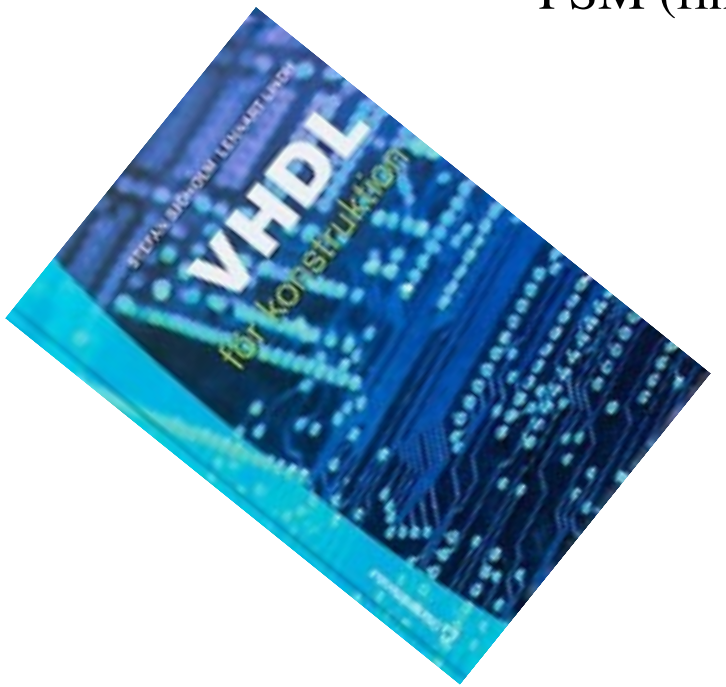
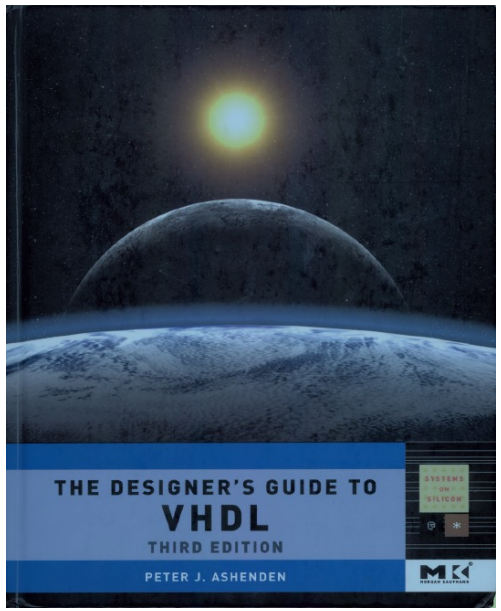
Digital project

VHDL – Intro +

useful things

FSM (finite state machine), if time.





Outline

- Why VHDL?
- HDL vs “regular programming language”
- The basics
- An example
  - Contains the basics of the basics
  - If you understand it you can pass the course!
- Common pitfalls!
- State machines if time (otherwise next time)

HDL – Code

VHDL:

VHSIC Hardware Description Language

Very High Speed Integrated Circuit

(Verilog)

Used for:

- Description
- Simulation
- Synthesis

Mor info : Part-1 VHDL Basic

Why VHDL?

- HDL is the method for designing digital systems today. (VHDL or VERILOG)
- VHDL is originally designed for simulation.

- **Simulation and synthesis to hardware are two different things!**

VHDL - just looks like a program!

