Thomas Forrester, Stephen Wunduke

University of Virginia School of Engineering and Applied Science

351 McCormick Rd, Charlottesville, VA 22904

National Radio Astronomy Observatory, 520 Edgemont Rd, Charlottesville, VA 22903

Field Programmable Gate Arrays:

Designing a User-Selectable Frequency Generator

Purpose and Scope

This document discusses work done for the 2020 National Radio Astronomy Observatory Undergraduate Summer Student Research Assistantship Program. Specifically, it describes using a field programming gate array (FPGA) to design and implement a user-selectable frequency generator.

How this Document is Organized

- **Introduction** - Discusses FPGAs and their role in radio astronomy.
- **Architecture Background** - Explains the purpose and goals of the project. This section is split up into two subsections. The Problem Background explains the purpose of the design. The Goals and Context section explains the main goals of the project and the topics learned throughout the summer.
- **Solution Background** - Explains in detail the design, implementation, and testing of the user-selectable frequency generator. This section is split up into four subsections. The Approaches section discusses the various approaches considered for the project design. The Design section explains the design and its operation in detail. The Implementation section explains how the design was implemented and includes calculations and code. The Testing section explains how the design was tested and includes test results and images of the final design.
- **Conclusion** – Summarizes the project.
- **Acknowledgements** - Recognizes those who made the project possible.
- **References** – Includes reference materials used throughout the project.
- **Appendix – FAQ** - Answers questions about future improvements for the project.

Introduction

An FPGA is an integrated circuit that can be reprogrammed by the user after it is manufactured [1]. FPGAs are very important pieces of technology and play a significant role in radio astronomy. They are used for many applications including, but not limited to: digital signal processing, data transportation, data reduction, and filtering. They can be found in the front-end

electronics of radio telescopes, correlators, monitoring and control systems, etc. Without FPGAs, astronomers would not be able to collect and analyze the amount of data they do today.

Architecture Background

Problem Background

In radio astronomy it is important to manipulate the frequency of incoming radio waves so electronic systems can process the data. A user-selectable frequency generator is a device that allows the user to supply a specified frequency to a circuit block inside a system-on-chip (SoC) design. This can be used in radio astronomy for applications such as up-conversion, down-conversion, and filtering.

Goals and Context

The main goal of this summer project was to learn about FPGAs, digital circuits, and how to develop a successful SoC design. At the end of the project, the knowledge and skills gained throughout the summer were used to design and implement a user-selectable frequency generator. This project was split up into four phases:

1. Digital logic circuit research

Before working with FPGAs, it is very important to build a foundational understanding of digital circuits. Without this fundamental knowledge, it is much more difficult to understand the inner workings of FPGAs. Learning these things before jumping into the world of FPGAs can save a tremendous amount of time and help engineers develop successful SoC designs. Because these concepts are so important, background research was the starting point of the summer project. During this phase, concepts such as Boolean logic, flip-flops, counters, multiplexers, static timing analysis, synchronous and asynchronous design, pipelining, etc., were researched in detail to ensure they were fully understood. Once the necessary concepts were understood, the focus shifted to learning how to code in the hardware descriptive language (HDL), Verilog.

2. Learning the hardware descriptive language, Verilog

HDLs are coding languages used to describe the structure and behavior of digital logic circuits. They are the primary languages used to code FPGAs. It is also possible to code FPGAs in programming languages such as Python; however, using languages such as this increases the abstraction layer, and the engineer does not have as much control over the design. The designer does not need to have a deep understanding of how FPGAs and digital circuits function. Therefore, this approach is not typically used. Writing code without understanding how the circuit operates is not feasible for complex large-scale SoC designs.

This phase of the project focused on translating newfound knowledge of digital electronics into Verilog. Because HDLs are used for describing the behavior of digital circuits, one can see why it is vital to understand how they work and the logic behind them. After building a foundation in both digital circuits and Verilog, the project moved into the FPGA research phase.

