



LUND
UNIVERSITY

EITF35: Introduction to Structured VLSI Design

Part 1.2.2: VHDL-1

Liang Liu
liang.liu@eit.lth.se



Outline

□ VHDL Background

- What is VHDL?
- Why VHDL?

□ Basic VHDL Component

- An example

□ FSM Design with VHDL

□ Simulation & TestBench



What is VHDL?

Very high speed integrated circuit

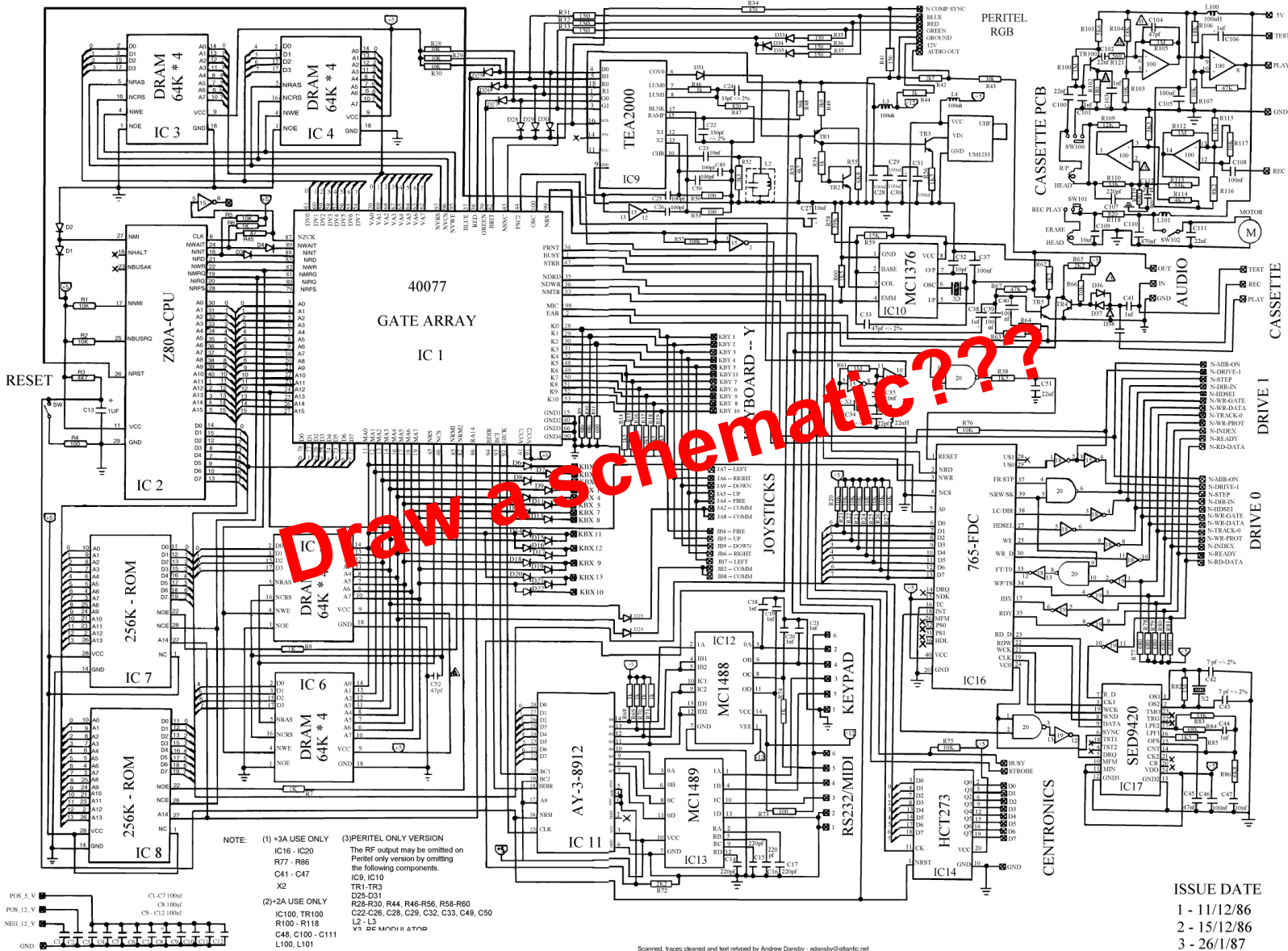
Hardware

Description

Language



Why use an HDL?



Scanned, traces cleaned and text relayed by Andrew Danzky - andanzky@att.net



ISSUE DATE
 1 - 11/12/86
 2 - 15/12/86
 3 - 26/1/87

Why use an HDL?

□ Advantages of VHDL

- Supports easy modeling of various abstraction levels
 - *from the gate level to the system level*
- Supported by all CAD Tools
- Technology independent
 - **easy to move VHDL code between different commercial platforms**
- Used by industry and academia worldwide – Specially in Europe

It will dramatically improve your productivity



VHDL vs. Verilog

- Equivalent for RTL modeling
- Both are industrial standards and are supported by most software tools
- **VHDL** more popular in Europe/**Verilog** in US & Asian
- **VHDL** is more **flexible** in syntax and usage
- For **high level** behavioral modeling, **VHDL** is better
 - Verilog does not have ability to define new data types
- **Verilog** has built-in **gate level** and **transistor level** primitives
 - Verilog is better than VHDL at below the RTL level.

VHDL

```
library IEEE;
use IEEE.STD_Logic_1164, all;

entity LATCH_IF_ELSEIF is
  port (En1, En2, En3, A1, A2, A3: in std_logic;
        Y: out std_logic);
end entity LATCH_IF_ELSEIF;

architecture RTL of LATCH_IF_ELSEIF is
begin
  process (En1, En2, En3, A1, A2, A3)
  begin
    if (En1 = '1') then
      Y <= A1;
    elseif (En2 = '1') then
      Y <= A2;
    elseif (En3 = '1') then
      Y <= A3;
    end if;
  end process;
end architecture RTL;
```

Verilog

```
module LATCH_IF_ELSEIF (En1, En2, En3, A1, A2, A3, Y);
  input En1, En2, En3, A1, A2, A3;
  output Y;

  reg Y;

  always @(En1 or En2 or En3 or A1 or A2 or A3)
    if (En1 == 1)
      Y = A1;
    else if (En2 == 1)
      Y = A2;
    else if (En3 == 1)
      Y = A3;
end module
```



Outline

□ VHDL Background

- What is VHDL?
- Why VHDL?

□ **Basic VHDL Component**

- An example

□ FSM Design with VHDL

□ Simulation & TestBench



Simple Tutorial to VHDL

□ For all **C/Matlab** people –

“Forget” everything you know!

□ Do NOT write if you don't know which kind of **HARDWARE** will be generated



Sample Design Flow

□ Design Target (Spec.)

- Design a single bit half adder with carry and enable
- Performance?

