

Synthesis of Timed Circuits using BDDs *

Robert A. Thacker
Computer Science Department
University of Utah
Salt Lake City, UT 84112

Chris J. Myers
Electrical Engineering Department
University of Utah
Salt Lake City, UT 84112

Abstract

This paper presents a tool which synthesizes timed circuits from reduced state graphs. Using timing information to reduce state graphs can lead to significantly smaller and faster circuits. The tool uses implicit techniques (binary decision diagrams) to represent these graphs. This allows us to synthesize larger, more complex systems which may be intractable with an explicit representation. We are also able to create a parameterized family of solutions, facilitating technology mapping.

1. Introduction

Asynchronous design has been gaining in popularity in recent years [5, 6, 8, 14]. Increasing clock speeds and larger ICs make it ever more difficult to maintain a globally synchronous environment. Asynchronous circuits eliminate these difficulties by removing clocks and using independent handshaking protocols. We work with a class of asynchronous circuits known as *timed circuits*, which use explicit timing information to optimize the implementation[10]. Using timing information makes it possible to greatly reduce the state space to be explored. It can also lead to much more efficient circuits, since eliminated states do not need to be considered when implementing the specification.

The goal of this work is to facilitate synthesis of efficient asynchronous circuits using implicit techniques. *Binary decision diagrams* (BDDs) are a simple, efficient method of representing and manipulating design information[3]. BDDs allow us to compactly represent very large state spaces, and traverse and manipulate them in reasonable amounts of time. They also allow us to derive whole families of results which can be easily evaluated. Having multiple results is very useful during technology mapping: one result may already have been mapped for another signal, or be easier to decompose than others. These options are not available if only one solution is found.

* This research is supported by a grant from Intel Corporation and NSF CAREER award MIP-9625014.

2. BDD representation of a reduced state graph

Our synthesis system starts with a circuit specification represented in a *reduced state graph* (RSG), which can be derived from many higher-level languages such as CHP and STGs [10], as well as more recently VHDL [16]. *State graphs* are a common intermediate form for many asynchronous CAD tools [7, 13, 15]. A RSG is a state graph wherein the number of reachable states has been reduced by considering timing information. A number of algorithms exist for doing the necessary timing analysis [2, 11]. A RSG is represented as a graph in which the vertices are *states* and the edges are possible *state transitions*. States are represented by a vector $\mathbf{x} = \langle x_1, x_2, \dots, x_n \rangle$, where each variable x_i represents a signal in the system. These variables may take on any one of the following values: 0 denotes a stable low signal, R denotes a signal enabled to rise, 1 denotes a stable high signal, and finally F denotes a signal enabled to fall. A transition between two states indicates that, upon the appropriate signal change, the system will change to the successor state.

For example, Figure 1(a) shows the block diagram for a self-resetting dynamic OR gate (SRDOR). When an input rises, the output rises. After some delay, the internal signal x will fall, causing the gate to be reset (the input is assumed to have fallen by this point). The RSG describing the behavior of this circuit is shown in Figure 1(b). In the initial state, where $\mathbf{x} = \langle i1, i2, x, a \rangle = (RR10)$, both $i1$ and $i2$ are enabled to rise, while a is stable low and x is stable high. This state may be exited either on the transition $i1+$ or $i2+$. If $i1$ rises, the state (F01R) is entered, in which either a may rise or $i1$ may fall, while $i2$ is stable low, and x is stable high. Notice that there is no edge for the possible transition $i1-$. Timing considerations allow us to determine that $a+$ always occurs first, so we eliminate (001R) as a reachable state. Also, note that the environment will provide only one of the transitions $i1+$ and $i2+$; it is assumed both will not be asserted simultaneously. The occurrence of $i1+$ therefore disables the transition $i2+$, and we enter state (F01R), not (FR1R).

