22

Behavioral synthesis : control schemes in question

B. Rouzeyre, D. Dupont
L.I.R.M.M., UMR CNRS C9928, Université Montpellier II
161 rue Ada, 34392 Montpellier Cedex 5, FRANCE
Tel : (33) 67148525, Fax: (33)67148500, e–mail :rouzeyre@lirmm.fr

Abstract

In many behavioral synthesis systems, a one–clock scheme is assumed for both controller and datapath implementation An alternative solution to this purely synchronous control scheme is presented here. It is based on tuning the clock period according to data–path delays in order to suppress idle times due to synchronism. This scheme is mainly attractive when datapath operations have heterogeneous delays. An implementation for such a control scheme is proposed. The silicon area overhead is very small while it allows a good performance enhancement.

Keywords

Behavioral synthesis, clocking, performance

1 INTRODUCTION

Since a decade, an emerging design technique referred as High Level Synthesis (HLS) or as behavioral synthesis has been developed in CAD industry and in many research centers. It is intended to automate the earliest steps of the ASIC's design task. It consists in generating automatically the architecture of a circuit at the register transfer level from its behavioral specification, generally given as an input algorithm (see Mc Farland 1988 or IEEE 1990) for a general presentation). Generically, the architecture consists of a data–path and of a controller in charge of sequencing the instructions of the input algorithm on the synthesized data–path.

The two main tasks of data path synthesis are scheduling and allocations. The performance of the future architecture is fixed mainly during the scheduling phase of the synthesis: the operations of the data flow graph (DFG) are apportioned among control steps (c–steps) that is, among states of the control machine, according to data dependencies and to design constraints. (e.g. Ku 1990, Paulin 1989, Paulin 1990, Cloutier 1990, Camposano 1991). The allocation phase consists in mapping the variables and operations of the input algorithm onto a minimal amount of hardware.

The underlying control scheme adopted for synthesis strongly influences the circuit performance as well as the scheduling process itself.

Generally a "synchronous" clocking scheme is taken : a Finite State Machine (F.S.M) implements the sequencing of the input algorithm and drives the data–path signals through combina-

G. Saucier et al. (eds.), *Logic and Architecture Synthesis*

tional logic. Depending whether there are data dependent transitions in the behavioral description or not, the lower bound T of the clock period is :

$T=T_{FSM} + T_{comb} + T_{datapath}$ or $T=Max\ (T_{FSM} + T_{comb}, T_{datapath})$.

T_{FSM} is the internal delay of the FSM for delivering the next state, T_{comb} is the delay of the control logic. $T_{datapath}$ is the time requested for completion of data transfers from register to register through functional units (f.u.s). Due to the synchronism of the controller, $T_{datapath}$ must be taken equal to the longest delay in the data–path, i.e. :

$$T_{datapath} = \underset{s \in \{c\_steps\}}{Max}\ d(s) \qquad \text{with } d(s) = \underset{\text{data–transfers during } s}{Max}\ T_{data–transfer}$$

where $T_{data–transfer}$ denotes the time requested for a data transfer from registers to register through operations.

Thus, in both cases, the period of the controller is fixed by the *slowest* operation (i.e. the slowest data–transfer on the data–path) over all c–steps. Consequently, unnecessary slack time intervals are added to the other c–steps. This may result in poor performance if the delays of operations are heterogeneous. For instance, if a ($\times$: 80 ns) is performed in a c–step, 50 ns are wasted at every c–step involving only (+ : 30 ns). Even for the same type of operation, this heterogeneity can be due to the f.u. type implementing the operations or simply due to the data bitwidth involved. For instance, in experiments given in Chaiyakul 1992, on the same example, d(s) varies from 63ns to 252ns as the data bitwidth ranges from 8 to 32 bits.

With regard to the whole performance of the circuit, the total idle time due to such a clocking scheme is:

$$T_{idle} = \sum_{s \in \{c\_steps\}} F_s \cdot \left(\underset{s}{Max}\ (d(s)) - d(s) \right)$$ where F_s denotes the number of times c–step s is activated. It can be very large in the data operations have heterogeneous delays..

Multi–cycling operations (i.e. making operations overlap c–steps) is a commonly mentioned trick for minimizing these idle times by making slow operations overlap c–steps (this can be done with pipeline or non–pipeline f.u.s). But, these idle times can not be totally removed if the delays of the operations cannot be expressed as integral multiples of a common delay. In Narayan 1992, a method for determining, prior to scheduling, the clock period that minimizes slack times is presented.

Table 1 summarizes scheduling results published in numerous papers (e.g. Hwang 91) on the benchmark filter example taken from Dewilde 1985 : multiplications are assumed to overlap two consecutive c–steps and additions are not chained.

