Lab 8: Fixed point Conversion

Victor Yuan

Microcontrollers and Embedded Systems

March 30, 2017

Introduction:

The goal of this lab was to learn how GPIO pinouts on a Raspberry Pi works.

Procedure:

Access SMU canvas on the Raspberry PI download the GPIO.TGZ file. Go to CMD prompt and locate the file. Use $ tar xfz GPIO.tgz to extract the file. Then open up codeblocks in developers mode with the command sudo codeblock. Import the files extracted from GPIO then wire the breadboard, LEDs and switches according to circuit diagram. GPIO 23-24 for the first 3 switches all connected to the ground. With GPIO 21, 18, 17 connected to the LEDs thru a 560 ohm resistor with a 3.3v voltage source.

Code:

The Code below was given in the GPIO.tgz file

```
@@@ Raspberry Pi version
    .equ    SHORT_DELAY,    50
    @@  These are the offsets for the GPIO registers that we will use
    .equ    GPFSEL0,    0x0000
    .equ    GPFSEL1,    0x0004
    .equ    GPFSEL2,    0x0008
    .equ    GPFSEL3,    0x000C
    .equ    GPFSEL4,    0x0010
    .equ    GPFSEL5,    0x0014
    .equ    GPSET0,     0x001C
    .equ    GPSET1,     0x0020
    .equ    GPCLR0,     0x0028
    .equ    GPCLR1,     0x002c
    .equ    GPLEV0,     0x0034

    .equ    GPPUD,      0x0094
    .equ    GPPUDCLK0,  0x0098
    .equ    GPPUDCLK1,  0x009C
```

```
@@  .data
@@ pofmt:   .asciz  "register offset is %08X and shift is %d\n"
    .text

@@@ enable/disable the internal pullup/pulldown resistors
@@@ set_pud(gpio_pin_num, direction)
@@@ direction: 0=float, 1=pull down, 2=pull up
set_pud:stmfd   sp!,{r0-r3}
    @@ Load the Base Address of the GPIO device
    ldr r3,=gpiobase    @ load pointer to the address of the gpio device
    ldr r3,[r3]     @ load address of the gpio device
    @@ Enable the internal pull-up resistors on button pins
    @@ Store direction in GPPUD
    str r1,[r3,#GPPUD]
    mov r2,#SHORT_DELAY
w1: subs    r2,r2,#1
    bcc w1
    @@ Write to GPPUDCLK0 to clock the signals into GPIO 22,23,24
    mov r2,#1
    lsl r2,r2,r0
    str r2,[r3,#GPPUDCLK0]
    mov r2,#SHORT_DELAY
w2: subs    r2,r2,#1
    bcc w2
    @@ Write to GPPUD to remove the control signal
    mov r2,#0
    str r2,[r3,#GPPUD]
    @@ Write to GPPUDCLK0/1 to remove the clock
    @@ Write to GPPUDCLK0 to clock the signals into GPIO 22,23,24
    mov r2,#0
    str r2,[r3,#GPPUDCLK0]
    ldmfd   sp!,{r0-r3}
    mov pc,lr
```

```
@@@ pin_offest finds the offset to the correct GPFSEL register
@@@ it returns the offset in r3, and the shift amount in r0
pin_offset:
    subs    r0,r0,#10   @ pins 0-9 are in GPFSEL0
    movlt   r3,#GPFSEL0
    blt have_offset
    subs    r0,r0,#10   @ pins 10-19 are in GPFSEL1
    movlt   r3,#GPFSEL1
    blt have_offset
    subs    r0,r0,#10   @ pins 20-29 are in GPFSEL2
    movlt   r3,#GPFSEL2
    blt have_offset
    subs    r0,r0,#10   @ pins 30-39 are in GPFSEL3
    movlt   r3,#GPFSEL3
    blt have_offset
    subs    r0,r0,#10   @ pins 40-49 are in GPFSEL4
    movlt   r3,#GPFSEL4
    blt have_offset
    sub r0,r0,#10   @ pins 40-49 are in GPFSEL5
    mov r3,#GPFSEL5
have_offset:
    add r0,r0,#10   @ make 0<=r0<=9
    add r0,r0,r0,asl #1 @ r0 *= 3

    @@ stmfd    sp!,{r0-r3,lr}
    @@ mov  r2,r0
    @@ mov  r1,r3
    @@ ldr  r0,=pofmt
    @@ bl   printf
    @@ ldmfd    sp!,{r0-r3,lr}

    mov pc,lr
```

```
@@@ gpio_dir_input sets the pin specified in r0 to be an input pin
gpio_dir_input:
    stmfd   sp!,{lr}
    @@ Load the Base Address of the GPIO device
    ldr r1,=gpiobase    @ load pointer to the address of the gpio device
    ldr r1,[r1]     @ load address of the gpio device
    bl  pin_offset  @ get the offset in r3 and shift in r0
    add r1,r1,r3    @ get pointer to the FPFSEL register
    ldr r3,[r1]     @ read the FPFSEL register
    mov r2,#7       @ create bit mask for 3 bits we need to clear
    lsl r2,r2,r0
    mvn r2,r2
    and r2,r2,r3    @ clear the 3 bits
    str r2,[r1]     @ write the FPFSEL register
    str r2,[r1]     @ write the FPFSEL register
    ldmfd   sp!,{pc}

@@@ gpio_dir_output sets the pin specified in r0 to be an output pin
gpio_dir_output:
    stmfd   sp!,{lr}
    @@ Load the Base Address of the GPIO device
    ldr r1,=gpiobase    @ load pointer to the address of the gpio device
    ldr r1,[r1]     @ load address of the gpio device
    bl  pin_offset  @ get the offset in r3 and shift in r0
    add r1,r1,r3    @ get pointer to the FPFSEL register
    ldr r3,[r1]     @ read the FPFSEL register
    mov r2,#7       @ create bit mask for 3 bits we need to clear
    lsl r2,r2,r0
    mvn r2,r2
    and r3,r2,r3    @ clear the 3 bits
    mov r2,#1       @ create bit mask for the bit we need to set
    lsl r2,r2,r0
    orr r2,r2,r3    @ set the 3 bits
    str r2,[r1]     @ write the FPFSEL register
    ldmfd   sp!,{pc}
```

