Model-Driven Design and Generation of New Multi-Facet Arbiters: From the Design Model to the Hardware Synthesis

Jer Min Jou, Yun-Lung Lee, and Sih-Sian Wu

***Abstract*—Designing of arbiters has become increasingly important due to their wide use in the areas such as multiprocessor systems-on-a-chip and on-chip or off-chip high-speed cross-bar switches and networks. In this paper, we proposed a new systematic model-driven flow for designing the new scalable multi-facet arbiters through a 3-phase process combined with the template-based modular design approach that includes the design model derivation phase, the architecture model (or template) design phase as well as the arbiter hardware implementation and generation phase. First, we described the phase 1 of the design flow of how to induce an arbiter design model in detail by careful analysis of arbiter design issues and systematic design space exploring the construction of the model. Then, we continue to discuss the phase 2 of how to derive an architecture model or template using the reusability, modularity and expansibility techniques. With both the design and the architecture models, designers can easily design or at least understand and thus choose many kinds of different but better arbiters efficiently. Finally, we have developed a scalable decentralized parallel tree structure and corresponding modular algorithms for efficient arbiter hardware implementation, which also is the final phase of our proposed 3-phase model-driven design flow. Moreover, based on this modular and reusable hardware implementation structure as well as the algorithms, we have designed a parametric arbiter generator that automatically generates various multi-facet arbiters. Using this hardware implementation design and the generator, not only a fast and small round-robin arbiter but also other type arbiters were designed and generated on the fly. The hardware implementation algorithms, the generator, and the experiment results all were given to verify their performances. To our knowledge, this is the first time that such a systematic model-driven design approach is proposed in the practical hardware design and such a multi-facet arbiter is designed.**

***Index Terms*—Architecture model, architecture template, decentralized parallel tree, design model, design space, generator, model-driven design flow, multi-facet arbiter.**

I. Introduction

WITH THE RAPID evolution of technology, resource arbitration has become one of the most critical problems in the design and implementation of embedded multi-processor systems-on-a-chip (MPSoCs) with thousands of cores [1], [2], where many intellectual properties (IPs) such as processor cores, memories, on-chip switches/networks, and peripheral units are integrated into a single die and extensively compete shared communication and/or computation resources each other. An often used shared communication resource in a MPSoC is on-chip switches. Fig. 1 shows such an on-chip switch connected to some IPs, which are treated as its input or output masters by the I/O ports. The request signals are sent by the input masters to request the usage authority of the on-chip switches. Then, these request signals are arbitrated by the arbiter and one of the masters is granted to monopolize the shared bus. As the number of the same or different types of masters increases in a single chip [2], the various resource contentions will be increased quickly. Therefore, the multi-facet, high performance, and low-cost design of the arbiters dealing with the serious different resource contentions need more attentions.

Many arbiter designs have been proposed [3]–[5]; however, the arbiter design spaces, the design models, and the architecture templates of them have not been analyzed and explored thoroughly. There is also a lack of a systematic design methodology similar to the platform-based design approach in [6] for designing new multi-facet arbiters with a high performance and/or a low cost.

Here, we have proposed the following new systematic 3-phase model-driven flow to efficient design and implement multi-facet arbiters: 1) the design model derivation phase; 2) the architecture model (or template) design phase; and 3) the arbiter hardware design and generation phase. To our knowledge, this is the first time that such a systematic 3-phase model-driven design methodology is proposed for the arbiter design.

The most significant problem in the arbiter design resides in its efficient and lower cost implementation. This difficulty arises in finding feasible implementation algorithms that could meet the following six objectives: 1) develop those algorithms with the scalable features based on an architecture model using a systematic design methodology; 2) choose a parallel implementation able to arbitrate the requests quickly; 3) minimize the critical path delay; 4) maintain their area cost at a minimum value; 5) allow decentralized input processing: that is each input request can be processed in an individual input processing unit independently; and 6) ensure simplicity

Manuscript received August 25, 2010; revised October 19, 2010 and February 12, 2011; accepted March 28, 2011. Date of current version July 20, 2011. This work was supported in part by the National Science Council of Taiwan, under Contract NSC-99-2221-E-006-224. This paper was recommended by Associate Editor D. Atienza.

The authors are with the Department of Electrical Engineering, National Cheng Kung University, Tainan 701, Taiwan (e-mail: jou@j92a21.ee.ncku.edu.tw; jackyli2k@gmail.com; sihsianwu@j92a21.ee.ncku.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2011.2139211

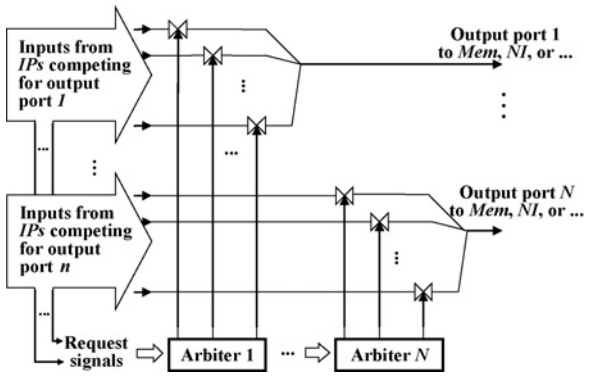

Fig. 1. N arbiters in a N×N crossbar switch.

in very large scale integration (VLSI) design by efficient and fast automatic generations.

To meet the growing demand for designing the low cost and/or high-speed arbiters, many arbiter implementations [3]–[5], [7]–[11] have been proposed. For on-chip or off-chip communication, the most popular arbiter scheme is the round-robin arbiter, since it is fair and starvation-free. The iSLIP [11] authors have implemented an $N \times N$ switch arbiters in hardware which they call a programmable priority encoder. Chao *et al.* [10] described the design of a round-robin arbiter for a packet switch. They referred to their hardware design as a ping pong arbiter. Using the same idea of the ping pong, another arbiter design, called switch arbiter (SA), was proposed in [3]. An SA is constructed by a tree structure composed of 4×4 *SA* nodes. An SA node consists of a D flip-flop, four priority encoders, a 4-bit ring counter, five 4-input OR gates, and four 2-input AND gates. The SA is the fastest among known arbiters, but it has a more complex structure. Zhengy and Yang [4] proposed two arbiters, PRRA and IPRRA, which are based on a binary search algorithm and have the least area but are slower.

In contrast to round-robin arbiters which divide the bandwidth (BW) of a shared resource into equal parts for every requester, BW-constraint arbiters are used in certain systems in which the bandwidth requirements of requesters significantly differ. In this circumstance, bandwidth requirements of requesters are unpredictable. Thus, BW-constraint arbiters arbitrate requesters according to their allocated bandwidth weights (BW weights), which indicate their relative quantity of bandwidth requirement. A larger allocated BW weight corresponds to a more required bandwidth. LOTTERYBUS [9] is a well-known BW-constraint arbiter that is often used in SoCs to arbitrate on-chip buses. LOTTERYBUS decides the granted request using a probability approach. Input requests with larger BW weights are more likely to be granted, and vice versa. Therefore, it avoids the starvation. If the arbiter can allocate a resource to requesters in proportion to their BW weights, the resource utilization will be maximized. Instead of enforcing an upper bound on a requested service, the credit-control static-priority arbiter (CCSP) proposed by Akesson *et al.* [5] enforced an upper bound on a provided service. It can provide appropriate bandwidth allocation when the bandwidth requirement of requesters is unknown. Each input of the CCSP accumulates the quantity of resources during each time unit according to the magnitude of the average BW request ratio r, where $0 \leq r \leq 1$, and consumes it when the request arrivals.

None of the above arbiters meets all of the six objectives. From the SoC design point of view the architectures of the arbiters have not been rightly structured to date. The problem resides in lack of the proper arbiter design model, architecture model, and modular hardware design and generation, which will be well addressed here by using 3-phase model-driven flow.

Each phase is organized as follows. First, the design model derivation phase (i.e., the phase 1 of the design flow) is described among Sections II–V. Then, the architecture model or template (i.e., the phase 2 of the design flow) is described in Section VI. Finally, the arbiter hardware implementation and generation phase (i.e., the phase 3 of the design flow) is described among Sections VII and VIII.

And the rest sections of this paper are organized as follows. Section II introduces the background for multi-facet arbiters. Section III describes some issues of the arbiter design, with the corresponding solutions provided in Section IV. Section V describes the proposed arbiter design model. The proposed arbiter architecture model and template are derived in Section VI. In Section VII, we derive and propose an efficient hardware implementation of the arbiter architecture model to speed up much the arbitration performance. The automatic generator is introduced in Section VIII before studying the design results of multi-facet arbiters with various arbitrating properties in Section IX. Finally, conclusions and future work are presented in Section X.

II. Background for Multi-Facet Arbiters

When a monopolized-only resource is shared by N masters, a unit of usage time of the resource (referred to as a time unit hereafter) can be allocated to only one master by an arbiter. Therefore, if many masters request the resource at the same time, an arbiter is needed to fairly and efficiently determine which master can use a time unit. In the following, we assume that the cycle time of an arbiter is the same as a time unit for simplicity of discussions, although, they can be different. The time interval at which the arbiter has serviced every master with a request is defined as an arbitration round, which is determined by the number of masters who request the resource simultaneously, but it does not exceed N time units, where N is the number of masters.

