A Very Fast and Low Power Pseudo-Incrementer for Address Bus Encoder/Decoder

Hadi Parandeh-Afshar [1], Ali Afzali-Kusha[1], and Ali Khakifirouz[2]

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

[2] Department of Electrical and Computer Engineering, Microsystems Technology Laboratories Massachusetts Institute of Technology, USA

hparande@ut.ac.ir, afzali@ut.ac.ir, khaki@mit.edu

***Abstract*—This paper presents a very fast yet low power pseudo incrementer structure which may be used in address bus encoders/decoders. This structure, which is based on the ripple carry incrementer, is much faster than the incrementer. Using this structure, the delay and the power of address bus encoders/decoders may be reduced considerably. Analytical and synthesis results show that the structure is faster than current incrementer circuits while its circuit area and power are much smaller than those of current fast incrementers.**

Index Terms—Bus encoder, Incrementer, Low Power, High Performance.

I. INTRODUCTION

For the design of high-speed low-power VLSI architectures efficient processing units which are optimized for the speed and power consumption should be used. Among different modules of a digital system, arithmetic blocks have a key effect in minimizing the power and the delay of the digital system. Recently, for many reasons such as the limited battery life, maximum allowed chip temperature, reliability issues, and cooling and packaging costs the power minimization has become a vital design objective. Consequently, low power design methodologies at different design level have received a significant attention. Since the circuit speed is also an important design objective, it is desired that wherever possible, these methodologies do not adversely affect the speed of the digital system.

In the state of the art digital systems, the energy dissipation per memory bus access could be unacceptably high. The large power consumption is due to the high capacitance of the buses. In a computer system, the generated addresses on an address bus are often in-sequence and two sequential addresses have the difference of one unit. Therefore, instead of placing sequential values on the bus, the value at the source (source word) is examined. If the value is only one unit greater than the previous address, then the address bus will remain unchanged. If this is not the case, the previous address is incremented and XORed by the current address and the results of the XOR is sent over the bus. Then, the same address should be constructed in the decoder. Among the efficient methods used for the address bus encoding are T0-XOR and a combination of T0-XOR and other methods [1][2]. The T0-XOR scheme, which is very efficient for consecutive addresses, an encoder (/decoder) is used for reducing the bus switching activities. Here, both the encoder and the decoder modules make use of an incrementer whose width is equal to that of the bus. Therefore, an incrementer is an essential component of the bus encoding/decoding blocks. As the bus width increases, the logic complexity, the delay, and the power consumption of this incrementer also increase proportional to the bus width. This increase can lead to the deterioration of the delay, the power, and the area of the overall structure.

In this paper, we propose a structure which plays the incrementer role for the T0-XOR encoder/decoder. With the proposed incrementer, the delay, the power-delay product, and the energy-delay product of the encoder/decoder improves while the power dissipation and the area overhead become less dependent on the bus width. The rest of this paper is organized as follows. In Section II, we provide a review of different incrementer structures while in Section III the proposed architecture is presented. Section IV shows a comparison between the proposed architecture and conventional incrementers. The summary and concluding remarks are given in Section V.

II. CONVENTIONAL INCREMENTER CIRCUITS

In this work, we have studied three incrementer architectures which include ripple-carry structure, binary-tree Carry Look Ahead structure [3], and Sklansky Prefix-adders [4][5]. 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 [6] which relates the required circuit area with the propagation time of each logic cell [4]. In this approach, each two-input monotonic gate (e.g., AND, NAND) is counted as one gate while an XOR is counted as two gates in terms of area and delay. In [3], 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. As the results show, the ripple carry incrementer has the minimum area while the Sklansky Prefix incrementer has the minimum delay.

TABLE I. CONVENTIONAL INCREMENTER FEATURES [3].

Incrementer	Circuit Area	Circuit Delay
Ripple Carry	$3.W$	$W+1$
Carry Look-Ahead	$6.W$	$4.\text{Log}_2W$
Sklansky Prefix	$1/2.W.\log_2W+2W$	$\text{Log}_2W + 2$

'*W*' is the incrementer width.

