**3/10/2021**

For this part 1 of the project, I first read through the project guide first to get an ideas on how to do the project and follow it steps. Step 1 of the project guide, I search up what pin that the I2C2\_SCL and I2C2\_SDA will be in google by searching for the beagle bone pin out. Then I found what I’m looking for which the SCL and SDA is in pin 19 and 20 respectively, I also notice that the pin going to need to be in mode 3 for both pin.

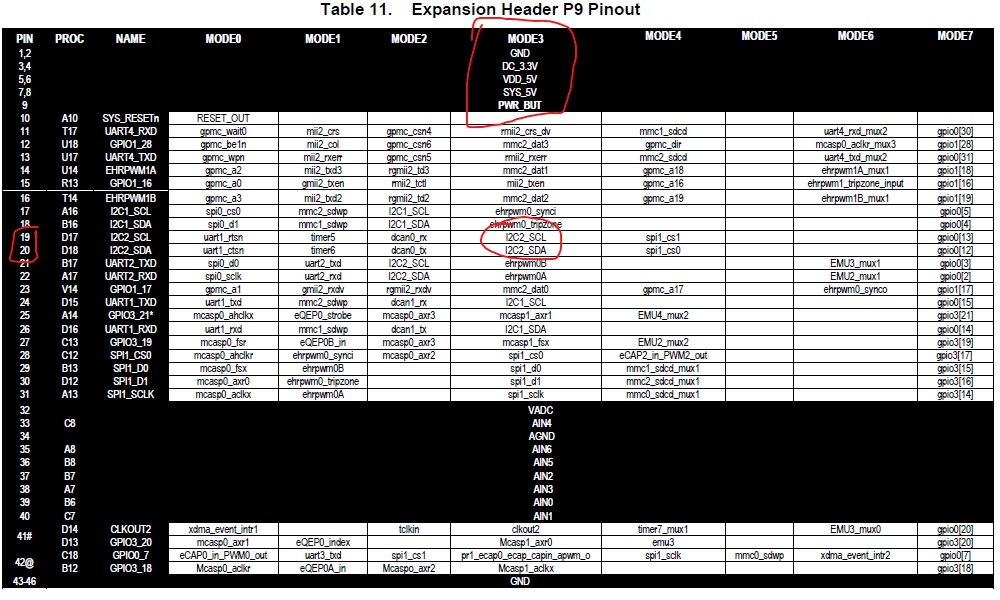
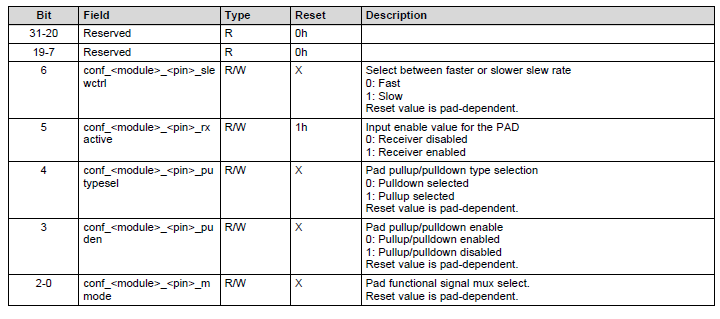


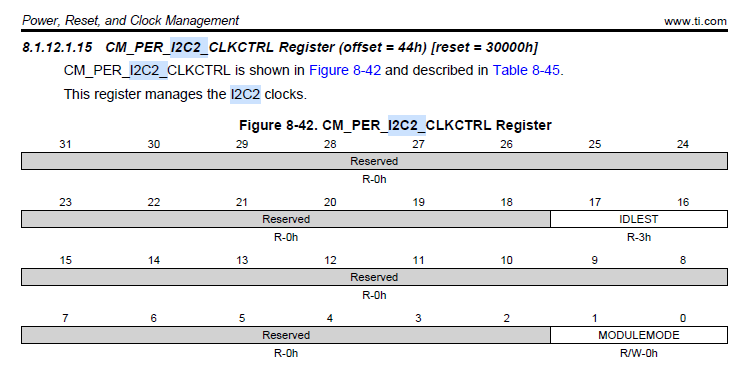
Table 11. P9 Pinout

So what I do next is to find out a way to switch the pin in 19 and 20 to mode 3 since by default both pin are in mode 0. I would need to find first need to find the control module base address which is 0x44E10000, then I need to find what is the register and offset of both P19/P20 to start switching them to mode 3. On table 11 picture it have a mode 0 column which tell you what register that both of the pin is which is UART1\_RTSN and UART1\_CTSN, so next I look for them both in the TI manual by Ctrl + F and type in the name of the register.

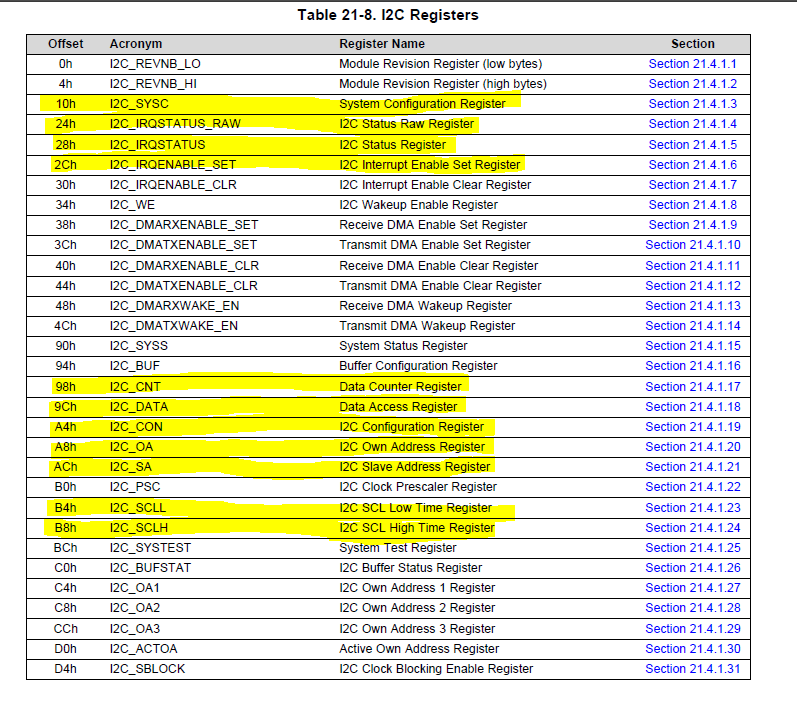


Once I got the offset of both registers, I look through the field description of both register and see that they both have the same description.

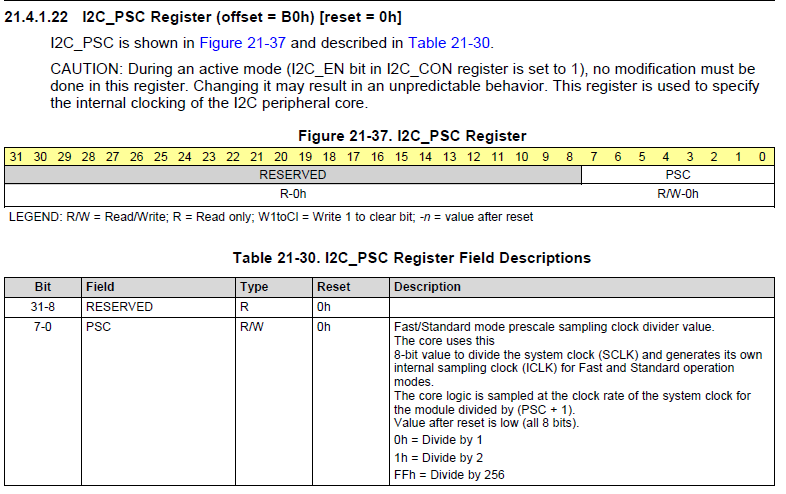


Since I want to have mode 3 which can be configurate using bit 0-2 that is used to select the mode. I want to have it to mode 3, have Pullup/Pulldown enable and enable Pullup then I want to have the receiver enable. Using the bit description field, I have determined that the value I need to read into both register is 0x33 to make them switch to mode 3, enable Pullup and receiver enabled. Step 2 project guide, I find the offset address of I2C2 CM\_PER\_CLK in the TI manual which is 0x44h in the manual. 

I just have to add this offset to the base address of CM\_PER which is 0x44E00000 which I found in the L4\_WKUP in the Hall Chapter 1-5 textbook. Then I just read in 0x02 to the address to wake up I2C2 clock control to enable I2C2, and I also go and look for the base address of I2C2 since it is needed for the project which the base address of I2C2 is 0x4819C000. After I got the I2C2 clock turn on, now we have to configurate the I2C2 clock scale for the SCLL and SCLH. I need to scale the clock down from 48 MHz down to 12 MHz, and SCLL/SCLH to have a 400 Kbps. I first read Step4 in the guide where it tell me to go to section 21.3.15 in the TI manual which I go to and see section 21.3.15.1 which talk about how to program I2C pre-scale which it tell me which register to go for scaling down the clock and the register is I2C\_PSC. I proceed to skim through the I2C configuration of the TI manual and then I see that there a I2C register table at section 21.4.1. So before anything else I went though the I2C configure section and highlight all the register that I’m going to be need so I can have them all in one place later to look for.



With all the register that I needed highlight, I go back and continue reading section 21.3.15.1 of pre-scale the I2C. I use the 21-9 table in the TI manual and look for the I2C Clock Pre-scaler Register which is 0xB0 then I click on the section of that register in the table to have it take me to the register description.



I read the register field descriptions and see that in order to scale the clock we just have to read in the value we want to divide 48 MHz to using bit 0-7. So, Since I want the clock to operate at 12 MHz that mean 48 MHz must be divide by 4 in order for me to obtain 12 MHz, that mean I just have to read in 0x03 into the offset of B0 of the I2C2. After I got the clock to scale down to 12 MHz the next things to do is to program the I2C2\_SCLL and I2C2\_SCLH to have 400Kps. I use table 21-8 again and go to the I2C\_SCLL and I2C\_SCLH section to read about them.

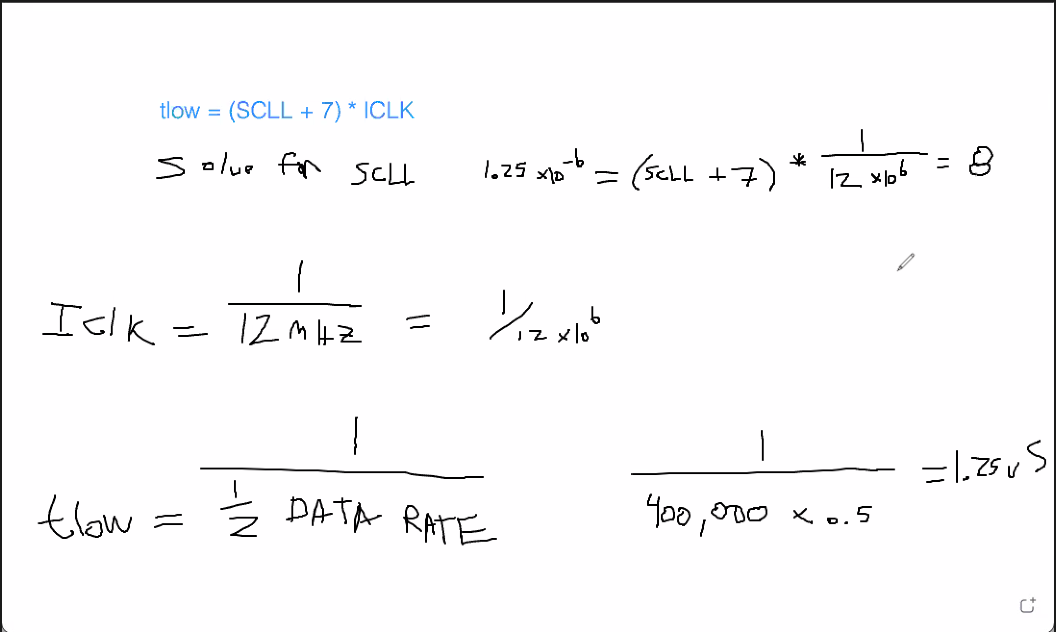


I first go to I2C\_SCLL section first and read through the field description then I see that there a formula on how to find the value for SCLL. I was confused on how to even solve this equation since I do not know how to find tLOW and ICLK which the TI manual do not talk about it. I went to the I2C\_SCLH section to see if there anything I can gain from looking there.

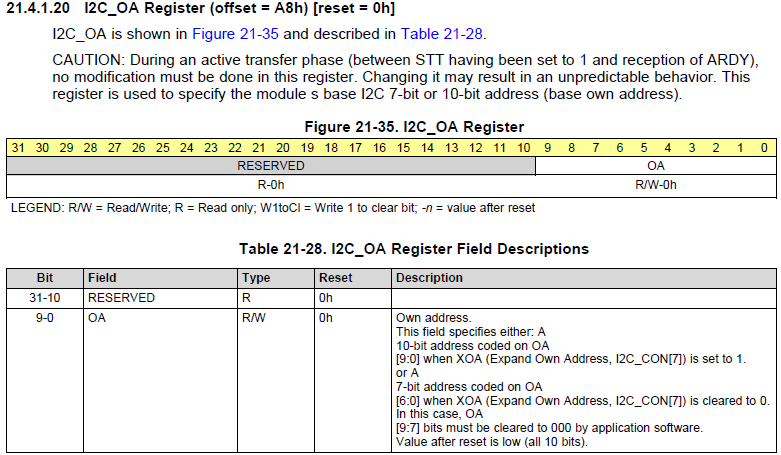


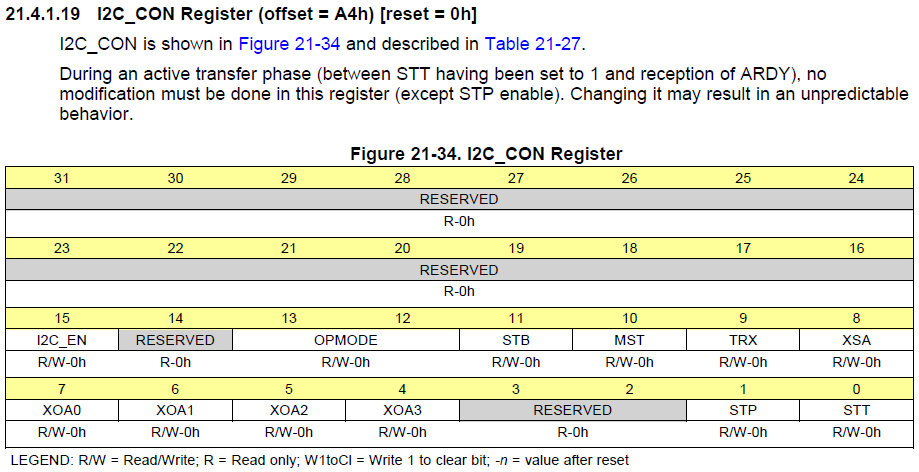
Looking at the field description and I see that there nothing I can gain from this SCLH since I don’t know what tHIGH and ICLK is. So I decide that I will wait for tomorrow and go to the lab session to ask the TA about the equation on how to solve them.

