8/11 Serial Clock Line (SDL) and Serial Data (SDA) signals are handshaking signals between DCE and DTE

372 textbook (pg 90):

LCD Module Datasheet

- ~~Slave Address: 0x78 (LCD datasheet, pg 7) on the I^2C bus~~
  ~~LSB: 0 (write mode) → 0x78 for write operation~~
  ~~1 (read mode) → 0x79 for read operation~~

8/20: WRONG. After discussion with TA, use 0x3C

- DDRAM address on pg 6 (LCD datasheet)
- Table of commands (pg 8-9, LCD datasheet)
- Timing characteristics (pg 10, LCD datasheet)

  f_{SCLK} = 300 KHz @ 3.3V (SCL clock f)

- Sample code for controller initialization, data transfer, and send ASCII string (pg 12-13, LCD datasheet
- ASCII? character table (pg 11, LCD datasheet)

8/18 . LCD Controller Datasheet (ST7036)

- ~~Page~~ I^2C info, bottom page 14 (LCD controller tech. ref)
- Slave address for LCD is "011 1100"

On Data line

- High-to-Low transition, while clock is high, is START condition (pg 14, LCD cont. tech. ref)
- Low-to-High transition, while clock is high, is STOP condition (pg 14)

- BBB is Master/transmitter, LCD is Slave device
- I^2C Transmission Steps (High Level): (pg 16 LCD Controller Ref)

  - START condition from Master, followed by Slave address
  - Slave acknowledge. After acknowledgement, 1+ command words follow. Command word includes a control byte (defines control/Reg select bit, Co/RS) and a data byte.

    Logic 0

    After command byte with cleared MSB (Co bit), only data bytes follow. Otherwise, another control byte will foll data byte

    RS bit status defines data byte as command data or RAM data
  - After last control byte and RS bit status,

    RS = logic 1, series of display data bytes will follow. Display bytes are stored in display RAM at address specified by data pointer

    RS = logic 0, series of command data bytes will follow. Command bytes will be decoded and change device settings based on commands

    Slave makes acknowledgement after each transmitted byte.
  - STOP condition from Master.

Detailed I^2C transmission steps to LCD
A. Write 0x78 to send slave address of LCD
B.

8/19

I^2C Signal Mapping (SCL and SDA)

- Change mode to connect "I2C1_SCL" to pin 17 for P9 connector } Project 2 Info
  "I2C1_SDA" to pin 18
- Control Module base address: 0x44E10000 (pg 180, BBB TRM)
- Control module register offsets

I2C1_SCL: conf_spi0_cs0, 0x95C (Pg 86, BBB Sys. ref)
I2C1_SDA: conf_spi0_d1, 0x958 (pg 1458, BBB tech ref)

- To map signals, (pg 1512, BBB tech. ref)

bits 0-2: Mode select → (Mode 2 for SCL AND SDA), 010
bits 3&4: Pullup/Pulldown enable(0)/disable(1) and Pad pulldown(0)/pullup selected(1)
bit 5: Write 1 to enable receiver, 0 to disable
bit 6: slew rate, fast(0)/slow(1)

SCL

bit#	6	5	4	3	2	1	0
bit val	1	1	1	0	0	1	0
hex	3			2			

SDA

bit#	6	5	4	3	2	1	0
bit val	1	1	1	0	0	1	0
hex	3			2			

8/22 update (after discussion with TA):
bit 6 needs to be
bit 5 = 1, need to receive acknowledge
bit 4 = 1, pullup selected to keep line high
bit 3 = 0, enable pullup/pulldown functionality

✓ ⟹ RMW (OR, set bits) to map I^2C signals to P9 connector

Write 0x32 to 0x44E1095C for SCL signal (I2C_SCL)
Write 0x32 to 0x44E10958 for SDA signal (I2C_SDA)

I^2C Module Clock

(pg 1268, BBB Tech Ref) (Pg 179 BBB Tech Ref)

⟹ Write 0x2 to CM_PER_I2C1_CLKCTRL at 0x44E00000 + 0x48

(Pg 1268 BBB Tech Ref)

I^2C base address

⟹ 0x4802A000 (Pg 181 BBB Tech Ref) (I2C1)

8/20

I^2C 7-Bit Addressing Format

START (1bit) Condition	Slave address (7 bits)	R/ $\overline{W}$	A C K.	Data (n bits)	A C K	Data (n bits)	A C K	1bit STOP condition