3. FPGA research

This phase of the project focused on learning about FPGAs and their architecture. In order to successfully implement an SoC design using an FPGA, it is important to understand the resources available within them. The topics researched in this phase covered everything related to how FPGAs work. This included configurable logic blocks (CLBs), slices, clocking resources, I/O banks, Mixed-Mode Clock Manager (MMCM), etc. This phase also covered how to choose the appropriate FPGA for the needs of a specific project.

This project used the Arty A7-35T development board produced by the Xilinx Corporation. The board features the Xilinx Artix-7 XC7A35TICSG324-1L FPGA [2]. I will be referring to this FPGA as the Artix-7 throughout this document. After learning how FPGAs function, the functionality and pin layout of the Arty A7-35T and Artix-7 was researched. Topics covered included: high range and high performance I/O banks, global and user defined clocks, resources available on the Artix-7, layout of I/O banks on the Artix-7, etc. Finally, after starting from the ground level and working up, all of the necessary concepts were understood and the project moved into the design phase.

4. Designing and implementing a user-selectable frequency generator

During this phase of the project, the user-selectable frequency generator was designed, implemented, and tested. Planning is very important when developing an SoC design. There are many different options when choosing the correct FPGA for a design. It is the designer's job to look at the project requirements and choose an FPGA that is fast enough, cost efficient, and contains the appropriate amount of resources. Because there are many ways to accomplish the same task, each design option must be explored. Before beginning the implementation portion of the user-selectable frequency generator, the project was carefully planned out. Many different design choices were considered. After choosing an appropriate one, each piece of it was coded and tested individually. Finally, each block of the design was connected and tested.

Solution Background

Approaches

Following is the design process used for this project:

- Understand the requirements
- Develop the design
- Write Verilog
- Turn textural description into gates that can be placed and routed to the part
- Upload the file to the part
- Test and resolve any issues

The following design options were considered for the user-selectable frequency generator:

- Using a single counter to output various frequencies
- Using a multiplexer to output various frequencies
- Using cascaded counters to output various frequencies

The design option chosen was to use a single counter to output various frequencies. This option allowed for efficient implementation and future modifications. The software used in the design and implementation of the project was Xilinx Vivado.

Design

The design is constructed as follows:

- MMCM generating a clock set to 100 MHz as the clock input for the counter
- 32-bit counter
- 2 dip switches to select the output frequency of the counter
- 2 LED lights connected to the dip switches to indicate the design is functioning properly

The frequency of the clock is 100 MHz so the clock period is 10 ns. The user can choose which of the 32 bits of the counter the output is dependent upon. When the chosen bit is 1, the output of the counter is high. When the chosen bit is 0, the output is low. For example, if the output is set to the 16^{th} bit of the counter, and the 16^{th} bit of the counter is 1 at the rising edge of the clock, then the output of the counter is high. If the 16^{th} bit is low at the rising edge of the clock, the output is low. The output frequency is dependent upon which bit is chosen and the position of the switches. The lower the bit chosen, the higher the output frequency. For example, if the 16^{th} bit is chosen, the output frequency is going to be higher than if the 26^{th} bit is chosen. Because the design uses two dip switches, there are four different options for switch placement: both switches off, only switch 0 on, only switch 1 on, and both switches on. Initially, when both switches are off, the counter counts by 1. When only switch 0 is on, the counter counts by 2. When only switch 1 is on, the counter counts by 4. Finally, when both switches are on, the counter counts by 8. Thus, the user can output up to eight times the original frequency. In this design, two LED lights are connected to the output of the counter in order to demonstrate that the design is working correctly. Each LED is set to a different bit of the counter output. The LEDs blink faster or slower depending on the bit chosen and position of the switches.

