Midterm/Final Exam Review

This review assignment will not be submitted for credit, but will help you study for both he coming exams (midterm and final). The review is compiled from old exam questions so will be in the same format you should expect for the exams; however, this review is not guaranteed to cover every possible type of question that may be on the exam. This review guide is also considerably longer than you should expect for the midterm exam. For the review you should look at the online Micropython documentation but on the exam pertinent excerpts from the documentation will be provided.

1. Please indicate (T)rue or (F)alse regarding each of the following statements. Only mark (T)rue if *every* aspect of the statement is true. [2 pts each]

   (a) A task-based system utilizing the priority scheduler may have problems if the tasks are not written to be cooperative resulting in tasks taking longer than desired and ultimately affecting the ability for tasks to run on time. If a high priority task takes longer to run than its specified period other lower priority tasks may not run at all.

   ☐ T

   ☐ F

   (b) Python lists are extremely powerful due to their multitude of features, however they are inefficient at storing data. If efficiency of data storage, for any type of data, is critically important, an object of `array.array` should be used to store the data rather than using a list.

   ☐ T

   ☐ F

   (c) Python integers are not stored in a simple binary format as with other languages like C, C++, or Assembly. Instead, several extra bytes of data are required to store information about the integer for Python to use internally. As arithmetic is performed with integers, the resulting output integers are resized automatically to always provide enough bits to store the resulting value.

   ☐ T

   ☐ F

2. Explain, in moderate detail, the difference between a Python class and an object of that class.

3. What does the `__init__()` method do in a Python class?

4. Explain the significance of the variable `self` in Python class methods.

5. Complete the following truth tables for binary operations:

A	B	A & B
0	0	
1	0	
0	1	
1	1	

A	B	$A \mid B$
0	0	
1	0	
0	1	
1	1	

A	B	$A \mid \overline{B}$
0	0	
1	0	
0	1	
1	1	

A	B	A ^ B
0	0	
1	0	
0	1	
1	1	

6. Assume that a Python integer called `reg_val` contains the binary number `0b10110101`.

   (a) What is the decimal value of this number in the following formats, *without* sign extension:

      i. Signed 8-bit

      ii. Unsigned 8-bit

      iii. Signed 32-bit

      iv. Unsigned 32-bit

   (b) What binary number would be produced after sign-extending `reg_val` from an 8-bit signed number to a 32-bit signed number? What is its decimal value?

   (c) What binary number would be produced after sign-extending `reg_val` from a 16-bit signed number to a 24-bit signed number? What is its decimal value?

   (d) Using only binary or bitwise operations, write a line of code that evaluates to `True` if the bit in position b3 is clear, and will evaluate to `False` otherwise.

      Now work out the binary operations below to show the correct evaluation.

   (e) Using only binary or bitwise operations, write a line of code that will assign the value `0b110` to bits b5 through b7 without changing the value of bits b0 through b4.

      Now work out the binary operations below to show the correct evaluation.

7. Write a Python generator function that computes “Tribonacci” numbers, similar to Fibonacci numbers. That is, write a generator function that yields values, T_n, from the sequence defined by:

$$\begin{aligned} T_1 &= 1 \\ T_2 &= 1 \\ T_3 &= 2 \\ T_n &= T_{n-1} + T_{n-2} + T_{n-3} \ \forall\, n \geq 4 \end{aligned}$$

Write the function so that you can choose a maximum length for the sequence with a parameter when a generator object is created from the generator function.

Now write a few lines of code to create a generator object from your function and print out values in the following format:

```
T1 = 1
T2 = 1
T3 = 2
T4 = 4
...
```

8. Write a Micropython callback function to be triggered by external interrupts using objects of `pyb.ExtInt`. Also write code to set up three different callbacks to the same callback function triggered by button presses on PC0, PC1, and PC2. In your callback function write code to determine which pin triggered the callback and then print the correct name of the pin using a standard print statement.

9. Describe, with words, how you would write to a register on an external I^{2}C device. That is, explain the procedure to select a particular device on the bus, to select a particular register on the device, and then to actually write the data. Assume that the data to be sent is the following 14-bit number `0b111011010110` which must be *left-aligned* when written to the peripheral device.

   Answer this question with a level of detail similar to that covered in lecture; that is you don't need to draw a full timing diagram, but you should answer the question from the perspective of behavior on the SDA and SCL lines.

10. Now write some code to create an object from `pyb.I2C` and and send the data from the preceding part of this question using `pyb.I2C.mem_write()`.

11. Now write some code to send the same data once again but this time without using the function `pyb.I2C.mem_write()`; instead use `pyb.I2C.send()`.

12. In lecture we discussed several varieties of GPIO configurations. Draw an example circuit and describe the correct configuration for the following connections to a GPIO pin:

    (a) An active-high button connected to an input on the MCU.

    (b) An active-low button connected to an input on the MCU.

    (c) An MCU output pin to an enable pin on an external peripheral device.

