Synthesis of Application-Specific Highly Efficient Multi-Mode Cores for Embedded Systems

LIH-YIH CHIOU, SWARUP BHUNIA, and KAUSHIK ROY
Purdue University

In this paper, we present a novel design methodology for synthesizing multiple configurations (or modes) into a single programmable core that can be used in embedded systems. Recent portable applications require reconfigurability of a system along with efficiency in terms of power, performance, and area. The field programmable gate arrays (FPGAs) provide a reconfigurable platform; however, they are slower in speed with significantly higher power and area than achievable by a customized application-specific integrated circuits (ASIC). Implementation of a system in either FPGA or ASIC represents a trade-off between programmability and design efficiency. In this work, we have developed techniques to realize efficient reconfigurable cores for a set of user-specified applications. The resultant system, named as *multimode* system, can easily switch configurations throughout the set of configurations it is designed for. A data flow graph transformation method coupled with efficient scheduling and allocation is used to automatically synthesize a *Multi-Mode* system from its behavior-level specifications. Experimental results on several applications demonstrate that our implementations can achieve about 60X power reduction on average and run 3.5X faster over corresponding FPGA implementations.

Categories and Subject Descriptors: C.3 [**Special-Purpose and Application-Based Systems**]: *Real-time and embedded systems*; B.5.2 [**Register-Transfer-Level Implementation**]: Design Aids—*Automatic synthesis*

General Terms: Design, Experimentation, Reliability, Performance

Additional Key Words and Phrases: Digital signal processing (DSP), application specific integrated circuits (ASIC), embedded systems, high level synthesis, synthesis, reconfigurable system

1. INTRODUCTION

State-of-the-art multimedia, communications, and consumer electronics applications are witnessing a rapid development toward integrating complex system on a chip (SoC). Such a sophisticated embedded system usually consists of a central control unit, coprocessors, A/D and D/A converters, and memory. Given the specifications of an application, the coprocessor can be realized in one of the

Authors' address: L.-Y. Chiou, S. Bhunia, and K. Roy, Department of Electrical and Computer Engineering, Purdue University, IN 47907; email: bhumias@purdue.edu.

following hardware implementations: (1) application specific integrated circuit (ASIC), (2) field programmable gate arrays (FPGAs), or (3) a set of instructions running on an application-specific processor. Various implementations of the same system allow trade-offs for optimizing hardware in terms of multiple design parameters such as power consumption, area, processing speed, and reconfigurability of the system.

There are number of advantages and disadvantages for these three implementations. An ASIC implementation has fully customized datapaths and logic. It allows designers 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 reconfiguring itself and cannot be used in a wide range of applications. FPGAs, on the other hand, consist of arrays of prefabricated logic blocks and chunks of user-programmable wire segments. FPGAs can provide a reconfigured implementation of a coprocessor. The property of reconfigurability allows designers to reuse resources in variety of applications. 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 require larger silicon area because of their generic reconfigurable platform. Another common method to implement a complex coprocessor is to use application-specific processors such as a DSP processor. DSP processors (DSPPs) are designed for general-purpose DSP applications; and hence, they are not area, performance, and power efficient.

1.1 Related Work

A multitude of research work have explored numerous methods to reduce 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. Dynamically reconfigurable FPGA systems use a dynamic allocation scheme that reallocates resources at run-time to achieve higher performance. Zhang and Ng [2000] have surveyed synthesis techniques for dynamically reconfigurable systems. On the other hand, Anderson et al. [1998] have introduced a coarse-grained FPGA architecture that allows designers to customize FPGA for a specific application.

Some researchers have focused on solving issues related to the synthesis of reconfigurable ASICs. va der Werf et al. [1992] have tried to find common substructures among different configurations and to generate a processing unit that can run in various configurations. They use iterative improvement and simulated annealing by minimizing the interconnection cost. Guerra et al. [1998] have developed behavioral synthesis techniques that may be extended to reconfigurable ASICs. The original technique uses built in self repair (BISR) to dynamically replace a faulty module by another heterogeneous module, thus providing a fault-tolerant design. If one considers the operation under a faulty situation as one of the possible operating configurations, their techniques are

applicable to reconfigurable ASICs. Two schemes are proposed in their paper: an iterative algorithm that is based on some heuristic costs and a transformation algorithm that changes original computation structures. Kim et al. [1997] have developed synthesis techniques for combining a finite set of applications into a reconfigurable core, referred as application-specific programmable processor (ASPP). They propose an application bundling strategy to group compatible applications together and then, for each group, they find required number of resources by checking the word-length and the type of functional operations. After the number and the type of resources are known, resource binding and scheduling are performed based on the goal of optimizing area.

Other researchers have explored the design of application specific instruction-set processors (ASIPs). Fisher [1999] discusses the challenges in designing ASIP. Jain et al. [1991] have identified key steps in designing ASIPs and provided brief survey on approaches for every step. In the work of Cousin et al. [2000], a multialgorithm synthesis technique for designing an application specific instruction-set processor (ASIP) is presented. An ASIP offers an instruction-set architecture, which is basically a stripped-down version of a DSP. The instructions of an ASIP for a set of target applications are determined by a process called instruction subsetting, starting from an instruction set for a particular DSPP. Recent work by Sun et al. [2002] and Gupta et al. [2002] suggest a design approach for ASIPs by augmenting custom instructions. Sun et al. propose a brand-and-bound method for selecting custom instructions. A set of near-the-best solutions are generated and then evaluated by logic synthesis. Gupta et al. propose use of a heuristic to evaluate the cost of the interdependence, which stems from added hardware when extending instructions. Then, they solve the problem of selecting instructions by formulating the problem as an optimization problem with the improved cost function. Glokler and Meyr [2001] apply power reduction techniques from logic level to system level on an ASIP and evaluate their effectiveness. Their analysis gives an idea about the potential power saving in an ASIP. There are also several emerging commercial products on extensible processor cores, such as Xtensa from Tensilica [1999] and ARCt-angent from ARC [2000], on the market. ASIP provides custom hardware, which is moderate in performance and efficient in terms of power cost. Although the ASIP offers flexibility throughout its small well-defined set of instructions, it requires compiler support and needs to reload microinstructions for every reconfiguring process. Furthermore, ASIPs significantly lack in the potential of customizability as an ASIC due to their instruction-set architecture.

