WRITING AN I2C DRIVER

**FOR**

**A NEW HAVEN LCD DISPLAY**

ECE 372 by Yusme Pradera

Introduction :

The I2C is a 2-wire serial interface that is used with many currently availaBL e 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**

What is I2C:

I2C is a bus type that comunicates with external I2C\_bus\_compatible devices . External components attached to the I2C bus can serially transmit/receive up to 8-bit data to/from the CPU device through the two-wire I2C interface.

For this project we will be looking at I2C1:

These are connectivity Attributes

I2C Clock and Reset management

The I2C controllers have separate bus interface and functional clocks. During power-down mode, the I2Cx\_SCL and I2Cx\_SDA are configured as inputs

I2C SDA: Serial Data Line

I2C SCL: Serial Clock Line

These two lines carry information between a device and other connected to the I2C bus.

Both SDA and SCL are bi-directional pins. They must be connected to a positive supply voltage via a pullup resistor. When the bus is free, both pins are high. The driver of these two pins has an open drain to perform the required wired-AND function.

There are Several ways to reset I2C

- A system Reset (PIRSTNA = 0). A device reset causes the system reset. All registers are reset to power up reset values.

- A software reset by setting the SRST bit in the I2C\_SYSC register

The I2C\_EN bit in the I2C\_CON register can be used to hold the I2C module in reset. When the system bus reset is removed (PIRSTNA = 1), I2C\_EN = 0 keeps the functional part of I2C module in reset state and all configuration registers can be accessed. I2C\_EN = 0 does not reset the registers to power up reset values.

Data Validity for I2C

Data on the SDA line must be stable during High period of the clock. The changes to the data line can only change when the clock signal on the SCL line is low.

Start and Stop of the I2C

The I2C module generates START and STOP conditions when it is configured as a master.

• START condition is a high-to-low transition on the SDA line while SCL is high.

• STOP condition is a low-to-high transition on the SDA line while SCL is high.

• The bus is considered to be busy after the START condition (BB = 1) and free after the STOP

condition (BB = 0).

I2C Operation

Each byte put on the SDA line is 8 bits long. The number of bytes that can be transmitted or received is restricted by the value programmed in the DCOUNT register. The data is transferred with the most significant bit (MSB) first. Each byte is followed by an acknowledge bit from the I2C module if it is in receiver mode

The I2C module supports 2 data formats

1- 7-bit/10-bit addressing format

2- 7-bit/10-bit addressing format with repeated start condition

The first byte after a start condition (S) always consists of 8 bits. In the acknowledge mode, an extra bit dedicated for acknowledgment is inserted after each byte. In the addressing formats with 7-bit addresses, the first byte is composed of 7 MSB slave address bits and 1 LSB R/nW bit. In the addressing formats with 10-bit addresses, the first byte is composed of 7 MSB slave address bits, such as 11110XX, where XX is the two MSB of the 10-bit addresses, and 1 LSB R/nW bit, which is 0 in this case.

Master transmitter data formats is shifted out on the serial data line SDA in synch with the self-generated clock pulses on the serial clock line SCL

The clock pulses are inhibited and SCL held low when the intervention of the processor is required (XUDF) after a byte has been transmitted.

Master receiver:

- entered after the slave address byte and bit R/W\_has been transmitted, if R/W\_ is high.

- Serial data bits received on bus line SDA are shifted in synch with the self-generated clock pulses on SCL.

- The clock pulses are inhibited and SCL held low when the intervention of the processor is required (ROVR) after a byte has been transmitted. At the end of a transfer, it generates the stop condition.

Slave Transmitter:

entered if the slave address byte is the same as its own address and bit R/W\_ has been transmitted, if R/W\_ is high.

- The slave transmitter shifts the serial data out on the data line SDA in synch with the clock pulses that are generated by the master device. It does not generate the clock but it can hold clock line SCL low while intervention of the CPU is required (XUDF)

Slave Receiver:

- serial data bits received on the bus line SDA are shifted-in in synch with the clock pulses on SCL that are generated by the master device.

- It does not generate the clock but it can hold clock line SCL

low while intervention of the CPU is required (ROVR) following the reception of a byte.

I2C Clock Generation only one master device generates the clock signal, SCL however, there are two or more master devices and the clock must be synchronized so that the data output can be compared.

The wired-AND property of the clock line means that a device that first generates a low period of the clock line overrules the other devices. At this high/low transition, the clock generators of the other devices are forced to start generation of their own low period. The clock line is then held low by the device with the longest low period, while the other devices that finish their low periods must wait for the clock line to be released before starting their high periods. A synchronized signal on the clock line is thus obtained, where the slowest device determines the length of the low period and the fastest the length of the high period.

Prescaler (SCLK/ICLK)

The I2C module is operated with a functional clock (SCLK) frequency that can be in a range of 12-100 MHz, according to I2C mode that must be used (an internal ~24 MHz clock (ICLK) is recommended in case of F/S operation mode e)

I2C Interrupts

The I2C module generates 12 types of interrupt:

These 12 interrupts are accompanied with 12 interrupt masks and flags defined in the I2C\_IRQENABLE\_SET and respectively I2C\_IRQSTATUS\_RAW registers. Note that all these 12 interrupt events are sharing the same hardware interrupt line.

1- addressed as slave,

2- bus free (stop condition detected),

3- access error,

4- start condition,

5- arbitration-lost,

6- noacknowledge,

7- general call,

8- registers-ready-for-access,

9- receive and transmit data,

10- receive and transmit draining

-

Interrupts of interest:

registers-ready-for-access interrupt:

Registers-ready-for-access interrupt (ARDY) is generated by the I2C when the previously programmed address, data, and command have been performed and the status bits have been updated. This interrupt is used to let the CPU know that the I2C registers are ready for access

receive and transmit data:

Receive interrupt/status (RRDY) is generated when there is received data ready to be read by the CPU from the I2C\_DATA register (see the FIFO Management subsection for a complete description of required conditions for interrupt generation). The CPU can alternatively poll this bit to read the received data from the I2C\_DATA register.

Transmit interrupt/status (XRDY):

is generated when the CPU needs to put more data in the I2C\_DATA register after the transmitted data has been shifted out on the SDA pin (see the FIFO Management subsection for a complete description of required conditions for interrupt generation). The CPU can alternatively poll this bit to write the next transmitted data into the I2C\_DATA register.

Receive draining interrupt (RDR) is generated when the transfer length is not a multiple of threshold value, to inform the CPU that it can read the amount of data left to be transferred and to enable the draining mechanism

Transmit draining interrupt (XDR) is generated when the transfer length is not a multiple of threshold value, to inform the CPU that it can read the amount of data left to be written and to enable the draining mechanism

When the interrupt signal is activated, the Local Host must read the I2C\_IRQSTATUS\_RAW register to define the type of the interrupt, process the request, and then write into these registers the correct value to clear the interrupt flag.

Interrupt and DMA events:

I2C has two DMA channels (Tx and Rx). I2C has one interrupt line for all the interrupt requests.

FIFO Management :

The I2C module implements two internal 32-bytes FIFOs with dual clock for RX and TX modes. The depth of the FIFOs can be configured at integration via a generic parameter which will also be reflected in I2C\_IRQSTATUS\_RAW.FIFODEPTH register.

