Available online at www.sciencedirect.com
SCIENCE DIRECT
ELSEVIER
Microprocessors and Microsystems 29 (2005) 145–153
MICROPROCESSORS AND MICROSYSTEMS
www.elsevier.com/locate/micpro

Memory reference caching for activity reduction on address buses

Tony Givargis*, David Eppstein

Department of Computer Science, Center for Embedded Computer Systems, University of California, Irvine, CA 92697-3425, USA

Accepted 5 August 2004
Available online 7 September 2004

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 a memory 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. Memory reference caching exploits the fact that memory references are likely to be made up of an interleaved set of sequential streams. Memory reference caching isolates these, otherwise interleaved, streams and limits the communication to an UDRC encoded message that identifies the particular stream, at the cost of at most a single bit-transition. Experiments with 18 embedded system as well as general applications show an average of 58% 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 a proposed encoder and decoder is 16 and 14 gates, respectively. The area overhead for a proposed encoder and decoder is equivalent to 2033 and 1858 2-input NAND gates, respectively.
Published by Elsevier B.V.

Keywords: Bus encoding; Embedded systems; Low power design; Redundant codes

1. Introduction

The energy consumption of electronic devices is becoming an increasingly essential concern when designing embedded systems, especially mobile computing devices [19] such as cell-phones and personal digital assistants (PDA). This is because such handheld 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 such 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 [20]. 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) [6,8,25], and this trend is likely to continue [19]. 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 article, 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 streams originated beyond the multilevel on-chip caches.

We introduce a general construction for UDRC, which provide multiple redundant encoding 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. The UDRC encoder and decoder will serve as a component in the design of our memory reference caching encoder and decoder.

Address reference caching exploits the fact that address references are likely to be made up of an interleaved set of

* Corresponding author. Tel.: +1 949 824 9357; fax: +1 949 824 8019.
E-mail addresses: givargis@ics.uci.edu (T. Givargis), eppstein@ics.uci.edu (D. Eppstein).

0141-9331/$ - see front matter Published by Elsevier B.V.
doi:10.1016/j.micpro.2004.08.001

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. Isolation of streams is done on each end of the address bus via a small cache that is used to record the tail of N recent references.

The remainder of this article 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 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. In our experimental section, we compare our results with those obtained by applying previous schemes surveyed in this section.

Stan and Burleson have introduced a scheme based on bus-invert codes to minimize switching activity of communication buses [17]. 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 1/2 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 [18]. Here, their approach uses two or more additional wires to achieve further reduction in the average Hamming distance between consecutive pairs of transmitted values. Stan and Burleson have further combined their above findings into a more general framework that allows for activity reduction via spatial redundancy (added wires), temporal redundancy (added cycles), or reduced supply voltage [5].

When the stream on a bus is made up of sequential values (e.g. address buses) Gray encoding [13,23] 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 buses in the present of on-chip caches), the overall effectiveness of Gray and T0 fades away.

A number of new techniques have been developed to improve upon the existing bus-invert, Gray, and T0 encoding schemes. Most of these new techniques (e.g. T0-BI [11], INC-XOR [20], and T0-XOR [7]) combine a number of basic encoding schemes into a single encoder. For example, T0-BI, encodes consecutive memory references using T0 and non-consecutive memory references using bus-invert. Most of these techniques achieve significant savings in switching activity reduction by taking into account the current and previous reference seen thus far. The approach proposed in the work achieves even further reduction in switching activity by taking into consideration a stream of references.

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 [14,15]. 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. In a more recent contribution, Benini et al. have also proposed an optimal technique for synthesizing custom bus encoders/decoders for a given application [3]. Their approaches yield good results at the expense of being application specific and not well suited for dynamic application sets.

Mamidipaka et al. have proposed an adaptive encoding scheme that significantly reduces bit-transition activity on address buses [10]. 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.

