4-WAY SUPERSCALAR DSP PROCESSOR FOR AUDIO CODEC APPLICATIONS

*Joon Seok Kim** *Sun Kook Yoo*** *Sung Wook Park** *Nam Hoon Jung**
*Woo Suk Ko** *Keun Sup Lee** *Dae Hee Youn**

*A.S.S.P. Laboratory, Department of Electronic Engineering ,Yonsei University
**Department of Medical Engineering, Yonsei University
mania@cyclon.yonsei.ac.kr

ABSTRACT

The recent audio CODEC (Coding/Decoding) algorithms are complex of several coding techniques, and can be divided into DSP tasks, controller tasks and mixed tasks. The traditional DSP processor has been designed for fast processing of DSP tasks only, but not for controller and mixed tasks. This paper presents a new architecture that achieves high throughput on both controller and mixed tasks of such algorithms while maintaining high performance for DSP tasks. The proposed processor, YSP-3, operates four functional units (Multiplier, two ALUs, Load /Store Unit) in parallel via 4-issue super-scalar instruction structure. The performance evaluation of YSP-3 has been done through the implementation of the common DSP algorithms and AC-3 decoder.

1. INTRODUCTION

The audio CODEC algorithm (ex. MPEG-2[1] or AC-3[2]) can be divided into three parts; (1) Controller tasks: parsing bit-stream, setting the operating mode and communicating with other systems; (2) DSP tasks: filtering, multiplying matrixes, and various transforms; (3) Mixed tasks: computing masking curves based on psycho-acoustic model, searching tables, extracting scale-factors and so on. The DSP tasks are composed of predetermined number of repetition on MAC computation. But the controller tasks and the mixed tasks are composed of many run-time determined procedures requiring simple computations (e.g. bit-wise logical operation, a multiplication, an addition, and a shift).

Considering the processing load of the CODEC algorithm, (2) takes the half, and (1), (3) take the other half [3]. In this point of view, the traditional DSP processors are inadequate for such CODEC application even though they show good performance in DSP tasks. The reason is that they have low performance in the controller and the mixed tasks. Their low throughput results from the fact that functional units (FU's), such as multiplier, ALU, barrel shifter, are connected in serial way. They are designed in such a way to process repetitive MAC computation fast and efficiently because the core of DSP algorithms is the repetition of MAC. That is, they can't fully utilize all of their FU's except when they execute MAC instruction.

To resolve the problem of the traditional DSP processor, and maintain their high performance for the DSP tasks, there have been many works [4][5][6]. In those works, the performance in controller and mixed tasks is enhanced, however, the problem of low efficiency and flexibility in utilization of their functional units exists yet. To resolve all of the problems, we connect FU's in parallel way, and let each unit process one instruction in parallel. We have decided four FU's, multiplier, two ALUs, and Load/Store unit, therefore, four instructions can be executed per one cycle. In this architecture, the proposed processor functions as a 4-issue RISC processor for the controller and the mixed tasks, which result in the better performance than a conventional DSP processor without degrading the performance for the DSP tasks.

While VLIW instruction structure is very good approach to this processor, the main disadvantage of that is the instruction code size explosion when a program does not offer enough parallelism at the instruction level [7][8]. Another approach is a super-scalar scheme. Though the issue process of the super-scalar scheme has overhead expense, it can be simple and low-cost task if we choose in-order issue and in-order completion policy with good arrangement of instructions at compile time [9].

2. ARCHITECTURE

The 4 way super-scalar DSP processor, YSP-3, has 3 pipeline stages.

