Slack Equalization Algorithm: Precise Slack Distribution for Low-Level Synthesis and Optimization

Chunhong Chen and *Majid Sarrafzadeh*

Department of Electrical and Computer Engineering
Northwestern University, Evanston, IL 60208
Email: chen@ece.nwu.edu , majid@ece.nwu.edu

Asbstract

In this paper we deal with the slack distribution problem for any given circuit and develop a Slack Equalization Algorithm to maximize the specific objective functions under the timing constraints. The application of our algorithm to voltage scaling for low power is also discussed. Experimental results demonstrate the effectiveness of the algorithm.

1 Introduction

With the growing popularity of personal computing and communication devices, the high-performance as well as low power consumption is much desirable in the synthesis of VLSI circuits and systems. Over the years, much research progress has been made towards the timing, area and power optimization [1-3]. However, when the designers look into the possibility of further improvement over circuit area/power, the timing constraints turn out to be a strong limiting factor. This trend can be exacerbated at lower levels (such as gate-level and physical-level) where relatively accurate timing models are available. Therefore, the tradeoff between delay and area/power has to be made, in general, to achieve the result with reasonably good quality. Since the timing performance of a circuit depends on critical paths, the logic gates on non-critical paths have typically large timing slacks (i.e. the difference between required time and arrival time). The problem is thus translated into how to exploit these excessive slacks effectively during the process of optimization. Most of prior efforts do not deal directly with the slacks.

In this paper we directly target analyzing the slack distribution and develop a *Slack-Equalization-Algorithm* to maximize the objective function under the timing constraints. This algorithm can find many applications in low-level synthesis and optimization ranging from gate-level power/area optimization to layout-level placement and routing design.

2 Our Algorithm

Consider a *directed-acyclic graph* **G**(**V**, **E**), where each node $v \in \mathbf{V}$ corresponds to a logic gate of the given circuit. Each edge $(u, v) \in \mathbf{E}$ means node u is a *fanin* of node v. Given a node delay model and timing constraints of the circuit, we can obtain the *slack* time, $s(v)$, for each node v by calculating its *arrival* time, $a(v)$, (*required* time, $r(v)$) through *forward* (*backward*) delay propagation. If the circuit initially meets the timing requirement, we have $s(v) \geq 0$, $\forall v \in \mathbf{V}$. Intuitively, the larger slack indicates more space left for area/power reduction while maintaining the circuit performance. Without loss of generality, we use $g_v(\varepsilon)$ to represent the area/power gain of node v with its delay penalty of ε . In the real world, $g_v(\varepsilon)$ is a monotonic function, that is, $g_v(\varepsilon_1) \geq g_v(\varepsilon_2)$ if $\varepsilon_1 \geq \varepsilon_2$.

Definition 2.1 (i) An edge $(u, v) \in \boldsymbol{E}$ in $\boldsymbol{G}$ is called *sensitive* if either $a(v) - a(u) = d(v)$ or $r(v) - r(u) = d(v)$. (ii) A directed path is called *sensitive* if: (a) the path consists of only sensitive edges, and (b) the slack of all nodes on the path is monotonously distributed[1] in the direction of the path. (iii) Two nodes u, $v \in \boldsymbol{V}$ are called *slack-sensitive* if there exists a sensitive path from u to v or from v to u in $\boldsymbol{G}$. Otherwise, they are called *slack-insensitive*.

[1] Assuming a directed path consists of $\{v_1, v_2, \cdots, v_m\}$. The monotonic slack distribution on the path implies that, if $s(v_m) \geq s(v_1)$, then $s(v_{i+1}) \geq s(v_i)$; if $s(v_1) \geq s(v_m)$, then $s(v_i) \geq s(v_{i+1})$, where $i = 1, \cdots, m-1$.

Definition 2.2 The *sensitive transitive closure graph* $\boldsymbol{G}_s = (\boldsymbol{V}, \boldsymbol{E}_s)$ of $\boldsymbol{G}$ is a directed graph such that there is an edge (v, w) in $\boldsymbol{G}_s$ if and only if there is a directed sensitive path from v to w in $\boldsymbol{G}$.

