# Design and Performance Verification of ALUs for 64-bit 8-Issue Superscaler Microprocessors Using 0.25um CMOS Technology

*Chua-Chin Wang, and Sheng-Hua Chen*
*Department of Electrical Engineering, National Sun Yat-Sen University Kaohsiung, Taiwan 80424*
*Email: ccwang@ee.nsysu.edu.tw*

*Shen -Fu Hsiao*
*Institute of Computer and Information Engineering National Sun Yat-Sen University Kaohsiung, Taiwan 80424*

*Chuan-Lin Wu*
*Department of Computer Science and Information Engineering National Chiao-Tung University Hsinchu, Taiwan 300*

## ABSTRACT

In this paper, we present designs of a set of four non-homogeneous ALUs which can be employed in the next generation 64-bit x86-compatible microprocessors. The entire design is realized by synthesizable Verilog RTL (register-transfer level) code. The gate level code is generated by Synopsys using COMPASS 0.6um 1P3M cell library, and UMC 0.25um 1P5M cell library. The correctness of the functionality of the individual ALU is verified in both RTL code and gate level code after the synthesization.

## 1. INTRODUCTION

The major functions of ALUs are aimed at the executions of fixed-point integer instructions and logic operations. We tend to design four non-homogeneous ALUs for a next generation 64-bit octal-issue x86-compatible CPU with superscaler and pipelining functions [10], [5]. The major critical components include 64-bit adders, a 64x64 multiplier, a 128/64 divider, 64-b barrel shifters, and logic units, [2], are all realized by Verilog synthesizable RTL code. Owing to the demand of superscaler architecture [4], certain components must be replicated and placed efficiently inside the ALUs in order for parallel processing of more than one instruction at a time, [6], [7]. The proposed 4 ALUs are finally synthesized with designated cell libraries, i.e., COMPASS 0.6um 1P3M cells and UMC (United Microelectronic Company) 0.25um 1P5M cells, by SYNOPSYS [1]. The results turned out to be very promising regarding the speed and area performance.

## 2. DESIGNS of ALUS for 64-BIT uPs

### 2.1 Key issues to be considered in the design

According to the specifications of [8] and [10], there are a total of 97 assembly instructions (called POPs, primitive opcode) to be implemented in ALUs despite the various operand lengths. Several factors must be considers when it comes to the realization of ALUs for all of the POPs required for the micro-architecture of the 64-bit advanced microprocessor.

1). spatial location for the placement of all functions units in the uP
2). POPs to be implemented in each ALU
3). trade-off between area and speed
4). partition and modulization of internal function blocks of each ALU

#### 2.1.1 Spatial location of ALUs

Referring to Fig. 1, the microarchitecture of the 8-issue superscaler uP is revealed, where 4 ALUs, 2 FPUs and 2 MMUs are required [10]. In order to maintain the granularity of the ALUs, they are placed between the reservation station (RS) and the result bus (Rb). The inputs to the ALUs are 154-bit data called "AluPop," while the outputs to the result bus include 89-bit "AluResult" and 16-bit "errorcode." The formats of these signals are shown in Fig. 2.

#### 2.1.2 POPs to be implemented

Besides the various length of operands, all of the POPs to be implemented in the ALU0 through ALU3 are determined by [8] and [10]. The code space for each POP is properly arranged such that the length of POP code (i.e., Opcode) is minimized, $\lceil 97 \rceil = 7$. As for the length of the operands, it is denoted by a field of two bits, "S1width" and "S2width" of source operand 1 and source operand 2, respectively. Meanwhile, in order to achieve the compatibility with x86 assembly code, the mnemonics of POPs are identical to those of 686 assembly language shown in [8] except a few new instructions.

