Dynamic Scheduling and Synchronization Synthesis of Concurrent Digital Systems under System-Level Constraints

Claudionor N. Coelho Jr.
Center for Integrated Systems
Stanford, CA 94305
coelho@pegasus.stanford.edu

Giovanni De Micheli
Center for Integrated Systems
Stanford, CA 94305

Abstract

We present in this paper a novel control synthesis technique for system-level specifications that are better described as a set of concurrent synchronous descriptions, their synchronizations and constraints. The proposed synthesis technique considers the degrees of freedom introduced by the concurrent models and by the environment in order to satisfy the design constraints.

Synthesis is divided in two phases. In the first phase, the original specification is translated into an algebraic system, for which complex control-flow constraints and quantifiers of the design are determined. This algebraic system is then analyzed and the design space of the specification is represented by a finite-state machine, from which a set of Boolean formulas is generated and manipulated in order to obtain a solution. This method contrasts with usual high-level synthesis methods in that it can handle arbitrarily complex control-flow structures, concurrency and synchronization by allowing the scheduling of the operations to change dynamically over time.

1 Introduction

We consider in this paper a control synthesis technique for system-level descriptions of synchronous systems that are better specified as a set of concurrent routines and their synchronizations. In these systems, the synthesis task consists of obtaining controllers that satisfies design and synchronization constraints.

The synthesis effort for these systems is highly dependent on the complexity of the constraints used to model the interactions of the system. When these design and synchronization constraints do not span over the concurrent parts of the design, conventional high-level synthesis techniques can be used to obtain reasonable (if not optimal) controllers. However, when considering more complex design and synchronization constraints that span over the concurrent parts of the design, or constraints that establish relations between the system to be synthesized and the environment, conventional high-level synthesis techniques often produce suboptimal results or even no results at all, due to their local scope in synthesizing each sequential component or data-flow graph at a time.

In order to satisfy these complex design and synchronization constraints, we propose here a novel technique, called *dynamic scheduling*, that generates a minimal pool of schedules satisfying the constraints at synthesis time, and selects the schedule that best matches the design and synchronization constraints at execution time.

This technique allows the synthesis procedure to consider the degrees of freedom introduced by the several concurrent parts of the design and by the environment during the synthesis of each controller, which allows the solution of problems that would not be synthesizable by conventional high-level synthesis tools. Due to its generality, this technique also allows the synthesis of concurrent synchronous circuits with arbitrary loop and conditional control structures, their synchronization, and external constraints, which are only handled by conventional high-level synthesis techniques in a limited way.

This paper is organized as follows. In Section 2, we present the abstraction of the specification into an algebraic system called the *algebra of control-flow expressions* (or CFEs). In Section 3, we show how to represent design constraints as CFEs. In Section 4, we show how the algebraic specification can be represented as a finite-state machine. In Section 5, we cast problem of selecting the schedules as an integer linear programming (ILP) problem. In Section 6, we show how the controllers for each part of the specification can be obtained such that operations are scheduled dynamically and synchronization constraints are satisfied. In Section 7, we compare our technique with existing methodologies for synthesis. In Section 8, we show how dynamic scheduling and synchronization synthesis can be used in the protocol conversion example. Finally, in Section 9 we present conclusions and possible extensions of this work. Figure 1 shows how the different sections of this paper relate to the synthesis flow in our tool.

2 Control-Flow Expressions

We present in this section a brief introduction to a subset of *process algebras* [1] that we use to model synchronous concurrent systems, called the algebra of control-flow expressions (or CFEs). A more complete presentation of modeling aspects of CFEs can be found in [7].

Control-flow expressions abstract away the data-flow information by focusing only on the control-flow of the specification. The data-flow operations are abstracted by two sets. The first set, called *action* set, associates a label with operations that execute in a **single cycle**. Multi-cycle operations can be represented by a sequence of single cycle actions. The second set, called *conditional* set, associates a label with the conditional guards of loops and alternative constructs. Boolean functions on the set of conditionals (also called *guards*) are assumed to trigger

