Power Aware Encoding for the Instruction Address Buses Using Program Constructs

Prakash Krishnamoorthy* and Meghanad D. Wagh*

Abstract— **This paper examines the address traces produced by various program constructs. By using the correlation induced by the program constructs within these traces, we develop a scalable bus encoding algorithm to significantly reduce the switching activity on the instruction address bus. Simulation results for Spec2000 benchmarks show that for modest coding complexity, the proposed scheme reduces switching activity on the instruction address bus by as much as 88% and the overall bus power by as much as 71%.**

Keywords: Low power, Switching activity, Address bus, Embedded systems.

1 Introduction

With the growing demand for mobile and embedded systems, power consumption has become an important design criteria. For many such systems built around processor cores or custom ASICs, off chip bus power [1] represents a dominant portion of the total system power. This work focuses on systems that use Harvard architecture employing independent data and instruction address buses. Examples of such systems range from simple embedded processor based systems to systems based on cores implemented on Programmable Gate arrays where bus activity is a significant source of power dissipation [2].

The problem of reducing power in on-chip address and data bases in deep submicron technologies has been approached on various fronts [3]. For such buses, crosstalk and leakage are major contributors towards total bus power as opposed to capacitance of the off-chip buses. In the former, attention has been given to reduction of crosstalk [4, 5] and leakage power consumption that arise due to the large number of buffers that are used to manage signal slew rates and intrinsic delays. Based on our experience working with off-chip buses, crosstalk is not the main factor as ample time and effort is spent in routing these bus traces on multilayer boards. Further, the components on the board exhibit little or no leakage. In these cases, interconnect bus capacitance is the major contributor to bus power and methods to reduce bus transition activity are extremely important. We do precisely that in our work where we show that a method based on program constructs delivers the best possible reduction in switching activity on the instruction address buses. We also demonstrate the power reduction by simulating and comparing our method with three best methods using spec2000 integer benchmarks run on the simple-scalar tool set [6]. For work on power reduction methods on data buses, the interested reader can refer elsewhere [1, 7].

Off-chip buses with their large capacitive loads consume a substantial amount of power for every transition. In an instruction address bus, when addresses change sequentially from 0 to 2^n, one can expect as many as $2^n - 1$ transitions on the bus. This number could be even larger (on an average, $n2^{n-1}$) if the 2^n address transitions are random. Naturally, reduction of switching activity associated with buses has attracted a lot of attention. Bus switching may be reduced by encoding the bus data. Such encoding typically takes into account the predictability of the data. Unfortunately, in a general computing environment where bus data is not deterministic, one can only exploit the statistical properties of the samples being transferred on the bus, and in particular, the temporal correlation of successive samples. The statistics that have been employed to date include properties of the individual samples (such as their weights), their temporal correlation and their variance (or localization). This paper shows that by exploiting the statistics resulting from common program constructs, one can further reduce the address bus switching activity significantly.

Bus encoding techniques can be grouped in three broad categories. The first category of encoding algorithms such as the *Bus Invert Method* [8], *Frequent Value codes* [9] and *Self organizing lists* [10] are designed to use statistics of individual samples. The Bus Invert Method and its variants judiciously complement the bus data based on its weight to guarantee an upper bound on the amount of switching activity. Frequent Value codes and Self organizing lists memorize addresses that appear on the address bus repeatedly and map them to code words with lower switching activity.

The second category involves techniques such as the *T0 code* [11, 12] and its variants which take advantage of the correlation between consecutive samples. T0 code is

*Department of Electrical and Computer Engineering, Lehigh University, Bethlehem, PA 18015. Tel: (610)758-4142, Email: prk2@Lehigh.Edu, mdw0@Lehigh.Edu

based on the principle that in the sequential code execution, instruction addresses increase by a constant stride. Since the uniformly increasing addresses can be easily regenerated at the receiver, bus contents may be frozen to completely eliminate bus switching. T0 codes require an additional bus line to indicate whether the addresses are in stride increments or random jumps. It is also possible to modify the T0 code such that the requirement of additional bus line is eliminated [13]. Encodings that use several combinations of *bus-invert* and *T0* encodings have also been developed [14]. The concept of correlation between consecutive samples can be extended to correlation between nearby samples. The *Offset XOR SM* and *Offset XOR SMC* codes minimize switching activity for addresses spaced by small offsets [13]. This may work well for data organized in small structures. While all these techniques work very well for sequential code, they deliver inferior performance when they encounter (large) loops, jumps and subroutine calls where there is a repetitive address pattern.

