

**ECAP5-DPROC**

RISC-V processor

# Architecture Document

**Revision: DRAFT-1.0.0**

**Issue Date: June 4, 2023**

DRAFT

Copyright (C) Clément Chaine

*ECAP5-DPROC is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.*

*ECAP5-DPROC is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.*

*You should have received a copy of the GNU General Public License along with ECAP5-DPROC. If not, see <http://www.gnu.org/licenses/>*

---

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Purpose . . . . .	2
1.2	Intended Audience and Use . . . . .	2
1.3	Product Scope . . . . .	2
1.4	Definitions and Acronyms . . . . .	2
1.5	References . . . . .	2
<b>2</b>	<b>Overall Description</b>	<b>3</b>
2.1	User needs . . . . .	3
2.2	Assumptions and Dependencies . . . . .	4
<b>3</b>	<b>Requirements</b>	<b>4</b>
3.1	External Interface Requirements . . . . .	4
3.2	Functional Requirements . . . . .	6
3.2.1	Register file . . . . .	6
3.2.2	Instruction decoding . . . . .	6
3.2.3	Instructions behaviors . . . . .	10
3.2.4	Exceptions . . . . .	24
3.2.5	Memory interface . . . . .	25
3.2.6	Debugging . . . . .	25
3.3	Nonfunctional Requirements . . . . .	25
<b>4</b>	<b>Configuration</b>	<b>26</b>
4.1	Constants . . . . .	26
4.2	Instanciation parameters . . . . .	26
<b>5</b>	<b>Functional Partitioning</b>	<b>27</b>
<b>6</b>	<b>Top Module</b>	<b>28</b>
<b>7</b>	<b>External Memory Module</b>	<b>29</b>
<b>8</b>	<b>Instruction Fetch Module</b>	<b>30</b>
8.1	Jump logic . . . . .	31
8.2	PC register . . . . .	33
8.3	Wishbone master . . . . .	34
8.4	Output logic . . . . .	35
<b>9</b>	<b>Decode Module</b>	<b>36</b>
<b>10</b>	<b>Register Module</b>	<b>37</b>
<b>11</b>	<b>Execute Module</b>	<b>38</b>

<b>12 Write-Back Module</b>	<b>39</b>
<b>13 Debug</b>	<b>40</b>

DRAFT

# 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 referring 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 targeting *Educational Computer Architecture Platform 5* (ECAP5). It will provide the main means of software execution in ECAP5.

## 1.4 Definitions and Acronyms

hardware-configurable

software-configurable

The bit indexing shall be described somewhere.

Byte size as well.

## 1.5 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
September 7, 2002	B.3	WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Core

## 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.

<b>ID</b>	U_INSTRUCTION_SET_01
<b>Description</b>	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.

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

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

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

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

The polarity of the reset signal mentioned in U\_RESET\_01 is not specified by the user.

<b>ID</b>	U_BOOT_ADDRESS_01
<b>Description</b>	The address at which ECAP5-DPROC jumps after the reset signal is deasserted shall be hardware-configurable.

The address mentioned in U\_BOOT\_ADDRESS\_01 can be either configured through hardware signals or can be selected at compile time.

<b>ID</b>	U_HARDWARE_INTERRUPT_01
<b>Description</b>	ECAP5-DPROC shall provide an signal which shall interrupt ECAP5-DPROC's execution flow while asserted.

<b>ID</b>	U_HARDWARE_INTERRUPT_02
<b>Description</b>	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.

<b>ID</b>	U_DEBUG_01
<b>Description</b>	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

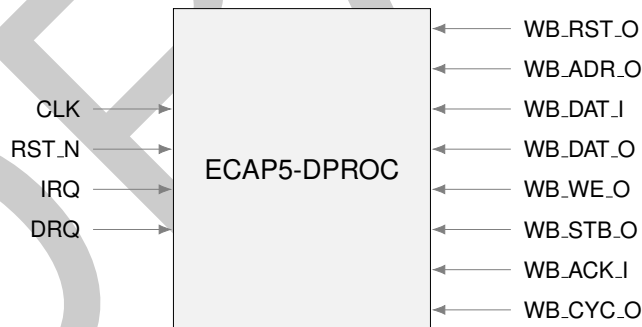


Figure 1: Schematic view of the external interface of ECAP5-DPROC

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

Table 1: ECAP5-DPROC control signals

NAME	TYPE	WIDTH	DESCRIPTION
WB_RST_O	O	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 2: ECAP5-DPROC memory interface signals

<b>ID</b>	I.CLK_01
<b>Description</b>	All inputs and outputs of ECAP5-DPROC shall belong to CLK's clock domain.
<b>Refers to</b>	

<b>ID</b>	I.RESET_01
<b>Description</b>	The RST_N signal shall hold ECAP5-DPROC in a reset state while asserted.
<b>Refers to</b>	U.RESET_01

<b>ID</b>	I.RESET_02
<b>Description</b>	RST_N polarity shall be active low.
<b>Refers to</b>	

<b>ID</b>	I.IRQ_01
<b>Description</b>	ECAP5-DPROC shall jump to a software-configurable address when input IRQ is asserted.
<b>Refers to</b>	U.HARDWARE_INTERRUPT_01, U.HARDWARE_INTERRUPT_02

<b>ID</b>	I.DIRQ_01
<b>Description</b>	TBD
<b>Refers to</b>	

<b>ID</b>	I.MEMORY_INTERFACE_01
<b>Description</b>	Signals from table 2 shall be compliant with the Wishbone specification.
<b>Refers to</b>	U.MEMORY_INTERFACE_02

Behavioral specification for symbols in table 2 is outlined in the functional requirements

section, subsection 3.2.5.

