Reference Caching Using Unit Distance Redundant Codes for Activity Reduction on Address Buses

Tony Givargis and David Eppstein
Department of Information and Computer Science
Center for Embedded Computer Systems
University of California, Irvine, CA 92697
{givargis,eppstein}@ics.uci.edu

Abstract

Switching activity on I/O pins of a chip is a measurable contributor to the total energy consumption of the chip. In this work, we present an encoding mechanism that reduces switching activity of external address buses by combining an address reference caching mechanism with Unit Distance Redundant Codes (UDRC). UDRC are codes that guarantee a Hamming distance of at most one between any pair of encoded symbols. Address reference caching exploits the fact that address references are likely to be made up of an interleaved set of sequential address streams. Reference caching isolates these, otherwise interleaved, streams and limits the communication to an UDRC encoded message that identifies the particular reference, at the cost of at most a single bit-transition. Experiments with 14 embedded system applications show an average of 60% reduction in switching activity, with the best and worse cases being 86% and 36% respectively.

Keywords

Bus encoding, embedded systems, low power design

1. Introduction

The energy consumption of electronic devices is becoming an increasingly essential concern when designing embedded systems, especially mobile computing devices [11]. This is because those devices draw their current from batteries that place a limited amount of energy at the system's disposal. Consequently, the lower the average power consumption of those devices, the longer they can operate between two recharge phases. Hence, their mobility is higher and this is a strong argument for preferring such devices to competitive devices.

Off-chip I/O and the associated buses have been shown to be a major contributor to a system's total energy consumption [12]. I/O power consumption is in direct proportion to the product of the switching activity present at the I/O (i.e., pins and attached bus wires) with the average capacitive loads of the switching elements. It has been shown that the capacitive load of off-chip I/O is orders of magnitude larger than that of internal switching nodes (e.g., transistors) [3][4][16], and this trend is likely to continue [11]. Thus, there exists an opportunity for reducing overall energy consumption by encoding/decoding the data prior/subsequent to transmission, at a small added internal energy cost, for a large saving in energy during off-chip transmission.

In this paper we present an encoding and decoding scheme that reduces switching activity of external address buses by combining an address reference caching mechanism with *Unit Distance Redundant Codes* (UDRC) to exploit the otherwise concealed correlation that exists in address streams originated beyond the multilevel on-chip caches.

We introduce a general construction for UDRC, which provide multiple redundant encodings of each possible symbol, in such a way that any arbitrary value can be encoded by a value at Hamming distance at most one from each previous codeword. Our construction uses an optimal number of bits for a given set of symbols.

Address reference caching exploits the fact that address references are likely to be made up of an interleaved set of short sequential address bursts. Reference caching isolates these streams and limits the communication to an UDRC encoded message that identifies the particular reference, at the cost of at most a single bit-transition.

The remainder of this paper is organized as follows. In Section 2, we summarize related previous work. In Section 3, we describe our proposed approach. In Section 4, we describe our experimental setup and show results. In Section 5, we state our conclusion.

2. Previous Work

Numerous approaches for reducing I/O energy consumption have been presented in the past. These approaches fall under two categories. The first category consists of techniques that optimize the memory hierarchy and data organization in order to eliminate the need for I/O in the first place. The second category consists of techniques that reduce the switching activity on buses by exploiting correlations present in streams carried by these buses. Here, we

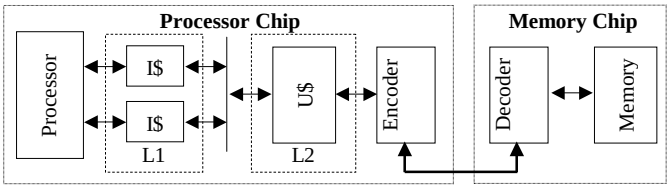

Figure 1: System architecture.

summarize related work in the latter category, as our approach is one of encoding. Furthermore, the former category of approaches can often be combined with suitable encoding approaches for added reduction in overall I/O energy.

