Abstraction Techniques for Verification of Multiple Tightly Coupled Counters, Registers and Comparators *

Yee-Wing Hsieh
Department of Electrical Engineering
University of Pittsburgh
hsieh@ee.pitt.edu

Steven P. Levitan
Department of Electrical Engineering
University of Pittsburgh
steve@ee.pitt.edu

Abstract

We present new non-deterministic finite state machine (NFSM) abstraction techniques for comparators based on the comparison difference of the two operands (e.g., counters) instead of the comparison order. One of the major advantages of the comparison difference abstractions is the ability to model the comparison of multiple tightly coupled counters. The abstraction techniques are integral to our semantic model abstraction methodology, where abstract models are generated based on semantic matching of behavioral VHDL models with known abstraction templates. Using NFSM models for counters, comparators, and registers, we have shown our approach can yield many orders of magnitude ($10^2 - 10^{11}$) reductions in state space size and substantial improvements in performance of formal verification runs.

1 Introduction

The traditional synthesis based technique in formal verification is to take a behavioral (e.g., VHDL) model, synthesize the design using this behavioral model, and then verify the resulting synthesis implementation of that design using various formal verification engines such as COSPAN [1], SMV [2], or Vis [3]. However, using synthesis based techniques, it is difficult to perform abstraction because the behavioral semantics of the model are lost at the RTL level.

A better approach is to verify designs using an abstract state space. Our Semantics-based Model Abstraction method, called SMA, extracts portions of the VHDL models whose semantic behavior matches known abstraction templates and replaces those portions of the model with functionally equivalent non-deterministic finite state machines (NFSM) [5]. We then verify system properties using the resulting abstract model. In our approach, semantic extraction identifies the abstract state space for model abstraction, instead of exploring the state space of the original, unreduced model.

2 Methodology

The main principle behind our model abstraction method consists of two tasks: (1) extracting portions of the VHDL models whose semantic behavior matches known abstraction templates, and (2) replacing those portions of the model with functionally equivalent non-deterministic finite state machines (NFSM).

To extract semantic attributes of a model for abstraction template matching, we first perform control/dataflow analysis on the VHDL model to determine signal/variable dependencies of assignments and expressions [4]. With the results from dependency analysis, we then identify semantic attributes from the model that match the semantic attributes of our abstraction templates. Each abstraction template has a corresponding class of NFSM abstraction that models the behaviors of those semantic attributes in the template. The number of states in the NFSM abstraction is determined by abstract data type consisting of non-overlapping *key values* and/or *symbolic values* extracted from the behavioral VHDL model for each signal/variable [5].

For counters, each key value in the abstract data type set is a deterministic state in the NFSM while each symbolic value is a non-deterministic state in the NFSM covering a range of values. Transitions are determined by the class of the abstract template matched, and hence, they are determined by the semantic attributes extracted from the model.

The general NFSM template for an up counter with wrap around is shown in Figure 1(a) and the general template for an up/down counter without wrap around is shown in Figure 1(b). In both cases, each solid circle/arrow is a deterministic state/transition, while each dashed circle/arrow is a non-deterministic state/transition. The cost of each counter abstraction is a non-deterministic variable, *ND_K*, which is assigned a non-deterministic value of {1,2} representing the number of clock cycles 1, or more than 1, away from the next deterministic state.

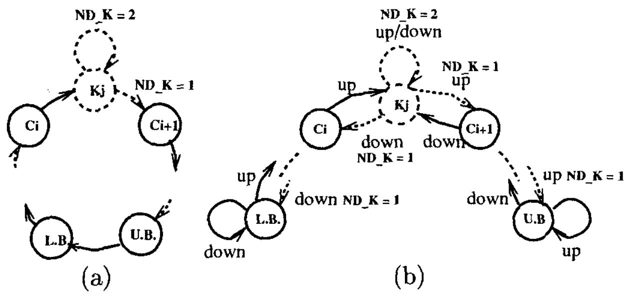

Figure 1: NFSM Template for (a) an Up Counter with Wrap Around (b) an Up/Down Counter without Wrap Around

A comparator by itself is a combinational circuit and thus does not contain any states. However, if one input or both inputs to the comparator is a register or counter, then the output of the comparator behaves like a state machine. We can model this behavior using a NFSM abstraction that maintains comparison order relations.

