A Model for Scheduling Protocol-Constrained Components and Environments

Steve Haynal Forrest Brewer
Department of Electrical and Computer Engineering
University of California, Santa Barbara, U.S.A.
haynal@umbra.ece.ucsb.edu, forrest@ece.ucsb.edu

ABSTRACT

This paper presents a technique for highly constrained event sequence scheduling. System resource protocols as well as an external interface protocol are described by non-deterministic finite automata (NFA). All valid schedules which adhere to interfacing constraints and resource bounds for flow graph described behavior are determined exactly. A model and scheduling results are presented for an extensive design example.

Keywords

Interface protocols, protocol-constrained scheduling, automata.

1. INTRODUCTION

Scheduling is an important problem occurring in diverse areas from manufacturing to networking to high-level synthesis of digital systems (HLS). Although there has been extensive work done in HLS scheduling, much of this work has disregarded how the final scheduled system must communicate to other systems. In particular, scheduling systems containing components with complex interface protocols is neglected. This situation is the norm in modern digital systems, and use of sequential protocols is likely to increase in future designs. This paper presents a technique which addresses data-flow scheduling subject to arbitrary sequential protocols. System resource usage protocols as well as an external interface protocol are described by non-deterministic finite automata (NFA). Next, constraints derived from a behavioral flow graph are applied to an implicit product NFA. Finally, reduced ordered binary decision diagram (ROBDD) symbolic reachability techniques are used to find all valid schedules exactly. A model and scheduling results are presented for an extensive design example.

We classify previous high level scheduling work into three categories: i) heuristic, ii) integer linear programming (ILP) and iii) symbolic methods. Heuristic schedulers (i.e [1][10]) find good solutions for large problems quickly but suffer with tightly constrained problems where early pruning decisions exclude candidates leading to superior solutions. ILP schedulers (i.e. [3][6]) exactly solve scheduling but have difficulties with time complexity and complex control constraint formulation. Symbolic methods (i.e. [2][4][7][8][11]) are often effective in finding exact solutions in highly constrained problem formulations but may suffer from representation explosion. The technique described in this paper falls in the symbolic methods category. The most closely related previous work is found in [2][11] where system timing and synchronization requirements are encapsulated in finite-state machine (FSM) descriptions. Our work differs in two ways. First, we introduce non-determinism as a preferred representation for protocols. The work described in [9] supports this decision. Second, and more importantly, our formulation is hierarchical and amenable to abstraction. We believe hierarchy and abstraction are key components in making symbolic techniques manageable.

2. PROBLEM DESCRIPTION

Input to this problem consists of three types of information. First, protocol interface NFAs are provided for all internal resource units and the external interface. For internal resource units, these automata models describe when local communication events may occur. Local communication is operand passing between local resource modules. External communication events are modeled by the external interface NFA. Second, a data flow graph (DFG) is provided. The behavior or algorithm to be implemented is given in terms of this graph. DFG nodes represent operations and arcs represent operands. In this case, nodes are executed by resources with potentially complex interface protocols. Finally, instance resource and operand register bounds are given.

The problem is to find a valid event sequence or schedule implementing the DFG described behavior, meeting all protocol constraints, and using only available resources. The technique presented here finds all valid schedules exactly.

3. PROBLEM FORMULATION

This section describes how the problem input information is used to construct a scheduling NFA which represents all valid execution sequences or schedules. The process involves building a product NFA from small local NFAs and applying constraints to this product NFA. ROBDDs provide efficient representation for these NFAs.

3.1 Operand NFAs

Each arc in the DFG is represented by an operand NFA (Fig. 1) in our formulation. The meanings of each state and transition may be inferred from the figure. The start state is in bold.

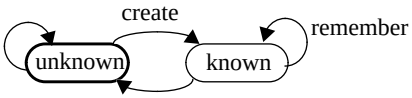

Figure 1. Operand NFA.

3.2 Resource NFAs

In our formulation, each node in the DFG is modeled by a resource NFA. Several instances of a given resource may be specified. The example resource NFA of Fig. 2 represents the protocol for a unit with a restricted bus. Two input operands are required but must be presented sequentially. Furthermore, input operands may not be accepted while an output operand is present.

