Architectural Design Issues in a Clockless 32-Bit Processor Using an Asynchronous HDL

Myeong-Hoon Oh, Young Woo Kim, Sanghoon Kwak, Chi-Hoon Shin, and Sung-Nam Kim

As technology evolves into the deep submicron level, synchronous circuit designs based on a single global clock have incurred problems in such areas as timing closure and power consumption. An asynchronous circuit design methodology is one of the strong candidates to solve such problems. To verify the feasibility and efficiency of a large-scale asynchronous circuit, we design a fully clockless 32-bit processor. We model the processor using an asynchronous HDL and synthesize it using a tool specialized for asynchronous circuits with a top-down design approach. In this paper, two microarchitectures, basic and enhanced, are explored. The results from a pre-layout simulation utilizing 0.13-μm CMOS technology show that the performance and power consumption of the enhanced microarchitecture are respectively improved by 109% and 30% with respect to the basic architecture. Furthermore, the measured power efficiency is about 238 μW/MHz and is comparable to that of a synchronous counterpart.

Keywords: Clockless processor, asynchronous circuit, asynchronous HDL, mircoarchitecture, EISC, TiDE, Haste.

Manuscript received Sept. 7, 2012; revised Dec. 24, 2012; accepted Jan. 11, 2013.

Myeong-Hoon Oh (phone: +82 42 860 1654, mhoonoh@etri.re.kr), Young Woo Kim (bartmann@etri.re.kr), and Chi-Hoon Shin (cshin@etri.re.kr) are with the Software Research Laboratory, ETRI, Daejeon, Rep. of Korea.

Sanghoon Kwak (kwsh@sogang.ac.kr) is with the Department of Electronics Engineering, Sogang University, Seoul, Rep. of Korea.

Sung-Nam Kim (ksn@etri.re.kr) is with the IT Convergence Technology Research Laboratory, ETRI, Daejeon, Rep. of Korea.

http://dx.doi.org/10.4218/etrij.13.0112.0598

I. Introduction

While semiconductor process technologies and integration capabilities have been significantly improved, their design complexity and verification costs have rapidly increased. Today, most digital circuits are designed using a synchronous design methodology, which provides the advantage of easy implementation with well-established design infrastructures. However, this approach is facing several technical problems. As design styles that must include multiple timing domains are becoming more and more prevalent, it is difficult to synchronize data among domains using a single global clock. Moreover, glue logics tend to be sensitive to metastability or latency overhead, and the considerable amount of power consumed by a clock-related circuit is one of the critical limitations for designing low-power circuits. In addition, drastic radiation spectra from clock spikes cause electromagnetic problems, such as interference.

An asynchronous circuit synchronizes and transfers data with a handshaking protocol instead of using a global clock. In an asynchronous circuit, data transfers between adjacent modules are localized within the related modules only, and this localized timing leads to an average-case delay. Hence, theoretically, an asynchronous circuit can guarantee a higher performance than synchronous circuits in which the clock frequency is determined by the slowest component (worst-case delay) [1], [2]. In addition, the absence of a global clock in an asynchronous circuit design allows designers to make circuits without consideration of the timing closure through clock-tree balancing and to implement on-demand operations that are also helpful for low-power implementation. Although the asynchronous design methodology does not completely guarantee low-power circuits, it has been successfully applied

in certain applications for the reasons mentioned previously [3]-[5]. Furthermore, an asynchronous circuit provides better electromagnetic compatibility, which causes less interference when using it with a sensitive receiver [1] owing to the nonexistence of spikes by a global clock. In addition, the characteristic in which every module in an asynchronous circuit communicates with one another using a handshake protocol assures desirable portability and reusability.

Although asynchronous circuits have the above merits, there are some limitations, such as the generation of hazard control signals, testability, and a lack of commercialized asynchronous CAD tools. Owing to these limitations, it is generally not easy for designers who are accustomed to a synchronous design style to make a system fully asynchronous. This may make it difficult to meet the time-to-market of a system and can affect the productivity of circuit designers during the initial phase of an asynchronous style design.

Since 1990, however, many research groups have been studying the implementation of asynchronous circuits and their design methodologies to overcome such limitations. Some techniques for the implementation of hazard-free asynchronous circuits have been developed at Stanford University and Columbia University. Phillips has also developed its own solution covering the testability of asynchronous circuits and applied this solution to its commercial products. In the case of CAD tools, the University of Manchester and Columbia University have both contributed to the fundamental background for the design automation of asynchronous circuits.

Based on the results from such studies, a number of vendors have emerged in the industry that are relevant to asynchronous circuit designs, such as Achronix Semiconductor, Elastix, Tiempo, Fulcrum Microsystems, Camgian Microsystems, and Octasic. They each have their own design flows and verification methods and provide solutions for applications requiring asynchronous technology. In particular, Handshake Solutions, which was a spin-off company from Phillips, developed the world's first commercial asynchronous CAD tool, called "TiDE" [6]. TiDE is based on a high-level asynchronous hardware description language called "Haste" [7]. Both products have been verified using many real applications and were successfully utilized in the design of a commercial asynchronous ARM9, ARM996HS [8]. In addition to commercial asynchronous processors, numerous academic research groups, such as the Tokyo Institute of Technology [9], the University of Manchester [10], and CalTech [11], have developed various styles of asynchronous microprocessors, most of which are prototype designs.

In fact, asynchronous circuits are certainly extending their domain as a type of globally asynchronous locally synchronous structure with research on efficient asynchronous global signaling methods [12]. The international technology roadmap for semiconductors expects that circuits using asynchronous signaling will account for nearly 50% of the total design types by 2025 [13].

On the other hand, to prove the usefulness of the asynchronous design methodology, the selection of a target system is important. Processors are playing a crucial role in industry, leading to a significant increase in demand and a manifold growth in their areas of application. The processors are also key components in digital systems and provide an easy way for a comparison among several design methodologies. Therefore, we select the processor as a target system.

When designing a processor, employing an existing commercially verified instruction set architecture (ISA) is favorable for applications and a verification of the final results. The Extended Instruction Set Computer (EISC) ISA [14] from AD Chips [15] was designed to reduce the code size and frequency of memory access. The high code density characteristic of EISC ISA is suited to embedded applications in that it requires a small memory area [15].

