Code example: Product of two complex numbers

Jordi Bonet

iTIC http://ocw.itic.cat

March 2020

multiplier.vhd (1/6)

```
library ieee;
use ieee.std_logic_1164.all, ieee.numeric_std.all;

entity multiplier is
  generic (WIDTH: natural:=5);
  port( clk,reset    : in std_logic;
        input_rdy    : in std_logic;
        a_r,a_i,b_r,b_i: in signed(WIDTH-1 downto 0);
        p_r,p_i      : out signed(2*WIDTH-1 downto 0));
end;

architecture arch of multiplier is
  -- datapath
  signal a_operand, b_operand : signed(WIDTH-1 downto 0);
  signal pp, pp1, pp2, sum : signed(2*WIDTH-1 downto 0);

  -- control seection
  signal a_sel,b_sel,
    pp1_ce, pp2_ce,
    sub, p_r_ce, p_i_ce : std_logic;

  -- control section (finite state machine)
  type multiplier_state is (step1, step2, step3, step4, step5);
  signal current_state, next_state : multiplier_state;
```

multiplier.vhd (2/6)

```
begin
  -- datapath
  a_operand <= a_r when a_sel='0' else a_i;
  b_operand <= b_r when b_sel='0' else b_i;

  pp <= a_operand*b_operand;

  pp1_reg : process(clk) is
  begin
    if rising_edge(clk) then
      if pp1_ce='1' then
        pp1 <= pp;
      end if;
    end if;
  end process;

  pp2_reg : process(clk) is
  begin
    if rising_edge(clk) then
      if pp2_ce='1' then
        pp2 <= pp;
      end if;
    end if;
  end process;
```

multiplier.vhd (3/6)

```
sum <= pp1 + pp2 when sub='0'else pp1-pp2;

p_r_reg : process(clk) is
begin
  if rising_edge(clk) then
    if p_r_ce='1' then
      p_r <= sum;
    end if;
  end if;
end process;

p_i_reg : process(clk) is
begin
  if rising_edge(clk) then
    if p_i_ce='1' then
      p_i <= sum;
    end if;
  end if;
end process;
```

multiplier.vhd (4/6)

```
-- control section (implemented with a finite state machine)
state_reg : process(clk,reset) is
begin
  if reset='1' then
    current_state <= step1;
  elsif rising_edge(clk) then
     current_state <= next_state;
  end if;
end process;
```

multiplier.vhd (5/6)

```
 next_state_logic : process(current_state, input_rdy) is
begin
  case current_state is
    when step1 =>
      if input_rdy = '0' then
        next_state <= step1;
      else
        next_state <= step2;
      end if;
     when step2 =>
        next_state <= step3;
     when step3 =>
        next_state <= step4;
     when step4 =>
        next_state <= step5;
     when step5 =>
      next_state <= step1;
  end case;
end process;
```

multiplier.vhd (6/6)

```
  output_logic : process (current_state) is
  begin
    a_sel <= '0';b_sel <= '0';
    pp1_ce <= '0';pp2_ce <= '0';
    sub <= '0'; p_r_ce <= '0';p_i_ce <= '0';
    case current_state is
      when step1 =>
        pp1_ce <= '1';
      when step2 =>
        a_sel <= '1'; b_sel <= '1';
        pp2_ce <= '1';
      when step3 =>
        sub <= '1'; p_r_ce <= '1';
        b_sel <= '1';
        pp1_ce <= '1';
      when step4 =>
        a_sel <= '1';
        pp2_ce <= '1';
      when step5 =>
        p_i_ce <= '1';
    end case;
  end process;

end;
```

multiplier_tb.vhd (1/5)

```
library ieee;
use ieee.std_logic_1164.all, ieee.numeric_std.all;

entity multiplier_tb is
  generic (t_WIDTH: natural:=5);
end;

architecture behav of multiplier_tb is
  signal t_a_r, t_a_i, t_b_r, t_b_i : signed(t_WIDTH-1 downto 0);
  signal t_p_r, t_p_i : signed(2*t_WIDTH-1 downto 0);
  signal t_clk, t_reset, t_input_rdy : std_logic;
begin
  dut: entity work.multiplier(arch)
    generic map (WIDTH=>t_WIDTH)
    port map (clk=>t_clk, reset=>t_reset, input_rdy=>t_input_rdy,
              a_r=>t_a_r, a_i=>t_a_i, b_r=>t_b_r, b_i=>t_b_i,
              p_r=>t_p_r, p_i=>t_p_i);
```

