|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | ***Portland State University*** [Maseeh College of Engineering and Computer Science](http://pdx.edu/cecs/) 1559170_300.jpg | | ECE 372 - PROJECT REPORT | | Project 3: I2C and PWM | | **2/21/2015** | |
|  |
| **Instructor: Douglas V.Hall**  **TA: Leela Yadlapalli** |
| Students: Hai Dang Hoang  PSUID: 946749560  Email: [danghai@pdx.edu](mailto:danghai@pdx.edu) |

**Introduction:**

I2C is a 2-wire serial interface that is used with many currently available devices and modules such A/D converters, displays, EEPROMS, etc. For this reason, most current processor have a built-in I2C controller, so you do not have to generate the I2C SCL and SDA signals by bit-banging GPIO pins.. The TI Sitara AM 3359 processor you will be using has 3 built-in I2C controller I2C(0-2). The overall goals of this project is display name on a New Haven 2x20 LCD display by using use I2C1 to communicate with the display.

**PART I**

1. **Analyze I2C:**
2. **Transmission I2C**

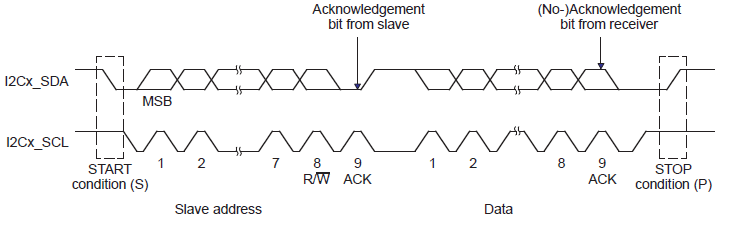


Figure shows the bit format for data transmission of a byte over an I2C bus. To start the transmission, the master pulls the SDA line from high to low while the SCL line is high. The master then pulses the SCL line and shifts out the data bits on SDA synchronously with the SCL pulses. Note that the Most Significant Bit, bit 7, of the data byte is shifted out first on the SDA line. If the slave receives the 8 data bits correctly, it synchronously pulls the SDA line low as an acknowledge signal to the master as shown at the end of the first byte in Figure.

When the slave releases the SCL line and it is pulled high by the external pull-up resistor, the master can send another byte. If a slave does not generate an acknowledge signal after a byte, the master can either generate a the bus to start a new transmission. For a stop condition the master allows the SDA line to transition high while the SCL line is high and for a repeated start condition, the SDA line is pulled low while the SCL line is high.

1. **Slave Address**

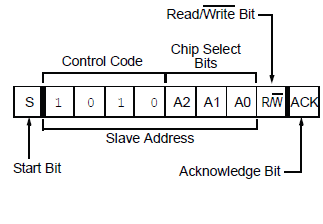


Figure show the format of a slave address. The slave address of device I2C\_LCD is : 0x78 ( 0111 1000). MCU will copy 7 bit ( start Most significant bit) from slave address device and fill in slave address register of I2C . Therefore, the value must be stored in I2C Slave Address register is 0x3C.

After initialize I2C, I get result:

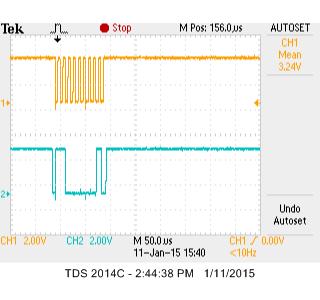


Figure show sending slave address, but slave address is not right. ACK is set, which prevent sending other data.

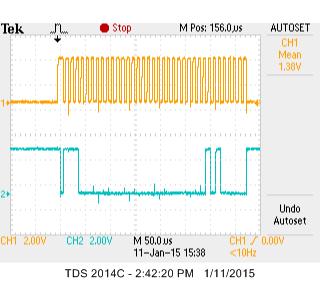
****

Figure show sending slave address. ACK is not set, which allow sending other data

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

* ENABLE CLOCK FOR I2C1

Storing 0x2 at address 0x44E00048 (CM\_PER\_I2C1\_CLKCTRL)

* CONTROL MODULE

**1**. Change pin 17 on P9 (SCL): Slow slew rate/ Pull Up/ Receiver enable/ Mode 2

By writing 0x72 to 0x44E10958 (0x44E10000 (Control Module Base) + 0x958 (offset of pin 958))

**2**. Change pin 17 on P9 (SDA): Slow slew rate/ Pull Up/ Receiver enable/ Mode 2

By writing 0x72 to 0x44E1095C (0x44E10000 (Control Module Base) + 0x95C (offset of pin 95C))

