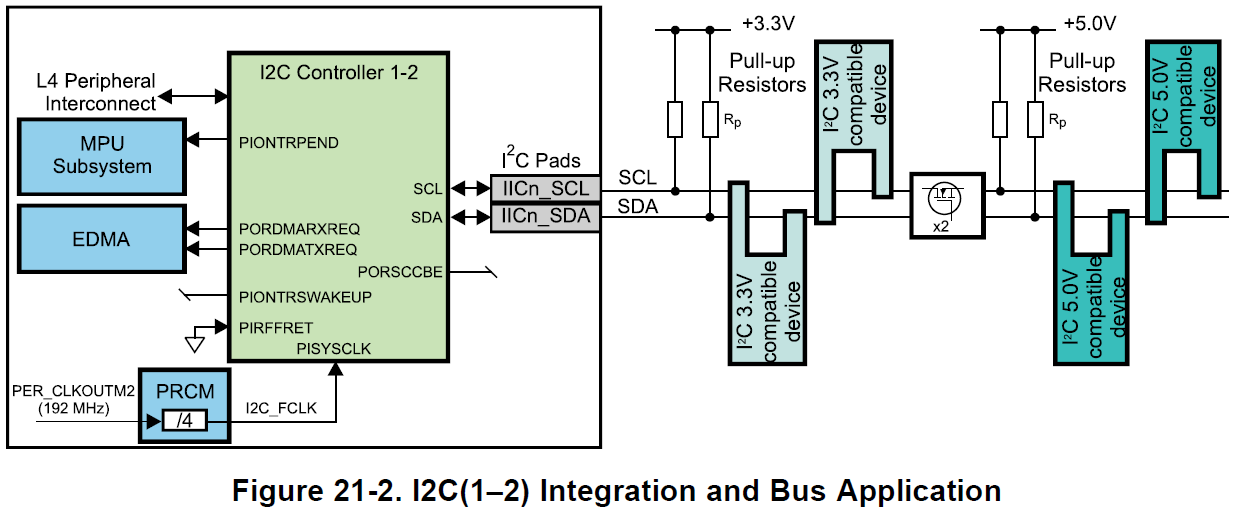
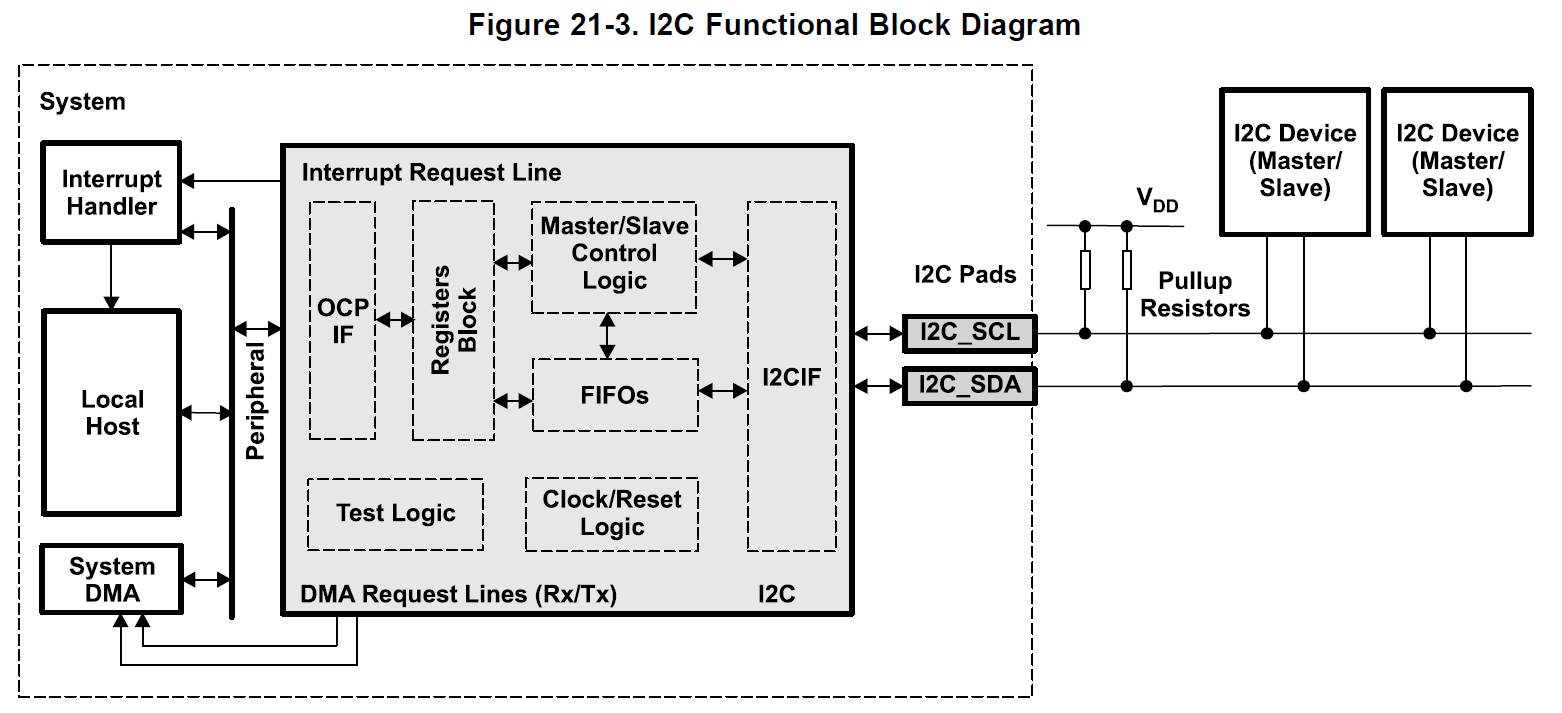
Target: Beagle Bone Black I2C1 Controller





