Exploiting the locality of memory references to reduce the address bus energy*

Enric Musoll† Tomás Lang‡ Jordi Cortadella§

†Dept. of Computer Arch.
Univ. Politècnica de Catalunya
Barcelona, Spain
(now with National Semiconductor Corp.,
Santa Clara, CA 95052)

‡Dept. of Elec. and Comp. Eng.
Univ. of California at Irvine
Irvine, CA 92697

§Dept. of Computer Arch.
Univ. Politècnica de Catalunya
Barcelona, Spain

Abstract

The energy consumption at the I/O pins is a significant part of the overall chip consumption. This paper presents a method for encoding an external address bus which lowers its activity and, thus, decreases the energy. This method relies on the locality of memory references. Since applications favor a few working zones of their address space at each instant, for an address to one of these zones only the offset of this reference with respect to the previous reference to that zone needs to be sent over the bus, along with an identifier of the current working zone. This is combined with a modified one-hot encoding for the offset. An estimate of the area and energy overhead of the encoder/decoder are given; their effect is small. The approach has been applied to two memory-intensive examples, obtaining a bus-activity reduction of about 2/3 in both of them. Comparisons are given with previous methods for bus encoding, showing significant improvement.

I. Introduction

The research on low-energy design has focused on the on-chip components despite the fact that the I/O energy can be as high as 80% of the total energy consumption of the chip [8]. In terms of the on-chip component and the I/O component, the dynamic energy per operation in CMOS is proportional to $C_{int} \cdot A_{int} + C_{I/O} \cdot A_{I/O}$, where the C's are the corresponding capacitances and the A's are the number of transitions (or activities). The internal activity A_{int} is generally much larger than $A_{I/O}$, while the internal capacitance C_{int} is generally much smaller than $C_{I/O}$ (three orders of magnitude: a typical value of $C_{I/O}$ is 50 pf [1], whereas C_{int} is about 50×10^{-3} pf) than $C_{I/O}$. The total energy consumption will decrease by reducing the number of transitions on the high-capacitance, off-chip side, although this may come at the expense of slightly increasing the number of transitions on the low-capacitance, on-chip side. Because the number of internal transitions is already large, increasing it by a comparatively small amount is likely to be insignificant.

In this paper, a method for encoding the address bus which lowers the bus activity and, thus, decreases the I/O energy is presented. This method relies on the locality of memory references. Since applications favor a few working zones of their address space at each instant, for an address to one of these zones only the offset of this reference with respect to the previous reference to that zone needs to be sent over the bus, along with an identifier of the current working zone. This is combined with a modified one-hot encoding for the offset.

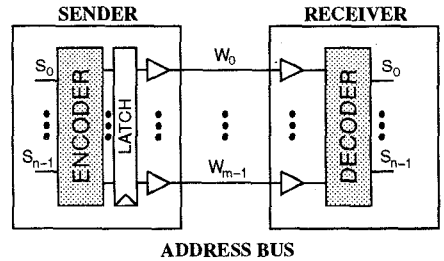


Figure 1: Encoding an n-bit address value into m wires.

The coding technique can be applied to any system in which there is a processing chip and an external memory. These systems can range from application-specific systems to general-purpose processors. We assume that there is no internal cache so that all data memory requests go through the bus. Moreover, the activity of the data bus is not considered in this paper. The scenario with respect to the address bus is the following (see Figure 1): the address value carried by n bits has to be transmitted over the address bus; a reduction in the switching activity of this bus is obtained at the cost of extra hardware in the form of an encoder on the sender device, a decoder on the receiver device, and a larger number of wires m.

Two memory-intensive examples are used to show that the encoding technique in this paper significantly reduces the activity in the address bus and that it outperforms other previously proposals (Gray and bus-invert).

A. Previous work

Three encoding techniques for reduced bus switching activity have been reported: one-hot [3], Gray [12] and bus-invert encoding [11].

