

## 2.996 / 6.971 – Labs 4, 5, and 6

### Guidelines and Deliverables

In these labs, you will solder up your PCB designs and write code for the TI MSP430F2274 microprocessor. The primary goal is to get some hand-on experience with putting a primarily SMD board together and with programming techniques for sensor-based designs. You can demonstrate the functionality of your custom block, but keep in mind that it is more important to get some experience with the microprocessor and standard block components because these will likely be used again for the projects.

#### **Lab 4 – M 10/1 and T 10/2 – Assembly and Programming Check**

Pretty simple – get your board put together. Brand-new soldering stations are available in the lab and we will be around to help, particularly if you've not worked with surface-mount components much before. Assemble the standard block first. In case you run out of time, you can always go back and finish your custom block later. Resist the urge to solder all of your components at once! Break up the board into a few steps so that you can test your soldering as you go. For example, start with the passive components and the power supply. Confirm your 3.3V and 5V lines before putting down the microprocessor and other chips. When you finish assembling, test that the board works by hooking up the JTAG programmer and attempting to program the microcontroller.

*Lab 4 Deliverables:*

- \_\_\_\_\_ Standard block assembled.
- \_\_\_\_\_ Functionality test: Hook up JTAG programmer.

#### **Lab 5 – M 10/8 and T 10/9 – Programming I**

Using the notes from lecture and the MSP430F2274 User's Guide, set up and test some of the basic microcontroller functions, such as the clocks, the digital IO, the ADC, and the UART. Go through the process of finding information in the User's Guide, particularly from the functional diagrams and register maps. Try to do the tasks listed below in order, getting them checked off as you go, and if you get stuck we'll be around to help.

*Lab 5 Deliverables:*

- \_\_\_\_\_ Set up 16MHz clock. Output ACLK on pin 2.0. (Set BIT0 in P2DIR and P2SEL.)
- \_\_\_\_\_ Configure digital outputs. Set some pins high and low. (Test: drive an LED.)
- \_\_\_\_\_ Use a timer to output a 1.00kHz square wave.
- \_\_\_\_\_ Set up the UART to continuously transmit a character over the USB serial connection, or the radio. (Use a standard baud rate: 9600bps, 19200bps, etc. There are

configuration guidelines for these rates in the User's Guide. Test with a terminal program.)

\_\_\_\_\_ Set up the ADC. Transmit an ADC result over the UART.

### **Lab 6 – M 10/15 and T 10/16**

Continue experimenting with the MSP430F2274. Set up an event-driven routine to communicate with the specified the protocol below. You will need to create a circular buffer to handle the input string and reply with the appropriate outgoing message. We will use a program to test its functionality by “requesting” ADC values or other information over the UART.

#### *Lab 6 Deliverables*

\_\_\_\_\_ Working circular buffer

\_\_\_\_\_ Toggle Pin Command works

\_\_\_\_\_ Set PWM Duty Cycle Command works

\_\_\_\_\_ Get ADC Value Command works

## Our test communications protocol:

### *General:*

- Each data packet consists of eight consecutive bytes. They are numbered 0 thru 7 in the order they are transmitted/received. Byte(0) is ALWAYS 0xFF and conversely 0xFF ALWAYS signifies the start of a new packet. (No other bytes in the packet can be 0xFF.)
- Byte(1) is a command that can indicate what of the rest of the packet is, what is being requested, what is being transmitted, etc.
- Byte(2)-Byte(5) can be used to send data. If there is no data associated with a command, they can be anything.
- Byte(6) is an 8-bit cyclic redundancy check on Byte(1)-Byte(5). See the lecture notes and sample code for more information on how to generate this.
- Byte(7) contains escaped character flags, which indicate if any of Byte(1)-Byte(5) had the value 0xFF. The most significant bit, Bit(7), is always zero. For  $0 \leq n \leq 5$ , Bit(n) is set if Byte(n+1) was escaped from 0xFF to 0xFE. The transmitting routine must do this by checking Byte(1)-Byte(6) and setting these flags as necessary. The receiving routine must change them back accordingly. See lecture notes and sample code for more information.

### *Toggle Pin:*

Command: 0xFF, 0x01, 0x00, 0x00, 0x00, 0x00, (CRC), (ESC)

Response: no transmission required, just turn on/off an LED

### *Set PWM Duty Cycle:*

Command: 0xFF, 0x02, (DUTY), 0x00, 0x00, 0x00, (CRC), (ESC)

Response: no transmission required, set the duty cycle of a PWM output (any frequency)

Details: (DUTY) = 0-255 maps to 0-100% on

### *Get ADC Value:*

Command: 0xFF, 0x03, (CH), 0x00, 0x00, 0x00, (CRC), (ESC)

Response: 0xFF, 0x03, (CH), (8MSB), (2LSB), 0x00, (CRC), (ESC)

Details: (CH) = analog channel (decimal 0-15)

(8MSB) = high 8 bits of 10-bit ADC reading

(2MSB) = right-justified low 2 bits of 10-bit ADC reading (0b000000xx)