In Proceedings of the IEEE Asilomar Conference on Signals, Systems, and Computers, Pacific Grove, California, Nov., 2010.

Design and Implementation of Real-time Signal Processing Applications on Heterogeneous Multiprocessor Arrays

Hsiang-Huang Wu[1], Chung-Ching Shen[1], Shuvra S. Bhattacharyya[1], Katherine Compton[2], Michael Schulte[3], Marilyn Wolf[4], and Tong Zhang[5]

[1]University of Maryland, College Park, MD, USA, {hhwu, ccshen, ssb}@umd.edu
[2]University of Wisconsin, Madison, WI, USA, kati@engr.wisc.edu
[3]AMD, Inc. Research and Advanced Development Labs, Austin, TX, USA
University of Wisconsin, Madison, Wis., USA, schulte@engr.wisc.edu
[4]Georgia Institute of Technology, Atlanta, GA, USA, wolf@ece.gatech.edu
[5]Rensselaer Polytechnic Institute, Troy, NY, USA, tzhang@ecse.rpi.edu

Abstract

Processing structures based on arrays of computational elements form an important class of architectures, which includes field programmable gate arrays (FPGAs), systolic arrays, and various forms of multicore processors. A wide variety of design methods and tools have been targeted to regular processing arrays involving homogeneous processing elements. In this paper, we introduce the concept of field programmable X arrays *(*FPXAs*) as an abstract model for design and implementation of heterogeneous multiprocessor arrays for signal processing systems. FPXAs are abstract structures that can be targeted for implementation on application-specific integrated circuits, FPGAs, or other kinds of reconfigurable processors. FPXAs can also be mapped onto multicore processors for flexible emulation. We discuss the use of dataflow models as an integrated application representation and intermediate representation for efficient specification and mapping of signal processing systems on FPXAs. We demonstrate our proposed models and techniques with a case study involving the embedding of an application-specific FPXA system on an off-the-shelf FPGA device.*

1 Introduction

FPGA technology is widely used in the implementation of signal processing systems. This technology has evolved rapidly to incorporate hard and soft processor cores, embedded memory structures, and specialized hardware subsystems [13]. A variety of useful design tools and mapping techniques have been developed for mapping signal processing applications into efficient FPGA implementations (e.g., see [11, 8]).

However, the application and exploitation of heterogeneous hardware structures in the context of FPGA system design remains largely ad hoc, and the usability of commercial tools in this area is limited, with a lack of rigorous integration between high level abstract modeling, and back-end simulation and synthesis.

In [12], a design methodology called dynamic hardware/software partitioning is proposed for a heterogeneous system that is composed of an FPGA and a microprocessor. In this approach, the FPGA is employed as a coprocessor to speed up computationally intensive loops, and the configurable logic fabric in the FPGA is used to support combinational circuits. In [9, 4], another kind of heterogeneous platform is targeted. This type of platform consists of a processor, bit-level reconfigurable part (FPGA), and word-level reconfigurable part, called *Field Programmable Function Array* (*FPFA*) *tiles*. FPFAs are build from FPFA tiles to accelerate intensive computations such as linear interpolation or fast Fourier transforms. Each FPFA tile is composed of ALUs and lookup tables.

We are concerned in this paper with a class of heterogeneous multiprocessor architectures, which we refer to as *field programmable X arrays*, (*FPXAs*). Like FPGAs, FPXAs can be viewed as arrays of configurable processing elements; however, in FPXAs the elements themselves can be arbitrary (typically coarse-grain) structures. Thus, the X in "FPXA" represents an arbitrary, possibly heterogeneous, collection of processing elements and memory subsystems, which are selected during the definition of a given FPXA architecture

2 FPXA Architecture

As discussed, an FPXA architecture is a collection of (possibly heterogeneous) processing elements (PEs), arranged in a grid and interconnected with a hardware-programmable routing network to provide post-fabrication flexibility. FPXAs will use the Globally Asynchronous, Locally Synchronous (GALS) model for design simplicity; inter-clock synchronization occurs only where a PE interfaces with the routing network. The communication network itself is a configurable

grid of horizontal and vertical circuit-switched (i.e., FPGA-like [1, 7]) routing; the amount of routing is implementation-dependent, guided by *structure vectors*, which are discussed below. Unlike FPGAs, data is routed as complete words on bundles of wires rather than individual bits on individual wires.