In the one-hot encoding technique, the bus is composed of $m = 2^n$ wires. An n-bit value is encoded for transmission by placing a '1' on the i-th wire where $0 \leq i \leq 2^n - 1$ is the binary value corresponding to the bit pattern, and '0' on the remaining $m - 1$ wires. One-hot encoding requires an exponential number (2^n) of wires (which often makes it an impractical choice), but guarantees that precisely one 0-to-1 and one 1-to-0 bit transi-

* This work was partially funded by CICYT grant TIC 95-0419

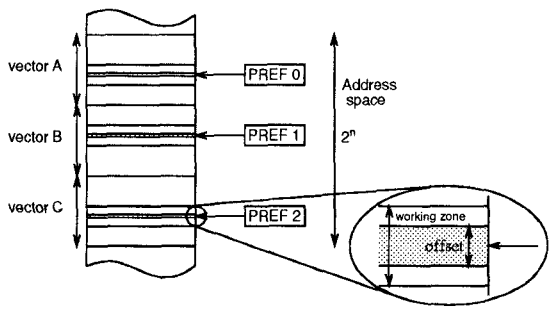


Figure 2: Address space with three vectors.

v value	One-hot encoding	Bit toggles	Modified one-hot encoding	Bit toggles
1	000010	-	000010	-
3	001000	2	001010	1
2	000100	2	001110	1
2	000100	0	001010	1
0	000001	2	001011	1

Value sent over the bus	Modified one-hot retrieval process		Receiver action
	1. XORing	2. One-hot retrieval	
(-) 010011	-	-	-
(1) 011011	001000	3	offset 3 (Pref #1)
(0) 011011	000000	†	same offset (Pref #0)
(0) 011010	000001	0	offset 0 (Pref #0)

Table 1: Example of the modified one-hot encoding and decoding process for $k = 6$.

tions occur when a value (different from the previous one) is sent over the bus.

Gray encoding is useful for sequences of consecutive values (for example, references to instructions) since adjacent values only differ in one bit, whereas for the binary representation adjacent values differ, on average, in two bits. Moreover, the Gray encoding technique requires no extra wires ($m = n$). However, unless the memory references present a high degree of sequentiality, almost no switching activity reduction is obtained when Gray encoding is applied to encode the address bus [12].

The bus-invert method works as follows: if the i-th value to be transmitted over the bus is S_i, then either S_i or $\overline{S_i}$ (i.e. the bit-wise inverse of S_i) is transmitted depending on which would result in a smaller number of bit transitions. Further, to tell the receiver what is being transmitted (S_i or $\overline{S_i}$), an extra wire is used to carry this polarity information. For uniform and independent distributions, this encoding technique works better for smaller values of n; therefore, the n-bit value may be divided into smaller groups and each group encoded independently by associating a polarity bit with each group. The major benefits of the bus-invert method are obtained when encoding buses with high activity.

The reduction of activity in the address bus was also addressed in [9] by mapping the data structures so that the number of transitions is reduced. This technique is complementary to the encoding proposed here, although the mapping might need to be modified to maintain the working zones.

II. Address bus encoding technique

In this section an overview of the proposed encoding technique for the address bus is given.

A. Working-Zone Encoding technique (WZE): Overview

The encoding method for the address bus presented in this paper overcomes the major drawback of the one-hot encoding, namely, the exponential number of wires needed to transmit the address, while maintaining the desirable property of a low switching activity. This is accomplished by taking into account the locality of the memory references [4]: the applications favor a few working zones of their address space at each instant of time. Specifically, given a reference to one of these zones, only the information of the offset of this reference with respect to the previous reference to that zone is sent over the bus, along with an identifier of the current working zone. This information of the offset is sent in *modified one-hot encoded* mode (explained later) using the entire bus bit-width so that very few bits (in some cases, none) vary with respect to the previous value sent, as explained in the next section.

For example, consider an application that works with three vectors (A, B and C) as shown in Figure 2. Memory references will be often interleaved among the three vectors and, because of the spatial locality, usually close to the previous reference to the vector. Thus, if both the sender and the receiver had three registers (henceforth named *Prefs*) holding the previous reference to each working zone, the sender would only need to send:

- the offset of the current memory reference with respect to the Pref associated to the current working zone
- an identifier of the current Pref.

Whenever there is a reference to a working zone not pointed by any Pref, it is not possible to send any offset; in such a case, the entire current memory reference has to be sent over the bus. Thus, an extra wire is needed to indicate that the value being sent over the bus is the current memory reference and not an offset. Moreover, when this occurs, a Pref has to be replaced with the current memory reference using an algorithm discussed later.

