# VHDL Syntax

## Digital System design using VHDL

* Examples yet tob e included

Text in square brackets is optional

* -- comment
* Entity Declaration
  + entity entity\_name is

[port (interface-signal-declaration);]

end [entity] [entity\_name];

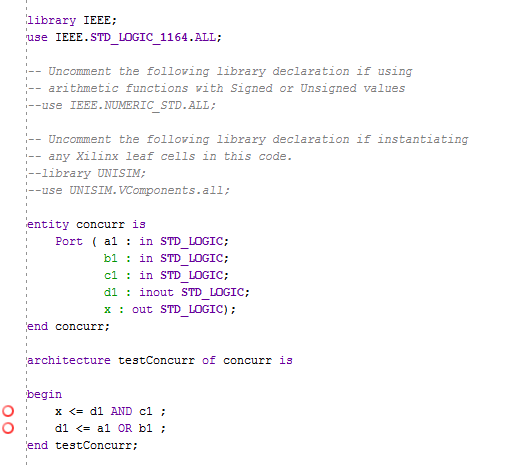
* Interface Signal Declaration
  + list-of-interface-signals : mode type [: = initial-value] ; list-of-interface-signals : mode type [: = initial-value] ; …
* Mode indicates the direction of information.
  + Types of Modes:
    - * in (input)
      * out (output)
      * inout (bidirectional generally output but can be used as an input when no output is specified or gate in in tristate mode) (ref: NPTEL lecture)
      * buffer (truly output signal but need to be read in the architecture of an entity)
      * linkage (connecting VHDL to non VHDL entities)
* Signal Assignment
  + signal\_name <= expression [after delay]
* Architecture Declaration
  + architecture architecture-name of entity-name is

[declarations]

begin

Architecture body

end [architecture] [architecture-name];

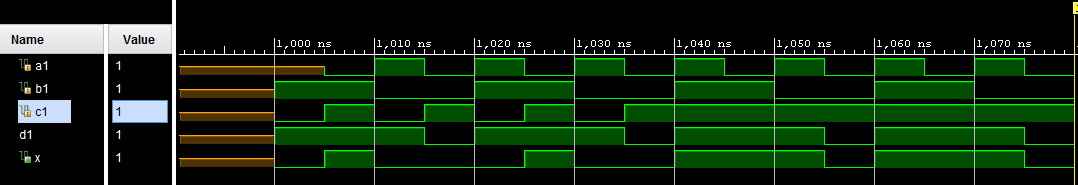


* Concurrent Statements:
  + Eg.

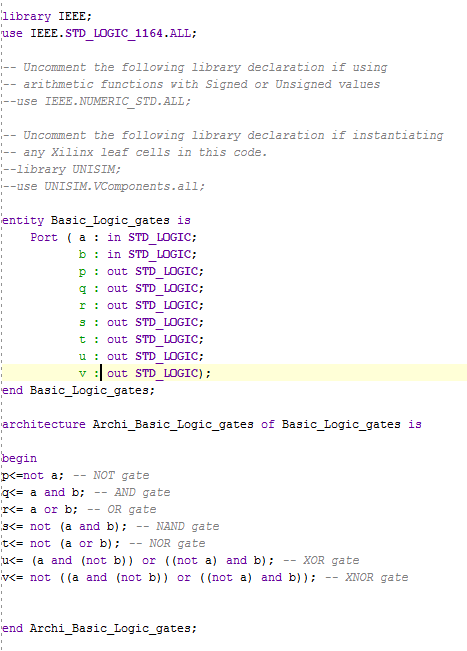
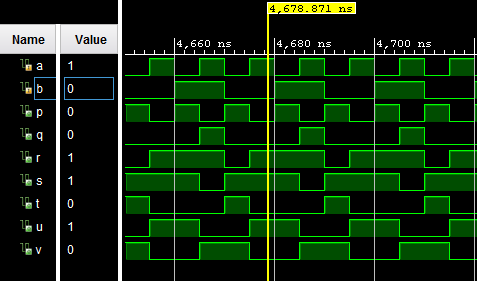
C <= A AND B; --A is inout B is input

A <= X OR Y; --X and Y are input

Whenever a signal on the right side changes the signal on the left is updated.



