# OpenRISC 1200 IP Core Specification (Preliminary Draft)

## REVISION HISTORY

| NUMBER | DATE | DESCRIPTION | NAME |
|---|---|---|---|
| v0.1 | 28/3/01 | First Draft | Damjan Lampret |
| v0.2 | 16/4/01 | First time published | Damjan Lampret |
| v0.3 | 29/4/01 | All chapters almost finished. Some bugs hidden waiting for an update. Awaiting feedback. | Damjan Lampret |
| v0.4 | 16/5/01 | Synchronization with OR1K Arch Manual | Damjan Lampret |
| v0.5 | 24/5/01 | Fixed bugs | Damjan Lampret |
| v0.6 | 28/5/01 | Changed some SPR addresses. | Damjan Lampret |
| v0.7 | 06/9/01 | Simplified debug unit. | Damjan Lampret |
| v0.8 | 30/08/10 | Adding information about FPU implementation, data cache write-back capability. PIC behavior update. Instruction list update. Update of bits in config registers, bringing into line with latest OR1200 - not entirely complete. | Julius Baxter |
| v0.9 | 12/9/10 | Clarified supported parts of OR1K instruction set. Updated core clock input information. Fixed up reference to instruction execute stage cycle table. Added divide cycles to execute stage cycle table. | Julius Baxter |
| 0.10 | 1/11/10 | Added FF1/FL1 instructions to supported instructions table. | Julius Baxter |
| v0.11 | 19/1/11 | Cache information update. Wishbone behavior clarification. Serial integer multiply/divide update. Reset address clarification | Julius Baxter |

# Contents

1 Introduction 1

- 1.1 OpenRISC Family . . . . . 1
- 1.2 OpenRISC 1200 . . . . . 2
  - 1.2.1 Features . . . . . 2

2 Architecture 3

- 2.1 CPU/FPU/DSP . . . . . 4
  - 2.1.1 Instruction unit . . . . . 4
  - 2.1.2 General-Purpose Registers . . . . . 4
  - 2.1.3 Load/Store Unit . . . . . 4
  - 2.1.4 Integer Execution Pipeline . . . . . 5
  - 2.1.5 MAC Unit . . . . . 5
  - 2.1.6 Floating Point Unit . . . . . 5
  - 2.1.7 System Unit . . . . . 5
  - 2.1.8 Exceptions . . . . . 5
- 2.2 Data Cache . . . . . 6
- 2.3 Instruction Cache . . . . . 7
- 2.4 Data MMU . . . . . 8
- 2.5 Instruction MMU . . . . . 9
- 2.6 Programmable Interrupt Controller . . . . . 11
- 2.7 Tick Timer . . . . . 11
- 2.8 Power Management Support . . . . . 11
- 2.9 Debug unit . . . . . 12
- 2.10 Clocks & Reset . . . . . 12
- 2.11 WISHBONE Interfaces . . . . . 13

3 Operation 14

- 3.1 Reset . . . . . 14
- 3.2 CPU/FPU/DSP . . . . . 14
  - 3.2.1 Instructions . . . . . 15
  - 3.2.2 Instruction Unit . . . . . 15

- 3.2.3 General-Purpose Registers . . . 15
- 3.2.4 Load/Store Unit . . . 15
- 3.2.5 Integer Execution Pipeline . . . 16
- 3.2.6 MAC Unit . . . 16
- 3.2.7 Floating Point Unit . . . 16
- 3.2.8 System Unit . . . 17
- 3.2.9 Exceptions . . . 17

3.3 Data Cache Operation . . . 18
- 3.3.1 Data Cache Load/Store Access . . . 18
- 3.3.2 Data Cache Line Fill Operation . . . 18
- 3.3.3 Cache/Memory Coherency . . . 19
- 3.3.4 Data Cache Enabling/Disabling . . . 19
- 3.3.5 Data Cache Invalidation . . . 19
- 3.3.6 Data Cache Locking . . . 19
- 3.3.7 Data Cache Line Prefetch . . . 19
- 3.3.8 Data Cache Line Flush . . . 19
- 3.3.9 Data Cache Line Invalidate . . . 20
- 3.3.10 Data Cache Line Write-back . . . 20
- 3.3.11 Data Cache Line Lock . . . 20
- 3.3.12 Data Cache inhibit with address bit 31 set . . . 20

3.4 Instruction Cache Operation . . . 20
- 3.4.1 Instruction Cache Instruction Fetch Access . . . 20
- 3.4.2 Instruction Cache Line Fill Operation . . . 20
- 3.4.3 Cache/Memory Coherency . . . 21
- 3.4.4 Instruction Cache Enabling/Disabling . . . 21
- 3.4.5 Instruction Cache Invalidation . . . 21
- 3.4.6 Instruction Cache Locking . . . 21
- 3.4.7 Instruction Cache Line Prefetch . . . 21
- 3.4.8 Instruction Cache Line Invalidate . . . 21
- 3.4.9 Instruction Cache Line Lock . . . 21

3.5 Data MMU . . . 21
- 3.5.1 Translation Disabled . . . 21
- 3.5.2 Translation Enabled . . . 21
- 3.5.3 DMMUCR and Flush of Entire DTLB . . . 22
- 3.5.4 Page Protection . . . 22
- 3.5.5 DTLB Entry Reload . . . 23
- 3.5.6 DTLB Entry Invalidation . . . 23
- 3.5.7 Locking DTLB Entries . . . 23
- 3.5.8 Page Attribute - Dirty (D) . . . 23

3.5.9 Page Attribute - Accessed (A) . . . . . 23
3.5.10 Page Attribute - Weakly Ordered Memory (WOM) . . . . . 23
3.5.11 Page Attribute - Write-Back Cache (WBC) . . . . . 23
3.5.12 Page Attribute - Caching-Inhibited (CI) . . . . . 23
3.5.13 Page Attribute - Cache Coherency (CC) . . . . . 24
3.6 Instruction MMU . . . . . 24
3.6.1 Translation Disabled . . . . . 24
3.6.2 Translation Enabled . . . . . 24
3.6.3 IMMUCR and Flush of Entire ITLB . . . . . 25
3.6.4 Page Protection . . . . . 25
3.6.5 ITLB Entry Reload . . . . . 25
3.6.6 ITLB Entry Invalidation . . . . . 25
3.6.7 Locking ITLB Entries . . . . . 25
3.6.8 Page Attribute - Dirty (D) . . . . . 26
3.6.9 Page Attribute - Accessed (A) . . . . . 26
3.6.10 Page Attribute - Weakly Ordered Memory (WOM) . . . . . 26
3.6.11 Page Attribute - Write-Back Cache (WBC) . . . . . 26
3.6.12 Page Attribute - Caching-Inhibited (CI) . . . . . 26
3.6.13 Page Attribute - Cache Coherency (CC) . . . . . 26
3.7 Programmable Interrupt Controller . . . . . 26
3.8 Tick Timer . . . . . 27
3.9 Power Management . . . . . 27
3.9.1 Clock Gating and Frequency Changing Versus CPU Stalling . . . . . 27
3.9.2 Slow Down Mode . . . . . 27
3.9.3 Doze Mode . . . . . 27
3.9.4 Sleep Mode . . . . . 27
3.9.5 Clock Gating . . . . . 28
3.9.6 Disabled Units Force Clock Gating . . . . . 28
3.10 Debug Unit . . . . . 28
3.10.1 Watchpoints . . . . . 28
3.10.2 Breakpoint Exception . . . . . 28
3.11 Development Interface . . . . . 28
3.11.1 Debugging Through Development Interface . . . . . 28
3.11.2 Reading PC, Load/Store EA, Load Data, Store Data, Instruction . . . . . 28
3.11.3 Reading and Writing SPRs Through Development Interface . . . . . 29
3.11.4 Tracking Data Flow . . . . . 29
3.11.5 Tracking Program Flow . . . . . 30
3.11.6 Triggering External Watchpoint Event . . . . . 30

**4 Registers** **31**

4.1 Registers list . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31

4.2 Register VR description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

4.3 Register UPR description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

4.4 Register CPUCFGR description . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34

4.5 Register DMMUCFGR description . . . . . . . . . . . . . . . . . . . . . . . . . . 35

4.6 Register IMMUCFGR description . . . . . . . . . . . . . . . . . . . . . . . . . . 35

4.7 Register DCCFGR description . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36

4.8 Register ICCFGR description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36

4.9 Register DCFGR description . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

**5 IO ports** **38**

5.1 Instruction WISHBONE Master Interface . . . . . . . . . . . . . . . . . . . . . . . 38

5.2 Data WISHBONE Master Interface . . . . . . . . . . . . . . . . . . . . . . . . . . 39

5.3 System Interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40

5.4 Development Interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40

5.5 Power Management Interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41

5.6 Interrupt Interface . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41

**A Core HW Configuration** **43**

**6 Bibliography** **44**

6.1 Bibliography . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44

**7 Index** **45**

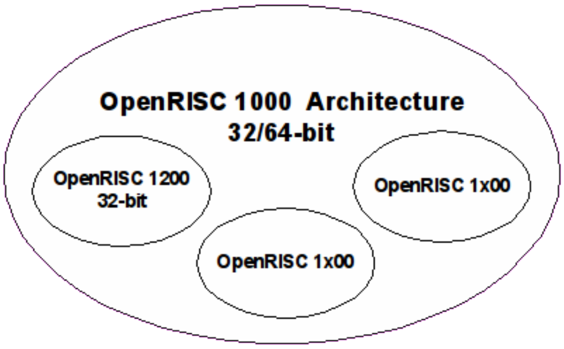
# Chapter 1

# Introduction

Purpose of this document is to define specifications of the OpenRISC 1200 implementation. This specification defines all implementation specific variables that are not part of the general architecture specification. This includes type and size of data and instruction caches, type and size of data and instruction MMUs, details of all execution pipelines, implementation of exception unit, interrupt controller and other supplemental units. This document does not cover general architecture topics like instruction set, memory addressing modes and other architectural definitions. See [or1000_manual] for more information about architecture.

## 1.1 OpenRISC Family

OpenRISC 1000 is architecture for a family of free, open source RISC processor cores. As architecture, OpenRISC 1000 allows for a spectrum of chip and system implementations at a variety of price/performance points for a range of applications. It is a 32/64-bit load and store RISC architecture designed with emphasis on performance, simplicity, low power requirements, scalability and versatility. OpenRISC 1000 architecture targets medium and high performance networking, embedded, automotive and portable computer environments.



All OpenRISC implementations, whose first digit in identification number is 1 , belong to OpenRISC 1000 family. Second digit defines which features of OpenRISC 1000 architecture are implemented and in which way they are implemented. Last two digits define how an implementation is configured before it is used in a real application.

However, at present the OR1200 is the only major RTL implementation of the OR1K architecture spec, and the OR1200 name has stuck, despite the high level of reconfigurability possible that would, strictly speaking, mean the core is either a OR1000, OR1300, etc. So, despite the various features that may or may not be implemented, the core is still only referred to as the OR1200.

## 1.2 OpenRISC 1200

The OR1200 is a 32-bit scalar RISC with Harvard microarchitecture, 5 stage integer pipeline, virtual memory support (MMU) and basic DSP capabilities. Default caches are 1-way direct-mapped 8KB data cache and 1-way direct-mapped 8KB instruction cache, each with 16-byte line size. Both caches are physically tagged. By default MMUs are implemented and they are constructed of 64-entry hash based 1-way direct-mpped data TLB and 64-entry hash based 1-way direct-mapped instruction TLB.

Supplemental facilities include debug unit for real-time debugging, high resolution tick timer, programmable interrupt controller and power management support. When implemented in a typical 0.18u 6LM process it should provide over 300 dhrystone 2.1 MIPS at 300MHz and 300 DSP MAC 32x32 operations, at least 20% more than any other competitor in this class. OR1200 in default configuration has about 1M transistors.

OR1200 is intended for embedded, portable and networking applications. It can successfully compete with latest scalar 32-bit RISC processors in his class and can efficiently run any modern operating system. Competitors include ARM10, ARC and Tensilica RISC processors.

### 1.2.1 Features

The following lists the main features of OR1200 IP core:

- All major characteristics of the core can be set by the user
- High performance of 300 Dhrystone 2.1 MIPS at 300 MHz using 0.18u process
- High performance cache and MMU subsystems
- WISHBONE SoC Interconnection Rev. B3 compliant interface

# Chapter 2

# Architecture

Figure 2.1 below shows general architecture of OR1200 IP core. It consists of several building blocks:

- CPU/FPU/DSP central block
- Direct-mapped data cache
- Direct-mapped instruction cache
- Data MMU based on hash based DTLB
- Instruction MMU based on hash based ITLB
- Power management unit and power management interface
- Tick timer
- Debug unit and development interface
- Interrupt controller and interrupt interface
- Instruction and Data WISHBONE host interfaces

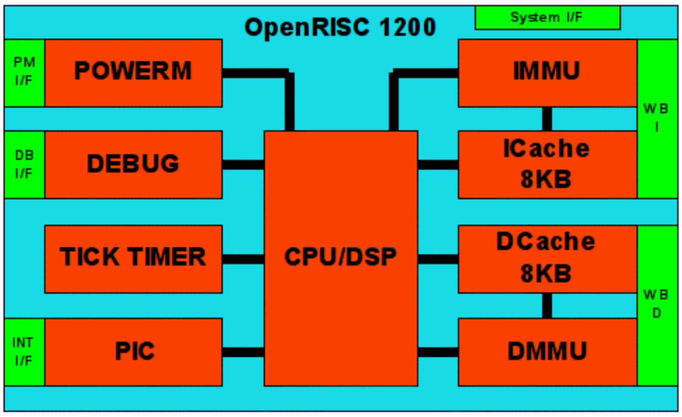

OpenRISC 1200
System I/F
PM I/F
POWERM
DB I/F
DEBUG
TICK TIMER
INT I/F
PIC
CPU/DSP
IMMU
ICache 8KB
DCache 8KB
DMMU
WB I
WB D

Figure 2.1: Core's Architecture

## 2.1 CPU/FPU/DSP

CPU/FPU/DSP is a central part of the OR1200 RISC processor. Figure 2.2 shows basic block diagram of the CPU/DSP. Not pictured are the FPU components. OR1200 CPU/FPU/DSP ony implements sections of the ORBIS32 and ORFPX32 instruction set. No ORBIS64, ORFBX64 or ORVDX64 instructions are implemented in OR1200.

