



A Pipelined RISC CPU

# Aquarius

(SuperH-2 ISA Compatible CPU Core)

**Thorn Aitch**

Rev 1.0 July 12, 2003



**OPENCORES.ORG**

## **Copyright**

Aquarius RTL codes and related documents are copyrighted by the author, but placed into the public domain.

Designs can be altered while keeping list of modifications "the same as in GNU" No money can be earned by selling the designs themselves, but anyone can get money by selling the implementation of the design, such as ICs based on some cores, boards based on some schematics or Layouts, and even GUI interfaces to text mode drivers. "The same as GPL SW" Any update to the design should be documented and returned to the design. Any derivative work based on the IP should be free under OpenIP License. Derivative work means any update, change or improvement on the design. Any work based on the design can be either made free under OpenIP license or protected by any other license. Work based on the design means any work uses the OpenIP Licensed core as a building block without changing anything on it with any other blocks to produce larger design. There is NO WARRANTY on the functionality or performance of the design on the real hardware implementation.

On the other hand, the SuperH-2 ISA (Instruction Set Architecture) executed by Aquarius is rigidly the property of Renesas Technology Corp., which has established on April 1<sup>st</sup>, 2003 by merging semiconductor groups of Hitachi and Mitsubishi. Then you have responsibility to judge if there are not any infringements to Renesas's rights regarding your Aquarius adoption into your design. By adopting Aquarius, the user assumes all responsibility for its use.

## **Trademark**

Aquarius is a name of 5-stages pipelined RISC CPU core that can execute SuperH-2 ISA. Aquarius is not registered trademark. If you officially use the name of Aquarius to external world, you have responsibility to care the legal issues.

## **Royalty Release**

I will not request you any royalties or other financial obligation for your Aquarius adoption to your design and production. But you have responsibility to judge the usage of SuperH-2 ISA, legally. I strongly recommend you should ask Renesas Technology Corp. when you

decide to adopt Aquarius into your products.

### **Patent Notice**

I have not cared that the information contained in this document and Aquarius deliverables cause infringement on the patent, copyright, trademark or trade secret rights of others. You have all responsibilities for determining if your design and products infringe on the intellectual property rights of others.

### **Disclaimers**

Aquarius deliverables including this document are not guaranteed. They may cause any damages to many things, for example, loss of properties, data, money, profits, life, or business. By adopting Aquarius, the user assumes all responsibility for its use. Aquarius deliverables are permanently preliminary, and is subject to change.

### **Contact to Author**

After release of Aquarius onto the OpenCores Organization site, you will be able to contact me via the organization's site. The email address is [thorn\\_aitch@opencores.org](mailto:thorn_aitch@opencores.org). If you find any bugs and strange descriptions, please feel free to inform me.

### **Trademark**

SuperH™ is a trademark of Renesas Technology Corp.

Virtex™ is a trademark of Xilinx, Inc.

Stratix™ is a trademark of Altera, Corp.

Each another proper noun might be a trademark of each rights holders.

## **Revision History**

Rev 0.1: May 1, 2003 by Thorn Aitch      Draft  
    Drafting out a tentative document

Rev 1.0: July 9, 2003 by Thorn Aitch      Release Version  
    A First Release Version

# Index

|   |           |
|---|-----------|
| <b>PART1. USING AQUARIUS.....</b>                             | <b>8</b>  |
| <b>1. INTRODUCTION.....</b>                                   | <b>9</b>  |
| 1.1. WHAT IS AQUARIUS .....                                   | 9         |
| 1.2. PURPOSE OF THIS PROJECT.....                             | 10        |
| 1.3. STRUCTURE OF CHAPTERS IN THIS DOCUMENT.....              | 11        |
| <b>2. SPECIFICATION OVERVIEW.....</b>                         | <b>12</b> |
| 2.1. AQUARIUS INSTRUCTION SET ARCHITECTURE .....              | 12        |
| 2.2. INTERRUPTS AND EXCEPTIONS .....                          | 12        |
| 2.3. DIFFERENCES BETWEEN AQUARIUS AND SUPERH-2.....           | 13        |
| 2.4. AQUARIUS BLOCK DIAGRAM.....                              | 14        |
| 2.5. EXAMPLES OF AQUARIUS BASED SoC.....                      | 16        |
| <b>3. PREPARATION.....</b>                                    | <b>18</b> |
| 3.1. PC ENVIRONMENT.....                                      | 18        |
| 3.2. CYGWIN .....   | 18        |
| 3.3. ICARUS VERILOG .....                                     | 18        |
| 3.4. GNU ASSEMBLER AND C COMPILER FOR SUPERH-2 .....          | 18        |
| 3.5. FPGA DEVELOPMENT TOOL.....                               | 20        |
| 3.6. FPGA BOARD.....  | 20        |
| <b>4. DELIVERABLES .....</b>                                  | <b>21</b> |
| 4.1. DOCUMENT .....   | 21        |
| 4.2. RTL RESOURCES.....                                       | 21        |
| 4.3. VERIFICATION RESOURCES .....                             | 23        |
| 4.4. FPGA RELATED RESOURCES .....                             | 24        |
| <b>5. AQUARIUS CPU INTERFACE SPECIFICATION: “CPU.V” .....</b> | <b>26</b> |
| 5.1. AQUARIUS CPU IN/OUT SIGNALS .....                        | 26        |
| 5.2. SYSTEM SIGNALS .....                                     | 26        |
| 5.3. “WISHBONE” COMPLIANT BUS SIGNALS .....                   | 27        |
| 5.4. HARDWARE EVENT SIGNALS (INTERRUPT).....                  | 30        |

|               |  |           |
|---------------|--|-----------|
| 5.5.          | SLEEP SIGNAL FOR LOW POWER MODE.....         | 32        |
| <b>6.</b>     | <b>SIMULATION TEST BENCH.....</b>            | <b>34</b> |
| 6.1.          | TOP LAYER: "TOP.V".....                      | 34        |
| 6.2.          | SIMULATION TEST BENCH: "TEST.V".....         | 34        |
| 6.3.          | PARALLEL I/O PORT (PIO): "PIO.V".....        | 34        |
| 6.4.          | SERIAL I/O (UART): "UART.V".....             | 37        |
| 6.5.          | SYSTEM CONTROLLER (SYS): "SYS.V".....        | 39        |
| 6.6.          | ON CHIP MEMORY: "MEMORY.V".....              | 42        |
| 6.7.          | SIMULATION TOOLS AND FLOWS.....              | 43        |
| <b>7.</b>     | <b>FPGA IMPLEMENTATION.....</b>              | <b>45</b> |
| 7.1.          | FPGA SYSTEM.....                             | 45        |
| 7.2.          | CIRCUIT OF FPGA BOARD.....                   | 46        |
| 7.3.          | CIRCUIT OF INTERFACE BOARD.....              | 46        |
| 7.4.          | FPGA CONFIGURATION.....                      | 49        |
| 7.5.          | RESULTS OF FPGA CONFIGURATION.....           | 49        |
| 7.6.          | APPLICATION PROGRAMS ON THE FPGA SYSTEM..... | 51        |
| <b>PART2.</b> | <b>INSIDE AQUARIUS CPU.....</b>              | <b>55</b> |
| <b>8.</b>     | <b>AQUARIUS CPU OVERVIEW.....</b>            | <b>56</b> |
| 8.1.          | AQUARIUS BLOCK DIAGRAM.....                  | 56        |
| 8.2.          | AQUARIUS CPU IN/OUT SIGNALS.....             | 58        |
| <b>9.</b>     | <b>OVERVIEW OF PIPELINE CONTROL.....</b>     | <b>59</b> |
| 9.1.          | PIPELINE AND STAGE.....                      | 59        |
| 9.2.          | PIPELINE OF EACH INSTRUCTION.....            | 60        |
| 9.3.          | REGISTER CONFLICT.....                       | 63        |
| 9.4.          | MEMORY ACCESS CONFLICT.....                  | 63        |
| 9.5.          | WHO ISSUES IF? WHO ISSUES ID?.....           | 64        |
| <b>10.</b>    | <b>DECODER UNIT.....</b>                     | <b>65</b> |
| 10.1.         | IN/OUT SIGNALS.....                          | 65        |

|            |   |            |
|------------|---|------------|
| 10.2.      | STRUCTURE OF DECODER UNIT .....                       | 67         |
| 10.3.      | SHIFTING CONTROL SIGNALS.....                         | 71         |
| 10.4.      | PIPELINE STALL .....                                  | 73         |
| 10.5.      | REGISTER FORWARDING .....                             | 76         |
| 10.6.      | EXAMPLES OF PIPELINE CONTROL.....                     | 76         |
| 10.7.      | CONTROL OF PROGRAM COUNTER .....                      | 76         |
| <b>11.</b> | <b>MEMORY ACCESS CONTROL UNIT .....</b>               | <b>81</b>  |
| 11.1.      | IN/OUT SIGNALS .....                                  | 81         |
| 11.2.      | WISHBONE'S ACK AND AQUARIUS' SLOT .....               | 81         |
| 11.3.      | INSTRUCTION FETCH CYCLE.....                          | 82         |
| 11.4.      | MEMORY ACCESS CYCLE.....                              | 83         |
| 11.5.      | IF-MA CONFLICT.....                                   | 85         |
| 11.6.      | BUS WIDTH OF INSTRUCTION FETCH CYCLE (IF_WIDTH) ..... | 86         |
| 11.7.      | READ MODIFY WRITE CYCLE (FOR INSTRUCTION TAS.B) ..... | 86         |
| 11.8.      | STATE MACHINE OF MEMORY ACCESS CONTROL UNIT.....      | 87         |
| <b>12.</b> | <b>DATA PATH UNIT.....</b>                            | <b>89</b>  |
| 12.1.      | IN/OUT SIGNAL TABLE .....                             | 89         |
| 12.2.      | STRUCTURE OF DATA PATH.....                           | 91         |
| <b>13.</b> | <b>MULTIPLIER UNIT .....</b>                          | <b>94</b>  |
| 13.1.      | IN/OUT SIGNAL TABLE .....                             | 94         |
| 13.2.      | ALGORITHM OF MULTIPLICATION .....                     | 94         |
| 13.3.      | STRUCTURE OF MULTIPLIER UNIT .....                    | 97         |
| 13.4.      | CONTROL OF MULTIPLICATION UNIT .....                  | 98         |
| 13.5.      | HOW TO IMPLEMENT SATURATING ACCUMULATION .....        | 99         |
| <b>14.</b> | <b>APPENDIX: AQUARIUS INSTRUCTION CODES .....</b>     | <b>104</b> |

# Part1. Using Aquarius



# 1. Introduction

## 1.1. What is Aquarius

Aquarius is a Core IP (Intellectual Property) of pipelined RISC CPU and can execute SuperH-2 instructions. Aquarius and related information are released to OpenCores Organization web site ([www.opencores.org](http://www.opencores.org)). You can freely download all necessary resources and latest updates from the site.

The reasons why I selected SuperH-2 ISA (Instruction Set Architecture) are as follows.

- (1) SuperH is a very popular CPU core. The software development environments such as C compiler have been well prepared. The GNU C compiler for SuperH is very famous and easy to get. The SuperH had been developed by Hitachi, Ltd. Now, semiconductor group of Hitachi has merged with same group of Mitsubish and new semiconductor company “Renesas Technology Corp.” has established in April, 2003.
- (2) SuperH-2 is a CPU for MCU (Micro Controller Unit). Then the CPU need not to handle complex exception recovering such as memory fault exception from MMU (Memory Managing Unit). This means SuperH-2 has simple structure, easiness to design, and it does not consume many logic gates and power.
- (3) All SuperH-2 instructions have 16bit length. It also makes the hardware very simple. And most important aspect from 16bit fixed length of instructions is that the object code size compiled from C source programs becomes very small.
- (4) And, I love SuperH.

Aquarius is a free and completed soft IP. So I believe that Aquarius can increase SuperH-2 ISA familiars.

Aquarius consists of RTL descriptions. The language is Verilog-HDL. You can implement Aquarius not only in your System LSI but also in your FPGA system. The Aquarius bus interface follows WISHBONE specification maintained by the OpenCores Organization ([www.opencores.org](http://www.opencores.org)), so you can easily connect Aquarius to many IPs registered in OpenCores web site.

During my Aquarius design, I only referred public SuperH document from Renesas such as SH-2 Programming Manual. Of course I could not reach Renesas' internal design information, so the Aquarius may NOT have completely same functionality as real SuperH-2 CPU core, however, Aquarius can execute all public instructions of SuperH-2.

The functionality of Aquarius has been verified by both methods of functional vector simulation and long run tests on FPGA board using program codes from GNU C Compiler and Assembler.

I have designed Aquarius without consuming money except for FPGA hardware. I have used free simulation tools and free FPGA configuration tool. You also do not need to buy expensive EDA tools.

I am not an expert designer of CPU core, so the current Aquarius may not have the best performance. I think efficiency of the design such as area consumption and operation frequency can be improved much more. If you find some improvements, please feel free to suggest your ideas to me.

Please enjoy the exciting deep IP design world. You can modify Aquarius to make your original system. I hope Aquarius will help system designers, university students and electronics hobbyists.

## **1.2. Purpose of this Project**

The main purpose of Aquarius Project is to provide everyone a pipelined RISC CPU core as one of the IPs for System LSI and FPGA system. You can get information about how to design actual useful RISC CPU.

The Aquarius has SuperH-2 compatible ISA, so I hope that SuperH familiarized people will increase more and more. Many embedded system, for example Robots, Industrial Systems, Measurement Instruments, and many kind of digital information systems controlled by embedded micro controllers, can be realized by SuperH-2 architecture.

I provide Aquarius without any license fee and royalty. You can freely get the latest

Aquarius IP codes from OpenCores Organization on the internet whenever you like. And I will introduce you the cheapest but excellent design environments via this document. You will be able to modify Aquarius and establish your original IP.

### **1.3. Structure of Chapters in this document**

This document consists of 2 parts. The first part describes how to use Aquarius, for example, explanation of interface signals, test bench and FPGA implementation. All readers should read first part. The second part shows inside Aquarius which is way of thought for designing a pipelined RISC CPU. If you want to understand the apparatus of pipelined RISC CPU and want to design your original CPU core, you should read second part, too.

## 2. Specification Overview

### 2.1. Aquarius Instruction Set Architecture

Aquarius is based on SuperH-2 Instruction Set Architecture (ISA). The SuperH-2 CPU has RISC-type instruction sets and 16 32bit-general-registers (R0-R15). All instructions have 16bits fixed length. The SuperH-2 is based on 5 stages pipelined architecture, so basic instructions are executed in one clock cycle pitch, which dramatically improves instruction execution speed. The CPU also has an internal 32-bit architecture for enhanced data processing ability such as multiply and accumulation like DSP functionality.

The detail document of SuperH-2 CPU architecture can be found in Renesas web site.

<http://www.renesas.com/>

Please reach to the SuperH product page and find the SH-2 related product documents. Then search document type of “Programming Manual” and find the “*SuperH RISC Engine SH-1/SH-2/SH-DSP Programming Manual*”. This manual includes explanations among SH-1, SH2 and SH2-DSP Instruction set. Please check up only SH-2 portions from this manual. But it does not describe about exception and interrupt. For that information, pick up product manual such as “SH7040 series Hardware Manual” and refer to chapters regarding Exception and Interrupt Controller.

### 2.2. Interrupts and Exceptions

Like SuperH-2 CPU, Aquarius can handle interrupt requests, such as NMI (non maskable interrupt) and IRQ (interrupt request). The interrupt priority level can be set from 0 to 16. The interrupt request whose priority level is higher than I bit (I3-I0) in SR (Status Register) will accepted by CPU. The priority of NMI is 16, so it is always accepted. The priority level and the vector number of IRQ can be informed from external circuit such as interrupt controller or system controller. If the priority level is zero, such interrupt will not be accepted. Once the interrupt is accepted, the interrupt exception will start. It copies the interrupt request level to I bit (I3-I0) in SR ,push SR and PC onto stack, fetch the vector address and branch to targeting interrupt service routine. To return from interrupt service routine, use RTE, which pops PC and SR and starts from the address of popped PC.

By the 4 bit priority control scheme, the interrupt can be nested.

The other exceptions such as CPU address error, DMA address error, TRAP Instruction, Illegal Instruction, Slot Illegal Instruction, Manual Reset and Power on Reset are fully supported by Aquarius.

### 2.3. Differences between Aquarius and SuperH-2

Aquarius can execute all public SuperH-2 instructions. But there are some functional differences between Aquarius and real SuperH-2 CPU.

#### (1) Improvement of Multiplication Cycle

Table 2.1 shows that the execution cycle of the multiplication related instructions of Aquarius are slightly different from SuperH-2's because I guess the structure of connection between CPU and Multiplier is changed from real SuperH-2. You can find some performance is improved. Especially, the pitch cycle reduction of MAC.L will improve performance of many real time applications. The details of pipeline control will be shown in later chapter.

| Instruction      | Aquarius                     | SuperH-2                     | Notes |
|------------------|------------------------------|------------------------------|-------|
| MAC.W @Rm+, @Rn+ | C=2, P=2, L=3                | C=2, P=2, L=2                |       |
| MAC.L @Rm+, @Rn+ | C=2, <b>P=2</b> , L=4        | C=2, <b>P=4</b> , L=4        |       |
| MULS/U.W Rm, Rn  | C=1, <b>P=1</b> , L=2        | C=1, <b>P=2</b> , L=2        |       |
| DMULS/U.L Rm, Rn | C=1, <b>P=3</b> , <b>L=2</b> | C=1, <b>P=4</b> , <b>L=4</b> |       |
| MUL.L Rm, Rn     | C=1, <b>P=3</b> , <b>L=2</b> | C=1, <b>P=4</b> , <b>L=4</b> |       |

C (Cycle): Instruction Execution Cycle if there is no contention. This is minimum cycle.

P (Pitch): Instruction Execution Pitch cycle if same instructions are repeated.

L (Latency): Latency cycle until STS, which is located just after me, and stores MACH/MACL to Rn.

**Table2.1. Differences of Instruction Execution Cycles between Aquarius and SuperH-2**

#### (2) Detection of Illegal Instruction

The real SuperH-2 decodes all illegal instructions. But in Aquarius, only the FF-line instructions (0xFFxx) are recognized as illegal instructions that bring up "Illegal Instruction Exception". Other "should-be illegal instructions" are not fully decoded, so these operations are seemed as "Undefined". Actually, the operation of undefined instructions will be just same as similar code's instruction. By this shortcut, the usage of area is reduced. Of course, the Slot Illegal Exception (in the case that a branch instruction placed at the

delay slot of delayed branch) is completely detected.

Even if you want Aquarius to detect all illegal instructions, you can easily modify the decode unit's RTL code.

### (3) Instruction Codes for Exception

Some instructions in F-line (0xFxxx) are used for launching exceptions. These are shown in Tabel2.2, which are not defined in actual SuperH ISA. In the CPU decoder, the hardware event, for example interrupt, exchanges a fetched instruction to another code (in Table1.2) which launches exception, and then changes the control sequence from normal instruction's one to the exception's. If these instructions exist in program code, corresponding exception will start, but will not have correct operation, such as interrupt priority control. I recommend you not to write the Exception Launch Instructions in program code.

| Instruction | Correct Code | Exception Sequence | Notes                            |
|-------------|--------------|--------------------|----------------------------------|
| 0xF7xx      | 0xF700       | Power On Rest      | Lower 8bit is used as vector No. |
| 0xF6xx      | 0xF602       | Manual Reset       | Lower 8bit is used as vector No. |
| 0xF3xx      | 0xF30A       | DMA Address Error  | Lower 8bit is used as vector No. |
| 0xF2xx      | 0xF209       | CPU Address Error  | Lower 8bit is used as vector No. |
| 0xF1xx      | 0xF10B       | NMI                | Lower 8bit is used as vector No. |
| 0xF0xx      | 0xF0xx       | IRQ                | Lower 8bit is used as vector No. |
| 0xFF04      | 0xFF04       | General Illegal    | Lower 8bit is used as vector No. |
| 0xFE06      | 0xFE06       | SLOT Illegal       | Lower 8bit is used as vector No. |

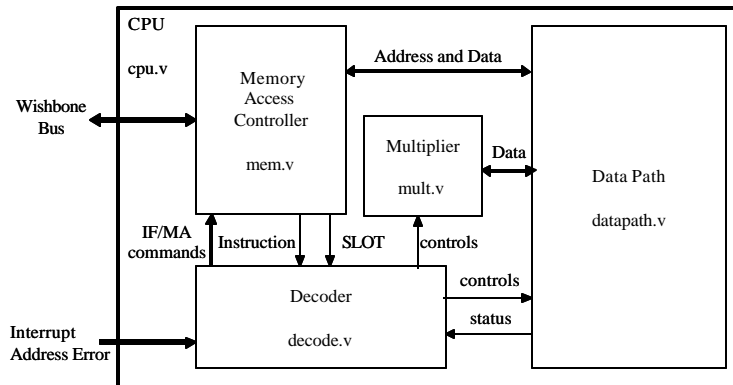
**Table2.2 Exception Launch Instruction**

### (4) ICE Support Instructions

Although the actual SuperH-2 may have dedicated instructions to support the ICE (in circuit emulator), Aquarius do not have, because those instructions are not released generally. In the test bench of Aquarius, I have implemented the "break" function by NMI (non maskable interrupt).

## 2.4. Aquarius Block Diagram

Figure 2.1 shows the block diagram of Aquarius CPU core.



**Figure2.1. Block Diagram of Aquarius**

Top layer of Aquarius is “CPU” which has WISHBONE compliant bus signals and accepts interruption related signals. The most important system signals such as clock and reset are not shown in this figure.

The Memory Access Controller handles instruction fetch and data read/write access. The operations of Memory Access Controller are fully controlled by Decoder unit. Memory Access Controller sends fetched instruction bit fields to the Decoder unit, and interchanges read/write data and its address with Data Path unit. Aquarius assumes the Wishbone bus is a Non-Harvard bus, then the simultaneous instruction fetch and R/W data access makes bus contention. Memory Access Controller handles such contention smoothly and informs the pipeline stall caused by the bus contention to Decoder unit. Also, the Memory Access Controller can sense each boundary of bus cycles (with wait state) from WISHBONE ACK signal. In Aquarius architecture (may be in SuperH-2 architecture as well), such bus cycle boundary corresponds to the pipeline’s slot edge. So the Memory Access Controller produces the most important pipeline control signal “SLOT” indicating pipeline slot edge.

The Data Path unit has registers you can see in programmer’s model in SuperH-2 manual such as General Registers (R0 to R15), Status Register (SR), Global Base Register (GBR), Vector Base Register (VBR), Procedure Register (PR) and Program Counter (PC). The Multiplication and Accumulate Registers (MACH/MACL) are found in Multiplication unit. The Data Path unit also has necessity operation resources such as ALU (Arithmetic and Logical operation Unit), Shifter, Divider, Comparator, temporary registers, many selectors,

interfaces to/from Memory Access Controller and Multiply unit, and several buses to connect each resource. The Data Path is fully controlled by control signals from Decoder unit.

Multiply unit has a 32bit x 16bit multiplier and its control circuits. A 16bit x 16bit multiply operation is executed in one clock cycle. A 32 bit x 32bit multiply operation is done in two clock cycles. Multiply unit also has the Multiplier and Accumulate Registers (MACH/MACL). The MACH/MACL are not only the final result registers of multiply or multiply-and-accumulation but also the temporary registers to hold the 48bit partial multiply result from 32bit x 16bit multiplier for 32bit x 32bit operation. The multiply instruction, for example MULS.L, clears the contents of MACH/MACL in early stage of the instruction operation. However the multiply and accumulate instruction, for example MAC.L, does not clear MACH/MACL before the operation. The MAC.L accumulates its own partial multiply result to initial MACH/MACL and then finalize the operation result. The major difference between multiply (MULS.L) and “multiply and accumulate” (MAC.L) is whether to clear or not to clear the MACH/MACL before the operation. And also, for MAC.L and MAC.W instruction, the accumulation adder in this unit has saturating function.

The Decoder unit is the fundamental CPU controller. It orders Memory Access Controller fetch instructions and then receives the instruction. The Decoder Unit decodes the instruction bit fields and judges the followed operations. Basically, the Decoder unit plays the role only for the instruction ID stage. But it throws many control signals for following EX, MA and WB stages toward Data Path unit, Multiplication unit, and Memory Access Controller. These control signals are kept and shifted with its pipeline flow at each slot edge until reaching to the target stage of the instruction. The Decoder unit detects every conditions of pipeline stalling, and makes each unit of CPU be controlled properly. Also, it controls not only simple 1 cycle instructions but also multi cycle instructions and exception's sequences such as interrupt and address error.

Detailed design description of each unit is found in Part 2.

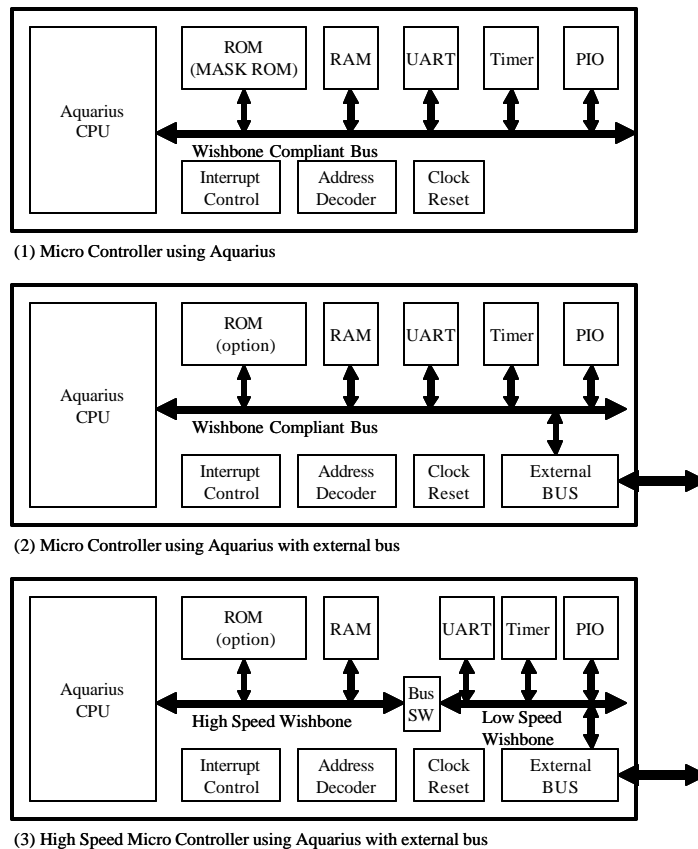
## **2.5. Examples of Aquarius based SoC**

Figure 2.2 shows some examples of SoC using Aquarius.



- (1) It is a simple micro controller that has CPU, ROM, RAM and some peripherals. Each module is connected by a common Wishbone bus.
- (2) It is same as (1) except it has external bus interface. If the external bus interface is designed properly, it can be connected any memories such as ROM, Burst ROM, SRAM, SDRAM and, if you desire, DDR may be possible.
- (3) If the bus operation frequency is high (for example, over 100MHz), one common Wishbone bus will not catch up with the frequency because of heavy load. In such case, I recommend you to divide the internal bus into as least two, one is the fast speed bus with only CPU and internal fast memories, and the other one is slow speed bus with many slow peripheral modules and external memory interface.

I provide Aquarius deliverables not only as CPU core but also as MCU like (1), which has ROM, RAM, UART, PORT and System Controller (interrupt and exception controller) etc.



**Figure2.2. Examples of System LSI using Aquarius**

## **3. Preparation**

This chapter describes my recommendations regarding necessary preparations before starting development. You do not need any expenses except PC environment and FPGA board.

### **3.1. PC environment**

This is the most important tool. Any Windows machines are OK. I still use SONY VAIO Notebook PCG-R505FR/D with Mobile PentiumIII 800MHz, 256MB RAM, and Microsoft Windows XP. Even such machine has enough power to design Aquarius. Of course, you need broadband internet connection such as xDSL to download many required resources.

The reason why I use Windows machine instead of Linux machine is that the most “free” FPGA development tools from FPGA vendors such as Xilinx and Altera run on only Windows environment.

### **3.2. Cygwin**

The simulator of Verilog-HDL codes and the compiler/assembler of SuperH-2 run on the UNIX environment. In order all tools to live together in Windows environment, the Cygwin is a good selection. Download the latest Cygwin system from <http://www.cygwin.com>, and full-install to your PC according to its instructions. After the Cygwin installation, many UNIX/Linux applications and all Windows applications simultaneously run on your PC without circumstances.

### **3.3. Icarus Verilog**

I think the most excellent free Verilog simulator is Icarus Verilog. Download Icarus from <http://www.icarus.com/eda/verilog/index.html> and install it from Cygwin console window according to Icarus’s installation document. If you have installed Cygwin with full packages, you will not encounter any problems.

### **3.4. GNU Assembler and C Compiler for SuperH-2**

To make verification program and to develop application program, the SuperH-2 assembler and compiler are necessary for you. Install them as follows.

(1) Download following files from <ftp://ftp.gnu.org/pub/gnu/>

```
binutils-2.13.1.tar.gz
```

```
gcc-2.95.3.tar.gz
```

```
gdb-5.2.1.tar.gz
```

(2) Download following file from <http://sources.redhat.com/newlib/>

```
newlib-1.10.0.tar.gz
```

(3) Place these 4 files under /usr/local/src.

(4) Install GNU binutils.

```
cd /usr/local/src
```

```
gzip -dc binutils-2.13.1.tar.gz | tar xvf -
```

```
cd binutils-2.13.1
```

```
mkdir work
```

```
cd work
```

```
../configure --prefix=/usr/local --target=sh-elf
```

```
make
```

```
make install
```

(5) Install GNU gcc and newlib.

```
cd /usr/local/src
```

```
gzip -dc newlib-1.10.0.tar.gz | tar xvf -
```

```
gzip -dc gcc-2.95.3.tar.gz | tar xvf -
```

```
cd gcc-2.95.3
```

```
ln -s ../newlib-1.10.0/newlib .
```

```
mkdir work
```

```
cd work
```

```
../configure --prefix=/usr/local --target=sh-elf --with-gnu-as  
--with-gnu-ld --with-dwarf2 --disable-multilib --enable-languages=c  
--with-newlib
```

```
make
```

```
make install
```

(6) Install GNU gdb.

```
cd /usr/local/src
```

```
gzip -dc gdb-5.2.1.tar.gz | tar xvf -
```

```
cd gdb-5.2.1
mkdir work
cd work
./configure --prefix=/usr/local --target=sh-elf
make
make install
```

### **3.5. FPGA development tool**

To implement your design to FPGA, you need FPGA development tool. The FPGA vendors release excellent free development tool which has editor, logic synthesizer, static timing analyzer, placer & router and configuration binary generator. In Aquarius project, I have been using Xilinx free ISE Webpack 5.x. Download it from following URL site and install it on your Windows environment. It has a nice Verilog syntax editor, so I have mainly used the editor in “Project Navigator” of ISE during Aquarius development.

[http://www.xilinx.com/xlnx/xil\\_prodcat\\_landingpage.jsp?title=ISE+WebPack](http://www.xilinx.com/xlnx/xil_prodcat_landingpage.jsp?title=ISE+WebPack)

### **3.6. FPGA Board**

To verify the logic design, implementing it to FPGA device is very good method. The FPGA plays a role as a hardware logic emulator, so the verification speed is much faster than vector logic simulation. And the CPU in FPGA can execute very large and long program quickly, so the verification quality will be improved.

I bought a board which has Xilinx VirtexE-300 (XCV300E). In my case, the board vendor name is HuMANDATA Ltd, and the product name is XSP-009-300. The site is <http://www.hdl.co.jp/> which unfortunately has only Japanese description. But this company opens their technical documents regarding the products on their site, freely.

You can find the board schematic, which can be read even by non-Japanese people, from <http://www.hdl.co.jp/ftpdata/xsp-009/XSP009.sch.pdf>.

I think you can find another good FPGA boards from many vendors around you. Or if you can get FPGA device, making a board by “DIY” is a good choice.

I added some external circuit such as LCD display, Hex Key board and I/F to RS232C to above board to make the verification be smooth. The detail circuit is described later.

## 4. Deliverables

This chapter shows the all deliverables of Aquarius project.

### 4.1. Document

`Aquarius.pdf` : this document (Adobe Acrobat Reader)

`Aquarius.doc` : this document (Microsoft Word)

### 4.2. RTL Resources

Verilog –HDL (RTL) of Aquarius CPU and its test bench

The set of RTL codes of Aquarius includes not only CPU RTL but also Simple MCU RTL that comprises CPU, ROM, RAM, PIO, UART and System Controller. The RTL codes except CPU are used as test bench of CPU. Of course, you can implement all RTL codes into your FPGA, and verify it much more efficiently like as I did. Figure 3.1 shows RTL structure of Aquarius MCU.

#### Test Bench comprises...

|                          |   |
|--------------------------|---|
| <code>timescale.v</code> | Timescale definition. All files include me. |
| <code>test.v</code>      | Test Bench                                  |
| <code>top.v</code>       | Top layer of MCU                            |

#### MCUcomprises...

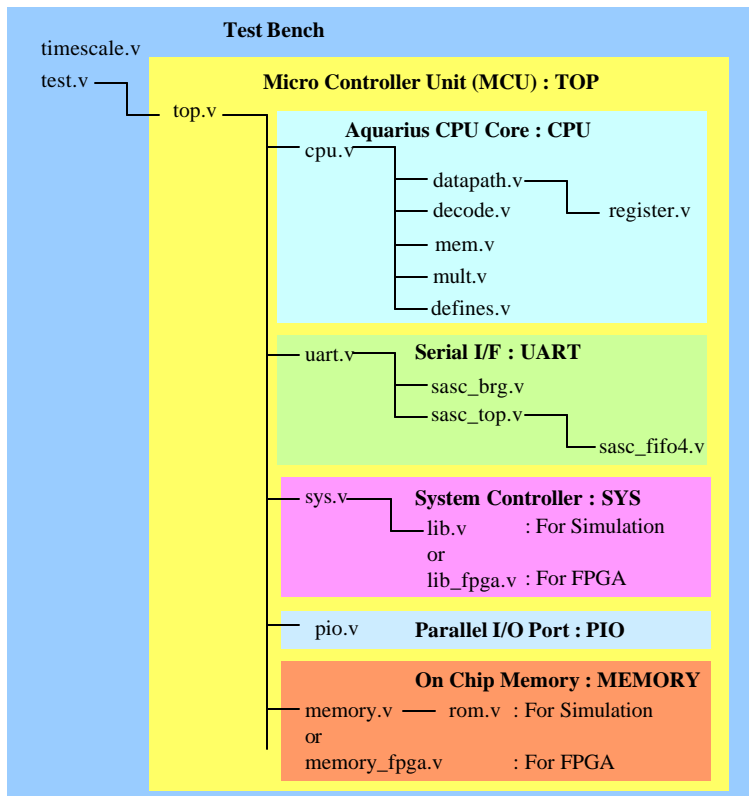
|                            |   |
|----------------------------|---|
| <code>top.v</code>         | Top layer of MCU  |
| <code>pio.v</code>         | Parallel IN/OUT Interface   |
| <code>memory.v</code>      | ROM(8KB) and RAM(8KB) for Verilog simulation.<br>ROM can be initialized from S-format binary code.                              |
| <code>rom.v</code>         | ROM description created by a converter from S-format.   |
| <code>memory_fpga.v</code> | " <code>memory.v</code> " for Xilinx FPGA's configuration (BlockRAM)<br>All area (16KB) can be initialized by INIT constraints. |
| <code>uart.v</code>        | UART (Universal Asynchronous Receiver/Transmitter)  |
| <code>sasc_brg.v</code>    | Baud Rate Generator   |
| <code>sasc_top.v</code>    | UART Body   |
| <code>sasc_fifo4.v</code>  | 4 step FIFO for UART Buffer   |

sys.v            System Controller that handles interrupts and exceptions  
           lib.v            A clock stop gate (SLEEP instruction) for Verilog sim.  
           lib\_fpga.v        “lib.v” for Xilinx FPGA configuration  
 cpu.v            Top layer of CPU (Aquarius)

**Aquarius CPU comprises...**

cpu.v            Top layer of CPU (Aquarius)  
 datapath.v       Data Path  
           register.v        General Registers R0-R15  
 decode.v            Instruction Decoder  
 mem.v            Memory Access Controller  
 mult.v            Multiplier  
 defines.v        Constant Parameters referred from CPU

The UART is based on the “Simple Asynchronous Serial Communication Device created by Rudolf Usselmann” downloaded from <http://www.opencores.org/cores/sasc/>.



**Figure 3.1 RTL Structure**

### 4.3. Verification Resources

I also provide simple but useful resources for logic verification and program development.

#### (1) Assembler Source Programs for Verilog simulation

I provide some example programs for Verilog simulation. You can find them under the directory “sha\_testsource”.

#### (2) Converter from S-format object to Verilog ROM description “rom.v”

The assembler can make S-format object. But it should be linked to Aquarius Verilog test bench. I made a simple binary converter from S-format to Verilog ROM description “rom.v”. This converter is “genrom.c” which is C source program. Compile it on your Cygwin console. Simply, do this.

```
$ gcc -o genrom.exe genrom.c
```

The usage is very simple. If your S-format binary name is “test.obj”, you can convert it to Verilog ROM description by typing as follows.

```
$ ./genrom test.obj
```

This operation creates “rom.v”, which is an 8Kbyte ROM.

Note that the “genrom” supports the S-Format which has only S0 (comment), S3 (4byte address) and S7 (end of record).

#### (3) Script to launch Assembler

The script named “asm” launches the GNU assembler, creates object code as an S-format file, and converts the S-Format object to a Verilog ROM description “rom.v”. The “asm” is very short script as follows.

---

```
#!/bin/bash

sh-elf-as -a $1 > lis
sh-elf-as -o a.out $1
sh-elf-objcopy -O srec --srec-forceS3 a.out obj
./genrom obj
```

---

The usage is also simple. If you have assemble source program named “test.src”, simply

type as follows.

```
$ ./asm test.src
```

This operation creates Verilog ROM description “rom.v” corresponding to “test.src”.