The general NFSM template for a non-wrap around up/down counter being compared to a register is shown in Figure 2a. The cost of this abstraction is a non-deterministic variable, *ND_D*, which is assigned a non-deterministic value of {1,2}, representing the distance from the current non-deterministic state (<) or (>) to the deterministic state (=) is 1 or more than 1.

The general NFSM template for two non-wrap around up/down counters being compared is shown in Figure 2b. The cost of this abstraction is a non-deterministic variable, *ND_D*, which is assigned a non-deterministic value of {1,2,3}, representing the distance from the current non-deterministic state (<) or (>) to the deterministic state (=) or to the non-deterministic states (<) or (>) is 1, 2 or more than 2.

For counters with the semantic attributes of loading, each comparator abstraction has an additional non-deterministic variable, *ND_COMP*, which specifies a non-deterministic comparison order when a counter loads the same symbolic value as the one to which it is being compared.

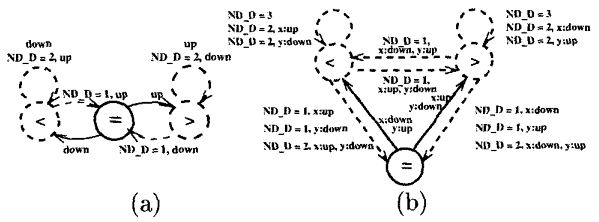

Figure 2: Comparison Order Abstraction Template for (a) an Up/Down Counter Being Compared with a Register (b) Two Up/Down Counters Being Compared

The comparison order abstraction presented above can model interactions between two tightly coupled counters. However, it can not model interactions between three or more tightly coupled counters due to the next state transitions depending on the non-deterministic variables, ND_D_i.

To illustrate this problem, let us examine three up/down counters, — x, y and z — being compared. Using the comparator abstraction technique presented earlier, the comparator abstractions for the three counters being compared with each other consists of the three interacting NFSMs shown in Figure 3. In this case, the comparison pairs (x comp y), (y comp z) and (x comp z) depend on non-deterministic variables ND_D_{xy}, ND_D_{yz} and ND_D_{xz}, respectively. As we recall, ND_D_i is assigned a non-deterministic value representing the distance between a counter and a register or two counters. In this case, ND_D_i is assigned a non-deterministic value 1, 2 or 3, representing the distance of 1, 2, or more than 2. Let us assume for a moment that x is greater than y, and y is greater than z which implies that x is greater than z by at least 2. In the comparator abstractions, all 3 NFSMs are in the (>) state and the next state transition for each of the 3 NFSMs depends on whether counters x, y and z count up, down or hold, and the value of the respective non-deterministic variables. This implies that there could be 27 (i.e., 3 * 3 * 3) possible next state outcomes among the 3 NFSMs. However, there really only 9 valid behaviors of three counters being compared with each other. For example, we expect (x comp z) to make the transition from the (>) state to the (<) state since x is greater than z by at least 2. However, this invalid transition does exist in the comparator abstraction. Consequently, the comparator abstraction technique presented earlier can not handle tightly coupled interactions among three counters, because the NFSM abstractions it generates contains behaviors not in the original model.

