Applications of FPGAs in Music and Computer Architecture Education

Siena College Department of Computer Science
Matthew Harrison, Professor Pauline White

May 6, 2019

Introduction

This independent study has been an opportunity for me to explore FPGA technology and gain a breadth of experience that will continue to benefit my research. The project focused on computer architecture and musical applications, while providing an annotated bibliography to explain how each source is relevant in the scope of this project and overall FPGA science.

Topics that I learned in the Assembly Language and Computer Architecture class were vital throughout this research. We explored how FPGAs could be used in this class to provide hands on experience with computer hardware, and how an FPGA system could provide an educational assembly language programming environment.

FPGAs are great tools for signal processing, so they are viable for a number of musical applications. In this study, we experimented with creating a tone sequencer and tone generator, and researched other methods used in creating FPGA based instruments.

1 Overview of FPGAs

Field Programmable Gate Arrays (FPGAs) are a semiconducting device composed of arrays of logic cells. Those cells can be reconfigured to implement countless logic operations for specific tasks. This re-programmable hardware is the basis of FPGA technology, but it is quite often taken much further than just these logic cells. Companies such as Digilent and Papilio make FPGA development boards, which include peripherals that can be easily incorporated into a design, such as HDMI, seven segment displays, audio jacks, and USB[6, 4]. Most development

boards also have sets of I/O pins so the FPGA can send and receive signals. These I/O ports can be used to implement Peripheral Modules (PMODs), which are external devices made to expand the capabilities of FPGA applications. PMODs can be sensors, connectors, additional memory, or a variety of components. [3]

1.1 Applications

The versatility of FPGA applications is nearly endless. Reprogrammable logic allows programmers to design custom systems for a specific task, but one of the most common uses is prototyping. FPGAs are fast, however, not usually the fastest solution. Application Specific Integrated Circuits (ASICs) are exactly what the title implies: printed ICs specialized for a specific application. These semiconducting chips are found in most modern electronics and were invented long before the FPGA. Because ASICs are static printed devices, the internal connections and gates are optimally placed, and they are therefore among the fastest known implementations of logical operations. Many silicon chip production companies like Intel or Texas Instruments begin product development on FPGA hardware. Designs can be reconfigured as many times as needed until the configuration is optimized, at which point it can be printed as an ASIC and used in a device. FPGAs are commonly used in Digital Signal Processing(DSP). They can quickly process data, and are capable of performing Fast Fourier Transforms to see audio/radio in a amplitude over frequency graph. It is also possible to implement a variety of filters, and connect to a variety of I/O devices.

2 Integrated Development Environments

2.1 Vivado Design Suite

Vivado provides all the necessary tools to design and program FPGA hardware. Designs can be made either by HDL programming in VHDL or Verilog, or using Vivado's IP Integrator. The Vivado IPI uses block diagrams to create graphic representations of a design which can then be validated and translated into an HDL wrapper. The IP Integrator wizard speeds up this flow by giving the option to run a connection automation, making all necessary connections such as connections to a clock source. Vivado also allows for High Level Synthesis, meaning the implementation of high level code, such as C, on FPGA IP. The Vivado IPI has blocks to resemble software components and even microprocessors.[6]

