ECAP5-DPROC

RISC-V processor

Architecture Document

Revision: 1.0.0-draft1

Issue Date: September 2, 2023

Contents

1 Introduction

1.1 Purpose

This documents aims at defining the requirements for ECAP5-DPROC as well as describing its architecture. Both user and product requirements will be covered.

1.2 Intended Audience and Use

This document targets hardware engineers who shall implement ECAP5-DPROC by refering to the described architecture. It is also intended for system engineers working on the integration of ECAP5-DPROC in ECAP5. Finally, this document shall be used as a technical reference by software engineers configuring ECAP5-DPROC through hardware-software interfaces.

1.3 Product Scope

ECAP5-DPROC is an implementation of the RISC-V instruction set architecture targetting *Educational Computer Architecture Platform 5* (ECAP5). It will provide the main means of software execution in ECAP5.

1.4 Conventions

Requirements shall be described here.

Requirement relationships :

- Composition
- Derivation
- Refinement
- Satisfy
- Verify
- Copy

The bit indexing shall be described somewhere.

Byte size as well.

Inputs missing from timing diagrams are considered low or undefined.

Italic names are timing diagram parameters.

1.5 Definitions and Abbreviations

hardware-configurable

software-configurable

1.6 References

Date	Version	Title
December 13, 2019	20191213	The RISC-V Instruction Set Manual Volume I: User-Level ISA
March 22, 2019	0.13.2	RISC-V External Debug Support
June 22, 2010	B.4	WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores

2 Overall Description

2.1 User needs

ECAP5 is the primary user for ECAP5-DPROC. ECAP5-DPROC could however be used as a standalone RISC-V processor. The following requirements define the user needs.

ID	U_INSTRUCTION_SET_01
Description	ECAP5-DPROC shall implement the RV32I instruction set.

In order to improve the usability of ECAP5-DPROC, it shall have a *von Neumann* architecture as it only requires one memory interface.

ID	U_MEMORY_INTERFACE_01
Description	ECAP5-DPROC shall access both instructions and data through a unique memory interface.

ID	U_MEMORY_INTERFACE_02
Description	ECAP5-DPROC's unique memory interface shall be compliant with the Wishbone specification.

ID	U_MEMORY_INTERFACE_03
Description	ECAP5-DPROC's unique memory interface shall be designed such that memory protocols can be interchanged at compile time.

TBC wishbone datasheet for external interface.

Table 1: Wishbone Datasheet for the memory interface

DESCRIPTION	**SPECIFICATION**
Revision level of the WISHBONE specification	B4
Type of interface	MASTER
Signal names for the WISHBONE interface	TBC
ERR_I support	No
RTY_I support	No
Supported tags	None
Port size	32-bit
Port granularity	8-bit
Maximum operand size	32-bit

Data transfer ordering	LITTLE ENDIAN
Sequence of data transfer	TBC
Clock constraints	TBC

ID	U_RESET_01
Description	ECAP5-DPROC shall provide a signal which shall hold ECAP5-DPROC in a reset state while asserted.

The polarity of the reset signal mentionned in `U_RESET_01` is not specified by the user.

ID	U_BOOT_ADDRESS_01
Description	The address at which ECAP5-DPROC jumps after the reset signal is de-asserted shall be hardware-configurable.

The address mentionned in `U_BOOT_ADDRESS_01` can be either configured through hardware signals or can be selected at compile time.

ID	U_HARDWARE_INTERRUPT_01
Description	ECAP5-DPROC shall provide a signal which shall interrupt ECAP5-DPROC's execution flow while asserted.

ID	U_HARDWARE_INTERRUPT_02
Description	ECAP5-DPROC shall jump to a software-configurable address when it is interrupted.

The memory address at which ECAP5-DPROC shall jump to when interrupted is not specified by the user.

ID	U_DEBUG_01
Description	ECAP5-DPROC shall be compliant with the RISC-V External Debug Support specification.

There is no performance goal required by ECAP5 for ECAP5-DPROC as ECAP5 is an educational platform.

2.2 Assumptions and Dependencies

Describe what the assumptions for the product are : Targeting the ecp5 family, based around opensource toolchains.

3 Requirements

3.1 External Interface Requirements

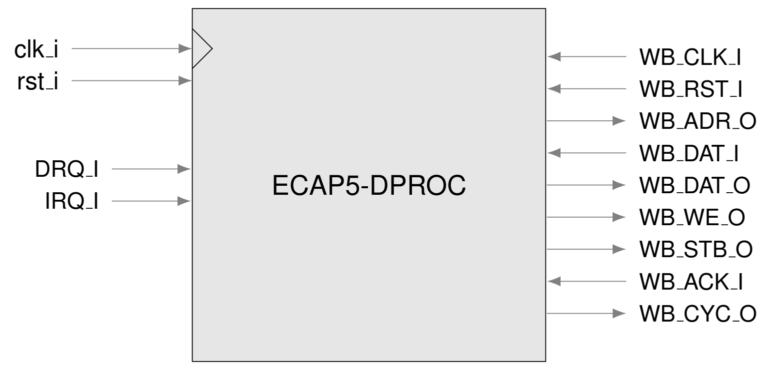


Figure 1: Schematic view of the external interface of ECAP5-DPROC

NAME	TYPE	WIDTH	DESCRIPTION
CLK	I	1	Clock input.
RST_N	I	1	Hardware reset. Active low.
IRQ_I	I	1	External interrupt request.
DRQ_I	I	1	Debug request.

Table 2: ECAP5-DPROC control signals

NAME	TYPE	WIDTH	DESCRIPTION
WB_CLK_I	I	1	TBD
WB_RST_I	I	1	TBD
WB_ADR_O	O	32	TBD
WB_DAT_I	I	32	TBD
WB_DAT_O	O	32	TBD
WB_WE_O	O	1	TBD
WB_STB_O	O		TBD
WB_ACK_I	I	1	TBD
WB_CYC_O	O		TBD

Table 3: ECAP5-DPROC memory interface signals

ID	I_CLK_01
Description	All inputs and outputs of ECAP5-DPROC shall belong to CLK's clock domain.

ID	I_RESET_01
Description	The RST_N signal shall hold ECAP5-DPROC in a reset state while asserted.
Derived From	U_RESET_01

ID	I_RESET_02
Description	RST_N polarity shall be active low.

ID	I_IRQ_01
Description	ECAP5-DPROC shall jump to a software-configurable address when input IRQ is asserted.
Derived From	U_HARDWARE_INTERRUPT_01, U_HARDWARE_INTERRUPT_02

ID	I_DIRQ_01
Description	TBC

ID	I_MEMORY_INTERFACE_01
Description	Signals from table 3 shall be compliant with the Wishbone specification.
Derived From	U_MEMORY_INTERFACE_02

Behavioral specification for symbols in table 3 is outlined in the functional requirements section, subsection 3.2.5.

3.2 Functional Requirements

3.2.1 Register file

ID	F_REGISTERS_01
Description	ECAP5-DPROC shall implement 32 user-accessible general purpose registers ranging from `x0` to `x31`.
Derived From	U_INSTRUCTION_SET_01

ID	F_REGISTERS_02
Description	Register `x0` shall always be equal to zero.
Derived From	U_INSTRUCTION_SET_01

ID	F_REGISTERS_03
Description	ECAP5-DPROC shall implement a `pc` register storing the address of the current instruction, user-accessible through the AUIPC instruction.
Derived From	U_INSTRUCTION_SET_01

3.2.2 Instruction decoding

Figure 2 outlines the different instruction encodings for the RV32I instruction set. The `opcode` parameter is a unique identifier for each instruction. The instruction encoding is infered from the opcode as there can only be one encoding per opcode.

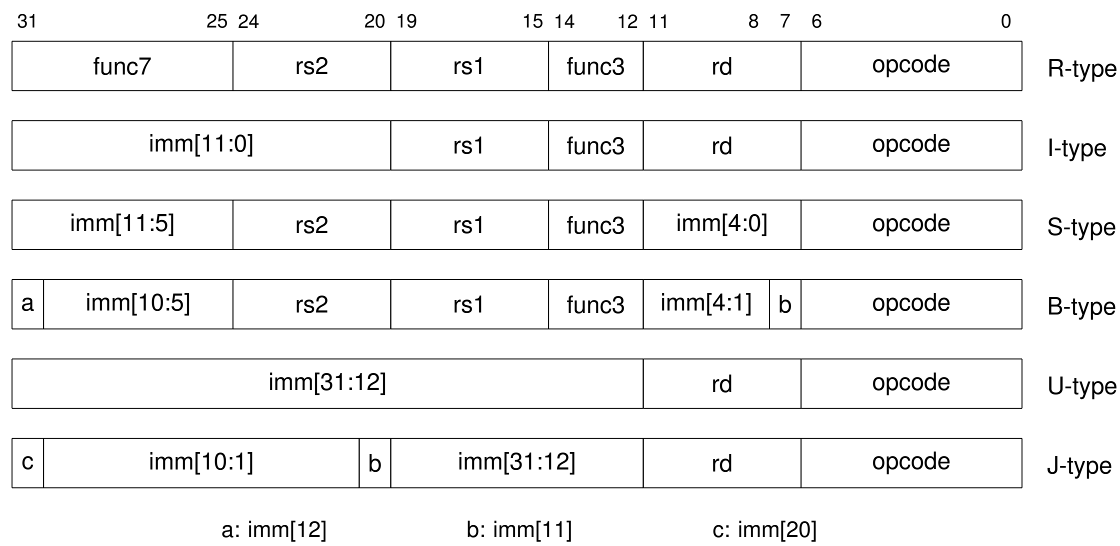


Figure 2: Instruction encodings of the RV32I instruction set

Immediate encoding

Only one immediate value can be encoded in one instruction. The value can be reconstructed from fragments of the following format : imm[x] representing the x^{th} bit or imm[x:y] representing bits from the x^{th} to the y^{th} both included.

ID	F_INSTR_IMMEDIATE_01
Description	Immediate values shall be sign-extended.
Derived From	U_INSTRUCTION_SET_01

ID	F_INSTR_IMMEDIATE_02
Description	The value of an instruction immediate shall be the concatenation of immediate fragments from the instruction encoding.
Derived From	U_INSTRUCTION_SET_01

ID	F_INSTR_IMMEDIATE_03
Description	Missing immediate fragments shall be replaced by zeros.
Derived From	U_INSTRUCTION_SET_01

RV32I is called a Load/Store ISA, meaning that instructions inputs and outputs are passed through registers or through an instruction immediate. There are specific instructions for loading and storing data into memory.

Instruction parameters

ID	F_INSTR_FIRST_PARAM_01
Description	Instructions encoded using the R-type, I-type, S-type and B-type shall take as their first parameter the value stored in the register designated by the `rs1` field.
Derived From	U_INSTRUCTION_SET_01

ID	F_INSTR_FIRST_PARAM_02
Description	Instructions encoded using the U-type and J-type shall take as their first parameter the immediate value encoded in the instruction.
Derived From	U_INSTRUCTION_SET_01

ID	F_INSTR_SECOND_PARAM_01
Description	Instructions encoded using the R-type, S-type and B-type shall take as their second parameter the value stored in the register designated by the `rs2` field.
Derived From	U_INSTRUCTION_SET_01

ID	F_INSTR_SECOND_PARAM_02
Description	Instructions encoded using the I-type shall take as its second parameter the immediate value encoded in the instruction.
Derived From	U_INSTRUCTION_SET_01

ID	F_INSTR_THIRD_PARAM_01
Description	Instructions encoded using the S-type and B-type shall take as their third parameter the immediate value encoded in the instruction.
Derived From	U_INSTRUCTION_SET_01

Instruction results

ID	F_INSTR_RESULT_01
Description	Instructions encoded using the R-type, I-type, U-type and J-type shall store their result in the register designated by the `rd` field.
Derived From	U_INSTRUCTION_SET_01

ID	F_INSTR_RESULT_02
Description	Instructions encoded using the S-type and B-type do not produce any result.
Derived From	U_INSTRUCTION_SET_01

Instruction variants

ID	F_INSTR_VARIANT_01
Description	Instructions encoded using the R-type, I-type, S-type and B-type shall use the `func3` field as a behavior variant selector.
Derived From	U_INSTRUCTION_SET_01

ID	F_INSTR_VARIANT_02
Description	Instructions encoded using the R-type shall use the `func7` field as a secondary behavior variant selector.
Derived From	U_INSTRUCTION_SET_01

Opcodes

Table 4 outlines the different opcodes values of the RV32I instruction set. Cells marked as *noimp* are for opcodes that are not implemented in ECAP5-DPROC.

opcode[1:0]	11							
opcode[4:2] / opcode[6:5]	000	001	010	011	100	101	110	111
00	LOAD	*noimp*	*noimp*	MISC-MEM	OP-IMM	AUIPC	*noimp*	*noimp*
01	STORE	*noimp*	*noimp*	*noimp*	OP	LUI	*noimp*	*noimp*
10	*noimp*	*noimp*	*noimp*	*noimp*	*noimp*	*noimp*	*noimp*	*noimp*
11	BRANCH	JALR	*noimp*	JAL	SYSTEM	*noimp*	*noimp*	*noimp*

Table 4: Opcode values for the RV32I instruction set.

