Reducing Parity Generation Latency through Input Value Aware Circuits

Yusuf Osmanlıoğlu, Y. Onur Koçberber, Oğuz Ergin
TOBB University of Economics and Technology
Söğütözü Cad. No: 43 Söğütözü, Ankara/TÜRKİYE
{yosmanlioglu, yokocberber, oergin}@etu.edu.tr

ABSTRACT

Soft errors caused by cosmic particles and radiation emitted by the packaging are an important problem in contemporary microprocessors. Parity bits are used to detect single bit errors that occur in the storage components. In order to implement parity logic, multiple levels of XOR gates are used and these XOR trees are known to have high delay. Many produced and consumed values inside a processor hold consecutive zeros and ones in their upper order bits. These values can be represented with less number of bits and hence are termed narrow. In this paper we propose a parity generator circuit design that is capable of generating the parity if the input value is narrow. We show that parity can be generated faster than a regular XOR tree implementation using our design for the values that can be represented using fewer bits.

Categories and Subject Descriptors

B.7.3 [**Integrated Circuits**]: Reliability and Testing – *Error-checking*

General Terms

Performance, Design

Keywords

Error Correcting Codes (ECC), Narrow Values, Parity Generation

1. INTRODUCTION

With every new generation of microprocessors, feature sizes shrink, supply voltages are reduced and die sizes are increased. It is expected that there will be more soft errors (or transient errors) that are caused by high energy cosmic particles and radiation from the packaging in the near future [8]. Error correcting codes and parity bits are widely used for detecting soft errors. These techniques generally rely on counting the number of ones inside a value and are known to have high delay overhead [3][7]. Input variations, are one of the parameters that effect circuit latency [2][9][10]. For different types of inputs, logic circuits don't generate results exactly with the same latency.

A circuit produces the result faster if the input vector applied to its inputs doesn't activate its longest path which has the highest latency. It is reported by many researchers that a large percentage of produced and consumed values can be represented by using less number of bits than is provided by the datapath of a processor [1][5][6]. These values (called the "narrow values") store unnecessary replicated sign bits in their upper part. Usually, circuits work faster with narrow values since the logic complexity decreases when the number of input bits is reduced. One example that exploits this fact is the "double pumped" ALUs of the Intel's Pentium 4 [4] which takes 3 fast cycles to produce a 32 bit result.

In this paper we propose a parity generator circuit that is able to generate the parity and ECC signals faster when the input value is narrow. We show that narrow value detector logic works faster than the lower level XOR tree and the output of the narrow value detector can be used to reduce the number of XOR levels that needs to be passed in order to generate the parity bit.

2. MODIFIED PARITY GENERATOR WITH VARIABLE LATENCY

Many produced and consumed values in contemporary microprocessors are known to be narrow [1][5][6]. A value is identified as "narrow" if it can be represented with fewer bits than the datapath width. Narrow values contain a large number of consecutive 1s or 0s in their upper order bits. These repeating bits can be compressed into a single sign bit.

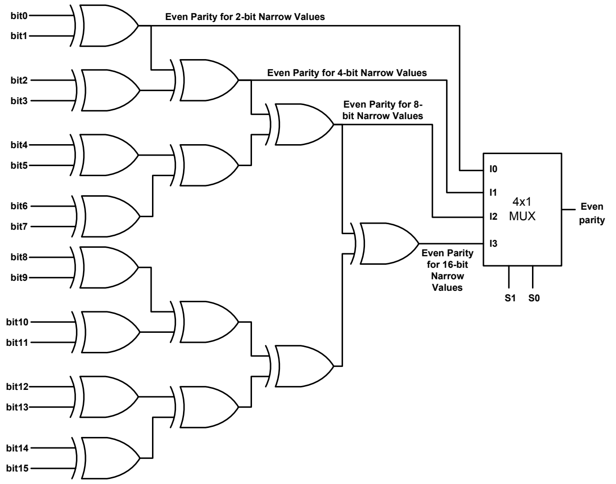
bit0
bit1
bit2
bit3
bit4
bit5
bit6
bit7
bit8
bit9
bit10
bit11
bit12
bit13
bit14
bit15
Even Parity for 2-bit Narrow Values
Even Parity for 4-bit Narrow Values
Even Parity for 8-bit Narrow Values
Even Parity for 16-bit Narrow Values
I0
I1
I2
I3
4x1 MUX
Even parity
S1
S0

Figure 1. Proposed parity generator circuit (Example is for 16 bits)

Since the result of an XOR operation on an input that contains all 0s or all 1s is always zero, calculation of parity for the upper order bits of the narrow values is unnecessary. As the result of XORing a value with zero is equal to the value itself, calculation of parity for the lower order bits is enough for values that has repeating zeros or ones in their upper order portion. Therefore it is possible to calculate parity bits faster for the narrow values by detecting them and generating the parity only for the lower part of the value.

