26/06/23

Time: 7.5h

Monday, June 26, 2023 8:03 AM

Start 8:00AM

I brought in a thumbdrive to pull the scripts off the old PI OS to put onto the new one — I also brought a microUSB reader to flash the new OS

Ximea Drivers/Software

From looking at the Ximea page on the RPI 4, I can see that they tested using Ubuntu 21.04 (64bit). I will install this version of Ubuntu onto the PI.

Eventually the server version without a GUI will likely be better to use, but I will use the desktop version as it will likely be easier to test/debug issues

I got the OS flashed onto the SD card. I want to grab the test files off of the old PI to run on the new one for testing.

I tried to boot the RPI 4 with the new OS, however it failed to boot…

I am going to install a newer distro that is included in the RPI imager software. This is Ubuntu 22.04.2

I wrote the OS to the SD and booted the PI. It did boot properly. Not I have to deal with the fun of connecting to the internet. It is supposed to open a page automatically to direct me to download the eduroam software.

But it is refusing to do that. And I cant used the Setup wifi to do anything other than wait for the prompt, full access to the internet is blocked. So I cant even download the installer myself

I finally got the package installed and have the eduroam network connected to the pi.

I am going to start installing software that I need to keep testing and developing on the new board

I installed the GCC compiler

I am going to install some of the I2C libraries I tried to use but ran into issues.

I got the GCC compiler installed.

I copied over the test files as well from the other PI

I realized I made a mistake before and the reason the compiler was not finding the i2c-dev.h file was because I typed i2d-dev.h…

I fixed that and the I2C script will compile

I have to run the compiled file with sudo to be able to access the I2C bus file

When I run the file, it runs properly, however it does not actually read any information from the arduino.

I opened the arduino IDE to view the arduino serial monitor, and when I run the script on the PI the serial moniter states that the arduino is writing to the PI.

IDK if there is a data formatting issue, or a timing issue or what is wrong

Lunch 12:00 - 12:30

I tried to change the slave address to make sure that the I2C script would only cause the arduino to write if the address matched.

I did this and it did no write anymore which is good. However when I changed it back to the matching address of 0x03, it no longer prints anything out.

I am not sure what I broke.

The i2cdetect -y 1 command is now showing a device at all available addresses. That does not make sense, it used to but a -- in all the addresses cause no devices were connected…

I unplugged the GND to the arduinos so they powered down.

I reran the command and found that all the device addresses reset to the -- that I expected.

When I reconnected the arduinos, I changed the address to 0x13.

When I ran the detect command, nothing came up….

I checked the circuit and the resistor on the clock line was pretty lose in the breadboard, I folded the leg over to make it thicker and reconnected it.

Now device 0x13 is showing on the detect but the communication is still not working.

The jumper connecting 5V fell off……

I now get a reading from the device however it is just strange characters…

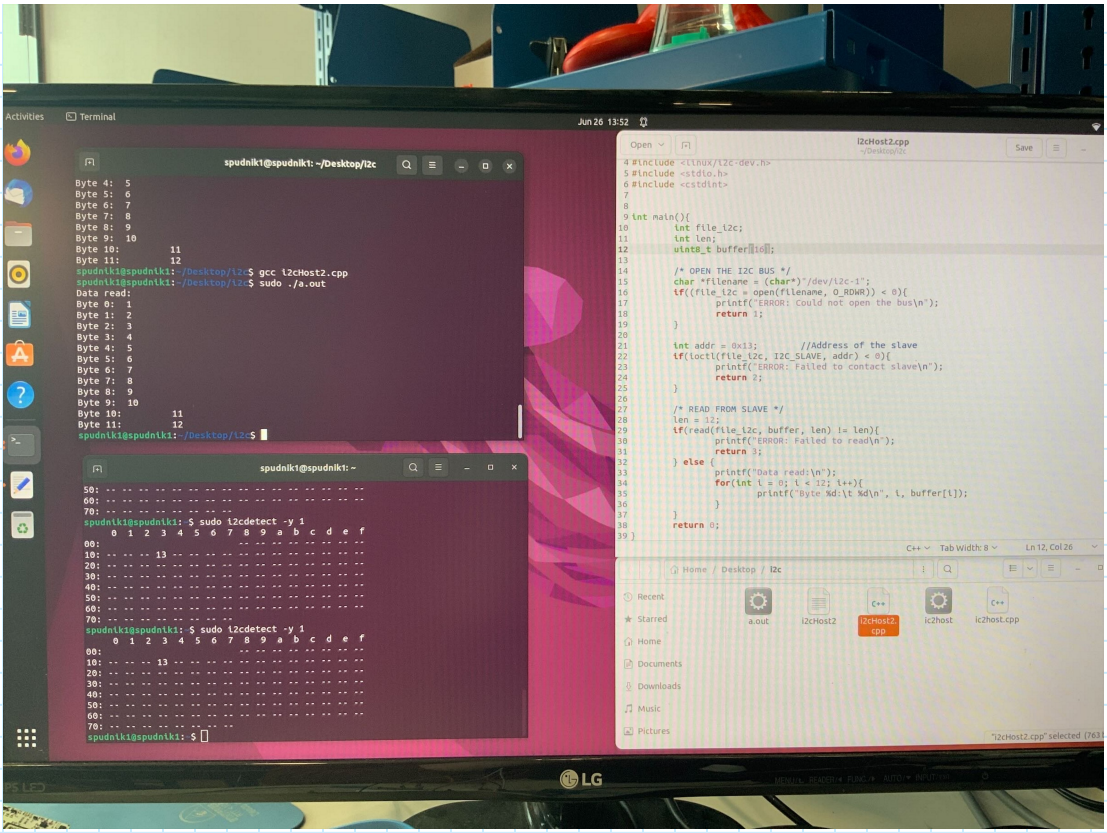
When I change the arduino to write the letter T it works

So I think there is just a decoding thing being done to my hex data which is causing it to give weird characters.

I changed the buffer to an uint8_t array

I also changed the print to print out one buffer value at a time.

It is working perfectly now

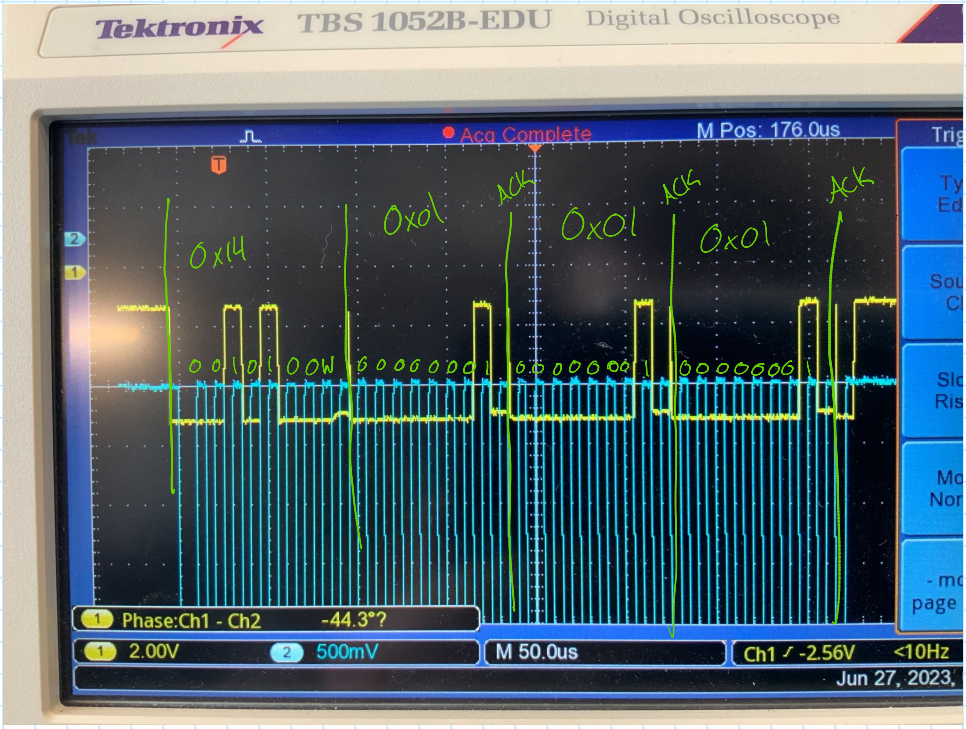



```arduino
void reportADC(){
  Serial.println("CALLED");
  uint16_t readings[12] = {
    0x0001,
    0x0002,
    0x0003,
    0x0004,
    0x0005,
    0x0006,
    0x0007,
    0x0008,
    0x0009,
    0x000A,
    0x000B,
    0x000C
  };
  for(int i = 0; i < 12; i++){
    Wire.write(readings[i]);
    Serial.print("Writing reading ");
    Serial.println(i+1);
  }
}
```

I can read the data properly

I added the example for the writing data. It is not going as well. I am not even getting a response on the Arduino to say that it is being accessed.

