VHDL: A "Crash" Course

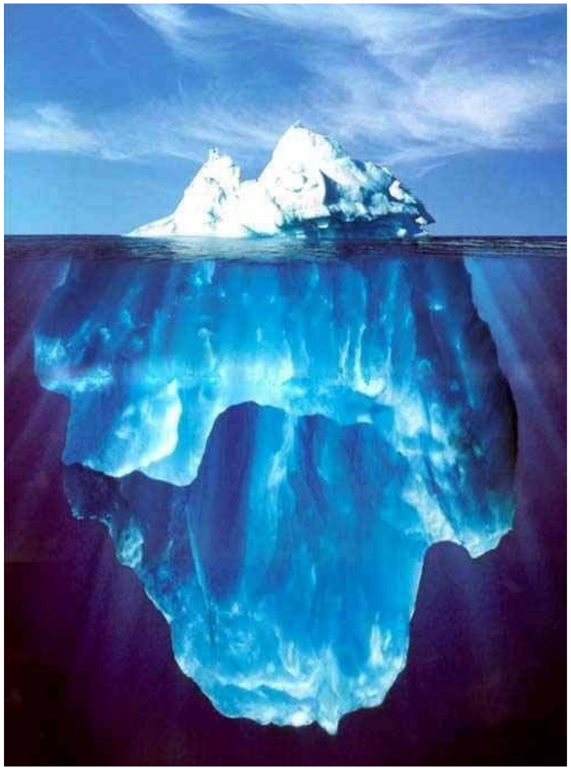
Dr. Manuel Jiménez

With contributions by: Irvin Ortiz Flores

Electrical and Computer Engineering Department
University of Puerto Rico - Mayaguez

Outline

- Background
- Program Structure
  - Types, Signals and Variables
- Description Styles
- Combinational Logic Design
- Finite State Machines
- Testbenches



What is VHDL?

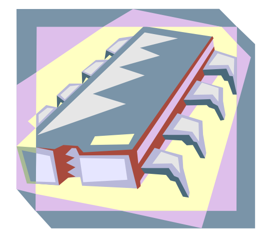
- **VHDL**: VHSIC Hardware Description Language
  - VHSIC= Very High Speed Integrated Circuit
- VHDL was created for modeling digital systems
  - Language subset used in HW synthesis
- Hierarchical system modeling
  - Top-down and bottom-up design methodologies

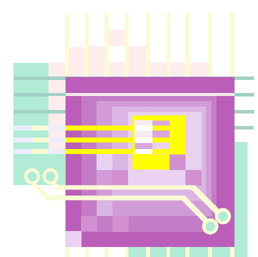
VHDL Retrospective

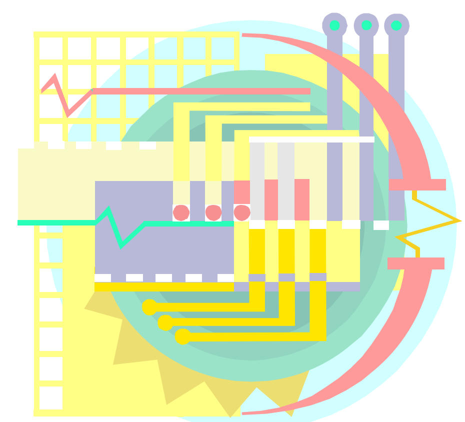
- VHDL is an IEEE and ANSI standard for describing digital systems
- Created in 1981 for the DoD VHSIC program
  - First version developed by IBM, TI, and Intermetric
  - First release in 1985
  - Standardized in 1987 and revised several times thereafter
    - Standard 1076, 1076.1 (VHDL-AMS), 1076.2, 1076.3
    - Standard 1164, VHDL-2006
  - Inherits many characteristics of ADA: Strong typed

