Panagiotis Mousouliotis – Aristotle University of Thessaloniki

Combinational Circuit Design using VHDL and FPGAs
Project 3
Design and In-Circuit Verification of a 16-bit ALU

ALU Design Specifications

An Arithmetic Logic Unit (ALU) is a combinational circuit that performs arithmetic and logical operations. **Figure 1** shows the specifications of the 16-bit ALU that will be implemented in this project.

Inputs :	A, B, S_OP		
Outputs :	RES, COUT, VOUT		
Behavior-Functionality			
S_OP (4 bits)	**RES (16 bits)**	**COUT (1 bit)**	**VOUT (1 bit)**
0000	A+B	Carry from the msb of the result	Two's complement overflow
0001	A−B	Logic 1 only if B>A, else logic 0	Two's complement overflow
0010	A XOR B	Z	Z
0011	A AND B	Z	Z
0100	A OR B	Z	Z
0101	LSL(A)	Z	Z
0110	LSR(A)	Z	Z
0111	ROL(A)	Z	Z
1000	ROR(A)	Z	Z
1001	ASR(A)	Z	Z
1010	SWAP(A)	Z	Z
OTHERS	Z	Z	Z

Notes :

```
LSL  : 1-bit Logical Shift Left
LSR  : 1-bit Logical Shift Right
ROL  : 1-bit Rotate Left
ROR  : 1-bit Rotate Right
ASR  : 1-bit Arithmetic Shift Right
SWAP : Swap half words
```

Z represents high impedance

Figure 1

Inputs `A` and `B` are the 16-bit operands. Input `S_OP` is the select operation signal. Output signals value depends on `S_OP` signal's value as shown in **Figure 1**. For example when `S_OP` signal's value is equal to `0000`, `RES` signal's value is equal to `A+B`, `COUT` signal's value is equal to the carry from the msb of the result and `VOUT` signal's value is equal to the two's complement overflow.

Design and In-Circuit Verification of a 16-bit ALU

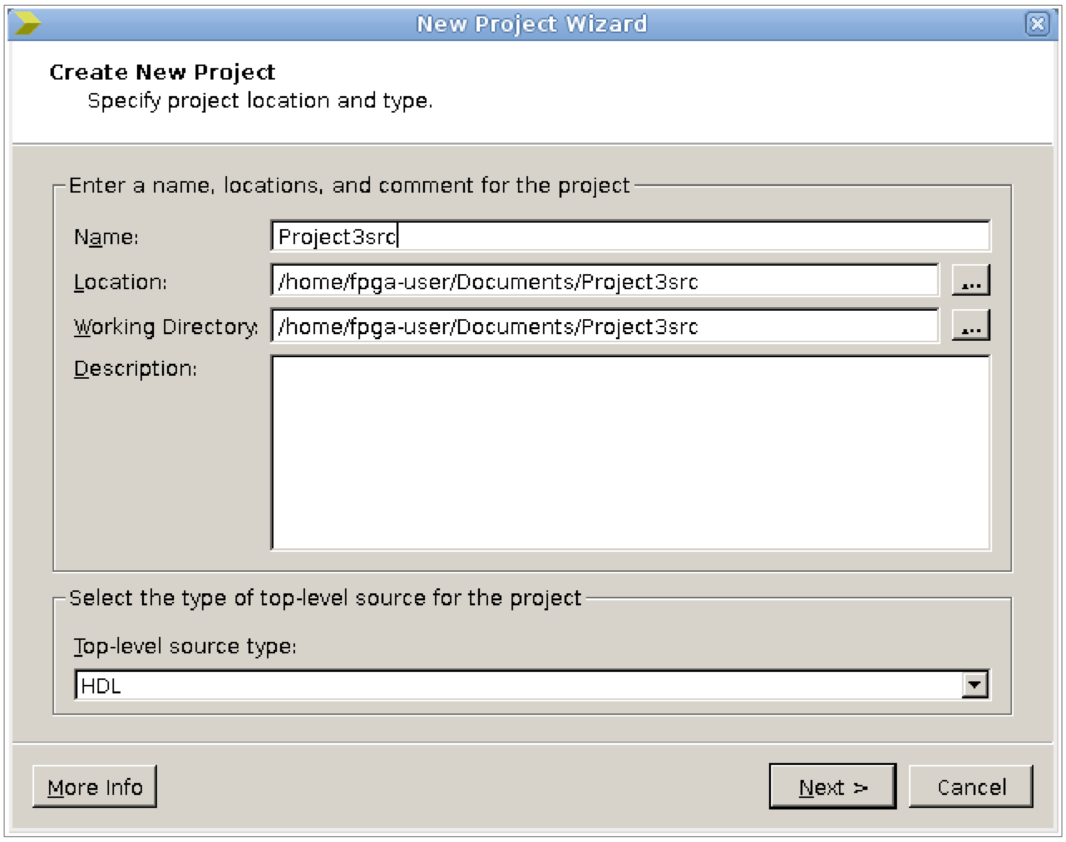
The following pages will demonstrate how to implement and verify in-circuit a 16-bit ALU using Xilinx ISE 13.3, VHDL and CORE Generator system as design entry, Xilinx Evaluation boards and ChipScope Pro tool for in-circuit verification.

1. Open Xilinx ISE :

Double click the ISE desktop icon. Alternatively open a new terminal and give the command “`./.bin/ise`” (without the quotes). For remote users, open a new terminal and give the following commands “`export DISPLAY=:1`” and “`./.remote/ise`” (without the quotes).

2. Create a new Xilinx ISE Project :

In the ISE Project Navigator window, select `File->New Project...`. In the New Project Wizard window, type `Project3src` in the Name field and `/home/fpga-user/Documents/Project3src` in the Location field and click Next (see **Figure 2**).


New Project Wizard

Create New Project
Specify project location and type.

Enter a name, locations, and comment for the project

Name:	Project3src
Location:	/home/fpga-user/Documents/Project3src
Working Directory:	/home/fpga-user/Documents/Project3src
Description:	

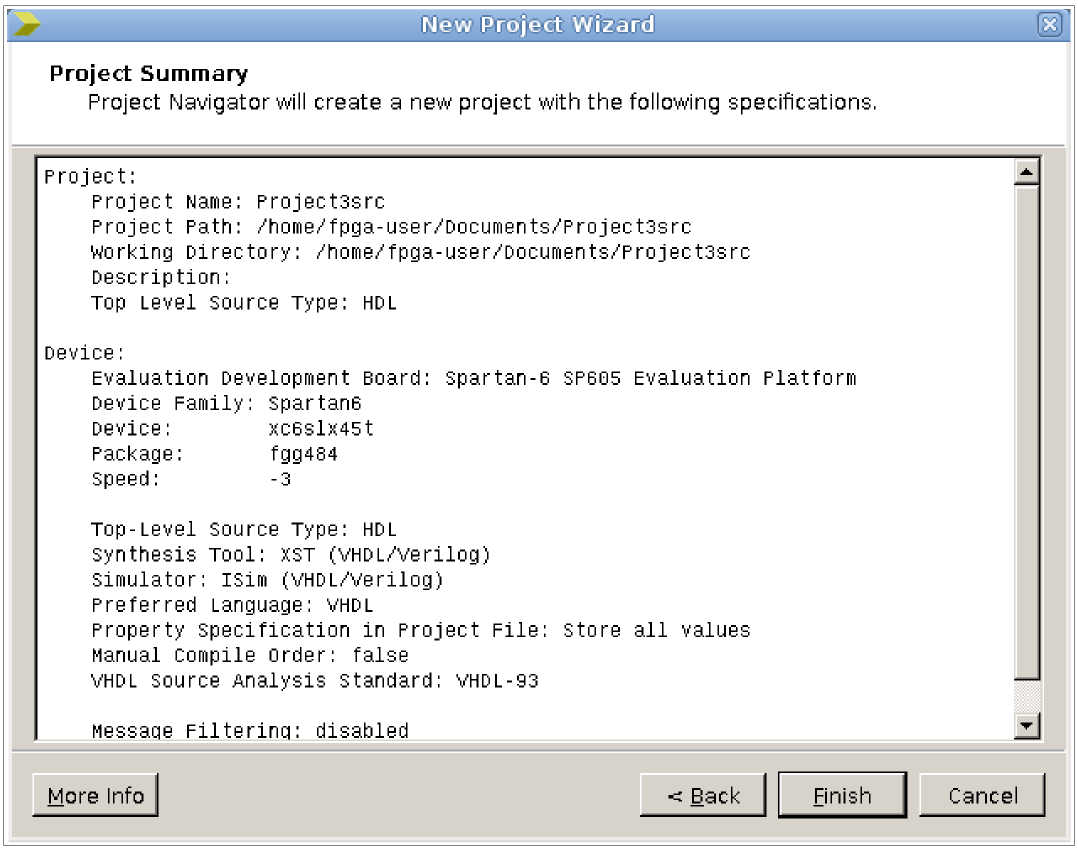
Select the type of top-level source for the project

Top-level source type: HDL

Figure 2

In the next New Project Wizard window, select the target Xilinx FPGA Evaluation board (e.g. the Spartan-6 SP605 Evaluation Platform) in the Evaluation Development Board field, select

VHDL in the Preferred Language field and click Next. The last New Project Wizard window must be similar to the one shown in **Figure 3**. Click Finish to create the project `Project3src` or navigate to the previous windows using Back to make corrections.

