Hindawi Publishing Corporation
EURASIP Journal on Applied Signal Processing
Volume 2006, Article ID 79595, Pages 1–9
DOI 10.1155/ASP/2006/79595

Highly Flexible Multimode Digital Signal Processing Systems Using Adaptable Components and Controllers

Vinu Vijay Kumar and John Lach

Charles L. Brown Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, VA 22904, USA

Received 1 October 2004; Revised 21 March 2005; Accepted 25 May 2005

Multimode systems have emerged as an area- and power-efficient platform for implementing multiple timewise mutually exclusive digital signal processing (DSP) applications in a single hardware space. This paper presents a design methodology for integrating flexible components and controllers into primarily fixed logic multimode DSP systems, thereby increasing their overall efficiency and implementation capabilities. The components are built using a technique called small-scale reconfigurability (SSR) that provides the necessary flexibility for both intermode and intramode reconfigurabilities, without the penalties associated with general-purpose reconfigurable logic. Using this methodology, area and power consumption are reduced beyond what is provided by current multimode systems, without sacrificing performance. The results show an average of 7% reduction in datapath component area, 26% reduction in register area, 36% reduction in interconnect MUX cost, and 68% reduction in the number of controller signals, with an average 38% increase in component utilization for a set of benchmark 32-bit DSP applications.

1. INTRODUCTION

The burgeoning demand for high performance DSP systems has spurred widespread research on efficient platforms for implementing arithmetic intensive applications characteristic of such systems. Based on these applications' high throughput requirements, fixed logic application-specific integrated circuits (ASICs) are normally the platform of choice. However, their lack of flexibility is disadvantageous in today's world of disparate and rapidly evolving standards and applications, which require the execution of a variety of DSP tasks. In the absence of flexibility, direct hardware implementation of all of the tasks is the only option and can be prohibitively expensive—even in this "transistors for free" era. This has led to the search for new methods for adding flexibility to otherwise fixed logic DSP circuits, without having to pay the large performance, area, and power penalties associated with field programmable gate arrays (FPGAs), DSP processors, or even application-specific instruction processors (ASIPs).

An emerging platform that has been proposed to address the flexibility issue in ASICs for DSP is "multimode" systems [1, 2]. Tasks that are timewise mutually exclusive are synthesized to the same hardware area, allowing the tasks to be separated temporally rather than spatially. When a particular task must be executed, the system switches to the appropriate hardware configuration "mode." Such a design platform can prove useful for many DSP systems. For example, a system jointly implementing two different standards (e.g., CDMA/GSM formats in a cell phone, different region DVD formats in a Universal DVD player, etc.), where only one mode needs to be active at any given time, can benefit from a multimode implementation.

However, current multimode systems are severely constrained in their capabilities and efficiency due to their limited reconfigurability. Reconfiguration between modes is accomplished by changing only the dataflow between components; the datapath components themselves do not change. The individual controllers for each mode are composed together into a single controller that also does not change between modes. Hence, such a system is inefficient and not very powerful, with only the interconnections changing between modes.

In this paper, we present a multimode DSP system design and synthesis technique that provides greater implementation flexibility by enabling reconfigurability in controllers and datapath components, as well as the interconnections. In addition, reconfiguration may be performed not only between modes but also within a single mode. This technique provides improved results in terms of resource requirements and/or task latency and power consumption (via increased component utilization), when compared to existing multimode synthesis techniques, enabling more powerful DSP applications. While these improvements would likely be lost if general-purpose reconfigurable devices (e.g., FPGAs) were

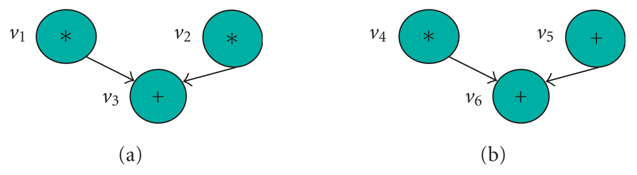

FIGURE 1: Sample DFGs.

used to provide the hardware flexibility, the technique presented here uses small-scale reconfigurability (SSR) [3, 4]. SSR provides hardware flexibility without the area, delay, power, and reconfiguration time penalties associated with general-purpose reconfigurable fabric.

1.1. Illustrative example

Consider the two dataflow graphs (DFGs) in Figure 1, each representing a different mode of a DSP system. The datapath and controller designs must be capable of implementing either mode.

1.1.1. Datapath design

Assuming the system is not pipelined and specifications dictate that modes have minimum latency, DFG 1a (Figure 1(a)) would require two multipliers (MULTs) and one adder (ADD), and DFG 1b (Figure 1(b)) would require one MULT and one ADD. If both tasks were implemented spatially separately, the total number of arithmetic components would be three MULTs and two ADDs.

However, if the tasks are timewise mutually exclusive, they may be implemented using existing multimode techniques in which components may be shared by both tasks, but must remain fixed. Such an implementation would require only two MULTs and one ADD, and the proper mode would be invoked based on the task to be implemented. While the tasks share datapath components, they require separate controllers. In addition, interconnect complexity is also high, as component interconnections have to support the dataflow in both product instances. For example, the adder component producing the overall output (mapped to nodes v_3 or v_6) gets its inputs either from the two multipliers (for DFG 1a) or from a multiplier and itself (for DFG 1b), requiring a MUX to be added at the inputs that was unnecessary in the separate single-mode implementations.

