Computer Architectures

Lab Report 1

Parameterizable ALU & Parameterizable REgister Bank

Y3839090 & Y3840426| Computer Architectures | ELE00009I

# **Session 1**

**ALUParam.vhd**

----------------------------------------------------------------------------------

--

-- Uni : University of York

-- Course : Electronic Engineering

-- Module : Computer Architectures

-- Engineers : Y3839090 & Y3840426

--

-- Create Date : 13:19:20 02/17/2017

-- Design Name : ALU\_param - Behavioral

-- Description : A paramateriable integer ALU.

--

----------------------------------------------------------------------------------

**library** IEEE**;**

**use** IEEE**.**STD\_LOGIC\_1164**.ALL;**

**use** IEEE**.**NUMERIC\_STD**.ALL;**

**use** work**.**DigEng**.ALL;**

**entity** ALU\_param **is**

**Generic** **(**

N **:** natural **:=** 8 -- data size in bits

**);**

**Port** **(**

A **:** **in** STD\_LOGIC\_VECTOR **(**N**-**1 **downto** 0**);**

B **:** **in** STD\_LOGIC\_VECTOR **(**N**-**1 **downto** 0**);**

X **:** **in** STD\_LOGIC\_VECTOR **(**log2**(**N**)-**1 **downto** 0**);** -- shift/rotate amount input

ctrl **:** **in** STD\_LOGIC\_VECTOR **(**3 **downto** 0**);** -- control signals from opcode

O **:** **out** STD\_LOGIC\_VECTOR **(**N**-**1 **downto** 0**);**

flags **:out** STD\_LOGIC\_VECTOR **(**7 **downto** 0**)** -- flags

**);**

**end** ALU\_param**;**

**architecture** Behavioral **of** ALU\_param **is**

-- internal signed signal for A and B inpy=uts

**signal** A\_itrn **:** SIGNED **(**N**-**1 **downto** 0**);**

**signal** B\_itrn **:** SIGNED **(**N**-**1 **downto** 0**);**

-- internal integer for X

**signal** X\_itrn **:** integer**;**

-- internal signed signal for output

**signal** O\_itrn **:** SIGNED **(**N**-**1 **downto** 0**);**

-- max positive and negitive N bit signed numbers

**constant** max\_pos **:** SIGNED **(**N**-**1 **downto** 0**)** **:=** **to\_signed((** 2 **\*\*** **(**N**-**1**)** **)** **-** 1**,** N**);**

**constant** max\_neg **:** SIGNED **(**N**-**1 **downto** 0**)** **:=** **to\_signed(** **-**2 **\*\*** **(**N**-**1**)** **,** N**);**

**begin**

**(Continued.)**

A\_itrn **<=** signed**(**A**);** -- converts A to signed and maps the result to A\_itrn

B\_itrn **<=** signed**(**B**);** -- converts A to signed and maps the result to B\_itrn

-- converts X to integer and maps the result to X\_itrn

X\_itrn **<=** **to\_integer(**unsigned**(**X**));**

-- converts O\_itrn to a plain std\_logic\_vector and maps it to O

O **<=** std\_logic\_vector**(**O\_itrn**);**

-- Main ALU multiplexer for each possible command

O\_itrn **<=**

A\_itrn **when** ctrl **=** "0000" **else** -- Output A

A\_itrn and B\_itrn **when** ctrl **=** "0100" **else** -- Output A & B

A\_itrn or B\_itrn **when** ctrl **=** "0101" **else** -- Output A || B

A\_itrn xor B\_itrn **when** ctrl **=** "0110" **else** -- Output A xor B

not A\_itrn **when** ctrl **=** "0111" **else** -- Output not A

A\_itrn **+** 1 **when** ctrl **=** "1000" **else** -- Output A + 1

A\_itrn **-** 1 **when** ctrl **=** "1001" **else** -- Output A - 1

A\_itrn **+** B\_itrn **when** ctrl **=** "1010" **else** -- Output A + B

A\_itrn **-** B\_itrn **when** ctrl **=** "1011" **else** -- Output A - B

**SHIFT\_LEFT** **(**A\_itrn **,** X\_itrn**)** **when** ctrl **=** "1100" **else** -- Output A sla X

**SHIFT\_RIGHT** **(**A\_itrn **,** X\_itrn**)** **when** ctrl **=** "1101" **else** -- Output A sra X

**ROTATE\_LEFT** **(**A\_itrn **,** X\_itrn**)** **when** ctrl **=** "1110" **else** -- Output A rotl X

**ROTATE\_RIGHT** **(**A\_itrn **,** X\_itrn**)** **when** ctrl **=** "1111" **else** -- Output A rotr X

**(others** **=>**'U'**);**

-- Overflow flag

flags**(**7**)** **<=**

-- Will overflow if you add one to the max positive value

'1' **when** ctrl **=** "1000" and A\_itrn **=** max\_pos **else**

-- Will overflow if you minus one to the max negitive value

'1' **when** ctrl **=** "1001" and A\_itrn **=** max\_neg **else**

-- Will overflow if two neg values added give a pos result

'1' **when** ctrl **=** "1010" and A\_itrn**(**N**-**1**)** **=** '1' and B\_itrn**(**N**-**1**)** **=** '1' and O\_itrn**(**N**-**1**)** **=** '0'

