|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | ***Portland State University*** [Maseeh College of Engineering and Computer Science](http://pdx.edu/cecs/) 1559170_300.jpg | | ECE 372 - PROJECT REPORT | | Project 2: UART | | **2/20/2015** | |
|  |
| **Instructor: Douglas V.Hall**  **TA: Leela Yadlapalli** |
| Students: Hai Dang Hoang  PSUID: 946749560  Email: [danghai@pdx.edu](mailto:danghai@pdx.edu) |

**Introduction:**

Specialized processors that convert an ASCII text string to spoken words are now available as chipsets or as stand-alone modules. The chipsets can be connected directly on the microprocessor buses in an embedded system and with that connection the ASCII string can be written directly to the chipset. For the stand-alone modules, the ASCII strings are usually sent over an RS-232C or I2C serial cable. For communicating with the RC Systems 8660 Text-to-speech module we have chosen for this exercise, I will use the UART5 device on a BeagleBone Black board that has an adapter to work with the RS232C signal levels.

Other than to have a little fun with a microprocessor, the major goals of this exercise are to give me some practice working with a programmable peripheral device that has multiple interrupt sources and some more practice using the **R3T3SD** process to successfully work my way through a new design project that is assigned to me on the job.

Specifically, the particular product you are going to be working on provides a verbal count down for an amateur rocket launcher and ignites the rocket thruster. After a button is pushed, the launcher program counts down “10, 9,8,7,6,5,4,3,2,1,0” at 1 second intervals and sends each count to the Speech Synthesizer. Immediately after it says 0, the program sends a signal that turns on an LED connected to GPIO1\_12 to indicate ignition

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

**PART I**

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

1. **Mainline**

* INITIALIZE STACK
* INITIALIZE GPIO

Set up GPIO1\_31 for detecting falling edge by storing 0x80000000 to 0x4804C14C (0x4804C000 (GPIO1\_Base) + 0x14C (offset of GPIO\_FALLINGEDGE\_DETECT)

Set up GPIO1\_31 for IRQ enable by storing 0x80000000 to 0x4804C034 (0x4804C000 (GPIO1\_Base) + 0x34 (offset of GPIO\_IRQSTATUS\_SET0)

* 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).

Enable UART5 interrupt by writing 0x4000 to 0x482000A8 (0x4000 for unmasking INT 46, 0x482000A8 = 0x48200000 (INTC\_Base) + 0xA8 ( Offset of INTC\_MIR\_CLEAR1).

* CONTROL MODULE

Change pin 37 on P8 to mode 4 by writing #4 to 0x44E108C0 (0x44E10000 (Control Module Base) + 0x8C0 (offset of pin 37))

Change pin 38 on P8 to mode 4 by writing #4 to 0x44E108C4 (0x44E10000 (Control Module Base) + 0x8C4 (offset of pin 38))

Change pin 31 on P8 to mode 6 by writing #6 to 0x44E108D8 (0x44E10000 (Control Module Base) + 0x8D8 (offset of pin 31))

Change pin 32 on P8 to mode 6 by writing #6 to 0x44E108DC (0x44E10000 (Control Module Base) + 0x8DC (offset of pin 32))

* INITIAL UART5

1. **Set up Baud rate for Transmitting**

Enable DLL and DLH access by writing 0x83 to 0x481AA00C (0x83 to enable Divisor latch and select 8 bit word length, 0x481AA00C = 0x481AA000 (UART Base) + 0x0C (offset of LCR register))

Choose UART 16x Mode by writing 0x00 to 0x481AA020 (0x00 to select UART 16x Mode, 0x481AA020 = 0x481AA000 (UART Base) + 0x20 (offset of MDR1 register)

Write 0x1A to 0x481AA000 (DLL register) and write 0x00 to 0x481AA004 (DLH register) to 115200 baud rate

1. **Initialize Tx and Rx Registers**

Clear bit 7 in LCR register to enable Tx and Rx registers

Enable THR and MSR interrupt by writing 0x0A to 0x481AA004 = 0x481AA000 (UART Base) + 0x04 (offset of IER\_UART register)

1. **Turn off FIFO**

Turn off FIFO by writing 0x00 to 0x481AA008 = 0x481AA000 (UART Base) + 0x08 (offset of FIFO Control Register).

* 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).

* LOOP forever ( Endless loop) : Wait for interrupt signal.

