



The Abdus Salam  
International Centre for Theoretical Physics



**310/1780-11**

**ICTP-INFN Advanced Training Course on  
FPGA and VHDL for Hardware Simulation and Synthesis  
27 November - 22 December 2006**

---

*VHDL & FPGA – Session 3*

***Nizar ABDALLH  
ACTEL Corp.  
2061 Stierlin Court  
Mountain View, CA 94043-4655  
U.S.A.***

---

***These lecture notes are intended only for distribution to participants***



# Lectures: VHDL & FPGA Architectures

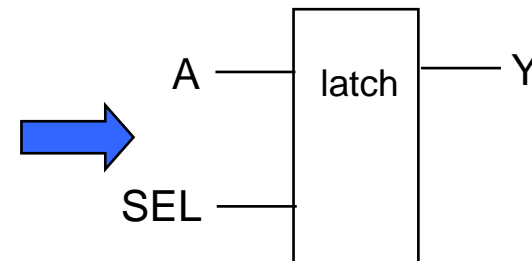


- Introduction to FPGA & FPGA Design Flow
- Synthesis I - Introduction
- Synthesis II - Introduction to VHDL
- Synthesis III - Advanced VHDL
- Design verification & timing concepts
- Programmable logic & FPGA architectures
- Actel ProASIC3 FPGA architecture

- A latch or flip-flop is inferred if all branches of an IF statement are not assigned
- Latch is inferred when if statement includes level value
- Flip-Flop is inferred when if statement detects an edge
- Simulator needs to hold previous output under certain conditions if no else statement is included

- Latch is inferred when if statement detects a level (0 or 1) and all branches of an IF statement are not assigned

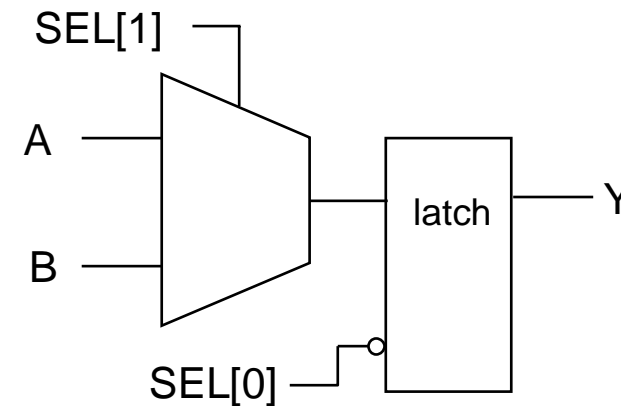
```
process (SEL, A)
begin
  if (SEL = '1') then Y <= A;
  end if ;
end process;
```



To avoid unwanted latches, include an ELSE condition

- CASE statements using “when others => null” can infer latches if type is std\_logic or std\_logic\_vector

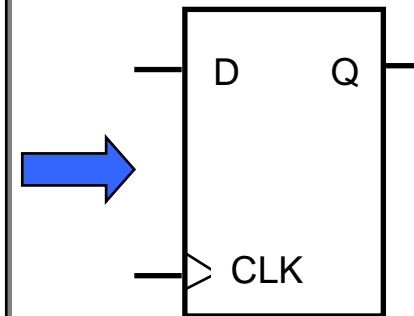
```
-- sel, A, B are std_logic
process (SEL, A, B)
begin
case SEL is
when "00" => Y <= A;
when "10" => Y <= B;
when others => null;
end case;
end process;
```



To avoid unwanted latches, actually define Y for the “others” condition, for example:  
when others => Y <= '0';

- Use Processes and IF statements to describe sequential logic
- IF statement detects clock edge
  - rising edge = `if (CLK'event and CLK='1')`
  - falling edge = `if (CLK'event and CLK='0')`

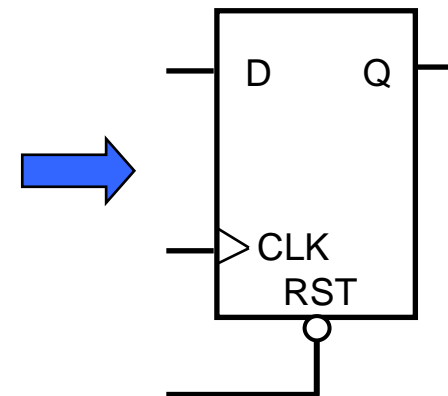
```
architecture BEHAVE of DF is
begin
  INFER: process (CLK) begin
    if (CLK'event and CLK = '1') then
      Q <= D;
    end if ;
  end process INFER;
end BEHAVE;
```



# Sequential Logic: Example 1

```
architecture FLOP of DFCLR is
begin
  INFER: process (CLK, RST)
  begin
    if (RST = '0') then
      Q <= '0';
    elsif (CLK'event and CLK = '1') then
      Q <= D;
    end if ;
  end process INFER;
end FLOP;
```

D flip-flop with asynchronous low reset and active high clock edge

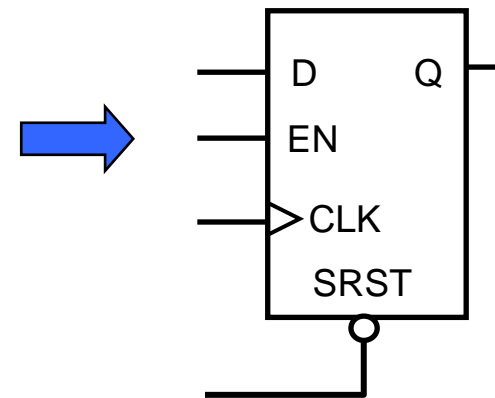




# Sequential Logic: Example 2

```
architecture FLOP of DFSLRHE is
begin
  INFER: process (CLK)
  begin
    if (CLK'event and CLK = '1') then
      if (SRST = '0') then
        Q <= '0';
      elsif (EN = '1') then
        Q <= D;
      end if ;
    end if ;
  end process INFER;
end FLOP;
```

D flip-flop with  
synchronous low reset,  
active high enable and  
rising edge clock



```
architecture FLOP of EN_FLOP is
begin
  INFER:process (CLK) begin
    if (CLK'event and CLK = '0') then
      if (EN = '0') then
        Q <= D;
      end if ;
    end if ;
  end process INFER;
end FLOP;
```

Will this model a positive edge or negative edge triggered flip-flop?