ID	F_OPCODE_ENCODING_01
Description	Instructions which use the LUI opcode shall be decoded as an U-type instruction.
Derived From	U_INSTRUCTION_SET_01

ID	F_OPCODE_ENCODING_02
Description	Instructions which use the AUIPC opcode shall be decoded as an U-type instruction.
Derived From	U_INSTRUCTION_SET_01

ID	F_OPCODE_ENCODING_03
Description	Instructions which use the JAL opcode shall be decoded as a J-type instruction.
Derived From	U_INSTRUCTION_SET_01

ID	F_OPCODE_ENCODING_04
Description	Instructions which use the JALR opcode shall be decoded as an I-type instruction.
Derived From	U_INSTRUCTION_SET_01

ID	F_OPCODE_ENCODING_05
Description	Instructions which use the BRANCH opcode shall be decoded as a B-type instruction.
Derived From	U_INSTRUCTION_SET_01

ID	F_OPCODE_ENCODING_06
Description	Instructions which use the LOAD opcode shall be decoded as an I-type instruction.
Derived From	U_INSTRUCTION_SET_01

ID	F_OPCODE_ENCODING_07
Description	Instructions which use the STORE opcode shall be decoded as a S-type instruction.
Derived From	U_INSTRUCTION_SET_01

ID	F_OPCODE_ENCODING_08
Description	Instructions which use the OP-IMM opcode shall be decoded as an I-type instruction.
Derived From	U_INSTRUCTION_SET_01

ID	F_OPCODE_ENCODING_09
Description	Instructions which use the OP opcode shall be decoded as a R-type instruction.
Derived From	U_INSTRUCTION_SET_01

ID	F_OPCODE_ENCODING_10
Description	Instructions which use the MISC-MEM opcode shall be decoded as an I-type instruction.
Derived From	U_INSTRUCTION_SET_01

ID	F_OPCODE_ENCODING_11
Description	Instructions which use the SYSTEM opcode shall be decoded as an I-type instruction.
Derived From	U_INSTRUCTION_SET_01

3.2.3 Instructions behaviors

LUI

ID	F_LUI_01
Description	The LUI behavior shall be applied when the opcode is LUI.
Derived From	U_INSTRUCTION_SET_01

ID	F_LUI_02
Description	The result of LUI shall be the value of its first parameter.
Rationale	The LUI instruction shall load the 20 upper bits of the instruction immediate into the destination register and fill the remaining bits with zeros. This is the default behavior for instruction immediates as stated in F_INSTR_IMMEDIATE_02 and F_INSTR_IMMEDIATE_03.
Derived From	U_INSTRUCTION_SET_01

AUIPC

ID	F_AUIPC_01
Description	The AUIPC behavior shall be applied when the opcode is AUIPC.
Derived From	U_INSTRUCTION_SET_01

ID	F_AUIPC_02
Description	The result of AUIPC shall be the sum of its first parameter and the address of the AUIPC instruction.
Derived From	U_INSTRUCTION_SET_01

JAL

ID	F_JAL_01
Description	The JAL behavior shall be applied when the opcode is JAL.
Derived From	U_INSTRUCTION_SET_01

ID	F_JAL_02
Description	The `pc` register shall be updated with the sum of the address of the JAL instruction with the first instruction parameter.
Derived From	U_INSTRUCTION_SET_01

ID	F_JAL_03
Description	The result of JAL shall be the address of the JAL instruction incremented by 4.
Rationale	The JAL instruction shall output the address to the following instruction for it to be used as a *return address* in the case of a function call.
Derived From	U_INSTRUCTION_SET_01

JALR

ID	F_JALR_01
Description	The JALR behavior shall be applied when the opcode is JALR and func3 is 0x0.
Derived From	U_INSTRUCTION_SET_01

ID	F_JALR_02
Description	The `pc` register shall be updated with the sum of the first and second parameters of the JALR instruction.
Derived From	U_INSTRUCTION_SET_01

ID	F_JALR_03
Description	The result of JALR shall be the address of the JALR instruction incremented by 4.
Rationale	The JALR instruction shall output the address to the following instruction for it to be used as a *return address* in the case of a function call.
Derived From	U_INSTRUCTION_SET_01

BEQ

ID	F_BEQ_01
Description	The BEQ behavior shall be applied when the opcode is BRANCH and func3 is 0x0.
Derived From	U_INSTRUCTION_SET_01

ID	F_BEQ_02
Description	When the first and second instruction parameters are equal, the `pc` register shall be updated with the signed sum of the address of the BEQ instruction with the third parameter.
Derived From	U_INSTRUCTION_SET_01

BNE

ID	F_BNE_01
Description	The BNE behavior shall be applied when the opcode is BRANCH and func3 is 0x1.
Derived From	U_INSTRUCTION_SET_01

ID	F_BNE_02
Description	When the first and second parameters are not equal, the `pc` register shall be updated with the signed sum of the address of the BNE instruction with the third parameter.
Derived From	U_INSTRUCTION_SET_01

BLT

ID	F_BLT_01
Description	The BLT behavior shall be applied when the opcode is BRANCH and func3 is 0x4.
Derived From	U_INSTRUCTION_SET_01

ID	F_BLT_02
Description	When the first parameter is lower than the second parameter using a signed comparison, the `pc` register shall be updated with the signed sum of the address of the BLT instruction with the third parameter.
Derived From	U_INSTRUCTION_SET_01

BGE

ID	F_BGE_01
Description	The BGE behavior shall be applied when the opcode is BRANCH and func3 is 0x5.
Derived From	U_INSTRUCTION_SET_01

ID	F_BGE_02
Description	When the first parameter is greater or equal to the second parameter using a signed comparison, the `pc` register shall be updated with the signed sum of the address of the BGE instruction with the third parameter.
Derived From	U_INSTRUCTION_SET_01

BLTU

ID	F_BLTU_01
Description	The BLTU behavior shall be applied when the opcode is BRANCH and func3 is 0x6.
Derived From	U_INSTRUCTION_SET_01

ID	F_BLTU_02
Description	When the first parameter is lower than the second parameter using an unsigned comparison, the `pc` register shall be updated with the signed sum of the address of the BLTU instruction with the third parameter.
Derived From	U_INSTRUCTION_SET_01

BGEU

ID	F_BGEU_01
Description	The BGEU behavior shall be applied when the opcode is BRANCH and func3 is 0x7.
Derived From	U_INSTRUCTION_SET_01

ID	F_BGEU_02
Description	When the first parameter is greater or equal to the second parameter using an unsigned comparison, the `pc` register shall be updated with the signed sum of the address of the BGEU instruction with the third parameter.
Derived From	U_INSTRUCTION_SET_01

LB

ID	F_LB_01
Description	The LB behavior shall be applied when the opcode is LOAD and func3 is 0x0.
Derived From	U_INSTRUCTION_SET_01

ID	F_LB_02
Description	The result of LB shall be the 8-bit value stored in memory at the address determined by the signed sum of its first and second parameters.
Derived From	U_INSTRUCTION_SET_01

ID	F_LB_03
Description	The remaining bits of the loaded value shall be filled with the value of its 7^{th} bit.
Derived From	U_INSTRUCTION_SET_01

LH

ID	F_LH_01
Description	The LH behavior shall be applied when the opcode is LOAD and func3 is 0x1.
Derived From	U_INSTRUCTION_SET_01

ID	F_LH_02
Description	The result of LH shall be the 16-bit value stored in memory at the address determined by the signed sum of its first and second parameters.
Derived From	U_INSTRUCTION_SET_01

ID	F_LH_03
Description	The remaining bits of the loaded value shall be filled with the value of its 15th bit.
Derived From	U_INSTRUCTION_SET_01

LW

ID	F_LW_01
Description	The LW behavior shall be applied when the opcode is LOAD and func3 is 0x2.
Derived From	U_INSTRUCTION_SET_01

ID	F_LW_02
Description	The result of LW shall be the 32-bit value stored in memory at the address determined by the signed sum of its first and second parameters.
Derived From	U_INSTRUCTION_SET_01

LBU

ID	F_LBU_01
Description	The LBU behavior shall be applied when the opcode is LOAD and func3 is 0x4.
Derived From	U_INSTRUCTION_SET_01

ID	F_LBU_02
Description	The result of LBU shall be the 8-bit value stored in memory at the address determined by the signed sum of its first and second parameters.
Derived From	U_INSTRUCTION_SET_01

ID	F_LBU_03
Description	The remaining bits of the loaded value shall be filled with zeros.
Derived From	U_INSTRUCTION_SET_01

LHU

ID	F_LHU_01
Description	The LHU behavior shall be applied when the opcode is LOAD and func3 is 0x5.
Derived From	U_INSTRUCTION_SET_01

ID	F_LHU_02
Description	The result of LHU shall be the 16-bit value stored in memory at the address determined by the signed sum of its first and second parameters.
Derived From	U_INSTRUCTION_SET_01

ID	F_LHU_04
Description	The remaining bits of the loaded value shall be filled with zeros.
Derived From	U_INSTRUCTION_SET_01

SB

ID	F_SB_01
Description	The SB behavior shall be applied when the opcode is STORE and func3 is 0x0.
Derived From	U_INSTRUCTION_SET_01

ID	F_SB_02
Description	The lowest byte of the second parameter of SB shall be stored in memory at the address determined by the signed sum of its first and third parameters.
Derived From	U_INSTRUCTION_SET_01

SH

ID	F_SH_01
Description	The SH behavior shall be applied when the opcode is STORE and func3 is 0x1.
Derived From	U_INSTRUCTION_SET_01

ID	F_SH_02
Description	The two lowest bytes of the second parameter of SB shall be stored in memory at the address determined by the signed sum of its first and third parameters.
Derived From	U_INSTRUCTION_SET_01

SW

ID	F_SW_01
Description	The SW behavior shall be applied when the opcode is STORE and func3 is 0x2.
Derived From	U_INSTRUCTION_SET_01

ID	F_SW_02
Description	The value of the second parameter of SB shall be stored in memory at the address determined by the signed sum of its first and third parameters.
Derived From	U_INSTRUCTION_SET_01

ADDI

ID	F_ADDI_01
Description	The ADDI behavior shall be applied when the opcode is OP-IMM and when func3 is 0x0.
Derived From	U_INSTRUCTION_SET_01

ID	F_ADDI_02
Description	The result of ADDI shall be the signed integer sum of its two parameters.
Derived From	U_INSTRUCTION_SET_01

ID	F_ADDI_03
Description	The result of ADDI shall be truncated to 32-bits.
Derived From	U_INSTRUCTION_SET_01

SLTI

ID	F_SLTI_01
Description	The SLTI behavior shall be applied when the opcode is OP-IMM and when func3 is 0x2.
Derived From	U_INSTRUCTION_SET_01

ID	F_SLTI_02
Description	The result of SLTI shall be 1 when the signed value of its first parameter is lower that the signed value of its second parameter. It shall be 0 otherwise.
Derived From	U_INSTRUCTION_SET_01

SLTIU

ID	F_SLTIU_01
Description	The SLTIU behavior shall be applied when the opcode is OP-IMM and when func3 is 0x3.
Derived From	U_INSTRUCTION_SET_01

ID	F_SLTIU_02
Description	The result of SLTI shall be 1 when the unsigned value of its first parameter is lower that the unsigned value of its second parameter. It shall be 0 otherwise.
Derived From	U_INSTRUCTION_SET_01

XORI

ID	F_XORI_01
Description	The XORI behavior shall be applied when the opcode is OP-IMM and when func3 is 0x4.
Derived From	U_INSTRUCTION_SET_01

ID	F_XORI_02
Description	The result of XORI shall be the result of a bitwise xor between its two parameters.
Derived From	U_INSTRUCTION_SET_01

ORI

ID	F_ORI_01
Description	The ORI behavior shall be applied when the opcode is OP-IMM and when func3 is 0x6.
Derived From	U_INSTRUCTION_SET_01

ID	F_ORI_02
Description	The result of ORI shall be the result of a bitwise or between its two parameters.
Derived From	U_INSTRUCTION_SET_01

ANDI

ID	F_ANDI_01
Description	The ANDI behavior shall be applied when the opcode is OP-IMM and when func3 is 0x7.
Derived From	U_INSTRUCTION_SET_01

ID	F_ANDI_02
Description	The result of ANDI shall be the result of a bitwise and between its two parameters.
Derived From	U_INSTRUCTION_SET_01

SLLI

ID	F_SLLI_01
Description	The SLLI behavior shall be applied when the opcode is OP-IMM and func3 is 0x1.
Derived From	U_INSTRUCTION_SET_01

ID	F_SLLI_02
Description	The result of SLLI shall be its first parameter shifted left by the amount specified by the first 5 bits of its second parameter.
Derived From	U_INSTRUCTION_SET_01

ID	F_SLLI_03
Description	Zeros shall be inserted in the lower bits when shifting.
Derived From	U_INSTRUCTION_SET_01

SRLI

ID	F_SRLI_01
Description	The SRLI behavior shall be applied when the opcode is OP-IMM, func3 is 0x5 and the 30^{th} bit of its second input is 0.
Derived From	U_INSTRUCTION_SET_01

ID	F_SRLI_02
Description	The result of SRLI shall be its first parameter shifted right by the amount specified by the first 5 bits of its second parameter.
Derived From	U_INSTRUCTION_SET_01

ID	F_SRLI_03
Description	Zeros shall be inserted in the upper bits when shifting.
Derived From	U_INSTRUCTION_SET_01

SRAI

ID	F_SRAI_01
Description	The SRAI behavior shall be applied when the opcode is OP-IMM, func3 is 0x5 and the 30^{th} bit of its second input is 1.
Derived From	U_INSTRUCTION_SET_01

ID	F_SRAI_02
Description	The result of SRAI shall be its first parameter shifted right by the amount specified by the first 5 bits of its second parameter.
Derived From	U_INSTRUCTION_SET_01

ID	F_SRAI_03
Description	The most significant bit of the first parameter shall be inserted in the upper bits when shifting.
Derived From	U_INSTRUCTION_SET_01

ADD

ID	F_ADD_01
Description	The ADD behavior shall be applied when the opcode is OP, func3 is 0x0 and func7 is 0x0.
Derived From	U_INSTRUCTION_SET_01

ID	F_ADD_02
Description	The result of ADD shall be the signed integer sum of its two parameters.
Derived From	U_INSTRUCTION_SET_01

ID	F_ADD_03
Description	The result of ADD shall be truncated to 32-bits.
Derived From	U_INSTRUCTION_SET_01

SUB

ID	F_SUB_01
Description	The SUB behavior shall be applied when the opcode is OP, func3 is 0x0 and func7 is 0x20.
Derived From	U_INSTRUCTION_SET_01

ID	F_SUB_02
Description	The result of SUB shall be the signed integer difference of its first parameter minus its second parameter.
Derived From	U_INSTRUCTION_SET_01

ID	F_SUB_03
Description	The result of SUB shall be truncated to 32-bits.
Derived From	U_INSTRUCTION_SET_01

SLL

ID	F_SLL_01
Description	The SLL behavior shall be applied when the opcode is OP and func3 is 0x1.
Derived From	U_INSTRUCTION_SET_01

ID	F_SLL_02
Description	The result of SLL shall be its first parameter shifted left by the amount specified by the first 5 bits of its second parameter.
Derived From	U_INSTRUCTION_SET_01

