Lecture 5: More on Testbenches

The objective of this supplemental material is to reinforce the concept of testbenches in VHDL.

1. Introduction

On alternative way to verify the correctness of a VHDL description of a design is to use testbenches.
A *testbench* is an enclosing VHDL model. Its name comes from the analogy with a real hardware testbench, on which a Device Under Test (DUT) is stimulated with signal generators and observed with signal probes. A VHDL testbench consists of an architecture body containing an instance of the component to be tested and processes that generate sequences of values on signals connected to the component instance. The architecture body may also contain processes that test the component instance produces the expected values on its output signals.

During this supplemental lab you will write the VHDL model for a registered ALU using a package, and test it using a testbench. Your ALU is capable of performing four operations on two operands as shown in Fig.1. The flag output is high (logic '1') whenever there is either an underflow or overflow on the C bus.

func(1:0)	Operation
00	Unsigned add
01	Signed add
10	Unsigned sub
11	Signed add

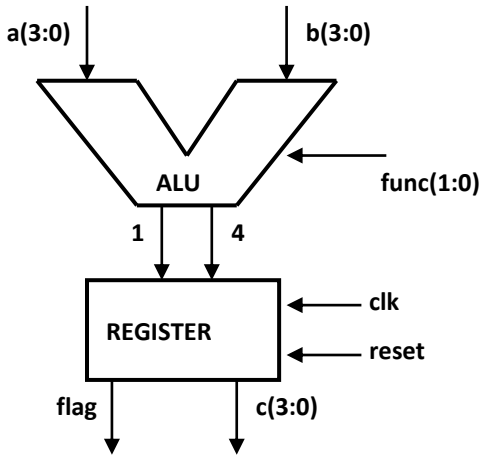

Figure 1 Simple ALU

2. Writing the package

As you already learned, a VHDL package is an important way of grouping a collection of related declarations that serve a common purpose. Usually, a package is a set of subprograms that provide operations on a particular type of data, or they might be just the set of declarations needed to model a design.
The important thing is that they can be collected together into a separate design unit that can be worked on independently and reused in different parts of a model or models.
The following VHDL code describes all the operations needed to implement the four basic operations of your simple ALU. Type it using any text editor (or using the VHDL editor of ISE WebPack) and save it as **alupack.vhd**.

```
-------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
-------------------------------------------
-- package declarations for procedures and constants
package addorsub is
-- set the default bus size
constant bussize : integer := 4;
-- set up a type for a bus of size bussize
subtype stdbus is signed (3 downto 0);
subtype lrgbus is signed (4 downto 0);
-- set the integer range for a bus of size bussize + 1
subtype medint is integer range -32 to 31;

-- extend performs a one bit signed or signed bit extension based
-- on the value of signex. signex=1 does a signed extension.
procedure extend (signal inbus : in stdbus; variable outbus : out
                  lrgbus; signex : in std_logic);

-- usadd performs signed or signed addition of two busses of size
-- bussize. the result is a signed or signed bus of size bussize
-- depending on signex (signex = 1 produces a signed result). reportf
-- indicates if there is an underflow or overflow.
procedure usadd (signal abus, bbus : in signed(bussize-1 downto 0);
                  signal result : out signed(bussize-1 downto 0);
                  signex : in std_logic;
                  signal reportf : out std_logic);

-- ussub performs signed or signed subtraction (abus - bbus)
-- of two busses of size bussize (signex=1 causes signed subtraction).
-- reportf =1 if there is an underflow or overflow.
procedure ussub (signal abus, bbus : in signed(bussize-1 downto 0);
                  signal result : out signed(bussize-1 downto 0);
                  signex : in std_logic;
                  signal reportf : out std_logic);
end addorsub;

-------------------------------------------
-- package body contains the procedure bodies.
package body addorsub is

procedure extend (signal inbus : in stdbus; variable outbus : out
                  lrgbus; signex : in std_logic) is
begin
outbus := (signex and inbus (bussize-1)) & inbus(bussize-1 downto 0);
end;

procedure usadd (signal abus, bbus : in signed(bussize-1 downto 0);
                  signal result : out signed(bussize-1 downto 0);
                  signex : in std_logic;
                  signal reportf : out std_logic) is
variable tempr : medint;

variable tempa : signed(bussize downto 0);
variable tempb : signed(bussize downto 0);
```

```
begin
-- sign/unsign extend abus and bbus to a bus of size bussize + 1;
extend(abus, tempa, signex);
extend(bbus, tempb, signex);
--perform signed addition
tempr := to_integer(tempa)+ to_integer(tempb);
-- check for overflows dependent on type of addition
if (signex = '0' and tempr > 15) then
      --overflow of signed addition
      reportf <= '1';
elsif (signex = '1' and (tempr > 7 or tempr < -8)) then
      -- overflow or underflow of signed addition
      reportf <= '1';
else
      reportf <= '0';
end if;
result <= to_signed(tempr, bussize);
end usadd;

procedure ussub (signal abus, bbus : in signed(bussize-1 downto 0);
                 signal result : out signed(bussize-1 downto 0);
                 signex : in std_logic;
                 signal reportf : out std_logic) is
variable tempr : medint;
variable tempa : signed(bussize downto 0);
variable tempb : signed(bussize downto 0);
begin
-- sign/unsign extend abus and bbus to a bus of size bussize+1;
extend(abus, tempa, signex);
extend(bbus, tempb, signex);
-- perform signed addition
tempr := to_integer(tempa)- to_integer(tempb);
-- check for overflows dependent on type of addition
if (signex = '0' and tempr < 0) then
      reportf <= '1';
elsif (signex = '1' and (tempr > 7 or tempr < -8)) then
      -- overflow or underflow of signed addition
      reportf <= '1';
else
      reportf <= '0';
end if;
result <= to_signed(tempr, bussize);
end ussub;

end addorsub; -- end of package body
-------------------------------------------
```