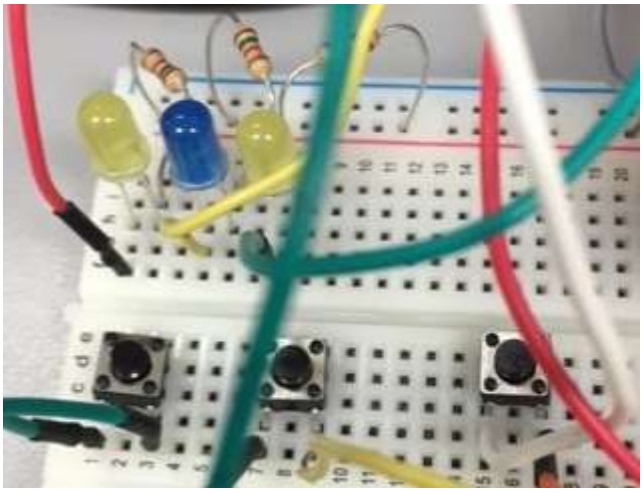
```
    .global main
main:   stmfd   sp!,{r4,lr}

    @@ map the devices into our address space
    bl  IO_init

    @@ Load the Base Address of the GPIO device
    ldr r4,=gpiobase    @ load pointer to the address of the gpio device
    ldr r4,[r4]     @ load address of the gpio device

    @@ Set the direction bits for the pins
    mov r0,#17
    bl  gpio_dir_output

    mov r0,#18
    bl  gpio_dir_input
    mov r0,#3
    bl  gpio_dir_output

    mov r0,#27
    bl  gpio_dir_output
    mov r0,#22
    bl  gpio_dir_input
    mov r0,#23
    bl  gpio_dir_input
    mov r0,#24
    bl  gpio_dir_input
```

```
mov r0,#22
mov r1,#2
bl  set_pud

mov r0,#23
mov r1,#2
bl  set_pud


mov r0,#24
mov r1,#2
bl  set_pud
```