III. PROPOSED ARCHITECTURE

The structure of the proposed architecture is based on the ripple carry incrementer. Fig. 1 shows the internal logic of a 4-bit ripple carry incrementer. The half adders, denoted by HA, are used for constructing the incrementers. In this figure, the critical delay is the delay of carry propagation through the AND gates chain. As the size of the chain grows, this delay increases making the delay of the most significant bit the largest one.

To alleviate the delay problem, we use the fact that when incrementing a number, the carry will not propagate if several consecutive bits are zero. This is achieved in the proposed solution by inserting flip-flops between each *M* bit slice. These slices are called RC (Ripple Carry) blocks. Each flip-flop stores the result of ANDing of *M* bits of the previous address which should be incremented (see Fig. 2). Inserting flip-flops in the AND gate chain produces a new chain of AND gates which is much shorter than the previous one. This way, we can speed up the carry propagation to the MSB bits. Indeed we have pipelined the incrementer by considering the consecutive addresses. This causes that delay of each stage goes down, while the total latency remains unchanged. This is possible by considering that the upper stages do not change from one cycle to next; so most of the time we can use previous values of these stages.

The new chain consists of two separate chains: before and after the flip-flop. The first part of this chain is inside the RC slices and performs AND of 'X' bits inside that slice to form the 'co' output. This is shown in Fig. 3, where a basic RC slice for *M* = 4 is shown. Compared to the circuit of Fig. 1, 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 RS blocks, where the 'ci' inputs of the RC blocks are constructed. Note that the 'ci' input of the last RC 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 will possibly 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', and, therefore, the generations of 'co' outputs of all M-bit blocks are performed in parallel.

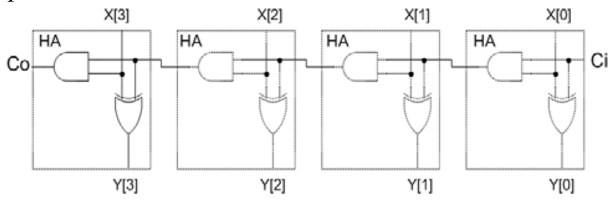

Fig. 1. 4-bit ripple carry incrementer.

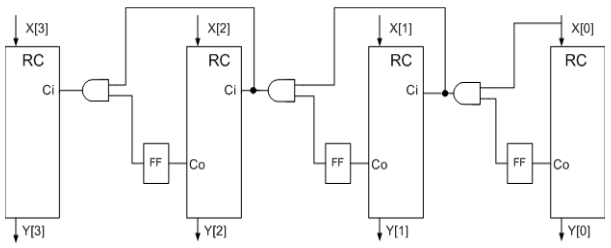

Fig. 2. General structure of proposed circuit.

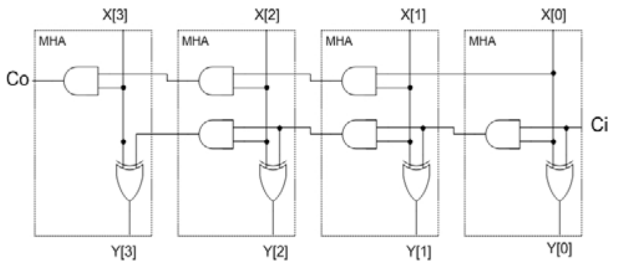

Fig. 3. RC block proposed circuit with *M* = 4.

To see how this circuit works, assume that the following sequential values should be generated by a 16-bit incrementer:

Time	Generated Address X[15:0]
0	0010 0101 1111 1101
1	0010 0101 **1111 1110**
2	0010 0101 **1111 1111**
3	0010 0110 0000 0000
4	0010 0110 0000 0001

Note that the '1' digits that are shown in bold font (bit 7 to bit 1) in time 1 are repeated at time 2. This means that bits (7 to 1) of the address in time 1 can be used instead of the same bits in time 2 to reduce the delay. Let us consider the case where all the 'X' bits for a slice (say, X [4:7] in time 2) is equal to those in time 1, then 'co' is generated in the previous cycle as 1 and is stored in the register. Now, if 'ci' for this block is 1, then for the next block, the 'ci' is generated to be 1, meaning that some of the addresses associated with this block should change. This way, we have managed to generate 'ci' for the next block faster which yields a quicker generation of the corresponding Y's.