3. Approach overview

3.1. System architecture

A system level architecture of the proposed technique is depicted in Fig. 1. Here, a processor and one or more levels

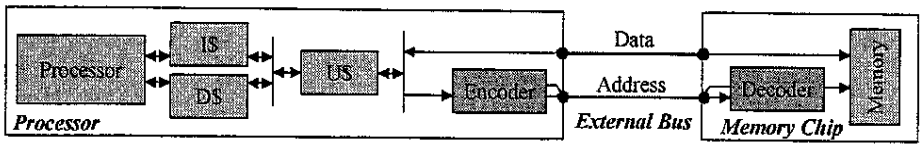

Fig. 1. System architecture.

of caches reside on a single chip. In our target architecture, separate instruction and data $L1$ caches are connected to a unified $L2$ cache. 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 consecutive access to the same memory location by an application appears as a single transaction on the bus. An initial memory access fills the instruction cache with the referenced data and subsequent accesses are carried out between the cache and processor.
2. The address 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 reference caching to eliminate bus activity during short burst, and separate 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 $0 \ldots N-1$, corresponding to the matched address cache location. The special control signal is an additional wire that is used solely amongst the encoder and decoder. 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 $0 \ldots N-1$, 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 $0 \ldots N-1$, is performed in an encoded fashion. We use UDRC encoding to accomplish this. These codes are further described in Section 3.3.

3.3. Unit distance redundant codes

UDRC provide multiple redundant encoding 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 four 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:

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 four 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:

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 Fig. 2 (Table 1). Encoding is slightly more complex. Here, we need to consider the last symbol that was encoded,

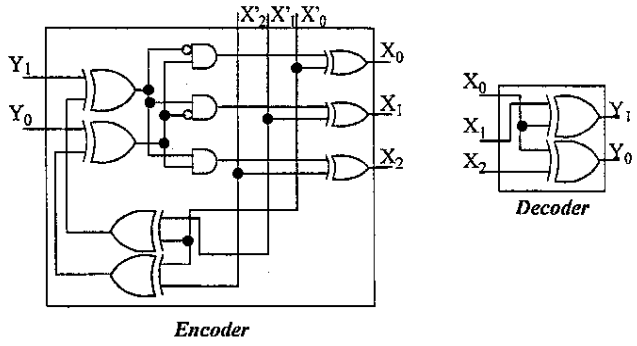

Fig. 2. UDRC encoder and UDRC decoder circuits.

Table 1
Hamming distance between four binary symbols

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

and encode the new symbol such that to preserve the unit Hamming distance property (Table 2). Given the most recently encoded binary symbol Y' into X', and the binary symbol Y, we can compute X as shown in Fig. 2.

We can show that UDRC encoding 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 encoding. Clearly, when the number of symbols is a power of two, we cannot do any better than that, since each encoding must have 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]

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

Table 2
Hamming distance between four UDRC symbols

	{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

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 $(Y' \oplus Y)^{\text{th}}$ bit in X' if $(Y' \oplus Y) \neq 0$[2]. For example, for 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 $101 \oplus 110 = 011$. Thus, we invert the third bit in $X'=0001001$ to get $X=0001101$. Table 3 gives short stream of values in binary and UDRC, along with associated Hamming distances.

3.4. Hardware architecture

The hardware circuit for the proposed reference caching encoder and decoder is relatively simple and efficient in terms of size and critical-path delay. The block diagram of an encoder and the corresponding decoder with a 4-element address caches are depicted in Fig. 3. Note that the UDRC encoder and decoder circuits are used as building blocks in our reference caching encoder and decoder. The UDRC encoder and decoder circuits were previously explained and shown in Fig. 2.

We are omitting the hardware necessary to implement the replacement policy. For this, schemes commonly used in cache design can be adopted [21]. Also, in our design,

[1] GF2 is a finite field of integers (modulo 2) standing for the *Galois filed* of order 2 [24].

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

Table 3
A stream of binary/UDRC encoded symbols and corresponding 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

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.