Basic Logic Gates



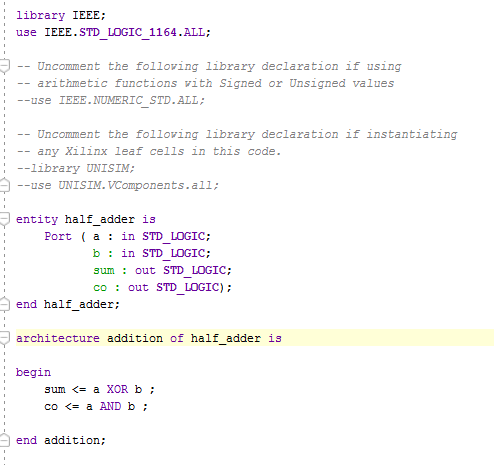
* Component: components within an architecture are declared at the start of the architecture using a component declaration form
  + component component-name

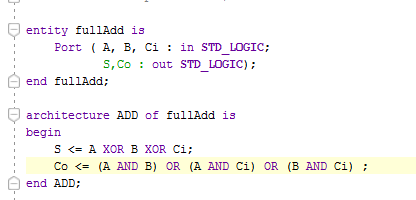
port(list-of-signals-and-their-types)

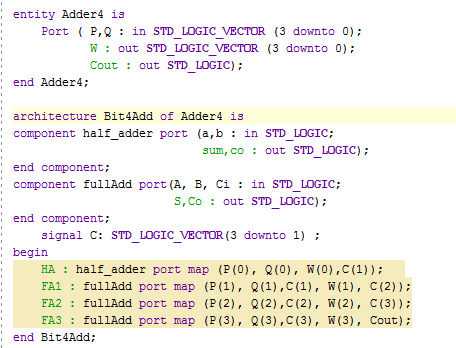
end component;

* port specific Assignment (a=>b)
* Using a component using label:
  + label: component-name port map (list-of-actual-signals) ;

*Adders*







* Process Declaration for Sequential Statements

Sequential Statements: statement within a process is called a sequential statement because it is executed sequentially. ‘if’ statements are always sequential.

* + process (sensitivity-list)

begin

sequential-statements

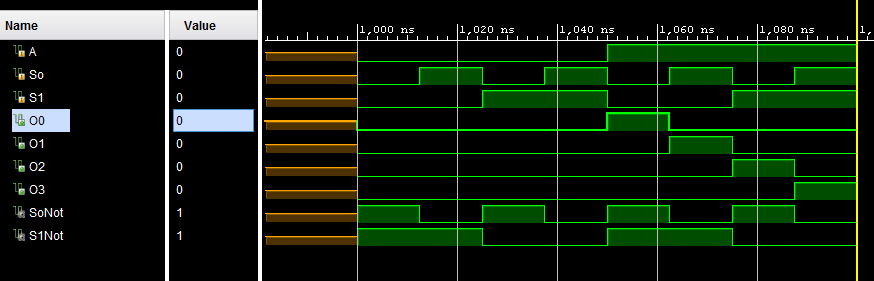
end process;

16.03.2018 Friday

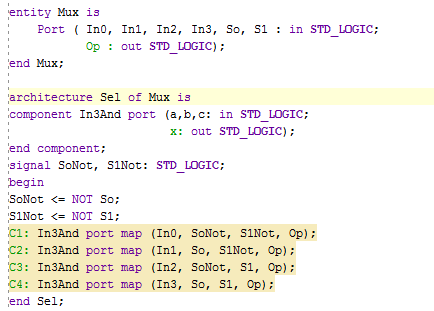
Examples:

Demultiplexer

|  |  |
| --- | --- |
|  |  |



Multiplexer:



* If statement

if condition then

sequential statements

elsif condition then

sequential statements

.