An arbiter considers the request signals provided by each master and some other information, including the following: 1) bandwidth (BW) request ratio of the masters: for example, if the allocated BW request ratio of three masters is 3:2:1, the first master is allocated three time units of the resource usage, the second master is allocated two time units of resource usage, and so on; 2) waiting latency (WL) of each master: this time is defined as the number of time units that the master waits for the arbiter's grant; 3) resource granularity: it indicates the maximum number of data units that a granted master could obtain (transfer) from (to) the shared resource per time unit. The data unit is defined as the basic unit of the quantum or bandwidth that the shared resource could

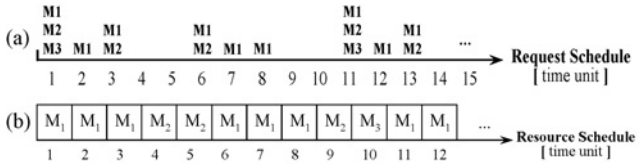

Fig. 2. (a) Request schedule for masters M1, M2, and M3. (b) Corresponding resource schedule.

provide; 4) arbitrating latency (AL) This indicates the total number of time units required by the arbiter to arbitrate a master request. In other words, the AL of a master could be defined as the total number of time units between the time the master's request is first granted to the time the request is fulfilled and finished; 5) priority of each master: this is used to determine the precedence of resource usage among masters. Finally, there are two arbitration interrupting behavior choices; 6) preemption; or 7) non-preemption.

III. Design Issues of Arbiters

In this section, four important design issues of arbiters are discussed in detail.

A. BW Request Quantity and Waiting Latency Tradeoff

In a conventional arbiter, the masters are usually arbitrated based on the values of their allocated BW request quantities, and the master with the largest value is granted first. The waiting latency, WL, of each master, therefore, depends on the corresponding BW request quantity. A master with a small allocated BW request quantity always has a large WL, and thus even starvation may occur among masters. Fig. 2(a) shows the request schedule for three masters, M_1, M_2, and M_3, with allocated BW request quantities 6, 3, and 1, respectively. M_i on the time axis indicates that the request of M_i is issued at that time. The corresponding resource schedule, shown in Fig. 2(b), is arbitrated according to the allocated BW request quantities of the masters. These three masters issue their requests simultaneously at time 1 and 11, and M_1 gets granted earlier than both M_2 and M_3 due to its larger BW request quantity. However, if M_2 or M_3 requires a lower WL for some reasons, the arbitration scheme that depends only on the BW request quantities of the masters no longer works, although this kind of arbitration is often reasonable and good.

B. Resource Granularity and Arbitrating Latency Tradeoff

Both the arbitrating latencies (ALs) of masters and resource utilization are influenced by the resource granularity. Conventionally, arbitration fairness of most arbiters is achieved by servicing each master in a round-robin manner for each arbitration round. In other words, each master gains at most one access to the shared resource in an arbitration round. The round-robin arbitration scheme is adopted by many arbiters because it is simple and starvation-free. However, there is a tightly-coupled relation between resource granularity and AL for the round-robin arbiters. The ALs are smaller for masters that use coarse-grain resource granularity, but this scheme may result in a severe resource over-allocation problem. For

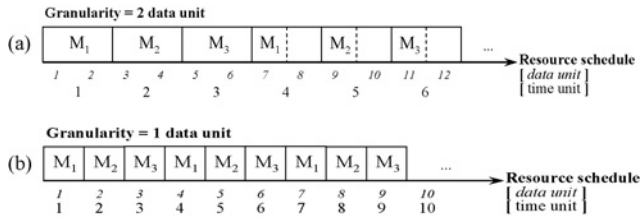

Fig. 3. (a) RSR with resource granularity = 2. (b) RSR with resource granularity = 1.

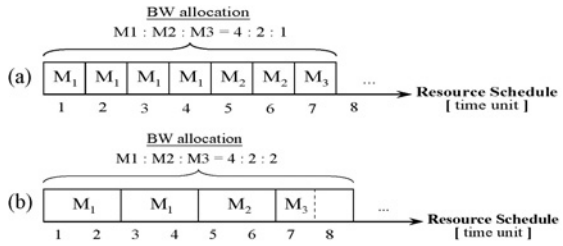

Fig. 4. (a) RSR with resource granularity = 1. (b) RSR with resource granularity = 2.

example, masters M_1, M_2, and M_3 request three data units of shared resource simultaneously, the resource schedule results (RSRs) based on 2 and 1 data units of resource granularity are shown in Fig. 3(a) and (b), respectively. The AL of M_1 (four time units) in Fig. 3(a) is smaller than that (seven time units) in Fig. 3(b). The ALs can hence be efficiently reduced for arbiters with coarse-grain resource granularity. However, it creates a resource over-allocation problem, e.g., each of the three masters in Fig. 3(a) wastes half a time unit at time 4, 5, and 6. Therefore, the tradeoff between AL and resource utilization is an important issue.

C. Resource Granularity and Fairness Tradeoff

Arbitration fairness can be influenced by the resource granularity. An arbiter is fair if: 1) it is work-conserving, i.e., it never leaves any resource idle if there are requesting masters, and 2) it arbitrates resource time units to masters in exact proportion to their BW request ratios. Using finer resource granularity can provide more precise bandwidth allocation and makes arbitration fairer. For three masters M_1, M_2, and M_3 with BW request quantities 4, 2, and 1, respectively, the RSRs corresponding to resource granularities of 1 and 2 are shown in Fig. 4(a) and (b), respectively. In Fig. 4(b), the arbiter wastes one time unit at time 8 since the resource granularity is larger than its request data unit.

D. Integrity of Arbiter Design

Conventional arbiters are usually designed for a single arbitration object, so the multiple arbitration requirements cannot be satisfied at the same time. For instance, a round-robin arbiter cannot continuously provide a small WL to the impatient masters; a linear-priority arbiter does not provide non-preemptive arbitration behaviors; a WL-aware arbiter gives a higher priority to masters with a strict WL requirement but a lower priority to others. These arbiters cannot be designed in an integrated, modular, and multi-facet manner due to the

lack of a comprehensive arbiter design, architecture models, and the relative systematic design flow.

IV. Solutions to the Design Issues of Arbiters

To solve the problems encountered in arbiter design, the following solutions are proposed.

A. Solution to the Issue of BW Request Quantity and Waiting Latency Tradeoff

It is not appropriate to arbitrate the masters, which have different WL requirements, based only on their BW request quantities. An additional arbitration input, called the priority, is added for each master; it is used to arbitrate the masters under the BW request ratio constraint. Masters with small WL requirements are assigned high priority. The arbiter grants the masters according to their priorities and BW request ratio. This priority-based BW request ratio arbitration can be classified further into non-equal-priority arbitration and equal-priority arbitration, which then should be implemented as a BW request ratio-based round-robin arbiter. However, how to combine the BW request ratio constraint and the priority? Should this be priority-based BW request ratio arbitration or BW request ratio-based priority arbitration? This is addressed later during the development of the proposed new arbiter design model.

B. Solution to the Issue of Resource Granularity and Arbitrating Latency Tradeoff

To deal with the coupling or tradeoff between resource granularity and AL, preemptive or non-preemptive arbitrations can be used to decouple resource granularity and AL. Preemptive and non-preemptive arbitrations belong to non-equal-priority-based arbitration. Preemptive arbitration allows the higher-priority masters to interrupt and preempt the resource usage of a lower-priority master, and thus, higher-priority masters will have smaller WLs than those of lower-priority masters.

Compared with preemption, non-preemptive arbitration can decrease the ALs of the lower-priority masters, but increase the WLs of higher-priority masters, which may introduce the deadlock or starvation. A tradeoff between WL and AL must thus be considered. To avoid the deadlocks or starvation caused by non-preemptive arbitration, a bandwidth-weight tuning scheme is proposed to regulate and limit bandwidth gained by the masters. The scheme is presented in Section VI. Again, should this be the priority-based preemptive arbitration or the preemption-based priority arbitration? This is solved in the construction of the arbiter design model following.

C. Solution to the Issue of Resource Granularity and Fairness Tradeoff

The tight relation between resource granularity and AL can be decoupled using a non-preemption arbitration design. Therefore, the finest resource granularity is adopted to allocate the shared resource to increase resource utilization while keeping fairness. To achieve fairness, a set of masters' BW request ratios is used, and a counting-based master isolation scheme is applied to protect the bandwidth of each master from being taken by other masters. Each master is given some time units, called BW quota. Every time the shared resource is granted to and used by a master, that master's BW quota is decreased. When a master uses up its BW quota, its succeeding requests are ignored by the arbiter to prevent the bandwidth of other masters from being taken.

D. Solution to the Issue of Integrity

Parametric and composite design concepts are applied to design an arbiter which implements versatile arbitrating schemes proposed above. The composite parametric arbiter template to be developed should be reconfigured modularly as a linear-priority arbiter with the preemption or non-preemption scheme to favor higher-priority masters and lower-priority masters with lower WLs and lower ALs, respectively. On the other side, the modular arbiter also could be reconfigured as an equal-priority arbiter like a round-robin arbiter to prevent masters from deadlocks or starvations. Moreover, the master isolation idea is implemented to maintain fairness between masters. That is we want to develop a new multi-facet arbiter design model and its architecture template (or model) in the following sections to integrate the above-mentioned solutions.

