Figure 1: FSM extraction applications.

Boolean value *a*, then node q must be initialized to $\overline{a}$ to avoid unit delay oscillation. Furthermore, the initialization of state nodes and their related nodes has to be done during the appropriate clock phases. This makes this technique very tedious and unsuitable for automation.

Kam and Subrahmanyam [12] use Anamos [4] to produce a set of network excitation functions for a switch-level network. These excitation functions are converted to a symbolic transition relation which is then transformed by means of a fixed point computation to yield a transition relation for the phase-level response of the network. Existential quantification to hide the clock yields the cycle-level transition relation. The fixed point computation done in this method computes the steady state phase-level response of a network. Our work shows that a symbolic simulator can compute the steady state phase-level response of a network and do so much more efficiently. As the number of phases increase in a clock cycle, the number of symbolic values introduced to compute the cycle-level transition relation increases. This can lead to very large Ordered Binary Decision Diagrams (OBDDs) [2]. Also for circuits like the one in figure 2, the fixed point computation will never converge [11]. Anamos identifies s1 and q as charge storage nodes and when Phi1 is 0, the circuit will not stabilize for similar initial values on s1 and q. Consider the symbolic values *a* on node s1 and *b* on node q. After one unit step the values on nodes s1 and q are $\overline{b}$ and $\overline{a}$. After one more unit step the values on nodes s1 and q are *a* and *b* . This unit delay oscillation will never cease and the circuit will not converge to a stable state. Static latches play an important role in modern digital designs. Not being able to handle such structures is a serious limitation of the above approach.

Another disadvantage of the above approach is that clocks are not factored in early enough in the extraction process. This will cause the fixed point computation not to converge even if the circuit oscillates only when the clocking constraints are violated [1].

The technique proposed in this paper does away with the disadvantages of the earlier approaches without sacrificing efficiency. From the transistor network a gate-level model is generated using Tranalyze

Figure 4: Example of a circuit implementing a FSM

Tranalyze analyzes the components in rank order. In order to ensure a rank ordering, Tranalyze breaks all zero delay cycles by inserting unit delays on some of the transistors. As an example, Tranalyze will insert a unit delay buffer to model the effect of the trickle inverter in figure 4. It does this by a simple greedy algorithm during the topological sorting of the components. This greedy algorithm does not guarantee that a minimum number of feedback loops are broken. While this can lead to more state nodes than necessary, in practice the greedy algorithm works well. We have not encountered any sub-optimal cycle breaking on any of the benchmarks that we have run so far.

After rank ordering, Tranalyze performs a symbolic switch-level analysis of each component. The analysis operates on a 4-valued logic set {0,1,X,Z}. The resultant gate-level model generated by Tranalyze is a sequential circuit with a zero delay combinational network and all storage elements are represented by unit delay buffers.

As an example, figure 5 shows the gate-level netlist obtained for the transistor network in figure 4. Tranalyze inserts unit delay buffers D1 and D2 to model the effect of stored charge on nodes t1 and t2. The third unit delay buffer D3 is used to break the zero delay cycle due to the trickle inverter. Tranalyze automatically detects that stored charge on nodes other than t1 and t2 does not affect the circuit operation. These 3 unit delay buffers are the storage elements for the transistor network.

The state nodes are a subset of the storage elements of the network. The extraction algorithm, which is described in section 3.5, identifies this subset along with the next-state and output functions.

3.3. Circuit Model

A circuit implementing a FSM has

- Clock inputs: C_1, C_2, ..., C_n
- Primary inputs: In_1, In_2, ..., In_m
- Primary outputs: O_1, O_2, ..., O_k
- State nodes: Sn_1, Sn_2, ..., Sn_l

Figure 5: Tranalyze generated network from figure 4.

The first three are given as a part of the circuit definition. For a complete definition of the clock, in addition to identifying the clock inputs, the sequence of phases P_1, P_2, ..., P_j which make up a clock cycle and the binary values of C_1, C_2, ..., C_n for each of these phases must be specified.

