Table of Contents

1 Home

Welcome to the OpenCEHardware RTL hardware module documentation template. This document provides a standardized structure for documenting hardware designs based on RTL. The template is designed to assist in organizing and presenting technical documentation for hardware modules, facilitating a clear and comprehensive description of the design.

1.1 Description

This documentation template is based on multiple sources and industry best practices. It includes detailed sections to cover all important aspects of an RTL hardware design, from the general description of the module to specific details of its HDL implementation.

Even though this template is designed for complex desings, its principles can be applied to smaller, simpler designs. It may also guide novice hardware architects toward potential upgrades for their designs.

Disclaimer

This template excludes any post-silicon or physical design implementation details, such as reliability floorplans, pin mapping, SRAM placement, chip area, power, physical debugging, etc. The focus of this template is on FPGA constraint RTL designs.

1.2 Navigation

The template is organized into the following sections, each explaining its purpose, general structure, tips, and examples.

Note

The References and Revisions sections serve a dual purpose, serving both as a example and keeping the actual revisions and references for the template.

- Revisions: Documentation on previous versions and changes made.
- Document Conventions: Definitions and abbreviations used in the document.
- Introduction: General description of the block and its features.
- Block Diagram: Visual representation of the block's microarchitecture.
- Configuration: Information about parameters, typedefs, and RTL interfaces.
- Protocols: Details of communication and operation protocols.
- Memory Map: Distribution of memory and resource allocation.
- Registers: Description of the registers used in the system.
- Clock Domains: Information about clocks and their management in the system.
- Reset Domains: Information about reset mechanisms and their domains.
- Interrupts: Management and handling of interrupts in the system.
- Arbitration: Arbitration mechanisms for access to shared resources.
- Debugging: Techniques and tools for system debugging.
- Synthesis: Summary and results of the design synthesis.

Microarchitecture:

- Sub-module 1: Details of the first sub-module of the microarchitecture.
- Sub-module 2: Details of the second sub-module of the microarchitecture.

1.3 Using the Template

This template is designed to be downloaded and adapted to the specific needs of each project. It is recommended to follow the provided structure and customize it according to the design requirements and project standards. Simpler designs may not require documentation for all sections. In such cases, it is recommended to indicate that the section is not needed rather than leaving it blank or excluding it from the document.

This template was created using MkDocs, a widely used tool for code documentation (e.g., ReadTheDocs). MkDocs uses Markdown, which is excellent for creating interactive, easy-to-understand documentation. It also integrates seamlessly with GitHub and can be published as a web page (e.g., using GitHub Pages). Additionally, it can be rendered as a PDF, as provided in this template. MkDocs supports themes that offer further customizability and aesthetics. This template uses Material for MkDocs, which offers a comprehensive guide for styling that is well worth exploring.

1.4 Acknowledgements

This template has been developed from various sources and industry standards to ensure comprehensive and useful documentation for RTL hardware design. Please check the References section for more information.

I. Block Name

2 Revisions

Continually maintain the following table of major and minor revisions of the specification, even if using a version control system like Git. Use a common version naming convention that can be shared with other related blocks. Always write dates in ISO standard form, i.e., YYYY-MM-DD [1].

In this case, the example table corresponds to the actual revisions of this document.

Date (YYYY-MM-DD)	**Version**	**Description of Changes**	**Author**
2024-08-06	`1.0`	Document creation.	Alejandro Chavarría

3 Document Conventions

This section lists the conventions in terms of definitions and abbreviations that will be used throughout the document. It allows the reader to become familiar with the authors' terminology and establishes a contract with the reader. External links can also be added for further insights.

3.1 Glossary

List of important words:

- **Word**: Meaning
- **Word**: Meaning
- **Word**: Meaning

3.2 Abbreviations

List of important abbreviations:

- **ABC**: Meaning
- **ABC**: Meaning
- **ABC**: Meaning

4 Introduction

This section provides a description of the block. It briefly covers, at a high level, what the block does and presents a "bird's-eye" black-box view of the top-level module. It discusses the goals and non-goals of the block, how it is intended to integrate into a larger system, lists standard protocols, highlights important performance requirements, and touches on debugging features. It outlines the design methodology (coding language, internal and third-party libraries and IPs), and anything else a verification engineer should know before writing the first draft of the test plan [1]. Most of these concepts can be expressed as a list of features, as seen in an I²C-Master Core Specification [2] example.

