High speed circuit design---time borrowing and time stealing

Xiaoyu Teng

Abstract

This paper introduces one of the high speed CMOS clocking design style --- time borrowing and time stealing. It starts with an introduction of traditional domino pipeline structure and its limits. In the following parts, time borrowing and time stealing and their usages are introduced respectively.

Introduction

Since in an edge-triggered system, the operation time of each pipeline partition will never equal to others and the longest case logic delay between two registers will determines the maximum clock frequency of the whole system, some fast logic partitions will have to be idle till the next system clock edge. For example, as shown in Figure 1, conventional flip-flops are used in an N-cycle pipeline block. This pipeline block consists of N combinational logic stages and separated by N+1 flip-flops, where Tdata(i) denotes the delay of ith combinational logic stages, and CLK represents the system clock. So, the delay time of all stages is the sum of Tdata(i). For this N-cycle pipeline block, the slowest stage delay limits the cycle time and could result in more severe degradation as N increases.

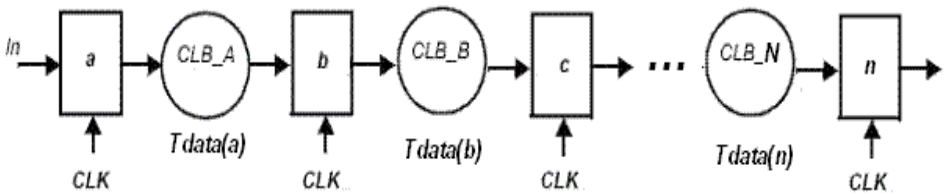

Figure 1 N stages domino pipeline block

In any circuit, the concept of how to fit more logic within each logic partition than is readily available always becomes an issue [1]. So every pipeline partition will want more time than allocated. Time borrowing and time stealing which allows one stage to pass slack time from fast logics to neighboring slower ones will remedy those problems.

Time borrowing

Time borrowing by definition is permitting logic to automatically use slack time from a previous cycle [1]. It always indicates the situation that logic partition in a pipeline structure use left over time from previous stage. And this passing of slack time from one cycle or phase to the next is automatic and without any additional circuitry or clock adjustments. The key advantage of slack

borrowing is that it allows logic between cycle boundaries to use more than one clock cycle while satisfying the cycle time constraint [2]. That is, if a sequential system works at a particular frequency and the sum of logic delay for a complete cycle is longer than one clock cycle, then slack time has been borrowed from next stages. It means that the clock rate could be higher than the slowest delay path.

Time borrowing happens due to the level sensitive of latches and precludes the use of edge-triggered logic and edge-triggered latching. Since to use an edge-triggered structure must require a clock arrival time adjustment at the circuit which violates the definition of time borrowing and using an earlier clock from left over time in previous partition is an extended case of time stealing. So time borrowing is ideally suitable for static logic in a two-phase clocking system with non-edge-triggered circuits and latches.

Assuming in a symmetric 50% duty cycle two-phase system, as shown in Figure 2, point *a* (input to *CLB_A*) is valid before the edge 2. This implies that the previous block did not use up the entire low phase of CLK2, which results in slack time (denoted by the shaded area). So, *CLB_A* can start computing as soon as point *a* becomes valid and uses the slack time to finish well before its allocated time (edge 3). Since L2 is a transparent latch, *c* becomes valid on the high phase of CLK2 and *CLB_B* starts to compute by using the slack provided by *CLB_A*. *CLB_B* completes before its allocated time (edge 4) and passes a small slack to the next cycle. As this picture indicates, the total cycle delay, that is the sum of the delay for *CLB_A* and *CLB_B*, is larger than the clock period. Since the pipeline behaves correctly, slack passing has taken place and a higher throughput has been achieved.