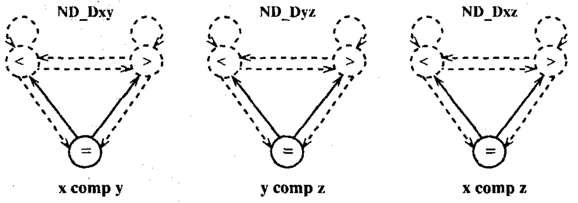

Figure 3: Abstraction for Three Tightly Coupled Comparators

We can resolve this problem by inserting additional deterministic states in the comparator abstractions to remove next state comparison relations' dependency on the non-deterministic variables. The resulting comparator abstraction is shown in Figure 4. In this case, deterministic states (< 0) and (< 1) represent x less than y by 1 and by 2, respectively, while non-deterministic state (< 2) represents x less than y by more than 2. Similarly, deterministic states (> 0) and (> 1) represent x greater than y by 1 and by 2, respectively, while non-deterministic state (> 2) represents x greater than y by more than 2. It is important to note that the transition from the non-deterministic states does not change the comparison result in any of the comparison pairs, which implies that the comparison result does not depend on the non-deterministic variables ND_D_i. Therefore, the relative order behavior among x, y and z is preserved.

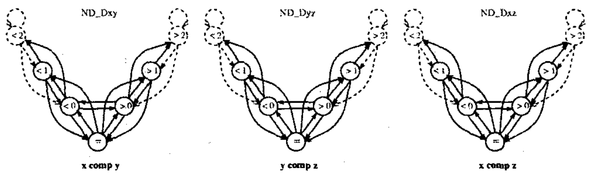

Figure 4: Modified Abstraction for Three Tightly Coupled Comparators

We can look at these extended comparator abstractions in a different way: as comparison difference instead of comparison order and derive the comparison order indirectly.

The abstraction template for the comparison difference between register-counter pairs is a 3-state NFSM shown in Figure 5a. In this case 0 represents deterministic differences ($x - y = 0$). while -2 and 2 represent non-deterministic differences ($x - y < 0$) and ($x - y > 0$). Variable *ND_D* is assigned a non-deterministic value of {1,2} which specifies whether the distance between x and y is 1, or more than 1. Variable *ND_DIFF* is assigned a non-deterministic value of {-1,0,1} when both counter x and counter y load the same symbolic value.

The abstraction template for the comparison difference between counter-counter pairs is a 5-state NFSM shown in Figure 5b. In this case -1, 0, and 1 represent deterministic differences ($x - y = -1$), ($x - y = 0$), and ($x - y = 1$). while -2 and 2 represent non-deterministic differences ($x - y < -1$) and ($x - y > 1$). Variable *ND_D* is assigned a non-deterministic value of {2,3,4} which specifies whether the distance between x and y is 2, 3, or more than 3. Variable *ND_DIFF* is assigned a non-deterministic value of {-2,-1,0,1,2} when both counter x and counter y load the same symbolic value.

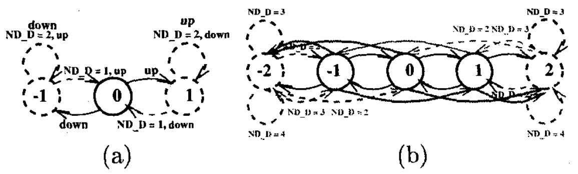

Figure 5: Comparison Difference Abstraction Template for (a) an Up/Down Counter Being Compared with a Register (b) Two Up/Down Counters Being Compared

Comparison relations among n tightly coupled counters can be modeled by maintaining an n-clique of pair-wise comparison difference relations among n counters without adding more states to the difference abstraction. For example, if three counters x, y, and z are compared with each other, we maintain a 3-clique of pair-wise comparison difference pairs (x - y), (y - z), and (x - z) to model all possible interactions between x, y, and z.

Replacing all the abstractions into the original model, we use COSPAN to verify the system properties using the resulting abstract model.

3 Experimental Results

In this section, we present our model abstraction experiments on five designs: a car seat controller, a good and a faulty version of robot grip controllers, and a good and a faulty version of maintain-comparison-order controllers. For these five designs, two sets of abstract models were generated using the two semantics-based model abstraction methods presented in this paper. One set of abstract models was produced using the SMA-ORD method (i.e., SMA method with the comparison order comparator abstractions), and another set was produced using the SMA-DIFF method (i.e., SMA method with the comparison difference comparator abstractions).