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

* Saved uses register and linked register on Stack
* Check if the interrupt come from UART5 by test bit 14 (int number 46) of the current value stored in INTC\_PENDING\_IRQ1 (0x482000B8: 0x48200000 (base address for INTC) + 0xB8 offset for INTC\_PENDING\_IRQ1)
* If the interrupt not come from UART ( bit 14 != 0)

Go to CheckButton procedure

* Else ( interrupt from UART)

Go to TLKR\_SVC ( Handshake for sending character)

* CheckButton:

Read INTC\_PENDING\_IRQ3 REGISTER at 0x482000F8, check bit 2

If bit 2 = 0

Go to PASS\_ON

Else, check if interrupt comes from GPIO1\_31

Read GPIO1\_IRQ\_STATUS REGISTER SET 0 at 0x4804C02C, check bit 31

If bit 31 = 0

Restore saved register

Return to wait loop

Else

Do Button\_SVC procedure

1. **Button Service Procedure :**

**Button\_SVC**

* **Turn off IRQ request from GPIO1\_31**

Writing 0x80000000 to 0x4804C02C (0x4804C000 (GPIO1\_base) + 0x2C (offset of GPIO1\_IRQ\_STATUS\_SET0))

* **Generate new IRQ generation**

Writing 0x1 to 0x48200048 (0x48200000 (INTC\_base) + 0x48 (offset of INTC\_CONTROL))

* **Enable THR and MSR interrupt**

Writing 0x0A to 0x481AA004 = 0x481AA000 (UART Base) + 0x04 (offset of IER\_UART register)

* **Restore saved registers and return to wait loop**

1. **TLKR\_SVC procedure**
2. **Save registers**
3. **Check CTS signal**

Read MSR register at 0x481AA018, check bit 4

If bit 4 = 0

{

Check THR status by reading LSR register at 0x481AA014, check bit 5

If bit 5 = 0

{

Restore saved registers

Return to mainline

}

Else

{

Disable Tx interrupt by writing 0x08 to 0x481AA004 (UART Base) + 0x04 (offset of IER\_UART register) to prevent spinning problem

}

Else,

{

Check THR status

Read LSR register at 0x481AA014, check bit 5

If bit 5 = 0

{

Restore saved registers

Return to mainline

}

Else, go to SEND to send a character

1. **SEND algorithm**

Enable THR and MSR interrupts by writing 0x0A to 0x481AA004 = 0x481AA000 (UART Base) + 0x04 (offset of IER\_UART register)

Get message store at CHAR\_PTR address

Get one character in the message and increment pointer by 1

Store the character in the THR register (0x481AA000)

Get the value store at CHAR\_COUNT address

Decrement the value by 1 and store it back to CHAR\_COUNT

Check if it is the last character by checking the current value after decrement

If the value is less than to zero counter

{

Restore the original message to CHAR\_PTR

Restore the original message’s length to CHAR\_COUNT

Disable THR interrupt by clearing bit 2 of IER\_UART (0x481AA004)

Restore saved registers

Return to mainline

}

Else, ( value > = 0 🡪 go back to mainline and continue sending next character)

Restore saved registers

Return to mainline.

**Data**

MESSAGE:

.byte 0xD ( 1st letter must be sent to the speaker)

// .byte 0x01, 0x39, 0x4F ( Command to change the voice, if you want)

.ascii “take me to your leader” (Message that the speaker will speak)

CHAR\_PTR: .word MESSAGE (Pointer for the message)

CHAR\_COUNT: .word 24 (Pointer for the message’s length counter)

.equ MESSAGE\_LEN, 24 ( Value to restore the counter)

**PART II**

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

1. **Mainline**

* INITIALIZE STACK
* INITIALIZE GPIO

Clear data out from GPIO1\_21 by writing 0x00200000 to 0x4804C190 (0x4804C000 (GPIO1 Base) + 0x190 (offset of GPIO\_CLEARDATAOUT)

Enable GPIO1\_21 as output by R-M-W 0xFFDFFFFF to 0x4804C134 (0x4804C000 (GPIO1 Base) + 0x134 (offset of GPIO\_OUTPUT\_ENABLE)

Set up GPIO1\_31 for detecting falling edge by storing 0x80000000 to 0x4804C14C (0x4804C000 (GPIO1\_Base) + 0x14C (offset of GPIO\_FALLINGEDGE\_DETECT)

Set up GPIO1\_31 for IRQ enable by storing 0x80000000 to 0x4804C034 (0x4804C000 (GPIO1\_Base) + 0x34 (offset of GPIO\_IRQSTATUS\_SET0)

* 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).

Enable UART5 interrupt by writing 0x4000 to 0x482000A8 (0x4000 for unmasking INT 46, 0x482000A8 = 0x48200000 (INTC\_Base) + 0xA8 ( Offset of INTC\_MIR\_CLEAR1).

Enable TIMER2 interrupt by writing 0x10 to 0x482000C8 (0x10 for unmasking INT 68, 0x482000C8 = 0x48200000 (INTC\_Base) + 0xC8 ( Offset of INTC\_MIR\_CLEAR2).