To represent the reachable state space as a BDD, a pre-

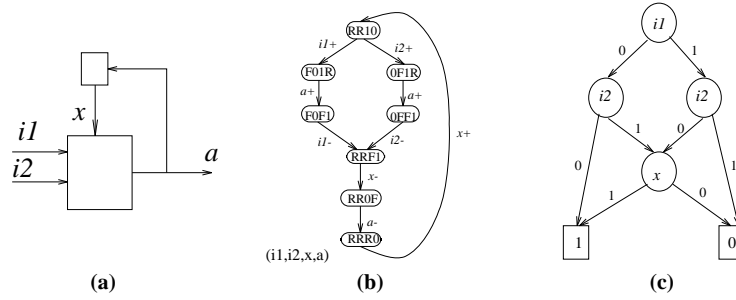


Figure 1. Self-resetting dynamic OR gate: (a) block diagram, (b) RSG, and (c) BDD for S .

predicate S on the vector \mathbf{x} is defined which returns true for all states reachable in any number of transitions from the initial state. Figure 1(c) shows the BDD for the state space predicate for the SRDOR example. The BDD S shows that the reachable states are those in which (1) both $i1$ and $i2$ are low, or (2) exactly one of $i1$ and $i2$ are high and x is also high.

The NextState function N is a predicate on $S \times S$ which returns true for all state pairs (s, s') for which s' may be reached from s in exactly one signal transition. The BDD N is analogous to S , but a complete path through the graph represents a pair of states (i.e., it passes through nodes representing \mathbf{x} and \mathbf{x}'), and the terminal node indicates whether a transition from S to S' is valid.

A complication arises from the use of timing in generating the RSGs. As mentioned before, when timing considerations show a state to be unreachable, it may be removed from the RSG. If we based our implementation only on the RSG, the signal enablings leading to each of these states would be lost, and the resulting circuit would be suboptimal. In the SRDOR example, a naive derivation of S and N actually represents the state graph found in Figure 2(a). This graph correctly describes the signal changes, but not the enablings, and produces the circuit found in Figure 2(b). A correct graph is shown in Figure 2(c) and produces the circuit shown in Figure 2(d). This circuit is smaller and faster than the circuit derived from the incorrect RSG.

The basic problem can be illustrated using the familiar diamond shown in Figure 3. The original speed-independent graph is shown in Figure 3(a). Because our timing analysis says that the signal b will always rise before a , we remove the state (1R) from the graph. If the correct enablings are not maintained, we end up with the less concurrent graph shown in Figure 3(b). The enabling of a is now delayed by the time necessary to fire b , and each cycle of the circuit is slowed by that amount. It should also be noted that the less concurrent circuit may not only be slower, but it may also be incorrect if it violates the original overall timing assumptions.

To maintain the correct enablings, we add to the RSG a transition to a “ghost state” whenever we find an enabling without a matching next state. This ghost state consists of

the same values as the original state, except that the enabled signal has changed phase. Now, when we compute N , the correct enablings are derived.

Figure 3(c) shows an example of a “haunted” graph where the state (1R) has been reinserted as a “ghost state”, with a transition from (RR). This path will never be taken, but it is essential that it be represented. In the SRDOR example, several ghost states are necessary, such as (001R) which would be reached from state (F01R) by the transition $i1-$.

3. A parameterized family of timed circuits

Our timed circuits are implemented by creating a function block for each output signal, consisting of a C-element with a *sum of products* (SOP) block for the set and another for the reset. The circuit may be implemented using a standard C-element (SC) structure using discrete gates or using a complex gate known as a *generalized C-element* (gC)[9].

Each “product” block implements a single *excitation region* (ER) for a given output signal. An excitation region for the output signal x is a maximally connected set of states in which the signal is enabled to change to a given value (i.e., $x = R$ or $x = F$). If the signal is rising in the region (i.e., $x = R$), it is called a *set region*, otherwise the region is called a *reset region*. We also define a set of *excited states* (ES), which is the union of the excitation regions for a given signal transition. For each signal transition, there is also an associated set of stable, or *quiescent*, states (QS). For a rising transition $x+$, this is the set of states where the signal is stable high, and is similarly defined for a falling transition. Given N , the BDD representations of ES, QS, and ER are straightforward to obtain.