In this study, we design an asynchronous 32-bit processor core: Asynchronous Low-power processor based on THe EISC Architecture (ALTHEA). ALTHEA is compatible with EISC ISA, using the TiDE tool. By achieving this and presenting quantitative analyses, our goal is to provide certain lessons that should be considered architecturally when designers implement an asynchronous pipelined processor, to hence show the feasibility of the asynchronous design methodology.

II. Design Background

1. Haste and TiDE Flow

Unlike synchronous design flows, most of which start from register transfer level specifications, TiDE accepts behavioral specification using Haste and hence allows designers to expend considerably less time and effort to make a circuit. However, a drawback when using Haste is that designers are not free from optimizing a circuit below the stage of the behavioral synthesis. The designers can only participate in describing the circuit with Haste, and it is very difficult to control the synthesis process at a fine-grain level because constraints are made manually. Nevertheless, TiDE is well known for being the most successful asynchronous design flow among language-based methods, owing to its ease of design and its stability for implemented circuits.

TiDE synthesizes a high-level specification described by Haste into an asynchronous gate-level netlist, as in [16], in which a compiler translates high-level languages into Verilog netlists. The base handshake protocol in TiDE is a four-phase

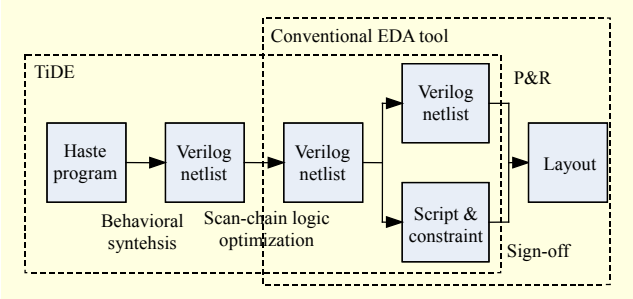


Fig. 1. TiDE tool flow.

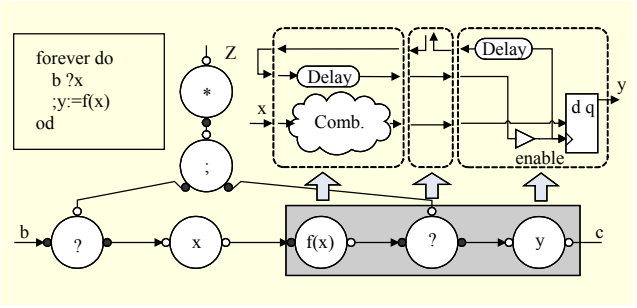


Fig. 2. Example of Haste code, handshake circuit, and their implementation.

bundled data assumption [17]. Figure 1 shows the design flow of TiDE. First, TiDE initially converts a Haste description into a corresponding Verilog netlist through a behavioral synthesis process. After the behavioral synthesis, TiDE optimizes the netlist by interacting with other conventional electronic design automation (EDA) tools. It also creates scripts and constraints to control the EDA tools to fabricate an ASIC from the netlist.

The behavioral synthesis process consists of generating handshake circuits and mapping them to standard cells. Handshake circuits are transformed from Haste descriptions using a syntax-directed compiling method [18]. The method converts every operator in a Haste specification into predefined components one by one. In the mapping step, the handshake circuits are synthesized into a netlist by mapping those components to standard cells according to a target technology. All components in TiDE are stored in a cell library DB targeting a certain technology, and each component has its own symbol, signal ordering, and schematic design. They are mainly used in the synthesis of control circuits and support various types of protocols based on a four-phase signaling. Representative components include a sequencer, parallel component, and repeater to respectively allow perpetual access in turn, simultaneously, and repetitively. In addition, TiDE has various components, such as a fork, join, merge, mux, and demux [6].

Figure 2 shows an example of a Haste code, its handshake circuit, and its implementation generated after synthesis. In the specification, the inner part between "forever do" and "od" iterates infinitely, and the operator "?" represents a channel input from left to right. That is, the data from channel "b" is stored into variable "x." The operator ":=" represents a variable assignment from right to left, that is, storing the result of the functional circuit into variable "y." This circuit consists of a repeater, a sequencer, and variable components.

In Fig. 2, the gray box in the handshake circuit represents a synthesis into the netlist shown in the blocks with the dotted lines. The functional circuit component includes a delay element to synchronize a data validation with an acknowledgement (ACK) signal. In addition, a variable component has a delay element to guarantee a hold time.

2. EISC Architecture

The EISC architecture processes 32-bit data with a native 16-bit instruction set. Because the EISC architecture utilizes 16-bit instructions, the density of the instruction code is quite high. According to the research in [15], the code density of the EISC architecture is 6.5% and 11.5% higher than that of the ARM THUMB and MIPS16, respectively.

Generally, a 32-bit reduced instruction set computer (RISC) processor using a 16-bit compressed instruction set requires a mechanism to deal with 32-bit immediate data. This occasionally causes inefficiency in making long-bit immediate values with small-bit opcodes because several memory access operations must be executed for a single operand of data. The EISC architecture has a special instruction, that is, a load extension register immediate (LERI), to minimize memory access. The LERI instruction consists of a 2-bit opcode and 14-bit immediate value fields. A 32-bit immediate value is composed of a maximum of three consecutive LERI instructions. There is no need for additional operations to form a 32-bit immediate value as in any other 32-bit processor with a 16-bit instruction set. This leads to 35.1% and 37.3% reductions in the data memory access rate compared to the ARM THUMB and MIPS16, respectively [15].

Another distinguishing feature of the EISC architecture with a 32-bit data bus and 16-bit instruction set is that the instruction fetch rate is twice that of the execution rate, on average, since two instructions are fetched in a single clock cycle.

These two typical characteristics lead to the generation of buffering instructions and immediate values before transferring them to the decode stage. In a pipelined structure with the EISC architecture, the instruction fetch stage can be decoupled from the decoding stage with an instruction queue.

Finally, another characteristic of the EISC architecture is as follows. The EISC architecture has 16 general-purpose registers, nine special-purpose registers, and three stack

pointers [14]. Most of the instructions are of RISC style, but some are of complex instruction set computer (CISC) style (for example, PUSH and POP). Their processing in ALTHEA will be presented later in this paper.