#### (4) Script to launch Verilog Simulation

After creating “rom.v”, now you can simulate Aquarius. First of all, prepare a text file “test.txt” that lists up all Verilog source files.

The script named “sim” launches the Icarus Verilog Simulator. The “sim” is very short script as follows.

---

```
#!/bin/bash

iverilog -o test -c test.txt
vvp -v test
```

---

By the Aquarius test bench “test.v”, the simulation results is created as “test\_result.txt” which is a trace list of bus cycle and important register contents.

### 4.4. FPGA related Resources

To implement Aquarius into Xilinx VirtexE, I have prepared some resources.

#### (1) Converter from S-format object to Xilinx BlockRAM INIT Constraints

In case of FPGA implementation, ROM should be configured by BlockRAM instead of “rom.v”, which is described by continuous “case” statements, to reduce the consumption of logic cells. The BlockRAM can be initialized by INIT statement in user constraints file (.ucf). So I made a converter from S-format object to INIT statement.

The converter is “genram.c”, which is also a C program, then compile it on your Cygwin console.

```
$ gcc -o genram.exe genram.c
```

The usage is very simple. If your S-format binary name is “test.obj”, you can convert it to INIT description by typing as follows.

```
$ ./genram test.obj
```

This operation creates “ram.dat”, which is 16Kbyte BlockRAM initialization.



The content of “ram.dat” is as follows.

---

```
INST "MEMORY_Mram_RAM0HH_inst_ramb_0" INIT_00 = 0000000000000000...;
INST "MEMORY_Mram_RAM0HH_inst_ramb_0" INIT_01 = 0000000000000000...;
INST "MEMORY_Mram_RAM0HH_inst_ramb_0" INIT_02 = 2121212121212121...;
INST "MEMORY_Mram_RAM0HH_inst_ramb_0" INIT_03 = 6765636100D02F2F...;
```

---

After creating “ram.dat”, add this content after the tail of your user constraints file (.ucf), or change all old INST statement. Then, configure your FPGA.

### (2) An example of User Constraints File (.ucf)

I provide an example of user constraints file (top.ucf) which corresponds to my FPGA system described later.

### (3) Some Applications for FPGA System

Following application programs are provided.

|                  |                         |
|------------------|-------------------------|
| Monitor Program  | shc_monitor_release_v1/ |
| LCD Test         | shc_lcdtest/            |
| Interrupt! Clock | shc_clock/              |

Details are described later (FPGA Implementation)

## 5. Aquarius CPU Interface Specification: “cpu.v”

### 5.1. Aquarius CPU IN/OUT Signals

The Aquarius CPU (“cpu.v”)’s IN/OUT signals are shown in Table5.1. In Aquarius CPU logic circuit, all signals are positive logic level and the changing timing is always at positive edge of CLK.

| <i>Class</i>       | <i>Signal Name</i> | <i>Direction</i> | <i>Meaning</i>     | <i>Notes</i> |
|--------------------|--------------------|------------------|--------------------|--------------|
| <b>System</b>      | CLK                | Input            | System clock       |              |
| <b>Signals</b>     | RST                | Input            | Power On Reset     |              |
| <b>Wishbone</b>    | CYC_O              | Output           | Cycle Output       |              |
| <b>Bus</b>         | STB_O              | Output           | Strobe Output      |              |
| <b>Signals</b>     | ACK_I              | Input            | Device Acknowledge |              |
|                    | ADR_O[31:0]        | Output           | Address Output     |              |
|                    | DAT_I[31:0]        | Input            | Read Data          |              |
|                    | DAT_O[31:0]        | Output           | Write Data         |              |
|                    | WE_O               | Output           | Write Enable       |              |
|                    | SEL_O[3:0]         | Output           | Byte Lane Select   |              |
|                    | TAG0_I (IF_WIDTH)  | Input            | Fetch Width        |              |
| <b>Hardware</b>    | EVENT_REQ_I[2:0]   | Input            | Event Request      |              |
| <b>Event</b>       | EVENT_INFO_I[11:0] | Input            | Event Information  |              |
| <b>(interrupt)</b> | EVENT_ACK_O        | Output           | Event Acknowledge  |              |
| <b>SLEEP</b>       | SLP                | Output           | Sleep Pulse        |              |

**Table5.1 Aquarius CPU IN/OUT Signals**

### 5.2. System Signals

#### (1) CLK

The clock input [CLK] coordinates all activities for the internal logic within the WISHBONE interconnect. All output signals are registered at the rising edge of [CLK]. All input signals are stable before the rising edge of [CLK].

#### (2) RST

The reset input [RST] forces the WISHBONE interface to restart. Furthermore, all internal

state machines are forced into an initial state.

When system power-on (cold start), [RST] should be asserted at least for 1 cycle. The Aquarius CPU senses [RST] asynchronously, so any glitch pulse should not be overlaid on [RST] signal. Aquarius Flip Flops are written as follows.

```
always @(posedge CLK or posedge RST)
{
    if (RST)
    {
    }
    else
    {
    }
}
```

If your in-house design rule inhibits asynchronous reset at Flip Flops, you can rewrite RTL codes of Aquarius as follows. Aquarius CPU can operate in synchronous reset manner.

```
always @(posedge CLK)
{
    if (RST)
    {
    }
    else
    {
    }
}
```

### 5.3. “WISHBONE” Compliant Bus Signals

The bus specification of Aquarius CPU is based on WISHBONE classic bus. It follows “Specification for the WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores Revision: B.3, Released: September 7, 2002”. The detail specification document is found in the OpenCores site.

#### (1) CYC\_O

The cycle output [CYC\_O], when asserted, indicates that a valid bus cycle is in progress.

The signal is asserted for the duration of all bus cycles. For example, during a Read Modify Write cycle caused by TAS.B (test and set instruction for semaphore protocol), there are two data transfers. The [CYC\_O] signal is asserted during the first data read, and remains its assertion until the last data write. The [CYC\_O] signal is useful for bus arbiter to prevent exchanging the current bus master to another device such as DMA controller during the TAS.B read modify write cycle.

**(2) STB\_O**

The strobe output [STB\_O] indicates a valid data transfer cycle. It is used to qualify various other signals on the interface such as [SEL\_O]. The SLAVE module asserts either the [ACK\_I] signals in response to every assertion of the [STB\_O] signal.

**(3) ACK\_I**

The acknowledge input [ACK\_I], when asserted, indicates the normal termination of a bus cycle. The [ACK\_I] creates CPU internal signal "SLOT" to indicate the edge of pipeline slot.

**(4) ADR\_O[31:0]**

The address output array [ADR\_O] is used to pass a binary address.

**(5) DAT\_I[31:0]**

The data input array [DAT\_I] is used to read binary data from external devices such as ROM, RAM and peripheral modules.

**(6) DAT\_O[31:0]**

The data output array [DAT\_O] is used to write binary data to external devices such as RAM and peripheral modules.

**(7) WE\_O**

The write enable output [WE\_O] indicates whether the current local bus cycle is a READ or WRITE cycle. The signal is negated during READ cycles, and is asserted during WRITE cycles.

**(8) SEL\_O[3:0]**

The select output array [SEL\_O] indicates where valid data is expected on the [DAT\_I] signal array during READ cycles, and where it is placed on the [DAT\_O] signal array during WRITE cycles. The minimum data granularity size is BYTE, so each SEL\_O corresponds to each byte lane. The data alignment position is described in detail later.

### **(9) TAG0\_I (IF\_WIDTH)**

All SuperH-2 instruction has 16bit length. The memory such as ROM and RAM connected to Aquarius CPU has 32bit data width. So, when CPU fetches its instruction from 32bit width memory, CPU can get 2 instructions. But if the device data width is 16bit, only one instruction can be sent to CPU at once fetch cycle. Latter case may happen when CPU fetches its instruction from 16bit width external bus, for example.

Therefore, WISHBONE glue logic must inform CPU the instruction fetch space's width.

If the address space is 32bit width, WISHBONE should return IF\_WIDTH=1, else should return IF\_WIDTH=0 before ACK\_I signal is asserted.

If CPU fetches instruction from address 0x2, 0x6, 0xa, 0xe, CPU can get only 1 instruction by the fetch cycle. But CPU knows such status, so WISHBONE need not check such case. The WISHBONE glue logic should care only the instruction sending band width that is the data width of accessing address space.

In other words, in case of instruction fetch, IF\_WIDTH has its meaning only when lower 2bit of address is 2'b00. The IF\_WIDTH informs CPU how many instructions should be fetched. But if the lower 2bit of address is 2'b10, CPU can get only one instruction regardless IF\_WIDTH. In latter case, CPU ignores the IF\_WIDTH.

[CAUTION] Aquarius CPU assumes that the internal bus width is always 32bit. If you connect the internal WISHBONE to 16bit/8bit external bus or peripheral modules, some glue bus control logic should be created to convert internal 32bit data to/from 8bit/6bit data with proper wait timing controls.

### **(10) Data Alignment Position**

Aquarius CPU is big-endian. The data width is 32 bit, memory data access granularity is byte, and instruction fetch granularity is 16 bit.


Table5.2 shows the data alignment position on WISHBONE data bus for each access.

In WISHBONE specification data sheet, 32bit operand size is called as "DWORD", but in

SuperH and Aquarius world, 32bit is called as “Long” or “Long Word”.

Note that in case of write operation, the unselected lanes have same write data as valid lane’s one. For example, in WORD writing to address 2(2’b10), the valid lane is bit15-0, but bit31-16 of DAT\_O has same data as bit15-0.

| Access Type      |         | Lane     | Lane     | Lane     | Lane    | Notes |
|------------------|---------|----------|----------|----------|---------|-------|
| Size             | Address | [31:24]  | [23:16]  | [15:8]   | [7:0]   |       |
|                  | Lower   | SEL [3]  | SEL[2]   | SEL[1]   | SEL[0]  |       |
|                  | 2bit    |          |          |          |         |       |
| Data Read Long   | 2’b00   | D[31:24] | D[23:16] | D[15:8]  | D[7:0]  |       |
| Data Read Word   | 2’b00   | D[15:8]  | D[7:0]   | ignored  | ignored |       |
|                  | 2’b10   | Ignored  | ignored  | D[15:8]  | D[7:0]  |       |
| Data Read Byte   | 2’b00   | D[7:0]   | ignored  | ignored  | ignored | DAT_I |
|                  | 2’b01   | Ignored  | D[7:0]   | ignored  | ignored |       |
|                  | 2’b10   | Ignored  | ignored  | D[7:0]   | ignored |       |
|                  | 2’b11   | Ignored  | ignored  | ignored  | D[7:0]  |       |
| Data Write Long  | 2’b00   | D[31:24] | D[23:16] | D[15:8]  | D[7:0]  |       |
| Data Write Word  | 2’b00   | D[15:8]  | D[7:0]   | D[15:8]  | D[7:0]  |       |
|                  | 2’b10   | D[15:8]  | D[7:0]   | D[15:8]  | D[7:0]  |       |
| Data Write Byte  | 2’b00   | D[7:0]   | D[7:0]   | D[7:0]   | D[7:0]  | DAT_O |
|                  | 2’b01   | D[7:0]   | D[7:0]   | D[7:0]   | D[7:0]  |       |
|                  | 2’b10   | D[7:0]   | D[7:0]   | D[7:0]   | D[7:0]  |       |
|                  | 2’b11   | D[7:0]   | D[7:0]   | D[7:0]   | D[7:0]  |       |
| Fetch IF_WIDTH=1 | 2’b00   | I0[15:8] | I0[7:0]  | I1[15:8] | I1[7:0] |       |
| Fetch IF_WIDTH=0 | 2’b00   | I[15:8]  | I[7:0]   | ignored  | Ignored | DAT_I |
| Fetch IF_WIDTH=* | 2’b10   | ignored  | ignored  | I[15:8]  | I[7:0]  |       |

 : Corresponding SEL\_O[n] is asserted.

**Table5.2 Data Alignment Position**

#### 5.4. Hardware Event Signals (Interrupt)

CPU should accept some requests from hardware events such as interrupt, address error and manual reset. These requests are informed to CPU by EVENT\_REQ[2:0] associated

with EVENT\_INFO[11:0]. The EVENT\_REQ[2:0] shows the kind of event request. The meanings are shown in Table5.3. In case of IRQ request, EVENT\_INFO[11:0] should also be valid. The upper 4bit of EVENT\_INFO shows the priority level of the requesting IRQ, and the lower 8bit of EVENT\_INFO shows its vector No. The vector address of IRQ equals to EVENT\_INFO[7:0] \* 4. Also see Table5.4.

The EVENT\_REQ and EVENT\_INFO should be asserted and be valid at same timing. CPU samples them at same timing (at the decode stage of pipeline). If the EVENT\_REQ do not show IRQ request, CPU ignores the EVENT\_INFO. After CPU samples the EVENT\_REQ (and EVENT\_INFO), CPU asserts EVENT\_ACK, which shows that the CPU accepts the hardware event request that is valid just at when EVENT\_ACK is being asserted. The EVENT\_REQ should be negated or should be change to next request just after EVENT\_ACK is asserted.

If the event can not be accepted by CPU; that happens in case of lower priority IRQ than I-bit in SR, interrupt request just after the instruction that masks interrupt (for example LDC/LDC.L), or all hardware exception events just after delayed branch instruction; the EVENT\_ACK is not asserted until the request signals can be accepted by CPU. Of course, if the event request is negated before CPU's sampling, the event request can not be accepted by CPU. The hardware event request timing is shown in Figure5.1 with internal pipeline controls.

| Hardware Event    | EVENT_REQ[2:0] | Notes                     |
|-------------------|----------------|---------------------------|
| NOP               | 3'b111         |                           |
| IRQ               | 3'b000         | Also use EVENT_INFO[11:0] |
| NMI               | 3'b001         |                           |
| CPU Address Error | 3'b010         |                           |
| DMA Address Error | 3'b011         |                           |
| Manual Reset      | 3'b110         |                           |

**Table5.3 Hardware Event Request Signal: EVENT\_REQ[2:0]**

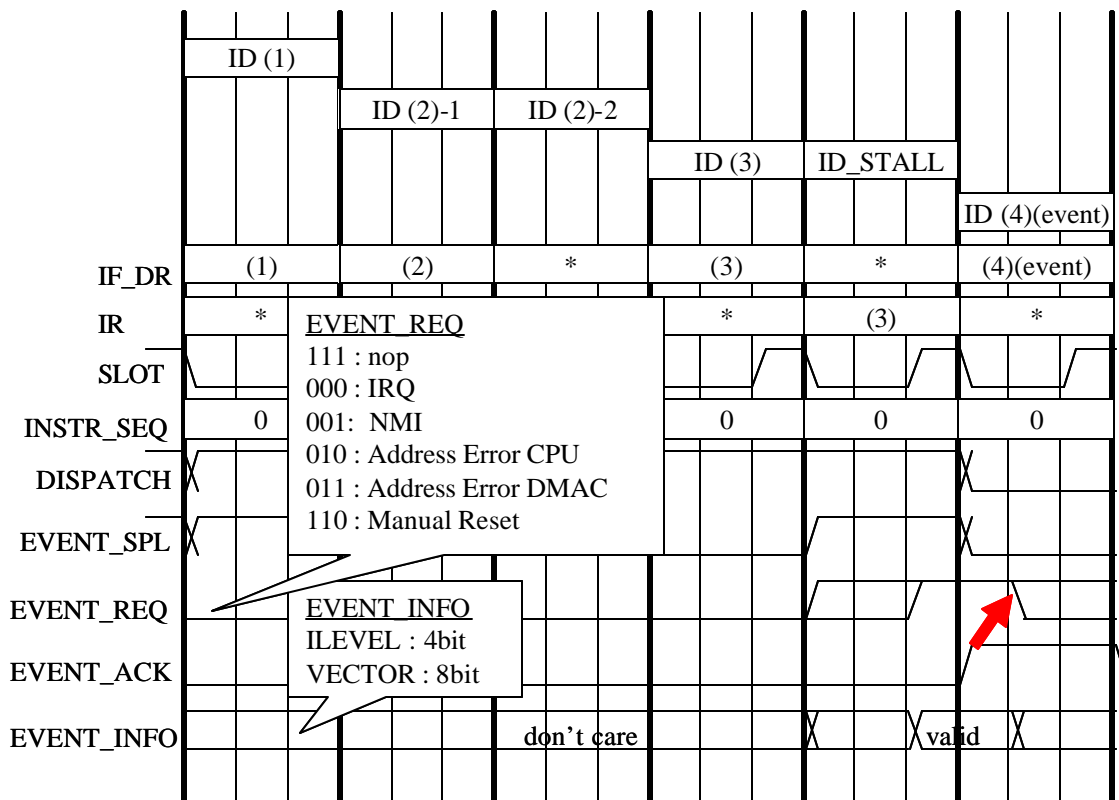
| Hardware Event Information | Meaning                                 | Notes |
|----------------------------|---|-------|
| EVENT_INFO[11:8]           | Priority level of requesting IRQ (4bit) |       |
| EVENT_INFO[7:0]            | Vector No. of requesting IRQ (8bit)     |       |

**Table5.4 Hardware Event Information: EVENT\_INFO[11:0]**

The priority among hardware exceptions should be determined by external circuits, which generates EVENT\_REQ and EVENT\_INFO. In SuperH-2 products, the priority order of exceptions is as follows.

- 1<sup>st</sup>: Power On Reset (Triggered by RST signal.)
- 2<sup>nd</sup>: Manual Reset
- 3<sup>rd</sup>: CPU/DMA Address Error
- 4<sup>th</sup>: NMI
- 5<sup>th</sup>: IRQ

The exceptions caused by instruction (Illegal, Slot-Illegal and Trap) have the lowest priority, but the external circuits need not to care, because the decoder unit in CPU detects them.



**Figure5.1 Hardware Event Request and Sampling Timing**

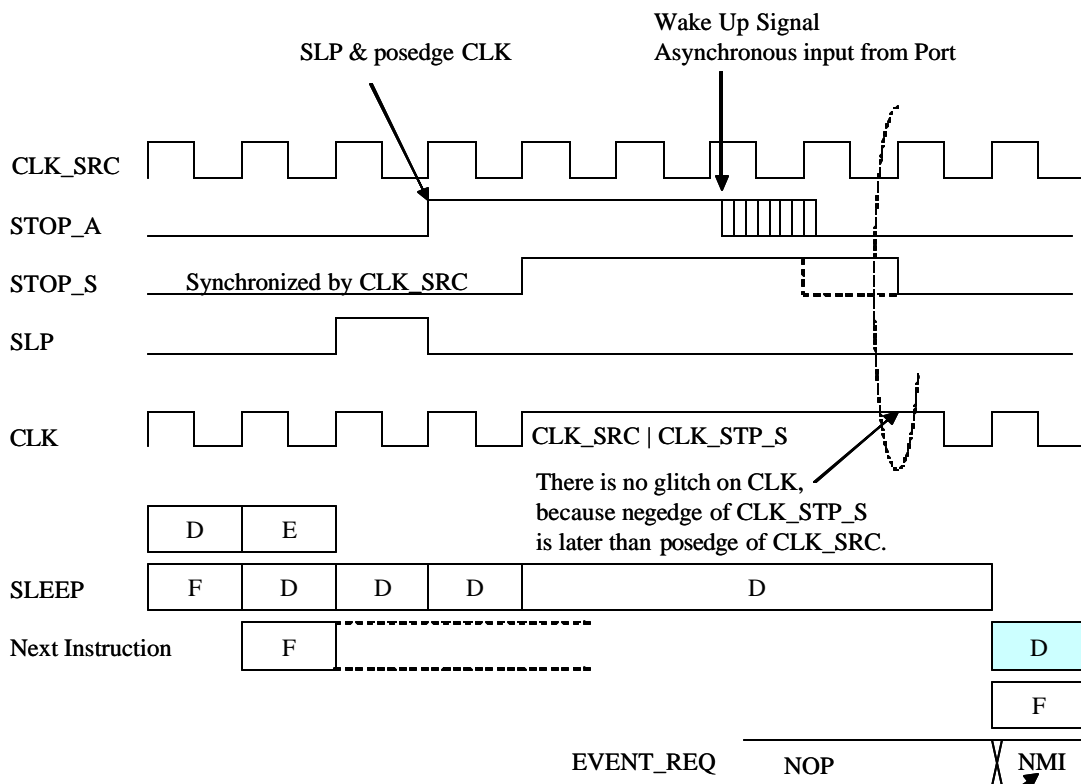
### 5.5. SLEEP signal for Low Power Mode

The SLP output is asserted by SLEEP instruction. The chip can stop its clock by SLP signal and can go to low power mode, if you desire.



The SLEEP timing is shown in Figure5.2. The CLK\_SRC is an original clock generated by, for example, XTAL oscillator. The CLK is made from CLK\_SRC by a gating logic and the CLK stops during SLEEP state. Of course you can stop CLK\_SRC by similar method (But, to wake up CLK\_SRC, you may need some delay timer to wait for the stable XTAL oscillation.) And by some wakeup signal such as NMI, the CLK is waked up. At the wakeup timing, if CPU finds a hardware event request, corresponding exception starts. Or, if there is no hardware event, the program starts from next instruction of SLEEP.

The actual low power mode should be implemented by whole chip designer. The Aquarius test bench includes very simple low power control logic, for your reference.



If you prepare the EVENT\_REQ at this slot, the next instruction of SLEEP is exchanged to the event sequence.

If not, the next instruction of SLEEP continues its operation, after the CLK wakes up. Even in latter case, if you place the opcode for NMI-emulation as the next instruction, you can get desired exception sequence only by the wake-up operation.

**Figure5.2 SLEEP and Low Power Timing**

## 6. Simulation Test Bench

This chapter describes the Aquarius test bench structure for the verification by the method of vector logic simulation.

### 6.1. Top Layer: “top.v”

As shown in Figure3.1, “top.v” is the top layer of Aquarius MCU. It combines among CPU, UART, System Controller, Parallel Port, and on chip memories.

In this chapter, “memory.v” and “lib.v” are assumed to be used, instead of “memory\_fpga.v” and “lib\_fpga.v”.

The system address map is shown in Table6.1. The “memory.v” has 8KB ROM (“rom.v”) and 8KB RAM. All CPU instruction should be verified in various memory access cycle and instruction fetch size. So, the memory access cycle and instruction fetch width are determined by its address; i.e. WISHBONE ACK and TAG0\_I(IF\_WIDTH) are generated in “top.v”.

The peripheral devices such as PIO, UART and SYS are located in 0xABCDxxxx area.

The top layer’s IN/OUT signals are shown in Table6.2. These signals correspond with author’s FPGA configuration. There are several LCD and KEY control signals from PIO module, and UART signals. See later chapter for detail FPGA board circuit.

### 6.2. Simulation Test Bench: “test.v”

The “test.v” is a test bench for Verilog simulation. It creates clock and some input signals (stimuli). Also it generate trace list file as a simulation result named “test\_result.txt”. For your own simulation, please modify “test.v”. When you simulate instructions of CPU by Verilog logic simulator, you need not care the operations of LCD, KEY and UART interfaces. You should care only bus transaction, register contents and signal levels and timings, etc. in case that your viewpoint of simulation is in Aquarius CPU operation.

### 6.3. Parallel I/O Port (PIO): “pio.v”

Parallel I/O Port (PIO) “pio.v” IN/OUT signals are shown in Table6.3. PIO has 2 32bit registers to control Port Pins. Parallel I/O Port (PIO) Registers are shown in Figure6.1. There are 4 byte-size registers for PORT Output and 4 byte-size registers for PORT Input.

Both registers for PORT Input and PORT Output have same address. If you read each register, you can access PORT Input, and if you write to each register, you can access PORT Output.

Each register is located in side-by-side address, so they can be accessed by byte, word or long operand size. PORT Output registers are reset to 0x00 when power on reset.

| Address               | Device                          | Size | Access | IF Width | Notes       |
|-----------------------|---------------------------------|------|--------|----------|-------------|
| 0x00000000-0x00001FFF | ROM                             | 8KB  | 1cyc   | 32bit    | A           |
| 0x00002000-0x00003FFF | RAM                             | 8KB  | 1cyc   | 32bit    | B           |
| 0x00004000-0x0000FFFF | Shadow of 0x00000000-0x00003FFF |      |        |          |             |
| 0x00010000-0x00011FFF | ROM                             | 8KB  | 4cyc   | 32bit    | Shadow of A |
| 0x00012000-0x00013FFF | RAM                             | 8KB  | 4cyc   | 32bit    | Shadow of B |
| 0x00014000-0x0001FFFF | Shadow of 0x00010000-0x00013FFF |      |        |          |             |
| 0x00020000-0x00021FFF | ROM                             | 8KB  | 1cyc   | 16bit    | Shadow of A |
| 0x00022000-0x00023FFF | RAM                             | 8KB  | 1cyc   | 16bit    | Shadow of B |
| 0x00024000-0x0002FFFF | Shadow of 0x00020000-0x00023FFF |      |        |          |             |
| 0x00030000-0x00031FFF | ROM                             | 8KB  | 4cyc   | 16bit    | Shadow of A |
| 0x00032000-0x00033FFF | RAM                             | 8KB  | 4cyc   | 16bit    | Shadow of B |
| 0x00034000-0x0003FFFF | Shadow of 0x00030000-0x00033FFF |      |        |          |             |
| 0x00040000-0xABCCFFFF | Shadow of 0x00000000-0x0003FFFF |      |        |          |             |
| 0xABCD0000-0xABCD00FF | PIO                             | 256B | 4cyc   | 32bit    |             |
| 0xABCD0100-0xABCD01FF | UART                            | 256B | 4cyc   | 32bit    |             |
| 0xABCD0200-0xABCD02FF | SYS                             | 256B | 4cyc   | 32bit    |             |
| 0xABCD0300-0xFFFFFFFF | Shadow of 0x00000000-0x0003FFFF |      |        |          |             |

**Table6.1 Address Map of the Test Bench**

| <b>Class</b>    | <b>Signal Name</b> | <b>Direction</b> | <b>Meaning</b>       | <b>Notes</b> |
|-----------------|--------------------|------------------|----------------------|--------------|
| <b>System</b>   | CLK_SRC            | Input            | System clock         |              |
| <b>Signals</b>  | RST_n              | Input            | Power On Reset       | Negated      |
| <b>Parallel</b> | LCDRS              | Output           | LCD Register Select  | PO[8]        |
| <b>I/O Port</b> | LCDRW              | Output           | LCD Read/Write       | PO[9]        |
|                 | LCDE               | Output           | LCD Enable Signal    | PO[10]       |
|                 | LCDDBO [7:0]       | Output           | LCD Data Bus Output  | PO[7:0]      |
|                 | LCDDBI [7:0]       | Input            | LCD Data Bus Input   | PI[7:0]      |
|                 | KEYYO [4:0]        | Output           | KEY Matrix Y Output  | PO[20:16]    |
|                 | KEYXI [4:0]        | Input            | KEY Matrix X Input   | PI[20:16]    |
| <b>UART</b>     | RXD                | Input            | Receive Serial Data  |              |
|                 | TXD                | Output           | Transmit Serial Data |              |
|                 | CTS                | Input            | Clear To Send        |              |
|                 | RTS                | Output           | Request To Send      |              |

**Table6.2 Top Layer IN/OUT Signals**

| <b>Class</b>    | <b>Signal Name</b> | <b>Direction</b> | <b>Meaning</b>              | <b>Notes</b> |
|-----------------|--------------------|------------------|-----------------------------|--------------|
| <b>System</b>   | CLK                | Input            | System clock                |              |
| <b>Signals</b>  | RST                | Input            | Power On Reset              |              |
| <b>Wishbone</b> | CE                 | Input            | Chip Select (Module Select) | STB          |
| <b>Bus</b>      | WE                 | Input            | Write Enable                |              |
| <b>Signals</b>  | SEL [ 3:0 ]        | Input            | Byte Lane Select            |              |
|                 | DATI [ 31:0 ]      | Input            | Data Input (Write Data)     |              |
|                 | DATO [ 31:0 ]      | Output           | Data Output (Read Data)     |              |
| <b>PORT</b>     | PI [ 31:0 ]        | Input            | Port Input                  |              |
|                 | PO [ 31:0 ]        | Output           | Port Output                 |              |

**Table6.3 Parallel I/O Port (PIO) Module IN/OUT Signals**

|   |          |          |          |          |          |          |          |
|---|----------|----------|----------|----------|----------|----------|----------|
| <b>[PORT Output] Address=0xABCD0000 W only reserved</b>                         |          |          |          |          |          |          |          |
| 31(7)   | 30(6)    | 29(5)    | 28(4)    | 27(3)    | 26(2)    | 25(1)    | 24(0)    |
| reserved  | reserved | reserved | reserved | reserved | reserved | reserved | reserved |
| <b>[PORT Output] Address=0xABCD0001 W only KEYYO (KEY Matrix Y-axis Output)</b> |          |          |          |          |          |          |          |
| 23(7)   | 22(6)    | 21(5)    | 20(4)    | 19(3)    | 18(2)    | 17(1)    | 16(0)    |
| reserved  | reserved | reserved | KY4      | KY3      | KY2      | KY1      | KY0      |
| <b>[PORT Output] Address=0xABCD0002 W only LCDCON (LCD Control Output)</b>      |          |          |          |          |          |          |          |
| 15(7)   | 14(6)    | 13(5)    | 12(4)    | 11(3)    | 10(2)    | 9(1)     | 8(0)     |
| reserved  | reserved | reserved | reserved | reserved | E        | R/W      | RS       |
| <b>[PORT Output] Address=0xABCD0003 W only LCDOUT (LCD Write Data Output)</b>   |          |          |          |          |          |          |          |
| 7(7)  | 6(6)     | 5(5)     | 4(4)     | 3(3)     | 2(2)     | 1(1)     | 0(0)     |
| DW7   | DW6      | DW5      | DW4      | DW3      | DW2      | DW1      | DW0      |
| <b>[PORT Input] Address=0xABCD0000 R only reserved</b>                          |          |          |          |          |          |          |          |
| 31(7)   | 30(6)    | 29(5)    | 28(4)    | 27(3)    | 26(2)    | 25(1)    | 24(0)    |
| reserved  | reserved | reserved | reserved | reserved | reserved | reserved | reserved |
| <b>[PORT Input] Address=0xABCD0001 R only KEYXI (KEY Matrix X-axis Input)</b>   |          |          |          |          |          |          |          |
| 23(7)   | 22(6)    | 21(5)    | 20(4)    | 19(3)    | 18(2)    | 17(1)    | 16(0)    |
| reserved  | reserved | reserved | KX4      | KX3      | KX2      | KX1      | KX0      |
| <b>[PORT Input] Address=0xABCD0002 R only reserved</b>                          |          |          |          |          |          |          |          |
| 15(7)   | 14(6)    | 13(5)    | 12(4)    | 11(3)    | 10(2)    | 9(1)     | 8(0)     |
| reserved  | reserved | reserved | reserved | reserved | E        | R/W      | RS       |
| <b>[PORT Input] Address=0xABCD0003 R only LCDIN (LCD Read Data Input)</b>       |          |          |          |          |          |          |          |
| 7(7)  | 6(6)     | 5(5)     | 4(4)     | 3(3)     | 2(2)     | 1(1)     | 0(0)     |
| DR7   | DR6      | DR5      | DR4      | DR3      | DR2      | DR1      | DR0      |

**Figure6.1 Parallel I/O Port (PIO) Registers**

#### 6.4. Serial I/O (UART): “uart.v”

The top layer has Serial I/O device (UART) “uart.v”, which is SASC (Simple Asynchronous Serial Communication Device) from the opencores.org IP. The SASC is not WISHBONE compliant IP, so some registers are added to connect SASC to WISHBONE bus. UART IN/OUT signals are shown in Table6.4, and its registers are shown in Figure6.2.

Each register is located in side-by-side address, so they can be accessed by byte, word or long operand size, but the UARTCON and UARTRXD/TXD should be accessed only by byte operand size.

| <b>Class</b>    | <b>Signal Name</b> | <b>Direction</b> | <b>Meaning</b>              | <b>Notes</b> |
|-----------------|--------------------|------------------|-----------------------------|--------------|
| <b>System</b>   | CLK                | Input            | System clock                |              |
| <b>Signals</b>  | RST                | Input            | Power On Reset              |              |
| <b>Wishbone</b> | CE                 | Input            | Chip Select (Module Select) | STB          |
| <b>Bus</b>      | WE                 | Input            | Write Enable                |              |
| <b>Signals</b>  | SEL[ 3:0 ]         | Input            | Byte Lane Select            |              |
|                 | DATI[ 31:0 ]       | Input            | Data Input (Write Data)     |              |
|                 | DATO[ 31:0 ]       | Output           | Data Output (Read Data)     |              |
| <b>UART</b>     | RXD                | Input            | Receive Serial Data         |              |
|                 | TXD                | Output           | Transmit Serial Data        |              |
|                 | CTS                | Input            | Clear To Send               |              |
|                 | RTS                | Output           | Request To Send             |              |

**Table6.4 Serial I/O (UART) IN/OUT Signals**

|   |          |          |          |          |          |       |       |
|---|----------|----------|----------|----------|----------|-------|-------|
| <b>[UART] Address=0xABCD0100 R/W UARTBG0 (Baud rate Generator Div0)</b>   |          |          |          |          |          |       |       |
| 31(7)   | 30(6)    | 29(5)    | 28(4)    | 27(3)    | 26(2)    | 25(1) | 24(0) |
| B07   | B06      | B05      | B04      | B03      | B02      | B01   | B00   |
| <b>[UART] Address=0xABCD0101 R/W UARTBG1 (Baud rate Generator Div1)</b>   |          |          |          |          |          |       |       |
| 23(7)   | 22(6)    | 21(5)    | 20(4)    | 19(3)    | 18(2)    | 17(1) | 16(0) |
| B17   | B16      | B15      | B14      | B13      | B12      | B11   | B10   |
| <b>[UART] Address=0xABCD0102 R only UARTCON (TXF=full_o, RXE=empty_o)</b> |          |          |          |          |          |       |       |
| 15(7)   | 14(6)    | 13(5)    | 12(4)    | 11(3)    | 10(2)    | 9(1)  | 8(0)  |
| reserved  | reserved | Reserved | reserved | reserved | reserved | TXF   | RXF   |
| <b>[UART] Address=0xABCD0103 R only / UARTRXD, W only / UARTTXD</b>       |          |          |          |          |          |       |       |
| 7(7)  | 6(6)     | 5(5)     | 4(4)     | 3(3)     | 2(2)     | 1(1)  | 0(0)  |
| TR7   | TR6      | TR5      | TR4      | TR3      | TR2      | TR1   | TR0   |

**Figure6.2 Serial I/O (UART) Registers**

The UARTBG0 and UARTBG1 are the registers to determine the serial baud rate. The

UARTBG0 and UARTBG1 are reset to 0x00 when power on reset. The expression to calculate the baud rate is shown below.

$$BaudRate = \frac{f(CLK)}{4} \times \frac{1}{(BG0 + 2) \times (BG1 + 1)} [bps]$$

Table6.5 shows some examples of baud rate setting.

| Baud Rate<br>[bps] | f (CLK)<br>[MHz] | UARTBG0   | UARTBG1    | Notes |
|--------------------|------------------|-----------|------------|-------|
| 1200               | 20               | 0x12 (18) | 0xCF (207) |       |
| 2400               | 20               | 0x12 (18) | 0x67 (103) |       |
| 4800               | 20               | 0x12 (18) | 0x33 (51)  |       |
| 9600               | 20               | 0x12 (18) | 0x19 (25)  |       |

**Table6.5 Examples of Baud Rate Settings**

The UARTCON has 2 flags; TXF and RXE. The TXF is 1 when transmit buffer is full. If TXF=0, you can write next transmit data. The RXE is 1 when receive buffer is empty. If RXE=0, you can read receive data. The TXF and RXE correspond to full\_o and empty\_o of SASC, respectively. Note that SASC has 4 byte depth FIFOs for both transmit buffer and receive buffer. In case of this top layer, TXF and RXE are not connected as interrupt signals, so you should poll these flags in your program. Generally, such flags should be treated as interrupt requests. You can easily modify the Aquarius RTL codes like this.

The UARTRXD and UARTTXD are the receive buffer and transmit buffer registers, which have same address. Read operation accesses to UARTRXD, and Write operation accesses to UARTTXD.

### 6.5. System Controller (SYS): “sys.v”

The System Controller (SYS) “sys.v” has following functions.

- (1) Generate Exception of Hardware Event.
  - NMI (by Address Break)
  - IRQ (by Interval Timer)
  - CPU Address Error (by watching WISHBONE bus transaction)
- (2) Emulate Exception of Hardware Event.

- NMI
  - IRQ
  - CPU Address Error
  - DMA Address Error
  - Manual Reset
- (3) Control priority level among the requests of hardware exception.
  - (4) Set IRQ priority level and vector number.
  - (5) 12bit Interval Timer to generate IRQ.
  - (6) Bus Address Break Function for debugging capability (NMI).
  - (7) Detect CPU Address Error by watching WISHBONE bus signals.
  - (8) SLEEP and Low Power Control, according to Figure5.2 manner.

The IN/OUT Signals of SYS are shown in Table6.6.