Now consider a flexible arithmetic component (FAC) capable of performing both addition and multiplication that is as fast as a multiplier but is smaller than the combined areas of an adder and multiplier (although larger than each individually). For example, the "morphable multiplier" uses the adder chains within the multiplier to perform addition with minimal area and no delay penalties [5]. The flexibility of a FAC is such that not only can it be a multiplier in one mode and an adder in another (intermode reconfiguration), but it could also be an adder and a multiplier in different control steps (c-steps) within the same mode (intramode reconfiguration).

Using FACs and intramode reconfiguration, DFG 1a can be implemented with one MULT and one FAC, with the FAC switching between a MULT (to execute v_2 in c-step 1) and an ADD (to execute v_3 in c-step 2). If DFG 1b were synthesized independently, the technique would allocate one MULT and one ADD. But given that the two DFGs are to be implemented in the same physical multimode space, they can be synthesized together, resulting in one MULT and one FAC. Given the assumption that one FAC is smaller than the combined area of an adder and a multiplier, area savings are achieved over existing limited reconfigurable multimode synthesis techniques. Component utilization is also increased, reducing wasted power consumption. Note that if only intermode reconfiguration is utilized (with the FAC functionality being fixed within a mode through all c-steps), the component allocation would be an ADD, a MULT, and a FAC, which still provides an improvement over the nonmultimode implementation.

This technique also inherently leverages any inter- and intra-DFG isomorphism. Using efficient binding of flexible components to nodes, the need for MUXes in the interconnection network is minimized without the need for any isomorphic subgraph identification and matching on the nodes in the DFG. Consider that nodes v_1 and v_4 in Figure 1 use the MULT component and nodes v_2, v_3, v_5, and v_6 use the FAC, so the component interconnects will not change from one mode to the other. Without the FAC, v_2 and v_5 would be executed on a MULT and ADD, respectively. Both operations' results would be input to the ADD, requiring a 2 : 1 MUX for the ADD and MULT to write to the same result register or for the ADD to read from different source registers for the different modes. For large system bit widths, MUXes become very expensive. The technique presented here minimizes the need for such MUXes (and control signals for the MUXes, which must be generated individually for each mode), while avoiding computationally intensive subgraph isomorphism identification algorithms [6].

1.1.2. Controller design

Conventional multimode or domain-specific customization approaches, where fixed datapath components are shared between tasks, require separate controllers, with the individual controllers typically MUXed at their outputs to form a composite controller. However, the composite controller in such designs often becomes complex enough to require a programmable microcode-based implementation, which is less area- and power-efficient and has lower performance than hardwired controllers.

SSR-based adaptable controllers can reduce these inefficiencies. Consider now a composite controller for the multimode system shown in Figure 1. Assume that the individual instance controllers have only minor differences; say, for example, one of their output functions is represented by f_1 in one and f_2 in the other. Assume f_1 and f_2 are defined as follows: $f_1 = a + b + ce + de$; $f_2 = ac + ad + bc + bd + e$. For a composite controller, both functions could be separately implemented on the same device, and selection between them for the specific mode could be performed with an output MUX. Using only 2-input fixed logic gates from a standard cell library such as the Lsi10k library, the total circuit cost for such an implementation is 19 inverter-equivalent gates.

The SSR-designed flexible controller provides the same flexibility more efficiently. f_1 and f_2 are jointly synthesized as a multi-output function, thus maximizing logic sharing between the functions and automatically obtaining the minimum implementation difference between them. The shared logic (in this case, the common subexpressions $X = a+b$ and $Y = c+d$) is then implemented in fixed logic, while the differences are implemented in configurable logic. In our example, the functions f_1 and f_2 are rewritten as $f_1 = X+e^*Y$ and $f_2 = X^*Y+e$, and jointly implemented using programmable interconnect for a total circuit cost of 12 inverter-equivalent gates.

An algorithmic framework for integrating flexible datapath and control components into multimode systems is presented in this paper. Conventional high-level synthesis tools do not take advantage of the range of flexibility provided by these components. For example, module allocation techniques for multifunction ALUs (e.g., using operation clustering, etc.) are not optimized for application-specific, limited flexibility addition. Other approaches for incorporating ALUs, in which allocation precedes synthesis, are ineffective for multimode systems since inter-DFG dependencies are not effectively extracted for optimum allocation. In the following sections, new algorithms and extensions to conventional algorithms are proposed for datapath synthesis, allocation, and binding, and for automatic control path synthesis for multimode DSP systems with flexible components and controllers.

2. BACKGROUND AND RELATED WORK

In order to leverage the increasing relative performance, area, and power benefits of hardware versus software, there is a trend towards implementing in hardware many algorithms and applications that had previously been done primarily in software. This is particularly evident with the advent of embedded system-on-a-chip (SOC) designs, in which embedded processors and application-specific circuitry share the processing load based on a designer-defined partition. For applications requiring several disparate performance-sensitive tasks, this often results in low resource utilization, a metric for hardware efficiency.