PEs in an FPXA communicate with the routing network through first-in-first-out (FIFO) buffers, to match the dataflow design paradigm. Each FIFO supports dual-clocks and provides synchronization to match the GALS design paradigm. An individual PE may have several FIFOs, some used for inputs, others for outputs. The "outside" of the FIFOs will connect to the wire bundles of the routing network through programmable multiplexers (for PE input FIFOs) or demultiplexers (for PE output FIFOs). Each wire bundle has an associated set of handshaking signals to indicate data availability and optionally, to provide backpressure for explicit buffer overflow prevention. For subsystems in which dataflow behavior is predictable, we will apply techniques for compile-time buffer analysis and self-timed, synchronization optimization to minimize the use of point-to-point backpressure synchronization [5, 10].

An example of an FPXA in shown Figure 1(a). This example illustrates that FPXAs can have non-uniform spacing between successive rows and columns (e.g., to accommodate heterogeneous-sized processing structures across different parts of the array).

Figure 1(b) shows an example of an FPXA architecture with hierarchical structures for cells and routing channels. This allows for FPXA-based design and analysis techniques to be incorporated within individual FPXA cells (i.e., to provide similarly structured interconnections of smaller scale processing resources).

To define a two-dimensional FPXA architecture precisely, four *structure vectors* are defined to compactly represent the architecture of an FPXA: X_{PE}, Y_{PE}, $X_{channel}$ and $Y_{channel}$. The vectors X_{PE} and Y_{PE} define the sizes (i.e., lengths and widths) of the PEs and define the number of PEs for the target FPXA architecture. The vectors $X_{channel}$ and $Y_{channel}$ define the capacities of the routing channels and indicate the numbers of switches associated with connections between PEs and the routing network. The X and Y structure vectors correspond to the horizontal and vertical axes, respectively, for the layout of PEs. Vector indices within structure vectors are used to identify specific rows and columns of the associated FPXA. For example, assuming the capacity of all channels is uniform at one unit, a possible structure vector representation for the FPXA architecture shown in Figure 1(a) is given below.

$$\begin{aligned} X_{PE} &= (1, 1, 2.5, 1, 1, 1.5, 2) \\ Y_{PE} &= (1, 1, 1.5, 1, 1, 1, 2, 1) \\ X_{channel} &= (1, 1, 1, 1, 1, 1) \\ Y_{channel} &= (1, 1, 1, 1, 1, 1, 1) \end{aligned}$$

In this example, the length and width of the PE located at coordinates $(3, 7)$ can be retrieved from $X_{PE}[3]$ and $Y_{PE}[7]$, and the size of the PE can be derived as $(2.5 \times 2 = 5)$.

We define the norm of the structure vector, $\|X_{PE}\|$, as the number of elements in the vector. Using this notation, the total number of the PEs in our example can be represented as

$$\|X_{PE}\| \times \|Y_{PE}\| = 7 \times 8 = 56. \quad (1)$$

Furthermore, the total size of all PEs can be derived as the sum of the elements in the matrix that is generated by the cross product $({X_{PE}}^T \times Y_{PE})$. This can be expressed as $sum({X_{PE}}^T \times Y_{PE})$, where the sum operator here represents the sum of all elements in a given matrix.

The $X_{channel}$ and $Y_{channel}$ vectors can be used in similar ways to extract information about the routing channels. For example, the size of the switch that connects the four PEs located at $(1, 6)$, $(1, 7)$, $(2, 6)$, and $(2, 7)$ is defined by $X_{channel}[6]$ and $Y_{channel}[1]$, and the total number of switches in the FPXA is