VHDL Uses

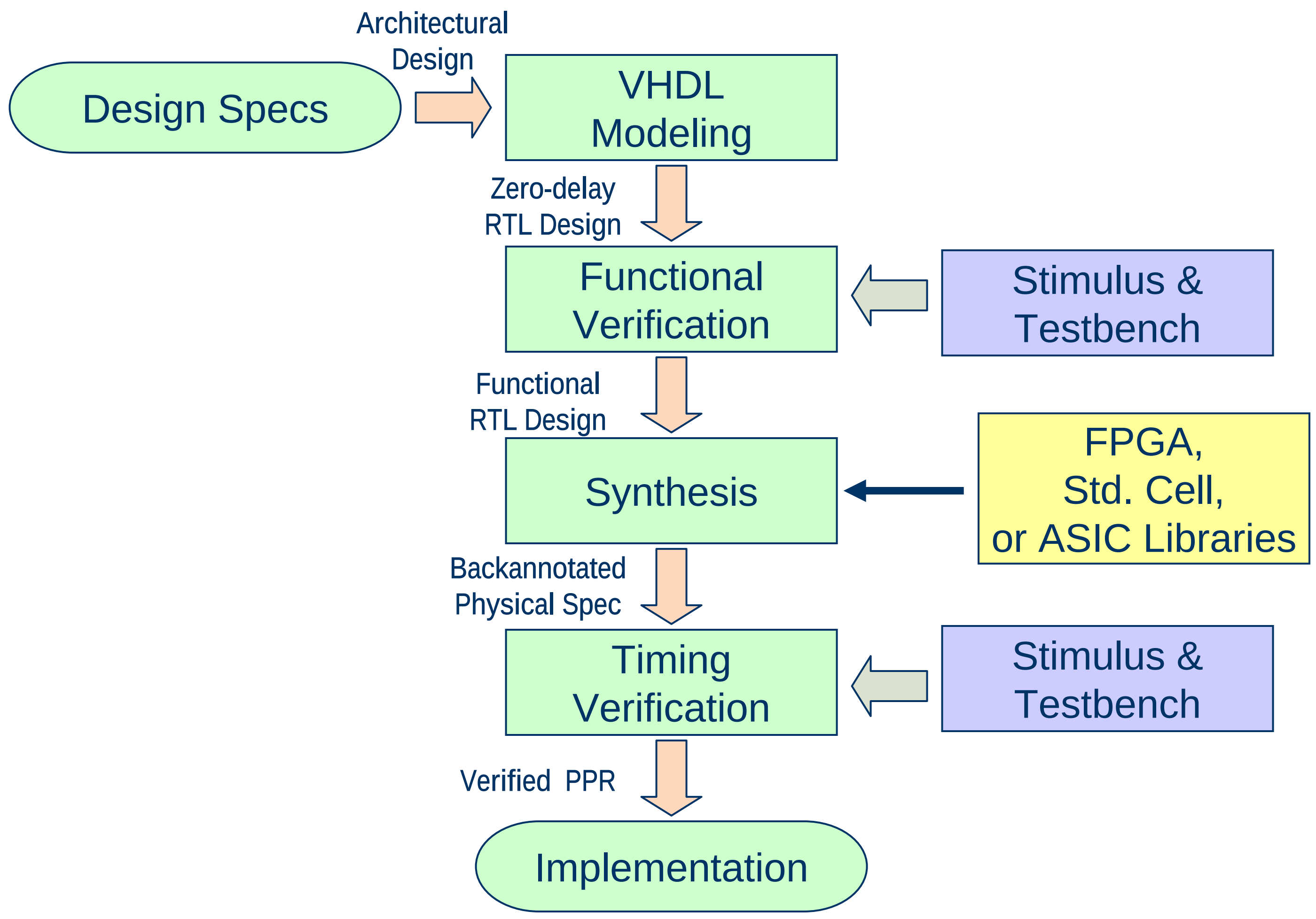
- Modeling of Digital Systems
  - Looks a High-level Language
- Synthesis of Digital Systems
  - Language Subset
- Synthesis Targets
  - FPGAs & FPLDs
  - ASICs
  - Custom ICs







VHDL-based Design Flow



Common VHDL Data Types

- Integer: Predefined in the range $-(2^{31})$ through $+(2^{31}-1)$. Subtypes can be declared
- Boolean: False, True
- Bit, std_logic: A single bit
- Bit_vector, std_logic_vector: Multiple bits
  - Range needs to be specified

Basic VHDL Program Structure

Library Inclusion

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
```

Entity Declaration

```
Entity Adder is
    port (A,B :  in   std_logic_vector(4 downto 0);
              Cin  : in   std_logic;
              Sum  : out  std_logic_vector(4 downto 0);
              Cout : out  std_logic);
End Adder;
```

Architecture Declaration