FIFO INTERRUPT

FIFO interrupt mode (relevant interrupts enabled via I2C\_IRQENABLE\_SET register), the processor is informed of the status of the receiver and transmitter by an interrupt signal. These interrupts are raised when receive/transmit FIFO threshold (defined by I2C\_BUF.TXTRSH or I2C\_BUF.RXTRSH) are reached; the interrupt signals instruct the Local Host to transfer data to the destination (from the I2C module in receive mode and/or from any source to the I2C FIFO in transmit mode).

FIFO polled mode

(I2C\_IRQENABLE\_SET.XRDY\_IE and I2C\_IRQENABLE\_SET.RRDY\_IE disabled

and DMA disabled), the status of the module (receiver or transmitter) can be checked

- by polling the XRDY and RRDY status registers (I2C\_IRQSTATUS\_RAW) (RDR and XDR can also be polled if draining feature

must be used).

The XRDY and RRDY flags are accurately reflecting the interrupt conditions mentioned in

Interrupt Mode. This mode is an alternative to the FIFO interrupt mode of operation, where the status of

the receiver and transmitter is automatically known by means of interrupts sent to the CPU

When Transmit Data

- Poll the transmit data ready interrupt flag bit (XRDY) in the I2C status register (I2C\_IRQSTATUS\_RAW),

- Use XRDY interrupt (I2C\_IRQENABLE\_SET.XRDY\_IE set)

- use the DMA TX (I2C\_BUF.XDMA\_EN set together with I2C\_DMATXENABLE\_SET) to write data into the data transmit register (I2C\_DATA).

- Use draining feature (I2C\_IRQSTATUS\_RAW.XDR enabled by I2C\_IRQENABLE\_SET.XDR\_IE)) if the transfer

length is not equal with FIFO threshold.

I2C1 address: 0x4802A000

INTC address: 0x48200000

I2C Registers : Page 4598 Sitara Manual

Offset Acronym Register Name

10h I2C\_SYSC System Configuration Register

2Ch I2C\_IRQENABLE\_SET I2C Interrupt Enable Set Register

30h I2C\_IRQENABLE\_CLR I2C Interrupt Enable Clear Register

24h I2C\_IRQSTATUS\_RAW I2C Status Raw Register

28h I2C\_IRQSTATUS I2C Status Register

98h I2C\_CNT Data Counter Register

B0h I2C\_PSC Register I2C Clock Prescaler Register

ACh I2C\_SA I2C Slave Address Register

B4h I2C\_SCLL I2C SCL Low Time Register

B8h I2C\_SCLH I2C SCL High Time Register

A4h I2C\_CON I2C Configuration Register

9Ch I2C\_DATA Data Access Register

94h I2C\_BUF Buffer Configuration Register

Under I2C\_IRQSTATUS\_RAW Register we have XRDY

- Transmit data ready IRQ status. Set to '1' by core when transmitter and when new data is requested. When set to '1' by core, an interrupt is signaled to MPUSS. Write '1' to clear



Figure 1. Signal waveform for sending and receiving data on I2C bus.

Every transmission on the bus will be handle by the master that pulls the SDA line from high to low while the SCL line is high. Devices with I2C bus capabilities can be internally programmed with a 7-bit address. The upper 7 bits of the first byte sent out by the master will contain the address of the slave that is to be written to or read from. The Most Significant bit, bit 7 of the data is shifted out first on the SDA line. To stop the transfer, 8 bits are sent to the slave and the SDA line low as an acknowledge signal to the master as shown at the end of the first byte. After the slave releases the SCL line and the external pull-up resistor pulls it high, the master can send another byte. 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.

. The slave address of device I2C\_LCD is 0x78 (0111 1000). The MCU will copy 7 bit, it will start with the most significant bit from the slave address device. Therefore, the value must be stored in the I2C Slave. Address register is 0x3C.

**PARTI - Communication with display using polled handshaking**

**Standard Program Structure and Algorithm:**

* Mainline
  + Enable the clock for **I2C1**
    - Store value 0x2 at CM\_PER\_I2C1\_CLKCTRL (0x44E00048)
  + Initialize control module
  + 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

* + 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

* **INITIAL I2C1**
  + Disable I2C\_EN by storing 0x0 at register 0x4802A0A4 ( I2C\_CON)
  + Disable auto Idle by storing 0x0 at register 0x4802A010 (I2C\_SYSC)
  + Select PSC, SCLL, and SCLH
    - Store 0x3 at address 0x4802A0B0 (I2C\_PSC)
    - Setting SCLL and SCLH : 0x36 by storing 0x36 at register 0x4802A0B4 ( I2C\_SCLL), and register 0x4802A0B8 (I2C\_SCLH)
    - Enable I2C\_EN by storing 0x8000 at register 0x4802A0A4 (I2C\_CON)
    - Load INT\_MIR\_CLEAR2 register address and unmask I2C1INT and INTC int 71
  + Clear all bit in I2C\_IRQSTATUS\_RAW by writing 0xFFFF at register 0x4802A028 (I2C\_IRQSTATUS) @ Write 1 to clear bit.
  + Sending Slave address 0x3C by storing 0x3C at register 0x4802A0AC (I2C\_SA).
  + Determine data counter for transmit by storing at register 0x4802A098 (I2C\_CNT).
    - * Load I2C1 base address + I2C\_CON offset 0xA4 and write 0x8603

@ Control Byte

* + - * MOV R1,#0x0

BL Transmit\_Data

@ Function set

* + - * MOV R1,#0x38

BL Transmit\_Data

@ Function set

* + - * MOV R1,#0x39

BL Transmit\_Data

* + - * + BL delay

@ Bias set

* + - * MOV R1,#0x14

BL Transmit\_Data

@ Contrast Set

* + - * MOV R1,#0x78

BL Transmit\_Data

@ Contrast set

* + - * MOV R1,#0x5E

BL Transmit\_Data

@ Follower control

* + - * MOV R1,#0x6D

BL Transmit\_Data

@ Display on

* + - * MOV R1,#0x0C

BL Transmit\_Data

@ Clear Display

* + - * MOV R1,#0x01

BL Transmit\_Data

@ Entry mode set

MOV R1,#0x06

BL Transmit\_Data ( Last data byte)

BL ACCESS\_HAND

BL ARDY\_OFF

BL TDelay

* **Display data on LCD**
  + Clear all bit in I2C\_IRQSTATUS\_RAW by writing 0xFFFF at register 0x4802A028 (I2C\_IRQSTATUS) @ Write 1 to clear bit.
  + Sending Slave address 0x3C by storing 0x3C at register 0x4802A0AC (I2C\_SA).
  + Determine data counter for transmit by storing at register 0x4802A098 (I2C\_CNT).
  + Configure transmit I2C START/STOP by storing 0x8603 at register 0x4802A0A4 (I2C\_CON).
  + Sending the data

@ Control Byte

* + - MOV R1,#0x80

BL Transmit\_Data.