An example parity generator circuit that has variable latency depending on the incoming value width is depicted in Figure 1. The XOR tree already calculates the parity values for different portions of the data. These internal nodes of the parity generator circuit are connected to a multiplexer circuit whose select inputs are connected to the narrow value identifier. When the narrow value identifier circuit signals a narrow value, corresponding parity value is selected and the overall parity is generated faster than the regular parity generator. We used regular combinational logic to implement the multiplexer in Figure 1. In order to identify narrow values we used the consecutive zero and one detector circuits that employs dynamic logic for fast operation. Output of these detector circuits are first precharged to V_{dd}. Afterwards the output node is discharged if any of the inputs is not zero (or one if inverted inputs are used). Zero detector circuits can be scaled for large number of inputs. Because of the increased capacitance of the output node, as the size of the narrow value that needs to be detected decreases, number of inputs for the zero detector increases which results in higher latency and energy dissipation.

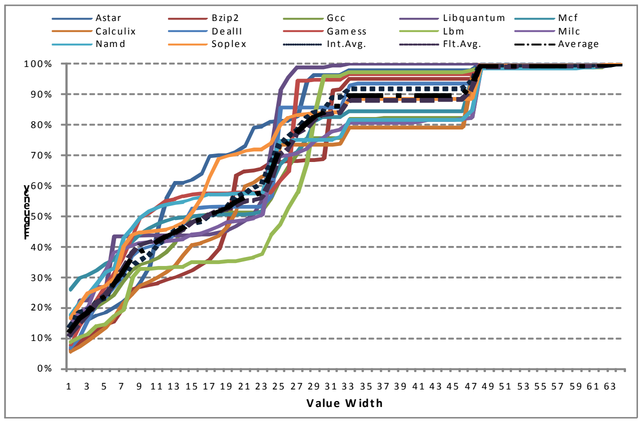

Figure 2.Width histogram of values written to the integer register file.

Table 1. Configuration of the Simulated Processor

Parameter	Configuration
Machine width	4-wide fetch, 4-wide issue, 4 wide commit
Window size	64 entry issue queue, 64 entry load/store queue, 128–entry ROB
Function units and latencies (total/issue)	4 Int Add (1/1), 1 Int Mult (3/1) / Div (20/19), 2 Load/Store (2/1), 2 FP Add (2), 1FP Mult (4/1) / Div (12/12) / Sqrt (24/24)
L1 I–cache	32 KB, 2–way set–associative, 64 byte line, 1 cycle hit time
L1 D–cache	64 KB, 4–way set–associative, 64 byte line, 2 cycles hit time
L2 Cache unified	2 MB , 8–way set–associative, 128 byte line, 6 cycles hit time
BTB	4K entry, 2–way set–associative
Branch Predictor	Combined with 1K entry Gshare, 8 bit global history, 4K entry bimodal, 1K entry selector
Memory	256 bit, 300 cycles first part, 1 cycle interpart
TLB	32 entry (I) – 2-way set-associative, 128 entry (D) – 16-way set associative, 12 cycles on miss

3. SIMULATION METHODOLOGY

In order to get an accurate idea of the percentage of narrow values inside current x86 processors, we used the PTLsim simulator [11] which is capable of executing 64-bit x86 instructions. We executed 12 of the SPEC 2006 benchmarks that we could compile and run on the simulator to observe the percentage of the narrow values in different workloads. All benchmarks were compiled with maximum optimizations to target the x86-64 instruction set. Each benchmark is run for 2 billion instructions. Table 1 shows the configuration of the simulated architecture. We used the Cadence design tools together with the UMC 90nm technology to get an accurate idea of the energy dissipation and latency numbers for the proposed technique.