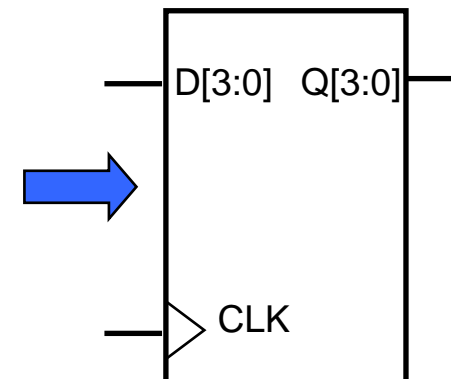
Is the enable synchronous or asynchronous?

Is the enable active high or active low?

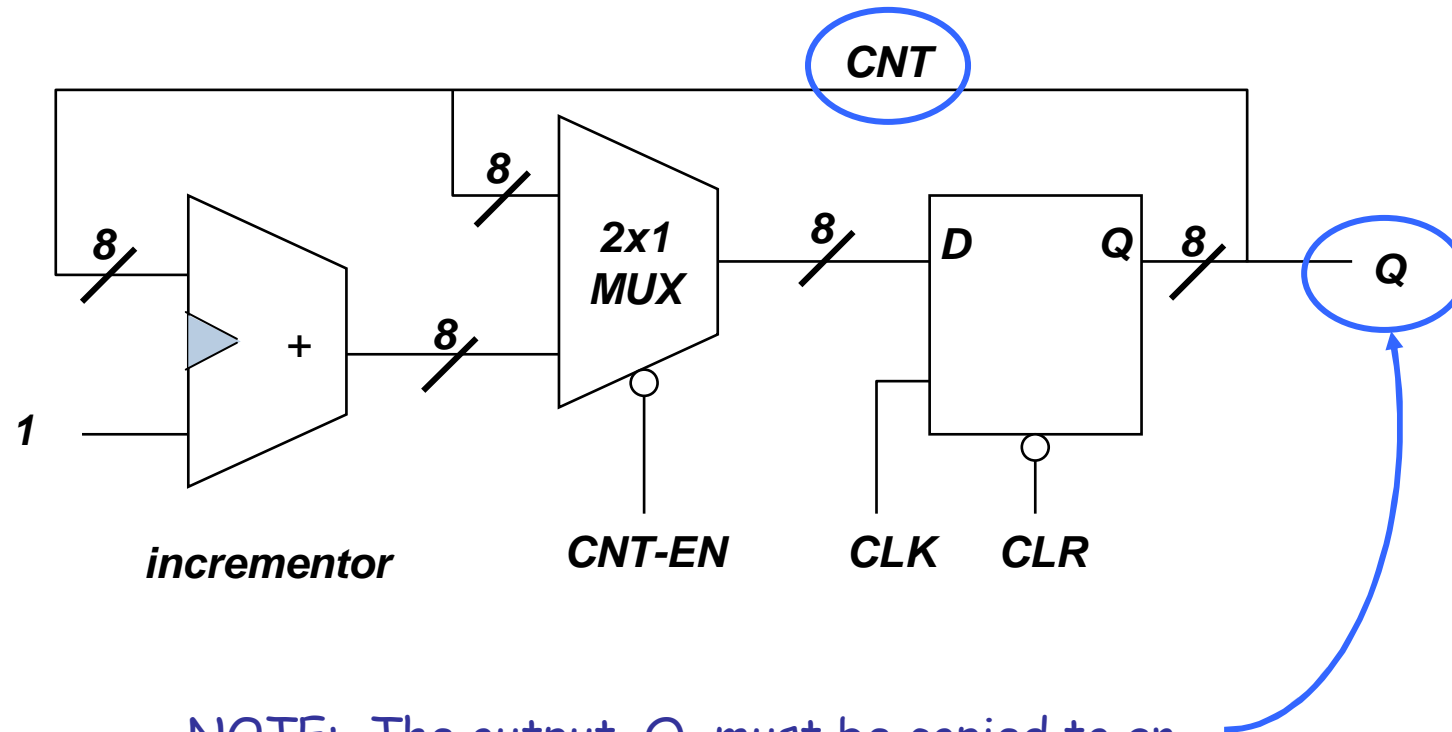
# Sequential Logic: Example 4

```
library ieee;
use ieee.std_logic_1164.all;
entity DF_4 is
port (D: in std_logic_vector(3 downto 0);
      CLK: in std_logic;
      Q: out std_logic_vector(3 downto 0));
end DF_4 ;
architecture FLOP of DF_4 is
begin
  INFER: process ← Where's the sensitivity list?
  begin
    wait until (CLK'event and CLK = '1');
    Q <= D;
  end process INFER;
end FLOP;
```

4-bit register  
using WAIT  
statement



# 8-bit Counter Example



NOTE: The output, Q, must be copied to an internal signal, CNT, since an output port can not appear on the right-hand side of an assignment operator

