Saturating Counters: Application and Design Alternatives

Israel Koren
Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA 01003
Email: koren@euler.ecs.umass.edu,

Yaron Koren
Bear Stearns, New York, NY 10179

Bejoy G. Oomman
Genesys Testware, Fremont, CA 94539

Abstract

We define a new class of parallel counters, Saturating Counters, which provide the exact count of the inputs that are 1 only if this count is below a given threshold. Such counters are useful in, for example, a self-test and repair unit for embedded memories in a system-on-a-chip. We describe this application and present several alternatives for the design of the saturating counter. We then compare the delay and area of the proposed design alternatives.

1. Introduction

Various designs of parallel counters to be used in multiplier units and other applications have been proposed and implemented (e.g., [1, 2]). Such designs use different basic building blocks like (3,2) counters, (7,3) counters and the like [3]. An (n,k) parallel counter has n input bits and produces a k-bit binary count of its inputs that are 1. Clearly, k must satisfy $2^k - 1 \geq n$ or $k \geq \lceil \log_2(n+1) \rceil$. We define here a new type of parallel counters which we call *saturating counters*. A saturating counter needs to provide the exact count of its inputs that are 1 only if this count is below a certain threshold, denoted by T. The exact output is less important when the number of inputs that are 1 exceeds the threshold T, as long as the output indicates that the threshold has been exceeded. Such a saturating counter is needed in the design of a self-test and repair circuit for large memories embedded in a system-on-a-chip. Note that the saturating counters considered here are different from those used in certain image processing applications and in microprocessors' branch prediction units. The latter normally saturate at their maximum count of $2^k - 1$ (and, sometimes also at their minimum count of 0) and all other results must be exact.

The necessary number of output bits of a saturating counter, denoted by k, does not have to satisfy the condition $k \geq \lceil \log_2(n+1) \rceil$. Instead, the inequality which must be satisfied is $k \geq \lceil \log_2(T+1) \rceil$. In principle, T can be any number smaller than n, however, a simpler and faster implementation can be achieved when T is a power of 2. Moreover, for the application considered in this paper, if the threshold is not a power of 2 we can still employ a saturating counter with $k = i + 1$ output bits where $i = \lceil \log_2 T \rceil$. We will therefore, focus in this paper on the special case of $[n, k]$ saturating counters with n inputs, a threshold of $T = 2^{k-1}$, and k output bits.

The paper is organized as follows. In Section 2 we describe the application that requires the design of a fast saturating counter. In Section 3 we present some design alternatives for saturating counters and in Section 4 we compare the delay and area of the various alternatives. Section 5 concludes the paper.

2. Self-Test and Repair for Embedded Memories in a System on a Chip

The high density and size of memory units, implemented either as separate ICs or as embedded memories, have resulted in an increasing number of manufacturing defects leading to low yields of high volume ICs. System-On-a-Chip (SOC) designs that contain megabits of embedded memory are now available from several companies. The manufacturing yield of these SOC products is strongly dependent on the yield of their embedded memory.

Spare memory rows and columns have traditionally been added to memory designs to replace defective rows, columns or individual cells. To perform such replacements, the defective rows, columns or cells must be identified first. In the past, dedicated external memory testers with fault diagnosis capabilities have been used. Following the identification of the defective cells, the chip is taken to a laser repair station and fuses are blown to replace faulty memory cells with spare memory cells [4].

To eliminate the costly memory tester from the chip manufacturing process, designers have started to incorporate Built-In Self-Test (BIST) circuitry into large memory units. Such circuitry is capable of executing memory tests to diagnose any error, which may be the result of either a manufacturing fault or a fault (intermittent or permanent) that occurs during the normal operation of the IC.

Designers of systems-on-a-chip have gone one step further, and several current designs include a Built-In Self-Test Diagnosis and Repair (BISTDR) circuit for the embedded memories in the SOC. The use of BISTDR not only enables permanent memory repair following manufacturing (hard repair), but also every time the system is powered up (soft repair). Hard repair can be done by laser blown fuses or by writing non-volatile re-configuration flip-flops, while soft repair uses only the latter [5, 6, 7].

