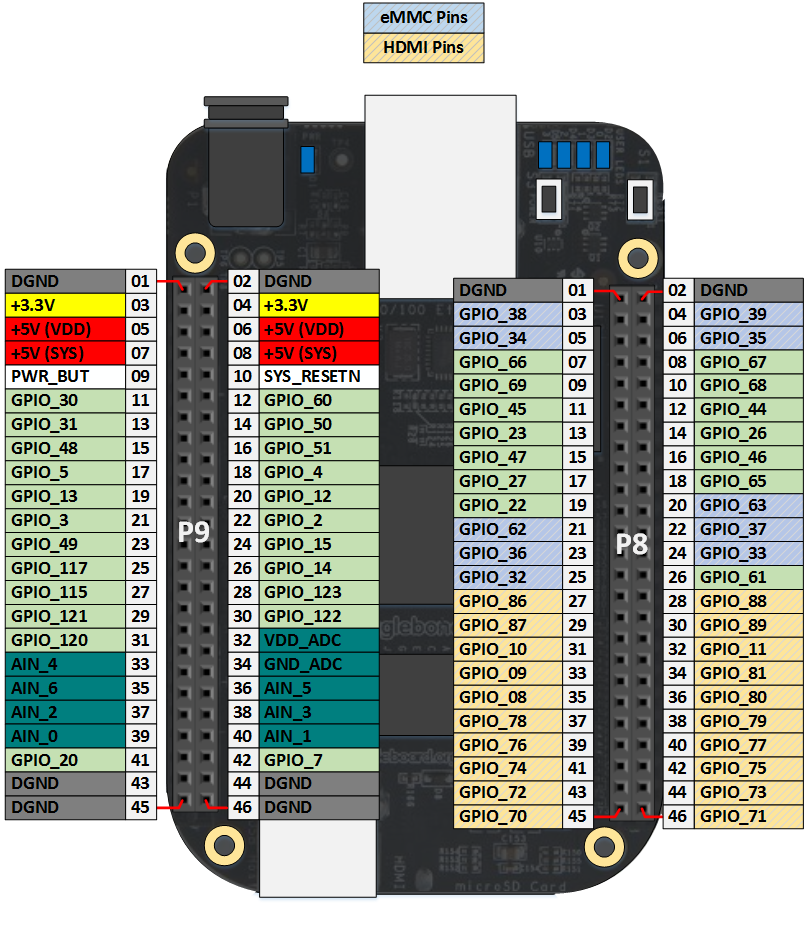
ECE371 – DESIGN PROJECT #2, FALL 2018

Josh Pradera

Introduction:

This project consists in tree sections. The first section consists in learning how to control the 4 USR led’s in the Beagle Bone Black. The light patter needs to be in a rotating patter like a theater marquee with delay loop timing. The second part of the project states developing an interrupt procedure that services and interrupt request from a debounced, push button switch. The push of the button will allow to stop and start the rotating of the led’s. The third part its design to use timers replacing the led loop.



The beagle bone black has many pins but also many of those are used for different functionalities in the board. In the diagram above the pins are colored with different colors so that functionalities can be observed.

Dark green pins are analog input pins. The analog input pins are labeled as AIN\_0, AIN\_1, AIN\_2, AIN\_3, AIN\_4, AIN\_5, AIN\_6 in the diagram above.

**Part1**

|  |  |
| --- | --- |
| **USER LED** | **GPIO PIN** |
| User led 0 | GPIO1\_21 |
| User led 1 | GPIO1\_22 |
| User led2 | GPIO1\_23 |
| User led 3 | GPIO1\_24 |

|  |  |
| --- | --- |
| GPIO0 base address | 0x44E07000 |
| GPIO1 base address | 0x4804C000 |
| GPIO2 base address | 0x481AC000 |
| GPIO3 base address | 0x481AE000 |

**Initializing all four GPIO1 from 21 to 24 for SetDataOut and ClearDataOut**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **GPIO** | **31** | **30** | **29** | **28** | **27** | **26** | **25** | **24** | **23** | **22** | **21** | **20** |
| Bit | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| ***HEX*** | 0 | | | | 1 | | | | E | | | |
| **GPIO** | **19** | **18** | **17** | **16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| Bit | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ***HEX*** | 0 | | | | 0 | | | | 0 | | | |
| **GPIO** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Bit | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ***HEX*** | 0 | | | | 0 | | | |

**Turn on USER LED 0**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **GPIO** | **31** | **30** | **29** | **28** | **27** | **26** | **25** | **24** | **23** | **22** | **21** | **20** |
| **Bit** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **HEX** | 0 | | | | 0 | | | | 2 | | | |
| **GPIO** | **19** | **18** | **17** | **16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| **Bit** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **HEX** | 0 | | | | 0 | | | | 0 | | | |
| **GPIO** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Bit** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **HEX** | 0 | | | | 0 | | | |

Hex: 0x00200000

**Turn on USER LED 1**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **GPIO** | **31** | **30** | **29** | **28** | **27** | **26** | **25** | **24** | **23** | **22** | **21** | **20** |
| **Bit** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| **HEX** | 0 | | | | 0 | | | | 4 | | | |
| **GPIO** | **19** | **18** | **17** | **16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| **Bit** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **HEX** | 0 | | | | 0 | | | | 0 | | | |
| **GPIO** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Bit** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **HEX** | 0 | | | | 0 | | | |

Hex: 0x00400000

**Turn on USER LED 2**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **GPIO** | **31** | **30** | **29** | **28** | **27** | **26** | **25** | **24** | **23** | **22** | **21** | **20** |
| **Bit** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| **HEX** | 0 | | | | 0 | | | | 8 | | | |
| **GPIO** | **19** | **18** | **17** | **16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| **Bit** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **HEX** | 0 | | | | 0 | | | | 0 | | | |
| **GPIO** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Bit** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **HEX** | 0 | | | | 0 | | | |

Hex: 0x00800000

**Turn on USER LED 3**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **GPIO** | **31** | **30** | **29** | **28** | **27** | **26** | **25** | **24** | **23** | **22** | **21** | **20** |
| **Bit** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **HEX** | 0 | | | | 1 | | | | 0 | | | |
| **GPIO** | **19** | **18** | **17** | **16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| **Bit** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **HEX** | 0 | | | | 0 | | | | 0 | | | |
| **GPIO** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Bit** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **HEX** | 0 | | | | 0 | | | |

