# A Fast INC-XOR Codec for Low Power Address Buses

H. Parandeh-Afshar[1,*], M. Saneei[1], A. Afzali-Kusha[1], M. Pedram[2]

[1]Nanoelectronics Center of Excellence, School of Electrical and Computer Engineering
University of Tehran, Tehran, Iran

[2]University of Southern California, Department of Electrical Engineering – Systems
Los Angeles, CA
[*]Currently on research leave in EPFL University, Lausanne, Switzerland

hparande@ut.ac.ir, mohsen.saneei@ece.ut.ac.ir, afzali@ut.ac.ir, pedram@usc.edu

## Abstract

This paper presents a very fast and low-power address bus encoder, whose critical path delay and area are only weakly dependent on the address bus width. Although the encoding algorithm of the proposed structure is the same as the INC-XOR encoding, its encoder and decoder architectures, called DX, are much faster. The DX architecture implements the INC-XOR encoding partially (partial DX architecture) or fully (registered DX architecture.) The partial implementation, which is faster and consumes less power and silicon area, is appropriate for cases where the size of the basic block (sequential addresses without branches or jumps) is bounded, e.g., by 256. The registered DX architecture uses a multi-stage pipelined structure with pseudo-incrementers to reduce the combinational delay of each pipeline stage. The two DX implementations (partial and registered) are compared with three conventional INC-XOR architectures realized by using the ripple carry, the carry look-ahead, and Sklansky prefix incrementers. The

results for the critical path delay, gate count, power-delay product, and energy-delay product show considerable improvements over the conventional implementations.

***Keywords:*** Low-Power Incrementer, Low-Power Address Bus Encoder/Decoder, Minimizing Switching Activity.

## I. Introduction

A computer system typically consists of a CPU, a memory controller, memory chips, and buses dedicated to providing the means for data transfer between the CPU and the memory. These buses form the performance bottleneck in many computer systems. At the same time, due to the high capacitance of these off-chip buses, the energy dissipation per memory bus access is quite high. These factors have served as the driving force for the research on design of low-power buses (for example, see [1]-[8].)

To lower the power dissipation of memory buses, one may utilize characteristics of the addresses encountered during the memory access. In a computer system, the generated addresses on an address bus are often in sequence and two sequential addresses have a difference of one unit (also called stride value.) For example in the T0 method [3][4], instead of placing sequential values on the bus, the address at the source (which is called the “source word”) is examined and if its value is exactly one unit greater than the previous address, then the address bus will be frozen. At the destination, if the address that is read from the bus (which is called the “code word”) is unchanged compared to the previous code word, then the decoded value is obtained by incrementing the previous code word by one.

One of the most efficient methods used for address bus encoding is the INC-XOR method [5] (called T0-XOR in [6].) This technique makes use of the decorrelating

characteristic of the XOR function (details are provided in the next section.) Both encoder and decoder modules of the INC-XOR codec utilize an adder whose width is equal to that of the bus. As the bus width increases, the delay, logic complexity, and power consumption of this adder must also increase proportional to the bus width. This increase can lead to the deterioration of the delay, power, and area parameters of the overall structure. T0-Concise or T0-C for short is yet another variant of T0 which eliminates the redundant bit [7].

In this paper, a method is introduced to improve the properties of the INC-XOR encoder/decoder modules. With the proposed method, the delay, power-delay product, and energy-delay product of the encoder/decoder improves while its power dissipation and area become less dependent on the bus width. The remainder of this paper is organized as follows. In Section II, we briefly describe INC-XOR memory bus encoding technique, while in Section III the proposed method (DX) is presented. Section IV shows a comparison between INC-XOR and DX and the synthesis results of DX method. The summary and concluding remarks are given in Section V.

## II. INC-XOR Encoding Algorithm

The INC-XOR code of [5][6], may be described as follows:

$$\text{Encoder:} \quad B^{(t)} = b^{(t)} \oplus (b^{(t-1)} + 1) \oplus B^{(t-1)}$$

$$\text{Decoder:} \quad b^{(t)} = B^{(t)} \oplus B^{(t-1)} \oplus (b^{(t-1)} + 1)$$

where $b^{(t)}$ and $B^{(t)}$ denote the source word and the code word at time $t$, respectively. At a given clock cycle, $t$, the encoder computes the incremented address of cycle $t - 1$ and compares it to the address generated at cycle $t$. If the incremented old address $(b^{(t-1)} + 1)$

and the new address ($b^{(t)}$) are equal, then the result of the left XOR in the encoder equation will be zero and the old code word ($B^{(t-1)}$) will be left on the bus. It can be easily seen that when the addresses are consecutive, no switching activity occurs (similar to the case of T0 code.) In general, the encoder/decoder structures of the INC-XOR codec are simpler than those of the T0-C. That is why we will focus on the INC-XOR codec from here on.