The process starts with a self-test operation performed internally in the memory unit. Once the faulty data bits and faulty addresses have been identified, the faulty data bits are replaced with spare data bits, and faulty words are replaced with spare words. Figure 1 shows a block diagram

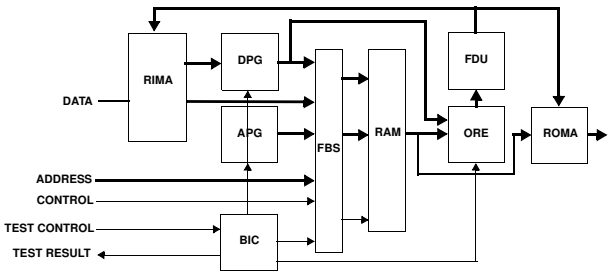

Figure 1. A block diagram of the BISTDR unit.

of the BISTDR unit, which consists of the following functional blocks: APG (Address Pattern Generator), DPG (Data Pattern Generator), ORE (Output Response Evaluator), BIC (BIST Interface Controller), FDU (Fault Diagnosis Unit), FBS (Function to BIST selector), RIMA (Repair Input Multiplexor Array) and ROMA (Repair Output Multiplexor Array).

The built-in self-repair is usually executed automatically during the power-on reset sequence of the SOC and must, therefore, be performed at system speed using the system clock. The test and repair process is done on the fly in a single cycle to avoid the need to store fault information. The Fault Diagnosis Unit (FDU) is therefore on the critical path in the BISTDR circuit since it has to identify the faulty bit(s), and make a repair decision (address

or data repair) within one memory read access cycle. This repair decision is based on the number of failing data bits at the current address, and the currently available repair resources (unused spare data bits and address locations). The failing bits are determined using an array of XOR gates which compares the memory output with the expected output. This produces a bit vector whose width equals the width of the memory. A bit in this vector will have a 1 if there is a mismatch at the corresponding bit position, and a 0 otherwise.

The critical path within the FDU includes a circuit that counts the number of 1's in this bit vector. This number has to be compared to the number of spare bits which is typically no larger than 8. If the number of bit failures exceeds the number of spares, the memory is not repairable. Therefore, it is sufficient to know the failing bit count accurately only when it is less than the number of spares available. The fast saturating counter we design must therefore have a threshold T, where T is equal to or slightly larger than the number of available spares, allowing us to select a threshold which is a power of 2. The number of inputs of the required saturating counter, n, is the width of the memory. Unlike stand-alone memory chips, embedded memories in SOC designs have no restriction on the data width due to pad limitations. Thus, embedded memories of width of up to 1024 are commonly used in SOCs. We should therefore design saturating counters for as many as 1024 input bits.

3. Saturating Counters - Design Alternatives

An $[n, k]$ saturating counter has n input bits denoted by $a_1, a_2, \cdots, a_n$, k output bits denoted by $s_{k-1}, s_{k-2}, \cdots, s_0$ and a corresponding threshold of $T = 2^{k-1}$. The output satisfies

$$(s_{k-1}, s_{k-2}, \cdots, s_0)_2 = \sum_{i=1}^{n} a_i$$
$$\text{if} \quad \sum_{i=1}^{n} a_i \leq 2^{k-1}$$

while

$$(s_{k-1}, s_{k-2}, \cdots s_0)_2 \in [2^{k-1}+1, 2^k-1]$$
$$\text{if} \quad \sum_{i=1}^{n} a_i > 2^{k-1}$$

For example, a $[1024, 4]$ saturating counter has 1024 inputs, a threshold of 8, and produces four output bits satisfying

$$(s_3, s_2, s_1, s_0)_2 = \sum_{i=1}^{1024} a_i$$

if there are at most eight input bits which equal 1, and $(s_3, s_2, s_1, s_0)_2 \in [9, 15]$ if there are nine or more input bits which equal 1.

A complete Wallace tree for 1024 inputs produces nine output bits and requires 16 levels of (3,2) counters. A straightforward way to implement a $[1024, 4]$ saturating counter is to use (3,2) counters in the columns with weights $2^2, 2^1$ and 2^0 but use only OR gates in the column with weight 2^3. This implementation, shown in Table 1, requires 11 levels of (3,2) counters plus one level of an OR gate, assuming that two levels of OR operations in column 2^3 can be completed in parallel to the operation of a single level of (3,2) counters in the $2^2, 2^1$ and 2^0 columns. Table 1 shows, for each level of the tree, the number of (3,2) or (2,2) counters required in every column, and the resulting number of intermediate results in every column. For example, in the second level of the tree, 114 (3,2) counters are used in the 2^0 column, producing 114 intermediate bits of weight 2^0 and 114 bits of weight 2^1, which are added to the 115 bits generated directly in the 2^1 column. The notation 9+2(OR_4) in the 2^3 column means that two levels of OR gates are used, 9 in the first level and 2 in the second.

