Raisin64 Documentation

Release 0.1

Christopher Parish

Oct 09, 2018

CONTENTS:

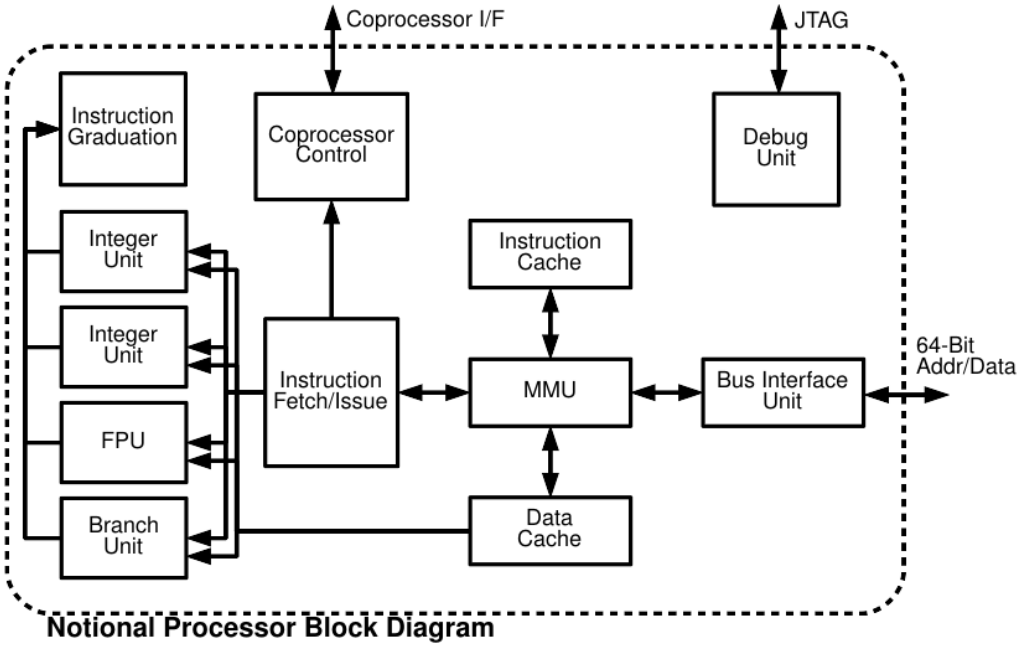
1 Raisin64 CPU **3**
- 1.1 Overview . . . 3
- 1.2 Pipeline Stages . . . 3
- 1.3 Caches . . . 3
- 1.4 MMU . . . 3
- 1.5 Interrupt Unit . . . 3
- 1.6 Debug Unit . . . 3
- 1.7 Verilog Module Index . . . 3

2 Code Snippets and Software **5**
- 2.1 Handling Interrupts . . . 5
- 2.2 Initializing the MMU . . . 5

3 Tools **7**
- 3.1 Assembler . . . 7
- 3.2 Debugging . . . 7

4 Nexys 4 DDR Reference Implementation **9**
- 4.1 SoC Peripherals . . . 9
- 4.2 Required Hardware . . . 9
- 4.3 Synthesizing the Core . . . 9

Raisin64 (*RISC Architecture with In-order Superscalar INterlocked-pipeline*) is a pure 64-bit CPU design created as part of an educational project. Architecturally similar to the MIPS R10000 and POWER3, Raisin64 is a superscalar design that employs multiple specialized pipelines for integer operations, floating point, load/store, etc. Unlike most superscalar designs, Raisin64 does not re-order instructions but instead provides a larger architectural register file of 64x64-bit registers.



