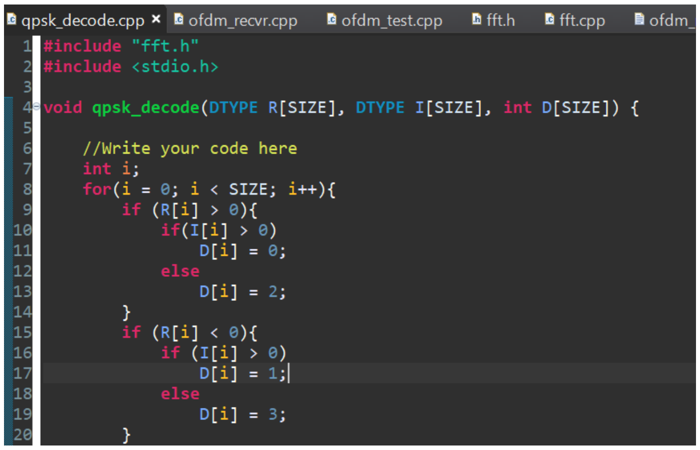
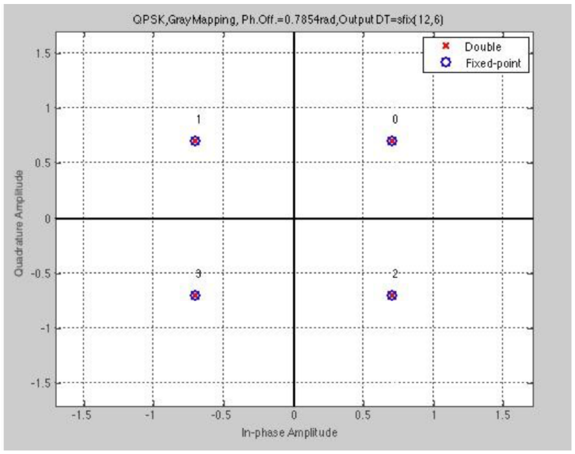
Report OFDM part 2 - Nate Mosk

Revisiting the OFDM now that I got my Csim. The OFDM receiver takes the output from the FFT and puts it into a QPSK demodulator. I used very simple code to find the quadrant that the FFT data is sitting in and then assigned a symbol.



```
#include "fft.h"
#include <stdio.h>

void qpsk_decode(DTYPE R[SIZE], DTYPE I[SIZE], int D[SIZE]) {

    //Write your code here
    int i;
    for(i = 0; i < SIZE; i++){
        if (R[i] > 0){
            if(I[i] > 0)
                D[i] = 0;
            else
                D[i] = 2;
        }
        if (R[i] < 0){
            if (I[i] > 0)
                D[i] = 1;
            else
                D[i] = 3;
        }
```

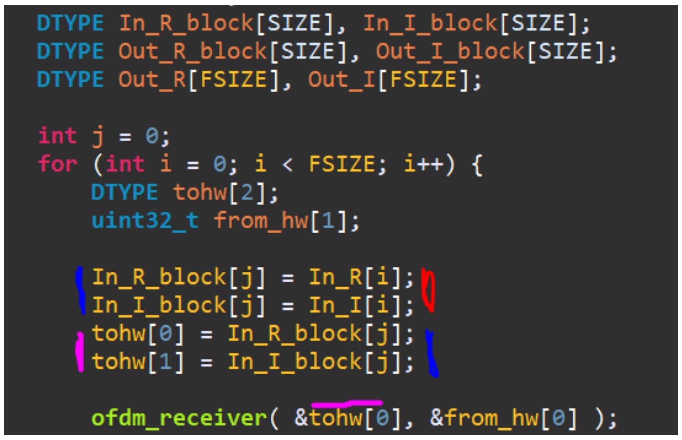


I transferred the OFDM receiver to the pynq board using vivado and jupyter.

I got stuck at the very end. While I'm able to get output from the receiver it doesn't match the golden output. I believe my hardware design and bitstream are good.

