Designing with a Microcontroller – EE v4

Safety: In this lab, voltages are less than 15 volts and this is not normally dangerous to humans. However, you should assemble or modify a circuit when power is disconnected and don't touch a live circuit if you have a cut or break in the skin.

Objective: This lab provides an opportunity to learn about the use of a microcontroller (MCU) and its interfacing to external devices. The lab uses the Stellaris® LM4F120 LaunchPad that has an ARM Cortex M4F (LM4F120H5QR) MCU in it.

The lab has two parts. Part I of lab provides and introduction and then uses C language programming to drive a 7-segment display. Part II builds on the first part and the student is to design a microcontroller solution to solve one of several tasks.

Background: The LM4F120 is a 32-bit ARM Cortex-M4F microprocessor that runs at 80 MHz with 256 kB flash memory, 32 kB RAM, 43 programmable general purpose I/O (GPIO) ports, two 12-bit ADC modules and 12 general purpose timers. Like most microcontrollers, the I/O ports in LM4F120 are memory mapped. To increase the readability of the code, we will use symbolic definitions for these ports. For example, GPIO_PORTF_DATA points to the memory location 0x4002.53FC that holds the status (or data) of Port F (see in the sample code how it is done). Note that, these definitions are user defined. While declaring these definitions, the appropriate memory address must be taken from datasheet [2]; for example, GPIO_PORTF_DATA can be found on p.615 of the datasheet. In order to use any GPIO, several control registers need to be configured. Some key registers are as follows: GPIO_PORTF_DIR (to set the input or output direction, p.616), GPIO_PORTF_AFSEL (to enable/disable alternate functions, p.625), GPIO_PORTF_DEN (to enable digital functionality, p.636), GPIO_PORTF_PCTL (to enable clock on port, p.641), and GPIO_PORTF_PUR (to enable pull-up resistors, p.631).

Bit Masking in Data Register Operation: A special mechanism is used in Stellaris® microcontrollers that allows programmer to individually modify the GPIO DATA registers with single instruction by using bits [9:2] of the address bus as a mask – this is known as bit masking. Refer to p.608 of the datasheet [2] for a detailed description. For an example, the base address of Port F is 0x4002.5000. If we want to read and write all 8-bits of this port, the offset value we need to add to the base address is 0x03FC (i.e., 0011_1111_1100 in binary). This is why in the starter code, GPIO_PORTF_DATA points to the memory location 0x4002.53FC, instead of 0x4002.5000. As a result, all read and write operations to GPIO_PORTF_DATA will access all 8-bits of Port F.

If we are interested in just bit 3 of Port F, we will add an offset value of 0x0020 (i.e., 0000_0010_0000 in binary) to the base address. In that case, GPIO_PORTF_DATA will point to memory location 0x4002.0020.

The lab will use Keil uVision4 toolchain (MDK-lite version 4.6) which is already installed on the lab computers. The user manual for the LaunchPad can be found in [1] (also posted on EE392 website). The user guide for uVision4 toolchain can be found on Keil's website: http://www.keil.com/support/man/docs/uv4

Part 1a – Starter project (< 1 hour)

Obtain from the technicians one Stellaris® LM4F120 LaunchPad and one four-digit 7-segment display module. Use the USB cable to connect the lab computer's USB port to the Power/ICDI micro-USB port at the top of the LaunchPad. Make sure the slide switch in the top left corner is in the right hand position.

In order to verify the LaunchPad circuit function and to become familiar with loading a program into the LaunchPad, you should run the starter project is provided on the course website. Using the lab computers in 2C61 or 2C80, download the project (starter.zip), unzip it and run the uVision4 project named "start_0.uvproj". Within uVision4, go to "Projects > Options for Target". Under the "utilities" tab, make sure that Stellaris ICDI is listed as the target driver for flash programming. Close the window and "Load" (download) the code into the flash memory of the MCU. To program the LaunchPad's MCU, the Power Select Switch must remain in "Debug" position and the USB cable must remain connected since the LaunchPad is powered from it. For other power options, consult p.11 of the user manual [1]. Press the RESET button (must do after every download) then press SW1 (active low) and observe the LED flashing (active high).

