FPGAs for Signal Processing

By John Davis, ECE '22

Abstract

This document serves as an overview of the considerations that must be taken to implement high-performance signal processing algorithms on FPGAs.

Introduction

Signals are everywhere. A signal is any piece of information from the sound waves that carry speech to the light waves that let us see. For the most part, signals are not useful by themselves and need to be transformed or processed to do something useful. This operation is called signal processing. In the example of sound waves, the human brain performs signal processing to decode speech sounds into words.

Digital signal processing (DSP) is a subclass of signal processing that uses digital systems to do mathematical operations on signals. Effective signal processing requires powerful digital systems that are very good at crunching numbers. For radio signals, the most powerful processors make use of FPGAs.

FPGAs are reconfigurable digital circuits that can effectively mimic complex digital behavior. In essence, they serve as a way for developers to create custom digital circuits without the manufacturing overhead or cost of having to fabricate custom silicon. FPGAs have close to the same performance as custom silicon chips, making them useful for high performance applications.

Digital signal processing is a common use case for FPGAs where traditional CPUs used by computers can be orders of magnitude too slow. The main advantage of FPGAs over CPUs is their ability to perform many complex calculations at the same time. This is made possible by the FPGA's inherent capability to configure its resources to work in parallel whereas a CPU at best can only perform a single instruction at a time.

Figures of Merit for DSP

Bandwidth

Bandwidth is a measure of the rate of data the system can intake. In a water pipe, this can be thought of as the diameter of the pipe. For a given speed of the water, a wider pipe will be able to carry more water overall because there is more water contained in each cross-section.

When continuous analog signals are converted to a digital representation, they are quantized both in time and in magnitude. Quantization error in magnitude is not an issue that will be discussed in this document as it is a function of the quality of the analog-to-digital converter hardware which is not part of the digital signal processor. Modern analog-to-digital converters can have bit depths of over 32bits meaning that such error is largely negligible.

Time quantization on the other hand is critical to the system's performance. Signals quantized in time are subject to the Nyquist theorem which states that the maximum recoverable frequency for a signal quantized in time is half the quantization sampling rate (Oppenheim).

Bandwidth is typically expressed in terms of its sample rate in samples per second. For a system with a sample rate of 1MHz, 1 million samples will pass through the input port per second. The output port will also pass 1 million samples every second.

It is important to note that system bandwidth gives zero information about the time delay between the input and output ports. Going back to the pipe analogy, a 100-mile-long pipe could pass 1 gallon per second all the time but it might take over a month for a water molecule to travel through the full length of the pipe.

Latency

Latency is a measure of the time delay between unprocessed data into the system and processed data out of the system. This is equivalent to the amount of

time it takes a water molecule to get through the pipe. An extreme example to demonstrate this could be a 100-mile pipe as thin as a human hair. If this pipe has the same rate of water molecules per second as the wide pipe, those water molecules would have to travel at very high speeds. As a result, it could take on the order of minutes rather than days for the water to travel those 100 miles.

Because digital systems tend to be clocked, data latency is usually measured in terms of samples. For example, a latency of 10ms in a 48kHz system would be expressed as 480 samples.

FPGA Hardware

Logic Cells

FPGAs are composed of thousands of small logic cells that are configured and connected to form the behavior of the final circuit. In an FPGA, the design is not stored in a cheap memory unit like a CPU program and instead is stored in each of the FPGA's logic cells. As a result, large "programs" are very expensive for FPGAs. FPGA cost increases with more logic cells, therefore the appropriately sized FPGA must be chosen for the target application (Davis).

DSP Blocks

FPGAs have specialized blocks capable of multiplying and accumulating large numbers within a single clock cycle. Such circuits would take up an unreasonable amount of space if they needed to be implemented in logic cells and are thus implemented in custom silicon. These blocks are typically allocated to tasks automatically by the FPGA's development software in order to meet the design constraints. As a result, the designer typically does not have to worry about interfacing with DSP blocks. This makes it easy for a design to be ported from one FPGA to another (Davis).

Block RAM

FPGAs usually have fast dual port memory blocks called Embedded Block RAM (EBR) that can be written to/read from in a single clock cycle. These blocks can be instantiated to store information when the FPGA configures itself on startup. This allows them to act as both a RAM and a ROM. This property is imperative to making fast custom algorithms because it allows for the use of lookup tables as a replacement for complicated logic. An example of a lookup table's use could be to calculate e^x (Davis).

Clock Rate

FPGA logic is subject to a maximum clock rate beyond which certain hardware such as DSP blocks and EBRs will not work. This limits the bandwidth and increases the latency of the FPGA for signal processing tasks. As a result, the FPGA's maximum clock rate needs to be considered when choosing the correct FPGA for a given task.

