MicroPython
Introduction (cont.)
MCHE 201 – Spring 2019

Dr. Joshua Vaughan
Rougeou 225
joshua.vaughan@louisiana.edu
@Doc_Vaughan
In-class Exercise 8

• Connect a pushbutton

• Turn on the green LED

• When the pushbutton is pressed
  - Turn on the red LED
  - Turn off the green LED

• When the button is pressed again
  - Turn off the red LED
  - Turn on the green LED
  - Print the time elapsed between button presses to the REPL
In-class Exercise 8 Setup

```python
import pyb  # import the pyboard module
import time  # import the time module

# Assign the names to the onboard LEDs
RED_LED = pyb.LED(1)
GREEN_LED = pyb.LED(2)

# Assign the input pin to variable input_pin
# We set it up as an input with a pulldown resistor
input_pin = pyb.Pin("X6", pyb.Pin.IN,
        pull=pyb.Pin.PULL_DOWN)

# Turn on the green LED
GREEN_LED.on()
```
In-class Exercise 8 Core Logic

In-class Exercise 9

• Connect a pushbutton
• Turn on the green LED
• Once the button is pressed the first time, turn off all LEDs.
• Then, turn on 1 LED every 1s until the button is pressed again
• When the button is pressed again, print the time elapsed between button pressed to the REPL
• If more than 5s elapses:
  - Print "You took too long!!!" to the REPL
  - Turn on only the green LED again
import pyb # import the pyboard module
import time # import the time module

# Assign the names to the onboard LEDs
RED_LED = pyb.LED(1)
GREEN_LED = pyb.LED(2)
YELLOW_LED = pyb.LED(3)
BLUE_LED = pyb.LED(4)

# Assign the input pin to variable input_pin
# We set it up as an input with a pulldown resistor
input_pin = pyb.Pin("X6", pyb.Pin.IN,
        pull=pyb.Pin.PULL_DOWN)

# Turn on the green LED
GREEN_LED.on()}
# This will loop forever, checking the button every 100ms

```python
while (True):
    # read the state of the input
    input_state = input_pin.value()

if (input_state):
    start_time = time.ticks_ms()  # save the current time
    GREEN_LED.off()
    print("Button pressed to start.")

    # We could also have another delay here, to force a
    # longer separation between the pressing of the button to
    # start the timer and pressing it to end the timer.
    time.sleep_ms(200)

    # If true, we are waiting for the second button
    button_timing = True
```
if (input_state):
    CODE TO RESET LED PATTERN
    print("Elapsed time = {}ms".format(time_elapsed))

    # Set button_timing to False because we are no longer in a
    # timing part of the algorithm.
    button_timing = False

    # We could also have another delay here, ...
    time.sleep_ms(200)
else:
    if time_elapsed > 5000:  # >5000ms
        print("You took too long!!!")

        # We no longer want to look for the "timing" button press
        button_timing = False

        # Turn off the LEDs
        CODE TO TURN OFF ALL LEDs

    elif time_elapsed > 4000:  # >4000ms
        BLUE_LED.on()

    elif time_elapsed > 3000:  # >3000ms
        YELLOW_LED.on()

    elif time_elapsed > 2000:  # > 2000ms
        GREEN_LED.on()

    elif time_elapsed > 1000:  # > 1000ms
        RED_LED.on()
In-class Exercise 10

- Connect
  - a pushbutton
  - the servomotor

- Start the servo at 0 degrees

- When the pushbutton is pressed:
  - move the servo to 30 degrees
  - pause 1 second
  - move the servo back to 0 degrees

- Only allow this to happen once per 30 seconds
In-class Exercise 10 Setup

```python
import pyb  # import the pyboard module
import time  # import the time module

# Assign the input pin to variable input_pin
# We set it up as an input with a pulldown resistor
input_pin = pyb.Pin("X6", pyb.Pin.IN,
                      pull=pyb.Pin.PULL_DOWN)

# For the pyboard Servo 1 is connected to X1, Servo 2
# to X2, Servo 3 to X3, and Servo 2 to X4
servo1 = pyb.Servo(1)

# Set the initial angle to 0
servo1.angle(0)
```
# This will loop forever, checking the button every 10ms

```python
while (True):
    input_state = input_pin.value()  # Check the button state

    if (input_state):
        print("The button was pressed. Moving the servo now.")

        # Move the servo to 30 deg
        servo1.angle(30)

        # sleep for 1 second
        time.sleep(1)

        # Now, move back to 0 deg
        servo1.angle(0)

        # sleep for 29 seconds to disallow any other action
        # during this time.
        time.sleep(29)

    else:
        print("Button not pressed. Wait, then check again.")

    time.sleep_ms(10)  # Sleep 10 milliseconds (0.01s)
```
Your Motor Driver

• Same chip as the one driving stepper motors on the MCHE201 board, but...