One approach that has been suggested to address the low resource utilization issue, as well as to enable more powerful hardware implementations of DSP systems, is to jointly synthesize different applications to build a unified datapath. By using separate controllers, the datapath may be animated to implement the various applications. This method was first explored in [2] as "multifunctional processing units." The work provided heuristic local search algorithms for the joint allocation of components so as to minimize interconnect. Designing application-specific programmable processors (ASPPs) by bundling similar applications and jointly synthesizing them has also been investigated [7]. Flexible datapaths have been proposed for fault tolerance purposes, with various configurations available to recover from components failures [8]. Most recently, a "spatially chained transformation" was introduced to enable dataflow graphs of different applications to be chained together for joint component allocation and binding [1]. The essential element in all of these efforts is that timewise mutually exclusive applications can reside in different configurations in the same physical area of a "multimode" system.

Domain-specific customization is a related approach for application-specific flexibility in reconfigurable systems [6, 9]. This approach involves creating a custom reconfigurable architecture to specifically implement a set of circuits from a given domain and be completely flexible within that domain. This is, in a sense, a mirror image of our approach. While ours is aimed at inserting small amounts of reconfigurability into primarily fixed logic circuits, domain-specific customization inserts fixed logic into circuits with primarily reconfigurable fabric. The synthesis techniques developed for such systems, therefore, address a different set of issues (e.g., template generation, isomorphic subgraph identification and matching, etc.) that is relevant to domain-specific customization. It is difficult to adapt these techniques to address issues specific to our problem, such as runtime reconfigurability within isomorphic subgraphs. The technique presented in this paper addresses such issues.

While hybrid FPGAs and reconfigurable cores provide hardware flexibility, their coarse integration of fixed logic and reconfigurable fabric result in significant area, performance, and power penalties. Techniques have therefore been explored to add flexibility to individual hardware components without the penalties associated with general-purpose reconfigurable arrays. By reusing adder chains within a multiplier, an area-efficient "morphable" multifunction component capable of both addition and multiplication was described in [5]. Such a unit is useful in DSP systems dominated by multiply-accumulate (MAC) chains. Another flexible component capable of both single-precision and double-precision floating point multiplication was described in [10]. Synthesis techniques that integrate flexible components into primarily fixed logic systems are detailed in [3]. That work augmented traditional force-directed list scheduling (FDLS) [11] for component scheduling and allocation using a hybrid library of fixed and reconfigurable arithmetic components, providing significant area savings for single-mode systems. The work presented here provides hybrid library synthesis techniques for multimode systems, yielding even greater savings.

3. EFFICIENT FLEXIBLE HARDWARE

Hardware flexibility is traditionally achieved with large-scale, general-purpose reconfigurable arrays such as FPGAs, which are significantly less efficient (in terms of area, delay, and power) than fixed logic. The logic is not as dense, delays are larger through SRAM-based lookup table (LUT) logic and programmable interconnects, and power consumption is greater due to the increased node capacitance.

The SSR design technique minimizes these penalties by inserting into a primarily fixed logic design only the flexibility that is required for a specific application. Reconfigurable logic and interconnect (e.g., SRAM-based LUTs, MUXes, SRAM-gated pass transistors, etc.) are finely integrated with fixed logic at a gate-level granularity. While some recent technologies contain both fixed and reconfigurable logic on the same chip, they are coarsely integrated. For example, some hybrid FPGAs contain a fixed logic core surrounded by general-purpose reconfigurable fabric. Domain-specific customization [6, 9], discussed in Section 2, provides another example. SSR allows for finer integration and application-specific implementation, providing the necessary flexibility with ASIC-like efficiency. In addition, the reconfiguration time is significantly shorter, as there is less to reconfigure. The tradeoff is the design effort and fabrication costs associated with all ASICs, but high-volume applications offset these costs, and many applications require ASIC performance.

The SSR design methodology can be applied to a range of designs and applications. The remainder of this section focuses on the use of SSR for designing FACs and adaptable finite state machines (FSMs), which enable efficient datapath and control flexibility.

3.1. Flexible arithmetic components

When designing a FAC, the similarities between the desired operations can be implemented in fixed logic, and reconfigurable logic and interconnect must be used to implement the differences. Therefore, the first step in designing a FAC with SSR is to determine the minimum distance between the operations to be implemented, thereby minimizing the need for reconfigurable hardware (and its associated penalties). Certain operations have inherently greater similarities than others, making them more conducive to SSR implementation. For example, adders and multipliers have similar substructures, resulting in an especially efficient flexible implementation.

Other DSP operation combinations may also be considered for FAC implementation: a wide bit width operation could be integrated with multiple operations of narrower width; several low-precision operations could be embedded within a high-precision operation [10]; a rarely used operation could be integrated within a high use operation (increasing hardware usage); and so on. In addition, reconfigurable rounding modes may be added so that the binary point of the output can be moved based on the inputs and desired rounding. This would address the rounding inaccuracy and scaling problems that plague conventional fixed-point components used in hardware signal processing. As stated in Section 1.1, for a FAC to provide area savings, its area must be smaller than the combined area of all of the operations implemented individually.