**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.*

**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**

0x95Cconf\_spi0\_cs0 0x2(010b) **(for MODE2 w/ pullup)**

* 0x2 used because no pull up/down required (external pull-up used?) and receiver is disabled since only one master is used.

**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/ pullup)**

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

* + 0x2 used because no pull up/down required (external pull-up used?) and receiver is disabled since only one master is used.
* **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**

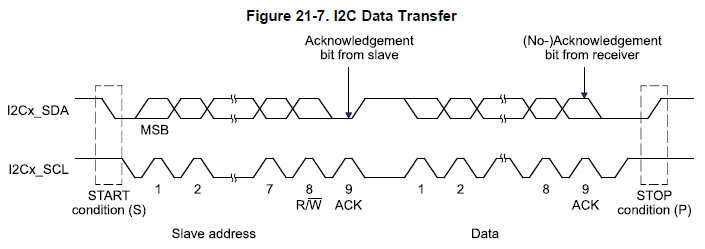
CM\_PER 0x44E00000 0x44E003FF Clock Module Peripheral Registers

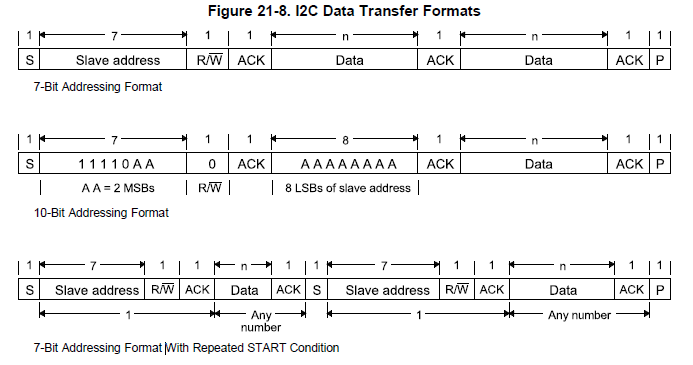
**Offset Clock Module Peripheral Register Write Value (To Enable)**

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**

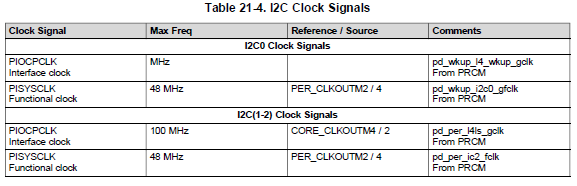
I2C1 0x4802A000 0x4802BFFF I2C1 Registers

* **Configure the Module Before Enabling it.**
  1. 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

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

**Offset I2C1 Register Write Value**

0xB0 I2C\_PSC (Clock Prescalar) 0x03



* 1. 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

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

**Offset I2C1 Register Write Value**

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

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

* 1. 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.