7-Bit Addressing (pg 4588, BBB Tech Ref

I^2C 48MHz ⟶ 12 MHz (pg 4590 BBB Tech Ref ⟶ Need Prescaler (21.3.9))

I^2C Clock Prescaler Register (I2C_PSC) offset: 0xB0 (pg 4636, BBB Tech Ref)

$\frac{48\ MHz}{12\ MHz} = 4$ ⟶ **Write 0x3 to I2C_PSC at 0x4802A0B0 to scale clock to 12 MHz**

(PSC+1)

I^2C 100 Kbps SCL (pg 4596, BBB Tech Ref 21.3.15.1) (pg 10, LCD ref) ← assuming 20% Duty cycle

I2C_SCLL @ Offset 0xB4 (pg 4637, BBB Tech Ref);
I2C_SCLH @ Offset 0xB8 (pg 4638, BBB Tech Ref)

$t_{LOW} = (SCLL+7) \cdot \text{Internal CLK} \Longrightarrow SCLL = \frac{t_{Low}}{ICLK} - 7 =$

8/20: TA said assume 50% DC, instead of 20%
: Datasheet says -tHIGH=...
After discussion with TA, use positive ⟶

$+t_{HIGH} = (SCLH+5) \cdot ICLK \Longrightarrow SCLH = \frac{+t_{HIGH}}{ICLK} - 5 =$

Assume 50% Duty Cycle @ 100 KHz $\Longrightarrow \frac{1}{100\,KHz} = 10\,\mu s \Longrightarrow 5\,\mu s$ high/low

$SCLL = \frac{t_{LOW}}{ICLK} - 7 = \frac{(5\,\mu s)}{1/12MHz} - 7 = 53 = 0x35$

$SCLH = \frac{+t_{HIGH}}{ICLK} - 5 = \frac{+(5\,\mu s)}{1/12\,MHz} - 5 = 55 = 0x37$

⟹ **Write 0x35 to I2C_SCLL at 0x4802A0B4**
Write 0x37 to I2C_SCLH at 0x4802A0B8 } To set data rate to 100 Kbps at 50% duty cycle

8/20 How to Program I^2C (pg 4596 BBB Tech Ref)

- Configure Slave address and DATA counter registers
  Slave: I2C_SA : offset 0x AC
  Data: I2C_CNT: offset 0x 98 } Pg 4598, BBB tech ref

⇒ Write 0x3C to I2C_SA at 0x4802A0AC to configure LCD address

Write 0x4 to I2C_CNT at 0x4802A098 to configure # bytes per transfer
↳ 4 bytes (START, slave address, data, STOP) (Maybe 5 bytes to include command?)

Pg 4597 BBB Tech Ref

- Initiate a Transfer
  Poll bit busy (BB) bit of I2C_IRQSTATUS_RAW. If bit=0, bus not busy and START/STOP (I2C_CON STT and STP conditions)

- Transmit Data
  Poll transmit data ready flag bit (XRDY) in I2C status register (I2C_IRQSTATUS_RAW), use the XRDY interrupt (I2C_IRQENABLE_SET.XRDY_IE set) to write data into data transmit register (I2C_DATA). Use draining feature (I2C_IRQSTATUS_RAW.XDR enabled by I2C_IRQENABLE_SET.XDR_IE) if transfer length ≠ with FIFO threshold

8/21 High Level Program List:

1. Turn on I^2C
2. Adjust CLK
3. Set Transmission rate ← Signed? ← Other steps as required from 21.3.15.3 in BBB Tech Ref
   Write to I2C_CON (pg 4631, BBB tech ref)
4. Turn on LCD

To Send:

1. Send slave address
2. Move cursor (except for first char)
3. Other steps from step 2 in ECE 372 text

I^2C Initialization:

1. Turn on I^2C module clk (pg 3, notes)
2. Map I^2C SCL and SDA signals (pg 3, notes)
3. Program prescaler (pg 4, notes)
4. Program 100 Kbps (pg 4, notes)
5. ~~Configure own address~~ (only for slave mode, don't need?)