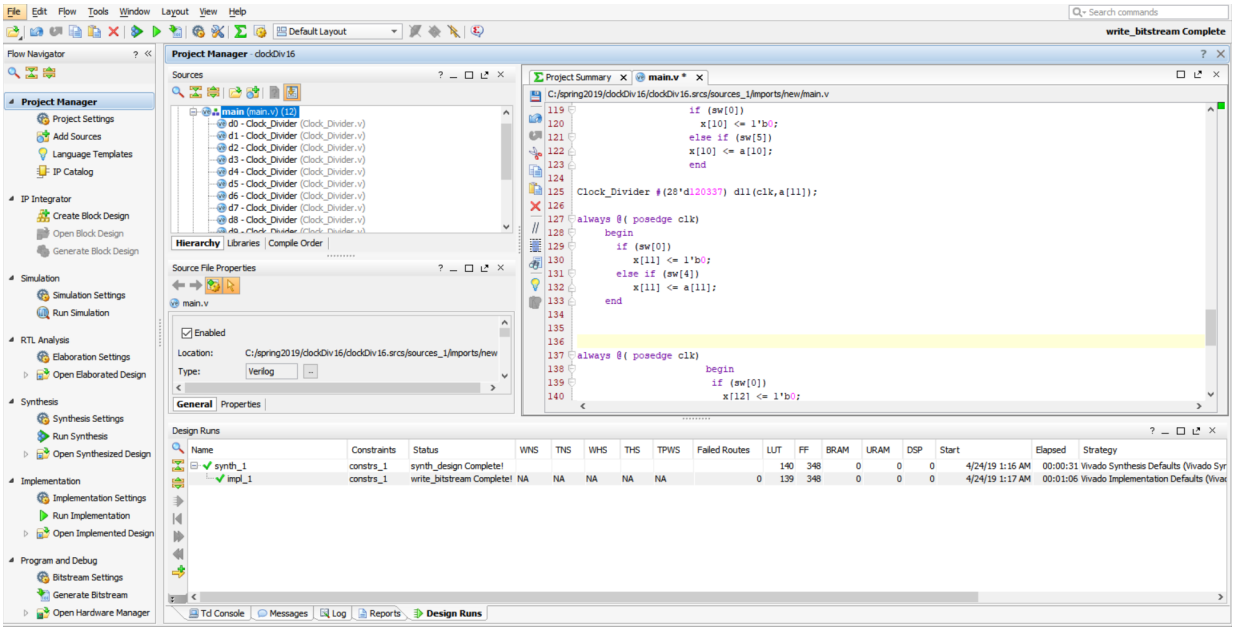

Figure 1: The Vivado Design Suite provides tabs to quickly navigate through stages of design flow, a source file hierarchy panel, a console/log window, and a window for code and block designs

2.2 System Generator

The Xilinx System Generator for Matlab uses the Simulink environment to create and implement FPGA designs. Simulink uses block designs to model systems. System Generator gives access to the default Simulink blockset, and a Xilinx blockset that allows a Simulink design to communicate with an FPGA, or design IP or bitstreams for a specific FPGA device. [6]

2.2.1 Hardware Co-Simulation

System Generator allows for a Simulation to run both on the host PC and the attached FPGA. This can be very useful for digital signal processing designs, particularly when validating output from the board. [6]

2.3 Software Development Kit

The Xilinx Software Development Kit (SDK) allows programmers to embed software designs on an FPGA. This can be done through microprocessor design such as the MicroBlaze IP, or using the dual core ARM Cortex processor built into the Zynq 7 series FPGAs. The Vivado IP integrator can interface an FPGAs

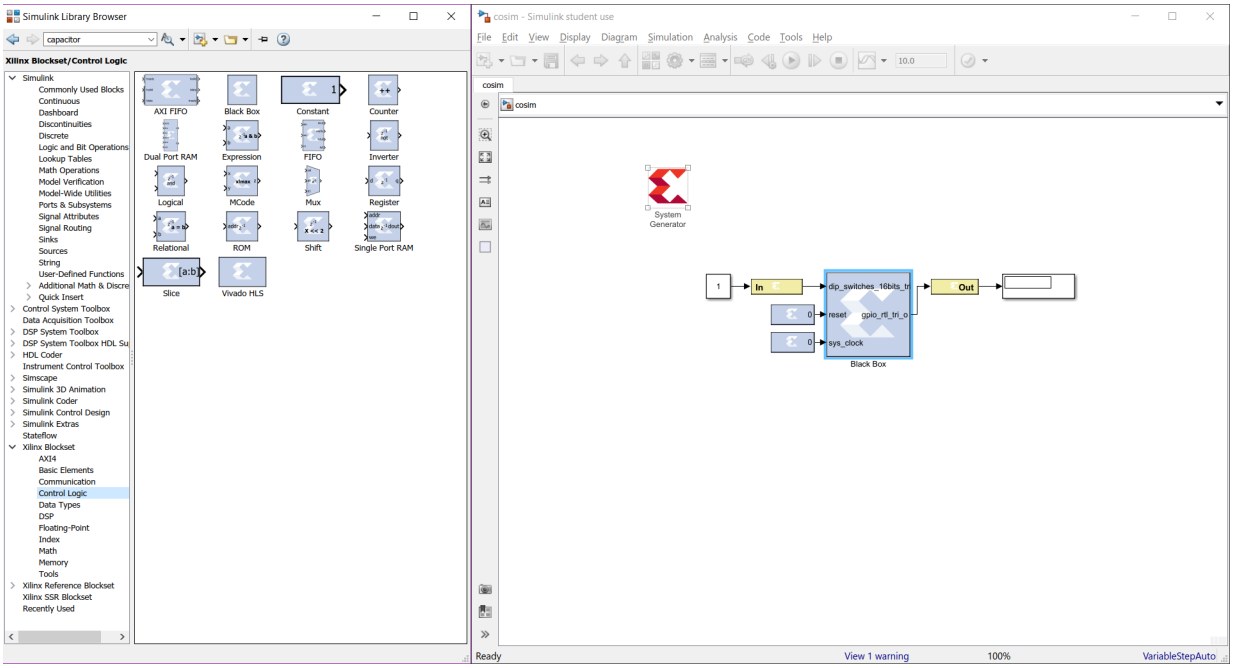

Figure 2: The left shows the library of Simulink and Xilinx blocks, the right is a cosimulation block design

