ECE271: Microcomputer Architecture and Applications — University of Maine

Lab #1: Joystick Button Input and LED Output

Week of 24 January 2022

Goals

1. Become familiar with the cross-development environment: (Keil μVision or Linux)
2. Write a C program that will run on the STM32L4 discovery board
3. Learn basics of GPIO configuration
4. Use GPIO for input (read from joystick button)
5. Use GPIO for output (display LED)
6. Learn about polling I/O

Grading Rubric for 271 Labs – Total of 20 points each

1. Pre-lab assignment (2 points)
2. Documentation and Maintainability (5 points)
3. Functionality and Correctness (5 points)
4. Lab Demonstration (5 points)
5. Something cool (3 points)

Pre-lab

1. Complete the pre-lab before attending lab. The pre-lab is in a separate pdf file, found on the website.

Lab Procedure

Using Chapter 14 of the text book as a reference, implement a C program that causes the red LED to light up when the joystick is pressed up, and the green LED to light up when the joystick is pressed down. Do this lab in three parts, as described below.

Part A – Lighting up the LEDs

1. There are two user-controllable LEDs (light-emitting diodes) on the STM32L4 discovery board. The red one is connected to GPIO-PB2 (GPIO Port B Pin 2) and the green one is connected to GPIO-PE8 (GPIO Port E Pin 8). You can see a schematic of how they are physically connected in Figure 1. The LEDs are surface mount and a bit hard to see; you can find their location on the board in Figure 2.

2. To turn on the LEDs you will do the following five steps:

   (a) Enable the clock for the corresponding GPIO ports (by default they are disabled).

   (b) Set the mode of the GPIO pins to be digital output (by default they are analog).

   (c) Set the push-pull/open-drain setting for the GPIO pins to push-pull.

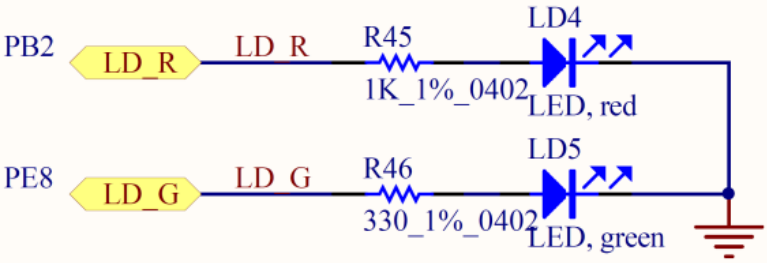

Figure 1: LED setup on the STM32L4

(d) Set the pull-up/pull-down setting for the GPIO pins to no pull-up/pull-down.

(e) Finally, set the output of the GPIO pin to have a value of 1 (corresponding to 3.3V)

3. For this lab we will program in C.

(a) First download the Lab1 template `ece271_lab1.zip`. This will be on the course website.

(b) See if you can build the template code. It should build, but not do anything yet.

i. Extract the template zip file somewhere you can find it.

ii. Start Keil μVision.

iii. Open the project from Keil, click on the uvproj file.

iv. Click on the "build" button to build. This is near the top left, and looks like a box with an arrow going into it. See Figure 3 if you have trouble finding the icon.

v. Now edit the `main.c` file in the editor. Modify the C as described below.

(c) Modify the `main.c` file to enable the registers you need to turn on the LEDs. You should have already calculated the values you need to write in the Pre-lab. If you want a reference for what you are programming see the "RM0351 Reference manual: STM32L4x5 and STM32L4x6 advanced Arm-based 32-bit MCUs"

You will use bitwise operations to set/clear the registers. Some of this might seem repetitive. Feel free to use function calls or other more advanced C methods if you feel like it helps.

i. Enable the GPIOA, GPIOB, and GPIOE blocks via the AHB2ENR register. You should have calculated the values for this in the pre-lab.

The provided `stm32l476xx.h` header file provides some helpers to make this easier. The memory-mapped I/O has been set up to point to the various structures with a volatile pointer.