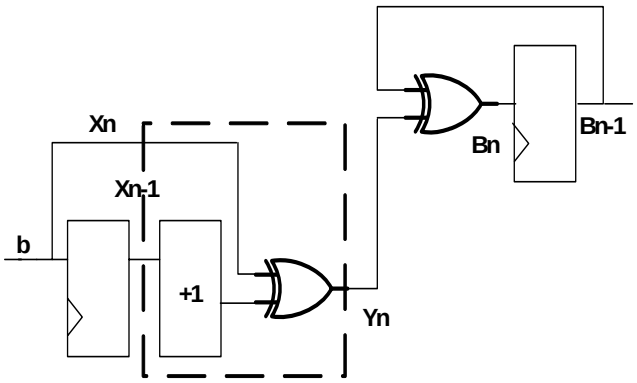



**Figure 1.** INC-XOR encoder architecture [5].

The encoder architecture for INC-XOR is shown in Figure 1 where $X_n$ is the current source word, $X_{n-1}$ is the previous source word, and $Y_n$ is calculated as the XOR of $X_n$ and $X_{n-1}+1$ by the left XOR gate. The right XOR gate performs transition signaling of $Y_n$ with respect to the previous code word on the bus. In this figure, all the lines have a width equal to *W* and, hence, *W* parallel XOR gates are shown as a simple boldface XOR gate. The encoder inserts one cycle delay between the arrival of the address *b* and the output of the encoded bus *B*. The critical path delay of the encoder is the delay of combinational logic between two registers (incrementer and XOR gates.) This delay can be critical in delay-constrained systems with wide buses. Additionally, note that glitches on *B* must be avoided because *B* is connected to large output buffers that should always be driven by clean and fast edges to eliminate excessive power dissipation and signal quality

deterioration [5]. Due to the fact that the address *b* is generated by a complex logic that produces glitches and misaligned transitions, the second register is used to filter out glitches and align the transitions on *B* to the clock edge (The register is needed even if the binary code, *i.e.*, no encoding, is used.) The decoder architecture is similar to the encoder architecture [5]. In the next section, we introduce a new method to improve the delay and the power delay and energy delay products of the INC-XOR encoder/decoder.

## III. The DX Architecture

As explained in the previous section, the delay and power consumption overhead of the incrementers in the INC-XOR encoder/decoder of the wide address buses may be unacceptable. This is especially important when the encoder is used in high performance VLSI designs where the decoder could be on the critical signal path whose delay would be of critical importance. To remedy the problem, the adder and the following XORs should be replaced by a simpler and faster architecture. Toward this end, we use the structure of Figure 2 as the internal logic of a 4-bit incrementer (ripple carry adder structure) with its successive XORs for proposing a faster and smaller pseudo-incrementer. The half adders, denoted by HA, are used for constructing the incrementers where for each bit slice, the longest path (shown by arrow) is the path that goes through two XOR gates. Hence, the delay of each bit of $Y_n$ is the delay of its right input plus the delay of two XOR gates. The delay of the right input is equal to the delay of the chain of its previous two-input AND (AND2) gates. As the size of the chain grows, this delay increases making the delay of the most significant bit the largest one. To reduce the delay of this structure, first note that $Y_n$ is generated by a 3 input XOR, which has been implemented by a two cascaded 2 input XOR gates. If we change the inputs of the two

XOR gates as shown in Figure 3, we can eliminate one of the XOR gates from the critical paths. We utilize the structure of Figure 3 in the INC-XOR encoder/decoder and call them the *DX encoder/decoder*.

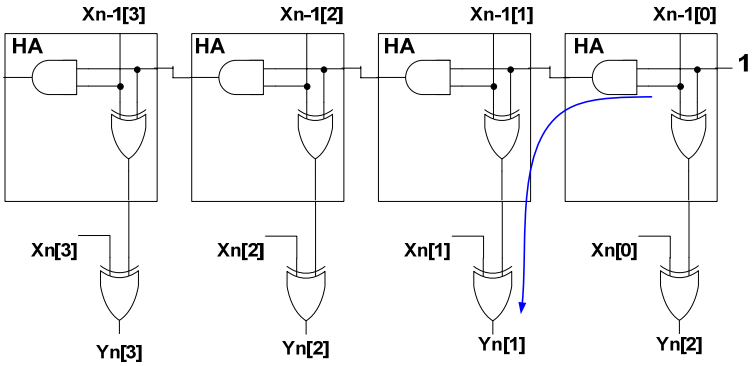



**Figure 2**. 4-bit incrementer with successive XORs.

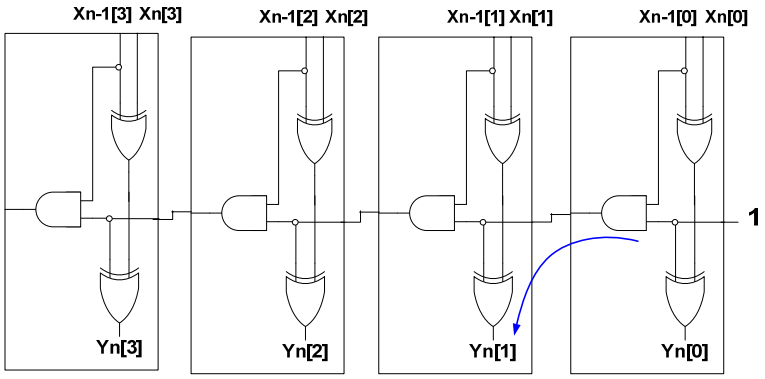



**Figure 3**. Modified version of the circuit shown in Figure 2**.**

Although one-level of XOR gates has been eliminated from the $Y_n$ generation path, the chain of AND gates still exists. This chain is the carry chain of the adder. The delay of this chain grows with the increase of the bus width, making this architecture unusable for high performance systems. To solve the problem, we should use a faster incrementer. In this work, we investigate three incrementer architectures which include ripple-carry structure, binary-tree Carry Look Ahead structure [10], and Sklansky Prefix-adders [11][12]. When comparing the incrementers, the performance level is greatly dependent

on the technology used for the implementation. To be able to evaluate each architecture independent of the target implementation, we adopted the model proposed in [13] which relates the required circuit area to the propagation time of each logic cell [11]. In this approach, each two-input monotonic gate (e.g., AND and NAND gates) is counted as one gate while an XOR gate is counted as two gates in terms of the area and the delay. In [10], the above structures have been compared by assuming that the width of the desired adder is '*W*'. The results of this comparison are given in Table I which shows the ripple carry incrementer has the minimum area while the Sklansky-Prefix incrementer has the minimum delay. Since for the encoder/decoder, we do not need a general purpose incrementer, in the following, we present two techniques for increasing the speed without sacrificing the power consumption.

