|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | ***Portland State University*** [Maseeh College of Engineering and Computer Science](http://pdx.edu/cecs/) 1559170_300.jpg | | ECE 371 - PROJECT REPORT | | Project 2: GPIO AND INTERRUPT CONTROLER | | **12/12/2014** | |
|  |
| **Instructor: Douglas V.Hall**  **TA: Leela Yadlapalli** |
| Students: Hai Dang Hoang  PSUID: 946749560  Email: [danghai@pdx.edu](mailto:danghai@pdx.edu) |

**Introduction:**

There are two major parts to this design project. The first part is to learn how to control GPIO pins to turn the 4 BeagleBone Black USR LEDs on and off in a rotating pattern like a theater marquee with delay loop timing. In the second part of this project, I will develop an interrupt procedure that services an interrupt request from a debounced, push-button switch. The first time the button is pushed, the interrupt procedure will start the rotating LED display. The next time the button is pushed, the interrupt procedure will stop the rotating LED display. For this project I will be using the 4 User LEDs on the BeagleBone Black boards and the output from a debounced, pushbutton switch that has been connected to GPIO1\_31 on each board.

**PART I: USER LEDS AND GPIO**

1. **GPIO Table:**

In this program, I will use GPIO1\_21, GPIO1\_22, GPIO1\_23, GPIO1\_24 ( LED on the BeagleBone Black board) to rotate.

**Table 1: Template for initializing GPIO1\_21-24 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 | | | |

Hex: 0x01E00000

**Table 2: Template for initializing GPIO1\_21-24 as output**

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

Hex: 0xFE1FFFFF

**Table 3: Template for turning on USER LED 0 only**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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

**Table 4: Word for turn on USER LED 1 only**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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

**Table 5: Word for turn on USER LED 2 only**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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

**Table 6: Word for turn on USER LED 3 only**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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

1. **Standard Program Structure and Algorithm:**
2. **Mainline:**

* Initialize STACK
* Initialize GPIO\_1

Initial base address for GPIO\_1 (0x4804C000)

Set GPIO1 bits 21-24 to low by write 0x01E00000 to GPIO1\_CLEARDATAOUT at 0x4804C190 (0x4804C000 + 0x190 offset for GPIO\_CLEARDATAOUT) (Table 1)

Set GPIO1 bits 21-24 to output by RMW 0xFE1FFFFF to GPIO1\_OE at 0x4804C134 (0x4804C000 + 0x134 offset for GPIO\_OE) ( Table 2)

* Initialize array for 4 USER LEDS to display ( LED\_Display )
* Initialize a register equal 0 ( Register control index to get value from pointer of status 4 LED from LED\_Display)
* Repeat

Call procedure to turn on a specific LED ( LED\_on)

Call procedure delay 2 seconds ( Delay)

Clear 4 LED ( Write 0x01E00000 into GPIO\_CLEARDATAOUT)

Compare Register ( store index of LED\_Display) with 12

If (Register == 12)

Reset register = 0 ( Back to 1st address of LED\_Display)

Else

Increment register by 4 to get the next address for turning on the next LED

* Until N/A ( Endless Loop)

1. **Procedure LED\_on:**

* Store register on Stack
* Load address of GPIO1\_SETDATAOUT at 0x4804C190
* Load a value from Pointer LED\_Display with an index register
* Turn on specific LED by storing the value into GPIO1\_SETDATAOUT
* Restore register from STACK
* Return to mainline

1. **Procedure Delay 2 seconds:**

* Store register on Stack
* Load value 0x00400000 for counting down
* Repeat

Decrement value by one

* Until value equals 0
* Restore register from STACK
* Return to mainline

1. **Initialize Data:**

* LED\_Display: .word 0x00200000, 0x00400000, 0x00800000, 01000000

0x00200000 : GPIO1\_21 on

0x00400000 : GPIO1\_22 on

0x00800000 : GPIO1\_23 on

0x01000000 : GPIO1\_24 on

* Initial STACK space

1. **Source code:**

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@ ECE 371 – PROGRAMMING PROJECT II

@----------------------------------------------------------------------------

@ USR LEDS AND GPIO

@ The program will turn the 4 BeagleBone Black USR LEDs on and off in a

@ rotating pattern like a theater marquee with delay loop timing.

@ The USR LEDS will rotate from USR LED 0 to USR LED 3 and go back to USR LED 0

@----------------------------------------------------------------------------

@ Hai Dang Hoang

@ Reference: Douglas V. Hall and Leela Yadlapalli

@ DECEMBER 18th, 2014

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

**.text**

**.global** \_start

**\_start:**

LDR R13, =STACK @ Create stack

ADDS R13,R13,#100 @ Point stack pointer at top of stack

@ Load value to turn off all three LEDs USR at bit 21,22,23 and 24 of GPIO1

LDR R0,=0x4804C000 @ Base address GPIO1 registers

LDR R7,=0x01E00000 @ GPIO1 21-24 off with GPIO1\_CLEARDATAOUT register

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

STR R7,[R4] @ Write to GPIO1\_CLEARDATAOUT register

@ Program GPIO1\_21-24 as outputs

ADD R5,R0,#0x0134 @ Make GPIO1\_OE register adress

LDR R6,[R5] @ Read current GPIO1\_OE Output Enable register

LDR R7,=0xFE1FFFFF @ Word to enable GPIO1\_21-24 as outputs

AND R6,R7,R6 @ Clear bits 21-24 ( Modify)

