Efficient Switching Activity Simulation Under a Real Delay Model Using a Bitparallel Approach

M. Bühler, M. Papesch, K. Kapp, U.G. Baitinger
ISE - IPVR, University of Stuttgart, Breitwiesenstr. 20-22, 70565 Stuttgart
Phone: +49 711 7816 216, Fax: +49 711 7816 250
Email: markus.buehler@informatik.uni-stuttgart.de

Abstract

Estimating switching activity is a crucial step in optimizing circuits for low power. In this paper, a fast gate level switching activity estimator for combinational circuits will be presented. The combination of event driven and bitparallel simulation allows for high accuracy due to the real delay model of the former while maintaining the speedup of the latter. This is demonstrated by detailed experimental results.

1 Introduction

Today power optimization and thus power estimation has become a major objective in digital circuit development. Charging and discharging of circuit nodes is the main source for power dissipation in digital CMOS circuits [1]. The average power dissipation is given by:

$$P = \sum_{\text{all nodes } i} \frac{1}{2} \cdot V_{DD}^2 \cdot C_{Li} \cdot f \cdot p_{Si} \qquad (1)$$

V_{DD} denotes the supply voltage, C_{Li} the sum of all parasitic capacitances attached to node *i*, *f* the clock frequency the circuit is operating at, and p_{si} the average number of output transitions per clock cycle of the gate driving node *i* [2].

While all other parameters are given by the technology or the circuit layout [2], p_{si} not only depends on the circuit structure but also on the statistical properties of the input signals applied to the circuit. Thus, the major objective of this paper is to efficiently estimate the switching activity p_{Si}.

Today, two main approaches exist to estimate switching activity on the logic level [3]: pattern simulation and probabilistic methods. The former relying on typical input patterns which are either known from high level simulation or randomly generated using the statistics of the primary inputs. The latter directly uses those statistics and propagates them through the circuit by symbolic simulation.

In pattern simulation the main problems are accuracy and runtime. Since accurate estimation requires a high number of input patterns, runtime increases with accuracy. Thus, most work in this field has been done in limiting the number of input patterns through e.g. Monte Carlo methods [4] and in improving runtime behavior [3, 5, 6].

The difficulties in symbolic simulation are related to accurately handling correlations introduced by the primary inputs or reconvergent fanout. Most methods to handle correlation are based on BDDs. However, they can become very large for large circuits. Advanced techniques can be found in [7] and [8].

This paper deals with speeding up pattern-based simulation using the bitparallel approach first published in [3]. The rest of the paper will be organized as follows: chapter 2 gives the most important definitions and reflects previous work on bitparallel simulation. In chapter 3 a new simulation method will be presented. It has been implemented in the computer program *TESA* (**T**ime parallel **E**stimation of **S**witching **A**ctivity). Chapter 4 summarizes the results that have been obtained by comparing *TESA* to bitwise simulation.

2 Preliminaries

2.1 Previous Work

In the near past, two approaches of bitparallel simulation for switching activity estimation have been proposed. The first one relies on exploiting the whole width of the processor word of the machine the simulator is running on [3]. Instead of simulating each clock cycle separately, 32 or 64 cycles are simulated at a time (fig. 1). Speedups between 2 and 74 compared to single bit simulation have been reported in [3] on a 64 bit machine.

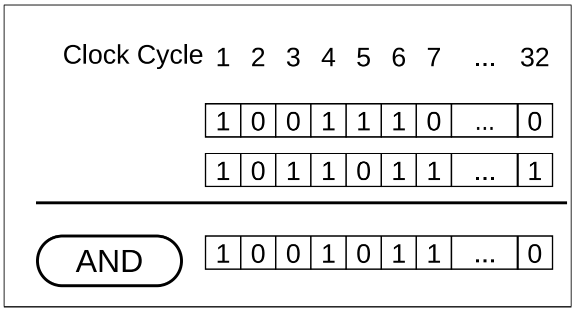

Figure 1: Wordwise AND

A second method has been proposed in [6], where the signals are represented as sets. A set is a sequence of ones. Only the number of the first and the last clock cycle of a sequence are stored. Thus all logic operations become set operations.

clk	1	2	3	4	5	6	7	8	9	...							
Bits	0	1	1	0	0	0	1	1	1	1	1	0	0	1	1	1	1
Blocks		2-3					8-12							14-17			

Figure 2: Signal representation with sets

This method is only efficient if there exist many signals with low activity, like the flip flop outputs in sequential circuits.