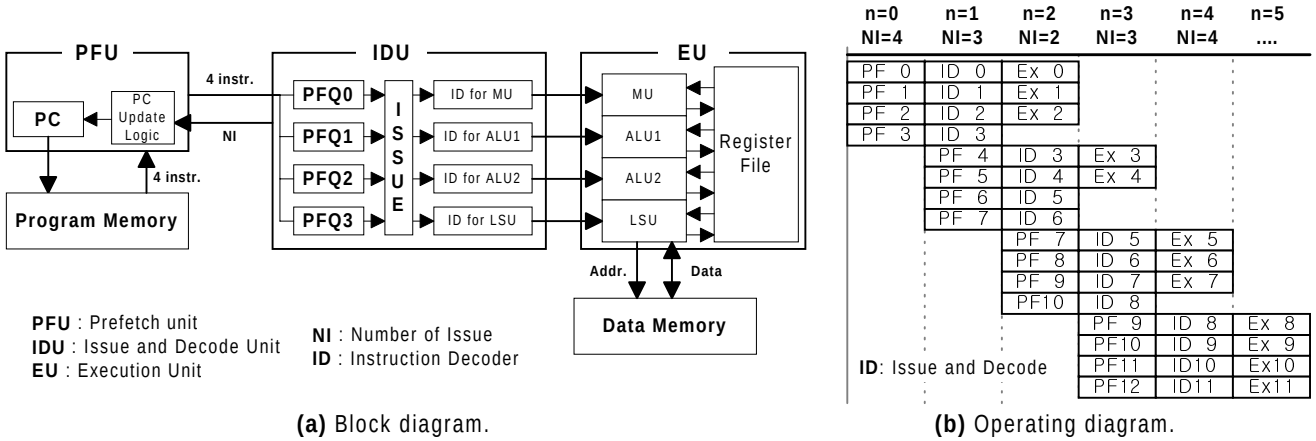
- ***Pre-Fetch***: Fetches the 4-instruction words from memory into PFQ's (pre-fetch queues) and updates program counter.
- ***Decode and Issue***: Decides the number of instructions that could be executed simultaneously (issue) among PFQ's and decodes them (decode).
- ***Execute***: Executes the issued instructions.

In order to operate 3 pipeline stages in parallel, YSP-3 has 3 independent units for each stage: Pre-Fetch Unit (PFU), Issue and Decode Unit (IDU), and Execution Unit (EU). Figure 1 illustrates overall architecture and pipeline operation of YSP-3.

2.1 Execution Unit

As mentioned previously, EU has 4 FU's: Multiplier unit (MU), ALU1, ALU2, and Load/Store Unit (LSU). Figure 2 shows the FU's in EU.

- MU multiplies the two 16-bit inputs, and carries the product to 32-bit result bus.



(a) Block diagram. **(b)** Operating diagram.

Figure 1. Overall architecture and operation.

- The 40-bit ALU1 is up to arithmetic and logical operations: addition/subtraction, increment, negation, conversion to absolute value, AND, OR, XOR, and NOT. In addition to the basic operations, ALU1 provides several special operations; (1) Arithmetic shifting: ALU1 has 40x16 bit barrel shifter; (2) Parsing bit-streams: ALU1 has bit-unpacking module to parse the bit-stream fast, because the form of compressed (or transmitted) data in CODEC algorithm is bit-stream. The module can parse the 1~16 bits per one cycle from 1k-byte internal FIFO, which is the temporal storage for input bit-stream. (3) Rounding: RND instruction is supported. It can be used to reduce the bit-width of the data when the 32-bit data is stored in 16-bit storage. With round-to-nearest-even scheme, YSP-3 statistically minimizes the errors of 40-bit fixed-point arithmetic [10].
- LSU accesses all data memories, which consist of internal memory (16-bit and 32-bit RAM, ROM) and external memory (16-bit RAM only). As the traditional DSP processor, it has the post-modifiable address generator providing linear, circular and bit-reverse addressing modes.
- ALU2 has the same 40-bit ALU as ALU1 to perform arithmetic and logical operations. Additionally, it has a special privilege to read a data from the internal data ROM. In case of DSP tasks, especially convolution (e.g. FIR, IIR filtering), a data sample in the data RAM and a filter coefficient in the data ROM are fetched and multiplied. YSP-3 has separate data and address buses for both RAM and ROM, the parallel access to them is possible to MAC without any data latency. While LSU has the higher priority in access the data ROM, ALU2 takes it up when LSU is busy for RAM. Post-modification of address is also provided using its own ALU.
- All FU's are designed to move an immediate data given by instruction to register file.
- Register File is composed of thirteen 16-bit General Purpose Registers (R0-R12), four 32-bit Double Word Registers (DWR0-DWR3), four 40-bit Accumulators (ACC0-ACC3), four 14 bit Address Registers (AR0-AR3), and two 14 bit Modify Registers (MR0, MR1). The DWR can be split into an upper and a lower 16-bit part, so it holds two 16-bit words or one 32-bit double precision data. AR is used to hold data address, and MR holds the value to modify AR. When AR and MR are used in pair for indirect addressing mode, selected AR is post-modified by MR. Every register except AR and MR in the register file has 4 write ports and 7 read ports. AR has 8 read and 5 write ports, and MR has 8 read and 4 write ports. If any input of FU's wants more bits than bit-width of selected register, then sign-extension is occurred (e.g., one input to the ALU1 is 16-bit GPR, then it is sign-extended to 40 bits).