**Table I.** Conventional Incrementer Features [10]. *W* is the incrementer width.

| Incrementer | Circuit Area | Circuit Delay |
|---|---|---|
| Ripple Carry | $3.W$ | $W+1$ |
| Carry Look-Ahead | $6.W$ | $4.\log_2 W$ |
| Sklansky Prefix | $1/2.W.\log_2 W+2W$ | $\log_2 W + 2$ |

### *A. Partial DX Architecture*

The first and simple solution to the above problem is to use the DX encoder described earlier for a small set of the least significant bits (LSBs). This idea is motivated by the fact that in practice, nearly one out of every seven consecutive instructions is a control transfer instruction [16]. Furthermore, an experimental study on several benchmarks presented in [16] reveals that, the address changes in some 90% of all jumps affect eight or fewer LSBs of the address bus. Finally, our own experiments on the SPEC2000 benchmark programs [14] show that in most cases, a series of sequential addresses has

fewer than 256 addresses, which results in changes of only the eight least significant bits (LSBs). This means that instead of applying the INC-XOR method to all the bits, we can use the DX encoder for the 8 least significant bits, and utilize simple XOR gates for the other most significant bits. This gives rise to very few transitions when the sequential changes of the address bus value correspond to the changes of bits other than the least 8 significant bits or when there is a jump out of this boundary. The additional transitions associated with the former case may be solved using techniques such as the Offset-XOR code [6]-[8].

The partial DX-architecture enables us to obtain almost the same power saving as applying the INC-XOR method to all bits. The method is depicted in Figure 4 which shows that the encoder simply sends the original 24 bits, which tend to toggle very infrequently, while the 8 LSB bits are coded by an 8-bit DX encoder. Since the DX coding is applied to 8 LSB bits, in the worst case, it could have an AND gate with 7 inputs (instead of chaining the AND2 gates) to reduce the delay. In the proposed architecture, when the address changes exceed 256, the outputs of the XOR gates show the bits that have changed to the other side for correcting the address. Finally, one should note that the number of the LSBs which are coded by the DX encoder depends on the application and its input data characteristics. There may exist applications where this number is larger than 8. In these cases where a higher number of LSBs than 8 should be invoked (e.g., because of many jumps with address changes requiring larger than 8 LSBs of the address bus), the use of partial DX architecture may lead to higher power consumption, and thus, the registered DX architecture, which is described next, should be used.

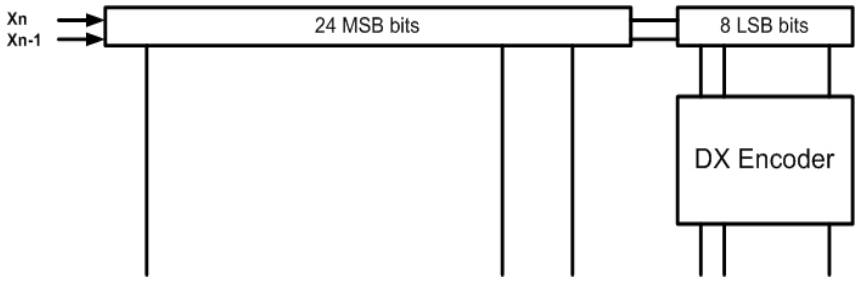



**Figure 4**. Partial DX Architecture.

### *B. Registered DX Architecture*

As discussed previously, the bottleneck in the DX encoder is the chain delay of the AND gates. To alleviate the problem, we use the fact that during the incrementation process, the carry will reach a specific bit position when all previous bits of the number (before it is incremented) are 1. This feature allows us to propose a pseudo-incrementer which is based on the ripple carry incrementer structure. In this architecture, registers are inserted in the carry path for reducing the propagation delay. This is different from a pipelined adder since no latency has been added to the circuit. In the proposed architecture, starting from the LSB, each group of $M$ consecutive bits of the input are placed into a *slice*. The input carry of 1 of the slice $i$ is propagated to the output carry when all of the $X_{n-1}$ input bits to slice $i$, ranging from bit position $M \times i$ to bit position $M \times (i + 1) – 1$, are 1.

When incrementing a number by one, the higher significance bits tend to change less frequently. This property allows us to use the registered values of the bits instead of the original values for deciding to propagate the input carry, cutting the long carry path to several shorter ones. Figure 5 shows the registered DX architecture. In this figure each elongated box with inputs $X_n[.]$ and $X_{n-1}[.]$ and outputs CO and $Y_n[.]$ represents a set of two XOR and one AND gates depicted in Figure 3. Clearly, each register stores the result

of ANDing of *M* bits of the previous address, $X_{n-1}$. Inserting registers in the AND gate chain produces a new chain of AND gates which is much shorter than the previous one. This reduces the combinational delay of each stage while keeping the total latency unchanged. In fact, we will present a pseudo-incrementer which is based on ripple-carry incrementer and may be effectively utilized in the INC-XOR encoder/decoder structure to reduce the combinational delay of each pipeline stage.

