第 23 卷第 6 期 半 导 体 学 报 Vol. 23, No. 6
2002 年 6 月 CHINESE JOURNAL OF SEMICONDUCTORS June, 2002

Hardware Optimization Technique of Full-Customized HW/SW Co-Design*

Tang Lei[1], Wei Shaojun[2] and Qiu Yulin[1]

(1 *Microelectronics R&D Center, The Chinese Academy of Sciences, Beijing* 100029, *China*)
(2 *Institute of Microelectronics, Tsinghua University, Beijing* 100084, *China*)

Abstract: The hardware optimization technique of mono-similarity-system generation is presented based on hardware/software(HW/SW) co-design. First, the coarse structure of sub-graphs' matching based on full-customized HW/SW co-design is put forward. Then, a universal sub-graphs' combination method is discussed. Next, a more advanced vertexes' compression algorithm based on sub-graphs' combination method is discussed with great emphasis. Experiments are done successfully with perfect results verifying all the formulas and the methods above.

Key words: HW/SW co-design; CDFG; mono-similarity-system; sub-graph; compound-graph; combine; usage-degree; cost

EEACC: 2570

CLC number: TN47 **Document code**: A **Article ID**: 0253-4177(2002)06-0637-08

1 Introduction

Along with the coming of ULSI, hardware/software (HW/SW) co-design methodology was put forward and researched by many savants. But most of the developed algorithms are IP-SOC based on HW/SW co-design methods. They are technically suitable for IP based half-customized co-design, not for a HW/SW co-design from scratch.

HWs, in an exact meaning, should be defined as all those functional units (FUs), while SW is only those, who control codes setting all the HWs' working states all the time, and organizing them to achieve the system's whole task. But in IP-SOC based on HW/SW co-design, HW is usually defined as non-programmable IP, such as ASIC, FPGA, etc, while SW is coarsely defined as programmable IP, such as MCU, DSP, etc. From this way, HW/SW co-synthesis methods for ULSI design are simply derived from High-Level Synthesis (HLS), which is very successfully used in VLSI design, only by replacing "+, -, *, /", the basic FUs of HLS, with IP modules. But, during this transformation, a "fine-grain" method was changed into a "coarse-grain" method. In such an IP-based method, the function of any IP can not be changed freely when this IP is used in some special control data flow graphs (CDFG), the performance of any IP also can not be changed freely when it is used

* Project supported by National Natural Science Foundation of China(No. 69676024)

Tang Lei male, was born in 1972, PhD candidate. His research topic is HW/SW co-design.
Wei Shaojun male, was born in 1958, professor. His research interests include ULSI design methodology, telecommunication ASIC design.
Qiu Yulin male, was born in 1942, professor. His research interests include high performance and low power VLSI design.

Received 31 August 2001, revised manuscript received 3 December 2001

under some special system restrictions. In one sentence, an actual trade-off between HW and SW can not be made freely according to the characteristics of system description and restriction. What is more, due to different IPs' different interface protocols, methods must be developed to solve interface synthesis problems[4,6]. So the results of half-customized design have not the most optimized costs.

Tasks scheduling of a task graphs (TG) or a CDFG is the core of design. Many IP based co-design algorithms allocated vertex tasks to IP modules with the smallest cost[1~3,5], but the inner function and scale of every IP is unchangeable. Many HLS algorithms for VLSI also reused basic operators properly to complete the whole CDFG, but the scheduling units are too small to achieve a ULSI. So the scale, function and performance selections of FUs are the core of a full-customized HW/SW co-design.

A real full-customized HW/SW co-design has been developed. In this paper, the hardware optimization technique of the full-customized HW/SW co-design is presented. First, the coarse structure of sub-graphs' matching based on HW/SW co-design is put forward. Then, efforts will be made mainly on the discussions of universal sub-graphs' combination technique and a more advanced vertexes' compression algorithm based the technique. Experiments are done with ideal results.