The third category of bus encoding techniques such as the *Work Zone Encoding* (WZE) [15] and the *Dynamic Sector Encoding* (DSE) [16] attempt to remedy the poor performance of previous bus encoding methods in the presence of multiple sequences of correlated addresses. These techniques partition program address traces into segments and assume that addresses within a segment are highly correlated. The *Dynamic Sector Encoding* uses two registers that track up to two sequences of correlated addresses at a time. This technique produces no switching for consecutive addresses that increase by a stride, but cannot cope with more than two address sequences. For every monitored zone in the WZE method, a zone register holds some address from that zone. The encoding algorithm transmits on the bus a zone identifier and the distance of the current address from the address in the relevant zone register. The *Work zone encoding* technique delivers superior performance in terms of switching activity reduction when encountering multiple address sequences, but the encoder and decoder themselves consume significant amount of power. So, between sector and work zone encoding, the effective method seems to be dependent on the ratio of the capacitance of internal nets to the external bus lines.

In this work, we present a bus encoding method whose encoder/decoder power consumption and switching activity reduction is far superior to that of *WZE* and *DSE* methods. The bus encoding algorithm presented here examines instruction address traces issued by the processor in response to program constructs such as sequential execution, loops, while statements, if-then-else blocks, jumps and subroutine calls. Each of these constructs produces address sequences that can be characterized as groups of addresses that change by a fixed stride. The starting address of each group depends on a specific branch address in the program. By saving these start addresses, the receiver can guess the current address on the bus from the group to which it belongs. Since minimal information about the address needs to be transferred, a power efficient bus transfer occurs.

The rest of the paper is organized as follows. Section II illustrates correlation between program constructs and the address trace. This section also presents the proposed *Program Construct Based Encoding (PCBE)* algorithm that uses this correlation. Simulation results comparing the performance of the PCBE algorithm with other bus encoding algorithms is presented in Section III. Section IV presents our conclusions.

2 New Bus Encoding Algorithm

There is a close relationship between program constructs and the switching activity on the Instruction address bus. This relationship provides significant insight to developing an efficient bus encoding algorithm.

Program instructions can be broadly classified as sequential and branching. Sequential constructs constitute about 70% of any typical program [17]. Non-branching portion of a code produces an instruction address trace in which consecutive addresses increase by a constant (known) stride. Branch control constructs such as if-then-else blocks, while statements, interrupts, jumps (including conditional jumps), loops and subroutine calls cause the instruction address to change abruptly to a value that has no relation to the prior address.

Most prior bus encoding algorithms considered each occurrence of a branch target address as a random *unknown* address. However, many branch target addresses repeat often in a program execution. Examples of these include branching addresses generated by program constructs such as the interrupt calls and loops. If an external event (such as a key being pressed) occurs, the program always calls the same interrupt address. Similarly, during all the iterations of a loop, the same branch address representing the beginning of the loop appears in the address trace.

It is well known that most program codes are temporally stable, i.e., they use the same segments of code and subroutines multiple times before changing the context and thereby, the suit of modules and subroutines being used. This implies that for an extended period of time, the same branch addresses representing the starting addresses of subroutines appear in the address trace. This same observation also applies to branch target addresses generated by conditional and non-conditional jumps, loops and while statements.

In order to take advantage of the repeatability of branch target addresses in the address trace, we propose a new

bus encoding technique called the *Program Construct Based Encoding (PCBE)* algorithm. Let n represent the number of address bits. PCBE algorithm uses one additional line, i.e. a bus that is $n+1$ bit wide. Encoders and decoders realizing PCBE algorithms use $k \leq n$ Branch Target Registers B_i, $0 \leq i < k$. These k registers hold k branch target addresses. In addition, another register, R, holds the stride incremented previous address.

Program Construct Based Encoding (PCBE) utilizes the fact that program constructs produce instruction address traces in the form of sequences of addresses. The addresses in each sequence are separated by constant stride and their starting addresses are random but are likely to occur multiple times within any given time span. The encoder (and the decoder) save these starting addresses in registers B_i and and the stride incremented value of the previous transmitted value in a register R. For every address that needs to be transmitted, encoder first determines if it is available in one of these registers. If the program instruction does not call for branching, the new address matches the one in R. In that event, the encoded bus contents are frozen to avoid switching transitions. If a match is found with B_i, only the i-th bit of the encoded address bus is complemented. Thus the power dissipation is restricted to one bus line.

