The Antares processor

Preliminary Draft
31 October 2018

Ángel Terrones
(Universidad Simón Bolívar)
(angelterrones@gmail.com)

This file documents the Antares processor.

Copyright © 2015 Ángel Terrones

Permission is granted to copy, distribute and/or modify this document under the terms of the MIT License. A copy of the license is included in the section entitled “MIT License”.

Table of Contents

1 Introduction ... 1

1.1 Getting Started ... 1
1.1.1 Supported Host Platforms ... 1
1.1.2 Obtaining the project source files ... 1
1.1.3 Documentation ... 1
1.2 Required Tools ... 1
1.2.1 Reference Design ... 2
1.2.1.1 Host System Tools ... 2
1.2.1.2 Target System Tools ... 2
1.2.1.3 EDA Tools ... 2
1.2.1.4 Debug Tools ... 2
1.3 Features & Limitations ... 2
1.4 Legal information ... 3

2 Architecture ... 4

2.1 Introduction ... 4
2.2 Features ... 4
2.3 CPU ... 4
2.3.1 Configuration ... 5
2.3.2 Instruction Decode Unit ... 6
2.3.3 Branch Unit ... 6
2.3.4 General-Purpose Registers (GPR) ... 6
2.3.5 Integer Execution Unit (ALU) ... 6
2.3.6 Load/Store Unit (LSU) ... 6
2.3.7 Hazard and Forwarding Unit ... 7
2.3.8 Exception Unit: Coprocessor 0 ... 7
2.3.9 Memory Map ... 8

3 Core Operation ... 9

3.1 Reset ... 9
3.2 Core ... 9
3.2.1 Instruction Set ... 9
3.2.2 Branch Unit ... 12
3.2.3 General-Purpose Registers (GPR) ... 12
3.2.4 Integer Execution Unit (ALU) ... 12
3.2.5 Load/Store Unit (LSU) ... 13
3.2.6 Hazard and Forwarding Unit ... 14
3.2.7 Exception Unit: Coprocessor 0 ... 14

4 Core Registers 15
- 4.1 General-Purpose Registers (GPR) 15
- 4.2 Coprocessor 0 Registers 15
  - 4.2.1 Status Register (SR) 16
  - 4.2.2 Cause Register 18
  - 4.2.3 Processor Identification Register (PRId) 18
  - 4.2.4 Configuration Register (Config) 18
  - 4.2.5 Configuration Register 1 (Config1) 19
  - 4.2.6 Bad Virtual Address Register (BadVAddr) 20
  - 4.2.7 Count/Compare Register 20
  - 4.2.8 Exception Program Counter 20
  - 4.2.9 Error Register 20

5 IO Ports 21
- 5.1 Instruction Interface 21
- 5.2 Data Interface 22
- 5.3 Interrupts Interface 22
- 5.4 System Interface 22

6 Core Configuration 23

7 Software Development Tools 24
- 7.1 MIPS Cross-compiler Toolchain (GCC) 24
  - 7.1.1 Instructions for Creating a MIPS Compiler Toolchain 24
    - 7.1.1.1 Required Files 24
    - 7.1.1.2 Environment Requirements 24
    - 7.1.1.3 Building the Toolchain 24
- 7.2 Simulation 25
  - 7.2.1 Run the simulation. 25

8 MIT License 28

Index 29

1 Introduction

The Antares is a synthesizable 32-bits soft processor core written in Verilog. It is compatible with the MIPS32 Release 1 processors and can execute code generated by a MIPS32 toolchain, with some restrictions.

The core comes includes a hardware multiplier and divider, and a vendor-independent BRAM module for simulation and synthesis.

1.1 Getting Started

This section discuss how to get the project files, and the documentation of the processor.

1.1.1 Supported Host Platforms

The majority of Antares's development occurs with tools that runs under the GNU/Linux operating system. All of the tools required to run the basic implementation are free, open source, and easily installable in any modern GNU/Linux distribution.

Support for the project under Cygwin on Microsoft Windows platforms cannot be assumed.

1.1.2 Obtaining the project source files

The complete project can be downloaded from the GitHub repository: https://github.com/AngelTerrones/Antares/archive/master.zip.

Or just clone the repository:

```
git clone https://github.com/AngelTerrones/Antares.git
```

1.1.3 Documentation

Being compatible with the MIPS32 Release 1 processor, the official documentation is applicable: http://www.imgtec.com/mips/architectures/mips32.asp. Note that the official documentation has been updated to include the new Release 6.

The Antares documentation (this file) can be find inside the folder `Documentation`.

1.2 Required Tools

The required tools can be divided into four groups:

- Host system tools: things like gcc, make, texinfo (for documentation), and others.
- Target system toolchain and software: the MIPS toolchain, with gcc cross-compiler, support libraries, the GNU debugger (gdb), MIPS port of various OSes and RTOS, etc.
- Electronic design automation (EDA) tools: preprocessors, simulators, FPGA tool suites, etc.
- Debug tools: tools providing control over the system on target

The first two items are likely to be the same for most of the designs using the Antares core. However, the final two can vary greatly depending on the FPGA vendor, part configuration and the application of the SoC.

1.2.1 Reference Design

The reference design is the minimal implementation required to test the Antares core: processor and memory.

To design, simulate, verify, compile and debug the reference design, the following tools are required:

1.2.1.1 Host System Tools

Standard suit of development tools: gcc, make.

1.2.1.2 Target System Tools

Suggested: Mentor Graphics Sourcery CodeBench Lite for MIPS ELF. Or build your own toolchain (See Section 7.1.1 [Toolchain Instructions], page 24).

1.2.1.3 EDA Tools

For RTL Simulation: Icarus Verilog.

1.2.1.4 Debug Tools

None. The target is pure simulation.

1.3 Features & Limitations

The following lists the main features of Antares IP core:

- Core:
  - Single-issue in-order 6-stage pipeline with full forwarding and hazard detection.
  - Harvard architecture, with separate instruction and data ports.
  - A subset of the MIPS32 instruction set. Includes: hardware multiplication, hardware division, MAC/MAS, load linked & store conditional.
  - Multi-cycle Hardware divider (Disabled by default).
  - Hardware multiplier (5-stages pipeline, disabled by default).
  - Hardware is Little-Endian.
  - Coprocessor 0 allows ISA-compliant interrupts, exceptions, and user/kernel modes.
  - Documentation in-source.
  - Vendor-independent code.

The following lists the main limitations of Antares IP core:

- Core:
  - No MMU.
  - No Cache.
  - No FPU. Only software-base floating point support (toolchain). Untested.
  - No support for reverse-endian mode.
  - No address space verification for the instruction port: Instruction address is always a kernel address.

1.4 Legal information

MIPS32 is a trademark of Imagination Technologies. This project is not affiliated in any way with Imagination Technologies. All other product names are trademarks or registered trademarks of their respective owners.

2 Architecture

2.1 Introduction

The Antares is a configurable 32-bit soft processor core with Harvard microarchitecture, 6-stage pipeline, compatible with the MIPS32 Release 1 processor.

This chapter introduce the Antares architecture, and describes the general architecture features.