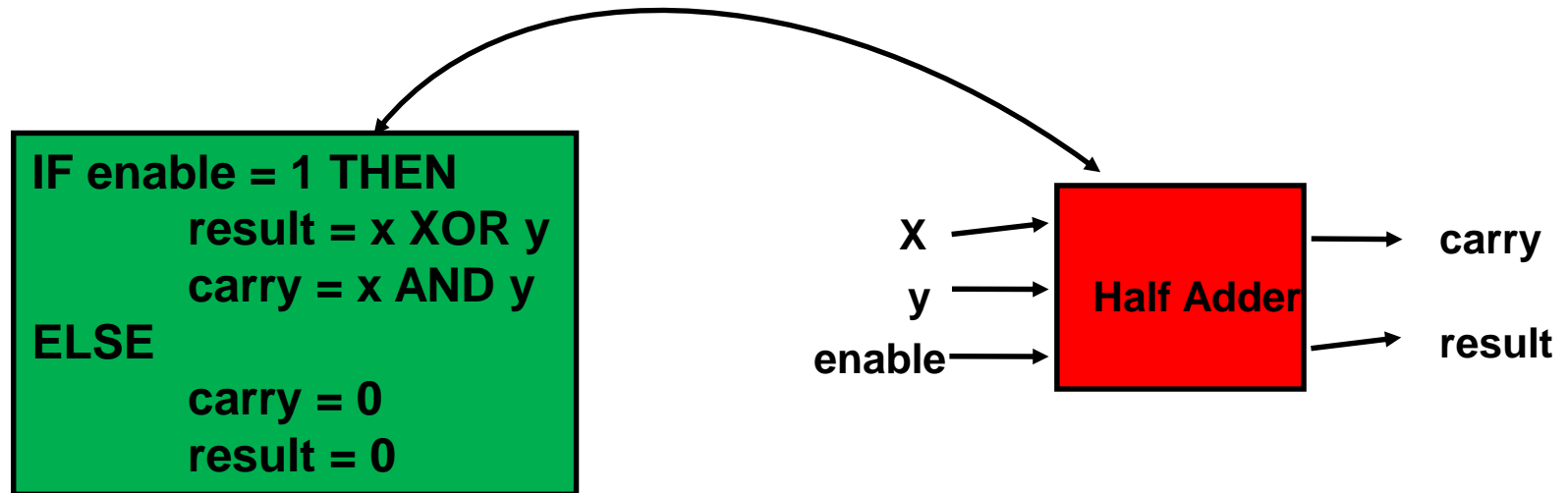
□ Detailed Functionality

- Passes results only on enable high and zero on enable low
- Result gets x plus y
- Carry gets any carry of x plus y



Step1: Behavioral Design

- Starting with an algorithm, a high level description of the adder is created.

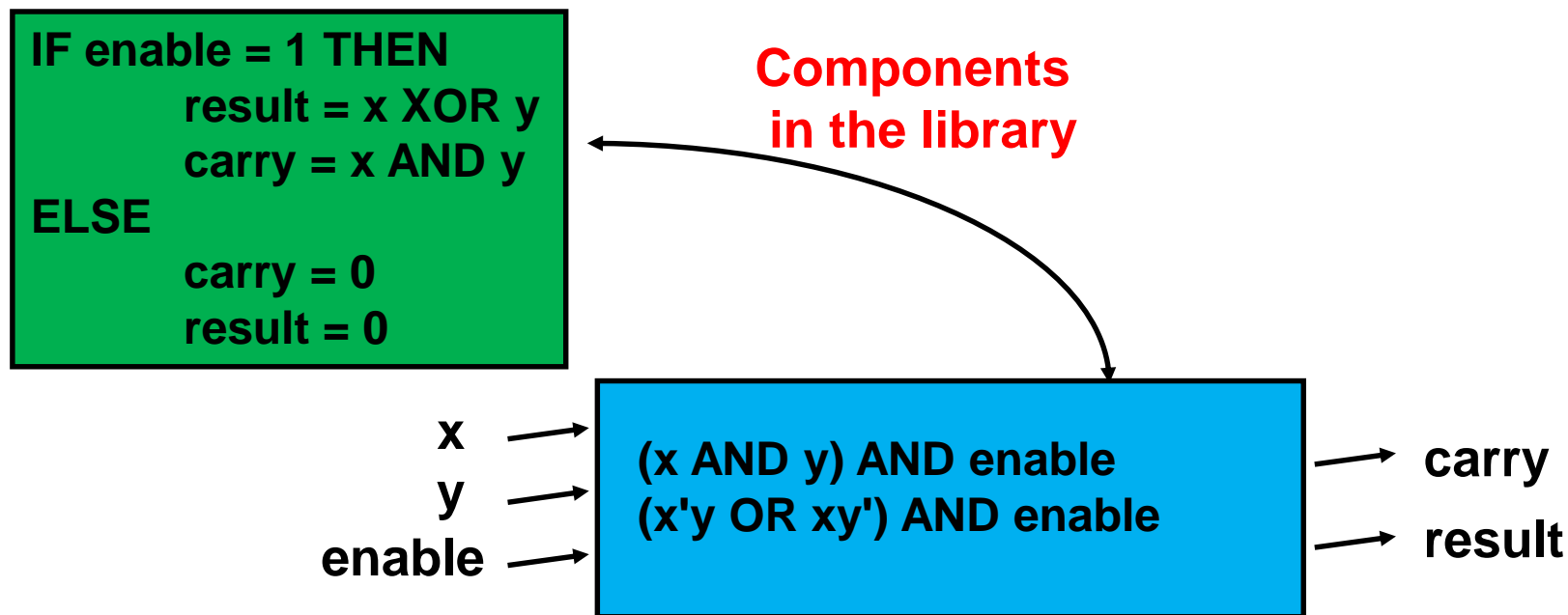


- The model can (MUST) be simulated at this high level description to verify correct understanding of the problem, e.g., Matlab



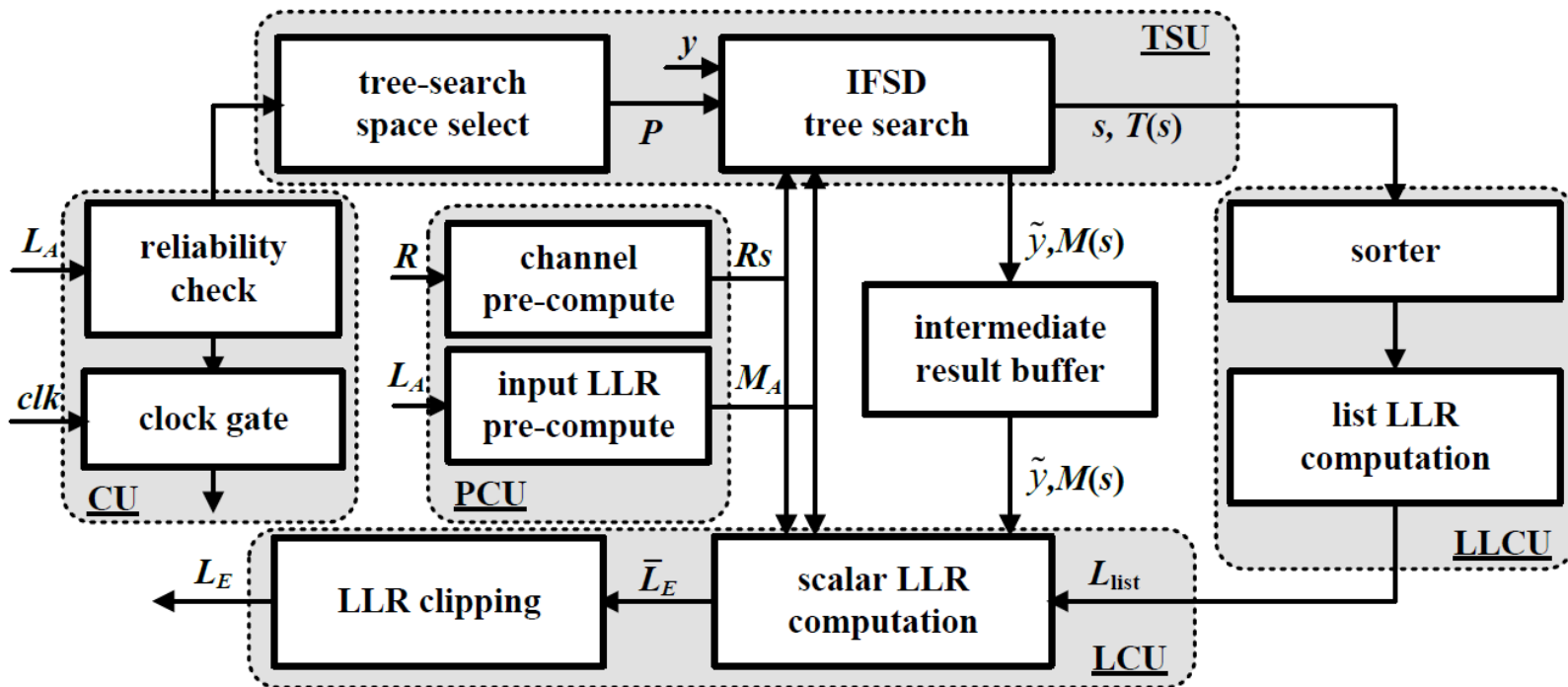
Step2: Circuit Design

□ A structural description is created at the “module” level, circuit design



Step2: Circuit Design

- A structural description is created at the “module” level, circuit design
- These “modules” should be pulled from a **library** of parts