@ Position of cursor

* + - MOV R1,#0x84

BL Transmit\_Data

@ Datasend

* + - MOV R1,#0x40

BL Transmit\_Data

* + - Load address memory storing DATA\_NAME

**DISPLAY**

* + Get char and update pointer ( LDRB R1,[R0],#0x1 )
  + Transmit data to LCD ( BL Transmit\_Data )
  + Decrement counter by One ( SUBS R2,R2,#0x1 )
  + Branch if not equal back to display (when decrement gets to zero finish )

IDLE:

NOP

B IDLE

**TRANSMIT\_DATA**

* Save stack registers
* Load I2C Data Register (I2C\_DATA) 0x4802A09C
* Storing data that want to transmit at register I2C\_DATA
* Branch to WAIT1
* BL XRDY\_OFF
* Restore saved registers and return to mainline
* Return to Mainline

**WAIT1- waiting to transmit**

* Push registers on Stack

**XRDY:**

* Load I2C\_IRQSTATUS\_RAW register 0x4802A024
* Get value from I2C\_IRQSTATUS\_RAW
* Test bit 4
* If Bit = 0 then branch back to XRDY then check again until bit = 1
* Otherwise bit =1 restore saved registers from Stack and Return to mainline

**ACCESS\_HAND:**

* Store registers on STACK

**ARDY:**

* Load I2C\_IRQSTATUS\_RAW register 0x4802A024
* Get value from I2C\_IRQSTATUS\_RAW
* Test bit 2
* If bit=0 branch back to ARDY check again until it bit= 1
* Else restore saved registers from Stack and return to mainline

**XRDY\_OFF**

* Store registers on Stack
* Load I2C\_IRQSTATUS register 0x4802A028
* Storing value 0x10 at register I2C\_IRQSTATUS (0x4802A028)
* restore saved registers and return to mainline

**ARDY\_OFF**

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

**TDELAY**

* restore values for saved registers and return to mainline

**WAIT**

* + Subtract 1
  + Branch if not equal back to WAIT
  + Restore saved registers and return to mainline

.data

DATA\_NAME:

.ascii “Josh Pradera”

.END

**High-Level algorithm**

@ ECE 372 PROJECT II

@ PART1

@ Project: Using I2C to send data from BBB to LCD

@ This project displays my name on I2C\_LCD ( New Haven 2x20 LCD display)

@ Josh Pradera

**.text**

**.global** \_start

**\_start:**

@ ENABLE CLOCK FOR I2C1

LDR R0,=0x44E00048 @ Enable Clock for I2C1 (CM\_PER\_I2C1\_CLKCTRL)

**MOV** R1,#0x2 @ Store 2 to enable

STR R1,[R0]

@ CONTROL MODULE I2C1

LDR R8,=0x44e10958 @ use Pin 17 on P9 for SCL

**MOV** R9,#0x72 @ Mode 2

STR R9,[R8] @ Store

LDR R8,=0x44e1095C @ use Pin 18 on P9 for SDA

**MOV** R9,#0x72 @ Mode 2

STR R9,[R8] @ Store

@ INITIALIZE I2C1

LDR R0,=0x4802A0A4 @ LOAD I2C\_CON Address

LDR R1,=0x0 @ Disable I2C\_EN

STR R1,[R0]

LDR R0,=0x4802A010 @ LOAD System Configuration Register

**MOV** R1,#0x0 @ Disable auto Idle

STR R1,[R0]

LDR R0,=0x4802A0B0 @ Load I2C Clock Prescaler Register

**MOV** R1,#0x3 @ Divide by 4

STR R1,[R0]

@ Setting SCLL and SCLH : 0x36

LDR R0,=0x4802A0B4 @ Load I2C\_SCLL Register

**MOV** R1,#0x36

STR R1,[R0]

LDR R0,=0x4802A0B8 @ Load I2C\_SCLH register

**MOV** R1,#0x36

STR R1,[R0]

LDR R0,=0x4802A0A8

**MOV** R1,#0x1

STR R1,[R0]

LDR R0,=0x4802A094 @ Load I2C\_BUF register

**MOV** R1,#0x0 @ Clearing the register will make RXTRSH/TXTRSH = 1

STR R1,[R0]

LDR R0,=0x4802A0A4 @ I2C Configuration Register

LDR R1,=0x8000 @ Enable I2C\_EN

STR R1,[R0]

@ Initializating LCD

LDR R0,=0x4802A028 @ Load IRQSTATUS Register

LDR R1,=0xFFFF @ Write 1 to clear all

STR R1,[R0]

@ Send Slave address 0x3C

**MOV** R1,#0x3C @ Slave address

LDR R0,=0x4802A0AC @ Load I2C Slave Address Register

STR R1,[R0]

**MOV** R1,#10 @ Load Data byte counter

LDR R0,=0x4802A098 @ Load Data Counter Register I2C CNT

STR R1,[R0]

LDR R0,=0x4802A0A4 @ LOAD I2C\_CON Address

LDR R1,=0x8603 @ I2C START/STOP

STR R1,[R0]

@ 0x0: Control Byte

**MOV** R1,#0x0

BL Transmit\_Data

@ 0x38: Func set

**MOV** R1,#0x38

BL Transmit\_Data

@ 0x39 : Func set

**MOV** R1,#0x39

BL Transmit\_Data

BL TDELAY

@ 0x14: Bias set

**MOV** R1,#0x14

BL Transmit\_Data

@ 0x78: Contrast set

**MOV** R1,#0x78

BL Transmit\_Data

@ 0x5E: Contrast Set

**MOV** R1,#0x5E

BL Transmit\_Data

@ 0x6D: Follower control

**MOV** R1,#0x6D

BL Transmit\_Data

@ 0x0C: Display on

**MOV** R1,#0x0C

BL Transmit\_Data

@ 0x01: Clear Display

**MOV** R1,#0x01

BL Transmit\_Data

@ 0x06: Entry mode set

**MOV** R1,#0x06

BL Transmit\_Data

@ Wait for access

BL ACCESS\_HAND

BL ARDY\_OFF

BL TDELAY

@ Display data on LCD

LDR R0,=0x4802A028 @ Load IRQSTATUS Register

LDR R1,=0xFFFF @ Write 0xFFFF to clear

STR R1,[R0]

@ Send Slave address 0x3C

**MOV** R1,#0x3C @ Slave address

LDR R0,=0x4802A0AC @ Load I2C Slave Address Register

STR R1,[R0]

**MOV** R2,#11 @ Data bytes (Counter)

LDR R0,=0x4802A098 @ Load Data Counter Register I2C CNT

STR R1,[R0]

LDR R0,=0x4802A0A4 @ LOAD I2C\_CON Address

LDR R1,=0x8603 @ I2C START/STOP

STR R1,[R0]

@ 0x80: Control byte

**MOV** R1,#0x80

BL Transmit\_Data

@ 0x84: Position start after the first 4 spaces of the LCD screen

**MOV** R1,#0x84

BL Transmit\_Data

@ 0x40: Send Data

**MOV** R1,#0x40

BL Transmit\_Data

LDR R0,=DATA\_NAME @ Load address memory DATA\_NAME

**DISPLAY:**

LDRB R1,[R0],#0x1 @ Get char and update the pointer

BL Transmit\_Data @ Transmit data to LCD

SUBS R2,R2,#0x1 @ Decrement counter by one

BNE DISPLAY @ If decrement equal zero then Finish

**IDLE**:

**NOP**

**B** **IDLE**

**Transmit\_Data:**

STMFD R13!,{R0,R14} @ Save stack registers

LDR R0,=0x4802A09C @ Load I2C1 Base address + I2C\_DATA register

STR R1,[R0] @ Store

BL WAIT1

BL XRDY\_OFF

LDMFD R13!,{R0,R14} @ Restore saved registers

**MOV** PC, R14 @ Return to mainline

@ When detecting an interrupt request (XRDY or RRDY type), the CPU can be programmed to write/read

@ the amount of data bytes specified by the corresponding FIFO threshold (I2C\_BUF.TXTRSH + 1 or

@ I2C\_BUF.RXTRSH + 1). In this case, the interrupt condition will be cleared and the next interrupt will

@ be asserted again when the XRDY or RRDY condition will be again met.