#### 2.1.3 Trade-off of area and functionality

The functionality of a uP denotes how many different instructions it can execute in given number of clock cycles. Ideally, we can use as many ALUs or functional modules as possible to achieve any degree of parallelism or superscaler. However, owing to the limitations of chip area and power dissipation, the reduction of the number of function modules is necessary. For those function modules which are less invoked statistically but occupy lots of chip area, only one such module will be kept in the ALUs, e.g.,

0-7803-5682-9/99/$10.00©1999 IEEE.

multiplier module and divider module. In contrast, for those function modules which are frequently invoked and consume small chip area, we tend to increase the number of these modules such that the throughput of the uP can be enhanced, e.g., conditional move modules and adder modules.

### 2.1.4 Partition and modulization

The partitioning of POPs and the modulization of relevant POPs are the most difficult task to conquer. Sophisticated skills and experience might be helpful. We divide the POPs into the following categories which then are individually decoded and implemented as a single internal module.

1). addition-related -- e.g., ADD, ADC, etc.
2). move-related -- e.g., CMOVO, CMOVNL, etc.
3). flag-related -- e.g., CLC, STC, etc.
4). logic-related -- e.g., NOT, AND, etc.
5). bit-operation-related -- e.g., BTS, BSF, etc.
6). BCD-conversion-related -- e.g., DAA, AAS, etc.
7). shift-related -- e.g., ROL, SHL, etc.
8). multiplication-related -- e.g., UMUL, MULP, etc.
9). division-related -- e.g., DIV, MODI, etc.
10). set-related -- e.g., SETO, SETS, etc.
11). others -- e.g., BOUND, BSWAP, etc.

Thus, the entire ALU design will be hierarchical and executable.

## 2.2 Internal Architecture of ALU0

Referring to Fig. 3, the floorplan of ALU0 is illustrated by block diagrams. There are a total of 14 modules in ALU0. Notably, in order to distinguish the operation codes used in all ALUs and the function modules, "Opcode" (fixed 7-bit in length) is employed to denote the POP representation to the ALUs, while "OpCode" (variable in length) is used to denote the code decoded in each function module. The functions of the modules are described as follows.

1). Control : It is the control unit of the entire data flow. In other words, it is a finite state machine. It decodes the Reset_, Op, AluRbFull_, ready, and FU_Complete signals to generate Stall_, Req_ internal OpCode, and enableflag. If the current POP is not done, the Stall_ will disable the Latch module such that the previous AluPop data will be kept.
2). Latch : It is used to latch the AluPop signal and then dispatch the data fields therein to appropriate modules.
3). Addset : It executes ADD, ADC, SBB, NEG, SUB, SUBSD, SUBGP, INC, and DEC instructions. Basically, it is composed of a fast 64-bit adder and glue logics
4). Logic : It executes OR, NOT: AND, and XOR instructions.
5). Bit Test : This module executes BT, BTS, BTR., and BTC POPs.
6). Bit Scan : BSF (bit scan forward) and BSR (bit scan reverse) are executed in this module.
7). Flags : All of the flag bit operations are executed in this module, including CLC, STC, CMC, SAFLAG, CLD, and STD.
8). BCD : Conversion between BCD and binary formats is done in this module, including DAA, DAS, AAA, and AAS.
9). Cmove : Many move operations are required in uP's applications. This module executes all of the conditional move operations, e.g., CMOVO, CMOVNO, CMOVB, CMOVNB, CMOVZ, CMOVNZ, and so on.
10). Shift : Usual shift and rotate operations are done in this module, including ROL, ROR, RCL, RCR, SHL, SHR, and SAR.
11). MISC : Those instructions which are hard to be classified are realized in this module. These instructions are BOUND, MOV, and BSWAP.
12). ALU_Part_4 : This module also executes some POPs which are difficult to be fit in certain modules. The POPs include EXCP, MGF, SXT, SXTH, and XDPL.
13). FU_OK (function units OK) : When all function modules are done, the OK signals are delivered to this module. Then, an FU_complete signal is asserted to trigger RPU such that the appropriate result data will be placed on Rb.
14). RPU (result processing unit) : This module selects the output of decoded function module, i.e., one of the "DESTs" (destination, i.e., the result of each individual function module), and placed the result on Rb if the Rb is not busy.