In the original publications both approaches have been restricted to the zero delay model (ZDM) which can result in important inaccuracies as will be shown in chapter 4. In [9] a first approach has been presented to extend bitparallel simulation by a real delay model (RDM). However, parallelism has only been applied to the logic operations. Transition, glitch and hazard detection is still performed sequentially, thus resulting in only a limited speedup compared to bitwise simulation. In the sequel a new method will be presented that exploits parallelism for all operations.

2.2 Signal Representation

Bitparallel signal representation is clock cycle oriented. It is assumed that a signal takes only one value per clock cycle. However, in real applications with non-zero delays, any internal signal of a circuit may change its value several times during the clock cycle due to different arriving times of the input signals. Figure 3 shows such a signal. Note, that Δt is the simulation time, i.e. the time that has passed since the beginning of the current clock cycle.

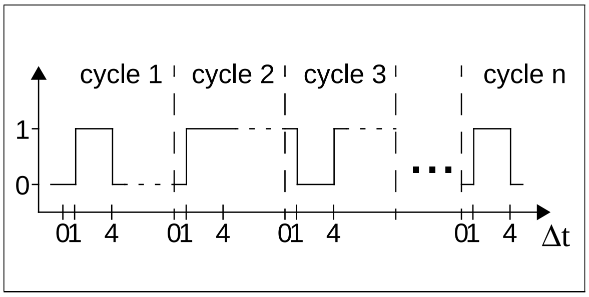

Figure 3: Internal signal in a RDM

It is obvious, that the signal representations of figures 1 and 2 cannot take into account the additional signal changes during the clock cycles. Hence, each signal waveform, traditionally being represented as a vector, has been extended to an array [9]. In this array, each line depicts the signal values at a specific time during the clock cycles (fig. 4). In the sequel this array will be called the *schedule* of a signal and the lines are referred to as vectors. E.g. $y_1 = 110\ldots1$ denotes row 1 in figure 4.

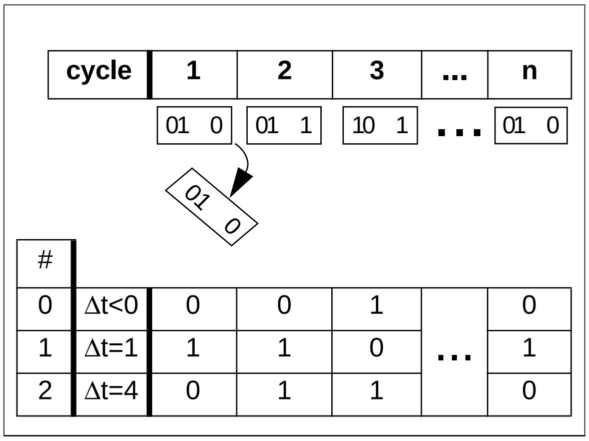

Figure 4: Bitparallel signal representation

Row 0 with Δt<0 denotes the signal value at the end of the last clock cycle, the steady state. Each line can now be represented using either word or set representation and all logic operations can be performed in parallel on complete lines.

2.3 Definitions

The use of the terms *transition*, *hazard* and *glitch* is sometimes contradictory in the existing literature. In the sequel, a *transition* denotes a complete signal change of a node either from 0 to 1 or from 1 to 0. If the time between two successive transitions is to short, e.g. below the gate's delay, the output doesn't reach its full signal swing. Such an incomplete transition will be called a *glitch*. A *hazard* is a transition that is caused by different arriving times of the signals at a gate's inputs.

e_i denotes an event or transition. e_i^r and e_i^f indicate rising and falling events, respectively. Finally e_i^x is the collective term for e_i^r and e_i^f. Of course, all events are represented as vectors.