1.2 Paper Overview

In this paper, we present a design methodology for combining a set of selected configurations into a single reconfigurable 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. We demonstrate how design of *MM* system can be seamlessly integrated into the conventional ASIC synthesis flow, with behavioral level specifications of the configurations as input. It allows the *MM* system designer to leverage ASIC-like customizability of data and

control paths. Unlike FPGAs, an *MM* system has limited reconfigurability, that is, it can be operated within 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 need for relatively small number of control signals. 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. Let us consider a mobile system where we can apply such an *MM* system to switch coding and encoding schemes or filtering methods according to environmental conditions or battery life. When the battery of the device is fully charged or plugged into a power outlet, the device can be operated in a high-performance mode. The device can be triggered to run in low-power mode (e.g., using a low-power coding technique using the same underlying reconfigurable hardware) when the energy of the battery drops below some threshold level.

Although there are few existing works [Kim et al. 1997; va der Werf et al. 1992] on synthesizing a number of applications into an ASIC, our approach differs from both in synthesis objective and approach. Unlike other works, the *MM* system proposed here targets realizing a set of compatible DSP applications into a single core for on-demand dynamic re-configurability based on power and performance requirements [Chiou et al. 2003]. For emerging wireless and multimedia devices, this kind or reconfigurability is very useful. In the design flow for *MM* system, we, thus, explore the opportunity to employ power optimization techniques with minimal impact on area. The *MM* system design allows individual configurations to meet their timing requirements by performing independent scheduling. But the rest of the design flow falls in place with conventional ASIC synthesis flow, with the help of a simple data flow graph concatenation process.

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

2. BACKGROUND

In this section, we briefly describe the conventional process of synthesizing a system from its behavioral-level specification. We explain the method of representing behavioral specifications for a system, scheduling control steps, and allocating resources. These steps do not depend on the target architecture, for example, ASIC or FPGA. In the next step, referred to as *technology mapping*, resources are mapped to specific hardware instances. Throughout the rest of the paper, we use the term *Single-Mode* (*SM*) system to represent the 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 flowgraph operations (e.g., additions or multiplications) and edges define the data dependencies between nodes

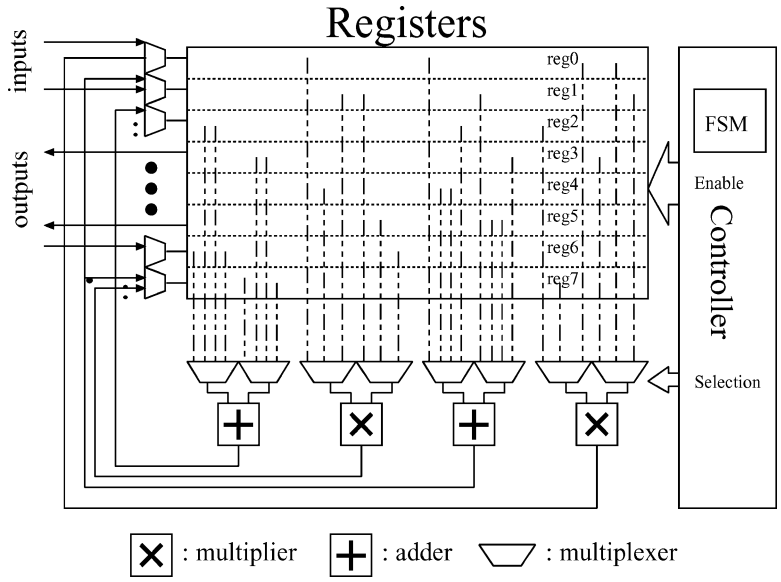


Fig. 1. Structural view of a hardware model.

[Gajski et al. 1992]. We can represent an *SM* system by only one DFG or an *MM* system by a set of DFGs, each corresponding to a configuration of the system. By representing algorithms in DFG format, we can conveniently process them in behavioral-level synthesis.

2.1 Scheduling and Allocation

Behavioral-level synthesis transforms the architectural-level specification into register transfer level (RTL) based on constraints on multiple design parameters. The synthesis process starts with scheduling operations, which determines at what time step an operation will be executed. After scheduling, allocation designates an operation to a specific hardware resource, for example, functional unit (FU) or register. In an *MM* system design, scheduling and allocation steps need to be appropriately modified to consider resource sharing across configurations.

2.2 Hardware Model

In the step following scheduling and allocation, datapath elements, controller, steering logic (multiplexers and buses), and input/output ports are mapped to hardware instances of the target architecture. Synthesis engines for ASICs can use various design styles for datapath. *Macro-cell-based* synchronous datapath design [Micheli 1994], which uses predesigned datapath macros for arithmetic operations, is typical to DSP circuits and we use it in synthesis of *MM* systems. Other possibilities, bus-oriented, bit-sliced, or array-based datapath, can also be used [Micheli 1994]. Figure 1 illustrates the structural view of a model for a specific ASIC implementation. Major components of the hardware model are FUs, registers, multiplexers, and control unit. It uses shared registers/latches

for storing intermediate results. Some hardware models use dedicated storage units to reduce interconnect overhead. The controller unit is commonly designed as a finite state machine (FSM) that generates control signals for the steering logic. While we have used this hardware model for *MM* system synthesis, we are not limited to this hardware model and the methodology can also be applied to other models with varying design styles of datapath, register, and controller.

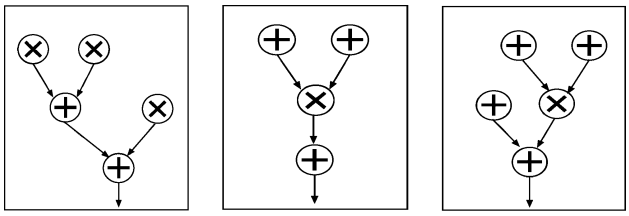
3. METHODOLOGY

In this section, we present the steps for implementing an *MM* system in details. First, we describe the process to transform the problem of designing *MM* systems into conventional scheduling and allocation problems. 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 specification of an *MM* system consists 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. An intuitive way to share resources across configurations would be using subgraph isomorphism to search for similar topology in the configurations. A simulated annealing-based strategy based on similar concept is explored in va der Werf et al. [1992], which does not prove to be efficient because of the complexity of the algorithm to find graph isomorphism. Conventional allocation algorithms used in ASIC synthesis can accomplish efficient resource sharing. Our idea is to apply these algorithms in *MM* system synthesis by appropriately integrating the DFGs into a single combined 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 as SPA-tially 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 adjacent DFGs. It is worth observing that only one of the configurations in a system can be active at a time. Hence, we can automatically share resources across configurations during allocation.

