

# Simultaneous Resource Binding and Interconnection Optimization Based on a Distributed Register-File Microarchitecture

JASON CONG

University of California, Los Angeles

YIPING FAN

AutoESL Inc.

and

JUNJUAN XU

University of California, Los Angeles

---

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 based on a *distributed register-file microarchitecture (DRFM)*, which makes efficient use of distributed embedded memory blocks as register files in modern FPGAs. DRFM contains multiple islands, each having a local register file, a functional unit pool and data-routing logic. Compared to 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 a 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. Given the scheduling solution and resource (functional units) constraints, two novel algorithms in the resource binding stage are developed based on DRFM: (i) A simultaneous DRFM clustering and binding algorithm, which decides the configuration of DRFM and the assignment of operations into islands with the focus on optimizing global connections; (ii) A data-forwarding scheduling algorithm, which takes advantage of the operation slacks to handle the read-port restriction of register files. On the Xilinx Virtex4 FPGA platform, experimental results with a set of real-life test cases show a 50% logic area reduction achieved by applying our approach, with a 14.6% performance improvement, compared to the traditional discrete-register-based approach. Also, experiments on small-size designs show that our algorithm produces the same number of total connections and at most one more maximum feeding-in connection compared to optimal solutions generated by ILP.

Categories and Subject Descriptors: B.5.1 [**Register-Transfer-Level Implementation**]: Design—*Data-path design*; B.5.2 [**Register-Transfer-Level Implementation**]: Design Aids—*Automatic synthesis*

General Terms: Algorithms, Design, Experimentation, Performance

Additional Key Words and Phrases: Behavioral synthesis, Resource binding, Distributed register file

---

A preliminary version of this work was presented in Proceedings of the 2006 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 709 - 715 [Cong et al. 2006].

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20YY ACM 1084-4309/20YY/0400-0001 \$5.00

## 1. INTRODUCTION

With the advancement of integrated circuit technology, interconnects have had 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. For field-programmable gate arrays (FPGAs), studies show that interconnects contribute 70 to 80% of the total area [Singh et al. 2002] and 75 to 85% of the total power [Kusse and Rabaey 1998]. Multiplexors, which are collections of interconnects without actual computational functionality, except for data routing, are particularly expensive for FPGA platforms.

At the register-transfer level, a multiplexor is required when multiple data sources feed into a single port of a resource instance (register or functional unit) in multiple control steps (*c*-steps). As shown in Fig. 1(a), the behavior of a design is represented in a scheduled dataflow graph (DFG). Round nodes represent operations, and numbered rectangles represent variables that need to be stored in registers. Those variables having a same number will share a common register. After resource binding using discrete registers, a datapath is generated, as shown in Fig. 1(b), where two multiplexors are needed to route the dataflows in 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 see 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 limited numbers of read and write ports, a centralized register file may not work for highly parallelized applications which require multiple simultaneous data reads 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 out in [Rixner et al. 2000], the area and power consumption of a register file grows *cubically* with its port number. Advanced FPGA devices, such as Virtex IV [Xilinx ] and Stratix II [Altera ], 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. In comparison, a distributed 3-register-file datapath would be more efficient in this case.

The use of distributed register files is further encouraged on platforms with rich on-chip memory or register-file IP blocks. For example, in Xilinx Virtex serial or 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 I shows

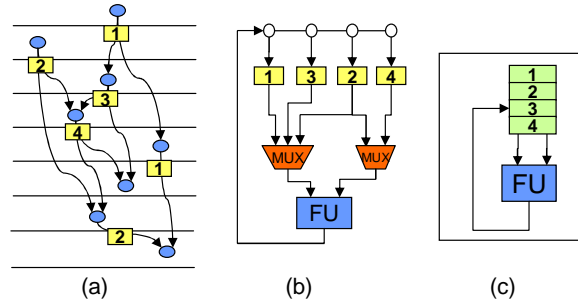


Fig. 1. Advantages of register files over discrete registers. (a) A scheduled dataflow graph with register binding indicated on each variable; (b) Binding using discrete registers; (c) Binding using a register file.

data related to memory blocks on Virtex4 [Xilinx ] and Stratix [Altera ]. Since we know that the implementation of multiplexors on FPGAs is very expensive [Chen et al. 2003] and register files are able to reduce the multiplexor use on such platforms, it is not surprising to see a 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:

(1) A distributed register-file microarchitecture (DRFM) is presented as a parameterizable microarchitecture template. DRFM contains multiple islands, each having a register file, a functional unit pool and data-routing logic. It imposes no specific restriction for scheduling, i.e., existing scheduling algorithms may not need to distinguish DRFM from the traditional discrete-register-based microarchitecture. DRFM will be particularly beneficial for reducing the interconnect complexity in FPGA designs.

(2) The properties of DRFM are investigated, and specific optimization goals, i.e., the *total number of inter-island connections* and the *maximum number of feeding-in connections* among all islands, are proposed for minimizing interconnect and multiplexor complexity. The complexity of the optimization problem is also analyzed.

(3) Given the scheduling solution and the resource constraint, a simultaneous resource clustering and binding algorithm is proposed to decide the DRFM configuration and target DRFM to optimize global interconnects directly.

(4) To handle the read-port restriction of the register files, a novel data-transfer scheduling algorithm is presented, taking advantage of operation slacks to minimize the required number of read ports.

(5) The resource binding problem based on DRFM is proved to be NP-hard. For the purpose of optimality study, an integer-linear-programming (ILP) formulation is presented to evaluate the quality of our heuristic.

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, problem formulation and complexity analysis in Section 4, the DRFM

<b>Xilinx XC-4VLX</b>	<b>60</b>	<b>80</b>	<b>100</b>	<b>160</b>	<b>200</b>
#18Kb BRAM	160	200	240	288	336
Dist. RAM(Kb)	416	560	768	1,056	1,392
<b>Altera EP1</b>	<b>S25</b>	<b>S30</b>	<b>S40</b>	<b>S60</b>	<b>S80</b>
#M512(512b)	224	295	384	574	767
#M4K(4Kb)	138	171	183	292	364
#M-(512Kb)	2	4	4	6	9

Table I. On-chip RAM blocks on Virtex4 and Stratix FPGA devices.

configuration and binding algorithm is discussed in Section 6. Section 7 presents the ILP formulation, and Section 8 discusses how to handle the register-file read-port restriction by using a data-forwarding algorithm. Extensions to CDFGs and operation chaining are discussed in Section 9. Experimental results are presented in Section 10, followed by our conclusions in Section 11.

## 2. RELATED WORK AND OUR CONTRIBUTION

There is extensive literature on general binding algorithms in high-level synthesis targeting discrete registers, where functional units access all registers directly and with assumed equal cost [Huang et al. 1990][Stok and Philipsen 1991][Gajski et al. 1992] [De Micheli 1994] [Chang and Pedram 1995][Gebotys 1997] [Chen and Cong 2004][Cong and Xu 2008]. However, the increasing interconnect effect encourages the research on architectures that exploit the physical locality by operating on data close to where it is stored. [Jeon et al. 2001] and [Kim et al. 2001] 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 cycles, which decouples communication and computation. Further improvement is shown in [Cong et al. 2004], which presents a Regular Distributed Register (RDR) microarchitecture and an architectural synthesis methodology, with the emphasis on multicycle on-chip communication for synchronous designs. The work in [Huang et al. 2007] targets memory-intensive applications and proposes a method for partitioning and scheduling array data and operations into distributed architectures. However, these microarchitectures do not use register files specifically or the on-chip embedded memories for register-file implementation.

One of the early research projects related to register file architecture in behavior synthesis is the Hyper system [Rabaey et al. 1991]. It proposed using 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

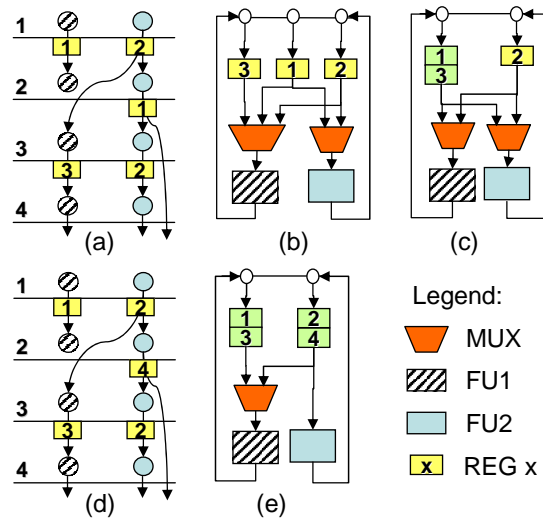


Fig. 2. Three binding solutions for the same scheduled DFG. (a) A scheduled DFG marked with binding information obtained by a traditional binding approach; (b) The discrete-register-based binding according to (a); (c) The binding using a register file derived from (b); (d) The scheduled DFG marked with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

with only 1-write-port register files. Binding solution (c) uses a register file derived from the traditional binding solution (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 same control step, as do registers 2 and 3. The more aggressive binding solution, (d to 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 in (a).

There has been research that focuses on the synthesis for minimizing register file (or memory module) numbers or port numbers, so that the interconnects can be optimized indirectly. In [Luthra et al. 2003] the authors proposed a hardware/software co-synthesis approach to allocate data to shared memories on FPGAs. Their algorithm uses lifetime information produced by a scheduling algorithm and minimizes the number of memory instances in order to simplify multiplexors indirectly. [Kim and Liu 1995a] and [Lee and Hwang 1995] discuss scheduling algorithms for minimizing memory numbers or port numbers. All of these approaches focus on scheduling techniques, while none of them focus on the resource binding stage.

In [Kim and Liu 1995b], the authors applied interconnect minimization techniques during variable allocation (after operation binding) for datapaths with multi-port memory modules. Our approach differs from [Kim and Liu 1995b] in that we consider the bindings for operations and variables together in a unified way, targeting an island-based microarchitecture template.

