**Target**: Beagle Bone Black I2C1 Controller

**Project Outline**

*Chapter 7*

*Review SCL and SDA signals.*

*Start Condition*

*Stop Condition*

*ACK*

*High-level initialization*

*Inspect P9 connector of the Beagle Bone Black for pin mappings.*

*Identify the I2C1\_SCL pin*

*Find the Control Module register to change Pin17 to the I2C1\_SCL signal.*

*Identify the I2C1\_SDA pin*

*Find the Control Module register to change Pin18 to the I2C1\_SDA signal.*

*The display module should already be physically connected already.*

*Initialize the Clock Module for I2C1*

*Study I2C section of the Sitara manual*

*Analyze 7-bit addressing mode*

*Follow high-level list of steps in the section to initialize the I2C controller.*

*Find the settings to reach 12 MHz clock*

*Find the settings to get a 100 kbps SCL for standard mode operation (F/S).*

*Create an initialization pin map list for the required registers.*

*Registers for initialization*

*Registers for transmission start*

*Registers for byte transfer when controller is ready*

*Construct the High-level algorithm for initialization and transmission start.*

*Should be the same as the chapter 7 example, minus the slave read.*

*Construct the Low-level algorithm initialization and transmission start.*

*Study the manual to determine how to get it in the desired mode.*

*Determine how you send characters to display*

*Make a list of desired initialization words*

*Make a list of words needed to display your name.*

*Either single height on two lines, or double height display mode.*

*Complete the High-level and low-level algorithms with these steps included.*

*Create a polled version for handshaking of the program.*

*Receive TA/Instructor sign-off*

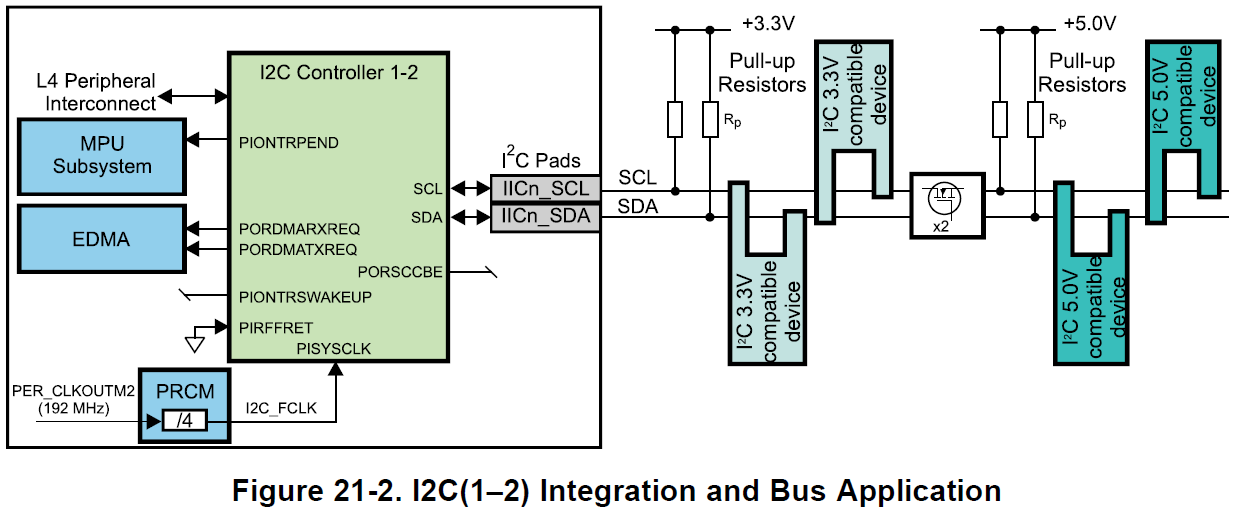
*Modify Algorithm to implement the handshaking on an interrupt basis with the interrupt controller.*

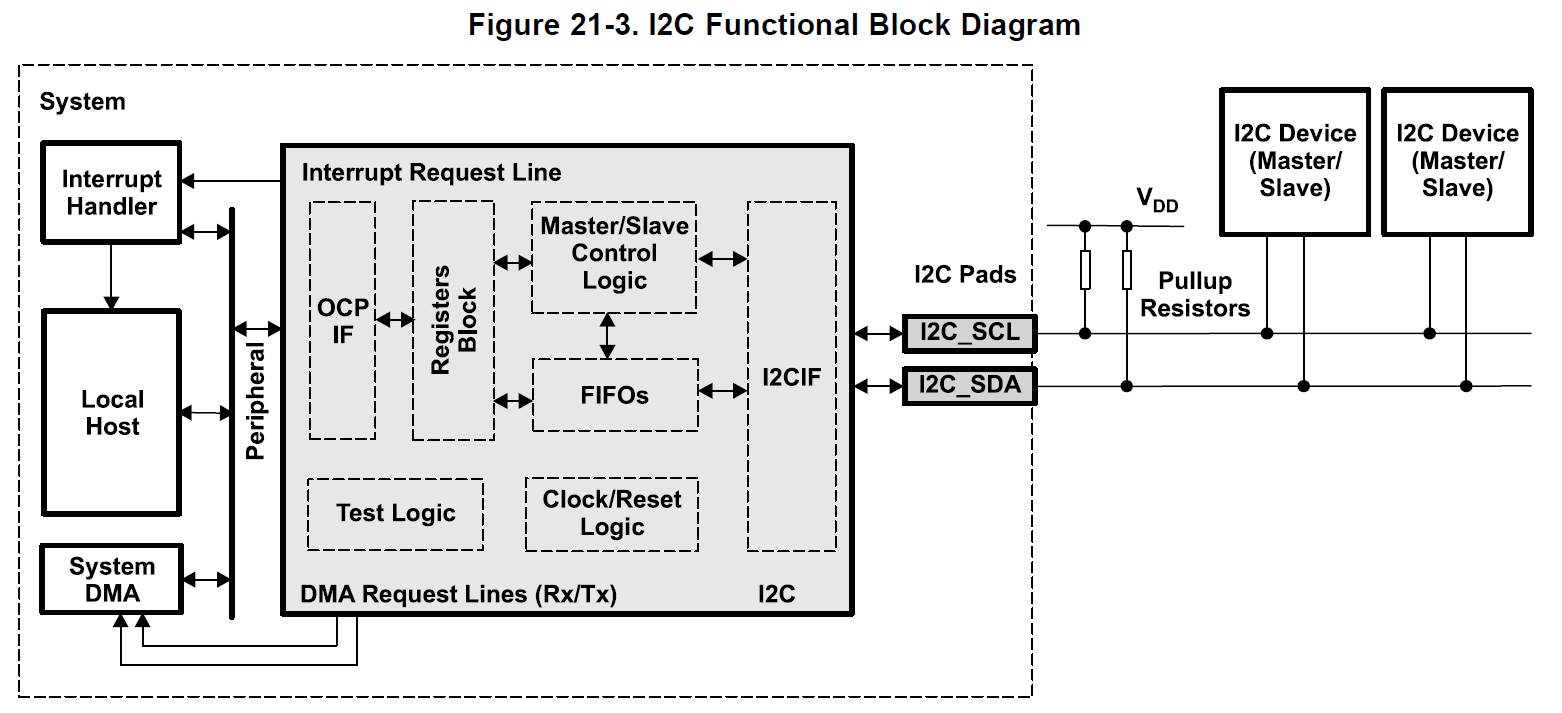
*Modify Program to implement handshaking on an interrupt basis.*

*Receive TA/Instructor Sign-off*

*Find how to make the display rotate around the screen to the right in a loop. Or blink on and off.*

**Diagrams for Reference:**





**Chapter 7:**

**Review SCL and SDA signals.**

* Both lines driven by open drain or open collector transistors.
  + The lines require pull up resistors because of this.
* Serial Data (SDA)
* Serial Clock Line (SCL)

**Start Condition.**

* At start of transmission (if the bus is available and that a master wants to send a message to a slave) the master pulls the SDA line from high to low, while the SCL line is high.
* The SCL line is then pulsed, shifting out the data bits on SDA synchronously with the SCL pulses.
  + The most significant bit is shifted out first on the SDA line.
* If the slave receives 8 bits correctly, it synchronously pulls the SDA line low as an acknowledge signal to the master
* The SCL line can be held low by the slave if time is needed to process the byte, forcing the master to insert wait states.
* When the slave releases the SCL line and it is pulled high by the external pull-up resistor. The master can then send another byte.
* If no acknowledge signal after a byte is generated, the master can either generate a stop condition on the bus to abort/end the transfer or assert a repeated start condition on the bus to start a new transmission.
* For a repeated start condition, the SDA line is pulled low while the SCL line is high.
* I2C\_IRQSTATUS: **BB** = 1

**Stop Condition.**

* The master allows the SDA line to transition from low to high while the SCL line is high.
* For a repeated start condition, the SDA line is pulled low while the SCL line is high.
* Either condition option is available after any transmission.
* I2C\_IRQSTATUS: **BB** = 0

**ACK.**

* The bit following each byte transmitted.
* The master will signal the end of the transmission to the slave by not pulling the acknowledge bit low for the last byte that was clocked out of the slave.

**Data transfer.**

* For any transmission on the bus, the master will first send out an address byte.
* 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 least significant bit of this byte will be a 0 for a write operation and a 1 for a read operation.
* After the master receives an acknowledge signal from the addressed slave, it then clocks out the data byte of the message.

**High-level initialization.**

**Inspect P9 connector of the Beagle Bone Black for pin mappings.**

* **At this point there was some confusion about the P8 Pin mappings for the I2C1 signals. Here is the information on the P9 connector from the BBB manual which cleared the confusion:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **PIN** | **PROC** | **NAME** | **MODE0** | **MODE1** | **MODE2** | **MODE3** | **MODE4** |
| 18 | B16 | I2C1\_SDA | spi0\_d1 | mmc1\_sdwp | I2C1\_SDA | ehrpwm0\_tripzone | pr1\_uart0\_rxd |
| 17 | A16 | I2C1\_SCL | spi0\_cs0 | mmc2\_sdwp | I2C1\_SCL | ehrpwm0\_synci | pr1\_uart0\_txd |

* **CONTROL\_MODULE** Base Address:

**Region Name Start Address (hex) End Address (hex) Description**

Control Module 0x44E10000 0x44E11FFF Control Module Registers

**Identify the I2C1\_SCL pin.**

* **PIN:** 17 **PROC:** A16 **NAME:** I2C1\_SCL pr1\_uart0\_txd

*From the TI MUX tool: AM33XX\_IOPAD(0x1****5c****, PIN\_INPUT |* ***MUX\_MODE2****) /\* (A16)* ***spi0\_cs0****.I2C1\_SCL \*/*

**Find the Control Module register to change Pin17 to the I2C1\_SCL signal.**

**Offset Control Module Register Write Value** **(for MODE2 w/o pullup)**

0x95Cconf\_spi0\_cs0 0x2(010b)

* 0x2 used because no pull up/down required and receiver enabled.