**else**

-- Will overflow if two pos values added give a neg result

'1' **when** ctrl **=** "1010" and A\_itrn**(**N**-**1**)** **=** '0' and B\_itrn**(**N**-**1**)** **=** '0' and O\_itrn**(**N**-**1**)** **=** '1'

**else**

-- Will overflow if a pos value is subtracted from a neg value gives a pos result

'1' **when** ctrl **=** "1011" and A\_itrn**(**N**-**1**)** **=** '1' and B\_itrn**(**N**-**1**)** **=** '0' and O\_itrn**(**N**-**1**)** **=** '0'

**else**

-- Will overflow if a neg value is subtracted from a pos value gives a neg result

'1' **when** ctrl **=** "1011" and A\_itrn**(**N**-**1**)** **=** '0' and B\_itrn**(**N**-**1**)** **=** '1' and O\_itrn**(**N**-**1**)** **=** '1'

-- If none of the above are true then the result hasn't overflown

**else** '0'**;**

**(Continued.)**

flags**(**6**)** **<=** '1' **when** O\_itrn **>=** 0 **else** '0'**;** -- grater than or equal to zero

flags**(**5**)** **<=** '1' **when** O\_itrn **<=** 0 **else** '0'**;** -- less than or equal to zero

flags**(**4**)** **<=** '1' **when** O\_itrn **>** 0 **else** '0'**;** -- grater than zero

flags**(**3**)** **<=** '1' **when** O\_itrn **<** 0 **else** '0'**;** -- less than zero

flags**(**2**)** **<=** '1' **when** O\_itrn **=** 1 **else** '0'**;** -- one flag

flags**(**1**)** **<=** '1' **when** O\_itrn **/=** 0 **else** '0'**;** -- not zero flag

flags**(**0**)** **<=** '1' **when** O\_itrn **=** 0 **else** '0'**;** -- zero flag

**end** Behavioral**;**

**ALUParamTB.vhd**

----------------------------------------------------------------------------------

--

-- Uni : University of York

-- Course : Electronic Engineering

-- Module : Computer Architectures

-- Engineers : Y3839090 & Y3840426

--

-- Create Date : 14:47:27 02/17/2017

-- Design Name : ALU\_param\_TB - TestBench

-- Description : A testbench for a 16 bit integer ALU.

--

----------------------------------------------------------------------------------

**LIBRARY** ieee**;**

**USE** ieee**.**std\_logic\_1164**.ALL;**

**USE** ieee**.**numeric\_std**.ALL;**

**USE** work**.**DigEng**.ALL;**

**entity** ALU\_param\_TB **is**

**end** ALU\_param\_TB**;**

**architecture** behavior **of** ALU\_param\_TB **is**

-- ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ====

-- From http://stackoverflow.com/a/24336034 By Morten Zilmer

-- Allows printing a std\_logic\_vector as a string that represents it's binary form.

-- ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ====

**function** to\_bstring**(**sl **:** std\_logic**)** **return** string **is**

**variable** sl\_str\_v **:** string**(**1 **to** 3**);** -- std\_logic image with quotes around

**begin**

sl\_str\_v **:=** std\_logic'**image(**sl**);**

**return** "" **&** sl\_str\_v**(**2**);** -- "" & character to get string

**end** **function;**

**(Continued.)**

**function** to\_bstring**(**slv **:** std\_logic\_vector**)** **return** string **is**

**alias** slv\_norm **:** std\_logic\_vector**(**1 **to** slv'**length)** **is** slv**;**

**variable** sl\_str\_v **:** string**(**1 **to** 1**);** -- String of std\_logic

**variable** res\_v **:** string**(**1 **to** slv'**length);**

**begin**

**for** idx **in** slv\_norm'**range** **loop**

sl\_str\_v **:=** to\_bstring**(**slv\_norm**(**idx**));**

res\_v**(**idx**)** **:=** sl\_str\_v**(**1**);**

**end** **loop;**

**return** res\_v**;**

**end** **function;**

-- ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ==== ====

-- converts an std\_logic\_vector to a string that represents it's signed value

**function** s\_tostr**(val** **:** std\_logic\_vector**)** **return** string **is**

**begin**

**return** integer'**image(** **to\_integer(**signed**(val))** **);**

**end** **function;**

-- Constants

**constant** M **:** NATURAL **:=** 16**;** -- We are testing a 16 bit ALU

**constant** wait\_time **:** TIME **:=** 10 ns**;**

-- Component Declaration for the Unit Under Test (UUT)