The primary inputs and primary outputs are specified as follows. In our example, the clock inputs to the circuit are Phi1 and Phi2. One clock period contains four phases, P_1 to P_4. To specify the clock completely the clock input values for all the four phases must be specified.

Clock Input	P_1	P_2	P_3	P_4
Phi1	0	1	0	0
Phi2	0	0	0	1

For each primary input the phases during which it is stable must be specified. In the example above, the primary inputs must be stable across the falling edge of Phi1. We specify this as:

Primary Input	P_1	P_2	P_3	P_4
In1	—	Stable	Stable	—
In2	—	Stable	Stable	—

Each row in the table above represents one input. This format also allows the specification of time multiplexed inputs. For example, if node In1 were time multiplexed, with one input being applied during phases 1 and 2 and the other during phase 4, this would be specified by giving two rows for In1 as below:

Primary Input	P_1	P_2	P_3	P_4
In1	Stable	Stable	—	—
In1	—	—	—	Stable

For each primary output the phase at the end of which the output is valid, and therefore can be sampled, must be specified. In the circuit in figure 4, the output is valid at the end of P_4.

Primary Output	P_1	P_2	P_3	P_4
Out	—	—	—	Sample

Each row represents one output. Time multiplexed outputs can be specified in the same way as time multiplexed inputs. The operation of a circuit during a clock cycle is modeled as described in section 3.5.

3.4. Symbolic Simulation

Symbolic simulation involves evaluating circuit behavior using symbolic values to encode a range of circuit operating conditions [6]. Our gate-level symbolic simulator operates on the ternary domain

v	$v.H$	$v.L$
0	0	1
1	1	0
X	1	1

Table 1: Dual rail encoding of {0,1,X} logic values.

Normal Mode	Oscillation Suppression Mode
$N.H \leftarrow E_N.H$	$N.H \leftarrow N.H \vee E_N.H$
$N.L \leftarrow E_N.L$	$N.L \leftarrow N.L \vee E_N.L$

Table 2: Updating node N in a symbolic simulator.

{0,1,X} (the 4th value Z never arises on a state node). The simulator encodes these 3 logic values by a pair of Boolean values. Each logic value v is represented by a pair of Boolean values, $v.H$ and $v.L$, according to the encoding in table 1.

The encoding in table 1 is extended to operate on symbolic values over the ternary domain. This is done by encoding each symbolic ternary value by a pair of Boolean functions. The Boolean functions are represented as OBDDs. Each node N in the network has a pair of Boolean functions $N.H$ and $N.L$, representing the current value of the node. Roughly speaking, the function $N.H$ (respectively $N.L$) encodes the set of conditions that try to pull the node to a high (respectively low) logic level. During simulation, inputs and storage nodes can be set to symbolic values. Normally, inputs are restricted to Boolean values. To set an input *In* to a Boolean symbolic value vin, the high rail of the node, *In.H* is set to the value vin, and the low rail of the node, *In.L*, is set to the complementary value $\overline{vin}$.

Each phase during simulation consists of a number of time steps. In each time step, the simulator computes the excitation at node N as a pair of functions, $E_N.H$ and $E_N.L$, to update the high and low encodings for the node. During normal operation the node simply gets updated with $E_N.H$ and $E_N.L$. A phase also has a maximum number of time steps. If the network does not stabilize within this number of time steps, the simulator enters an oscillation suppression mode. In this mode, nodes that do not stabilize are updated to the disjunction of the high and low encodings for old node value and the excitation value as shown in table 2. The effect of this is to set circuit nodes to symbolic values which evaluate to X under unstable conditions. This mode guarantees convergence within a fixed number of steps. Thus, on each phase the simulator evaluates the steady state response of the network.

At the end of the phase, the output and storage nodes can be observed. Some applications require state nodes to be restricted to Boolean values. In such applications, an X value on a state node implies faulty operation. Our technique can identify all cases that lead to such faulty operation. At the end of a phase, the conjunction $(N.H \wedge N.L)$ represents the set of conditions that can lead to an X value on a node N.