We also provide an algorithmic specification of the reference caching encoder and decoder that are presented in this work. The encoder algorithm is given first:

Algorithm 1 (encoder)
State: $R_1, R_2, R_3 \ldots R_N$ cache elements
State: X previous UDRC code
Input: A_{in} address input to encoder
Output: A_{out} address output by encoder
Output: *Control* control signal output by encoder

```
for i in 1 to N do
   if (Ri + 4) = Ain then
      Control := 1
      Ri := Ain
      X := UDRC_encode(i, X)
      mask = ~(2N - 1)
      Aout = (Aout & mask) | X
      return
   end if
end for
i := LRU(R1, R2, R3 ... RN)
Ri := Ain
```

The decoder algorithm is given next:

Algorithm 2 (decoder)
State: $R_1, R_2, R_3 \ldots R_N$ cache elements
Input: A_{in} address input to decoder
Input: *Control* control signal input to decoder
Output: A_{out} address output by decoder

```
if Control then
   mask := 2N - 1
   i := UDRC_decode(Ain & mask)
   Ri := Ri + 4
   Aout := Ri
else
   i := LRU(R1, R2, R3 ... RN)
   Ri := Ain
end if
```

4. Experiments

For our experiments, we have used 14 typical embedded system applications that are part of the PowerStone benchmark [12] (Table 4). The applications include a JPEG image decoder called *jpeg*, a modem protocol processor called *v42*, a Unix compression utility called *compress*, a CRC checksum algorithm called *crc*, an encryption algorithm called *des*, an engine controller called *engine*, an FIR filter called *fir*, a group 3 fax decoder called *g3fax*, a sorting algorithm called *ucbqsort*, an image rendering algorithm called *blit*, a POCSAG communication protocol for paging applications called *pocsag*, and a few other embedded applications. In addition, we have experimented with four (*vortex*, *gcc*, *crafty*, *mcf*) very large applications from the integer SPEC CPU 2000 benchmark applications [22].

For bus stream generation, we have used a simulation model [9] of a chip based on the system architecture depicted in Fig. 1. The target processor of this simulator is a 32-bit MIPS R3000. The caches are organized into

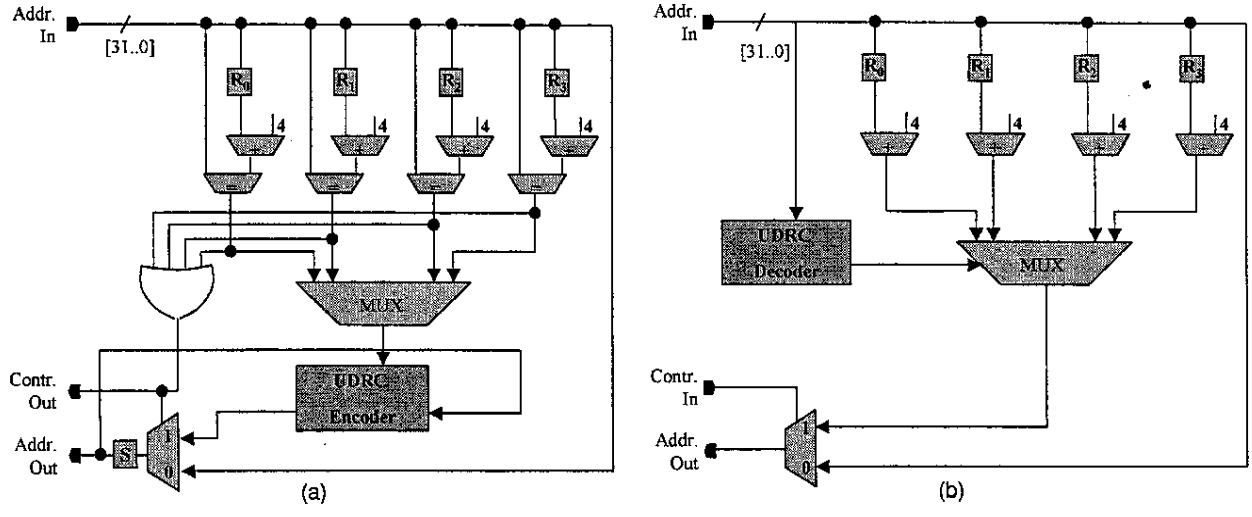

Fig. 3. Reference caching hardware architecture: (a) encoder, (b) decoder.