STR R6,[R5] @ Write to GPIO1 Output Enable register

@ Create a array storing status of LED\_Display

LDR R1,=LED\_Display @ Initialize pointer array for turning 4 USR LEDS on

MOV R5, #0x0 @ R5 Register(Index) control status of LED\_Display

LDR R7,=0x01E00000 @ Value for clear all 4 USR LEDS

@ Rotating 4 USR LEDS ( Loop forever )

**LOOP:**

BL LED\_on @ 4 USR LEDS run from right to left (GPIO1.21-> GPIO1.24)

BL Delay @ Call delay 2s Procedure

STR R7,[R4] @ Clear all 4 USR LEDS ( R4 current stores address of GPIO1\_CLEARDATAOUT)

CMP R5,#12 @ Check index, if index >12 -> index = 0

MOVEQ R5,#0x0 @ If equals -> Reset index = 0

ADDNE R5,R5,#4 @ If not, increment Pointer by 4

B LOOP @ Jump to LOOP

@ Procedure for rotating 4 USR LEDS

**LED\_on:**

STMFD R13!,{R2-R5,R14} @ Store uses registers on Stack

ADD R2,R0,#0x194 @ Load address of GPIO1\_SetDataOut

LDR R3,[R1,R5] @ Load value for turn in USR LED Depending on R5 (Index)

STR R3,[R2] @ Turn on led by storing the value into GPIO1\_SET\_DATA\_OUT

LDMFD R13!,{R2-R5,R14} @ Restore saved resigisters

MOV PC,R14 @ Return to mainline

@ Procedure for delaying 2s

**Delay:**

STMFD R13!,{R9,R14} @ Save uses registers

LDR R9,=0x00400000 @ Intialize delay loop counter

**NEXT:**

SUBS R9,R9,#0x1 @ Decrement loop counter

BNE NEXT @ Until loop counter equal 0

LDMFD R13!,{R9,R14} @ Restore values for saved registers

MOV PC, R14 @ Return to mainline

**.data**

**LED\_Display:** **.word** 0x00200000, 0x00400000, 0x00800000, 0x01000000 @ Values for turn in 4 USR LEDS

@ 0x00200000: USR1 GPIO1\_21 on

@ 0x00400000: USR2 GPIO1\_22 on

@ 0x00800000: USR3 GPIO1\_23 on

@ 0x01000000: USR4 GPIO1\_24 on

**STACK:** .rept 256

**.byte** 0x00

.endr

.end

**PART II: ROTATING LED AND INTERRUPT**

1. **Standard Program Structure**
2. **Mainline**

* Initialize STACK
* Initialize GPIO1

Initial base address for GPIO1 (0x4804C000)

Set GPIO1 bits 21-24 to low by write 0x01E00000 to GPIO1\_CLEARDATAOUT at 0x4804C190 (0x4804C000 + 0x190 offset for GPIO\_CLEARDATAOUT)

Set GPIO1 bits 21-24 to output by RMW 0xFE1FFFFF to GPIO1\_OE at 0x4804C134 (0x4804C000 + 0x134 offset for GPIO\_OE)

Set GPIO1\_31 to detect falling edge by RMW 0x80000000 to GPIO1\_FALLINGDETECT at 0x4804C14C (0x4804C000 + 0x14C offset for GPIO\_FALLINGDETECT)

Enable GPIO\_31 request on GPIO1\_IRQSTATUS\_SET\_0 by write 0x80000000 to 0x4804C034

* Initialize INT

Enable GPIO\_1 interrupt by enable Int number 98 of the INTC (write 0x04 to 0x482000E8: 0x48200000 (base address for INTC) + 0xE8 offset for INTC\_MIR\_CLEAR3)

* Initialize array for 4 USER LEDS to display ( LED\_Display )
* Initialize a register equal 0 ( Register control index to get value from pointer of status 4 LED from LED\_Display)
* Enable the Processor IRQ

Copy current value in CPSR into a register

Clear bit 7 of the current CPSR

Write the modified result back to CPSR ( 8 bit lowest)

* Repeat ( Loop forever and wait for interrupt signal)

Load Flag Pointer

Read data from flag pointer

If ( data ==1)

Go back to Loop (forever) and waiting for signal interrupt changes flag

Else

Call Procedure Turn\_on ( Same part 1)

Call Procedure Delay 2 seconds ( Same part 1)

Clear 4 LED ( Write 0x01E00000 into GPIO\_CLEARDATAOUT)

Compare Register ( store index of LED\_Display) with 12

If (Register == 12)

Reset register = 0 ( Back to 1st address of LED\_Display)

Else

Increment register by 4

* Repeat Until N/A ( Endless Loop)

1. **Interrupt Procedure (INT\_DIRECTOR)**

* Saved uses register and linked register on Stack
* Check if the interrupt come from GPIO1 by test bit 2 (int number 98) of the current value stored in INTC\_PENDING\_IRQ3 (0x482000F8: 0x48200000 (base address for INTC) + 0xF8 offset for INTC\_PENDING\_IRQ3)
* If the interrupt not come from GPIO\_1

Restore save register

Restore CPSR

Go to wait Loop ( Use special instruction: SUBS PC,LR,#4)

* Else

Check if the interrupt comes from GPIO1 bit 31 by test bit 31 (GPIO1\_31) of the current value store in GPIO1\_IRQSTATUS\_0 (0x4804C02C: 0x4804C000 + 0x2C offset for GPIO\_IRQSTATUS\_0).