Hex: 0x01000000

To enable bit 21, 22, 23,24 as output we need to write 0 to each of those bits of the GPIO1\_OE registers

|  |  |
| --- | --- |
| **Enable bit as Output** | **Word to turn USR logic High when output** |
| Pin 21 : 0xFFDFFFFF | 0x00200000 |
| pin 22: 0xFFBFFFFF | 0x00400000 |
| Pin 23 : 0xFF7FFFFF | 0x00800000 |
| Pin 24 : 0xFEFFFFFF | 0x01000000 |
| Pin 21-24: 0xFE1FFFFF | 0x01E00000 |

**Standard program structure and algorithm:**

* Initialize the stack
* Point stack pointer to the top of the stack
* Initialize clock for GPIO Modules (LDR R0,=0x02)
* Load address of CM\_PER\_GPIO1\_CLKCTRL register
* Write #02 to register
* Initialize base address for GPIO1 (0x4804C000)
* @ Turn Off Value for all the USR leds of GPIO1
* Set GPIO1 bits 21-24 low (0x01E00000) by using ClearDataOut (GPIO1 base address + ClearDataOut offset) = (0x4804C190)
* Write to To GPIO1 CLEARDATAOUT
* @ Set GPIO1\_21-24 as outputs
* Set GPIO1 bits 21-24 as outputs (0xFE1FFFFF) by Read, Modify, Write to GPIO\_OE (GPIO1 base address + GPIO\_OE offset) = 0x4804C134

@ Loop NEXT to turn on and off the leds

NEXT: next loop will run infinetly.

@ Light 1

* Get word for GPIO1\_21 (#0x00200000)
* Load address to SETDATAOUT to turn led on (0x4804C194)
* @ Delay
* Branch to a delay loop that will count to 2 seconds before coming back to mainline
* Then Turn Off the led before turning on the next one
* Get word for GPIO1\_21 (#0x00200000)
* Load address to CLEARDATAOUT to turn led on (0x4804C190)

@Light 2

* Get word for GPIO1\_22 (#0x00400000)
* Load address to SETDATAOUT to turn led on (0x4804C194)
* @ Delay
* Branch to a delay loop that will count to 2 seconds before coming back to mainline
* Then Turn Off the led before turning on the next one
* Get word for GPIO1\_22 (#0x00400000)
* Load address to CLEARDATAOUT to turn led on (0x4804C190)

@Light 3

* Get word for GPIO1\_21 (#0x00800000)
* Load address to SETDATAOUT to turn led on (0x4804C194)
* @ Delay
* Branch to a delay loop that will count to 2 seconds before coming back to mainline
* Then Turn Off the led before turning on the next one
* Get word for GPIO1\_23 (#0x00800000)
* Load address to CLEARDATAOUT to turn led on (0x4804C190)

@Light 4

* Get word for GPIO1\_24 (#0x01000000)
* Load address to SETDATAOUT to turn led on (0x4804C194)
* @ Delay
* Branch to a delay loop that will count to 2 seconds before coming back to mainline
* A Return for the loop (B NEXT)

Delaying the led’s to turn by 1 second

* delay:
* Store register on stack
* Load value 0x00400000 for delay time of 1 second
* Decrement value by 1 until = 0
* Restore register from Stack
* Return to mainline

After completing Part 1 I had several problems where the T.A. helped me

* The value for the progressing the leds one by one every second was changed from 0x00800000 to 0x00400000
* Also creating the loop and using, STR R1,[R0, #0x194], to turn on the led in one line by using the offset.

**PART 1 Algorithm**

@ Turning and rotating USR leds

@ Josh Pradera

@ ECE 371

**.text**

**.global** \_start

**\_start:**

LDR R13, =STACK @Initialize the Stack

ADDS R13, R13, #100 @ Points stack pointer to the top of the stack

LDR R0,=0x02 @ Value to enable the clock for GPIO MODULES

LDR R1,=0x44E000AC @ Address of CM\_PER\_GPIO1\_CLKCTRL REgister

STR R0,[R1] @ write #02 to register

LDR R0,=0x4804C000 @ BASE address for GPIO1 register

@ Turn off value for all the USR leds of GPIO1

MOV R1, #0x01E00000 @ Get Word to make all pin 21-24 off

ADD R4, R0, #0x190 @ Add base address of GPIO1 with offset for ClearDataOut

STR R1, [R4] @ Write to GPIO1 ClearDataOut (0x4804C190)

@ Set GPIO1\_21-24 as outputs

ADD R6, R0, #0x0134 @ Make GPIO\_OE register address

LDR R7, [R6] @ Get GPIO1\_OE output enable register

MOV R8, #0xFE1FFFFF @ Get word to enable pins as outputs

AND R7, R8, R7 @ (Modify), clear bits 21-24

STR R7, [R6] @ (Write) to GPIO1 Output Enable register

**NEXT:**

@Light 1

MOV R1, #0x00200000 @ Get word for GPIO1\_Pin21

STR R1,[R0, #0x194] @ Load Address to SETDATAOUT to turn On pin 21

@delay

BL delay @ Branch to delay implementation

MOV R1, #0x00200000 @ Get word for pin 21

STR R1,[R0, #0x190] @ Load Address to CLEARDATAOUT to tur off GPIO1\_pin

@Light 2

MOV R1, #0x00400000 @ Get word for GPIO1\_Pin22

STR R1,[R0, #0x194] @ Load Address to SETDATAOUT to turn On pin 22

@delay

BL delay @ Branch to delay implementation

MOV R1, #0x00400000 @ Get word for GPIO1\_Pin22

STR R1,[R0, #0x190] @ Load Address to CLEARDATAOUT to tur off GPIO1\_pin 22

@Light 3

MOV R1, #0x00800000 @ Get word for GPIO1\_Pin23

STR R1,[R0, #0x194] @ Load Address to SETDATAOUT to turn On pin 23

@delay

BL delay @ Branch to delay implementation

MOV R1, #0x00800000 @ Get word for GPIO1\_Pin23

STR R1,[R0, #0x190] @ Load Address to CLEARDATAOUT to tur off GPIO1\_pin 23

@Light 4

MOV R1, #0x01000000 @ Get word for GPIO1\_Pin24

STR R1,[R0, #0x194] @ Load Address to SETDATAOUT to turn On pin 24

@delay

BL delay @ Branch to delay implementation

MOV R1, #0x01000000 @ Get word for GPIO1\_Pin24

STR R1,[R0, #0x190] @ Load Address to CLEARDATAOUT to tur off GPIO1\_pin 24

B NEXT

**delay:**

STMFD SP!, {R5, LR} @ Store register on stack

MOV R5,#0x00400000 @ Load delay value for 2s

**NEXT1**

NOP

SUBS R5,#1 @ Subtract 1

BNE NEXT1

LDMFD SP!, {R5, LR} @ Restore saved registers

MOV PC, R14 @ Return to mainline

**.data**

**.align** 2

**STACK:** .rept 256

**.byte** 0x00

.endr

.end

**Part 2 Creating an Interrupt Procedure for Servicing a Button Push**

1. Problems encountered:
   * Understanding The example in the book ( Figure 5-14 )and setting up GPIO2
   * Things needed :
   * GPIO2 base address ( 0x481AC000 )
   * Load address of CM\_PER\_GPIO2\_CLKCTRL register(0x44E000B0)
   * Since we are using GPIO2\_1 for the button understanding that bit 1 corresponds to the word ( 0x00000002 )