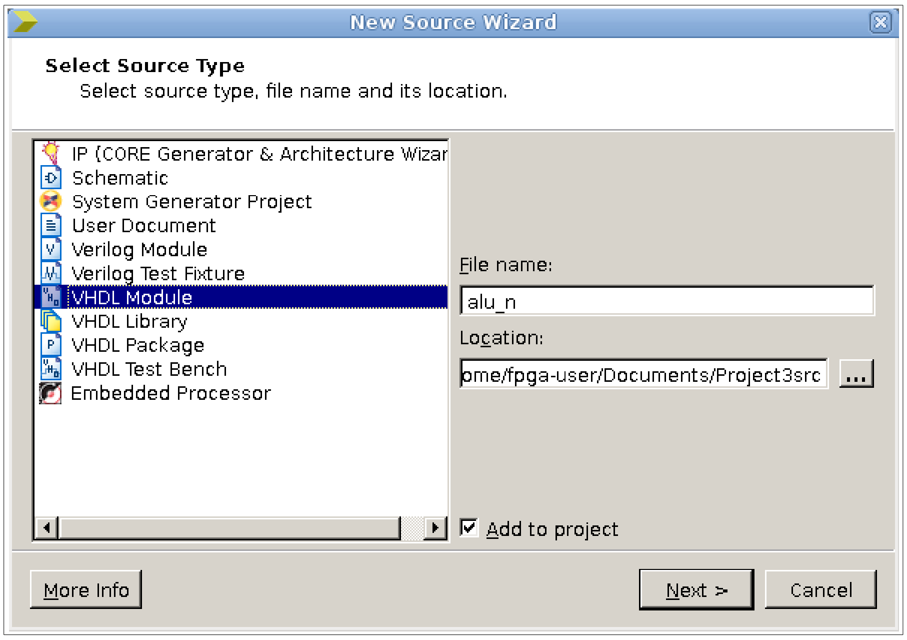

New Project Wizard
Project Summary
Project Navigator will create a new project with the following specifications.
More Info
< Back
Finish
Cancel

```
Project:
    Project Name: Project3src
    Project Path: /home/fpga-user/Documents/Project3src
    Working Directory: /home/fpga-user/Documents/Project3src
    Description:
    Top Level Source Type: HDL

Device:
    Evaluation Development Board: Spartan-6 SP605 Evaluation Platform
    Device Family: Spartan6
    Device:        xc6slx45t
    Package:       fgg484
    Speed:         -3

    Top-Level Source Type: HDL
    Synthesis Tool: XST (VHDL/Verilog)
    Simulator: ISim (VHDL/Verilog)
    Preferred Language: VHDL
    Property Specification in Project File: Store all values
    Manual Compile Order: false
    VHDL Source Analysis Standard: VHDL-93

    Message Filtering: disabled
```

Figure 3

3. Add New Source in the Project3src project :

In the ISE Project Navigator window, select `Project->New Source...`. In the New Source Wizard window type `alu_n` in the File Name field, select VHDL Module and click Next (see **Figure 4**). The next New Source Wizard window must be completed as shown in **Figure 5**.


New Source Wizard

Select Source Type
Select source type, file name and its location.

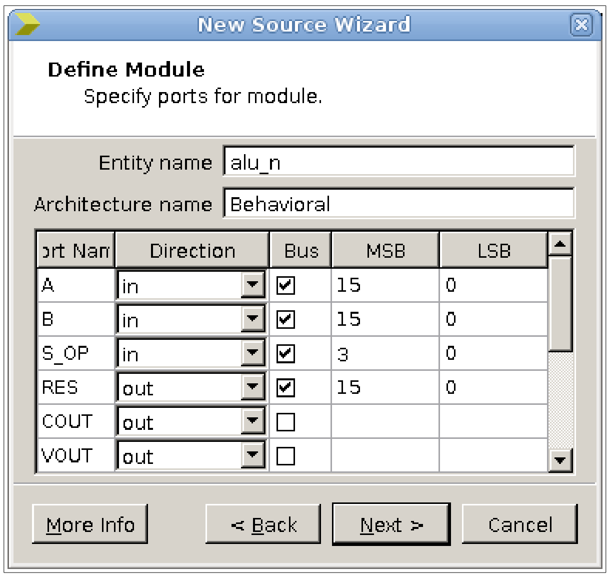
- IP (CORE Generator & Architecture Wizar
- Schematic
- System Generator Project
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

File name: alu_n

Location: ome/fpga-user/Documents/Project3src

☑ Add to project

Figure 4


New Source Wizard

Define Module
Specify ports for module.

Entity name: alu_n

Architecture name: Behavioral

ort Nam	Direction	Bus	MSB	LSB
A	in	☑	15	0
B	in	☑	15	0
S_OP	in	☑	3	0
RES	out	☑	15	0
COUT	out	☐		
VOUT	out	☐		

Figure 5

Clicking Next and Finish, the file `alu_n.vhd` is created.

4. Define the behavior of the `alu_n` VHDL module :

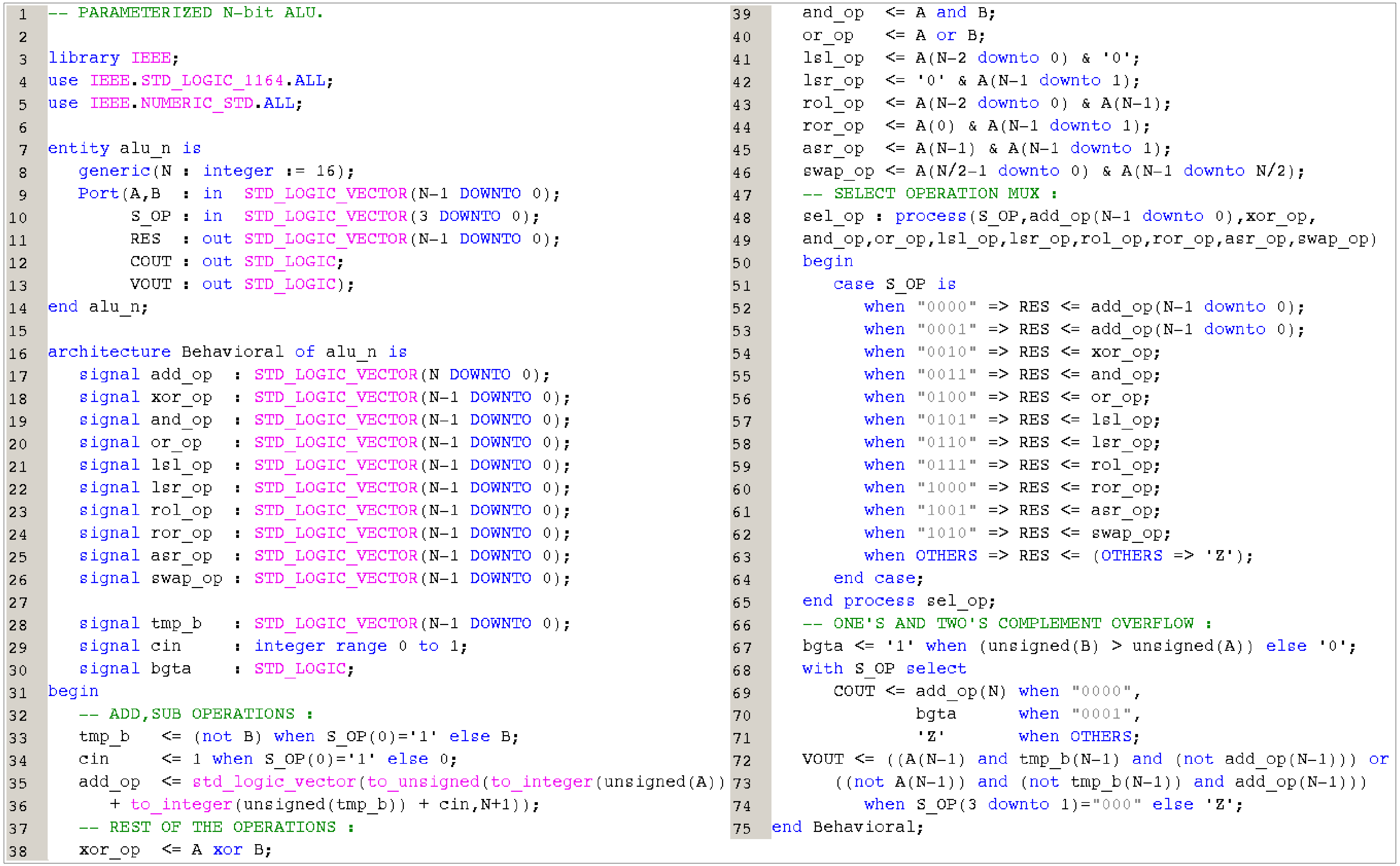
Remove the green highlighted lines (the comments) and edit the `alu_n.vhd` file to look like the one shown in **Figure 6**.

In **Figure 6**, line `5` states that the `NUMERIC_STD` package will be used. In this package the "+" operator is defined which is used to add signal values in the architecture body – to implement the `A+B` and the `A-B` operations.

In line 8 the generic statement is used to make the ALU description more general. The default value of the N parameter is 16, but this can easily change. The operators that are used to describe the parameterized length of a signal don't require any package statement (e.g. the "–" in the (N–1 downto 0) expression in line 9).

Lines 17-30 (architecture declarative region) define the signals used in the architecture body for internal circuit connections. The cin signal is of integer type. An integer signal represents a binary number. Using the range keyword the number of signal's bits can be declared (line 29). The cin signal's range is from 0 to 1. Integers in this range can be represented by one bit.

Lines 33-36 describe the A+B and the A–B operations. For example when S_OP(0)='1', tmp_b signal's value is equal to not B and cin signal's value is equal to 1. In lines 35-36, after the required conversions, the values of signals A, tmp_b and cin are added. In this case A+tmp_b+cin = A+(not B)+1 = A+(-B) = A-B. When S_OP(0)='0', tmp_b signal's value is equal to B and cin signal's value is equal to 0. In this case A+tmp_b+cin = A+B+0 = A+B.



```vhdl
-- PARAMETERIZED N-bit ALU.

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity alu_n is
   generic(N : integer := 16);
   Port(A,B  : in  STD_LOGIC_VECTOR(N-1 DOWNTO 0);
        S_OP : in  STD_LOGIC_VECTOR(3 DOWNTO 0);
        RES  : out STD_LOGIC_VECTOR(N-1 DOWNTO 0);
        COUT : out STD_LOGIC;
        VOUT : out STD_LOGIC);
end alu_n;

architecture Behavioral of alu_n is
   signal add_op  : STD_LOGIC_VECTOR(N DOWNTO 0);
   signal xor_op  : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
   signal and_op  : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
   signal or_op   : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
   signal lsl_op  : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
   signal lsr_op  : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
   signal rol_op  : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
   signal ror_op  : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
   signal asr_op  : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
   signal swap_op : STD_LOGIC_VECTOR(N-1 DOWNTO 0);

   signal tmp_b   : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
   signal cin     : integer range 0 to 1;
   signal bgta    : STD_LOGIC;
begin
   -- ADD,SUB OPERATIONS :
   tmp_b   <= (not B) when S_OP(0)='1' else B;
   cin     <= 1 when S_OP(0)='1' else 0;
   add_op  <= std_logic_vector(to_unsigned(to_integer(unsigned(A))
      + to_integer(unsigned(tmp_b)) + cin,N+1));
   -- REST OF THE OPERATIONS :
   xor_op  <= A xor B;
   and_op  <= A and B;
   or_op   <= A or B;
   lsl_op  <= A(N-2 downto 0) & '0';
   lsr_op  <= '0' & A(N-1 downto 1);
   rol_op  <= A(N-2 downto 0) & A(N-1);
   ror_op  <= A(0) & A(N-1 downto 1);
   asr_op  <= A(N-1) & A(N-1 downto 1);
   swap_op <= A(N/2-1 downto 0) & A(N-1 downto N/2);
   -- SELECT OPERATION MUX :
   sel_op : process(S_OP,add_op(N-1 downto 0),xor_op,
   and_op,or_op,lsl_op,lsr_op,rol_op,ror_op,asr_op,swap_op)
   begin
      case S_OP is
         when "0000" => RES <= add_op(N-1 downto 0);
         when "0001" => RES <= add_op(N-1 downto 0);
         when "0010" => RES <= xor_op;
         when "0011" => RES <= and_op;
         when "0100" => RES <= or_op;
         when "0101" => RES <= lsl_op;
         when "0110" => RES <= lsr_op;
         when "0111" => RES <= rol_op;
         when "1000" => RES <= ror_op;
         when "1001" => RES <= asr_op;
         when "1010" => RES <= swap_op;
         when OTHERS => RES <= (OTHERS => 'Z');
      end case;
   end process sel_op;
   -- ONE'S AND TWO'S COMPLEMENT OVERFLOW :
   bgta <= '1' when (unsigned(B) > unsigned(A)) else '0';
   with S_OP select
      COUT <= add_op(N) when "0000",
              bgta      when "0001",
              'Z'       when OTHERS;
   VOUT <= ((A(N-1) and tmp_b(N-1) and (not add_op(N-1))) or
      ((not A(N-1)) and (not tmp_b(N-1)) and add_op(N-1)))
         when S_OP(3 downto 1)="000" else 'Z';
end Behavioral;
```