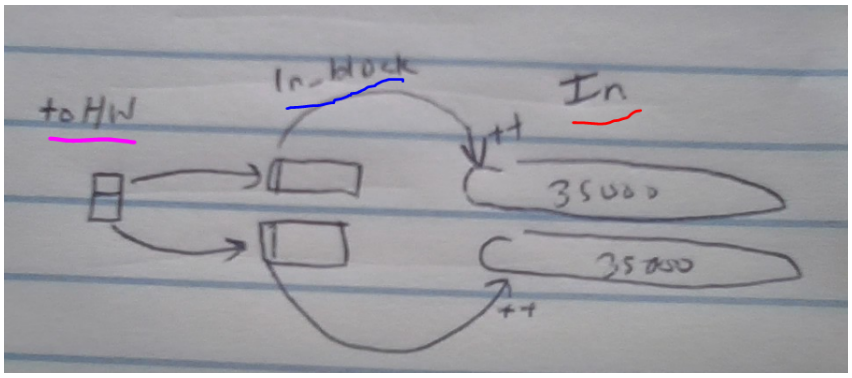
The problem is likely my python code that inputs the input.dat values into the hardware on the pynq board.

I looked very closely at the source code:

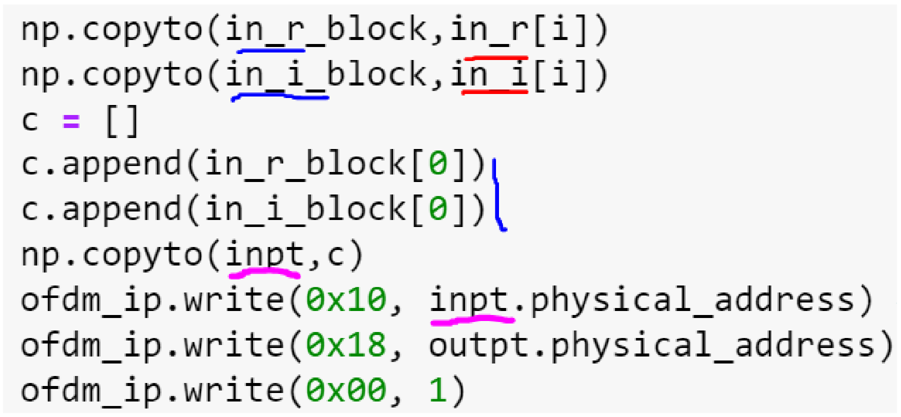


```
DTYPE In_R_block[SIZE], In_I_block[SIZE];
DTYPE Out_R_block[SIZE], Out_I_block[SIZE];
DTYPE Out_R[FSIZE], Out_I[FSIZE];

int j = 0;
for (int i = 0; i < FSIZE; i++) {
    DTYPE tohw[2];
    uint32_t from_hw[1];

    In_R_block[j] = In_R[i];
    In_I_block[j] = In_I[i];
    tohw[0] = In_R_block[j];
    tohw[1] = In_I_block[j];

    ofdm_receiver( &tohw[0], &from_hw[0] );
```

Clearly, we input a two dimensional array (pink) with our pair of values to the program, and the function has pointers to those values. It appears that we need toHW to point to a block the size of 1024, our FFT size. The first element of the In_blocks (blue) needs to point to or be equal to our input values (red). (See highly technical diagram below).

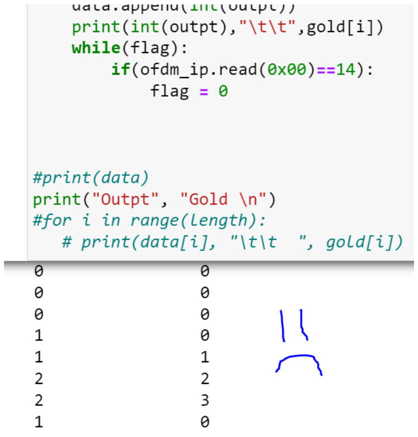


So I tried to copy this set up:



```
np.copyto(in_r_block,in_r[i])
np.copyto(in_i_block,in_i[i])
c = []
c.append(in_r_block[0])
c.append(in_i_block[0])
np.copyto(inpt,c)
ofdm_ip.write(0x10, inpt.physical_address)
ofdm_ip.write(0x18, outpt.physical_address)
ofdm_ip.write(0x00, 1)
```

But I couldn't get the golden to match. I know I'm extremely close, I just need to find the correct way to arrange the python arrays before writing to my pynq board.