**component** ALU\_param

**generic**

**(**

N **:** NATURAL -- The number of bits this alu will operate on

**);**

**port**

**(**

A **:** **IN** STD\_LOGIC\_VECTOR**(**M**-**1 **downto** 0**);**

B **:** **IN** STD\_LOGIC\_VECTOR**(**M**-**1 **downto** 0**);**

X **:** **IN** STD\_LOGIC\_VECTOR**(**log2**(**M**)-**1 **downto** 0**);**

ctrl **:** **IN** STD\_LOGIC\_VECTOR**(**3 **downto** 0**);**

O **:** **OUT** STD\_LOGIC\_VECTOR**(**M**-**1 **downto** 0**);**

flags **:** **OUT** STD\_LOGIC\_VECTOR**(**7 **downto** 0**)**

**);**

**end** **component;**

--Inputs

**signal** A **:** STD\_LOGIC\_VECTOR**(**M**-**1 **downto** 0**)** **:=** **(others** **=>** '0'**);**

**signal** B **:** STD\_LOGIC\_VECTOR**(**M**-**1 **downto** 0**)** **:=** **(others** **=>** '0'**);**

**signal** X **:** STD\_LOGIC\_VECTOR**(**log2**(**M**)-**1 **downto** 0**)** **:=** **(others** **=>** '0'**);**

**signal** ctrl **:** STD\_LOGIC\_VECTOR**(**3 **downto** 0**)** **:=** **(others** **=>** '0'**);**

--Outputs

**signal** O **:** STD\_LOGIC\_VECTOR**(**M**-**1 **downto** 0**);**

**signal** flags **:** STD\_LOGIC\_VECTOR**(**7 **downto** 0**);**

**(Continued.)**

**type** TEST\_VECTOR\_ARRAY **is** **ARRAY(**NATURAL **range** **<>)** **of** TEST\_VECTOR**;**

**constant** test\_vectors **:** TEST\_VECTOR\_ARRAY **:=** **(**

-- CTRL, A , B , X , O , FLAGS

-- A

**(**"0000"**,** X"0000"**,** X"0000"**,** X"0"**,** X"0000"**,** "01100001"**),**

**(**"0000"**,** X"FF9C"**,** X"0000"**,** X"0"**,** X"FF9C"**,** "00101010"**),**

**(**"0000"**,** X"03E8"**,** X"0000"**,** X"0"**,** X"03E8"**,** "01010010"**),**

**(**"0000"**,** X"0001"**,** X"0000"**,** X"0"**,** X"0001"**,** "01010110"**),**

-- A and B

**(**"0100"**,** X"0000"**,** X"0000"**,** X"0"**,** X"0000"**,** "01100001"**),**

**(**"0100"**,** X"FF94"**,** X"FC17"**,** X"0"**,** X"FC14"**,** "00101010"**),**

**(**"0100"**,** X"03E8"**,** X"5213"**,** X"0"**,** X"0200"**,** "01010010"**),**

**(**"0100"**,** X"FFFF"**,** X"10E0"**,** X"0"**,** X"10E0"**,** "01010010"**),**

**(**"0100"**,** X"55FF"**,** X"AAAA"**,** X"0"**,** X"00AA"**,** "01010010"**),**

-- A or B

**(**"0101"**,** X"0000"**,** X"0000"**,** X"0"**,** X"0000"**,** "01100001"**),**

**(**"0101"**,** X"FF94"**,** X"FC17"**,** X"0"**,** X"FF97"**,** "00101010"**),**

**(**"0101"**,** X"03E8"**,** X"5213"**,** X"0"**,** X"53FB"**,** "01010010"**),**

**(**"0101"**,** X"FFFF"**,** X"10E0"**,** X"0"**,** X"FFFF"**,** "00101010"**),**

**(**"0101"**,** X"5555"**,** X"AAAA"**,** X"0"**,** X"FFFF"**,** "00101010"**),**

-- A xor B

**(**"0110"**,** X"0000"**,** X"0000"**,** X"0"**,** X"0000"**,** "01100001"**),**

**(**"0110"**,** X"FF94"**,** X"FC17"**,** X"0"**,** X"0383"**,** "01010010"**),**

**(**"0110"**,** X"03E8"**,** X"5213"**,** X"0"**,** X"51FB"**,** "01010010"**),**

**(**"0110"**,** X"FFFF"**,** X"10E0"**,** X"0"**,** X"EF1F"**,** "00101010"**),**

**(**"0110"**,** X"5555"**,** X"AAAA"**,** X"0"**,** X"FFFF"**,** "00101010"**),**

-- not A

