

A Methodology for Optimized FPGA Design of Signal Processing Kernels

Zachary K. Baker and Viktor K. Prasanna
University of Southern California, Los Angeles, CA USA
zbaker@halcyon.usc.edu, prasanna@ganges.usc.edu

Abstract

*This paper presents a methodology for producing FPGA designs that are optimized for some subset of frequency, latency, precision, FPGA area, or energy. Through design exploration using high-level models and fast design tools such as Handel-C, we develop a better understanding of design trade-offs and move quickly toward an optimized design. Matrix multiplication is presented as an example of this technique. Using a MAC-op per time*area performance optimization metric, a design is described which outperforms a generic parallel design compiled by the Handel-C compiler by 2.8x.*

1 Introduction

FPGAs provide flexible signal and image processing that can adapt to changing needs in the field. However, high-speed, low-power and area designs are difficult to produce. The purpose of this research is to develop an algorithmic approach toward matching application implementations at an algorithmic and behavioral level with a FPGA architecture at the structural level.

Given some performance metrics we provide a methodology to optimize designs to suit those metrics. In this paper we optimize for minimum time and area. Here, area is defined as the number of Xilinx logic slices occupied on the FPGA. A slice, in terms of logic, is composed of two four-bit look-up tables. Time is the latency in clock cycles multiplied by the frequency of the device. Other metrics suitable for use in optimization are the minimization of memory usage, interconnect use and delay, maximizing the amount of parallelism, effective use of embedded resources such as multipliers and RAM.

Developing a high performance data processing architecture can require making many decisions in terms of meeting performance criteria. Latency, frequency, energy, area, and device limitations all require careful analysis and understanding before a design is undertaken. One option, of course, is to implement all possibly optimal designs and do full optimization on each until the needed performance is achieved. We propose that there is a better solution, in which

the design trade-offs are understood at a more sophisticated level than rules of thumb, and more architectures, and more design options within those architecture classes, are explored.

In this paper, we explore the notion of using a combination of high-level performance estimation and design space exploration, quick and dirty implementations of many architectures using high-level design tools such as Celoxica Handel-C, and final optimization using robust description languages such as VHDL. This paper is broken into a methodology section, where our technique is explored fully, an example in which we apply our methodology to the matrix multiplication problem, and a section in which we discuss the strategy we are developing for the formal packaging of the intellectual property developed through the use of this methodology – not just the final design, but all of the intermediate performance models and parameter dependencies and trade-offs as well.

2 Methodology

Our approach to FPGA design can be summarized as follows: first, the problem is defined and possible algorithms and architectures are defined as possible solutions. Next, we develop a set of algorithms that can implement an application and a detailed understanding of the trade-offs in space and time complexity of those algorithms. Given a large set of possible architectures and structural variations on those architectures, the number of possible solutions can be dramatically large.

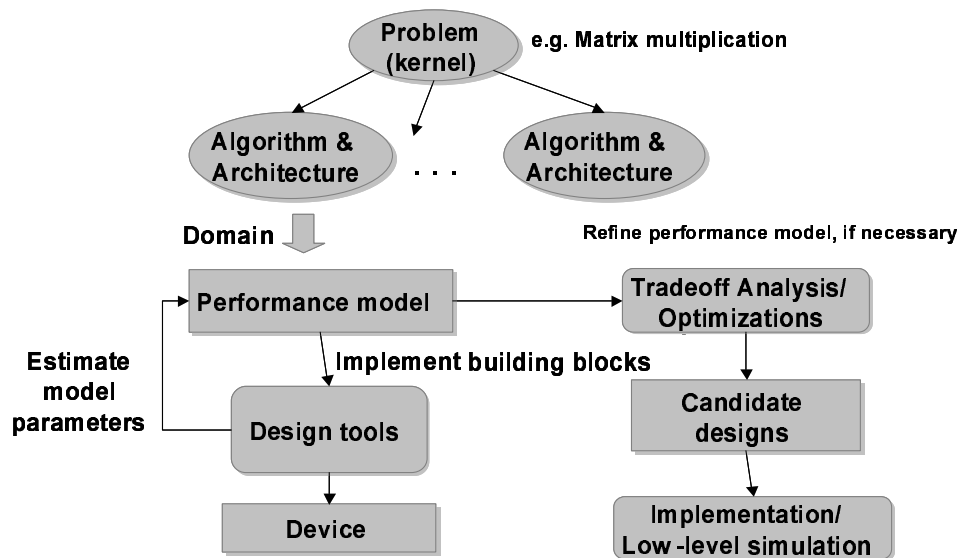


Figure 1: Flowchart for our approach to design optimization)

In a sense, we provide a hybrid top-down and bottom-up approach. That is, we have a set of top-level design goals and algorithms to implement a particular application. Given an application like matrix multiplication, we have a rough understanding regardless of implementation of the memory requirements and number of processing elements that can be effectively parallelized.