The above explanation was based on with the assumption that the corresponding input bits in the two consecutive cycles are the same. If this is not case, again the circuit will work correctly. To show this, consider the case that the corresponding bits in the two cycles are not the same. The only case, in consecutive sequences, that this difference affects the output of the first chain is the case where all the M bits of the input in the previous time are 1's (this is the only case where the output of the first AND gate chain is 1) and in the next cycle, they all become 0 making the output of the AND gate chain 0 while the register has stored 1. Note that the output of the second AND gate, whose inputs are the output of the flip-flop and the 'ci' of the current block, is the 'ci' of the next block. Since the latter input for this special case is 0, then the output of the flip-flop, which is 1, is masked ensuring the correct operation of the circuit. For other cases, which occur rarely in the stream of instruction addresses, the inputs are not consecutive. In these cases, the proposed pseudo incrementer may generate a value different from the incremented value of new input. This is the reason for calling the structure '*pseudo* incrementer'. Here, the result is corrected for the next consecutive address. As the decoder uses the same structure, no address conflicts occurs and the original address is recovered in the decoder side.

These basic M-bit RC blocks are cascaded for wider buses. Fig. 2 shows a 32-bit pseudo incrementer in which four 8-bit RC blocks have been cascaded. As explained, by using intermediate flip-flops, the original functionality of incrementer will be kept, while the critical path of the incrementer is reduced considerably. For an incrementer wider than RC blocks, e.g., 32 or 64 bits, the critical path delay is equal to the maximum delay of the RC blocks plus the propagation delay of the last block 'ci' input. The longest path through each RC block is equal to the generation delay of the output bit for the last AND gate in the block. The propagation delay of the last block 'ci' input is reduced by using a balanced tree of 2-input AND gates.

From the above discussion, it may be concluded that the number of flip-flops, which are used in the architecture, depends on the widths of the RC block and the whole pseudo incrementer. It will be shown in the next section, for a 64-bits bus, the optimum number of blocks is four leading to 15 flip-flops. This width is chosen to minimize both the delay and the area of the circuit.

IV. Results and Discussion

In this section, the efficiency of the proposed technique is evaluated by comparing its results by those of the conventional incrementer architectures. Given the small number of gates on the critical path and the weak dependence of the delay on the bus width, a higher performance and a lower power consumption is expected to be achieved for the proposed technique compared to those of the conventional incrementers. To evaluate the power dissipation and the delay of the incrementers, the architectures described in this paper were implemented in Verilog HDL at the RT level and simulated for the functional verification. Then, a prototype was synthesized using the Synopsys Design Compiler with the TSMC 0.18μm library designed for the operation at 1.6V.

A. Analytical Timing Analysis

There are different choices for the width of each RC block depending on the incrementer width. Here, we obtain an approximate relation between the maximum delay of the incrementer as a function of the incrementer width and the width of the RC block. Suppose that the bus width is W and each RC slice is M bit wide. As explained in Section II, the model used in [6] is used for computing the delay and the area of incrementer. The critical path of the pseudo incrementer will pass through the last slice of the RC 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 'ci' input, then the delay of 'ci' (τ_{ci}) is given by

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

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, ($M-1$) AND gates, and one XOR gate. As the XOR gate is a two level gate, its delay is 2 and, hence, the internal delay of each RC (τ_{int}) block is given by

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

where a balanced tree of 2-input AND gates is used for constructing the AND products. Therefore, the total delay of the pseudo incrementer (τ_{inc}) is obtained as

$$\tau_{\mathrm{inc}} = \log_2(W/M - 1) + \log_2(M - 1) + 2 \qquad (3)$$

This analysis may be used in obtaining the optimum M for any W. For example, in the case of a 64-bit address bus width, the optimum M is four (see Fig. 4). With this width, the delay of the pseudo incrementer is about 7.

