LETTER

Communication Synthesis for Interconnect Minimization Targeting Distributed Register-File Microarchitecture

Juinn-Dar HUANG[†], *Member*, Chia-I CHEN[†a)], Yen-Ting LIN[†], *and* Wan-Ling HSU[†], *Nonmembers*

SUMMARY In deep-submicron era, wire delay is becoming a bottleneck while pursuing even higher system clock speed. Several distributed register (DR) architectures have been proposed to cope with this problem by keeping most wires local. In this article, we propose a new resource-constrained communication synthesis algorithm for optimizing both inter-island connections (IICs) and latency targeting on distributed register-file microarchitecture (DRFM). The experimental results show that up to 24.7% and 12.7% reduction on IIC and latency can be achieved respectively as compared to the previous work.

key words: *communication synthesis, distributed register-file microarchitecture, interconnect minimization, resource binding, scheduling*

1. Introduction

As advancing into the deep-submicron (DSM) era, interconnect is becoming one of the most crucial issues for electronic circuit and system designs. The system performance, power, and area are all deeply affected by interconnects, especially for global ones [1]. It is reported that interconnects are responsible for over 50% of the overall dynamic power for a microprocessor in 130 nm technology [2]. Previous studies have also shown that interconnect is overwhelmingly dominating the total area and power in FPGA applications [3].

There are several approaches proposed to deal with the critical issue arisen from long interconnects. Globally-asynchronous locally-synchronous (GALS) designs adopt handshaking protocols for communication over long interconnects [4]. In a synchronous latency-insensitive system (LIS), special pipelining elements, named relay stations, are inserted to break a long interconnect into shorter wire segments for sustaining high operating clock frequency [5]. Furthermore, several types of distributed register (DR) architectures, in which the whole system is divided into several logic clusters, are also broadly studied [6]–[13]. In general, all DR-based architectures try to keep most of interconnects local within a cluster and thus minimize the number of required inter-cluster interconnects for better area and performance result.

The distributed register-file microarchitecture (DRFM) is one of the DR-based architectures and is recently proposed in [6], [7], which takes full advantage of those platforms with a rich set of distributed embedded memory blocks. The overall DRFM architecture and the island architecture are shown in Fig. 1. A DRFM instance is composed of multiple islands. Each island contains input routing logic, local register file, and functional units (FUs). The local register file is used to store computation results produced by internal FUs. It is also responsible for feeding data operands to internal FUs and external FUs located in other islands. While utilizing DRFM, one should be aware that how to map operations of a target system into islands can have a significant impact on the final outcome in terms of area and performance [6], [7]. Hence, developing an intelligent synthesis algorithm targeting DRFM is important and needs extensive studies further.

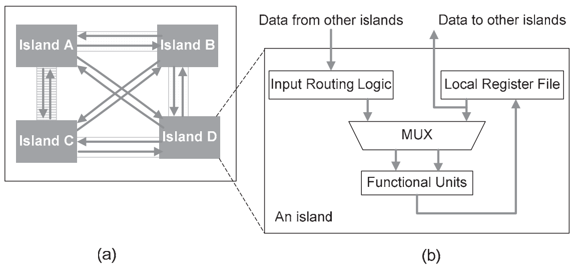

Fig. 1 (a) The DRFM architecture, and (b) the island architecture in DRFM.

Given a scheduled data-flow graph (DFG) of a target system and number of available islands, synthesis for DRFM is considered as a resource-constrained resource binding problem in [6], [7]. The notion of *inter-island connection* (IIC) is presented to better estimate the actual cost of global interconnects. It also demonstrates that the number of IICs after synthesis is highly correlated with the final area size and system performance. Hence the number of IICs can be regarded as an evaluating metric for quality of result (QoR) at early design phases.

In addition, an iterative two-step synthesis approach for IIC minimization is also proposed in [6], [7]. Though the corresponding results show 50% area reduction along with 7.8% performance improvement, a better approach can still be anticipated because the given input DFG is already scheduled in the first place and is not allowed being altered throughout the entire synthesis process, which unavoidably prevents certain optimization opportunities. Meanwhile, [8] also tries to minimize the number of IICs by a two-stage scheme. Given a timing constraint, it first partitions operation nodes into islands based on their connectivity then per-

Manuscript received August 2, 2010.
Manuscript revised October 31, 2010.