To me this means that the I2C is not actually being written, but I have no idea how to debug this issue as its only essentially one line of code



I connected the scope to the SDA (Ch1) and SCL (Ch2) lines

The script is supposed to address 0x14 with W command then write
0x01
0x01
0x01

Clearly this is writing to the I2C bus properly

I tried to read the read comand on the scope as well however I did not get any logical output

I installed the Ximea software and changed the USB memory default as instructed.
https://www.ximea.com/support/projects/apis/wiki/Raspberry_Pi_4_Benchmarks

Leave 4:00

27/06/23

Time: 2.75h

Tuesday, June 27, 2023 7:57 AM

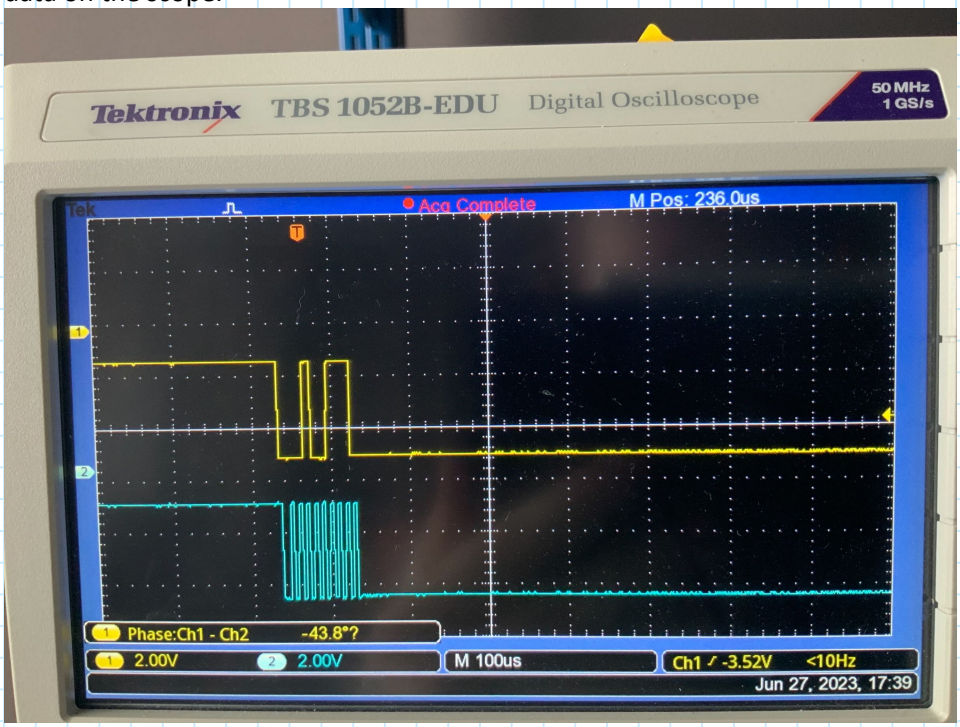
Start 8:00 AM

At the end of the day yesterday I was working on the I2C on the RPI. I was able to get the read to work with an arduino and I got the data I expected.

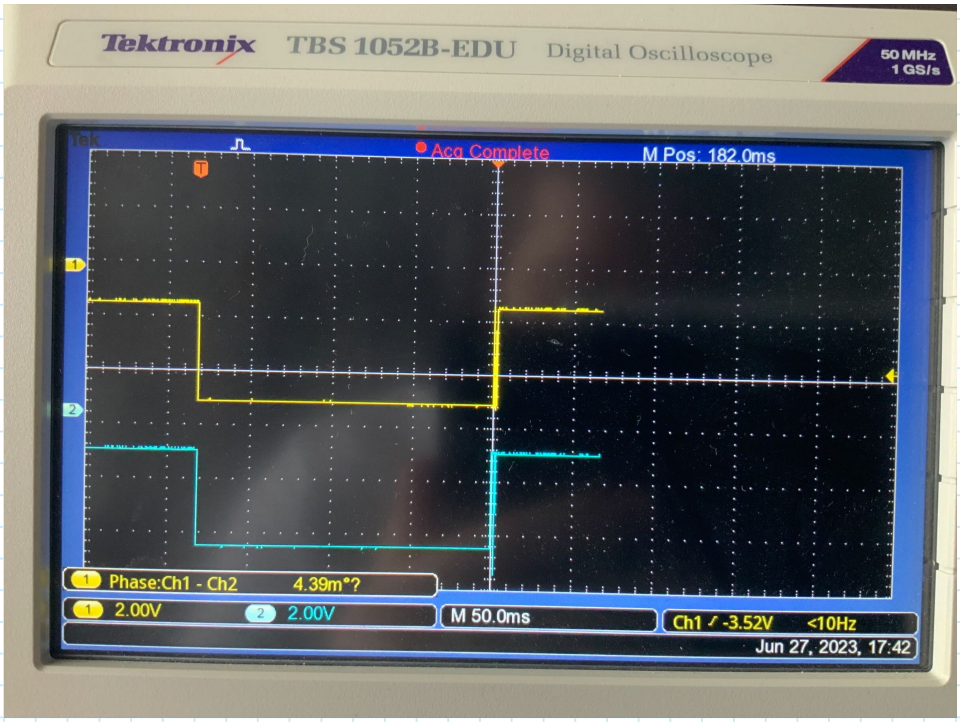
The write however did not work as well. I connected it to the scope and I was able to see the digital logic for the write command. I found that the logic on the SDA and SCL lines were what I was expecting with the data I told it to write.

I tried to connect the scope for the read command and I found that It was not reading the lines properly -- likely that the timing was too long for me to see all the transaction, but even the beginning did not match what I expected to see.

I may try to understand why that isnt working and get a good reading on the read command to see the data on the scope.

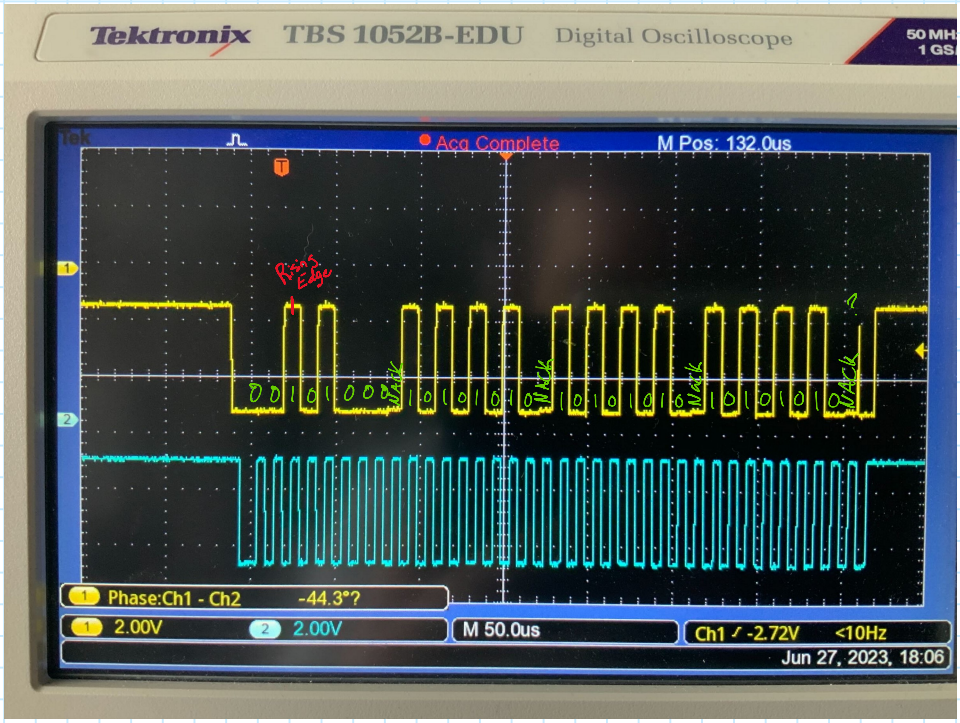


The read command is sending the correct address at the beginning but then both the clock and data line are held low -- I am not sure if this is being done by the PI or on the Arduino. The client may be stretching the clock to write to the bus



After ~180ms the lines are released.

Back to the write command -- Nick mentioned that the ACK bits look weird. Because they are just a small pulse on the falling edge. That made me think that the Arduino should be the one generating the ACK bit as it is the one receiving the data. But the Arduino is never acknowledging that it is being called in the first place so I would not expect it to be generating any ACK signals…



Write
0xAA
0xAA
0xAA
Lo
0x14

I tried to change the output to not write any data but just all the Arduino. There is no form of acknowledge signal from the arduino after the address, maybe I can check for this in the read signal to see cause I know that it is communicating with the arduino.

I tried changing the read values that the Arduino is supposed to send back, and this is not actually working… Somehow it happened to show the 1-12 that I tested with, but if I change any of the numbers it breaks…

Even if I use a number like 0x000C (12) which I already used -- it will work in one position but not the others which really does not make sense.

