

Introduction to VHDL

- Newer and most popular HDL (of two) for industry designs. (VHDL 1987)
- Created in 1984 to specify and simulate circuits (portmanteau of “Verify Logic”)
- Is an IEEE specified language (IEEE 1364 released in 1995)
- Used to specify physical circuit behavior, not for procedural algorithms
- Is text-based & platform independent - any editor can be used
- Can define high-level behavioral or low-level structural models
- Can be simulated pre-synthesis without timing data (“logical simulation”)
- Can be synthesized to define a physical circuit
- Can be simulated post-synthesis with accurate timing (“physical simulation”)
- Dozens of CAD Verilog tools: Cadence, Mentor, Xilinx, Aldec, etc.

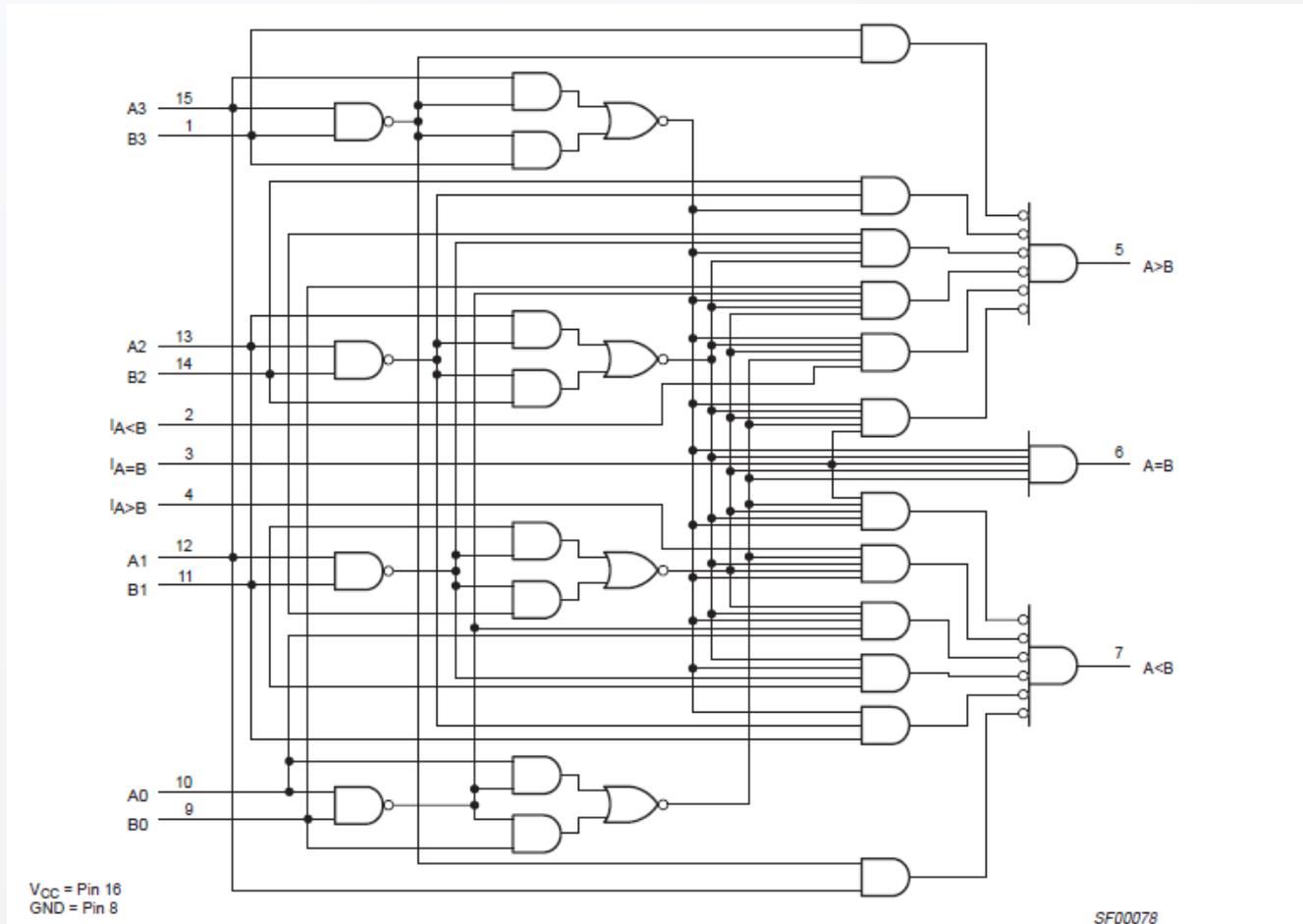
Behavioral vs. Structural Models

- “Behavioral” Verilog defines outputs as functions of inputs, without describing any circuit components or modules that might be used in constructing the circuit.
- “Structural” Verilog is a form of netlist, defining a circuit as a collection of subordinate components or modules and their interconnecting wires.
- Behavioral descriptions are more abstract, higher-level, quicker and easier to write, easier for others to understand and follow, and largely self documenting.
- Structural descriptions are often used when existing IP blocks can be reused. They are easier to debug, easier to analyze (for timing and area), and can be easier to optimize.