[†]The authors are with the Department of Electronics Engineering and Institute of Electronics, National Chiao Tung University, Hsinchu, Taiwan, R.O.C.
a) E-mail: cichen.ee94g@nctu.edu.tw

DOI: 10.1587/transfun.E94.A.1151

forms scheduling. Then it employs *data-forwarding* through idle islands (i.e., *bubbles*) to resolve data access conflicts among data transfers, if any. That is, data transfers intentionally detour via idle islands before reaching their destinations to avoid access conflicts. However, unlike [6], [7], the problem dealt in [8] is classified as a timing-constrained resource binding problem instead of a resource-constrained one.

In this article, we propose a new resource-constrained resource binding algorithm for both IIC and performance optimization targeting DRFM. Given a resource constraint (i.e., number of available islands), the proposed algorithm applies an iterative binding-then-rescheduling process first, and then invokes an access conflict removal procedure. At each control step (cstep), operation nodes scheduled at the current cstep are appropriately assigned to islands first, and then rescheduling is applied to expand the solution space so that a better synthesis output can be produced. The rescheduling and rebinding process also tries to minimize data access conflicts due to limited read ports at the same time. Finally, an access conflict removal procedure is invoked to ensure that no data access conflicts are left at the end of the proposed algorithm. The experimental results confirm that our algorithm does produce better outcomes with 21.0–24.7% fewer IICs and about 12% fewer control steps on average than the prior art.

The rest of this article is organized as follows. Section 2 presents the motivations of this work. Section 3 details our proposed synthesis algorithm for DRFM. The experimental results and analyses are then given in Sect. 4. Finally, the concluding remarks are given in Sect. 5.

2. Motivations

Here we reveal two key observations. First, the solution space is quite limited in [6], [7] since the given scheduled DFG is not allowed being altered, which is also mentioned in [8]. Given a scheduled DFG as shown in Fig. 2(a), if there are two available islands (I_A and I_B) and the given DFG cannot be altered, then the optimal synthesis (i.e., resource binding) result is presented in Fig. 2(b). The (operation) nodes within the same shaded region are mapped into the same island. Apparently, the solution in Fig. 2(b) needs two IICs. However, if rescheduling is allowed, a better solution can be obtained as shown in Fig. 2(c), where only one IIC is demanded. Note that the total number of required control steps in the new solution remains unchanged, which means the IIC reduction does not come from a tradeoff with system performance.

Furthermore, the number of write port of a local register file is restricted to one but no restriction is put on the number of read ports in the original DRFM. However, it is not practical that the number of read ports is assumed unlimited since a register file with a large number of read ports is both slow and area-consuming. Assume that each local register file possesses only two read ports, then as shown in Fig. 3(a), access conflicts on read port occur at cstep 4 be-

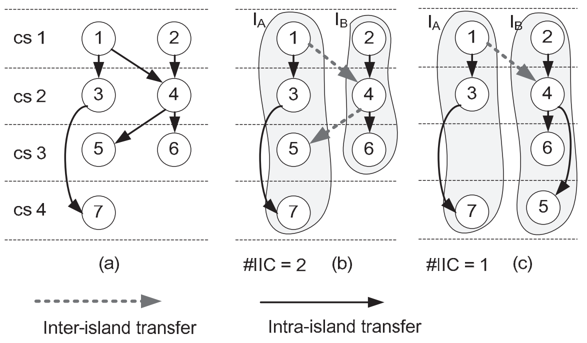

Fig. 2 (a) The scheduled DFG, (b) the scheduled and bound DFG, and (c) the scheduled and bound DFG after rescheduling.

