BEHAVIORAL SYNTHESIS OF ASYNCHRONOUS SYSTEMS : A METHODOLOGY

Joseph Okito DEDOU - Daniel CHILLET - Olivier SENTIEYS

LASTI - ENSSAT - Université de Rennes I,
6 rue de Kérampont,
BP 447, 22305 Lannion, France,
Tel: +33 2-96-46-50-30; fax: +33 2-96-46-66-75
e-mail: dedou@enssat.fr

ABSTRACT

In asynchronous system, initiation and completion of operations are events that can occur at any instant and the execution time is data dependent. Thus if an asynchronous timing model is considered, we can provide scheduling and resource allocation methods which will have a significant impact on the performance and the area of the final implementation. Until now the different methods proposed for High-Level Synthesis (HLS) do not apply to the above topics. This paper proposes a methodology for the scheduling of asynchronous systems. It is based on the average delay.

1. INTRODUCTION

For a few years, there has been a revival of interest in asynchronous system design. This is due to the fact that it has been presented as an alternative to synchronous systems. They can be loosely viewed as systems with no global clock. In the aspect of the consumption, the main advantage of these circuits is the elimination of all the useless consumption in synchronous circuits. Notably, it provides the elimination of the power consumption of the clock buffers and of the logic gate commutations that do not contain useful information. This consumption is approximatively 30 % of the total consumption of the circuit. A comparative study between synchronous logic and asynchronous techniques in term of consumption and area have been realized by Kim and Sridhar [1]. They have shown that the use of asynchronous techniques allows us to improve the consumption of a factor of 2.5 to 10 even with an increase of the area in the same proportions.

In "the previous work" section, we will see that different methods have been proposed to design asynchronous systems. Unfortunately, these methods do not deal with High-Level Synthesis (HLS) issues such as scheduling, resource-allocation etc. Indeed, with no clock-controlled time step, the scheduling problem in an asynchronous system can not be viewed as a partitioning of operations into steps as it has been defined for the synchronous systems in [2].

In an asynchronous system, operations have delays which are data-dependent and time is considered as a continuous variable. Therefore, we have to consider a new timing model to find a scheduling strategy. Since one of their main features is to exhibit average computation time, it will be an excellent opportunity to use it as a timing model.

For this purpose, it is necessary to build a library of components including parameters such as the average computation time or the delay as a function of the inputs or even a probability distribution of the delay. Thus, a statistical methodology to estimate the average computation time of operators such as adder, substractor or multiplier have been proposed in a previous paper [3]. As can be seen in Table 1, the estimated values (ignoring the control delay) of this method are similar to the results given by the probabilistic methods proposed in [4, 5]. The aim of this paper is to propose a methodology for behavioral synthesis of asynchronous systems. It is organized as follows. Section 2 reviews related works on the asynchronous systems design. Section 3 presents our scheduling strategy based on average computation time for High-Level Synthesis of Asynchronous Systems.

Nbr of bits	16	32
$T_{average}$ probabilistic (ns)	4.92	6.191
$T_{average}$ statistic (ns)	5.176	6.331
relative error %	5.20	2.26

Table 1: average delays of ripple carry adder: statistic vs probabilistic method

2. PREVIOUS WORKS

Recent works in asynchronous synthesis can be roughly classified into two categories. The first approach is analogous to the logic synthesis in the synchronous systems terminology. These methods are based on the manipulation of formal specifications such as signal transition graphs (STG) and Petri nets. In [6], [7] and [8] several algorithms have been proposed for the synthesis of asynchronous circuits from behavior description using signal transition graphs (STG). Indeed, an STG is a form of interpreted Petri nets where the transitions in the nets are used as transitions of signals in the control circuits. Several CAD tools based on this specification have been made available. The most well known is PETRIFY. It is used for manipulating concurrent specifications and asynchronous controllers synthesis [9].

