Synthesis of Application-Specific Highly-Efficient Multi-Mode Systems for Low-Power Applications

Lih-Yih Chiou, Swarup Bhunia and Kaushik Roy
School of Electrical and Computer Engineering,
Purdue University, West Lafayette, IN 47907, USA
lihyih, bhunias, kaushik@purdue.edu

Abstract

We present a novel design methodology for synthesizing multiple configurations (or modes) into a single programmable system. Many DSP and multimedia applications require reconfigurability of a system along with efficiency in terms of power, performance and area. FPGAs provide a reconfigurable platform, however, they are slower in speed with significantly higher power consumption than achievable by a customized ASIC. In this work, we have developed techniques to realize an efficient reconfigurable system for a set of user-specified configurations. A data flow graph transformation method coupled with efficient scheduling and allocation are used to automatically synthesize the system from its behavioral level specifications. Experimental results on several applications demonstrate that we can achieve about 60X power reduction on average with about 4X improvement in performance over corresponding FPGA implementations.

1. Introduction

State-of-the-art multimedia, communications and consumer electronics applications are witnessing a rapid development towards integrating complex system on a chip (SoC). Given the specifications of an application, it can be realized in one of the following hardware implementations: 1) Application Specific Integrated Circuit (ASIC), 2) Field Programmable Gate Arrays (FPGAs), or 3) a set of instructions running on a general-purpose processor. Alternate implementations of the same system allow trade-off between optimizing hardware in terms of multiple design parameters such as power, area, processing speed and reconfigurability of the system.

An ASIC allows designer to optimize the hardware resources for one or more of the design parameters. However, an ASIC implementation is not flexible since it does not allow to reconfigure itself and cannot be used in a wide range of applications. FPGAs, on the other hand, are arrays of pre-fabricated logic blocks and wire segments that are user-programmable. Although FPGAs have the capability of programming functional units and wires, it has several inherent limitations. FPGAs usually consume much higher power than an ASIC implementation. They also have higher performance penalty and larger area because of their generic reconfigurable platform. Another common method to implement a complex system with a given behavioral specification is to design a software program for running on a DSP processor. While it has the shortest turn-around time, DSP processors are designed for general-purpose DSP applications and hence, they are not area, performance and power efficient.

There has been a multitude of research work exploring efficient synthesis techniques for ASICs and FPGAs [1, 2, 3, 4]. A number of researchers have explored the possibility of reducing the gap between ASIC and FPGAs, thus merging the advantages of both worlds. Several researchers have addressed the issue of incorporating programmability into an ASIC implementation, while some have investigated more efficient utilization of FPGA resources to achieve improvement in performance and area. Guerra et al. have developed a behavioral synthesis technique for reconfigurable ASICs [5]. The technique uses Built In Self Repair (BISR) to dynamically replace a faulty module by another heterogeneous module, thus providing a fault-tolerant design method. Anderson et al. have introduced a coarse-grained FPGA architecture that allows designers to customize FPGAs for specific applications [2]. Zhang et al. report synthesis techniques for dynamically reconfigurable systems [4]. Dynamically reconfigurable systems use a dynamic allocation scheme that re-allocates resources at run-time to achieve higher performance and to minimize the number of hardware resources required to implement a complex system.

In this paper, we propose a novel architectural platform for realizing a set of selected configurations into a single system. We refer to the resultant system as a *Multi-Mode* (*MM*) system, in the sense that it can be configured to operate in multiple modes or configurations. Unlike FPGA, an *MM* system has limited reconfigurability, i.e., it can be programmed only through the small set of configurations it is designed for. However, *MM* systems can perform on-the-fly reconfiguration more efficiently than dynamically reconfigurable FPGAs due to the relatively small number of control

1530-1591/03 $17.00 © 2003 IEEE

signals needed. Easy and fast reconfiguration coupled with efficiency in terms of area and power consumption makes an *MM* system very attractive for power-conscious applications that monitor power during run-time.

The rest of the paper is organized as follows. Section 2 gives an overview of the conventional synthesis method. Section 3 describes the process for synthesizing *MM* systems from behavioral level specifications. Section 4 discusses the scope for efficiently searching the design space during implementation of an *MM* system. Section 5 presents experimental results on two different applications and section 6 concludes the paper.

