ASYNCHRONOUS TIMING MODEL FOR HIGH-LEVEL SYNTHESIS OF DSP APPLICATIONS

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 an asynchronous system, initiation and completion of operations are events that can occur at any instant and the operations have delays which are data dependent. Thus if an asynchronous timing model is considered, we can provide scheduling, resource-allocation strategy. Since one of the principal feature of the asynchronous systems is to exhibit average computation time, it will be interesting to use it as a timing model. In this paper we present a statistical approach to derive the average computation time of asynchronous components, first step toward High Level Synthesis. This method allows to reduce the critical path which in the case of a real-time application will be an excellent issue to reduce the number of operators.

1 INTRODUCTION

Since 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 the synchronous systems. They can be loosely viewed as systems with no global clock. Thus by eliminating the global clock, asynchronous systems avoid the problem of clock-skew and would provide faster and less power consuming solution [1].
In the previous work, we will see that the different methods proposed to design asynchronous systems do not deal with High-level synthesis issues such as scheduling, resource-allocation etc. which will have a significant impact on the performance and area on the final implementation. With no clock-controlled time step, i.e., the scheduling problem in an asynchronous system can not be viewed as a partitioning of operations into steps as in synchronous systems [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 the principal feature of the asynchronous system is to exhibit average computation time, it will be interesting to use it as a timing model.
For this purpose, it is necessary to built a library of components including parameters such as average computation time or delay as function of the inputs or even a probability distribution of the delay. This can provide an improvement of the simulators.
This paper presents a statistical methodology to estimate the average computation time of operators such as adder, substractor or multiplier. The paper is organized as follows. Section 2 reviews related work on the asynchronous systems design. Section 3 presents our methodology based on statistical assumptions to derive the average computation time of asynchronous components. Results and conclusions are presented in section 4.

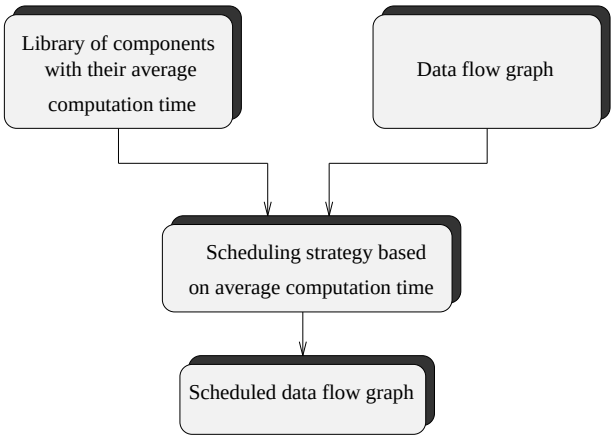

Figure 1: Our design flow for scheduling strategy

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 [3], [4], [5] several algorithms have been proposed for the synthesis of asynchronous circuits from behavior description using signal transition graph (STG). An STG is a form of interpreted Petri nets where the transitions in the nets are inter-

preted as transition of signals in the control circuits.

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 [6]. In [7], an integrated design environment called SHILPA for the specification, simulation, analysis and synthesis of self-timed asynchronous circuits has been presented. Others methods proposed to use TANGRAM (language for concurrent systems specification) for the behavioral specification [8].
Unfortunately, these methods do not deal with High-level synthesis issues such as scheduling, resource-allocation. To our knowledge, despite the two algorithms presented in [9], there is a lack of research in the area of asynchronous-system behavioral synthesis.

As discussed above, our main goal is to provide a method which deals with the problem of scheduling, and resource-allocation by considering an asynchronous timing model as shown in figure 1. In this order of idea, we have to define a method to calculate the average computation time of the different components of the library. Until now, the research to derive the average computation time of an operator has been mainly focused on the ripple carry adder [10]. It is due to the fact that the computation time of such an adder depends on the critical carry propagation chain. It is known that the sum of two bits can produce a valid carry value independently on the preceding carry if either the two bits are both 1 or both 0. Thus, assuming statistical distribution of the operand, the probability for the sum of two bits to produce an anticipate (generate) carry is $p = \frac{1}{2}$, [11]. Therefore, we have to determine the longest critical chain of the carry propagation path to find the addition time of two operands. This idea has been used in [12] for the estimation of the energy consumption average in a ripple carry adder.
By considering two numbers of N bits, we define a vector of N-1 bits which is obtained in the following way. If the carry C_i is known in advance then put the value 1 at P_i in the vector else put the value 0. Finally, the average addition time for a ripple carry adder is obtained by using the following expression.

$$E = 1 + \frac{1}{2^{N-1}} * \sum_{i=0}^{j} U(i) \qquad (1)$$

where $j = 2^{N-1} - 1$ and $U(i)$ is the length of the longest string of 0s in the expression of P. More details can be found in [10],[13].

