Abstract:
Spectrum analyzer is a device that receives a radio frequency (RF) signal and displays frequency components of the signal to the user. We developed a low-cost SDR-based spectrum analyzer, called DAEDALUS, with an intuitive touch interface for a frequency range from 70 MHz up to 6 GHz. We leveraged the processing power of open-source tools, such as Linux and GNU Radio (GR), as a means of keeping the cost down. Linux allowed us to integrate the ZedBoard with the AD9364 RF transceiver. This project gave us the opportunity to become knowledgeable about system-on-chip (SoC) development, spectrum analysis, RF communications, programmable logic (PL), and microcontroller (MCU) development tools.

1. Introduction
Currently, cost is a prohibiting factor when it comes to spectrum analyzers. Laboratory-grade spectrum analyzers often cost tens of thousands of dollars which prevents undergraduate students or recreational users from visually exploring the fields of SDR and RF communications. Traditional SDR kits are bulky, containing many individual components, and lack the ability to be used outside of a laboratory.

A common flaw of SDR systems is that they fail to meet the real-time requirements, and tend to be used in more passive applications where analysis of the data is done after the fact. To work around this issue and increase the system’s response time, critical computationally-intensive operations can be off-loaded from the processing system into programmable logic using a hardware description language (HDL). In order to reduce system size and do away with tedious integrations between a processor and an FPGA, companies have developed SoCs, which include both a processing system (PS) and PL. Two such SoCs are the Zynq and the CycloneV made by Xilinx and Altera, respectively.

2. Approach
When setting out to design the SDR-based spectrum analyzer [1], the initial focus was on determining the capabilities provided to the user and how the user would interact with the system. It was desired to have an intuitive interface, allowing the user to quickly become familiar with its operation. For the data plots that were to be displayed, we knew that we would need to keep these plots simple, as the processing power is limited on the ARM processor. The Qt framework was selected for the development of graphical user interface (GUI) showing both the power spectrum density and waterfall plots. We also selected GQRX to act as an SDR receiver that can use GR blocks to support its functionality.
While considering how our system should sample, we learned that sampling could be improved if the technique of using in-phase and quadrature (I/Q) samples was employed (Figure 1). The significance of this technique greatly affected our decision on a platform for our RF front-end.

![I/Q Sampling](image)

In the equation above, the left side can be represented in polar form and the right side in the Cartesian form. This representation captures the phase of the signal in the magnitude of the I and Q components for each time period. It allows for increased bandwidth coverage, as having two samples cuts the Nyquist-required sampling rate in half, and also allows for many different demodulation techniques.

Another important operation we needed to consider, was the discrete Fourier transform (DFT) which allows us to monitor the frequency components of the signal. The DFT is defined as:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{j2\pi kn}{N}}, \quad k = 0, 1, ..., N - 1$$

The DFT is an intensive computation that, if sequentially computed on the ARM processor of the Zynq SoC, would be too slow to achieve real-time performance. The methods to increase computational efficiency are commonly referred to as fast Fourier transforms (FFT) [2]. The common theme amongst these different algorithms is that they take the DFT, which is $O(N^2)$, and turn it into a smaller computation that is $O(N\log_2N)$. While this decrease in computational intensity is beneficial, these algorithms present additional operations that are not as straightforward as the simple DFT [3].

In order to meet sampling requirements and implement hardware acceleration, we decided to use a system that consists of the Analog Devices AD9364 RF agile transceiver interfaced with the Avnet's ZedBoard development board. The ZedBoard consists of various peripherals that allow us to achieve our system goals with the heart of the board being the Zynq SoC IC. In order to meet the development platform attribute we have chosen to run the Linux operating system on the Zynq SoC. This operating system gives us the ability to run many different applications and development environments, including SDR tools, such as GR.

The ability for our system to be reconfigurable is also a desirable attribute from a development standpoint. This allows users to develop additional features for the system. The current utilization of the PL sits at 31%, allowing room for additional DSP operations to be offloaded. GR also allows for application development on the software side, leading to an endless amount of possible configurations for our system.

3. System functional decomposition

System functional decomposition is shown in Figure 2. Due to the nature of a Level-1 design, this figure does not represent the components used to perform these tasks, but instead states the functions implemented.

The RX path (Figure 3) begins with the low noise amplifier (LNA) which provides amplification of the low power RF signals received. Next, the mixers are used to bring the monitored signal down to baseband. Baseband is commonly a narrow frequency range located around 0Hz which is referred to as the origin. Mixers perform this task by multiplying the monitored signal by the frequency of the local oscillator set by the user, which brings the desired signal down to the baseband. This task takes advantage of the convolution property which states that a multiplication in the time domain corresponds to convolution in the frequency domain.
The programming of the local oscillator starts in the PS side of the system where function calls to the libiio library are made. Libiio is the library that abstracts the hardware controls enabling easier development. The libiio library proceeds to pass the signal down to the PL side of our system. In the PL system, the values for adjustment are written to the required registers on the AD9364.

