RYAN BENTZ

ECE 372

PROJECT 2

3/16/2018

List of Deliverables

1. **Part I**
   - **A. Problem Description**
   - **B. High-Level Algorithm**
   - **C. Low-Level Algorithm**
   - **D. Project Code**

2. **Part II**
   - **E. Problem Description**
   - **F. High-Level Algorithm**
   - **G. Low-Level Algorithm**
   - **H. Project Code**

3. **Appendix**
   - **I. Supporting Documentation**
   - **J. Design Log**
   - **K. Signed Statement**

1. Part I

A. Problem Description

Display your name on a New Haven 2x20 LCD display by using the Beagle Bone Black I2C1 controller to communicate with the display. Use a polling method similar to the method shown in Hall chapter 7.

B. High-Level Algorithm

set up the stacks
initialize peripheral clocks
    - initialize the I2C clock for I2C 1
delay 1s and wait for peripheral clocks

INITIALIZE THE I2C:
initialize GPIO pins for I2C
    - set the mode for the SDA pin
    - set the mode for the SCL pin
configure the I2C module
    - program the prescaler for 12 MHz I2C clock
    - program I2C clock to obtain 100 Kbps
    - take the I2C module out of reset
initialize the I2C module
    - configure I2C mode register
    - enable interrupt masks
    - configure the slave address
delay 1s and wait for display module to be ready

INITIALIZE DISPLAY:
- transmit Function Set
    -Interface = 8 bits, Two line display mode, Use normal instruction set
- wait 1 ms
- transmit Function Set
    -Interface = 8 bits, Two line display mode, Use extension instruction set
- wait 1 ms
- transmit Bias Set
    - Bias = 1/5
- transmit Contrast Set
    - Set contrast bit C3
- transmit Power/ICON/Contrast Control
    - ICON display ON, Booster circuit ON, Set contrast bit C5
- wait 1 ms
- transmit Follower Control
- Internal follower circuit ON, set the amplified ration of the internal voltage generator

- wait 1 ms
- transmit Display ON/OFF
    - Turn on display, Cursor OFF, Blink OFF
- transmit Clear Display
    - clear the display
- transmit Entry Mode Set
    - cursor moves to the right
- wait 1 ms

Transmit command to display "RYAN BENTZ"

MAIN:
WHILE forever
    do nothing

I2C TRANSMIT PROCEDURE:
wait for the bus to be ready
initialize the data counter register
write the data to be transmitted to the FIFO
set module to master mode on every transfer
configure the start/stop bits
begin transmitting

DELAY:
Load countdown value
WHILE countdown > 0
    decrement value

C. Low-Level Algorithm

set up the stacks
initialize peripheral clocks
    - CMPER.IC21_CLKCTRL: initialize the I2C clock for I2C 1
        - *write 0x02 to CMPER.I2C1_CLKCTRL: 0x44E0 0048*
delay 1s and wait for peripheral clocks

INITIALIZE I2C:
initialize GPIO pins for I2C
    - CONTROL.conf_spi0_cs0: set the mode for the SDA pin
        - *write 0x2 to CONTROL.conf_spi0_cs0: 0x44E1 095C*
    - CONTROL.conf_spi0_d1: set the mode for the SCL pin
        - *write 0x2 to CONTROL.conf_spi0_d1: 0x44E1 0958*
configure the I2C module
    - I2C_PSC: program the prescaler for 12 MHz I2C clock

- *write 0x100 to I2C_PSC: 0x4802 A0B0*
- I2C_SCLL/SCLH: program I2C clock to obtain 100 Kbps
  - program SCL low time
    - *write 0x35 to I2C_SCLL: 0x4802 A0B4*
  - program SCL high time
    - *write 0x37 to I2C_SCLH: 0x4802 A0B8*
- take the I2C module out of reset
  - *write 0x8000 to I2C_CON: 0x4802 A0A4*

initialize the I2C module

- configure I2C mode register without setting STT and STP
  - *write 0x8C00 to I2C_CON: 0x4802 A0A4*
- configure the slave address
  - *write 0x3C to I2C_SA: 0x4802 A0AC*

wait 40 ms

INITIALIZE DISPLAY:

- transmit Function Set: *0x00, 0x38 to 0x3C*
  - load address of init string in R0
  - load number of bytes to send in R1
  - call I2C Transmit
- wait 1 ms
- transmit Function Set: *0x00, 0x39 to 0x3C*
  - load address of init string in R0
  - load number of bytes to send in R1
  - call I2C Transmit

- wait 1 ms
- transmit Bias Set, Contrast Set, Power/ICON/Contrast Control: *0x00, 0x14, 0x78, 0x5E*
  - load address of init string in R0
  - load number of bytes to send in R1
  - call I2C Transmit
- wait 1 ms
- transmit Follower Control, Display ON/OFF, Clear Display, Entry Mode Set
  - *0x00, 0x6D, 0x0C, 0x01, 0x06 to 0x3C*
  - load address of init string in R0
  - load number of bytes to send in R1
  - call I2C Transmit
- wait 1 ms

transmit command to display "RYAN BENTZ"

I2C TRANSMIT PROCEDURE:
R0 = memory address of bytes
R1 = number of bytes to send

push registers on to stack
wait for the bus to be ready
    - *read 0x1000 from I2C_IRQSTATUS_RAW: 0x4802 A024*
initialize the data counter register
    - *write number of bytes to I2C_COUNT: 0x4802 A098*
write the data to be transmitted to the FIFO
    - *write data to I2C_DATA: 0x4802 A09C*
set module to master mode on every transfer
    - *write 0x400 to I2C_CON: 0x4802 A0A4*
configure the start/stop bits
    - write 0x3 to I2C_CON: 0x4802 A0A4
begin transmitting
pop registers off stack and return

DELAY:
Load countdown value
WHILE countdown > 0
    decrement value

D. Project Code

See attached.

Part II

E. Problem Description

Display your name on a New Haven 2x20 LCD display by using the Beagle Bone Black I2C1 controller to communicate with the display. Use interrupts to communicate with the display and display your name on the screen.

F. High-Level Algorithm

set up the stacks
initialize peripheral clocks
    - initialize the I2C clock for I2C 1
delay 1s and wait for peripheral clocks

initialize GPIO pins for I2C
    - set the mode for the SDA pin
    - set the mode for the SCL pin
configure the I2C module
    - program the prescaler for 12 MHz I2C clock
    - program I2C clock to obtain 100 Kbps
    - take the I2C module out of reset
initialize the I2C module
    - configure I2C mode register
    - enable interrupt masks
    - configure the slave address
delay 1s and wait for display module to be ready

Initialize the interrupt controller
Initialize interrupts in the processor
Enable NACK and Bus Free I2C interrupts
Set ARDY flag to get transfer started

MAIN:
WHILE FOREVER
    Do nothing

INT DIRECTOR:
check that I2C was source of interrupt
IF I2C was source of interrupt
    go to I2C interrupt handler
ELSE
    reset the interrupt controller
    exit IRQ mode

I2C INTERRUPT HANDLER:
check if NACK response was received
IF NACK was received
    go to NACK handler
(ELSE it was the bus free or ARDY interrupt)
clear the ARDY interrupt
check if an initialization byte needs to be sent
IF initialization byte needs to be sent
    get address for initialization bytes
    get address for sequence offset
    load control byte
    go to I2C TX SETUP
ELSE we are sending character bytes
    check if message byte needs to be sent
    IF message byte needs to be sent
        get address for character message
        get address for sequence offset
        load control byte
        go to I2C TX SETUP
    ELSE transfer is finished
        mask I2C interrupts
        disable I2C interrupts
        reset interrupt controller
        return main