The SYS has 2 32bit length registers; INTCTL and BRKADR. These are shown in Figure6.3. Both registers should be accessed only by long word operand size. The INTCTL is reset to 0x00000FFF, and the BRKADR is reset to 0x00000000 when power on reset.

| <b>Class</b>    | <b>Signal Name</b>   | <b>Direction</b> | <b>Meaning</b>               | <b>Notes</b>                |
|-----------------|----------------------|------------------|------------------------------|-----------------------------|
| <b>System</b>   | CLK_SRC              | Input            | System clock Source          |                             |
| <b>Signals</b>  | CLK                  | Output           | CLK , which stops at SLEEP   |                             |
|                 | SLP                  | Input            | SLEEP request from CPU       |                             |
|                 | WAKEUP               | Input            | Wakeup Request               |                             |
|                 | RST                  | Input            | Power On Reset               |                             |
|                 | <b>Wishbone</b>      | CE               | Input                        | Chip Select (Module Select) |
| <b>Bus</b>      | WE                   | Input            | Write Enable                 |                             |
| <b>Signals</b>  | SEL[ 3 : 0 ]         | Input            | Byte Lane Select             |                             |
|                 | ACK                  | Input            | Bus Acknowledge              |                             |
|                 | DATI[ 31 : 0 ]       | Input            | Data Input (Write Data)      |                             |
|                 | DATO[ 31 : 0 ]       | Output           | Data Output (Read Data)      |                             |
|                 | STB                  | Input            | Strobe (Bus monitor to BRK)  |                             |
|                 | ADR[ 31 : 0 ]        | Input            | Address (Bus monitor to BRK) |                             |
| <b>Hardware</b> | EVENT_REQ[ 2 : 0 ]   | Output           | Event Request                |                             |
| <b>Events</b>   | EVENT_INFO[ 11 : 0 ] | Output           | Event Information (IRQ)      |                             |
|                 | EVENT_ACK            | Input            | Event Acknowledge from CPU   |                             |

**Table6.6 System Controller (SYS) IN/OUT Signals**



**[SYS] Address=0xABCD0200 R/W INTCON (Interrupt Control)**

|       |       |       |       |       |          |       |       |
|-------|-------|-------|-------|-------|----------|-------|-------|
| 31    | 30    | 29    | 28    | 27    | 26       | 25    | 24    |
| E_NMI | E_IRQ | E_CER | E_DER | E_MRS | reserved | TMRON | BRKON |
| 23    | 22    | 21    | 20    | 19    | 18       | 17    | 16    |
| ILVL3 | ILVL2 | ILVL1 | ILVL0 | IVEC7 | IVEC6    | IVEC5 | IVEC4 |
| 15    | 14    | 13    | 12    | 11    | 10       | 9     | 8     |
| IVEC3 | IVEC2 | IVEC1 | IVEC0 | TMR11 | TMR10    | TMR9  | TMR8  |
| 7     | 6     | 5     | 4     | 3     | 2        | 1     | 0     |
| TMR7  | TMR6  | TMR5  | TMR4  | TMR3  | TMR2     | TMR1  | TMR0  |

**[SYS] Address=0xABCD0204 R/W BRKADR (Break Address)**

|       |       |       |       |       |       |       |       |
|-------|-------|-------|-------|-------|-------|-------|-------|
| 31    | 30    | 29    | 28    | 27    | 26    | 25    | 24    |
| ADR31 | ADR30 | ADR29 | ADR28 | ADR27 | ADR26 | ADR25 | ADR24 |
| 23    | 22    | 21    | 20    | 19    | 18    | 17    | 16    |
| ADR23 | ADR22 | ADR21 | ADR20 | ADR19 | ADR18 | ADR17 | ADR16 |
| 15    | 14    | 13    | 12    | 11    | 10    | 9     | 8     |
| ADR15 | ADR14 | ADR13 | ADR12 | ADR11 | ADR10 | ADR9  | ADR8  |
| 7     | 6     | 5     | 4     | 3     | 2     | 1     | 0     |
| ADR7  | ADR6  | ADR5  | ADR4  | ADR3  | ADR2  | ADR1  | ADR0  |

**Figure6.3 System Controller (SYS) Registers**

INTCTL : Interrupt Control Register

- E\_NMI            Emulate NMI. Write only bit. Read 0 only.  
When you write 1, NMI exception sequence will start.
- E\_IRQ            Emulate IRQ. Write only bit. Read 0 only.  
When you write 1, IRQ exception sequence will start if the IRQ priority level is higher than I bit in SR.  
The priority level and the vector number of the IRQ is specified by ILVL3-ILVL0 and IVEC7-IVEC0 bits in INTCTL register.
- E\_CER            Emulate CPU Address Error. Write only bit. Read 0 only.  
When you write 1, CPU Address Error exception will start.
- E\_DER            Emulate DMA Address Error. Write only bit. Read 0 only.

|             |   |
|-------------|---|
|             | When you write 1, DMA Address Error exception will start.                                   |
| E_MRES      | Emulate Manual Reset. Write only bit. Read 0 only.  |
|             | When you write 1, Manual Reset exception will start.  |
| TMRON       | When 1, 12 bit Interval Timer starts.   |
|             | When 0, the Interval Timer stops.   |
| BRKON       | When 1, start to compare BRKADR with WISHBONE address, and if these are equal, request NMI. |
| ILVL3-ILVL0 | IRQ priority level to be requested (makes EVENT_INFO[11:8])                                 |
| IVEC7-IVEC0 | IRQ vector number to be requested (makes EVENT_INFO[7:0])                                   |
| TMR11-TMR0  | 12 bit Interval Timer. When 0x000, it requests IRQ.   |

BRKADR : Break Address Register

|            |   |
|------------|---|
| ADR31-ADR0 | Break address to be compared to WISHBONE address.<br>It is valid only when BRKON=1. |
|------------|---|

### 6.6. On Chip Memory: “memory.v”

The memory module “memory.v” has 8KB ROM and 8KB RAM. The address map has been shown in Table6.1. The bit pattern of ROM is specified by “rom.v” description. The memory module’s IN/OUT signals are shown in Table6.7.

| <i>Class</i>    | <i>Signal Name</i> | <i>Direction</i> | <i>Meaning</i>              | <i>Notes</i> |
|-----------------|--------------------|------------------|-----------------------------|--------------|
| <b>System</b>   | CLK                | Input            | System clock                |              |
| <b>Signals</b>  | RST                | Input            | Power On Reset              |              |
| <b>Wishbone</b> | CE                 | Input            | Chip Select (Module Select) | STB          |
| <b>Bus</b>      | WE                 | Input            | Write Enable                |              |
| <b>Signals</b>  | SEL[ 3:0 ]         | Input            | Byte Lane Select            |              |
|                 | ADR[ 13:0 ]        | Input            | Address                     |              |
|                 | DATI[ 31:0 ]       | Input            | Data Input (Write Data)     |              |
|                 | DATO[ 31:0 ]       | Output           | Data Output (Read Data)     |              |

**Table6.7 On-Chip Memory IN/OUT Signals**

## 6.7. Simulation Tools and Flows

(1) Have you already installed all tools such as Cygwin, GNU binutils, GNU C compiler and Aquarius deliverables for program development and Verilog simulation?

In following explanations, I assume the tool placing is like below.

```
~      (home directory)
|-----CPU      (directory)
|-----*.v      (Verilog Sources)
|-----test.txt (list of Verilog sources for simulation)
|-----sim      (simulation launch script)
|-----asm      (assembler launch script)
|-----sha_testsource      (directory)
|-----testalu.src      (ALU check program)
```

(2) To simulate instruction or program, make your assembler source file. Some examples are located in the directory "sha\_testsource" of Aquarius deliverables. In these examples, basically, all instruction sequence to be verified are simulated on all memory space attributes among combinations of no-wait or with-wait, and 32bit or 16bit instruction fetch area (IF\_WIDTH). Now, suppose you are trying "testalu.src" to check ALU functions.

(3) Assemble it. From your Cygwin console window, type...

```
$ cd ~/CPU
$ ./asm sha_testsource/testalu.src
```

If no errors, you will find following files.

```
lis      assembler list file
obj      s-format object file
rom.v    Verilog ROM description
```

(4) Prepare "test.txt" in which Verilog source file names are listed as follows.

```
// source file list      |      datapath.v      |      cpu.v
defines.v                |      mult.v           |      rom.v
timescale.v              |      decode.v          |      memory.v
register.v                |      mem.v             |      pio.v
```

```

sasc_brg.v          |          uart.v          |          top.v
sasc_fifo4.v       |          lib.v           |          test.v
sasc_top.v         |          sys.v           |

```

Ok, now you can simulate Aquarius Verilog RTL codes. Type...

```
$ ./sim
```

If no errors, you will find following file.

```
test_result.txt  simulation result trace list
```

(5) Check this file. Are you success?

The "test\_result.txt" is like this.

---

```

COUNT#  CR CSAWI SEL-  ADR----  DATI----  DATO----  PC-----  EVR  EVI  A|S-INST-Q-D-IFDR-IR--...
00000000  00  xxxxx  xxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxx  xxx  x|x  xxxxx  x  x  xxxxx  xxxxx...
00000001  01  xxxxx  xxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx  111  000  1|1  f700  1  0  xxxxx  f700...
00000002  01  0001x  xxxx  xxxxxxxx  00000000  xxxxxxxx  xxxxxxxx  111  000  1|1  f700  1  0  xxxxx  f700...
00000003  01  0001x  xxxx  xxxxxxxx  00000000  xxxxxxxx  xxxxxxxx  111  000  0|1  f700  2  0  xxxxx  f700...
00000004  01  0001x  xxxx  xxxxxxxx  00000000  xxxxxxxx  xxxxxxxx  111  000  0|1  f700  3  0  xxxxx  f700...
00000005  01  11101  1111  00000000  00000400  xxxxxxxx  xxxxxxxx  111  000  0|1  f700  4  0  xxxxx  f700...
00000006  01  11101  1111  00000004  fffd0000  xxxxxxxx  xxxxxxxx  111  000  0|1  f700  5  0  xxxxx  f700...
00000007  01  11101  1111  00000400  ee00dd01  xxxxxxxx  00000400  111  000  0|1  f700  6  1  xxxxx  f700...
.....

```

---

This file is created by \$fdisplay() statement in test bench script "test.v", and shows WISHBONE bus signals, CPU internal buses and registers et al. in trace list manner. You can modify "test.v" to see other signals.

The simulation stop condition is determined by simulation clock cycle counts in "test.v" description. This "test.v" is one of the examples for you, so you may modify it for your favorite simulation.

## 7. FPGA Implementation

This chapter shows you my FPGA system and Aquarius implementation to FPGA.

### 7.1. FPGA System

As described before, I have been using existing FPGA board XSP-009-300 manufactured by HuMANDATA, Ltd., which has one Xilinx VirtexE XCV300E. This board also has configuration circuit by JTAG or FLASH ROM, and power supply circuit. I think you can find similar FPGA boards from many vendors around you.

I made a FPGA System by connecting handmade interface board, which has LCD display, KEY matrix and RS-232C interface.

Figure 7.1 shows the whole view of FPGA verification system. Figure 7.2 shows both interface board and FPGA board. These 2 boards are connected back to back each other. The system block diagram is Figure7.3.

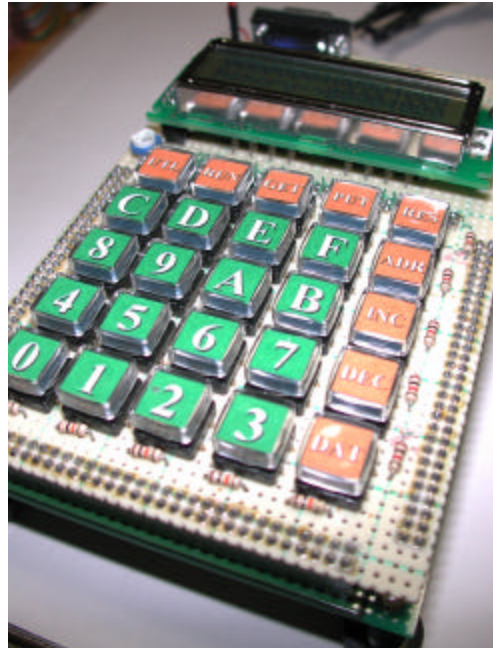
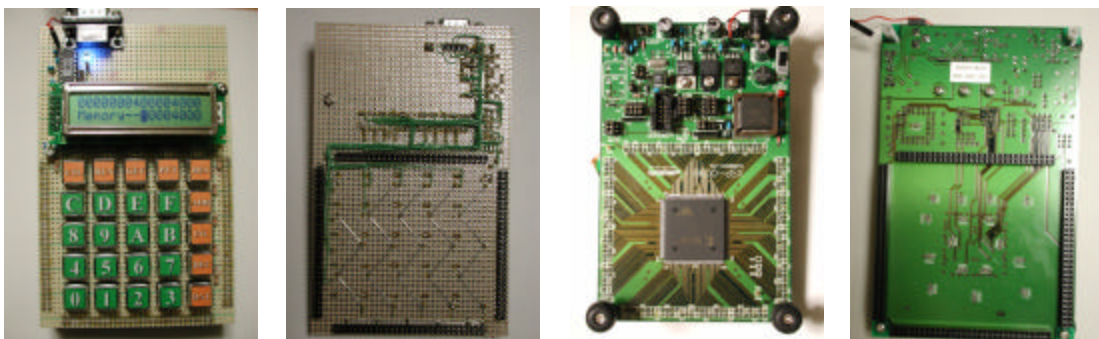
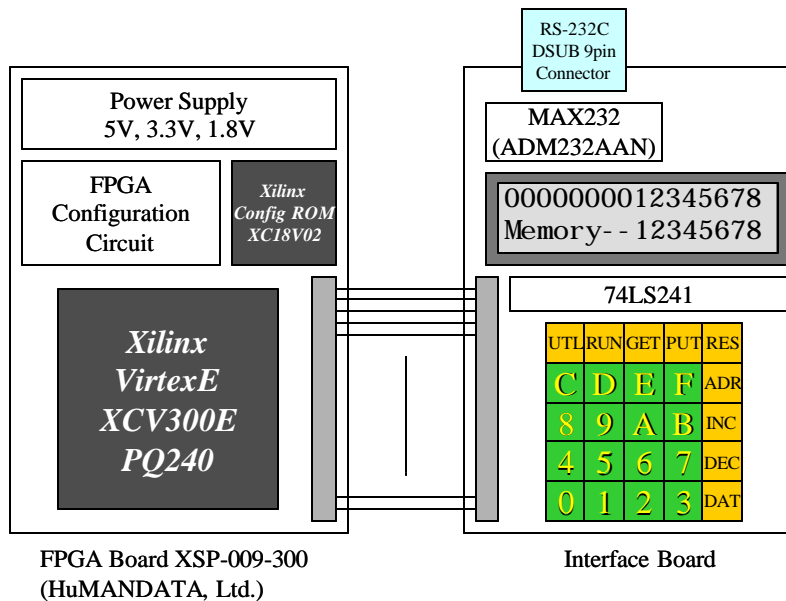


Figure7.1 FPGA Verification System



(A) I/F board (B) I/F Board (bottom) (C) FPGA board (D) FPGA board (bottom)

Figure7.2 Picture of Each Board



**Figure7.3 Block diagram of FPGA System**

## 7.2. Circuit of FPGA Board

The circuit schematic of FPGA Board (XSP-009-300) is found in following URL; <http://www.hdl.co.jp/ftpdata/xsp-009/XSP009.sch.pdf>. In my case, the FPGA operating frequency is set to 20MHz.

## 7.3. Circuit of Interface Board

Figure 7.4 shows the circuit of Interface Board.

### (1) LCD Display Interface

As LCD character display, I use SUNLIKE 16 columns x 2 rows LCD Display SC-1602B. It operates by commands via its bus interface. You can find detail documents regarding mechanical data, electrical characteristics, initialization methods and operation commands from <http://www.lcd-modules.com.tw/>.

The bus interface is bi-direction, so, I use 74LS241 buffers to make interface with the FPGA. Note that 100ohm resistors are inserted between 74LS241 output and FPGA input because the FPGA don't have 5V tolerant input buffer. Xilinx recommends using current limit resistor at 5V signal input.

### (2) RS-232C Interface

To implement the RS-232C Interface, I adopt the MAX232 compatible IC ANALOG DEVICES ADM232AAN. The FPGA interface also needs 5V tolerant resistors. The DSUB-9 connector is linked supposing cross cable.

### (3) KEY Matrix Interface

The Key Matrix Interface has 25 keys to input hex data, some commands and reset. The 1Kohm resistors are necessary to avoid conflict on FPGA output pins when multiple keys are pushed. Instead of 1Kohm resistors, it is good idea that you use discrete diodes, connecting each anode to switch and cathode to FPGA port.

### (4) FPGA Pin Configuration

In case of above FPGA System, the FPGA's pin configuration that corresponds to "top.v" is as follows (Also refer to Table6.2). These statements should be described in User Constraints File (.ucf) before you configure the FPGA.

---

```
NET "CLK_SRC" LOC = "p92";
NET "RST_n" LOC = "p42";
NET "TXD" LOC = "p46";
NET "RXD" LOC = "p47";
NET "RTS" LOC = "p48";
NET "CTS" LOC = "p49";
NET "LCDRW" LOC = "p4";
NET "LCDRS" LOC = "p3";
NET "LCDE" LOC = "p5";
NET "LCDDBO<7>" LOC = "p17";
NET "LCDDBO<6>" LOC = "p13";
NET "LCDDBO<5>" LOC = "p12";
NET "LCDDBO<4>" LOC = "p11";
NET "LCDDBO<3>" LOC = "p10";
NET "LCDDBO<2>" LOC = "p9";
NET "LCDDBO<1>" LOC = "p7";
NET "LCDDBO<0>" LOC = "p6";
NET "LCDDBI<7>" LOC = "p27";
NET "LCDDBI<6>" LOC = "p26";
NET "LCDDBI<5>" LOC = "p24";
NET "LCDDBI<4>" LOC = "p23";
NET "LCDDBI<3>" LOC = "p21";
NET "LCDDBI<2>" LOC = "p20";
NET "LCDDBI<1>" LOC = "p19";
NET "LCDDBI<0>" LOC = "p18";
NET "KEYYO<4>" LOC = "p28";
NET "KEYYO<3>" LOC = "p31";
NET "KEYYO<2>" LOC = "p33";
NET "KEYYO<1>" LOC = "p34";
NET "KEYYO<0>" LOC = "p35";
NET "KEYXI<4>" LOC = "p41";
NET "KEYXI<3>" LOC = "p40";
NET "KEYXI<2>" LOC = "p39";
NET "KEYXI<1>" LOC = "p38";
NET "KEYXI<0>" LOC = "p36";
```

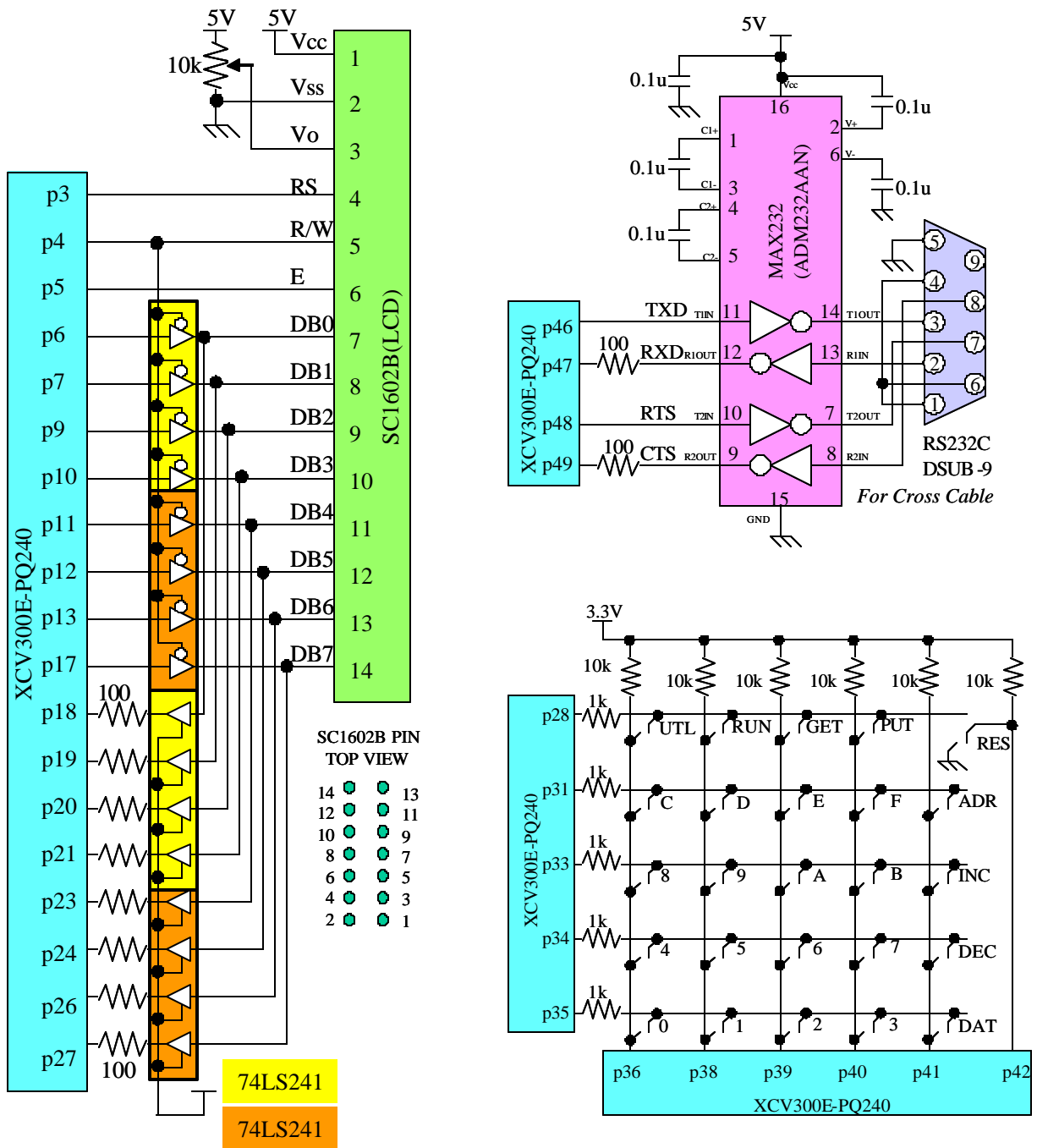


Figure7.4 Circuit of Interface Board



## 7.4. FPGA Configuration

Launch the Xilinx ISE Webpack 5.x, select the device to yours, and add following Verilog sources to your project.

|                         |                            |                          |
|-------------------------|----------------------------|--------------------------|
| <code>cpu.v</code>      | <code>memory_fpga.v</code> | <code>sasc_top.v</code>  |
| <code>datapath.v</code> | <code>mult.v</code>        | <code>sys.v</code>       |
| <code>decode.v</code>   | <code>pio.v</code>         | <code>test.v</code>      |
| <code>defines.v</code>  | <code>register.v</code>    | <code>timescale.v</code> |
| <code>lib_fpga.v</code> | <code>sasc_brg.v</code>    | <code>top.v</code>       |
| <code>mem.v</code>      | <code>sasc_fifo4.v</code>  | <code>uart.v</code>      |

Make user constraints file (`top.ucf`) to specify pin assignment, timing constraints and BlockRAM initial value. In Aquarius deliverables, I prepare an example file `top.ucf`. To initialize contents of BlockRAM, use "genram" utility described before and append INST statements to `top.ucf` (default `top.ucf` already have INST statement, so you should replace all INST statements to new ones generated by "genram".)

Ok, then compile from the "top" module, and configure your FPGA.

## 7.5. Results of FPGA Configuration

### (1) Xilinx VirtexE (XCV300E)

Regarding the FPGA system mentioned above, Table7.1 shows the performance results by Xilinx VirtexE XCV300E-8PQ240, which has 3072 slices. On chip memories are configured by BlockRAM. Under the speed-priority synthesizing, total usage of logic slices is beyond the device, unfortunately. In author's FPGA System, although the device has been configured by area-priority synthesis, the device operates 20MHz frequency under the typical condition (power supply voltage and ambient temperature).

### (2) Altera Stratix (EP1S10)

For technical reference, I tried to configure Aquarius into Altera Stratix EP1S10, which has 10570 logic elements. Table7.2 shows the summary. And the detail utilization of logic cells is shown in Table7.3. In this case, on chip memories are implemented by Synchronous SRAM components and the multiplier (in `mult.v`) is implemented by internal DSP unit.

| Synthesis    | Top   | Slices | Consumed | Frequency | Notes             |
|--------------|-------|--------|----------|-----------|-------------------|
| <b>Area</b>  | top.v | 2923   | 95%      | 15MHz     |                   |
| <b>20MHz</b> | cpu.v | 2635   | 86%      | 15MHz     |                   |
| <b>Speed</b> | top.v | 3135   | 102%     | 25MHz     | XCV300E overflows |
| <b>20MHz</b> | cpu.v | 2753   | 90%      | 21MHz     |                   |

**Table7.1 Results of Xilinx VertexE (XCV300E-8PQ240) with Webpack ISE 5.2 (SP3)**

| Synthesis             | Top   | Cells | Consumed | Frequency | Notes |
|-----------------------|-------|-------|----------|-----------|-------|
| <b>Normal</b>         | top.v | 7919  | 75%      | 31MHz     |       |
| <b>No constraints</b> | cpu.v | 7499  | 71%      | 31MHz     |       |

**Table7.2 Results of Altera Stratix (EP1S10F780C5ES) with Quartus II 2.2 Web Edition (SP2)**

| Compilation Hierarchy Node       | Logic Cells | Registers | Memory Bits | DSP Elements | DSP 9x9 | DSP 18x18 | DSP 36x36 | Pins | Virtual Pins | LUT-Only LCS | Register-Only LCS | LUT/Registers LCS |
|----------------------------------|-------------|-----------|-------------|--------------|---------|-----------|-----------|------|--------------|--------------|-------------------|-------------------|
| top                              | 7919 (41)   | 1458      | 131072      | 8            | 0       | 0         | 1         | 36   | 0            | 6461 (37)    | 102 (1)           | 1356 (3)          |
| cpu:CPU                          | 7499 (0)    | 1207      | 0           | 8            | 0       | 0         | 1         | 0    | 0            | 6292 (0)     | 31 (0)            | 1176 (0)          |
| datapath:DATAPATH                | 5309 (2508) | 681       | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 4628 (2370)  | 0 (0)             | 681 (138)         |
| lpm_counter:PC_rtl_0             | 31 (0)      | 31        | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 31 (0)            |
| alt_counter_stratix:wysi_counter | 31 (31)     | 31        | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 31 (31)           |
| register:REGISTER                | 2770 (2770) | 512       | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 2258 (2258)  | 0 (0)             | 512 (512)         |
| decode:DECODE                    | 827 (827)   | 183       | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 644 (644)    | 26 (26)           | 157 (157)         |
| mem:MEM                          | 371 (371)   | 179       | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 192 (192)    | 5 (5)             | 174 (174)         |
| mult:MULT                        | 992 (992)   | 164       | 0           | 8            | 0       | 0         | 1         | 0    | 0            | 828 (828)    | 0 (0)             | 164 (164)         |
| lpm_mult:mult_422                | 0 (0)       | 0         | 0           | 8            | 0       | 0         | 1         | 0    | 0            | 0 (0)        | 0 (0)             | 0 (0)             |
| mult_lhj:auto_generated          | 0 (0)       | 0         | 0           | 8            | 0       | 0         | 1         | 0    | 0            | 0 (0)        | 0 (0)             | 0 (0)             |
| memory:MEMORY                    | 17 (0)      | 0         | 131072      | 0            | 0       | 0         | 0         | 0    | 0            | 17 (0)       | 0 (0)             | 0 (0)             |
| ram:RAM                          | 17 (17)     | 0         | 131072      | 0            | 0       | 0         | 0         | 0    | 0            | 17 (17)      | 0 (0)             | 0 (0)             |
| lpm_ram_dq:LPM_RAM_DQ_HH         | 0 (0)       | 0         | 32768       | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 0 (0)             |
| altsyncram:altsyncram_component  | 0 (0)       | 0         | 32768       | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 0 (0)             |
| lpm_ram_dq:LPM_RAM_DQ_HL         | 0 (0)       | 0         | 32768       | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 0 (0)             |
| altsyncram:altsyncram_component  | 0 (0)       | 0         | 32768       | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 0 (0)             |
| lpm_ram_dq:LPM_RAM_DQ_LH         | 0 (0)       | 0         | 32768       | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 0 (0)             |
| altsyncram:altsyncram_component  | 0 (0)       | 0         | 32768       | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 0 (0)             |
| lpm_ram_dq:LPM_RAM_DQ_LL         | 0 (0)       | 0         | 32768       | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 0 (0)             |
| altsyncram:altsyncram_component  | 0 (0)       | 0         | 32768       | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 0 (0)             |
| pio:PIO                          | 33 (33)     | 16        | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 17 (17)      | 16 (16)           | 0 (0)             |
| sys:SYS                          | 101 (101)   | 63        | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 38 (38)      | 1 (1)             | 62 (62)           |
| uart:UART                        | 228 (43)    | 168       | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 60 (17)      | 53 (9)            | 115 (17)          |
| sasc_brg:BRG                     | 33 (31)     | 25        | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 8 (8)        | 2 (2)             | 23 (21)           |
| lpm_counter:cnt_rtl_0            | 2 (0)       | 2         | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 2 (0)             |
| alt_counter_stratix:wysi_counter | 2 (2)       | 2         | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 2 (2)             |
| sasc_top:TOP                     | 152 (63)    | 117       | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 35 (20)      | 42 (10)           | 75 (33)           |
| sasc_fifo4:rx_fifo               | 46 (42)     | 37        | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 9 (9)        | 16 (16)           | 21 (17)           |
| lpm_counter:rp_rtl_0             | 2 (0)       | 2         | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 2 (0)             |
| alt_counter_stratix:wysi_counter | 2 (2)       | 2         | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 2 (2)             |
| lpm_counter:wp_rtl_0             | 2 (0)       | 2         | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 2 (0)             |
| alt_counter_stratix:wysi_counter | 2 (2)       | 2         | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 2 (2)             |
| sasc_fifo4:tx_fifo               | 43 (39)     | 37        | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 6 (6)        | 16 (16)           | 21 (17)           |
| lpm_counter:rp_rtl_0             | 2 (0)       | 2         | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 2 (0)             |
| alt_counter_stratix:wysi_counter | 2 (2)       | 2         | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 2 (2)             |
| lpm_counter:wp_rtl_0             | 2 (0)       | 2         | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 2 (0)             |
| alt_counter_stratix:wysi_counter | 2 (2)       | 2         | 0           | 0            | 0       | 0         | 0         | 0    | 0            | 0 (0)        | 0 (0)             | 2 (2)             |

**Table7.3 Detail Utilization of Logic Cells in Altera Stratix EP1S10**

## 7.6. Application Programs on the FPGA System

I include some simple application programs in Aquarius deliverables. All applications are developed by GNU C compiler for SuperH-2.

Each startup program (`cr0.s`) and linker script (`sh.x`) is located in directory “startup” under each application directory. The “Makefile” is prepared for all applications so as to compile and link by typing “\$ make”, and to cleanup objects by typing “\$ make clean”.

In my FPGA system, the BlockRAM contains all application provided here. All applications are combined into one object file “`ram.srec`”. You can make BlockRAM's INIT statements by “`genram`” utility.

(1) **Monitor Program:** directory “`shc_monitor_release_v1`”

Using LCD display, key board and RS-232C I/F, this monitor program has very basic debug capability such as Memory Editor, Program Loader from PC, Jumping to Program and Debugging utilities such as Setting a Break point and Reading Registers. The source code is “`main.c`”. This program is located from address 0x00000000 (here is vector table). It starts by power on reset. Below, I simply explain an example session of the monitor. Please refer Figure7.5.

(A) Memory Editor

(a) Startup

The top line shows memory address and its data. Left 8 hex number is address. Right 8 hex number is data. Always shows only long-word sized data.

The bottom line has 4 byte entry space. You can enter new hex number here.

(b) Address Increment

The “INC” key increases address by 4 byte, and shows the data at new address.

(c) Address Decrement

The “DEC” key decreases address by -4 byte, and shows the data at new address.

(d) Enter Address

(e) Set Address

If you want to see another address of memory, enter new 4 byte address in bottom line, and push “ADR” key. If your entry is not in multiples of 4, lower 2 bits of address are cleared to 0, to avoid address error.

- (f) Enter Data to be written
- (g) Write Data and Increment Address

If you want to change data in displaying address, enter new 4 byte data in bottom line, and push “DAT” key. Then the memory content is updated and the displayed address increases 4 byte.

- (h) Verify

Use decrement key (and also increment key) to verify the memory contents.

|  |  |   |
|--|--|---|
| <b>000000000000400</b><br><b>Memory-- 0000400</b>    | <b>000000400004000</b><br><b>Memory-- 00004000</b>       | <b>FFFFFFFFC000005D0</b><br><b>Memory-- 000005D0</b>  |
| (a) When Startup                                     | (b) “INC” Address Increment                              | (c) “DEC” Address Decrement                           |
| <b>FFFFFFFFC000005D0</b><br><b>Memory-- 00003800</b> | <b>00003800FFFFFFFF</b><br><b>Memory-- FFFFFFFF</b>      | <b>00003800FFFFFFFF</b><br><b>Memory-- 12345678</b>   |
| (d) Enter 32bit Hex (address)                        | (e) “ADR” Address Set                                    | (f) Enter 32bit Hex (data)                            |
| <b>00003804FFFFFFFF</b><br><b>Memory-- FFFFFFFF</b>  | <b>0000380012345678</b><br><b>Memory-- 12345678</b>      | <b>Get S-Format (S3)</b><br><b>Please send. . . .</b> |
| (g) “DAT” Write & increment                          | (h) “DEC” Verify, OK!                                    | (i) “GET” Wait for Program Load                       |
| <b>Get S-Format (S3)</b><br><b>00003190-----OK!</b>  | <b>0000300000003008</b><br><b>Memory-- 00003008</b>      | <b>00003008D805480B</b><br><b>Memory-- D805480B</b>   |
| (j) Loading Program                                  | (k) Finish Loading Program                               | (l) “ADR” Set Branch Target                           |
| <b>00003008D805480B</b><br><b>Run-- Good Luck!</b>   | <b>&lt;SuperH in FPGA&gt;</b><br><b>@ABCDEFGHIJKLMNO</b> | <b>BRK- Func Select?</b><br><b>1: REG 2: BRK SET</b>  |
| (m) “RUN” Go !                                       | (n) Running Program                                      | (o) “UTL” Break Function Select                       |
| <b>Set Break Point.</b><br><b>Address?00003800</b>   | <b>Break Accepted.</b><br><b>Address?00003800</b>        | <b>000000000000400</b><br><b>Memory-- 00003800</b>    |
| (p) “1” Enter Break Address                          | (q) “DAT” Set Break Point                                | (r) Try to access 0x00003800                          |
| <b>000000000000400</b><br><b>NMI /BRK: Goto Mon</b>  | <b>SR : 00000100</b><br><b>PC : 0000136C</b>             | <b>R0 : 0000069C</b><br><b>R1 : 0000136C</b>          |
| (s) “ADR” Access to 0x00003800                       | (t) “UTL”-”2” Display Registers                          | (u) Continue to Hit any key                           |

**Figure7.5 Example Session of the Monitor Program**

## (B) Program Loader

- (i) Program Loading from PC
- (j) Now Loading
- (k) Finish Loading

You can download S-Format(S3) object file (ASCII Text file) from PC via RS-232C line. In default, 1200bps, 8bit non-parity. You can change the baud rate by changing monitor program source. (Or directly change UARTBRG0 and UARTBRG1 register by the monitor function.)

The acceptable S-Format records are only S0 (comment), S3 (actual object), S7 (end of record). If you use “asm” script for assembler, or “Makefile” for C program in Aquarius deliverables, they make suitable S-Format object file (\*.srec) for this monitor. After preparing object file on your PC, push “GET” key, then the FPGA system waits for sending data. Send object by ASCII file from any proper terminal application in your PC. During transfer, LCD shows top address of every record and the result of check sum test. If the monitor finds checksum error, the transfer will stop. When the monitor receives S7 record, it stops program loading, and shows the address of first record which have been received.

For convenience of explanation, please suppose you downloaded “shc\_lcdtest/main.srec”, which is LCD test program.

## (C) Run

- (l) Set Target Address
- (m) Go to program
- (n) Now, running program

The top address of LCD test program is 0x00003000. But the top address has vector table. The actual start address is 0x00003008, so, set address to this. Then push “RUN” key, the program will start. This “RUN” function is implemented by JSR instruction. So, if your program ends by RST instruction, the control will return to the monitor. Of course, if you want to stop your program, push “RES” key for reset, any time.

## (D) Debug Utility

- (o) Select Break Function

- (p) Enter Break Address
- (q) Set Break Point
- (r) Try a Break
- (s) Break happens

If you want to set break point, push “UTL” key and “2”. And enter the break address you want. To confirm the break operation, push “DAT”. Suppose you set 0x00003800 as break point. Now let’s access 0x00003800 by the monitor. The monitor reports BRK has happened. If you push any key, the control will return to monitor.

The break happens only when the WISHBONE address is just equal to the address you set as break point.

Once break happens, the break setting is cleared (the break address that is set in register “BRKADR” is kept but “BRKON” bit in “INTCON” register is cleared).

- (t) Select Register Reading
- (u) Check all Registers in CPU

You can examine register contents just when the break happens. By “UTL”-“1” key, you can see all CPU registers.

(2) **LCD Test:** directory “shc\_lcdtest”

Display all characters on LCD display. It is a very simple program. The source code is “main.c”. This program is located from address 0x00003000 (here is reset vector). To start it, jump to 0x00003008 by monitor program.

(3) **Interrupt! Clock:** directory “shc\_clock”

This is a digital clock. The time base is interrupt (IRQ) from interval timer. The timer requests IRQ in every  $50 \times 2^{12}$  [ns] @20MHz operation. The IRQ service routine controls internal software counter, and displays time. The source code is “main.c”. This program is located from address 0x00002000 (here is vector table). To start it, jump to 0x00002400 by monitor program.

You can adjust the clock. Push “DAT” key, enter hour. Again push “DAT”, enter minute. Again “DAT”, enter second. Finally push “DAT”, then, clock starts.