3 The Simulation Algorithm

Taking the signal representation of chapter 2.2, the simulation algorithm can be divided into four steps:

```
For all gates:
  1.perform logic operation
  2.delay determination and scheduling
  3.glitch removal
  4. count transitions and hazards
```

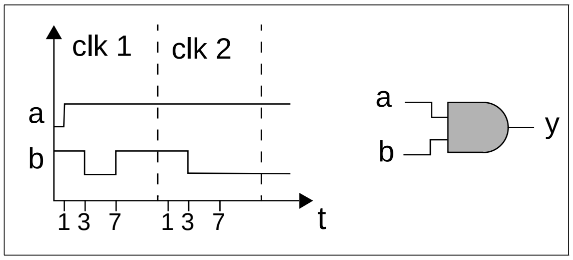

Figure 5: AND gate with input waveforms

In order to explain the algorithm in detail the AND-gate of figure 5 will be used as an example. Figure 6 shows the the first two clock cycles of the schedules of its input waveforms.

#	cycle	1	2
0	Δt<0	0	1
1	Δt=1	1	1

Signal a

#	cycle	1	2
0	Δt<0	1	1
1	Δt=3	0	0
2	Δt=7	1	0

Signal b

Figure 6: Schedules of the input waveforms

3.1 Logic Operation

First of all, the initial values of the output *y* is computed by performing the logic operation of the AND gate on its initial input values. The result is entered into the first line of the output schedule as event 0 (Δ*t<0*) in figure 7:

$$y_0 = a_0 \cdot b_0 = 01 \cdot 11 = 01 \qquad (2)$$

3.2 Delay Determination and Scheduling

Then Δt proceeds to the first event on the inputs which occurs at Δ*t=1*. Again, the logic operation is performed:

$$y_1 = a_1 \cdot b_0 = 11 \cdot 11 = 11 \qquad (3)$$

The result, however, cannot directly be entered into the output schedule, since the different delays for falling and rising edges must be taken into account, t_{down} and t_{up} respectively. Hence, the directions of the transitions must be determined using the following equations:

$$\text{event: } e_i = y_i \oplus y_{i-1} \qquad (4)$$

$$\text{falling event: } e_i^f = e_i \cdot y_{i-1} \qquad (5)$$

$$\text{rising event: } e_i^r = e_i \cdot y_i \qquad (6)$$

Now, the correct values for the delays can be applied. For the following processing, it is convenient to temporarily enter e_i^x into the schedule instead of the logic values y_i. Thus the schedule contains only events except for row 0 that holds the initial values of the clock cycles. The events are marked with an asterisk.

event #	**cycle** / **Δt**	1	2
0	<0	0	1
1^*	1+t_{up}	1	0

Figure 7: Output schedule of y at Δt=1

In the example, only $e_1^r = 10$ yields a non-zero result. It will be entered into the output schedule at time Δ$t=1+t_{up}$. Since this is the first event on signal *y*, no glitch detection is necessary.

3.3 Glitch Removal

Now simulation time Δ*t* proceeds to the next input event which occurs at Δ*t=3* on input *b*. Again, the logic operation is performed, resulting in:

$$y_2 = a_1 \cdot b_1 = 11 \cdot 00 = 00 \qquad (7)$$

Equations 5 and 6 indicate no rising edge but two falling ones: $e_2^f = 11$. It will be scheduled at Δ$t=3+t_{down}$. Before this new value can be entered into the output schedule, glitches are neglected and must be filtered out. There are two conditions that must be fulfilled so that the current and a previously scheduled event e_i^x and e_j, respectively, cause one or more glitches:

1. e_j has not taken place yet: its scheduled time Δt_{e_j}

is later than the actual simulation time: $\Delta t_{e_j} > \Delta t$

2. e_i^x shows at least one event in the same clock cycle (same column) as e_j.

In that case, some columns in e_j and e_i^x compensate each other. Those are filtered out using the following equations:

$$e_j' = (e_j \oplus e_i^x) \cdot e_j \tag{8}$$

$$e'^x_i = (e_j \oplus e_i^x) \cdot e_i^x \tag{9}$$

Equation 8 must be iteratively applied to all events e_j with scheduled time $\Delta t_{e_j} > \Delta t$. In each iteration e_i^x must be updated using equation 9. Fortunately, these operations are not very time consuming, since there are usually only few events e_j with $\Delta t_{e_j} > \Delta t$.

