Journal of Theoretical and Applied Information Technology
31st August 2014. Vol. 66 No.3
© 2005 - 2014 JATIT & LLS. All rights reserved·

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

HARDWARE-SOFTWARE PARTITIONING ALGORITHM BASED ON BINARY SEARCH TREES AND GENETIC ALGORITHM TO OPTIMIZE LOGIC AREA FOR SOPC

[1]SONIA DIMASSI, [2]MEHDI JEMAI, [3] BOURAOUI OUNI, [4]ABDELLATIF MTIBAA

Laboratory of Electronic and microelectronic,

University of Monastir, Monastir 5000, TUNISIA

E-mail: [1]sdimassi@yahoo.com, [2]jmehdie@gmail.com, [3]ouni_bouraoui@yahoo.fr, [4]abdellatif.mtibaa@enim.rnu.tn

ABSTRACT

This paper presents an approach based on hardware/software partitioning to minimize the logic area of System on a Programmable Chip (SOPC) while respecting a time constraint. Our contribution focuses on introducing a new hardware/software partitioning algorithm. This algorithm is based on the principle of Binary Search Trees (BST) and genetic algorithms. It aims to define the tasks that will run on the Hardware (HW) part and those that will run on the Software (SW) part. The proposed algorithm will determine the best partition that will reduce the number of tasks used by the HW and increase the number of tasks used by the SW and thereafter the area will be reduced. The results show that our algorithm significantly reduces the logic area compared to other well known algorithms.

Keywords: *Logic area, Hardware/software partitioning algorithm, Binary search trees, Genetic algorithms, SOPC.*

1. INTRODUCTION

Using a System on a Programmable Chip (SOPC) is increasingly common in embedded applications. A SOCP is a circuit comprising multiple functions such as one or more processors, one or more reconfigurable areas, a signal processor DSP (Digital Signal Processor), various peripherals and memory or analog parts. These circuits are increasingly used because of their small size and reduced costs compared to the use of various circuits for performing the same function. Therefore, many hardware and software techniques must be developed to satisfy specific constraints in terms of area, performance, power consumption, etc.

The term "Co-design" appeared in the early 1990s to mark a new way of thinking about the design of integrated circuits and systems. The co-design of software and hardware became necessary to meet the requirements of the embedded systems' market. Indeed, the emergence of multimedia systems (mobile phones, game consoles, etc.) resulted in a greater complexity of the electronics and economic competition requires a shorter design time. Many research teams have addressed the problem of hardware/software partitioning [1], [2], [3], [4] and [5]. Nevertheless, several specific tools that are related to platform simulation (or emulation) showed a genuine interest to help the designer in the design stage. Automation would guide the designer to decide the partitioning. Indeed, the partitioning problem is extremely difficult and depends on technological parameters (speed, consumption, etc.), application (architecture), economic parameters (cost of design and manufacturing) and "sociological" parameters (security, maintainability, testability, etc.).

Thus, in this paper, we present an effective approach based on hardware/software partitioning, Binary Search Trees (BST) and genetic algorithms to implement a data flow graph on SOPC circuit while minimizing the logic area. In this paper, we have implemented the hardware tasks of the graph in the Left Sub Tree (LST) of our binary search tree and the software tasks in the Right Sub-Tree (RST). However, the implementation of the hardware modules may degrade the design in terms of area. The main objective of our hardware/software partitioning approach is to balance all the design parameters to find a better trade-off between the logic area of the application and its execution time.

This paper is structured following six parts. After the introduction, we give an overview of the related works; in the third section, we present the hardware/software partitioning model. The fourth section shows the problem formulation and our suggested algorithm. In the fifth part, we present the experiments and their results. Finally, we end up with a conclusion.

2. RELATED WORKS

Cutting or partitioning hardware/software is an important phase of the system design. It consists of seeking the best compromise hardware/software and then deciding whether the implementation of the different parts of the system will be hardware or software. In general, the software is used to reduce the cost of the design and the hardware is used to increase the performance. Many techniques and algorithms have been proposed to assist the designer in this task. The ultimate goal is to automate this task.