Henceforth, this technique is named *WZE* for *W*orking-*Z*one *E*ncoding.

B. Modified one-hot encoding for the offset

To send the offset a variation of one-hot encoding is used. This encodes a value v ($-k/2 \leq v \leq k/2 - 1$) into k bits. If the standard one-hot encoding is used the following activity is obtained:

- If the previous value was also a one-hot encoded offset the activity is two transitions if the offsets are different and zero if the offsets are equal.
- If the previous value was not an offset the activity is of $k/2$ transitions on average.

To reduce further the activity we propose the *modified one-hot encoding*. This **encoding** consists of the following two steps:

1. one-hot encoding of v into k bits
2. XORing the one-hot encoding of v with the previous value sent over the k-bit bus.

Table 1 (first part) illustrates with an example this process for $k = 6$. With this encoding the activity is exactly 1 for both situations considered before.

Moreover, to achieve also zero transitions when the offsets are the same we could resend the previous value sent. However, to increase the frequency of these cases, we take into account the expected regularity in the references to each working zone. Consequently, we resend the previous value sent whenever the current offset is the same as the previous offset for the current working zone.

In summary, the following cases may occur for the encoding of the offset :

Compare the current offset with the previous offset used when referencing the current working zone

- both are the same: send again the previous value sent over the bus

- they are different: send the modified one-hot encoded value of the offset

The **decoding** in the receiver is done also in two steps: XORing the value that it receives with the previous one, and retrieving the one-hot of the result (this process is henceforth named *modified one-hot retrieval* process). When the XORing produces a 0, the two values were the same which is interpreted as a repetition of the previous offset to that same working zone. Table 1 (second part) illustrates the decoding process.

III. Algorithm and implementation issues

This section presents the detailed algorithm of the encoding technique and depicts the hardware implementation. We consider the case in which the whole n-bit word is used for the modified one-hot encoding ($k = n$); the modifications for $k < n$ are straightforward.

A. Encoding/decoding algorithm

Let B be the number of working zones. Then the bus consists of three fields: the n bits of the original address bus (henceforth named `word`), $\log_2(B)$ bits to specify the working zone (henceforth named `indent`) and one bit to indicate whether the value sent is an offset (called `Pref_miss`). Therefore $m = n + \lceil \log_2(B) + 1 \rceil$ extra wires are required (see Figure 1). For clarity, in this work B is considered to be a power of two for clarity; in this case, then, $m = n + \log_2(B) + 1$.

Moreover, the registers used in the encoding algorithm are:

- **prev_ident** contains the value of `ident` of the previous offset
- **prev_sent** is the previous value sent over `word`
- $\mathbf{Pref}_j$ contains the last address to working zone j
- $\mathbf{prev\_off}_j$ is the offset used by the previous reference to working zone j

The current memory address (*current*) is encoded as follows:

1. for $1 \le i \le B$ compute $\Delta_i = current - \mathbf{Pref}_i$
2. if among the Δ_i obtained in 1. there exists a Δ_r such that $-n/2 \le \Delta_r \le n/2 - 1$ then $offset = \Delta_r$ and
   (a) send `Pref_miss` = 0
   (b) send `ident` = r
   (c) if $offset = \mathbf{prev\_off}_r$ then
       send `word` = **prev_sent**
   (d) else
       send `word` = modified-one-hot($offset$)
   (e) load $\mathbf{Pref}_r = current$
       load $\mathbf{prev\_off}_r = offset$
       load **prev_ident** = r

   else
   (f) send `Pref_miss` = 1
   (g) send `ident` = **prev_ident**
   (h) send `word` = $current$
   (i) load $\mathbf{Pref}_j = current$ (value of j depending on replacement algorithm)
       leave $\mathbf{prev\_off}_j$ as before
3. load **prev_sent** = `word`

Note that:

- step 1 can be implemented based on any of the search algorithms used in caches (direct mapped, w-way set associative, fully associative). In this paper, a fully-associative search is considered (as it is shown later with the examples, a very small number of Prefs are needed). In this paper, the offset is defined to be symmetrical with respect to *current* to allow positive as well as negative offsets. However, positive/negative-only offsets may also be defined
- the content of `ident` in the *else* part of step 2 is meaningless; thus, the previous value (**prev_ident**) is sent again so that no useless activity is generated
- step 2(i) can be implemented following any of the replacement algorithms used in caches (pseudo-random, LRU). In this paper, LRU replacement is considered. Also note that $\mathbf{prev\_off}_j$ is not modified since no previous offset is known
- in step 2(h) the bus-inverted value of *current* may be sent over `word` to further reduce the bus activity. However, in the examples we have analyzed, the additional activity reduction is small and it does not pay off the additional increase in wire overhead (one more wire at least)
- this encoding algorithm is transparent to the application being executed, that is, the application does not have to specify the current Pref nor which Pref is replaced. However, the algorithm is easily modified for the case in which special instructions exist to handle the Prefs.

Similarly, the decoding algorithm for the receiver is derived. The registers involved are

- **prev_received** is the previous value received from `word`
- $\mathbf{prev\_off}_j$ and $\mathbf{Pref}_j$ (as in the encoding algorithm)

and the steps to obtain the current memory address are

1. if `Pref_miss` = 0 then
   (a) xor = **prev_received** XOR `word`
   (b) if $xor = 0$ then
       $current = \mathbf{Pref}_{\text{ident}} + \mathbf{prev\_off}_{\text{ident}}$
       leave $\mathbf{prev\_off}_{\text{ident}}$ as before
       else
       $current = \mathbf{Pref}_{\text{ident}}$ + one-hot-retrieve(xor)
       load $\mathbf{prev\_off}_{\text{ident}}$ = one-hot-retrieve(xor)
   (c) load $\mathbf{Pref}_{\text{ident}} = current$

   else
   (d) $current$ = `word`
   (e) load $\mathbf{Pref}_j = current$ (using the same replacement algorithm as sender)
       leave $\mathbf{prev\_off}_j$ as before
2. load **prev_received** = `word`

Activity in address bus

The steps of the algorithm that generate activity in the address bus are (sender side):

- 2(a), if the previous value of `Pref_miss` is '1' (activity: 1 transition)
- 2(b), if the previous Pref used is not r (average activity: $\log_2(B)/2$ transitions)
- 2(c), (activity: 0 transitions)
- 2(d), (activity: 1 transition)
- 2(f), if the previous value of `Pref_miss` is '0' (activity: 1 transition)
- 2(h), (average activity: $n/2$ transitions).

Thus, the activity at the address bus is

$$A = N_{2(a)} + (\log_2(B)/2)N_{2(b)} + 2N_{2(d)} + N_{2(f)} + (n/2)N_{2(h)}$$

where N_{step} is the number of times *step* generates some activity.

B. Implementation issues

Figure 3 shows an illustration of the hardware schematic of the encoder for $n = 16$ and two Prefs. We estimate that the implementation requires about 700 gates and 50 flip-flops. The decoding algorithm takes less hardware and is not shown. The effect of this hardware on the overall chip area is very small. For example, a video-coder chip contains approximately 400K gates. Thus, the area overhead is only 0.25%.

An increase in the number of Prefs increases the complexity of the implementation of the coder and decoder. Although more Prefs would reduce the word activity in the bus, since more working zones can be defined, we expect that for most applications at most four Prefs would be effective. Moreover, the increase in Prefs would increase the number of wires in the bus and it also might increase the activity of the `ident` field.