**Identify the I2C1\_SDA pin.**

* **PIN:** 18 **PROC:** B16 **NAME:** I2C1\_SDA pr1\_uart0\_rxd

*From the TI MUX tool: AM33XX\_IOPAD(0x1****58****, PIN\_INPUT |* ***MUX\_MODE2****) /\* (B16)* ***spi0\_d1****.I2C1\_SDA \*/*

**Find the Control Module register to change Pin18 to the I2C1\_SDA signal.**

**Offset Control Module Register Write Value (for MODE2 w/o pullup)**

0x958conf\_spi0\_d1 0x2(010b)

* 0x2 used because no pull up/down required and receiver enabled.
* **Initially, it wasn’t clear whether the printed circuit board with the LCD module had a pull-up resistor, so the initial assumption was that it has an external pull-up resistor to simplify settings for the students. Hence the write value to the pins didn’t include initialization for pull-up resistors on the signal lines.**

**The display module should already be physically connected already.**

**Initialize the Clock Module for I2C1.**

* **The I2C section didn’t explicitly state what Interface Register controls the Interface Clock for the I2C1 module. Looking into the Power, Reset, and Clock Management section, Table 8-30 provided the missing information.**
* **CLOCK MODULE PERIPHERAL REGISTER** Base Address:

**Region Name Start Address (hex) End Address (hex) Description**

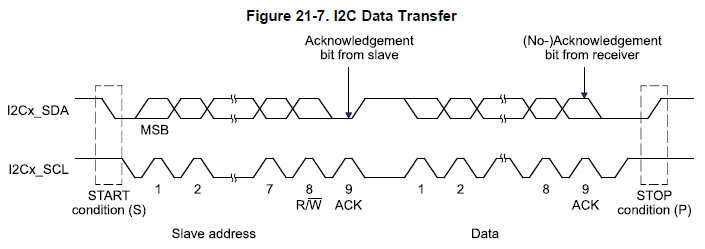
CM\_PER 0x44E00000 0x44E003FF Clock Module Peripheral Registers

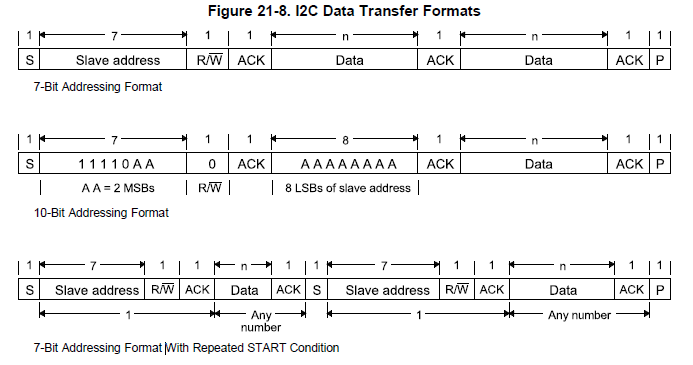
**Offset Clock Module Peripheral Register Write Value (To Enable)**

0x48 CM\_PER\_I2C1\_CLKCTRL 0x2

**Study I2C section of the Sitara manual:**

**Analyze 7-bit addressing mode.**





* **DCOUNT** = # of Data Bytes capable of being Transmitted or Received.
* 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**.

**Receive/Transmit Sequence**

bit **7** + bit **6** + bit **5**+ bit **4** + bit **3** + bit **2** + bit **1** + bit **0** + **R/** bit (when addressing) + **Acknowledge** bit(when in acknowledge mode)

* There are always 8 bits after the **start condition**.
* In **acknowledge mode**, an extra bit is inserted after each byte.
* 7 – bit addressing format with or without repeated start condition
  + - The first byte is composed of 7 MSB slave address bits and 1 ‘LSB R/ bit.
* R/ bit determines the direction of transmission of the following data Bytes.
* **Master Modes**:
  + Transmitter**:**
    - Assembled data is shifted out on the serial data line **SDA** in sync with the self-generated clock pulses on the serial clock line **SCL**.
    - Clock pulses are inhibited and **SCL** held low when the intervention of the processor is required (**XUDF**) after a byte has been transmitted.
  + Receiver**:** 
    - Mode can only be entered from master transmitter mode.
    - After the slave address byte and bit **R/** are transmitted, the mode is entered if **R/** is high (read). Serial data bits received on bus line **SDA** are shifted in sync with the self-generated clock pulses on **SCL**.
    - Clock pulses are inhibited and **SCL** held low when the intervention of the processor is required (**ROVR**) after a byte has been transmitted.
    - At end of transfer, the stop condition is generated.
* **Slave Modes:**
  + Transmitter:
    - Mode can only be entered from slave receiver mode.
    - The slave transmitter is entered if the slave address byte is the same as its own address and **R/** has been transmitted high (master receiver/read)
    - Slave transmitter shifts the serial data out on the data line **SDA** in sync with the clock pulses that are generated by the master device.
    - It doesn’t generate the clock, but can hold the clock line **SCL** low while intervention of the CPU is required (**XUDF**).
  + Receiver:
    - Serial data bits received on the bus line **SDA** are shifted-in in sync with the clock pulses on **SCL** that are generated by the master device.
    - It doesn’t generate the clock, but can hold the clock line **SCL** low while intervention of the CPU is required (**ROVR**).

**Follow high-level list of steps in the section to initialize the I2C controller.**

* **I2C1 REGISTER** Base Address:

**Region Name Start Address (hex) End Address (hex) Description**

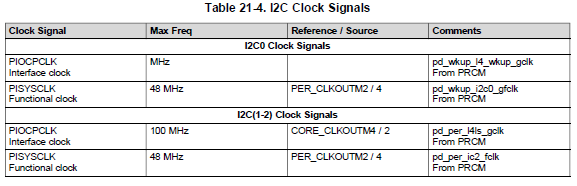
I2C1 0x4802A000 0x4802BFFF I2C1 Registers

* **Configure the Module Before Enabling it.**
  + Program the Prescalar.
    - Obtain an approximately 12-MHz I2C module Clock.
    - I2C\_PSC = x ; this value is to be calculated and is dependent on the System clock frequency.
    - **PER\_CLKOUTM2** = 192 MHz on Reset
    - **SCLK** (Functional Clock) = 48 MHz on Reset (=192/4)
    - **ICLK** Target of 12 MHz

**Offset I2C1 Register Write Value**

0xB0 I2C\_PSC (Clock Prescalar) 0x03

* + Program the I2C clock.
    - Obtain 100Kbps
    - SCLL = x and SCLH = x ; these values are to be calculated and are dependent on the System clock frequency.
    - **tLOW =** (**SCLL** + 7) \* t**ICLK**
    - **tHIGH** = (**SCLH** + 5) \* t**ICLK**
    - Assuming 50% duty cycle @ 100kbps
    - T = 10s = t**LOW** +t**HIGH** = 2t**LOW**
    - t**LOW** = 5 s
    - t**HIGH** = 5s
    - t**ICLK** = 83ns



**Offset I2C1 Register Write Value**

0xB4 I2C\_SCLL (SCL Low Time Register) 0x35

0xB8 I2C\_SCLH (SCL High Time Register) 0x37

* + Configure its own address.
    - I2C\_OA = x ; Only in case of I2C operating mode (F/S mode).
    - Sets address of master, the top 3 bits must be cleared by software when not operating on 10-bit mode.

**Offset I2C1 Register Write Value**

0xA8 I2C\_OA (Own Address Register) 0x001

* + Take the I2C module out of reset.
    - I2C\_CON:I2C\_EN = 1;

**Offset I2C1 Register Write Value (To Enable bit only)**

0xA4 I2C\_CON (Configuration Register) 0x8000

* **Initialization Procedure**
  + Configure the I2C Mode Register.
    - I2C\_CON bits.

**Offset I2C1 Register Write Value (To configure Mode)**

0xA4 I2C\_CON (Configuration Register) 0x600 (R-M-W, should be enabled)

1. Enable Interrupt Masks.
   * + I2C\_IRQENABLE\_SET, if using interrupt for Tx/Rx of data.
     + In **Polled mode** **XRDY** and **RRDY** are disabled, as well as DMA.
       - All interrupts will be disabled in **Polled Mode**.
     + In **Interrupt Mode** **XRDY** and **RRDY** will be enabled.

**Offset I2C1 Register Write Value**

0x2C I2C\_IRQENABLE\_SET (Interrupt Enable Set Register) 0x0018 (Interrupts Enabled)

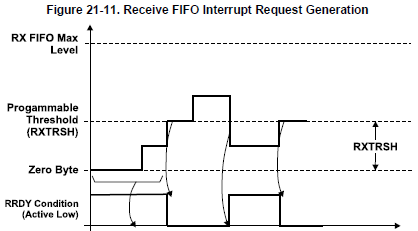
**-OR- -OR- -OR-**

0x2C I2C\_IRQENABLE\_SET (Interrupt Enable Set Register) 0x0000 (Polling, No Interrupts)

* + - 1 interrupt to MPU Subsystem (**I2C1INT**).

**Int Number Acronym/name Source Signal Name**

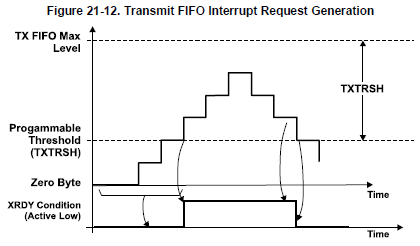
71 I2C1INT I2C1 POINTRPEND



* + - An interrupt is generated whenever the signal is active.
    - The **RRDY** signal must be cleared by the CPU by writing a 1 in the corresponding interrupt flag.
    - If the condition is still present after clearing another interrupt will be generated.
      * Ensure the condition has changed first (I.e. RX FIFO count is below **RXTRSH**).
    - The local host can be configured to read the value of the RX FIFO Threshold + 1.
    - When detecting an interrupt request the CPU can be programmed to write/read the amount of data bytes specified by the corresponding FIFO threshold **TXTRSH** + 1 or **RXTRSH** + 1.

**Offset I2C1 Register Write Value (Clear RRDY)**

0x28 I2C\_IRQSTATUS (Status Register) 0x0008



* + - Interrupt request is generated when the condition is achieved, When the TX FIFO is empty.
    - When detecting an interrupt request the CPU can be programmed to write/read the amount of data bytes specified by the corresponding FIFO threshold **TXTRSH** + 1 or **RXTRSH** + 1.

**Offset I2C1 Register Write Value (Clear XRDY)**

0x28 I2C\_IRQSTATUS (Status Register) 0x0010

* + - Data Buffer Register (IDBR or I2C\_BUF on Sitara) influences the Interrupts.

**Offset I2C1 Register Write Value (To initialize)**

0x94 I2C\_BUF (Buffer Configuration Register) 0x0000 (Threshold is 1 byte)

1. Enable the DMA and Program the DMA controller.
   * + ~~I2C\_BUF and I2C\_DMA/RX/TX/ENABLE\_SET only in case of I2C operating mode (F/S mode), and if using DMA for Tx/Rx of data.~~ (NOT USED in this project).

* **Configure Slave Address and DATA Counter Registers**
  + Configure slave address using I2C\_SA.
    - From the Newhaven NHD‐C0220BiZ‐FSW‐FBW‐3V3M data sheet:

Slave Address = 0x3C

* + - From the Sitronix Dot Matrix LCD Controller/Driver:

Slave Address = 0x3C to 0x3F

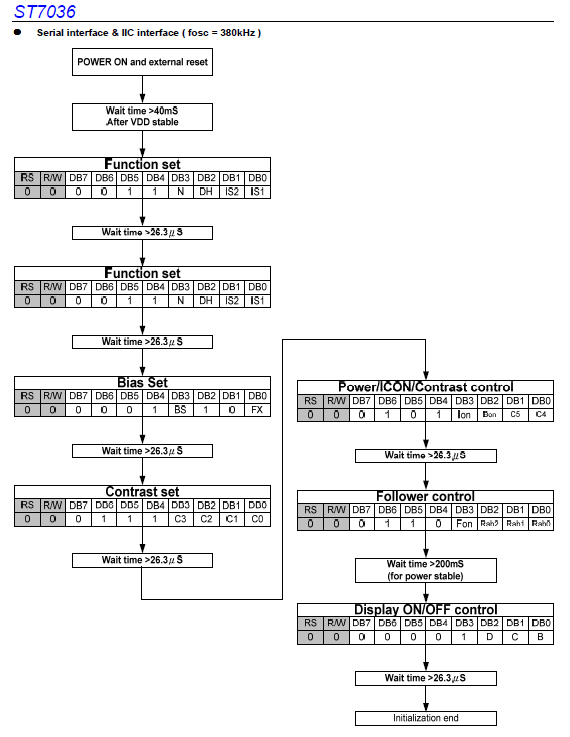
**Offset I2C1 Register Write Value (To configure Mode)**

0xAC I2C\_SA (Slave address Register) 0x3C

* + Configure the number of bytes associated with transfer.
    - I2C\_CNT is a 16-bit counter which decrements by 1 for every byte received or transmitted through I2C.
    - Must be re-programmed after each transfer operation stop condition.
    - According to the example given by the device controller data sheet at least 7 bytes must be transferred for the first

**Offset I2C1 Register Write Value (# of Bytes before STOP)**

0x98 I2C\_CNT (Data Count Register) 0x9



* **Initiate a Transfer**
  + Poll the bus busy (BB) bit in the I2C Status Register I2C\_IRQSTATUS\_RAW.
    - This bit indicates the state of the serial bus.
    - In master mode, controlled by software.