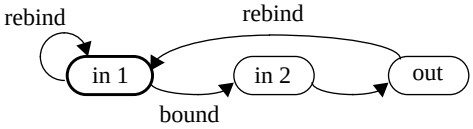

Figure 2. Example of Bus Restricted Resource NFA.

3.3 Binding NFAs

Several DFG nodes may be bound to the same resource instance. Consequently, it is necessary to distinguish *which* DFG node is bound to a resource instance at any given time. To make this distinction, a binding NFA is paired with each resource instance NFA. Fig. 3 shows a binding NFA capable of binding to two possible DFG nodes plus a null node. When this binding NFA is in a DFG node state, then the local operands of the mated resource instance NFA map directly to operands accepted or produced by that DFG node. Which local operand maps to which DFG operand is specified by the designer. Finally, constraints are added later to restrict a binding NFA's rebind transitions (a change in state) to occur only in sync with its mated resource NFA's rebind transitions.

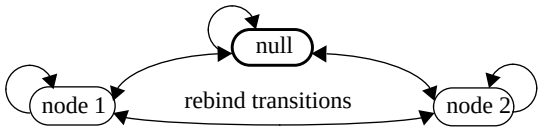

Figure 3. Binding NFA.

3.4 Interface Protocol NFA

The interface protocol NFA describes when DFG operand input or output transactions may occur given external timing constraints. It is similar in construction to a resource NFA (Fig. 2) with states associated with input and output events. Unlike a resource NFA, there is no need for a mated binding NFA.

3.5 Implication Constraints

To create a scheduling NFA, implication constraints are applied between operand, resource, binding and interface protocol NFAs. For example, let *Q* be the proposition, "The interface protocol NFA is transiting to a state where input of operand 1 is allowed" and let *P* be the proposition, "The operand NFA for operand 1 has a create transition." The desired implication would then be $P \rightarrow Q$ or if *P* is true, then *Q* must also be true. In the ROBDD structure, an implication is constructed as,

$$P \rightarrow Q = \overline{P}\overline{\overline{Q}} \quad \text{where } P \text{ and } Q \text{ are ROBDDs.} \tag{1}$$

3.5.1 Operand Create Implications

DFG operands are only allowed to be created (their NFA transits from unknown to known) when they are available from the interface protocol or a bound resource instance NFA. For each operand *i* expected from the protocol interface, the implication is,

$$i_{create} \rightarrow interface_{i_{ns}} \tag{2}$$

where i_{create} is the create transition of the expected operand NFA and $interface_i$ is any transition to a next state, *ns*, where operand *i* is available in the interface protocol.

A DFG operand *i* is available from a resource instance, *r*, when two conditions are true. First, *r*'s paired binding NFA, *b*, must be transiting to a next state bound to the DFG node, *nd*, producing *i*. Second, suppose *l* is a local operand of *r* which maps to DFG operand *i* when the first condition is true. Then *r* must also be transiting to a next state where local operand *l* is available. Furthermore, since any resource instance capable of producing DFG operand *i* may actually produce *i*, all capable resource instances must be examined. Formally, for each expected DFG operand *i*, this is described as,

$$i_{create} \rightarrow \sum (r_{l_{ns}} b_{nd_{ns}}) \tag{3}$$

where the summation is over all capable resource instance and mated binding NFAs.

3.5.2 Operand Accept Implications

Resource and interface protocol NFAs are only allowed to transit to states requiring operands if the required operands will exist. For the interface protocol, this is enforced for each required DFG operand *i* with,

$$interface_{i_{ns}} \rightarrow i_{known_{ns}} \tag{4}$$

where $interface_i$ is any transition to a next state requiring operand *i* and i_{known} is any transition to the known state of operand *i*.

The implication describing this for a local resource and mated binding NFA for each required DFG operand *i* is,

$$r_{l_{ns}} b_{nd_{ps}} \rightarrow i_{known_{ns}} \tag{5}$$

where r_l, b_{nd} and i_{known} are as described earlier. By writing this implication in terms of the present state, *ps*, of the binding NFA, it is possible to create a resource NFA which produces an output operand and rebinds to a new DFG node during the same cycle.