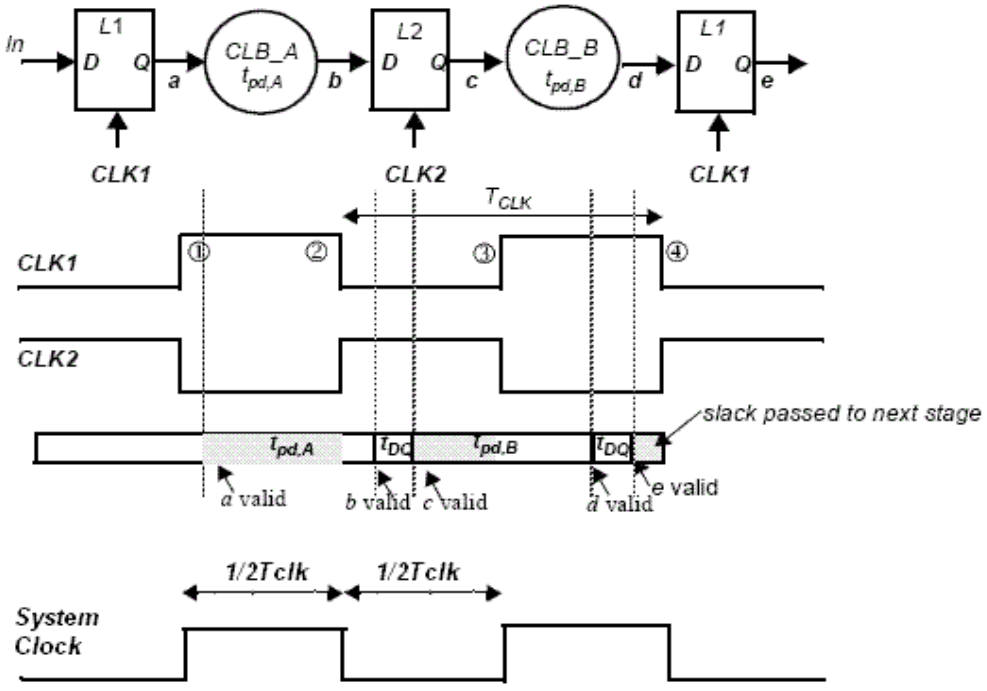

Figure 2 An example of time borrowing

And another important character related to slack borrowing is the maximum possible slack that can be passed across cycle boundaries. In Figure 2, it is easy to see that the earliest time that *CLB_A* can start computing is edge 1. This happens if the previous logic block did not use any of its allocated time (CLK1 high phase) or if it finished by using slack from previous stages. Therefore, the maximum time that can be borrowed from the previous stage is 1/2 cycle or *Tclk*/2. Similarly, *CLB_B* must finish its operation by edge 4. This implies that the maximum logic cycle delay is equal to 1.5 * *Tclk*. However, the overall logic delay still does not exceed the time available of *N* * *TCLK*.

On the other hand, although time borrowing can bring design flexibility and relax the time constrain, in practice, time verification of each clock cycle boundaries is not trivial. Since in a two-phase latch based design, the introduction of an additional latch per stage results in twice the number of minimum timing need to be checked. And a two-phase design needs two clock phases which could complicate the design so much. So in practice, only one clock phase is to be globally distributed and its complementary clock phase is generated locally. In commercial tools, only a few has such capabilities such as Synopsys's Design Compiler. And mostly time borrowing is only allowed in exceptional cases, which are carefully verified individually.

Examples of time borrowing

Time borrowing has been traditionally used to reduce the effect of clock skew and jitter on maximum clock frequency [3]. And it also has been widely used in critical delay path of high speed circuits especially in high speed adder designs [4].

Since time borrowing can automatically average out delay variations along a path caused by process variation and inaccuracies, time borrowing is used to alleviate the mean maximum clock frequency degradation caused by within-die parameter variations [5]. And the result shows a 4-6% mean maximum clock frequency improvement and a corresponding 10% average energy savings with the factors of N equals to 3, 6, 9 and 12, as shown in Figure 3.