V. Development of the New Multi-Facet Arbiter Design Model

After the analyses and studies of arbiter issues and their solutions, the design and development of a new arbiter design model with multi-facet using the design space exploration technique is described in detail below.

A. Design Space Exploration of the Arbiter Design Model

The design of the integrated arbiter design model with the design space exploration method involves exploring alternate arbiter configurations to generate an efficient design model. Based on the arbiter design solutions above, the three essential factors of arbitration are priority, preemption, and bandwidth. These factors are explored to derive and build the proposed multi-facet arbiter design model. How many essential factor configurations (and their inverses) does the design space have? How many valid arbiter designs are there? What is the processing order of the essential factors in the multi-facet design model for the design of a valid arbiter? In the design space, every essential factor affects the arbitration. The design space of a multi-facet arbiter with some or all of those factors or their inverses may be wide, so an exact design model has to be explored and designed for efficient arbiter design.

Assume that the three essential factors priority, preemption, and bandwidth, are labeled as I, E, and W, respectively, and that they indicate that the arbiter will be realized with the equal-priority, the preemptive and the bandwidth-constraint schemes, respectively. I', E', and W' indicate that the arbiters will be realized with the non-equal-priority, the non-preemptive, and the non-bandwidth-constraint schemes, respectively. For example, an arbiter with I, E', and W' has equal-priority, (then) non-preemption, and (finally) non-bandwidth-constraint properties (circuit modules). Note that

the order of those factors is important; the hardware structure of the $IE'W'$ arbiter in the design model (and space) has a priority control module followed by a non-preemption module and then by a bandwidth constraint control module, i.e., it is a composite design. The design space of an arbiter design model with/without the three essential factors and their inverses has at most 78 different configurations (combinations); each configuration may or may not form a feasible or valid multi-facet arbiter. Arbiters designed with one, two, and three essential factors form three different sets of valid or invalid configurations of the multi-facet arbiters: S_1, S_2, and S_3 respectively, as follows. The whole design space $\boldsymbol{S}$ of the arbiters is the union of sets S_1, S_2, and S_3, and then 78 configurations will correspond to *possibly* 78 different arbiters

$S_1 = \{ I, E, W, I', E', W'\}$

$S_2 = \{ IE, EI, I'E, EI', IE', E'I, I'E', E'I', IW, WI, I'W, WI', WI', W'I, I'W', W'I', EW, WE, E'W, WE', EW', W'E, E'W', W'E'\}$

$S_3 = \{ IEW, IWE, EIW, EWI, WIE, WEI, I'EW, I'WE, EI'W, EWI', WI'E, WEI', IE'W, IWE', E'IW, E'WI, WIE', WE'I, IEW', IW'E, EIW', EW'I, W'IE, W'EI, I'E'W, I'WE', E'I'W, E'WI', WI'E', WE'I', I'EW', I'W'E, EI'W', EW'I', W'I'E, W'EI', IE'W', IW'E', E'IW', E'W'I, W'IE', W'E'I, I'E'W', I'W'E', E'I'W', E'W'I', W'I'E', W'E'I'\}$

$S = S_1 \cup S_2 \cup S_3$.

Some of the 78 configurations are invalid (not a multi-facet arbiter) and should be removed to prune the design space. The final pruned design space is then used to derive the proposed multi-facet arbiter design model. The non-trivial process of pruning the design space is omitted due to space limitations. The final number of the valid configurations in set $\boldsymbol{S}$ for the multi-facet arbiter design space is reduced to 6; that is WI, $W'I$, $WI'E$, $WI'E'$, $W'I'E$, and $W'I'E'$.

B. Development of the Multi-Facet Arbiter Design Model

Based on the pruned design space explored above, the proposed arbiter design model is derived below. The processing order of the three essential factors in the space is as follows: bandwidth W, then priority *I*, and finally preemption *E*, which is the order in the design model. Each factor is an option considered from the right (top) side to the left (bottom) side in the arbiter design model of Table I based on the concept of the VLSI top-down design methodology. The arbiter design is classified into six types from the outer side to the inner side in the arbiter design model (i.e., from left to right in Table I): non-BW-constraint and equal-priority arbitration, non-BW-constraint and non-equal-priority arbitration with/without preemption, BW-constraint and equal-priority arbitration, and BW-constraint and non-equal-priority arbitration with/without preemption, which correspond to the six valid configurations (or arbiters) above.

The model in Table I first divides the arbiter design from left to right into two main parts, one with the BW constraint and one without the BW constraint, depending on the factor, BW request quantity, that represents whether masters have essential allocated bandwidths. Based on the issues discussed above, there is a problem for arbitration without BW constraint, "Problem I: Unfair," which can be solved by using the BW constraint as shown in the bottom-left part of Table I ("Problem I is solved"), however, it generates another problem, "Problem II: WL is coupled with the allocated BW request quantity." Based on the model, Problem II can be solved by using the factor, priority, which is in the second column at the bottom of Table I ("Problem II is solved"). Moreover, there is a problem in the non-equal-priority arbitration, "Problem III: WL is not directly controlled by the priority." In other words, can the WLs of the masters be controlled just by regulating their priorities? This problem can be solved by "Problem III is solved" in the third column of Table I. However, preemption leads to a high AL; this problem can be alleviated by using the non-preemption arbitration ("Problem IV is solved").

TABLE I
NEW MULTI-FACET ARBITER DESIGN MODEL

<table>
<tr><td rowspan="2">No BW constraint: Problem I: Unfair.</td><td colspan="2">Equal priority:
Problem IV: Large AL for most masters.</td><td>W′I
arbiter</td></tr>
<tr><td>Non-equal priority:
Problem III: WL is not directly controlled by priority.</td><td>Non-preemptive:
Problem IV is solved:
Reduce AL for lower-priority masters.</td><td>W′I′E′
arbiter</td></tr>
<tr><td rowspan="3">BW constraint: Problem I is solved. Problem II: WL coupling with BW ratio problem.</td><td rowspan="2"></td><td>Preemptive:
Problem III is solved:
WL is inversely proportional to priority.
Problem IV: Large AL for most masters.</td><td>W′I′E
arbiter</td></tr>
<tr><td colspan="2">Equal priority:
Problem II is solved: WL is decoupled with BW ratio; and WLs of all masters are equal
Problem IV: Large AL for most masters.</td><td>WI
arbiter</td></tr>
<tr><td>Non-equal priority:
Problem II is solved:
WL is decoupled with BW ratio.
Problem III: WL is not directly controlled by priority.</td><td>Non-preemptive:
Problem IV is solved:
Reduce AL for lower-priority masters.</td><td>WI′E′
arbiter</td></tr>
<tr><td></td><td>Preemptive:
Problem III is solved:
WL is inversely proportional to priority.
Problem IV: Large AL for most masters.</td><td>WI′E
arbiter</td></tr>
</table>

WL: waiting latency, *AL*: arbitrating latency.

With this model (Table I), the appropriate path starting from the left-most block and ending at the right-most block in Table I can be selected to obtain the required properties, factors, and key points during arbiter design. More importantly, this model can be used to systematically derive an efficient architecture model (i.e., a template) for plainly designing a composite multi-facet arbiter.

VI. DERIVATION OF THE ARCHITECTURE MODEL

The arbiter design model developed above only provides the key arbitrating properties and factors that need to be considered during the design of an arbiter. It does not provide any information about how to design an efficient and/or flexible architecture template and hardware of the multi-facet arbiter. In this section, a reusable, expansible and modular architecture model is derived for the multi-facet arbiter design model in Table I.

Algorithm 1 PREM($ri_0 \sim ri_{N-1}$, $g_0 \sim g_{N-1}$)

Input: $ri_0 \sim ri_{N-1}$(N requests (masked), see Fig. 5)
Output: $g_0 \sim g_{N-1}$(N output grants)
begin
  for $i = 0$ **to** $N-1$ **do** $g_i \leftarrow 0$;
  for $i = 0$ **to** $N-1$ **do begin**
    if $ri_i == 1$ **then** $g_i \leftarrow 1$;
    break;
  end-for
end

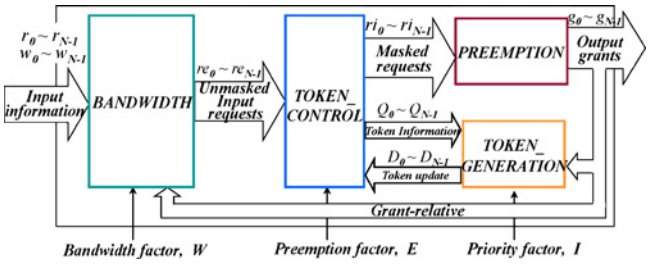

Fig. 5. New multi-facet arbiter architecture model, MAM, for arbiters in Table I.