**Offset I2C1 Register Read Mask (Read BB)**

0x24 I2C\_IRQSTATUS\_RAW (I2C status Raw Register) 0x1000

* + If the register is cleared to 0, configure START/STOP I2C\_CON: **STT** / I2C\_CON: **STP** condition to initiate a transfer, only in the case of I2C operating mode (F/S mode).
    - To start a transmission with a start condition:
      * **MST** = 1
      * **TRX** = 1
      * **STT** = 1 (Resets to 0 after Start Condition generated)

**Offset I2C1 Register Write Value (To Create Start)**

0xA4 I2C\_CON (Configuration Register) 0x1 (Read Modify (OR) Write)

* + - To end a transmission with a stop condition
      * **STP** = 1 (Resets after Stop condition generated)
        + Stop condition will generate when **DCOUNT** passes 0.
        + If this bit is not set before **DCOUNT** = 0 the stop condition is not generated and the **SCL** line is held low by the master.
        + The master can then restart the transmission by setting the **STT** bit to 1.

**Offset I2C1 Register Write Value (To Create Stop Condition)**

0xA4 I2C\_CON (Configuration Register) 0x2 (Read Modify (OR) Write)

* **Receive Data**
  + Poll the receive data ready interrupt flag bit **RRDY** in the I2C Status Register I2C\_IRQSTAUTS\_RAW.

**Offset I2C1 Register Read Mask (Read RRDY)**

0x24 I2C\_IRQSTATUS\_RAW (Interrupt Status Raw Register) 0x00000008

* + Use the **RRDY** interrupt from the I2C\_IRQENABLE\_SET register, to read the received data in the data receive register I2C\_DATA.
    - But only for Interrupt version of program
* **Transmit Data**
  + Poll the transmit data ready interrupt flag bit (**XRDY**) in the I2C status register I2C\_IRQSTATUS\_RAW.

**Offset I2C1 Register Read Mask (Read XRDY)**

0x24 I2C\_IRQSTATUS\_RAW (I2C Status Raw Register) 0x00000010

* + Use the **XRDY** interrupt from I2C\_IRQENABLE\_SET register to write data into the data transmit register, I2C\_DATA.
    - But only for Interrupt version of program

**Find the settings to reach 12 MHz clock.**

* Work done previously in High-Level Initialization steps, result:

**Find the settings to get a 100 kbps SCL for standard mode operation (F/S).**

* I2C Standard, low-speed operation is 0 to 100KHz, work done previously in High-Level Initialization Steps, Result:

**Create an initialization pin map list for the required registers.**

* **Module Configuration Registers:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **I2C\_PSC** Values for 12 MHz **ICLK** | | | | | | | | | |
| **Field** | **RESERVED** | **PSC** | | | | | | | |
| **Bits** | **31-8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Write**  **Decimal** | 0 | 3 | | | | | | | |
| **Write**  **Binary** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| **Write**  **Hex** | 0 | 0 | | | | 3 | | | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **I2C\_SCLL** Values for 5us **tLOW** | | | | | | | | | |
| **Field** | **RESERVED** | **SCLL** | | | | | | | |
| **Bits** | **31-8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Write**  **Decimal** | 0 | 53 | | | | | | | |
| **Write**  **Binary** | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| **Write**  **Hex** | 0 | 3 | | | | 5 | | | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **I2C\_SCLH** Values for 5us **tHIGH** | | | | | | | | | |
| **Field** | **RESERVED** | **SCLH** | | | | | | | |
| **Bits** | **31-8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Write**  **Decimal** | 0 | 55 | | | | | | | |
| **Write**  **Binary** | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| **Write**  **Hex** | 0 | 3 | | | | 7 | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **I2C\_OA** Values for Master Address | | | | | | | | | | | | | |
| **Field** | **RESERVED** | **RESERVED** | **RESERVED** | **OA** | | | | | | | | | |
| **Bits** | **31-12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Write**  **Binary** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **Write**  **Hex** | 0 | 0 | | | | 0 | | | | 1 | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Initial Settings for **I2C\_CON** to bring out of Reset | | | | | | | | | | | | | | | | | | | | | |
| **Field** | **RESERVED** | **I2C\_EN** | **RESERVED** | **OPMODE** | | **STB** | **MST** | **TRX** | **XSA** | **XOA0** | | **XOA1** | | **XOA2** | | **XOA3** | | **RESERVED** | | **STP** | **STT** |
| **Bits** | **31-16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | | **7** | | **6** | | **5** | | **4** | **3** | **2** | **1** | **0** |
| **Write**  **Binary** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 0 | | 0 | | 0 | | 0 | 0 | 0 | 0 | 0 |
| **Write**  **Hex** | 0 | 8 | | | | 0 | | | | | 0 | | | | | | | 0 | | | |

* **Initialization Registers:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Write value to **I2C\_CON** for Master Transmitter Mode | | | | | | | | | | | | | | | | | | | | | |
| **Field** | **RESERVED** | **I2C\_EN** | **RESERVED** | **OPMODE** | | **STB** | **MST** | **TRX** | **XSA** | **XOA0** | | **XOA1** | | **XOA2** | | **XOA3** | | **RESERVED** | | **STP** | **STT** |
| **Bits** | **31-16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | | **7** | | **6** | | **5** | | **4** | **3** | **2** | **1** | **0** |
| **Write**  **Binary** | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | | 0 | | 0 | | 0 | | 0 | 0 | 0 | 0 | 0 |
| **Write**  **Hex** | 0 | 8 | | | | 6 | | | | | 0 | | | | | | | 0 | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Initialization Settings for **I2C\_IRQENABLE\_SET** Register (All disabled if POLLING) | | | | | | | | | | | | | | | | | |
| **Field** | **RESERVED** | **RESERVED** | **XDR\_IE** | **RDR\_IE** | **RESERVED** | **ROVR** | **XUDF** | **AAS\_IE** | **BF\_IE** | **AERR\_IE** | **STC\_IE** | **GC\_IE** | **XRDY\_IE** | **RRDY\_IE** | **ARDY\_IE** | **NACK\_IE** | **AL\_IE** |
| **Bits** | **31-16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Write**  **Binary** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Write**  **Hex** | 0 | 0 | | | | 0 | | | | 0 | | | | 0 | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Initialize Values for **I2C\_BUF** Register (NO DMA) | | | | | | | | | | | | | | | | | |
| **Field** | **RESERVED** | **RDMA\_EN** | **RXFIFO\_CLR** | **RXTRSH** | | | | | | **XDMA\_EN** | **TXFIFO\_CLR** | **TXTRSH** | | | | | |
| **Bits** | **31-16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Write Decimal** |  | | | 0 | | | | | |  | | 0 | | | | | |
| **Write**  **Binary** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Write**  **Hex** | 0 | 0 | | | | 0 | | | | 0 | | | | 0 | | | |

* **Pre-Transmission Registers:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **I2C\_SA** Values for Slave Address | | | | | | | | | | | | | |
| **Field** | **RESERVED** | **RESERVED** | **RESERVED** | **SA** | | | | | | | | | |
| **Bits** | **31-12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Write**  **Binary** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| **Write**  **Hex** | 0 | 0 | | | | 3 | | | | C | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Initialization Settings for **I2C\_CNT** Register (Depends on task) | | | | | | | | | | | | | | | | | |
| **Field** | **RESERVED** | **DCOUNT** | | | | | | | | | | | | | | | |
| **Bits** | **31-16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Decimal**  **Write** | **0** | **To be set before each transmission (dependent)** | | | | | | | | | | | | | | | |
| **Write**  **Binary** | 0 | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| **Write**  **Hex** | 0 | ? | | | | ? | | | | ? | | | | ? | | | |

* **Initiate Transmission Registers:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Read Mask Value to read **BB** from **I2C\_IRQSTATUS\_RAW** Register (Doesn’t matter if interrupts are enabled) | | | | | | | | | | | | | | | | | |
| **Field** | **RESERVED** | **RESERVED** | **XDR** | **RDR** | **BB** | **ROVR** | **XUDF** | **AAS** | **BF** | **AERR** | **STC** | **GC** | **XRDY** | **RRDY** | **ARDY** | **NACK** | **AL** |
| **Bits** | **31-16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Read**  **Binary** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Read**  **Hex** | 0 | 1 | | | | 0 | | | | 0 | | | | 0 | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Start/Stop Condition write on **I2C\_CON** to initiate transfer | | | | | | | | | | | | | | | | | | | | | | | |
| **Field** | **RESERVED** | **I2C\_EN** | **RESERVED** | **OPMODE** | | **STB** | **MST** | **TRX** | **XSA** | **XOA0** | | **XOA1** | | **XOA2** | | **XOA3** | | **RESERVED** | | | **STP** | | **STT** |
| **Bits** | **31-16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | | **7** | | **6** | | **5** | | **4** | | **3** | **2** | | **1** | **0** |
| **Write**  **Binary** | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | | 0 | | 0 | | 0 | | 0 | | 0 | 0 | | 1 | 1 |
| **Write**  **Hex** | 0 | 8 | | | | 6 | | | | | 0 | | | | | | | | 3 | | | | |

* **Receive Data Registers:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Read Mask Value to read RRDY from **I2C\_IRQSTATUS\_RAW** Register (Doesn’t matter if interrupts are enabled) | | | | | | | | | | | | | | | | | |
| **Field** | **RESERVED** | **RESERVED** | **XDR** | **RDR** | **BB** | **ROVR** | **XUDF** | **AAS** | **BF** | **AERR** | **STC** | **GC** | **XRDY** | **RRDY** | **ARDY** | **NACK** | **AL** |
| **Bits** | **31-16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Read**  **Binary** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| **Read**  **Hex** | 0 | 0 | | | | 0 | | | | 0 | | | | 8 | | | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Read from **I2C\_DATA** Access Register | | | | | | | | | |
| **Field** | **RESERVED** | **DATA** | | | | | | | |
| **Bits** | **31-8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Read**  **Binary** | 0 | ? | ? | ? | ? | ? | ? | ? | ? |
| **Read**  **Hex** | 0 | ? | | | | ? | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Clear RRDY flag using **I2C\_IRQSTATUS** Register | | | | | | | | | | | | | | | | | |
| **Field** | **RESERVED** | **RESERVED** | **XDR** | **RDR** | **BB** | **ROVR** | **XUDF** | **AAS** | **BF** | **AERR** | **STC** | **GC** | **XRDY** | **RRDY** | **ARDY** | **NACK** | **AL** |
| **Bits** | **31-16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Write**  **Binary** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| **Write**  **Hex** | 0 | 0 | | | | 0 | | | | 0 | | | | 8 | | | |

* **Transmit Data Registers:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Read Mask Value to read **XRDY** from **I2C\_IRQSTATUS\_RAW** Register (Doesn’t matter if interrupts are enabled) | | | | | | | | | | | | | | | | | |
| **Field** | **RESERVED** | **RESERVED** | **XDR** | **RDR** | **BB** | **ROVR** | **XUDF** | **AAS** | **BF** | **AERR** | **STC** | **GC** | **XRDY** | **RRDY** | **ARDY** | **NACK** | **AL** |
| **Bits** | **31-16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Read**  **Binary** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **Read**  **Hex** | 0 | 0 | | | | 0 | | | | 1 | | | | 0 | | | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Write to **I2C\_DATA** Access Register | | | | | | | | | |
| **Field** | **RESERVED** | **DATA** | | | | | | | |
| **Bits** | **31-8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Write**  **Binary** | 0 | ? | ? | ? | ? | ? | ? | ? | ? |
| **Write**  **Hex** | 0 | ? | | | | ? | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Clear **XRDY** flag using **I2C\_IRQSTATUS** Register | | | | | | | | | | | | | | | | | |
| **Field** | **RESERVED** | **RESERVED** | **XDR** | **RDR** | **BB** | **ROVR** | **XUDF** | **AAS** | **BF** | **AERR** | **STC** | **GC** | **XRDY** | **RRDY** | **ARDY** | **NACK** | **AL** |
| **Bits** | **31-16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| **Write**  **Binary** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| **Write**  **Hex** | 0 | 0 | | | | 0 | | | | 1 | | | | 0 | | | |