2.2 Features

The following lists the main features of Antares IP core:

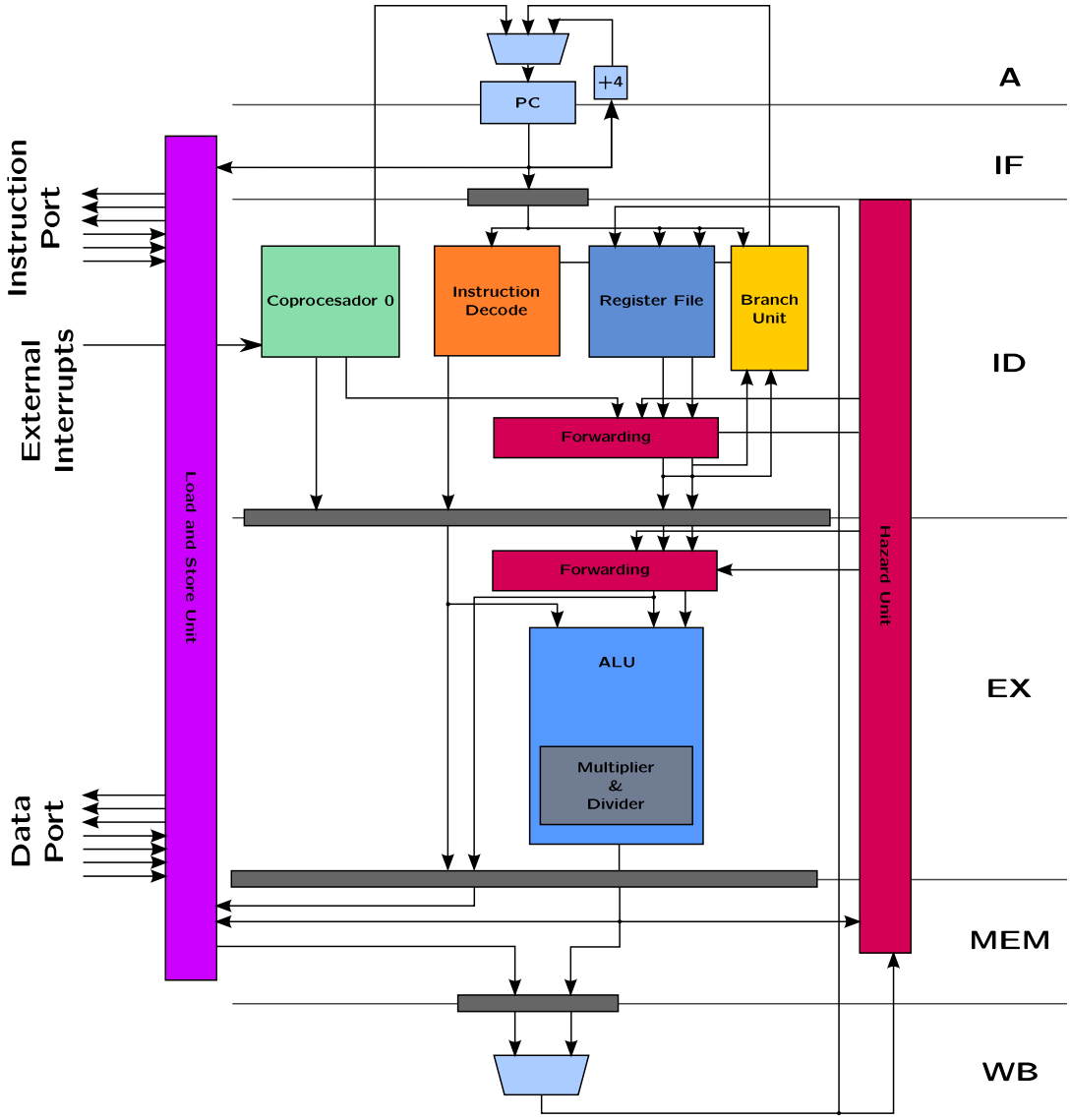
- Single-issue in-order 6-stage pipeline with full forwarding and hazard detection.
- Harvard architecture, with separate instruction and data ports.
- A subset of the MIPS32 instruction set. Includes: hardware multiplication, hardware division, MAC/MAS, load linked / store conditional.
- Multi-cycle Hardware divider (Disabled by default).
- Hardware multiplier (5-stages pipeline, disabled by default).
- Hardware is Little-Endian.
- Coprocessor 0 allows ISA-compliant interrupts, exceptions, and user/kernel modes.
- Documentation in-source.
- Vendor-independent code.

2.3 CPU

This section describes the Antares core design structure. It consist of several building blocks:

- Instruction Decode Unit.
- Branch Unit.
- General-Purpose Registers (GPR).
- Integer Execution Unit (ALU).
- Load/Store Unit (LSU).
- Hazard and Forwarding Unit.
- Exception Unit (Coprocessor 0).

As shown in the following figure, the Antares processor uses a 6-stage pipeline which is fully bypassed and interlocked. The bypass logic is responsible for forwarding results back through the pipeline, allowing most instructions to be effectively executed in a single cycle. The interlock is responsible for detecting read-after-write (RAW) hazards and stalling the pipeline until the hazard has been resolved, avoiding the need to insert `NOP` instructions between dependent instructions.



The six pipeline stages are:

- Address (A): The address of the instruction to execute is calculated.
- Fetch (IF): The Instruction is read from memory.
- Decode (ID): The instruction is decoded and operands are either fetched from the register file, or bypassed from the pipeline. PC-relative branches are calculated here. Access to the coprocessor 0 is performed.
- Execute (EX): The integer operation specified by the instruction is performed.
- Memory (MEM): Memory access (load/store).
- Write-back (WB): Result produced are written back to the register file.

2.3.1 Configuration

It is possible to configure the Antares core using the following parameters (see Hardware/verilog/antares_core.v):

`ENABLE_HW_MULT`
: Enable hardware support for multiplication instructions.

`ENABLE_HW_DIV`
: Enable hardware support for division instructions.

`ENABLE_HW_CLO_Z`
: Enable hardware support for the CLO and CLZ instructions.

2.3.2 Instruction Decode Unit

This unit performs the instruction decode and generates all the processor control signals but those related with the pipeline control.

This unit is implemented in Hardware/verilog/antares/antares_control_unit.v.

2.3.3 Branch Unit

This unit detects branch and jump instructions, and check if the branch should be taken.

This unit is implemented in Hardware/verilog/antares_branch_unit.v

2.3.4 General-Purpose Registers (GPR)

Like the MIPS specification, the Antares core implements 32 general-purpose 32-bit registers.

The Antares implements the general-purpose registers as one asynchronous dual-port memory.

This unit is implemented in Hardware/verilog/antares_reg_file.v

2.3.5 Integer Execution Unit (ALU)

The core implements the following types of 32-bits integer instructions:

- Arithmetic instructions.
- Logical instructions.
- Compare instructions.
- Shift instructions.

Most integer instructions can execute in one cycle. The exceptions are the multiplication and division instructions, if they are enabled. This unit is implemented in /Hardware/verilog/antares_alu.v.