For example, to set the GPIOCEN GPIO block C enable bit into the AHB2ENR register, you would do something like the following.

```
RCC->AHB2ENR |= (1<<2);
```

ii. Now set the GPIOB Pin 2 MODER output register to be the right value for output, as calculated in the prelab.

```
GPIOB->MODER
```

iii. Do the same for the push-pull setting in

```
GPIOB->OTYPER
```

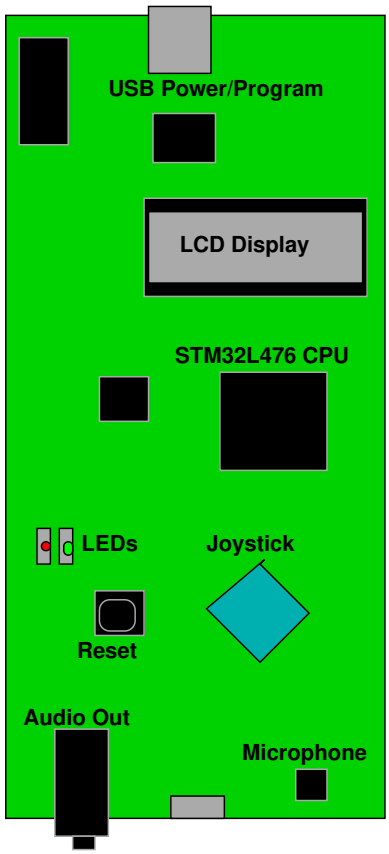

Figure 2: Rough diagram of components on STM32L4 board.

iv. Set the no-pull-up no pull-down setting in
`GPIOB->PUPDR`

v. Finally to actually enable/disable the LED we will need to set the proper bit in the output-data register
`GPIOB->ODR`
We didn't do this in the pre-lab, so to turn on GPIO on pin#X just set bit#X to 1.

vi. Repeat the above steps, but do them for GPIOE pin 8 for the green LED.

vii. At the end of your code put an infinite loop to keep the code from executing off the end of your program. This would look something like
`while(1) ;`

(d) While doing this, be sure to comment your code appropriately!

(e) Now compile/build your code.

i. Press the "build" button. Check the window at the bottom for warnings and especially for errors. The IDE will show little error icons by your code too if it detects any.

ii. Now plug the board into your laptop with the USB cable if you haven't already.

iii. Now download your code to the board by pressing the "download" button. Again see Figure 3 for guidance on what that looks like.