multiplier_tb.vhd (2/5)

```
-- clock of the system
clk_process: process
begin
  t_clk <= '0';
  wait for 300 us;
  for i in 1 to 20 loop
    t_clk <= not t_clk;
    wait for 500 us;
  end loop;
  wait;
end process;

-- reset signal
reset_process: process
begin
  t_reset<='1';
  wait until rising_edge(t_clk);
  wait for 100 us;
  t_reset<='0';
  wait;
end process;
```

multiplier_tb.vhd (3/56)

```
-- input_rdy signal
input_rdy_process: process
begin
  t_input_rdy<='0';
  for i in 1 to 3 loop
    wait until rising_edge(t_clk);
  end loop;
  wait for 100 us;
  t_input_rdy<='1';
  wait until rising_edge(t_clk);
  wait for 100 us;
  t_input_rdy<='0';
  wait;
end process;
```

multiplier_tb.vhd (4/5)

```
-- (a_r, a_i) and (b_r, b_i) signals
ab_process: process
  procedure apply_test
    (p_t_a_r, p_t_a_i,p_t_b_r, p_t_b_i : in integer) is
  begin
    t_a_r<=to_signed(p_t_a_r,t_a_r'length);
    t_a_i<=to_signed(p_t_a_i,t_a_i'length);
    t_b_r<=to_signed(p_t_b_r,t_b_r'length);
    t_b_i<=to_signed(p_t_b_i,t_b_i'length);
  end procedure;
begin
  for i in 1 to 3 loop
    wait until rising_edge(t_clk);
  end loop;
  wait for 100 us;
  apply_test(2,1,1,3);
  wait;
end process;
```