Figure 6

In lines `35-36` the “+” operator is used to add the values of `A`, `tmp_b` and `cin` signals. The `A` and `tmp_b` signal values are first converted to the `unsigned` type and then to the `integer` type. The type conversions are necessary because the “+” operator is not defined for signals of the `STD_LOGIC_VECTOR` type in the `NUMERIC_STD` package. A `STD_LOGIC_VECTOR` signal's value must first be converted into `unsigned` and then into `integer`, because there is no direct way to convert a `STD_LOGIC_VECTOR` signal's value into `integer`. **Figure 7** shows the steps required to convert a `STD_LOGIC_VECTOR` signal value into `integer` and vice versa [1]. In the `to_unsigned()` function (**Figure 7**, INTEGER → UNSIGNED conversion) the `'LENGTH` is called an attribute. The `B'LENGTH` expression is used to specify the length of signal `B`.

Signal definitions :

```
signal A : STD_LOGIC_VECTOR(N-1 DOWNTO 0);
signal B : UNSIGNED(N-1 DOWNTO 0);
signal C : INTEGER;
```

STD_LOGIC_VECTOR → UNSIGNED conversion :
Method : type casting.

```
B <= UNSIGNED(A);
```

UNSIGNED → INTEGER conversion :
Method : use of the “to_integer()” function of the `NUMERIC_STD` package.

```
C <= to_integer(B);
```

INTEGER → UNSIGNED conversion :
Method : use of the “to_unsigned()” function of the `NUMERIC_STD` package.

```
B <= to_unsigned(C,N);
```

OR

```
B <= to_unsigned(C,B'LENGTH);
```

UNSIGNED → STD_LOGIC_VECTOR conversion :
Method : type casting.

```
A <= STD_LOGIC_VECTOR(B);
```

Figure 7

In lines `41-46` (**Figure 6**) the concatenate operator, `&`, is used to assign the value of a group of signals to a signal. For example in line `41` the most significant bit (`lsl_op(N-1)`) of the `lsl_op` signal is assigned with the value of `A(N-2)` and the least significant bit (`lsl_op(0)`) is assigned with the value `'0'`, because the concatenate operator groups the signal values in this order.

In lines `48-65` (**Figure 6**) a process statement is used. A process statement is a concurrent statement and can be used like any other concurrent statement in the architecture body [2]. All concurrent statements can be optionally tagged (see line `48`). This statement describes a multiplexer. The select signal of the multiplexer is the `S_OP` signal and its other inputs are the signals that hold the result of each operation (defined in lines `35-46`). Multiplexer's output is the output signal `RES` of the `alu_n` entity.

The expression `(unsigned(B) > unsigned(A))` compares the values of signals `A` and `B` as unsigned numbers. The `unsigned` conversion is used because the “>” operator is not defined for signals of the `STD_LOGIC_VECTOR` type in the `NUMERIC_STD` package and because the desirable comparison must be done considering `A` and `B` signals as unsigned numbers.

Instead of the `NUMERIC_STD` package, the `STD_LOGIC_ARITH` and the `STD_LOGIC_UNSIGNED` packages could be used. The `STD_LOGIC_ARITH` package defines two signal types, SIGNED and UNSIGNED. These types are identical to the `STD_LOGIC_VECTOR` type because they represent an array of `STD_LOGIC` signals. The purpose of the SIGNED and UNSIGNED types is to allow the designer to indicate in the VHDL code what kind of number representation is being used. The SIGNED type is used in code for circuits that deal with signed (2's complement) numbers, and the UNSIGNED type is used in code that deals with unsigned numbers. The `STD_LOGIC_UNSIGNED` package specifies that `STD_LOGIC_VECTOR` signals should be treated like UNSIGNED signals. Similarly, a `STD_LOGIC_SIGNED` package specifies that `STD_LOGIC_VECTOR` signals should be treated like SIGNED signals. The `STD_LOGIC_ARITH` package, and hence the `STD_LOGIC_UNSIGNED` and `STD_LOGIC_SIGNED` packages, are not actually a part of the VHDL standards. They are provided by Synopsys Inc., which is a vendor of CAD software. However, these packages are included with most CAD systems that support VHDL, and they are widely used in practice [2]. **Figure 8** shows the changes that must be done in the code of **Figure 6** when the `STD_LOGIC_ARITH` and the `STD_LOGIC_UNSIGNED` packages are used.

Replace	With

use IEEE.NUMERIC_STD.ALL;	use IEEE.STD_LOGIC_ARITH.ALL;
	use IEEE.STD_LOGIC_UNSIGNED.ALL;

add_op <= std_logic_vector(to_unsigned(to_integer(unsigned(A)) + to_integer(unsigned(tmp_b)) + cin,N+1));	add_op <= conv_std_logic_vector(conv_integer(A) + conv_integer(tmp_b) + cin,N+1);
bgta <= '1' when (unsigned(B) > unsigned(A)) else '0';	bgta <= '1' when (B > A) else '0';

Figure 8

5. Behavioral simulation of the `alu_n` module :

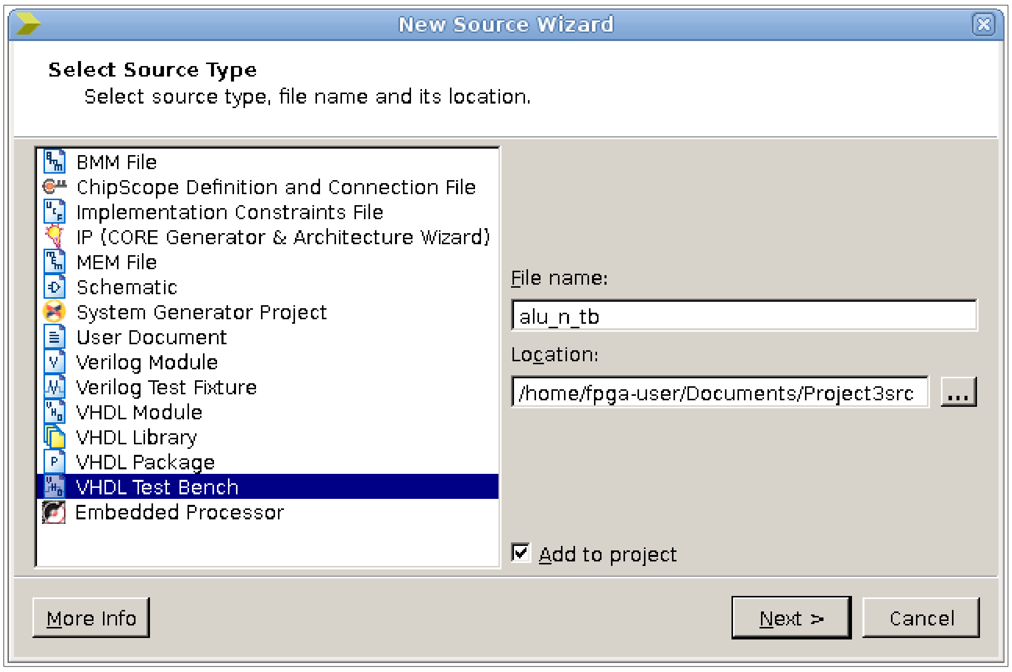
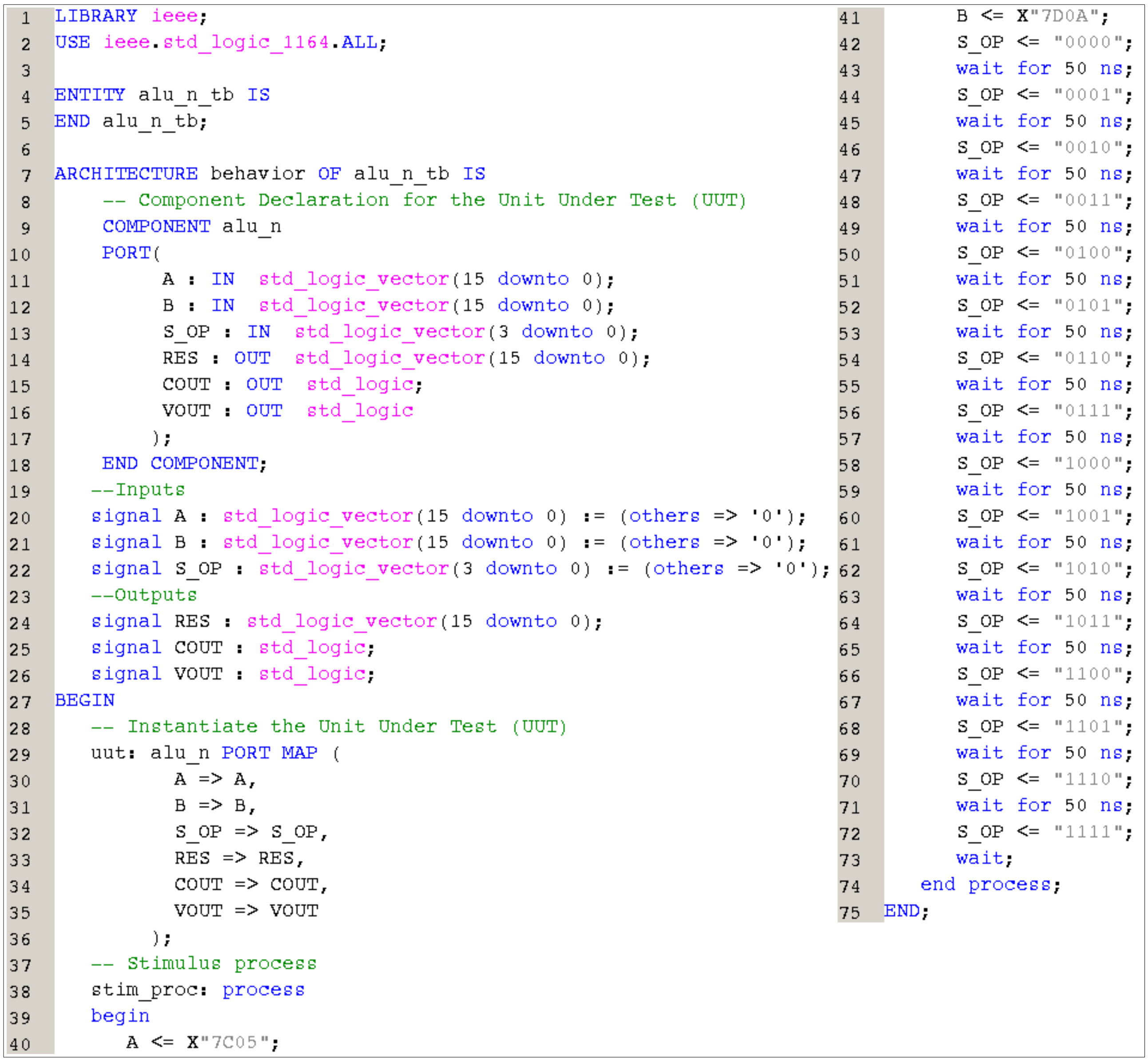


