Designing Architecture for Multi-Accelerator Applications

Riley Wood
Computer Architecture Lab
Tufts University
rjw245@gmail.com

May 13, 2016

Abstract

This paper summarizes my research on hardware accelerators and details the work I have done regarding the development of an architecture to support multi-accelerator applications. Computer architects are preparing to confront an impending limit on the number of transistors on chip that can be used at any given time. This utilization wall will limit the extent to which computers can benefit from increasing transistor counts. Specializing the transistors on chip is one solution which seems to be gaining traction among the research community and in industry. As researchers pursue this option, they have begun to develop new accelerator-rich architectures (ARAs) to handle the challenges posed by specialization. For researchers to conclusively evaluate and compare these architectures, they will need a catalog of applications which use many accelerators. This work focuses on the design of an architecture upon which such applications can be built. The architecture is developed for the Zynq-7000 line of systems-on-chip and serves as a baseline architecture for ARA research. This work was conducted during the spring of 2016 under the supervision of Professor Mark Hempstead as part of an independent study in hardware accelerators.

1 Introduction

The Motivation for Specialization It used to be that as transistors grew smaller, chip power consumption would remain constant due to a principle known as Dennard scaling [4]. Since 2006, Dennard scaling has begun to fail due to the fact that submicron sized transistors are more susceptible to leakage current and other non-ideal phenomena [10]. This means that placing more transistors on chip incurs greater power consumption, yet modern computers are unable to consume any more power and still be kept cool. For example, the power consumption of Intel CPUs plateaued back in 2006 [6]. To continue increasing the number of transistors on chip and stay within power budget, designers will have to selectively under-clock or turn off portions of the chip. As the number of transistors on chip grows, a smaller and smaller percentage of them will be usable at any given time. This *utilization wall*, as it is called by Venkatesh et al. [15], threatens to limit the extent to which computers can benefit from increasing transistor counts in the future.

Several solutions to the utilization wall have been proposed, including shrinking the number of transistors on chip, "dimming" regions of the chip, developing new, more efficient transistor technologies, and specializing groups of transistors. The first of these, shrinking transistor counts, clearly means the end of Moore's Law scaling. This has been called the "most pessimistic" solution to the problem [14], and understandably, many in industry are hesitant to make such a big concession. "Dimming" technologies make the transistors on chip more energy-efficient, thereby increasing the number of transistors that can be used while staying within power budgets. However, these techniques often cause a decrease in performance. Dynamic voltage and frequency scaling (DVFS) is a common technique to reduce the power consumption of transistors when high performance is not required. Some work has been done to make DVFS even more effective for modern architectures [7]. Other work targets energy efficiency by trying to make transistors more reusable. For example, DySER from the University of Wisconsin-Madison is an element of the processor pipeline that can be reconfigured to implement one of multiple functions [5]. Because DySER is reconfigurable, energy is only expended on the data path that is currently configured. This saves energy compared to architectures where idle data paths still consume power. The third approach—the development of new transistor tech—addresses the failure of Dennard scaling by abandoning CMOS altogether. Some of the work in this field includes tunnel field-effect transistors (TFETs) [12] which use quantum tunneling to implement a switch, and nano electro-mechanical switches (NEMS) [2] which take advantage of the power efficiency of nano-switch technology. Lastly, specialization addresses power limitations by spreading computation across an array of specialized blocks of transistors, called accelerators. Rather than have a monolithic processor do the majority of computation and offload a small portion to specialized hardware (as is the norm today), the vision for specialization in the future has computation bouncing between many specialized regions of the chip [14]. Because there are more transistors on chip than can be used at any one time, area can be "spent" on accelerators specialized for certain computations. These accelerators use fewer transistors than a general-purpose processor to implement specific functionality. Thus, on a fully specialized system, the computer will use a minimal number of transistors at any given time, thereby maintaining performance gains despite the utilization wall.

Accelerators are already being adopted in industry. The International Technology Roadmap for Semiconductors (ITRS) released a report in 2007 which predicted that chips will contain almost 1500 accelerators by the year 2022 [1]. Today, accelerators can be found in common devices like the iPhone [13] and the PlayStation 3 [8]. Many have looked to specialization as a promising answer to the utilization wall, but for accelerators to solve the problem entirely, they will need to become pervasive and numerous in future architectures. Integrating accelerators in large numbers and moving away from large, general-purpose processors requires significant work in the development of new architectures.