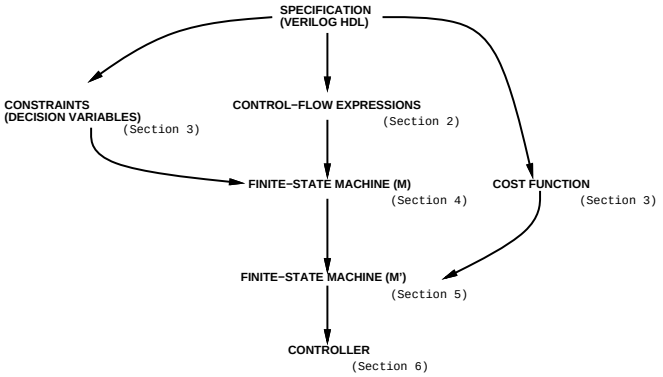

Figure 1: *Flow of the synthesis procedure from a system-level specification*

Composition	HL Representation	CF Expression
Sequential	**begin** P; Q **end**	$p \cdot q$
Parallel	**fork** P; Q **join**	$p \| \| q$
Alternative	**if** (C) P ; **else** Q ;	$c : p + \overline{c} : q$
Loop	**while** (C) P ;	$(c : p)^*$
	wait $(!C)$ P ;	$(c : 0)^* \cdot p$
Infinite	**always** P ;	p^ω

Table 1: *Link between Verilog HDL Constructs and Control-Flow Expressions*

the execution of actions.

In order to represent the control-flow of a design, control-flow expressions incorporate the usual high-level language constructs, namely the constructs for sequential, alternative, loop, unconditional repetition and parallel composition. In addition to these usual high-level constructs, two special symbols, 0 and ϵ represent respectively a single cycle action that does not perform any computation and executes in one cycle and a null computation that executes in zero time. For example, Table 1 shows the representation of the control-flow constructs for Verilog HDL in terms of control-flow expressions. In this figure, we assume that p and q denote the control-flow expressions representing the Verilog HDL code for P and Q, the conditional c abstracts the operation C, and the guards c and $\overline{c}$ encapsulate the conditions for the alternative and loop choices that must be taken during the execution of the Verilog HDL program.

Example 1 [Ethernet Coprocessor Synchronization] In this example, we show how to abstract a design in terms of a control-flow expression. In the ethernet controller specified as a set of concurrent routines [2], we focus on the routines that communicate with a microprocessor through a shared bus, namely the routines *DMArcvd*, *DMAxmit* and *enqueue*.

```
module DMArcvd;
always
begin
   bus write;
   process data
end
endmodule
```

```
module DMAxmit;
always
begin
   initialize vars.
   wait (tx ready);
   bus read;
end
endmodule
```

```
module enqueue;
always
begin
   wait (free bus);
   bus read;
end
endmodule
```

Figure 2: *Abstracted behaviors for DMArcvd, DMAxmit and enqueue*

From an abstraction and reduction of the original specification that captures only the bus accesses, we can obtain the descriptions of Figure 2, specified at high-level.

If we assume that a denotes the action of accessing the bus, that executes in one cycle, $\overline{c}$ denotes the guard "transmission is ready" and that $\overline{x}$ denotes the guard that "bus is free", then we can represent the routines presented above by the following control-flow expressions:

$$\begin{aligned} \textit{DMArcvd} &= (a \cdot 0)^\omega \\ \textit{DMAxmit} &= (0 \cdot (c : 0)^* \cdot a)^\omega \\ \textit{enqueue} &= ((x : 0)^* \cdot a)^\omega \end{aligned}$$

In these control-flow expressions, we can see the timing relation among the different components. In the control-flow expression representing the routine *DMArcvd*, the bus access (action a) is executed every other cycle. Thus, 0 acts as an internal computation that can be abstracted for the purposes of representing the system's behavior in terms of bus accesses. In the control-flow expression for the routine *DMAxmit*, the bus access occurs only after the guard c is *false*. In this module, it can be seen that first some internal computation is executed in the first cycle, that we represent here by 0, followed by a pooling of c for an arbitrary number of cycles until it becomes *false*. In this CFE, and also in the next one, action 0 in the loop represents an idle cycle in the execution model. In the control-flow expression for the routine *enqueue*, the bus access occurs only when the bus is free, which is represented here by x being *false* □

3 Constraint Representation

Constraints are properties that should be satisfied by any implementation. Examples of constraints are minimum and maximum timing constraints, possible resource bindings for an operation, and possible or necessary synchronizations. We specify constraints as control-flow expressions by rewritting the control-flow expression of the specification, and by adding new control-flow expressions whose the sole purpose is to ensure that the specification observes the constraints. These constraints specified as CFEs act as *passive processes* of VHDL [10], but with the difference that these expressions are used to guide the synthesis tool during the synthesis, and not as watchdogs during simulation. We will consider in this paper only synchronization and scheduling constraints. Binding and more complex constraint specifications can be found in [8].