The next function is low pass filtering which is used to remove the potential aliasing. The first low pass filter is a transimpedance amplifier with a programmable 3dB corner frequency between 1MHz and 70MHz. The second filter is a 3rd order Butterworth filter with programmable corner frequency of 200KHz to 39.2MHz. All filtering rates in our system are typically selected upon initial configuration of our system.

After the signal has been conditioned, it is ready to be sampled by the ADC. The AD9364 has a 12-bit third order continuous time delta-sigma modulator ADC. When the sample rate is increased, our system can provide coverage of a larger bandwidth. However, if the sample rate becomes high enough our system begins to drop samples because of the limitations of the ARM processor. Dropping samples is not desirable as it can make catching bursty signals, like LTE, much more difficult to notice. We decided that we did not want to limit the user when it came to selecting the sample rate, so they should be cognizant of the tradeoffs. The user can adjust the sample rate at any point via our touch screen interface. These adjustments will write directly to the AD9364 via libiio functions.

Following the ADC in the RX flow path are finite impulse response (FIR) filters with built in down samplers. The process of filtering a signal and then down sampling this signal is called decimation. The final two blocks in the RX path are to provide gain for our system and another FIR filter. The gain control block can be used to increase the power of the signal before it is passed on to the rest of the system. The FIR filter is 128 tap filter, which can be programmed as desired for final signal conditioning before being passed along. Additionally, the FIR filter block provides the capability to perform decimation by a factor of 1, 2, or 4. The combination of these two blocks allows for final signal conditioning.

Once the signal has successfully made it through the RX path of AD9364, it is processed by the DFT. That allows us to take the discrete time samples we obtained from the RF front-end and convert them to the frequency domain. This conversion to the frequency domain allows our users to perform the RF spectrum analysis that they desire. Because the FFT can be done in parallel, the DFT/FFT operations were implemented on the PL.

One final DSP topic is windowing. GQRX uses GR blocks to implement various types of windows and allows the user to choose which one to use. Each of these windows have trade-offs, their own benefits and drawbacks. Some
trade-offs include the attenuation of side-lobes at the expense of limiting the main lobe. These changes affect the amount of spectral leakage present within the spectral representation of a signal [5].

4. FPGA Development
Having experience working with other embedded SDR systems, we found it important to hardware accelerate our design. Since our system focuses around the DFT and FFT algorithms, it was in our best interest to implement them on the PL of our SoC. Our PL design is very specific for the RF front-end board that we chose to use: the AD-FMCOMMS4. The AD-FMCOMMS4 hosts the AD9364 transceiver, which is controlled through a mapping of registers that are represented in the PL. The logic required to operate our RF front-end was provided to us from Analog Devices, Inc. The reference PL design is shown in Figure 4.

![Figure 4: HDL design](image)

![Figure 5: Datapath and controller design](image)

All of the blocks shown in the base design are used to allow the PS to communicate with the PL, which communicates with the RF front-end. This communication is done through direct memory access (DMA) and AXI bus communication. The PS, hosting Linux, is then able to provide the Linux kernel and Linux user with mechanisms to access data flowing within the PL. From this base design, we worked towards implementing our own digital logic, capable of taking the IQ samples directly off of the board and passing them through a DFT IP block before making those values available to the Linux system. To do this we developed a pseudocode, which next we synthetized into a datapath and controller. Figure 5 shows the datapath and controller interacting with one another and Figure 6 shows the datapath architecture containing all digital hardware structures necessary to provide desired functionality.

The data lines in the above datapath are 12 bits long, which corresponds to our ADCs bit resolution. Samples will be written into input buffers that hold N 12-bit entries (recall: N is the size of our DFT transform). The controller (Figure 7), is responsible for handling the synchronization of storing input samples, inputting them into the DFT, storing the DFT blocks outputs, and then outputting these samples, in sequence, to a packing block that is responsible for sending 64-bit vectors to the processing system.
The controller makes use of flags that are set and cleared by the datapath in order to bring the functionality specified by pseudocode. In addition to implementing the functionality shown in the preceding diagrams, we also needed specialty blocks to manipulate the data to put it into a format that we can use. In the PL, the IQ samples are represented as 16-bit quantities - 16 bits for I and 16 bits for Q. We needed to modify these 16-bit quantities into their original 12-bit format. This process is shown in Figure 8.

5. GUI development
The GUI for our project was built on GQRX open source SDR application. As GQRX utilizes GRs gr-osmosdr source to retrieve samples, which then get passed into the flow graph (Figure 9). Before it makes its way into the application specific software receiver, GQRX calls GRs FFT module to convert time domain data into the frequency domain. Then, utilizing the QT framework, it displays both the power spectral density and its corresponding waterfall plot.