Common program

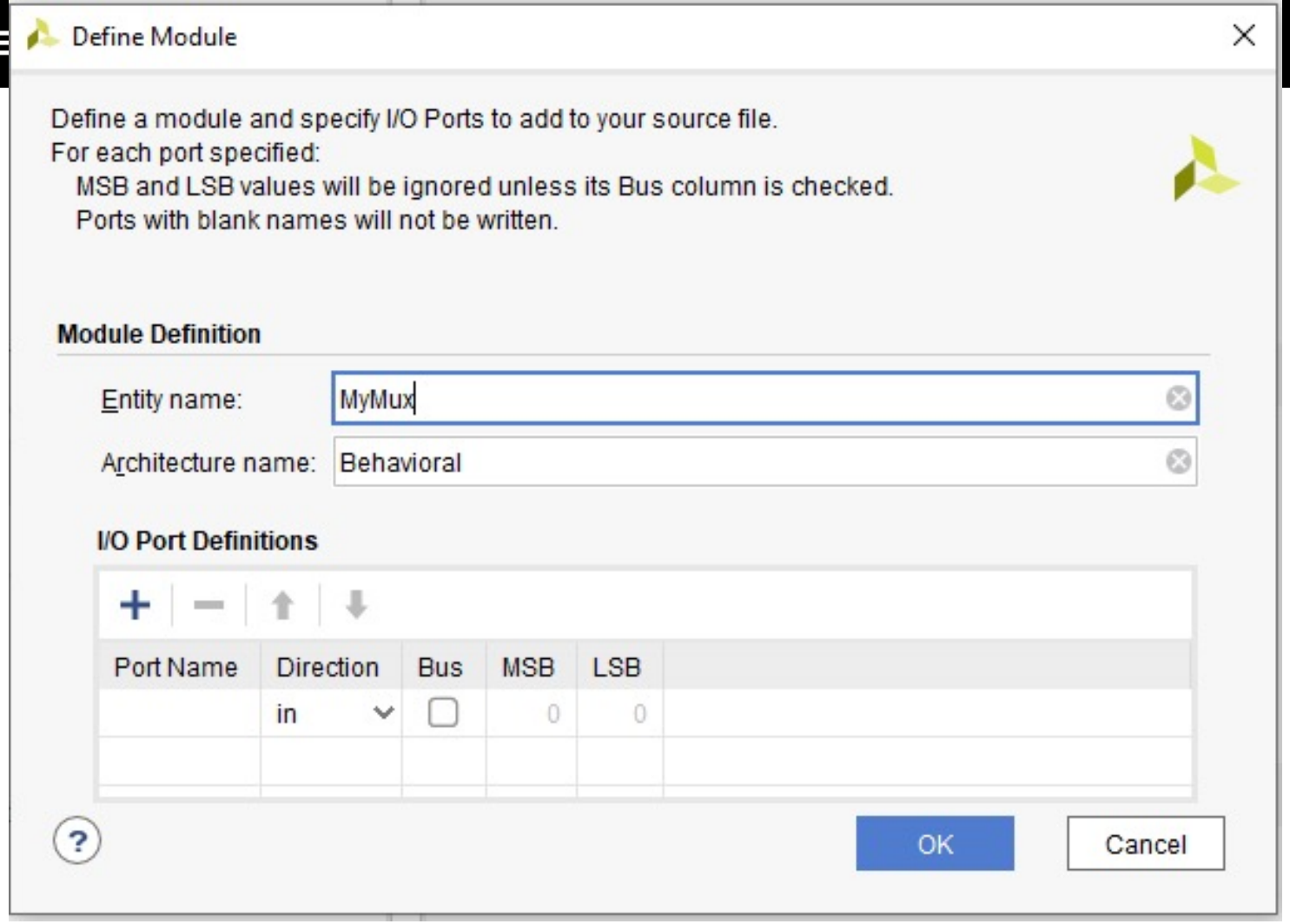
- Compiled
- Runs on a processor
- **Strictly in sequence**
- (Possibly multiple processors)

VHDL

- Simulated (~ effective)
- Translated into hardware
- **Strictly parallel**
- Simulation and hardware differ in several ways.

VHDL “lingo”

- Two things always go together in VHDL
  - Not that “module” is not a VHDL term though!
- **Entity** declaration
  - Defines the external interface to the module
- **Architecture** body
  - The stuff that describes what a module does internally
  - Different types of architectures
  - Multiple architectures can be bound to one entity



In

Out

*InOut – **Avoid, if you do not have a bidirectional signal***

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity MyMux is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           C : in STD_LOGIC;
           Ctrl : in STD_LOGIC;
           Ut : out STD_LOGIC);
end MyMux;

architecture Behavioral of MyMux is

begin
 -- write your
 A = B -- Syntax error is highlighted
 -- code here
end Behavioral;
```

VHDL basics

Example: Simple AND gate

Libraries with built-in functions

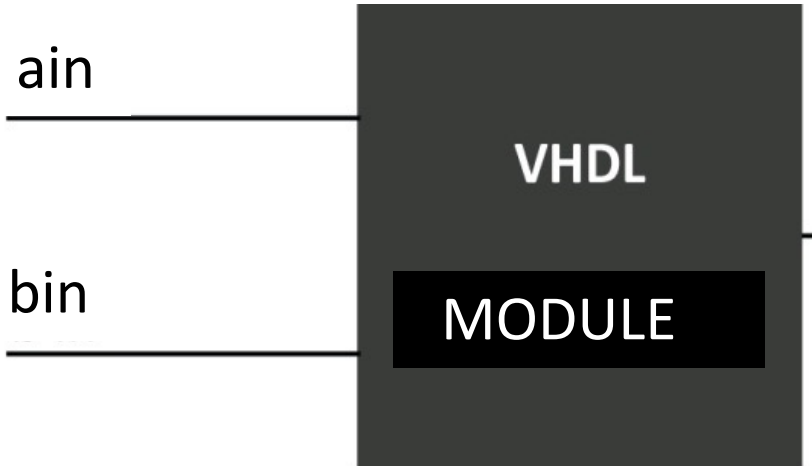
Entity

Architecture

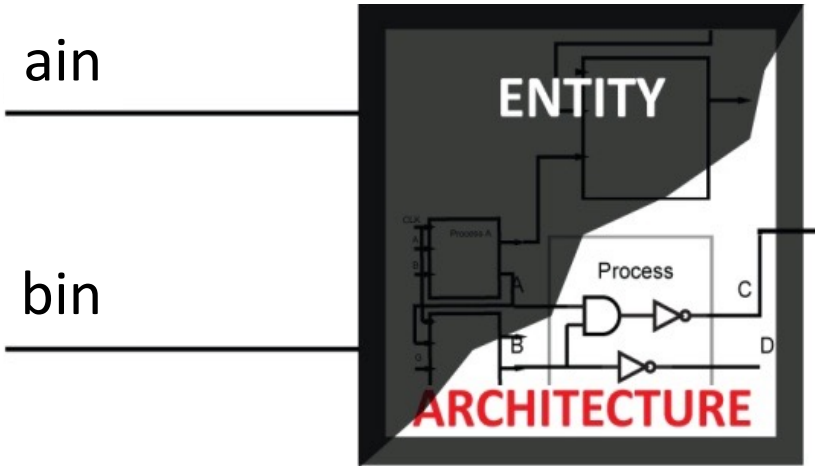
```
-- AND gate
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY and2 IS
   PORT(ain:IN STD_LOGIC;
        bin:IN STD_LOGIC;
        cout:OUT STD_LOGIC);
END and2;