Now study the starter code (starter_0.c) and the comments after every line – it will help you better understand the code, syntax and configuration of LM4F120 MCU. The code is to simply turn on the onboard red, blue and green LEDs in sequence when the switch (SW1) is pressed. The switch and LEDs are all connected to Port F.

LaunchPad Interface: Before you go to the next section and build your circuit, you should carefully note the non-sequential pin-out diagram of the LaunchPad shown in Fig. 1. Although the LM4F120H5QR MCU has 43 GPIO, only 35 of them are available through the LaunchPad. They are: 6 pins of Port A (PA2-PA7), 8 pins of Port B (PB0-PB7), 4 pins of Port C (PC4-PC7), 6 pins of Port D (PD0-PD3, PD6-PD7), 6 pins of Port E (PE0-PE5), and 5 pins of Port F (PF0-PF4). In addition, there are two ground, one 3.3V, one 5V (VBUS), and one reset pins available on the LaunchPad.

Pins PC0-PC3 are left off as they are used for JTAG debugging. Pins PA0-PA1 are also left off as they are used to create a virtual COM port to connect the LaunchPad to PC. These pins should not be used for regular I/O purpose.

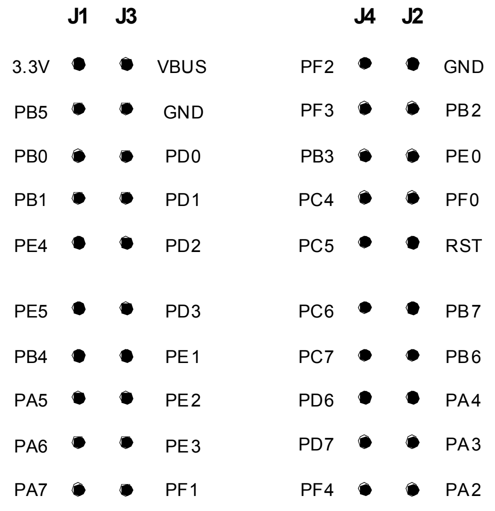


Fig. 1 Header pins on the LaunchPad (EK-LM4F120XL)

Part 1b: Display system using 7-segments (< 2 hours)

In this experiment, we will program the LaunchPad and four-digit 7-segment display module to display "UOFS". In addition, when SW1 is pressed, the display is to flash the letters "H", "E", "L", "L", "O" one at a time then settle back to "UOFS". All display segments should be turned off with a noticeable delay between the display of each letter.

Note carefully the diagram of the display module. Obtain other components shown in Fig. 2 and connect the circuit. The LaunchPad board is sufficient to supply +5 V to Vdd of the 7-segment display module - connect the display power pins directly to VBUS and GND of J3. For now, you may omit the pot (required for **Part 1c**). The experiment uses C language to program the MCU. Using the code template (provided on the course website), write a program to accomplish the task. You will need to open a new project on Keil uVision4. Note that display latches are connected to PA4-PA7. Using the Latch Enable ($\overline{LE}$) pin, your program can update the display digits one at a time.