Implementation

Verilog was used for the implementation of this design. The code uses hierarchal methodology and consists of a 32-bit counter, an MMCM running a clock at 100 MHz, and a top file connecting the design together. The counter was implemented first and has an asynchronous active-low reset. It counts by 1, 2, 4, and 8, depending upon the position of the switches. The implementation is shown below in figure 1.

```
// Specify the time unit to be 1ns and the precision to be 1ps
`timescale 1ns / 1ps

// Instantiate a module with 3 inputs and 1 output
module counter (sw,
                rstn,
                clk,
                out);

    // Input for sw0 and sw1 on Arty board
    input [1:0] sw;

    // Input for clock
    input clk;

    // Input for reset
    input rstn;

    // Output for ld4 and ld5 on Arty board
    output [1:0] out;

    // 32 bit register for the counter
    reg [31:0] ld;

    // ld4 on the board blinks based on the value of the 26th bit of the register
    // T = 10ns, (10 * 10^(-9)) * 2^26 = 0.67 seconds * 2. ld4 will blink once wvery 1.34 seconds
    assign out[0] = ld[26];

    // ld5 on the board blinks based on the value of the 24th bit of the register
    // T = 10ns, (10 * 10^(-9)) * 2^24 = 0.17 seconds * 2. ld5 will blink once every 0.34 seconds
    assign out[1] = ld[24];


    // Define functionality
    always @(posedge clk or negedge rstn)
    begin

    // Async active-low reset
    if(!rstn)
      ld <= 32'b0000_0000_0000_0000_0000_0000_0000_0000;
      else

        // If sw0 and sw1 are off (2'b00), the original frequency is output
        // If sw0 is on (2'b01), twice the original frequency is output
        // If sw1 is on (2'b10), four times the original frequency is output
        // If sw0 and sw1 are on (2'b11), eight times the original frequency is output
        case(sw)
        2'b00 : ld <= ld + 1;
        2'b01 : ld <= ld + 2;
        2'b10 : ld <= ld + 4;
        2'b11 : ld <= ld + 8;
        endcase
    end
endmodule
```

Figure 1: Verilog Code for 32-Bit Counter

The first LED was set to the 26th bit of the counter, and the second was set to the 24th. These bits were chosen because it was easy to see the LEDs blinking in this range. The frequency was calculated for each LED as follows:

The first LED is dependent upon the 26th bit of the counter, and the clock period is 10ns.

$$(10 * 10^{-9}) * 2^{26} = 0.67\ seconds$$

$$0.67 * 2 = 1.34\ seconds$$

$$\frac{1}{1.34} = 0.75\ Hz$$

Thus, the frequency of the first LED when both switches are off is 0.75 Hz. By flipping switch 0, the counter counts by 2 rather than 1, thus the frequency is doubled and is now 1.5 Hz. Flipping switch 1 gives four times the frequency (3 Hz), and flipping both switches gives eight times the frequency (6 Hz).

The second LED is dependent upon the 24th bit of the counter, and the clock is 10ns.

$$(10 * 10^{-9}) * 2^{24} = 0.17\ seconds$$

$$0.17 * 2 = 0.34\ seconds$$

$$\frac{1}{0.34} = 2.94\ Hz$$

For LED 2 the frequencies that can be output with the selected bit are 2.94 Hz, 5.88 Hz, 11.76 Hz, and 23.52 Hz. The user can change the bit that the output is dependent upon in order to produce a wide range of frequencies. Next, the MMCM was implemented. It was constructed using the clocking wizard feature in Vivado. The clock for the design was set at 100 MHz because the Arty A7-35T features a 100 MHz oscillator [2]. Finally, everything was connected together in the top file. This file can be seen below in figure 2.

```
// Specify the time unit to be 1ns and the precision to be 1ps
`timescale 1ns / 1ps

// Instantiate a module with 3 inputs and 3 outputs
module top (sw,
            rstn,
            locked,
            clkin,
            clkout,
            out);

    // Input for sw0 and sw1 on Arty board
    input [1:0] sw;

    // Input for reset pin
    input rstn;

    // Output for locked pin
    output locked;

    // Input for clock input pin
    input clkin;

    // Output for clock output pin
    output clkout;

    // Output for ld4 and ld5 on Arty board
    output [1:0] out;

   // Instantiate a counter and specify port connections
   counter c0 (.sw(sw),
               .clk(clkin),
               .out(out),
               .rstn(rstn));

   // Instantiate an MMCM and specify port connections
   clk_wiz_0 tm (.clk_out1(clkout),     // output clk_out1
                 .reset(rstn),          // input reset
                 .locked(locked),       // output locked
                 .clk_in1(clkin));      // input clk_in1
endmodule
```