3.5. Extraction Algorithm

The gate-level network generated by Tranalyze consists of zero delay gates and unit delay buffers which represent its storage elements. Let Sn_1, Sn_2, ..., Sn_s be the nodes that are outputs of these storage elements. The extraction algorithm consists of these steps.

1. For each node Sn_j, introduce a boolean variable vsn_j and set the node to the symbolic value vsn_j.

2. For i = 1 to Number of clock phases:
   (a) Set the clock inputs to the values corresponding to the ith clock phase.
   (b) For each primary input In_j, introduce a boolean variable in_j and set the input to the symbolic value in_j if the specification indicates that the input value is stable in phase P_i. If nothing is specified for the input in phase P_i, set the input to X.
   (c) Simulate the circuit until it is stable. Oscillation suppression guarantees that it will stabilize.
   (d) If the output node O_j is specified as having a valid value at the end of the i^{th} clock phase then set the output function OF_j to the symbolic value on node O_j.

3. Let V_0 be the set of all Boolean variables vsn_j representing state values appearing in the support sets of OF_1, OF_2, ... OF_k.

   Compute V_i as shown below until $V_i = V_{i-1}$.

   $$V_i = V_{i-1} \cup \{\ vsn_k \mid vsn_k \text{ is in support set of function for some node } Sn_j, \text{ where } vsn_j \in V_{i-1}\}$$

   When $V_i = V_{i-1}$ the set of nodes Sn_j corresponding to the Boolean variables vsn_j in V_i are the true state nodes of the circuit.

   The set of Boolean functions on Sn_j at the end of the simulation describe the next state (clock-level) function.

The oscillation suppression mode in the symbolic simulator allows us to extract the FSM for oscillating circuits where the oscillation never appears on the circuit output. This is an advantage over the fixed-point computation of [12] which will not converge on such circuits.

3.6. Example

We illustrate some of the ideas in our extraction algorithm with the help of the circuit in figure 4. Of course, as experimental results show, we can deal with much larger circuits. Given a transistor-level implementation of the circuit in figure 4, Tranalyze generates the gate-level circuit shown in figure 5. This circuit contains three storage nodes D1, D2 and D3. A subset of the set of storage nodes in the circuit constitute the set of state nodes. This subset depends on the clocking strategy, the phase in which the inputs are applied and the phase in which the outputs are observed.

Given the clocking and the IO described in section 3.3, this circuit has no state nodes. It is easy to see this because the output sampled at the end of the 4th phase is a function of only the inputs applied at nodes In1 and In2 during the falling edge of Phi1. The output function in this case is $\mathsf{Out} = \overline{\mathsf{In1}} \wedge \overline{\mathsf{In2}}$.

If the output is sampled at the end of the second phase instead of the fourth, the circuit contains one state node D3. This is because the value stored in the static latch shows up at the output. The output and the next state functions are given below, where $\mathsf{PS(D3)}$ is the present state of state node D3 and $\mathsf{NS(D3)}$ is the next state of D3.

$\mathsf{Out} = \overline{\mathsf{PS(D3)}}$

$\mathsf{NS(D3)} = \mathsf{In1} \wedge \mathsf{In2}$

If the output is sampled at the end of the second phase and the clocks Phi1 and Phi2 are interchanged, then the circuit contains two state nodes D1 and D2. In this case, the sampled output value is a function of the charge stored at nodes D1 and D2. The output and the next state functions in this case are:

$\mathsf{Out} = \overline{\mathsf{PS(D1)}} \wedge \overline{\mathsf{PS(D2)}}$

$\mathsf{NS(D1)} = \mathsf{In1}$

$\mathsf{NS(D2)} = \mathsf{In2}$

4. Implementation

A specification language was defined to specify IO and clocking of a circuit. From circuit specifications in this language and the storage nodes in the circuit, the set of simulation patterns needed for FSM extraction is generated.

A gate-level symbolic simulator developed at Carnegie Mellon University was modified to allow setting of storage nodes, sampling the outputs and performing the iterative computation to identify the state nodes.