In our SRDOR example, let us consider the excitation region for $x-$. In the naive graph, this region is just $\{(00F1)\}$. In the “haunted” version, it is extended to $\{(RRF1), (OFF1), (F0F1)\}$. The quiescent set for the same transition is $\{(RR0F)\}$ (in the naive derivation it is $\{(000F)\}$).

In [4], a parameterized family of decompositions is investigated one at a time by adding additional variables. We extend this idea to synthesis by representing the potential

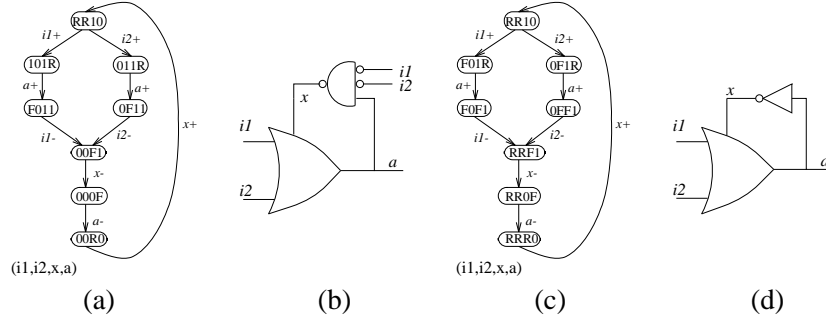


Figure 2. SRDOR: (a) incorrect enablings and (b) circuit, (c) correct enablings and (d) circuit.

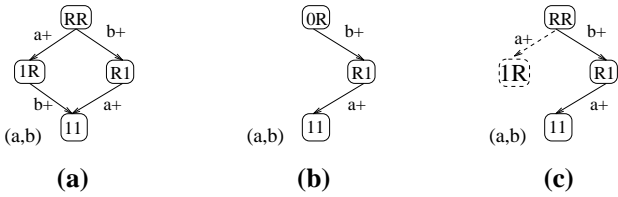


Figure 3. Simple diamond:(a) speed independent, (b) timed with incorrect enablings, (c) timed with correct enablings, and a transition to a “ghost state”.

“product” covers for each excitation region. Our covers are represented by a series of implications of the form $(\chi_i \Rightarrow x_i) \wedge (\chi_{n+i} \Rightarrow \neg x_i)$. The χ variables enable us to consider separately the positive and negative phases of each signal, and indicate the possibility of using that signal *and* phase in a cover. These implications will be ANDed together to produce a BDD which represents every possible potential single cube cover of the corresponding ER, i.e., $C_0 = \bigwedge_{i=1}^n \Psi_{i,0}$, where $\Psi_{i,0} = [(\chi_{i,0} \Rightarrow x_i) \wedge (\chi_{n+i,0} \Rightarrow \neg x_i)]$.

Occasionally an excitation region is found which cannot be covered by a single cube. To solve this problem, we generalize the “product” block to a SOP block to represent this region. If a single cube solution is not found, a second (or third, etc.) initial cover is created, and ORed together with the preceding initial cover (i.e., $C = C_0 \vee C_1 \vee \dots \vee C_m$).

In order to create a valid timed circuit implementation, it is necessary to define the states a cover must include, may include, and may not include. The correctness constraints discussed here were developed in [1] for speed-independent circuits and extended to timed circuits in [12]. In a gC implementation, the allowed growth regions include all states in ES and QS for the corresponding signal transition. This *covering constraint* prevents the gate from being pulled up and down simultaneously or changing values at the wrong time. In a SC implementation, additional internal signals are introduced by the use of discrete gates. In order to prevent the introduction of hazards, additional restrictions are placed on the states allowed in the cover. These restrictions

ensure that each cover makes a single monotonic acknowledged transition when it is actively changing the output and makes no transitions at any other time. The covering constraint is modified to include *only* states from this ER or the corresponding QS. This ensures that only one “product” block is on at a time, so the transition can be acknowledged by a transition on the output. In addition, the cover may only be entered through the excitation region. This *entrance constraint* guarantees a single monotonic transition, with no unacknowledged glitch in the function block.