I2C NACK HANDLER:
check if we are doing an initialization
IF we are doing any initialization
    load initialization bytes
    load sequence offset
ELSE we are sending message bytes
    load message bytes
    load sequence offset
go to previous byte
decrement offset
go to I2C TX SETUP

I2C TX SETUP:
RO = Offset address
R1 = Stream address
get the next byte to transmit
load the control byte into the FIFO
load the data byte into the FIFO

increment the stream offset

I2C TX BEGIN:
update the count register
set module to master mode
configure the start and stop bits to begin transfer

DELAY:
Load countdown value
WHILE countdown > 0
    decrement value

G. Low-Level Algorithm

set up the stacks
initialize peripheral clocks
    - CMPER.IC21_CLKCTRL: initialize the I2C clock for I2C 1
        - *write 0x02 to CMPER.I2C1_CLKCTRL: 0x44E0 0048*
delay 1s and wait for peripheral clocks

INITIALIZE I2C:
initialize GPIO pins for I2C
    - CONTROL.conf_spi0_cs0: set the mode for the SDA pin
        - *write 0x2 to CONTROL.conf_spi0_cs0: 0x44E1 095C*
    - CONTROL.conf_spi0_d1: set the mode for the SCL pin
        - *write 0x2 to CONTROL.conf_spi0_d1: 0x44E1 0958*
initialize GPIO pins for I2C
    - CONTROL.conf_spi0_cs0: set the mode for the SDA pin
        - *write 0x2 to CONTROL.conf_spi0_cs0: 0x44E1 095C*
    - CONTROL.conf_spi0_d1: set the mode for the SCL pin
        - *write 0x2 to CONTROL.conf_spi0_d1: 0x44E1 0958*
configure the I2C module
    - I2C_PSC: program the prescaler for 12 MHz I2C clock
        - *write 0x100 to I2C_PSC: 0x4802 A0B0*
    - I2C_SCLL/SCLH: program I2C clock to obtain 100 Kbps
        - program SCL low time
            - *write 0x35 to I2C_SCLL: 0x4802 A0B4*
        - program SCL high time
            - *write 0x37 to I2C_SCLH: 0x4802 A0B8*
    - take the I2C module out of reset
        - *write 0x8000 to I2C_CON: 0x4802 A0A4*
initialize the I2C module
    - configure I2C mode register without setting STT and STP
        - *write 0x8C00 to I2C_CON: 0x4802 A0A4*

- enable interrupt masks
- configure the slave address
    - *write 0x3C to I2C_SA: 0x4802 A0AC*

wait 40 ms

Initialize the interrupt controller
- *write 0x80 to INTC_MIR_CLEAR2: 0x4820 00C8*

Initialize interrupts in the processor
Enable NACK and Bus Free I2C interrupts
- *write 0x106 to I2C_IRQENABLE_SET: 0x4802 A02C*

Set ARDY flag to get transfer started
- *write 0x4 to I2C_IRQSTATUS_RAW: 0x4802 A024*

MAIN:
WHILE FOREVER
    Do nothing

INT DIRECTOR:
check that I2C was source of interrupt
- *read 0x80 from INTC_PENDING_IRQ2: 0x4820 00D8*

IF I2C was source of interrupt
    go to I2C interrupt handler
ELSE
    reset the interrupt controller
    exit IRQ mode

I2C INTERRUPT HANDLER:
check if NACK response was received
- *read 0x2 from I2C_IRQ_STATUS: 0x4802 A028*

IF NACK was received
    go to NACK handler
(ELSE it was the bus free or ARDY interrupt)
clear the Bus Free interrupt
- *write 0x4 to I2C_IRQ_STATUS: 0x4802 A028*

check if an initialization byte needs to be sent
IF initialization byte needs to be sent
    get address for initialization bytes
    get address for sequence offset
    load control byte
    go to I2C TX SETUP
ELSE we are sending character bytes
    check if message byte needs to be sent
    IF message byte needs to be sent

get address for character message
get address for sequence offset
load control byte
go to I2C TX SETUP
ELSE transfer is finished
mask I2C interrupts
- write 0x12 to I2C_IRQENABLE_CLEAR: 0x4802 A030
disable I2C interrupts
- write 0x08 to INTC_MIR_SET2: 0x4802 00CC
reset interrupt controller
exit IRQ mode