The hardware multiplier is implemented as a 5-stage pipeline. This unit is implemented in Hardware/verilog/antares_multiplier.v.

The hardware divider is implemented as a FSM. This unit is implemented in Hardware/verilog/antares_div.v.

2.3.6 Load/Store Unit (LSU)

The load/store unit (LSU) transfers data between the core pipeline and the CPU internal bus. This unit will stall the master pipeline in case of data dependency.

The main features:

- Load and store implemented in hardware, atomic instructions included.
- Aligned access for fast memory access.

This unit requires the following operands:

- Address.
- Source data (for store instructions).
- Destination data (for load instructions).

This unit is implemented in Hardware/verilog/antares_load_store_unit.v

2.3.7 Hazard and Forwarding Unit

The hazard and forwarding unit implements the logic that is responsible for forwarding results back through the pipeline, allowing most instructions to be effectively executed in a single cycle. Also, this unit checks for data dependency (hazards) and interlocks the pipeline pipeline until the hazard has been resolved, avoiding the need to insert `NOP` instructions between dependent instructions.

This unit is implemented in Hardware/verilog/antares_hazard_unit.v

2.3.8 Exception Unit: Coprocessor 0

This module allows interrupts, traps, system calls and other exceptions. This unit implements only a subset of the MIPS coprocessor 0.

The supported exceptions sources:

`EXC_AdEL` Address error on load. This is either and attempt to get outside of `kuseg` when in user mode, or an attempt to read a word or halfword at a misaligned address.

`EXC_AdES` Address error on store. This is either and attempt to get outside of `kuseg` when in user mode, or an attempt to write a word or halfword at a misaligned address.

`EXC_DBE` Data bus error.

`EXC_Tr` Condition met on one of the conditional trap instructions.

`EXC_Ov` Overflow from arithmetic instructions.

`EXC_Sys` Executed a `syscall` instruction.

`EXC_Bp` Executed a `break` instruction. Used by debuggers.

`EXC_RI` Instruction code not recognized, or not legal.

`EXC_CpU` Tried to run a coprocessor instruction, but the appropriate coprocessor is not enabled in `SR(CU3-0)`.

`EXC_AdIF` Address error on instruction fetch. This is either and attempt to get outside of `kuseg` when in user mode, or an attempt to read a word or halfword at a misaligned

`EXC_IBE` Instruction bus error.

`EXC_Int` External interrupt.

The CPU Control Registers implemented in this core:

`Status (SR)`
: This register holds the processor status.

`Cause`
: This register holds the exception cause.

`Processor ID (PRId)`
: The Processor Identification register. Used to identify the CPU.

`Config`
: CPU Resource Information and Configuration.

`Config1`
: CPU Resource Information and Configuration 1.

`Bad Virtual Address (BadVAddr)`
: This register holds the address whose use led to and exception.

`Count`
: This register provide a simple general-purpose interval timer that runs continuously and that can be programmed to interrupt. This is a 32-bit counter that counts up continually at the CPU's pipeline clock rate. When `Count` reaches the maximum 32-bits unsigned value, it overflows quietly back to zero.

`Compare`
: This register provide a simple general-purpose interval timer that runs continuously and that can be programmed to interrupt. When the `Count` register increments to a value equal to `Compare`, the interrupt is raised. The interrupt remains asserted until cleared by a subsequent write to `Compare`.

`Exception Program Counter (EPC)`
: This register holds the address of the return point for the current exception.

`Error Exception Program Counter (ErrorEPC)`
: The read/write ErrorEPC register contains the virtual address at which instruction processing can resume after servicing an error.

This unit is implemented in Hardware/verilog/antares_cpzero.v

2.3.9 Memory Map

Due to the lack of MMU, virtual addresses are equal to physical addresses. Also, Antares does not implement the standard MIPS memory map, instead, it divides the 4GB memory space into 3 regions:

`Antares_SEG_0`
: Maps internal memory. Total size: 256 MB.

`Antares_SEG_1`
: Maps IO devices. Total size: 256 MB.

`Antares_SEG_2`
: Maps external memory. Total size: 3.5 GB.

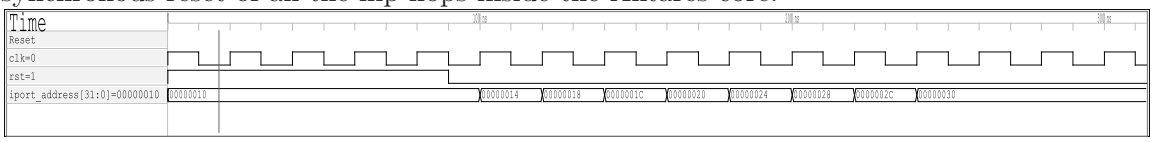
3 Core Operation

This chapter describes the operation of the Antares core.

3.1 Reset

The Antares core has one synchronous reset signal.

The following image shows the Antares reset sequence. The reset signal is connected to the synchronous reset of all the flip-flops inside the Antares core.



3.2 Core

The core implements a subset of the MIPS32 Release 1 architecture.

3.2.1 Instruction Set

The following list all the instructions implemented by Antares:

`ADD`	Add Word
`ADDI`	Add Immediate Word
`ADDIU`	Add Immediate Unsigned Word
`ADDU`	Add Unsigned Word
`AND`	And
`ANDI`	And Immediate
`BEQ`	Branch on Equal
`BGEZ`	Branch on Greater Than or Equal to Zero
`BGEZAL`	Branch on Greater Than or Equal to Zero and Link
`BGTZ`	Branch on Greater Than Zero
`BLEZ`	Branch on Less Than or Equal to Zero
`BLTZ`	Branch on Less Than Zero
`BLTZAL`	Branch on Less Than Zero and Link
`BNE`	Branch on Not Equal
`BREAK`	Breakpoint
`CLO`	Count Leading Ones in Word
`CLZ`	Count Leading Zeros in Word
`DIV`	Divide Word

