

Tool Integration for Signal Processing Architectural Exploration

Ramsey Hourani, Ravi Jenkal, W. Rhett Davis, Winser Alexander
Electrical and Computer Engineering Department
North Carolina State University
Raleigh, NC 27695, USA
(919) 513-2839
{rhouran, rsjenkal, rhett_davis, winser}@ncsu.edu

Abstract—This paper presents a design environment for efficiently generating and analyzing application-specific Intellectual Property (IP) cores for system level signal processing algorithms. We present our view of a framework that integrates existing electronic design automation (EDA) tools to alleviate the designer from manually constructing the hardware models and analyzing their performance. Our framework is well suited for designers with a range of signal processing and hardware expertise. Our framework builds the dedicated IP cores and estimates the performance such as area, critical path delay, and latency within seconds. We estimated area and delay within 5% accuracy when compared to actual synthesized designs. Our framework also compares different hardware designs for various digital signal processing (DSP) algorithms and allows the designer to make final decisions in selecting an optimized IP core. We used a GUI-based framework invoked from MATLAB to build and analyze the hardware designs. The end products are optimized hardware designs described in SystemC and Verilog code. We illustrate the use of our framework via two case studies: finite impulse response (FIR) filters and adaptive channel equalizers.

I. INTRODUCTION

The goal of most emerging System-on-Chip (SoC) designs is to provide an architecture that is specialized for a certain class of applications and can therefore deliver a higher level of throughput at lower power consumption than previously possible. The International Technology Roadmap for Semiconductors (ITRS) calls for design tools and methodologies to support approaches for designing at the system-level [1]. Two important levels of this approach are system-level synthesis and IP reuse. The cost and effort of generating such tools and IPs are so great that they are not available to the general user. As a result, designers continue to create more tools that are specialized for a small user base. The resulting confusion in using these tools increases the design complexity. A solution for mitigating the design process is providing open source IP cores and system-level synthesizers. This paper presents our view of what a framework for creating and analyzing open-source IP cores should look like. We discuss our framework using case studies of signal processing algorithms such as filters and adaptive equalizers.

The possibilities of modeling a functional algorithm at the hardware level are so great that the efforts spent in designing and analyzing the possible architectures significantly reduces designer productivity and increases time-to-market. One of the main obstacles designers face when creating and analyzing

their designs at RTL is an increase in refinement efforts and an increase in synthesis times. To this end, we present our framework for efficiently designing application-specific hardware architectures to model signal processing algorithms. The goal of our framework is to improve hardware architectural exploration by guiding designers through the design space. Designers can use our framework to estimate area, latency, throughout, power dissipation, and computational precision for each architecture within seconds. Our framework produces SystemC [4] and Verilog hardware models that describe the functionality of the algorithm at the hardware level. Designers with limited hardware experience can select application-specific IP cores from a set of optimized designs.

Various methodologies have emerged to synthesize designs described using high level languages such as C^{++} and MATLAB [2] [3]. The idea behind high level synthesis is to allow the synthesis tools to optimize the designs using algorithms that minimize the critical path delays and/or minimize the total design area. This method improves designer productivity that would otherwise require him or her to manually optimize the design. Synthesis tool makers, such as Synopsys, provide IP cores for basic signal processing applications such as finite (FIR) and infinite (IIR) impulse response filters. Such cores are designed for general purpose signal processing applications and therefore may not be optimized for application-specific algorithms.

The rest of this paper is organized as follows: Section II presents a survey of frameworks and techniques for architectural exploration of hardware designs using different levels of design abstraction. Section III introduces our framework for guiding the designer through the architectural design space. Section IV presents two case studies illustrating the use and effectiveness of our framework: FIR filters and adaptive channel equalizers. Section V evaluates the validity of our framework in choosing optimized IP cores for application-specific signal processing algorithms, and section VI presents concluding remarks.

II. RELATED WORK

By nature, hardware designers prefer to make final decisions in selecting optimized hardware architectures for their systems. Hawley *et al.* presented a compiler for generating different hardware architectures for FIR filters targeting application

specific integrated circuits (ASIC) [5]. Their work discussed methods for improving the throughput of a design, such as pipelining, and decreasing the hardware complexity, such as symmetry properties in linear phase FIR filters. Their work also provided a means for comparing the performance of the different filter structures by analyzing their area and delay. These techniques have not made it into mainstream synthesis tools, such as Design Compiler [6]. The development and maintenance cost would, therefore, be borne by the designers to optimize their filter structures rather than the EDA companies.