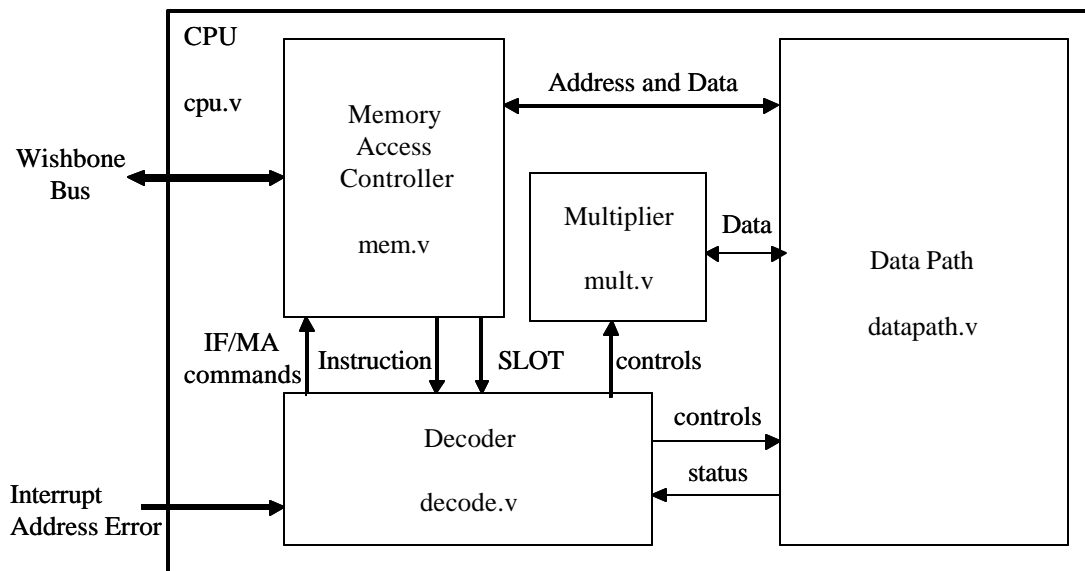
## Part2. Inside Aquarius CPU

## 8. Aquarius CPU Overview

This chapter shows overview of CPU, again.

### 8.1. Aquarius Block Diagram

Figure 8.1 shows the block diagram of Aquarius CPU core.



**Figure 8.1. Block Diagram of Aquarius**

Top layer of Aquarius is “CPU” which has WISHBONE compliant bus signals and accepts interruption related signals. The most important system signals such as clock and reset are not shown in this figure.

The Memory Access Controller handles instruction fetch and data read/write access. The operations of Memory Access Controller are fully controlled by Decoder unit. Memory Access Controller sends fetched instruction bit fields to the Decoder unit, and interchanges read/write data and its address with Data Path unit. Aquarius assumes the Wishbone bus is a Non-Harvard bus, then the simultaneous instruction fetch and R/W data access makes bus contention. Memory Access Controller handles such contention smoothly and informs the pipeline stall caused by the bus contention to Decoder unit. Also, the Memory Access



Controller can sense each boundary of bus cycles (with wait state) from Wishbone ACK signal. In Aquarius architecture (may be in SuperH-2 architecture as well), such bus cycle boundary corresponds to the pipeline's slot edge. So the Memory Access Controller produces the most important pipeline control signal "SLOT" indicating pipeline slot edge.

The Data Path unit has registers you can see in programmer's model in SuperH-2 manual such as General Registers (R0 to R15), Status Register (SR), Global Base Register (GBR), Vector Base Register (VBR), Procedure Register (PR) and Program Counter (PC). The Multiplication and Accumulate Registers (MACH/MACL) are found in Multiplication unit. The Data Path unit also has necessity operation resources such as ALU (Arithmetic and Logical operation Unit), Shifter, Divider, Comparator, temporary registers, many selectors, interfaces to/from Memory Access Controller and Multiply unit, and several buses to connect each resource. The Data Path is fully controlled by control signals from Decoder unit.

Multiply unit has a 32bit x 16bit multiplier and its control circuits. A 16bit x 16bit multiply operation is executed in one clock cycle. A 32 bit x 32bit multiply operation is done in two clock cycles. Multiply unit also has the Multiplier and Accumulate Registers (MACH/MACL). The MACH/MACL are not only the final result registers of multiply or multiply-and-accumulation but also the temporary registers to hold the 48bit partial multiply result from 32bit x 16bit multiplier for 32bit x 32bit operation. The multiply instruction, for example MULS.L, clears the contents of MACH/MACL in early stage of the instruction operation. However the multiply and accumulate instruction, for example MAC.L, does not clear MACH/MACL before the operation. The MAC.L accumulates its own partial multiply result to initial MACH/MACL and then finalize the operation result. The major difference between multiply (MULS.L) and "multiply and accumulate" (MAC.L) is whether to clear or not to clear the MACH/MACL before the operation. And also, for MAC.L and MAC.W instruction, the accumulation adder in this unit has saturating function.

The Decoder unit is the fundamental CPU controller. It orders Memory Access Controller fetch instructions and then receives the instruction. The Decoder Unit decodes the instruction bit fields and judges the followed operations. Basically, the Decoder unit plays the role only for the instruction ID stage. But it throws many control signals for following

EX, MA and WB stages toward Data Path unit, Multiplication unit, and Memory Access Controller. These control signals are kept and shifted with its pipeline flow at each slot edge until reaching to the target stage of the instruction. The Decoder unit detects every conditions of pipeline stalling, and makes each unit of CPU be controlled properly. Also, it controls not only simple 1 cycle instructions but also multi cycle instructions and exception's sequences such as interrupt and address error.

## 8.2. Aquarius CPU IN/OUT Signals

The Aquarius CPU (“cpu.v”)’s IN/OUT signals are shown in Table8.1.

| <b>Class</b>       | <b>Signal Name</b> | <b>Direction</b> | <b>Meaning</b>     | <b>Notes</b> |
|--------------------|--------------------|------------------|--------------------|--------------|
| <b>System</b>      | CLK                | Input            | System clock       |              |
| <b>Signals</b>     | RST                | Input            | Power On Reset     |              |
| <b>Wishbone</b>    | CYC_O              | Output           | Cycle Output       |              |
| <b>Bus</b>         | STB_O              | Output           | Strobe Output      |              |
| <b>Signals</b>     | ACK_I              | Input            | Device Acknowledge |              |
|                    | ADR_O[31:0]        | Output           | Address Output     |              |
|                    | DAT_I[31:0]        | Input            | Read Data          |              |
|                    | DAT_O[31:0]        | Output           | Write Data         |              |
|                    | WE_O               | Output           | Write Enable       |              |
|                    | SEL_O[3:0]         | Output           | Byte Lane Select   |              |
|                    | TAG0_I (IF_WIDTH)  | Input            | Fetch Width        |              |
| <b>Hardware</b>    | EVENT_REQ_I[2:0]   | Input            | Event Request      |              |
| <b>Event</b>       | EVENT_INFO_I[11:0] | Input            | Event Information  |              |
| <b>(interrupt)</b> | EVENT_ACK_O        | Output           | Event Acknowledge  |              |
| <b>SLEEP</b>       | SLP                | Output           | Sleep Pulse        |              |

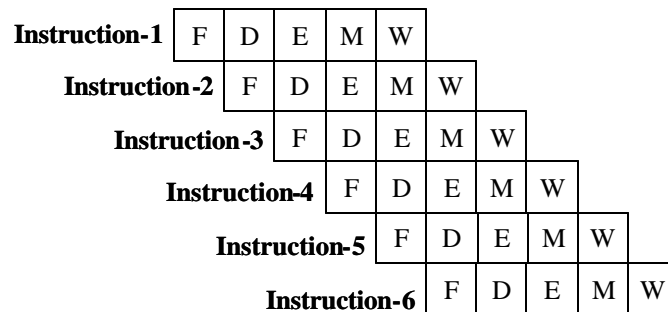
**Table8.1 Aquarius CPU IN/OUT Signals**

## 9. Overview of Pipeline Control

This chapter describes the basis of pipeline controls in Aquarius CPU.

### 9.1. Pipeline and Stage

The CPU executes its instructions with pipelined controls as shown in Figure9.1.



**Figure9.1 CPU Pipeline**

The pipeline has following 5 stages, basically.

#### (1) IF : Instruction Fetch (“F”)

It fetches instruction code from memory. If the bus width is 32bit and the lower 2bit of accessing address is 2'b00, the IF stage can fetch 2 instructions, because the length of each instructions is fixed in 16bit. If the bus width is 16bit or the lower 2bit of accessing address is not 2'b00, the IF stage can fetch only 1 instruction.

#### (2) ID : Decode (“D”)

It decodes fetched instruction code and controls whole CPU operation. The ID is the most important stage because all operations in each block of CPU are fully controlled by ID. The ID stage asserts many control signals to EX (`datapath.v`), MA (`mem.v`), and WB (`datapath.v`). Of course if the instruction code is multiplication related one, the ID activates multiplication unit (`mult.v`). Each control signal is shifted along with pipeline and activates each stage.

The ID also issues coming IF stage, and the IF forwards new instruction to ID stage. Then the CPU operation can continue.

If the hardware event signal is asserted, the ID samples it and switches its operation from fetched instruction's to the sequence of the hardware event exception.

**(3) EX : Execute**

According to controls from ID stage, the EX stage executes register-register operation, or address calculation for next MA stage. It can also issue multiplication related commands to multiplier unit (`mult.v`).

**(4) MA : Memory Access**

According to controls from ID stage, the MA stage reads/writes data from/to memory. The Aquarius CPU has non-Harvard bus, so the simultaneous IF and MA raise the bus contention. In this case, MA has the higher priority, so the IF is stalled by the MA.

**(5) WB : Write Back**

According to controls from ID stage, the WB writes back the memory read data to the register Rn. The WB is located at the pipeline tail of memory load instruction.

## 9.2. Pipeline of each Instruction

All instructions do not always have 5 stages. Figure9.2 shows some pipeline examples of typical instructions.

**(1) ALU Operation**

The instruction of register-register operation has only 3 stages; IF, ID and EX. The register-register operation is executed in EX stage, including register read, ALU operation, and register write.

**(2) Memory Store**

The instruction of store to memory has 4 stages; IF, ID, EX and MA. The memory access address is calculated in EX stage, and the write data is also prepared in EX stage.

**(3) Memory Load**

The instruction of load from memory has 5 stages; IF, ID, EX, MA and WB. The memory access address is calculated in EX stage. The load data is stored to register in WB stage. If the register to be written back is *NOT* same as the register which is used in following instruction, there is no contention, so the pipeline flows without stall.

The EX stage in the later instruction, which uses the written back data in the WB, can be executed at same timing as the WB by the grace of the forwarding apparatus.

#### **(4) Memory Load with Register Contention**

If the register to be written back is *SAME* as the register which is used in following instruction, the register contention happens. The ID stage of following instruction is stalled.

#### **(5) Branch Operation**

The branch instruction has multiple cycles. In the red square of Figure9.2 (5), you can find 3 pipelines. This means the BT (taken) instruction executes in 3 cycles. Generally, the multiple cycle instructions consist of multiple pipelines. In case of BT, the 1<sup>st</sup> pipeline calculates the address of branch target, the 2<sup>nd</sup> pipeline issues instruction fetch of branch target and increments PC, and the 3<sup>rd</sup> pipeline issues fetch of the next instruction of branch target and increments PC. The details of PC control are described later.

The previous instruction of BT has issued a instruction fetch, but the fetched code will be overwritten by the IF (of branch target) issued by the 2<sup>nd</sup> pipeline of the BT before sending to ID stage of target instruction. This extra instruction fetch is called “overrun fetch”. The codes fetched by overrun fetch are ignored.

#### **(6) Delayed Branch**

The delayed branch has 2 pipelines. The 1<sup>st</sup> pipeline calculates the address of branch target, the 2<sup>nd</sup> pipeline issues instruction fetch of branch target and increments PC.

The IF of instruction in delay slot, which has been issued by the previous instruction of the delayed branch, does not disappear (is not overwritten), then the instruction in delay slot is executed correctly before the branch target instruction.

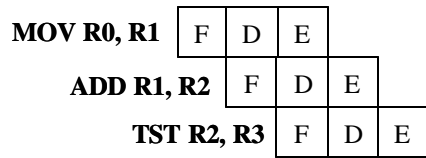
The branch instruction which consists of 2 pipelines becomes delayed branch with delayed slot, and the branch which has 3 pipelines becomes normal branch.

#### **(7) Multiplication**

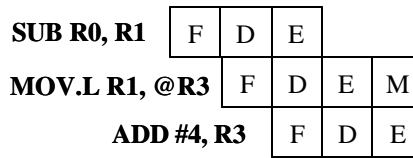
The multiplication related instructions have multiplier stage (“m”) on the pipeline tail. If the result register MACH/MACL does not conflict with followed instruction, there is no pipeline stall. The details of pipeline of multiplication are described in later chapter.

#### **(8) Multiplication with Register Contention**

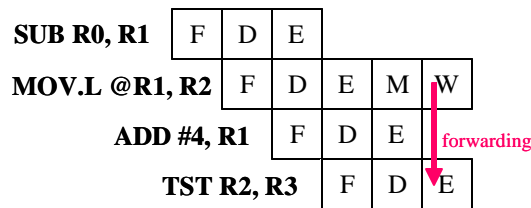
If the result registers MACH/MACL conflict with followed instruction, pipeline stall happens. The details of contention of multiplication related instructions are described in later chapter.



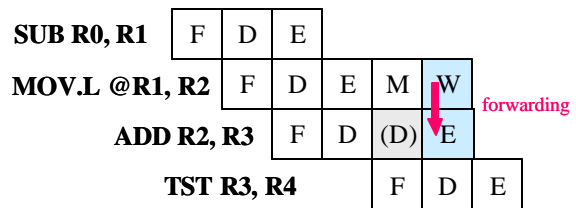
(1) ALU Operation



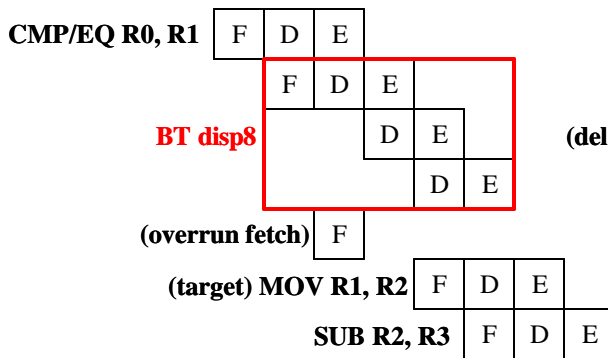
(2) Memory Store



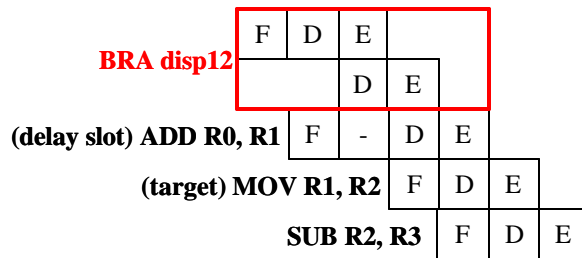
(3) Memory Load (w/o stall)



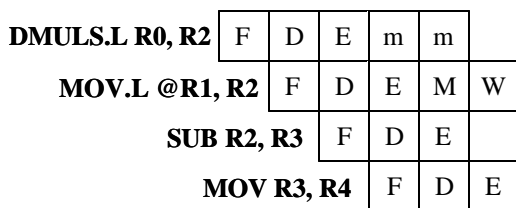
(4) Register Contention by Memory Load (w/ stall)



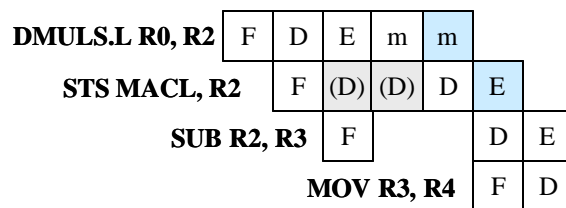
(5) Branch Operation



(6) Delayed Branch Operation



(7) Multiplication



(8) Multiplication (w/ stall)

Figure9.2 Pipeline of each Instruction

### 9.3. Register Conflict

As described previous section, the memory load instruction may cause register contention. See Figure9.2 (3) and (4).

### 9.4. Memory Access Conflict

The Aquarius CPU has non-Harvard bus, so the simultaneous IF and MA raise the bus contention, as shown in Figure9.3.

If the bus width is 32bit and the lower 2bit of accessing address is 2'b00, the IF stage can fetch 2 instructions, and the following IF stage does not need to produce actual memory read cycle. The IF stage, which issues actual bus cycle, is shown as “F”, and the IF stage, which does not issue real bus cycle and take the instruction from internal buffer, is shown as “f” in the figure. The simultaneous “M” and “F” cause the contention and the pipeline is stalled, but “M” and “f” does not conflict.

Note that, if you locate load/store instruction at long word boundary (address=4n), the MA stage of the instruction does not conflict with IF of post instruction (Figure9.3 (1)), otherwise (address=4n+2), it conflicts (Figure9.3 (2)).

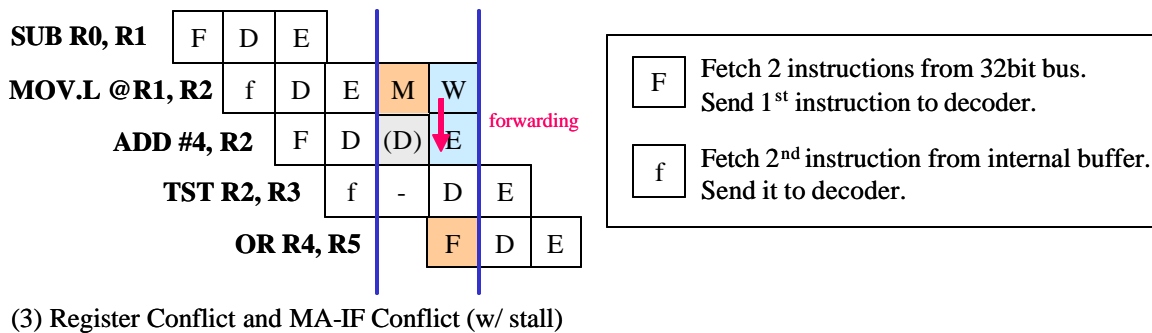
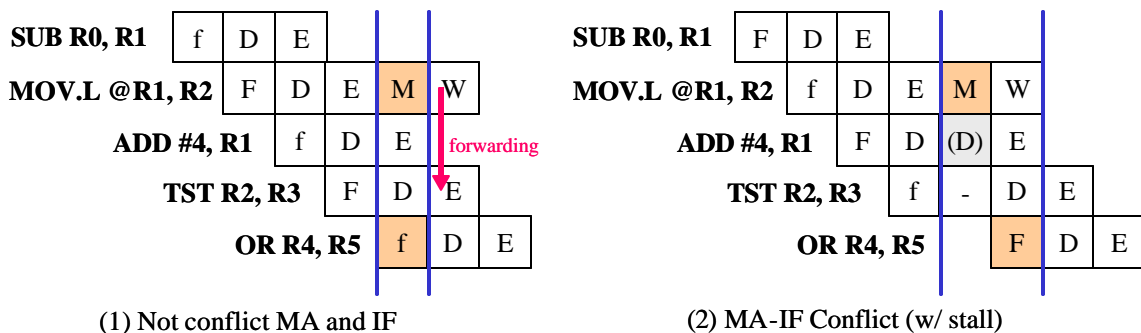


Figure9.3 Conflict between MA and IF

### 9.5. Who issues IF? Who issues ID?

The ID stage fundamentally controls whole CPU operations. No one issues ID stage. ID stage continues by itself.

The ID stage issues not only EX, MA and WB stages, but also the IF stage of followed instruction as shown in Figure9.4. After the power on reset, at the last sequence of power on reset exception, the IF stages of 1<sup>st</sup> instruction and 2<sup>nd</sup> instruction are issued by the last two decode stages in the exception sequence. Each IF stage of all followed instructions is issued by similar manner.

By the issued IF, the corresponding ID stage can get next instruction so that each ID can keep its continuance.

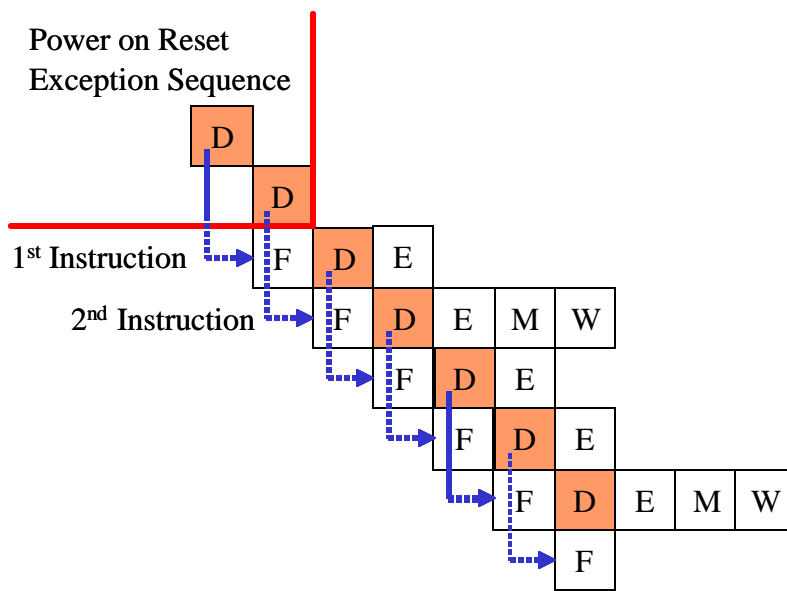


Figure9.4 IF Issue



## 10. Decoder Unit

This chapter describes the details of decoder unit (`decode.v`).

### 10.1. IN/OUT Signals

Table10.1 shows all in/out signals of decoder unit.

| Class                      | Direction      | Name           | From / To                         | Meaning   | Notes |
|----------------------------|----------------|----------------|-----------------------------------|---|-------|
| System Signals             | input          | CLK            | EXTERNAL                          | clock   |       |
|                            | input          | RST            | EXTERNAL                          | reset   |       |
| Pipeline Slot              | input          | SLOT           | mem.v                             | pipeline slot   |       |
| Instruction Fetch Controls | output         | IF_ISSUE       | mem.v                             | fetch request   |       |
|                            | output         | IF_JP          | mem.v                             | fetch caused by jump  |       |
|                            | input          | [15:0] IF_DR   | mem.v                             | fetch instruction   |       |
|                            | input          | IF_BUS         | mem.v                             | fetch access done to external bus                                     |       |
| Memory Access Controls     | input          | IF_STALL       | mem.v                             | fetch and memory access contention                                    |       |
|                            | output         | MA_ISSUE       | mem.v                             | memory access request   |       |
|                            | output         | KEEP_CYC       | mem.v                             | request read-modify-write (To be issued on READ-CYC to keep CYC 0 on) |       |
|                            | output         | MA_WR          | mem.v                             | memory access kind : Write(1)/Read(0)                                 |       |
| Multiply Controls (1)      | output         | [1:0] MA_SZ    | mem.v                             | memory access size : 00 byte, 01 word, 10 long, 11 inhibited          |       |
|                            | output         | MULCOM1        | mult.v                            | Mult M1 Latch Command   |       |
|                            | output         | [7:0] MULCOM2  | mult.v                            | Mult M2 Latch Command   |       |
|                            | output         | WRMACH, WRMACL | mult.v                            | Write MACH/MACL   |       |
| General Register Controls  | input          | MAC_BUSY       | mult.v                            | multiplier busy signal (negate at final operation state)              |       |
|                            | output         | RDREG_X        | datapath.v                        | read REG to X   |       |
|                            | output         | RDREG_Y        | datapath.v                        | read REG to Y   |       |
|                            | output         | WRREG_Z        | datapath.v                        | write REG from Z  |       |
|                            | output         | WRREG_W        | datapath.v                        | write REG from W  |       |
|                            | output         | [3:0] REGNUM_X | datapath.v                        | specify REG number reading to X                                       |       |
|                            | output         | [3:0] REGNUM_Y | datapath.v                        | specify REG number reading to Y                                       |       |
| output                     | [3:0] REGNUM_Z | datapath.v     | specify REG number writing from Z |   |       |
| output                     | [3:0] REGNUM_W | datapath.v     | specify REG number writing from W |   |       |
| ALU                        | output         | [4:0] ALUFUNC  | datapath.v                        | ALU function  |       |
| Memory Access Data         | output         | WRMAAD_Z       | datapath.v                        | write MAAD from Z   |       |
|                            | output         | WRMADW_X       | datapath.v                        | write MADW from X   |       |
|                            | output         | WRMADW_Y       | datapath.v                        | write MADW from Y   |       |
|                            | output         | RDMADR_W       | datapath.v                        | read MADR to W  |       |

**Table10.1 Decoder IN/OUT signals (1)**

| Class                               | Direction | Name             | From / To       | Meaning                               | Notes |
|-------------------------------------|-----------|------------------|-----------------|---------------------------------------|-------|
| Multiply Controls (2)               | output    | [1:0] MACSEL1    | datapath.v      | MAC Selector 1                        |       |
|                                     | output    | [1:0] MACSEL2    | datapath.v      | MAC Selector 2                        |       |
|                                     | output    | RDMACH_X         | datapath.v      | read MACH to X                        |       |
|                                     | output    | RDMACL_X         | datapath.v      | read MACL to X                        |       |
|                                     | output    | RDMACH_Y         | datapath.v      | read MACH to Y                        |       |
|                                     | output    | RDMACL_Y         | datapath.v      | read MACL to Y                        |       |
| SR Controls                         | output    | RDSR_X           | datapath.v      | read SR to X-bus                      |       |
|                                     | output    | RDSR_Y           | datapath.v      | read SR to Y-bus                      |       |
|                                     | output    | WRSR_Z           | datapath.v      | write SR from Z-bus                   |       |
|                                     | output    | WRSR_W           | datapath.v      | write SR from W-bus                   |       |
| Latch S bit                         | output    | MAC_S_LATCH      | datapath.v      | latch S bit before MAC operation      |       |
| GBR Controls                        | output    | RDGBR_X          | datapath.v      | read GBR to X-bus                     |       |
|                                     | output    | RDGBR_Y          | datapath.v      | read GBR to Y-bus                     |       |
|                                     | output    | WRGBR_Z          | datapath.v      | write GBR from Z-bus                  |       |
|                                     | output    | WRGBR_W          | datapath.v      | write GBR from W-bus                  |       |
| VBR Controls                        | output    | RDVBR_X          | datapath.v      | read VBR to X-bus                     |       |
|                                     | output    | RDVBR_Y          | datapath.v      | read VBR to Y-bus                     |       |
|                                     | output    | WRVBR_Z          | datapath.v      | write VBR from Z-bus                  |       |
|                                     | output    | WRVBR_W          | datapath.v      | write VBR from W-bus                  |       |
| PR Controls                         | output    | RDPR_X           | datapath.v      | read PR to X-bus                      |       |
|                                     | output    | RDPR_Y           | datapath.v      | read PR to Y-bus                      |       |
|                                     | output    | WRPR_Z           | datapath.v      | write PR from Z-bus                   |       |
|                                     | output    | WRPR_W           | datapath.v      | write PR from W-bus                   |       |
|                                     | output    | WRPR_PC          | datapath.v      | write PR from PC                      |       |
| PC Controls                         | output    | RDPC_X           | datapath.v      | read PC to X                          |       |
|                                     | output    | RDPC_Y           | datapath.v      | read PC to Y                          |       |
|                                     | output    | WRPC_Z           | datapath.v      | write PC from Z                       |       |
|                                     | output    | INCPC            | datapath.v      | increment PC                          |       |
|                                     | output    | IFADSEL          | datapath.v      | IF_AD selector                        |       |
| Immediate and Displacement Controls | output    | [15:0] CONST_IFD | datapath.v      | Constant Value from Instruction Field |       |
|                                     | output    | CONST_ZERO4      | datapath.v      | Const = unsigned lower 4bit           |       |
|                                     | output    | CONST_ZERO42     | datapath.v      | Const = unsigned lower 4bit * 2       |       |
|                                     | output    | CONST_ZERO44     | datapath.v      | Const = unsigned lower 4bit * 4       |       |
|                                     | output    | CONST_ZERO8      | datapath.v      | Const = unsigned lower 8bit           |       |
|                                     | output    | CONST_ZERO82     | datapath.v      | Const = unsigned lower 8bit * 2       |       |
|                                     | output    | CONST_ZERO84     | datapath.v      | Const = unsigned lower 8bit * 4       |       |
|                                     | output    | CONST_SIGN8      | datapath.v      | Const = signed lower 8bit             |       |
|                                     | output    | CONST_SIGN82     | datapath.v      | Const = signed lower 8bit * 2         |       |
|                                     | output    | CONST_SIGN122    | datapath.v      | Const = signed lower 12bit * 2        |       |
|                                     | output    | RDCONST_X        | datapath.v      | read CONST to X                       |       |
| output                              | RDCONST_Y | datapath.v       | read CONST to Y |                                       |       |
| Forwarding                          | output    | REG_FWD_X        | datapath.v      | forward REG from W to X               |       |
|                                     | output    | REG_FWD_Y        | datapath.v      | forward REG from W to Y               |       |

**Table10.1 Decoder IN/OUT signals (2)**

| Class                 | Direction | Name                  | From / To  | Meaning                                     | Notes |
|-----------------------|-----------|-----------------------|------------|---|-------|
| Compare Controls      | output    | [ 2 : 0 ] CMPCOM      | datapath.v | define comparator operation (command)       |       |
| Shifter Controls      | output    | [ 4 : 0 ] SFTFUNC     | datapath.v | Shifter Function                            |       |
|                       | output    | RDSFT_Z               | datapath.v | read SFTOUT to Z-BUS                        |       |
|                       | input     | T_BCC                 | datapath.v | T value for Bcc judgement                   |       |
|                       | output    | T_CMPSET              | datapath.v | reflect comparator result to T              |       |
|                       | output    | T_CRYSET              | datapath.v | reflect carry/borrow out to T               |       |
| T bit                 | output    | T_TSTSET              | datapath.v | reflect tst result to T                     |       |
| Q Bit                 | output    | T_SFTSET              | datapath.v | reflect shifted output to T                 |       |
| M bit                 | output    | QT_DV1SET             | datapath.v | reflect DIV1 result to Q and T              |       |
| Controls              | output    | MQT_DV0SET            | datapath.v | reflect DIV0S result to M, Q and T          |       |
|                       | output    | T_CLR                 | datapath.v | clear T                                     |       |
|                       | output    | T_SET                 | datapath.v | set T                                       |       |
|                       | output    | MQ_CLR                | datapath.v | clear M and Q                               |       |
| TEMP Register         | output    | RDTEMP_X              | datapath.v | read TEMP to X-bus                          |       |
|                       | output    | WRTEMP_Z              | datapath.v | write to TEMP from Z-bus                    |       |
| Controls              | output    | WRMAAD_TEMP           | datapath.v | output MAAD from TEMP                       |       |
|                       | input     | [ 2 : 0 ] EVENT_REQ   | EXTERNAL   | event request                               |       |
| Hardware Events       | output    | EVENT_ACK             | EXTERNAL   | event acknowledge                           |       |
|                       | input     | [ 11 : 0 ] EVENT_INFO | EXTERNAL   | event information (ILEVEL[3:0] VECTOR[7:0]) |       |
|                       | output    | RST_SR                | datapath.v | reset SR                                    |       |
| SR and I bit Controls | input     | [ 3 : 0 ] IBIT        | datapath.v | I bit in SR                                 |       |
|                       | output    | [ 3 : 0 ] ILEVEL      | datapath.v | IRQ Level                                   |       |
|                       | output    | WR_IBIT               | datapath.v | Write ILEVEL to I bit in SR                 |       |
| SLEEP                 | output    | SLP                   | EXTERNAL   | Sleep output                                |       |

**Table10.1 Decoder IN/OUT signals (3)**

## 10.2. Structure of Decoder Unit

Figure10.1 shows whole structure of decoder unit. The huge truth table generates all control signals for each block in CPU. The huge truth table (combinational circuit) receives 2 input signal groups. One is INSTR\_STATE[15:0] and the other is INSTR\_SEQ[3:0].

The INSTR\_STATE[15:0] shows the instruction code that should be processed in decoder unit. The IR register is reset by RST signal, so that the initial state of INSTR\_STATE[15:0] is set to `POWER\_ON\_RESET(16'hF700).

The INSTR\_STATE[15:0] is basically same as IF\_DR, which is fetched instruction code. But if interrupt or hardware exception event is detected, the INSTR\_STATE[15:0] is replaced to corresponding exception code according to EVENT\_REQ[2:0] and EVENT\_INFO[11:0] ,

then the signal IF\_DR\_EVT[15:0] is created. Here, the some necessary controls for masking interrupt or hardware exception are performed, that is,

(1) All exceptions are masked after delayed branch (i.e. the instruction in branch slot is never replaced to exception sequence) using DELAY\_SLOT signal which comes from the huge truth table..

(2) Some specific instructions such as LDC/LDC.L mask interrupt (i.e. an instruction just after the instruction which masks interrupt is never replaced to interrupt sequence) using MASKINT signal from huge truth table.

(3) If the priority level of IRQ is less than I bit in SR, the interrupt request should be ignored.

The IF\_DR is updated by memory controller regardless of instruction sequence because the decoder itself requests IF as its own operation. So, IF\_DR (IF\_DR\_EVT) should be latched to IR register, if the instruction needs multiple cycles (including memory waits and pipeline stalls).

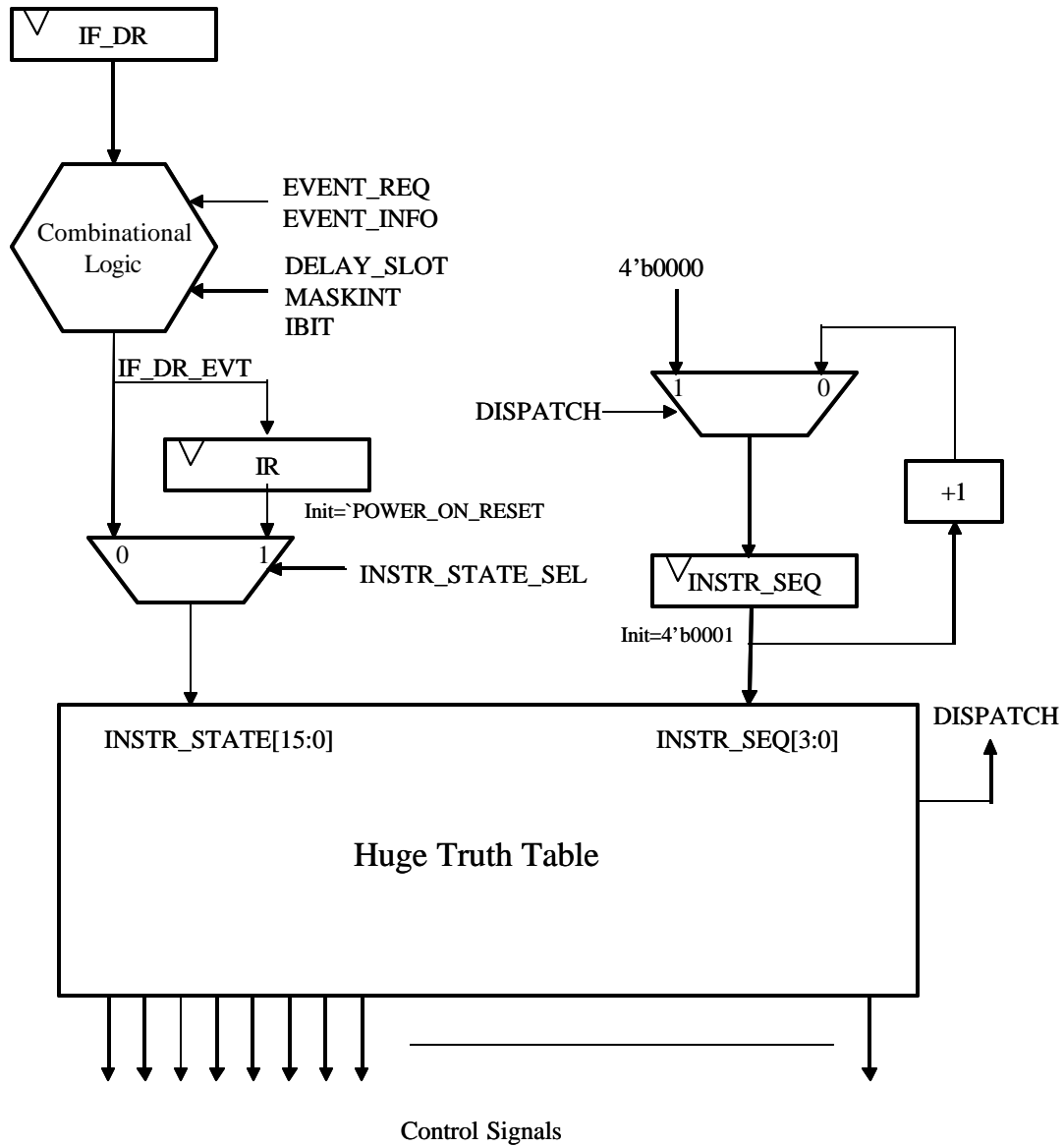
The INSTR\_SEQ[3:0] has its meanings only when the executing instruction has multiple cycles. Its default value is 4'b0000, and the multi-cycle instruction increments INSTR\_SEQ to make multiple pipelines as shown in, for example, Figure9.2 (BT, BRA).

The reset state of INSTR\_SEQ (when RST asserted) is set to 4'b0001 to begin power on reset sequence, because the value 4b0000 has specific meaning for the control of the decoder's state machine, as shown in later (Table10.2).