* Initializing the INT Controller took me a while to understand by reading the book and the help of ALEX finally helped me visualize better the setup
  + Int #32 corresponds to bit 1 in the MIR1 register
  + Load the address of INT\_MIR\_CLEAR1 register (0x482000A8)
  + To unmask bit 1 of the MIR1 register we write ( MOV R2,#0x01 )
* There was misalignment with the stacks that Alex help me to figure out to finally get the assembly code working. The solution was to add .align 2 before the STACKS implementation in the .data section

**Standard Program structure**

@ Base address for GPIO2 (0x481AC000)

* Initialization setup of the assembly code
  + .text
  + .global start
  + .global INT\_DIRECTOR
  + \_start:
* Initialize the stack. There are 2 STACKS : STACK0 and STACK1
* @ STACK0
* Point the base to stack for SVC mode ( LDR R13, =STACK0 )
* Point stack pointer to the top of the stack (ADD R13, R13, #0x1000 )
* Switch to IRQ mode ( CPS #0x12 )
* @ STACK1
* Point to IRQ stack
* Point to top of STACK1
* Back to SVC mode ( CPS #0x13 )
* @ Turn on GPIO1 CLK
* Initialize clock for GPIO Modules (LDR R0,=0x02)
* Load address of CM\_PER\_GPIO1\_CLKCTRL register(0x44E000AC)
* Write #02 to register
* Load Address of GPIO1 (0x4804C000)
* @ Turn on GPIO2 CLK
* Initialize clock for GPIO Modules (LDR R0,=0x02)
* Load address of CM\_PER\_GPIO2\_CLKCTRL register(0x44E000B0)
* Write #02 to register
* Initialize base address for GPIO2 (0x481AC000)
* @ Turn Off Value for all the USR leds of GPIO1
* Set GPIO1 bits 21-24 low (0x01E00000) by using ClearDataOut (GPIO1 base address + ClearDataOut offset) = (0x481AC190)
* Write to GPIO1\_CLEARDATAOUT
* Load Value to turn led off on GPIO2\_1 (#0x00000001)
* Write to GPIO2\_CLEARDATAOUT register
* @ Set GPIO1-21\_24 as outputs
* Make GPIO\_OE registers (base address + #0x0134)
* (Read) the current GPIO1\_OE output enable register
* Get the word that enable pins as outputs (#0xFE1FFFFF)
* (MODIFY) clear all bit 21\_24
* (WRITE) to GPIO1 Output Enable register
* @ Detect falling edge of GPIO2\_1 and also enable to assert POINTRPEND1
* Load address for GPIO2\_FALLINGDETECT register(0x481AC000 + #0x14C)
* Load Value for bit 1 (0x00000002)
* Read GPIO2\_FALLINGDETECT register
* Modify By using ( ORR R3, R3, R2)
* Write back to R3
* Write the offset of GPIO2\_IRQSTATUS\_SET\_0 register (#0x34)
* Enable GPIO2\_1 on POINTRPEND1
* @ Initialize INTC
* Load the address of INT\_MIR\_CLEAR1 register (0x482000A8)
* Value to unmask INTC INT 32, GPIOINT1A
* WRITE to INTC\_MIR\_CLEAR1 register
* @ Make sure processor IRQ enabled in CPSR
* @ Master IRQ enable
* Copy CPSR to R3
* Clear bit 7 with (BIC R3, #0x80)
* Write back to CPSR
* @ Wait for interrupt LOOP:
* Load pointer to Flag to check for the state of the button
* Load the value in another register (R3) so that we can proceed to compare with the value 0
* Get the value 0 into R1
* Compare R3 with R1
* If flag = 1, then jump back to loop
* Otherwise the flag = 0 and branch to Next to rotate the led
* @ NEXT loop where the leds rotate
* @ Led 1
* Load value of GPIO1 in register R0 to use SETDATAOUT and CLEARDATAOUT
* Get word for GPIO1\_21 ( #0x00200000 )
* Load address to SETDATAOUT to turn on the led ( base address of GPIO1 + offset ) = (0x4804C000 + #0x194 )
* Branch to delay to accomplish the 1 second delay
* Get word for GPIO1\_21 ( #0x00200000 )
* Load address to CLEARDATAOUT to turn off GPIO1\_21 (0x4804C000 + #0x190 )
* @ Led 2
* Load value of GPIO1 in register R0 to use SETDATAOUT and CLEARDATAOUT
* Get word for GPIO1\_22 ( #0x00400000 )
* Load address to SETDATAOUT to turn on the led ( base address of GPIO1 + offset ) = (0x4804C000 + #0x194 )
* Branch to delay to accomplish the 1 second delay
* Get word for GPIO1\_22 ( #0x00400000 )
* Load address to CLEARDATAOUT to turn off GPIO1\_22 (0x4804C000 + #0x190 )
* @ Led 3
* Load value of GPIO1 in register R0 to use SETDATAOUT and CLEARDATAOUT
* Get word for GPIO1\_23 ( #0x00800000 )
* Load address to SETDATAOUT to turn on the led ( base address of GPIO1 + offset ) = (0x4804C000 + #0x194 )
* Branch to delay to accomplish the 1 second delay
* Get word for GPIO1\_23 ( #0x00800000 )
* Load address to CLEARDATAOUT to turn off GPIO1\_23 (0x4804C000 + #0x190 )

@ Led 4

* Load value of GPIO1 in register R0 to use SETDATAOUT and CLEARDATAOUT
* Get word for GPIO1\_24 ( #0x01000000 )
* Load address to SETDATAOUT to turn on the led ( base address of GPIO1 + offset ) = (0x4804C000 + #0x194 )
* Branch to delay to accomplish the 1 second delay
* Get word for GPIO1\_24 ( #0x01000000 )
* Load address to CLEARDATAOUT to turn off GPIO1\_24 (0x4804C000 + #0x190 )