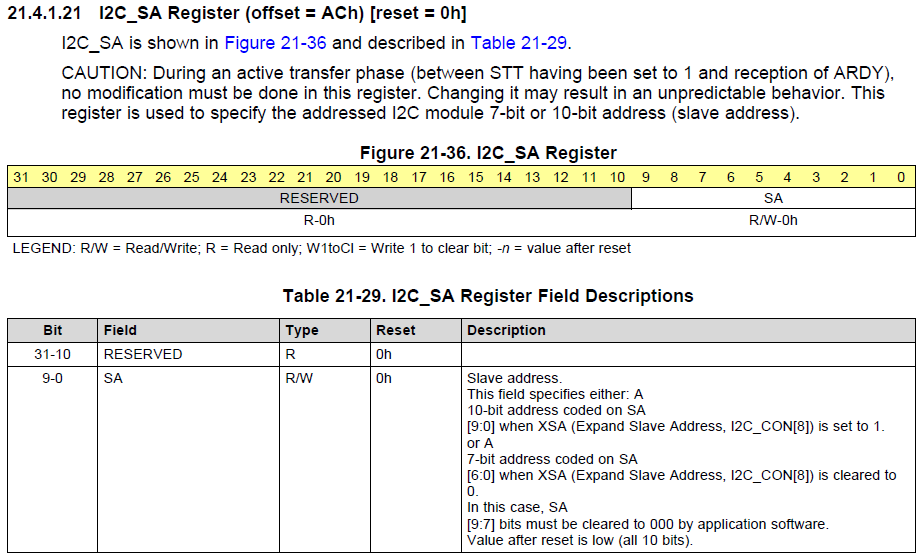
**3/11/2021**

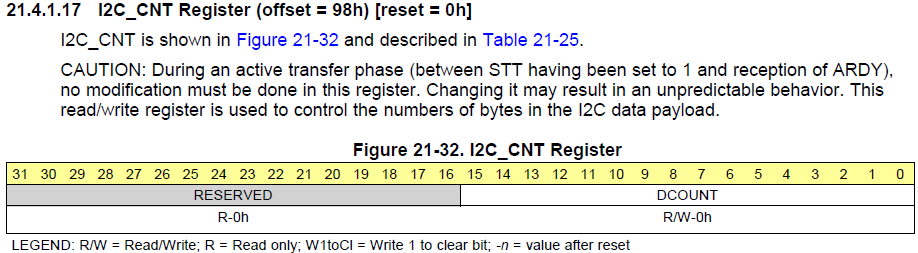
After the lab session I now know how to find tLOW, tHIGH and the ICLK, so tLOW/tHIGH are equal so when you find one then you know the other one. To find tLOW/tHIGH, you just have to which you will end up with 1.25x10^-6 for both tLOW and tHIGH. Then for the ICLK all you need to do is take 1 divide by the internal clock rate that you set which in this case I set it to 12 MHz, so it will be 1/12MHz. 

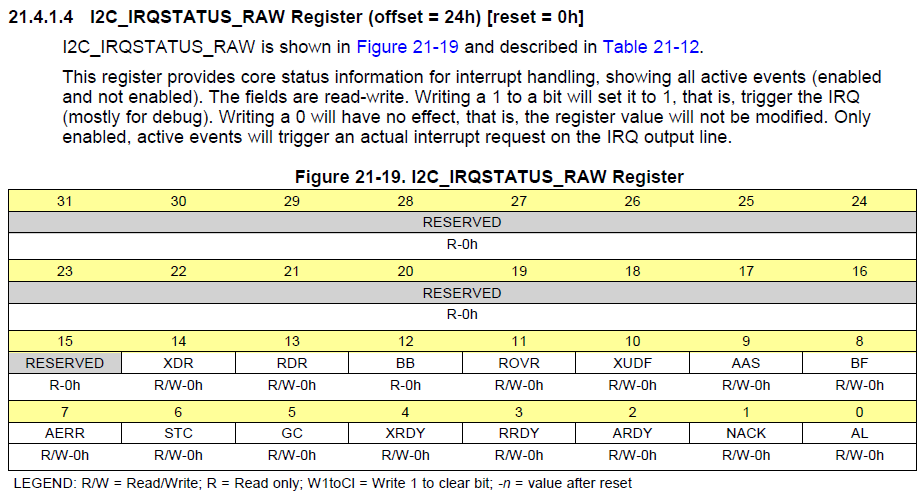
With the information I just use substitute and then solve for SCLL which end up to be 8 which I convert it to hex number to read into the register and it turn out to be 0x08 as the value. Then with the SCLH I do the same thing to it using the same number with . I convert it into hex so that I can read it into the SCLH register which turn out to be 0x0A. I continue to follow the TI manual on how to program I2C, and the next thing I need to do is to do to configurate the I2C own address, I use table 21-8 to go to the I2C\_OA register field description.



With the description field I decide that I will set the I2C own address to 0x00, and read the value to the I2C\_OA register. After the own address is set, then I must take the I2C module out of reset with the I2C\_CON register, so once again using the table 21-8 I move to the section that talk about the I2C\_CON.

So in order to take the I2C out of reset we have to enable the I2C\_EN bit that is in the I2C\_CON which is the 15th bit by make it to be a 1, then we also want to have it to be in master mode and transmission mode to send the data. To make it in master mode I must write to bit 10th and for transmission mode bit 9th, with all the bit I need to write to the I2C\_CON register after the conversion I have to write 0x8600 to the I2C\_CON register to take it out of reset. After that I keep reading the procedure in the TI manual which currently I am on initialization procedure, it tell me to configurate the I2C\_CON bit which I have already done and enable the interrupt. Since I read in the project guide that we are not using the I2C\_DMA/RX/TX, so I decide that I will be skipping this section because it wants me to enable and set up interrupt which we won’t be using. The next step is to configurate the slave address and set how many byte we going to send out, so that the I2C know where to send the data to when it starting the transmission. I first go the field bit description of the slave register to read what each bit do, and it look very similar to the OA register. 

The project guide already mention that it want us to use 0x60 as the slave address, so I just need to read that value into the I2C\_SA register to config the slave address. Then after that I have to set the count to 2 byte only since that how much data the PCA can read at the time. 

So I write 0x02 into the I2C\_CNT register to have it only send out 2 byte at the time in order to no overload the PCA. With all the set up and initialize is done for the I2C, the next step that I need to do is start initiate a transfer with the PCA, and to do this I have to poll the BB (Bus Busy) bit in the I2C status register. I have to check whether or not it is clear to 0 which mean bus is not busy, otherwise I have to keep on checking. To do the checking I have to read in the value that is being store in the I2C\_IRQSTATUS\_RAW register that the TI manual tell me to and test bit 12th which is the BB bit. 

After I have check if the bus is not busy then, I have to configurate the START/STOP bit in the I2C\_CON register. They are the 1st and 2nd bit which is the STT and STP bit for transferring. So I read in 0x8603 since I do want to keep the I2C in master mode and transmission mode for the data, after that I read in 0x02 into the I2C\_CNT register to have the I2C know that it will be reading in 2 byte only. Once I have that done I check to make sure that the XRDY bit in the I2C\_IRQSTATUS\_RAW register is being set at 1, if it is then I proceed to go to the wait loop to wait for 0.2 seconds before sending out the data. With all that is done I can finally send out my data to the I2C, to send out the data all I must do is write the byte I want to send to the I2C\_DATA register to have it to be send out. Since the project guide don’t say what byte I’m suppose to read out, so I just decide to send out a 0x00 to the data as a test.

**Code For Part 1:**

**.text**

**.global** \_start

**.global** INT\_DIRECTOR

**\_start:**

@@@@@@@@SET UP THE STACK@@@@@@@@@@@@@@@@@@@@@@@@@@@

LDR R13,=STACK1 @Point to base of STACK for svc mod

**ADD** R13,R13,#0x1000 @Point to top of STACK

CPS #0x12 @Switch to IRQ mode

LDR R13,=STACK2 @Point to IRQ mode

**ADD** R13,R13,#0x1000 @Point to top of STACK

CPS #0x13 @Back to SVC mode

@==================================================

@Wake up I2C2 clock

LDR R1, =0x44E00044 @base address of CM\_PER\_I2C2\_CLKTRL

LDR R2, =0x02 @value to wake up the clock

STR R2,[R1] @store the value to the I2C2 clock

@==================================================

@Before enable the I2C2

@Setting up Prescale value for the clock to get 12Mhz

LDR R1, =0x4819C000 @base address of I2C2

**ADD** R2, R1, #0xB0 @offset of I2C2 CLock Prescaler Register

**MOV** R3, #0x3 @setting value to divide by 4

STR R3, [R2] @divide 48Mhz by 4 to obtain 12Mhz

@set up the Low time for I2C2\_SCLL

**ADD** R2, R1, #0xB4 @off set of I2C2\_SCLL register

**MOV** R3, #0x08 @ Value to set it to 400Kps. 8 in decimal and 0x08 in hex

STR R3, [R2] @store the value to I2C2\_SCLL

@set up the High time for I2C2\_SCHL

**ADD** R2, R1, #0xB8 @offset of I2C2\_SCHL register

**MOV** R3, #0x0A @value to set it to 400Kps. 10 in decimal and 0x0A in hex

STR R3, [R2] @store the value to I2C2\_SCHL

@Configure its own address

**ADD** R2, R1, #0xA8 @Address of I2C2\_OA

**MOV** R3, #0x00 @Value to reset it

STR R3, [R2] @store the value to I2C2\_OA

@Take the I2C module out of reset

**ADD** R2, R1, #0xA4 @address of I2C2\_CON

**MOV** R3, #0x8600 @enable the 15 bits to enable the module

STR R3, [R2]

@==================================================

@Initializing and setting up I2C2

@Switching Pin 19 and Pin 20 in BeagleBone to Mode 3 for I2C2\_SCL and I2C2\_SDA

LDR R1, =0x44E1097C @base address of UART\_RSTN

LDR R3, =0x33 @value to switch to mode 3, set the PullUp/PullDown enable and select PullUp, and Enable the receiver

STR R3, [R1] @switch to mode 3

LDR R1, =0x44E10978 @base address of UART\_CTSN

STR R3, [R1] @value to switch to mode 3, set the PullUp/PullDown enable and select PullUp, and Enable the receiver

@====================================================

@ Configure Slave Address and DATA counter register

@setting up DATA counter

LDR R1, =0x4819C000 @base address of I2C2

**ADD** R2, R1, #0x98 @offset of Data counter register

**MOV** R3, #0x02 @value to write 2 characters

STR R2, [R2] @store the value to count

@setting up the slave

**ADD** R2, R1, #0xAC @address of IC2C2 Slave register

LDR R4, [R2] @load the value to modify

**MOV** R5, #0x60 @value to have

STR R5, [R2] @store the value to the slave

@===================================================

**B** SEND

@===================================================

@Sending out data

**SEND:**

BL DELAY

BL POLL\_BB

BL START

BL POLL\_XRDY

BL DELAY

@Send data to I2C2\_DATA

LDR R0,=0x4819C09C

**MOV** R2, #0x00

STR R2, [R0]