Suppose **TF**(v) denotes the set of all *transitive fanins* and *fanouts* of node v. If the delay of node v increases by ε, its slack is reduced by exactly ε, and the slack of any node $u \in$ **TF**(v) may or may not change, depending on the current value of $s(u)$. Here are three cases to consider (their proof is omitted):

(1) if $s(u) \geq s(v)$, the slack of u will decrease by ε unless u and v are slack-insensitive.

(2) if $(s(v) - \varepsilon) < s(u) < s(v)$, the slack of u will decrease by (s(u) – s(v) + ε) unless u and v are slack-insensitive.

(3) if $s(u) \leq (s(v) - \varepsilon)$, the slack of u remains unchanged.

To obtain the maximum of total gain, it is preferable to keep the minimum number of nodes whose slacks are reduced by the increased delay of node v. In this sense, case (3) is our best choice. To this end, we first select the set of nodes, $\mathbf{Q_m}$, with maximum slack, s_{max}, in **G** and construct a *transitive graph*, $\mathbf{G_m}$, on $\mathbf{Q_m}$. $\mathbf{G_m}$ is called the *slack-equivalent graph*. Within $\mathbf{G_m}$, selecting nodes in *Maximal Independent Set* for the delay increase of ε produces the maximum gain. The outline of algorithm is described below.

Slack-Equalization-Algorithm (SEA)

Step 1 Find the set of nodes, $\mathbf{Q_m}$, in G with maximum slack, s_{max}.

Step 2 Construct a *slack-equivalent graph*, $\mathbf{G_m}$, on $\mathbf{Q_m}$.

Step 3 Find the *Maximal Independent Set* (**MIS**) of $\mathbf{G_m}$.

Step 4 Increase the delay of each node $v \in$ **MIS** by ε.

Step 5 Repeat *Step* 1 ~ 4 until $s_{max} < \varepsilon$.

The main feature of **SEA** is to get maximum gain using the minimum reduction of node slacks. At each step, the slack of any node whose slack is less than s_{max} will not be affected if only ε is no more than ($s_{max} - s_{max\text{-}1}$), where $s_{max\text{-}1}$ is the second largest slack of all nodes in $\boldsymbol{G}$. Therefore, the reasonable value of ε is $s_{max} - s_{max\text{-}1}$. Assuming the circuit contains m groups of nodes with different slacks, the whole process for the node selection can be completed after (m – 1) passes. Considering different gain functions, we generally need to assign *weight*(v) = $g_v(\varepsilon)$ for each node v and, hence, use *Maximal Weighted Independent Set* (MWIS) to take place of MIS in the above algorithm. The specific $g_v(\varepsilon)$ should be determined according to the specific optimization objective which depends on the specific application.

3 Applications

The **SEA** can find many applications in low-level synthesis and optimization such as gate resizing for area/power reduction, generation of performance constraints for module placement and routing, and two-voltage techniques for low power design. Here we show its application to low power design using dual supply voltages [4,5]. We can prove that the gain function for this purpose is given by $g_v(\varepsilon) = \dfrac{C(v) \cdot E(v)}{d(v)} \cdot \varepsilon$ (the detailed derivation can be available by contacting authors). *Usami* first proposed a dual-voltage approach based on the so-called *Cluster Voltage Scaling* (**CVS**) [4]. The basic idea is to use *Depth-First Search* from primary outputs for finding gates which can operates at low supply voltage without violating the given timing constraints. We implemented the **SEA** and **CVS** under *SIS* environment and tested them on a set of *MCNC*'91 benchmark circuits. Table 1 shows their comparison. Under the same timing constraints, **SEA** obtained 17% (on average) more gates operating at the low

voltage (V_{ddl}) than **CVS**, as shown in column 5 of Table 1. In terms of power reduction, the **SEA** is about 5% (on average) better than the **CVS**, taking into account the power overhead of *level converters* required. For individual circuits, the power improvement is as high as 29% (for circuit *dalu*). The last column of this table lists their CPU time on a SUN SPARC station 5 with 32MB RAM.