## 2.3 Internal Architecture of ALU1

Referring to Fig. 4, the only difference between ALU1 and ALU0 is an extra "Shift_2" module in ALU1. The purpose of Shift_2 is to execute double precision shift of given data, i.e., 128-bit data shift. Since this module consumes large chip area and the number of appearances of the double precision shifting POP codes is statistically small, we tend to use one single module in all of the four ALUs. Another reason to insert this module in ALU1 is the limitation of POP code space

## 2.4 Internal Architecture of ALU2

The function modules employed in ALU2, as shown in Fig. 5, are mainly for multiplication operations. As we explained before, since the chip area for 64 64 multiplication operation is quite large, we need to discard some modules which are less utilized in application programs. Thus, the modules used in ALU2 are Addset, ALU_Part_4, Multiplier Set, BCD, CMove, SET Group, and MISC. Those modules which are either not in ALU0 or different from the same module

in ALU0 are introduced in the follows.

1). ALU_Part_4 : Besides the POPs implemented in the ALU_Part_1 of ALU0, several POPs are included in this module, which are XSB, XSL, MRG, and XTR.
2). Multiplier Set : It is the core module of ALU2. The multiplication-related POPs are executed in this module, including UMUL, MUL, MULL, AAD, UMULP, and MULP.
3). SET Group : All of the set,, instructions are executed in this module, including SETO, SETNO, SETB, SETNB, and so on.

### 2.5 Internal Architecture of ALU3

The core of ALU3, as shown in Fig. 6, is the divider module which realizes all of the division-related POPs. There are a total of 8 POPs to be executed in the "Divider" module, including DIVRNI, MODI, UDIV, DIV, AAM, UDIVP, DIVP, and LOPND. Note that the requirement of the integer division in a 64-bit uP is capable of executing 128/64 division. Hence, LOPND, i.e., "load operand", is needed to load 128-bit data in two consecutive clock cycles.

Note that the integer divider in this ALU adopts a digit-recurrence mixed 2-4 radix approach [2]. A prototype chip of this integer divider has been fabricated and tested. (CIC no. : T06-87C-09) Fig. 7 shows the die photo of the divider.

## 3. SYNTHESIZATION and VERIFICATION

The implementation of the four ALUs is realized by Verilog synthesizable RTL code. All of the Verilog code has been tested under the NSC98 simulator in Dept. of CSIE of National Chiao-Tung University, and it has turned out to be functionally correct. Thus, in order to have a clear picture whether the developed RTL code of the ALUs meets the speed and area budget of the 64-bit advanced uP, the synthesizable RTL codes are respectively synthesized by SYNOPSYS using COMPASS 0.6um 1P3M cell library and UMC 0.25 um 1P5M cell library. The results are summarized in Table 1.

According to the results shown in Table 1, a few things are to be noted. First, the chip area of the four ALUs are roughly in the same order. This provides an advantage when it comes to the placement and routing phase of the uP integration. Second, though the delay of the critical path employing 0.25um CMOS technology is very large, which is 33.72 ns, it does not necessarily imply that the operating clock frequency is around 30 MHz. In fact, these long delays are resulted from the instructions requiring multiple operating cycles, e.g., multiplications, divisions and addition/subtraction. The measured fastest operating clock period is 4.0 ns, which indicates a 250 MHz clock. Besides, if the full-custom design approach is adopted, the critical delay should be even smaller by a factor of 2 or 3. Third, the reason why ALU2 is much larger than other ALUs is the area of the 64x64 multiplier composed of logic gates. This part of design can be replaced with faster and smaller multiplier designs, e.g., CPL, or DPL designs. Four, the long delay generated by ALU3 is owing to the division steps of the integer divider. Because the digit-recurrence approach is utilized to realize this divider, the steps to complete an integer division is uncertain and more than three clock cycles.

Compared to other ALU designs of multiple issues superscaler microprocessors, [3] and [9], our proposed non-homogeneous four-ALU design reveals the superiority regarding area and speed as shown in Table 2.