Figure 9

In the ISE Project Navigator window select `Project->New Source...`. In the New Source Wizard window type `alu_n_tb` in the File Name field, select VHDL Test Bench and click Next (see **Figure 9**). In the next New Source Wizard window associate the `alu_n_tb` file with the `alu_n` module, click Next and then click Finish to create the `alu_n_tb.vhd` testbench file. Modify the `alu_n_tb.vhd` testbench file to look like the one in **Figure 10**.

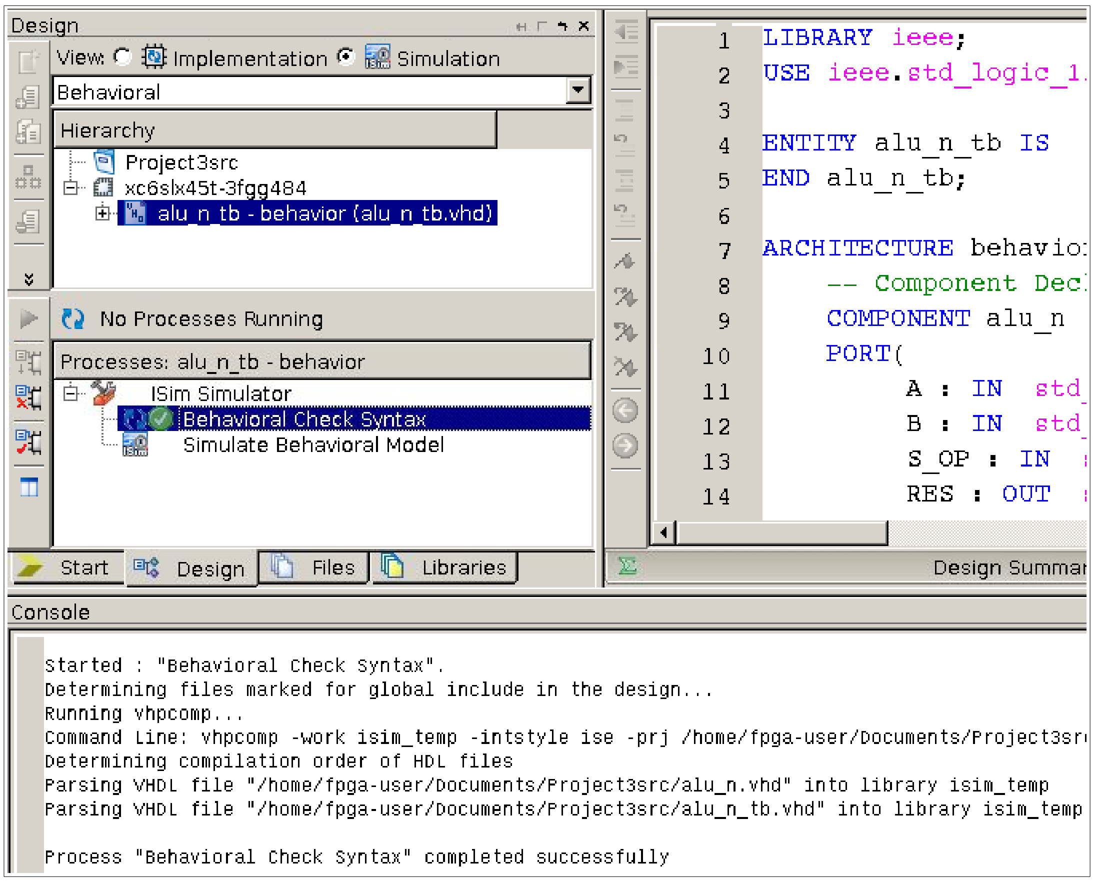


```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  ENTITY alu_n_tb IS
5  END alu_n_tb;
6
7  ARCHITECTURE behavior OF alu_n_tb IS
8      -- Component Declaration for the Unit Under Test (UUT)
9      COMPONENT alu_n
10     PORT(
11          A : IN  std_logic_vector(15 downto 0);
12          B : IN  std_logic_vector(15 downto 0);
13          S_OP : IN  std_logic_vector(3 downto 0);
14          RES : OUT  std_logic_vector(15 downto 0);
15          COUT : OUT  std_logic;
16          VOUT : OUT  std_logic
17         );
18     END COMPONENT;
19    --Inputs
20    signal A : std_logic_vector(15 downto 0) := (others => '0');
21    signal B : std_logic_vector(15 downto 0) := (others => '0');
22    signal S_OP : std_logic_vector(3 downto 0) := (others => '0');
23    --Outputs
24    signal RES : std_logic_vector(15 downto 0);
25    signal COUT : std_logic;
26    signal VOUT : std_logic;
27 BEGIN
28    -- Instantiate the Unit Under Test (UUT)
29    uut: alu_n PORT MAP (
30           A => A,
31           B => B,
32           S_OP => S_OP,
33           RES => RES,
34           COUT => COUT,
35           VOUT => VOUT
36         );
37    -- Stimulus process
38    stim_proc: process
39    begin
40       A <= X"7C05";
41       B <= X"7D0A";
42       S_OP <= "0000";
43       wait for 50 ns;
44       S_OP <= "0001";
45       wait for 50 ns;
46       S_OP <= "0010";
47       wait for 50 ns;
48       S_OP <= "0011";
49       wait for 50 ns;
50       S_OP <= "0100";
51       wait for 50 ns;
52       S_OP <= "0101";
53       wait for 50 ns;
54       S_OP <= "0110";
55       wait for 50 ns;
56       S_OP <= "0111";
57       wait for 50 ns;
58       S_OP <= "1000";
59       wait for 50 ns;
60       S_OP <= "1001";
61       wait for 50 ns;
62       S_OP <= "1010";
63       wait for 50 ns;
64       S_OP <= "1011";
65       wait for 50 ns;
66       S_OP <= "1100";
67       wait for 50 ns;
68       S_OP <= "1101";
69       wait for 50 ns;
70       S_OP <= "1110";
71       wait for 50 ns;
72       S_OP <= "1111";
73       wait;
74    end process;
75 END;
```

Figure 10

Statement in lines `40-41` (**Figure 10**) assign values to `A` and `B` signals using the hexadecimal value representation. Signals `A` and `B` carry these values until the end of the simulation.

In the ISE Project Navigator window select Simulation, select `alu_n_tb` testbench file, expand ISim Simulator and double-click Behavioral Check Syntax to check `alu_n_tb` testbench's syntax (see **Figure 11**).



```
LIBRARY ieee;
USE ieee.std_logic_1

ENTITY alu_n_tb IS
END alu_n_tb;

ARCHITECTURE behavio
    -- Component Dec
    COMPONENT alu_n
    PORT(
         A : IN  std_
         B : IN  std_
         S_OP : IN
         RES : OUT
```

```
Started : "Behavioral Check Syntax".
Determining files marked for global include in the design...
Running vhpcomp...
Command Line: vhpcomp -work isim_temp -intstyle ise -prj /home/fpga-user/Documents/Project3sr
Determining compilation order of HDL files
Parsing VHDL file "/home/fpga-user/Documents/Project3src/alu_n.vhd" into library isim_temp
Parsing VHDL file "/home/fpga-user/Documents/Project3src/alu_n_tb.vhd" into library isim_temp

Process "Behavioral Check Syntax" completed successfully
```

Figure 11

Right-click Simulate Behavioral Model and select Process Properties. Change Simulation Run Time to 800 ns and click OK. Now double-click Simulate Behavioral Model to run the simulation as specified by the `alu_n_tb` testbench file and the ISim Simulation Run Time parameter. Select `View->Zoom->To Full View` in ISim simulator window. Right-click on signals `A`, `B` and `RES` and select Radix to change the representation radix (first to Unsigned Decimal and then to Signed Decimal). Click on the waveform to verify the right execution of addition and subtraction (see **Figure 12**).

In **Figure 12**, a 2's complement overflow is generated during addition (`VOUT='1'`). This is correct because `A` and `B` represent positive numbers (in 2's complement form) and the result of the addition is negative in 2's complement form (bottom waveform in **Figure 12**). A carry is generated during subtraction (`COUT='1'`). This is also correct because `B` is larger than `A` and that means that a borrow is used to perform the subtraction (see the upper waveform in **Figure 12**). The borrow is equal to 2^{16} and the result of the subtraction for the upper waveform in **Figure 12** can be interpreted as follows : RES=(A+ 2^{16})-B. The subtraction result in 2's complement form is correct and a 2's complement overflow is not generated.

Right-click on signals `A`, `B`, and `RES` and select `Radix->Binary` to check the results of the remaining operations.

Close the ISim simulator window.