@Check for button push

* Load pointer to Flag to check for the state of the button
* Load the value in another register (R3) so that we can proceed to compare with the value 0
* Get the value 0 into R1
* Compare R3 with R1
* If flag = 1, then jump back to loop
* Otherwise the flag = 0 and branch to Next to rotate the led

@ Loop for delay

* Store registers on stack by using STMFD SP!, {R5, LR}
* Load register ( R5 ) with value to obtain the 1 second shift of the led
* Loop for subtraction NEXT1
* Subtract the register by 1
* Branch if not equal back to the subtraction loop NEXT1
* Otherwise Restore saved register LDMFD SP!, {R5, PC}
* @Setting the INT\_DIRECTOR
* Push registers on stack by using: STMFD SP!, {R0-R12, LR}
* Load address of INTC\_PENDING\_IRQ1 register ( 0x482000B8 )
* Read INTC\_PENDING\_IRQ2 register
* test bit 0 (INT#32) correspond to bit 0 ( TST R1, #0x00000001 )
* **IF** not from GPIOINT1A, go back to wait loop , **Else**  ( BEQ PASS\_ON )
* Load GPIO2\_IRQSTATUS\_0 register address ( 0x481AC02C )
* Read the status of the register
* Check if bit 1 is equal to 1 by using ( TST R1,#0x00000002 )
* If bit 1 = 1, branch to BUTTON\_SVC ( BNE BUTTON\_SVC )
* If bit 1 = 0, branch to PASS\_ON to go back to wait loop
* @ PASS\_ON implementation
* Restore the registers ( LDMFD SP!, {R0-R3, LR} )
* Pass execution on to wait LOOP for now ( SUBS PC, LR, #4 )
* @ BUTTON\_SVC implementation
* Load GPIO2\_IRQSTATUS\_0 register address ( 0x481AC02C )
* Load Value turns off GPIO2\_1 Interrupt request ( #00000002 )
* Write to GPIO2\_IRQSTATUS\_0 register
* @ Turn OFF NEWIRQA bit in INTC\_CONTROL, so that the processor can respond to the new IRQ
* Load address of INT\_CONTROL register ( 0x48200048 )
* Load Value to clear bit 0 ,turn off the new IRQ bit in the INTC\_CONTROL register so the processor can respond ( MOV R1, #01 )
* Write to INTC\_CONTROL register
* @ Checking to record the second press of the button and turn OFF or ON the led
* Load pointer to Flag to check for the state of the button ( R2 )
* Load the value in another register (R3)
* Proceed (EOR) with the Flag Value set in .data ( Flag: .byte 0xFF )
* FF EOR FF = 0, 0 EOR FF = 1.
* Store the value

@ Return:

* Restore registers by using ( LDMFD SP!,{R0-R12,LR} )
* Return from IRQ interrupt procedure
* @ .data setup
* When using stacks and dealing with words and bytes its safe to use .align 2 to avoid the program from crashing
* Setup Flag value (Flag: .byte 0xFF )
* @ There are two stacks

STACK0 and STACK1

They will use .rept 1021 and .word 0x0000

* .END

**Code for PART 2**

@ Part 2 project 2 ECE 371

@ Josh Pradera

@ Interrupt procedure that services an interrupt request from a debounced push button switch. When button its pushed

@ The leds will rotate if the button is pushed they will turn OFF, if the button its pushed again the leds will be ON rotating.

**.text**

**.global** \_start

**.global** INT\_DIRECTOR

**\_start:**

@ Stack0

LDR R13, =STACK0 @ Point to base of stack for SVC mode

ADD R13, R13, #0x1000 @ Points stack pointer to the top of the stack

CPS #0x12 @ Switch to IRQ mode

@ Stack1

LDR R13, =STACK1 @ Point to IRQ stack

ADD R13, R13,#0x1000 @ Point to top of the STACK1

CPS #0x13 @ Back to SVC mode

@ Turn On GPIO1 CLK

LDR R0,=0x02 @ Value to enable the clock for GPIO MODULES

LDR R1,=0x44E000AC @ Address of CM\_PER\_GPIO1\_CLKCTRL Register

STR R0,[R1] @ write #02 to register

@ Turn On GPIO2 CLK

LDR R0, =0x02 @ Value to enable the clock for GPIO MODULES

LDR R1, =0x44E000B0 @ Address of CM\_PER\_GPIO2\_CLKCTRL Register

STR R0, [R1] @ write #02 to register

@ Turn off value for all the USR leds of GPIO1

LDR R0,=0x4804C000 @ BASE address for GPIO1 register

ADD R1,R0, #0x134 @ Add output enable register of GPIO1

ADD R4, R0, #0x190 @ Add base address of GPIO1 with offset for ClearDataOut

LDR R1, =0x01E00000 @ Get Word to make all pin 21-24 off/01E00000

STR R1, [R4] @ Write to GPIO1 ClearDataOut (0x4804C190)