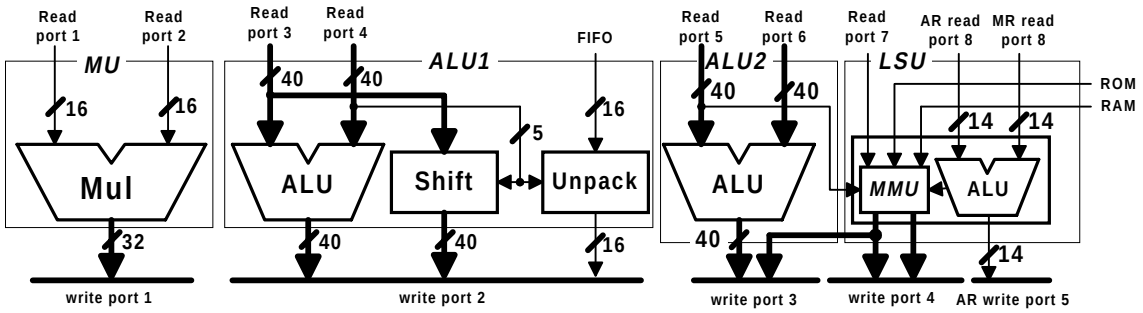


Figure 2. Execution unit.

2.2 Pre-fetch Unit and Issue-and-Decode Unit

For simplicity, we choose in-order issue and in-order completion policy, though in-order issue policy has lower issue rate than out-of-order. However, if instructions are arranged at compile time to avoid conflicts and dependencies, it can have the same issue rate as out-of-order policy. If instructions are issued in order, in-order completion is guaranteed because all FU's are designed to perform in a single cycle.

In PFU, PC (Program Counter) holds the first address of four instructions to be fetched. PFU fetches 4 instructions from @(PC) to @(PC+3), and stores them in PFQ0-PFQ3. The PC at n'th cycle is

$PC(0) = 0,$
$PC(n+1) = PC(n) + NI(n)$

, where NI(n) given by IDU is the number of issued instructions at n'th cycle. When one of the program flow control instructions, such as CALL, RTN(Return), JUMP, BR(branch), and LOOP is issued, NI(n) is set to 4.

IDU issues from PFQ0 to PFQ3 sequentially (in-order). It checks the resource conflict which arises when two instructions must use the same resources at the same time (ex. two of four instructions require multiplier unit) and true data dependency which arises when an instruction uses a value produced by a previous instruction. IDU meets any conflict or dependency, it stops issuing immediately and then assigns issued instructions to FU's.

Instruction has 25 bit width and is divided into five sub-fields, (1) op-code field (5-bit), (2) destination register field (5-bit), (3) memory addressing mode field (2-bit), (4) 2'nd source register field (5-bit), and (5) 1'st source register/AR and MR pair field (8-bit). In YSP-3, the resource conflict is checked by op-code field and data dependency by 5-bit destination field and 13 bit 1'st and 2'nd source operands fields.

The instructions that cause abrupt change of PC, for example, JUMP, BR (conditional jump), CALL, RTN (return), and LOOP is always issued solely to discard the instructions in PFQ's. For PFU has the all resources to control the program flow control (PC, PC Stack, Loop Counter, and Loop Stack), it supervises and manages the program flow by modifying it's resources. For branch, YSP-3 provides three branch prediction strategies by following instructions: BRT (predict branch would be taken), BRN (predict branch would not be taken), BRD (delayed branch). While BRT or BRN may generate one-cycle latency in case of failure on branch prediction, BRD removes that latency. Lastly, zero latency loop capability is provided.