In order to keep the OBDD sizes small, a few general principles were followed for manual ordering of the Boolean variables. Boolean variables corresponding to the primary inputs appeared above the Boolean variables corresponding to state nodes. Among the primary inputs, the variables corresponding to the control signals appeared above the variables for the data signals. In some cases the Boolean variables corresponding to the input nodes and the state nodes were interleaved to get smaller OBDD sizes.

5. Experimental Results

5.1. Extraction Results

We have extracted the FSM for a number of different circuits and the results are presented in tables 3 and 4.

For every circuit in tables 3 and 4, the second column (# Trans) and the third column (# Node) give the number of transistors and the number of nodes in the switch level network. The fourth column (# Store) and the fifth column (# State) give the number of storage nodes in the circuit and the number of state elements in the extracted FSM.

Columns 6 and 7 give the space and time figures for our extraction process. Column 6 (Max/FSM OBDD) gives the maximum number of OBDD nodes created during the symbolic simulation, and the total number of OBDD nodes needed to represent the next state and the output functions. The seventh column gives the FSM extraction time in seconds. The times in this column have been measured on a SparcStation 5 with 32MB memory.

In table 3 we compare the results of FSM extraction using our technique with that presented in [13] on an identical set of benchmarks. This table has two extra columns which contain the results reported in [13]. In column eight (Z/N OBDD), the number corresponding to Z is the number of OBDD nodes required to represent the step-level transition relation for the circuits. In the same column, the number corresponding to N is the number of OBDD nodes required to represent the cycle-level transition relation. The last column is the extraction time for the circuits in seconds measured on a DECstation 5000/240.

Table 4 shows the extraction results for different sizes of a Mead-Conway style moving data stack [15] with a combinational controller

Circuit				Extraction Results			Results from [13]	
Circuit	# Trans	# Node	# Store	# State	Max/FSM OBDD	Ext Time	Z/N OBDD	Ext Time
seradd	38	24	4	1	100/12	0.1	127/25	0.2
sermult1	60	35	8	3	147/8	0.2	697/111	2.8
sermult2	76	43	12	5	151/11	0.3	945/191	3.7
sermult4	108	59	20	9	159/15	0.4	1441/351	5.2
sermult8	171	91	36	17	175/23	0.7	2733/767	9.4
sermult16	298	155	68	33	207/39	1.4	4417/1311	17.0
adder1	38	25	2	0	127/6	0.1	168/30	0.4
adder2	74	45	4	0	162/15	0.2	417/57	1.7
adder4	146	85	8	0	295/42	0.6	915/111	9.0
adder6	218	125	12	0	512/81	1.1	1413/165	36.5
adder8	290	165	16	0	813/132	1.7	1911/219	122
accum1	36	24	4	1	133/6	0.2	246/43	0.6
accum2	72	43	8	2	174/19	0.3	586/84	2.8
accum4	144	81	16	4	349/57	0.9	1266/166	15.8
accum6	216	119	24	6	701/135	1.7	1946/248	52.9
accum8	288	157	32	8	1071/173	2.4	2626/330	219
shift2	16	13	4	1	117/2	0.1	69/18	0.1
shift8	64	37	16	7	129/8	0.4	291/78	0.4
shift8*2	128	70	32	14	146/16	0.7	589/155	1.1
shift16	128	69	32	15	145/16	0.7	587/158	0.9
shift32	256	133	64	31	177/32	1.3	1179/318	2.5
shift32*4	1024	520	256	124	372/128	5.3	7014/1517	30.7
shift32*8	2048	1036	512	248	632/256	10.6	14038/3033	94.1

Table 3: Extraction results on benchmarks from [13]