3.6 Rebind Synchronization Constraints

A binding NFA may only rebind to a new DFG node when its mated resource NFA transits through a rebind transition. This synchronization is enforced for each resource *r* and binding *b* NFA pair with the constraint,

$$r_{rebind} b_{rebind} \tag{6}$$

where r_{rebind} and b_{rebind} are the rebind transitions of the resource and mated binding NFAs respectively. Furthermore, it is wasteful for the resource NFA to transit through a bound transition with a null binding. Consequently, for each resource *r* and binding *b* NFA pair, the constraint,

$$r_{bound} \rightarrow \overline{b_{null_{ns}}} \tag{7}$$

where r_{bound} is the bound transition of the resource NFA (Fig. 2) and b_{null} is any transition to a next state of null in the mated binding NFA, is applied to the scheduling NFA.

3.7 Memory Constraints

Only a finite number, *n*, of storage elements may be available to store DFG operands. Let **A** be the set of all combinations of at most *n* operands from the set of all DFG operands. Then

$$\sum_{\alpha \in A} \left(\prod_{j \in \alpha} j_{remember} \right) \quad \text{is allowed.} \quad (8)$$

This constraint essentially limits the number of operand remember transitions which may coexist in any transition of the scheduling NFA.

4. SCHEDULING SOLUTIONS

The product of all local NFAs and constraints described in Section 3 form a scheduling NFA. Every possible valid schedule is a path in this scheduling NFA from a starting state set $S_i(V)$, where no operands are known and all resources are null bound, to a termination state set $S_f(V')$, where all desired operands have existed. Each shortest path from $S_i(V)$ to $S_f(V')$ represents a minimum latency schedule.

We leverage symbolic reachable state analysis techniques to determine the existence of valid schedules. Let the scheduling NFA be defined by the four-tuple (V, δ, $S_i(V)$, $S_f(V')$) where V is the finite, non-empty set of states, $\delta: V \rightarrow V'$ is the next-state function and $S_i(V)$ and $S_f(V')$ are sets of initial and final states respectively. Starting with $S_i(V)$, reachable state analysis is performed. Once completed, if $S_f(V')$ is not present in the reachable state set, then no schedules are possible with the current constraints and scheduling terminates. On the other hand, if $S_f(V')$ is present, then valid schedules do exist and we can use the technique described in [4] to find a shortest path and hence a minimum latency schedule. Finally, we are not bound to perform complete reachable state analysis but may use refinements, optimizations and other techniques to find any desired subset of paths or schedules in the scheduling NFA.

5. 2-POINT DFT EXAMPLE

We develop a 2-point DFT example in detail to demonstrate the versatility of our protocol-based scheduler. Fig. 4 shows the DFG used in this example. Although this DFG appears simple enough to be handled by traditional scheduling techniques, the advantage of our method is the ability to tightly define data transfer protocols and resource constraints.

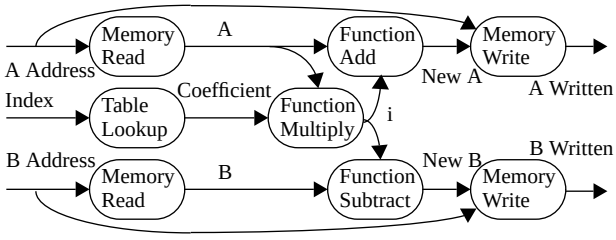

Figure 4. 2-Point DFT Example DFG.

Fig. 5 shows the interface protocol constraint NFA for this example. In reality, this constraint describes an external controller which computes correct indices and memory addresses. Due to controller and communication bandwidth limitations, the index, A's memory address and B's memory address must be passed in three consecutive cycles. After this, the controller non-deterministically does not proceed to the next iteration until it knows that both computed terms have been successfully written to memory.

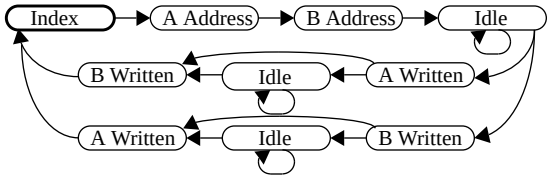

Figure 5. DFT Interface Protocol.