However, we also have detailed information on the area and time effects of various hardware implementation decisions, such as the area and speed characteristics for a specific type of memory, the space/time trade-offs for an increasing number of processing elements on an FPGA, or the limited amount of a particular resource on the FPGA.

The first step is design exploration through high-level models. Each architecture class, from linear arrays to iterative signal processing architectures, has certain hardware and control requirements and provides certain well understood performance benefits. After making various choices about architecture, pipeline depth, memory types, implementation algorithm, etc., we have a set of several designs that merit further implementation.

At this point, we turn to a high-level hardware implementation language, Handel-C, by Celoxica for the next stage of our design exploration. Handel-C allows us to produce quick “sketch” implementations of complex applications quickly, in the span of a few hours or days. We are not particularly concerned about the correctness of the implementations, or squeezing performance out of the devices. We are simply interested in getting a high quality estimate of potential performance than could be determined by high-level estimation alone.

After analyzing the field of Handel-C results, one or two architectures with promising preliminary results are chosen for final implementation and optimization in VHDL. While it is feasible to consider continuing with Handel-C and not transferring the designs to a traditional description language such as VHDL, we find that while Handel-C is excellent for describing an architecture quickly it is much more difficult to progress to the performance that would be expected of VHDL without far more effort that would be required otherwise. We find Handel-C to be a difficult language to precisely specify hardware, particularly pipeline structures with efficient data transfer. Due to the system the language uses to convert high-level code lines into state machines, the each of expressing concurrency possible with VHDL is lost.

The set of design options produced after going through this process is valuable for future designs as chip resources and synthesis tools evolve.

In this paper we provide an example of matrix multiply on FPGA. First, we define the problem. Then, we provide an analysis of the area and time trade-offs, discuss the effects of various design “knobs” on the characteristics of the final implementation, and finally provide sample numbers for a baseline, generic implementation using the Celoxica Handel-C compiler, and optimized, pipeline code written in VHDL. Our results clearly show that an intelligent design, optimized for various conditions, will outperform a generic implementation for the design goals.

	Freq	Cycles	Slices	uSec	MAC/(sec * slice)
VHDL optimized for AT	150	256	2083	1.7	1090k
VHDL optimized for energy	150	256	2200	1.7	949k
HandelC: 16 conf. mult (outer loop)	43	1346	1137	31.3	115k
HandelC: 16 conf. mult (inner loop)	53	1058	528	19.9	388k
HandelC: 16 embedded mults (outer)	40	1638	1651	40	60k
HandelC: 16 embedded mults (inner)	55	1637	807	29.8	170k
Xilinx reported results	150	3375	251	22.5	597k
conf HandelC: linear array	100	1359	10k	13.6	
embed HandelC	113	1359	10.2		

Figure 2: Summary of performance results for various configurations of 16x16 multiply on the VirtexII

3 Matrix Multiply

Matrix multiplication is a kernel operation in many signal and image processing applications. Common implementations of matrix multiply proceed according to the basic algorithm, using three loops and a single multiply-accumulate register in the center of the loops. Parallel architectures can be derived by unrolling any of the loops and using more multipliers, but depending on the choice this can add significant hardware and programming complexity in the number of memory ports required, multiplexing hardware, or delay for the final summation of products. We present results using a hand-optimized VHDL and machine-optimized Handel-C code of a different approach, the linear array of multipliers. Our group pioneered the linear array multiplier architecture in [1]. In our implementation of the general architecture, we choose to use $n=16$ multipliers, although the architecture can handle smaller divisors of n . By organizing the architecture so that a minimum of hardware is needed to control the flow of data the linear array can achieve very high clock rates in a minimum of clock ticks.

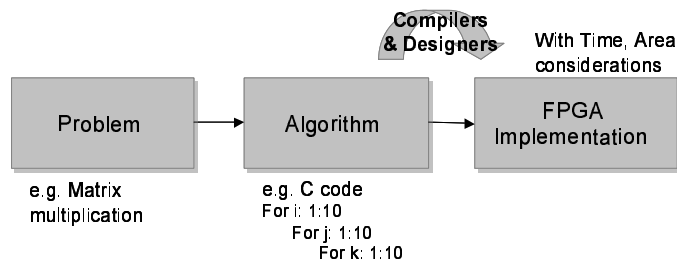


Figure 3: A methodology from algorithm to bitstream)

We optimize the design for the $n^2 = 16 \times 16$ case, and have defined performance to be equal to

$$Perf_{AT} = \frac{n^3}{\#Slices \times Time} \quad (1)$$

By including area in performance, we are able to take into account the effect of increased numbers of processing elements and the area differences for various types of memory. This is especially relevant in an era of deep pipelines and huge caches where small performance improvements are bought at the cost of dramatic increases in area. Also, the operations per area-second performance measure gives us a more interesting design space in which to work.