**(**"0111"**,** X"0000"**,** X"0000"**,** X"0"**,** X"FFFF"**,** "00101010"**),**

**(**"0111"**,** X"FFFF"**,** X"0000"**,** X"0"**,** X"0000"**,** "01100001"**),**

**(**"0111"**,** X"8111"**,** X"0000"**,** X"0"**,** X"7EEE"**,** "01010010"**),**

**(**"0111"**,** X"0001"**,** X"0000"**,** X"0"**,** X"FFFE"**,** "00101010"**),**

-- A + 1

**(**"1000"**,** X"0000"**,** X"0000"**,** X"0"**,** X"0001"**,** "01010110"**),**

**(**"1000"**,** X"FFFF"**,** X"0000"**,** X"0"**,** X"0000"**,** "01100001"**),**

**(**"1000"**,** X"7fff"**,** X"0000"**,** X"0"**,** X"8000"**,** "10101010"**),**

-- A - 1

**(**"1001"**,** X"0000"**,** X"0000"**,** X"0"**,** X"ffff"**,** "00101010"**),**

**(**"1001"**,** X"0001"**,** X"0000"**,** X"0"**,** X"0000"**,** "01100001"**),**

**(**"1001"**,** X"8000"**,** X"0000"**,** X"0"**,** X"7fff"**,** "11010010"**),**

-- A + B

**(**"1010"**,** X"0000"**,** X"0000"**,** X"0"**,** X"0000"**,** "01100001"**),**

**(**"1010"**,** X"0310"**,** X"0a00"**,** X"0"**,** X"0D10"**,** "01010010"**),**

**(**"1010"**,** X"09E2"**,** X"f43e"**,** X"0"**,** X"FE20"**,** "00101010"**),**

**(**"1010"**,** X"7fff"**,** X"0001"**,** X"0"**,** X"8000"**,** "10101010"**),**

**(Continued.)**

**type** TEST\_VECTOR\_ARRAY **is** **ARRAY(**NATURAL **range** **<>)** **of** TEST\_VECTOR**;**

**constant** test\_vectors **:** TEST\_VECTOR\_ARRAY **:=** **(**

-- CTRL, A , B , X , O , FLAGS

-- A

**(**"0000"**,** X"0000"**,** X"0000"**,** X"0"**,** X"0000"**,** "01100001"**),**

**(**"0000"**,** X"FF9C"**,** X"0000"**,** X"0"**,** X"FF9C"**,** "00101010"**),**

**(**"0000"**,** X"03E8"**,** X"0000"**,** X"0"**,** X"03E8"**,** "01010010"**),**

**(**"0000"**,** X"0001"**,** X"0000"**,** X"0"**,** X"0001"**,** "01010110"**),**

-- A and B

**(**"0100"**,** X"0000"**,** X"0000"**,** X"0"**,** X"0000"**,** "01100001"**),**

**(**"0100"**,** X"FF94"**,** X"FC17"**,** X"0"**,** X"FC14"**,** "00101010"**),**

**(**"0100"**,** X"03E8"**,** X"5213"**,** X"0"**,** X"0200"**,** "01010010"**),**

**(**"0100"**,** X"FFFF"**,** X"10E0"**,** X"0"**,** X"10E0"**,** "01010010"**),**

**(**"0100"**,** X"55FF"**,** X"AAAA"**,** X"0"**,** X"00AA"**,** "01010010"**),**

-- A or B

**(**"0101"**,** X"0000"**,** X"0000"**,** X"0"**,** X"0000"**,** "01100001"**),**

**(**"0101"**,** X"FF94"**,** X"FC17"**,** X"0"**,** X"FF97"**,** "00101010"**),**

**(**"0101"**,** X"03E8"**,** X"5213"**,** X"0"**,** X"53FB"**,** "01010010"**),**

**(**"0101"**,** X"FFFF"**,** X"10E0"**,** X"0"**,** X"FFFF"**,** "00101010"**),**

**(**"0101"**,** X"5555"**,** X"AAAA"**,** X"0"**,** X"FFFF"**,** "00101010"**),**

-- A xor B

**(**"0110"**,** X"0000"**,** X"0000"**,** X"0"**,** X"0000"**,** "01100001"**),**

**(**"0110"**,** X"FF94"**,** X"FC17"**,** X"0"**,** X"0383"**,** "01010010"**),**

**(**"0110"**,** X"03E8"**,** X"5213"**,** X"0"**,** X"51FB"**,** "01010010"**),**

**(**"0110"**,** X"FFFF"**,** X"10E0"**,** X"0"**,** X"EF1F"**,** "00101010"**),**

**(**"0110"**,** X"5555"**,** X"AAAA"**,** X"0"**,** X"FFFF"**,** "00101010"**),**

-- not A

**(**"0111"**,** X"0000"**,** X"0000"**,** X"0"**,** X"FFFF"**,** "00101010"**),**

**(**"0111"**,** X"FFFF"**,** X"0000"**,** X"0"**,** X"0000"**,** "01100001"**),**

**(**"0111"**,** X"8111"**,** X"0000"**,** X"0"**,** X"7EEE"**,** "01010010"**),**

**(**"0111"**,** X"0001"**,** X"0000"**,** X"0"**,** X"FFFE"**,** "00101010"**),**

-- A + 1