The second category focuses on the synthesis of asynchronous systems by the interconnection of pre-defined asynchronous modules. These methods attempt to compile behavioral descriptions in a high-level language like CSP and deriving a structural netlist in terms of asynchronous blocks [10]. In [11], an integrated design environment called SHILPA, for the specification, simulation, analysis and synthesis of self-timed asynchronous circuits

0-7803-5471-0/99/$10.00©1999 IEEE

has been presented. Others methods proposed to use TANGRAM (language for concurrent systems specification) for the behavioral specification [12].
Unfortunately, these methods do not deal with High-Level Synthesis issues such as scheduling and resource allocation. To our knowledge, despite the two algorithms presented in [13], there is a lack of research in this area.

3. ARCHITECTURAL SYNTHESIS : A METHODOLOGY

In a general manner, the goal of the architectural synthesis is to generate automatically a structural description of an architecture from a behavioral specification of an application in a high level language. First, the specification is compiled into a Data Flow Graph (DFG) that represents the operations and the dependencies between the data. Then, it is divided into four main phases: selection, allocation, scheduling and binding.

- **Selection :** its goal is to select from a complete library of operators, the components whose characteristics satisfy the application, time and cost constraints.
- **Allocation :** it consists in determining the exact quantity of operators and the intervals of time where they are usable. So to satisfy the imposed constraints, it has to take into account temporal and spatial parallelism. It enables the handling of the hardware components to implement [14].
- **Scheduling :** it is defined as the allocation of a date of execution to each operation in the data flow graph.
- **Binding :** it consists in allocating each operation to an operator. This task has also the responsibility of the allocation of data to registers or memory banks.

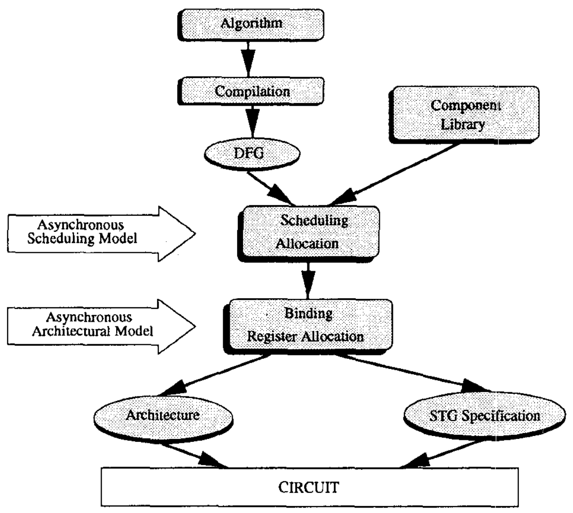

Figure 1: Our Design Flow for Asynchronous High-Level Synthesis

The aim of our research, is to propose a complete method for high-level synthesis of asynchronous systems (see figure 1). Therefore, we must have another definition for several classical concepts in HLS such as :

- Operators timing model ;
- Asynchronous scheduling ;
- Asynchronous architecture model (binding).
- Interface with logic synthesis

In this paper, we put the emphasis on scheduling and binding concepts. Thus, at the architectural level, we take an asynchronous model of the circuit architecture. For the algorithmic level, we decided to have a natural specification like for the synchronous systems since there is no necessity for an asynchronous concept at this level.

3.1. Scheduling

As we have mentioned, scheduling is the allocation of a date of execution to each operation of the application. In the synchronous systems, this execution date corresponds with the control step, and in the majority of cases, it depends on the slowest operation (worst-case delay). Therefore, asynchronous systems present two significant differences. Indeed, the time is a continuous variable, the operation beginning and its completion are events that can occur at any instant. Moreover, the execution delay of the operations are data dependent. Consequently, in asynchronous systems, with no global clock, the scheduling problem can not be viewed as the partitioning of operations in discreet time, but rather as the definition of a partial order of operation execution.