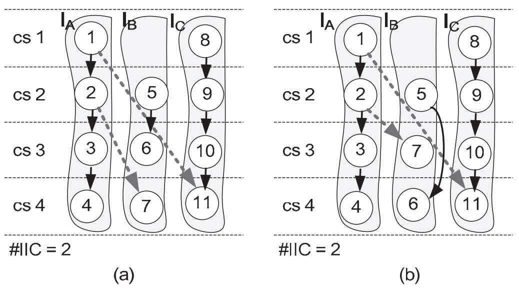

Fig. 3 (a) A scheduled and bound DFG with access conflicts, and (b) another scheduled and bound DFG without access conflicts.

cause there are three data transfers want to access the registers in island I_A — the data transfer $DT_{3,4}$, $DT_{2,7}$ and $DT_{1,11}$. Apparently, at least one of these three read accesses has to be postponed, which consequently increases the latency of the DFG from four to five. The approach in [6], [7] deals with this problem by *data-forwarding*, which adds buffers into the input routing logic of every island. Namely, the requested data item is delivered to the destination island in advance and then stored in input buffers to avoid the read port congestion. In that approach, each data-forwarding consumes one read port and takes one cstep. However, incorporating read port restriction during scheduling and binding can minimize those read access conflicts without increasing hardware cost (i.e., input buffers). Another DFG, as shown in Fig. 3(b), demonstrates that it is possible to keep the latency still four without introducing any data access conflicts if the read port restriction is properly deliberated.

3. Proposed Algorithm

The problem formulation of this work is as follows: *Given a DFG and a resource constraint* (*the number of islands*), *obtain a scheduled and bound DFG with the minimized latency as well as minimize the number of required IICs*.

The overall flow of the proposed method is shown in Fig. 4. Given a DFG, list-scheduling is first performed to obtain an initial scheduling result and followed by the iterative cstep-by-cstep binding-then-rescheduling process. In each iteration, two operations, *island assignment* (binding) and *IIC refinement* (rescheduling and rebinding), are applied

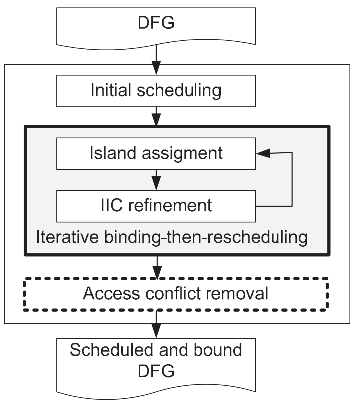

Fig. 4 The overall flow of the proposed algorithm.

consecutively. At the *k*th iteration in cstep-by-cstep fashion, island assignment first binds the operations scheduled in the *k*th control step into the partially scheduled and bound DFG (from the first to the $(k-1)$th control step); IIC refinement then reschedules and rebinds the DFG from the first to the *k*th control step entirely to potentially produce a more globally-optimized outcome. The way used for island assignment in this work is similar to the *horizontal assignment* adopted in [6], [7]. Namely, island assignment is formulated as a *minimum-weighted bipartite matching problem*, where a weight on an edge represents the number of extra IICs induced by the corresponding matching. However, the aforementioned algorithm does not allow rescheduling and generally produces a locally optimized solution. Hence, an IIC refinement process is proposed to look for a better result from the expanded solution space via rescheduling. More details are described in Sect. 3.1. Meanwhile, the IIC refinement procedure is also capable of handling the read port restriction. The details are given in Sect. 3.2. After the iterative phase, an access conflict removal process is followed to eliminate all remaining access conflicts, if any. The procedure simply postpones any conflict accesses that cannot be removed in the previous iterative binding-then-rescheduling process. In the very end of the proposed flow, a scheduled and bound DFG with minimized IICs is derived.

3.1 IIC Refinement

In this article, a special node (in black) called *bubble*, is inserted to explicitly indicate that the corresponding island is idle at that specific cstep. As depicted in Fig. 5(a), the two bubbles *a* and *b* suggest that I_B is idle at cstep 2 and 4.

The proposed IIC refinement process improves the *vertical refinement* developed in [6], [7]. Both of them are based on KL algorithm [14], which is broadly used in partitioning-related problems. Within the process, nodes and bubbles are swapped for IIC minimization. A swap can be made between two nodes or between a node and a bubble. A swap is considered feasible only on two conditions: (i) nodes must be unlocked, and (ii) data dependency must be preserved after swapping. A feasible swap pair of

