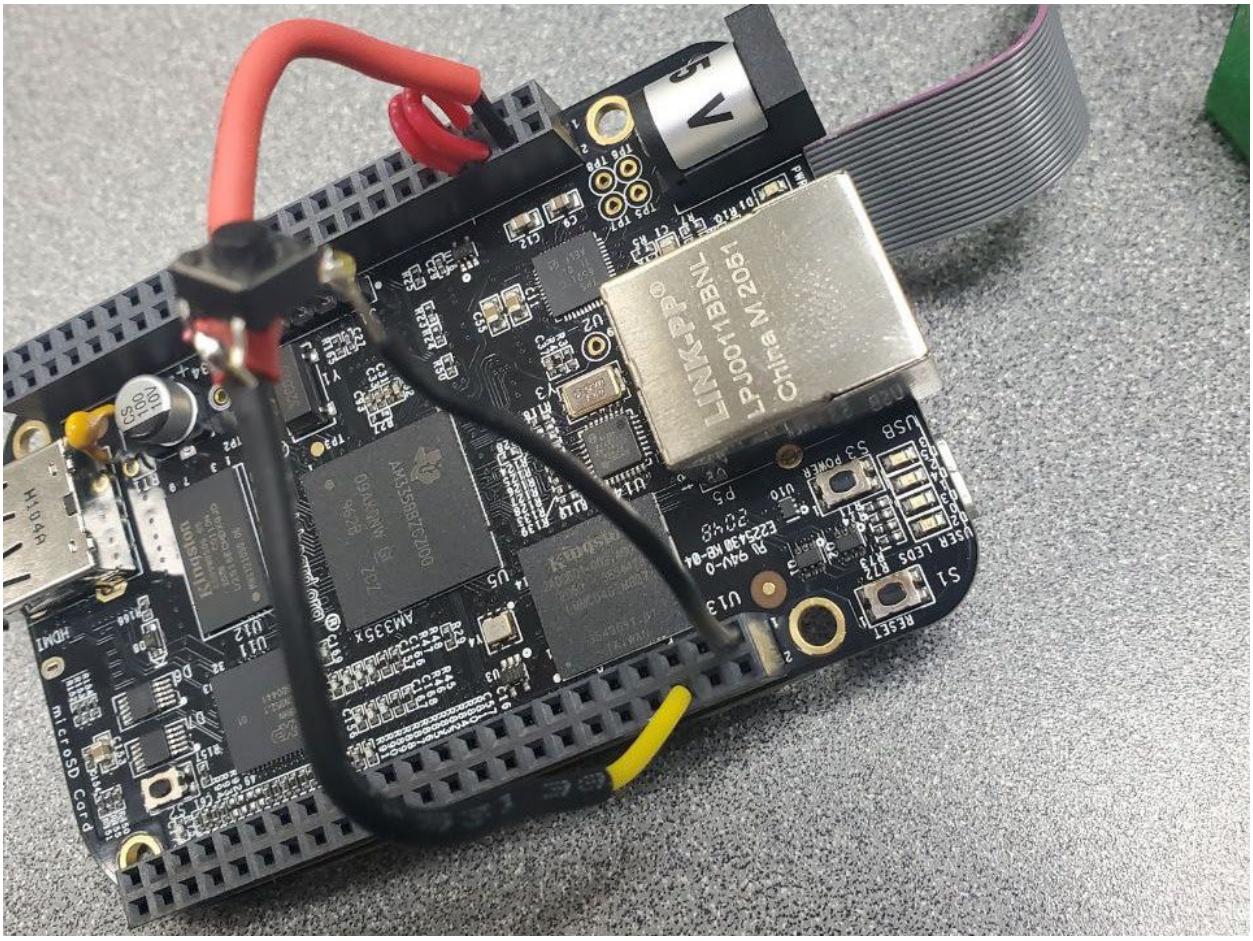
Philip Nevins
ECE 371 Microprocessors
Design Project 2, Part 2

# Design Log

## Entry Date 12/1/2022

In Part 2 of our Design Project, we are tasked with installing a push button that will draw power from the 3.3V pin that holds a logic HIGH on the P8 connector pin that goes to GPIO1_3 on the processor. Pin 6 on the P8 side is this location. When the button is pushed, it will assert a logic LOW. This will trigger an IRQ interrupt that does different functions, which will be explained later. Since I have my own BBB, I went to the Engineer Prototype Lab at Portland State University and soldered the button to 3 wires and a 10k ohm resistor, with the resistor in series under the red wire covering. Then I took a picture to save the pin layout needed for the push button. For my second time soldering, this went very smoothly.



**Entry Date 12/3/2022**

Bit Template information from Part 1 that will be needed for Part 2

GPIO1_21 (LED0)
Enable Output – FFDFFFFF
Write High – 0x00200000

GPIO1_22 (LED1)
Enable Output - FFBFFFFF
Write High – 0x00400000

GPIO1_23 (LED2)
Enable Output - FF7FFFFF
Write High – 0x00800000

GPIO1_24 (LED3)
Enable Output – FEFFFFFF
Write High – 0x01000000

Delay Loop 1 Second
0x00333333
(This was derived from the example in the book, where the delay is 5 seconds. So, we take 0x00FFFFF and divide it by 5, to get 0x00333333)

GPIO1 Clock Addr - 0x44E000AC
GPIO1_SETDATAOUT Addr - 0x4804C194
GPIO1_CLEARDATAOUT Addr - 0x4804C190

GPIO1 21-24
Enable Output – 0xFE1FFFFF
This enables all 4 LEDs, which is what we will use for the cyclone eye.