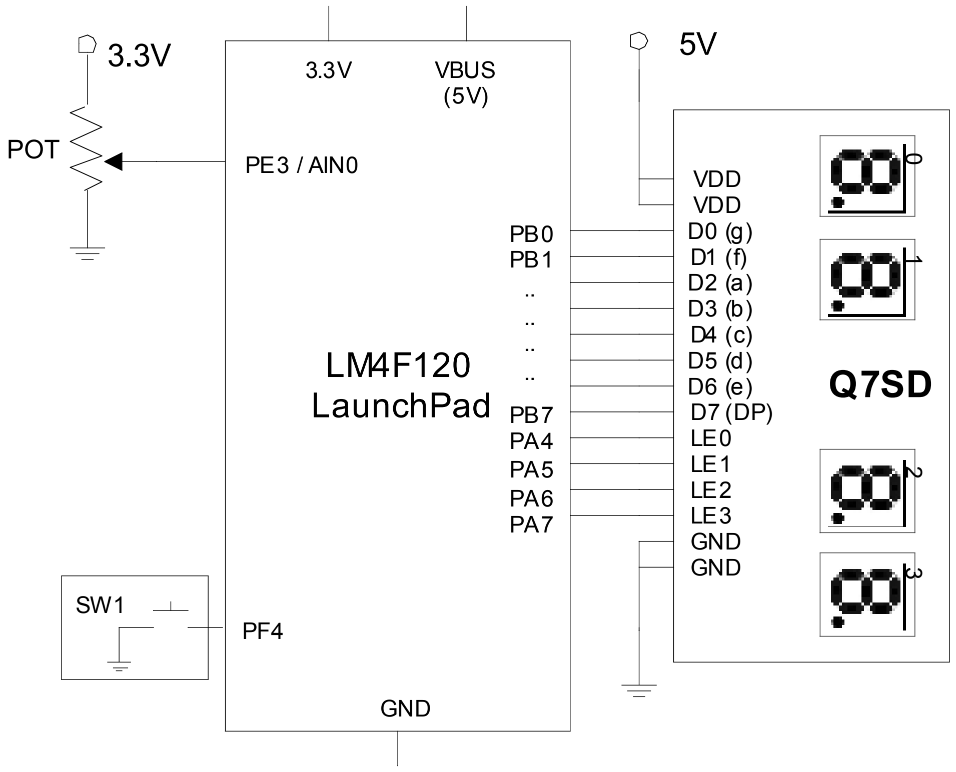


Fig. 2 Connection to 7-Segment Display

Part 1c: Analog-to-Digital Converter (optional)

In this experiment, we will use one ADC module to sample the voltage across the pot (shown in Fig. 2). The sampled voltage will be converted to a hex value and displayed on the display board. For a full resolution, i.e., 12-bits, we will need three 7-segment displays.

LM4F120 MCU has two 12-bit ADC modules. Each module is controlled by a number of registers and offers a variety of options. The module has four sequencers. For simplicity, we will use only sequencer 3 since it captures only one sample per trigger and stores the sample into the corresponding FIFO (p.755). The sampling rate can be varied from 125 ksps to 1 Msps. The initialization and configuration instructions for the ADC module can be found on p.770 of the datasheet. Some key registers are described below:

SYSCTL_RCGCADC – to enable clock for ADC module, p.322
ADC0_PC – to select sampling rate, p.840 (use 125ksps)
ADC0_ACTSS – to select the sample sequencer 3, p.774
ADC0_EMUX – to set the trigger mode, p.785 (trigger by processor)
ADC0_SSMUX3 – to select the ADC channel, p.825 (select channel 0)
ADC0_SSCTL3 – to configure the sequencer clock, p.826 (disable TS0 D0, enable IE0 END0)

Once SYSCTL_RCGCADC is configured, a delay of 3-5 system clocks must be added for the clock to settle down. It is recommended to use two dummy operations as done before. The sample sequencer must be disabled before configuring. Once properly configured, it is enabled again. The ADC is triggered by writing 0x0008 to ADC0_PSSI register (p.796). When the conversion is done, bit 3 of ADC0_RIS register is set to "1" by the hardware. So, by polling this bit, we will know when the conversion is complete. After conversion, the 12-bit ADC result is available from ADC0_SSFIFO3 register (p.810).

It is important to note here that, the reference voltages for the ADC module can be adjusted using VREFP (pin 2 of MCU) and VREFN (pin 3 of MCU) – see p.762 of datasheet; however, these two pins cannot be accessed using the LaunchPad where they are internally tied to 3.3V and ground respectively (see p.21 in [1]).

