Platform-Based Resource Binding Using a Distributed Register-File Microarchitecture

Jason Cong, Yiping Fan, Wei Jiang
Computer Science Department
University of California, Los Angeles, CA 90095, USA
{cong, fanyp, wjiang}@cs.ucla.edu

ABSTRACT

Behavior synthesis and optimization beyond the register transfer level require an efficient utilization of the underlying platform features. This paper presents a platform-based resource-binding approach using a *distributed register-file microarchitecture (DRFM)* that makes efficient use of distributed embedded memory blocks as register files in modern FPGAs. A DRFM contains multiple islands, each having a local register file, a functional unit pool and data-routing logic. Compared with the traditional discrete-register counterpart, a DRFM allows use of the platform-featured on-chip memory or register-file IP blocks to implement its local register files, and this results in substantial saving of multiplexing logic and global interconnects. DRFM provides a useful architectural template and a direct optimization objective for minimizing inter-island connections for synthesis algorithms. Based on DRFM, we propose a novel binding algorithm focusing on the minimization of the inter-island connections. By applying our approach, significant reductions on multiplexors and global-interconnections are observed. On the Xilinx Virtex II FPGA platform, our experimental results show a 2X logic area reduction and a 7.8% performance improvement, compared with the traditional discrete-register-based approach.

Categories and Subject Descriptors

B.5.2 [**Hardware**]: Design Aids—*automatic synthesis*

General Terms

Algorithms, Design, Experimentation

Keywords

Behavior Synthesis, resource binding, distributed register file

1. INTRODUCTION

With the advancement of the integrated circuit technology, interconnects have an increasingly large impact on the quality of results (QoR). The shrinking cycle time—combined with the growing resistance-capacitance delay, die size, and average interconnect length—result in the increasing ratio of interconnect delay, especially global interconnect delay, which does not scale well with feature size. The area and power of interconnects have by far outweighed the area and power of functional units and registers. A recent work from Intel [30] shows that interconnects consume around 51% of the total dynamic power of the microprocessors in a 0.13um technology, and projects that interconnects can consume up to 80% of the dynamic power in future technologies. For field-programmable gate arrays (FPGAs), studies show that interconnects contribute 70 to 80% of the total area [37] and 75 to 85% of the total power [25]. Multiplexors, which are collections of interconnects without actual computational functionality, except for data routing, are particularly expensive for FPGA platforms. It is shown that the area, delay and power data of a 32-to-1 multiplexor are almost equivalent to an 18-bit multiplier in 100*nm* technology in FPGA designs [3].

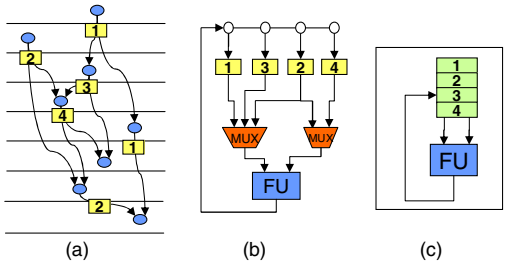

Figure 1: Advantages of register file over discrete registers. (a) A scheduled data-flow graph with register binding indicated on each variable; (b) Binding using discrete registers; (c) Binding using a register file.

At the register-transfer level, a multiplexor is required when multiple data sources feed into a single port at multiple control steps (c-steps). As shown in Fig. 1(a), the behavior of a design is represented in a scheduled data-flow graph (DFG). After resource binding using discrete registers, a datapath is generated, as shown in Fig. 1(b), where two multiplexors are needed to route the data flows at different c-steps. This is indeed the optimal datapath using discrete registers, if one functional unit is the hard resource constraint. The datapath can be improved if a register-file microarchitecture is applied, as shown in Fig. 1(c), where there is no multiplexor required at all. In fact, we can view that the multiplexors in the first datapath are absorbed and replaced by the dedicated decoder of the 1-write, 2-read-port register file in the second datapath.

However, due to the limitation of numbers of the read and write ports, a centralized register file may not work for highly parallelized applications which require multiple simultaneous data reads

ICCAD'06 November 5-9, 2006, San Jose, CA

Table 1: On-chip RAM Blocks on Virtex II and Stratix FPGAs

Xilinx XC-2V	2000	3000	4000	6000	8000
#18Kb BRAM	56	96	120	144	168
Dist. RAM(Kb)	336	448	720	1,056	1,456
Altera EP1	**S25**	**S30**	**S40**	**S60**	**S80**
#M512(512b)	224	295	384	574	767
#M4K(4Kb)	138	171	183	292	364
#M-(512Kb)	2	4	4	6	9

and writes. The port numbers of register files are limited because the implementation cost of a register file is very sensitive to its port number. As pointed in [36], the area and power consumption of a register file grows *cubically* with its port number. Advanced FPGA devices, such as Virtex IV [15] and Stratix II [14], are not able to implement register files with more than two write ports in their on-chip memory blocks. Suppose the DFG in Fig. 1 were duplicated three times horizontally; a 3-write, 6-read port register file would then be required, which is very expensive if not impossible to implement. Apparently, a distributed 3-register-file datapath would be more efficient in this case.

The use of distributed register files is further encouraged on the platforms with rich on-chip memory or register-file IP blocks. A key issue in platform-based design methodology is how to sufficiently use the resources (or services) provided by the underlying platform or technology [20]. In designs for modern system-on-a-chip (SoC) or field-programmable SoC, on-chip memories can be instantiated as pre-optimized IP modules. The resultant area and performance benefits are remarkable. For example, in Xilinx Virtex II and Altera Stratix devices, memory IP blocks are abundantly distributed on the chips, so that the implementation of register files on them is “free" if they are not used for other data storage and the resource capacity bound is not exceeded. TABLE 1 shows data related to memory blocks on Virtex II [15] and Stratix [14]. Since we know that the implementation of multiplexors on FPGAs is very expensive [3] and register files are able to reduce the multiplexor use on such platforms, it is not surprising to see dramatic improvement in area and performance when using on-chip memories to implement distributed register files.