**(**"1000"**,** X"0000"**,** X"0000"**,** X"0"**,** X"0001"**,** "01010110"**),**

**(**"1000"**,** X"FFFF"**,** X"0000"**,** X"0"**,** X"0000"**,** "01100001"**),**

**(**"1000"**,** X"7fff"**,** X"0000"**,** X"0"**,** X"8000"**,** "10101010"**),**

-- A - 1

**(**"1001"**,** X"0000"**,** X"0000"**,** X"0"**,** X"ffff"**,** "00101010"**),**

**(**"1001"**,** X"0001"**,** X"0000"**,** X"0"**,** X"0000"**,** "01100001"**),**

**(**"1001"**,** X"8000"**,** X"0000"**,** X"0"**,** X"7fff"**,** "11010010"**),**

-- A + B

**(**"1010"**,** X"0000"**,** X"0000"**,** X"0"**,** X"0000"**,** "01100001"**),**

**(**"1010"**,** X"0310"**,** X"0a00"**,** X"0"**,** X"0D10"**,** "01010010"**),**

**(**"1010"**,** X"09E2"**,** X"f43e"**,** X"0"**,** X"FE20"**,** "00101010"**),**

**(**"1010"**,** X"7fff"**,** X"0001"**,** X"0"**,** X"8000"**,** "10101010"**),**

**(Continued.)**

-- A - B

**(**"1011"**,** X"0000"**,** X"0000"**,** X"0"**,** X"0000"**,** "01100001"**),**

**(**"1011"**,** X"0310"**,** X"0a00"**,** X"0"**,** X"F910"**,** "00101010"**),**

**(**"1011"**,** X"09E2"**,** X"0BC2"**,** X"0"**,** X"FE20"**,** "00101010"**),**

**(**"1011"**,** X"8000"**,** X"0001"**,** X"0"**,** X"7fff"**,** "11010010"**),**

-- A sla x

**(**"1100"**,** X"0000"**,** X"0000"**,** X"0"**,** X"0000"**,** "01100001"**),**

**(**"1100"**,** X"0000"**,** X"0000"**,** X"4"**,** X"0000"**,** "01100001"**),**

**(**"1100"**,** X"1111"**,** X"0000"**,** X"1"**,** X"2222"**,** "01010010"**),**

**(**"1100"**,** X"1111"**,** X"0000"**,** X"3"**,** X"8888"**,** "00101010"**),**

**(**"1100"**,** X"5555"**,** X"0000"**,** X"9"**,** X"AA00"**,** "00101010"**),**

-- A sra x

**(**"1101"**,** X"0000"**,** X"0000"**,** X"0"**,** X"0000"**,** "01100001"**),**

**(**"1101"**,** X"0000"**,** X"0000"**,** X"4"**,** X"0000"**,** "01100001"**),**

**(**"1101"**,** X"8888"**,** X"0000"**,** X"1"**,** X"C444"**,** "00101010"**),**

**(**"1101"**,** X"8888"**,** X"0000"**,** X"3"**,** X"f111"**,** "00101010"**),**

**(**"1101"**,** X"AAAA"**,** X"0000"**,** X"9"**,** X"FFD5"**,** "00101010"**),**

-- A rotl x

**(**"1110"**,** X"0000"**,** X"0000"**,** X"2"**,** X"0000"**,** "01100001"**),**

**(**"1110"**,** X"8888"**,** X"0000"**,** X"1"**,** X"1111"**,** "01010010"**),**

**(**"1110"**,** X"8101"**,** X"0000"**,** X"1"**,** X"0203"**,** "01010010"**),**

-- A rotr x

**(**"1111"**,** X"0000"**,** X"0000"**,** X"2"**,** X"0000"**,** "01100001"**),**

**(**"1111"**,** X"1111"**,** X"0000"**,** X"1"**,** X"8888"**,** "00101010"**),**

**(**"1111"**,** X"1081"**,** X"0000"**,** X"1"**,** X"8840"**,** "00101010"**),**

-- THIS TEST WILL FAIL

**(**"1010"**,** X"0000"**,** X"0000"**,** X"0"**,** X"FFFF"**,** "10000001"**)**

**);**

**begin**

-- Instantiate the Unit Under Test (UUT)

uut**:** ALU\_param

**generic** **map**

**(**

N **=>** M

**)**

**port** **map**

**(**

A **=>** A**,**

B **=>** B**,**

X **=>** X**,**

ctrl **=>** ctrl**,**

O **=>** O**,**

flags **=>** flags

**);**

**(Continued.)**

-- Stimulus process

stim\_proc**:** **process**

**begin**

-- hold reset state for 100 ns.