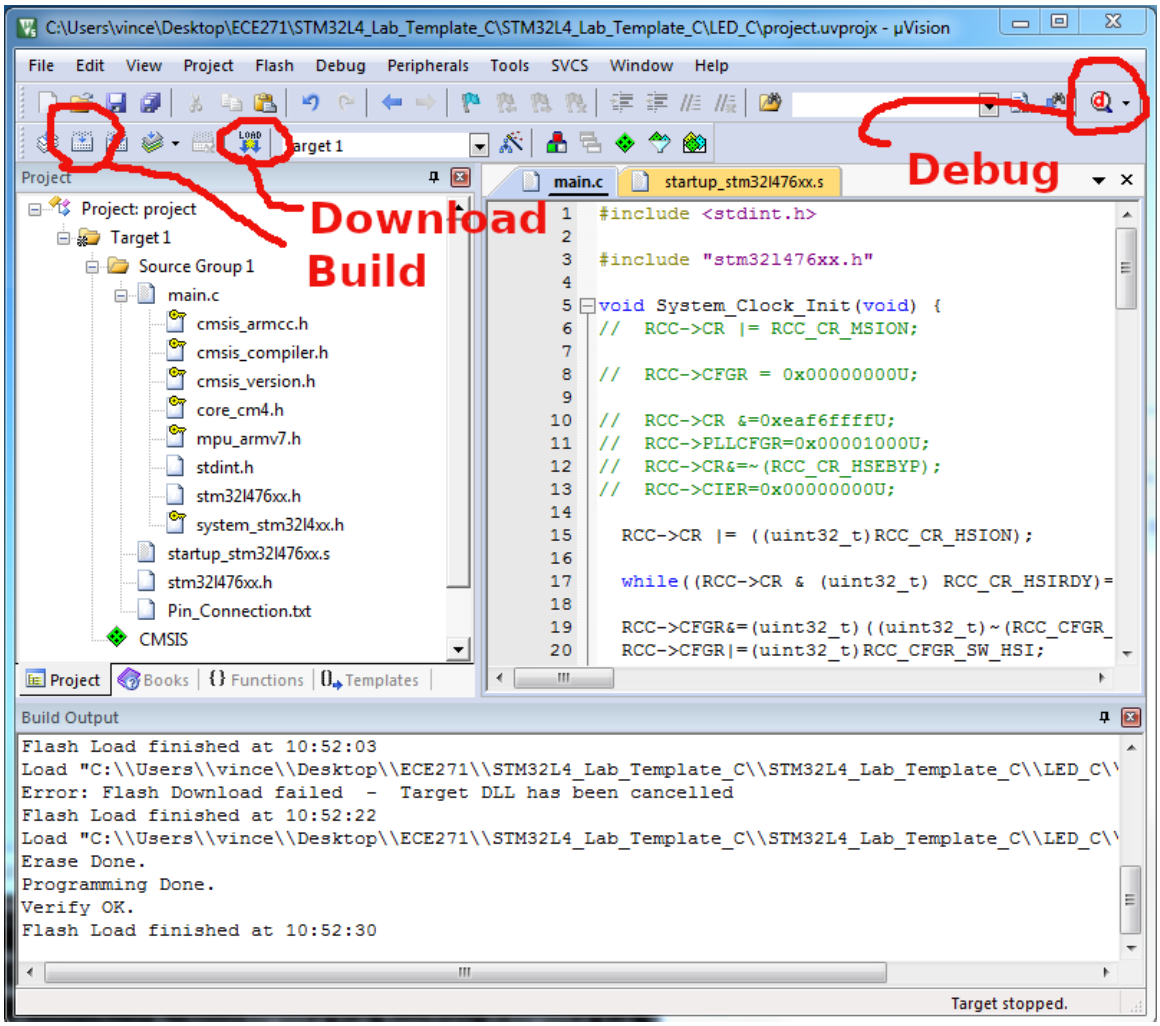

Figure 3: Useful things to click in μVision.

NOTE there is a demo program that comes with the board that might confuse Keil so it will refuse to program. There are a few ways to get around this. One that is known to work is described in Figure 4.

It is possible also if your code is buggy to confuse things so much that Keil can't program the board any more. In that case you might need to use the ST-Link program to zero out the memory so you can try again.

(f) If the download/flash went well, it's time to run your program

i. Click on the "Debug" button, which looks like a red magnifying glass. This will bring you into the debug section. To return to the build section just click on the red magnifying glass again.

ii. Click on the "Run" button. It's on the left and looks like a piece of paper with a down arrow next to it.

iii. If all goes well your program ran and your LEDs should light up!

iv. You can do some other interesting stuff with the debugger. You can reset to the beginning. You can single-step your code, which let's you see each step as it happens. You can monitor

memory, and variables, and even poke around with variables too.

(g) If your code isn't working you will have to debug it. The debugging options might help.

(h) If something goes so poorly that μVision won't flash your board anymore, you might have to use ST-Link to erase your board and the carefully step through the code to see what has gone wrong.