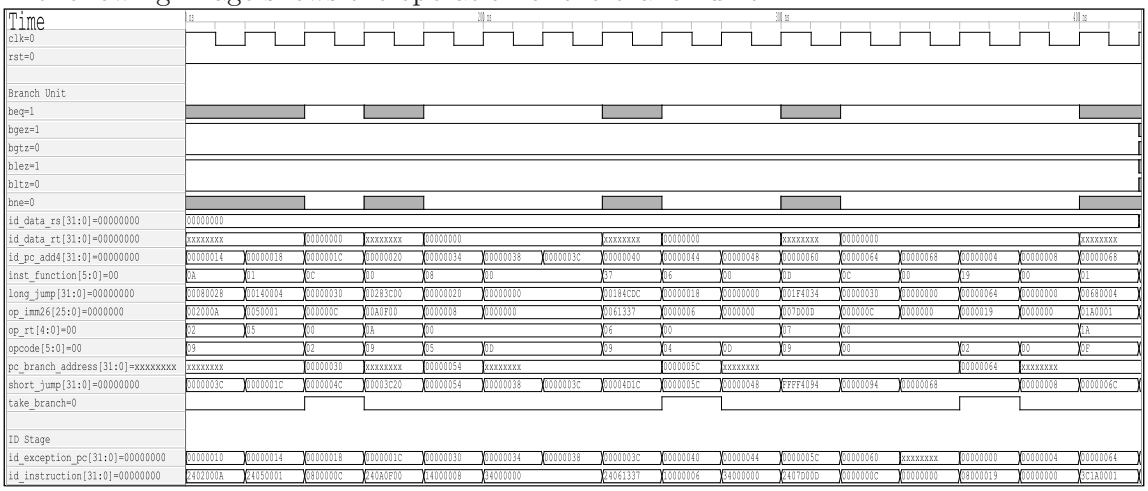
DIVU	Divide Unsigned Word
ERET	Exception Return
J	Jump
JAL	Jump and Link
JALR	Jump and Link Register
JR	Jump Register
LB	Load Byte
LBU	Load Byte Unsigned
LH	Load Halfword
LHU	Load Halfword Unsigned
LL	Load Linked Word
LUI	Load Upper Immediate
LW	Load Word
MADD	Multiply and Add Word to Hi, Lo
MADDU	Multiply and Add Unsigned Word to Hi,Lo
MFC0	Move from Coprocessor 0
MFHI	Move From HI Register
MFLO	Move From LO Register
MOVN	Move Conditional on Not Zero
MOVZ	Move Conditional on Zero
MSUB	Multiply and Subtract Word to Hi, Lo
MSUBU	Multiply and Subtract Word to Hi,Lo
MTC0	Move to Coprocessor 0
MTHI	Move to HI Register
MTLO	Move to LO Register
MULT	Multiply Word
MULTU	Multiply Unsigned Word
NOR	Not Or
OR	Or
ORI	Or Immediate
SB	Store Byte
SC	Store Conditional Word

`SH`	Store Halfword
`SLL`	Shift Word Left Logical
`SLLV`	Shift Word Left Logical Variable
`SLT`	Set on Less Than
`SLTI`	Set on Less Than Immediate
`SLTIU`	Set on Less Than Immediate Unsigned
`SLTU`	Set on Less Than Unsigned
`SRA`	Shift Word Right Arithmetic
`SRAV`	Shift Word Right Arithmetic Variable
`SRL`	Shift Word Right Logical
`SRLV`	Shift Word Right Logical Variable
`SUB`	Subtract Word
`SUBU`	Subtract Unsigned Word
`SW`	Store Word
`SYSCALL`	System Call
`TEQ`	Trap if Equal
`TEQI`	Trap if Equal Immediate
`TGE`	Trap if Greater or Equal
`TGEI`	Trap if Greater or Equal Immediate
`TGEIU`	Trap if Greater or Equal Immediate Unsigned
`TGEU`	Trap if Greater or Equal Unsigned
`TLT`	Trap if Less Than
`TLTI`	Trap if Less Than Immediate
`TLTIU`	Trap if Less Than Immediate Unsigned
`TLTU`	Trap if Less Than Unsigned
`TNE`	Trap if Not Equal
`TNEI`	Trap if Not Equal Immediate
`XOR`	Exclusive OR
`XORI`	Exclusive OR Immediate

3.2.2 Branch Unit

This unit detects branch and jump instructions, and check if the branch should be taken.

The following image shows the operation of the branch unit.



3.2.3 General-Purpose Registers (GPR)

General-Purpose register file can supply two read operands each clock cycle and store one result in a destination register.

3.2.4 Integer Execution Unit (ALU)

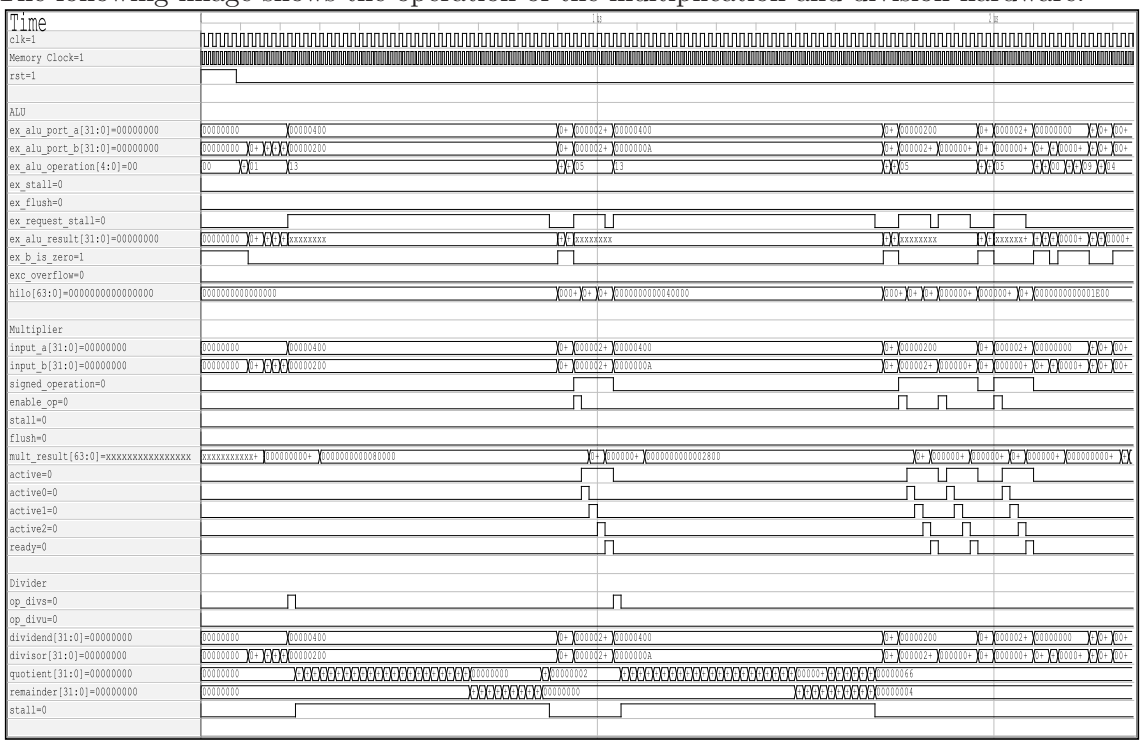
The core implements the following types of 32-bits integer instructions:

- Arithmetic instructions.
- Logical instructions.
- Compare instructions.
- Shift instructions.

Most integer instructions can execute in one cycle. The exceptions are the multiplication and division instructions, as it can be seen in the following table:

Instruction Group	**Clock cycles to execute**
Arithmetic except multiply/divide	1
Multiply	5
Divide	34
Compare	1
Logical	1
Shift	1

The following image shows the operation of the multiplication and division hardware.