Note that the implementation depicted in Table 1 will produce a result of 8 if the number X of input bits which equal 1 satisfies $X mod\ 8 = 0$, e.g., 16, 32 and so on. If such a situation is not allowed, a threshold of $T = 16$ can be selected. However, for the application at hand the probability of such an event occurring was deemed to be negligible.

The average expected number of defective memory cells in a single row is less than 4, with a standard deviation of less than 2, making the probability of 16 defective cells in one row practically zero.

In [1] Jones and Swartzlander have compared the design of parallel counters using only (3,2) or (2,2) counters to designs using more complex counters like (7,3), (15,4) and (31,5). They have analyzed the delay and area of different implementations and concluded that designs based on (3,2) and (2,2) counters only are generally superior. We therefore decided not to experiment with counters like (7,3), (15,4) and (31,5). However, in recent years (4;2) compressors [3] have become common in parallel multiplier designs, and very efficient implementations for them have been proposed (e.g., [8]). Consequently, we studied the possibility of using (4;2) compressors instead of (3,2) counters in one or more levels of the saturating counter. Table 2 shows that if (4;2) compressors are used in levels 1 through 5, the total number of levels is reduced from 12 to 9. (4;2) compressors, though, have a higher delay than (3,2) counters. However, if the delay of a (4;2) compressor is only about 50% larger than the delay of a (3,2) counter, the overall delay of the [1024,4] saturating counter still decreases when (4;2) compressors are used. Detailed delay comparisons are reported in the next section.

Tables 1 and 2 were obtained using an online saturating counter simulator which is available at [9].

3.1. ($\{m, i, j\}$, 3) units

Re-examining Tables 1 and 2, one can notice that the last few stages achieve only a small reduction in the number of bits but incur a high delay. One could replace the last four stages in Table 1, which reduce the number of bits from (5,5,2,1) to (1,1,1,1), by a look-up table with 2^{5+5+2} inputs and 4 outputs. However, a simpler and probably faster (for most technologies) solution exists which takes advantage of the saturating nature of the counter. This solution uses a special 3-column ($\{$5,5,2$\}$,3) unit, as shown in Table 3. If we wish to apply the same approach to the [1024,4] saturating counter which uses (4;2) compressors (see Table 2), a ($\{$13,8,3$\}$,3) unit could be used. The design principles of such units are presented next.

2^3	2^2	2^1	2^0	Level
			341(3,2)	
		341	342	1
		113(3,2)	114(3,2)	
	113 113	114+115 229	114 114	2
	37(3,2)	76(3,2)	38(3,2)	
37 37	76+39 115	38+76 115	38 38	3
9+2(OR$_4$)	38(3,2)	38(3,2)	12(3,2)	
4+38 42	38+39 77	12+39 51	14 14	4
10+3(OR$_4$)	25(3,2)	17(3,2)	4(3,2)	
3+25 28	17+27 44	4+17 21	6 6	5
7+1(OR$_4$)	14(3,2)	7(3,2)	2(3,2)	
4+14 18	7+16 23	2+7 9	2 2	6
4+1(OR$_4$)	7(3,2)	3(3,2)	1(2,2)	
3+7 10	3+9 12	1+3 4	1 1	7
2+1(OR$_4$)	4(3,2)	1(3,2)		
1+4 5	1+4 5	2 2	1 1	8
1+0(OR$_4$)	1(3,2)	1(2,2)		
2+1 3	3+1 4	1 1	1 1	9
0(OR$_4$)	2(2,2)			
3+2 5	2 2	1 1	1 1	10
1(OR$_4$)	1(2,2)			
2+1 3	1 1	1 1	1 1	11
1(OR$_4$)				12
1	1	1	1	

Table 1. A $[1024, 4]$ saturating counter using (3,2) and (2,2) counters. It uses 892 (3,2) counters, 5 (2,2) counters and 49 4-input OR gates.

