ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.2, February 2007, pp.220−228
DOI: 10.1360/jos180220

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

系统级综合中结合资源分配的调度算法*

吴　强[1+]，边计年[2]，薛宏熙[2]

[1](湖南大学　计算机与通信学院,湖南　长沙　410082)
[2](清华大学　计算机科学与技术系,北京　100084)

Scheduling with Resource Allocation for System-Level Synthesis

WU Qiang[1+], BIAN Ji-Nian[2], XUE Hong-Xi[2]

[1](College of Computer and Communication, Hu'nan University, Changsha 410082, China)
[2](Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)
+ Corresponding author: Phn: +86-731-8821715, Fax: +86-731-8821715, E-mail: wuqiang@hnu.cn, http://lecs.hnu.cn

Wu Q, Bian JN, Xue HX. Scheduling with resource allocation for system-level synthesis. *Journal of Software*, 2007,18(2):220−228. http://www.jos.org.cn/1000-9825/18/220.htm

Abstract: In system-level synthesis, the allocation of resources is always decided by the designer or explored in the outer-most loop. In this paper, a heuristic scheduling algorithm is proposed to find the resource allocation during its running process. It determines the appropriate number of required resource instances based on the system partition in scheduling, and generates the corresponding resource allocation, scheduling and assignment solution. Such an algorithm can simplify the system-level design exploration to a procedure of system partitioning, scheduling and evaluation, and can improve the exploration efficiency. Experimental results show the feasibility and validity of the approach.
Key words: task scheduling; resource allocation; heuristic algorithm; design space exploration; system-level synthesis

摘　要: 在系统级综合中,资源的分配通常由设计者指定,或在设计空间搜索的最外层循环中进行枚举探索.提出了一种结合资源分配的启发式调度算法.它根据当前系统划分的结果,在调度过程中寻找合适的所需资源实例的数目,从而确定系统的资源分配以及调度指派方案.应用该调度算法可使设计空间搜索过程简化为划分、调度和评估三个步骤,省去了最外层的资源分配枚举循环,提高了搜索效率.实验结果验证了该算法的可行性和有效性.
关键词: 任务调度;资源分配;启发式算法;设计空间搜索;系统级综合
中图法分类号: TP301　　文献标识码: A

1 Introduction

System design exploration is very important for system-level synthesis (SLS) of embedded systems, which

* Supported by the National Natural Science Foundation of China under Grant Nos.90207017, 60236020 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2003AA115110 (国家高技术研究发展计划(863))

Received 2003-11-11; Accepted 2005-07-08

attempts to find a best design solution according to the performance, power consumption and price goals[1,2]. Intensive research efforts have been made to address this issue. Many of them assume a fixed architecture or provided by the designer. In Ref.[3], the author adopts a fixed architecture template that consists of one microprocessor and several logic blocks. In Ref.[4], the system implementation architecture is interactively improved with the SystemC based co-simulation tool manually. In SpecSyn, the architecture specification is supplied by the designer, and is evaluated and refined in the succeeding steps[5]. Later in Ref.[5], Peng and Abdi proposed algorithms to perform automatic model refinements for the architectures provided by the designer or generation tools. In M Dziri's full SoC design flow, VCC is employed to do the architectural exploration, which needs manual interactions too[7].

Some research works consider the automated architecture generation, which concerns the partitioning, resource allocation, and task scheduling and assignment problems. In Ref.[8], a method is presented to do flexible design exploration with architectural allocation, where available types and maximal number of resources are predetermined. In Véstias' rapid prototyping platform, architectures are explored in outer loop of the co-synthesis flow. Resource instances are inserted one-by-one until the maximal number of resource instance is reached[9]. In SOS, Prakash and Parker proposed a mixed integer linear programming model to automatically synthesize an architecture with arbitrary topology[10]. But their algorithm has difficulty to deal with large systems due to its high time complexity. Wolf then used a heuristic approach to deal with this problem[11]. In his algorithm, resources are allocated before partitioning, and then reduced after scheduling by eliminating resource instances without tasks assigned on them. Xie and Wolf extended this work to deal with conditional task graphs in Ref.[12]. In most of these efforts, scheduling algorithms always reside in the inner-most loop of the design exploration to provide evaluation of the design solution. Partitioning algorithms are responsible to optimize the partition decision, which is often placed outside the scheduling procedure. Resource allocation is often placed in the outer-most loop of the design exploration.