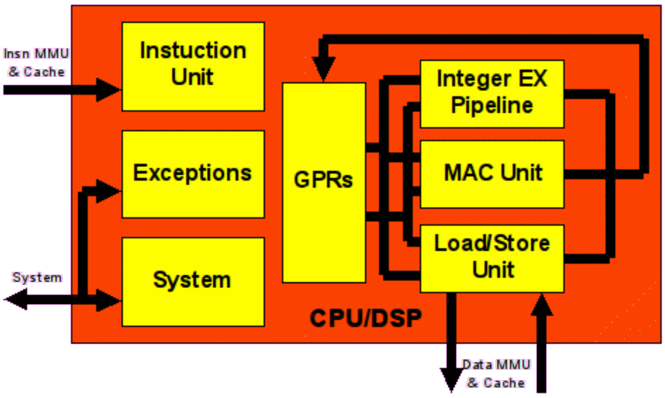


Figure 2.2: CPU/FPU/DSP Block Diagram

### 2.1.1 Instruction unit

The instruction unit implements the basic instruction pipeline, fetches instructions from the memory subsystem, dispatches them to available execution units, and maintains a state history to ensure a precise exception model and that operations finish in order. It also executes conditional branch and unconditional jump instructions.

The sequencer can dispatch a sequential instruction on each clock if the appropriate execution unit is available. The execution unit must discern whether source data is available and to ensure that no other instruction is targeting the same destination register.

Instruction unit handles only ORBIS32 and, optionally, a subset of the ORFPX32 instruction class. Some ORFPX32 and all ORFPX3264 and ORVDX64 instruction classes are not supported by the OR1200 at present.

### 2.1.2 General-Purpose Registers

OpenRISC 1200 implements 32 general-purpose 32-bit registers. OpenRISC 1000 architecture also support shadow copies of register file to implement fast switching between working contexts, however this feature is not implemented in current OR1200 implementation.

OR1200 implements general-purpose register file as two synchronous dual-port memories with capacity of 32 words by 32 bits per word.

### 2.1.3 Load/Store Unit

The load/store unit (LSU) transfers all data between the GPRs and the CPU’s internal bus. It is implemented as an independent execution unit so that stalls in memory subsystem only affect master pipeline if there is a data dependency.

The following are LSU’s main features:

- all load/store instruction implemented in hardware (atomic instructions included)
- address entry buffer
- pipelined operation

- aligned accesses for fast memory access

When load and store instructions are issued, the LSU determines if all operands are available. These operands include the following:

- address register operand
- source data register operand (for store instructions)
- destination data register operand (for load instructions)

### 2.1.4 Integer Execution Pipeline

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

- Arithmetic instructions
- Compare instructions
- Logical instructions
- Rotate and shift instructions

Most integer instructions can execute in one cycle. For details about timing see Table 3.2.

### 2.1.5 MAC Unit

The MAC unit executes DSP MAC operations. MAC operations are 32x32 with 48-bit accumulator. MAC unit is fully pipelined and can accept new MAC operation in each new clock cycle.

### 2.1.6 Floating Point Unit

The FPU implementation is based on two other FPUs available from OpenCores.org. For the comparison and conversion functions, parts were taken from the FPU project by Rudolf Usselmann, and for the arithmetic operations, the fpu100 project by Jidan Al-Eryani was converted to Verilog HDL.

All ORFPX32 instructions except for lf.madd.s and lf.rem.s are supported when the FPU is enabled in the OR1200 configuration.

### 2.1.7 System Unit

The system unit connects all other signals of the CPU/FPU/DSP that are not connected through instruction and data interfaces. It also implements all system special-purpose registers (e.g. supervisor register).

### 2.1.8 Exceptions

Core exceptions can be generated when an exception condition occurs. Exception sources in OR1200 include the following:

- External interrupt request
- Certain memory access condition
- Internal errors, such as an attempt to execute unimplemented opcode
- System call
- Internal exception, such as breakpoint exceptions

Exception handling is transparent to user software and uses the same mechanism to handle all types of exceptions. When an exception is taken, control is transferred to an exception handler at an offset defined by for the type of exception encountered. Exceptions are handled in supervisor mode.

## 2.2 Data Cache

The default configuration of OR1200 data cache is 8-Kbyte, 1-way direct-mapped data cache, which allows rapid core access to data. However data cache can be configured according to Table 2.1.

Table 2.1: Possible Data Cache Configurations of OR1200

| | **Direct mapped** |
|---|---|
| 16B/line, 256 lines, 1 way | 4KB |
| 16B/line, 512 lines, 1 way | **8KB (default)** |
| 16B/line, 1024 lines, 1 way | 16KB |
| 32B/line, 1024 lines, 1 way | 32KB |

It is possible to operate the data cache with write-through or write-back strategies, however write-back is currently experimental.

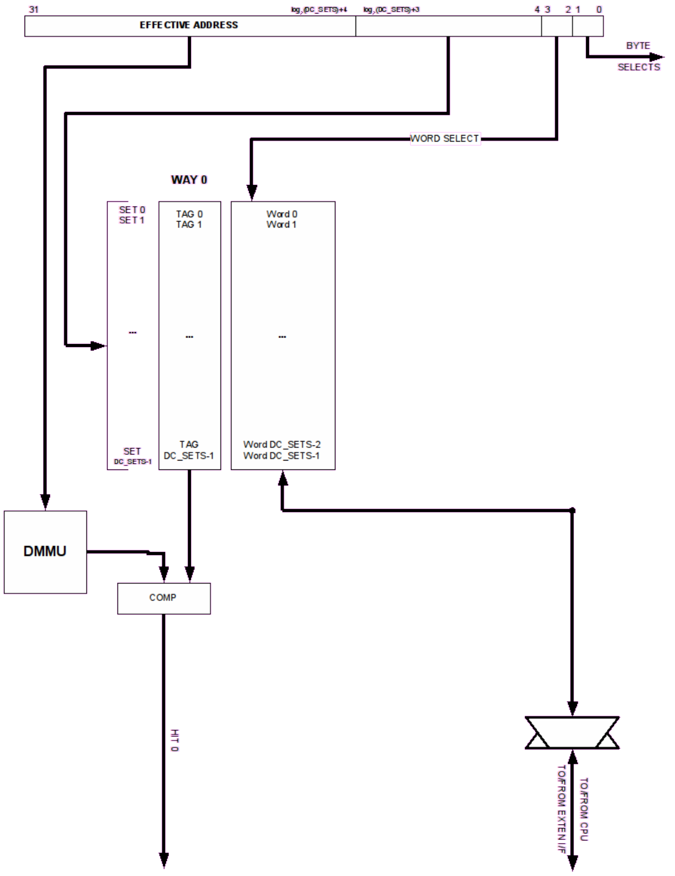
Features:

- data cache is separate from instruction cache (Harvard architecture)
- data cache implements a least-recently used (LRU) replacement algorithm within each set
- the cache directory is physically addressed. The physical address tag is stored in the cache directory
- write-through or write-back operation
- entire cache can be disabled, lines invalidated, flushed or forced to be written back, by writing to cache special purpose registers

On a miss, and appropriate conditions, the cache line is filled or emptied (written back) with 16-byte bursts. The burst fill is performed as a critical-word-first operation; the critical word is simultaneously written to the cache and forwarded to the requesting unit, thus minimizing stalls due to cache fill latency. Data cache provides storage for cache tags and performs cache line replacement function.

Data cache is tightly coupled to external interface to allow efficient access to the system memory controller.

The data cache supplies data to the GPRs by means of a 32-bit interface to the load/store unit. The LSU provides all logic required to calculate effective addresses, handles data alignment to and from the data cache, and provides sequencing for load and store operations. Write operations to the data cache can be performed on a byte, half-word or word basis.


31
log2(DC_SETS)+4
log2(DC_SETS)+3
4 3
2 1
0
EFFECTIVE ADDRESS
BYTE
SELECTS
WORD SELECT
WAY 0
SET 0
SET 1
...
SET DC_SETS-1
TAG 0
TAG 1
...
TAG DC_SETS-1
Word 0
Word 1
...
Word DC_SETS-2
Word DC_SETS-1
DMMU
COMP
HIT 0
TO/FROM EXTEN I/F
TO/FROM CPU

Each line contains four contiguous words from memory that are loaded from a cache line aligned boundary. As a result, cache lines are aligned with page boundaries.

## 2.3 Instruction Cache

The default configuration of OR1200 instruction cache is 8-Kbyte, 1-way direct mapped instruction cache, which allows rapid core access to instructions. However instruction cache can be configured according to Table 2.2.

Table 2.2: Possible Instruction Cache Configurations of OR1200

| | **Direct mapped** |
|---|---|
| 16B/line, 32 lines, 1 way | 512B |
| 16B/line, 256 lines, 1 way | 4KB |
| 16B/line, 512 lines, 1 way | **8KB (Default)** |
| 16B/line, 1024 lines, 1 way | 16KB |
| 32B/line, 1024 lines, 1 way | 32KB |

Features:

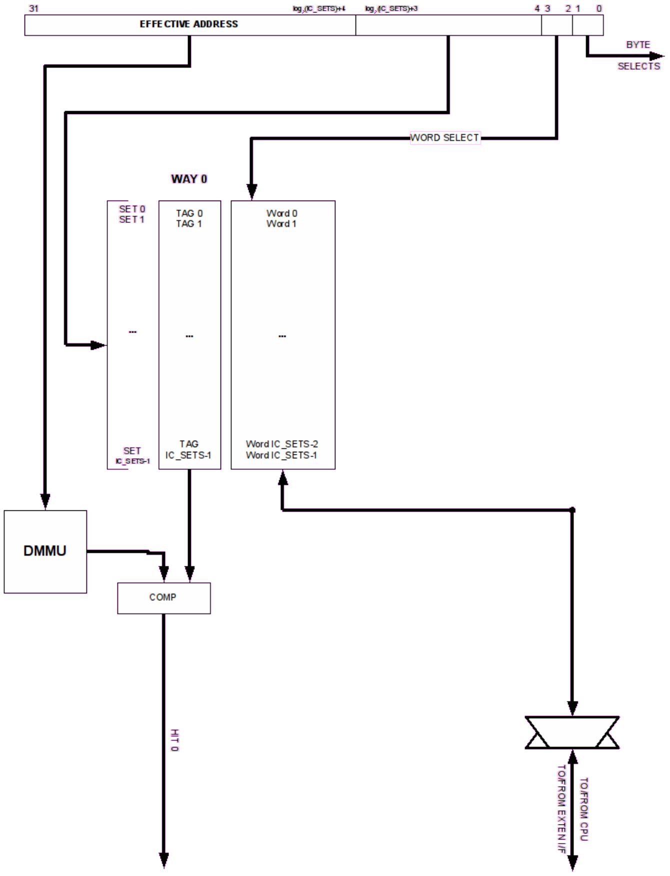
- instruction cache is separate from data cache (Harvard architecture)
- instruction cache implements a least-recently used (LRU) replacement algorithm within each set LRU

- the cache directory is physically addressed. The physical address tag is stored in the cache directory
- it can be disabled or invalidated by writing to cache special purpose registers

On a miss, the cache is filled in with 16-byte bursts. The burst fill is performed as a critical-word-first operation; the critical word is simultaneously written to the cache and forwarded to the requesting unit, thus minimizing stalls due to cache fill latency. Instruction cache provides storage for cache tags and performs cache line replacement function.

Instruction cache is tightly coupled to external interface to allow efficient access to the system memory controller.

The instruction cache supplies instructions to the instruction sequencer by means of a 32-bit interface to the instruction fetch subunit. The instruction fetch subunit provides all logic required to calculate effective addresses.


31
log2(IC_SETS)+4
log2(IC_SETS)+3
4 3 2 1 0
EFFECTIVE ADDRESS
BYTE SELECTS
WORD SELECT
WAY 0
SET 0
SET 1
...
SET IC_SETS-1
TAG 0
TAG 1
...
TAG IC_SETS-1
Word 0
Word 1
...
Word IC_SETS-2
Word IC_SETS-1
DMMU
COMP
HIT 0
TO/FROM EXTEN I/F
TO/FROM CPU

Each line contains four contiguous words from memory that are loaded from a line-size aligned boundary. As a result, cache lines are aligned with page boundaries.

## 2.4 Data MMU

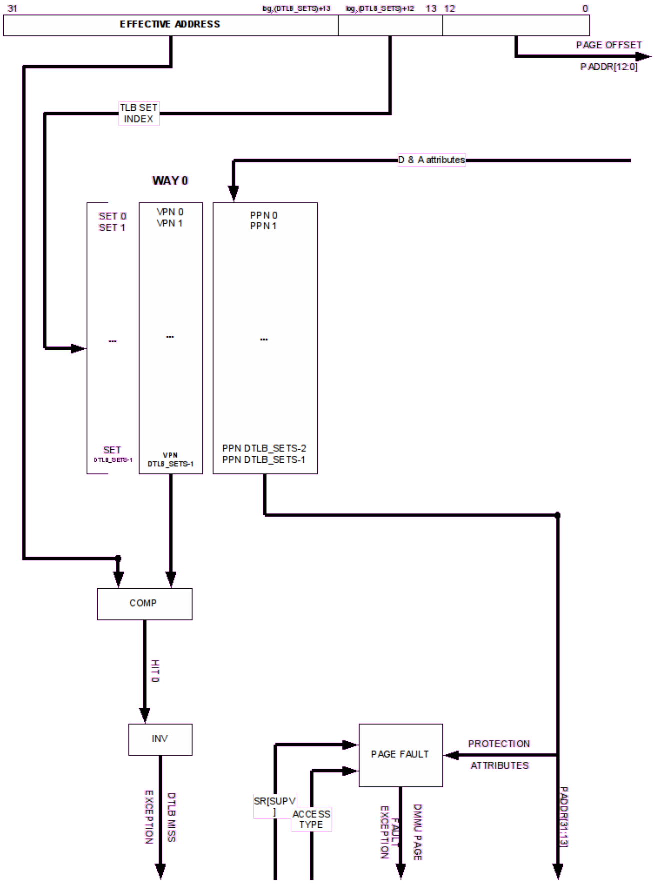
The OR1200 implements a virtual memory management scheme that provides memory access protection and effective-to-physical address translation. Protection granularity is as defined by OpenRISC 1000 architecture - 8-Kbyte and 16-Mbyte pages.