Using the code template (provided on the course website), write a C program to repeatedly sample the voltage across the pot connected to port PE3. Vary the pot and observe the displayed output. Verify correct operation by creating a map of display output versus input voltage. Take five readings from the display and find the average quantization error.

Part 2 – Student's Own Design

The second part of this experiment allows for more advanced design using a microcontroller and the results of Part I. Students should select one of the following tasks develop their own procedure for completion and verification.

1. **Dimmer** - Use the internal pulse width modulator (PWM) to control the light intensity of a LED (i.e. simulating a dimmer action). The dimmer control voltage should come from the potentiometer used in Part 1. The 7-segment display should be used in the same manner to display the control voltage. The output pulse waveform and duty cycle are to be displayed on an oscilloscope. Hints: Connect the remaining potentiometer components of Fig. 2. Extend the program (written in part 1c) to use the internal PWM. See PWM information given in the Appendix.

   Once the program is compiled and downloaded into the MPU, vary the pot and observe the dimming action. Take measurements to verify correct system operation and the accuracy of your design.

2. **Warning Lamp** – This task is to produce a visible warning system for tank pressure. You are given a voltage developed by a pressure transducer (for this lab, use the potentiometer of Part Ic) and you are to flash a lamp at 0.5 Hz for normal operation, 1 Hz

for mild overpressure, 2 Hz for danger and 4 Hz for extreme danger. The duty cycle of the lamp should be 25% in each case. The nominal pressure transducer voltages for each of these cases are 0.5 V, 1.5 V, 2.5 V and 3.5 V. You should select appropriate voltage thresholds for your four cases. Develop a procedure to verify the accuracy.

3. **Remote Control** – The task is to remotely control the intensity of a lamp in four steps. Hint: You might use PWM or you might control current using two or more output pins. The remote control might be one of the following: USB keyboard, TV remote control, flashlight (laser pointer not recommended), sound control, or any other suitable device. Develop a procedure to assess the range and reliability of your design.
4. **Frequency Counter** – The task is to measure the frequency of a 0-5 V square wave input signal in the frequency range 100 Hz to 2000 Hz and use the two digit 7-segment display to indicate the frequency. The only requirement on the frequency display is that it be monotonic with input voltage. Develop a procedure to verify system operation and that the measurement is monotonic.
5. **Stepper Motor Control** – This task is to operate a stepper motor using a microcontroller. Seven 6-wire motors are available for use.
6. **UPC Bar Code Reader** – This task is to read the first digit of the manufacturer code and display on one 7-segment readout used in Part Ib. Some UPC bar code information can be found here http://www.morovia.com/education/symbology/upc-a.asp
7. **Student's Own Task** – Students are encouraged to propose their own experiment.

References

1. Stellaris® LM4F120 LaunchPad User Manual (SPMU289A–Revised December 2012): http://www.ti.com/tool/sw-ek-lm4f120xl
2. LM4F120H5QR Datasheet (Aug 29, 2012): http://www.ti.com/lit/gpn/lm4f120h5qr
3. Helpful resources: http://users.ece.utexas.edu/~valvano/arm/index.htm
4. http://processors.wiki.ti.com/index.php/Stellaris_LM4F120_LaunchPad_Unbox