In the following example, we show how to represent constraints in control-flow expressions by means of the specification of synchronization sets and by a quantification of the design space.

Example 2 [Synchronization Constraints] In the Example 1, the routine *enqueue* synchronizes with some "free bus" signal, which was not specified. In fact, this signal depends on the time the two routines *DMArcvd* and *DMAxmit* access the bus, i.e. the routine *enqueue* must access the bus whenever neither *DMAxmit* nor *DMArcvd* access the bus. In order to specify the constraint that the bus should be free when the *enqueue* executes, we use multisets of actions that never execute at the same time. This set is called here the *NEVER* set. In this example, *NEVER* $= \{a, a\}$, since no two bus accesses should execute at the same time (similarly, we have a synchronization constraint specifying that a multiset of actions should always execute at the same time, and we call this set the *ALWAYS* set [8]). □

In addition to considering the synchronization sets represented by the *ALWAYS* and *NEVER* sets, we also represent quantitatively the design space in control-flow expressions by decision variables.

Example 3 [Decision Variables] In the case we want to synthesize the synchronization between *enqueue* and the routines *DMArcvd* and *DMAxmit* in the previous example, we consider x of *enqueue* to be a *decision variable*, i.e. whenever x is *false*, then the *enqueue* will be allowed to access the bus. The synthesis task then becomes the assignment of values to x over time.

Thus, the set of guards in a control-flow expressions is divided into two sets of variables: the set of conditionals (which is a part of the original specification) — c in this case — and the set of decision variables — x. Decision variables will be assigned over time during the synthesis of the controllers and they will determine a scheduling of the operations over time that satisfies synchronization constraints. The reader should notice that any assignment over time to these decision variables represent a possible choice of a schedule for the bus access in the routine *enqueue*.

A controller satisfying the synchronization constraint mentioned above for routine *enqueue* can be given by the CFE $(0 \cdot (\overline{c} : 0 \cdot 0)^* \cdot a)^\omega$. This CFE can be obtained from the original control-flow expression $((x : 0)^* \cdot a)^\omega$ by first transforming the original CFE into $((x_0 : 0 + \overline{x}_0 : a) \cdot (x_1 : 0 \cdot (x_2 : 0 + \overline{x}_2 : a))^* \cdot a)^\omega$. Then, x_0 and x_2 are assigned the value *true* because the routine *DMArcvd* accesses the bus every other cycle beginning in the first cycle; and the variable x_1 is assigned $\overline{c}$ because the routine *enqueue* can only use the bus when the routine *DMAxmit* is not transmitting data (captured by guard $\overline{c}$ that means *transmission is ready*), as presented previously. □

4 Finite-State Representation of the Design Space

In order to obtain the solution satisfying the constraint of a bus free of conflicts proposed in the Example 2, we first had to convert the CFE $((x : 0) \cdot a)^\omega$ into an equivalent form with respect to unrolling. Then, Boolean expressions were assigned to the instances of the decision variable x in this new expression. In this section, we will show how we can obtain an equivalent finite-state machine representation of the original specification satisfying design constraints. This finite-state machine will represent the solution space in a compact way, since the different choices allowed by the synthesis tool are guarded by decision variables.

In order to build the finite-state representation, we will use the notion of the *irredundant suffixes* of a control-flow expression. Informally, a *suffix* of a control-flow expression p is a subexpression of p. The set of *irredundant suffixes* of p is defined to be the set of suffixes of p such that no two suffixes of the set differ in the number of times a loop or infinite composition is unrolled, in the different permutations of the subexpressions in alternative and parallel compositions, and in the use of ϵ as the null element for sequential composition. A more formal description of *suffixes* and *irredundant suffixes* of a control-flow expression can be found in [8].

Example 4 [Suffixes of a Control-flow Expression] The suffixes of the control-flow expression $(a \cdot b \cdot c)^\omega$, i.e. the unconditional repetition of a, followed by b, followed by c, are $(a \cdot b \cdot c)^\omega$, $a \cdot b \cdot c \cdot (a \cdot b \cdot c)^\omega$, $b \cdot c \cdot (a \cdot b \cdot c)^\omega$, $c \cdot (a \cdot b \cdot c)^\omega$, $a \cdot b \cdot c \cdot a \cdot b \cdot c \cdot (a \cdot b \cdot c)^\omega$, $b \cdot c \cdot a \cdot b \cdot c \cdot (a \cdot b \cdot c)^\omega$, etc ... □