In the example the decision whether there are glitches or not depends upon t_{up}. If $1+t_{up}<3$, there is no glitch resulting in the new schedule of figure 8.

event #	cycle / Δt	1	2
0	<0	0	1
1*	$1+t_{up}$	1	0
2*	$3+t_{down}$	1	1

Figure 8: Schedule at Δt=3 if t_{up}<2

Otherwise, if $1+t_{up} \geq 3$, equation 8 is applied to event e_1 resulting in:

$$e'_1 = (e_2^f \oplus e_1) \cdot e_1 = 00 . \tag{10}$$

That means e_1 can be removed. e_2^f becomes:

$$e'^f_2 = (e_2^f \oplus e_1) \cdot e_2^f = 01 . \tag{11}$$

Figure 9 shows the resulting schedule. Note, that the events in the schedule have no direction attribute. It should be emphasized, that all the logic operations in the equations 5-9 can be performed using one of the parallel methods outlined in chapter 2.1. Thus maintaining the runtime advantages of both methods.

event #	cycle / Δt	1	2
0	<0	0	1
2*	$3+t_{down}$	0	1

Figure 9: Schedule at Δt=3 if t_{up}≥2

3.4 Counting Transitions and Hazards

After all input events have been processed the numbers of transitions tr_y and hazards tr_h of the output y can be counted. In order to do so it must be distinguished between the set and the word approach. In the former, the schedule with $n+1$ rows (n events plus the initial values) is scanned using equation 12:

$$tr_y = \sum_{\text{row } r=1}^{n} \sum_{\text{for all sets } i \text{ of } r} E_i - S_i + 1 \tag{12}$$

Where E_i and S_i denote the index of the end and the start of set i, respectively.

For the word approach a method relying on a lookup table (LUT) has been proposed in [3]. It is also applied here with some minor modifications.

The LUT performs the assignment of 16 bit values to their hamming weight. Each row in the schedule is split into 16 bit words. Using the LUT the number of events in the words, corresponding to signal toggles, can be determined. Thus, employing equation 13 the total number of transitions tr_y of the output y can be easily summed up.

$$tr_y = \sum_{\text{row } r=1}^{n} \sum_{\text{all words } w \text{ of } r} LUT(w) \tag{13}$$

During the transition counting phase the events are replaced by the corresponding logic values. In order to compute the number of hazards tr_h, useful transitions u are computed first for both approaches:

$$u = y_0 \oplus y_n . \tag{14}$$

Where y_0 and y_n denote the logic values of the steady states at the beginning and at the end of the clock cycles, respectively. The number of useful transitions tr_u is computed using equation 12 or 13. The number of hazards tr_h then results in:

$$tr_h = tr_y - tr_u . \tag{15}$$

4 Experimental Results

The algorithms of chapter 3 have been implemented in the computer program *TESA*. It has been extensively tested on several benchmark circuits from the ISCAS85 and ISCAS89 benchmark sets. On the ISCAS89 benchmarks only the combinational part has been simulated. All simulation runs were performed on a SUN Sparc Ultra 2, 300 MHz, using 32 bit word width. For each circuit 10,000 randomly generated input patterns have been simulated. The average switching activity for the primary inputs was chosen to 0.5. For the sequential circuits the flip flops have been cut out, their outputs have become additional primary inputs. For the sake of realism their activity has been determined using RTL simulation.

Figure 10 summarizes the results. *CPU/s word* is the absolute runtime of the word approach in seconds. The columns *speedup word* and *set* denote the speedup of wordwise and set simulation, respectively, compared to single bit simulation. The latter having been performed with *TESA* in single bit mode in order to avoid influences of implementation differences, different time models etc. of other simulators. The theoretical value of 32 times speedup for wordwise simulation has not been reached on the average due to some overhead during scheduling and glitch detection. But an average speedup of more than 20 is still a good result. For some smaller circuits, the speed-up is even higher than 32. The reason is not absolutely clear yet but it may be caused by unaligned memory accesses in the single bit version. The set approach generally performs worse than wordwise simulation.