FACs built using SSR avoid the large performance, area, and power penalties associated with FPGAs and DSP processors. For example, we have presented a flexible component capable of executing a 4-bit fixed-point addition, subtraction, multiplication, or comparison [3]. The areas of the various components normalized in terms of the comparator (which is the smallest of the fixed components at 52 inverter-equivalent gates) were obtained as Comparator: X, Adder/Subtractor: 1.44X, Multiplier: 4.5X, Limited Flexible Unit (LFU; Adder/Subtractor and Multiplier): 4.81X, and Full Flexible Unit (FFU; Comparator, Adder/Subtractor, and Multiplier): 5.31X. The flexible units were therefore of very reasonable size compared to the fixed logic units. To compare these inverter-equivalent gate counts to an FPGA implementation, the largest fixed component (i.e., the multiplier) was considered. The 4-bit multiplier implemented on an FPGA with 4-input LUTs required 82 LUTs (using the Xilinx ISE software package). At an approximate area cost of 80 inverter-equivalent gates for each LUT, this is equivalent to 6560 inverters or 126.15X. This does not include the interconnect network, which consumes the majority of reconfigurable fabric area. Reconfiguring the flexible components is also efficient since it just involves changing the configuration bits for a single 4-input look-up table and select signals for internal MUXes as opposed to reconfiguring large areas of an FPGA. Small-scale reconfigurability clearly provides hardware flexibility with significantly less area than general-purpose FPGAs. Since FPGAs are, in general, more efficient than DSP processors when customized for a particular application, SSR provides greater benefits overall.

While SSR provides area savings, delay penalties must also be considered, as the length of a c-step may increase with flexible components. For the 4-bit components in [3], the relative path delays of the components were Comparator: Y, Adder/Subtractor: 2Y, Multiplier: 3.67Y, LFU: 4.67Y, and FFU: 5Y, with the approximation that all combinational gates have equal delays. Therefore, assuming the length of the c-step was defined by the multiplier delay, the use of an LFU or FFU will increase the c-step length by 27% and 36%, respectively. These increases must be traded off with the area savings provided by hybrid library scheduling and allocation. In addition, these delays are significantly less than those of an FPGA, revealing the delay benefit of SSR.

It must be noted that the specific multiplier chosen for comparison is that with the shortest critical path. For other multiplier structures, the percentage delay increase would be smaller. For example, the results in this paper use an augmentation of the "morphable multiplier" [5], which is capable of implementing both multiplication and addition (in fact, it can perform two data-independent additions in parallel) with the same delay as a fixed logic multiplier. Given the regularity with which MAC operations occur in DSP algorithms, this FAC provides significant benefits for multimode DSP systems. This component and the augmentations performed are discussed in Section 5.

3.2. Flexible controllers

Controllers in application-specific multimode systems are typically designed as a composition of individual controllers, one for each mode. These controllers can be microcoded or hardwired FSMs. Hardwired FSMs are the more common choice given their smaller area and higher performance. However, when one considers designing a single flexible controller capable of being adapted for each mode, microcode has been the traditional option, as different microinstructions can be loaded for each mode. But as with FPGAs, the area, performance, and power costs of microcoded controllers make them significantly less efficient that hardwired FSMs.

SSR can again be used to find the minimum distance between the various mode controllers to implement an efficient flexible hardwired FSM. Consider that an FSM can be defined as a six-tuple $\langle S, I, O, S_0, \delta, \lambda \rangle$, where S is the set of states, I is the input set, O is the output set, S_0 is the initial state, δ is the state transition function set, and λ is the output function set. For an FSM to be made flexible, the δ and λ functions can be implemented using SSR such that they can be configured to implement the various mode controllers (including the control signals setting the FAC functionalities), and the cardinality of S is defined by the maximum S cardinality of the individual mode controllers. Such an implementation is likely to be significantly more efficient than either separate hardware controllers or programmable microcode.

The adaptable FSM is optimally implemented as follows. The set of state transition functions defined by δ is fed to a logic synthesis tool and jointly synthesized as a multiple-output function circuit with logic sharing between the functions. The shared logic in the final synthesized circuit is the logic common to all of the functions and is implemented in fixed logic. The logic specific to each function is implemented in reconfigurable logic (using LUTs and/or MUXes) and changes between contexts. The process is repeated for the λ set.

This process is efficient as long as the intermode functions differ only slightly from each other. The multimode system synthesis techniques detailed in the following section help to minimize these differences, ensuring an efficient implementation. It should be noted that the number of configuration control bits is limited due to the small scale nature of SSR. Therefore, the extra logic needed for generating these bits in the flexible FSMs is small compared to the savings obtained in simplifying the steering logic for the interconnect MUXes.

4. MULTIMODE DSP SYSTEM SYNTHESIS

While the SSR design methodology helps to enable efficient flexible hardware, a multimode DSP system's efficiency is ultimately driven by high-level synthesis, which ensures the flexibility is used efficiently. General multimode synthesis techniques are emerging, but this section presents the first multimode DSP system synthesis technique that incorporates FACs and adaptable FSMs. In fact, it is noted by the designers of the morphable multiplier that no synthesis techniques exist that make use of such components [5].