III. Basic Asynchronous Microarchitecture of ALTHEA Processor

1. Pipeline Stages

A five-stage pipeline with six functional blocks is the basic microarchitecture of ALTHEA, as shown in Fig. 3. The five stages are the instruction fetch (IF), decode (DE), execution (EX), memory (ME), and write back operations. Four of the five stages in the pipeline steps are in accordance with four functional blocks, and two additional blocks are a register file (RF) and coprocessor (CP).

All communication between blocks, as well as data transfers among subblocks in a functional block, is performed by an asynchronous protocol based on a four-phase bundled data assumption. Handshake channels separating each stage and block are depicted as simple data signals in Fig. 3. However, they accompany request (Sr, Rr) and ACK (Sa, Ra) signals of a sender and a receiver to control data transfers during implementation, as described in the lower section of Fig. 3.

Each functional block in the basic microarchitecture generates a valid or dummy output to send to its receiving blocks every cycle, regardless of the need for data transfers. By marking and delivering a valid bit added to a corresponding handshake channel, a sender indicates to the receiver whether the current transferred data is effective or not. This concept is similar to that of the data transfer between stages on a synchronous pipeline and is used for the basic architecture to ensure design ease.

The characteristics of the architectural viewpoint in the basic microarchitecture of ALTHEA can be summarized as follows:

- typical RISC style five-stage pipeline;
- Harvard architecture;
- asynchronous communication among blocks;
- decoupled IF stage;
- register file access through handshake channels;
- data transfer with valid bits similar to a synchronous one.

2. Operation of Each Block

The IF block consists of three functional units and two queues between them, the prefetcher and LERI folding unit fetches instructions. If the fetched instruction is an LERI, it collects consecutive LERI instructions to form a temporary instruction and store the instruction into a fetch queue. The

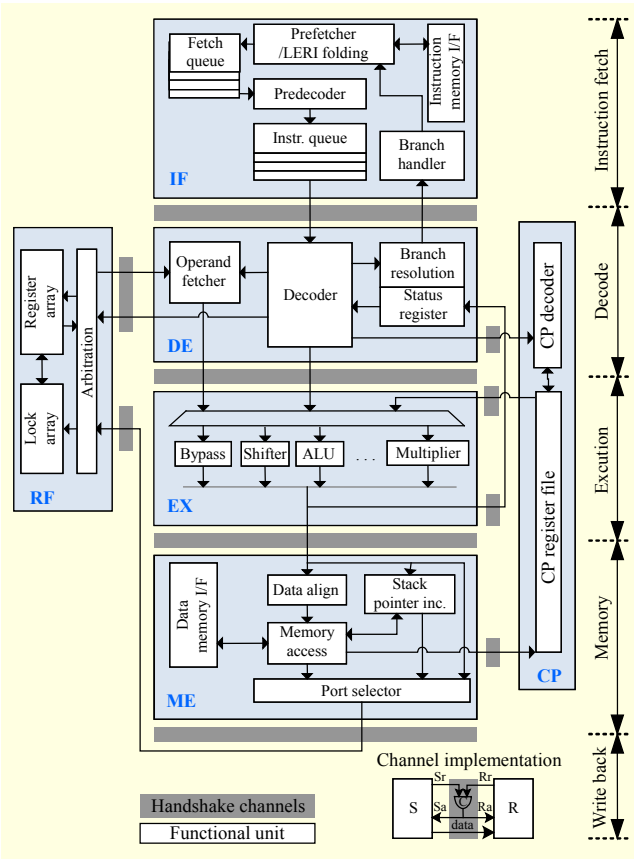


Fig. 3. Basic asynchronous microarchitecture of ALTHEA.

predecoder unit then reads a temporary instruction from the fetch queue, decodes it, generates a final instruction, and stores the instruction in the instruction queue. When informed that the instruction sent is a branch instruction or whether the IF block should be halted, the branch handler unit deals with the branch or the halting case.

The DE block performs three major functions: decoding the instructions from the IF block, accessing the register file, and preparing the operands. Each instruction from the IF block contains additional predecoding information that indicates one of 13 groups that the instruction belongs to. The DE block of ALTHEA has 13 individual subdecoders that are in accordance with each instruction group. Since a newly arrived instruction is decoded in only one of the subdecoders, other subdecoders are prohibited to operate. This reduces additional signal transitions and dynamic power consumption.

After the decoding process, access to register files and a generation of pipeline control commands are provided. The DE block sends the address of the source registers and destination lock information to the RF block. Finally, the operands and control information that are required from the EX and the ME block are generated.

The RF block has 16 general-purpose registers and nine

special-purpose registers. Unlike the register file designed using a synchronous method, there is no way to know when the correct value is written back to the register in an asynchronous register file. Note that the execution time of each pipeline stage in ALTHEA varies according to the type of instruction and whether there is data dependency. The RF block in ALTHEA provides a special locking mechanism [19] to prevent malfunctions in reading from and writing to the same register. When the DE block tries to apply a read access to a certain register, the operand fetch operation is postponed until the corresponding lock bit of the register is set to a valid state, meaning that the write access of the previous instruction to the same register is complete. The RF block has multiple read and write ports to access the registers simultaneously. Contrary to the EISC architecture, the RF block is separated from the DE block in ALTHEA to reduce the complexity of the DE block.

Arithmetic operations with operands are done in the EX block, which consists of several execution units, such as a shifter, an ALU, and a multiplier. According to the control signals from the DE block, only one execution unit is selected. In the case of a MOVE instruction group, data is bypassed to perform a simple register-to-register data transfer. Each execution unit chooses its operands among four handshake channels based on the valid bits located on the channels. Most of the instructions employ one or two operand values, but the STORE and MULTIPLY instruction groups take three and four operand values, respectively. The four-bit status values that reflect the final result of an execution are forwarded to the DE block through a handshake channel to handle conditional branches.

The ME block provides an interface with data memory to the LOAD or STORE instruction groups. In addition, automatic additive and subtractive calculations for a stack pointer, which indicates the memory address of the PUSH/POP instruction groups, are carried out in this block. Including a bypass route from the EX block, three final results of the ME block are conveyed to the RF block, accompanied by three individual handshake channels with valid bits.

Finally, the CP block supports the instruction to control a coprocessor and exchange state information. The CP block has peculiar but simple register files and functional units.