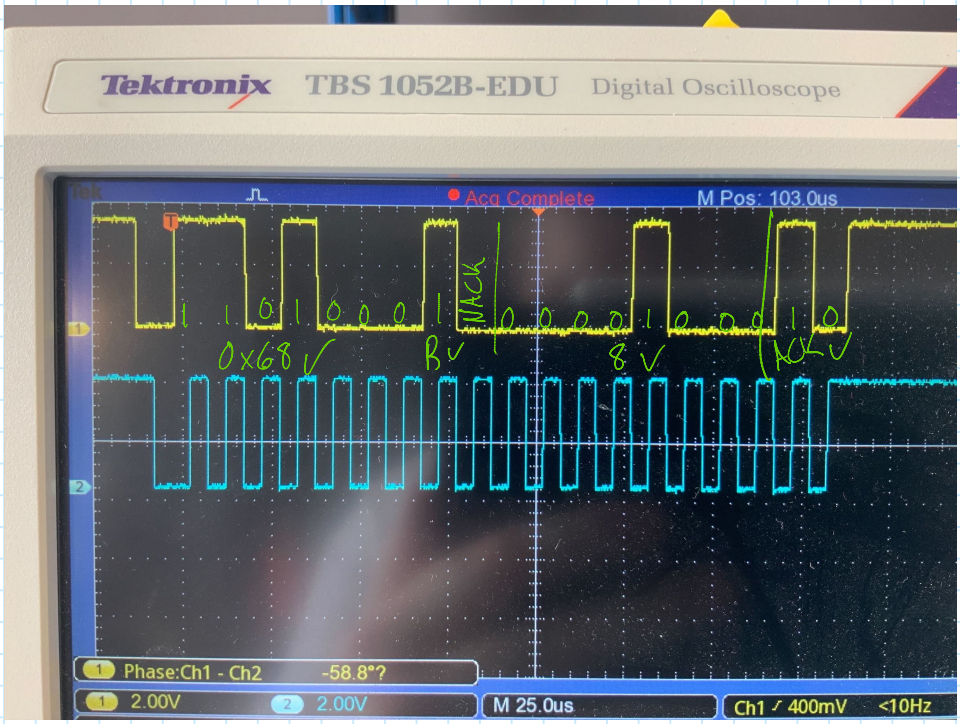
I was writing in 16bit, which is 2 bytes, so really I should be expecting 24 bytes for the 12 signals…..

Maybe that was the issue, and the software is only getting half a number in each positon

I went to Megan to get an I2C sensor to see if I can get a better signal on the scope. The read function is just holding the clock low so I do not understand where the data is coming from. And I am not sure if it is the Arduino holding the clock or the PI -- so maybe I can use an off the shelf I2C sensor to get a baseline of what I should be seeing, and see if it is the PI that has the problem or my arduino.

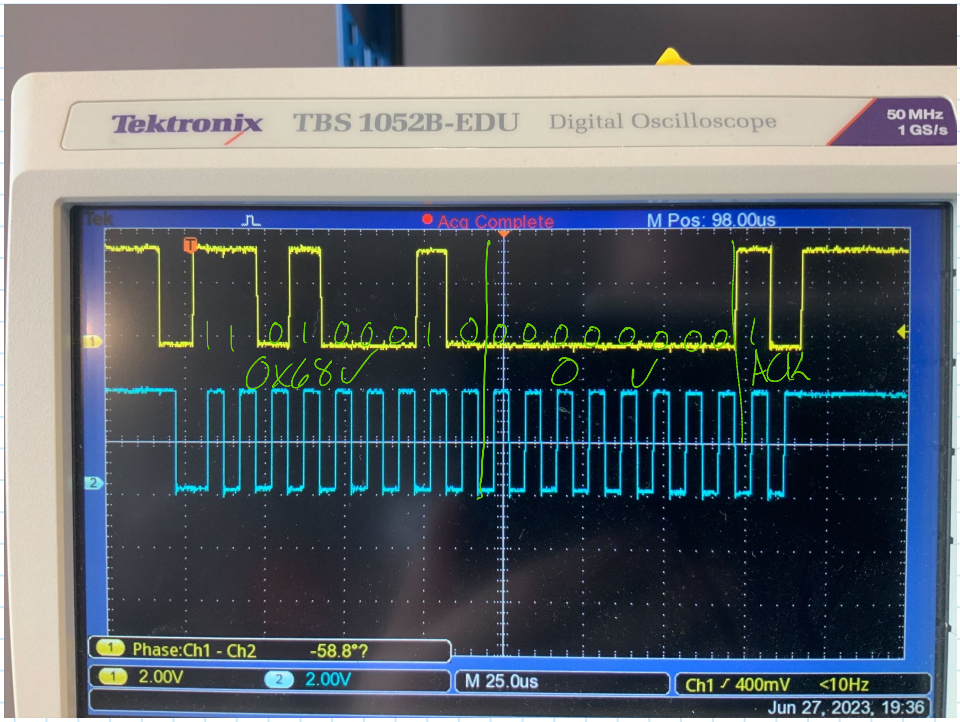
I got a 6 axis accelerometer/gyro - SEN0142

I can use the *sudo i2cdetect -y 1* command to find the address of the device -- which is 0x6



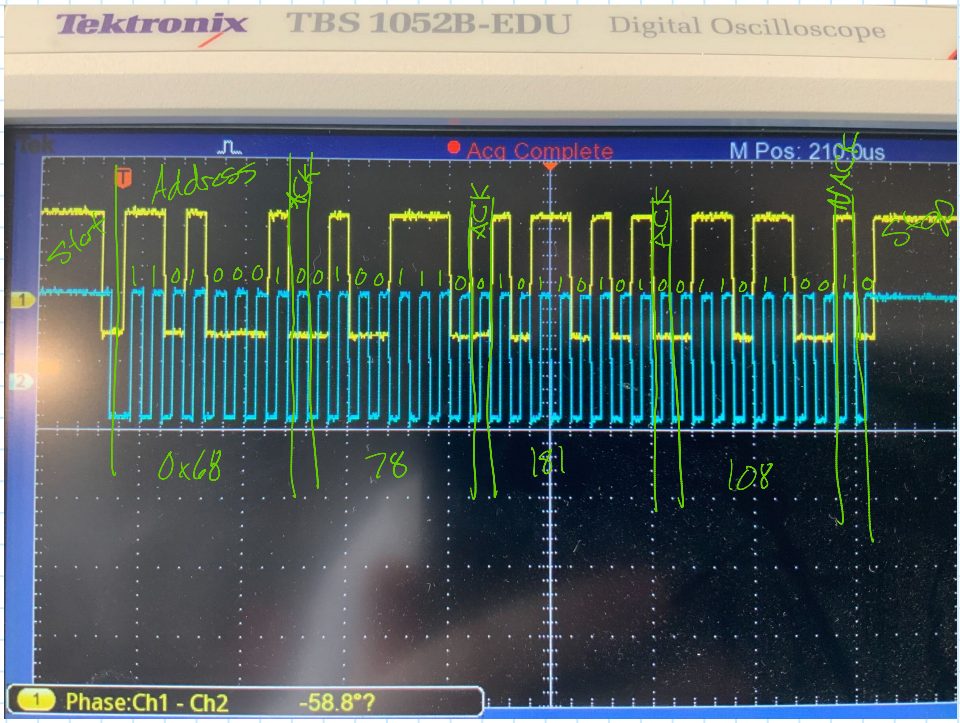
I read an 8 on the PI

So there is no ACK after the address, however there is one after the transmission



I read a 0 on the PI

Not sure why its reading 0, but the logic matches the reading so that is good



Expected result
78
181
108

So ACK appears to be low, and NACK is high and marks the end of transmission

leave 10:45AM

28/06/23 Time: 8h

June 28, 2023 8:04 AM

Start 8:00AM

Yesterday I had observed that the ACK bit in the transmission was being held low to send an ACK.

> Acknowledge (ACK)/Not Acknowledge (NACK) Sequence
>
> The I 2C specification defines the Acknowledge sequence as a logic low state of the SDA line during the 9th SCL pulse for any successfully transferred byte. During this time, the transmitter must relinquish control of the SDA line to the receiver.
>
> Microchip Technology
> https://onlinedocs.microchip.com › ...
>
> Acknowledge (ACK)/Not Acknowledge (NACK) Sequence

This is correct.

Also the last ACK was high, or NACK. However I remember reading in the PIC datasheet that a NACK is generated to indicate the end of the transmission.

So it appears that the device can read the I2C sensor properly, and whatever issues going on before was with the Arduino software and not the RPI.

For the write command, I already determined that the digital logic of the transmission appears correct, however there is question about the ACK/NACK bits. But it makes sense that they are not making sense whenever the client (Arduino) is the one that is supposed to generate these bits and it is not recognizing it is being called.

I am not sure if I can test a write command to the accelerometer. Maybe somewhere in the datasheet it will have a write command I can use to see if it works. Or at lease read the digital logic and make sure it appears to be correct.

The datasheet online for this sensor is useless. It is 4 pages and 2 of those pages are "sample" arduino code that does next to nothing. It does use a library for the sensor so maybe the intialize command in the library writes to the chip.

```
void MPU6050_Base::setFreefallDetectionThreshold(uint8_t threshold) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_FF_THR, threshold, wireObj);
}
```

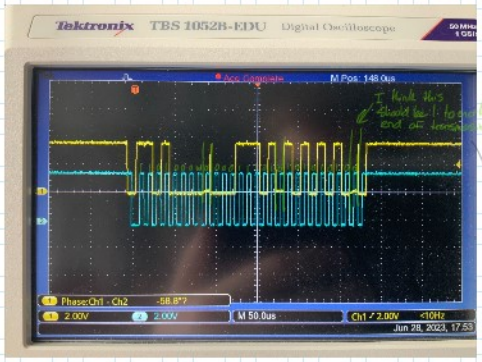
Address is 0x68

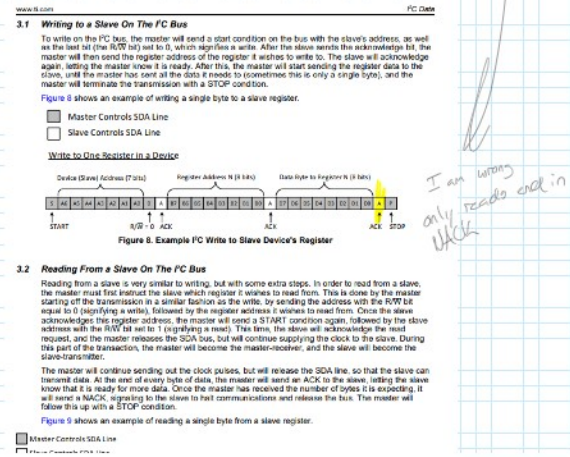
```
#define MPU6050_RA_FF_THR           0x1D
#define MPU6050_RA_FF_DUR           0x1F
```

```
// FF_THR register
uint8_t getFreefallDetectionThreshold();
void setFreefallDetectionThreshold(uint8_t threshold);
```

Theshold is uint8
wireObj is used as the last argument on every transmission. Im assuminng it is just something the I2C libarry used needs.