MATLAB is an attractive tool for signal processing algorithm developers. Including MATLAB in the design flow for generating and analyzing hardware designs bridges the gap between algorithm development and hardware IP core generation. PyGen is an example of a design flow that generates hardware designs for field programmable gate arrays (FPGA) starting from a high level language such as MATLAB and Python [7]. PyGen is an *add-on* tool used to estimate the energy dissipation in FPGA-based designs with errors as low as 3%. Other researchers have provided design methodologies and EDA tools for improving algorithmic refinement to hardware. Examples of this include the work of Banerjee *et al.* They developed tools and integrated verification flows for automatically developing and mapping MATLAB algorithms onto FPGAs and ASICs [2], [3], [8]. Their work founded the company AccelChip [9] which now markets these tools and flows. The hardware architectures generated by these tools are ideal for general-purpose applications. However, designs can be further optimized based on the application. For example, a polyphase FIR filter would be ideal for multi-rate signal processing applications such as discrete wavelet transforms [10], or an interleaved filter structure provides better hardware utilization for multi-signal processing algorithms [11].

Miyaoka *et al.* demonstrated that accurate area and throughput estimators can be developed using synthesizable hardware units modeled at RTL [12]. They were able to estimate area and throughput using mathematical equations that were functions of bit widths and IP core type. Their performance estimations were accurate to within 3% of actual measurements. This method of performance estimation assumes that the designer manually constructed the hardware architectures and, therefore, has detailed knowledge of IP core layout. Designers would like to achieve this level of estimation accuracy applied from a higher level of design environment.

Table I summarizes the main features of the frameworks discussed in this survey and compares it to our proposed framework. The ‘‘Hardware Architecture’’ column refers to whether the generated hardware designs were optimized for specific or general-purpose applications. Application-specific architectures typically perform better than architectures designed for general applications. The ‘‘Designer Expertise’’ column refers to the level of hardware expertise required to design optimized hardware architectures. The details of our framework are presented in the next section.

TABLE I
SURVEY OF FRAMEWORKS

Framework	Metric Estimation	Hardware Architecture	Designer Expertise	Target (ASIC/FPGA)
Hawley [5]	Area/Delay	Specific	High	ASIC
PyGen [7]	Energy	General	Low	FPGA
AccelChip [9]	Area/Delay	General	Low	both
Miyaoka [12]	Area/Delay	General	High	-
This work	Area/Delay	Specific	Hi - Lo	both

III. PERFORMANCE ANALYSIS FRAMEWORK

In this section, we present our framework for integrating common EDA tools used for generating signal processing IP cores and synthesizing the hardware designs for performance evaluation. Our framework provides the user options for selecting optimized application-specific hardware designs. Figure 1 illustrates an overview of our framework. Our GUI-

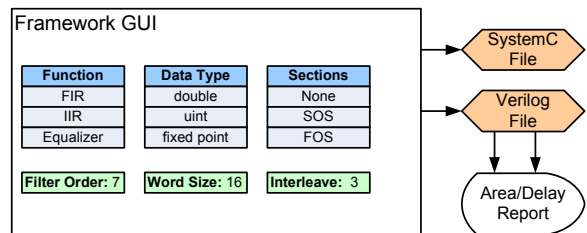


Fig. 1. Framework Overview.

based framework generates SystemC and Verilog IP cores that describe the behavior of the algorithm at the hardware detail. The SystemC models are easily integrated into system level designs using appropriate interfaces and wrappers, while the Verilog models are used to accurately estimate pertinent hardware performance metrics, such as area, delay, power dissipation, and computational accuracy within seconds. We invoke our framework from MATLAB using a GUI. Our framework comes equipped with design options for improving both algorithmic functionality and hardware performance. Examples of parameters we include in our framework are pipelining for parallel implementations, interleaving for multi-signal processing, multi-rate data processing techniques, and second-order-sectioning (SOS) for improved computational accuracy. A screen snapshot of our GUI is shown in Figure 10.