3. Performance Analysis of Each Block

Figure 4 shows the normalized performance of each block based on the standards of the best-case throughput in the CP block when the basic microarchitecture is designed and synthesized at the gate level. The performance of the CP block in which relatively simple operations are executed is the highest. The IF block also has a very short cycle time since the inner operations of the predecoding or LERI folding and the external channel output to the DE block run concurrently, owing to a four-stage instruction queue. However, these two blocks do not influence the performance improvement of the entire system. The occurrence rate of instructions related to a coprocessor is remarkably low, and the CP block does not work in accordance with the main pipeline. Furthermore, the IF block fetches instructions and stores them in the queue in advance until the queue becomes full, regardless of the pipeline states.

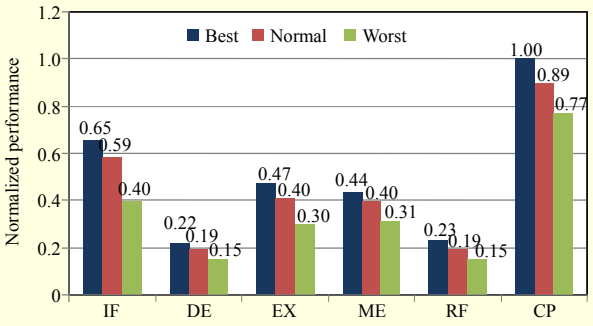


Fig. 4. Normalized performance of each block.

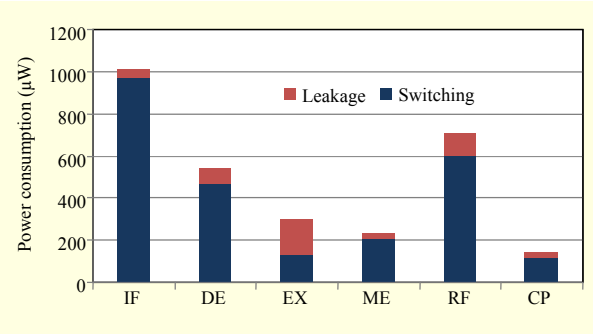


Fig. 5. Power consumption of each block.

On the other hand, the result indicates that the DE and RF blocks have low performances. Two major reasons for the performance degradation in the RF block are the register locking mechanism for resolving a data dependency and the arbitration between the read and write operations that occurs at a register file asynchronously. In the case of the DE block, the complexity of decoding the EISC instructions, additional channel communications for the register locking mechanism with the RF block, and dealing with branch instructions are the main reasons for the increased cycle time.

Clearly, the entire performance of the basic version of ALTHEA is vulnerable to this unbalanced pipeline structure in that the throughput of the pipeline totally depends on its slowest stage. The dispersion of power consumption, which

consists of static and dynamic features in each block, is shown in Fig. 5. The dissipated power of the IF block is the largest among all blocks because it always works on fetching instructions from memory, predecoding them, and managing an instruction queue as long as the queue is not full. The RF block, which most of the instructions access in reading the operands and writing the resulting data, consumes a considerable amount of power.

Comparing the DE block with the EX block, which performs real arithmetic operations, the dynamic power of the former is three times greater than the latter. It is inferred from this observation that the internal functions of the DE block must have a complicated design, making the occurrence of redundant activities possible. The amount of leakage power in each block is directly associated with its area. The EX block has the largest area among all the blocks.

The lesson learned from two prior evaluations with each block is to improve the DE and RF blocks so as to preferentially balance all stages. The next section explains this in detail.

IV. Enhanced Asynchronous Microarchitecture of ALTHEA Processor

1. Considerations for Enhancement

In the case of the DE block, complex decoding circuitries caused by an irregular opcode field of EISC ISA and intricate control schemes used to process a CISC-style instruction (for example, multi-operand PUSH/POP) are the major reasons for a degraded performance. In addition, many arbitrations among the read and write requests and the dependency resolution by the register locking mechanism have a decisive effect on the performance of the RF block. The complex structure also hinders a further reduction of the power consumption in the DE and RF blocks. In addition, channels unnecessarily communicating among blocks can lead to an increase in dynamic power. To raise the performance and reduce power consumption, an enhanced microarchitecture is developed and tested. Some major considerations for an enhanced version of the ALTHEA microarchitecture are summarized as follows:

- addition of a pipeline stage between IF and DE blocks to segment complex CISC instructions into simple multiple RISC instructions;
- separation of the DE block into two pipeline stages;
- simplified arbitration scheme for read and write operations in the RF block;
- on-demand handshake channels that only communicate with each other when necessary.

A block diagram of the enhanced microarchitecture of ALTHEA is depicted in Fig. 6, and details of the enhancements are explained in the following subsections.

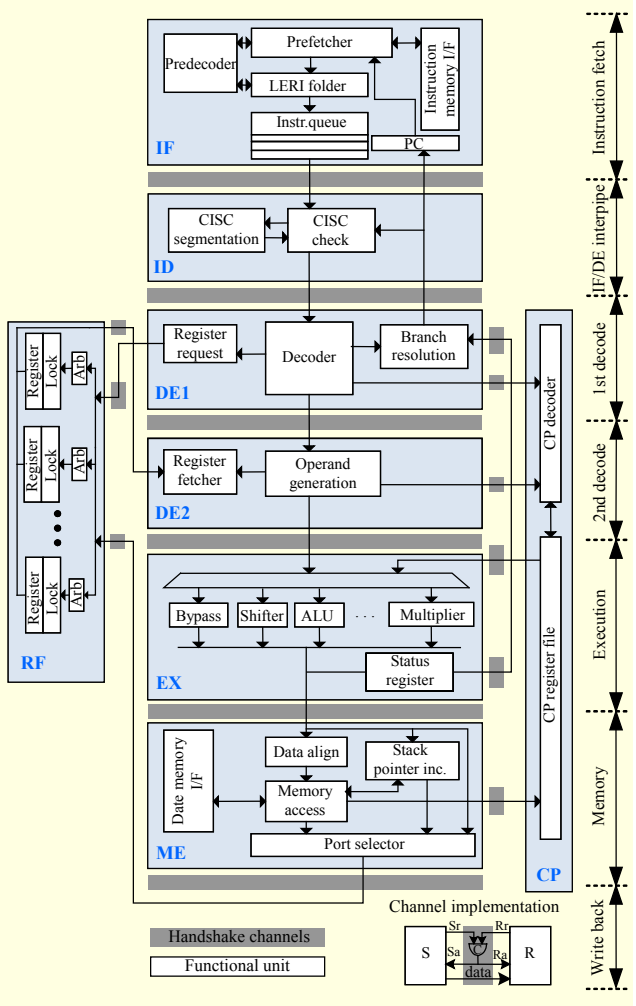


Fig. 6. Enhanced asynchronous microarchitecture of ALTHEA.