An ($\{m, i, j\}$, 3) unit, shown in Figure 2, is a saturating parallel counter which receives m inputs of weight 2^{k-1}, i inputs of weight 2^{k-2} and j inputs of weight 2^{k-3}. It produces three outputs of weights 2^{k-1}, 2^{k-2} and 2^{k-3} where $2^{k-1} = T$ is the threshold of the saturating counter.

We restrict our discussion to the case where $2 \leq$

2^3	2^2	2^1	2^0	Level
			205(4;2)	
		410	205	1
		90(4;2)	41(4;2)	
	90+90 180	91+41 132	41 41	2
	41(4;2) 1(3,2)	28(4;2)	8(4;2)	
82+1 83	42+28 70	28+8 36	9 9	3
20+5(OR_4)	15(4;2) 1(2,2)	7(4;2) 1(3,2)	2(4;2)	
30+1+8 39	16+7+1 24	8+2 10	2 2	4
9+3(OR_4)	5(4;2)	2(4;2)	1(2,2)	
3+10 13	6+2 8	2+1 3	1 1	5
3+1(OR_4)	2(3,2) 1(2,2)	1(3,2)		
1+3 4	3+1 4	1 1	1 1	6
1(OR_4)	2(2,2)			
1+2 3	2 2	1 1	1 1	7
	1(2,2)			
3+1 4	1 1	1 1	1 1	8
1(OR_4)				9
1	1	1	1	

Table 2. A $[1024, 4]$ saturating counter using (4;2) compressors, (3,2) and (2,2) counters. It uses 444 (4;2) compressors, 5 (3,2) counters, 6 (2,2) counters and 44 4-input OR gates.

2^3	2^2	2^1	2^0	Level
			341(3,2)	
		341	342	1
		113(3,2)	114(3,2)	
	113 113	114+115 229	114 114	2
	37(3,2)	76(3,2)	38(3,2)	
37 37	76+39 115	38+76 115	38 38	3
9+2(OR_4)	38(3,2)	38(3,2)	12(3,2)	
4+38 42	38+39 77	12+39 51	14 14	4
10+3(OR_4)	25(3,2)	17(3,2)	4(3,2)	
3+25 28	17+27 44	4+17 21	6 6	5
7+1(OR_4)	14(3,2)	7(3,2)	2(3,2)	
4+14 18	7+16 23	2+7 9	2 2	6
4+1(OR_4)	7(3,2)	3(3,2)	1(2,2)	
3+7 10	3+9 12	1+3 4	1 1	7
2+1(OR_4)	4(3,2)	1(3,2)		
1+4 5	1+4 5	2 2	1 1	8
1(OR_4)	($\{5,5,2\}$,3)			9
1	1	1	1	

Table 3. A $[1024, 4]$ saturating counter using a ($\{5, 5, 2\}, 3$) unit. It uses 891 (3,2) counters, one (2,2) counter, 43 4-input OR gates and a ($\{5, 5, 2\}, 3$) unit.

$j \leq 3$, for which the maximum carry from the position of weight 2^{k-3} to the position of weight 2^{k-2} is 1. Thus, we have $i + 1$ bits of weight 2^{k-2} to be added. $s_{k-3} = y_1 \oplus \cdots \oplus y_j$ can be replaced, if needed, by $s_{k-3} = y_1 + \cdots + y_j$. For $j \leq 3$ this simplification is not needed since the delay of the two (or less) XOR gates will be smaller than the delay of the gates required to generate s_{k-1}. The contribution of $y_1, \cdots, y_j$ to s_{k-2} can be represented by an additional input of weight 2^{k-2}, i.e.,

$$x_{i+1} = \begin{cases} y_1 y_2 & \text{for } j = 2 \\ y_1 y_2 + y_1 y_3 + y_2 y_3 & \text{for } j = 3 \end{cases}$$

The truth table for the $(\{m, i, j\}, 3)$ unit is:

$z_1 + \cdots + z_m$	$x_1 \cdots x_i \, x_{i+1}$	s_{k-1}	s_{k-2}
1	ϕ	1	ϕ
0	All 0's	0	0
0	A single 1	0	1
0	Two or more 1's	1	ϕ

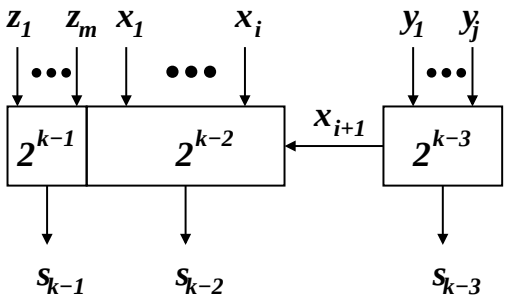

Figure 2. An ($\{m, i, j\}, 3$) unit.

