

High-Level State Machine Specification and Synthesis

Andreas Kuehlmann

Reinaldo A. Bergamaschi

IBM Research Division

Thomas J. Watson Research Center

Yorktown Heights, N.Y., U.S.A.

Abstract

Current synthesis methodologies based on hardware-description languages focus mainly on two distinct levels: behavior and register-transfer levels. In many practical cases, however, the initial specification lies between a pure behavioral description and a completely structural one. This paper presents a method and algorithms for exploring the design space between the register-transfer and behavioral levels. The method consists of the specification of a high-level state machine, which combines the advantages of a specific control structure, by means of states and transitions, with the flexibility of behavioral descriptions inside each high-level state. High-level synthesis techniques are used for synthesizing this machine. As a result, the user has control over the final controller implementation and is able to perform high-level trade-offs between control and data-path.

1 Introduction

Most high-level synthesis systems divide the synthesis problem into two subtasks: the generation of a data-path (allocation) and a corresponding control part (scheduling) [1]. This division makes it easier to handle design complexity and allows the use of specialized optimization techniques, such as state minimization [2] and encoding [3] on the FSM, and resource sharing on the data-path. However it may lead to inefficient designs. For example, flag variables are often introduced to store special status conditions of the machine and are used in conditional operations to affect the control-flow. Usually, these variables are stored in the data-path and used as inputs to the controller. However, in many cases better results can be achieved by moving these variables into the controller. Similarly, there are cases where optimizations can be achieved by moving variables from the controller into the data-path [4].

In current high-level synthesis systems, it is possible to influence the schedule by specifying constraints [5] or by inserting special statements in the description,

such as the *Next* statement in ISPS. However, the effect of these constraints depends on the scheduling algorithm used. Therefore it is difficult to influence, in a deterministic way, the final controller organization without knowledge about the scheduling algorithm.

At the logic level, it is possible to fully specify the functionality of the controller and data-path. However, the controller and data-path are fixed and trade-offs between control and data part require changing the description.

This paper presents a method and algorithms for exploring the design space between the register-transfer and behavioral levels. The goal is to be able to describe a design using a behavioral hardware-description language in such a way that extra control information can be specified by the designer, and taken into account during synthesis.

2 High-Level State Machines

2.1 Definition

A behavioral description can be represented as a control-flow and a data-flow graph. Given a behavioral specification, it is possible to partition it with respect to its control and data-flow characteristics.

Given a partition $B = (B_1, B_2, \dots, B_n)$ of the behavioral specification, a High-Level State Machine is defined as: $HLSM = [S, C, B, \delta, \rho]$, where: $S = (S_1, S_2, \dots, S_n)$ is the set of high-level states; $C = (C_1, C_2, \dots, C_m)$ is the set of control conditions which result from conditional operations; $\delta : S \times C \rightarrow S$ is the high-level state transition function, which specifies the next state depending on the present state and the conditions; $\rho : S \leftrightarrow B$ is the partition relation which assigns each high-level state S_i to a unique part B_i of the behavioral description which is to be performed if S_i is active.

Figure 1 illustrates the general concept of high-level state machines. Note that each B_i consists of a general sequential description which may include all types of control structures (for example, function calls, condi-

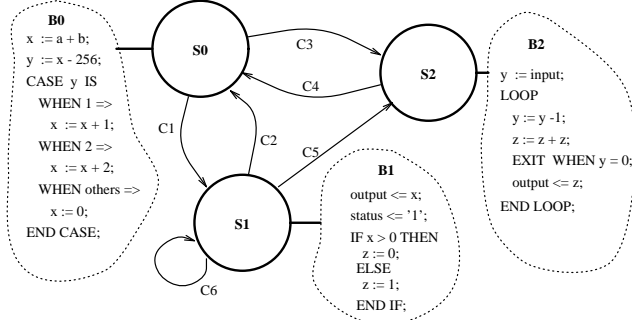


Figure 1: General structure of a high-level state machine

tionals and loops). Therefore each B_i may be scheduled into several states of the final controller.

A HLSM encompasses several description levels: from a pure behavioral description, where all operations are put into one high-level state, to a completely specified description, where all states in the final controller are defined (in this case the high-level states are equivalent to the final controller states). Synthesis of HLSMs can be performed at several levels, ranging from fully *respecting* the high-level state boundaries, to completely *overriding* them.

2.2 High-Level State Machine Specification

Finite-state machines are commonly specified by means of state variables. Similarly, HLSMs can also be specified using *high-level state variables*.

In order to avoid complexity problems and also to allow the designer to fully specify the partition of the behavioral description, the following restrictions on the specification of HLSMs are imposed:

- A HLSM is specified by a behavioral description, in which a single variable, chosen by the designer, is used as the high-level state variable. An initial value for this variable must be given.
- The control-flow graph must depend on the values of the state variable, in a way that each operation is associated unambiguously with a single value of the state variable.