The steps in Figure 2 detail the synthesis methodology and are explained via the subsequent example.

```
(1) input DFGs and system and DFG-specific constraints;
(2) identify the set of potential arithmetic components;
(3) traverse DFGs for critical paths;
(4) for each DFG (in order of increasing slack) HFDLS (DFG);
    HAL (DFG);
(5) c-step matching (scheduled DFGs);
(6) for each DFG (in order of decreasing resource usage)
    bind (DFG);
(7) design controller.
```

Figure 2: Multimode system synthesis methodology.

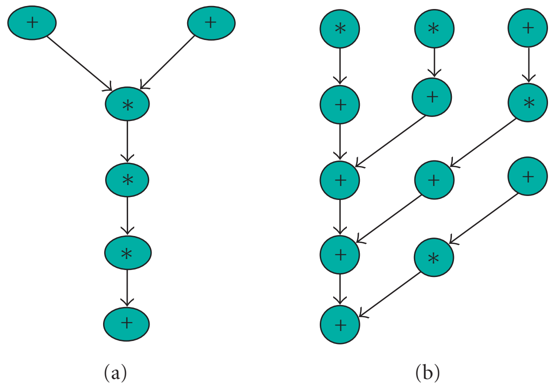

Figure 3: Sample DFGs.

In step 1, the inputs to the system are the individual DFGs, representing the signal processing modes and an optional overall delay and/or resource constraint on the whole system. Parameters associated with each DFG, such as the precision, data width, and maximum tolerable latency (or optionally, the resource constraint), are also input to the system. (The results in Section 5 are for deriving the minimum resource allocation under latency requirements. Future work will address other scenarios.)

Consider the two DFGs in Figure 3. Assume that the latency constraints on DFG 3a (Figure 3(a)) and DFG 3b (Figure 3(b)) are eight c-steps and seven c-steps, respectively. At this time, a preprocessing step, step 2, is performed that identifies the set of arithmetic components that could possibly execute each operation, including satisfying operations of various bit widths, such as implementing a 32-bit addition with two 16-bit adders. For this example, the total resource set includes ADDs, MULTs, and the augmented morphable multipliers. Assume that all operations have the same bit width and that the c-step latencies of an ADD and MULT are one and two, respectively, in both the fixed logic components and the FAC. As stated in Section 3.1, the FAC can also do two data-independent additions in a single c-step. Given that FAC reconfiguration time is minimal due to the SSR implementation, FACs can change functionality within the same mode (i.e., intramode reconfiguration).

The DFGs are then traversed in step 3 to find the critical paths based on data dependencies and the operation latency assumptions. Counting additions as one c-step and multiplies as two, the longest paths on DFG 3a and DFG 3b are eight and six steps, respectively. Therefore, to meet the latency constraints of each DFG, there is no c-step slack for DFG 3a and one c-step for DFG 3b. This determines the DFG scheduling order, with the DFG with the least slack scheduled first.

Therefore, DFG 3a is scheduled first in step 4 using hybrid force-directed list scheduling (HFDLS) and hybrid allocation (HAL) [3]. Unlike the conventional FDLS algorithm that lowers the concurrency of same-type operations per c-step, the HFDLS algorithm uses a modified force calculation such that an overall balance in the number of operations per c-step is achieved. The HFDLS algorithm has the same worst case computational complexity as FDLS: $O(n^2)$, where n is the number of nodes to be scheduled in each DFG. After scheduling, general multifunction ALU allocation algorithms can be used by considering the flexible component as a kind of limited ALU. However, the bulk of the work in creating an optimum schedule requiring the minimum number of flexible components is done by HFDLS, and a simple allocation algorithm, such as HAL, is sufficient to allocate components on these scheduled graphs. HAL uses principles from set theory to produce an exact minimum module allocation set. The algorithm has a computational complexity of $O(m)$, where m is the number of adder and multiplier nodes to be allocated. Further details on both HFDLS and HAL can be found in [3].

Given that DFG 3a has no slack, it must be scheduled in the fewest number of c-steps. The resulting scheduled DFG is shown in Figure 4. From this schedule, it is clear that two ADDs and one MULT would be necessary if only fixed components were available. However, this DFG can be implemented with a single FAC. Given the relative size of the various components, this represents a large reduction in area, even within a single mode. (This echoes the results in [3], which focus on FACs in single-mode systems.)

Once the first DFG is scheduled, the other DFGs are each scheduled (in the order of increasing slack) so that they meet their individual latency requirements. For each DFG, an attempt is made to meet the required latency without more resources than are currently allocated. When the resource allocation must be increased to meet a latency requirement, the global resource allocation is updated. While an increase in the allocation set may enable already scheduled DFGs to reduce latency by rescheduling, the algorithm does not do so, as the DFG latency requirements have already been met. Instead, this resource slack is exploited during binding, as discussed below. When all of the DFGs are scheduled, the final resource allocation set, including both fixed and flexible components, is known.

When DFG 3b is scheduled using this approach, it is obvious that it cannot be scheduled with only one FAC. Therefore, the number of resources must be increased. The minimum resource set to schedule DFG 3b, while meeting its latency requirement, is two FACs, as opposed to 2 ADDs and 2 MULTs if FACs were not available. The resulting scheduled

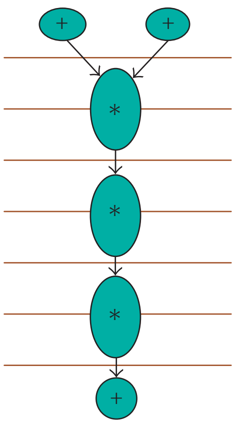

FIGURE 4: Scheduled DFG for DFG 3a.