Figure 2: Verilog Code for Top Module

In the top module, all of the inputs and outputs were specified for the design. Both a counter (c0) and a clock (tm) were instantiated, and the port connections were made. Looking at figure 2 above, port connections were made using the port names and a dot (.). For example, .clk(clkin) means that clk in the counter module is connected to clkin in the top module [3]. This was done for each input and output in the counter and clock modules. After the Verilog code was written, I/O planning was done in Vivado. The pin connections and I/O planning view can be seen below in figures 3 and 4.

Name	Direction	Board Part Pin	Board Part Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco
All ports (8)									
out (2)	OUT					☑	35	LVCMOS33*	3.300
out[1]	OUT				J5	☑	35	LVCMOS33*	3.300
out[0]	OUT				H5	☑	35	LVCMOS33*	3.300
sw (2)	IN					☑	16	LVCMOS33*	3.300
sw[1]	IN				C11	☑	16	LVCMOS33*	3.300
sw[0]	IN				A8	☑	16	LVCMOS33*	3.300
Scalar ports (4)									
clkin	IN				E3	☑	35	LVCMOS33*	3.300
clkout	OUT				U8	☑	34	LVCMOS33*	3.300
locke	OUT				T8	☑	34	LVCMOS33*	3.300
rstn	IN				C2	☑	35	LVCMOS33*	3.300

Figure 3: Pin Connections for Final Design

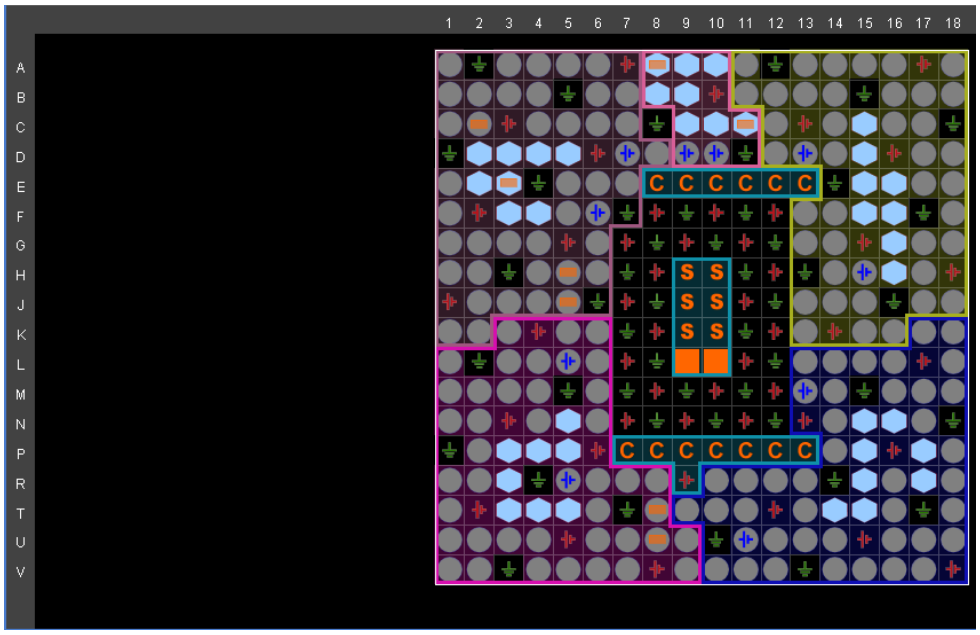

Figure 4: I/O Planning View of the Artix-7 35T in Xilinx Vivado

The pin connections were chosen based on the pin layout of the Arty A7-35T. In figure 4, the orange rectangles within the gray circles and blue hexagons represent pin connections. For example, in figure 3, out[1] is connected to pin J5. In figure 4, there is an orange rectangle in J5. After all of the connections were made, the design was uploaded onto the board and tested.