This paper addresses the problem of full utilization of register files during behavior synthesis. In particular, the contributions of this paper are as follows:

i) A distributed register-file microarchitecture (DRFM) is presented as a flexible synthesis template. DRFM contains multiple islands, each having a register file, a functional unit pool and data-routing logic. DRFM will be particularly beneficial for reducing the interconnect complexity in FPGA designs.

ii) The properties of DRFM are investigated, and an explicit optimization goal, the total number of *inter-island connections*, is proposed for minimizing interconnect and multiplexor complexity.

iii) A resource binding algorithm is proposed that targets DRFMs to minimize the inter-island connections directly.

The organization of the paper is as follows. After the discussion on related work in Section 2, the DRFM concept is presented in Section 3. Following the preliminaries and problem formulation in Section 4, the DRFM binding algorithm is discussed in Section 5. Section 6 discusses how to extend our approach to handle generalized cases. Experimental results are presented in Section 7, followed by conclusions in Section 8.

2. RELATED WORK

There is extensive literature on general binding algorithms in high-level synthesis [8, 11, 38]. The previous work can be roughly categorized into two major groups. The first group performs simultaneous functional unit and register binding. Representative algorithms include simulated annealing [5, 9, 24], simulated evolution [29], and integer linear programming (ILP) [13, 35]. Since the subtasks of behavioral synthesis are highly interrelated, simultaneous optimization approaches try to consider all the involved optimization parameters together for globally better results. However, the major concern for these algorithms is their scalability towards optimizing large designs. The second group solves register and functional unit binding separately. Representative algorithms include clique partitioning [39], weighted bipartite-matching [16], and network flow [22, 2, 12]. These algorithms may achieve very good results for the single task at hand, but it is unclear how much gain can be relayed to the final product.

The increasing interconnect effect encourages the research on architectures that exploit the physical locality by operating on data close to where it is stored. Examples of this are the multicluster architecture [10] and the multicomputer processor-DRAM chip model [7]. In behavior synthesis research, [18] and [21] proposed a distributed-register architecture, where registers are distributed so that each functional unit can perform a computation by reading/writing data from/to the local dedicated registers. Data transfers between different functional units are regarded as global communications that may take multiple cyclesFurther improvement is shown in [6], which presents a Regular Distributed Register (RDR) microarchitecture and an architectural synthesis methodology, with the emphasis on multicycle on-chip communication for synchronous designs. However, these microarchitectures do not use register files specifically or the on-chip embedded memories for register-file implementation.

There are previous research projects related to register file architecture in behavior synthesis. The developers of the Hyper system [34] proposed to use a register file to replace the cluster of discrete registers driving each multiplexor (if feasible). However, the register files are introduced only during the post-process after traditional binding is accomplished, and the authors did not have a register-file-based microarchitecture in mind before this step. Since a good interconnect structure using discrete registers is not necessarily good for a register-file-based microarchitecture, opportunities for optimizing interconnects and multiplexors may be lost in this approach. A simple example is illustrated in Fig. 2. Suppose we are provided with only 1-write-port register files, binding solution (c) uses a register file derived from the traditional binding (a,b), and reduces the 3-to-1 multiplexor to a 2-to-1 multiplexor. Note that registers 1 and 2 cannot be grouped into a register file since they have a “write" competition at a control step, as do registers 2 and 3. The more aggressive binding solution, Fig. 2(d,e), uses two distributed register files by taking our approach (discussed in later sections), and eliminates one multiplexor. Note that solution (d) introduces a new register element 4 which replaces register 1 at control step 3 in Fig. 2(a).

There has been research that focuses on the synthesis for minimizing register file (or memory module) numbers or port numbers [28, 27]. Many distributed-storage approaches were proposed to overcome the drawbacks of centralized storage organizations in the architectural synthesis domain. A sequencer-based architecture is proposed in [1], where a sequencer is either a stack or queue. A data routing approach in [26] tries to find better ways to transfer data among datapath components to reduce interconnect complexity. A distributed VLIW architecture is discussed in [17], and applications are mapped onto this architecture by fully exploring the temporal and spacial locality of both computations and communi-

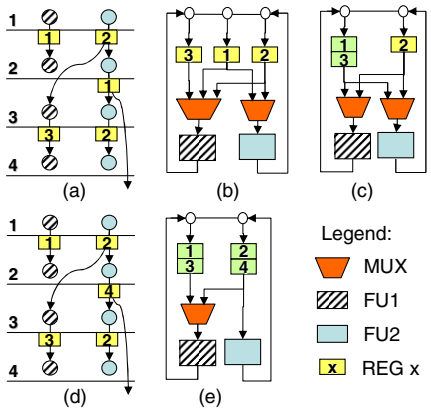

Figure 2: Three binding solutions for the same scheduled DFG. (a) A scheduled DFG marked with binding information obtained from a traditional binding approach; (b) The discrete-register-based binding according to (a); (c) The binding using a register file derived from (b); (d) The scheduled DFG marked with binding information obtained from our distributed-register-file-based approach; (e) The binding derived from (d).

cations. These approaches focus on scheduling techniques, while none of them focus on the resource binding stage. In [23], the authors applied interconnect minimization techniques during variable allocation (after operation binding) for datapaths with multi-port memory modules. Our approach differs from [23] in that we consider the bindings for operations and variables together in a unified way, targeting an island-based microarchitecture template.