2 Problem formulation

The task of HW/SW co-design is:

(1) Receive a CDFG and system restrictions as inputs;

(2) Through HW/SW partitioning and system structure generating, output a design with the most optimized costs.

Usually, the varying trends of delay and area are opposite. When a design has both HW and SW, the more the HW accounts for, the bigger the area will be, but the shorter the delay is. Then, according to the system restriction type, the most optimized costs drive at one of the following 3 types:

(1) The shortest delay (T) under area restriction (S_{res}).

(2) The smallest area (S) under delay restriction (T_{res}).

(3) The smallest value (C) of dividing price (S) by performance ($1/T$).

So, the essentiality of HW/SW co-design is to make a restrictions-oriented trade-off between HW and SW.

Accoreling to some rules of sub-graphs' matching, one can divide a data flow graph (DFG) longitudinally into many similar sub-graphs (SGs), and combine them into several compound-graphs (CGs) according to their similarity, then, map every CG directly to a multifunctional hardware unit, so these units (i. e. HWs) can complete the the system's whole task under the controlling of control codes (i. e. SW). That is the coarse structure of the idea of SGs' matching based HW/SW co-design.

If only one CG was combined, and only one multifunctional hardware unit was generated, then the design is called as a mono-similarity-system (MSS), else, a multi-similarity-system (USS) will be generated. In this paper, USS will not be discussed. Three cost functions of a MSS can be defined as:

Area: $$S_{MSS} = S_{unit} + dS_{mem} + S_{others} \quad (1)$$

Delay: $$T_{MSS} = d(T_{mem} + T_{unit}) \quad (2)$$

Coordination: $$C_{MSS} = S_{MSS}T_{MSS} \quad (3)$$

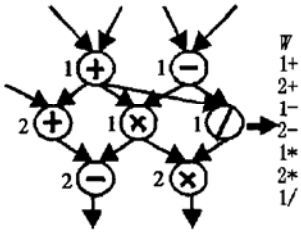
where

d = amount of SGs in DFG; S_{unit} = circuit area of CG; S_{mem} = area of every item of control codes; S_{others} = area variance of registers and multiplexers at the IO ports of CG, due to the different partition and combination of SGs, which could be calculated after CG generation with the method of living and dead periods; T_{mem} = delay of control circuits in every control step; T_{unit} = delay of CG in every control step.

3 Universal sub-graphs′ combination (USGC) technique

3.1 Associate matrix and MA algorithm

In a SG, vertexes can be divided according to their types of operation. For the same type of operation, vertexes can be sorted by their execution sequences. So, an associate matrix (AM) describing a SG can be constructed. Figure 1 shows an example, where the element $a_{i,j}$ of the matrix represents the edge starting from vertex i, and ending on vertex j in the SG.



	W	1+	2+	1−	2−	1*	2*	1/
W	0	B	L	B	0	0	0	0
1+	0	0	R	0	0	L	0	L
2+	0	0	0	0	L	0	0	0
1−	0	0	0	0	0	R	0	R
2−	L	0	0	0	0	0	0	0
1*	0	0	0	0	R	0	L	0
2*	L	0	0	0	0	0	0	0
1/	0	0	0	0	0	0	R	0

Fig. 1　A sub-graph and its AM　L: an edge to the left input of the object vertex; R: an edge to the right input; 0: nothing; $B = L + R$; W: outside of this sub-graph

When there is such a cluster of data flow SGs: for any vertex operation type, the vertex amounts are the same in all SGs, then matrix addition (MA) algorithm can be used to get the CG structure of these SGs.

MA is an algorithm of making a sum of all the SGs′ associate matrixes by adding all the elements with the same subscripts in different matrixes. The rules of element addition are:

(1) The value of an element can be 0, L, R, or B;

(2) Any element will not change its value when it is added to 0, or to the same value;

(3) In all of the other conditions, the sum is B.

3.2 USGC

USGC method has been developed and completed. When there is a cluster of arbitrary SGs, USGC method includes 2 steps.