We developed our framework to accommodate designers with various levels of signal processing and hardware expertise. Our framework automatically builds and analyzes different architectures to accommodate designers with limited hardware experience. In such cases, our framework reports the performance of all designs for a particular algorithm. We provide the framework user with bar charts and area/delay curves that the designer uses for selecting an optimized architecture. We accomplish these tasks by invoking a set of EDA tools typically used for measuring the performance of hardware designs. The designer makes final architectural decisions when selecting an optimized hardware design. The

process of selecting an application-specific and optimized hardware design typically executes in the order of minutes using our framework. This significantly reduces design efforts that would typically require hours or even days if this process were performed manually.

A. Framework Design and Analysis Process

Figure 2 illustrates the process we used to accurately and efficiently explore different architectures for signal processing algorithms. The user invokes our GUI-based framework from

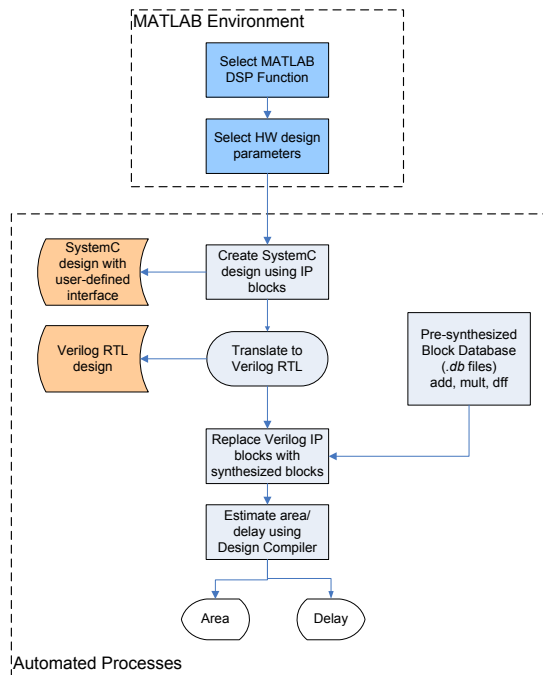


Fig. 2. Process for estimating hardware performance metrics.

MATLAB. He or she then selects the desired algorithm to analyze. Examples of signal processing algorithms we included in our framework to date are filters, adaptive equalizers, and discrete wavelet transforms for speech and image processing. The user then selects system level design parameters, such as filter order and finite precision word sizes. Our framework automatically builds the SystemC models for each architecture. The variations in the architectures depend on the pipelining techniques used to reduce critical path delays. The SystemC models allow the designer to integrate the hardware architectures into their functional C^{++} or MATLAB algorithms for functional verification. We used a SystemC to Verilog RTL translator to automatically translate the SystemC models to Verilog [13]. The translator is capable of translating SystemC procedural code and hierarchical modules programmed using synthesizable SystemC semantics [14]. This saves the designer time that would otherwise be spent manually converting the SystemC code to Verilog. We verified the designs we converted from SystemC to Verilog in terms of functionality, area, and critical path delay. In all cases, the SystemC designs matched the Verilog designs. We verified the functionality of our designs using the SystemC models. Simulations performed using the SystemC models were approximately five times

faster than simulations performed using the Verilog models for our designs [15].

We generated our hardware architectures using IP-based designs. We used basic blocks such as adders, subtractors, multipliers, and delay units to construct the hardware designs. We characterized the performance of each IP block by synthesizing the block using Synopsys Design Compiler [6]. We used a $0.18\mu m$ standard cell library for synthesizing and characterizing all of our hardware designs presented in this paper. We replaced each block with the synthesized cores and estimated the area and delay metrics. We automated this process by using a scripting system called SSHAFT (System to Silicon Hierarchical Flow Tool) that invoked the Synopsys synthesis tool to automatically execute a set of commands that result in area and timing reports [16] [17].

IV. CASE STUDIES AND EXPERIMENTAL RESULTS

We used two case studies of common signal processing algorithms to illustrate the effectiveness of our framework: FIR filters and adaptive channel equalizers. Our goal was to use our framework to automatically generate and analyze different hardware architectures for both algorithms. We used our framework to display the area and delay results for each design, allowing the user to compare the performance of each architecture and, therefore, make final architectural decisions.

A. Finite Impulse Response (FIR) Filters

FIR filters are typically used in signal processing applications such as image and speech processing for communication systems. The basic architectures for FIR filters include Direct Form I, Direct Form II, and Transpose Form [11]. We used our framework to generate 8 architectures for the FIR filter. The first 4 architectures were pipelined variations of the Direct Form I structure and the second 4 architectures were pipelined variations of the Transpose Form. Figure 3 shows the different computational cells that were generated using our framework. Figure 4 compares the estimated metrics to the actual synthesized designs for different architectures of a 16-bit, 32-tap FIR filter.