I2C NACK HANDLER:
check if we are doing an initialization
IF we are doing any initialization
load initialization bytes
load sequence offset
ELSE we are sending message bytes
load message bytes
load sequence offset
go to previous byte
decrement offset
go to I2C TX SETUP

I2C TX SETUP:
RO = Offset address
R1 = Stream address
get the next byte to transmit
load the control byte into the FIFO
- write data to I2C_DATA: 0x4802 A09C
load the data byte into the FIFO
- write data to I2C_DATA: 0x4802 A09C
increment the stream offset

I2C TX BEGIN:
update the count register
- write 0x2 to I2C_COUNT: 0x4802 A098
set module to master mode
- write 0x400 to I2C_CON: 0x4802 A0A4
configure the start and stop bits to begin transfer
- write 0x3 to I2C_CON: 0x4802 A0A4
reset interrupt controller
return from IRQ mode

DELAY:
Load countdown value
WHILE countdown > 0
    decrement value

H. Project Code

See attached.

Appendix

I. Supporting Documentation

Pin Connections

Pin	BeagleBone	AM3358	Name	Mode
I2C1_SCL	P9: pin 17	A16	spi0_cs0	2
I2C1_SDA	P8: pin 18	B16	csi0_d1	2

I2C Initialization

See AM3358 Technical Reference Manual **21.3.15:** How to Program I2C and **21.3.151.2:** Initialization Procedure.

1. Set I2C Internal clock to 12 MHz
   48 MHz bus clock per Hall instructions
   Prescaler = 48 / 12 = 4 = 0x100

2. Set I2C data rate to 100 Kbps

   ST7036 Requirements: LOW = 160 ns (min), HIGH = 20 ns (min)
   Assuming 1 bit per cycle, 100 Kbps = 100 kHz.
   100 kHz = 10us period.
   At 50% duty cycle, 100 kHz= 5us HIGH time and 5us LOW time.

   SCLL:
   See p. 4637 for SCLL (SCL Low Time)
   Low Time: t_{LOW} = (SCLL + 7) * ICLK time period

$$SCLL = \frac{t_{LOW}}{ICLK_{period}} - 7 = \frac{5\ us}{83.3\ ns} - 7 = 60 - 7 = 53 = 0x35$$

   SCLH:
   See p. 4638 for SCLH (SCL High Time)
   High Time: t_{HIGH} = (SCLH + 5) * ICLK time period

$$SCLL = \frac{t_{HIGH}}{ICLK_{period}} - 5 = \frac{5\ us}{83.3\ ns} - 5 = 60 - 5 = 55 = 0x37$$

3. Configure own address.
   Not necessary since the I2C is in master transmitter mode only for this project.

4. Take I2C module out of reset.
   - Bit 15: I2C Module enable

5. Configure the I2C mode register
   - Bit 10: Enable master transmitter mode
   - Bit 9: Enable transmitter mode

I2C Transmission

See AM3358 Technical Reference Manual **21.3.15.4:** Initiate Transfer and **21.3.15.6:** Transmit Data.

1. **Check that the bus is ready**
   I2C_IRQSTATUS_RAW: BF flag indicates if bus free or not. This read only bit is set to 1 by the device when the I2C bus became free (after a transfer is ended on the bus stop condition detected). See p. 4603 for more information.

2. **Initialize DCOUNT register with number of bytes to send**

   I2C_DCOUNT: This 16-bit countdown counter decrements by 1 for every byte received or sent through the I2C interface. A write initializes DCOUNT to a saved initial value. When DCOUNT reaches 0, the core generates a stop condition if a stop condition was specified (I2C_CON.STP = 1) and the ARDY status flag is set to 1 in the I2C_IRQSTATUS_RAW register. Note that DCOUNT must not be reconfigured after I2C_CON.STT was enabled and before ARDY is received. See p. 4629 for more information.