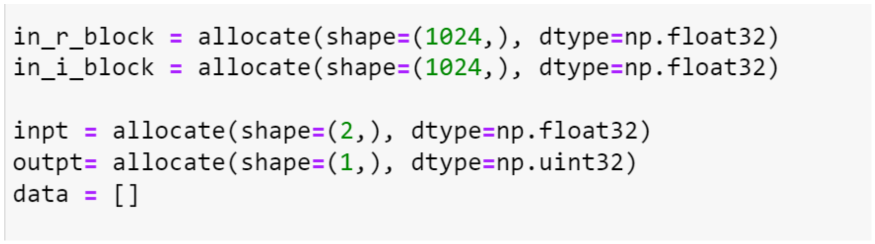


```
    print(int(outpt),"\t\t",gold[i])
    while(flag):
        if(ofdm_ip.read(0x00)==14):
            flag = 0


#print(data)
print("Outpt", "Gold \n")
#for i in range(length):
   # print(data[i], "\t\t  ", gold[i])
0                   0
0                   0
0                   0
1                   0
1                   1
2                   2
2                   3
1                   0
```

At least I'm getting output, and that's quite an accomplishment itself.

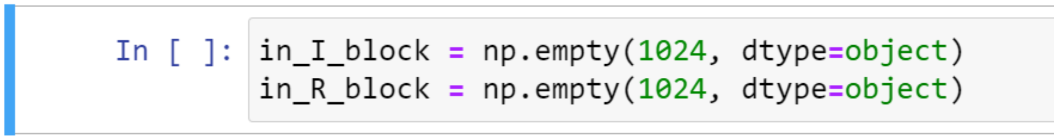
If I had more time to troubleshoot, the next thing I'd try messing with the input objects. Maybe if I use and empty object instead of "allocate" , the pointer scheme will work??

This is the current set up:



```
in_r_block = allocate(shape=(1024,), dtype=np.float32)
in_i_block = allocate(shape=(1024,), dtype=np.float32)

inpt = allocate(shape=(2,), dtype=np.float32)
outpt= allocate(shape=(1,), dtype=np.uint32)
data = []
```

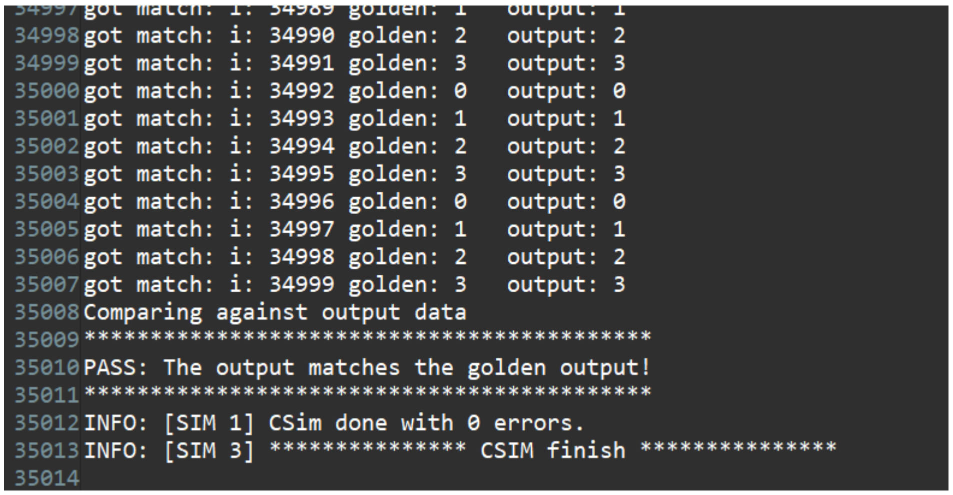
Perhaps I could try using np.empty like below and pointing to the first element of in_I_block[0].



```
In [ ]: in_I_block = np.empty(1024, dtype=object)
        in_R_block = np.empty(1024, dtype=object)
```

All in all this was a good project and I learned a lot. I'd love to see someone's successful build.

I will still feel successful about getting the golden output in vitis and even getting integer output from the fpga on the board is pretty cool.



```
got match: i: 34989 golden: 1   output: 1
got match: i: 34990 golden: 2   output: 2
got match: i: 34991 golden: 3   output: 3
got match: i: 34992 golden: 0   output: 0
got match: i: 34993 golden: 1   output: 1
got match: i: 34994 golden: 2   output: 2
got match: i: 34995 golden: 3   output: 3
got match: i: 34996 golden: 0   output: 0
got match: i: 34997 golden: 1   output: 1
got match: i: 34998 golden: 2   output: 2
got match: i: 34999 golden: 3   output: 3
Comparing against output data
*******************************************
PASS: The output matches the golden output!
*******************************************
INFO: [SIM 1] CSim done with 0 errors.
INFO: [SIM 3] *************** CSIM finish ***************
```