ID	F_SLL_03
Description	Zeros shall be inserted in the lower bits when shifting.
Derived From	U_INSTRUCTION_SET_01

SLT

ID	F_SLT_01
Description	The SLT behavior shall be applied when the opcode is OP and func3 is 0x2.
Derived From	U_INSTRUCTION_SET_01

ID	F_SLT_02
Description	The result of SLT shall be 1 when the signed value of its first parameter is lower that the signed value of its second parameter. It shall be 0 otherwise.
Derived From	U_INSTRUCTION_SET_01

SLTU

ID	F_SLTU_01
Description	The SLTU behavior shall be applied when the opcode is OP and func3 is 0x3.
Derived From	U_INSTRUCTION_SET_01

ID	F_SLTU_02
Description	The result of SLTU shall be 1 when the unsigned value of its first parameter is lower that the unsigned value of its second parameter. It shall be 0 otherwise.
Derived From	U_INSTRUCTION_SET_01

XOR

ID	F_XOR_01
Description	The XOR behavior shall be applied when the opcode is OP and func3 is 0x4.
Derived From	U_INSTRUCTION_SET_01

ID	F_XOR_02
Description	The result of XOR shall be the result of a bitwise xor between its two parameters.
Derived From	U_INSTRUCTION_SET_01

SRL

ID	F_SRL_01
Description	The SRL behavior shall be applied when the opcode is OP, func3 is 0x5 and func7 is 0x0.
Derived From	U_INSTRUCTION_SET_01

ID	F_SRL_02
Description	The result of SRL shall be its first parameter shifted right by the amount specified by the first 5 bits of its second parameter.
Derived From	U_INSTRUCTION_SET_01

ID	F_SRL_03
Description	Zeros shall be inserted in the upper bits when shifting.
Derived From	U_INSTRUCTION_SET_01

SRA

ID	F_SRA_01
Description	The SRA behavior shall be applied when the opcode is OP, func3 is 0x5 and func7 is 0x20.
Derived From	U_INSTRUCTION_SET_01

ID	F_SRA_02
Description	The result of SRA shall be its first parameter shifted right by the amount specified by the first 5 bits of its second parameter.
Derived From	U_INSTRUCTION_SET_01

ID	F_SRA_03
Description	The most significant bit of the first parameter shall be inserted in the upper bits when shifting.
Derived From	U_INSTRUCTION_SET_01

OR

ID	F_OR_01
Description	The OR behavior shall be applied when the opcode is OP and func3 is 0x6.
Derived From	U_INSTRUCTION_SET_01

ID	F_OR_02
Description	The result of OR shall be the result of a bitwise or between its two parameters.
Derived From	U_INSTRUCTION_SET_01

AND

ID	F_AND_01
Description	The AND behavior shall be applied when the opcode is OP and func3 is 0x7.
Derived From	U_INSTRUCTION_SET_01

ID	F_AND_02
Description	The result of AND shall be the result of a bitwise and between its two parameters.
Derived From	U_INSTRUCTION_SET_01

FENCE TBC

ECALL TBC

EBREAK TBC

3.2.4 Exceptions

ID	F_INSTR_ADDR_MISALIGNED_01
Description	An Instruction Address Misaligned exception shall be raised when the target address of a taken branch or an unconditional jump if not four-byte aligned.
Derived From	U_INSTRUCTION_SET_01

ID	F_MISALIGNED_MEMORY_ACCESS_01
Description	A Misaligned Memory Access exception shall be raised when the target address of a load/store instruction is not aligned on the referenced type size.
Derived From	U_INSTRUCTION_SET_01

3.2.5 Memory interface

Memory accesses

ID	F_MEMORY_INTERFACE_01
Description	Both instruction and data accesses shall be handled by a unique external memory interface.
Derived From	U_MEMORY_INTERFACE_01

Wishbone protocol

The following requirements are extracted from the Wishbone specification.

ID	F_WISHBONE_DATASHEET_01
Description	The memory interface shall comply with the Wishbone Datasheet provided in section 2.1.
Derived From	U_MEMORY_INTERFACE_02

ID	F_WISHBONE_RESET_01
Description	The memory interface shall initialize itself at the rising edge of `wb_clk_i` following the assertion of `wb_rst_i`.
Derived From	U_MEMORY_INTERFACE_02

ID	F_WISHBONE_RESET_02
Description	The memory interface shall stay in the initialization state until the rising edge of `wb_clk_i` following the deassertion of `wb_rst_i`.
Derived From	U_MEMORY_INTERFACE_02

ID	F_WISHBONE_RESET_03
Description	Signals `wb_stb_o` and `wb_cyc_o` shall be deasserted while the memory interface is in the initialization state. The state of all other memory interface signals are undefined in response to a reset cycle.
Derived From	U_MEMORY_INTERFACE_02

ID	F_WISHBONE_TRANSFER_CYCLE_01
Description	The memory interface shall assert `wb_cyc_o` for the entire duration of the memory access.
Rationale	TBC what wb_cyc_o does.
Derived From	U_MEMORY_INTERFACE_02

ID	F_WISHBONE_TRANSFER_CYCLE_02
Description	Signal `wb_cyc_o` shall be asserted no later than the rising edge of `wb_clk_i` that qualifies the assertion of `wb_stb_o`.
Derived From	U_MEMORY_INTERFACE_02

ID	F_WISHBONE_TRANSFER_CYCLE_03
Description	Signal `wb_cyc_o` shall be deasserted no earlier than the rising edge of `wb_clk_i` that qualifies the deassertion of `wb_stb_o`.
Derived From	U_MEMORY_INTERFACE_02

ID	F_WISHBONE_ACK_01
Description	The memory interface shall operate normally when `ack_i` is held in the asserted state.
Derived From	U_MEMORY_INTERFACE_02

ID	F_WISHBONE_STB_01
Description	The following signals shall be valid when `stb_o` is asserted : `adr_o`, `dat_o`, `sel_o` and `we_o`.
Derived From	U_MEMORY_INTERFACE_02

ID	F_WISHBONE_CYCLES_01
Description	The memory interface shall implement the single read cycle as described in figure 3.
Derived From	U_MEMORY_INTERFACE_02

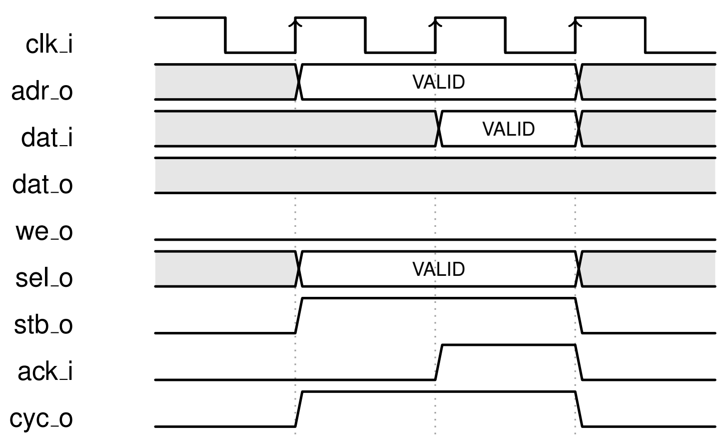


Figure 3: Timing diagram of the single read cycle of the wishbone memory interface

Table 5: Description of the single read cycle of the wishbone memory interface defined in figure 3.

CLOCK EDGE	DESCRIPTION
0	The memory interface presents a valid address on `adr_o` The memory interface deasserts `we_o` to indicate a READ cycle The memory interface presents a bank select `sel_o` to indicate where it expects data. The memory interface asserts `cyc_o` to indicate the start of the cycle. The memory interface asserts stb_o to indicate the start of the phase.
1	Valid data is provided on `dat_i`. `ack_i` is asserted to indicate valid data. It shall be noted that wait states may be inserted before asserting `ack_i`, thereby allowing it to throttle the cycle speed. Any number of wait states may be added.
2	The memory interface latches data on `dat_i`. The memory interface deasserts `stb_o` and `cyc_o` to indicate the end of the cycle. The `ack_i` signal is deasserted in response to the deassertion of `stb_o`.

ID	F_WISHBONE_CYCLES_02
Description	The memory interface shall implement the single write cycle as described in figure 4.
Derived From	U_MEMORY_INTERFACE_02

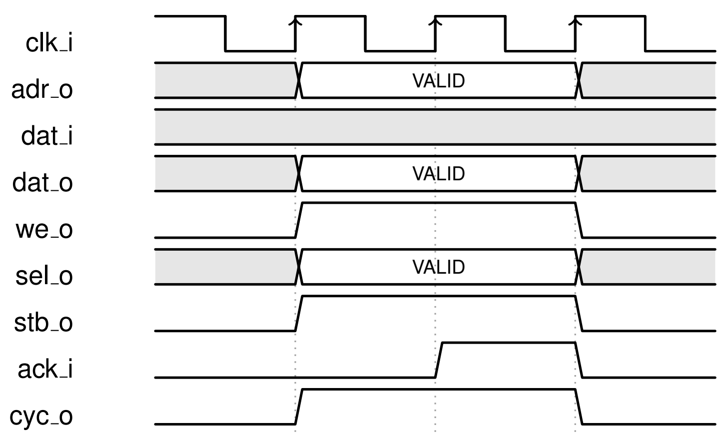


Figure 4: Timing diagram of the single write cycle of the wishbone memory interface

Table 6: Description of the single write cycle of the wishbone memory interface defined in figure 4.

CLOCK EDGE	DESCRIPTION
0	The memory interface presents a valid address on `adr_o` The memory interface presents valid data on `dat_o` The memory interface asserts `we_o` to indicate a WRITE cycle The memory interface presents a bank select `sel_o` to indicate where it sends data. The memory interface asserts `cyc_o` to indicate the start of the cycle. The memory interface asserts stb_o to indicate the start of the phase.
1	`ack_i` is asserted in response to `stb_o` to indicate latched data. It shall be noted that wait states may be inserted before asserting `ack_i`, thereby allowing it to throttle the cycle speed. Any number of wait states may be added.
2	The memory interface deasserts `stb_o` and `cyc_o` to indicate the end of the cycle. The `ack_i` signal is deasserted in response to the deassertion of `stb_o`.

ID	F_WISHBONE_TIMING_01
Description	The clock input `clk_i` shall coordinate all activites for the internal logic within the memory interface. All output signals of the memory interface shall be registered at the rising edge of `clk_i`. All input signals of the memory interface shall be stable before the rising edge of `clk_i`.
Rationale	As long as the memory interface is designed within the clock domain of `clk_i`, the requirement will be satisfied by using the place and route tool.
Derived From	U_MEMORY_INTERFACE_02

Caches

TBC Mention about no cache in revision 1.0.0

3.2.6 Debugging

TBC

3.3 Nonfunctional Requirements

ID	N_FORMAL_PROOF_01
Description	Each part of ECAP5-DPROC shall be formally proven when possible, otherwise thouroughly tested

These can be : performance, safety, security, usability, scalability.

4 Configuration

4.1 Constants

Table 7: Constants used for implementing the RISC-V ISA

NAME	TYPE	WIDTH	DESCRIPTION	VALUE

4.2 Instanciation parameters

ECAP5-DPROC can be parameterized at build-time through instanciation parameters. Default constant values are defined for such parameters.

Table 8: Instanciation parameters of ECAP5-DPROC

NAME	TYPE	WIDTH	DESCRIPTION	DEFAULT VALUE

5 Architecture overview

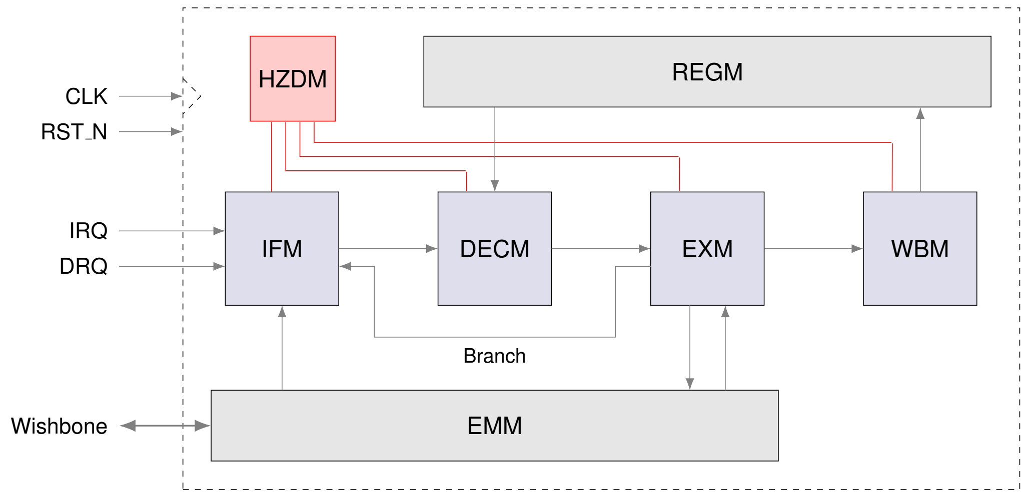


Figure 5: Schematic view of the architecture of ECAP5-DPROC

5.1 Clock domains

To simplify the design of revision 1.0.0, each module of ECAP5-DPROC belong to a unique clock domain.

5.2 Pipeline stages

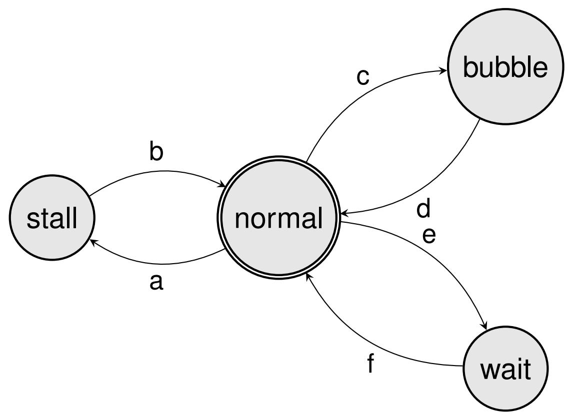
ECAP5-DPROC is built around a pipelined architecture with the following stages :

- The instruction fetch stage loads the next instruction from memory.
- The decode stage handles the instruction decoding to provide the next stage with the different instruction input values including reading from internal registers.
- The execute stage implements all arithmetic and logic operations. This includes load and store operations to the memory.
- The write-back stage which handles storing instructions outputs to internal registers.