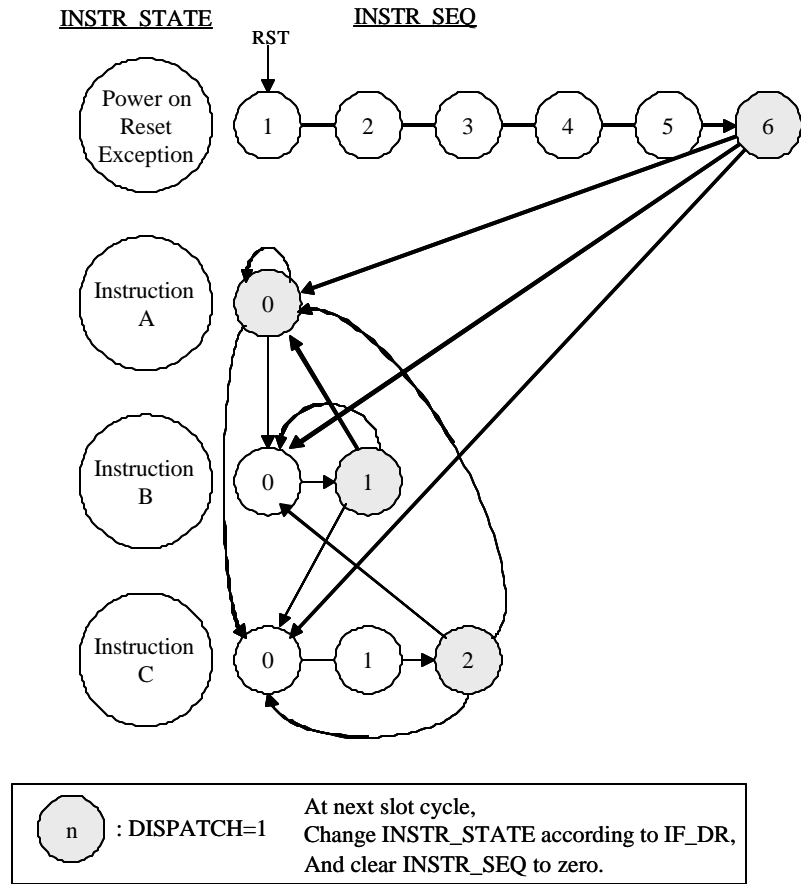
The combination of INSTR\_STATE and INSTR\_SEQ specify whole control signal states via the huge truth table. This combinational circuit also outputs a signal DISPATCH. Its assertion indicates that the pipeline stage of instruction is final. If the DISPATCH is asserted, INSTR\_STATE should be updated according to IF\_DR or proper exception code (IF\_DR\_EVT), and INSTR\_SEQ should be reset to zero.

The detail state controls are shown in Table10.2. The way of controls depends on status of pipeline stall. The signals NEXT\_ID\_STALL and ID\_STALL indicate the status of pipeline stall. The signal meanings are describes in later section.

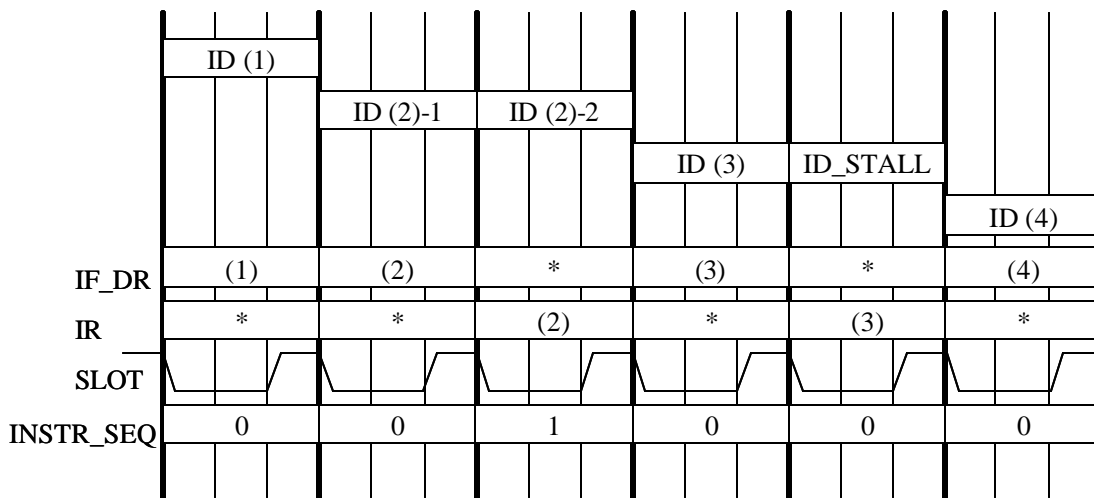
Figure10.2 shows a operation image of the decoder state machine. And Figure10.3 shows a basic example of ID stage operation.



**Figure10.1 State Machine in Decoder Unit**



**Figure10.2 State Transition**



**Figure10.3 Basic Operation of State Machine in Decoder Unit**

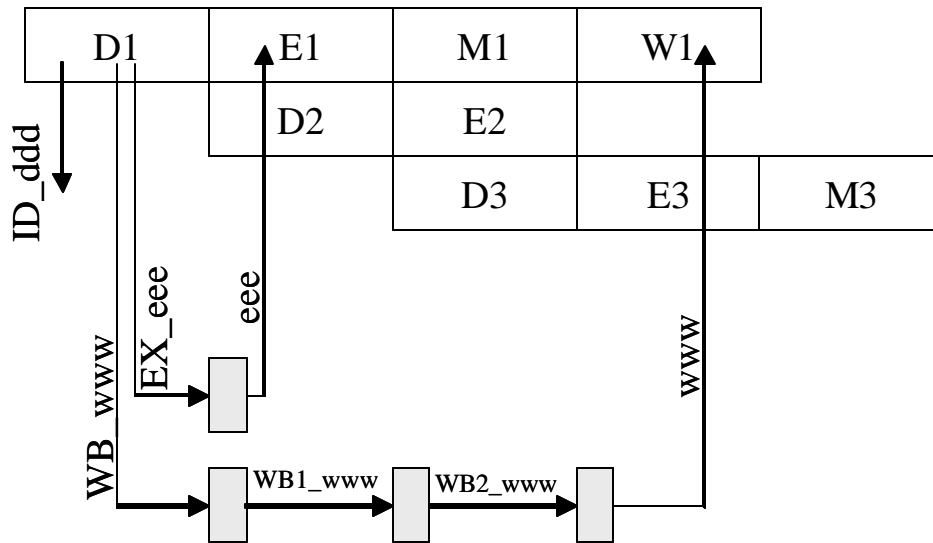
| Input |               |          |          |           | Output      | @Next Slot |           | Notes   |
|-------|---------------|----------|----------|-----------|-------------|------------|-----------|---|
| SLOT  | NEXT_ID_STALL | ID_STALL | DISPATCH | INSTR_SEQ | INSTR_STATE | INSTR_SEQ  | IR        |   |
| 0     | *             | *        | *        | *         | IR          | Keep       | Keep      | Not Changed   |
| 1     | 0             | 0        | 0        | >=0001    | IR          | +1         | Keep      | During Multi-Cycle Instruction                                  |
| 1     | 0             | 0        | 0        | =0000     | IF DR EVT   | +1         | IF DR EVT | First ID Stage of Multi-Cycle Instruction                       |
| 1     | 0             | 0        | 1        | >=0001    | IR          | Clear0     | Keep      | Final ID Stage of Multi-Cycle Instruction                       |
| 1     | 0             | 0        | 1        | =0000     | IF DR EVT   | Clear0     | IF DR EVT | ID Stage of Single Cycle Instruction                            |
| 1     | 0             | 1        | 0        | >=0001    | IR          | +1         | Keep      | Stalled Last Slot during Multi-Cycle Instruction                |
| 1     | 0             | 1        | 0        | =0000     | IR          | +1         | Keep      | Stalled Last Slot of first ID stage of Multi-Cycle Instruction  |
| 1     | 0             | 1        | 1        | >=0001    | IR          | Clear0     | Keep      | Stalled Last Slot of Final ID Stage of Multi-Cycle Instruction  |
| 1     | 0             | 1        | 1        | =0000     | IR          | Clear0     | Keep      | Stalled Last Slot of ID Stage of Single Cycle Instruction       |
| 1     | 1             | 0        | 0        | >=0001    | IR          | Keep       | Keep      | Stalled First Slot during Multi-Cycle Instruction               |
| 1     | 1             | 0        | 0        | =0000     | IF DR EVT   | Keep       | IF DR EVT | Stalled First Slot of first ID stage of Multi-Cycle Instruction |
| 1     | 1             | 0        | 1        | >=0001    | IR          | Keep       | Keep      | Stalled First Slot of Final ID Stage of Multi-Cycle Instruction |
| 1     | 1             | 0        | 1        | =0000     | IF DR EVT   | Keep       | IF DR EVT | Stalled First Slot of ID Stage of Single Cycle Instruction      |
| 1     | 1             | 1        | 0        | >=0001    | IR          | Keep       | Keep      | Stalling Slot during Multi-Cycle Instruction                    |
| 1     | 1             | 1        | 0        | =0000     | IR          | Keep       | Keep      | Stalling Slot of first ID stage of Multi-Cycle Instruction      |
| 1     | 1             | 1        | 1        | >=0001    | IR          | Keep       | Keep      | Stalling Slot of Final ID Stage of Multi-Cycle Instruction      |
| 1     | 1             | 1        | 1        | =0000     | IR          | Keep       | Keep      | Stalling Slot of ID Stage of Single Cycle Instruction           |

**Table10.2 State Controls of Decoder Unit**

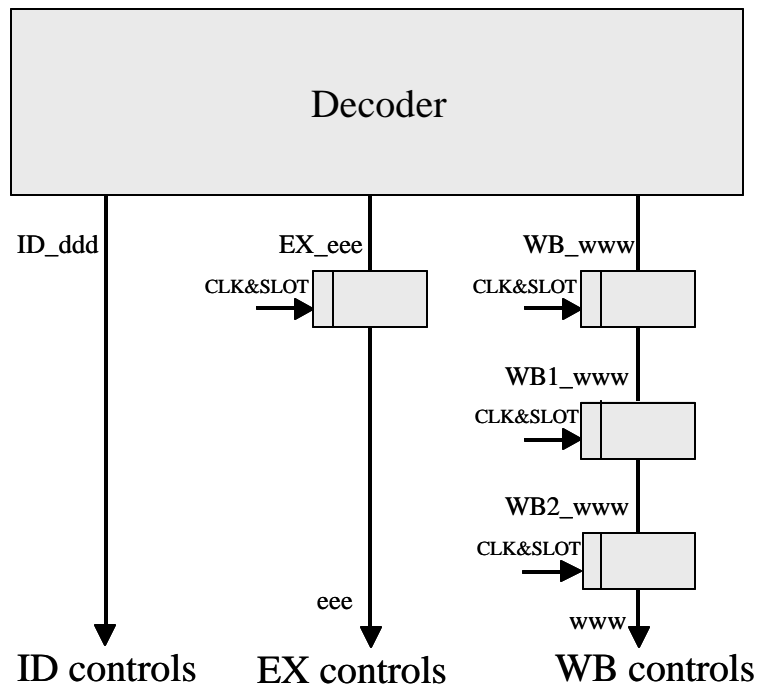
### 10.3. Shifting Control Signals

The decoder unit makes many control signals to control whole CPU blocks. These control signals are generated by the huge truth table as described above. The output timing of all control signals is always on ID stage. But these signals should control not only ID stage operations but also EX, MA and WB stage operations. So the signals which control EX or WB should be shifted as shown in Figure10.4. Actually, the flip-flops shown in Figure10.5 are used to shift each control signal.

Note that MA controls are performed in EX stage because how to issue the MA can be determined in EX stage, in which address of MA is calculated and write data is prepared.



**Figure 10.4** Shifting Control Signals



**Figure 10.5** Shifting Circuit

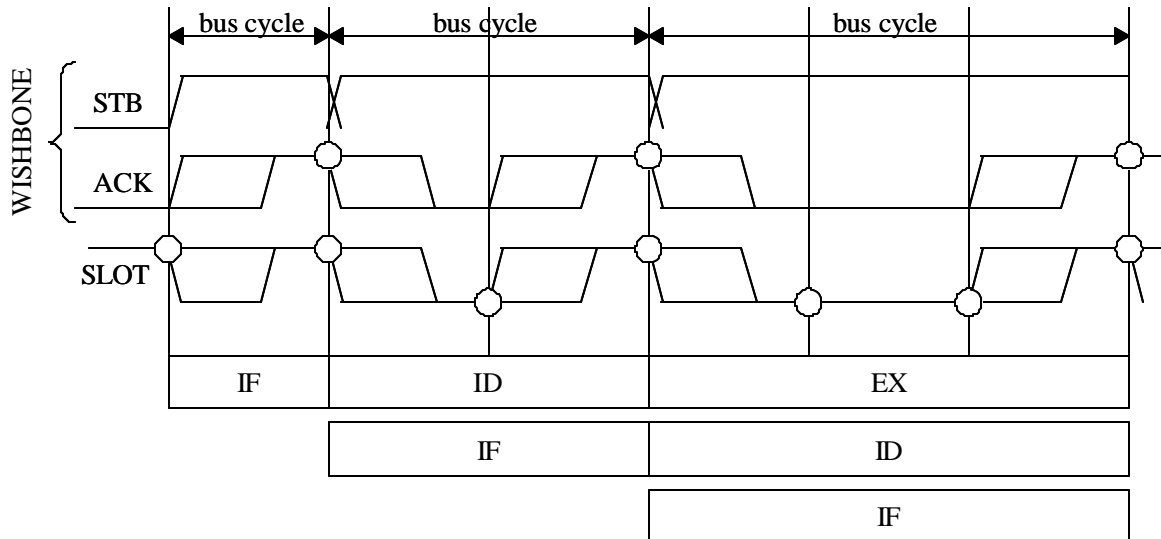


## 10.4. Pipeline Stall

The pipeline is stalled by following 4 reasons.

### [1] Wait States on Instruction Fetch (IF) or Data Access (MA)

All pipeline slots are synchronized to memory access. The signal SLOT from `mem.v` indicates the each slot edge. If there is no memory access or there is memory access without wait state, the pipeline slot do not stalls (SLOT=1). If there is memory access with wait state, the pipeline stalls (SLOT=0) until SLOT signal is asserted (it means the wait state finishes). So the clock inputs of whole flip-flops for controls are gated by SLOT signal. Only by this clock gating, such kind of pipeline stall is fully controlled. See Figure10.6.



**Figure10.6 Bus Wait State and Slot control**

### [2] Conflict IF and MA

As you know, the simultaneous IF and MA conflicts and make the pipeline stalls as shown in Figure9.3. The memory access controller (`mem.v`) detects IF-MA conflicts and informs the events to decoder unit using a signal `IF_STALL`. The circuit for conflict detection receives `IF_STALL`, then controls pipeline stalling.

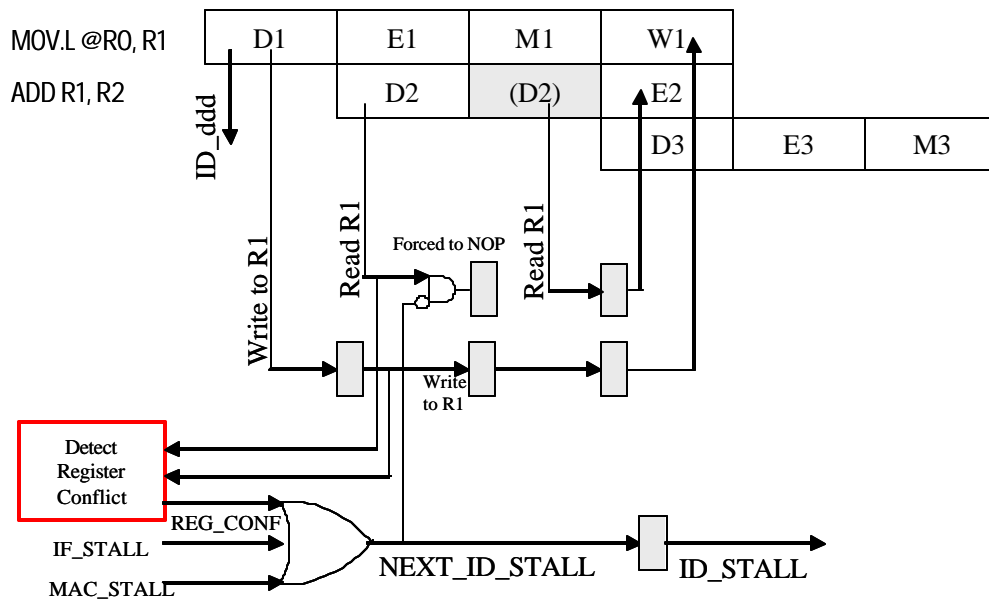
### [3] Multiplication Contention

As shown in, for example, Figure9.2 (8), the multiplication related instruction may

cause pipeline stall. This is controlled by using MAC\_BUSY signal from `mult.v` and signals WB\_MAC\_BUSY, EX\_MAC\_BUSY and MAC\_STALL\_SENSE from the huge truth table in decoder unit. By the methods described in later section, the multiplication related stall signal MAC\_STALL is generated. The circuit for conflict detection receives MAC\_STALL, then controls pipeline stalling.

#### [4] Register Contention

Memory load instruction may cause register contention with followed instruction which uses the write back data of the previous load instruction. The circuit for conflict detection watches pipeline control signals as shown in Figure 10.7 and make a conflict indicate signal REG\_CONF. In this case, MOV.L @R0,R1 and ADD R1, R2 cause R1 conflict. At the ID stage of ADD, the control signal to read R1 (EX\_RDREG\_X or EX\_RDREG\_Y) for ADD instruction and the *shifted* control signal to write back to R1 (WB1\_WRREG\_W) for MOV.L instruction are asserted simultaneously. This means there is R1 conflict, so the ID stage of ADD should be stalled.



**Figure 10.7 Detecting Register Conflict**

Regarding above [2] [3] [4], IF\_STALL, MAC\_STALL and REG\_CONF are ored and the NEXT\_ID\_STALL is created.

The NEXT\_ID\_STALL means that the ID stall continues by at least next slot.

Note that the ID stage with NEXT\_ID\_STALL=1 should force to NOP the control signals for each CPU block because the stage has no meanings regarding execution of the instruction.

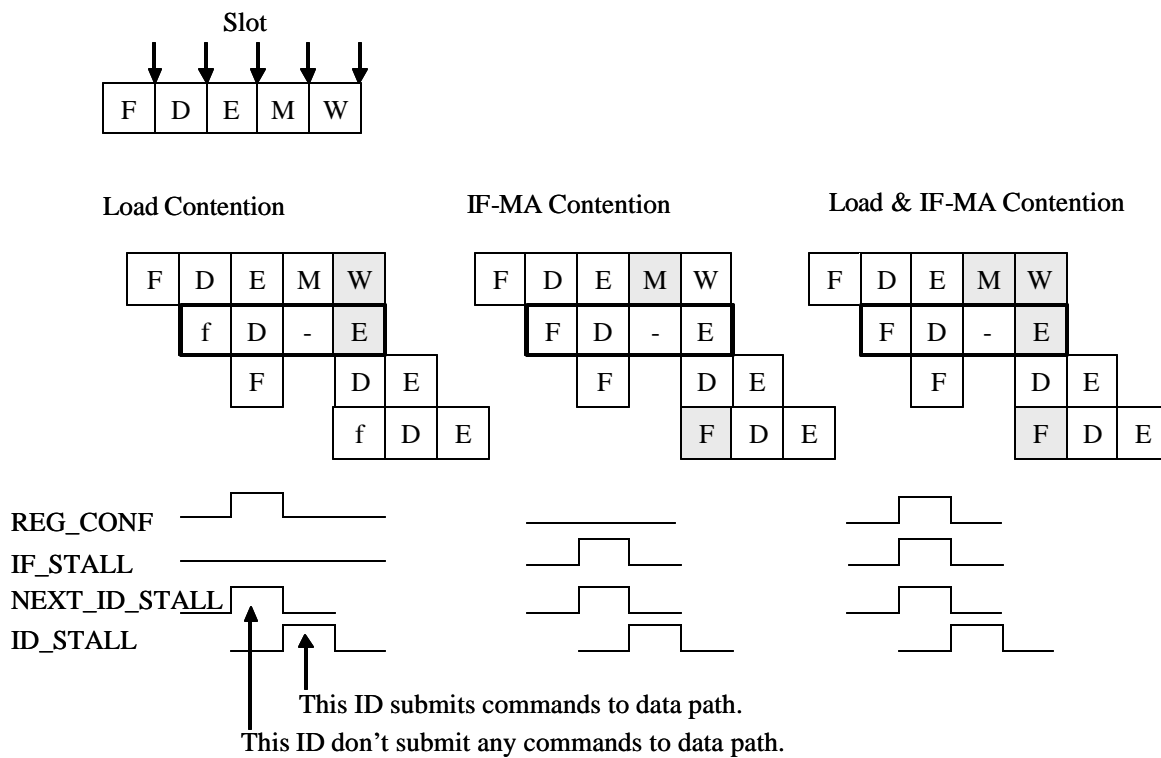
If NEXT\_ID\_STALL is asserted, the ID\_STALL should be asserted at the next slot.

See Figure10.8 regarding some examples of the stall control.

The meanings of combination of NEXT\_ID\_STALL and ID\_STALL are shown in Table10.3.

| NEXT_ID_STALL | ID_STALL | Meanings                                       | Control Signals |
|---------------|----------|--|-----------------|
| 0             | 0        | No Pipeline Stalls                             | Active          |
| 0             | 1        | ID is stalled. The stalled slot is final one.  | Active          |
| 1             | 0        | ID is stalled. The stalled slot is first one.  | Force to NOP    |
| 1             | 1        | ID is stalled. The stalled slot will continue. | Force to NOP    |

**Table10.3 Combination of NEXT\_ID\_STALL and ID\_STALL**



**Figure10.8 Controls of ID stall**

## 10.5. Register Forwarding

As shown in Figure9.2 and Figure9.3, register forwarding should be implemented not to reduce CPU cycle performance. After the memory load instruction, the register forwarding may be needed. This situation can be detected by watching some control signals. This is very similar to the detection of register conflict. If register content should be forwarded from Write Back Bus (W-BUS) to Register Read Bus X (X-BUS) in the data path unit, the signal REGFWD\_X is asserted. If register content should be forwarded from W-BUS to Register Read Bus Y (Y-BUS), the signal REGFWD\_Y is asserted. Actual forwarding transfer is performed in data path unit. Also see the chapter of data path unit.

Some examples are described in next section.

## 10.6. Examples of Pipeline Control

From Figure10.9 to Figure10.11 shows some examples of pipeline controls including stall control and register forwarding.

### (1) Memory Load Contention (Figure10.9)

The Slot4 detects register contention, then REG\_CONF is asserted. The ID at slot4 (ADD) is stalled. The Slot6 forwards write back data of MOV.L to EX stage of ADD.

### (2) Contention of IF and MA (Figure10.10)

The EX of MOV.L asserts MA\_ISSUE, and the IF of ADD asserts IF\_ISSUE. This means the IF-MA confliction. Then, the memory access controller returns IF\_STALL at slot4 and the ID in slot4 (ADD) is stalled.

### (3) Delayed Branch (Figure10.11)

The 1st EX of BRA (slot4) makes fetch address of SUB (target), and 2nd EX make fetch address of AND.

## 10.7. Control of Program Counter

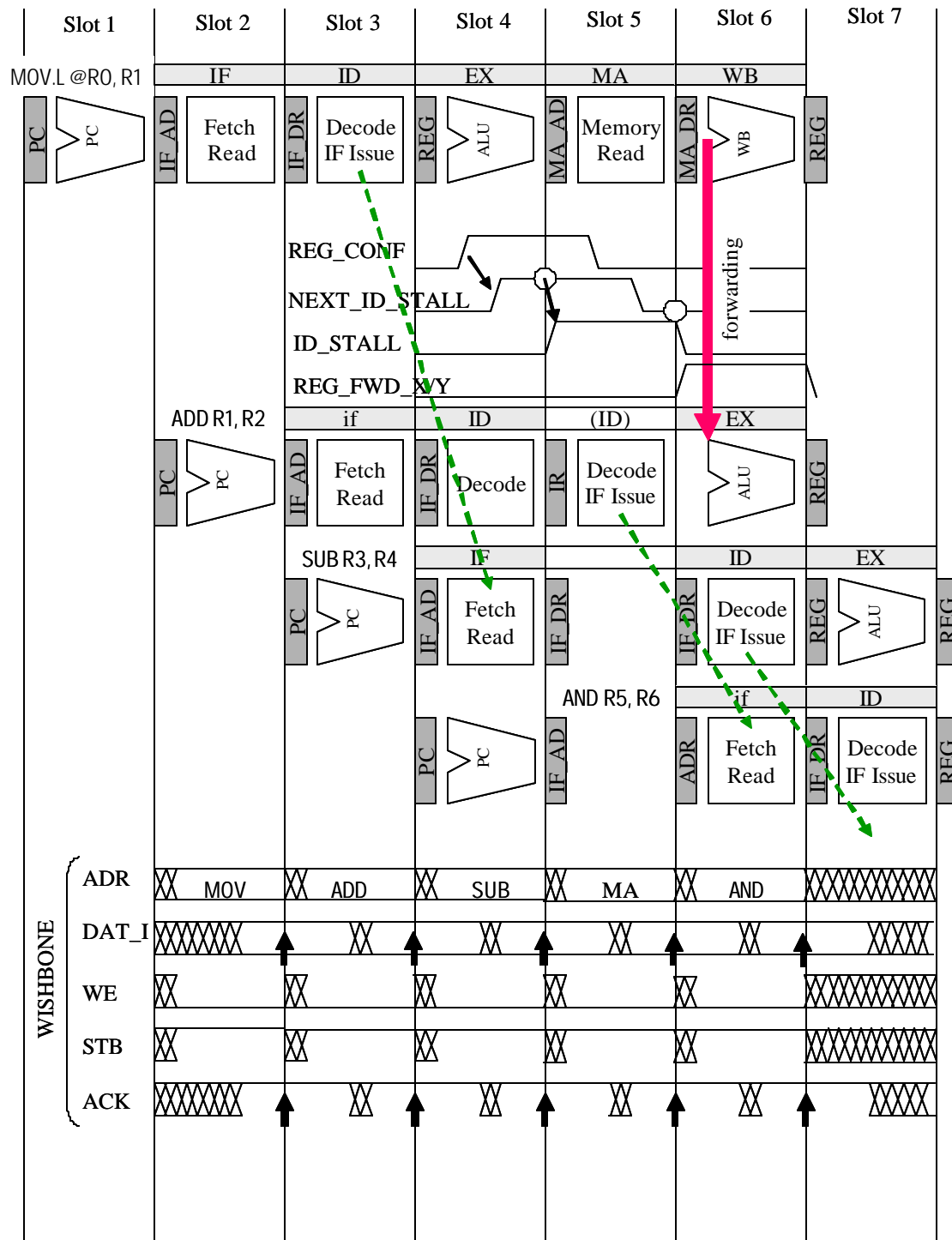
As described in last chapter, the ID stage issues IF stage. This is rigidly true. But the timing of changing PC has 2 cases.

[1] When program runs straight forward, the PC is incremented at ID stage.

[2] When program branches, the PC is changed at EX stage.

Figure10.12 shows good example of PC controls including exception sequence and branch operation.

## Memory Load Contention



**Figure 10.9 Memory Load Contention**

## Contention of IF and MA

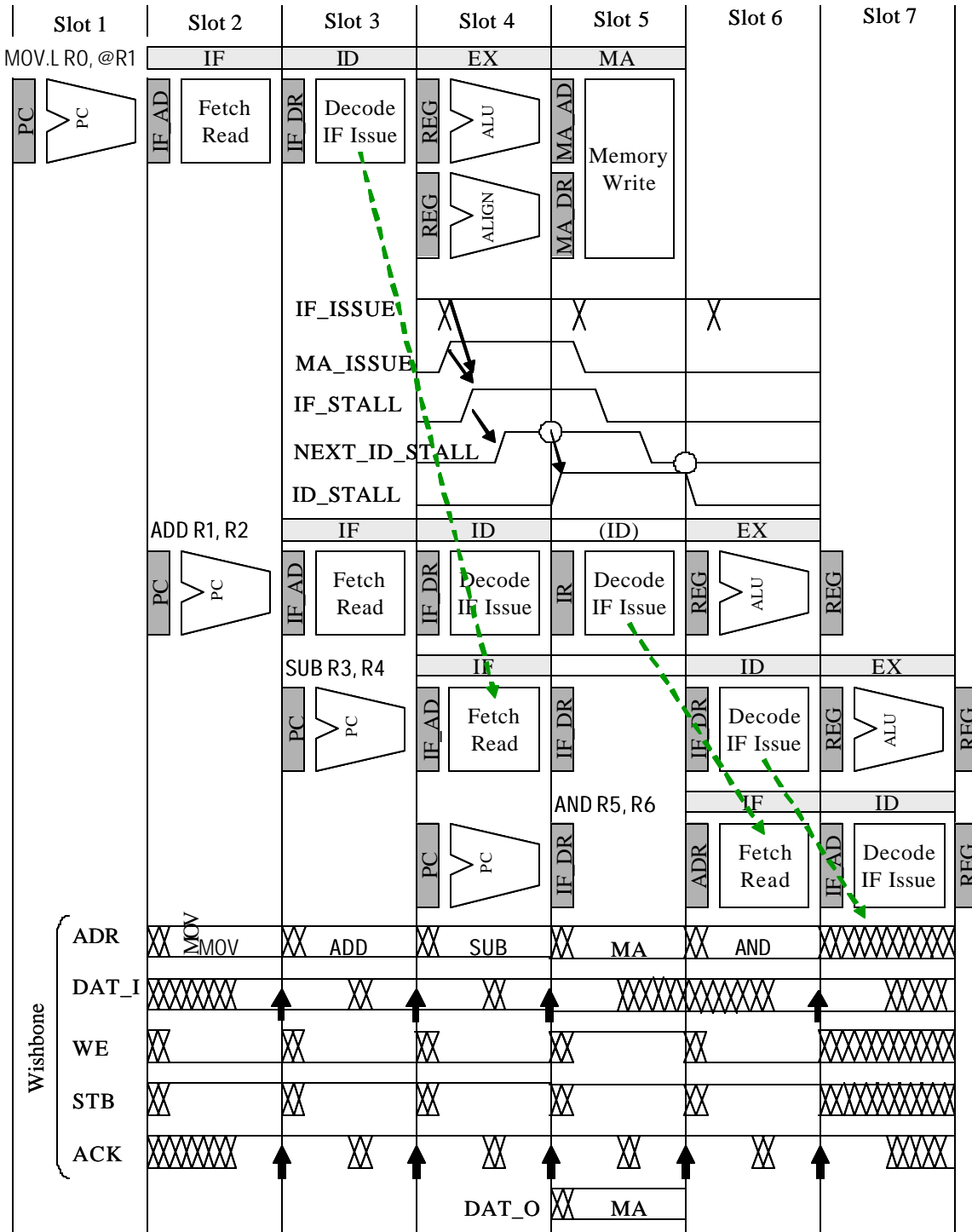
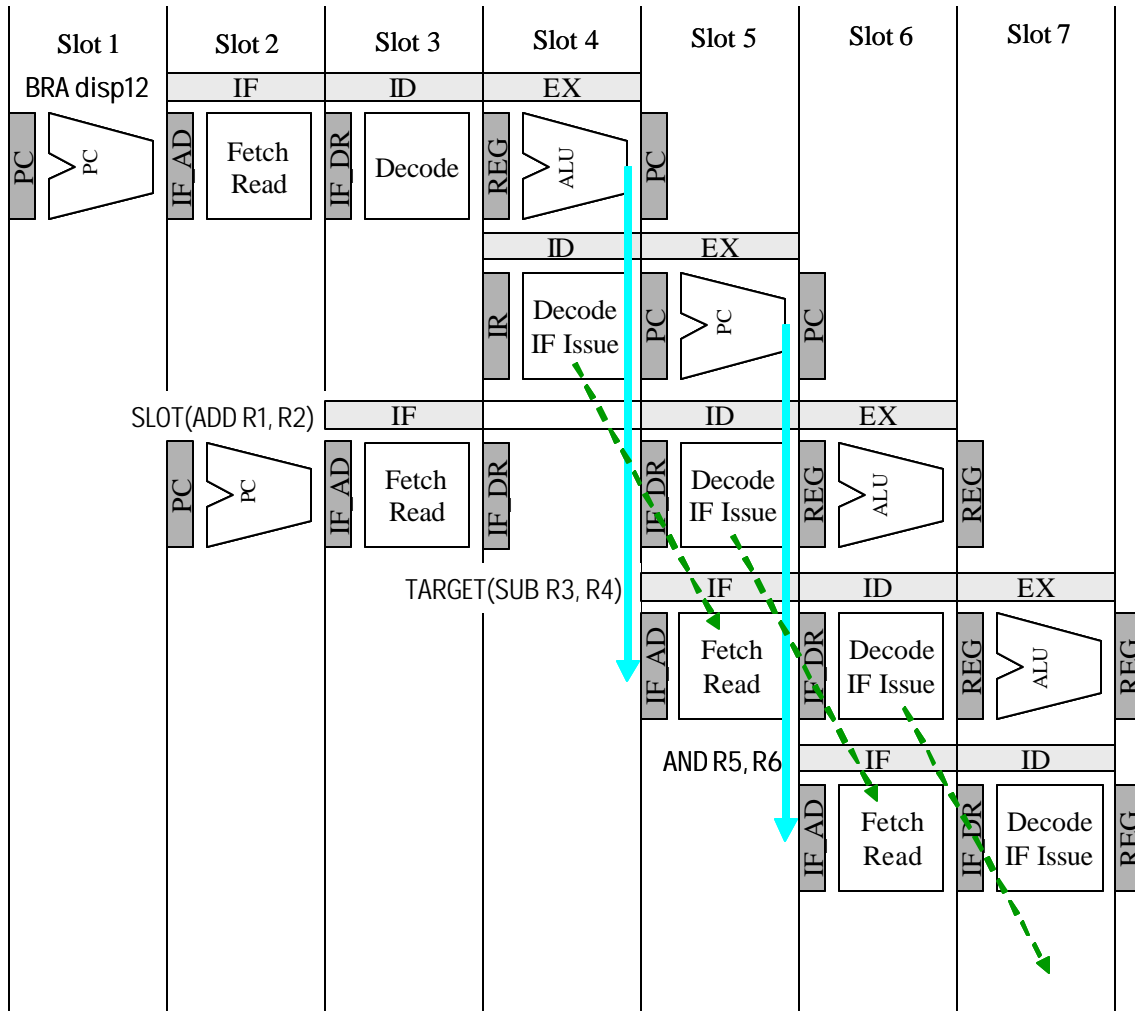
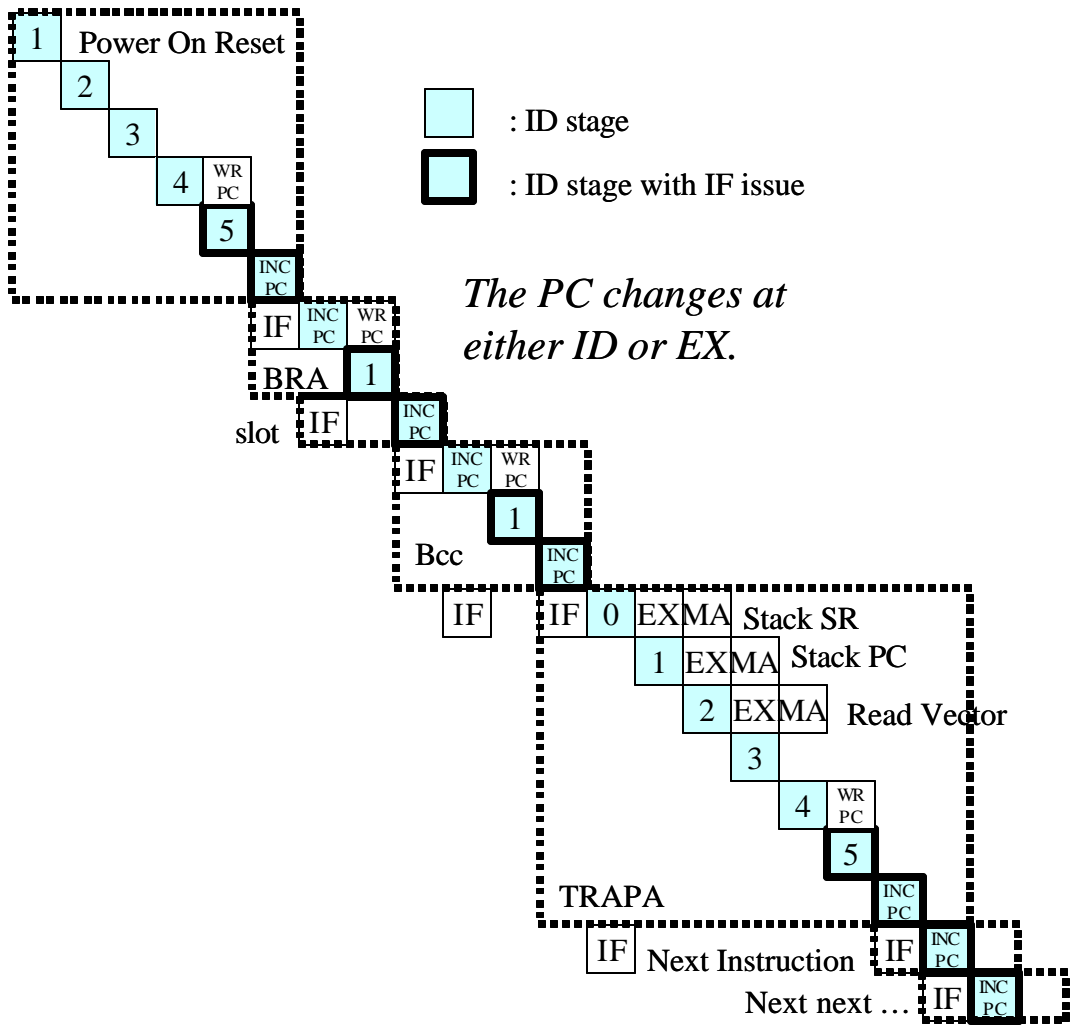


Figure 10.10 Contention of IF and MA

## Delayed Branch



**Figure 10.11 Control of Delayed Branch**



**Figure10.12 PC Controls**



# 11. Memory Access Control Unit

This chapter describes the details of memory access control unit (mem.v).