The works introduced above are all in the field of behavioral synthesis, which generates application-specific FPGA/ASIC designs. For general processors, there

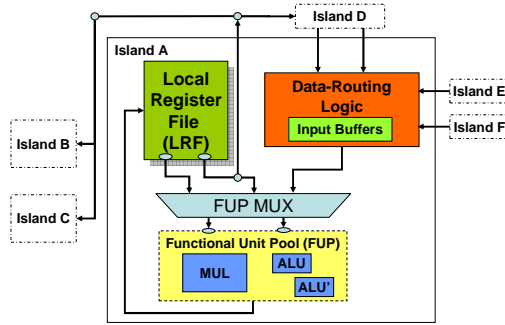


Fig. 3. Illustration of an island in a distributed register-file microarchitecture (DRFM) instance.

have been also extensive research regarding distributed architectures [Farkas et al. 1997] [Rixner et al. 1998] [Dally and Lacy 1999] [Khailany et al. 2001] [Seznec et al. 2002] [Bunchua 2004]. In [Bunchua 2004], the author proposed replacing the centralized register file used in traditional processors with distributed register files and showed the impact of different register-file configurations on performance, area and power. Besides the difference in microarchitecture, such as the data-transfer method among different register files and the organization of each cluster, the fundamental distinction of [Bunchua 2004] and our approach is that [Bunchua 2004] determines the processor configuration in advance and then maps applications onto it dynamically or statically, while our approach optimizes and configures the whole datapath targeting one specific design.

### 3. DISTRIBUTED REGISTER-FILE MICROARCHITECTURE

The essential insight behind many approaches discussed in Section 2 is that communication (or data transfers) should be localized as much as possible so that the interconnect effect is minimized. 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 multiplexers 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 multiplexers, if used, are usually much smaller compared to those in the datapath using discrete registers. Hereafter, let  $\mathbb{M} = \{\mathcal{I}_1, \dots, \mathcal{I}_K\}$  denote a DRFM configuration with  $K$  islands. Suppose  $\mathcal{I}$  is an island; we use  $LRF(\mathcal{I})$  and  $FUP(\mathcal{I})$  to represent its LRF and FUP, respectively.

In an ideal DRFM configuration, each LRF is restricted to only *1-write-port* but there is no restriction on the read-port number. No data replication is allowed in this datapath, i.e., a variable can only be stored in one register element of a fixed

register file during its lifetime. We will investigate how to handle the read-port restriction in Section 8.

DRFM provides many advantages in behavior synthesis. First, it is a semi-regular microarchitecture template. Although it has a write-port restriction on each LRF, it provides much more flexibility than the traditional VLIW and DSP architectures because DRFM has no restrictions on data-routing structures and configurations of the LRF and FUP. Those flexible configurations should be determined by the application and synthesis algorithms. Second, DRFM provides a template and specific optimization goals for synthesis algorithms. For example, the data-routing logic should be optimized by any synthesis algorithm targeting DRFMs to minimize the interconnect and multiplexor complexity. Last, modern FPGA platforms are very efficient for implementing DRFMs, given their rich on-chip memory resources. Table I in Section 1 summarizes the total number of memories for Virtex4 serial. We can see that memories are abundant and well distributed on the whole chip. Given a reasonable DRFM configuration, where most dataflows happen on the intra-island interconnects, a good physical synthesis tool will place all the resources belonging to a single DRFM island physically together. Therefore, intra-island interconnects do have better timing than inter-island interconnects. The design and optimization goal of our proposed algorithm in Section 6 will be based on this property of DRFM. Experiments in Section 10 show that our assumption conforms to the reality.

## 4. PRELIMINARIES AND DRFM BINDING PROPERTIES

### 4.1 Preliminaries

The behavioral kernels of an application to be synthesized are represented as *dataflow 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)*. We use  $\mathbb{T}$  to denote the total number of control steps. Hereafter, without explicit mention, we assume **simplified DFGs**, where each operation  $v$  takes exactly one c-step and produces exactly one output variable. We use the **same notation for an operation and the variable it produces**.

*Definition 4.1.* In a scheduled DFG  $G$ , let  $T(v)$  denote the c-step where operation  $v$  is scheduled. An operation  $u$  is *compatible to*  $v$ , if  $T(u) < T(v)$ . The compatibility 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  $E_c = \{(u, v) | u \text{ is compatible to } v, \forall u, v \in V\}$ .

Note that the definition of compatibility in this paper is different from the traditional one, which requires  $u$  and  $v$  to have the same operation type in addition to their scheduling relationship. In the scheduled DFG of Fig. 4, there are compatibility edges from  $v_1$  to  $v_2$ , and from  $v_7$  to  $v_{10}$ , etc., which are not explicitly shown in the figure.

Operations  $u$  and  $v$  are *incompatible* if there is no compatibility edge  $(u, v)$  or  $(v, u)$  in  $G_c$ . That is,  $u$  and  $v$  are incompatible if and only if  $T(u) = T(v)$ .

*Definition 4.2.* The *lifetime* of a variable  $u$ , denoted as  $\mathcal{L}(u)$ , is defined as the

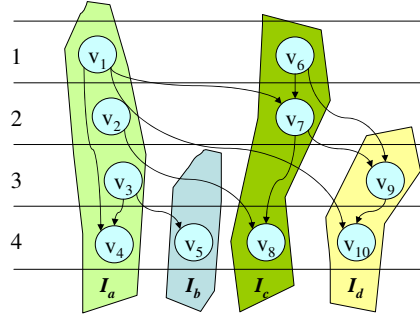


Fig. 4. A scheduled dataflow graph and four node-disjoint chains (compatibility edges are not drawn explicitly).

$c$ -step interval from its generation to its last consumption. Variables  $u$  and  $v$  are *lifetime-conflicting* (or simply *conflicting*) if and only if their lifetime intervals have overlap.

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

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.<sup>1</sup> Two operations cannot share one functional unit if they have a different functional type or 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 the same functional unit, while variables  $v_6$  and  $v_9$  may share the same register to hold their value since their lifetimes are disjointed.

A valid resource 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 timing and area of the datapath dramatically.

## 4.2 DRFM Binding

Using the definition of DRFM in Section 3, within island  $\mathcal{I}$ , a local operation issued in functional unit pool  $FUP(\mathcal{I})$  always writes its output variable into local register file  $LRF(\mathcal{I})$ , which has only one write port. Therefore, in a valid *resource binding of  $G$  onto DRFM  $\mathbb{M}$* , if operation  $v$  is assigned to  $FUP(\mathcal{I})$ , then its variable  $v$  must be stored into  $LRF(\mathcal{I})$  at  $c$ -step  $T(v)$ , and any other operation cannot write data into  $LRF(\mathcal{I})$  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 4.3.* A resource binding of a scheduled DFG  $G(V, E)$  on DRFM  $\mathbb{M}$ , denoted as  $\mathcal{B}(G, \mathbb{M})$ , is a function  $\mathcal{B} : V \rightarrow \mathbb{M}$ . In a feasible  $\mathcal{B}$ , if operation  $u$  and  $v$  are incompatible,  $\mathcal{B}(u) \neq \mathcal{B}(v)$ .

<sup>1</sup>Here we assume simple resource binding. More complicated binding allowing variables to be replicated among registers is outside of the topic.

Definition 4.3 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.

Based on Definition 4.3, we have 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 takes exactly 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.,  $\mathcal{I} = \mathcal{B}(v)$  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  $\mathcal{I}_a = \{v_1, v_2, v_3, v_4\}$  is bound into an island, as is chain  $\mathcal{I}_c = \{v_6, v_7, v_8\}$ . In this example, at least four islands are required to obtain a feasible binding, since there are at least four chains in this scheduled DFG. This is also indicated by the following property.

**THEOREM 4.4.**  $\mathcal{B}(G, \mathbb{M})$  will not be feasible if the number of the islands in  $\mathbb{M}$  is less than the minimum number of node-disjoint chains in  $G_c$ .

### 4.3 Inter-Island Connections

*Definition 4.5.* In a feasible DRFM binding solution  $\mathcal{B}(G, \mathbb{M})$ , for dataflow  $(u, v)$ , if  $\mathcal{B}(u) \neq \mathcal{B}(v)$ , we call the dataflow an *inter-chain* or *global* dataflow; otherwise it is an *intra-chain* or *local* dataflow.

Let us suppose that in Fig. 4 the binding solution corresponds to the four shaded chains (islands),  $\{\mathcal{I}_a, \mathcal{I}_b, \mathcal{I}_c, \mathcal{I}_d\}$ ; 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 at different c-steps and also 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  $\mathcal{I}_a$  and  $\mathcal{I}_c$ . In contrast, dataflows  $(v_6, v_9)$  and  $(v_7, v_9)$  must use two different global connections between  $\mathcal{I}_c$  and  $\mathcal{I}_d$  since they are consumed at the same c-step.

In a feasible DRFM binding  $\mathcal{B}(G, \mathbb{M})$ , for two different islands  $\mathcal{I}_i, \mathcal{I}_j \in \mathbb{M}$ , two dataflows  $(u_i, u_j)$  and  $(v_i, v_j)$ , where  $\mathcal{I}_i = \mathcal{B}(u_i) = \mathcal{B}(v_i)$  and  $\mathcal{I}_j = \mathcal{B}(u_j) = \mathcal{B}(v_j)$ , **can** share a common **inter-island connection** if and only if  $u_j \neq v_j$ ; i.e., the two dataflows are consumed by two different and compatible operations.

*Definition 4.6.* In a feasible DRFM binding  $\mathcal{B}(G, \mathbb{M})$ , the number of *inter-island connections* (*IICs*) between islands  $\mathcal{I}_i$  and  $\mathcal{I}_j$ , denoted as  $IIC_{\mathcal{B}}(\mathcal{I}_i, \mathcal{I}_j)$ , is the minimum number of connections required to carry all the dataflows between these two islands (chains) under the sharing conditions. The total number of inter-island connections of  $\mathcal{B}(G, \mathbb{M})$  is defined as  $Total\_IIC(\mathcal{B}) = \sum_{\forall \mathcal{I}_i, \mathcal{I}_j \in \mathbb{M}, \mathcal{I}_i \neq \mathcal{I}_j} IIC_{\mathcal{B}}(\mathcal{I}_i, \mathcal{I}_j)$ . The maximum number of feeding-in connections among all islands of  $\mathbb{M}$  is defined

as  $Max\_IIC(\mathcal{B}) = Max_{\forall \mathcal{I}_i \in \mathbb{M}} \{ \sum_{\forall \mathcal{I}_j \in \mathbb{M}, \mathcal{I}_j \neq \mathcal{I}_i} IIC_{\mathcal{B}}(\mathcal{I}_j, \mathcal{I}_i) \}$ .