Table 1 Comparison of SEA and CVS Algorithm on a Set of Benchmark Circuits

Circuit	# Total Gates	Delay (*ns*)	Initial Total Power	#V_{ddl} Gates (% of total gates)		Final Power (SEA) (% of init. total power)		Total Power Red. %		CPU Time (secs)	
				SEA	CVS	Gates	LCs	SEA	CVS	SEA	CVS
9symml	200	23.6	1339.4	63.5	0	85.4	4.6	10.0	0	1.22	0.67
C1908	464	52.0	2189.1	62.7	28.2	75.9	10.6	13.5	7.5	3.45	3.83
C5315	1509	51.2	8179.8	83.8	56.1	67.3	5.3	27.4	15.4	59.93	28.35
C880	379	42.8	1972.4	86.3	62.5	68.9	2.4	28.7	19.4	3.35	1.57
alu2	364	49.4	1444.7	68.7	23.1	78.8	5.4	15.8	4.0	2.92	2.93
apex6	788	35.0	3242.6	94. 9	93.0	68.3	1.5	30.2	29.0	15.63	4.38
apex7	249	23.0	1111.4	67.1	71.7	76.7	6.3	17.0	18.9	1.35	0.80
b9	159	14.6	686.6	68.6	57.9	76.7	6.2	17.1	15.0	0.55	0.37
c8	130	13.2	660.2	34.6	38.5	88.7	3.5	7.8	7.3	0.33	0.30
dalu	1477	109.8	6846.7	91.7	29.5	64.0	2.5	33.5	4.7	114.28	97.62
des	2685	66.4	13755.0	84.4	77.9	61.2	3.6	35.2	31.6	389.62	54.20
frg1	175	23.4	918.0	62.9	6.9	75.8	6.8	17.4	1.5	0.77	0.77
frg2	872	38.0	3234.2	76.7	80.2	71.6	6.2	22.2	21.2	15.67	6.88
i1	65	13.0	207.6	67.7	61.5	78.3	4.4	17.3	17.6	0.15	0.10
i3	310	9.2	1557.2	1.9	1.9	99.9	0.0	0.1	0.1	1.37	1.55
i5	243	26.6	857.8	67.9	78.2	81.7	2.1	16.2	20.1	1.40	0.47
i6	373	21.2	1865.9	32.7	22.5	86.1	7.8	6.1	4.1	3.02	1.58
i7	487	23.8	2570.6	65.7	88.3	79.7	5.8	14.5	20.6	13.23	3.02
i8	1085	45.6	4506.3	78.5	53.1	69.5	7.8	22.7	17.8	27.90	15.68
rot	860	42.6	3799.6	89.0	72.1	68.6	2.8	28.6	21.5	19.43	4.07
term1	244	20.0	1247.3	66.4	27.5	75.1	8.5	16.4	6.7	1.18	1.00
Average				**67.4**	**49.1**	**76.1**	**5.0**	**18.9**	**13.5**		

We are currently working on applying **SEA** at the physical level (placement and routing). We expect that the effectiveness of **SEA** can be further confirmed by its applications in many issues of logic and layout synthesis.

References

[1] H. Toutai, W. Moon, R. Brayton, and A. Wang, "Performance-Oriented Technology Mapping", In Proc. M.I.T. Conference on Advanced Research in VLSI, 1990.

[2] K. Chaudhary and M. Pedram, "Computing the Area versus Delay Trade-off Curves in Technology Mapping", IEEE Transactions on CAD, vol.14, no.12, Dec. 1995.

[3] D-S Chen and M. Sarrafzadeh, "An Exact Algorithm for Low Power Library-Specific Gate Re-sizing", IEEE/ACM Design Automation Conference, 1996.

[4] K. Usami and M. Horowitz, "Cluster Voltage Scaling Technique for Low-Power Design", International Symposium on Low Power Design, pp.3-8,1995.

[5] K. Usami, et al, "Automated Low Power Technique Exploiting Multiple Supply Voltage Applied to a Media Processor", Custom Integrated Circuit Conference, pp.131-134, 1997.