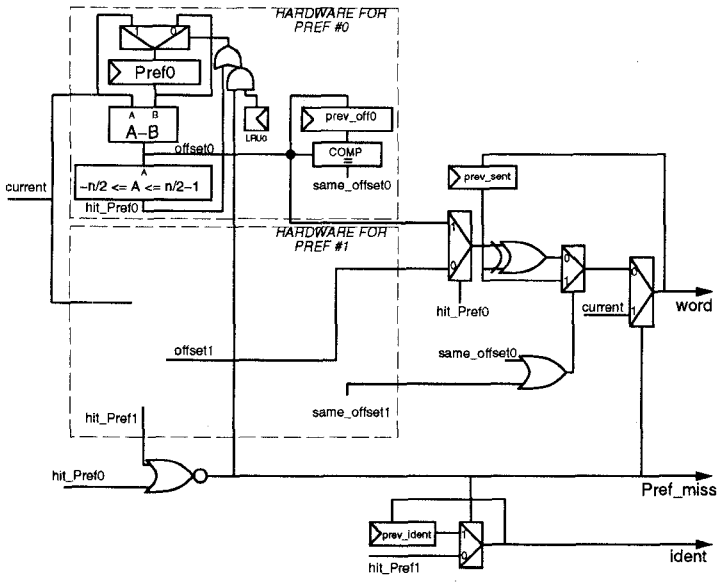
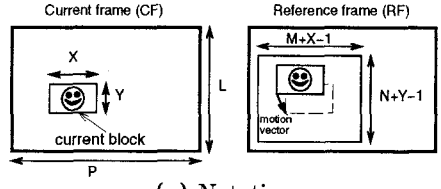


Figure 3: Hardware implementation of the encoder (fully associative, LRU replacement, two Prefs). The logic for Pref #1 is the same as for Pref #0 and it is therefore not shown.



(a) Notation

```
for g = 0 to ⌈P/X⌉ − 1
  for h = 0 to ⌈L/Y⌉ − 1
    optimal = ∞
    for i = −⌊M/2⌋ to ⌊(M−1)/2⌋
      for j = −⌊N/2⌋ to ⌊(N−1)/2⌋
        partial = 0
        for k = 0 to X − 1
          for l = 0 to Y − 1
            NV = CF(X · g + k, Y · h + l)
            RV = RF(X · g + i + k, Y · h + j + l)
            partial = partial + |NV − RV|
        if partial < optimal then
          optimal = partial
          MV(g, h) = [i, j]^T
```

(b) Basic Algorithm

Figure 5: Motion estimation.

C. Energy overhead evaluation

Let us now evaluate the impact that the encoder hardware has on the energy savings. A pessimistic estimation of the energy consumption of the encoder can be done assuming that each gate of the encoder switches with a probability of 0.5 (due to data correlation, the transition probability of the gate outputs of a circuit is less than 0.5 [5]). Since the encoder implementation takes about 700 gates, it presents less than 350 internal transitions/reference. Assuming that $C_{I/O} \approx C_{int} \cdot 10^3$ (see the introduction), the energy overhead of the encoder algorithm is less than 0.35 I/O transitions/reference.

IV. Examples

In this section, two memory-intensive examples (*Motion estimation* and *Quicksort*) are used to show that the encoding technique explained in the previous section significantly reduces the activity in the address bus. The results reported assume that the address bus is only used by data references. However, the encoding technique can be applied when the address bus is shared between data and instruction references. To guarantee the exclusive use of the address bus for data references, the examples use the architecture shown in Figure 4, where the code is stored in a ROM. The data is completely stored in an external memory addressed through a 16-bit address bus. Examples of processors with a 64K address space are DSP16xx, DSP32C and DSP32xx (AT&T Microelectronics), DSP561xx (Motorola), TMS320C2x, TMS320C209, TMS320C5x and TMS320C54x (Texas Instruments), or Z894xx (Zilog). Similar results are expected for an address bus with a larger bit-width.

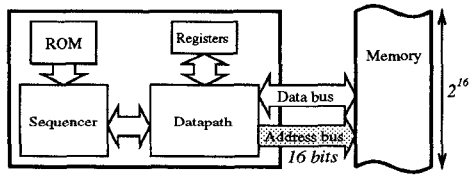


Figure 4: Hardware architecture.

A. Motion estimation

The Motion-estimation algorithm [6] is used in video transmission to lower the bandwidth of the network where the video is being transmitted. As shown in Figure 5(a), the video transmission is composed of frames of $P \times L$ pixels. For the Motion-estimation algorithm, the frames are divided into blocks of $X \times Y$ pixels. Blocks of the current frame (CF) are compared with blocks in the previous (reference) frame (RF) and the best match is selected; in order to limit the number of comparisons, this search is restricted to an appropriate reference window of $(M+X-1) \times (N+Y-1)$ pixels, i.e. each block of the current frame is matched to $M \times N$ blocks in the reference frame. Only the motion vector is transmitted instead of all the pixels of the block. This is illustrated in Figure 5(a). In this paper, the following values are used for the parameters of the algorithm: $P = L = 128$ and $M = N = X = Y = 8$. Moreover, each pixel is 1 byte.