3 CONTRIBUTION

As it can be seen, the method shown above is restricted to the ripple carry adder. Here, we present a statistical methodology to derive the average computation time of the asynchronous components. The idea is to provide a method which can be applied to all the components of the library. Notice that, those components have been described in the dual-rail technique, in which a value x is encoded as pair (x.t, x.f) [14] (see table 2). In addition to this, a third code word is used to represent the invalid value. Thus , the completion signal is valid when the output is valid, and invalid when the output is invalid (figure 2). They have been written in RTL VHDL and synthesized using the AVANT! CAD tools (formerly COMPASS).

N bits	16	32	64	96
E-1	3.24	4.29	5.31	5.9
Average (ns)	4.92	6.191	7.43	8.14

Table 1: Results obtained with probabilistic method for a ripple carry adder

Value x	encoded value
0	(0,1)
1	(1,0)
invalid	(0,0)

Table 2: Dual-rail encoded

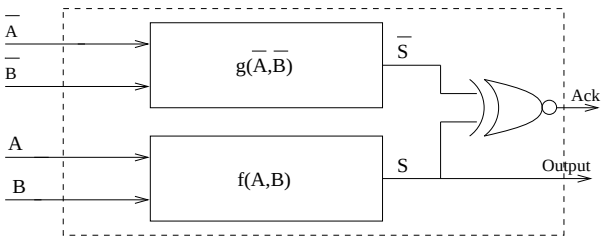

Figure 2: Architectural model of the component

In the case of an adder, $\overline{S}$ can be obtained simply using the relation: $\overline{S} = \overline{A} + \overline{B} + 1$. Since we have: $-A = \overline{A} + 1$.
The goal of the methodology presented, is to derive the average computation time of an operator with a certain confidence level by simulating the operator with a serie of samples since it is data dependent. This technique has been already used in [15] to estimate power dissipation in operators synthesized by Logic Synthesis tool. Statistical methodology has been used also in [16] for power estimation in sequential circuits.

To ensure the correctness of the results, each simulation is done with files containing input vectors chosen at random. First of all, we have to choose the number

of simulation n, *i.e* the number of observations, which will guarantee the satisfaction of the good value of the average. According to the Central Theorem, for a normal law, a good approximation of the average can be obtained if the number of samples and the number of simulations are both great enough [17]. In practice, this theorem can be applied when the number of simulations exceeds thirty.

In our case, the number of observations is less than thirty. Therefore we have to use **the Student distribution** [17]. Assuming that the average computation time obeys to the reduced central normal law, we can derive the theoretical average computation time by using the average given by the different samples. We show in figure 3 that for a ripple carry adder, the convergence of the average delay is quickly obtained for sample of 1000 data. The result is similar in comparison to the theoretical average delay which can be calculate with the formula 1 (see table 2). More accuracy can be obtained on the average estimation by deriving an interval which will contain the theoretical average computation time with a certain confidence level equal to 100*(1-α)%. Under the Student distribution, the bounds of this interval can be found by using the following expression:

$$\frac{|T - T_{moy}|}{T_{moy}} < \frac{t_{\alpha/2} S_T}{T_{moy}\sqrt{N}}$$

where:

- T_{moy} represents the average of the sample.
- N the number of simulation done.
- $t_{\alpha/2}$ is a t distribution with $N-1$ degree of freedom.
- S_T the variance calculated with the following equation.

$$S_T^2 = \frac{\sum_{i=1}^{N} (T_i - T_{moy})^2}{N-1}$$

First of all, we apply the method to a simple ripple carry adder without taking into account the completion signal. Table 3 presents the results of the average of computation time for the ripple carry adder (for 16 and 32 bits) described in ES2 0.7μm CMOS technology. The column labeled *Error* shows the relative error made to determine the interval which will contain the theoretical estimation with regard to the estimation given by the method for the average. In comparison to the result of the probabilistic method (table 2), there are a certain convergence. In the case of a simple multiplier, the average computation time is about 39.16 ns instead of 62.32 ns which is the worst-case delay.

The main contribution of this methodology is that, it can be applied not only to the ripple carry adder but also to all the components contained in our library. As proposed in [18], we can decompose the computation

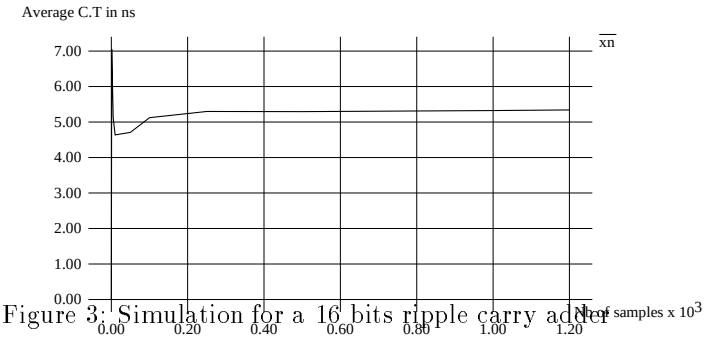

Figure 3: Simulation for a 16 bits ripple carry adder

time of an asynchronous component with the following expression where β is a function depending on the data ($\beta < 1$, in the case of synchronous component $\beta = 1$), and T_{sync} a constant.

$$T_{ct} = \beta * T_{worst-case} + T_{sync} \qquad (2)$$

If we consider the equation 2, we can demonstrate that the computation time for an asynchronous component is better than his synchronous counterpart. In term of speed, asynchronous systems is a good opportunities for real-time systems. In table 4 we present the results obtained for an asynchronous adder described in VHDL synthesizer with the AVANT! CAD tool using a sample of 2000 data and also the different bounds obtained for different levels of confidence, in other words the interval which will contain the theoretical average computation time for the component. Figure 4 shows that the computation time of asynchronous component can be adjusted to a known probability law.