BL DELAY

@===================================================

@Bus free for transmiting

**POLL\_BB:**

LDR R1, =0x4819C024 @base address of I2C2\_IRQSTATUS\_RAW

LDR R2, [R1] @load the value in

**CMP** R2, #0x1000 @check if the bit is clear or set

BEQ POLL\_BB @keep Polling if the Bus is busy

**MOV** PC, R14 @return

**POLL\_XRDY:**

LDR R0, =0x4819C024 @base address of I2C2\_IRQSTATUS\_RAW

LDR R2,[R0] @load in the address

TST R2, #0x10 @Check bit 4 for XRDY

BEQ POLL\_XRDY @if it not set then keep Polling

**MOV** PC, R14 @else return

**START:**

LDR R1, =0x4819C0A4 @base address of I2C2\_CON

LDR R2, =0x8603 @Turn on the STT and STP, STT = 1 and STP = 1, Start and STop

STR R2, [R1] @store the value to register

**MOV** PC, R14 @Return

**DELAY:**

LDR R1, =0x00040000 @delay for 0.2 seconds

**WAIT\_LOOP:**

SUBS R1,R1,#1

BNE WAIT\_LOOP

**MOV** PC, R14

**.data**

**.align** 2

**STACK1:** .rept 1024

**.word** 0x0000

.endr

**STACK2:** .rept 1024

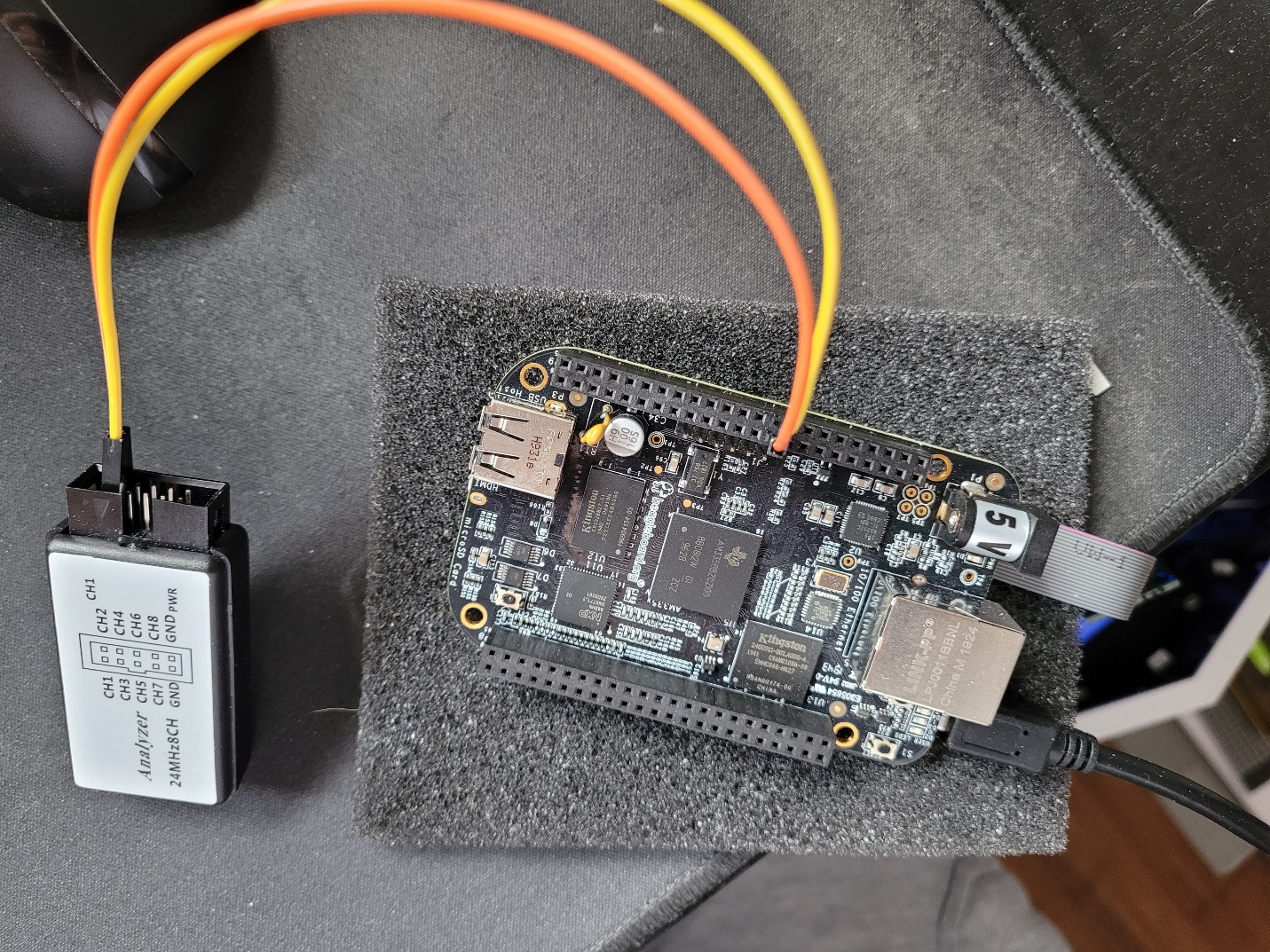
**.word** 0x0000

.endr

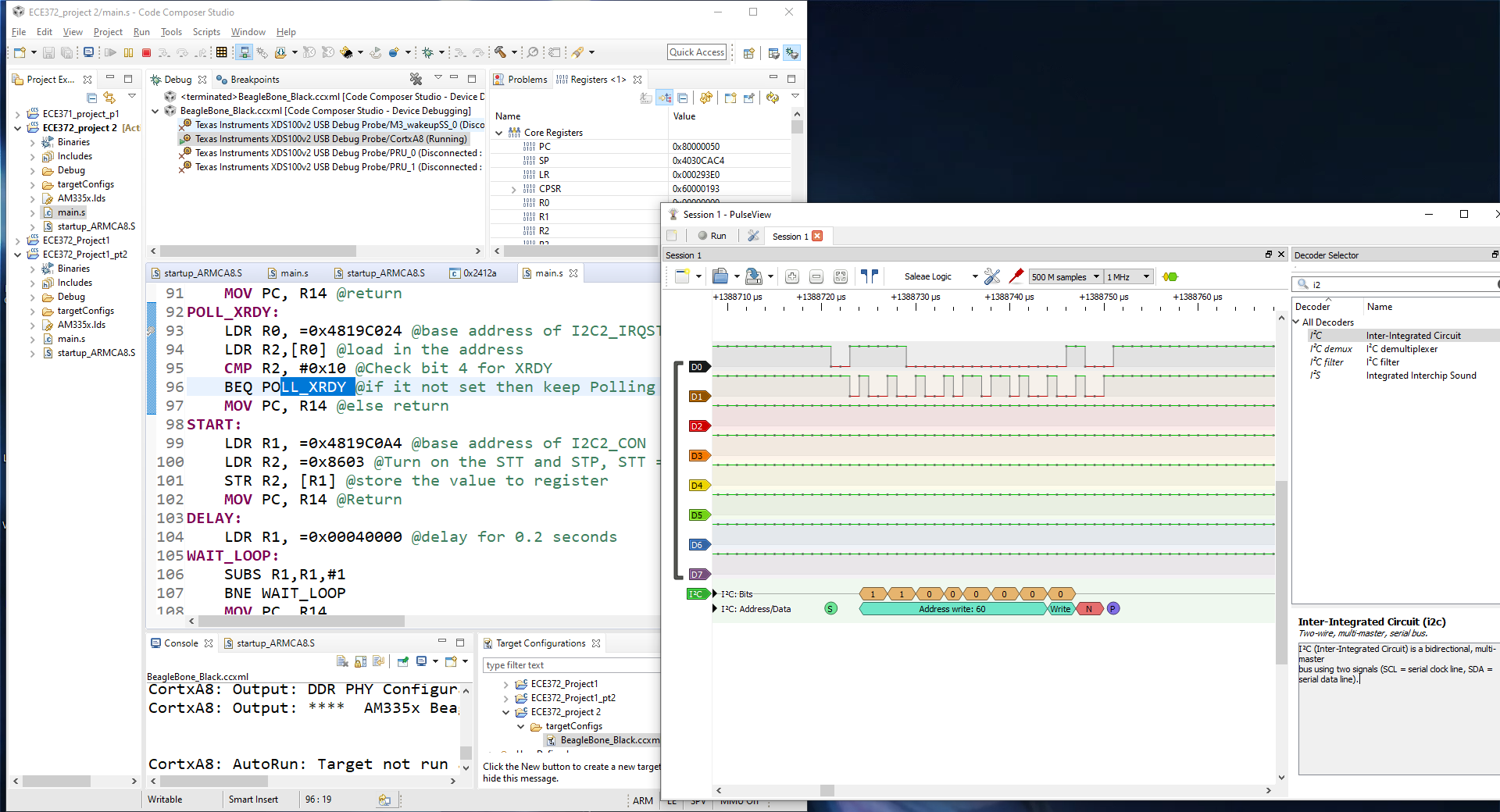
.end

**3/12/2021**

I finish writing up the code yesterday and then start moving on testing phrase which I hook up the BeagleBone Black with the logic analyzer. I hook up CH1 to the SCL and CH2 to SDA which is pin 19 and 20 on the BeagleBone respectively. I use PulseView as the software to look at the wave form, I download it in this [link](https://sigrok.org/wiki/Downloads) and follow the instruction they give to set up the application. I build the code and there is not any error coming from my code, and then plug in the logic analyzer to my computer. When I hit run in the PulseView I have an error of device not plug in, it seems like my computer do not recognize the logic analyzer. So, I go google to see why my PulseView is not being recognize and it seem like it has to do with my USB port. The solution that I see online is I must download a software call Zadig which will make my computer to recognize the USB from my logic analyzer. When I finally have Zadig install and that there is no more error that say there nothing detect in my USB port, I hit the run button in PulseView and run my program at the same time. There was nothing that I can see in the D0 and D1 line, they both just a straight line for some reason. I was confused and don’t know what to do, so I decide to head to the lab hours to get some help using PulseView since I don’t know how PulseView work at all.



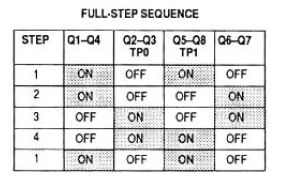
After going to the office hour, the TA help me set up PulseView and tell me how to use it. It seems like I have not chosen the waveform line for I2C bus which enable the application to show me what the pulse for the I2C. Then I did not set up the correct sample size and frequency, so I have to set up my I2C line in the PulseView and set the logic analyzer channel to the I2C in PulseView by setting D0 to SCL and D1 to SDA. I set sample size to be 50 M since I want to have a big sample size and set the frequency at 1MHz. With all that set up on the PulseView, I run the program again and PulseView at the same time. I finally got some reading on the D0 and D1 line and there some reading on the I2C. I have to zoom in the I2C to check whether or not it is working and when I zoom in, I see that their data being send through my I2C bus. After mail the TA and got the confirm that it what I’m looking for when I’m sending out the data, with this I have finish part 1 of project 2, and I’m ready to move on to part 2 of the project where I have to control a motor with the I2C.



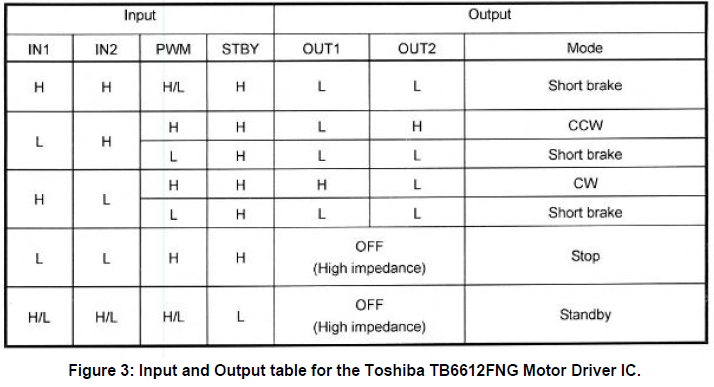
**3/17/2021**

I download the PDF of the project guide and start reading the requirement for the second part of the project then go through the suggested procedure. The first step it saying to read the spec of the motor that we going to use for the lab, I skim through the specs of the motor and move on to step 2. Step 2 talk about the logic that is required to make the motor turn each time and how the motor spin using the field coil. I skim through this part too and move on to step 3 which tell you about the Full-Step and Half Step, but I also just skim through it and not really work out how to make each switch change to match with the Sequence table. Step 4 was to look at the data sheet for the NXT PCA9685 schematic, I went to the website and do a quick skim of the schematic. Step 5 want to have me read the first 4 pages of the data sheet which really confuse me cause the Adafruit website do not have any documentation that I can read about the 16 LEDs, so I just skip this step and move on to the next step. Step 6, have me go back to page 124 of the Hall textbook to refresh my memory on the H-Bridge so I went over page 124 to refresh my memory on how H-bridge work. For step 7 I just skim through it and move on to step 8 as it only wants me to look at the input and output table of the Toshiba TB6612FNG. Step 9 through step 10, I skim through all of the step to see what they want me to do, but I really don’t get what I am really confuse on all these steps on how to actually make the motor move. I went to read the project guide from top to bottom again and see that there a link for the PCA9685 which I click on it and it take me to a product page that have a documentation of the chip. I open the data sheet and start reading through it as I go through the project guide again to see if there anything new I can learn. As I skim through the PCA data sheet I become really lost with all these information that I am supposed to be taking in which I don’t know what I am suppose to be reading. So I decide to go to the lab tomorrow to have some of my question be answer and help on being point to the right direction on where to read.

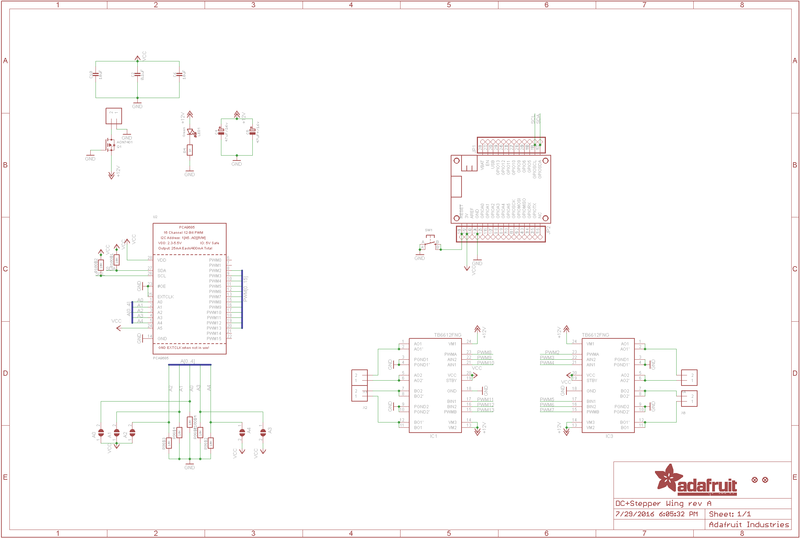
**3/18/2021**

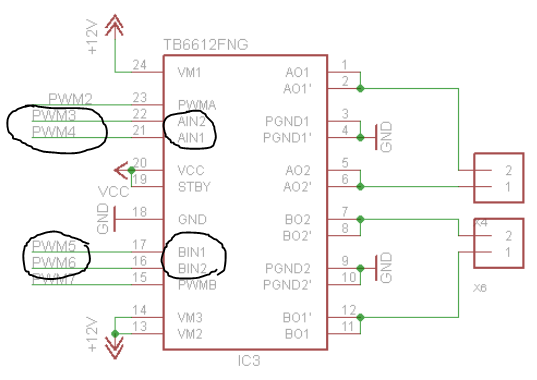
After getting some of my question being clarify, over the project guide and the PCA data sheet, I now know what to do for the project. So I go through step 4 through step 8 of the project guide as I also go over the PCA section 7.2 which it tell you about the control register and have a list of all the LED register. What I have learn from the labs hour is that PWM is basically LED, and that the guide in the project is refering them as PWM hence there was some confusing when I was skimming through the project. Go through guide again starting from step 4 which is talking about the steping sequence of full-step and half-step, but for the project I need to do a full step. 

That is the Full-Step sequence that I need to have, and Q1-Q4 is AIN1, Q2-Q3 is AIN2, Q5-Q6 is BIN1, Q6-Q7 is BIN2. To make the output to be matching with that step table I have to go and look at the table in step 7 which is the truth table for the H-bridge.

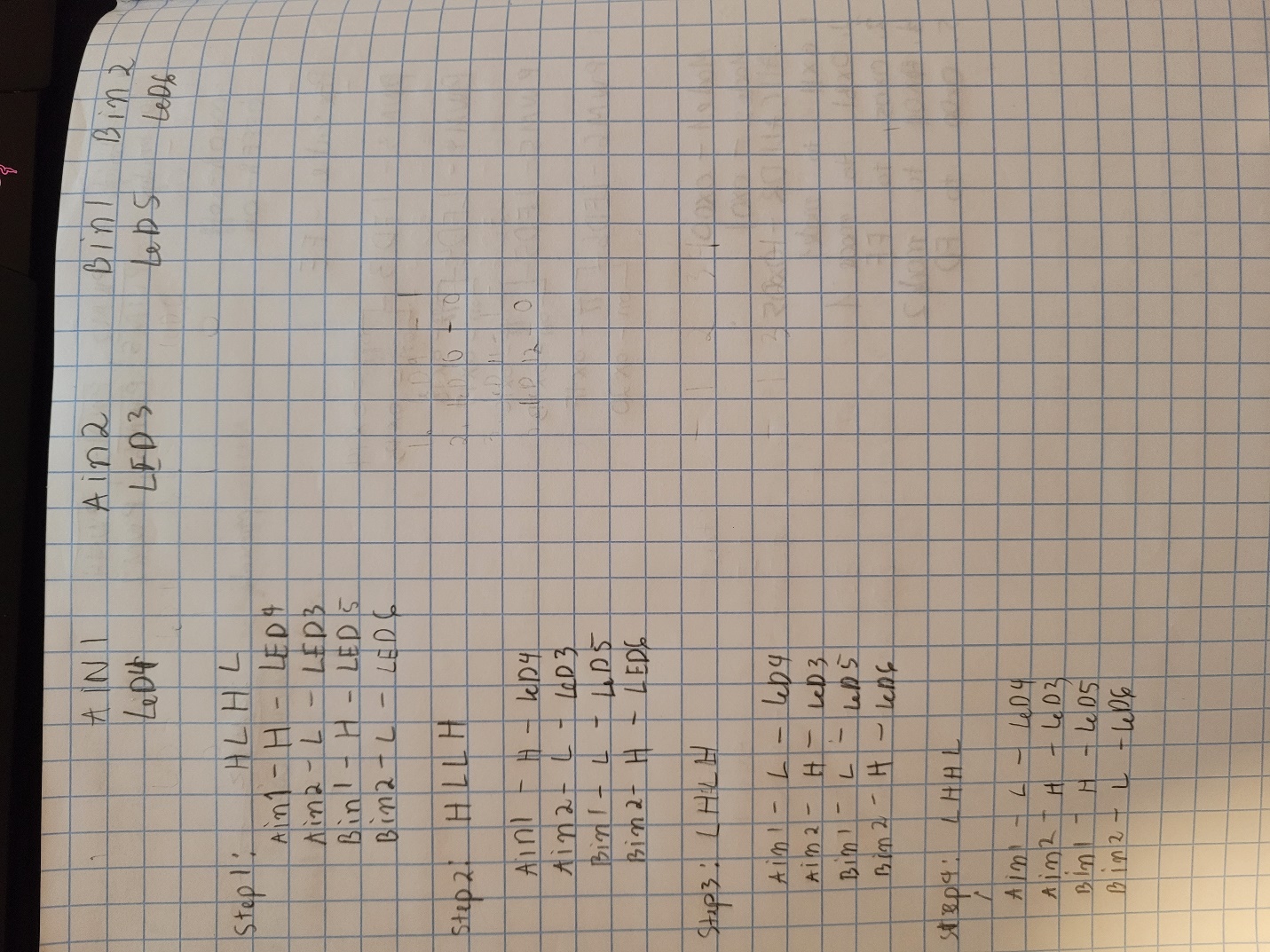


Using this table table I can start writing a psuedo code for my stepper function that going to make the motor move. The next thing I need to do is that I need to find out what LED to use for the chip which I can find the information for this using the website from step 4 of the project guide.