$$\|X_{channel}\| \times \|Y_{channel}\| = 6 \times 7 = 42. \quad (2)$$

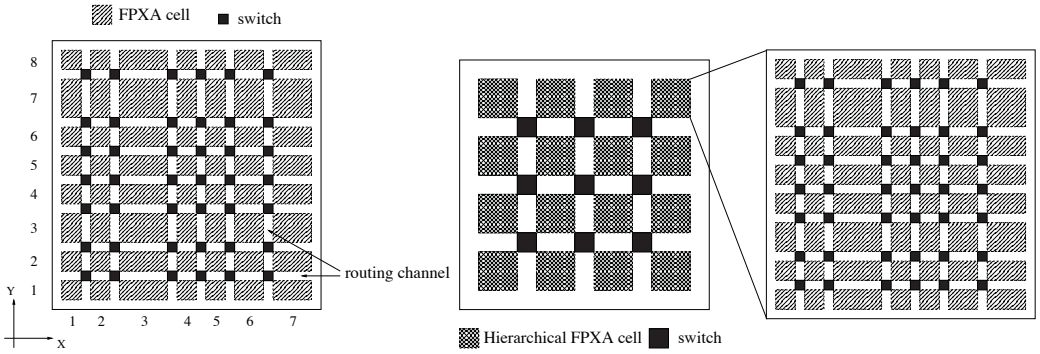
The total size of all switches can be computed as $sum({X_{channel}}^T \times Y_{channel})$. To determine the sizes of the routing channels, we compute the sizes of the vertical and horizontal routing channels separately. For the vertical routing channels, this size can be computed as $sum(X_{channel}) \times (sum(Y_{PE}) + sum(Y_{channel}))$; for the horizontal routing channels, the size can be computed similarly as $sum(Y_{channel}) \times (sum(X_{PE}) + sum(X_{channel}))$.

If structure vectors are determined beforehand, then the relative placement of PEs within an FPXA is determined based on these vectors. The mapping of PEs to actual component types (processor cores or memory subsystems) and the specific types of routing components that are used are then constrained by the dimensions of the associated coarse grain "cells" within the FPXA. Based on the concept of cell-based design flow, specific components are selected and bound to FPXA cells from pre-defined libraries of FPXA PEs and routing modules.

On the other hand, an FPXA designer or synthesis flow may first determine the desired collection of processing, storage, and routing components and their placement within the targeted FPXA. The structure vectors are then be determined by this placement, and used to help annotate application mapping and other follow-on design processes.

Figure 2 illustrates an overall design flow for FPXA-based design. The dashed line indicates considerations that need to be take into account jointly, and the feedback arrows indicate iterative refinement of the FPXA architecture. To support such a design flow and exploit the structure of FPXA designs, we describe in Section 3 an approach to FPXA modeling that can be used as a basis for functional simulation, performance, analysis, and modeling of alternative application-architecture mappings.

In this paper, we develop FPXA concepts in the context of two-dimensional integrated circuit technology. However, FPXA concepts can be extended readily to three-dimensional



(a) Illustration of an FPXA Architecture. (b) Hierarchical FPXA Architecture.

Figure 1. Possible FPXA architectures.

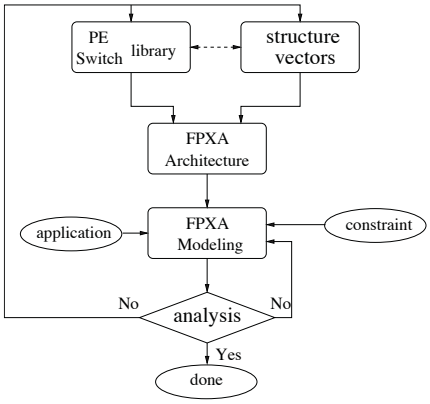


Figure 2. A design flow for FPXA development.

design. For implementation in 3D technology, we can extend the structure vectors with two additional vectors — Z_{PE} and $Z_{channel}$ — to capture the added spatial dimension. Exploration of FPXA architectures and design techniques for 3D technology is an interesting direction for further investigation.

3 FPXA Modeling

In this section, we develop a model called the *FPXA graph* to represent the mapping of an application onto an FPXA architecture. The original application model is assumed to be based on dataflow semantics, which are widely used in the design and implementation of signal processing systems (e.g., see [2]). The FPXA graph is also formulated in terms of dataflow principles, and thus the FPXA graph provides a formal representation that captures relevant characteristics of application-architecture interactions in a candidate FPXA implementation.