3. **Write data to FIFO register**
   I2C_DATA: Transmit/Receive data FIFO endpoint. When read, this register contains the received I2C data. When written, this register contains the byte value to transmit over the I2C data.

4. **Set module Master Mode**

   I2C_CON [MST]: Master/slave mode (I2C mode only). When this bit is cleared, the I2C controller is in the slave mode and the serial clock (SCL) is received from the master device. When this bit is set, the I2C controller is in the master mode and generates the serial clock. Note: This bit is automatically cleared at the end of the transfer on a detected stop condition, in case of arbitration lost or when the module is configured as a master but addressed as a slave by an external master. Value after reset is low.0h = Slave mode. 1h = Master mode. See p. 4631-4633 for more information.

5. **Configure Start and Stop bits**

   I2C_CON [STP]: (I2C master mode only). This bit can be set to a 1 by the CPU to generate a stop condition. It is reset to 0 by the hardware after the stop condition has been generated. The

stop condition is generated when DCOUNT passes 0. When this bit is not set to 1 before the end of the transfer (DCOUNT= 0), the stop condition is not generated and the SCL line is hold to 0 by the master, which can re-start a new transfer by setting the STT bit to 1. Value after reset is low. 0h = No action or stop condition detected. 1h = Stop condition queried. See p. 4631-4633 for more information.

I2C_CON [STT]: Start condition (I2C master mode only). This bit can be set to a 1 by the CPU to generate a start condition. It is reset to 0 by the hardware after the start condition has been generated. The start/stop bits can be configured to generate different transfer formats. Value after reset is low. STT = 1, STP = 1, Conditions = Start-Stop (DCOUNT=n), Bus Activities = S-A-D..(n)..D-P. 0h = No action or start condition detected. 1h = Start condition queried. See p. 4631-4633 for more information.

New Haven Display Initialization

Note that the display requires a specific amount of time to process commands so the polling mode allows for a delay between instructions. The interrupt mode sends commands at a much faster rate. It appears that the time needed to process commands is actually quite small and can be done much faster. The polling mode could be done without the wait period but for this purpose it is acceptable.
See p.11 of the New Haven Display datasheet.

Command Instructions vs. Data Instructions

See p.16 of ST7036 datasheet.

A command word consists of a control byte, which defines Co and RS, plus a data byte. The last control byte is tagged with a cleared most significant bit (i.e. the continuation bit Co). After a control byte with a cleared Co bit, only data bytes will follow. The state of the RS bit defines whether the data byte is interpreted as a command or as RAM data. All addressed slaves on the bus also acknowledge the control and data bytes. After the last control byte, depending on the RS bit setting; either a series of display data bytes or command data bytes may follow. If the RS bit is set to logic 1, The data pointer is automatically updated and the data is directed to the intended ST7036i device. If the RS bit of the last control byte is set to logic 0, these Only the addressed slave makes the acknowledgement after each byte. At the end of the transmission the I2C

RS = 0: command bytes will be decoded and the setting of the device will be changed according to the received commands.
RS = 1: these display bytes are stored in the display RAM at the address specified by the data pointer.

Co = 0: Last control byte to be sent. Only a stream of data bytes is allowed to follow.
Co = 1: Another control byte will follow the data byte unless a STOP condition is received.