Let us take Figure 2 as an example. There are three configurations (applications) 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 scheduled configurations as shown in Figure 3, where the dummy nodes (shaded circles) serve only as temporary connecting nodes. We, then, apply a minimum-resource allocation algorithm on this concatenated configuration. The final system as shown in Figure 1, has two multipliers, two adders, a few multiplexers, and registers. This straightforward example illustrates the key ideas, which we will describe in more detail in the next Section 3.2.



(a) Mode 1 (b) Mode 2 (c) Mode 3

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

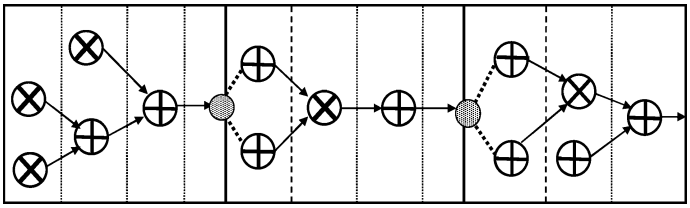


Fig. 3. Transformed DFG before allocation. Shaded circles are dummy nodes.

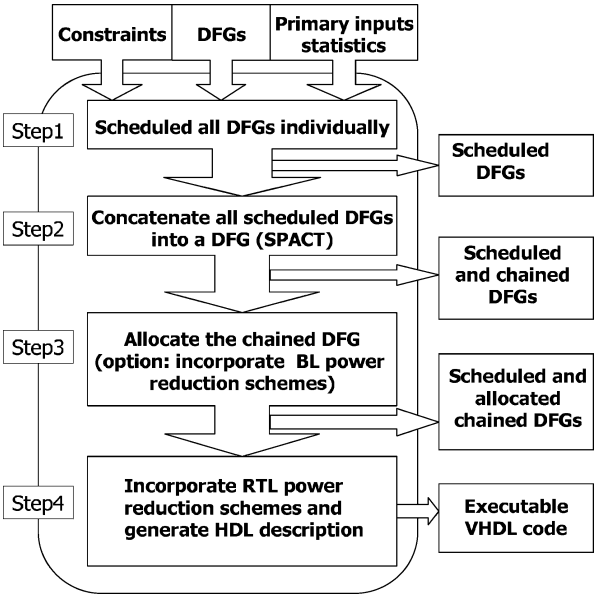


Fig. 4. Main steps for realizing *Multi-Mode* system.

3.2 Synthesis Flow for *Multi-Mode* Systems

The synthesis flow for *MM* systems is shown in Figure 4. First, we perform scheduling on all DFGs individually under given timing/resource constraints. Scheduling assigns a time step to each operation satisfying the constraints on respective DFGs. We perform scheduling before SPACT for the following

Table I. Several Alternative Scheduling/Allocation Strategies for Designing *MM* Systems

	SPACT-MR	SPACT-RC	SPACT-SIM
Objective	Minimum resource under time constraint	Minimum time under resource constraint	Minimum time & power under resource constraint
Scheduling	FDS	List	List
Allocation	WBMM	WBMM-I	WBMM-II

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, such as force-directed scheduling (FDS) [Paulin and Knight 1989] or List scheduling [Jain et al. 1991; Nestor and Thomas 1986] and others, is usually a polynomial function of the number of nodes in the DFG, scheduling an individual DFG 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 [Gebotys 1995].

In the second step, we perform SPACT, which concatenates all the scheduled DFGs into one large DFG (Figure 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 third step, we allocate resources for the concatenated DFG. At this stage, we focus on binding operations to specific FUs and dataflow 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. After we obtain time step and resource assignment from previous steps, we have enough information to integrate power reduction schemes at register-transfer level (RTL).

Power is an important optimization objective in the design of an *MM* system. We have detailed some useful power reduction strategies in Section 4.1. Some of the power optimization strategies can be applied during the allocation and binding step (step 3 in Figure 4), while others can be applied on the netlist generated after the allocation step, which includes gate-level and layout-level power optimizations. It is worth noting that, power and timing are most often conflicting design requirements and meeting specific goals in terms of some parameters may require design changes at an earlier stage, for example, rescheduling of some of the configurations. An executable VHDL description is generated at step 4. Then, the description can be used by industrial CAD tools to generate a transistor-level netlist and a physical layout.

3.3 Application of Different Scheduling/Allocation Algorithms

We now describe how we can use alternative scheduling and allocation algorithms to meet distinct design objectives for *MM* system designs. Table I lists three different synthesis approaches with different design goals.

In Sections 3.3.1–3.3.3, we will explain the impact of different scheduling and allocation strategies on *MM* system design.

3.3.1 *Minimum Resource Approach: SPACT-MR.* In this approach, listed as SPACT-MR in the second column of Table I, we use minimum resources under time constraints. We apply FDS to schedule individual DFGs given the maximum allowable time steps for every configuration. FDS is a heuristic-based approach and has shown its effectiveness in time-constrained scheduling [Gajski et al. 1992]. For allocation, we use weighted bipartite maximum matching (WBMM) algorithm [Huang et al. 1990]. Since we use several variations of the WBMM algorithm in our synthesis approaches, we briefly describe the key ideas behind WBMM. Primarily, there are two tasks in the WBMM algorithm: (1) to classify operations into several clusters, and (2) to perform matching. The WBMM algorithm first partitions operations of the same types into clusters, where each cluster contains operations whose time steps are overlapped with one another. It finds an assignment between members of one cluster and the available resources. The WBMM algorithm not only finds the minimum number of resources, but also binds operations to resources. While SPACT-MR can be an efficient approach in terms of area, it is expected to be less efficient with respect to power consumption since it simply minimizes the number of resources.

3.3.2 *Resource-Constrained Approach: SPACT-RC.* One might want to explore the design space by increasing the number of resources to offset power dissipation. In the resource-constrained approach, listed in column 3 of Table I, we use the List scheduling algorithm to assign operations to appropriate time steps [Micheli 1994] considering the requirement for the maximum number of resources. For allocation, we want all available resources to be used as evenly as possible. We use a modified WBMM (MWBMM-I) algorithm, which uses the maximum number of available resources specified as a constraint. SPACT-RC does not optimize the number of resources and may not be as efficient as SPACT-MR in area, but may be suitable for cases where power is a more important design concern.