2. CISC Instruction Segmentation

The EISC instruction set contains some CISC-style instructions. These instructions require multicycle execution on a single instruction; thus, they need a finite state machine (FSM) to execute. In the basic microarchitecture, the FSM slows down the speed of the decoder because of added complexity in identifying CISC instructions and managing state transitions. To improve the performance related to the execution of CISC instructions, a dedicated pipeline stage, namely an ID (IF/DE inter-pipe) block, is added between the IF and DE blocks. The main purpose of the ID block is to identify a CISC instruction and segment the instruction into multiple single-RISC instructions with a single operand.

When the ID block receives a new instruction from the IF block, the ID block identifies the CISC instruction first. If the

Table 1. Example of CISC instruction segmentation.

Basic Microarchitecture	Enhanced Microarchitecture
PUSH %R00, %R01, %R02	PUSH %R00 // First of push PUSH %R01 // Middle of push PUSH %R02 // End of push
POP %R02, %R01, %R00	POP %R00 // First of pop POP %R01 // Middle of pop POP %R02 // End of pop

instruction is a normal RISC instruction, then it is forwarded to the next stage without any modifications. On the contrary, if the instruction is of the PUSH/POP instruction group, the ID block divides the instructions, as described in Table 1. Eliminating the FSM for CISC instructions in the DE block of the enhanced microarchitecture results in simple and fast operations for the decoding instructions.

3. DE Block Partition

As stated in the previous section, in the basic microarchitecture, three main functions (instruction decoding, RF access, and operand and control generation) are performed in the DE block. Among these major functions, the RF access process is longest, followed by the operand and control generation process. To improve the performance of the DE block, we divide the block into two. One is the DE1 block, which is used to decode instructions, request access to a register file, and generate control signals. The other is the DE2 block, which is used to obtain data from the register file and create operands. This separation can contribute to maintaining better balance in the pipeline by reducing the difference between stage latencies. Also, we can save additional time in the channel communications when accessing the register file by overlapping the processes of the RF block between the DE1 and DE2 blocks.

Figure 7 shows a comparison between the basic and enhanced microarchitectures with an example of a typical two-operand instruction. In this case, Inst0 is "ADD %R01, %R00." In the basic version, the three processes consume 17.2%, 50.5%, and 32.3% of the total execution time, respectively, and the execution time of Inst0 in the DE stage is about 62.7 ns, as shown in Fig. 7(a). On the contrary, Fig. 7(b) shows that the latency for Inst0 in the DE1 stage is remarkably reduced to 13.9 ns in the enhanced version, owing to a large decrease in the time for RF access. Actually, the ratio of the time consumption for the RF request becomes only 24.6%. See the ratio of time for RF access (50.5%) in the basic version.

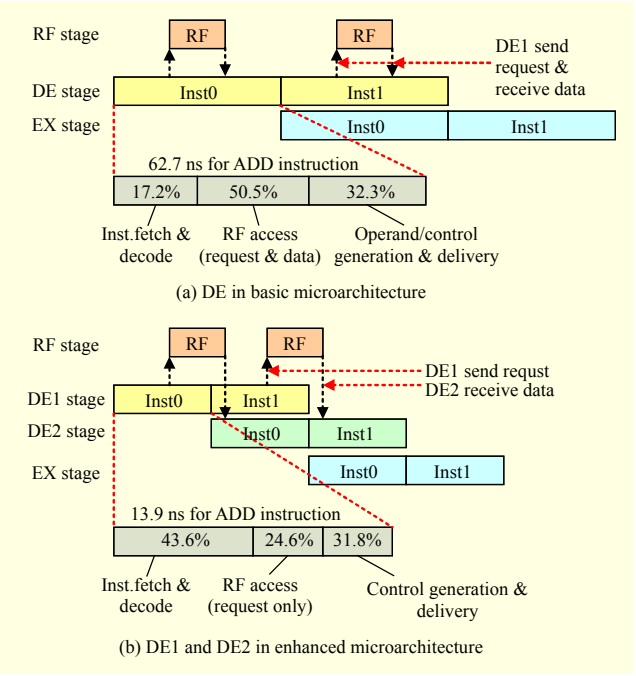


Fig. 7. Enhancement in DE block.

4. Structural Enhancement of RF Block

Because of its complexity in dealing with the generation of multiple operands and the prevention of data hazards in the pipeline, the register file is also one of the performance bottlenecks in a basic microarchitecture. In EISC ISA, the maximum required number of read operands is four in the case of a multiply instruction. In a write back stage, two 32-bit write data ports are required to store 64-bit multiplication data, and one write data port indicates a stack point for the execution of multi-operand PUSH/POP instructions.

In a basic microarchitecture, registers in the RF block are designed in the form of a global register array using the Haste keyword "ramreg array." A conceptual block diagram representing the keyword "ramreg array" is shown in Fig. 8(a). The register array supports only one read operation or one write operation at a given time. Therefore, all read access operations and all write access operations should be serialized. Indeed, arbitration circuits between read and write requests cannot help being complex for correct operations since read and write requests are generated regardless of the DE and ME stages, respectively. Finally, if one of the read requests has an unresolved data dependency, the rest of the requests to the RF block are suspended until the data dependency is solved because all requests are serialized.