3. DISTRIBUTED REGISTER-FILE MICROARCHITECTURE

The essential insight behind many approaches discussed in Section 2 is that communication should be localized as much as possible to minimize the interconnect effect. With a similar insight in mind, we present a distributed register-file microarchitecture (DRFM) for resource binding in behavior synthesis.

Fig. 3 presents one of the multiple computational islands of this microarchitecture. Each island contains a *local register file (LRF)*, a *functional unit pool (FUP)*, and *data-routing logic*. The LRF plays a key role in an island, since it is used to store the value produced from the internal FUP of the island. The LRF also provides data to the FUPs in this and the external islands. Data-routing logic is, as implied by its name, used for routing data from external islands. The multiplexors on the front of the FUP may be used to select correct data, either from the LRF or data-routing logic, at each control step. Note that these multiplexors if used, are usually much smaller compared with those in the datapath using discrete registers. Hereafter, let $\mathbb{M} = \{I_1, ..., I_K\}$ denote a DRFM instance with K islands. Suppose I is an island, we use $LRF(I)$ and $FUP(I)$ to represent its LRF and FUP, respectively.

For simplicity, we will first present the ideal DRFM configuration, where each LRF is restricted to only *1-write-port* but there is no restriction on the read-port number, and no data replication is allowed, i.e., a variable can only be stored in one register element of a fixed register file during its lifetime. Generalized cases will be discussed in Section 6.

DRFM provides many advantages in behavior synthesis. First, it is a semi-regular microarchitecture template. Although it has

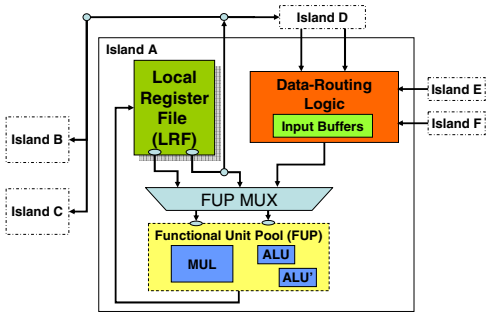

Figure 3: Illustration of an island in a distributed register-file microarchitecture (DRFM) instance.

a write-port restriction on each LRF, it provides much more flexibility than the traditional VLIW and DSP architectures because DRFM has no restriction on data-routing structures and other configurations of the LRF and FUP. The size and configuration of the islands may be unbalanced to fit the application behavior. The flexible configurations should be determined by the applications and synthesis algorithms. Second, DRFM provides a template and explicit optimization goals for synthesis algorithms. Conceptually it encourages a binding of computations with their related data closely within one island, and allows synthesis algorithms to focus on global inter-island communications. In particular, the data-routing logic should be optimized by any synthesis algorithm targeting DRFMs. Last, modern FPGA platforms should be very efficient for implementing DRFMs, given their rich on-chip memory resource.

4. PRELIMINARIES AND PROBLEM FORMULATION

The behavioral kernels of an application to be synthesized are represented as *data-flow graphs (DFGs)*. A DFG is a directed acyclic graph (DAG), $G(V,E)$, where every node represents a computational operation, such as an addition or a multiplication, and every directed edge (u,v) represents a *dataflow* produced by operation u and consumed by v. In a *scheduled DFG*, every operation is assigned into a *control step (c-step)*. Hereafter, without explicit mention, we assume **simplified DFGs**, where each operation takes exactly one c-step and produces exactly one output variable, and where all the nodes are functionally compatible. (Extensions to general cases are straightforward and omitted in the paper.) We use the **same notation for an operation and the variable it produces**.

DEFINITION 1. *In a scheduled DFG G, let $T(v)$ denote the c-step where v is scheduled. Operation u is* compatible to *v, represented as $u \lessdot v$, if $T(u) < T(v)$. The $\lessdot$ relation is a partial order. The* compatibility graph *with respect to scheduled DFG $G(V,E)$ is denoted as $G_c(V,E_c)$, where E_c is called the* compatibility-edge set, *and $E_c = \{(u,v) | u \lessdot v, \forall u,v \in V\}$.*

In the scheduled DFG of Fig. 4, there are compatibility edges $(v_1, v_2$, and $(v_7, v_{10}$, etc., which are not explicitly drawn.

DEFINITION 2. *Operations u and v are* incompatible, *denoted as $u \leftrightarrow v$, if there is no compatibility edge (u,v) or (v,u) in G_c. Obviously, $u \leftrightarrow v$ if and only if $T(u) = T(v)$.*

DEFINITION 3. *The* lifetime *of variable u, denoted as $\mathcal{L}(u)$, is defined as the c-step interval from its generation to its last consumption. Variables u and v are* lifetime-conflicting, *denoted as $u \bowtie v$, if and only if their lifetimes overlap.*

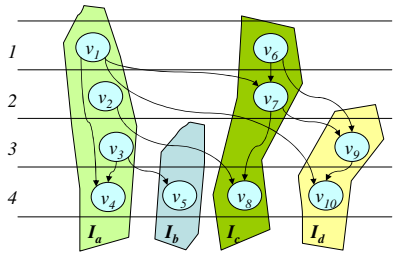

Figure 4: A scheduled data-flow graph and four node-disjoint chains (compatibility edges are not drawn explicitly).

