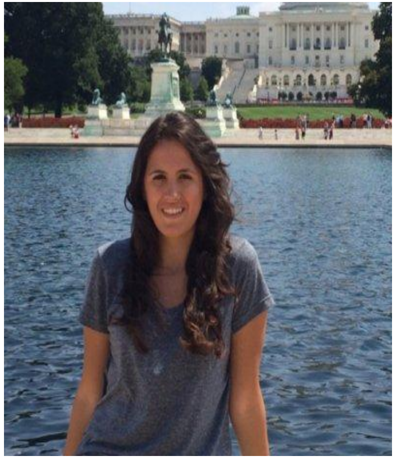
SDR in ORBIT:

Spectrum Sensing

WINLAB Summer Internship 2015

Team



Christina Baaklini
Electrical and Computer Engineering
Rutgers University

Michael Collins
Electrical and Computer Engineering
Rutgers University

Nick Cooper
High School Student
Montgomery High School

Nicole DiLeo
Electrical and Computer Engineering
Rutgers University

SDR in ORBIT

Three groups currently working with software-defined radio:

Indoor Localization

Spectrum Sensing

LTE-Unlicensed

Software-defined radio - radio communication system in which components that would usually be implemented using hardware are instead implemented using software

SDR in ORBIT

- The ORBIT testbed has an array of Universal Software Radio Peripherals for use in software-defined radio applications.
- USRPs can transmit or receive signals ranging from DC to 6 GHz.
- Controlled by software applications such as GNU Radio

Spectrum Sensing

Project Goals

- Using ORBIT, configure radio receiver(s) to collect IQ time samples
- Process the samples to obtain frequency-domain data
- Analyze frequency data to identify any unknown signals
- Repeat with modified receiver carrier frequency, sampling rate, etc. to scan the available frequency spectrum for signals
- Implement methods above in real-time

Spectrum Sensing

Project Divisions

- **CPU Implementation** - Mike Collins, Nicole DiLeo
    - Design of signal processing algorithms, data visualization tools
    - Implementation in MATLAB and C++
- **FPGA Implementation** - Christina Baaklini, Nick Cooper
    - Performance improvement on high-speed processor
    - Implementation in VHDL

CPU Division

Current Progress

- Basic Research in Digital Signal Processing
- Familiarization with ORBIT Framework
- Familiarization with MATLAB Signal Processing Tools
- Design of ORBIT Grid Experiments
- Development of MATLAB Spectrogram Script
- Beginning of C++ Implementation

CPU Division

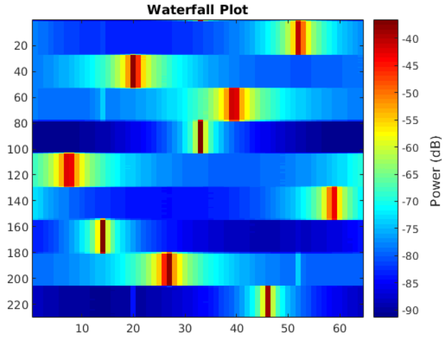
Preliminary Research and Training

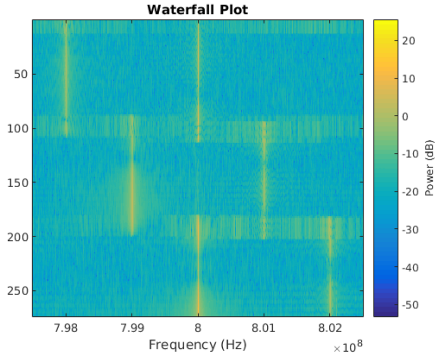
- Learned basic Digital Signal Processing concepts such as:
  - ***Sampling*** - measurement of analog signal at discrete time intervals
  - ***Quantization*** - conversion of a continuous range of values into discrete values using a certain number of bits
  - ***Nyquist Frequency*** - twice the highest frequency of the continuous-time signal
- Learned to use the ORBIT Measurement Framework and Wiserd application to run tests and take measurements on ORBIT
- Learned MATLAB signal processing tools such as:
  - **fft/ifft** - Fast Fourier Transform and Inverse
  - **fft_shift** - Adjusts zero-frequency component in FFTs

CPU Division

ORBIT Grid Experiments

- Used the ORBIT Experiment Description Language and the Wiserd application to run experiments on the ORBIT testbed
- Started with one transmitter and one receiver and collected preprocessed frequency-domain data
- Expanded experiments to multiple transmitters/receivers and extracted raw IQ time samples