* INITIAL I2C1

1. Disable I2C\_EN by storing 0x0 at register 0x4802A0A4 ( I2C\_CON)
2. Disable auto Idle by storing 0x0 at register 0x4802A010 (I2C\_SYSC)
3. Select PSC, SCLL, and SCLH

* ICLK = 48Mhz /I2C\_PSC. Choose I2C\_PSC =4 because ICLK = 48/4 = 12Mhz
* Frequency = ICLK/ [(SCLL+7) + (SCLH+5)]. According to Technical Reference Manual, I2C\_SCLL = I2C\_SCLH = x. 🡪 Solving x = 0x36
* Therefore, setting PSC 3 for dividing 4 by storing 0x3 at address 0x4802A0B0 (I2C\_PSC)
* Setting SCLL and SCLH : 0x36 by storing 0x36 at register 0x4802A0B4 ( I2C\_SCLL), and register 0x4802A0B8 (I2C\_SCLH)

1. Enable I2C\_EN by storing 0x8000 at register 0x4802A0A4 (I2C\_CON)

* TRANSMIT DATA BYTE TO INITIALIZE I2C\_LCD

1. Clear all bit in I2C\_IRQSTATUS\_RAW by writing 0xFFFF at register 0x4802A028 (I2C\_IRQSTATUS) 🡪 Write 1 to clear bit.
2. Sending Slave address 0x3C by storing 0x3C at register 0x4802A0AC (I2C\_SA).
3. Determine 10 data counter for transmit by storing 10 at register 0x4802A098 (I2C\_CNT).
4. Configure transmit : I2C enable/ Master/ Transmit/ STT/ STP by storing 0x8603 at register 0x4802A0A4 (I2C\_CON).
5. Sending the data byte (10 bytes)

* MOV R1,#0x0 🡪 Control Byte

Call Procedure Transmit\_Data

* MOV R1,#0x38 🡪 Function set

Call Procedure Transmit\_Data

* MOV R1,#0x39 🡪 Function set

Call Procedure Transmit\_Data

* Call Procedure delay 10ms
* MOV R1,#0x14 🡪 Bias set

Call Procedure Transmit\_Data

* MOV R1,#0x78 🡪 Contrast Set

Call Procedure Transmit\_Data

* MOV R1,#0x5E 🡪 Power/ICON control/ Contrast set

Call Procedure Transmit\_Data

* MOV R1,#0x6D 🡪 Follower control

Call Procedure Transmit\_Data

* MOV R1,#0x0C 🡪 Display on

Call Procedure Transmit\_Data

* MOV R1,#0x01 🡪 Clear Display

Call Procedure Transmit\_Data

* MOV R1,#0x06 🡪 Entry mode set

Call Procedure Transmit\_Data ( Last data byte)

Call Procedure Wait\_access

Call Procedure Turn\_off\_ARDY

Call Procedure Delay 10ms

* TRANSMIT DATA BYTE TO DISPLAY MY NAME

1. Clear all bit in I2C\_IRQSTATUS\_RAW by writing 0xFFFF at register 0x4802A028 (I2C\_IRQSTATUS) 🡪 Write 1 to clear bit.
2. Sending Slave address 0x3C by storing 0x3C at register 0x4802A0AC (I2C\_SA).
3. Determine 11 data counter for transmit by storing 10 at register 0x4802A098 (I2C\_CNT).
4. Configure transmit : I2C enable/ Master/ Transmit/ STT/ STP by storing 0x8603 at register 0x4802A0A4 (I2C\_CON).
5. Sending the data byte (11 bytes)

* MOV R1,#0x80 🡪 Control Byte

Call Procedure Transmit\_Data.

* MOV R1,#0x86 🡪 Position of cursor

Call Procedure Transmit\_Data

* MOV R1,#0x40 🡪 Datasend

Call Procedure Transmit\_Data

* Load address memory storing my name String

While (Counter !=0)

{

Decrement counter by 1;

Get ascii character and update pointer from memory

Call Procedure Transmit\_Data

}

1. **TRANSMIT\_DATA PROCEDURE**

* Push uses registers on Stack
* Load I2C Data Register (I2C\_DATA) 0x4802A09C
* Storing data that want to transmit at register I2C\_Data
* Call Procedure Wait\_transmit
* Call Procedure Turn\_off\_XRDY
* Restore values for saved registers
* Return to Mainline

1. **WAIT\_TRANSMIT PROCEDURE**

* Push uses registers on Stack
* Create Label Wait\_XRDY
* Load I2C\_IRQSTATUS\_RAW register 0x4802A024
* Get value from I2C\_IRQSTATUS\_RAW
* Test bit 4
* If Bit = 0 go back to Label Wait\_XRDY check again until it equals 1
* Else (Bit =1) restore values for saved registers from Stack
* Return to Mainline