2. Background

In this section, we briefly describe the general process of synthesizing a system from its behavioral specifications. Throughout the rest of the paper, we use the term *Single-Mode* (*SM*) system to represent ASIC implementation of a given configuration.

The computation flow of an algorithm is commonly represented using a data flow graph (DFG). In a DFG, nodes represent operations (e.g. additions or multiplications) and edges define the data dependencies between nodes [1]. We can represent a *SM* system by only one DFG and an *MM* system by a set of DFGs, each corresponding to a configuration of the system.

Behavioral-level synthesis transforms the architectural level specification into register transfer level (RTL), based on constraints on multiple design parameters. This is performed by the process of scheduling, which determines at what time step an operation will be executed. After scheduling, allocation designates an operation to a specific hardware resource e.g. functional unit (FU) or register.

In the step following scheduling and allocation, datapath elements, controller, steering logic circuits (multiplexers and buses) and input/output ports are mapped to hardware instances of the target architecture. Synthesis engines for ASICs use different schemes for datapath designs for different design styles. *Macro-cell-based* synchronous datapath design is typical to DSP circuits and we use it in synthesis of *MM* systems [3]. Fig.1 illustrates the structural view of a hardware model for an ASIC implementation. Major components of the hardware model are FUs, registers, multiplexers and a control unit. While we have used this hardware model for *MM* system synthesis, we are not limited to this model and the methodology can also be applied to other models.

3. Methodology

In this section, we explain the main steps for implementing an *MM* system in details. First, we describe the process to transform the problem of designing *MM* systems into

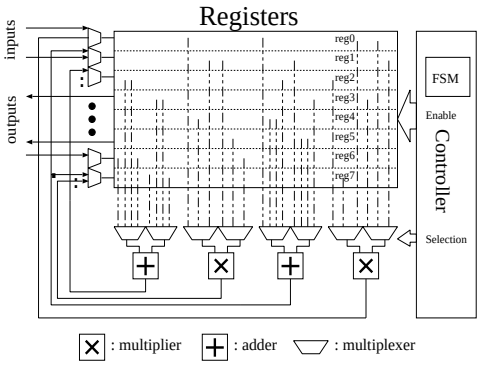

Figure 1. Structural view of the hardware model for an example.

scheduling and allocation problems, as used in conventional synthesis process. Then we describe how different scheduling and allocation algorithms can be applied to the synthesis of *MM systems* to meet different design objectives.

3.1. Transformation of Design Specifications

The input specifications for an *MM* system consist of a set of DFGs corresponding to the set of configurations for which the *MM* system is to be designed. One of the goals of an *MM* system design is to minimize area by reusing resources effectively among different configurations. Conventional scheduling and allocation algorithms used in ASIC synthesis can accomplish resource sharing efficiently. Our idea was to apply these algorithms in *MM* system synthesis by appropriately representing the set of DFGs into a single unified DFG. With this objective, we use a transformation method that utilizes spatial concatenation of DFGs to represent the set of DFGs into a single DFG and, thus, integrate the *MM* system design flow into the conventional ASIC design flow. We refer this transformation process as SPAtially Chained Transformation (SPACT). The key step in SPACT is to chain all DFGs into a single serially connected configuration with a dummy node between two

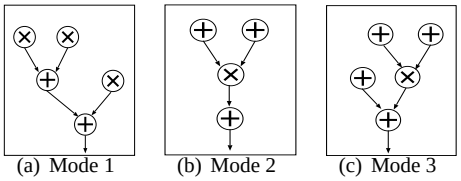

Figure 2. Three different configurations (represented with DFGs) to be implemented into a *Multi-Mode* system.

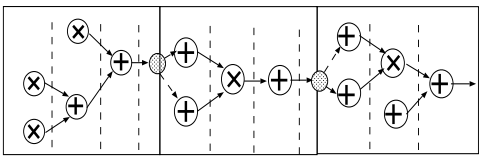

Figure 3. Transformed DFG before allocation. Shaded circles are dummy nodes.