Circuit Size Metrics				Extraction Results		
Circuit	# Trans	# Node	# Store	# State	Max/FSM OBDD	Ext time
stack16	198	106	32	16	322/51	1.5
stack32	326	170	64	32	530/99	3.1
stack64	528	298	128	64	946/195	6.4
stack128	1094	554	256	128	1778/387	14.8
stack256	2118	1066	512	256	3442/771	42.1
stack512	4166	2090	1024	512	6770/1539	171.9
sram16	244	110	21	16	1188/140	6.0
sram32	362	148	37	32	2722/316	11.7
sram64	692	274	73	64	6522/700	23.6
sram128	1126	412	137	128	15014/1532	57.5
sram256	2176	790	273	256	34370/3324	119.8
sram512	3826	1320	529	512	76738/7164	260.2
sram1024	7476	2586	1057	1024	173144/15340	528.0
μP Ctrlr	5926	1017	87	8	97132/1325	49.6

Table 4: Extraction results from stack, SRAM and microprocessor controller.

and for different sizes of a static RAM circuit. The last benchmark is the micro-coded control circuit from the Hector Microprocessor [16].

5.2. Discussion of Results

All circuit nodes in a switch-level circuit (column 3) are potential state nodes. After processing by Tranalyze, the resultant gate-level representation contains far fewer storage nodes (column 4). Given the circuit clocking and IO behavior, the FSM extractor eliminates many of these storage nodes to yield a small number of state nodes (column 5). Thus, the final representation of the circuit is a much simplified version of the initial transistor netlist. For example, the Hector microprocessor controller (table 4, last row) contains 1017 nodes in the extracted transistor netlist. The Tranalyze, generated gate-level netlist contains 87 storage elements. After the FSM extraction is done, only 8 state nodes remain in the clock-level FSM representation. In table 4, for the various memory circuits, the number of state nodes in the FSM representation reduces to the number of memory bits.

Table 3 also compares the results reported in [13] (columns 8,9) with the results obtained from our extractor (columns 6,7) on the same set of benchmarks. In most benchmarks, our extraction times[1] and the number of OBDD nodes generated in our extraction process are at least an order of magnitude smaller than the figures reported in [13]. The reasons for this improvement are explained below.

Representing the unit step transition relation, Z, requires very large OBDDs. This is because the relation must encode every possible mode of circuit operation. It does not take advantage of any circuit-specific information that may be available e.g., that clocks do not overlap, that they occur in a specific sequence or that in precharged circuits the precharge phase always occurs before the evaluate phase. Symbolic simulation takes advantage of such circuit-specific information and this results in smaller OBDD sizes and smaller extraction times. As can be seen from the table, the maximum OBDD size generated during symbolic simulation of a complete clock cycle is much smaller than the OBDD size required to represent just the unit step transition relation. In the benchmarks the number of OBDD nodes required to represent the FSM in terms of its next state and output functions is always less than the number of OBDD nodes required to represent the cycle level transition relation.

The moving data stack in table 4 shows a linear increase in the maximum OBDD size and the number of OBDD nodes required for the next state and output function with increasing circuit size. The number of state nodes is half the number of storage nodes. In case of SRAM, the OBDD size and the extraction time increase linearly with the circuit size.

[1] The extraction time in [13] and our work have been measured on two different machines, But, roughly speaking, these have similar performance.

6. Further Work

If a state node has a symbolic ternary value during the computation of V_i, the extraction process is halted with an error. The ternary value indicates that a state node can also take on the unknown value X for certain combinations of input and initial state values. The presence of X's indicates an inconsistent combination of state and input values which can arise because of faulty circuit structures or non-optimal cycle breaking. Because an X at a state node is treated as an error condition, we are unable to deal with don't cares at the inputs or constraints between state node values in a circuit.

A solution to non-optimal cycle breaking is symbolic trajectory evaluation (STE) [7] which is a modified form of symbolic simulation. In STE nodes can be asserted to symbolic values and as the simulation proceeds, the circuit state is updated subject to the constraints of the node assertions. In this process a Boolean function called OKT is computed. This function indicates the cases for which the circuit is not over-constrained by the assertions. For example, if the input and the output of an inverter are asserted respectively to the symbolic Boolean values a and b, the circuit will be over constrained for cases where $a = b$, and the function OKT will be $a \oplus b$. When STE is used to extract a FSM from a circuit with non-optimal cycle breaking, OKT can indicate the constraint under which the extracted FSM must operate. OKT can be considered as an auxiliary output of the FSM which, if true, indicates that the FSM is operating correctly.