reprogrammable hardware with the microprocessor, which can then be opened with SDK to implement a software design on the hardware. SDK allows for the creation and debugging of software, and can be used to run a design on an FPGA. If the design supports communication with the host PC, the SDK terminal can be used.[6]

3 Computer Architecture

The bulk of FPGA design relies on register transfer level logic. In Assembly Language and Computer Architecture courses, problems are approached using Reduced Instruction Set Computer (RISC) Architecture. Programming in this way requires using registers to control data from memory, using combinational and sequential logic, and understanding computer clock speeds.These skills are all essential to programming FPGAs.

Some concepts in Assembly Language and Computer Architecture can be difficult to visualize and interact with, which is why FPGAs are a viable tool for lab education. Simple FPGA programs can allow you to create, modify, and test multiplexers, encoders, adders, latches, flip-flops, and logic gates, while also requiring the use of registers to handle data.

Faculty at the University of Ljubljana in Slovenia designed an FPGA based education system for their computer architecture classes. They use a Hypothetical Microprocessor (HIP), which is a hardware simulation of a pipelined CPU capable of depicting the processor's state at a given moment. They designed a custom Integrated Development Environment (IDE) and Graphical User Interface (GUI) specialized for enforcing students' understanding of complicated computer architecture topics. The HIP is programmed using fifty-two 32-bit instructions with RISC behavior, allowing students to use the same type of instructions as in a CPU emulator environment, while being able to envision every step of the instruction pipeline and data output using a debugger. [5] The use of FPGAs in these classes can be a challenge due to a steep learning curve in FPGA design, and excessive time spent synthesizing and implementing designs, and generating bitstreams.

4 Musical Applications

As mentioned in section 1.1, FPGAs are ideal for DSP designs. This is not only limited to reading signals, there are multiple ways to generate signals with FPGAs to create music as well. The most explored method was using clock dividers. The Basys3 board has a clock speed of 100 MHz, but by writing code that will turn an output on and off at a division of that speed, we get audible frequencies. This method is how some 20th century synthesizers worked, only using much bigger circuitry. This was also used in the sequencer project, discussed in section 5.2, and the tone generator in section 5.3. These designs use hardware descriptive code to create basic instruments without software.

Another common tactic is to use a wavetable. This is done by storing all the possible signals in a table, and outputting them depending on input, typically from a piano keyboard. One period of each waveform is stored, and played in a loop. The frequency of this waveform can be changed by looping faster or slower. Signals in a wavetable can be derived from a recording, or designed to give a particular sound. Wavetables can be made with hardware design, but are often implemented in software using a microprocessor design, and MIDI communication support for the keyboard. [9] Tables can also store audio files. Similar to a wave table, output is chosen based on input. Instead of a waveform, this method plays back an audio file of the note.

Waldorf has announced the Kyra synthesizer, powered by FPGA technology. This instrument will use multiple oscillators and wavetables to achieve a multitude of sounds and effects [8]. The Kyra synthesizer sets an example of FPGAs' potential in music. Many of their included features can be elegantly implemented on FPGA, and will certainly be relevant for further research on this topic.

5 Projects

5.1 Logic Gates

While exploring how we can use FPGAs to teach computer architecture, we used both System Generator and Vivado to create NAND gate structures. Experimenting with logic gates on FPGAs can be a great exercise in hardware programming and logic, but it does require a significant amount of background knowledge or learning time, and it can take several minutes of computing time every time Vivado synthesizes the design, implements it, and generates the bitstream.