In the scheduled DFG of Fig. 4, $\mathcal{L}(v_1) = [2,4]$, $\mathcal{L}(v_6) = [2,3]$, and $\mathcal{L}(v_9) = [4,4]$. Obviously, $v_1 \bowtie v_6$ and $v_1 \bowtie v_9$.

In a valid *resource binding* of a scheduled DFG G, each operation is assigned to a functional unit, and each variable is assigned to a register. Two operations cannot share one functional unit if they are incompatible. Two variables cannot share one register if they are lifetime-conflicting. In the DFG of Fig. 4, operations v_1 and v_6 cannot share a functional unit, while variables v_6 and v_9 may share a register to hold their value since their lifetimes are disjointed.

A valid binding defines a complete datapath, including the multiplexors required to connect the functional units and registers. In addition to the numbers of the functional units and registers, the multiplexor structures usually impact the quality of the result dramatically.

4.1 DRFM Binding and Its Properties

Using the definition of DRFM in Section 3, within island I, a local operation issued in functional unit pool $FUP(I)$ always writes its output variable into local register file $LRF(I)$, which has only one write port. Therefore, in a valid *resource binding of G onto DRFM* $\mathbb{M}$, if operation v is assigned to $FUP(I)$, then its result variable must be stored into $LRF(I)$ at c-step $T(v)$, and any other operation cannot write data into $LRF(I)$ at $T(v)$. Noting this, and if we ignore the detailed way in which variables are allocated and addressed within a register file, we have the following definition.

DEFINITION 4. *A resource binding of scheduled DFG $G(V,E)$ on DRFM $\mathbb{M}$, denoted as $\mathcal{B}(G,\mathbb{M})$, is a function $\mathcal{B}: V \rightarrow \mathbb{M}$. $\mathcal{B}$ is* feasible *if and only if $u \leftrightarrow v$ implies $\mathcal{B}(u) \neq \mathcal{B}(v)$, $\forall u, v \in V$.*

Definition 4 is simpler than the traditional definition of discrete-register-based binding because it unifies the binding solution for both operations and variables. It is a hint leading to a cleaner problem formulation on DRFM binding.

PROBLEM 1. ***General DRFM Binding Problem***. *Given scheduled DFG G and DRFM $\mathbb{M}$, find a feasible resource binding $\mathcal{B}(G,\mathbb{M})$, so that its quality is optimized.*

However, in practice, the *quality* of a binding $\mathcal{B}(G,\mathbb{M})$, i.e., the resulting DRFM configuration, is determined by several interrelated factors, such as the number of islands, the size of each LRF, and the organizations of the FUPs and the data-routing logic. Furthermore, the cost of a DRFM configuration is platform dependent, since different technologies have very different costs for implementing the same DRFM instance. Therefore, without additional constraints given, Problem 1 is vague and hard for optimization.

Checking the properties of a DRFM binding, we expect a relaxed but more concrete formulation. Using Definition 4, it is easy to show that for any feasible $\mathcal{B}(G,\mathbb{M})$, the operations bound in the same island must be in a chain in G_c. Since each island has only one write port in its LRF, and each operation exactly takes one c-step and produces one variable, the operations bound in the same FUP must be scheduled into different c-steps. If the operations are sorted according to their c-steps, the compatibility edges among adjacent operations will form a *chain* in G_c.

Hereafter, for binding solution $\mathcal{B}(G,\mathbb{M})$, we will not distinguish an island and its associated chain in G_c; i.e., $I = \mathcal{B}(v)$ represents both the island to which v is bound and the chain in G_c that contains v. For the example in Fig. 4, chain $I_a = \{v_1, v_2, v_3, v_4\}$ is bound into an island, as is chain $I_c = \{v_6, v_7, v_8\}$. It is obvious that in this example, four islands are required to produce a feasible binding, since there are at least four chains in this scheduled DFG. This is also indicated by the following property.

THEOREM 1. *$\mathcal{B}(G,\mathbb{M})$ will not be feasible if the number of the islands in $\mathbb{M}$ is less than the minimum number of node-disjoint chains in G_c.*

4.2 Inter-Island Connections

DEFINITION 5. *In a feasible DRFM binding solution $\mathcal{B}(G,\mathbb{M})$, for dataflow (u,v), if $\mathcal{B}(u) \neq \mathcal{B}(v)$, the dataflow is an* inter-chain *or* global *dataflow; otherwise it is a* intra-chain *or* local *dataflow.*

Let us suppose that in Fig. 4 the binding solution corresponds to the four shaded chains (islands), $\{I_a, I_b, I_c, I_d\}$; the dataflow edges (v_1, v_4), (v_6, v_7), etc., are intra-chain dataflows; and (v_1, v_{10}), (v_6, v_9), etc., are inter-chain dataflows.

Intuitively, the local dataflows within a chain are carried through local physical connections between the LRF and FUP, while inter-chain dataflows have to be carried by global inter-island connections. Since DRFM assumes point-to-point inter-island connections, two dataflows can share a global connection, if and only if they are produced from a common chain and consumed in another common chain at different c-steps. In the same example of Fig. 4, dataflows (v_1, v_7) and (v_2, v_8) may share a global connection between island I_a and I_c. In contrast, dataflows (v_6, v_9) and (v_7, v_9) must use two different global connections between I_c and I_d since they are consumed at the same c-step.