The Challenges for Accelerator-Rich Architectures The shift toward specialized computation brings about new challenges for architectures in areas such as on-chip communication, memory structure, resource sharing and contention between cores, and others. As computation becomes less centralized, it will be important for accelerators to communicate with one another without the assistance of a central controller (which would become a bottleneck as the number of accelerators grows). This is also called streaming. To stream data from one accelerator to another, the accelerators must all agree on a common interface. Architectures have been developed which allow for such an interface, such as the Accelerator Store which uses FIFOs to send data between accelerators [9]. They do this to avoid using a DMA engine, which they believe would artificially constrain the interfaces of accelerators in the system. Conversely, another architecture, AXR-CMP [3], uses a DMA to interface the accelerators. These two architectures are at odds in terms of how they implement streaming.

The Value of Accelerator-Rich Applications Mention the RoboBee [18].

Requirements for the Testbed Architecture Talk about the requirements of the application (streaming, processor level of access) and the needs of researchers (Dave in specific, who needs the memory interconnect to be interchangeable with his own custom design. Hence the ”MODULAR INTERCONNECT” block. But I guess you talk about that specifically in the technical documentation).

2 Technical Discussion

In this section, I will discuss the details of my many-accelerator architecture. This architecture is intended to be instantiated on a Zynq-7000 All Programmable SoC [16]. These systems-on-chip contain a general-purpose ARM processor and reconfigurable FPGA fabric. Developers can instantiate custom hardware in the FPGA for the processor to use. In this project, a ZedBoard was used [17].

I used Vivado v2015.4 to design my architecture. The block diagram of the architecture is shown in Appendix A. The diagram consists of several key blocks. These are:

- **Zynq Processing System:** The general-purpose processor in the Zynq SoC. Exposes a clock, reset, and AXI port to the FPGA fabric.
- **AXI Central DMA:** Used to move data between the processor’s main memory and the three FFT accelerators.
- **AXI Interconnect Crossbars:** There are several AXI crossbars in the design which establish a master-slave hierarchy. More on this later in this section.

2.1 Master-Slave Hierarchy

Talk about how the Zynq has control of the DMA has control of the FFT. But also, the Zynq can circumvent the DMA and access the memories of each accelerator directly.

3 Evaluation

It is important to validate my design to ensure its functionality. Here I discuss my test procedure, my findings, and my conclusions on how well my design meets the requirements. The architecture evaluated here is shown in Appendix A.

3.1 Validating the FFT Accelerator

After synthesizing the strided FFT from MachSuite [11] in Vivado HLS, it was important to verify that the accelerator actually computed an FFT correctly. To proceed, the FFT accelerator’s results must exactly match those of the FFT computed in software. My results show that these expectations are met—the FFT hardware accelerators compute exactly the same results as the FFT software.

3.1.1 Methodology

The results of each of the three FFT accelerators present in the architecture were compared against the results of the MachSuite C code in order to validate them. The MachSuite C code was copied and pasted into the Xilinx SDK where it was compiled for the Zynq processor. The Zynq processor then computed seven FFTs using the same inputs: one in software; the other six on each of the three hardware accelerator instances, either using a DMA engine or not. Results from each were checked for equality using the function in Listing 1. The results of the hardware FFT are passed as arguments `real_hw` and `img_hw`, and the results of the software FFT are `real_sw` and `img_sw`.

```
static void compare_results(double real_hw[FFT_SIZE], double
      real_sw[FFT_SIZE], double img_hw[FFT_SIZE], double
     img_sw[FFT_SIZE]) {
  int i;
  for(i=0; i<FFT_SIZE; i++){
    if(real_hw[i] != real_sw[i] or img_hw[i] != img_sw[i]) {
      xil_printf("ERROR: Different results at bin %d\n", i);
    }
  }
}
```

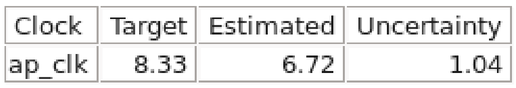
Listing 1: FFT Validation Function

3.1.2 Results

No differences arose between the results of the software FFT and any of the hardware-accelerated FFTs. This means that all three hardware FFTs correctly implement the strided FFT algorithm.

3.2 Examining Accelerator-Rich System Performance Relative to Software

With the previous test, I showed that the FFTs compute the correct results. With this test, I demonstrate the usefulness of the DMA engine and clock scaling in improving the performance of the FFT in hardware versus in software.