• We have to control it manually, rather than over i2c
Pulse Width Modulation (PWM)

Method to control speed of Brushed DC motors

Voltage vs. Time

$V_{\text{max}}$
Pulse Width Modulation (PWM)

Method to control speed of Brushed DC motors

Voltage $V_{\text{max}}$

Time

Full voltage 1/2 of the time = 50% duty cycle
Pulse Width Modulation (PWM)

Method to control speed of Brushed DC motors

Voltage

Time

$V_{\text{max}}$

Full voltage $1/10$ of the time $= 10\%$ duty cycle
The pyboard – Timers for PWM

MicroPython pyboard
PYBv1.1

V+: 3.6v - 16v power input (supplied by USB when USB connected)
3V3: regulated 3.3v output only, max 250mA
VBAT: FET protected supply battery input
VBACK: backup battery input
A3V3: analog reference connected to 3V3 via inductor
X17 is pulled to GND via 4.7k resistor when USB pressed
P2-P5 are connected to the 4 LEDs
SD_SW = A8 is used for SD card switch
MMA_INT = B2 is used for accelerometer interrupts
MMA_AVDD = A10 is used for accelerometer power
connect BOOT0 to 3V3 and press RST to enter DFU mode

Image via micropython.org
Hardware Setup – MotorA

Notice the orientation of the SparkFun breakout. It's top is facing the bottom of this drawing.
### Table from TB6612 Specification Sheet

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
<th>Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN1</td>
<td>IN2</td>
<td>PWM</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>H/L</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L</td>
</tr>
<tr>
<td>H</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>H</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(High impedance)</td>
</tr>
<tr>
<td>H/L</td>
<td>H/L</td>
<td>H/L</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(High impedance)</td>
</tr>
</tbody>
</table>

Table from TB6612 Specification Sheet
# We need to set up two digital outputs that we will
# use to control the direction of the motor.
A1_pin = pyb.Pin("X8", pyb.Pin.OUT_PP,
           pull=pyb.Pin.PULL_DOWN)
A2_pin = pyb.Pin("X7", pyb.Pin.OUT_PP,
           pull=pyb.Pin.PULL_DOWN)

# We also need to set up a third pin to control the
# speed of the motor. This output needs to have PWM
# capabilities, so it should be connected to a pin
# that has an associated timer on the pyboard
PWM_pin = pyb.Pin("X6")
PWM_TIMER = pyb.Timer(2, freq=20000)
PWM_CHANNEL = PWM_TIMER.channel(1, pyb.Timer.PWM,
                                 pin=PWM_pin)
MotorA Basic Control

# to turn the motor in one direction,
# A1 should be high and A2 low
#
# To move in the other direction,
# A1 should be low and A2 high
A1_pin.value(1)
A2_pin.value(0)

# We set the speed by setting the duty cycle of the PWM
# command in the range 0-100. 0 means stop
PWM_CHANNEL.pulse_width_percent(50)

What’s a better way to do this?
Using the TB6612.py File

- Place TB6612.py on your pyboard
- Add import TB6612 to your main.py
import TB6612  # import the file containing our TB6612 motor code

# We need to set up two digital outputs that we will use to
# control the direction of the motor. In this case, the TB6612
# class file will handle the low-level settings for the pins
A1_PIN = "X8"
A2_PIN = "X7"

# We also need to set up a third pin to control the speed of the
# motor. This output needs to have PWM capabilities. If these
# are defined correctly here, then the TB6612 class file will
# handle the low-level set up
PWM_PIN = "X6"
PWM_TIMER = 2
PWM_CHANNEL = 1

# Now, create a motor instance, calling upon the TB6612 class
motorA = TB6612.motor(PWM_PIN, A1_PIN, A2_PIN, PWM_TIMER,
                      PWM_CHANNEL)
Basic Use with TB6612.py

# To start the motor, issue
# the .set_speed() command with a speed
# between -100 and 100
motorA.set_speed(50)

# To stop the motor, set its speed to 0
motorA.set_speed(0)

# Or, use the .stop() method
motorA.stop()
MCHE201 Magnetic Switch

- Switch closes when the magnet gets close
- Wire and process just like a pushbutton
MCHE201 Magnetic Switch
Magnetic Switch Code

```python
import pyb  # import the pyboard module
import time  # import the time module

# Assign the input pin to variable input_pin
# We set it up as an input with a pulldown resistor

# This will loop forever, checking the button every 100ms
while (True):
    # read the state of the input
    input_state = input_pin.value()

    if (input_state):
        print("The magnet is close by.")
    else:
        print("The magnet is not close by.")

    # Sleep 100 milliseconds (0.1s)
    time.sleep_ms(100)
```
MCHE201 Small Solenoid

- Retracts when coils are energized
- Spring return when not
- Use a MOSFET to control it and pyboard to control the MOSFET

(https://bildr.org/2012/03/rfp30n06le-arduino/)

Image from: https://www.sparkfun.com/products/11015
Small Solenoid Hardware Setup

• Do *NOT* leave the solenoid powered
• Connect to "upper" screw terminal labeled 5V
import pyb  # import the pyboard module
import time  # import the time module

# Assign the output pin to variable mosfet_pin
# We set it up as an output with a pulldown resistor

# This loop will run 5 times
for counter in range(5):
    print("MOSFET on. Solenoid should retract.")
    solenoid.high()
    time.sleep_ms(100)  # Sleep 100ms

    print("MOSFET off. Solenoid should extend via spring.")
    solenoid.low()
    time.sleep(1)  # Sleep 1 second
All of the code contained in this lecture is available at the MCHE201 Class Repository on GitHub:

https://github.com/DocVaughan/MCHE201---Intro-to-Eng-Design