3.2.1 Unifying all the SG vertex scales

In a cluster of arbitrary SGs, the vertex amounts of some types of operation in different SGs may be varied. But if the vertex amount of any type in any SG can be expended to its maximal amount used in all these SGs, all the SGs′ vertex scale of any type of operation could be unified. In fact, the vertex scale of any type of operation for SG could be expended to its maximal amount used in all these SGs in the following way. When some expended vertex is not really in this SG, just isolate it in the SG, i. e., and suppose that there is no edge connecting this vertex with other vertexes, or with the outside of this SG.

3.2.2 Combining SGs

After unifying all the SG vertex scale of any type of operation, AM algorithm can be used to get the CG structure as well.

Figure 2 shows the CDFG of 3rd order gray-markel ladder filter in the left and SGs partition, combination in the right. The following addition shows these SGs′ AM algorithm. The CG is structured from the result matrix, i. e., the associate matrix of CG.

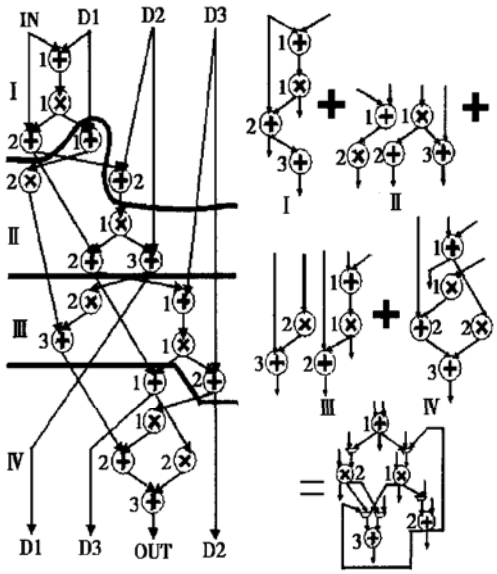


Fig. 2　Third order gray-markel ladder filter

$$\begin{bmatrix} & W & + & + & + & * & * \\ W & 0 & b & l & r & l & 0 \\ + & 0 & 0 & 0 & 0 & r & 0 \\ + & l & 0 & 0 & l & 0 & 0 \\ + & l & 0 & 0 & 0 & 0 & 0 \\ * & 0 & 0 & r & 0 & 0 & 0 \\ * & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & b & l & r & b & l \\ 0 & 0 & 0 & 0 & 0 & r \\ l & 0 & 0 & 0 & 0 & 0 \\ l & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & r & l & 0 & 0 \\ l & 0 & 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & b & l & r & l & b \\ 0 & 0 & 0 & 0 & r & 0 \\ l & 0 & 0 & 0 & 0 & 0 \\ l & 0 & 0 & 0 & 0 & 0 \\ l & 0 & r & 0 & 0 & 0 \\ 0 & 0 & 0 & l & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & b & b & 0 & l & l \\ l & 0 & 0 & 0 & 0 & r \\ 0 & 0 & 0 & l & r & 0 \\ l & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & r & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & b & b & r & b & b \\ l & 0 & 0 & 0 & r & r \\ l & 0 & 0 & l & r & 0 \\ l & 0 & 0 & 0 & 0 & 0 \\ l & 0 & r & l & 0 & 0 \\ l & 0 & 0 & b & 0 & 0 \end{bmatrix}$$

According to the above method,

$$S_{\text{unit}} = LS_{\text{e}} + \sum_{i=1}^{a} (U_{\max,i} S_{\text{op},i}) + XS_{\text{mux}} \quad (4)$$

$$T_{\text{unit}} = T_{\max,\text{path}} \quad (5)$$

where

S_{e} = circuit area of every edge; a = amount of vertex operation types in all SGs; $S_{\text{op,i}}$ = area of the ith type vertex; S_{mux} = area of every multiplexer; L, X = amount of edges and multiplexers inside CG calculated by MA algorithm; $U_{\max,i}$ = maximal amount of the ith type vertexes used in all these SGs, i. e., the amount of the ith type vertexes in CG; $T_{\max,\text{path}}$ = maximal delay in all SGs.