Clock	Target	Estimated	Uncertainty
ap_clk	8.33	6.72	1.04

Figure 1: Timing characteristics taken from the C Synthesis Report for MachSuite's Strided FFT, generated by Vivado HLS. Targeting a clock frequency of 120MHz.

3.2.1 Methodology

The FFT was re-synthesized using Vivado HLS with a target clock frequency of 120MHz. The timing portion of the C synthesis report is printed in Figure 1. The block diagram shown in Appendix A was synthesized with the peripheral clock (`FCLK_CLK0` from the Zynq processor) set to 100MHz and then 120MHz. At each frequency, execution time was recorded for the FFT in software, in hardware using the DMA engine, and in hardware not using the DMA engine. The XTime library from Xilinx was used to time sections of code, as shown in Listing 2.

```
XTime tStart, tEnd;
XTime_GetTime(&tStart);
// Region of interest goes here
XTime_GetTime(&tEnd);
time = (tEnd - tStart);
xil_printf("Region of interest took %d timer cycles.\n",time
    );
```

Listing 2: XTime example

The number of timer cycles spent computing each critical section was recorded. Dividing by the timer clock rate yields the time in seconds. The following critical sections were selected for timing:

- Sending inputs to the FFT
- Computation of the FFT
- Retrieving results from the FFT

No time was spent sending nor retrieving data in software since it operates on arrays in main memory. The FFT accelerator, on the other hand, requires data to be copied to and from its scratchpad memory. The DMA engine is used to move inputs and outputs between main memory and the FFT's scratchpad memory. When not using the DMA, the Zynq processor must manually copy the data. In all cases, the hardware implementation is synchronous, meaning the Zynq processor initiates each input/output copy one at a time and then must wait while each transfer takes place.

3.2.2 Results

Figures 2 and 3 show the time spent executing the FFT in software running entirely on the Zynq processor versus each step in hardware under different constraints. The runtime for each of the three FFT accelerators is shown at two different clock rates and with and without a DMA engine.

One will notice that using the DMA drastically speeds up execution. On average, the DMA speeds up all data transfer by approximately 16x as compared to the processor manually copying data. Unsurprisingly, the time spent computing the FFT is unaffected by the use of a DMA engine. Figure 3 gives a more

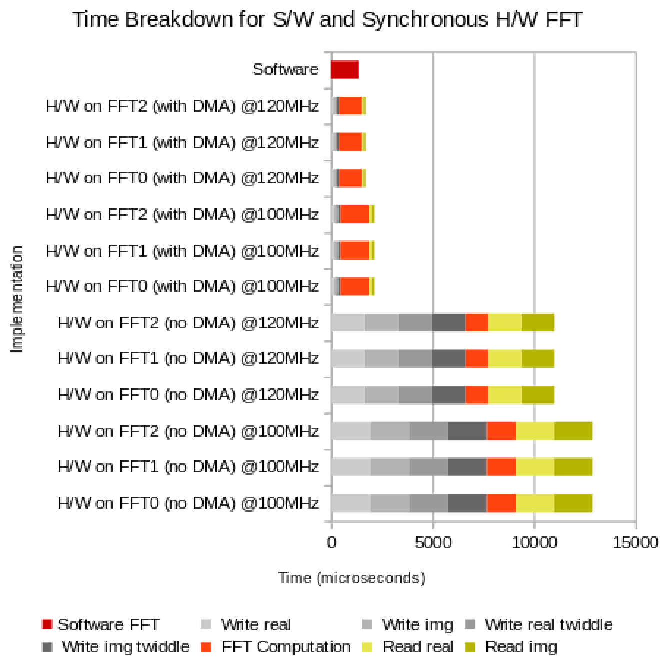


Figure 2: Time breakdown of an FFT done in software versus in hardware, showing all implementations (with/without a DMA engine, and with peripherals clocked at 100MHz and 120MHz). Each critical step is highlighted in a different color. Copying of input operands is highlighted in shades of gray, the actual computation in shades of red, and copying back results in shades of yellow.