$IIC_{\mathcal{B}}(\mathcal{I}_i, \mathcal{I}_j)$  must be not larger than the maximum number of fanins of all the operations of the chain  $\mathcal{I}_j$ .

For Fig. 4 we have the results shown below, given that dataflows  $(v_1, v_7)$  and  $(v_2, v_8)$  can share an inter-island connection, while  $(v_6, v_9)$  and  $(v_7, v_9)$  cannot share one.

$$\begin{aligned}
 Total\_IIC(\mathcal{B}) &= IIC_{\mathcal{B}}(\mathcal{I}_a, \mathcal{I}_b) + IIC_{\mathcal{B}}(\mathcal{I}_a, \mathcal{I}_c) + IIC_{\mathcal{B}}(\mathcal{I}_a, \mathcal{I}_d) \\
 &\quad + IIC_{\mathcal{B}}(\mathcal{I}_b, \mathcal{I}_a) + IIC_{\mathcal{B}}(\mathcal{I}_b, \mathcal{I}_c) + IIC_{\mathcal{B}}(\mathcal{I}_b, \mathcal{I}_d) \\
 &\quad + IIC_{\mathcal{B}}(\mathcal{I}_c, \mathcal{I}_a) + IIC_{\mathcal{B}}(\mathcal{I}_c, \mathcal{I}_b) + IIC_{\mathcal{B}}(\mathcal{I}_c, \mathcal{I}_d) \\
 &\quad + IIC_{\mathcal{B}}(\mathcal{I}_d, \mathcal{I}_a) + IIC_{\mathcal{B}}(\mathcal{I}_d, \mathcal{I}_b) + IIC_{\mathcal{B}}(\mathcal{I}_d, \mathcal{I}_c) \\
 &= 1 + 1 + 1 + 0 + 0 + 0 + 0 + 0 + 2 + 0 + 0 + 0 \\
 &= 5 \\
 Max\_IIC(\mathcal{B}) &= Max\{1, 1, 3\} = 3
 \end{aligned}$$

As shown in Section 8, the dataflows from island  $i$  to  $j$  are very likely sharable by a common physical connection. In general,  $IIC$  is not easy to compute because of the sharing-relations, which are determined by the compatibility relations. The calculation of  $IIC$  is equivalent to solving a graph-coloring problem for the *incompatibility-graph* of dataflows from one island to the other. However, the incompatibility-relations among the operations within one island is very sparse, and the  $IIC$  numbers can be computed fairly fast.

## 5. PROBLEM DEFINITION AND COMPLEXITY ANALYSIS

### 5.1 Problem Definition

The inter-island connections are critical to the final DRFM qualities (also shown in Section 10.1), since for any feasible  $\mathcal{B}(G, \mathbb{M})$ , the input-port number of island  $\mathcal{I}$  is equal to the number of inter-island connections feeding into  $\mathcal{I}$ . For example, island  $A$  in Fig. 3 has four input ports because there are four global connections feeding into it. Fig. 5(a) and (b) show two schemes of DFG partitioning. Fig. 5(b) has more inter-island connections and more MUXes due to its failure to transfer as many dataflows through intra-island connections as possible. This implies that the complexity of the data-routing logic, which impacts the design area and critical path timing, is determined by the inter-island connections, and it suggests the following problem formulation.

**PROBLEM 1. *DRFM Configuration and Binding for Minimum Inter-Island Connections.*** *Given a scheduled DFG  $G(V, E)$  and the resource (functional units) constraint, find a feasible DRFM configuration  $\mathbb{M}$ , and perform resource binding  $\mathcal{B}(G, \mathbb{M})$ , so that both  $Total\_IIC(\mathcal{B})$  and  $Max\_IIC(\mathcal{B})$  are minimized.*

The task of DRFM configuration is to decide the number of islands,  $K$ , and cluster the available resources into these  $K$  islands; that is, to decide  $FUP$  for each island.

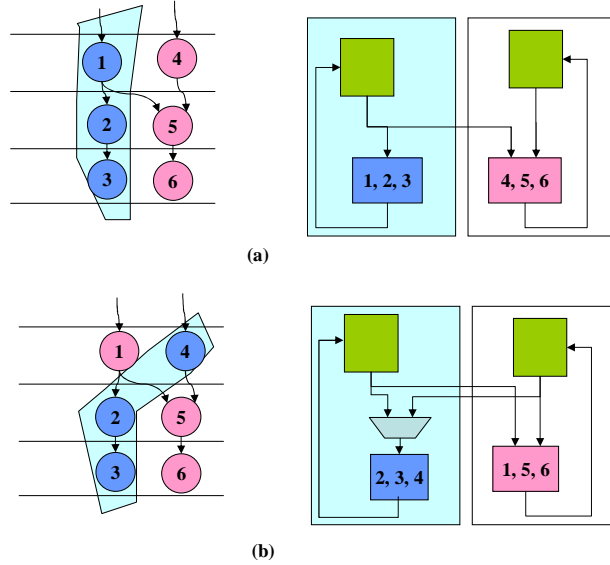


Fig. 5. Two partitioning schemes.

Since minimizing  $Total\_IIC(\mathcal{B})$  is expected to have a more global impact on design qualities, we give  $Total\_IIC(\mathcal{B})$  a higher priority over  $Max\_IIC(\mathcal{B})$  during optimization.

**PROBLEM 2. DRFM Configuration and Binding for Minimum Total Inter-Island Connections.** *Given a scheduled DFG  $G(V, E)$  and the resource (functional units) constraint, find a feasible DRFM configuration  $\mathbb{M}$ , and perform resource binding  $\mathcal{B}(G, \mathbb{M})$ , so that  $Total\_IIC(\mathcal{B})$  is minimized.*

We will show that Problem 1 and 2 are both NP-hard. The difference between Problem 1 and Problem 2 is that Problem 2 only minimizes  $Total\_IIC(\mathcal{B})$ . Since Problem 1 is harder than Problem 2, to prove Problem 1 is NP-hard, we only need to prove Problem 2 is NP-hard.

## 5.2 Complexity Analysis

In [Mandal et al. 1998], the authors prove that Problem PA3U1 is NP-hard, which is defined as follows. *Given a set of variables which have been placed in a particular memory with three uniform reading ports and a set of circuit points which read variables from the memory at specific control steps, the port assignment problem is to assign the three memory ports to these circuit points such that (a) all the accesses in each control step are satisfied and (b) the cost of multiplexers in front of circuit points is minimized.* In the following, we will reduce Problem PA3U1 to Problem 2 and prove it is NP-hard.

**THEOREM 5.1.** *Problem 2 is NP-hard.*

**PROOF.** Fig. 6 shows an instance of Problem PA3U1, where the memory has three reading ports,  $p_1$ ,  $p_2$ , and  $p_3$ . There are  $k$  circuit points,  $c_i, 1 \leq i \leq k$ .

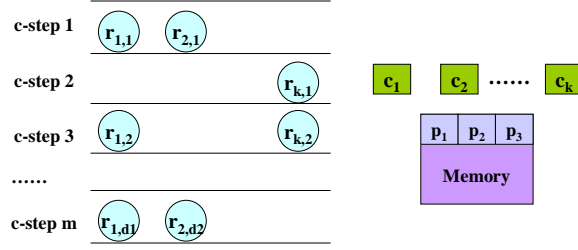


Fig. 6. An instance of Problem PA3U1.

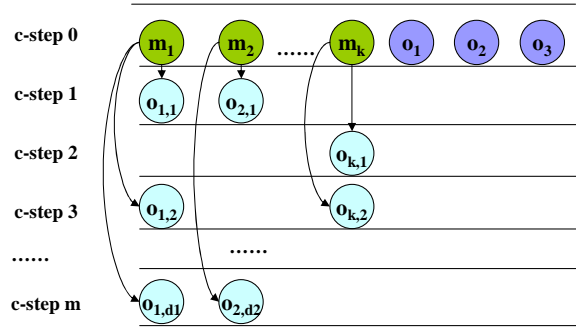


Fig. 7. An instance of Problem 2.

Each circuit point,  $c_i$ , has a queue of  $d_i$  number of variable-reading operations,  $r_{i,j}$ ,  $1 \leq j \leq d_i$ , which are distributed in c-step 1 to  $m$ .

The corresponding instance of Problem 2, a scheduled DFG, is shown in Fig. 7. For each circuit point  $c_i$ ,  $1 \leq i \leq k$ , there is a node  $m_i$  in the DFG scheduled at step 0. For the three reading ports of memory, there are three nodes,  $o_1$ ,  $o_2$ , and  $o_3$ , which are also scheduled at step 0. For each reading operation  $r_{i,j}$ , there is a node  $o_{i,j}$  in the DFG, which is scheduled at the same step as  $r_{i,j}$ . Each  $o$  ( $o_1$ ,  $o_2$ ,  $o_3$ ,  $o_{i,j}$ ) node has one output to be stored in register files, whose representing edges are omitted in the DFG of Fig. 7 for the purpose of simplification. At last, there is a dataflow from  $m_i$  to  $o_{i,j}$ ,  $1 \leq j \leq d_i$ .

The reduced instance is configured so that all  $o$  nodes have the same functional type  $f(o)$  and thus can share functional units, and all  $m$  nodes have a functional type  $f(m)$ , different from  $f(o)$ . The given resource constraint includes three functional units of type  $f(o)$  and  $k$  functional units of type  $f(m)$ .

Obviously, the reduction from Problem PA3U1 to Problem 2 is polynomial.

Due to the aforementioned assumption of one writing port in register files, all the nodes at step 0 have to be assigned to different islands. Therefore, there is only one feasible DRFM configuration, which has  $k + 3$  islands, and each island includes exactly one functional unit.