DEFINITION 6. *In a DRFM binding $\mathcal{B}(G,\mathbb{M})$, between islands I_i and I_j, a connection can be* shared *by two dataflows if their consumer operations are different and compatible. The number of* inter-island connections (IICs) *between islands I_i and I_j, denoted as $IIC_{\mathcal{B}}(I_i, I_j)$, is the minimum number of connections required to carry the dataflows between the islands (chains) under the sharing conditions. The total number of inter-island connections of $\mathcal{B}(G,\mathbb{M})$ is defined as $IIC(\mathcal{B}) = \sum_{\forall I_i, I_j \in \mathbb{M}, I_i \neq I_j} IIC_{\mathcal{B}}(I_i, I_j)$.*

Using this definition, it is not difficult to see that in Fig. 4, $IIC = 5$, given that dataflows (v_1, v_7) and (v_2, v_8) share an inter-island connection, while (v_6, v_9) and (v_7, v_9) cannot share one. In general, the incompatibility-relations among the operations within one island are sparse, and the *IIC* numbers can be computed fairly fast.

The inter-island connections are critical to the final DRFM quality (also shown in Section 7.1), since for any feasible $\mathcal{B}(G,\mathbb{M})$, the input-port number of island I is equal to the number of inter-island connections feeding into I. For example, island A in Fig. 3 has four input ports because there are four global connections feeding into it. This fact implies that the complexity of the data-routing logic is determined by the inter-island connections, and it suggests the following relaxed problem formulation.

PROBLEM 2. ***DRFM Binding for Minimum Inter-Island Connections***. *Given scheduled DFG $G(V,E)$ and DRFM $\mathbb{M}$, find a feasible resource binding $\mathcal{B}(G,\mathbb{M})$ so that $IIC(\mathcal{B})$ is minimized.*

5. AN ITERATIVE DRFM BINDING ALGORITHM

However, Problem 2 is still not easier to solve than the traditional binding problem for connectivity optimization [32]. It is not difficult to show that a port-assignment problem of three-port memories (NP-complete problem PA3U in [31]) can be reduced to a restricted version of Problem 2. The detailed proof is omitted in this paper, and the readers are referred to [31] and [32] for similar proofs. In addition, the global connections among DRFM islands may be shared by multiple dataflow edges (see Section 4.2); and this property makes Problem 2 different from the classic graph partitioning problem, where the edges are static.

We apply an iterative control-step-by-control-step heuristic approach to solving Problem 2. Each iteration contains two phases: horizontal bipartite-matching-based assignment and vertical local-search-based refinement. The algorithm assumes that the island number K is given for a DRFM configuration $\mathbb{M} = \{I_1, ..., I_K\}$. In practice, the island number may be provided by designers as a constraint, or by an aggressive search for an "optimal" number by calling the algorithm multiple times. Each iteration of the algorithm takes in the current partial binding solution $\mathcal{B}'(G, \mathbb{M})$ and constructs an updated solution. The algorithm terminates when a complete binding is obtained.

5.1 Horizontal Assignment

In the horizontal phase, we apply a minimum-weighted bipartite matching algorithm to obtain a reasonable initial solution for the second phase. A similar idea was presented in [16] for general datapath allocation. Each horizontal phase considers the set of operations Θ within one c-step. Since the operations in Θ are pairwise incompatible, they must be assigned onto different islands.

Given Θ and the current partial binding $\mathcal{B}'(G, \mathbb{M})$, we construct a weighted bipartite graph $G_{bp}(V_\Theta \bigcup V_\mathbb{M}, V_\Theta \times V_\mathbb{M})$ as follows:

Step 1. For each operation v there is a node $n(v) \in V_\Theta$, and for each island I there is a node $m(I) \in V_\mathbb{M}$.

Step 2. For each possible assignment of v to island I, build an edge $(n(v), m(I)) \in V_\Theta \times V_\mathbb{M}$.

Step 3. Assign weight for each edge $(n(v), m(I))$. The weight is the number of the *new* inter-island connections introduced by the assignment of v to I.

A minimum-weighted bipartite matching $E_{match} \subseteq V_\Theta \times V_\mathbb{M}$ for G_{bp} can be computed optimally in $O(|V_\mathbb{M}|^3)$ [33]. For each edge in $(n(v), m(I)) \in E_{match}$, we bind operation v to island I and update $\mathcal{B}'$ accordingly.

Obviously, any matching E_{match} of bipartite graph G_{bp} corresponds to a feasible binding of Θ to $\mathbb{M}$, and the total weight of E_{match} equals the cost of the attempted binding, or the number of newly introduced inter-island connections. Therefore, the updated binding solution produced by the minimum-weighted bipartite matching is optimal among all the possible bindings of Θ to $\mathbb{M}$, as in the following conclusion.

THEOREM 2. *The binding of Θ to $\mathbb{M}$ produced by the above algorithm introduces a minimum number of new inter-island connections to the current partial DRFM-binding solution.*

However, the matching algorithm performs the binding in an "horizontal" fashion in the scheduled DFG, and cannot predict its impact on the future iterations.

5.2 Vertical Refinement

At the second phase of each iteration, we apply a vertical local-search-based refinement for the current partial solution. The refinement process uses an idea similar to the Kernighan-Lin algorithm [19], despite the fundamental difference between our problem and the classic graph-partitioning problem. It reassigns an operation to a different chain in order to overcome the "greediness" introduced by the horizontal phase, while the refinement phase benefits in runtime from the good initial solution constructed in the horizontal assignment phase. The algorithm is described in the following steps:

Step 1. Set all the operations in the current partial solution to be unlocked for movement.

Step 2. Find a movement of an unlocked operation from its current chain to another such that the gain, i.e., *IIC* reduction, is the maximum (even if the gain is negative) among all of the possible movements. This operation is locked then, and the movement history is recorded.

Step 3. If the previous movement introduces an incompatibility in the binding, we apply the same movement as in Step 2 for the unlocked operation of the incompatible pair. Repeat Step 3 until no incompatibility is introduced.