Thus the problem that we will seek (asynchronous scheduling) to solve can be defined as follows :

From a library of components including their average computation delays, for a data flow graph (DFG) and a cost constraint, define a partial order of the different operations execution in order to minimize the average delay estimation of the global system represented by the DFG so as to obtain an optimal behavior.

Thus, at the beginning, the algorithm is compiled into an acyclic graph G(N, A) where N represents the set of nodes and A the set of arcs.

Each node N_i of N represents an operation realized by the application, and an arc a_{ij} from N_i to N_j, denotes that N_j succeeds N_i in the DFG. For each node, we are going to determine its $Start_{time}$ and its $Completion_{time}$.

- $Start_{time}$: It is defined as the minimal date from which an operation can be executed.
- $Completion_{time}$: It is defined as the completion date of a given operation. An estimation of this $Completion_{time}$ is given by the following expression.

$$\mathbf{Completion_{time}} = \mathbf{Start_{time}} + \delta_{\mathbf{moy}}$$

where δ_{moy} represents the average delay of the operation realized by the corresponding node.

Thus, if a node N_i precedes a node N_j then :

$$\mathbf{Start_{time}(N_j) = MAX_{N_i \in P_{N_j}}\,(Completion_{time}(N_j))}$$

We denote :
S_{N_i}: the set of nodes successors of N_i.
P_{N_i}: the set of nodes predecessors of N_i.

Firstly, we are going to define two rules of priority which will be used to elaborate the strategy aiming to determine the partial order of execution of the DFG.

Priority 1 :

The goal is to determine the order in which, "ready" nodes ,i.e whose predecessors have been scheduled, have to be scheduled. For this purpose, it is necessary to evaluate the length of the DFG critical path L_{cp}. Then we define for each node N_i its depth in the graph $L(N_i)$ with the following expression:

$$\mathbf{L(N_i) = L_{cp} - L_s(N_i)}$$

where $L_s(N_i)$ represents the length of the path formed by N_i's successors i.e S_{N_i}.

Thus the priority is given to the node that will have the smallest depth $L(N_i)$.

N.B : *If a node N_i belongs to several paths, $L(N_i)$ will be determined according to the longest path among the former.*

Priority 2 :

Its goal is to allow the choice between two nodes using the same operators and which have the same priority according to priority 1. In this case, the priority is granted to the node that will have the smallest $Start_{time}$.

The proposed algorithm is described below :

1. Calculate the critical path length L_{cc}.
2. Calculate the depth L_{N_i} of each node.
3. For each operation type, establish an ordered list of ready node (i.e those whose predecessors have been scheduled) according to the rules of priority defined above.
4. Schedule each node of each list: from the $Start_{time}$ of the node concerned, find an available operator during a period at least equal to the average delay of the operation represented by this node. In other words, find an interval of duration greater than or equal to δ_{moy} whose minimal bound would be at least equal to the $Start_{time}$ of the node.
5. Determine the ready nodes and return to step 3.

N.B *If the minimal bound found differs from $Start_{time}$, then $Start_{time}$ will take this value. So it is necessary to evaluate a new $Start_{time}$ for the successor nodes.*

As an example we take a Finite Impulse Response Filter which equation is given below :

$$Yn = \sum_{i=1}^{n} x_{i-1} * h_{n-i}$$

The ordering and allocation result is given in figure2. In asynchronous systems, the operation control is not centralized and their delays are not fixed. In order to exploit this potential for optimization, we will define two notions. The idea behind these definitions, is to increase the speed of the systems :

- **Fixed sequence :** two operations are in fixed sequence, if they are linked by a precedence constraint.
- **Variable sequence :** two operations are in variable sequence if they have no precedence constraint.The control for a variable sequence is similar to a First In First Out.