3.3.3 *Signal Similarity-Based Approach: SPACT-SIM.* A variation to the above two approaches that can reduce power consumption without performance and area penalties is listed in column 4 of Table I. The approach uses the concept of signal similarity [Chiou et al. 2001a] that is derived from the signal strength as described in Chiou et al. [2001b] and employs the concept in the allocation stage. Essentially, signal strength is used 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 minimized. Here, the scheduling algorithm remains the same as that of earlier approaches. Brief description of signal strength and signal similarity is given below.

Signal strength that is derived from world-level signal statistics provides us means to analyze switching activity of a datapath component, with only one parameter per input signal [Chiou et al. 2001b]. The signal strength, η, is

defined as the number of bits needed to represent the average signal power. η is given as

$$\eta = \log_2 \left((2^{N-1} - 1) \times \frac{\sqrt{E(X^2[n])}}{d} + 1 \right) \tag{1}$$

where $X[n]$ represents an input sequence to a data-dominated system and is assumed to be a stationary Gaussian process. $E(X^2[n])$ is the average signal power of $X[n]$, and N is the number of bits used to represent the signal value. $\sqrt{E(X^2[n])}$ equals the standard deviation for zero-mean signals. All signals are assumed to be uniformly quantized in a dynamic range of $\pm d$ and are represented in sign magnitude form using N bits. With the given statistics of an input sequence (such as average, variance, and correlation), we can compute η of the sequence by using Eq. (1). The signal strength provides a concise way to express the effective bit length of a signal.

Signal strength can be used to improve power consumption of an *MM* system by binding operations smartly. This is based on the observation that switching activity (therefore power consumption) of shared resources can be reduced if we can effectively combine operations based on signal strength. It is known that switching occurs when two successive data signals have different values. By comparing the signal strength of two relevant signals, we can gain some insight about their differences. With the differences available, we select operations whose input signals have a closer value of signal strength to share a resource. This is because the closer the signal strength, the smaller the difference in effective bit length of signals. Therefore, the expected switching activity would be less.

The signal similarity is a way to measure the difference among signals and is defined as follows. The signal similarity, ($\mathcal{S}$), of two operations is given as

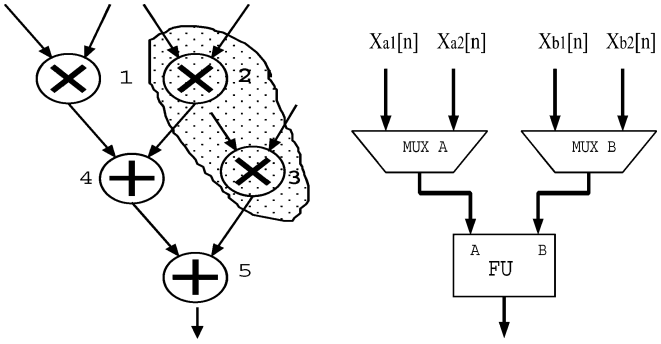
$$\mathcal{S} = \frac{1}{D_s + 1} \tag{2}$$

$$D_s = \frac{1}{k-1} \left(\sum_{i=1}^{k-1} |\eta_{a_i} - \eta_{a_{i+1}}| + \sum_{i=1}^{k-1} |\eta_{b_i} - \eta_{b_{i+1}}| \right) \tag{3}$$

where k is number of sharing for a particular resource.

The similarity is a function of signal strength for signals, which are inputs to a shared resource [Chiou et al. 2001a]. An example of sharing a resource is illustrated in Figure 5, where a FU is shared by two operations. Figure 5(a) shows two operations, represented by node 2 and node 3, which are selected to share one resource, a multiplier in this case. The corresponding implementation in hardware is shown in Figure 5(b). Multiplexers are placed before the input A and B of the FU and used to switch data sequences from different sources. Specifically, the data sequence through input A (or B) is interleaved sequences between two input signal profiles, one from node 2 and the other from node 3.

The similarity, $\mathcal{S}$, indicates the closeness of two input signal profiles. Each input signal profile has two sequences. Let us take Figure 5(b) as an example. The sequences of the first signal profile, $X_{a1}[n]$ and $X_{b1}[n]$, are to be applied to



(a) DFG (b) Hardware implementation

Fig. 5. Two views of sharing a resource.

two inputs of a shared FU, which are marked as A and B as shown in Figure 5(b). D_s is the distance of ηs (signal strength) of two signal profiles. The ηs of the first signal profile are denoted as η_{a1} and η_{b1}, while η_{a2} and η_{b2} represent two sequences of the second signal profile. The smaller the D_s is, the higher is the similarity ($\mathcal{S}$).

The *MM* designs can benefit from incorporating the similarity analysis into allocation algorithms. Since the WBMM algorithm need to compute the cost of binding during the matching stage, the cost of similarity can be integrated into the cost of binding. Hence, the binding rule of similarity is still applicable, that is, the greater the signal similarity among operations that can share the same FU, the less the switching power consumed by the FU. Therefore, similarity-based allocation can effectively reduce power consumption within a configuration.

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 needs the element to be designed for minimum delay and the other for minimum power, then we may not be able to share the element. Hence, the design of an *MM* system represents a trade-off between amount of resource sharing and independent customizability of individual configurations.

Design of an *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 in 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 have developed an integrated synthesis flow for *MM* systems, which works on behavioral level description of configurations provided in the form of DFGs and generates RTL-level VHDL description for the resultant *MM* system. Scheduling of individual configurations is performed based on the timing constraint for that configuration. Once scheduled, the DFGs are transformed into an unified representation to be used by the allocation tool. Selection of hardware model has its impact on the complexity and the efficiency of the allocation process. Let us consider the choice of word-length of an arithmetic resource as an example. Arithmetic units of the same type but disparate word-lengths may reduce the amount of resource sharing and complicate the allocation process, while uniform word-length has the potential to increase hardware overhead. In our implementation, we have used uniform word-length for different operation types for the sake of simplicity. As mentioned in Section 3.2, various power reduction techniques can be applied to the transformed DFG during allocation and in the following stages. We have realized the following power optimization techniques in our synthesis tool.

4.1 Reducing the Overhead Due to Sharing

We can, however, exploit different techniques to optimize performance and power overhead due to sharing resources across configurations. Sections 4.1.1–4.1.3 describe three simple techniques to deal with extra complexity in steering logic.