@ Set GPIO1\_21-24 as outputs

LDR R0,=0x4804C000 @ BASE address for GPIO1 register

ADD R6, R0, #0x0134 @ Make GPIO\_OE register address

LDR R7, [R6] @ (READ) GPIO1\_OE output enable register

MOV R8, #0xFE1FFFFF @ Get word to enable pins as outputs

AND R7, R8, R7 @ (Modify), clear bits 21-24

STR R7, [R6] @ (Write) to GPIO1 Output Enable register

@ Detect falling edge on GPIO2\_1 and enable assert on POINTRPEND1

LDR R4, = 0x481AC000 @ Load base address GPIO2

ADD R1, R4, #0x14C @ Address for GPIO2\_FALLINGDETECT register

MOV R2, #0x00000002 @ Load value for bit 1

LDR R3, [R1] @ Read GPIO2\_FALLINGDETECT register

ORR R3, R3, R2 @ Modify

STR R3, [R1] @ Write back

ADD R1, R4, #0x34 @ Offset for GPIO\_IRQSTATUS\_SET\_0 register

STR R2, [R1] @ Enable GPIO2\_1 on POINTRPEND1

@ Initialize INTC

LDR R1,=0x482000A8 @ Address of INTC\_MIC\_CLEAR1 register

MOV R2,#0x01 @ Value to unmask INTC INT 32, GPIOINT1A

STR R2,[R1] @ Write to INC\_MIR\_CLEAR1 register

@ Master IRQ enable

@ Make sure the processor IRQ enabled in CPSR

MRS R3, CPSR @ Copy CPSR to R3

BIC R3, #0x80 @ Clear bit 7

MSR CPSR\_c, R13 @ Write back to CPSR

@ Wait for interrupt

**LOOP:**

LDR R2, =Flag @ Load pointer to Flag to check the state of the button

LDRB R3, [R2] @ Load value from Flag

MOV R1, #0 @ Load zero to R1 to compare

CMP R3, R1 @ Compare with value from Flag

BEQ LOOP @ If Flag = 1, jump to loop,

B NEXT @ Otherwise rotate the leds

**NEXT:**

LDR R0,=0x4804C000 @ BASE address for GPIO1 register

@Light 1

MOV R1, #0x00200000 @ Get word for GPIO1\_Pin21

STR R1,[R0, #0x194] @ Load Address to SETDATAOUT to turn On pin 21

@delay

BL delay @ Branch to delay implementation

MOV R1, #0x00200000 @ Get word for pin 21

STR R1,[R0, #0x190] @ Load Address to CLEARDATAOUT to tur off GPIO1\_pin 21

@Light 2

MOV R1, #0x00400000 @ Get word for GPIO1\_Pin22

STR R1,[R0, #0x194] @ Load Address to SETDATAOUT to turn On pin 22

@delay

BL delay @ Branch to delay implementation

MOV R1, #0x00400000 @ Get word for GPIO1\_Pin22

STR R1,[R0, #0x190] @ Load Address to CLEARDATAOUT to tur off GPIO1\_pin 22

@Light 3

MOV R1, #0x00800000 @ Get word for GPIO1\_Pin23

STR R1,[R0, #0x194] @ Load Address to SETDATAOUT to turn On pin 23

@delay

BL delay @ Branch to delay implementation

MOV R1, #0x00800000 @ Get word for GPIO1\_Pin23

STR R1,[R0, #0x190] @ Load Address to CLEARDATAOUT to tur off GPIO1\_pin 23

@Light 4

MOV R1, #0x01000000 @ Get word for GPIO1\_Pin24

STR R1,[R0, #0x194] @ Load Address to SETDATAOUT to turn On pin 24

@delay

BL delay @ Branch to delay implementation

MOV R1, #0x01000000 @ Get word for GPIO1\_Pin24

STR R1,[R0, #0x190] @ Load Address to CLEARDATAOUT to tur off GPIO1\_pin 24

LDR R2, =Flag @ Load pointer to Flag to check the state of the button

LDR R3, [R2] @ Load value from Flag

MOV R1, #0 @ Load zero to R1 to compare

CMP R1, R3 @ Compare with value from Flag

BEQ LOOP @ If Flag = 1, jump to loop,

B NEXT @ Otherwise rotate the leds

**delay:** STMFD SP!, {R5, LR} @ Store register on stack

LDR R5, =0x003FFFFF @ Load delay value for 1s

**NEXT1:** SUBS R5, R5, #1 @ Subtract 1

BNE NEXT1

LDMFD SP!, {R5, PC} @ Restore saved registers

**INT\_DIRECTOR:**

STMFD SP!, {R0-R12, LR} @ Push registers on stack

LDR R0, = 0x482000B8 @ Address of INTC\_PENDING\_IRQ1 register

LDR R1,[R0] @ Read INTC\_PENDING\_IRQ2 register

TST R1, #0x00000001 @ test bit 0 (INT#32) correspond to bit 0

BEQ PASS\_ON @ IF not from GPIOINT2A, go back to wait loop , Else

LDR R0,=0x481AC02C @ Load GPIO2\_IRQSTATUS\_0 register address

LDR R1, [R0] @ Read STATUS registers

TST R1,#0x00000002 @ Check if bit 1 = 1

BNE BUTTON\_SVC @ If bit 1 = 1, then button its pushed

BEQ PASS\_ON @ If bit 1 = 0, go back to wait loop

**PASS\_ON:**

LDMFD SP!, {R0-R12, LR} @ Restore registers

SUBS PC, LR, #4 @ Pass execution on to wait LOOP for now

**BUTTON\_SVC:**

LDR R0,=0x481AC02C @ Load GPIO2\_IRQSTATUS\_0 register address

MOV R1, #00000002 @ Value turns off GPIO2\_1 Interrupt request

STR R1,[R0] @ Write to GPIO2\_IRQSTATUS\_0 register

@ Turn OFF NEWIRQA bit in INTC\_CONTROL, so that the processor can respond to the new IRQ

LDR R0,=0x48200048 @ Address of INT\_CONTROL register

MOV R1, #01 @ Value to clear bit 0 ,turn off the new IRQ bit @in the INTC\_CONTROL register so the processor can respond

@ to another IRQ interrupt

STR R1,[R0] @ Write to INTC\_CONTROL register

@ Checking to record the second press of the button and turn OFF or ON the leds

LDR R2, =Flag @ Load pointer to Flag to check the state of the button

LDR R3, [R2] @ Load value from Flag

EOR R3,R3,#0xFF @ Compare the value of Flag ( FF EOR FF = 0 )

STR R3, [R2] @ Store value

**Return:**

LDMFD SP!,{R0-R12,LR} @ Restore registers

SUBS PC,LR,#4 @ Return from IRQ interrupt procedure

**.data**

**.align** 2

**Flag:** **.byte** 0xFF

**.align** 2

**STACK0:** .rept 1024

**.word** 0x0000

.endr

**STACK1:** .rept 1024

**.word** 0x0000

.endr

.end

**MODIFY STARTUP FILE to take execution to int\_director**

**.extern** INT\_DIRECTOR

**.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 @ 0x18 IRQ interrup goes to here**

@ LDR pc, [pc,#-8] @ 0x18 IRQ interrupt

LDR pc, [pc,#-8] @ 0x1C FIQ interrupt

**.long** Entry

**.long** 0

**.long** SVC\_Handler

**.long** 0

**.long** 0

**.long** 0

**.long** 0

**.long** 0

/\* External interrupts \*/

**Part 3- Using interrupt from timer**

* This section also needs the modified startup file for the Interrupt setup.
* Interrupt associated with Timer4 is #92
* This corresponds to bit 28 of the INTC\_MIR\_CLR2 register at Offset ( 0xC8 ) from the INT base address of 0x48200000

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **GPIO** | **31** | **30** | **29** | **28** | **27** | **26** | **25** | **24** | **23** | **22** | **21** | **20** |
| **Bit** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **HEX** | 1 | | | | 0 | | | | 0 | | | |
| **GPIO** | **19** | **18** | **17** | **16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** |
| **Bit** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **HEX** | 0 | | | | 0 | | | | 0 | | | |
| **GPIO** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Bit** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **HEX** | 0 | | | | 0 | | | |