Considering the load-store architecture of the RISC-V instruction set, the choice was made, for revision 1.0.0, to include the memory stage of the typical 5-stage pipeline within the execute stage. This will decrease the latency while keeping a similar throughput, as any memory access will inevitably produce a pipeline stall as of revision 1.0.0.

5.2.1 Pipeline stall

In order to handle pipeline stalls, a handshaking mechanism is implemented between each stages, allowing the execution flow to be stopped. A stall can be either triggered by a stage itself or requested by the hazard module.



a : stage stalls, b : stage unstalls, c : input valid = 0, d : input valid = 1, e : output ready = 0, f : output valid = 1,

Figure 6: State diagram of the operating modes of pipeline stages

Pipeline stages located at the start and end of the pipeline do not implement the bubble and wait modes respectively.

The following points describe the behavior of the different modes :

- A stage in **normal** mode shall operate as described by its different functional behaviors.
- A stage in **stall** mode shall deassert its input ready signal and output valid signal while waiting to unstall.
- A stage in **bubble** mode shall operate as normal but taking a nop instruction as input instead of the data provided by the preceding stage.
- A stage in **wait** mode shall deassert its input ready signal and wait until going back to normal mode.

In case of a stall, the stalling stage deasserts its input ready signal leading to preceding stages waiting for completion. The stalling stage deasserts its output valid signal leading to following stages taking a bubble as their input.

The figure 7 is a diagram of the stall behavior on a 5-stage pipeline. By stalling the 3rd stage, this example provides a representative visualisation of all the stalling cases of a 4-stage pipeline.

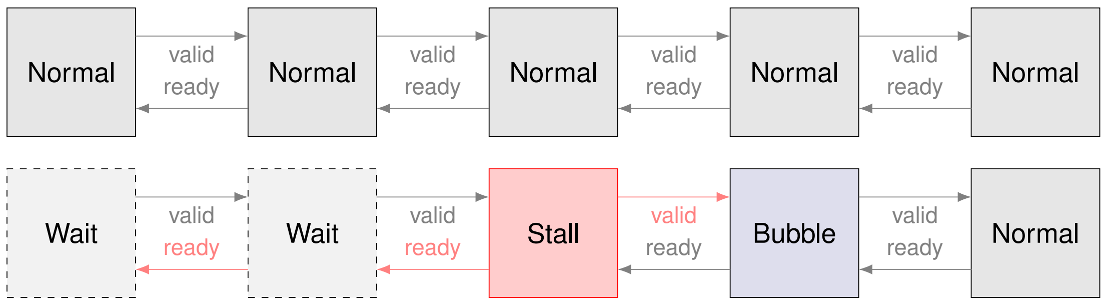


Figure 7: Diagram of a pipeline stall behavior on a 6-stage pipeline

Figure 8 outlines the resolution of a pipeline stall on stage 3. By stalling the 3rd stage, this example provides a representative visualisation of all the stalling cases of a 4-stage pipeline.

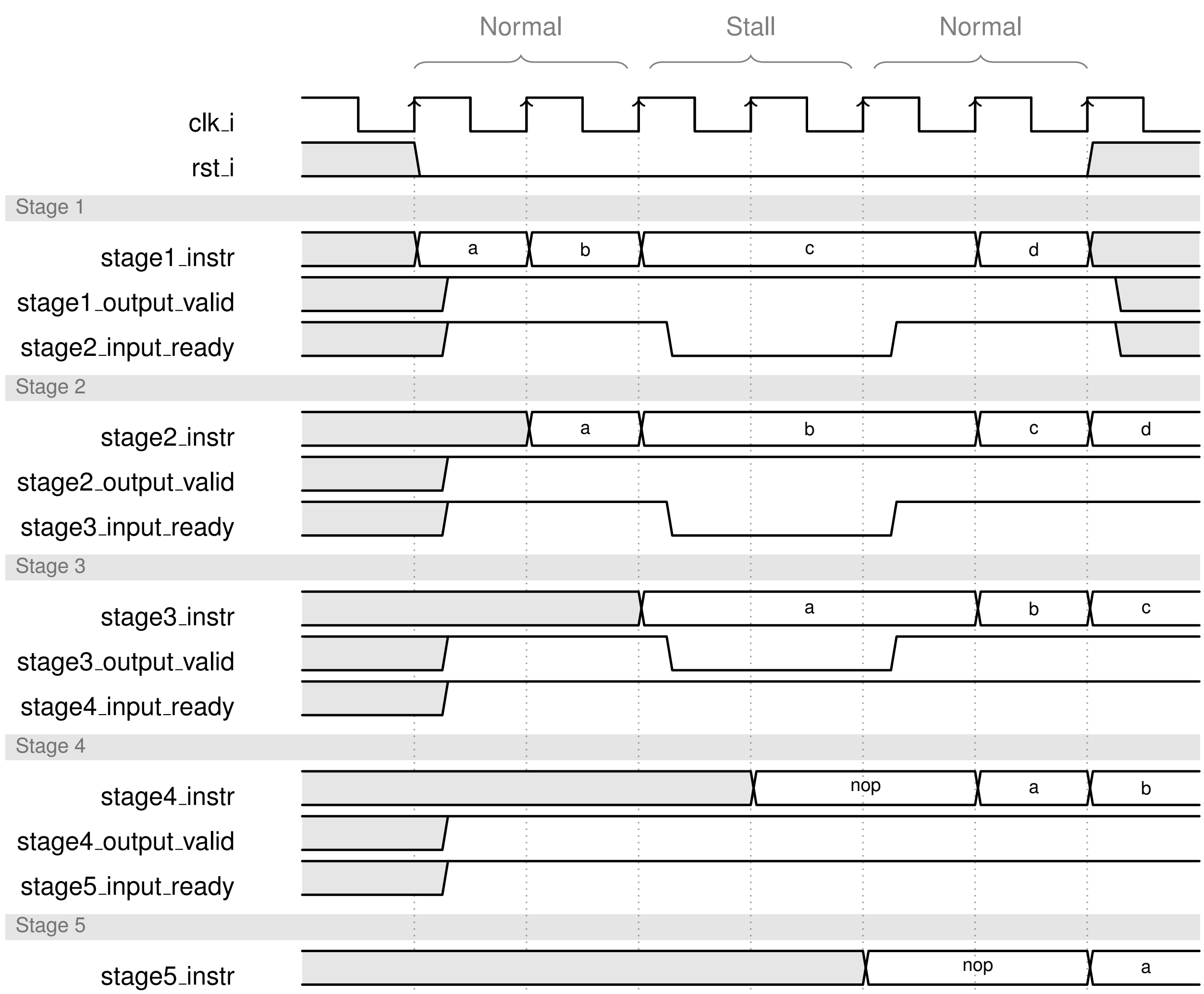


Figure 8: Timing diagram of a pipeline stall resolution behavior on a 6-stage pipeline

5.3 Hazard management

5.3.1 Structural hazard

For the scope of this document, are designated as structural hazards all cases when a stage is unable to finish its processing within the required time before the next clock cycle.

A pipeline stall is produced in case of structural hazards. It shall be noted that the some of the performance impact of this kind of hazard could be mitigated but this feature is not included in revision 1.0.0.

5.3.2 Data hazard

A data hazard occurs when an instruction (A) uses the result of a previous instruction (B) which is still being processed in the pipeline.

A pipeline stall is produced in case of data hazards so that B is able to finish before A uses its result. It shall be noted that some of the performance impact of this kind of hazard could be mitigated but this feature is not included in revision 1.0.0.

5.3.3 Control hazard

A control hazard occurs when a jump or branch instruction is executed, as instructions following the jump/branch are already being processes through the pipeline when the jump/branch happens.

Instructions following the jump/branch are replaced by a nop instruction through the use of the bubble mode of the pipeline stages. This operation is designated as *bubble drop*. It shall be noted that some of the performance impact of this kind of hazard could be mitigated but this feature is not included in revision 1.0.0.

5.4 Functional partitioning

The design is split into the following functional modules :

- **External Memory Module** (EMM) is in charge of accessing memory and peripherals.
- **Instruction Fetch Module** (IFM) is in charge of implementing the instruction fetch stage.
- **Decode Module** (DECM) is in charge of implementing the decode stage.
- **Register Module** (REGM) implements the internal registers.
- **Execute Module** (EXM) is in charge of implementing the execute stage.
- **Write-Back Module** (WBM) is in charge of implementing the write-back stage.

- **Hazard Module** (HZDM) handles the detecting of data and control hazards as well as triggering associated pipeline stalls and bubble drops.

6 Register Module

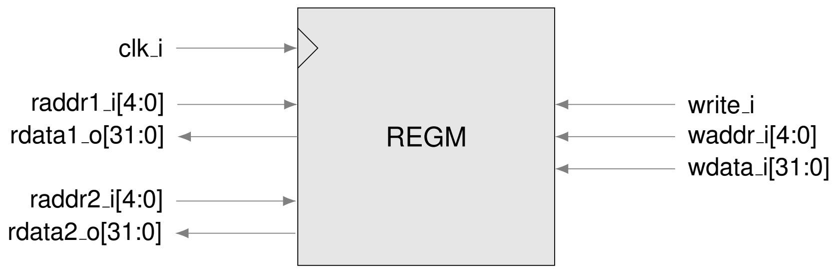


Figure 9: Schematic view of the Register Module

6.1 Interface

The register module implements the 32 internal registers of ECAP5-DPROC. It has two reading port and one writing port. The signals are described in table 9.

Table 9: Register Module interface signals

NAME	TYPE	WIDTH	DESCRIPTION
clk_i	I	1	Clock input.
FIRST READING PORT			
raddr1_i	I	5	Register selector.
rdata1_o	O	32	Selected register value.
SECOND READING PORT			
raddr2_i	I	5	Register selector.
rdata2_o	O	32	Selected register value.
WRITING PORT			
waddr_i	I	5	Register selector.
write_i	I	1	Asserted to indicate a write.
wdata_i	I	32	Data to be written.

6.2 Specification

6.2.1 Upstream requirements

The table 10 outlines the upstream requirements applicable to the Register Module.

Table 10: Upstream requirements applicable to the External Memory Module

ID
F_REGISTERS_01
F_REGISTERS_02

6.2.2 Functional requirements

ID	D_REGM_REGISTERS_01
Description	The register module shall implement 32 general purpose registers ranging from `x0` to `x31`.
Derived From	F_REGISTERS_01

ID	D_REGM_READ_PORT_01
Description	The register module shall provide two asynchronous reading ports for accessing registers from `x0` to `x31`.
Derived From	F_REGISTERS_01

ID	D_REGM_WRITE_PORT_01
Description	The register module shall provide one synchronous writing port for modifying registers from `x0` to `x31`.
Derived From	F_REGISTERS_01

ID	D_REGM_WRITE_PORT_02
Description	When both reading and writing to the same registers, the read operation shall be performed before the write operation.
Derived From	F_REGISTERS_01

ID	D_REGM_WRITE_PORT_03
Description	The value of register `x0` shall not be changed during a write operation, remaining the constant zero.
Derived From	F_REGISTERS_02

6.3 Behavior

6.3.1 Read behavior

When reading, `rdata1_i` and `rdata2_i` output, on the rising edge of `clk_i`, the value of the register respectively selected by `raddr1_i` and `raddr2_i`.

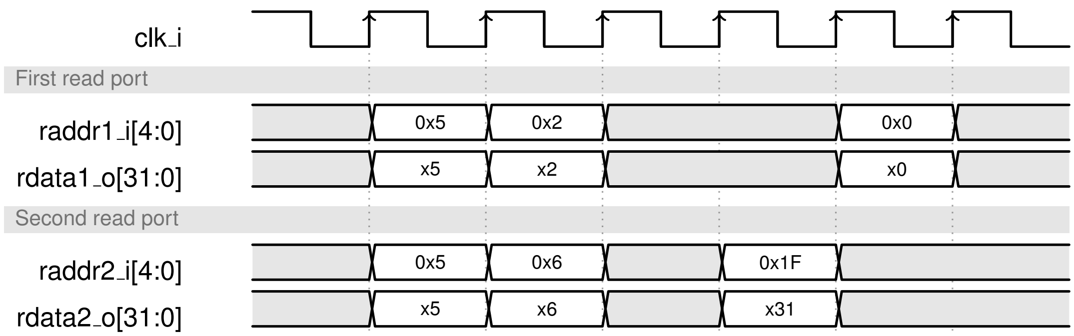


Figure 10: Timing diagram of the read behavior of the register module

6.3.2 Write behavior

A register write happens on the rising edge of `clk_i` when `write_i` is asserted, writing the value `wdata_i` in the register selected by `waddr_i`.

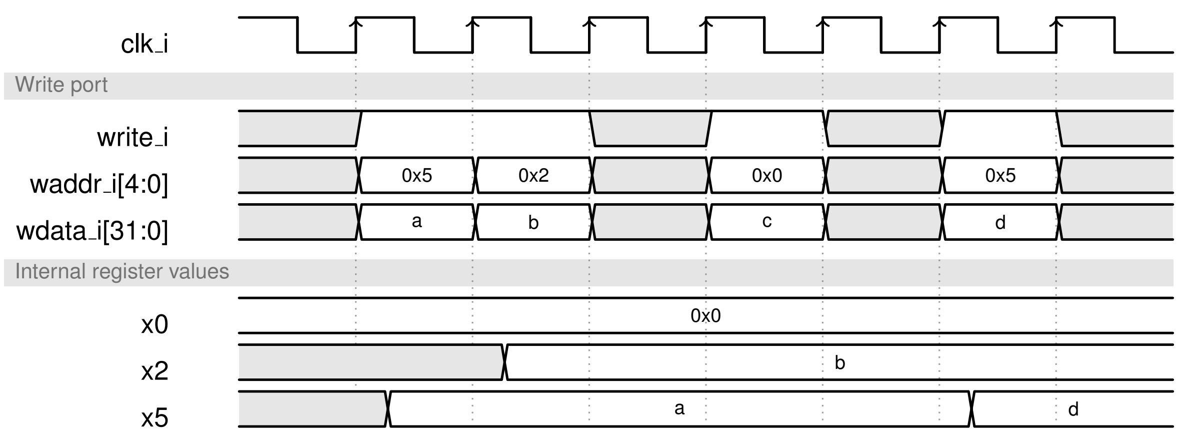


Figure 11: Timing diagram of the write behavior of the register module

6.3.3 Read-before-write behavior

When both read and write operations are requested on the same register at the same time, the write operation happens during the next clock cycle. Considering read requests are performed during the second stage of the pipeline while write requests are performed during the fourth stage, this behavior reduces the potential hazards induces by the pipelined architecture.