```
architecture a_adder of adder is
signal AC,BC,SC : std_logic_vector(5 downto 0);
begin
     AC <='0' & A;
     BC <='0' & B;
     SC <= unsigned(AC) + unsigned(BC) + Cin;
     Cout <= SC(5);
     Sout <= SC(4 downto 0);
end a_adder;
```

Entity Declaration

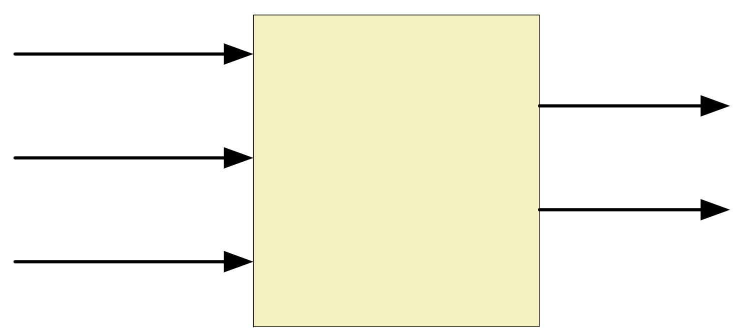
- Specifies interface
- States port's name, mode, & type
- Mode can be IN, OUT, or INOUT
- Port type can be from a single bit to a bit vector

Entity name

Port length

```
ENTITY Adder IS
 PORT (A,B  : IN    STD_LOGIC(4 DOWNTO 0);
        Cin  : IN    STD_LOGIC;
        Sum  : OUT   STD_LOGIC(4 DOWNTO 0);
        Cout : OUT  STD_LOGIC);
END Adder;
```

Port names　Port mode　Port type



Architecture Declaration

- Describes the internal operation of an entity
- Several architectures can be associated to one entity
- States which components, signals, variables, and constants will be used

An Architecture Example

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
Entity Adder is
      port (A,B :  in   std_logic_vector(4 downto 0);
            Cin  : in   std_logic;
            Sum  : out  std_logic_vector(4 downto 0);
            Cout : out  std_logic);
End Adder;

architecture a_adder of adder is

signal AC,BC,SC : std_logic_vector(5 downto 0);

begin
     AC <='0' & A;
     BC <='0' & B;
     SC <= unsigned(AC) + unsigned(BC) + Cin;
     Cout <= SC(5);
     Sout <= SC(4 downto 0);
end a_adder;
```

Library declaration section

Architecture declaration

Associated entity

Signal declaration. Also can be placed component, constants, types, declarations.

Concurrent Statements: Processed at the same time. Also component instantiations, and processes can be placed.

Signals Vs Variables (1/2)

- Signals
  - Can exist anywhere, like wires
  - Connect components or carry information between processes
  - When inside a process, its value is updated when the process suspends
  - Signal assignment operator: **<=**
- Variables
  - Can only exist inside a process
  - Behave like local HLL variables holding temporary values
  - Values updated right after assignment. Sequence matters
  - Variable assignment operator: **:=**

Concurrent Vs. Sequential Code

- Concurrent Statements
  - Occur typically outside a process
  - Take place concurrently, i.e. with simulation clock stopped
  - Uses of SIGNALS and processes
- Sequential Statements
  - Occur only inside a process
  - Are executed sequentially, i.e. one after another
  - Uses VARIABLES and functions

Signals Vs Variables (2/2)

- Signals
  - Initial values: A=5, B=15, X=10
  - Final values: A=10, B=5

```
Sigproc: process(A,X)
Begin
 A <= X;
 B <= A;
End process Sigproc;
```

- Variables
  - Initial values: A=5, B= 15, X=10
  - Final values: A=10, B=10

```
Sigproc: process(X)
Variable A,B : integer;
Begin
 A := X;
 B := A;
End process Sigproc;
```

Three-Bit Binary Counter

```
entity countl is
   port( clock, enable: in bit;
         qa: out integer range 0 to 7);
end countl;

architecture countarch of countl is
begin
   process (clock)
   variable count: integer range 0 to 7;
   begin
      if (clock'event and clock ='1') then
            if enable = '1' then
               count:=count + 1;
            end if;
      end if;
      qa <= count after 10 ns;
   end process;
end countarch;
```

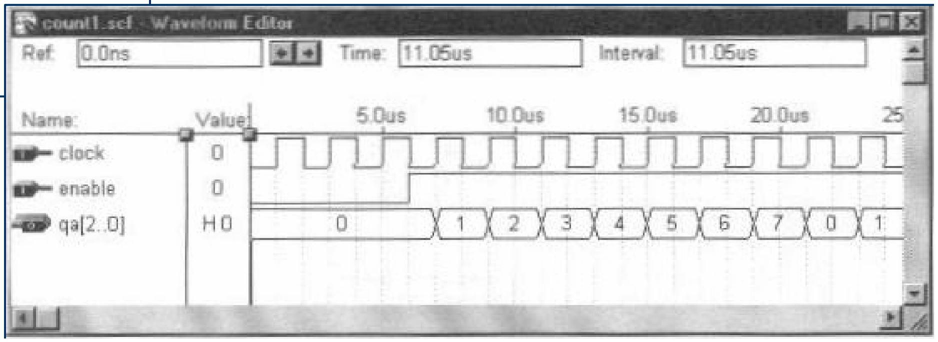
Sensitivity list. Process is executed each time one of this parameters change.