So if I transmit
0x1D 0xAA to 0x68 maybe I can get a reasonable response.





3.1 Writing to a Slave On The I²C Bus

To write on the I²C bus, the master will send a start condition on the bus with the slave's address, as well as the last bit (the R/W bit) set to 0, which signifies a write. After the slave sends the acknowledge bit, the master will then send the register address of the register it wishes to write to. The slave will acknowledge again, letting the master know it is ready. After this, the master will start sending the register data to the slave, until the master has sent all the data it needs to (sometimes this is only a single byte), and the master will terminate the transmission with a STOP condition.

Figure 8 shows an example of writing a single byte to a slave register.

Master Controls SDA Line

Slave Controls SDA Line

Write to One Register in a Device

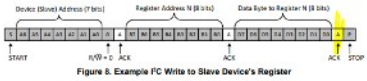


Figure 8. Example I²C Write to Slave Device's Register

I am wrong only reads out in NACK

3.2 Reading From a Slave On The I²C Bus

Reading from a slave is very similar to writing, but with some extra steps. In order to read from a slave, the master must first instruct the slave which register it wishes to read from. This is done by the master starting off the transmission in a similar fashion as the write, by sending the address with the R/W bit equal to 0 (signifying a write), followed by the register address it wishes to read from. Once the slave acknowledges the register address, the master will send a START condition again, followed by the slave address with the R/W bit set to 1 (signifying a read). This time, the slave will acknowledge the read request, and the master releases the SDA bus, but will continue supplying the clock to the slave. During this part of the transaction, the master will become the master-receiver, and the slave will become the slave-transmitter.

The master will continue sending out the clock pulses, but will release the SDA line, so that the slave can transmit data. At the end of every byte of data, the master will send an ACK to the slave, letting the slave know that it is ready for more data. Once the master has received the number of bytes it is expecting, it will send a NACK, signaling to the slave to halt communications and release the bus. The master will follow this up with a STOP condition.

Figure 9 shows an example of reading a single byte from a slave register.

Master Controls SDA Line

Slave Controls SDA Line

Read From One Register in a Device

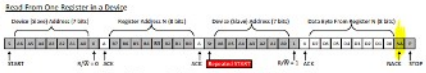


Figure 9. Example I²C Read from Slave Device's Register

Now I actually know the format of the data coming over the I2C bus — I remember initially I was confused as there was mention of a count register being used, but that clearly is not relevant as no other sources have used that

I should not say it is not relevant — but it is not inherent to I2C and it is not coming over the bus

I may review the datasheet now that I have a better understanding and try and rewrite some PIC code to match my understanding

I was thinking about how I was going to do the I2C on the actual ADCS software. Because there are multiple devices connected to the bus and I need to make sure that only one is being called at a time.

If everything I2C is being done in the same thread, I should just be able to make my class open and close the bus in each function -- this way the host cant call two devices at the same time or overlap transmissions.

The problem I see with this is that if any other things need to connect via I2C that are not part of the ADCS thread, I may call the function twice at the same time and try and send 2 commands over the bus.

Another solution would be to have the I2C as its own thread that can be accessed by the others, I would then need a queue system to manage when multiple sections of code call for the I2C at the same time.

Ideally that can be avoided as that is much more complex but .

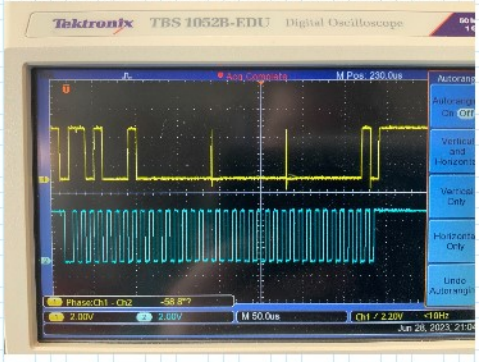
I think that currently these are the only devices on the I2C bus, however a battery management system is not in the current outline as Nick said we were dropping the monitering chips.

Another one of these PIC chips could be used to do that, and that may be outside of the ADCS thread. However as I do not expect it to be time critical, it may be able to be included in the thread to avoid the complexity of managing the I2C bus.

Looking at the ConOps spreadsheet that appears to be all that will be needed. I believe the S-band is on SPI and I was planning to connect the UHF to UART.

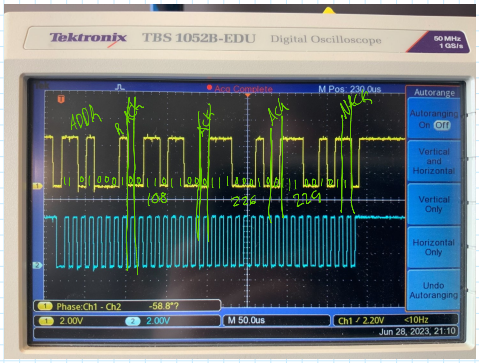
The S-band is SPI -- which is good because then it gets its own dedicated transmission line which will not limit the bandwidth.

IDK what I was saying above about making it open and close the bus. That should not matter cause it can only be called once at a time in the main thread so I can leave the bus open.



```c
#include "i2c-lib/i2c.h"
#include <cstdint>


int main(){

        i2c I2C("/dev/i2c-1");
        int addr = 0x68;
        int buffer = I2C.readBus(addr, 3);
        printf("READING");
        for(int i = 0; i < 3; i++){
                //printf("%d\t", buffer[i]);
        }
        printf("\n");
        I2C.closeBus();
        return 0;
}
```

I created a class for the I2C with simpler read write functions that should be easier to interface with in the main code. Attached is an example testing the read script and the data on the bus to see if it is working.



Currently the writeBus function is having compilation issues, but the readBus function appears to work properly.

hardwareInterfacing\i2c.cpp

```cpp
#include "i2c.h"

i2c::i2c(const char* i2cBus){
    char *filename = (char*)i2cBus;

    /*Open I2C Bus*/
    if((file_i2c = open(filename, O_RDWR)) < 0){
        printf("ERROR:\tFailed to open I2C bus.\n");
    }
}

uint8_t* i2c::readBus(uint8_t address, int length){
    if(ioctl(file_i2c, I2C_SLAVE, address) < 0){
        printf("ERROR:\t Failed to contact client");
    }
    if(read(file_i2c, buffer, length) != length){
        printf("ERROR:\tFailed to read bytes from client\n");
    }
    return buffer;
}

int i2c::writeBus(uint8_t address, int length, int buffer){

    if(ioctl(file_i2c, I2C_SLAVE, address) < 0){
        printf("ERROR:\t Failed to contact client");
        return 2;
    }
    /*
    if (write(file_i2c, buffer, length) != length)
    {
        printf("ERROR:\tFailed to write to slave.\n");
        return 4;
    }
    */
    return 0;
}

i2c::closeBus(){
    close(file_i2c);
}
```

hardwareInterfacing\i2c.h

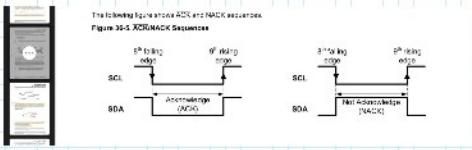
```cpp
#ifndef I2C_H
#define I2C_H

/*I2C libraries*/
#include <unistd.h>             //Needed for I2C port
#include <fcntl.h>              //Needed for I2C port
#include <sys/ioctl.h>          //Needed for I2C port
#include <linux/i2c-dev.h>      //Needed for I2C port

#include <cstdint>
#include <stdio.h>
#include <cstring>

class i2c {
    private:
        int addr;
        int file_i2c;
        char *filename;
        uint8_t buffer[60];

    public:
        void i2c(const char* i2cBus);
        uint8_t* readBus(uint8_t address, int length);
        int writeBus(uint8_t address, int length, int buffer);
        void closeBus();

};

#endif
```

E:\i2c-lib_TEST.cpp

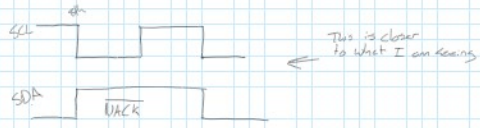
```cpp
#include "i2c-lib/i2c.h"
#include <cstdint>


int main(){

    I2c I2C("/dev/i2c-1");
    int addr = 0x68;
    uint8_t *buffer = I2C.readBus(addr, 8);
    printf("READING");
    for(int i = 0; i < 3; i++){
        printf("%d\t", buffer[i]);
    }
    printf("\n");
    I2C.closeBus();
    return 0;
}

```

30 min Lunch

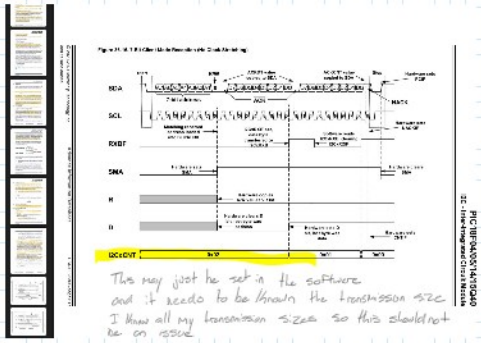
Review Datasheet on I2C again



Not exactly what I was seeing on the scope



The datasheet does not make it seem like the CNT is optional, but it also does not show the data coming in over the bus like I originally thought



I really want to just plug it in and monitor the registers and go one step at a time — like get the device to detect it is being called, and detect R\W, and see what happens in the Rx and Tx buffers

I think that is the only way to make real progress

Thinking about what I did previously with the I2C library, I am going to run into issues cause I want to use individual functions to control the PWM and ADC for each channel and those will not have access to a global I2C class that is not defined in the function.