**Registers for initialization**

* Module Configuration

**Offset Control Module Register Value** **Description**

* + 0x95Cconf\_spi0\_cs0 0x2Write for **MODE2** w/o pullup
  + 0x958conf\_spi0\_d1 0x2 Write for **MODE2** w/o pullup

**Offset Clock Module Peripheral Register Write** **Description**

* + 0x48 CM\_PER\_I2C1\_CLKCTRL 0x2 Write to Enable Peripheral

**Offset I2C1 Register Value Description**

* + 0xB0 I2C\_PSC (Clock Prescalar Register) 0x03 Write for **ICLK** of 12MHz
  + 0xB4 I2C\_SCLL (SCL Low Time Register) 0x35 Write for **tLOW** to get 100kbps
  + 0xB8 I2C\_SCLH (SCL High Time Register) 0x37 Write for **tHIGH** to get 100kbps
  + 0xA8 I2C\_OA (Own Address Register) 0x001 Write to configure Own Address
  + 0xA4 I2C\_CON (Configuration Register) 0x8000 Write to enable I2C1 module
* Module Initialization

**Offset I2C1 Register Value Description**

* + 0xA4 I2C\_CON (Configuration Register) 0x600 Mode = Master, Transmitter
  + 0x2C I2C\_IRQENABLE\_SET (Interrupt Enable) 0x0000 Polling, No Interrupts (Part 1)
  + 0x94 I2C\_BUF (Buffer Configuration Register) 0x0000 Txt and Rx Thresholds.
* Pre-Transmission Initialization

**Offset I2C1 Register Value Description**

* + 0xAC I2C\_SA (Slave Address Register) 0x3C Newhaven Display = 0x78
  + 0x98 I2C\_CNT (Data Count Register) 0x9 9 bits to initialize

**Registers for transmission start**

* Transmission

**Offset I2C1 Register Value Description**

* + 0x24 I2C\_IRQSTATUS\_RAW (I2C Status Raw) 0x1000 Read **BB**, see if Bus is free
  + 0xA4 I2C\_CON (Configuration Register) 0x3 Start/Stop Condition begin

**Registers for byte transfer when controller is ready**

* Receive Data

**Offset I2C1 Register Value Description**

* + 0x24 I2C\_IRQSTATUS\_RAW (I2C Status Raw) 0x00000008 Read **RRDY** see if data is ready
  + 0x9C I2C\_DATA (Data Access Register) ? Read Received Data Byte
  + 0x28 I2C\_IRQSTATUS (Status Register) 0x00000008 If **RRDY** 1, clear **RRDY**
* Transmit Data

**Offset I2C1 Register Value Description**

* + 0x24 I2C\_IRQSTATUS\_RAW (I2C Status Raw) 0x00000010 Read **XRDY** see if data needed
  + 0x9C I2C\_DATA (Data Access Register) ? Write Data Byte
  + 0x28 I2C\_IRQSTATUS (Status Register) 0x0010 If **XRDY** 1, clear **XRDY**

**Construct the High-level algorithm for initialization and transmission start (should be the same as the chapter 7 example, minus the slave read**)

**High Level:**

**Initialization**.

* Configure pin connections on P9 connector for enable **SCL** and **SDA** in MODE2
* enable I2C1 Clock.
* Get ICLK of 12 MHz
* get 100kbps SCL/SDA signals
* Configure Own Address
* Enable I2C1 module
* Configure mode. Mode = Master, Transmitter
* Polling, no Interrupts, Part 1.
* set Transmit and Receive threshold .

**Transmission sequence.**

* Set slave address value
* Set number of transmission Bytes
* Poll the **BB** flag bit to check bus status.
* Queue Start/Stop Condition. Activities = S-A-D(n)-P.
* See if data can be written to **I2C\_DATA**.

**WHILE**:

* If **XRDY** is 1 FIFO is ready for data. Write to **I2C\_DATA** (Data Access Register).
  + Transmit Byte.
* If 1, Read-Modify-Write **I2C\_IRQSTATUS** (Status Register) to clear **XRDY.**

**TO SEND CHARACTERS:**

* Poll the **BB** flag bit to check bus status.
* Queue Start/Stop Condition. Activities = S-A-D(n)-P.
* See if data can be written to **I2C\_DATA**.

**WHILE**:

* If **XRDY** is 1 FIFO is ready for data. Write to **I2C\_DATA** (Data Access Register).
  + Transmit Byte.
* If 1, Read-Modify-Write **I2C\_IRQSTATUS** (Status Register) to clear **XRDY.**

**Construct the Low-level algorithm initialization and transmission start.**

* Write 0x2 to **conf\_spi0\_cs0** offset 0x95C to enable (**SCL**) for MODE2 w/o pullup
* Write 0x2 to **conf\_spi0\_d1** offset 0x958 to enable (SDA) for MODE2 w/o pullup
* Write 0x2 to **CM\_PER\_I2C1\_CLKCTRL** offset 0x48 to enable I2C1 Clock.
* Write 0x03 to **I2C\_PSC** (Clock Prescalar Register) offset 0xB0 for ICLK of 12 MHz
* Write 0x35 to **I2C\_SCLL** (SCL Low Time Register) offset 0xB4 for **tLOW** to get 100kbps (5us-Low)
* Write 0x37 to **I2C\_SCLH** (SCL High Time Register) offset 0xB8 for **tHIGH** to get 100kbps (5us-High)
* Write 0x001 to **I2C\_OA** (Own Address Register) offset 0xA8 to configure Own Address
* Write 0x8000 to **I2C\_CON** (Configuration Register) offset 0xA4 to take out of reset, enable I2C1 module
* Read-Modify-Write 0x600 to **I2C\_CON** (Configuration Register)offset 0xA4 to configure mode.
* Write 0x0000 to **I2C\_IRQENABLE\_SET** (Interrupt Enable Set Register) offset 0x2C to enable Polling
* Write 0x0000 to **I2C\_BUF** (Buffer Configuration Register) offset 0x94 to set Transmit and Receive .
* Write 0x3C to **I2C\_SA** (Slave Address Register) offset 0xAC Slave address value
* Write 0x9 to **I2C\_CNT** (Data Count Register) offset 0x98 to set number of transmission Bytes
* Read mask 0x00001000 from **I2C\_IRQSTATUS\_RAW** (I2C Status Raw Register) offset 0x24 to check bus status.
* Read-Modify-Write 0x00000003 to **I2C\_CON** (Configuration Register) offset 0xA4 to queue Start/Stop Cond.

**While DCOUNT >0:**

* + Read mask 0x00000010 from **I2C\_IRQSTATUS\_RAW** (I2C Status Raw Register) offset 0x24 to see if write ready
  + If **XRDY** is 1 FIFO is ready for data. Write to **I2C\_DATA** (Data Access Register) offset 0x9C .
  + If 1, Write 0x00000010 to **I2C\_IRQSTATUS** (Status Register) offset 0x28 to clear **XRDY.**

**To Send Characters:**

* Write 0x9 to **I2C\_CNT** (Data Count Register) offset 0x98 to set number of transmission Bytes
* Read mask 0x00001000 from **I2C\_IRQSTATUS\_RAW** (I2C Status Raw Register) offset 0x24 to check bus status.
* Read-Modify-Write 0x00000003 to **I2C\_CON** (Configuration Register) offset 0xA4 to queue Start/Stop Cond.

**While DCOUNT >0:**

* + Read mask 0x00000010 from **I2C\_IRQSTATUS\_RAW** (I2C Status Raw Register) offset 0x24 to see if write ready
  + If **XRDY** is 1 FIFO is ready for data. Write to **I2C\_DATA** (Data Access Register) offset 0x9C .
  + If 1, Write 0x00000010 to **I2C\_IRQSTATUS** (Status Register) offset 0x28 to clear **XRDY.**

**Study the device manual to determine how to get it in the desired mode.**

* **This section took quite a while to analyze the instruction scheme used by the Dot Matrix Controller and New haven display module. I used the example code provided by Newhaven to double check the Instruction sequence for initialization as compared to the instructions provided by Sitronix.**

**Determine Capabilities**.

(when EXT option pin connect to VSS, the instruction set follow below table)

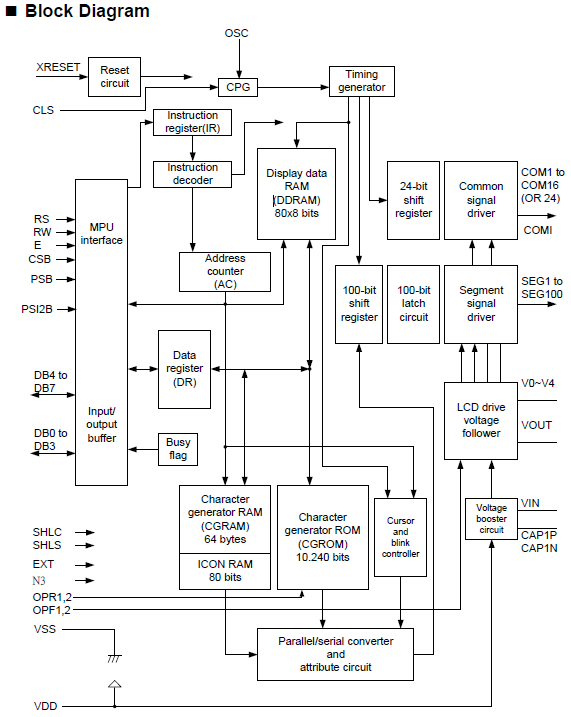
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | | | | | | | | | **Description** | **n Time** | | |
|  |  |  |  |  |  |  |  | **DB1** | **DB0** | **OSC=**  **380kHz** | **OSC=**  **540kHz** | **OSC=**  **700kHz** |
| Clear Display | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Write 20H to DDRAM. and set DDRAM address to 00H from AC | **1.08**  **ms** | **0.76**  **ms** | **0.59**  **ms** |
| Return Home | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | x | Set DDRAM address to 00H from AC and return cursor to its original position if shifted. The contents of DDRAM are not changed. | **1.08**  **ms** | **0.76**  **ms** | **0.59**  **ms** |
| Entry Mode Set | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | I/D | S | Sets cursor move direction and specifies display shift. These operations are performed during data write and read. | **26.3** µ**s** | **18.5** µ**s** | **14.3** µ**s** |
| Display ON/OFF | 0 | 0 | 0 | 0 | 0 | 0 | 1 | D | C | B | D=1:entire display on C=1:cursor on B=1:cursor position on | **26.3** µ**s** | **18.5** µ**s** | **14.3** µ**s** |
| Function Set | 0 | 0 | 0 | 0 | 1 | DL | N | DH | IS2 | IS1 | DL: interface data is 8/4 bits N: number of line is 2/1  DH: double height font  IS[2:1]: instruction table select | **26.3** µ**s** | **18.5** µ**s** | **14.3** µ**s** |
| Set DDRAM  Address | 0 | 0 | 1 | AC6 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | Set DDRAM address in address counter | **26.3** µ**s** | **18.5** µ**s** | **14.3** µ**s** |
| Read Busy Flag and Address | 0 | 1 | BF | AC6 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | Whether during internal operation or not can be known by reading BF. The contents of address counter can also be read. | **0** | **0** | **0** |
| Write Data to RAM | 1 | 0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | Write data into internal RAM (DDRAM/CGRAM/ICONRAM) | **26.3** µ**s** | **18.5** µ**s** | **14.3** µ**s** |
| Read Data from RAM | 1 | 1 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | Read data from internal RAM (DDRAM/CGRAM/ICONRAM) | **26.3** µ**s** | **18.5** µ**s** | **14.3** µ**s** |