4. RESULTS AND DISCUSSIONS

Success of the proposed technique depends on the presence of the narrow values in common workloads. Luckily a large percentage of produced and consumed values in the contemporary workloads are narrow. Figure 2 and Figure 3 shows the percentage of narrow values written to the integer register file and the immediate field of the issue queue respectively for spec 2006 benchmarks. For each benchmark, a separate line shows what percentage of the operands have smaller width than that is indicated by the number on the x-axis. As Figure 2 reveal, almost 40% of the values that are written back to the registers can be represented with 8 bits or less for spec 2006 benchmarks on the average. Some individual benchmarks show even higher percentages.

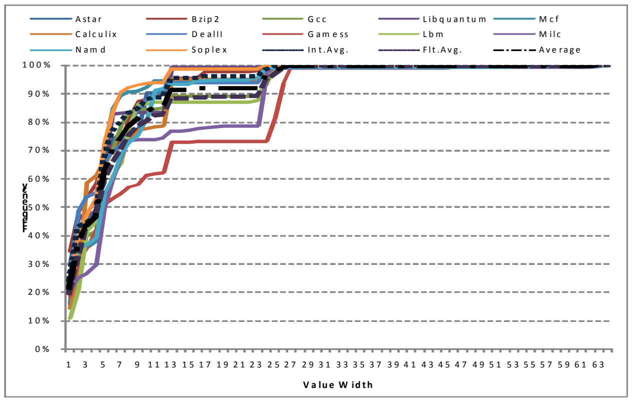

Figure 3.Width histogram of the immediate values.

As the Figure 3 presents, the immediate values that belong to the instructions that are written the issue queue require even fewer bits than it is provided by the storage space. On the average across all benchmarks, more than 75% of the immediate values can be represented by only 8 bits.

Table 2. Delay and Energy Dissipation of XOR Parity Generator

Bits	TPLH (ps)	TPHL (ps)	Energy (fJ)
2	23,818	15,050	103,100
4	70,467	40,661	158,679
8	138,173	70,189	223,118
16	228,580	102,712	297,649
32	343,169	136,761	384,299
36	344,082	142,656	427,167
64	483,846	170,950	486,092

Delay of the parity generator circuit depends on the delay of a single 2-input XOR gate. Since the XOR gate is not a simple gate to implement, its latency is higher than other devices. Delay and energy dissipation of the parity generator circuit for different number of inputs is show in Table 2. Columns in Table 2 show the energy dissipation and the FO4 delay numbers of parity generators with 2, 4, 8, 16, 32, 36, 64 inputs. TPLH column shows the delay for the output signal to go from low to high and TPHL column shows the delay for the output signal to go from high to low. As the table reveals, the delay of the parity generator circuit increases with increasing number of inputs. Delay of a consecutive zero detector circuit is relatively small when compared to the parity generator circuit. FO4 delay and energy dissipation of the circuit is shown in the Table 3. Delay of the consecutive zero detector circuit has two components: precharging delay and evaluation delay. Circuit cannot be used until its output node is precharged to logic 1. Therefore the total minimum delay of the circuit is the total of these two delay components. Note that, even the

Table 5. Bit Patterns for the ECC bits for spec 2006 benchmarks

		Floating Point Register File							Integer Register File							Immediates in the Issue Queue						
		1	2	4	8	16	32	64	1	2	4	8	16	32	64	1	2	4	8	16	32	64
ECC1	Int.	52%	0%	0%	0%	18%	3%	27%	16%	1%	3%	7%	17%	5%	50%	47%	0%	4%	1%	2%	0%	45%
	Flt.	49%	0%	0%	0%	0%	0%	51%	16%	2%	7%	7%	16%	7%	46%	35%	1%	4%	3%	5%	0%	52%
ECC2	Int.	52%	0%	0%	0%	9%	2%	37%	17%	1%	3%	7%	16%	5%	52%	44%	1%	3%	1%	1%	0%	50%
	Flt.	50%	0%	0%	0%	0%	2%	48%	16%	2%	5%	6%	16%	8%	47%	34%	2%	4%	3%	4%	0%	53%
ECC3	Int.	52%	0%	0%	0%	6%	1%	41%	16%	1%	3%	7%	18%	5%	50%	38%	1%	5%	1%	1%	0%	54%
	Flt.	50%	0%	0%	0%	0%	0%	49%	14%	2%	5%	6%	16%	7%	50%	32%	2%	4%	3%	4%	0%	54%
ECC4	Int.	52%	0%	0%	0%	16%	32%	0%	33%	1%	3%	11%	12%	41%	0%	80%	0%	1%	1%	1%	18%	0%
	Flt.	53%	0%	0%	0%	0%	47%	0%	38%	0%	1%	9%	14%	38%	0%	63%	0%	0%	2%	3%	31%	0%
ECC5	Int.	52%	0%	0%	0%	21%	27%	0%	35%	1%	3%	2%	20%	39%	0%	80%	0%	1%	0%	1%	19%	0%
	Flt.	51%	0%	0%	0%	0%	49%	0%	37%	0%	1%	3%	20%	39%	0%	63%	0%	0%	1%	4%	31%	0%
ECC6	Int.	52%	0%	0%	0%	0%	48%	0%	32%	1%	3%	2%	9%	53%	0%	80%	0%	1%	0%	1%	19%	0%
	Flt.	59%	0%	0%	0%	0%	41%	0%	37%	0%	1%	3%	6%	53%	0%	63%	0%	0%	1%	2%	33%	0%
ECC7	Int.	56%	0%	1%	43%	0%	0%	0%	22%	1%	3%	74%	0%	0%	0%	32%	1%	4%	63%	0%	0%	0%
	Flt.	61%	0%	1%	37%	0%	0%	0%	15%	1%	4%	79%	0%	0%	0%	27%	1%	2%	69%	0%	0%	0%