Also, the designers were moving towards a mixed approach to design a system at a reasonable cost while meeting the performance imposed. A portion was performed with programmable components, it is the software part. The other part was conducted with specific hardware components in the application, it is the hardware part. The combined use of software and hardware resources required to design new methods to find the best trade-off between software and hardware parts (software / hardware partitioning) and enable them to design simultaneously.

In the design system, an optimization method generally consists of applying an optimization algorithm on the set of sub-functions of the specification. A partitioning algorithm is an optimization algorithm that seeks to minimize or maximize one or more criteria, such as the area, the execution time, consumption etc. In fact, an optimization algorithm can overcome the problems of estimating these criteria for one or more target architectures and to seek one or more optimized realizations for a given problem. [6] has introduced such heuristics and [7] conducted a comparison of several minimization algorithms implemented in the hardware/software partitioning.

In the literature, the problem addressed by the software/hardware partitioning, was meant to reduce the overall cost of the implementation in terms of hardware resources and improving performance in terms of execution time. Indeed, as opposed to hardware, the implementation of a software module requires more flexibility and less cost, but more execution time.

The exploration of the design space usually requires a partitioning step. That it is manual or automatic; its purpose is to spread the "functions" of the application on the software and hardware parts of the target architecture. This process is repeated until a solution or a set of solutions has been found satisfactory. The partitioning problem is very complex (NP-complete) and many approaches have been developed. In this context, we find the exact algorithms that are based on the Integer Linear Programming (ILP) [8], [9] uses the PACE partitioning [10] based on a dynamic programming algorithm [11] and branch and bound in [12]. The disadvantage of these algorithms is that they are very slow and can only be applied to small graphs. So, to solve these problems, researchers have tried heuristic algorithms that are more flexible and effective as the network throughput [13], simulated annealing [14] and [15], Tabu search [16], genetic algorithm [17], combined algorithm [18] and greedy algorithm [19]. We note also that there are many methods that rely on scheduling algorithms [20], [21], [22], [23] are combined with steps of selecting components.

Among the tools and methods of partitioning, we mention the following:

- COSYMA [15] is an environment for the exploration of the process of co-synthesis.
- Lycos [9] is a co-design environment that allows the exploration of the design space systems composed of a microprocessor and a hardware accelerator.
- SpecSyn [24], [25] is an environment of co-design, which is before the hardware / software synthesis. The heart of the methodology is the paradigm "SER" (Specify-Explore Refine).
- POLISHED [26] is a co-design environment that starts from the system specification and goes down to logic synthesis and the synthesis software.
- PICO (Program In, Chip Out) [27] is a co-design environment to generate systems compound of a VLIW or EPIC processor dedicated to the target application, a hierarchy of cache memories and non-programmable accelerator (systolic deviation).

As mentioned earlier, these optimization algorithms seek one or more achievements optimized for a given problem. In this paper, we have suggested an algorithm for hardware/software partitioning based on a binary search tree and a genetic algorithm that minimizes the logic area of the SOPC circuit.

Journal of Theoretical and Applied Information Technology
31st August 2014. Vol. 66 No.3
© 2005 - 2014 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3. HARDWARE/SOFTWARE PARTITIONING MODEL

The HW/SW partitioning model, defined in this section, considers the following characteristics: granularity, metrics associated with the functional blocks, computational model, representation of the solution and the cost function.

The behavioral description given in a high-level language is transformed into a data flow graph. Each node $v_i \in V$ corresponds to a part of the application that itself belongs to the base granularity (a single instruction or a basic block). Hence, a data flow graph G (V; E) is a directed to the acyclic graph describing the dependencies between the operations of an application. Where $V = \{v_1, v_2, \dots, v_n\}$ is the set of nodes, n is the number of nodes and E is the set of edges $\{e_{ij} | 1 \leq i, j \leq n\}$.