**wait** **for** 100 ns**;**

-- run the test for every set of data

**for** i **in** test\_vectors'**range** **loop**

-- assign test inputs

ctrl **<=** test\_vectors**(**i**).**ctrl**;**

A **<=** test\_vectors**(**i**).**A**;**

B **<=** test\_vectors**(**i**).**B**;**

X **<=** test\_vectors**(**i**).**X**;**

-- wait long enough for the ALU to process

**wait** **for** wait\_time**;**

-- check that the actual output is the same as the expect output

**assert** O **=** test\_vectors**(**i**).**O

**report** " [ERR!] Test " **&** integer'**image(**i**)&**

" Actual output did not equal expected output: Actual " **&** s\_tostr**(**O**)** **&**

" , Expected " **&** s\_tostr**(**test\_vectors**(**i**).**O**)**

**severity** error**;**

-- check that the actual flags is the same as the expect flags

**assert** flags **=** test\_vectors**(**i**).**flags

**report** " [ERR!] Test " **&** integer'**image(**i**)&**

" Actual flags did not equal expected flags : Actual " **&** to\_bstring**(**flags**)** **&**

" , Expected " **&** to\_bstring**(**test\_vectors**(**i**).**flags**)**

**severity** error**;**

-- if there were no isses report that the test was successful

**assert** not **(** O **=** test\_vectors**(**i**).**O and flags **=** test\_vectors**(**i**).**flags **)**

**report** " [ OK ] Test " **&** integer'**image(**i**)&** " was successful!"

**severity** note**;**

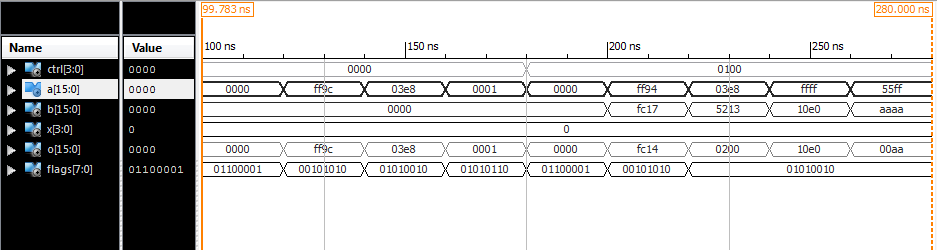
**wait** **for** wait\_time**;**

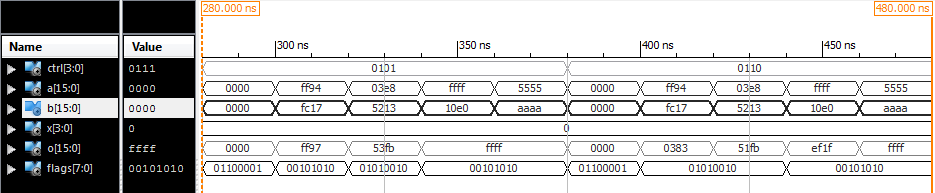
**end** **loop;**

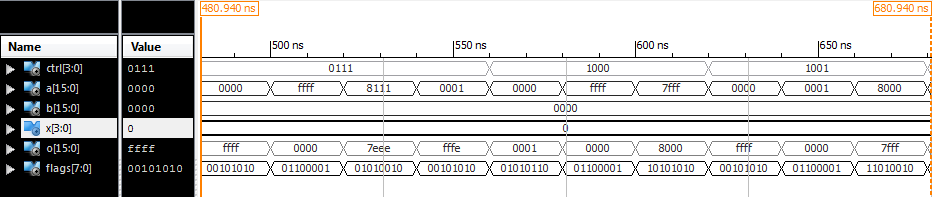
**wait;**

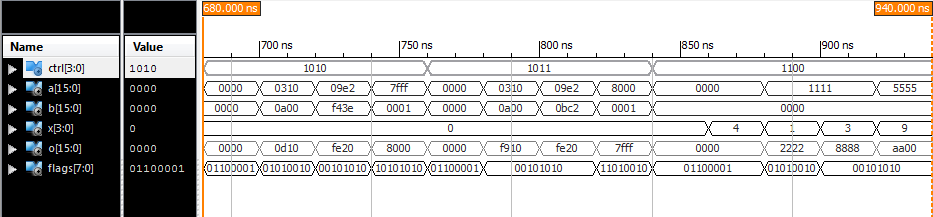
**end** **process;**

**end;**

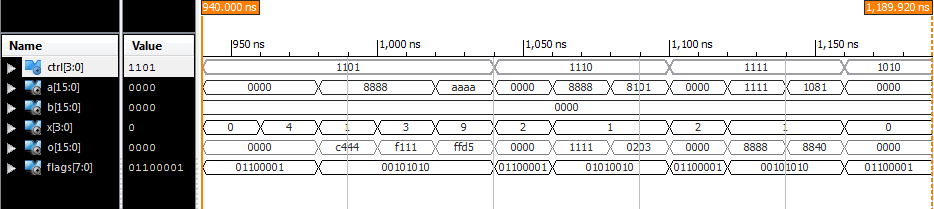
**Simulation Screenshot**

****Shows the operation of giving A as the output (OP 0000), and giving the result of A AND B as the output (OP 0100)

****Shows the operation of giving the result of A OR B as the output (OP 0101), and giving the result of A XOR B as the output (OP 0110)

