ECE177: Programming I: From C...

Lab #5 — Larson Scanner

Week of 2 March 2026

1 Objectives

- Complete all of the steps outlined in the comments of `kitt.c`
- Program the Pico breadboard with the modified `kitt.c` file and confirm working output

2 Materials

- Assembled and working Pico breadboard
- Laptop
- Laptop charger

3 Procedure

1. Make sure to read all of the directions carefully before getting started.
2. Download the `lab05_code.zip` file here:
   `https://web.eece.maine.edu/~vweaver/classes/ece177/labs/lab05_code.zip`
3. Extract the zip file in your `Documents/ece177/labs/` directory that you created in lab01.
4. In VS Code, use the Raspberry Pi Pico Extension to import the `lab05_code` folder as a project.

4 Modify the Provided Code

1. Edit the provided `kitt.c` file
2. Below each task is briefly described, there are more details in the comments of the code on what you need to do
3. Make sure to read all of the comments carefully before getting started so as to not miss any of the instructions.
4. Be sure to comment your code!
5. Test your code often! Don't try to write it all at once!
6. As with Lab#4 ideally you can just press the VS Code "Run" button and it will compile and upload your program to the Pico. If it doesn't you might have to find the `kitt.uf2` file in the `build` subdirectory and copy it over to the Pi Pico virtual drive.

Section A: Update the Code Comments

(a) Update the code comments at the top of the file with your name, the date, and a brief description of the lab.

Section B: Make the code a valid C program

(a) First add a proper `int main()` declaration in Section B near the start of the file.

(b) When it becomes necessary, add any needed variables right after the `main()` function you created

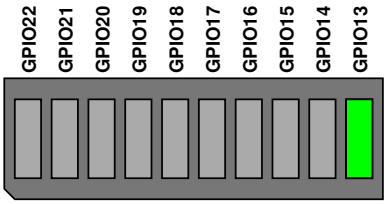
(c) Skip to the end of the file and put do the following:

i. Add an infinite loop that calls `jump_to_MSD()` (this enables the "Press "*" and "D" to reprogram feature)

ii. Add a proper `return` statement at the end

iii. Don't forget a closing curly brace.

Section C: Enable the right-most LED (GPIO13)



(a) Initialize the GPIO13 line. We provide a routine called `gpio_init()` that takes one argument, the GPIO number. For example, to initialize GPIO26 you would do `gpio_init(26);` Do something similar to initialize GPIO13.

(b) Clear out GPIO13 to 0. We want to reset the line to no voltage before we start making other changes to this. We provide a routine called `gpio_put()` that can do this. It takes two arguments, the GPIO number and an output value 0 or 1 (0 means 0 Volts, 1 means 3.3 volts). For example, to set GPIO26 to 3.3V you would do `gpio_put(26,1);`. Add code to the program that sets GPIO13 to 0.

(c) Set GPIO13 to be an output. You can set GPIO pins to have various functions, including INPUT and OUTPUT. For this lab we want output. We provide a function called `gpio_set_dir()` that takes two arguments, the GPIO number and whether it's INPUT (`GPIO_IN`) or OUTPUT (`GPIO_OUT`). Add code to the program that sets GPIO13 to `GPIO_OUT`.

(d) Now turn on the GPIO13 pin to light up the LED. Use `gpio_put()` like before, but this time set GPIO13 to the value 1.

(e) Next have your program sleep for 1 second. You will remember from last lab the way to do this is to use the sleep-millisecond function `sleep_ms();`

(f) Now turn on the GPIO13 pin to turn off the LED. Use `gpio_put()` like before, but this time set GPIO13 to the value 0.

(g) After doing this compile and run your code and fix any bugs. You should be able to switch back to program mode by holding "*" and "D" like last lab.