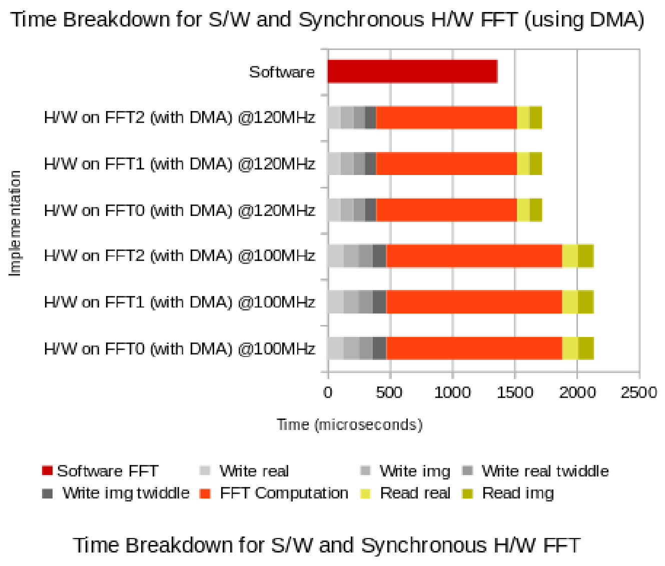


Figure 3: Close-up view of execution time for the hardware implementations with a DMA engine.

close-up view of how the hardware accelerator with DMA compares to software.

When increasing the clock frequency of the FFT accelerator, DMA engine, and AXI interconnect from 100MHz to 120MHz, one can see that every step of the hardware implementation speeds up, whether using the DMA or not. This is because the DMA and the Zynq processor are both constrained in how fast they can transfer data by the speed of the interconnect.

Finally, it is important to note that the software implementation is faster than hardware in all cases in terms of total execution time. However, this includes the time spent copying data to and from the FFT accelerator. In Figure 3, one can see that at 120MHz, the actual computation (shown in red) is completed faster in hardware than in software. The synchronous data transfer which comes before and after computation lengthens execution time such that there is no speedup over the software implementation. Transferring data asynchronously could amortize this overhead by allowing the Zynq processor to do other useful work while data is moved by a DMA.

4 Project Execution Evaluation

My original task was to develop an application that uses many accelerators in a way that highlights their varied interactions in a many-accelerator architecture. Along the way, I discovered that designing the underlying architecture itself would take a significant amount of time. Decisions needed to be made about how data would be moved between accelerators, what interface accelerators would need to comply with. To make these decisions and implement them, I needed to learn a number of programs—such as the Xilinx SDK, Vivado, and Vivado HLS—and also research hardware accelerators, accelerator-rich architectures, and general principles for architectural-level design.

Ultimately, I was unable to begin work on the application itself, but I have accomplished an important piece of this project in developing and validating the underlying architecture. To date, I have validated the current architecture design, highlighted areas for improvement and further development, and packaged the architecture so that someone else can easily pick up where I left off.