Most designers write behavioral Verilog, except when reusing existing IP.

Behavioral vs. Structural Models

Example: 4-bit magnitude comparator



GT <= '1' when A > B else '0';
 LT <= '1' when A < B else '0';
 EQ <= A nor B;

Design Units

- **Entity:** “Bounding box” for components
- **Architecture:** Behavioral description
- **Package:** Organization tool
- **Configuration:** Organization tool
- **File:** Good for large simulation I/O

Libraries

- **Work:** Defines project, stores all current work
- **External libraries:** Stores previous work

In a VHDL design session...

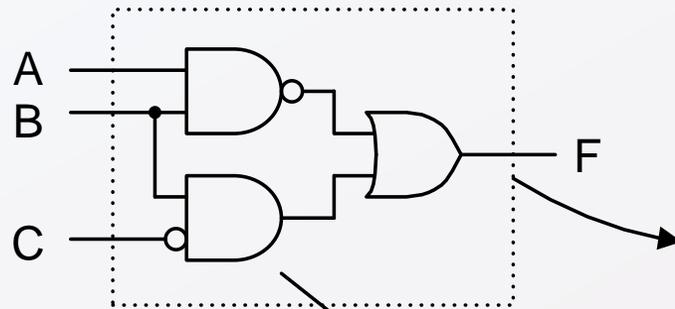
Valid source files (design units) make a project

All source files must be stored in valid library

- Tools create and use “work” library automatically
- All other libraries must be listed in source file

Completed source files can be moved from work library to external library at later time

A First Look: Entity/Architecture



Bounding box
represented by entity
statement; behavior by
architecture statement

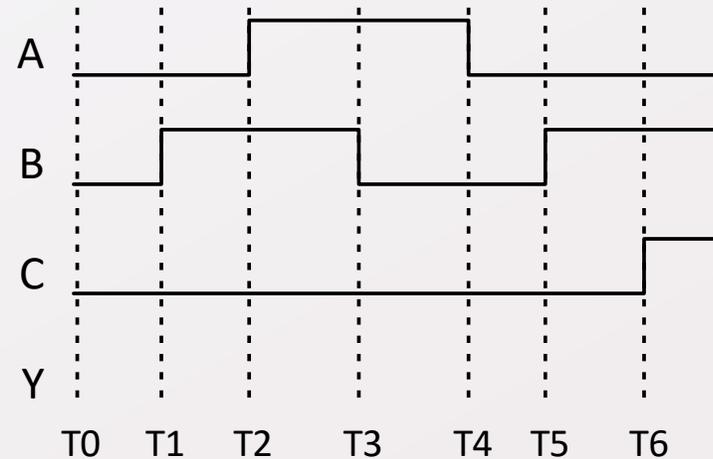
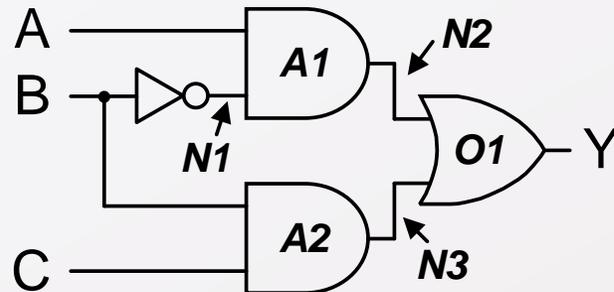
```
library ieee;  
use ieee.std_logic_1164.all;
```

```
entity Example is  
  port (A,B,C   : in  STD_LOGIC;  
        Y       : out STD_LOGIC);  
end Example
```

```
architecture behavioral of Example is  
begin  
  Y <= (not (A and B) or (B and not C));  
end behavioral;
```