Table 2.3: Possible Data TLB Configurations of OR1200

| | **Direct mapped** |
|---|---|
| 16 entries per way | 16 DTLB entries |
| 32 entries per way | 32 DTLB entries |
| 64 entries per way | **64 DTLB entries (default)** |
| 128 entries per way | 128 DTLB entries |

Features:

- data MMU is separate from instruction MMU
- page size 8-Kbyte
- comprehensive page protection scheme
- direct mapped hash based translation lookaside buffer (DTLB) with the default of 1 way and the following features:
  - miss and fault exceptions
  - software tablewalk
  - high performance because of hashed based design
  - variable number DTLB entries with default of 64 per each way


31
log2(DTLB_SETS)+13
log2(DTLB_SETS)+12
13
12
0
EFFECTIVE ADDRESS
PAGE OFFSET
PADDR[12:0]
TLB SET INDEX
D & A attributes
WAY 0
SET 0
SET 1
...
SET DTLB_SETS-1
VPN 0
VPN 1
...
VPN DTLB_SETS-1
PPN 0
PPN 1
...
PPN DTLB_SETS-2
PPN DTLB_SETS-1
COMP
HIT 0
INV
EXCEPTION
DTLB MISS
SR[SUPV]
ACCESS TYPE
PAGE FAULT
PROTECTION
ATTRIBUTES
EXCEPTION
FAULT
DMMU PAGE
PADDR[31:13]

The MMU hardware supports two-level software tablewalk.

## 2.5 Instruction MMU

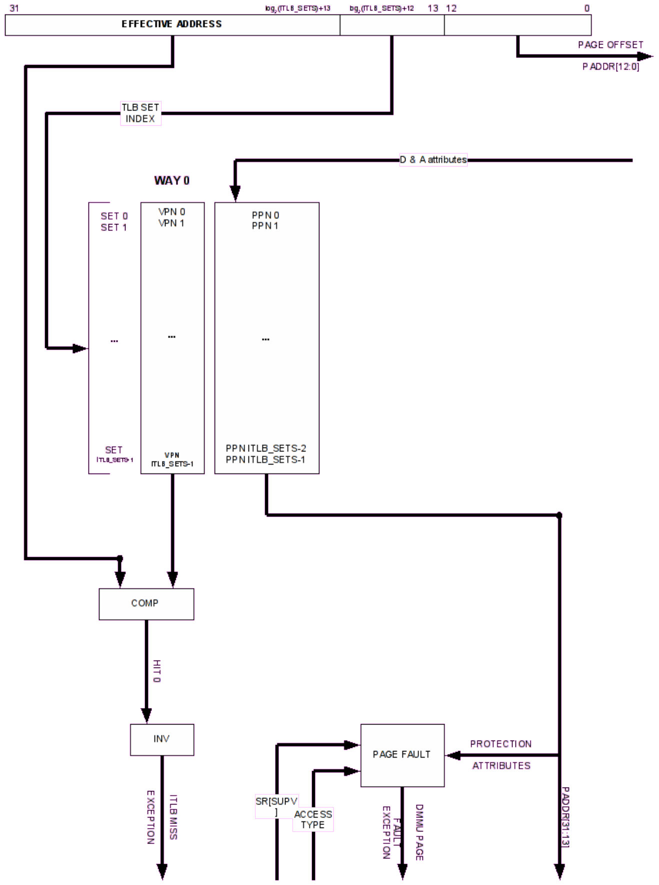
The OR1200 implements a virtual memory management scheme that provides memory access protection and effective-to-physical address translation. Protection granularity is as defined by OpenRISC 1000 architecture - 8-Kbyte and 16-Mbyte pages.

Table 2.4: Possible Instruction TLB Configurations of OR1200

| | **Direct mapped** |
|---|---|
| 16 entries per way | 16 DTLB entries |
| 32 entries per way | 32 DTLB entries |
| 64 entries per way | **64 DTLB entries (default)** |
| 128 entries per way | 128 DTLB entries |

Features:

- instruction MMU is separate from data MMU
- pages size 8-Kbyte
- comprehensive page protection scheme
- 1 way direct-mapped hash based translation lookaside buffer (ITLB) with the following features:
  - miss and fault exceptions
  - software tablewalk
  - high performance because of hashed based design
  - Variable number of ITLB entries with default of 64 entries per way


31
log2(ITLB_SETS)+13
log2(ITLB_SETS)+12
13
12
0
EFFECTIVE ADDRESS
PAGE OFFSET
P ADDR[12:0]
TLB SET INDEX
D & A attributes
WAY 0
SET 0
SET 1
...
SET ITLB_SETS-1
VPN 0
VPN 1
...
VPN ITLB_SETS-1
PPN 0
PPN 1
...
PPN ITLB_SETS-2
PPN ITLB_SETS-1
COMP
HIT 0
INV
EXCEPTION
ITLB MISS
SR[SUPV]
ACCESS TYPE
PAGE FAULT
PROTECTION
ATTRIBUTES
FAULT EXCEPTION
DMMU PAGE
PADDR[31:13]

The MMU hardware supports two-level software tablewalk.

## 2.6 Programmable Interrupt Controller

The interrupt controller receives interrupts from external sources and forwards them as low or high priority interrupt exception to the CPU core.

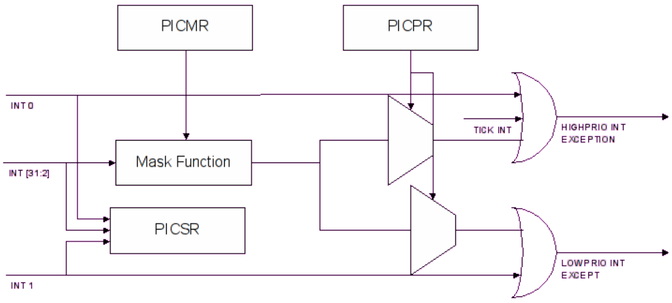


Figure 2.3: Block Diagram of the Interrupt Controller

Programmable interrupt controller has three special-purpose registers and 32 interrupt inputs. Interrupt input 0 and 1 are always enabled and connected to high and low priority interrupt input, respectively.

30 other interrupt inputs can be masked and assigned low or high priority through programming special-purpose registers.

## 2.7 Tick Timer

OR1200 implements tick timer facility. Basically this is a timer that is clocked by RISC clock and is used by the operating system to precisely measure time and schedule system tasks.

OR1200 precisely follow architectural definition of the tick timer facility:

- Maximum timer count of 2^32 clock cycles
- Maximum time period of 2^28 clock cycles between interrupts
- Maskable tick timer interrupt
- Single run, restartable or continues timer

Tick timer operates from independent clock source so that doze power management mode can be implemented.

## 2.8 Power Management Support

To optimize power consumption, the OR1200 provides low-power modes that can be used to dynamically activate and deactivate certain internal modules.

OR1200 has three major features to minimize power consumption:

- Slow and Idle Modes (SW controlled clock freq reduction)
- Doze and Sleep Modes (interrupt wake-up)

Table 2.5: Power Consumption

| **Power Minimization Feature** | **Approx Power Consumption Reduction** |
|---|---|
| Slow and Idle mode | 2x - 10x |
| Doze mode | 100x |
| Sleep mode | 200x |
| Dynamic clock gating | N/A |

Slow down mode takes advantage of the low-power dividers in external clock generation circuitry to enable full functionality, but at a lower frequency so that a power consumption is reduced. PMR[SDF] 4 bits are broadcasted on pm_clksd and external clock generation for the RISC should adapt RISC clock frequency according to the value on pm_clksd.

When software initiates the doze mode, software processing on the core suspends. The clocks to the RISC internal modules are disabled except to the tick timer. However any other on-chip blocks can continue to function as normal. The OR1200 will leave doze mode and enter normal mode when a pending interrupt occurs.

In sleep mode, all OR1200 internal units are disabled and clocks gated. Optionally implementation may choose to lower the operating voltage of the OR1200 core. The OR1200 should leave sleep mode and enter normal mode when a pending interrupt occurs.

Dynamic Clock gating (unit clock gating on clock by clock basis) is not supported by OR1200.

## 2.9 Debug unit

Debug unit assists software developers to debug their systems. It provides support only for basic debugging and does not have support for more advanced debug features of OpenRISC 1000 architecture such as watchpoints, breakpoints and program-flow control registers.

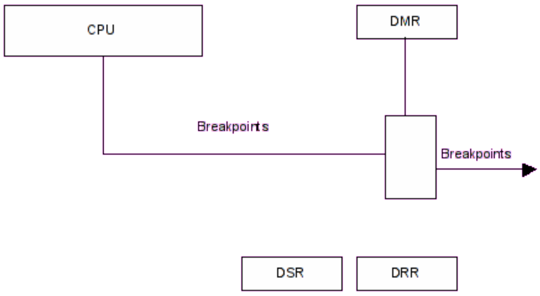


Figure 2.4: Block Diagram of Debug Unit

Watchpoints and breakpoints are events triggered by program- or data-flow matching the conditions programmed in the debug registers. Breakpoints unlike watchpoints also suspend execution of the current program-flow and start breakpoint exception.

## 2.10 Clocks & Reset

The OR1200 core has a clock input each for the instruction and data Wishbone interface logic, and for the CPU core. Clock input clk_cpu clocks everything inside the Wishbone interfaces. Data Wishbone interface is clocked by dwb_clk_i, instruction Wishbone interface is clocked by iwb_clk_i.

OR1200 has asynchronous reset signal. Reset signal rst, when asserted high, immediately resets all flip-flops inside OR1200. When deasserted, OR1200 will start reset exception.

## 2.11 WISHBONE Interfaces

Two WISHBONE interfaces connect OR1200 core to external peripherals and external memory subsystem. They are WISHBONE SoC Interconnection specification Rev. B3 compliant. The implementation implements a 32-bit bus width and does not support other bus widths.

Wishbone registered-feedback incrementing burst accesses occur when not disabled, and cache lines are filled. The burst size (beats) is determined by the cache line size.

WISHBONE
COMPATIBLE

# Chapter 3

# Operation

This section describes the operation of the OR1200 core. For operations that pertain to the architectural definitions, see [or1000_manual].

## 3.1 Reset

OR1200 has one asynchronous reset signal that can be used by a soft and hard reset on a higher system hierarchy levels.

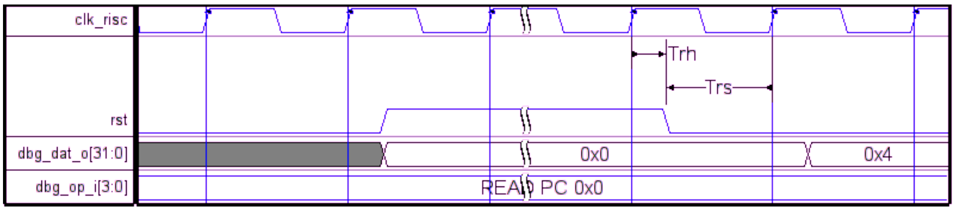


Figure 3.1: Power-Up and Reset Sequence

Figure 3.1 shows how asynchronous reset is applied after powering up the OR1200 core. Reset is connected to asynchronous reset of almost all flip-flops inside RISC core. Special care must be taken to ensure hold and setup times of all flip-flops compared to main RISC clock.

If system implements gated clocks, then clock gating can be used to ensure proper reset timing.

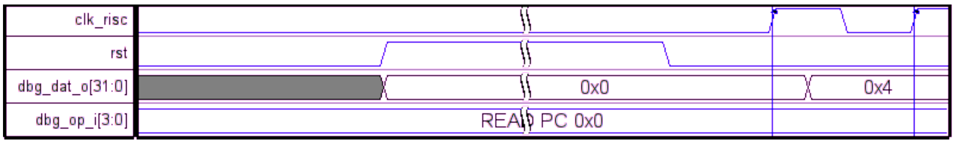


Figure 3.2: Power-Up and Reset Sequence w/ Gated Clock

The address the PC assumes at hard reset (assertion of external reset signal) is definable at synthesis time, via the OR1200_BOOT_ADR define. This is not to be confused with the ability to set the exception prefix address with the EPH bit.

## 3.2 CPU/FPU/DSP

CPU/FPU/DSP is implementation of the 32-bit part of the OpenRISC 1000 architecture and only a subset of all features is implemented.

### 3.2.1 Instructions

The following table lists the instructions implemented in the OR1200. Those optionally implemented are indicated as such.