4 Vertexes′ compression algorithm based combination technique

4.1 Resource synthesis

Resource synthesis, or resource combination, is to combine different operations into a compound-operator (CO). The basis of combination is that the operations for combining have some similarity so that they can share the common circuits, and the total area could be decreased greatly with little increase of delay. In fact, any two operations have internal similarity to some extent.

Table 1 shows several modules designed by us. Every CO is designed with area and delay as small as possible. Table 1 indicates that the more the operations are combined into one CO, the higher the area compression efficiency is.

4.2 Vertexes′ compression algorithm

USGC method is to combine SGs based on graphs similarity, while vertexes′ compression algorithm (VCA) is to further recombine the SGs after USGC based on different vertex types inner similarity.

In a CG combined with USGC, the vertex amount of every type equals to its maximal-usage-degree in all SGs. So, in any vertex type, there must be several vertexes with large serial numbers in idle state during the most working time of CG. On the other hand, when these low efficient vertexes are busy in some SGs, there may be other types′ vertexes with equal amount of idle in the same SGs. VCA tries to delete all these low efficient vertexes and to move their tasks to other types′ idled vertexes in those SGs. Because the other types′ idled vertexes may also be busy in the rest SGs, they must be converted into COs via resource synthesis.

Table 1 A module library

	Single-operations					Edge	Mem
	+	−	*	/	SIG		
s	1	1	10	10	26.5	0.2	0.1
t	1	1	5	5	15.5	0.2	1.5

	Compound-operations					
	+ &−	* &/	* &−	SIG&/	* &/&−	SIG&* &/
s	1.1	12	10	28.5	12	28.7
t	1.1	* 5.2 /6.7	* 5.2 − 1.2	SIG 15.7 /6.7	* 5.2 /6.7 − 1.2	SIG 15.7 * 5.2 /6.7

(SIG: A module calculates sinθ or cosθ, it receives θ as one input, receives function selection as another.)

In practice, VCA tries to reduce every type′s vertex amount of CG one by one from its maximal-usage-degree (MUD) to secondly-maximal-usage-degree (SMUD) in all the SGs. The detail of VCA is shown in Fig. 3, where $U_{i,j}$ ($U_{i,j}$ = 0 is permitted) = usage-degree (UD) of ith type vertex in the jth SG; $PU_{i,j}$ = ith type vertex′s pseudo-usage-degree

(PUD) in the jth SG; $U_{smax,i}$ = SMUD of ith type vertex. If some kth type vertex only has two usage-degrees in all SGs—— a maximal-usage-degree $U_{max,k}$ and a minimal-usage-degree $U_{min,k}$, then set $U_{smax,k} = (U_{max,k} + U_{min,k})/2$. If some kth type vertex only has one usage-degree in all SGs, then these SGs are entirely congruent on this vertex type, and the VCA is not needed here.

```
for (int i= 1; i< a+ 1; i+ + ) {
  for (int j= 1; j< d+ 1; j+ + )
    PU[i][j]= U[i][j];
}
for (int k= 1; k< a+ 1; k+ + ) {
  w[k]= Umax[k]- Usmax[k];
  find all those f[k] SGs that possess Umax[k] k^th type
vertexes;
  int i= 1;
  while (Umax[k]> Usmax[k] && i< a+ 1) {
    ascertain the value of LUmax[i][k];
    /* LUmax[i][k] is the i^th type vertex's local-maximal-
PUD in those f[k] SGs * /
    v[i]= Usmax[i]- LUmax[i][k];
    if (v[i]> 0) {
      if (w[k]< = v[i]) {
        convert w[k] i^th type vertexes into w[k] i & k
compound-operators;
        remove w[k] k^th type vertexes;
        Umax[k]= Usmax[k];
        add w[k] to the f[k] SGs' i^th type vertex's UDs;
        LUmax[i][k]= LUmax[i][k]+ w[k]; }
      else {
        convert v[i] i^th type vertexes into v[i] i & k com-
pound-operators;
        remove v[i] kth type vertexes;
        w[k]= w[k]- v[i];
        Umax[k]= Umax[k]- v[i];
        add v[i] to the f[k] SGs i^th type vertex's UDs;
        LUmax[i][k]= LUmax[i][k]+ v[i]; }
    set the i type vertex's PUD of f[k] SGs with LUmax
[i][k];
    }
    i+ + ;
  }
}
```

