ECE 1181: Microcontrollers and Embedded Systems Lab
Spring 2024 – Professor Joe Camp

Lab 9: General Purpose Input/Output (GPIO) Pins (4/18 or 4/19)

Hackathon Assignment (see end of document - due by end of lab 4/25 or 4/26)

Kits due by end of lab 11/16 or 11/17 – 20% penalty on Hackathon per day late

Reminder: Each lab report should have the following.
1. Name and SMU ID
2. Lab Session number & objective of the lab assignment
3. Description of the program and if writing new code, algorithm in pseudocode (25%)
4. The actual code used comments (25%)
5. Answers to any questions on the lab description, results, screenshots, and verification (50%)

- *Reports must be submitted before the start of the next lab.*

Description:

We will now start to interface the Red Pitaya. You will begin by working with output (3 LEDs as pictured in Figure 1). To do so, you will need to connect LEDs, resistors, and jumpers to and from General Purpose Input/Output (GPIO) pins on the Red Pitaya. One end of the jumpers (resistor side) will go into the programmable GPIO pins. The other end (LED side) will go into a ground pin of the Red Pitaya. You can verify your circuitry by taking the jumper connected to the programmable GPIO pins and hard wiring it to the 5v pins of the Red Pitaya. This will simply turn on the LED if everything is correct in the circuit. Then, you can return the jumper to the respective programmable GPIO pins and start the programming tasks that progresses from changing files to creating a countdown of your loop control variable on the LEDs. Lastly, there is a bonus of using a button and counting the number of pushes of that button.

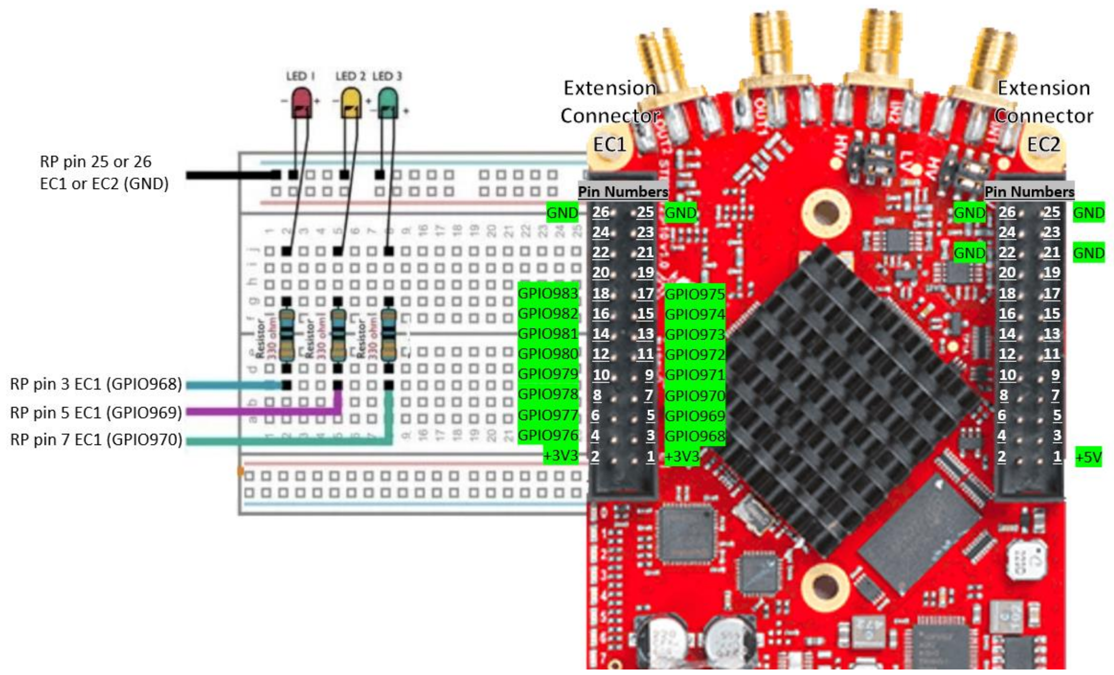

Figure 1: The layout of the LEDs, resistors, and Red Pitaya GPIO pins
(source: https://redpitaya.readthedocs.io/en/latest/developerGuide/hardware/125-14/extent.html).