Table 3.1: Instructions implemented in OR1200

| Instruction mnemonic | Optional | Instruction mnemonic | Optional | Instruction mnemonic | Optional | Instruction mnemonic | Optional |
|---|---|---|---|---|---|---|---|
| l.add | | l.lws | | l.sfges | | l.trap | |
| l.addc | Yes | l.lwz | | l.sfgeu | | l.xor | |
| l.addi | | l.mac | Yes | l.sfgts | | l.xori | |
| l.and | | l.maci | Yes | l.sfgtu | | lf.add.s | Yes |
| l.andi | | l.macrc | Yes | l.sfleu | | lf.div.s | Yes |
| l.bf | | l.mfspr | | l.sflts | | lf.ftoi.s | Yes |
| l.bnf | | l.movhi | | l.sfltu | | lf.itof.s | Yes |
| l.div | Yes | l.msb | Yes | l.sfne | | lf.mul.s | Yes |
| l.ff1 | Yes | l.mtspr | | l.sh | | lf.sfeq.s | Yes |
| l.fl1 | Yes | l.mul | Yes | l.sll | | lf.sfge.s | Yes |
| l.j | | l.muli | Yes | l.slli | | lf.sfgt.s | Yes |
| l.jal | | l.nop | | l.sra | | lf.sfle.s | Yes |
| l.jalr | | l.or | | l.srai | | lf.sflt.s | Yes |
| l.jr | | l.ori | | l.srl | | lf.sfne.s | Yes |
| l.lbs | | l.rfe | | l.srli | | lf.sub.s | Yes |
| l.lbz | | l.rori | | l.sub | Yes | | |
| l.lhs | | l.sb | | l.sw | | | |
| l.lhz | | l.sfeq | | l.sys | | | |

For a complete description of each instruction's format refer to [or1000_manual].

### 3.2.2 Instruction Unit

Instruction unit generates instruction fetch effective address and fetches instructions from instruction cache. Each clock cycle one instruction can be fetched. Instruction fetch EA is further translated into physical address by IMMU.

### 3.2.3 General-Purpose Registers

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

GPRs can be also read and written through development interface.

### 3.2.4 Load/Store Unit

LSU can execute one load instruction every two clock cycles assuming load instruction have a hit in the data cache. Execution of store instructions takes one clock cycle assuming they have a hit in the data cache.

LSU performs calculation of the load/store effective address. EA is further translated into physical address by DMMU.

Load/store effective address and load and store data can be also accessed through development interface.

### 3.2.5 Integer Execution Pipeline

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

- Arithmetic instructions
- Compare instructions
- Logical instructions
- Rotate and shift instructions

Table 3.2: Execution Time of Integer Instructions

| **Instruction Group** | **Clock Cycles to Execute** |
|---|---|
| Arithmetic except Multiply/Divide | 1 |
| Multiply | 3 |
| Divide | 32 |
| Compare | 1 |
| Logical | 1 |
| Rotate and Shift | 1 |
| Others | 1 |

Table 3.2 lists execution times for instructions executed by integer execution pipeline. Most instructions are executed in one clock cycle.

Integer multiply can be either serial or parallel implementations. Serial operations require one clock cycle per bit of operand, which is 32-cycles on the OR1200. At present no synthesis tools support division operators, and so the serial option must be used.

### 3.2.6 MAC Unit

MAC unit executes l.mac instructions. MAC unit implements 32x32 fully pipelined multiplier and 48-bit accumulator. MAC unit can accept one new l.mac instruction each clock cycle.

Care should be taken when executing l.macrc (MAC read and clear) too soon after the final l.mac instruction as the operation may still be underway and the result will not be valid in time. It is recommended at least 3 other instructions (or just l.nops) are inserted between the final l.mac and l.macrc.

### 3.2.7 Floating Point Unit

The floating point unit has a mechanism to stall the processor pipeline until processing has completed.

The following table indicates the number of cycles per operation

Table 3.3: Execution time of floating point instructions

| **Operation** | **Cycles** |
|---|---|
| Add/subtract | 10 |
| Multiply | 38 |
| Divide | 37 |
| Compare | 2 |
| Convert | 7 |

### 3.2.8 System Unit

System unit implements system control and status special-purpose registers and executes all l.mtspr/l.mfspr instructions.

### 3.2.9 Exceptions

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 faulting instruction is saved in EPCR registers and the machine state is saved to ESR registers

Table 3.4: List of Implemented Exceptions

| **Exception Type** | **Vector Offset** | **Causing Conditions** |
|---|---|---|
| Reset | 0x100 | Caused by reset. |
| Bus Error | 0x200 | Caused by an attempt to access invalid physical address. |
| Data Page Fault | 0x300 | Generated artificially by DTLB miss exception handler when no matching PTE found in page tables or page protection violation for load/store operations. |
| Instruction Page Fault | 0x400 | Generated artificially by ITLB miss exception handler when no matching PTE found in page tables or page protection violation for instruction fetch. |
| Low Priority External Interrupt | 0x500 | Low priority external interrupt asserted. |
| Alignment | 0x600 | Load/store access to naturally not aligned location. |
| Illegal Instruction | 0x700 | Illegal instruction in the instruction stream. |
| High Priority External Interrupt | 0x800 | High priority external interrupt asserted. |
| D-TLB Miss | 0x900 | No matching entry in DTLB (DTLB miss). |
| I-TLB Miss | 0xA00 | No matching entry in ITLB (ITLB miss). |
| System Call | 0xC00 | System call initiated by software. |
| Floating point exception | 0xD00 | FP operation caused flags in FPCSR to become set. |
| Trap | 0xE00 | Trap instruction was decoded |

The OR1200 exception support does not include support for range exceptions or fast context switching.

## 3.3 Data Cache Operation

### 3.3.1 Data Cache Load/Store Access

Load/store unit requests data from the data cache and stores them into the general-purpose register file and forwards them to integer execution units. Therefore LSU is tightly coupled with the data cache.

If there is no data cache line miss nor DTLB miss, load operations take two clock cycles to execute and store operations take one clock cycle to execute. LSU does all the data alignment work.

Data can be written to the data cache on a word, half-word or byte basis. Since data cache only operates in write-through mode, all writes are immediately written back to main memory or to the next level of caches.

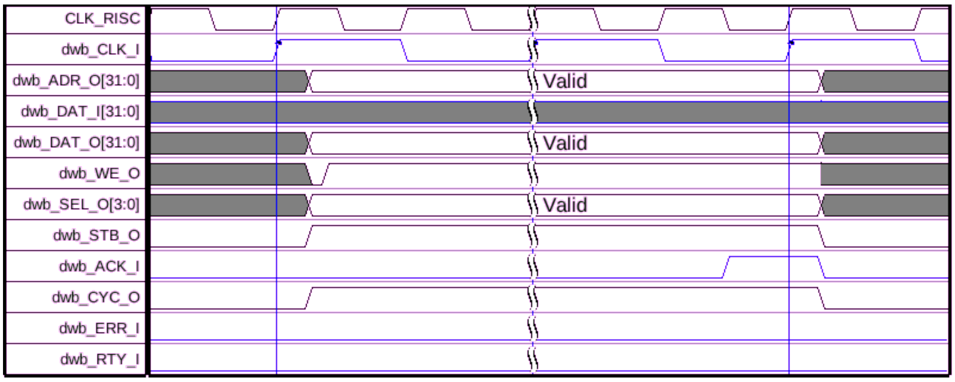

CLK_RISC
dwb_CLK_I
dwb_ADR_O[31:0]
Valid
dwb_DAT_I[31:0]
dwb_DAT_O[31:0]
Valid
dwb_WE_O
dwb_SEL_O[3:0]
Valid
dwb_STB_O
dwb_ACK_I
dwb_CYC_O
dwb_ERR_I
dwb_RTY_I

Figure 3.3: WISHBONE Write Cycle

Figure 3.3 shows how a write-through cycle on data WISHBONE interface is performed when a store instruction hits in the data cache. If `dwb_ERR_I` or `dwb_RTY_I` is asserted instead of usual `dwb_ACK_I`, bus error exception is invoked.

### 3.3.2 Data Cache Line Fill Operation

When executing load instruction and a cache miss occurs, depending on whether the cache uses write-through or write-back strategy and the line is clean or invalid, a 4 beat sequential read burst with critical word first is performed. If the strategy is write-back and the line is dirty, the line is first written back to memory. The critical word is forwarded to the load/store unit to minimize performance loss because of the cache miss.

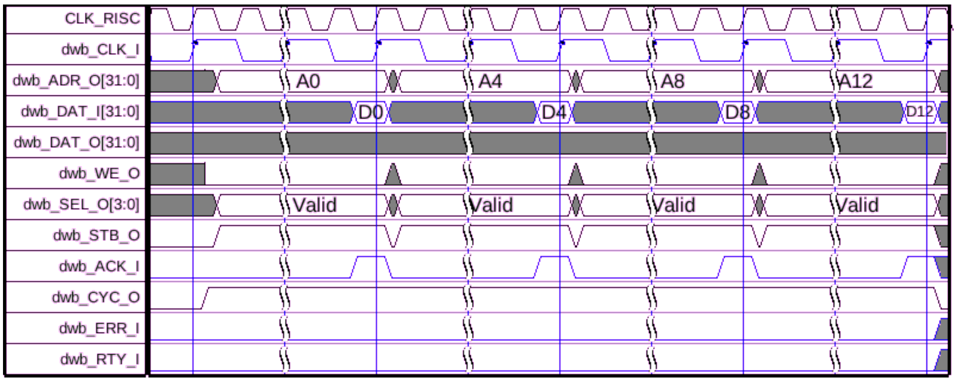

CLK_RISC
dwb_CLK_I
dwb_ADR_O[31:0]
A0
A4
A8
A12
dwb_DAT_I[31:0]
D0
D4
D8
D12
dwb_DAT_O[31:0]
dwb_WE_O
dwb_SEL_O[3:0]
Valid
Valid
Valid
Valid
dwb_STB_O
dwb_ACK_I
dwb_CYC_O
dwb_ERR_I
dwb_RTY_I

Figure 3.4: WISHBONE Block Read Cycle

Figure 3.4 shows how a cache line is read in WISHBONE read block cycle composed out of four read transfers. If `dwb_ERR_I` or `dwb_RTY_I` is asserted instead of usual `dwb_ACK_I`, bus error exception is invoked.

When executing a store instruction with the cache in write-through strategy, and a cache miss occurs, the write is simply put on the bus and no caching occurs. If it is a miss and the cache is in write back strategy and the line is valid and clean or invalid, a 4 beat sequential read burst to fill the line is performed, and the the write to cache occurs. If storing and a cache miss occurs, and the desired line is valid and dirty, it is first written back to memory before the desired line is read.

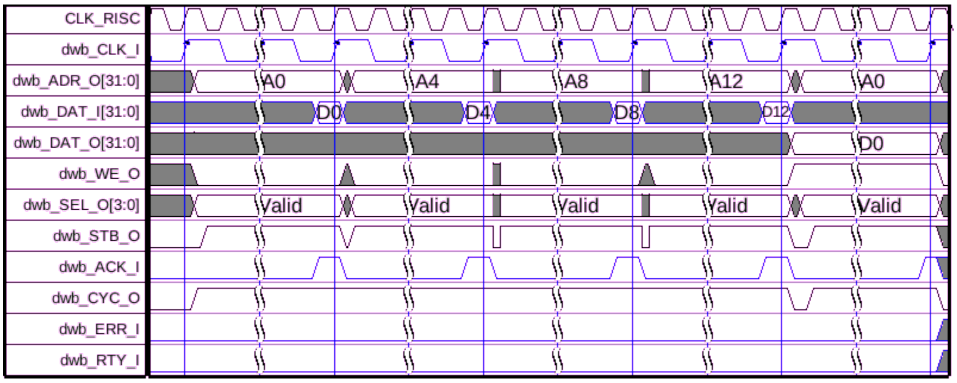


Figure 3.5: WISHBONE Block Read/Write Cycle

Figure 3.5 shows how a cache line is read in WISHBONE read block cycle followed by a write transfer. If `dwb_ERR_I` or `dwb_RTY_I` is asserted instead of usual `dwb_ACK_I`, bus error exception is invoked.

### 3.3.3 Cache/Memory Coherency

Data cache in OR1200 operates in either write-through or write-back mode, definable at synthesis time, for default use, and runtime when DMMU is used. There is currently no coherency support between local data cache and caches of other processors.

### 3.3.4 Data Cache Enabling/Disabling

Data cache is disabled at power up. Entire data cache can be enabled by setting bit SR[DCE] to one. Before data cache is enabled, it must be invalidated.

### 3.3.5 Data Cache Invalidation

Data cache in OR1200 does not support invalidation of entire data cache. Normal procedure to invalidate entire data cache is to cycle through all data cache lines and invalidate each line separately.

### 3.3.6 Data Cache Locking

Data cache implements way locking bits in data cache control register DCCR. Bits LWx lock individual ways when they are set to one.

### 3.3.7 Data Cache Line Prefetch

Data cache line prefetch is optional in the OpenRISC 1000 architecture and is not implemented in OR1200.

### 3.3.8 Data Cache Line Flush

Operation is performed by writing effective address to the DCBFR register.

When a cache line is valid and clean, or the cache is in write-through strategy, the line is invalidated and no write-back occurs.

### 3.3.9 Data Cache Line Invalidate

Data cache line invalidate invalidates a single data cache line. Operation is performed by writing effective address to the DCBIR register. If cache is in write-back strategy, it is best to use the line flush function.

### 3.3.10 Data Cache Line Write-back

Operation is performed by writing effective address to the DCBWR register.

If cache is in write-through strategy, this operation is ignored as no lines will be cached and dirty, capable of being written back.

### 3.3.11 Data Cache Line Lock

Locking of individual data cache lines is not implemented in OR1200.

### 3.3.12 Data Cache inhibit with address bit 31 set

If DMMU is disabled, by default all addresses with bit 31 of the address asserted high will cause the data cache to be inhibited, meaning no reads or writes are cached.

If the DMMU is enabled, it is possible for any address to be inhibited or not, and in these modes the cache behaves accordingly.

## 3.4 Instruction Cache Operation

### 3.4.1 Instruction Cache Instruction Fetch Access

Instruction unit requests instruction from the instruction cache and forwards them to the instruction queue inside instruction unit. Therefore instruction unit is tightly coupled with the instruction cache.

