Cycle-based Simulation with Decision Diagrams

Raimund Ubar[1], Adam Morawiec[2], Jaan Raik[1]

[1] Tallinn Technical University, Estonia
Department of Computer Engineering
Raja 15, EE0026 Tallinn

[2] TIMA Laboratory
Modelisation et Verification des Systemes Digitaux
BP 53, 46, Avenue Felix Viallet, 38041 Grenoble Cedex 9, France

Abstract

This paper addresses the problem of efficient functional simulation of synchronous digital systems. A technique based on the use of Decision Diagrams (DD) for representing the functions of a design at RT and behavioural level is introduced. The DD evaluation technique is combined with cycle based simulation mechanism to achieve the significant speed up of the simulation execution. Experimental results are provided for demonstrating the efficiency gain of this method in comparison to the event-driven simulation.

1. Introduction

Growing complexity of the nowadays designs fostered by the progress in the technology requires analogous increase in the design productivity. Several ways to augment the productivity are explored in the research. Among them, shift to the higher levels of automation through the introduction of e.g. high level synthesis, belongs to the most important ones.

Associated with each design action is the appropriate verification step to be performed to ensure the proper functionality of the final design. Here, two complementary methods are developed: simulation and formal verification. Former allows the observation of the behaviour of the design under specific conditions and input vectors while the latter permit to prove that the specific properties of the design hold for all possible input vectors and internal states (in the case of model checking).

In each of those verification approaches the complexity of the verified design is a critical issue. Event driven simulation requires to handle the signal drivers for all signals and ports of the circuits and to update their values during the simulation cycles. This mechanism allows for detailed results of the behaviour of the circuit but is computationally expensive what in consequence increase the time of the simulation execution.

Several methods shortly discussed in the next section have been proposed to overcome the problem posed by the circuit complexity: e.g. abstraction mechanisms or the application of BDDs or branching programs for fast discrete function evaluation. However, none of these methods has applied decision diagrams as a representation of design functionality at the higher level of abstraction (going up to the behavioural level).

This paper focuses on the application of decision diagrams (DDs) also called alternative graphs to represent the functionality of the synchronous system at the RT or behavioural level and combine them with the cycle-based simulation paradigm to improve the simulation speed. DDs are built separately for data-path and control path of the design. In the simulation step DDs are evaluated using the input and previous state values in order to determine the next state and output function of the design.

In our work we focus on hardware description language based design. In particular our work is done with the use of VHDL, however the methods are sufficiently general to be applied to other languages (e.g. Verilog).

The improvement of the efficiency of simulation has been addressed in two different ways: by application of various abstraction mechanisms and by utilisation of binary decision diagrams.

Gruenbacher et. al. [1] proposes abstraction methods for improving the simulation efficiency while raising the level at which the design is represented. Cycle-based simulation is treated as one of possible timing abstraction. This work does not apply representation by decision

diagrams for efficient function evaluation. McGeer et al. [2] applied MDD representation for bit vector discrete function representation and evaluation for logic level cycle-based simulation. Another approach based on BDD and branching programs is presented P. Ashar and S. Malik [3] for efficient logic function evaluation. As the previous approach it can be applied for the simulation at the logic level. The paper [4] presents a cycle-based simulation technique for synchronous circuits which combines a BDD-based logic level cycle simulator with fast hierarchical direct evaluation of high-level functional units stored in a library. The mentioned approaches using BDDs, focus uniquely on the logic level simulation and are not addressing the higher level of abstraction.

The paper is organised as follows: section 3 presents the related works in the domain of simulation performance improvement. In the section 4 decision diagrams are described. The following section presents the application of decision diagrams to the representation of the sequential circuits. The cycle based simulation mechanisms while using DDs is described in the section 6. Some results from practical experiments are presented in section 7 followed by the conclusions and references.