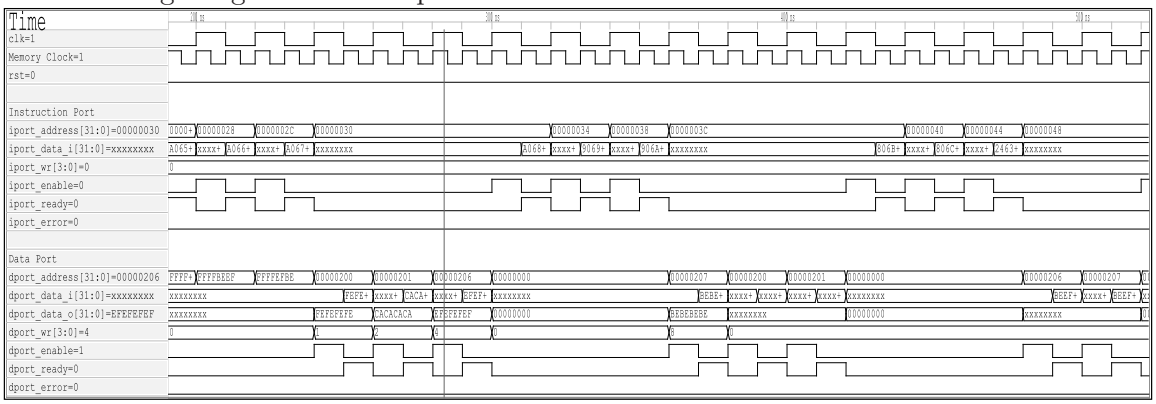


3.2.5 Load/Store Unit (LSU)

The load/store unit (LSU) transfers data between the core pipeline and the CPU internal bus. This unit will stall the master pipeline in case of data dependency.

The LSU can execute one load instruction every two clock cycle. Execution of store instructions takes two clock cycle. But if the memory system (or the interconnect) use a clock with double frequency (as can be seen in the next figure), load/store instructions can take one clock.

The following image shows the operation of this unit.



As can be seen in the previous figure, the enable signal is asserted while the ready signal from the bus is not asserted. The ready signal keeps asserted through one bus clock cycle because the enable signal is asynchronous and the ready signal is synchronous.

3.2.6 Hazard and Forwarding Unit

The hazard and forwarding unit implements the logic that is responsible for forwarding results back through the pipeline, allowing most instructions to be effectively executed in a single cycle. Also, this unit checks for data dependency (hazards) and interlocks the pipeline until the hazard has been resolved, avoiding the need to insert `NOP` instructions between dependent instructions.

3.2.7 Exception Unit: Coprocessor 0

The core implements a precise exception model. This means that when an exception is taken, the following conditions are met:

- Subsequent instructions in program flow are discarded.
- Previous instructions finish and write back their results.
- The address of the offended instruction is stored in the `EPC` (or `ErrorEPC` in case of reset or NMI).

4 Core Registers

This chapter describes all the registers inside the Antares core. All registers are 32-bit wide.

4.1 General-Purpose Registers (GPR)

This section list the 32 general-purpose registers.

Register Number	Register Name	Description
0	zero	Always equal to zero.
1	at	Assembler temporary. Used by the assembler.
2	v0	Return value from a function call.
3	v1	Return value from a function call.
4	a0	First parameter for a function call.
5	a1	Second parameter for a function call.
6	a2	Third parameter for a function call.
7	a3	Fourth parameter for a function call.
8	t0	Temporary variable. Does not need to be preserved
9	t1	Temporary variable. Does not need to be preserved
10	t2	Temporary variable. Does not need to be preserved
11	t3	Temporary variable. Does not need to be preserved
12	t4	Temporary variable. Does not need to be preserved
13	t5	Temporary variable. Does not need to be preserved
14	t6	Temporary variable. Does not need to be preserved
15	t7	Temporary variable. Does not need to be preserved
16	s0	Function variable. Must be preserved.
17	s1	Function variable. Must be preserved.
18	s2	Function variable. Must be preserved.
19	s3	Function variable. Must be preserved.
20	s4	Function variable. Must be preserved.
21	s5	Function variable. Must be preserved.
22	s6	Function variable. Must be preserved.
23	s7	Function variable. Must be preserved.
24	t8	Temporary variable.
25	t9	Temporary variable.
26	k0	Kernel use register.
27	k1	Kernel use register.
28	gp	Global pointer.
29	sp	Stack pointer.
30	fp/s8	Stack frame pointer, or subroutine variable.
31	ra	Return address of the last subroutine call.

4.2 Coprocessor 0 Registers

This section list all the MIPS compatible status control registers implemented in the Antares core.

Register Number	Register Name	Description
8	`Bad Virtual Address (BadVAddr)`	This register holds the address whose use led to and exception.
9	`Count`	This register provide a simple general-purpose interval timer that runs continuously and that can be programmed to interrupt. This is a 32-bit counter that counts up continually at the CPU's pipeline clock rate. When `Count` reaches the maximum 32-bits unsigned value, it overflows quietly back to zero.
11	`Compare`	This register provide a simple general-purpose interval timer that runs continuously and that can be programmed to interrupt. When the `Count` register increments to a value equal to `Compare`, the interrupt is raised. The interrupt remains asserted until cleared by a subsequent write to `Compare`.
12	`Status (SR)`	This register holds the processor status.
13	`Cause`	This register holds the exception cause.
14	`Exception Program Counter (EPC)`	This register holds the address of the return point for the current exception.
15	`Processor ID (PRId)`	The Processor Identification register. Used to identify the CPU.
16	`Config`	CPU Resource Information and Configuration.
16	`Config1`	CPU Resource Information and Configuration 1.
30	`Error Exception Program Counter (ErrorEPC)`	The read/write ErrorEPC register contains the virtual address at which instruction processing can resume after servicing an error.

4.2.1 Status Register (SR)

Description of the coprocessor 0 Status Register. Unimplemented fields will always read as zero:

`Status_CU_321[2:0] (bits 31-29)`
: Enable access control to coprocessors 3-1. Always zero.

`Status_CU_0 (bit 28)`
: Enable access control to coprocessor 0.

`Status_RP (bit 27)`
: Reduce power. CPU dependent. Always zero.

`Status_FR (bit 26)`
: Mode switch: Set 1 to expose 32 double-sized floating point registers to software. Set 0 to make them act as pairs of 32-bit registers. Always zero.

`Status_RE (bit 25)`
: Reverse endianness in user mode. Always zero.

`Status_MX (bit 24)`
: Enable for DSP or MDMX ASE instructions. Always zero.

`Status_PX (bit 23)`
: Enable 64-bit instructions with 32-bit addressing. Always zero.

`Status_BEV (bit 22)`
: Boot exception vectors: when `BEV` is 1, CPU uses the ROM (`kseg1`) space exception entry point (bootstrap). `BEV` is usually set to 0 in running systems.

`Status_TS (bit 21)`
: TLB shutdown. Always zero.

`Status_SR (bit 20)`
: A soft reset occurred. Always zero.

`Status_NMI (bit 19)`
: A non-maskable interrupt occurred.

`Status_RES[1:0] (bits 18-17)`
: Reserved. Always zero.

`Status_HALT (bit 16)`
: Stop CPU. MSUB specific implementation.

`Status_IM[7:0] (bits 15-8)`
: Interrupt mask.