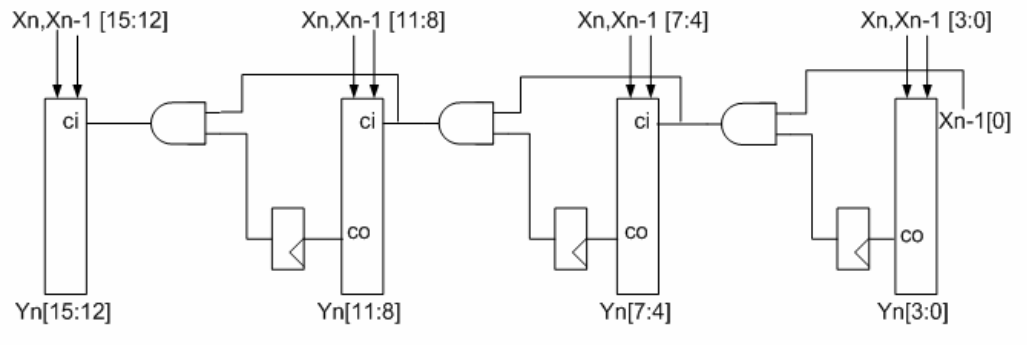
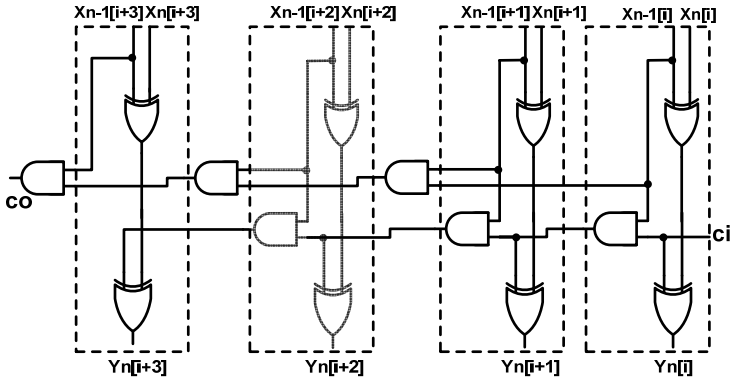



**Figure 5**. Modified version of DX architecture (called registered DX.)

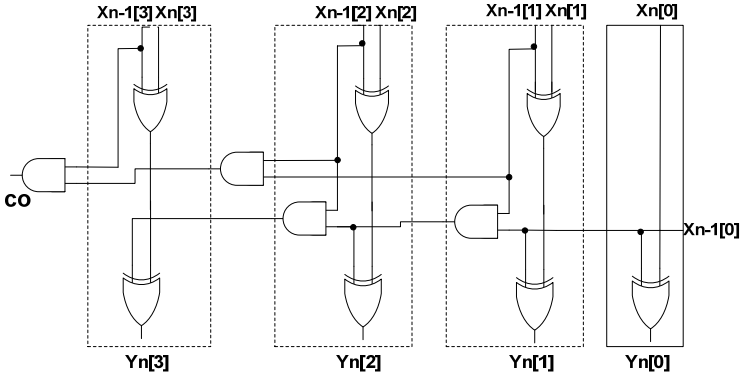
The new chain consists of two separate chains: before and after the register. The first part of this chain is inside the DX slices and performs AND of $X_{n-1}$ bits inside that slice to form the 'co' output. This is depicted in Figure 6, where a basic slice of DX encoder for $M = 4$ is shown. Compared to the circuit of Figure 3, a new AND gate chain has been added. This output is registered so that it can be used in the next clock cycle. The second part of the new chain is out of the DX blocks, where the 'ci' inputs of the DX blocks are constructed. Note that the 'ci' input of the last DX block has the longest propagation delay. Except for the first clock, the 'ci' input may be considered as the input carry bit of the block. If the input is 1, some of the addresses associated with that block may change while if it is 0, no address bit for that block will change for cycle *n*.

Note that the ‘co’ output of each block is independent of the ‘ci’ input and only depends on ‘$X_{n-1}$’, and, therefore, the generations of ‘co’ outputs of all *M*-bit blocks are performed in parallel. Finally, the circuit for the first slice is slightly different from the circuit shown in Figure 6(a). As shown in Figure 6(b), the ‘ci’ input of the first block is connected to $X_{n-1}[0]$ and the ‘co’ output is independent of $X_{n-1}[0]$. It is imperative to mention that for the cases where the inputs are not consecutive, the proposed pseudo incrementer may generate a value different from the incremented value of the new input. This is the reason for calling the structure ‘*pseudo* incrementer’. In this case, the result is corrected for the next consecutive address. Because the decoder uses the same structure, no address conflicts occurs and the original address is recovered in the decoder side.

The basic blocks of Figure 6 can be cascaded for encoding/decoding of wider buses. In Figure 5, a 16-bit modified DX encoder has been developed with 4-bit basic blocks where the ‘co’ output of each 4-bit block is registered. By insertion of this register, the original functionality of encoder will be kept, while the critical path of the encoder is reduced considerably. Using this architecture and choosing a proper width for each basic block, the delay of the encoder/decoder becomes as small as the delay of each basic block. The basic block width varies as a function of the bus width. A proper width selection for the basic block is essential since the delay of the encoder and the number of registers that are used between the basic blocks both depend on this parameter.




(a)




(b)

**Figure 6**. (a) Basic block (other than the first block) of registered DX architecture for $M = 4$. (b) First block of registered DX architecture for $M = 4$.