Suggestion: Always draw block diagram before coding



Step3: VHDL Coding

Entity Declaration

- An entity declaration describes the *interface* of the component
- PORT clause indicates input and output ports
- An entity can be thought of as a *symbol* for a component

```
library IEEE;  
use IEEE.std_logic_1164.all;  
ENTITY half_adder IS  
    PORT (x, y, enable: IN bit;  
          carry, result: OUT bit);  
END half_adder;  
ARCHITECTURE data_flow OF half_adder IS  
BEGIN  
    carry <= (x AND y) AND enable;  
    result <= (x XOR y) AND enable;  
END data_flow;
```



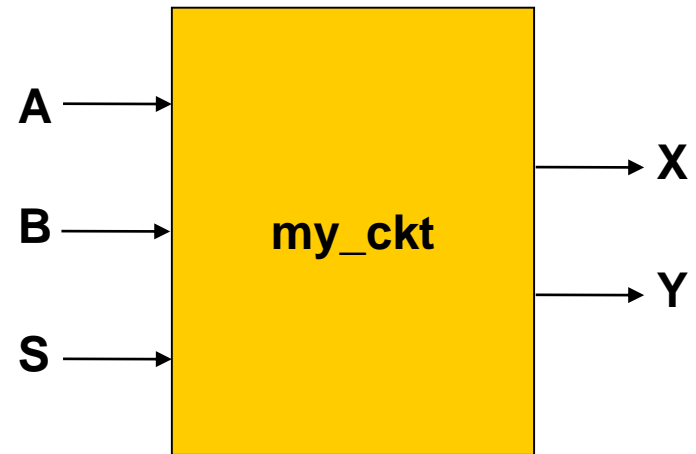
See Packages in IEEE library:

<http://www.csee.umbc.edu/portal/help/VHDL/stdpkg.html>



VHDL Entity

```
• entity my_ckt is  
  port (  
    A: in bit;  
    B: in bit;  
    S: in bit;  
    X: out bit;  
    Y: out bit  
  );  
end my_ckt;
```



- Name of the circuit
- User-defined
- Filename same as circuit name
- Example.
 - Circuit name: `my_ckt`
 - Filename: `my_ckt.vhd`



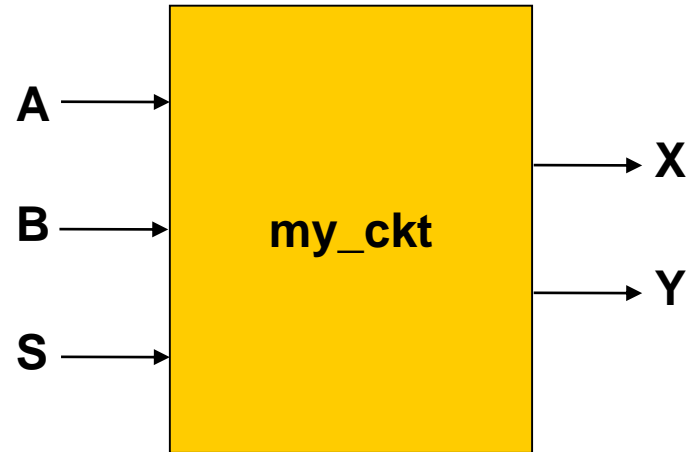
VHDL Entity

- entity `my_ckt` is
port (

```
A: in bit;  
B: in bit;  
S: in bit;  
X: out bit;  
Y: out bit
```

```
);  
end entity my_ckt;
```

Port names or
Signal names
`i_clk_r`, `i_rst_n`



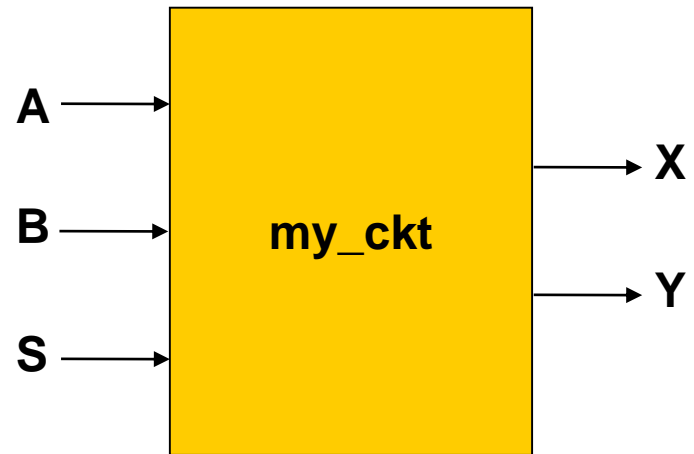
VHDL Entity

- `entity my_ckt is`
`port (`

```
A: in bit;  
B: in bit;  
S: in bit;  
X: out bit;  
Y: out bit
```

```
);
```

```
end my_ckt;
```



Direction of port

3 main types:

- **in:** Input
- **out:** Output
- **inout:** Bidirectional



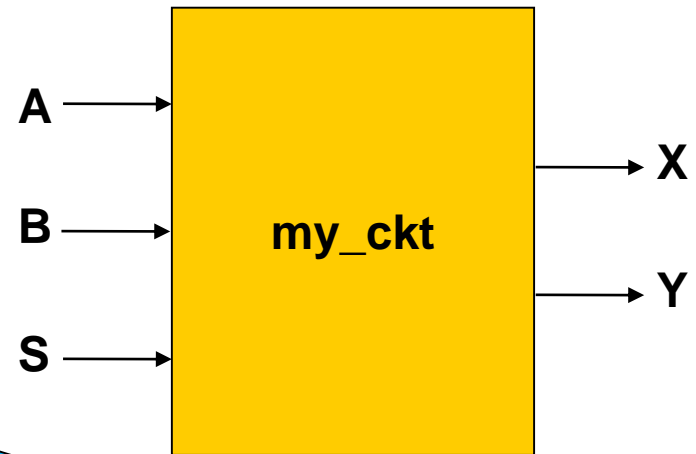
VHDL Entity

• `entity my_ckt is`
`port (`

```
A: in bit;  
B: in bit;  
S: in bit;  
X: out bit;  
Y: out bit
```

`);`

`end my_ckt;`



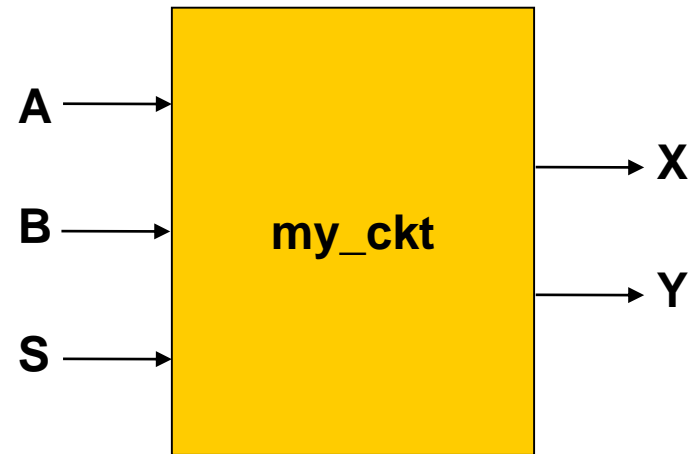
Datatypes:

- bit, bit_vector
- integer
- **std_logic,**
std_logic_vector
- User-defined



VHDL Entity

```
• entity my_ckt is
  port (
    A: in bit;
    B: in bit;
    S: in bit;
    X: out bit;
    Y: out bit;
  );
end my_ckt;
```



Note the absence of semicolon “;” at the end of the last signal and the presence at the end of the closing bracket



VHDL Coding

□ Architecture

- A pattern, a template, a way of doing it
- Architecture declarations describe the **operation of the component**
- **Many architectures** may exist for one entity

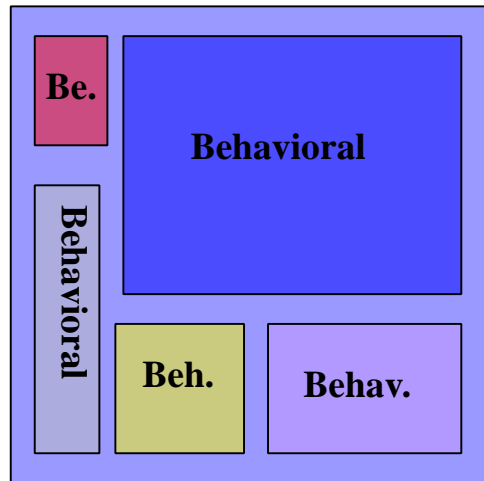
```
library IEEE;  
use IEEE.std_logic_1164.all;  
ENTITY half_adder IS  
    PORT (x, y, enable: IN bit;  
          carry, result: OUT bit);  
END half_adder;  
ARCHITECTURE data_flow OF half_adder IS  
BEGIN  
    carry <= (x AND y) AND enable;  
    result <= (x XOR y) AND enable;  
END data_flow;
```



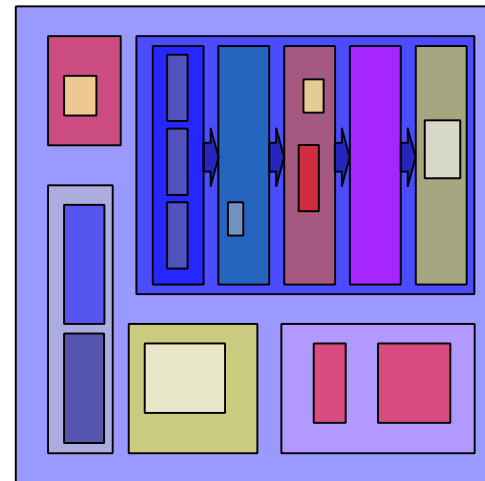
Architecture

□ Basically three types of architectures:

- Dataflow: how is the data transmitted from input to output
- Behavioral: using sequential processes
- Structural: top level, component instantiation, concurrent processes



Fully
behavioral



Pipelined
structural

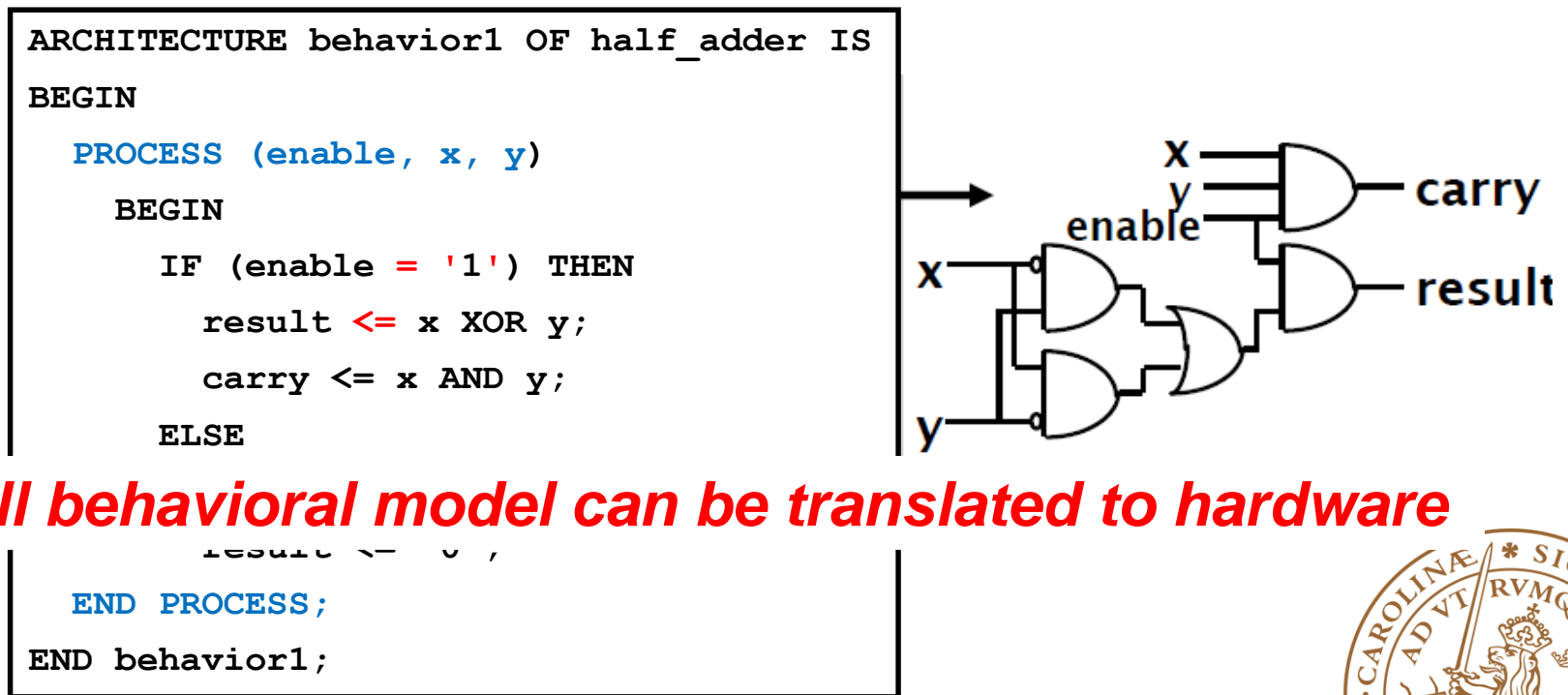


Architecture Body # 1

□ Behavioral Architecture: Describes the algorithm performed by the module, e.g., FSM

□ May contain

- Process statements
- Sequential statements
- Signal assignment statements



Not all behavioral model can be translated to hardware



Architecture Body # 2

□ **Structural architecture:** Implements a module as a composition of components (modules), a textual description of a **schematic**