Figure 5 shows an area/delay curve for comparing the performances of the different filter designs. The total time required for generating the filter designs and estimating the performances was approximately three minutes (22 seconds per design). Whereas, synthesizing all the designs to extract the actual area and delay measurements required 90 minutes (11 minutes per design).

The results of Figure 5 show that the area increased as we optimized the design for critical path delay, which is what one would expect. However, when we compared architectures 1 and 5, we saw that there was a 200% increase in throughput with architecture 5 for a 6% increase in area. Therefore, architecture 5 would be an appropriate filter design, optimized for both area and delay. We used our framework to make conclusions such as this within minutes versus hours.

We compared the number of transistors for the filters generated using our framework to the filter structures generated using Hawley's FIR compiler [5]. We also compared

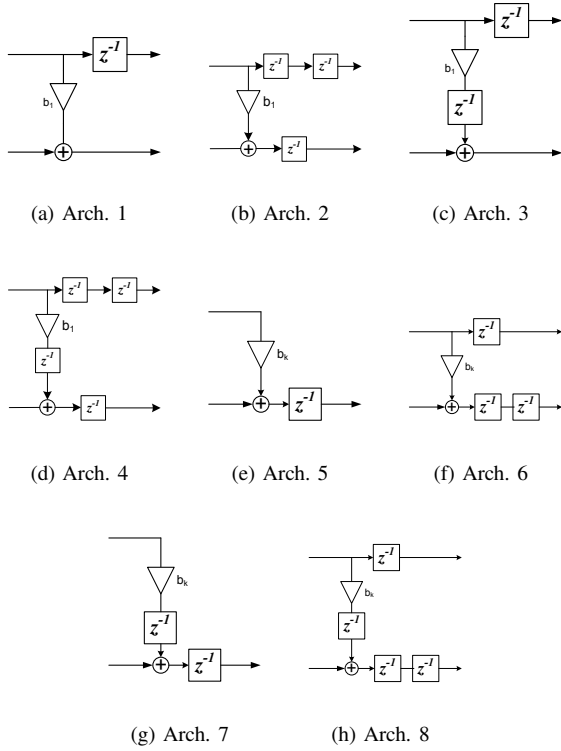


Fig. 3. Computational cells for FIR filter architectures

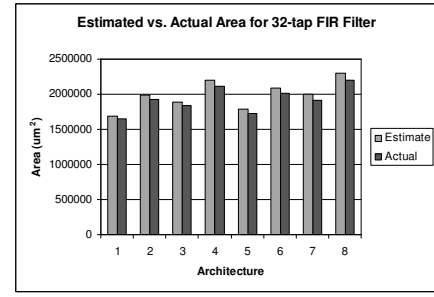
the FIR filters generated via our framework to a behavioral model FIR filter described using Verilog and a DesignWare Transpose Form FIR filter IP provided by Synopsys [18]. The performance of the behavioral FIR filter matched our non-pipelined Direct Form I structure (architecture 1), whereas the performance of the DesignWare IP filter matched our non-pipelined Transpose Form structure (architecture 5). The results are summarized in Table II.

TABLE II
FILTER PERFORMANCE COMPARISON (N^o OF TRANSISTORS)

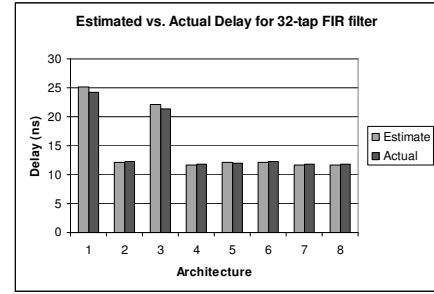
Filter Structure	FIR Compiler [5]	Behavioral Verilog	Synopsys DesignWare [18]	Our Framework
Cascade Form	14,000	11,800	15,000	12,000
Polyphase	46,000	36,000	-	38,000
Linear Phase	66,000	55,000	-	58,000