ARCHITECTURE arch_and2 OF and2 IS
BEGIN
   cout<=ain AND bin;
END arch_and2;
```

VHDL - Looks like programming. but the differences are significant

VHDL basics




```
-- AND gate
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY and2 IS
    PORT(ain:IN STD_LOGIC;
         bin:IN STD_LOGIC;
         cout:OUT STD_LOGIC);
END and2;
```




```
ARCHITECTURE arch_and2 OF and2 IS
BEGIN
    cout<=ain AND bin;
END arch_and2;
```

More info : Part-1 VHDL Basic

VHDL basics cont.

Example

Entity

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY and_or IS
     PORT(      a:IN STD_LOGIC;
                b:IN STD_LOGIC;
                c:IN STD_LOGIC;
          y_conc:OUT STD_LOGIC;
           y_seq:OUT STD_LOGIC);
END and_or;
```

If we look at the code from another perspective we can also see two types of code inside the archtiecture

- *Concurrent code*

  Parallel code, things happen at the same time

- Sequential code

  The code is interpreted in a sequence, line by line

  Such code has to be written in a **process**

  The whole process is concurrent with other code

We focus on sequential code!

VHDL basics cont.

Example concurrent vs sequential code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY and_or IS
    PORT(     a:STD_LOGIC;
              b:STD_LOGIC;
              c:IN STD_LOGIC;
        y_conc:OUT STD_LOGIC;
         y_seq:OUT
STD_LOGIC);
END and_or;
```

Architecture

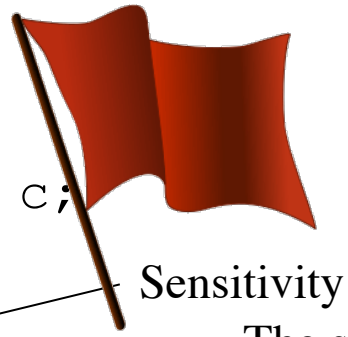
```
ARCHITECTURE arch_and_or OF and_or
IS
SIGNAL x_conc:STD_LOGIC;
SIGNAL x_seq:STD_LOGIC;

BEGIN
    x_conc <= a AND b;
    y_conc <= x_conc OR c;

    seq:PROCESS(a,b,c)
    BEGIN
        x_seq <= a AND b;
        y_seq <= x_seq OR c;
    END PROCESS seq;
END arch_and_or;
```

Internal signals

Concurrent code

Process name

Sequential code



Sensitivity list

The signals that trigger (activate) the process

The sensitivity list ONLY has to do with simulation!

Basic VHDL structures

CASE statement

The CASE statement has similarities to the WITH statement.

The structure is