In this discussion, we refer to the dataflow representation of the signal processing application to be implemented as the *application graph*. Examples of a simple application graph and an FPXA architecture, along with associated FPXA graphs are shown in Figure 3(a). A wide variety of efficient techniques for analyzing and optimizing signal processing dataflow graphs have been developed, and the FPXA graph provides a framework by which some of these techniques can be adapted for FPXA implementation.

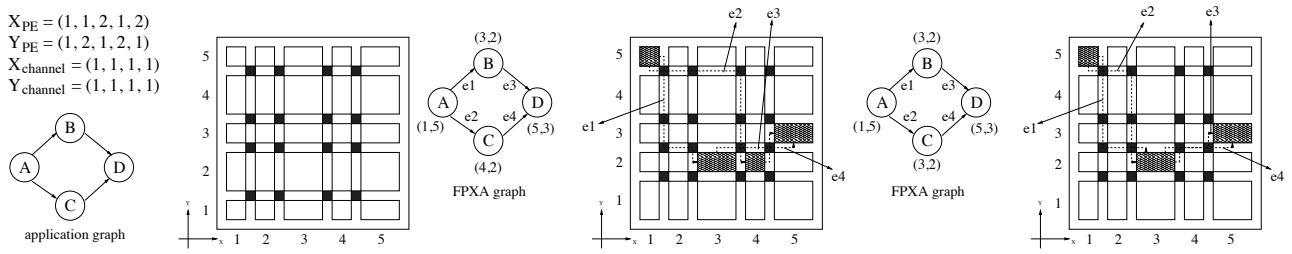
The vertices (actors) in an FPXA graph correspond to actors in the corresponding application graph. When constructing an FPXA graph, it is assumed that each actor from the application graph has been mapped to a specific PE in the target FPXA. The location of the corresponding PE is annotated along with each actor in the FPXA graph, as shown in Figure 3(b), and Figure 3(c). In addition, for each pair of application graph actors that is connected by an edge in the application graph, communication costs can be annotated on corresponding FPXA graph edges. Similarly, attributes such as timing and power consumption characteristics can be annotated on FPXA actors to link such actors to relevant dataflow graph analyses.

The FPXA graph allows more than one actor to be mapped onto the same PE if the target PE has sufficient resources to accommodate all of those actors. For example, Figure 3(c) illustrates an FPXA graph where actors B and C are mapped to the same PE.

4 Experiments

We demonstrate a prototype of an FPXA architecture instance on an FPGA (Xilinx Virtex-5 XC5VLX330FF1760-1) with the Viterbi decoder application. We use the Xilinx PlanAhead [14] tool and the associated *Pblock* feature as central part of this prototyping process. A Pblock in the Xilinx PlanAhead tool can be viewed as a design construct that allows one to partition an FPGA design into "smaller, more manageable physical blocks."

In our design context, each Pblock is configured to be empty or to accommodate a single PE. Each PE in turn is implemented as a collection of configurable logic blocks (CLBs). For simplicity, we normalize the length and width of the FPGA such that $sum(X_{PE})$ and $sum(Y_{PE})$ are both equal to 1. Also, we fix $X_{channel}$ and $Y_{channel}$ since the channel capacities and switches in targeted FPGA are fixed already. The structure vectors are defined as



(a) A simple application graph and an FPXA architecture. (b) An FPXA graph of an implementation. (c) Another FPXA graph for the same application.

Figure 3. Examples of FPXA modeling.

$$X_{PE} = (\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}), \text{ and} \quad (3)$$

$$Y_{PE} = (\frac{1}{12}, \frac{2}{12}, \frac{4}{12}, \frac{5}{12}). \quad (4)$$

To demonstrate the use of this FPXA substrate, we map a Viterbi decoder to the architecture. The application is modeled using VSDF (Synchronous Data Flow for VLSI) semantics [6]. The application model is illustrated in Figure 4, where actor bm computes and normalizes the branch metric; actor b_acs chooses the survivor path; and actor reg_ex stores the outcome.