Testing

To begin testing, the functionality of the code was tested using a behavioral simulation in Vivado. To make testing more efficient, the two outputs were set to bits 2 and 4 of the counter rather than 24 and 26. Out[0] was dependent upon the 2nd bit and out[1] was dependent upon the 4th bit. Each of the switch positions were tested to ensure the counter was counting correctly and the output reflected the expected result. The simulations can be seen below in figures 5-8.

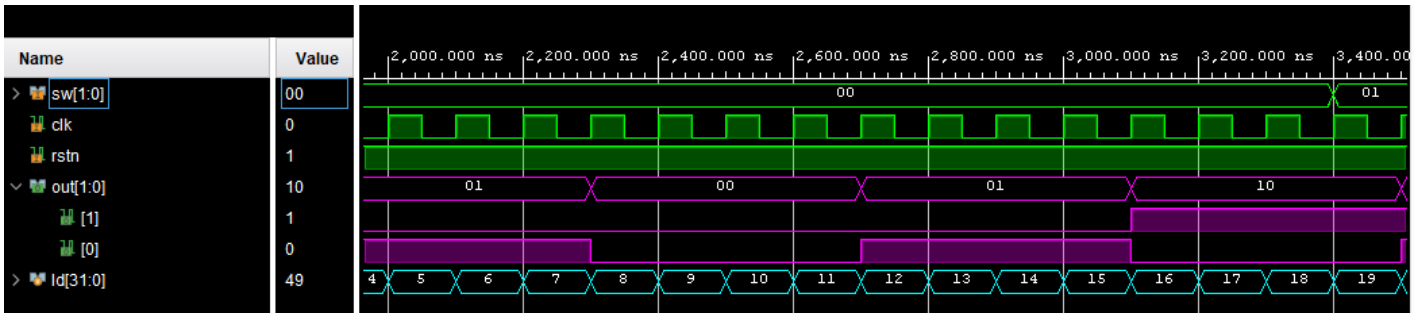

Figure 5: Counter Behavioral Simulation with Switches Off

Looking at figure 5, the design functioned as expected. When both switches were off, the counter counted up by 1 at the rising edge of the clock, and the output accurately reflected the value of the 2nd and 4th bit of the counter. For example, the number 6 in binary is 0110. Looking at figure 5, when the counter reached 6, out[0] was high because the 2nd bit of the counter was 1. In addition, out[1] was low because the 4th bit of the counter was 0.

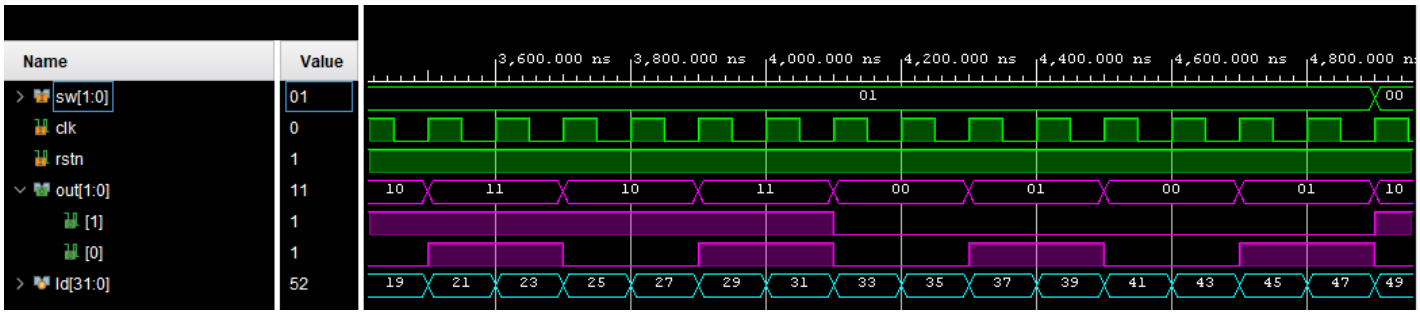

Figure 6: Counter Behavioral Simulation with Switch 0 On

Looking at figure 6, the design functioned as expected when only switch 0 was on. The counter counted up by 2 at the rising edge of the clock, and the output accurately reflected the value of the 2nd and 4th bit of the counter.