3. Writing the VHDL description of the ALU

The following VHDL code describes the ALU, which uses the functions declared and implemented in the package **alupack**. The ALU should have a register to latch the output. Type it using any text editor and save it as **alu.vhd**.

```
-------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
```

```
use WORK.addorsub.all;
-------------------------------------------
entity alu is
      port (a, b : in stdbus;
      func : in std_logic_vector(1 downto 0);
      clk, reset : in std_logic;
      flag : out std_logic;
      c : out stdbus);
end alu;
-------------------------------------------
architecture rtl of alu is
signal intflag : std_logic;
signal intbus : stdbus;
begin

regp : process (clk, reset)
begin
if (reset = '1') then
      flag <= '0';
      c <= "0000";
elsif (clk'event and clk = '0') then
      flag <= intflag;
      c <= intbus;
end if;
end process regp;

alup : process(a, b, func)
begin
if func(1) = '0' then
      usadd(a, b, intbus, func(0), intflag);
else
      ussub(a, b, intbus, func(0), intflag);
end if;
end process alup;

end rtl;
-------------------------------------------
```

4. Writing the testbench

The following VHDL code represents the testbench. It generates inputs for and monitors the outputs from the ALU. The testbench compares the actual outputs with expected outputs and prints out if a test is successful or not. Note that you do not need a stimulus file when you work with testbenches; the design is stimulated with stimulus generated inside the testbench.

Type the following VHDL code and save it as **testbench.vhd**.

```
-------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use WORK.addorsub.all;
-------------------------------------------
entity testbench is
end testbench;
```

```
-------------------------------------------
architecture test of testbench is
type table_type1 is array (0 to 5) of signed (3 downto 0);
type table_type2 is array (0 to 3) of std_logic_vector (1 downto 0);
constant inputa : signed := "0000";
constant inputb : signed := "0000";
constant outc : table_type1 := ("0001", "0011", "0101", "0111", "1001","1011");
constant outgen : table_type2 := ("00", "01", "10", "11");
signal cbus : signed (3 downto 0);
signal flag : std_logic;
signal abus : signed (3 downto 0) := "0000";
signal bbus : signed (3 downto 0) := "0000";
signal clk : std_logic;
signal reset : std_logic;
signal sel : std_logic_vector (1 downto 0) := "00";

component alu
port (a, b : in stdbus;
      func : in std_logic_vector(1 downto 0);
      clk, reset : in std_logic;
      flag : out std_logic;
      c : out stdbus);
end component;
for alu_inst : alu use entity work.alu(rtl);

begin
alu_inst : alu port map (abus, bbus, sel, clk, reset, flag, cbus);

clkp : process
begin
clk <= '1', '0' after 50 ns;
wait for 100 ns;
end process clkp;

rset : process
begin
reset <= '1', '0' after 100 ns;
wait for 1 ms;
end process rset;

testp : process
begin
wait for 100 ns; -- this is needed for reset to finish
for j in 0 to 1 loop -- test for unsigned & signed add
      sel <= outgen(j);
      for i in 0 to 5 loop
            abus <= inputa + TO_SIGNED(i, 4);
            bbus <= inputb + TO_SIGNED(i+1, 4);
            wait for 51 ns;
            assert (cbus = outc(i))
            report "Result is not correct"
            severity warning;
            wait for 49 ns;
      end loop;
end loop;
for j in 2 to 3 loop -- test for unsigned & signed sub
      sel <= outgen(j);
```

```
      for i in 0 to 5 loop
            abus <= inputa + TO_SIGNED(i, 4);
            bbus <= inputb + TO_SIGNED(i+1, 4);
            wait for 51 ns;
            assert (cbus = "1111”)
            report “Result is not correct”
            severity warning;
            wait for 49 ns;
      end loop;
end loop;
assert false
report “Test Complete”
severity error;
end process testp;

end test;
-----------------------------------------------
```

Read thoroughly the above files to understand the functionality of the testbench, then:

- Use Aldec HDL simulator to simulate alu.vhd together with alupack.vhd. Create your own input signals (as in lab#1) to stimulate the four basic operations performed by the ALU and verify its correctness.
- Simulate testbench.vhd (together with alu.vhd and alupack.vhd) to verify the ALU. Notice that using testbeches saves your time.

5. Lab assignment

You are required to modify the ALU design such that it can be implemented with ISE WebPack and verified on the Atlys board. You must add a clock divider to provide a clock frequency of 1 Hz to the ALU unit. The clock divider uses as input the 100 MHz signal of the Atlys board.
Use output c(3:0) to drive LEDs. The LEDs must display either a number between 0-15 for unsigned operations, or a number between 0-7 for the signed operations. The output "flag" should drive the left most LED. As inputs a(3:0) and b(3:0) use all eight slide-switche. As func(1:0) use the two push-buttons. Synthesize and implement this modified ALU and download its bitstream file to the board to configure the FPGA. Verify the correct operation.