total delay of the largest zero detector with 62 inputs, is well below the delay of the 64 bit parity generator XOR tree which has a latency of 483ps. On the other hand the delay of the parity circuit with 8-bit inputs is equal to 138ps which is a significantly smaller number when compared to a wide 64-bit parity generator circuit. If it is detected that the parity of an incoming input value can be calculated by just employing an 8-bit XOR tree circuit, the parity generation latency can be reduced.

Table 3. Delay and Energy Dissipation of Consecutive Zero Detector

Bit	TPHL (ps)	Precharge (ps)	Energy (fJ)
8	36,427	18,683	44,900
16	65,471	30,990	69,713
32	117,340	74,720	118,477
48	174,086	77,592	162,450
56	200,639	90,189	181,092
60	214,281	96,564	190,516
62	215,557	97,113	195,572

Although the latency of the 56-bit zero detector circuit is not as low as the delay of 8-bit parity generator, the selection inputs of the multiplexer shown in Figure 1 will be ready well before the 64 bit parity generator circuit generates a result but after the 8-bit parity is generated. The only downside of the proposed technique is to pay a delay penalty of the multiplexer circuit when the incoming value turns out to be wide. Table 4 shows the FO4 delay figures of the combinational multiplexer circuits that we use to implement the selection logic in Figure 1. If the proposed circuit is designed to exploit a large spectrum of narrow values, the number of inputs on the multiplexer increases. This increase comes together with a delay overhead which should be taken into account when making the design choice.

Table 4. Delay of the Multiplexer

Inputs	TPLH (ps)	TPHL (ps)
2	67,265	56,567
4	135,798	117,048
8	203,744	178,545

In order to try the proposed circuitry and reduce the complexity of the multiplexer in Figure 1 we chose to identify only 8-bit narrow values so that the system generates parity value faster for the values that can be represented with 8 bits. In this case, since the upper order 56 bits have to be identical, two 56 bit leading zero and leading one detector circuits are included. Outputs of these circuits are ORed and directly connected to the select input of the multiplexer shown in Figure 1. When the input value is an 8 bit narrow value, parity result can be generated as soon as the XOR result for the lowest order 8-bits is calculated and this 8-bit parity result propagates through the multiplexer. Note that the narrow value identification and 8-bit parity generation operation is done in parallel and the maximum of both delays is required to pass before the select input of the multiplexer can be considered stable. When the Hamming code is applied to 64 bits, 7 additional bits are used for encoding the value. The first three of these additional bits (termed ECC 1, 2 and 3 for convenience) are 36 bits long, following three bits (ECC 4, 5, 6) are 32 bits long and the last bit, ECC 7, is 8 bits long.