Example 5 [Irredundant Suffixes of a CFE] The set of irredundant suffixes of $(a \cdot b \cdot c)^\omega$ are $(a \cdot b \cdot c)^\omega$, $b \cdot c \cdot (a \cdot b \cdot c)^\omega$ and $c \cdot (a \cdot b \cdot c)^\omega$, since all other suffixes of $(a \cdot b \cdot c)^\omega$ correspond to different number of unrollings of the infinite composition, e.g. $a \cdot b \cdot c \cdot (a \cdot b \cdot c)^\omega$ is equivalent to $(a \cdot b \cdot c)^\omega$. □

We consider here the finite-state Mealy machine $M = (I, O, Q, \delta, \lambda, q_0)$ to represent the design space, where I is the set of input variables to M, O is the set of output variables of M, Q is the set of states of M, q_0 is the initial state of M, δ is the transition function of M, i.e. $\delta : Q \times 2^I \rightarrow Q$, and λ is the output function of M, i.e. $\lambda : Q \times 2^I \rightarrow 2^O$.

The Mealy machine M is related to a finite-state machine representation of p in the following way. The set of input variables of M corresponds to the set of conditionals and decision variables of p. The set of outputs of M corresponds to the multiset of actions of p. For each irredundant suffix s of p, we associate a state $q_s \in Q$. In particular, the initial state q_0 corresponds to q_p.

In order to associate the transition (δ) and output (λ) functions of M with the suffixes of p, let the control-flow expressions s_1 and s_2 be irredundant suffixes of p such that $s_1 = f : \{a_1, \cdots, a_n\} \cdot s_2 + t$, for f being a Boolean function over the set of conditional and decision variables, $\{a_1, \cdots, a_n\}$ being some multiset of actions $a_1, \cdots, a_n$, and t some control-flow expression. Since s_1 and s_2 are suffixes of p, then $\delta(q_{s_1}, f) = s_2$ and $\lambda(q_{s_1}, f) = \{a_1, \cdots, a_n\}$ in M. We can interpret these equations in the following way. The multiset of actions $\{a_1, \cdots, a_n\}$ represents the actions that execute at the same time in s_1 whenever guard f is *true*. Since every action executes in one cycle, $\{a_1, \cdots, a_n\}$ also executes in one cycle. Thus, s_2 represents the state of the system obtained by a one-cycle simulation of s_1 if the Boolean guard f is *true*.

Note that the Mealy machine described above does not take into consideration any synchronization constraints

(*ALWAYS* and *NEVER* sets). Thus, we still have to remove from M the transitions that violate these synchronization constraints (*invalid transitions*) and the states that are unreachable due to the deletion of some transitions (*unreachable states*).

Valid transitions are obtained by checking that in a transition if at least a certain action is included in some multiset of actions of the *ALWAYS* set, then all actions in this multiset should be executed in the transition. Furthermore, this transition should not include any multiset of actions of the *NEVER* set. Reachable states are obtained by eliminating the states of M which are not reachable from the initial state after removing invalid transitions and other unreachable states.

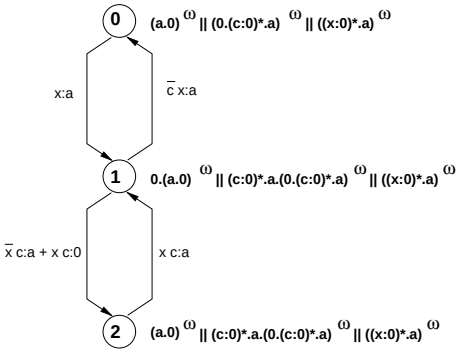

Figure 3: *Mealy machine for Synchronization Problem in Ethernet Coprocessor*

Example 6 [Mealy Machine for Synchronization in Ethernet Coprocessor] For the ethernet coprocessor system of Example 1 represented by the control-flow expression $(a \cdot 0)^{\omega} \| (0 \cdot (c : 0)^* \cdot a)^{\omega} \| ((x : 0)^* \cdot a)^{\omega}$, if we consider the sets *ALWAYS* $= \emptyset$ and *NEVER* $= \{\{a, a\}\}$, we obtain the finite-state Mealy machine of Figure 3 that is composed only by reachable states and valid transitions. □

5 Optimal Assignment of Decision Variables

The finite-state Mealy machine $M = (I, O, Q, \delta, \lambda, q_0)$ obtained in Section 4 with Q representing the set of reachable states and δ representing the set of valid transitions contains the possible degrees of freedom that the synthesis can use during the synthesis of the description.