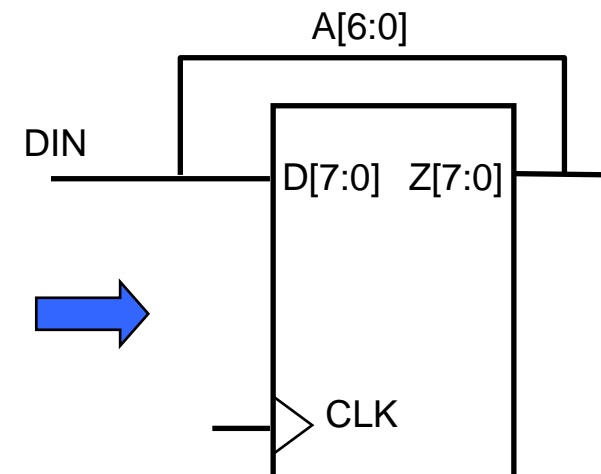
## 8-bit Counter Example (cont'd)



```
library ieee; use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity COUNTER is
  port (CLK,CNT_EN,CLR:in std_logic;
        Q          :out std_logic_vector(7 downto 0));
end COUNTER;
architecture BEHAVE of COUNTER is
  signal CNT:std_logic_vector(7 downto 0);
begin
  FIRST: process (CLK, CLR)
  begin
    if (CLR = '0') then
      CNT <= "00000000";
    elsif (CLK'event and CLK = '1') then
      if (CNT_EN = '0') then
        CNT <= CNT + '1';
      end if ;
    end if ;
  end process FIRST;
  Q <= CNT;
end BEHAVE;
```

# 8-bit Shift Register

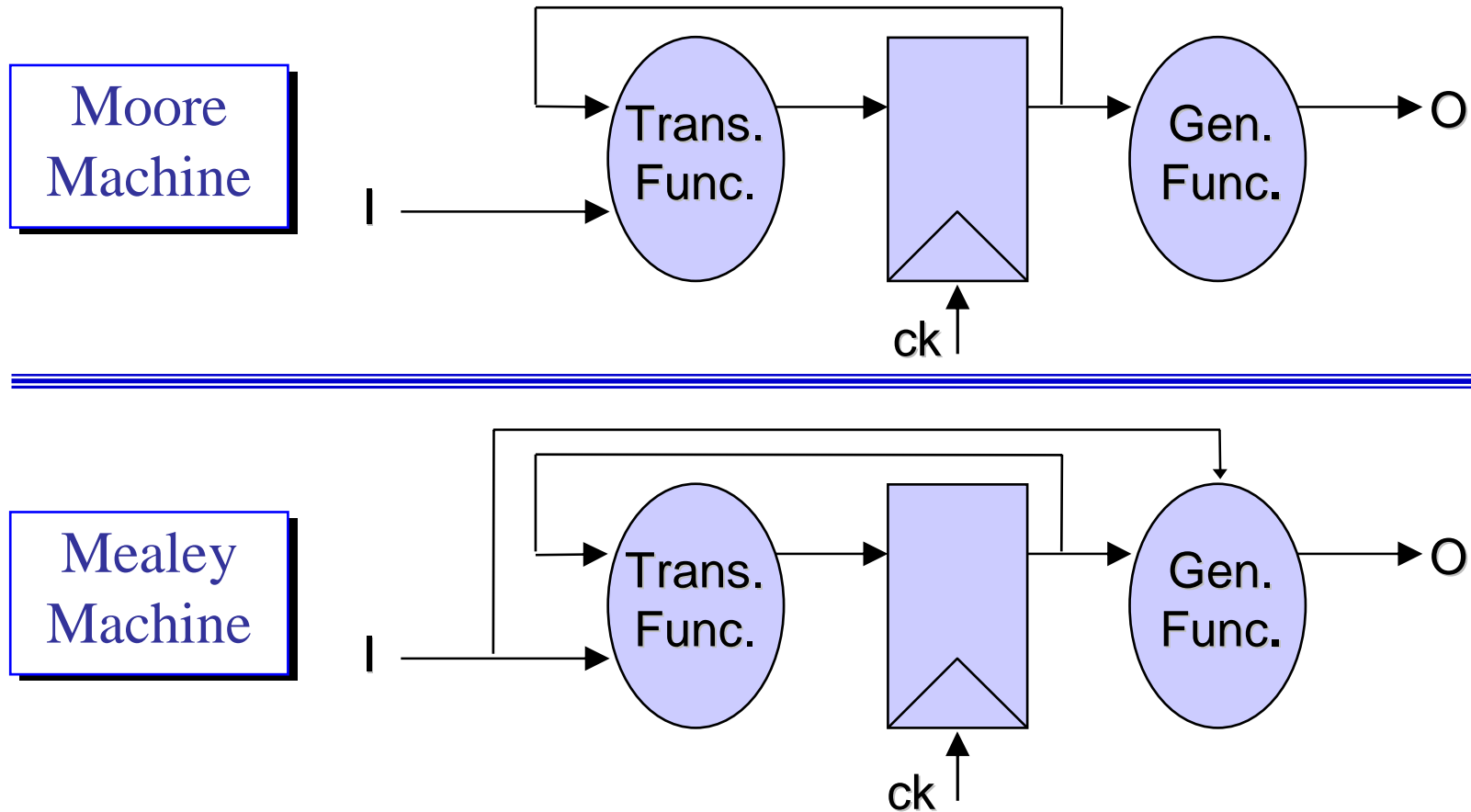
```
library ieee;
use ieee.std_logic_1164.all;
entity SHIFTER is
port(CLK, DIN: in std_logic;
      Z: out std_logic_vector(7 downto 0));
end SHIFTER;
architecture RTL of SHIFTER is
signal A: std_logic_vector(7 downto 0);
begin
process (CLK)
begin
if (CLK'event and CLK='1') then
A <= A (6 downto 0) & DIN; -- shift left
end if;
end process;
Z <= A;
end RTL
```