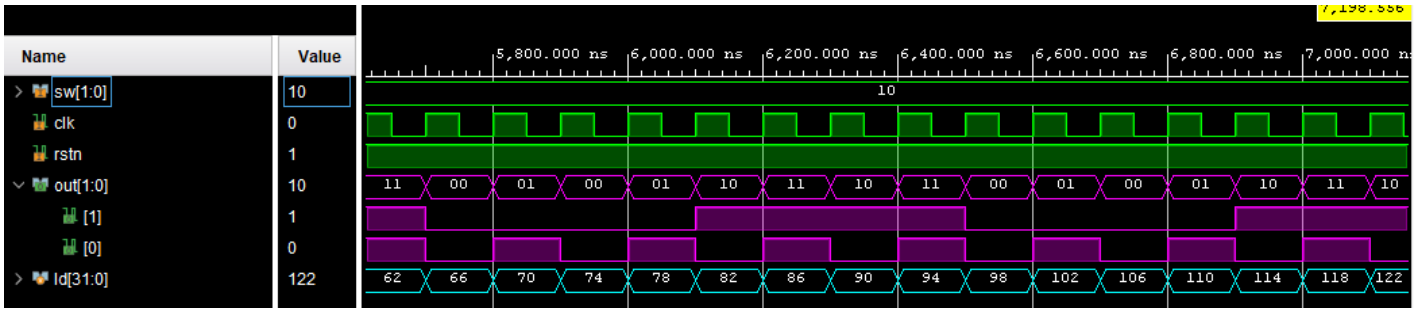

Figure 7: Counter Behavioral Simulation with Switch 1 On.

Looking at figure 7, the design functioned as expected when only switch 1 was on. The counter counted up by 4 at the rising edge of the clock, and the output accurately reflected the value of the 2nd and 4th bit of the counter.

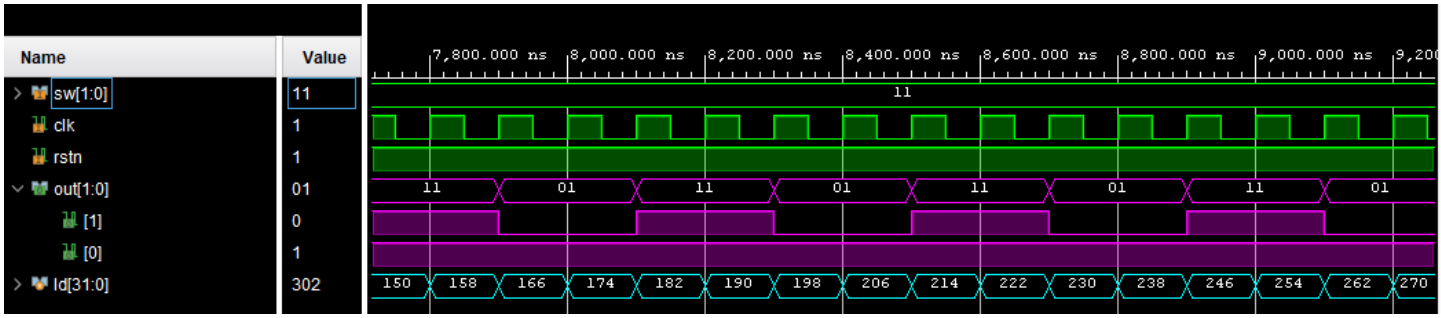

Figure 8: Counter Behavioral Simulation with Switches On

Looking at figure 8, the design functioned as expected when both switches were on. The counter counted up by 8 at the rising edge of the clock, and the output accurately reflected the value of the 2nd and 4th bit of the counter. Once the design was proven to operate properly, it was uploaded to the board and tested. Just as with the behavioral simulation, the board was tested in each switch position to make sure it was functioning properly. It functioned as expected in all cases, and the change in frequency was reflected through the LED lights blinking at different frequencies. The test for switch 0 and switch 1 both being on can be seen below in figures 9-10.