For this lab, we will be following Chapter 8 on Programming GPIO Pins fairly closely. To get the correct pin numbers, you will need to refer to the above figure. However, before we program the steps that the chapter outlines, we first need to verify that our circuit configuration is correct. In particular, the lab consists of the following steps:

1. Wire your breadboard to your Raspberry Pi according to the picture in Figure 1 above.
2. Now, one by one take the connector out of GPIO pins 968, 969, and 970 on the Red Pitaya and connect it to a 5v source (Red Pitaya pin 1 on extension connector 2). If the LED and resistor wiring is correct, each LED will light up when you connect it to the end of the jumper that was previously in GPIO pins 968, 969, and 970 into the 5v (Red Pitaya pin 1 on extension connector 2).
3. In order to use the Linux gpio api, we need to use the appropriate fpga bit-stream (logic analyser bitstream is the default one on boot). In order to apply this, simply run this command in the command line:
   ```
   cat /opt/redpitaya/fpga/classic/fpga.bit > /dev/xdevcfg
   ```
4. Read through pages 145-147 of Chapter 8. As you read, be aware that the pin numbers listed in the book are not associated with the Red Pitaya device we are using. Refer to the above Figure 1 for the correct pin numbers.
5. Based on what you just read, modify the */sys/class/gpio/gpio968/direction* and */sys/class/gpio/gpio968/value* pins to make the LED that is connected to GPIO pin 968 to light up. Repeat this for GPIO pin 969 and GPIO pin 970.
6. Once the bulbs are all lit, you are now ready to proceed to reading through pages 148-152. You will need to create the file gpiomacros.s from the modified versions of Listing 8-1 below and the file main.s from the modified version of Listing 8-2 below. You will also need the modified fileio.s from the last lab (Lab 8: Modified Listing 7-1).
7. Create a makefile (you only need main.s to assemble and link because it will pull the other files it needs in the include statements).
8. Once your program builds successfully, you can now run your resulting executable file. You may need to run this with "sudo" out front of the executable command to ensure you have the appropriate administrative privileges.
9. Change the .data field in gpiomacros.s to add 1 second so that you can see that the lights are changing slowly. Depending on your makefile, you may need to make a small change to main.s (e.g., add an extra line at the top of the source file and resave) to make the "make" reassemble your program.
10. Once you have confirmed this longer time interval between the lights changing, change the main program to reflect the value of r6 in binary on

the LEDs and start it counting down from 7 instead of 10. In other words, make one LED the 1's digit (2^0), one LED the 2's digit (2^1), and one LED the 4's digit (2^2).

11. Show your program working over Zoom to your TA or take a video of your working program and include it in a zip file of your report submission. Also, include your code in the zip file.
12. Read carefully and follow the directions on pages 158-164.
13. Once your program builds successfully, you can now run your resulting executable file. You will need to run this with "sudo" out front of the executable command to ensure you have the appropriate administrative privileges.
14. Change the .data field in gpiomem.s to add 1 second so that you can see that the lights are changing slowly. Depending on your makefile, you may need to make a small change to main.s (e.g., add an extra line at the top of the source file and resave) to make the "make" reassemble your program.
15. Once you have confirmed this longer time interval between the lights changing, change the main program to reflect the value of r6 in binary on the LEDs and start it counting down from 7 instead of 10. In other words, make one LED the 1's digit (2^0), one LED the 2's digit (2^1), and one LED the 4's digit (2^2).
16. Show your program working over Zoom to your TA or take a video of your working program and include it in a zip file of your report submission. Also, include your code in the zip file.
17. **BONUS (+10):** Choose your favorite design from either the Linux system calls approach or the memory-mapped approach and create a counter that shows up as a binary number on your LEDs each time a button is pushed. You will have to wire the button to another GPIO port in a similar fashion as the LEDs pictured in Figure 1. Then, you will need to modify the code to have another pin that is setup as an input for the GPIO port that you connected to the button.

Code Given: Listings 7-1 (fileio.s from Ch. 7, Lab 8), 8-1 (Modified below), 8-2 (Modified below), 8-3, and 8-4.