## 11.1. IN/OUT Signals

Table11.1 shows all in/out signals of memory access control unit.

(Although a signal IF\_BUS goes to decoder unit, it is not used.)

| Class                      | Direction | Name             | From / To  | Meaning  | Notes |
|----------------------------|-----------|------------------|------------|--|-------|
| System Signals             | input     | CLK              | EXTERNAL   | clock  |       |
|                            | input     | RST              | EXTERNAL   | reset  |       |
| WISHBONE Bus Signals       | output    | CYC              | EXTERNAL   | cycle output   |       |
|                            | output    | STB              | EXTERNAL   | strobe   |       |
|                            | input     | ACK              | EXTERNAL   | external memory ready  |       |
|                            | output    | [ 31 : 0 ] ADR   | EXTERNAL   | external address   |       |
|                            | input     | [ 31 : 0 ] DATI  | EXTERNAL   | external data read bus   |       |
|                            | output    | [ 31 : 0 ] DATO  | EXTERNAL   | external data write bus  |       |
|                            | output    | WE               | EXTERNAL   | external write/read  |       |
|                            | output    | [ 3 : 0 ] SEL    | EXTERNAL   | external valid data position                                     |       |
|                            | input     | IF_WIDTH         | EXTERNAL   | external fetch space width (IF_WIDTH)                            |       |
| SLOT                       | output    | SLOT             | ALL in CPU | pipeline slot edge   |       |
| Instruction Fetch Commands | input     | IF_ISSUE         | decode.v   | fetch request  |       |
|                            | input     | IF_JP            | decode.v   | fetch caused by jump   |       |
|                            | input     | [ 31 : 0 ] IF_AD | datapath.v | fetch address  |       |
|                            | output    | [ 15 : 0 ] IF_DR | datapath.v | fetch instruction  |       |
|                            | output    | IF_BUS           | decode.v   | fetch access done to extenal bus                                 |       |
|                            | output    | IF_STALL         | decode.v   | fetch and memory access contention                               |       |
| Memory Access Commands     | input     | MA_ISSUE         | decode.v   | memory access request  |       |
|                            | input     | KEEP_CYC         | decode.v   | request read-modify-write (Asserted on READ-CYC to keep CYC_O=1) |       |
|                            | input     | MA_WR            | decode.v   | memory access kind : Write(1)/Read(0)                            |       |
|                            | input     | [ 1 : 0 ] MA_SZ  | decode.v   | memory access size<br>00 byte, 01 word, 10 long, 11 inhibited    |       |
|                            | input     | [ 31 : 0 ] MA_AD | datapath.v | memory access address  |       |
|                            | input     | [ 31 : 0 ] MA_DW | datapath.v | memory write data  |       |
|                            | output    | [ 31 : 0 ] MA_DR | datapath.v | memory read data   |       |

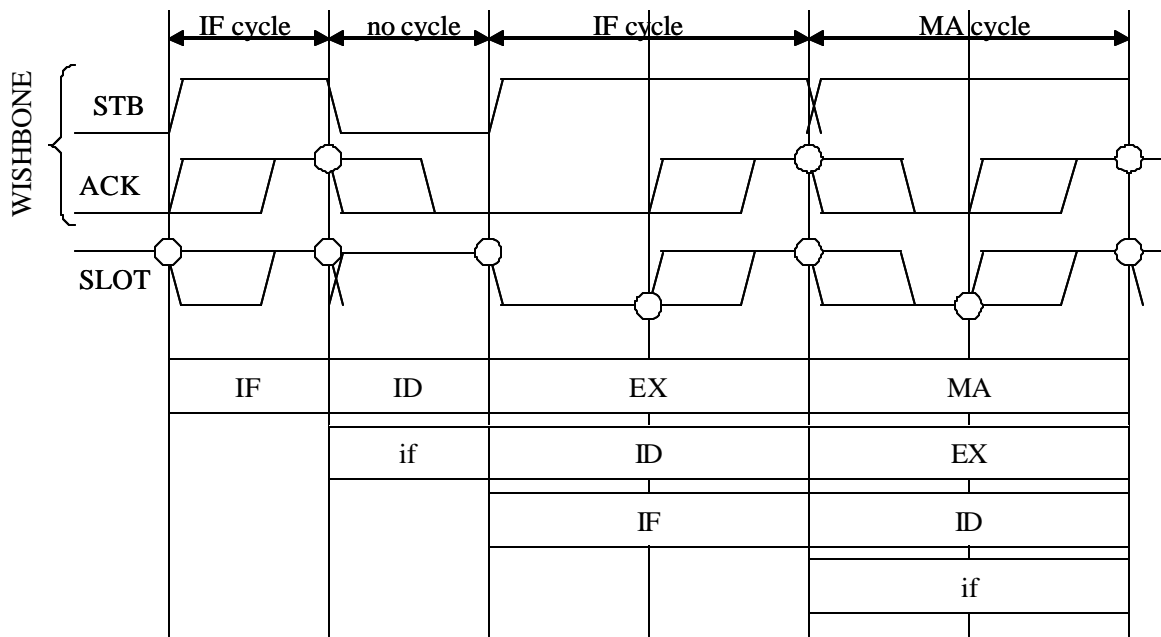
**Table11.1 Memory Access Control Unit IN/OUT Signals**

## 11.2. WISHBONE's ACK and Aquarius' SLOT

As described last chapter, the signal SLOT indicates the pipeline slot edges, and is created from WISHBONE's ACK signal in the memory access control unit. The clocks of each

flip-flop in Aquarius CPU are gated by SLOT signal, so that the pipeline stall derived from memory access cycle is easily controlled.

The waveform of SLOT is very similar to ACK, except that the SLOT is asserted if there is no memory access cycle, as shown in Figure 11.1's second slot. If external memory is accessed, the waveform of SLOT follows to ACK signal.



**Figure 11.1 WISHBONE's ACK and Aquarius' SLOT**

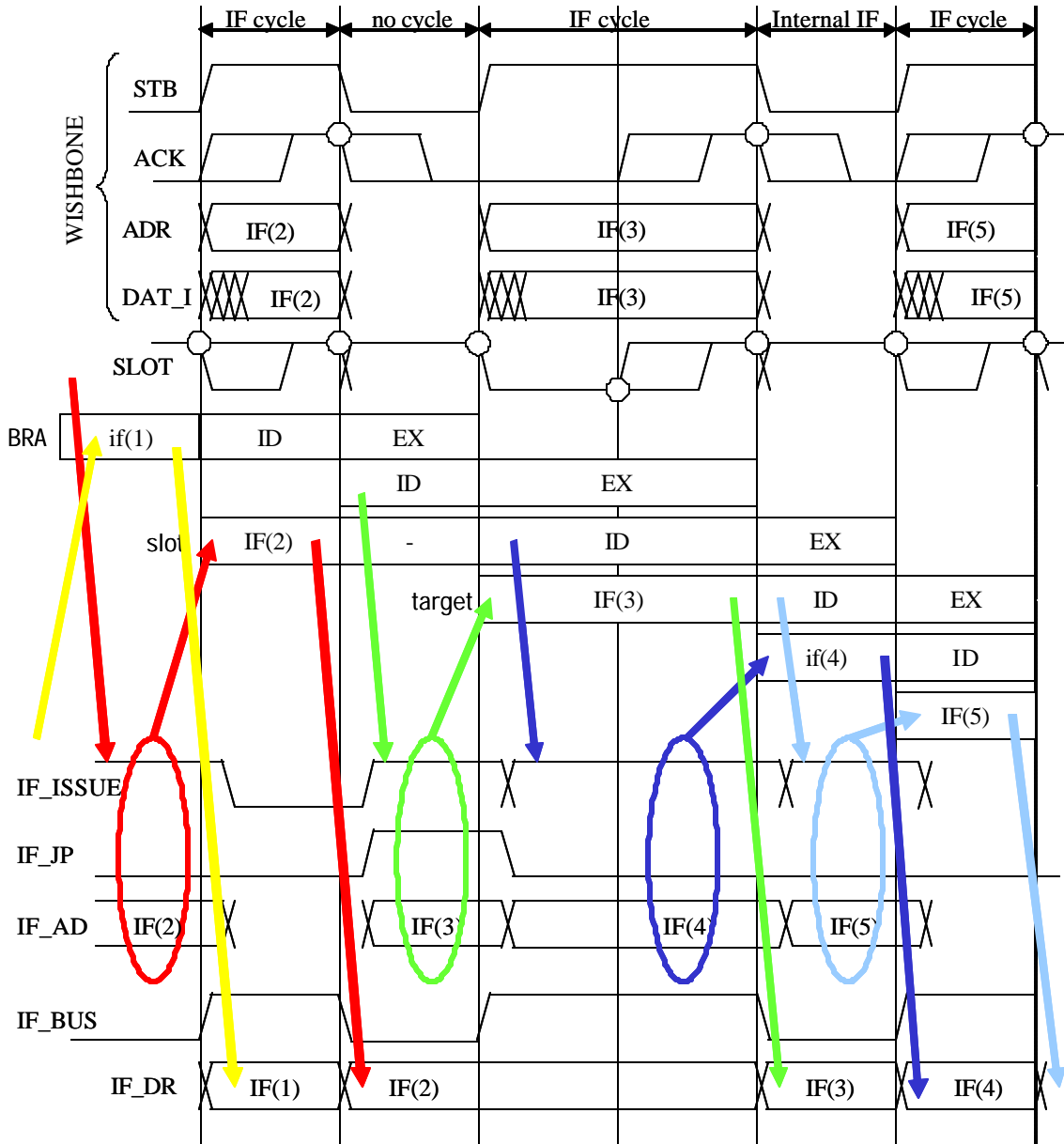
### 11.3. Instruction Fetch Cycle

The decoder unit requests instruction fetch to the memory access control unit. Some examples of instruction fetch controls are shown in Figure 11.2.

The instruction fetch starts at next slot of IF\_ISSUE=1. When IF\_ISSUE=1, IF\_AD[31:0] and IF\_JP should be valid state. IF\_AD[31:0] shows the address of instruction which the decoder unit want to get.

If external bus width is 32bit (IF\_WIDTH=1), 2 instructions are fetched simultaneously. This means the memory access control unit creates actual memory access for instruction fetch every two slots, using internal fetch buffer. But if the instruction fetch is created by Jump or Branch, the fetch should actually access the memory even if the internal fetch buffer has been valid. So, Jumping operation by instruction or exception sequence should

inform such state to memory access control unit by asserting IF\_JP with IF\_ISSUE. The fetched instruction IF\_DR[15:0] is valid at next slot of corresponding IF cycle.



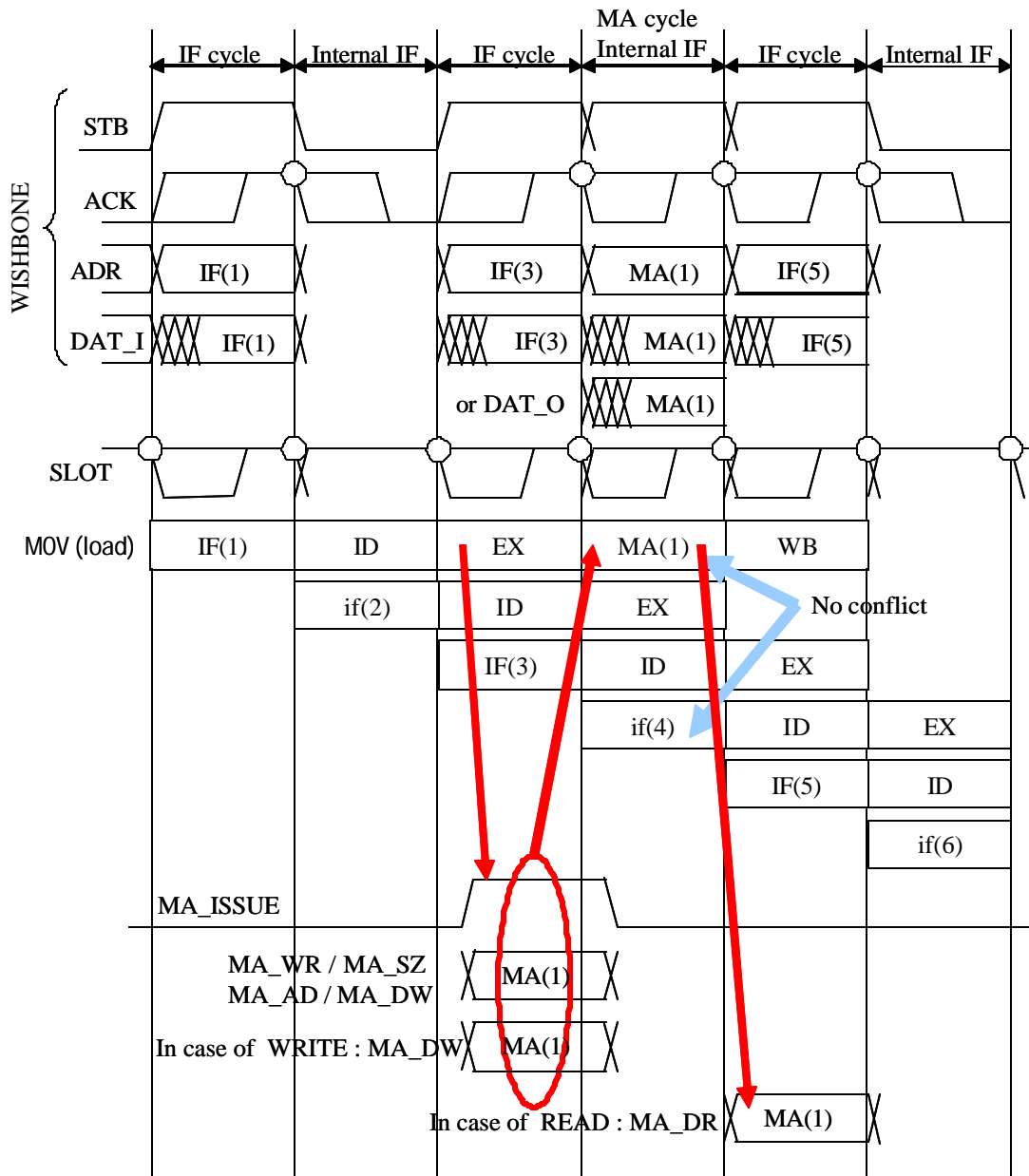
**Figure11.2 Instruction Fetch Cycle**

### 11.4. Memory Access Cycle

Figure11.3 shows memory access control. Similar to instruction fetch, MA starts at next slot of MA\_ISSUE=1. Some attribute information such as access size MA\_SZ[1:0], access

direction MA\_RW, address MA\_AD[31:0] and, if write access, write data MA\_WD[31:0] should be valid when IF\_ISSUE=1.

The write data in MA\_DW[31:0] should be valid in its LSB side when access size is smaller than long word. The read data MA\_DR[31:0] is valid with sign extended at next slot of corresponding MA cycle.



**Figure11.3 Memory Access Cycle**

### 11.5. IF-MA Conflict

Figure 11.4 shows the IF-MA conflict. At 3<sup>rd</sup> slot, IF\_ISSUE and MA\_ISSUE are asserted at same time. If the IF should get a instruction from external memory (not from internal instruction buffer), IF\_ISSUE=1 & MA\_ISSUE=1 means IF-MA contention.

When IF and MA conflict, the memory access control unit asserts the signal IF\_STALL and inform such situation to the decoder unit. The memory access control unit starts MA cycle first, and after the MA, it begins IF cycle.

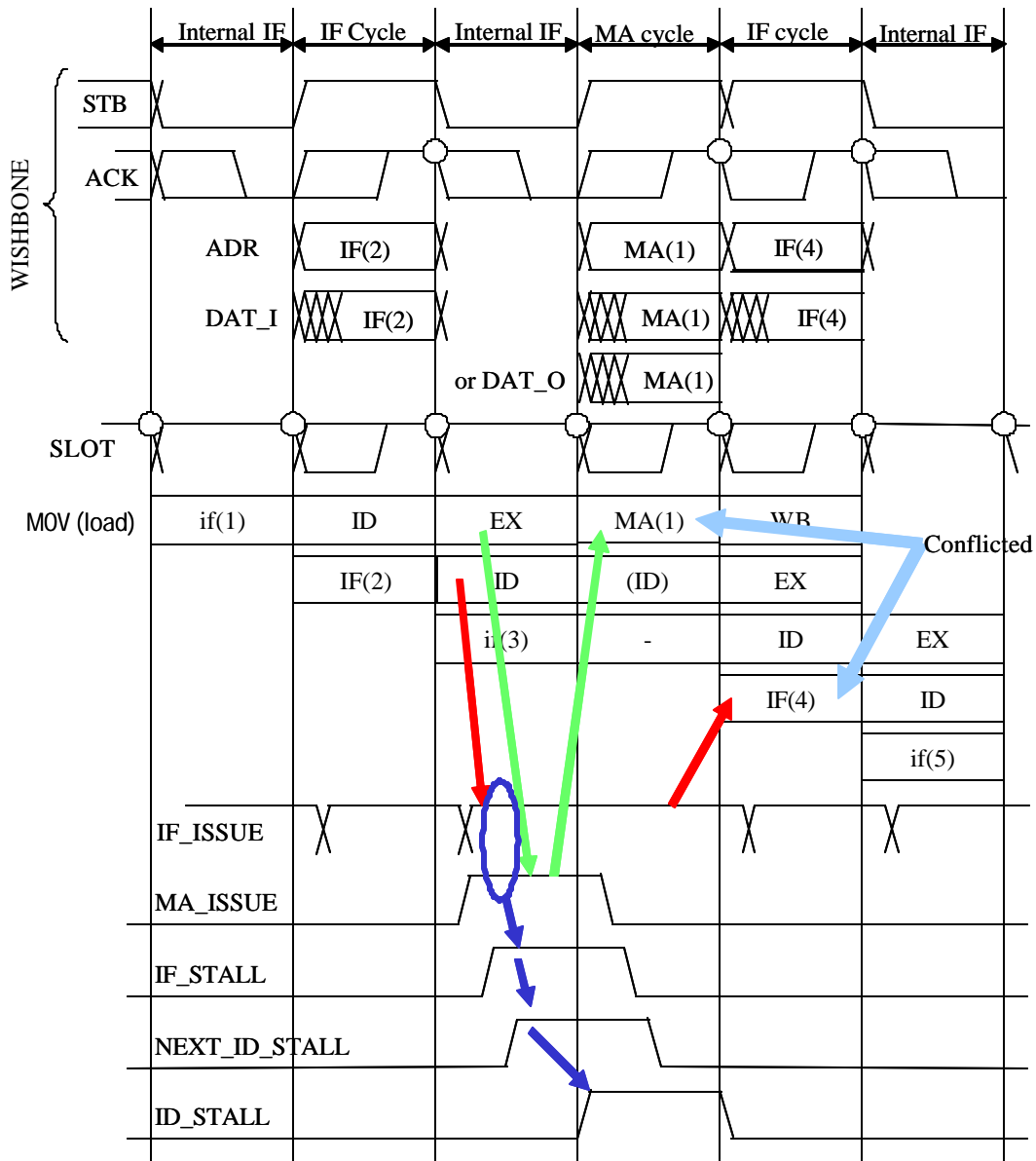


Figure 11.4 IF-MA Conflict

### 11.6. Bus Width of Instruction Fetch Cycle (IF\_WIDTH)

If data width of instruction fetch space is 32bit, the WISHBONE bus should return IF\_WIDTH=1, or it should return IF\_WIDTH=0. If IF\_WIDTH=0, internal fetch buffer of the memory access control unit can get only one instruction, so next instruction fetch requested by decoder unit should produce actual memory access, as shown in Figure 11.5. Note that IF\_WIDTH has its meaning only when lower 2bit of fetch address is 2'b00 as described in Part 1.

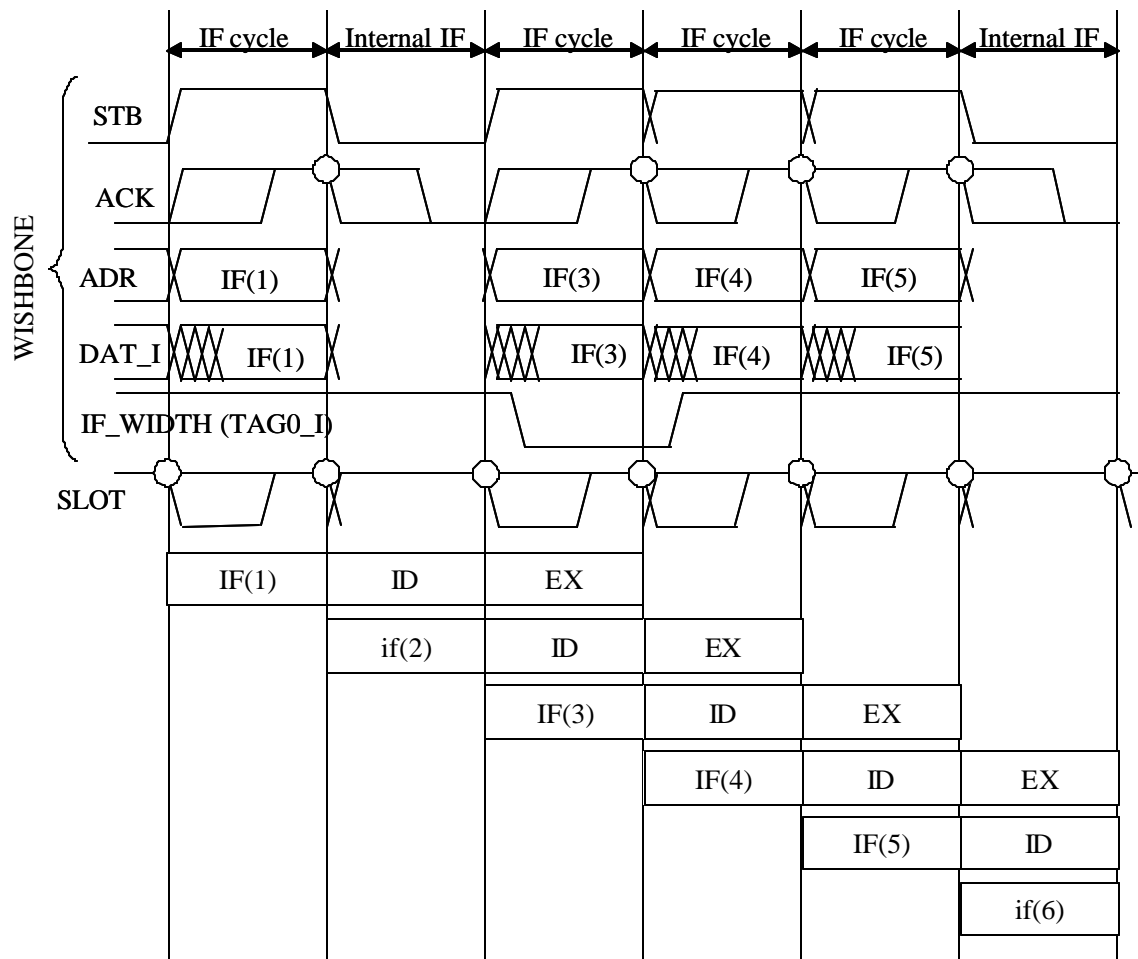


Figure 11.5 Bus Width of Instruction Fetch Cycle (IF\_WIDTH)

### 11.7. Read Modify Write Cycle (for Instruction TAS.B)

The specification of WISHBONE bus has read-modify-write cycle, in which no bus arbitration is granted between read and write. During read-modify-write cycle, the bus master should keep CYC signal high. The TAS.B (test and set) instruction requires such

read-modify-write cycle. To achieve this, the memory access control unit receives KEEP\_CYC signal from decoder unit, as shown in Figure 11.6.

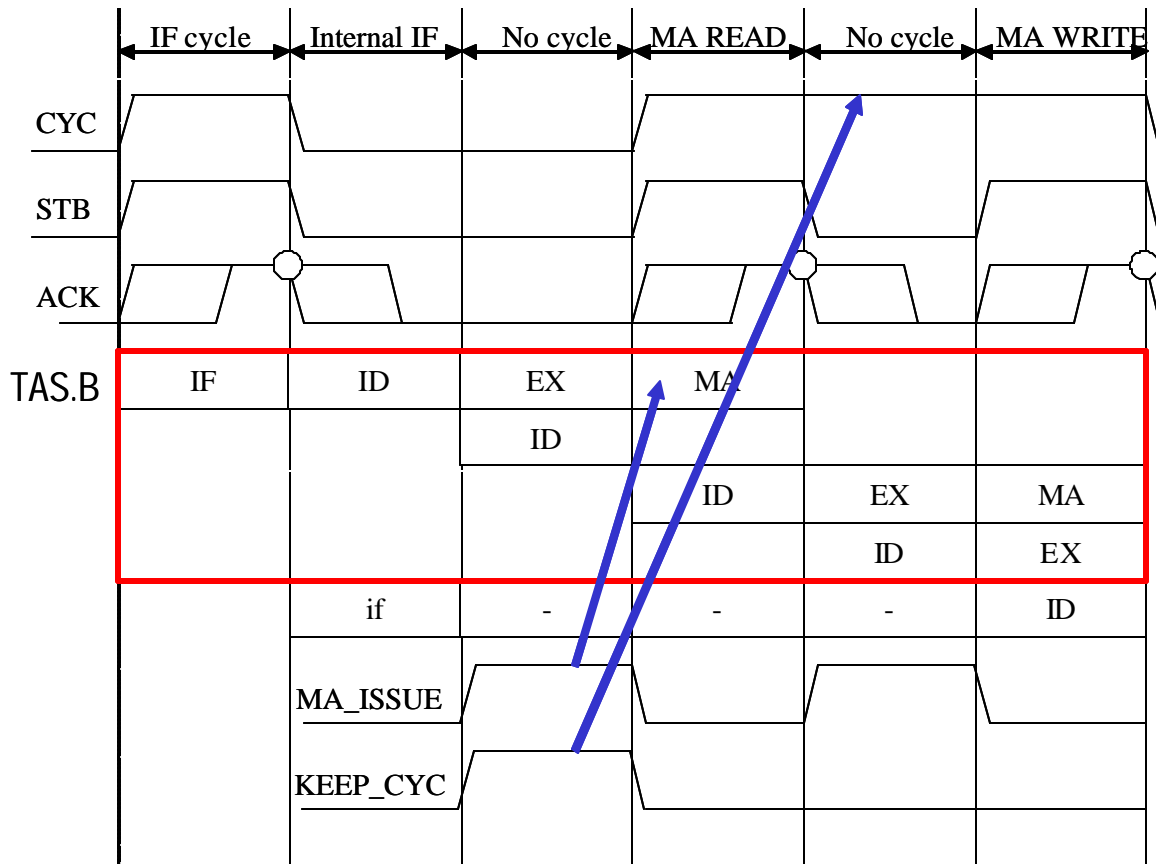


Figure 11.6 Read Modify Write Cycle

### 11.8.State Machine of Memory Access Control Unit

Table 11.2 shows state definition of memory access control unit. Table 11.3 shows key signals regarding instruction fetch. And Table 11.4 shows state transitions.

This memory access control unit assumes the CPU bus (WISHBONE) is non-Harvard bus. If you want to modify Aquarius to support Harvard bus, what you should do is (1) modifying the memory access control unit to connect both instruction bus and data bus, and (2) making decoder unit inform to memory access controller if the MA cycle is PC relative or not. Note that, even if you adopt Harvard bus with Aquarius, the PC relative instructions (MOV.L/W @(disp, PC), Rn) must access to instruction space, so this access conflicts to IF cycle. The memory controller for Harvard bus should still return IF\_STALL to decoder.

| State | Symbol      | Meaning  |
|-------|-------------|--|
| S0    | S_IDLE      | Idle state   |
| S1    | S_IFEX      | Instruction fetch with external memory read access             |
| S2    | S_MAEX      | Data access with external memory read/write access             |
| S3    | S_MAEX_IFPD | Data access with pending instruction fetch (IF-MA conflict)    |
| S4    | S_IDLE_IFKP | Idle state but internal instruction buffer keeps a instruction |
| S5    | S_IFIN      | Instruction fetch from internal instruction buffer             |
| S6    | S_MAEX_IFKP | Data access with keeping a instruction in the buffer           |
| S7    | S_MAEX_IFIN | Data access from memory and Instruction fetch from buffer      |

**Table11.2 State definition of memory access control unit**

| Signal   | Meaning   |
|----------|---|
| IF_KEEP  | Instruction fetch from long boundary address & IF_WIDTH=1 (32bit width) |
| IF_FORCE | Next instruction fetch is from long boundary   IF_JP=1                  |

**Table11.3 Key signals regarding instruction fetch control**

| State |      | Reason for state transition                      | State |      | Reason for state transition                               |
|-------|------|--|-------|------|---|
| Now   | Next |  | Now   | Next |   |
| S0    | S0   | no event   | S4    | S0   | n/a   |
|       | S1   | by fetch request                                 |       | S1   | by fetch request: IF_FORCE=1                              |
|       | S2   | by data access request                           |       | S2   | n/a   |
|       | S3   | by both fetch request and data access request    |       | S3   | by both fetch request and data access request: IF_FORCE=1 |
|       | S4   | n/a  |       | S4   | no event  |
|       | S5   | n/a  |       | S5   | by fetch request: IF_FORCE=0                              |
|       | S6   | n/a  |       | S6   | by data access request                                    |
|       | S7   | n/a  |       | S7   | by both fetch request and data access request: IF_FORCE=0 |
| S1    | S0   | no event: IF_KEEP=0                              | S5    | S0   | no event  |
|       | S1   | by fetch request: IF_FORCE=1 or IF_KEEP=0        |       | S1   | by fetch request  |
|       | S2   | by data access request: IF_KEEP=0                |       | S2   | by data access request                                    |
|       | S3   | by both fetch and data access request: IF_KEEP=0 |       | S3   | by both fetch request and data access request             |
|       | S4   | no event: IF_KEEP=1                              |       | S4   | n/a   |
|       | S5   | by fetch request: IF_FORCE=0 and IF_KEEP=1       |       | S5   | n/a   |
|       | S6   | by data access request: IF_KEEP=1                |       | S6   | n/a   |
|       | S7   | by both fetch and data access request: IF_KEEP=1 |       | S7   | n/a   |
| S2    | S0   | no event   | S6    | S0   | n/a   |
|       | S1   | by fetch request                                 |       | S1   | by fetch request: IF_FORCE=1                              |
|       | S2   | by data access request                           |       | S2   | n/a   |
|       | S3   | by both fetch request and data access request    |       | S3   | by both fetch request and data access request: IF_FORCE=1 |
|       | S4   | n/a  |       | S4   | no event  |
|       | S5   | n/a  |       | S5   | by fetch request: IF_FORCE=0                              |
|       | S6   | n/a  |       | S6   | by data access request                                    |
|       | S7   | n/a  |       | S7   | by both fetch request and data access request: IF_FORCE=0 |
| S3    | S0   | n/a  | S7    | S0   | no event  |
|       | S1   | always   |       | S1   | by fetch request  |
|       | S2   | n/a  |       | S2   | by data access request                                    |
|       | S3   | n/a  |       | S3   | by both fetch request and data access request             |
|       | S4   | n/a  |       | S4   | n/a   |
|       | S5   | n/a  |       | S5   | n/a   |
|       | S6   | n/a  |       | S6   | n/a   |
|       | S7   | n/a  |       | S7   | n/a   |

**Table11.4 State transition of memory access control unit**



## 12. Data Path Unit

This chapter describes the details of data path unit (`datapath.v`).

### 12.1.IN/OUT Signal Table

Table12.1 shows all in/out signals of data path unit.

| Class                     | Direction | Name               | From / To  | Meaning  | Notes |
|---------------------------|-----------|--------------------|------------|--|-------|
| System Signals            | input     | CLK                | EXTERNAL   | clock  |       |
|                           | input     | RST                | EXTERNAL   | reset  |       |
| SLOT                      | input     | SLOT               | mem.v      | cpu pipe slot  |       |
| General Register Controls | input     | RDREG_X            | decode.v   | read Rn to X-bus   |       |
|                           | input     | RDREG_Y            | decode.v   | read Rn to Y-bus   |       |
|                           | input     | WRREG_Z            | decode.v   | write Rn from Z-bus  |       |
|                           | input     | WRREG_W            | decode.v   | write Rn from W-bus  |       |
|                           | input     | [ 3 : 0 ] REGNUM_X | decode.v   | register number to read to X-bus                           |       |
|                           | input     | [ 3 : 0 ] REGNUM_Y | decode.v   | register number to read to Y-bus                           |       |
|                           | input     | [ 3 : 0 ] REGNUM_Z | decode.v   | register number to write from Z-bus                        |       |
| ALU                       | input     | [ 4 : 0 ] ALUFUNC  | decode.v   | ALU function   |       |
|                           | output    | [ 31 : 0 ] MA_AD   | datapath.v | memory access address                                      |       |
| Memory Access Controls    | output    | [ 31 : 0 ] MA_DW   | datapath.v | memory write data  |       |
|                           | input     | [ 31 : 0 ] MA_DR   | datapath.v | memory read data   |       |
|                           | input     | WRMAAD_Z           | decode.v   | output MA_AD from Z-bus                                    |       |
|                           | input     | WRMADW_X           | decode.v   | output MA_DW from X-bus                                    |       |
|                           | input     | WRMADW_Y           | decode.v   | output MA_DW from Y-bus                                    |       |
|                           | input     | RDMADR_W           | decode.v   | input MA_DR to W-bus                                       |       |
| Multiplier Controls       | output    | [ 31 : 0 ] MACIN1  | datapath.v | data1 to mult.v  |       |
|                           | output    | [ 31 : 0 ] MACIN2  | datapath.v | data2 to mult.v  |       |
|                           | input     | [ 1 : 0 ] MACSEL1  | decode.v   | select data of MACIN1<br>(00:from X, 01:from Z, 1?:from W) |       |
|                           | input     | [ 1 : 0 ] MACSEL2  | decode.v   | select data of MACIN2<br>(00:from Y, 01:from Z, 1?:from W) |       |
|                           | input     | [ 31 : 0 ] MACH    | datapath.v | physical data of MACH                                      |       |
|                           | input     | [ 31 : 0 ] MACL    | datapath.v | physical data of MACL                                      |       |
|                           | input     | RDMACH_X           | decode.v   | read MACH to X-bus   |       |
|                           | input     | RDMACL_X           | decode.v   | read MACL to X-bus   |       |
|                           | input     | RDMACH_Y           | decode.v   | read MACH to Y-bus   |       |
|                           | input     | RDMACL_Y           | decode.v   | read MACL to Y-bus   |       |

**Table12.1 Data Path Unit IN/OUT Signals (1)**

| Class                       | Direction | Name                 | From / To | Meaning  | Notes |
|-----------------------------|-----------|----------------------|-----------|--|-------|
| Status Register Controls    | input     | RDSR_X               | decode.v  | read SR to X-bus                                     |       |
|                             | input     | RDSR_Y               | decode.v  | read SR to Y-bus                                     |       |
|                             | input     | WRSR_Z               | decode.v  | write SR from Z-bus                                  |       |
|                             | input     | WRSR_W               | decode.v  | write SR from W-bus                                  |       |
|                             | output    | MAC_S                | mult.v    | latched S bit in SR (= SR[S])                        |       |
|                             | input     | MAC_S_LATCH          | decode.v  | latch command of S bit in SR                         |       |
| GBR Controls                | input     | RDGBR_X              | decode.v  | read GBR to X-bus                                    |       |
|                             | input     | RDGBR_Y              | decode.v  | read GBR to Y-bus                                    |       |
|                             | input     | WRGBR_Z              | decode.v  | write GBR from Z-bus                                 |       |
|                             | input     | WRGBR_W              | decode.v  | write GBR from W-bus                                 |       |
| VBR Controls                | input     | RDVBR_X              | decode.v  | read VBR to X-bus                                    |       |
|                             | input     | RDVBR_Y              | decode.v  | read VBR to Y-bus                                    |       |
|                             | input     | WRVBR_Z              | decode.v  | write VBR from Z-bus                                 |       |
|                             | input     | WRVBR_W              | decode.v  | write VBR from W-bus                                 |       |
| Procedure Register Controls | input     | RDPR_X               | decode.v  | read PR to X-bus                                     |       |
|                             | input     | RDPR_Y               | decode.v  | read PR to Y-bus                                     |       |
|                             | input     | WRPR_Z               | decode.v  | write PR from Z-bus                                  |       |
|                             | input     | WRPR_W               | decode.v  | write PR from W-bus                                  |       |
|                             | input     | WRPR_PC              | decode.v  | write PR from PC                                     |       |
| Program Counter Controls    | input     | RDPC_X               | decode.v  | read PC to X-bus                                     |       |
|                             | input     | RDPC_Y               | decode.v  | read PC to Y-bus                                     |       |
|                             | input     | WRPC_Z               | decode.v  | write PC from Z-bus                                  |       |
|                             | input     | INCPC                | decode.v  | increment PC (PC+2->PC)                              |       |
|                             | input     | IFADSEL              | decode.v  | select IF_AD output from INC(0) or Z-bus(1)          |       |
|                             | output    | [ 31 : 0 ] IF_AD     | mem.v     | instruction fetch address                            |       |
| Constant Value Controls     | input     | [ 15 : 0 ] CONST_IFD | decode.v  | instruction fetch data to make constant              |       |
|                             | input     | CONST_ZERO4          | decode.v  | take constant from lower 4 bit as unsigned value     |       |
|                             | input     | CONST_ZERO42         | decode.v  | take constant from lower 4 bit as unsigned value * 2 |       |
|                             | input     | CONST_ZERO44         | decode.v  | take constant from lower 4 bit as unsigned value * 4 |       |
|                             | input     | CONST_ZERO8          | decode.v  | take constant from lower 8 bit as unsigned value     |       |
|                             | input     | CONST_ZERO82         | decode.v  | take constant from lower 8 bit as unsigned value * 2 |       |
|                             | input     | CONST_ZERO84         | decode.v  | take constant from lower 8 bit as unsigned value * 4 |       |
|                             | input     | CONST_SIGN8          | decode.v  | take constant from lower 8 bit as signed value       |       |
|                             | input     | CONST_SIGN82         | decode.v  | take constant from lower 8 bit as signed value * 2   |       |
|                             | input     | CONST_SIGN122        | decode.v  | take constant from lower 12 bit as signed value * 2  |       |
|                             | input     | RDCONST_X            | decode.v  | read constant to X-bus                               |       |
|                             | input     | RDCONST_Y            | decode.v  | read constant to Y-bus                               |       |

**Table12.1 Data Path Unit IN/OUT Signals (2)**

| Class                               | Direction | Name              | From / To | Meaning                               | Notes |
|-------------------------------------|-----------|-------------------|-----------|---------------------------------------|-------|
| Forwarding                          | input     | REG_FWD_X         | decode.v  | register forward from W-bus to X-bus  |       |
|                                     | input     | REG_FWD_Y         | decode.v  | register forward from W-bus to Y-bus  |       |
| Comparator                          | input     | [ 2 : 0 ] CMPCOM  | decode.v  | define comparator operation (command) |       |
| Shifter                             | input     | [ 4 : 0 ] SFTFUNC | decode.v  | Shifter Function                      |       |
| Controls                            | input     | RDSFT_Z           | decode.v  | read SFTOUT to Z-BUS                  |       |
| T bit<br>Q bit<br>M bit<br>Controls | output    | T_BCC             | decode.v  | T value for Bcc judgement             |       |
|                                     | input     | T_CMPSET          | decode.v  | reflect comparator result to T        |       |
|                                     | input     | T_CRYSET          | decode.v  | reflect carry/borrow out to T         |       |
|                                     | input     | T_TSTSET          | decode.v  | reflect tst result to T               |       |
|                                     | input     | T_SFTSET          | decode.v  | reflect shifted output to T           |       |
|                                     | input     | QT_DV1SET         | decode.v  | reflect DIV1 result to Q and T        |       |
|                                     | input     | MQT_DV0SET        | decode.v  | reflect DIV0S result to M, Q and T    |       |
|                                     | input     | T_CLR             | decode.v  | clear T                               |       |
|                                     | input     | T_SET             | decode.v  | set T                                 |       |
|                                     | input     | MQ_CLR            | decode.v  | clear M and Q                         |       |
| TEMP                                | input     | RDTEMP_X          | decode.v  | read TEMP to X-bus                    |       |
| Register                            | input     | WRTEMP_Z          | decode.v  | write to TEMP from Z-bus              |       |
| Controls                            | input     | WRMAAD_TEMP       | decode.v  | output MAAD from TEMP                 |       |
| SR and I bit<br>Controls            | input     | RST_SR            | decode.v  | reset SR                              |       |
|                                     | output    | [ 3 : 0 ] IBIT    | decode.v  | I bit in SR                           |       |
|                                     | input     | [ 3 : 0 ] ILEVEL  | decode.v  | IRQ Level                             |       |
|                                     | input     | WR_IBIT           | decode.v  | Write ILEVEL to I bit in SR           |       |

**Table12.1 Data Path Unit IN/OUT Signals (3)**

## 12.2. Structure of Data Path

Figure12.1 shows the block diagram of data path unit. It also shows basic relationship among data path, decoder, multiplier and memory access controller. In data path RTL description (datapath.v), the general registers R0-R15 (register.v) are located in under data path layer. Data path has 4 internal buses.

