IEICE TRANS. FUNDAMENTALS, VOL.E93–A, NO.7 JULY 2010

PAPER

A Hierarchical Criticality-Aware Architectural Synthesis Framework for Multicycle Communication

Chia-I CHEN[†a)], *Nonmember and* **Juinn-Dar HUANG**[†], *Member*

SUMMARY In deep submicron era, wire delay is no longer negligible and is becoming a dominant factor of the system performance. To cope with the increasing wire delay, several state-of-the-art architectural synthesis flows have been proposed for the distributed register architectures by enabling on-chip multicycle communication. In this article, we present a new performance-driven criticality-aware synthesis framework CriAS targeting regular distributed register architectures. To achieve high system performance, CriAS features a hierarchical binding-then-placement for minimizing the number of performance-critical global data transfers. The key ideas are to take time criticality as the major concern at earlier binding stages before the detailed physical placement information is available, and to preserve the locality of closely related critical components in the later placement phase. The experimental results show that CriAS can achieve an average of 14.26% overall performance improvement with no runtime overhead as compared to the previous art.

key words: *multicycle communication, architectural synthesis, high-level synthesis, performance-driven, criticality-driven*

1. Introduction

As advancing into the deep-submicron (DSM) era, interconnect delay is becoming inevitable due to resistance-capacitance delay, coupling effect, inductance, multiple-gigahertz operating frequency, and so on [1]–[3]. In architectural synthesis, the system clock cycle time is determined by the maximum sum of delay of both functional units (FUs) and associated interconnects. If the delay of long wires (especially for global interconnects) is still neglected in the synthesis flow, unexpected large delay introduced by long wires after physical mapping (floorplanning, placement, and routing) is very likely to make a serious impact on the whole system performance due to lengthened clock cycle time. Therefore, global interconnects have been becoming the performance bottleneck when pursuing higher system speed, which also brings on so-called interconnect-limited VLSI architectures [4]. To overcome this problem, several synthesis flows are proposed to estimate long interconnect delay by applying preliminary floorplanning and thus obtain better synthesis results [5]–[7].

Conventionally, the centralized register (CR) architecture is commonly presumed in high-level synthesis. In a CR-based architecture, an FU is expected to access any register within one clock cycle. Though the device speed generally increases as the manufacturing process advances, the wire delay does not scale as well as the feature size. Global wire delay gradually dominates and significantly lengthens the system cycle time. Hence, previous studies propose several similar distributed register (DR) architectures to overcome this issue [8]–[19]. In a DR-based architecture, the whole system is divided into several clusters and each cluster contains its own local FUs and registers. Consequently, the inter-cluster interconnect delay can be fully isolated from the intra-cluster delay. The latter includes the *local* wire delay within a cluster and is supposed to be finished within one cycle, while the former is the *global* data transfer delay between different clusters and is allowed being completed in multiple clock cycles. Therefore, DR-based architectures can not only alleviate the increase of cycle time due to the long interconnect delay but also enable simultaneous computation and communication.

Though allowing multicycle global data transfer can reduce the impact on system clock speed in a DR-based architecture, performance improvement is still limited by the inaccurate delay estimation of long wires. Therefore, authors in [10] propose the regular distributed register (RDR) architecture and the corresponding synthesis methodology, named multicycle architectural synthesis system (MCAS). Due to the highly regular layout, it is applicable to provide a table of the accurate interconnect delay between each cluster pair in this architecture. With this lookup table, long wire delay can be estimated in a very precise fashion. MCAS takes the behavioral design specification as the input. Then, it goes through resource allocation, FU binding, scheduling-driven placement and post-layout scheduling with re-binding procedures to get the synthesis result (RTL description and the corresponding physical mapping information). During FU binding, MCAS tries to minimize the number of all potential global data transfers (pGDTs) while only timing-critical global transfers can actually degrade the performance. That is, MCAS does not consider the time criticality of the transfer when performing FU binding. Moreover, its fine-grained FU-level placer fails to preserve the locality of related critical FUs due to the inherent unstable nature of any simulated annealing (SA) engine. In short, though MCAS can already do a reasonably good job, there is still room for further performance improvement.

In this article, we propose a hierarchical performance-driven criticality-aware architectural synthesis flow, named CriAS, targeting the RDR-based architecture family [10]–[16]. CriAS features a hierarchical binding-then-placement

Manuscript received November 12, 2009.

[†]The authors are with the Department of Electronics Engineering and the Institute of Electronics, National Chiao Tung University, Hsinchu, Taiwan, R.O.C.

a) E-mail: cichen.ee94g@nctu.edu.tw

DOI: 10.1587/transfun.E93.A.1300

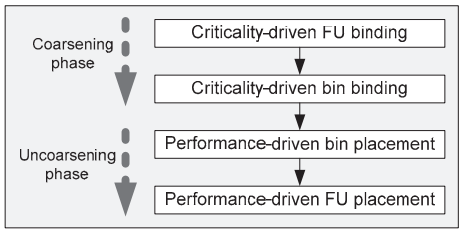

Fig. 1 The hierarchical binding-then-placement scheme.