2. Decision Diagrams

Consider a component (subnetwork) f of a digital sytem S as a function $y=f(x)$ where $y=(y_1,\ldots y_n)$ and $x=(x_1,\ldots x_m)$ are vector variables. The function f is defined on $X=X_1\times\ldots\times X_m$ with values $y \in Y = Y_1\times\ldots\times Y_n$, and both, the domain X and the range Y are finite sets of values. x_i, $i = 1,2,\ldots m$, are input or state variables of the component f, whereas y_j , $j = 1,2,\ldots n$, are output or next state variables. The values of variables may be Boolean, Boolean vectors, integers. For representing functions $y = f(x)$ the decision diagrams are used [6, 7].

Definition 1. A DD is a directed acyclic graph $G=(M,\Gamma,x)$ where M is a set of nodes, Γ is a relation in M, and $\Gamma(m)\subset M$ denotes the set of successor nodes of $m\in M$. The nodes $m\in M$ are marked by labels $x(m)$. The labels can be: variables x_i, algebraic expressions of x_i, or constants.

For nonterminal nodes m, where $\Gamma(m) \neq \varnothing$, an onto function exists between the values of $x(m)$ and the successors $m^e\in\Gamma(m)$ of m. By m^e we denote the successor of m for the value $x(m)=e$. The edge (m, m^e) which connects nodes m and m^e is called *activated* iff there exists an assignment $z(m)=e$. Activated edges which connect m_i and m_j make up an *activated path* $l(m_i,m_j)$. An activated path $l(m^0,m^T)$ from the initial node m^0 to a terminal node m^T is called *full activated path*.

Definition 2. Decision Diagram $G_y=(M,\Gamma,x)$ represents a function $y = f(x)$ iff for each value x, a full path in G_y to a terminal node m^T is activated, where $x(m^T) = y$ is valid.

As an example, a subnetwork of a digital system and its DD are depicted in Figure 1. Here, R_1 and R_2 are registers

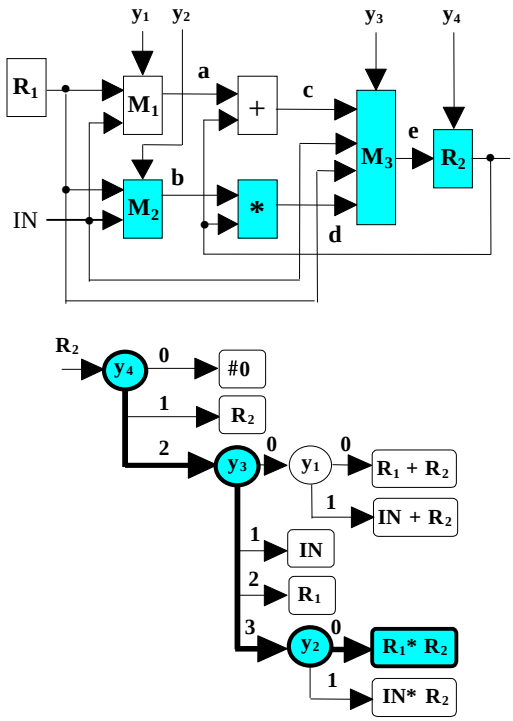

Figure 1. Decision Diagram for a subnetwork

(R_2 is also output), M_1, M_2 and M_3 are multiplexers, + and * denote adder and multiplier, IN is input bus, y_1, y_2, y_3 and y_4 serve as input control variables, and a,b,c,d,e denote internal buses. In the DD, the control variables y_1, y_2, y_3 and y_4 are labelling internal decicion nodes of the DD with their values shown at edges. The terminal nodes are labelled by constant $\#0$ (reset of R_2), by word variables R_1 and R_2 (data transfers to R_2), and by expressions related to data manipulation operations of the network. By bold lines and coloured nodes, a full activated path in the DD is shown from $x(m^0)=y_4$ to $x(m^T)=R_1*R_2$, which corresponds to the pattern $y_4=2$, $y_3=3$, and $y_2=0$. By coloured boxes, the activated part of the network at this pattern is denoted.