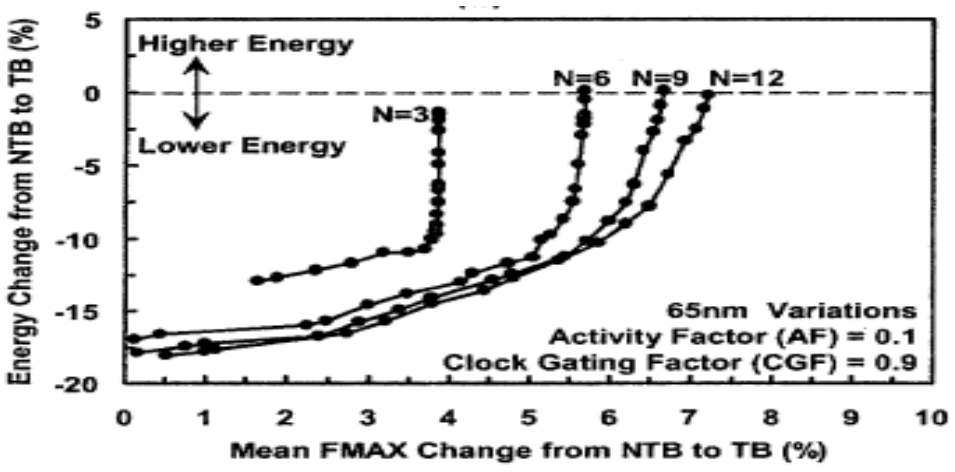

Figure 3 Tradeoffs between clock frequency and energy savings improvement

Time stealing

Time stealing gains evaluation time by taking it from the next cycle and is ideally suited for dynamic logic in two-phase system [1].

Generally speaking, time stealing refers to some logical partition needs additional time for evaluation but cannot use left over time from previous cycles or phase as the case of time borrowing. Therefore the most significant difference between time stealing and time borrowing is that time stealing will not automatically use the left over time. It has to be forced to steal evaluation time from subsequence cycle or phase and leave less time to that cycle or phase which is achieved by adjusting clock arrival time. Since this additional time is obtained by adjusting clock arrival time, it is usually used in edge triggered logic. So if a dynamic logic needs more time to evaluate, it has to increase its phase time by widening the active clock time. And this can only be done by shifting the rising edge earlier or falling edge later. This means instead of using symmetric 50%duty cycle 2 phase system as shown in time borrowing, it has to use asymmetric duty cycle to gain additional time.

Example of time stealing

In paper *Leakage Power-Aware Clock Skew Scheduling: Converting Stolen Time into Leakage Power Reduction* [6]**,** time stealing has been used to reduce leakage power and thus total power dissipation for the first time. It shows a technique (Leakage Aware Clock Skew Scheduling, L-CSS) that combining threshold voltage assignment and gate sizing (TVA/GS) and time stealing to reduce power. They have an important assumption or observation: “the amount of power reduction achievable through TVA/GS for a fast block will be different than what is possible with the same amount of slack in a slow block”. Slow blocks need larger amounts of slack to enjoy significant power reductions while keeping the clock period intact. So they use the concept of “time stealing” to steal slack from fast blocks to be utilized for leakage power reduction in slow blocks. And they proposed a technique called leakage aware clock skew scheduling (L-CSS) and its effectiveness is largely depended by how balanced the sequential circuit is. There needs to be a higher number of fast combinational blocks than the timing critical slow blocks. L-CSS technique is mainly comprised of three major steps. First, gate sizing is performed to obtain an initial solution optimized for timing and area. Then, the clock skew scheduling and slack distribution are unified into a linear program power formulation. The timing slacks allocated to each gate and the clock arrival time for each flip-flop is computed. Based on that, a standard cell from the technology library is chosen to replace the one in the initial solution, such that the power reduction is maximized while the timing constraints are still met. In the third step, a final round of gate resizing is performed to fix possible clock period violations. For proving their result, they use TSMC 65nm standard cell technology libraries to characterize the timing and power of ISCA89 benchmark circuits. And as the result shown in table 1 which LVT indicates the low Vth library, the L-CSS techniques has a superior performance than TVA/GS alone: 18.79% of average leakage power reduction and 6.15% of total power.