8/21: Confirmed, don't need since configured in Master Mode (discussion with TA)

21.3.15.1, BBB TRM → 6. Take I^2C module out of reset (bit 15=1, I2C_CON, offset 0xA4) (pg 4631, BBB TRM)

.1.3.15.2, BBB TRM → 7. Configure I2C mode registers bits (I2C_CON)

bit 14, reserved → 0
bit 13-12 → 00 to select I^2C fast/standard mode
bit 11 → 0 to select normal mode
bit 10 → 1 for master mode
9 → 1 for transmitter mode
8-4 → 0 for 7-bit address mode
3-2 → reserved, 0
1 → 0 to reset STOP condition
0 → 0 to reset START condition

*Not using interrupts or DMA (Steps 2 & 3 of 21.3.15.2 pg 4596 BBB TRM)

bit #	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
bit val	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
hex	8				6				0				0			

halfword to write to I2C_CON

⇒ **Write 0x8600 to I2C_CON at 0x4802A0A4 to configure I^2C**

Initiate Transmission

I2C_IRQSTATUS_RAW (offset 0x24, pg 4603 BBB TRM)

8/21: Need to Poll bit 12 (0x100

Bus Busy (BB) bit 12 → 0 = bus is free
1 = bus is busy

Transmit Ready (XRDY) bit 4 → 0 = Transmission in progress
1 = Transmit data ready

I2C_CON (offset 0xA4)

(bit 1) STOP bit → 1 to set condition
(bit 0) START bit → 0 to reset condition

Transmit data

I2C_IRQENABLE_SET (offset 0x2C, pg 4611 BBB TRM)

XRDY_IE (bit 4) → 0 = Transmit data ready IRQ disabled
1 = Transmit data ready IRQ enabled

8/21 Step 5 of Suggested procedure from Project Problem Statement

Register List: (pg 4598, BBB TRM)
- I2C-IRQSTATUS_RAW (see pg 6, log)
- I2C_DATA (offset 0x9C) (pg 4630, BBB TRM)
- I2C_CON (Log pg 6)
- I2C_SA (Log pg 5)
- I2C_PSC (Log pg 3)
- I2C_SCLL (Log pg 4)
- I2C_SCLH (Log pg 4)

I2C_DATA: bits 0-7 are data bits

STEP 6 from Project Problem Statement

See pg 87-88 in ECE 372 text, as well as pg 16 in ST7036 Datasheet

High Level (Transfer Data)
- Initialize button & I^2C
- Polling for button press in start/wait loop

Initiate Transfer 21.3.15.4 { When pressed, branch to BB poll procedure (POLL_BB)
POLL_BB: poll bit 12 (0x1000), b
IFbit 12=0, THEN assert start condition by writing 0x1 to I2C_CON (bit 0)

21.3.15.6 Transmit Data {
- Polling XRDY (bit 4, I2C_IRQSTATUS_RAW) in POLL_XRDY procedure
POLL_XRDY; IF bit 4=1, start SEND procedure
- SEND:
Set XRDY_IE bit (bit 14) in I2C_IRQENABLE_SET
Write byte to send to I2C_DATA

8/22 Step 7 from Project Problem Statement (ST7036 controller datasheet)

- During Read/Write operation, two 8-bit registers (data register DR, instruction register IR)
  - ST7036 will handle transferred data from BBB automatically (pg 14)

- Address Counter automatically decreases by 1 after Read/Write to DDRAM (display data ram). When RS=LOW and R/W=HIGH, address counter can be read through DB0-DB6 ports. (pg 17)

- LCD can display 80 characters max (pg 18)

- Character Table: row first (B0-B3), column second (B4-B7) (pg 22)

- Useful LCD instructions (pg 26)

- LCD instruction description (pg 29)

- Reset/Initialization steps (done automatically on powerup) (pg 38)
  - Need to turn on display and cursor

- Timing Characterics (pg 56)

- Characters to send Table (base on pg 22)

CHARACTER	B0-B3	B4-B7	Hex
S	0011	0101	53
h	1000	0110	68
a	0001	0110	61
u	0101	0111	75
n	1110	0110	6E
	0000	0010	20
C	0011	0100	43
r	0010	0111	72
i	1001	0110	69
p	0000	0111	70
p	0000	0111	70
e	0101	0110	65
n	1110	0110	6E

- Initialize LCD after powerup

DISPLAY ON/OFF:

hex 0xF

RS	R/W	DB7	6	5	4	3	2	1	DB0
0	0	0	0	0	0	1	1	1	1

DB2: 1=display ON, 0=OFF
DB1: 1=cursor ON, 0=OFF
DB0: 1=cursor blink ON, 0=OFF

Needed? seems to set up in powerup (pg 38)

FUNCTION SET:

RS	R/W	DB7	6	5	4	3	2	1	DB0
0	0	0	0	1	1	0	1	0	0

DB4: 1=8-bit, 0=4-bit
DB3: 1=2-line display, 0=1-line
DB2: 1=double height
DB0 & DB1: 0=normal instruction

8/22 Write command to turn on display: (based on pg 17)

Co	RS						
0	0	0	0	0	0	0	0

Write 0x00 to I2C_DATA
Write 0x0F to I2C_DATA to turn on display (pg 8, log)
(NOTE: need polling to send each instruction)

8/23 High Level Control/Data

1. Poll BB bit (bit 12) in I2C_IRQSTATUS_RAW
2. IF BB=0, assert STT bit (bit 0) in I2C_CON
3. Poll XRDY (bit 4) in I2C_IRQSTATUS_RAW

Send data

1. IF XRDY=1,

Polling Button:

- offset 0x138 (pg 5004, BBB TRM) GPIO1 base address: 0x4804C000 (pg 182, BBB TR
- Test bit 30 (0x40000000)

8/23 After discussion with TA, I found out I was mistaken that the ST7036 automatically initializes. I must first send the 10 initialization instructions) from pg 44 of the ST7036 datasheet in the initialization section of code. The list ofe data bytes are:

LCD Initialization
0x00
0x38
0x39
0x14
0x78
0x5E
0x6A
0x0C
0x01
0x06

LCD initialization steps:
1. Poll BB
2. IF BB=0, write 0x1 to I2C1_CON(RMW) to assert start condition
3. Poll XRDY
4. IF XRDY=1, write control command (starting with 0x00, then move down list) (listed above) to I2C1_DATA
5. Repeat from step 3 withnext consecutive command
6. Write 0x2 to I2C1_CON (RMW) to assert STOP condition

Send character data steps:
1. Write 0x8 to I2C_CNT
2. Poll button
3. Poll BB
4. Assert START condition
5. Poll XRDY
6. WRITE 0x40 to I2C_DATA to indicate character data (set RS bit, bit 6)
7. Repeat from step 5 and write character byte from log page 8 (Repeat for each char)
8. Write 0x2 to assert STOP condition

9/3 Ran code... didn't work. I didn't even get the LCD to initialize and display boxes. After working with TA I found that writing 0x8603 to I2C1_CON allowed the beaglebone to assert START and STOP automatically. I previously thought I had to manually assert START and STOP by writing to I2C1_CON twice

I also added a delay after the LCD initialization sending loop. I now get the boxes on the LCD showing that everything is initialized, but I sporadically get parts or all my message to display.

To get the message to display at all was to send 0x80, 0x06, and 0x40 before but with the message.

9/5 After meticulously going line-by-line through my code, I couldn't find a way to make my message reliably diplay on the LCD. With the TA, we connected a logic analyzer to the LCD lines. We saw that there were two separate sends occuring as expected, but not all the bytes were being sent in the correct send loops. I then stepped through my code using CCS's disassembly view

With the TA's suggestions, I first checked that the I2C1_CNT was being loaded correctly with the number of bytes to be sent as well as decrementing correctly. After that checked out, I made sure that STOP was asserted when I2C1_CNT reached 0 and that the loop counter was synced with I2C1_CNT. With I2C1_CNT and the loop counter functioning as expected, I continue stepping through the message send in disassembly to see if I saw issues. I saw no issues stepping through and checking relevant registers. After completing the send loop in this manner, my message was displayed on the LCD! The TA and I concluded the my code was correct, but had a timing issue.

I ultimately added another delay in the message send loop to allow the LCD to process each character sent. I now had my message reliably send.

I finally implemented the button with changling the value of my delays since the button checking had inherent delay.

I checked for a button press by polling GPIO1_DATAIN before each message byte was sent. I was advised by the TA to use the interrupt code from the previous project to simplify the work, but I didn't like having interrupts and polling in the same code. Do one or the other haha!

Implementing the button completed part 1 of ECE 372 project 2. I display my name on an LCD one character at a time with each consecutive button press.