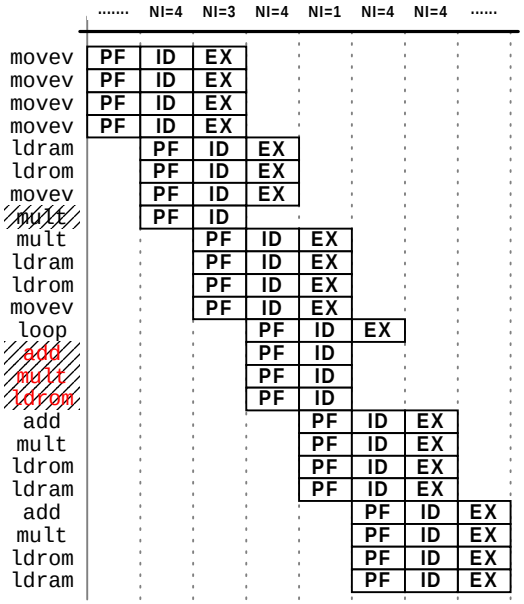
3. OPERATION

Figure 3 illustrates program and execution for a 64-tap FIR filtering with coefficient h(i), input x(n) and output y(n), which is a typical example for digital signal processing applications. For this operation, all units, MU, ALU1, ALU2, and LSU, work as follows:

(1) An address pointer ar0 is used to load the data elements, x(n) from the data RAM into r0.; (2) an address pointer ar1 is used to load the coefficients, h(i) into r1; (3) ar0, ar1 are post-incremented using the value of mr0. LDRAM is executed in LSU,

```
movev  ar0, ^sample    ; ar0= ptr. of x(n)
movev  ar1, ^coeff     ; ar1= ptr. of h(i)
movev  ar2, ^output    ; ar2= ptr. of y(k)
movev  mr0, #1         ; mr0= post-increment
                       ; value

ldram  r0, (ar0, mr0) ; r0= x(0)
ldrom  r1, (ar1, mr0) ; r1= h(0)
movev  acc1, #0       ; clear acc1

mult   acc0, r0, r1   ; acc0=x(0)*h(0)
ldram  r0, (ar0, mr0) ; r0= x(1)
ldrom  r1, (ar1, mr0) ; r1= h(1)
movev  cntr, #62      ; repeat 62 times

loop   :end           ; following 4 instr.

 add  acc1, acc1, acc0; acc1 = acc1 + acc0(n)
 mult acc0, r0, r1    ; acc0 = x(n)*h(i)
 ldrom  r1, (ar1, mr0); r1 = h(i++)
end:
 ldram  r0, (ar0, mr0); r0 = x(n++)

add  acc1, acc1, acc0 ; acc1=acc1+acc0(62)
mult acc0, r0, r1     ; acc0=x(63)*h(63)

add  acc1, acc1, acc0 ; acc1=acc1+acc0(63)

shift  acc1, acc1, #8 ; bit-width adjustment
                      ; 40 bit -> 32 bit

rnd    r0, acc1       ; round(32bit->16bit)

store  r0, (ar2, mr0) ; y(k++)=r0
```

(a) 64-tap FIR filter code.



(b) Pipeline diagram (The shade means the instruction in it can not be executed concurrently).

Figure3. FIR filter code and pipelined execution.

and LDROM in ALU2, as mentioned 2.1.; (4) r0 and r1 are multiplied into 40-bit acc0 in MU; (5) ALU1 accumulates acc0 into acc1.

Thus, the 64 times MAC computation is executed in 67 cycles using pipeline manner. The each empty line in the Figure 3.(a) isolates the instructions that can be issued together.

4. IMPLEMENTATION DETAILS AND PERFOMANCE

The YSP-3 has been described in VHDL and synthesized using a standard cell of 0.6 um-3ML CMOS technology. The total gate count of YSP-3 is 43993.3 gates and a performance of 33 MHz has been achieved. YSP-3 has the peak performance of 132 MIPS when the issue rate is 4, which is optimal. Even though the issue rate is 1, the smallest, YSP-3 has at least 33 MIPS.