## IV. Results and Discussion

In this section, the efficiency of the proposed technique is evaluated by comparing its results by those of the conventional INC-XOR architecture given in Figure 1. Two different DX architectures were presented in the above. Although the partial DX encoder is very simple, when the sequential addresses on the bus overflow with respect to the 256 bound, some transitions will occur on the bus. Given the small number of gate delays on the critical path and the weak dependence of the delay on the bus width, higher performance and lower power consumption is achieved by the proposed technique

compared to the conventional INC-XOR encoder/decoder. To assess the improvements quantitatively, we need to evaluate the power dissipation and the delay of the encoder and the decoder. For this purpose, the encoder and decoder architectures described in the previous sections have been implemented in Verilog HDL at the RT level, simulated for functional verification, and a prototype has been synthesized using Synopsys Design Compiler with the a 0.18μm standard CMOS library with a supply voltage of 1.8V.

### *A. Analytical Timing Analysis*

There are different choices for the width of each DX block depending on the bus width. Here, we obtain an approximate relation between the maximum delay of the incrementer of the encoder as a function of the bus width and the width of the DX block. Suppose that the bus width is $W$ and each DX slice is $M$ bit wide. As explained, the model used in [13] is utilized for computing the delay and the area of incrementer. The critical path of the pseudo incrementer will pass through the last slice of the DX block, where the $W^{th}$ bit ($Y[W-1]$) is constructed. First, consider the 'ci' input of this slice which is generated using $(W/M-1)$ AND gates. If the balanced tree of 2-input AND gates is used for constructing the 'ci' input, then the delay of 'ci', $\tau_{ci}$, (with the delay of an AND gate delay set as a unit delay) will be given by

$$\tau_{ci} = \lceil \log_2 (W/M - 1) \rceil \qquad (1)$$

We have used "$\lceil X \rceil$" to denote the ceiling operation to the nearest integer number which is equal to or larger than $X$. On the other hand, note that the most significant bit in each slice has the maximum delay in that slice. This delay is equal to the sum of the delays of the 'ci' input to that slice and the internal delay of the DX block ($\tau_{int}$). The latter delay, which is the delay of $(M-1)$ AND gates, is given by

$$\tau_{int} = \lceil \log_2 (M - 1) \rceil \qquad (2)$$

where a balanced tree of 2-input AND gates is assumed. Therefore, the total delay of the pseudo incrementer ($\tau_{inc}$) is obtained as

$$\tau_{inc} = \lceil \log_2 (W/M - 1) \rceil + \lceil \log_2 (M - 1) \rceil \qquad (3)$$

The area of the proposed incrementer is the sum of the areas consumed by the DX blocks, the external AND gates, and the flip-flops. First, notice that because the AND chain that starts from the 'ci' input inside each basic block should be constructed by a balanced tree AND network for a faster generation (Figure 6 corresponds to a ripple carry format and not a balanced tree one), extra AND gates are required to concurrently compute the required intermediate AND products for the ripple carry adder, which are not obtained from the balanced AND tree network. For example, when the AND network has eight inputs, seven AND gates are required for constructing the tree while three other AND gates are required for computing the intermediate AND values. Therefore, on average 1.25 AND gates per input are needed for constructing the 'ci' tree of the slice. The tree inputs are the first $M - 1$ bits of $X_{n-1}$ (see Figure 6 (a)). As the 'co' output of the slice, which is obtained by ANDing of the last $M - 1$ bits of $X_{n-1}$, is registered, there is no need to use a balanced tree. Based on this count and considering the $M$ XOR gates, which is assumed to be equivalent to $(2 \times M)$ AND gates, the area of the incrementer logic inside each slice equals to $4.25 \times M - 2.25$ (AND gate) area unit. This area should be multiplied by $W/M$ which is the number of the slices in the DX encode/decoder. Similarly, the area for the external AND gate network can be estimated as $1.25 \times (W/M - 1)$ when it is constructed by a balanced tree. On the other hand, there are $(W/M - 1)$ flip-

flops in the design. According to the TSMC 0.18 standard library cell data book, each flip-flop area with asynchronous set and reset and enable input is equal to the area of the five AND gates. Hence, the total area of the incrementer ($A_{inc}$) is obtained from

$$A_{inc} = 4.25 \times W + 4 \times W/M - 6.25 \qquad (4)$$

The above analysis may be used for obtaining the optimum $M$ for any $W$. For example, in the case of a 32-bit address bus width, by differentiating (3) with respect to $M$, the optimum of $M$ in terms of delay is calculated to be four. Using the above equations, the optimum $M$ in terms of delay has been computed for different bus widths and the areas and the delays of the proposed circuit have been compared with those of the conventional incrementers. Table II shows the comparison results for different incrementer structures with different widths. As evident from the results, the proposed structure has the lowest delay while its circuit area is comparable to the fastest conventional circuit.

**Table II.** Area and delay comparison of conventional incrementers and the proposed circuit.

| Incrementer | 8-bit | | 16-bit | | 24-bit | | 32-bit | | 40-bit | |
|---|---|---|---|---|---|---|---|---|---|---|
| | AREA | DELAY | AREA | DELAY | AREA | DELAY | AREA | DELAY | AREA | DELAY |
| Ripple Carry | 24 | 9 | 48 | 17 | 72 | 25 | 96 | 33 | 120 | 41 |
| Carry Look-Ahead | 48 | 12 | 96 | 16 | 144 | 18 | 192 | 20 | 240 | 21 |
| Sklansky Prefix | 28 | 5 | 64 | 6 | 103 | 7 | 144 | 7 | 186 | 8 |
| Proposed Circuit | 35 | 3 | 72 | 4 | 112 | 5 | 145 | 5 | 184 | 6 |

### *B. Synthesis Results*

The switching activities of the gate-level implementation of the encoding/decoding circuits were obtained by simulating the synthesized blocks with the streams of addresses. We have used Synopsys Power Compiler tool to correlate the switching

activities to the power dissipation. Table III shows the performance parameters for the 40-bits encoders and decoders of the both partial and registered DX architectures and the conventional INC-XOR schemes using a 0.18μm CMOS technology with 1.8V supply voltage and 180-MHz clock frequency. The conventional INC-XOR encoder and decoder were implemented using ripple carry and carry look-ahead, and also Sklansky-Prefix incrementers. The clock frequency of 205-MHz was chosen to accommodate the slowest critical path of 4.87 ns in the circuits. The results show that the proposed structures have smaller delays compared to the INC-XOR encoder/decoder architectures. The partial DX encoder/decoder has the smallest delay and power consumption among all structures. However, one should note that there could be more transitions on the bus in comparison with the registered DX and the conventional INC-XOR architectures if the sequential address series leads to changes in bits other than the least 8 significant bits or a jump to an address out of 256 bound occurs.