## 3.2 Functional Requirements

### 3.2.1 Register file

<b>ID</b>	F_REGISTERS_01
<b>Description</b>	ECAP5-DPROC shall implement 31 user-accessible general purpose registers ranging from x0 to x31.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_REGISTERS_02
<b>Description</b>	Register x0 shall be hardwired to the constant zero.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_REGISTERS_03
<b>Description</b>	ECAP5-DPROC shall implement a pc user-accessible register storing the address of the current instruction.
<b>Refers to</b>	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 inferred from the opcode as there can only be one encoding per opcode.

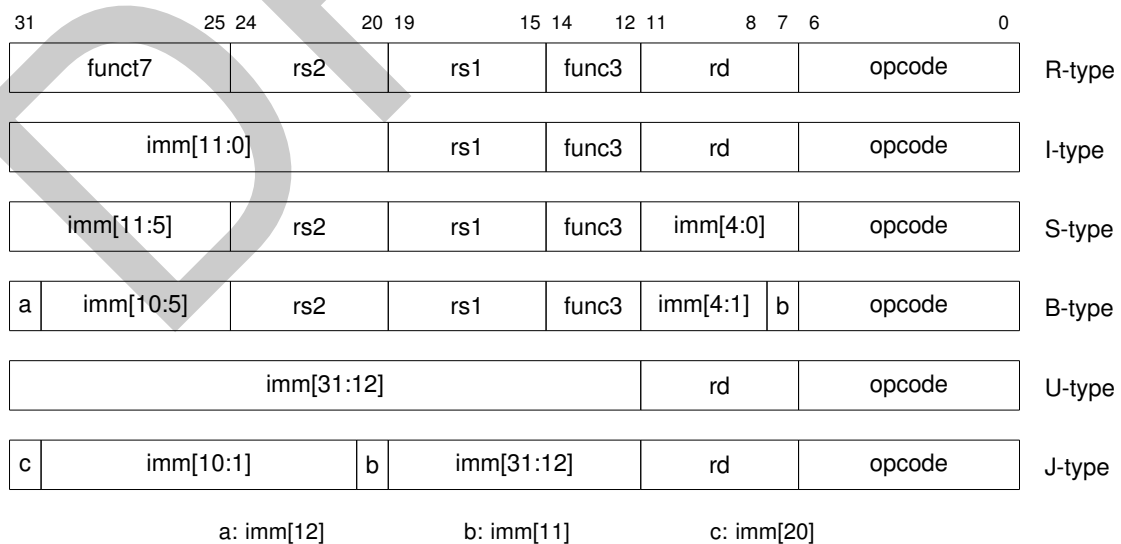


Figure 2: Instruction encodings of the RV32I instruction set



## Immediate encoding

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

<b>ID</b>	F_INSTR_IMMEDIATE_01
<b>Description</b>	Immediate values shall be sign-extended.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_INSTR_IMMEDIATE_02
<b>Description</b>	The value of an instruction immediate shall be the concatenation of immediate fragments from the instruction encoding.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_INSTR_IMMEDIATE_03
<b>Description</b>	Missing immediate fragments shall be replaced by zeros.
<b>Refers to</b>	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 inputs

<b>ID</b>	F_INSTR_FIRST_INPUT_01
<b>Description</b>	Instructions encoded using the R-type, I-type, S-type and B-type shall take as their first input the value stored in the register designated by the $rs1$ parameter.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_INSTR_FIRST_INPUT_02
<b>Description</b>	Instructions encoded using the U-type and J-type shall take as their first input the immediate value encoded in the instruction.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_INSTR_SECOND_INPUT_01
<b>Description</b>	Instructions encoded using the R-type, S-type and B-type shall take as their second input the value stored in the register designated by the $rs2$ parameter.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_INSTR.SECOND.INPUT_02
<b>Description</b>	Instructions encoded using the I-type shall take as its second input the immediate value encoded in the instruction.
<b>Refers to</b>	U_INSTRUCTION.SET_01

<b>ID</b>	F_INSTR.THIRD.INPUT_01
<b>Description</b>	Instructions encoded using the S-type and B-type shall take as their third input the immediate value encoded in the instruction.
<b>Refers to</b>	U_INSTRUCTION.SET_01

## Instruction outputs

<b>ID</b>	F_INSTR.OUTPUT_01
<b>Description</b>	Instructions encoded using the R-type, I-type, U-type and J-type shall store their result in the register designated by the <code>rd</code> parameter.
<b>Refers to</b>	U_INSTRUCTION.SET_01

<b>ID</b>	F_INSTR.OUTPUT_02
<b>Description</b>	Instructions encoded using the S-type and B-type do not produce any result.
<b>Refers to</b>	U_INSTRUCTION.SET_01

## Instruction variants

<b>ID</b>	F_INSTR.VARIANT_01
<b>Description</b>	Instructions encoded using the R-type, I-type, S-type and B-type shall use the <code>func3</code> parameter as a behavior variant selector.
<b>Refers to</b>	U_INSTRUCTION.SET_01

<b>ID</b>	F_INSTR.VARIANT_02
<b>Description</b>	Instructions encoded using the R-type shall use the <code>func7</code> parameter as a secondary behavior variant selector.
<b>Refers to</b>	U_INSTRUCTION.SET_01

## Opcodes

Table 3 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]	000	001	010	011	100	101	110	111
opcode[6:5]								
00	LOAD	<i>noimp</i>	<i>noimp</i>	MISC-MEM	OP-IMM	AUIPC	<i>noimp</i>	<i>noimp</i>
01	STORE	<i>noimp</i>	<i>noimp</i>	<i>noimp</i>	OP	LUI	<i>noimp</i>	<i>noimp</i>
10	<i>noimp</i>	<i>noimp</i>	<i>noimp</i>	<i>noimp</i>	<i>noimp</i>	<i>noimp</i>	<i>noimp</i>	<i>noimp</i>
11	BRANCH	JALR	<i>noimp</i>	JAL	SYSTEM	<i>noimp</i>	<i>noimp</i>	<i>noimp</i>

Table 3: Opcode values for the RV32I instruction set.