If the interrupt come from GPIO\_1 bit 31

Go to BUTTON\_SVC

Else

Restore saved register

Return from IRQ interrupt procedure

1. **Button Service Procedure (if the switch is pressed)**

* Turn off GPIO1\_31 interrupt request by write 0x80000000 to GPIO1\_IRQSTATUS\_0 (0x4804C02C: 0x4804C000 + 0x2C offset for GPIO\_IRQSTATUS\_0)
* Enable new IRQ generation by write 0x1 to INTC\_CONTROL (0x48200048: 0x48200000 (base address for INTC) + 0x48 offset for INTC\_CONTROL)
* Load Pointer of Flag to check current state
* Read data from Flag memory
* If ( Flag ==1)

Assign Flag = 0 for next press

Store it into flag memory

Go to Done!

* Else

Assign Flag = 1 for next press

Store it into flag memory

* Clear 4 LED ( Write 0x01E00000 into GPIO\_CLEARDATAOUT)
* Reset Register control LED\_Display
* Restore saved register
* Return from IRQ interrupt procedure ( SUBS PC, LR, #4)

1. **Initialize Data:**

* LED\_Display: ( Same Part 1)
* Initial STACK space
* Flag: .word 0x0 ( 0 🡪 Rotating LED, 1🡪 Turn off)

1. **Source code:**

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@ ECE 371 – PROGRAMMING PROJECT II

@----------------------------------------------------------------------------

@ INTERRUPT CONTROLLER

@ In the second part of this project, I will develop an interrupt procedure that

@ services an interrupt request from a debounced, push-button switch. The first

@ time the button is pushed, the interrupt procedure will start the rotating LED @ display. The next time the button is pushed, the interrupt procedure will stop @ the rotating LED display.

@----------------------------------------------------------------------------

@ Hai Dang Hoang

@ Reference: Douglas V. Hall and Leela Yadlapalli

@ DECEMBER 18th, 2014

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

**.text**

**.global** \_start

**.global** INT\_DIRECTOR

**\_start:**

LDR R13, =STACK @ Point to base of Stack

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

LDR R0, =0x4804C000 @ Base Address for GPIO1 registers

ADD R4,R0,#0x190 @ Address of GPIO1\_ClearDATAOUt register

MOV R7,#0x01E00000 @ Load value to turn off LED on GPIO1\_21-24

STR R7,[R4] @ Write to GPIO1\_ClearDataOUt register

@ Program GPIO1\_21-24 as output

ADD R1,R0,#0x0134 @ Make GPIO1\_OE register address

LDR R6,[R1] @ Read current GPIO1 Output Enable register

LDR R7, =0xFE1FFFFF @ Word to enable GPIO1\_21-24 as output (0 enables)

AND R6,R7,R6 @ Clear bit 21-24 ( Modify)

STR R6, [R1] @ Write to GPIO1 Output Enable register

@ Detect falling edge on GPIO1\_31 and enable to assert POINTRPEND1

ADD R1, R0, #0x14C @ R1 = address of GPIO1\_FallingDetect register

LDR R2,=0x80000000 @ Load value for bit 31

LDR R3,[R1] @ Read GPIO1\_FallingDetect register

ORR R3,R3,R2 @ Modify ( set bit 31)

STR R3,[R1] @ Write back

ADD R1,R0,#0x34 @ Create address of GPIO1\_IRQSTATUS\_SET\_0 register

STR R2,[R1] @ enable GPIO1\_31 request on POINTRPEND1

@ Initialize INTC

LDR R1, =0x482000E8 @ Address of INTC\_MIR\_Clear3 register

MOV R2,#0x04 @ Value to unmask INTC INT 98 GPIOINT1A

STR R2,[R1] @ Write to INTC\_MIR\_CLEAR3 register

@ Load status LED\_Display

LDR R1,=LED\_Display @ Initialize pointer array for turning 4 USR LEDS on

MOV R5, #0x0 @ R5 Register(Index) control status of LED\_Display

LDR R7,=0x01E00000 @ Value for clear all 4 USR LEDS

@ 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

@ Wait for interrupt

**LOOP:**

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

LDR R3,[R2] @ Load value from Flag memory

CMP R3,#0x1 @ Compare with 1

BEQ LOOP @ Jump again LOOP if flag equals 1, Else rotating LED

BL LED\_on @ 4 USR LEDS run from right to left (GPIO1.21 -> GPIO1.24)

BL Delay @ Call delay 2s Procedure

STR R7,[R4] @ Clear all 4 USR LEDS ( R4 current stores address of GPIO1\_CLEARDATAOUT)

CMP R5,#12 @ Check index, if index >12 -> index = 0

MOVEQ R5,#0x0 @ If equals -> Reset index = 0

ADDNE R5,R5,#4 @ If not, increment Pointer by 4

B LOOP

@ Procedure for rotating 4 USR LEDS

**LED\_on:**

STMFD R13!,{R2-R5,R14} @ Store uses registers on Stack

ADD R2,R0,#0x194 @ Load address of GPIO1\_SetDataOut

LDR R3,[R1,R5] @ Load value for turn in USR LED Depending on R5 (Index)