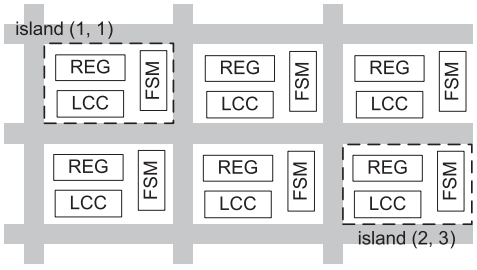

Fig. 2 The RDR-based architecture.

methodology, where an intermediate transformation called *bin level* is introduced. A bin is a fixed-capacity container of FUs. CriAS is aware that a global transfer lying on a critical path can potentially induce extra latency once the FU placement is determined later. Hence, unlike MCAS, CriAS takes the criticality into consideration to minimize the number of potential *critical* global data transfers (pCGDTs) instead of pGDTs at early binding stages, where the FU placement information is still not available yet. In CriAS, critical operations tend not to be bound into different FUs so that critical global data transfers can be minimized. Similarly, FUs lying on critical paths tend not to be bound into different bins to maximize the performance, where the bins are designated to preserve the locality of related critical FUs. After hierarchical binding, CriAS performs simulated annealing (SA)-based coarse-grained bin placement to finalize the locations of bins (so as FUs). Since FUs within a bin remain together during placement, the locality of closely related critical FUs can thus be well preserved and a better result can be expected. Then a fine-grained FU placement is performed to further improve the placement quality at the FU level due to higher FU mobility. The hierarchical scheme is shown in Fig. 1 — FU binding and bin binding form the coarsening phase while the uncoarsening one is composed of bin- and FU-level placement. The experimental results show that CriAS does provide better synthesis outcomes with higher performance than the prior art.

The rest of this article is organized as follows. The RDR-based architectures and MCAS procedures are briefly introduced in Sect. 2. Section 3 presents the key observations and motivations of our work. The proposed synthesis flow CriAS is then described in Sect. 4, followed by the experimental results in Sect. 5. Finally, the concluding remarks are given in Sect. 6.

2. Preliminaries

2.1 RDR-Based Architecture

Synthesis flows targeting DR-based architectures can be classified according to the interconnect delay models they adopt. The synthesis task is relatively easier with *zero* inter-cluster delay; however this delay model appears oversimplified [17], [18]. The synthesis flow considering *unit* inter-cluster delay makes a move toward reality but still not close enough [20]. To be even more practical, the delay model must be geometry-aware. However, with such an interconnect delay model, synthesis task is inherently more complicated and accuracy of delay estimation does affect the quality of final synthesis result deeply.

Since the inaccurate delay estimation of long wires impacts the system cycle time, a regular distributed register architecture and its corresponding synthesis methodology, RDR/MCAS, is proposed in the first place to solve this problem [10]. In an RDR-based architecture, a chip is partitioned into a two-dimensional array of *islands*. Figure 2 illustrates a 2×3 RDR-based architecture. Each island consists of a local register file (REG), a control finite state machine (FSM), and a configurable logic computational cluster (LCC) which can implement arbitrary random logic and datapaths. The size of an island is determined to ensure that all local computation and communication can be completed in a single clock cycle. On the other hand, global data transfers, which deliver data from one island to the other, are allowed taking multiple clock cycles. Hence the interconnect delay between two clusters can be easily estimated by in-between Manhattan distance. As shown in Fig. 2, a global data transfer between *island*(1, 1) and *island*(2, 3) takes multiple (three) cycles. The notion of on-chip multicycle communication makes traditional architectural synthesis a much more difficult work. It is because whether a data transfer is local or global can only be resolved after placement is completed, but the latency of a transfer must be available at the scheduling stage, which is before placement. The first dedicated synthesis flow MCAS, briefly described in Sect. 2.2 later, is therefore proposed to deal with this problem.

There are two similar variants of the original RDR, RDR-Pipe and RDR-GRS, which can further reduce the required interconnect resource. More details about these two variants can be found in [13]–[15]. Nevertheless, the synthesis algorithm proposed in this article is applicable to all the RDR-based architectures.

2.2 MCAS Flow

MCAS targets the RDR architecture and the overall flow is shown in Fig. 3. The inputs consist of the behavioral design description in synthesizable C/VHDL and the architecture specification. MCAS first generates the control/data flow graph (CDFG) from the given behavioral description through SUIF infrastructure [21] and Machine

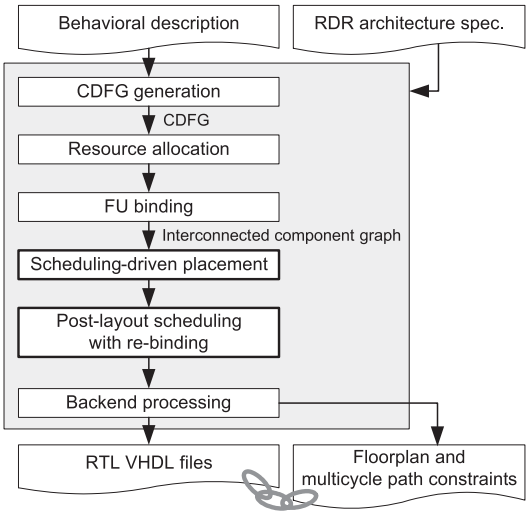

Fig. 3 The overall flow of MCAS.

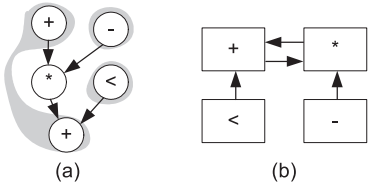

Fig. 4 (a) A bound DFG, and (b) the corresponding ICG.

SUIF [22]. Resource allocation based on force-directed scheduling (FDS) [23] is then performed to minimize the resource usage without violating the timing constraint. After resource allocation, the algorithm proposed in [8] is employed for functional unit binding to minimize the number of potential global data transfers. After FU binding, an interconnected component graph (ICG) is derived through the bound CDFG. The ICG describes all the data transfers among different FUs. Figure 4 gives an example of bound DFG and the corresponding ICG. The kernel of MCAS consists of scheduling-driven placement and post-layout scheduling with re-binding. Scheduling-driven placement employs an SA-based placer considering not only approximated wirelength but also system performance [24], [25]. Given physical location information, placement-driven scheduling with re-binding further condenses the system latency. The algorithm is based on a force-directed list-scheduling (FDLS) framework [23] and the idea of dynamic critical path scheduling [9], [26]. The backend procedures include register/port assignment and datapath/FSM generation. In the end of the MCAS flow, synthesizable VHDL RTL code with corresponding physical placement information is generated.