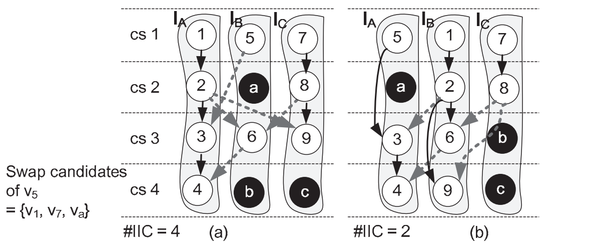

Fig. 5 (a) The DFG at the beginning of the iteration, and (b) the DFG at the end of the iteration.

node *u* and node/bubble *v* is denoted as (u, v). The gain of a swap pair is defined as how many IICs it can reduce, i.e., the difference between the numbers of IICs before and after the swap. The gain of a swap pair (u, v) is denoted as $g_{u,v}$. All feasible swap pairs are collected into the feasible swap pair set (FSPS). After performing an actual swap, FSPS and gains of swap pairs are updated accordingly. The key steps of IIC refinement are described as follows:

(i) Set all operation nodes unlocked.
(ii) Find a swap pair with the largest gain from FSPS.
(iii) Swap the pair then lock the operation node.
(iv) Update FSPS and recalculate the gains of pairs in FSPS.
(v) Repeat (ii) to (iv) until FSPS is empty.
(vi) Keep the fist *k* swaps and undo the rest if the partial gain sum of the first *k* swaps is the largest and positive; go to (i).
(vii) Otherwise, terminate IIC refinement.

For example, a partially scheduled and bound DFG is shown in Fig. 5(a) with an IIC number equal to four. Initially, the gains of all feasible swap pairs in FSPS are calculated. Then the swap pair (9, *b*) is selected to be swapped and node 9 is locked after the swap. The FSPS and the gains are therefore needed to be further updated accordingly. This process is not terminated until FSPS is empty. As a result, only the first three swaps, including $(9, b)$, $(1, 5)$ and $(2, a)$, are actually desired. The resultant DFG at the end of this iteration is shown in Fig. 5(b) and it merely requires two IICs instead of four in Fig. 5(a).

3.2 Coping with Read Port Restriction

As illustrated in Fig. 3, considering the read port restriction during scheduling and binding can effectively minimize access conflicts without increasing hardware cost. Therefore, an augmented IIC refinement process is further presented in this subsection.

During IIC refinement, a secondary gain $h_{u,v}$ of the swap pair (u, v) is defined as the decreased number of access conflicts once the swap takes place. The number of access conflicts of an island at a specific cstep is calculated as the difference between the number of demanded register-file accesses and the number of read ports a register-file actually owns.

The restriction on read port size of a register-file is set to two in this work. As revisiting the case shown in Fig. 3(a), the primary gain $g_{6,7}$ is zero and the secondary gain $h_{6,7}$ is one because there is one (no) conflict at cstep 4 in island I_A before (after) node swapping.

The second step of IIC refinement described in Sect. 3.1 is therefore modified as follows: find a swap pair with the largest primary gain from FSPS; if there are many pairs with the same largest primary gain, choose the one with the largest secondary gain. By means of exploiting the extra secondary gain for tie breaking during scheduling and binding, the read port restriction is well deliberated, and access conflicts can thus be minimized.

4. Experimental Results

The proposed method has been implemented in C++/Linux environment and all experiments were conducted on a workstation with an Intel Xeon 3.2 GHz CPU and 4 GB RAM. For fair and comprehensive comparisons, two different synthesis flows are created. Given an input DFG and a resource constraint, list scheduling is first performed to provide an initial scheduling result for both flows. Then, *Flow1* implements the approach proposed in [6], [7], while *Flow2* carries out the proposed approach. The access conflict removal procedure is then conducted as a post-processing for both flows.

The basic information of the test cases (DFGs), which are frequently used in high-level synthesis field, is given in the first three columns in Table 1. Two configurations are deliberated in our experiments — synthesis is performed without (with) a resource constraint in Configuration 1 (2), respectively. In Configuration 1, the number of islands is set as the minimum number that still guarantees the synthesis outcome with the minimum latency obtained from ASAP scheduling; that is, there is in fact no resource constraint at all. However, the assumption about unlimited available hardware resource is impractical in the real world. Hence, in Configuration 2, for every test case, the number of available islands is reduced by half.