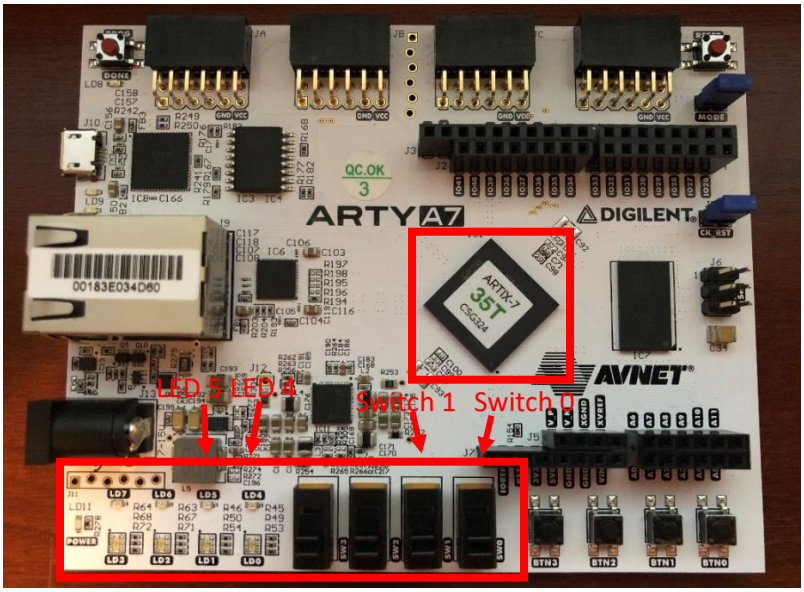

Figure 9: Arty A7-35T

Figure 9 shows the Arty A7-35T development board with the key parts used in this design highlighted in red. The bottom box contains LED 4 and LED 5, as well as switch 0 and switch 1. The top box contains the Artix-7 35T FPGA.

