Optimal Placement of Registers in Data Paths for Low Power Design

Christian V. Schimpfle, Sven Simon and Josef A. Nossek
Institute for Network Theory and Circuit Design
Technical University Munich
Arcisstr. 21, 80333 Munich, Germany
Email: chsc@nws.e-technik.tu-muenchen.de

Abstract– In this paper a new probabilistic approach for determining spurious switching activity in data paths is presented. Spurious switching activity results from logic paths with different propagation delays. The proposed methodology profits from the regularity of the data path structure. A "glitch-weight" is computed for each node such that retiming can be applied to the circuit using glitching activity instead of delays. Retiming thus manages the placement of registers to compensate different path delays. The methodology is especially suited to convert combinational circuits into pipelined data paths. By considering additional timing constraints a design can be optimized in terms of timing and glitching activity simultaniously. Some typical data path examples are given in order to show how to apply the proposed methodology.

I. Introduction

In recent years timing optimization and low power design methodologies have gained a lot of interest. Retiming and pipelining algorithms, usually applied for improving the speed of a design have been found suitable for reducing spurious switching activity (*glitches*) and thereby reducing the power consumption of a circuit [2]. Speed improvements due to retiming can also be used to scale down the supply voltage without a loss in performance [3]. In this paper a probabilistic method for estimating glitching activity in data paths is described. The regularity of data paths is used to simplify the computation of a glitch-weight for all nodes of the circuit. Retiming is then applied exclusively to reduce these glitches. Besides power considerations, timing constraints can be taken into account.

Retiming is only one possible application for the presented glitching activity estimation method. It may also be applicable for comparing different circuit realisations in terms of switching activity as well as for resynthesis [5] procedures.

This paper is structured as follows: In section II the glitching activity estimation method for regular data paths and the determination of the glitch-weights will be described. Section III compares this probabilistic method with an experimental glitch-counting method. In section IV a retiming algorithm specially suited for minimizing spurious switching activity is presented. Section V shows some experimental results, a conclusion is given in section VI.

II. Estimation of Glitching Activity in Data Paths

Glitches at the output of a logic gate can occur when the input signals arrive at different times. The delay since a signal x_i arrives at the input of the gate is $d(x_i)$. For example, consider a full-adder with three inputs x_1, x_2, x_3 and outputs s (sum) and c (carry). For simplicity it is assumed that $P(x_i) = P(\overline{x}_i)$ is true for every input signal, then $P(x_i) = P(\overline{x}_i)$ is also valid for the outputs s and c of the full-adder. On the assumption that the full-adder is considered as one single gate, glitches occur only at the outputs. Let $P_{glitch}^{d(x_1)\neq d(x_2)=d(x_3)}(x_i)$ be the probability that a glitch occurs at x_i if $d(x_1) \neq d(x_2) = d(x_3)$. For statistically independend input signals $x1, x2, x3$ the joint probability is the product of all signal probabilities: $P(x_1)\cdot P(x_2)\cdot P(x_3) = P(x_1x_2x_3)$. Than

$$\begin{aligned} P_{glitch}^{d(x_1)\neq d(x_2)=d(x_3)}(s) &= P(x_1x_2x_3)\cdot P(\overline{x}_1x_2\overline{x}_3) \\ &+ P(x_1x_2x_3)\cdot P(\overline{x}_1\overline{x}_2x_3)) + \cdots \\ &= 16\cdot P^2(x_1x_2x_3) \end{aligned} \quad (1)$$

is the probability that a glitch occurs at the sum output and

$$\begin{aligned} P_{glitch}^{d(x_1)\neq d(x_2)=d(x_3)}(c) &= P(\overline{x}_1x_2x_3)\cdot P(x_1x_2\overline{x}_3) \\ &+ P(\overline{x}_1x_2x_3)\cdot P(x_1\overline{x}_2x_3)) + \cdots \\ &= 4\cdot P^2(x_1x_2x_3) \end{aligned} \quad (2)$$

0-7803-3583-X/97 $10.00 ©1997 IEEE

is the probability that a glitch occurs at the carry output. With the above assumptions it is clear that

$$P_{glitch}^{d(x_1)\neq d(x_2)=d(x_3)} = P_{glitch}^{d(x_2)\neq d(x_1)=d(x_3)} = P_{glitch}^{d(x_3)\neq d(x_1)=d(x_2)}$$

is true for both outputs. If all the three input signals arrive at different times, the probabilities for glitching at the outputs are:

$$p_{glitch}^{d(x_1)\neq d(x_2)\neq d(x_3)}(s) = \cdots = 32 \cdot P^2(x_1x_2x_3)$$
$$p_{glitch}^{d(x_1)\neq d(x_2)\neq d(x_3)}(c) = \cdots = 8 \cdot P^2(x_1x_2x_3). \quad (3)$$

The signals in a circuit can be considered to be uncorrelated if all the states of the circuit are equally likely. Although in general this might not be true, it is a good approximation for data path circuits. Data paths have highly connected signal transition graphs (STGs) which include most of the possible states. All the states have similar in-degrees (number of edges that enter the state). Therefore it is a realistic assumption to consider the inputs of all gates or adders in a data path to be uncorrelated [6].