3. Design representation

Consider a system $S=(F,N)$ as a set F of components (or subnetworks) represented by functions $y=f(x)$ and a network N connecting these components. The system is represented by a set of variables $Z=\{IN,OUT,INT,REG\}$ defined by relationships of component functions $f\in F$. Here IN, OUT, INT and REG represent correspondingly the sets of primary input, primary output, internal bus and system state (register) variables. The set of components can be divided into control part F_C and datapath F_D, $F = F_C \cup F_D$.

Definition 3. A set of DDs $\{G_y\}$ represent a digital system $S=(F,N)$ if for each function $y=f(x)$ in F there exists a graph $G_y=(M,\Gamma,x)$. The set $\{G_y\}$ is called DD-model for the system S.

Note, in the DD-model we do not have the network *N* explicitly given. In the DD-model we suppose that two variables connected through the network *N* have the same name. In other words, the set $\{G_y\}$ of the DD-model represents a set of graphs connected by variables.

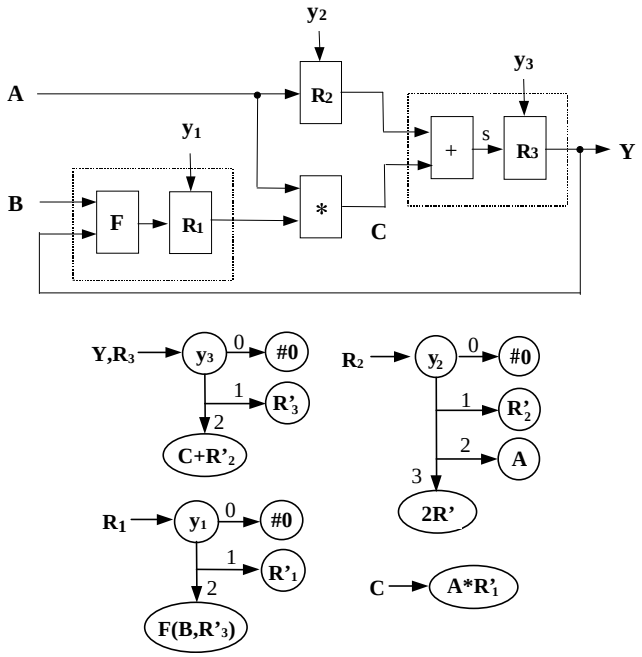

Figure 2. DDs for a datapath of a digital system

To generate DDs for components of the datapath, at first, the DD for the control logic of the component should be created. If the binary level will be implemented in the DD-model, the methods for creating structural BDDs [6] can be used. In that case, each node in the graph will represent a signal path in the gate-level control logic. Hence, the structure of the control circuit will be represented in terms of signal paths in the DD-model. If the RT-level graphs are to be created, we consider higher level (integer) variables which represent control fields of instructions, microinstructions or control buses. By using control variables, a DD is built up with one or more decision nodes in each path of the graph, and, in general, with more than two output edges from each decision node. To each decision node, a multiplexer or decoder as the structural part of the module corresponds. After creation of the decision part of the DD, all the paths in the model will terminate with nodes labelled by expressions for data transfer, data manipulation or constants.

As an example, a datapath and its corresponding DD-model are depicted in Figure 2. The DD-model consists of graphs G_{R2}, G_C, G_{R1}, G_{R3} for representing the functions of register R_2, multiplier C and two sub-networks R_1 and R_3, surrounded by dotted lines. In this example, y_1, y_2, and y_3 serve as control inputs, A and B are data inputs, R_1, R_2, and R_3 serve as data register variables (by apostrophe the previous state is denoted), C is the output variable of the multiplier and input variable for the adder, and Y is the primary output of the datapath. In nonterminal nodes, only control variables are used as labels. Terminal nodes are labelled by data transfer, data manipulation expressions or constants (reset). Each node has a strong relation to the structure of the datapath: nonterminal nodes represent the control logic in modules (subnetworks), the nodes labelled by data variables represent buses, and the nodes labelled by expressions represent data manipulation logic.