Table 4
PowerStone/SPEC benchmarks, address stream lengths, and bit transition statistics

Application	Stream length	Bit-transitions/reference						
		Binary (32-bit bus)	Proposed (33-bit bus)	Bus invert (33-bit bus)	Gray (32-bit bus)	T0 (33-bit bus)	T0-BI (32-bit bus)	INC-XOR (32-bit bus)
adpcm	1076	2.499	0.749	2.420	2.078	1.587	1.583	1.484
bcnt	300	2.523	0.760	2.360	2.080	1.703	1.624	1.621
blit	2196	2.486	0.343	2.452	2.079	1.727	1.641	1.522
des	1968	2.517	1.294	2.440	2.079	1.736	1.605	1.574
compress	7872	2.180	1.135	2.164	2.080	1.365	1.236	1.183
crc	444	2.624	1.074	2.441	2.079	1.921	1.855	1.791
engine	412	2.444	1.563	2.422	2.078	1.976	1.958	1.930
jpeg	157,700	1.798	0.525	1.651	1.636	1.011	0.952	0.905
fir	520	2.437	1.192	2.327	2.079	1.692	1.576	1.516
g3fax	1336	2.257	0.590	2.222	2.079	0.833	0.820	0.747
pocsag	884	2.373	1.249	2.314	2.079	1.756	1.700	1.681
qurt	304	2.605	1.497	2.582	2.079	2.099	1.920	1.897
ucbqsort	764	2.408	1.175	2.384	2.080	1.580	1.485	1.432
v42	24,348	2.436	0.858	2.336	2.080	1.458	1.381	1.350
vortex	948,526	2.531	0.758	2.215	2.058	1.332	1.312	1.238
gcc	526,356	2.435	0.860	2.154	2.121	1.652	1.515	1.464
crafty	1,254,856	1.985	1.002	1.856	1.956	1.544	1.528	1.468
mcf	758,652	2.553	1.548	2.512	2.125	1.982	1.974	1.816
Average	73,8251	2.394	1.010	2.292	2.051	1.609	1.537	1.479

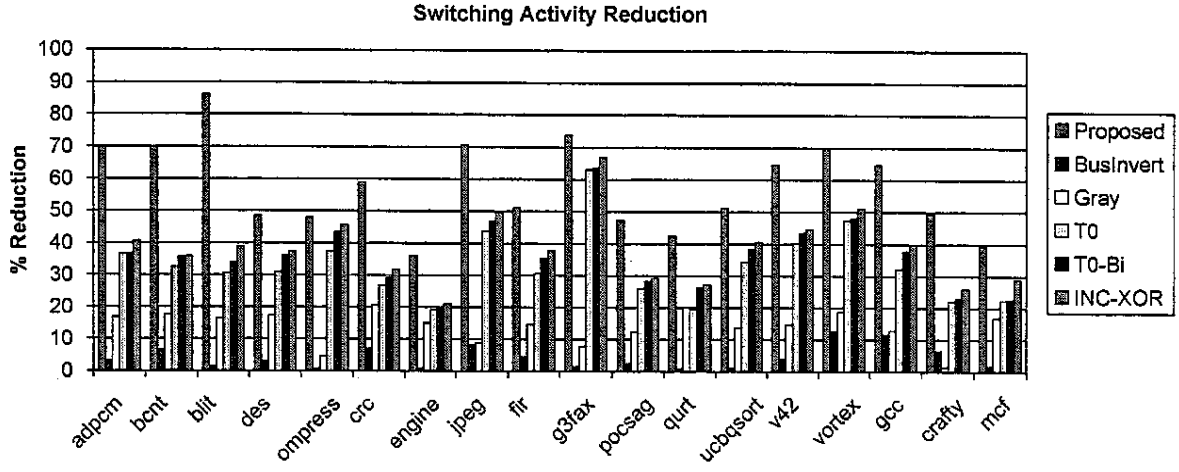

Fig. 4. Switching activity reduction of a number of bus encoding schemes.

an 8K byte, 2-way, 16-bytes/line instruction cache, a 16K byte, 2-way, 16-bytes/line data cache, and a 32K, 2-way, 16-byte/line unified cache. 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, T0-BI, and INC-XOR encoders and decoders for comparison purposes. A summary of the average number of transitions per reference for various encoding schemes is given in the following table.