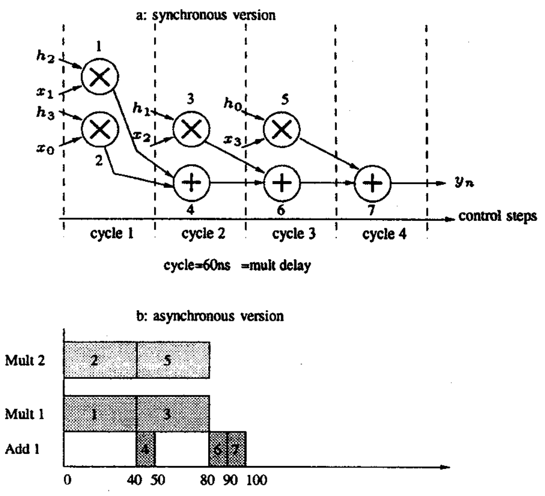

Figure 2: Asynchronous vs Synchronous scheduling

Consequently, the execution order of two operations which are in fixed sequence will be that given by the scheduling task. In the case of a variable sequence, the execution order depend on the data processed by those operations. In this way, if an operation can be allocated to several operators, then his execution will be done by the operator which is available at the moment where his data are valid. The advantage of having a variable sequence is to optimize the use of the different operators and the increasing of the circuit speed.

3.2. Binding

Binding consists of allocating each operation to an operator. This task is also in charge of the allocation of data to registers or memory banks. For this task, we will use a similar approach to synchronous systems. But, in comparison to the synchronous binding (static), asynchronous systems give the possibility to have a dynamic binding. This is possible since the operation delays depend on the data processed. To achieve this, we define the resource mobility as the number of the same operator. Therefore we say that :

- if the resource of the same type mobility is greater than one, we will have operations with dynamic binding (see figure 4.a).

- if the resource mobility is equal to one, we will have static binding (see figure 4.b).

NB : *In figure 4, selectors are used to select the operation which must be executed and the distributors for the operator which will execute the operation selected and also the specified output. Controllers served to ensure the different types of sequences (control)*

To illustrate that, we take the example of the FIR-4. In this example (see figure 2), **operation 3** (N_3) is executed by *multiplier 1.* Supposing that, the delay needed for **operation 1** (N_1) for a given data is greater than the average delay of the multiplier and the one needed for **operation 2** (N_2) is smaller than δ_{moy} (NB : delay are data dependent). In this case *the multiplier 2* will be available before *multiplier 1*. Since **operations 1, 2, 3 and 5** (N_1, N_2, N_3 and N_5) are in variable sequence, i.e there are no precedence constraint between them therefore, **operation N_3** can be executed by the *multiplier 1*, as well as the *multiplier 2*. In this particular case,

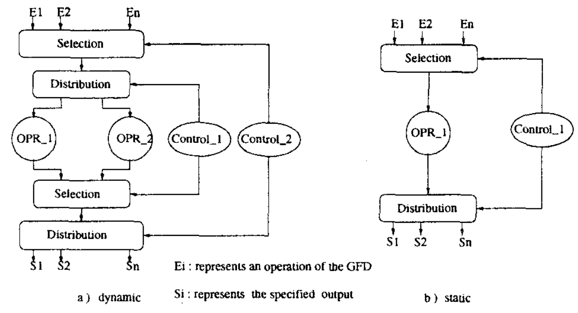

Figure 3: Static and dynamic binding

the best choice is to execute this operation by the multiplier 2 in order to increase the speed of the circuit. This can be realized by using the architecture presented in figure 4. Selection and distribution are used, the former to select the best operation to execute, and the latter to select the correct output.

In the example proposed, the estimated average computation time given by the asynchronous scheduling is about 100 ns. In comparison to the result given by synchronous scheduling (figure 2), the gain in term of speed is about 40%. However asynchronous components lead to an increased area in the same proportion. To reach an equivalent speed, synchronous systems will use more components.