To overcome these obstacles, we redesign the RF block in a manner in which a single global array is changed into 25 individual 32-bit registers and lock registers, as shown in

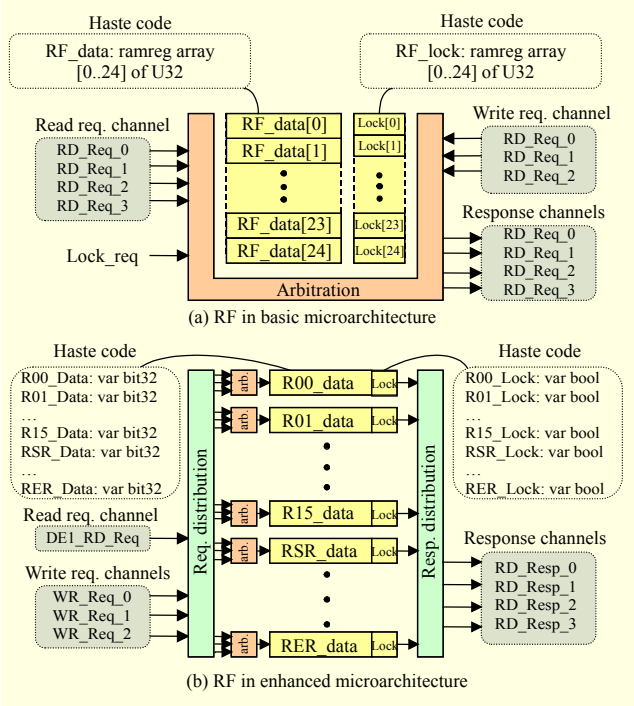


Fig. 8. Enhancement in RF block.

Fig. 8(b). By separating the registers, any register can be independently and simultaneously accessed by read or write requests as long as there is no data dependency. Performance degradations owing to a pause in RF access, caused by serializing requests, are soothed.

Another modification in the RF block is that all read requests are merged into a single handshake channel. Separated handshake channels in a basic microarchitecture cause a sophisticated arbitration circuitry and slow down the access time. By removing arbitrations for read requests and separating the registers, the access time can be reduced.

5. On-Demand Channel

Data transfers between blocks through handshake channels in a basic microarchitecture are done in such a way that a receiver block unconditionally latches onto input data at every handshake communication, whether it performs operations with the data or not. The receiver block distinguishes the effective input data using the value in the separate valid bit on each handshake channel. Therefore, both the sender block and the receiver block must perform meaningless channel communications in cases of invalid transfers. This operation has a baleful influence on both the performance and power consumption of ALTHEA. In other words, by activating the channels only when data transfers are required, we can reduce unnecessary access and contribute to a faster and low-powered ALTHEA.

Table 2. Example of channel description.

	Basic microarchitecture	Enhanced microarchitecture
DE or DE2	If (Opa and Opb condition) valida = 1 || validb = 1 || validc = 0 || validd = 0 . . . . ; Opa ! {valida, opa_reg} || Opb ! {validb, opb_reg} || Opc ! {validc, 32'h00000000} || Opd ! {validd, 32'h00000000}	If (Opa and Opb condition) Opa ! opa_reg || Opb ! opb_reg
EX	Opa_ex ? opa || Opb_ex ? opb || Opc_ex ? opc || Opd_ex ? Opd ;if (operanda and b is needed) if (Opa_ex[32]==1) operanda = Opa_ex[31:0] || if (Opb_ex[32]==1) operandb =Opb_ex[31:0]	If (operanda and b is needed) operanda ? Opa || operandb ? Opb

Table 2 explains this on-demand style of channel communication as an example of data transfers between the DE and EX blocks. For some input channels (Opa, Opb, Opc, and Opd) of the EX block, Haste-based pseudocodes describe the behaviors of the DE and EX blocks according to both microarchitectures in executing an instruction using two operands, operanda and operandb. In the DE block of a basic microarchitecture, at the checking step of the output conditions of both Opa and Opb, valid values of each channel (valida, validb, validc, validd) are set to high or low. All channels containing each valid bit are activated simultaneously and latched onto an internal register (Opa_ex, Opb_ex, Opc_ex, Opd_ex) in the EX block. Finally, it uses operand data values (operanda, operandb) by checking the valid bits. However, in the enhanced microarchitecture, only the required operands are transferred without communication of Opc and Opd.

The on-demand-style channel is more favorable for fast and low-powered communication between the DE and EX blocks. In fact, all the instructions in ALTHEA except a multiply instruction group use one or two operands, and it is thus highly reasonable to prevent all channel communication from occurring, especially Opc and Opd.

V. Design and Simulation

1. Design Flow and Simulation Environment

The left side of Fig. 9 illustrates the design flow of ALTHEA

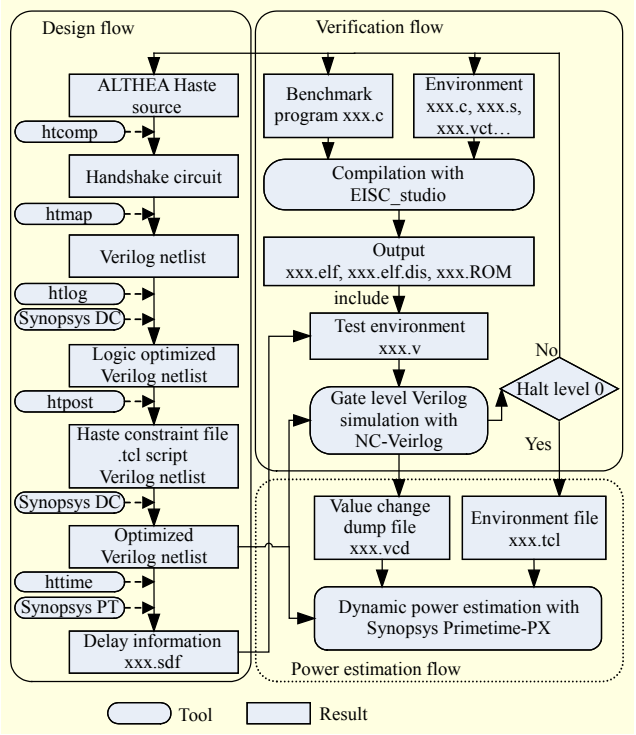


Fig. 9. Design, verification, and power estimation flow of ALTHEA.

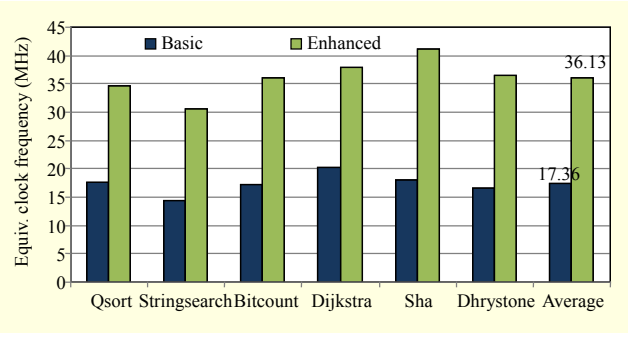


Fig. 10. Performance comparison using various benchmarks.