3. Motivations

The key observations and motivations of this article arise from the deficiency in FU binding as well as the use of SA-based placer at the fine-grained FU level in MCAS. The details are described in Sects. 3.1 and 3.2, respectively.

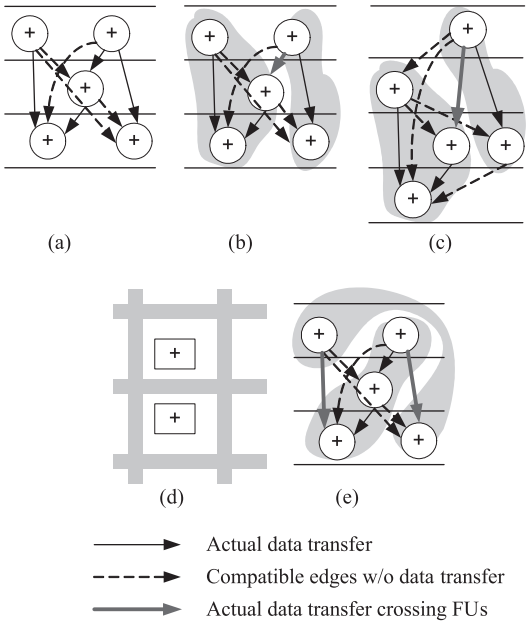

Fig. 5 (a) A scheduled DFG, (b) FU binding result in [8] and [10], (c) modified schedule for (b), (d) an example placement, and (e) another FU binding result.

3.1 FU Binding

Since the location information is unknown yet, the FU binding algorithm proposed in [8] then adopted by MCAS [10] is designed to minimize the number of potential global data transfers (pGDTs) crossing islands. Given a scheduled DFG, a weighted compatibility graph is built. A high/low weight is assigned to a pair of compatible operations if there is a/no data transfer between them. Then the maximum weighted cliques that cover the graph are identified as the binding result. An example scheduled DFG with compatible edges is shown in Fig. 5(a), in which the solid lines representing the compatible edges with actual data transfers (high weights) and the dotted lines representing the compatible edges without data transfers (low weights). Figure 5(b) shows the outcome with the shaded zones after applying FU binding in [8] and [10]. The result is quite good in terms of the given cost function because only one data transfer needs to cross two different FUs, thus the number of pGDTs is merely one. However, even if these two FUs are placed in two neighboring islands as shown in Fig. 5(d), one extra clock cycle is required to complete that global data transfer. Hence the original schedule has to be modified as the one shown in Fig. 5(c) to fulfill the updated timing constraint. Unfortunately, the new schedule takes one more control step because that global transfer exactly lies on the critical path. Consider another feasible binding result shown in Fig. 5(e), it does not look so good at first because two data transfers can be global (i.e., the number of pGDTs is two). However, given the same placement, it needs no ex-

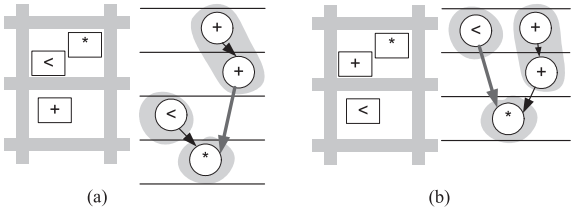

Fig. 6 Two different placements with the corresponding schedules for the same bound DFG.

tra control step to get a valid schedule because those two global data transfers are non-critical. Through this example, it should be noticed that the criticality of global data transfers is much more important than the total number of global data transfers while performing FU binding. Thus our proposed framework tends to minimize the number of potential critical global data transfers (pCGDTs) instead of pGDTs.

3.2 FU-Level Placer

The scheduling (performance)-driven placement procedure in MCAS is performed by an SA-based placer operating only at the FU level. That is, the atomic operating element is a fine-grained single FU. Though the placer itself is performance-driven, it is very hard to prevent those FUs that should stay together from being separated due to the inherent unstable nature of any SA engine. Figure 6 illustrates two different FU placement results with the corresponding schedules for the same bound DFG. The result shown in Fig. 6(a) is worse than that in Fig. 6(b) because the former locates the adder and the multiplier in different islands and thus turns the critical data transfer between them global. To avoid this situation, a hierarchical placement strategy is introduced. An FU container, named bin, is defined. It should be a good idea to pack those closely related critical FUs into a bin in the first place. Then the following SA-based placer is intended to operate at the coarse-grained bin level. As a result, the placer can only decide the locations of bins and the desired strong locality among related critical FUs within a bin is well preserved. Afterward a fine-grained FU-level placer, which takes the result of bin-level placement as the starting point, is applied. The FU-level placer is expected to further improve synthesis results since a larger solution space can be explored without the bin restriction.

4. Criticalities-Aware Synthesis

In this section, we present our hierarchical performance-driven criticality-aware architectural synthesis flow, named CriAS, which targets the RDR-based architecture family. As mentioned, CriAS features a hierarchical binding-then-placement strategy to facilitate higher system performance. The first idea is to minimize the number of pCGDTs at earlier synthesis stages. The advantage of considering pCGDTs instead of pGDTs is that criticality is simply a better performance metric before the placement detail is available, as demonstrated in Sect. 3. The criticality-driven hierarchical binding strategy consisting of both FU-level and bin-level binding is proposed to carry out this idea, where a *bin* is a fixed-capacity container of FUs. The other key idea is that placement is performed at two different levels hierarchically instead of the fine-grained FU level only, which keeps related critical FUs staying as close as possible.