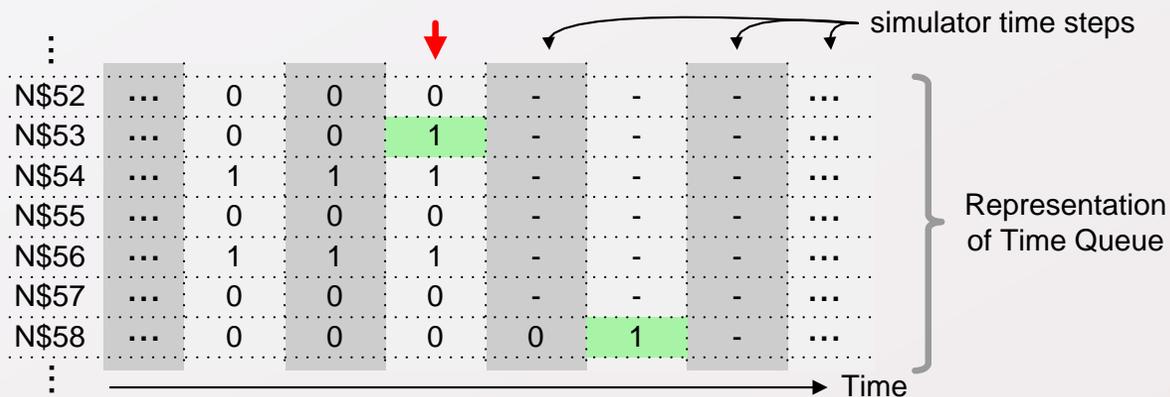
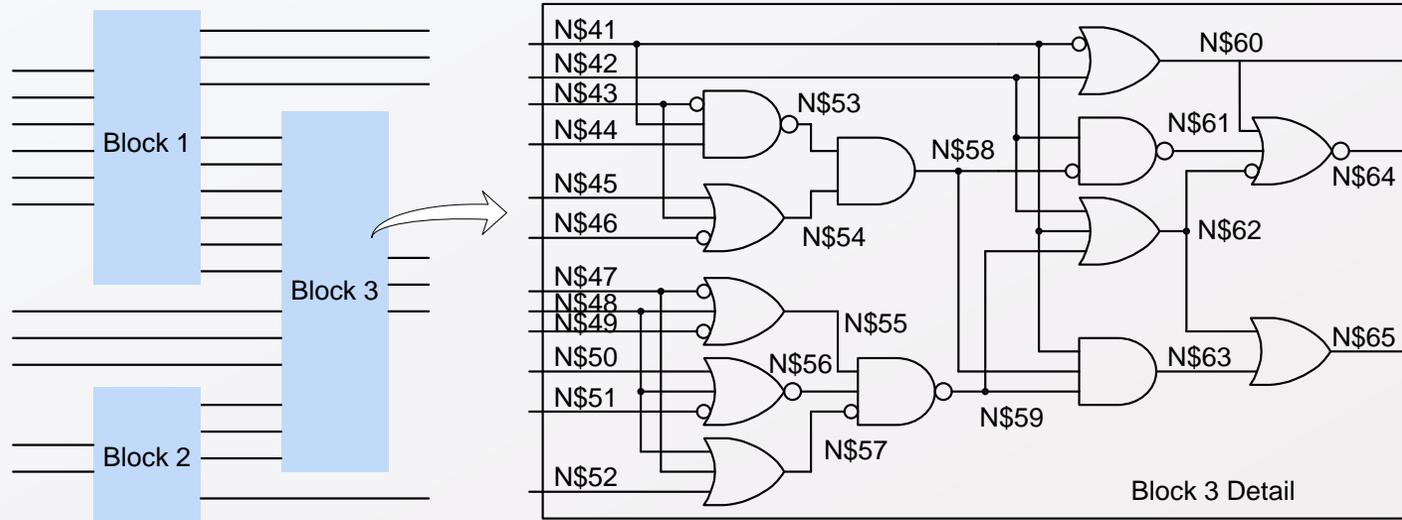
Basic Concept: Sequential vs. Concurrent Models

- A sequential processing algorithm defines a set of steps that are taken in a specific order
- Concurrent processing steps occur whenever new input data is available, with no implied sequence relationship between separate concurrent processes
- Consider a simulation of a 2-input mux: At what time(s) should gate A1 be simulated? What about gate O1?



- Computer time vs. Simulation time

Modeling Physical Time



How the Simulator Works

Checks time queue

Finds current changes, evaluates instance

Schedules new value

Some basic “models” supplied: AND, OR, NOT, etc, all others up to user

Delta delay vs. user supplied delays

Signal Assignments

`<=` is assignment operator; schedules a change

Implicit Process

```
Y <= '1';      Y <= A;
```

Internal functions: AND, OR, NOT, etc.

```
Y <= A and B;
```

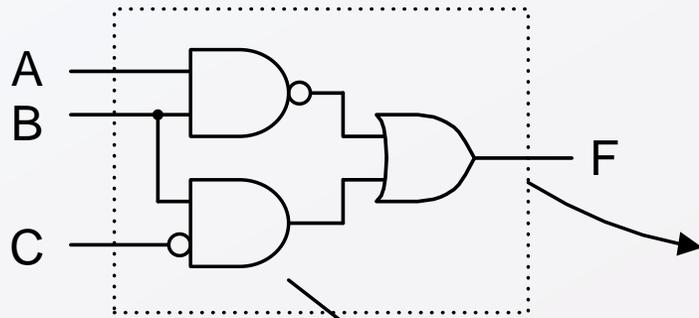
“and” only evaluated if A or B change state; Y is scheduled for a later change