Figure 12 provides an example with a read request on the first port, although this behavior also applies to the second reading port.

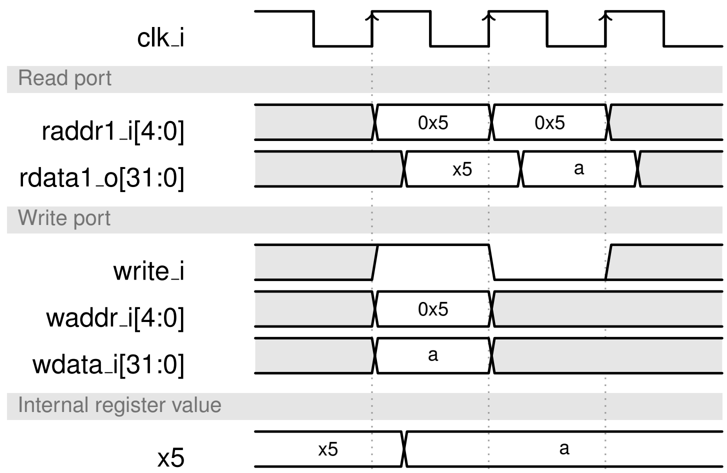


Figure 12: Timing diagram of the read-before-write behavior of the register module

7 External Memory Module

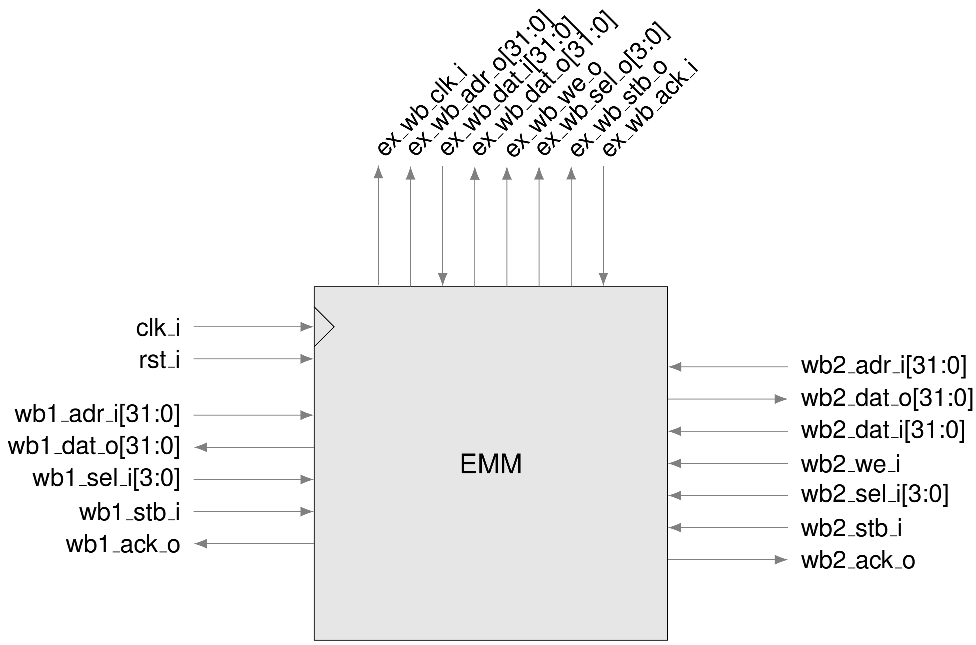


Figure 13: Schematic view of the External Memory Module

7.1 Interface

The external memory module implements the memory arbiter necessary to have process the two wishbone slave memory requests through a unique external memory interface. The signals are described in table 11.

Table 11: External Memory Module interface signals

NAME	TYPE	WIDTH	DESCRIPTION
clk_i	I	1	Clock input.
rst_i	I	1	Reset input.
FIRST WISHBONE SLAVE			
wb1_adr_i	I	32	Wishbone read address.
wb1_dat_o	O	32	Wishbone read data.
wb1_sel_i	I	4	Wishbone byte selector.
wb1_stb_i	I	1	Wishbone handshaking signal asserted when emitting a request.
wb1_ack_o	O	1	Acknowledge. Indicates a normal termination of a bus cycle.

SECOND WISHBONE SLAVE			
wb2_adr_i	I	32	Wishbone read address.
wb2_dat_o	O	32	Wishbone read data.
wb2_dat_i	I	32	Wishbone write data.
wb2_we_i	I	1	Wishbone write enable.
wb2_sel_i	I	4	Wishbone byte selector.
wb2_stb_i	I	1	Wishbone handshaking signal asserted when emitting a request.
wb2_ack_o	O	1	Acknowledge. Indicates a normal termination of a bus cycle.
EXTERNAL MEMORY ACCESS			
ex_wb_clk_i	I	1	TBC
ex_wb_adr_o	O	32	Wishbone read address.
ex_wb_dat_i	I	32	Wishbone read data.
ex_wb_dat_o	O	32	Wishbone write data.
ex_wb_we_o	O	1	Wishbone write enable.
ex_wb_sel_o	O	4	Wishbone byte selector.
ex_wb_stb_o	O	1	Wishbone handshaking signal asserted when emitting a request.
ex_wb_ack_i	I	1	Acknowledge. Indicates a normal termination of a bus cycle.

7.2 Specification

7.2.1 Upstream requirements

Table 12 outlines the upstream requirements applicable to the External Memory Module.

Table 12: Upstream requirements applicable to the External Memory Module

ID
F_MEMORY_INTERFACE_01
F_WISHBONE_DATASHEET_01
F_WISHBONE_RESET_01
F_WISHBONE_RESET_02
F_WISHBONE_RESET_03
F_WISHBONE_TRANSFER_CYCLE_01

F_WISHBONE_TRANSFER_CYCLE_02
F_WISHBONE_TRANSFER_CYCLE_03
F_WISHBONE_ACK_01
F_WISHBONE_STB_01
F_WISHBONE_CYCLES_01
F_WISHBONE_CYCLES_02
F_WISHBONE_TIMING_01

7.2.2 Functional requirements

7.3 Behavior

TBC

8 Instruction Fetch Module

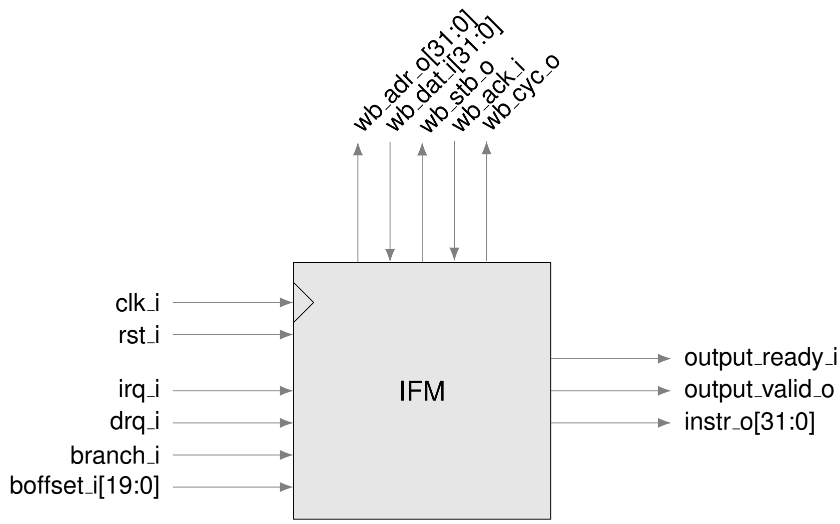


Figure 14: Schematic view of the Instruction Fetch Module

8.1 Interface

The instruction fetch module handles fetching from memory the instructions to be executing. The signals are described in table 13.

Table 13: Instruction Fetch Module interface signals

NAME	TYPE	WIDTH	DESCRIPTION
clk_i	I	1	Clock input.
rst_i	I	1	Reset input.
JUMP LOGIC			
irq_i	I	1	External interrupt request.
drq_i	I	1	External debug request.
branch_i	I	1	Branch request.
boffset_i	I	20	Branch offset from the pc of the instruction in the execute stage (pc - 8).
WISHBONE MASTER			
wb_adr_o	O	32	Wishbone read address.
wb_dat_i	I	32	Wishbone read data.
wb_sel_o	O	4	Wishbone byte selector.

wb_stb_o	O	1	Wishbone handshaking signal asserted when emitting a request.
wb_ack_i	I	1	Acknowledge. Indicates a normal termination of a bus cycle.
OUTPUT LOGIC			
output_ready_i	I	1	Output handshaking signal asserted when the destination is ready to receive the output
output_valid_o	O	1	Output handshaking signal asserted when the output is valid.
instr_o	O	32	Instruction output.

8.2 Specification

8.2.1 Upstream requirements

The table 14 outlines the upstream requirements applicable to the Instruction Fetch Module.

Table 14: Upstream requirements applicable to the Instruction Fetch Module

ID
F_MEMORY_INTERFACE_01

8.2.2 Functional requirements

8.3 Behavior

This module doesn't contain any prefetch mechanism as there is no performance requirement for version 1.0.0. This will lead to a performance bottleneck due to the number of cycles needed for fetching instructions from memory.

8.3.1 Normal behavior

During normal operation, the instruction fetch module sequentially emits memory requests to address stored in the `pc` register. Upon successful request, `pc` is incremented to point to the following instruction.

This behavior induces a pipeline stall due to the delay of the wishbone interface.

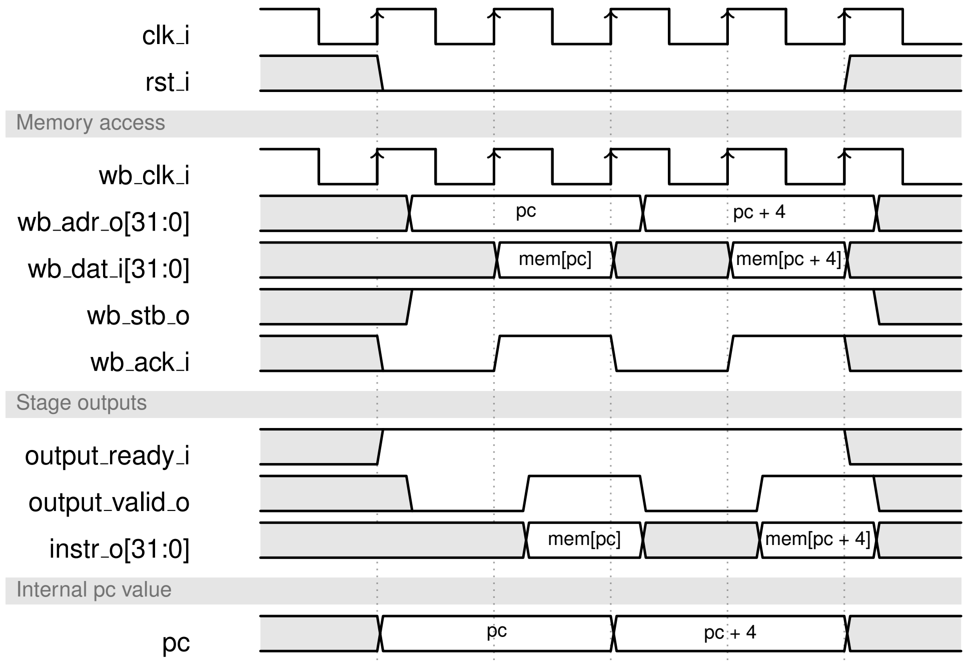


Figure 15: Timing diagram of the normal behavior of the instruction fetch module

8.3.2 Reset behavior

Once a reset occurs, `pc` is loaded with the boot address before returning to normal operation.

This behavior doesn't induce any pipeline stall *per se* as the pipeline is reset during this operation.

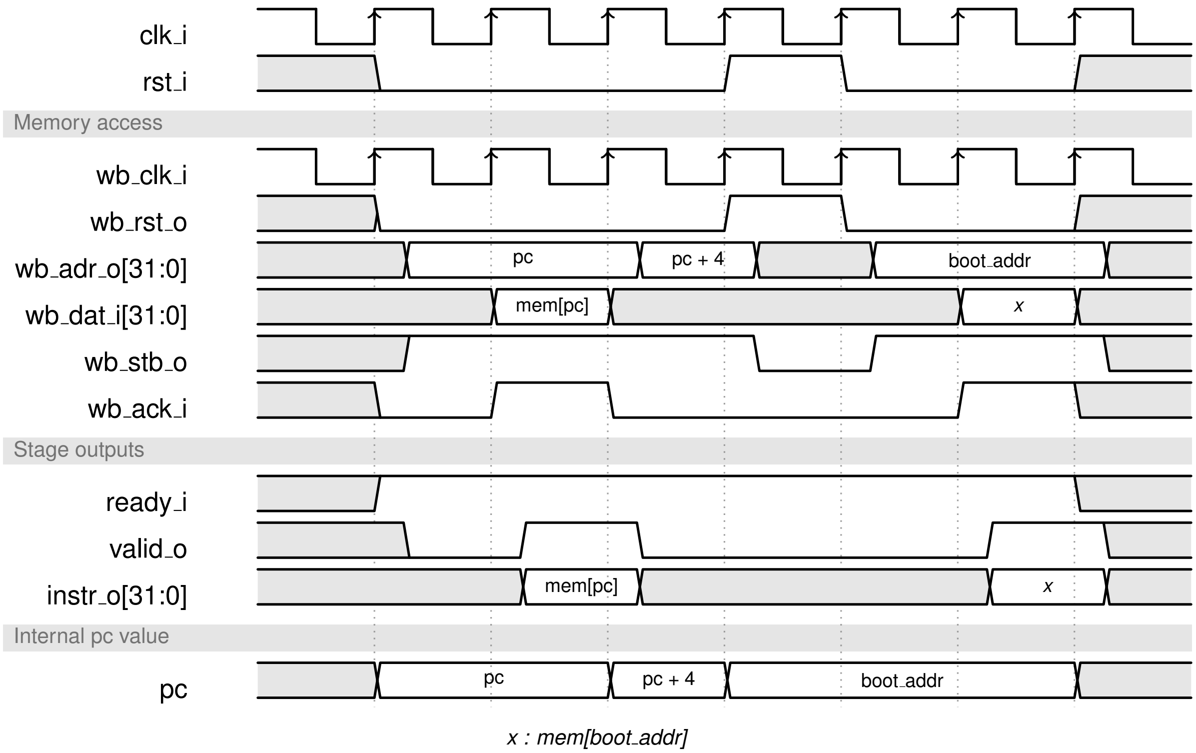


Figure 16: Timing diagram of the reset behavior of the instruction fetch module

8.3.3 Resource busy behavior

The instruction fetch module is capable of handling wait states from memory through the stalling mechanism.