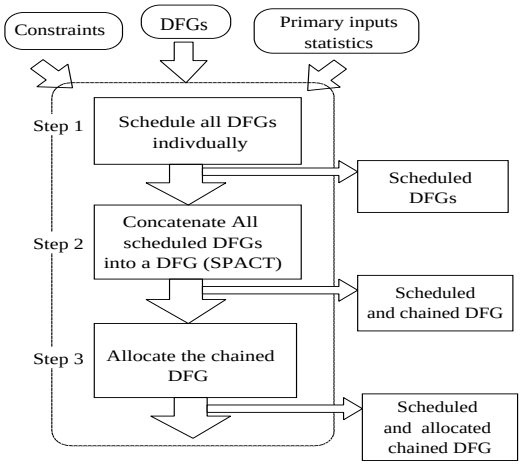

Figure 4. Main steps for realizing a *Multi-Mode* system

successive DFGs. It is worth observing that a resource can only be used at one specified control step at a time and only one of the configurations in a system can be active during the operation of an *MM* system. Hence, we can automatically share resources across configurations.

Let us take Fig.2 as an example. There are three configurations to be combined into an *MM* system, where each configuration is represented with a DFG. Let us assume that we have a design constraint to complete the operations in three time steps for all three modes. We first perform scheduling for each configuration independently and concatenate the scheduled configurations as shown in Fig.3, where the dummy nodes serve only as temporary connecting nodes. We, then, apply an allocation algorithm on this concatenated configuration. The final system, as shown in Fig. 1, has two multipliers, two adders, a few multiplexers and several registers.

3.2. Synthesis Flow for *Multi-Mode* Systems

The synthesis flow for *MM* systems is shown in Fig.4. We consider the timing/resource constraints on individual DFGs, while scheduling them. We perform scheduling before SPACT for the following reasons. 1) The run time for scheduling can be significantly reduced due to the fact that we schedule one DFG at a time. Because the complexity of common scheduling algorithms e.g. FDS, list scheduling etc. is usually a polynomial function of the number of nodes in the DFG, scheduling individual DFGs is more efficient with respect to run time than scheduling the chained DFG. 2) The resource sharing across configurations is not affected by the order of scheduling and concatenation. It is also possible to put SPACT before scheduling and allocation in the synthesis flow. This kind of ordering is necessary when one wants to use combined scheduling and allocation algorithms [6], where scheduling and allocation run simultaneously.

In the second step, we perform SPACT which concatenates all the scheduled DFGs into one large DFG (Fig.3). Because configurations are combined such that they have no overlaps in control steps, nodes of different configurations can be bound to the same resource whenever possible, during the allocation phase. In the next step, we allocate resources for the concatenated DFG. At this stage, we focus on binding operations to specific FUs and data flow edges to registers. It can be noted that the complexity of chaining is linear in terms of the number of total nodes in the DFGs and hence, the complexity of the synthesis process is determined by the scheduling and allocation algorithms.

We now describe how we can use alternative scheduling and allocation algorithms to meet distinct design objectives. In sections 3.3 and 3.4, we will explain the impact of different scheduling and allocation strategies on *MM* system design.

3.3. Resource-constrained Approach: SPACT-RC

In the resource-constrained approach, we use the list scheduling algorithm to assign operations to appropriate time steps [3]. For allocation, we want all available resources to be used as evenly as possible. Therefore, we use a variation of the weighted bipartite maximum matching (WBMM) algorithm [7]. The modified WBMM (MWBMM-I) algorithm binds operations to resource according to the specified resource constraints. In this approach, one can explore the design space by increasing the number of resources to improve power dissipation without harming performance. The performance of the *MM* system may improve if the scheduling algorithm can utilize the increased number of resources to reduce the number of control steps.

3.4. Signal Similarity-based Approach: SPACT-SIM

In this approach, the scheduling algorithm remains unchanged and we use signal statistics to dictate the process

of resource sharing during allocation. The allocation algorithm (MWBMM-II), also a modified version of WBMM, decides which operations will be shared such that the number of switching can be optimized. We use the concept of signal similarity [9] that is derived from the signal strength, as discussed in [10]. The greater the signal similarity among operations that can share the same FU, the less the switching power consumed by the FU. Hence, similarity-based allocation can effectively reduce power consumption in an *MM* system while maintaining performance.