Step 4. Repeat Steps 2 to 3 until all operations are locked.

Step 5. Find the first L movements which will not introduce incompatibility, such that their total gain is the maximum partial sum of the entire historical movement list. These L movements are committed, and the rest are recovered.

Step 6. Repeat Steps 1 to 5 until no movement is committed.

The entire binding algorithm calls the horizontal assignment and vertical refinement in an interleaved fashion; each iteration adds a set of operations within one control step, and gradually constructs the complete solution.

The readers are encouraged to apply this algorithm on the simple example in Fig. 4. One optimal solution will have operation v_9 assigned to chain I_c, instead of I_d, and finally $IIC = 4$.

After the operation(variable)-to-island binding, we perform a detailed binding within each island. In particular, a register file is allocated for the set of variables assigned to it. Traditional register binding techniques, such as graph-coloring and left-edge algorithms [8], may be conducted to minimize the size of each register file by sharing a register element for multiple compatible variables. Functional unit binding is trivial since the operations within an island are pairwise c-step-compatible.

Although we present the DRFM binding algorithm for data-flow graph, there is no fundamental difficulty to extend it for general control-data-flow graphs (CDFGs), since the algorithm itself does not require the directed-acyclic property for the underlying scheduled graph (a scheduled CDFG will be a general state-transition diagram (STG)). However, the lifetime and compatibility analysis should be more sophisticated for a scheduled CDFG over DFG. In addition, in a CDFG-to-DRFM binding solution, the operations assigned into the same island may form a node-disjoint cycle, instead of a simple chain in the DFG case.

6. EXTENSIONS TO GENERAL CASES

The previous algorithm produces a feasible DRFM binding solution. However, it ignores the read-port limitations of register files. In a c-step, if several operations in multiple chains consume the variables produced from the same chain I, then multiple read ports are needed by $LRF(I)$. As shown in Fig. 4, on c-step 4, four operations access three variables v_1, v_2, and v_3, which are produced from chain I_a; therefore $LRF(I_a)$ needs at least three read ports.

There is an opportunity to reduce the read-port number requirement by spreading simultaneous reads throughout different c-steps, using the slacks of dataflows. In the DFG of Fig. 4, for dataflow (v_2, v_8), which is produced in chain I_a and consumed in I_c, if we

Table 2: Sensitivity of QoR to Inter-Island Connections on Design DIR

IIC	MUX	SLICE	LUT	FF	D(ns)
26	89	451	707	263	9.61
27(3.8%)	93(4.5%)	482(6.9%)	787	257	10.14
36(38%)	102(15%)	616(37%)	1046	263	10.77
39(50%)	108(21%)	658(46%)	1110	271	10.02

could transfer the value from $LRF(I_a)$ to some buffer in I_c at c-step 3, then at c-step 4 v_8 can be accessed from the local buffer instead of $LRF(I_a)$. This way, a read port is saved for $LRF(I_a)$.

To support this mechanism, we need refinement on the DRFM to allow selective variable replication. In particular, we add a set of storage elements, namely *input buffers*, into the data-routing logic for each island, and thus allow direct data routes from an external LRF to the input buffer, as shown in Fig. 3. A scheduling algorithm for the data transfers is applied to meet the read-port constraints and to minimize the input buffers.

In addition, other general cases in practical applications, such as multicycle/pipelined operations, operation chaining, and multiple operation types, should be handled carefully in the real DRFM binding implementation.

7. EXPERIMENTAL RESULTS

The binding algorithms for DRFMs are implemented in UCLA xPilot synthesis framework [4]. In this framework a behavioral description in C is first parsed and optimized into a data-flow graph. The synthesis engine begins with latency-driven scheduling and generates a scheduled DFG. The DRFM binding algorithm discussed in Section 5 is then applied on the scheduled DFG and a set of DRFM templates to explore a desired DRFM binding. At last, a backend program generates VHDL RTL, which is accepted by existing logic synthesis and physical design tools. This experiment targets on the Xilinx Virtex II FPGA platform [15], using ISE v7.1 as the downstream tool.

The test cases, PR, LEE, CHEN, and DIR, are different discrete-cosine transformation algorithms, featuring extensive addition, subtraction and multiplication operations.

7.1 Sensitivity to Inter-Island Connections

TABLE 2 shows how global inter-island connections are correlated with the QoR on design DIR. For the same scheduling result, we perform four different DRFM binding approaches: one random binding approach and the optimizing algorithm with three different efforts. The first two columns of TABLE 2 list the inter-island connection numbers (IIC) and total multiplexer-input counts (MUX) of the resulting datapath, reported by our synthesis system. The third to fifth columns are the resource results reported by Xilinx ISE after place-and-route, namely the slice, LUT, and flip-flop (FF) counts. In the Virtex II device, a slice contains two LUTs and two FFs. The slice count represents the total resource usage, and the LUT and FF numbers show the resource distribution. The last column, D, is the achievable clock period (or path delay) reported by ISE's static timing analyzer. We set the timing constraints to 8*ns* for all the experiments.

Overall, the table shows a consistently proportional relation among the inter-island connection numbers, multiplexer-input counts, and the design area numbers (increased ratios shown in the parentheses). The delay numbers vary within a reasonable range, while the minimal-area solution has the best performance. The results suggest that the minimization of the inter-island connections is indeed the right optimization goal for resource binding on DRFMs.

