Processor Project Documentation

# Log

|  |  |
| --- | --- |
| Date | Info |
| 27.11.2016 | Initial version of the documentation |
| 28.11.2016 | Added ALU operation codes. Added entity interfaces for multiplier and adder\_subtractor |
| 30.11.2016 | Changed bit count to 8 as 4 seemed to be too small |

Contents

[Log 2](#_Toc468306568)

[Introduction 4](#_Toc468306569)

[Project Directory structure 5](#_Toc468306570)

[Top Level Structure 6](#_Toc468306571)

[Control Block 7](#_Toc468306572)

[Registers 8](#_Toc468306573)

[Arithmetic Logical Unit 9](#_Toc468306574)

[Adder Subtractor 9](#_Toc468306575)

[Multiplier 9](#_Toc468306576)

[Shifter 10](#_Toc468306577)

# Introduction

This project is completely done by Toni Lammi. The meaning of this project is to learn basics of vhdl and produce a virtual 8-bit processor on Altera Cyclone II FPGA board. Initially the processor is meant to be a 8-bit processor but the designs are designed to be easily scaled for larger bit-counts. This is done by using generic statements. If a block doesn’t support scalability, it is named with postfix of bit count: e.g. multiplier\_8bit.vhd.

# Project Directory structure

The project contains following directories:

db/ contains all temporary files created by the synthesis

do\_files/ has all the .do files used by ModelSim

doc/ cointains all the documentation not included in .vhd files (e.g. this file).

sim/ is where the model sim is started from

syn/ directory from which the Quartus synthesis tool is run.

tb/ contains all test bench files. Files are named with DUV’s name and postfix \_tb.vhd

vhd/ contains all .vhd files.

work/ is the working directory for MultiSim

# Top Level Structure

This processor consists of 3 main blocks: Control Unit, Registers, and Arithmetic Logical Unit. The first one is the block that controls the whole functionality of the design. Registers contain the values the processor works on: e.g. stack pointer and program counter. Lastly, ALU is the block that handles all the logical and arithmetic operations.

# Control Block

# Registers

# Arithmetic Logical Unit

ALU takes in 3 values with custom bit width. Two of the inputs are operands and one is used for identifying the operation that is done using these inputs. ALU also contains an output bus with same bit width as the inputs and multiple status flag outputs. With some operations, the second input can be omitted. **ALU is completely composed of combinatorial logic.**

|  |  |
| --- | --- |
| Operation ID (in hexadecimal) | Functionality |
| 0x00 | No operation |
| 0x01 | Add |
| 0x02 | Subtract operand 2 from operand 1 |
| 0x03 | Multiply unsigned |
| 0x04 | Multiply signed |
| 0x05 | Bitwise and |
| 0x06 | Bitwise or |
| 0x07 | Bitwise xor |
| 0x08 | Bitwise complement |
| 0x09 | Bitwise nor |
| 0x0a | Bitwise nand |
| 0x0b | Logical shift left, shift [operand 1] with [operand 2] bits |
| 0x0c | Logical shift right, shift [operand 1] with [operand 2] bits |
| 0x0c | Increment |
| 0x0e | Decrement |
| 0x0f |  |

ALU operation results are stored in [TBA: which register?].

## Adder Subtractor

Block used for both binary adding and subtracting binary (both signed and unsigned) numbers. Functionality is described in the tables below.

|  |  |
| --- | --- |
| Generics | Functionality |
| bit\_width\_g | Bus bit width |

|  |  |  |
| --- | --- | --- |
| Inputs | Width | Functionality |
| a\_in | bit\_width\_g | operand 1 |
| b\_in | bit\_width\_g | operand 2, added to or subtracted from operand 1 |
| mode\_in | 1 | ‘0’ for adding, ‘1’ for subtracting |

|  |  |  |
| --- | --- | --- |
| Outputs | Width | Functionality |
| result\_out | bit\_width\_g | Result of operation |
| overflow\_out | 1 | ‘1’ if an overflow happened in operation |

## Multiplier

Block for multiplying binary numbers. Same design can be used for both signed and unsigned multiplying with different generics.

|  |  |
| --- | --- |
| Generics | Functionality |
| input\_bit\_width\_g | Width of the input busses |
| signed\_mode\_g | ‘0’ for unsigned functionality, ‘1’ for signed |

|  |  |  |
| --- | --- | --- |
| Inputs | Width | Functionality |
| a\_in | input\_bit\_width\_g | Operand 1 |
| b\_in | input\_bit\_width\_g | Operand 2 |

|  |  |  |
| --- | --- | --- |
| Outputs | Width | Functionality |
| result\_out | input\_bit\_width\_g \* 2 | Result of operation |

## Shifter

This is a block for bit shifting. The input data is shifted by given bits to the left. Negative values cause shift to right.

|  |  |
| --- | --- |
| Generics | Functionality |
| bit\_width\_g | Bit width of the inputs and outputs |

|  |  |  |
| --- | --- | --- |
| Inputs | Width | Functionality |
| data\_in | bit\_width\_g | Data to be shifted |
| shift\_in | bit\_width\_g | Amount of shifting |

|  |  |  |
| --- | --- | --- |
| Outputs | Width | Functionality |
| data\_out | data\_width\_g | Shifted data |