Variable declaration

Variable assignment operator

Sequential statements.

Signal assignment operator

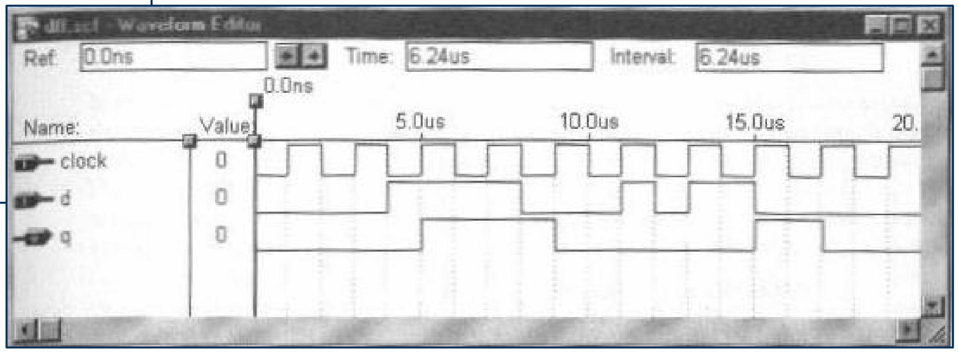


A "D" Flip-Flop

```
entity dff is
port (  d,clock : in bit;
        q: out bit);
end dff;

architecture arch of dff is
begin
    process (clock)
    begin
        if(clock'event and clock=1) then
            if(d=‘1’) then
                q <= ‘1’;
            else
                q <= ‘0’;
            end if;
        end if;
    end process;
end arch;
```

Refers to the rising edge of the clock

Explicit comparisons and assignments to port and signals uses ‘ ’ for one bit and “ ” for multiple bits



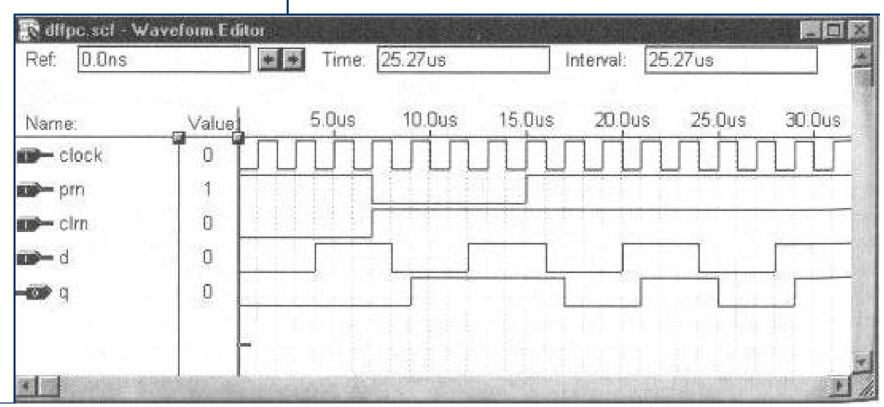
D-type flip-flop with active low preset and clear inputs

```
-- The pm and din signals are asynchronous
entity dffpc is
    port(d,clrn,prn,clock: in bit;
        q out bit);
end dffpc;
architecture arch of dffpc is
begin
    process (clock)
    begin
        if(clock'event and clock = '1') then
            if(d='l' and prn='l' and clrn='l') then
                q <= '1';
            elsif(d='0' and prn='l' and clrn ='1') then
                q <='0'.
            end if;
            --handle active low preset
            if(prn='0' and clrn='l') then
                q <= '1';
            end if;
            --handle active low clear
            if(clrn='0' and prn='l') then
                q <= '0';
            end if;
        end if;
    end process;
end arch;
```

Coments begin with --

Logical operators *and*, *or*, *not*, *nand*, *xor* are defined in the language

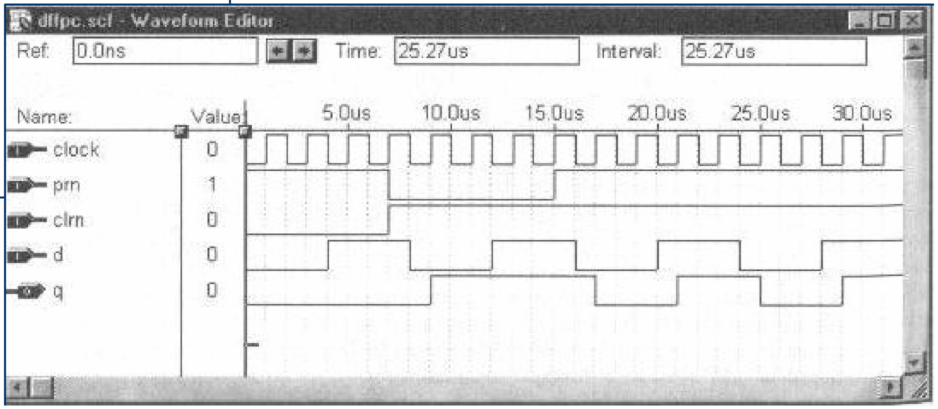
elsif is used instead of the *else if* of C language.