In this paper, we propose a scheduling algorithm which produces allocation, schedule and assignment in one run. With such a scheduling algorithm, the design exploration flow can be simplified to an iterative procedure of partitioning, scheduling and evaluation, eliminating the outer-most architectural exploration loop. This, in our point of view, will be helpful for a fast and efficient design exploration at system level. Such a design flow has been introduced into a system-level synthesis framework for SoC design.

2 System Model

2.1 Functional model

We use the task graph[13] as the functional description of the system, which is a directed acyclic graph (DAG), $G=\langle V,E\rangle$, with node set V and edge set E. Each node v_i represents a task of the system. Each edge $\langle v_i,v_j\rangle$ represents the data dependency and communication between the two connected nodes.

Weights associated with nodes include task type tt and a deadline dl. Task type indicates what type of function performed by this node. Deadline indicates that the task execution of the node must finish before the time designated. Weight associated with edges is the communication data quantity cq, which indicates how much data will be transferred when the communication of this edge is committed.

2.2 Architectural model

Resources that implement the system function are modeled as processing elements (PE) and communication channels (CH). A PE is a component that executes the tasks, which can be a microprocessor with local memory and internal bus, or an application specific integrated circuit (ASIC), or even an IP core. A CH is a component that executes the communication between tasks, which can be a bus with access arbitrator, or a shared memory with management circuit, or even an IP core implementing a particular communication protocol. The whole system can be viewed as several PEs connected with a network of CHs as outlined in Fig.1.

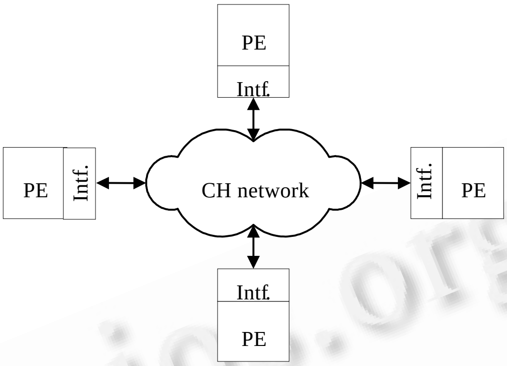


Fig.1 Architectural model

In system synthesis, nodes in task graph are mapped to PEs, while edges to CHs. A resource library is built to hold the attributes of the available PEs and CHs.

The attributes of chip area, price and idle power consumption are associated with each PE type only, while the execution time and power consumption are associated with task types. Different types of PE will have different combination of these attribute values. Apparently, a type of PE can execute several types of tasks, and a type of task can run on many candidate types of PEs.

For CHs, concerned attributes include the average chip area per link, average price per link, idle power consumption, average transfer speed, and average transfer power consumption per bit in correspondence with interface types. With the interface type, we mean the communication protocol and interface of a CH instance. Obviously, transfer speed and power consumption are not necessarily the same under different interface types. Furthermore, interface type of a CH instance should be compatible with those of PE instances it connects. Here, we assume that the interface type of a PE is determined by its own type.

3 Scheduling Algorithm

3.1 Solution representation

As mentioned in the first section, the resource allocation will be produced during the scheduling process. It can be represented with two sets of instances, one for PEs, the other for CHs. The label of each instance indicates the resource type and the serial number of the instances of this type.

Allocation $A::=\langle PEA,CHA\rangle$, where $PEA::=\{pe_{00},\dots,pe_{ij},\dots,pe_{nm}\}$, $CHA::=\{ch_{00},\dots,ch_{kl},\dots,ch_{pq}\}$.

We appoint CH type 0 as the internal link CH type, and always allocate one instance of this type with label ch_{00}. All the communications occurring between the nodes on the same PE instance will be placed on this instance.

The representation of the schedule and assignment is relatively simple and straightforward as follows.