CPU Division

MATLAB Spectrogram Script

- Converts time-domain signal into frequency-domain
- Applies moving average filter to reduce noise
- Calculates power magnitudes at given frequencies
- Generates a waterfall plot
- Plots individual FFTs and applies a simple peak-finding algorithm

CPU Division

C++

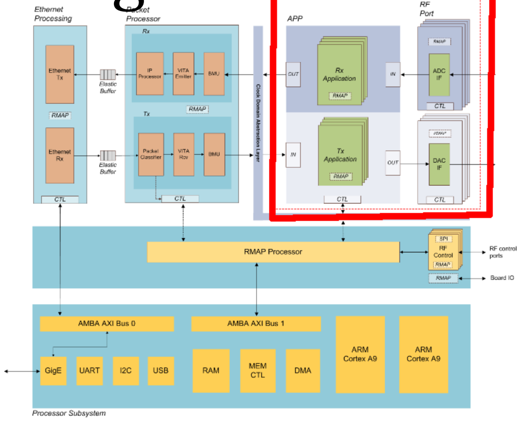
```
function [ffts,moving_avg,peaks]=spectro(m,c_fr,s_fr,k,o,w,avg)
% m = row matrix of IQ samples
% c_fr = carrier frequency
% s_fr = sampling frequency
% k = size of FFTs
% o = overlap between FFTs (between 0 and 1)
% w = row matrix of size k to be used as a window function
% avg = number of ffts to be averaged together
o = 1-o; N = numel(m);
start = @(j) k*o*j+1; % beginning of each FFT
stop = @(j) start(j)+k-1; % end of each FFT
ffts = [];

i = 0;
fprintf('Generating FFTs ... ');
while stop(i) < N
        s = m(start(i):stop(i));
        s2 = w.*s;
        s2f = fft(s2,k);
        s2f_shift = fftshift(s2f);
        ffts = [ffts;s2f_shift];
        i = i+1;
end
fprintf('Done\n');
```

```
void fft_avg::spectro() {
        overlap_ = 1-overlap_;
        unsigned int N = iq_samples_.size();
        int index = 0;

        vector<complex<float> > s;
        vector<complex<float> > s2;
        s.resize(fft_size_);
        s2.resize(fft_size_);
        empty_vector_.resize(fft_size_, 0);

        out_ = (fftwf_complex*) &(empty_vector_.front());
        plan_ = fftwf_plan_dft_1d(fft_size_, in_, out_, FFTW_FORWARD,
        FFTW_ESTIMATE);

        while (stop(index, fft_size_ , overlap_) < N) {
                for (unsigned int i = start(index, fft_size_ , overlap_);
                i <= stop(index, fft_size_, overlap_);  i++) {
                        s.push_back(iq_samples_[i]);
                        s2.push_back((window_[i])*(s[i]));
                }
                in_ = (fftwf_complex*) &(s2.front());
                fftwf_execute(plan_);
                fft_data_.push_back(empty_vector_);
                index++; }
}
```

VHDL/FPGA Division

Goals

- Implement spectrum sensing programs onto a Xilinx FPGA (ZedBoard)
- Use this to identify available frequencies using an FMC receiver.