The graph model used throughout this paper is equivalent with the graph model in [1]. A network contains vertices v connected by directed edges e. Each vertex has its specific propagation delay. An edge that goes from vertex u to vertex v is written $u \xrightarrow{e} v$. Every adder or logic gate in the network will be considered as a vertex. For every vertex a matrix $D(u,x)$ is calculated, where every entry $d(u,x)$ denotes the propagation delay from a primary input u to an input x of the vertex. Fig. 1 shows a network example with four vertices, each with a inertial delay 1 and five primary inputs a,b,c,d,e. Matrix $D(u,x)$ for vertex $v4$ is given by:

$$D(u,x) = \begin{array}{c} a \\ b \\ c \\ d \\ e \end{array} \begin{pmatrix} 1 & \infty & \infty \\ 1 & \infty & \infty \\ \infty & 2 & \infty \\ \infty & 2 & \infty \\ \infty & \infty & 0 \end{pmatrix},$$

where a delay ∞ denotes that there is no connection $u \longrightarrow x$. Now the probability that a input signal

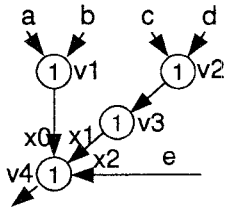

Figure 1: An example for different delayed signals to a vertex (v_4).

to a vertex changes its value can be evaluated as follows: let $P_{change}(y|x)$ be the probability that the output y of a vertex changes if input x changes. For a path k from a primary input u to signal y_i, $k : u \longrightarrow y_1 \longrightarrow y_2 \cdots y_{i-1} \longrightarrow y_i$, the probability that y_i changes if u changes is:

$$P_{change,k}(y_i|u) = P_{change}(y_1|u) \cdot P_{change}(y_2|y_1) \cdot P_{change}(y_3|y_2) \cdots P_{change}(y_{i-1}|y_i) \quad (4)$$

and further:

$$P_{change}(y_i|u) = \sum_k P_{change,k}(y_i|u). \quad (5)$$

Now matrix $D(u,x)$ of a vertex v is analysed. Glitches may occur if for any columns $x = \mu$ and $x = \nu$, where $\mu \neq \nu$ and rows $u = \alpha$ and $u = \beta$, where $\alpha \neq \beta$: $d(\alpha,\mu) \neq d(\beta,\nu)$. Let x_i be an input signal of v with different delay to all other input signals of v. Then the probability that a glitch occurs at the output of v can be evaluated by replacing all products $P(x_i) \cdot P(\overline{x}_i)$ in (1), (2) or (3) with the probability that signal x_i changes if the primary input u to the path $u \longrightarrow \ldots \longrightarrow x_i$ changes $P_{change}(x_i|u)$. For the evaluation of the total glitch probability $P_T(v)$ at the output of vertex v the following algorithm **EG** (evaluate glitch probability) can be given:

Algorithm EG:

FOR($i = 1$ to $N-1$)
  FOR($j = 1$ to M)
    FOR($k = 1$ to M)
      IF$(d(i,j) \neq d(i+1,k))$
        $P_T(v) = P_T(v) + P_{glitch}(v)$

where N is the number of columns and M is the number of rows of $D(u,x)$. The total glitch probability is the expected amount of glitching per input transition. Note that $P_T(v)$ can be greater than one for certain architectures, e.g. the adder chain in Fig. 2.

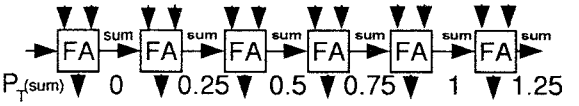

Figure 2: Glitch probability in an adder chain with equally distributed inputs.