- Typically include:
  - At least 2 process statements (one **MUST** control the clocking)
  - **IF-THEN-ELSE** statements
  - **CASE** statements
  - **User defined types to hold current state and next state**
- Transitions depend on current state and optionally, the inputs
- Outputs depend on:
  - **Current state (Moore machine)**
  - **Current state & inputs (Mealy machine)**
- Definition:

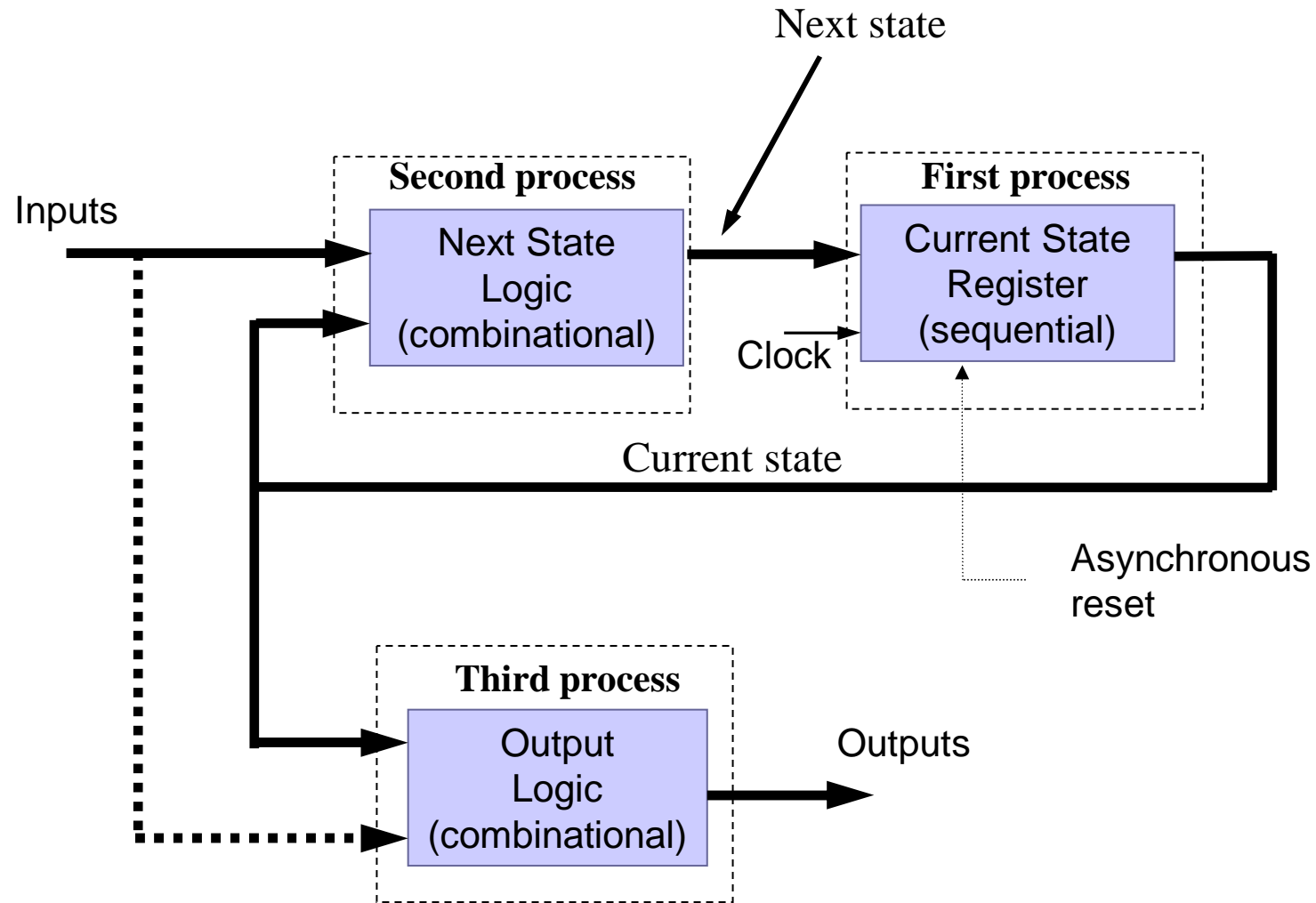
$$\begin{aligned} \text{State (t+1)} &\leq F(i1, \dots, in, \text{State}(t)) \\ \text{Output} &\leq F(i1, \dots, in, \text{State}(t)) \end{aligned}$$