We are tasked with reading through Hall Chapter 5, pages 207 – 219. This is the section that describes how the interrupt procedures work. There is an interrupt vector table that shows the addresses for the different interrupts, which is shown below. We are specifically interested in using the IRQ interrupt, which is highlighted below. This is the system IRQ interrupt procedure, but we want to use our own IRQ interrupt procedure. How to achieve this will be explained below

| Exception | Mode | Vector Address |
|---|---|---|
| Reset | SVC | 0x00000000 |
| Undefined Instruction | UND | 0x00000004 |
| Software Interrupt | SVC | 0x00000008 |
| Abort (Instruction Fetch Fault) | ABT | 0x0000000C |
| Abort (Data Fetch Fault) | ABT | 0x00000010 |
| Vector Reserved | | 0x00000014 |
| IRQ Interrupt | IRQ | 0x00000018 |
| FIQ Interrupt | FIQ | 0x0000001C |

**Figure 5-1** Interrupt Vector Table assignments for ARM-based processors

Next, we are tasked with reading through Hall Chapter 5, pages 230 – 238. In this section, we learn the steps needed to initialize the microprocessor to acknowledge and execute IRQ interrupts.

Below, we will summarize the different steps for each procedure within the program we are tasked with designing and other important details that will be used to derive the high- and low-level algorithms, as well as the translation to assembly language.

**Steps for Initializing IRQ Interrupts with Detailed Explanations Below**

1. Turn on GPIO1 module clock
2. Initialize GPIO_3 to recognize and generate an interrupt signal for a falling edge on an input signal from the push-bottom switch connected to it
3. Initialize the required registers in the AM335x Interrupt Controller, INTC
4. Hook the IRQ interrupt vector and chain it to your INT_DIRECTOR procedure
5. Enable the processor IRQ input by clearing bit 7 of the Current Program Status Register

[1] Write #0x02 to CM_PER_GPIO1_CLKCTRL (0x44E000AC) register

[2] We must write a HIGH to bit 3 of the GPIO1_FALLINGDETECT (0x4804C14C) register, using the READ, MODIFY, WRITE method. Then we write to the address of GPIO1_IRQSTATUS_SET_0 and enable GPIO1_3 request on POINTRPEND1

[3] We must load the address of INTC_MIR_CLEAR3 Register (0x482000E8), then use the value #0x04 to unmask INTC INT 98, GPIONT1A, then write to INTC_MIR_CLEAR3 register

[4] We must also go into the startup_ARMCA8.s file and change the IRQ 0x18 interrupt LDR line to “b (interrupt function name)”. This allows for our procedure to be called instead of the system IRQ interrupt procedure. We use this method instead of the Hook IRQ vector and chain BUTTON_SVC procedure.

[5] Clear bit 7 of the CPSR_c, by copying it to a register and using BIC #0x80

```
    .section .isr_vector
    .align 4
    .globl  __isr_vector
__isr_vector:
        LDR   pc, [pc,#24]       @ 0x00 Reset
        LDR   pc, [pc,#-8]       @ 0x04 Undefined Instruction
        LDR   pc, [pc,#24]       @ 0x08 Supervisor Call
        LDR   pc, [pc,#-8]       @ 0x0C Prefetch Abort
        LDR   pc, [pc,#-8]       @ 0x10 Data Abort
        LDR   pc, [pc,#-8]       @ 0x14 Not used
        b INT_DIRECTOR           @Branch to our IRQ interrupt procedure
        LDR   pc, [pc,#-8]       @ 0x1C FIQ interrupt
        .long  Entry
        .long  0
        .long  SVC_Handler
        .long  0
        .long  0
        .long  0
        .long  0
        .long  0
```

**Figure 1** startup_ARMCA8.s file after editing to target our IRQ interrupt procedure [4]

**Steps for Button Service Procedure (BUTTON_SVC)**

1. Turn off GPIO1_IRQSTATUS_0
2. Clear bit 0 in INTC_CONTROL

**Steps for IRQ Interrupt Procedure (INT_DIRECTOR)**

1. Push registers to stack
2. Test bit 2 in INTC_PENDING_IRQ3 to check if its from GPIOINT1A
3. If its not from GPIOINT1A, return to pass_on
4. Test GPIO1_IRQSTATUS_0, bit 3
5. If its 1, go to button_svc
6. If its 0, go to pass_on

**Steps for Pass On Procedure (PASS_ON)**

1. We do this procedure to restore the registers we pushed to the stack in the IRQ Interrupt Procedure. Then we return to the wait loop

Additional points from my research using the textbook and the TI data book are listed below. This was translated from the example problems to the required specifications in the design specifications.

1. INT_PENDING_IRQ3 register at 0x4820000F8
2. GPIO1_IRQSTATUS_0 register at 0x4804C02C
3. GPIO1_3 – 0x00000008 – to be used when checking if button is pressed and to turn off the INTC Interrupt Request at GPIO1_3
4. INTC_CONTROL register at 0x48200048
5. To switch to IRQ Mode, use CPS #0x12
6. To switch to SVC Mode, use CPS #0x13

**High-Level Algorithm for Part 2 Single LED**

Initialize STACK for SVC mode
Initialize STACK for SVC mode

Turn on GPIO1 module clock
Initialize GPIO_3 to recognize and generate an interrupt signal for a falling edge on an input signal from the push-bottom switch connected to it
Initialize the required registers in the AM335x Interrupt Controller, INTC
Enable the processor IRQ input

Wait loop

INT_DIRECTOR Procedure
Push registers on stack
Read and test bit 2 of the intc pending irq3 register
    IF NOT from GPIONT1A, go to pass_on procedure
    ELSE
        Test Bit 3 of IRQ STATUS
        IF Bit 3 = 1 THEN
            Go to button service procedure
        ELSE go to pass_on