The BDD for the valid cover, VC, is constructed such that it returns all implementations that completely cover the corresponding excitation region and possibly cover other states as allowed by the correctness conditions. Any satisfying assignment of the remaining BDD is a valid implementation: if a χ variable appears in the positive phase, the implied variable must appear in the cover; if it appears in the negative phase, the variable cannot be included; and if it does not appear at all, its use is at the designers discretion.

4. Results and conclusions

The complete BDD timed circuit synthesis procedure is shown in Algorithm 4.1. First, we derive the characteristic functions for the state graph (S) and NextState relation (N). A correct NextState relation requires that the graph first be populated with appropriate “ghost” states. For each output signal, we then decompose the graph into appropriate quiescent sets (QS), excited sets (ES), and excitation regions (ER). The set of violating states (V) for each ER is then found, and a valid cover is derived (i.e., one which includes the ER , but no part of V). If no valid cover can be found, the potential cover (C) is expanded, and the covering step is retried. In this fashion, we transparently derive multicube covers as necessary.

This algorithm has been automated within the CAD tool ATACS and applied to the design of a large number of circuits (see Table 1). The first five are standard speed-independent benchmarks, and the rest are timed circuits. The table gives the size of the state space both in states (Φ) and transitions (Γ) (for the explicit representation) and BDD nodes (for the implicit representation). The runtimes repor-

Algorithm 4.1 (Synthesize)

```

bdd_List Synthesize(RSG G) {
  S = FindStateGraphBDD(G);
  G = Haunt(G);
  N = NextState(G);
  Foreach output  $x_i$  in  $\mathbf{x}$  {
    Foreach  $x_i^*$  in  $\{x_i^+, x_i^-\}$  {
      C = Generate  $C_0$ ;
      If ( $*$  = +) {
        QS( $x_i^*$ ) = exist_q( $\mathbf{x}'$ ,  $x_i \wedge x_i' \wedge N$ );
        ES( $x_i^*$ ) = exist_q( $\mathbf{x}'$ ,  $\neg x_i \wedge x_i' \wedge N$ );
      } else {
        QS( $x_i^*$ ) = exist_q( $\mathbf{x}'$ ,  $\neg x_i \wedge \neg x_i' \wedge N$ );
        ES( $x_i^*$ ) = exist_q( $\mathbf{x}'$ ,  $x_i \wedge \neg x_i' \wedge N$ );
      }
      ER( $x_i^*$ ) = FindER( $\mathbf{x}$ , N, ES);
      Foreach  $ER_k$  in ER {
        Do {
          if (gC) then
            V = S  $\wedge$   $\neg$ ES  $\wedge$   $\neg$ QS;
          else {
            V1 = S  $\wedge$   $\neg$ ERk  $\wedge$   $\neg$ QS;
            V2 = TRANS( $\mathbf{x}' \rightarrow \mathbf{x}$ , exist_q( $(\mathbf{x},$ 
              N( $\mathbf{x}, \mathbf{x}') \wedge \neg C(\mathbf{x}) \wedge C(\mathbf{x}') \wedge \neg ER_k(\mathbf{x}')$ ));
            V = V1  $\vee$  V2;
            VC = univ_q( $\mathbf{x}$ , ( $\neg C \vee \neg V$ )  $\wedge$  ( $\neg ER_k \vee C$ ));
            If (VC =  $\emptyset$ )
              C = C  $\vee$  Generate next  $C_i$ ;
          } While (VC =  $\emptyset$ );
          add VC to set_of_results;
        }
      }
    }
  }
  Return set_of_results
}

```

Figure 4. Function to synthesize circuit from a reduced state graph G

ted are in milliseconds on a 200 MHz PentiumPro workstation with 32 Mbytes of memory. The sixth column of each table reports the runtime to translate the explicit representation of the state space to two BDDs. We compare the runtimes of our BDD synthesis method to a heuristic single-cube algorithm [10], and to a general multi-cube algorithm [1]. For each algorithm, we compare the runtimes for generating both a gC and a SC implementation. An entry of “fail” indicates that an algorithm did not complete due to limitations in time and space.