. ‑ -- More elsif statements

.

else

sequential statements

end if;

* Tick Event for detecting changes in a signal

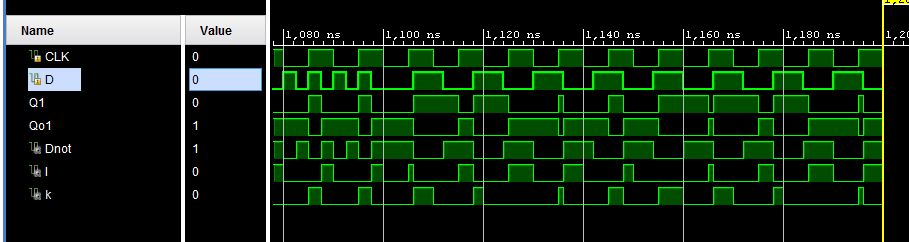
[Signal\_name]‘event

Read as “tick event”

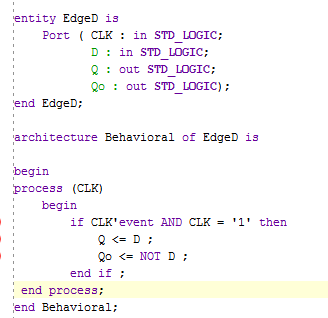
Edge triggered clock > clock’event returns a value 1 when clock changes.

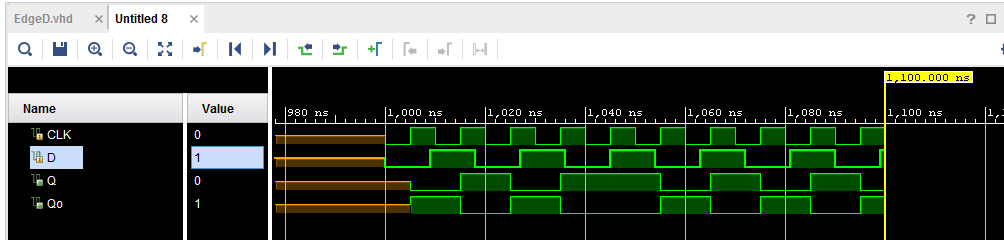
D LATCH

|  |  |  |
| --- | --- | --- |
| SR | D LATCH |  |
|  |  |  |



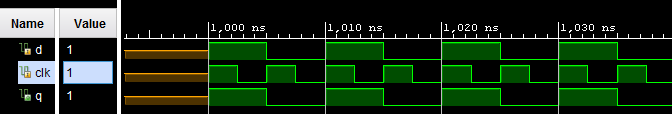
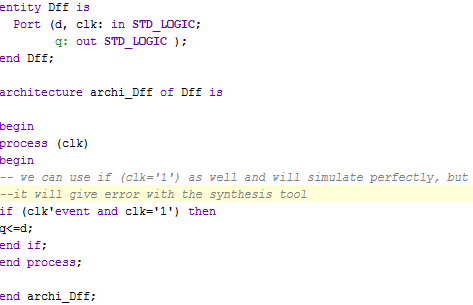
D Positive Edge Triggered:





*D Flip-Flop*

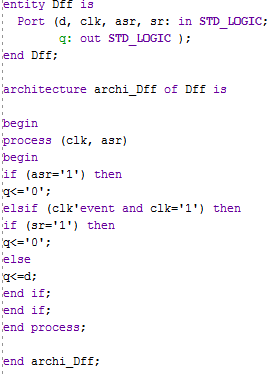
* Code for Simple D-flip flop

* D-flip flop with Asynchronus and Synchronus Reset

For asynchronous reset, we add the respective signal (asr) outside the flip-flop assignment nest, and include this signal in the sensitivity list of process as well.

For synchronous reset, we add the respective signal (sr) inside the flip-flop assignment nest, and there is no need to include this signal in the sensitivity list.