Table 3: Comparisons of Three Approaches: Discrete-Register Binding, Unoptimized DRFM Binding, and Optimized DRFM Binding

	IIC	MUX	SLICE	LUT	FF/RAM	D(ns)
PR	-	123	807	1030	694/0	11.7
	27	103	404(50%)	713	179/6	11.2
(86,7)	24	90	369(8.7%)	640	162/7	10.2
LEE	-	123	717	988	576/0	12.3
	25	107	382(47%)	689	184/6	11.5
(98,7)	23	93	333(13%)	599	178/5	11.0
CHEN	-	132	1032	1335	866/0	12.3
	40	108	540(48%)	976	118/10	12.2
(102,10)	28	88	427(21%)	728	140/10	11.3
DIR	-	126	838	1210	670/0	9.6
	39	108	658(21%)	1110	271/5	10.0
(228,10)	26	89	451(31%)	707	263/5	9.6

7.2 Comparisons of QoR

For a fair comparison, we implemented a discrete-register binding algorithm presented in [3], which in turn is an enhancement of the binding algorithm in [16] and uses the bipartite-weighted-matching heuristic to minimize multiplexors for low-power FPGA designs. As reported in [3], the binding results are much better than the traditional left-edge algorithm [8], which allocates a minimum number of registers but frequently generates complex multiplexor structures. In addition, we ran through another *unoptimized* DRFM binding flow for comparison. This unoptimized flow performs a random assignment of the operations onto the given islands, only complying with the compatibility relations.

TABLE 3 shows the comparisons of the QoR for the three flows. For each test case, the first row shows the results of the discrete-register-based approach, and the second and third rows are the results of the DRFM bindings without and with optimizations, respectively. Each cell in the first column contains the design name and a parenthesized pair. In each pair, the first number is the operation count of the DFG, and the second is the resulting island number. In these experiments, the island numbers are determined by the number of node-disjoint chains of a scheduled DFG, as discussed in Section 4. In practice, the island number may be specified by designers as a constraint, or determined by a search automatically. The rest of the columns have the same meaning as those in TABLE 2, except that column "FF/RAM" also lists the number of RAM blocks used to implement register files. Note that the results for discrete-register datapaths use no RAM blocks, since no register file is applied. The table also shows that the RAM block numbers may not equal the resulting island number, since occasionally the variables produced in an island may be lifetime-compatible with each other, and thus they can be merged into a single register. In other words, an LRF may be reduced into a register, so that no RAM block is needed.

On average for this set of test cases, the unoptimized DRFM binding results achieve a 41% area reduction over the discrete-register-based approach. Between the second and third rows for each test cases, the slice counts consistently reflect the impact of the inter-island connections and multiplexor counts. Further area reductions achieved by the optimized approach vary from 8.7% to 31%, and are 18% on average. Finally, the optimized DRFM binding approach achieves more than 2X logic area reduction when compared with the discrete-register-based approach, with a 7.8% clock-period reduction on average.

8. CONCLUSIONS

The distributed register-file microarchitecture (DRFM) enables efficient use of distributed embedded memory blocks in modern FPGAs. It provides a useful architectural template for behavior synthesis and a direct optimization objective: minimizing inter-island connections. A novel DRFM binding algorithm is presented towards this objective directly. On the Xilinx Virtex II device, our experiments show a 2X logic area reduction, with a 7.8% improvement in design performance, when compared with a traditional discrete-register-based approach. The results are consistent with the significant reductions on global-interconnections and multiplexors achieved by our approach.

9. ACKNOWLEDGMENTS

This research is supported by the Semiconductor Research Corporation, Gigascale Silicon Research Center, and grants from the Altera Corporation, Magma Design Automation, Inc., and Xilinx, Inc. under the California / MICRO program.

10. REFERENCES

[1] M. Aloqeely and C. Y. R. Chen, "Sequencer-based data path synthesis of regular iterative algorithms," in *Proc. of the 31st Conference on Design Automation*, New York, 1994, pp. 155–160.

[2] J.-M. Chang and M. Pedram, "Register allocation and binding for low power," in *Proc. Conf. on Design Automation*, 1995, pp. 29–35.

[3] D. Chen, J. Cong, and Y. Fan, "Low-power high-level synthesis for FPGA architectures," in *Proc. International Symposium on Low Power Electronics and Design*, 2003.

[4] D. Chen, J. Cong, Y. Fan, G. Han, W. Jiang, , and Z. Zhang, "xpilot: A platform-based behavioral synthesis system," in *SRC TechCon'05, Portland, OR*, Nov. 2005.

[5] K. Choi and S. P. Levitan, "A flexible datapath allocation method for architectural synthesis," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 4, no. 4, pp. 376–404, 1999.

[6] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang, "Architecture and synthesis for on-chip multi-cycle communication," *IEEE Trans. Computer-Aided Design*, pp. 550–564, Apr. 2004.

[7] W. J. Dally and S. Lacy, "VLSI architecture: past, present and future," in *Proc. Conf. on Advanced Research in VLSI*, Mar. 1999, pp. 232–241.

[8] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, Inc., 1994.

[9] S. Devadas and A. Newton, "Algorithms for hardware allocation in data path synthesis," *IEEE Trans. Computer-Aided Design*, vol. 8(7), pp. 768–781, July 1989.

[10] K. I. Farkas, N. P. Jouppi, P. Chow, and Z. Vranesic, "The multicluster architecture: reducing cycle time through partitioning," in *Proc. 30th Int. Symp. on Microarchitecture*, Dec. 1997, pp. 149–159.

[11] D. Gajski, N. Dutt, A. Wu, and S. Lin, *High-Level Synthesis ÍC Introduction to Chip and System Design*. Kulwer Academic Publishers, 1992.