The last two columns present the number of potential implementations for the entire circuit. This number is the product of the possible covers for each individual excitation region. For example, in the gC implementation of the SRDOR, the set region for x has 8 solutions, the reset region has 2, and the 2 set regions and the reset region for a each have 8 solutions, which makes a total of $8 \times 2 \times 8 \times 8 \times 8 = 8192$. The SC implementation is more restricted so it only has 80 possible solutions. Obviously the user could not be

expected to consider each of these solutions separately. Filters would be used to reduce the set to those having reasonable implementations in the target technology (e.g. a circuit requiring a CMOS gate with 20 inputs can probably be discounted entirely). This process could also be used to factor common subexpressions and enable component sharing.

We observe that although it is somewhat slower than the heuristic single-cube algorithm, our BDD synthesis method never fails, and in comparable runtime, finds BDD representations for a huge number of possible synthesis solutions. The heuristic algorithm fails when multi-cube covers are required. Our BDD method typically takes more than an order of magnitude less time than the general algorithm while still finding all possible solutions. The general algorithm fails on the smaller examples as the resolution of cyclic covering tables is not implemented. The general algorithm fails on larger examples such as *trimos-send* due to limitations in time and space.

This paper presents a new synthesis method for timed circuits which utilizes BDDs. We formulated a BDD representation for state spaces which have been reduced using timing information. We use ghost transitions to preserve accurate signal enabling information. We have developed BDD formulations and algorithms for both standard-C and generalized C-element implementation styles. These algorithms find all valid covers for each excitation region (if necessary, by transparently finding minimal multi-cube covers). Our BDD synthesis method performs nearly as fast as a heuristic single-cube method, and it can perform more than an order of magnitude better than a general multi-cube algorithm. The major advantage of the BDD synthesis method is that a parameterized family of solutions is found while earlier algorithms merely found a single solution. Considering all possible valid implementations will greatly facilitate technology mapping, which we are beginning to investigate.

While at the moment we are using an explicit state graph derived by ATACS, in the future we plan to extend our algorithm to derive the BDD representation of the reduced state space directly from a higher-level specification. We are also investigating methods for choosing covers for various cost functions. Finally, we plan to extend our work to the technology mapping of timed circuits.

5. Acknowledgments

We are especially grateful to Dr. Steve Burns from Intel Corporation for numerous insightful discussions about BDDs. We would also like to thank Wendy Belluomini, Dr. Ganesh C. Gopalakrishnan, Luli Josephson, Hans Jacobson, Brandon Bachman, and Eric Mercer for their illuminating comments on this paper. Finally, we would like to extend our thanks to Dr. David Long of AT&T Bell Labs for writing a great BDD package.

Table 1. Experimental results for timed benchmarks. Time values are given in milliseconds. An entry of *fail* indicates that the synthesis did not complete.

Examples	Φ	Γ	S	N	S & N gen. time	Synthesis time						Solutions found	
						single-cube		general		BDD		gC	SC
						gC	SC	gC	SC	gC	SC	gC	SC
pe-rcv-ifc	65	76	77	274	20.5	28	43	fail	fail	71	319	2e68	1e64
pe-send-ifc	117	213	78	315	40	48	73	2300	fail	130	752	2e51	7e46
trimos-send	336	888	29	480	163	fail	fail	970	fail	268	2e4	4e9	6e8
xyz	8	10	1	20	0.7	fail	fail	212	230	3.8	15	384	288
etlatch	93	206	40	230	27	fail	fail	510	4e3	131	2e3	4e10	2e10
scsiSVT	15	20	10	59	2.8	4	4	230	230	5.4	18	2e5	3e4
slatch	30	46	35	147	8.3	8	10	440	720	22	68	2e17	7e15
elatch	37	61	38	178	11	10	12	460	860	30	87	3e14	5e12
JSPslatch	30	46	35	141	8.1	8	10	440	720	21	65	2e17	7e15
JSPelatch	37	61	38	186	10.7	10	13	450	866	30	93	3e14	5e12
mmuopt	46	90	27	159	13	10	12	360	550	19	105	7e11	8e8
CTRL	116	169	109	510	86	57	81	2e3	fail	90	680	2e37	1e35
SEL	53	89	62	309	28	17	21	740	4e3	48	300	4e31	8e26
SELOpt	73	302	45	247	168	21	28	730	fail	46	247	8e27	8e23
srдор	18	22	5	28	2.6	3	8	190	190	2.7	8	8e3	82
srдand	10	50	4	36	2.7	4.2	4.5	150	155	2.6	8	512	96
srдаoi	19	31	14	99	5.7	4.4	4.8	fail	215	8	27	8e6	3e5
cnt2	24	40	10	60	4.4	7	8	440	540	8.6	27	1e6	9