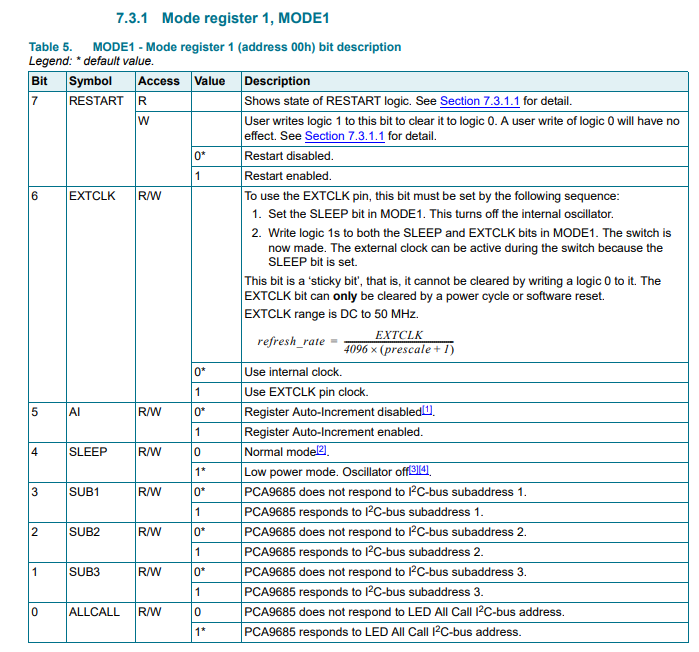




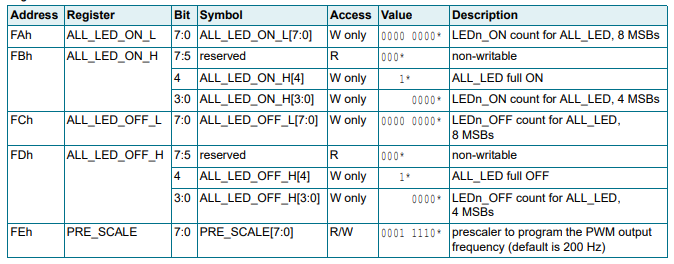
I found that PWM3 is for AIN2, PWM4 is for AIN1, PWM6 is for BIN1, PW7 is for BIN2 using the schematic that is on the adafruit website. With everything I need I can now write a psuedo code for stepper function.

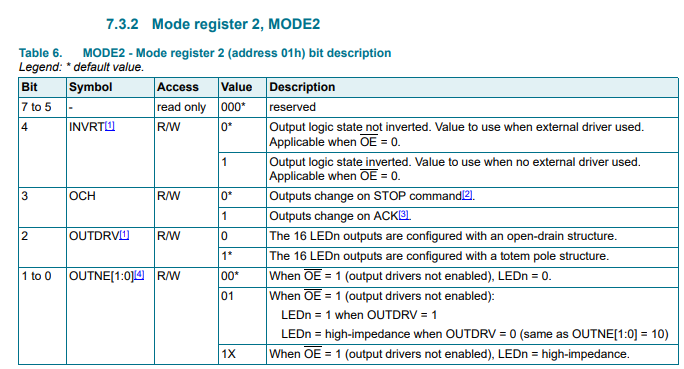


After I have the stepper all figure out the stepper function, the next step I need to do is figure out how to set up the PCA which there are instruction on how to on step 12 in the project guide. I was confused on how to do the initializing because I don’t know what address I am sending stuff to because usually the address we send in is a 32-bit address. The PCA manual only give an 8-byte offset of the chip register, so I do not know what to do. I schedule a meet up with TA, and he give me a quick run down that since the chip can’t take in too much data, so that offset is the address that I am supposed to be sending stuff too. The chip can only read in 2 data at the time, the first data I send in will be the address and the second data will be what I want to write to that address. This clear up a lot of my confusion on where to read the value to configurate the PCA to. With that I start re-read step 12 in the project guide again and try to find the value to configurate my PCA with the data sheet. First thing I need to do is to set it to sleep and respond to all call, this information can be find in page 14 of the data sheet which it list the address of where you want to write to have the PCA go sleep and respond to all call.



I must write a 1 in the 0th bit and 1 in the 4th bit to make it go sleep and respond to all LED call which make the value to be 0x11 and I have to write this value to Mode 1 register which have an offset of 0x00. Next I have to set the pre-scale mode for 1 KHz, so I scroll through the PCA to find the section that talk about pre-scale mode and the page that talk about pre-scaling is page 25 in the data sheet.



I would have to write to address 0xFE to access the pre-scale, and now I have to find the right value to put in to make it to 1KHz. Using the equation down at section 7.3.5 what value I end up with is 0x05, and that is what I need to write to address 0xFE to make this pre-scale to 1KHz. Then I must restart the PCA, set it to respond to all call, internal clock and on normal mode. To have it restart, respond to all call, and internal clock I just need to read in 0x81 to mode 1 which is show in section 7.3.1 of the data sheet for PCA that show you the bit description field. Then the next step is I must set up the totem pulse structure and non-inverted, this information can be find in section 7.3.2 in the data sheet of the PCA.

Which I have to read into address of mode 2 to set the totem pulse structure and non-inverted, the value I will be need to read to mode 2 is 0x04. Then after that I must zero and make all the LED to be off, which can be done by writing to address 0xFD which is the address of ALL\_LED\_OFF\_H that I obtain on section 7.3.4 of the PCA data sheet. The value that is need to be written to the address is 0x00, this will make sure that all the LED will be off and you only be using just the LED you choose. With everything I need to have for part 2 of the project, I start writing my code in Code Composer.

**Code for Part 2 V1:**

**.text**

**.global** \_start

**.global** INT\_DIRECTOR

**\_start:**

@@@@@@@@SET UP THE STACK@@@@@@@@@@@@@@@@@@@@@@@@@@@

LDR R13,=STACK1 @Point to base of STACK for svc mod

**ADD** R13,R13,#0x1000 @Point to top of STACK

CPS #0x12 @Switch to IRQ mode

LDR R13,=STACK2 @Point to IRQ mode

**ADD** R13,R13,#0x1000 @Point to top of STACK

CPS #0x13 @Back to SVC mode

@==================================================

@Wake up I2C2 clock

LDR R1, =0x44E00044 @base address of CM\_PER\_I2C2\_CLKTRL

LDR R2, =0x02 @value to wake up the clock

STR R2,[R1] @store the value to the I2C2 clock

@==================================================

@Initializing and setting up I2C2

@Switching Pin 19 and Pin 20 in BeagleBone to Mode 3 for I2C2\_SCL and I2C2\_SDA

LDR R1, =0x44E1097C @base address of UART\_RSTN

LDR R3, =0x33 @value to switch to mode 3, set the PullUp/PullDown enable and select PullUp, and Enable the receiver

STR R3, [R1] @switch to mode 3

LDR R1, =0x44E10978 @base address of UART\_CTSN

STR R3, [R1] @value to switch to mode 3, set the PullUp/PullDown enable and select PullUp, and Enable the receiver

@==================================================

@Reset the I2C to clear the System.

LDR R1, =0x44E00000 @Base address of CM\_PER

**ADD** R2, R1, #0xBC @SYSTEST address

**MOV** R3, #0xFFFFBFFF

STR R3, [R2]

**ADD** R2, R1, #0xA4 @I2C\_CON

**MOV** R3, #0xFFFF7FFF @value to disable I2C

STR R3, [R2]

**ADD** R2, R1, #0x10

**MOV** R3, #0x02 @value to reset the I2C

STR R3, [R2]

@==================================================

@Before enable the I2C2

@Setting up Prescale value for the clock to get 12Mhz

LDR R1, =0x4819C000 @base address of I2C2

**ADD** R2, R1, #0xB0 @offset of I2C2 CLock Prescaler Register

**MOV** R3, #0x3 @setting value to divide by 4

STR R3, [R2] @divide 48Mhz by 4 to obtain 12Mhz

@set up the Low time for I2C2\_SCLL

**ADD** R2, R1, #0xB4 @off set of I2C2\_SCLL register

**MOV** R3, #0x08 @ Value to set it to 400Kps. 8 in decimal and 0x08 in hex

STR R3, [R2] @store the value to I2C2\_SCLL

@set up the High time for I2C2\_SCHL

**ADD** R2, R1, #0xB8 @offset of I2C2\_SCHL register

**MOV** R3, #0x0A @value to set it to 400Kps. 10 in decimal and 0x0A in hex

STR R3, [R2] @store the value to I2C2\_SCHL

@Configure its own address

**ADD** R2, R1, #0xA8 @Address of I2C2\_OA

**MOV** R3, #0x00 @Value to reset it

STR R3, [R2] @store the value to I2C2\_OA

@Take the I2C module out of reset

**ADD** R2, R1, #0xA4 @address of I2C2\_CON

**MOV** R3, #0x8600 @enable the 15 bits to enable the module

STR R3, [R2]

@====================================================

@ Configure Slave Address and DATA counter register

@setting up DATA counter

LDR R1, =0x4819C000 @base address of I2C2

**ADD** R2, R1, #0x98 @offset of Data counter register

**MOV** R3, #0x02 @value to write 2 characters

STR R3, [R2] @store the value to count

@setting up the slave

**ADD** R2, R1, #0xAC @address of IC2C2 Slave register

**MOV** R5, #0x60 @value to have 60

STR R5, [R2] @store the value to the slave

@===================================================

**B** SET\_UP

@===================================================

**SET\_UP:**

LDR R1, =COUNTER

LDR R2, [R1]

**CMP** R2, #7

BNE SET\_UP2

@go to step

**B** SEND\_STEP\_LOOP

**SET\_UP2:**

BL DELAY

BL START

**B** INSTRUCTION

@===================================================

@infinite loop

**INF\_LOOP:**

**NOP**

**B** INF\_LOOP

@===================================================

**INSTRUCTION:**

@Send first byte

LDR R0, =NXP\_PTR @load the pointer

LDR R1, [R0] @read in the character

LDRB R4,[R1], #1 @increment the pointer

STR R1, [R0] @Store back the new pointer

LDR R5, =0x4819C09C @load in I2C2\_DATA

STRB R4, [R5] @Send data to I2C2\_DATA

@Send second byte

LDR R0, =NXP\_PTR @load the pointer

LDR R1, [R0] @read in the character

LDRB R4,[R1], #1 @increment the pointer

STR R1, [R0] @Store back the new pointer

LDR R5, =0x4819C09C @load in I2C2\_DATA

STRB R4, [R5] @Send data to I2C2\_DATA

@====================

LDR R1, =0x4819C024

**MOV** R2, #0x10

STRB R2, [R1]

@increment counter

LDR R1, =COUNTER @load in the counter

LDR R2, [R1] @get the value being store there

**ADD** R2, R2, #1 @Increment the value at COUNTER

STRB R2, [R1] @Store the new value back to COUNTER

**B** SET\_UP

@=============================

@stepping through

**SEND\_STEP\_LOOP:**

LDR R1, =STEP\_COUNT

LDR R2, [R1]

**CMP** R2, #200

BNE FULL\_STEP

BEQ INF\_LOOP

**FULL\_STEP:**

**B** STEP1

**STEP1:**

@LED4 HIGH

@LED3 LOW

@LED5 HIGH

@LED6 LOW

@make LED3\_ON\_H to full low

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x13 @LED3\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x00 @value to make LED3 full low

STR R2, [R1] @Send the value to make LED to full on

@Make LED4\_ON\_H full high

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x17 @LED4\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x10 @value to make LED4 full high

STR R2, [R1] @Send the value to make LED to full on

@make LED5\_ON\_H full high

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x1B @LED5\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x10 @value to make LED5 full high

STR R2, [R1] @Send the value to make LED to full on

@make LED6\_ON\_H full low

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x1F @LED6\_ON\_H

STRB R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x00 @value to make LED6 full low