VHDL/FPGA Division

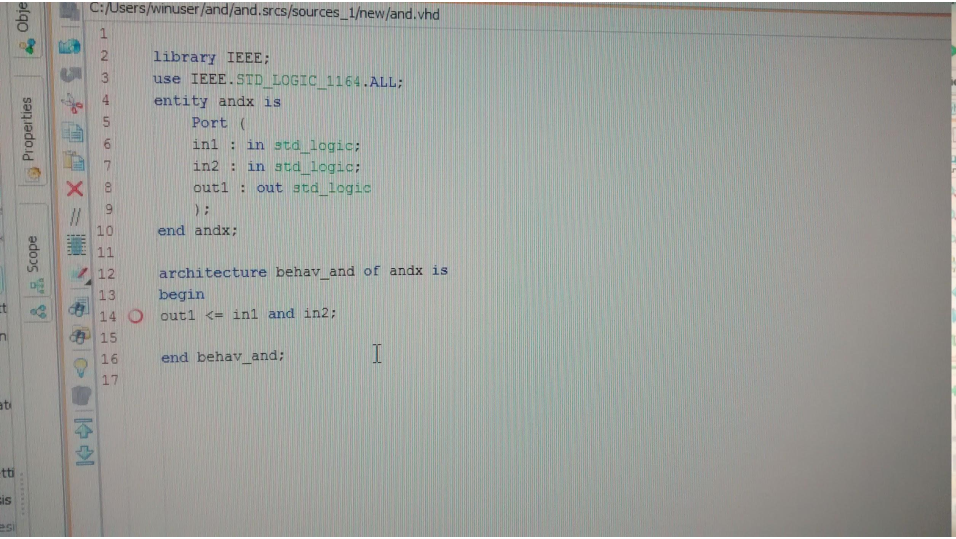
Progress

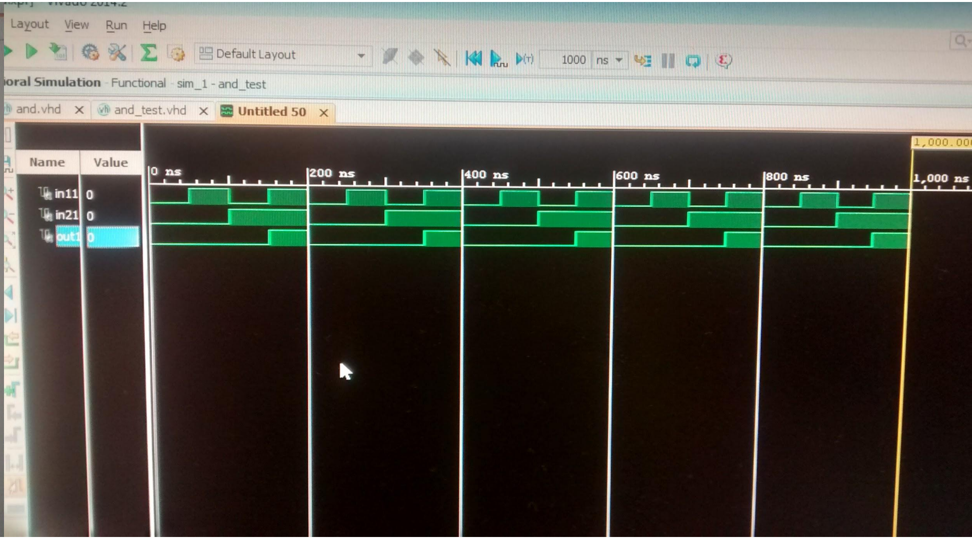
- Over the course of the last two weeks we have been learning the basics of vhdl coding and implementation onto an FPGA
- Programs created and sent to ZedBoard:
  - Simple Combinatorial Logic
  - Binary Counter (Sequential Logic)
  - Basic 4-1 multiplexer

VHDL/FPGA Division

Examples (Code and Sim Waveforms)

Simple Combinatorial Logic (and gate)