If the new address is neither in R, nor in B_i, $0 \leq i < k$, then the encoded bus must carry this whole address. In addition, the extra bus line that was added is complemented[1]. Note that the extra bus line is not interpreted as the *miss* signal; rather, its logical value switches every time there is *miss*. This helps limit transitions on this line only to the times when new addresses are transmitted. The new address (without the extra bit) is also recorded in one of the B_i registers. We use the round robin algorithm to allocate a B_i register. This is simple and still takes advantage of the temporal stability of the branching addresses. In other words, the round-robin procedure allows our algorithm to store the branch addresses of the innermost constructs of a program, which have a very high likelihood of being executed frequently. The PCBE encoder algorithm is shown in Fig. 1. The corresponding PCBE decoder algorithm is shown in Fig. 2. For proper operation, the B_i registers in the encoder and the decoder are initialized to match for each i and the R registers are set equal to each other as well. (They all may be initialized to 0 on power-up.) In addition, a one bit storage *prev_miss* is set to 0 in both the encoder and the decoder. Once synchronized this way, the decoder R and B_i registers mirror the contents of the corresponding registers in the encoder for ever. The algorithms ensures that the intended addresses are reproduced correctly.

Our algorithm produces a switching activity of *one* bit for a branch address which has already been stored in

[1]In Fig. 1, $n+1$ bit bus is used, whose 0-th bit is used to carry this *miss* information and bits 1 through n carry the new address.

Encoder(input: addr)
  if $(addr == R)$
    $index = prev\_index$;
    $addr_sent = prev\_addr$;
    $R = R + stride$;
  else
    for $(i = 0; i < targets; i++)$
      if $(B_i == addr)$
        $index = i$;
        $addr\_sent = (prev\_addr\ XOR\ 2^{index})$;
        $R = B_{index} + stride$;
        $miss = 0$;
      else
        $miss = 1$;
  if$(miss = 1)$
    $index = round\_robin()$;
    $B_{index} = addr$;
    $R = addr + stride$;
    $addr\_sent = addr$;
  $addr\_sent = (addr\_sent) * 2$
      $+(miss\ XOR\ prev\_miss)$;
  $prev\_addr = addr\_sent$;
  $prev\_index = index$;
  if $(miss = 1)$
    $prev\_miss = miss\ XOR\ prev\_miss$;
  return $addr\_sent$;

Figure 1: Proposed encoder algorithm

Decoder(input: $addr\_sent$)
  $miss = (prev\_miss)\ XOR\ (addr\_sent\%2)$;
  if $(miss = 0)$
    if $(addr\_sent/2 == prev\_addr)$
      $index = prev\_index$;
      $addr\_received = R$;
      $R = R + stride$;
    else
      $index = log_2((addr\_sent/2)\ XOR\ prev\_addr)$;
      $addr\_received = B_{index}$;
      $R = B_{index} + stride$;
  else if $(miss = 1)$
    $index = round\_robin()$;
    $addr\_received = addr\_sent/2$;
    $B_{index} = addr\_sent/2$;
    $R = (addr\_sent/2) + stride$;
    $prev\_miss = miss\ XOR\ prev\_miss$;
  $prev\_addr = addr\_sent/2$;
  $prev\_index = index$;
  return $addr\_received$;

Figure 2: Proposed decoder algorithm

one of the B_i registers, zero transitions when addresses match the contents of the R register and an average of $(n/2)+1$ transitions on a miss.

Since the PCBE algorithm is specifically designed to work with program constructs, it is worthwhile to evaluate its performance for the common branching constructs.