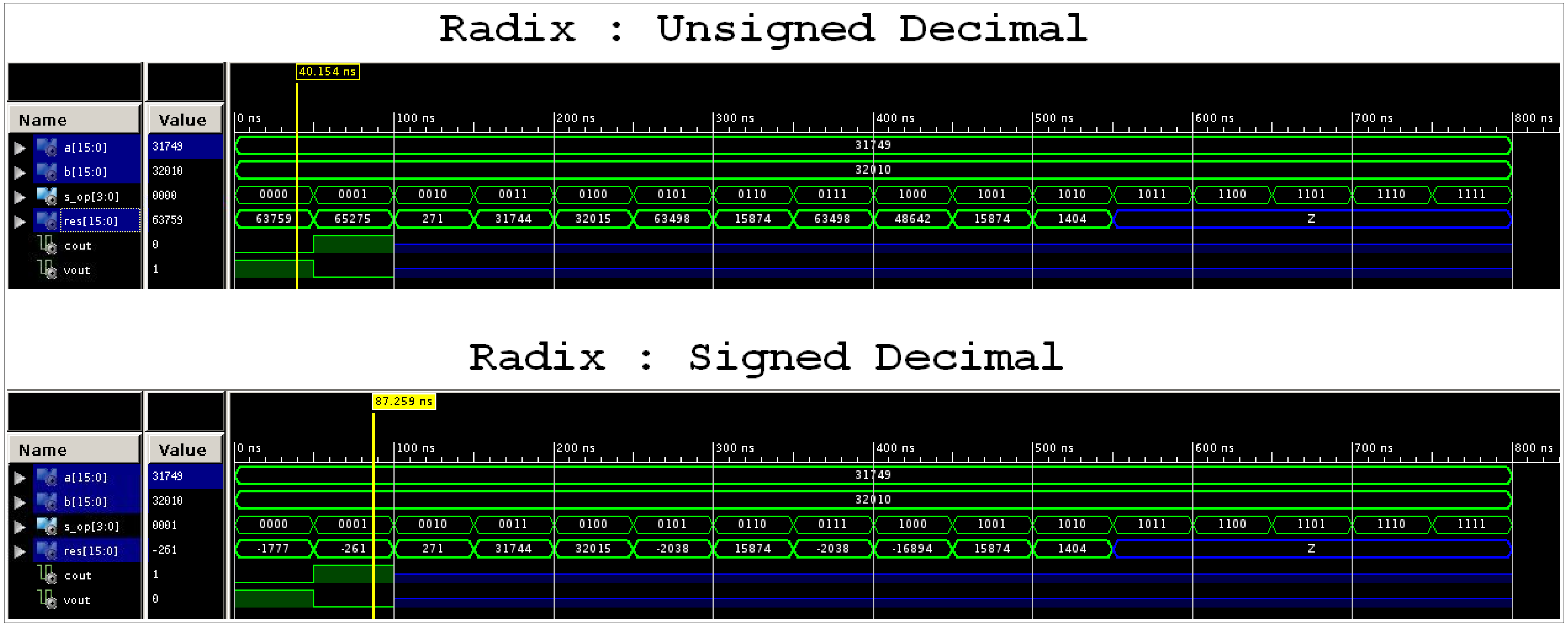
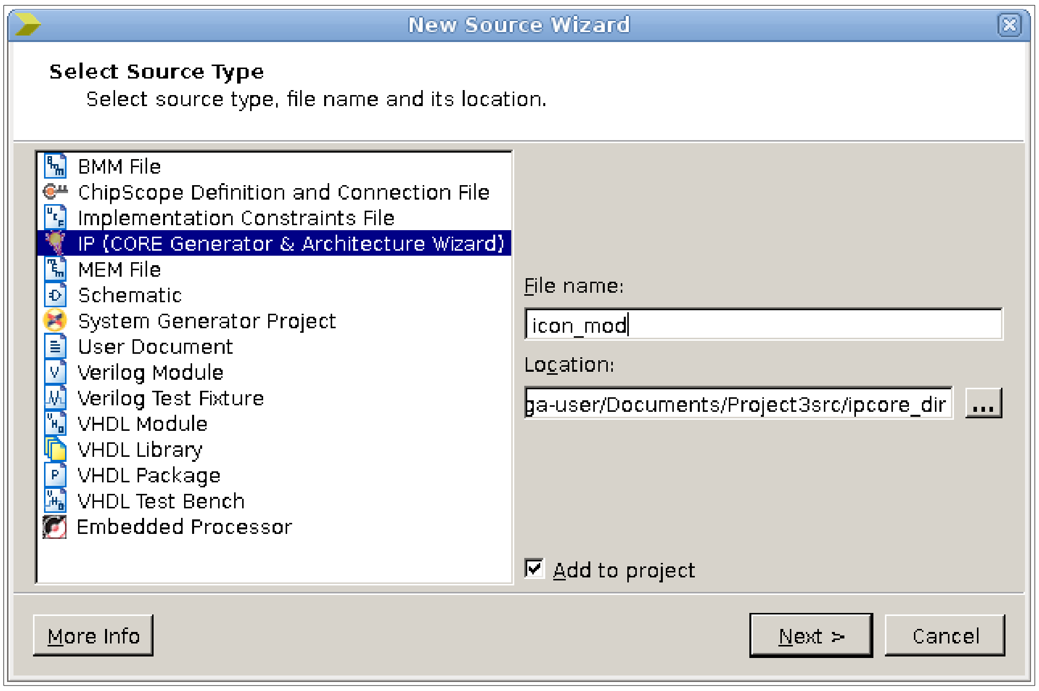

Radix : Unsigned Decimal
40.154 ns
Name
Value
a[15:0] 31749
b[15:0] 32010
s_op[3:0] 0000
res[15:0] 63759
cout 0
vout 1
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
31749
32010
0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
63759 65275 271 31744 32015 63498 15874 63498 48642 15874 1404 Z
Radix : Signed Decimal
87.259 ns
Name
Value
a[15:0] 31749
b[15:0] 32010
s_op[3:0] 0001
res[15:0] -261
cout 1
vout 0
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
31749
32010
0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
-1777 -261 271 31744 32015 -2038 15874 -2038 -16894 15874 1404 Z

Figure 12

6. Generate ICON and VIO cores :

In the ISE Project Navigator window, select `Project->New Source...`. In the New Source Wizard window, type `icon_mod` in the File Name field, select IP (CORE Generator & Architecture Wizard) and click Next (see **Figure 13**).


New Source Wizard

Select Source Type
Select source type, file name and its location.

- BMM File
- ChipScope Definition and Connection File
- Implementation Constraints File
- IP (CORE Generator & Architecture Wizard)
- MEM File
- Schematic
- System Generator Project
- User Document
- Verilog Module
- Verilog Test Fixture
- VHDL Module
- VHDL Library
- VHDL Package
- VHDL Test Bench
- Embedded Processor

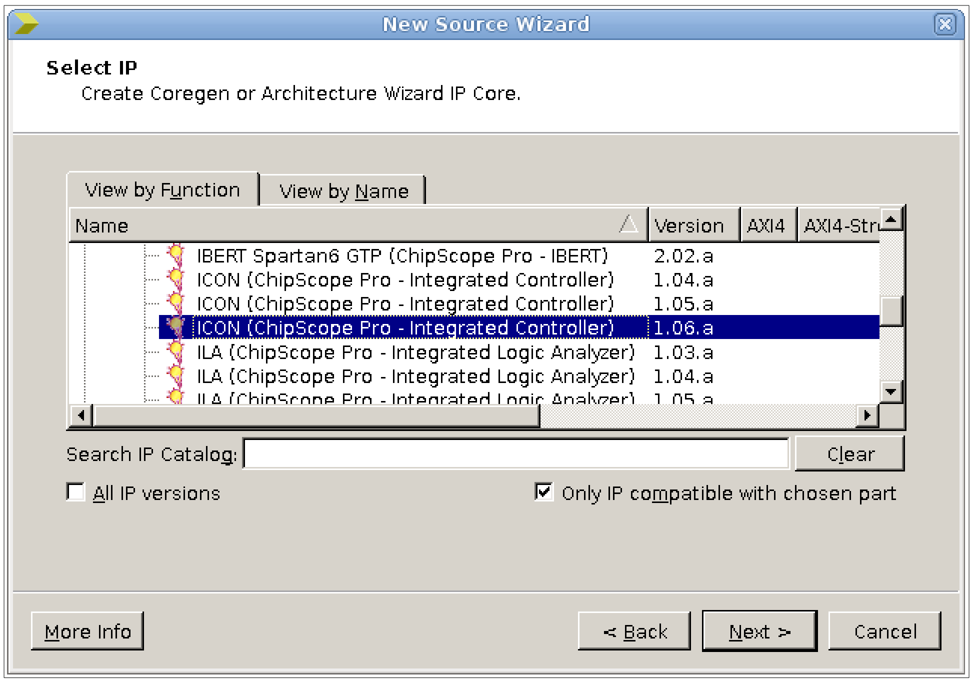
File name: icon_mod

Location: ga-user/Documents/Project3src/ipcore_dir

☑ Add to project

Figure 13

In the New Source Wizard window (Select IP), check Only IP compatible with chosen part, expand Debug & Verification, expand ChipScope Pro, select ICON (ChipScope Pro – Integrated Controller) version 1.06.a, click Next and Finish (see **Figure 14**).


New Source Wizard

Select IP
Create Coregen or Architecture Wizard IP Core.

Name	Version	AXI4	AXI4-Str
IBERT Spartan6 GTP (ChipScope Pro - IBERT)	2.02.a		
ICON (ChipScope Pro - Integrated Controller)	1.04.a		
ICON (ChipScope Pro - Integrated Controller)	1.05.a		
ICON (ChipScope Pro - Integrated Controller)	1.06.a		
ILA (ChipScope Pro - Integrated Logic Analyzer)	1.03.a		
ILA (ChipScope Pro - Integrated Logic Analyzer)	1.04.a		
ILA (ChipScope Pro - Integrated Logic Analyzer)	1.05.a		

Search IP Catalog:

☐ All IP versions
☑ Only IP compatible with chosen part

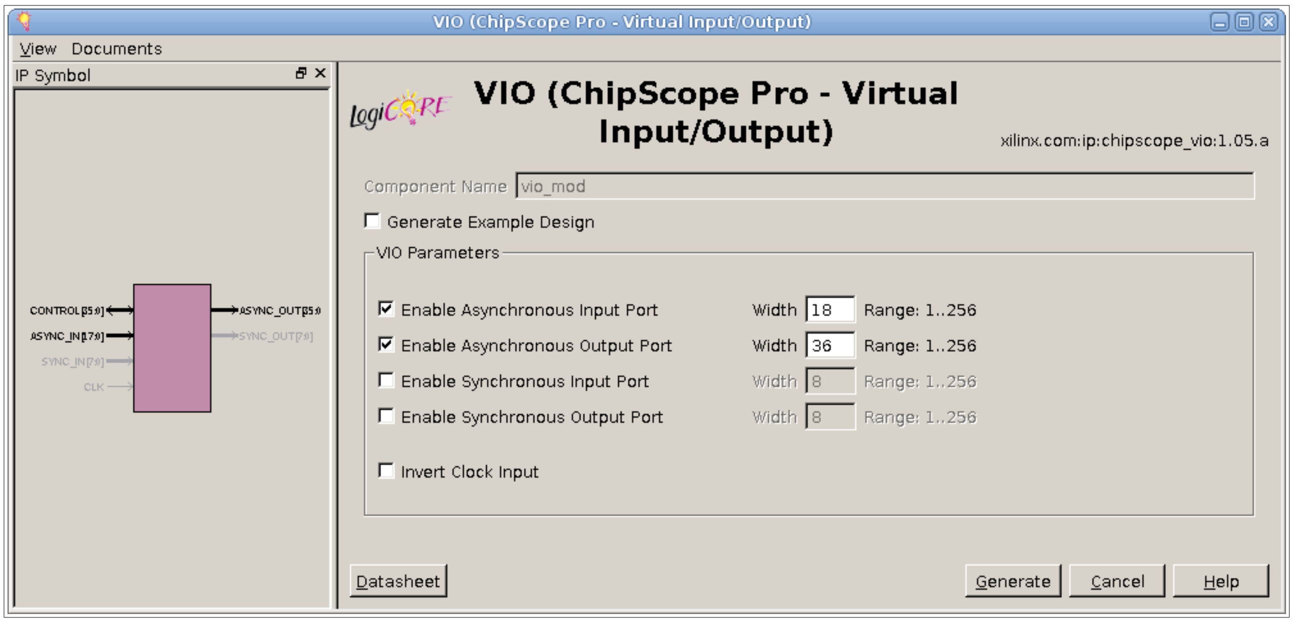
Figure 14

In the ICON (ChipScope Pro - Integrated Controller) window click Generate.