* INITIALIZATION TIMER2

Enable clock for TIMER2 by writing 0x2 to 0x44E00080 ( CM\_PER\_TIMER2\_CLKCTRL)

Reset the software by storing 0x1 to 0x48040010 ( 0x48040000 (Timer2\_base + 0x10 (offset of Timer OCP configuration)).

Setting value for TCRR by storing 0xFFFF8300 to 0x4804003C ( 0x48040000 (Timer2\_base + 0x3C (offset of Timer Counter Register)).

Setting value for TLDR by storing 0xFFFF8300 to 0x48040040 (0x48040000 (Timer2\_base) + 0x40 (offset of Timer Load Register)).

Enable auto reload by writing 0x3 to 0x48040038 (0x3 for enable Auto-reload timer and start the timer, 0x48040038 =0x48040000 (Timer2\_base) + 0x38 (offset of Timer Control Register)).

Enable IRQ overflow interrupt for TIMER 2 by writing 0x2 to 0x4804002C (0x2 to enable IRQ for Overflow, 0x4804002C = 0x48040000 (Timer2\_base) + 0x2C (offset of Timer Interrupt Enable Set Register)).

* CONTROL MODULE

Change pin 37 on P8 to mode 4 by writing #4 to 0x44E108C0 (0x44E10000 (Control Module Base) + 0x8C0 (offset of pin 37))

Change pin 38 on P8 to mode 4 by writing #4 to 0x44E108C4 (0x44E10000 (Control Module Base) + 0x8C4 (offset of pin 38))

Change pin 31 on P8 to mode 6 by writing #6 to 0x44E108D8 (0x44E10000 (Control Module Base) + 0x8D8 (offset of pin 31))

Change pin 32 on P8 to mode 6 by writing #6 to 0x44E108DC (0x44E10000 (Control Module Base) + 0x8DC (offset of pin 32))

* INITIAL UART5

1. **Set up Baud rate for Transmitting**

Enable DLL and DLH access by writing 0x83 to 0x481AA00C (0x83 to enable Divisor latch and select 8 bit word length, 0x481AA00C = 0x481AA000 (UART Base) + 0x0C (offset of LCR register))

Choose UART 16x Mode by writing 0x00 to 0x481AA020 (0x00 to select UART 16x Mode, 0x481AA020 = 0x481AA000 (UART Base) + 0x20 (offset of MDR1 register)

Write 0x1A to 0x481AA000 (DLL register) and write 0x00 to 0x481AA004 (DLH register) to 115200 baud rate

1. **Initialize Tx and Rx Registers**

Clear bit 7 in LCR register to enable Tx and Rx registers

Enable THR and MSR interrupt by writing 0x0A to 0x481AA004 = 0x481AA000 (UART Base) + 0x04 (offset of IER\_UART register)

1. **Turn off FIFO**

Turn off FIFO by writing 0x00 to 0x481AA008 = 0x481AA000 (UART Base) + 0x08 (offset of FIFO Control Register).

1. **Sending 0xD on UART THR ( Avoid 1s second delay after pressing button).**

Storing 0xD at address 0x481AA000 ( UART Transmit buffer)

* 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).

LOOP forever ( Endless loop) : Wait for interrupt signal

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

* Saved uses register and linked register on Stack
* Check if the interrupt come from GPIO by test bit 2 (int number 96) of the current value stored in INTC\_PENDING\_IRQ3 (0x482000F8: 0x48200000 (base address for INTC) + 0xF8 offset for INTC\_PENDING\_IRQ3).

If bit 2 =1 ( GPIO interrupt)

Jump to CheckButton

Else ( Not GPIO interrupt)

{

Check if the interrupt come from UART by test bit 14 (int number 46) of the current value stored in INTC\_PENDING\_IRQ1 (0x482000B8: 0x48200000 (base address for INTC) + 0xB8 offset for INTC\_PENDING\_IRQ1)

If bit 16 ==1 (UART interrupt)

Go to TLKR\_SVC ( Handshake for sending character)

Else

Go back to mainline

}

* CheckButton:

Check if interrupt comes from GPIO1\_31

Read GPIO1\_IRQ\_STATUS REGISTER SET 0 at 0x4804C02C, check bit 31

If bit 31 = 0 ( Not interrupt comes from GPIO1\_31)

Restore saved register

Return to wait loop

Else ( Interrupt comes from GPIO1\_31)

Do Button\_SVC procedure

1. **Button Service Procedure :**

**Button\_SVC**

* **Turn off IRQ request from GPIO1\_31**

Writing 0x80000000 to 0x4804C02C (0x4804C000 (GPIO1\_base) + 0x2C (offset of GPIO1\_IRQ\_STATUS\_SET0))