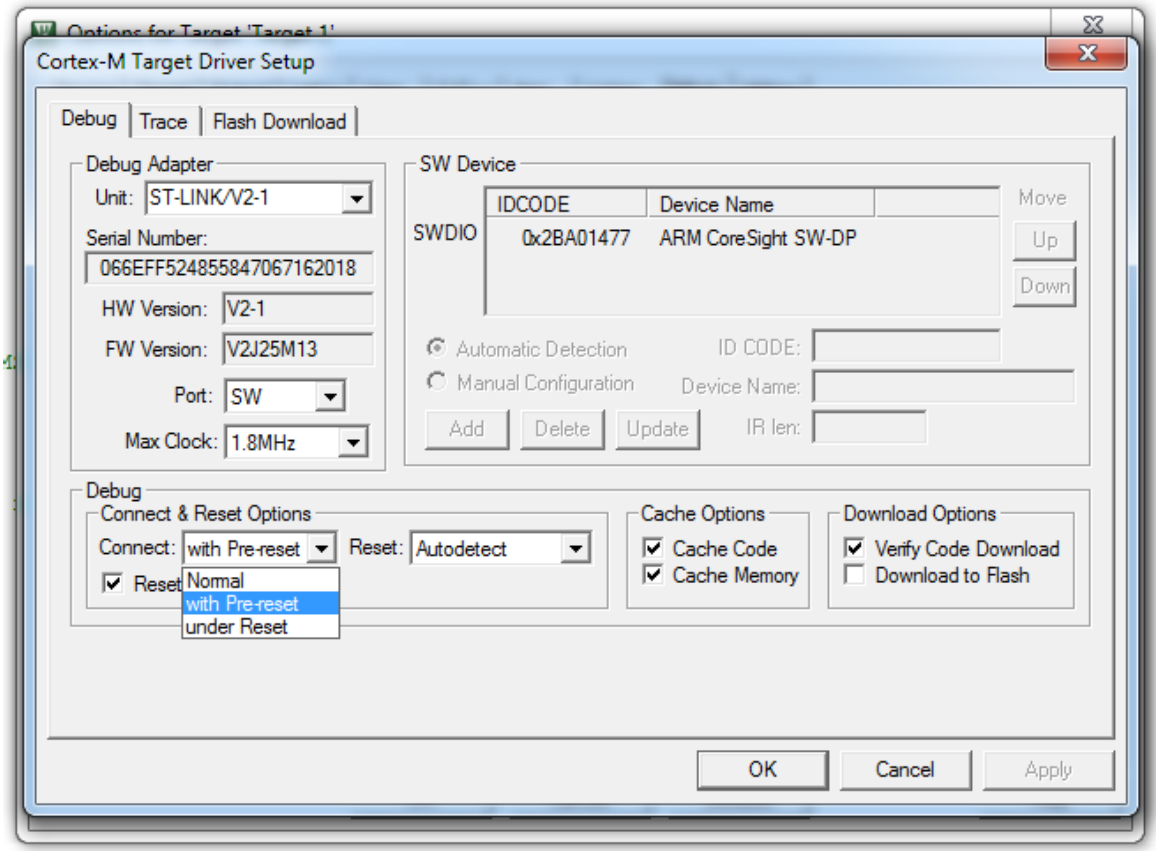

Figure 4: The "Target Not Found" error: When you program the STM32L4 board for the first time it might give this error, as the demo is running and interferes with the programming by setting the board into a low-power mode. One way to fix is in Keil to click "Project", "Options for Target", follow "Debug" and then "Settings" and change the connect from "normal" to "with pre-reset".

Part B – Joystick Buttons

1. There is a light-blue diamond shaped Joystick on the STM32L4 board (you can find its location in Figure 2). This consists of 5 buttons; the 4 directions as well as center when pressed. These buttons are connected to PA0, PA1, PA5, PA2, and PA3 as shown in Figures 5 and 6).

2. As you can see, there is a common input to the joystick which is pulled up to 3V. When any of the buttons are pressed this creates a connection to this 3V. These outputs are hooked to the various GPIO pins. There are resistors and capacitors providing hardware debouncing. Note that only the center has a pull down resistor; the others are left floating. If we want to read the output of the other buttons properly we will have to configure a GPIO internal pull-down resistor.