We should remember that the set I of M corresponds to the set of conditionals and decision variables of the control-flow expression, so our synthesis problem will be an assignment of the decision variables to Boolean variables that minimizes some cost function. Note that each decision variable will be decided for each state in the worst case, so depending on the state of the system, different choices may be selected.

The problem of synthesizing $M = (I, O, Q, \delta, \lambda, q_0)$ corresponds to finding a submachine $M' = (I', O, Q', \delta', \lambda', q_0)$ such that $Q' \subseteq Q$, $\delta' \subseteq \delta$, $\lambda' \subseteq \lambda$, and I' being the set of conditional variables of I, since the decision variables are assigned Boolean values by the procedure described in this section. In addition to that, M' is also the minimum cost submachine of M with respect to the objective function specified by the designer [8]. This objective function orders the different solutions with respect to minimum latency or minimum area. Minimum area is represented by associating a cost function β with each resource corresponding to an action. Thus, each action a whose cost is defined as $\beta(a)$, will increase the cost of the implementation if a transition of M with output a is also a transition of M'.

The selection of a cost function for minimum latency can also be computed in a similar way. If some decision variable x determines when the execution of a action will be delayed by one cycle, which we represent by action 0, then the cost of the implementation will be increased every time there is a transition with x being *true* in M'.

The submachine M' should further satisfy some other constraints. For example, although several different schedules and bindings may exist for an action, only one such schedule and binding must be taken at any particular cycle, i.e. M' should be able to make deterministic choices about its schedules and bindings at any state $q \in Q$.

In the synthesis of M' from M, we have to identify which states will be included in M' and which transitions will be part of the transition function for M'. In order to determine the states of M which will be part of the states of M', we create a Boolean variable y_p for each state q_p of M. If the Boolean variable y_p is set to 1, our interpretation will be that the state q_p will belong to M'. We will denote the state q_p by p in the remaining of this section.

In order to determine a subset of the transitions of M', we subdivide each guard f of a transition $\delta(q_p, f)$ into two conjoined parts. The first part contains only decision variables and the second part contains only conditional variables. Let us call the first part $f_{\mathbf{X}}$ and the second part $f_{\mathbf{C}}$. Now, for each state q_p and for each different Boolean formula $f_{\mathbf{C}}$ of q_p, we create a Boolean variable $x_{(q_p, f_{\mathbf{C}})}$ for each decision variable $x \in \mathbf{x}$. The solution of the integer linear programming (ILP) problem will determine assignments to variables $x_{(q_p, f_{\mathbf{C}})}$, which in turn will determine assignments of Boolean values to $f_{\mathbf{X}}$. If $f_{\mathbf{X}}$ is evaluated to 0, then the transition $\delta(q_p, f)$ will not belong to M'. If $f_{\mathbf{X}}$ is evaluated to 1, $\delta(q_p, f)$ will belong to M'. This notation will be used in the description of the set of characteristic equations for the example below.

Example 7 [ILP Equations for Synchronization Problem] Let us consider the finite-state Mealy machine of Figure 3 for the synchronization problem presented in Example 2. For this finite-state machine, the set of 01-ILP equations that quantifies the valid assignments to the decision variable x is shown below.

$$
\begin{aligned}
y_0 &= 1 \\
y_1 - (x_{(0,1)} y_0 \vee x_{(2,c)} y_2) &= 0 \\
y_2 - y_1(\overline{x_{(1,c)}} \vee x_{(1,c)}) &= 0
\end{aligned}
$$

$$
\begin{aligned}
x_{(0,1)} &= 1 \\
x_{(1,\overline{c})} &= 1 \\
x_{(1,c)} + \overline{x_{(1,c)}} &= 1
\end{aligned}
$$

$$x_{(2,c)} = 1$$

$$(x_{(0,1)} \vee \overline{y_0})(x_{(1,c)}x_{(1,\overline{c})} \vee \overline{y_1})(x_{(2,c)} \vee \overline{y_2}) = 0$$

The first set of equations represent the transition relation of M in terms of the decision variables and states. The first state of M (0) is always a state of M'. State 1 will be a state of M' if 0 is a state of M' and the transition $\delta(0, x)$ is in M', which is represented by assigning 1 to $x_{(0,1)}$; or if state 2 is a state of M' and the transition $\delta(2, xc)$ is in M', which is represented by assigning 1 to the Boolean variable $x_{(2,c)}$. A similar reasoning yields the third equation.