* + Write 0x10000000 to this address to enable TIMER4 interrupt in INTC
  + Turn ON the clock to the TIMER4 module!
  + Base address of clock module ( CM\_PER = 0x44E00000 )
  + CM\_PER\_TIMER4 register offset = 0x88
  + Write 0x02 to the register to enable the clock to the module
  + CLKSEL\_TIMER4\_CLK Register (offset = 10h)
  + But the offset will really be 0x510, because CM\_DPLL 0x44E0\_0500
  + Write 0x02 to the resultant address ( 0x44E00510 ) to select the 32.768 KHZ clock
  + Base address for Timer4 is 0x48044000
  + Reset Timer4 by writing 0x1 to the timer4 configuration at offset 0x10
  + Write 0x2 to Timer4 IRQ\_ENABLE\_SET register at offset 0x2C to enable timer4 to generate and interrupt signal when the counter overflows
  + If we count 32,768 pulses per second this will be 0x00008000 pulses for a time of 1 second.
  + TLDR = 0x10000000-0x00008000 = 0xFFFF8000

**Standard Program structure**

MAINLINE

* Set up stacks for supervisor mode IRQ mode
* Turn on GPIO1 clock
* Turn on GPIO2 clock
* Set GPIO1-21\_24 for led off (low)
* Set GPIO1\_21-24 as outputs
* Initialize INTC
  + - Reset INTC, Timer4 interrupt on INTC # 92, GPIO2\_1 on INTC # 32
    - Write 0x2 to INTC\_SYSCONFIG at 0x48200010 to reset INTC
    - Write 0x10000000 to 0x482000C8 to enable INTC # 92 TIMER4 interrupt input at bit 28
    - Write 0x01 to 0x482000A8 to Enable INTC # 32 GPIO2\_1 Interrupt
* Turn ON Timer4 CLK
  + - Write 0x2 to CM\_PER\_TIMER4\_CLKCTRL at 0x44E00000+0x88
* Set Timer clock frequency multiplexer for 32.768 KHz
  + - Write 0x02 to PRCMCLKSEL\_TIMER4 register address 0x44E00510
* Initialize timer register for desired count , overflow interrupt generation
  + - Write 0x1 to Timer4 CFG register at 0x48044010 to reset TIMER4
    - Write 0x2 to Timer4 IRQENABLE\_SET register at 0x4804402C
    - Write 0XFFFFC000 to Timer4 TLDR register at 0x48044040 to get .5 seconds
    - Write 0xFFFFC000 to Timer4 TCRR register at 0x4804403C to get 0.5 second
* Enable IRQ input by clear bit 7 in CPSR
* Wait for interrupt loop

INT\_DIRECTOR

* Save registers
* Check if interrupt from button
  + - Read INTC\_PENDING\_IRQ1 register at 482000B8
    - If Bit 0 = 0 then not GPIO1, go check for timer
    - If Bit 0 = 1 then check if GPIO2\_1 from button
    - Read GPIO2\_1\_IRQ\_STATUS REGISTER at 0x481AC02C
    - If bit 1 = 1 , Then go to BUTTON\_SVC
    - If bit 1 = 0 then Enable new IRQ response in INTC by
      * Write 0x1 to INTC\_CONTROL register at 48200048 to allow new IRQ
      * Restore registers and return to wait loop
    - Check in INTC if interrupt from Timer4
      * Read word from INT\_PENDING\_IRQ2 register at 0x482000D8
      * Test with 0x10000000
      * NO, write 0x2 to INTC\_CONTROL register at 48200048 to allow new IRQ
      * Restore registers and return
    - If yes then check if timer4 overflow
      * Read value from timer4 IRQSTATUS register at 0x48040028
      * Test with 0x00000002
      * If bit 1 = 1 go to Loop rotating Leds
      * If bit 1 = 0 then write 0x1 to INTC\_CONTROL register at 48200048 to allow new IRQ
      * Restore registers and return to loop
    - BUTTON\_SVC
      * Load GPIO\_2 IRQSTATUS\_0 register address ( 0x481AC02C )
      * Value to turn OFF GPIO2\_1 interrupt request (#0x00000002)
      * Write to GPIO2\_IRQSTATUS\_0 register
      * Turn OFF New IRQ bit in INT\_CONTROL so that the processor can respond to the new IRQ
      * Load address of INT\_CONTROL register
      * Value to clear bit 0 (#01)
      * Write to INT\_CONTROL register
      * Start timer4 and set for auto reload by write 0x03 to TCLR at 0x48044038
      * Write 0x1 to INTC\_CONTROL register at 48200048 to allow new IRQ
      * Branch if not equal to STOP
    - LED :
      * Set a flag to one to indicate that the leds are ON
      * Load address of GPIO1 register for the leds
      * Turn off timer4 overflow interrupt request
      * Write a 0x2 to IRQSTATUS register at 0x48044028
      * Reset TIMER4 overflow IRQ request
      * Write

When pushing the button to start the leds we must reset timer overflow IRQS

* + - * Write a 1 to bit 1 to both set and clear registers with ( 0x2)
      * ENABLE IRQ
    - Toggle:
      * Load the address of GPIO1
      * Compare register 9 with a number to decide witch led to branch to
      * LEds 1, 2, 3, 4 will be chosen if the register R9 its at that value when the comparison happens
    - Stop
      * Clear all the leds
      * Set flag to zero
      * Turn all the leds off using word #0x01E00000 by RMW ( clear data out (offset #0x190) )
      * Load base address of Timer4 write #0x2 set and clear registers
      * IRQENABLE SET ( #0x2C ) IRQENABLE CLEAR ( #0x30 )
    - PASS\_ON\_2:
      * Load address of INC control register ( 0x48200048 )
      * Write ( #0x01 ) to reset
      * And write
      * Return to wait loop
      * Restore registers and pass execution to wait loop for now-
      * By using LDMFD SP!, {R0-R3, LR} and SUBS PC, LR, #4
      * Use of .aling 2 for stack purposes
      * .END

Code for Part 3

@ Project 2\_ Part 3 ECE 371

@ Josh Pradera

@ When the output from a debounced switch on GPIO2\_1 goes low, the program produces an IRQ to the processor

@ The IRQ service at procedure for this request will make 4 leds step through every second and start TIMER4

@ When the timer 4 count overflows this interrupt, the leds will be toggled.