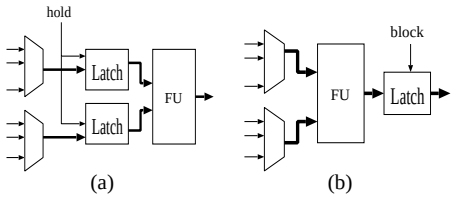

Figure 5. Power Reduction by Signal gating: a) operand isolation: holding; b) output blocking

4. Design Space Exploration

It is important to note that the scope for design space exploration in *MM* system design is considerably different from that of an ASIC design. An ASIC can be optimized for either power or performance or some other design parameters. Most often the design goal for an ASIC is to meet specific constraints on some or all of these parameters. In designing an *MM* system, on the other hand, we can optimize different configurations for different specified constraints. For example, if we are implementing two different configurations of FIR filters into an *MM* system, then we can optimize one configuration for power and the other for performance.

The ability to customize different configurations for different design objectives, has its impact on the amount of resource-sharing possible among the configurations. If a particular datapath element can be shared between two different configurations, but one of them need the element to be designed for minimum delay and the other for minimum power, then mostly likely we cannot share it. Hence, the design of *MM* system represents a trade-off between the amount of resource sharing and independent customizability of the configurations.

Design of a *MM* system also brings in several other optimization issues. Sharing of resources between two different configurations incorporates extra multiplexers/busses in front of the datapath elements. These elements consume power and cause extra delay in the design. Hence, we need to take additional care to optimize the impact of multiplexers/busses on the power and performance of the design. The select signals for these multiplexers are generated from the controller, which becomes more complex with more sharing among resources. The power and performance overhead due to additional complexity in the controller is usually very small and we can ignore the impact of the controller in meeting the design objective.

We can, however, exploit different techniques to optimize performance and power overhead due to sharing resources across configurations. Signal gating and increasing the number of resources are two simple techniques to deal with the overhead due to extra complexity in steering logic. Operand isolation [8] and output blocking are two common signal gating techniques which can be applied to reduce unnecessary switches in an *MM* system. The idea of operand isolation is to gate inputs of a component such that unnecessary switching in idle components can be prevented [8]. A component may remain idle in a time step, if it is used at some other time step or it is not used in the particular mode of operation. We need to consider both the cases to implement signal gating in *MM* system. *Holding* as shown in the Fig.5 puts extra latches at the inputs of FUs, which may be a concern for both performance and power. However, gating the inputs has large impact on power reduction in unused resources. Output blocking refers to gating the output of a component when the output is broadcasted to selected components. It is realized by inserting transparent latches at outputs of selected components, especially functional units. We need to consider extra delay and area due to the latch while making a decision about output blocking.

Another possible way to reduce delay and dynamic power due to sharing is to increase the number of resources, as mentioned in section 3.3. The most favorable situation in terms of performance and power would be to make enough resources available for every operation without any sharing. This increase in resources incurs an area penalty in the *MM* system. Furthermore, there is an optimum number of resources beyond which we can get only marginal improvement. Hence the design decision about the number of resources should consider these issues.

5. Experimental Results

In this section, we demonstrate the effectiveness of our methodology for implementing *MM* systems. We have studied two different examples of *MM* system design with varying level of complexity. The performance, power and area results for the designs are compared with ASIC and FPGA implementations. We also discuss how application of alternative synthesis approaches, presented in section 3, allows trade-off among different design optimizations. *MM* systems can achieve about 45% area saving over individual

SM implementations on an average with 7% average performance loss. The overhead in power consumption is about 19% on the average, compared to the power in corresponding *SM* systems. Furthermore, we show that *MM* systems have significant performance and power advantages over state-of-the-art FPGA implementations.

We have developed a synthesis tool that can realize a *MM* system from a set of DFGs. The tool integrates the transformation procedure (SPACT) with several scheduling and allocation strategies. The prototype tool has been implemented using programming language C++ with interface scripts in PERL. We use industry-standard analysis tools for evaluating the synthesized designs. The area is measured from the layout generated by Silicon Ensemble, while the delay is obtained by running PathMill. Power consumption of a design is reported from the output of PowerMill simulations. We use standard CMOS cells in 0.25 μm technology with supply voltage 2.5V, to map hardware instances in both *MM* and *SM* systems.