Scheduled

General Source File Structure

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity circuit_name is  
    port (list of inputs, outputs and type);  
end circuit_name;  
  
architecture arch_name of circuit_name is  
begin  
    (statements defining circuit go here);  
end arch_name;
```

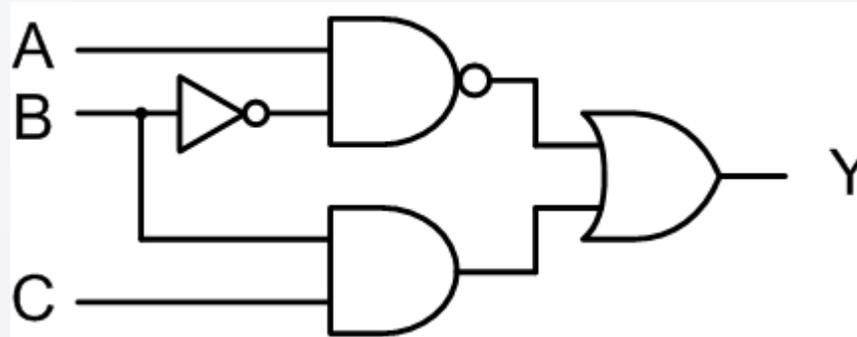
Entity/Architecture



Bounding box
represented by entity
statement; behavior by
architecture statement

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity Example is  
    port (A,B,C    : in  STD_LOGIC;  
          Y        : out STD_LOGIC);  
end Example  
  
architecture behavioral of Example is  
begin  
    Y <= (not (A and B) or (B and not C));  
end behavioral;
```

Entity



```
entity my_circuit is  
    port (A, B, C : in  STD_LOGIC;  
          Y      : out STD_LOGIC);  
end my_circuit;
```

More about **types...**

Predefined (internal) types:

bit, boolean, character, integer, severity level, string, time

Good for modeling, not for synthesis

Need to model wires: STD_LOGIC

```
PACKAGE std_logic_1164 IS
-----
-- logic state system (unresolved)
-----
TYPE std_ulogic IS ( 'U', -- Uninitialized
                    'X', -- Forcing Unknown
                    '0', -- Forcing 0
                    '1', -- Forcing 1
                    'Z', -- High Impedance
                    'W', -- Weak Unknown
                    'L', -- Weak 0
                    'H', -- Weak 1
                    '-' -- Don't care
                    );
```

Functions with std_logic type

(from std_logic_1164)

```

CONSTANT and_table : stdlogic_table := (
--      -----
--      |  U   X   0   1   Z   W   L   H   -   |  |
--      -----
--      ( 'U', 'U', '0', 'U', 'U', 'U', '0', 'U', 'U' ), -- | U |
--      ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- | X |
--      ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ), -- | 0 |
--      ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ), -- | 1 |
--      ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- | Z |
--      ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- | W |
--      ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ), -- | L |
--      ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ), -- | H |
--      ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ) -- | - |
);

```

```

FUNCTION "and" ( l : std_ulogic; r : std_ulogic ) RETURN UX01 IS
-- pragma built_in SYN_AND
-- pragma subpgm_id 184
BEGIN
--synopsys synthesis_off
    RETURN (and_table(l, r));
--synopsys synthesis_on
END "and";

```

Architecture Statement

architecture *arch_name* **of** *circuit_name* **is**

Declaration area – declared items can go in package

begin

Concurrent area – assignment statements go here

end *arch_name*;

Architecture Statement

```
architecture arch_name of circuit_name is
```

```
signal a, b, c : in std_logic
```

```
type my_type is (type1, type2, type3)
```

```
signal d, e : my_type
```

```
begin
```

```
    a <= b or c;
```

```
    d <= type 1;
```

```
    e <= d;
```

```
end arch_name;
```

Concurrent Statements;
order does not matter

Busses in VHDL

A, B: **STD_LOGIC_VECTOR** (7 downto 0);

C, D: **STD_LOGIC_VECTOR** (0 to 7);

Bus assignments: Always left-most to left-most

A <= B; -- works as desired

A <= C; -- flips values

A <= B(6 downto 0) & B(7) – rotate left

A <= B(6 downto 0) & '0' – shift left

Signal Assignments

Simple: $Y \leq A \text{ or } B;$

Selected (example)

```
sel : STD_LOGIC_VECTOR (1 downto 0);
```

```
with sel select
```

```
  y <= a when "00";  
    b when "01";  
    c when "10";  
    d when others;
```