4.1 Features

In a list of sentences, this section expresses what the block is capable of and its most notable characteristics.

4.1.1 Integration

- **Functionality A**
- **Functionality B**
- **Functionality C**

4.1.2 Performance

- **Functionality A**
- **Functionality B**
- **Functionality C**

4.1.3 Design

- **Functionality A**
- **Functionality B**
- **Functionality C**

4.1.4 Debugging

- **Functionality A**
- **Functionality B**
- **Functionality C**

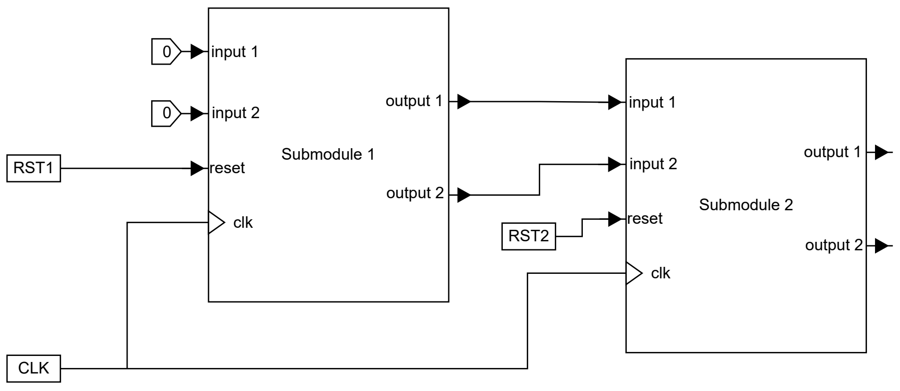
5 Top-Level Block Diagram

The top-level block diagram should indicate the block boundary and all major interfaces. Draw the top-level submodules and show how they connect internally, but avoid displaying excessive internal implementation detail. The structure of the top-level block diagram should correspond 1:1 with the contents of the top-level RTL module. Avoid "free-floating" logic at the top level (everything should be encapsulated in submodules) [1].

Tip

Markdown allows embedded HTML. With the help of diagramming software (like draw.io), you can hyperlink the diagram to different sections of the document or even external links.

5.1 Example



6 Configuration

This section includes tables of typedefs, design parameters, and interfaces. In general, it addresses any configurability encoded into the design via RTL. Include the main parameters of the block (both private and shared), the top-level module parameters, and preprocessor macros that set global constants. Do not include parameters or macros that are derived from others. Ensure to describe any constraints and assumptions about reasonable or default values. Only explain the types that are necessary to fully define the parameters and interfaces [1].

6.1 Parameters

Parameter Name	Type	Description	Default Value	Range/Possible Values
`PARAM1`	`int`	Brief description of the parameter.	`10`	`0` to `100`
`PARAM2`	`float`	Brief description of the parameter.	`0.5`	`0.0` to `1.0`
`PARAM3`	`bool`	Brief description of the parameter.	`true`	`true`, `false`

6.2 Typedefs

Typedef Name	Type	Description
`typedef_name1`	`struct`	Brief description of what the typedef defines.
`typedef_name2`	`enum`	Brief description of what the typedef defines.
`typedef_name3`	`union`	Brief description of what the typedef defines.

6.3 Interfaces

This section includes a table of top-level module interfaces. Group related ports as a single interface. Show the directions and types of the ports, describe the interface's purpose, and follow common naming conventions. Mention the use of any standard protocols. Avoid excessive abbreviations. The directions, types, and names of the ports should match the RTL ports 1:1. The port directions should be from the block's perspective (as in the RTL) [1].

6.3.1 Interface 1

Port Name	Direction	Type	Description
`port_name`	Input	`type`	Brief description of what this port does.
`port_name`	Output	`type`	Brief description of what this port does.
`port_name`	In/Out	`type`	Brief description of what this port does.

Protocol Use: Mention any standard protocol used by this interface here, if applicable.

6.3.2 Interface 2

Port Name	Direction	Type	Description
`port_name`	Input	`type`	Brief description of what this port does.
`port_name`	Output	`type`	Brief description of what this port does.
`port_name`	In/Out	`type`	Brief description of what this port does.

Protocol Use: Mention any standard protocol used by this interface here, if applicable.

7 Protocols

For all interfaces that use a standard industry protocol or an internal/proprietary protocol, list them here and link to the relevant specifications that govern those protocols. If any interfaces use custom protocols that are not defined elsewhere, define them in detail here, with one subsection per protocol. Make sure to define any protocols that involve more than one interface [1]. For custom protocols, feel free to add as much information as needed so they can be easily understood, used, and tested.