□ **Contains**

- **Component, Signal declarations**

- *define the components (gates) and wires to be used*

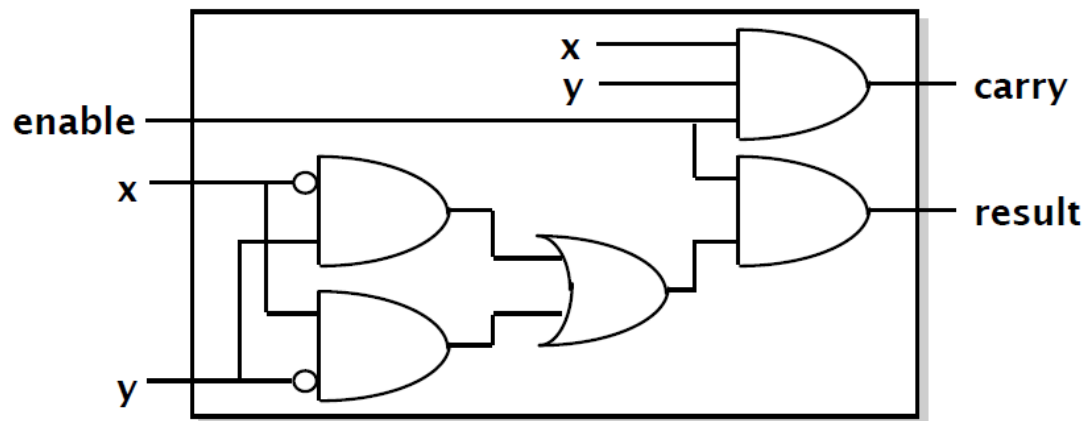
- *entity ports are treated as signals*

- **Component instances**

- *instances of previously declared entity/architecture pairs*

- **Port maps** in component instances

- *connect signals to component ports*



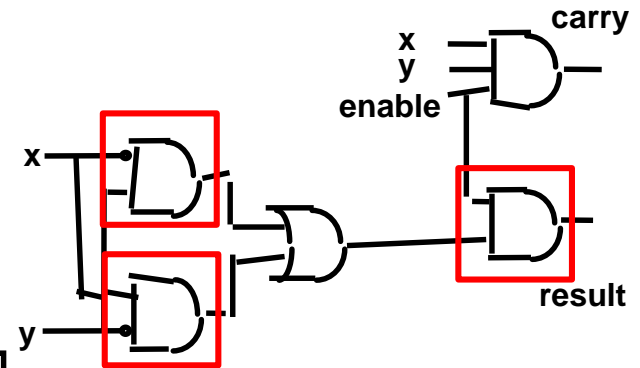
Architecture Body # 2 (cntd.)

```
ENTITY not_1 IS
    PORT (a: IN bit; output: OUT bit);
END not_1;

ARCHITECTURE data_flow OF not_1 IS
BEGIN
    output <= NOT(a);
END data_flow;
```

```
ENTITY and_2 IS
    PORT (a,b: IN bit; output: OUT bit);
END and_2;

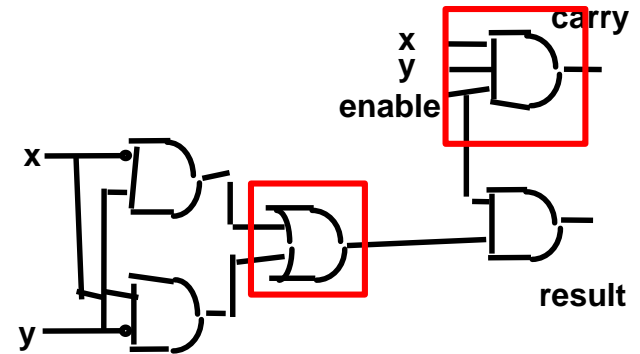
ARCHITECTURE data_flow OF and_2 IS
BEGIN
    output <= a AND b;
END data_flow;
```



Architecture Body # 2 (cntd.)

```
ENTITY or_2 IS
    PORT (a,b: IN bit; output: OUT bit);
END or_2;

ARCHITECTURE data_flow OF or_2 IS
BEGIN
    output <= a OR b;
END data_flow;
```



```
ENTITY and_3 IS
    PORT (a,b,c: IN bit; output: OUT bit);
END and_3;

ARCHITECTURE data_flow OF and_3 IS
BEGIN
    output <= a AND b AND c;
END data_flow;
```



Architecture Body # 2 (cntd.)

ARCHITECTURE structural OF half adder IS

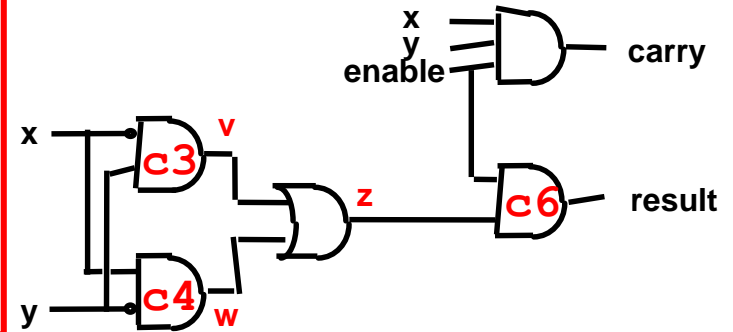
```
COMPONENT and2 PORT(a,b: IN bit; output: OUT bit); END COMPONENT;  
COMPONENT and3 PORT(a,b,c: IN bit; output: OUT bit); END COMPONENT;  
COMPONENT or2 PORT(a,b: IN bit; output: OUT bit); END COMPONENT;  
COMPONENT not1 PORT(a: IN bit; output: OUT bit); END COMPONENT;
```

```
SIGNAL v,w,z,nx,nz: BIT;
```

BEGIN

```
c1: not1 PORT MAP (a=>x,output=>nx);  
c2: not1 PORT MAP (y,ny);  
c3: and2 PORT MAP (nx,y,v);  
c4: and2 PORT MAP (x,ny,w);  
c5: or2 PORT MAP (v,w,z);  
c6: and2 PORT MAP (enable,z,result);  
c7: and3 PORT MAP (x,y,enable,carry);
```

END structural;



Advantages of Structural description

□ Hierarchy

- Allows for the simplification of the design

□ Component Reusability

- Allows the re-use of specific components of the design (Latch, Flip-flops, half-adders, etc)