Shaun Crippen
ECE 372 Project 2

I^{2}C DRIVER ALGORITHMS, HIGH LEVEL

INITIALIZATIONS

1. Enable GPIO1 module clock
2. Enable I2C1 module clock
3. Set up falling edge detection on GPIO1_30 (button)
4. Change pin mapping on BBB P9 connectors to SCL and SDA lines, set up pullup resistor, fast slew rate
5. Scale I2C1 clock down to 12MHz
6. Set I2C1 data rate to 100Kbps @ 50% duty cycle
7. Configure I2C1 for master mode, transmission mode, 7-bit addressing, and take out of reset
8. Configure LCD slave address in BBB
9. Set number of command bytes to transmit
10. Wait for I2C bus free
11. Load START/STOP conditions
12. Wait for transmit buffer empty
13. Send 10 LCD initialization bytes
14. Wait for LCD to process sent commands

MESSAGE TRANSMISSION

1. Set number of command/character bytes to transmit
2. Wait for I2C bus free
3. Load START/STOP conditions
4. Wait for transmit buffer empty
5. Wait for button press
6. Send 16 message bytes, including delay after each byte sent to allow LCD to process data

Shaun Crippen

ECE 372 Project 2

I^{2}C DRIVER ALGORITHMS, LOW LEVEL

INITIALIZATIONS

```
1.  Enable GPIO1 module clock
        Write 0x2 to CM_PER_GPIO1_CLKCTRL at 0x44E000AC
2.  Enable I2C1 module clock
        Write 0x2 to CM_PER_I2C1_CLKCTRL at 0x44E00048
3.  Set up falling edge detection on GPIO1_30 (button)
        Load word from GPIO1_FALLINGDETECT at 0x4804C14C
        Set bit by “ORing” word with 0x40000000
        Write result back to GPIO1_FALLINGDETECT at 0x4804C14C
4.  Change pin mapping on BBB P9 connectors to SCL and SDA lines, set up pullup resistor, fast slew rate
        Write 0x32 to I2C_SDA at 0x44E10958
        Write 0x32 to I2C_SCL at 0x44E1095C
5.  Scale I2C1 clock down to 12MHz
        Write 0x3 to I2C1_PSC at 0x4802A0B0
6.  Set I2C1 data rate to 100Kbps @ 50% duty cycle
        Write 0x35 to I2C1_SCLL at 0x4802A0B4
        Write 0x37 to I2C1_SCLH at 0x4802A0B8
7.  Configure I2C1 for master mode, transmission mode, 7-bit addressing, and take out of reset
        Write 0x8600 to I2C1_CON at 0x4802A0A4
8.  Configure LCD slave address in BBB
        Write 0x3C to I2C1_SA at 0x4802A0AC
9.  Set number of command bytes sent to transmit
        Write 0xA to I2C1_CNT at 0x4802A098
10. Wait for I2C bus free
        Test 0x1000 (bit 12) in I2C1_IRQSTATUS_RAW at 0x4802A024, stop polling when bit 12 = 0
11. Load START/STOP conditions
        Write 0x8603 to I2C1_CON at 0x4802A0A4
12. Wait for transmit buffer empty
        Test 0x10 (bit 4) in I2C1_IRQSTATUS_RAW at 0x4802A024, stop polling when bit 4 = 1
13. Send 10 LCD initialization bytes
        Load command byte string pointer
        Load loop counter to 0xA
        WHILE loop counter > 0
                Load next command byte, post increment pointer
                Write loaded byte in I2C1_DATA at 0x4802A09C
        END WHILE loop
14. Wait for LCD to process send commands
        Delay loop for 0x1000000
```

MESSAGE TRANSMISSION

```
1.  Set number of command/character bytes to transmit
        Write 0xA to I2C1_CNT at 0x4802A098
2.  Wait for I2C bus free
        Test 0x1000 (bit 12) in I2C1_IRQSTATUS_RAW at 0x4802A024, stop polling when bit 12 = 0
3.  Load START/STOP conditions
        Write 0x8603 to I2C1_CON at 0x4802A0A4
4.  Wait for transmit buffer empty
        Test 0x10 (bit 4) in I2C1_IRQSTATUS_RAW at 0x4802A024, stop polling when bit 4 = 1
5.  Wait for button press
```

```
        Test 0x40000000 (bit 30) in GPIO1_DATAIN at 4804C138, stop polling when bit 30 = 0
6.  Send 16 message bytes, including delay after each byte sent to allow LCD to process data
        Load message byte string pointer
        Load loop counter to 0x10
        WHILE loop counter > 0
                Load next message byte, post increment pointer
                Write loaded byte in I2C1_DATA at 0x4802A09C
                Delay loop for 0x10000
        END WHILE loop
```