Initialization Sequence

1. Function Set Command: 0x38
   - Bus mode = 8-bits, Display mode = 2 line, Display font = normal, Instruction set = normal

2. Function Set Command: 0x39
   - Bus mode = 8-bits, Display mode = 2 line, Display font = normal, Instruction set = extended
3. Bias Set Command: 0x14
   - Bias = 1/5
4. Contrast Set Command
   - Contrast bit C3 = 1
5. Power/ICON/Control Command
   - ICON display = on, Booster circuit = on, C5 contrast bit = 1
6. Follower Control Command
   - Follower circuit = on, Internal follower = 0x5
7. Display ON/OFF Command
   - Display = ON, Cursor = OFF, Blink = OFF
8. Return Home

Command byte: 0x00

Data bytes: 0x38, 0x39, 0x14, 0x78, 0x5E, 0x6D, 0x6D, 0x0C, 0x01, 0x06, 0x02

J. Design Log

2/23/18:

- Gathered I2C information from reference manual and project guidelines
- Pin connections for I2C2 SDA and SCL
  I2C1_SCL = A16 (spi0_cs0) = mode 2
  I2C1_SDA = B16 (spi0_d1) = mode 2
  Control module: 0x44E1_00000
  spi0_cs0 = 0x95C: write 0x11 to [0:2] mode bits
  spi0_d1 = 0x958: write 0x11 to [0:2] mode bits
- ST0736
  Slave Address: 0111 1100 = 0x7C
  COMMANDS
  Display ON/OFF (3 bytes): 0x7C, 0x00, 0x0F
  Clear Display (3 bytes): 0x7C, 0x00, 0x01
  CHECK BUSY FLAG
  - have to wait until BUSY flag is low
  RS = LOW, RW = HIGH

2/24/18:

- Further research on the communication protocol for the ST0736. Unsure about the whole procedure they outline in the initialization procedure and if they are setting all the different parts of the counter and address and all that.

- Assuming for now that all we need to get going is to turn on the display and clear the display before writing text to the display.
- Created a template for the string of bytes to send for the different commands. Even though we could stack commands together in the 80-bytes of the RAM, going to just do things one at a time.
- Typical turn around on the display is in the microseconds range for it to process data so set the typical wait period for 10 ms. Typical protocol for part 1 will be to send the command/data and go into a while loop that polls the display until it is ready for another command. (make this a routine)
- Back of ST7036 actually has the initialization procedure. Going to update with that wait 40 ms.
- Researching initialization functions and initialization sequence for the display. Datasheet shows the initialization sequence but not sure what to set all the bits to. Going to have to email Charles to see what he knows about how the display is set up. There is a sample initialization sequence in the datasheet but not sure if that is related or not.
- Research for I2C initialization procedure:
  Set I2C clock at 12 MHz:
  48 MHz source bus clock per Hall instructions
  I2C_PSC = 0xB0 = I2C Prescaler Register
  48 / 12 = 4 = 0x100
  ICLCK = Internal I2C clock = 12 MHz
  Write 0x100 to [0:7] PSC bits
  Set I2C data rate to 100Kbps
  ST7036 Requirements: LOW = 160 ns (min), HIGH = 20 ns (min)
  100 Kbps = 10 us period. @ 50% duty cycle = 5 us HIGH and 5 us LOW
  I2C_SCLL = 0xB4 = SCL Low Time Register
  Write 53d to SCLL
  I2C_SCLH = 0xB8 = SCL High Time Register
  Write 55d to SCLH
- I2C Configuration Register:
  Enable Module, Fast/Standard Mode, enable start byte, master mode, transmitter mode, slave = 7-bit address mode, stop condition enabled, start condition enabled
- I2C Slave Address Register:
  ST7036 = 0x3E
  I2C_DATA = 0x9C = Data Register for number of bytes to transmit
  - Write number of bytes to [7:0]

3/2/18:

- Wrote the first draft of the algorithm and code for part 1.

3/3/18:

- Debugging code. I can initiate transfers now but when I sent the slave address, the slave responds with NACK signal. Polling the XRDY flag gets stuck in an infinite loop after transfer. According to the FIFO management section, I think this means that we haven't sent the right amount of bytes for XRDY to go back to 0.