Schedule $S::=\langle VS,ES\rangle$, where $VS::=\{\langle v_i,ts_i,pe_{ij}\rangle\}, VE::=\{\langle e_t,ts_t,ch_{kl}\rangle\}$.

Other terms and expressions used are listed below: $ASAP(n_i)$ is the As Soon As Possible start time of the node or edge n_i; $ALAP(n_i)$ is the As Late As Possible start time of the node or edge n_i; $SLACK(n_i)$ is defined as $SLACK(n_i)=ALAP(n_i)-ASAP(n_i)$ which serves as a measure of priority of the node or edge n_i.

3.2 Main flow

We choose a list-scheduling scheme as the main flow of the algorithm as shown in Fig.2.

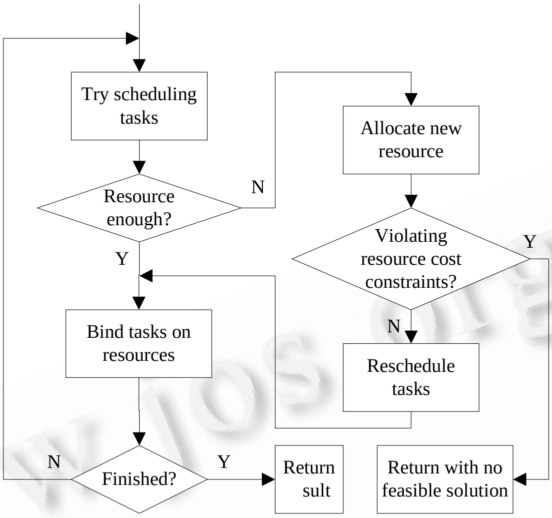


Fig.2 Main flow

It initializes the ready list with the source nodes of the task graph, then repeats to select a task or communication from the ready list, schedule and bind it, and insert all successors of the scheduled task or communication into the ready list. When the ready list is empty, which means all tasks and communications have been scheduled and bound, it finishes and returns the result to the design exploration procedure.

Note in the above procedure, the ASAP and ALAP start time are initialized at the beginning of the scheduling process with the assumption of infinite resources. During the scheduling process, they will be updated to reflect actual resource allocation and schedule of the nodes and edges.

3.3 Scheduling of nodes and edges

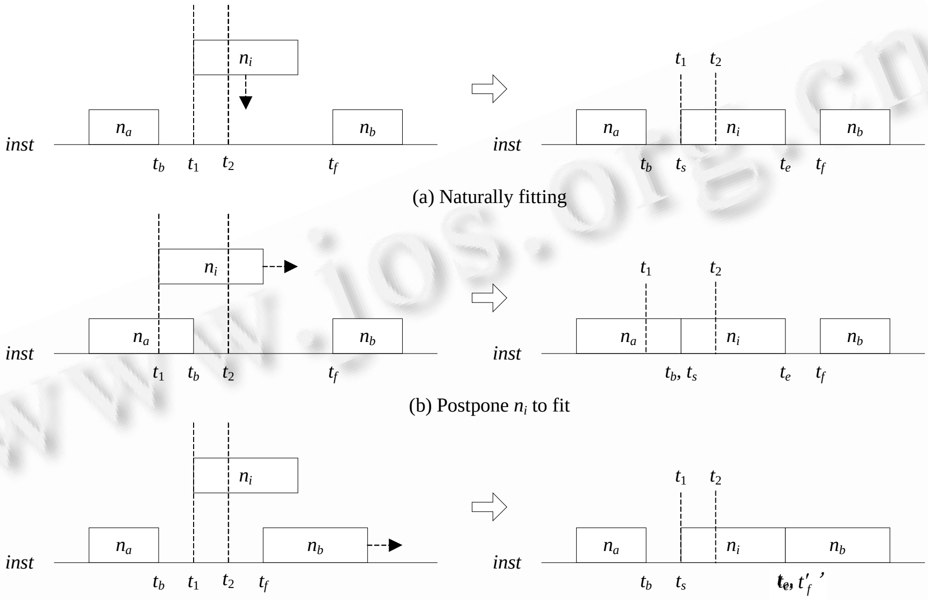
The scheduling of nodes and edges on the resource instances are alike, which is outlined below:

1. Let n_i be the node or edge to be scheduled. Collect all the instances which have the same type as that of n_i's partition into the set *vinsts*. If *vinsts*=∅, allocate a new instance of this type and insert it into *vinsts*;
2. For each instance *inst* in *vinsts*, check if there exists a vacant interval on it for n_i. If so, *inst*→*vains*;
3. If *vains*=∅, allocate a new instance for n_i. Reschedule the previously delayed nodes or edges to utilize this new instance and determine the start time for n_i on it;
4. Else *vains* is not empty, select an instance that can provide the earliest start time for n_i from *vains*. Schedule and assign n_i with this start time as the n_i on this instance;
5. If the start time of n_i is greater than its ASAP start time, then record n_i as a delayed node or edge.

In the above procedure, the algorithm will check all the current resource instances of n_i's type, say *vinsts*, to find a available vacant interval fit for n_i. Figure 3 gives the details of this check on an instance *inst* from *vinsts*. In the figure, t_1 and t_2 are the $ASAP(n_i)$ and $ALAP(n_i)$ respectively. t_d is the run time of n_i on this type of resource. A possible vacant interval for n_i is $[t_b,t_f]$. t_b and t_f are the end time of n_a and start time of n_b respectively, which have been already scheduled and assigned on the *inst*. The term "possible interval" means $t_b \le t_2$.

- If $t_b \le t_1 \wedge t_f \ge t_1 + t_d$, the n_i can be naturally fitted in the interval, as shown in Fig.3(a).

- If $t_1 \le t_b \le t_2 \wedge t_f \ge t_b + t_d$, the n_i can be pushed and fitted in the interval $[t_b, t_b + t_d]$, as shown in Fig.3(b).
- If $t_b \le t_1 \wedge \ge t_b + t_d \ge t_f$, we can push n_b to fit n_i in the interval $[t_1, t_1 + t_d]$, satisfying that pushing n_b will not cause n_b violate its deadline, as shown in Fig.3(c).
- If $t_1 \le t_b \le t_2 \wedge \ge t_b + t_d \ge t_f$, this can be regarded as the combination of the above two cases. We can push n_i and n_b to fit n_i in the interval $[t_b, t_b + t_d]$, satisfying that pushing n_b will not cause n_b violate its deadline. For other cases, there is no vacant interval fit for n_i on *inst*.



(a) Naturally fitting

(b) Postpone n_i to fit

(c) Postpone n_b to fit

Fig.3 Fit n_i in vacant interval on *inst*

3.4 Rescheduling of delayed nodes and edges

After the check of the available instances, all the instances that can accommodate n_i will be found out and stored in *vains* with the corresponding start time. But cases may happen that no instances can provide n_i a vacant interval. In such cases, a new instance is allocated for n_i. Rescheduling of the previously scheduled nodes and edges on other instances of the same resource type is performed to make use of the newly allocated instance. Intuitively, only those nodes or edges that are delayed in the previous scheduling are worthy of considering, since rescheduling un-delayed nodes or edges will not make them start earlier or occupy less resource. Rescheduling is described in below.

1. Let *newinst* be the newly allocated instance for n_i. Schedule n_i on *newinst* at its ASAP start time;
2. Find all the delayed nodes or edges that can be rescheduled to the *newinst*, store them in the set *resched*;
3. Select the node or edge n_k with the minimal slack in *resched*, and try to schedule it on *newinst*. During this process, postponing operation may also be performed to get n_k fit in the vacant intervals on *newinst*;
4. If n_k can be rescheduled earlier on *newinst*, move it to *newinst*. Update its ASAP start time, as well as those of its successors;
5. Delete n_k from *resched*. If *resched*=∅, stop, else goto step 3.

Note that the postponed operation performed in the rescheduling is a little different from that in the original scheduling process. In rescheduling, the postponed operation should not postpone any edges or nodes even behind their originally scheduled start time. In this sense, the postponed rules are tighter for rescheduling than for the original scheduling.