## 4. CONCLUSION

In this work, we have presented feasible four non-homogeneous ALUs for advanced 64bit 8-issue x86-compatible microprocessors. The proposed four ALUs not only provide the facility of superscaler and pipelining, but also meet the critical timing constraint. The synthesizable RTL code are fully verified and tested, and the results of synthesis turn out to be promising in the future applications.

## 5. REFERENCES


[1] K. Baty, "Design Ware," Reading : pp. B-3 to B-12, Synopsys, Inc. 1996.
[2] M. D. Ercegovac, and T. Lang, "Division and square root - digit-recurrence algorithms and implementations," Reading : Kluwer Academic Publishers, 1994.
[3] N. Gaddis, and J. Lotz, "A 64-b quad-issue CMOS RISC microprocessor," *IEEE J. of Solid-State Circuits*, vol. 31, no. 11, pp. 1697-1702, Nov. 1996
[4] L. Gwennap, "Intel's P6 uses decoupled superscaler design," *Microprosessor Report*, vol. 9, no. 2, Feb. 1995.
[5] L. Gwennap, "Klamath extends P6 family," *Microprocessor Report*, vol. 11, no. 2, Feb. 1997.
[6] J. P. Hayes, "Computer architecture and organization," McGraw-Hill, Inc., 1988.
[7] K. Hwang, "Computer arithmetic principles, architectures, and designs," Reading :John Wiley & Sons, 1979.
[8] "Pentium Pro Family Developer's Manual," Intel, 1996.
[9] N. Vasseghi, K. Yeager, E. Sarto, and M. Seddighnezhad, "200-MHz Superscalar RISC microprocessor," *IEEE J. of Solid-State Circuits*, vol. 31, no. 11, pp. 1675-1685, Nov. 1996.
[10] C.-L. Wu, "2nd year technical report of an advanced 64-bit microprocessor," Reading: National Science Council, June 1998.

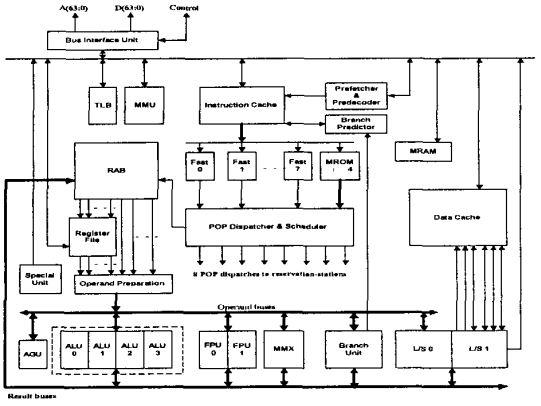


Fig. 1 : The microarchitecture of the 64-b superscalar uP

**AluPop format**

| 153 | 146 | 144 | 142 | 140 | 76 | 12 | 6 | 0 |
|---|---|---|---|---|---|---|---|---|
| Ready | Op | Dtsz | S1width | S2width | Src1 | Src2 | rFlag | Tagin |
| 1 | 7 | 2 | 2 | 2 | 64 | 64 | 6 | 6 |

**AluResult format**

| 85 | 84 | 23 | 17 | 15 | 7 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Req_ | Stall_ | ResultOut (DEST) | FlagsOut | Dsize | Exception | TagOut | E |
| 1 | 1 | 64 | 6 | 2 | 8 | 6 | 1 |