Once the system is represented under this model, values for the metrics are associated to each node v_i. The following metrics are used: software latency ($L_S(v_i)$), occupied hardware area in slice ($A(v_i)$), and the hardware latency ($L_H(v_i)$).

In this model, a partitioning solution is expressed as an indicator vector Xm that is defined as follows: Xm=Xm(i); Where: $i \in [1,n]$ and Xm (i) = 1, if node (i) will be implemented in hardware; however, Xm (i) = 0, if the node (i) will be implemented in software.

Hence, our optimization problem can be modeled as follows:

$$\begin{cases} minimize \sum_{v_i \in G} X_m(i) \mathrm{A}(v_i) & \textbf{(1)} \\ subject\ to\ \mathrm{L(G)} \leq \mathrm{T} & \textbf{(2)} \end{cases}$$

Where :-T is the temporal constraint

-L(G) is the whole latency of the graph G

4. PROBLEM FORMULATION AND PROPOSED ALGORITHM

The trees are mainly the data structure used to store ordered data and according to Knuth they are the largest non-linear structure involved in the computer science. They are widely used in all fields, because they are well adapted to the natural representation of organized and homogeneous information, and they have a great speed and a handling convenience. We find this structure in all computing areas, whether for the example of compilation (syntax trees to represent expressions or possible productions of language), imaging (quaternary trees), algorithmic (for example it is the support of sorting methods or management information in tables), or in the fields of artificial intelligence (game trees, decision trees, resolution trees etc).

The method of storage and retrieval of information by a binary tree is well known to programmers and is frequently used. They are also interesting because they optimize the access time to information. Our purpose behind using binary search trees is to reduce our search space and to have an optimized data access time. Recall, first of all, the bulk of this method. In computer science, a BST, sometimes also called an ordered or sorted binary tree, is a node-based binary tree data structure where each node has a comparable key (and an associated value) and satisfies the restriction that the key in any node is larger than the keys in all nodes in that node's left sub-tree and smaller than the keys in all nodes in that node's right sub-tree. Each node has no more than two child nodes. Each child must either be a leaf node or the root of another binary search tree. The left sub-tree contains only nodes with keys less than the parent node; the right sub-tree contains only nodes with keys greater than the parent node. The main advantage of binary search trees is that it remains ordered, which provides a quicker search time than many other data structures. The common properties of binary search trees are as follows [28]:

- The left sub-tree of a node contains only nodes with keys less than the node's key.
- The right sub-tree of a node contains only nodes with keys greater than the node's key.
- The left and right sub-tree each must also be a binary search tree.
- Each node can have up to two successor nodes.
- There must be no duplicate nodes.
- A unique path exists from the root to every other node.

To achieve our partitioning algorithm HW/SW, we relied on the principle of Binary Search Trees BST, which aims to reduce the search space. Our idea was to build a binary search tree whose root is a virtual node that we have defined as the average of the larger and the smaller size of a module. By definition of a BST, the left sub-tree will contain modules with a small size and the right sub-tree will contain those with large sizes. In this way, we will have a HW/SW partitioning with tasks, that will run on the HW part, were in the left sub tree and the ones, that will run on the SW part, were in the right sub-tree. The question that arises is if this partitioning HW/SW is the optimal one according to our time constraint or not. Two cases may arise: the first case in which the application execution time of the realized partitioning is less than our time constraint, so in this case, we will find what are the HW tasks that we can migrate to the SW part and vice versa in the second

case where the application execution time exceeds the time constraint. In this way, we have reduced our search space, in fact, instead of performing a search in a whole binary tree; we search in the left or right

sub-tree as appropriate. This investigation tasks in question will be carried out according to the principle of genetic algorithms. The pseudocode of our proposed algorithm is shown in figure 1.