Fig. 3 Vertexes' compression algorithm

4.3 CG's generation by VCA

After VCA, CG's all vertexes are regenerated, and in any SG, the every vertex's location in the CG has been refixed, so edges' combination method is just the same as USGC. Thus CG's all edges are also regenerated. What is more, an edge compression algorithm (ECA) has also been developed, which is not included in this paper.

According to the above method,

$$S_{unit} = L_v S_e + \sum_{i=1}^{a} (U_{v,i} S_{op,i}) + \sum_{i=a+1}^{a+b} (U_{v,i} S_{cop,i}) + X_v S_{mux} \quad (6)$$

$$T_{unit} = T_{max,path} \quad (7)$$

where L_v, X_v, $U_{v,i}$ = amount of edges, interior multiplexers and the ith type vertexes of CG after VCA; b = amount of COs; $S_{cop,i}$ = area of the ith CO type; $T_{max,path}$ = maximal delay in all SGs after VCA, which is longer than $T_{max,path}$ due to the structural change of some vertexes.

4.4 Characteristics of VCA

(1) VCA can greatly reduce the amount of total vertexes in CG, especially when some types' vertex UDs in SGs are distributed very anomaly. Moreover, after VCA, all the vertexes of CG can be redivided according to both the operation types and the amounts of compound operations, so VCA could be rerun, based on the existing results, if the first compression effect is not yet very patent.

(2) VCA requires that the total vertex amounts in different SGs are equal coarsely, thus all types' vertex MUDs can be evenly distributed in different SGs. For extreme example, if a SG has all types' vertex MUDs, VCA is no use. But no other way can be used here. In fact, all the vertexes of this SG are also all the vertexes of CG.

(3) Area compression effect is decided not only by VCA's vertex amount compression effect, but also by different operations' resource synthesis efficiency.

5 Combination efficiency

The combination efficiency of above two techniques can be calculated as:

$$EFF = S_{CDFG}/(dS_{unit}) \quad (8)$$

where

S_{CDFG} = total area of CDFG when the whole system is designed with pure HW.

EFF has a range from 0 to 1, and the bigger the EFF is, the more efficient the combination will be.

EFF is also influenced by SGs′ partitioning method from CDFG. A universal SGs′ partitioning (USGP) method has already been developed by us, but it is not included in this paper.

6 Experimental results

6.1 Experiment of VCA

VCA has been completed with C++ Builder 5.5. In order to test the validity of VCA, a test-data-generator (TDG) was also designed. When the amount of SGs, the amount of operation types and the rough vertex amount of every SG are given to TDG, it will randomly output every SG′s every type′s vertex amount to the main program of VCA for test. An example with 5 SGs and 5 operations (A→E) is shown in Table 2, the figures in Table 2 represent vertex amounts. The compression process and result are shown in Fig. 4 and Table 3. The each pane in Fig. 4 represents a vertex, and the capital in each pane represents the vertex type.

