The Princeton University Behavioral Synthesis System

Wayne Wolf      Andrés Takach      Chun-Yao Huang      Richard Manno
Ephrem Wu
Department of Electrical Engineering Princeton University Princeton NJ 08544

Abstract

The Princeton University Behavioral Synthesis System (PUBSS) is a high-level synthesis system targeted to control-dominated machines. PUBSS accepts a VHDL subset, in which the design can be described as multiple communicating processes plus registers, and generates a register-transfer implementation. This paper describes the compiler with emphasis on the automaton-based algorithms which allow efficient specification and manipulation of control-dominated systems.

1 Introduction

The Princeton University Behavioral Synthesis System (PUBSS) is designed for the efficient specification and optimization of control-dominated architectures. PUBSS compiles a VHDL behavior model, written as multiple communicating processes and data registers, into an efficient register-transfer implementation. Control-dominated architectures are important because many application-specific ICs (ASICs) are primarily control [1] and because ASICs, with their intense time-to-market and design cost pressures, are ideal candidates for high-level synthesis.

The key to synthesis of control-dominated architectures is communication—control is often much easier to design as concurrent processes which communicate between each other. As an example, consider a multiple queue machine [2]—the machine accepts enqueue and dequeue commands for multiple queues whose data is stored in a shared memory. The natural architectural description of the machine has three processes: enqueue, dequeue, and memory management. Coordination of access to shared memory is arbitrated by a single machine, the memory manager; at other times, the enqueue and dequeue processes can run independently. Describing the system as one large process is much more difficult and error-prone.

Compiling concurrent architectures requires new ways to represent and manipulate scheduling information alongside control:

- The designer must be able to easily specify both the component machines in the architecture and the constraints on communication between them. Control-dominated architectures do not have a standard concurrent architecture—the architectural decomposition is highly problem-dependent. Previous work in high-level synthesis has concentrated on the compilation of large uniprocessors or, in the case of DSPs, on compilation into highly specialized processing networks.

- Control structures specified as multiple processes are harder to schedule because communication between the processes must be coordinated. While some communication may require handshaking, much interprocess communication in control structures can be statically scheduled—the times of operations can be determined during chip design. Previous work has concentrated on scheduling single processes.

PUBSS is based on the behavior finite-state machine (BFSM) model [3], which captures scheduling constraints in a finite-state model. Compiling VHDL into a network of communicating BFSMs gives us powerful algorithmic tools for manipulation of the design's control structure: events can be moved through control structures and systems built from communicating components can be combined and repartitioned.

The greatest progress in high-level synthesis has been made by developing compilation algorithms for a target domain and/or target architecture. The SAW system from CMU [4] and the ADAM system from USC [5] are both aimed at datapath-controller architectures, often found in CPUs and other data-intensive processors; many other datapath-controller synthesis systems have been built [6, 7, 8]. Successful synthesis systems for digital signal processing include the Cathedral systems [9] and a family of compilers built at GE [10].

Compared to datapath and DSP synthesis, relatively little is understood about the synthesis of control-dominated architectures. Existing scheduling techniques [8, 11] work on a single machine but are not easily extended to communicating machines. Several groups [12, 13] have identified specification and scheduling of distributed control as a critical research problem.

The Value Trace (VT) proposed by Snow [14] and formalized by McFarland [15], is the prototypical data/control flow graph representation. A number of design representations have since been developed [16, 17, 13, 18, 19]. Several projects have represented system control as a state transition graph [18, 19], but they have not used the state machine to represent scheduling constraints on the control operations themselves.
2 Behavior FSMs

The behavior FSM model embeds scheduling information in a finite-state model. PUBSS models a design's control structure as a set of communicating behavior FSMs. A BFSM is a state machine whose inputs and outputs are partially ordered in time; in contrast, a standard register-transfer machine's I/O behavior is totally ordered in time because we know its possible input and output values on every clock cycle. A BFSM's I/O behavior is specified as events whose times are constrained by linear inequalities. The inequalities can be represented by weighted, directed graph edges: edge direction shows the direction of the inequality and edge weight gives the minimum time between the events. In Figure 1, the machine will look for the input sequence \( i_1 = 0, i_1 = 1 \), with at least one cycle between the two events. The machine then produces the output event sequence \( o_1 = 0, o_2 = 0 \), with exactly one cycle separating the outputs.