5. EXPERIMENTS AND RESULTS

To confirm our approach, we have implemented the 16-DCT task graph on FPGA Xilinx Virtex®-5. The Xilinx Virtex®5 development kit enables a high performance embedded design in Xilinx FPGAs.

```
1: Begin
2: Initialize the number of the generation size and the temporal constraint $T_{const}$;
3: Build the Binary Search Tree (BST)
4: Assign the Left Sub-Tree (LST) to the hardware part and the Right Sub-Tree (RST) to the Software part of
-: architecture;
5: Calculate the execution time $T_{ex}$;
6: If ($T_{ex} \leq T_{const}$) then
7:    Initialize the first generation $P_0$ with the individuals of the LST;
8:    Calculate the fitness of each individual in $P_0$;
9:    Copy the individual with the smallest fitness to the solution;
10:   while (termination conditions) do
11:      Select two individuals ($g_1$, $g_2$) from the current generation;
12:      Perform crossover on ($g_1$, $g_2$) to produce two new individuals ($g_{c1}$, $g_{c2}$);
13:      If (min {fitness ($g_{c1}$), fitness ($g_{c2}$)} $\leq$ min {fitness ($g_1$), fitness ($g_2$)}) then
14:         If ($T_{exgc} \leq T_{const}$) then // where the $T_{exgc}$ is the execution time of the crossover individuals //
15:            Accept the crossover;
16:         else
17:            Reject the crossover with $g_{c1} = g_1$, $g_{c2} = g_2$;
18:         end if
19:      else
20:         Reject the crossover with $g_{c1} = g_1$, $g_{c2} = g_2$;
21:      end if
22:      Perform mutation on $g_{c1}$ to produce $g_{m1}$;
23:      If (min {fitness ($g_{m1}$)} $\leq$ min {fitness ($g_{c1}$)}) then
24:         If ($T_{exgm} \leq T_{const}$) then // where the $T_{exgm}$ is the execution time of the mutation individuals //
25:            Accept the mutation;
26:         else
27:            Reject the mutation with $g_{m1} = g_{c1}$;
28:         end if
29:      else
30:         Reject the mutation with $g_{m1} = g_{c1}$;
31:      end if
32:      Perform step 22-31 on $g_{c2}$ to produce $g_{m2}$;
33:      Calculate the fitness of each individual in current generation;
34:      If (the smallest fitness of the current generation $\leq$ fitness (solution)) then
35:         Copy the individual with the smallest fitness to the solution;
36:      end if
37:      Increase the generation number;
38:   end while
39:   Return solution x[i] with i $\in$ [1, n]; // with n is the number of the LST nodes//
40: else
41:   Perform step 7-39 to produce solution x[i] with i $\in$ [1, n]; // with n is the number of the RST nodes//
42: end if
43: Return the final solution of the Hardware-software partitioning;
44: End
```

Figure 1: Pseudo-code

In our approach, the software resource is the PowerPC and the hardware resources are configurable logic blocs (CLBs). Hence, to compute the parameters of each node and to access to the PowerPC, we have used Xilinx ISE tool and Xilinx EDK tool. These Xilinx design tools provide resources and timing report incorporates timing delay and resources to provide a comprehensive area and timing summary of the design. Our algorithm has been written in JAVA language and executed under Windows-7 on Acer-PC (Intel Core 2 Duo T5500; 1.66 GHz; 1GB of RAM). In order to demonstrate the effectiveness of the proposed algorithm, we compare it to Tabu, simulated annealing and genetic algorithm. The simulation results are presented in table 1.

Table 1: Design Results

Algorithm	Run time (ms)	Latency (ns)	Area (Slice)
Proposed algorithm	766	2664	2202
Tabu algorithm	58281	2704	2757
simulated annealing algorithm	843	3012	2214
Genetic algorithm	40375	2928	2274

To evaluate the design results shown in table 1, we have introduced the following metric **β**

$$\beta = \frac{L}{A_{max} - A_L} \qquad (3)$$

A$_{max}$: all nodes of the graph are implemented to the hardware part of the architecture.