- **Sequential**: There is no switching activity on the bus when processing sequential addresses.
- **If-then-else construct**: Bus activity occurs only if branches are taken. Each branching, whether to go to the else code or to skirt around it, implies an average of $(n/2+1)$ bit changes. Since one of the branch must be taken, the construct causes $(n/2+1)$ bit transitions on its first occurrence. Note that if the same *if-then-else* code is encountered again within a relatively short period of time, then it is quite likely that the required branch address is in one of the B_i registers and the branching activity in that case will only be 1 bit.
- **For loop**:For this construct, the branching back to the starting of the loop for the first iteration requires PCBE to send the new branching address causing an average of $(n/2+1)$ transitions on the encoded bus. The remaining iterations use the branch address saved to an B_i register and require only 1 bit change on the encoded bus to indicate that branching address. Thus, if a t iteration loop is encountered, then the total expected bit changes on the PCBE bus would only be $(n/2+t)$. If the same loop is encountered again with the branching address still in one of the B_i registers, then the construct will cause only $(t-1)$ transitions on the encoded bus.
- **While loop**: For this construct, the first iteration implies branching to the start of the while loop, a new address, with an average of $(n/2+1)$ bus transitions. However, for the remaining iterations, this address would be in one of the B_i registers and would imply only a single bit change to transmit it on the encoded bus. After all the iterations of the while loop are completed, control branches to the instructions following the while block. This causes, on an average, another $(n/2+1)$ transitions on the bus. Thus the cost of a *while* construct doing t iterations is $(n+t+1)$ bits transitions on its first occurrence. Note however that on subsequent occurrences, the branching addresses would likely be in the B_i registers dropping the total cost of the construct to only $(t+1)$.
- **Call-and-Return**: Clearly, every time a *call* is executed, a new subroutine start address needs to be transmitted over the bus. When a *return* is executed, the program address changes abruptly again and the new address has to transmitted as well. Thus the *call-and-return* construct causes $(n+2)$ bit changes on the encoded address bus. If the subroutine is encountered again, the branch addresses may still be in the B_i registers implying only 2 bit changes on the address bus for the entire construct.

3 Performance Comparison

To evaluate the efficiency of the bus encoding scheme of Section II, we used address traces generated by the spec2000 benchmark programs using a Simple-Scaler/ ALPHA ver. 3.0 tool set [6]. We compared our *Program Construct Based Encoding* (PCBE) algorithm with three best algorithms, namely the *Work Zone Encoding* (WZE) and the *Dynamic Sector Encoding* (DSE) algorithms and *T0-C encoding* methods.

We synthesized the bus encoding and decoding logic for PCBE, WZE, DSE and T0-C using Synopsys synthesis tools and then mapped to a 1.2 Volt, 0.13μ CMOS library. The I/O Voltage was assumed to be 2.5 Volts. Parasitic information (net and cell) was then backannoted into Primepower (Synopsys gate level power analysis tool) and a gate level simulator. Address traces from spec2000 benchmarks were fed into the gate level simulation tool to obtain accurate switching activity information of internal nets using the real delays derived through backannotation. Finally, Synopsis tool *Primepower* was invoked to obtain power dissipation for the various bus encoding schemes. The power dissipation numbers reported in this paper include power dissipation in the encoder, the decoder and the bus. For these simulations, we have used a bus access frequency of 50Mhz and a bus capacitance of 10pf per line.

The total power dissipation (including the power used by the encoders and decoders) of various address bus encoding schemes for five Spec2000 benchmark programs is given in Table 1. The power required without any encoding and the percent saving due to each scheme is also indicated in this table. One can clearly see that the proposed *Program Construct Based Encoding* (PCBE) has substantially higher power savings as compared to other methods for every benchmark program tested.

Note that the Dynamic Sector Encoding (DSE) scheme using exactly two sectors has only one implementation. The T0-C method has a single implementation as well. On the other hand, Work Zone Encoding (WZE) and Program Construct Based Encoding (PCBE) are scalable schemes; WZE allows multiple zones and PCBE allows varying number of registers to store target addresses. In our study, we investigated WZE implementations with up to 16 zones and PCBE implementations with and same number of registers, Table 2 compares the power dissipation of WZE and PCBE for larger number of zones/registers. The percent power saving over

Table 1: Power dissipation (mW) and savings (%) for various bus encoding schemes when WZE uses 2 zones and PCBE uses 2 registers.