The switching activity reduction, as a percentage, for a number of encoding approaches is summarized in Fig. 4. As shown, on the average, our approach reduced switching activity by 58%, T0-XOR by 38%, T0-BI by 36%, T0 by 33%, Gray by 14%, and bus-invert by 4%. On the average, and based on published results[3], the beach solution approach reduced switching activity by 42% and 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

[3] We note that these experiments were performed on a different set of benchmarks.

Table 5
Power consumption, improvement, and breakdown

Application	Standard platform power (mW)	Proposed platform power (mW)	Overall power reduction (%)	Standard platform Power distribution (%)				Proposed platform power distribution (%)					
				Bus	Caches	Memory	Processor	Bus	Encoder	Decoder	Caches	Memory	Processor
adpcm	108.8	85.2	21.7	33.8	16.9	21.5	27.7	13.0	1.5	1.0	21.6	27.5	35.4
bcnt	29.3	26.9	8.1	12.7	23.1	25.6	38.6	4.2	0.5	0.3	25.1	27.9	42.0
blit	117.0	111.3	4.9	5.8	21.2	25.9	47.1	0.8	0.1	0.1	22.3	27.2	49.5
des	134.5	129.7	3.6	11.0	23.8	21.2	44.0	5.8	1.1	0.7	24.7	22.0	45.7
compress	481.0	470.0	2.3	7.2	22.1	22.0	48.7	3.8	0.7	0.5	22.7	22.5	49.8
crc	28.6	28.3	1.0	2.1	24.1	25.8	48.0	0.9	0.2	0.1	24.3	26.1	48.5
engine	37.0	36.8	0.6	3.3	22.8	24.8	49.1	2.1	0.4	0.2	22.9	24.9	49.4
jpeg	10,017.1	9143.1	8.7	13.4	16.8	22.8	47.0	4.3	0.5	0.3	18.4	25.0	51.5
fir	31.5	27.9	11.6	27.3	17.2	20.1	35.4	15.1	1.7	1.0	19.5	22.7	40.0
g3fax	107.1	91.4	14.7	22.3	18.3	18.1	41.3	6.8	1.3	0.8	21.5	21.2	48.4
pocsag	67.7	63.2	6.6	17.6	24.5	20.8	37.1	9.9	1.1	0.7	26.2	22.3	39.7
qurt	24.2	23.2	4.1	15.9	16.3	25.2	42.7	9.5	1.7	1.1	16.9	26.3	44.5
ucbqsort	63.8	61.6	3.5	9.2	24.8	25.8	40.2	4.7	0.8	0.5	25.7	26.7	41.6
v42	2002.5	1740.3	13.1	22.5	15.2	21.6	40.7	9.1	1.0	0.7	17.5	24.8	46.9
vortex	103,232.3	94,068.0	8.9	13.8	20.9	23.9	41.3	4.5	0.5	0.3	23.0	26.3	45.3
gcc	46,303.2	43,736.2	5.5	9.5	21.2	21.4	47.9	3.5	0.4	0.2	22.5	22.7	50.7
crafty	685,594.5	654,809.5	4.5	13.5	24.8	19.1	42.6	7.1	1.4	0.9	26.0	20.0	44.6
mcf	59,140.7	57,267.3	3.2	11.0	21.5	18.2	49.3	6.9	0.7	0.5	22.2	18.8	50.9
Average	54,176.6	47,884.4	11.6	15.4	21.9	23.6	44.6	6.2	0.9	0.5	22.4	24.2	45.8