In presenting our evaluation results, we refer to two different experiments corresponding to two separate examples of *MM* systems. In both the experiments, EXP-I and EXP-II, we realize alternative filtering strategies. Three different configurations in EXP-I are respectively, a 7-tap FIR filter (FIR7) in direct form, a 4th-order IIR filter in direct form II (IIR4D2) and a 4th-order IIR filter in parallel form (IIR4P). These are low pass filters with distinct structures. In EXP-II, we implement four configurations of FIR filters with different taps - 7, 11, 15 and 19 respectively, all in the direct form, to perform low-pass filtering.

5.1. Area and Delay

Table 1 lists area and critical-path delay for both *SM* and *MM* design. The *MM* system can achieve about 45% of average area reduction over the cumulative area of corresponding *SM* implementations while incurring only about 7% performance penalty on average. The physical layout area includes the area occupied by interconnection and the control unit. The reduction in area is largely due to sharing resources effectively within and across configurations. Although there is extensive sharing in a *MM* design, the delay in the critical path is only affected marginally. The increase in delay is mostly contributed by additional delay in the steering logic.

EXP	Mode	Area (μm^2)	Critical-path Delay (ns)	Area Reduction	Delay Overhead
I	IIR4D2	796556	15.89	-45.4%	+9.6%
	IIR4P				+8.7%
	FIR7a				+7.6%
II	FIR7b	962361	16.53	-44.6%	+12.4%
	FIR11				+9.4%
	FIR15				+8.1%
	FIR19				+5.1%

Table 1. Area Reduction and Delay Overhead for *MM* systems over *SM* systems

MM System		Power	Power
EXP	Mode	(mW)	Overhead
I	IIR4D2	72.48	+9.7%
	IIR4P	109.82	+17.4%
	FIR7a	62.27	+11.6%
II	FIR7b	62.98	+19.0%
	FIR11	102.81	+25.3%
	FIR15	136.13	+30.2%
	FIR19	182.96	+19.2%

Table 2. Power Consumption for *MM* systems over *SM* systems

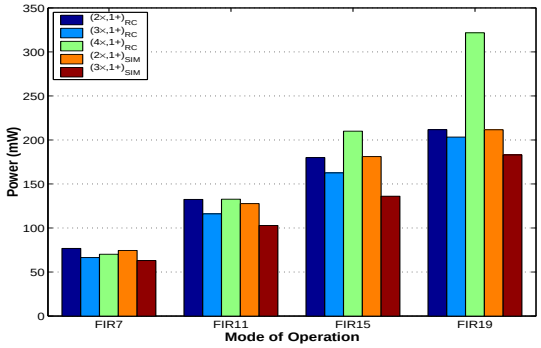

Figure 6. Power consumption for the *MM* system under various resource constraints and synthesis techniques for EXP-II.

5.2. Power Consumption

In Table 2, we show the power consumption results for *SM* and *MM* systems. We have observed that a particular configuration of an *MM* system consumes about 17% more power on an average than corresponding *SM* implementations. The power values are computed at 50 MHz frequency of operations. As in the case of delay, the penalty in power consumption is dominated by extra complexity in multiplexers. We can reduce the power overhead by simple gating techniques presented in section 4.

We plot the improvement in power consumption using SPACT-RC and SPACT-SIM approaches in Fig.6 for EXP-II. Five *MM* designs with resource constraints from (2×, 1+) to (4×, 1+) have been compared. We can observe that the power consumption can be significantly reduced by us-

	FIR7b	FIR11	FIR15	FIR19
$(3\times,1+)_{RC}$	66.35	116.07	162.64	203.26
$(3\times,1+)_{SIM}$	62.98	102.81	136.13	182.96
Reduction(%)	5	11	16	10

Table 3. The power reduction in $(3\times,1+)_{RC}$ and $(3\times,1+)_{SIM}$ for Fig.6. (in mW)

	FPGA @10MHz Power (mW)	*MM* system @10MHz Power (mW)	Ratio FPGA/*MM*
FIR7	1328	20.7	64.15
FIR11	2172	32.7	66.42
FIR15	2816	44.4	63.42
FIR19	3080	57.6	53.47

Table 4. Comparison of power consumption between FPGA and *MM* implementations

ing both appropriate number of resources and SPACT-SIM approach. Table 3 summarizes the power reduction between $(3\times,1+)_{RC}$ and $(3\times,1+)_{SIM}$. It can be observed that SPACT-SIM provides improvement of about 10.5% on average. We obtain this improvement only at the cost of estimating typical input statistics during allocation.