As to the control part of the system, we generate DDs for the output and the next-state behaviour for each finite state machine (FSM). In the case of Moore automata, both DDs can be joined. As labels for the decision nodes, input and previous state variables of the FSM are used. Each pattern for these variables prescribes a path through the decision tree which would terminate with a node labelled by expression (or constant) to define the next state and the output behaviour of the FSM. In Figure 3, a FSM for controlling the data path in Figure 2 is depicted. To represent the FSM, a DD is created for the vector function: $q,y_1,y_2,y_3 = f(q',R'_2=0)$. The predicate $R'_2=0$ is used here to represent a flag variable for reporting the state of the datapath. We have two decision nodes in the graph for analysing the previous state q' of the FSM and the flag $R'_2=0$. The terminal nodes are labelled by constants (the values to be assigned to the vector variable q,y_1,y_2,y_3).

q'	x	q	$y_1\ y_2\ y_3$	Activities
0		1	1 1 0	y_3:R_3 := 0
1		2	2 2 1	y_1:R_1 := B, y_2:R_2 := A
2	$R'_2=0$	0	1 1 2	y_3:R_3 := A∗B
2	$R'_2\neq 0$	2	1 2 1	y_2:R_2 := A

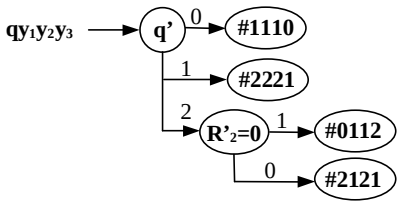

Figure 3. DD-model for the control part

For compression of the model, the graph superposition procedure proposed in [6] for gate-level strucural graph synthesis can be generalized for the RT-level case. An example of a superpositioned DD for the given subnetwork is shown in Figure 1. Since the bus variables *a,b,c,d,e* are disappeared from the DD, the complexity of the model is reduced which helps to speed the cycle-based simulation.

4. Cycle-based simulation

Cycle-based simulation of synchronous digital systems is performed on a cycle-by-cycle basis. It assumes that

there exist (one or many) clock signals in the circuit and all inputs of the systems remain unchanged while evaluating their values in the simulation cycle. The results of simulation report only the final values of the output signals in the current simulation cycle.

In the event driven simulation each computation is proportional to the number of signals both internal and external in the design (number of drivers and updates). In DDs, only the changes of signals on which depend the outputs are used for evaluation what allows to compute only the necessary information without the overhead associated with evaluation of intermediate values.

The idea of the cycle-based simulation on DDs is the following. The DDs are ranked in such a way that when a DD is simulated his arguments should be all either specified or calculated. So, the simulation starts with DDs that depend only on input or state variables, i.e. they are specified either by the input pattern or by the previous state (from the previous clock cycle). Afterwards, also these DDs which depend on the internal variables which, however, have been already calculated in the current cycle, are simulated. In such a way there is a need to assess the output value of each DD, but the evaluation is performed only one time in a cycle. In the event driven simulation, each change of a signal necessitates the recalculation of other signals dependent on it. So, in one simulation cycle there might be several consecutive recalculations of the same signal what is very costly in terms of execution time.

In DDs during simulation, not all nodes are traced. Only the arguments of traversed nodes are required for counting. From that aspect, additional gain in simulation speed is achieved with DDs in comparison to other simulation models. As an example in Figure 1 during the simulation for the control pattern $y_4=2$, $y_3=3$, $y_2=0$, only a path through nodes y_4, y_3, y_2 should be traced with calculation $R_2=R_1*R_2$. This is the maximum length $(L=3)$ of the path traversed in the DD. The shortest one is $L=1$, and the average (for the case when the probabilities of values of y_i are equal) $L=2/3*1+1/3*(1/2*2+1/2*3)=1,5$. In the traditional case of network simulation, always the following sequence of operations should be calculated: $a=f_1(y_1,R_1,IN)$, $b=f_2(y_2,R_1,IN)$, $c=a+R_2$, $d=b*R_2$, $e=f_3(y_3, c,d,R_1,IN)$, $R_2=f_4(y_4,e,R_2)$.