7.1 Standard Protocols

7.1.1 Protocol 1

- **Description**: Brief description of the protocol.
- **Specification**: Link to the protocol specification.

7.1.2 Protocol 2

- **Description**: Brief description of the protocol.
- **Specification**: Link to the protocol specification.

7.2 Custom Protocols

7.2.1 Custom Protocol 1

- **Description**: Detailed description of the custom protocol.
- **Involved Interfaces**: List of interfaces that use this protocol.
- **Data Format**: Definition of the data format exchanged.
- **Communication Sequence**: Details about the communication sequence between interfaces.

7.2.2 Custom Protocol 2

- **Description**: Detailed description of the custom protocol.
- **Involved Interfaces**: List of interfaces that use this protocol.
- **Data Format**: Definition of the data format exchanged.
- **Communication Sequence**: Details about the communication sequence between interfaces.

8 Memory Map

This section provides a clear and organized overview of how memory is distributed in the system. It includes a table that shows memory addresses, specific regions assigned to different system components (such as ROM, RAM, peripherals, CSRs, etc.), and any relevant details about the size, access permissions, and purpose of each region. Detailed memory resource allocation facilitates the design, development, use, and debugging of the system.

8.1 Memory Map Table

Memory Address	**Size**	**Region**	**Description**
0x00000000 - 0x0000FFFF	64 KB	ROM (Firmware)	Contains the boot code and firmware.
0x00010000 - 0x0001FFFF	64 KB	RAM (Data)	Stores variable data and stacks.
0x00020000 - 0x0002FFFF	64 KB	Peripheral A	Control and data registers of Peripheral A.
0x00030000 - 0x0003FFFF	64 KB	Peripheral B	Control and data registers of Peripheral B.
0x00040000 - 0x0004FFFF	64 KB	RAM (Stack)	Reserved space for the stack and temporary storage.

9 Registers

This section contains all the registers of the block, organized in tables. This includes user registers, CSRs, and other registers (e.g., debugging).

9.1 User Registers

9.1.1 User Registers Table

Register Name	Abbreviation	Address	Fields and Offsets	Access Permissions (HW/SW)	Description
USER_REG_1	UR1	0x0100	FIELD1 (0-7)	R/W	Description of User Register 1
USER_REG_2	UR2	0x0104	FIELD1 (0-15)	R	Description of User Register 2

9.2 CSRs (Configuration/Status Registers)

List major categories of configuration/status registers (CSRs). Define all CSRs here or link to an external document. Every CSR should define its name, address, fields and offsets, hardware and software access permissions, and a description of what each field does [1].

Note

Ideally, this subsection should link to CSR documentation that is auto-generated from a single-source-of-truth source code. An open-source industry tool used for this purpose is PeakRDL.

9.2.1 CSRs Table

CSR Name	Abbreviation	Address	Fields and Offsets	Access Permissions (HW/SW)	Description
CSR_NAME_1	CSR1	0x0000	FIELD1 (0-7), FIELD2 (8-15)	R/W	Description of CSR 1
CSR_NAME_2	CSR2	0x0004	FIELD1 (0-3), FIELD2 (4-7), FIELD3 (8-15)	R/W	Description of CSR 2

9.3 Other Registers

9.3.1 Other Registers Table

Register Name	Abbreviation	Address	Fields and Offsets	Access Permissions (HW/SW)	Description
DEBUG_REG_1	DR1	0x0200	FIELD1 (0-7), FIELD2 (8-15)	R/W	Description of Debug Register 1
DEBUG_REG_2	DR2	0x0204	FIELD1 (0-3), FIELD2 (4-7), FIELD3 (8-15)	R/W	Description of Debug Register 2