A.1. Basic Algorithm

A straightforward algorithm for the Motion-estimation example (henceforth named *Basic Algorithm*) is shown in Figure 5(b). With the parameters of the algorithm shown before, the Basic Algorithm presents 2.0×10^6 references to memory and 4.8 transitions/reference (without encoding), resulting in 9.8×10^6 transitions in the address bus[1].

Table 2 shows the transitions/reference, the total number of transitions, and the ratio (encoded/non encoded) for:

- the WZE encoding technique explained in Section II with two Prefs
- Gray encoding

[1]These numbers hardly vary whatever the relative non-interleaved memory mapping of the data structures. In this work, CF is mapped at address 0, followed by RF and MV.

Motion estimation: Basic Algorithm			
	Transitions/ reference	Transitions ($\times 10^6$)	Ratio
non encoded	4.8	9.8	1
WZE (two Prefs)	2.5	5.2	0.53
Gray	5.5	11.2	1.14
bus-invert (two groups)	4.5	9.2	0.94

Table 2: Activity reduction for the Motion-estimation example (Basic Algorithm).

- bus-invert encoding with two groups of eight bits each[2].

Note that the number of memory references does not vary. The following conclusions are derived:

- the largest activity reduction (47%) is obtained with the WZE technique with two Prefs. The results with one and more than two Prefs are not shown since, in the first case, there is no significant activity reduction and, in the second case, no significant improvements are obtained with respect to using two Prefs
- no activity reduction is obtained with Gray encoding. Instead, the activity is increased. The reason is that the memory references mainly switch between the current and reference frames, and the distance between two consecutive 8-bit memory references is in several cases a power of two. For example, consider the following two consecutive memory references: 0000100 and 0100100. The latter one is the former plus 32. Without any encoding, one bit toggles. With the Gray encoding technique, the references are 0000110 and 0110110 respectively. Now, two bits toggle
- the bus-invert encoding with two groups of 8 bits each reduces the activity only 6%.

A.2. QR Algorithm

The energy of the address bus is proportional to the number of transitions; thus, the energy can be decreased by reducing the number of memory references and/or reducing the number of transitions/reference.

In the previous section, the number of transitions/reference was reduced by using the WZE technique. In this section, the Basic Algorithm is modified (as done in blocked algorithms [2]) using a set of extra registers to store some frame pixels that are reused in the near future. Those pixels do not have to be fetched again from memory, thus the number of memory references is reduced. However, the energy consumption of the extra registers has to be taken into account.

Care must be taken in not increasing the number of transitions/reference since the total number of transitions could be larger than in the original case. The number of transitions/reference is actually reduced with the following two complementary approaches:

- by referencing more sequentially the memory with a proper use of the extra registers. Note that the average number of transitions in the address bus for a sequence of consecutive references is two (remember that the Basic Algorithm presents 4.82 transitions/reference due to the switching of the memory references between the current and reference frames)
- by applying the WZE technique as it was done in the previous section for the Basic Algorithm.

[2]To fairly compare the WZE and the bus-invert techniques, the latter is evaluated for two groups (implying two extra wires) since the former uses two Prefs (implying two extra wires as well), so that the wire overhead in both techniques coincide.

Motion estimation: QR Algorithm							
		R					
		0	16	32	48	64	
Q	1	0.56	0.49	0.41	0.34	0.26	*references*
		4.75	4.42	3.95	3.25	2.18	*trans/ref*
		16	32	48	64	80	*registers*
	2	0.50	0.42	0.35	0.28	0.20	
		3.13	3.02	2.85	2.59	2.14	
		32	64	96	128	160	
	4	0.47	0.39	0.32	0.24	0.17	
		2.20	2.18	2.17	2.14	2.07	
		64	128	192	256	320	
	8	0.45	0.38	0.30	0.23	0.15	
		1.72	1.75	1.80	1.87	2.00	
		128	256	384	512	640	
	16	0.44	0.37	0.29	0.22	0.15	
		1.51	1.56	1.63	1.75	2.00	
		256	512	760	1024	1280	

Table 3: Motion estimation (QR Algorithm): total references ($\times 10^6$), transitions/reference, and number of registers.