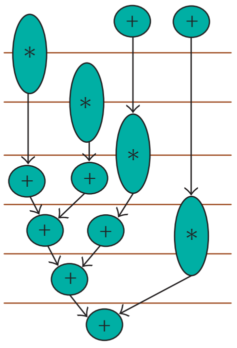

FIGURE 5: Scheduled DFG for DFG 3b.

DFG is shown in Figure 5. (Note that the schedule would be different if only fixed components were available, as HFDLS often produces different schedules than traditional FDLS.) Even though we increased the component allocation, we do not reschedule DFG 3a. (Note that in this case the schedule would not change anyway.)

If the DFGs have different numbers of c-steps, step 5 matches c-steps across the DFGs. The goal is to maximize same-control-step component usage, thus minimizing the functional differences between the various mode controllers for an efficient adaptable FSM design. This matching is done using maximal weighted matching, which can be solved in polynomial time [11]. The weight assigned to each control-step-connecting edge in the matching graph is the number of resources common to the connected c-steps.

In the example here, there are only 2 DFGs, making it a case of bipartite matching. Given that DFGs 3a and 3b have eight and seven c-steps, respectively, there is only a slack of one c-step for the matching process. So c-step 1 in DFG 3b can match with either c-step 1 or 2 in DFG 3a, c-step 2 can match with either c-step 2 or 3, and so forth. Given the balanced resource needs of each c-step, the edges all have equal weight. Therefore, by convention, the same number c-steps are matched across DFGs (1 to 1, 2 to 2, etc.).

Step 6 binds operations to components, starting with the DFG that last set the resource allocation, as it is typically the mode with the highest component utilization. Maximal weighted matching is used to minimize interconnect, MUXes, and registers [13]. An important change to traditional binding is that the graph is constructed with compatibility edges drawn from operation nodes not only to components of that type but also to FACs, albeit with a smaller "component match" weight factor.

In binding the other DFGs, an effort is made to minimize interconnect, MUX, and register overhead above that set by the base DFG as well as to simplify the subsequent controller design. The key benefit provided by FACs is that subgraphs within a DFG with different operations are actually isomorphic in both nodes and edges if the disparate operations can be bound to the same FAC. Therefore, the binding algorithm is likely to find a larger number of subgraphs and individual nodes that share inputs/outputs. These benefits are further enhanced by the resource slack in the DFGs that were not rescheduled after additional resources were allocated.

The binding result for DFGs 3a and 3b are shown in Figure 6. The different block shadings represent the two FACs. Note that the darkly shaded component always outputs to itself, except for c-step 1 in DFG 3b, and the lightly shaded component always feeds the same input port of the other component. This matching helps minimize the interconnect, MUXes, and registers.

The final step is controller design. As discussed in Section 3.2, SSR and multiple output logic synthesis enable the controllers for all of the modes to be implemented in the same physical space, with their similarities implemented in fixed logic and interconnect and their differences in reconfigurable logic. The outputs of the FSM include the control signals for the MUXes, register enables, and FAC settings.

None of these steps in this process are more computationally complex than what is currently done for multimode system synthesis, but as the results show in the following section, the area and power savings can be significant.

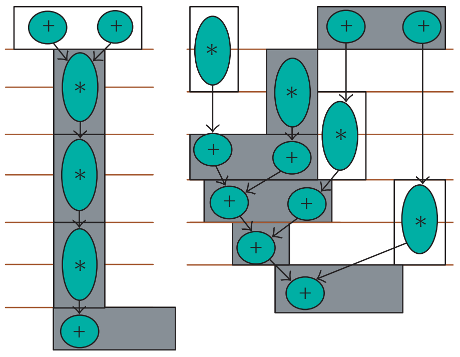

Figure 6: Bounded DFGs.

5. RESULTS

The methodology presented in this paper has been evaluated by synthesizing multimode DSP systems with runtime reconfiguration. The base DFGs used are well-known DSP instances from the high-level synthesis literature. The parameters of these DFGs, in terms of number of nodes, minimum latency, and so forth, span a wide range and are representative of the kind of DFGs that occur in multimode systems. ELLIP is a fifth-order elliptic digital filter with 33 operations and minimum latency of 13 c-steps [12], EDGE is an edge detector with 241 operations and minimum latency of 121 c-steps [13], ARFILT is an autoregressive filter with 28 operations and minimum latency of 8 c-steps [14], FDCT is a fast discrete Fourier transform instance with 42 operations and minimum latency of 6 c-steps [15], and FIRFILT is a 16-point FIR filter with 23 operations and minimum latency of 9 c-steps [16].

The datapath is assumed to be 32-bit wide in all of the example systems. The component library consists of 32-bit fixed logic adders, multipliers, and FACs capable of both addition and multiplication (including two data-independent ADDs in one c-step). Other FACs (including those not specific to DSP applications) will be considered as part of future work. As discussed in Section 3.1, the FAC is based on the morphable multiplier [5]. The component areas in terms of the number of constituent transistors are as follows: ADD = 1306, MULT = 6150, FAC = 6860. The multiplier and multiply-configured FAC have latencies of two c-steps. Pipelined versions of these components can be built at a cost of approximately 1000 additional transistors each, which would be necessary if the required maximum latency was less than what non-pipelined components allowed. Reconfiguring this component simply involves sending the appropriate select signal to internal MUXes and is virtually instantaneous.