For example, the structure is:

Main > setMotx > I2C
Main > setMoty > I2C

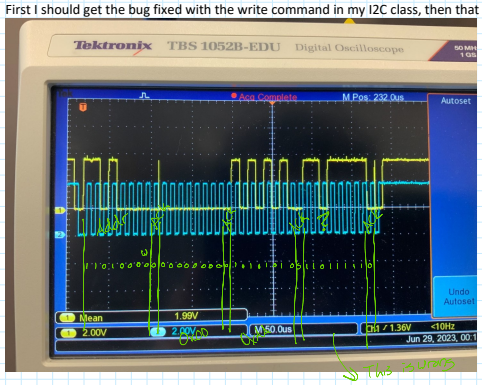
I can't use the same I2C class instance in the different sub functions

Maybe if I change the I2C to just a file with functions as opposed to a class I could use it in all the sub functions since the class instance would not be needed.

I might also be able to pass the I2C object to the functions that need it during initialization of the other classes.

So long as I define the I2C object at the beginning

First I should get the bug fixed with the write command in my I2C class, then that should all be functioning properly, and I can start making the functions to send the I2C signals, then I can figure out how to interface all of those things in I2C



Write command

Addr = 0x68

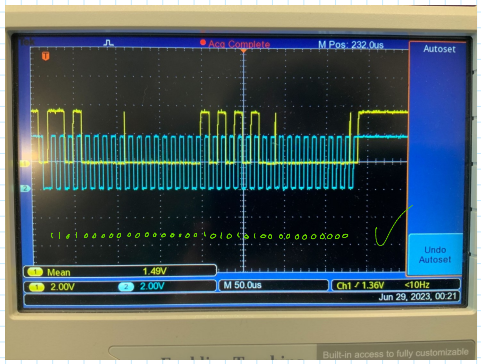
0x00
0xAA
0x00

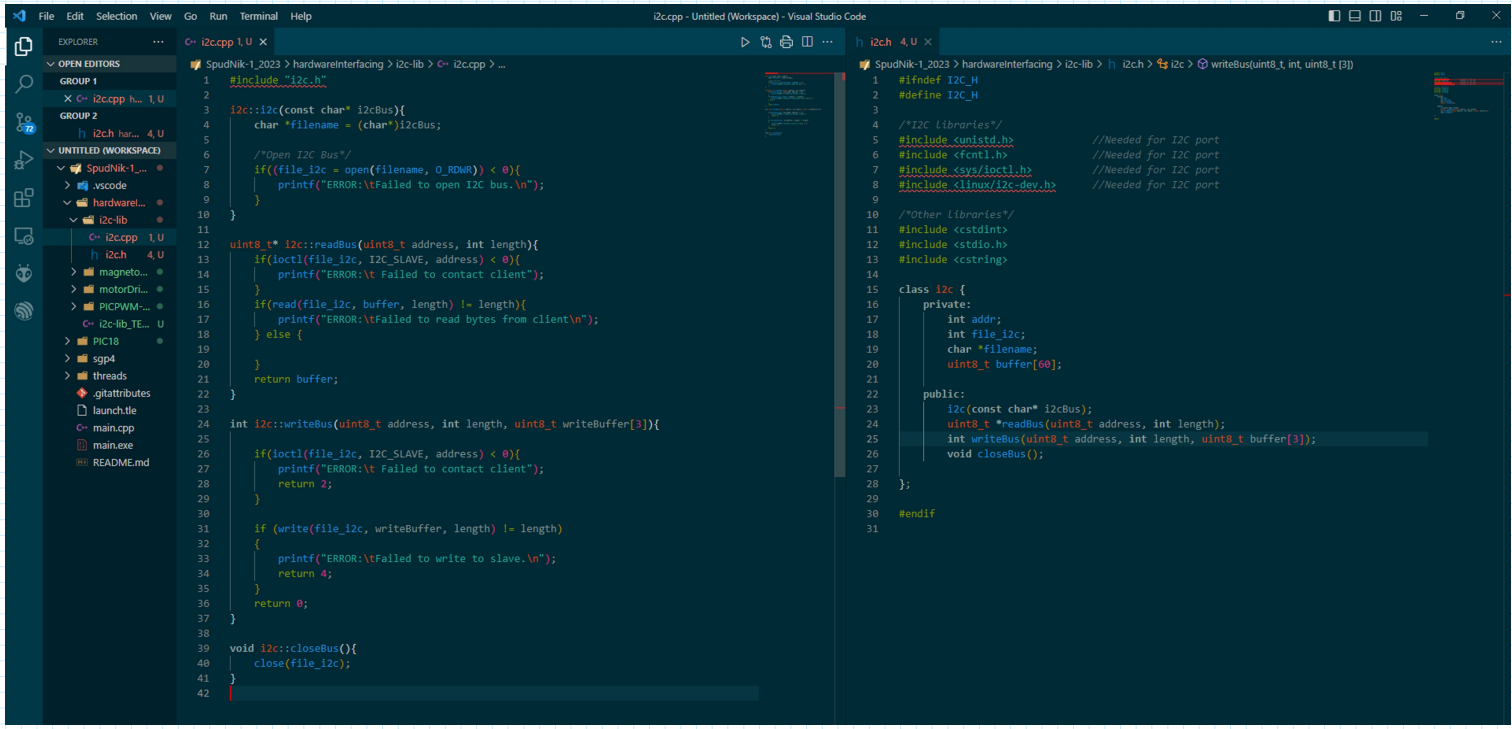
I fixed the compilation issue with the writeBus function. It was just datatype issues.

I tried to do a write command in the test script and it almost works, except the last byte is not correct....

Im going to try the exact same write in my other script not using the class to make sure that works.

Never mind, software is fine, I am dumb and indexed the message as byte0, byte1, byte3 instead of byte2. User error.





I reduced the buffer size to 3 bytes cause right now that is the biggest I need. Could be increased if things change.

Lenz 4:30

29/06/23

Thursday, June 29, 2023 8:02 AM

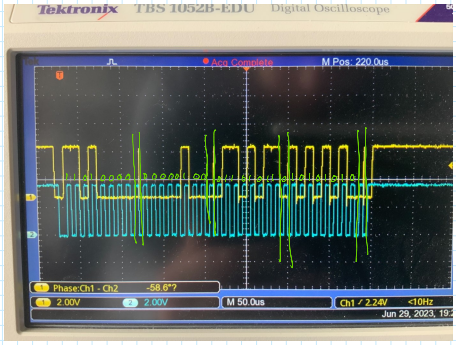
8AM start

I went back to revising my code

I documented everything and then put it onto the PI to test it.

Obviously it did not work right away. I went through error codes until it finally worked.

I also created a BASH script to compile it cause it is much easier than having to retype all the files required every time.

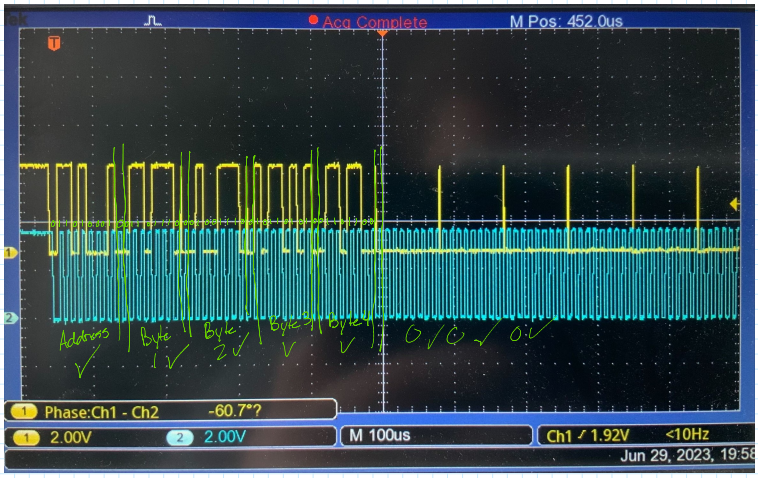


This was a command to write 0x04, 0x6B, 0xAA to 0x68

transmission should be

Address: 01101000
Byte 1: 00000100
Byte 2: 01101011
Byte 3: 10101010

IT IS RIGHT!!