The power dissipation was estimated using SPEC2000 benchmark programs [14]. The results are based on averaging over four programs named *"parser"*, *"vortex"*, *"gcc"*, and *"art"* when they were executed on the LEON processor [15]. The encoder and decoder were used on the AMBA (Advance Microcontroller Bus Architecture) bus of the LEON processor at the master (for the processor core) and slave (the main memory) side, respectively. Although, the area and power consumptions of the registered DX encoder/decoder are higher compared to those of the INC-XOR architecture with the ripple carry adder, the delay reduction of the registered DX is considerably less in this case. If the carry look-ahead adder or Sklansky-Prefix structure is used in the incrementer, the area, the delay, and the power of the registered DX encoder/decoder are

less. Both the power delay and the energy delay products of the registered DX encoder/decoder are much lower compared to the other implementations of the conventional INC-XOR encoder/decoder.

**Table III.** Performance Parameters for DX and INC-XOR.

| Parameter | DX | | INC-XOR (Ripple-Carry) | INC-XOR (Carry Look-Ahead) | INC-XOR (Sklansky Prefix Adder) |
|---|---|---|---|---|---|
| | Partial | Registered | | | |
| $P_{encoder} + P_{decoder}$ (mW) | 0.213 | 0.277 | 0.256 | 0.398 | 0.288 |
| Critical Path (ns) | 0.96 | 1.68 | 4.87 | 3.08 | 2.39 |
| Encoder Area ($\mu m^2$) | 4773 | 8970 | 6912 | 9677 | 8934 |
| PDP ($10^{-12}$J) | 0.204 | 0.465 | 1.246 | 1.225 | 0.688 |
| $PD^2P$ ($10^{-15}$Js) | 0.196 | 0.781 | 6.071 | 3.775 | 1.645 |

It is imperative to mention that the proposed implementation of the INC-XOR is registered-based (see Figure 5) and, hence, the use of encoding only increases the latency without affecting the delay. Therefore, the bus speed will not be affected. In general, any encoding system has an encoder and a decoder at the both ends of the bus imposing some timing, area, and power overheads. In low-power applications where minimizing the power is the primary concern, the encoding should be used to lower the overall power consumption (*encoder_power* + *bus_power* + *decoder_power*) compared to the case of no coding where only the *bus_power* exists. If the power of the encoding case is lower, then one should check the timing (delay/latency) and the area overheads of the encoding case. If these overheads are within the acceptable range for that application, then the use of the encoding is justified. Obviously, as the bus length increases, the relative weight of fixed-complexity overheads becomes smaller.

To assess the efficiency of the technique, we have estimated the average power (per line) associated with the switching of the on-chip address bus lines in the 180nm

technology. For estimating the bus power, the average transition per bus line was computed by running SPEC2000 benchmarks. Using this parameter and the data about the global interconnect width, spacing, thickness, and the dielectric height numbers reported in [17] for the 180nm technology, the power dissipation for each bus width and line length was computed. Table IV shows the average power consumptions of the on-chip buses. For the buses with coding, the power dissipations of the registered DX encoder and decoder shown in Table V have been included. As the results show, the power difference between the coding and non-coding cases increases with the increasing the length. It should be noted that since the power overhead of the encoder/decoder is less for the case of the partial DX architecture when compared to the case of the registered DX architecture, even more power saving will be achieved if the coding technique is used with these encoder/decoder.

**Table IV.** Power consumption (mW) for different on-chip address bus widths and line lengths with (C) and without coding (NC).

| Length (mm) | 8-bit | | 16-bit | | 24-bit | | 32-bit | | 40-bit | |
|---|---|---|---|---|---|---|---|---|---|---|
| | C | NC | C | NC | C | NC | C | NC | C | NC |
| 1 | 0.234 | 0.332 | 0.277 | 0.353 | 0.310 | 0.363 | 0.334 | 0.370 | 0.350 | 0.376 |
| 1.5 | 0.249 | 0.409 | 0.292 | 0.434 | 0.326 | 0.447 | 0.350 | 0.456 | 0.367 | 0.463 |
| 2 | 0.265 | 0.499 | 0.310 | 0.530 | 0.344 | 0.545 | 0.369 | 0.556 | 0.385 | 0.565 |
| 2.5 | 0.278 | 0.569 | 0.324 | 0.604 | 0.358 | 0.621 | 0.383 | 0.634 | 0.400 | 0.644 |
| 3 | 0.293 | 0.650 | 0.339 | 0.690 | 0.374 | 0.709 | 0.400 | 0.724 | 0.417 | 0.735 |