****Shows the operation of giving the result of NOT A as the output (OP 0111), giving the result of A + 1 as the output (OP 1000) and giving the result of A-1 (OP 1001)

Shows the operation of giving the result of A+B as the output (OP 1010), giving the result of A - B as the output (OP 1011) and giving the result of an arithmetic shift of input A by X bits left as the output (OP 1100)

****Shows the operation of giving the result of an arithmetic shift of input A by X bits right as the output (OP 1101), giving the result of a left rotation of A by X bits as the output (OP 1110) and giving the result of a right shift of A by X bits as the output (OP 1111). It also shows an A+B operation (OP 1010) used for the purpose of showing an erroneous set of inputs/outputs to verify the correct operation of assert clause. The error is seen below in the ISim console output.

**ISim Console Output**

ISim P.28xd (signature 0xa0883be4)

This is a Full version of ISim.

Time resolution is 1 ps

WARNING: Simulation object /alu\_param\_tb/test\_vectors was not traceable in the design for the following reason:

ISim does not yet support tracing of constant and generic multi-dimensional arrays.

Simulator is doing circuit initialization process.

Finished circuit initialization process.

at 110 ns: Note: [ OK ] Test 0 was successful! (/alu\_param\_tb/).

at 130 ns: Note: [ OK ] Test 1 was successful! (/alu\_param\_tb/).

at 150 ns: Note: [ OK ] Test 2 was successful! (/alu\_param\_tb/).

at 170 ns: Note: [ OK ] Test 3 was successful! (/alu\_param\_tb/).

at 190 ns: Note: [ OK ] Test 4 was successful! (/alu\_param\_tb/).

at 210 ns: Note: [ OK ] Test 5 was successful! (/alu\_param\_tb/).

at 230 ns: Note: [ OK ] Test 6 was successful! (/alu\_param\_tb/).

at 250 ns: Note: [ OK ] Test 7 was successful! (/alu\_param\_tb/).

at 270 ns: Note: [ OK ] Test 8 was successful! (/alu\_param\_tb/).

at 290 ns: Note: [ OK ] Test 9 was successful! (/alu\_param\_tb/).

at 310 ns: Note: [ OK ] Test 10 was successful! (/alu\_param\_tb/).

at 330 ns: Note: [ OK ] Test 11 was successful! (/alu\_param\_tb/).

at 350 ns: Note: [ OK ] Test 12 was successful! (/alu\_param\_tb/).

at 370 ns: Note: [ OK ] Test 13 was successful! (/alu\_param\_tb/).

at 390 ns: Note: [ OK ] Test 14 was successful! (/alu\_param\_tb/).

at 410 ns: Note: [ OK ] Test 15 was successful! (/alu\_param\_tb/).

at 430 ns: Note: [ OK ] Test 16 was successful! (/alu\_param\_tb/).

at 450 ns: Note: [ OK ] Test 17 was successful! (/alu\_param\_tb/).

at 470 ns: Note: [ OK ] Test 18 was successful! (/alu\_param\_tb/).

at 490 ns: Note: [ OK ] Test 19 was successful! (/alu\_param\_tb/).

at 510 ns: Note: [ OK ] Test 20 was successful! (/alu\_param\_tb/).

at 530 ns: Note: [ OK ] Test 21 was successful! (/alu\_param\_tb/).

at 550 ns: Note: [ OK ] Test 22 was successful! (/alu\_param\_tb/).

at 570 ns: Note: [ OK ] Test 23 was successful! (/alu\_param\_tb/).

at 590 ns: Note: [ OK ] Test 24 was successful! (/alu\_param\_tb/).

at 610 ns: Note: [ OK ] Test 25 was successful! (/alu\_param\_tb/).

at 630 ns: Note: [ OK ] Test 26 was successful! (/alu\_param\_tb/).

at 650 ns: Note: [ OK ] Test 27 was successful! (/alu\_param\_tb/).

at 670 ns: Note: [ OK ] Test 28 was successful! (/alu\_param\_tb/).

at 690 ns: Note: [ OK ] Test 29 was successful! (/alu\_param\_tb/).

at 710 ns: Note: [ OK ] Test 30 was successful! (/alu\_param\_tb/).

at 730 ns: Note: [ OK ] Test 31 was successful! (/alu\_param\_tb/).