PASS_ON Procedure
Turn off INTC & GPIO1_3 Interrupt request
Turn off NEQIRQA bit in INTC Control
Turn off all 4 LEDs (encase button is pressed during LED ON function)
Set LED Flag to ON
Restore registers
Return to wait loop

BUTTON_SVC Procedure
Turn off INTC Interrupt request and GPIO port
Clear bit 0 in INTC_Control
(removed and below is added in its place) Go to Cyclone Eye Loop (From Part 1)
**Compare Flag to #0x1
IF Flag = 1 THEN

** Set Flag to OFF
Go to Single LED Loop (From Part 1) **

(Inside Single LED Loop)
** Set Flag to ON
(Before branch to delay loop in each section,)
**IF new IRQ request submitted THEN
Go to PASS_ON
ELSE continue with loop **

**High-Level Algorithm for Part 2 Cyclone Eye**

Initialize STACK for SVC mode
Initialize STACK for SVC mode

Turn on GPIO1 module clock
Initialize GPIO_3 to recognize and generate an interrupt signal for a falling edge on an input signal from the push-bottom switch connected to it
Initialize the required registers in the AM335x Interrupt Controller, INTC
Enable the processor IRQ input

Wait loop

INT_DIRECTOR Procedure
Push registers on stack
Read and test bit 2 of the intc pending irq3 register
IF NOT from GPIONT1A, go to pass_on procedure
ELSE
Test Bit 3 of IRQ STATUS
IF Bit 3 = 1 THEN
Go to button service procedure
ELSE go to pass_on

PASS_ON Procedure
Turn off INTC & GPIO1_3 Interrupt request
Turn off NEQIRQA bit in INTC Control
Turn off all 4 LEDs (encase button is pressed during LED ON function)
Set LED Flag to ON
Restore registers
Return to wait loop

BUTTON_SVC Procedure
Turn off INTC Interrupt request and GPIO port
Clear bit 0 in INTC_Control
(removed and below is added in its place) Go to Cyclone Eye Loop (From Part 1)

**Compare Flag to #0x1
IF Flag = 1 THEN
	** Set Flag to OFF
	Go to Cyclone Eye Loop (From Part 1) **

(Inside Cyclone Eye Loop)
** Set Flag to ON
**Before branch to delay loop in each section,
IF new IRQ request submitted THEN
	Go to PASS_ON
ELSE continue**

**Entry Date 12/4/2022**

Today, I wrote out the low-level algorithm for Part 2 of the design project. It is shown below. After this is complete, I will translate this algorithm into assembly language and attempt to get the program working, where you just turn on the single LED. Then it should be a simple swap to update the working single LED program to work with the cyclone eye loop. You would just do the same thing you do with the single LED, but within each of the cyclone eye LED functions within the loop.

**Low-Level Algorithm for Part 2 Single LED**

Initialize Stack1 pointer for SVC Mode
Turn on IRQ Mode using CPS #0x12
Initialize Stack2 pointer for IRQ Mode
Turn on SVC Mode using CPS #0x13
Load R5 with #0x01

Wait Loop that waits for IRQ Interrupt Request

INT_DIRECTOR Procedure
Push registers to stack
Read INT_PENDING_IRQ3 register at 0x4820000F8 (INTC Base + Offset F8)
Test Bit 2, to see if GPIO1 POINTPREND1 from GPIO1A
If bit 2 = 0, restore registers and return from interrupt
Else read GPIO1_IRQSTATUS_0 register at 0x4804C02C (GPIO Base + Offset 0x2C)
Test bit 3 with 0x00000008 to see if GPIO1_3 button is pressed
IF bit 3 = 0 Then go to PASS_ON
Else go to BUTTON_SVC

PASS_ON Procedure
Use #0x08 to turn off GPIO1_3 & INT Interrupt Request at address #0x4804C02C
Use #0x01 to clear bit 0 in INTC_Control Register #0x48200048

Use #0x1E000000 to turn off LED3 via GPIO1_CLEARDATAOUT #0x4804C190
** Load R8 with #0x01 to signify LED Flag ON
Restore registers that were pushed to the stack inside INT_DIRECTOR
Return to wait loop to wait for another interrupt

BUTTON_SVC Procedure
Turn off INTC Interrupt Request at GPIO1_3 using 0x00000008
Clear bit 0 using 0x01 of the INTC_CONROL register at 0x48200048
** Compare Flag (R8) to #0x01
IF Flag = 1, Go to LED3 Procedure
ELSE go to PASS_ON Procedure**

LED3 Procedure
** Load R8 with #0x00 to signify LED Flag OFF

LED3 Loop from Part 1
(Inside each LED ON / OFF function, add this before Load Delay Timer value)
**Load GPIO_IRQSTATUS_0 Register Address #0x4804C02C
Read STATUS Register
Test STATUS Register with #0x08
If bit 3 = 1, go to PASS_ON procedure
Else continue on with the loop **

**Low-Level Algorithm for Part 2 Cyclone Eye**

Initialize Stack1 pointer for SVC Mode
Turn on IRQ Mode using CPS #0x12
Initialize Stack2 pointer for IRQ Mode
Turn on SVC Mode using CPS #0x13
Load R5 with #0x01

Wait Loop that waits for IRQ Interrupt Request

INT_DIRECTOR Procedure
Push registers to stack
Read INT_PENDING_IRQ3 register at 0x4820000F8 (INTC Base + Offset F8)
Test Bit 2, to see if GPIO1 POINTPREND1 from GPIO1A
If bit 2 = 0, restore registers and return from interrupt
Else read GPIO1_IRQSTATUS_0 register at 0x4804C02C (GPIO Base + Offset 0x2C)
Test bit 3 with 0x00000008 to see if GPIO1_3 button is pressed
IF bit 3 = 0 Then go to PASS_ON
Else go to BUTTON_SVC

