DAT093
Introduction to Electronic System Design

Accessing a component from within a process

Sven Knutsson
svenk@chalmers.se
Dept. Of Computer Science and Engineering
Chalmers University of Technology
Gothenburg
Sweden

Introduction

We will look on how to use a component from a process.

Let´s s say that we want to use or earlier full adder from a clocked process.

The full adder is implemented as a block of hardware so it can not be placed within the process.

We must assign signals to the in- and outputs of the full adder and then assign values to these signals in the process.

Let´s look at our code.

VHDL code

We have the entity

```
ENTITY component_from_process IS
   PORT(a:IN STD_LOGIC;
        b:IN STD_LOGIC;
        cin:IN STD_LOGIC;
        clk:IN STD_LOGIC;
        s:OUT STD_LOGIC;
        cout:OUT STD_LOGIC);
END component_from_process;
```

VHDL code cont.


```
ENTITY component_from_process IS
   PORT(a:IN STD_LOGIC;
        b:IN STD_LOGIC;
        cin:IN STD_LOGIC;
        clk:IN STD_LOGIC;
        s:OUT STD_LOGIC;
        cout:OUT STD_LOGIC);
END component_from_process;
```


And we have the setup part of our architecture

```
ARCHITECTURE arch_component_from_process OF
                        component_from_process IS

   COMPONENT full_adder IS
      PORT(a:IN STD_LOGIC;
           b:IN STD_LOGIC;
           cin:IN STD_LOGIC;
           s:OUT STD_LOGIC;
           cout:OUT STD_LOGIC);
   END COMPONENT full_adder;

   SIGNAL a_comp_signal:STD_LOGIC;
   SIGNAL b_comp_signal:STD_LOGIC;
   SIGNAL cin_comp_signal:STD_LOGIC;
   SIGNAL s_comp_signal:STD_LOGIC;
   SIGNAL cout_comp_signal:STD_LOGIC;
BEGIN
   U0:COMPONENT full_adder
      PORT MAP(a=>a_comp_signal,
               b=>b_comp_signal,
               cin=>cin_comp_signal,
               s=>s_comp_signal,
               cout=>cout_comp_signal);
```

VHDL code cont.

And finally we have the process

```
full_adder_process:
   PROCESS(clk)
   BEGIN
      IF RISING_EDGE(clk) THEN
         a_comp_signal <= a;
         b_comp_signal <= b;
         cin_comp_signal <= cin;
         s <= s_comp_signal;
         cout <= cout_comp_signal;
      END IF;
   END PROCESS full_adder_process;
```

```
ENTITY component_from_process IS
   PORT(a:IN STD_LOGIC;
        b:IN STD_LOGIC;
        cin:IN STD_LOGIC;
        clk:IN STD_LOGIC;
        s:OUT STD_LOGIC;
        cout:OUT STD_LOGIC);
END component_from_process;

ARCHITECTURE arch_component_from_process OF
                  component_from_process IS
   COMPONENT full_adder IS
      PORT(a:IN STD_LOGIC;
           b:IN STD_LOGIC;
           cin:IN STD_LOGIC;
           s:OUT STD_LOGIC;
           cout:OUT STD_LOGIC);
   END COMPONENT full_adder;

   SIGNAL a_comp_signal:STD_LOGIC;
   SIGNAL b_comp_signal:STD_LOGIC;
   SIGNAL cin_comp_signal:STD_LOGIC;
   SIGNAL s_comp_signal:STD_LOGIC;
   SIGNAL cout_comp_signal:STD_LOGIC;
BEGIN
   U0:COMPONENT full_adder
      PORT MAP(a=>a_comp_signal,
               b=>b_comp_signal,
               cin=>cin_comp_signal,
               s=>s_comp_signal,
               cout=>cout_comp_signal);
```

VHDL code cont.

Let´s study the process

```
full_adder_process:
   PROCESS(clk)
   BEGIN
      IF RISING_EDGE(clk) THEN
         a_comp_signal <= a;
         b_comp_signal <= b;
         cin_comp_signal <= cin;
         s <= s_comp_signal;
         cout <= cout_comp_signal;
      END IF;
   END PROCESS full_adder_process;
```

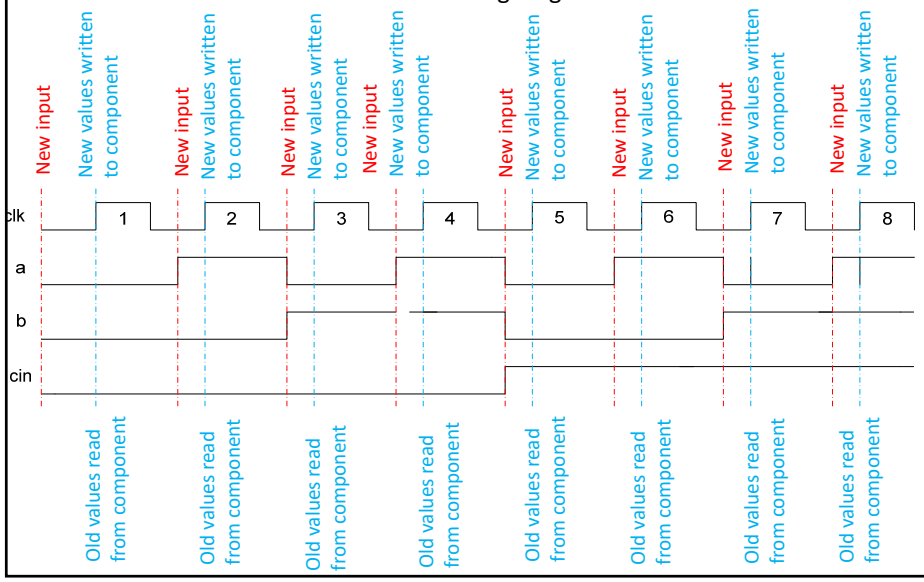
On each positive flank of our clock we will assign values to the full adder and read its outputs.

But the values we assign will not update the full adder until we reach the end of the process.

So what we read from the full adders outputs are actually related to the input values we assigned in the clock period before that.

VHDL code cont.

Let´s look at the timing diagram



VHDL code cont.

Let´s zom in