* **Generate new IRQ generation**

Writing 0x1 to 0x48200048 (0x48200000 (INTC\_base) + 0x48 (offset of INTC\_CONTROL))

* **Enable THR and MSR interrupt**

Writing 0x0A to 0x481AA004 = 0x481AA000 (UART Base) + 0x04 (offset of IER\_UART register)

* **Enable Timer interrupt by storing 0x03 at address 0x48040038**
* **Turn off LED**

Turn off GPIO1\_21 by writing 0x00200000 to 0x4804C190 = 0x4804C000 (GPIO1 Base) + 0x190 (offset of GPIO1\_CLEAR\_DATA\_OUT)

* **Restore saved registers and return to wait loop**

1. **TLKR\_SVC procedure**
2. **Save registers**
3. **Check CTS signal**

Read MSR register at 0x481AA018, check bit 4

If bit 4 = 0

{

Check THR status by reading LSR register at 0x481AA014, check bit 5

If bit 5 = 0

{

Restore saved registers

Return to mainline

}

Else

{

Disable Tx interrupt by writing 0x08 to 0x481AA004 (UART Base) + 0x04 (offset of IER\_UART register) to prevent spinning problem

}

Else,

{

Check THR status

Read LSR register at 0x481AA014, check bit 5

If bit 5 = 0

{

Restore saved registers

Return to mainline

}

Else, go to SEND to send a character

1. **SEND algorithm**

Enable THR and MSR interrupts by writing 0x0A to 0x481AA004 = 0x481AA000 (UART Base) + 0x04 (offset of IER\_UART register)

Get message store at CHAR\_PTR address

Get one character in the message

Compare to check character 0xD

If Character == 0xD

Go to Check\_TIMER\_interrupt

Else ( not character 0xD)

Reload\_timer by Storing 0xFFFF8300 at Timer TCRR register ( make sure 1s delay when it see character 0xD)

Store the character in the THR register (0x481AA000)

Get the value store at CHAR\_COUNT address

Decrement the value by 1 and store it back to CHAR\_COUNT

Check if it is the last character by checking the current value after decrement

(If the value is less than to zero counter)

{

Restore the original message to CHAR\_PTR

Restore the original message’s length to CHAR\_COUNT

Disable THR interrupt by clearing bit 2 of IER\_UART (0x481AA004)

Turn ON LED by writing 0x00200000 to 0x4804C194 = 0x4804C000 (GPIO1 Base) + 0x194 (offset of GPIO1\_SET\_DATA\_OUT)

Restore saved registers

Return to mainline

}

Else, ( value > = 0 🡪 go back to mainline and continue sending next character)

Restore saved registers

Return to mainline.

1. **Check\_TIMER\_interrupt procedure:**

Read INTC\_PENDING\_IRQ2 REGISTER at 0x482000D8, test with 0x4

If Bit 2 == 0 ( Interrupt does not come from Timer)

{

Restore saved registers

Return to mainline

}

Else ( Bit 2 ==1)

{

Read address of Timer 2 IRQ Status (0x48040028), test with 0x2

If bit 1 ==0 ( Not overflow)

{

Restore saved registers

Return to mainline

}

Else ( bit 1==1 🡪 Overflow)

Go to IRQ\_Timer

}

1. **IRQ\_Timer:**

**Turn off IRQ request from overflow**

Writing 0x2 to 0x48040028 (0x48040028 (Timer0\_base) + 0x28 (offset of Timer\_IRQ STATUS))

**Generate new IRQ generation**

Writing 0x1 to 0x48200048 (0x48200000 (INTC\_base) + 0x48 (offset of INTC\_CONTROL))

**Sending 0xD UART THR**

Storing 0xD at address 0x481AA000 ( UART Transmit buffer)

**Update pointer for next character ( after 0xD)**

Load value from CHAR\_PTR

Add value with 0x1

Store back to CHAR\_PTR

**Restore saved registers**

**Return to mainline**

**Data**

**MESSAGE:**

**.ascii** "Ten"

**.byte** 0xD

**.ascii** "Nine"

**.byte** 0xD

**.ascii** "Eight"

**.byte** 0xD

**.ascii** "Seven"

**.byte** 0xD

**.ascii** "Six"

**.byte** 0xD

**.ascii** "Five"

**.byte** 0xD

**.ascii** "Four"

**.byte** 0xD

**.ascii** "Three"

**.byte** 0xD

**.ascii** "Two"

**.byte** 0xD

**.ascii** "One"

**.byte** 0xD

**.ascii** "Zero "

**.byte** 0xD

**CHAR\_PTR:** **.word** MESSAGE

**CHAR\_COUNT:** **.word** 56

.EQU MESSAGE\_LEN,56

**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