Conditional (example)

```
sel : STD_LOGIC_VECTOR (1 downto 0);
```

```
  y <= a when (sel = "00") else  
    b when (sel = "01") else  
    c when (sel = "10") else  
    d;
```

Process Statements: Explicit

```
process (x, y, z) begin
```

Sequential statements;
order can matter

```
end process
```

Sequential Statements: If/then

```
if (condition) then a <=b;  
    else a <= c;  
endif;
```

```
if (condition) then a <=b;  
    elsif (condition 2) a <= c;  
endif
```

Sequential Statement: case

```
case (var) is
  when a =>
    x <= y;
  when b =>
    x <= z;
  when c =>
    x <= w;
end case;
```

Memory

Inferred in underspecified sequential statement

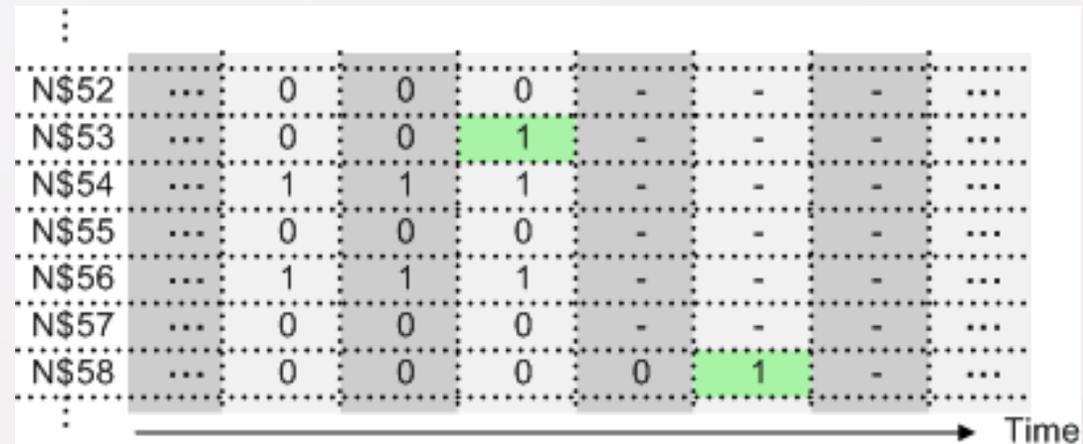
But first, we need *attributes*

x'event

a'active

y'last_event

b'quiet



More than 40 available...

Memory inferred in process statement

Example: D flip-flop with asynch reset

```
process (clk, rst) begin
  if rst = '1' then Q <= 0;
    elsif (CLK'event and CLK='1') then Q <= D;
  end if;
end process;
```

Note that the *IF* statement is underspecified

What if rst = '0'? Clk = '0'?

Some Code Examples...

Muxes

4:1 selected

```
entity mux_select is
  port ( I3, I2, I1, I0: in std_logic;
        sel : in std_logic_vector (1 downto 0);
        Y   : out std_logic);
end mux_select;
```

```
architecture behavioral of mux_select is
Begin
  with sel select
    Y <= I0 when "00";
        I1 when "01";
        I2 when "10";
        I3 when others;
end behavioral;
```

4:1 Bus selected

```
entity busmux_select is
  port ( I3, I2, I1, I0: in std_logic_vector (7 downto 0);
        sel : in std_logic_vector (1 downto 0);
        Y   : out std_logic_vector (7 downto 0));
end busmux_select;
```

```
architecture behavioral of busmux_select is
Begin
  with sel select
    Y <= I0 when "00";
        I1 when "01";
        I2 when "10";
        I3 when others;
end behavioral;
```

4:1 Bus Mux Conditional

```
entity mux_cond is  
  port ( A, B, C : in std_logic_vector (7 downto 0);  
         Sel      : in std_logic_vector (1 downto 0);  
         Y        : out std_logic_vector (7 downto 0));  
end mux_cond;  
  
architecture behavioral of mux_cond is  
  Begin  
    Y <= (A or not C) when (Sel = "00") else  
         (A xor B) when   (Sel = "01") else  
         not A when       (Sel = "10") else  
         (B nand C);  
end behavioral;
```

Decoders

```
entity decoder is
  port ( in:      in std_logic_vector (1 downto 0);
        Y:      out std_logic_vector (7 downto 0));
end decoder;
```

```
architecture behavioral of decoder is
```

```
Begin
```

```
  with in select
```

```
  Y <=  "0001" when "00";
        "0010" when "01";
        "0100" when "10";
        "1000" when others;
```

```
end behavioral;
```

```
entity seven_seg_dec is
```

```
  port (bin: in STD_LOGIC_VECTOR (3 downto 0);
        segout : out STD_LOGIC_VECTOR (6 downto 0));
end seven_seg_dec;
```

```
architecture behavioral of seven_seg_dec is
```

```
begin
```

```
  with bin select
```

```
  segout <=  "1111110" when "0000";
             "0110000" when "0001";
             .
             .
             "0000001" when others;
```

```
end behavioral;
```