References

- [1] P. A. Beerel, C. J. Myers, and T. H.-Y. Meng. Automatic synthesis of gate-level speed-independent circuits. Technical Report CSL-TR-94-648, Stanford University, November 1994.
- [2] W. Belluomini and C. J. Myers. Efficient timing analysis algorithms for timed state space exploration. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 88–100, April 1997.
- [3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, Aug. 1986.
- [4] S. M. Burns. General condition for the decomposition of state-holding elements. In *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems*, April 1996.
- [5] B. Coates, A. Davis, and K. Stevens. The Post Office experience: Designing a large asynchronous chip. *Integration, the VLSI journal*, 15(3):341–366, Oct. 1993.
- [6] S. B. Furber, P. Day, J. D. Garside, N. C. Paver, and J. V. Woods. A micropipelined ARM. In *VLSI '93*, 1993.
- [7] L. Lavagno. *Synthesis and Testing of Bounded Wire Delay Asynchronous Circuits from Signal Transition Graphs*. PhD thesis, U.C. Berkeley, Nov. 1992. Technical report UCB/ERL M92/140.
- [8] A. Marshall, B. Coates, and P. Siegel. Designing an asynchronous communications chip. *IEEE Design & Test of Computers*, 11(2):8–21, 1994.
- [9] A. J. Martin. Programming in VLSI: from communicating processes to delay-insensitive VLSI circuits. In C. Hoare, editor, *UT Year of Programming Institute on Concurrent Programming*. Addison-Wesley, 1990.
- [10] C. J. Myers. *Computer-Aided Synthesis and Verification of Gate-Level Timed Circuits*. PhD thesis, Stanford University, 1995.
- [11] C. J. Myers and T. H.-Y. Meng. Synthesis of timed asynchronous circuits. *IEEE Transactions on VLSI Systems*, 1(2):106–119, June 1993.
- [12] C. J. Myers, T. G. Rokicki, and T. H.-Y. Meng. Automatic synthesis of gate-level timed circuits with choice. In *16th Conference on Advanced Research in VLSI*, pages 42–58. IEEE Computer Society Press, 1995.
- [13] E. Pastor and J. Cortadella. Polynomial algorithms for the synthesis of hazard-free circuits from signal transition graphs. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 250–254. IEEE Computer Society Press, Nov. 1993.
- [14] C. K. van Berkel, R. Burgess, J. Kessels, A. Peeters, M. Roncken, and F. Saeijs. A fully-asynchronous low-power error corrector for the digital compact cassette player. In *IEEE International Solid-State Circuits Conference*, 1994.
- [15] A. V. Yakovlev, A. Petrov, and L. Rosenblum. Synthesis of asynchronous control circuits from symbolic signal transition graphs. In S. Furber and M. Edwards, editors, *Asynchronous Design Methodologies*, volume A-28 of *IFIP Transactions*, pages 71–85. Elsevier Science Publishers, 1993.
- [16] H. Zheng and C. J. Myers. Specification and compilation of mixed-timed systems using vhdl. forthcoming paper.