X-bus : Data from each register resource

Y-bus : Data from each register resource

Z-bus : Data from results of ALU or Shifter

W-bus : Data from memory load (to be written back to each register resource)

In the data path, all resources are fully controlled by decoder unit. So, there are no state machines in data path unit.

The register forwarding paths are shown in top of Figure12.1 as direct paths from W to X and W to Y.

The T bit, Q bit and M bit in Status Register (SR) are created from several signals as shown in Figure12.2, Figure12.3 and Figure12.4.

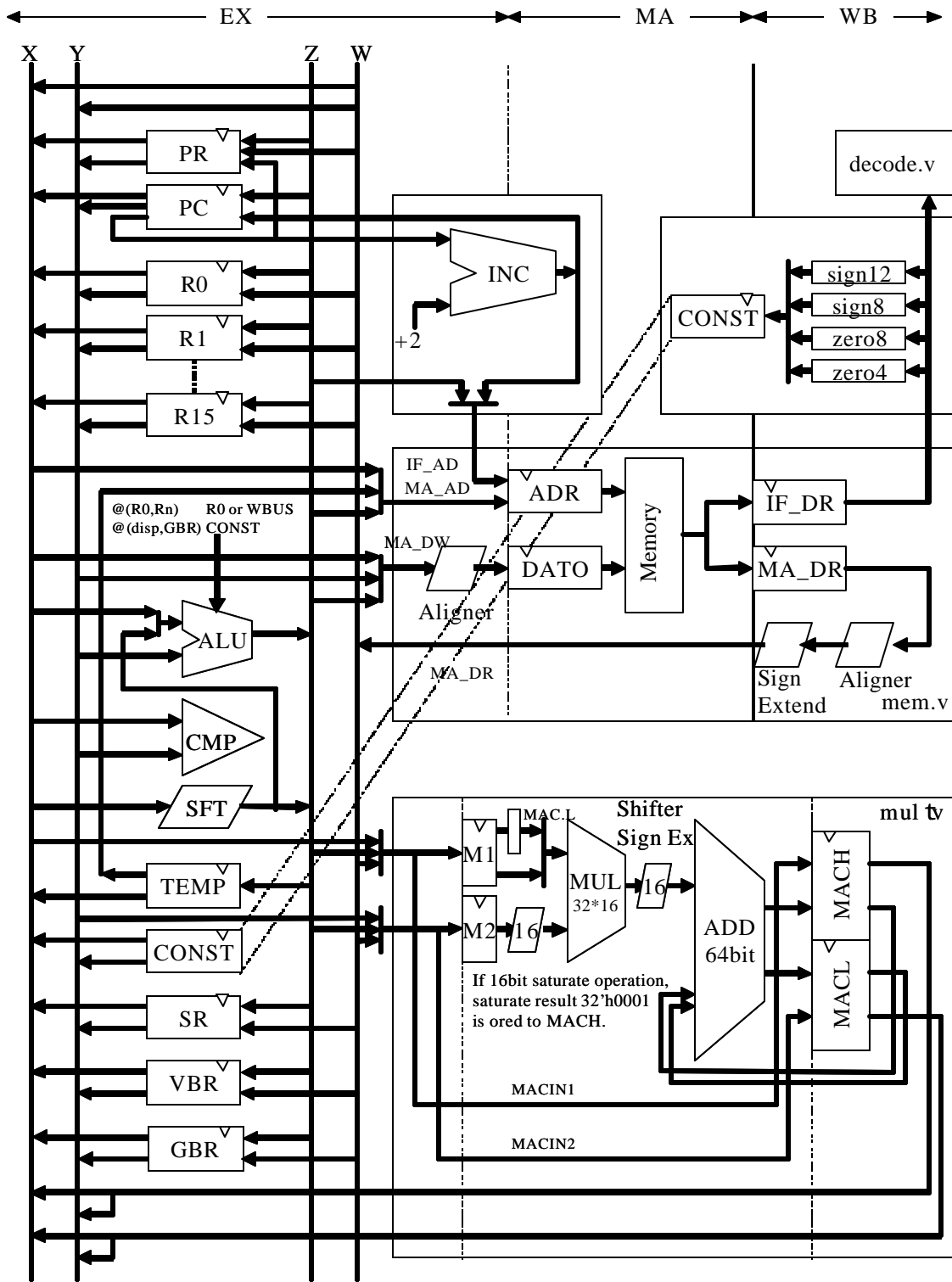
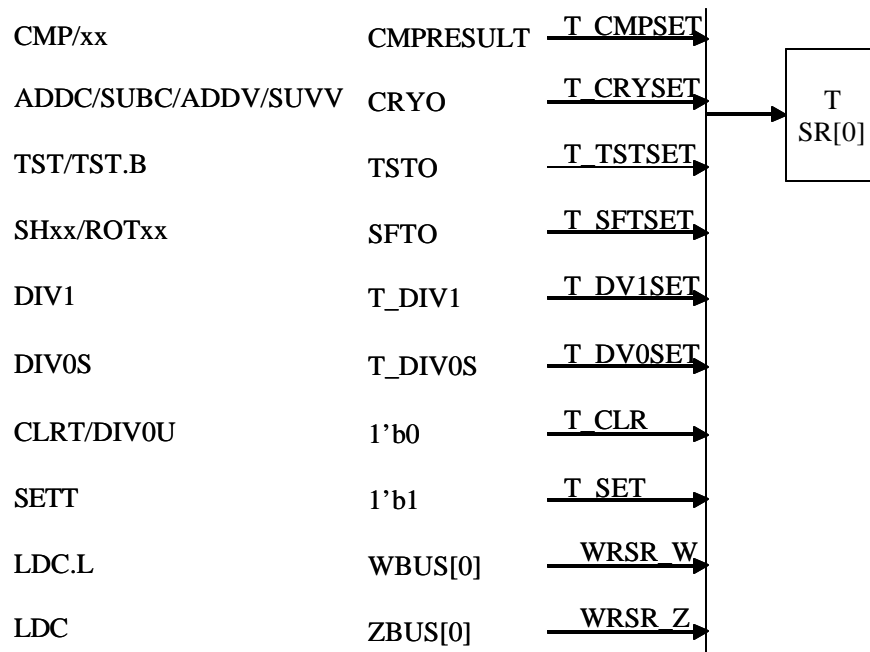
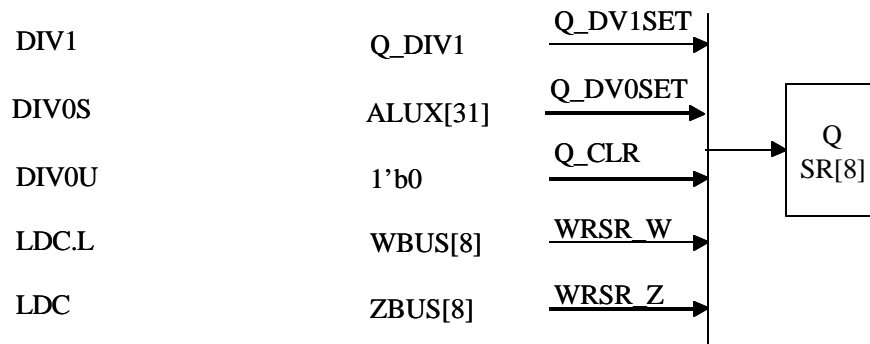


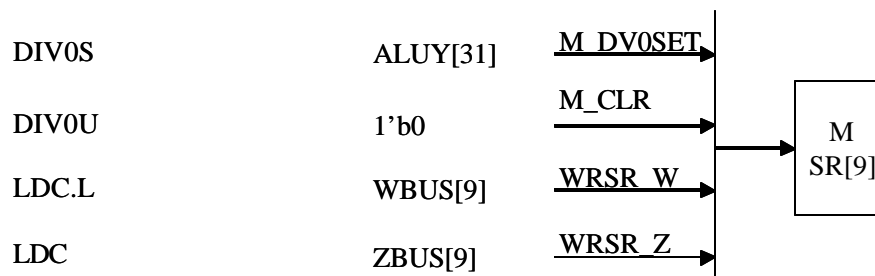
Figure12.1 Block Diagram of Data Path Unit



**Figure12.2 Generating T bit**



**Figure12.3 Generating Q bit**



**Figure12.4 Generating M bit**

## 13. Multiplier Unit

This chapter describes the details of multiplier unit (`mult.v`).

### 13.1. IN/OUT Signal Table

Table13.1 shows all in/out signals of multiplier unit.

| Class          | Direction | Name              | From / To    | Meaning                                       | Notes |
|----------------|-----------|-------------------|--------------|---|-------|
| System Signals | input     | CLK               |              | clock   |       |
|                | input     | RST               |              | reset   |       |
| SLOT           | input     | SLOT              |              | cpu pipe slot                                 |       |
|                | input     | MULCOM1           |              | M1 latch command                              |       |
| Mult Commands  | input     | [ 7 : 0 ] MULCOM2 |              | M2 latch and mult engage command              |       |
|                |           | NOP               | 0 0000000 00 |   |       |
|                |           | DMULS.L           | 1 0111101 BD |   |       |
|                |           | DMULU.L           | 1 0110101 B5 |   |       |
|                |           | MAC.L             | 1 0001111 8F |   |       |
|                |           | MAC.W             | 1 1001111 CF |   |       |
|                |           | MULL              | 1 0000111 87 |   |       |
|                |           | MULS.W            | 1 0101111 AF |   |       |
|                |           | MULU.W            | 1 0101110 AE |   |       |
| S bit          | input     | MAC_S             |              | S-bit in SR                                   |       |
| Data Interface | input     | WRMACH, WRMACL    |              | write MACH and MACL directly from data path   |       |
|                | input     | [ 31 : 0 ] MACIN1 |              | input data 1                                  |       |
|                | input     | [ 31 : 0 ] MACIN2 |              | input data 2                                  |       |
|                | output    | [ 31 : 0 ] MACH   |              | output MACH                                   |       |
|                | output    | [ 31 : 0 ] MACL   |              | output MACL                                   |       |
| Status         | output    | MAC_BUSY          |              | busy signal (negate at final operation state) |       |

**Table13.1 Multiplier Unit IN/OUT Signals**

### 13.2. Algorithm of Multiplication

Basically, this multiplier design assumes that it is implemented by using Macro Module of Multiplier for FPGA. So, existence of *unsigned* 32bit x 16bit (or similar) multiplier is supposed. Now, let me define some symbols to explain.

As[N-1:0] = Assumed as Signed N bit

Au[N-1:0] = Assumed as Unsigned N bit     Au[x]=As[x] (each bit is same)

Bs[N-1:0] = Assumed as Signed N bit

Bu[N-1:0] = Assumed as Unsigned N bit     Bu[x]=Bs[x] (each bit is same)

**(1) Signed 32bit x 32bit**

$$As[31:0] = -2^{31} \times Au[31] + Au[30:0]$$

$$Bs[31:0] = -2^{31} \times Bu[31] + Bu[30:0]$$

$$\begin{aligned} &As[31:0] \times Bs[31:0] \\ &= 2^{62} \times Au[31] \times Bu[31] \\ &\quad - 2^{31} \times Au[31] \times Bu[30:0] \\ &\quad - 2^{31} \times Bu[31] \times Au[30:0] \\ &\quad + Au[30:0] \times Bu[30:0] \\ &= 2^{62} \times Au[31] \times Bu[31] \\ &\quad - 2^{31} \times Au[31] \times Bu[30:0] \\ &\quad - 2^{31} \times Bu[31] \times Au[30:0] \\ &\quad + 2^{16} \times Au[30:0] \times Bu[30:16] \\ &\quad + Au[30:0] \times Bu[15:0] \\ &= P1 - P2 - P3 + P4 + P5 \end{aligned}$$

$$P1 = 2^{62} \times Au[31] \times Bu[31]$$

$$P2 = 2^{31} \times Au[31] \times Bu[30:0]$$

$$P3 = 2^{31} \times Bu[31] \times Au[30:0]$$

$$P4 = 2^{16} \times Au[30:0] \times Bu[30:16]$$

$$P5 = Au[30:0] \times Bu[15:0]$$

**(2) Unsigned 32bit x 32bit**

$$Au[31:0] = +2^{31} \times Au[31] + Au[30:0]$$

$$Bu[31:0] = +2^{31} \times Bu[31] + Bu[30:0]$$

$$\begin{aligned} &Au[31:0] \times Bu[31:0] \\ &= 2^{62} \times Au[31] \times Bu[31] \\ &\quad + 2^{31} \times Au[31] \times Bu[30:0] \\ &\quad + 2^{31} \times Bu[31] \times Au[30:0] \\ &\quad + Au[30:0] \times Bu[30:0] \\ &= 2^{62} \times Au[31] \times Bu[31] \\ &\quad + 2^{31} \times Au[31] \times Bu[30:0] \\ &\quad + 2^{31} \times Bu[31] \times Au[30:0] \\ &\quad + 2^{16} \times Au[30:0] \times Bu[30:16] \\ &\quad + Au[30:0] \times Bu[15:0] \\ &= P1 + P2 + P3 + P4 + P5 \end{aligned}$$

$$P1 = 2^{62} \times Au[31] \times Bu[31]$$

$$P2 = 2^{31} \times Au[31] \times Bu[30:0]$$

$$P3 = 2^{31} \times Bu[31] \times Au[30:0]$$

$$P4 = 2^{16} \times Au[30:0] \times Bu[30:16]$$

$$P5 = Au[30:0] \times Bu[15:0]$$

### (3) Signed 16bit x 16bit

$$As[15:0] = -2^{15} \times Au[15] + Au[14:0]$$

$$Bs[15:0] = -2^{15} \times Bu[15] + Bu[14:0]$$

$$As[15:0] \times Bs[15:0]$$

$$= 2^{30} \times Au[15] \times Bu[15]$$

$$- 2^{15} \times Au[15] \times Bu[14:0]$$

$$- 2^{15} \times Bu[15] \times Au[14:0]$$

$$+ Au[14:0] \times Bu[14:0]$$

$$= P1 - P2 - P3 + P4$$

$$P1 = 2^{30} \times Au[15] \times Bu[15]$$

$$P2 = 2^{15} \times Au[15] \times Bu[14:0]$$

$$P3 = 2^{15} \times Bu[15] \times Au[14:0]$$

$$P4 = Au[14:0] \times Bu[14:0]$$

### (4) Unsigned 16bit x 16bit

$$Au[15:0] = +2^{15} \times Au[15] + Au[14:0]$$

$$Bu[15:0] = +2^{15} \times Bu[15] + Bu[14:0]$$

$$Au[15:0] \times Bu[15:0]$$

$$= 2^{30} \times Au[15] \times Bu[15]$$

$$+ 2^{15} \times Au[15] \times Bu[14:0]$$

$$+ 2^{15} \times Bu[15] \times Au[14:0]$$

$$+ Au[14:0] \times Bu[14:0]$$

$$= P1 + P2 + P3 + P4$$

$$P1 = 2^{30} \times Au[15] \times Bu[15]$$

$$P2 = 2^{15} \times Au[15] \times Bu[14:0]$$

$$P3 = 2^{15} \times Bu[15] \times Au[14:0]$$

$$P4 = Au[14:0] \times Bu[14:0]$$

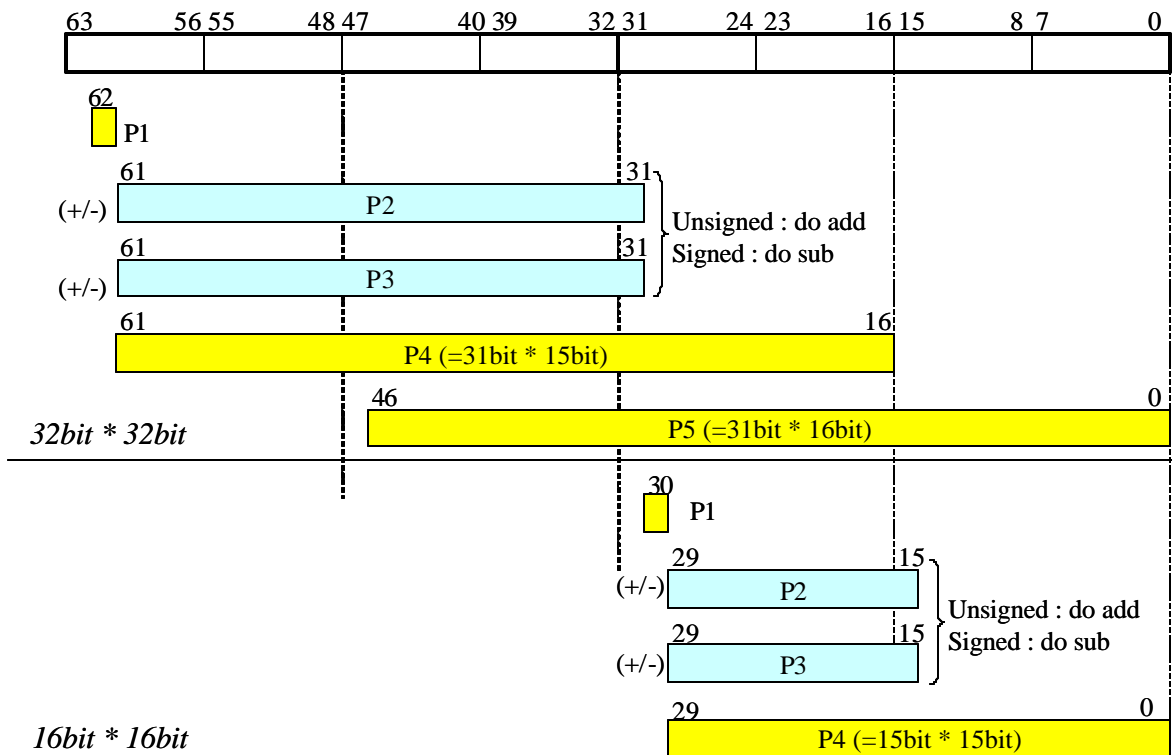
Pn are partial result to be accumulated.

Gathering above way of thoughts, Figure13.1 shows the methods of multiplication. The bit size of multiplication macro module should be at least 31bit x 16bit.

In case of 32bit multiplication, the calculation needs 2 steps. In first step, P4 is accumulated to MAC with preparing P2+P3, and in second step, P1, P2 and P3 are accumulated to MAC with 16bits shifting.

In case of 16bit multiplication, the calculation needs only 1 step. In the step, P1, P2, P3 and P4 are accumulated to MAC at once.





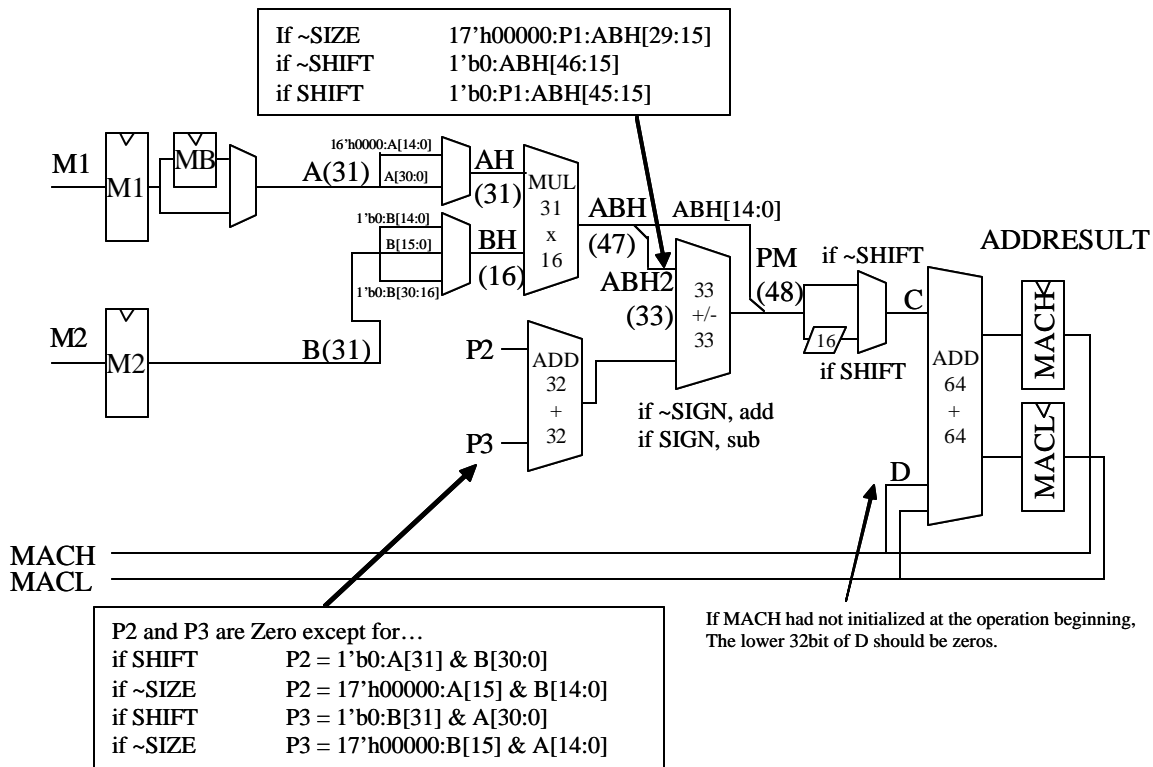
**Figure13.1 Algorithm of Multiplication**

### 13.3. Structure of Multiplier Unit

According to algorithm shown in Figure13.1, the multiplication unit has designed as shown in Figure13.2. The some control signals are created by internal state machine as shown in Table13.2. The 64bit accumulation adder should have saturation capability. It is described later.

|                |                      | SIZE | SIGN | SHIFT | Notes |
|----------------|----------------------|------|------|-------|-------|
| Signed         | 1 <sup>st</sup> step | 1    | 1    | 0     |       |
| 32bit          | 2 <sup>nd</sup> step | 1    | 1    | 1     |       |
| Unsigned       | 1 <sup>st</sup> step | 1    | 0    | 0     |       |
| 32bit          | 2 <sup>nd</sup> step | 1    | 0    | 1     |       |
| Signed 16bit   |                      | 0    | 1    | 0     |       |
| Unsigned 16bit |                      | 0    | 0    | 0     |       |

**Table13.2 Control Signals in Multiplication Unit**



**Figure13.2 Block Diagram of Multiplier Unit**

### 13.4. Control of Multiplication Unit

The decoder unit sends two kinds of multiplication command to multiplication unit. One is MULCOM1 which is latch signal of input data MACIN1[31:0]. Another is MULCOM2[7:0] which has 2 meanings; latch signal of input data MACIN2[31:0] and operation class. The MULCOM2[7] means latch signal. And MULCOM2[6:0] is same as {INSTR\_STATE[14:12], INSTR\_STATE[3:0]}. If MULCOM2[7]=0, it is NOP. Figure13.3 to Figure13.6 shows the timing position of each command. "M1" is MULCOM1, "M2" is MULCOM2. In the figures, the MAC value is determined at timing position with "MAC" and an arrow.

The instructions related to multiplication execute in multi cycles. So, if post instruction uses the result of MAC, it may be stalled.

Each multiplication instruction's decode stage asserts EX\_MAC\_BUSY or WB\_MAC\_BUSY to indicate busy state of MAC register.

On the other hand, each multiplication related instruction asserts MAC\_STALL\_SENSE at

decode stage to declare that it will use MAC resources.

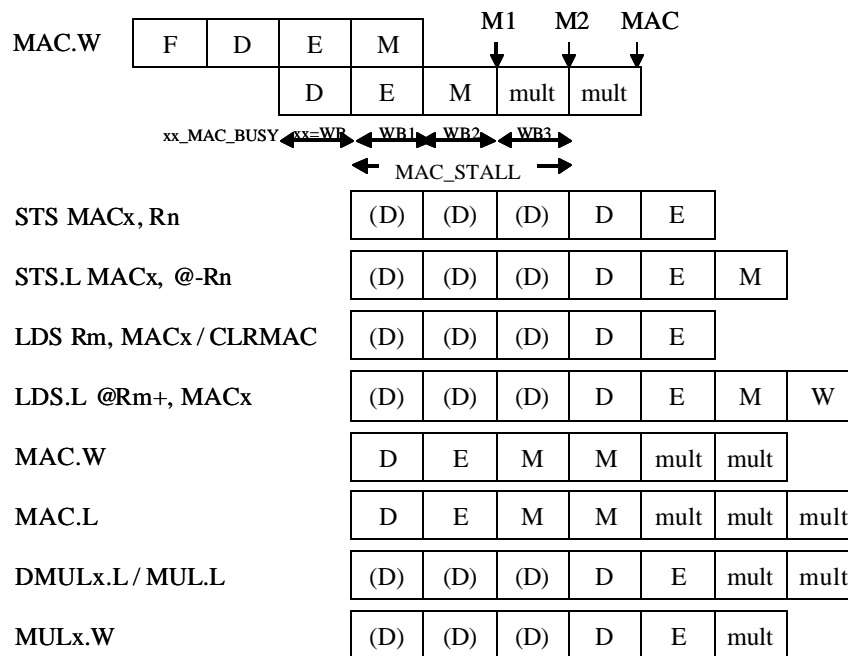
In the decoder unit, MAC\_STALL signal is created from each “pipeline shifted” xx\_MAC\_BUSY signal, MAC\_STALL\_SENSE and MAC\_BUSY (from multiplier unit which indicates second “m” stage from the last). The MAC\_STALL is used in decoder unit to control pipeline stall as shown in Figure10.7. Figure13.3 to Figure13.6 also shows how many stall cycle is necessary in the MAC conflict situation.

By the way, for example in Figure13.3, the stall counts of DMULxL / MUL.L / MULx.W can be reduced to 2 from 3, but such reduction has no meaning because the results (MACx) of these instructions should be stored to registers or memories once.

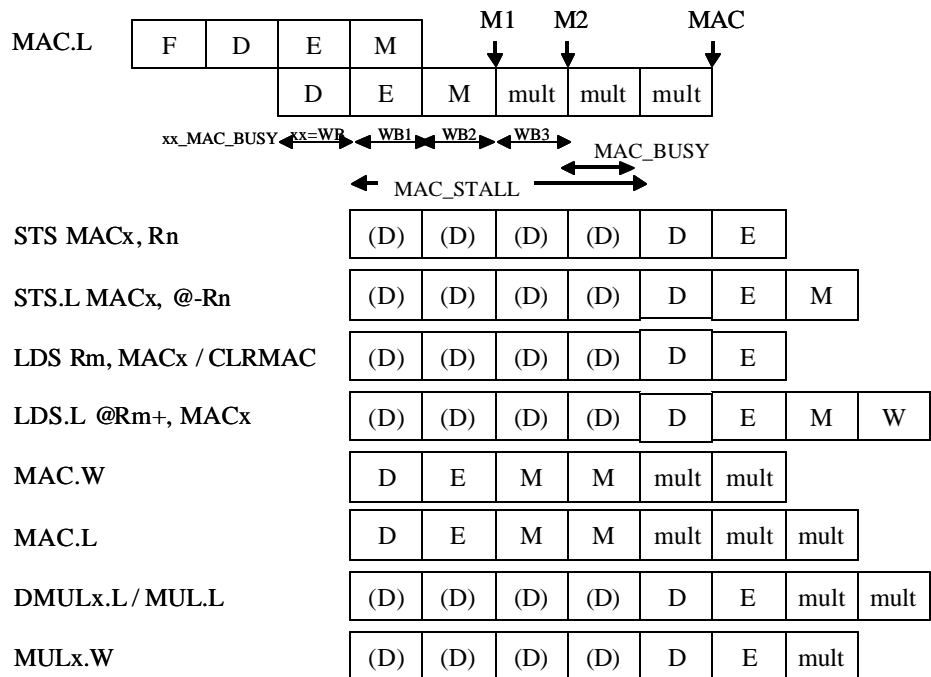
Note that the S bit in SR should be latched at second ID stage of instruction MAC.x, to avoid changing S during MAC operation. (The instruction after MAC may change S bit.)

### 13.5.How to implement Saturating Accumulation

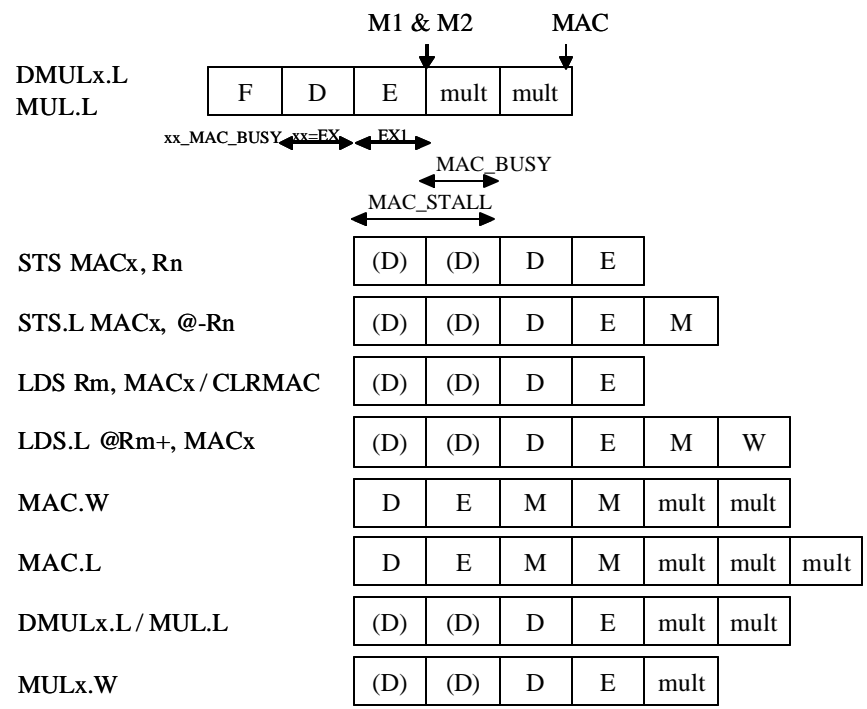
In Figure13.2, the 64 bit adder ADDRRESULT = MAC + C should have saturating function for MAC.W and MAC.L (S=1). If S=1, MAC.W should saturate between 32’h80000000 to 32’h7FFFFFFF, and MAC.L should saturate between 64’hFFFF800000000000 to 64’h00007FFFFFFFFFFFFF. To simplify explanation, consider only latter case.



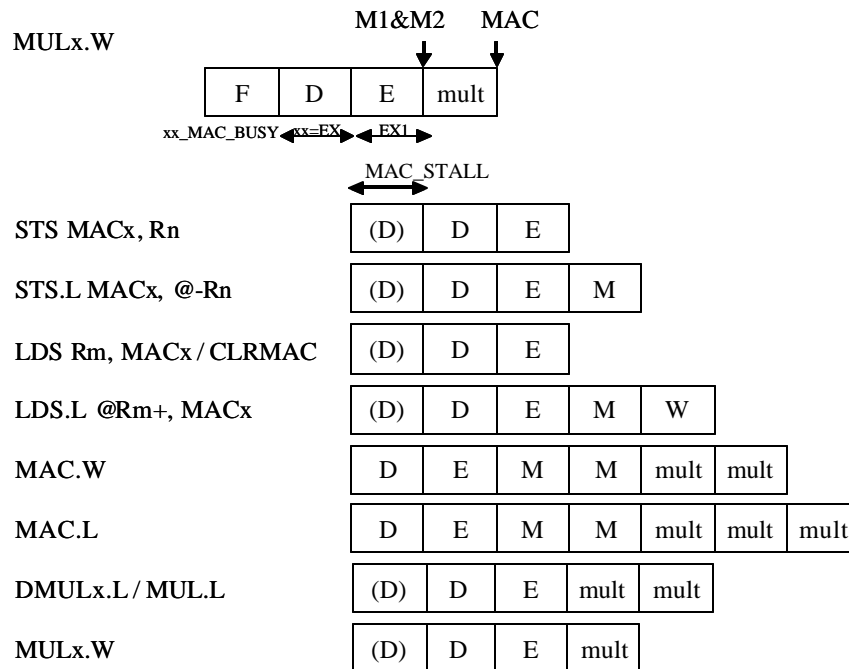
**Figure13.3 Conflict MAC.W and its post instruction**



**Figure13.4 Conflict MAC.L and its post instruction**



**Figure13.5 Conflict DMULx.L / MULL and its post instruction**

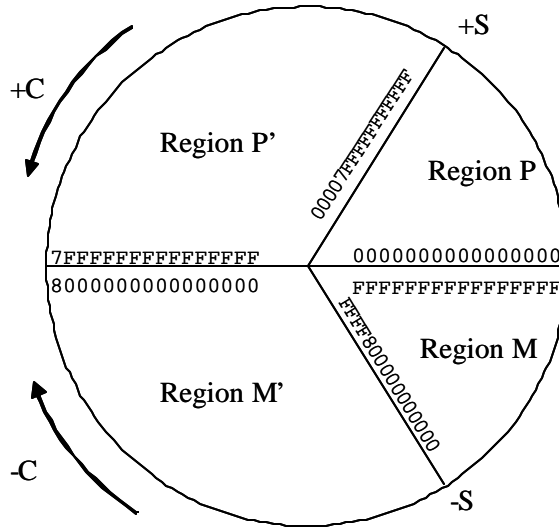


**Figure 13.6 Conflict MULx.W and its post instruction**

One of the simplest implementation of saturation is only to cut carry chain in adder circuit at proper position. But in this case, if the initial value has already been out of saturating value, the final result will not correct one. So, we should consider the initial accumulator value may be any value.

Figure 13.7 shows way of thought for correct saturation. The angle is accumulator's value = MAC. 0 degree means  $64'0000000000000000$ , 180 degrees means  $64'7FFFFFFF7FFFFFFF$ . Now, the desired saturation values is shown as +S and -S. And this plane is divided into 4 regions; P, P', M and M'. In this plane, we want to do operation add. If we add positive value (+C), the angle of MAC moves counterclockwise; if we add negative value (-C), the angle of MAC moves clockwise.

Table 13.3 shows all combinations of MAC angle movement. And from Table 13.4 to table 13.7 shows the compaction process of combinations. I implemented saturating operation according to Table 13.7.