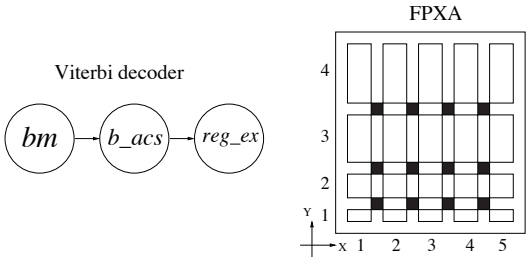


Figure 4. Viterbi decoder and targeted FPXA architecture.

We consider two constraints when mapping the Viterbi decoder onto the targeted FPXA architecture: 100% PE utilization, and a timing constraint of 100 MHz. We consider three FPXA floorplans, as shown in Table 1. Here, each floorplan specifies the location of actors bm, b_acs, and reg_ex in the format (x, y), which is in terms of the structure vector notation introduced in Section 2.

From the results of synthesis, we can quickly check whether or not the PE utilization constraint is satisfied. For floorplan *fp1*, the LUT, FD_LD, SLICEL and SLICEM utilizations of actor reg_ex are 154%, 151%, 184% and 184%, respectively. Following the design flow shown in Figure 2, we try *fp2* due to the violation of the utilization constraint in *fp1*, and in deriving *fp2* we map actor reg_ex to a lager PE.

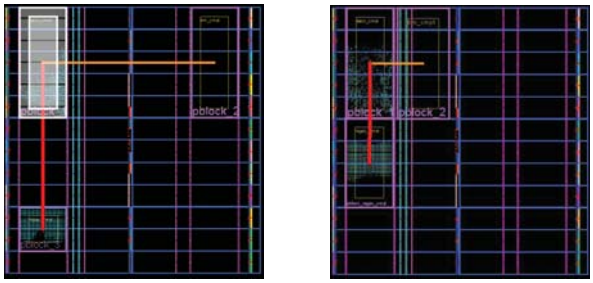
However, after synthesis, we find that for *fp2*, the timing of the critical path is 10.088 ns, which violates the 100 MHz (10 ns period) timing constraint. In an effort to satisfy both of the given constraints, we revise the mapping to the one represented by *fp3*.

The synthesis result for this third mapping shows that the timing for the critical path is 9.05 ns, and all utilization levels are below 100%. The result thus confirms that our FPXA architecture is capable of accommodating the Viterbi decoder application under the given constraints, and provides a specific FPXA configuration that achieves the constraint.

The physical views of the candidate floorplans *fp2* and *fp3* are illustrated in Figure 5.

Table 1. Analysis of three alternative FPXA floorplans for the Viterbi decoder.

	bm	b_acs	reg_ex	utilization	timing (100 Mhz)
fp1	(5,4)	(1,4)	(1,1)	fail	N/A
fp2	(5,4)	(1,4)	(1,2)	pass	fail
fp3	(2,4)	(1,4)	(1,3)	pass	pass



(a) FPXA floorplan *fp2*. (b) FPXA floorplan *fp3*.

Figure 5. Physical views of floorplans.

5 FPXA Design Methods

In this section, we discuss important directions for further investigation in applying the FPXA abstraction to develop novel system-level design tools. These directions include memory system synthesis, dynamic resource binding, and mapping high level dataflow graphs onto FPXA platforms.

The memory system is often the most limiting performance constraint for embedded computing systems. Insufficient attention has been paid in the literature to the synthesis of custom memory systems. Building a memory system out of a library of memory components as in an FPXA is harder than in the custom SoC case where memory components can be designed to more precise specifications. Memory system synthesis builds a memory subsystem from a collection of memory and network components. Given the impact of memory system behavior on performance, a useful direction for further work is the exploration of FPXA synthesis methodologies that perform memory system allocation before scheduling, followed by iterative improvement over the allocation/scheduling procedure.

Dynamic resource binding based on the FPXA abstraction is another interesting direction for future investigation. When it is not possible to use true dynamic retargeting of code to the heterogeneous FPXA components, this behavior can be approximated using static partitioning and compiling, but dynamic binding of computation to resource [3]. We can statically partition applications, and identify code that is well-suited to the different resource types. Ideally, this partitioning will identify code segments that are suitable for implementation each on multiple FPXA resource types. Each section is then compiled (at design time) for each resource on which it may execute. At runtime, an individual system will dynamically choose how to implement the computations, based on resource availability, resource demand, performance requirements, energy concerns, or other criteria. This method also allows applications to be platform-agnostic, provided that each FPXA on which it will execute contains a resource type able to execute at least one version of each computation.