Stan and Burleson have introduced a scheme based on *bus-invert* codes to minimize switching activity of communication buses [9]. Their approach computes the Hamming distance between the current value and previously transmitted value and inverts (bit wise negates) the current value if the distance is greater than ½ of the bit-width of the bus. Here, an additional bit (i.e., bus wire) is used to signal the inversion to the receiver. Their approach works well when the stream exhibits randomness, as in data buses. Stan and Burleson have introduced a scheme based on *limited weight* codes, which are a generalization of the bus-invert codes [10]. Here, their approach uses two or more additional wires to achieve further reduction in the average Hamming distance between consecutive pairs of transmitted values.

When the stream on a bus is made up of sequential values (e.g., address buses) *Gray* encoding [14] can be used to reduce the switching activity to exactly one bit-transition per transmitted value. To improve upon this, when the stream on a bus is made up of sequential values, *T0* encoding [2] can be used to reduce the switching activity to exactly zero bit-transition per transmitted value. However, in general, as buses exhibit lesser amounts of sequential behavior (e.g., off-chip address buses in the present of on-chip caches), the overall effectiveness of Gray and T0 encoding fades away.

Musoll et al. have proposed a scheme, called *working zone* encoding, where a very small set of centerline values that are recently observed on the bus are cached on the encoder/decoder ends [8]. Subsequently, if the current value to be transmitted is within a small range of one of the cached values, than, the offset and cache index is transmitted. Their approach exploits the locality of reference that is associated with locality of reference present at the application level, especially those that access multiple arrays. However, in the presence of on-chip caches, especially multi-level caches, address streams tend to be composed of a large number of highly sequential and short (corresponding to a cache line) but scattered bursts, which exhaust the small set of cashed centerlines.

Benini et al. have proposed an encoding scheme, called the *beach solution*, which is application dependent [1]. Here, the address stream of an application is statistically analyzed and consequently a custom encoder and a custom decoder are synthesized that would minimize switching activity when that application is executed. Their approach yields good results at the expense of being application specific and not well suited for systems with dynamic application sets.

Mamidipaka et al. have proposed an adaptive encoding scheme that significantly reduces bit-transition activity on address buses [7]. Their approach does not add redundancy in space (e.g. wires) or time (e.g., cycles). Here, an adaptive technique is used that is based on self-organizing lists to achieve reduction in bit-transition activity by exploring the spatial and temporal locality of the addresses.

For brevity, we have only surveyed a small set of encoding schemes. In our experimental section we will refer back to some of these techniques for further comparison and analysis.

3. Proposed Approach

3.1 Overview

A system level architecture of the proposed technique is depicted in Figure 1. Here, a processor and one or more levels of caches (e.g., instruction/data *L1* cashes connected to a unified *L2* cache) reside on a single chip. In turn, the address bus of the lowest level cache is connected to an off-chip memory via the encoder and decoder. The encoder/decoder transparently send/receive the address values generated by the cache controller with the objective of reducing bit switching activity on the off-chip pins and associated wires. Given our system assumption, we note that caches serve as *filters* that impose certain structure to the address stream as seen externally. Based on experiments and stream analysis we can summarize the following behavior:

1. Repeated access to the same location by an application appears as a single transaction on the bus.
2. The stream is composed of interleaved bursts of consecutive references. Moreover, the

distance between consecutive accesses is that of the processor's machine-word size (typically 4-bytes). The length of these bursts is that of the line size of the lowest level cache.

3. Consecutive references are either exactly one machine-word apart or very far away, but seldom otherwise.
4. At any given time, there exist a working set of these bursts that are interleaved. These burst often are continuation of a recently seen burst.
5. The interleaving behavior is a result of cache lines being written back to make room for new lines, which interrupts the application level sequentially that may exist (e.g., in accessing a large array).

Based on these observations, we propose a reference caching scheme that eliminates switching during short burst, and separates multiple interleaved streams comprising the current working set.

3.2 Reference Caching

Reference caching works as follows. We maintain two small identical N-element *address caches* one each on the encoder and decoder ends. When transmitting a new address value, the encoder compares the new address value to each of the *N* elements in its address cache. More specifically, the encoder adds a constant offset (e.g., the machine-word size of processor) to each cached element prior to the comparison. On a match (i.e., hit), the encoder asserts a special control signal and sends an index, a number in the range of *0 … N−1*, corresponding to the matched address cache location. On a miss, the encoder de-asserts the special control signal, sends the actual address value verbatim, and stores the new address value into its least recently used address cache location.