In the worst case, rescheduling will cause the check on all the previously scheduled tasks and communications, which costs at most $O(n+m)$ time. Combined with the $O(n+m)$ time of the list-scheduling scheme of the main algorithm flow, the total time complexity in the worst case will be $O((n+m)^2)$. Here n is the number of the nodes, and m is the number of edges of the task graph.

4 Experimental Results

4.1 Feasibility

We implement the scheduling algorithm in C++ and take some tests on a v880 machine running Sun Solaris. 8 types of PE and 4 types of CH are generated by TGFF as the resource library. Then we apply the scheduling algorithm with the partitioning algorithm on a task graph example generated by TGFF shown in Fig.4. The screen shots of the results outputted by the scheduling algorithm at 3 steps of the design exploration procedure are shown in Fig.5. In the figure, white boxes are PE or CH instances, while the grey bars on them are nodes or edges assigned on these instances. Grey lines are used to indicate relations of nodes and edges. For example, node 0 connects node 1 with an edge, namely 0. Then, a line is drawn from the bar *t*0_0 to the bar *a*0_0, and a line is from bar *a*0_0 to bar *t*0_1.

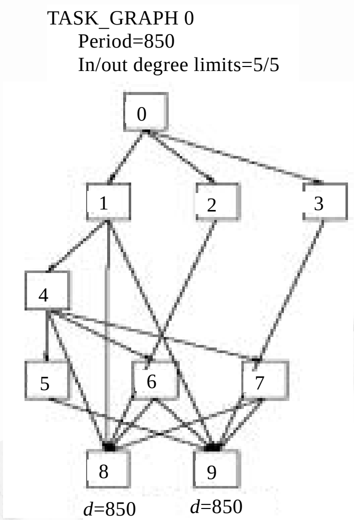
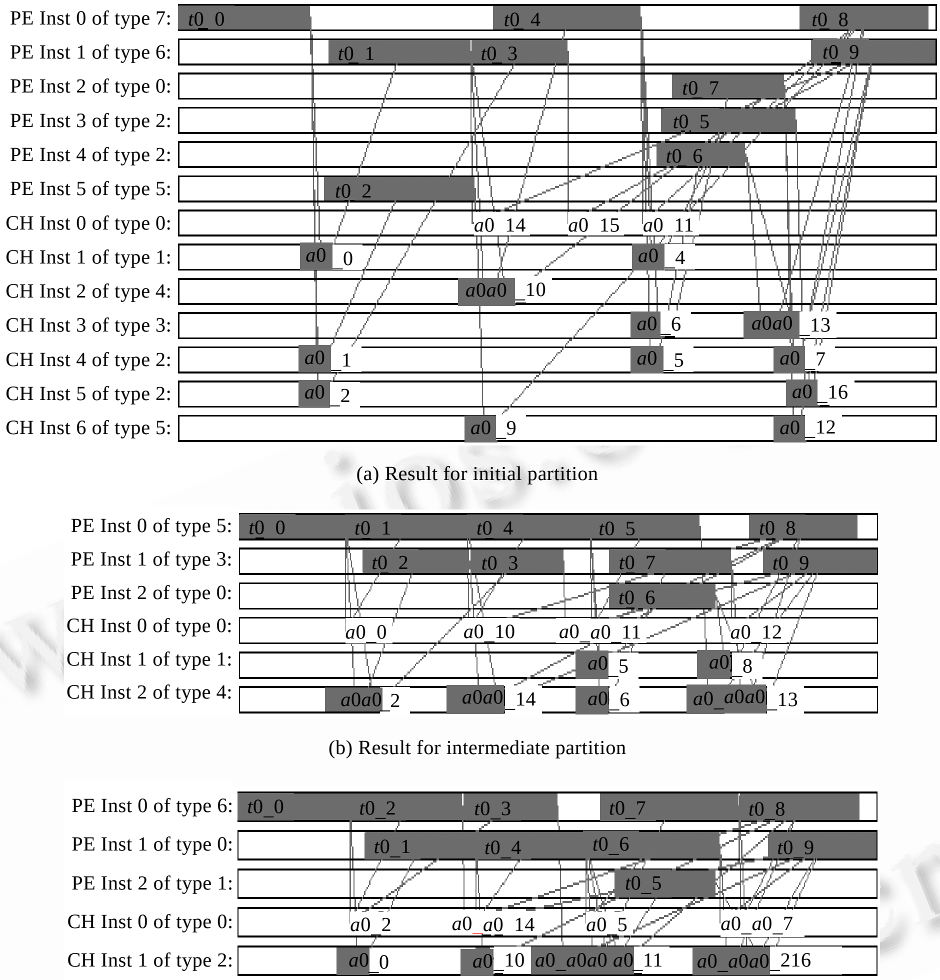