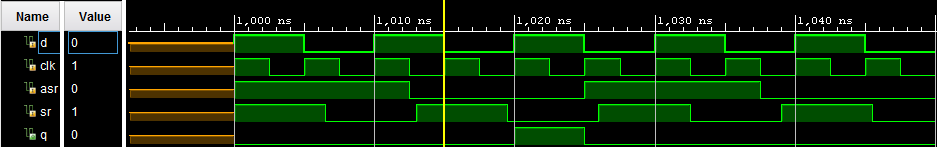
**Note:- Any combinational logic**

**that we assign inside the “if (clk'event and clk='1') then” condition will have a Flip-Flop at the end of it.**

a

Combi. Logic

D FF



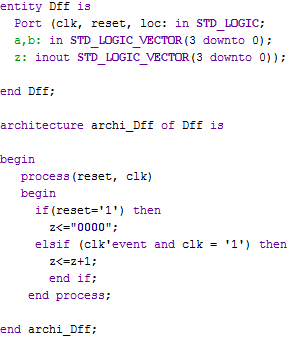
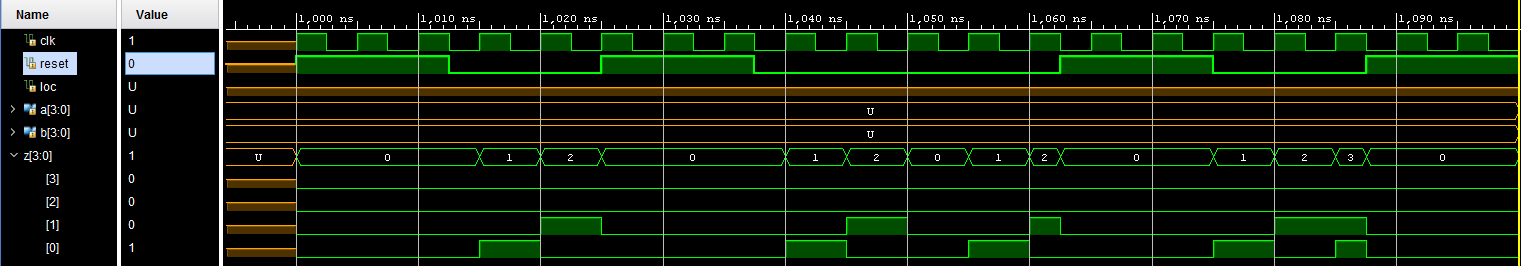
D-Flipflop

clk

*Counters*

Below is the code for a 4 bit synchronous counter with asynchronous reset available-





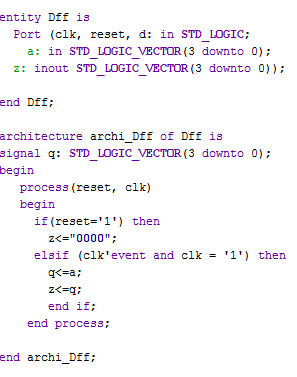
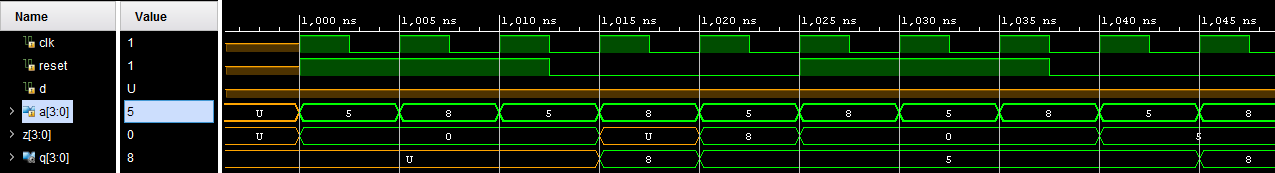
The declaration of ‘a’,’b’ and ‘loc’ are to be ignored here. They are nowhere needed in the code.

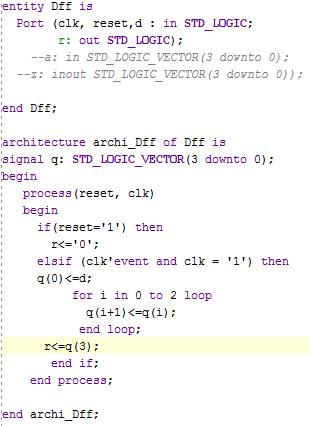
Since we are using “+” operator here, we also have to include the arithmetic operator library.