<b>ID</b>	F_OPCODE_ENCODING_01
<b>Description</b>	Instructions which use the LUI opcode shall be decoded as an U-type instruction.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_OPCODE_ENCODING_02
<b>Description</b>	Instructions which use the AUIPC opcode shall be decoded as an U-type instruction.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_OPCODE_ENCODING_03
<b>Description</b>	Instructions which use the JAL opcode shall be decoded as a J-type instruction.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_OPCODE_ENCODING_04
<b>Description</b>	Instructions which use the JALR opcode shall be decoded as an I-type instruction.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_OPCODE_ENCODING_05
<b>Description</b>	Instructions which use the BRANCH opcode shall be decoded as a B-type instruction.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_OPCODE_ENCODING_06
<b>Description</b>	Instructions which use the LOAD opcode shall be decoded as an I-type instruction.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_OPCODE_ENCODING_07
<b>Description</b>	Instructions which use the STORE opcode shall be decoded as a S-type instruction.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_OPCODE_ENCODING_08
<b>Description</b>	Instructions which use the OP-IMM opcode shall be decoded as an I-type instruction.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_OPCODE_ENCODING_09
<b>Description</b>	Instructions which use the OP opcode shall be decoded as a R-type instruction.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_OPCODE_ENCODING_10
<b>Description</b>	Instructions which use the MISC-MEM opcode shall be decoded as an I-type instruction.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_OPCODE_ENCODING_11
<b>Description</b>	Instructions which use the SYSTEM opcode shall be decoded as an I-type instruction.
<b>Refers to</b>	U_INSTRUCTION_SET_01

### 3.2.3 Instructions behaviors

#### LUI

<b>ID</b>	F_LUI_01
<b>Description</b>	The LUI behavior shall be applied when the opcode is LUI.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_ADDI_02
<b>Description</b>	The output of LUI shall be the value of its first input.
<b>Rationale</b>	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.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## AUIPC

<b>ID</b>	F_AUIPC_01
<b>Description</b>	The AUIPC behavior shall be applied when the opcode is AUIPC.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_AUIPC_02
<b>Description</b>	The output of AUIPC shall be the sum of its first input and the address of the AUIPC instruction.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## JAL

<b>ID</b>	F_JAL_01
<b>Description</b>	The JAL behavior shall be applied when the opcode is JAL.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_JAL_02
<b>Description</b>	The <code>pc</code> register shall be updated with the sum of the address of the JAL instruction with the first instruction input.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_JAL_03
<b>Description</b>	The output of JAL shall be the address of the JAL instruction incremented by 4.
<b>Rationale</b>	The JAL instruction shall output the address to the following instruction for it to be used as a <i>return address</i> in the case of a function call.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## JALR

<b>ID</b>	F_JALR_01
<b>Description</b>	The JALR behavior shall be applied when the opcode is JALR and func3 is 0x0.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_JALR_02
<b>Description</b>	The <code>pc</code> register shall be updated with the sum of the first and second inputs of the JALR instruction.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_JALR_03
<b>Description</b>	The output of JALR shall be the address of the JALR instruction incremented by 4.
<b>Rationale</b>	The JALR instruction shall output the address to the following instruction for it to be used as a <i>return address</i> in the case of a function call.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## BEQ

<b>ID</b>	F_BEQ_01
<b>Description</b>	The BEQ behavior shall be applied when the opcode is BRANCH and func3 is 0x0.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_BEQ_02
<b>Description</b>	When the first and second instruction inputs are equal, the <code>pc</code> register shall be updated with the signed sum of the address of the BEQ instruction with the third input.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## BNE

<b>ID</b>	F_BNE_01
<b>Description</b>	The BNE behavior shall be applied when the opcode is BRANCH and func3 is 0x1.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_BNE_02
<b>Description</b>	When the first and second inputs are not equal, the pc register shall be updated with the signed sum of the address of the BNE instruction with the third input.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## BLT

<b>ID</b>	F_BLT_01
<b>Description</b>	The BLT behavior shall be applied when the opcode is BRANCH and func3 is 0x4.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_BLT_02
<b>Description</b>	When the first input is lower than the second input using a signed comparison, the pc register shall be updated with the signed sum of the address of the BLT instruction with the third input.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## BGE

<b>ID</b>	F_BGE_01
<b>Description</b>	The BGE behavior shall be applied when the opcode is BRANCH and func3 is 0x5.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_BGE_02
<b>Description</b>	When the first input is greater or equal to the second input using a signed comparison, the pc register shall be updated with the signed sum of the address of the BGE instruction with the third input.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## BLTU

<b>ID</b>	F_BLTU_01
<b>Description</b>	The BLTU behavior shall be applied when the opcode is BRANCH and func3 is 0x6.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_BLTU_02
<b>Description</b>	When the first input is lower than the second input using an unsigned comparison, the pc register shall be updated with the signed sum of the address of the BLTU instruction with the third input.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## BGEU

<b>ID</b>	F_BGEU_01
<b>Description</b>	The BGEU behavior shall be applied when the opcode is BRANCH and func3 is 0x7.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_BGEU_02
<b>Description</b>	When the first input is greater or equal to the second input using an unsigned comparison, the pc register shall be updated with the signed sum of the address of the BGEU instruction with the third input.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## LB

<b>ID</b>	F_LB_01
<b>Description</b>	The LB behavior shall be applied when the opcode is LOAD and func3 is 0x0.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_LB_02
<b>Description</b>	The output of LB shall be the 8-bit value stored in memory at the address determined by the signed sum of its first and second inputs.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_LB_03
<b>Description</b>	The remaining bits of the loaded value shall be filled with the value of its 7 <sup>th</sup> bit.
<b>Refers to</b>	U_INSTRUCTION_SET_01