Modified Listing 8-1 (In order to fit the changed lines in bold and red):

```
@ Various macros to access the GPIO pins
@ on the Raspberry Pi.
@
@ R8 - file descriptor.
@

.include "fileio.s"

@ Macro nanoSleep to sleep .1 second
```

ECE 1181: Microcontrollers and Embedded Systems Lab
Spring 2024 – Professor Joe Camp

```
@ Calls Linux nanosleep service which is funct 162.
@ Pass a reference to a timespec in both r0 and r1
@ First is input time to sleep in secs and nanosecs.
@ Second is time left to sleep if interrupted
.macro nanoSleep
        ldr     r0, =timespecsec
        ldr     r1, =timespecsec
        mov     r7, #sys_nanosleep
        svc     0
.endm
.macro GPIOExport   pin
        openFile        gpioexp, O_WRONLY
        mov             r8, r0          @ save the file desc
        writeFile       r8, \pin, #2

        flushClose      r8
.endm
.macro GPIODirectionOut pin
        @ copy pin into filename pattern
        ldr             r1, =\pin
        ldr             r2, =gpiopinfile
        add             r2, #20
        ldrb            r3, [r1], #1 @ load pin and post incr
        strb            r3, [r2], #1 @ store to filename and post incr
        ldrh            r3, [r1]
        strh            r3, [r2]
        openFile        gpiopinfile, O_WRONLY
        mov             r8, r0          @ save the file descriptor
        writeFile       r8, outstr, #3
        flushClose      r8
.endm
.macro GPIOWrite pin, value
        @ copy pin into filename pattern
        ldr             r1, =\pin
        ldr             r2, =gpiovaluefile
        add             r2, #20
        ldrb            r3, [r1], #1    @ load pin and post increment
        strb            r3, [r2], #1    @ store to filename and post increment
        ldrh            r3, [r1]
        strh            r3, [r2]
        openFile        gpiovaluefile, O_WRONLY
        mov             r8, r0   @ save the file descriptor
        writeFile       r8, \value, #1
        flushClose      r8
.endm
```

```
.data
timespecsec:        .word 0
timespecnano:       .word 100000000
gpioexp:       .asciz "/sys/class/gpio/export"
gpiopinfile:   .asciz "/sys/class/gpio/gpioxxx/direction"
gpiovaluefile: .asciz "/sys/class/gpio/gpioxxx/value"
outstr:        .asciz "out"
               .align 2 @ save users of this file having to do this.
```

Modified Listing 8-2 (Here the pin numbers simply need to be changed to the correct ones that match the Red Pitaya GPIO index. Changed lines in bold and red):

```
@
@ Assembler program to flash three LEDs connected to
@ the Raspberry Pi GPIO port.
@
@ r6 - loop variable to flash lights 10 times
@
.include "gpiomacros.s"

.global _start @ Provide program starting address to linker

_start: GPIOExport pin968
        GPIOExport pin969
        GPIOExport pin970
        nanoSleep

        GPIODirectionOut pin968
        GPIODirectionOut pin969
        GPIODirectionOut pin970
        @ set up a loop counter for 10 iterations
        mov    r6, #10

loop:   GPIOWrite pin968, high
        nanoSleep
        GPIOWrite pin968, low
        GPIOWrite pin969, high
        nanoSleep
        GPIOWrite pin969, low
        GPIOWrite pin970, high
        nanoSleep
        GPIOWrite pin970, low
        @decrement loop counter and see if we loop
@ Subtract 1 from loop register
@ setting status register.
        subs r6, #1
@ If we haven't counted down to 0 then loop
        bne loop

_end:   mov R0, #0 @ Use 0 return code
        mov R7, #1 @ Command code 1 terminates
        svc 0 @ Linux command to terminate

pin970: .asciz "968"
pin969: .asciz "969"
pin968: .asciz "970"
low: .asciz "0"
high: .asciz "1"
```

You have completed this lab when:

Demonstrate to your lab instructor that you have completed the steps above.

Hackathon Portion: You may choose to do the push buttons (#10 above) for a Hackathon grade instead of the +10 bonus for the lab or you can write code that will address areas of confusion for you. If you do both, you can use the latter as your Hackathon grade (max 100%) and have the +10 for your lab grade.

The Hackathon document should be separate from your lab description and include:
1. Name and SMU ID
2. Lab Session number & objective of the lab assignment
3. Description of the areas of weakness (50%)
4. Code used, screen shot of it running, and a discussion of why you wrote it, what it does, and what it taught you (2% per line that has all of these items)

- *This report is unique in that it should be submitted by the end of this week's lab session so that the students, TAs, and professor can all prepare for the following week's mid-term exam.*
- *Also, note that attendance in days of class (Tuesday 4/18 and Thursday 4/20) may count for up to 10% of this Hackathon grade per day attended*