4.1.1 *Signal Gating.* Operand isolation and output blocking are two common signal gating techniques that can be applied to reduce unnecessary signal switching 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 [Mussol and Cortadella 1995]. A component may remain idle in a time step, if it is not used in the particular mode of operation, or it is used at a different time step. We need to consider both cases to implement signal gating in *MM* systems. There are two possible ways to gate the inputs: *enabling* and *holding* as shown in Figure 6. *Holding* the inputs requires extra latches, 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 latches at outputs of selected components, especially functional units. Since there is usually a time difference

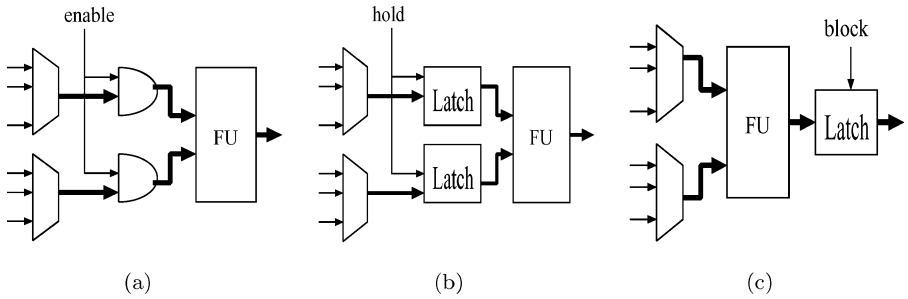
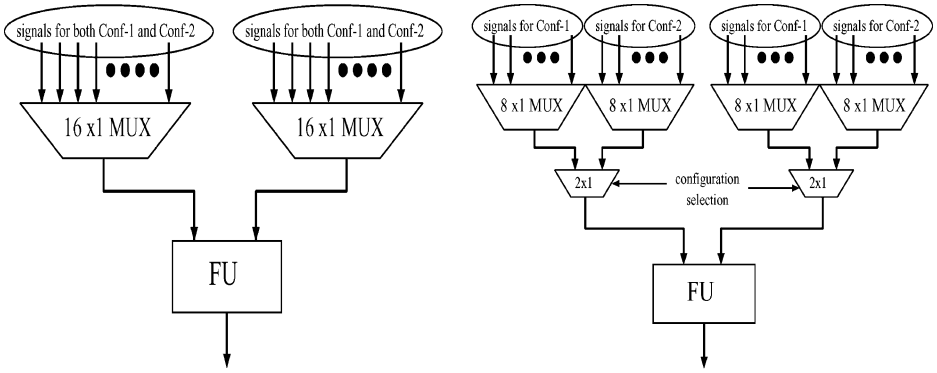


Fig. 6. Power reduction by signal gating: (a) operand isolation: enabling; (b) operand isolation: holding; (c) output blocking.

between the generation of the first and the last bit of output, the output bits will keep switching during the period of computation. By disabling latches when the computation is still in progress, unnecessary switching will be kept isolated from other components. Hence, power can be saved in the system. We also need to consider extra delay and area due to the latch while making a decision about output blocking.

4.1.2 *Increasing the Number of Resources.* Another possible way to reduce delay and dynamic power due to sharing is to increase the number of resources, as mentioned in Section 3.3.2. The most favorable situation in terms of performance and power would be to make enough resources available for every operation without any sharing. For Figure 2(a), the minimum number of resources under the constraint of three control-steps are two multipliers ($\times$) and one adder ($+$), and is represented as ($2\times$, $1+$) for simplicity. To avoid sharing, we need to have ($3\times$, $2+$). This increase in resources incurs an area penalty in the *MM* system. Furthermore, there are 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.

4.1.3 *Hierarchical Design of Multiplexers.* Increased complexity of the interconnection units, mainly the multiplexers may significantly contribute to the system power, due to their internal switching. Extra complexity in the multiplexers arises due to extensive sharing of arithmetic or storage resources within and across configurations as shown in Figure 7(a). In this example, a functional unit is shared among two configurations. Two multiplexers are used for two inputs of the FU, respectively. Each multiplexer is shared across configurations. One may carefully partition multiplexers according to configurations, that is, only signals that belong to the same configuration will share the same multiplexer. Therefore, the size of multiplexer can be effectively reduced and hence the power consumption. Figure 7(b) shows such an implementation with two levels of multiplexers. On the other hand, the number of multiplexer levels may increase due to the partitioning based on configurations. The resulting systems may have longer critical-path delay, which represents a trade-off between power and delay.



(a) MUXes without partition (b) MUXes with partition

Fig. 7. Design trade-offs for the multiplexers.

5. EXPERIMENTAL RESULTS

In this section, we demonstrate the effectiveness of our methodology for design of *MM* systems. We have studied three different examples of *MM* system design with varying levels of complexity. The area, delay, and power of the designs are compared with FPGA and ASIC implementations. We also discuss how application of different synthesis approaches, presented in Section 3, allows trade-off among different design objectives. *MM* systems can achieve about 45% area saving over individual *SM* implementations with 7% average performance loss. The overhead in power consumption is about 22% on the average, compared to 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 an *MM* system from a set of DFGs. The tool integrates the transformation procedure (SPACT) with several scheduling and allocation strategies. The tool has been implemented using programming language C++ and PERL. We use industry-standard analysis tools for evaluating the synthesized designs. The area is measured from the layout generated by Silicon Ensemble [Systems 2000], while the delay is obtained by running PathMill [Synopsys 2001a]. Power consumption of a design is reported from the output of PowerMill [Synopsys 2001b] simulations. We use standard CMOS cells in 0.25 μm technology with supply voltage of 2.5 V, to map hardware instances in both *MM* and *SM* systems.