D Flip-Flop with Asynchronous Preset and Clear

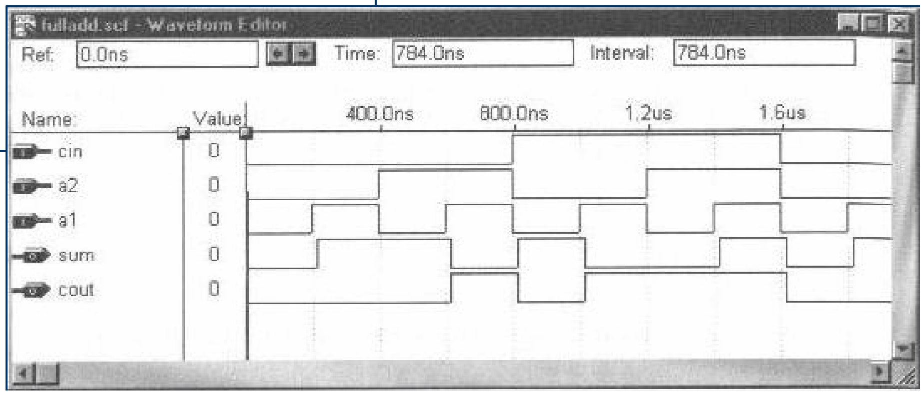
```
entity dffapc is
   port(clock, d, prn, clrn : in bit;
      q : out bit);
end dffapc;
architecture archl of dffapc is
begin
   process(clock, clrn, prn)
   variable reset, set: integer range 0 to 1;
   begin
      if(prn='0') then
         q <= '1';
      elsif (clrn='0') then
         q <= '0';
      elsif (clock'event and clock='1') then
         q <= d;
      end if;
   end process;
end archl;
```

Integer range definition. *Range 0 to 1* defines one bit.



Full Adder

```
library ieee;
use ieee.std_logic 1164.all;
entity fulladd is
   port( al,a2,cin: in std_logic;
         sum,cout: out std_logic);
end fulladd;

architecture fulladd of fulladd is
begin
   process(al,a2,cin)
   begin
      sum <= cm xor al xor a2;
      cout <= (al and a2) or (cin and (al xor a2));
   end process;
end fulladd;
```

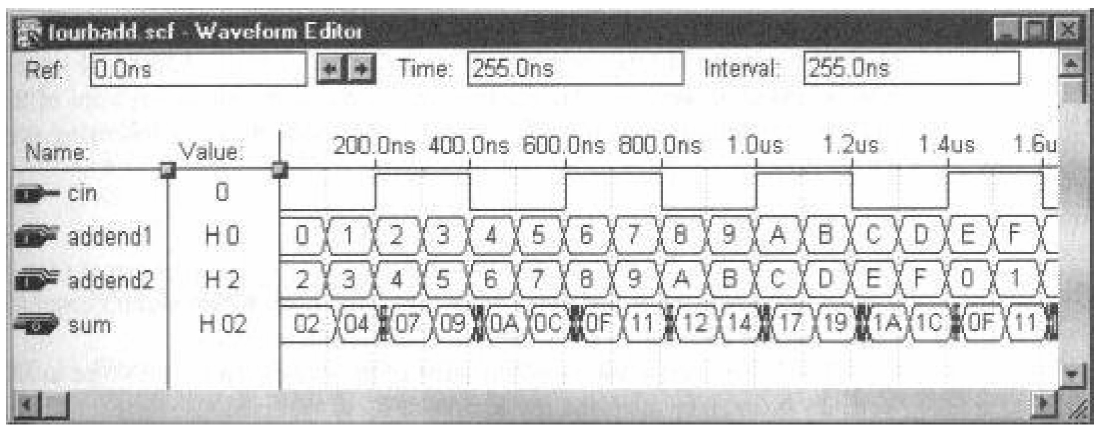


Four Bit Adder

```
--A VHDL 4 bit adder
entity fourbadd is
   port ( cin: in integer range 0 to 1;
          addendl:in integer range 0 to 15;
          addend2:in integer range 0 to 15;
          sum: out integer range 0 to 31);
end fourbadd;
architecture a4bitadd of fourbadd is
begin
   sum <= addendl + addend2 + cin;
end a4bitadd;
```