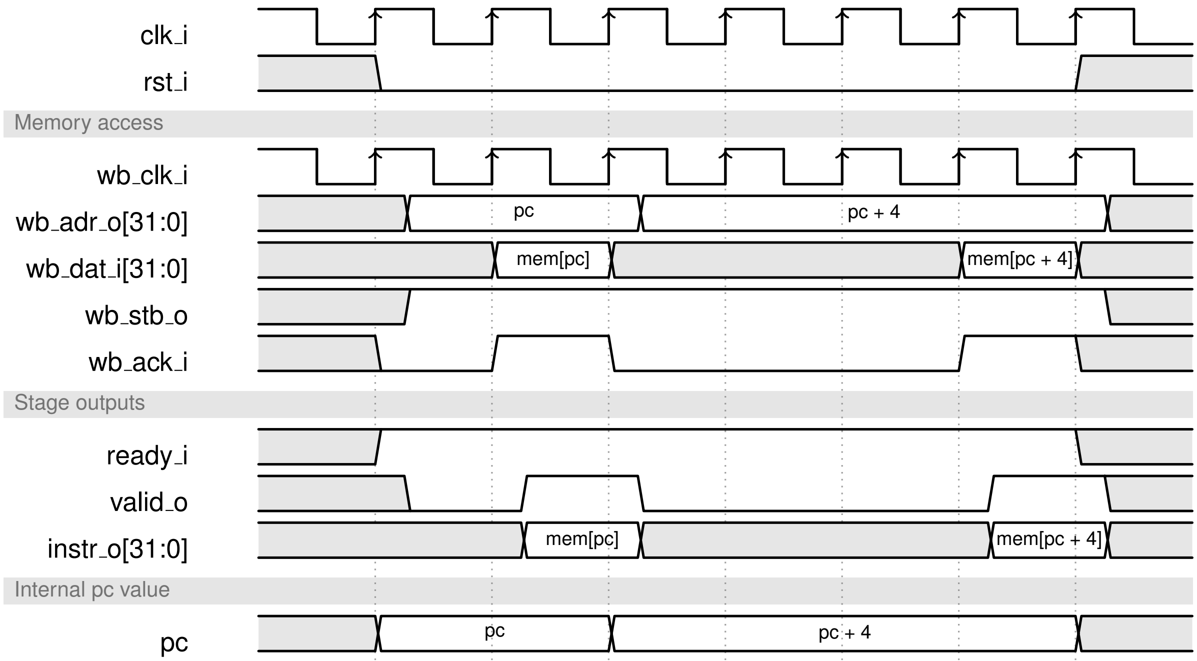


Figure 17: Timing diagram of the memory resource busy behavior of the instruction fetch module

8.3.4 Jump behavior

TBC

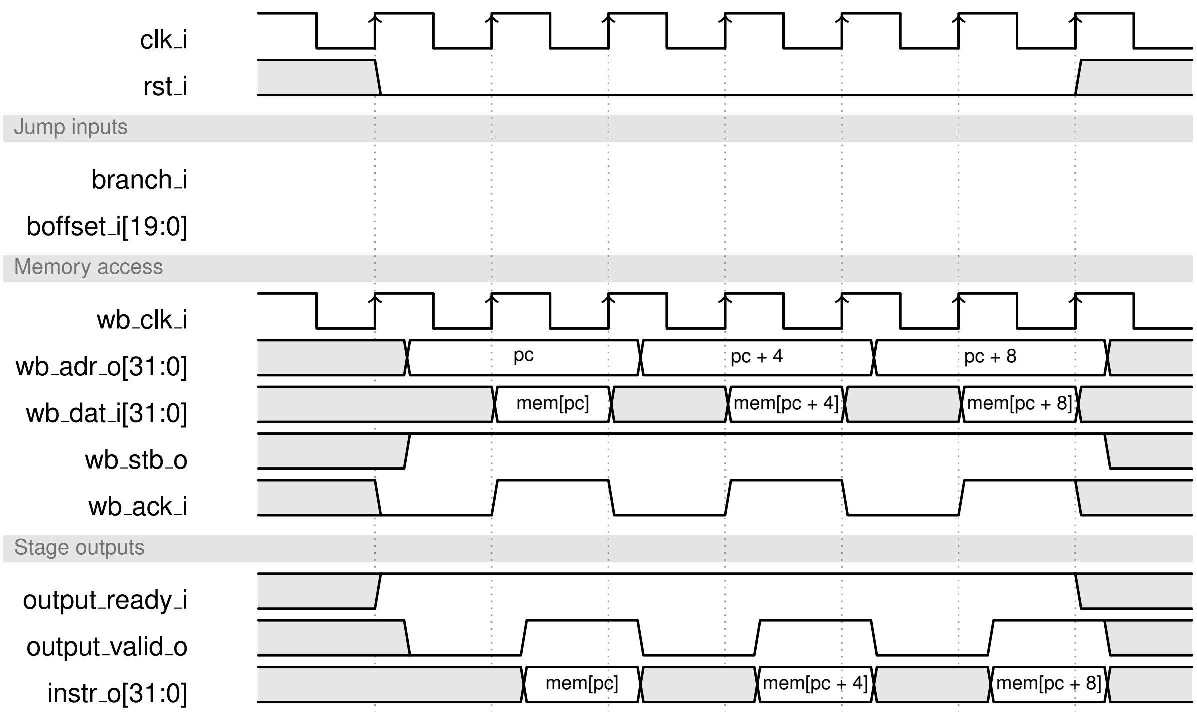


Figure 18: Timing diagram of the jump behavior of the instruction fetch module for branch events

Figure 19: Timing diagram of the jump behavior of the instruction fetch module for interrupt events

8.3.5 Hazard behaviors

Hazard behaviors are described in section 5.2.1.

9 Decode Module

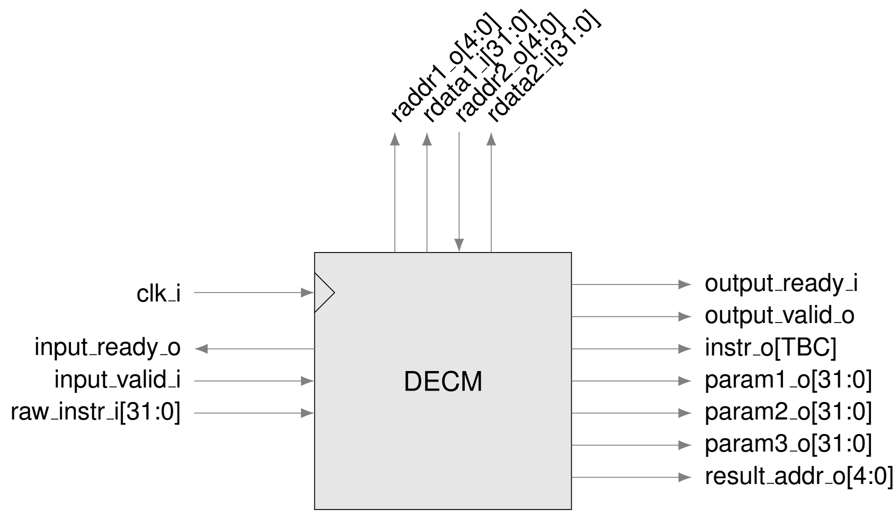


Figure 20: Schematic view of the Decode Module

9.1 Interface

The decode module implements TBC. The signals are described in table 15.

Table 15: Decode Module interface signals

NAME	TYPE	WIDTH	DESCRIPTION
clk_i	I	1	Clock input.
INPUT LOGIC			
input_ready_o	O	1	Input handshaking signal asserted when ready to receive inputs.
input_valid_i	I	1	Input handshaking signal asserted when the provided inputs are valid.
raw_instr_i	I	32	Raw instruction fetched from memory.
REGISTER ACCESS			
raddr1_o	O	5	Register selector for the first read port.
rdata1_i	I	32	Register value selected by raddr1_o.
raddr2_o	O	5	Register selector for the second read port.
rdata2_i	I	32	Register value selected by raddr2_o.
OUTPUT LOGIC			

output_ready_i	I	1	Output handshaking signal asserted when the destination is ready to receive the output.
output_valid_o	O	1	Output handshaking signal asserted when the output is valid.
instr_o	O	TBC	Decoded instruction.
param1_o	O	32	First instruction parameter.
param2_o	O	32	Second instruction parameter.
param3_o	O	32	Third instruction parameter.
result_addr_o	O	32	Instruction result destination address.

9.2 Specification

9.2.1 Upstream requirements

The table 16 outlines the upstream requirements applicable to the Decode Module.

Table 16: Upstream requirements applicable to the Decode Module

ID
F_INSTR_IMMEDIATE_01
F_INSTR_IMMEDIATE_02
F_INSTR_IMMEDIATE_03
F_INSTR_FIRST_PARAM_01
F_INSTR_FIRST_PARAM_02
F_INSTR_SECOND_PARAM_01
F_INSTR_SECOND_PARAM_02
F_INSTR_THIRD_PARAM_01
F_INSTR_RESULT_01
F_INSTR_RESULT_02
F_INSTR_VARIANT_01
F_INSTR_VARIANT_02
F_OPCODE_ENCODING_01
F_OPCODE_ENCODING_02
F_OPCODE_ENCODING_03
F_OPCODE_ENCODING_04
F_OPCODE_ENCODING_05

F_OPCODE_ENCODING_06
F_OPCODE_ENCODING_07
F_OPCODE_ENCODING_08
F_OPCODE_ENCODING_09
F_OPCODE_ENCODING_10
F_OPCODE_ENCODING_11

9.2.2 Functional requirements

9.3 Behavior

TBC

9.3.1 Hazard behaviors

Hazard behaviors are described in section 5.2.1.

10 Execute Module

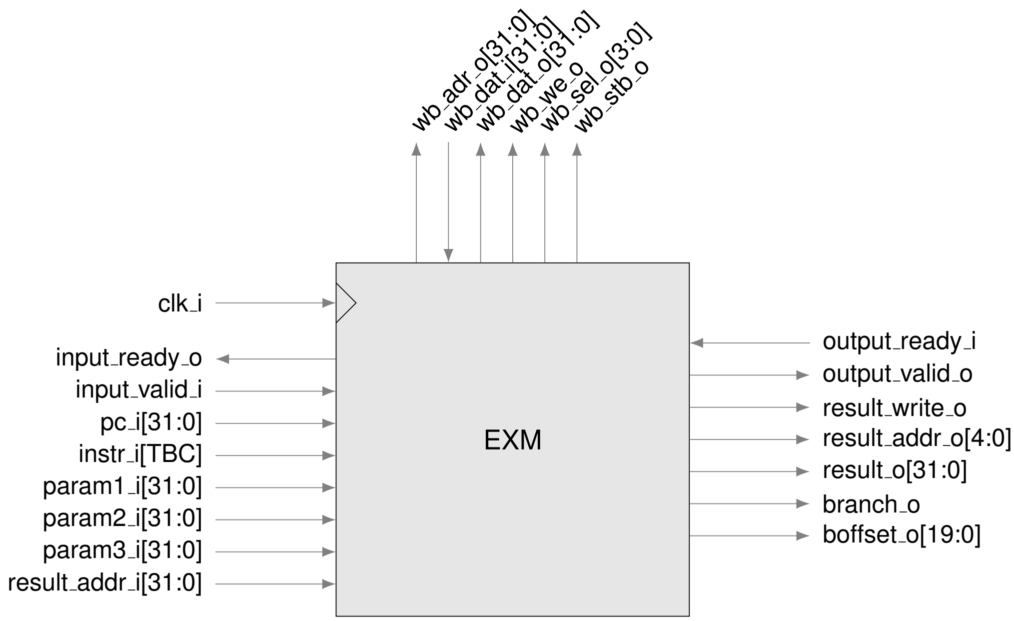


Figure 21: Schematic view of the Execute Module

10.1 Interface

The execute module implements TBC. The signals are described in table 17.

Table 17: Execute Module interface signals

NAME	TYPE	WIDTH	DESCRIPTION
clk_i	I	1	Clock input.
INPUT LOGIC			
input_ready_o	O	1	Input handshaking signal asserted when ready to receive inputs.
input_valid_i	I	1	Input handshaking signal asserted when the provided inputs are valid.
pc_i	I	32	Program counter of the current instruction.
instr_i	I	TBD	Current instruction.
param1_i	I	32	First instruction parameter.
param2_i	I	32	Second instruction parameter.
param3_i	I	32	Third instruction parameter.
WISHBONE MASTER			

wb_adr_o	O	32	Wishbone read address.
wb_dat_i	I	32	Wishbone read data.
wb_dat_o	O	32	Wishbone write data.
wb_we_o	O	1	Wishbone write-enable.
wb_sel_o	O	4	Wishbone byte selector.
wb_stb_o	O	1	Wishbone handshaking signal asserted when emitting a request.
wb_ack_i	I	1	Wishbone handshaking signal asserted when received data is valid.
OUTPUT LOGIC			
output_ready_i	I	1	Output handshaking signal asserted when the destination is ready to receive the output.
output_valid_o	O	1	Output handshaking signal asserted when the output is valid.
result_write_o	O	1	Asserted when the instruction shall store its result.
result_addr_o	O	5	Address of the register where the result shall be stored.
result_o	O	32	Instruction result.
branch_o	O	1	Branch request.
boffset_o	O	20	Branch offset from pc_i

10.2 Specification

10.2.1 Upstream requirements

The table 18 outlines the upstream requirements applicable to the Execute Module.