B. Adaptive Channel Equalizer

We investigated how our framework analyzed the performance for other algorithms beyond filters. We used the case study of a 12-tap adaptive channel equalizer [19] [20]. Our framework automatically generated and estimated the performance of different hardware designs for the adaptive equalizer. Our framework modeled the equalizer architectures using three blocks: FIR filter, comparator, and a coefficient-update block. We used the computational cells shown in Figure 3 (a)-(d) for generating the FIR filter, two different architectures for the comparator (Figure 6), and two different computational



(a) Area



(b) Delay

Fig. 4. Estimated vs. actual metrics for 16-bit, 32-tap FIR filter designs

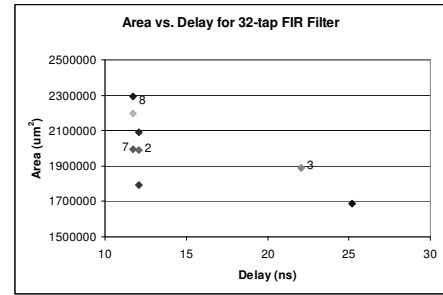


Fig. 5. Area vs. Delay for FIR filter.

cells for the coefficient-update block (Figure 7). Therefore, our framework generated 16 different hardware designs for the adaptive equalizer. The design of the computational cells are explained in greater detail in [15].

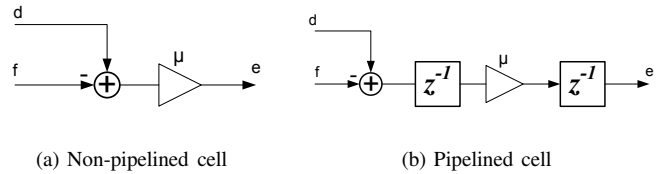


Fig. 6. Comparator architectures

Figure 8 illustrates the architectural rankings for the different adaptive equalizer designs using the estimated metrics and compares it to the actual results. The estimations were

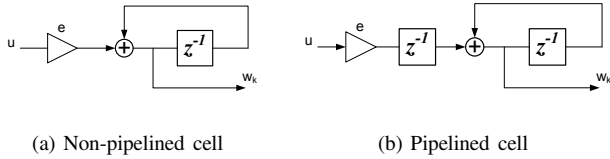
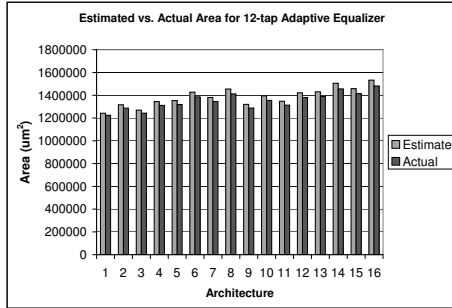
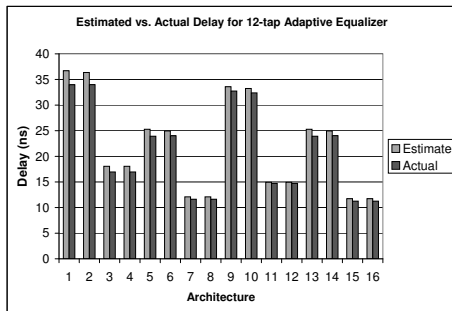


Fig. 7. Coefficient-update architectures

within a 5% accuracy of actual measurements. We used an



(a) Area



(b) Delay

Fig. 8. Estimated vs. actual metrics for adaptive equalizer designs

area/delay curve once more, shown in Figure 9, to analyze the performance of the different adaptive equalizer architectures. Our framework reported area and delay performances back

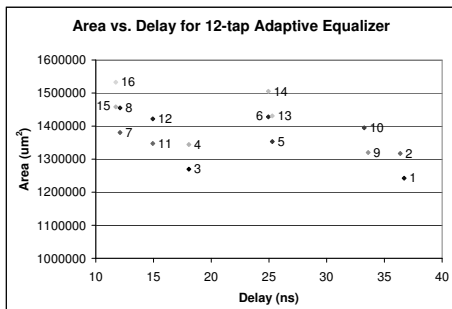


Fig. 9. Area vs. Delay for adaptive equalizer.

to the designer. The designer would then use his or her expertise in making final architectural decisions. For example, architecture 1 was optimized for area, while architecture 15

was optimized for critical path delay. However, architecture 3 was optimized for both area and critical path delay. The user remains in control of final architectural decisions, while alleviating him or her from manually designing and analyzing each architecture. The total time required for generating and estimating the performance of all 16 equalizer designs was approximately five minutes (20 seconds per design). Whereas, synthesizing all the designs to extract the actual area and delay measurements required around 112 minutes (7 minutes per design). We did not take into account the time required to manually design each equalizer architecture at RTL.