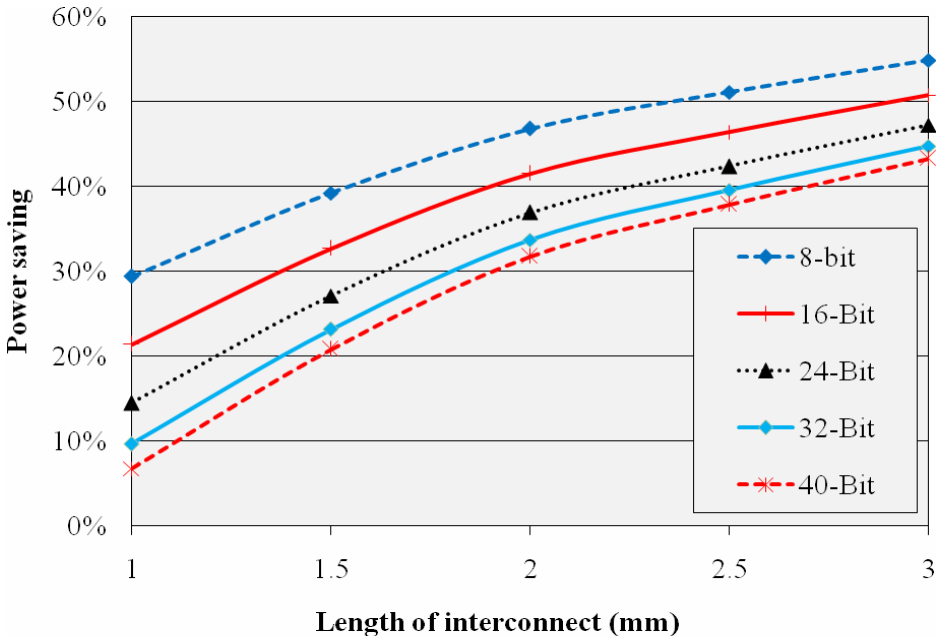
**Table V.** Power consumed in the registered DX encoder and decoder for different bus widths.

| Bus Width | $P_{encoder} + P_{decoder}$ (mW) |
|---|---|
| 8-bit | 0.173 |
| 16-bit | 0.212 |
| 24-bit | 0.235 |
| 32-bit | 0.266 |
| 40-bit | 0.277 |

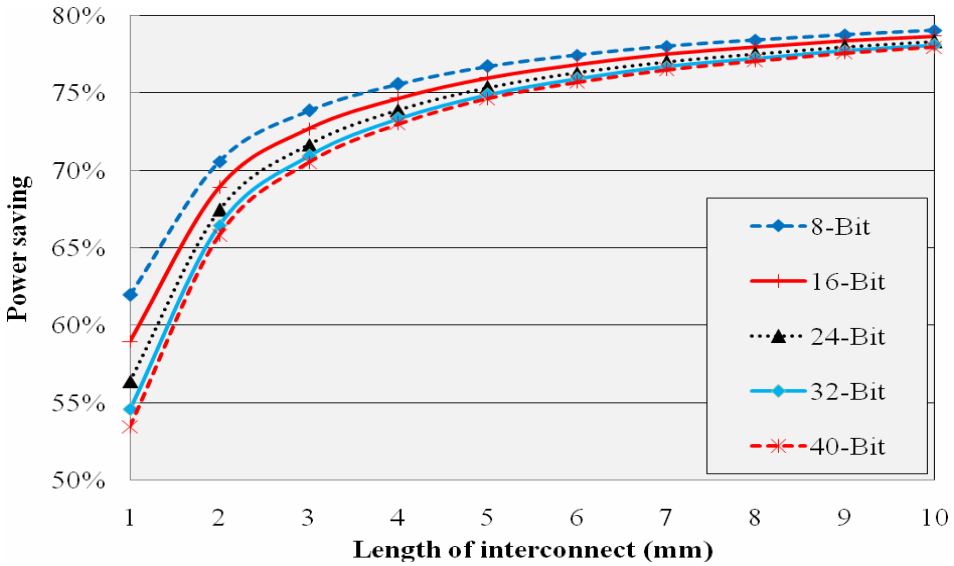
Now we consider the use of the coding for off-chip addressing of the memory. Table VI contains the power estimation for off-chip busses as a function of the bus bit-width and line length with and without coding. A bus capacitance per length of 1pF/mm has been assumed in these calculations [18][19]. In the case of off-chip buses, since the switching power associated with the bus is much higher, the power savings obtained are much more compared to the case of the on-chip buses. It should be mentioned that as the results presented in Table IV and Table VI reveals the power dissipation of coding case increases more aggressively with the bus bit width as opposed to the bus length. The exact opposite is true for the non-coded case. The reasons for this behavior is the dependence (independence) of the encoder/decoder power overhead with the bus bit width (length) and the diminished effect of the bus power in the overall power consumption when the coding is used. Finally, we have plotted the power savings achieved through coding as a function of the length for different bus widths in Figure 7. As expected, the results show that as the bus length increases, the amount of power saving also increases.