`Status_KX (bit 7)`
: Enable 64-bit address space in kernel mode.

`Status_SX (bit 6)`
: Enable 64-bit address space in supervisor mode.

`Status_UX (bit 5)`
: Enable 64-bit address space in user mode.

`Status_KSU[1:0] (bits 4-3)`
: CPU privilege level: 0 for kernel, 1 for supervisor, 2 for user. Regardless of this setting, the CPU is in kernel mode whenever the `EXL` or `ERL` bits are set following an exception. The Antares core does not implement the supervisor mode.

`Status_ERL (bit 2)`
: Error level.

`Status_EXL (bit 1)`
: Exception level.

`Status_IE (bit 0)`
: Global interrupt enable. Note that `EXL` or `ERL` inhibit all interrupts, regardless.

4.2.2 Cause Register

Description of the coprocessor 0 Cause Register. Unimplemented fields will always read as zero:

`Cause_BD (bit 31)`
: Exception victim is in the delay slot.

`Cause_CE[1:0] (bits 29-28)`
: Coprocessor error: unusable coprocessor.

`Cause_IV (bit 23)`
: Write this bit to 1 to use a special exception entry point for interrupts.

`Cause_WP (bit 22)`
: Enable watchpoints exception mode. Always zero.

`Cause_IP[7:0] (bits 15-8)`
: Pending hardware interrupts.

`Cause_ExcCode[4:0] (bits 6-2)`
: Exception code.

4.2.3 Processor Identification Register (PRId)

Description of the coprocessor 0 Processor Identification Register. Read only registers:

`ID_Options[7:0] (bits 31-24)`
: Company options. Always zero.

`ID_CID[7:0] (bits 23-16)`
: Company ID. Always zero.

`ID_PID[7:0] (bits 15-8)`
: CPU ID. Always zero.

`ID_Rev[7:0] (bits 7-0)`
: Revision. Always zero.

4.2.4 Configuration Register (Config)

Description of the coprocessor 0 Configuration Register. Read only registers:

`Config_M (bit 31)`
: Continuation bit. Reads 1 if another configuration register is available.

`Config_Impl[14:0] (bits 30-16)`
: Implementation-dependent configuration flags.

`Config_BE (bit 15)`
: Endiannes. The Antares core us little endian.

`Config_AT[1:0] (bits 14-13)`
: MIPS32 CPU.

`Config_AR[2:0] (bits 12-10)`
: Architecture revision level: MIPS32 Release 1.

`Config_MT[2:0] (bits 9-7)`
: MMU type: none.

`Config_VI (bit 3)`
: Set 1 if the L1 I-cache is indexed and tagged with virtual (program) addresses. For this implementation, the L1 I-cache do not use virtual addresses.

`Config_K0[2:0] (bits 2-0)`
: `kseg0` coherency algorithm: uncached, cached, implementation dependent.

4.2.5 Configuration Register 1 (Config1)

Description of the coprocessor 0 Configuration Register 1. Read only registers:

`Config1_M (bit 31)`
: Continuation bit.

`Config1_MMU[5:0] (bits 30-25)`
: MMU size.

`Config1_IS[2:0] (bits 24-22)`
: Number of cache index positions: 64 x 2^S.

`Config1_IL[2:0] (bits 21-19)`
: Zero means no cache at all. Else, number of cache line size: 2^(L+1) bytes.

`Config1_IA[2:0] (bits 18-16)`
: Cache associativity: (A + 1).

`Config1_DS[2:0] (bits 15-13)`
: Number of cache index positions: 64 x 2^S.

`Config1_DL[2:0] (bits 12-10)`
: Zero means no cache at all. Else, number of cache line size: 2^(L+1) bytes.

`Config1_DA[2:0] (bits 9-7)`
: Cache associativity: (A + 1).

`Config1_C2 (bit 6)`
: 1 if the coprocessor 2 is implemented.

`Config1_MD (bit 5)`
: 1 if the MDMX ASE is implemented.

`Config1_PC (bit 4)`
: There is at least one performance counter implemented.

`Config1_WR (bit 3)`
: Reads 1 if the CPU has at least one watchpoint register.

`Config1_CA (bit 2)`
: Reads 1 when the MIPS16e compressed-code instruction set is available.

`Config1_EP (bit 1)`
: Reads 1 if an EJTAG debug unit is provided.

`Config1_FP (bit 0)`
: A floating-point unit is attached.

4.2.6 Bad Virtual Address Register (BadVAddr)

This register holds the address whose use led to an exception: it is set on an MMU-related exception, on an attempt by a user program to access addresses outside `kuseg`, or if an address is wrongly aligned. After any other exception it is undefined. This is not set after a bus error.

4.2.7 Count/Compare Register

These registers provide a simple general-purpose interval timer that runs continuously and that can be programmed to interrupt.

`count` is a 32-bit counter that counts up continually at the CPU's pipeline clock rate.

When `count` reaches the maximum 32-bit unsigned value, it overflows quietly back to zero. You can read `count` to find the current "time". You can also write `count` at any time, but it is normal practice not to do so.

`compare` is a 32-bit read/write register. When `count` increments to a value equal to `compare`, the interrupt is raised. The interrupt remains asserted until cleared by a subsequent write to `compare`.

To produce a periodic interrupt, the interrupt handler should always increment `compare` by a fixed amount (not an increment to `count`, because the period would then get stretched slightly by interrupt latency). The software needs to check for the possibility that a late interrupt response might lead it to set `compare` to a value that `count` has already passed.

4.2.8 Exception Program Counter

This register holds the address of the return point for the current exception. The instruction causing (or suffering) the exception is at `EPC`.

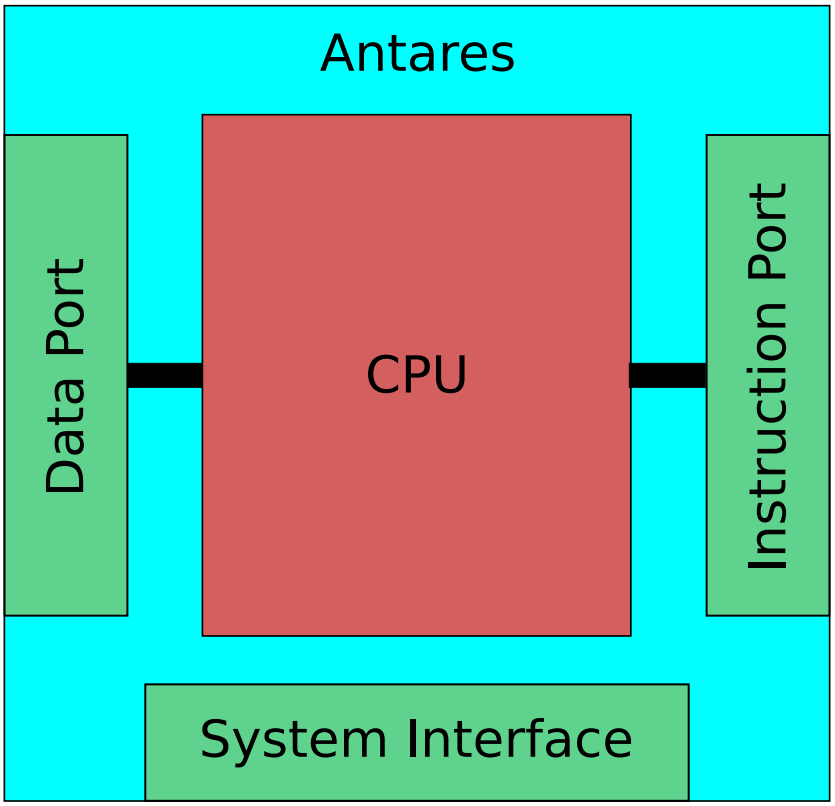
4.2.9 Error Register

The read/write ErrorEPC register contains the virtual address at which instruction processing can resume after servicing an error, a reset or a non-maskable interrupt.