## LH

<b>ID</b>	F_LH.01
<b>Description</b>	The LH behavior shall be applied when the opcode is LOAD and func3 is 0x1.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_LH.02
<b>Description</b>	The output of LH shall be the 16-bit value stored in memory at the address determined by the signed sum of its first and second inputs.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_LH.03
<b>Description</b>	The remaining bits of the loaded value shall be filled with the value of its 15 <sup>th</sup> bit.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## LW

<b>ID</b>	F_LW.01
<b>Description</b>	The LW behavior shall be applied when the opcode is LOAD and func3 is 0x2.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_LW.02
<b>Description</b>	The output of LW shall be the 32-bit value stored in memory at the address determined by the signed sum of its first and second inputs.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## LBU

<b>ID</b>	F_LBU.01
<b>Description</b>	The LBU behavior shall be applied when the opcode is LOAD and func3 is 0x4.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_LBU.02
<b>Description</b>	The output of LBU shall be the 8-bit value stored in memory at the address determined by the signed sum of its first and second inputs.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_LBU_03
<b>Description</b>	The remaining bits of the loaded value shall be filled with zeros.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## LHU

<b>ID</b>	F_LHU_01
<b>Description</b>	The LHU behavior shall be applied when the opcode is LOAD and func3 is 0x5.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_LHU_02
<b>Description</b>	The output of LHU shall be the 16-bit value stored in memory at the address determined by the signed sum of its first and second inputs.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_LHU_04
<b>Description</b>	The remaining bits of the loaded value shall be filled with zeros.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## SB

<b>ID</b>	F_SB_01
<b>Description</b>	The SB behavior shall be applied when the opcode is STORE and func3 is 0x0.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SB_02
<b>Description</b>	The lowest byte of the second input of SB shall be stored in memory at the address determined by the signed sum of its first and third inputs.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## SH

<b>ID</b>	F_SH_01
<b>Description</b>	The SH behavior shall be applied when the opcode is STORE and func3 is 0x1.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SH_02
<b>Description</b>	The two lowest bytes of the second input of SB shall be stored in memory at the address determined by the signed sum of its first and third inputs.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## SW

<b>ID</b>	F_SW_01
<b>Description</b>	The SW behavior shall be applied when the opcode is STORE and func3 is 0x2.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SH_02
<b>Description</b>	The value of the second input of SB shall be stored in memory at the address determined by the signed sum of its first and third inputs.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## ADDI

<b>ID</b>	F_ADDI_01
<b>Description</b>	The ADDI behavior shall be applied when the opcode is OP-IMM and when func3 is 0x0.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_ADDI_02
<b>Description</b>	The output of ADDI shall be the signed integer sum of its two inputs.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_ADDI_03
<b>Description</b>	The output of ADDI shall be truncated to 32-bits.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## SLTI

<b>ID</b>	F_SLTI_01
<b>Description</b>	The SLTI behavior shall be applied when the opcode is OP-IMM and when func3 is 0x2.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SLTI_02
<b>Description</b>	The output of SLTI shall be 1 when the signed value of its first input is lower than the signed value of its second input. It shall be 0 otherwise.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## SLTIU

<b>ID</b>	F_SLTIU_01
<b>Description</b>	The SLTIU behavior shall be applied when the opcode is OP-IMM and when func3 is 0x3.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SLTIU_02
<b>Description</b>	The output of SLTI shall be 1 when the unsigned value of its first input is lower than the unsigned value of its second input. It shall be 0 otherwise.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## XORI

<b>ID</b>	F_XORI_01
<b>Description</b>	The XORI behavior shall be applied when the opcode is OP-IMM and when func3 is 0x4.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_XORI_02
<b>Description</b>	The output of XORI shall be the result of a bitwise xor between its two inputs.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## ORI

<b>ID</b>	F_ORI_01
<b>Description</b>	The ORI behavior shall be applied when the opcode is OP-IMM and when func3 is 0x6.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_ORI_02
<b>Description</b>	The output of ORI shall be the result of a bitwise or between its two inputs.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## ANDI

<b>ID</b>	F_ANDI.01
<b>Description</b>	The ANDI behavior shall be applied when the opcode is OP-IMM and when func3 is 0x7.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_ANDI.02
<b>Description</b>	The output of ANDI shall be the result of a bitwise and between its two inputs.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## SLLI

<b>ID</b>	F_SLLI.01
<b>Description</b>	The SLLI behavior shall be applied when the opcode is OP-IMM and func3 is 0x1.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SLLI.02
<b>Description</b>	The output of SLLI shall be its first input shifted left by the amount specified by the first 5 bits of its second input.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SLLI.03
<b>Description</b>	Zeros shall be inserted in the lower bits when shifting.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## SRLI

<b>ID</b>	F_SRLI.01
<b>Description</b>	The SRLI behavior shall be applied when the opcode is OP-IMM, func3 is 0x5 and the 30 <sup>th</sup> bit of its second input is 0.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SRLI.02
<b>Description</b>	The output of SRLI shall be its first input shifted right by the amount specified by the first 5 bits of its second input.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SRLI_03
<b>Description</b>	Zeros shall be inserted in the upper bits when shifting.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## SRAI

<b>ID</b>	F_SRAI_01
<b>Description</b>	The SRAI behavior shall be applied when the opcode is OP-IMM, func3 is 0x5 and the 30 <sup>th</sup> bit of its second input is 1.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SRAI_02
<b>Description</b>	The output of SRAI shall be its first input shifted right by the amount specified by the first 5 bits of its second input.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SRAI_03
<b>Description</b>	The most significant bit of the first input shall be inserted in the upper bits when shifting.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## ADD