On the decoder end, when the special control signal is seen asserted, the received index, a number in the range of *0 … N–1*, is used to fetch the corresponding address value from the address cache. This value is then incremented by the same constant offset used in the encoder and passed to the memory controller. If the special control signal is seen de-asserted, the received address value is stored into the address cache at the least recently used location, and passed verbatim to the memory controller.

For the above scheme to work, both the encoder and decoder must use the same algorithm to track the least recently used element. Moreover, the two address caches must reset to arbitrary but identical states (i.e., cache values). To further reduce the switching activity, the transmission of the index, a number in the range of *0 … N−1*, is performed in an encoded fashion. We use UDRC encodings to accomplish this. These codes are further described in the next section.

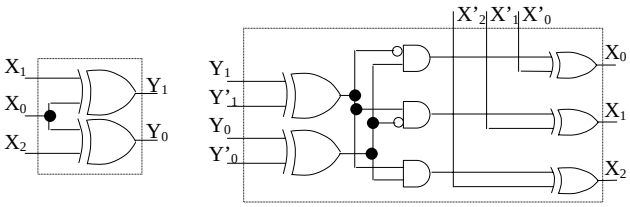

Figure 2: Decoder (left) and encoder (right) circuits.

3.3 Unit Distance Redundant Codes

UDRC provide multiple redundant encodings for each possible symbol, in such a way that any arbitrary value can be encoded by a value at Hamming distance at most one from each previous codeword. For example, consider the 4 symbols 0, 1, 2, and 3 that would normally be encoded in binary as 00, 01, 10, and 11. Here, the Hamming distances between pairs are:

	00	01	10	11
00	0	1	1	2
01	1	0	2	1
10	1	2	0	1
11	2	1	1	0

The total switching is 16 and there are 16 pairs, thus, the average switching is 16/16=1, as expected. Now consider the following redundant codes for the same 4 symbols. We encode the symbol 0 as any of {000, 111}, 1 as any of {100, 011}, 2 as any of {010, 101}, and 3 as any of {001, 110}. Here, the minimum Hamming distance between pairs of codes, from any set representing our symbols, are:

	{000,111}	{100,011}	{010,101}	{001,110}
{000,111}	0	1	1	1
{100,011}	1	0	1	1
{010,101}	1	1	0	1
{001,110}	1	1	1	0

Here, the total switching is 12, thus, the average switching is 12/16=0.75, a reduction of 25%.

Let us now consider the encoder and decoder circuits for the same example. Given a 3-bit UDRC encoding *X*, we can decode it into a 2-bit binary symbol *Y*, as shown in Figure 2. Encoding is slightly more complex. Here, we need to consider the last symbol that was encoded, and encode the new symbol such that to preserve the unit Hamming distance property. Given the most recently encoded binary symbol *Y'* into *X'*, and the binary symbol *Y*, we can compute *X* as shown in Figure 2.

We can show that UDRC encodings exist for any number of symbols. The proof is by construction. If we have 2^k binary symbols (i.e., *k*-bit binary values), we use $(2^k–1)$-bit UDRC encodings. Clearly, when the number of symbols is a power of two, we cannot do any better than that, since each encoding must have

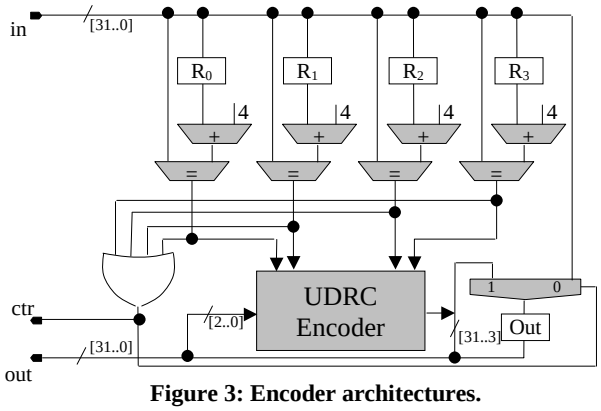

Figure 3: Encoder architectures.