The component allocations and resource utilizations for various DFG combinations are shown in Table 1. The DFGs were synthesized for minimum area under the imposed latency constraint. The first three columns (SMFixed: single-mode fixed) show the area for fully implementing each DFG individually with fixed components. The next three columns (MMFixed: multimode fixed) show the synthesis results for a fixed logic multimode DSP system. Both SMFixed and MMFixed were obtained using conventional FDLS and allocation [1]. Finally, the synthesis results for a flexible DSP system with intramode reconfiguration using the method presented here are shown (MMFlex: multimode flexible), including the datapath area savings and resource utilization increase over the fixed logic multimode DSP system. We have shown in [3] that performance gains in flexible single-mode systems are attributable to both the use of FACs and the synthesis algorithm. The modified synthesis and allocation procedures presented here are therefore necessary to fully utilize the benefits provided by FACs and flexible controllers for multimode systems as well. It is important to note that domain-specific synthesis and conventional multifunction ALU allocation techniques could possibly be adapted to produce similar, but intermode reconfiguration only, systems. However, the intramode reconfiguration enabled by the technique presented here contributes a substantial portion of the benefit over conventional fixed multimode DSP systems.

Table 1: Component allocations and utilizations.

MM system	SMFixed				MMFixed				MMFlex					Improvement (%)	
	+	*	Area	Util. (%)	+	*	Area	Util. (%)	+	*	FAC	Area	Util. (%)	Area	Util. (%)
AR, FDCT, FIR	7	14	95 242	17.9%	4	8	54 424	37.5%	1	6	2	51 926	44.7%	4.59%	19.2%
ELLIP, FIR	5	4	31 130	29.1%	3	2	16 218	52.3%	1	0	2	15 026	73.2%	7.35%	40.0%
ARFILT, FDCT	6	12	81 636	29.7%	4	8	54 424	44.6%	0	6	2	50 620	56.2%	6.99%	26.0%
FIR, EDGE	4	4	29 824	36.1%	3	2	16 218	57.8%	1	0	2	15 026	90.1%	7.35%	55.9%
ELLIP, EDGE	4	4	29 824	33.6%	3	2	16 218	53.7%	1	0	2	15 026	81.5%	7.35%	51.8%

Table 2: Registers, MUXes, and control signals.

MM system	MMFixed			MMFlex			Improvement (%)		
	Reg	2 : 1 MUX	Ctrl	Reg	2 : 1 MUX	Ctrl	Reg	2 : 1 MUX	Ctrl
AR, FDCT, FIR	18	83	249	14	51	53	22.22%	38.55%	78.71%
ELLIP, FIR	11	42	84	8	23	25	27.27%	45.24%	70.24%
ARFILT, FDCT	14	69	138	12	45	47	14.29%	34.78%	65.94%
FIR, EDGE	11	29	58	8	22	24	27.27%	24.14%	58.62%
ELLIP, EDGE	7	27	54	4	16	18	42.86%	40.74%	66.67%

Table 3: Controller logic areas.

MM system	MMFixed	MMFlex	Improvement
	Controller area	Controller area	(%)
AR, FDCT, FIR	1094	709	35.19%
ELLIP, FIR	416	263	36.78%
AR, FDCT	506	354	30.04%
FIR, EDGE	294	133	54.76%
ELLIP, EDGE	266	202	24.06%

The increased resource utilization also results in less wasted power consumption. For all three implementations, turning off components that are completely unused in a mode will help reduce power, but the overhead of turning components on/off prevents intramode component shut down.

While these datapath area savings are significant, Table 2 shows that even larger savings are provided in terms of registers, MUXes, and control signals. The register and 2 : 1 MUX reductions, due primarily to the binding process, are especially valuable, as the 32-bit wide datapath makes these components large. The ~70% reduction in the number of control lines (ctrl) to the datapath from the controller, which among other things helps to simplify placement and routing, is obtained as a result of both the binding process as well as the use of adaptable controllers.

The controller logic area is also reduced, as shown in Table 3. Since separate controllers need not be built, the logic is greatly simplified. The area numbers shown are in terms of inverter-equivalent gates and are only for the combinational portion of the controller that implements the output functions, including any LUTs in the case of the adaptable controller. The flip-flops and other memory elements in both the fixed controller and the adaptable controller are the same and are hence not included in the area results.

6. CONCLUSIONS

This paper presented an approach for synthesizing multimode DSP systems with a hybrid library of fixed and flexible arithmetic components and adaptable controllers. The implementation capabilities and efficiency of the multimode system platform is greatly increased by the extra hardware flexibility provided by small-scale reconfigurability, without the large area, performance, and power penalties associated with general-purpose reconfigurable fabric. The intramode reconfiguration and the scheduling, allocating, and binding flexibility provided by the FACs result in significant datapath and control area savings and wasted power consumption reduction over existing multimode DSP system synthesis techniques.