In the second group of equations, we represent set of valid assignments for each state and conditional expression. The first equation states that the only possible choice for state 0 is to make a transition to state 1, and thus, $x_{(0,1)}$ should be assigned to 1. Similarly, when c is *false* on state 1, since the only possible choice is a transition to state 0, this transition should be a transition of M'. In the transition between states 1 and 2, there are two possible choices when c is *true*, and only one of those transitions should be assigned to M'.

In the third set of equations, we guarantee that for any causality constraint of the type $a \cdot (x : 0)^* \cdot b$, where a and b are actions and x is a decision variable, at least one state of M' will have x assigned to *false*, i.e. b will eventually be scheduled.

An objective function for minimum latency can be given by *min* $y_0 x_{(0,1)} + y_1(x_{(1,\overline{c})} | x_{(1,c)}) + y_2 x_{(2,c)}$, where in this equation $|$ represents Boolean disjunction and $+$ represents arithmetic addition.

An assignment satisfying these equations is given by $y_0 = y_1 = y_2 = 1$, $x_{0,1} = x_{1,\overline{c}} = x_{2,c} = 1$, $x_{1,c} = 0$. □

The general method for obtaining the set of ILP equations for a machine M can be found in [8]. It was not added here due to the lack of space.

6 Controller Synthesis Using Dynamic Scheduling and Synchronization Synthesis

We show in this section how the assignment procedure for the decision variables of a system-level design will lead to the dynamic scheduling of operations over time.

When considering the synthesis problem for concurrent descriptions, we may require the selection of different schedules over time for an operation. Let us consider the precedence constraint $o_1 \rightarrow o_2$ (operation o_1 must be executed before operation o_2 is executed) of a model that executes concurrently with some other models. Because of synchronization constraints or tight resource constraints, the synchronization among the concurrent parts of the design may require that the relative schedule of o_2 with respect to o_1 to depend on the other models. In this case, the schedule of operation o_2 will be determined from a pool of schedules at execution time. We define the dynamic selection of a schedule from a pool of schedules the *dynamic scheduling* problem.

The situation presented in the previous paragraph also occurs in the domain of control-flow expressions. For example, in the control-flow expression $a \cdot (x : 0)^* \cdot b$, where a and b are actions and x is a decision variable, we may have a situation in which x is assigned three consecutive times 1 and a following 0 (resulting in the schedule $a \cdot 0 \cdot 0 \cdot 0 \cdot b$) for some sequence of states of M' and a 0 (resulting in the schedule $a \cdot b$) for some other sequence of states of M'. This implies that depending on the state of execution of the other concurrent models, either the first or the second schedule would be selected.

We now show how we can obtain a controller for the original models from M' that uses the dynamic scheduling and synchronization. This controller is obtained by reverting the assignments to the original specification. More formally, for the submachine $M' = (I', O, Q', \delta', \lambda', q_0)$, for which an assignment to the decision variables of the system represented by the CFE $p = p_1 || \cdots || p_n$ was obtained in the previous section, we build machines $M_i = (I_i, O_i, Q', \delta_i, \lambda_i, q_0)$ for each concurrent part p_i of p. M_i will be a controller for this concurrent part of p.

The set I_i of inputs of M_i corresponds to the set of conditional variables I. The set of outputs of M_i corresponds to the multiset of actions of p_i, which is a subset of O, respectively, since p_i is a subexpression of p. The transition function δ_i has the same transitions of δ'. The output function λ_i is a restriction of λ' such that the outputs are restricted to O_i.

Let us interpret this new transition function δ_i and the output function λ_i. Suppose we computed the finite-state machine representation N for p_i alone. In this finite-state machine representation, let us assume a state transition and an output generation that is dependent on some decision variable. After synthesizing the finite-state representation for p, and obtaining M_i, the transition of N was replaced by one or more transitions which depended only on the conditional variables. Even if the number of states in N and M_i does not agree, there will be equivalent transitions for N and M_i such that for each two equivalent states of N and M_i, there will be two corresponding transitions. This equivalence exists because p_i is a subexpression of p. Thus, this change in the transition function can be interpreted as if the decision variable of p_i were assigned the Boolean expression associated with the transition of δ_i. Thus, this mechanism allows the dynamic reconfigurability of the system according to the system's state, based on the conditionals.

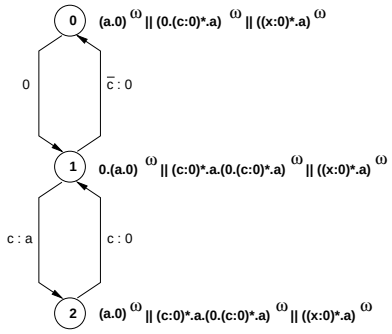

Figure 4: *Finite-State Machine for Control-Flow Expression* $((x : 0)^*.a)^\omega$

Example 8 [Controller for model *enqueue***]** For the ethernet coprocessor example discussed before, where $p =$ *DMAxmit*||*DMArcvd*||*enqueue*, machine M' is given by the

finite-state machine of Figure 3 after assigning the instances of the decision variable x to the values determined in the previous section.