3. To read the state of the joystick you will have to do the following:

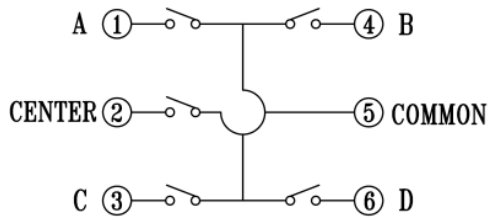

Figure 5: Joystick setup on the STM32L4

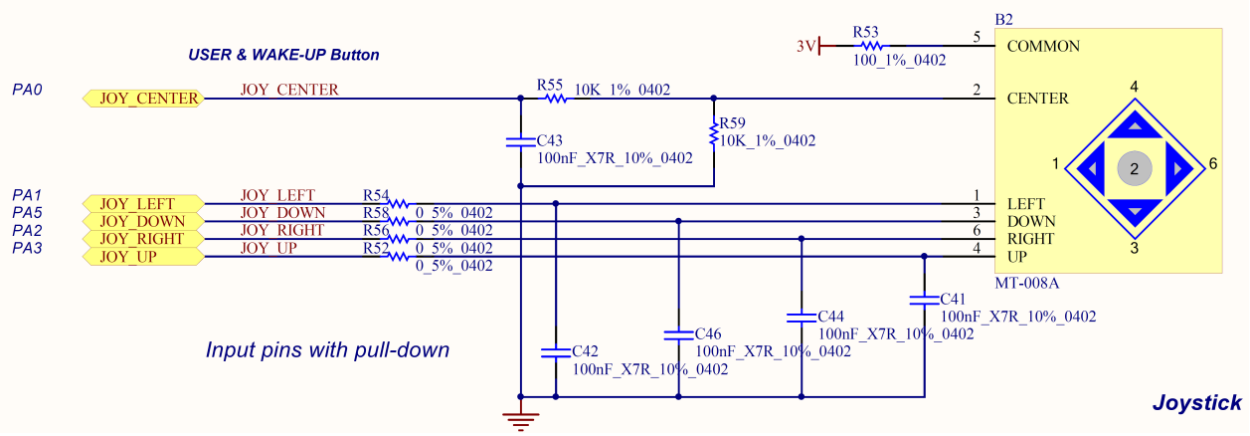

Figure 6: Joystick connections. Note that capacitors and resistors are there to provide **hardware denouncing**. Also note that only the center button has a pull-down resistor connected.

(a) Enable the clock for the corresponding GPIO ports (by default they are disabled).

(b) Set the mode of the GPIO pins to be input (by default they are analog).

(c) Set the pull-up/pull-down setting for the GPIO pins to have a pull-down.

(d) Finally, read the status of the corresponding GPIO pin.

4. You will do the above in C.

(a) Modify your code to have an infinite loop. Each time through the loop read the status of the UP and DOWN GPIOs. If UP is high, then light the red LED, otherwise turn it off. If DOWN is high, then light the green LED. Otherwise turn it off.

i. You should have already enabled GPIO bank A in Part A of the lab. If you didn't, make sure your code properly sets the enable bit.

ii. Now enable the joystick GPIO pins so that the proper pins in MODER are set to be inputs.
`GPIOA->MODER`

iii. Set each joystick GPIO to be pull-down in the PUPDR register.

iv. Finally to read the value read the corresponding bit in the IDR (input-data) register. To read the status on GPIO pin#X just check if bit#X is 1 or 0. You can use a bitwise and instruction with a proper mask to check this.

(b) Once everything goes well, the two LEDs should be light when the button is pressed. If it doesn't work you will have to debug your code to find out what is wrong.

Part C – Something Cool

Do something cool! You can come up with something on your own, but here is a list of ideas you can use.