For performance measurement, the cycle-by-cycle simulator of YSP-3 has been written in C language. We have written hand-code assembly program for common DSP algorithms and AC-3 decoder, and completed simulation in the simulator.

Table 1 shows the average execution time for decoding each frame of 5.1 channel AC-3 bit-streams. Bit allocation decoding, which is the most complex process mainly composed of the mixed tasks [3], is executed in 2.561 ms. Mantissa decoding process including multiplication of coupling coordinates, which also consists of mixed tasks, has the execution time of 3.327ms. The key to achieve such a high performance on controller and mixed tasks is to utilize all the functional units in full and flexible manner.

The performances for four DSP algorithms are shown in Table 2, which shows that YSP-3 has also good performance for DSP tasks. The YSP-3 performs the radix-2 1024-point complex FFT, and real FFT in 1.094 ms and 657.2 us, respectively. It is notable that each FFT includes round operation at every butterfly.

Table 1. Performance on the part of 449 kbps AC-3 algorithm

The Part of AC-3 Algorithm	Execution Time
Exponent Decoding	380.8 us
Bit Allocation Decoding	2.561 ms
Mantissa Decoding	3.327 ms
Channel Decoupling	142.9 us

Table 2. Performance on the DSP tasks

The Part of AC-3 Algorithm	Execution Time
100-tap FIR filter	3.303 us
Two 8x8 matrix product	15.70 us
Radix-2 1024 point complex FFT	1.094 ms
Radix-2 1024 point real FFT	657.2 us

5. CONCLUSION

The 4-way super-scalar processor, which performs controller and mixed tasks efficiently as well as DSP tasks, is presented. The proposed processor has 4 functional units aligned in parallel way, and adopts the 4-way super-scalar instruction structure. Because YSP-3 can operate each functional unit independently, it is suitable for controller and mixed tasks that consist of several simple operations. Besides, it can operate 4 functional units in pipelined manner to process MAC efficiently, it also shows good performance on the DSP tasks.

YSP-3 provides very high degree of flexibility in exploiting functional units, so it can maximize the performance on any application by arranging the instructions to avoid conflict or dependency. In not only the audio CODEC application, but also

multimedia and telecommunication applications, the YSP-3 is well suited as a core technology for an ASIC design, so it offers a simplified system design with high performance at low cost. As we are trying to construct the C compile for this model, the application would be implemented without additional efforts.

6. REFERENCES

[1] ISO/IEC JTC1/SC29/WG11 No.703 “Generic Coding of Moving Pictures and Associated Audio – CD 13818-3(Part3. MPEG-Audio)” Mar., 1994.

[2] Advanced Television Systems Committee(ATSC) Standard Doc. A/52, “Digital Audio Compression Standard(AC-3)”, Nov., 1994.

[3] Steve Vernon “Design and implementation of AC-3 coders”, *IEEE Transactions on Consumer Electronics*, Vol. 41, No. 3, pp. 754-759, Aug., 1995.

[4] Christoph Baumhof, “A Novel 32 Bit RISC Architecture Unifying RISC and DSP”, *ICASSP*, pp. 587-590, 1997.

[5] He Qing and Hou Chao Huan, “RNIW: A novel general-purpose DSP architecuture”, *ICASSP*, pp. 3302-3305, 1996.

[6] H. Sato, E. Holmann, “A dual-issue RISC processor for multimedia signal processing”, *ICASSP*, pp. 591-594, 1997.

[7] Colwell, R. P. et al., “A VLIW architecture for a trace scheduling compiler”., *IEEE Transactions on Computers*, 37: (8) (1998).

[8] Joseph A. Fisher, “Very Long Instruction Word Architectures and the ELI-512”, *Proceedings of the 10th Symposium on Conputer Architecture*, pp. 140-150, IEEE, June, 1983.

[9] Mike Johnson, “Superscalar Microprocessor Design”, *Prentice Hall, Inc.*, pp. 17-24, 1991

[10] Istael Koren, Computer Arithmetic Algorithms, *Prentice-Hall International Editions*, pp. 58-68, 1993.