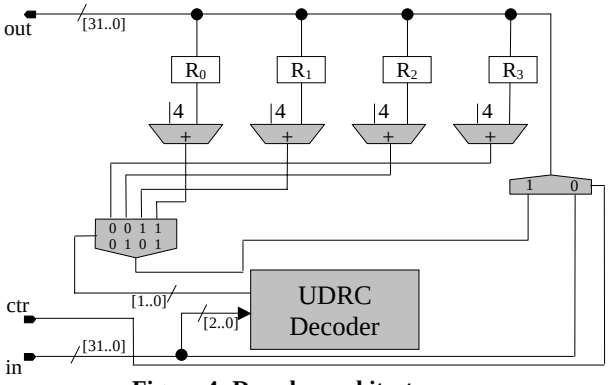

Figure 4: Decoder architectures.

2^k–1 distinct neighbors. If the number of symbols is not a power of two, we round up to the next power of two, and are at most a factor of two away from the optimal number of bits needed to encode a given set of symbols.

Let us first consider the construction of the decoder. Suppose that we want to decode the 7-bit UDRC encoding $X_6X_5X_4X_3X_2X_1X_0$ back to the 3-bit binary symbol $Y_2Y_1Y_0$. We compute over the two-element Galois field GF2[1]:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \end{bmatrix} = \begin{bmatrix} Y_2 \\ Y_1 \\ Y_0 \end{bmatrix}.$$

This produces a 3-bit binary symbol for each 7-bit UDRC encoding. More generally, in the case of k-bit binary symbols, the first matrix would have (2^k–1) columns, k rows, and its elements would be 1, 2 … k–1 in binary down each column. This matrix multiplied by the UDRC (2^k–1)-bit encoding X would yield the k-bit binary symbol Y.

Now we consider the encoder. To get a one-bit change from an UDRC encoding X' representing the binary symbol Y' to another UDRC encoding X representing the new binary symbol Y, we invert the $(Y' \oplus Y)^{th}$ bit in X' if $(Y' \oplus Y) \neq 0$[2].

For example, consider the UDRC encoding X=0001001. The binary symbol is Y=101 as computed over GF2:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}.$$

Now suppose we like to encode a new binary symbol Y=110. We compute 101⊕110=011. Thus, we invert the third bit in X'=0001001 to get X=0001101. The following table gives short stream of values in binary and UDRC, along with associated Hamming distances.

Symbol	Binary	UDRC	Hamming Binary	Hamming UDRC
5	101	0001001	-	-
6	110	0001101	2	1
2	010	0000101	1	1
5	101	1000101	3	1
1	001	1001101	1	1
7	111	1101101	2	1
4	100	1101001	2	1

3.4 Architecture

The hardware architecture for the proposed encoder and decoder is relatively simple and efficient in terms of size and critical-path delay. The block diagram of an encoder with a 4-element address cache is depicted in Figure 3. The corresponding decoder is depicted in Figure 4. The UDRC decoder circuit was previously explained and shown in Figure 2. Likewise, the UDRC encoder was previously explained and shown in Figure 2. The particular UDRC encoder used in the architecture of Figure 3 takes as input a one-hot encoding[3] instead of the binary representation of the previous value. For brevity, we have omitted the

[1] GF2 is a finite field of integers (modulo 2) standing for the *Galois filed* of order 2 [15].

[2] Note that, here, the least significant bit is the first bit, and the most significant is the 7th bit.

[3] One-hot encoding uses a dedicated line for each symbol it encodes to reduced the switching activity to 2 transitions per transmission.

circuit, as it is a simple modification of the encoder shown in Figure 2.

We have omitted the hardware necessary to implement the replacement policy. For this, schemes commonly used in cache design can be adopted [13]. Also, in our design, the address caches are accessed in parallel for added performance. Furthermore, the cache elements are pre-incremented by the offset value eliminating the adders from residing on the critical-path.

4. Experiments

For our experiments, we have used 14 typical embedded system applications that are part of the PowerStone benchmark [6]. Among others, the applications include a JPEG image decoder called *jpeg*, a modem protocol processor called *v42*, a compression engine called *compress*, a CRC checksum algorithm called *crc*, an encryption algorithm called *des*, an engine controller called *engine*, an FIR filter called *fir*, a fax decoder called *g3fax*, a sorting algorithm called *ucbqsort*, and an image rendering algorithm called *blit*.