In the QR Algorithm proposed in this paper, Q blocks of the current frame are processed at a time. Since one block of the current frame has to be matched to MN blocks in the reference frame, up to MN registers are needed for each of the Q blocks to store the temporal values of the matching process. We call R the actual number of registers to hold these temporal values for each of the Q blocks ($0 \leq R \leq MN$). When all the R registers hold a valid temporal value, the rest ($MN - R$) of the temporal values have to be stored in memory (in a new data structure that did not appear in the Basic Algorithm). Thus, the total number of extra registers in the QR Algorithm is $Q(2Y + R)$. For a more detailed explanation of the QR Algorithm refer to [7].

The inclusion of the extra registers multiplies by about three the number of instructions of the QR Algorithm with respect to the Basic Algorithm.

Non-encoded

Table 3 shows the number of references and the number of registers for different combinations of R and Q. Note that, for these values, the number of registers is $Q(16 + R)$, and that $1 \leq Q \leq 16$ and $0 \leq R \leq 64$. The number of references has been significantly reduced with respect to the Basic algorithm (for example, for ($Q = 2$,$R = 16$), it goes down from 2×10^6 to 0.42×10^6). Moreover,

- for each value of Q (R), larger reductions in the number of references is obtained for larger values of R (Q)
- for the same number of registers, the minimum number of references is obtained for the case with the largest R
- a larger number of registers does not imply fewer references. For example, with (Q, R)=(4,16), 128 registers are used, but the algorithm presents more memory references than with (Q, R)=(1,64), where only 80 registers are needed. The conclusion is that, to reduce the number of references, it is better to increase R rather than Q.

Working-Zone Encoding

This section considers the activity reductions obtained with the WZE technique with two Prefs (more Prefs do not significantly improve the results) when applied to the QR Algorithm, and a comparison is made to the Gray and bus-invert encoding techniques.

Table 4 presents the activity ratios for each encoding technique given a number of registers (restricted to 512 for practical reasons). These activity ratios are given with respect to the non-encoded case with the same number of registers and fewer number of transitions. Moreover, the WZE column is

Motion estimation: QR Algorithm					
Regs.	(Q, R)	Ratio			
		WZE (two Prefs)		Gray	bus-invert (two groups)
		(A)	(B)		
16	(1,0)	0.32	0.39	0.87	0.76
48	(1,32)	0.30	0.38	0.81	0.79
64	(1,48)	0.25	0.36	0.76	0.82
80	(1,64)	0.18	0.34	0.59	0.90
128	(2,48)	0.24	0.38	0.73	0.84
256	(4,48)	0.24	0.40	0.70	0.89
512	(8,48)	0.23	0.43	0.67	0.93

Table 4: Summary for the Motion-estimation QR Algorithm with encoding: best configuration given a number of registers ($\leq$ 512). Ratio is given with respect to the non-encoded case with the same number of registers and smallest number of transitions. The ratio for the WZE encoding is given without (A) and with (B) the energy overhead of the encoder.

```
Quicksort (vector, l, r){
  if r > l then
    pivot = vector[r]
    i = l - r
    j = r
    do temporal_i = vector[i]; i = i + 1
    while temporal_i < pivot
    do temporal_j = vector[j]; j = j - 1
    while temporal_j > pivot
    if i ≥ j then break
    vector[i] = temporal_j
    vector[j] = temporal_i
  vector[i] = pivot
  vector[r] = temporal_i
  Quicksort (vector, l, i - 1)
  Quicksort (vector, i + 1, r)
}
```

Figure 6: Quicksort.

split into two subcolumns to show the effect of the energy overhead derived in Section III.C. Note that there is an optimal number of registers ($Q = 1$ and R=64). The conclusion is that the WZE clearly outperforms the Gray and bus-invert encoding techniques for this example.