STRB R2, [R1] @Send the value to make LED to full low

BL DELAY

**B** SEND\_STEP\_LOOP

@============================

**START:**

**POLL\_BB:**

LDR R1, =0x4819C024 @base address of I2C2\_IRQSTATUS\_RAW

LDR R2, [R1] @load the value in

TST R2, #0x1000 @check if the bit is clear or set

BNE POLL\_BB @keep Polling if the Bus is busy

@setting up DATA counter

LDR R1, =0x4819C000 @base address of I2C2

**ADD** R2, R1, #0x98 @offset of Data counter register

LDR R3, =0x02 @value to write 2 characters

STRB R3, [R2] @store the value to count

@============================

LDR R1, =0x4819C0A4 @base address of I2C2\_CON

LDR R2, =0x8603 @Turn on the STT and STP, STT = 1 and STP = 1, Start and Stop

STRB R2, [R1] @store the value to register

**POLL\_XRDY:**

LDR R0, =0x4819C024 @base address of I2C2\_IRQSTATUS\_RAW

LDR R2,[R0] @load in the address

TST R2, #0x10 @Check bit 4 for XRDY

BEQ POLL\_XRDY @if it not set then keep Polling

**MOV** PC, R14 @Return

**DELAY:**

**LDR** R1, =0x00040000

**WAIT\_LOOP:**

SUBS R1,R1,#1

BNE WAIT\_LOOP

**MOV** PC, R14

**.data**

**.align** 2

**STACK1:** .rept 1024

**.word** 0x0000

.endr

**STACK2:** .rept 1024

**.word** 0x0000

.endr

**NXP\_INTRUCTION:**

**.byte** 0x00 @mode1

**.byte** 0x11

**.byte** 0x00 @mode1

**.byte** 0x81

**.byte** 0xFE @Pre\_scale

**.byte** 0x05

**.byte** 0x01 @mode2

**.byte** 0x04

**.byte** 0xFD @ALL\_LED\_OFF

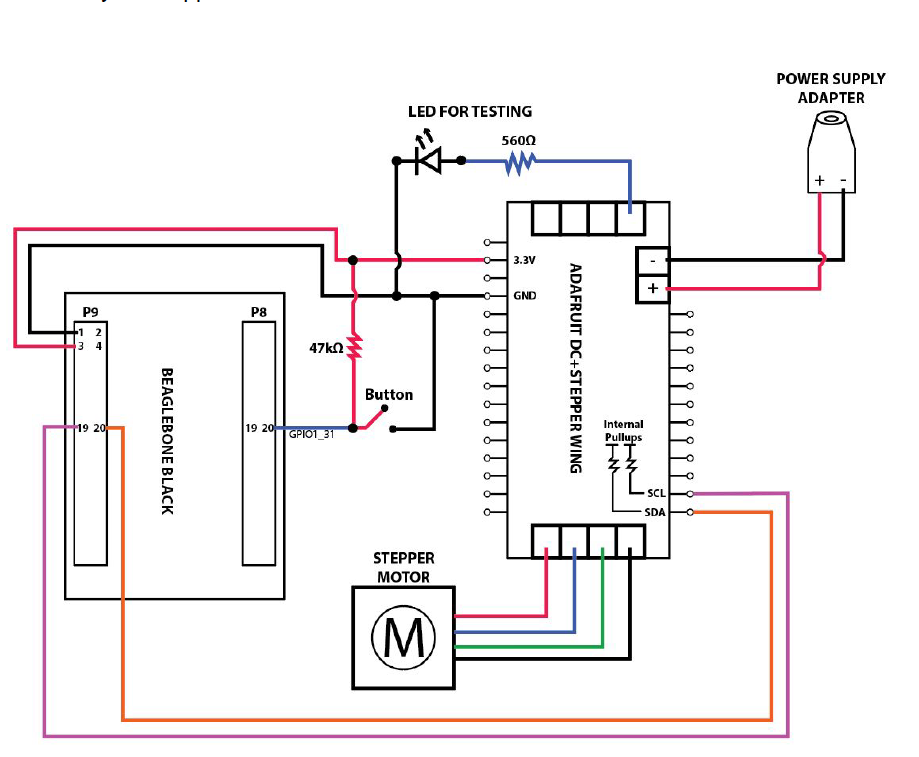
**.byte** 0x00

**NXP\_PTR:** **.word** NXP\_INTRUCTION

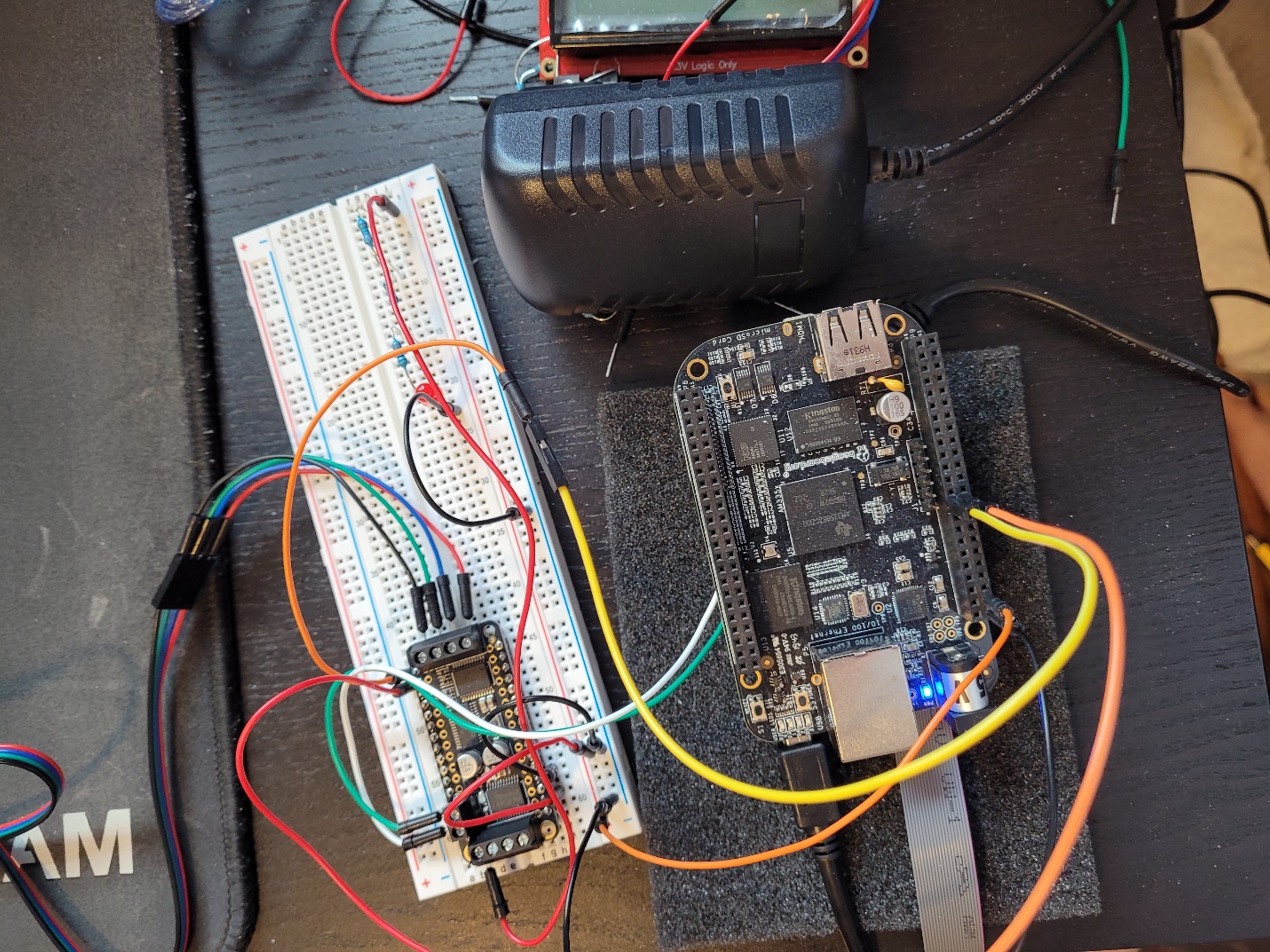
**COUNTER:** **.word** 0

**STEP\_COUNT:** **.word** 0

.end

I copy and paste the code for part 1 of the project over to my part 2 so that I can use part 1 as the base and start from there. There some changes that I make for part 1 code just for part 2, which I combine my POLL\_BB, POLL\_XRDY, START all in 1 function which is START so that when I have to send out a byte I don’t have to call 3 functions but instead I just have to call 1. After ward I make a pointer to an array of byte that will be the instruction byte which will be send to the PCA for setting up. Then I make an instruction function that will be sending those PCA set up bit that I just find and send it to the I2C2\_DATA register. I look at my old code for project 1 to use the pointer array, and I sent 2 byte every instruction since the PCA can only take 2 at the time. After every send instruction I have a delay loop until the next instruction is going to be send in, so I just make a loop and have it keep sending in the next instruction till there is no more instruction to be send. Once all the set-up instruction has been send, I create a STEP function that will be sending out my STEP and have it loop for 200 times. I make STEP1 only using the pseudo code that I have make because I want to see if my pseudo code work and if it does then the motor should move just a bit when it is going through STEP1 function. When I finish writing my code the next thing, I need to do is I start to hook up the hardware using the schematic in the project guide. 

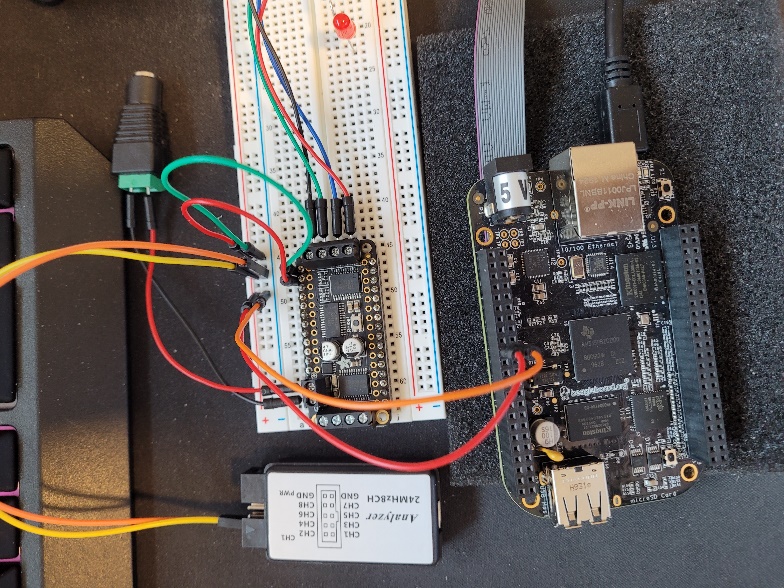
I wire my hardware with the same color that the schematic use, and as I wire my hardware I didn’t not hook up the button to the beagle bone at all.



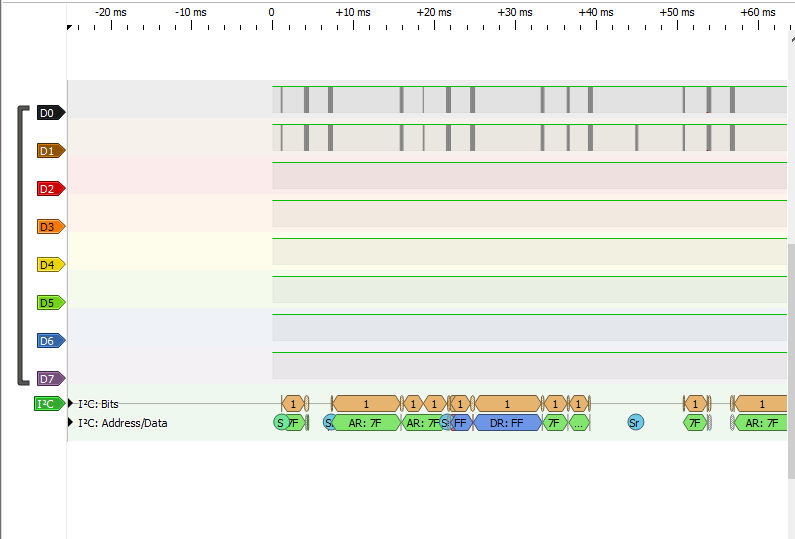
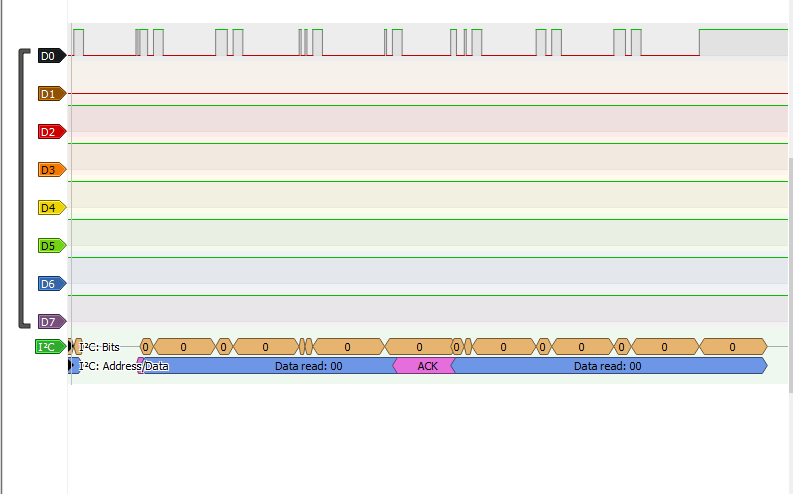
After I have everything hook up, I start to build my program and fix all the small bug that I have mainly just syntax error once I have it all fix my program is fully built and ready to be tested. I turn on the BeagleBone Black and plug in the power for the H-Bridge. First test run of the driver, I hit run on the program and check to see if the motor is moving or not and the result is that nothing is happening. When I hit pause, I see that there is a data abort as an error in the interrupt table, which make me instantly go to my .data section and add in some. align 2 command. What I have learn in project 1 is that a lot of data abort error come from your .data section which is why I know where to go for the error. Once I fix that I built the program again and then run it, but the result is still the same the motor does not make a single movement at all. Which confuse me very much, so I decide to go over my code again to make sure that everything is correct and there is nothing wrong with my code. I go through each register and check whether the address is correct and the value I am reading into them is correctly. There was nothing wrong I can see in the code which I know of, so I decide to double check my hardware next to make sure that it is correctly connect. I check the hardware wire but there isn’t anything that I see that is also off about it so I decide to wait for tomorrow lab time to ask about what is wrong.

**3/19/2021**

After I go to the lab hour which the TA tell me to try to see if I am getting any signal from the motor port like voltage reading. So I use a voltmeter and then run my program then I try to check if there any reading from the IN pint that led to the motor. When I do the reading, I see that there is no voltage going to motor at all which is the reason why it is not moving when I run my code. I then check to see if there any reading on the power source to confirm that there is power going into the bridge, and I got a reading of 12V which indicate that there is power going through the bridge. I do a final check on my wiring and re-tight everything, and then run the program again but this give me the same result which is the motor still not moving. This only mean one thing and that is there is something wrong with my code that I need to check. I went over my code again to check for if there anything that I could have miss, but there is nothing that show up for me that is missing. I check the way I set up my I2C2 and initializing it, and everything look right. What I decide to do next is to check for the signal to see if I am sending the correct thing, so I hook up the logic analyzer to the H-bridge to find out whether my data is being sent in correctly.



I run the program again and hit play on both of the application then after a few second I can see where my problem is which is I am not sending the right data. There is a lot of data being sent through out my I2C which I know that I should be sending just 10 instruction byte and the step but the I2C line look like there is more than just that going to the H-bridge. So, I decide to zoom in to the I2C line to see what is being read to the H-bridge chip.



I see that these are not what I plan to send do the PCA at all, and that it seems like I am sending nothing to the PCA from the look of the data that is being read into. I go through the code again and check to see if I am doing everything correctly, and there nothing I think I can see that is wrong. So what I do next is to hook the logic analyzer directly to the BeagleBone just like how I have it in part 1 to see if I am reading out the data correctly. The result still the same and I still get data read is 00, which make me decide to go through the code line by line in Code Composer to see what is wrong. At first, I thought it was my INSTRUCTION function which handle the instruction byte that needs to be sent to configurate the PCA that is wrong. I make a break point there and start to go through the instruction, when I go through my START function which then I see that for some reason I am stuck at the part where I check for the BB bit because I keep on looping back to START. So, I open the register tab and start reading in what is BB bit is being set to and in the register, I see that it is being set at 1 all the time which mean that the bus is busy and preventing me from sending the next data bit in. I do not know why this is happening, I try to read over the project guide again to see if there anything that can help me with this. But there nothing in the project guide that talk about this so I go through the TI manual section 21.3.15 again to see if there anything that can help me in there. I read through each step again and there is nothing that I see that can help with the BB being set at 1 and not switching back to 0. What I decide to do next is I decide to google on why the BB bit is keep being set at 0, and what I find is that someone is also having this problem and the solution of it is to write 0x8600 to the I2C\_CON register again before you start transmitting the data. But I remember I already done this already when I was setting up my I2C2 and initialize it, that I already write 0x8600 to the I2C\_CON register already. After going through the google search with no other answer other than reading 0x8600 to the I2C\_CON register, so what I do next is I try to insert another 0x8600 to I2C\_CON that in my START function right after my first time checking for BB because it seems like it go through perfectly at first but just the second time START get run that it stuck.