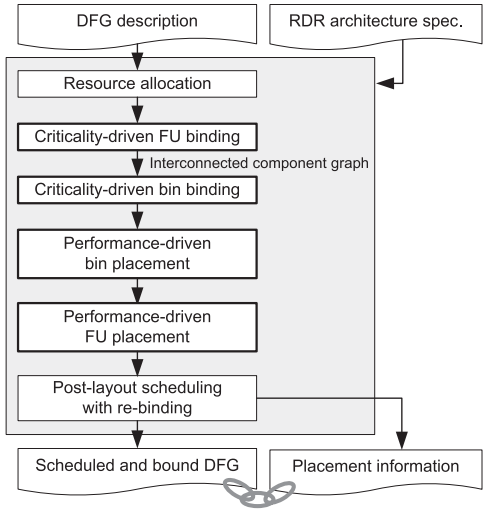

Fig. 7 The overall flow of CriAS.

Figure 7 shows the overall flow of CriAS. The input of CriAS contains the given DFG and the RDR architecture specification. After processing the input DFG, FDS is performed for resource allocation and the initial scheduling result can then be obtained. Based on the initial schedule, the criticality-driven FU binding is applied for minimizing the number of pCGDTs, which is likely to boost the system performance. The compatibility graph with proper edge weight settings is built first based on the criticality of the edges. Then, the FU binding problem can be further modeled as a *min-cost flow* problem, and thus optimally solved accordingly. More technical details are discussed in Sect. 4.1.

As mentioned in Sect. 3.2, the closely related critical FUs can be packed into a bin first before starting SA-based placement. A bin here is defined as an FU container whose capacity is set as the same as the island capacity. In the bin binding process, the directed multigraph edges originally used to describe multiple data transfers between two FUs in an interconnected component graph (ICG) are replaced by a single undirected edge between two of them. Again, the edge weights are determined based on the criticality of the edges. The bin binding problem can then be formulated as a *capacity-constrained k-way min-cost partitioning* problem, and thus solved accordingly. After the hierarchical binding procedures, performance-driven SA-based placers at the coarse-grained bin level (so-called bin placement) and then the fine-grained FU level (so-called FU placement) are performed. These procedures are detailed in Sects. 4.2 and 4.3.

After the physical location information is available, the scheduling and binding information should be updated accordingly to preserve the validity of synthesis result. Then,

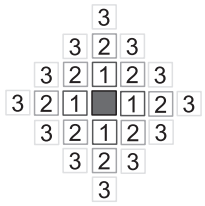

Fig. 8 The Manhattan distance label from the central island.

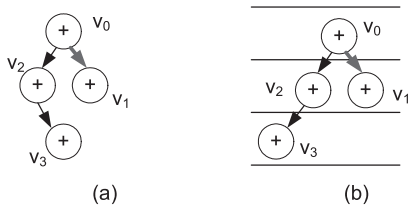

Fig. 9 (a) A DFG before scheduling, and (b) after scheduling.

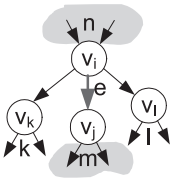

Fig. 10 The number of affected paths of the edge e.

a similar post-layout scheduling with re-binding procedure proposed in [10] is also performed here. The output of CriAS consists of a scheduled/bound DFG as well as the corresponding physical placement information of all FUs.

4.1 Criticality-Driven FU Binding

As mentioned above, the criticality of global data transfers should be the main concern while performing FU binding. To minimize the number of pCGDTs, an edge-weighted compatibility graph is built first. The weight of an edge e, $criticality(e)$, is defined as (1).

$$criticality(e) = \begin{cases} 0 & \text{not a data transfer} \\ \alpha \cdot lf(e) + \beta \cdot sr(e) + \gamma \cdot ap(e) & \text{a data transfer} \end{cases} \quad (1)$$

In (1), $criticality(e)$ is determined by the weighted sum of three terms — the first one represents the location flexibility; the second represents the reciprocal of the available slack; and the last represents the number of affected paths. α, β and γ are adjustable weighting factors but should be properly chosen to ensure that the prior term dominates the subsequent one. Conceptually, $criticality(e)$ is designed to indicate the severity level of the performance impact if the edge e becomes a global data transfer. In other words, the higher the weight of a global data transfer is, the worse the overall performance can be.

The location flexibility of an edge e, $lf(e)$, is defined as (2), where $cstep(v)$ represents the scheduled control step of a node v.

$$lf(e) = \frac{1}{\left[cstep(v_j) - cstep(v_i)\right]^2} \quad (2)$$

The location flexibility conceptually represents how hard to locate the two related operations, v_i and v_j, without violating the timing constraint. Assume the difference between $cstep(v_j)$ and $cstep(v_i)$ is n and v_i is located at the central island in Fig. 8, there is no timing violation if v_j can be placed into any island with the distance label $\leq n$. The feasible locations for v_j forms a diamond region of islands and its size is proportional to n^2. In other words, it gets more difficult to meet the timing constraint during placement as the location flexibility, which is inversely proportional to n^2, gets larger. Note that the performance can very likely be degraded if a high-criticality data transfer becomes global.