    (d) An MCU output pin connected to an enable pin on an external peripheral device that, itself, is programmed to drive the enable pin low internally when a fault condition occurs.

13. Describe the main differences between I^2C, SPI, and UART; include example use-cases that are suited to each protocol.

14. We've thoroughly discussed motors in lecture. Describe in words the difference between a stepper motor and a traditional DC brushed motor. Be thorough in your explanation and discuss some of the differences from the perspective of the motor construction in addition to its operation, e.g. that a stepper motors doesn't have brushes and why that is the case.

15. Consider the following snippet of code meant to wait for character input entered through PuTTY or a similar terminal interface and then to parse the character input. The snippet includes several bugs, some of which may throw exceptions and some of which may not. Note that blocking code is not considered a bug.

```
import pyb
ser_port = pyb.USB_VCP()

while not ser_port.any():
    break
char_in = USB_VCP.read()

if char_in = 'P':
    print('You pressed 'P', didn't you!')
elif char_in = 'Q'
   print('Quit that!')
else char_in = 'R' or 'r':
    # do nothing
else:
    ser_port.write(char_in)
```

Identify three unique bugs and rewrite the associated lines of code to eliminate the bugs. Do not change the functionality of the code.

16. The content related to this final question will be on the final exam, not the midterm. You'll need to complete the IMU lab before you can handle an exam question like this. However, you should still complete this problem at some point because *your final will have a question regarding the same I2C hardware peripheral.*

    The MCP23008 Port Expander from Microchip Technology is a type of integrated circuit that provides additional GPIO (general purpose input/output) pins that can be read or controlled over a serial communication line. To complete the following questions, you will need to refer to the datasheet for the MCP23008 port expander: `https://ww1.microchip.com/downloads/en/DeviceDoc/MCP23008-MCP23S08-Data-Sheet-20001919F.pdf`.

    You may also benefit from reviewing the Micropython I^{2}C driver documentation: `https://docs.micropython.org/en/latest/library/pyb.I2C.html`, and also the datasheet for the STM32L476: `https://www.st.com/resource/en/datasheet/stm32l476rg.pdf`.

    Complete the following:

    (a) In reference to the pinout description table in the datasheet for the MCP23008, select appropriate CPU pins on the STM32L476 to use for I^{2}C communication. Select appropriate pins for I^{2}C2 specifically. Additionally, consider the hardware address input pins on the port expander and how they must be connected for the device to operate properly.

    i. Draw a very simple schematic showing how the I^{2}C interface is connected between the STM32L476 and how the hardware address pins are connected. Be sure to label the signals and pin names clearly.

    ii. Write a short snippet of code to create an object of the `pyb.I2C` class properly configured to interface with the MCP23008.

    iii. Assume that the initializer for the `pyb.I2C` class *does not* configure the pins needed for proper I^{2}C communication. Write several lines of code to initialize all the pins required for I^{2}C. *Hint:* make sure to set the alternate function parameter when creating the pin objects.

    iv. How did you know which pins to connect and what alternate functions to use? Write a sentence or two describing the procedure for determining this information from the provided datasheets and references.

(b) You have been asked to use the I^{2}C port expander to drive a set of eight LEDs arranged sequentially in a straight line.

For this question assume that all eight pins on the port expander are connected to LEDs in a *common anode* configuration. That is, the anode of all eight LEDs are connected directly to a voltage source and the cathode of each LED is connected to one of the eight GPIO pins through a current limiting resistor.

The eight LEDs are to be used as indicator LEDs for the state of charge of a battery operated device. The LED indicators work in the following fashion:

- In its off state and default or idle state the device disables the LED indicators for reduced power consumption.
- If a button (similar to the blue user button on the Nucleos) is depressed for more than two seconds, the device toggles between on and off. When the device is powered on, the LEDs blink in unison three times quickly in succession.
- If the device is powered on and the button is pressed once, quickly, the device will illuminate the appropriate number of LEDs to show the state of charge of the battery. If the device is low on battery (less than 12.5% charge) the LEDs blink in unison four times in quick succession.
- Regardless if the device is on or off, when it is plugged in to charge, the LEDs display the current charging status. That is, the LEDs display a progress bar as the device charges, each LED indicating 12.5% of full battery charge. As the LEDs fill up, the "pending" LED should blink. For example, if the battery is 55% charged the first four LEDs should be fully illuminated while the fifth LED should be blinking and the remaining three LEDs should be off.

Complete the following:

i. Begin by designing a finite state machine that represents the required functionality for this system. Draw a state transition diagram and indicate the required inputs and outputs and annotate the diagram clearly.

ii. Write one or more "helper" functions in Python that will allow you to easily interface with the LEDs through the port expander.

iii. Write a Python generator function to implement the finite state machine that you designed for the system. In the states of your task you can use the helper functions written to perform the LED interfacing.

iv. Describe your design with a few sentences or paragraphs. What frequency should your task use to make sure the state machine handles the design requirements? Does your design assume other code is running like other tasks or interrupts? If so, describe any dependency your state machine has on things defined outside the task.