**Figure13.7 Way of thought for Saturating Accumulation**

| Initial MAC | C(Rotation) | MAC+C | ADDRESS  | RESULT | Notes      |
|-------------|-------------|-------|----------|--------|------------|
| P           | +           | P     | OK       |        |            |
| P           | +           | P'    | 00007FFF |        |            |
| P           | +           | M'    | 00007FFF |        |            |
| P           | +           | M     | 00007FFF |        |            |
| P'          | +           | P     | 00007FFF |        | Impossible |
| P'          | +           | P'    | 00007FFF |        |            |
| P'          | +           | M'    | 00007FFF |        |            |
| P'          | +           | M     | 00007FFF |        |            |
| M'          | +           | P     | OK       |        |            |
| M'          | +           | P'    | 00007FFF |        | Impossible |
| M'          | +           | M'    | FFFF8000 |        |            |
| M'          | +           | M     | OK       |        |            |
| M           | +           | P     | OK       |        |            |
| M           | +           | P'    | 00007FFF |        |            |
| M           | +           | M'    | 00007FFF |        | Impossible |
| M           | +           | M     | OK       |        |            |
| P           | -           | P     | OK       |        |            |
| P           | -           | P'    | FFFF8000 |        | Impossible |
| P           | -           | M'    | FFFF8000 |        |            |
| P           | -           | M     | OK       |        |            |
| P'          | -           | P     | OK       |        |            |
| P'          | -           | P'    | 00007FFF |        |            |
| P'          | -           | M'    | FFFF8000 |        | Impossible |
| P'          | -           | M     | OK       |        |            |
| M'          | -           | P     | FFFF8000 |        |            |
| M'          | -           | P'    | FFFF8000 |        |            |
| M'          | -           | M'    | FFFF8000 |        |            |
| M'          | -           | M     | FFFF8000 |        | Impossible |
| M           | -           | P     | FFFF8000 |        |            |
| M           | -           | P'    | FFFF8000 |        |            |
| M           | -           | M'    | FFFF8000 |        |            |
| M           | -           | M     | OK       |        |            |

**Table13.3 All combinations of angle movement (1)**

| Initial MAC | C(Rotation) | MAC+C  | ADDRRESULT        | Notes      |
|-------------|-------------|--------|-------------------|------------|
| P /M        | +/-         | P /M   | OK                |            |
| P /M        | +/-         | P' /M' | 00007FFF/FFFF8000 |            |
| P /M        | +/-         | M' /P' | 00007FFF/FFFF8000 |            |
| P /M        | +/-         | M /P   | 00007FFF/FFFF8000 |            |
| P' /M'      | +/-         | P /M   | Impossible        | Don't care |
| P' /M'      | +/-         | P' /M' | 00007FFF/FFFF8000 |            |
| P' /M'      | +/-         | M' /P' | 00007FFF/FFFF8000 |            |
| P' /M'      | +/-         | M /P   | 00007FFF/FFFF8000 |            |
| M' /P'      | +/-         | P /M   | OK                |            |
| M' /P'      | +/-         | P' /M' | Impossible        | Don't care |
| M' /P'      | +/-         | M' /P' | FFFF8000/00007FFF | Caution!   |
| M' /P'      | +/-         | M /P   | OK                |            |
| M /P        | +/-         | P /M   | OK                |            |
| M /P        | +/-         | P' /M' | 00007FFF/FFFF8000 |            |
| M /P        | +/-         | M' /P' | Impossible        | Don't care |
| M /P        | +/-         | M /P   | OK                |            |

**Table13.4 All combinations of angle movement (2)**

| Initial MAC | C(Rotation) | MAC+C  | ADDRRESULT        | Notes      |
|-------------|-------------|--------|-------------------|------------|
| P /M        | +/-         | P /M   | OK                |            |
| P /M        | +/-         | P' /M' | 00007FFF/FFFF8000 |            |
| P /M        | +/-         | - /+   | 00007FFF/FFFF8000 |            |
| P' /M'      | +/-         | P /M   | Impossible        | Don't care |
| P' /M'      | +/-         | P' /M' | 00007FFF/FFFF8000 |            |
| P' /M'      | +/-         | - /+   | 00007FFF/FFFF8000 |            |
| M' /P'      | +/-         | P /M   | OK                |            |
| - /+        | +/-         | M' /P' | FFFF8000/00007FFF | Caution!   |
| M' /P'      | +/-         | M /P   | OK                |            |
| M /P        | +/-         | P /M   | OK                |            |
| - /+        | +/-         | P' /M' | 00007FFF/FFFF8000 |            |
| M /P        | +/-         | M /P   | OK                |            |

**Table13.5 Compressed combinations of angle movement (1)**

| Initial MAC | C(Rotation) | MAC+C  | ADDRRESULT        | Notes    |
|-------------|-------------|--------|-------------------|----------|
| + /-        | +/-         | P /M   | OK                |          |
| + /-        | +/-         | P' /M' | 00007FFF/FFFF8000 |          |
| + /-        | +/-         | - /+   | 00007FFF/FFFF8000 |          |
| - /+        | +/-         | P /M   | OK                |          |
| - /+        | +/-         | M' /P' | FFFF8000/00007FFF | Caution! |
| - /+        | +/-         | M /P   | OK                |          |
| - /+        | +/-         | P' /M' | 00007FFF/FFFF8000 |          |

**Table13.6 Compressed combinations of angle movement (2)**

| Initial MAC | C(Rotation) | MAC+C  | ADDRRESULT        | Notes    |
|-------------|-------------|--------|-------------------|----------|
| * /*        | +/-         | P /M   | OK                |          |
| * /*        | +/-         | P' /M' | 00007FFF/FFFF8000 |          |
| + /-        | +/-         | - /+   | 00007FFF/FFFF8000 |          |
| - /+        | +/-         | M' /P' | FFFF8000/00007FFF | Caution! |
| - /+        | +/-         | M /P   | OK                |          |

**Table13.7 Compressed combinations of angle movement (3)**

# 14. Appendix: Aquarius Instruction Code

Aquarius instruction codes are compatible to SuperH-2.

Table14.1 shows all instruction codes and their controls.

| Class | Mnemonic            | Code Binary                     | Code Hex   | Step  | X  | Y  | Z       | ALU    | CMP     | SFT | Others                          |
|-------|---------------------|---------------------------------|------------|-------|----|----|---------|--------|---------|-----|---------------------------------|
| ALU   | STC SR, Rn          | 0 0 0 0 n n n n 0 0 0 0 0 1 0   | 0 ## 0 2   |       |    |    | SR      | Rn     | THRU    |     |                                 |
| ALU   | STC GBR, Rn         | 0 0 0 0 n n n n 0 0 0 1 0 0 1 0 | 0 ## 1 2   |       |    |    | GBR     | Rn     | THRU    |     |                                 |
| ALU   | STC VBR, Rn         | 0 0 0 0 n n n n 0 0 1 0 0 0 1 0 | 0 ## 2 2   |       |    |    | VBR     | Rn     | THRU    |     |                                 |
| BRA   | BSRF Rm             | 0 0 0 0 m m m m 0 0 0 0 0 0 1 1 | 0 ## 0 3   |       | Rm |    | PC      | PC     | ADD     |     | CurPC->PR                       |
| BRA   | BSRF Rm             | 0 0 0 0 m m m m 0 0 0 0 0 0 1 1 | 0 ## 0 3   |       |    |    |         |        |         |     | IFADSEL, IF_JP                  |
| BRA   | BRBF Rm             | 0 0 0 0 m m m m 0 0 1 0 0 0 1 1 | 0 ## 2 3   |       | Rm |    | PC      | PC     | ADD     |     |                                 |
| BRA   | BRBF Rm             | 0 0 0 0 m m m m 0 0 1 0 0 0 1 1 | 0 ## 2 3   |       |    |    |         |        |         |     | IFADSEL, IF_JP                  |
| Store | MOV.B Rm, @(R0, Rn) | 0 0 0 0 n n n n m m m m 0 1 0 0 | 0 ## ## 4  |       | Rn | Rm | MAAD    | ADDR0  |         |     | R0+Rn->MAAD, Rm->MADW, WR.B     |
| Store | MOV.W Rm, @(R0, Rn) | 0 0 0 0 n n n n m m m m 0 1 0 1 | 0 ## ## 5  |       | Rn | Rm | MAAD    | ADDR0  |         |     | R0+Rn->MAAD, Rm->MADW, WR.W     |
| Store | MOV.L Rm, @(R0, Rn) | 0 0 0 0 n n n n m m m m 0 1 1 0 | 0 ## ## 6  |       | Rn | Rm | MAAD    | ADDR0  |         |     | R0+Rn->MAAD, Rm->MADW, WR.L     |
| MULT  | MUL.L Rm, Rn        | 0 0 0 0 n n n n m m m m 0 1 1 1 | 0 ## ## 7  |       |    |    |         |        |         |     | Rn->M1, Rm->M2, MUL.L           |
| ALU   | CLR.T               | 0 0 0 0 0 0 0 0 0 0 1 0 0 0     | 0 0 0 8    |       |    |    |         |        |         |     | 0->T                            |
| ALU   | SET.T               | 0 0 0 0 0 0 0 0 0 1 1 0 0 0     | 0 0 1 8    |       |    |    |         |        |         |     | 1->T                            |
| ALU   | CLR.MAC             | 0 0 0 0 0 0 0 0 0 1 0 1 0 0     | 0 0 2 8    |       |    |    |         |        |         |     | 0-MACH/MACL                     |
| ALU   | NOP                 | 0 0 0 0 0 0 0 0 0 0 1 0 0 1     | 0 0 0 9    |       |    |    |         |        |         |     | NOP                             |
| ALU   | DIV0U               | 0 0 0 0 0 0 0 0 0 1 1 0 0 1     | 0 0 1 9    |       |    |    |         |        |         |     | DIV0U                           |
| ALU   | MOV.T Rn            | 0 0 0 0 n n n n 0 1 0 1 0 1 0   | 0 ## 2 9   |       |    |    |         | Rn     | ADD     |     | if T=1, 0+1->Rn else 0->Rn      |
| ALU   | STS MACH, Rn        | 0 0 0 0 n n n n 0 0 0 0 1 0 1 0 | 0 ## 0 10  |       |    |    | MACH    | Rn     | THRU    |     |                                 |
| ALU   | STS MACL, Rn        | 0 0 0 0 n n n n 0 0 1 0 1 0 1 0 | 0 ## 1 10  |       |    |    | MACL    | Rn     | THRU    |     |                                 |
| ALU   | STS PR, Rn          | 0 0 0 0 n n n n 0 1 0 1 0 1 0   | 0 ## 2 10  |       |    |    | PR      | Rn     | THRU    |     |                                 |
| BRA   | RTS                 | 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1   | 0 0 0 11   | 1st   |    |    |         |        |         |     |                                 |
| BRA   | RTS                 | 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1   | 0 0 0 11   | 2nd   |    |    |         |        |         |     | IFADSEL, IF_JP                  |
| SLEEP | SLEEP               | 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1   | 0 0 1 11   | multi |    |    |         |        |         |     | SLEEP sequence                  |
| RTE   | RTE                 | 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1   | 0 0 2 11   | multi |    |    |         |        |         |     | RTE sequence                    |
| Load  | MOV.B @(R0, Rm), Rn | 0 0 0 0 n n n n m m m m 1 1 0 0 | 0 ## ## 12 |       | Rm |    | MAAD    | ADDR0  |         |     | R0+Rm->MAAD, RD.B, MADR->Rn     |
| Load  | MOV.W @(R0, Rm), Rn | 0 0 0 0 n n n n m m m m 1 1 0 1 | 0 ## ## 13 |       | Rm |    | MAAD    | ADDR0  |         |     | R0+Rm->MAAD, RD.W, MADR->Rn     |
| Load  | MOV.L @(R0, Rm), Rn | 0 0 0 0 n n n n m m m m 1 1 1 0 | 0 ## ## 14 |       | Rm |    | MAAD    | ADDR0  |         |     | R0+Rm->MAAD, RD.L, MADR->Rn     |
| MULT  | MAC.L @Rm+, @Rn+    | 0 0 0 0 n n n n m m m m 1 1 1 1 | 0 ## ## 15 | 1st   | Rm | 4  | Rn      | ADD    |         |     | Rn->MAAD, RD.L, MADR->M1        |
| MULT  | MAC.L @Rm+, @Rn+    | 0 0 0 0 n n n n m m m m 1 1 1 1 | 0 ## ## 15 | 2nd   | Rm | 4  | Rm      | ADD    |         |     | Rm->MAAD, RD.L, MADR->M2, MAC.L |
| Store | MOV.L Rm, @Rn       | 0 0 0 1 n n n n m m m m d d d d | 1 ## ## ## |       | Rn | Rm | MAAD    | ADDCN  |         |     | 0d*4+Rn->MAAD, Rm->MADW, WR.L   |
| Store | MOV.B Rm, @Rn       | 0 0 1 0 n n n n m m m m 0 0 0 2 | ## ## 0    |       | Rn | Rm | MAAD    | THRU   |         |     | Rm->MADW, WR.B                  |
| Store | MOV.W Rm, @Rn       | 0 0 1 0 n n n n m m m m 0 0 1 2 | ## ## 1    |       | Rn | Rm | MAAD    | THRU   |         |     | Rm->MADW, WR.W                  |
| Store | MOV.L Rm, @Rn       | 0 0 1 0 n n n n m m m m 0 0 1 0 | ## ## 2    |       | Rn | Rm | MAAD    | THRU   |         |     | Rm->MADW, WR.L                  |
| Store | MOV.B Rm, @-Rn      | 0 0 1 0 n n n n m m m m 0 1 0 2 | ## ## 4    |       | Rn | -1 | Rn/MAAD | ADD    |         |     | Rm->MADW, WR.B                  |
| Store | MOV.W Rm, @-Rn      | 0 0 1 0 n n n n m m m m 0 1 0 1 | ## ## 5    |       | Rn | -2 | Rn/MAAD | ADD    |         |     | Rm->MADW, WR.W                  |
| Store | MOV.L Rm, @-Rn      | 0 0 1 0 n n n n m m m m 0 1 1 0 | ## ## 6    |       | Rn | -4 | Rn/MAAD | ADD    |         |     | Rm->MADW, WR.L                  |
| ALU   | DIV0S Rm, Rn        | 0 0 1 0 n n n n m m m m 0 1 1 1 | ## ## 7    |       |    |    |         |        |         |     | DIV0S                           |
| ALU   | TST Rm, Rn          | 0 0 1 0 n n n n m m m m 1 0 0 2 | ## ## 8    |       | Rn | Rm |         | AND    |         |     | result->T                       |
| ALU   | AND Rm, Rn          | 0 0 1 0 n n n n m m m m 1 0 0 1 | ## ## 9    |       | Rn | Rm | Rn      | AND    |         |     |                                 |
| ALU   | XOR Rm, Rn          | 0 0 1 0 n n n n m m m m 1 0 1 0 | ## ## 10   |       | Rn | Rm | Rn      | XOR    |         |     |                                 |
| ALU   | OR Rm, Rn           | 0 0 1 0 n n n n m m m m 1 0 1 1 | ## ## 11   |       | Rn | Rm | Rn      | OR     |         |     |                                 |
| ALU   | CMP/STR Rm, Rn      | 0 0 1 0 n n n n m m m m 1 1 0 0 | ## ## 12   |       | Rn | Rm |         |        | CMP/STR |     | result->T                       |
| ALU   | XTRCT Rm, Rn        | 0 0 1 0 n n n n m m m m 1 1 0 1 | ## ## 13   |       | Rn | Rm | Rn      | XTRCT  |         |     |                                 |
| MULT  | MULU.W Rm, Rn       | 0 0 1 0 n n n n m m m m 1 1 1 0 | ## ## 14   |       |    |    |         |        |         |     | Rn->M1, Rm->M2, MULU.W          |
| MULT  | MULS.W Rm, Rn       | 0 0 1 0 n n n n m m m m 1 1 1 1 | ## ## 15   |       |    |    |         |        |         |     | Rn->M1, Rm->M2, MULS.W          |
| ALU   | CMP/EQ Rm, Rn       | 0 0 1 1 n n n n m m m m 0 0 0 3 | ## ## 0    |       | Rn | Rm |         | CMP/EQ |         |     | result->T                       |
| ALU   | CMP/HS Rm, Rn       | 0 0 1 1 n n n n m m m m 0 0 1 3 | ## ## 2    |       | Rn | Rm |         | CMP/HS |         |     | result->T                       |
| ALU   | CMP/GE Rm, Rn       | 0 0 1 1 n n n n m m m m 0 0 1 1 | ## ## 3    |       | Rn | Rm |         | CMP/GE |         |     | result->T                       |
| ALU   | DIV1 Rm, Rn         | 0 0 1 1 n n n n m m m m 0 1 0 3 | ## ## 4    |       |    |    |         |        |         |     | DIV1                            |
| MULT  | DMULU.L Rm, Rn      | 0 0 1 1 n n n n m m m m 0 1 0 1 | ## ## 5    |       |    |    |         |        |         |     | Rn->M1, Rm->M2, DMULU.L         |
| ALU   | CMP/HI Rm, Rn       | 0 0 1 1 n n n n m m m m 0 1 1 0 | ## ## 6    |       | Rn | Rm |         | CMP/HI |         |     | result->T                       |
| ALU   | CMP/GT Rm, Rn       | 0 0 1 1 n n n n m m m m 0 1 1 1 | ## ## 7    |       | Rn | Rm |         | CMP/GT |         |     | result->T                       |
| ALU   | SUB Rm, Rn          | 0 0 1 1 n n n n m m m m 1 0 0 3 | ## ## 8    |       | Rn | Rm | Rn      | SUB    |         |     |                                 |
| ALU   | SUBC Rm, Rn         | 0 0 1 1 n n n n m m m m 1 0 1 0 | ## ## 10   |       | Rn | Rm | Rn      | SUBC   |         |     |                                 |
| ALU   | SUBV Rm, Rn         | 0 0 1 1 n n n n m m m m 1 0 1 1 | ## ## 11   |       | Rn | Rm | Rn      | SUBV   |         |     |                                 |
| ALU   | ADD Rm, Rn          | 0 0 1 1 n n n n m m m m 1 1 0 0 | ## ## 12   |       | Rn | Rm | Rn      | ADD    |         |     |                                 |
| MULT  | DMULS.L Rm, Rn      | 0 0 1 1 n n n n m m m m 1 1 0 1 | ## ## 13   |       |    |    |         |        |         |     | Rn->M1, Rm->M2, DMULS.L         |
| ALU   | ADDC Rm, Rn         | 0 0 1 1 n n n n m m m m 1 1 1 0 | ## ## 14   |       | Rn | Rm | Rn      | ADDC   |         |     |                                 |
| ALU   | ADDD Rm, Rn         | 0 0 1 1 n n n n m m m m 1 1 1 1 | ## ## 15   |       | Rn | Rm | Rn      | ADDD   |         |     |                                 |

Table14.1 Aquarius Instruction Codes (1)



| Class   | Mnemonic            | Code Binary        | Code Hex | Step  | X    | Y   | Z       | ALU   | CMP    | SFT    | Others                       |
|---------|---------------------|--------------------|----------|-------|------|-----|---------|-------|--------|--------|------------------------------|
| ALU     | SHLL Rn             | 0100nnnn00000000   | 4##00    |       | Rn   |     | Rn      |       |        | SHLL   |                              |
| ALU     | DT Rn               | 0100nnnn00010000   | 4##10    |       | Rn   | -1  | Rn      | ADD   |        |        | result->T                    |
| ALU     | SHAL Rn             | 0100nnnn00100000   | 4##20    |       | Rn   |     | Rn      |       |        | SHAL   |                              |
| ALU     | SHLR Rn             | 0100nnnn00000014   | 4##01    |       | Rn   |     | Rn      |       |        | SHLR   |                              |
| ALU     | CMP/PZ Rn           | 0100nnnn00010001   | 4##11    |       | Rn   |     |         |       | CMP/PZ |        | result->T                    |
| ALU     | SHAR Rn             | 0100nnnn00100014   | 4##21    |       | Rn   |     | Rn      |       |        | SHAR   |                              |
| Store   | STS.L MACH,@-Rn     | 0100nnnn00000104   | 4##02    |       | Rn   | -4  | Rn/MAAD | ADD   |        |        | MACH->MADW,WR.L              |
| Store   | STS.L MACL,@-Rn     | 0100nnnn00010104   | 4##12    |       | Rn   | -4  | Rn/MAAD | ADD   |        |        | MACL->MADW,WR.L              |
| Store   | STS.L PR,@-Rn       | 0100nnnn00100104   | 4##22    |       | Rn   | -4  | Rn/MAAD | ADD   |        |        | PR->MADW,WR.L                |
| STC     | STC.L SR,@-Rn       | 0100nnnn00000114   | 4##03    | multi | Rn   | -4  | Rn/MAAD | ADD   |        |        | SR->MADW,WR.L                |
| STC     | STC.L GBR,@-Rn      | 0100nnnn00010114   | 4##13    | multi | Rn   | -4  | Rn/MAAD | ADD   |        |        | GBR->MADW,WR.L               |
| STC     | STC.L VBR,@-Rn      | 0100nnnn00100114   | 4##23    | multi | Rn   | -4  | Rn/MAAD | ADD   |        |        | VBR->MADW,WR.L               |
| ALU     | ROTL Rn             | 0100nnnn00001004   | 4##04    |       | Rn   |     | Rn      |       |        | ROTL   |                              |
| ALU     | ROTCL Rn            | 0100nnnn00101004   | 4##24    |       | Rn   |     | Rn      |       |        | ROTCL  |                              |
| ALU     | ROTR Rn             | 0100nnnn00001014   | 4##05    |       | Rn   |     | Rn      |       |        | ROTR   |                              |
| ALU     | CMP/PL Rn           | 0100nnnn00010101   | 4##15    |       | Rn   |     |         |       | CMP/PL |        | result->T                    |
| ALU     | ROTCR Rn            | 0100nnnn00101014   | 4##25    |       | Rn   |     | Rn      |       |        | ROTCR  |                              |
| Load    | LDS.L @Rm+,MACH     | 0100mmmm00001104   | 4##06    |       | Rm   | 4   | Rm      | ADD   |        |        | Rm->MAAD,RD.L,MADR->MACH     |
| Load    | LDS.L @Rm+,MACL     | 0100mmmm00011104   | 4##16    |       | Rm   | 4   | Rm      | ADD   |        |        | Rm->MAAD,RD.L,MADR->MACL     |
| Load    | LDS.L @Rm+,PR       | 0100mmmm00101104   | 4##26    |       | Rm   | 4   | Rm      | ADD   |        |        | Rm->MAAD,RD.L,MADR->PR       |
| LDC(IM) | LDC.L @Rm+,SR       | 0100mmmm00001114   | 4##07    | multi | Rm   | 4   | Rm      | ADD   |        |        | Rm->MAAD,RD.L,MADR->SR       |
| LDC     | LDC.L @Rm+,GBR      | 0100mmmm00011114   | 4##17    | multi | Rm   | 4   | Rm      | ADD   |        |        | Rm->MAAD,RD.L,MADR->GBR      |
| LDC     | LDC.L @Rm+,VBR      | 0100mmmm00101114   | 4##27    | multi | Rm   | 4   | Rm      | ADD   |        |        | Rm->MAAD,RD.L,MADR->VBR      |
| ALU     | SHLL2 Rn            | 0100nnnn000010004  | 4##08    |       | Rn   |     | Rn      |       |        | SHLL2  |                              |
| ALU     | SHLL8 Rn            | 0100nnnn000110004  | 4##18    |       | Rn   |     | Rn      |       |        | SHLL8  |                              |
| ALU     | SHLL16 Rn           | 0100nnnn001010004  | 4##28    |       | Rn   |     | Rn      |       |        | SHLL16 |                              |
| ALU     | SHLR2 Rn            | 0100nnnn000010014  | 4##09    |       | Rn   |     | Rn      |       |        | SHLR2  |                              |
| ALU     | SHLR8 Rn            | 0100nnnn000110014  | 4##19    |       | Rn   |     | Rn      |       |        | SHLR8  |                              |
| ALU     | SHLR16 Rn           | 0100nnnn001010014  | 4##29    |       | Rn   |     | Rn      |       |        | SHLR16 |                              |
| ALU     | LDS Rm,MACH         | 0100mmmm000010104  | 4##010   |       | Rm   |     |         |       |        |        | Rm->MACH                     |
| ALU     | LDS Rm,MACL         | 0100mmmm000110104  | 4##110   |       | Rm   |     |         |       |        |        | Rm->MACL                     |
| ALU     | LDS Rm,PR           | 0100mmmm001010104  | 4##210   |       | Rm   |     | PR      | THRUX |        |        |                              |
| BRA     | JSR @Rm             | 0100mmmm000010114  | 4##011   | 1st   | Rm   |     | PC      | THRUX |        |        | CurPC->PR                    |
| BRA     | JSR @Rm             | 0100mmmm000010114  | 4##011   | 2nd   | Rm   |     |         |       |        |        | IFADSEL,IF_JP                |
| RMW     | TAS.B @Rn           | 0100nnnn000110114  | 4##111   | 1st   | Rn   |     | MAAD    | THRUX |        |        | RD.B                         |
| RMW     | TAS.B @Rn           | 0100nnnn000110114  | 4##111   | 2nd   | MADR | 080 | MADW    | AND   | CMP/Z  |        | WR.B,result->T               |
| BRA     | JMP @Rm             | 0100mmmm001010114  | 4##211   | 1st   | Rm   |     | PC      | THRUX |        |        |                              |
| BRA     | JMP @Rm             | 0100mmmm001010114  | 4##211   | 2nd   | Rm   |     |         |       |        |        | IFADSEL,IF_JP                |
| ALU(IM) | LDC Rm,SR           | 0100mmmm000011104  | 4##014   |       | Rm   |     | SR      | THRUX |        |        |                              |
| ALU(IM) | LDC Rm,GBR          | 0100mmmm000111104  | 4##114   |       | Rm   |     | GBR     | THRUX |        |        |                              |
| ALU(IM) | LDC Rm,VBR          | 0100mmmm001011104  | 4##214   |       | Rm   |     | VBR     | THRUX |        |        |                              |
| MULT    | MAC.W @Rm+,@Rn+     | 0100nnnnmmmm11114  | 4####15  | 1st   | Rm   | 4   | Rn      | ADD   |        |        | Rn->MAAD,RD.W,MADR->M1       |
| MULT    | MAC.W @Rm+,@Rn+     | 0100nnnnmmmm11114  | 4####15  | 2nd   | Rm   | 4   | Rn      | ADD   |        |        | Rm->MAAD,RD.W,MADR->M2,MAC.W |
| Load    | MOV.L @(disp,Rm),Rn | 0101nnnnmmmmddddd5 | 6####    |       | Rm   |     | MAAD    | ADDCN |        |        | Od*4+Rm->MAAD, RD.L,MADR->Rn |
| Load    | MOV.B @Rm, Rn       | 0110nnnnmmmm00006  | 6####0   |       | Rm   |     | Rm      | MAAD  |        | THRUY  | RD.B,MADR->Rn                |
| Load    | MOV.W @Rm, Rn       | 0110nnnnmmmm00016  | 6####1   |       | Rm   |     | Rm      | MAAD  |        | THRUY  | RD.W,MADR->Rn                |
| Load    | MOV.L @Rm, Rn       | 0110nnnnmmmm00106  | 6####2   |       | Rm   |     | Rm      | MAAD  |        | THRUY  | RD.L,MADR->Rn                |
| ALU     | MOV Rm,Rn           | 0110nnnnmmmm00116  | 6####3   |       | Rm   |     | Rn      |       |        | THRUY  |                              |
| Load    | MOV.B @Rm+ ,Rn      | 0110nnnnmmmm01006  | 6####4   |       | Rm   | 1   | Rm      | ADD   |        |        | Rm->MAAD,RD.B,MADR->Rn       |
| Load    | MOV.W @Rm+ ,Rn      | 0110nnnnmmmm01016  | 6####5   |       | Rm   | 2   | Rm      | ADD   |        |        | Rm->MAAD,RD.W,MADR->Rn       |
| Load    | MOV.L @Rm+ ,Rn      | 0110nnnnmmmm01106  | 6####6   |       | Rm   | 4   | Rm      | ADD   |        |        | Rm->MAAD,RD.L,MADR->Rn       |
| ALU     | NOT Rm,Rn           | 0110nnnnmmmm01116  | 6####7   |       | Rm   |     | Rn      |       |        | NOT    |                              |
| ALU     | SWAP.B Rm,Rn        | 0110nnnnmmmm10006  | 6####8   |       | Rn   |     | Rm      |       |        | SWAPB  |                              |
| ALU     | SWAP.W Rm,Rn        | 0110nnnnmmmm10016  | 6####9   |       | Rn   |     | Rm      |       |        | SWAPW  |                              |
| ALU     | NEGC Rm,Rn          | 0110nnnnmmmm10106  | 6####10  |       | Rm   |     | Rn      |       |        | NEGC   |                              |
| ALU     | NEG Rm,Rn           | 0110nnnnmmmm10116  | 6####11  |       | Rm   |     | Rn      |       |        | NEG    |                              |
| ALU     | EXTU.B Rm,Rn        | 0110nnnnmmmm11006  | 6####12  |       | Rm   |     | Rn      |       |        | EXTUB  |                              |
| ALU     | EXTU.W Rm,Rn        | 0110nnnnmmmm11016  | 6####13  |       | Rm   |     | Rn      |       |        | EXTUW  |                              |
| ALU     | EXTS.B Rm,Rn        | 0110nnnnmmmm11106  | 6####14  |       | Rm   |     | Rn      |       |        | EXTSB  |                              |
| ALU     | EXTS.W Rm,Rn        | 0110nnnnmmmm11116  | 6####15  |       | Rm   |     | Rn      |       |        | EXTSW  |                              |

Table14.1 Aquarius Instruction Codes (2)

| Class | Mnemonic             | Code Binary      | Code Hex   | Step  | X     | Y    | Z    | ALU   | CMP    | SFT | Others                        |
|-------|----------------------|------------------|------------|-------|-------|------|------|-------|--------|-----|-------------------------------|
| ALU   | ADD #imm,Rn          | 0111nnnniiiiiiii | 7## ## ##  |       | Rn    | si   | Rn   | ADD   |        |     |                               |
| Store | MOV.B R0,@(disp,Rn)  | 10000000nnnndddd | 80 ## ##   |       | Rn    | R0   | MAAD | ADDCN |        |     | 0d*1+Rn->MAAD, R0->MADW,WR.B  |
| Store | MOV.W R0,@(disp,Rn)  | 10000001nnnndddd | 81 ## ##   |       | Rn    | R0   | MAAD | ADDCN |        |     | 0d*2+Rn->MAAD, R0->MADW,WR.W  |
| Load  | MOV.B @(disp,Rm),R0  | 10000100mmmmdddd | 84 ## ##   |       | Rm    |      | MAAD | ADDCN |        |     | 0d*1+Rm->MAAD, RD.B,MADR->R0  |
| Load  | MOV.W @(disp,Rm),R0  | 10000101mmmmdddd | 85 ## ##   |       | Rm    |      | MAAD | ADDCN |        |     | 0d*2+Rm->MAAD, RD.W,MADR->R0  |
| ALU   | CMP/EQ #imm R0       | 10001000iiiiiiii | 88 ## ##   |       | R0    | si   |      |       | CMP/EQ |     | result->T                     |
| Bcc   | BT disp              | 10001001dddddddd | 89 ## ##   | 1st   | PC    | sd*2 | PC   | ADD   |        |     | if ~T then NOP and DISPATCH   |
| Bcc   | BT disp              | 10001001dddddddd | 89 ## ##   | 2nd   |       |      |      |       |        |     | IFADSEL,IF_JP                 |
| Bcc   | BT disp              | 10001001dddddddd | 89 ## ##   | 3rd   |       |      |      |       |        |     | NOP operation                 |
| Bcc   | BF disp              | 10001011dddddddd | 811 ## ##  | 1st   | PC    | sd*2 | PC   | ADD   |        |     | if T then NOP and DISPATCH    |
| Bcc   | BF disp              | 10001011dddddddd | 811 ## ##  | 2nd   |       |      |      |       |        |     | IFADSEL,IF_JP                 |
| Bcc   | BF disp              | 10001011dddddddd | 811 ## ##  | 3rd   |       |      |      |       |        |     | NOP operation                 |
| Bcc/S | BT/S disp            | 10001101dddddddd | 813 ## ##  |       | PC    | sd*2 | PC   | ADD   |        |     | if result                     |
| Bcc/S | BF/S disp            | 10001111dddddddd | 815 ## ##  |       | PC    | sd*2 | PC   | ADD   |        |     | if result                     |
| Load  | MOV.W @(disp,PC),Rn  | 1001nnnndddddddd | 9 ## ##    |       | PC    | 0d*2 | MAAD | ADD   |        |     | RD.L,MADR->Rn                 |
| BR    | BRA disp             | 1010dddddddddddd | 10 ## ##   | 1st   | PC    | sd*2 | PC   | ADD   |        |     |                               |
| BR    | BRA disp             | 1010dddddddddddd | 10 ## ##   | 2nd   |       |      |      |       |        |     | IFADSEL,IF_JP                 |
| BR    | BSR disp             | 1011dddddddddddd | 11 ## ##   | 1st   | PC    | sd*2 | PC   | ADD   |        |     | CurPC->PR                     |
| BR    | BSR disp             | 1011dddddddddddd | 11 ## ##   | 2nd   |       |      |      |       |        |     | IFADSEL,IF_JP                 |
| Store | MOV.B R0,@(disp,GBR) | 11000000dddddddd | 120 ## ##  |       | GBR   | R0   | MAAD | ADDCN |        |     | 0d*1+GRB->MAAD, R0->MADW,WR.B |
| Store | MOV.W R0,@(disp,GBR) | 11000001dddddddd | 121 ## ##  |       | GBR   | R0   | MAAD | ADDCN |        |     | 0d*2+GRB->MAAD, R0->MADW,WR.W |
| Store | MOV.L R0,@(disp,GBR) | 11000010dddddddd | 122 ## ##  |       | GBR   | R0   | MAAD | ADDCN |        |     | 0d*4+GRB->MAAD, R0->MADW,WR.L |
| TRAP  | TRAPA #imm           | 11000011iiiiiiii | 123 ## ##  | multi | PC    | 0i*4 | PC   | ADD   |        |     | TRAPA sequence                |
| Load  | MOV.B @(disp,GBR),R1 | 11000100dddddddd | 124 ## ##  |       | GBR   |      | MAAD | ADDCN |        |     | 0d*1+GRB->MAAD, RD.B,MADR->R0 |
| Load  | MOV.W @(disp,GBR),R1 | 11000101dddddddd | 125 ## ##  |       | GBR   |      | MAAD | ADDCN |        |     | 0d*2+GRB->MAAD, RD.W,MADR->R0 |
| Load  | MOV.L @(disp,GBR),R1 | 11000110dddddddd | 126 ## ##  |       | GBR   |      | MAAD | ADDCN |        |     | 0d*4+GRB->MAAD, RD.L,MADR->R0 |
| ALU   | MOVA @(disp,PC),R0   | 11000111dddddddd | 127 ## ##  |       | PC&FC | 0d*4 | R0   | ADD   |        |     |                               |
| ALU   | TST #imm,R0          | 11001000iiiiiiii | 128 ## ##  |       | R0    | 0i   |      | AND   |        |     | result->T                     |
| ALU   | AND #imm,R0          | 11001001iiiiiiii | 129 ## ##  |       | R0    | 0i   | R0   | AND   |        |     |                               |
| ALU   | XOR #imm,R0          | 11001010iiiiiiii | 1210 ## ## |       | R0    | 0i   | R0   | XOR   |        |     |                               |
| ALU   | OR #imm,R0           | 11001011iiiiiiii | 1211 ## ## |       | R0    | 0i   | R0   | OR    |        |     |                               |
| RMW   | TST.B #imm,@(R0,GBR) | 11001100iiiiiiii | 1212 ## ## | 1st   | GBR   | R0   | MAAD | ADD   |        |     | RD.B                          |
|       |                      |                  |            | 2nd   | MADR  | 0i   |      | AND   |        |     | result->T                     |
| RMW   | AND.B #imm,@(R0,GBR) | 11001101iiiiiiii | 1213 ## ## | 1st   | GBR   | R0   | MAAD | ADD   |        |     | RD.B                          |
|       |                      |                  |            | 2nd   | MADR  | 0i   | MADW | AND   |        |     | WR.B                          |
| RMW   | XOR.B #imm,@(R0,GBR) | 11001110iiiiiiii | 1214 ## ## | 1st   | GBR   | R0   | MAAD | ADD   |        |     | RD.B                          |
|       |                      |                  |            | 2nd   | MADR  | 0i   | MADW | XOR   |        |     | WR.B                          |
| RMW   | OR.B #imm,@(R0,GBR)  | 11001111iiiiiiii | 1215 ## ## | 1st   | GBR   | R0   | MAAD | ADD   |        |     | RD.B                          |
|       |                      |                  |            | 2nd   | MADR  | 0i   | MADW | OR    |        |     | WR.B                          |
| Load  | MOV.L @(disp,PC),Rn  | 1101nnnndddddddd | 13 ## ##   |       | PC&FC | 0d*4 | MAAD | ADD   |        |     | RD.L,MADR->Rn                 |
| ALU   | MOV #imm,Rn          | 1110nnnniiiiiiii | 14 ## ##   |       |       | si   | Rn   | THRU  |        |     |                               |
| EVENT | Illegal Instruction  | 11111111*****    | 1515 ## ## | multi |       |      |      |       |        |     |                               |
| EVENT | Slot Illegal         | 111111100000110  | 1514 0 6   | multi |       |      |      |       |        |     |                               |
| EVENT | IRQ                  | 11110010*****    | 152 ## ##  | multi |       |      |      |       |        |     |                               |
| EVENT | NMI                  | 1111001100001011 | 153 0 11   | multi |       |      |      |       |        |     |                               |
| EVENT | Address Error (CPU)  | 1111010000001001 | 154 0 9    | multi |       |      |      |       |        |     |                               |
| EVENT | Address Error (DMAC) | 1111010100001010 | 155 0 10   | multi |       |      |      |       |        |     |                               |
| EVENT | Manual Reset         | 111101100000010  | 156 0 2    | multi |       |      |      |       |        |     |                               |
| EVENT | Power on Reset       | 11110111*****    | 157 ## ##  | multi |       |      |      |       |        |     |                               |

Table14.1 Aquarius Instruction Codes (3)