Table 1 Scheduling results for several synthesis parameters

Adders	*Multipliers.*	*2 stage pipeline Mult.*	*# c–steps*	*# c–steps with only +*	*# c–steps with × (and +)*
1	1	–	28	8	20
2	1	–	21	5	16
2	2	–	18	10	8
3	3	–	17	9	8
1	–	1	28	17	11
2	–	1	19	8	11
3	–	1	18	7	11
3	–	2	17	9	8
2	–	2	18	11	7

T_{idle} can be obtained on the following way:
- if d(+) = 1/2 d(×), then Tidle = 0
- if d(+) < 1/2 d(×), then d(s)=1/2 d(×) and Tidle = (1/2 d(×) – d(+)) * column 5
- if d(+) > 1/2 d(×), then d(s)=d(+) and Tidle = (d(+) – 1/2 d(×)) * column 6

2 ADAPTATIVE CONTROL

2.1 Principle

For non–pipelined designs, a controller architecture allowing to adjust independently the delay of every c–step to the minimum leads to a higher speed since there are no idle times. Such a controller structure is depicted in Figure 1.

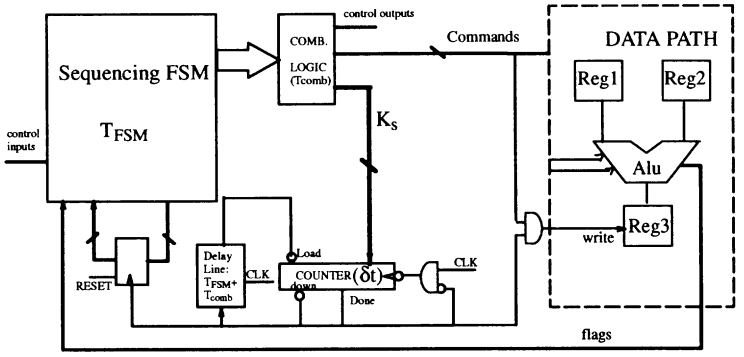

Figure 1 Controller structure for adjusted clocking

It is mainly composed of a counter in charge of computing the delay d(s) of every c–step s. It is loaded with a parameter K_s, depending on s. After K_s clock periods, it delivers a rising edge on its output which is used as the clock for the FSM. Once the outputs of the FSM and the outputs of the combinational logic are stable, i.e. after a $T_{FSM} + T_{comb}$ delay, the counter is loaded with the new value of K_s. Due to its simplicity , such a counter can have a very fast clock (typically up to 100Mhz). The K_s coefficients are taken equal to $K_s = \lceil d(s)/Tclk \rceil$ where Tclk is the clock period. The delay line on Figure 1 delivers the load signal of the counter once the combinational block outputs are stable, i.e. after $T_{FSM} + T_{comb}$.

This structure can be simplified for data path dominated designs, i.e. if $T_{FSM} + T_{comb}$ is of the same order as T_{clk}. The delay line is removed, K_s+1 instead of K_s is sent to the counter, its output signal enables directly the clock of the FSM.

Doing so, the idle time of every c–step is reduced to d(s)–K_s.Tclk, instead of being Max(d(s))–d(s) with a "synchronous" control. It must be noticed that, due to the simplicity of the added hardware, Tclk can be very short (see below) involving so a very small total idle time.

With such a clocking scheme, the execution time of the circuit is $\Sigma F_s.[T_{FSM} + T_{comb}+d(s)]$ (instead of being $\Sigma F_s.[T_{FSM} + T_{comb}+T_{datapath}]$ for synchronous control).

2.2 Implementation

Figure 2 gives the mask of such a 4–bit counter with its peripheral logic. It has been designed using the Cadence silicon compiler and the 1.2 μm ES2 module library. Its frequency is 87 Mhz while its silicon area is 0.24 mm^2. Let's recall that its frequency is bitwidth independent while its area growths as log(max Ks). On the taken behavioral description example, depending on data–transfers involved at every c–step, d(s) ranges from 15ns to 140 ns. The behavioral description contains 13 c–steps that leads to a controller implementation with $T_{FSM}+T_{COMB}$ = 40ns.

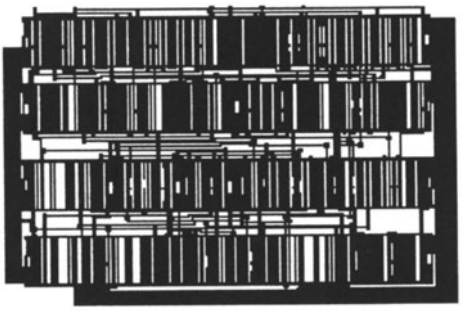

Figure 2 Layout example of a controller allowing "adjusted control"

Thus, on this example, idle times are at most 11ns per control step. Using a "synchronous" control, the clock should be set to 180ns at least. Just considering the fastest operation, the idle time would be at least equal to (140–15)*number of executions.