STR R3,[R2] @ Turn on led by storing the value into GPIO1\_SET\_DATA\_OUT

LDMFD R13!,{R2-R5,R14} @ Restore saved resigisters

MOV PC,R14 @ Return to mainline

@ Procedure for delaying 2s

**Delay:**

STMFD R13!,{R9,R14} @ Save uses registers

LDR R9,=0x00400000 @ Intialize delay loop counter

**NEXT:**

SUBS R9,R9,#0x1 @ Decrement loop counter

BNE NEXT @ Until loop counter equal 0

LDMFD R13!,{R9,R14} @ Restore values for saved registers

MOV PC, R14 @ Return to mainline

**INT\_DIRECTOR:**

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

LDR R0,=0x482000F8 @ Address 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 back to wait loop, Else

LDR R0,=0x4804C02C @ Load GPIO1\_IRQSTATUS\_0 register address

LDR R1,[R0] @ Read Status register

TST R1,#0x80000000 @ Check if bit 31=1

BNE BUTTON\_SVC @ If bit 31=1, then button pushed

BEQ PASS\_ON @ If bit 31=0, then go to back to wait loop

**PASS\_ON:**

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

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

**BUTTON\_SVC:**

LDR R1,=0x80000000 @ Value to turn off GPIO1\_31 Interrupt request

@ This will turn off INTC interrupt request also

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 @ Address of INTC\_CONTROL register

MOV R1,#0x1 @ Value to clear bit 0

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

MOV R1,#0x01E00000 @ Load value to turn ON or OFF for 4 USER LED

LDR R0,=Flag @ Load pointer of Flag to check current state

LDR R3,[R0] @ Load value from Flag memory

CMP R3,#0x1 @ Compare with 1

MOVEQ R3,#0x0 @ If equals, Change flag to 0 for next press

STREQ R3,[R0] @ Store it into memory Flag

BEQ Done @ Jump to Done

MOV R3,#0x1 @ If not equals, Change flag to 1 for next press

STR R3,[R0] @ Store it into memory Flag

LDR R0,=0x4804C190 @ Load address of GPIO1\_CLEARDATAOUT register

STR R1,[R0] @ Write 0x01E00000 to GPIO1\_CLEARDATAOUT register

MOV R5,#-4 @ Reset Register control LED\_Display

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

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

**.align** 2

**.data**

**LED\_Display:** **.word** 0x00200000, 0x00400000, 0x00800000, 0x01000000 @ Values for turn in 4 USR LEDS

@ 0x00200000: USR1 GPIO1\_21 on

@ 0x00400000: USR2 GPIO1\_22 on

@ 0x00800000: USR3 GPIO1\_23 on

@ 0x01000000: USR4 GPIO1\_24 on

**Flag:** **.word** 0x0 @ Aside a memory to check condition

@ 0 -> LED turn on ( Rotating)

@ 1 -> LED turn off

**.align** 2

**STACK:** .rept 1024

**.word** 0x0

.endr

.end

**PART III: IMPROVEMENT FOR PART II**

**States:**

I use the Flag memory that set aside to control many states.

00: Rotating LED

01: Turn off

10: LED count up by binary

11: LED count down by binary

**Improvement for LED array for counting binary:**

In the original program, LED array is initialized such that the first value is used for set GPIO1\_21 high, the second value is used for set GPIO1\_22 high, the third value is used for set GPIO1\_23 high and the last value is used for set GPIO1\_24 high.

**LED\_display:** .word 0x00200000, 0x00400000, 0x00800000, 0x01000000

In this program, I add more array for controlling LED for counting binary. By using the GPIO template Figure 4-14 in Chapter 4, LED\_Display\_Binary array for binary counter will be:

**LED\_Display\_Binary:** **.word** 0x00000000, 0x00200000, 0x00400000, 0x00600000 @0000-0011

**.word** 0x00800000, 0x00A00000, 0x00C00000, 0x00E00000 @0100-0111

**.word** 0x01000000, 0x01200000, 0x01400000, 0x01600000 @1000-1011

**.word** 0x01800000, 0x01A00000, 0x01C00000, 0x01E00000 @1100-1111

1. **Standard Program Structure**
2. **Mainline**

* Initialize STACK
* Initialize GPIO1

Initial base address for GPIO1 (0x4804C000)

Set GPIO1 bits 21-24 to low by write 0x01E00000 to GPIO1\_CLEARDATAOUT at 0x4804C190 (0x4804C000 + 0x190 offset for GPIO\_CLEARDATAOUT)

Set GPIO1 bits 21-24 to output by RMW 0xFE1FFFFF to GPIO1\_OE at 0x4804C134 (0x4804C000 + 0x134 offset for GPIO\_OE)

Set GPIO1\_31 to detect falling edge by RMW 0x80000000 to GPIO1\_FALLINGDETECT at 0x4804C14C (0x4804C000 + 0x14C offset for GPIO\_FALLINGDETECT)

Enable GPIO\_31 request on GPIO1\_IRQSTATUS\_SET\_0 by write 0x80000000 to 0x4804C034

* Initialize INT

Enable GPIO\_1 interrupt by enable Int number 98 of the INTC (write 0x04 to 0x482000E8: 0x48200000 (base address for INTC) + 0xE8 offset for INTC\_MIR\_CLEAR3)