Circuit	Leakage Power (μW)			Relative Imp. w.r.t. TVA/GS	Total Power (μW)			Relative Imp. w.r.t. TVA/GS
	LVT	TVA/GS	L-CSS		LVT	TVA/GS	L-CSS	
s27	30.08	12.69	12.69	0.00%	269.86	162.08	159.79	1.41%
s298	119.01	72.53	32.04	55.83%	571.66	565.79	509.68	9.92%
s344	178.71	108.25	82.51	23.78%	965.05	923.65	861.16	6.77%
s349	179.47	108.32	82.53	23.81%	969.61	929.48	864.87	6.95%
s382	169.39	113.46	76.23	32.81%	847.06	818.58	671.35	17.99%
s386	264.58	233.33	230.9	1.04%	1316.9	1313.7	1310	0.28%
s400	154.92	84.59	51.47	39.15%	752.13	632.26	535.4	15.32%
s420	212.24	139.66	139.63	0.02%	779.36	730.79	698.76	4.38%
s444	160.29	112.05	52.82	52.86%	790.06	776.77	591.39	23.87%
s510	221.66	146.9	122.37	16.70%	1478.5	1491.5	1450.9	2.72%
s526	171.05	56.07	29.47	47.44%	741.12	624.05	576.98	7.54%
s641	333.14	223	207.16	7.10%	814.89	715.26	699.14	2.25%
s713	390.19	248.46	237.31	4.49%	890.17	752.06	737.88	1.89%
s820	265.23	98.52	101.45	-2.97%	1704.8	1534.9	1537.2	-0.15%
s832	265.57	107.66	103.31	4.04%	1689.4	1531.4	1518.5	0.84%
s838	333.17	166.43	161.69	2.85%	821.52	665.47	641.08	3.67%
s953	360.73	238.46	136.69	42.68%	1805.2	1780.6	1646.7	7.52%
s1196	402.7	191.44	191.22	0.11%	1745.8	1536.3	1535.3	0.07%
s1238	408.93	183.18	182.94	0.13%	1733.4	1508.1	1507.2	0.06%
s1423	584.69	242.35	200.19	17.40%	1061.1	710.48	654.46	7.88%
s1488	725.02	542.2	526.47	2.90%	2606.5	2483.8	2463.7	0.81%
s1494	685	468.36	458.65	2.07%	2436.6	2259.1	2220.3	1.72%
s5378	1608.9	913.04	904.19	0.97%	6288.9	5632.8	5620.2	0.22%
s9234	4310.1	2226.3	1314.9	40.94%	7047.2	4971.3	4048.4	18.56%
s13207	4650.2	1554.7	1082.7	30.36%	8341.2	5263.5	4782.6	9.14%
s15850	5469.2	1581.6	991.87	37.29%	9781.6	5892.1	5233.7	11.17%
s35932	21123	18352	16560	9.76%	79156	76737	74127	3.40%
s38417	12194	4982.2	3395.8	31.84%	25066	17978	16099	10.45%
s38584	11970	1327.9	1069	19.50%	29844	18914	18596	1.68%
AVG				18.79%				6.15%

Table 1 Total power consumption for TVA/GS and L-CSS techniques and relative improvement of L-CSS with respect to TVA/GS

Reference

[1] Kerry Bernstein, et al. High Speed CMOS Design Styles. Kluwer Academic Publishers.

[2] Jan M.Rabaey and Anantha Chandrakasan. Digital Integrated Circuits. Printice Hall Electronics and Vlsi Series.

[3] D. Harris and M. A. Horowitz. Skew-Tolerant Domino Circuits. IEEE Journal of Solid-state Circuits, Vol. 32, No. 11, pp. 1702-1711, November, 1997.

[4] Gunok Jung and Gerald E. Sobelman. HIGH-SPEED ADDER DESIGN USING TIME BORROWING AND EARLY CARRY PROPAGATION.

[5] Keith Bowman, et al. Time-Borrowing Multi-Cycle On-Chip Interconnects for Delay Variation Tolerance.

[6] Min Li and Seda Ogrenci Memik. Leakage Power-Aware Clock Skew Scheduling: Converting Stolen Time into Leakage Power Reduction.