# FSM: Two Machine Types

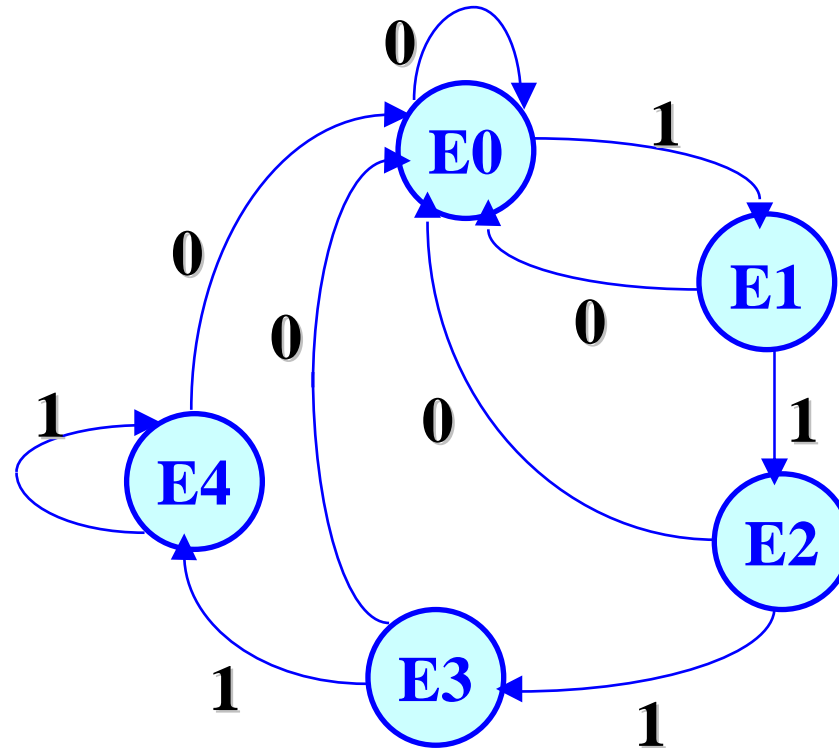




# Common FSM Structure in VHDL



# FSM Example: 4 Consecutive Ones Counter



# FSM Example: 4 Consecutive Ones Counter (cont'd)



```
--  
Entity counter is port (ck, I, reset: in bit; O: out bit);  
End counter;  
  
Architecture automate of counter is  
type STATE_TYPE is (E0, E1, E2, E3, E4);  
signal CURRENT_STATE, NEXT_STATE: STATE_TYPE;  
-- pragma CUR_STATE CURRENT_STATE;  
-- pragma NEX_STATE NEXT_STATE;  
-- pragma CLOCK ck;  
  
begin  
    Process(CURRENT_STATE, I, reset)  
    begin  
        if (reset = '1') then  
            NEXT_STATE <= E0;  
            O <= '0';  
        else
```

# FSM Example: 4 Consecutive Ones Counter (cont'd)



```
case CURRENT_STATE is
    WHEN E0 =>
        if (I='1') then
            NEXT_STATE <= E1;
        else
            NEXT_STATE <= E0;
        end if;
        O <= '0';

    WHEN E1 =>
        if (I='1') then
            NEXT_STATE <= E2;
        else
            NEXT_STATE <= E0;
        end if;
        O <= '0';
```

# FSM Example: 4 Consecutive Ones Counter (cont'd)



```
    WHEN E2 =>
        if (I='1') then
            NEXT_STATE <= E3;
        else
            NEXT_STATE <= E0;
        end if;
        O <= '0';

    WHEN E3 =>
        if (I='1') then
            NEXT_STATE <= E4;
        else
            NEXT_STATE <= E0;
        end if;
        O <= '0';
```

# FSM Example: 4 Consecutive Ones Counter (cont'd)



```
        WHEN E4 =>
            if (I='1') then
                NEXT_STATE <= E4;
            else
                NEXT_STATE <= E0;
            end if;
            O <= '1';

        WHEN others =>
            assert ('1')
            report "Illegal State";

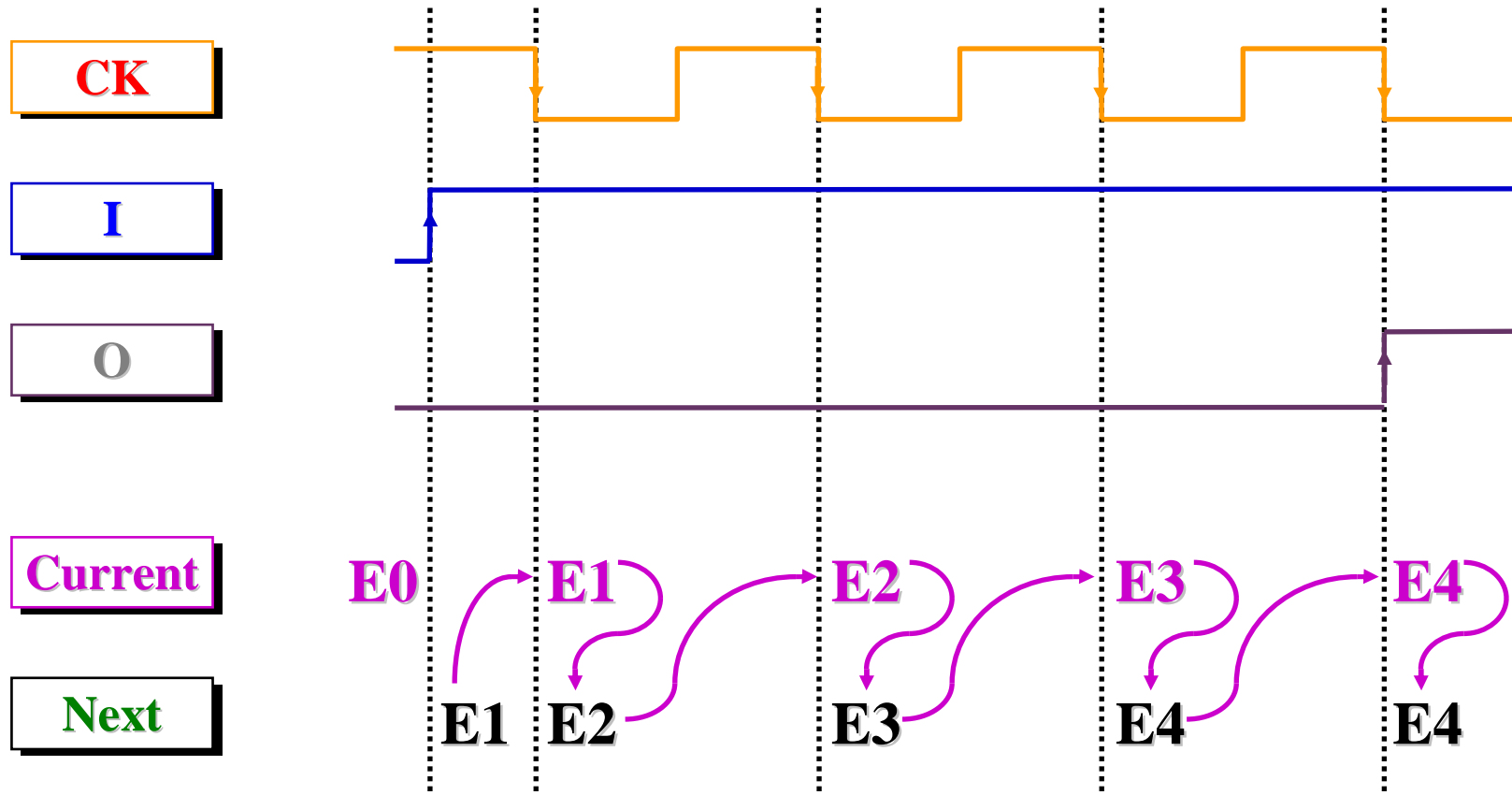
    end case;
end if;
end process;
```