**Table VI.** Power consumption (mW) for different off-chip address bus widths/lengths with (C) and without coding (NC).

| Length (cm) | 8-bit | | 16-bit | | 24-bit | | 32-bit | | 40-bit | |
|---|---|---|---|---|---|---|---|---|---|---|
| | C | NC | C | NC | C | NC | C | NC | C | NC |
| 1 | 0.336 | 0.884 | 0.386 | 0.939 | 0.421 | 0.966 | 0.448 | 0.986 | 0.466 | 1.001 |
| 2 | 0.465 | 1.581 | 0.522 | 1.679 | 0.562 | 1.726 | 0.592 | 1.763 | 0.612 | 1.789 |
| 3 | 0.590 | 2.256 | 0.655 | 2.396 | 0.698 | 2.464 | 0.731 | 2.516 | 0.753 | 2.554 |
| 4 | 0.711 | 2.913 | 0.784 | 3.094 | 0.831 | 3.181 | 0.866 | 3.249 | 0.890 | 3.298 |
| 5 | 0.838 | 3.600 | 0.919 | 3.823 | 0.969 | 3.931 | 1.008 | 4.015 | 1.034 | 4.075 |
| 6 | 0.958 | 4.247 | 1.045 | 4.510 | 1.100 | 4.638 | 1.141 | 4.736 | 1.169 | 4.808 |
| 7 | 1.086 | 4.939 | 1.181 | 5.245 | 1.240 | 5.393 | 1.284 | 5.507 | 1.314 | 5.591 |
| 8 | 1.204 | 5.581 | 1.307 | 5.927 | 1.369 | 6.094 | 1.416 | 6.223 | 1.448 | 6.317 |
| 9 | 1.333 | 6.276 | 1.444 | 6.665 | 1.509 | 6.853 | 1.559 | 6.998 | 1.594 | 7.104 |
| 10 | 1.460 | 6.964 | 1.579 | 7.396 | 1.648 | 7.605 | 1.701 | 7.766 | 1.738 | 7.883 |




(a)




(b)

**Figure 7**. The power savings achieved through coding as a function of the length for different bus widths. (a) On-chip buses (b) off-chip buses.

## V. Summary and Conclusion

We presented new encoder/decoder architecture for the INC-XOR encoding method to reduce the delay, the power delay product, and the energy delay product. Both the

analytical and simulation results showed that the proposed architecture was very fast in comparison to the conventional encoder/decoder architectures implemented using the ripple carry, carry look-ahead, and Sklansky-prefix incrementers. The simulations results showed considerable improvements for these parameters in the case of the proposed structure. An important feature of the proposed architecture is that its delay is weakly dependent on the address bus width compared to the conventional INC-XOR encoder/decoder architecture. Therefore, the proposed architecture improves the features of the INC-XOR address bus encoder/decoder.

## References


[1] M. R. Stan and W. P. Burleson, "Bus-Invert Coding for low-Power I/O," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 3, no. 1, Mar. 1995, pp. 49-58.

[2] Y. Shin, S. I. Chae, and K. Choi, "Partial Bus-Invert Coding for Power Optimization of System Level Bus," *Proc. Int'l Symp. on Low Power Electronics and Design*, Aug. 1998, pp. 127-129.

[3] L. Benini, G. De Micheli, E. Macii, D. Sciuto, and C. Silvano, "Asymptotic Zero-Transition Activity Encoding for Address Buses in Low-Power Microprocessor-Based Systems," *Proc. of 7th Great Lakes Symp. on VLSI*, Mar. 1997, pp. 77-82.

[4] L. Benini, G. De Micheli, E. Macii, D. Sciuto, and C. Silvano, "Address Bus Encoding Techniques for System-Level Power Optimization," *Proc. of Design Automation and Test in Europe*, 1998, pp. 861-866.

[5] S. Ramprasad, N. Shanbhag, and I. Hajj, “A Coding Framework for Low Power Address and Data Buses,” *IEEE Trans. Very Large Scale Integration Systems*, vol. 7, June 1999, pp. 212-221.

[6] W. Fornaciari, M. Polentarutti, D. Sciuto, and C. Silvano, “Power Optimization of System-Level Address Buses Based on Software Profiling,” *Proc. of Int’l Conference on Hardware-Software Codesign and System Synthesis*, 2000, pp. 29-33.

[7] Y. Aghaghiri, F. Fallah, and M. Pedram, “Irredundant address bus encoding for low power,” *Proc. of Int'l Symp. on Low Power Electronics and Design,* Aug. 2001, pp. 82-187.

[8] Y. Aghaghiri, F. Fallah, and M. Pedram, “ALBORZ: address level bus power optimization,” *Proc. of Int'l Symp. on Quality of Electronic Designs*, Mar. 2002, pp. 470-475.

[9] H. Mehta, R. M. Owens, and M. J. Irwin, “Some Issues in Gray Code Addressing,” *Proc. of the* $6^{th}$ *Great Lakes Symposium on VLSI,* March 1996, pp.178-180.

[10] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press, New York, 2000.

[11] Reto Zimmermann, *"Binary Adder Architectures for Cell-Based VLSI and their Synthesis*, Ph.D. thesis, Swiss Federal Institute of Technology, Zurich, 1997.

[12] J. Sklansky, “Conditional sum addition logic,” *IRE Transactions on Electronic Computers*, volume EC-9, no. 6, June 1960, pp. 226–231.

[13] A. Tyagi, “A reduced-area scheme for carry-select adders,” *IEEE Transactions on Computers*, vol. 42, no. 10, October 1993, pp. 1163–1170.

[14] SPEC Benchmark data, http://www.spec.org.

[15] LEON2 Processor (2005), http://www.gaisler.com/products/leon2/leon.html.

[16] *Computer Architecture, A Quantitative Approach*, John L. Henessey and David A. Patterson, 3rd Ed., 2002.

[17] Predictive Technology Model (PTM): http://www.eas.asu.edu/~ptm/.

[18] Ioannis Koryfidis, "Power Aware HW/SW Partitioning for a DVB-H Receiver Module," *M.Sc. Thesis*, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, 2006.

[19] Mahesh Mamidipaka, Nikil Dutt, and Dan Hirschberg, "Efficient Power Reduction Techniques for Time Multiplexed Address Buses," in *Proceedings of ISSS'02*, October 2–4, 2002, pp. 207-212.