Section D: Enable two LEDs using a bitmap

(a) First, comment out all of your code in Section C

(b) Next, we're going to enable all 10 GPIOs for the LED bargraph.

i. Use a for loop, and iterate from GPIO13 to GPIO22 (inclusive).

ii. If you add new variables (for example loop iterators) add them to the top of the file after `main()`

iii. Reminder, you'll want to make the following calls for each of the GPIOs, just like you did in Section C

```
gpio_init();
gpio_put();
gpio_set_dir();
```

(c) Turn on both GPIO13 and GPIO22 at once by using bit manipulations

i. As described in class, the GPIOs can be accessed by an MMIO register. A struct pointer is set up so that this register can be accessed via a normal C struct access

ii. To set the GPIOs in this struct just read or write from the struct field: `sio_hw->gpio_out`

iii. In this struct, the gpio_out field is a 32-bit integer. The high (biggest, left-most bit) maps to GPIO31 and the lowest (smallest, right-most bit) corresponds to GPIO0.
TODO: diagram

iv. Use this `sio_hw->gpio_out` variable to turn on GPIO13 and GPIO22

A. NOTE: you cannot just assign a value to it, like `sio_hw->gpio_out=(1<<13);` This will set bit 13 to 1, but it will also force all the other bits to 0 and this can break things if they held important values.

B. You will need to read/modify/write the value. One way to do this is read `sio_hw->gpio_out` into a temporary variable, set the bit you want with BITWISE OR, then write the temporary variable back to `sio_hw->gpio_out`

C. If you are skilled at C you can combine all those steps into one statement, but you don't have to for this lab.

(d) Delay for 1 second using `sleep_ms()`

(e) Turn off both GPIO13 and GPIO22 at once by using bit manipulations

i. Use the `sio_hw->gpio_out` variable to turn off GPIO13 and GPIO22

A. NOTE: you cannot just assign 0 to it, like `sio_hw->gpio_out=0;` This will force all the bits to 0 and this can break things if they held important values.

B. You will need to "mask" off the bits you want to clear. To do this you want to use BITWISE AND and BITWISE NOT. If you remember your truth tables, if you BITWISE AND, if your mask variable has a 1, then the value stays the same, but if your mask has a 0 it will clear it to 0. So to clear a particular bit you want to AND it with a value that is ALL ONES except for the one bit you want to clear which is ZERO. One way to create a mask like that is something like `mask=~(1<<13);` which sets bit 13 then flips all bits.

C. You will need to read/modify/write the value. One way to do this is read `sio_hw->gpio_out` into a temporary variable, mask off the bits with BITWISE AND, then write the temporary variable back to `sio_hw->gpio_out`

D. If you are skilled at C you can combine all those steps into one statement, but you don't have to for this lab.

(f) Delay for 1 second using `sleep_ms()`

(g) Compile and test your code. Do not continue until your code is working.

Section E: Animate the LEDs

(a) Now we will implement the "Larson Scanner" where we have the bargraph LEDs make a right to left then left to right pattern.
TODO: example image
You will have one loop that will shift the LED to the left, lighting GPIO13, GPIO14, GPIO15, ..., GPIO22 in turn. Then there will be a second loop that will start with GPIO22, GPIO21, GPIO20, .., down to GPIO13.

(b) You will add this code *inside* the existing infinite loop at the end of the program that currently has the `jump_to_MSD()` in it.

i. Leave the call to `jump_to_MSD()` inside the loop and put all the following code immediately following it (also inside the infinite loop)

ii. Use a new variable of 32 bits, initialized so that bit 13 is 1. All other bits shall be zero. This will be used to turn on and off the correct LED.

iii. We will first shift the LED left. Create a for loop with the appropriate number of iterations to cycle through the 10 LEDs. This loop should:

A. Turn on the LED using bitwise OR with your state variable.

B. Delay for 100ms

C. Turn off the LED using bitwise AND masking with your state variable

D. Shift your state variable once to the left

iv. After this loop, have another that does the same as above, but starts with GPIO22 being set and the shifts to the right each time until it gets back down to GPIO13.

(c) Run the code on the Pico, and you should get an animated LED pattern that runs forever.

5 Grading/Checkoff

1. Upload your `kitt.c` file to BrightSpace.

2. Upload a picture of your board running the code at the end showing some of the bargraph LED lit up.

3. TAs will check the output of your final program running on your hardware. They will ask you to demonstrate the lab requirements and will enter grades after the requested information has been uploaded on the assignment rubric.

4. TAs: Enter grades only after verifying full functionality of the code running, the requested information has been uploaded, and the assignment has been submitted.