[12] C. H. Gebotys, "Low energy memory and register allocation using network flow," in *Proc. of the 34th Conference on Design Automation*, 1997, pp. 435–440.

[13] C. Gebotys and M. Elmasry, "Optimal synthesis of high-performance architectures," *IEEE J. Solid-State Circuits*, vol. 27(3), pp. 389–397, Mar. 1992.

[14] *Altera Website*, http://www.altera.com.

[15] *Xilinx Website*, http://www.xilinx.com.

[16] C.-Y. Huang, Y.-S. Chen, Y.-L. Lin, and Y.-C. Hsu, "Data path allocation based on bipartite weighted matching," in *Proc. of the 27th Conference on Design Automation*, 1990, pp. 499–504.

[17] M. F. Jacome, G. de Veciana, and V. Lapinskii, "Exploring performance tradeoffs for clustered VLIW ASIPs," in *Proc. International Conference on Computer-Aided Design*, 2000, pp. 504–510.

[18] J. Jeon, D. Kim, D. Shin, and K. Choi, "High-level synthesis under multi-cycle interconnect delay," in *Proc. of the Asia and South-Pacific Design Automation Conference*, Jan. 2001, pp. 662–667.

[19] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical J.*, vol. 49, no. 2, Feb. 1970.

[20] K. Keutzer, S. Malik, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli, "System level design: Orthogonalization of concerns and platform-based design," *IEEE Trans. Computer-Aided Design*, vol. 19(12), pp. 1523–1543, Dec. 2000.

[21] D. Kim, J. Jung, S. Lee, J. Jeon, and K. Choi, "Behavior-to-placed RTL synthesis with performance-driven placement," in *Proc. Int. Conf. on Computer Aided Design*, Nov. 2001, pp. 320–326.

[22] T. Kim and C. L. Liu, "An integrated data path synthesis algorithm based on network flow method," *Proc. of the IEEE Custom Integrated Circuits Conference*, vol. 1-4, pp. 615–618, May 1995.

[23] ——, "A new approach to the multiport memory allocation problem in data path synthesis," *Integr. VLSI J.*, vol. 19, no. 3, pp. 133–160, 1995.

[24] P. Kollig and B. M. Al-Hashimi, "Simultaneous scheduling, allocation and binding in high level synthesis," *Elect. Let.*, vol. 33, Aug. 1997.

[25] E. Kusse and J. Rabaey, "Low-energy embedded FPGA structures," in *Proc. of the 1998 International Symposium on Low Power Electronics and Design (ISLPED'98)*, New York,, 1998, pp. 155–160.

[26] D. Lanneer, M. Cornero, G. Goossens, and H. De Man, "Data routing: a paradigm for efficient data-path synthesis and code generation," in *Proc. of the 7th International Symposium on High-level Synthesis (ISSS'94)*, 1994, pp. 17–22.

[27] H.-D. Lee and S.-Y. Hwang, "A scheduling algorithm for multiport memory minimization in datapath synthesis," in *Proc. of the Asia and South-Pacific Design Automation Conference*, 1995, pp. 93–100.

[28] M. Luthra, S. Gupta, N. Dutt, and R. G. A. Nicolau, "Interface synthesis using memory mapping for an FPGA platform," in *Proc. 21st International Conference on Computer Design*, 2003, pp. 140–145.

[29] T. A. Ly and J. T. Mowchenko, "Applying simulated evolution to high level synthesis," *IEEE Trans. Computer-Aided Design*, vol. 12(3), pp. 389–409, Mar. 1993.

[30] N. Magen, A. Kolodny, U. Weiser, and N. Shamir, "Interconnect-power dissipation in a microprocessor," in *Proc. of the 2004 International Workshop on System Level Interconnect Prediction (SLIP'04)*, 2004, pp. 7–13.

[31] C. A. Mandal, P. P. Chakrabarti, and S. Ghose, "Some new results in the complexity of allocation and binding in data path synthesis," *Computers and Mathematics with Applications*, vol. 35, no. 10, pp. 93–105, 1998.

[32] B. M. Pangrle, "On the complexity of connectivity binding," *IEEE Trans. Computer-Aided Design*, vol. 10(11), pp. 1460–1465, 1991.

[33] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1982.

[34] J. M. Rabaey, C. Chu, P. Hoang, and M. Potkonjak, "Fast prototyping of datapath-intensive architectures," *IEEE Des. Test. Comput.*, vol. 8, no. 2, pp. 40–51, 1991.

[35] M. Rim, R. Jain, and R. De Leone, "Optimal allocation and binding in high-level synthesis," in *Proc. of the 29th ACM/IEEE Conference on Design Automation (DAC'92)*, 1992, pp. 120–123.

[36] S. Rixner, W. J. Dally, B. Khailany, P. R. Mattson, U. J. Kapasi, and J. D. Owens, "Register organization for media processing," in *Proc. of the 6th Intenational Symposium on High-Performance Computer Architecture*, 2000, pp. 375–386.

[37] A. Singh, G. Parthasarathy, and M. Marek-Sadowska, "Efficient circuit clustering for area and power reduction in FPGAs," *ACM Trans. Design Automation of Electronic Systems*, vol. 7, no. 4, pp. 643–663, 2002.

[38] L. Stok and W. Philipsen, "Module allocation and comparability graphs," *IEEE International Sympoisum on Circuits and Systems*, vol. 11-14 vol.5, pp. 2862–2865, June 1991.

[39] C.-J. Tseng and D. Siewiorek, "Automated synthesis of data paths in digital systems," *IEEE Trans. Computer-Aided Design*, vol. 5(3), pp. 379–395, July 1986.