**Instruction table 0(IS[2:1]=[0,0])**

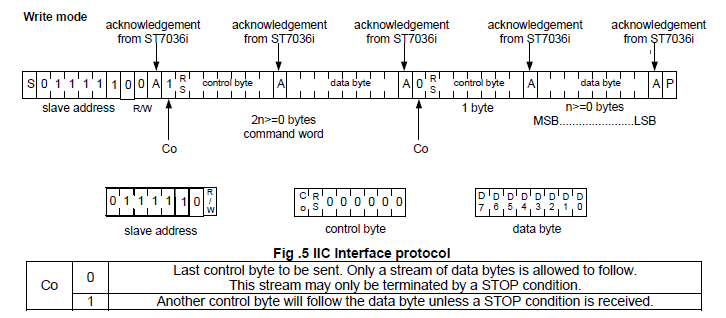
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cursor or Display Shift | 0 | 0 | 0 | 0 | 0 | 1 | S/C | R/L | X | X | S/C and R/L:  Set cursor moving and display shift control bit, and the direction, without changing DDRAM data. | **26.3** µ**s** | **18.5** µ**s** | **14.3** µ**s** |
| Set CGRAM | 0 | 0 | 0 | 1 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | Set CGRAM address in address counter | **26.3** µ**s** | **18.5** µ**s** | **14.3** µ**s** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction table 1(IS[2:1]=[0,1])** | | | | | | | | | | | | | | |
| Bias Set | 0 | 0 | 0 | 0 | 0 | 1 | BS | 1 | 0 | FX | BS=1:1/4 bias BS=0:1/5 bias  FX: fixed on high in 3-line application and fixed on low in other  applications. | **26.3** µ**s** | **18.5** µ**s** | **14.3** µ**s** |
| Set ICON  Address | 0 | 0 | 0 | 1 | 0 | 0 | AC3 | AC2 | AC1 | AC0 | Set ICON address in address counter. | **26.3** µ**s** | **18.5** µ**s** | **14.3** µ**s** |
| Power/ICON Control/ Contrast Set | 0 | 0 | 0 | 1 | 0 | 1 | Ion | Bon | C5 | C4 | Ion: ICON display on/off  Bon: set booster circuit on/off C5,C4: Contrast set for internal follower mode. | **26.3** µ**s** | **18.5** µ**s** | **14.3** µ**s** |
| Follower Control | 0 | 0 | 0 | 1 | 1 | 0 | Fon | Rab 2 | Rab 1 | Rab 0 | Fon: set follower circuit on/off Rab2~0:  select follower amplified ratio. | **26.3** µ**s** | **18.5** µ**s** | **14.3** µ**s** |
| Contrast Set | 0 | 0 | 0 | 1 | 1 | 1 | C3 | C2 | C1 | C0 | Contrast set for internal follower mode. | **26.3** µ**s** | **18.5** µ**s** | **14.3** µ**s** |

**Determine Internal Registers**



**Determine address mechanism.**



* There are 3 bits that control the flow of data control to the slave device.
  + RS:
    - Select Registers
    - 0 for Instruction register (Write) or Busy flag & address counter (on read),
    - 1 for Data register (write and read)
  + R/W
    - Select read or write
    - 0 for Write
    - 1 for Read
  + Co
    - Control bit
    - 0 for last control byte to be sent, followed only by a stream of data bytes, followed by stop condition.
    - 1 another control byte will follow the following data byte unless a stop condition is received.

**Determine maximum SCL clock frequency.**

* **SCL** maximum clock frequency: 300KHz @ 2.7 - 4.5V
* **SCL** minimum high period (**tHIGH**): 0.6s
* **SCL** minimum low period (**tLOW**): 2.5s

**Determine how you send characters to display.**

**Make a list of desired initialization words.**

* #define CTRLMOD\_BASE 0x44E10000
* #define CM\_PER\_BASE 0x44E00000
* #define **I2C\_BASE 0x4802A000**
* #define **SLAVE\_ADDR** 0x3C
* #define **WRITE\_TO** 0x00
* #define **FUNCTION\_SET0** 0x38
* #define **FUNCTION\_SET1** 0x39
* #define **BIAS\_SET** 0x14
* #define **CONTRAST\_SET** 0x78
* #define **PWR\_ICON\_CON\_SET** 0x5E
* #define **FOLLOWER\_SET\_ON** 0x6D
* #define **DISPLAY\_SET\_ON** 0x0C
* #define **CLR**\_**DISPLAY** 0x01
* #define **ENTRY\_MODE** 0x06
* #define **CGRAM\_SET** 0x41
* #define **SET\_DDRAM** 0b10000001

**Make a list of words needed to display your name**.

* #define **WRITE\_TO**0x00
* #define **CLR\_DISPLAY** 0x01
* #define **ENTRY\_MODE**  0x06
* #define **CGRAM\_SET** 0x40

**Either single height on two lines, or double height display mode**.

* **N** = 1 for double lines.

**Complete the High-level and low-level algorithms with these steps included.**

**High Level:**

**Initialization**.

* Configure pin connections on P9 connector for enable **SCL** and **SDA** in MODE2
* enable I2C1 Clock.
* Get ICLK of 12 MHz
* get 100kbps SCL/SDA signals
* Configure Own Address
* Enable I2C1 module
* Configure mode. Mode = Master, Transmitter
* Polling, no Interrupts, Part 1.
* set Transmit and Receive threshold .

**Transmission sequence.**

* Set slave address value
* Set number of transmission Bytes
* Poll the **BB** flag bit to check bus status.
* Queue Start/Stop Condition. Activities = S-A-D(n)-P.
* See if data can be written to **I2C\_DATA**.

**WHILE**:

* If **XRDY** is 1 FIFO is ready for data. Write to **I2C\_DATA** (Data Access Register).
  + Transmit Byte.
* If 1, Read-Modify-Write **I2C\_IRQSTATUS** (Status Register) to clear **XRDY.**

**TO SEND CHARACTERS:**

* Poll the **BB** flag bit to check bus status.
* Queue Start/Stop Condition. Activities = S-A-D(n)-P.
* See if data can be written to **I2C\_DATA**.

**WHILE**:

* If **XRDY** is 1 FIFO is ready for data. Write to **I2C\_DATA** (Data Access Register).
  + Transmit Byte.
* If 1, Read-Modify-Write **I2C\_IRQSTATUS** (Status Register) to clear **XRDY.**

**Construct the Low-level algorithm initialization and transmission start.**

* Write 0x2 to **conf\_spi0\_cs0** offset 0x95C to enable (**SCL**) for MODE2 w/o pullup
* Write 0x2 to **conf\_spi0\_d1** offset 0x958 to enable (SDA) for MODE2 w/o pullup
* Write 0x2 to **CM\_PER\_I2C1\_CLKCTRL** offset 0x48 to enable I2C1 Clock.
* Write 0x03 to **I2C\_PSC** (Clock Prescalar Register) offset 0xB0 for ICLK of 12 MHz
* Write 0x35 to **I2C\_SCLL** (SCL Low Time Register) offset 0xB4 for **tLOW** to get 100kbps (5us-Low)
* Write 0x37 to **I2C\_SCLH** (SCL High Time Register) offset 0xB8 for **tHIGH** to get 100kbps (5us-High)
* Write 0x001 to **I2C\_OA** (Own Address Register) offset 0xA8 to configure Own Address
* Write 0x8000 to **I2C\_CON** (Configuration Register) offset 0xA4 to take out of reset, enable I2C1 module
* Read-Modify-Write 0x600 to **I2C\_CON** (Configuration Register)offset 0xA4 to configure mode.
* Write 0x0010 to **I2C\_IRQENABLE\_SET** (Interrupt Enable Set Register) offset 0x2C to enable Polling
* Write 0x0000 to **I2C\_BUF** (Buffer Configuration Register) offset 0x94 to set Transmit and Receive .
* Write 0x3C to **I2C\_SA** (Slave Address Register) offset 0xAC Slave address value
* Write 0x9 to **I2C\_CNT** (Data Count Register) offset 0x98 to set number of transmission Bytes
* Read mask 0x00001000 from **I2C\_IRQSTATUS\_RAW** (I2C Status Raw Register) offset 0x24 to check bus status.
* Read-Modify-Write 0x00000003 to **I2C\_CON** (Configuration Register) offset 0xA4 to queue Start/Stop Cond.

**While DCOUNT >0:**

* + Read mask 0x00000010 from **I2C\_IRQSTATUS\_RAW** (I2C Status Raw Register) offset 0x24 to see if write ready
  + If **XRDY** is 1 FIFO is ready for data. Write to **I2C\_DATA** (Data Access Register) offset 0x9C .
    - Switch ( NUM\_DBYTES – DCOUNT)
      * WRITE\_TO 0x00
      * FUNCTION\_SET0 0x38
      * FUNCTION\_SET1 0x39
      * BIAS\_SET 0x14
      * CONTRAST\_SET 0b01110101
      * PWR\_ICON\_CON\_SET 0x5E
      * FOLLOWER\_SET\_ON 0x6D
      * DISPLAY\_SET\_ON 0x0C
      * CLR\_DISPLAY 0x01
      * ENTRY\_MODE 0x06
  + If 1, Write 0x00000010 to **I2C\_IRQSTATUS** (Status Register) offset 0x28 to clear **XRDY.**

**To Send Characters:**

* Write 0xD to **I2C\_CNT** (Data Count Register) offset 0x98 to set number of transmission Bytes
* Read mask 0x00001000 from **I2C\_IRQSTATUS\_RAW** (I2C Status Raw Register) offset 0x24 to check bus status.
* Read-Modify-Write 0x00000003 to **I2C\_CON** (Configuration Register) offset 0xA4 to queue Start/Stop Cond.

**While DCOUNT >0:**

* + Read mask 0x00000010 from **I2C\_IRQSTATUS\_RAW** (I2C Status Raw Register) offset 0x24 to see if write ready
  + If **XRDY** is 1 FIFO is ready for data. Write to **I2C\_DATA** (Data Access Register) offset 0x9C .
    - Switch ( NUM\_DBYTES – DCOUNT)
      * WRITE\_TO\_WCO 0x80
      * SET\_DDRAM 0b10000001
      * WRITE\_TO\_RAM 0x40
      * ‘M’
      * ‘I’
      * ‘k’
      * ‘e’
      * ‘ ‘
      * ‘E’
      * ‘s’
      * ‘c’
      * ‘u’
      * ‘e’
  + If 1, Write 0x00000010 to **I2C\_IRQSTATUS** (Status Register) offset 0x28 to clear **XRDY.**

**Create a polled version for handshaking of the program.**

* See attached.
* A bizarre error where a space after my offset value in the HWREG macro caused a compiler error because it is an illegal character for the operation. Removing the space after the offset value allowed compilation.
* Had a hell of a time getting the communications to work because the wires were screwed up on the device I was working on. Thankfully I decided to check the hardware first, because I was pretty certain my algorithm was correct.

**Receive TA/Instructor sign-off.**

**Modify Algorithm to implement the handshaking on an interrupt basis with the interrupt controller.**

* Write 0x2 to **conf\_spi0\_cs0** offset 0x95C to enable (**SCL**) for MODE2 w/o pullup
* Write 0x2 to **conf\_spi0\_d1** offset 0x958 to enable (SDA) for MODE2 w/o pullup
* Write 0x2 to **CM\_PER\_I2C1\_CLKCTRL** offset 0x48 to enable I2C1 Clock.
* Write 0x03 to **I2C\_PSC** (Clock Prescalar Register) offset 0xB0 for ICLK of 12 MHz
* Write 0x35 to **I2C\_SCLL** (SCL Low Time Register) offset 0xB4 for **tLOW** to get 100kbps (5us-Low)
* Write 0x37 to **I2C\_SCLH** (SCL High Time Register) offset 0xB8 for **tHIGH** to get 100kbps (5us-High)
* Write 0x001 to **I2C\_OA** (Own Address Register) offset 0xA8 to configure Own Address
* Write 0x8000 to **I2C\_CON** (Configuration Register) offset 0xA4 to take out of reset, enable I2C1 module
* Read-Modify-Write 0x600 to **I2C\_CON** (Configuration Register)offset 0xA4 to configure mode.
* Write 0x0010 to **I2C\_IRQENABLE\_SET** (Interrupt Enable Set Register) offset 0x2C to enable Polling
* Write 0x0000 to **I2C\_BUF** (Buffer Configuration Register) offset 0x94 to set Transmit and Receive .
* Write 0x3C to **I2C\_SA** (Slave Address Register) offset 0xAC Slave address value

**Initiate Transfer**:

* Write to **I2C\_CNT** (Data Count Register) to set number of transmission Bytes
* Read **I2C\_IRQSTATUS\_RAW** (I2C Status Raw Register) **BB** flag bit to check bus status.
* Read-Modify-Write **I2C\_CON** (Configuration Register) to queue Start/Stop Condition. Activities = S-A-D(n)-P.