# FSM Example: 4 Consecutive Ones Counter (cont'd)



```
Process (ck)
  begin
    if (ck = '0' and not ck'stable) then
      CURRENT_STATE <= NEXT_STATE;
    end if;
  end process;
end counter;
```

# FSM Example: 4 Consecutive Ones Counter (cont'd)



*What happens if a glitch occurs on the input I?*



- All Signals which are Assigned to within a Clocked Process Have Registers on their Outputs
  
- Signal Assignments within a Process are Effective only before the wait (implicit or explicit) Statement

- **Three step process:**
  - **Simulate RTL vs. specification**
  - **Simulate structural (using VITAL) vs. RTL**
  - **Simulate structural (using VITAL) with back-annotated timing**
  
- **Procedure**
  - **Use testbench or manually apply stimulus**
  - **Check for correct results and produce a trace file**
  
- **Choices**
  - **Use vendor-specific stimulus file (non-portable)**
  - **Write generic VHDL testbench**

- Written by designer using standard VHDL
  - Portable to any VHDL simulator
- Creates a new level of design hierarchy
  - Component instantiation of design under test
  - VHDL processes to apply stimulus and record outputs
- Uses VHDL textio package
  - Read or write to ASCII data files
  - Input test vectors (times and values)
  - Tabular trace, print-on-change, strobe

- Writes out text messages during simulation
- Useful for timing checks, out of range conditions, etc.
- Four levels
  - **Failure**
  - **Error**
  - **Warning**
  - **Note**

```
assert (Y > 2)  
report "SETUP VIOLATION"  
severity Warning;
```

- Suspends execution of the process or sub-program

- Usage:

- **wait**
- **wait for <time>**
- **wait until <condition>**
- **wait on <signals>**

```
wait for 10 NS;
```

```
wait until X > 10;
```

- Remember!

Processes with a sensitivity list cannot have a WAIT statement

## ■ A sequential waveform can be generated using

- **Multiple signal assignments in a single concurrent signal assignment**

```
ENABLE <= '0', '1' after 100 ns,  
'0' after 180 ns, '1' after 210 ns;
```

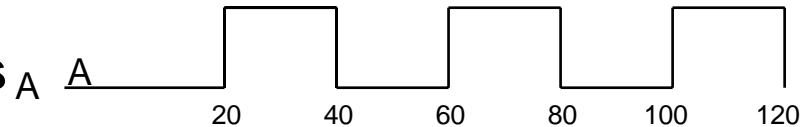
- **sequential signal assignments in a process**

```
. . .  
process  
begin  
    ENABLE <= '0';  
    wait for 100 ns;  
    ENABLE <= '1';  
    wait for 80 ns;  
    ENABLE <= '0';  
    wait for 30 ns;  
    ENABLE <= '1';  
    wait;  
end process;  
. . .
```

## ■ Waveforms with

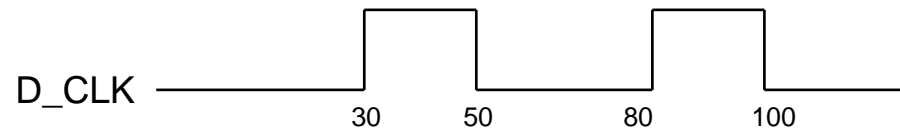
- **Constant 50% duty cycle can be created with a single concurrent signal assignment or in a process**

```
. . .  
A <= not A after 20 ns;  
. . .
```



- **Varying on-off delays can be created using a process statement**

```
. . .  
CLK: process  
    constant OFF_PERIOD: TIME:= 30 ns;  
    constant ON_PERIOD: TIME:= 20 ns;  
begin  
    wait for OFF_PERIOD;  
    D_CLK <= '1';  
    wait for ON_PERIOD;  
    D_CLK <= '0';  
end process;  
. . .
```