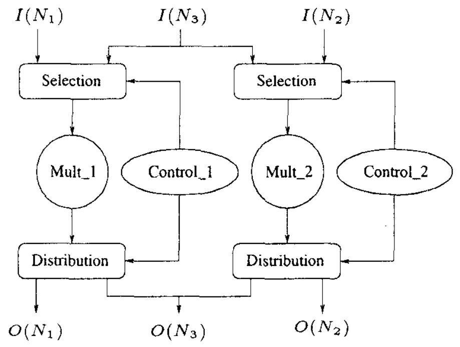

Figure 4: Model for dynamic binding : FIR-4 example

4. CONCLUSION

In this paper we have presented a method for scheduling and binding asynchronous systems based on the average delay of the operators. This method is integrated in the **BSS framework** (Breizh Synthesis System http://archi.enssat.fr/bss). In comparison to the synchronous method, where the allocation of operations to operator is static, we can have dynamic allocation. However, the drawback is the increasing of the interconnection complexity. Therefore, if we want to overcome this problem, we must introduce a new constraint at the interconnection level. We have also discussed about an asynchronous binding allowing us to generate an architecture using dynamic or static scheduling. This work is in progress by the automatic generation of a VHDL model for the simulation of the complete architecture and also an interface with the PETRIFY [9] synthesis tools.

5. REFERENCES

[1] S. Kim and R. Sridhar. Comparison of Power Consumption amoung Asynchronous Design Styles with their Synchronous Counterpartsaeuss con. *IEEE*, pages 7–10, 1995.

[2] D. Gajski et al. *High-Level Synthesis - introduction to Chip and Systems Design*. kluwer Academic Publishers, 1992.

[3] O. J. Dedou, D. Chillet, and O. Sentieys. Asynchronous Timing Model for High Level Synthesis for DSP Applications. In *SIGNAL PROCESSING IX: Theories and applications*, volume 1, pages 475–478. EUSIPCO-98, September 1998.

[4] Alessandro De Gloria and Mauro Olivieri. Statistical carry lookahead adders. *IEEE Trans. on Computers*, 45(3):340–347, March 1996.

[5] V. Varsharvsky, V. Marakhovsky, and M. Tsukisaka. Data controlled delays in the asynchronous design. In *In Proc. International Symposium on Circuits and systems*, volume 4, pages 153–155, May 1996.

[6] T.A Chu. Synthesis of Self-Timed VLSI Circuits from Graph-theoritic Specifications. In *Proc. international Conf. Computer Design*, pages 220–223. IEEE Computer Society Press, 1987.

[7] L. Lavagno, K. Keutzer, and A.S. Vincentelli. Algorithms for synthesis of hazard-free asynchronous circuits. In *28th ACM/IEEE Design Automation Conference*, pages 302–308, 1991.

[8] K.J. Lin and C.S. Lin. A realization algorithm of asynchronous circuits from STG. *IEEE*, pages 322–326, 1992.

[9] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transaction on information and systems*, E80-D(3):315–325, March 1997.

[10] A.J.Martin. Programming in vlsi: From communicating process to delay-insentive circuits. Caltech-cs-tr-89-1, Departement of Computer Science, California Institut of Technologie, 1989.

[11] V. Akella and G Gopalakrishnan. SHILPA: A High-Level Synthesis System for Self-Timed Circuits. In *Int. Conf. on Computer-Aided Design*, pages 587–591. IEEE Society Press, November 1992.

[12] C van Berkel, J. Kessel, M. Roncken, R. Saeijs, and F. Schalij. The vlsi-programming langage tangram and its translating into handshake circuits. pages 384–389, 1991.

[13] R.M. Badia and J. Cortadella. High-Level Synthesis of Asynchronous Degital Circuits: Scheduling Strategies. Technical report, Architectural of computer departement/ Catalunya Polytechnic University, November 1992.

[14] O. Sentieys, J.Ph. Diguet, J.L. Philippe, and E. Martin. Hardware module selection for real time pipeline architectures using probabilistic cost estimation. In *IEEE ASIC conference*, Rochester, USA, September 1996.