* Initialize a register equal 0 ( Register control index to get value from pointer of status 4 LED from LED\_Display and LED\_Display\_Binary)
* Enable the Processor IRQ

Copy current value in CPSR into a register

Clear bit 7 of the current CPSR

Write the modified result back to CPSR ( 8 bit lowest)

* Repeat ( Loop forever and wait for interrupt signal)

Load Flag Pointer

Load value from Flag Pointer

If ( flag == 0)

Load Pointer LED\_Display

Call procedure Rotating LED

Else if ( flag ==1)

Call Procedure turn off

Else if ( flag == 2)

Load LED\_Display\_Binray pointer

Call Procedure Count up

Else

Call Procedure Count down

* Until N/A( Endless Loop)

1. **Interrupt Procedure (INT\_DIRECTOR)**

* Saved uses register and linked register on Stack
* Check if the interrupt come from GPIO1 by test bit 2 (int number 98) of the current value stored in INTC\_PENDING\_IRQ3 (0x482000F8: 0x48200000 (base address for INTC) + 0xF8 offset for INTC\_PENDING\_IRQ3)
* If the interrupt not come from GPIO\_1

Restore save register

Restore CPSR

Go to wait Loop ( Use special instruction: SUBS PC,LR,#4)

* Else

Check if the interrupt comes from GPIO1 bit 31 by test bit 31 (GPIO1\_31) of the current value store in GPIO1\_IRQSTATUS\_0 (0x4804C02C: 0x4804C000 + 0x2C offset for GPIO\_IRQSTATUS\_0).

If the interrupt come from GPIO\_1 bit 31

Go to BUTTON\_SVC

Else

Restore saved register

Return from IRQ interrupt procedure

1. **Button Service Procedure (if the switch is pressed)**

* Turn off GPIO1\_31 interrupt request by write 0x80000000 to GPIO1\_IRQSTATUS\_0 (0x4804C02C: 0x4804C000 + 0x2C offset for GPIO\_IRQSTATUS\_0)
* Enable new IRQ generation by write 0x1 to INTC\_CONTROL (0x48200048: 0x48200000 (base address for INTC) + 0x48 offset for INTC\_CONTROL)
* Load Pointer of Flag to check current state
* Read data from Flag memory
* If ( Flag ==0)

Assign Flag = 1 for next press

Store it into flag memory

Go to Done!

* Else if ( Flag == 1)

Assign Flag = 2

Store it into flag memory

Go to Done!

* Else if ( Flag == 2)

Assign Flag = 3

Store it into flag memory

Go to Done!

* Else

Assgin Flag = 0

Store it into memory

* Restore saved register
* Return from IRQ interrupt procedure ( SUBS PC, LR, #4)

1. **Initialize Data:**

* LED\_Display: ( Same Part 1)
* LED\_Display\_Binary
* Initial STACK space
* Flag: .word 0x0

**II. Source code:**

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@ ECE 371 – PROGRAMMING PROJECT II

@----------------------------------------------------------------------------

@ INTERRUPT CONTROLLER ( Extra Credit)

@ In the third part of this project, I will develop an interrupt procedure that

@ services an interrupt request from a debounced, push-button switch. There are

@ 4 states for pressing:

@ 00 -> LEDs rotate .

@ 01 -> LED turn off.

@ 10 -> LED count up by binary

@ 11 -> LED count down binary

@ After 4th pressing,if I continue to press the button ( 5th press), it will return to @ the State 00 ( LEDS rotate).

@

@----------------------------------------------------------------------------

@ Hai Dang Hoang

@ Reference: Douglas V. Hall and Leela Yadlapalli

@ DECEMBER 18th, 2014

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

**.text**

**.global** \_start

**.global** INT\_DIRECTOR

**\_start:**

LDR R13, =STACK @ Point to base of Stack

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

LDR R0, =0x4804C000 @ Base Address for GPIO1 registers

ADD R4, R0, #0x190 @ Address of GPIO1\_ClearDATAOUt register

MOV R7, #0x01E00000 @ Load value to turn off LED on GPIO1\_21

STR R7,[R4] @ Write to GPIO1\_ClearDataOUt register

@ Program GPIO1\_21-24 as output

ADD R1,R0,#0x0134 @ Make GPIO1\_OE register address

LDR R6,[R1] @ Read current GPIO1 Output Enable register

LDR R7, =0xFE1FFFFF @ Word to enable GPIO1\_21-24 as output ( 0 enables)

AND R6,R7,R6 @ Clear bit 21-24 ( Modify)

STR R6, [R1] @ Write to GPIO1 Output Enable register

@ Detect falling edge on GPIO1\_31 and enable to assert POINTRPEND1

ADD R1, R0, #0x14C @ R1 = address of GPIO1\_FallingDetect register

LDR R2,=0x80000000 @ Load value for bit 31

LDR R3,[R1] @ Read GPIO1\_FallingDetect register

ORR R3,R3,R2 @ Modify ( set bit 31)