Table 5 shows the widths of the values formed by the inputs used to generate the corresponding ECC bit in the floating point register file, integer register file and the immediate field of the issue queue respectively. For each ECC bit, the percentage of data value widths are shown on average for integer and floating point benchmarks of spec 2006 benchmark suite. The columns reserved for 64 bit ECC are all 0% and shaded to show that these are not applicable to ECC bits 4, 5, 6 and 7. Similarly, ECC bit 7 just uses 8 bit inputs and therefore does not have a percentage in the 16 and 32 bit columns. Results show that a large percentage of the ECC input bits are in fact themselves narrow and the use of the proposed technique can reduce the latency of the generation of ECC signals for most of the input combinations. Especially the immediate values that come from the compiler and are stored inside the issue queue seem to be mostly narrow. It is possible to generate the Hamming Code bits faster than the regular values for most of the immediate values stored inside the issue queue.

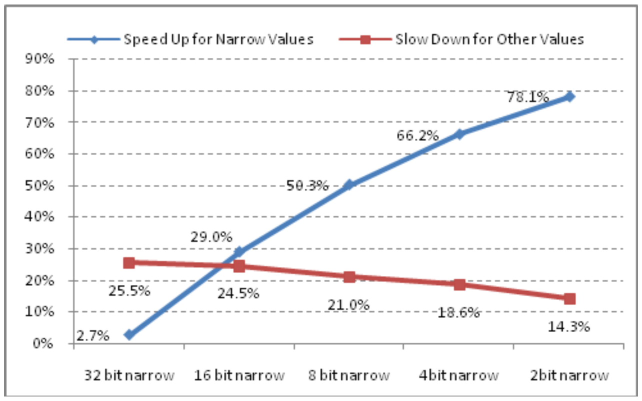

Figure 4. 64 bit XOR parity generator speedup and slow down rates.

Figure 4 shows the results obtained from our circuit level simulations on the benefits of the proposed scheme when applied on a 64 bit parity generator. The line with the diamond shaped data points shows the speedup achieved by using the proposed scheme for different narrow input definitions when compared to the 64 bit baseline parity generator. The other line with the square shaped data points shows the increase in the parity generation delay when the incoming value is wide with a narrow definition shown on the x axis. Our simulation results show that the 64 bit parity is generated in 105 ps for 2-bit narrow values and 553 ps for regular 64 bit values using the proposed technique. This is a 78.1% decrease in parity generation latency for 2 bit values and a 14.3% increase in latency when generating parity for the other values. As the proposed technique is used with smaller narrow value definitions, the reduction in parity generation latency increases and the slowdown for regular values decreases. However as the narrowness is defined smaller, the percentage of values that can benefit from the proposed technique decreases as shown in Figures 2 and 3.

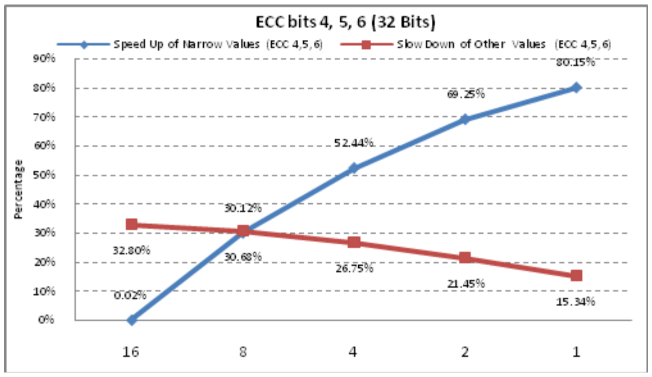

Figure 5 . Speedup and slowdown for generating ECC bits 4, 5, 6.

36 bits, 32 bits and 8 bits parity generators are also simulated to show that ECC values are generated faster. Simulation results revealed that if there are less than 3 levels of logic between the multiplexer and the selected internal node of the parity generator, there is no effect on the critical path delay. However, as Table 5 reveals most of the values in different structures are 1 bit narrow, which is the highest logic level possible to reach the output. 36 Bit ECC simulation showed that 1 bit narrow value is generated 80,2% faster in 68 ps. Values other than 1 bit narrow is slowed down 15,1 % , generated in 396ps. Figure 5 shows the results for ECC bits 4, 5 and 6 which use 32 bit input operands. Similar to the parity generator results shown in Figure 4, it is possible to generate ECC signals faster when the output of the XOR gates that operate with the least significant bits is used.