# Testbench Example: 8-Bit Counter



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

entity TESTBENCH is
end TESTBENCH;

architecture BEHAVE of TESTBENCH is

    component COUNTER
    port (CLK,CNT_EN,CLR:in std_logic;
         Q:out std_logic_vector(7 downto 0));
    end component;
    signal CLKIN,ENABLE,RESET:std_logic;
    signal Qout:std_logic_vector(7 downto 0);

begin -- Instantiate Counter
U1:COUNTER port map(CLK=>CLKIN, CNT_EN=>ENABLE,
                   CLR=>RESET, Q=>Qout);
```

Testbench entity does not include ports

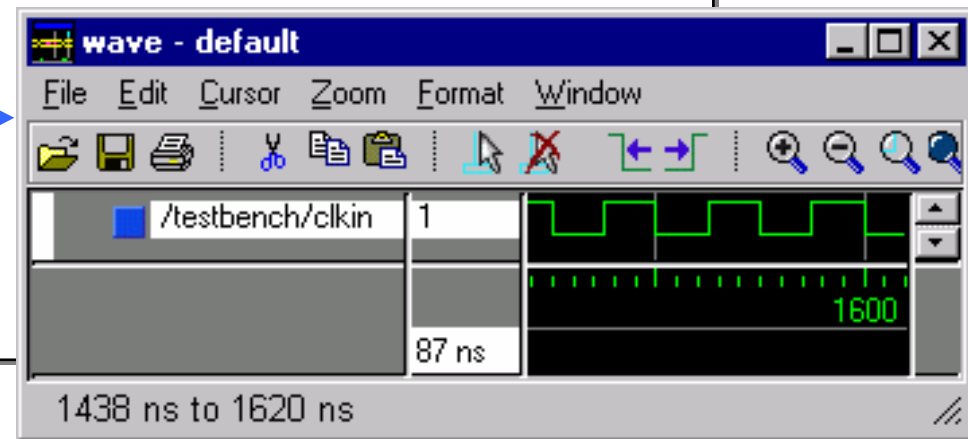
Counter declared within testbench

Counter instantiated within testbench



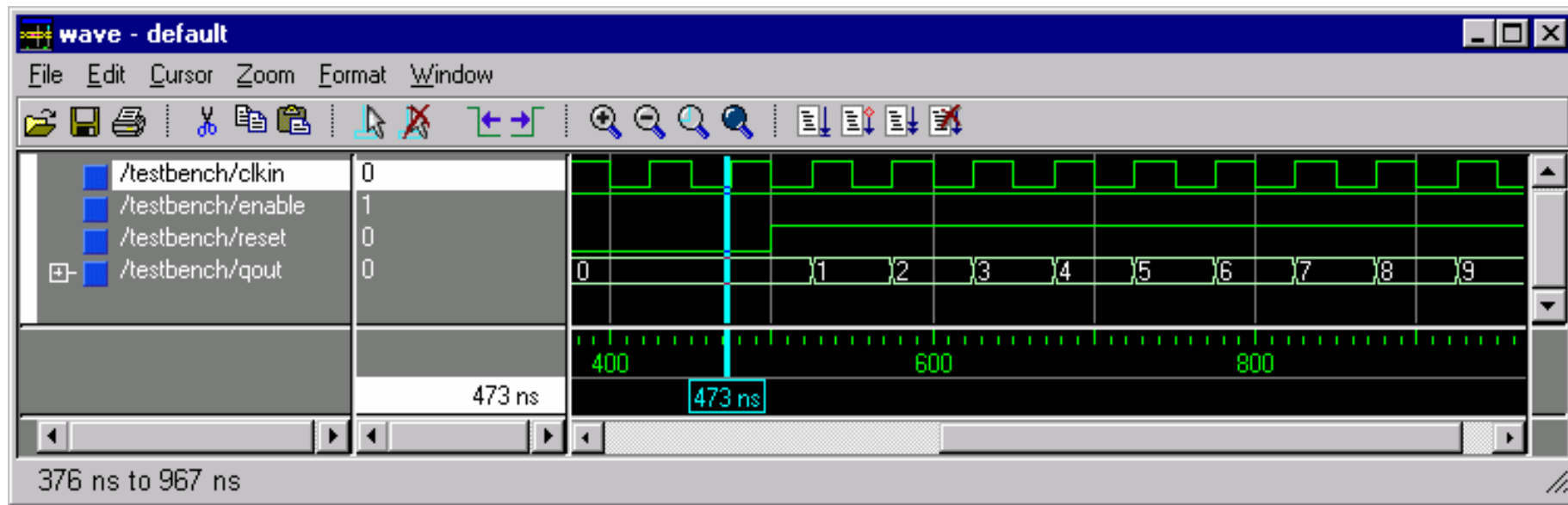
# Testbench Example: 8-Bit Counter (cont.)

```
-- initialize inputs, and toggle the reset line
INIT: process begin
    ENABLE <= '1';
    RESET  <= '1';
    wait for 250 ns;
    RESET  <= '0';
    wait for 250 ns;
    RESET  <= '1';
    wait; -- this instruction suspends the init process
end process INIT;
-- process to cause clock to toggle (20 MHz)
CLK_TOG: process begin
    CLKIN  <= '0';
    wait for 25 ns;
    CLKIN  <= '1';
    wait for 25 ns;
end process CLK_TOG;
end BEHAVE;
```



# Testbench Example: 8-Bit Counter (cont.)