The table lookup protocol is shown in Fig. 6. Two cycles after an index is presented, a stored coefficient is produced. A unique behavior is that the coefficient remains available for two cycles. Furthermore, a new index may be provided during the second cycle of coefficient availability.

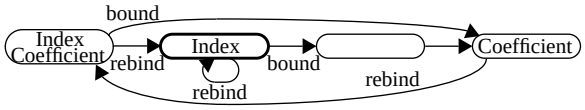

Figure 6. DFT Table Lookup Resource.

The memory resource uses the protocol detailed in Fig. 7. The data and address busses are time multiplexed with addresses accepted on odd cycles and data passed on even cycles or vice versa. The read protocol requires an address and provides the requested data after three cycles. The write protocol requires an address and the write data in two consecutive cycles. Three cycles later a write acknowledge is produced. A new address may be accepted during the same cycle that a write acknowledge is produced.

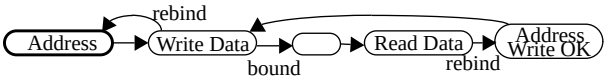

Figure 7. DFT Memory Resource.

Fig. 8 shows the arithmetic processor protocol. This unit performs three floating-point operations: addition, subtraction and multiplication. Due to limited communication bandwidth, input operands 1 and 2 and the result operand must be passed during separate cycles. An add or subtract result is produced two cycles after the last input. Given the higher complexity of multiplication, its result is produced three cycles after the last input. For addition and multiplication, ordering of the input operands is irrelevant but for subtraction operand ordering is important. In this example, operand 1 must be accepted first.

The protocol in Fig. 8 is an example of alternative behaviors. There are two valid start states and three variations of correct behavior. A flexibility of our formulation is this ability to handle numerous alternatives. As long as one valid path exists containing all required input and output operands for a DFG node, symbolic exploration will not fail. Additional resource operands not required by the DFG are ignored.

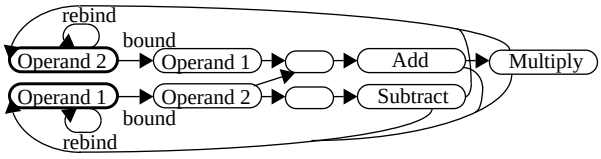

Figure 8. Arithmetic Processor Resource.

6. RESULTS

A tool was developed to demonstrate the feasibility of our scheduling technique. It was written in python and utilized a standard BDD library. The reported results were produced on a 400 MHz Pentium PII system running Linux with 512MB of memory. These results were duplicated with runtimes 3 to 4 times longer on a 166 MHz Pentium laptop Linux system with 32MB of memory. Table 1 presents results for various resource configurations of the example presented in Section 5. The first three columns list instances of these available resources. Since by observation, A

Address and B Address from the DFG in Fig. 4, are needed at both the beginning and end, these operands are given dedicated storage which is not included in the *Memory Registers* column.

As one might expect, scheduling performance is best for configurations with the least amount of freedom. Increasing instances of any resource causes a resulting increase in CPU time. All solutions are exact and produced in reasonable time. Although a minimum latency schedule is reported, all valid schedules of all lengths were actually computed. As far as we know, we are the first to report exact solutions for protocol-constrained scheduling problems of this type.

TABLE 1: 2-Point DFT Results

Memory Registers	*Memory Ports*	*Arithmetic Processors*	*Minimum Latency Schedule*	*CPU Time in Seconds*
1	1	1	26	2.2
2	1	1	20	6.2
3	1	1	20	8.2
2	2	1	15	11.9
2	1	2	18	54.5
3	2	1	15	20.0
3	1	2	18	50.4
no limit	2	2	13	15.7
2	1	2*	18	164.0
2	2	1*	15	42.4
2	2	1*	15*	54.1

Arithmetic processors marked with an asterisk use a slightly different protocol than described in Fig. 8. First, the multiply extra cycle penalty is removed. Second, a new penalty for reconfiguration is added. Any multiplication following an add or subtract or any add or subtract following a multiplication pays an extra cycle penalty for reconfiguration. In these cases we see the same results for a minimum latency schedule but with a possibly simpler arithmetic processor and control structure.

