2/25/2021

- I am now starting part one of the project.
  - I am reading the documentation for part one of the project.
  - Configuring P9 on B3 for I2C (**Control Module** base: 0x44E1_0000)
    - **I2C2_SCL pin 19**
      - Mode0: uart1_rtsn (offset: 97Ch)
      - Need Mode3: I2C2_SCL
      - So RMW with 0x33 to offset
    - **I2C2_SDA pin 20**
      - Mode0: uart1_ctsn (offset: 978h)
      - Need Mode3: I2C2_SDA
      - So RMW with 0x33 to offset
  - I2C clock configuration (CM_PER base: 0x44E0_0000)
    - **CM_PER_I2C2_CLKCTRL** (offset: 44h)
      - Write 0x2 to enable

2/26/2021

- I am continuing to follow the project 2 part 1 guide whilst researching the AM335x manual (gathering the needed registers by following the I2C procedure in the manual)
  - I2C module configuration
    - Need 12MHz (ICLK) so divide 48MHz (SCLK) module clock with prescaler register (I2C_PSC)
      - Need 48 divided by 4. Following the register equation (PSC + 1) we load 3 to the register to get that 4.
        - Write #0x3 to **I2C_PSC** (offset: B0h)
    - Want 400Kbps.
      - I need to set SCLL and SCLH to some value given the equations from the manual. However, I am not sure what values to plug in for both tLOW and tHIGH.
      - Register equations are
        - SCLL: tLOW = (SCLL + 7) * ICLK
        - SCLH: tHIGH = (SCLH + 5) * ICLK
    - Configure its own address
      - Write #0x40 to **I2C_OA** (offset: A8h)
        - I think I can set it to whatever I want except 0x60 (slave address)
    - Take I2C out of reset
      - Write 0x8000 to **I2C_CON** (offset: A4h)
  - I2C Initialization procedure information
    - Configure I2C mode register
      - RMW 0x600 with **I2C_CON** (offset: A4h)
        - Set to master transmitter (master receiver would have TRX equal to zero instead)
        - **I believe this is what edits the $R/\overline{W}$ bit in I2C transmission**

            - Or just write 0x8600
        - Skipping steps 2 and 3 for now since I am not installing interrupts and not using the DMA
    - Configure slave address and DATA counter
        - Slave address will be 0x60
            - Write 0x60 to **I2C_SA** (offset: ACh)
        - Normally, there would be at least a 1 byte data transfer
            - However, for part one we are only transmitting the address of the slave. Therefore, load 0 to register. Load 1 or more to register for compete I2C transmission. **<u>Load 1 for part two of this project.</u>**
            - Write 0x1 to **I2C_CNT** (offset: 98h)
    - Initiate a transfer
        - Poll bit 12 of **I2C_IRQSTATUS_RAW** (offset: 24h) to see if line is busy
            - If 0, configure start/stop. I want I2C activity to be S-A-D..(n)..D-P so both bits need to be set.
                - Start: bit 0 of **I2C_CON** (offset: A4h)
                - Stop: bit 1 of **I2C_CON** (offset: A4h)
                    - RMW #0x3 into **I2C_CON** (offset: A4h)
- For this project, I will only be transmitting data. So we can skip the receive data portion of the procedure altogether. Also, for part one of this project we are sending just the slave address, not actual data. So **<u>this next part is for part two of this project</u>** and included for completeness of the I2C transmission.
    - Transmit data
        - Poll XRDY (bit 4) in **I2C_IRQSTATUS_RAW** (offset: 24h)
            - If 1, send next byte to **I2C_DATA** (offset: 9Ch)
- I noticed that interrupt generation and handling will add much more registers/complexity to the procedure above. The polling method is mostly ready since most of the interrupt registers are initially disabled.

2/27/2021

- I am now starting to write my high level algorithm for part one
- I received help for tLOW and tHIGH from Tyler
    - 400KHz gets a period of 2.5μs. A desired duty cycle of 50% gets tLOW=tHIGH=1.25μs. ICLK is the period of the 12Mhz clock.
    - Solving for SCLL and SCLH with the equations above
        - Write 0x08 to **I2C_SCLL** (offset: B4h)
        - Write 0x0A to **I2C_SCLH** (offset: B8h)

2/28/2021

- The high level algorithm is finished and I am moving onto the low level algorithm for part one
- I have finished the low level algorithm and moving on to writing the assembly code. I will convert it into C afterwards.
- I have finished writing the code both in C and ASM.
    - Used past ASM programs and the C program tutorial handout as guides

3/1/2021

- I am now setting up my workstation (B3, logic analyzer, etc)
    - Learning how to work the logic analyzer with the correct software/setup
        - Had to use Zadig to get a driver for this logic analyzer.
        - Using PulseView for reading the logic analyzer
- Debugging/testing will also be done today.
    - First run through of the program, I am not getting any results from the logic analyzer.