A$_L$: the logic area consumed by the graph

L: the whole latency of the graph

Therefore, based on the above equation, a partitioning algorithm is classified to be good if it decreases the value of **β**.

Table 2: Design Results

	Proposed algorithm	Tabu algorithm	simulated annealing algorithm	Genetic algorithm
β	1.153	1.541	1.311	1.308

Based on the above design results shown in table 2, we show that our algorithm is the best one in terms of **β** value. Indeed, our algorithm provides a gain reaching 25.2% compared to Tabu algorithm, of 12.05 % compared to simulated annealing algorithm and of 11.85 % compared to the Genetic algorithm.

6. CONCLUSION

Many methods and algorithms have addressed the problem of hardware/software partitioning; they have emerged in the late 90s without providing satisfactory solutions. The evolution of design, the characteristics of the components and the complexity of applications and architectures are certainly responsible for the ineffectiveness of the solutions in this field. In this context, we have proposed a hardware/software partitioning algorithm based on binary search trees and genetic algorithms to determine the best partition that minimizes the logic area. Compared to Tabu, simulated annealing and genetic algorithm, our proposed algorithm has provided the better design results in term of the logic area.

REFRENCES:

[1] Bouraoui Ouni, Ramzi Ayadi and Abdellatif Mtibaa, "Combining Temporal Partitioning and Temporal Placement Techniques for Communication Cost Improvement" *Advances in Engineering Software, Elsevier Publishers*, Volume 42, Issue 7, July 2011, pp : 444-451.

[2] Ouni, B. , Ayadi, R., Mtibaa, A. "Temporal partitioning of data flow graph for dynamically reconfigurable architecture", *Journal of Systems Architecture,* Volume 57, Issue 8, September 2011, Pages 790-798

[3] Bouraoui OUNI, Abdellatif MTIBAA, "Optimal placement of modules on partially reconfigurable device for reconfiguration time improvement", *Microelectronics International published by Emerald Group Publishing Limited*, volume 29, Issue 2, 2012, Pages 101-107.

[4] Ramzi Ayadi, Bouraoui Ouni and Abdellatif Mtibaa, "A Partitioning Methodology that Optimizes the Communication Cost for Recongurable Computing Systems" *International Journal of Automation and Computing (IJAC), Institute of Automation and Springer-Verlag Publishers*, Volume 9, N° 3, June 2012, Pages 280-287.

[5] Mehdi Jemai, Sonia Dimassi, Bouraoui Ouni and Abdellatif Mtibaa, "Optimization of logic area for System on Programmable Chip based on hardware-software partitioning", *International Conference on Embedded Systems and Applications (ICESA) Hammamet-Tunisia*,March 2014.

[6] SIARRY P., Application des métaheuristiques d’optimisation en électronique, [RE 8], traité

recherche,Techniques de l'Ingénieur, 2002.

[7] LOPEZ-VALLEJO M., LOPEZ J.C., "On the Hardware-Software Partitioning Problem: System Modeling and Partitioning Techniques", *ACM Transactions On Design Automation of Electronic Systems (TODAES)*, 2003, vol. 8, n° 3, pp. 269-297.

[8] S. Banerjee, E. Bozorgzadeh, and N. D. Dutt, "Integration physical constraints in hw-sw partitioning for architectures with partial dynamic reconfiguration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 11, 2006, pp. 1189 -1202.

[9] J. Madsen and al. Lycos," the lyngby co-synthesis system", *Design Automation for Embedded Systems*, 2(2), March 1997.

[10] P. V. Knudsen and J. Madsen. Pace, "A dynamic programming algorithm for hardware/software partitioning", *In International Symposium on Hardware/Software Codesign (CODES)*, Pittsburgh, USA, March 1996.

[11] J. Wu and T. Srikanthan, "Low-complex dynamic programming algorithm for hardware/software partitioning," *Information processing letters*, vol. 98, no. 2, 2006, pp. 41-46.