Table 2 An example with 5 SGs and 5 types of operation

		Operation					Sum
		A	B	C	D	E	
UD	SG_1	1	3	5	8	10	27
	SG_2	2	4	8	5	6	25
	SG_3	2	2	8	9	5	26
	SG_4	4	3	2	9	7	25
	SG_5	5	5	4	5	6	25
MUD		5@SG_5	5@SG_5	8@$SG_{2,3}$	9@$SG_{3,4}$	10@SG_1	37
SMUD		4	4	5	8	7	28

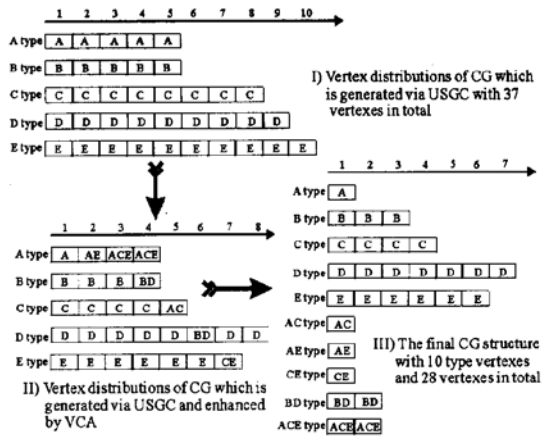


Fig. 4 Number and type distributions of all CG vertexes

Table 3 VCA results of the 5 SGs

		UD					New
		SG_1	SG_2	SG_3	SG_4	SG_5	MUD
Vertex type	A	1	1	1	1	1	1@5SGs
	B	3	3	2	3	3	3@4SGs
	C	4	4	4	2	4	4@4SGs
	D	7	5	7	7	5	7@3SGs
	E	6	6	5	6	6	6@4SGs
	AC	1	1	1	0	1	1@4SGs
	AE	1	1	1	1	1	1@5SGs
	CE	1	1	1	1	0	1@4SGs
	BD	1	1	2	2	2	2@3SGs
	ACE	2	2	2	2	2	2@5SGs
Sum		27	25	26	25	25	28

Table 3 indicates the following results.

(1) VCA is entirely correct.

(2) VCA is very efficient on compressing vertex amounts. Here, every type′s vertex amount has been reduced to its old SMUD. After vertex′s redivision according to the new operation types, almost every SG′s every type vertex′s amount has reached its new MUD. And the final CG′s vertex amount is almost equal to SG_1.

(3) Because some operation may undertake the tasks of two or more other operation types separately in different SGs, some COs may include three or more types of operation. As mentioned in 4.1, multi-operation combination is entire reality and should be encouraged.

6.2 Experiment of VCA enhanced MSS design flow

MSS design flow has been completed and several practical experiments have been done with ideal results. For example, Figure 5 shows the CDFG of Fresnel transition function. Design library is shown in Table 1. When system restriction S_{res} = 145, partitioning process is shown in Fig. 5, the main results are shown in Table 4.

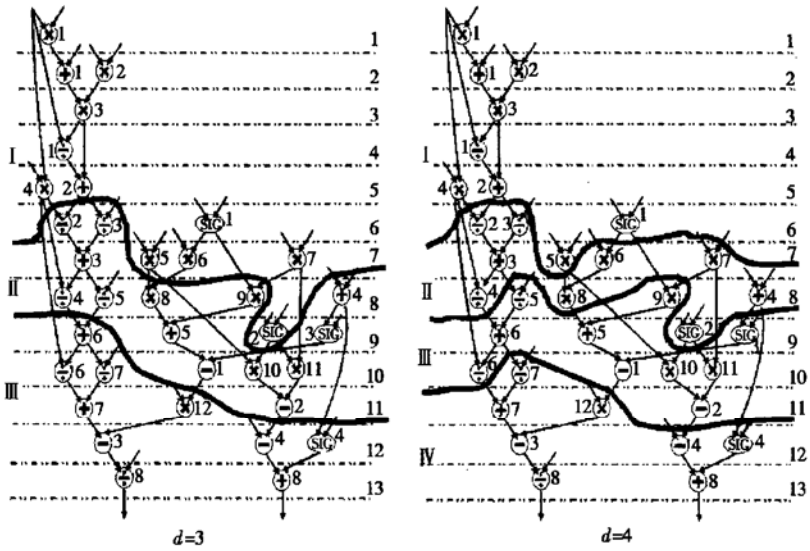


Fig. 5 CDFG of the Fresnel transition function

Table 4 Partitioning results of Fresnel transition function

d		T	S	C	EFF
3	USGC	67.8	179.1	12143	0.63
	VCA	68.4	144.3	9870	0.77
4	USGC	78.8	121.3	9558	0.70
	VCA	78.8	98.1	7730	0.85

Table 4 indicates following characteristics.