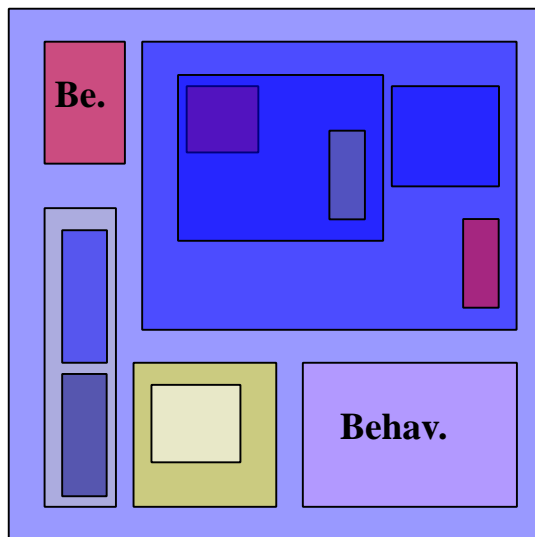
□ Design Independent

- Allows for replacing and testing components without redesigning the circuit



Architecture - Mixing Behavioral and Structural

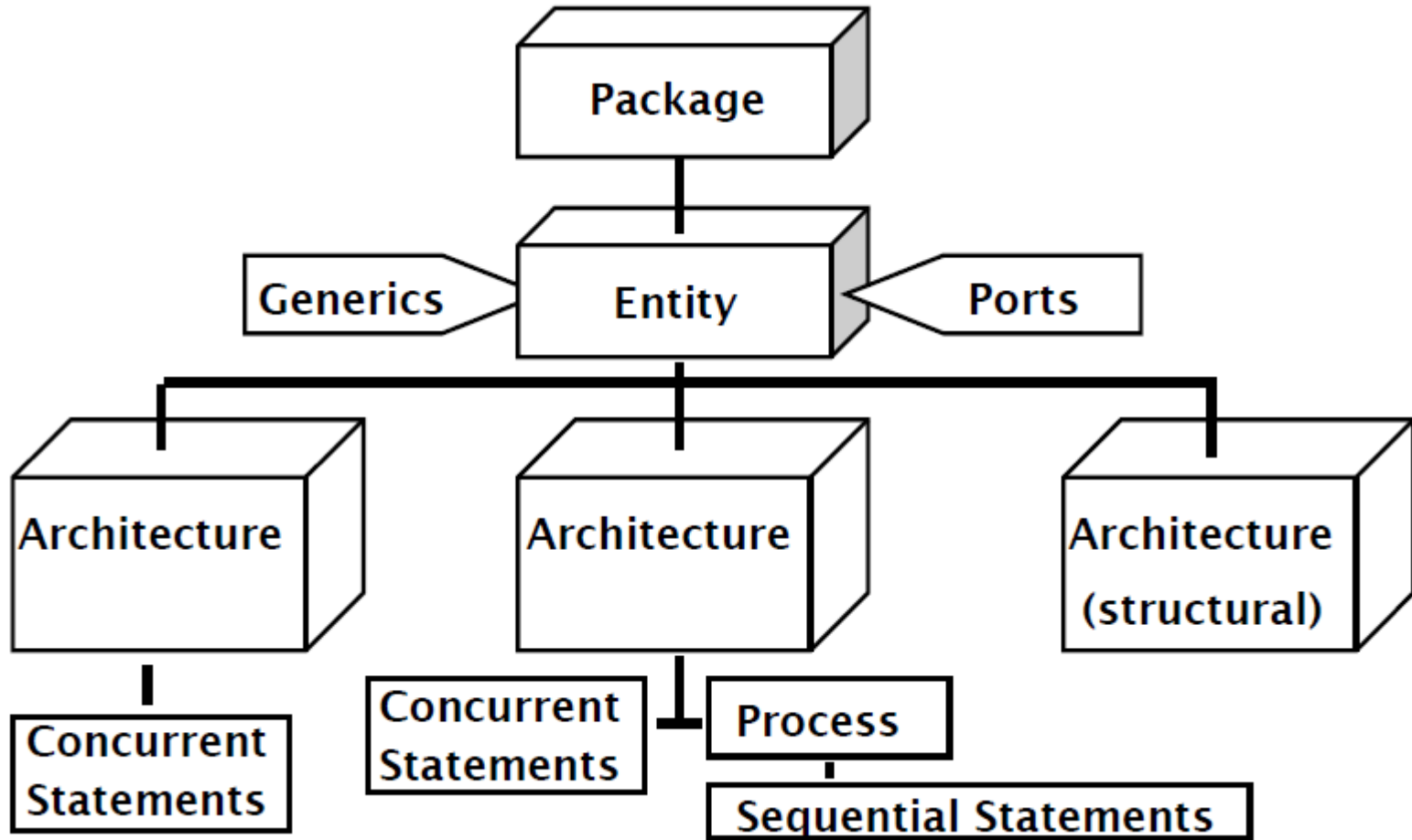
- An architecture may contain both behavioral and structural parts
 - Process statements and component instances
- Example: Register-Transfer-Level (RTL) model
 - **data path** described **structurally** (component)
 - **control section** described **behaviorally**



Partially behavioral.
& structural.



Summary



VHDL Process

- ❑ Contains a set of sequential statements to be executed sequentially
- ❑ Can be interpreted as a **circuit part** enclosed inside a black box
- ❑ Process statements:

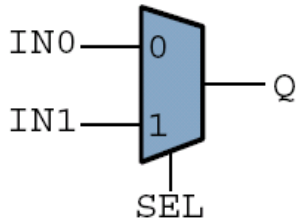
```
name_label: process (sensitivity list)
    variable declarations...
begin
    sequential statements...
- if ... then ... [else | elsif ...] end if;
- for n in 0 to 7 loop...
- case b is ...
- s := z sll shamt;
- i := a + b; --variable assignment, only in processes
- c <= i; --signal assignment!
end process namelabel;
```



VHDL Process: Example 1

- A process is activated when a signal in the sensitivity list changes its value

Writing **combinational** components:



architecture BEHAV of MUX2 is

begin

```
process (INO, IN1, SEL);
```

sensitivity list

```
begin
```

```
Q <= IN0;
```

default assignment

```
if( SEL = '0' ) then
```

```
Q <= IN0;
```

```
elsif( SEL = '1' ) then
```

```
Q <= IN1;
```

```
end if;
```

```
end process;
```

```
end BEHAV;
```

For a component to be combinational:

1. All inputs **MUST** be present in the sensitivity list!
2. All outputs **MUST** be assigned on every run!

If conditions 1 and 2 above are not fulfilled,
THE SYNTHESIZED DESIGN WILL NOT WORK!

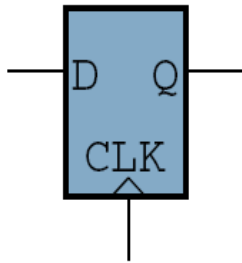


VHDL Process: Example 2

Enable register with synchronous reset

```
process (clk)
begin
    if (clk'event and clk='1')
    then
        if (Reset = '0') then
            Q <= '0';
        elseif enable='1' then
            Q <= D;
        end if;
    end if;
end process ;
```

Writing **sequential** (clocked) components:



```
architecture GOOSE of FLIPFLOP is
begin
```

```
    process ( CLK )
    begin
        if (CLK = '1') and (CLK'event) then
            Q<=D;
        end if;
    end process;
```

```
end GOOSE;
```

the SENSITIVITY LIST

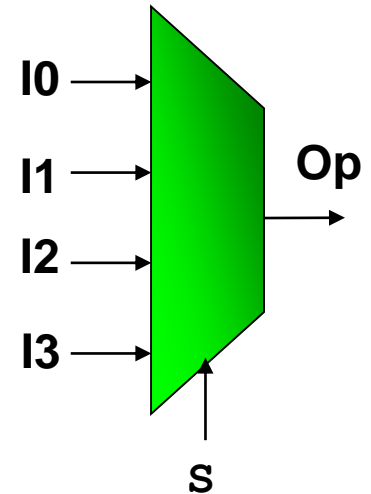
When a signal present in the sensitivity list changes, the process is run once, top to bottom.



Case Statement

□ Example: Multiplexer

```
architecture behv1 of Mux is
begin
process (I3, I2, I1, I0, S)  --inputs to process
begin -- use case statement
  case S is
    when "00" => Op <= I0;  --sequential statements
    when "01" => Op <= I1;
    when "10" => Op <= I2;
    when "11" => Op <= I3;
    when others => Op <= I0;
  end case;
end process;
end behv1;
```



Outline

□ VHDL Background

- What is VHDL?
- Why VHDL?

□ Basic VHDL Component

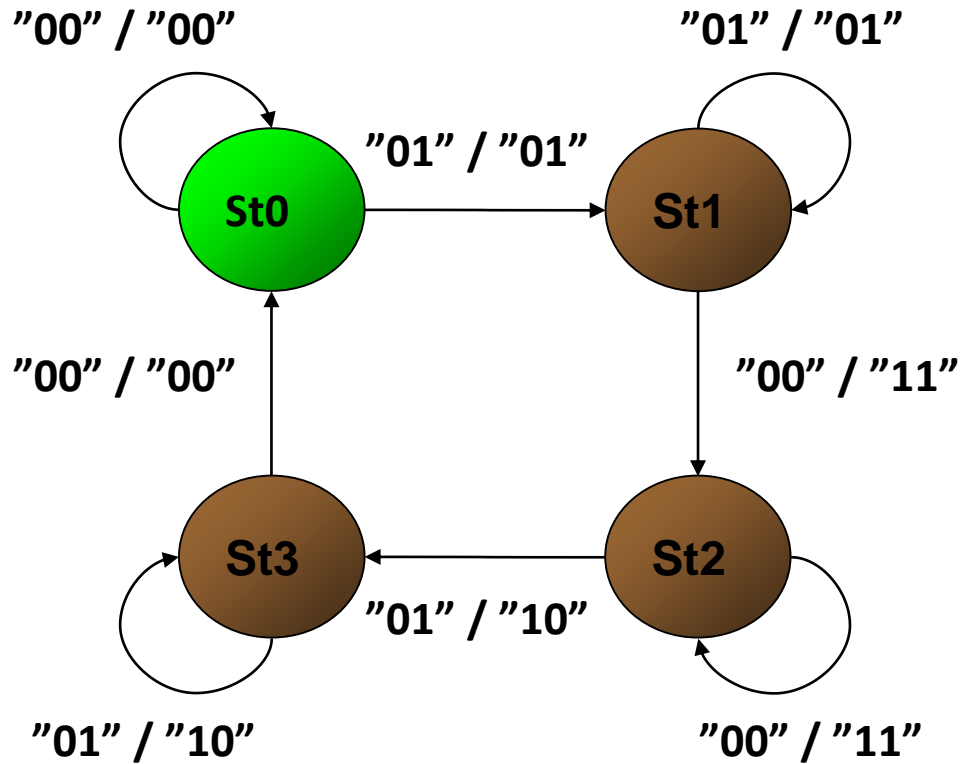
- A example

□ **FSM Design with VHDL**

□ Simulation & TestBench



Finite State Machine (FSM)