B. Quicksort

The Quicksort [10] is a divide-and-conquer algorithm that sorts a vector of elements. In this paper, the vector has 64K elements of 1 byte each, thus occupying the entire address space. Figure 6 shows the algorithm implemented in this paper (the first call is *Quicksort (vector,0,64K-1)*).

The total number of memory references in this example is data dependent (it is then less predictive than in the Motion-estimation example); the number of references is different when sorting a vector which is already sorted than when sorting a vector initialized with pseudo-random values. Henceforth, this last case is considered in this paper. The algorithm in Figure 6(a) for a pseudo-randomly initialized vector presents 1.9×10^6 memory references and 4.1 transitions/reference, resulting in 7.6×10^6 total transitions in the address bus.

Table 5 shows the transition/reference, the total number of transitions and the ratio (encoded/non encoded) for the same encoding techniques in Table 2. The following conclusions are derived:

- the largest activity reduction (59%, including the energy overhead of the encoder derived in Section III.C) is obtained with the WZE technique using two Prefs. The results with one and more than two Prefs are not shown for the same reason stated for the Motion-estimation example
- both the Gray and bus-invert encodings reduce the activity 15%.

Quicksort			
	Transitions/ reference	Transitions ($\times 10^6$)	Ratio
non encoded	4.1	7.6	1
WZE (two Prefs)	1.4	2.5	0.33 (0.41)
Gray	3.4	6.4	0.85
bus-invert (two groups)	3.5	6.5	0.85

Table 5: Activity reduction for the Quicksort example. The ratio in parenthesis for the WZE technique takes into account the energy overhead of the encoder.

V. Conclusions

This paper presents an encoding technique for low-energy address buses that significantly outperforms other encoding techniques (namely, Gray and bus-invert) as shown in two memory-intensive examples, and that presents a small area and energy overhead. Energy reductions of about 2/3 have been obtained for the examples. This technique may be applied to other DSP applications such as the FFT, digital filters, and, in general, to any memory-intensive application.

Moreover, it has been shown that reductions of two orders of magnitude can be obtained by properly using registers to hold data values that are referenced in the near future. This reduction is in addition to the one obtained with the encoding technique presented in this paper.

References

[1] A. Bellaouar and M.I. Elmasry. *Low-power digital VLSI design: circuits and systems.* Kluwer Academic Publishers, 1995.

[2] S. Carr and K. Kennedy. Compiler blockability of numerical algorithms. In *Proc. of the Supercomputing'92 Conference*, pages 114–124, 1992.

[3] A.P. Chandrakasan and R.W. Brodersen. *Low Power Digital CMOS Design.* Kluwer Academic Publishers, 1995.

[4] J.L. Hennessy and D.A. Patterson. *Computer Architecture: a quantitative approach.* Morgan Kaufmann Publishers, 2nd edition edition, 1995.

[5] P.E. Landman and J.M. Rabaey. Activity-sensitive architectural power analysis. *IEEE Trans. on CAD*, 15(6):571–587, June 1996.

[6] C. Lin and S. Kwatra. An adaptive algorithm for motion compensated colour image coding. *IEEE Globecom*, 1984.

[7] E. Musoll, T. Lang, and J. Cortadella. Exploiting the locality of memory references to reduce the address bus energy. Technical Report UPC-DAC-1996-60, ftp://ftp.ac.upc.es/pub/reports/DAC/1996/UPC-DAC-1996-60.ps.Z, Dept. of Computer Architecture, Polytechnic Univ. of Catalonia, December 1996.

[8] C.A. Neugebauer and R.O. Carlson. Comparison of wafer scale integration with VLSI packaging approaches. *IEEE Trans. on Components, Hybrids, and Manufacturing Technology*, pages 184–189, June 1987.

[9] P.R. Panda and N.D. Dutt. Reducing address bus transitions for low power memory mapping. In *Proc. EDAC*, March 1996.

[10] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, second edition, 1992.

[11] M.R. Stan and W.P. Burleson. Bus-invert coding for low power I/O. *IEEE Trans. on VLSI Syst.*, pages 49–58, 1995.

[12] C-L. Su, C-Y. Tsui, and A.M. Despain. Saving power in the control path of embedded processors. *IEEE Design & Test of Comp.*, pages 24–30, winter 1994.