To derive the multi-facet architecture model, called MAM, from the arbiter design model of Table I is just like to implement the multi-facet arbiter's architecture template from the Table. In contrast to the top-down or left-to-right design methodology of SoCs used in the design of the arbiter design model in Table I, the bottom-up or right-to-left SoC implementation methodology is applied for the arbiter architecture and template implementation from Table I. Thus, with the bottom-up order of the essential factors preemption, priority and bandwidth from the right to the left in the arbiter design model of Table I, we first derive and implement the arbiter architecture starting from the preemption property (the rightmost of the arbiter design model) using the *PREEMPTION* module, which has the arbiter request signals as the inputs and the grant signals as the outputs to realize the preemption arbitration. It may preempt the lower-priority requesters and gives the grant to the requester with the highest property when all those requests come at the same time. The algorithm of the *PREEMPTION* module, denoted as PREM, is designed as Algorithm 1, where $ri_0 \sim ri_{N-1}$ are the signals masked by the *TOKEN_CONTROL* module, and $g_0 \sim g_{N-1}$ are the output grants.

Next, the non-preemptive and the priority properties, top and right, respectively, from the preemption property in the design model of Table I are considered. To realize the non-preemptive and/or the equal-priority arbitrating properties based on the *PREEMPTION* module, a masking operation that implements the non-preemptive and the equal-priority properties on the input (left) side of the *PREEMPTION* module by masking undesired input requests of the *PREEMPTION* module is adopted. To do this, it is required that: 1) know which input requests are desired under a certain arbitrating property, and 2) mask those undesired input requests. To achieve 1, a TOKEN_GENERATION module, which generates the tokens (or masking information) used at the current cycle by referencing the grant-relative information from the *PREEMPTION* module at the previous time cycle, is applied. To achieve 2, a *TOKEN_CONTROL* module, which masks and filters out the undesired input requests according to the tokens generated by the *TOKEN_ GENERATION* module, is used.

Algorithm 2 $TG(I, ri_0 \sim ri_{N-1}, g_0 \sim g_{N-1}, Q_0 \sim Q_{N-1}, D_0 \sim D_{N-1})$

Input: I, $ri_0 \sim ri_{N-1}$, $g_0 \sim g_{N-1}$(*N grants, see Fig. 5*)
Input: $Q_0 \sim Q_{N-1}$(*N internal states*)
Output : $D_0 \sim D_{N-1}$(*N tokens*)
begin
  $anyRI_0 \leftarrow 0$; // N internal wires
  $anyRI_1 \leftarrow ri_0$; // ri_0: I/P from *PREM module*
  if $I == 1$ // for equal-priority arbitration
  then begin
    for $i = 0$ **to** *N-1* **do begin**
      if $i \geq 2$ **then** $anyRI_i \leftarrow ri_{i-1}$ **or** $anyRI_{i-1}$;
      $D_i \leftarrow Q_i$ **and** $anyRI_i$;
    end
  end
  else begin // for non-equal-priority arbitration (I')
      for $i = 0$ **to** *N-1* **do** $D_i \leftarrow g_i$;
      . . .
  end
  if ... **begin** ... **end** ***//expansible*** for any other arbitration
  output all D_i
end

In the*TOKEN_GENERATION* module, the tokens can be generated differently according to the equal-priority, non-equal-priority, or other arbitrating properties by setting the parameters I or other parameters from outside. For example, the tokens in non-preemptive arbitration are used to tell the arbiter which higher-priority input requests should be blocked such that the lower-priority input request can be granted continuously; the tokens in the equal-priority arbitration are used to inform the arbiter which input requests will be granted during the present arbitration round. Thus, the function of the *TOKEN_GENERATION* module is *expandable*. The algorithm of the *TOKEN_GENERATION* module, denoted as TG, is designed as Algorithm 2, where $Q_0 \sim Q_{N-1}$ are the internal states of the *TOKEN_ CONTROL* module, and $D_0 \sim D_{N-1}$ are the token information. The reconfiguration parameter, I, is used to configure the priority arbitrating property.

The algorithm of the *TOKEN_CONTROL* module denoted as TC, which masks the undesired input requests according to the tokens generated by TG, is designed as Algorithm 3, where $re_0 \sim re_{N-1}$ are signals which have not yet been masked by the *TOKEN_CONTROL* module, and the reconfiguration parameter, E, is used to configure the preemption arbitrating property. The main function of the TC module is used to filter out the input requests according to D's masking information, and thus realize the equal-priority or the non-preemptive arbitrating property.

Based on the algorithm, the tokens generated from the *TOKEN_GENERATION* module can be used to mask some input requests to avoid undesired ones from entering the

Algorithm 3 *TC($re_0 \sim re_{N-1}$, $D_0 \sim D_{N-1}$, E, $ri_0 \sim ri_{N-1}$, $Q_0 \sim Q_{N-1}$)*

Input: E, $re_0 \sim re_{N-1}$ (N unmasked requests), $D_0 \sim D_{N-1}$(TG I/P)
Output: $ri_0 \sim ri_{N-1}$, (N masked requests), $Q_0 \sim Q_{N-1}$ (to TG)
begin
  for *i = 0* **to** *N-1* **do** $Q_i \leftarrow 1$;
  for *each arbitration time unit* **do begin**
    for *i = 0* **to** *N-1* **do** $Q_i \leftarrow D_i$;
    if *E == 1 // for preemptive arbitration under I'*
    then for *i = 0* **to** *N-1* **do** $ri_i \leftarrow re_i$;
    else begin *// non-preemptive arbitr. under I' or others*
      for *i = 0* **to** *N-1* **do** $ri_i \leftarrow re_i$ **and** Q_i;
    end
  end-for
end

Algorithm 4 *BW($r_0 \sim r_{N-1}$, $w_0 \sim w_{N-1}$, $g_0 \sim g_{N-1}$, W, $re_0 \sim re_{N-1}$)*

Input: W, $r_0 \sim r_{N-1}$ (N input requests), $g_0 \sim g_{N-1}$(N grants)
Input: $w_0 \sim w_{N-1}$ (N bandwidth requests set by requesters)
Output: $re_0 \sim re_{N-1}$ (N unmasked requests, see Fig. 5)
begin
  load $\leftarrow$ 0; // control var. for bandwidth quota loading
  for *i* = 0 **to** *N*-1 **do** $QB_i \leftarrow w_i$; // initial bandwidth quota
  for *each arbitration time unit* **do**
    if W == 1 // with bandwidth constraints
    then begin
      if *load*== 1 **then for** *i* = 0 **to** *N*-1 **do**
      $QB_i \leftarrow QB_i + w_i$;
      else begin
        for *i* = 0 **to** *N*-1 **do**
        if QB_i== 0 **then** $re_i \leftarrow 0$;
        else begin
          $re_i \leftarrow r_i$; **if** g_i== 1 **then** $QB_i \leftarrow QB_i - 1$
        end
      end–for
    end
  end
  else // without bandwidth constraints
    for *i* = 0 **to** *N*-1 **do** $re_i \leftarrow r_i$;
  if *all re_i are* 0 **then** *load* $\leftarrow$ 1; // new bandwidth quota
end–for
end

PREEMPTION module to violate the required arbitrating property. In addition, if we want to realize any special arbitrating properties not listed in the design model, can the arbiter architecture model provide? The answer is yes, i.e., the *TOKEN_GENERATION* module is expandable. To realize a special arbitrating property, we just only need to design a new token generation scheme, and then add it into the TG algorithm as shown. Thus, a new desired arbiter is formed.

Finally, we include the bandwidth property in our MAM to fully implement all functions in the design model of Table I. Since the bandwidth-constraint arbitration only determines the BW allocation ratio of requesters, it does not affect the order of granting. Thus, the *BANDWIDTH* module can be simply designed and added as the inputs of the *TOKEN_CONTROL* module to perform bandwidth-constraint arbitration. The main function of the *BANDWIDTH* module is like a bandwidth limiter, which ensures that the required bandwidth allocation for each requester is carried out. Similar to the masking actions in the *TOKEN_CONTROL* module, the *BANDWIDTH* module masks the input requests which have exhausted their available bandwidths to avoid violating the required bandwidth allocation. Thus, the algorithm of the *BANDWIDTH* module, denoted as BW, is designed as Algorithm 4, where $r_0 \sim r_{N-1}$ are the input requests, and $w_0 \sim w_{N-1}$ are the bandwidth weight values. The reconfiguration parameter, *W*, is used to configure the bandwidth arbitrating property.

For requesters with bandwidth requirements, the *BANDWIDTH* module must realize bandwidth–constraint arbitration. To do this, the *BANDWIDTH* module uses the concept of requester isolation, which assigns each requester with an integer value called, a BW quota (QB). Every time a shared resource is used by a requester, its BW quota is decreased by 1 unit as time increases by one time unit. When a requester uses up its BW quota, its succeeding requests are filtered out by the *BANDWIDTH* module to prevent the bandwidth of other requesters from being taken.

With these four modules described above, the complete arbiter architecture model (and template) called the multi-facet arbiter architecture model, MAM, for the multi-facet design model in Table I is derived as shown in Fig. 5. In it, only the *PREEMPTION* module is essential, and designers can remove other modules and/or add new modules on the model (template), i.e., it has reusability, composability, modularity and expandability. An example is given below.

In Fig. 5, three reconfiguration parameters, *W*, *I*, and *E*, corresponding to the three essential factors bandwidth, priority, and preemption in the arbiter design model, respectively, are added to the arbiter architecture model. They can be used to reconfigure the architecture with various arbitrating properties. The composite and modular design and these reconfiguration parameters, the proposed MAM could implement six various arbiters outlined in the arbiter design model by re-arranging and connecting the four modules.