For bus stream generation, we have used a simulation model [5] of a chip based on the system architecture depicted in Figure 1. The target processor of this simulator is a 32-bit MIPS R3000. The caches are organized into an 8K byte, 2-way, 16-bytes/line instruction cache, a 16K byte, 2-way, 16-bytes/line data cache, and a 32K byte, 2-way, 16-byte/line unified cache. A summary of the address trace sizes and the total number of binary bit transition for each application is shown in the following table.

Application	Stream length	Bit transitions
adpcm	1076	2689
bcnt	300	757
blit	2196	5460
des	1968	4954
compress	7872	17160
crc	444	1165
engine	412	1007
jpeg	157700	283498
fir	520	1267
g3fax	1336	3015
pocsag	884	2098
qurt	304	792
ucbqsort	764	1840
v42	24348	59320

We have implemented models of the proposed encoder and decoder and have simulated the application traces to obtain the total switching activity. Our encoder and decoder have address caches of size 4 and use the least recently used (LRU) replacement policy.

In addition we have implemented bus-invert, Gray, T0, UDRC, and one-hot encoders and decoders for comparison purposes. Note that both UDRC and one-hot encoding for 32-bit buses is prohibitive in practice, as the number of wires needed would be 2^{32}. They are shown for evaluation proposes only.

The switching activity reduction, as a percentage, for a number of encoding approaches is summarized in Figure 5. As shown, on the average, our approach reduced switching activity by 60%, UDRC by 58%, T0 by 34%, one-hot by 17%, and bus-invert by 3%. On the average, and based on published results, the beach solution approach reduced switching activity by 42% while the working zone approach reduced switching activity by 30% [1].

In the case of *blit*, our approach reduced the switching activity the most, namely, 86%. In the case of *engine* our approach reduced the switching activity the least, namely 36%. The best and worse cases are explained as follows. The *engine* example is not a memory intensive application. Instead, it is highly control dominated with many branches and jumps, thus, much of the memory access is dominated by instruction fetches with little access pattern correlations. In contrast, *blit* is dominated by memory accesses that are exploited by our approach.

We have also created synthesizeable RTL models of the encoder and decoder architectures depicted in Figure 3 and Figure 4. We have synthesized these models using Synopsys synthesis tools and measured the area as well as the critical-path delay. The maximum performance penalty (i.e., critical-path delay) for the encoder and decoder is 16 and 14 gates, respectively. The area overhead for the encoder and decoder is equivalent to 2033 and 1858 2-input NAND gates respectively. We have experimented with larger address caches for the encoder and decoder architectures. Our experiments show that the area and delay increase is proportional to the encoder/decoder address cache size (i.e., doubling the size of the address cache approximately doubles the area and critical-path delay.)

5. Conclusion

We have presented an encoding and decoding scheme for address buses to minimize the switching activity at the I/O pins and associated off-chip wires. Our approach caches memory references in order to isolate multiple interleaved sequential streams that make up the majority of data transmitted over an address bus of a system with on-chip caches. Furthermore, UDRC encodings are used to reduce the small amount of switching overhead necessary for reference indexing. Experiments with 14 typical embedded system applications show an average of 60% reduction in switching activity, with the best and worse cases being 86% and 36% respectively. The maximum performance penalty (i.e., critical-path delay) for the encoder and decoder is 16 and 14 gates, respectively.