The first experiment (EXP-I) involves design of an add-compare-select (ACS) network, or called ACSnet. ACSnet is used in Viterbi decoders, which are widely applied for decoding convolution codes [Lin and Daniel J. Costello 1983]. An ACS module and an ACSnet are shown in Figure 8. The number of ACS modules in an ACSnet is computed as 2^{K-1}, where K represents the constraint length used for encoding. Three different configurations of ACSnet correspond to three distinct constraint lengths, that is, $K = 5$, 6, and 7 with the number of

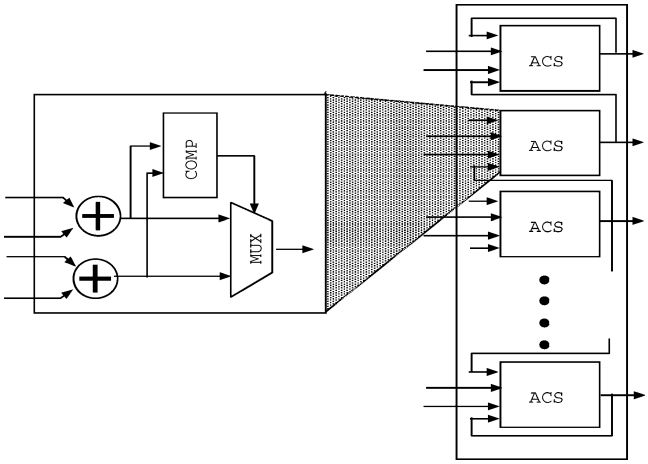


Fig. 8. The structure of an ACS module and the ACSnet (on the right).

Table II. Synthesis Approaches and Power Reduction Schemes Used in Each Experiment

Design Style	EXP-I (ACSnet)		EXP-II (Hybrid Filters)		EXP-III (FIR Filters)	
	Synthesis	Power Reduction	Synthesis	Power Reduction	Synthesis	Power Reduction
SM and MM	SPACT-MR	Enabling	SPACT-RC SPACT-SIM	I/P Holding O/P Blocking	SPACT-RC SPACT-SIM	I/P Holding O/P Blocking

ACS modules 16, 32, and 64, respectively. In the second and third experiments (EXP-II, EXP-III), we realize various filtering strategies. In these experiments, we use FIR filters in direct form, IIR filters in direct form II (IIR-D2), and IIR filters in parallel form (IIR-P) [Proakis and Manolakis 1996]. Three different configurations in EXP-II are, respectively, a 7-tap FIR filer (FIR7) in the 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-III, 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.

Table II summarizes different synthesis approaches and power optimization schemes, discussed in Section 3, that are applied to individual experiments. In each experiment, in addition to the *MM* system, we implement the *SM* systems for all its configurations. We apply the SPACT-RC and SPACT-SIM synthesis approaches to EXP-II and EXP-III and SPACT-MR to EXP-I only. This is because ACSnet needs a well-structured hardware, which is little affected by increasing resources (as in SPACT-RC) or by signal-similarity based resource allocation (as in SPACT-SIM) on system performance and power. On the other hand, EXP-II and EXP-III are sensitive to resource constraints and allocation strategies. As additional power reduction techniques, we use enabling for operand isolation in EXP-I and holding and output blocking in EXP-II and EXP-III. While enabling/disabling of inputs can effectively reduce power of ACSnet, the overhead of these methods is significant for EXP-II and EXP-III, where input holding and output blocking work better. In the rest of the section, we use

Table III. Area Reduction for *MM* Systems Over *SM* Systems

EXP	Mode	Area_{SM} (μm^2)	Area_{MM} (μm^2)	Area Reduction $\frac{\text{Area}_{MM} - \sum \text{Area}_{SM_i}}{\sum \text{Area}_{SM_i}}$
I	K = 4	214365	1,274,635	−14.6%
	K = 5	425103		
	K = 6	853776		
II	IIR4D2	459680	796,556	−45.4%
	IIR4P	462400		
	FIR7a	538020		
III	FIR7b	324328	962,361	−44.6%
	FIR11	405130		
	FIR15	448900		
	FIR19	558756		

Table IV. Delay Overhead for *MM* Systems Over *SM* Systems

EXP	Mode	Critical-Path Delay for SM Designs (ns)	Critical-Path Delay for MM Design (ns)	Delay Overhead $\frac{\text{Delay}_{MM} - \text{Delay}_{SM}}{\text{Delay}_{SM}}$
I	K = 4	4.20	4.42	+5.2%
	K = 5	4.23		+4.5%
	K = 6	4.33		+2.1%
II	IIR4D2	14.50	15.89	+9.6%
	IIR4P	14.65		+8.7%
	FIR7a	14.77		+7.6%
III	FIR7b	14.71	16.53	+12.4%
	FIR11	15.11		+9.4%
	FIR15	15.29		+8.1%
	FIR19	15.72		+5.1%

notation $(resource\ set)_{approach}$ to represent that a design is implemented using a specific synthesis approach with specific resource constraints, for example, $(4\times, 2+)_{SIM}$ indicates the design is realized using SPACT-SIM approach under the resource constraint of four multipliers and two adders.

5.1 Area and Delay

Tables III and IV list respectively area and critical-path delay for both *SM* and *MM* designs. The *MM* system can achieve up to 45% of area reduction over cumulative area of the corresponding *SM* implementations while incurring only about 7% performance penalty on average. The physical layout area includes the area occupied by interconnection and a control unit. The reduction in area is achieved largely due to sharing resources effectively within and across configurations. Although there is extensive sharing in an *MM* design, the delay in the critical path is only affected marginally as shown in Table IV. The increase in delay is mostly contributed by additional delay in the steering logic, which becomes more complex due to resource sharing across configurations.

5.2 Power Consumption

Table V shows the power consumption results for *SM* and *MM* systems. We have observed that a particular configuration of an *MM* system consumes about

Table V. Power Consumption for *MM* Systems Over *SM* Systems

MM System		Power_{SM}	Power_{MM}	Power Overhead
EXP	Mode	(mW)	(mW)	$\frac{\text{Power}_{MM}-\text{Power}_{SM}}{\text{Power}_{SM}}$
I	K = 4	5.74	6.84	+19.0%
	K = 5	11.35	13.52	+19.1%
	K = 6	23.7	24.17	+1.8%
II	IIR4D2	66.07	72.48	+9.7%
	IIR4P	93.52	109.82	+17.4%
	FIR7a	55.78	62.27	+11.6%
III	FIR7b	52.92	62.98	+19.0%
	FIR11	82.06	102.81	+25.3%
	FIR15	104.57	136.13	+30.2%
	FIR19	153.50	182.96	+19.2%