A certain variation of the speedup among different circuits can be noticed (e.g. c3540). Obviously it doesn‘t depend on the circuit size. But it was observed that for c3540 the schedules become very large compared to other circuits, resulting in a performance penalty.

circuit	gates	CPU/s	speedup		h/%
		word	word	set	
c432	160	0.9	21.0	2.2	36.8
c880	383	7.1	11.4	2.0	38.6
c1908	880	22.0	12.7	2.2	47.2
c3540	1669	1098.5	1.7	1.6	49.2
c5315	2307	204.7	4.3	1.5	49.5
c6288	2416	37.7	31.8	1.1	97.1
c7552	3512	221.8	4.8	1.4	52.7
s386	168	0.2	42.9	4.8	10.5
s820	299	0.3	39.4	5.4	9.6
s1196	552	3.2	22.9	6.5	20.7
s5378	2961	4.4	30.9	4.8	14.7
s9234	5830	8.5	24.8	8.1	22.3
s13207	8625	6.9	22.8	4.3	17.8
s15850	10374	9.0	18.1	7.0	22.4
average			20.6	3.8	32.3
σ			12.2	2.3	23.1

Figure 10: Results on ISCAS-Benchmarks

Finally h/% denotes the percentage of hazards on all transitions. Due to their average rate of 32% and their high standard deviation, they are not neglible.

5 Conclusion

Accurate switching activity estimation is a crucial step during the design of low power circuits. On gate level, *TESA* offers a very fast estimation option by combining event based simulation and bitparallel approaches. On several benchmark circuits it has proven an average speedup of more than 20 compared to bitwise simulation at the same accuracy. With only slight modifications *TESA* could be used as an ordinary logic simulator as well. The current implementation is limited to combinational circuit and doesn’t take into account the gate loads for delay modeling. An extension to sequential circuits has been presented in [11] for a ZDM simulator. Future work will include the integration of this extension into *TESA* and the extension of the time models.

6 References

[1] Monteiro et al., *Estimation of Average Switching Activity in Combinational Logic Circuits Using Symbolic Simulation*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 16, No. 1, January 1997, pp. 121-127

[2] Najm F., *Transition Density: A New Measure of Activity in Digital Circuits*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 12, No. 2, February 1992, pp. 310-323

[3] P.H.Schneider, *PAPSAS: A Fast Switching Activity Simulator*, PATMOS’95, 1995, pp. 351-360

[4] Burch R. et al., *A Monte Carlo Approach for Power Estimation*, IEEE Transactions on VLSI Systems, vol. 1, no. 1, March 1993, pp. 63-71.

[5] F. Dresig et al., *Simulation and Reduction of CMOS Power Dissipation at Logic Level*, European Conference on Design Automation (EDAC), 1993, pp. 341-346

[6] M.Bühler, D. Dallmann, U. G. Baitinger, *Switching Activity Analysis Using a Set Theoretical Approach, GI/ITG/GME Workshop 1998, Paderborn, ISBN 3-931466-35-3*

[7] Marculescu R., Marculescu D., Pedram M., *Efficient Power Estimation for Highly Correlated Input Streams*, 32nd Design Automation Conference DAC, 1995, pp. 628-634

[8] Chou T., Roy K., Prasad S., *Estimation of Circuit Activity Considering Signal Correlations and Simultaneous Switching*, IEEE/ACM International Conference on CAD-94, 1994, pp. 300-303

[9] Kapp K., Bühler M., Dallmann D., Baitinger U.G., *TESA: Timeparallel Estimation of Switching Activity under a Real Delay Model*, ICECS’98, Lisbon, Portugal, 1998

[10] Chou T., Roy K., Prasad S., *Estimation of Circuit Activity Considering Signal Correlations and Simultaneous Switching*, IEEE/ACM International Conference on CAD-94, 1994, pp. 300-303

[11] N. Gouders, R. Kaibel, *PARIS A Parallel Pattern Fault Simulator for Synchronous Sequential Circuits,* IEEE International Conference on Computer-Aided Design, ICCAD, pp. 542-545, 1991