Fig.4 A task graph example

Sub-Diagram Fig.5(a) is the result of the initial randomly-generated partition. Sub-Diagram Fig.5(b) corresponds to an intermediate partition during the whole optimization process. Sub-Diagram Fig.5(c) is the result corresponding to the final partition solution. It can be seen that the result for the final partition allocates 3 PE instances with type 6, 0 and 1 respectively, as well as only 1 CH instance of type 2. Communications other than those scheduled on the only CH instance are all executed on the internal link. This can be viewed as an architecture consisting of two processors with a dedicated functional component connected with a system bus.

We also take experiments on other task graph examples and the resultant PE and CH numbers are listed in Table 1. Note that the internal link CH is not counted in the CH number. Apparently, the scheduling algorithm can produce a reasonable result under the given partition.



(a) Result for initial partition

(b) Result for intermediate partition

(c) Result for final partition

Fig.5 Results for partitioning steps

4.2 Run time

To examine the run time feature of the proposed scheduling algorithm, we take experiments on 5 task graph examples with various node numbers of 10 to 200. These task graphs are also generated by TGFF. We repeatedly run the program on these 11 task graph examples for 100 times with different partitions, record the time consumed and calculate the average run time of each graph size. The results are collected in Table 2 below. Note the run time is recorded in millisecond.

Obviously, the scheduling algorithm runs very fast, no more than 0.06 second for the task graph with 50 nodes and 109 edges, and about 3.2 second for 200 nodes and 432 edges. The increasing trend of the run time complies with the analysis in previous section, which indicates a time complexity of $O((n+m)^2)$.

It should be noted that in our experiments, the scheduling algorithm is executed with the partitioning algorithm, which generates and accepts partitions under optimization rule. For each run, a large number of partitions are generated and compared. But the whole process runs smoothly and quickly, owing to the simplification of the entire design exploration procedure introduced by the proposed scheduling with resource allocation heuristic. We believe this will be very advantageous in the design exploration of SLS for SoC designs.

Table 2 Results of task graph examples

Task graph	Node num.	Edge num.	Arch. (PE /CH num.)	Avg. run time (ms)	Max. run time (ms)
T10	10	17	3/1	2.2	2.5
T20	20	47	4/3	9.6	10.1
T30	30	82	5/5	28.6	29.3
T40	40	91	7/5	36.2	38.4
T50	50	109	7/5	56.7	59.6
T60	60	159	9/5	141.4	163.0
T70	70	181	10/5	195.6	230.1
T80	80	203	11/5	267.2	329.9
T90	90	213	15/7	341.4	362.6
T100	100	235	15/7	444.6	485.2
T200	200	432	30/13	3203.0	3335.1

5 Conclusion

In this paper, a heuristic algorithm that can perform allocation and assignment along with the scheduling is presented. The original idea is based on the observation that the allocation of the resources can be deduced from the partition decision and the resource requirement arisen in the scheduling and assignment. In the scheduling, tasks and communications are postponed within their slacks to get fit in vacant intervals on resource instances. Rescheduling is performed to make use of the newly allocated instances. Preliminary experiments show the feasibility of the algorithm. Reasonable allocation, scheduling and assignment solution can be obtained for a given partition. Such a scheduling algorithm can simplify design exploration flow to an iterative procedure of partitioning, scheduling and evaluation, which will be helpful for the efficiency in the system-level synthesis. Currently we are attempting to integrate the proposed algorithm with the front-end compiler under development into a system-level synthesis framework, which is intended to transform the system-level functionality described with C or VHDL to the synthesizable RTL codes for system implementation.