**Wait:**

Do nothing.

**Interrupt Handler:**

* Check **INTC\_PENDING\_IRQ2**  of **Interrupt Controller** offset 0xD8, mask value 0x80

If interrupt is from **I2C1:**

* Check individual **XRDY** IRQ Status from **IRQ\_STATUS\_RAW** I2C offset 0x24, bit mask value 0x10
* Clear interrupt enable of **XRDY** on I2C offset 0x30, write-to-clear value 0x10

If entering initialization segment of display:

* Run display initialization block.
* Update current byte to transmit by subtracting **DCOUNT** at I2C\_CNT register offset 0x98 from the # of Bytes to transfer (10) :
  + Upon each entry a switch is used to write to **I2C\_DATA** in order;
    - Switch ( NUM\_DBYTES – DCOUNT)
    - WRITE\_TO 0x00
    - FUNCTION\_SET0 0x38
    - FUNCTION\_SET1 0x39
    - BIAS\_SET 0x14
    - CONTRAST\_SET 0b01110101
    - PWR\_ICON\_CON\_SET 0x5E
    - FOLLOWER\_SET\_ON 0x6D
    - DISPLAY\_SET\_ON 0x0C
    - CLR\_DISPLAY 0x01
    - ENTRY\_MODE 0x06
      * Set a conditional variable “**isdisplayinit**” for switching to next operation on next interrupt.
    - Read-Modify-Write **I2C\_IRQSTATUS** (Status Register) to clear **XRDY** after each write.
  + **XRDY** interrupt is enabled again in **I2C\_IRQENABLE\_SET** register, offset 0x2C, write value 0x10.

**Wait:**

Back to wait while waiting for FIFO to unload.

**Interrupt Handler:**

* Check **INTC\_PENDING\_IRQ2**  of **Interrupt Controller** offset 0xD8, mask value 0x80

If interrupt is from **I2C1:**

* Check individual **XRDY** IRQ Status from **IRQ\_STATUS\_RAW** I2C offset 0x24, bit mask value 0x10
* Clear interrupt enable of **XRDY** on I2C offset 0x30, write-to-clear value 0x10

If this is the second entry.:

* “**isdisplayinit**” moves process flow in handler to next initialization routine in preparation to send the characters.
* **INTC\_NEW\_IRQ** is written to at offset 0x48 of the **Interrupt Control** module, with value 0x1.
* Read-Modify-Write 0x8600 to **I2C\_CON** (Configuration Register)offset 0xA4 to configure mode.
* Write 0x3C to **I2C\_SA** (Slave Address Register) offset 0xAC Slave address value
* Write to **I2C\_CNT** (Data Count Register) to set number of transmission Bytes
* Read **I2C\_IRQSTATUS\_RAW** (I2C Status Raw Register) **BB** flag bit to check bus status.
* Modify another conditional bit “**issendinit**” to allow final flow control to send characters in interrupt handler.
* Write 0x0010 to **I2C\_IRQENABLE\_SET** (Interrupt Enable Set Register) offset 0x2C to enable next interrupt.
* Read-Modify-Write **I2C\_CON** (Configuration Register) to queue Start/Stop Condition. Activities = S-A-D(n)-P

**Wait:**

* Back to wait while waiting for FIFO to unload.

**Interrupt Handler:**

* Check **INTC\_PENDING\_IRQ2**  of **Interrupt Controller** offset 0xD8, mask value 0x80

If interrupt is from **I2C1:**

* Check individual **XRDY** IRQ Status from **IRQ\_STATUS\_RAW** I2C offset 0x24, bit mask value 0x10
* Clear interrupt enable of **XRDY** on I2C offset 0x30, write-to-clear value 0x10

Move into character send routine.

* Update current byte to transmit by subtracting **DCOUNT** at I2C\_CNT register offset 0x98 from the # of Bytes to transfer (10) :
  + Upon each entry a switch is used to write to **I2C\_DATA** in order;
    - Switch ( NUM\_DBYTES – DCOUNT)
      * WRITE\_TO\_WCO 0x80
      * SET\_DDRAM 0b10000001
      * WRITE\_TO\_RAM 0x40
      * ‘M’
      * ‘I’
      * ‘k’
      * ‘e’
      * ‘ ‘
      * ‘E’
      * ‘s’
      * ‘c’
      * ‘u’
      * ‘e’
    - Read-Modify-Write **I2C\_IRQSTATUS** (Status Register) to clear **XRDY** after each write.
  + **XRDY** interrupt is enabled again in **I2C\_IRQENABLE\_SET** register, offset 0x2C, write value 0x10.
* End of program.

**Modify Program to implement handshaking on an interrupt basis**.

* See attached.

**Receive TA/Instructor Sign-off.**

**Find how to make the display rotate around the screen to the right in a loop. Or blink on and off.**

****

**POLLED VERSION:**

/\*\* ECE 372 Project 2

\* Utilizes I2C1 to communicate with display board.

\* Polling Version

\* Author: Michael Escue

\*/

// Define Indirect Addressing Macro for Registers

#define HWREG(x) (\*((volatile unsigned int \*)(x)))

// Common Defines

#define TRUE 1

#define FALSE 0

#define DELAY\_COUNT 100000

// Base Module Defines

#define CTRLMOD\_BASE 0x44E10000

#define CM\_PER\_BASE 0x44E00000

#define I2C1\_BASE 0x4802A000

#define DEBUGSS\_DRM\_BASE 0x4B160000

// Control Module Defines

#define CONF\_SPI0\_CS0\_SCL 0x95C

#define CONF\_SPI0\_D1\_SDA 0x958

#define MODE2 0x22 //0x22 seemed to be working.

//#define MODE2\_NOREC 0x2 BOth pins must be receivers.

// Peripheral Control Module Defines

#define CM\_PER\_I2C1\_CLKCTRL 0x48

#define CLK\_ENABLE 0x2

// DRM Register Offset Defines

#define I2C\_1\_SUSPEND\_CTRL 0x22C

// Register Address Offset Defines

#define I2C\_SA 0xAC

#define I2C\_CNT 0x98

#define I2C\_DATA 0x9C

#define I2C\_IRQSTATUS\_RAW 0x24

#define I2C\_CON 0xA4

#define I2C\_PSC 0xB0

#define I2C\_SCLL 0xB4

#define I2C\_SCLH 0xB8

#define I2C\_OA 0xA8

#define I2C\_IRQENABLE\_SET 0x2C

#define I2C\_BUF 0x94

#define I2C\_BUFSTAT 0xC0

// I2C Register Values

#define \_12MHZ\_CLK 0x03

#define \_tLOW\_5MICROSEC 0x35

#define \_tHIGH\_5MICROSEC 0x37

#define OWNADDR 0x01

#define I2C1\_ENABLE 0x8000

#define IRQ\_DISABLED 0x0000

// Data Byte Defines

#define WRITE\_TO 0x00

#define WRITE\_TO\_RAM 0x40

#define WRITE\_TO\_WCO 0x80

#define FUNCTION\_SET0 0x38

#define FUNCTION\_SET1 0x39

#define BIAS\_SET 0x14

#define CONTRAST\_SET 0b01110101

#define PWR\_ICON\_CON\_SET 0x5E

#define FOLLOWER\_SET\_ON 0x6D

#define DISPLAY\_SET\_ON 0x0C

#define CLR\_DISPLAY 0x01

#define ENTRY\_MODE 0x06

#define CGRAM\_SET 0x41

#define SET\_DDRAM 0b10000001

// Mask Defines

#define DCOUNT\_VAL 0x0000FFFF

#define XRDY\_BIT 0x00000010

#define XRDY\_RDY 0x00000010

#define RRDY\_BIT 0x00000008

#define RRDY\_RDY 0x00000008

#define BF\_BIT 0x00001000

#define BUS\_IS\_FREE 0

#define TXTRSH\_VAL 0x0000003F

#define RXTRSH\_VAL 0x00003F00

#define AERR\_BIT 0x00000080

#define DATA\_VAL 0xFF

#define BUFSTAT\_VAL 0x0000003F

#define ARDY\_BIT 0x00000004

#define TXFIFO\_CLR\_BIT 0x00000040

//I2C Communication Defines

#define SLAVE\_ADDR 0b0111100

#define NUM\_OF\_DBYTES 10

#define START\_COND 0x00000001

#define STOP\_COND 0x00000002

#define MASTER\_TX\_MODE 0x600

#define NAME\_BYTE\_LENGTH 13

// Variables

unsigned int x;

unsigned int y;

volatile unsigned int USR\_STACK[100];

volatile unsigned int IRQ\_STACK[100];

void wait(void){

while(1){

// Endless loop

};

}

void delay(unsigned long int y){

while(y>0){

y--;

}

}

void stack\_init(void){

//SET UP STACKS

//init USR stack

asm("LDR R13, =USR\_STACK");

asm("ADD R13, R13, #0x1000");

//init IRQ stack

asm("CPS #0x12"); //Switch to IRQ mode

asm("LDR R13, =IRQ\_STACK");

asm("ADD R13, R13, #0x1000");

asm("CPS #0x13"); //Switch to User Mode

}

void irq\_enable(void){

asm("mrs r0, CPSR");

asm("bic r0, r0, #0x80");

asm("msr CPSR\_c, R0");

}

//Not used in this polling version

void int\_handler(void){

if(HWREG(0x482000D8) == 0x20000000)

{

}

asm("LDMFD SP!, {LR}");

asm("LDMFD SP!, {LR}");

asm("SUBS PC, LR, #0x4");

}

void set\_debug(void){

HWREG(DEBUGSS\_DRM\_BASE + I2C\_1\_SUSPEND\_CTRL) = 0x9;

}

int is\_bus\_free(void){

x = HWREG(I2C1\_BASE + I2C\_IRQSTATUS\_RAW); //"Read mask 0x00001000 from I2C\_IRQSTATUS\_RAW (I2C Status Raw Register) offset 0x24 to check bus status.

x = (x & BF\_BIT); //Mask.

if(x == BUS\_IS\_FREE) return 1;

else return 0;

}

int is\_i2c\_write\_ready(void){

x = HWREG(I2C1\_BASE + I2C\_IRQSTATUS\_RAW); //�Read mask 0x00000010 from I2C\_IRQSTATUS\_RAW (I2C Status Raw Register) offset 0x24 to see if write ready�

x = (x & XRDY\_BIT); //Mask.

if(x == XRDY\_RDY) return 1;

else return 0;

}

void clear\_i2c\_write\_ready(void){

HWREG(I2C1\_BASE + I2C\_IRQSTATUS\_RAW) = XRDY\_BIT;

}

int is\_i2c\_read\_ready(void){

x = HWREG(I2C1\_BASE + I2C\_IRQSTATUS\_RAW); //�Read mask 0x00000008 I2C\_IRQSTATUS\_RAW (I2C Status Raw Register) offset 0x24 see if data is ready.�

x = (x & RRDY\_BIT); //Mask.

if(x == RRDY\_RDY) return 1;

else return 0;

}

void startstop\_condition(void){

x = HWREG(I2C1\_BASE + I2C\_CON); //"Read-Modify-Write 0x3 to I2C\_CON (Configuration Register) offset 0xA4 to queue Start/Stop Condition.�

x = (x | START\_COND | STOP\_COND); //Mask.

HWREG(I2C1\_BASE + I2C\_CON) = x; //Write back.

}

void start\_condition(void){

x = HWREG(I2C1\_BASE + I2C\_CON); //"Read-Modify-Write 0x3 to I2C\_CON (Configuration Register) offset 0xA4 to queue Start/Stop Condition.�

x = (x | START\_COND); //Mask.

HWREG(I2C1\_BASE + I2C\_CON) = x; //Write back.