PASS_ON Procedure
Use #0x08 to turn off GPIO1_3 & INT Interrupt Request at address #0x4804C02C

Use #0x01 to clear bit 0 in INTC_Control Register #0x48200048
Use #0x1E000000 to turn off LED0-3 via GPIO1_CLEARDATAOUT #0x4804C190
** Load R8 with #0x01 to signify LED Flag ON
Restore registers that were pushed to the stack inside INT_DIRECTOR
Return to wait loop to wait for another interrupt

BUTTON_SVC Procedure
Turn off INTC Interrupt Request at GPIO1_3 using 0x00000008
Clear bit 0 using 0x01 of the INTC_CONROL register at 0x48200048
** Compare Flag (R8) to #0x01
IF Flag = 1, Go to Cyclone Eye Procedure
ELSE go to PASS_ON Procedure**

Cyclone Eye Procedure
** Load R8 with #0x00 to signify LED Flag OFF

Cyclone Eye Loop from Part 1
(Inside each LED ON / OFF function, add this before Load Delay Timer value)
**Load GPIO_IRQSTATUS_0 Register Address #0x4804C02C
Read STATUS Register
Test STATUS Register with #0x08
If bit 3 = 1, go to PASS_ON procedure
Else continue on with the loop **

Notes during programming: No issues! I was surprised it worked the first time! This is why we follow the “fast is slow” rule!

I achieved the goal for today. I was able to get the IRQ Interrupt to work when the button is pressed, and then the single LED and cyclone eye turns on. In the next day or two I will derive the algorithms for, and translate them into assembly, for turning off the LEDs and returning to the wait loop when the button is pressed again. My initial idea is to have an “IF/ELSE” statement in the button_svc procedure. Currently, it just turns on the single LED & cyclone eye and cycles through that loop endlessly.

To implement this “IF/ELSE” statement, we load a register with 0x01 (R5), and then compare that to a “flag register” (R6). This will set the Z flag and we can use that to determine if the button_svc should branch to Cyclone Eye or Turn LEDs Off.

Within each of the branches, we will adjust the flag register.

Inside the Single LED / Cyclone Eye Loop, we will set R8 to #0x00. This will signify the program needs to go to Turn LEDs Off the next time the button is pressed.

Inside the PASS ON procedure, we will set R8 to #0x01. This will signify the program needs to go to Single LED / Cyclone Eye Loop the next time the button is pressed.

**Entry Date 12/5/2022**

We need to figure out a way to efficiently make an ON/OFF flag for the LED, so each button press will switch from LEDs being OFF, to LEDs doing the cyclone eye from Part 1 (ON). We can use Bit 0 in a register to signify the ON/OFF flag. HIGH = on, LOW = off. We can use the TST command to compare the flag register (R8) to binary_0001

With the TST function, it will set the Z flag if they are equal, and clear the Z flag if they are now.

We could implement this by doing the following

Compare Flag to #0x1
IF Flag = 1 THEN
    ** Set Flag to OFF
    Go to Cyclone Eye Loop
ELSE
    ** Set Flag to ON
    Turn Off all 4 LEDs
    Return to wait loop

We also need to check if a new IRQ request has been submitted inside all the LED ON / OFF loops. We can use this from the INT_DIRECTOR, but instead of going to BUTTON_SVC, we go to PASS_ON. So when you trigger an interrupt during the LED loops, it sends you to the PASS_ON that sends you back to the mainline / wait loop.

Test Bit 3 of IRQ STATUS
IF Bit 3 = 1 THEN
    Go to pass_on procedure
ELSE continue

** Signifies Updated High- and Low-Level Algorithm from 12/3/2022 **

**Entry Date 12/7/2022**

Today, I will be translating the ** sections of the algorithms to attempt to finish the program and achieve the required functionality that is stated in the design specifications. This means that, you will press the button once and the cyclone eye will come on. Then you press the button again, and the cyclone eye will turn off. You can repeat this endlessly.

Notes during programming:
I have gotten the button to turn the cyclone eye on, then when you press it, it resets the cyclone eye. It will not turn off the LEDs and I cannot figure out why this is happening. I am emailing the TA to inquire about my issue.

I will update the design log tomorrow with his response

**Entry Date 12/8/2022**

This was his response to my inquiry about why this is happening

"Your button is bouncing so you get multiple presses.  this is veeery common but you should see it work _sometimes_.  To test this set a breakpoint at INT_DIRECTOR and go through step by step."

** Signifies a code update from todays continued debugging

I noticed that R8 was always going back to 0x01 when it left the cyclone eye loop and entered the wait loop after being set to 0x00 inside the cyclone eye loop. I determined this by setting break points and stepping through the program, and I saw this happening in Register 8. I could not explain or figure out the issue. I was originally doing ADD / SUB R8, #0x01. This seemed to be the root cause of the issue, because once I switched to doing a MOV R8, #0x00 / #0x01 where needed, it fixed the issue.

I also was having debouncing issues, because I was only using a 10k ohm resistor in series between the button and the 3.3V… When I added another 20k ohm resistor in series, I was able to get it to work perfectly each time. Without the 20k ohm resistor, there are debouncing issues, where the Cyclone Eye will turn on, then turn off, then turn on, and then LED3 will turn on only, and then turn off. Or the cyclone eye would start, then restart, then restart. This makes sense from what Luis said, where its very common for the button to bounce and cause multiple presses.