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

**Offset I2C1 Register Write Value**

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

* 1. 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**
  1. Configure the I2C Mode Register.
     + I2C\_CON bits.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initial Settings for I2C\_CON** | | | | | | | | | | | | | | | | | | | | | |
| **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 | | | | E | | | | | 0 | | | | | | | 0 | | | |

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

0xA4 I2C\_CON (Configuration Register) 0xE00 (R-M-W (OR), module 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.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization Settings for I2C\_IRQENABLE\_SET Register (Interrupts in use, all disabled for 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 | 1 | 1 | 0 | 0 | 0 |
| **Write**  **Hex** | 0 | 0 | | | | 0 | | | | 1 | | | | 8 | | | |

**Offset I2C1 Register Write Value**

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

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**

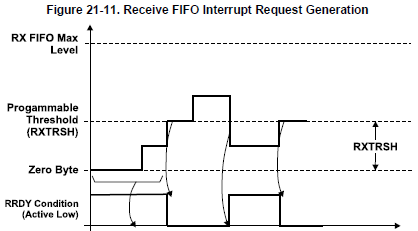
71 I2C1INT I2C1 POINTRPEND

* Data Buffer Register (IDBR or I2C\_BUF on Sitara)

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

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

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

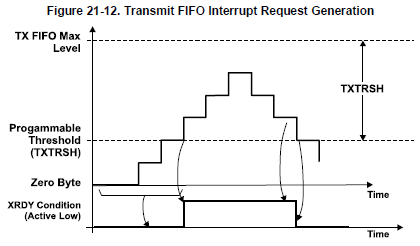


* An interrupt is generated whenever the signal is active.
* The **RRDY** signal must be cleared by the CPU by writing a 1 in the corresponding interrupt flag.
* If the condition is still present after clearing another interrupt will be generated.
  + Ensure the condition has changed first (I.e. RX FIFO 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.

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

**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.

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

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

0x28 I2C\_IRQSTATUS (Status Register) 0x0010

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**
  1. Configure slave address using I2C\_SA.
     + From the Newhaven NHD‐C0220BiZ‐FSW‐FBW‐3V3M data sheet:

Slave Address = 0x78

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

Slave Address = 0x3C to 0x3F

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| **Write**  **Hex** | 0 | 0 | | | | 7 | | | | 8 | | | |

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

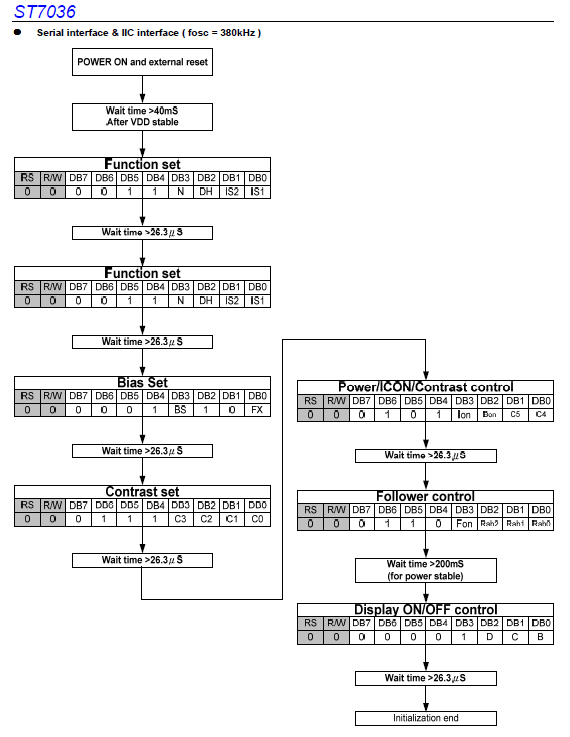
0xAC I2C\_SA (Slave address Register) 0x78

* 1. 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

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Initialization Settings for I2C\_CNT Register (Interrupts in use)** | | | | | | | | | | | | | | | | | |
| **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** | | | | | | | | | | | | | | | |
| **Write**  **Binary** | 0 | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? | ? |
| **Write**  **Hex** | 0 | ? | | | | ? | | | | ? | | | | ? | | | |