The resulting Boolean expressions are:

$s_{k-2} = x_1 + \cdots + x_i + x_{i+1}$ and

$s_{k-1} = z_1 + \cdots + z_m + (x_1x_2 + x_1x_3 + \cdots + x_ix_{i+1})$

We can substitute x_{i+1} into the expressions for s_{k-2} and s_{k-1}. This would result in product terms with up to three literals in s_{k-1}, i.e., fan-in $\leq$ 3. Notice that the simplified Boolean equation for s_{k-2} may in fact produce s_{k-2}=1 even if the correct value is 0, but only if s_{k-1}=1. Therefore, the probability of producing an output of 8, when the number X of input bits which equal 1 satisfies $X mod\ 8 = 0$ and $X > 8$ (e.g., X=16), is lower than the corresponding probability for the saturating counters of the types depicted in Tables 1 and 2.

To calculate the delay and area of the proposed ($\{m, i, j\}, 3$) unit, some further analysis is required. The total number of signals in an implementation of s_{k-1} after the first level of gates (OR gates for the z inputs and AND gates for the remaining terms) is

$$N_2 = \left(\begin{array}{c} i+1 \\ 2 \end{array} \right) + m\ div\ fi + m\ mod\ fi$$

if $j = 2$, and

$$N_2 = \left(\begin{array}{c} i+1 \\ 2 \end{array} \right) + 2i + m\ div\ fi + m\ mod\ fi$$

if $j = 3$, where fi is the maximum fan-in allowed.

The table below shows the number of signals (after the first level of gates, i.e., OR gates for the z inputs and AND gates for the x_px_q and $x_py_sy_t$ terms) and the number of logic levels for the special case of fan-in $fi = 4$, $m \leq 4$ and $j = 2$.

i	2	3	4	5	6	7	8
N_2	4	7	11	16	22	29	37
# of levels	2	3	3	3	4	4	4

For fan-in $fi \geq 4$ the total number of gate levels is therefore $1 + \lceil \log_4 N_2 \rceil$.

The exact benefit of using an ($\{m, i, j\}, 3$) unit instead of several levels of (3,2) and (2,2) counters is highly dependent on its circuit implementation. For simplicity, we will assume for the numerical results summarized in the next section that the ($\{m, i, j\}, 3$) unit is implemented using basic logic gates with a delay of Δ_G for an OR or AND gate with fan-in=fi or less. The (3,2) and (2,2) counters are implemented using 2-input XOR gates whose delay is denoted by Δ_{XOR}.

4. Numerical Results

Figure 3 compares the delay of an [n,4] saturating counter (for n=72, 136, 264, 520 and 1032) implemented in four different ways: using (3,2) and (2,2) counters only, allowing the use of (4;2) compressors as well, and allowing all types of counters including the special ($\{m, i, j\}, 3$) unit. The latter has two implementations, one with fan-in $fi = 4$ and another with $fi = 3$. Only the basic design, which is restricted to the use of (3,2) and (2,2) counters, is unique. The remaining three designs have multiple possible implementations and the delay of the fastest implementation for that type is shown. The delays in Figure 3 are measured in terms of Δ_{XOR} under the assumptions that $\Delta_G = 0.5\Delta_{XOR}$, $\Delta_{(2,2)} = \Delta_{XOR}$, $\Delta_{(3,2)} = 2\Delta_{XOR}$ and $\Delta_{(4:2)} = 3\Delta_{XOR}$. The results indicate that the use of (4;2) compressors reduces the delay by no more than one Δ_{XOR} in the given range of inputs, but