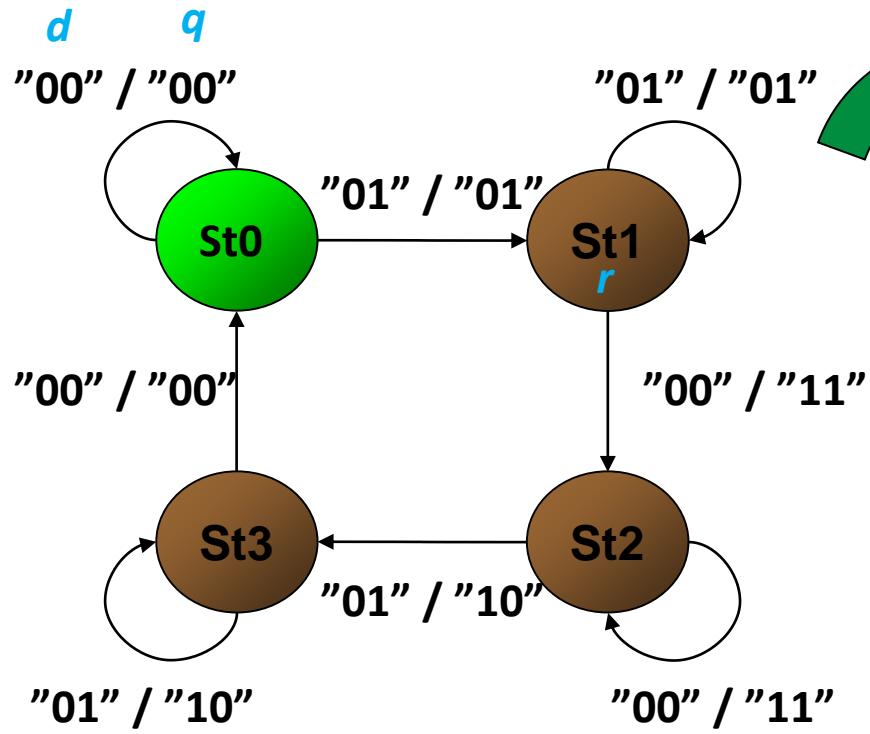


Output of a Mealy machine is **state** and **input** dependent

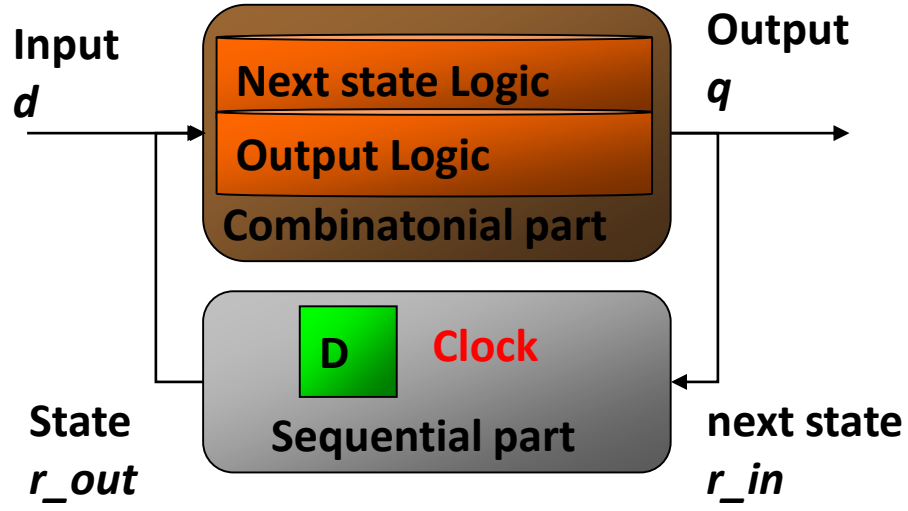
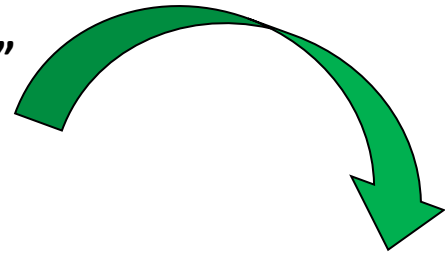
A Typical state machine



Transforming a State Diagram into HW



Typical FSM



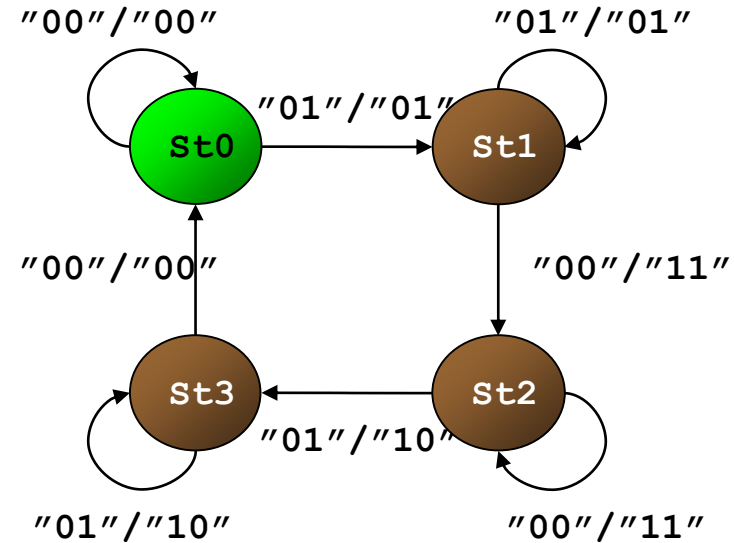
Generic Architecture for FSMs



VHDL Realization of FSMs

Entity declaration

```
library IEEE;  
use IEEE.STD_LOGIC_1164.all;  
entity state_machine is  
    port (clk : in STD_LOGIC;  
          reset : in STD_LOGIC;  
          input : in STD_LOGIC_VECTOR(1 downto 0);  
          output : out STD_LOGIC_VECTOR(1 downto 0)  
          );  
end state_machine;
```



VHDL Realization of FSMs (cont'd)

Architecture declaration (combinational part)

```
architecture implementation of state machine is
```

```
type state_type is (st0,st1,st2,st3); -- Defines states;  
signal state, next_state : state_type;
```

```
begin
```

```
combinational : process (input,state)
```

```
begin
```

```
case (state) is -- Current state and input dependent
```

```
when st0 => if (input = "01") then  
             next_state <= st1;  
             output <= "01"
```

```
           else ...  
           end if;
```