V. FRAMEWORK EVALUATION

The results of Figures 4 and 8 illustrate that the architectural rankings using the estimated performance metrics were similar to the rankings using the actual measurements. We used our framework to estimate area and delay metrics within 5% accuracy of actual measurements for the designs considered in this paper. We automated much of the design and analysis steps to promote consistencies in architectural ranking.

The SystemC models allowed us to integrate our hardware architectures into C^{++} functional models. We found that interfacing our SystemC models with our functional models was seamless through the use of wrappers and interfaces defined in both C^{++} and SystemC. We used scripting to automatically construct each hardware design considered in our framework. We also used SSHAFT to invoke a sequence of synthesis commands for estimating area and delay metrics. This process provided an accurate estimate for area and delay in a fraction of the actual synthesis times. The savings in design and analysis efforts allowed us to further explore the architectural design space or fine-tune current hardware architectures to meet stringent design constraints.

We intended for our framework to be used as a means of guiding the designer through the architectural design space. We developed our framework such that it integrated EDA tools typically used for modeling, designing, and synthesizing hardware designs. Our goal was to efficiently and accurately provide the designer with pertinent hardware performance metrics for a wide range of architectures. Our framework supplied the user with bar charts for area and delay metrics and area/delay curves for comparing the performances of different hardware designs. This permitted the user to make final architectural decisions using the charts and graphs. Therefore, the user makes lower level decisions earlier in the design process and within minutes using our framework.

VI. CONCLUSION

This paper presented a framework for accurately and efficiently exploring architectural variations for the implementation of signal processing algorithms. Our framework generated hardware IP cores described using both SystemC and Verilog. The SystemC models allowed us to seamlessly integrate the hardware architectures into our functional C^{++} and MATLAB models. Whereas the Verilog models were used to extract pertinent hardware performance metrics such as area and critical path delay. We used a performance estimation technique where

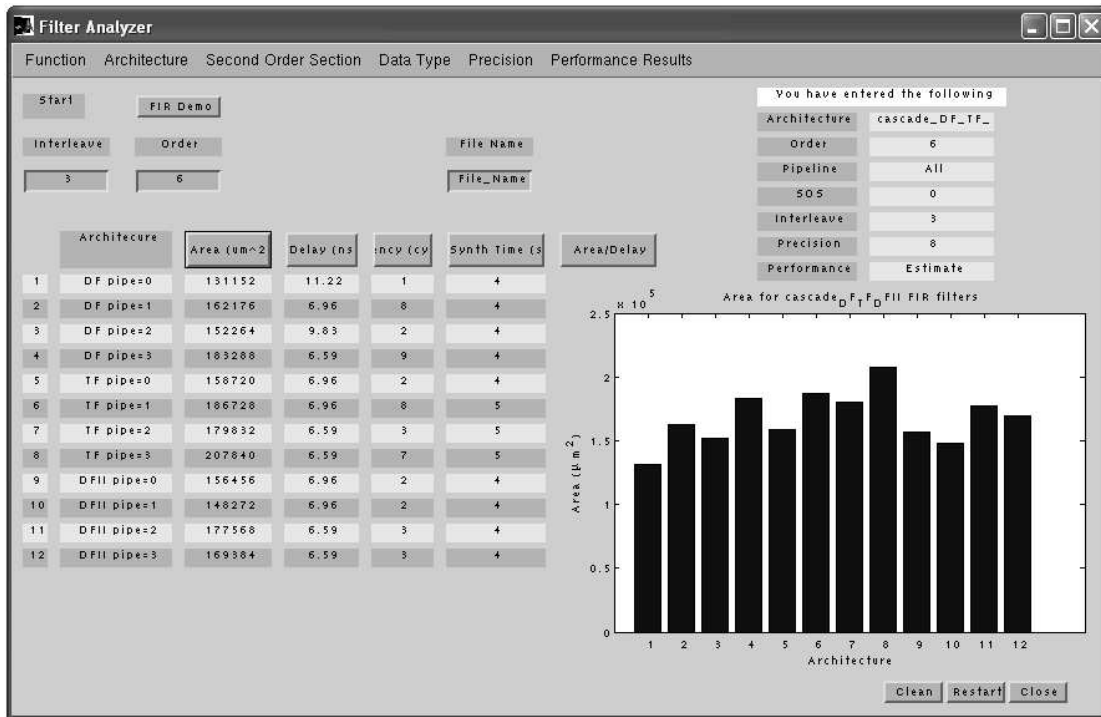


Fig. 10. Snapshot of our vision for a MATLAB-based GUI framework.