Shifter

```
entity my_shift is
```

```
  port ( din:      in std_logic_vector (7 downto 0);  
        r, d, en: in std_logic;  
        dout:     out std_logic_vector (7 downto 0));
```

```
end my_shift;
```

```
architecture my_shift_arch of my_shift is
```

```
begin
```

```
  dout <= "00000000" when en = '0' else  
    din(6 downto 0) & din(7) when (r = '1' and d = '0') else  
    din(0) & din(7 downto 1) when (r = '1' and d = '1') else  
    din(6 downto 0) & '0' when (r = '0' and d = '0') else  
    '0' & din(7 downto 1);
```

```
end my_shift_arch;
```

Arithmetics

ALU

```
entity ALU is
  port ( A, B : in std_logic_vector (7 downto 0);
        Sel  : in std_logic_vector (1 downto 0);
        Y    : out std_logic_vector (7 downto 0));
end ALU;

architecture behavioral of ALU is
  Begin
    With sel select
      Y <= (A + B) when "00",
          (A + "00000001") when "01",
          (A or B) when "10",
          (A and B) when others;
end behavioral;
```

Memory Devices

```

entity DFF is
  port (D, clk, rst, ce : in std_logic;
        Q      : out std_logic);
end DFF;
architecture behavioral of DFF is
begin
  process(clk, rst)
  begin
    if rst = '1' then Q <= '0';
      elsif (clk'event and clk='1')
        then if ce = '1' then Q <= D;
          end if;
        end if;
      end process;
    end dff_arch;
  
```

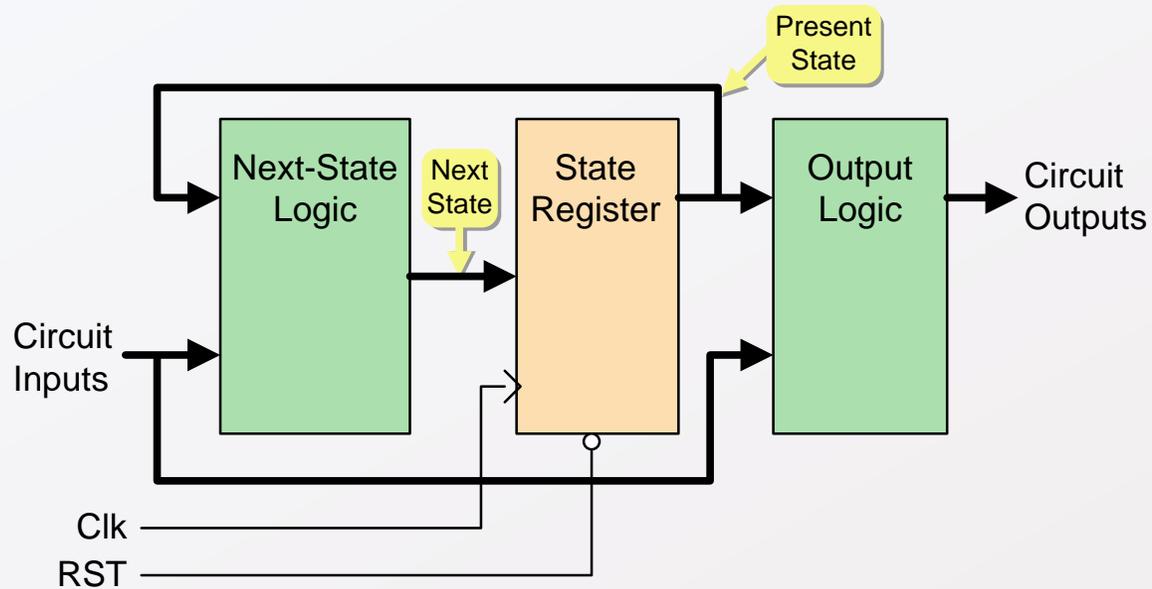
```

entity DFF is
  port (D, clk, rst : in std_logic;
        Q      : out std_logic);
end DFF;
architecture behavioral of DFF is
begin
  process(clk, rst)
  begin
    if (clk'event and clk='1') then
      if rst = '1' then Q <= '0';
        else Q<=D;
      end if;
    end process;
  end dff_arch;
  