<b>ID</b>	F_ADD_01
<b>Description</b>	The ADD behavior shall be applied when the opcode is OP, func3 is 0x0 and the 30 <sup>th</sup> bit of its second input is 0.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_ADD_02
<b>Description</b>	The output of ADD shall be the signed integer sum of its two inputs.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_ADD_03
<b>Description</b>	The output of ADD shall be truncated to 32-bits.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## SUB

<b>ID</b>	F_SUB_01
<b>Description</b>	The SUB behavior shall be applied when the opcode is OP, func3 is 0x0 and the 30 <sup>th</sup> bit of its second input is 1.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SUB_02
<b>Description</b>	The output of SUB shall be the signed integer difference of its first input minus its second input.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SUB_03
<b>Description</b>	The output of SUB shall be truncated to 32-bits.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## SLL

<b>ID</b>	F_SLL_01
<b>Description</b>	The SLL behavior shall be applied when the opcode is OP and func3 is 0x1.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SLL_02
<b>Description</b>	The output of SLL shall be its first input shifted left by the amount specified by the first 5 bits of its second input.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SLL_03
<b>Description</b>	Zeros shall be inserted in the lower bits when shifting.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## SLT

<b>ID</b>	F_SLT_01
<b>Description</b>	The SLT behavior shall be applied when the opcode is OP and func3 is 0x2.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SLT_02
<b>Description</b>	The output of SLT shall be 1 when the signed value of its first input is lower than the signed value of its second input. It shall be 0 otherwise.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## SLTU

<b>ID</b>	F_SLTU_01
<b>Description</b>	The SLTU behavior shall be applied when the opcode is OP and func3 is 0x3.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SLTU_02
<b>Description</b>	The output of SLTU shall be 1 when the unsigned value of its first input is lower than the unsigned value of its second input. It shall be 0 otherwise.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## XOR

<b>ID</b>	F_XOR_01
<b>Description</b>	The XOR behavior shall be applied when the opcode is OP and func3 is 0x4.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_XOR_02
<b>Description</b>	The output of XOR shall be the result of a bitwise xor between its two inputs.
<b>Refers to</b>	U_INSTRUCTION_SET_01



## SRL

<b>ID</b>	F_SRL_01
<b>Description</b>	The SRL behavior shall be applied when the opcode is OP, func3 is 0x5 and the 30 <sup>th</sup> bit of its second input is 0.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SRL_02
<b>Description</b>	The output of SRL shall be its first input shifted right by the amount specified by the first 5 bits of its second input.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SRL_03
<b>Description</b>	Zeros shall be inserted in the upper bits when shifting.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## SRA

<b>ID</b>	F_SRA_01
<b>Description</b>	The SRA behavior shall be applied when the opcode is OP, func3 is 0x5 and the 30 <sup>th</sup> bit of its second input is 1.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SRA_02
<b>Description</b>	The output of SRA shall be its first input shifted right by the amount specified by the first 5 bits of its second input.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_SRA_03
<b>Description</b>	The most significant bit of the first input shall be inserted in the upper bits when shifting.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## OR

<b>ID</b>	F_OR_01
<b>Description</b>	The OR behavior shall be applied when the opcode is OP and func3 is 0x6.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_OR_02
<b>Description</b>	The output of OR shall be the result of a bitwise or between its two inputs.
<b>Refers to</b>	U_INSTRUCTION_SET_01

## AND

<b>ID</b>	F_AND_01
<b>Description</b>	The AND behavior shall be applied when the opcode is OP and func3 is 0x7.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_AND_02
<b>Description</b>	The output of AND shall be the result of a bitwise and between its two inputs.
<b>Refers to</b>	U_INSTRUCTION_SET_01

**FENCE** TBD

**ECALL** TBD

**EBREAK** TBD

### 3.2.4 Exceptions

<b>ID</b>	F_INSTR_ADDR_MISALIGNED_01
<b>Description</b>	An Instruction Address Misaligned exception shall be raised when the target address of a taken branch or an unconditional jump is not four-byte aligned.
<b>Refers to</b>	U_INSTRUCTION_SET_01

<b>ID</b>	F_MISALIGNED_MEMORY_ACCESS_01
<b>Description</b>	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.
<b>Refers to</b>	U_INSTRUCTION_SET_01

### 3.2.5 Memory interface

<b>ID</b>	F_ENDIANNESS_01
<b>Description</b>	Memory accesses shall use the little-endian format.
<b>Refers to</b>	

Outline requirements to be compliant with the Wishbone specification.

### 3.2.6 Debugging

## 3.3 Nonfunctional Requirements

<b>ID</b>	N_FORMAL_PROOF_01
<b>Description</b>	Each part of ECAP5-DPROC shall be formally proven when possible, otherwise thoroughly tested
<b>Refers to</b>	

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

## 4 Configuration

### 4.1 Constants

### 4.2 Instanciation parameters

DRAFT

## 5 Functional Partitioning

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 instruction behaviors. This includes performing integer operations as well as accessing memory.
- The write-back stage which handles storing instructions outputs to internal registers.