Appendices

A Architecture block diagram

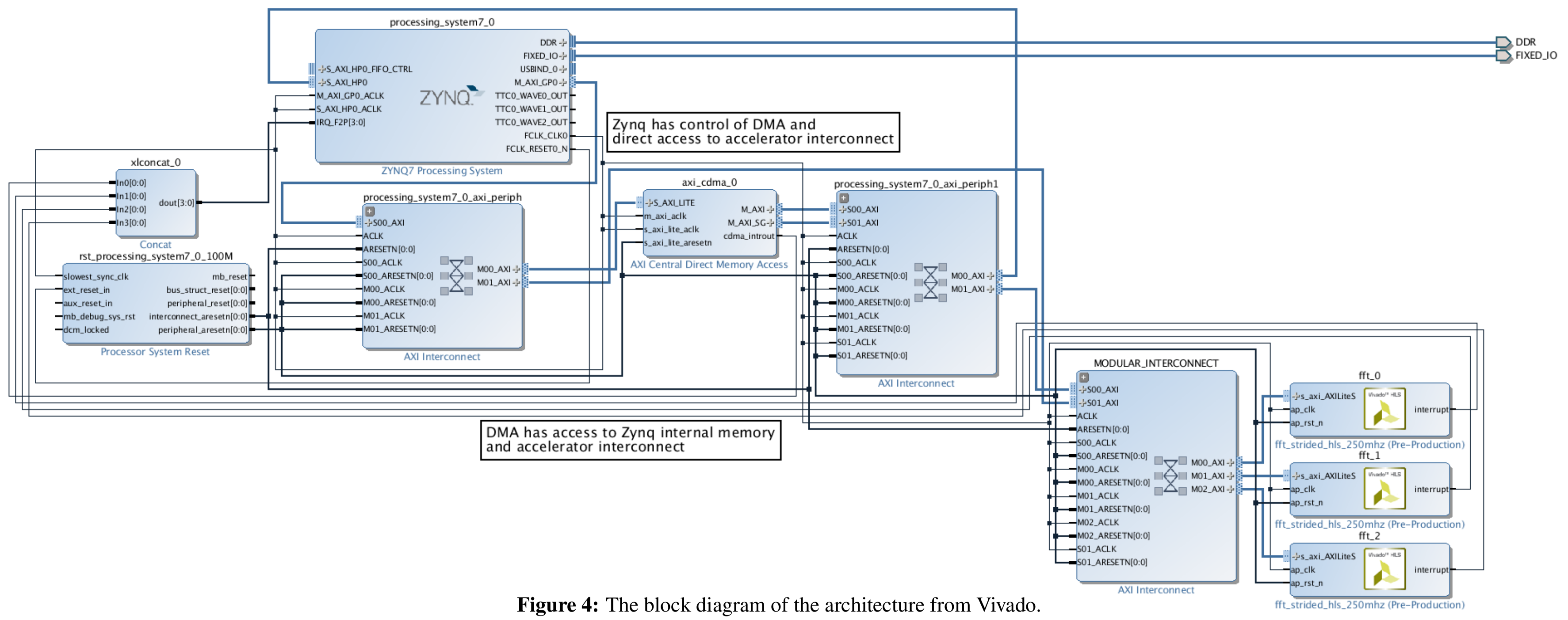


Figure 4: The block diagram of the architecture from Vivado.
Composed of a Zynq processor, AXI Central DMA,
and three FFT accelerators synthesized by Vivado HLS.

References

[1] International technology roadmap for semiconductors 2007 edition system drivers, 2007.

[2] F. Chen, M. Spencer, R. Nathanael, C. Wang, H. Fariborzi, A. Gupta, H. Kam, V. Pott, J. Jeon, T. J. K. Liu, D. Markovic, V. Stojanovic, and E. Alon. Demonstration of integrated micro-electro-mechanical switch circuits for vlsi applications. In *2010 IEEE International Solid-State Circuits Conference - (ISSCC)*, pages 150–151, Feb 2010.

[3] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman. Architecture support for accelerator-rich cmps. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 843–849, June 2012.

[4] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, Oct 1974.

[5] V. Govindaraju, C. H. Ho, and K. Sankaralingam. Dynamically specialized datapaths for energy efficient computing. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, pages 503–514, Feb 2011.

[6] J. Hruska. The death of cpu scaling: From one core to many and why were still stuck, Feb 2012.

[7] W. Kim, M. S. Gupta, G. Y. Wei, and D. Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, pages 123–134, Feb 2008.

[8] M. Kistler, M. Perrone, and F. Petrini. Cell multiprocessor communication network: Built for speed. *IEEE Micro*, 26(3):10–23, May 2006.

[9] M. Lyons, M. Hempstead, G. Y. Wei, and D. Brooks. The accelerator store framework for high-performance, low-power accelerator-based systems. *IEEE Computer Architecture Letters*, 9(2):53–56, Feb 2010.

[10] A. McMenamin. The end of dennard scaling, Apr 2013.

[11] B. Reagen, R. Adolf, Y. S. Shao, G. Y. Wei, and D. Brooks. Machsuite: Benchmarks for accelerator design and customized architectures. In *Workload Characterization (IISWC), 2014 IEEE International Symposium on*, pages 110–119, Oct 2014.

[12] A. C. Seabaugh and Q. Zhang. Low-voltage tunnel transistors for beyond cmos logic. *Proceedings of the IEEE*, 98(12):2095–2110, Dec 2010.

[13] Y. S. Shao, S. Xi, V. Srinivasan, G.-Y. Wei, and D. Brooks. Toward cache-friendly hardware accelerators. *Proc. Sensors to Cloud Architectures Workshop (SCAW), in conjunction with HPCA 2015*, 2015.

[14] M. B. Taylor. Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 1131–1136, June 2012.

[15] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor. Conservation cores: Reducing the energy of mature computations. In *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV, pages 205–218, New York, NY, USA, 2010. ACM.

[16] Xilinx. Zynq-7000 all programmable soc. `http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html`.

[17] ZedBoard.org. Zedboard. `http://zedboard.org/product/zedboard`.

[18] X. Zhang, M. Lok, T. Tong, S. Chaput, S. K. Lee, B. Reagen, H. Lee, D. Brooks, and G. Y. Wei. A multi-chip system optimized for insect-scale flapping-wing robots. In *2015 Symposium on VLSI Circuits (VLSI Circuits)*, pages C152–C153, June 2015.