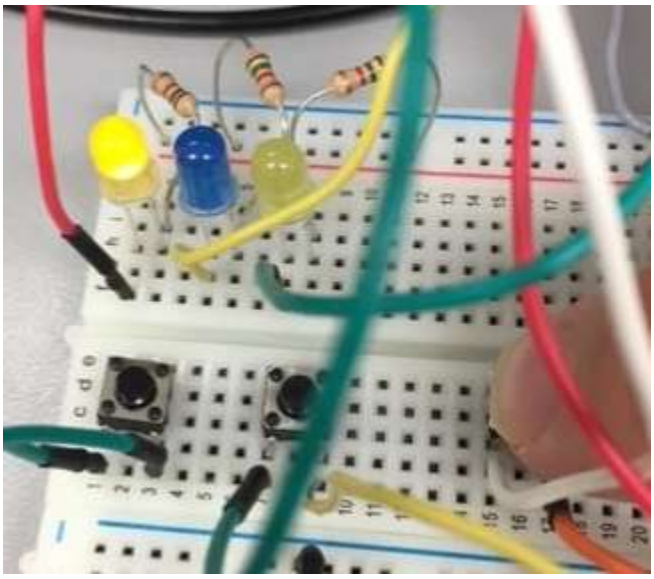
```
loop:
    @@ Read the state of the buttons
    @@ Read GPIO Level Register 0 (GPLEV0)
    ldr r0,[r4,#GPLEV0]

    mov r1,#0       @ will be used to set outputs
    mov r2,#0       @ will be used to clear outputs

    tst r0,#(1<<22) @ check state of pin 22
    orreq   r1,r1,#0x20000  @ if should be set (LED off)
    orrne   r2,r2,#0x20000  @ if should be clear (LED on)

    tst r0,#(1<<23) @ check state of pin 23
    orreq   r1,r1,#0x40000  @ if should be set (LED off)
    orrne   r2,r2,#0x40000  @ if should be clear (LED on)

    tst r0,#(1<<24) @ check state of pin 24
    orreq   r1,r1,#0x8000000    @ if should be set (LED off)
    orrne   r2,r2,#0x8000000    @ if should be clear (LED on)

    str r2,[r4,#GPSET0] @ set some pins
    str r1,[r4,#GPCLR0] @ clear some pins

    b   loop

    ldmfd   sp!,{r4,pc}
```

Results:

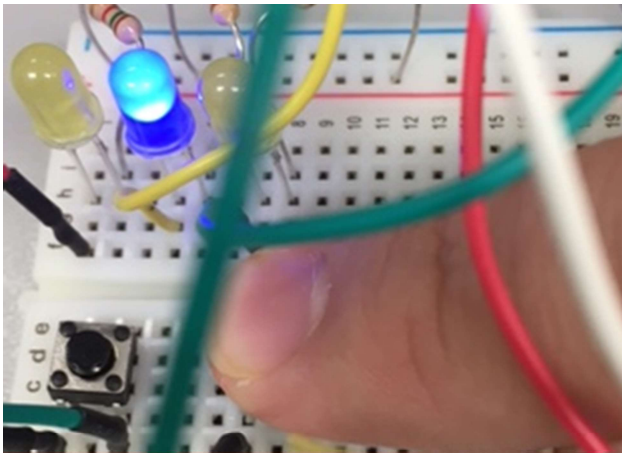
When no buttons are pressed



1st light on

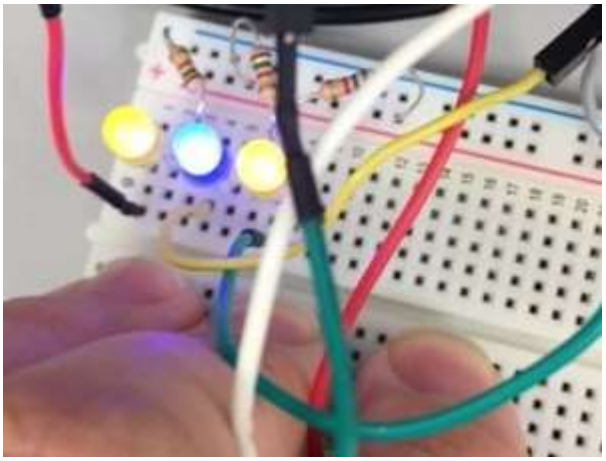


2nd light on



3rd light on

All lights Turning on



Conclusion:

The code worked as intended, the light turns on only when the switch is pressed, and turns off when the switch isn’t pressed.