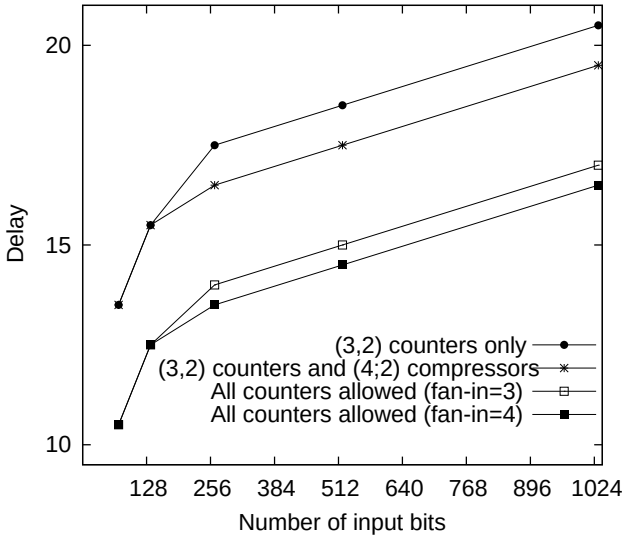

Figure 3. Delay comparison (the delay unit is Δ_{XOR}).

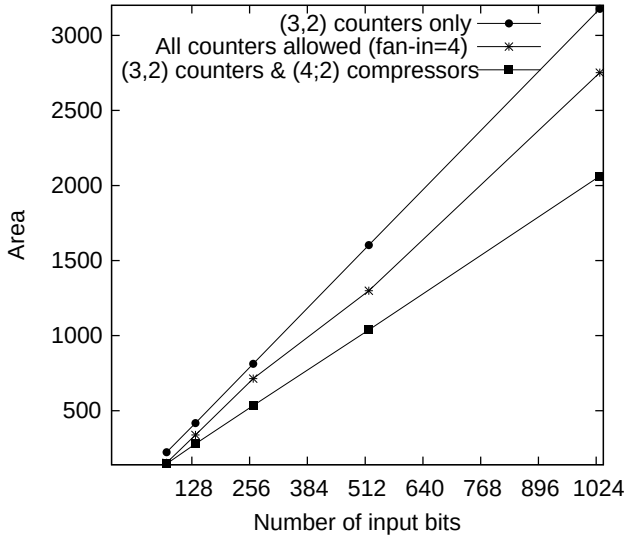

Figure 4. Area comparison (the area unit is A_{XOR}).

using in addition an $(\{m, i, j\}, 3)$ unit further reduces the delay by 3 Δ_{XOR} or 2.5 Δ_{XOR} for fan-in $fi = 4$ or 3, respectively.

Figure 4 shows the estimated area for three of the design alternatives. The area is measured in terms of A_{XOR} under the assumptions that $A_G = 0.5A_{XOR}$, $A_{(2,2)} = 1.5A_{XOR}$, $A_{(3,2)} = 3.5A_{XOR}$ and $A_{(4:2)} = 4.5A_{XOR}$. The area of an $(\{m, i, j\}, 3)$ unit is determined by the number of gates required. Figure 4 shows that the reduction in total area due to the use of (4;2) compressors increases with the number of inputs (under the above-mentioned area ratios assumption). The use of an $(\{m, i, j\}, 3)$ unit may increase the total area but the area will still be lower than that of the basic design using only (3,2) and (2,2) counters.

To make a decision regarding the use of an $(\{m, i, j\}, 3)$ unit, the designer should consider the delay as well as the area. A measure like *Area* $\times$ *Delay*^{2} can help, and Figure 5 shows that the use of an $(\{m, i, j\}, 3)$ unit is beneficial.

As mentioned above, a design using (4;2) compressors and an $(\{m, i, j\}, 3)$ unit is not unique and therefore one can trade off area and delay. Figure 6 illustrates such a tradeoff for n=520 and 1032 where the basic designs (using only (3,2) and (2,2) counters) are also shown for reference. Note that a reasonable reduction in the area can be achieved with some increase in delay. Further increases in the delay will only marginally reduce the area, and thus are not advisable.

5. Conclusion

Saturating counters have been defined and several design alternatives have been presented and evaluated. The motivation for this study was the need to design such a counter as part of a self test and repair unit for an embedded memory in a system on a chip. The saturating counter that has been implemented uses (3,2) counters and an $(\{m, i, j\}, 3)$ unit. It has been implemented using the Perfect SAGE standard cell library for 0.15micron TSMC CMOS from Artisan. A preliminary design which did not use an $(\{m, i, j\}, 3)$ unit did not satisfy the timing requirements.