A solution of Problem PA3U1 is derived from a feasible solution of Problem 2 in the following way. If  $o_{i,j}$  is bound to the same island as  $o_q$ ,  $1 \leq q \leq 3$ , the reading operation  $r_{i,j}$  will access port  $p_q$ . Since nodes scheduled in the same c-step in Fig. 7 will not be assigned to a common island, *reading* operations occurring in the same

c-step in Fig. 6 will not access the same reading port either. Hence, the solution of Problem PA3U1, derived from a feasible solution of Problem 2, is also feasible.

Since there is no dataflow among nodes  $m_i$ , there would be no inter-island connections required among the  $k$   $m$ -islands. This is also true for the three  $o$ -islands holding all  $o$  nodes. Hence, inter-island connections are only required between the  $k$   $m$ -islands and the three  $o$ -islands. If nodes  $o_{i,j_1}$  and  $o_{i,j_2}$ , which have the same input nodes  $m_i$ , are assigned to the same island, dataflows  $m_i \rightarrow o_{i,j_1}$  and  $m_i \rightarrow o_{i,j_2}$  can share the same inter-island interconnect. This is exactly the sharing rule between circuit points and memory ports: if a circuit point reads two variables from a same memory port, only one connection is needed. Therefore, *Total\_IIC* of Problem 2 equals the cost of multiplexers of Problem PA3U1.

Based on the above facts, we conclude that an optimal solution of Problem 2 leads to an optimal solution of Problem PA3U1, thus proving Problem 2 is NP-hard.

□

From Theorem 5.1, we have the following conclusion.

**THEOREM 5.2.** *Problem 1 is NP-hard.*

## 6. AN INCREMENTAL RESOURCE-CONSTRAINED DRFM CONFIGURATION AND BINDING ALGORITHM

Problem 1 is not easier to solve than the traditional binding problem for connectivity optimization [Pangrle 1991]. In addition, the global connections among DRFM islands may be shared by multiple dataflow edges (see Section 4.3); and this property makes Problem 1 different from the classic graph partitioning problem, where the edges are static.

In this section we will introduce the incremental resource clustering and binding algorithm with the goal of optimizing inter-island connections.

### 6.1 Algorithm Flow

Before introducing the algorithm, we give the following definition, which will be used in DRFM configuration.

*Definition 6.1.* In a DRFM configuration  $\mathbb{M}$ , islands  $\mathcal{I}_i$  and  $\mathcal{I}_j$  are combinable if and only if  $FUP(\mathcal{I}_i) \cap FUP(\mathcal{I}_j) = \emptyset$ ; that is, these two islands do not have functional units of the same type in common.

Due to the assumed limitation of only one writing port for each register file (Section 3), a valid binding solution would not assign two or more operations scheduled at the same clock cycle into the same island. Therefore, for operations of the same type, only one functional unit of the corresponding type is needed for each island, and putting two or more functional units of the same type into one island would be a waste of resources. On the other hand, functional units of different types are allowed and necessarily reside in the same island, since they could perform different functionalities at different cycles.

Fig. 8 shows the main flow of the incremental configuration and binding algorithm. The basic idea is that we start from DRFM configurations with large numbers of islands and perform binding based on them. Then islands having a large

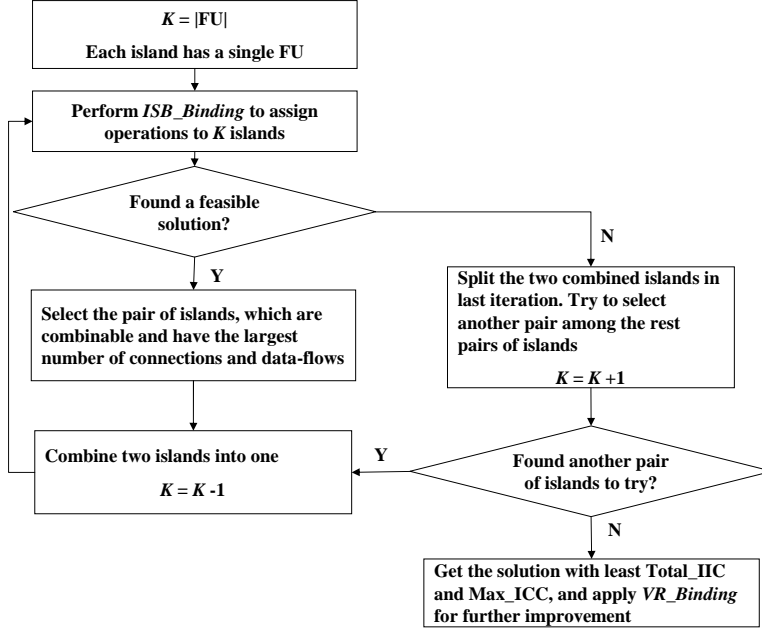


Fig. 8. Incremental algorithm flow for DRFM configuration and binding.

amount of communications are combined together gradually to hide inter-island dataflows.

In the algorithm, we select the total number of available functional units as the initial total island number, which means that in the initial  $\mathbb{M}$ , there are  $|\text{FU}|$  number of islands and each island contains only one single functional unit. *ISB\_Binding* is then performed, targeting  $\mathbb{M}$  to assign operations to islands. The optimization goal of *ISB\_Binding* is to minimize inter-island connections. Details will be introduced in Section 6.2. In the following iterations, those islands which are tightly coupled will be grouped together to form new islands. We always select the pair of islands which have not been tried and also have the largest number of connections and dataflows between them. In this way, both inter-island physical interconnects and logical dataflows are minimized. Whenever DRFM configuration is updated, *ISB\_Binding* will be called to verify if there exists a valid binding solution. If yes, the latest DRFM configuration will be accepted as the starting point for the posterior island combinations. Otherwise, the newly combined islands will be split and the remaining pairs of combinable islands will be tried. As the incremental process continues, islands are becoming more and more compact and inter-island connections will be hidden and become intra-island connections gradually. When no more islands can be combined, the iterations stop and the best DRFM configuration and binding solution in the sense of inter-island interconnects is retrieved. A post-processing refinement, *VR\_Binding*, is then performed to achieve further optimization. *VR\_Binding* will be introduced in detail in Section 6.3.

Here, we use both  $Total\_IIC(\mathcal{B})$  and  $Max\_IIC(\mathcal{B})$  to evaluate the quality of solutions. As explained in Section 4.3,  $Total\_IIC(\mathcal{B})$  has a higher optimizing priority

over  $Max\_IIC(\mathcal{B})$ . Therefore, we always choose those solutions with the minimum number of  $Total\_IIC(\mathcal{B})$  and then use  $Max\_IIC(\mathcal{B})$  to break the tie. If there is still more than one solution remaining, we choose the one having the smallest number of islands. The cost functions used in  $ISB\_Binding$  and  $VR\_Binding$  also evaluate  $Total\_IIC(\mathcal{B})$  and  $Max\_IIC(\mathcal{B})$  in the same priority.

During the island reorganization, we only apply one simple scheme: island combination. This scheme works well only for designs with a few types of functional units. Otherwise, a more intelligent DRFM configuration mechanism, which should be capable of combining, splitting and recomposing, is needed to achieve better DRFM configurations.

Note that for general cases with multi-cycle/pipelined operations or register files with more than one write port, combinability is not necessary for island combination.

## 6.2 $ISB\_Binding$ Algorithm

For each DRFM configuration  $\mathbb{M}$ , we apply an iterative control-step-by-control-step bipartite approach,  $ISB\_Binding$ , to assign operations to islands in  $\mathbb{M}$  with the goal of minimizing connections. Each iteration 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.

In each iteration, we consider the set of operations  $\Theta$  within the current c-step. Since the operations in  $\Theta$  are pairwise incompatible, they must be assigned onto different islands. We apply a minimum-weighted bipartite matching algorithm to obtain an assignment solution. A similar idea was presented in [Huang et al. 1990] for general datapath allocation.

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

- (1) For each operation  $v$  there is a node  $n(v) \in V_\Theta$ , and for each island  $\mathcal{I}_i$  there is a node  $m(\mathcal{I}_i) \in V_{\mathbb{M}}$ .
- (2) For each operation  $v$  and island  $\mathcal{I}_i$ , if  $FUP(\mathcal{I}_i)$  owns a functional unit of the same type as  $v$ , build an edge  $(n(v), m(\mathcal{I}_i)) \in V_\Theta \times V_{\mathbb{M}}$ .
- (3) Assign cost for each edge  $(n(v), m(\mathcal{I}_i))$ .

The *cost of the attempted binding* of operation  $v$  to  $\mathcal{I}_i$  is defined as

$$cost_b(v, \mathcal{I}_i) = \alpha * New\_ICC(v, \mathcal{I}_i) + \beta * Max\_ICC(v, \mathcal{I}_i),$$

where  $New\_ICC(v, \mathcal{I}_i)$  is the number of the *new* inter-island connections introduced by the assignment of  $v$  to  $\mathcal{I}_i$ , and the value of  $Max\_ICC(v, \mathcal{I}_i)$  is 1 if and only if  $\mathcal{I}_i$  has the largest number of feeding-in connections in the current partial solution.  $\alpha$  is the weight for global inter-island connections, and  $\beta$  is the weight for the maximum number of feeding-in connections. Since we prefer solutions with globally minimized connections, we give  $\alpha$  a larger value over  $\beta$ . In experiments, we set  $\alpha$  as the total number of operations in  $G$ , and  $\beta$  as 1.

A minimum-weighted bipartite matching  $E_{match} \subseteq V_\Theta \times V_{\mathbb{M}}$  for  $G_{bp}$  can be computed optimally in  $O(n^2 * \log(n) + n * e)$  [Fredman and Tarjan 1987], where  $n$  is the total number of nodes of  $G_{bp}$ , and  $e$  is the total number of edges of  $G_{bp}$ . For each edge in  $(n(v), m(\mathcal{I}_i)) \in E_{match}$ , we bind operation  $v$  to island  $\mathcal{I}_i$ , and update  $\mathcal{B}'$ .