Figure 4 shows the latency-area tradeoff for matrix multiplication. The data points are estimated based on the number of registered, processing elements, and local memory used for a given implementation. We can choose to sacrifice time performance for area, or vice versa, depending on other design constraints.

Using the Xilinx ISE 4.1i and Mentor Graphics ModelSim 5.5e for the Xilinx Virtex II FPGA, we first executed tests of various memory types and sizes and determined their effects on the area and speed. With knowledge of the characteristics of the Virtex II embedded multipliers, registers, I/O bandwidth, etc., we choose to implement the linear array architecture for matrix multiply. Assuming a continuous stream of matrices, the architecture yields a result each cycle, for 256 cycles to completion. However, this efficiency comes at a price in terms of area— the hand-optimized design is the largest of all the implemented designs due to the added memory and repeated hardware components. We compare our results against various implementations compiled by Celoxica Handel-C. We chose Handel-C as a baseline because it provides a fast method for producing FPGA designs and handles many of the 'details' that we have control over/have to deal with in VHDL. As it is fairly easily to modify Handel-C codes, we provide results for outer and inner loop parallelization, with configured and embedded multipliers.

We also implemented matrix multiplication using the linear array approach in Handel-C. Handel-C provides a remarkable ability to capture a wide range of resolution in many problems. Here, we have implemented both a generic three-loop algorithm type approach without concerning ourselves with timing or any performance enhancements, and we have generated a signal-based implementation that works far more efficiently than the blind approach, but only achieves 50% of the performance of the VHDL implementation, because of an inefficient implementation of the data path. One interesting methodology that we developed during the implementation of these architectures in Handel-C was the use of the automatic optimization features for the datapath. Handel-C provides a library of macros that allow efficient blocking through the use of wait loops and write_ready and read_ready signal lines. The key advance here is that a designer can use the macros as a crutch until the architecture is perfectly synchronized. At this point the optimizing compiler observes that the ready signals are always true and removes the related hardware, leaving system as a fully-optimized synchronous signal-based design.

In Figure 2 we see that our custom VHDL design, with full knowledge of the tradeoffs and performance measures, outperforms the best of the generic

REFERENCES

parallelizations in Handel-C by 3x, and the area-optimized Xilinx design by 1.8x[1, 2]. By trading area for time performance, we produce a significantly faster design. Both Handel-C and Xilinx provide much smaller designs, but with a significantly larger latency for the Xilinx design and a much slower frequency for the Handel-C design.

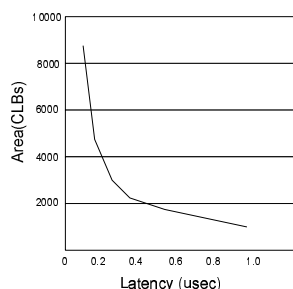


Figure 4: Area-Latency tradeoffs for matrix multiplication (n=12)

We are interested in further defining a formalization of this methodology. We believe the intermediate algorithmic and architectural understanding is of significant value, even after a certain instance has been defined for a particular set of optimization goals. We call the IP produced by this methodology I^2P . I^2P is not Intellectual Property in the conventional sense. Instead, is an *intelligent*, portable collection of algorithms and methodologies for an application, providing the designer with all of the tools necessary to produce an area, time, or energy efficient design, regardless of platform.

Given the parameters of the target hardware platform, I^2P provides an understanding of the design trade-offs for the algorithm on a particular platform, and gives the designer the ability to perform Design Space Exploration without requiring the designer to have a complete understanding of the application. I^2P allows a designer to produce implementations of algorithms specifically tailored for the situation at hand.

In conclusion, this paper presents a methodology for producing FPGA designs that are optimized for some subset of frequency, latency, precision, FPGA area, or energy. Matrix multiplication is presented as an example of this technique.

References

- [1] Ju-Wook Jang, Seonil Choi, and Viktor K. Prasanna, Area and Time Efficient Implementation of Matrix Multiplication on FPGAs, The First IEEE International Conference on Field Programmable Technology (FPT), December 2002.
- [2] Xilinx Application Note, Virtex-II Series and Xilinx ISE 4.1 i Design Environment, <http://www.xilinx.com>, 2001.

REFERENCES

- [3] Seonil Choi, Ronald Scrofano, Viktor K. Prasanna, and Ju-Wook Jang, Energy-Efficient Signal Processing Using FPGAs, Eleventh ACM International Symposium on Field-Programmable Gate Arrays (FPGA 2003), February 2003.
- [4] Celoxica Handel-C Development Environment, <http://www.celoxica.com>, 2003.