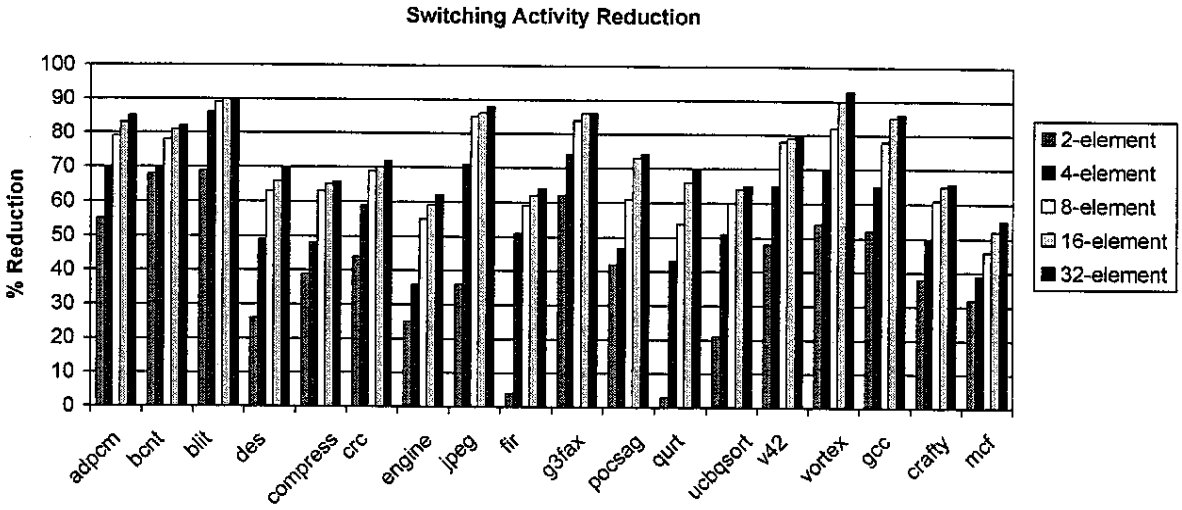

Fig. 5. Switching activity reduction of proposed scheme with different address cache sizes.

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.

In addition to measuring switching activity reduction, we have measured, at system level, the power consumption of a target platform implemented according to the architecture of Fig. 1. We have incorporated processor, cache, memory, and bus models based on CACTI [16] and WATCH [4] into our simulation framework to estimate the power consumption of the entire systems for the encoding/decoding scheme presented in this work and described above. We have modeled the encoder and decoder parts of the architecture at the RTL level and have synthesized the models to a gate-level description using the Synopsys synthesis tools. Then, using the Synopsys gate-level power analysis tools, we have measure the average encode/decode power using the Synopsys gate-level power analyzer. Our results are compiled into Table 5. Here, we have provided the absolute power consumption of the entire platform as well as the percent power consumption breakdown for each component of the platform. For our target platform, and on the average, our proposed technique reduced the power consumption by 12%. This reduction takes into account the added power consumption of the encoder and decoder circuits.

We have also experimented with encoder/decoder architectures of varying address cache size. Our results are shown in Fig. 5. The average switching activity reduction with address cache size 2, 4, 8, and 16 are 40, 58, 69, and 75%, respectively. Note that as the address cache size becomes larger, the switching activity is reduced. However, this reduction is not linear and a diminishing return is observed as the address cache size is increased. Moreover, the reduction in switching activity must be weight against the increase in encoder/decoder complexity and the resulting delay and area overhead. We consider the delay and area overhead next (Fig. 6).

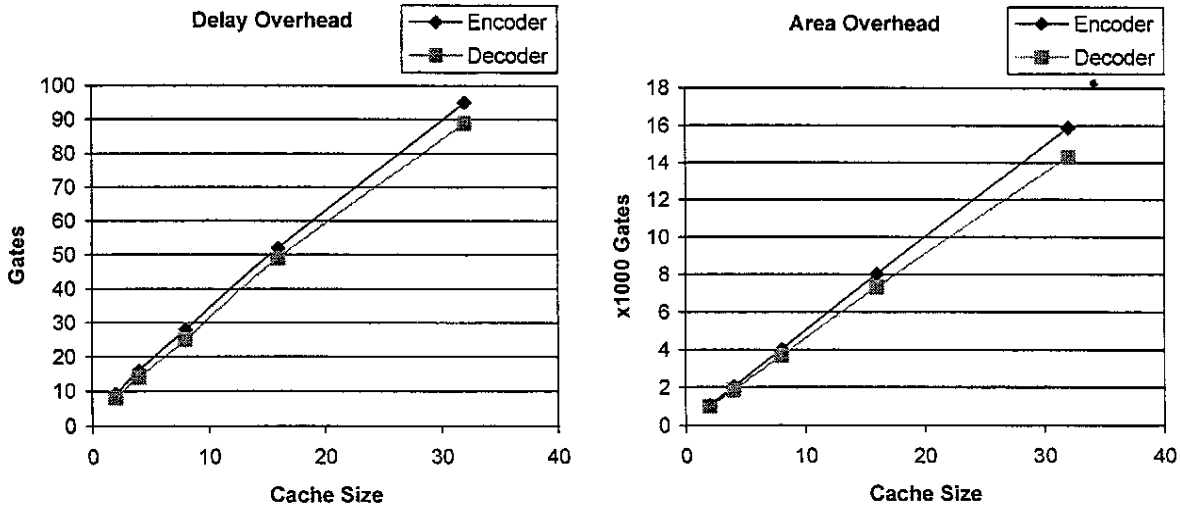

Fig. 6. Combined delay and area overhead of encoder/decoder architecture pairs.