In [8] a method was presented for synthesis of DDs from VHDL where the fine-grained timing is replaced by a coarse timing, which helps to get rid of unnecessary details from the model not needed in cycle based simulation.

An example [8] of a VHDL description and its cycle-based DD-model is represented in Figure 4 to illustrate the simulation procedure. For the first three processes of the VHDL description, DDs for calculating *state, enable_in, reg_cp* are created, whereas for the last process, DDs for *next-state, outreg, fin,reg_cp_com reg* are created. After superpositioning [8] of the model, only two DDs remain.

Simulation results of 6 clock cycles on these DDs are depicted in Table 1. The paths traced on DDs for the first cycle are shown by coloured nodes in Figure 4. Only a part of the whole model (two decision nodes instead of eight) was processed, and no timing data (clock event variables like falling and rising edges) are used in calculation, which results in a high speed of simulation in general.

```
entity rd_pc is
  port
    ( clk, rst : in bit; rb0 : in bit; enable : in bit; reg_cp : out bit ;
    reg : out bit ; outreg : out bit ; fin : out bit ) ; end rd_pc ;
architecture archi_rd_pc of rd_pc is type STATETYPE is
  (state1, state2);
  signal state, nstate: STATETYPE ; signal enable_in : bit ;
  signal reg_cp_comb : bit ;
begin
  process(clk, rst) - - process P(state)
  begin
    if rst='1' then state <= state1 ;
    elsif (clk'event and clk='1') then state <=  nstate ;
  end if ; end process ;
  process(clk, enable) - - process P(enable_in)
  begin
    if clk='1' then   enable_in <= enable ;
  end if ; end process ;
  process(clk, reg_cp_comb) - - process P(reg_cp)
  begin
    if clk='0' then   reg_cp <= reg_cp_comb ;
  end if ; end process ;
  comb: process (state, rb0, enable_in) - - process P(nstate, out)
  begin
    case state is
      when state1 => outreg <= '0' ; fin <= '0' ;
        if (enable_in='0') then   nstate <= state1 ;
                    reg <= '1' ; reg_cp_comb <= '0' ;
                else  nstate <= state2 ; reg <= '1' ;
                    reg_cp_comb <= '1' ; end if ;
      when state2 =>
        if (rb0='1') then   nstate <= state2 ; reg <= '0' ;
                  reg_cp_comb <= '1'; outreg <= '0'; fin <= '0';
          elsif (enable_in='0') then   nstate <= state1 ; reg <= '0' ;
                  reg_cp_comb <= '0'; outreg <= '1'; fin <= '1';
                  else  nstate <= state2 ; reg <= '0' ;
                  reg_cp_comb <= '0'; outreg <= '0'; fin <= '1';
end if ;  end case ;  end process ;  end archi_rd_pc ;
```

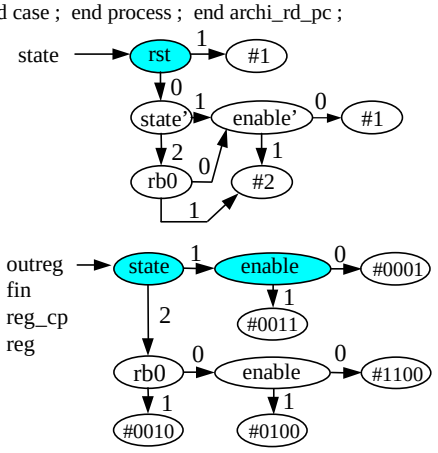

Figure 4. DD-model for the VHDL design

To fully exploit the advantage provided by the separation of the combinational part of the design from the

sequential part the dependency analysis can be performed on the combinatorial blocks. This is done in order to find the best ordering of the evaluation of blocks to ensure that the outputs of the blocks are calculated only when necessary to avoid an overhead in evaluation.

cycle	1	2	3	4	5	6
rst	1	0	0	0	0	0
enable	0	1	1	1	0	0
rb0	x	x	1	0	0	0
state	1	1	2	2	2	1
outreg	0	0	0	0	1	0
fin	0	0	0	1	1	0
reg_cp	0	1	1	0	0	0
reg	1	1	0	0	0	1

Table 1. Simulation results for 6 clock cycles

5. Experimental results

A prototype of a simulator based on the decision diagrams representation and cycle-based paradigm has been tested on benchmark circuits described in Table 3.