Effective Resource Utilization

Efficient FPGA development is an art. It is the responsibility of the designer to utilize the FPGA's hardware to its maximum potential. There are a few general strategies for effective resource allocation that every designer should be aware of.

State Machines to Emulate CPU Behavior

An FPGA can be used similarly to a CPU. An example of such hardware could be a multiplier that calculates the product of two variables, putting the product in a third variable. Each state in the state machine works similarly to a CPU assembly instruction, moving data between the necessary variables and moving to the next instruction (Patel). The sacrifice in this case is both latency and throughput since it trades hardware complexity for clock cycles. The latency per sample is determined by the number of states in the state machine and the throughput is one sample per each run of the state machine from beginning to end.

Unlike a CPU instruction, each state does not have to execute a single task. If the same FPGA hardware is not being utilized twice in the same instruction, there is no issue with performing multiple computation steps within a single clock cycle.

Pipelining

Pipelining is a similar strategy to using state machines in that it splits the task into a set of steps, however each step has its own dedicated hardware. This has the advantage of giving a throughput of one sample per clock cycle.

A good analogy for this is a bucket brigade where

firefighters line up in a human chain, passing buckets from one person to the next. In this way, data (the buckets) move through the chain with a throughput of one sample per clock cycle, but the latency is determined by the length of the chain.

The disadvantage when compared to state machines is that algorithms with many multiplication steps will use up the available DSP blocks. The choice to pipeline versus using a state machine is one that should be made by the designer.

Large Combinational Logic

Combinational logic occurs when the output of a system is determined solely by its current inputs. Each step of a state machine or pipeline tends to be made of small chunks of combinational logic, but those can be combined into a very large chunk of logic by feeding each step into the next with no latches between (Rashid). Because such algorithms have large propagation delays, they tend to span multiple clock cycles, though less than the time required if the algorithm was implemented using either method above.

Such systems tend to be very resource intensive but have the lowest latency. Because they tend to take more than one clock cycle to complete, large combinational circuits will have worse throughput compared to pipelines, but better throughput than state machines. For the most part, large combinational circuits should be avoided.

Nontrivial Mathematical Functions

For complicated mathematical functions, there is no simple in-built hardware for calculation. In most cases, approximations can be used.

Power Series Approximations

Power series approximations are an important tool to create custom nonlinear functions. The most well-known power series is the Taylor series. It has use cases in many applications (Dawkins).

The Taylor series is defined as follows where a is the center point:

$$\begin{aligned} f(x) &= \sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x-a)^n \\ &= f(a) + f'(a)(x-a) \\ &+ \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 \\ &+ \dots \end{aligned}$$

Note that the only thing actually being calculated is $(x\text{-}a)^n$. Everything else is precomputed. The error between $f(x)$ and the Taylor series decreases as the number of terms increases, so some experimentation is required to maximize performance. Error also increases as x diverges from the center point a. Another strategy can be to choose a set number of terms and run an optimization function such as least squares to optimize the function's performance over a certain range of x values, letting the optimizer choose the coefficients and a.

Lookup Tables

Lookup tables are essentially arrays that map an index to a precomputed value. They can be easily linearly interpolated and are good for cases where power series approximations cannot be used. The main limiting factor with lookup tables is that they can eat up a lot of block RAM.

Conclusion

Sitting at the crossroads of configurability and speed, FPGAs are uniquely positioned to implement DSP algorithms. To make full use of their capabilities, it is critical to know the right tools and techniques. When developing for FPGAs, thoughtful design is the difference between what is possible and what is impossible.

References

1. Davis, J. (2021). *Techniques for Audio Processing with Low-Power FPGAs.* https://davisynth.com/wp-content/uploads/2021/11/White_Paper_Davisynth_1_3.pdf

2. Dawkins, P. (2020). *Section 4-16 : Taylor series.* Lamar University. https://tutorial.math.lamar.edu/Classes/CalcII/TaylorSeries.aspx

3. Fryad M. Rashid, P.-L. C. (2015). *Combinational logic circuits.* https://people.cs.clemson.edu/~yfeaste/855Assignments/presentations/Team6-Combinational%20Logic%20Circuit.pdf

4. Oppenheim, A., Willsky, A., Nawab, S., Hamid, w., Hern´andez, G., & Young, I. (1997). *Signals & systems.* Prentice Hall. https://books.google.com/books?id=g2750K3PxRYC

5. Patel, M. K. (2017). *Fpga Designs with VHDL.* https://vhdlguide.readthedocs.io/en/latest/vhdl/fsm.htm