**WAIT1:**

STMFD R13!,{R0-R1,R14} @ Store stack registers

**XRDY:**

LDR R0,=0x4802A024 @ I2C1 base address + IRQ STATUS RAW

LDR R1,[R0] @ data from IRQ\_STATUS RAW

TST R1,#0x10 @ Test Bit 4

BEQ XRDY

LDMFD R13!,{R0-R1,R14} @ Restore saved registers

**MOV** PC, R14 @ Return to mainline

@ Registers-ready-for-access interrupt (ARDY) is generated by the I2C when the previously programmed

@ address, data, and command have been performed and the status bits have been updated. This

@interrupt is used to let the CPU know that the I2C registers are ready for access

**ACCESS\_HAND:**

STMFD R13!,{R0-R1,R14} @ Store registers on STACK

**ARDY:**

LDR R0,=0x4802A024 @ I2C1 base address + IRQ STATUS RAW OFFSET

LDR R1,[R0] @ data from IRQ\_STATUS RAW

TST R1,#0x04 @ Test Bit 2

BEQ ARDY

LDMFD R13!,{R0-R1,R14} @ Restore saved registers

**MOV** PC, R14 @ Return to mainline

@The CPU can alternatively poll this bit to write the next transmitted data into the I2C\_DATA register.

**XRDY\_OFF:**

STMFD R13!,{R0-R1,R14} @ Store registers on Stack

LDR R0,=0x4802A028 @ Load I2C\_IRQSTATUS resgister

**MOV** R1,#0x10 @ Clear XRDY

STR R1,[R0]

LDMFD R13!,{R0-R1,R14} @ Restore saved registers

**MOV** PC, R14 @ Return to mainline

@ Mode: I2C Master transmit, Others: STP = 1, ARDY Set Condition:

@ DCOUNT = 0.

@ This read/clear only bit, when set to 1, indicates that the previously

@ programmed data and command (receive or transmit, master or

@ slave) has been performed and status bit has been updated.

@ The CPU uses this flag to let it know that the I2C registers are ready

@ to be accessed again.

**ARDY\_OFF:**

STMFD R13!,{R0-R1,R14} @ Store registers on Stack

LDR R0,=0x4802A028 @ LoadI2C base adrress + I2C\_IRQSTATUS Offset

**MOV** R1,#0x4 @ Clear ARDY

STR R1,[R0]

LDMFD R13!,{R0-R1,R14} @ Restore saved registers

**MOV** PC, R14 @ Return to mainline

**TDELAY:**

STMFD R13!,{R0-R1,R14} @ Store registers on Stack

LDR R0,=0xFFFF @ TDELAY

**WAIT:**

SUBS R0,R0,#1

BNE WAIT

LDMFD R13!,{R0-R1,R14} @ Restore saved registers

**MOV** PC, R14 @ Return to mainline

**.data**

**DATA\_NAME:**

**.ascii** "Josh Pradera"

.END

**PART 2 - Communication with display using Interrupt-based handshaking**

This part of the projects Implements the interrupt-based handshaking.

* + **I2C1 address**: 0x4802A000
  + **INTC address:** 0x48200000
* **I2C Registers : Page 4598 Sitara Manual**

|  |  |  |
| --- | --- | --- |
| Offset | Acronym | Register Name |
| 10h | I2C\_SYSC | System Configuration Register |
| 2Ch | I2C\_IRQENABLE\_SET | I2C Interrupt Enable Set Register |
| 30h | I2C\_IRQENABLE\_CLR | I2C Interrupt Enable Clear Register |
| 24h | I2C\_IRQSTATUS\_RAW | I2C Status Raw Register |
| 28h | I2C\_IRQSTATUS | I2C Status Register |
| 98h | I2C\_CNT | Data Counter Register |
| B0h | I2C\_PSC Register | I2C Clock Prescaler Register |
| ACh | I2C\_SA | I2C Slave Address Register |
| B4h | I2C\_SCLL | I2C SCL Low Time Register |
| B8h | I2C\_SCLH | I2C SCL High Time Register |
| A4h | I2C\_CON | I2C Configuration Register |
| 9Ch | I2C\_DATA | Data Access Register |
| 94h | I2C\_BUF | Buffer Configuration Register |

**Standard Program Structure and Algorithm:**

* Mainline
  + Enable the clock for **I2C1**
    - Store value 0x2 at CM\_PER\_I2C1\_CLKCTRL (0x44E00048)
  + Initialize control module
  + 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

* + 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

* **INITIAL I2C1**
  + Disable I2C\_EN by storing 0x0 at register 0x4802A0A4 ( I2C\_CON)
  + Disable auto Idle by storing 0x0 at register 0x4802A010 (I2C\_SYSC)
  + Select PSC, SCLL, and SCLH
    - Store 0x3 at address 0x4802A0B0 (I2C\_PSC)
    - Setting SCLL and SCLH : 0x36 by storing 0x36 at register 0x4802A0B4 ( I2C\_SCLL), and register 0x4802A0B8 (I2C\_SCLH)
    - Enable I2C\_EN by storing 0x8000 at register 0x4802A0A4 (I2C\_CON)
    - Load INT\_MIR\_CLEAR2 register address and unmask I2C1INT and INTC int 71
  + Clear all bit in I2C\_IRQSTATUS\_RAW by writing 0xFFFF at register 0x4802A028 (I2C\_IRQSTATUS) @ Write 1 to clear bit.
  + Sending Slave address 0x3C by storing 0x3C at register 0x4802A0AC (I2C\_SA).
  + Determine data counter for transmit by storing at register 0x4802A098 (I2C\_CNT).
  + Load the I2C\_IRQENABLE\_SET register and turn on XRDY\_IE
* **INT\_DIRECTOR:**
  + First thing to do its pull the busy bit form the IRQ\_STUTAS\_RAW register
    - Load IRQ\_STATUS\_RAW register + offset 0x24
    - This is to check if the busy bit its free
    - IF the Busy bit its free proceed otherwise branch to IDLE ( BNE IDLE )
    - Proceeding : Write 0x8603 to the I2C configuration register to configure both the START and STOP bit( We want to do them both at the same time)
      * Load I2C IRQ\_STATUS\_Raw + I2C\_CON and write 0x8603