7. ACKNOWLEDGMENTS

This work is supported in part by the National Science Foundation under Grant numbers CCR-0105626 and EHS-0410526 and by the Woodrow W. Everett, Jr. SCEEE Development Fund in cooperation with the Southeastern Association of Electrical Engineering Department Heads.

REFERENCES

[1] L.-Y. Chiou, S. Bhunia, and K. Roy, "Synthesis of application-specific highly-efficient multi-mode systems for low-power applications," in *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE '03)*, pp. 96–101, Munich, Germany, March 2003.

[2] A. van der Werf, M. J. H. Peek, E. H. L. Aarts, J. L. Van Meerbergen, P. E. R. Lippens, and W. F. J. Verhaegh, "Area optimization of multi-functional processing units," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design (ICCAD '92)*, pp. 292–299, Santa Clara, Calif, USA, November 1992.

[3] V. Vijay Kumar and J. Lach, "Designing, scheduling, and allocating flexible arithmetic components," in *Proceedings of 13th International Conference on Field Programmable Logic and Applications (FPL '03)*, pp. 1166–1169, Lisbon, Portugal, September 2003.

[4] V. Vijay Kumar and J. Lach, "Heterogeneous redundancy for fault and defect tolerance with complexity independent area overhead," in *Proceedings of 18th IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT '03)*, pp. 571–578, Boston, Mass, USA, November 2003.

[5] S. M. S. A. Chiricescu, M. A. Schuette, R. Glinton, and H. Schmit, "Morphable multipliers," in *Proceedings of 12th International Conference on Field Programmable Logic and Applications (FLP '02)*, pp. 647–656, Montpellier, France, September 2002.

[6] K. Compton and S. Hauck, "Flexibility measurement of domain-specific reconfigurable hardware," in *Proceedings of ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays (FPGA '04)*, pp. 155–161, Monterey, Calif, USA, February 2004.

[7] K. Kim, R. Karri, and M. Potkonjak, "Synthesis of application specific programmable processors," in *Proceedings of ACM/IEEE 34th Design Automation Conference (DAC '97)*, pp. 353–358, Anaheim, Calif, USA, June 1997.

[8] L. M. Guerra, M. Potkonjak, and J. M. Rabaey, "Behavioral-level synthesis of heterogeneous BISR reconfigurable ASIC's," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 1, pp. 158–167, 1998.

[9] E. Bozorgzadeh, S. O. Memik, R. Kastner, and M. Sarrafzadeh, "Pattern selection: customized block allocation for domain-specific programmable systems," in *Proceedings of International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA '02)*, Las Vegas, Nev, USA, June 2002.

[10] G. Even, S. M. Mueller, and P.-M. Seidel, "A dual mode IEEE multiplier," in *Proceedings of 2nd Annual IEEE International Conference on Innovative Systems in Silicon (ISIS '97)*, pp. 282–289, Austin, Tex, USA, October 1997.

[11] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASIC's," *IEEE Transactions on Computer-Aided design of Integrated Circuits and Systems*, vol. 8, no. 6, pp. 661–679, 1989.

[12] S. Park and K. Choi, "Performance-driven high-level synthesis with bit-level chaining and clock selection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 2, pp. 199–212, 2001.

[13] R. M. Haralick and L. G. Shapiro, *Computer and Robot Vision*, Addison-Wesley, Reading, Mass, USA, 1992.

[14] K. Högstedt and A. Orailoglu, "Integrating binding constraints in the synthesis of area-efficient self-recovering microarchitectures," in *Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD '94)*, pp. 331–334, Cambridge, Mass, USA, October 1994.

[15] D. J. Mallon and P. B. Denyer, "A new approach to pipeline optimisation," in *Proceedings of European Design Automation Conference (EDAC '90)*, pp. 83–88, Glasgow, Scotland, UK, March 1990.

[16] R. Karri and A. Orailoglu, "High-level synthesis of fault-secure microarchitectures," in *Proceedings of 30th ACM/IEEE International Conference on Design Automation (DAC '93)*, pp. 429–433, Dallas, Tex, USA, June 1993.

Vinu Vijay Kumar is a Design Engineer in the DSP Design Division of Texas Instruments. His research interests include reconfigurable systems, design methodologies for DSP system synthesis, and physical design of SoC systems. He received the B.E. degree from PSG College of Technology, India, in 2000, and the M.S. degree in electrical engineering and the Ph.D. degree in computer engineering from the University of Virginia, Charlottesville, Va, USA, in 2002 and 2005, respectively. He is a Member of the IEEE and Eta Kappa Nu.

John Lach received the B.S. degree from Stanford University, USA, in 1996, and the M.S. and Ph.D. degrees in electrical engineering from the University of California, Los Angeles, USA, in 1998 and 2000, respectively. Since 2000, he has been an Assistant Professor in the Charles L. Brown Department of Electrical and Computer Engineering at the University of Virginia, Charlottesville, Va, USA. His primary research interests include dynamically adaptable and real-time embedded systems, computer-aided design techniques for very-large-scale integration, general-purpose and application-specific processor designs, and wearable technologies for aged independence. He is a Senior Member of the IEEE, and a Member of the ACM, IEEE Computer Society, IEEE Circuits and Systems Society, ACM SIGDA, and Eta Kappa Nu.