In the ISE Project Navigator window, select `Project->New Source...`. In the New Source Wizard window type `vio_mod` in the File Name field, select IP (CORE Generator & Architecture Wizard) and click Next.

In the New Source Wizard window (Select IP), check Only IP compatible with chosen part, expand Debug & Verification, expand ChipScope Pro, select VIO (ChipScope Pro – Virtual Input/Output) version 1.05.a, click Next and Finish.

In the VIO (ChipScope Pro - Virtual Input/Output) window, check Enable Asynchronous Input Port (Width 18), check Enable Asynchronous Output Port (Width 36) and click Generate (see **Figure 15**). The VIO output port will feed the `alu_n` module input ports (16-bit `A` + 16-bit `B` + 4-bit `S_OP` = 36 bits) and the VIO input port will read the `alu_n` module output port (16-bit `RES` + 1-bit `COUT` +1-bit `VOUT` = 18 bits).



VIO (ChipScope Pro - Virtual Input/Output)

xilinx.com:ip:chipscope_vio:1.05.a

Component Name: vio_mod

☐ Generate Example Design

VIO Parameters

☑ Enable Asynchronous Input Port	Width 18	Range: 1..256
☑ Enable Asynchronous Output Port	Width 36	Range: 1..256
☐ Enable Synchronous Input Port	Width 8	Range: 1..256
☐ Enable Synchronous Output Port	Width 8	Range: 1..256

☐ Invert Clock Input

Figure 15

The `alu_n` module and the two cores (ICON and VIO) will be used as components in a single VHDL module (`alu_n_vio` module – top-module). The implementation of `alu_n_vio` module will be used in ChipScope Pro Analyzer for `alu_n` module verification [3], [4].

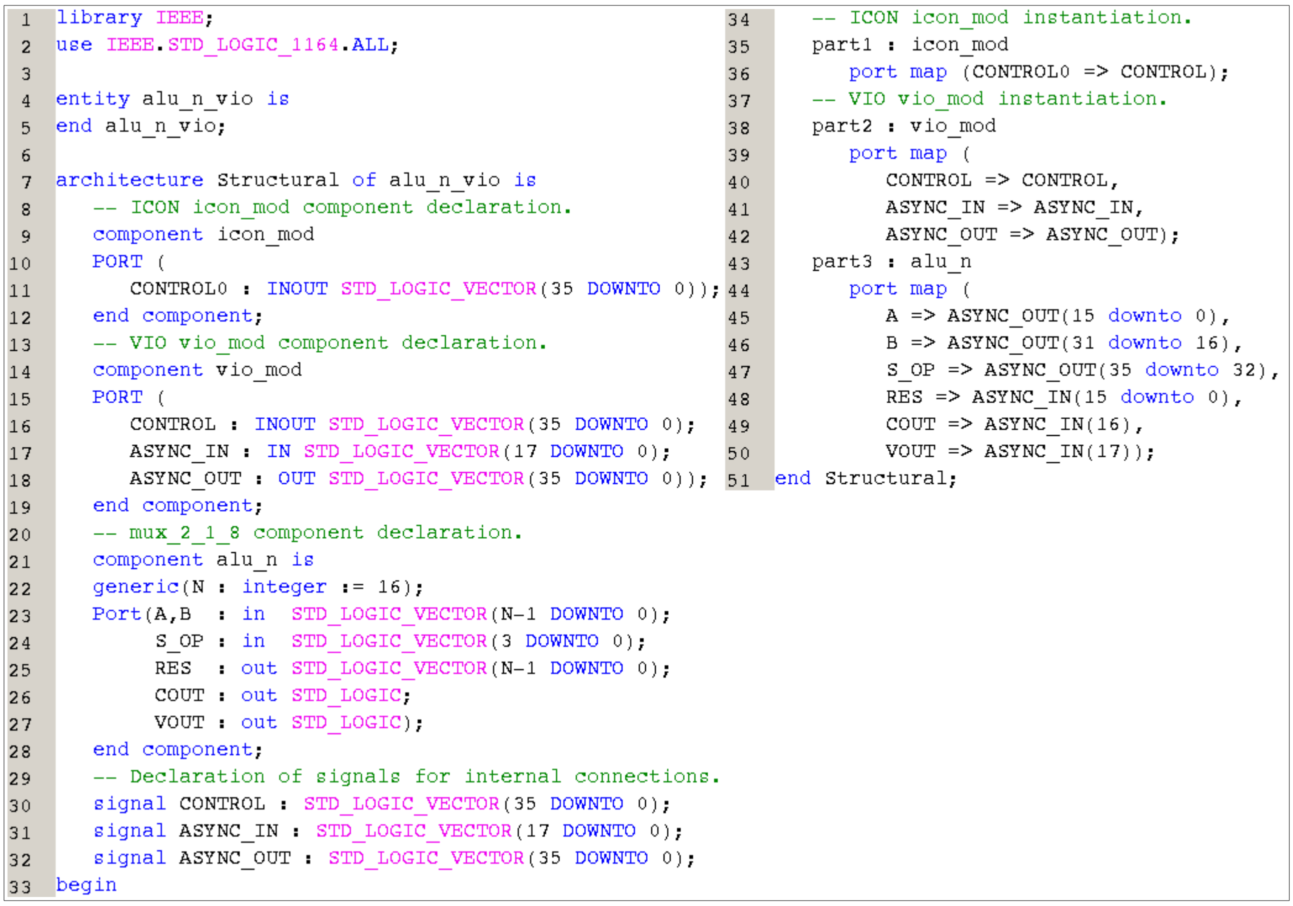
7. Create and Define the structure of the `alu_n_vio` VHDL module :

In the ISE Project Navigator window, select `Project->New Source...`. In the New Source Wizard window type `alu_n_vio` in the File Name field, select VHDL Module and click Next. In the next New Source Wizard window (Define Module) change the Architecture name field from `Behavioral` to `Structural`, click Next and Finish.

Module `alu_n_vio` doesn't have any ports (no ports were defined in module creation) because no FPGA general purpose I/O ports will be used for `alu_n` module verification. JTAG Boundary Scan (BSCAN) dedicated FPGA ports will be used (through the ICON core – the ICON core provides an interface between the JTAG Boundary Scan (BSCAN) component of the FPGA and the VIO core) [3].

In the ISE Project Navigator window, select Implementation, select the `icon_mod` core, expand CORE Generator and double-click View HDL Instantiation Template. Copy the component declaration of the `icon_mod` core in the `alu_n_vio` module's architecture declarative region. Copy the instance declaration of the `icon_mod` core in the `alu_n_vio` module's architecture body and name the instance `part1`. Follow the above-mentioned steps for `vio_mod` core instantiation in the `alu_n_vio` module and name the instance `part2`.

Copy the entity declaration of the `alu_n` module to the `alu_n_vio` module's architecture declarative region and edit it (in the `alu_n_vio` module) to form the component declaration of the `alu_n` module. **Figure 16** shows how to complete `alu_n_vio` module's description.



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity alu_n_vio is
end alu_n_vio;