@ Control Byte

* + - * MOV R1,#0x0

BL Transmit\_Data

@ Function set

* + - * MOV R1,#0x38

BL Transmit\_Data

@ Function set

* + - * MOV R1,#0x39

BL Transmit\_Data

* + - * + BL delay

@ Bias set

* + - * MOV R1,#0x14

BL Transmit\_Data

@ Contrast Set

* + - * MOV R1,#0x78

BL Transmit\_Data

@ Contrast set

* + - * MOV R1,#0x5E

BL Transmit\_Data

@ Follower control

* + - * MOV R1,#0x6D

BL Transmit\_Data

@ Display on

* + - * MOV R1,#0x0C

BL Transmit\_Data

@ Clear Display

* + - * MOV R1,#0x01

BL Transmit\_Data

@ Entry mode set

MOV R1,#0x06

BL Transmit\_Data ( Last data byte)

BL ACCESS\_HAND

BL ARDY\_OFF

BL TDelay

* **Display data on LCD**
  + Clear all bit in I2C\_IRQSTATUS\_RAW by writing 0xFFFF at register 0x4802A028 (I2C\_IRQSTATUS) @ Write 1 to clear bit.
  + Sending Slave address 0x3C by storing 0x3C at register 0x4802A0AC (I2C\_SA).
  + Determine data counter for transmit by storing at register 0x4802A098 (I2C\_CNT).
  + Configure transmit I2C START/STOP by storing 0x8603 at register 0x4802A0A4 (I2C\_CON).
  + Sending the data

@ Control Byte

* + - MOV R1,#0x80

BL Transmit\_Data.

@ Position of cursor

* + - MOV R1,#0x84

BL Transmit\_Data

@ Datasend

* + - MOV R1,#0x40

BL Transmit\_Data

* + - Load address memory storing DATA\_NAME

**DISPLAY**

* + Get char and update pointer ( LDRB R1,[R0],#0x1 )
  + Transmit data to LCD ( BL Transmit\_Data )
  + Decrement counter by One ( SUBS R2,R2,#0x1 )
  + Branch if not equal back to display (when decrement gets to zero finish )

IDLE:

NOP

B IDLE

**TRANSMIT\_DATA**

* Save stack registers
* Load I2C Data Register (I2C\_DATA) 0x4802A09C
* Storing data that want to transmit at register I2C\_DATA
* Branch to WAIT1
* BL XRDY\_OFF
* Restore saved registers and return to mainline
* Return to Mainline

**WAIT\_TO\_TRANSMIT ( WAIT1 )**

* Push registers on Stack

**XRDY:**

* Load I2C\_IRQSTATUS\_RAW register 0x4802A024
* Get value from I2C\_IRQSTATUS\_RAW
* Test bit 4
* If Bit = 0 then branch back to XRDY then check again until Bit = 1
* Otherwise (Bit =1) restore saved registers from Stack and Return to mainline

**ACCESS\_HAND:**

* Store registers on STACK

**ARDY:**

* Load I2C\_IRQSTATUS\_RAW register 0x4802A024
* Get value from I2C\_IRQSTATUS\_RAW
* Test bit 2
* If bit=0 branch back to ARDY check again until it bit= 1
* Else restore saved registers from Stack and return to mainline

**XRDY\_OFF**

* Store registers on Stack
* Load I2C\_IRQSTATUS register 0x4802A028
* Storing value 0x10 at register I2C\_IRQSTATUS (0x4802A028)
* restore saved registers and return to mainline

**ARDY\_OFF**

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

**TDELAY**

* restore values for saved registers and return to mainline

**WAIT**

* + **Subtract 1**
  + **Branch if not equal back to WAIT**
  + **Restore saved registers and return to mainline**

.data

DATA\_NAME:

.ascii “Josh Pradera”

.END

**High Level Algorithm PART 2**

@ ECE 372 PROJECT II

@ Part 2

@ Project: Using I2C to send data from BBB to LCD

@ This project displays my name on I2C\_LCD ( New Haven 2x20 LCD display)

@ Josh Pradera

**.text**

**.global** \_start

**\_start:**

@ ENABLE CLOCK FOR I2C1

LDR R0,=0x44E00048 @ Enable Clock for I2C1 (CM\_PER\_I2C1\_CLKCTRL)

**MOV** R1,#0x2 @ Store 2 to enable

STR R1,[R0]

@ CONTROL MODULE I2C1

LDR R8,=0x44e10958 @ use Pin 17 on P9 for SCL

**MOV** R9,#0x72 @ Mode 2

STR R9,[R8] @ Store

LDR R8,=0x44e1095C @ use Pin 18 on P9 for SDA

**MOV** R9,#0x72 @ Mode 2

STR R9,[R8] @ Store

@ INITIALIZE I2C1

LDR R0,=0x4802A0A4 @ LOAD I2C\_CON Address

LDR R1,=0x0 @ Disable I2C\_EN

STR R1,[R0]

LDR R0,=0x4802A010 @ LOAD System Configuration Register

**MOV** R1,#0x0 @ Disable auto Idle

STR R1,[R0]

LDR R0,=0x4802A0B0 @ Load I2C Clock Prescaler Register

**MOV** R1,#0x3 @ Divide by 4

STR R1,[R0]

@ Setting SCLL and SCLH : 0x36

LDR R0,=0x4802A0B4 @ Load I2C\_SCLL Register

**MOV** R1,#0x36

STR R1,[R0]

LDR R0,=0x4802A0B8 @ Load I2C\_SCLH register

**MOV** R1,#0x36

STR R1,[R0]

LDR R0,=0x4802A0A8

**MOV** R1,#0x1

STR R1,[R0]

LDR R0,=0x4802A094 @ Load I2C\_BUF register

**MOV** R1,#0x0 @ Clearing the register will make RXTRSH/TXTRSH = 1

STR R1,[R0]

LDR R0,=0x4802A0A4 @ I2C Configuration Register

LDR R1,=0x8000 @ Enable I2C\_EN

STR R1,[R0]

@ Initialize the INTC for I2C1

LDR R0,=0x4802A0C8 @ Load address of INTC\_MIR\_CLEAR2 register

**MOV** R1, #0x80 @ Unmask INTC INT 71, I2C1INT.

STR R1, [R0] @Write

@ Initializating LCD

LDR R0,=0x4802A028 @ Load IRQSTATUS Register

LDR R1,=0xFFFF @ Write 1 to clear all

STR R1,[R0]

@ Send Slave address 0x3C

**MOV** R1,#0x3C @ Slave address

LDR R0,=0x4802A0AC @ Load I2C Slave Address Register

STR R1,[R0]

**MOV** R1,#10 @ Load Data byte counter

LDR R0,=0x4802A098 @ Load Data Counter Register I2C CNT

STR R1,[R0]

LDR R0,=0x4802A02C @ I2C1 base address + I2C\_IRQENABLE\_SET OFFSET

**MOV** R1, #0x10 @ Set XRDY\_IE bit 4

STR R1, [R0] @ Write

**INT\_DIRECTOR:**

@ Firts Check FOR BB

LDR R0,=0x4802A024 @ I2C\_IRQSTATUS\_RAW Register base address + offset #0x24

TST R1, #0x00 @ Value to determine the Busy Bit is free.