**.text**

**.global** \_start

**.global** INT\_DIRECTOR

**\_start:**

LDR R13,=STACK1 @ Point to base of STACK for SVC mode

ADD R13, R13, #0x1000 @ point to top of stack

CPS #0x12 @ Switch to IRQ mode

LDR R13,=STACK2 @ Point to IRQ Stack

ADD R13, R13, #0x1000 @ Point to top of Stack

CPS #0x13 @ back to SVC mode

@ Turn On GPIO1 CLK

LDR R0,=0x02 @ Value to enable the clock for GPIO MODULES

LDR R1,=0x44E000AC @ Address of CM\_PER\_GPIO1\_CLKCTRL REgister

STR R0,[R1] @ write #02 to register

@ Turn On GPIO2 CLK

LDR R0, =0x02 @ Value to enable the clock for GPIO MODULES

LDR R1, =0x44E000B0 @ Address of CM\_PER\_GPIO2\_CLKCTRL REgister

STR R0, [R1] @ write #02 to register

LDR R0,=0x4804C000 @ GPIO1 Base Address

ADD R4, R0, #0x190 @ GPIO1\_CLEARDATAOUT register

MOV R7, #0x01E00000 @ LOAD VALUE TO TURN OFF ALL LEd

STR R7, [R4] @ TURN OFF ALL LED'S,

ADD R2, R0, #0x134 @ GPIO1\_OE REGISTER ADDRESS offset

LDR R6, [R2] @ READ GPIO1 ENABLE REGISTER

LDR R7,=0xFE1FFFFF @ WORD TO ENABLE GPIO1 ALL LED'S AS AN OUTPUT

AND R6, R7, R6 @ CLEAR BITS 21, 22, 23, 24 USR0-USR3,

STR R6, [R2] @ WRITE TO GPIO1\_CLEARDATAOUT REGISTER.

@ Detect falling edge on GPIO2\_1 and enable to assert POINTERPEND1

@ Write 0x00000002 to 0x481AC14C to set GPIO2\_1 to detect the falling edge

LDR R1,=0x481AC000 @GPIO2 Base Address

ADD R2, R1, #0x14C @ R2 = address of GPIO2\_FALLINGDETECT

MOV R3, #0x00000002 @ load value for bit 1

LDR R4, [R2] @ READ GPIO2\_FALLINGDETECT

ORR R4, R4, R3 @ MODIFY (set bit 1)

STR R4, [R2] @ WRITE

@ Enable GPIO module to send interrupt request to INTC:

ADD R2, R1, #0x34 @ Address of GPIO2\_IRQSTATUS\_SET\_0

STR R3, [R2] @ enable GPIO2\_1 request on POINTERPEND1

@ Initialize INTC :

LDR R2,=0x48200000 @ Address of INTC base register,

@ 0x2 to INTC\_SYSCONFIG at 0x48200010 to reset INTC

MOV R3, #0x2 @ Value to reset INTC

STR R3, [R2, #0x10] @ Write to INTC\_SYS\_CONFIG register

@ 0x10000000 to 0x482000C8 enable INTC #92 Timer4:

MOV R3, #0x10000000 @ unmask INTC #92, MIR2 bit 28, THEREFORE -- IRQ2.

STR R3, [R2, #0xC8] @ INTC base 4820 + offset 0xC8 INTC\_MIR\_CLEAR2

@ For the button, write 0x00000001 to INTC\_MIR\_SET1 register at address 0x482000A8

MOV R3, #0x00000001 @ value to unmask INTC INT 32. GPIOINT2A

STR R3, [R2, #0xA8] @ WRITE to INTC\_MIR\_CLEAR1 register.

@ Turn on Timer4 clock:

@ Write 0x02 to Base Address of CM\_PER 0x44E00000 base + offset 0x88 for CM\_PER\_TIMER4\_CLKCTRL

MOV R3, #0x2 @ Value to enable TIMER4 CLK

LDR R1,=0x44E00088 @ Address CM\_PER\_TIMER4 CLK

STR R3, [R1] @ Turn on CLK

@Set Timer clock frequency MUX for 32K Hz:

@Write 0x2 to PRCM CLKSEL\_TIMER4 register at address 0x44E00510

LDR R1,=0x44E00510 @ Address of PRCM CLKSEL\_TIMER4 register

STR R3, [R1] @ Select 32K CLK for timer 4, by writing 0x2

@ Initialize Timer 4 registers, with count, overflow interrupt generation

@ Write 0x1 to TIMER4\_CFG at 0x48044010 to reset Timer4

LDR R1,=0x48044000 @ Base address TIMER4 registers

MOV R3, #0x1 @ value to reset TIMER4

STR R3, [R1, #0x10] @ Write to TIMER4 CONFIG register

MOV R3, #0x2 @ Get value to enable Overflow interrupt

STR R3, [R1, #0x2C] @ Write to TIMER4 IRQENABLE\_SET

LDR R3,=0xFFFF8000 @ Count value for 1 second

STR R3, [R1, #0x40] @ Timer4 TLDR load register

STR R3, [R1, #0x3C] @ Write to Timer4 TCRR count 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

MOV R8, #0x0 @ set flag to zero

@ wait for interrupt

**LOOP:** NOP

B LOOP

**INT\_DIRECTOR:**

STMFD SP!, {R0-R3, LR} @ push registers on stack

LDR R0,=0x482000B8 @ Address of INTC\_PENDING\_IRQ1

LDR R1, [R0] @ Read INTC\_PENDING\_IRQ1 register

TST R1, #0x00000001 @ test bit 0 of IRQ 1

BEQ TCHK @ if not from button push, Check if Timer 4, else

@ Check for button pushed

LDR R0,=0x481AC02C @ Load GPIO2\_IRQSTATUS\_0 register address

LDR R1, [R0] @ READ status register, to see if button pushed

TST R1, #0x00000002 @ Check if bit 1 = 1

BNE BUTTON\_SVC @ If bit 1 = 1, then button pushed!

LDR R0,=0x48200048 @ Else, go back address of INTC\_PENDING\_IRQ2 register TIMER4

MOV R3, #01 @ value to clear bit 0

STR R3, [R0] @ Write to INTC\_CONTROL register.