Machine M_i, which is the controller for the model *enqueue*, is shown in Figure 4. This controller corresponds to the CFE $(0 \cdot (\overline{c} : 0 \cdot 0) \cdot a)^{\omega}$, which was shown earlier to be a solution to this problem. □

It should be noted that the finite-state machine we obtain by the procedure above does not guarantee any minimality with respect to the number of states, but it only gives a finite-state machine that satisfies the original constraints and has a minimal cost. We use the state minimizer *Stamina* [14] to obtain the minimum number of states for the finite-state machine implementing the control-flow expression p_i. In fact, in Example 8, an implementation with minimum number of states can be obtained with just 2 states.

7 Comparisons and Limitations

In this section, we compare our procedure to schedule, bind and synthesize synchronization schemes with previous approaches. Although a great deal of work has been reported for scheduling and binding techniques [4, 9, 5, 6, 15, 3, 13, 11], we will compare our method with exact methods that were solved as a 0-1 ILP, such as [6, 13].

When compared to other 0-1 ILP techniques [6, 13], our methodology appears as a more general technique for the scheduling problem, because it can handle loops, synchronization and multi-rate execution of concurrent models. However, the penalty paid for a more general method is the number of variables to be solved by the ILP, which is greater by a constant factor with respect to these other exact approaches.

If we consider the finite-state machine representation M of a control-flow expression p with n_s states, n_c conditionals and n_d decision variables, then the number of variables in the worst case will be on the order of $\mathcal{O}(n_s n_d 2^{n_c})$. Note, however, that in practical terms this upper bound is never reached, since not all decision variables will be evaluated in every state and not all possible expressions on conditionals are evaluated at the same time. If we compare this complexity with the complexity of the 0-1 ILP method of [6], we note that n_s is related to the number of control-steps an operation can be scheduled in [6], n_d is related with the number of operations to be scheduled in [6] and $n_c = 1$ in [6], since no conditional paths are allowed.

8 Implementation and Results

We implemented a program to synthesize controllers with dynamic schedules from control-flow expressions in 12,000 lines of C, and a 0-1 ILP solver using Binary Decision Diagrams (BDDs) in 3,000 lines of C.

Since the technique presented in this paper is targetted for the synthesis of concurrent systems under synchronization, which is a new area, there are no standard benchmarks yet. Thus, instead of comparing our approach with the existing techniques for scheduling and binding using standard benchmarks, we will show an application of this technique for optimizing synchronous circuits for protocol conversion from high-level specifications.

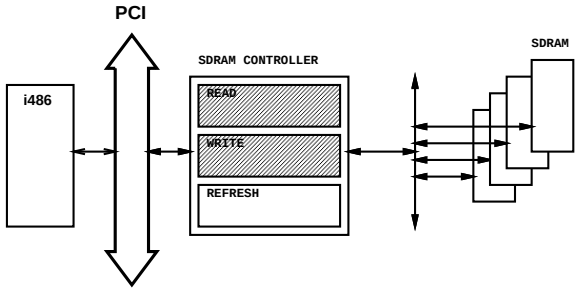

Figure 5: *Protocol Conversion for PCI bus computer*

8.1 Protocol Conversion

In this section, we show how we can use synchronization synthesis in order to synthesize the controller for converting the PCI bus protocol [12] into a synchronous DRAM protocol. In particular, we will provide here the conversion between reading and writing cycles of a PCI bus into synchronous DRAM cycles. Figure 5 shows the diagram of computer using a PCI bus, and a synchronous DRAM (SDRAM) memory bank. Both protocols allows single or burst mode transfers, with the difference that SDRAMs burst mode are limited to at most 8 transfers on the same row that are one cycle apart from each other.

Informally, a PCI bus cycle begins with a address phase, followed by one or more data phases. Wait states can be inserted in the data phase by either the microprocessor or by the memory. For burst mode transactions, we assume here a linear increment of the address space.

The synchronous DRAM reading protocol begins by a row address selection (RAS) phase followed by a column address selection (CAS) phase. After the CAS phase, and a fixed number of cycles, the SDRAM will produce data at a rate of one word/cycle.