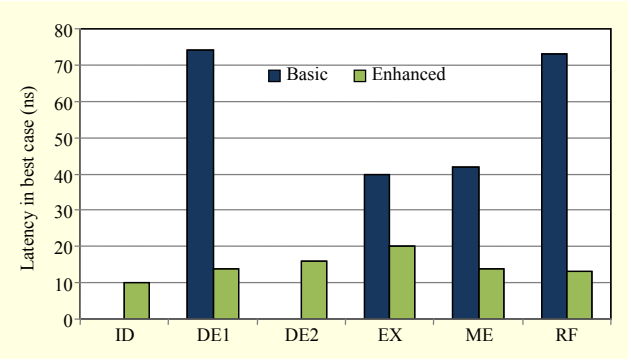


Fig. 11. Latency comparison of each block.

based on TiDE. The initial source codes described with Haste are compiled into handshake circuits and converted to gate-level netlists by htcomp and htmap commands, respectively. To optimize the netlists, TiDE supports htlog and htpost commands, which are used to generate partially compact netlists in terms of area and several constraint files to operate together with standard EDA tools. Finally, we obtain an integrated netlist at a pre-layout level with Synopsys Design Compiler. Also, standard delay format files are generated by an httime command and Synopsys PrimeTime for a more realistic timing simulation. The employed technology is a UMC 0.13-μm CMOS process.

The verification environment of ALTHEA at a pre-layout level is shown on the right side of Fig. 9. Using a compiler of the EISC architecture, we generate ROM codes of the corresponding benchmark programs of MiBench [20] and Dhrystone [21] suite and insert the binary codes into a Verilog test code for instantiating the optimized Verilog netlist from the TiDE tool flow. To confirm the functionality, we check the level of HALT instruction at the timing simulation with a Cadence NC-Verilog simulator after inserting test codes that generate the zero halt level at the end of each benchmark program. The bottom of Fig. 9 shows the flow used to estimate dynamic power consumption at the gate level. We also verify our designs on FPGA by using the verification flow that the TiDE tool provides [6].

2. Simulation Results

To analyze the effect of architectural considerations in terms of performance and power consumption, we measure the quantitative results of the basic and the enhanced versions of ALTHEA.

Figure 10 shows the equivalent clock frequencies of each version in executing various benchmark programs based on the standards of the maximum clock frequency of synchronous AE32000.

In the enhanced version, a division of the DE block of the basic version into three blocks (ID, DE1, DE2) and a modification of the RF block clearly reduces the latencies of the DE and RF blocks. Although two additional handshake channels are added from the ID block to the DE2 block, this improvement in the structural level leads to a 109% performance increase on average compared with the basic version. Actually, as in Fig. 11, which shows the latency in each block of both versions, all blocks in the enhanced version have a shorter time, and, in particular, the latency of the DE block, which is a bottleneck in the performance, decreases greatly.

The EX block has the largest latency (20 ns) in the enhanced version, so the maximal throughput is around 50 MHz (1/20 ns) by rule of thumb with an ideal assumption of a

Table 3. Power comparison.

Version	Power consumption (mW)		
	Best	Normal	Worst
Basic	16.5	7.9	3.9
Intermediate	6.9	3.5	1.9
Enhanced	11.6	5.9	3.0

pipeline structure. However, the average measured performance, as shown in Fig. 10, is about 36 MHz, and the difference between the maximal expectation and real observation seems to be large. The major cause for this gap is that the enhanced version resolves data dependency problems by stalling the pipeline with the register locking mechanism.

Before making a power comparison between two versions of ALTHEA, we measure the power consumption of the "intermediate" version, which adopts only on-demand channels and the ID stages to evaluate the power impact of the architectural considerations described in section IV. Table 3 lists the power measurement for the three versions regarding the three corner conditions. The reduction ratio of the power consumption in the intermediate version compared with the basic version is nearly 60%. Changing all handshake communications into on-demand channels can significantly save dissipated power during unnecessary data transfers between blocks. Furthermore, the logics of the DE block are simplified by removing operations related with the PUSH/POP group, and non-PUSH/POP instructions can bypass the ID stage.

The reason for a greater power consumption in the enhanced version compared with the intermediate version is that the RF block becomes more complex, and the switching activity of the blocks and channels increase at the same rate as the performance improvement. However, the enhanced version still reduces 30% of the power consumption compared with the basic version. Accordingly, it is found that the enhanced version has the best power efficiency, and its value is calculated at 238 μW/MHz for a normal case.

A commercialized synchronous AE32000 was fabricated using 0.18-μm CMOS technology, and its maximum working clock frequency is 140 MHz [15]. It was also reported that the simulated power efficiency of the AE32000 core without a cache is about 400 μW/MHz. Although a fair comparison is impossible owing to the different technologies, it is probable that the ALTHEA of the enhanced version is competitive with its synchronous counterpart in terms of power efficiency. As for the performance, we can apply a data forwarding mechanism for solving data dependencies, as in normal synchronous pipelined processors. Thus, we are expecting that ALTHEA's maximum equivalent clock frequency can reach or surpass that of ARM966HS (77 MHz) [8], which is a 32-bit asynchronous commercial ARM processor and was designed using the same design flow of TiDE.

VI. Conclusion and Future Work

In this paper, we designed and verified a 32-bit clockless processor core (ALTHEA) at a pre-layout level using an asynchronous HDL and a top-down design and verification flow.

Based on the basic microarchitecture, which consists of six blocks for a five-stage pipeline, we tried to ascertain any problems and suggested their solutions in terms of an architectural view. Finally, we designed an enhanced version of ALTHEA based on a seven-stage pipeline structure, after applying several ideas.

Simulation results show that the performance and power consumption of the enhanced version increases by 109% and decreases by 30%, respectively. The measured power efficiency of ALTHEA is about 238 μW/MHz, which is a value competitive with that of a synchronous EISC processor.