The area of the proposed incrementer is the sum of the areas consumed by the RC blocks, external AND gates, and flip-flops. Each RC block has 4.($M-1$) area unit. Also, there are ($W/M-1$) external AND gates and flip-flops. For the sake of analytical analysis, we use a pass-transistors implementation of flip-flops and, hence, the area of each flip-flop is considered approximately equal to two AND gates area. The total area of the incrementer (A_{inc}) is obtained from

$$A_{inc} = 4W - W/M - 3 \qquad (4)$$

Using the above computations and assuming that the pseudo incrementer is 64-bit wide and the RC is 4-bit wide, Table II shows the comparison results for different incrementer structures. As evident from the results, the proposed structure has the lowest delay while its circuit area is smaller than fastest conventional structure.

Top

B. Synthesis Results

The switching activities of the gate-level implementation of the incrementer circuits were obtained by simulating the synthesized blocks with the random streams of addresses. We have used Synopsys Design Power to correlate the switching activities to the power dissipation. Table III shows the performance parameters for the 64-bits incrementers of proposed architectures and the conventional incrementer structures using a 0.18µm CMOS technology with a supply voltage of 1.6V. The results show that the proposed structure has a lower delay compared to those of the conventional incrementers while its area and power are much smaller than those of the previous fast structures.

V. Summary and Conclusion

We presented new pseudo incrementer architecture suitable for address bus encoder/decoder. Both the analytical and synthesis results showed that the proposed architecture was very fast in comparison with the conventional incrementer architectures, while its circuit area was also much smaller than that of conventional structure. In addition, the simulations results showed a considerable improvement in reducing the power consumption was achieved in the proposed incrementer.

References

[1] W. Fornaciari, M. Polentarutti, D. Sciuto, and C. Silvano, "Power Optimization of System-Level Address Buses Based on Software Profiling," *CODES*, pp. 29-33, 2000.

[2] S. Ramprasad, N. Shanbhag, and I.N. Hajj, "A Coding Framework for Low-Power Address and Data Buses," *IEEE Transaction on Very Large Scale Integration Systems*, June 1999.

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

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

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

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

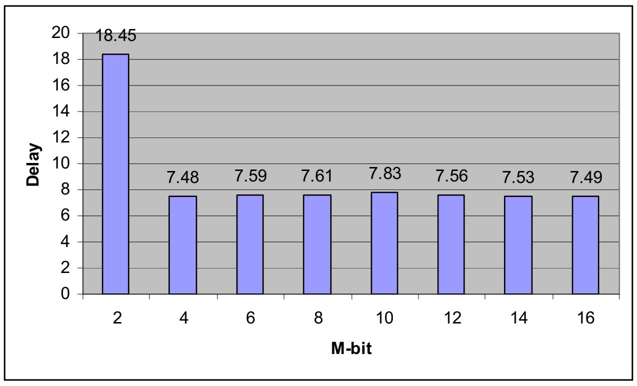

Fig. 4. Delay of proposed circuit as a function of different slice widths.

TABLE II. Comparison of Conventional Incrementers and Proposed Circuit.

Incrementer	Circuit Area	Circuit Delay
Ripple Carry	192	65
Carry Look-Ahead	384	24
Sklansky Prefix	320	8
Proposed Circuit	237	7

TABLE III. Performance Comparison.

Incrementer	Ripple Carry	Carry Look Ahead	Sklansky Prefix Adder	Proposed Circuit
Power (mW)	0.297	0.534	0.476	0.320
Critical Path (ns)	7.64	3.74	1.01	0.88
# of Transistors	14890	30533	27017	18776
PDP (10^{-12}J)	2.269	1.997	0.480	0.281
PD^{2}P (10^{-15}Js)	17.33	7.468	0.484	0.247

Top