3/13/2021

- I am starting to read the documentation for project 2 part 2
    - Familiarizing myself with the motor
    - Familiarizing myself with the Adafruit feather motor controller
        - Along with the NXP PCA 9685
    - Gathered all necessary datasheets and put them in the project folder for future reference. (Along with the B3 and AM 335x ref manuals from previous projects)

3/15/2021

- I am now setting up the high level algorithm
- I am now setting up the low level algorithm

3/18/2021

- I am now writing the code to the point in the low level algorithm
- I still have to gather the data that is needed to transfer over I2C
    - I have made a chart that tells me what switches need to be turned/off in the motor driver
    - The chart also tells me the ledn_on signals I need.
    - I structured and created two arrays for the data to be sent.
- I am now setting up the motor circuit

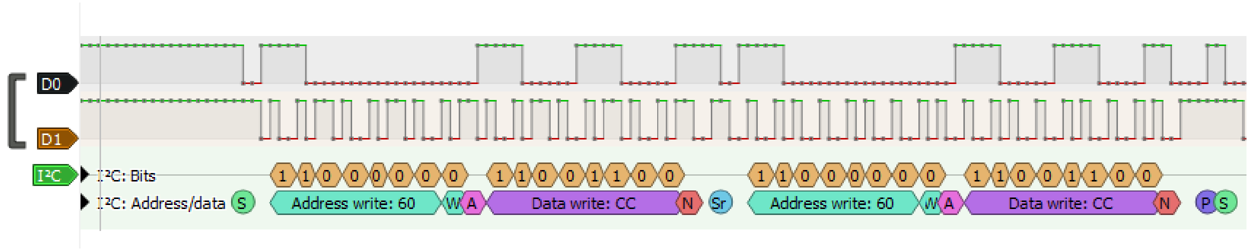
3/19/2021

- I have confirmed my data with Tyler and started debugging
- I figured out (using the analyzer) that the data isn't being sent at all which was also the problem with part 1.
- **Upon further debugging and help from Tyler, we figured out that with certain edits to the code data will be sent. The big three edits were:**
    - Add stack
    - <u>**Convert ORing into just equals**</u>
        - This made me convert each start/stop configuration into it's full register value 0x8603

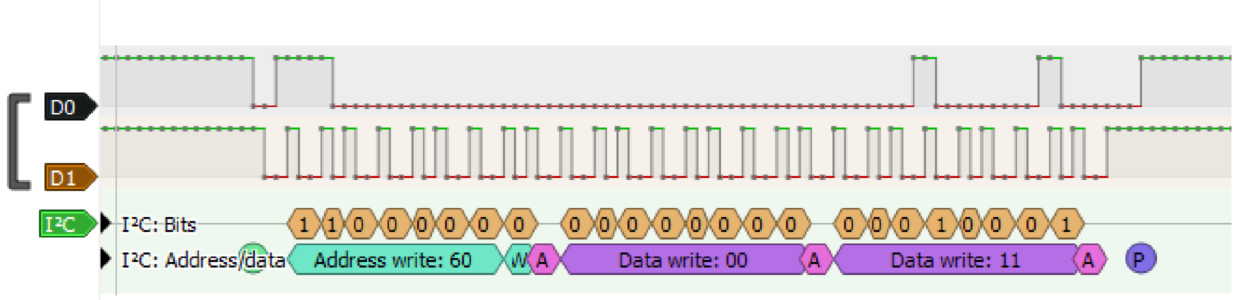
3/21/2021

- I have finished the programs for polling in part one and interrupts for part 2 with delay.
    - There is still a over current issue
        - I looped the initialization of the motor board to keep stepping. The board still turns off in between which adds an extra undesired delay
    - <u>**Upon further investigation, I found that the I2C originally wasn't working because of the ORing (adding stacks is unnecessary).**</u>

- Here is the analyzer plot for part one:



- Part two plot looks similar. However, each transmission has a delay with the necessary data (2 bytes instead of 1).



- Final findings:
  - In the interrupt code
    - Global initialized variables did not hold value throughout the program. At some point the variables have some extreme value that is unexplainable.
    - Timer interrupts were dropped because of the issue above. Adding timers at this point would just cause further problems. Also, there is that debugging issue when using timers which would cause long debugging sessions and confusion.
  - The polling method works fine.