STR R3, [R1] @ Write back

ADD R1,R0,#0x34 @ Create address of GPIO1\_IRQSTATUS\_SET\_0 register

STR R2,[R1] @ enable GPIO1\_31 request on POINTRPEND1

@ Initialize INTC

LDR R1, =0x482000E8 @ Address of INTC\_MIR\_Clear3 register

MOV R2,#0x04 @ Value to unmask INTC INT 98 GPIOINT1A

STR R2,[R1] @ Write to INTC\_MIR\_CLEAR3 register

@ Load status LED\_Display

MOV R5, #0x0 @ R5 Register(Index) control status of LED\_Display array

LDR R7,=0x01E00000 @ Value for clear all 4 USR LEDS

@ 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

@ Wait for interrupt

**LOOP:**

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

LDR R3,[R2] @ Load value from Flag memory

CMP R3,#0x0 @ Compare with 0

LDREQ R1, =LED\_Display @ Load Pointer LED\_display for rotating LED

BLEQ RotatingLED @ Call Procedure Rotating LED

CMP R3,#0x01 @ Compare with 1 ( Check Pressing button for LED turn of

BLEQ Turn\_offLED @ If equals, turn off LED

CMP R3,#0x02 @ Compare with 2 ( Check Pressing button for LED turn on by binary)

LDREQ R1, =LED\_Display\_Binary @ Load Pointer LED\_display\_Binary for count on by binary

BLEQ Count\_up @ If Equals -> Count up by binary

BLNE Count\_down @ Else, Count down by binary

B LOOP

**RotatingLED:**

BL LED\_on @ Call Procedure LED turn on

BL Delay @ Call delay 2s procedure

STR R7,[R4] @ Clear all 4 USR LEDS

CMP R5,#12 @ Check index, if index >12 -> index = 0

MOVEQ R5,#0 @ If equals -> Reset index =0

ADDNE R5,R5,#4 @ If not, increment Pointer by 4

B LOOP

**Turn\_offLED:**

STR R7,[R4] @ Clear all 4 USR LEDS

B LOOP

**Count\_up:**

BL LED\_on @ Call Procedure LED turn on

BL Delay @ Call delay 2s procedure

STR R7,[R4] @ Clear all 4 USR LEDS

CMP R5,#60 @ Check index, if index >60 (15x4) -> index = 0

MOVEQ R5,#0 @ If equals -> Reset index =0

ADDNE R5,R5,#4 @ If not, increment Pointer by 4

B LOOP

**Count\_down:**

BL LED\_on @ Call Procedure LED turn on

BL Delay @ Call delay 2s procedure

STR R7,[R4] @ Clear all 4 USR LEDS

CMP R5,#0 @ Check index, if index=0 -> index = 60

MOVEQ R5,#60 @ If equals -> Reset index =60

SUBNE R5,R5,#4 @ If not, Decrement Pointer by 4

B LOOP

@ Procedure for rotating 4 USR LEDS

**LED\_on:**

STMFD R13!,{R2-R5,R14} @ Store uses registers on Stack

ADD R2,R0,#0x194 @ Load address of GPIO1\_SetDataOut

LDR R3,[R1,R5] @ Load value for turn in USR LED Depending on R5 (Index)

STR R3,[R2] @ Turn on led by storing the value into GPIO1\_SET\_DATA\_OUT

LDMFD R13!,{R2-R5,R14} @ Restore saved resigisters

MOV PC,R14 @ Return to mainline

@ Procedure for delaying 1s

**Delay:**

STMFD R13!,{R9,R14} @ Save uses registers

LDR R9,=0x00400000 @ Intialize delay loop counter

**NEXT:**

SUBS R9,R9,#0x1 @ Decrement loop counter

BNE NEXT @ Until loop counter equal 0

LDMFD R13!,{R9,R14} @ Restore values for saved registers

MOV PC, R14 @ Return to mainline

@ Interrupt

**INT\_DIRECTOR:**

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

LDR R0,=0x482000F8 @ Address 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 back to wait loop, Else

LDR R0,=0x4804C02C @ Load GPIO1\_IRQSTATUS\_0 register address

LDR R1,[R0] @ Read Status register

TST R1,#0x80000000 @ Check if bit 31=1

BNE BUTTON\_SVC @ If bit 31=1, then button pushed

BEQ PASS\_ON @ If bit 31=0, then go to back to wait loop

**PASS\_ON:**

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

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

**BUTTON\_SVC:**

LDR R1,=0x80000000 @ Value to turn off GPIO1\_31 Interrupt request

@ This will turn off INTC interrupt request also

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 @ Address of INTC\_CONTROL register

MOV R1,#0x1 @ Value to clear bit 0

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

MOV R1,#0x01E00000 @ Load value to turn ON or OFF for 4 USR LED

LDR R0,=Flag @ Load pointer of Flag to check current state

LDR R3,[R0] @ Load value from Flag memory

CMP R3,#0x0 @ Compare with 0

MOVEQ R3,#0x1 @ If equal, Change flag to 1 ( State 1)

STREQ R3,[R0] @ Store it into memory flag

BEQ Done @ Done

CMP R3,#0x01 @ Check flag = 1 or not

MOVEQ R3,#0x2 @ If equal, Change Flag to 2 ( State 2)

STREQ R3,[R0] @ Store it in Memory

MOVEQ R5,#0 @ Reset R5 register for controlling display LED

BEQ Done @ Done

CMP R3,#0x02 @ Check flag = 2 or not

MOVEQ R3,#0x3 @ If equal, Change Flag to 3 ( State 3)

STREQ R3,[R0] @ Store it in Memory

SUBEQ R5,R5,#4 @ Need to subjact R5 by 4 because it added 4 from State 2 before checking

BEQ Done @ Done

MOV R3,#0x0 @ Currentlly, flag=3 and change to 0 ( State 0)

STR R3,[R0] @ Store it in memory.