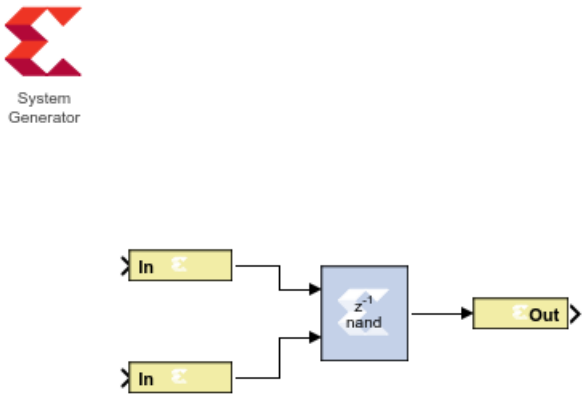

Figure 3: System generator IP that was exported to vivado

5.2 Sequencer

Following a tutorial by Lincoln Tran on instructables.com, I created an 8 step sequencer, able to play a unique pitch on 8 different beats to make a looping pattern. The method of creating sine waves to output in this design was to toggle an output on an off according to a frequency derived from the clock signal. Unlike many other music applications which use software to store tables of frequencies that may be outputted, this design creates its output purely from hardware. To implement this design on a Basys3 board, rather than the Nexus2 the creator used,

I had to create a ucf constraint file in Vivado, being sure to use the names for the pins as declared in the top module of the design.

5.3 Tone Generator

To try out some of music related techniques I learned, I programmed the Basys3 board to play every note between A4 at 440 Hz, and a5 at 880 Hz depending on which switches on the board are on. This uses a clock divider, similar to the how the sequencer generates its tones. To implement switches, I programmed a flip-flop to read the state of the switches at every step of the board's 100 MHz clock and output to the speaker accordingly.

```
Clock_Divider #(28'd120337) dl1(clk,a[11]);

always @( posedge clk)
     begin
       if (sw[0])
          x[11] <= 1'b0;
       else if (sw[4])
          x[11] <= a[11];
     end
```

Figure 4: This Verilog code shows the instantiation of a clock division, and the flip-flop that reads the reset and tone switches

Conclusions

Throughout this independent study, I gained valuable experience in computer architecture, analog synthesis, and FPGA design. The structure of the study allowed me to pursue a passion for studying music technology, while learning more about computer and FPGA architecture. The music related research in this independent study has provided the basis for future projects, such as implementing a MIDI keyboard. Further research in computer architecture education will likely involve looking into designs like the HIP, as that is the most plausible way to use FPGAs in the Assembly Language and Computer Architecture course.

References

[1] Blaine C. Readler *VHDL By Example*. Full Arc Press, 2014.

[2] Blaine C. Readler *Verilog By Example*. Full Arc Press, 2011.

Readler's texts provide a plethora of code references for VHDL and Verilog, the most common FPGA languages. Working through examples in these books provides diverse experience in hardware descriptive programming, but they are also helpful for topic specific reference.

[3] David Romano *Make: FPGAs*. Maker Media, 2016.

This text gives beginner level tutorials for a variety of practical applications that highlight the strengths of FPGAs.

[4] Simon Monk *Programming FPGAS: Getting Started with Verilog*. McGraw-Hill Education, 2017.

This book focuses on designs for the Papilio One board, but the information is relevant for most development boards. Monk gives information on logic and FPGAs, then continues to provide a number of examples to familiarize the user with verilog.

[5] Patricio Bulić, Veselko Guštin, Damjan Šonc, Andrej Štrancar *An FPGA-Based Integrated Environment for Computer Architecture*. University of Ljubljana, 2010.

This paper documents how FPGAs are used in Assembly Language classes at the University of Ljubljana. See section 3.

[6] Xilinx Inc., *Self-Paced Tutorials*
`https://www.xilinx.com/support/documentation-navigation/self-paced-tutorials/see-all-tutorials.html`

These tutorials cover a variety of areas in FPGA design. Some tutorials come with files to be modified.

[7] Akshay Sridharan, *Verilog Syntax and Structure*
`https://www.utdallas.edu/ akshay.sridharan/index_files/Page5212.htm`

This website was used as a Verilog reference guide. The tables showing

logical syntax are a useful quick reference for programming and debugging. The website also explores other features of the Verilog language.

[8] Synthtopia, *Waldorf Kyra Synthesizer at SynthPlex 2019*
`https://www.synthtopia.com/content/2019/04/02/waldorf-kyra-synthesizer-at-synthp`

This is a summary of the announce of one of the first FPGA based synthesizers on the market. It lists a variety of specifications and information about the instrument.

[9] McGill University, *Wavetable Synthesis*
`https://www.music.mcgill.ca/ gary/307/week4/wavetables.html`

This webpage summarizes the concept of wavetable synthesis and how it is used for audio.