If there is no instruction cache line miss nor ITLB miss, instruction fetch operation takes one clock cycle to execute.

Instruction cache cannot be explicitly modified like data cache can be with store instructions.

### 3.4.2 Instruction Cache Line Fill Operation

On a cache miss, a 4 beat sequential read burst with critical word first is performed. Critical word is forwarded to the instruction unit to minimize performance loss because of the cache miss.

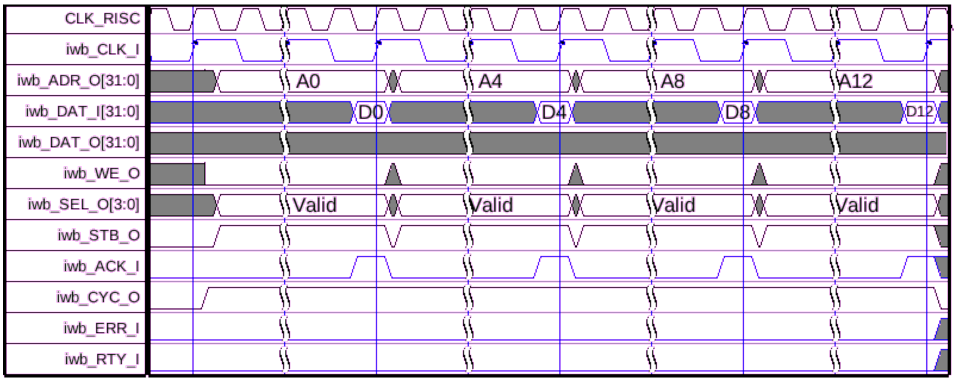


Figure 3.6: WISHBONE Block Read Cycle

Figure 3.6 shows how a cache line is read in WISHBONE read block cycle composed out of four read transfers. If `iwb_ERR_I` or `iwb_RTY_I` is asserted instead of usual `dwb_ACK_I`, bus error exception is invoked.

### 3.4.3 Cache/Memory Coherency

OR1200 is not intended for use in multiprocessor environments. Therefore no support for coherency between local instruction cache and caches of other processors or main memory is implemented.

### 3.4.4 Instruction Cache Enabling/Disabling

Instruction cache is disabled at power up. Entire instruction cache can be enabled by setting bit SR[ICE] to one. Before instruction cache is enabled, it must be invalidated.

### 3.4.5 Instruction Cache Invalidation

Instruction cache in OR1200 does not support invalidation of entire instruction cache. Normal procedure to invalidate entire instruction cache is to cycle through all instruction cache lines and invalidate each line separately.

### 3.4.6 Instruction Cache Locking

Instruction cache implements way locking bits in instruction cache control register ICCR. Bits LWx lock individual ways when they are set to one.

### 3.4.7 Instruction Cache Line Prefetch

Instruction cache line prefetch is optional in the OpenRISC 1000 architecture and is not implemented in OR1200.

### 3.4.8 Instruction Cache Line Invalidate

Instruction cache line invalidate invalidates a single instruction cache line. Operation is performed by writing effective address to the ICBIR register.

### 3.4.9 Instruction Cache Line Lock

Locking of individual instruction cache lines is not implemented in OR1200.

## 3.5 Data MMU

### 3.5.1 Translation Disabled

Load/store address translation can be disabled by clearing bit SR[DME]. If translation is disabled, then physical address used to access data cache and optionally provided on `dwb_ADDR_O`, is the same as load/store effective address.

### 3.5.2 Translation Enabled

Load/store address translation can be enabled by setting bit SR[DME]. If translation is enabled, it provides load/store effective address to physical address translation and page protection for memory accesses.

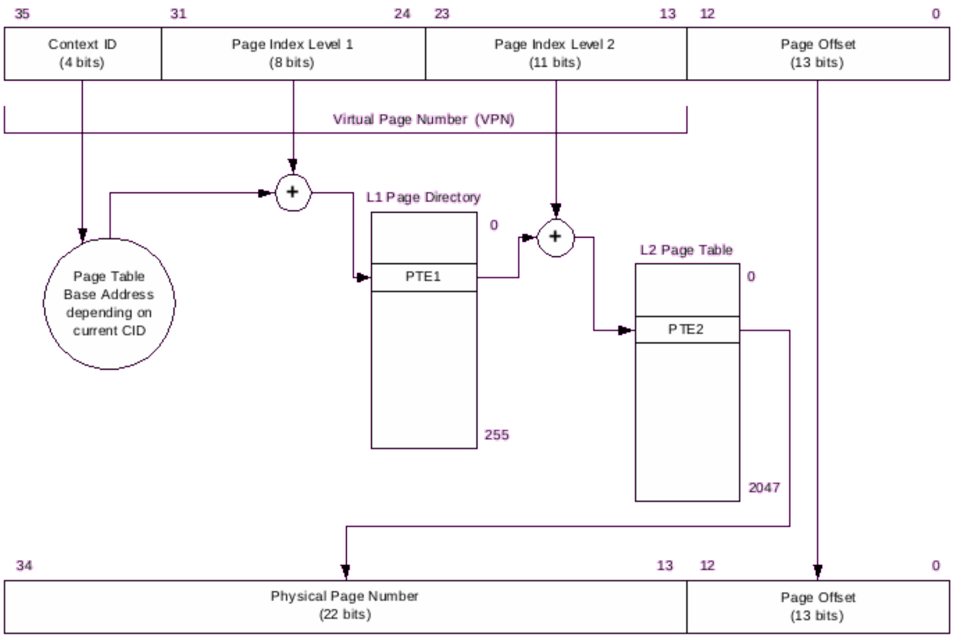

35
31
24
23
13
12
0
Context ID
(4 bits)
Page Index Level 1
(8 bits)
Page Index Level 2
(11 bits)
Page Offset
(13 bits)
Virtual Page Number (VPN)
+
L1 Page Directory
0
PTE1
255
Page Table
Base Address
depending on
current CID
+
L2 Page Table
0
PTE2
2047
34
13
12
0
Physical Page Number
(22 bits)
Page Offset
(13 bits)

Figure 3.7: 32-bit Address Translation Mechanism using Two-Level Page Table

In OR1200 case, page tables must be managed by operating system's virtual memory management subsystem. Figure 3.7 shows address translation using two-level page table. Refer to [or1000_manual] for one-level page table address translation as well as for details about address translation and page table content.

### 3.5.3 DMMUCR and Flush of Entire DTLB

DMMUCR is not implemented in OR1200. Therefore page table base pointer (PTBP) must be stored in software variable. Flush of entire DTLB must be performed by software flush of every DTLB entry separately. Software flush is performed by manually writing bits from the TLB entries back to PTEs.

### 3.5.4 Page Protection

After a virtual address is determined to be within a page covered by the valid PTE, the access is validated by the memory protection mechanism. If this protection mechanism prohibits the access, a data page fault exception is generated.

The memory protection mechanism allows selectively granting read access and write access for both supervisor and user modes. The page protection mechanism provides protection at all page level granularities.

Table 3.5: Protection Attributes for Load/Store Accesses

| **Protection attribute** | **Meaning** |
|---|---|
| DTLBWyTR[SREx] | Enable load operations in supervisor mode to the page. |
| DTLBWyTR[SWEx] | Enable store operations in supervisor mode to the page. |
| DTLBWyTR[UREx] | Enable load operations in user mode to the page. |
| DTLBWyTR[UWEx] | Enable store operations in user mode to the page. |

Table 3.5 lists page protection attributes defined in DTLBWyTR pregister. For the individual page appropriate strategy out of seven possible strategies programmed with the PPI field of the PTE. Because OR1200 does not implement DMMUPR, translation of PTE[PPI] into suitable set of protection bits must be performed by software and written into DTLBWyTR.

### 3.5.5 DTLB Entry Reload

OR1200 does not implement DTLB entry reloads in hardware. Instead software routine must be used to search page table for correct page table entry (PTE) and copy it into the DTLB. Software is responsible for maintaining accessed and dirty bits in the page tables.

When LSU computes load/store effective address whose physical address is not already cached by DTLB, a DTLB miss exception is invoked.

DTLB reload routine must load the correct PTE to correct DTLBWyMR and DTLBWyTR register from one of possible DTLB ways.

### 3.5.6 DTLB Entry Invalidation

Special-purpose register DTLBEIR must be written with the effective address and corresponding DTLB entry will be invalidated in the local DTLB.

### 3.5.7 Locking DTLB Entries

Since all DTLB entry reloads are performed in software, there is no hardware locking of DTLB entries. Instead it is up to the software reload routine to avoid replacing some of the entries if so desired.

### 3.5.8 Page Attribute - Dirty (D)

Dirty (D) attribute is not implemented in OR1200 DTLB. It is up to the operating system to generate dirty attribute bit with page protection mechanism.

### 3.5.9 Page Attribute - Accessed (A)

Accessed (A) attribute is not implemented in OR1200 DTLB. It is up to the operating system to generate accessed attribute bit with page protection mechanism.

### 3.5.10 Page Attribute - Weakly Ordered Memory (WOM)

Weakly ordered memory (WOM) attribute is not needed in OR1200 because all memory accesses are serialized and therefore this attribute is not implemented.

### 3.5.11 Page Attribute - Write-Back Cache (WBC)

Write-back cache (WBC) attribute is not implemented as the data cache cannot be configured at run time to be write-back enabled if write-through strategy was selected at synthesis-time.

### 3.5.12 Page Attribute - Caching-Inhibited (CI)

Caching-inhibited (CI) attribute is not implemented in OR1200 DTLB. Cached and uncached regions are divided by bit 30 of data effective address.

Table 3.6: Cached and uncached regions

| Effective Address | Region |
|---|---|
| 0x00000000 - 0x3FFFFFFF | Cached |
| 0x40000000 - 0x7FFFFFFF | Uncached |
| 0x80000000 - 0xBFFFFFFF | Cached |
| 0xC0000000 - 0xFFFFFFFF | Uncached |

Uncached accesses must be performed when I/O registers are memory mapped and all reads and writes must be always performed directly to the external interface and not to the data cache.

### 3.5.13 Page Attribute - Cache Coherency (CC)

Cache coherency (CC) attribute is not needed in OR1200 because it does not implement support for multiprocessor environments and because data cache operates only in write-through mode and therefore this attribute is not implemented.

## 3.6 Instruction MMU

### 3.6.1 Translation Disabled

Instruction fetch address translation can be disabled by clearing bit SR[IME]. If translation is disabled, then physical address used to access instruction cache and optionally provided on iwb_ADDR_O, is the same as instruction fetch effective address.

### 3.6.2 Translation Enabled

Instruction fetch address translation can be enabled by setting bit SR[IME]. If translation is enabled, it provides instruction fetch effective address to physical address translation and page protection for instruction fetch accesses.

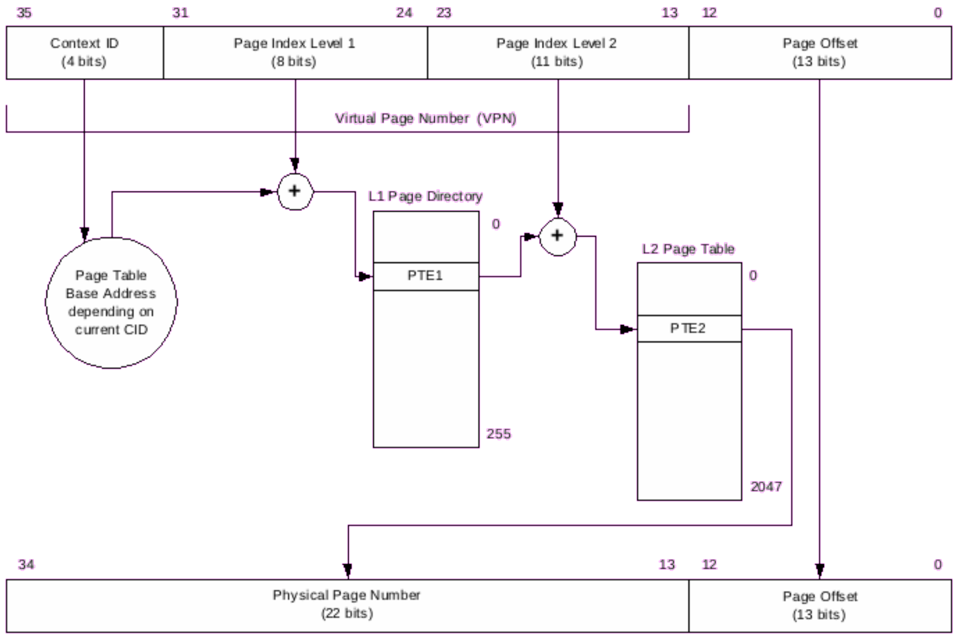

35
31
24
23
13
12
0
Context ID (4 bits)
Page Index Level 1 (8 bits)
Page Index Level 2 (11 bits)
Page Offset (13 bits)
Virtual Page Number (VPN)
+
L1 Page Directory
0
PTE1
255
+
L2 Page Table
0
PTE2
2047
Page Table Base Address depending on current CID
34
13
12
0
Physical Page Number (22 bits)
Page Offset (13 bits)

Figure 3.8: 32-bit Address Translation Mechanism using Two-Level Page Table

In OR1200 case, page tables must be managed by operating system s virtual memory management subsystem. Figure 3.8 shows address translation using two-level page table. Refer to [or1000_manual] for one-level page table address translation as well as for details about address translation and page table content.

### 3.6.3 IMMUCR and Flush of Entire ITLB

IMMUCR is not implemented in OR1200. Therefore page table base pointer (PTBP) must be stored in software variable. Flush of entire ITLB must be performed by software flush of every ITLB entry separately. Software flush is performed by manually writing bits from the TLB entries back to PTEs.

### 3.6.4 Page Protection

After a virtual address is determined to be within a page covered by the valid PTE, the access is validated by the memory protection mechanism. If this protection mechanism prohibits the access, an instruction page fault exception is generated.