With the proposed multi-facet arbiter architecture model in Fig. 5, we can realize one or more arbitrating properties simultaneously by setting appropriate reconfiguration parameters in the arbiter architecture model, i.e., the template is multi-facet and reusable. Seven multi-facet arbiters with corresponding specific arbitrating properties were designed and named using the three parameters in the arbiter design model of Table I. Parameter *W* indicates that the arbiter has the ability to control bandwidth allocation among requesters. Parameter *I* indicates that the arbiter realizes equal-priority arbitration. Parameter *E* denotes the arbiter is preemptive while realizing non-equal priority arbitration. Parameter *E* does not exist in the equal-priority arbiter since the issue of preemption is not considered. The relation between reconfiguration parameters and six of the seven arbiters is shown in the right-most column of Table I. Each of the first six arbiters has one corresponding arbitrating property in the arbiter design model in Table I. The seventh

arbiter is a whole multi-facet arbiter, as shown in Fig. 5; it can be reconfigured to behave any arbitrating property by setting the appropriate reconfiguration parameters W, I and E as shown in Fig. 5 and Table I.

Note that the template in Fig. 5 is reusable, composite, modular and expandable. Take the WI' E arbiter design as an example. Using the MAM, it can be directly obtained by setting the parameters W, I, and E to 101. Or, we can remove the modules *TOKEN_CONTROL* and *TOKEN_GENERATION* from the figure of the MAM, and connect the two remaining modules directly to obtain the design. In other words, the proposed multi-facet arbiter can be flexibly configured and synthesized with some feature(s) (e.g., $W'I$, $W'I'E'$, WI, and so on) reconfigurable at run-time or fixed at compile-time (synthesis).

VII. Hardware Realization of the Multi-Facet Arbiter Architecture Model

The unified and flexible multi-facet arbiter architecture model, MAM, includes an integrated and composite architecture template and module algorithms, which can be used to derive arbiter hardware. However, MAM does not provide information on how to efficiently implement efficient and/or flexible hardware for the multi-facet arbiter. If the arbiter hardware is implemented directly from the N-input MAM, the speed and area complexities all are $O(N)$; for large-scale arbitration, such cost (of speed and area) is high and thus not suitable for hardware implementation. How to realize cost-efficiency arbiter hardware in terms of circuit speed and area from MAM is still a problem, and will be solved here.

To keep the basic integrated architecture template of the MAM unchanged while maintaining a high performance and/or a low cost, the decentralized modular structure and the parallel hardware implementation are the only ways for implementing efficient hardware for the template. A thorough examination of the MAM architecture template and the respective algorithms of *TOKEN_CONTROL*, *TOKEN_GENERATION*, and *BANDWIDTH* modules shows that they can be implemented using a decentralized parallel tree. First, the algorithm of the *PREEMPTION* module is examined in detail; we found that it can be designed high performance with a parallel tree. Then, due to the modular property of the MAM architecture template, we can combine those respective modules into an integrated multi-facet arbiter with a decentralized and parallel tree hardware structure, called DPT, efficiently, where the *TOKEN_CONTROL, TOKEN_ GENERATION* and *BANDWIDTH* modules are designed and located decentralizedly on individual leaves of the DPT, and the *PREEMPTION* module forms the parallel tree structure of the DPT. How to develop and design this efficient DPT structure and its hardware modules for the multi-facet arbiters are topics of the following subsections.

A. Derivation of the DPT

There are many parallel tree structures, including the binary tree, the quad tree, and the octal tree, with at most two, four, and eight branches at a node, respectively. To find the most appropriate parallel tree structure for the proposed hardware model, the quad tree is used as an example to design the DPT. After the structure of the DPT is designed, the DPT design cases (binary tree, quad tree, and octal tree) are analyzed and the best one for decentralized parallel arbitration is determined.

Algorithm 5 *N_Input_Multi-facet_Arbiter*

Input: $r_0 \sim r_{N-1}$, $w_0 \sim w_{N-1}$, W, I, E, *rst* (*reset signal*)
Output : $g_0 \sim g_{N-1}$(*N output grants*)
begin
  $L \leftarrow \lceil log_4 N \rceil$; // assume $N > 4$ here
  for $i = 0$ **to** $(N/4 - 1)$ **do-parallel**
    LeafNodeArbit($r_{4i} \sim r_{4i+3}$, $w_{4i} \sim w_{4i+3}$, W, I, E, *set_rr*, $g_{4i} \sim g_{4i+3}$, $gg_{i+(4^{L-1}-4)/3}$, $rg_{i+(4^{L-1}-4)/3}$);
end-for
NonleafTree ($rg_0 \sim rg_{(4^L-4)/3-1}$, $gg_0 \sim gg_{(4^L-4)/3-1}$);
4InputRoot(*rst*, $rg_0 \sim rg_3$, $gg_0 \sim gg_3$, *set_rr*); // a $\boldsymbol{R^4}$ *root*
output $g_0 \sim g_{N-1}$;
end

Based on the algorithm of the *PREEMPTION* module, a quad tree-based N-input *DPT* can be derived and defined as a complete 4-ary tree with $L = log_4 N$ levels. Nodes in the *DPT* are partitioned into levels. The node at level 0 is called the root node, which has two cases, one with two and one with four children, respectively, depending on N. In the first case, if $log_2 N$ is even, then the root node with four children is designed, denoted as R^4. In the second case, the root node has two children and is denoted as R^2. The internal nodes, I_i, of the *DPT* are labeled from $i = 0$ to $i = M - 1$ (where M is defined later) from left to right, from the high level to the low level (toward leaves). If the root node is R^4 and $L \geq 2$, then $M = 4(4^{L-2} - 1)/3$; otherwise, $M = 2(4^{L-2} - 1)/3$. Nodes at level $log_4 N - 1$ are called *leaf* nodes, denoted as l_j, and labeled from $j = 0$ to $j = N/4 - 1$ from left to right. Input requests r_{4k}, r_{4k+1}, r_{4k+2}, and r_{4k+3} are processed at *leaf* node l_k. *Leaf* nodes include the combined function of modules *TOKEN_CONTROL, TOKEN_GENERATION, BANDWIDTH*, and the initial part of *PREEMPTION*.

The DPT is highly scalable with input request number N; following its tree structure, large-input DPTs can be recursively designed and constructed from small-input DPTs. An N-input DPT can be constructed using four (*N/4*)-input, or two (*N/2*)-input DPTs and one root node, R^4 or R^2, as shown in Fig. 6(a) and (b), respectively. Consider the root node with four children case as an example for the following design. The DPT hardware algorithm, *N_Input_Multi-facet_Arbiter*, of the multi-facet arbiter with size N is designed as Algorithm 5.

In the hardware algorithm of *N_Input_Multi-facet_Arbiter* and for any other module, the keyword **do-parallel** means that the operations in its scope are executed and interconnected in parallel to reflect their parallel design features. In *N_Input_Multi-facet_Arbiter*, there are three sub-hardware units: the leaf nodes called *LeafNodeArbit*(...), the internal nodes called *NonleafTree*(...), and one root node called *4InputRoot* in the DPT structure to be designed; their designs are described in detail in the following sections.

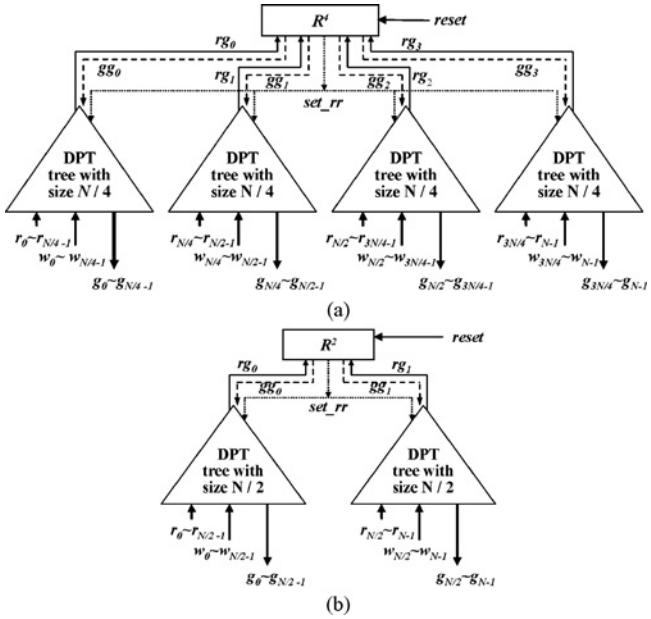

Fig. 6. Recursive construction of DPT trees with (a) 4-input root node and (b) 2-input root node.