I think my library actually Works!!!!!!!

```
spudnik1@spudnik1:~/Desktop/PWMADCtest$ sudo ./a.out
        Sun Sensor Readings
==================================
Reading 1:      28238
Reading 2:      46444
Reading 3:      0
Reading 4:      0
Reading 5:      0
Reading 6:      0
Reading 7:      0
Reading 8:      0
Reading 9:      0
Reading 10:     0
Reading 11:     0
Reading 12:     0
==================================
spudnik1@spudnik1:~/Desktop/PWMADCtest$
```

Address: 0x68 - R → 01101001

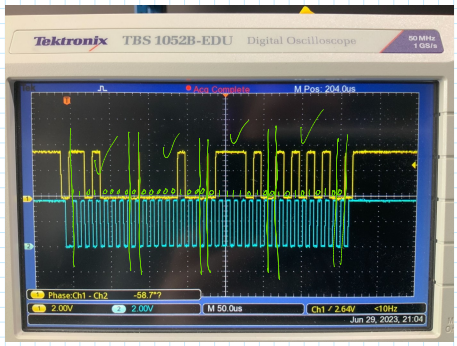
Reading 1: 28238 → 01101110
01001110

Reading 2: 46444 → 10110101
01011100

then 0's

2.260 ms − 20 µs = 2.26ms − 0.02ms

2.24ms for ADS read



pwm. set MagZ (0xAA)

Transmission should be

0x68 → 01101000

0x04 → 00000100

0x7A → 01111010

0xAA → 10101010

i2c.cpp

i2c.h

vault\TEST_picPWM-lib_picADC-lib_i2c-lib\i2c-lib\i2c.cpp

```cpp
#include "i2c.h"

i2c::i2c(const char* i2cBus){
    /* Constructor
        Parameter:
            i2cBus  -   Name of the file used to read and write to the I2C bus.
                        This parameter has a default value of "/dev/i2c-1"
    */

    char *filename = (char*)i2cBus; //Declare pointer to the I2C bus file

    /*Open I2C Bus*/
    if((file_i2c = open(filename, O_RDWR)) < 0){
        printf("ERROR:\tFailed to open I2C bus.\n");
    }
}

uint8_t* i2c::readBus(uint8_t address, int length){
    /*
        readBus(uint8_t address, int length) - This method is used to read from a client on the I2C bus.
        Parameters:
            address     -   7bit address of the client to read data from
            length      -   Length of the transmission in bytes

        Return:
            buffer      -   pointer to a uint8_t array containing the bytes read from the I2C bus
    */

   //Contact the client
    if(ioctl(file_i2c, I2C_SLAVE, address) < 0){
        printf("ERROR:\t Failed to contact client");
    }

    //Read data from the client
    if(read(file_i2c, buffer, length) != length){
        printf("ERROR:\tFailed to read bytes from client\n");
    }
    return buffer;
}

int i2c::writeBus(uint8_t address, int length, uint8_t writeBuffer[3]){
    /*
    writeBus(uint8_t address, int length, uint8_t buffer[n]) - This method is to write to a client on the I2C bus
            Parameters:
                address     -   7bit address of the client to read data from
                length      -   Length of the transmission in bytes
                buffer      -   array containing the bytes to be written to the bus. Index 0 of the array will be the first byte transmitted

            Return:
```

vault\TEST_picPWM-lib_picADC-lib_i2c-lib\i2c-lib\i2c.h

```cpp
#ifndef I2C_H
#define I2C_H

/*I2C libraries*/
#include <unistd.h>             //Needed for I2C port
#include <fcntl.h>              //Needed for I2C port
#include <sys/ioctl.h>          //Needed for I2C port
#include <linux/i2c-dev.h>      //Needed for I2C port

/*Other libraries*/
#include <cstdint>
#include <stdio.h>
#include <cstring>

class i2c {
    /*  This class is to provide a simple I2C interface to allow data to be
        written to and read from the bus.

        The constructor takes a single argument with is the I2C bus file location
        This is set to a default value of "/dev/i2c-1" which should be correct for the RPI

        The class has three methods:

        readBus(uint8_t address, int length) - This method is used to read from a client on the I2C bus.
            Parameters:
                address     -   7bit address of the client to read data from
                length      -   Length of the transmission in bytes

            Return:
                buffer      -   pointer to a uint8_t array containing the bytes read from the I2C bus

        writeBus(uint8_t address, int length, uint8_t buffer[n]) - This method is to write to a client on the I2C bus
            Parameters:
                address     -   7bit address of the client to read data from
                length      -   Length of the transmission in bytes
                buffer      -   array containing the bytes to be written to the bus. Index 0 of the array will be the first byte transmitted

            Return:
                0           -   The transmission was successful
                ~0          -   An error occured

        closeBus() - This method is to close the I2C bus
     */
    private:
        /* Declare private variables */
        int file_i2c;       //Variable to read and write to the I2C file
        char *filename;     //Pointer to the I2C bus file
        uint8_t buffer[24]; //Read buffer variable
```

```
                0           -   The transmission was successful
                ~0          -   An error occured
    */

    //Contact the client on the I2C bus
    if(ioctl(file_i2c, I2C_SLAVE, address) < 0){
        printf("ERROR:\t Failed to contact client");
        return 2;
    }

    //Write the bytes from the buffer to the device on the bus
    if (write(file_i2c, writeBuffer, length) != length)
    {
        printf("ERROR:\tFailed to write to slave.\n");
        return 4;
    }

    return 0;
}

void i2c::closeBus(){
    /*closeBus() - This method is to close the I2C bus
    */
    close(file_i2c);
}
```

```
    public:
    /* Delcare methods */
        i2c(const char* i2cBus = "/dev/i2c-1");
        uint8_t *readBus(uint8_t address, int length);
        int writeBus(uint8_t address, int length, uint8_t buffer[3]);
        void closeBus();

};

#endif
```

1:40 lunch
Back 2:10

I spoke with Nick. He mentioned that I should be able to pull the bus up to 3.3V and connect the MCU to the bus without any converters. This makes sense so I tried it.

I connected the MCU to the bus and ran script to see what worked. Nothing works. Reading the datasheet, there is supposed to be a flag that is set by the hardware whenever a start condition is detected on the bus. This should be a good place to start as the addressing, data, or anything else should not matter.

I used to debug to monitor the bit to see if it was being set whenever I ran the PI script. Nothing.

I have had issues in the past with the debugger where it does not like live updating values, it does better with a break in the code.

I put in the main loop an if condition that checks the bit, and if it is set, there is a breakpoint so the debugger will hault the code.

Again this never triggers.

The datasheet saus that this is supposed to be entirely done in hardware, none of my software should matter. The only thing that I can see making a difference would be the pins not being configured properly.

Note:

1. Bidirectional pin. The corresponding input must select the same pin.

This note is under the PPS output table for the I2C pins

I guess they need to be configured as an input as well.

PPS - Peripheral Pin Select Module

.....continued

Peripheral	PPS Input Register	14-Pin Devices					20-Pin Devices				
		Default Pin Selection at POR	Register Reset Value at POR	Available Input Port			Default Pin Selection at POR	Register Reset Value at POR	Available Input Port		
SPI2 Client Select	SPI2SSPPS	RA0	'b000 000	A	—	C	RA1	'b000 001	A	B	C
I2C1 Clock	I2C1SCLPPS(1)	RC0	'b010 000	A	—	C	RB6	'b001 110	A	B	C
I2C1 Data	I2C1SDAPPS(1)	RC1	'b010 001	A	—	C	RB4	'b001 100	A	B	C
UART1 Receive	U1RXPPS	RC5	'b010 101	A	—	C	RB5	'b001 101	A	B	C
UART1 Clear to Send	U1CTSPPS	RC4	'b010 100	A	—	C	RB7	'b001 111	A	B	C
UART2 Receive	U2RXPPS	RC1	'b010 001	A	—	C	RC1	'b010 001	A	B	C
UART2 Clear to Send	U2CTSPPS	RC2	'b010 010	A	—	C	RC2	'b010 010	A	B	C
UART3 Receive	U3RXPPS	RA4	'b000 100	A	—	C	RC3	'b010 011	A	B	C
UART3 Clear to Send	U3CTSPPS	RA5	'b000 101	A	—	C	RC5	'b010 101	A	B	C