Integer type allows addition, subtraction and multiplication. Need the following statement at the library declaration section:

use IEEE.STD_LOGIC_ARITH.all

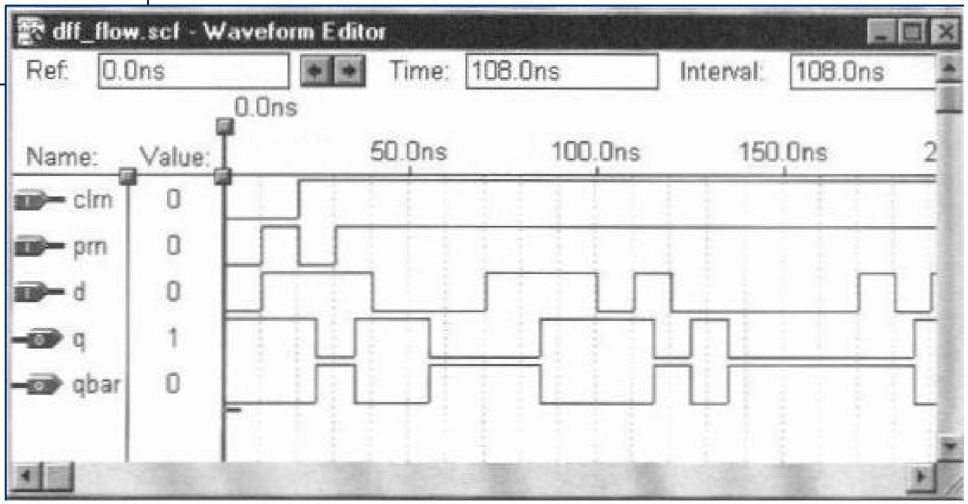


VHDL Description Styles

- Dataflow: Uses concurrent signal assignments
- Behavioral: Relies on process to implement sequential statements
- Structural: Describes the interconnections among components of a system. Requires hierarchical constructs.
- Mixed Method: Combines the three styles.

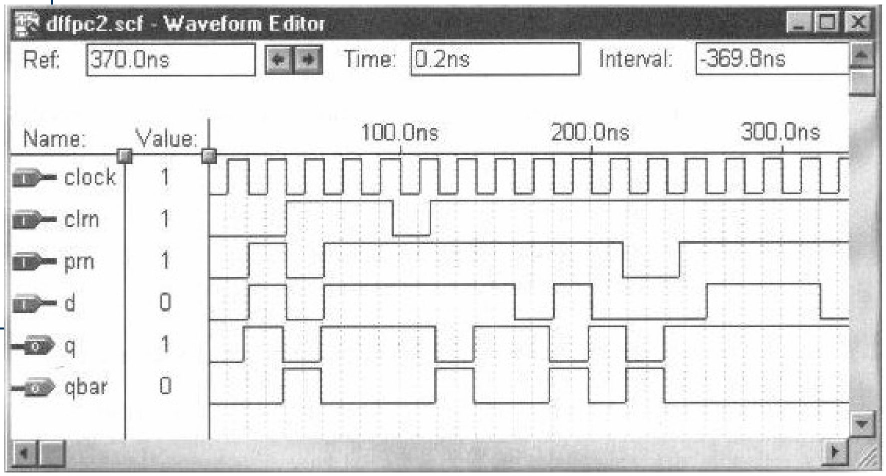
D Flip-Flop Dataflow

```
--D flip-flop dataflow
--Includes preset and clear
entity dff_flow is
   port ( d, prn, clrn: in bit;
          q,qbar: out bit);
end dff_flow;

architecture archl of dff_flow is
begin
   q <= not prn or (clrn and d);
   qbar <= prn and (not clrn or not d);
end archl;
```



Behavioral D Flip-Flop

```
--Active low preset and clear inputs
entity dffpc2 is
   port(d,clock,clrn,prn:in bit;
       q,qbar:out bit;
end dffpc2;

architecture arch of dffpc2 is
begin
   process(clock,clrn,prn)
      begin
      if(clock'event and clock = '1')
      then
         q <= not prn or (clrn and d);
         qbar <= prn and (not clrn or
         not d);
      end if;
   end process;
end arch;
```