- Slave address should be 0x7C but the display is not responding to the slave address. Going to switch to 10-bit address to see if that works. I2C_CON.XSA = 1. Changing the 10-bit address did not work. Looking into whether the pins need to be explicitly configured as open drain.
- Apparently somewhere in the documentation for the LCD game it is said that the slave address is 0x3C. This is not what the datasheet indicates. Will have to ask why this is.
- Per the logic analyzer, I am now sending all my data and it is getting acknoledged appropriately but nothing is displaying on the screen. Will have to look into the initialization procedure for the display and figure what is going wrong.
- I am sending the slave byte explicitly. I can get rid of that.
- Everything seems to be going good and the data is being sent but nothing is displaying on the screen. I tried sending the byte streams associated the display initialization code and it works now. I don't really understand that bytes that they are sending or what they are initializing but following the code works.
- Working on part 2 now. Handshake signals that we need.
  INTC.MIR2 = 0x4820 00C4
  MIR_CLEAR2 = 0xC8
  MIR_SET2 = 0xCC
  I2C1 = Interrupt 71 = Bit 7 in MIR2
- Setting up part 2.
- Initialization procedure is going to load the count register and the FIFO and then enable the busy flag register to begin transmission. Inside of the INT DIRECTOR you check if the count register is greater than zero. If the count is greater than zero, then there is data to send and you should initate the transfer with the start and stop bits. If the count register is empty, then mask the interrupt to cancel the transfer.
- For part 2, I think this can be done without interrupt handshaking. I'm going to set up a timer to give the display time to process the instructions. I am also setting it up so that I have one continues stream of bytes and instead send them in three groups.

3/4/18:

- I am experiencing some weird behavior when doing a first run after a hard reset. The instructions being sent are not going through all the way and the display returns a NACK response when sending the 8 byte string. Ideally, I would have to set something up to handle the NACK response and attempt a resend but for this project, I just split the 8 bytes into two different groups and works great on first run after a hard reset. Breaking the initialization string into smaller groups gets it working correctly.
- I finished part 2 by implement a timer to wait the period of time required to send the commands to the display. I'm not really sure what the instructions mean by saying to implement the handshaking with interrupts because it can be done without interrupts. I e-mailed Charles so we will see what he says.

3/7/18:

- After talking with Charles I don’t think I am doing this right and I need to make use of the I2C interrupts. Getting rid of the timer is what I need to do. I think that I can use the XRDY bit to use as a signal for when to send the next byte.

- Having problems initiating transfers using the I2C interrupts. The AERR bit continues to be a problem and is the only interrupt that occurs. Even after I clear it, it becomes enabled again as soon as I enter IRQ mode.
- XRDY is not what I thought it was. XRDY tells you if the module is ready to send data once it has been loaded past the threshold. It doesn't tell you when the bus is necessarily ready. Switching to the bus free flag would be better and we can use that to signal when the transfer is complete.
- I figured out the interrupts. You need to at least initialize the transfer once to get things going. XRDY is not really what I thought it was. XRDY lets you know that the FIFO has data and it is ready for transmission. So, the new scheme is that I get the transfer going by setting ARDY in the IRQ RAW register to trigger the interrupt. And from there use the bus free flag to refill the FIFO when the transmission is complete.
- Last problem now is that the first letter of my name is not printing on the screen. I looked at the logic analyzer and it looks like the characters are being sent and received properly but for some reason this letter is getting cut off.
- I think the problem was that I needed to send a RETURN HOME instruction to the display before sending the display data. Its hard to tell because code composer constantly crashes.
- I got it working now with the RETURN HOME instruction at the end of the initialization sequence. All finished.

3/16/18:

- Did the final demo today and I was still experiencing issues with the letters not fully getting displayed. For some reason the last character doesn't get printed some of the time. Charles mentioned that I had the prescaler set up wrong and I did not account for the value + 1 in the register. I changed it and it worked so it is possible that this was the problem.

K. Signed Statement

I developed this program on my own with no help from anyone but the teacher and the TA and I did not give any help to anyone else.

Ryan Bentz