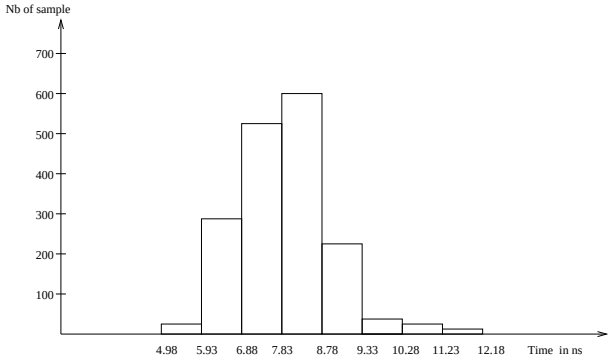

Figure 4: Computation time histogram

4 CONCLUSIONS

In this paper, we have presented a statistical methodology to derive the average computation time of asynchronous components. For a fully asynchronous version of a ripple carry adder (see figure 2), with a 16 bits

N=16 bits

average (ns)	variance	standard deviation	T_{cc} (ns)
5.108	0.014	0.122	18.5

1 - α	min (ns)	max (ns)	error %
90 %	4.991	5.224	4.56
95 %	4.956	5.260	5.95
99 %	4.856	5.360	9.86

N=32 bits

average (ns)	variance	standard deviation	T_{cc}
6.247	$9.94.10^{-4}$	0.031	33.98

1 - α	min (ns)	max (ns)	error %
90 %	6.217	6.277	0.96
95 %	6.208	6.286	1.25
99 %	6.182	6.312	2.07

Table 3: Average computation time results for RCA (16 and 32 bit) with statistical method

format, the result is approximatively about 8.6 ns. In comparison to a synchronous version where the critical path is equal to 18.5 ns (for a ES2 0.7μm CMOS technology), we have a considerable improvement. Considering for instance, regular asynchronous Digital Signal Processing algorithms (e.g. adaptive filtering, FFT, speech coder, etc.), the gain in term of speed can go up to 46 %, in comparison with a synchronous version of the same algorithms. However, asynchronous components lead to an increased area in the same proportion, but, to reach an equivalent speed, the synchronous version will use more components.

This is an important step toward high level synthesis for asynchronous systems.

simulation	$T_{average}(ns)$
1	8.679
2	8.729
3	8.64
4	8.701
5	8.687
6	8.693
average	8.657
variance	$5.09.10^{-3}$
standard deviation	$7.13.10^{-2}$

1 - α	min (ns)	max (ns)	error %
90 %	8.595	8.719	1.43
95 %	8.576	8.738	1.869
99 %	8.523	8.792	3.1

Table 4: Average computation time for an asynchronous adder

References

[1] S. Hauck. Asynchronous design methodologies: An overview. *Proceeding of the IEEE*, 83(1):69–93, January 1995.

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

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

[4] 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.

[5] K.J. Lin and C.S. Lin. A realization algorithm of asynchronous circuits from STG. In *EDAC-ETC-EUROASIC*, pages 322–326. IEEE Computer Society Press, 1992.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] Luis A. Montalvo, Keshab K. Pahri, and Janardham H. Estimation of average energy consumption of ripple carry adder based on average length carry chains. *VLSI Signal Processing*, 9:189–198, 1996.

[13] Bachar El Hassan. *Architecture VLSI asynchrone utilisant la logique differentielle pr charge: Application aux op rateurs arith mtiques*. PhD thesis, I.N.P Grenoble, 1995.

[14] Christian.D Nielsen. Evaluation of Function Block Designs. ID-TR 135, Dept of Computer Science: Tecnical University of Denmark, 1994.

[15] S. Gailhard, O. Sentieys, N. Julien, and E. Martin. Area/Time/Power space exploration in module selection for DSP high level synthesis. In *PATMOS'97, Louvain-la-Neuve, 8-10 September 1997, 35-44*, 1997.

[16] I. N Najj Najm, S Goel. Power estimation in sequential circuits. In *DAC*, pages 623–627, 1995.

[17] A.A. Sveshnikov, editor. *Problems in Probability Theory, Mathematical Statistics and Theory of Random Functions*. Dover publications, inc. New York.

[18] Mark A. Franklin and Tienyo Pan. Performance comparison of asynchronous adders. *IEEE*, pages 117–125, 1994.