Now the program works as intended!!

```
@Phil Nevins
@ECE 371 Microprocessor
@Design Project 2, Part 2 Single LED
@This program will use a pushbutton to trigger an interrupt and cycle an LED
@The program will do this exactly: push button, LED3 cycle on, push button,
@LED3 cycle off, push button, LED3 cycle on...
@Program uses R0-R3, R5-R8

.text
.global _start
.global INT_DIRECTOR

_start:

LDR R13, =STACK1          @Point to base of STACK1 for SVC mode
ADD R13, R13, #0x1000     @Point to top of STACK1
CPS #0x12                       @Switch to IRQ mode
LDR R13, =STACK2          @Point to IRQ STACK2
ADD R13, R13, #0x1000     @Point to top of STACK2
CPS #0x13                       @Back to SVC mode

@Turn on GPIO1 CLK
MOV R0, #0x02             @Value to enable CLK for GPIO module
LDR R1, =0x44E000AC       @ADDR OF CM_PER_GPIO1_CLKCTRL Register
STR R0, [R1]              @Write #02 to register
LDR R0, =0x4804C000       @Base ADDR for GPIO1 Registers

@Detect Falling Edge on GPIO1_3 and eable to assert POINTRPEND1
ADD R1, R0, #0x14C        @R1 = ADDR of GPIO1_FALLINGDETECT Register
MOV R2, #0x00000008       @Load value for Bit 3 (GPIO1_3)
LDR R3, [R1]              @Read GPIO1_FALLINGDETECT register
ORR R3, R3, R2                  @Modify (set bit 3)
STR R3, [R1]              @Write back
ADD R1, R0, #0x34         @Addr of GPIO1_IRQSTATUS_SET_0 Register
STR R2, [R1]              @Enable GPIO1_3 request on POINTRPEND1

@Initialize INTC
LDR R1, =0x482000E8       @ADDR of INTC_MIR_CLEAR3 Register
MOV R2, #0x04             @Value to unmask INTC INT 98, GPIONT1A
STR R2, [R1]              @Write to INTC_MIR_CLEAR3 Register

@Make sure processor IRQ enabled in CPSR
MRS R3, CPSR              @Copy CPSR to R3
BIC R3, #0x80             @Clear bit 7
MSR CPSR_c, R3                  @Write back to CPSR

@Program GPIO1_21-24 as output
LDR R0, =0xFE1FFFFF             @Load word to program GPIO1_21-24 to output
LDR R1, =0x4804C134             @Addr of GPIO1_OE Register
LDR R2, [R1]                    @Read GPIO1_OE Register
AND R2, R2, R0                        @Modify word read in with R0
STR R2, [R1]                    @Write back to GPIO1_OE Register

@Wait for interrupt
```

```
WaitLoop: NOP
            B WaitLoop

INT_DIRECTOR:
STMFD SP!, {R0-R3, LR}     @Push registers on stack
LDR R0, =0x482000F8        @ADDR of INTC_PENDING_IRQ3 Register
LDR R1, [R0]               @Read INTC_PENDING_IRQ3 Register
TST R1, #0x00000004        @Test Bit 2
BEQ PASS_ON                      @Not from GPIOINT1A, go to wait loop, Else
LDR R0, =0x4804C02C        @Load GPIO1_IRQSTATUS_0 Register ADDR
LDR R1, [R0]               @Read STATUS Register
TST R1, #0x00000008        @Test if bit 3 = 1
BNE BUTTON_SVC                   @If 1, go to button_svc
BEQ PASS_ON                      @If 0, go to wait loop

PASS_ON:
      MOV R1, #0x00000008        @Value to turn off GPIO1_3 & INTC Interrupt request
      STR R1, [R0]               @Write to GPIO1_IRQSTATUS_0 Register

            @turn off NEWIRQA bit in INTC_CONTROL, so processor can respond to new
IRQ
      LDR R0, =0x48200048        @ADDR of INTC_CONTROL Register
      MOV R1, #0x01                    @Value to clear bit 0
      STR R1, [R0]               @Write to INTC_CONTROL Register

      MOV R5, #0x1E00000         @Load word to target GPIO1_21-24
      LDR R6, =0x4804C190        @Load addr of GPIO1_CLEARDATAOUT
      STR R5, [R6]               @Write to GPIO1_CLEARDATAOUT (This turns LED1-4 OFF)

      MOV R8, #0x01
      LDMFD SP!, {R0-R3, LR}     @Restore Registers
      SUBS PC, LR, #4                  @Pass execution onto wait LOOP

BUTTON_SVC:
      MOV R1, #0x00000008        @Value to turn off GPIO1_3 & INTC Interrupt request
      STR R1, [R0]               @Write to GPIO1_IRQSTATUS_0 Register
@turn off NEWIRQA bit in INTC_CONTROL, so processor can respond to new IRQ
      LDR R0, =0x48200048        @ADDR of INTC_CONTROL Register
      MOV R1, #0x01                    @Value to clear bit 0
      STR R1, [R0]               @Write to INTC_CONTROL Register

TST R8, #0x01
BNE LED3Function
BEQ PASS_ON

LED3Function:
MOV R8, #0x00

@LED3
LED3ON:                                      @Turn on LED3 Function
      MOV R5, #0x01000000        @Load word to target GPIO1_24
      LDR R6, =0x4804C194        @Load addr of GPIO1_SETDATAOUT
      STR R5, [R6]               @Write to GPIO1_SETDATAOUT (This turns LED ON)

      @Check if new IRQ Request has been submitted
```

```
	LDR R0, =0x4804C02C		@Load GPIO1_IRQSTATUS_0 Register ADDR
	LDR R1, [R0]			@Read STATUS Register
	TST R1, #0x00000008		@Test if bit 3 = 1