For every vertex in the circuit a glitch-weight can be determined. In section IV the sum of the glitch-weights of all vertices in the circuit shall then be minimized by optimal placement of registers using retiming. The switching power caused by a input transition at gate v connected to a gate v_i, where $v \longrightarrow v_i$ is the only connection from v to v_i is given by:

$$P_{switch,v_i} = \frac{1}{2} V_{dd}^2 \cdot C_{Lv_i} \cdot \alpha(v). \quad (6)$$

V_{dd} is the supply voltage and C_{Lv_i} is the input capacitance of gate v_i. The activity factor $\alpha(v)$ is the sum of necessary output transitions $n(v)$ per input transition and the glitch probability $P_T(v)$:

$$\alpha(v) = n(v) + P_T(v). \tag{7}$$

With the power consumed for charging C_{Lv_i} caused by a spurious transition at gate v

$$P_{sp,v_i} = \frac{1}{2} V_{dd}^2 \cdot C_{Lv_i} \cdot P_T(v), \tag{8}$$

the glitch-weight g_v for gate v can be defined:

$$g_v = \sum_{i}^{fanout_v} P_{sp,v_i} * \rho_{v_i}. \tag{9}$$

Propagation factor $\rho_v \leq 1$ in (9) takes into account, that glitches which are shorter than the propagation delay of a gate are reduced in their voltage swing and their length [4]. $fanout_v$ is the transitive fanout of gate v.

III. Examples

In this section a comparison is made between the proposed methodology of section II and a experimental glitch-counting method. Two data path circuits are used for examples: a 5x5-bit array multiplier and a 4x4-bit Wallace tree multiplier. The glitch-counting tool extracts the timed boolean functions of every node in the circuit and then counts exactly all the spurious transitions for a given set of input vectors. For both circuits 10000 equally distributed random input vectors were used. In Fig. 3 and Fig. 4 the results of the comparison between the two methods are shown. The bars symbolize the average amount of glitches at the outputs per input transition.

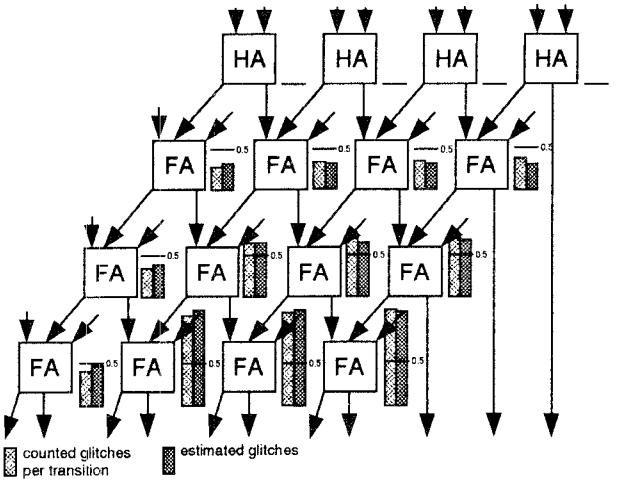

Figure 3: Spurious switching activity in a 5x5-bit array multiplier.

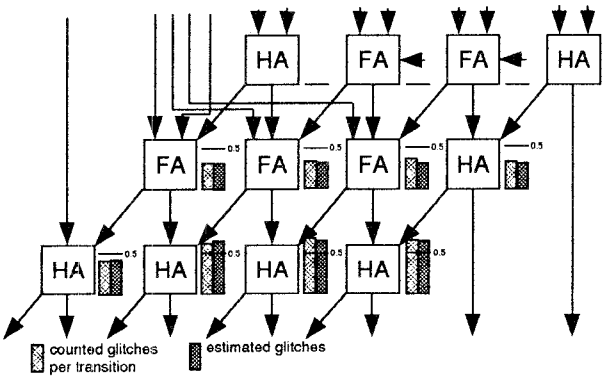

Figure 4: Spurious switching activity in a 4x4-bit Wallace tree multiplier.

IV. Retiming for Low Power

Spurious switching activity can be drastically reduced by introducing latches or registers in the circuit. Different path lengths to the inputs of a gate can be compensated, so that the input data to the gate is synchronised and glitches cannot appear at the output. Another effect is that latches and registers are barriers for glitches as data can only pass through when the clock signal appears. Retiming, introduced in [1] to improve the speed of a design by rearranging register positions is now applied to reduce the switching activity.

For simplicity the following retiming methodology is described for acyclic network graphs. The method is in general also applicable for cyclic graphs.

All primary inputs of the network are virtually connected with all outputs by a "dummy-vertex" with zero propagation delay. To every vertex v in the network a vertex-weight $r(v)$ is assigned. In a first initialization step all $r(v)$ are set to 0. Beginning at the primary inputs of the network, every vertex is observed stepwise in the order of the logical depth. In every step the glitch-weight is determined. If the glitch-weight for a vertex v is greater than a given maximum acceptable glitch-weight c_p, the vertex-weight is set to $r(v) = 1$. For every following vertex v_j of v $r(v_j)$ is also set to 1 until again vertex v or a register is reached. Now a new register distribution is calculated. For every edge e, $u \xrightarrow{e} v$ in the network the new register count $w_r(e)$ after retiming can be calculated from the actual register count $w(e)$:

$$w_r(e) = w(e) + r(v) - r(u)$$

If for any of the input edges of vertex v, where $g_v > c_p$, the edge weight $w_r(e) > 0$ and $w(e) = 0$, the input edge is "marked". This means that a register was placed in this edge because of high glitching activity at the output of the considered vertex. After the new register distribution is calcu-

lated, all $r(v)$ are set to 0 and the algorithm starts again at the primary inputs with the observation of the vertices. If in one of the retiming steps a register is removed from a marked edge so that $w_r(e) = 0$ leading to $g_v > c_p$ at the following vertex, the algorithm stops. In this case the maximum acceptable glitch-weight c_p is not feasable. By stopping the algorithm for this case, oszillations due to altering register positions are not possible and the algorithm always terminates. If c_p is feasable, the proposed algorithm causes that registers are placed at all inputs of vertices where $g_v > c_p$ and glitching is reduced.

A binary search algorithm has been implemented to find the minimum feasable glitch weight $c_{p_{min}}$, starting with a given initial glitch-weight c_{init}.

Besides low power considerations, timing constraints can also be involved so that a timing and power optimized design can be achieved. In this case the vertex-weight $r(v)$ will be incremented if $g_v > c_p$ or if $\Delta(v) > c_t$, where $\Delta(v)$ is the critical path delay to vertex v and c_t is the desired minimum clock period.

V. Experimental Results

The retiming algorithm combined with the gliching activity estimation tool was implemented as a program and tested on some data path examples: a 5x5-bit array multiplier, a 8x8-bit Wallace tree multiplier and a cascade of three 4-bit CORDIC stages applicable for DSP. The CORDIC was implemented with carry-ripple adders and with carry-save adders. All circuits were implemented in $1\mu m$ technology. To show the power saving effect of retiming more clearly, latches realised by simple transmission gates are used for this experiments instead of registers. Thereby additional power consumption caused by registers is avoided. Before starting the optimization algorithm, the allowed latch stages are placed at the primary inputs of the circuits. Hence, running the algorithm converts the combinatorial circuits into pipelined data paths. The pipelined circuits where then simulated with SPICE in terms of switching power. Table 1 shows the results for the four circuits with zero, one and two register stages. Note that the power consumption per transition for two stages of latches in the Wallace tree multiplier is higher than for only one stage. In some positions where two latches follow each other without any logic in between, they have to be replaced by registers which leads to additional power consumption. In the Wallace tree this additional power is more than the power saved due to glitch reduction.

Circuit	latch stages	total latches	$\frac{switching\ power}{Transition} / mW$
8x8 Wallace tree	0	0	3.169
	1	39	2.699
	2	57	2.789
5x5 array multipl.	0	0	1.203
	1	17	0.914
	2	30	0.904
CORDIC (carry-ripple)	0	0	1.338
	1	8	1.085
	2	16	0.967
CORDIC (carry-save)	0	0	2.549
	1	18	2.082
	2	34	2.045

Table 1: Switching power consumption in data path circuits with different pipeline stages.

VI. Conclusions

In this paper a new method for fast estimation of glitching activity in data paths is presented. The regularity of data path circuits allows to simplify the evaluation of glitch probabilities. A comparison of the new method with a exact glitch counting method shows that the approximation gives quite accurate results. A retiming algorithm that uses glitching activity for the optimization criteria was applied to reduce the switch power consumption of data path circuits.

The presented experimental results show that the proposed methodologies can help to achieve significant power savings. Using the regularity of a design to simplify the estimation of spurious power consumption is a promising approach for future research.

References

[1] C. E. Leiserson, J. B. Saxe: Retiming Synchronous Circuitry. *Algorithmica pp.5-35, 1991.*

[2] J. Monteiro, S. Devadas, A. Ghosh: Retiming Sequential Circuits for Low Power. *Proc. ICCAD 1993.*

[3] F. Najm: Transition Density, A Stochastic Measure of Activity in Digital Circuits. *Proc. 28th Design Automation Conference, pp. 644-449, 1991.*

[4] M. Eisele, J. Berthold: Dynamic Gate Delay Modelling for Accurate Estimation of Glitch Power at Logic Level. *5th Int. Workshop on PATMOS, Oct. 1995, Oldenburg, pp. 190-201.*

[5] C. K. Lennard, A. R. Newton: On Estimation Accuracy for Guiding Low-Power Resynthesis. *IEEE Trans. on Computer Aided Design, Vol. 15, No. 6, pp. 644-664, June 1996*

[6] A. Ghosh, S. Devadas, K. Keutzer, J. White: Estimation of Average Switching Activity in Combinational and Sequential Circuits. *29th ACM/IEEE Design Automation Converence, pp. 253-259, June 1992.*