As with register-transfer FSMs, we prefer to write the system behavior as a state transition graph. Figure 2 shows a state transition graph fragment for a BFSM. The conditions and outputs are specified as I/O event constraint graphs, consisting of I/O events and the linear constraints on their times. Most constraints are within a transition, but constraints across states are allowed. Execution of a transition may require more than one clock cycle: the transition to S3 requires exactly two cycles to recognize the input condition and at least two cycles to generate the output.

Multiple communicating machines are easily specified as a network of BFSMs. Scheduling constraints of one component in the network may affect the feasible schedules of other components in the network. Scheduling constraints of the components interact when the components must communicate. Such synchronous transfer of information is essential to avoid the overhead associated with handshake communication. By scheduling most communication to be synchronous we can specify or partition designs into smaller components without the overhead incurred by handshaking.

Both Probst [20] and Devadas and Keutzer [21] have proposed alternatives to the BFSM model. We find Probst's model cumbersome, and it has a less-direct relationship to register-transfer implementations than does the BFSM model. Devadas and Keutzer use a nondeterministic FSM to specify scheduling requirements. Nondeterminism is best at abstracting function but is more limited in the abstraction of time.

3 System modeling

Compilation starts with a behavior model written by the user in VHDL. The VHDL subset understood by PUBSS allows description of architectures as multiple communicating machines. That subset is mapped onto a reasonably unrestricted architecture of behavior FSMs and registers.

Figure 3 illustrates PUBSS's VHDL subset and the architectural schema onto which the VHDL is mapped. The VHDL model is described as one or more processes. Processes may communicate in either of two ways. Commu-
our generic VHDL model

```
-- signals connect processes
signal full_in, ..., data => '0';
-- registers store data
head : unsigned_register
generic map (Addrwidth)
port map (head_in, head_out, head_readwrite, head_ack);
-- processes describe behavior
main: process
begin
  wait on enq'transaction, ...;
  if reset = '1' then
    head_in <= unsigned(0, Addrwidth);
end main;
```

```
Figure 3: The VHDL subset and the target architecture.
```

```
Figure 4: The structure of PUBSS.
```

execution on general-purpose CPUs. Many statements are required for simulation which have no synthesis meaning: the wait on in the example, assignments to the unknown value, etc. The registers used in the descriptions are library components developed for PUBSS; simulation models for the various register types are kept in a separate library. The compiler is made more complex by these problems but the result is a description format natural to the VHDL programmer.

4 The compilation process
The sequence of operations presently performed by PUBSS is shown in Figure 4. Compilation separates the operations specified in the VHDL into a datapath, which holds the shared registers, plus sequencers.

- **vhdl2bfsm** generates a BFSM network description from the VHDL source. It performs simple source-level optimizations such as constant folding. It also algorithms based on the program dependence graph [22] to move statements to increase parallelism [23].

- **collapse** forms the product of the component BFSMs, transforming the architecture to have one master BFSM and any declared registers. Collapsing the component BFSMs together statically schedules direct communication between them.

- **schedule** finds a schedule for the system BFSM. It uses constraint-solving algorithms [24] to assign times to input and output events along paths in the behavior state transition graph.

- **vhdlgen** generates a register-transfer style VHDL description of the scheduled system.

The BFSM network model is critical to successful compilation of control. The state machine semantics of BFSM let us both analyze the communication between system.
Figure 5: An example of BFSM network collapsing.
components and change the system's partitioning. **Collapsing** a BFSM network, or forming the product of the network's components, is a key operation. Collapsing a network into a single BFSM lets us analyze the system's communication—generating the product machine computes all possible communications between the component BFSMs while eliminating combinations of actions that cannot occur.