}

void stop\_condition(void){

x = HWREG(I2C1\_BASE + I2C\_CON); //"Read-Modify-Write 0x3 to I2C\_CON (Configuration Register) offset 0xA4 to queue Start/Stop Condition.�

x = (x | STOP\_COND); //Mask.

HWREG(I2C1\_BASE + I2C\_CON) = x; //Write back.

}

void config\_master\_transmitter(void){

x = HWREG(I2C1\_BASE + I2C\_CON); //"Read-Modify-Write 0xE00 to I2C\_CON (Configuration Register)offset 0xA4 to configure mode."

x = (x | MASTER\_TX\_MODE); //Mask.

HWREG(I2C1\_BASE + I2C\_CON) = x; //Write back.

}

void set\_buf\_txtrsh(unsigned int y){

y = (y & TXTRSH\_VAL);

x = HWREG(I2C1\_BASE + I2C\_BUF);

y = (y | x);

HWREG(I2C1\_BASE + I2C\_BUF) = y; //"Write 0x0000 to I2C\_BUF (Buffer Configuration Register) offset 0x94 to set Transmit and Receive .�

}

void set\_buf\_rxtrsh(unsigned int y){

y = ((y<<8) & RXTRSH\_VAL);

x = HWREG(I2C1\_BASE + I2C\_BUF);

y = (y | x);

HWREG(I2C1\_BASE + I2C\_BUF) = y; //"Write 0x0000 to I2C\_BUF (Buffer Configuration Register) offset 0x94 to set Transmit and Receive .�

}

void set\_num\_databytes(unsigned int y){

y = (y & DCOUNT\_VAL);

//Number of Data Bytes pre-transmission.

HWREG(I2C1\_BASE + I2C\_CNT) = y; //"Write 0x9 to I2C\_CNT (Data Count Register) offset 0x98 to set number of transmission Bytes"

}

void write\_to\_bus(unsigned char x){

x = (x & DATA\_VAL );

HWREG(I2C1\_BASE + I2C\_DATA) = x; //Write to data bus.

clear\_i2c\_write\_ready();

delay(2000);

}

void set\_slave\_addr(unsigned int x){

//Slave address pre-transmission.

HWREG(I2C1\_BASE + I2C\_SA) = x; //"Write 0x78 to I2C\_SA (Slave Address Register) offset 0xAC Slave address value"

}

void i2c\_init(void){

//P9 Connector settings.

HWREG(CTRLMOD\_BASE + CONF\_SPI0\_CS0\_SCL) = MODE2; //�Write 0x2 to conf\_spi0\_cs0 offset 0x95C to enable (SCL) for MODE2 w/o pullup�

HWREG(CTRLMOD\_BASE + CONF\_SPI0\_D1\_SDA) = MODE2; //�Write 0x2 to conf\_spi0\_d1 offset 0x958 to enable (SDA) for MODE2 w/o pullup�

//Enable Clock to I2C1.

HWREG(CM\_PER\_BASE + CM\_PER\_I2C1\_CLKCTRL) = CLK\_ENABLE; //�Write 0x2 to CM\_PER\_I2C1\_CLKCTRL offset 0x48 to enable I2C1 Clock.�

//Configure I2C1.

HWREG(I2C1\_BASE + I2C\_PSC) = \_12MHZ\_CLK; //�Write 0x03 to I2C\_PSC (Clock Prescalar Register) offset 0xB0 for ICLK of 12 MHz�

HWREG(I2C1\_BASE + I2C\_SCLL) = \_tLOW\_5MICROSEC; //"Write 0x35 to I2C\_SCLL (SCL Low Time Register) offset 0xB4 for tLOW to get 100kbps (5us-Low)"

HWREG(I2C1\_BASE + I2C\_SCLH) = \_tHIGH\_5MICROSEC; //"Write 0x37 to I2C\_SCLH (SCL High Time Register) offset 0xB8 for tHIGH to get 100kbps (5us-High)"

HWREG(I2C1\_BASE + I2C\_OA) = OWNADDR; //�Write 0x001 to I2C\_OA (Own Address Register) offset 0xA8 to configure Own Address�

HWREG(I2C1\_BASE + I2C\_CON) = I2C1\_ENABLE; //�Write 0x8000 to I2C\_CON (Configuration Register) offset 0xA4 to take out of reset, enable I2C1 module�

config\_master\_transmitter();

HWREG(I2C1\_BASE + I2C\_IRQENABLE\_SET) = IRQ\_DISABLED; //"Write 0x0000 to I2C\_IRQENABLE\_SET (Interrupt Enable Set Register) offset 0x2C to enable Polling�

set\_buf\_txtrsh(0);

set\_buf\_rxtrsh(0);

}

void init\_display(void){

unsigned int current\_DCOUNT;

set\_slave\_addr(SLAVE\_ADDR);

set\_num\_databytes(NUM\_OF\_DBYTES);

while(is\_bus\_free() != TRUE){

}

startstop\_condition();

while((HWREG(I2C1\_BASE + I2C\_BUFSTAT) & BUFSTAT\_VAL) > 0){

current\_DCOUNT = HWREG(I2C1\_BASE + I2C\_BUFSTAT) & BUFSTAT\_VAL;

if(is\_i2c\_write\_ready()){//If ready to write

switch(NUM\_OF\_DBYTES-current\_DCOUNT){

case 0:

write\_to\_bus(WRITE\_TO);

break;

case 1:

write\_to\_bus(FUNCTION\_SET0);

break;

case 2:

write\_to\_bus(FUNCTION\_SET1);

break;

case 3:

write\_to\_bus(BIAS\_SET);

break;

case 4:

write\_to\_bus(CONTRAST\_SET);

break;

case 5:

write\_to\_bus(PWR\_ICON\_CON\_SET);

break;

case 6:

write\_to\_bus(FOLLOWER\_SET\_ON);

break;

case 7:

write\_to\_bus(DISPLAY\_SET\_ON);

break;

case 8:

write\_to\_bus(CLR\_DISPLAY);

break;

case 9:

write\_to\_bus(ENTRY\_MODE);

break;

}

}

else {

}

}

while(((HWREG(I2C1\_BASE + I2C\_CNT) & BUFSTAT\_VAL) < NUM\_OF\_DBYTES) && ((HWREG(I2C1\_BASE + I2C\_CNT) & BUFSTAT\_VAL) != 0)) {

delay(1000);

}

HWREG(I2C1\_BASE + I2C\_CON) = 0;

}

void send\_name(void){

HWREG(I2C1\_BASE + I2C\_IRQSTATUS\_RAW) = 0x00000114;

HWREG(I2C1\_BASE + I2C\_CON) = I2C1\_ENABLE; //�Write 0x8000 to I2C\_CON (Configuration Register) offset 0xA4 to take out of reset, enable I2C1 module�

config\_master\_transmitter();

unsigned int current\_DCOUNT;

set\_slave\_addr(SLAVE\_ADDR);

set\_num\_databytes(NAME\_BYTE\_LENGTH);

while(is\_bus\_free() != TRUE){

}

startstop\_condition();

while((HWREG(I2C1\_BASE + I2C\_BUFSTAT) & BUFSTAT\_VAL) > 0){

current\_DCOUNT = HWREG(I2C1\_BASE + I2C\_BUFSTAT) & BUFSTAT\_VAL;

if(is\_i2c\_write\_ready()){//If ready to write

switch(NAME\_BYTE\_LENGTH-current\_DCOUNT){

case 0:

write\_to\_bus(WRITE\_TO\_WCO);

break;

case 1:

write\_to\_bus(SET\_DDRAM);

break;

case 2:

write\_to\_bus(WRITE\_TO\_RAM);

break;

case 3:

write\_to\_bus('M');

break;

case 4:

write\_to\_bus('i');

break;

case 5:

write\_to\_bus('k');

break;

case 6:

write\_to\_bus('e');

break;

case 7:

write\_to\_bus(' ');

break;

case 8:

write\_to\_bus('E');

break;

case 9:

write\_to\_bus('s');

break;

case 10:

write\_to\_bus('c');

break;

case 11:

write\_to\_bus('u');

break;

case 12:

write\_to\_bus('e');

}

}

}

}

int main(void){

set\_debug();

stack\_init();

i2c\_init();

init\_display();

send\_name();

wait();

return 1;

}

**INTERRUPT VERSION:**

/\*\* ECE 372 Project 2

\* Utilizes I2C1 to communicate with display board.

\* Interrupt Version

\* Author: Michael Escue

\*/

// Define Indirect Addressing Macro for Registers

#define HWREG(x) (\*((volatile unsigned int \*)(x)))

// Common Defines

#define TRUE 1

#define FALSE 0

#define DELAY\_COUNT 100000

// Base Module Defines

#define CTRLMOD\_BASE 0x44E10000

#define CM\_PER\_BASE 0x44E00000

#define I2C1\_BASE 0x4802A000

#define DEBUGSS\_DRM\_BASE 0x4B160000

#define INTC\_BASE 0x48200000

// Interrupt Module Defines

#define INTC\_PENDING\_IRQ2 0xD8

#define INTC\_MIR\_CLEAR2 0xC8

#define INTC\_CONTROL 0x48

// Interrupt Register Values

#define INTC\_NEW\_IRQ 0x1

#define INTC\_MIR\_CLEAR\_BIT 0x8

#define INTC\_PENDING\_BIT 0x80

// Control Module Defines

#define CONF\_SPI0\_CS0\_SCL 0x95C

#define CONF\_SPI0\_D1\_SDA 0x958

#define MODE2 0x22 //Both pins enabled as receivers in mode 2.

// Peripheral Control Module Defines

#define CM\_PER\_I2C1\_CLKCTRL 0x48

#define CLK\_ENABLE 0x2

// DRM Module Defines

#define I2C\_1\_SUSPEND\_CTRL 0x22C // Used for proper suspension of I2C during debug.

// I2C Module Defines

#define I2C\_SA 0xAC

#define I2C\_CNT 0x98

#define I2C\_DATA 0x9C

#define I2C\_IRQSTATUS\_RAW 0x24

#define I2C\_IRQSTATUS 0x28

#define I2C\_CON 0xA4

#define I2C\_PSC 0xB0

#define I2C\_SCLL 0xB4

#define I2C\_SCLH 0xB8

#define I2C\_OA 0xA8

#define I2C\_IRQENABLE\_SET 0x2C

#define I2C\_BUF 0x94

#define I2C\_BUFSTAT 0xC0

#define I2C\_IRQENABLE\_CLR 0x30

// I2C Register Values

#define \_12MHZ\_CLK 0x03

#define \_tLOW\_5MICROSEC 0x35

#define \_tHIGH\_5MICROSEC 0x37

#define OWNADDR 0x01

#define I2C1\_ENABLE 0x8000

#define IRQ\_DISABLED 0x0000

#define IRQ\_ENABLED 0x0010

// Data Byte Defines

#define WRITE\_TO 0x00

#define WRITE\_TO\_RAM 0x40

#define WRITE\_TO\_WCO 0x80

#define FUNCTION\_SET0 0x38

#define FUNCTION\_SET1 0x39

#define BIAS\_SET 0x14

#define CONTRAST\_SET 0b01110101

#define PWR\_ICON\_CON\_SET 0x5E

#define FOLLOWER\_SET\_ON 0x6D

#define DISPLAY\_SET\_ON 0x0C

#define CLR\_DISPLAY 0x01

#define ENTRY\_MODE 0x06

#define CGRAM\_SET 0x41

#define SET\_DDRAM 0b10000001

// Mask Defines

#define DCOUNT\_VAL 0x0000FFFF

#define XRDY\_BIT 0x00000010

#define XRDY\_RDY 0x00000010

#define RRDY\_BIT 0x00000008

#define RRDY\_RDY 0x00000008

#define BF\_BIT 0x00001000

#define BUS\_IS\_FREE 0

#define TXTRSH\_VAL 0x0000003F

#define RXTRSH\_VAL 0x00003F00

#define AERR\_BIT 0x00000080

#define DATA\_VAL 0xFF

#define BUFSTAT\_VAL 0x0000003F

#define ARDY\_BIT 0x00000004

#define TXFIFO\_CLR\_BIT 0x00000040

//I2C Communication Defines

#define SLAVE\_ADDR 0b0111100

#define NUM\_OF\_DBYTES 10

#define START\_COND 0x00000001

#define STOP\_COND 0x00000002

#define MASTER\_TX\_MODE 0x600

#define NAME\_BYTE\_LENGTH 13

// Variables

unsigned int x;

unsigned int y;

unsigned int isdisplayinit;

unsigned int issendinit;

unsigned int current\_DCOUNT;

volatile unsigned int USR\_STACK[100];

volatile unsigned int IRQ\_STACK[100];

void delay(unsigned long int y){

while(y>0){

y--;