An example of a state node constraint is the use of one-hot state logic where, for correct operation, the circuit must always satisfy the constraint that in a given set of state nodes only one is high at a time. We can not easily express such constraints in our present framework. A possible solution to this as well as don't cares at the inputs is the use of parametrized symbolic simulation [10].

7. Conclusion

This work has described a new technique to extract a clock-level FSM from a transistor netlist. It has shown that symbolic simulation is a powerful tool to extract FSMs from transistor netlists and it has demonstrated the feasibility of the technique empirically. This technique is efficient and on many circuits it scales well with increasing circuit size. On many benchmarks it takes at least an order of magnitude less time and space than computing the transition relation through fixed point computation. It can handle a wide range of circuit designs including static storage structures and time multiplexed inputs. This is an advance over earlier approaches which are incapable of handling static storage structures. The extraction process is completely automated and it requires minimal information about the circuit. A limitation of the approach is the non-optimal cycle breaking by Tranalyze, but in practice it has caused no problems on a wide range of circuits. The main contribution of this work has been to develop a new FSM extraction technique based on symbolic simulation. It has brought out issues in the extraction process including some limitations of the approach.

8. Acknowledgements

Timothy Kam provided useful discussions and the benchmarks in [13].

References

[1] D. L. Beatty, "A Methodology for Formal Hardware Verification with Application to Microprocessors," Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, August 1993.

[2] R. E. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," IEEE Transactions on Computers, C-35(8), 1986.

[3] R. E. Bryant, "Extraction of Gate Level Models from Transistor Circuits by Four-Valued Symbolic Analysis," ICCAD, 1991.

[4] R. E. Bryant, "Boolean Analysis of MOS Circuits," IEEE Transactions on CAD of Integrated Circuits CAD-6(4), July, 1987.

[5] R. E. Bryant, D. Beatty, K. Cho, and T. Sheffler, "COSMOS: A Compiled Simulator for MOS Circuits," 24th DAC, 1987.

[6] R. E. Bryant, "Symbolic Simulation—Techniques and Applications," DAC, 1990.

[7] R. E. Bryant, C. J. H. Seger, "Formal Verification of Digital Circuits Using Symbolic Ternary System Models," Computer Aided Verification, pages 121 - 146, 1990.

[8] S. Bose, and A. Fisher, "Automatic Verification of Synchronous Circuits Using Symbolic Logic Simulation and Temporal Logic," IMEC-IFIP International Workshop on Applied Formal Methods For Correct VLSI Design, 1989.

[9] O. Coudert, J. C. Madre and C. Berthet, "Verifying temporal properties of sequential machines without building their state diagrams," CAV, 1990.

[10] P. Jain, and G. Gopalakrishnan, "Efficient Symbolic Simulation-Based Verification Using the Parametric Form of Boolean Expressions," IEEE Transactions on CAD of Integrated Circuits, CAD-13(8), August, 1994.

[11] T. Kam, Personal Communication, July, 1994.

[12] T. Kam, and P.A.Subrahmanyam,"State Machine Abstraction from Circuit Layouts using BDD's: Applications in Verification and Synthesis," EDAC, 1992.

[13] T. Kam, and P.A.Subramanyam, "Comparing Layouts with HDL Models: A Formal Verification Technique," ICCD, 1992.

[14] Kenneth L. McMillan, "Symbolic Model Checking - An approach to the state explosion problem," Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, May 1992.

[15] C. A. Mead, and L. A. Conway, "Introduction to VLSI Systems," Addison-Wesley, Reading, Mass. and London, 1980.

[16] T.K.Miller III, B.B.Bhuva, R.L.Barnes, J-C. Duh, H.B. Lin, and D. E. Van den Bout, "The HECTOR microprocessor," ICCD, 1986.