BNE **IDLE**

LDR R0,=0x4802A0A4 @ LOAD I2C\_CON Address

LDR R1,=0x8603 @ I2C START/STOP

STR R1,[R0]

@ 0x0: Control Byte

**MOV** R1,#0x0

BL Transmit\_Data

@ 0x38: Func set

**MOV** R1,#0x38

BL Transmit\_Data

@ 0x39 : Func set

**MOV** R1,#0x39

BL Transmit\_Data

BL TDELAY

@ 0x14: Bias set

**MOV** R1,#0x14

BL Transmit\_Data

@ 0x78: Contrast set

**MOV** R1,#0x78

BL Transmit\_Data

@ 0x5E: Contrast Set

**MOV** R1,#0x5E

BL Transmit\_Data

@ 0x6D: Follower control

**MOV** R1,#0x6D

BL Transmit\_Data

@ 0x0C: Display on

**MOV** R1,#0x0C

BL Transmit\_Data

@ 0x01: Clear Display

**MOV** R1,#0x01

BL Transmit\_Data

@ 0x06: Entry mode set

**MOV** R1,#0x06

BL Transmit\_Data

@ Wait for access

BL ACCESS\_HAND

BL ARDY\_OFF

BL TDELAY

@ Display data on LCD

LDR R0,=0x4802A028 @ Load IRQSTATUS Register

LDR R1,=0xFFFF @ Write 0xFFFF to clear

STR R1,[R0]

@ Send Slave address 0x3C

**MOV** R1,#0x3C @ Slave address

LDR R0,=0x4802A0AC @ Load I2C Slave Address Register

STR R1,[R0]

**MOV** R2,#11 @ Data bytes (Counter)

LDR R0,=0x4802A098 @ Load Data Counter Register I2C CNT

STR R1,[R0]

LDR R0,=0x4802A0A4 @ LOAD I2C\_CON Address

LDR R1,=0x8603 @ I2C START/STOP

STR R1,[R0]

@ 0x80: Control byte

**MOV** R1,#0x80

BL Transmit\_Data

@ 0x84: Position start after the first 4 spaces of the LCD screen

**MOV** R1,#0x84

BL Transmit\_Data

@ 0x40: Send Data

**MOV** R1,#0x40

BL Transmit\_Data

LDR R0,=DATA\_NAME @ Load address memory DATA\_NAME

**DISPLAY:**

LDRB R1,[R0],#0x1 @ Get char and update the pointer

BL Transmit\_Data @ Transmit data to LCD

SUBS R2,R2,#0x1 @ Decrement counter by one

BNE DISPLAY @ If decrement equal zero then Finish

**IDLE**:

**NOP**

**B** **IDLE**

**Transmit\_Data:**

STMFD R13!,{R0,R14} @ Save stack registers

LDR R0,=0x4802A09C @ Load I2C1 Base address + I2C\_DATA register

STR R1,[R0] @ Store

BL WAIT1

BL XRDY\_OFF

LDMFD R13!,{R0,R14} @ Restore saved registers

**MOV** PC, R14 @ Return to mainline

@ When detecting an interrupt request (XRDY or RRDY type), the CPU can be programmed to write/read

@ the amount of data bytes specified by the corresponding FIFO threshold (I2C\_BUF.TXTRSH + 1 or

@ I2C\_BUF.RXTRSH + 1). In this case, the interrupt condition will be cleared and the next interrupt will

@ be asserted again when the XRDY or RRDY condition will be again met.

**WAIT1:**

STMFD R13!,{R0-R1,R14} @ Store stack registers

**XRDY:**

LDR R0,=0x4802A024 @ I2C1 base address + IRQ STATUS RAW

LDR R1,[R0] @ data from IRQ\_STATUS RAW

TST R1,#0x10 @ Test Bit 4

BEQ XRDY

LDMFD R13!,{R0-R1,R14} @ Restore saved registers

**MOV** PC, R14 @ Return to mainline

@ Registers-ready-for-access interrupt (ARDY) is generated by the I2C when the previously programmed

@ address, data, and command have been performed and the status bits have been updated. This

@interrupt is used to let the CPU know that the I2C registers are ready for access

**ACCESS\_HAND:**

STMFD R13!,{R0-R1,R14} @ Store registers on STACK

**ARDY:**

LDR R0,=0x4802A024 @ I2C1 base address + IRQ STATUS RAW OFFSET

LDR R1,[R0] @ data from IRQ\_STATUS RAW

TST R1,#0x04 @ Test Bit 2

BEQ ARDY

LDMFD R13!,{R0-R1,R14} @ Restore saved registers

**MOV** PC, R14 @ Return to mainline

@The CPU can alternatively poll this bit to write the next transmitted data into the I2C\_DATA register.

**XRDY\_OFF:**

STMFD R13!,{R0-R1,R14} @ Store registers on Stack

LDR R0,=0x4802A028 @ Load I2C\_IRQSTATUS resgister

**MOV** R1,#0x10 @ Clear XRDY

STR R1,[R0]

LDMFD R13!,{R0-R1,R14} @ Restore saved registers

**MOV** PC, R14 @ Return to mainline

@ Mode: I2C Master transmit, Others: STP = 1, ARDY Set Condition:

@ DCOUNT = 0.

@ This read/clear only bit, when set to 1, indicates that the previously

@ programmed data and command (receive or transmit, master or

@ slave) has been performed and status bit has been updated.

@ The CPU uses this flag to let it know that the I2C registers are ready

@ to be accessed again.

**ARDY\_OFF:**

STMFD R13!,{R0-R1,R14} @ Store registers on Stack

LDR R0,=0x4802A028 @ LoadI2C base adrress + I2C\_IRQSTATUS Offset

**MOV** R1,#0x4 @ Clear ARDY

STR R1,[R0]

LDMFD R13!,{R0-R1,R14} @ Restore saved registers

**MOV** PC, R14 @ Return to mainline

**TDELAY:**

STMFD R13!,{R0-R1,R14} @ Store registers on Stack

LDR R0,=0xFFFF @ TDELAY

**WAIT:**

SUBS R0,R0,#1

BNE WAIT

LDMFD R13!,{R0-R1,R14} @ Restore saved registers

**MOV** PC, R14 @ Return to mainline

**.data**

**DATA\_NAME:**

**.ascii** "Josh Pradera"

.END

**PART 3 – This section it’s for extra credit**

The program implements the use of analog output (PWM) to control the brightness of the backlight in the heaven 20x2 LCD.

Pulse Width Modulation (PWM) is used to get analog results with digital means. The use of digital control makes possible to create a square wave by switching a signal ON and OFF. The voltages when the signal goes ON and OFF can range from 5volts (ON) and 0volts (OFF). The duration of the ON time is called the pulse width. To get various analog values the pulse with is changed. The signal will be steady between 0v and 5v controlling the brightness with different duty cycles.

**Low level Algorithm**

* 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 GPIO2\_1 for detecting falling edge by storing 0x00000002 to 0x481AC14C (0x481AC000 (GPIO2\_Base) + 0x14C (offset of GPIO\_FALLINGEDGE\_DETECT)
* Set up GPIO2\_1 for IRQ enable by storing 0x00000002 to 0x481AC034 (0x481AC000 (GPIO2\_Base) + 0x34 (offset of GPIO\_IRQSTATUS\_SET0)
* INITIALIZE INTC

Enable GPIO\_2 interrupt by enable Int number 32 of the INTC (write 0x10000000 to 0x482000A8: 0x48200000 (base address for INTC) + 0xA8 offset for INTC\_MIR\_CLEAR1).