**START:**

**POLL\_BB:**

LDR R1, =0x4819C024 @base address of I2C2\_IRQSTATUS\_RAW

LDR R2, [R1] @load the value in

TST R2, #0x1000 @check if the bit is clear or set

BNE POLL\_BB @keep Polling if the Bus is busy

@reset I2C

LDR R1, =0x4819C0A4

LDR R2, =0x8600

STR R2, [R1]

@setting up DATA counter

LDR R1, =0x4819C000 @base address of I2C2

**ADD** R2, R1, #0x98 @offset of Data counter register

LDR R3, =0x02 @value to write 2 characters

STRB R3, [R2] @store the value to count

@============================

LDR R1, =0x4819C0A4 @base address of I2C2\_CON

LDR R2, =0x8603 @Turn on the STT and STP, STT = 1 and STP = 1, Start and Stop

STRB R2, [R1] @store the value to register

**POLL\_XRDY:**

LDR R0, =0x4819C024 @base address of I2C2\_IRQSTATUS\_RAW

LDR R2,[R0] @load in the address

TST R2, #0x10 @Check bit 4 for XRDY

BEQ POLL\_XRDY @if it not set then keep Polling

**MOV** PC, R14 @Return

I then run the program again and this time I see that there are some reading through my I2C line, then I zoom in and check to see if it is the correct data that suppose to be read in. The result is that the I2C line is correctly sending out my signal this time, so I start hooking back the SCL and SDA back to the H-Bridge and see if there any change. After I finish hooking back all the hardware back, I run the program again but there still no movement from the motor still. I hook up the logic analyzer to the SCL and SDA line again to see if the data getting mess up again. But it is reading in the correct data to the PCA so I use the voltmeter to check for if there any voltage going in and out of the pin. With the voltage meter I determine that there isn’t voltage going out to the motor at all, there is some voltage that is going out but they are very small for some reason because I’m getting 1- 2V reading on 1 pin and 0V on the rest of the pin.

**3/24/2021**

I continue from last week where I manage to get the data being read in correctly from the but there is no voltage reading. I have talk with the TA and it seem like I am using the wrong register for my LED because we only going to be using LED\_ON\_H only but I am using a mix of LED\_ON\_H and LED\_ON\_L which that might be the reason why it is not working the way that it should be. So I change all my LED\_ON\_L to LED\_ON\_H in my step function and then run the program again, but there still no change to the motor. It still not move at all, so what I do next was going over all the guide again, and make sure to triple check my code to see if there anything that is wrong with the code. I still don’t see anything that could be wrong with the code which make me wonder if I need to write out the full step instead of just step1 function, so I start add in step 2 – 4 into the code.

Code part 2 V2:

**.text**

**.global** \_start

**.global** INT\_DIRECTOR

**\_start:**

@@@@@@@@SET UP THE STACK@@@@@@@@@@@@@@@@@@@@@@@@@@@

LDR R13,=STACK1 @Point to base of STACK for svc mod

**ADD** R13,R13,#0x1000 @Point to top of STACK

CPS #0x12 @Switch to IRQ mode

LDR R13,=STACK2 @Point to IRQ mode

**ADD** R13,R13,#0x1000 @Point to top of STACK

CPS #0x13 @Back to SVC mode

@==================================================

@Wake up I2C2 clock

LDR R1, =0x44E00044 @base address of CM\_PER\_I2C2\_CLKTRL

LDR R2, =0x02 @value to wake up the clock

STR R2,[R1] @store the value to the I2C2 clock

@==================================================

@Initializing and setting up I2C2

@Switching Pin 19 and Pin 20 in BeagleBone to Mode 3 for I2C2\_SCL and I2C2\_SDA

LDR R1, =0x44E1097C @base address of UART\_RSTN

LDR R3, =0x33 @value to switch to mode 3, set the PullUp/PullDown enable and select PullUp, and Enable the receiver

STR R3, [R1] @switch to mode 3

LDR R1, =0x44E10978 @base address of UART\_CTSN

STR R3, [R1] @value to switch to mode 3, set the PullUp/PullDown enable and select PullUp, and Enable the receiver

@==================================================

@Reset the I2C to clear the System.

LDR R1, =0x44E00000 @Base address of CM\_PER

**ADD** R2, R1, #0xBC @SYSTEST address

**MOV** R3, #0xFFFFBFFF

STR R3, [R2]

**ADD** R2, R1, #0xA4 @I2C\_CON

**MOV** R3, #0xFFFF7FFF @value to disable I2C

STR R3, [R2]

**ADD** R2, R1, #0x10

**MOV** R3, #0x02 @value to reset the I2C

STR R3, [R2]

@==================================================

@Before enable the I2C2

@Setting up Prescale value for the clock to get 12Mhz

LDR R1, =0x4819C000 @base address of I2C2

**ADD** R2, R1, #0xB0 @offset of I2C2 CLock Prescaler Register

**MOV** R3, #0x3 @setting value to divide by 4

STR R3, [R2] @divide 48Mhz by 4 to obtain 12Mhz

@set up the Low time for I2C2\_SCLL

**ADD** R2, R1, #0xB4 @off set of I2C2\_SCLL register

**MOV** R3, #0x08 @ Value to set it to 400Kps. 8 in decimal and 0x08 in hex

STR R3, [R2] @store the value to I2C2\_SCLL

@set up the High time for I2C2\_SCHL

**ADD** R2, R1, #0xB8 @offset of I2C2\_SCHL register

**MOV** R3, #0x0A @value to set it to 400Kps. 10 in decimal and 0x0A in hex

STR R3, [R2] @store the value to I2C2\_SCHL

@Configure its own address

**ADD** R2, R1, #0xA8 @Address of I2C2\_OA

**MOV** R3, #0x00 @Value to reset it

STR R3, [R2] @store the value to I2C2\_OA

@Take the I2C module out of reset

**ADD** R2, R1, #0xA4 @address of I2C2\_CON

**MOV** R3, #0x8600 @enable the 15 bits to enable the module

STR R3, [R2]

@====================================================

@ Configure Slave Address and DATA counter register

@setting up DATA counter

LDR R1, =0x4819C000 @base address of I2C2

**ADD** R2, R1, #0x98 @offset of Data counter register

**MOV** R3, #0x02 @value to write 2 characters

STR R3, [R2] @store the value to count

@setting up the slave

**ADD** R2, R1, #0xAC @address of IC2C2 Slave register

**MOV** R5, #0x60 @value to have 60

STR R5, [R2] @store the value to the slave

@===================================================

**B** SET\_UP

@===================================================

**SET\_UP:**

LDR R1, =COUNTER

LDR R2, [R1]

**CMP** R2, #7

BNE SET\_UP2

@go to step

**B** SEND\_STEP\_LOOP

**SET\_UP2:**

BL DELAY

BL START

**B** INSTRUCTION

@===================================================

@infinite loop

**INF\_LOOP:**

**NOP**

**B** INF\_LOOP

@===================================================

**INSTRUCTION:**

@Send first byte

LDR R0, =NXP\_PTR @load the pointer

LDR R1, [R0] @read in the character

LDRB R4,[R1], #1 @increment the pointer

STR R1, [R0] @Store back the new pointer

LDR R5, =0x4819C09C @load in I2C2\_DATA

STRB R4, [R5] @Send data to I2C2\_DATA

@Send second byte

LDR R0, =NXP\_PTR @load the pointer

LDR R1, [R0] @read in the character

LDRB R4,[R1], #1 @increment the pointer

STR R1, [R0] @Store back the new pointer

LDR R5, =0x4819C09C @load in I2C2\_DATA

STRB R4, [R5] @Send data to I2C2\_DATA

@increment counter

LDR R1, =COUNTER @load in the counter

LDR R2, [R1] @get the value being store there

**ADD** R2, R2, #1 @Increment the value at COUNTER

STRB R2, [R1] @Store the new value back to COUNTER

**B** SET\_UP

@=============================

@stepping through

**SEND\_STEP\_LOOP:**

LDR R1, =STEP\_COUNT

LDR R2, [R1]

**CMP** R2, #200

BNE FULL\_STEP

BEQ INF\_LOOP

**FULL\_STEP:**

**B** STEP1

**STEP1:**

@LED4 HIGH

@LED3 LOW

@LED5 HIGH

@LED6 LOW

@make LED3\_ON\_H to full low

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x13 @LED3\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x00 @value to make LED3 full low

STR R2, [R1] @Send the value to make LED to full on

@Make LED4\_ON\_H full high

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x17 @LED4\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x10 @value to make LED4 full high

STR R2, [R1] @Send the value to make LED to full on

@make LED5\_ON\_H full high

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x1B @LED5\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x10 @value to make LED5 full high

STR R2, [R1] @Send the value to make LED to full on

@make LED6\_ON\_H full low

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x1F @LED6\_ON\_H

STRB R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x00 @value to make LED6 full low

STRB R2, [R1] @Send the value to make LED to full low

BL DELAY

**B** STEP2

**STEP2:**

@LED4 HIGH

@LED3 LOW

@LED5 LOW

@LED6 HIGH

@make LED3\_ON\_H to full low

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x13 @LED3\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x00 @value to make LED3 full low

STR R2, [R1] @Send the value to make LED to full on

@Make LED4\_ON\_H full high

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x17 @LED4\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x10 @value to make LED4 full high

STR R2, [R1] @Send the value to make LED to full on

@make LED5\_ON\_H full LOW

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x1B @LED5\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x00 @value to make LED5 full LOW

STR R2, [R1] @Send the value to make LED to full on

@make LED6\_ON\_H full HIGH

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x1F @LED6\_ON\_H

STRB R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x10 @value to make LED6 full HIGH

STRB R2, [R1] @Send the value to make LED to full low

BL DELAY

**B** STEP3

**STEP3:**

@LED4 LOW

@LED3 HIGH

@LED5 LOW

@LED6 HIGH

@make LED3\_ON\_H to full HIGH

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x13 @LED3\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x10 @value to make LED3 full HIGH

STR R2, [R1] @Send the value to make LED to full on

@Make LED4\_ON\_H full LOW

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x17 @LED4\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x00 @value to make LED4 full LOW

STR R2, [R1] @Send the value to make LED to full on

@make LED5\_ON\_H full LOW

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x1B @LED5\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x00 @value to make LED5 full LOW

STR R2, [R1] @Send the value to make LED to full LOW

@make LED6\_ON\_H full HIGH

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x1F @LED6\_ON\_H

STRB R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x10 @value to make LED6 full HIGH

STRB R2, [R1] @Send the value to make LED to full HIGH

BL DELAY

**B** STEP4

**STEP4:**

@LED4 LOW

@LED3 HIGH

@LED5 HIGH

@LED6 LOW

@make LED3\_ON\_H to full HIGH

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x13 @LED3\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x10 @value to make LED3 full HIGH

STR R2, [R1] @Send the value to make LED to full on

@Make LED4\_ON\_H full high

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x17 @LED4\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x00 @value to make LED4 full high

STR R2, [R1] @Send the value to make LED to full on

@make LED5\_ON\_H full high

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x1B @LED5\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x10 @value to make LED5 full high

STR R2, [R1] @Send the value to make LED to full on

@make LED6\_ON\_H full low

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x1F @LED6\_ON\_H

STRB R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x00 @value to make LED6 full low

STRB R2, [R1] @Send the value to make LED to full low

@increment counter

LDR R1, =STEP\_COUNT @load in the counter

LDR R2, [R1] @get the value being store there

**ADD** R2, R2, #1 @Increment the value at

STRB R2, [R1] @Store the new value back to

BL DELAY

**B** SEND\_STEP\_LOOP

@============================

**START:**

**POLL\_BB:**

LDR R1, =0x4819C024 @base address of I2C2\_IRQSTATUS\_RAW

LDR R2, [R1] @load the value in

TST R2, #0x1000 @check if the bit is clear or set

BNE POLL\_BB @keep Polling if the Bus is busy

@reset I2C

LDR R1, =0x4819C0A4

LDR R2, =0x8600

STR R2, [R1]

@setting up DATA counter

LDR R1, =0x4819C000 @base address of I2C2

**ADD** R2, R1, #0x98 @offset of Data counter register

LDR R3, =0x02 @value to write 2 characters

STRB R3, [R2] @store the value to count

@============================

LDR R1, =0x4819C0A4 @base address of I2C2\_CON

LDR R2, =0x8603 @Turn on the STT and STP, STT = 1 and STP = 1, Start and Stop

STRB R2, [R1] @store the value to register

**POLL\_XRDY:**

LDR R0, =0x4819C024 @base address of I2C2\_IRQSTATUS\_RAW

LDR R2,[R0] @load in the address

TST R2, #0x10 @Check bit 4 for XRDY

BEQ POLL\_XRDY @if it not set then keep Polling

**MOV** PC, R14 @Return

**DELAY:**

**LDR** R1, =0x00040000

**WAIT\_LOOP:**

SUBS R1,R1,#1

BNE WAIT\_LOOP

**MOV** PC, R14

**.data**

**.align** 2

**STACK1:** .rept 1024

**.word** 0x0000

.endr

**STACK2:** .rept 1024

**.word** 0x0000

.endr

**.align** 2

**NXP\_INTRUCTION:**

**.byte** 0x00 @mode1

**.byte** 0x11

**.byte** 0x00 @mode1

**.byte** 0x81

**.byte** 0xFE @Pre\_scale

**.byte** 0x05

**.byte** 0x01 @mode2

**.byte** 0x04

**.byte** 0xFD @ALL\_LED\_OFF

**.byte** 0x00

**.align** 2

**NXP\_PTR:** **.word** NXP\_INTRUCTION

**COUNTER:** **.word** 0

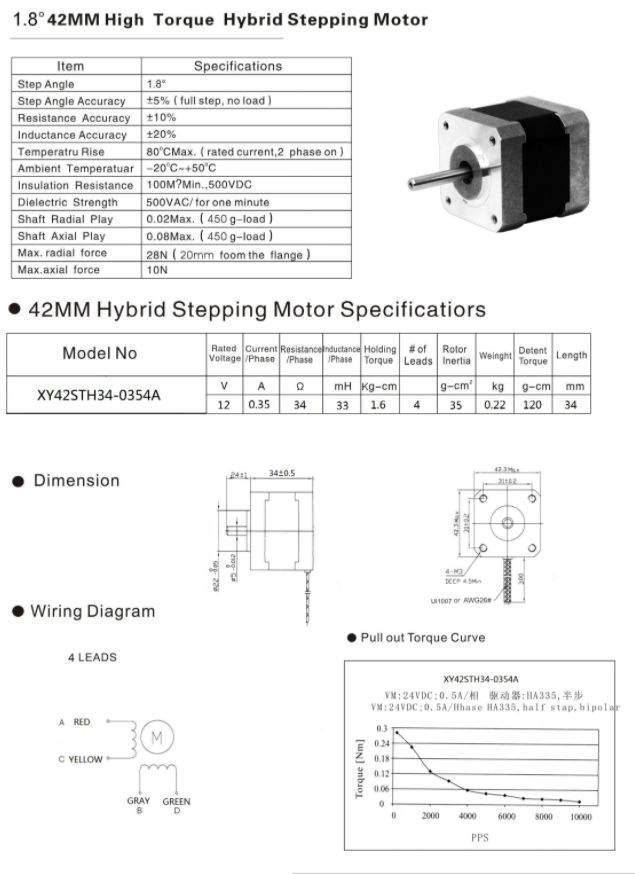
**STEP\_COUNT:** **.word** 0

.end

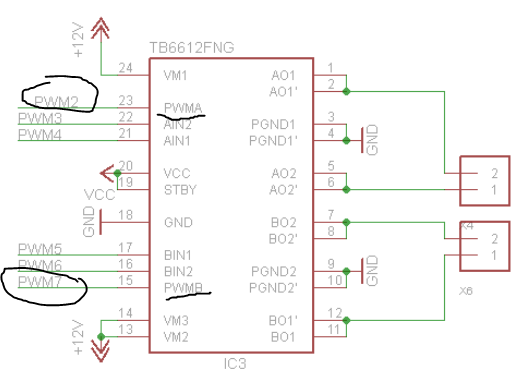
After making the fix I run the program again, and this time even though the motor is not moving but I can hear a ticking noise. I hold the motor on my hand, and I feel there some kind of movement that is happening inside the motor. It sounds like the motor is trying to move but it either being stop by something or that it too weak to move, but I have make some progress today. Since I was able to get it to some what work. I just need to figure out why it is not moving but only make ticking sound.