architecture Structural of alu_n_vio is
   -- ICON icon_mod component declaration.
   component icon_mod
   PORT (
      CONTROL0 : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0));
   end component;
   -- VIO vio_mod component declaration.
   component vio_mod
   PORT (
      CONTROL : INOUT STD_LOGIC_VECTOR(35 DOWNTO 0);
      ASYNC_IN : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
      ASYNC_OUT : OUT STD_LOGIC_VECTOR(35 DOWNTO 0));
   end component;
   -- mux_2_1_8 component declaration.
   component alu_n is
   generic(N : integer := 16);
   Port(A,B  : in  STD_LOGIC_VECTOR(N-1 DOWNTO 0);
        S_OP : in  STD_LOGIC_VECTOR(3 DOWNTO 0);
        RES  : out STD_LOGIC_VECTOR(N-1 DOWNTO 0);
        COUT : out STD_LOGIC;
        VOUT : out STD_LOGIC);
   end component;
   -- Declaration of signals for internal connections.
   signal CONTROL : STD_LOGIC_VECTOR(35 DOWNTO 0);
   signal ASYNC_IN : STD_LOGIC_VECTOR(17 DOWNTO 0);
   signal ASYNC_OUT : STD_LOGIC_VECTOR(35 DOWNTO 0);
begin
   -- ICON icon_mod instantiation.
   part1 : icon_mod
      port map (CONTROL0 => CONTROL);
   -- VIO vio_mod instantiation.
   part2 : vio_mod
      port map (
         CONTROL => CONTROL,
         ASYNC_IN => ASYNC_IN,
         ASYNC_OUT => ASYNC_OUT);
   part3 : alu_n
      port map (
         A => ASYNC_OUT(15 downto 0),
         B => ASYNC_OUT(31 downto 16),
         S_OP => ASYNC_OUT(35 downto 32),
         RES => ASYNC_IN(15 downto 0),
         COUT => ASYNC_IN(16),
         VOUT => ASYNC_IN(17));
end Structural;
```

Figure 16

In the ISE Project Navigator window, select Implementation, right-click on `alu_n_vio` module and select `Select as Top Module`, select the `alu_n_vio` module, expand Synthesize – XST and double-click Check Syntax to ensure `alu_n_vio` module's correctness.

8. Synthesize and Implement the `alu_n_vio` design :

In the ISE Project Navigator window select Implementation, select the `alu_n_vio` module, expand Synthesize – XST and double-click View RTL Schematic to view the RTL schematic of the synthesized `alu_n_vio` module. In the Set RTL/Tech Viewer Startup Mode select Start with a Schematic of the top-level block and click OK. Double-click on the synthesized `alu_n_vio` module (black-box-like) graphic to view the schematic shown in **Figure 17**.

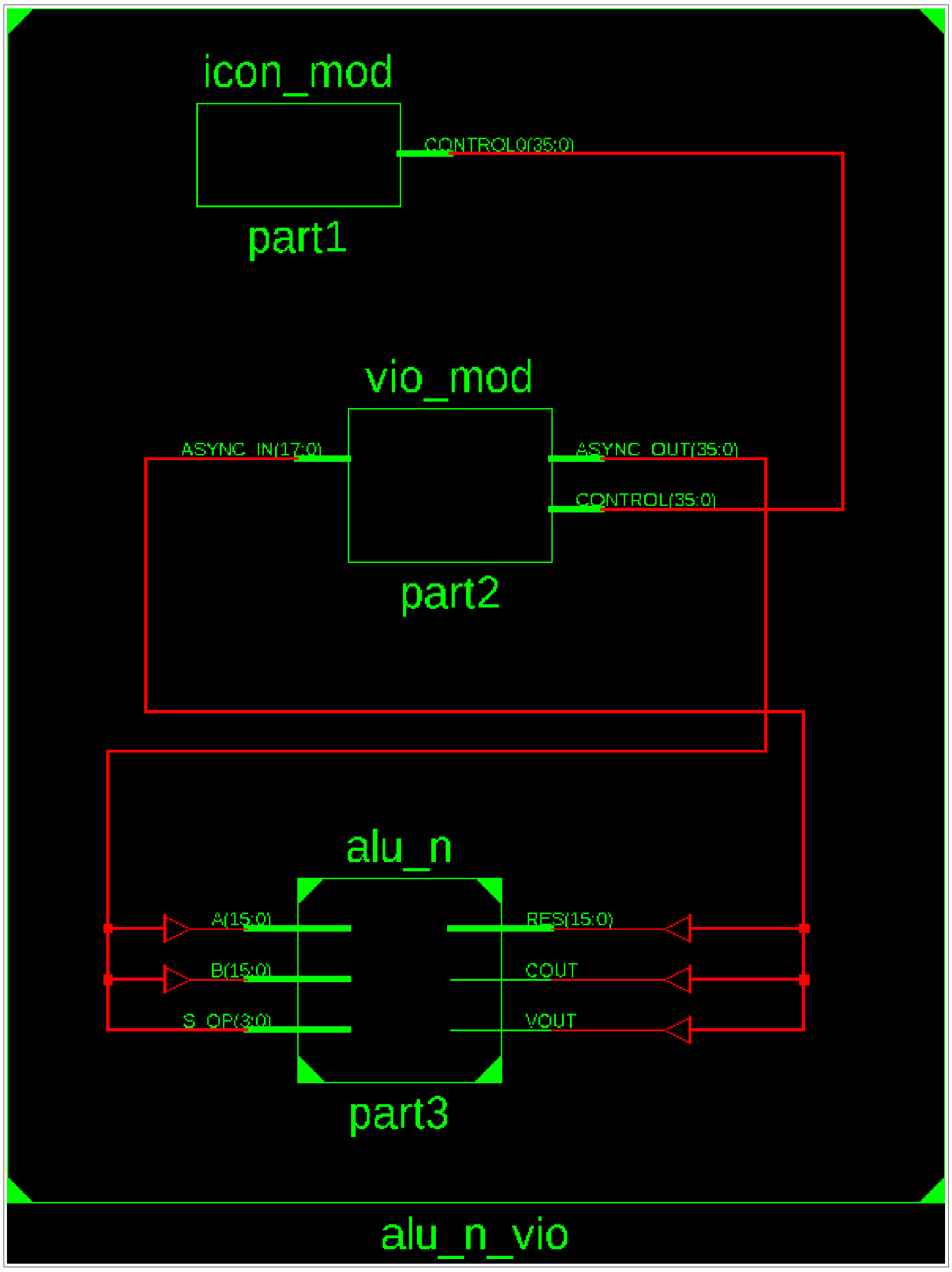
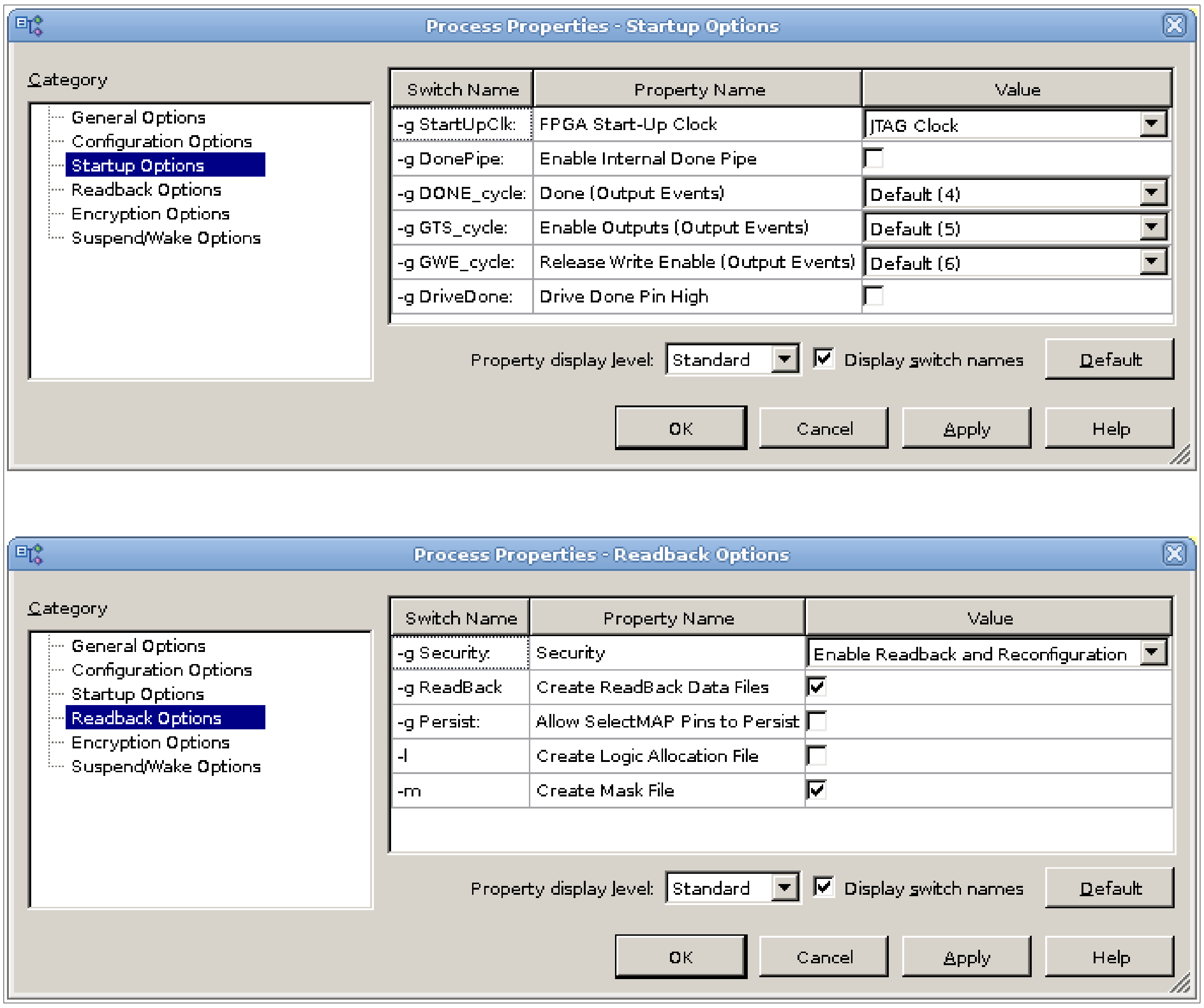


Figure 17

To implement the `alu_n_vio` module, in the ISE Project Navigator window select Implementation, select the `alu_n_vio` module and double-click Implement Design.

9. Generate Programming (Configuration) File :

In the ISE Project Navigator window select Implementation, select the `alu_n_vio` module, right-click Generate Programming File and select Process Properties. In the Process Properties – Startup Options window select Startup Options category and change FPGA Start-Up Clock property to JTAG Clock, then select Readback Options category and check Create Readback Data files and Create Mask File properties (see **Figure 18**). Click OK to close the Process Properties – Startup Options window. The above changes have been made in order to program the FPGA and verify FPGA programming using JTAG.



Switch Name	Property Name	Value
-g StartUpClk:	FPGA Start-Up Clock	JTAG Clock
-g DonePipe:	Enable Internal Done Pipe	☐
-g DONE_cycle:	Done (Output Events)	Default (4)
-g GTS_cycle:	Enable Outputs (Output Events)	Default (5)
-g GWE_cycle:	Release Write Enable (Output Events)	Default (6)
-g DriveDone:	Drive Done Pin High	☐

Switch Name	Property Name	Value
-g Security:	Security	Enable Readback and Reconfiguration
-g ReadBack	Create ReadBack Data Files	✓
-g Persist:	Allow SelectMAP Pins to Persist	☐
-l	Create Logic Allocation File	☐
-m	Create Mask File	✓

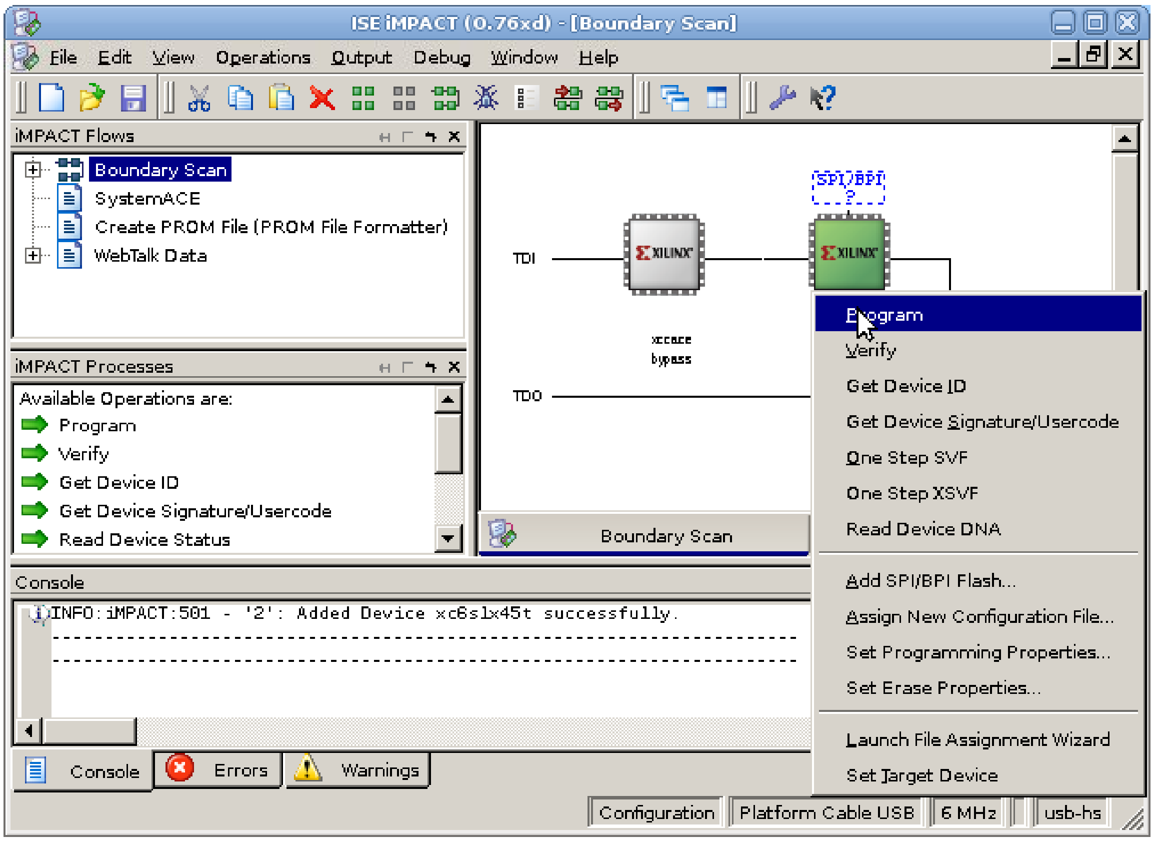
Figure 18

Double-click Generate Programming File to complete programming file generation.

10. Configure the FPGA :

In the ISE Project Navigator window select Implementation, select the `alu_n_vio` module, double-click Configure Target Device and click OK to open Impact. Impact software tool is used for FPGA configuration. In the ISE iMPACT window double-click Boundary Scan.

Right-click on Right click to Add Device or Initialize JTAG chain and select Initialize Chain. Click Yes to the Auto Assign Configuration Files Query Dialog, select Bypass for the xccace device, navigate to `/home/fpga-user/Documents/Project3src`, select the `alu_n_vio.bit` file and click Open for the xc6slx45t Spartan-6 FPGA. Click No to the Attach SPI or BPI PROM window. Select Device 2 in the Device Programming Properties window and check the Verify property. Right-click on the xc6slx45t Spartan-6 FPGA and select Program (see **Figure 19**).