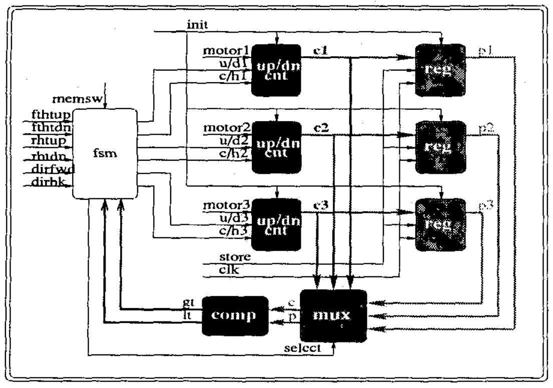

Figure 6: Car Seat Controller Block Diagram

The block diagram of the car seat controller is shown in Figure 6. The system property we want to verify is *the restoration of the car seat from any current position to the last stored position.*

For the car seat controller, both SMA-ORD and SMA-DIFF methods performed the same abstractions for the three 8-bit counters and the three 8-bit registers. However, different 3-state NFSM comparator abstractions were generated by the two abstraction methods for each of three counter-register comparison pairs.

The abstraction for each of the three up/down counters (that keeps track of the current seat position), contains three key values $\{0,k,255\}$, where the non-deterministic state k covers values from 1 to 254. The abstraction contains a NFSM (shown in Figure 7) that covers the entire range of seat movement.

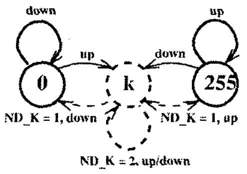

Figure 7: NFSM for Car Seat Up/Down Counter

Using the SMA-ORD method, the abstraction for each of the three counter-register comparison pairs is partitioned into three comparison relations: $(c = p)$, $(c < p)$, and $(c > p)$. The comparison order abstraction contains a 3-state NFSM, similar to the one shown in Figure 2a. On the other hand, using the SMA-DIFF method, the difference abstraction for each of the three counter-register comparison pairs is a 3-state NFSM similar to the one shown in Figure 5a. In this case, the comparison results are derived from the three difference abstractions.

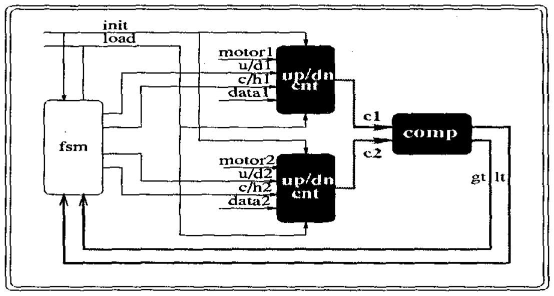

Figure 8: Robot Grip Controller Block Diagram

The block diagram of the robot grip controllers is shown in Figure 8. For the robot grip controllers the system property we want to verify is *grip object from any left grip and right grip positions.*

The two robot grip controllers, A and B, are exactly the same except for the FSMs. Controller A conforms to design specifications and can grip objects placed between any two grip positions. On the other hand, a bug in the FSM of the controller B causes overshoot of the grip movement and fails to grip objects in some grip positions.

The two robot-grip controllers contain two tightly coupled 8-bit counters being compared to each other. For the two counters, both the SMA-ORD and the SMA-DIFF methods performed the same abstractions.

The abstraction for each of the two up/down counters (that keeps track of the current left/right grip position), contains five key values $\{0,k_1,31,k_2,63\}$, where the non-deterministic state k_1 covers values from 1 to 30 while the non-deterministic state k_2 covers values from 32 to 62. The abstraction contains a NFSM (shown in Figure 9) that covers the entire range of grip movement.

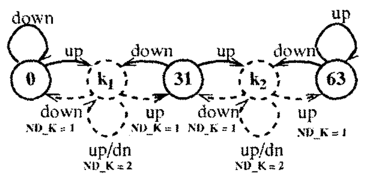

Figure 9: NFSM for Robot Controller Up/Down Counter

In this case, for the counter-counter comparison pair, the SMA-ORD method generated a 3-state NFSM abstraction similar to the one shown in Figure 2b, while the SMA-DIFF method generated a 5-state NFSM abstraction similar to the one shown in Figure 5b.