Table 18: Upstream requirements applicable to the Execute Module

ID
F_LUI_01
F_LUI_02
F_AUIPC_01
F_AUIPC_02
F_JAL_01
F_JAL_02
F_JAL_03
F_JALR_01

F_JALR_02
F_JALR_03
F_BEQ_01
F_BEQ_02
F_BNE_01
F_BNE_02
F_BLT_01
F_BLT_02
F_BGE_01
F_BGE_02
F_BLTU_01
F_BLTU_02
F_BGEU_01
F_BGEU_02
F_LB_01
F_LB_02
F_LB_03
F_LH_01
F_LH_02
F_LH_03
F_LW_01
F_LW_02
F_LBU_01
F_LBU_02
F_LBU_03
F_LHU_01
F_LHU_02
F_LHU_03
F_SB_01
F_SB_02
F_SH_01
F_SH_02
F_SW_01

F_SW_02
F_ADDI_01
F_ADDI_02
F_ADDI_03
F_SLTIU_01
F_SLTIU_02
F_XORI_01
F_XORI_02
F_ORI_01
F_ORI_02
F_ANDI_01
F_ANDI_02
F_SLLI_01
F_SLLI_02
F_SLLI_03
F_SRLI_01
F_SRLI_02
F_SRLI_03
F_SRAI_01
F_SRAI_02
F_SRAI_03
F_ADD_01
F_ADD_02
F_ADD_03
F_SUB_01
F_SUB_02
F_SUB_03
F_SLL_01
F_SLL_02
F_SLL_03
F_SLT_01
F_SLT_02
F_SLTU_01

F_SLTU_02
F_XOR_01
F_XOR_02
F_SRL_01
F_SRL_02
F_SRL_03
F_SRA_01
F_SRA_02
F_SRA_03
F_OR_01
F_OR_02
F_AND_01
F_AND_02

10.2.2 Functional requirements

10.3 Behavior

10.3.1 LUI behavior

The LUI behavior emits a register write request with the first input value. This value is the sign-extended immediate of the instruction, computed in the decode stage.

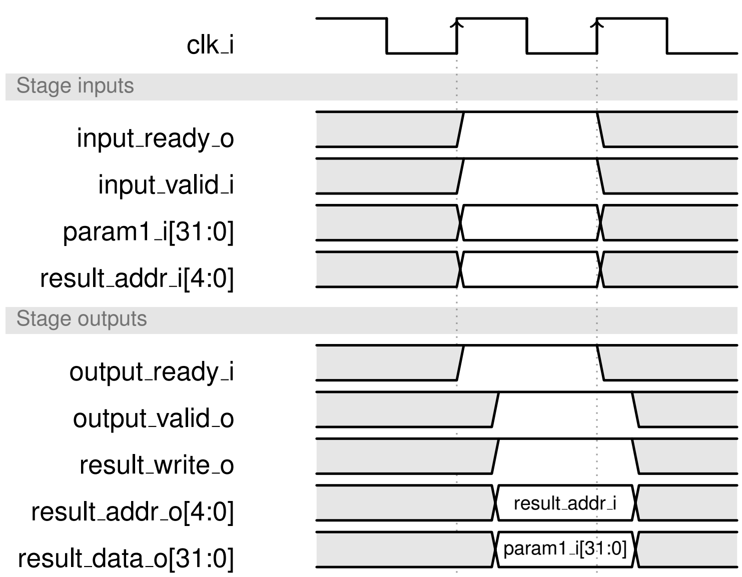


Figure 22: Timing diagram of the LUI behavior of the execute module

10.3.2 AUIPC behavior

The AUIPC behavior computes the sum of the first input with the program counter. This first input is the sign-extended immediate of the instruction, computed in the decode stage.

A register write request is emitted to store the value.

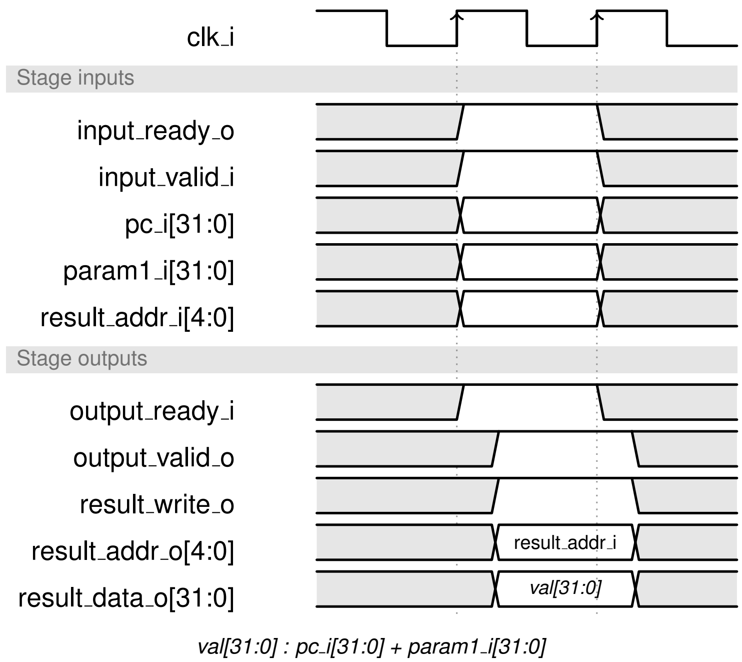


Figure 23: Timing diagram of the AUIPC behavior of the execute module

10.3.3 Unconditional jump behavior

JAL

The JAL behavior emits a branch request to the address offset provided by the first input.

A register write request is emitted to store the address of the following instruction.

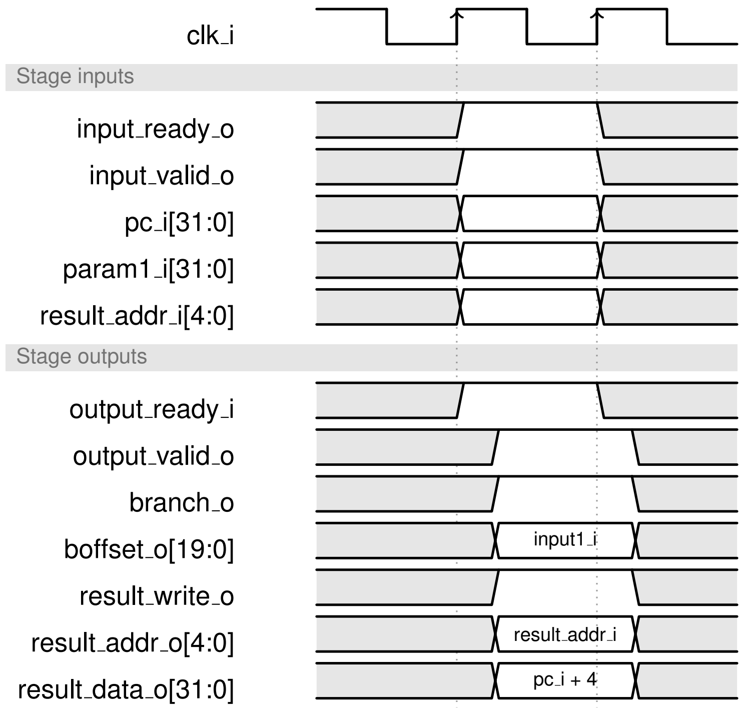


Figure 24: Timing diagram of the jump behavior of the execute module for the JAL instruction

JALR

The JALR behavior emits a branch request to the address offset provided by the sum of the first and second input.

A register write request is emitted to store the address of the following instruction.

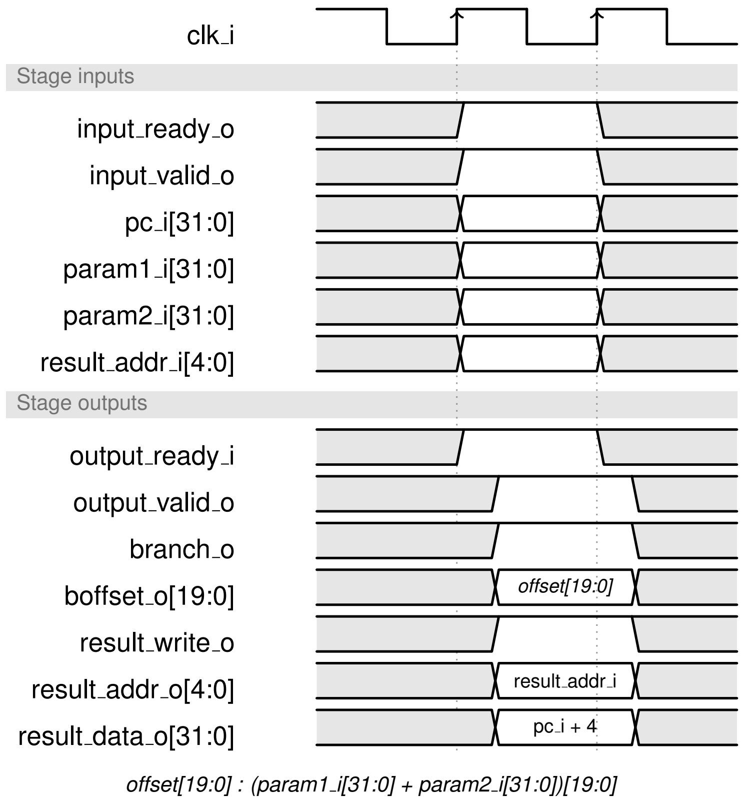


Figure 25: Timing diagram of the jump behavior of the execute module for the JALR instruction

10.3.4 Branch behavior

The branch behavior compares the first two inputs depending on the instruction variant. The BEQ and BNE variants check whether the two inputs are equal or not equal respectively. The BLT and BLTU variants check whether the first input is lower than the second input using a signed and unsigned comparison respectively. The BGE and BGEU variants check whether the first input is greater or equal to the second input using a signed and unsigned comparison respectively.

A branch request is emitted in case of successful comparison, providing the address offset given in the third input.

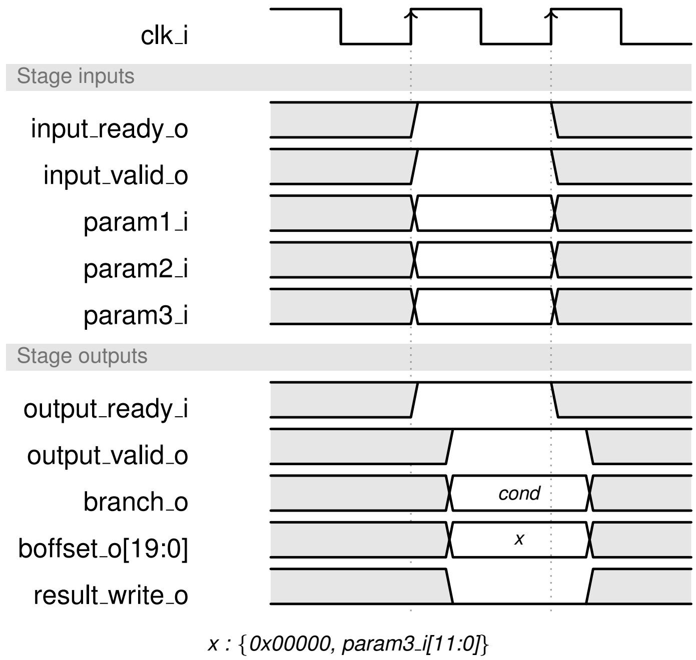


Figure 26: Timing diagram of the branch behavior of the execute module

Table 19: Description of the parameters of figure 26.

VARIANTS	PARAMETERS	DESCRIPTION
BEQ	cond = if(param1_i = param2_i) 1 else 0	1 when the first param is equal to the second param. 0 otherwise.
BNE	cond = if(param1_i = param2_i) 0 else 1	1 when the first param is not equal to the second param. 0 otherwise.
BLT	cond = if(param1_i $<$ param2_i) 1 else 0	1 when the first param is lower than the second param using a signed comparison. 0 otherwise.
BLTU	cond = if(unsigned(param1_i) $<$ unsigned(param2_i)) 1 else 0	1 when the first param is lower than the second param using an unsigned comparison. 0 otherwise.
BGE	cond = if(param1_i $\geq$ param2_i) 1 else 0	1 when the first param is greater or equal to the second param using a signed comparison. 0 otherwise.

BGEU	cond = if(unsigned(param1_i) $\geq$ unsigned(param2_i)) 1 else 0	1 when the first param is greater or equal to the second param using an unsigned comparison. 0 otherwise.

10.3.5 Load behavior

The load behavior fetches an 8/16/32-bit word from memory at the address given in the first instruction input for the LB/LBU, LH/LHU and LW instructions respectively. The value is zero-extended to 32-bits for the U variants while it is signed extended to 32-bits otherwise.

A register write request is then emitted to store the value.

This behavior induces a pipeline stall due to the memory access time.