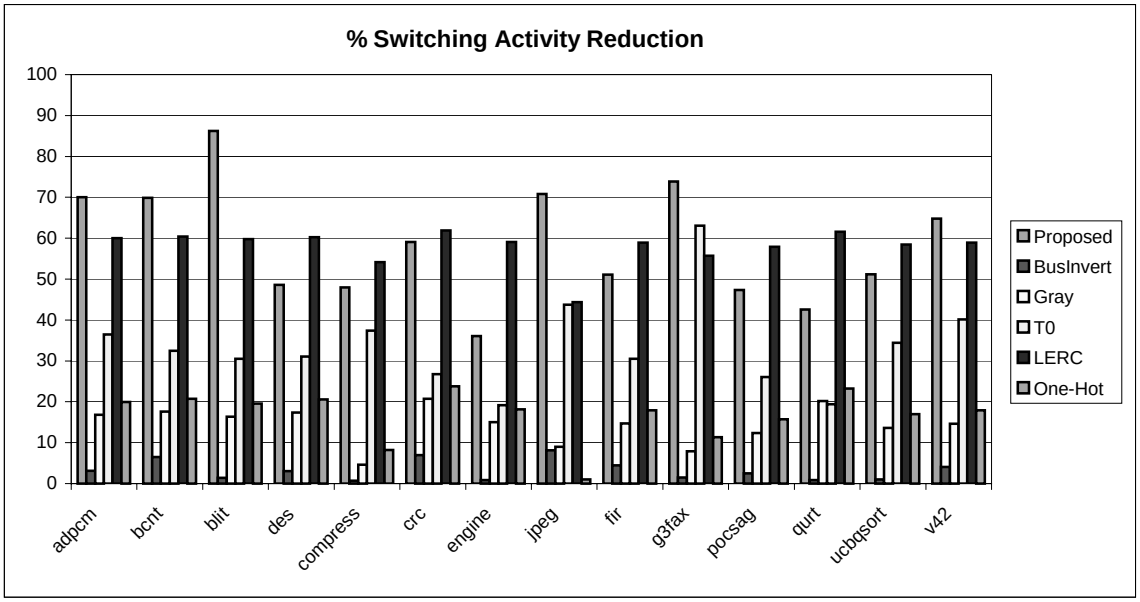

Figure 5: Summary of experimental results.

The area overhead for the encoder and decoder is equivalent to 2033 and 1858 2-input NAND gates respectively.

6. References

[1] L. Benini, G. De Micheli, E. Macii, M. Poncino, S. Quer. Power Optimization of Core-Based Systems by Address Bus Encoding. IEEE Transactions on Very Large Scale Integration Systems, December 1998.

[2] L. Benini, G. De Micheli, E. Macii, D. Sciuto, C. Silvano. Address Bus Encoding Techniques for System-Level Power Optimization. Design Automation and Test in Europe, February 1998.

[3] J.H. Chern et al. Multilevel Metal Capacitance Models for CAD Design Synthesis Systems. IEEE Electron Device Letters, January 1992.

[4] T. Givargis, F. Vahid. Interface Exploration for Reduced Power in Core-Based Systems. International Symposium on System Synthesis, December 1998.

[5] T. Givargis, F. Vahid, J. Henkel. System-Level Exploration for Pareto-Optimal Configurations in Parameterized Systems-on-a-Chip. International Conference on Computer-Aided Design, November 2001.

[6] A. Malik, B. Moyer, D. Cermak. A Lower Power Unified Cache Architecture Providing Power and Performance Flexibility. International Symposium on Low Power Electronics and Design, June 2000.

[7] M. Mamidipaka, D. Hirshberg, N. Dutt. Low Power Address Encoding using Self-Organizing Lists. International Symposium on Low Power Electronics and Design, August 2001.

[8] E. Musoll, T. Lang, J. Cortadella. Exploiting the Locality of Memory References to Reduce the Address Bus Energy. International Symposium on Low Power Electronics and Design, August 1997.

[9] M.R. Stan, W.P. Burleson. Bus-Invert Coding for Low Power I/O. IEEE Transactions on Very Large Scale Integration Systems, March 1995.

[10] M.R. Stan, W.P. Burleson. Limited-Weight Codes for Low Power I/O. International Workshop on Low Power Design, April 1994.

[11] National Technology Roadmap for Semiconductors. Semiconductor Industry Association, 2001.

[12] A. Raghunathan, N.K. Jha, S. Dey. High-level Power Analysis and Optimization. Kluwer Academic Publishers, Norwell, MA, 1998.

[13] J. Smith, J. Goodman. Instruction Cache Replacement Policies and Organizations. IEEE Transactions on Computers, 1985.

[14] C.L. Su, C.Y. Tsui, A.M. Despain. Saving Power in the Control Path of Embedded Processors. IEEE Design and Test of Computers, October 1994.

[15] I.M. Vinogradov. Elements of Number Theory. Dover Publishing, 1954.

[16] N.H.E. Weste, K. Eshraghian. Principles of CMOS VLSI Design. Addison Wesley, 1998.