1. **TURN\_OFF\_XRDY PROCEDURE**

* Push uses registers on Stack
* Load I2C\_IRQSTATUS register 0x4802A028
* Storing value 0x10 at register I2C\_IRQSTATUS (0x4802A028)
* restore values for saved registers
* Return to Mainline

1. **WAIT\_ACCESS PROCEDURE**

* Push uses registers on Stack
* Create Label Wait\_ARDY
* Load I2C\_IRQSTATUS\_RAW register 0x4802A024
* Get value from I2C\_IRQSTATUS\_RAW
* Test bit 2
* If Bit = 0 go back to Label Wait\_ARDY check again until it equals 1
* Else (Bit =1) restore values for saved registers from Stack
* Return to Mainline

1. **TURN\_OFF\_ARDY PROCEDURE**

* Push uses registers on Stack
* Load I2C\_IRQSTATUS register 0x4802A028
* Storing value 0x4 at register I2C\_IRQSTATUS (0x4802A028)
* restore values for saved registers
* Return to Mainline

**Data**

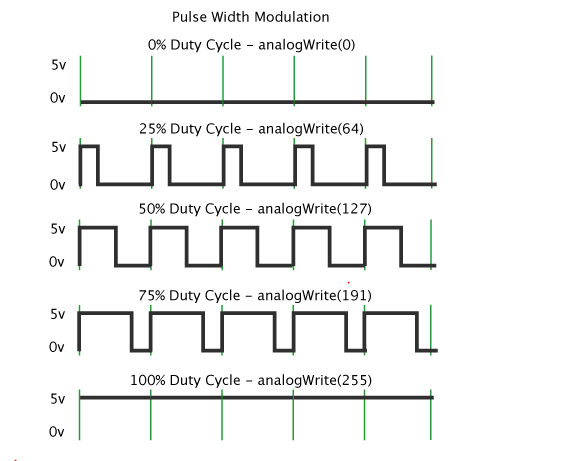
MyName:

.ascii “Hai Dang”

**PART II**

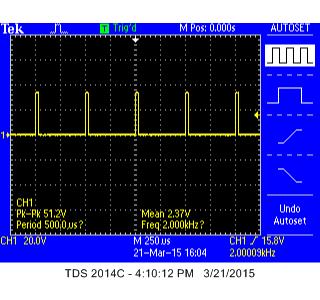
1. **Analyze PWM ( Pulse Width Modulation):**

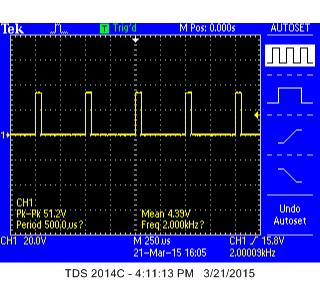
The program implements the use of analog output ( PWM) to control brightness back light of I2C LCD.

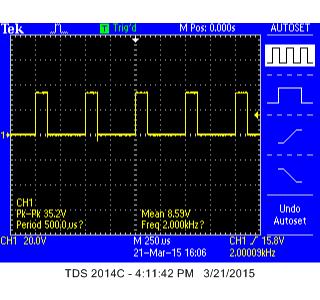


According to Timothy Hirzel, Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5V Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, I change, or modulate, that pulse width. If I repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED. In the graphic above, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency.

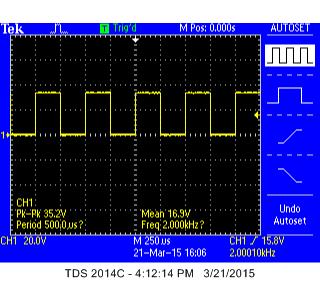
After initializing PWM and setting some value duty cycle, I get the result:

 6.25% duty cycle

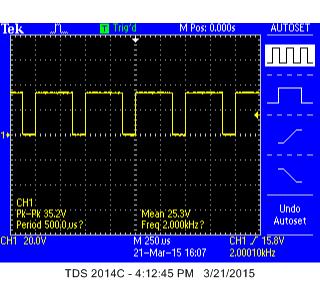
 12.5% duty cycle



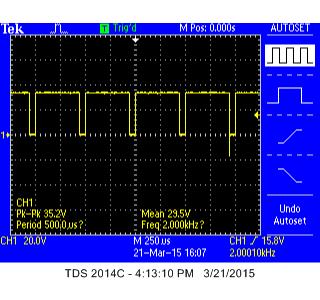
25% duty cycle



50% duty cycle



75% duty cycle



80% duty cycle

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

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 PWM
* Enable Clock PWM by storing 0x2 at register 0x44E000CC (CM\_PER\_EPWMSS1\_CLKCTRL)
* Select Time-base clock enable for PWMSS1 by storing 0x2 at register 0x44E10664 (PWMSS\_CTRL)
* Select Pin Mux (Control Module ) for PWM by storing 0x6 at register 0x44E10848 (CONF\_GPMC\_A1) 🡪 Choosing Mode 6
* Disable PWM stopping on debug events, Select Time-Base Clock is divide 1, Select Up-count mode, and select Free run by storing 0xC000 at register 0x48302200 (TBCTL)
* Set period by storing value 0xC350 for Time-base period at register 0x4830220A (TBPRD)
* Set compare A value of the period to get duty cycle (2.5% duty cycle) by storing 0x186 at register 0x48302212 (CMPA)
* Configure output to select output A (EPWMxA) with forcing EPWMxA output. In addition, when counter equals active CMPA, counter increase by storing 0x1A at register 0x48302216 (AQCTLA)
* Using R4 Register hold memory store value of duty cycle
* Using R5 Register is pointer control memory
* I2C1
* The same as **Part 1**
* Sending data to initialize I2C\_LCD and display the name is the same as **Part 1**
* 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

B LOOP

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 =0 ( Not GPIO interrupt)

Go to PASS\_ON

Else

{

Check interrupt coming from GPIO31 by reading GPIO1\_IRQ\_STATUS REGISTER SET 0 at 0x4804C02C, check bit 31

If Yes 🡪 Go to BUTTON\_SVC

Else No 🡪 Go to PASS\_ON

}

PASS\_ON: Restore registers from STACK

Pass execution onto wait LOOP

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))
* Check R5 register (Pointer control memory of value duty cycle)

If (R5 == 40)

* + Reset R5=0

Else

* + R5= R5 +4; (Increment pointer by 4)
* **Set period again after pressing Button**
* Get value duty cycle from Memory
* Storing value duty cycle at register 0x48302212 (CMPA)
* **Restore saved registers and return to wait loop**

**Data**

@ Value set for duty cycle for PWM ( Pulse Width Modulation)

**value:** **.word** 0x186, 0x30D, 0x61A, 0xC35, 0x186A, 0x30D4, 0x61A8, 0x927C, 0xAAE6, 0xB71B

**.data**

**MyName:**

**.ascii** "Hai Dang"

**PART III**

Program implement Clock with real time. At the beginning program, it show my name. After pressing button, the real time clock appears. Furthermore, user can use button to reset value of real time clock and control brightness of backlight.

00:00:00

Unit\_Hour

Unit\_Second

Tenth\_Hour

Tenth\_Minute

Unit\_Minute

Tenth\_Second

**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 INTC
  + - 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 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)).
* INITIALIZE PWM
* Enable Clock PWM by storing 0x2 at register 0x44E000CC (CM\_PER\_EPWMSS1\_CLKCTRL)
* Select Time-base clock enable for PWMSS1 by storing 0x2 at register 0x44E10664 (PWMSS\_CTRL)
* Select Pin Mux (Control Module ) for PWM by storing 0x6 at register 0x44E10848 (CONF\_GPMC\_A1) 🡪 Choosing Mode 6
* Disable PWM stopping on debug events, Select Time-Base Clock is divide 1, Select Up-count mode, and select Free run by storing 0xC000 at register 0x48302200 (TBCTL)
* Set period by storing value 0xC350 for Time-base period at register 0x4830220A (TBPRD)
* Set compare A value of the period to get duty cycle (2.5% duty cycle) by storing 0x186 at register 0x48302212 (CMPA)
* Configure output to select output A (EPWMxA) with forcing EPWMxA output. In addition, when counter equals active CMPA, counter increase by storing 0x1A at register 0x48302216 (AQCTLA)
* Using R4 Register hold memory store value of duty cycle
* Using R5 Register is pointer control memory
* I2C1
* The same as **Part 1**
* Sending data to initialize I2C\_LCD and display the name is the same as **Part 1**
* 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

B LOOP

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 == 0)

Go to Check timer interrupt

Else (bit 2==1)

{

Check interrupt coming from GPIO31 by reading GPIO1\_IRQ\_STATUS REGISTER SET 0 at 0x4804C02C, check bit 31

If Yes 🡪 Go to BUTTON\_SVC

Else No 🡪 Go to PASS\_ON

}

Check\_TIMER\_interrupt:

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

If Bit 4== 0 ( Interrupt does not come from Timer)

{

Restore saved registers

Return to mainline

}

Else ( Bit 4 ==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: (Timer Overflow)**

**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))

**Call Procedure UPDATE\_CLOCK**

**Display clock on I2C\_LCD**

1. Clear all bit in I2C\_IRQSTATUS\_RAW by writing 0xFFFF at register 0x4802A028 (I2C\_IRQSTATUS) 🡪 Write 1 to clear bit.
2. Sending Slave address 0x3C by storing 0x3C at register 0x4802A0AC (I2C\_SA).
3. Determine 12 data counter for transmit by storing 10 at register 0x4802A098 (I2C\_CNT).
4. Configure transmit : I2C enable/ Master/ Transmit/ STT/ STP by storing 0x8603 at register 0x4802A0A4 (I2C\_CON).

**5)** Sending the data byte (12 bytes)

* MOV R1,#0x80 🡪 Control Byte

Call Procedure Transmit\_Data

* MOV R1,#0x89 🡪 Position cursor

Call Procedure Transmit\_Data

* MOV R1,#0x40 🡪 Data send

Call Procedure Transmit\_Data

* MOV R1,#0x20 🡪 Space character ‘ ’

Call Procedure Transmit\_Data

* Get value from Tenth\_Hour memory

Call Procedure Transmit\_Data

* Get value from Unit\_Hour memory

Call Procedure Transmit\_Data

* MOV R1,#0x3A 🡪 Character: ‘**:**’

Call Procedure Transmit\_Data

* Get value from Tenth\_Minute memory

Call Procedure Transmit\_Data

* Get value from Unit\_Minute memory

Call Procedure Transmit\_Data

* MOV R1,#0x3A 🡪 Character: ‘**:**’

Call Procedure Transmit\_Data

* Get value from Tenth\_Second memory

Call Procedure Transmit\_Data

* Get value from Unit\_Second memory

Call Procedure Transmit\_Data

Call Procedure Wait\_access

Call Procedure Turn\_off\_ARDY

Call Procedure Delay 10ms

**Restore saved registers**

**Return to mainline**

1. **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))
* Check R5 register (Pointer control memory of value duty cycle)

If (R5 == 40)

* + Reset R5=0

Else

* + R5= R5 +4; (Increment pointer by 4)
* **Set period again after pressing Button**
* Get value duty cycle from Memory
* Storing value duty cycle at register 0x48302212 (CMPA)
* **Call Procedure BACK\_UP\_CLOCK**
* **Display BACK\_UP\_CLOCK on I2C\_LCD**

The same way to display on I2C\_LCD in IRQ\_Timer part

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

1. **UPDATE CLOCK PROCEDURE:**

**High level algorithm for Update clock procedure ( Translate from high level algorithm to Assembly code).**

Increment Unit\_Second ( When timer overflow happen, it create 1 second delay)

If Unit\_Second != ‘9’ (0x39)

{

Unit\_Second++;

Go to Finish

}

Else

{

Unit\_Second = ‘0’ (Reset to 0x30)

Tenth\_Second++;

If ( Tenth\_Second !=5)

{

Tenth\_Second++;

Go to finish

}

Else

{

Tenth\_Second = ‘0’ (Reset to 0x30)

Unit\_minute ++;

If ( Unit\_Minute != ‘9’)

{

Unit\_minute++;

Go to Finish;

}

Else

{

Unit\_Minute = ‘0’;

Tenth\_Minute++

If (Tenth\_Minute != ‘5’)

{

Ten\_Minute++;

Go to Finish;

}

Else

{

Unit\_Hour++

If Hour ==24

Call Procedure BACK\_UP\_CLOCK

}

}

}

}

1. **BACK\_UP\_CLOCK PROCEDURE**

Simply storing value 0x30 (‘0’) into memory Tenth\_Hour, Unit\_Hour, Tenth\_Minute, Unit\_Minute, Tenth\_Second, Unit\_Second.

**Data**

**.data**

@ Data for Duty cycle PWM

**value:** **.word** 0x186, 0x30D, 0x61A, 0xC35, 0x186A, 0x30D4, 0x61A8, 0x927C, 0xAAE6, 0xB71B

@ Data for value clock 00:00:00

@ Unit and tenth of Second

**unit\_second:** **.word** 0x30

**tenth\_second:** **.word** 0x30

@ Unit and tenth of Minute

**unit\_minute:** **.word** 0x30

**tenth\_minute:** **.word** 0x30

@ Unit and tenth of Hour

**unit\_hour:** **.word** 0x30

**tenth\_hour:** **.word** 0x30

**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