2.3 Considerations for high level synthesis

Such a control scheme has consequences on the synthesis process itself since the delay of control steps is no longer a constant but tuned to the delay of operations :
– the cost function to be considered during the scheduling of operations is no longer the number of c–steps but the sum of the delays of the c–steps. This makes most of the published scheduling algorithms inadequate. So we have developed a new scheduler (see Rouzeyre 1994). The results given in section 3 show that adaptative control minimizes not only idle times but also reduces the number of c–steps and thus T_{FSM}.
– the second one, related to the first one, is the importance of the choice of the functional units types for the operations of the input algorithm (module selection and module type binding) since the delay of the operations depends on this choice (and thus d(s)).
– It is clear that the profit of using such a clocking scheme versus "synchronous" control will be all the better that the operations delays are very dissimilar. Thus this control model is very adequate when the design space is very large. This is the case when there are many operations types in the behavioral description or when the synthesis system offers the possibility of implementing several operations of the same type (lets say additions) using different functional units types (lets say ripple–carry and look–ahead carry adders) with different delays in the same design (e.g. Timmer 1993).

Let us illustrate these points on the D.F.G. fragment in Figure 3.a. The different possible execution times of those three functions are given in Figure 3.b in which 0i represents a functional unit able to implement the operation. Figure 3.c depicts the scheduling solution according to synchronous control (cost function : the number of c_step). The clock period is then at least equal to 230 ns or 180 ns depending on the binding of operations to functional units (Figure 3.d, 3.e). Figure 3.f shows the scheduling solution with an adjusted clocking scheme (considering the cost function as the sum of the delay of the c_steps. The binding is given in Figure 3.g. The execution time for this fragment is now equal to (80+50)ns=130ns versus 180ns. Results given in Table 1 show that, conversely to what this example suggests, the use of an adaptative control scheme does not increase the number of c–steps). These points are addressed and detailed in Rouzeyre 1994 a.

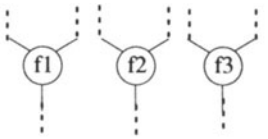

Figure 3.a Fragment of D.F.G.

time(ns)	O1	O2	O3
f1	230	70	50
f2	180	80	—
f3	—	—	70

Figure 3.b Delays and function sets are given for illustration purpose

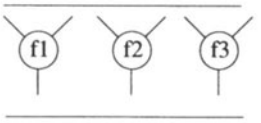

Figure 3.c Scheduling solution with the number of c_steps as cost function

	O1	O2	O3
f1	230	–	–
f2	–	80	—
f3	—	—	70

Figure 3.d **T=230ns**

	O1	O2	O3
f1	–	70	–
f2	180	–	—
f3	—	—	70

Figure 3.e **T=180ns**

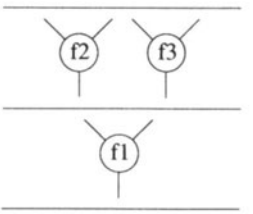

Figure 3.f Scheduling solution with the sum of the delays of the c_steps as cost function

	O1	O2	O3
f2	–	80	—
f3	—	—	70

T1=80ns

	O1	O2	O3
f1	–	–	50

T2=50ns

Figure 3.g **T=130ns**

Figure.3 (Module binding / scheduling / control scheme) impact on circuit performance.

3 RESULTS

We have applied this control scheme to the fifth order filter from (e.g. Dewilde 1987). The DFG is composed of 26 additions and 8 multiplications.

First experiment : In order to demonstrate the efficiency of this method with the usual scheduling hypothesis, we have first assumed a synchronous clocking scheme. The delay of the 40ns for the adders and 100ns for the multipliers. Mult–P are two–stage pipeline multipliers with a 2×50ns delay. Cycle time is taken at 50ns (possible chaining of additions is inhibited). The critical path length is 17 cycles. Table 1 summarizes the results. The scheduling results (number of c–steps) have been obtained from literature (e.g. Hwang1991). The fifth column gives the sum of

the slack times Tidle due to the synchronous clocking (under the assumptions on the delays) with Tidle = number of c–steps with only additions multiplied by (50–40)ns.

Second experiment : Then we have assumed an adjusted clocking scheme and the same hypothesis on f.u. delays. Results are reported in the right side of Table 1 using and ad hoc scheduling algorithms. It must be noticed, as mentioned before, that not only the total delay has been reduced but also the number of c–steps. This c–step reduction can not be obtained by using a synchronous control model during scheduling and adjusting the c–step delay afterwards. It is clear that the gain on performance evolves in the same way as the difference between the chosen clock period and the delay of additions.