Also, since we are also using “z” in the RHS of assignment operator, we have to declare it as type ‘inout’.

*Shift Registers*

A basic **4 bit 2 stage right shift register**.



The adjoining figure shows the code for a **single bit 4   
stage right shift register** using “**for loop”.**

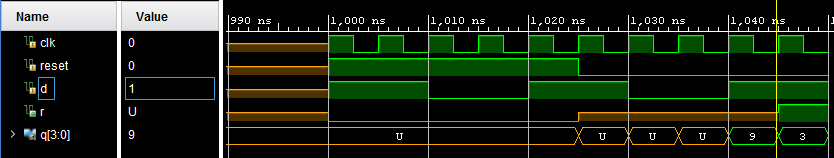
**For Loop Syntax:-**

for (looping variable) in (\_\_) to (\_\_) loop  
 …statements…;  
 ….statements…;  
end loop;

**Example:-**  
for i in 0 to 2 loop  
 ….statements…;   
 end loop;

**Note:-** “For loop” are purely sequential commands and can only be used with process/function/procedure. The equivalent concurrent code is “Generate Loop” command

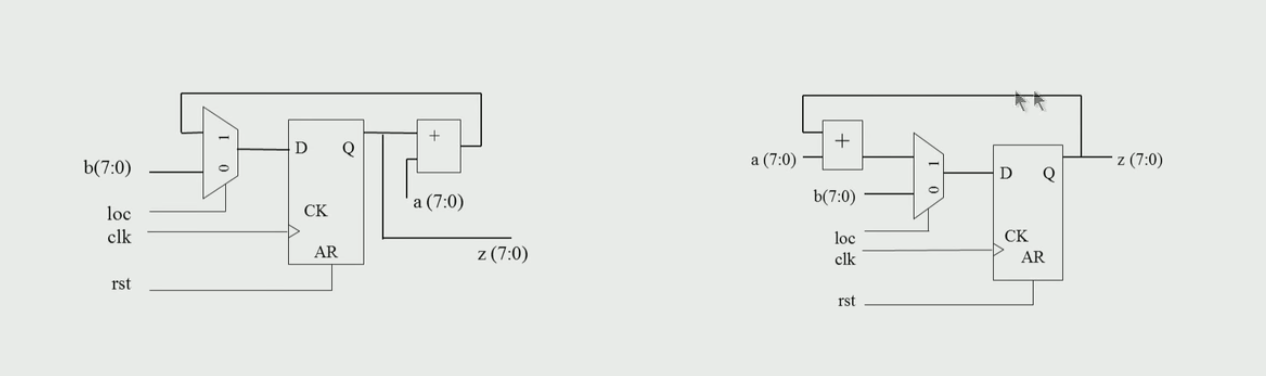
Given below is the output for the VHDL code above. As it is 4 stage shift register, the input takes 4 consecutive clock cycle to be reflected at the output.



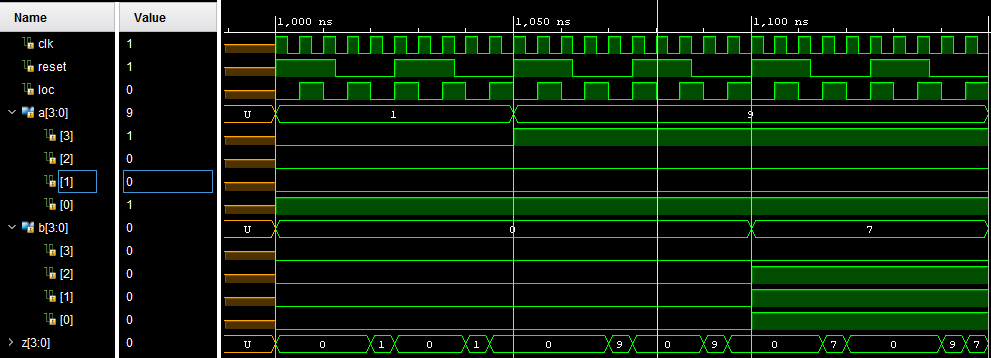
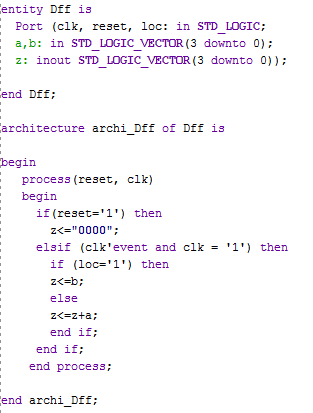
*Sample Question*