**3/25/2021**

Since I was able to make it make a ticking noise yesterday, so I plan to see if I can get it to move today. I use a voltmeter and check the pin that connect to the motor as I run the code and I can see that there is more voltage going through the pin but it very small for some reason. I got 4V in one of the pin and 1-2 V for all the other pin which make me think that maybe I am doing something wrong with the code again this time. I start with going over the code again to see if I miss anything but there nothing that can show me what is wrong. So I pull out the PCA manual to check all the LED\_ON\_H register that I have to see if they are correct and it seem like they all is correct. Then I search up on the motor that we are using and manage to find some data sheet of it.



I see that in the wiring diagram there a yellow and gray leads which when I look at my own motor it doesn’t have those 2 colors but instead it have blue and black. Which make me wonder if it was my wiring that is wrong, so I ask the TA about it and what I learn is that those 2 are the yellow and gray since that is an old data sheet. This make that my wiring is correct still, so I went back to check on my code and start to go over each step. As I was going through each step I was wondering if it was my step that is wrong, so what I did was I mix up the logic a bit to see if it would make any different in the code but when I run the program with the logic mix up the motor just stay silent. There is no more ticking noise coming from it and it like there is no more voltage anymore, so I took the voltmeter and check for voltage and I found that there is no more voltage going through the pin anymore. I then revert the code back to the way I have it at first and then run it again, this time I finally hear the motor start making ticking noise again. Which mean that I must have got the logic correctly, so I keep going through the code again to see if there anything else I am missing. I start looking though the schematic of the PCA again to see if I choose the right LED. Then there something I see which is the PWM7 and PWM2 I see that they are connect to PWMA and PWMB which make me wonder if this need to be turn on for power, but since the project guide said we don’t mess with PWM so I was planning to just ignore it.



But since there nothing else that I can think of that will help so I decide to just put this in and see if it helps regardless. I search up the address of LED7 and LED2 then I turn them both on in my STEP1 function as a test. Then I run the program again, this time the motor makes a very loud tick noise, and it is moving. But from what I can see it sound like the motor is trying to spin but there is something blocking it, it makes a very loud sound of tick like a gear that trying to spin but keep hitting a blockage. This make me realize that I am heading to the right direction and turning on the PWM pin was the right decision. Now I just have to solve why motor is not spinning, my first thought was my logic might be wrong again but my then scrap the ideas off since the motor is moving. So I thought that I must have turn on the PWM in the wrong places, and maybe I need to put it somewhere else. I was planning to move it around to see where is the right spot that I can put it, and I thought to myself that putting it in the stepper function isn’t the right place since I think this need to be turn on before the stepping hence, I decide to put it in my set up where I send all my instruction byte as a test. I run the program again and this time the motor still ticking loudly, but this time I can see that it is turning a little bit. It a very small turn but it is turning, but for some reason it will sometime just move in one place or move backward and then forward again. The movement is janky but at least this is a good start place for me, now I just need to figure out why this is happening. I send a video to the TA in hope he can give me some insight to why this is happening. What I first thought to do is go through my step line by line this time to see if the value is being read in correctly and check to see what step is causing that weird movement. As I was going through the step line by line I notice that my PCA is burning up as I see smoke coming off of it so I quickly unplug the power source and wait for the PCA to cool down before doing anything. I thought that I must have fried the PCA board, so I check all the pin so see if any of the pin in the chip is fried but the pin doesn’t look burn at all so I can safely assume that no harm was done to the chip. I take the whole H-Bridge off and check all the other pin while shinning a flash light, but there nothing I see that is out of the ordinary, but my wire is all melt so I would have to change out the wire.



I switch out new wire and start running the program again, and this time nothing is running there is no more movement form the motor, but I have not change anything yet but the wire. Which make me think that I might have fried the PCA board.

**3/26/2021**

I start a new day from yesterday disaster which I might have fried the PCA board, so I load up the program and hook up the power and everything back again. I run the program this time, but nothing is happening, and the motor still not moving. So I decide that I’m going to take everything apart to check for if any of the part is damage, then I re-wire everything again with a new wire. After I finish doing that, I run the program again and this time the motor is moving again, it still doing the same thing as before moving backward and forward with a loud tick noise. But this mean that none of my part are damage by the incident, I decide that I will not run the code line by line anymore since this cause me to almost fried the PCA. I email the TA to get some inside on why my motor is moving this way. The solution that was being propose to me is my wiring is wrong, or my step is out of order, or my delay is wrong. With that I decide to start with the wiring first by start switching around the wire that is connecting the H-Bridge and the motor. I start messing around with the combination of the wire by first switching the red and the green wire, but all these does is making it go from spinning clockwise to counterclockwise. Then I switch the blue and black but that still do the same thing as me switching the red and green which is making the motor go counterclockwise. I start mixing up the red to green, green to red, blue to black and black to blue, but this just make the motor to start making a loud ticking sound and it not even moving anymore. With me messing around with the wiring I learn that it is not my wiring that is wrong, so I decide to mix around my step to see if it was my step that is out of order. I start moving my step around to see if it makes any difference, but this is also making no different since it either make my motor make a loud sound from the gear not moving or it move backward. I thought it might be the logic on my step rather than the order of my step, so I went over the logic again using the project guide, PCA manual and the schematic just to do my logic again. But even after I go over the logic again there still no different since I do not see anything is wrong with the current logic I have right now. There was nothing wrong with the wiring and the step which leave me with only one thing left and that is the delay. I go back and read the project guide again, and on step 11 of the project guide I see that the delay time I been using is wrong this whole time since I need a count of 5000 between each command which is my instruction bit and a count of 400,000 between my step which I do not have. I start to convert the 5000 and the 400,000 into hex value and make 2 delay loops. The 5000 will now be the delay between the byte I am sending and the 400,000 between my steps. I then start to run the program after those fixes, but the motor still not moving the way I want. It still moving very janky, and it seem like that new delay did not help at all. So, I am back to square one with this, but then I think what if I get rid of the delay loop all at once and see what happen when there is no delay between each step. I delete all the delay step between each step and then run the program, my motor makes a really loud sound but it moves completely difference this time. It moves very fast, but it definitely just make a complete 360 movement with no janky movement like forward and backward. This confirm that it is my delay that is wrong, so with this information I start to put in delay again but this time I start with just a count of 1000 and I keep increasing the count with each run so I can see what is wrong. As I keep on increasing the count, the motor starting to run smoother and smoother, which mean I am on the right track. I just need to find that right count to make the motor to run smoothly with no weird movement, so I keep on increasing the count until I hit 11000 on my count and the motor look smooth for me. With this I can conclude that I have finally finish part 2 of the project.

**Final code for part 2:**

**.text**

**.global** \_start

**.global** INT\_DIRECTOR

**\_start:**

@@@@@@@@SET UP THE STACK@@@@@@@@@@@@@@@@@@@@@@@@@@@

LDR R13,=STACK1 @Point to base of STACK for svc mod

**ADD** R13,R13,#0x1000 @Point to top of STACK

CPS #0x12 @Switch to IRQ mode

LDR R13,=STACK2 @Point to IRQ mode

**ADD** R13,R13,#0x1000 @Point to top of STACK

CPS #0x13 @Back to SVC mode

@==================================================

@Wake up I2C2 clock

LDR R1, =0x44E00044 @base address of CM\_PER\_I2C2\_CLKTRL

LDR R2, =0x02 @value to wake up the clock

STR R2,[R1] @store the value to the I2C2 clock

@==================================================

@Initializing and setting up I2C2

@Switching Pin 19 and Pin 20 in BeagleBone to Mode 3 for I2C2\_SCL and I2C2\_SDA

LDR R1, =0x44E1097C @base address of UART\_RSTN

LDR R3, =0x33 @value to switch to mode 3, set the PullUp/PullDown enable and select PullUp, and Enable the receiver

STR R3, [R1] @switch to mode 3

LDR R1, =0x44E10978 @base address of UART\_CTSN

STR R3, [R1] @value to switch to mode 3, set the PullUp/PullDown enable and select PullUp, and Enable the receiver

@==================================================

@Reset the I2C to clear the System.

LDR R1, =0x44E00000 @Base address of CM\_PER

**ADD** R2, R1, #0xBC @SYSTEST address

**MOV** R3, #0xFFFFBFFF

STR R3, [R2]

**ADD** R2, R1, #0xA4 @I2C\_CON

**MOV** R3, #0xFFFF7FFF @value to disable I2C

STR R3, [R2]

**ADD** R2, R1, #0x10

**MOV** R3, #0x02 @value to reset the I2C

STR R3, [R2]

@==================================================

@Before enable the I2C2

@Setting up Prescale value for the clock to get 12Mhz

LDR R1, =0x4819C000 @base address of I2C2

**ADD** R2, R1, #0xB0 @offset of I2C2 CLock Prescaler Register

**MOV** R3, #0x3 @setting value to divide by 4

STR R3, [R2] @divide 48Mhz by 4 to obtain 12Mhz

@set up the Low time for I2C2\_SCLL

**ADD** R2, R1, #0xB4 @off set of I2C2\_SCLL register

**MOV** R3, #0x08 @ Value to set it to 400Kps. 8 in decimal and 0x08 in hex

STR R3, [R2] @store the value to I2C2\_SCLL

@set up the High time for I2C2\_SCHL

**ADD** R2, R1, #0xB8 @offset of I2C2\_SCHL register

**MOV** R3, #0x0A @value to set it to 400Kps. 10 in decimal and 0x0A in hex

STR R3, [R2] @store the value to I2C2\_SCHL

@Configure its own address

**ADD** R2, R1, #0xA8 @Address of I2C2\_OA

**MOV** R3, #0x00 @Value to reset it

STR R3, [R2] @store the value to I2C2\_OA

@Take the I2C module out of reset

**ADD** R2, R1, #0xA4 @address of I2C2\_CON

**MOV** R3, #0x8600 @enable the 15 bits to enable the module

STR R3, [R2]

@====================================================

@ Configure Slave Address and DATA counter register

@setting up DATA counter

LDR R1, =0x4819C000 @base address of I2C2

**ADD** R2, R1, #0x98 @offset of Data counter register

**MOV** R3, #0x02 @value to write 2 characters

STR R3, [R2] @store the value to count

@setting up the slave

**ADD** R2, R1, #0xAC @address of IC2C2 Slave register

**MOV** R5, #0x60 @value to have 60

STR R5, [R2] @store the value to the slave

@===================================================

**B** SET\_UP

@===================================================

**SET\_UP:**

LDR R1, =COUNTER

LDR R2, [R1]

**CMP** R2, #7

BNE SET\_UP2

@go to step

**B** SEND\_STEP\_LOOP

**SET\_UP2:**

BL DELAY

BL START

**B** INSTRUCTION

@===================================================

@infinite loop

**INF\_LOOP:**

**NOP**

**B** INF\_LOOP

@===================================================

**INSTRUCTION:**

@Send first byte

LDR R0, =NXP\_PTR @load the pointer

LDR R1, [R0] @read in the character

LDRB R4,[R1], #1 @increment the pointer

STR R1, [R0] @Store back the new pointer

LDR R5, =0x4819C09C @load in I2C2\_DATA

STRB R4, [R5] @Send data to I2C2\_DATA

@Send second byte

LDR R0, =NXP\_PTR @load the pointer

LDR R1, [R0] @read in the character

LDRB R4,[R1], #1 @increment the pointer

STR R1, [R0] @Store back the new pointer

LDR R5, =0x4819C09C @load in I2C2\_DATA

STRB R4, [R5] @Send data to I2C2\_DATA

@increment counter

LDR R1, =COUNTER @load in the counter

LDR R2, [R1] @get the value being store there

**ADD** R2, R2, #1 @Increment the value at COUNTER

STRB R2, [R1] @Store the new value back to COUNTER

**B** SET\_UP

@=============================

@stepping through

**SEND\_STEP\_LOOP:**

LDR R1, =STEP\_COUNT

LDR R2, [R1]

**CMP** R2, #200

BNE FULL\_STEP

BEQ INF\_LOOP

**FULL\_STEP:**

**B** STEP1

**STEP1:**

@LED4 HIGH

@LED3 LOW

@LED5 HIGH

@LED6 LOW

@make LED3\_ON\_H to full low

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x13 @LED3\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x00 @value to make LED3 full low

STR R2, [R1] @Send the value to make LED to full on

@Make LED4\_ON\_H full high

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x17 @LED4\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x10 @value to make LED4 full high

STR R2, [R1] @Send the value to make LED to full on

@make LED5\_ON\_H full high

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x1B @LED5\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x10 @value to make LED5 full high

STR R2, [R1] @Send the value to make LED to full on

@make LED6\_ON\_H full low

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x1F @LED6\_ON\_H

STRB R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x00 @value to make LED6 full low

STRB R2, [R1] @Send the value to make LED to full low

BL DELAY2

**B** STEP2

**STEP2:**

@LED4 HIGH

@LED3 LOW

@LED5 LOW

@LED6 HIGH

@make LED3\_ON\_H to full low

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x13 @LED3\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x00 @value to make LED3 full low

STR R2, [R1] @Send the value to make LED to full on

@Make LED4\_ON\_H full high

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x17 @LED4\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x10 @value to make LED4 full high

STR R2, [R1] @Send the value to make LED to full on

@make LED5\_ON\_H full LOW

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x1B @LED5\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x00 @value to make LED5 full LOW

STR R2, [R1] @Send the value to make LED to full on

@make LED6\_ON\_H full HIGH

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x1F @LED6\_ON\_H

STRB R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x10 @value to make LED6 full HIGH

STRB R2, [R1] @Send the value to make LED to full low

BL DELAY2

**B** STEP3

**STEP3:**

@LED4 LOW

@LED3 HIGH

@LED5 LOW

@LED6 HIGH

@make LED3\_ON\_H to full HIGH

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x13 @LED3\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x10 @value to make LED3 full HIGH