[12] K. Chatha and R. Vemuri, "Hardware-software partitioning and pipelined scheduling of transformative applications", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, no. 3, 2002, pp. 193-208.

[13] H. Liu and D. F. Wong, "Efficient network flow based multiway partitioning with area and pin constraints", *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 17, no. 1, 1998, pp. 50–59.

[14] Z. Peng and K. Kuchcinski, "An Algorithm for Partitioning of Application Specific Systems," *Proc. European Conf. Design Automation (EDAC'93)*, 1993, pp.316-321.

[15] A. Österling, Th. Benner, R. Ernst, D. Herrmann, Th. Scholz, and W. Ye. "Hardware/Software Co-Design : Principles and Practice, chapter The COSYMA", *System. Kluwer Academic Publisher*, 1997.

[16] T. Wiangtong, P.Y.K. Cheung, and W. Luk, "Comparing Three Heuristic Search Methods for Functional Partitioning in Hardwaresoftware Codesign," *Design Automation for Embedded Systems*, vol. 6, no. 4, 2002, pp. 425-449.

[17] Hong jun He; Qiang Dou and Weixia Xu , "Hardware/Software Partitioning for Heterogeneous Multicore SoC Using Genetic Algorithm", *IEEE Intelligent System Design and Engineering Application (ISDEA),* 2012, pp 1267-1270.

[18] Yu Jiang, Hehua Zhang, Xun Jiao, Xiaoyu Song, William N. N. Hung, Ming Gu, and Jiaguang Sun," Uncertain Model and Algorithm for Hardware/Software Partitioning", *IEEE Computer Society Annual Symposium on VLSI*, 2012.

[19] K.S. Chatha and R. Vemuri, "Magellan: Multiway Hardware- Software Partitioning and Scheduling for Latency Minimization of Hierarchical Control-Dataflow Task Graphs," *Proc. Ninth. Hardware/Software Codesign (CODES '01)*, 2001, pp. 42-47.

[20] T. Grandpierre, C. Lavarenne, and Y. Sorel. "Optimized rapid prototyping for real-time embedded heterogeneous multiprocessors", *In International Symposium on Hardware/Software Codesign (CODES)*, Roma, Italy, May 1999.

[21] L. Bianco, M. Auguin, G. Gogniat, and A. Pegatoquet, "A path based partionning algorithm for time constrained embedded systems design", *In International Symposium on Hardware/Software Codesign (CODES),* Seattle,USA, March 1998.

[22] R. Szymanek and K. Kuchcinski. "Design space exploration in system level synthesis under memory constraints", *In Euromicro conference*, Milano, Italy, March 1999.

[23] P. Eles, K. Kuchcinski, Z. Peng, A. Doboli, and P. Pop, "Scheduling of conditional process graphs for the synthesis of embedded systems". *In Design Automation and Test in Europe Conference (DATE)*, Paris, France, February 1998.

[24] D. D. Gajski, F. Vahid, S. Narayan, and J. Gong, "System-level exploration with specsyn", *In ACM/IEEE Design Automation Conference (DAC)*, San Francisco, USA, 1998.

[25] D.D. Gajski, F. Vahid, S. Narayan, and J. Gong. Specsyn, "An environment supporting the specify-explore-refine paradigm for hardware/software system design", *IEEE Transaction on VLSI Systems*, 6(1) :84-100, March 1998.

[26] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, and B. Tabbara, "Hardware-Software Co-Design of Embedded Systems : The Polis Approach. Kluwer" *Academic Publisher*, June 1997.

[27] B. R. Rau and M. Schlansker, "Embedded computer architecture and automation". *IEEE Computer*, 2001, pages 75-83.
[28] Gilberg, R.; Forouzan, B. (2001), "8", Data Structures: A Pseudocode Approach With C++, Pacific Grove, CA: Brooks/Cole, p. 339, ISBN 0-534-95216-X