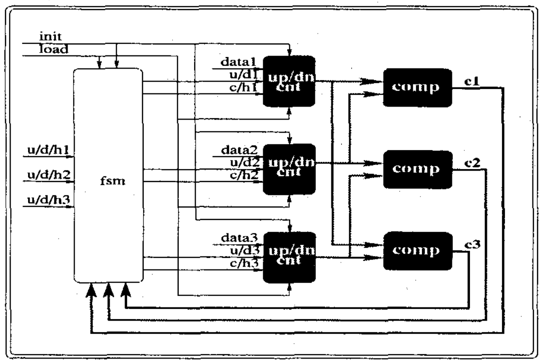

Figure 10: Maintain Comparison order Controller Block Diagram

The block diagram of the maintain-comparison-order controllers is shown in Figure 10. The system property we want to verify is *all invalid counter input combinations are rejected so that the initial comparison order is maintained.*

The two maintain-comparison-order controllers contain three tightly coupled counters being compared to each other. Using the SMA-DIFF method, abstractions were performed on the three 5-bit counters and a 3-clique of pair-wise comparison difference abstractions among the three counters. The abstraction for each of the three counters is a 3-state NFSM shown in figure 11. The abstraction for each of the three counter-counter comparison pairs is a NFSM abstraction similar to the one shown in Figure 5b. In this case, the SMA-ORD method does not work because the two controllers contain more than two tightly coupled counters being compared.

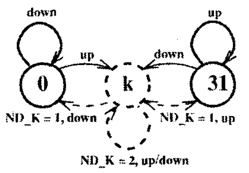

Figure 11: NFSM for Maintain-Comparison-Order Controller Up/Down Counter

To demonstrate the effectiveness of our model abstraction method, we compare the performance of formal verification runs on the five designs using the formal verification tool COSPAN. As a basis for comparison, we first verify the system properties using the original models. Next, we compare the performance of formal verification runs using three sets of abstract models. One set of abstract models is reduced by COSPAN's property-dependent *localization reduction* (LR) method, The other two sets are reduced by SMA-ORD and SMA-DIFF methods, respectively. Thus, our model abstraction experiments consists of four sets of formal verifications runs: Original, LR, SMA-ORD, SMA-DIFF. The results of the verification runs are summarized in Tables 4a and 4b. The table is split for ease of reading.

Table 4a shows the cost of verifying the system properties of the four designs in terms of state space size and the number of non-deterministic selections (i.e., verification overhead introduced by non-deterministic variables). In contrast, Table 4b shows the performance of the same verification runs in terms of the number of states searched and the CPU time.

From the two tables, the SMA-ORD and the SMA-DIFF models for all the designs have a state space size many orders of magnitude ($10^2 - 10^{11}$) smaller when compared to the original unreduced model. Furthermore, in all the examples, the verification performance is consistently improved with 241 to 255 fold reduction in CPU times using the SMA-ORD models than COSPAN's LR models.

For the car seat controller, the SMA-DIFF models and the SMA-ORD models have the same state space size and require the same number of non-deterministic selections. However, due to the structural differences in the comparator abstractions, the number of states searched is 3.7 fold smaller and the CPU time is 4.1 fold smaller using the SMA-DIFF models than SMA-ORD models.

For the two robot grip controllers, the SMA-DIFF models have a 1.6 fold larger state space size and a 1.7 fold larger number of non-deterministic selections than the SMA-ORD models. As a result, the SMA-DIFF models have 1.1 to 1.3 fold larger number of states searched and 2.8 to 3.3 fold larger CPU times than the SMA-ORD models. This example illustrates the trade-off between state space size and non-deterministic selection in the two abstraction methods.

For the two maintain-comparison-order controllers, the SMA-DIFF models have a state space size that is 62.2 fold smaller and have a CPU time that is 95.9 fold smaller than the original unreduced model.