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

**THEOREM 6.2.** *The binding of  $\Theta$  to  $\mathbb{M}$  produced by the above algorithm introduces a minimum number of new inter-island connections to the current partial DRFM-binding solution and increases the maximum number of feeding-in connections as little as possible.*

Globally, the matching algorithm performs the binding in a “horizontal” fashion in the scheduled DFG, and cannot calculate its impact on the future iterations. For this reason, we perform a post refinement to further improve the results.

### 6.3 Post-Refinement

After we get a reasonable and valid DRFM configuration and binding solution, we apply a vertical local-search-based refinement, *VR\_Binding*, for further improvement. The refinement process uses an idea similar to the Kernighan-Lin algorithm [Kernighan and Lin 1970], 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 *ISB\_Binding*, while the refinement benefits in runtime from the good initial solution obtained from *ISB\_Binding*. The algorithm is described in the following steps:

- (1) Set all the operations in the current partial solution to be unlocked for movement.
- (2) Find a movement of an unlocked operation from its current chain to another such that the gain 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.
- (3) Repeat Step 2 until all operations are locked.
- (4) Find the first  $K$  movements, such that their total gain is the maximum partial sum of the entire historical movement list. These  $K$  movements are committed, and the rest are recovered.
- (5) Repeat Steps 1 to 4 until no movement is committed.

We use the simple example in Fig. 4 to illustrate the process. At Step 2, the maximal gain is obtained by moving operation  $v_9$  from chain  $\mathcal{I}_d$  to chain  $\mathcal{I}_c$ , which reduces both *Total\_IIC* and *Max\_ICC* by one. The other movements either increase or maintain the interconnect cost. Therefore, at Step 4 we will commit the first  $K = 1$  movements and recover the rest. The following iterations will not improve the partitioning further. Finally, we get the optimal solution with *Total\_IIC* = 4 and *Max\_ICC* = 2.

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 [De Micheli 1994], 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.

## 7. OPTIMALITY STUDY

One potential problem of the algorithm proposed in Section 6 is that the searching may be stuck in local minimal solutions. To evaluate the effectiveness and optimality of the heuristic, we formulate Problem 1 as an ILP formulation. ILP either maximizes or minimizes an objective function of a set of variables, subject to a group of linear equation and inequality constraints and integral restrictions on all of the variables.

In the following we assume the given resource constraints include only adders and multipliers. However, the ILP formulation can be easily extended for general cases.

Let  $A$  be the number of adders, and  $M$  be the number of multipliers. We give each resource instance an identifier ranging from 1 to  $A + M$ . Also, the maximum number of islands in a feasible DRFM configuration is also  $A + M$ , when each island has a single functional unit. Let  $O$  be the group of operations in DFG. We first define the following variables.

$x_{i,p}$ : Operation  $i$  is assigned to island  $p$ ,  $\forall i \in O, \forall p \in [1, A + M]$ .

$c_{p,q}$ : The number of connection from island  $p$  to island  $q$ ,  $\forall p, q \in [1, A + M], p \neq q$ .

$a_p$ : The number of adders in island  $p$ ,  $\forall p \in [1, A + M]$ .

$m_p$ : The number of multipliers in island  $p$ ,  $\forall p \in [1, A + M]$ .

$z_p$ : The number of feeding-in connections of island  $p$ .

$max\_in$ : The maximum number of feeding-in connections among all islands, i.e.,  $Max\_ICC$ .

All the variables have non-negative integer values.  $z_p$  and  $max\_in$  are equal or greater than 0, and all the other variables can only be 0 or 1.

Based on the above definitions, the constraint that each operation is assigned to one and only one island is described below.

$$\sum_{\forall p \in [1, A + M]} x_{i,p} = 1 \quad \forall i \in O \quad (1)$$

The operations scheduled at the same c-step cannot be assigned to the same island.

$$\sum_{\forall i \in O, T(i)=t} x_{i,p} \leq 1 \quad \forall p \in [1, A + M], \forall t \in [1, \mathbb{T}] \quad (2)$$

For dataflow from  $o_i$  to  $o_j$ , suppose that  $o_i$  is assigned to island  $p$ , and  $o_j$  is assigned to island  $q$ . There will be a connection from island  $p$  to island  $q$ .

$$c_{p,q} \geq x_{i,p} + x_{j,q} - 1 \quad \text{for all dataflow } i \rightarrow j, \forall p, q \in [1, A + M], p \neq q \quad (3)$$

Given an operation  $k$ , suppose its inputs are produced by operations  $i$  and  $j$ ,  $i \neq j$ . If operation  $k$  is assigned to island  $q$ , and both  $i$  and  $j$  are assigned to island  $p$ , then there will be two connections from island  $p$  to island  $q$ .

$$c_{p,q} \geq x_{i,p} + x_{j,p} + x_{k,q} - 1 \quad \forall k \in O, k\text{'s inputs are from } i \text{ and } j, i \neq j, \forall p, q \in [1, A+M], p \neq q \quad (4)$$

The number of feeding-in connections for island  $p$  is the sum of connections coming from all the other islands.

$$z_p = \sum_{\forall q \in [1, A+M], q \neq p} c_{q,p} \quad (5)$$

$max\_in$  is the maximum of the feeding-in connections of all islands.

$$max\_in \geq z_p \quad \forall p \in [1, A+M] \quad (6)$$

If there is at least one addition assigned to island  $p$ , there will be an adder in island  $p$ ,<sup>2</sup>. The same constraint applies to multipliers.

$$a_p \geq x_{i,p} \quad \forall i \in O, i \text{ is addition} \quad (7)$$

$$m_p \geq x_{i,p} \quad \forall i \in O, i \text{ is multiplication} \quad (8)$$

The given resource constraints cannot be violated.

$$\sum_{\forall p \in [1, A+M]} a_p \leq A \quad (9)$$

$$\sum_{\forall p \in [1, A+M]} m_p \leq M \quad (10)$$

The objective function is the weighted sum of  $Total\_IIC$  and  $Max\_IIC$ .

$$Min : \alpha * \sum_{\forall p \in [1, A+M]} z_p + \beta * max\_in \quad (11)$$

## 8. HANDLING READ-PORT RESTRICTIONS

The algorithm in Section 6 produces a feasible DRFM binding solution. However, it ignores the read-port limitations of register files. A large number of read-ports would increase the accessing timing of register files and thus jeopardize the timing of the whole design. In a c-step, if several operations in multiple chains consume the variables produced from the same chain  $\mathcal{I}$ , then multiple read ports are needed by  $LRF(\mathcal{I})$ . 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  $\mathcal{I}_a$ ; therefore  $LRF(\mathcal{I}_a)$  needs at least three read ports.

<sup>2</sup>As introduced in Section 6.1 due to the assumed limitation of only one writing port for register files, each island needs at most one functional unit for each functional type. Therefore,  $a_p$  is 0-1 variable.

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  $\mathcal{I}_a$  and consumed in  $\mathcal{I}_c$ , if we could transfer the value from  $LRF(\mathcal{I}_a)$  to some buffer in  $\mathcal{I}_c$  at c-step 3, then at c-step 4  $v_8$  can access the local buffer to retrieve the data instead of accessing  $LRF(\mathcal{I}_a)$ . This way, a read port is saved for  $LRF(\mathcal{I}_a)$ .

To support this mechanism, we need a refinement on the DRFM to allow selective variable replication. In particular, we add a set of storage elements, namely *input buffer*, 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. Such direct data routes are called *data-forwarding paths*. A *data forwarding*  $\chi(u, v, t)$  reads out the value  $u$  from  $LRF(\mathcal{B}(u))$  and writes it into an input buffer of  $\mathcal{B}(v)$  through a data-forwarding path at c-step  $t$ .

The problem of rescheduling a set of dataflows on data-forwarding paths under read-port constraints, named *data-forwarding scheduling*, is defined as follows.

**PROBLEM 3. *Data-Forwarding Scheduling.*** *Given a positive number  $N$  and an island  $\mathcal{I} \in \mathbb{M}$  with respect to a feasible DRFM binding  $\mathcal{B}(G, \mathbb{M})$ ; for each dataflow  $(u, v)$  where  $\mathcal{B}(u) = \mathcal{I}$  and  $\mathcal{B}(v) \neq \mathcal{I}$ , schedule a data forwarding  $\chi(u, v, t)$  where  $t \in [T(u) + 1, T(v)]$ , such that at any c-step there is no more than  $N$  simultaneous data forwardings.*

Obviously, a successful solution to Problem 3 guarantees that no more  $N$  simultaneous reads to  $LRF(\mathcal{I})$  happen at any c-step, so that  $N$  read ports are sufficient for  $LRF(\mathcal{I})$ . If no solution is returned, it indicates that the given read-port number  $N$  is too tight. In this case, as a final resort, the register file is duplicated to increase its read-port number.

For each feasible data forwarding  $\chi(u, v, t)$ ,  $t \in [T(u) + 1, T(v)]$  must hold; i.e., it has a start time  $T(u) + 1$  and deadline  $T(v)$ . In addition, each data forwarding takes exactly one c-step and requires exactly one read port of its source register file. If we view a data forwarding as a task with unit execution time, and view read ports as processors, then each task requires a processor, and they have identical execution times and nonequal deadlines and ready times. Furthermore, the tasks have no precedence (or dependency) relation. Therefore, the problem is a special case of the *deadline scheduling of tasks with ready time*. This problem is solvable in  $O(n^2)$  by an earliest deadline first (EDF) algorithm [Blazewicz 1979], where  $n$  is the number of tasks.

Revisions of the algorithm in [Blazewicz 1979] are needed for minimizing the input-buffer numbers and for special cases. For example, when two dataflows are produced by the same operation, they can share one read port; when two dataflows feed into the same island but at different c-steps, they may be able to share an input buffer.

## 9. EXTENSIONS TO GENERAL CASES

### 9.1 CDFG

Although we present the DRFM binding algorithm for dataflow graphs, there is no fundamental difficulty in extending it for general control-data-flow graphs (CD-

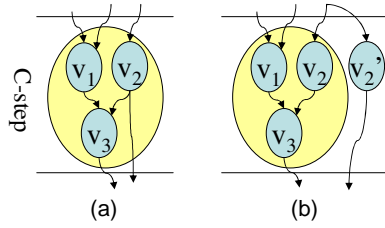


Fig. 9. An operation-chaining example. (a) Collapse the chained operations into a complex operation, which has 2 outputs; (b)  $v_2$  is duplicated so that each operation has 1 output.