Next, by definition, an operation should have no mobility in a given scheduled DFG. However, operations not lying on critical paths do have certain mobility before scheduling. Figure 9 depicts a DFG before and after scheduling. According to Fig. 9(a), the operation (vertex) v_1 is originally not critical. That is, the performance remains the same even if v_1 is scheduled to the third control step as shown in Fig. 9(b). However, (2) gives the same weight for all three data transfers while the data transfer between v_0 and v_1 is actually not as critical as the other two. Therefore, the reciprocal of the slack on an edge e, $sr(e)$, is defined as (3) and is intended to point out this difference.

$$sr(e) = \frac{1}{slack(e)} \quad (3)$$

Conventionally, the slack on a vertex comes from the difference of scheduled control step between *as late as possible* (*ALAP*) scheduling and *as soon as possible* (*ASAP*) scheduling [27]. With the same idea, here we extend the idea of slack to the edge. The term *as short as possible* (*ASAPe*) of an edge $e(v_i, v_j)$, defined in (4), represents the minimal possible length the edge can have. Similarly, the term *as long as possible* (*ALAPe*) of an edge $e(v_i, v_j)$, defined in (5), represents the maximal possible length the edge can have. The suffix letter e in both terms is used to avoid possible confusion between the edge version and the vertex one. As a result, the slack on an edge e is defined as the difference between $ALAPe(e)$ and $ASAPe(e)$, and shown in (6). Again, an edge e with a smaller slack (larger $sr(e)$) is more performance-critical.

$$ASAPe(e) = \max\left(0,\ \ ASAP(v_j) - ALAP(v_i) - 1\right) \quad (4)$$

$$ALAPe(e) = ALAP(v_j) - ASAP(v_i) \quad (5)$$

$$slack(e) = ALAPe(e) - ASAPe(e) \quad (6)$$

At last, $ap(e)$ defined in (7) is used to indicate how many paths are affected by how the edge $e(v_i, v_j)$ gets processed. Figure 10 gives such an example. Conceptually, an edge e with larger $ap(e)$ should be handled with more care since the result can affect more timing paths.

$$ap(e) = in\text{-}degree(v_i) \times out\text{-}degree(v_j) \quad (7)$$

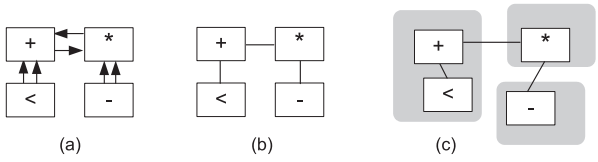

Fig. 11 (a) An example ICG, (b) the corresponding modified ICG, and (c) an IBG for a possible bin binding solution.

After properly setting the criticality weights of all edges in the compatibility graph, the FU binding problem is formulated as finding a solution in which the weight sum of all pCDGTs is minimal. This FU binding problem can already be optimally solved by the min-cost flow algorithm described in [28]. In the end, the resultant ICG is derived as the output of FU binding.

4.2 Criticality-Driven Bin Binding

As mentioned, it may not be a good idea to merely perform placement at the fine-grained FU level due to the inherent unstable nature of an SA-based placer. Alternatively, a set of closely related performance-critical FUs can be first clustered into a coarse-grained bin as an atomic operating element during placement. A bin is defined as a fixed-capacity container of FUs for preserving the locality. The capacity of a bin is set identical to that of an island, which constrains the number of FUs can be packed into a bin. The process about how to validly pack FUs into bins is named *bin binding*.

Here, we present a criticality-driven bin binding approach. At the beginning, the ICG $G(V, E)$, which enumerates all data transfers between FUs and is represented as a directed multigraph, is transformed into an edge-weighted undirected simple graph, named the *modified ICG* $H(V, F)$. H has the same vertex set of G. An undirected edge connecting two vertices (FUs) in H implies that there exists at least one directed edge between those two vertices (regardless of the direction) in G. The weight of an edge $f \in F$ is defined as the sum of location flexibility of all edges $e \in E$ that are mapped to f during G-to-H transformation, and is shown in (8) where $lf(e)$ is identical as (2). Conceptually, if two FUs are connected by an edge with high weight, they should be packed into the same bin for minimizing the number of pCGDTs.

$$w(f) = \sum_{\forall e \in E \text{ mapped to } f \in F} lf(e) \tag{8}$$

Figures 11(a) and 11(b) gives an example ICG and the corresponding modified ICG. After properly setting edge weights, the bin binding problem can be formulated as the capacity-constrained k-way min-cost partitioning problem and thus solved accordingly by an algorithm similar to that proposed in [29]. Note that each FU node has a weight indicating the (hardware) resource usage, thus the sum of node weights of FUs that are packed into the same bin cannot exceed the given bin capacity. That is why the partitioning is capacity-constrained. The bin binding result is indicated

```
Partitioning ( H ) {  // bin binding
// input: a modified ICG H(V, F)
1.  Set V_B = {} ; // a set of bins
2.  while( ! V.empty() ) {
3.    Set B = {} ; // a set of nodes (a bin)
4.    Pick the max weighted net f_max in F ;
5.    Add the two incident nodes into B ;
6.    Remove the two incident nodes from V ;
7.    Modify the network ;
8.    while( ! B.full() and ! V.empty() ) {
9.      Pick the max weighted net f_max
          connected to nodes in B;
10.     Add the incident node to B ;
11.     Remove the incident node from V ;
12.     Modify the network ;
13.   } // end of inner while
14.   Add B into V_B;
15. } // end of outer while
16. Build an edge set E_B in which every edge
      implies a connected bin pair ;
17. return G_B ;
18. // output: an IBG G_B = (V_B, E_B) }
```

Fig. 12 The pseudo code of capacity-constrained k-way min-cost partitioning algorithm for bin binding.

using an interconnected bin graph (IBG). For example, a possible IBG for the modified ICG shown in Fig. 11(b) is depicted in Fig. 11(c). The pseudo code of our partitioning algorithm is given in Fig. 12. At the end of this procedure, all FUs are partitioned into a set of bins, which serve as the input to the next stage — bin placement.