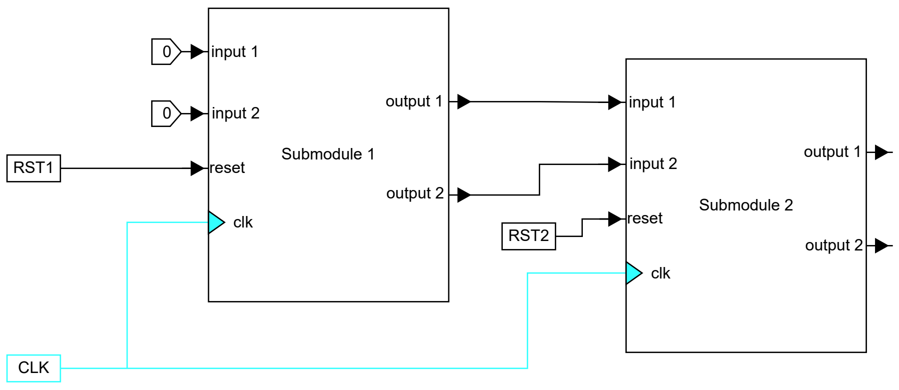
10 Clock Domains

This section provides a detailed overview of the clock domains used in the design. For each clock, indicate its nominal frequency and the supported dynamic range. Include the same top-level block diagram as before, but this time annotate it to show which submodules are in each clock domain. Clock domain crossings in the data path should be explicitly drawn and encapsulated within one or more submodules. For clocks used for "backbone" functions spanning many submodules (e.g., a CSR bus in its own clock domain), indicate this clearly and refer to another document or appropriate section for details [1].

10.1 Clock Domain Table

Clock Domain	Nominal Frequency	Supported Dynamic Range
Clock Domain 1	XX MHz	YY MHz - ZZ MHz
Clock Domain 2	AA MHz	BB MHz - CC MHz
Clock Domain 3	DD MHz	EE MHz - FF MHz

10.2 Annotated Block Diagram



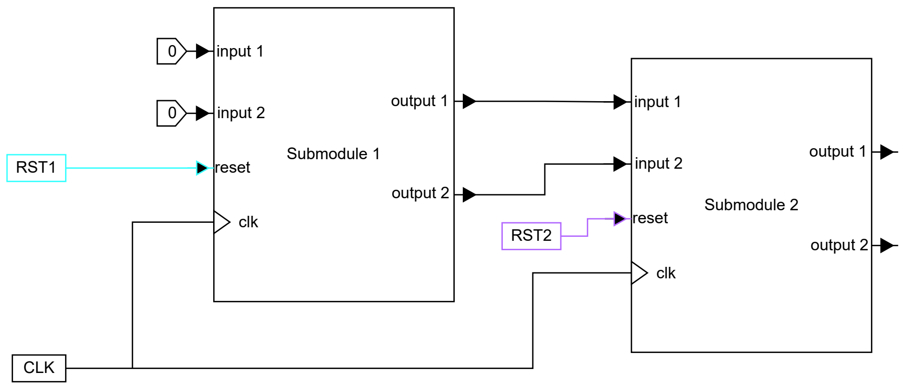
11 Reset Domains

This section provides an overview of the reset domains used in the design. For each reset, specify whether it is synchronous or asynchronous, its active level (high or low), and, if synchronous, the associated clock. Include the same top-level block diagram as before, but this time annotate it to show which submodules are in each reset domain. Reset domain crossings in the datapath should be explicitly shown and encapsulated within one or more submodules. For resets used for "backbone" functions that span multiple submodules (e.g., a CSR bus in its own reset domain), clearly indicate this and defer detailed information to an appropriate document or section. If the reset protocol is custom to this block, include a subsection defining the relevant procedures. Otherwise, cite other documents that provide these details [1].

11.1 Reset Domains Table

Reset Name	Synchronous/Asynchronous	Active High/Low	Associated Clock (if synchronous)	Description
RESET_1	Synchronous	Active Low	CLK1	Description of RESET_1
RESET_2	Asynchronous	Active High	N/A	Description of RESET_2

11.2 Annotated Block Diagram



11.3 Custom Reset Procedures

If the reset protocol is custom for this block, provide a detailed description of the procedures and mechanisms here. Otherwise, cite the relevant documentation.

11.3.1 Custom Reset Procedure

1. Step 1 of the procedure.
2. Step 2 of the procedure.
3. Step 3 of the procedure.

11.4 References to External Documents

For standard reset protocols, refer to the following documents:

- Standard Reset Protocol Document 1
- Standard Reset Protocol Document 2

12 Interrupts

This section describes the interrupts implemented in the design. For each interrupt, specify its name, the associated interrupt controller, number, priority, type (e.g., level or edge), triggering mechanism (e.g., rising/falling edge or high/low level), handling method (e.g., synchronous or asynchronous), and a brief description.