Table 1 reports the experimental results without read port restriction, which means that only the primary gain is deliberated during node swapping. The results clearly demonstrate that the proposed algorithm does outperform the prior art. Table 1 also suggests that average IIC reduction in Configuration 2 is better than that in Configuration 1. It is because the number of available islands in Configuration 2 is roughly a half of that in Configuration 1, which also reduces the total number of required IICs in Configuration 2. Therefore, the effect of eliminating an IIC is more significant in Configuration 2 than in Configuration 1.

Table 2 gives the results with the read port restriction, which is set to two for all test cases. The results indicate that Flow2 achieves on average 21.5% and 24.7% IIC reduction in Configuration 1 and 2 respectively as compared with Flow1, which is pretty much the same as the ones with-

Table 1 The experimental results without the read port restriction.

test case	#nodes	#edges	Configuration 1					Configuration 2				
			#island	latency	#IIC (Flow1)	#IIC (Flow2)	#IIC reduction	#island	latency	#IIC (Flow1)	#IIC (Flow2)	#IIC reduction
fir2	40	39	5	11	7	5	28.6%	2	21	2	1	50.0%
fir1	44	43	6	11	8	7	12.5%	3	17	4	3	25.0%
lee	49	62	6	9	11	10	9.1%	3	18	6	5	16.7%
cos	82	91	12	8	27	24	11.1%	6	16	14	12	14.3%
honda	105	104	10	15	17	14	17.6%	5	23	9	8	11.1%
wribmp	106	88	16	7	18	14	22.2%	8	14	14	10	28.6%
dir	127	126	11	15	24	17	29.2%	5	27	11	8	27.3%
chem	342	327	24	15	61	38	37.7%	12	29	41	28	31.7%
fft16	414	672	32	14	204	180	11.8%	16	27	97	83	14.4%
u5ml	564	557	29	26	102	71	30.4%	14	42	55	41	25.5%
Avg.							21.0%					24.5%

Table 2 The experimental results with the read port restriction (= 2).

test case	Configuration 1							Configuration 2						
	latency LB[1]	latency (Flow1)	latency (Flow2)	lat inc[2]	#IIC (Flow1)	#IIC (Flow2)	#IIC red[3]	latency LB[1]	latency (Flow1)	latency (Flow2)	lat inc[2]	#IIC (Flow1)	#IIC (Flow2)	#IIC red[3]
fir2	11	11	11	0.0%	7	5	28.6%	21	21	21	0.0%	2	1	50.0%
fir1	11	12	11	9.1%	8	7	12.5%	17	17	17	0.0%	4	3	25.0%
lee	9	12	9	33.3%	11	10	9.1%	18	22	18	22.2%	6	5	16.7%
cos	8	9	8	12.5%	27	24	11.1%	16	17	16	6.3%	14	12	14.3%
honda	15	16	15	6.7%	17	13	23.5%	23	25	23	8.7%	9	8	11.1%
wribmp	7	8	7	14.3%	18	12	33.3%	14	15	14	7.1%	14	10	28.6%
dir	15	16	15	6.7%	24	18	25.0%	27	29	27	7.4%	11	8	27.3%
chem	15	18	15	20.0%	61	43	29.5%	29	33	29	13.8%	41	26	36.6%
fft16	14	16	14	14.3%	204	184	9.8%	27	41	27	51.9%	97	84	13.4%
u5ml	26	27	26	3.8%	102	69	32.4%	42	46	42	9.5%	55	42	23.6%
Avg.				12.1%			21.5%				12.7%			24.7%

[1]: Lower bound of latency; [2]: Latency increase of Flow1 over Flow2; [3]: #IIC reduction of Flow2 over Flow1

out the read port restriction. Furthermore, as stated before, the process for access conflict removal may increase the latency if there do exist non-removable access conflicts after binding-then-rescheduling. The first three columns for both configurations in Table 2 indicate the lower bound of latency obtained from Table 1 and two resultant latencies given by two different synthesis flows. The proposed Flow2 achieves the minimum latency for all test cases in both configurations while Flow1 increases the average latency by about 12%, which shows that the proposed method can handle the read port restriction better.