4.3 Hierarchical Performance-Driven Placement

The hierarchical performance-driven placement consists of a coarse-grained bin-level placement and a fine-grained FU-level placement, which regard a bin and an FU as an atomic component during operation, respectively. Like MCAS, both of our performance-driven placers are SA-based. Within the kernel of our SA-based placers, list-scheduling-based timing analysis on the bound DFG is performed first. After timing analysis, the more critical a net is, the higher the weight is assigned to the net. Then the placers try to locate those bins/FUs connected by heavily-weighted nets as close as possible. Meanwhile, more technical details about the SA-based placer kernel can be found in [10], [24], [25].

The bin-level placer actually performs mapping between bins and islands. As mentioned, the capacity of a bin is set identical to that of an island, thus the procedure for placing bins into islands is actually a one-to-one mapping. Next, the FU-level placer follows the bin-level placer. It takes the result produced by the bin-level placer as the initial solution and merely fine-tunes the placement outcome instead of deriving a whole new one. Note that the major difference here is that an FU instead of a bin is considered as an atomic operation. In the beginning of FU-level placement, bins are 'unpacked' and FUs originally packed into the same bin can then be independently relocated. In other words, FUs are allowed to be freely moved across islands

without the bin constraint anymore.

5. Experimental Results

5.1 Experimental Environment Setting

Our CriAS system has been implemented in C++/Linux environment on a workstation with an Intel Xeon 2 GHz CPU and 14 GB RAM. The target RDR-based architecture consists of M × N islands. The value of (M, N) pair is dynamically adjustable and depends on the size of the input case. In our experiments, given a test case, M and N are selected to keep the overall island utilization between 70–80% and to make the aspect as square as possible. Meanwhile, the parameter α, β and γ, which should be properly chosen to ensure that the prior term dominates the subsequent one as mentioned, are set to 10^5, 100, and 2, respectively, for all the test cases.

For fair and comprehensive comparisons, three different synthesis flows are implemented, as shown in Fig. 13. Flow1 is devoted to mimic the original MCAS [10]; Flow2 is the same as Flow1 except for performing the criticality-driven FU binding instead; and the proposed CriAS is referred as Flow3. The comparisons between Flow1 and Flow2 disclose how effective the criticality-driven FU binding is; the comparisons between Flow2 and Flow3 reveal how well the bin binding strategy and hierarchical placement can do; and the comparisons between Flow1 and Flow3 demonstrate the difference of overall synthesis quality between MCAS and CriAS.

The test cases are chosen from different benchmark sets [30]–[32], which are frequently adopted for evaluation purpose. The basic information of these test cases (DFGs) is given in Table 1. The first column lists the name of the test case; the second and third columns describe the numbers of nodes and edges, respectively; the dimension parameters (M, N) and the overall resource utilization are shown from the fourth column to the sixth. The last column reports the minimum possible latency obtained by ASAP scheduling, which considers no resource constraint.

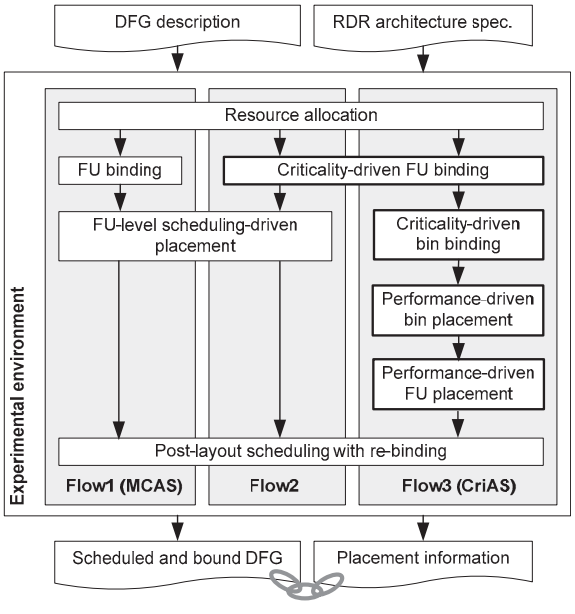

Fig. 13 Three different synthesis flows.

5.2 Experimental Results and Discussions

The experimental results are shown in Table 2. The second to the fourth columns report the required latency after synthesis using Flow1 to Flow3, respectively. The fifth and sixth columns give the latency improvement in percentage for Flow2 over Flow1 and Flow3 over Flow1, respectively. The experimental results show that roughly 9% performance improvement can be achieved on average by the proposed criticality-driven FU binding. The overall synthesis result produced by CriAS is 14.26% better than that by MCAS on average. Besides, 5.36% improvement (14.26–8.90%) is contributed jointly by bin binding and two-level hierarchical placement. In summary, the experimental results clearly demonstrate that CriAS outperforms the existing flow MCAS due to the effectiveness of the proposed criticality-driven FU binding, bin binding, and hierarchical placement techniques.