Third experiment : Table 2 presents results with a richer f.u. library : adders 40 ns, multipliers 100 ns, alus (+ : 50ns, × : 110ns). The sixth column give results obtained from literature with a standard clocking. In the last column, the total delay is computed with a 55 ns clock cycle, with regard to the f.u. delays and assuming that multiplications overlap two c–steps. This shows again that "adjusted control" minimizes the total delay not only by suppressing the idle times but also by minimizing the number of c–steps.

Table 1 : experiments with synchronous control (clock period 50ns) and adjusted control

			Synchronous control			*Adjusted control*	
Adders	*Mult.*	*Mult–P*	*c–steps*	T_{idle}	*total delay*	*c–steps*	*total delay*
1	≥1	–	28	80 ns	1400 ns	27	1320 ns
2	≥2	–	18	100 ns	950 ns	17	800 ns
≥3	≥3		17	90 ns	850 ns	16	760 ns
1	–	≥1	28	170 ns	1400 ns	28	1230 ns
2	–	1	19	80 ns	950 ns	19	870 ns
≥3	–	≥2	17	90 ns	850 ns	17	760 ns
2	–	≥2	18	110 ns	900 ns	18	790 ns

Table 2

			Adjusted control		*Synhronous control*	
Adders	*Mult.*	*Alu*	*c–steps*	*total delay*	*c–steps*	*total delay*
–	–	2	19	1190 ns	23	1265 ns
1	1	1	18	1000 ns	–	–
1	–	2	17	900 ns	19	1035ns

4 CONCLUSION

We propose a control scheme for high level synthesis systems that allow to minimize the idle time when sequencing data operations. It is mainly valuable for large behavioral description when performance is addressed. This is achieved by using a counter that drives the clock signal of the controller.

The silicon area offset is constant and minor to the controller area itself while the gain on performance is all the better that the heterogeneity of data instructions is high.

5 REFERENCES

M.Mc.Farland, A.Parker, R.Camposano (1988) Tutorial on High–Level Sythesis. In Proc. of 25th Design Automation Conference, pp 330–336.

IEEE Design and Test of computers (1990). Special issue on high level synthesis. Vol.7, No.5, Oct. 1990.

D.Ku, G. De Micheli (1990) Relative scheduling under timing constraints. Proc. 27th DAC, pp. 59–64.

P.G. Paulin (1989) Scheduling and binding algorithm for high level synthesis. Proc. 26th DAC, 1989, pp. 1–6.

R. Cloutier, D. Thomas (1990) The combination of scheduling, allocation, and mapping in a single algorithm. Proc. 27th DAC, pp. 71–76.

P.G. Paulin, J.P. Knight (1989) Force–directed scheduling for the behavioiral synthesis of ASIC's. IEEE Trans. on CAD, Vol8, No6, pp. 661–679, June 1989.

R. Camposano (1991) Path–based scheduling for synthesis. Proc. IEEE Transactions on CAD. Vol.10., No.4, January 1991, pp. 85–93.

V. Chaiyakul, A. Wu, D. Gajski (1992) Timing Models for High Level Synthesis. Proc. EuroDac 92, pp. 60–64.

S. Narayan, D.Gajski (1992) System Clock Estimation based on Clock Slack Minimization. Proc. EuroDac 92, pp. 66–71.

C. Hwang, J. Lee, Y. Hsu (1991) A formal approach to the scheduling problem in high level synthesis. Proc. IEEE Transactions on CAD. Vol.10., No.4, April 1991, pp. 464–475.

P. Dewilde, E. Deprettere, R. Nouta (1987) Parallel and pipelined VLSI implementation of signal processing algorithms. In VLSI and Modern Signal Processing. S.Y.Kung,H.J.Whitehouse, T.kailath Editors. Prentice Hall. pp. 257–260.

B. Rouzeyre, D. Dupont, G. Sagnes (1994) Component Selection, Scheduling and Control Schemes for High Level Synthesis" Prop. EDAC–ETC–EUROASIC 94, pp. 482–489.

A. Timmer, M. Heijligers, L. Stok, J. Jess (1993) Module selection and selection using unrestricted libraries. Proc. Euroasic–Edac 93, pp.547–551.

C. Hwang, J. Lee, Y. Hsu (1991) Formal approach to the scheduling problem in high level synthesis. Proc. IEEE Transactions on CAD. Vol.10., No.4, April 1991, pp. 464–475.

6 BIOGRAPHY

B. Rouzeyre is Associate Professor at the the ISIM engineering school at the University of Montpellier. He got his PhD degree in 1984 in CAD. He heads the Architecture Group within the Microelectronics Department at LIRMM. Its main research interests are behavioral synthesis of ASICs and testing. He authored and coauthored more than 30 papers on this field.

D. Dupont received PhD degree in computer Aided Design in 1994 from the University of Sciences & Technology, Montpellier II, FRANCE. He have been working since 1990 with the "High Level Synthesis" Department at LIRMM.