D Flip-Flop Structural

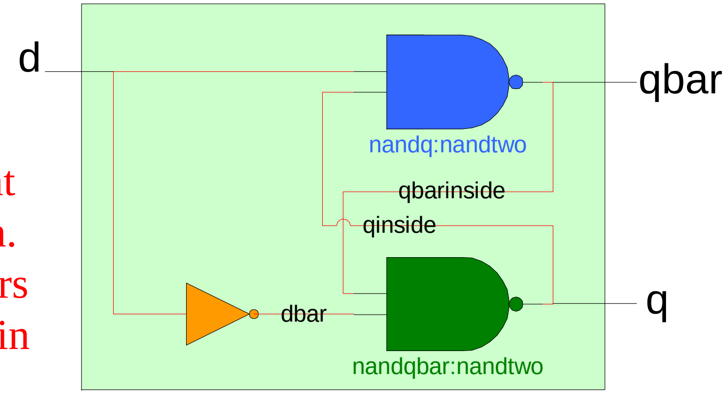
```
entity dff_str is
   port (d :in bit;
         q,qbar:out bit);
end dff_str;

architecture adff_str of dff_str is
component nandtwo
   port(x, y: in bit;
       z:out bit);
end component;
signal qbarinside, qinside, dbar: bit;
begin
   nandq:nandtwo
   port map(qinside, d,qbarinside);

   nandqbar:nandtwo
   port map(qbarinside,dbar, qinside);

   dbar <= not d;
   q <= qinside;
   qbar <= qbarinside;
end adff_str;
```

Component declaration. Port appears exactly as in the entity declaration.

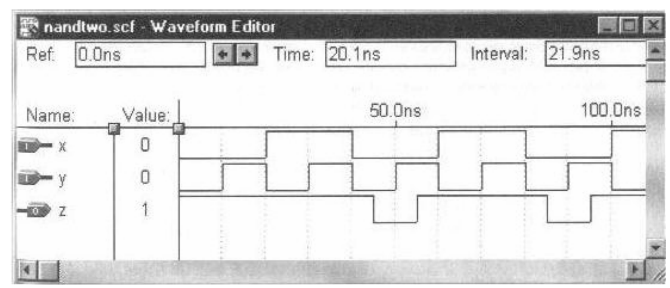
Entity name

Component instantiation label

Component instantiation. Connections are made by correspondence




```
--A two input nand gate
entity nandtwo is
   port(x, y:in bit;
        z :out bit);
end nandtwo;
architecture anandtwo of nandtwo
is
begin
   z <= not(x and y);
end anandtwo;
```



A Sequence Detector

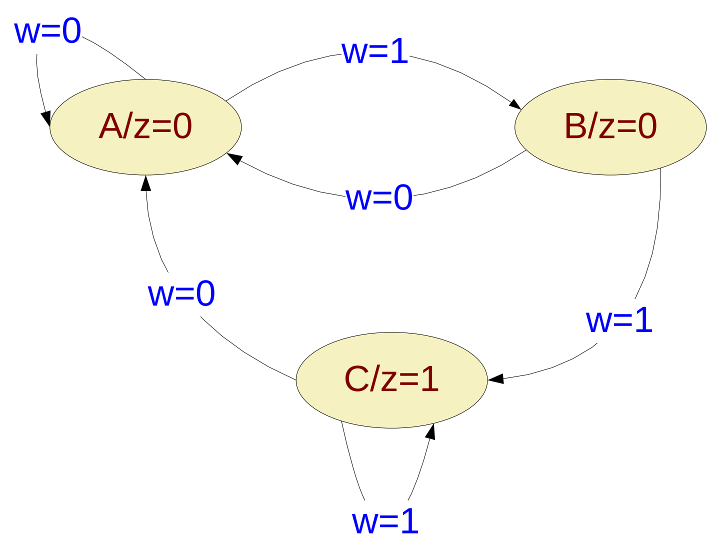
```
entity simple is
port (    clock, resetn, w: in stdlogic;
          z:out std logic);
end simple;

architecture behavior of simple is
type state_type is (a, b, c);
signal y: state_type ;
begin
   process (resetn, clock)
   begin
      if resetn = ‘0’ then
         y <= a;
      elsif (clock’event and clock = ‘l’) then
         case y is
            when a =>
              if w=’0’ then
                   y <= a;
              else
                   y <= b;
              end if;
```