STR R2, [R1] @Send the value to make LED to full on

@Make LED4\_ON\_H full LOW

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x17 @LED4\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x00 @value to make LED4 full LOW

STR R2, [R1] @Send the value to make LED to full on

@make LED5\_ON\_H full LOW

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x1B @LED5\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x00 @value to make LED5 full LOW

STR R2, [R1] @Send the value to make LED to full LOW

@make LED6\_ON\_H full HIGH

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x1F @LED6\_ON\_H

STRB R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x10 @value to make LED6 full HIGH

STRB R2, [R1] @Send the value to make LED to full HIGH

BL DELAY2

**B** STEP4

**STEP4:**

@LED4 LOW

@LED3 HIGH

@LED5 HIGH

@LED6 LOW

@make LED3\_ON\_H to full HIGH

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x13 @LED3\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x10 @value to make LED3 full HIGH

STR R2, [R1] @Send the value to make LED to full on

@Make LED4\_ON\_H full high

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x17 @LED4\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x00 @value to make LED4 full high

STR R2, [R1] @Send the value to make LED to full on

@make LED5\_ON\_H full high

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x1B @LED5\_ON\_H

STR R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x10 @value to make LED5 full high

STR R2, [R1] @Send the value to make LED to full on

@make LED6\_ON\_H full low

BL DELAY @delay until it ready for the next instruction

BL START @Start Polling

LDR R1, =0x4819C09C @load in I2C2\_DATA

LDR R2, =0x1F @LED6\_ON\_H

STRB R2, [R1] @send the data to I2C2\_DATA

LDR R2, =0x00 @value to make LED6 full low

STRB R2, [R1] @Send the value to make LED to full low

@increment counter

LDR R1, =STEP\_COUNT @load in the counter

LDR R2, [R1] @get the value being store there

**ADD** R2, R2, #1 @Increment the value at

STRB R2, [R1] @Store the new value back to

BL DELAY2

**B** SEND\_STEP\_LOOP

@============================

**START:**

**POLL\_BB:**

LDR R1, =0x4819C024 @base address of I2C2\_IRQSTATUS\_RAW

LDR R2, [R1] @load the value in

TST R2, #0x1000 @check if the bit is clear or set

BNE POLL\_BB @keep Polling if the Bus is busy

@reset I2C

LDR R1, =0x4819C0A4

LDR R2, =0x8600

STR R2, [R1]

@setting up DATA counter

LDR R1, =0x4819C000 @base address of I2C2

**ADD** R2, R1, #0x98 @offset of Data counter register

LDR R3, =0x02 @value to write 2 characters

STRB R3, [R2] @store the value to count

@============================

LDR R1, =0x4819C0A4 @base address of I2C2\_CON

LDR R2, =0x8603 @Turn on the STT and STP, STT = 1 and STP = 1, Start and Stop

STRB R2, [R1] @store the value to register

**POLL\_XRDY:**

LDR R0, =0x4819C024 @base address of I2C2\_IRQSTATUS\_RAW

LDR R2,[R0] @load in the address

TST R2, #0x10 @Check bit 4 for XRDY

BEQ POLL\_XRDY @if it not set then keep Polling

**MOV** PC, R14 @Return

**DELAY:**

**MOV** R1, #5000

**WAIT\_LOOP:**

SUBS R1,R1,#1

BNE WAIT\_LOOP

**MOV** PC, R14

**DELAY2:**

**MOV** R1, #11000

**WAIT\_LOOP2:**

SUBS R1, R1, #1

BNE WAIT\_LOOP2

**MOV** PC, R14

**.data**

**.align** 2

**STACK1:** .rept 1024

**.word** 0x0000

.endr

**STACK2:** .rept 1024

**.word** 0x0000

.endr

**.align** 2

**NXP\_INTRUCTION:**

**.byte** 0x00 @mode1

**.byte** 0x11

**.byte** 0x00 @mode1

**.byte** 0x81

**.byte** 0xFE @Pre\_scale

**.byte** 0x05

**.byte** 0x01 @mode2

**.byte** 0x04

**.byte** 0xFD @ALL\_LED\_OFF

**.byte** 0x00

**.byte** 0x0F @LED2\_ON\_H

**.byte** 0x10

**.byte** 0x23 @LED7\_ON\_H

**.byte** 0x10

**.align** 2

**NXP\_PTR:** **.word** NXP\_INTRUCTION

**COUNTER:** **.word** 0

**STEP\_COUNT:** **.word** 0

.end

**High Level algorithm part 1:**

1. Set up the Stack.
2. Wake up the I2C2 clock.
3. Program the Pre-scaler from 48 MHz to 12 MHz for the I2C module clock.
4. Find the set the Low time and High time for the I2C clock for it to work at 100Kps or 400Kps.
5. Configurate the I2C own address and then take it out of reset.
6. Initialize I2C2.
7. Configure Slave address and DATA counter register.
8. Set up the Transmission Procedure; check for if the bus is busy if it busy then wait till the bus is free, set the Start and Stop condition.
9. Wait for 0.2 second.
10. Transmitting the data.

**Low Level algorithm part 1:**

1. Set up the Stack.

* Point to the top of the stack with 0x1000.
* Switch to the IRQ mode using CPS #12.
* Then point to top of the stack in IRQ mode.
* Switch back to supervisor mode.

1. Wake up the I2C2 clock.

* Read in 0x02 to the CM\_PER\_I2C2\_CLKTRL (0x44E00044) register

1. Program the Pre-scaler from 48 MHz to 12 MHz for the I2C module clock.

* Set 48MHz to 12Mhz by reading in 0x03 to the I2C2\_PSC (0x4819C00B0) register.

1. Find the set the Low time and High time for the I2C clock for it to work at 100Kps or 400Kps.

* Set I2C2\_SCLL (0x4819C00B4) register to 400Kbps by reading 0x08 to it,
* Set I2C2\_SCHL (0x4819C00B8) register to 400Kbps by reading in 0x0A to it.

1. Configurate the I2C own address and then take it out of reset.

* Read in 0x00 to I2C2\_OA (0x4819C00A8) register to set it own address.
* Then set bit 15th, 10th and 9th bit in the I2C2\_CON (0x4819C00A4) register which is 0x8600.

1. Initialize I2C2.

* Set up Pin 19 and Pin 20 for the I2C2\_SCL/ I2C2\_SDA line by switching it to mode 3 and enable the PullUp and receiver.
* Read in 0x33 to the UART1\_CTSN (0x44E1097C) and UART1\_RSTN (0x44E10978).

1. Configure Slave address and DATA counter register.

* Make the slave address to be 0x60 by reading it to the I2C2\_SA (0x4819C00AC) register.
* Set the count in the I2C2\_CNT (0x4819C0098) to 2 bytes only.

1. Set up the Transmission Procedure; check for if the bus is busy if it busy then wait till the bus is free, set the Start and Stop condition, wait for 0.2 seconds before transmission.

* Read in the data from I2C2\_IRQSTATUS\_RAW (0x4819C0024) register and test the BB bit to see if it a 1 or 0.
* If the bus is not busy, then move on else keep polling till it not busy.
* If the bus not busy, then set the STT and STP in the I2C2\_CON (0x4819C0A4) and then move on.
* Check for the XRDY bit to see if it is clear if it is not clear then keep waiting otherwise go to a delay loop.

1. Wait for 0.2 second.
2. Transmitting the data.

* Send a byte to I2C2\_DATA (0x4819C009C) register

**High Level algorithm part 2:**

**Initializing I2C2:**

1. Set up the Stack.
2. Wake up the I2C2 clock.
3. Program the Pre-scaler from 48 MHz to 12 MHz for the I2C module clock.
4. Find the set the Low time and High time for the I2C clock for it to work at 100Kps or 400Kps.
5. Configurate the I2C own address and then take it out of reset.
6. Initialize I2C2.
7. Configure Slave address and DATA counter register.

**Start the transmission:**

1. Set up the Transmission Procedure; check for if the bus is busy if it busy then wait till the bus is free, set the Start and Stop condition.
2. Then check the XRDY bit to see if it is set, if not set then keep on waiting otherwise move on.

**Send the instructions to PCA:**

1. Load in the value in the counter pointer and check it value to see if it equal to 7, if it not equals to 7 then go to SEND\_INTRUCTION else go SEND\_STEP.
2. Load in the instruction pointer and read in the value at the first position.
3. Increase the pointer by 1 and store it back.
4. Sent the value to the I2C2 bus line.
5. Load in the count pointer and read in the value.
6. Increase the value by 1 and then store it back to the count pointer.
7. Call the delay loop and then call START function. Keep looping till the value in the count is 7.

**Send Full Step sequence:**

1. Load in the value in the step\_count pointer and check to see if it 200, if it is then go INF\_LOOP else go FULL\_STEP.

**Sending Full Step:**

1. For step1 make LED3 to low, LED4 to high, LED5 to high and LED6 to low.
2. For step2 make LED3 to low, LED4 to high, LED5 to low and LED6 to high.
3. For step3 make LED3 to high, LED4 to low, LED5 to low and LED6 to high.
4. For step4 make LED3 to high, LED4 to low, LED5 to high and LED6 to low.

**Low Level algorithm Part 2:**

**Initializing I2C2:**

1. Set up the Stack.

* Point to the top of the stack with 0x1000.
* Switch to the IRQ mode using CPS #12.
* Then point to top of the stack in IRQ mode.
* Switch back to supervisor mode.

1. Wake up the I2C2 clock.

* Read in 0x02 to the CM\_PER\_I2C2\_CLKTRL (0x44E00044) register

1. Program the Pre-scaler from 48 MHz to 12 MHz for the I2C module clock.

* Set 48MHz to 12Mhz by reading in 0x03 to the I2C2\_PSC (0x4819C00B0) register.

1. Find the set the Low time and High time for the I2C clock for it to work at 100Kps or 400Kps.

* Set I2C2\_SCLL (0x4819C00B4) register to 400Kbps by reading 0x08 to it,
* Set I2C2\_SCHL (0x4819C00B8) register to 400Kbps by reading in 0x0A to it.

1. Configurate the I2C own address and then take it out of reset.

* Read in 0x00 to I2C2\_OA (0x4819C00A8) register to set it own address.
* Then set bit 15th, 10th and 9th bit in the I2C2\_CON (0x4819C00A4) register which is 0x8600.

1. Initialize I2C2.

* Set up Pin 19 and Pin 20 for the I2C2\_SCL/ I2C2\_SDA line by switching it to mode 3 and enable the PullUp and receiver.
* Read in 0x33 to the UART1\_CTSN (0x44E1097C) and UART1\_RSTN (0x44E10978).

1. Configure Slave address and DATA counter register.

* Make the slave address to be 0x60 by reading it to the I2C2\_SA (0x4819C00AC) register.
* Set the count in the I2C2\_CNT (0x4819C0098) to 2 bytes only.

**Start the transmission:**

1. Set up the Transmission Procedure; check for if the bus is busy if it busy then wait till the bus is free, set the Start and Stop condition.

* Check for the BB bit in the I2C\_IRQSTATUS\_RAW register (0x4819C0024).
* Read 0x8603 into I2C\_CON register (0x4819C0A4) to set the STT and STP condition.

1. Then check the XRDY bit to see if it is set, if not set then keep on waiting otherwise move on.

* Check the XRDY bit in the I2C\_IRQSTATUS\_RAW (0x4819C0024) register to see if it 1 or not.

**Send the instructions to PCA:**

1. Load in the value in the counter pointer and check it value to see if it equal to 7, if it not equals to 7 then go to SEND\_INTRUCTION else go SEND\_STEP.

* Load in a value in =COUNTER pointer and compare it to #7.
* If it not equal, then BNE to SEND\_INTRUCTION.
* Else then BEQ SEND\_STEP.

1. Load in the instruction pointer and read in the value at the first position.

* Load in value from the NXP\_PTR by LDR R0, =NXP\_PTR.

1. Increase the pointer by 1 and store it back.

* LDRB R2, [R1], #1 to increment the pointer by 1.
* STRB R1, [R0] to store it back.

1. Sent the value to the I2C2 bus line.

* Send the data to I2C\_DATA (0x4819C009C).

1. Load in the count pointer and read in the value.

* Load in the COUNTER pointer, LDR R1, =COUNTER.

1. Increase the value by 1 and then store it back to the count pointer.

* ADD R2, R2, #1 to increment.
* STRB R2, [R1] to store back the value.

1. Call the delay loop and then call START function. Keep looping till the value in the count is 7.

**Send Full Step sequence:**

1. Load in the value in the step\_count pointer and check to see if it 200, if it is then go INF\_LOOP else go FULL\_STEP.

* LDR R1, =STEP\_COUNT to load in the count value.
* CMP R1, #200 check to see if it 200.
* BNE FULL\_STEP if it not.
* Else BEQ INF\_LOOP.

**Sending Full Step:**

1. For step1 make LED3 to low, LED4 to high, LED5 to high and LED6 to low.

* To set LED3 low, send 0x13 to the I2C\_DATA for address of the LED3 then send 0x00 to make LED3 low.
* To set LED4 high, send 0x17 to the I2C\_DATA for address of the LED4 then send 0x10 to make LED4 high.
* To set LED5 high, send 0x1B to the I2C\_DATA for address of the LED5 then send 0x10 to make LED5 high.
* To set LED6 high, send 0x1F to the I2C\_DATA for address of the LED6 then send 0x00 to make LED6 low.

1. For step2 make LED3 to low, LED4 to high, LED5 to low and LED6 to high.

* To set LED3 low, send 0x13 to the I2C\_DATA for address of the LED3 then send 0x00 to make LED3 low.
* To set LED4 high, send 0x17 to the I2C\_DATA for address of the LED4 then send 0x10 to make LED4 high.
* To set LED5 low, send 0x1B to the I2C\_DATA for address of the LED5 then send 0x00 to make LED5 low.
* To set LED6 high, send 0x1F to the I2C\_DATA for address of the LED6 then send 0x10 to make LED6 high.

1. For step3 make LED3 to high, LED4 to low, LED5 to low and LED6 to high.

* To set LED3 high, send 0x13 to the I2C\_DATA for address of the LED3 then send 0x10 to make LED3 high.
* To set LED4 low, send 0x17 to the I2C\_DATA for address of the LED4 then send 0x00 to make LED4 low.
* To set LED5 low, send 0x1B to the I2C\_DATA for address of the LED5 then send 0x00 to make LED5 low.
* To set LED6 high, send 0x1F to the I2C\_DATA for address of the LED6 then send 0x10 to make LED6 high.

1. For step4 make LED3 to high, LED4 to low, LED5 to high and LED6 to low.

* To set LED3 high, send 0x13 to the I2C\_DATA for address of the LED3 then send 0x10 to make LED3 high.
* To set LED4 low, send 0x17 to the I2C\_DATA for address of the LED4 then send 0x00 to make LED4 low.
* To set LED5 high, send 0x1B to the I2C\_DATA for address of the LED5 then send 0x10 to make LED5 high.
* To set LED6 high, send 0x1F to the I2C\_DATA for address of the LED6 then send 0x00 to make LED6 low.