Dennis Zang
704766877
April 20, 2019

Assignment 2

Problem 1: Hexiwear & Raspberry Pi

1) The standard clock frequency used in the MCU core of the Hexiwear is 120 MHz.

2)

S.No.	Sensor	Part Number	Function
1	3 axis Accelerometer	NXP FXOS8700CQ	Used to measure the acceleration/orientation along x,y,z
2	3 axis Digital Gyroscope	NXP FXAS21002	Used to measure and determine the absolute orientation/direction of the device
3	Pressure Sensor	NXP MPL3115A2R1	Used to measure pressure [Pascals] /altitude [meters] and temperature [degrees Celsius]
4	Humidity and Temperature Sensor	MEAS HTU21D	Used to measure local humidity and temperature
5	Heart-rate Sensor	Maxim's MAX3010x	Used to measure the frequency of one' heart rate

3) According to page 21 of the datasheet for the NXP FXOS8700CQ, the maximum acceleration the sensor can measure is about 8g, given that the detection range is set to the maximum at +-8g. The usable threshold range for freefall detection should be between +-100mg and +-500mg, according to page 22 of the same specifications.

4) The key difference between the Raspberry Pi and the Arduino is that the Raspberry Pi has dedicated I/O (for devices like display and camera) and a functioning operating system (vs the firmware the Arduino uses to simply execute code).

5) The CPU architecture used in Raspberry Pi is a single core 1GHz ARM v7 processor (BCM2835).

6) If you draw too much power from the Raspberry Pi, the device will be unusable, as the Raspberry Pi has regulators to switch off current if the current being drawn is not acceptable.

Problem 2: Digital Event Timestamping

For problem 2, I did as the isntructions asked (with DigitalIn, DigitalOut, and PwmOut), and a bit more. For the PTD2 port, I made use of InterruptIn to add interrupt handling, so the program will independently handle when the voltage on the PTA10 port rises. In addition, I also added an instance of EventQueue on a separate thread, so that way, the handler functions can execute in order separately from the pwm (and because printf is not allowed in ISR context). To handle timing, I used an instance of timer to create timestamps for the outputs.

Problem 3: Handling User Input

For problem 3, I added an instance of Serial to allow for input and output from the user's terminal. However, unlike problem 2, I added a thread that runs indefinitely in the main routine, which constantly polls for user input and immediately updates the pwm if necessary.
(I have considered attaching handler functions to rx interrupts, but that cannot work with PwmOut as PwmOut makes use of mutexes.)

Problem 4: Generation of Digital Waveforms

For problem 4, I manually made an infinite loop that turns the DigitalIn port on and off with specific delays. I also attached a handler function to rx interrupts on RawSerial to handle the inputs and managing the updates.
(Note: The program has worked perfectly prior to the recording and testing. The current program under testing cannot handle large user inputs and has bugs for smaller values of the PWM period.)
(Updating the global variables shouldn't have triggered mutex errors, but there are some reported bugs on this. The program behavior is also erratic and not constant [as it worked completely earlier but not now].)