MOV R5,#-4 @ Reset R5 register for controlling display LED

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

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

**.align** 2

**.data**

**LED\_Display:** **.word** 0x00200000, 0x00400000, 0x00800000, 0x01000000

@ Values for turn in 4 USR LEDS

@ 0x00200000: USR1 GPIO1\_21 on

@ 0x00400000: USR2 GPIO1\_22 on

@ 0x00800000: USR3 GPIO1\_23 on

@ 0x01000000: USR4 GPIO1\_24 on

**LED\_Display\_Binary:** **.word** 0x00000000, 0x00200000, 0x00400000, 0x00600000

**.word** 0x00800000, 0x00A00000, 0x00C00000, 0x00E00000

**.word** 0x01000000, 0x01200000, 0x01400000, 0x01600000

**.word** 0x01800000, 0x01A00000, 0x01C00000, 0x01E00000

**Flag:** **.word** 0x0 @ Aside a memory to check condition (State)

@ 00 -> LEDs rotate

@ 01 -> LED turn off

@ 10 -> LED count up binary

@ 11 -> LED count down binary

**.align** 2

**STACK:** .rept 1024

**.word** 0x0

.endr

.end

**PART IV: TIMER**

**Introduction:**

In the part 1. I produce a delay 2 seconds with a delay loop. To implement a delay loop, I simply load a calculated value in a register and cycled execution around a tight loop that just decrements the value in the register down to zero. The processor clock cycles required to cycle around the loop multiple times produces a delay. There are many problems with the delay loop method of producing a desired delay. Therefore, in this part, I will use interrupt of Timer to procedure exactly 0.5 second from real life. The program simply toogle the LED exactly 0.5 second.

**Discussion:**

1. INTC Initialization for timer:

The INTC can handle interrupt requests from up to 128 sources. It divided into four, 32-bit banks from MIR0-MIR3. The main initialization step I have to do in the INTC for a basic system is to unmask the interrupt coming from the Timer module. According to the ARM Cortex A-8 Interrupts numbers table in Appendix E, the POINTR\_PEND signal from Timer module is connected to Int number 66 of INTC. Ints #64-#95 are enabled/disabled by the corresponding bits in the MIR2 register. To unmask bit 2 of the MIR2 register, I simply write 0x00000004 to the INTC\_MIR\_SET2 register at offset C8.

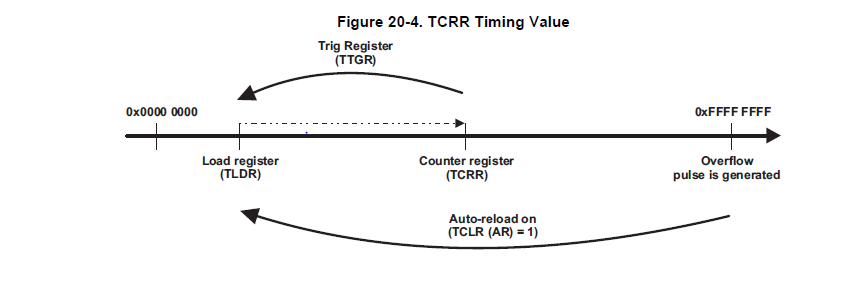
1. Idea about using timer ( Function):

The general-purpose timer is an upward counter. It supports three functional modes: Timer mode, capture mode, and compare mode. In this program, I will use the Timer mode. Specifically, the timer can be started and stopped at any time through the Timer Control Register (TCLR ST bit). The Timer Counter Register (TCRR) can be loaded when stopped or on the fly ( while counting). TCRR can be loaded directly by a TCRR Write access with the new timer value.

In the one shot mode ( TCLR AR bit = 0), the counter is stopped after counting overflow ( counter value remains at zero).

When the auto-reload mode is enabled ( TCLR AR bit =1), the TCRR is reloaded with the TLDR value after a counting overflow

An interrupt can be issued on overflow if the overflow interrupt enable bit is set in the timer Interrupt Enable Register ( TIER OVF\_IT\_ENA bit = 1).



My idea is that I initial value for TCRR (Counter register), I need to find exactly the value for TCRR to create delay 0.5 second. It will count from TCRR to 0xFFFFFFFF. When it reaches 0xFFFFFFFF, overflow interrupt signal based on OVF\_IT\_ENA flag sent. In this time, the new value from TCLR will auto reload the new value for TCRR in order to count for next time. In order to find exactly the value for TCRR, it is as follows:

( 0xFFFFFFFF – TCRR +1) x Timer Clock Period x Clock Divider

Or :

( 0xFFFFFFFF – TCRR +1)/ 32000 = 0.5 second

* TLDR = 0xFFFFC180

1. **Standard Program Structure and Algorithm:**

I only discuss about initial timer register and interrupt part because initial GPIO is the same the previous part.