Note that the number of inter-island connections (IICs) is different from the number of inter-island transfers (IITs). Multiple IITs can share an IIC as long as they have the same source-destination island pair as well as different arrival times. In general, the number of IITs is commonly used for power estimation of on-chip communication, while the number of IICs is mostly used to estimate the cost of global interconnects. Nevertheless, during synthesis, it is not always possible to reduce both IICs and IITs at the same time; in other word, there is a tradeoff between area/performance (IIC) and power (IIT) optimization. In this article, the proposed synthesis algorithm merely focuses on IIC minimization.

5. Conclusion

The number of IICs has been reported to better model the global interconnect cost and then can be considered as a major QoR evaluation metric at early design stages in DRFM. In this article, we propose a resource-constrained synthesis algorithm for IIC minimization. The iterative binding-then-rescheduling procedure is first utilized for island assignment. A better island binding result can be expected because the solution search space is significantly expanded through rescheduling. The proposed algorithm also incorporates the consideration of read port restriction into scheduling and binding procedures to minimize the potential access conflicts. A post-processing procedure is then conducted to eliminate all remaining access conflicts.

The experimental results indicate that the proposed algorithm reduces the number of IICs by 21.0–24.7% on average as compared to the prior art. While adopting the read port restriction, the proposed method also outperforms the previous work by about 12% in terms of average latency. As a result, the proposed algorithm should be regarded as a better alternative while performing architectural synthesis targeting DRFM.

Acknowledgment

This work was supported in part by the National Science Council of Taiwan under Grant NSC 99-2220-E-009-008.

References

[1] International Technology Roadmap for Semiconductors, Semiconductor Industry Association, 2007.
[2] N. Magen, A. Kolodny, U. Weiser, and N. Shamir, "Interconnect-power dissipation in a microprocessor," Proc. Int'l Workshop System Level Interconnect Prediction, pp.7–13, 2004.
[3] A. Singh, G. Parthasarathy, and M. Marek-Sadowska, "Efficient circuit clustering for area and power reduction in FPGAs," ACM Trans. Des. Autom. Electron. Syst., vol.7, no.4, pp.643–663, Oct. 2002.
[4] D.M. Chapiro, "Globally-asynchronous locally-synchronous systems," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1984.
[5] L.P. Carloni, K.L. McMillan, A. Saldanha, and A.L. Sangiovanni-Vincentelli, "A methodology for correct-by-construction latency insensitive design," Proc. Int'l Conf. Computer Aided Design, pp.309–315, 1999.
[6] J. Cong, Y. Fan, and W. Jiang, "Platform-based resource binding using a distributed register-file," Proc. Int'l Conf. Computer Aided Design, pp.709–715, 2006.
[7] J. Cong, Y. Fan, and J. Xu, "Simultaneous resource binding and interconnection optimization based on a distributed register-file microarchitecture," ACM Trans. Des. Autom. Electron. Syst., vol.14, no.3, pp.1–31, May 2009.
[8] K. Lim, Y. Kim, and T. Kim, "Interconnect and communication synthesis for distributed register-file microarchitecture," Computers & Digital Techniques, IET, vol.3, no.2, pp.162–174, March 2009.
[9] D. Kim, J. Jung, S. Lee, J. Jeon, and K. Choi, "Behavior-to-placed RTL synthesis with performance-driven placement," Proc. Int'l Conf. Computer Aided Design, pp.320–325, 2001.
[10] J. Cong, Y. Fan, G. Han, X. Yang, and Z. Zhang, "Architecture and synthesis for on-chip multicycle communication," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.23, no.4, pp.550–564, April 2004.
[11] C.-I Chen and J.-D. Huang, "A hierarchical criticality-aware architectural synthesis framework for multicycle communication," IEICE Trans. Fundamentals, vol.E93-A, no.7, pp.1300–1308, July 2010.
[12] A. Ohchi, N. Togawa, M. Yanagisawa, and T. Hothuki, "High-level synthesis algorithms with floorplanning for distributed/shared-register architectures," Proc. Int'l Symp. VLSI Design, Automation and Test, pp.164–167, 2008.
[13] S. Gao, K. Seto, S. Komatsu, and M. Fujita, "Pipeline scheduling for array based reconfigurable architectures considering interconnect delays," Proc. Int'l Conf. Field-Programmable Technology, pp.137–144, Dec. 2005.
[14] B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," Bell Syst. Tech. J., pp.291–307, Feb. 1970.