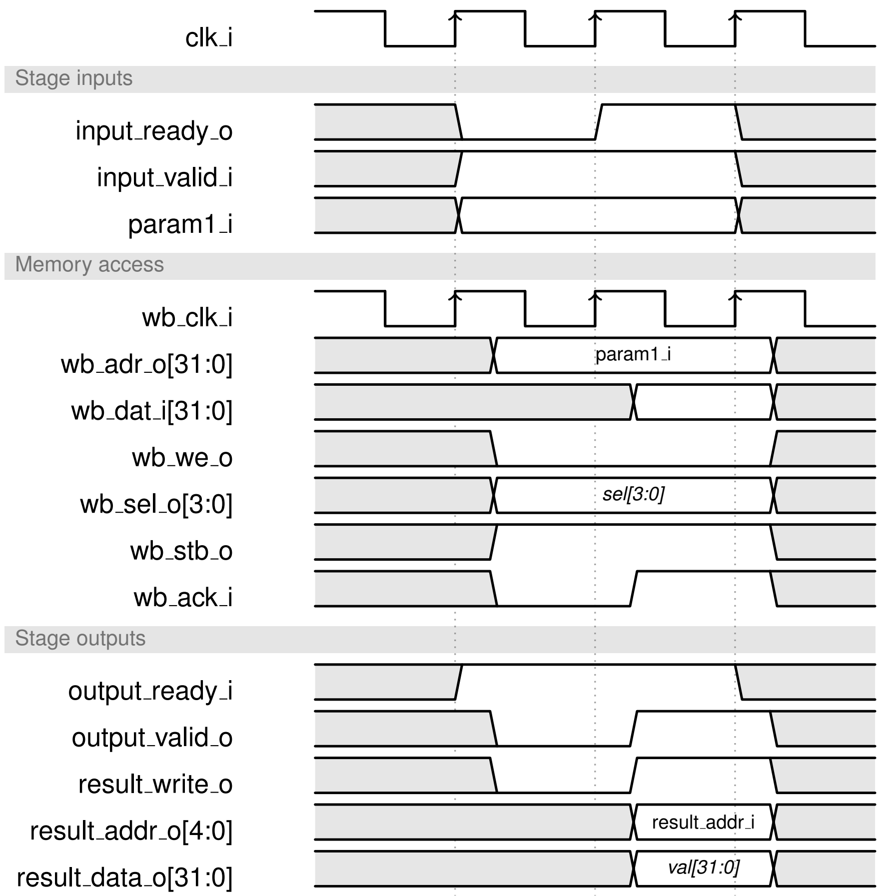


Figure 27: Timing diagram of the load behavior of the execute module

Table 20: Description of the parameters of figure 27.

VARIANTS	PARAMETERS	DESCRIPTION
LB	val[31:0] = {24{wb_data_i[7]}, wb_data_i[7:0]}	The sign-extended 8-bit data retrieved from memory.
	sel[3:0] = 0b0001	Only the lowest significant byte is selected.
LH	val[31:0] = {16{wb_data_i[15]}, wb_data_i[15:0]}	The sign-extended 16-bit data retrieved from memory.
	sel[3:0] = 0b0011	Only the two lowest significant bytes are selected.
LW	val[31:0] = wb_data_i[31:0]	The 32-bit data retrieved from memory.
	sel[3:0] = 0b1111	All bytes are selected.
LBU	val[31:0] = {0x000000, wb_data_i[7:0]}	The zero-extended 8-bit data retrieved from memory.
	sel[3:0] = 0b0001	Only the lowest significant byte is selected.
LHU	val[31:0] = {0x0000, wb_data_i[15:0]}	The zero-extended 16-bit data retrieved from memory.
	sel[3:0] = 0b0011	Only the two lowest significant bytes are selected.

10.3.6 Store behavior

The store behavior writes a 8/16/32-bit word to memory at the address given in the first instruction input for the SB, SH and SW instructions respectively.

Although there is no need for the pipeline to wait for the data to be written, this behavior induces a pipeline stall in revision 1.0.0 as a write request takes at least two cycles to complete while the memory interface shall be ready to receive new requests in the next pipeline cycle.

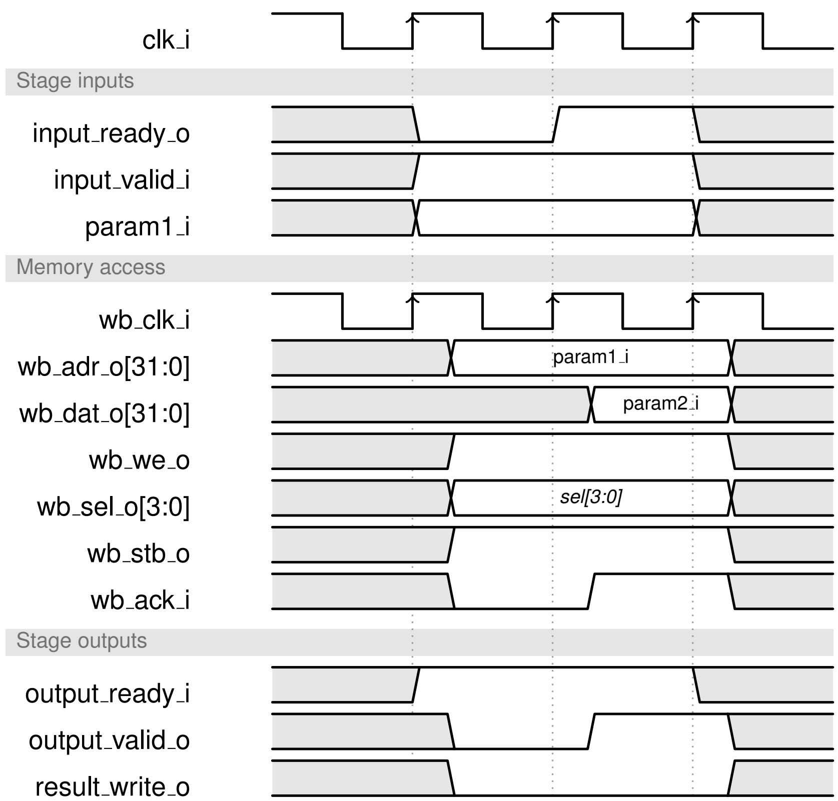


Figure 28: Timing diagram of the store behavior of the execute module

Table 21: Description of the parameters of figure 28.

VARIANTS	PARAMETERS	DESCRIPTION
SB	sel[3:0] = 0b0001	Only the lowest significant byte is selected.
SH	sel[3:0] = 0b0011	Only the two lowest significant bytes are selected.
SW	sel[3:0] = 0b1111	All the bytes are selected.

10.3.7 Arithmetic and logic behavior

The arithmetic and logic behavior applies to OP and OP-IMM opcodes. As DECM handles the processing of instruction inputs, both OP and OP-IMM instructions have the same associated behavior in EXM.

The arithmetic and logic behavior computes the following integer operations : signed sum, signed substraction, bitwise exclusive-or, bitwise or, bitwise and, signed comparison, unsigned comparison, left logical shift, right logical shift and right arithmetic shift.

A register write request is then emitted to store the value.

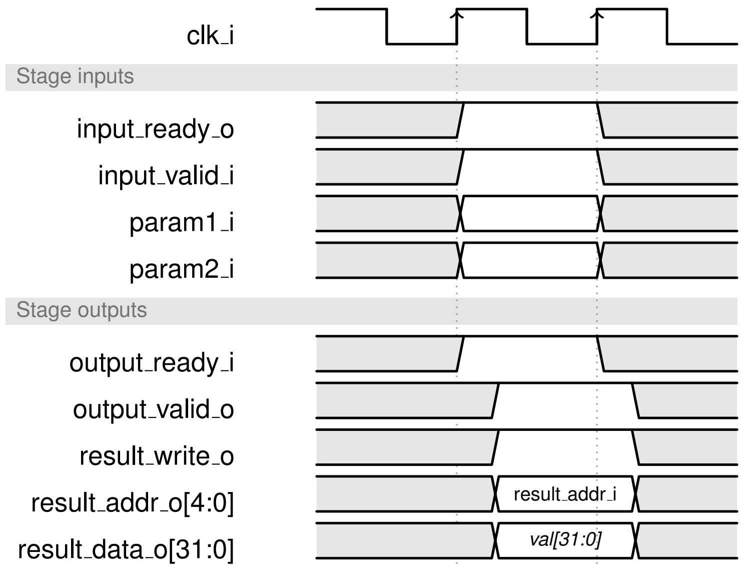


Figure 29: Timing diagram of the arithmetic and logic behavior of the execute module

Table 22: Description of the parameters of figure 29.

VARIANTS	PARAMETERS	DESCRIPTION
ADD/ADDI	val[31:0] = param1_i + param2_i	The signed sum of the first and second param.
SUB	val[31:0] = param1_i - param2_i	The signed difference of the first param minus the second param.
XOR/XORI	val[31:0] = param1_i $\oplus$ param2_i	The bitwise exclusive-or of the first and second params.
OR/ORI	val[31:0] = param1_i $\vee$ param2_i	The bitwise or of the first and second params.
AND/ANDI	val[31:0] = param1_i $\wedge$ param2_i	The bitwise and of the first and second params.

SLT/SLTI	val[31:0] = if(param1_i $<$ param2_i) 1 else 0	1 when the first param is lower than the second param using a signed comparison. 0 otherwise.
SLTU/SLTIU	val[31:0] = if(unsigned(param1_i) $<$ unsigned(param2_i)) 1 else 0	1 when the first param is lower than the second param using an unsigned comparison. 0 otherwise.
SLL/SLLI	val[31:0] = $\{$param1_i[31 - param2_i[4:0] : 0], param2_i[4:0]$\{0\}\}$	The first param is shifted left by the amount specified by the first 5 bits of the second param. The right bits are filled with zeros.
SRL/SRLI	val[31:0] = $\{$param2_i[4:0]$\{0\}$, param1_i[31 : param2_i[4:0]]$\}$	The first param is shifted right by the amount specified by the first 5 bits of the second param. The left bits are filled with zeros.
SRA/SRAI	val[31:0] = $\{$param2_i[4:0]$\{$param1_i[31]$\}$, param1_i[31 : param2_i[4:0]]$\}$	The first param is shifted right by the amount specified by the first 5 bits of the second param. The left bits are filled with the sign bit of the first param.

10.3.8 Hazard behaviors

Hazard behaviors are described in section 5.2.1.

11 Write-Back Module

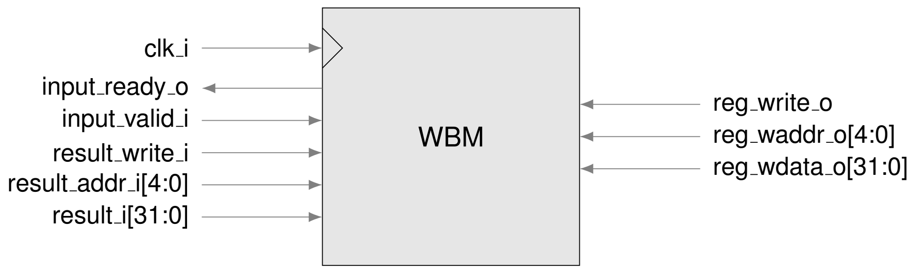


Figure 30: Schematic view of the Write-Back Module

11.1 Interface

The write-back module implements TBC. The signals are described in table 23.

Table 23: Write-back Module interface signals

NAME	TYPE	WIDTH	DESCRIPTION
clk_i	I	1	Clock input.
INPUT LOGIC			
input_ready_o	O	1	Input handshaking signal asserted when ready to receive inputs.
input_valid_i	I	1	Input handshaking signal asserted when the provided inputs are valid.
result_write_i	I	1	Asserted when the instruction shall output a value to a register.
result_addr_i	I	5	Address of the register to be written to.
result_i	I	32	Value to be written to the register.
REGISTER ACCESS			
reg_write_o	0	1	Register selector for the write port.
reg_waddr_o	O	5	Asserted when a write to the selected register shall happen.
reg_wdata_o	O	32	Data to be written to the selected register.

11.2 Specification

11.2.1 Upstream requirements

The table 24 outlines the upstream requirements applicable to the Write-Back Module.

Table 24: Upstream requirements applicable to the Write-Back Module

ID
F_INSTR_RESULT_02

11.2.2 Functional requirements

11.3 Behavior

11.3.1 Instruction with output behavior

TBC : reg port directly wired to stage ff

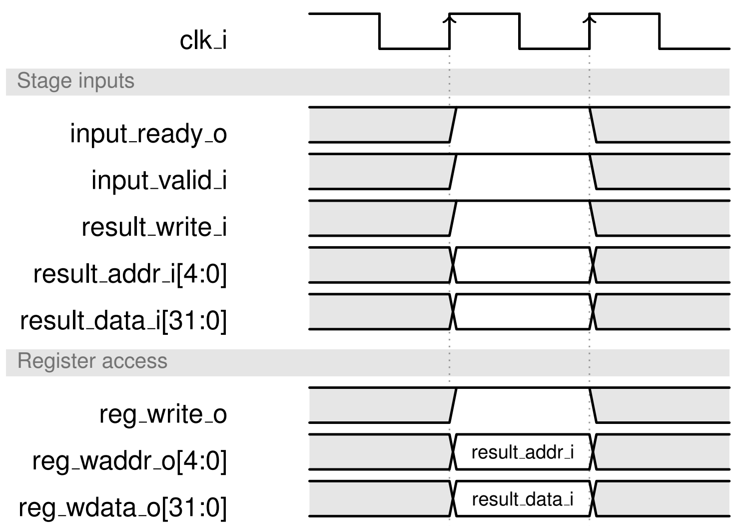


Figure 31: Timing diagram of the instruction with output behavior of the write-back module

11.3.2 Instruction without output behavior

TBC : reg port directly wired to stage ff

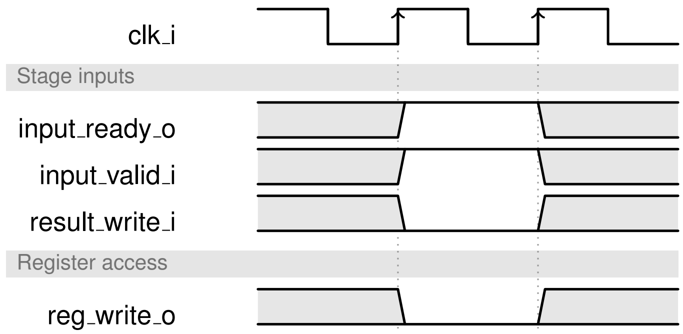


Figure 32: Timing diagram of the instruction without output behavior of the write-back module

11.3.3 Hazard behaviors

Hazard behaviors are described in section 5.2.1.

12 Harzard Module

Figure 33: Schematic view of the Hazard Module

12.1 Interface

The hazard module implements TBC. The signals are described in table 25.

Table 25: Hazard Module interface signals

NAME	TYPE	WIDTH	DESCRIPTION
clk_i	I	1	Clock input.

12.2 Specification

12.2.1 Upstream requirements

The table 26 outlines the upstream requirements applicable to the Hazard Module.

Table 26: Upstream requirements applicable to the Hazard Module

ID

12.2.2 Functional requirements

12.3 Behavior

12.3.1 Data hazard behavior

TBC

Figure 34: Timing diagram of the data-hazard behavior of the hazard module

12.3.2 Control hazard behavior

TBC

Figure 35: Timing diagram of the control hazard behavior of the hazard module

13 Debug