.....continued

Address	Name	Bit Pos.	7	6	5	4	3	2	1	0
0x025B	CWG1PPS	7:0				PORT[2:0]			PIN[2:0]	
0x025C ... 0x025D	Reserved									
0x025E	MD1CARLPPS	7:0				PORT[2:0]			PIN[2:0]	
0x025F	MD1CARHPPS	7:0				PORT[2:0]			PIN[2:0]	
0x0260	MD1SRCPPS	7:0				PORT[2:0]			PIN[2:0]	
0x0261	CLCIN0PPS	7:0				PORT[2:0]			PIN[2:0]	
0x0262	CLCIN1PPS	7:0				PORT[2:0]			PIN[2:0]	
0x0263	CLCIN2PPS	7:0				PORT[2:0]			PIN[2:0]	
0x0264	CLCIN3PPS	7:0				PORT[2:0]			PIN[2:0]	
0x0265 ... 0x0268	Reserved									
0x0269	ADACTPPS	7:0				PORT[2:0]			PIN[2:0]	
0x026A	SPI1SCKPPS	7:0				PORT[2:0]			PIN[2:0]	
0x026B	SPI1SDIPPS	7:0				PORT[2:0]			PIN[2:0]	
0x026C	SPI1SSPPS	7:0				PORT[2:0]			PIN[2:0]	
0x026D	SPI2SCKPPS	7:0				PORT[2:0]			PIN[2:0]	
0x026E	SPI2SDIPPS	7:0				PORT[2:0]			PIN[2:0]	
0x026F	SPI2SSPPS	7:0				PORT[2:0]			PIN[2:0]	
0x0270	I2C1SDAPPS	7:0				PORT[2:0]			PIN[2:0]	
0x0271	I2C1SCLPPS	7:0				PORT[2:0]			PIN[2:0]	
0x0272	U1RXPPS	7:0				PORT[1:0]			PIN[2:0]	
0x0273	U1CTSPPS	7:0				PORT[1:0]			PIN[2:0]	
0x0274	UxRXPPS	7:0					PORT		PIN[2:0]	
0x0275	UxCTSPPS	7:0					PORT		PIN[2:0]	
0x0276	U3RXPPS	7:0				PORT[1:0]			PIN[2:0]	
0x0277	U3CTSPPS	7:0				PORT[1:0]			PIN[2:0]	

21.8.1 xxxPPS

Name: xxxPPS

Peripheral Input Selection Register

Bit	7	6	5	4	3	2	1	0
			PORT[2:0]			PIN[2:0]		
Access			R/W	R/W	R/W	R/W	R/W	R/W
Reset			m	m	m	m	m	m

Bits 5:3 – PORT[2:0] Peripheral Input PORT Selection[1]
See the PPS Input Selection Table for the list of available Ports and default pin locations.
Reset States: POR = mmm
All other Resets = uuu

Value	Description
010	PORTC
001	PORTB
000	PORTA

Bits 2:0 – PIN[2:0] Peripheral Input PORT Pin Selection[2]
Reset States: POR = mmm
All other Resets = uuu

Value	Description
111	Peripheral input is from PORTx Pin 7 (Rx7)
110	Peripheral input is from PORTx Pin 6 (Rx6)
101	Peripheral input is from PORTx Pin 5 (Rx5)
100	Peripheral input is from PORTx Pin 4 (Rx4)
011	Peripheral input is from PORTx Pin 3 (Rx3)
010	Peripheral input is from PORTx Pin 2 (Rx2)
001	Peripheral input is from PORTx Pin 1 (Rx1)
000	Peripheral input is from PORTx Pin 0 (Rx0)

Notes:
1. The Reset value 'm' is determined by device default locations for that input.
2. Refer to the **"Pin Allocation Table"** for details about available pins per port.

So I2C1SDAPPS = 0b010000 I2C1SCLPPS = 0b010001

```
/*Configure pins*/
//Set pins
TRISCbits.TRISC0 = 0;
TRISCbits.TRISC1 = 0;
//Set pins to open drain
ODCONCbits.ODCC0 = 1;
ODCONCbits.ODCC1 = 1;

//Set pins to SDA and SCL output
RC0PPS = 0x22;  //Set pin C0 to SDA
RC1PPS = 0x21;  //Set pin C1 to SCL

//Set pins to SDA and SCL input
I2C1SDAPPS = 0b010000;  //Set C0 to SDA
I2C1SCLPPS = 0b010001; //Set C1 to SCL
```

This did not fix the issue, but I definitely did need that

There is definitely a stop condition occurring on the bus.

Important: The pin locations for SDA and SCL are remappable through the Peripheral Pin Select (PPS) registers. If new pin locations for SDA and SCL are desired, user software must configure the INLVLx, SLRCONx, ODCONx, and TRISx registers for each new pin location. The Rxyi2C registers cannot be used since they are dedicated to the default pin locations. Additionally, the internal pull-ups for non-I²C pins are not strong enough to drive the pins; therefore, external pull-up resistors must be used.

I thought I was using the default pins, however I was reading for the 14pin model not the 20pin. Maybe I will just change to the default for now to just see if I can get something

SCL is RB6
SDA is RB4

I commented out all the pin configuration and connected to the default pins. This again did not give me any success.

SLRCONx slew rate control
INLVLx input level control

Slew rate is the steepness of the rising and falling edges. I want as sharp as possible leaving to cleared should be fine.

The voltage on the lines is holding around 2.8V according to the scope.

2.8/3.3 = 0.8485 which is above the 70% threshold of the I2C module

Maybe I need to set the CNT register? That would make no sense but for the start condition being detected but why not.

```
I2C1CON0bits.MODE = 0b000; //Set to client 7bit w/o masking
//I2C1PIEbits.SCIE = 1;       //Enable interrupt when start condition detected

I2C1CNTbits.CNT = 0x0003;

I2C1ADR0 = 0b0010001;   //Set 7bit address

I2C1CON0bits.EN = 1;    //Enable I2c
```


36.5.11 I2CxCNT

Name: I2CxCNT
Address: 0x028C

I2C Byte Count Register[1,2]

Bit	15	14	13	12	11	10	9	8
	CNT[15:8]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	CNT[7:0]							
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 15:0 – CNT[15:0] Byte Count

Condition	Description
If receiving data:	Count value decremented on 8th falling SCL edge when a new byte is loaded into I2CxRXB
If transmitting data:	Count value is decremented on the 9th falling SCL edge when a new byte is moved from I2CxTXB

Notes:

1. It is recommended to write this register only when the module is Idle (MMA = 0 or SMA = 0), or when the module is clock stretching (CSTR = 1 or MDR = 1).
2. CNTIF is set on the 9th falling SCL edge when I2CxCNT = 0.

Why is the register not being detected…

That is weird

I checked the datasheet and I am using the correct name.

Maybe I wont worry about that cause I think that it should be detecting the start condition regardless

```
void __interrupt(irq(I2C1)) ISR(void){
    if(I2C1PIEbits.SCIE && I2C1PIRbits.SCIF){
        //i2cStart();
    }
    return;
}
```

Even though this is not recognizing the "I2C1" name, it builds, and it will not build if that changes to something else is doesn't recognize like "I21"

The definition of a start condition is pulling the data line low while the clock is still high. I can do that by just unplugging the data line. But there is no start condition generated when this happens. Like the board is not detecting a start condition.

30/06/23

Friday, June 30, 2023 8:07 AM

Start 8:00AM

Looking online there is an I2C.h library for the PIC microchips available that may be something to try if I can't make any progress

Every resource online is basically just using the code configurator tool

Five registers are available to configure and control digital I/O pins

1. TRISx: Sets the direction as either input or output.
2. ANSELx: Determines if an analog capable pin works as an analog input or digital I/O.
3. LATx: Used to output values for a digital pin.
4. PORTx: Reads the input value of a digital pin.
5. WPUx: Enables the internal weak pull-up.

I can use this to ensure the pins can be read properly

Next week
- email Camera guy for quote/delivery
- keep messing with I2C
- figure out interrupt timers on PIC

I got into the lab and went to start messing with the PIC and now the programmer doesn't even work.

I found that when I disconnect the PIC from the circuit and just power it off my PC, I can program it and it works.

Additionally I see when the PIC is connected to the circuit the other slave device does not power on. When it is disconnected it does power on...

Clearly there is something not connected right in the circuit that is causing issues.

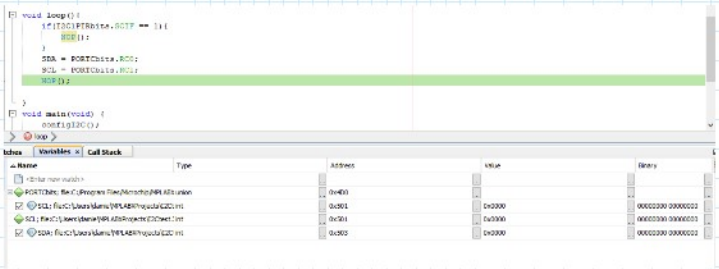
I realized I made a mistake when reconnecting the device and used the wrong header on the bottom of the board. I fixed that issue so it no longer prevents the other device from turning on, however it still will not program when connected to the circuit.

It will program fine when connected to the circuit AND my PC through USB.

Maybe there is not enough power on the bus? Or one of the power/ground jumpers is broken.

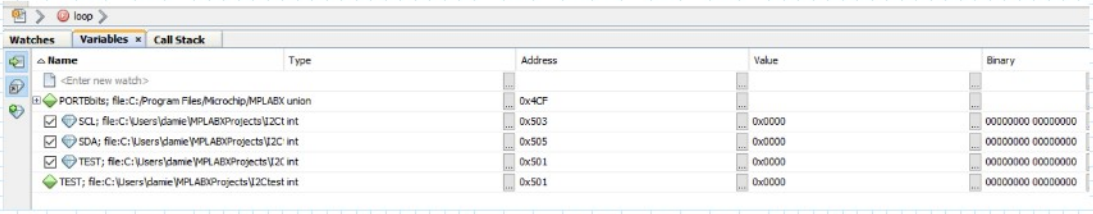
The arduino sheild board has 3.15V across the 3.3V and GND pins going to the PIC. This means that the PIC has to be getting at least 3.15V on its VDD and VSS pins.