1. **Mainline**

* Base address of Timer register: 0x44E05000
* Reset software before using by storing bit 1 into TIOCP\_CFG register ( 10h offset)
* Enable timer interrupt by storing 0x4 into INTC\_MIR\_CLEAR2 Register
* Initial value 0xFFFFC180 for Timer TLDR and Timer TCRR register for counter
* Enable overflow interrupt timer by store 2 into TIME\_IRQENABLESET ( 2Ch offset)
* Set Auto-reload timer by storing 3 into Time\_TCLR (38h offset)

1. **Interrupt Procedure:**

* Load Address of INTC\_PENDING\_IRQ2
* Read INTC\_PENDING\_IRQ2 register
* Test bit 2
* If not from POINTR-PEND 🡪 Go to PASS\_ON
* Else

Load value from Timer IRQ\_Status

If Overflow ( OVF\_EN\_FLAG == 1)

Go to IRQ\_timer\_exit

Else

Toogle LED GPIO1\_21

Label: IRQ\_timer\_exit:

* Clear timer interrupts by storing bit 7 into IRQSTATUS

Label: PASS\_ON:

* Turn off INTC\_Control, processor respond to new IRQ by storing bit 1 into INTC\_Control register
* Restore register from Stack
* SUBS PC, LR,#4.

**II. Source code:**

@ Timer

@ Runs on BeagleBone Black Board with cape.

@ Toogle LED Exactly 0.5s Using Timer interrupt

@ Copyright Douglas V. Hall Fall 2014

@ Hai Dang Hoang

.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

CPS #0x13 @ Back to SVC mode

LDR R0,=0x4804C000 @ Base address for GPIO1 registers

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

MOV R7, #0x00200000 @ Load value to turn off LED on GPIO1\_21

STR R7,[R4] @ Write to GPIO1\_CLEARDATAOUT register

@ Program GPIO1\_21 as output

ADD R1,R0,#0x0134 @ Make GPIO1\_OE register address

LDR R6,[R1] @ READ current GPIO1 Output Enable register

LDR R7,=0xFFDFFFFF @Word to enable GPIO1\_21 as output(0 enables)

AND R6,R7,R6 @ Clear bit 12 (MODIFY)

STR R6, [R1] @ WRITE to GPIO1 Output Enable register

@ Initialize INTC

@ Reset

LDR R1, =0x48200010 @ INTC\_SYSCONFIG

MOV R2,#2 @ Value for reset

STR R2,[R1] @ Software Reset

@ Enable timer interrupt

LDR R1,=0x482000C8 @ Address of INTC\_MIR\_CLEAR2 register

MOV R2,#0x04 @Value to unmask INTC INT 66, Timer0, POINTR\_PEND

STR R2,[R1] @ Write to INTC\_MIR\_CLEAR2 register

@ Initialize Timer 0

LDR R1, =0x44E05000 @ Base address of Timer

@ Reset

MOV R2,#1 @ Value for reset

STR R2,[R1,#0x10] @ Store Timer Config ( TIOCP\_CFG)

@ Prime the timer

LDR R2, =0xFFFFC180 @ 0.5 second count from 0xFFFFC180 to 0xFFFFFFFF

STR R2, [R1,#0x40] @ Store it into Timer TLDR

STR R2, [R1,#0x3C] @ Store it into Timer TCRR

@ Enable overflow interrupt

MOV R2,#2

STR R2,[R1,#0x2C] @ Store it Timer (IRQENABLE\_SET)

@ Start Auto-reload timer

MOV R2,#3 @ Auto-reload, start

STR R2,[R1,#0x38] @ Store it in Time TCLR

@ 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

@ Wait for interrupt

LOOP: NOP

B LOOP

INT\_DIRECTOR:

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

@ Check that this is the timer interrupt

LDR R0,=0x482000D8 @ Address of INTC-PENDING\_IRQ2 register

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

TST R1,#0X00000004 @ TEST BIT 2

BEQ PASS\_ON @ Not from POINTR\_PEND, go to back to wait loop, Else

@ Check that this is the overflow interrupt

LDR R0, =0x44E05000

LDR R1,[R0,#0x28] @ Load value from Timer IRQ Status

TST R1, #2 @ Check Overflow

BEQ IRQ\_timer\_exit

@ Toogle LED USR1

LDR R3,=0x00200000

LDR R1,=Counter @ Load Counter

LDR R2,[R1]

TST R2,#1

LDREQ R0,=0x4804C194 @ SetDataOut 1-> LED is on

STREQ R3,[R0]

LDRNE R0,=0x4804C190 @ ClearDataout 0-> LED is off

STRNE R3,[R0]

ADD R2,R2,#1 @ Add 1

STR R2,[R1] @ Store it into counter

IRQ\_timer\_exit:

@ Clear timer interrupts

LDR R1, =0x44e05028 @ Time0\_IRQ Status (IRQSTATUS)

LDR R2, [R1]

MOV R2,#7 @ Capture, overflow , and match

STR R2, [R1]

PASS\_ON:

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

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

MOV R1, #01 @ Value to clear bit 0

STR R1,[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

.align 2

SYS\_IRQ: .WORD 0 @ Location to store systems IRQ address

.data

.align 2

Counter: .word 0x0

STACK1: .rept 1024

.word 0x0000

.endr

STACK2: .rept 1024

.word 0x0000

.endr

.END

**Signed Statement**

I developed and wrote this program by myself with no help from anyone except the instructor and the T.A. and I did not give help to anyone else