at 750 ns: Note: [ OK ] Test 32 was successful! (/alu\_param\_tb/).

at 770 ns: Note: [ OK ] Test 33 was successful! (/alu\_param\_tb/).

at 790 ns: Note: [ OK ] Test 34 was successful! (/alu\_param\_tb/).

at 810 ns: Note: [ OK ] Test 35 was successful! (/alu\_param\_tb/).

at 830 ns: Note: [ OK ] Test 36 was successful! (/alu\_param\_tb/).

at 850 ns: Note: [ OK ] Test 37 was successful! (/alu\_param\_tb/).

at 870 ns: Note: [ OK ] Test 38 was successful! (/alu\_param\_tb/).

at 890 ns: Note: [ OK ] Test 39 was successful! (/alu\_param\_tb/).

at 910 ns: Note: [ OK ] Test 40 was successful! (/alu\_param\_tb/).

at 930 ns: Note: [ OK ] Test 41 was successful! (/alu\_param\_tb/).

at 950 ns: Note: [ OK ] Test 42 was successful! (/alu\_param\_tb/).

at 970 ns: Note: [ OK ] Test 43 was successful! (/alu\_param\_tb/).

at 990 ns: Note: [ OK ] Test 44 was successful! (/alu\_param\_tb/).

at 1010 ns: Note: [ OK ] Test 45 was successful! (/alu\_param\_tb/).

at 1030 ns: Note: [ OK ] Test 46 was successful! (/alu\_param\_tb/).

at 1050 ns: Note: [ OK ] Test 47 was successful! (/alu\_param\_tb/).

at 1070 ns: Note: [ OK ] Test 48 was successful! (/alu\_param\_tb/).

at 1090 ns: Note: [ OK ] Test 49 was successful! (/alu\_param\_tb/).

at 1110 ns: Note: [ OK ] Test 50 was successful! (/alu\_param\_tb/).

at 1130 ns: Note: [ OK ] Test 51 was successful! (/alu\_param\_tb/).

at 1150 ns: Note: [ OK ] Test 52 was successful! (/alu\_param\_tb/).

at 1170 ns: Error: [ERR!] Test 53 Actual output did not equal expected output: Actual 0 , Expected -1

at 1170 ns: Error: [ERR!] Test 53 Actual flags did not equal expected flags : Actual 01100001 , Expected 10000001

ISim>

**ISim Console Output**

=========================================================================

\* HDL Synthesis \*

=========================================================================

Synthesizing Unit <ALU\_param>.

Related source file is "E:\University\\_Second Year\Computer Architectures\assesment\MyMicroProccessor\Lab\_1\ParameterizableALU\ALU\_param.vhd".

N = 8

Found 8-bit adder for signal <A\_itrn[7]\_B\_itrn[7]\_add\_27\_OUT> created at line 69.

Found 8-bit adder for signal <A\_itrn[7]\_GND\_5\_o\_add\_31\_OUT> created at line 1253.

Found 8-bit subtractor for signal <A\_itrn[7]\_B\_itrn[7]\_sub\_26\_OUT<7:0>> created at line 70.

Found 8-bit subtractor for signal <A\_itrn[7]\_GND\_5\_o\_sub\_30\_OUT<7:0>> created at line 1320.

Found 8-bit shifter rotate right for signal <A\_itrn[7]\_X\_itrn[30]\_rotate\_right\_17\_OUT> created at line 3021

Found 8-bit shifter rotate left for signal <A\_itrn[7]\_X\_itrn[30]\_rotate\_left\_19\_OUT> created at line 3012

Found 8-bit shifter arithmetic right for signal <A\_itrn[7]\_X\_itrn[30]\_shift\_right\_21\_OUT> created at line 2982

Found 8-bit shifter logical left for signal <A\_itrn[7]\_X\_itrn[30]\_shift\_left\_23\_OUT> created at line 2973

Found 8-bit 13-to-1 multiplexer for signal <O> created at line 27.

Found 8-bit comparator greater for signal <flags<3>> created at line 99

Found 8-bit comparator greater for signal <flags<4>> created at line 100

Summary:

inferred 1 Adder/Subtractor(s).

inferred 2 Comparator(s).

inferred 16 Multiplexer(s).

inferred 4 Combinational logic shifter(s).

Unit <ALU\_param> synthesized.

=========================================================================

HDL Synthesis Report

Macro Statistics

# Adders/Subtractors : 1

8-bit addsub : 1

# Comparators : 2

8-bit comparator greater : 2

# Multiplexers : 16

1-bit 2-to-1 multiplexer : 6

8-bit 2-to-1 multiplexer : 10

# Logic shifters : 4

8-bit shifter arithmetic right : 1

8-bit shifter logical left : 1

8-bit shifter rotate left : 1

8-bit shifter rotate right : 1

# Xors : 1

8-bit xor2 : 1

=========================================================================