Algorithm 6 *LeafNodeArbit*($r_0 \sim r_3$, $w_0 \sim w_3$, *W*, *I*, *E*, *set_rr*, $g_0 \sim g_3$, *en*, *rg*)

Input: $r_0 \sim r_3$, $w_0 \sim w_3$, *W*, *I*, *E*, *en*, *set_rr* (*initial signal*)
Output: $g_0 \sim g_3$, *rg* (*the request to upper level*)
begin
  for *each arbitration time unit* **do**
    for $i = 0$ **to** 3 **do-parallel** *// 4 parallel I/Ps processing*
      $re_i \leftarrow$ *1_InputBW*(*set_rr*, r_i, w_i, g_i, *W*, *en*, re_i);
      $ri_i \leftarrow$ *1_InputTC*(*set_rr*, re_i, D_i, *E*, ri_i, *en*, Q_i);
    end-for
    TG(*I*, $ri_0 \sim ri_3$, $g_0 \sim g_3$, $Q_0 \sim Q_3$, $D_0 \sim D_3$);
    PREM($ri_0 \sim ri_3$, $g_0 \sim g_3$);
    If *en* == 0 **then begin**
      $g_0 \leftarrow 0$; $g_1 \leftarrow 0$; $g_2 \leftarrow 0$; $g_3 \leftarrow 0$;
    end
    $rg \leftarrow ri_0$ **or** ri_1 **or** ri_2 **or** ri_3; // the next level req.
    output $g_0 \sim g_3$, *rg*;
  end-for
end

B. Design of the LeafNodeArbit Structure

The LeafNodeArbit(...) algorithm is designed to efficiently implement any 4-input arbiter listed in Table I based on the algorithms TC, TG, BW, and PREM of modules TOKEN_CONTROL, TOKEN_GENERATION, BANDWIDTH, *and PREEMPTION* of the hardware template in MAM, respectively. Due to modular and structural features in the architecture template algorithm of MAM, the design of the hardware algorithm, LeafNodeArbit(...), is straightforward as shown in Algorithm 6, where *en* is the enable signal, *set_rr* is used to reset the *LeafNodeArbit module*, and *rg* is the request signal to the upper level.

In *LeafNodeArbit*(...), the hardware algorithms of *1_InputBW*(...) and *1_InputTC*(...) are the same as architecture algorithms *BW* and *TC*, respectively, except that they are for just one request input. They are designed as Algorithms 7 and 8.

Algorithm 7 1_*InputTC*(*set_rr*, re_i, D_i, *E*, ri_i, *en*, Q_i)

Input: set_rr, re_i (***1*** *unmasked request input),* D_i, *E, en*
Output: ri_i(*masked request*), Q_i
begin
  if *en ==1* **then** $Q_i \leftarrow D_i$; // *token update*
  if *set_rr == 1* **then** $Q_i \leftarrow 1$; // *set initial value*
  if *E == 1 // for preemption arbitration under I'*
  then $ri_i \leftarrow re_i$
  else $ri_i \leftarrow re_i$ **and** Q_i; *// non-preemp. arbitr. under I' or others.*
end

Algorithm 8 *1_InputBW*(*load*, r_i, w_i, g_i, *W*, *en*, re_i)

Input: load (*BW loading*), r_i(***1*** *request input*), g_i, w_i, *W, en*
Output: re_i (*unmasked request*)
begin
  if *load* == 1 **then** $QB_i \leftarrow QB_i + w_i$;
  if*W* == 1 // with bandwidth constraints
  then
    if QB_i == 0 **then** $re_i \leftarrow 0$ // used up the BW quota
    else begin
      $re_i \leftarrow r_i$;
      if g_i == 1 **and** *en* == 1 **then** $QB_i \leftarrow QB_i$ -1;
    end
  end
  else $re_i \leftarrow r_i$; // without bandwidth constraints
end

The complete hardware diagram of a 4-input *LeafNodeArbit* is shown in Fig. 7. There are four main parts in this arbiter, namely *1_InputBW*, *1_InputTC*, *TG*, and *PREM* modules. The four *1_InputBW* modules are used to realize the 4-input bandwidth–constraint arbitration in parallel by setting parameter *W*. The signal w_i is used to indicate the bandwidth weight of the request (i.e., r_i) whose value is stored in the bandwidth quota (*QB*). We apply a down-counter to record and regulate the *QB*. When the request receives a grant signal (i.e., the g_i signal is high), the value of the down-counter will decrease by one. In the bandwidth-constraint case (i.e., the W signal is high), the *1_inputBW* module acts like a mask circuit that mask the request which used up its *QB* (i.e., the value of *QB* is zero), so the masked request cannot propagate to re_i (i.e., the re_i signal is low regardless of value of the r_i signal); otherwise, the request signal, r_i, is bypassed to the re_i. In the non-bandwidth-constraint case (i.e., the *W* signal is low), the request signal, r_i, is always bypassed to re_i.

The four *1_InputTC* modules are used to realize the 4-input preemptive or non-preemptive arbitration by setting parameter *E*. There is a memory unit (denoted as *Q*) which is updated by the *D* signal generated from the *TG* module. When it is used to realize the preemptive arbitration, i.e., the *E* signal is high, the re_i signal is bypassed to ri_i, i.e., $ri_i \leftarrow re_i$; otherwise, the re_i

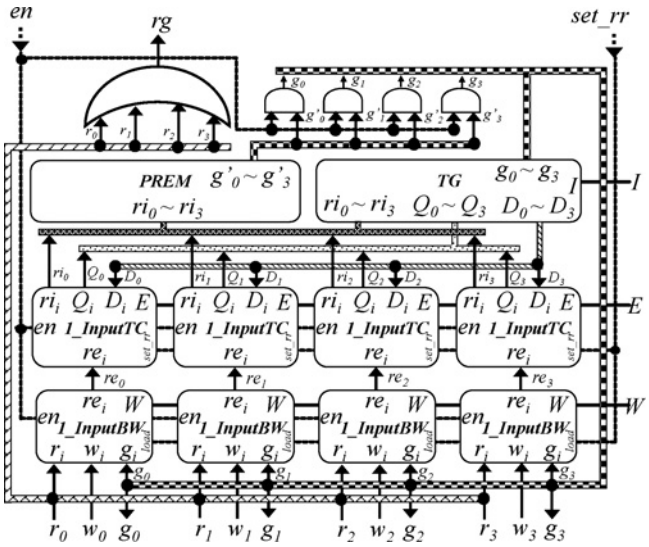

Fig. 7. Hardware diagram of a 4-input *LeafNodeArbit*.

signal is ANDed with the Q signal to realize non-preemptive arbitration, i.e., $ri_i \leftarrow re_i \cdot Q$.

The main function of the *TG* module is to generate the D signals, masking information, to inform the *TC* to mask input requests to realize the equal-priority or the non-preemptive arbitrating property by using multiplexers. The function of the *PREM* module is a linear-priority selector. The outputs of the *PREM* module, i.e., g'_0 ~g'_3, are ANDed with the *en* signal.

The *rg* signal is used to inform the upper level, the *NonleafTree*, whether there is any request signal in it. We use the *set_rr* signal to preset the Q value in the *1_InputTC* module and load the w_i value into *QB* in the *1_InputBW* module. The signal *en* which is received from the upper level, the *NonleafTree*, is used to enable four *1_InputBW* and four *1_InputTC* modules and perform AND logic operation with g'_0 to g'_3 signals. After finishing the designs of hardware implementation algorithms for *LeafNodeArbit*(...) and their relative sub-hardware modules, the hardware algorithms of functions *NonleafTree*(...) and *4InputRoot*(...) are designed as shown below.

C. Design of the NonleafTree Structure

The *NonleafTree* structure is composed of *Internal_Node*s based on the *PREEMPTION* module in the hardware template of *MAM*. If an *Internal_Node* is the leftmost (middle-left, middle-right, and rightmost) child node of its parent node, the input *en* is connected to the output gg_0 (gg_1, gg_2, and gg_3) from the parent node. An *Internal_Node* σ at level $L-2$ is the parent node of four leaf nodes. The inputs rg_0, rg_1, rg_2, and rg_3 of σ are connected to the corresponding output *rg* of its four children leaf nodes, respectively. The hardware algorithm, *NonleafTree*(...), is designed as Algorithm 9.

The hardware design of all *Internal_Node*s and their connecting patterns in the *DPT* is the same, making it scalable and reusable. An *Internal_Node* is used to implement a 4-input linear-priority selector that always selects the input with the highest priority in it as the granted output as follows. It has one output *rg*, which represents the representative request of any request in its four inputs.

Algorithm 9 *NonleafTree*($rg_0 \sim rg_{(4^{\lceil log_4 N\rceil}-4)/3-1}$, $gg_0 \sim gg_{(4^{\lceil log_4 N\rceil}-4)/3-1}$)

```
Input: rg4~rg_{(4^⌈log4 N⌉−4)/3−1}, gg0~gg3
Output: rg0~rg3, gg4~gg_{(4^⌈log4 N⌉−4)/3−1}
begin
  L ←⌈log4 N⌉; // the number of levels
  for i = L-2 downto 1 do
    for j = 0 to 4^i - 1 do-parallel
      RG ← 4(4^i-1)/3 + 4j;
      GP ← (4^i-1)/3 + j -1;
      Internal_Node(rg_RG~rg_RG+3, gg_GP,
                    gg_RG~gg_RG+3, rg_GP);
    end-for
  end-for
  output rg0~rg3, gg4~ gg_{(4^L−4)/3−1};
end
```

Algorithm 10 *Internal_Node*($rg_0 \sim rg_3$, *en*, $gg_0 \sim gg_3$, *rg*)

```
Input: rg0~rg3, en
Output: gg0~gg3, rg
begin
    rg ← rg0 or rg1 or rg2 or rg3;
    if en == 1 then
      if rg0 == 1 then gg0 ←1
      else begin gg0 ← 0;
        if rg1 == 1 then gg1 ← 1
      else begin gg1 ← 0;
        if rg2 == 1 then gg2← 1
      else begingg2←1;
        if rg3 == 1 then gg3← 1 else gg3 ← 0;
      end
    end
  end
  else begin gg0 ← 0; gg1 ← 0; gg2 ← 0; gg3 ← 0; end
  output gg0~gg3, rg;
end
```

D. Design of the Root Node

The case of a root node with four children (nodes) is taken as an example (the process is the same for the root with two children). The root has four request inputs marked as rg_0, rg_1, rg_2, and rg_3 from its four children, respectively. It also provides four grant outputs gg_0, gg_1, gg_2, and gg_3 to notify its leftmost, middle-left, middle-right, and rightmost children, respectively. The main function of the root nodes is to realize the preemption arbitration *PREM*; the hardware algorithms for two cases of the root node are shown in Algorithm 11.