Figure 2 shows an example of a HLSM specification (a) which implements a 16-bit integer division and the corresponding high-level state transitions diagram (b). Variable sv was chosen as the high-level state variable, which defines a partition with four high-level states. The operations in each state can be easily obtained by traversing the mutually exclusive branches from the

Case statement on sv (operation 1). State transitions are derived based on the operations which assign values to sv (operations 7, 8, 9, 15, 16, 22 and 25).

3 Synthesis of HLSMs

A high-level state in a HLSM may contain general sequential specifications. Therefore, synthesis of HLSMs requires the use of high-level synthesis algorithms such as scheduling and allocation.

The algorithms described in this section are related to the HIS system [7]. The same techniques can be easily adapted to other systems which use different control and data-flow representations. HIS uses control and data-flow graphs as inputs to the scheduling and allocation algorithms. For HLSM applications these graphs are constructed based on the control and data-flow of the partitions B_i and on the high-level state transition diagram of the HLSM.

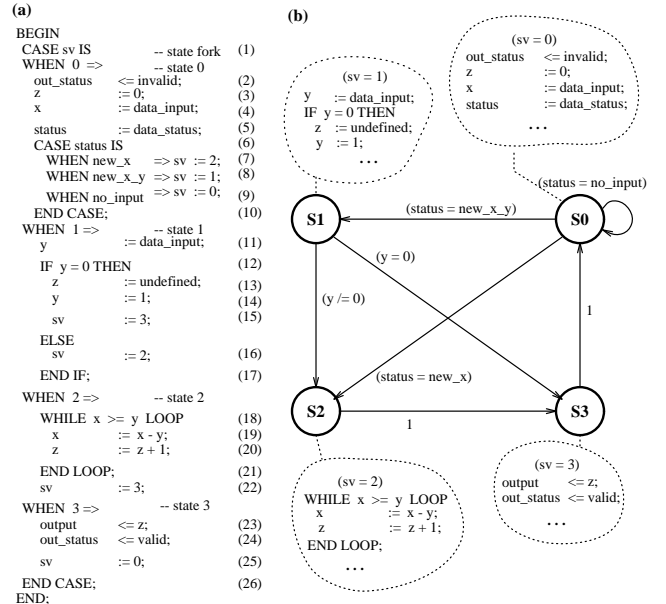


Figure 2: Example of HLSM description: (a) HDL source, (b) high-level state transition diagram

3.1 Control and Data-Flow Graphs for HLSMs

The *HLSM control-flow graph* is generated using the following algorithm:

- Generate the original control and data-flow graphs [6] from the behavioral description.
- Select the state variable (user defined) and determine the set of values assumed by it. These values

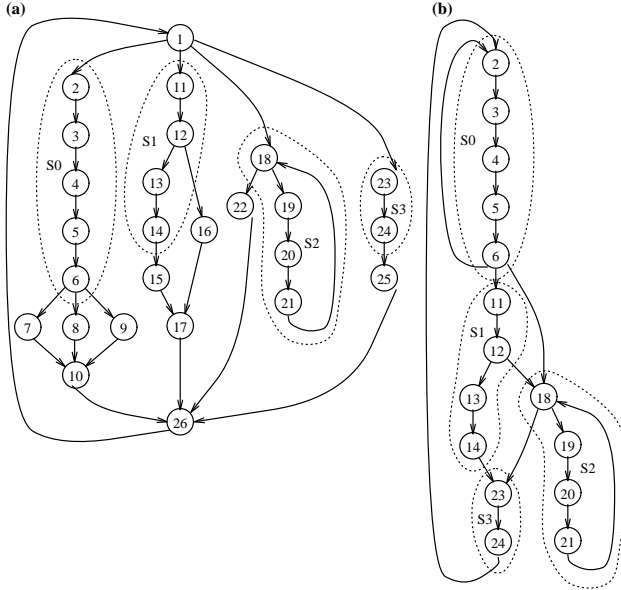


Figure 3: (a) Original control-flow graph, (b) HLSM control-flow graph.

constitute the set of high-level states S . The initial value of the state variable corresponds to the initial state of the HLSM.

- Traverse the original control-flow graph and determine the partitions B_i which are associated with each value of the state variable.
- Determine the high-level state transitions from the operations which assign values to the state variable.
- Reconnect the B_i according to the high-level state transition diagram. The new graph is the HLSM control-flow graph.

Figure 3a shows the original control-flow graph which is derived from the input description in Figure 2a. The resulting HLSM control-flow graph after removing the partitioning and reconnecting the parts is shown in figure 3b. Note that this graph is not a serial-parallel graph and its description by means of structured hardware description languages (e.g., VHDL) is not straight forward.