(1) When using USGC, the most optimized design is the partition at d = 4 with S_{MSS} = 121.3 and T_{MSS} = 78.8; but when using VCA, the most optimized design is the partition at d = 3 with S_{MSS} = 144.3 and T_{MSS} = 68.4.

(2) The values of EFF show that both USGC and VCA techniques have good combination efficiency, but VCA does have better efficiency than USGC.

(3) With the same partition results, VCA combination method will get smaller area than USGC, but get almost the similar delay as USGC. So VCA combination method will sure to get smaller coordination cost than USGC.

Hardware descriptions of those designs have been written in VHDL and were tested with Model Sim-99.

7 Conclusion

In this paper, two hardware combination techniques of our full-customized HW/SW co-design are presented. In brief, with USGC, the vertex amount of every type of operation in the CG equals to its MUD, while VCA tries to reduce every type's vertex amount of CG one by one from MUD to SMUD. All the experiments and result incontrovertibly verify our formulas' validity and our all methods' superiority.

References

[1] Eles Petru, Peng Zebo. System synthesis with VHDL. Kluwer

Academic Publishers, 1998: 63

[2] Liu Huiqun, Wong D F. Integrated partitioning and scheduling for HW/SW co-design. Proceedings of Codes/CASH'98, 1998: 609

[3] Bianco Luc, Auguin Michel, Gogniat Guy, et al. A path analysis based partitioning for time constrained embedded systems. Proceedings of Codes/CASH'98, 1998: 85

[4] Gogniat Guy, Auguin Michel, Bianco Luc, et al. Communication synthesis and HW/SW integration for embedded system design. Proceedings of Codes/CASH'98, 1998: 49

[5] Saha D, Mitra R S, Basu Anupam. Hardware software partitioning using genetic algorithm. Proceedings of the 10th International Conference on VLSI Design, 1997: 155

[6] Pop Paul, Eles Petru, Peng Zebo. Scheduling with optimized communication for time-triggered embedded systems. Proceedings of Codes/CASH'99, 1999: 178

全定制软/硬件协同设计中的硬件优化技术*

汤 磊[1] 魏少军[2] 仇玉林[1]

(1 中国科学院微电子中心, 北京 100029)

(2 清华大学微电子所, 北京 100084)

摘要: 提出了基于单相似系统生成的软/硬件协同设计中的硬件优化技术. 介绍了一种基于子图匹配软/硬件协同设计技术的大致框架, 引进通用子图群合并算法, 并着重讨论了基于节点压缩优化技术的高效子图群合并算法. 实验结果很好地证明了所有上述理论的正确性以及算法的有效性.

关键词: 软/硬件协同设计; CDFG; 单相似系统; 子图; 复合图; 合并; 使用度; 代价

EEACC: 2570

中图分类号: TN47 **文献标识码**: A **文章编号**: 0253-4177(2002)06-0637-08

* 国家自然科学基金资助项目(批准号: 69676024)

汤 磊 男, 1972 年出生, 博士研究生, 主要研究方向为 ULSI 设计方法学中的软/硬件协同设计技术.

魏少军 男, 1958 年出生, 教授, 主要研究领域包括 ULSI 设计方法学、通讯专用集成电路设计等.

仇玉林 男, 1942 年出生, 研究员, 主要研究领域为高性能、低功耗集成电路设计.

2001-08-31 收到, 2001-12-03 定稿