Table 3 shows the runtime for each test case in all three flows. All the test cases can be finished within 70 seconds. For every test case the runtime of Flow2 is roughly the same as that of Flow1 since the only difference between these two flows is just the weight assignment approach within the FU binding procedure. Meanwhile, though it seems that the

Table 1 Basic information of the input DFGs.

test case	#nodes	#edges	M	N	uti (%)	latency
arf	28	30	2	2	72.73%	8
lee	49	62	2	2	75.00%	9
feed	53	50	2	2	75.23%	7
cos1	66	76	3	3	79.01%	8
mcm	94	128	2	2	75.96%	8
idc	114	164	3	3	77.41%	16
jpeg_f	134	169	3	2	74.56%	13
fft16	414	672	6	4	72.25%	14

Table 2 Synthesis results of the three flows.

test case	Flow1 (latency)	Flow2 (latency)	Flow3 (latency)	2-to-1 (%)	3-to-1 (%)
arf	17	16	14	5.88%	17.65%
lee	15	13	13	13.33%	13.33%
feed	13	11	11	15.38%	15.38%
cos1	19	18	17	5.26%	10.53%
mcm	14	12	12	14.29%	14.29%
idc	34	32	28	5.88%	17.65%
jpeg_f	24	22	20	8.33%	16.67%
fft16	35	34	32	2.86%	8.57%
avg.				8.90%	14.26%

Table 3 Runtime of the three flows.

test case	Flow1 (sec)	Flow2 (sec)	Flow3 (sec)
arf	0.02	0.01	0.02
lee	0.05	0.05	0.06
feed	0.11	0.11	0.12
cos1	0.18	0.18	0.21
mcm	0.89	0.85	0.98
idc	0.60	0.59	0.62
jpeg_f	1.62	1.58	1.84
fft16	63.66	65.5	65.31

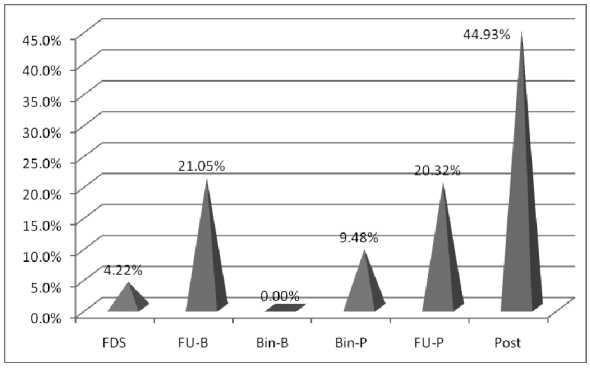

Fig. 14 Runtime share of procedures in CriAS.

proposed CriAS (Flow3) performs more complicated operations than Flow2 does, the consumed runtime does not increase at all. The major reasons are: 1) the bin-level placer deals with a relatively smaller problem size after bin binding; 2) the FU-level placer starts with a very good initial solution, which facilitates a quick convergence.

The average runtime share of each major procedure in CriAS over all the test cases is shown in Fig. 14. FDS represents the resource allocation procedure, where force-directed scheduling is applied; FU-B and Bin-B are for hierarchical binding; Bin-P and FU-P are for the two-level placers; at last, Post indicates the post-processing procedure — post-layout scheduling with re-binding. The most time consuming procedure Post, which is proposed in [10], is invoked in all the three flows. The Post algorithm, which performs placement-driven scheduling with re-binding, is based on an FDLS framework [23]. Its time complexity is roughly estimated as $O(L \cdot N^3)$, where L is the target number of control steps (i.e., latency constraint) and N is the number of operation nodes. The second most time consuming procedure FU-B adopts a linear programming solver, lp_solve [33], for the min-cost flow problem. Both FU-P and Bin-P are SA-based placers and the complexity analysis is essentially similar to that of T-VPlace (the placement part of VPR) [24], [25]. The timing complexity of FDS is $O(L^2 \cdot N^3)$, where L is the target number of control steps and N is the number of operation nodes. More details of the force-directed scheduling (FDS) can be found in [23], [27]. The pseudo code of Bin-B has been shown in Fig. 12 and the time complexity is $O(E \cdot U^2)$, where U is the number of FUs and E is the number of connections between FUs.

6. Conclusion

In this article, we present a hierarchical performance-driven criticality-aware synthesis framework CriAS targeting generic RDR-based multicycle architectures. CriAS features four key techniques: criticality-driven FU binding, criticality-driven bin binding, performance-driven bin-level placement, and performance-driven FU-level placement. During FU/bin binding, the pCGDT instead of the pGDT is concerned. The FU binding problem is modeled as the min-cost flow problem and optimally solved. The bin binding problem is then formulated as a capacity-constrained k-way min-cost partitioning problem and solved accordingly. The hierarchical bin/FU-level placement is employed to keep related critical FUs staying as close as possible. The comprehensive experimental results demonstrate that CriAS achieves an average of 14.26% overall performance improvement without runtime increase as compared to the prior art. The results apparently suggest that CriAS is currently a better synthesis solution for applications targeting the RDR-based architecture supporting multicycle communication design paradigm.

Acknowledgment

This work was supported in part by the National Science Council of Taiwan under Grant NSC 98-2220-E-009-021.