5 IO Ports

The Antares core have several interfaces:

- Instruction interface.
- Data interface.
- Interrupts interface.
- System interface.



5.1 Instruction Interface

This interface is used to connect the Antares core to the memory subsystem for the purpose of fetching instructions.

Port	Width	Direction	Description
`iport_data_i`	32	Input	Data input
`iport_ready`	1	Input	Data ready
`iport_error`	1	Input	Bus error
`iport_address`	32	Output	Memory address
`iport_wr`	4	Output	Byte select in write mode
`iport_enable`	1	Output	Enable operation

5.2 Data Interface

This interface is used to connect the Antares core to the memory subsystem for the purpose of reading and writing data.

Port	Width	Direction	Description
dport_data_i	32	Input	Data input
dport_ready	1	Input	Data ready
dport_error	1	Input	Bus error
dport_address	32	Output	Memory address
dport_data_o	32	Output	Data output
dport_wr	4	Output	Byte select in write mode
dport_enable	1	Output	Enable operation

5.3 Interrupts Interface

Inputs for interfacing external peripheral's interrupt outputs to the Antares core.

Port	Width	Direction	Description
interrupts	5	Input	External interrupts
nmi	1	Input	Non-maskable interrupt

5.4 System Interface

Connects resert, clock and other system signals to the Antares core.

Port	Width	Direction	Description
clk	1	Input	Main clock
rst	1	Input	Synchronous reset
halted	1	Output	CP0 Status Register, bit 16. Stop mode

6 Core Configuration

This chapter describes parameters that are set by the user of the core and defines the configuration of the core. Parameters must be set by the user before actual use of the core in simulation or synthesis.

Parameter Name	Range	Default	Description
ENABLE_HW_MULT	0-1	1	Enable hardware support for multiplication instructions.
ENABLE_HW_DIV	0-1	1	Enable hardware support for division instructions.
ENABLE_HW_CLO_Z	0-1	1	Enable hardware support for the CLO and CLZ instructions.

7 Software Development Tools

This chapter describes the software used to test and develop code for the Antares core.

The Software folder contains tools and libraries to develop code for the Antares core.

The Simulation folder contains scripts needed to simulate the core using Icarus Verilog.

7.1 MIPS Cross-compiler Toolchain (GCC)

The easy way to get a MIPS cross-compiler toolchain is using the Mentor Graphics Sourcery CodeBench Lite for MIPS ELF. This compiler is based on GCC, Binutils and Newlib.

Or, if you prefer to create your own toolchain ("hard way"), you need to follow the instructions in Section 7.1.1 [Toolchain Instructions], page 24.

7.1.1 Instructions for Creating a MIPS Compiler Toolchain

These are instructions for creating a MIPS cross-compiler toolchain based on GCC, Binutils, and Newlib.

7.1.1.1 Required Files

The required files needed to build the MIPS cross-compiler (you can use updated versions of these files):

- gcc-4.9.1.tar.bz2 : http://gcc.gnu.org/mirrors.html
- binutils-2.24.tar.bz2 : http://ftp.gnu.org/gnu/binutils/
- mpfr-3.1.2.tar.bz2 : http://www.mpfr.org/mpfr-3.1.2/
- mpc-1.0.2.tar.gz : http://www.multiprecision.org/
- gmp-6.0.0a.tar.bz2 : http://gmplib.org/
- newlib-2.1.0.tar.gz : ftp://sources.redhat.com/pub/newlib/index.html

7.1.1.2 Environment Requirements

- Linux, BSD, Cygwin, or another Unix-like environment.
- Recent compiler tools (tested with GCC 4.8.2).
- Bash or bask-like shell (or adjust the instructions below).

7.1.1.3 Building the Toolchain

- Set environment variables:

```
export TARGET=mips-elf
export PREFIX=[any directory]/mips32/mips_tc
export PATH=$PATH:$PREFIX/bin
```

- Unpack everything:

```
bzip2 -dc binutils-2.24.tar.bz2 | tar xf -
bzip2 -dc gcc-4.9.1.tar.bz2 | tar xf -
bzip2 -dc mpfr-3.1.2.tar.bz2 | tar xf -
bzip2 -dc gmp-6.0.0a.tar.bz2 | tar xf -
gzip -dc mpc-1.0.2.tar.gz | tar xf -
gzip -dc newlib-2.1.0.tar.gz | tar xf -
```

- Move (or symlink) GCC dependency packages:

```
mv gmp-6.0.0 gcc-4.9.1/gmp
mv mpc-1.0.2 gcc-4.9.1/mpc
mv mpfr-3.1.2 gcc-4.9.1/mpfr
mv newlib-2.1.0/newlib gcc-4.9.1/newlib
mv newlib-2.1.0/libgloss gcc-4.9.1/libgloss
```

- Build binutils:

```
mkdir binutils-build && cd binutils-build
../binutils-2.24/configure --prefix=$PREFIX --target=$TARGET \
    --disable-nls
make
make install
cd ..
```

- build GCC:

```
mkdir gcc-build && cd gcc-build
../gcc-4.9.1/configure --prefix=$PREFIX --target=$TARGET \
    --with-newlib  --without-headers --with-gnu-ld --with-gnu-as \
    --disable-libssp --disable-nls --enable-c99 --enable-long-long \
    --enable-languages=c
make
make install
cd ..
```

At this point you have a complete toolchain located at `$PREFIX`.

7.2 Simulation

The Simulation folder contains scripts needed to simulate the core.

The folder layout:

- bench: testbenchs for the core and SoC.
- run: simulation makefile.
- scripts: scripts to compile the core, and simulate it.
- tests: demos written in assembler.

7.2.1 Run the simulation.

To simulate the core, follow the makefile instructions:

- Change directory to `<project directory>\Simulation\run`.
- Execute `make` to get the help screen.

Make Help Screen

The usage: `make TARGET VARIABLES`

Make Targets

`check` Check verilog files found in `Hardware` and `Simulation/bench/` directory.

`list_asm_tests`
List all assembler files inside the `Simulation/tests/asm/` folder.

`list_c_tests`
List all C projects inside the `Simulation/tests/c/` folder.