In its non-accelerated form, GQRX uses GRs FFT block to compute the power spectrum and then plots it multiple times per second. The system itself was edited to add a DFT core which utilizes the programmable logic of the Zedboard, further enhancing the performance of the system. To accommodate this change within GQRX, a bypass around the software version of the DFT block was added and inside of the gr-osmosdr module a DFTconverter block was developed to convert the data from its original form into one that can be plotted.
The base GQRX application was designed for a standard PC utilizing a keyboard and mouse, but we wanted to provide a touch interface. We designed a slider to allow the user to easily navigate the frequencies inside the spectrum. This slider is mapped to the appropriate frequency range of our device (70MHz to 6GHz). Another feature added was a drop-down menu specific for the AD-FMCOMMS4 where different sampling rates specific to the board could easily be selected.

6. Experimentation

System development was associated with multiple tests validating individual components and their integration. Our first experiment was to ensure that we had functioning GR installed on our personal computers. In order to test this functionality, we used the RTL-Dongle, which is a low-cost SDR receiver, to see if we could sample a radio signal and successfully plot the DFT of the signal. The DFT output was verified by implementing the FM demodulator [7], and this ensured that we had installed the GR correctly (Figure 10).

As we tested the AD9364 and the Zynq Soc, we knew that the ARM processor was likely to be the limiting factor in our system and this test allowed us to see how limited it was. We obtained baseline data using both the RTL-Dongle and the HackRF. The sample rate limitation of these devices were 2Msps for the RTL-Dongle and 20Msps for the HackRF [8]. Since the testing of these devices were performed on personal computers there was no worry about processing power being the limiting factor. The results from this test can be seen in Table 2. Sample rates achieved by both the HackRF and RTL-Dongle approximately match the limitation stated, proving that our GR block works. We can outperform the RTL-Dongle but we were unable to reach the same sample rate as the HackRF because the HackRF was having its samples processed by a personal computer while our system was processing all of these samples on the ARM processor. From these results we determined that our system can handle sample rates between 1Msps to 4.5Msps before it starts dropping samples.
We tested both the DFT IP core as well as our software to convert the packed hardware signal (Figure 11). Once the data was converted into the I/Q representation, the data was written into the test bench of our DFT IP core. Through simulation we determined that the system hardware acceleration was functioning properly. Final testing was performed using system interface for a selected radio station as shown in Figure 12.

### Table 2: Throughput test results

<table>
<thead>
<tr>
<th>Sample Rate</th>
<th>ZBAD</th>
<th>RTL-Dongle</th>
<th>HackRF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9 Mps</td>
<td>Unable</td>
<td>1.05E+06</td>
<td>965948</td>
</tr>
<tr>
<td>1 Mps</td>
<td>1.02E+06</td>
<td>9.96E+06</td>
<td>9.70E+05</td>
</tr>
<tr>
<td>1.5 Mps</td>
<td>1.52E+06</td>
<td>1.53E+06</td>
<td>1.47E+06</td>
</tr>
<tr>
<td>2 Mps</td>
<td>2.03E+06</td>
<td>2.07E+06</td>
<td>1.98E+06</td>
</tr>
<tr>
<td>2.5 Mps</td>
<td>2.55E+06</td>
<td>2.60E+06</td>
<td>2.48E+06</td>
</tr>
<tr>
<td>3 Mps</td>
<td>3.04E+06</td>
<td>2.46E+06</td>
<td>2.98E+06</td>
</tr>
<tr>
<td>4 Mps</td>
<td>4.05E+06</td>
<td>9.99E+05</td>
<td>3.98E+06</td>
</tr>
<tr>
<td>5 Mps</td>
<td>4.58E+06</td>
<td>1.10E+06</td>
<td>4.98E+06</td>
</tr>
<tr>
<td>10 Mps</td>
<td></td>
<td>1.0E+07</td>
<td></td>
</tr>
<tr>
<td>20 Mps</td>
<td></td>
<td>2.0E+07</td>
<td></td>
</tr>
<tr>
<td>21 Mps</td>
<td></td>
<td>2.1E+07</td>
<td></td>
</tr>
<tr>
<td>25 Mps</td>
<td></td>
<td>2.19E+07</td>
<td></td>
</tr>
<tr>
<td>30 Mps</td>
<td></td>
<td>1.00E+05</td>
<td></td>
</tr>
</tbody>
</table>

Figure 11: DFT IP core testing flowchart and simulation

Figure 12: Testing for 90.9 MHz radio station

References


[4] 4909 2 fig02.png (PNG Image, 26081833 pixels) - Scaled (25%).