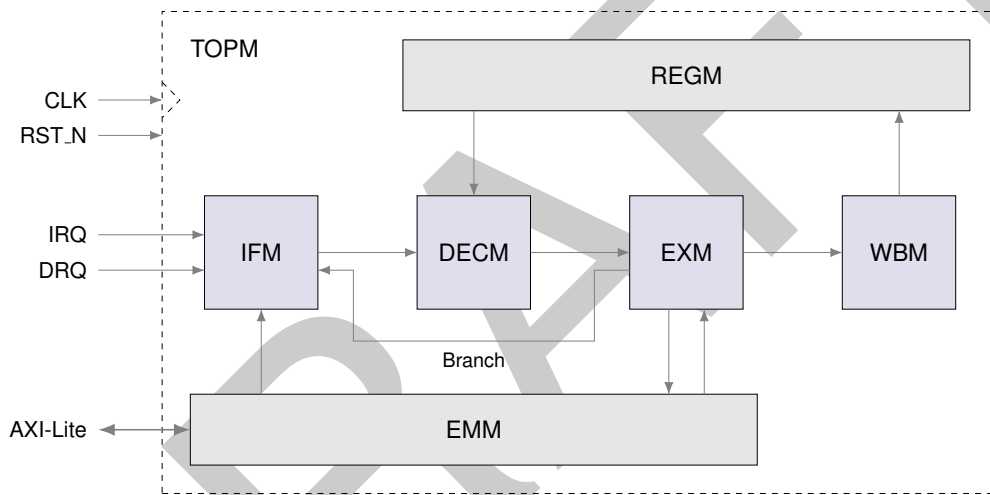


Figure 3: Schematic view of the architecture of ECAP5-DPROC

The design is split into the following functional modules :

- The **Top Module** (TOPM) which integrates all other modules.
- The **External Memory Module** (EMM), in charge of accessing memory and peripherals.
- The **Instruction Fetch Module** (IFM), in charge of implementing the instruction fetch stage.
- The **Decode Module** (DECM), in charge of implementing the decode stage.
- The **Register Module** (REGM), implementing the internal registers.
- The **Execute Module** (EXM), in charge of implementing the execute stage.
- The **Write-Back Module** (WBM), in charge of implementing the write-back stage.

## 6 Top Module

Handshaking and bubbling

DRAFT

## 7 External Memory Module

DRAFT

## 8 Instruction Fetch Module

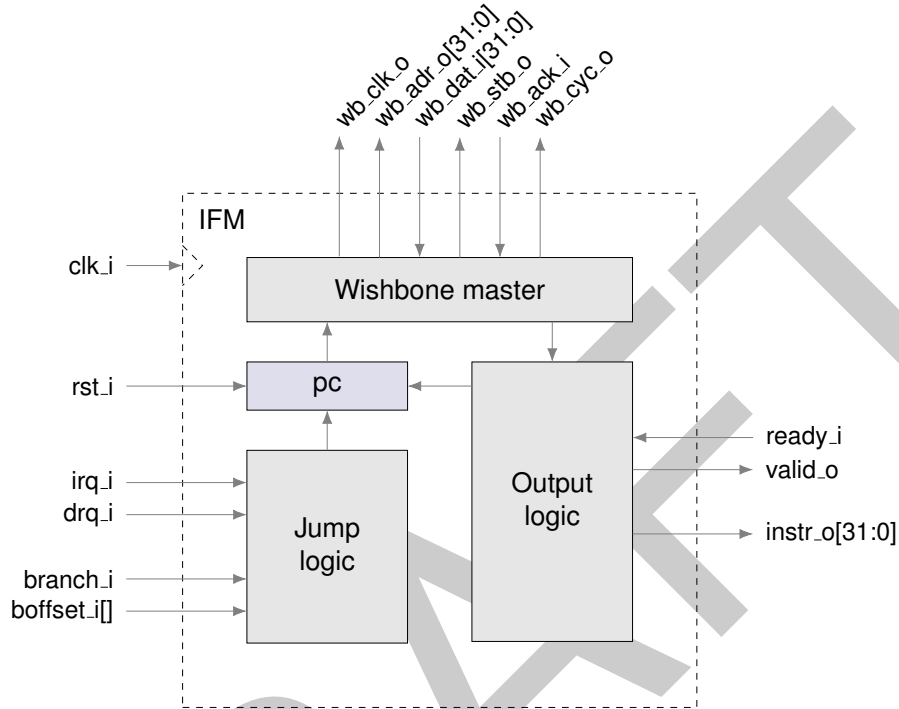


Figure 4: Schematic view of the Instruction Fetch Module

The instruction fetch module handles fetching from memory the instructions to be executing. The signals are described in table 4.

Table 4: Instruction Fetch Module interface signals

NAME	TYPE	WIDTH	DESCRIPTION
clk_i	I	1	Clock input.
rst_i	I	1	Reset input.
<b>JUMP LOGIC</b>			
irq_i	I	1	External interrupt request.
drq_i	I	1	External debug request.
branch_i	I	1	Branch request.
boffset_i	I	TBC	Branch offset. TBC
<b>WISHBONE MASTER</b>			
wb_clk_o	O	1	Wishbone clock output. This is hardwired to clk_i.
wb_adr_o	O	32	Wishbone read address.



wb_dat.i	I	32	Wishbone read data.
wb_stb.o	O	1	Strobe output indicates a valid data transfer cycle.
wb_ack.i	I	1	Acknowledge. Indicates a normal termination of a bus cycle.
wb_cyc.o	O	1	Cycle. Indicates that a valid bus cycle is in progress.
<b>OUTPUT LOGIC</b>			
ready.i	I	1	Asserted when the output is ready to be received.
valid.o	O	1	Asserted when the output is ready to be sent.
instr.o	O	32	Instruction to be executed.

pc stores the value of the next instruction to be loaded from memory. It is connected to the wishbone master which performs the memory read. The read data is transferred to the output logic, in charge of handling the pipeline's handshaking protocol. The value of pc can be either incremented by the output logic, reset by rst.i or loaded with a specific value through the jump logic.

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

## 8.1 Jump logic

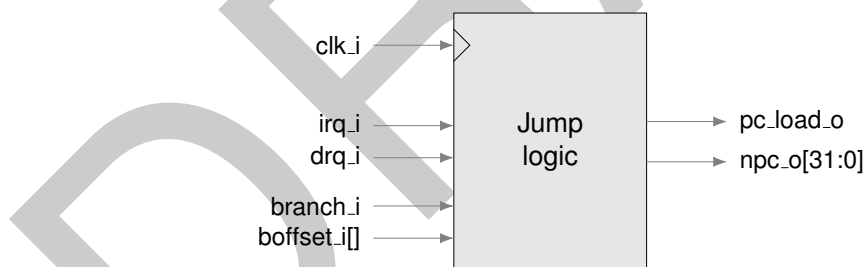


Figure 5: Schematic view of the interface of the Jump Logic

The jump logic loads a value into the pc register based on inputs. Its interface is described in table 5.