Reference: NPTEL Lecture No.-21

The task is to write a VHDL code for the block circuit given below.



* We should always at first reorient or align the circuit into standard structure that is known to us
* We should use the basic circuit coding of Mux/Demux, Counters, Adders, Flip/Flop and Registers to begin with



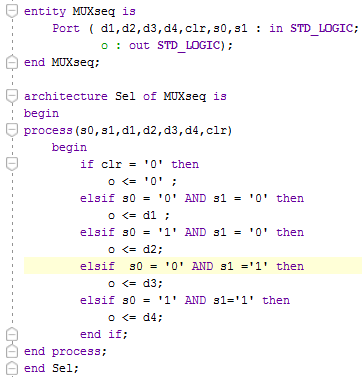
Since we are using “+” operator here, we also have to include the arithmetic operator library.

Also, since we are also using “z” in the RHS of assignment operator, we have to declare it as type ‘inout’.

Multiplexer/De-Multiplexer

There are multiple ways of simulating a Mux/Demux circuit. The simplest one involves utilization of **if-then-else** command syntax.

Since if-then-else is a sequential command, we can only use it within a process/function/procedure (Sequential programs are only written under process/function/procedure under Architecture blog of the main code)



* **WAIT Statements**: Process cannot have both wait statements and sensitivity list. A VHDL code will not simulate if there is no Sensitivity list or wait statement.

process

begin

sequential statements

wait statement

seq st

wait statement

end process

* Types of Wait statements:
  + wait on sensitivity list;

Eg. wait on A,B,C;

* + wait for time-expression;

Eg. wait for 5ns; -- Can be used for simulation not for synthesis

* + wait until Boolean-expression;

Eg. wait until A=B;

Waits until A or B changes and proceeds if the expression is true

* Multiple drivers for the same output
  + If several VHDL statements in a process update the same signal at a given time, the last value overrides.

Eg.

process(CLK)

begin

if CLK’event and CLK=’0’ then

Q <= A; Q<=B; Q<=C;

end if

end process

Q will have the value of C after every negative edge of clock.

# Transport Delay Vs Intertial delay

Default delay is inertial delay.

Inertial Delay: Delay the output signal by T and reject any pulse of width less than T.

Rejection pulse width can also be specified differently but it should be more than the inertial delay.

Eg. signal\_name <= reject pulse\_width after delay\_time

Rejects the pulse width less than pulse\_width specified and delays it by the delay\_time.

Transport Delay: Only delays the output does not reject any pulse.

Eg. Z1 <= transport X after 10ns

# WHEN ELSE STATEMENT

It is a concurrent statement

signal\_name <= expression1 **when** condition1 **else** expression2 **when** condition2 **else** expression3…

Select Statement:

Concurrent statement

with sel select

F <= Io when “00” ‑when sel is equal to “00”

I1 when “10” ‑ and so on…

General:

with expression\_s select

signal\_s <= expression1 [after delay\_time] when choice1,

expression2 [after delay\_time] when choice2,

--and so on

Expression\_n [after delay\_time] when others (if any)

Within a process

Sequential statement

case expression is

when choice0 => sequential statement1

when choice1 => sequential statement2

when choice2 => sequential statement3

when others => sequential statement4

end case