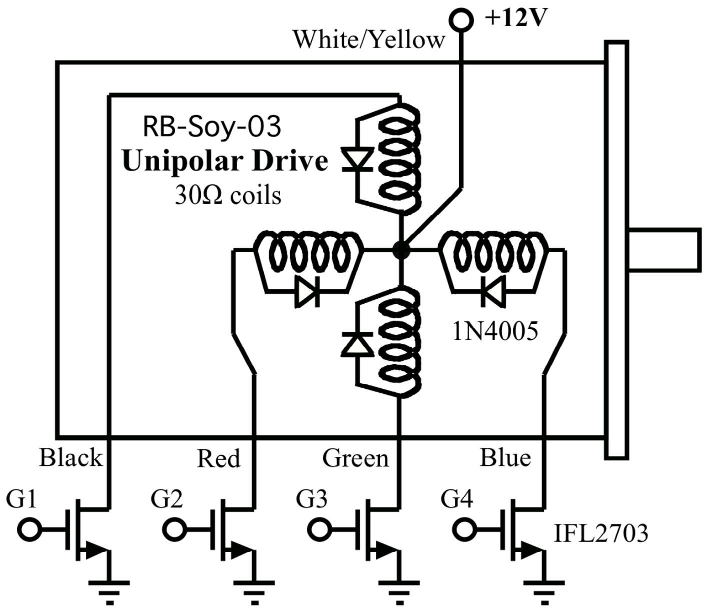
Appendix – Stepper Motor Information

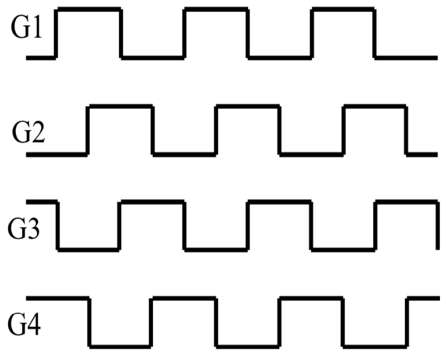
A stepper motor is used when precision control of movement is needed - the drive shaft rotates a precise number of degrees as current is applied to each coil in sequence. You can often salvage steppers from old printers or disk drives. The Department has seven RB-Soy-03 stepper motors with center-tapped windings that allow for Unipolar drive circuits. (see references for bipolar "H" Drive – no center tap but 8 transistors are required).

The microcontroller does not have sufficient output current capability to drive each coil, so use a transistor to drive each coil. The diagram shows IRF2703 FET transistors [2] and, since $V_{DS} < 0.2$ V for gate voltage greater than 3 volts, there is little heat dissipation in the transistors.

A "snubber" diode (sometimes called a freewheeling diode) is required when a coil is turned off. A large back-emf is generated by the collapsing magnetic flux in the coil when the supply current is suddenly reduced. This voltage could damage the transistor, so the diode provides a path for the current to briefly continue. The voltage is generated in the opposite direction from that needed to drive the coil, hence the diode is wired in "backwards". A convenient alternate drive method uses the UNL2803 Darlington transistor array that includes snubber diodes, Unfortunately, the high Darlington saturation voltage ($V_{CE} > 1.0$ V) results in excessive heat dissipation in the package.

There are 3 modes of driving the motor - see "Stepper Basics" in [3]. Illustrated below are gate control waveforms for the "standard" high torque regime. Can you change the direction of the motor? How would you change the speed of rotation?





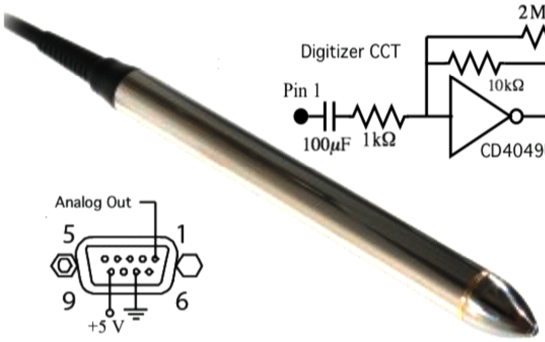
"Standard" Drive

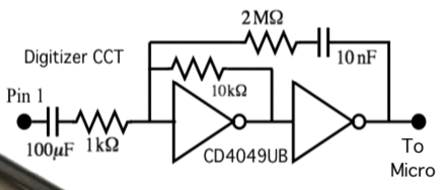
[1] http://www.robotshop.com/ca/rbsoy03-soyo-unipolar-stepper-motor.html
[2] http://www.datasheetarchive.com/IRL2703-datasheet.html#
[3] http://www.robotshop.com/ca/content/PDF/stepper-motor-documentation-27964.pdf