E. Different DPT Tree Structures Evaluation

Why is a quad tree with 4-input *PREEMPTION* modules used to recursively construct the DPT structure instead of the 2-input binary tree or the 8-input octal tree? Taking the design of a typical 16-input WI arbiter as an example, 2-, 4-, and 8-input *PREEMPTION* modules are examined, respectively. With the Synopsys Design Vision (2007.03-sp3)

TABLE II
CRITICAL PATH TIME DELAY RESULTS OF 16-INPUT *DPT*S DESIGNED WITH A BINARY TREE, A QUAD TREE, AND AN OCTAL TREE

	Binary tree	Quad tree	Octree
Timing (ns)	1.87	1.74	2.41

Algorithm 11 *4InputRoot*(rst, $rg_0 \sim rg_3$, $gg_0 \sim gg_3$, set_rr)

```
Input: rg0~rg3, rst (reset signal)
Output: gg0~gg3, set_rr (signal for initialization)
begin
    if rg0 == 1 then gg0 ← 1
    else begin gg0 ← 0;
      if rg1 == 1 then gg1 ← 1
      else begingg1 ← 0;
      if rg2 == 1 then gg2 ←1
      else begin gg2 ← 0;
      if rg3 == 1 then gg3 ← 1 else gg3 ← 0;
      end
    end
end
set_rr ← rst or (rg0 or rg1 or rg2 or rg3)' ;
output gg0~ gg3, set_rr;
end
```

synthesis tool targeting TSMC 0.18 μm technology, Table II shows the design results of the three tree design cases, for which the internal nodes are constructed using 2-, 4-, and 8-input *PREEMPTION* modules, respectively. The 16-input DPT has 1.87, 1.74, and 2.41 ns delays at the critical path for the binary tree, the quad tree, and the octal tree structures, respectively. The quad tree structure has the best performance, so it is adopted in the design. The experiment results show that a tree with more branches in its nodes does not necessarily lead to a shorter critical path delay. The reason is that although the tree with larger branches has a lower height at the non-leaf part, which shortens the delays of the up and down arbitration traces in the tree, it has larger module delays at leaf nodes which still process input requests in a sequential manner, and thus makes the total delay longer.

VIII. DESIGN AND DEVELOPMENT OF THE ARBITER GENERATOR

Based on the features of the modularity, scalability and reusability of the DPT design, an automatic multi-facet arbiter generator, called C AG, which automatically generates a synthesizable Verilog description for an N-input arbiter by combining N single input *BWs* and N single-input TCs, $N/4$ 4-bit TGs, and as many 4-input *PREEMPTION* modules as possible in the DPT structure, is efficiently designed. CAG also automatically generates a testbench file, which is used to give inputs to the generated arbiter, and a waveform file, which is used to record the arbitrating results of the testbench file, for designers to verify the correctness of the generated arbiter. The CAG cooperates with the synthesis tool and invokes it to automatically synthesize the generated arbiter. It also generates

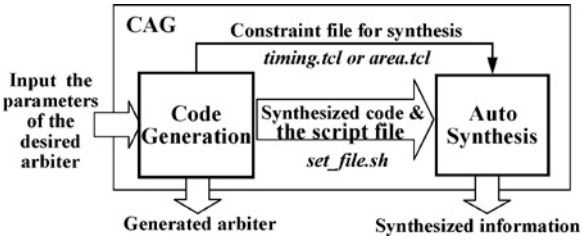

Fig. 8. Block diagram of CAG.

TABLE III
CRITICAL PATH DELAYS (IN TERMS OF FO4 DELAY) FOR SIX ARBITERS

N / Arbiters	4	8	16	32	64	128	256	512
$W'I$	7.52	12.86	15.77	18.31	20.38	23.53	25.47	28.50
$WI'E'$	8.00	10.19	12.61	14.92	16.98	22.80	24.01	26.44
$W'I'E$	1.33	2.06	3.52	5.58	8.37	12.49	14.43	17.71
WI	13.46	17.22	20.13	22.32	24.50	29.96	32.26	36.26
$WI'E'$	14.55	17.59	20.50	23.41	25.96	30.20	31.17	34.69
$WI'E$	20.98	22.80	30.32	36.99	46.21	56.88	59.07	64.52

a post-synthesis Verilog description for the generated arbiter and post-synthesis information about the timing, area, and power consumption.

The CAG mainly consists of two parts: code generation (CG), and automatic synthesis (AS), as shown in Fig. 8. CG generates the arbiter according to the input choices set by the user, and AS invokes the synthesis tool to synthesize the arbiter and generates the synthesized information.

IX. AREA AND PERFORMANCE RESULTS

CAG was implemented in the *C* programming language, and run to generate all *MAM* arbiters listed in Table I for $N = 4, 8, 16, 32, 64, 128, 256$, and 512 using the *Synopsys* Design Vision (2007.03-sp3) synthesis tool and the TSMC 0.18 μm cell library. Tables III and IV show all timing and area results of the six arbiters generated by the CAG tool, respectively; here, all timing results and area results are shown in the number of delay units in terms of a fanout-out-4 inverter gate and the number of the equivalent 2-input NAND gates (NAND2XL in TSMC standard library), respectively; all use the best AT-product synthesis optimization target. CAG is very efficient. Its run time is very small and does not exceed 3 ms for all cases on a 3 GHz Pentium-D PC with 2 GB of main memory. From Tables III and IV, we can find that the proposed multi-facet arbiter hardware designs are scalable in terms of input size N for both area complexity and time complexity. Especially, the delay time complexity of the proposed arbiter is an efficient $O(\log_4 N)$ with its critical path being from the requests to the grants.

Until now, we have not seen any other similar multi-facet arbiter like ours, therefore, we will use the proposed $W'I$ arbiters generated by CAG to compare them with well-known round-robin arbiters SA [3], PRRA and IPRRA [4] in next sub section using the same synthesis tool and the same library described above. On the other hand, for the BW-constraint arbiters such as LOTTERYBUS [9] and CCSP [5] arbiters, we

TABLE IV
AREA RESULTS (NO. OF NAND2XL) FOR SIX ARBITERS

N / Arbiters	*4*	*8*	*16*	*32*	*64*	*128*	*256*	*512*
W′I	63	131	262	533	1071	2150	4317	8606
W′I′E′	77	143	214	451	1099	1628	3850	7177
W′I′E	13	31	106	277	409	501	1328	2346
WI	569	985	1545	3417	7274	12 647	25 105	48 385
WI′E′	503	923	1480	3138	6355	11 573	24 172	46 215
WI′E	337	612	1037	2272	4376	7898	16 381	34 222

TABLE V
TIMING RESULTS (IN TERMS OF FO4 DELAY) FOR *W′I*, *SA*, *PRRA*, AND *IPRRAs*

N / Arbiters	*4*	*8*	*16*	*32*	*64*	*128*	*256*	*512*
W′I	7.52	12.86	15.77	18.31	20.38	23.53	25.47	28.50
SA	9.34	11.16	12.25	16.25	19.16	23.17	26.08	29.96
PRRA	12.01	15.52	19.04	22.68	26.20	29.72	33.23	36.75
IPRRA	11.16	14.68	18.56	21.71	25.35	29.35	33.35	37.48

compute the time and area complexities of them, and compare them with the proposed *WI* BW-constraint arbiter theoretically in Section IX-B.

A. Comparison of Round-Robin Arbiters

The round-robin *W′I* arbiter generated by CGA is compared to existing efficient round-robin arbiters SA [3], PRRA, and IPRRA [4]. Table V shows the timing results of the designs in terms of FO4 delay. Although the delay of *W′I* is slightly higher than that of *SA* for the cases with a small number of inputs, it outperforms the *SA* for the cases with a large number of inputs, and guarantees fairness. Additionally, the *SA* is not fair for the non-uniformly distributed requests [4], and the growing demand for a large number of inputs is the necessaries in the modern MPSoCs with thousands of cores [1], [2]. From the above point of view, our *W′I* still has the advantage of the performance.

W′I is faster than other designs with delay time improvements of 25.3% over PRRA and 21.5% over IPRRA for the 512-input case. As shown in Fig. 9, the critical path delays of SA and *W′I* grow with $\log_4 N$ while others grow with $\log_2 N$. The time delay of *W′I* grows slower than that of SA because of its scalable design for internal nodes and the root node in the DPT structure. The timing improvements of *W′I* over PRRA and IPRRA are more significant for large scales since the DPT has fewer levels compared to those of PRRA and IPRRA.