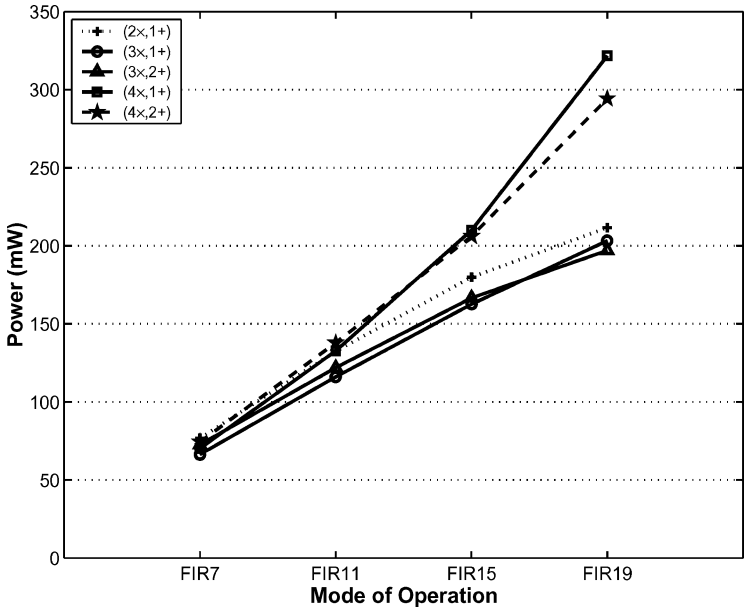


Fig. 9. Power consumption of the *MM* systems under various resource constraints using SPACT-RC technique for EXP-III.

17% (on an average) more power than corresponding *SM* implementation. 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 gating techniques presented in Section 4.

In Figure 9, we plot the power consumption of *MM* systems under different resource constraints for EXP-III. One may only get marginal reduction in power beyond some critical number of resources. For this particular case, the *MM* system cannot achieve more power reduction beyond what can be achieved for $(3\times, 1+)$. Similar experiment for EXP-II reveals that the critical resource set is $(4\times, 2+)$ for this design.

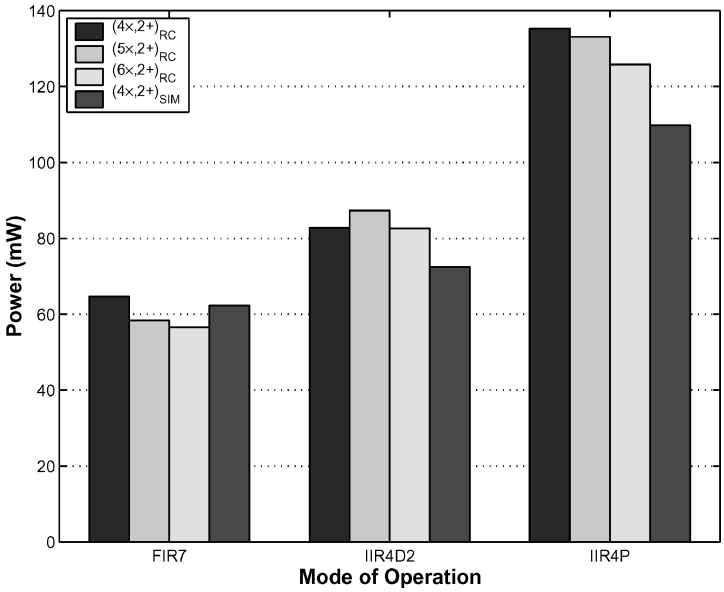


Fig. 10. Power consumption of *MM* systems under various resource constraints and synthesis techniques for EXP-II.

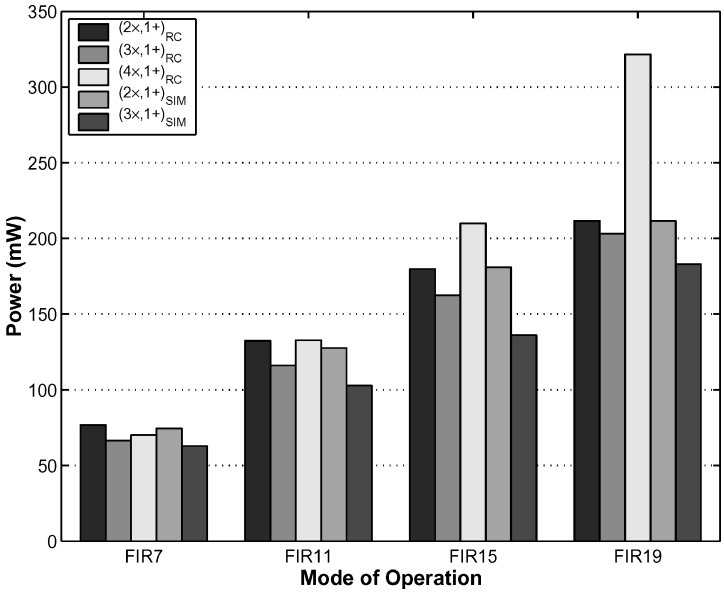


Fig. 11. Power consumption of *MM* systems under various resource constraints using different techniques for EXP-III.

We plot the improvement in power consumption using SPACT-SIM in Figures 10 and 11 for EXP-II and EXP-III, respectively. It can be observed that SPACT-SIM provides improvement of about 11% on average. In Figure 10, we consider four *MM* designs: three using constraints $(4\times, 2+), (5\times, 2+)$, and

Table VI. The Power Reduction in $(4\times, 2+)_{RC}$ and $(4\times, 2+)_{SIM}$ for Figure 10 (in mW)

	FIR7a	IIR4D2	IIR4P
$(4\times, 2+)_{RC}$	64.65	82.79	135.27
$(4\times, 2+)_{SIM}$	62.27	72.48	109.82
Reduction(%)	4	12	19

Table VII. Power Reduction in $(3\times, 1+)_{RC}$ and $(3\times, 1+)_{SIM}$ for Figure 11 (in mW)

	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 VIII. Comparison of Power Consumption Between FPGA and *MM* Implementations

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

$(6\times, 2+)$ are implemented with SPACT-RC, and the other using constraint of $(4\times, 2+)$ is implemented with SPACT-SIM. Similarly, in Figure 11 for EXP-III, five *MM* designs with resource constraints from $(2\times, 1+)$ to $(4\times, 1+)$ have been compared. In both cases, we can observe that the power consumption can be significantly reduced by using SPACT-SIM. Tables VI and VII summarize the power reduction between $(4\times, 2+)_{RC}$ and $(4\times, 2+)_{SIM}$ for EXP-II and $(3\times, 1+)_{RC}$ and $(3\times, 1+)_{SIM}$ for EXP-III.