`rtlsim` Simulates a single ASM test, and places all outputs (waveforms, regdump, logs) in Simulation/out folder

`rtlsim-c` Simulates a single C test, and places all outputs (waveforms, regdump, logs) in `Simulation/out` folder

`rtlsim-all`
Simulates all ASM tests, and places all outputs (waveforms, regdump, logs) in `Simulation/out` folder

`clean` Clean temporary files inside the `Simulation` folder.

`distclean`
Clean all temporary files (includes the `Software/utils` folder).

Make Variables

`TB=<verilog testbench>`
For `rtlsim` and `rtlsim-c` targets. Specifies the testbench file for simulation.

`TEST=<ASM test>`
For `rtlsim` and `rtlsim-c` targets. Specifies the assembler/C test.

`MEM_SIZE=<memory size (bytes)>`
For `rtlsim` and `rtlsim-c` targets. Specifies system memory.

`DSEG_SIZE=<Data segment size>`
For `rtlsim` and `rtlsim-c` targets. Specifies the size for the data segment.

`TIMEOUT=<Simulation timeout>`
For `rtlsim` and `rtlsim-c` targets.

`DUMPVCD=<Generate waveform file>`
For `rtlsim` and `rtlsim-c` targets. Enable the dump of variables to waveform file.

Make Examples

Some examples, for each target:

```
make
make help
make check
make list_asm_tests
make list_c_tests
```

```
make rtlsim TB=tb_core TEST=<asm-test> MEM_SIZE=4096 DSEG_SIZE=1024 \
    TIMEOUT=100000 DUMPVCD=0
make rtlsim-c TB=tb_core TEST=<c-test> MEM_SIZE=4096 DSEG_SIZE=1024 \
    TIMEOUT=100000 DUMPVCD=0
make rtlsim-all TB=tb_core MEM_SIZE=4096 DSEG_SIZE=1024 TIMEOUT=100000 \
    DUMPVCD=0
make clean
make distclean
```

8 MIT License

Copyright (c) 2015 Angel Terrones (angelterrones@gmail.com)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Index

A

ADD ... 9
ADDI ... 9
ADDIU ... 9
ADDU ... 9
AND ... 9
ANDI ... 9
Antares_SEG_0 ... 8
Antares_SEG_1 ... 8
Antares_SEG_2 ... 8
Architecture ... 4

B

Bad Virtual Address (BadVAddr) ... 8
BEQ ... 9
BGEZ ... 9
BGEZAL ... 9
BGTZ ... 9
BLEZ ... 9
BLTZ ... 9
BLTZAL ... 9
BNE ... 9
BREAK ... 9

C

Cause ... 8
Cause_BD (bit 31) ... 18
Cause_CE[1:0] (bits 29-28) ... 18
Cause_ExcCode[4:0] (bits 6-2) ... 18
Cause_IP[7:0] (bits 15-8) ... 18
Cause_IV (bit 23) ... 18
Cause_WP (bit 22) ... 18
CLO ... 9
CLZ ... 9
Compare ... 8
Config ... 8
Config_AR[2:0] (bits 12-10) ... 19
Config_AT[1:0] (bits 14-13) ... 18
Config_BE (bit 15) ... 18
Config_Impl[14:0] (bits 30-16) ... 18
Config_K0[2:0] (bits 2-0) ... 19
Config_M (bit 31) ... 18
Config_MT[2:0] (bits 9-7) ... 19
Config_VI (bit 3) ... 19
Config1 ... 8
Config1_C2 (bit 6) ... 19
Config1_CA (bit 2) ... 19
Config1_DA[2:0] (bits 9-7) ... 19
Config1_DL[2:0] (bits 12-10) ... 19
Config1_DS[2:0] (bits 15-13) ... 19
Config1_EP (bit 1) ... 20
Config1_FP (bit 0) ... 20
Config1_IA[2:0] (bits 18-16) ... 19
Config1_IL[2:0] (bits 21-19) ... 19
Config1_IS[2:0] (bits 24-22) ... 19
Config1_M (bit 31) ... 19
Config1_MD (bit 5) ... 19
Config1_MMU[5:0] (bits 30-25) ... 19
Config1_PC (bit 4) ... 19
Config1_WR (bit 3) ... 19
Core Configuration ... 23
Core Operation ... 9
Core Registers ... 15
Count ... 8

D

DIV ... 9
DIVU ... 10

E

ENABLE_HW_CLO_Z ... 6
ENABLE_HW_DIV ... 6
ENABLE_HW_MULT ... 6
ERET ... 10
Error Exception Program Counter (ErrorEPC) ... 8
EXC_AdEL ... 7
EXC_AdES ... 7
EXC_AdIF ... 7
EXC_Bp ... 7
EXC_CpU ... 7
EXC_DBE ... 7
EXC_IBE ... 7
EXC_Int ... 7
EXC_Ov ... 7
EXC_RI ... 7
EXC_Sys ... 7
EXC_Tr ... 7
Exception Program Counter (EPC) ... 8

I

ID_CID[7:0] (bits 23-16) ... 18
ID_Options[7:0] (bits 31-24) ... 18
ID_PID[7:0] (bits 15-8) ... 18
ID_Rev[7:0] (bits 7-0) ... 18
Introduction to Antares ... 1
IO Ports ... 21

J

J ... 10

JAL 10
JALR 10
JR 10

L

LB 10
LBU 10
LH 10
LHU 10
LL 10
LUI 10
LW 10

M

MADD 10
MADDU 10
MFC0 10
MFHI 10
MFLO 10
MOVN 10
MOVZ 10
MSUB 10
MSUBU 10
MTC0 10
MTHI 10
MTLO 10
MULT 10
MULTU 10

N

NOR 10

O

OR 10
ORI 10

P

Processor ID (PRId) 8

S

SB 10
SC 10
SH 11
SLL 11
SLLV 11
SLT 11
SLTI 11
SLTIU 11
SLTU 11
Software Development Tools 24
source files, downloading 1
SRA 11
SRAV 11
SRL 11
SRLV 11
Status (SR) 8
Status_BEV (bit 22) 17
Status_CU_0 (bit 28) 16
Status_CU_321[2:0] (bits 31-29) 16
Status_ERL (bit 2) 17
Status_EXL (bit 1) 17
Status_FR (bit 26) 17
Status_HALT (bit 16) 17
Status_IE (bit 0) 18
Status_IM[7:0] (bits 15-8) 17
Status_KSU[1:0] (bits 4-3) 17
Status_KX (bit 7) 17
Status_MX (bit 24) 17
Status_NMI (bit 19) 17
Status_PX (bit 23) 17
Status_RE (bit 25) 17
Status_RES[1:0] (bits 18-17) 17
Status_RP (bit 27) 16
Status_SR (bit 20) 17
Status_SX (bit 6) 17
Status_TS (bit 21) 17
Status_UX (bit 5) 17
SUB 11
SUBU 11
supported host platforms 1
SW 11
SYSCALL 11

T

TEQ 11
TEQI 11
TGE 11
TGEI 11
TGEIU 11
TGEU 11
TLT 11
TLTI 11
TLTIU 11
TLTU 11
TNE 11
TNEI 11

X

XOR 11
XORI 11