I guess I can just connect the shield to my PC for additional power

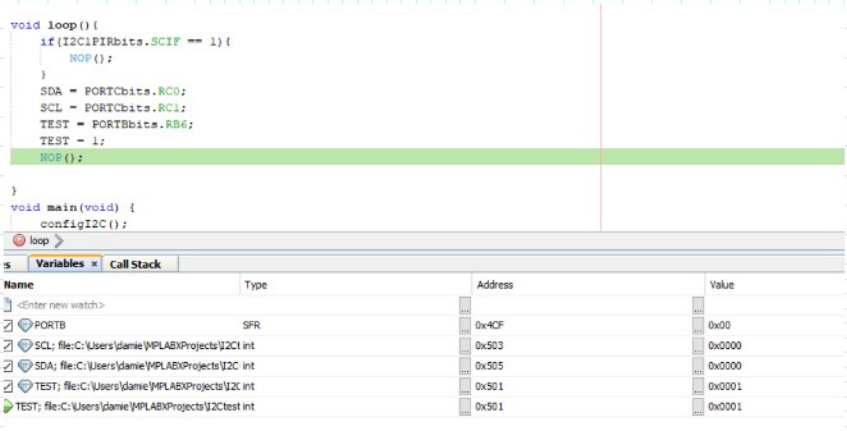


The SDA and SCL lines are reading low even though the pins are pulled high.

Let me connect to another pin that is not configured as an I2C pin to see if that causes an issue



My test pin is also reading low even though it is at 3.3V, meaning the previous test tells me nothing.



The debugger can accurately read the variable when ever I set it manually, which means it isnt being set. Maybe I need a different command to read it

Important:

- Writes to PORTx are actually written to the corresponding LATx register. Reads from PORTx register return actual I/O pin values.
- The PORT bit associated with the $\overline{MCLR}$ pin is read-only and will read '1' when the $\overline{MCLR}$ function is enabled (LVP = 1 or (LVP = 0 and MCLRE = 1))
- Refer to the **"Pin Allocation Table"** for details about $\overline{MCLR}$ pin and pin availability per port
- Unimplemented bits will read back as '0'

TRISB	SFR	0x4C7	0xF0	11110000

Pin 7 is properly set to an input (bit 7 = 1)

I found a forum post where someone said you need to clear all the ANSEL bits to set everything to digital not analog.

```
15      //SDA1 == RC0
16
17      ANSELA = 0;
18      ANSELB = 0;
19      ANSELC = 0;
20
```

Very stupid that this isnt the default but now I can actually read pins.

Name	Type	Address	Value	Binary
<Enter new watch>				
PORTB	SFR	0x4CF	0x40	01000000
PORTBbits; file:C:/Program Files/Microchip/MPLABX	union	0x4CF		
SCL; file:C:\Users\damie\MPLABXProjects\I2Ct	int	0x503	0x0001	00000000 00000001
SDA; file:C:\Users\damie\MPLABXProjects\I2C	int	0x505	0x0001	00000000 00000001
TEST; file:C:\Users\damie\MPLABXProjects\I2C	int	0x501	0x0001	00000000 00000001
TEST; file:C:\Users\damie\MPLABXProjects\I2Ctest	int	0x501	0x0001	00000000 00000001
TRISB	SFR	0x4C7	0xF0	11110000

```
65
66  void loop(){
67      if(I2C1PIRbits.SCIF == 1){
68          NOP();
69      }
70  //      SDA = PORTCbits.RC0;
71  //      SCL = PORTCbits.RC1;
72  //      TEST = PORTBbits.RB6;
73  //      //STEST = 1;
74  //      NOP();
75
```

I FINALLY GOT IT TO DETECT THE START CONDITION!!!!!!!!!!!!!!!!!!
SOMETHING!!!!!!!!!!

The ISR isnt triggering, but at least I know that the hardware is detecting the start condition.

It is also setting the stop condition flag which makes sense since the stop condition would have occurred as well. It is interesting that both flags stay set at the same time.

I want to see what is happening right away, if the address read and compared? Or do I do that manually. Additionally, is the bit being set to say that the last address had a R/W bit

I2CS1STAT0 bit 6 is true if client mode is activated -- meaning address was compared and matched, bit 4 indicates read/~write

```
54
55  void i2cStart(){
56      NOP();
57
58      while(I2C1STAT1bits.RXBF == 0); //Wait for buffer to fill
59      I2C1CON1bits.ACKDT = 1;     //ACK
60
61  };
62
63  void __interrupt(irq(I2C1)) ISR(void){
64      if(I2C1PIEbits.SCIE && I2C1PIRbits.SCIF){
65          i2cStart();
66      }
67      return;
68  }
69
70  void loop(){
```

i2cStart

Watches | Variables | Call Stack

Name	Type	Address	Value	Binary
<Enter new watch>				
I2C1PIR	SFR	0x29A	0x05	00000101
I2C1STAT0	SFR	0x298	0x00	00000000

Nothing is currently being automatically set in the STAT0 register. Maybe I can enable it to automatically do stuff with another register.

I2C1CON2 bit 4 - Address Buffer disable
    When set to 0, address is automatically loaded into the I2C1ADB0

Maybe I set this and then in my code compare the two addresses -- or I can just view the address received and make sure it is reading everything right.

I enabled the autoloading and nothing was loaded into the register. I also monitored the RXB register and nothing was loaded. I am wondering if I am haulting the software too quickly.

I am going to delay for a ms before haulting

configI2C

Output | Watches | Variables | Call Stack

Name	Type	Address	Value	Binary
<Enter new watch>				
I2C1ADB0	SFR	0x28E	0x00	00000000
I2C1ADB0	SFR	0x28E	0x00	00000000
I2C1ADB1	SFR	0x28F	0x00	00000000
I2C1ADR0	SFR	0x290	0x11	00010001
I2C1ADR1	SFR	0x291	0x10	00010000
I2C1ADR2	SFR	0x292	0x11	00010001
I2C1ADR3	SFR	0x293	0x10	00010000
I2C1PIR	SFR	0x29A	0x05	00000101
I2C1RXB	SFR	0x28A	0x00	00000000
I2C1RXB	SFR	0x28A	0x00	00000000
I2C1STAT0	SFR	0x298	0x00	00000000
I2C1TXB	SFR	0x28B	0x00	00000000

None of the I2C buffers are being loaded with anything.

I was thinking that the PIR stop bit should not be triggering if I have no delay before haulting. Because if that were the case the I2C bus would have already ended before the PIC even had time to do a single instruction which does not make sense.

I paused the debugger before giving any start condition and found that the stop flag was already set. I thought maybe this was an error from a previous run so in the setup I cleared the Stop bit flag.

Still the stop bit is being set without a start ever occuring.

That could explain why nothing is being loaded into registers, cause the PIC is saying the stop condition has already occurred.

A stop condition is defined as releasing the SDA line high while the SCL line is high.

If I use the scope to detect a rising edge, I will know if a stop condition occurs when the Single measurement triggers.

The scope never detects a change however the stop condition is still being set. Meaning there is definitely a software issue causing this cause nothing is happening on the bus.

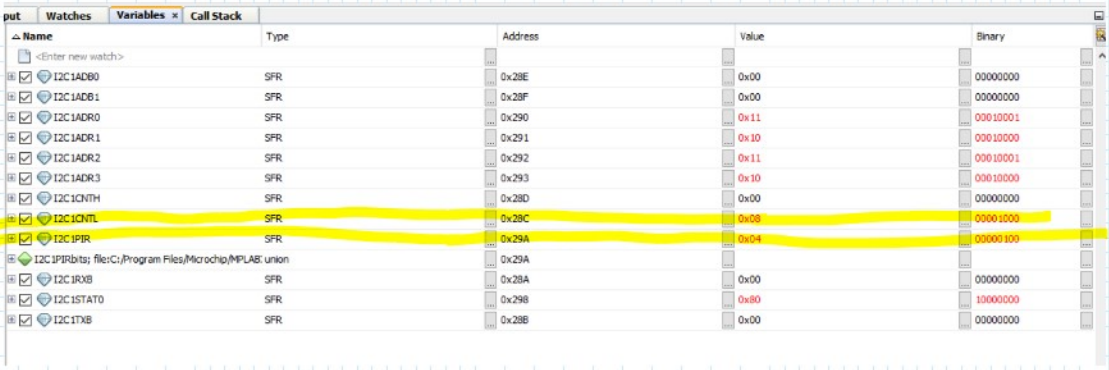
I read in the datasheet a Stop condition can also be generated when the CNT reaches 0. Still no idea why the CNT exists as it has never be mentioned in any other I2C information source.

I tried to set it the usual way with I2C1CNT but the software wont compiler... It is a read/write register.

I guess I can try the address 0x028C

```
I2C1CNTH = 0;
I2C1CNTL = 0x08;
```

You have to set it as a high and low register



Even with the CNT register > 0, the stop condition is still being set.

The stop bit can also be set if I2C1CON1 P bit is set. This is supposed to be for when the device is a host, but I will clear it anyway to be sure.

The client is not supposed to be able to make a stop condition. Everything in the datasheet is saying the client only sets the stop condition flag when a stop condition occurs on the bus. But this is not happening.

I gave up on the I2C for today.

I went and did complete documentation for all the libraries. I created a readme.md for each library and also made a simple example script

I updated the GITHUB as well.