12.1 Interrupt Table

Interrupt Name	Controller	Number	Priority	Type	Triggers	Handling	Description
IRQ_1	Main	0	1	Edge	Falling	Synchronous	Description of IRQ_1
IRQ_2	Aux	1	3	Level	High	Asynchronous	Description of IRQ_2

13 Arbitration, Fairness, QoS, and Forward Progress Guarantees

This section addresses how the design handles arbitration between multiple traffic classes or concurrent transaction types that share resources or interfaces. It outlines the arbitration policies, fairness properties, QoS features, and forward progress guarantees [1].

13.1 Arbitration and Fairness

Define the arbitration policy for shared resources or interfaces. Describe how multiple traffic classes or transaction types are prioritized and how fairness is ensured. If applicable, detail any configurability features that allow adjustments to the arbitration policy or QoS settings. Clearly state if any traffic classes are unfairly treated and discuss the implication [1].

- **Arbitration Policy**: Describe the policy used for arbitration (e.g., round-robin, priority-based).
- **Fairness**: Explain how fairness is maintained among traffic classes.
- **Configurability**: Detail any features that allow users to control arbitration or QoS.

13.2 Quality-of-Service (QoS)

Discuss the QoS mechanisms implemented in the design. Describe how the system ensures different levels of service based on traffic class or transaction type, and how these mechanisms impact system performance [1].

- **QoS Features**: List and describe the QoS features supported by the design.
- **Impact on Performance**: Explain how QoS affects the performance of the system.

13.3 Forward Progress Guarantees

Ensure that the design provides forward progress guarantees, meaning that it avoids deadlock and livelock conditions. State any assumptions about the external system required to guarantee forward progress. Provide a high-level outline of the proof for these guarantees, and indicate if it can be formally verified [1].

- **Deadlock and Livelock Prevention**: Describe how the design prevents deadlock and livelock.
- **Assumptions**: State any assumptions needed for forward progress.
- **Proof Outline**: Provide a brief outline of the proof or verification approach for forward progress guarantees.

14 Debugging

This section describes the debugging mechanisms and non-mission-mode features available in the design. It includes details about debugging registers, debugging interfaces, and any special features that facilitate diagnostics and testing.

Warning

Testbenches do not count as debugging mechanisms!

15 Synthesis

This section presents the synthesis results of the design across different FPGAs. It includes a table displaying performance, area, and other relevant parameters for each evaluated FPGA.

15.1 Synthesis Results Table

FPGA	Maximum Frequency (MHz)	Area (LUTs)	Area (FFs)	Area (BRAMs)	Area (DSPs)	Comments
FPGA_A	200	5000	3000	10	20	Description of performance and area for FPGA_A
FPGA_B	250	4800	2800	12	18	Description of performance and area for FPGA_B
FPGA_C	180	5200	3100	8	22	Description of performance and area for FPGA_C

15.2 Additional Comments (Optional)

In this section, provide additional comments on the synthesis results, including observations on performance, optimization, and recommendations for future improvements or adjustments.

15.2.1 Observations

- **Performance:** Comment on the differences in maximum frequency achieved across different FPGAs and possible reasons for these differences.
- **Area:** Analyze how the area used varies among different FPGAs and if there are opportunities for optimization.
- **Recommendations:** Offer recommendations for design improvements based on the synthesis results.

16 Verification

This section details the types of tests applied to the block and presents relevant benchmark results.

16.1 Test Environment

Describe the tools, simulators, or testbeds used for verification and any relevant configuration or parameters for the tests.

16.1.1 Test Environment Table

Tool	Version	Relevant Configuration
Simulator X	2024.1	Configuration A, Parameter B
Tool Y	3.2.1	Configuration C, Parameter D

16.2 Tests

Describe the different types of tests applied to the block, such as functional tests, regression tests, and specific verification methodologies used (e.g., simulation, formal verification, emulation).

16.2.1 Tests Table

Test Type	Description	Tools Used
Functional Test	Verification that the block meets functional specifications.	Simulator X, Tool Y
Regression Test	Ensuring recent changes do not introduce errors in the design.	Simulator Z, Test Suite W
Formal Verification	Rigorous mathematical verification of design properties.	Tool A, Tool B

16.2.2 Test Results

Show the results of the tests applied to the system, ideally in a table format.

16.3 Benchmarks

Include benchmark results, such as performance metrics, comparisons with expected outcomes, and block performance under different conditions.