It is important to note that using the SMA-ORD and SMA-DIFF models, every verification run completed or produced correct counter-examples to identify faults in the designs in a short period of computation time. However, using the original unreduced models, some examples terminated prematurely after running out of swap space.

Property Verified	State Space Size†				ND-Selection			
	Original	LR	SMA-ORD	SMA-DIFF	Original	LR	SMA-ORD	SMA-DIFF
Car Seat Controller								
Mem Pos	1.08e+18	1.08e+18	7.56e+07	7.56e+07	108	108	3456	3456
Robot Grip Controller (Good)								
Grip Obj	1.97e+06	1.97e+06	2.59e+04	1.60e+05	1024	1024	900	1500
Robot Grip Controller (Faulty)								
Grip Obj	2.10e+07	1.05e+07	9.60e+04	1.60e+05	4096	4096	900	1500
Maintain Comparison Order (Good)								
Kept Ord	3.36e+08	3.36e+08	n/a	3.47e+07	884736	884736	n/a	729000
Maintain Comparison Order (Faulty)								
Kept Ord	3.36e+08	3.36e+08	n/a	3.47e+07	884736	884736	n/a	729000

† Verification runs performed using 70MHz SUN SPARC5 with 128MB main memory and 736MB swap space.
() Verification run terminated due to insufficient swap space.

Table 4a: Verification Cost

Property Verified	States Searched				CPU Time† (seconds)			
	Original	LR	SMA-ORD	SMA-DIFF	Original	LR	SMA-ORD	SMA-DIFF
Car Seat Controller								
Mem Pos	(27839126)	(16353651)	89741	23978	(2932316)	(1824030)	177306	42349
Robot Grip Controller (Good)								
Grip Obj	44768	43152	334	374	20823	17069	67	192
Robot Grip Controller (Faulty)								
Grip Obj	24603	24357	203	261	10845	9891	41	134
Maintain Comparison Order (Good)								
Kept Ord	327702		n/a	1372	2771530	2771970	n/a	28904
Maintain Comparison Order (Faulty)								
Kept Ord	262169		n/a	1109	2191080	2191282	n/a	23234

† State space size includes both the system model and the environment model.

Table 4b: Verification Performance

4 Conclusions

In this paper, we presented new NFSM abstraction techniques for comparators based on comparison difference instead of comparison order. Using comparison difference abstractions, we can model multiple tightly coupled counters being compared. Using NFSM models for counters, comparators, and registers, we have shown our approach can yield many orders of magnitude (10^2 – 10^{11}) reductions in state space size and substantial improvements in performance of formal verification runs.

The abstraction algorithm can be extended for counters that increment or decrement by values other than one with some caveats for non-modulo N counters. Furthermore, performance optimizations such as reducing non-deterministic overhead by enumerating symbolic constants or merging mutually exclusive non-deterministic variables can also be integrated into the abstraction algorithm.

Acknowledgments The authors would like to thank Dr. Robert Kurshan of Lucent Technologies at Bell Laboratories for providing us with the COSPAN formal verification tool.

References

[1] Z. Har'El and R. P. Kurshan, *COSPAN User's Guide.* AT&T Bell Laboratories, February 1993.

[2] K. L. McMillan, *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

[3] R. K. Brayton, et al., "Vis: A system for verification and synthesis," in *Proc. of Conference on Computer-Aided Verification*, pp. 428–432, July 1996.

[4] Y.-W. Hsieh and S. P. Levitan, "Control/data-flow analysis for vhdl semantic extraction," *Journal of Information Science and Engineering*, vol. 14, pp. 547–565, September 1998.

[5] Y.-W. Hsieh and S. P. Levitan, "Nfsm generation for semantics based model abstraction," in *4th IEEE International High Level Design Validation and Test Workshop*, pp. 138–145, November 1999.

[6] Y.-W. Hsieh, *Model Abstraction via Semantic Extraction of Behavioral VHDL Descriptions for Formal Verification.* PhD thesis, Dept. of Electrical Engineering, University of Pittsburgh, 2000.