Notional Processor Block Diagram

Major features of the Raisin64 include:

- **Bits:** 64-bit
- **Design:** RISC
- **Type:** Register-Register
- **Branching:** Condition Code
- **Endianness:** Big
- **Page Size:** 16KB Fixed
- **Virtual Address Size:** 47-Bits
- **Page Table:** Three Level
- **Registers:** 61 (R0 = 0)

CHAPTER

ONE

RAISIN64 CPU

1.1 Overview

1.2 Pipeline Stages

1.3 Caches

1.4 MMU

1.5 Interrupt Unit

1.6 Debug Unit

1.7 Verilog Module Index

The structured hierarchy of source code is also available in the module index:

1.7.1 Verilog Module Index

Raisin64.v

```
/*
 * Raisin64 CPU
 */

module raisin64 (
    //# {{clocks|Clocking}}
    input clk,
    input rst_n,

    //# {{data|Memory Interface}}
    input[63:0] mem_din,
    output[63:0] mem_dout,
    output[63:0] mem_add,

```

(continues on next page)

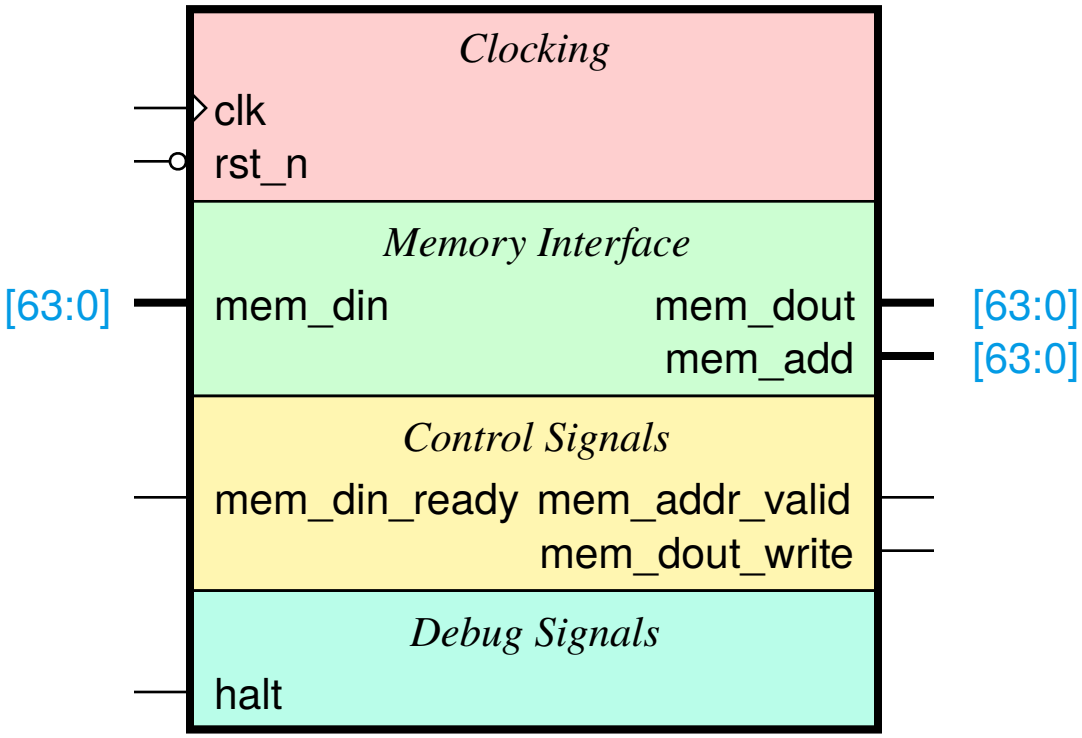


Fig. 1: Raisin64.v

(continued from previous page)

```
15      //# {{control|Control Signals}}
16      output mem_addr_valid,
17      output mem_dout_write
18      input mem_din_ready,
19
20      //# {{debug|Debug Signals}}
21      input halt);
22
23  endmodule
```

CHAPTER

TWO

CODE SNIPPETS AND SOFTWARE

2.1 Handling Interrupts

2.2 Initializing the MMU

CHAPTER

THREE

TOOLS

- *Assembler*
- *Debugging*
  - *Getting OpenOCD*

3.1 Assembler

3.2 Debugging

3.2.1 Getting OpenOCD

CHAPTER

FOUR

NEXYS 4 DDR REFERENCE IMPLEMENTATION

4.1 SoC Peripherals

4.2 Required Hardware

4.3 Synthesizing the Core