GCC and DIFFEQ are the HLSynth benchmarks, MULT8X8 is an 8-bit multiplier using Robertson's algorithm. The experiments have been performed on Pentium II 200MHz computer with 128MB RAM running Windows NT. For experiments each circuit is simulated for 200000 random input vectors. Execution time is measured using internal system procedure *GetProcessTimes*. The simulators used for experiments are ModelSim PE/Plus 4.7b and VeriBest VHDL SysSim 98.0. The improvement in the simulation performance while using DD based approach over event-driven simulators ranges between 3 to 9 times for the faster simulator and 9 to 37 times in the case of the slower simulator for measured benchmarks. The detailed results are presented in the Table 3. All results are in seconds; "Ratio" is the event-driven simulation time to DD-based simulation time ratio.

Circuit	Inputs	Outputs	Control States	Complexity	
				Gates	F/F
GCD	10	4	8	227	15
DIFFEQ	82	32	6	4195	115
MULT8X8	18	16	8	1058	95

Table 2. Benchmark circuits

6. Conclusions

A new conceptual approach based on high-level decision diagrams for simulation of digital systems was developed. The approach allows a uniform graph based description of VLSI designs on different levels of abstraction with uniform simulation procedures based on path tracing on DDs. DDs allow to uniformly describe a wide class of digital systems on mixed logical and functional levels. This class contains random logic, traditionally treated at the gate level, as well as complex digital circuits like microprocessors, controllers etc., traditionally described at the procedural or RTL levels. Unlike the HDL-based descriptions, DDs give an excellent formal basis for diagnostic analysis of digital systems, and allow to create more efficient CAD tools than event-driven HDL-based simulators for functional simulation as well as for fault analysis and testing purposes. Due to the fact that only a part of DDs should be traced during simulation and due to neglecting the time specific information inherent in HDL descriptions, the speed of simulation on DDs can be drastically increased in comparison to traditional event-driven simulators.

References

[1] H. Gruenbacher, M. Khosravipour, G. Gridling: "Improving Simulation Efficiency by Hierarchical Abstraction Transformations", System Level Design Language Workshop, Lausanne, Switzerland, 1998

[2] P. McGeer et al: "Fast Discrete Function Evaluation Using Decision Diagrams", ICCAD'95, pp 402-407

[3] P. Ashar, S. Malik: "Fast Functional Simulation Using Branching Programs", ICCAD Conf, 1995, pp 408-412

[4] Y. Luo, T. Wongsonegoro, A. Aziz: "Hybrid Techniques for Fast Functional Simulation", DAC 1998

[5] S.P. Smith, J. Larson : "A High Performance VHDL Simulator with Integrated Switch and Primitive Modelling ", Computer HD Languages and their Applications, Elsevier, IFIP, 1990

[6] R.Ubar. "Test Synthesis with Alternative Graphs.", IEEE DesignTest of Comput., Spring 1996, pp.48-57

[7] J.Raik, R.Ubar. Sequential Circuit Test Generation Using Decision Diagram Models. Published in the same Proceedings.

[8] R. Leveugle, R. Ubar. "Synthesis of DDs from Clock-Driven Multi-Process VHDL for Test Generation.", MIXDES'98, Lodz, Poland, June 1998, pp.353-358.

Circuit	DD based simulator	VHDL Simulator 1			VHDL Simulator 2		
		Compilation	Simulation	Ratio	Compilation	Simulation	Ratio
GCD	3.89	0.63	13.13	3.4	3.58	33.98	8.7
DIFFEQ	8.96	0.69	81.51	9.1	3.81	331.62	37.0
MULT8X8	5.79	0.72	25.38	4.4	4.18	64.81	11.2

Table 3. Experiments with cycle-based DD-model for the VHDL