References

[1] International Technology Roadmap for Semiconductors, Semiconductor Industry Association, 2007.
[2] D. Matzke, "Will physical scalability sabotage performance gains?" Computer, vol.20, pp.37–39, 1997.
[3] L.P. Carloni and A.L. Sangiovanni-Vincentelli, "Coping with latency in SOC design," IEEE Micro, vol.22, pp.24–35, 2002.
[4] W.J. Dally, "Interconnect-limited VLSI architecture," IEEE Int'l Conf. Interconnect Technology, 1999.
[5] Y. Mori, V. Moshnyaga, H. Onodera, and K. Tamaru, "A performance-driven macro-block placer for architectural evaluation of ASIC designs," Proc. Annual IEEE Int'l ASIC Conf. and Exhibit, pp.233–236, Sept. 1995.
[6] V. Moshnyaga and K. Tamaru, "A placement driven methodology for high-level synthesis of sub-micron ASIC's," Proc. Int'l Symp. Circuits and Systems, vol.4, pp.572–575, May 1996.
[7] P. Prabhakaran and P. Banerjee, "Parallel algorithms for simultaneous scheduling, binding and floorplanning in high-level synthesis," Proc. Int'l Symp. Circuits and Systems, vol.6, pp.372–376, May 1998.
[8] 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, Nov. 2001.
[9] J. Jeon, D. Kim, D. Shin, and K. Choi, "High-level synthesis under multi-cycle interconnect delay," Proc. Asia and South Pacific Design Automation Conf., pp.662–667, Jan. 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, "CriAS: A performance-driven criticality-aware synthesis flow for on-chip multicycle communication architecture," Proc. Asia and South Pacific Design Automation Conf., pp.67–72, Jan. 2009.
[12] S.-H. Huang, C.-H. Chiang, and C.-H. Cheng, "Three-dimension scheduling under multi-cycle interconnect communications," IEICE Electronics Express, vol.2, no.4, pp.108–114, Feb. 2005.
[13] J. Cong, Y. Fan, and Z. Zhang, "Architecture-level synthesis for automatic interconnect pipelining," Proc. Design Automation Conf., pp.602–607, June 2004.
[14] W.-S. Huang, Y.-R. Hong, J.-D. Huang, and Y.-S. Huang, "A multicycle communication architecture and synthesis flow for global interconnect resource sharing," Proc. Asia and South Pacific Design Automation Conf., pp.16–21, Jan. 2008.
[15] Y.-J. Hong, Y.-S. Huang, and J.-D. Huang, "Simultaneous data transfer routing and scheduling for interconnect minimization in multicycle communication architecture," Proc. Asia and South Pacific Design Automation Conf., pp.19–24, Jan. 2009.
[16] A. Ohchi, N. Togawa, M. Yanagisawa, and T. Ohtsuki, "High-level synthesis algorithms with floorplaning for distributed/shared-register architectures," Proc. Int'l Symp. VLSI Design, Automation and Test, pp.164–167, April 2008.
[17] J. Cong, Y. Fan, and W. Jiang, "Platform-based resource binding using a distributed register-file microarchitecture," Proc. Int'l Conf. Computer Aided Design, pp.709–715, Nov. 2006.
[18] K. Lim, Y. Kim, and T. Kim, "Interconnect and communication synthesis for distributed register-file microarchitecture," Proc. Design Automation Conf., pp.765–770, June 2007.
[19] 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.
[20] A. Terechko, E.L. Thenaff, M. Garg, J. van Eijndhoven, and H. Corporaal, "Inter-cluster communication models for clustered VLIW processors," Proc. Int'l Symp. High Performance Computer Architecture, 2003.
[21] SUIF 2 Compiler System. [Online]. Available: http://suif.stanford.edu/suif/suif2/
[22] M. Smith and G. Holloway, "An introduction to machine SUIF and its portable libraries for analysis and optimization," Division of Engineering and Applied Sciences, Harvard University, 2002.
[23] P. Paulin and J. Knight, "Force-directed scheduling for behavioral synthesis of ASICs," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.8, no.6, pp.661–679, June 1989.
[24] A. Marquardt, V. Bets, and J. Rose, "Timing-driven placement for FPGAs," Proc. Int'l Symp. Field Programmable Gate Arrays, pp.203–213, Feb. 2000.
[25] VPR: Versatile packing, placement and routing for FPGAs. [Online]. Available: http://www.eecg.toronta.edu/~vaughn/vpr/vpr.html
[26] Y. Kwok and I. Ahmad, "Dynamic critical-path schedule: An effective technique for allocating task graphs to multiprocessors," IEEE Trans. Parallel Distrib. Syst., vol.7, no.5, pp.506–521, May 1996.
[27] G.D. Micheli, Synthesis and Optimization of Digital Circuits, McGraw-Hill, New York, 1994.
[28] R. Ahuja, T. Magnanti, and J. Orlin, Network flows: Theory, algorithms, and applications, Prentice Hall, 1993.
[29] C. Tseng and D.P. Seiwiorek, "Automated synthesis of data paths in digital systems," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.5, no.3, pp.379–395, July 1986.
[30] MCAS: multicycle architectural synthesis system. [Online]. Available: http://cadlab.cs.ucla.edu/software_release/mcas/
[31] ExPRESS group. [Online]. Available: http://express.ece.ucsb.edu/
[32] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, Introduction to algorithms, 2nd ed., the MIT press, 2001.
[33] lp_solve: An open source mixed integer linear programming (MILP) solver. [Online]. Available: http://sourceforge.net/projects/lpsolve/

Chia-I Chen received the B.S. degree in Electronics Engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2005, where she is currently working toward the Ph.D. degree in the Institute of Electronics. Her current research interests include high-level synthesis and computer architecture.

Juinn-Dar Huang received the B.S. and Ph.D. degrees in Electronics Engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1992 and 1998, respectively. He is currently an Associate Professor in the Department of Electronics Engineering and the Institute of Electronics, National Chiao Tung University. His current research interests include high-level synthesis, design verification, 3D IC architecture/CAD, and microprocessor design. He has served in the Organizing Committees of IEEE/ACM ASP-DAC 2010 and SASIMI 2010. He has been the Secretary General of Taiwan IC Design Society (TICD) from 2004 to 2008, the Technical Program Committee Vice-Chair of VLSI Design/CAD Symposium 2008, the Technical Program Committee member of IEEE/ACM DATE 2008/2010, and the Organizing Committee member of IEEE International Conference on Field-Programmable Technology (ICFPT) 2008. He is a member of the IEEE, ACM, and Phi Tau Phi.