The memory protection mechanism allows selectively granting execute access for both supervisor and user modes. The page protection mechanism provides protection at all page level granularities.

Table 3.7: Protection Attributes for Instruction Fetch Accesses

| **Protection attribute** | **Meaning** |
|---|---|
| ITLBWyTR[SXEx] | Enable execute operations in supervisor mode of the page. |
| ITLBWyTR[UXEx] | Enable execute operations in user mode of the page. |

Table 3.7 lists page protection attributes defined in ITLBWyTR pregister. For the individual page appropriate strategy out of seven possible strategies programmed with PPI field of the PTE. Because OR1200 does not implement IMMUPR, translation of PTE[PPI] into suitable set of protection bits must be performed by software and written into ITLBWyTR.

### 3.6.5 ITLB Entry Reload

OR1200 does not implement ITLB entry reloads in hardware. Instead software routine must be used to search page table for correct page table entry (PTE) and copy it into the ITLB. Software is responsible for maintaining accessed bit in the page tables.

When LSU computes instruction fetch effective address whose physical address is not already cached by ITLB, an ITLB miss exception is invoked.

ITLB reload routine must load the correct PTE to correct ITLBWyMR and ITLBWyTR register from one of possible ITLB ways.

### 3.6.6 ITLB Entry Invalidation

Special-purpose register ITLBEIR must be written with the effective address and corresponding ITLB entry will be invalidated in the local ITLB.

### 3.6.7 Locking ITLB Entries

Since all ITLB entry reloads are performed in software, there is no hardware locking of ITLB entries. Instead it is up to the software reload routine to avoid replacing some of the entries if so desired.

### 3.6.8 Page Attribute - Dirty (D)

Dirty (D) attribute resides in the PTE but it is not used by the IMMU.

### 3.6.9 Page Attribute - Accessed (A)

Accessed (A) attribute is not implemented in OR1200 ITLB. It is up to the operating system to generate accessed attribute bit with page protection mechanism.

### 3.6.10 Page Attribute - Weakly Ordered Memory (WOM)

Weakly ordered memory (WOM) attribute is not needed in OR1200 because all instruction fetch accesses are serialized and therefore this attribute is not implemented.

### 3.6.11 Page Attribute - Write-Back Cache (WBC)

Write-back cache (WBC) attribute resides in the PTE but it is not used by the IMMU.

### 3.6.12 Page Attribute - Caching-Inhibited (CI)

Caching-inhibited (CI) attribute is not implemented in OR1200 ITLB. Cached and uncached regions are divided by bit 30 of instruction effective address.

Table 3.8: Cached and uncached regions

| Effective Address | Region |
|---|---|
| 0x00000000 - 0x3FFFFFFF | Cached |
| 0x40000000 - 0x7FFFFFFF | Uncached |
| 0x80000000 - 0xBFFFFFFF | Cached |
| 0xC0000000 - 0xFFFFFFFF | Uncached |

### 3.6.13 Page Attribute - Cache Coherency (CC)

Cache coherency (CC) attribute resides in the PTE but it is not used by the IMMU.

## 3.7 Programmable Interrupt Controller

PICMR special-purpose register is used to mask or unmask up to 30 programmable interrupt sources. PICPR special-purpose register is used to assign low or high priority to maximum of 30 interrupt sources.

PICSR special-purpose register is used to determine status of each interrupt input. Bits in PICSR represent status of the interrupt inputs and the actual interrupt must be cleared in the device that is the source of a pending interrupt.

The PIC implementation in the OR1200 differs from the architecture specification. The PIC instead offers a latched level-sensitive interrupt.

Once an interrupt line is latched (i.e. its value appears in PICSR), no new interrupts can be triggered for that line until its bit in PICSR is cleared. The usual sequence for an interrupt handler is then as follows.

1. Peripheral asserts interrupt, which is latched and triggers handler.

2. Handler processes interrupt.
3. Handler notifies peripheral that the interrupt has been processed (typically via a memory mapped register).
4. Peripheral deasserts interrupt.
5. Handler clears corresponding bit in PICSR and returns.

It is assumed that the peripheral will de-assert its interrupt promptly (within 1-2 cycles). Otherwise on exiting the interrupt handler, having cleared PICSR, the level sensitive interrupt will immediately retrigger.

## 3.8 Tick Timer

Tick timer facility is enabled with TTMR[M]. TTCR is incremented with each clock cycle and a high priority interrupt can be asserted whenever lower 28 bits of TTCR match TTMR[TP] and TTMR[IE] is set.

TTCR restarts counting from zero when match event happens and TTMR[M] is 0x1. If TTMR[M] is 0x2, TTCR is stoped when match event happens and TTCR must be changed to start counting again. When TTMR[M] is 0x3, TTCR keeps counting even when match event happens.

## 3.9 Power Management

### 3.9.1 Clock Gating and Frequency Changing Versus CPU Stalling

If system doesn t support clock gating and if changing clock frequency in slow down mode is not possible, CPU can be stalled for certain number of clock cycles. This is much lower benefit on power consumption however it still reduces power consumption.

### 3.9.2 Slow Down Mode

Slow down mode is software controlled with the 4-bit value in PMR[SDF]. Lower value specifies higher expected performance from the processor core. Usually PMR[SDF] is dynamically set by the operating system s idle routine, that monitors the usage of the processor core.

PMR[SDF] is broadcast on `pm_clksd`. External clock generator should adjust clock frequency according to the value of `pm_clksd`. Exact slow down factors are not defined but 0xF should go all the way down to 32.768 KHz.

With `pm_clksd` equal to 0xF, `pm_lvolt` is asserted. This is an indication for the external power supply to lower the voltage.

### 3.9.3 Doze Mode

To switch to doze mode, software should set the PMR[DME]. Once an interrupt is received by the programmable interrupt controller (PIC), `pm_wakeup` is asserted and external clock generation circuitry should enable all clocks. Once clocks are running RISC is switched back again to the normal mode and PMR[DME] is cleared.

When doze mode is enabled, `pm_dc_gate`, `pm_ic_gate`, `pm_dmmu_gate`, `pm_immu_gate` and `pm_cpugate` are asserted. As a result all clocks except `clk_tt` should be gated by external clock generation circuitry.

### 3.9.4 Sleep Mode

To switch to sleep mode, software should set the PMR[SME]. Once an interrupt is received by the programmable interrupt controller (PIC), `pm_wakeup` is asserted and external clock generation should enable all clocks. Once clocks are running, RISC is switched back again to the normal mode and PMR[SME] is cleared.

When sleep mode is enabled, `pm_dc_gate`, `pm_ic_gate`, `pm_dmmu_gate`, `pm_immu_gate`, `pm_cpu_gate` and `pm_tt_gate` are asserted. As a result all clocks including `clk_tt` should be gated by external clock generation circuitry.

In sleep mode, `pm_lvolt` is asserted. This is an indication for the external power supply to lower the voltage.

### 3.9.5 Clock Gating

Clock gating feature is not implemented in OR1200 power management.

### 3.9.6 Disabled Units Force Clock Gating

Units that are disabled in special-purpose register SR, have their clock gate signals asserted. Cleared bits SR[DCE], SR[ICE], SR[DME] and SR[IME] directly force assertion of `pm_dc_gate`, `pm_ic_gate`, `pm_dmmu_gate` and `pm_immu_gate`.

## 3.10 Debug Unit

Debug unit can be controlled through development interface or it can operate independently programmed and handled by the RISC s resident debug software.

### 3.10.1 Watchpoints

OR1200 debug unit does not implement OR12000 architecture watchpoints.

### 3.10.2 Breakpoint Exception

Which breakpointDMR2[WGB] bits specify which watchpoints invoke breakpoint exception. By invoking breakpoint exception, target resident debugger can be built.

Breakpoint is broadcast on development interface on `dbg_bp_o`.

## 3.11 Development Interface

**Note**
The information in this section is to be reviewed. It is the author's opinion that the debug interface is now largely provided by the SPR mappings, and no special sideband functions exist aside from stalling and resetting the core.

An additional *development and debug interface IP* core may be used to connect OpenRISC 1200 to standard debuggers using IEEE.1149.1 (JTAG) protocol.

### 3.11.1 Debugging Through Development Interface

The DSR special-purpose register specifies which exceptions cause the core to stop the execution of the exception handler and turn over control to development interface. It can be programmed by the resident debug software or by the development interface.

The DRR special-purpose register is specifies which event caused the core to stop the execution of program flow and turned over control to the development interface. It should be cleared by the resident debug software or by the development interface.

The DIR special-purpose register is not implemented.

### 3.11.2 Reading PC, Load/Store EA, Load Data, Store Data, Instruction

Crucial information like program counter (PC), load/store effective address (LSEA), load data, store data and current instruction in execution pipeline can be asynchronously read through the development interface.

Table 3.9: Development Interface Operation Commands

| **dbg_op_i[2:0]** | **Meaning** |
|---|---|
| 0x0 | Reading Program Counter (PC) |
| 0x1 | Reading Load/Store Effective Address |
| 0x2 | Reading Load Data |
| 0x3 | Reading Store Data |
| 0x4 | Reading SPR |
| 0x5 | Writing SPR |
| 0x6 | Reading Instruction in Execution Pipeline |
| 0x7 | Reserved |

Table 3.9 lists operation commands that control what is read or written through development interface. All reads except reads and writes of SPRs are asynchronous.

### 3.11.3 Reading and Writing SPRs Through Development Interface

For reads and write to SPRs `dbg_op_i` must be set to 0x4 and 0x5, respectively.

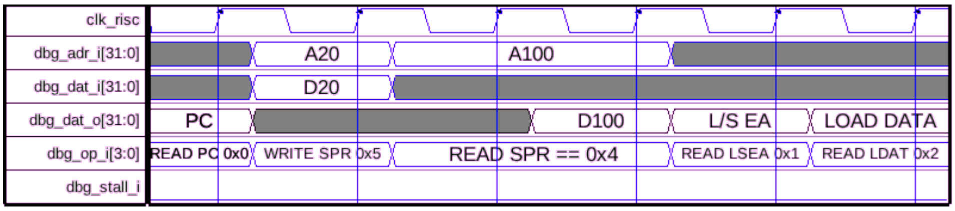


Figure 3.9: Development Interface Cycles

Figure 3.9 shows development interface cycles. Writes must be synchronous to the main RISC clock positive edge and should take one clock cycle. Reads must take two clock cycles because access to synchronous cache lines or to TLB entries introduces one clock cycle of delay.

If required, external debugger can stop the CPU core by asserting `dbg_stall_i`. This way it can have enough time to read all interesting registers from the RISC or guarantee that writes into SPRs are performed without RISC writing to the same registers.

### 3.11.4 Tracking Data Flow

An external debugger can monitor and record data flow inside the RISC for debugging purposes and profiling analysis. This is accomplished by monitoring status of the load/store unit, load/store effective address and load/store data, all available at the development interface.

Table 3.10: Status of the Load/Store Unit

| **dbg_lss_o[3:0]** | **Load/Store Instruction in Execution** |
|---|---|
| 0x0 | No load/store instruction in execution |
| 0x1 | Reserved for load doubleword |
| 0x2 | Load byte and zero extend |
| 0x3 | Load byte and sign extend |
| 0x4 | Load halfword and zero extend |
| 0x5 | Load halfword and sign extend |

Table 3.10: (continued)

| dbg_lss_o[3:0] | Load/Store Instruction in Execution |
|---|---|
| 0x6 | Load singleword and zero extend |
| 0x7 | Load singleword and sign extend |
| 0x8 | Reserved for store doubleword |
| 0x9 | Reserved |
| 0xA | Store byte |
| 0xB | Reserved |
| 0xC | Store halfword |
| 0xD | Reserved |
| 0xE | Store singleword |
| 0xF | Reserved |

External trace buffer can capture all interesting data flow events by analyzing status of the load/store unit available on `dbg_lss_o`. Table 3.10 lists different status encoding for the load/store unit.

### 3.11.5 Tracking Program Flow

An external debugger can monitor and record program flow inside the RISC for debugging purposes and profiling analysis. This is accomplished by monitoring status of the instruction unit, PC and fetched instruction word, all available at the development interface.

Table 3.11: Status of the Instruction Unit

| dbg_is_o[1:0] | Instruction Fetch Status |
|---|---|
| 0x0 | No instruction fetch in progress |
| 0x1 | Normal instruction fetch |
| 0x2 | Executing branch instruction |
| 0x3 | Fetching instruction in delay slot |

External trace buffer can capture all interesting program flow events by analyzing status of the instruction unit available on `dbg_is_o`. Table 3.11 lists different status encoding for the instruction unit.

### 3.11.6 Triggering External Watchpoint Event

Figure 3.10 shows how development interface can assert `dbg_ewt_I` and cause watchpoint event. If programmed, external watchpoint event will cause a breakpoint exception.

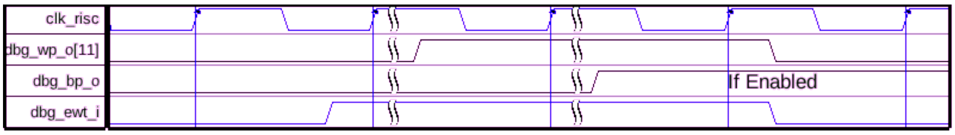


Figure 3.10: Assertion of External Watchpoint Trigger

# Chapter 4

# Registers

This section describes all registers inside the OR1200 core. Shifting *GRP* number 11 bits left and adding *REG* number computes the address of each special-purpose register. All registers are 32 bits wide from software perspective. *USER MODE* and *SUPV MODE* specify the valid access types for each register in user mode and supervisor mode of operation. R/W stands for read and write access and R stands for read only access.