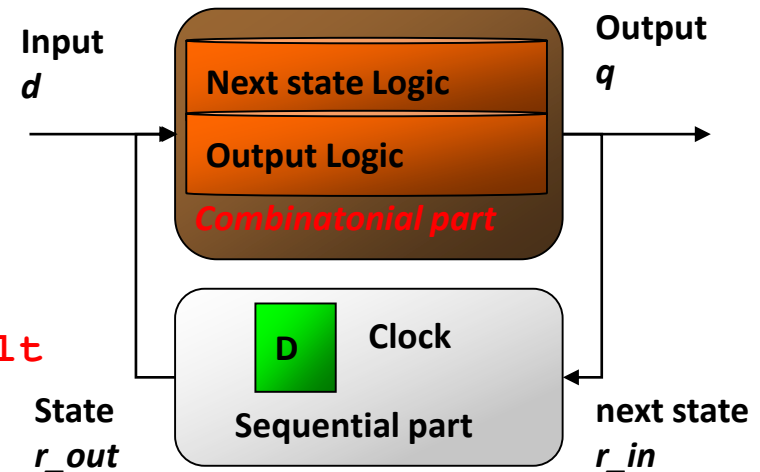
```
when ....
```

```
when others =>
```

```
    next_state <= st0; -- Default  
    output <= "00";
```

```
end case;
```

```
end process;
```



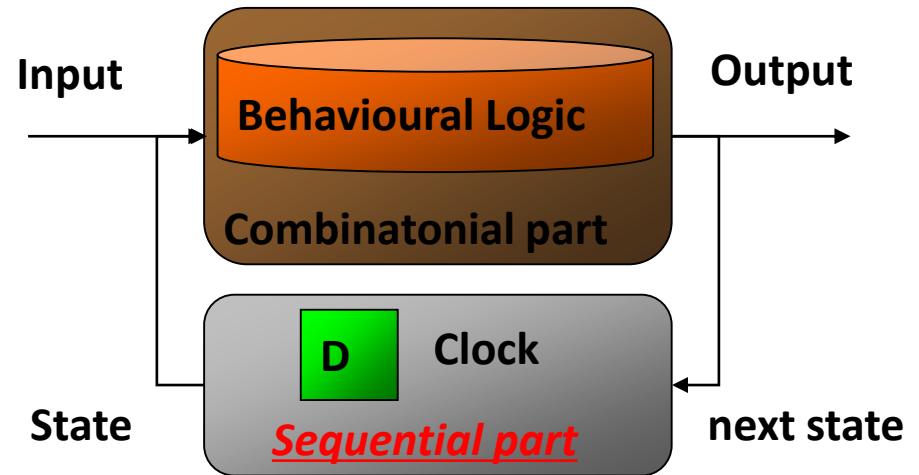
Suggestion: Use enumerate data type for states



VHDL Realization of FSMs (cont'd)

□ Architecture declaration (sequential part)

```
synchronous : process (clk)
begin
  if (clk'event and clk = '1') then
    if reset = '1' then
      state <= st0;
    else
      state <= next_state;
    end if;
  end if;
end process;
end architecture;
```



Generic Architecture for FSMs

Suggestion: Separate the processes of Comb. And Seq.



Outline

□ VHDL Background

- What is VHDL?
- Why VHDL?
- How to code VHDL?

□ Basic VHDL Component

- A example

□ FSM Design with VHDL

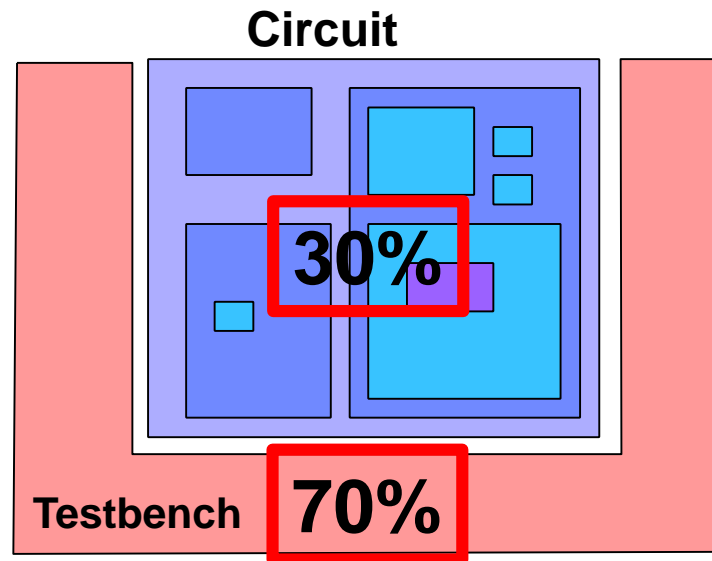
□ Simulation & TestBench



Testbench and Simulation

□ Testing: Testbench and Circuit

- The testbench models the environment our circuit is situated in.
- Provides stimuli (input to circuit) during simulation.
- May verify the output of our circuit against test vectors.
- Verification should be conducted at **ALL** design levels.



Testbench Example

Top level entity connecting the circuit to the testbench

```
entity testbench is
end testbench;
architecture test of testbench is
```

Component declaration of the circuit being tested

```
component circuit is
port (clk, reset, inputs, outputs);
end circuit;
signal inputs, outputs, clk, reset : type;
```

Clock generator

```
begin
clk_gen: process
begin
    if clk='1' then clk<='0';
    else clk<='1'; end if;
    wait for clk_period/2;
end process;
```

Reset signal generator

```
reset <= '1', '0' after 57 ns;
```

Component instantiation of the circuit being tested

```
device: circuit
    port map (clk, reset, inputs, outputs);
```

The tester, which generates stimuli (inputs) and verifies the response (outputs)

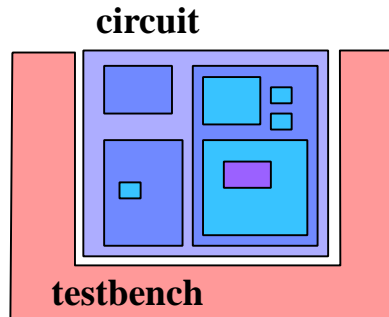
```
tester: process (clk, reset)
    begin
        ...
    end process;
end testbench;
```

– The tester could also be a separate component.



Testbench and Simulation: Testing larger circuits

□ Divide and conquer

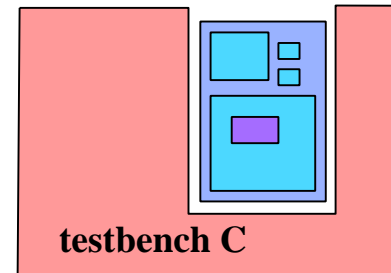
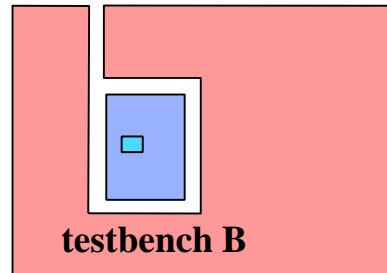
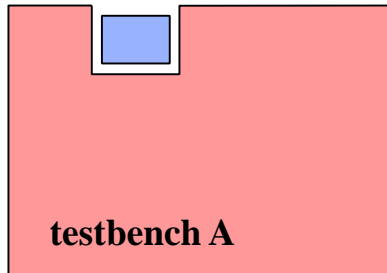


Test (simulation) fails !

What then ?

How can I find the bug ?

□ 3 subcomponents -> 3 subtests:



This will localize the problem or problems! Repeat the procedure if a faulty component consists of subcomponents, etc.

Overall Test is still needed!!!



Recommendation Readings

□ Mujtaba Hamid, “Writing Efficient Testbenches”, Xilinx Application Note

http://www.xilinx.com/support/documentation/application_notes/xapp199.pdf

□ “VHDL Test Bench Tutorial”, University of Pennsylvania

<http://www.seas.upenn.edu/~ese171/vhdl/VHDLTestbench.pdf>



Questions?