The minimum latency schedule marked with an asterisk uses a modified interface protocol which is intended to explore alternative controllers. A address, B address and the index may now be produced by the controller in any consecutive order. Although this added freedom increases the runtime by 12 seconds, there is no gain in the minimum latency schedule.

7. FUTURE WORK

The models used in this paper were chosen with great care to be amenable to future work with scheduling hierarchy and abstraction. A scheduled external interface NFA and a resource NFA have interchangeable meanings. This allows for a general protocol NFA to be a vehicle for refinement and abstraction in a hierarchy. This protocol NFA can be a resource instance NFA at one level of hierarchy or an external interface NFA at another level. With a bottom up design flow through the hierarchy, internal complexity of lower levels is hidden from higher levels since only external communication events are propagated up. With a top down design flow, local freedom of lower levels is restricted by the protocol NFA of the higher level. The entire synthesis and scheduling process involves refining all protocol NFAs through repeated constraint propagation when coexecuting protocols at adjacent hierarchy levels. We believe that such a hierarchical model is necessary when synthesizing and scheduling systems of meaningful scale.

Although simple looping structures were present in the example, our future work will address loops in a more general way. The method described in [8] provides a starting point. Furthermore, control structures were not directly addressed in this short paper. Our previous work in [4] demonstrates a possible way of adding control. Finally, our use of symbolic reachability to determine valid schedules will be refined by related work in symbolic traversal techniques for verification.

8. CONCLUSIONS

This paper presented a model and technique for representing all valid schedules of a data flow graph mapped to a protocol-intensive environment. Both an external interface protocol as well as internal resource protocol constraints were adhered to. All valid schedules were modeled exactly using a ROBDD NFA composed of local smaller protocol NFAs and additional constraints applied between local NFAs. An extensive design example with results showed the versatility of this technique.

9. REFERENCES

[1] R. Camposano, “Path-Based Scheduling for Synthesis”, *IEEE Trans. CAD/ICAS*, vol. 10, no. 1, pp. 85-93, Jan. 1991.

[2] C. N. Coelho Jr, G. De Micheli, “Dynamic Scheduling and Synchronization Synthesis of Concurrent Digital Systems under System-Level Constraints”, *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 175-181, 1994.

[3] C. H. Gebotys and M. I. Elmasry, “Global Optimization Approach for Architectural Synthesis”, *IEEE Trans. CAD/ICAS*, vol. 12, no. 9, pp. 1266-1278, Sep. 1993.

[4] S. Haynal and F. Brewer, “Efficient Encoding for Exact Symbolic Automata-Based Scheduling”, *Proc. IEEE Int. Conf. Computer-Aided Design*, to appear, 1998.

[5] H. Hulgaard S.M. Burns, T. Amon, G. Borriello, “An Algorithm for Exact Bounds on the Time Separation of Events in Concurrent Systems”, *IEEE Transactions on Computers*, vol. 44, no.11, pp. 1306-1317, Nov. 1995.

[6] C.-T. Hwang and Y.-C. Hsu, “A Formal Approach to the Scheduling Problem in High Level Synthesis”, *IEEE Trans. CAD/ICAS*, vol. 10, no. 4, pp. 464-475, Apr. 1991.

[7] C. Monahan and F. Brewer, “Scheduling and Binding Bounds for RT-Level Symbolic Execution”, *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 230-235, 1997.

[8] I. Radivojevic and F. Brewer, “A New Symbolic Technique for Control-Dependent Scheduling”, *IEEE Trans. CAD/ICAS*, vol. 15, no. 1, pp. 45-57, Jan. 1996.

[9] A. Seawright and F. Brewer, “Clairvoyant: A Synthesis System for Production-Based Specification”, *Proc. IEEE Trans. on VLSI Systems*, vol. 2, no. 2, pp. 172-185, June 1994.

[10] K. Wakabayashi and H. Tanaka, “Global Scheduling Independent of Control Dependencies Based on Condition Vectors”, *Proc. 29th ACM/IEEE Design Automation Conf.*, pp. 112-115, 1992.

[11] J. C.-Y. Yang, G. De Micheli, and M. Damiani, “Scheduling and Control Generation with Environmental Constraints based on Automata Representations”, *IEEE Trans. CAD/ICAS*, vol. 15, no. 2, pp. 166-183, Feb. 1996.