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

0x98 I2C\_CNT (Data Count Register) 0x[Transmission Dependent]



* **Initiate a Transfer**
  1. 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.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **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** |
| **Write**  **Binary** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **Write**  **Hex** | 0 | 1 | | | | 0 | | | | 0 | | | | 0 | | | |

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

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

* 1. 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)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Start Condition for Transmission 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 | | 0 | 1 |
| **Write**  **Hex** | 0 | 8 | | | | E | | | | | 0 | | | | | | | | 1 | | | | |

**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.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Stop Condition generation on I2C\_CON to end 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 | 0 |
| **Write**  **Hex** | 0 | 8 | | | | E | | | | | 0 | | | | | | | | 2 | | | | |

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

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

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

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

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

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

* 1. 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**
  1. Poll the transmit data ready interrupt flag bit (**XRDY**) in the I2C status register I2C\_IRQSTATUS\_RAW.

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

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

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

* 1. 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.

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

* I2C Standard, low-speed operation is 0 to 100KHz

Create an initialization pin map list for the required registers.

Registers for initialization

* Module Configuration

**Offset I2C1 Register Value Description**

* 1. 0xB0 I2C\_PSC (Clock Prescalar Register) 0x03 Write for ICLK of 12MHz
  2. 0xB4 I2C\_SCLL (SCL Low Time Register) 0x35 Write for tLOW to get 100kbps
  3. 0xB8 I2C\_SCLH (SCL High Time Register) 0x37 Write for tHIGH to get 100kbps
  4. 0xA8 I2C\_OA (Own Address Register) 0x001 Write to configure Own Address
  5. 0xA4 I2C\_CON (Configuration Register) 0x8000 Write to enable I2C1 module
* Module Initialization

**Offset I2C1 Register Value Description**

* 1. 0xA4 I2C\_CON (Configuration Register) 0xE00 Mode = Master, Transmitter
  2. 0x2C I2C\_IRQENABLE\_SET (Interrupt Enable) 0x0000 Polling, No Interrupts (Part 1)
  3. 0x94 I2C\_BUF (Buffer Configuration Register) 0x0000 Txt and Rx Thresholds.
* Pre-Transmission Initialization

**Offset I2C1 Register Value Description**

* 1. 0xAC I2C\_SA (Slave Address Register) 0x78 Newhaven Display = 0x78
  2. 0x98 I2C\_CNT (Data Count Register) 0x[n] n = Depends on Transmission

Registers for transmission start

* Transmission

**Offset I2C1 Register Value Description**

* 1. 0x24 I2C\_IRQSTATUS\_RAW (I2C Status Raw) 0x1000 Read BB, see if Bus is free
  2. 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**

* 1. 0x24 I2C\_IRQSTATUS\_RAW (I2C Status Raw) 0x00000008 Read RRDY see if data is ready
  2. 0x9C I2C\_DATA (Data Access Register) ? Read Received Data Byte
  3. 0x28 I2C\_IRQSTATUS (Status Register) 0x00000008 If RRDY "1", clear RRDY
* Transmit Data

**Offset I2C1 Register Value Description**

* 1. 0x24 I2C\_IRQSTATUS\_RAW (I2C Status Raw) 0x00000010 Read XRDY see if data is needed
  2. 0x9C I2C\_DATA (Data Access Register) ? Write Data Byte
  3. 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).

Initialization.

Transmission sequence.

* Send address.
* Assert Start Condition.
* Write to register
* Assert Stop Condition.
* Wait (65ms in example).
* Write address.
* Assert Start Condition
* Write to register.
* Assert repeated Stop Condition

Construct the Low-level algorithm initialization and transmission start.

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

Determine Capabilities.

Determine Internal Registers

Determine address mechanism.

Determine maximum SCL clock frequency.

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.

* After masking and IRQ/FIQ selection, and before priority sorting is done, the interrupt status is readable from the MPU\_INTC.INTC\_PENDING\_IRQn
  + Bank 2, INT 71
  + INTC\_MIR\_CLEAR2
  + INTC\_PENDING\_IRQ2
  + INTC\_CONTROL

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.