## 4.1 Registers list

Table 4.1: List of All Registers

| **Grp #** | **Reg #** | **Reg Name** | **USER MODE** | **SUPV MODE** | **Description** |
|---|---|---|---|---|---|
| 0 | 0 | VR | - | R | Version Register |
| 0 | 1 | UPR | - | R | Unit Present Register |
| 0 | 2 | CPUCFGR | - | R | CPU Configuration Register |
| 0 | 3 | DMMUCFGR | - | R | Data MMU Configuration Register |
| 0 | 4 | IMMUCFGR | - | R | Instruction MMU Configuration Register |
| 0 | 5 | DCCFGR | - | R | Data Cache Configuration Register |
| 0 | 6 | ICCFGR | - | R | Instruction Cache Configuration Register |
| 0 | 7 | DCFGR | - | R | Debug Configuration Register |
| 0 | 16 | PC | - | R/W | PC mapped to SPR space |
| 0 | 17 | SR | - | R/W | Supervision Register |
| 0 | 20 | FPCSR | - | R/W | FP Control Status Register |

Table 4.1: (continued)

| Grp # | Reg # | Reg Name | USER MODE | SUPV MODE | Description |
|---|---|---|---|---|---|
| 0 | 32 | EPCR0 | - | R/W | Exception PC Register |
| 0 | 48 | EEAR0 | - | R/W | Exception EA Register |
| 0 | 64 | ESR0 | - | R/W | Exception SR Register |
| 0 | 1024-1055 | GPR0-GPR31 | - | R/W | GPRs mapped to SPR space |
| 1 | 2 | DTLBEIR | - | W | Data TLB Entry Invalidate Register |
| 1 | 1024-1151 | DTLBW0MR0-DTLBW0MR127 | - | R/W | Data TLB Match Registers Way 0 |
| 1 | 1536-1663 | DTLBW0TR0-DTLBW0TR127 | - | R/W | Data TLB Translate Registers Way 0 |
| 2 | 2 | ITLBEIR | - | W | Instruction TLB Entry Invalidate Register |
| 2 | 1024-1151 | ITLBW0MR0-ITLBW0MR127 | - | R/W | Instruction TLB Match Registers Way 0 |
| 2 | 1536-1663 | ITLBW0TR0-ITLBW0TR127 | - | R/W | Instruction TLB Translate Registers Way 0 |
| 3 | 0 | DCCR | - | R/W | DC Control Register |
| 3 | 2 | DCBFR | W | W | DC Block Flush Register |
| 3 | 3 | DCBIR | W | W | DC Block Invalidate Register |
| 3 | 4 | DCBWR | W | W | DC Block Write-back register |
| 4 | 0 | ICCR | - | R/W | IC Control Register |
| 4 | 256 | ICBIR | W | W | IC Block Invalidate Register |
| 5 | 256 | MACLO | R/W | R/W | MAC Low |
| 5 | 257 | MACHI | R/W | R/W | MAC High |
| 6 | 16 | DMR1 | - | R/W | Debug Mode Register 1 |
| 6 | 17 | DMR2 | - | R/W | Debug Mode Register 2 |
| 6 | 20 | DSR | - | R/W | Debug Stop Register |
| 6 | 21 | DRR | - | R/W | Debug Reason Register |
| 8 | 0 | PMR | - | R/W | Power Management Register |

Table 4.1: (continued)

| Grp # | Reg # | Reg Name | USER MODE | SUPV MODE | Description |
|---|---|---|---|---|---|
| 9 | 0 | PICMR | - | R/W | PIC Mask Register |
| 9 | 2 | PICSR | - | R/W | PIC Status Register |
| 10 | 0 | TTMR | - | R/W | Tick Timer Mode Register |
| 10 | 1 | TTCR | R* | R/W | Tick Timer Count Register |

Table 4.1 lists all OpenRISC 1000 special-purpose registers implemented in OR1200. Registers VR and UPR are described below. For description of other registers refer to [or1000_manual].

## 4.2 Register VR description

Special-purpose register VR identifies the version (model) and revision level of the OpenRISC 1000 processor. It also specifies possible standard template on which this implementation is based.

Table 4.2: VR Register

| Bit # | Access | Reset | Short Name | Description |
|---|---|---|---|---|
| 5:0 | R | Revision | REV | Revision number of this document. |
| 15:6 | R | 0x0 | - | Reserved |
| 23:16 | R | 0x00 | CFG | Configuration should be read from UPR and configuration registers |
| 31:24 | R | 0x12 | VER | Version number for OR1200 is fixed at 0x1200. |

## 4.3 Register UPR description

Special-purpose register UPR identifies the units present in the processor. It has a bit for each implemented unit or functionality. Lower sixteen bits identify present units defined in the OpenRISC 1000 architecture. Upper sixteen bits define present custom units.

Table 4.3: UPR Register

| Bit # | Access | Reset | Short Name | Description |
|---|---|---|---|---|
| 0 | R | 1 | UP | UPR present |
| 1 | R | 1 | DCP | Data cache present[†] |

Table 4.3: (continued)

| Bit # | Access | Reset | Short Name | Description |
|---|---|---|---|---|
| 2 | R | 1 | ICP | Instruction cache present[†] |
| 3 | R | 1 | DMP | Data MMU present[†] |
| 4 | R | 1 | IMP | Instruction MMU present[†] |
| 5 | R | 1 | MP | MAC present[†] |
| 6 | R | 1 | DUP | Debug unit present[†] |
| 7 | R | 0 | PCUP | Performance counters unit not present[†] |
| 8 | R | 1 | PMP | Power Management Present[†] |
| 9 | R | 1 | PICP | Programmable interrupt controller present |
| 10 | R | 1 | TTP | Tick timer present |
| 11 | R | 1 | FPP | Floating point present[†] |
| 23:12 | R | X | - | Reserved |
| 31:24 | R | 0xXXXX | CUP | The user of the OR1200 core adds custom units. |

[†]: if enabled at synthesis time

## 4.4 Register CPUCFGR description

Special-purpose register CPUCFGR identifies the capabilities and configuration of the CPU.

Table 4.4: CPUCFGR Register

| Bit # | Access | Reset | Short Name | Description |
|---|---|---|---|---|
| 3:0 | R | 0x0 | NSGF | Zero number of shadow GPR files |
| 4 | R | 0 | HGF | No half GPR files[†] |
| 5 | R | 1 | OB32S | ORBIS32 supported |
| 6 | R | 0 | OB64S | ORBIS64 not supported |
| 7 | R | 1 | OF32S | ORFPX32 supported[‡] |
| 8 | R | 0 | OF64S | ORFPX64 not supported |
| 9 | R | 0 | OV64S | ORVDX64 not supported |

[†]: If disabled at synthesis time

[‡]: If FPU enabled at synthesis time

## 4.5 Register DMMUCFGR description

Special-purpose register DMMUCFGR identifies the capabilities and configuration of the DMMU.

Table 4.5: DMMUCFGR Register

| **Bit #** | **Access** | **Reset** | **Short Name** | **Description** |
|---|---|---|---|---|
| 1:0 | R | 0x0 | NTW | One DTLB way |
| 4:2 | R | 0x4 - 0x7 | NTS | 16, 32, 64 or 128 DTLB sets |
| 7:5 | R | 0x0 | NAE | No ATB Entries |
| 8 | R | 0 | CRI | No DMMU control register implemented |
| 9 | R | 0 | PRI | No protection register implemented |
| 10 | R | 1 | TEIRI | DTLB entry invalidate register implemented |
| 11 | R | 0 | HTR | No hardware DTLB reload |

## 4.6 Register IMMUCFGR description

Special-purpose register IMMUCFGR identifies the capabilities and configuration of the IMMU.

Table 4.6: IMMUCFGR Register

| **Bit #** | **Access** | **Reset** | **Short Name** | **Description** |
|---|---|---|---|---|
| 1:0 | R | 0x0 | NTW | One ITLB way |
| 4:2 | R | 0x4 - 0x7 | NTS | 16, 32, 64 or 128 ITLB sets |
| 7:5 | R | 0x0 | NAE | No ATB Entries |
| 8 | R | 0 | CRI | No IMMU control register implemented |
| 9 | R | 0 | PRI | No protection register implemented |
| 10 | R | 1 | TEIRI | ITLB entry invalidate register implemented |
| 11 | R | 0 | HTR | No hardware ITLB reload |

## 4.7 Register DCCFGR description

Special-purpose register DCCFGR identifies the capabilities and configuration of the data cache.

Table 4.7: DCCFGR Register

| Bit # | Access | Reset | Short Name | Description |
|---|---|---|---|---|
| 2:0 | R | 0x0 | NCW | One DC way |
| 6:3 | R | 0x4 - 0x7 | NCS | 16, 32, 64 or 128 DC sets |
| 7 | R | 0x0 | CBS | 16-byte cache block size |
| 8 | R | 0 | CWS | Cache write-through strategy[†] |
| 9 | R | 1 | CCRI | DC control register implemented |
| 10 | R | 1 | CBIRI | DC block invalidate register implemented |
| 11 | R | 0 | CBPRI | DC block prefetch register not implemented |
| 12 | R | 0 | CBLRI | DC block lock register not implemented |
| 13 | R | 1 | CBFRI | DC block flush register implemented |
| 14 | R | 1 | CBWBRI | DC block write-back register implemented[‡] |

[†]: If disabled at synthesis time

[‡]: If FPU enabled at synthesis time

## 4.8 Register ICCFGR description

Special-purpose register ICCFGR identifies the capabilities and configuration of the instruction cache.

Table 4.8: ICCFGR Register

| Bit # | Access | Reset | Short Name | Description |
|---|---|---|---|---|
| 2:0 | R | 0x0 | NCW | One IC way |
| 6:3 | R | 0x4 - 0x7 | NCS | 16, 32, 64 or 128 IC sets |
| 7 | R | 0x0 | CBS | 16-byte cache block size |

Table 4.8: (continued)

| Bit # | Access | Reset | Short Name | Description |
|---|---|---|---|---|
| 8 | R | 0 | CWS | Cache write-through strategy |
| 9 | R | 1 | CCRI | IC control register implemented |
| 10 | R | 1 | CBIRI | IC block invalidate register implemented |
| 11 | R | 0 | CBPRI | IC block prefetch register not implemented |
| 12 | R | 0 | CBLRI | IC block lock register not implemented |
| 13 | R | 1 | CBFRI | IC block flush register implemented |
| 14 | R | 0 | CBWBRI | IC block write-back register not implemented |

## 4.9 Register DCFGR description

Special-purpose register DCFGR identifies the capabilities and configuration of the debut unit.

Table 4.9: DCFGR Register

| Bit # | Access | Reset | Short Name | Description |
|---|---|---|---|---|
| 3:0 | R | 0x0 | NDP | Zero DVR/DCR pairs[†] |
| 4 | R | 0 | WPCI | Watchpoint counters not implemented |

[†]: If hardware breakpoints disabled at synthesis time

# Chapter 5

# IO ports

OR1200 IP core has several interfaces. Figure 5.1 below shows all interfaces:

- Instruction and data WISHBONE host interfaces
- Power management interface
- Development interface
- Interrupts interface

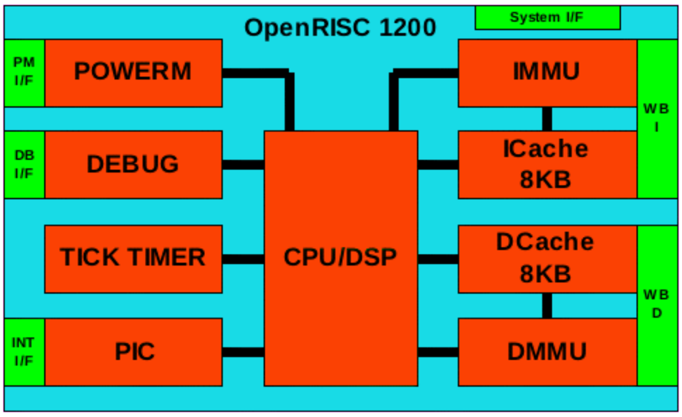

OpenRISC 1200
System I/F
PM I/F
POWERM
DB I/F
DEBUG
TICK TIMER
INT I/F
PIC
CPU/DSP
IMMU
ICache 8KB
DCache 8KB
DMMU
WB I
WB D

Figure 5.1: Core's Interfaces

## 5.1 Instruction WISHBONE Master Interface

OR1200 has two master WISHBONE Rev B compliant interfaces. Instruction interface is used to connect OR1200 core to memory subsystem for purpose of fetching instructions or instruction cache lines.

Table 5.1: Instruction WISHBONE Master Interface' Signals

| Port | Width | Direction | Description |
|---|---|---|---|
| iwb_CLK_I | 1 | Input | Clock input |
| iwb_RST_I | 1 | Input | Reset input |
| iwb_CYC_O | 1 | Output | Indicates valid bus cycle (core select) |

Table 5.1: (continued)

| Port | Width | Direction | Description |
|---|---|---|---|
| iwb_ADR_O | 32 | Outputs | Address outputs |
| iwb_DAT_I | 32 | Inputs | Data inputs |
| iwb_DAT_O | 32 | Outputs | Data outputs |
| iwb_SEL_O | 4 | Outputs | Indicates valid bytes on data bus (during valid cycle it must be 0xf) |
| iwb_ACK_I | 1 | Input | Acknowledgment input (indicates normal transaction termination) |
| iwb_ERR_I | 1 | Input | Error acknowledgment input (indicates an abnormal transaction termination) |
| iwb_RTY_I | 1 | Input | In OR1200 treated same way as iwb_ERR_I. |
| iwb_WE_O | 1 | Output | Write transaction when asserted high |
| iwb_STB_O | 1 | Outputs | Indicates valid data transfer cycle |

## 5.2 Data WISHBONE Master Interface

OR1200 has two master WISHBONE Rev B compliant interfaces. Data interface is used to connect OR1200 core to external peripherals and memory subsystem for purpose of reading and writing data or data cache lines.