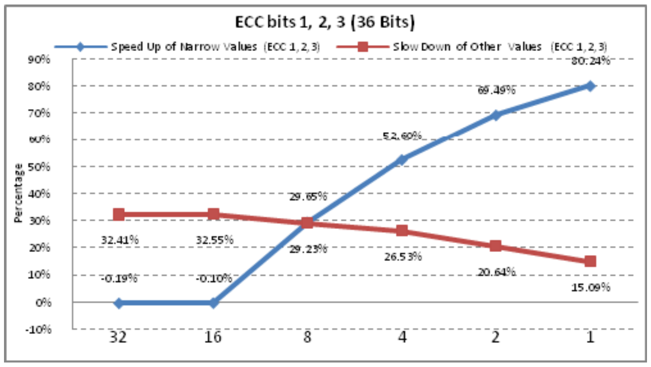

Figure 6. Speedup and slowdown for generating ECC bit 1, 2 and 3.

Figure 6 shows the speedup and slowdown graphs for ECC bits 1, 2 and 3 which use 36 bit values as input vectors. The figure is also similar to the Figures 4 and 5 in showing an increase in latency benefits as the input values get narrower. Results show that the proposed design itself is capable of reduction the latency up to 80%. However as the values get narrower, the frequency of such values also decreases which reduces the number of values that can be covered by the proposed scheme. For 32 and 16 bit inputs, our scheme shows no benefits for narrow values and shows only the worst case critical path delay from the inputs to the output of the multiplexer. Application of the proposed technique to ECC bit 7 results in some harsh tradeoffs as only 8 bits are used to generate the signal. Our simulation results show that it is possible to reduce the latency of the signal generation by around 50% for 1 bit narrow values (all 0 or all 1s). However in that case, ECC bit is generated 42% slower for any value larger than 1 bit. Defining the narrow value with 2 bits also show 23% latency reduction at the expense of 59% slowdown in wider values. For ECC bit 7 largest benefits are achieved in the floating point register file as more than 55% of the values written to the floating point register file can be represented with a single bit.

4. CONCLUSIONS

We proposed a parity generator circuit which generates parity faster if the incoming value is narrow. Proposed circuit relies on the observation that many of the produced and consumed values in a microprocessor are narrow. Our proposal can be used by microarchitects when full parity protection is not usable due to its high delay overhead. If the workload of a microprocessor contains a high percentage of narrow values and partial protection satisfies the reliability targets for a given data holding component, proposed solution can be used to generate parity only for the narrow values. Alternatively this circuit can generate parity in one clock cycle for narrow values and in two clock cycles for wider values. The proposed technique reduces the parity generation latency of 64 bit values by 50% for 8 bit narrow values. Considering the fact that around 70% of the immediate values written to the immediate field of the issue queue and around 40% of the value written to the integer register file can be expressed with only 8 bits, the coverage of the proposed scheme is quite high. However the proposed scheme may lead to partial parity protection as it slows down the parity generation for values larger than 8 bit by around 21%.

5. ACKNOWLEDGMENTS

This work was supported in part by the Scientific and Technological Research Council of Turkey (TUBITAK) through research grant 107E043.

6. REFERENCES

[1] Ergin, O., et al., “Exploiting Narrow Values for Soft Error Tolerance”, IEEE Computer Architecture Letters, 2006

[2] Ernst, D., et al., “Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation”, in *MICRO*, 2003

[3] Furutani, K., et al., “A Built-In Hamming Code ECC Circuit for DRAMs”, IEEE Journal of Solid–State Circuits, Vol. 24, No: 1, February 1989.

[4] Hinton, G., et al., “The Microarchitecture of the Pentium 4 Processor", *Intel Technology Journal*, Q1, 2001

[5] Hu, J., et al., “In-Register Duplication: Exploiting Narrow-Width Value for Improving Register File Reliability”, in DSN 2006.

[6] Loh, G., “Exploiting Data-Width Locality to Increase Superscalar Execution Bandwidth”, in *MICRO*, 2002

[7] Phelan, R., “Addressing Soft Errors in ARM Core-based Designs”, White Paper, ARM, December 2003

[8] Semiconductors Industry Association (SIA), International Technology Roadmap for Semiconductors 2005, http://www.itrs.net/Links/2005ITRS/Home2005.htm

[9] Unsal, O.S., et al., “Impact of Parameter Variations on Circuits and Microarchitecture”, IEEE Micro Magazine, Vol. 26, No. 6, November/December 2006, pp.30-39.

[10] Vera, X., Unsal, O., González. A., “X-Pipe: An Adaptive Resilient Microarchitecture for Parameter Variations”, in ASGI 2006.

[11] Yourst, M. T., ‘PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator’. ISPASS 2007.