Table VI shows the area results of the designs in terms of the number of NAND2XL gates. Compared with PRRA and IPRRA, *W′I* requires fewer NAND2XL gates in all cases due to its efficient DPT structure. *W′I* also has lower area costs than those of *SA* in all cases since most of its nodes are implemented in scalable combinational logic except leaf nodes. *W′I* has the lowest area cost of all known round-robin arbiters. As shown in Fig. 10, the area costs of all designs grow linearly with *N*. The area improvements of *W′I* over the other designs are proportional to the number of input requests due to the proposed scalable design, simple implementation, and the complete design model and template.

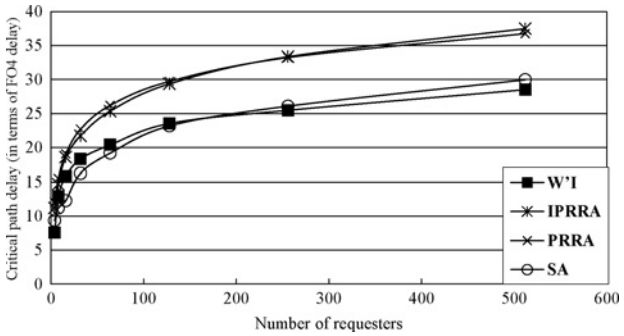

Fig. 9. Growing curves of critical path delays (in terms of FO4 delay) for four arbiters.

TABLE VI
AREA RESULTS FOR *W′I*, *SA*, *PRRA*, AND *IPRRA*

N / Arbiters	*4*	*8*	*16*	*32*	*64*	*128*	*256*	*512*
W′I	63	131	262	533	1071	2150	4317	8606
SA	72	198	420	894	1814	3685	7395	14 846
PRRA	72	161	341	702	1423	2865	5749	11 518
IPRRA	71	155	326	668	1355	2728	5484	10 965

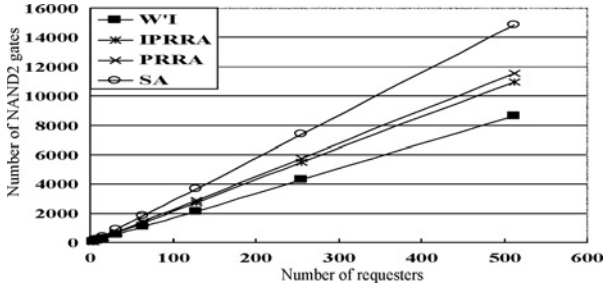

Fig. 10. Area comparison of *W′I*, IPRRA, PRRA, and SA in terms of the number of NAND-2 gates.

In summary, the performance and area of the proposed *W′I* arbiter all are almost better than others. One of main reasons is that the proposed arbiter uses the DPT tree, which is a decentralized quad tree structure. The tree used in PRRA and IPRRA is a centralized binary tree with a ring interconnection connecting each input; SA also uses a centralized quad tree. So, the proposed multi-facet arbiter could obtain the batter timing and area results.

B. Comparisons of BW-Constraint Arbiters

In this section, the WI arbiter is compared to existing BW-constraint arbiters LOTTERYBUS [9] and CCSP [5] only in terms of their delay time and area complexities according to their respective hardware structures, because their detailed circuit designs are unknown.

All the BW-constraint arbiters use BW weights to indicate the BW quantity held by each requester. For simplicity, it is assumed the width of the BW weight value is *b*. After some non-trivial computations (omitted here), the time and area complexities of LOTTERYBUS, CCSP and *WI* were obtained

TABLE VII
AREA AND TIME COMPLEXITIES OF BW-CONSTRAINT ARBITERS

Arbiters	Delay Complexity	Area Complexity
LOTTERYBUS	$O(\log_2 blog_2 N)$	$\geq O(b^2 N)$
CCSP	$O(\log_2 N)$	$O(b^2 N)$
Our *WI*	$O(\max(\log_4 N, \log_2 b))$	$O(bN)$

as shown in Table VII. Generally, the value of b, the width of the BW weight, is usually small compared with the value of N, the number of requesters. Thus, *WI* generated by the CAG tool with the DPT structure based on the MAM model has the smallest critical path delay and the smallest area complexity compared to other BW-constraint arbiters. Its time and area complexities are equal to $O(\log_4 N)$ and $O(N)$, respectively.

X. CONCLUSION

In this paper, we presented an efficient new multi-facet arbiter design and implementation, its automatic generator, as well as a first example of the new model-driven systematic design flow from 3-phase model-driven design methodology. The designs are reusable, composite and modular, expansible, and finally efficiently scalable. The former three features are inherited from its new design and architecture models, the latter is due to the usage of the decentralized parallel tree implementation and the modular implementation techniques, which not only help us to design a new fast and small round-robin arbiter and other type efficient arbiters, but also let the automatic generation of those designs quickly. The design approach presented in here is a first time that such a systematic 3-phase model-driven template-based design methodology is proposed for the multi-facet arbiter design. In the future, we shall extend and generalize the model-driven systematic design approach and the system to design other hardware and some special arbiters such as *WiMAX* real time arbiter [12].

ACKNOWLEDGMENT

The authors would like to thank Prof. Y.-L. Jeang and Y.-Y. Chen for their help.

REFERENCES

[1] A. Kumar, B. Mesman, H. Corporaal, J. van Meerbergen, and H. Yajun, "Global analysis of resource arbitration for MPSoC," in *Proc. 9th EUROMICRO Conf. Digital Syst. Des.*, 2006, pp. 71–78.
[2] S. Borkar, "Thousand core chips: A technology perspective," in *Proc. ACM/IEEE 44th Des. Autom. Conf.*, Jun. 2007, pp. 746–749.
[3] E. S. Shin, V. J. Mooney, and G. F. Riley, "Round-robin arbiter design and generation," in *Proc. ISSS*, 2002, pp. 243–248.
[4] S. Q. Zheng and M. Yang, "Algorithm-hardware codesign of fast parallel round-robin arbiters," *IEEE Trans. Parallel Distributed Syst.*, vol. 18, no. 1, pp. 84–95, Jan. 2007.
[5] B. Akesson, L. Steffens, E. Strooisma, and K. Goossens, "Real-time scheduling using credit-controlled static-priority arbitration," in *Proc. Embedded Real-Time Comput. Syst. Applicat.*, 2008, pp. 3–14.
[6] K. Keutzer, S. Malik, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincentelli, "System level design: Orthogonalization of concerns and platform-based design," *IEEE Trans. Comput.-Aided Des.*, vol. 19, no. 12, pp. 1523–1543, Dec. 2000.
[7] F. Wang and M. Hamdi, "Fast fair arbiter design in packet switches," in *Proc. High Performance Switching Routing*, 2005, pp. 472–476.
[8] K. Lee, S.-J. Lee, and H.-J. Yoo, "A high-speed and lightweight on-chip crossbar switch scheduler for on-chip interconnection networks," in *Proc. Eur. Solid-State Circuits Conf.*, Sep. 2003, pp. 453–456.
[9] K. Lahiri, A. Raghunathan, and G. Lakshminarayana, "LOTTERYBUS: A new high-performance communication architecture for system-on-chip designs," in *Proc. Des. Autom. Conf.*, 2001, pp. 15–20.
[10] H. J. Chao, C. H. Lam, and X. Guo, "A fast arbitration scheme for terabit packet switches," in *Proc. IEEE Global Telecommun. Conf.*, Dec. 1999, pp. 1236–1243.
[11] P. Gupta and N. McKeown, "Designing and implementing a fast crossbar scheduler," *IEEE Micro*, vol. 19, no. 1, pp. 20–28, Jan.–Feb. 1999.
[12] T.-M. Tsai, "Method for real-time uplink traffic scheduler in WiMAX mobile system," *SoC Tech. J.*, vol. 10, pp. 48–56, May 2009.

Jer Min Jou received the Ph.D. degree in electrical engineering and computer science from National Cheng Kung University, Tainan, Taiwan, in 1987.

In 1989, he was an Associate Professor with the Department of Electrical Engineering, National Cheng Kung University, where he is currently a Professor. His current research interests include systems-on-a-chip hardware-software co-design, system design, application-specific integrated circuit design/synthesis, very-large-scale integration computer-aided design, and asynchronous circuit design.

Dr. Jou was the recipient of the Distinguished Paper Citation from the 1987 IEEE ICCAD Conference, Santa Clara, CA, the Longterm Best Paper Award from the Acer Foundation in 1998 and 1999, the First Level of 2001 IP Competition sponsored by the Ministry of Education, China, the Best Paper Award from the Workshop on Consumer Electronics and Signal Processing in 2005, the Best Paper Award from the International Computer Symposium in 2008, and the SpringSoft EDA Award in 2010.

Yun-Lung Lee received the B.S. degree in electronic engineering from Feng Chia University, Taichung, Taiwan, in 2005. He is currently working toward the Ph.D. degree in electrical engineering from the Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan.

His current research interests include processor systems and arbiter design.

Mr. Lee won the Best Paper Award from the International Computer Symposium in 2008 and the SpringSoft EDA Award in 2010.

Sih-Sian Wu received the B.S. degree in electrical engineering from National Cheng Kung University, Tainan, Taiwan, in 2005. He is currently working toward the M.S. degree in electrical engineering from the Department of Electrical Engineering, National Cheng Kung University.

His current research interests include very large scale integration design and chip design.