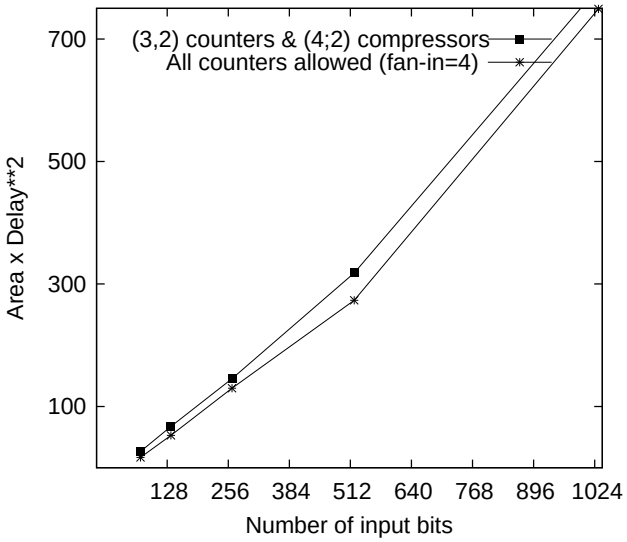

Figure 5. Area $\times$ Delay^2 comparison.

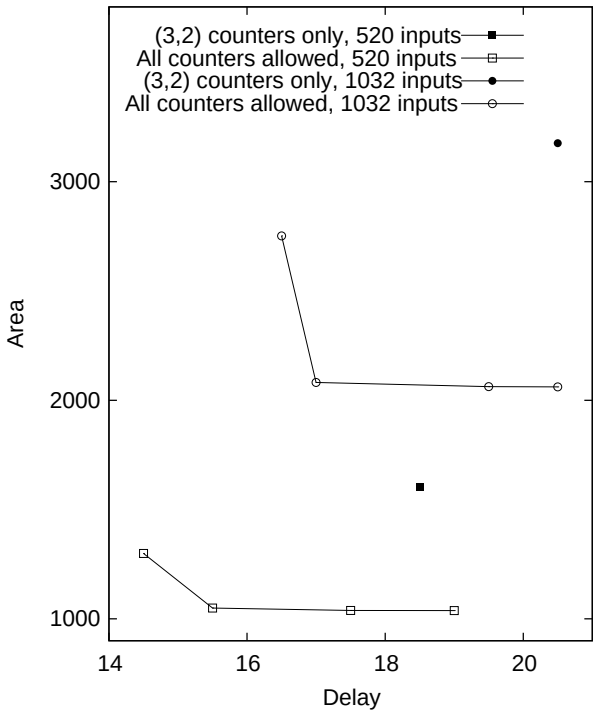

Figure 6. Area vs. delay tradeoff.

References

[1] R. F. Jones and E. E. Swartzlander, "Parallel Counter Implementation," *Journal of VLSI Signal Processing,* pp. 223-232, 1994.

[2] E. E. Swartzlander, "Parallel Counters," *IEEE Trans. on Computers*, Vol. C-22, pp. 1021-1024, 1973.

[3] I. Koren, *Computer Arithmetic Algorithms*, 2nd edition, A K Peters, Natick, MA, 2002.

[4] I. Koren and Z. Koren, "Defect Tolerant VLSI Circuits: Techniques and Yield Analysis," *Proceedings of the IEEE*, Vol. 86, pp. 1817-1836, Sept. 1998.

[5] H. C. Ritter and B. Muller, "Built-In Test Processor for Self-Testing Repairable Random Access Memories," *Proceedings of the International Test Conference*, 1987, pp. 1078-1084.

[6] R. Treur and V. K. Agarwal, "Built-In Self-Diagnosis for Repairable Embedded RAMs," *IEEE Design & Test of Computers,* June 1993, pp. 24-33.

[7] Y. Nagura *et. al,* "Test cost reduction by At-speed BISR for embedded DRAMS," *Proceedings of the International Test Conference*, 2001, pp. 182-186.

[8] N. Ohkubo, M. Suzuki, *et. al.*, "A 4.4-ns CMOS 54 $\times$ 54-b Multiplier Using Pass-Transistor Multiplexor," *IEEE Journal of Solid-State Circuits*, vol.30, pp. 251–256, Mar. 1995.

[9] http://www.ecs.umass.edu/ece/koren/arith/simulator/SatCount/