* 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).

IDLE 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 0 (int number 32) of the current value stored in INTC\_PENDING\_IRQ1 (0x482000B8: 0x48200000 (base address for INTC) + 0xB8 offset).

If bit 0 =0 ( Not GPIO interrupt)

Go to PASS\_ON

Else

{

Check interrupt coming from GPIO2\_1 by reading GPIO2\_IRQ\_STATUS REGISTER SET 0 at 0x481AC02C, check bit 1( #0x00000002 )

If Yes  Go to BUTTON\_SVC

Else No  Go to PASS\_ON

}

PASS\_ON: Restore registers from STACK

Pass execution onto wait IDLE

1. **Button Service Procedure :**

**Button\_SVC**

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

Writing 0x00000002 to 0x481AC02C (0x481AC000 (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**

**High Level Algorithm**

@ ECE 372 PROJECT II

@ PART3

@ Project: Using I2C to send data from BBB to LCD

@ This project displays my name on I2C\_LCD ( New Haven 2x20 LCD display)

@ Implementing PWM to chnage the brightness of the LCD

@ Josh Pradera

**.text**

**.global** \_start

**.global** INT\_DIRECTOR

**\_start:**

@ Initialize the STACKS

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, #0x01E00000 @ Load value to turn off

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

@ Turn ON GIPO 1 and 2 CLK

@ 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

@ Initialize GPIO

@Enable GPIO1\_21 as an output

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

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

LDR R7,=0xFE1FFFFF @ Word to enable GPIO1\_21,22,23,24 as output

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

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

@ Initialize the Falling Edge

LDR R0,=0x481AC000 @ Base address for GPIO2 registers

**ADD** R1, R0, #0x14C @ Falling edge detect

LDR R2, =0x00000002 @ Detect by GPIO2\_1

LDR R3, [R1] @ Read current value in GPIO1\_FALLING\_EDGE\_DETECT

ORR R3, R3, R2 @ Modify the current value

STR R3, [R1] @ Write modified value back to GPIO1\_FALLINGEDGEDECTECT

**ADD** R1, R0, #0x34 @ Get address of GPIO1\_IRQSTATUS\_SET0

STR R2, [R1] @ Enable GPIO1\_31 request on GPIO1\_IRQSTATUS\_SET0

@ 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, #0x10000000 @ Value to unmask INTC INT 92, DMTIMER4, POINTR\_PEND

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

@ Enable Button Interrupt

LDR R1,=0x482000A8 @ Address INTC\_MIR\_CLEAR1

**MOV** R2, #0x00000001 @ Value to unmask INTC INT 32

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

@ Initialize PWM

@ This register manages the PWMSS1 clocks.( CM\_PER\_EPWMSS1\_CLKCTRL )

LDR R0,=0x44E000CC @ Address CM\_PER\_EPWMSS1\_CLKCTRL

**MOV** R1,#0x2 @ Enable Clock for PWM

STR R1,[R0]

@ Module Control PWM Module

LDR R0,=0x44E10664 @ Address CONTROL MODULE address (0x44E10000)of pwmss\_ctrl (Offset 664)

**MOV** R1,#0x2 @ Select Timebase clock enable for PWMSS1

STR R1,[R0]

@ Pin PWM output Pin Mux

LDR R0,=0x44E10848 @ Address of conf\_gpmc\_a2 (conf\_gpmc\_a1--844 )

**MOV** R1,#0x6 @ Choose Mode 6

STR R1,[R0]

@ Initialize PWM

LDR R0,=0x48302200 @ Address Time-Base Control Register

LDR R1,=0xC000 @ Select Time-base Clock is divide 1

@ Select Up-count mode

@ Select Free run

STRH R1,[R0]

@ Set the period

LDR R0,=0x4830220A @ Address Time-Base Period Register

LDR R1,=0xC350 @ Select Value 0xC350 for Time-base period

STRH R1,[R0]

@ Comparing a value of the period to get dusty cycle

LDR R0,=0x48302212 @ Address Counter-Compare A Register

LDR R1,=0x186 @ Select 2.5% duty cycle

STRH R1,[R0]

@ Configuring Output

LDR R0,=0x48302216 @ Output A (EPWMxA)

**MOV** R1,#0x1A @ Select force EPWM xA output high

@ When counter equals active CMPA, counter increase

STRH R1,[R0]

LDR R4,=value @ R4 Register hold memory store value of duty cycle

**MOV** R5,#0x0 @ R5 Register is pointer control memory

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

@ ENABLE CLOCK FOR I2C1

LDR R0,=0x44E00048 @ Enable Clock for I2C1 (CM\_PER\_I2C1\_CLKCTRL)

**MOV** R1,#0x2 @ Store 2 to enable

STR R1,[R0]

@ CONTROL MODULE I2C1

LDR R8,=0x44e10958 @ use Pin 17 on P9 for SCL

**MOV** R9,#0x72 @ Mode 2

STR R9,[R8] @ Store

LDR R8,=0x44e1095C @ use Pin 18 on P9 for SDA

**MOV** R9,#0x72 @ Mode 2

STR R9,[R8] @ Store

@ INITIALIZE I2C1

LDR R0,=0x4802A0A4 @ LOAD I2C\_CON Address

LDR R1,=0x0 @ Disable I2C\_EN

STR R1,[R0]

LDR R0,=0x4802A010 @ LOAD System Configuration Register

**MOV** R1,#0x0 @ Disable auto Idle

STR R1,[R0]

LDR R0,=0x4802A0B0 @ Load I2C Clock Prescaler Register

**MOV** R1,#0x3 @ Divide by 4

STR R1,[R0]

@ Setting SCLL and SCLH : 0x36

LDR R0,=0x4802A0B4 @ Load I2C\_SCLL Register

**MOV** R1,#0x36

STR R1,[R0]

LDR R0,=0x4802A0B8 @ Load I2C\_SCLH register

**MOV** R1,#0x36

STR R1,[R0]

LDR R0,=0x4802A0A8

**MOV** R1,#0x1

STR R1,[R0]

LDR R0,=0x4802A094 @ Load I2C\_BUF register

**MOV** R1,#0x0 @ Clearing the register will make RXTRSH/TXTRSH = 1

STR R1,[R0]

LDR R0,=0x4802A0A4 @ I2C Configuration Register

LDR R1,=0x8000 @ Enable I2C\_EN

STR R1,[R0]

@ Initializating LCD

LDR R0,=0x4802A028 @ Load IRQSTATUS Register

LDR R1,=0xFFFF @ Write 1 to clear all

STR R1,[R0]

@ Send Slave address 0x3C

**MOV** R1,#0x3C @ Slave address

LDR R0,=0x4802A0AC @ Load I2C Slave Address Register

STR R1,[R0]

**MOV** R1,#10 @ Load Data byte counter

LDR R0,=0x4802A098 @ Load Data Counter Register I2C CNT

STR R1,[R0]

LDR R0,=0x4802A0A4 @ LOAD I2C\_CON Address

LDR R1,=0x8603 @ I2C START/STOP

STR R1,[R0]