Collapsing a BFSM network requires much more care than generating the product of a system of register-transfer FSMs. Register-transfer communication is implicitly synchronized, since all I/O is fully scheduled; in contrast, input and output events in communicating BFSMs must be paired together. Register-transfer transitions are also guaranteed to take exactly one cycle, while BFSM transitions may take zero or more cycles; as a result, one component BFSM may have completed a transition and moved to a state while another component is still in the middle of its transition's events.

The collapsing algorithm symbolically executes the components in unison to determine the system behavior. For any of the component BFSMs to change state, the component must receive the set of input events which trigger a transition. Input events of a component BFSM are matched with corresponding output events that are supplied either by the network environment, through the network's primary inputs, or by any of the other BFSMs, through the nets that connect the BFSMs. Each output event communicates a Boolean value to the input events that is matched with. Matching events are constrained to occur at the same time, i.e., they inherit each other's constraints.

Collapsing starts out with all component BFSMs in their behavior reset state. We let the environment to the network supply primary input events with events that exercise every possible transition of each of the component BFSMs. As soon as one of the component BFSM changes state we form a new composite state. Primary input/output events that where consumed/produced are kept in the product machine while the rest of the events are thrown away (their constraints are propagated back to primary IO events). We find all next states from the current state and proceed in breadth-first search fashion until all states in the network are found.

Figure 5 gives an example of BFSM product formation. A wire c connects one an output of B2 to an input of B1. The output events generated by B2 on wire c must be matched to input events expected by B1; constraints follow the matched events from one component to another.

BFSM network collapsing has a variety of uses during compilation. One important use is modular constraint specification [25], which lets us separate internal and external scheduling constraints. Most hardware description systems require the designer to fold constraints imposed by external interfaces into the description of the system under design. Using BFSM networks, the external system's interface is described as one or more BFSMs. The system under design deals with the external interface through a simplified protocol. The external constraints are imposed on the system under design by collapsing the constraint machine into the system machines. Collapsing uniformly handles both simple and data-dependent scheduling constraints.

Forming the product of BFSMs is also the key to scheduling systems specified internally as communicating machines. Scheduling communicating machines requires imposing the scheduling constraints from the communicating machines upon each other. Communication may be scheduled either dynamically or statically: communication times are static when they can be determined at compile time; communication scheduling is dynamic when the communication time is data-dependent and must be coordinated by handshaking during chip execution.

Though some inter-component communication must be dynamic, a majority of communication in typical designs can be scheduled statically. Statically-scheduled communication is faster because the data is transferred in a single cycle; dynamically-scheduled communication adds handshaking overhead. However, the scheduling algorithm must work from the constraints of both machines—the schedule is limited by the union of the components' constraints. Traditional scheduling methods are not designed to simultaneously schedule multiple components [12]; communication must be scheduled manually— the designer must manually determine when each component can send/receive data and add constraints to each component to enforce the communication times—or all communication must be performed by handshaking.

By collapsing all the component machines into a single BFSM, we can combine all the constraints which limit internal communication. Scheduling the system when it is modeled as a network of communicating machines would require constantly comparing candidate schedules to be sure that they do not violate any component's constraints; scheduling the system as a single large BFSM is conceptually simpler and easier to program. State explosion has not been a serious problem for two reasons: data is segregated from sequencers and kept in registers, which are not collapsed into the BFSMs; and since much timing information is kept as scheduling constraints, BFSMs have fewer states than their register-transfer implementations.

## 5 Conclusions

Control-dominated machines are an important domain for high-level synthesis. PUBSS introduces new methods for architectural optimization of control-dominated systems: modular constraint generation and joint scheduling of communicating components are both made possible by the BFSM network model used to represent the design.

### Acknowledgements

Miriam Leeser of Cornell University collaborated with us on the development of the BFSM model. Danwen Messenger of AT&T designed the original version of the multiple queue machine. This work was supported by the Semiconductor Research Corporation under contract 91-DJ-179 and by the National Science Foundation under contract MIP-9009960 (including a Research Opportunity for Undergraduates award).

### References