	BNE PASS_ON

	LDR R7, =0x00333333		@Load Delay Timer value (one second)
	BL Delay1Sec			@Branch to delay timer function
	B LED3OFF				@Branch to L2R_LED3OFF

LED3OFF:				@Turn off LED3 Function
	MOV R5, #0x01000000		@Load word to target GPIO1_24
	LDR R6, =0x4804C190		@Load addr of GPIO1_CLEARDATAOUT
	STR R5, [R6]			@Write to GPIO1_CLEARDATAOUT (This turns LED OFF)

@Check if new IRQ Request has been submitted
	LDR R0, =0x4804C02C		@Load GPIO1_IRQSTATUS_0 Register ADDR
	LDR R1, [R0]			@Read STATUS Register
	TST R1, #0x00000008		@Test if bit 3 = 1
	BNE PASS_ON

	LDR R7, =0x00333333		@Load Delay Timer value (one second)
	BL Delay1Sec			@Branch to delay timer function
	B LED3ON				@Branch to L2R_LED2ON

Delay1Sec:				@One Second Delay Loop
	SUBS R7, R7, #1			@Subtract 1 from delay timer value
	BNE Delay1Sec			@Loop until delay timer value is 0
	MOV PC, LR				@Branch back to LEDON/OFF
@END OF CODE FROM PART 1

.data
.align 2
STACK1:	.rept 1024			@Stack1
	.word 0x0000
	.endr

STACK2:	.rept 1024			@Stack2
	.word 0x0000
	.endr
.END
```

```
@Phil Nevins
@ECE 371 Microprocessor
@Design Project 2, Part 2 Cyclone Eye
@This program will use a pushbutton to trigger an interrupt and cycle an LED
@The program will do this exactly: push button, LED Cyclone on, push button,
@LED cyclone off, push button, LED cyclone on...
@Program uses R0-R3, R5-R8

.text
.global _start
.global INT_DIRECTOR

_start:

LDR R13, =STACK1          @Point to base of STACK1 for SVC mode
ADD R13, R13, #0x1000     @Point to top of STACK1
CPS #0x12                 @Switch to IRQ mode
LDR R13, =STACK2          @Point to IRQ STACK2
ADD R13, R13, #0x1000     @Point to top of STACK2
CPS #0x13                 @Back to SVC mode

@Turn on GPIO1 CLK
MOV R0, #0x02             @Value to enable CLK for GPIO module
LDR R1, =0x44E000AC       @ADDR OF CM_PER_GPIO1_CLKCTRL Register
STR R0, [R1]              @Write #02 to register
LDR R0, =0x4804C000       @Base ADDR for GPIO1 Registers

@Detect Falling Edge on GPIO1_3 and eable to assert POINTRPEND1
ADD R1, R0, #0x14C        @R1 = ADDR of GPIO1_FALLINGDETECT Register
MOV R2, #0x00000008       @Load value for Bit 3 (GPIO1_3)
LDR R3, [R1]              @Read GPIO1_FALLINGDETECT register
ORR R3, R3, R2            @Modify (set bit 3)
STR R3, [R1]              @Write back
ADD R1, R0, #0x34         @Addr of GPIO1_IRQSTATUS_SET_0 Register
STR R2, [R1]              @Enable GPIO1_3 request on POINTRPEND1

@Initialize INTC
LDR R1, =0x482000E8       @ADDR of INTC_MIR_CLEAR3 Register
MOV R2, #0x04             @Value to unmask INTC INT 98, GPIONT1A
STR R2, [R1]              @Write to INTC_MIR_CLEAR3 Register

@Make sure processor IRQ enabled in CPSR
MRS R3, CPSR              @Copy CPSR to R3
BIC R3, #0x80             @Clear bit 7
MSR CPSR_c, R3            @Write back to CPSR

@Program GPIO1_21-24 as output
LDR R0, =0xFE1FFFFF             @Load word to program GPIO1_21-24 to output
LDR R1, =0x4804C134             @Addr of GPIO1_OE Register
LDR R2, [R1]                    @Read GPIO1_OE Register
AND R2, R2, R0                  @Modify word read in with R0
STR R2, [R1]                    @Write back to GPIO1_OE Register

@Wait for interrupt
```

```
WaitLoop: NOP
            B WaitLoop

INT_DIRECTOR:
STMFD SP!, {R0-R3, LR}     @Push registers on stack
LDR R0, =0x482000F8        @ADDR of INTC_PENDING_IRQ3 Register
LDR R1, [R0]               @Read INTC_PENDING_IRQ3 Register
TST R1, #0x00000004        @Test Bit 2
BEQ PASS_ON                @Not from GPIOINT1A, go to wait loop, Else
LDR R0, =0x4804C02C        @Load GPIO1_IRQSTATUS_0 Register ADDR
LDR R1, [R0]               @Read STATUS Register
TST R1, #0x00000008        @Test if bit 3 = 1
BNE BUTTON_SVC             @If 1, go to button_svc
BEQ PASS_ON                @If 0, go to wait loop

PASS_ON:
      MOV R1, #0x00000008       @Value to turn off GPIO1_3 & INTC Interrupt request
      STR R1, [R0]              @Write to GPIO1_IRQSTATUS_0 Register

@turn off NEWIRQA bit in INTC_CONTROL, so processor can respond to new IRQ
      LDR R0, =0x48200048       @ADDR of INTC_CONTROL Register
      MOV R1, #0x01             @Value to clear bit 0
      STR R1, [R0]              @Write to INTC_CONTROL Register

      MOV R5, #0x1E00000        @Load word to target GPIO1_21-24
      LDR R6, =0x4804C190       @Load addr of GPIO1_CLEARDATAOUT
      STR R5, [R6]              @Write to GPIO1_CLEARDATAOUT (This turns LED1-4 OFF)

      MOV R8, #0x01
      LDMFD SP!, {R0-R3, LR}    @Restore Registers
      SUBS PC, LR, #4           @Pass execution onto wait LOOP

BUTTON_SVC:
      MOV R1, #0x00000008       @Value to turn off GPIO1_3 & INTC Interrupt request
      STR R1, [R0]              @Write to GPIO1_IRQSTATUS_0 Register

@Turn off NEWIRQA bit in INTC_CONTROL, so processor can respond to new IRQ
      LDR R0, =0x48200048       @ADDR of INTC_CONTROL Register
      MOV R1, #0x01             @Value to clear bit 0
      STR R1, [R0]              @Write to INTC_CONTROL Register

TST R8, #0x01
BNE CycloneEye
BEQ PASS_ON

CycloneEye:
MOV R8, #0x00

@LED3
LED3ON:                         @Turn on LED3 Function
      MOV R5, #0x01000000       @Load word to target GPIO1_24
      LDR R6, =0x4804C194       @Load addr of GPIO1_SETDATAOUT
      STR R5, [R6]              @Write to GPIO1_SETDATAOUT (This turns LED ON)

@Check if new IRQ Request has been submitted
```

```
        LDR R0, =0x4804C02C       @Load GPIO1_IRQSTATUS_0 Register ADDR
        LDR R1, [R0]              @Read STATUS Register
        TST R1, #0x00000008       @Test if bit 3 = 1
        BNE PASS_ON

        LDR R7, =0x00333333       @Load Delay Timer value (one second)
        BL Delay1Sec              @Branch to delay timer function
        B LED3OFF                 @Branch to L2R_LED3OFF


LED3OFF:                          @Turn off LED3 Function
        MOV R5, #0x01000000       @Load word to target GPIO1_24
        LDR R6, =0x4804C190       @Load addr of GPIO1_CLEARDATAOUT
        STR R5, [R6]              @Write to GPIO1_CLEARDATAOUT (This turns LED OFF)

@Check if new IRQ Request has been submitted
        LDR R0, =0x4804C02C       @Load GPIO1_IRQSTATUS_0 Register ADDR
        LDR R1, [R0]              @Read STATUS Register
        TST R1, #0x00000008       @Test if bit 3 = 1
        BNE PASS_ON

        LDR R7, =0x00333333       @Load Delay Timer value (one second)
        BL Delay1Sec              @Branch to delay timer function
        B L2R_LED2ON              @Branch to L2R_LED2ON

@LED2
L2R_LED2ON:                       @Turn on LED2 Function
        MOV R5, #0x00800000       @Load word to target GPIO1_23
        LDR R6, =0x4804C194       @Load addr of GPIO1_SETDATAOUT
        STR R5, [R6]              @Write to GPIO1_SETDATAOUT (This turns LED ON)

@Check if new IRQ Request has been submitted
        LDR R0, =0x4804C02C       @Load GPIO1_IRQSTATUS_0 Register ADDR
        LDR R1, [R0]              @Read STATUS Register
        TST R1, #0x00000008       @Test if bit 3 = 1
        BNE PASS_ON

        LDR R7, =0x00333333       @Load Delay Timer value (one second)
        BL Delay1Sec              @Branch to delay timer function
        B L2R_LED2OFF             @Branch to L2R_LED2OFF

L2R_LED2OFF:                      @Turn off LED2 Function
        MOV R5, #0x00800000       @Load word to target GPIO1_23
        LDR R6, =0x4804C190       @Load addr of GPIO1_CLEARDATAOUT
        STR R5, [R6]              @Write to GPIO1_CLEARDATAOUT (This turns LED OFF)

@Check if new IRQ Request has been submitted
        LDR R0, =0x4804C02C       @Load GPIO1_IRQSTATUS_0 Register ADDR
        LDR R1, [R0]              @Read STATUS Register
        TST R1, #0x00000008       @Test if bit 3 = 1
        BNE PASS_ON

        LDR R7, =0x00333333       @Load Delay Timer value (one second)
        BL Delay1Sec              @Branch to delay timer function
        B L2R_LED1ON              @Branch to L2R_LED1ON
```

```
@LED1
L2R_LED1ON:                     @Turn on LED1 Function
       MOV R5, #0x00400000      @Load word to target GPIO1_22
       LDR R6, =0x4804C194      @Load addr of GPIO1_SETDATAOUT
       STR R5, [R6]             @Write to GPIO1_SETDATAOUT (This turns LED ON)

@Check if new IRQ Request has been submitted
       LDR R0, =0x4804C02C      @Load GPIO1_IRQSTATUS_0 Register ADDR
       LDR R1, [R0]             @Read STATUS Register
       TST R1, #0x00000008      @Test if bit 3 = 1
       BNE PASS_ON

       LDR R7, =0x00333333      @Load Delay Timer value (one second)
       BL Delay1Sec             @Branch to delay timer function
       B L2R_LED1OFF            @Branch to L2R_LED1OFF

L2R_LED1OFF:                    @Turn off LED1 Function
       MOV R5, #0x00400000      @Load word to target GPIO1_22
       LDR R6, =0x4804C190      @Load addr of GPIO1_CLEARDATAOUT
       STR R5, [R6]             @Write to GPIO1_CLEARDATAOUT (This turns LED OFF)

@Check if new IRQ Request has been submitted
       LDR R0, =0x4804C02C      @Load GPIO1_IRQSTATUS_0 Register ADDR
       LDR R1, [R0]             @Read STATUS Register
       TST R1, #0x00000008      @Test if bit 3 = 1
       BNE PASS_ON

       LDR R7, =0x00333333      @Load Delay Timer value (one second)