LDMFD SP!, {R0-R3, LR} @ Restore registers = POP

SUBS PC, LR, #4 @ Pass execution on to wait LOOP for now

**TCHK:**

LDR R3,=0x482000D8 @ Address of INTC PENDING IRQ2 register

LDR R0, [R3] @ Get value

TST R0, #0x10000000 @ Check if interrupt is from TIMER4

BEQ PASS\_ON @ If not TIMER4 interrupt, return; if YES, check overflow

@ If yes, then we have TIMER4 interrupt, go check for overflow

LDR R3,=0x48044028 @ Address of TIMER4 IRQSTATUS register

LDR R0, [R3] @ get value

TST R0, #0x2 @ check bit 1

BNE LED @ If we have overflow, then go to toggle LEDs

@ else go back to wait loop

**PASS\_ON:**

LDR R0,=0x48200048 @ Else, go back

MOV R3, #01 @ value to clear bit 0

STR R3, [R0] @ Write to INTC\_CONTROL register.

LDMFD SP!, {R0-R3, LR} @ Restore registers

SUBS PC, LR, #4 @ Pass execution on to wait LOOP for now

**BUTTON\_SVC:**

LDR R0,=0x481AC02C @ Load GPIO2\_IRQSTATUS\_0 register address, text p. 131

MOV R1, #0x00000002 @ Value turns off GPIO2\_1 interrupt request

@ also INTC interrupt request

STR R1, [R0] @ Write to GPIO2\_IRQSTATUS\_0 register

@ Start Timer4, and set for auto reload

MOV R3, #0x03 @ load value of auto reload timer and start

LDR R1,=0x48044038

STR R3, [R1]

@ Turn OFF NEWIRQA bit in INTC\_CONTROL

LDR R0,=0x48200048 @ Address of INTC\_CONTROL

MOV R1, #01 @ Clear Bit 0

STR R1, [R0] @ Write to INTC\_CONTROL register

LDR R0,=0x4804C000 @ LOAD ADDRESS OF GPIO1 REGISTER for led

TST R8, #0x1 @ Check if bit 1 = 1

BNE STOP @ If bit = 1, then button pushed

**LED:**

MOV R8, #0x1 @ set FLAG to 1, This means that the leds are ON

LDR R0,=0x4804C000 @ LOAD ADDRESS OF GPIO1 REGISTER for LEDs

@ Turn off Timer4 Overflow Interrupt request:

@ Write 0x2 to IRQSTATUS register at 0x48044028

LDR R1,=0x48044028 @ load address of Timer4 IRQSTATUS register

MOV R2, #0x2 @ value to reset timer4 overflow IRQ request (bit1)

STR R2, [R1] @ write.

LDR R1,=0x48044000 @ BASE ADDRESS OF TIMER4,

MOV R3, #0x2 @ WRITE A 1 TO BIT 1 OF BOTH SET AND CLR REGISTERS,

STR R3, [R1, #0x2C] @ IRQENABLE\_SET

**TOGGLE:**

LDR R0,=0x4804C000 @ LOAD ADDRESS OF GPIO1

CMP R9, #01 @ Is the value 1?

BEQ ONE @ If yes, branch to ONE

CMP R9, #02 @ Is the Value 2?

BEQ TWO @ IF yes, branch to TWO

CMP R9, #03 @ Is the value 3?

BEQ TREE @ If yes, branch to tree

CMP R9, #04 @ Is the value 4?

BEQ FOUR @ Branch to Four if its Equal

BNE ONE @ Jump to Led one for ther case there is no specified number

**ONE:**

@Light 1

@ Turn On Led 1 , but turn OFF leds 2.3 and 4

MOV R9, #0x2

MOV R1, #0x01C00000 @ Get word for GPIO1\_Pin22, 23 and 24

STR R1,[R0, #0x190] @ Load Address to CLEARDATAOUT

MOV R1, #0x00200000 @ Get word for GPIO1\_Pin21

STR R1,[R0, #0x194] @ Load Address to SETDATAOUT to turn On pin 21

B PASS\_ON\_2

**TWO:**

MOV R9, #0x3

@Light 2

MOV R1, #0x01A00000 @ Get word for GPIO1\_Pin21, 23 and 24

STR R1,[R0, #0x190] @ Load Address to CLEARDATAOUT

MOV R1, #0x00400000 @ Get word for GPIO1\_Pin22

STR R1,[R0, #0x194] @ Load Address to SETDATAOUT to turn On pin 22

B PASS\_ON\_2

**TREE:**

MOV R9, #0x4

@Light 3

MOV R1, #0x01600000 @ Get word for GPIO1\_Pin21, 22 and 24

STR R1,[R0, #0x190] @ Load Address to CLEARDATAOUT

MOV R1, #0x00800000 @ Get word for GPIO1\_Pin23

STR R1,[R0, #0x194] @ Load Address to SETDATAOUT to turn On pin 23

B PASS\_ON\_2

**FOUR:**

MOV R9, #0x1

@Light 4

MOV R1, #0x00F00000 @ Get word for GPIO1\_Pin24

STR R1,[R0, #0x190]

MOV R1, #0x01000000 @ Get word for GPIO1\_Pin24

STR R1,[R0, #0x194] @ Load Address to SETDATAOUT to turn On pin 24

B PASS\_ON\_2

**STOP:**

@ Clear all the leds

LDR R0,=0x4804C000 @ LOAD ADDRESS OF GPIO1 REGISTER for LEDs

MOV R8, #0x0 @ set FLAG to 0

ADD R4, R0, #0x190 @ GPIO1\_CLEARDATAOUT register

MOV R7, #0x01E00000 @ LOAD VALUE TO TURN OFF ALL LEDs

STR R7, [R4] @ Turn the leds OFF

@ When Pushing the button to stop the LEDs, MUST disable timer overflow IRQ's.

LDR R1,=0x48044000 @ BASE ADDRESS OF TIMER4,

MOV R3, #0x2 @ WRITE A 1 TO BIT 1 OF BOTH SET AND CLR REGISTERS,

STR R3, [R1, #0x2C] @ IRQENABLE\_SET

STR R3, [R1, #0X30] @ IRQENABLE\_CLR

**PASS\_ON\_2:**

@ INTC reset for the new IRQ

LDR R1,=0x48200048 @ address of INTC control register

MOV R3, #0x01

STR R3, [R1] @ Write to INTC\_CONTROL register

@return to wait loop

LDMFD SP!, {R0-R3, LR} @ Restore registers

SUBS PC, LR, #4 @ Pass execution on to wait LOOP for now

**.align** 2

**SYS\_IRQ:** .WORD 0 @location to store systems’ IRQ address!

**.data**

**.align** 2

**STACK1:** .rept 1024

**.word** 0x0000

.endr

**STACK2:** .rept 1024

**.word** 0x0000

.endr

.END

**I JOSH PRADERA developed and wrote this program by myself with NO help from anyone except the instructor and/or the T.A. and I did not give any assistance to anyone else.**