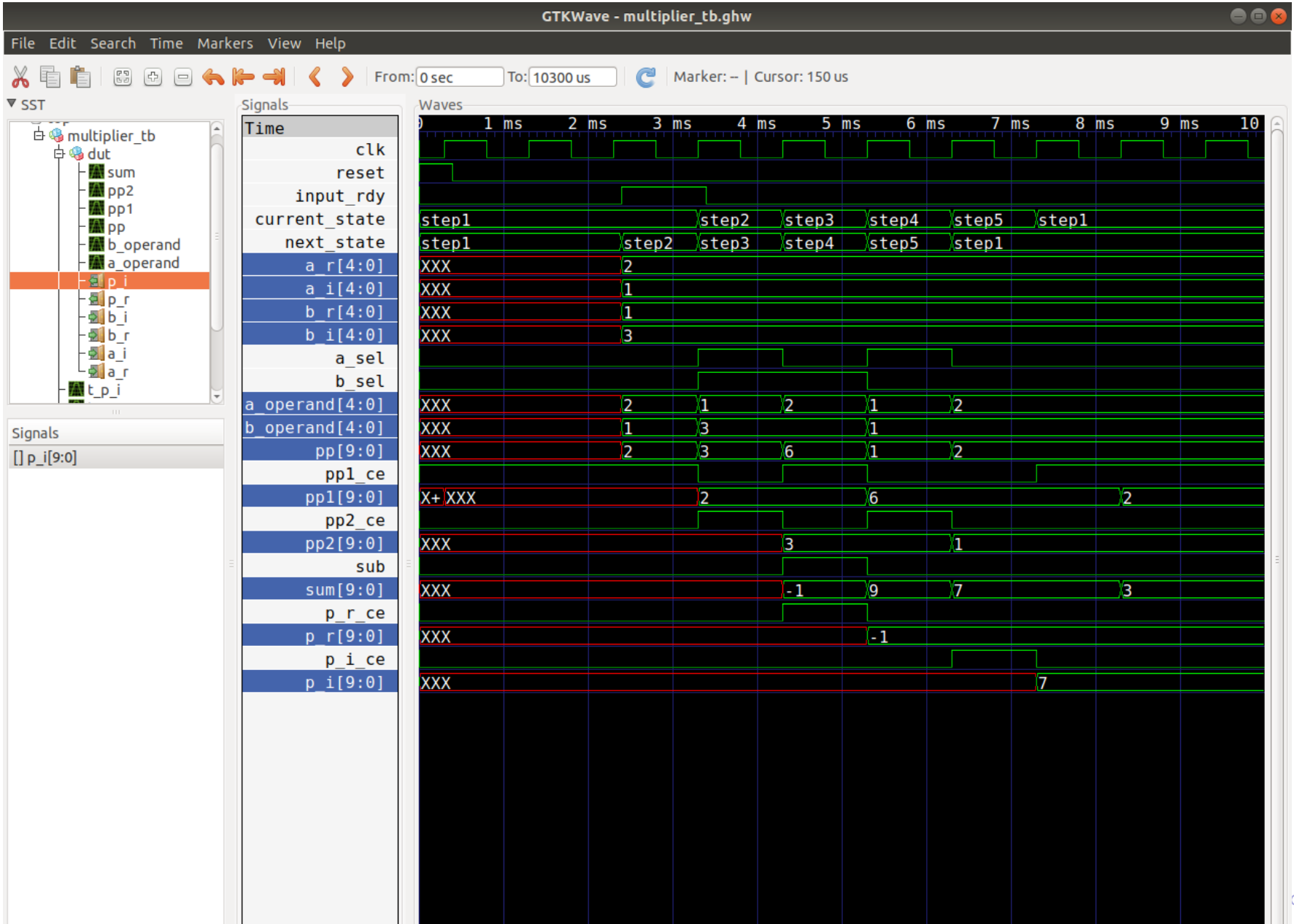
multiplier_tb.vhd (5/5)

```
  -- verification
  assert_process: process
    procedure p_assert
      (p_t_p_r, p_t_p_i: in integer) is
    begin
      assert t_p_r=to_signed(p_t_p_r,t_p_r'length)
       report "bad p_r computation" severity warning;
      assert t_p_i=to_signed(p_t_p_i,t_p_i'length)
       report "bad p_i computation" severity warning;
    end procedure;
  begin
        for i in 1 to 8 loop
      wait until rising_edge(t_clk);
    end loop;
        wait for 100 us;
    p_assert(-1,7);--5th rising_edge clk after input_rdy
    wait;
  end process;

end architecture;
```

Makefile

```
#Top-Level Entity
TLE = multiplier_tb
STOP_TIME = 200ms
GHDL_SIM_OPT = #--stop-time=$(STOP_TIME)
FORMAT = wave
EXTENSION = ghw
GHDL_SIM_RES = --$(FORMAT)=$(TLE).$(EXTENSION)
WAVEFORM_VIEWER = gtkwave

.PHONY: clean
all: compile elaborate run view
compile:
   ghdl -a *.vhd
elaborate:
   ghdl -e $(TLE)
run:
   ghdl -r $(TLE) $(GHDL_SIM_OPT) $(GHDL_SIM_RES)
view:
   $(WAVEFORM_VIEWER) $(TLE).$(EXTENSION)
clean:
   rm *.$(EXTENSION) work-obj93.cf *~
```

gtkwave


GTKWave - multiplier_tb.ghw
File Edit Search Time Markers View Help
From: 0 sec
To: 10300 us
Marker: -- | Cursor: 150 us
SST
multiplier_tb
dut
sum
pp2
pp1
pp
b_operand
a_operand
p_i
p_r
b_i
b_r
a_i
a_r
t_p_i
Signals
[] p_i[9:0]
Signals
Time
clk
reset
input_rdy
current_state
next_state
a_r[4:0]
a_i[4:0]
b_r[4:0]
b_i[4:0]
a_sel
b_sel
a_operand[4:0]
b_operand[4:0]
pp[9:0]
pp1_ce
pp1[9:0]
pp2_ce
pp2[9:0]
sub
sum[9:0]
p_r_ce
p_r[9:0]
p_i_ce
p_i[9:0]
Waves
1 ms
2 ms
3 ms
4 ms
5 ms
6 ms
7 ms
8 ms
9 ms
10
step1
step2
step3
step4
step5
step1
XXX
-1
7