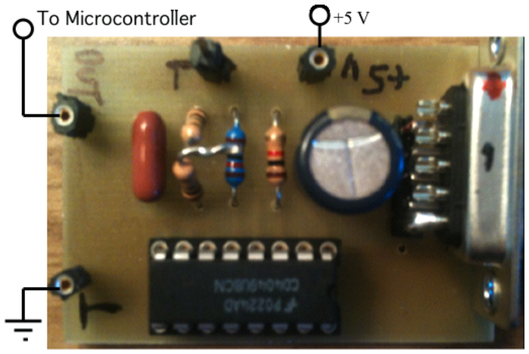
Appendix – UPC Bar Code Reader Information

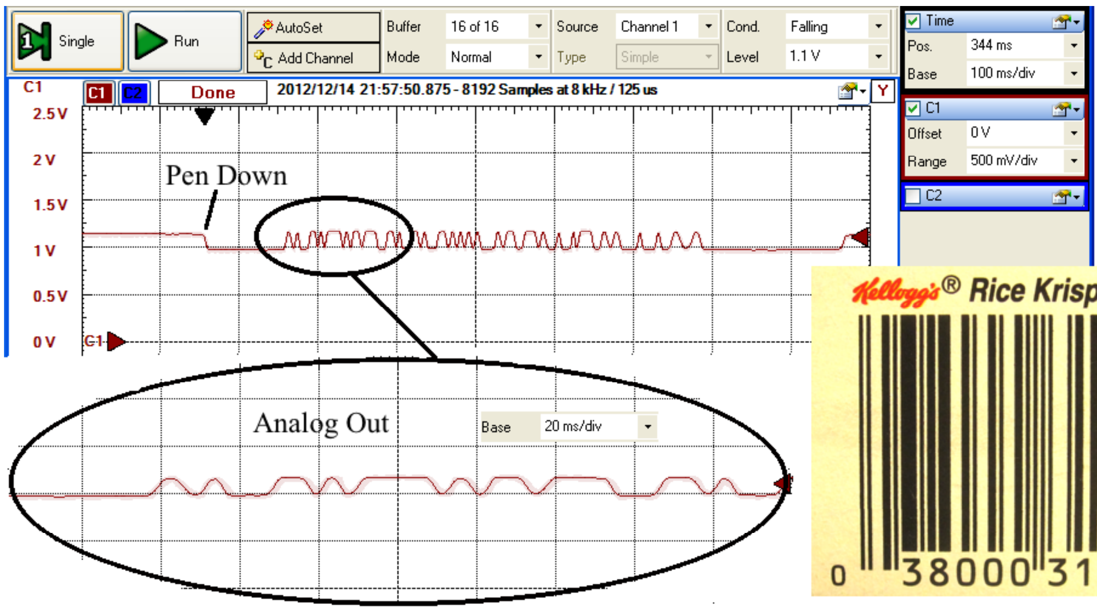
We have 5 scanner wands that use +5 volt supply and have analog output. An interface circuit amplifies and digitizes this analog output for use by the microcontroller. See images.

The UPC bars begin and end with "thin black, thin white, thin black". These narrow "guard" bars can be used to "train" the software. If the wand is moved at constant speed, the pulse period of these initial bars serves as a reference to measure the width of the remaining bars.









2012/12/14 21:57:50.875 - 8192 Samples at 8 kHz / 125 us

Buffer 16 of 16
Mode Normal
Source Channel 1
Type Simple
Cond. Falling
Level 1.1 V
Time
Pos. 344 ms
Base 100 ms/div
C1
Offset 0 V
Range 500 mV/div
C2
Base 20 ms/div

Information on UPC structure: http://www.morovia.com/education/symbology/upc-a.asp

Details on UPC code: http://www.wikihow.com/Read-12-Digit-UPC-Barcodes

Documents on a representative handheld UPC scanner wand can be found here:
http://www.currentdirections.com/hardware/honeywell/st6100.html

END