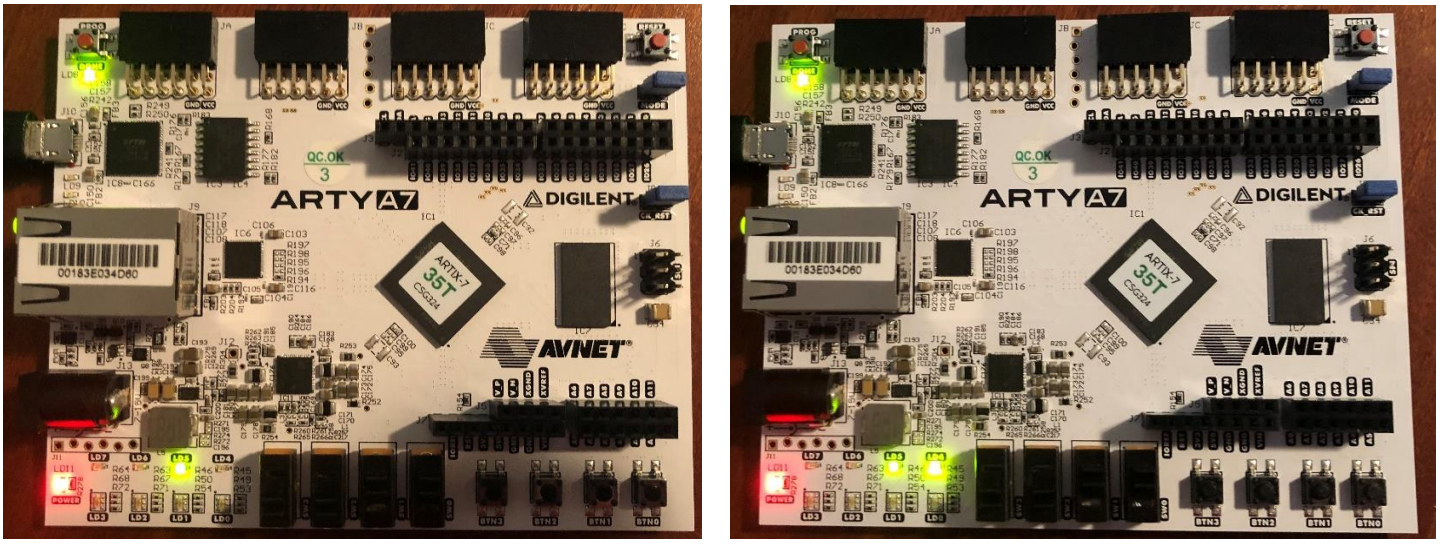

Figure 10: Arty A7-35T with Switch 0 and Switch 1 On

Looking at figure 10, the design functioned as expected. LED 4 and 5 began to blink at a higher frequency when both switches were on. Because the frequency was eight times the original, LED 4 was blinking at a frequency of 6 Hz, and LED 5 was blinking at a frequency of 23.52 Hz. At these frequencies, LED 4 was blinking very rapidly, and LED 5 appeared not to be blinking at all. Overall, the design was successful.

Conclusion

Throughout this summer project I gained a tremendous amount of valuable knowledge and skills that will help me moving forward in my career. I learned about digital electronic systems, how to plan and design SoC designs, how to code in Verilog, and how to work with FPGAs. In addition, I learned what it takes to design and build data acquisition and electronic instrumentation systems. Using the skills learned throughout the summer, I was able to design and build a fully functioning user-selectable frequency generator. This project gave me the opportunity to learn and appreciate real engineering work. FPGAs are so important in radio astronomy and today's electronics. This technology is going to become more important and advanced as time goes on. After completing this summer project, I feel confident that I can continue to work with data acquisition systems and develop successful FPGA designs.

Acknowledgements

I thank my mentor, Stephen Wunduke, for teaching me and guiding me through this summer project. I also thank the National Radio Astronomy Observatory for providing me with this amazing opportunity.

Reference Materials

[1] *What is an FPGA? Field Programmable Gate Array*. Accessed on: Aug. 23, 2020. [Online]. Available: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html

[2] *Arty A7 Reference Manual.* Accessed on: Aug. 23, 2020. [Online]. Available: https://reference.digilentinc.com/reference/programmable-logic/arty-a7/reference-manual

[3] *Verilog Module Instantiations*. Accessed on: Aug. 23, 2020. [Online]. Available: https://www.chipverify.com/verilog/verilog-module-instantiations

Xilinx user guides and documents:

https://www.xilinx.com/support/documentation/application_notes/xapp520_7Series_HPIO.pdf

https://www.xilinx.com/support/documentation/user_guides/ug471_7Series_SelectIO.pdf

https://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf

https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf

https://www.xilinx.com/support/documentation/user_guides/ug475_7Series_Pkg_Pinout.pdf

https://www.xilinx.com/support/packagefiles/a7packages/xc7a35tcsg324pkg.txt

Useful resources:

https://public.nrao.edu/telescopes/radio-telescopes/

https://timetoexplore.net/blog/arty-fpga-verilog-02

https://www.chipverify.com/verilog/verilog-tutorial

https://www.ni.com/documentation/en/labview-comms/latest/fpga-targets/configurable-logic-blocks/

https://www.viewpointusa.com/IE/wp/fpga-basics-under-the-hood/

Appendix - FAQ

Q: What improvements can be made to this project moving forward?

A: A table listing all the frequency options for the device can be constructed. This would allow the user to know the bit and switch combination to use in order to generate a specific frequency. In addition, the design can be modified using cascaded counters as opposed to a single counter. This would allow the user to further scale-down the output frequency.