we characterized the performance of lower level IP blocks such as adders, multipliers, and delay units. This process allowed us to estimate the overall area and delay of our IP-based hardware designs within seconds. Our estimations were within 5% accuracy of actual area and delay measurements. We illustrated the utilization of our framework using two case studies: FIR filters and adaptive channel equalizers.

Further efforts will be spent towards including a wider range of IP cores typically used for other signal processing applications. By doing so, our framework promotes design reuse and refinement consistency for architectural exploration of system level algorithms. We are currently investigating similar techniques for estimating power dissipation for our IP cores. Additionally, we are considering several techniques for analytically and empirically estimating the computational precision of the different architectures. This will allow us to further reduce analysis times and increase architectural explorations.

ACKNOWLEDGEMENTS

The work of Ramsey Hourani was supported by the ONR/HBEC Future Engineering Faculty Fellowship Program.

REFERENCES

[1] The International Technology Roadmap for Semiconductors, available at <http://public.itrs.net>

[2] P. Banerjee, et al., "Overview of a compiler for synthesizing MATLAB programs onto FPGAs", *IEEE Transactions on VLSI Systems*, vol. 12, pp. 312-324, March 2004

[3] M. Haldar et al., "Automated synthesis of pipelined designs on FPGAs for signal and image processing applications described in MATLAB(R)", *Design Automation Conference*, pp. 645-648, Feb. 2001

[4] OSCI, "SystemC Version 2.0 User's Guide", Jan. 2002. Available at <http://www.systemc.org>

[5] R. A. Hawley, B. C. Wong, Thu-Ji Lin; J. Laskowski, H. Samueli, "Design techniques for silicon compiler implementations of high-speed FIR digital filters", *IEEE Journal of Solid-State Circuits*, vol. 31, no.5, pp. 656-667, May 1996

[6] Synopsys Design Compiler, Available online at http://www.synopsys.com/products/logic/design_compiler.html

[7] Jingzhao Ou, V. K. Prasanna, "PyGen: a MATLAB/Simulink based tool for synthesizing parameterized and energy efficient designs using FPGAs", *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 47-56, April 2004

[8] P. Banerjee, "An overview of a compiler for mapping MATLAB programs onto FPGAs", *Design Automation Conference*, pp. 477-482, Jan. 2003

[9] AccelChip. Available online at <http://www.accelchip.com/>

[10] P. P. Vaidyanathan, "Multirate Systems and Filter Banks", Prentice Hall, Englewoods Cliffs, NJ, 07632

[11] P. Pirsch, "Architectures for Digital Signal Processing", John Wiley & Sons, 1998, New York, NY, 10158

[12] Y. Miyaoka et al., "Area/delay estimation for digital signal processor cores", *Proceedings of the ASP-DAC 2001*, pp 156-161, Jan. 2001

[13] OpenSoCDesign. Available online at <http://www.opensocdesign.com/>

[14] Synopsys. "Describing Synthesizable RTL in SystemC", Jan. 2002. Available online at <http://www.synopsys.com>

[15] R. Hourani, W. E. Alexander, T. Raithatha, "A Hardware Performance Analysis Framework for Architectural Exploration of DSP Systems", *In Global Signal Processing Expo (GSPx)*, Oct. 2005

[16] W. R. Davis, "Getting high-performance silicon from system-level design", *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pp. 238-243, Feb. 2003

[17] MUSE, "Methodologies for User-friendly System-on-a-chip Experimentation", Available online at <http://www.ece.ncsu.edu/muse/sshaft/>

[18] Synopsys, "DesignWare Building Block IP, High-Speed Digital FIR Filter", Available online at <http://www.synopsys.com/>, Sep. 2005

[19] B. Farhang-Boroujeny, "Adaptive Filter: Theory and Applications", John Wiley & Sons, 1998, New York, NY, 10158

[20] N. R. Shanbhag and K. K. Parhi, "Pipelined adaptive DFE architectures using relaxed look-ahead", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 43, pp. 1368-1385, June 1995