Table 5: Jump logic interface signals

NAME	TYPE	WIDTH	DESCRIPTION
clk.i	I	1	Clock input.

irq_i	I	1	External interrupt request.
drq_i	I	1	External debug request.
branch_i	I	1	Branch request.
boffset_i	I	TBC	Branch offset. TBC
pc_load_o	O	1	Asserted when the pc register shall be updated.
npc_o	O	32	Value used to update the pc register.

Figures 6 and 7 describe the behavior of the jump logic. The values to be loaded in memory are hardcoded and set at compile time.

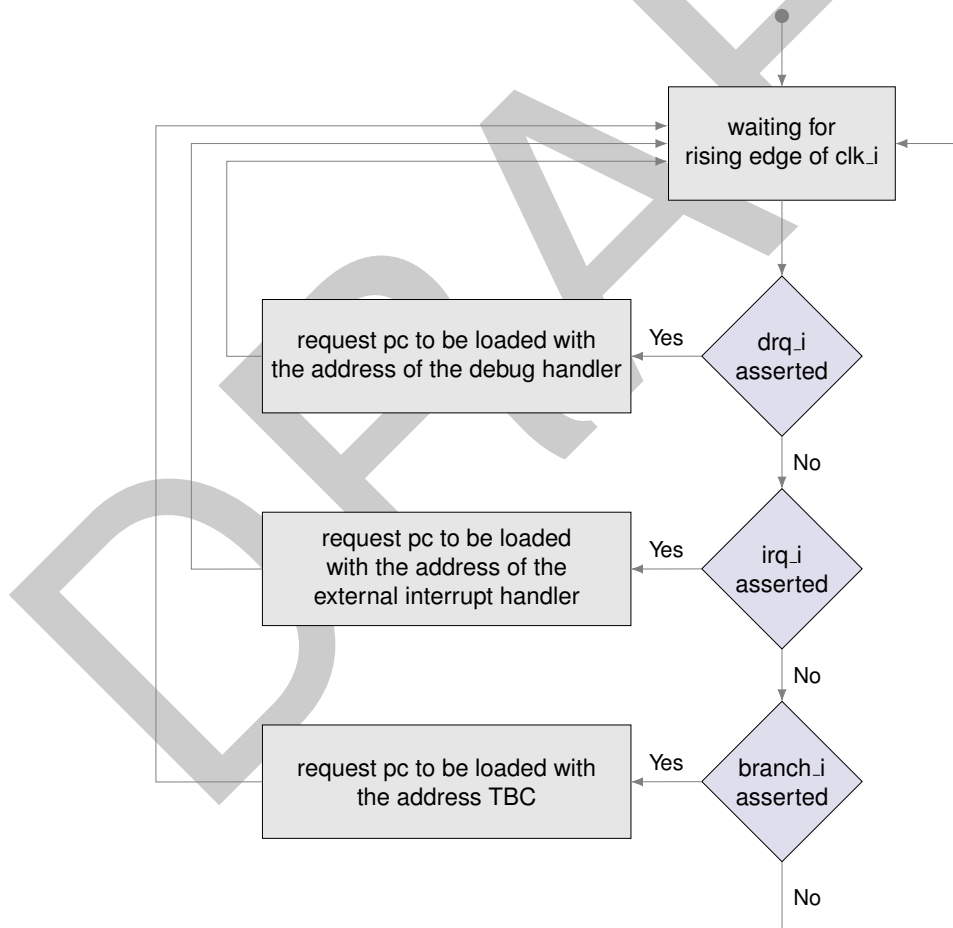


Figure 6: Activity diagram of the jump logic behavior

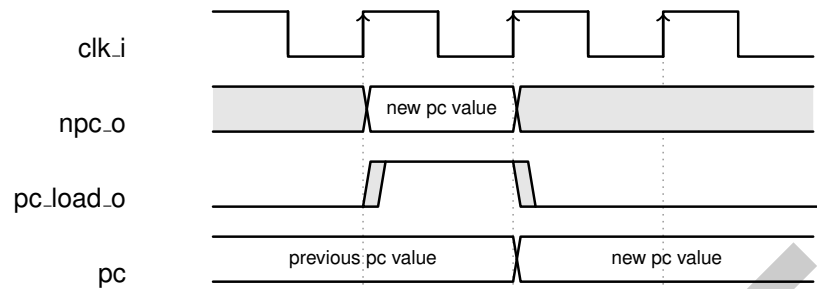


Figure 7: Timing diagram for the output port of the jump logic.