Table 5.2: Data WISHBONE Master Interface' Signals

| Port | Width | Direction | Description |
|---|---|---|---|
| dwb_CLK_I | 1 | Input | Clock input |
| dwb_RST_I | 1 | Input | Reset input |
| dwb_CYC_O | 1 | Output | Indicates valid bus cycle (core select) |
| dwb_ADR_O | 32 | Outputs | Address outputs |
| dwb_DAT_I | 32 | Inputs | Data inputs |
| dwb_DAT_O | 32 | Outputs | Data outputs |
| dwb_SEL_O | 4 | Outputs | Indicates valid bytes on data bus (during valid cycle it must be 0xf) |
| dwb_ACK_I | 1 | Input | Acknowledgment input (indicates normal transaction termination) |
| dwb_ERR_I | 1 | Input | Error acknowledgment input (indicates an abnormal transaction termination) |
| dwb_RTY_I | 1 | Input | In OR1200 treated same way as dwb_ERR_I. |
| dwb_WE_O | 1 | Output | Write transaction when asserted high |

Table 5.2: (continued)

| Port | Width | Direction | Description |
|---|---|---|---|
| dwb_STB_O | 1 | Outputs | Indicates valid data transfer cycle |

## 5.3 System Interface

System interface connects reset, clock and other system signals to the OR1200 core.

Table 5.3: System Interface Signals

| Port | Width | Direction | Description |
|---|---|---|---|
| Rst | 1 | Input | Asynchronous reset |
| clk_cpu | 1 | Input | Main clock input to the RISC |
| clk_dc | 1 | Input | Data cache clock |
| clk_ic | 1 | Input | Instruction cache clock |
| clk_dmmu | 1 | Input | Data MMU clock |
| clk_immu | 1 | Input | Instruction MMU clock |
| clk_tt | 1 | Input | Tick timer clock |

## 5.4 Development Interface

Development interface connects external development port to the RISC s internal debug facility. Debug facility allows control over program execution inside RISC, setting of breakpoints and watchpoints, and tracing of instruction and data flows.

Table 5.4: Development Interface

| Port | Width | Direction | Description |
|---|---|---|---|
| dbg_dat_o | 32 | Output | Transfer of data from RISC to external development interface |
| dbg_dat_i | 32 | Input | Transfer of data from external development interface to RISC |
| dbg_adr_i | 32 | Input | Address of special-purpose register to be read or written |
| dbg_op_I | 3 | Input | Operation select for development interface |
| dbg_lss_o | 4 | Output | Status of load/store unit |
| dbg_is_o | 2 | Output | Status of instruction fetch unit |
| dbg_wp_o | 11 | Output | Status of watchpoints |
| dbg_bp_o | 1 | Output | Status of the breakpoint |
| dbg_stall_i | 1 | Input | Stalls RISC CPU core |

Table 5.4: (continued)

| Port | Width | Direction | Description |
|---|---|---|---|
| dbg_ewt_i | 1 | Input | External watchpoint trigger |

## 5.5 Power Management Interface

Power management interface provides signals for interfacing RISC core with external power management circuitry. External power management circuitry is required to implement functions that are technology specific and cannot be implemented inside OR1200 core.

Table 5.5: Power Management Interface

| Port | Width | Direction | Generation | Description |
|---|---|---|---|---|
| pm_clksd | 4 | Output | Static (in SW) | Slow down outputs that control reduction of RISC clock frequency |
| pm_cpustall | 1 | Input | - | Synchronous stall of the RISC's CPU core |
| pm_dc_gate | 1 | Output | Dynamic (in HW) | Gating of data cache clock |
| pm_ic_gate | 1 | Output | Dynamic (in HW) | Gating of instruction cache clock |
| pm_dmmu_gate | 1 | Output | Dynamic (in HW) | Gating of data MMU clock |
| pm_immu_gate | 1 | Output | Dynamic (in HW) | Gating of instruction MMU clock |
| pm_tt_gate | 1 | Output | Dynamic (in HW) | Gating of tick timer clock |
| pm_cpu_gate | 1 | Output | Static (in SW) | Gating of main CPU clock |
| pm_wakeup | 1 | Output | Dynamic (in HW) | Activate all clocks |
| pm_lvolt | 1 | Output | Static (in SW) | Lower voltage |

## 5.6 Interrupt Interface

Interrupt interface has interrupt inputs for interfacing external peripheral s interrupt outputs to the RISC core. All interrupt inputs are evaluated on positive edge of main RISC clock.

Table 5.6: Interrupt Interface

| Port | Width | Direction | Description |
|---|---|---|---|
| pic_ints | PIC_INTS | Input | External interrupts |

# Appendix A

# Core HW Configuration

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

Table A.1: Core HW configuration table

| Variable Name | Range | Default | Description |
|---|---|---|---|
| EADDR_WIDTH | 32 | 32 | Effective address width |
| VADDR_WIDTH | 32 | 32 | Virtual address width |
| PADDR_WIDTH | 24 - 36 | 32 | Physical address width |
| DATA_WIDTH | 32 | 32 | Data width / Operation width |
| DC_IMPL | 0 - 1 | 1 | Data cache implementation |
| DC_SETS | 256-1024 | 512 | Data cache number of sets |
| DC_WAYS | 1 | 1 | Data cache number of ways |
| DC_LINE | 16 - 32 | 16 | Data cache line size |
| IC_IMPL | 0 - 1 | 1 | Instruction cache implementation |
| IC_SETS | 32-1024 | 512 | Instruction cache number of sets |
| IC_WAYS | 1 | 1 | Instruction cache number of ways |
| IC_LINE | 16-32 | 16 | Instruction cache line size in bytes |
| DMMU_IMPL | 0 - 1 | 1 | Data MMU implementation |
| DTLB_SETS | 64 | 64 | Data TLB number of sets |
| DTLB_WAYS | 1 | 1 | Data TLB number of ways |
| IMMU_IMPL | 0 - 1 | 1 | Instruction MMU implementation |
| ITLB_SETS | 64 | 64 | Instruction TLB number of sets |
| ITLB_WAYS | 1 | 1 | Instruction TLB number of ways |
| PIC_INTS | 2 - 32 | 20 | Number of interrupt inputs |

# Chapter 6

# Bibliography

## 6.1 Bibliography

[1] [or1000_manual] Damjan Lampret et al. OpenRISC 1000 System Architecture Manual. 2004.

# Chapter 7

# Index

**Numbers**
1200, 2

**A**
Address Translation
    Data, 21
    Instruction, 24
Architecture
    Harvard, 7

**B**
Bibliography, 44
Breakpoint, 28

**C**
Cache, 20
cache, 6, 7, 18
cache directory, 8
Cache Line Lock, 21
clk_cpu, 40
clk_dc, 40
clk_dmmu, 40
clk_ic, 40
clk_immu, 40
clk_tt, 40
clock, 12
Clock Gating, 27
Clock gating, 12, 28
Coherency, 21
coherency, 19
Configuration, 43
CPU, 4, 14
CPUCFGR, 31, 34

**D**
Data, 8, 21–24
Data Flow, 29
DATA_WIDTH, 43
dbg_adr_i, 40
dbg_bp_o, 40
dbg_dat_i, 40
dbg_dat_o, 40
dbg_ewt_i, 41
dbg_is_o, 40
dbg_lss_o, 40
dbg_op_I, 40
dbg_stall_i, 40
dbg_wp_o, 40
DC_IMPL, 43
DC_LINE, 43
DC_SETS, 43
DC_WAYS, 43
DCBFR, 32
DCBIR, 32
DCBWR, 32
DCCFGR, 31, 36
DCCR, 32
DCFGR, 31, 37
Debug Unit, 28
Debug unit, 12
Debugging, 28
Development Interface, 28
DMMU, 20
DMMU_IMPL, 43
DMMUCFGR, 31, 35
DMMUCR, 22
DMR1, 32
DMR2, 32
Doze, 27
DRR, 32
DSP, 4, 14
DSR, 32
DTLB, 18, 22, 23
DTLB_SETS, 43
DTLB_WAYS, 43
DTLBEIR, 32
DTLBW0MR0-DTLBW0MR127, 32
DTLBW0TR0-DTLBW0TR127, 32
DTLBWyMR, 23
DTLBWyTR, 23
dwb_ACK_I, 39
dwb_ADR_O, 39
dwb_CLK_I, 39
dwb_CYC_O, 39
dwb_DAT_I, 39
dwb_DAT_O, 39
dwb_ERR_I, 39

dwb_RST_I, 39
dwb_RTY_I, 39
dwb_SEL_O, 39
dwb_STB_O, 40
dwb_WE_O, 39

**E**
EADDR_WIDTH, 43
EEAR0, 32
EPCR0, 32
ESR0, 32
Exception handling, 5
exception model, 17
Exception sources, 5
Exceptions, 17
External Watchpoint Event, 30

**F**
Family, 1
Fetch, 20
Floating Point Unit, 5
floating point unit, 16
Flush, 19, 25
FPCSR, 31
FPU, 4, 5, 14

**G**
General-purpose register, 15
GPR0-GPR31, 32

**H**
Hardware
    Configuration, 43
Harvard, 7

**I**
IC_IMPL, 43
IC_LINE, 43
IC_SETS, 43
IC_WAYS, 43
ICBIR, 32
ICCFGR, 31, 36
ICCR, 32
IMMU_IMPL, 43
IMMUCFGR, 31, 35
IMMUCR, 25
inhibit, 20
Instruction, 9, 24–26
Instruction List, 15
Instruction MMU, 24
Instruction unit, 15
Integer Execution, 5, 16
interrupt, 11
Invalidate, 21
invalidate, 20
Invalidation, 21
invalidation, 19
IO ports, 38
ITLB, 25
ITLB_SETS, 43
ITLB_WAYS, 43
ITLBEIR, 32
ITLBW0MR0-ITLBW0MR127, 32
ITLBW0TR0-ITLBW0TR127, 32
iwb_ACK_I, 39
iwb_ADR_O, 39
iwb_CLK_I, 38
iwb_CYC_O, 38
iwb_DAT_I, 39
iwb_DAT_O, 39
iwb_ERR_I, 39
iwb_RST_I, 38
iwb_RTY_I, 39
iwb_SEL_O, 39
iwb_STB_O, 39
iwb_WE_O, 39

**L**
l.add, 15
l.addc, 15
l.addi, 15
l.and, 15
l.andi, 15
l.bf, 15
l.bnf, 15
l.div, 15
l.ff1, 15
l.fl1, 15
l.j, 15
l.jal, 15
l.jalr, 15
l.jr, 15
l.lbs, 15
l.lbz, 15
l.lhs, 15
l.lhz, 15
l.lws, 15
l.lwz, 15
l.mac, 15
l.maci, 15
l.macrc, 15
l.mfspr, 15
l.movhi, 15
l.msb, 15
l.mtspr, 15
l.mul, 15
l.muli, 15
l.nop, 15
l.or, 15
l.ori, 15
l.rfe, 15
l.rori, 15
l.sb, 15
l.sfeq, 15
l.sfges, 15

l.sfgeu, 15
l.sfgts, 15
l.sfgtu, 15
l.sfleu, 15
l.sflts, 15
l.sfltu, 15
l.sfne, 15
l.sh, 15
l.sll, 15
l.slli, 15
l.sra, 15
l.srai, 15
l.srl, 15
l.srli, 15
l.sub, 15
l.sw, 15
l.sys, 15
l.trap, 15
l.xor, 15
l.xori, 15
lf.add.s, 15
lf.div.s, 15
lf.ftoi.s, 15
lf.itof.s, 15
lf.madd.s, 5
lf.mul.s, 15
lf.rem.s, 5
lf.sfeq.s, 15
lf.sfge.s, 15
lf.sfgt.s, 15
lf.sfle.s, 15
lf.sflt.s, 15
lf.sfne.s, 15
lf.sub.s, 15
load/store unit (LSU), 4
Lock, 20
locking, 19
low-power, 11
LRU, 7
LSU, 15

**M**
MAC, 5, 16
MACHI, 32
MACLO, 32
miss, 20
MMU
    Data, 8
    Instruction, 9
Mode
    Doze, 27
    Sleep, 27
    Slow Down, 27

**O**
OpenRISC
    1200, 2
    Family, 1
OpenRISC 1200
    Instruction List, 15
ORBIS32, 4
ORBIS64, 4
ORFBX64, 4
ORFPX32, 4, 5
ORFPX3264, 4
ORVDX64, 4

**P**
PADDR_WIDTH, 43
Page Attributes
    Data, 23, 24
    Instruction, 26
Page Protection
    Data, 22
    Instruction, 25
page tables, 22
PC, 31
PIC, 26
PIC_INTS, 43
pic_ints, 41
PICMR, 33
PICSR, 33
Pipeline
    Integer Execution, 5, 16
pm_clksd, 41
pm_cpu_gate, 41
pm_cpustall, 41
pm_dc_gate, 41
pm_dmmu_gate, 41
pm_ic_gate, 41
pm_immu_gate, 41
pm_lvolt, 41
pm_tt_gate, 41
pm_wakeup, 41
PMR, 32
power consumption, 11
Power Management, 27
Prefetch, 21
prefetch, 19
program counter, 28
Program Flow, 30
Programmable Interrupt Controller, 26
Protection, 8
PTE, 23

**R**
Register
    CPUCFGR, 34
    DCCFGR, 36
    DCFGR, 37
    DMMUCFGR, 35
    ICCFGR, 36
    IMMUCFGR, 35
    UPR, 33

    VR, 33
Registers, 31
registers, 4
Registers list, 31
reset, 12, 14
Rst, 40

**S**
Sleep, 27
Slow Down, 27
SR, 31
System unit, 17
system unit, 5

**T**
Tick Timer, 27
timer, 11
TTCR, 33
TTMR, 33

**U**
UPR, 31, 33

**V**
VADDR_WIDTH, 43
virtual memory management, 8
VR, 31, 33

**W**
Watchpoints, 28
WISHBONE, 13
Write-back, 20
write-back, 18
write-through, 18, 20