
         elseif hand_shake == 0 then   'pulse ended
             pulse = 0
             low zero
         endif
       else 'dial changed
           setpos = leftpos   'move servo to the left
           gosub setduty
       endif
    wend
```

```
end



;----[Subroutines]----------------------------------------------------
setduty:       'subroutin to set the duty cycle for the HPWM
  duty = setpos
  CCP1CON.4 = duty.0   ' Store duty to registers as
  CCP1CON.5 = duty.1   ' a 10-bit word
  CCPR1L = DUTY >> 2
return

switchPos: 'switching the servo position setpoint
   if (setPos == leftPos) then
   setpos = rightpos
      else
      setPos = leftPos
   endif
return



readposition:       'reads the servo position potentiometer
   adcin 2, position
RETURN
```