References:

[1] Wolf W. A decade of hardware/software codesign. IEEE Computer, 2003,36(4):38−43.

[2] Eles P, Kuchcinski K, Peng Z. System Synthesis with VHDL. Boston: Kluwer Academic Publishers, 1998.

[3] Xiong ZH, Li SK, Chen JH. Hardware/Software partitioning based on dynamic combination of genetic algorithm and ant algorithm. Journal of Software, 2005,16(4):503−512 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/16/503.htm

[4] Jang H, Kang M, Lee M, Chae K, Lee K, Shim K. High-Level system modeling and architecture exploration with SystemC on a network SoC: S3C2510 case study. In: Figueras J, ed. Proc. of the Design, Automation and Test in Europe. Paris: IEEE Press, 2004. 538−543.

[5] Gajski D, Vahid F, Narayan S, Gong J. SpecSyn: An environment supporting the specify-explore-refine paradigm for hardware/software system design. IEEE Trans. on Very Large Scale Integration Systems, 1998,6(1):84−100.

[6] Peng J, Abdi S, Gajski D. Automatic model refinement for fast architecture exploration. In: Sherlekar SD, ed. Proc. of the Conf. on Asia South Pacific Design Automation/VLSI Design. Bangalore: IEEE Computer Society, 2002. 332−337.

[7] Dziri M, Samet F, Wagner F, Cesario W, Jerraya A. Combining architecture exploration and a path to implementation to build a complete SoC design flow from system specification to RTL. In: Yasuura H, ed. Proc. of the Aisa South Pacific Design Automation Conf. Kitakyushu: IEEE Computer Society, 2003. 219−224.

[8] Ernst R, Haubelt C, Richter K, Teich J. System design for flexibility. In: Franca J, ed. Proc. of the Design, Automation and Test Conf. in Europe. Paris: IEEE Computer Society, 2002. 854−861.

[9] Véstias M, Neto H. System-Level co-synthesis of dataflow dominated applications on reconfigurable hardware/software architectures. In: Kordon F, Henkel J, eds. Proc. of the 13th IEEE Int’l Workshop on Rapid System Prototyping. Darmstadt: IEEE Computer Society, 2002. 130−137.

[10] Prakash S, Parker AC. A design method for optimal synthesis of application-specific heterogeneous multiprocessor systems. In: Gottlieb A, ed. Proc. of the 19th Annual Int'l Symp. on Computer Architecture. Queensland: IEEE Computer Society, 1992. 75−80.

[11] Wolf W. An architectural co-synthesis algorithm for distributed embedded computing systems. IEEE Trans. on VLSI Systems, 1997,5(2):218−229.

[12] Xie Y, Wolf W. Allocation and scheduling of conditional task graphs in co-synthesis. In: Jerraya A, ed. Proc. of the Design, Automation and Test Conference in Europe. Munich: IEEE Computer Society, 2001. 620−625.

[13] Dick R, Rhodes D, Wolf W. TGFF: Task graphs for free. In: Borriello G, Jerraya A, Lavagno L, eds. Proc. of the 6th Int'l Workshop on Hardware/Software Codesign. Seattle: IEEE Computer Society, 1998. 97−101.

附中文参考文献:

[3] 熊志辉,李思昆,陈吉华.遗传算法与蚂蚁算法动态融合的软硬件划分.软件学报,2005,16(4):503−512. http://www.jos.org.cn/1000-9825/16/503.htm

WU Qiang was born in 1974. He is an associate professor of Hunan University. His current research areas are SOC oriented system design automation and reconfigurable computing.

XUE Hong-Xi was born in 1938. He is a professor of Tsinghua University. His research area is digital system design automation.

BIAN Ji-Nian was born in 1945. He is a professor of Tsinghua University and a CCF senior member. His research area is SOC oriented system design automation.