@ 0x0: Control Byte

**MOV** R1,#0x0

BL Transmit\_Data

@ 0x38: Func set

**MOV** R1,#0x38

BL Transmit\_Data

@ 0x39 : Func set

**MOV** R1,#0x39

BL Transmit\_Data

BL TDELAY

@ 0x14: Bias set

**MOV** R1,#0x14

BL Transmit\_Data

@ 0x78: Contrast set

**MOV** R1,#0x78

BL Transmit\_Data

@ 0x5E: Contrast Set

**MOV** R1,#0x5E

BL Transmit\_Data

@ 0x6D: Follower control

**MOV** R1,#0x6D

BL Transmit\_Data

@ 0x0C: Display on

**MOV** R1,#0x0C

BL Transmit\_Data

@ 0x01: Clear Display

**MOV** R1,#0x01

BL Transmit\_Data

@ 0x06: Entry mode set

**MOV** R1,#0x06

BL Transmit\_Data

@ Wait for access

BL ACCESS\_HAND

BL ARDY\_OFF

BL TDELAY

@ Display data on LCD

LDR R0,=0x4802A028 @ Load IRQSTATUS Register

LDR R1,=0xFFFF @ Write 0xFFFF to clear

STR R1,[R0]

@ Send Slave address 0x3C

**MOV** R1,#0x3C @ Slave address

LDR R0,=0x4802A0AC @ Load I2C Slave Address Register

STR R1,[R0]

**MOV** R2,#11 @ Data bytes (Counter)

LDR R0,=0x4802A098 @ Load Data Counter Register I2C CNT

STR R1,[R0]

LDR R0,=0x4802A0A4 @ LOAD I2C\_CON Address

LDR R1,=0x8603 @ I2C START/STOP

STR R1,[R0]

@ 0x80: Control byte

**MOV** R1,#0x80

BL Transmit\_Data

@ 0x84: Position start after the first 4 spaces of the LCD screen

**MOV** R1,#0x84

BL Transmit\_Data

@ 0x40: Send Data

**MOV** R1,#0x40

BL Transmit\_Data

LDR R0,=DATA\_NAME @ Load address memory DATA\_NAME

**DISPLAY:**

LDRB R1,[R0],#0x1 @ Get char and update the pointer

BL Transmit\_Data @ Transmit data to LCD

SUBS R2,R2,#0x1 @ Decrement counter by one

BNE DISPLAY @ If decrement equal zero then Finish

@ 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

**IDLE**:

**NOP**

**B** **IDLE**

**INT\_DIRECTOR:**

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

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

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

TST R1,#0x00000001 @ TEST BIT 0

BEQ PASS\_ON @ Not from GPIOINT1A, Button, go to back to wait loop, Else

LDR R0,=0x481AC02C @ Load GPIO1\_IRQSTATUS\_0 register address

LDR R1,[R0] @ Read Status register

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

BNE BUTTON\_SVC @ If bit 1=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 R0,=0x481AC02C

LDR R1,=0x00000002 @ Value to turn off GPIO2\_1 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

**CMP** R5,#40 @ Check Pointer is out of 40

MOVEQ R5,#0x0 @ If yes Reset R5=0

ADDNE R5,#0x4 @ If No Increment pointer by 4

@ Setting period after pressing button

LDR R0,=0x48302212 @ Address Counter-Compare A Register

LDR R1,[R4,R5] @ Get value duty cycle from Memory

STRH R1,[R0] @ Store

**Done:**

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

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

**Transmit\_Data:**

STMFD R13!,{R0,R14} @ Save stack registers

LDR R0,=0x4802A09C @ Load I2C1 Base address + I2C\_DATA register

STR R1,[R0] @ Store

BL WAIT1

BL XRDY\_OFF

LDMFD R13!,{R0,R14} @ Restore saved registers

**MOV** PC, R14 @ Return to mainline

@ When detecting an interrupt request (XRDY or RRDY type), the CPU can be programmed to write/read

@ the amount of data bytes specified by the corresponding FIFO threshold (I2C\_BUF.TXTRSH + 1 or

@ I2C\_BUF.RXTRSH + 1). In this case, the interrupt condition will be cleared and the next interrupt will

@ be asserted again when the XRDY or RRDY condition will be again met.

**WAIT1:**

STMFD R13!,{R0-R1,R14} @ Store stack registers

**XRDY:**

LDR R0,=0x4802A024 @ I2C1 base address + IRQ STATUS RAW

LDR R1,[R0] @ data from IRQ\_STATUS RAW

TST R1,#0x10 @ Test Bit 4

BEQ XRDY

LDMFD R13!,{R0-R1,R14} @ Restore saved registers

**MOV** PC, R14 @ Return to mainline

@ Registers-ready-for-access interrupt (ARDY) is generated by the I2C when the previously programmed

@ address, data, and command have been performed and the status bits have been updated. This

@interrupt is used to let the CPU know that the I2C registers are ready for access

**ACCESS\_HAND:**

STMFD R13!,{R0-R1,R14} @ Store registers on STACK

**ARDY:**

LDR R0,=0x4802A024 @ I2C1 base address + IRQ STATUS RAW OFFSET

LDR R1,[R0] @ data from IRQ\_STATUS RAW

TST R1,#0x04 @ Test Bit 2

BEQ ARDY

LDMFD R13!,{R0-R1,R14} @ Restore saved registers

**MOV** PC, R14 @ Return to mainline

@The CPU can alternatively poll this bit to write the next transmitted data into the I2C\_DATA register.

**XRDY\_OFF:**

STMFD R13!,{R0-R1,R14} @ Store registers on Stack

LDR R0,=0x4802A028 @ Load I2C\_IRQSTATUS resgister

**MOV** R1,#0x10 @ Clear XRDY

STR R1,[R0]

LDMFD R13!,{R0-R1,R14} @ Restore saved registers

**MOV** PC, R14 @ Return to mainline

@ Mode: I2C Master transmit, Others: STP = 1, ARDY Set Condition:

@ DCOUNT = 0.

@ This read/clear only bit, when set to 1, indicates that the previously

@ programmed data and command (receive or transmit, master or

@ slave) has been performed and status bit has been updated.

@ The CPU uses this flag to let it know that the I2C registers are ready

@ to be accessed again.

**ARDY\_OFF:**

STMFD R13!,{R0-R1,R14} @ Store registers on Stack

LDR R0,=0x4802A028 @ LoadI2C base adrress + I2C\_IRQSTATUS Offset

**MOV** R1,#0x4 @ Clear ARDY

STR R1,[R0]

LDMFD R13!,{R0-R1,R14} @ Restore saved registers

**MOV** PC, R14 @ Return to mainline

**TDELAY:**

STMFD R13!,{R0-R1,R14} @ Store registers on Stack

LDR R0,=0xFFFF @ TDELAY

**WAIT:**

SUBS R0,R0,#1

BNE WAIT

LDMFD R13!,{R0-R1,R14} @ Restore saved registers

**MOV** PC, R14 @ Return to mainline

**.align** 2

**SYS\_IRQ:** @ Location to store system IRQ address

**.data**

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

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

**.data**

**DATA\_NAME:**

**.ascii** "Josh Pradera"

**STACK1:** .rept 1024

**.word** 0x0000

.endr

**STACK2:** .rept 1024

**.word** 0x0000

.endr

.END