5.3 Proposed *MM* System versus FPGA Implementation

How much better is an *MM* system over the FPGA implementation for the same configuration? We implemented the configurations of EXP-II in FPGA using SPACT-MC synthesis approach with minimum resources. We then implemented the same configurations as an *MM* system using the same synthesis approach. The *MM* system is mapped to 0.25 μm CMOS technology, while the FPGA implementation uses the Xilinx Virtex FPGA family fabricated in 0.22 μm CMOS process Xilinx [2001]. Both the *MM* and FPGA implementations use 2.5 V supply. The power consumption for the FPGA implementations is analyzed and reported by the Xilinx XPower tool [Xilinx 2002]. Table VIII 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 only at 16 MHz. Hence, the operating frequency of the FPGA-based system is about 3.5X slower than that of the *MM* system, although the FPGA family uses a more advanced process.

The power consumption is about 53X–66X higher in FPGA. For applications requiring dynamic reconfigurability, therefore, the *MM* systems not only provide significantly faster operations but also orders of magnitude better power consumption.

6. CONCLUSIONS

We have proposed a design framework that can be used to design reconfigurable cores referred as *MM* systems. An *MM* system can be customized 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 integrate the synthesis process of an *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 an *MM* system. The idea of effectively sharing resources among different configurations can generate a configurable core that is efficient in terms of die area. Furthermore, the performance and power of different configurations can be optimized independently, which has large potential in present-day multimedia portable applications.

REFERENCES

ANDERSON, J., SHETH, S., AND ROY, K. 1998. A coarse-grained FPGA architecture for high-performance fir filtering. In *Proceedings of 1998 ACM/SIGDA 6th International Symposium on Field Programming Gate Arrays*. 234–243.

ARCCORES. 2000. (http://www.arccores.com), A. I. Arctangent processor.

CHIOU, L., BHUNIA, S., AND ROY, K. 2003. Synthesis of application-specific highly efficient multi-mode systems for low-power applications. In *IEEE Design, Automation and Test in Europe Conference*.

CHIOU, L., MUHAMMAD, K., AND ROY, K. 2001a. DSP datapath synthesis for low-power applications. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 2, 1165–1168.

CHIOU, L., MUHAMMAD, K., AND ROY, K. 2001b. Signal strength based switching activity modeling and estimation for dsp applications. *VLSI Design*. 233–243.

COUSIN, J.-G., SENTIEYS, O., AND CHILLET, D. 2000. Multi-algorithm asip synthesis and power estimation for dsp applications. *International Symposium on Circuits and Systems*. II.621–624.

FISHER, J. 1999. Customized instruction sets for embedded processors. In *Proceedings of ACM/IEEE Design Automation Conference*. 253–257.

GAJSKI, D., DUTT, N., WU, A., AND LUDWIG, J. 1992. *High-Level Synthesis*. Kluwer Academic Publishers, Boston, MA.

GEBOTYS, C. 1995. *ILP Model for Simultaneous Scheduling and Partitioning for Low Power System Mapping*. Tech. rep., University of Waterloo, Department of Electrical and Computer Engineering, VLSI Group. April.

GLOKLER, T. AND MEYR, H. 2001. Power reduction for ASIPS: A case study. In *IEEE Workshop on Signal Processing Systems*. 235–246.

GUERRA, L., POTKONJAK, M., AND RABAEY, J. 1998. Behavioral-level synthesis of heterogeneous bisr reconfigurable ASIC's. *IEEE Trans. VLSI Syst. 6*, 1 (Mar.), 158–167.

GUPTA, T., KO, R., AND BARUA, R. 2002. Compiler-directed customization of asip cores. In *Proceedings of 10th International Symposium on Hardware/Software Codesign*. 97–102.

HUANG, C.-Y., CHEN, Y.-S., LIN, Y.-L., AND HSU, Y.-C. 1990. Data path allocation based on bipartite weighted matching. In *Proceedings of ACM/IEEE Design Automation Conference*. 499–504.

JAIN, R., MUJUMDAR, A., SHARMA, A., AND WANG, H. 1991. Empirical evaluation of some high-level synthesis scheduling heuristics. In *Proceedings of ACM/IEEE Design Automation Conference*. 210–215.

KIM, K., KARRI, R., AND POTKONJAK, M. 1997. Synthesis of application specific programmable processors. 353–358.

LIN, S. AND DANIEL J. COSTELLO, J. 1983. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, Englewood Cliffs, NJ.

MICHELI, G. D. 1994. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, New York.

MUSSOL, E. AND CORTADELLA, J. 1995. High-level synthesis techniques for reducing the activity of functional units. In *International Symposium on Low Power Design*. 99–104.

NESTOR, J. AND THOMAS, D. 1986. Behavioral synthesis with interfaces. In *International Conference on Computer Aided Design*. 112–115.

PAULIN, P. AND KNIGHT, J. 1989. Force-directed scheduling for the behavioral synthesis of ASIC. *IEEE Trans. Comput.-Aided Des. Integrated Circuits Syst. 8*, 6 (June), 661–678.

PROAKIS, J. G. AND MANOLAKIS, D. G. 1996. *Digital Signal Processing: Principles, Algorithms and Applications*. Prentice Hall, Englewood Cliffs, NJ.

SUN, F., RAVI, S., RAGHUNATHAN, A., AND JHA, N. 2002. Synthesis of custom processors based on extensible platforms. In *IEEE/ACM International Conference on Computer Aided Design*. 641–648.

SYNOPSYS. 2001a. *PathMill Reference Guide*. Synopsys Inc.

SYNOPSYS. 2001b. *PowerMill Reference Guide*. Synopsys Inc.

SYSTEMS, C. D. 2000. *Envisia Silicon Ensemble Place-and-Route Reference*. Cadence Design Systems Inc.

TENSILICA. 1999. (http://www.tensilica.com), T. I. Xtensa microprocessor.

VA DER WERF, A., AERTS, E., PEEK, M., VAN MEERBERGEN, J., LIPPENS, P., AND VERHAEGH, W. 1992. Area optimization of multifunction processing units. In *International Conference on Computer Aided Design*. 292–299.

XILINX. 2001. (http://www.xilinx.com), X. I. Xilinx virtex data sheet.

XILINX. 2002. *FPGA XPower Tutorial*. Xilinx Inc.

ZHANG, X. AND NG, K. W. 2000. A review of high-level synthesis for dynamically reconfigurable fpgas. *Microprocess. Microsyst. 24*, 199–211.

Received January 2003; revised July 2003; accepted September 2003