Type declaration

Signal definition using a defined type

Case statement declaration

Item under test



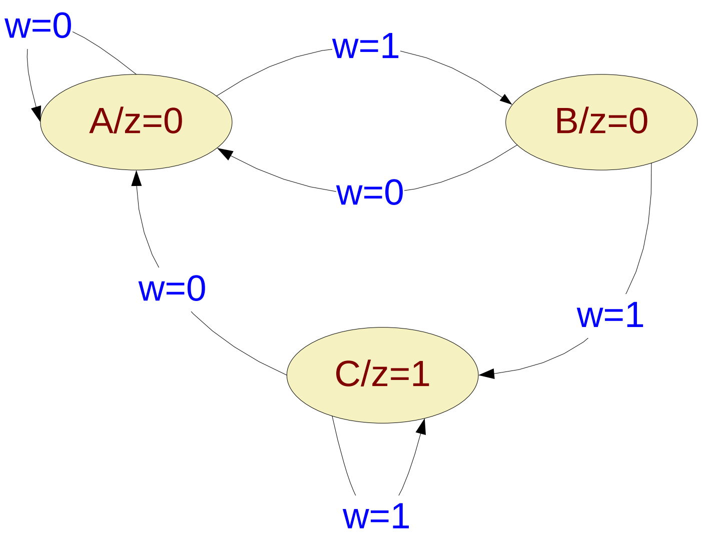
Clock Cycle	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
W:	0	1	0	1	1	0	1	1	1	0	1
Z:	0	0	0	0	0	1	0	0	1	1	0

A Secuence Detector (continued)

```
          when b =>
            if w='0' then
                 y <= a;
            else
                 y <= c;
            end if;
          when c =>
            if w='0'then
                 y <= a;
            else
                 y <= c;
            end if;
       end case;
    end if;
  end process;
  z <= '1' when y=c else '0';
end behavior;
```

Case statement declaration

Conditional signal assignment

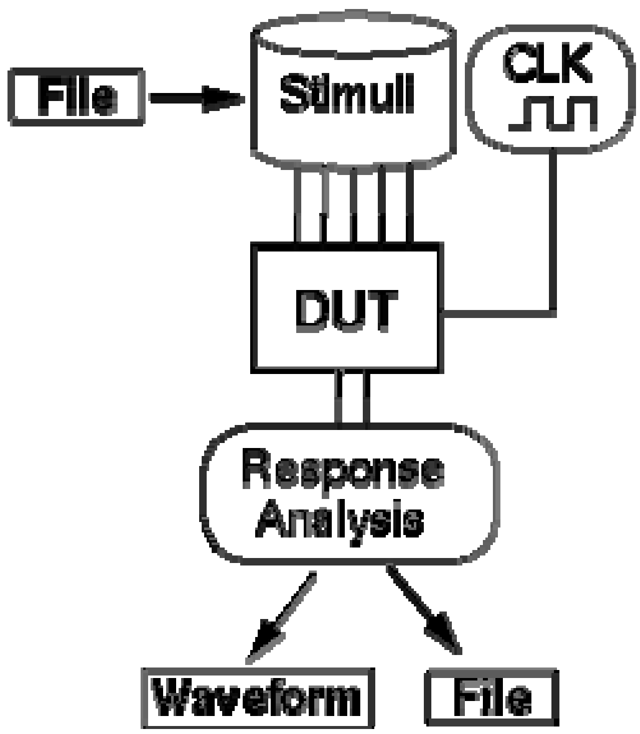


Clock Cycle	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
W:	0	1	0	1	1	0	1	1	1	0	1
Z:	0	0	0	0	0	1	0	0	1	1	0

Testbenches

- Stimuli transmitter to DUT (testvectors)
- Needs not to be synthesizable
- No ports to the outside
- Environment for DUT
- Verification and validation of the design
- Several output methods
- Several input methods

Example of a testbench



Example Testbench

```
entity TB_TEST is
  end TB_TEST;
architecture BEH of TB_TEST is
    -- component declaration of the DUT
    -- internal signal definition
  begin
    -- component instantiation of the DUT
    -- clock generation
    -- stimuli generation
  end BEH;
```

Example Testbench

```
entity TB_TEST is
        end TB_TEST;

        architecture BEH of TB_TEST is
          component TEST
            port(CLK      : in std_logic;
                 RESET  : in std_logic;
                 A          : in integer range 0 to 15;
                 B          : in std_logic;
                 C          : out integer range 0 to 15);
        end component;

          constant PERIOD  : time := 10 ns;
          signal W_CLK      : std_logic := '0';
          signal W_A, W_C   : integer range 0 to 15;
          signal W_B            : std_logic;
          signal W_RESET   : std_logic;
        begin
          DUT : TEST
            port map(CLK      => W_CLK,
                     RESET  => W_RESET,
                     A          => W_A,
                     B          => W_B,
                     C          => W_C);
          . . .
```

Questions?