```
list
File Edit Markers Prop Window
ns /testbench/clkin
delta /testbench/enable
      /testbench/reset
      /testbench/qout
488 +0      1 1 0 0
538 +0      1 1 1 1
588 +0      1 1 1 2
638 +0      1 1 1 3
Default dataset: sim
```



```
process (A, B, SEL)
begin
    if (SEL='1') then
        OUT <= A;
    else
        OUT <= B;
    end if;
end process;
```

- Sensitivity list must consist of all signals that are read inside the process
  - Synthesis tools often ignore sensitivity list, but simulation tools do not...
  - A forgotten signal will lead to difference in behavior of the simulated model and the synthesized design

# Common Issues: Combinational Processes



```
process (A, B, SEL)
begin
  if (SEL='1') then OUT <= A;
  else OUT <= B;
  end if;
end process;
```

```
process
begin
  if (SEL='1') then
    OUT <= A;
  else
    OUT <= B;
  end if;
wait on A, B, SEL;
end process;
```

- Can use WAIT ON instead of sensitivity list
- But not both!

```
process
begin
    if (condition)
        wait on CLK'event and CLK=1;
    end if;
end process;
```

- Every path through a process body without sensitivity list must have a *wait*
  - **Otherwise the process can hang**

# Common Issues: Mistakenly Inferences Latches



```
process (A, B)
begin
    if (condition_1)
        X <= A + B;
    elsif (condition_2)
        X <= X - B;
    end if;
end process;
```

- Remember, incomplete assignments imply latches
  - In the above example, if neither condition\_1 nor condition\_2 is true then X will retain its value ... basically, X is stored in a latch
  - If you are writing combinational logic, make sure that every output gets assigned a value along each path (e.g. if statements, case statements) through the process body
  - In general, latches are not recommended anyway in synchronous designs (not testable via scan paths)

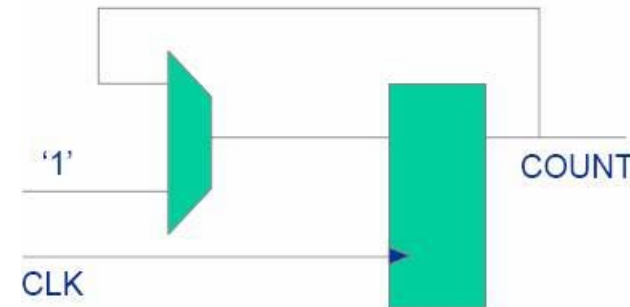
# Common Issues: The Problem with Latches



- Most EDA software tools have difficulty with latches because of transparency
  - **Timing analysis must consider both open and closed cases**
  - **Test vector generation is complicated**
  - **Latches are not scan testable**
  
- Good design practice:
  - **ASICs and FPGAs are a flip-flop's world**
  - **Don't use latches unless you absolutely have to**
  
- Poorly coded if and case statements can yield unintended latches

# Common Issues: Implicit Register Inference

```
process (A, B)
begin
    wait until CLK'event and CLK=1;
    if (COUNT >= 9) then
        COUNT <= 0;
    else
        COUNT <= COUNT +1;
    end if;
end process;
```



- Storage registers are synthesized for all signals that are driven within a clocked process
- Storage registers are also synthesized for all variables that are read before being updated



# Common Issues: Reset (or Set) in Synthesis



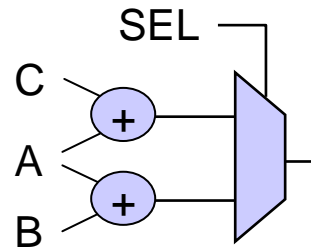
```
process
begin
  wait until CLK'event and CLK=1;
  if (RST='1') then
    -- synchronous reset
  else
    -- combinational code
  end if;
end process;
```

```
Process (CLK, RST)
begin
  if (RST='1') then
    -- asynchronous reset
  elsif (CLK'event and
                                               CLK=1) then
    -- combinational code
  end if;
end process;
```

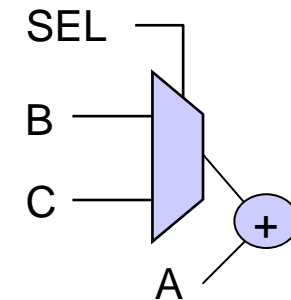
- Must reset all registers, otherwise synthesized chip won't work
  - **Unlike simulation, you can't set initial values in synthesis!**
- Asynchronous reset possible only with a process that has a sensitivity list

# Common Issues: Coding Style Influence

```
process(A, B, C, SEL)
begin
  if (SEL='1') then
    Z <= A + B;
  else
    Z <= A + C;
  end if;
end process;
```



```
Process (A, B, C, SEL)
  variable tmp: bit;
begin
  if (SEL='1') then
    tmp := B;
  else
    tmp := C;
  end if;
  Z <= A + tmp;
end process;
```



- Structure of initially generated hardware is determined by the VHDL code itself
  - Synthesis optimizes that initially generated hardware, but cannot do dramatic changes
  - Therefore, coding style matters!

- IF-THEN-ELSIF-THEN-...-ELSE maps to a chain of 2-to-1 multiplexers

```
if (COND1) then OUT <= X1;  
elsif (COND2) then OUT <= X2;  
...  
else OUT <= Xn;
```

- CASE maps to a single N-to-1 multiplexer

```
...  
case EXPRESSION is  
when VALUE1 =>  
    OUT <= X1;  
when VALUE2 =>  
    OUT <= X2;  
...  
when others =>  
    OUT <= Xn;  
end case;  
...
```

# Common Issues: Let the tool do the Synthesis



- Don't do synthesis by hand!
  - Do not come up with Boolean functions for outputs of arithmetic operator
  - Let Synthesis tool decide which adder, multiplier to use
  - You will only restrict the synthesis process
- Let synthesis tool decide the numeric encoding of the FSM states
  - Use enumerated type for state
- Split into multiple simpler processes
- Keep module outputs registered
  - Simplifies timing constraints