A third general area for investigation is the mapping of high level dataflow representations onto FPXA platforms, using intermediate representations such as the FPXA graph defined in Section 3. As discussed in Section 3, dataflow graphs are used extensively in the design and implementation of signal processing systems, and a wide variety of techniques have been developed for mapping signal processing dataflow graphs onto various kinds of platforms (e.g., see [2]). However, the heterogeneity, coarse grain processing element structure, and self-time routing network characteristics of FPXAs present novel challenges and opportunities for dataflow graph mapping techniques.

6 Conclusions

In this paper, we have developed an abstract model called *field programmable X arrays* (*FPXAs*) for design and implementation of signal processing applications on heterogeneous platforms. The FPXA concept defines a broad class of architectures, where heterogeneous collections of processing resources and self-timed communication channels can be configured and utilized based on the given set of targeted applications, and implementation constraints. We have also introduced structure vectors and FPXA graphs as ways of representing FPXA implementations for convenient, compact modeling and connection to dataflow graph analysis, respectively. Using a Viterbi decoder, we have demonstrated the prototyping of FPXAs on off-the shelf FPGA devices, and we have also provided a simple demonstration of iterative, constraint-driven FPXA design space exploration. Finally, we have motivated in some detail a number of useful directions for future work, which include the exploration of FPXA design tools for memory system synthesis, dynamic resource binding, and dataflow graph mapping.

7 Acknowledgments

This research was supported in part by the U.S. National Science Foundation under Awards 0720536, 0720596, 0823989, and 0824040.

References

[1] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.

[2] S. S. Bhattacharyya, E. Deprettere, R. Leupers, and J. Takala, editors. *Handbook of Signal Processing Systems*. Springer, 2010.

[3] W. Fu and K. Compton. An execution environment for reconfigurable computing. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, 2005.

[4] P. M. Heysters, J. Smit, G. J. M. Smit, and P. J. M. Havinga. Mapping of dsp algorithms on field programmable function arrays. In *FPL '00: Proceedings of the Roadmap to Reconfigurable Computing, 10th International Workshop on Field-Programmable Logic and Applications*, pages 400–411, London, UK, 2000. Springer-Verlag.

[5] H. Kee, S. S. Bhattacharyya, and J. Kornerup. Efficient static buffering to guarantee throughput-optimal FPGA implementation of synchronous dataflow graphs. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 136–143, Samos, Greece, July 2010.

[6] A. Kerihuel, A. Kerihuel, R. Mcconnell, R. Mcconnell, and S. Rajopadhye. Vsdf: Synchronous data flow for vlsi. Technical report, 1994.

[7] G. Lemieux and D. Lewis. *Design of Interconnection Networks for Programmable Logic*. Springer, 2004.

[8] J. Mcallister, R. Woods, R. Walke, and D. Reilly. Multidimensional DSP core synthesis for FPGA. *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, 43(2–3), June 2006.

[9] G. J. M. Smit, P. J. M. Havinga, L. T. Smit, P. M. Heysters, and M. A. J. Rosien. Dynamic reconfiguration in mobile systems. In *Proc. of FPL2002*, pages 162–170. Springer Verlag, 2002.

[10] S. Sriram and S. S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. CRC Press, second edition, 2009.

[11] T. Stefanov, C. Zissulescu, A. Turjan, B. Kienhuis, and E. Deprettere. System design using Kahn process networks: the Compaan/Laura approach. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, February 2004.

[12] G. Stitt, R. Lysecky, and F. Vahid. Dynamic hardware/software partitioning: a first approach. In *DAC '03: Proceedings of the 40th annual Design Automation Conference*, pages 250–255, New York, NY, USA, 2003. ACM.

[13] W. Wolf. *FPGA-Based System Design*. Prentice Hall, 2004.

[14] Xilinx, Inc. *PlanAhead User Guide, v 11.4*, 2009.