Fig. 2 : I/O data format of the ALUs

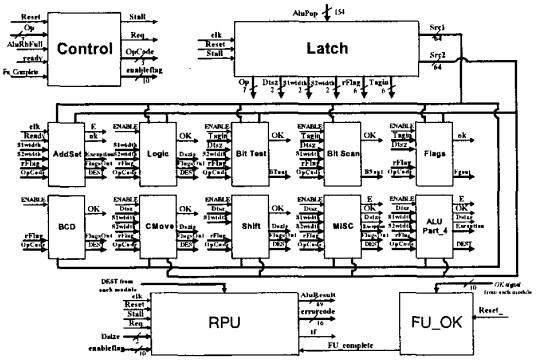


Fig. 3 : ALU0 Block Diagram

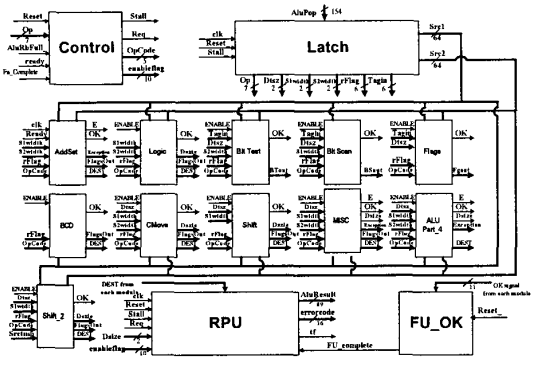


Fig. 4 : ALU1 Block Diagram

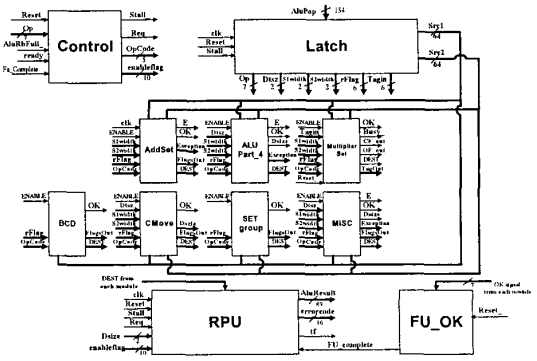


Fig. 5 : ALU2 Block Diagram

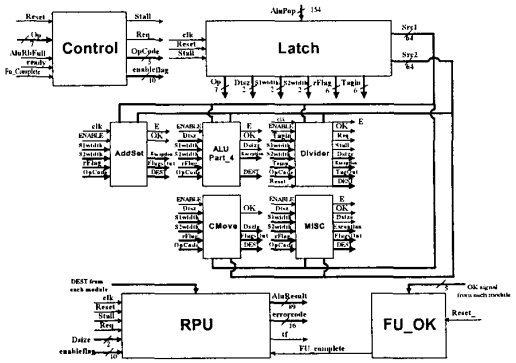


Fig. 6 : ALU3 Block Diagram

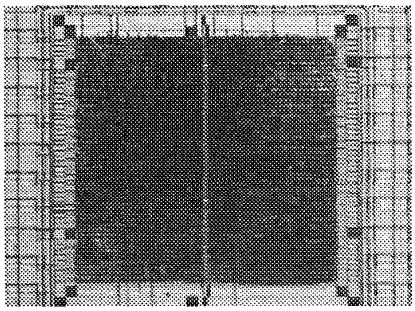

Fig. 7 : Die photo of the divider

| Module Name | 0.6um 1P3M | | 0.25um1P5M | |
|---|---|---|---|---|
| | Area (gate count) | Critical Delay | Area (gate count) | Critical Delay |
| ALU0 | 30028.75 | 18.30ns | 22858.25 | 5.08ns |
| ALU1 | 43358.57 | 19.33ns | 35915.50 | 11.56ns |
| ALU2 | 48469.56 | 65.80ns | 41801.53 | 33.72ns |
| ALU3 | 30512.07 | 67.15ns | 26804.50 | 27.67ns |

Table 1 : The performance comparison of ALU0 through ALU3 using different cell library.(wire load is set to 0.1 pF, and the load of each module is set to a maximum of 0.9 pF)

| ALU Design | Quantity | Area | clock freq. |
|---|---|---|---|
| 64-b 4-issue[1] | 8 | 482,500(transistors) | 180 MHz |
| 64-b 4-issue[13] | 2 | 643,000(transistors) | 200 MHz |
| 64-b 8-issue, ours | 4 | 122,840(gates) | 250 MHz |

Table 2 : The performance comparison of the other ALU designs and ours