Benchmark	No encoding	Sector Encoding		Work Zone Encoding		T0-C Encoding		PCBE	
Program	Power	Power	savings	Power	savings	Power	Savings	Power	Savings
VORTEX	3.55	1.59	55.22%	2.35	33.88%	1.46	58.85%	1.03	70.97%
GCC	3.59	1.93	46.26%	2.70	24.78%	1.76	50.91%	1.33	62.91%
GZIP	3.38	1.36	59.66%	2.23	33.93%	1.44	57.42%	1.01	70.16%
PARSER	3.41	1.60	53.16%	2.41	29.39%	1.51	55.75%	1.08	68.37%
TWOLF	3.39	1.60	52.72%	2.34	31.10%	1.48	56.23%	1.05	68.93%

Table 2: Power dissipation (mW) and power savings (%) for WZE and PCBE for larger number of zones/registers. The power dissipation without any encoding for each benchmark is given in Table 1.

Coding	Work Zone 4 zones		PCBE 4 regs		Work Zone 8 zones		PCBE 8 regs		Work Zone 16 zones		PCBE 16 regs	
Benchmark	mW	sav.	mW	sav,	mW	sav.	mW	sav.	mW	sav.	mW	sav.
VORTEX	2.87	19.15%	1.08	69.53%	3.63	-2.11%	1.34	62.29%	5.36	-50.99%	1.53	56.79%
GCC	3.13	12.82%	1.27	64.48%	3.88	-8.10%	1.45	59.58%	5.65	-57.42%	1.63	54.49%
GZIP	2.69	20.49%	1.06	68.59%	3.50	-3.45%	1.21	64.24%	4.94	-45.96%	1.49	56.03%
PARSER	2.76	19.15%	1.03	69.73%	3.50	-2.56%	1.10	67.76%	5.29	-54.98%	1.22	64.19%
TWOLF	2.77	18.33%	0.99	70.84%	3.54	-4.31%	1.18	65.14%	5.33	-57.15%	1.43	57.86%

the non-encoded bus shown in the table suggests that the reduction in switching power due to larger number of zones/registers may not always be able to offset the increase in power due to larger encoders and decoders. Table 3 shows the power requirement of different encoder/ decoder circuits.

Table 3: A comparison of the Encoder/decoder power (mW) for various encoding schemes.

Design	encoder	decoder	total
SE	0.206	0.267	0.473
TO-C:	0.328	0.316	0.644
WZE: 2 zones	0.512	0.956	1.468
WZE: 4 zones	0.809	1.161	1.970
WZE: 8 zones	1.386	1.356	2.742
WZE: 16 zones	2.366	2.118	4.484
PCBE: 2 reg.	0.108	0.106	0.213
PCBE: 4 reg.	0.137	0.130	0.266
PCBE: 8 reg.	0.268	0.266	0.534
PCBE: 16 reg.	0.397	0.415	0.812

From the simulation results, one can infer that for the assumed bus capacitance of 10pf, the PCBE-2 and PCBE-4 provide the best possible power savings. These methods save as much as 70% of total bus power with as much as 88% reduction in bus switching activity. Larger configurations of the PCBE methods (PCBE-8 and PCBE-16) become effective in cases where the off-chip bus capacitance is greater than 10pf.

Our algorithm uses only one additional bus line as compared to WZE which needs $1 + \lceil log_2 z \rceil$ additional bus lines where z equals number of zones used. Moreover, the hardware complexity of the our encoding algorithm is much less as compared to *WZE* for a given number of registers used to track addresses. Both *PCBE* and *WZE* methods are scalable, i.e., the power savings can be traded with hardware complexity. One can use *PCBE-8* and *PCBE-16* with larger values of on chip bus capacitances when the excess power consumed by the encoder/decoder is offest by switching power reduction in the bus.

It is significant to note that the presence of arithmetic circuitry (adder) in the encoded address path, in itself introduces enormous number of transitions in the *DSE*, *T0-C* and *WZE* methods. One needs to arrest this activity from propagating further. These transitons due to data path delay imbalance can defeat the very purpose of bus encoding. In contrast, PCBE uses equality comparators in the encoder path thereby removing major implementation level challenges and yielding a delay balanced data path.

Finally, for the technology used in these studies, the encoder and decoder hardware complexity for the four schemes, namely T0-C, DSE, WZE (2 registers) and PCBE is (2 registers) was 2110, 2701, 2713 and 1554 gates respectively. At the same time, the delays for the four schemes were 3.34, 18.92, 19.02, 4.13 nsecs. respectively.