5.3. *MM* Over FPGA

How much better is an *MM* system over the FPGA implementations of the same configurations in terms of performance and power? We implemented the configurations of EXP-II in FPGA using SPACT-RC synthesis approach with minimum resources. We then implemented the same configurations as a *MM* system using the same synthesis approach. The *MM* system is mapped to $0.25\mu m$ CMOS technology, while the FPGA implementation uses the Xilinx Virtex FPGA family fabricated with $0.22\mu m$ CMOS process. Both the *MM* and FPGA implementations use 2.5 v supply voltage. The power consumption for the FPGA implementation is analyzed and reported by the Xilinx XPower tool. Table 4 shows the power consumption and operating conditions for both the implementations. Based on the maximum critical-path delay, the *MM* system can operate at 58 MHz, while the FPGA can operate at 16 MHz. Hence, the operating frequency of the FPGA-based system is about 4x slower than that of the *MM* system, although the FPGA family uses slightly more advanced process. The power consumption while operating at 10MHz is about 53X-66X higher in FPGA. For applications requiring dynamic reconfigurability, therefore, the *MM* systems not only provide significantly faster operations but also much better power usage.

6. Conclusions

We have proposed a novel architectural framework, which can customize datapaths and control logic as in ASIC and can, at the same time, allow easy dynamic reconfigurability throughout a small set of configurations. The most important advantage of the new framework is that we can easily integrate the synthesis process of a *MM* system into the conventional synthesis flow for an ASIC. Hence, the rich varieties of scheduling and allocation algorithms available in high-level synthesis can be easily applied to design a *MM* system. The idea of effectively sharing resources among different configurations can generate a system that is efficient in terms of area. Furthermore, the performance and power of different configurations can be optimized independently, which has its potential in present-day DSP and multimedia applications.

References

[1] D. Gajski and N.Dutt and A. Wu, and J.T. Ludwig. *High-Level Synthesis*. Kluwer Academic Publishers, 1992.

[2] J. R. Anderson, S. Sheth, and K. Roy. A Coarse-Grained FPGA Architecture for High-Performance FIR Filtering. In *Proceedings of 1998 ACM/SIGDA sixth International Symposium on Field Programming Gate Arrays*, pages 234–243, 1998.

[3] G. D. Micheli. *Synthesis and Optimization of Digital Circuits*. McGRAW-HILL, 1994.

[4] X. Zhang and K. W. Ng. A review of High-Level Synthesis for Dynamically Reconfigurable FPGAs. *Microprocessors and Microsystems*, 24:199–211, 2000.

[5] L. M. Guerra, M. Potkonjak, and J. M. Rabaey. Behavioral-Level Synthesis of Heterogeneous BISR Reconfigurable ASIC's. *IEEE Trans. on VLSI Systems*, 6(1):158–167, March 1998.

[6] C. Gebotys. ILP Model for Simultaneous Scheduling and Partitioning for Low Power System Mapping. Technical report, University of Waterloo, Department of Electrical and Computer Engineering, VLSI Group, April 1995.

[7] C.-Y. Huang, Y.-S. Chen, Y.-L. Lin, and Y.-C. Hsu. Data Path Allocation Based on Bipartite Weighted Matching. In *Proceedings of ACM/IEEE Design Automation Conference*, pages 499–504, June 1990.

[8] E. Mussol and J. Cortadella. High-Level Synthesis Techniques for Reducing the Activity of Functional Units. In *International Symposium on Low Power Design*, pages 99–104, April 1995.

[9] L.-Y. Chiou, K. Muhammad, and K. Roy. DSP Datapath Synthesis for Low-Power Applications. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 1165–1168, May 2001.

[10] L.-Y. Chiou, K. Muhammad, and K. Roy. Signal Strength Based Switching Activity Modeling and Estimation for DSP Applications. *VLSI DESIGN*, pages 233–243, Feb 2001.