We have also created RTL models of the encoder and decoder architectures depicted in Fig. 3. 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 of Fig. 3 is 16 and 14 gates, respectively. The area overhead for the encoder and decoder of Fig. 3 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.) Error! Reference source not found. gives the delay and area overhead for encoders and decoders of address cache size 2, 4, 8, 16, and 32. In our experiments, encoders and decoders with cache size set to four performed optimally.

5. Conclusions

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 encoding is used to reduce the small amount of switching overhead necessary for reference indexing. Experiments with 14 typical embedded system applications show an average of 58% reduction in switching activity, with the best and worse cases being 86% and 36% respectively, using a 4-element cache encoder and decoder. The maximum performance penalty (i.e. critical-path delay) for the 4-element encoder and decoder is 16 and 14 gates, respectively. The area overhead for the 4-element cache encoder and decoder is equivalent to 2033 and 1858 2-input NAND gates, respectively.

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, 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] L. Benini, A. Macii, E. Macii, M. Poncino, R. Scarsi, Synthesis of low-overhead interfaces for power-efficient communication over wide buses, Design Automation Conference, 1999.

[4] D. Brooks, V. Tiwari, M. Martonosi, A framework for architectural-level power analysis and optimizations, International Symposium on Computer Architecture, 2000.

[5] W.P. Burleson, Low-power encodings for global communication in CMOS VLSI, IEEE Transactions on Very Large Scale Integration Systems 5 (4) (1997) 444–455.

[6] J.H. Chern, et al., Multilevel metal capacitance models for CAD design synthesis systems, IEEE Electron Device Letters January (1992).

[7] W. Fornaciari, M. Polentarutti, D. Sciuto, C. Silvano, Power optimization of system-level address buses based on software profiling, International Workshop on Hardware/Software Codesign, 2000.

[8] T. Givargis, F. Vahid, Interface exploration for reduced power in core-based systems, International Symposium on System Synthesis, 1998.

[9] 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, 2001.

[10] M. Mahesh, D. Hirshberg, N. Dutt, Low power address encoding using self-organizing lists, International Symposium on Low Power Electronics and Design, 2001.

[11] M. Mahesh, D. Hirschberg, N. Dutt, Efficient power reduction techniques for time multiplexed address buses, International Symposium on System Synthesis, 2002.

[12] 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 2000.

[13] H. Mehta, R.M. Owens, M.J. Irwin, Some issues in gray code addressing, IEEE Great Lakes Symposium on VLSI, Ames, IA, 1996, pp. 178–180.

[14] 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, 1997.

[15] E. Musoll, T. Lang, J. Cortadella, Working-zone encoding for reducing the energy in microprocessor address buses, IEEE Transactions on Very Large Scale Integration Systems 6 (4) (1998) 568–572.

[16] G. Reinman, N. Jouppi, CACTI 2.0: An Integrated Cache Timing and Power Model, Compaq Technical Report, Western Research Laboratory, February 2000.

[17] M.R. Stan, W.P. Burleson, Bus-invert coding for low power I/O, IEEE Transactions on Very Large Scale Integration Systems 1995.

[18] M.R. Stan, W.P. Burleson, Limited-weight codes for low power I/O, International Workshop on Low Power Design 1994.

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

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

[21] J. Smith, J. Goodman, Instruction cache replacement policies and organizations, IEEE Transactions on Computers 1985.

[22] SPEC CPU 2000, Standard performance evaluation standards, http://www.spec.org

[23] 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).

[24] I.M. Vinogradov, Elements of Number Theory, Dover, New York, 1954.

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