```

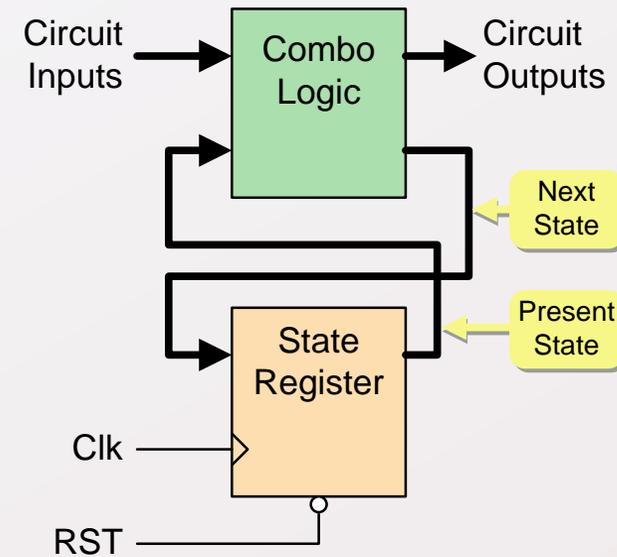
```

entity DFFReg8 is
  port (D      : in std_logic_vector (7 downto 0);
        clk, rst : in std_logic;
        Q      : out std_logic_vector (7 downto 0));
end DFFReg8;
architecture behavioral of DFFReg8 is
begin
  process(clk,rst)
  begin
    if rst = '1' then Q <= '0';
      elsif (clk'event and clk='1') then Q <= D;
    end if;
  end process;
end dff_arch;
  
```

State Machines



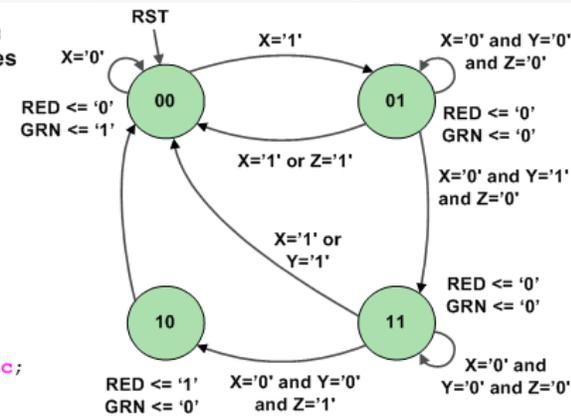
Mealy Model State Machine



Combined Model State Machine

State Machines

Behavioral VHDL example for encoding a state machine with user-defined state codes



```

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity fsm_ex is
  port (CLK, RST, X, Y, Z: in std_logic;
        RED, GRN : out std_logic);
end fsm_ex;

architecture state_codes of fsm_ex is

  signal ps, ns : std_logic_vector (1 downto 0);

Begin

  comb_logic: process (ps, x, y, z) begin
    case ps is
      when "00" =>
        RED <= '0'; GRN <= '1';
        if (X='1') then ns <= "01";
        else ns <= "00";
        end if;
      when "01" =>
        RED <= '0'; GRN <= '0'
        if (X='0' and Y='1' and Z='0') then ns <= "11";
        elsif (X='1' or Z='1') then ns <= "00";
        else ns <= "01";
        end if;
      when "11" =>
        RED <= '0'; GRN <= '0';
        if (X='0' and Y='0' and Z='1') then ns <= "10";
        elsif (X='1' or Y='1') then ns <= "00";
        else ns <= "11";
        end if;
      when "10" =>
        RED <= '1'; GRN <= '0';
        ns <= "00";
    end case;
  end process;

  state_reg: process (clk, rst) begin
    if rst = '1' then ps <= "00";
    elsif (CLK'event and CLK='1') then ps <= ns;
    end if;
  end process;

end state_codes;

```

Behavioral VHDL example for encoding a state machine with synthesized state codes

```

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity fsm_ex is
  port (CLK, RST, X, Y, Z: in std_logic;
        RED, GRN : out std_logic);
end fsm_ex;

```

```

architecture enumerated of fsm_ex is

```

```

  type sreg_type is (init, xdet, ydet, zdet);
  signal ps, ns : sreg_type;

```

```

Begin

```

```

  comb_logic: process (ps, x, y, z) begin
    case ps is
      when init =>
        RED <= '0'; GRN <= '1';
        if (X='1') then ns <= xdet;
        else ns <= init;
        end if;
      when xdet =>
        RED <= '0'; GRN <= '0';
        if (X='0' and Y='1' and Z='0') then ns <= ydet;
        elsif (X='1' or Z='1') then ns <= init;
        else ns <= xdet;
        end if;
      when ydet =>
        RED <= '0'; GRN <= '0';
        if (X='0' and Y='0' and Z='1') then ns <= zdet;
        elsif (X='1' or Y='1') then ns <= init;
        else ns <= ydet;
        end if;
      when zdet =>
        RED <= '1'; GRN <= '0';
        ns <= init;
    end case;
  end process;