## 8.2 PC register

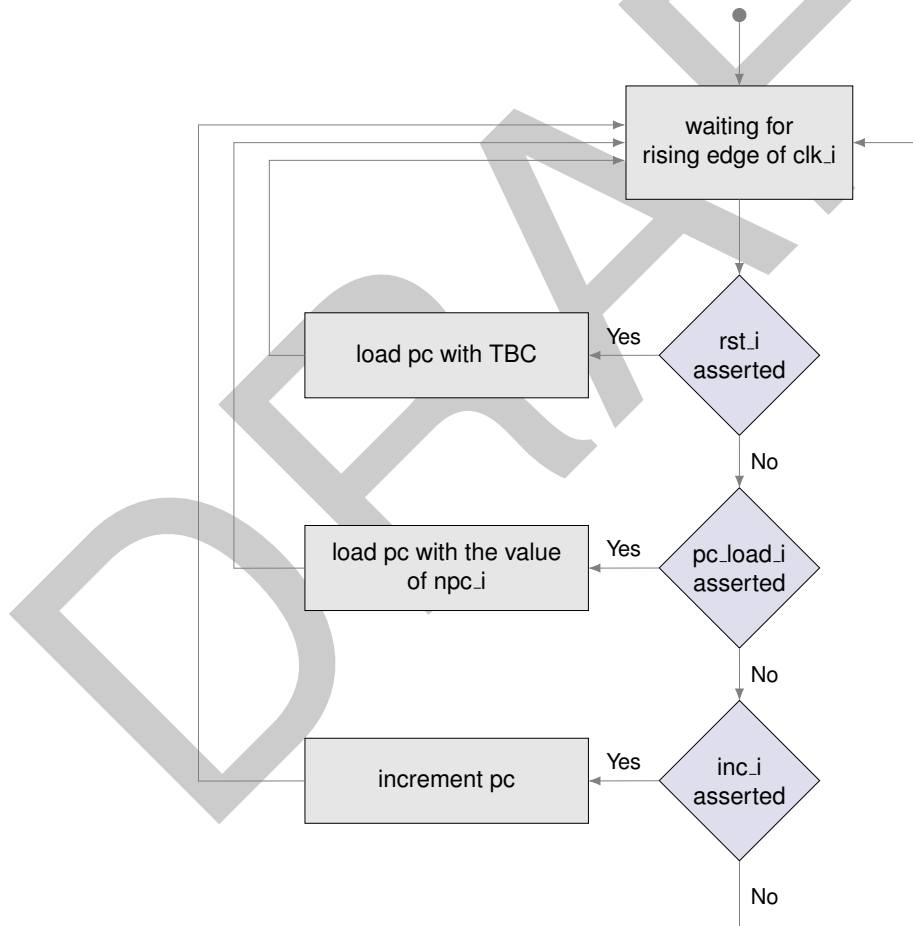


Figure 8: Activity diagram of the pc register

## 8.3 Wishbone master

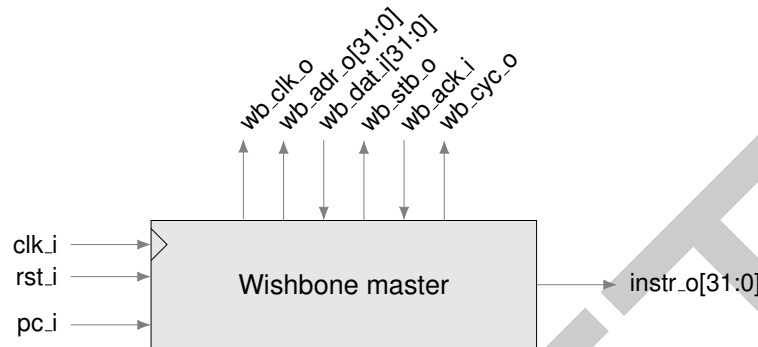


Figure 9: Schematic view of the interface of the Wishbone Master

The wishbone master fetches from memory the instruction to be executed. Its interface is described in table 6.

Table 6: Instruction Fetch Module interface signals

NAME	TYPE	WIDTH	DESCRIPTION
clk.i	I	1	Clock input.
rst.i	I	1	Reset input.
<b>WISHBONE MASTER</b>			
wb_clk_o	O	1	Wishbone clock output. This is hardwired to clk.i.
wb_adr_o	O	32	Wishbone read address.
wb_dat_i	I	32	Wishbone read data.
wb_stb_o	O	1	Strobe output indicates a valid data transfer cycle.
wb_ack_i	I	1	Acknowledge. Indicates a normal termination of a bus cycle.
wb_cyc_o	O	1	Cycle. Indicates that a valid bus cycle is in progress.
<b>OUTPUT</b>			
instr_o	O	32	Instruction to be executed.

## 8.4 Output logic

DRAFT

## 9 Decode Module

DRAFT

## 10 Register Module

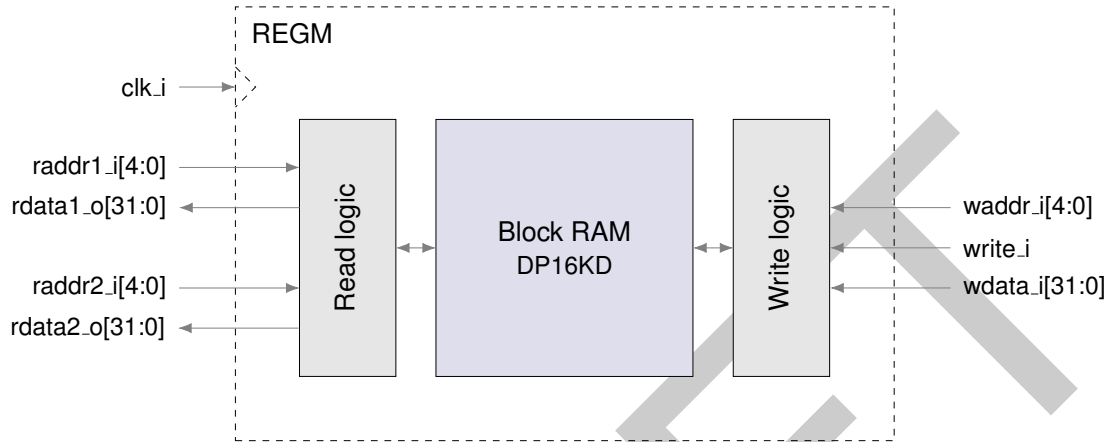


Figure 10: Schematic view of the Register Module

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 7.

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

Table 7: Register Module interface signals

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. 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.

## 11 Execute Module

DRAFT



## 12 Write-Back Module

DRAFT

## 13 Debug

DRAFT