By construction, each high-level state transition in the HLSM corresponds to exactly one control-flow edge in the HLSM control-flow graph. By marking these edges the original HLSM boundaries can be recognized by the scheduling algorithm.

3.2 Control and Data-Path Synthesis

For the synthesis of the final FSM controller and data-path the scheduling and allocation algorithms implemented in the HIS system [5, 8, 9] allow the user to explore the design space by experimenting with several synthesis strategies:

Strategy 1: The final controller should have the same state transition diagram as the HLSM.

In this case the final controller is completely determined by the HLSM description. Scheduling detects the edges in the HLSM control-flow graph which correspond to high-level state transitions (marked edges), and generates new states at those points. This case is only possible if the operations in each high-level state can be scheduled in one controller state. The scheduling for the example in figure 2a according to this strategy leads to a controller with the same state/transition diagram shown in figure 2b.

Strategy 2: The scheduler should take into account the high-level state boundaries plus additional constraints.

In this case the scheduler *respects* the high-level state boundaries (as in strategy 1), but if there are additional constraints, the operations in some high-level states may be scheduled into several controller states. However, operations belonging to different high-level states are definitely scheduled in different controller states. The constraints can result from specific control statements (e.g. *LOOP* statements) or user defined area, delay or data-dependency constraints. Figure 4 shows the resulting schedule for the example in figure 2a enforcing a constraint of one ALU for operations “ \geq ” and “ $-$ ” (18 and 19). Due to this constraint, the high-level state 2 needs to be split into 2 states.

Strategy 3: The scheduler may override the high-level state boundaries.

In this case, the marked high-level state boundaries may be completely or partially overridden by the scheduling algorithm, so that new schedules based solely on the HLSM control and data-flow graphs can be explored. Resource constraints may also be considered. Figure 5 shows the resulting FSM controller produced by scheduling the control-flow graph in figure 3b, disregarding the high-level state boundaries and with no constraints.

Strategy 4: Synthesis of the original control and data-flow graphs.

Since no specific syntactical or semantical constructs are used for the HLSM description, scheduling and allocation can always be applied to the original control and data-flow graphs. In this case the high-level state variable is realized as an ordinary register in the data-path. Resource constraints may also be considered.

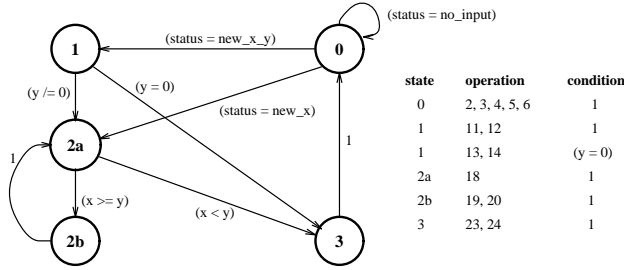


Figure 4: Final FSM and schedule of operations for DIVIDER using strategy 2 and a constraint of 1 ALU $[\geq, -]$.

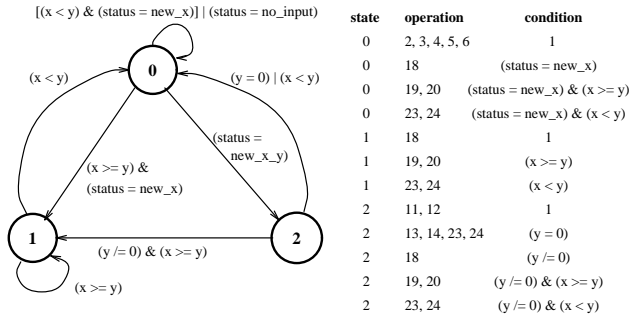


Figure 5: Final FSM and schedule of operations for DIVIDER using strategy 3 with no constraints.

4 Results

The algorithms for synthesis of high-level state machines described in this paper were implemented in the HIS system [7]. The design examples were described as HLSMs and used to test the various synthesis strategies: Table 1 presents the results according to the strategies described in section 3.2. Subcases (a) and (b) correspond to constraints due to loops and user given constraints respectively.

The results of synthesis respecting the HLSM boundaries and without user defined constraints are shown in case 1 and 2a. Examples FFSM, GFSM, GCD, and DIVIDER preserve the state transition diagram of the HLSM in the controller FSM (strategy 1) whereas, for CAMERA, COUNTERARRAY, and INTEG the operations in the high-level states could not always be scheduled in one controller state (strategy 2).

Cases 3a show the results for synthesis strategy 3 which ignores the high-level state boundaries and applies no user constraints. This strategy was not applied to examples FFSM and GFSM, since this would affect the behavior of the machines.

Cases 2b and 3b (strategy 2 and 3 respectively)

show the effect of additional area constraints on the final implementation. For examples GCD and DIVIDER the synthesis system was directed to use one ALU for four and two arithmetic operations respectively. As a result the final number of cells could be reduced at the expense of a slower schedule.