4 Conclusions

In this paper, we present a novel approach for power reduction in off-chip instruction address buses. The primary difference between our approach and those of others is that our algorithm does not classify branching addresses as random new addresses; rather, it associates them with program constructs. Its ability to distinguish between addresses generated by regular strides from those generated by branching constructs allows our algorithm to reduce the switching activity on the address bus by a significant amount. It is a scalable algorithm and its complexity can be chosen to take advantage of varying bus capacitances.

References

[1] D. C. Suresh, B. Agrawal, W. Najjar, and J. Yang, "Low power electronics and design," in *Proc. of the 2005 Int. Symp. on SLPED*, pp. 319–322, Aug. 8–10 2005.

[2] Y. Aghaghiri, , and M. Pedram, "Beam: Bus encoding based on instruction-set-aware memories," in *Proc. of Asia South Pacific Design Automation Conference*, (Kitakyushu, Japan), ACM/IEEE, Jan 2003.

[3] H. Kaul, D. Sylveter, M. A. Anders, and R. K. Krishnamurthy, "Design and analysis of spacial encoding circuits for peak power reduction in on-chip buses," *IEEE Trans. Very Large Scale Integration Systems*, vol. 13, pp. 1225–1238, Nov. 2005.

[4] C.-G. Lyuh and T. Kim, "Low power bus encoding with crosstalk delay elimination [SoC]," in *15th Annual IEEE Int. ASIC/SOC Conf.*, pp. 389–393, Sept. 25–28 2002.

[5] P. Subrahmanya, R. Manimegalai, V. Kamakoti, and M. Mutyam, "A bus encoding technique for power and cross-talk minimization," in *Proc. of the 17th Int. Conf. on VLSI Design*, pp. 443–448, 2004.

[6] T. Austin, E. Larson, and D. Ernest, "SimpleScaler: An infrastructure for computer system modeling," *IEEE Computer*, vol. 35, pp. 59–67, Feb 2002.

[7] D. C. Suresh, B. Agrawal, W. Najjar, and J. Yang, "VALUE: variable length value encoder for off-chip data buses," in *IProc. of the 2005 Int. Conf. on Computer Design*, pp. 631–633, Oct. 2–5 2005.

[8] M. R. Stan and W. P. Burleson, "Bus-invert coding for low-power i/o," *IEEE Trans. on VSLI*, vol. 3, pp. 49–58, Mar 1995.

[9] J. Yang and R. Gupta, "Fv encoding for low-power data i/o," in *Proc. of Symp. on Low Power Electronics and Design*, (Huntington Beach, CA), pp. 84–87, ACM, Aug 2001.

[10] M. Mamidipaka, D. Hirschberg, and N. Dutt, "Low power address encoding using self-orgnaizing lists," in *International Symp. on Low Power Design*, (Huntington Beach, CA), pp. 188–193, ACM, Aug 2001.

[11] L. Benini, G. D. Micheli, E. Macii, D. Sciuto, and C. Silvano, "Asymtotic zero-transition activity encoding for address buses in low power microprocessor-based systems," in *IEEE 7th Great Lakes Symp. on VLSI*, (Urbana, IL), pp. 77–82, IEEE, Mar 1997.

[12] W. Fornaciari, M. Polentarutti, D. Sciuto, and C. Silvano, "Power optimization of system level address buses based on software profiling," in *CODES*, pp. 29–33, 2000.

[13] Y. Aghaghiri, F. Fallah, and M. Pedram, "Irredundant address bus encoding for low power," in *Proc. of Symp. on Low Power Electronics and Design*, (Huntington Beach, CA), pp. 182–187, ACM, Aug 2001.

[14] L. Benini, G. D. Michelli, E. Macii, D. Sciuto, and C. Silvano, "Address bus encoding techniques for system level power optimization," in *Design Automation and Test in Europe*, pp. 861–866, 1998.

[15] E. Musoll, T. Lang, and J. Cortadella, "Working-zone encoding for reducing the energy in microprocessor address buses," *IEEE Trans. on VLSI Systems*, vol. 6, pp. 568–572, Dec 1998.

[16] Y. Aghaghiri, F. Fallah, and M. Pedram, "Reducing transitions on memory buses using sector based encoding technique," in *Proc. of Symp. on Low Power Electronics and Design*, (Monterey, CA), pp. 190–195, ACM, Aug 2002.

[17] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, 2nd ed., 1995.