}

}

void stack\_init(void){

//SET UP STACKS

//init USR stack

asm("LDR R13, =USR\_STACK");

asm("ADD R13, R13, #0x1000");

//init IRQ stack

asm("CPS #0x12"); //Switch to IRQ mode

asm("LDR R13, =IRQ\_STACK");

asm("ADD R13, R13, #0x1000");

asm("CPS #0x13"); //Switch to User Mode

}

void irq\_enable(void){

asm("mrs r0, CPSR");

asm("bic r0, r0, #0x80");

asm("msr CPSR\_c, R0");

}

void set\_debug(void){

HWREG(DEBUGSS\_DRM\_BASE + I2C\_1\_SUSPEND\_CTRL) = 0x9;

}

int is\_bus\_free(void){

x = HWREG(I2C1\_BASE + I2C\_IRQSTATUS\_RAW); //"Read mask 0x00001000 from I2C\_IRQSTATUS\_RAW (I2C Status Raw Register) offset 0x24 to check bus status.

x = (x & BF\_BIT); //Mask.

if(x == BUS\_IS\_FREE) return 1;

else return 0;

}

int is\_i2c\_write\_ready(void){

x = HWREG(I2C1\_BASE + I2C\_IRQSTATUS\_RAW); //�Read mask 0x00000010 from I2C\_IRQSTATUS\_RAW (I2C Status Raw Register) offset 0x24 to see if write ready�

x = (x & XRDY\_BIT); //Mask.

if(x == XRDY\_RDY) return 1;

else return 0;

}

void clear\_i2c\_write\_ready(void){

HWREG(I2C1\_BASE + I2C\_IRQSTATUS\_RAW) = XRDY\_BIT;

}

int is\_i2c\_read\_ready(void){

x = HWREG(I2C1\_BASE + I2C\_IRQSTATUS\_RAW); //�Read mask 0x00000008 I2C\_IRQSTATUS\_RAW (I2C Status Raw Register) offset 0x24 see if data is ready.�

x = (x & RRDY\_BIT); //Mask.

if(x == RRDY\_RDY) return 1;

else return 0;

}

void startstop\_condition(void){

x = HWREG(I2C1\_BASE + I2C\_CON); //"Read-Modify-Write 0x3 to I2C\_CON (Configuration Register) offset 0xA4 to queue Start/Stop Condition.�

x = (x | START\_COND | STOP\_COND); //Mask.

HWREG(I2C1\_BASE + I2C\_CON) = x; //Write back.

}

void start\_condition(void){

x = HWREG(I2C1\_BASE + I2C\_CON); //"Read-Modify-Write 0x3 to I2C\_CON (Configuration Register) offset 0xA4 to queue Start/Stop Condition.�

x = (x | START\_COND); //Mask.

HWREG(I2C1\_BASE + I2C\_CON) = x; //Write back.

}

void stop\_condition(void){

x = HWREG(I2C1\_BASE + I2C\_CON); //"Read-Modify-Write 0x3 to I2C\_CON (Configuration Register) offset 0xA4 to queue Start/Stop Condition.�

x = (x | STOP\_COND); //Mask.

HWREG(I2C1\_BASE + I2C\_CON) = x; //Write back.

}

void config\_master\_transmitter(void){

x = HWREG(I2C1\_BASE + I2C\_CON); //"Read-Modify-Write 0xE00 to I2C\_CON (Configuration Register)offset 0xA4 to configure mode."

x = (x | MASTER\_TX\_MODE); //Mask.

HWREG(I2C1\_BASE + I2C\_CON) = x; //Write back.

}

void set\_buf\_txtrsh(unsigned int y){

y = (y & TXTRSH\_VAL);

x = HWREG(I2C1\_BASE + I2C\_BUF);

y = (y | x);

HWREG(I2C1\_BASE + I2C\_BUF) = y; //"Write 0x0000 to I2C\_BUF (Buffer Configuration Register) offset 0x94 to set Transmit and Receive .�

}

void set\_buf\_rxtrsh(unsigned int y){

y = ((y<<8) & RXTRSH\_VAL);

x = HWREG(I2C1\_BASE + I2C\_BUF);

y = (y | x);

HWREG(I2C1\_BASE + I2C\_BUF) = y; //"Write 0x0000 to I2C\_BUF (Buffer Configuration Register) offset 0x94 to set Transmit and Receive .�

}

void set\_num\_databytes(unsigned int y){

y = (y & DCOUNT\_VAL);

//Number of Data Bytes pre-transmission.

HWREG(I2C1\_BASE + I2C\_CNT) = y; //"Write 0x9 to I2C\_CNT (Data Count Register) offset 0x98 to set number of transmission Bytes"

}

void write\_to\_bus(unsigned char x){

x = (x & DATA\_VAL );

HWREG(I2C1\_BASE + I2C\_DATA) = x; //Write to data bus.

clear\_i2c\_write\_ready();

delay(2000);

}

void set\_slave\_addr(unsigned int x){

//Slave address pre-transmission.

HWREG(I2C1\_BASE + I2C\_SA) = x; //"Write 0x78 to I2C\_SA (Slave Address Register) offset 0xAC Slave address value"

}

void init\_display\_initiate(void){

set\_slave\_addr(SLAVE\_ADDR);

set\_num\_databytes(NUM\_OF\_DBYTES);

startstop\_condition();

}

void send\_name\_initiate(void){

HWREG(INTC\_BASE + INTC\_CONTROL) = INTC\_NEW\_IRQ;

config\_master\_transmitter();

set\_slave\_addr(SLAVE\_ADDR);

set\_num\_databytes(NAME\_BYTE\_LENGTH);

issendinit = 0;

HWREG(I2C1\_BASE + I2C\_IRQENABLE\_SET) = XRDY\_BIT;

startstop\_condition();

}

void i2c\_init(void){

//P9 Connector settings.

HWREG(CTRLMOD\_BASE + CONF\_SPI0\_CS0\_SCL) = MODE2; //�Write 0x2 to conf\_spi0\_cs0 offset 0x95C to enable (SCL) for MODE2 w/o pullup�

HWREG(CTRLMOD\_BASE + CONF\_SPI0\_D1\_SDA) = MODE2; //�Write 0x2 to conf\_spi0\_d1 offset 0x958 to enable (SDA) for MODE2 w/o pullup�

//Enable Clock to I2C1.

HWREG(CM\_PER\_BASE + CM\_PER\_I2C1\_CLKCTRL) = CLK\_ENABLE; //�Write 0x2 to CM\_PER\_I2C1\_CLKCTRL offset 0x48 to enable I2C1 Clock.�

//Configure I2C1.

HWREG(I2C1\_BASE + I2C\_PSC) = \_12MHZ\_CLK; //�Write 0x03 to I2C\_PSC (Clock Prescalar Register) offset 0xB0 for ICLK of 12 MHz�

HWREG(I2C1\_BASE + I2C\_SCLL) = \_tLOW\_5MICROSEC; //"Write 0x35 to I2C\_SCLL (SCL Low Time Register) offset 0xB4 for tLOW to get 100kbps (5us-Low)"

HWREG(I2C1\_BASE + I2C\_SCLH) = \_tHIGH\_5MICROSEC; //"Write 0x37 to I2C\_SCLH (SCL High Time Register) offset 0xB8 for tHIGH to get 100kbps (5us-High)"

HWREG(I2C1\_BASE + I2C\_OA) = OWNADDR; //�Write 0x001 to I2C\_OA (Own Address Register) offset 0xA8 to configure Own Address�

HWREG(I2C1\_BASE + I2C\_CON) = I2C1\_ENABLE; //�Write 0x8000 to I2C\_CON (Configuration Register) offset 0xA4 to take out of reset, enable I2C1 module�

config\_master\_transmitter();

HWREG(INTC\_BASE + INTC\_MIR\_CLEAR2) = 0xFFFFFFFF;

HWREG(I2C1\_BASE + I2C\_IRQENABLE\_SET) = IRQ\_ENABLED; //"Write 0x0000 to I2C\_IRQENABLE\_SET (Interrupt Enable Set Register) offset 0x2C to enable Polling�

set\_buf\_txtrsh(0);

set\_buf\_rxtrsh(0);

HWREG(INTC\_BASE + INTC\_CONTROL) = INTC\_NEW\_IRQ;

}

void init\_display(void){

current\_DCOUNT = HWREG(I2C1\_BASE + I2C\_BUFSTAT) & BUFSTAT\_VAL;

switch(NUM\_OF\_DBYTES-current\_DCOUNT){

case 0:

write\_to\_bus(WRITE\_TO);

break;

case 1:

write\_to\_bus(FUNCTION\_SET0);

break;

case 2:

write\_to\_bus(FUNCTION\_SET1);

break;

case 3:

write\_to\_bus(BIAS\_SET);

break;

case 4:

write\_to\_bus(CONTRAST\_SET);

break;

case 5:

write\_to\_bus(PWR\_ICON\_CON\_SET);

break;

case 6:

write\_to\_bus(FOLLOWER\_SET\_ON);

break;

case 7:

write\_to\_bus(DISPLAY\_SET\_ON);

break;

case 8:

write\_to\_bus(CLR\_DISPLAY);

break;

case 9:

write\_to\_bus(ENTRY\_MODE);

isdisplayinit = 0;

break;

}

HWREG(I2C1\_BASE + I2C\_IRQENABLE\_SET) = XRDY\_BIT;

}

void send\_name(void){

current\_DCOUNT = HWREG(I2C1\_BASE + I2C\_BUFSTAT) & BUFSTAT\_VAL;

switch(NAME\_BYTE\_LENGTH-current\_DCOUNT){

case 0:

write\_to\_bus(WRITE\_TO\_WCO);

break;

case 1:

write\_to\_bus(SET\_DDRAM);

break;

case 2:

write\_to\_bus(WRITE\_TO\_RAM);

break;

case 3:

write\_to\_bus('M');

break;

case 4:

write\_to\_bus('i');

break;

case 5:

write\_to\_bus('k');

break;

case 6:

write\_to\_bus('e');

break;

case 7:

write\_to\_bus(' ');

break;

case 8:

write\_to\_bus('E');

break;

case 9:

write\_to\_bus('s');

break;

case 10:

write\_to\_bus('c');

break;

case 11:

write\_to\_bus('u');

break;

case 12:

write\_to\_bus('e');

}

HWREG(I2C1\_BASE + I2C\_IRQENABLE\_SET) = XRDY\_BIT;

}

//Not used in this polling version

void int\_handler(void){

x = HWREG(INTC\_BASE + INTC\_PENDING\_IRQ2); //Check for shared interrupt.

x = (x & INTC\_PENDING\_BIT); // Check bit 7

if(x == INTC\_PENDING\_BIT){

x = HWREG(I2C1\_BASE + I2C\_IRQSTATUS\_RAW); //Check IRQ status.

if((x & XRDY\_BIT) == XRDY\_BIT){

HWREG(I2C1\_BASE + I2C\_IRQENABLE\_CLR) = XRDY\_BIT; //Disable interrupt

if(isdisplayinit != 0)

init\_display();

else{

if (issendinit != 0)

send\_name\_initiate();

else

send\_name();

}

}

}

HWREG(INTC\_BASE + INTC\_CONTROL) = INTC\_NEW\_IRQ;

asm("LDMFD SP!, {LR}");

asm("LDMFD SP!, {LR}");

asm("SUBS PC, LR, #0x4");

}

void wait(void){

while(1){

}

}

int main(void){

set\_debug();

stack\_init();

i2c\_init();

irq\_enable();

init\_display\_initiate();

wait();

return 1;

}