       BL Delay1Sec             @Branch to delay timer function
       B LED0ON                 @Branch to L2R_LED0ON

@LED0
LED0ON:                                 @Turn on LED0 Function
       MOV R5, #0x00200000      @Load word to target GPIO1_21
       LDR R6, =0x4804C194      @Load addr of GPIO1_SETDATAOUT
       STR R5, [R6]             @Write to GPIO1_SETDATAOUT (This turns LED ON)

@Check if new IRQ Request has been submitted
       LDR R0, =0x4804C02C      @Load GPIO1_IRQSTATUS_0 Register ADDR
       LDR R1, [R0]             @Read STATUS Register
       TST R1, #0x00000008      @Test if bit 3 = 1
       BNE PASS_ON

       LDR R7, =0x00333333      @Load Delay Timer value (one second)
       BL Delay1Sec             @Branch to delay timer function
       B LED0OFF                @Branch to L2R_LED0OFF

LED0OFF:                        @Turn off LED0 Function
       MOV R5, #0x00200000      @Load word to target GPIO1_21
       LDR R6, =0x4804C190      @Load addr of GPIO1_CLEARDATAOUT
       STR R5, [R6]             @Write to GPIO1_CLEARDATAOUT (This turns LED OFF)

@Check if new IRQ Request has been submitted
       LDR R0, =0x4804C02C      @Load GPIO1_IRQSTATUS_0 Register ADDR
```

```
	LDR R1, [R0]		@Read STATUS Register
	TST R1, #0x00000008	@Test if bit 3 = 1
	BNE PASS_ON

	LDR R7, =0x00333333	@Load Delay Timer value (one second)
	BL Delay1Sec		@Branch to delay timer function
	B R2L_LED1ON		@Branch to R2L_LED1ON

@This is where it reverses direction
@LED1
R2L_LED1ON:			@Turn on LED1 Function
	MOV R5, #0x00400000	@Load word to target GPIO1_22
	LDR R6, =0x4804C194	@Load addr of GPIO1_SETDATAOUT
	STR R5, [R6]		@Write to GPIO1_SETDATAOUT (This turns LED ON)

@Check if new IRQ Request has been submitted
	LDR R0, =0x4804C02C	@Load GPIO1_IRQSTATUS_0 Register ADDR
	LDR R1, [R0]		@Read STATUS Register
	TST R1, #0x00000008	@Test if bit 3 = 1
	BNE PASS_ON

	LDR R7, =0x00333333	@Load Delay Timer value (one second)
	BL Delay1Sec		@Branch to delay timer function
	B R2L_LED1OFF		@Branch to R2L_LED1OFF

R2L_LED1OFF:			@Turn off LED1 Function
	MOV R5, #0x00400000	@Load word to target GPIO1_22
	LDR R6, =0x4804C190	@Load addr of GPIO1_CLEARDATAOUT
	STR R5, [R6]		@Write to GPIO1_CLEARDATAOUT (This turns LED OFF)

@Check if new IRQ Request has been submitted
	LDR R0, =0x4804C02C	@Load GPIO1_IRQSTATUS_0 Register ADDR
	LDR R1, [R0]		@Read STATUS Register
	TST R1, #0x00000008	@Test if bit 3 = 1
	BNE PASS_ON

	LDR R7, =0x00333333	@Load Delay Timer value (one second)
	BL Delay1Sec		@Branch to delay timer function
	B R2L_LED2ON		@Branch to R2L_LED2ON

@LED2
R2L_LED2ON:			@Turn on LED2 Function
	MOV R5, #0x00800000	@Load word to target GPIO1_23
	LDR R6, =0x4804C194	@Load addr of GPIO1_SETDATAOUT
	STR R5, [R6]		@Write to GPIO1_SETDATAOUT (This turns LED ON)

		@Check if new IRQ Request has been submitted
	LDR R0, =0x4804C02C	@Load GPIO1_IRQSTATUS_0 Register ADDR
	LDR R1, [R0]		@Read STATUS Register
	TST R1, #0x00000008	@Test if bit 3 = 1
	BNE PASS_ON

	LDR R7, =0x00333333	@Load Delay Timer value (one second)
	BL Delay1Sec		@Branch to delay timer function
	B R2L_LED2OFF			@Branch to R2L_LED2OFF
```

```
R2L_LED2OFF:                        @Turn off LED2 Function
      MOV R5, #0x00800000       @Load word to target GPIO1_23
      LDR R6, =0x4804C190       @Load addr of GPIO1_CLEARDATAOUT
      STR R5, [R6]              @Write to GPIO1_CLEARDATAOUT (This turns LED OFF)

            @Check if new IRQ Request has been submitted
      LDR R0, =0x4804C02C       @Load GPIO1_IRQSTATUS_0 Register ADDR
      LDR R1, [R0]              @Read STATUS Register
      TST R1, #0x00000008       @Test if bit 3 = 1
      BNE PASS_ON

      LDR R7, =0x00333333       @Load Delay Timer value (one second)
      BL Delay1Sec              @Branch to delay timer function
      B LED3ON                        @Branch to R2L_LED3ON

Delay1Sec:                          @One Second Delay Loop
      SUBS R7, R7, #1               @Subtract 1 from delay timer value
      BNE Delay1Sec             @Loop until delay timer value is 0
      MOV PC, LR                    @Branch back to LEDON/OFF
@END OF CODE FROM PART 1

.data
.align 2
STACK1:       .rept 1024              @Stack1
              .word 0x0000
              .endr

STACK2:       .rept 1024              @Stack2
              .word 0x0000
              .endr
.END
```

By signing this statement, I affirm that I did not give any help to any other person, did not receive any help from any other person, except TA and Instructor and did not obtain any information from the Internet or other sources.

Signature __Philip A Nevins_12/8/2022