FGs). The main extension is regarding the change of the compatibility definition. That is, operations scheduled at the same step but under exclusive conditions (e.g. in different branches of an *if* or *case* statement) are also compatible and allowed to be allocated into the same island. Although the algorithm presented in Section 6 can be applied to CDFGs without any change, it may result in inferior solutions. The reason is that those compatible operations under the new compatibility definition are not allowed to reside in the same island, which decreases optimization opportunities and may lead to larger inter-island connections.

To handle the new compatibility with CDFGs, we extend the *ISB\_Binding* algorithm presented in Section 6.2 in the following way. For operations  $\Theta$  in each control step, we first divide  $\Theta$  into disjoint groups such that operations in the same group are incompatible with each other and thus must be assigned to different islands. The division of operations  $\Theta$  is performed as follows:

- (1) If all operations in  $\Theta$  are grouped, exit; otherwise, create a new group  $g$ .
- (2) Select an ungrouped operation  $v$ . If  $v$  is incompatible with all operations in group  $g$ , put  $v$  into  $g$ .
- (3) Repeat Step 2 until no more operations can be grouped into  $g$ .
- (4) Goto Step 1.

After group division, we apply the bipartite matching algorithm to sequentially assign each group to islands. Since compatible operations are in different groups, they have a chance to be allocated in the same island. In step (2) of the bipartite graph construction, besides the resource availability requirement, we also require that operation  $v$  is compatible with all operations which are already assigned to island  $\mathcal{I}_i$ .

In addition, the post-refinement in Section 6.3 will use the new compatibility definition when moving operations among islands.

## 9.2 Chaining

Operation chaining may introduce tricky situations in DRFM binding [Stok 1992] [De Micheli 1994]. We try to collapse and assign an entire operation chain into one island to maintain the computation locality. When we collapse a set of chained operations, the resulting larger complex operation could have multiple outputs. An example is illustrated in Fig. 9(a), where a 2-output complex operation is formed. The multi-output situation causes a difficulty for our problem formulation, which assumes single-output operations only (for the 1-write-port restriction for register

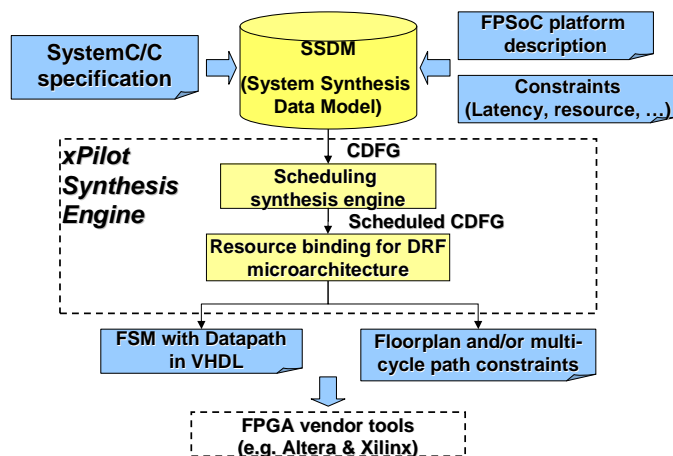


Fig. 10. Synthesis flow for DRF microarchitecture.

files). There are several ways to handle this special case: (i) If a collapsed operation is not too large and the output number is small, we perform necessary operation-duplications and split the large operation into several single-output operations (as shown in Fig. 9(b)); (ii) Otherwise, we can bind the complex operation into a special island that has a multiple-write-ports register file (or simply discrete registers).

## 10. EXPERIMENTAL RESULTS

The binding algorithms for DRFMs are implemented in the UCLA xPilot synthesis framework [Cong et al. 2006]. The complete synthesis flow is illustrated in Fig. 10. In this framework a behavioral description in C/SystemC is first parsed and optimized into a dataflow graph. The synthesis engine begins with latency-driven scheduling and generates a scheduled DFG/CDFG. The DRFM configuration and binding algorithm is then applied on the scheduled DFG/CDFG to explore a desired DRFM configuration and binding solution. The data-forwarding algorithm described in Section 8 accepts the DRFM binding and tries to meet given LRF read-port constraints. At last, a backend program generates VHDL RTL, which is accepted by existing logic synthesis and physical design tools. In this work we report the results of experiments targeting the Xilinx Virtex4 FPGA platform [Xilinx], using ISE v9.1 as the downstream tool.

A set of real-life test cases is used in our experiments. The pure DFG test cases include several different discrete-cosine transformation algorithms, such as DIR, LEE and WANG, and several DSP programs, such as HONDA and MCM [Srivastava and Potkonjak 1995]. Three other test cases, MATMUL, CFTMDL and CFT1ST, are CDFGs from MediaBench [Lee et al. 1997] and an FFT package [FFT]. All the benchmarks are data intensive applications.

IIC	SLICE	LUT	FF	CLK(ns)
26	885	1516	197	9.31
30	984	1646	222	9.69
45	1024	1755	187	10.44

Table II. Sensitivity of QoR to inter-island connections on design *DIR*.

### 10.1 Sensitivity to Inter-Island Connections

Table II shows how global inter-island connections are correlated with the QoR on design *DIR*. For the same scheduling result, we perform three different DRFM binding approaches: one random binding approach and the optimizing algorithm with two different efforts; hence we obtain three solutions. The first column of Table II lists the inter-island connection numbers (IIC) of the resulting datapath reported by our synthesis system. The second to fourth columns are the resource results reported by Xilinx ISE after place-and-route, namely the slice, LUT, and flip-flop (FF) counts. In the Virtex4 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, CLK, is the achievable clock period (or path delay) reported by ISE’s static timing analyzer. We set the timing constraints as  $8ns$  for all the experiments.

Overall, the table consistently shows a proportional relation among the inter-island connection numbers and the design area numbers. The delay numbers vary within a reasonable range, while the minimal area solution has the best performance. The results suggest that the minimization of inter-island connections is indeed the right optimization goal for resource binding on DRFMs (Section 5.1).

### 10.2 Comparisons of QoR

For a fair comparison, we implemented a discrete-register binding algorithm presented in [Chen and Cong 2004], which is to optimize multiplexers during register binding. We applied the same algorithm for functional unit binding. As reported in [Chen and Cong 2004], the binding results are much better than the traditional left-edge algorithm [De Micheli 1994], which allocates a minimum number of registers but frequently generates complex multiplexor structures. It is also better than the bipartite algorithm [Chen et al. 2003]. In addition, we ran through another three flows for comparison, listed as follow.

- (1) *Discrete\_Register*: Resource binding based on discrete registers [Chen and Cong 2004].
- (2) *RF\_No\_Cost*: Do not optimize connections in *ISB\_Binding* and only perform a random assignment of the operations onto the islands, only complying with compatibility relations. And do not perform *VR\_Binding*.
- (3) *RF\_ICC\_T*: In cost function  $cost_b(v, \mathcal{I}_i)$ , only consider the total number of ICC; that is, we set  $\beta$  as 0.
- (4) *RF\_ICC\_T&M*: The proposed flow and cost function in Section 6.

Table III shows a comparison of the QoR for the above four flows. The first column shows the benchmarks. The numbers of “1” and “2” attached at the end

	Discrete_Register			RF_No_Cost			RF_ICC_T			RF_ICC_T&M		
	SLICE	LUT	FF/RAM	SLICE	LUT	FF/RAM	SLICE	LUT	FF/RAM	SLICE	LUT	FF/RAM
dir1	1,897	3,209	784	1,116	1,923	215/20	1,095	1,843	229/17	951	1,594	245/16
dir2	2,038	3,374	792	1,106	1,936	186/24	1,096	1,840	278/20	976	1,686	198/18
honda1	1,246	2,085	487	519	878	113/4	423	712	115/3	398	670	107/3
honda2	1,183	2,035	430	585	935	165/4	427	719	89/6	504	845	129/4
lee1	1,216	2,206	233	798	1,500	141/7	757	1,404	119/10	736	1,323	207/3
lee2	1,260	2,316	221	949	1,689	151/7	791	1,452	192/6	768	1,383	171/5
mcm1	1,809	3,159	542	1263	2,225	157/25	1,095	1,920	163/20	1272	2,228	211/18
mcm2	2,042	3,496	690	1,211	2,071	214/20	1,062	1,749	245/18	960	1,629	174/19
wang1	1,204	2,085	297	672	1,243	241/9	630	1,188	222/8	674	1,285	248/12
wang2	988	1,765	284	598	1102	216/8	625	1159	247/11	579	1057	221/8
matmul1	888	1,492	435	313	469	163/10	253	427	100/13	257	435	102/13
matmul2	911	1,537	431	386	618	155/10	228	391	83/12	244	410	113/12
cftmdl1	2,020	3,534	784	1,240	1,956	483/18	1,205	1,903	492/16	1,243	1,967	480/18
cftmdl2	2,142	3,872	767	1,383	2,217	537/16	1,316	2,127	477/16	1,206	1,961	495/16
cftblst1	4,809	8,676	1,615	2,211	3,484	911/20	2,268	3,599	906/21	2,186	3,489	908/20
cftblst2	4,837	8,882	1,587	2,303	3,704	914/21	2,380	3,846	941/18	2,389	3,836	921/18
Avg.	1	1	-	54%	50%	-	49%	50%	-	50%	49%	-

Table III. Resource comparison of four experimental flows.

mean different resource-constraint settings. For each flow we list the resource results (slice, LUT, and FF/RAM) reported by Xilinx ISE after place-and-route. These columns have the same meaning as those in Table II, except that columns “FF/RAM” also list 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. This table also shows that the register-file-based approach still uses some discrete registers. The reason is that 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. From the results in Table III, we can see that all the three register-file-based flows achieve around 50% resource reductions on average due to the saving of multiplexers.<sup>3</sup>