16.3.1 Benchmarks Table

Metric	Value	Comments
Maximum Frequency	200 MHz	Meets performance expectations.
Latency	10 ns	Within acceptable limits.
Resource Usage	5000 LUTs, 3000 FFs	Efficient in terms of utilized resources.

16.3.2 Benchmarks Results

Show the results of the benchmarks applied to the system, ideally in a table format.

16.4 Issues and Resolutions

Brief discussion of any issues found during verification and how they were resolved.

16.4.1 Issues and Resolutions Table

Issue	Description	Resolution
Simulation Error	Results inconsistent with expectations.	Adjusted simulator configuration.
Timing Error	Deviation in response time.	Optimized design in critical path.

16.5 Verification Summary

Provide a brief summary of the block's results from the verification and benchmark processes.

I.I Microarchitecture

17 Submodule 1

17.1 Description

Provide a description of the submodule here. Explain its function, role within the larger system, and any important design considerations or features.

17.2 I/O Table

Detail the submodule's input and output signals, including their name, direction, type, and description.

17.2.1 Input Table

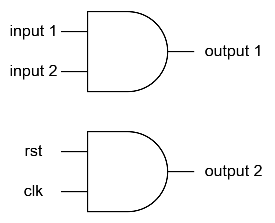
Input Name	Direction	Type	Description
`input_signal_1`	Input	`logic`	Description of `input_signal_1`
`input_signal_2`	Input	`logic`	Description of `input_signal_2`

17.2.2 Output Table

Output Name	Direction	Type	Description
`output_signal_1`	Output	`logic`	Description of `output_signal_1`
`output_signal_2`	Output	`logic`	Description of `output_signal_2`

17.3 Submodule Diagram

Include a diagram of the submodule here, showing its inputs, outputs, and how they are connected internally. Ensure the diagram is clear and properly labeled to facilitate understanding.



17.4 SystemVerilog Implementation

Include a brief description of the SystemVerilog code for the submodule, highlighting key parts of the implementation if needed for a clearer understanding.

Tip

Mkdocs allows a plethora of highlighting and cues for better code documentation. Read more.

17.4.1 Example Code

```
module Submodule (
    input  logic input_signal_1,
    input  logic input_signal_2,
    output logic output_signal_1,
    output logic output_signal_2
);
  // Description of the submodule's functionality

  // Module logic
  always_ff @(posedge clk) begin
    // Implementation of the functionality
  end
endmodule
```

18 Submodule 2

18.1 Description

Provide a description of the submodule here. Explain its function, role within the larger system, and any important design considerations or features.

18.2 I/O Table

Detail the submodule's input and output signals, including their name, direction, type, and description.

18.2.1 Input Table

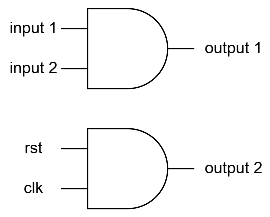
Input Name	Direction	Type	Description
`input_signal_1`	Input	`logic`	Description of `input_signal_1`
`input_signal_2`	Input	`logic`	Description of `input_signal_2`

18.2.2 Output Table

Output Name	Direction	Type	Description
`output_signal_1`	Output	`logic`	Description of `output_signal_1`
`output_signal_2`	Output	`logic`	Description of `output_signal_2`

18.3 Submodule Diagram

Include a diagram of the submodule here, showing its inputs, outputs, and how they are connected internally. Ensure the diagram is clear and properly labeled to facilitate understanding.



18.4 SystemVerilog Implementation

Include a brief description of the SystemVerilog code for the submodule, highlighting key parts of the implementation if needed for a clearer understanding.

Tip

Mkdocs allows a plethora of highlighting and cues for better code documentation. Read more.

18.4.1 Example Code

```
module Submodule (
    input  logic input_signal_1,
    input  logic input_signal_2,
    output logic output_signal_1,
    output logic output_signal_2
);
  // Description of the submodule's functionality

  // Module logic
  always_ff @(posedge clk) begin
    // Implementation of the functionality
  end
endmodule
```

19 Authors

This documentation template has been developed as part of a graduation project in Computer Engineering. Below are the authors responsible for the creation and development of this document:

- **Alejandro Chavarría**
- **Alejandro Soto**

This documentation template has been designed to provide a clear and comprehensive structure for the technical description of RTL hardware modules. We extend our gratitude to all collaborators and sources that contributed to the creation of this template.

For more information about the project, please refer to the associated GitHub organization OpenCEHardware.