In case 4 (strategy 4) the scheduling and allocation algorithms are applied to the original control and data-flow graphs. This case produced an implementation with the same performance as in case 1 or 2, in terms of operations per clock cycle. However for most examples (except INTEG and GFSM) the number of cells needed for this implementation was larger than in case 1 or 2.

HLSM Descr.	Strat.	Controller		Data Path		Cells	
		States/Trans.	Reg.-bits	Reg./R.bits	Mux.inp./M.bits		
camera	5/12	(2a)	6/12	3	0/0	8/8	100
		(3a)	14/27	4	0/0	4/8	188
		(4)	6/16	3	1/3	14/26	265
		(2a)	6/8	3	5/32	18/105	785
counter-array	4/4	(3a)	4/5	2	4/31	21/140	862
		(4)	3/6	2	6/33	20/119	788
		(2a)	5/6	3	2/16	6/20	441
integ	2/2	(3a)	5/7	3	2/16	8/36	450
		(4)	4/6	2	3/17	8/22	435
		(1)	7/28	3	0/0	7/21	125
FFSM	7/28	(4)	1/1	0	1/3	14/42	156
		(1)	5/17	3	0/0	5/15	81
GFSM	5/17	(4)	1/1	0	1/3	10/30	75
		(1)	3/5	2	2/32	8/128	944
GCD	3/4	(2b)	5/7	3	2/32	8/128	850
		(3a)	2/4	1	2/32	22/352	1358
		(3b)	4/6	2	3/33	12/192	991
		(4)	2/4	1	3/34	11/134	973
divider	4/7	(1)	4/8	2	3/48	9/114	1061
		(2b)	5/9	3	3/48	9/114	1026
		(3a)	3/7	2	3/48	21/306	1353
		(3b)	4/8	2	3/50	15/210	1176
		(4)	2/4	1	4/50	13/122	1088

Table 1: Results of synthesis for the various strategies (not all strategies were applicable for all examples).

5 Conclusions

This paper presented algorithms for the synthesis of high-level state machines. A HLSM combines, in the same description, the advantages of a specified control structure with the flexibility of behavioral descriptions inside each high-level state. By describing a design in terms of a HLSM, the user can influence precisely the final controller implementation and can perform high-level trade-offs between control and data-path. This technique effectively bridges the gap between RTL synthesis and high-level synthesis, by giving the user more control over the synthesis results at the same time that it uses high-level synthesis algorithms for scheduling and allocation. Furthermore, by using procedure calls in the input description, hierarchical HLSMs can be

specified and synthesized by this approach.

The results showed the effectiveness of the approach. In many cases smaller implementations were obtained by synthesizing the designs respecting the control partitioning present in the HLSM description.

References

- [1] M. C. McFarland, A. C. Parker, and R. Camposano, "The high-level synthesis of digital systems," *Proceedings of the IEEE*, pp. 301–318, February 1990.
- [2] G. D. Hachtel, J.-K. Rho, F. Somenzi, and R. Jacoby, "Exact and heuristic algorithms for the minimization of incompletely specified state machines," in *Proc. of The Europ. Conf. on Design Automation*, (Amsterdam), pp. 184–191, IEEE, Febr. 1991.
- [3] T. Villa and A. Sangiovanni-Vincentelli, "NOVA: State assignment of finite state machines for optimal two-level logic implementations," *IEEE Trans. on Computer-Aided Design*, vol. CAD-9, pp. 905–924, Sept. 1990.
- [4] W. Wolf, "Decomposing data machines," in *Proc. of The Europ. Conf. on Design Automation*, (Amsterdam), IEEE, Febr. 1991.
- [5] R. A. Bergamaschi, R. Camposano, and M. Payer, "Scheduling under resource constraints and module assignment," *INTEGRATION*, vol. 12, Dec. 1991.
- [6] R. Camposano and R. M. Tabet, "Design representation for the synthesis of behavioral VHDL models," in *Proceedings 9th International Symposium on Computer Hardware Description Languages and Their Applications*, (Washington, D.C.), Elsevier Science Publishers B.V., June 1989.
- [7] R. Camposano, R. A. Bergamaschi, C. Haynes, M. Payer, and S. M. Wu, "The IBM high-level synthesis system," in *High-Level VLSI Synthesis* (R. Camposano and W. Wolf, eds.), pp. 79–104, Kluwer Academic Publishers, 1991.
- [8] R. Bergamaschi, R. Camposano, and M. Payer, "Data-path synthesis using path analysis," in *Proceedings of the 28th ACM/IEEE Design Automation Conference*, pp. 591–596, ACM/IEEE, June 1991.
- [9] R. Camposano, "Path-based scheduling for synthesis," *IEEE Trans. on Computer-Aided Design*, vol. CAD-10, pp. 85–93, Jan. 1991.