<sup>3</sup>For design *lee* and *wang*, register-file-based flows do not decrease discrete registers significantly. The reason is that to make each island physically compact, we do not put arithmetic FUs and I/O FUs in the same island. The scheduling of *lee* and *wang* determines that the execution of most I/O operations conflict with each other and thus have to be assigned to different I/O FUs in different islands. Therefore, most of the variables from these I/O FUs reside in different islands and thus are stored in discrete registers. That is, most I/O operations have their own dedicated registers. On the other hand, variables from arithmetic FUs and I/O FUs share registers in the discrete-register flow. In spite of the marginal reduction on discrete registers, our approach has

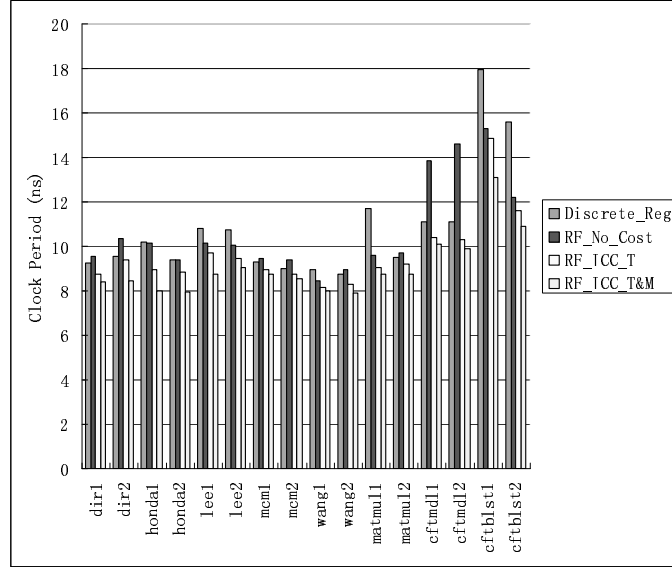


Fig. 11. Timing results of four experimental flows.

Fig. 11 shows the comparison of clock-period results by the four experimental flows, and Table V lists the total number of inter-island connections and maximum number of feeding-in connections by the three flows targeting DRFM, but with different cost functions. For half of the cases, the timing of flow *RF\_No\_Cost* is even worse than *Discrete\_Register* due to its large number of global connections, which increases the size of data-routing logic and also puts extensive pressure on physical placement and routing. On average, the timing of *RF\_No\_Cost* is almost the same as *Discrete\_Register*. On the other hand, *RF\_ICC\_T* and *RF\_ICC\_T&M* are 9.3% and 14.6% better than *Discrete\_Register*, respectively. The even better performance achieved by *RF\_ICC\_T&M* is due to its extra optimization on the feeding-in connections, which could help relieve local congestion.

Table IV lists the power consumption of four flows given by xPower, which comes with the Xilinx ISE toolset. The results show that compared with *Discrete\_Register*, the register-file-based flows consume almost the same amount of power, only 1% higher on average. The main reason is that register files are implemented with block RAMs, which consume a large amount of power and eliminate the power savings from DRFM's better area and interconnections.

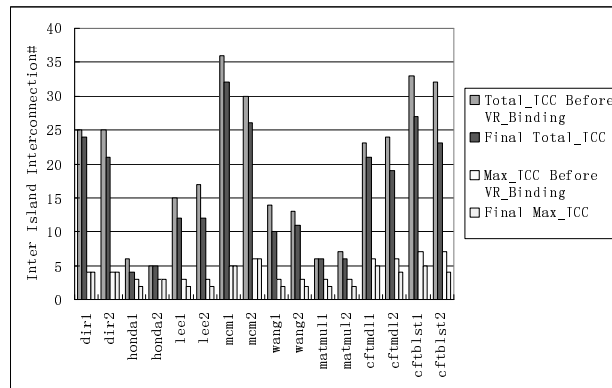
The above experimental results indicate that register-file-based architectures can lead to great resource reduction, but to achieve better timing performance, synthesis algorithms should optimize global-connection carefully.

---

much less MUX due to the fact that all the other variables produced by arithmetic FUs are stored in register files.

	Discrete_Reg(W)	RF_No_Cost(W)	RF_ICC_T(W)	RF_ICC_T&M(W)
dir1	0.910	0.931	0.938	0.935
dir2	0.911	0.953	0.932	0.928
honda1	0.802	0.800	0.798	0.798
honda2	0.802	0.801	0.801	0.801
lee1	0.901	0.913	0.918	0.906
lee2	0.903	0.913	0.904	0.909
mcm1	0.912	0.954	0.942	0.940
mcm2	0.913	0.933	0.929	0.930
wang1	0.903	0.915	0.914	0.919
wang2	0.901	0.912	0.912	0.913
matmul1	1.368	1.374	1.381	1.381
matmul2	1.356	1.374	1.377	1.377
cftmdl1	1.390	1.410	1.406	1.393
cftmdl2	1.393	1.409	1.391	1.406
cftblst1	1.434	1.427	1.430	1.427
cftblst2	1.436	1.431	1.427	1.436
Avg.	1	1.014	1.011	1.010

Table IV. Power comparison of four experimental flows.


 Fig. 12. Interconnection reduction by *VR\_Binding*.

### 10.3 Effectiveness of *VR\_Binding*

Fig. 12 illustrates the comparison of interconnect results before and after performing *VR\_Binding*. *VR\_Binding* reduces *Total\_ICC* by from 1 to 9, and reduces *Max\_ICC* by 1 or 2. This shows that *VR\_Binding* does help improve the quality of DRFM binding.

	RF_No_Cost		RF_ICC_T		RF_ICC_T&M	
	Total_ICC	Max_ICC	Total_ICC	Max_ICC	Total_ICC	Max_ICC
dir1	45	8	24	5	24	4
dir2	50	9	26	5	26	4
honda1	8	3	3	3	4	2
honda2	6	4	5	4	5	3
lee1	25	5	13	3	13	2
lee2	31	5	13	4	12	2
mcm1	52	10	25	6	25	5
mcm2	43	11	27	7	26	6
wang1	19	4	10	3	10	2
wang2	18	5	12	3	11	2
matmul1	8	4	6	3	6	2
matmul2	12	4	6	3	6	2
cftmdl1	31	8	22	6	21	5
cftmdl2	28	7	20	5	19	4
cftblst1	35	7	27	6	27	5
cftblst2	32	7	24	5	23	4

Table V. Connection comparison of three different cost functions.

#### 10.4 Optimality Study

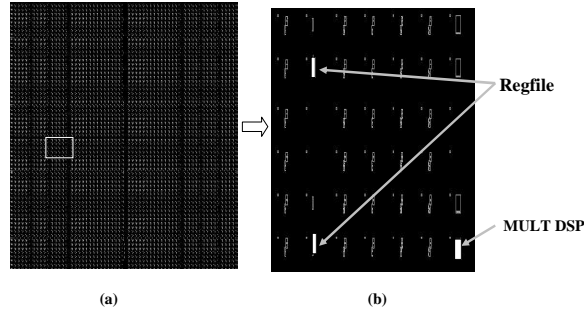
To study the optimality of the binding algorithm in Fig. 8, we tested it against the ILP formulation with small designs of around 15 operations. For medium- or large-sized designs used in Section 10.2, ILP fails to obtain final solutions due to its long runtime. Table VI lists the results of three flows. The second and third columns show the results from ILP, which are optimal in terms of both *Total\_ICC* and *Max\_ICC*. The fourth column (Time) shows the CPU running time of ILP in seconds. The remaining six columns show results from *RF\_No\_Cost* and *RF\_ICC\_T&M*. We can see that *RF\_ICC\_T&M* achieves the optimal results on almost all the designs, except that for “test6”, *Max\_ICC* of *RF\_ICC\_T&M* is one higher than the optimal results.

#### 10.5 Results of FPGA Placement

As mentioned in Section 3, one of the important factors impacting the qualities of the DRFM microarchitecture is that all the resources in a same island need to be placed closely. If this is not the case, the timing of designs would be jeopardized due to long intra-island interconnects. Also, the optimization goal of the proposed binding algorithm might not be right either. Since most dataflows are carried through intra-island connections rather than inter-island connections due to optimization of the binding algorithm, we expect that FPGA physical synthesis tools could make the right decisions for us based on the generated VHDL files. Fig. 13(a) shows the graphical resource placement of Xilinx Virtex4 device xc4vlx60

	ILP			RF_No_Cost			RF_ICC_T&M		
	Total_ICC	Max_ICC	Time(s)	Total_ICC	Max_ICC	Time(s)	Total_ICC	Max_ICC	Time(s)
Test1	3	2	140	6	2	0	3	2	1
Test2	6	2	1387	7	3	0	6	2	0
Test3	8	2	2584	12	2	0	8	2	1
Test4	4	2	749	6	3	0	4	2	1
Test5	4	2	123	5	2	0	4	2	0
Test6	4	1	362	5	3	0	4	2	0
Avg.	1	1	-	1.4	1.5	-	1	1.16	-

Table VI. Comparison against ILP.


 Fig. 13. Part of placement result of *chem1*.

	1a1m	1a3m	3a1m	3a3m	5a1m	5a3m
lee	7/7	12/10	0/0	1/1	0/0	0/0
chem	7/7	5/5	1/1	0/0	0/0	2/2
dir	0/0	0/0	0/0	0/0	1/1	1/1

Table VII. Results of data-forwarding scheduling.

in FPGA Editor [Xilinx]. Fig. 13(b) shows the zoomed-in detail of the rectangle area in Fig. 13(a) for design *dir1*. The left two bars are register file, and the bar on the right side is multiplier DSP. They logically belong to the same island in the DRFM configuration. The other resources, like adders and data-routing logics, are implemented around them. Hence, FPGA tools did place an island's resources closely enough. We examined other islands and observed the same results.

## 10.6 Results of Handling Read-Port Limitations

Table VII lists results of the data-forwarding scheduling algorithm for handling the read-port number restriction of LRFs. In this experiment, the functional unit constraints are set as  $xAyM$  and indicate that  $x$  ALUs and  $y$  multipliers are provided. During the DRFM binding stage, we choose  $x + y$  as the number of islands (i.e.,

each functional unit forms an FUP). For data-forwarding scheduling, we set  $N$  to 2, meaning that the read-port number is restricted to no more than 2. The first number in each cell is the number of rescheduled dataflows, and the second is the allocated input buffer. From the experiment, we observed that for all the test cases after DRFM binding, the resulting read-port numbers are already less than 4. The data-forwarding scheduling algorithm returns success for all the cases, reschedules up to 12 dataflows, and allocates up to 10 input buffers. The table also shows that in many cases no reschedule is required, since the DRFM binding solutions meet the read-port restrictions already. Note that the results are not monotone with the resource constraints, since the binding and data-forwarding algorithms are discrete and sensitive to the DFG scheduling, which in turn is very sensitive to resource constraints.