```

```

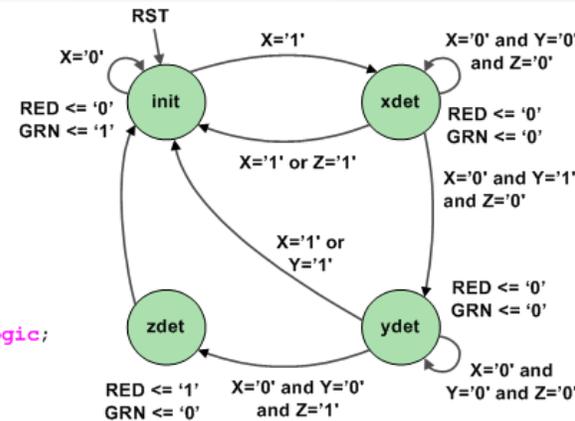
  state_reg: process (clk, rst) begin
    if rst='1' then ps <= init;
    elsif (CLK'event and CLK='1') then ps <= ns;
    end if;
  end process;

```

```

end enumerated;

```



Structural VHDL

Structural VHDL is essentially a netlist language. The engineer must define:

- all entities/components
- all internal signals
- all interconnects

Component Declarations

Component Instantiations

Port Map statements

Structural Code Example

```
entity RCA is
  port ( A, B   : in std_logic_vector (3 downto 0);
        S     : out std_logic_vector (3 downto 0);
        Cout  : out std_logic);
end RCA;
```

```
architecture structural of RCA is
```

```
  component HA
    port (A, B   : in std_logic;
          S, Cout : out std_logic);
  end component;
```

```
  component FA
    port (A, B   : in std_logic;
          S, Cout : out std_logic);
  end component;
```

```
  signal CO : std_logic_vector (3 downto 0);
```

```
Begin
```

```
  C0: HA port map (A=>A(0), B=>B(0), S=>S(0), Cout=>CO(0));
  C1: FA port map (A=>A(1), B=>B(1), Cin=>CO(0), S=>S(1), Cout=>CO(1));
  C2: FA port map (A=>A(2), B=>B(2), Cin=>CO(1), S=>S(2), Cout=>CO(2));
  C3: FA port map (A=>A(3), B=>B(3), Cin=>CO(2), S=>S(3), Cout=>CO(3));
```

```
end behavioral;
```

Components and signals are declared in the *Declaration Area* between the Architecture and Begin statements

Components can be instantiated anywhere within the Architecture statement

Behavioral vs. Structural Designs

Behavioral Designs are abstract and high-level

Quicker, easier, no efficiency penalty

Easier to read, maintain, reuse

BUT....

Hard to analyze at a detailed level

Hard to incrementally modify

Structural Designs are detailed and low-level

Easy to simulate, analyze, and locate problems

Easy to “tweak” small parts of the design

Easy to import/export IP

BUT....

Harder to create, document, and reuse

Usually not as efficient

Design for Synthesis

Many references in Xilinx support area

For complex circuits, use coregen, XMD, Xilinx IP blocks, and other tools to generate IP where ever possible

Infer rather than instantiate Xilinx primitives

Pipeline complex designs

Experiment!

Test Benches

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
```

} Standard file header containing library and package definitions

```
entity lab5test_bench is
end lab5test_bench;
```

} An “empty” entity statement required for all test bench source files. The entity name can be any legal string.

```
architecture test of lab5test_bench is
```

```
    component ex1
        port( a, b, c : in std_logic;
              y : out std_logic);
    end component;
```

} The entity under test (EUT) must be declared as a component. The port must exactly match the port statement from the EUT.

```
    signal a, b, c, y : std_logic;
```

} All signals that attach to entity port pins must be declared as signals.

```
begin
```

```
    EUT: ex1 port map(a => a,
                     b => b,
                     c => c,
                     y => y);
```

} The EUT must be instantiated. The port map statement maps the declared signals to the port pins of the EUT. It is common to use matching signal and port pin names.

```
    process begin
```

```
        a <= '0';
        b <= '0';
        c <= '0';
        wait for 100 ns;
        a <= '1';
        wait for 100 ns;
        b <= '1';
        wait for 100 ns;
        c <= '1';
        wait for 100 ns;
        a <= '0';
        wait for 100 ns;
        b <= '0';
        wait for 100 ns;
        c <= '0';
```

} Statements to define input stimulus are placed in a process statement so that the “wait” statement can be used to control the passage of time.

```
    end process;
end test;
```