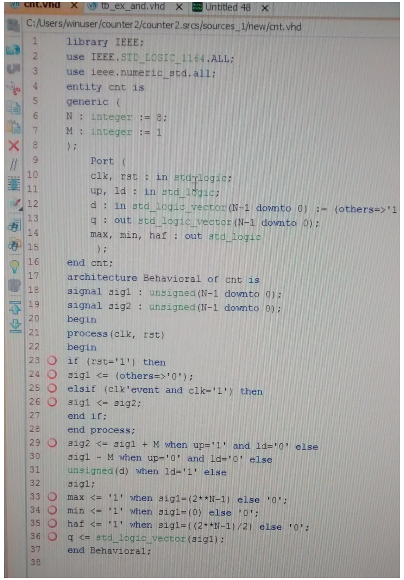


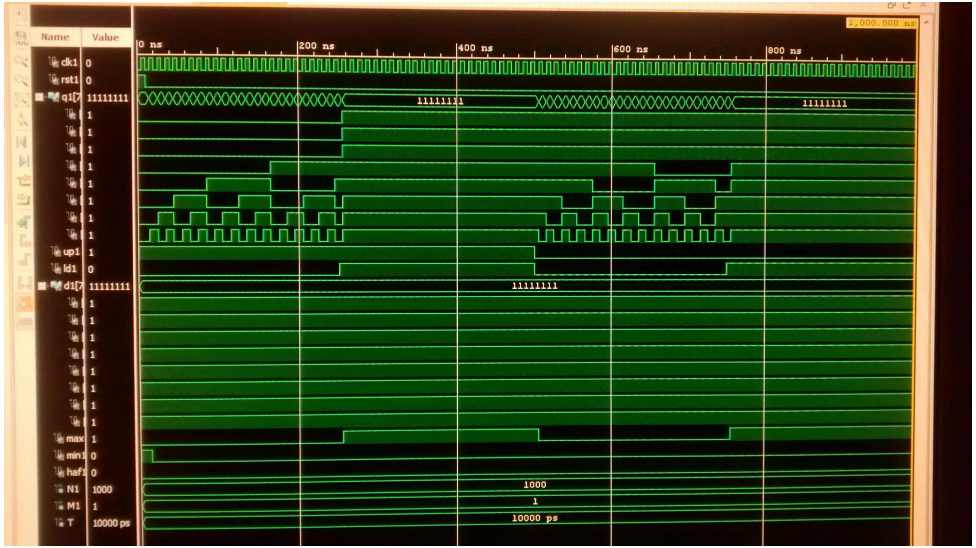


VHDL/FPGA Division

Examples (Cont.)

Sequential Circuit(Binary Counter)

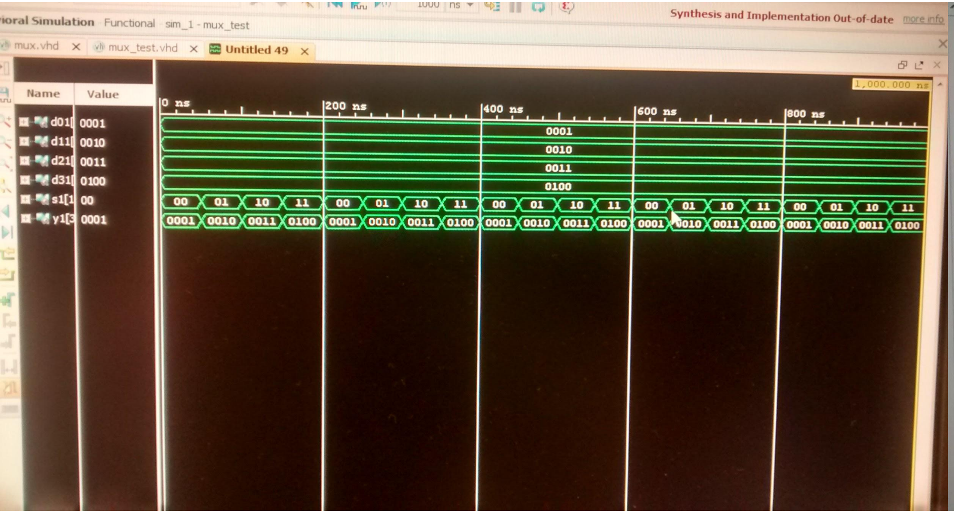


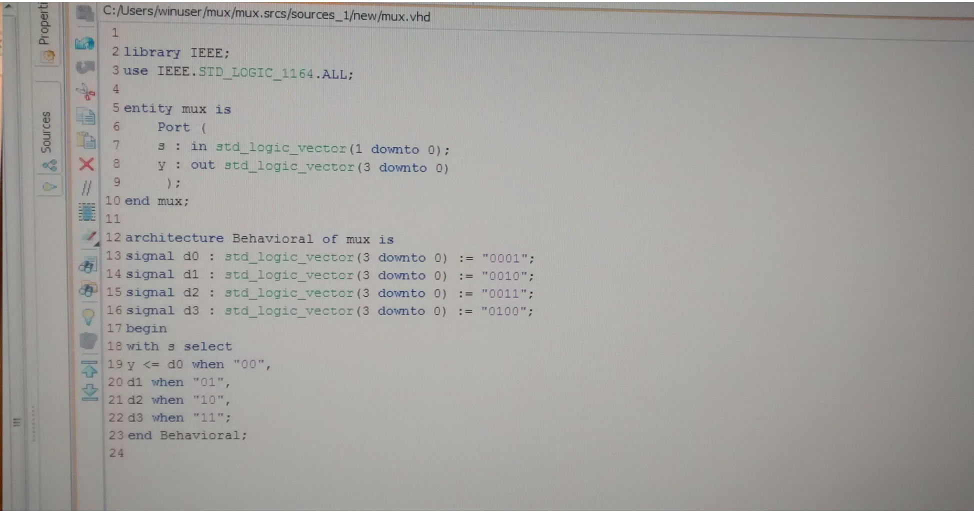


VHDL/FPGA Division

Examples (Cont.)

4-1 Multiplexer





Moving Forward

CPU Division

- Continue C++ implementation
- Start running tests with real-time analysis
- Create a user interface for scanning the frequency spectrum

VHDL/FPGA Division

- Apply VHDL to enhance SDR features on ZedBoard
- Eventually run tests with receiver attached to board