```
INFO:iMPACT:501 - '2': Added Device xc6slx45t successfully.
----------------------------------------------------------------------
----------------------------------------------------------------------
```

Figure 19

After the Program Succeeded message close the ISE iMPACT window (click No to iMPACT-Save Project window).

11. Analyze design using ChipScope :

In the ISE Project Navigator window select Implementation, select the `alu_n_vio` module and double-click Analyze Design Using ChipScope. Click the Open Cable/Search JTAG Chain icon under the File menu in ChipScope Pro Analyzer window (see **Figure 20**) and click OK to the ChipScope Pro Analyzer (JTAG Chain Device Order) window.

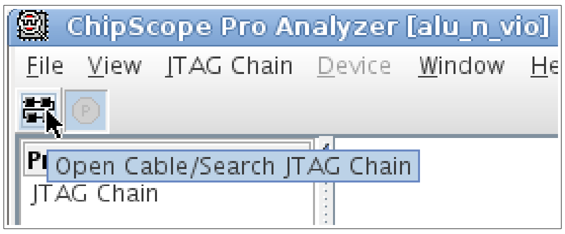

ChipScope Pro Analyzer [alu_n_vio]
File View JTAG Chain Device Window He
Open Cable/Search JTAG Chain
JTAG Chain

Figure 20

Double-click the VIO Console (see **Figure 21**) to open it.

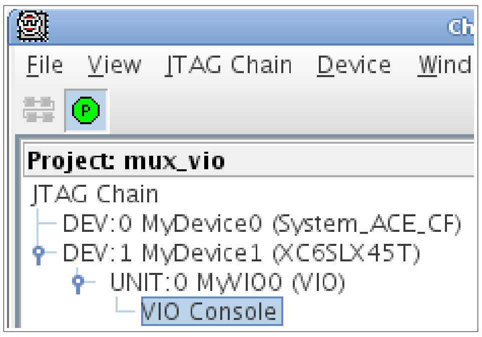
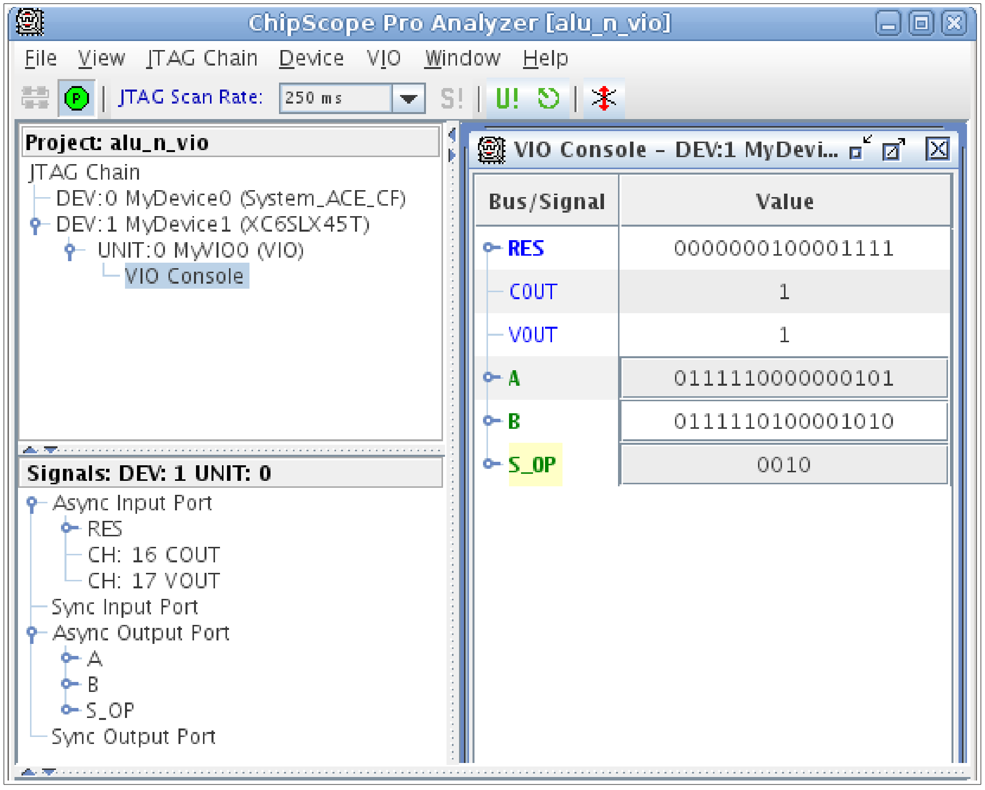

File View JTAG Chain Device Wind
Project: mux_vio
JTAG Chain
DEV:0 MyDevice0 (System_ACE_CF)
DEV:1 MyDevice1 (XC6SLX45T)
UNIT:0 MyVIO0 (VIO)
VIO Console

Figure 21

Select signals `AsyncIn[0]` to `[15]`, right-click on them and select `Move to Bus->New Bus`. Right-click on the newly created `AsyncIn` bus, select `Rename` and type `RES` to the Input window. Right-click on the `AsyncIn[16]` signal, select `Rename` and type `COUT` to the Input window. Right-click on the `AsyncIn[17]` signal, select `Rename` and type `VOUT` to the Input window. Select signals `AsyncOut[0]` to `[15]`, right-click on them and select `Move to Bus->New Bus`. Right-click on the newly created `AsyncOut` bus, select `Rename` and type `A` to the Input window. Select signals `AsyncOut[16]` to `[31]`, right-click on them and select `Move to Bus->New Bus`. Right-click on the newly created `AsyncOut_1` bus, select `Rename` and type `B` to the Input window. Select signals `AsyncOut[31]` to `[35]`, right-click on them and select `Move to Bus->New Bus`. Right-click on the newly created `AsyncOut_2` bus, select `Rename` and type `S_OP` to the Input window.

Right-click on each bus and select the desired radix (`Bus Radix`). Give values to A, B and S_OP buses and observe the `alu_n` module outputs. **Figure 22** shows the execution of the A XOR B operation.



Bus/Signal	Value
RES	0000000100001111
COUT	1
VOUT	1
A	0111110000000101
B	0111110100001010
S_OP	0010

Figure 22

The `COUT` and `VOUT` outputs don't carry the `'Z'` value as they were supposed to; they carry the logic value `'1'` instead. This is due a synthesis warning that states : `WARNING:Xst:2042 - Unit alu_n: 18 internal tristates are replaced by logic (pull-up yes): COUT, RES<0>, RES<10>, RES<11>, RES<12>, RES<13>, RES<14>, RES<15>, RES<1>, RES<2>, RES<3>, RES<4>, RES<5>, RES<6>, RES<7>, RES<8>, RES<9>, VOUT`. This warning is generated because the Spartan-6 configurable logic blocks (CLBs) don't have internal tristate buffers [5]. So all the tristate buffers in the design are converted to logic that behaves consistently (`'Z'` is replaced by `'1'`). If the `alu_n` module was implemented using general purpose FPGA pins, this warning would not have been generated because the Spartan-6 inpout/output blocks (IOBS) contain tristate buffers [6].

References

[1] Edwin Naroska, comp.lang.vhdl Frequently Asked Questions And Answers, http://www.vhdl.org/comp.lang.vhdl/

[2] Stephen Brown and Zvonko Vranesic, Fundamentals of Digital Logic with VHDL Design, 3rd Edition, McGraw-Hill, 2009

[3] Xilinx Corporation, LogiCORE IP ChipScope Pro Integrated Controller (ICON) (v1.05a), 2011, http://www.xilinx.com

[4] Xilinx Corporation, LogiCORE IP ChipScope Pro Virtual Input/Output (VIO) (1.04a), 2011,

http://www.xilinx.com

[5] Xilinx Corporation, Spartan-6 FPGA Configurable Logic Block User Guide, 2010, http://www.xilinx.com

[6] Xilinx Corporation, Spartan-6 FPGA SelectIO Resources User Guide, 2010, http://www.xilinx.com