We implemented the four models for the reading and writing cycles of the PCI local bus and the SDRAM in 230 lines of a high-level subset of Verilog HDL. These models are predefined libraries that can synchronize with any circuit. We thus use the technique of synchronization synthesis in order to synthesize a combined controller that is smaller than the two separate controllers.

Table 2 shows the number of states for the controllers in terms of a Mealy machine, when each part is synthesized separately, and when the controller for both models is generated as a single controller. (which is highly desirable, since both parts are highly synchronized).

Model	States PCI	States SDRAM	States PCI + SDRAM	Execution Time
READ	7	15	34	3.5 s
WRITE	6	7	30	1.6 s

Table 2: *PCI/SDRAM Protocol Conversion Example*

9 Conclusions

We considered in this paper the synthesis problem for system-level designs specified as a set of concurrent and

interacting parts. We presented a tool to allow the synthesis of concurrent descriptions, that considered arbitrarily complex control-flows, concurrency and synchronization. For these specifications, the conventional solution of having a single schedule for the different operations was not adequate. Thus, we developed a method that at execution time selected the best (with respect to a cost function) schedule from a pool of schedules, which were determined by external constraints.

The specification and the constraints were modelled by an algebraic model called control-flow expressions, and the design space was modelled by a finite-state representation. The dynamic scheduling and synchronization synthesis was then performed on this finite-state representation, and this synthesis problem was solved exactly with an 0-1 ILP formulation.

As future work, we are currently investigating how to reduce the number of variables to be solved by the ILP solver. One solution would the abstraction of the concurrent models with respect to the synchronization. This would reduce the number of states of each abstracted model, and would thus reduce the number of states of the finite-state machine representation for the control-flow expression of the full specification.

Acknowledgements

This research was sponsored by NSF/ARPA, under grant No. MIP 9115432. The first author was supported by the scholarship 200212/90.7 from CNPq/Brazil, and also by a fellowship from Fujitsu Laboratories of America.

References

[1] J. C. M. Baeten. *Process Algebra*. Cambridge University Press, 1990.

[2] Benchmarks of the 1992 high-level synthesis workshop.

[3] D. Filo, D. C. Ku, C. N. Coelho Jr., and G. De Micheli. Interface optimization for concurrent systems under timing constraints. *IEEE Transactions on VLSI Systems*, 1(3):268–281, September 1993.

[4] L. Hafer and A. Parker. Automated synthesis of digital hardware. *IEEE Transactions on CAD/ICAS*, c-31(2), February 1982.

[5] Y.-H. Hung and A. C. Parker. High-level synthesis with pin constraints for multiple-chip designs. In *Proceedings of the Design Automation Conference*, pages 231–234, June 1992.

[6] C.-T. Hwang, J.-H. Lee, and Y-C Hsu. A formal approach to the scheduling problem in high-level synthesis. *IEEE Transactions on CAD/ICAS*, 10(4):464–475, April 1991.

[7] C. N. Coelho Jr., D. Ku, and G. De Micheli. An algebra for modeling concurrent digital circuits. In *ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, 1993.

[8] C. N. Coelho Jr. and G. De Micheli. Analysis and synthesis of concurrent digital circuits using control-flow expressions. Technical report, Stanford University, 1994.

[9] T. Kim, J. W. S. Liu, and C. L. Liu. A scheduling algorithm for conditional resource sharing. In *Proceedings of the International Conference on Computer-Aided Design*, pages 84–87, Santa Clara, November 1991.

[10] R. Lipsett, C. Schaefer, and C. Ussery. *VHDL: Hardware Description and Design*. Kluwer Academic Publishers, 1989.

[11] P. Marwedel. Matching system and component behaviour in mimola synthesis tool. In *Proceedings of the European Design Automation Conference*, pages 146–156, March 1990.

[12] *PCI Local Bus Specification Revision 2.0*.

[13] I. Radivojević and F. Brewer. Symbolic techniques for optimal scheduling. In *Proceedings of the Synthesis and Simulation Meeting and International Interchange – SASIMI*, pages 145–154, Nara, Japan, October 1993.

[14] J.-K. Rho, G. D. Hachtel, F. Somenzi, and R. M. Jacoby. Exact and heuristic algorithms for the minimization of incompletely specified state machines. *IEEE Transactions on CAD/ICAS*, 13(2):167–177, February 1994.

[15] K. Wakabayashi and H. Tanaka. Global scheduling independent of control dependencies based on condition vectors. In *Proceedings of the Design Automation Conference*, pages 112–115, June 1992.