For our future work, we will try to make pipeline stages more harmonious to reduce the latency difference among all stages and employ a simple forwarding scheme to resolve data hazards efficiently. We will also consider applying a bottom-up-based design methodology in which designers can get involved in all of the design stages, unlike TiDE tool flow.

References

[1] C.H. Van Bekel et al., "Applications of Asynchronous Circuits," *Proc. IEEE*, vol. 87, no. 2, Feb. 1999, pp. 223-233.

[2] C.J. Myers, *Asynchronous Circuit Design*, NY: John Wiley & Sons, Inc., July 2001.

[3] J. Kessels and R. Marston, "Designing Asynchronous Standby Circuits for a Low-Power Pager," *Proc. IEEE*, vol. 87, no. 2, Feb. 1999, pp. 257-267.

[4] B.Z. Tang et al., "A Low Power Asynchronous GPS Baseband Processor," *Proc. IEEE 18th Int. Symp. Asynchronous Circuits Syst.*, 2012, pp. 33-40.

[5] S. Bo et al., "Reducing Power Consumption of Floating-Point Multiplier via Asynchronous Technique," *Proc. 4th Int. Conf. Comput. Inf. Sci.*, Aug. 2012, pp. 1360-1363.

[6] Handshake Solutions, TiDE Manual, 2009

[7] Handshake Solutions, Haste Manual, 2009

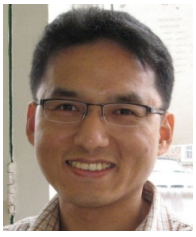
[8] A. Bink and R. York, "ARM996HS: The First Licensable, Clockless 32-Bit Processor Core," *IEEE Micro*, vol. 27, no. 2, 2007, pp. 58-68.

[9] A. Takamura et al., "TITAC-2: An Asynchronous 32-bit Microprocessor Based on Scalable-Delay-Insensitive Model," *Proc. IEEE Int. Conf. Computer Design*, Oct. 1997, pp. 288-294.
[10] J. Garside et al., "AMULET3 Revealed," *Proc. IEEE Int. Symp. Adv. Research Asynchronous Circuits Syst.*, Apr. 1999, pp. 51-59.
[11] A. Martin et al., "Three Generations of Asynchronous Microprocessors," *IEEE Design Test Computers*, Nov. 2003, pp. 9-17.
[12] M.-H. Oh and S. Kim, "Asynchronous 2-Phase Protocol Based on Ternary Encoding for On-Chip Interconnect," *ETRI J.*, vol. 33, no. 5, Oct. 2011, pp. 822-825.
[13] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors*, 2011.
[14] Advanced Digital Chips Inc., *Instruction Set Reference Manual for AE32000: a 32-bit EISC microprocessor*, Nov. 2008.
[15] H. Lee, P. Beckett, and B. Appelbe, "High-Performance Extendable Instruction Set Computing," *Proc. 6th ACSAC*, Jan. 2001, pp. 89-94.
[16] G.N.T. Huong and S.W. Kim, "GCC2Verilog Compiler Toolset for Complete Translation of C Programming Language into Verilog HDL," *ETRI J.*, vol. 33, no. 5, Oct. 2011, pp. 731-740.
[17] S.B. Furber and P. Day, "Four-Phase Micropipeline Latch Control Circuits," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 4, no. 2, June 1996, pp. 247-253.
[18] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design — A Systems Perspective*, Norwell, MA: Kluwer Academic Publishers, 2001.
[19] N.C. Paver et al., "Register Locking in An Asynchronous Microprocessor," *Proc. IEEE Int. Conf. Computer Design*, Oct. 1992, pp. 351-355.
[20] M.R. Guthaus et al., "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," *IEEE Int. Workshop Workload Characterization*, Dec. 2001, pp. 3-14.
[21] R. Weicker, "Dhrystone: A Synthetic Systems Programming Benchmark," *Commun. ACM*, vol. 27, no. 10, Oct. 1984, pp. 1013-1030.

Myeong-Hoon Oh received his PhD in information and communications engineering from Gwangju Institute of Science and Technology (GIST), Gwangju, Rep. of Korea, in 2005. He has been with ETRI, Daejeon, Rep. of Korea, since 2005 as a senior engineer. From 2006 to 2011, he was an associate professor at the University of Science and Technology (UST), Daejeon, Rep. of Korea. His current research focuses on digital circuit design, relevant embedded systems, cloud computing hardware design, and cloud computing standardization. He also has been an editor for developing the Recommendation of Cloud Desktop as a Service in ITU-T SG13.

Young Woo Kim received his BS, MS, and PhD in electronics engineering from Korea University, Seoul, Rep. of Korea, in 1994, 1996, and 2001, respectively. He was an associate professor at the University of Science and Technology, Daejeon, Rep. of Korea, during 2009 and 2010. In 2001, he joined ETRI, Daejeon, Rep. of Korea. His recent research has focused on asynchronous processors and computer system development. His current research interests are high-speed networks and supercomputing system architecture.

Sanghoon Kwak received his BS in computer engineering from Dongguk University, Seoul, Rep. of Korea, in 1998. He received his MS and PhD in information and communications engineering from GIST in 2000 and 2009, respectively. He is working as a research professor in the Department of Electronics Engineering, Sogang University, Seoul, Rep. of Korea, since Oct. 2011. His current research interests include network-on-chip synthesis in 3D ICs and design methodology for GALS (globally asynchronous and locally synchronous) systems.



Chi-Hoon Shin received his BS in computer science from Ajou University, Rep. of Korea, in 2005, and MS and PhD in computer science from the University of Science and Technology (UST), Daejeon, Rep. of Korea, in 2007 and 2011, respectively. He joined ETRI, Daejeon, Rep. of Korea, in 2011, where he is currently a senior researcher. His major interests include computer architecture, asynchronous circuits, sensor networks, and algorithms. He is a member of IEICE and IEEE.

Sung-Nam Kim was born in Chungnam, Rep. of Korea, in 1968. He received his BS, MS, and PhD in electronics engineering from Korea University, Seoul. Rep. of Korea, in 1991, 1993, and 1998, respectively. In 1999, he joined ETRI, Daejeon, Rep. of Korea, where he is currently a principal researcher. His current research interests include cloud computing hardware platform architecture, high-speed interconnect technology, low-power digital circuit design, and asynchronous circuit design technology.