1. Have the LEDs start blinking when you press the joystick button and stop when you press it again.
2. Write a program that sends a Morse Code message using one of the LED. You will have to research how to create a delay in C.
3. Use an oscilloscope to show the voltage on the RED led and the voltage on the output of the up joystick button. Find the latency (how long it takes) between the button being pressed and the LED lighting up.
4. Use the software logic analyzer provided by the MDK-KEIL software to analyze the input and output signals.
5. Change the output clock speed and use an oscilloscope to see how that affects the output speed.

Table 1: STM32L4 board pin connections.

Peripheral	Purpose	Pin	Peripheral	Purpose	Pin
Joystick MT-008A	Center	PA0	LCD	VLCD	PC3
	Left	PA1		COM0	PA8
	Right	PA2		COM1	PA9
	Up	PA3		COM2	PA10
	Down	PA5		COM3	PB9
User LEDs	LD4 Red	PB2		SEG0	PA7
	LD5 Green	PE8		SEG1	PC5
CS43L22 Audio DAC i2c 0x94	SAI1_MCK	PE2		SEG2	PB1
	SAI1_FS	PE4		SEG3	PB13
	SAI1_SCK	PE5		SEG4	PB15
	SAI1_SD	PE6		SEG5	PD9
	I2C1_SCL	PB6		SEG6	PD11
	I2C1_SDA	PB7		SEG7	PD13
	Audio_RST	PE3		SEG8	PD15
MP34DT01 MEMS MIC	Audio_DIN	PE7		SEG9	PC7
	Audio_CLK	PE9		SEG10	PA15
LSM303C eCompass	MAG_CS	PC0		SEG11	PB4
	MAG_INT	PC1		SEG12	PB5
	MAG_DRDY	PC2		SEG13	PC8
	MEMS_SCK	PD1		SEG14	PC6
	MEMS_MOSI	PD4		SEG15	PD14
	XL_CS	PE0		SEG16	PD12
	XL_INT	PE1		SEG17	PD10
L3GD20 Gyro	MEMS_SCK	PD1		SEG18	PD8
	MEMS_MOSI	PD4		SEG19	PB14
	MEMS_MISO	PD3		SEG20	PB12
	GYRO_CS	PD7		SEG21	PB0
	GYRO_INT1	PD2		SEG22	PC4
	GYRO_INT2	PB8		SEG23	PA6
ST-Link V2	USART_TX	PD5	USB OTG	OTG_Pwr_On	PC9
	USART_RX	PD6		OTG_FS_OvrCurrent	PC10
	SWDIO	PA13		OTG_FS_VBUS	PC11
	SWCLK	PA14		OTG_FS_ID	PC12
	SWO	PB3		OTG_FS_DM	PA11
	3V3_REG_ON	PB3?		OTG_FS_DP	PA12
Quad SPI Flash Memory	QSPI_CLK	PE10	Clock	OSC32_IN	PC14
	QSPI_CS	PE11		OSC32_OUT	PC15
	QSPI_D0	PE12		OSC_IN	PH0
	QSPI_D1	PE13		OSC_OUT	PH1
	QSPI_D2	PE14			
	QSPI_D3	PE15			

Lab Demo

Student Name:_____ TA:_____ Date:_____

1. Submit your code
    - Complete a README with the post-lab (next page) answers.
    - Make sure the code is properly commented.
    - Submit your code via brightspace.
2. Demo your implementation to your lab TA.
3. Answer the following questions and show to the TA.
    - Why did we configure the LED pins to be push-pull rather than open-drain?
    - What is GPIO output speed? What is the default speed? Did changing speeds affect the lab?

Post-Lab

- Place your answers to the question in a file `Readme.md`
- In the future we will submit with git, but for this first lab please submit your code and the readme file via brightspace.

1. The joystick buttons on the STM32L4 board have hardware debouncing. An example of this can be seen in Figure 5. Explain briefly how this works.
2. Debouncing can also be done in software. Explain how this could be done in software.
3. Each GPIO in has for programmable output speeds. Low, medium, fast, and high. The slew rate can be up to 80MHz. Why is the "low" speed recommended for controlling LEDs? (Hint: energy and electromagnetic interference)