## 11. 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. Two novel algorithms are proposed during the resource binding stage: (i) The DRFM configuration and binding algorithm focuses on the minimization of inter-island connections; (ii) The data-forwarding scheduling algorithm takes advantage of the operation slacks to handle the read-port restriction of register files. On the Xilinx Virtex4 device, our experimental results show a 50% logic area reduction, with a 14.6% improvement in design performance, when compared to a traditional discrete-register-based approach. The results are consistent with the significant global-interconnect and multiplexor reductions achieved by our approach. Also, experiments show that our algorithm achieves the same number of total connections and at most one more maximal feeding-in connection compared to optimal solutions that are generated by ILP formulation.

In the paper we ignored many important factors in practical applications, such as pipelined operations. These cases should be handled carefully in the real DRFM binding implementation.

Despite the encouraging results, several directions may be investigated for further improvement on the DRFM synthesis topic. Since scheduling determines the parallelism of the scheduled DFG, it will greatly impact the result of DRFM binding. Forward-looking heuristics during scheduling [Wong et al. 2002] and scheduling-aware partitioning for interconnect optimization [Lim et al. 2007] shall be very beneficial to the final DRFM quality. Although our current binding algorithm is efficient and flexible, a more global and less greedy binding algorithm could be developed to further minimize inter-island connections. Also, our binding algorithm does not directly consider register-file optimization. The reduction of the number of register files and the number of reading/writing ports will lead to better power consumption.

## REFERENCES

- ALTERA. *Altera Website*. <http://www.altera.com>.
- BLAZEWICZ, J. 1979. Deadline scheduling of tasks with ready times and resource constraints. *Information Processing Letters* 8(2), 60–63.
- ACM Transactions on Design Automation of Electronic Systems, Vol. V, No. N, Month 20YY.

- BUNCHUA, S. 2004. Fully distributed register files for heterogeneous clustered microarchitectures. In *Thesis, School of Electrical and Computer Engineering, Georgia Institute of Technology*.
- CHANG, J.-M. AND PEDRAM, M. 1995. Register allocation and binding for low power. In *Proc. of the 32nd Conference on Design Automation*. 29–35.
- CHEN, D. AND CONG, J. 2004. Register binding and port assignment for multiplexer optimization. In *Proc. of the Asian and South Pacific Design Automation Conference*. 68–73.
- CHEN, D., CONG, J., AND FAN, Y. 2003. Low-power high-level synthesis for FPGA architectures. In *Proc. International Symposium on Low Power Electronics and Design*.
- CONG, J., FAN, Y., HAN, G., JIANG, W., AND ZHANG, Z. 2006. Platform-based behavior-level and system-level synthesis. In *Proc. of IEEE International SOC Conference*. Austin, Texas. (Invited Paper), 199–202.
- CONG, J., FAN, Y., HAN, G., YANG, X., AND ZHANG, Z. 2004. Architecture and synthesis for on-chip multi-cycle communication. 550–564.
- CONG, J., FAN, Y., AND JIANG, W. 2006. Platform-based resource binding using a distributed register-file microarchitecture. In *Proc. ACM/IEEE International Conference on Computer Aided Design*. San Jose, California.
- CONG, J. AND XU, J. 2008. Simultaneous fu and register binding based on network flow method. In *Proc. of the 2008 Design, Automation and Test in Europe*. Munich, Germany, 1057–1062.
- DALLY, W. J. AND LACY, S. 1999. VLSI architecture: past, present and future. In *Proc. 20th Conf. on Advanced Research in VLSI*. 232–241.
- DE MICHELI, G. 1994. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, Inc.
- FARKAS, K. I., JOUPPI, N. P., CHOW, P., AND VRANESIC, Z. 1997. The multicluster architecture: reducing cycle time through partitioning. In *Proc. 30th Int. Symp. on Microarchitecture*. 149–159.
- FFT. *FFT package*. <http://momonga.t.u-tokyo.ac.jp/ooura/fft.html>.
- FREDMAN, M. AND TARJAN, R. E. 1987. Fibonacci heaps and their uses in improved network optimization algorithms. In *Journal of the ACM*. 596–615.
- GAJSKI, D., DUTT, N., WU, A., AND LIN, S. 1992. *High-Level Synthesis C Introduction to Chip and System Design*. Kulwer Academic Publishers.
- GEBOTYS, C. H. 1997. Low energy memory and register allocation using network flow. In *Proc. of the 34th Conference on Design Automation*. 435–440.
- HUANG, C., RAVI, S., RAGHUNATHAN, A., AND JHA, N. K. 2007. Generation of heterogeneous architectures for memory-intensive applications through high-level synthesis. *IEEE Trans. Very Large Scale Integration Systems (TVLSI)* 15, 1191–1204.
- HUANG, C.-Y., CHEN, Y.-S., LIN, Y.-L., AND HSU, Y.-C. 1990. Data path allocation based on bipartite weighted matching. In *Proc. of the 27th Conference on Design Automation*. 499–504.
- JEON, J., KIM, D., SHIN, D., AND CHOI, K. 2001. High-level synthesis under multi-cycle interconnect delay. In *Proc. of the Asia and South-Pacific Design Automation Conference*. 662–667.
- KERNIGHAN, B. W. AND LIN, S. 1970. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal* 49, 2 (Feb.).
- KHAILANY, B., DALLY, W., KAPASI, U., MATTSON, P., NAMKOONG, J., OWENS, J., B.TOWLES, CHANG, A., AND RIXNER, S. 2001. Imagine: Media processing with streams. In *IEEE Micro*.
- KIM, D., JUNG, J., LEE, S., JEON, J., AND CHOI, K. 2001. Behavior-to-placed RTL synthesis with performance-driven placement. In *Proc. Int. Conf. on Computer Aided Design*. 320–326.
- KIM, T. AND LIU, C. 1995a. An integrated data path synthesis algorithm based on network flow method. *Custom Integrated Circuits Conference, 1995.*, *Proc. of the IEEE 1995 1-4*, 615–618.
- KIM, T. AND LIU, C. L. 1995b. A new approach to the multiport memory allocation problem in data path synthesis. *Integr. VLSI J.* 19, 3, 133–160.
- KUSSE, E. AND RABAAY, J. 1998. Low-energy embedded FPGA structures. In *Proc. of the 1998 International Symposium on Low Power Electronics and Design (ISLPED'98)*. New York,, 155–160.

- LEE, C., POTKONJAK, M., AND MANGIONE-SMITH, W. H. 1997. MediaBench: A tool for evaluating and synthesizing multimedia and communications systems. In *International Symposium on Microarchitecture*. 330–335.
- LEE, H.-D. AND HWANG, S.-Y. 1995. A scheduling algorithm for multiport memory minimization in datapath synthesis. In *Proc. of the Asia and South-Pacific Design Automation Conference*. 93–100.
- LIM, K.-H., KIM, Y., AND KIM, T. 2007. Interconnect and communication synthesis for distributed register-file microarchitecture. In *Proc. of the 44th Annual Conference on Design Automation*. San Diego, California, 765–770.
- LUTHRA, M., GUPTA, S., DUTT, N., AND NICOLAU, R. G. A. 2003. Interface synthesis using memory mapping for an FPGA platform. In *Proc. 21st International Conference on Computer Design*. 140–145.
- MANDAL, C. A., CHAKRABARTI, P. P., AND GHOSE, S. 1998. Some new results in the complexity of allocation and binding in data path synthesis. *Computers and Mathematics with Applications* 35, 10, 93–105.
- PANGRLE, B. M. 1991. On the complexity of connectivity binding. *10(11)*, 1460–1465.
- RABAIEY, J. M., CHU, C., HOANG, P., AND POTKONJAK, M. 1991. Fast prototyping of datapath-intensive architectures. *8, 2*, 40–51.
- RIXNER, S., DALLY, W., KAPASI, U., KHAILANY, B., LOPEZ-LAGUNAS, A., MATTSON, P., AND OWENS, J. 1998. A bandwidth-efficient architecture for media processing. In *Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture*.
- RIXNER, S., DALLY, W. J., KHAILANY, B., MATTSON, P. R., KAPASI, U. J., AND OWENS, J. D. 2000. Register organization for media processing. In *Proc. of the 6th International Symposium on High-Performance Computer Architecture*. 375–386.
- SEZNEC, A., TOULLEC, E., AND ROCHECOUSTE, O. 2002. Register write specialization register read specialization: A path to complexity-effective wide-issue superscalar processors. In *Proceedings of the 35th Annual ACM/IEEE International Symposium on Microarchitecture*.
- SINGH, A., PARTHASARATHY, G., AND MAREK-SADOWSKA, M. 2002. Efficient circuit clustering for area and power reduction in FPGAs. *ACM Trans. Design Automation of Electronic Systems* 7, 4, 643–663.
- SRIVASTAVA, M. B. AND POTKONJAK, M. 1995. Optimum and heuristic transformation techniques for simultaneous optimization of latency and throughput. *3, 1*, 2–19.
- STOK, L. 1992. False loops through resource sharing. In *Proc. of the 1992 IEEE/ACM international conference on Computer-aided design*.
- STOK, L. AND PHILIPSEN, W. 1991. Module allocation and comparability graphs. *IEEE International Symposium on Circuits and Systems 11-14 vol.5*, 2862–2865.
- WONG, J. L., MEGERIAN, S., AND POTKONJAK, M. 2002. Forward-looking objective functions: concept & applications in high level synthesis. In *Proc. of the 39th Conference on Design automation (DAC'02)*. ACM Press, New York, NY, USA, 904–909.
- XILINX. *Xilinx Website*. <http://www.xilinx.com>.

Received ;