```
[Case label:]
CASE selectorSignal IS
    WHEN value1 =>
        sequential code;
    WHEN value1 =>
        sequential code;
   [WHEN value2 =>
        sequential code;]
   [WHEN others =>
        sequential code;]
END CASE [Case label];
```

All cases have the same priority

The CASE label is optional but it enhances the readability of the code

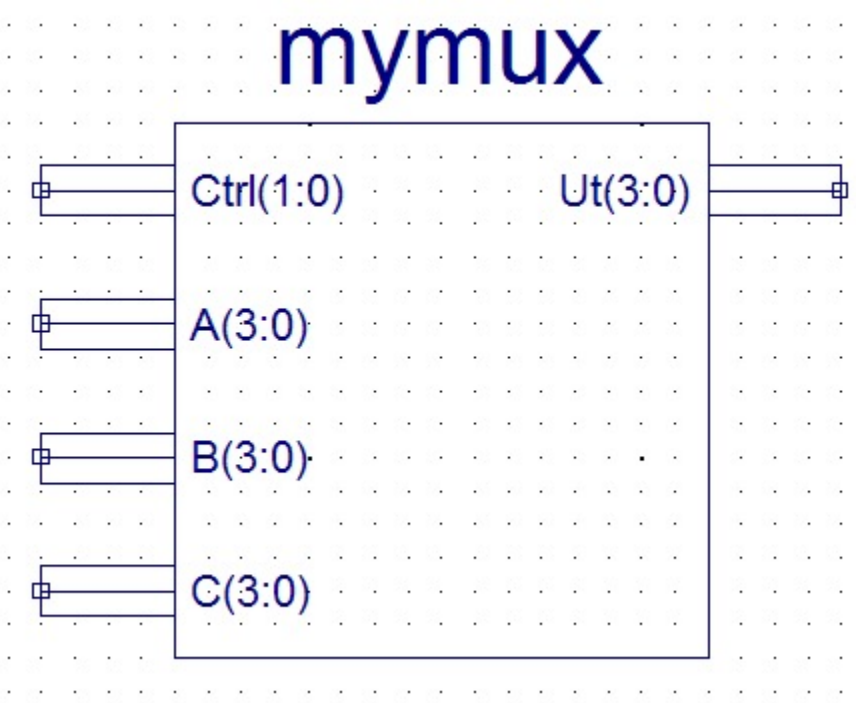
If the `WHEN` cases don´t cover all of the possible values for `selectorSignal` we must include the `OTHERS` clause

VHDL Ex

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;
--*          Led0
--*           ----
--* Led5 |Led6| Led1
--*           ----
--* Led4 |Led3| Led2
--*           ----
entity Bcd2Led is
    Port ( Bcd : in std_logic_vector(3 downto 0);
           Led : out std_logic_vector(6 downto 0));
end Bcd2Led;
```

```
architecture Behavioral of Bcd2Led is

begin

   p1:process(Bcd)
   begin
        case Bcd is
            when "0000"=>
                  Led<="1111110"; -- 0
            when "0001"=> Led<="0110000"; -- 1
            when "0010"=> Led<="1101101"; -- 2
            when others => null;       -- the rest
        end case;
   end process;
end Behavioral;
```

```
entity MyMux is
   Port ( Ctrl : in std_logic_vector(1 downto 0);
          A : in std_logic_vector(3 downto 0);
          B : in std_logic_vector(3 downto 0);
          C : in std_logic_vector(3 downto 0);
          Ut : out std_logic_vector(3 downto 0));
end MyMux;

architecture Behavioral of MyMux is

Begin
p1:process(Ctrl,A,B,C)
   begin
        case Ctrl is
             when "00"=>Ut<=A;
             when "01"=> Ut<=B;
             when "10"=> Ut<=C;
             when others => Ut<="1010";-- the rest
        end case;
   end process p1;
end Behavioral;
```



when "00"=> Ut<=A;

VHDL basics cont.

Basic VHDL structures

IF statement cont.

Examples

BAD

```
IF (a='0') THEN
   y <= '1';
END IF;
```

Basic IF structure. Avoid since we don´t give any value to `y` when `a='1'`.

This means that we need a memory element, a flip-flop, to remember the value when the IF statement is false

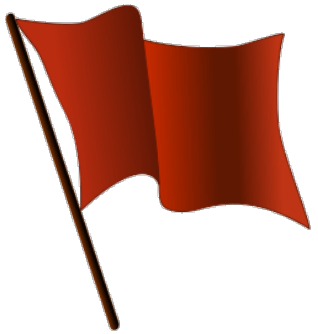
GOOD

```
IF (a='0') THEN
   y <= '1';
ELSE
   y <= '0';
END IF;
```

IF structure with complete assignment. Use this instead.

No memory element needed

VHDL – Synthesis



Clocked processes must be sensitive to one flank.

Use the features **rising_edge(clk)**

- Make plausibility analysis on number of gates and number of latches. Study the gate schematic after synthesis.
- ”Think Hardware”

– Think of hardware when writing the VHDL code.
(unfortunately requires training and experience)

NOTE! Synthesis! The simulator can do without

VHDL basics cont.

Free in Xilinx

Basic VHDL structures

Synchronous code with asynchronous reset

Basic structure

```
async_reset:PROCESS(clk,reset)
BEGIN
   IF (reset = '1') THEN
      asynchronous_reset_code;
   ELSIF (rising_edge(clk)) THEN
      synchronous_main_code;
   END IF;
END PROCESS async_reset;
```

Both signals must be able to trigger the process

No other signals should be included in the sensitivity list

Level triggered `reset`

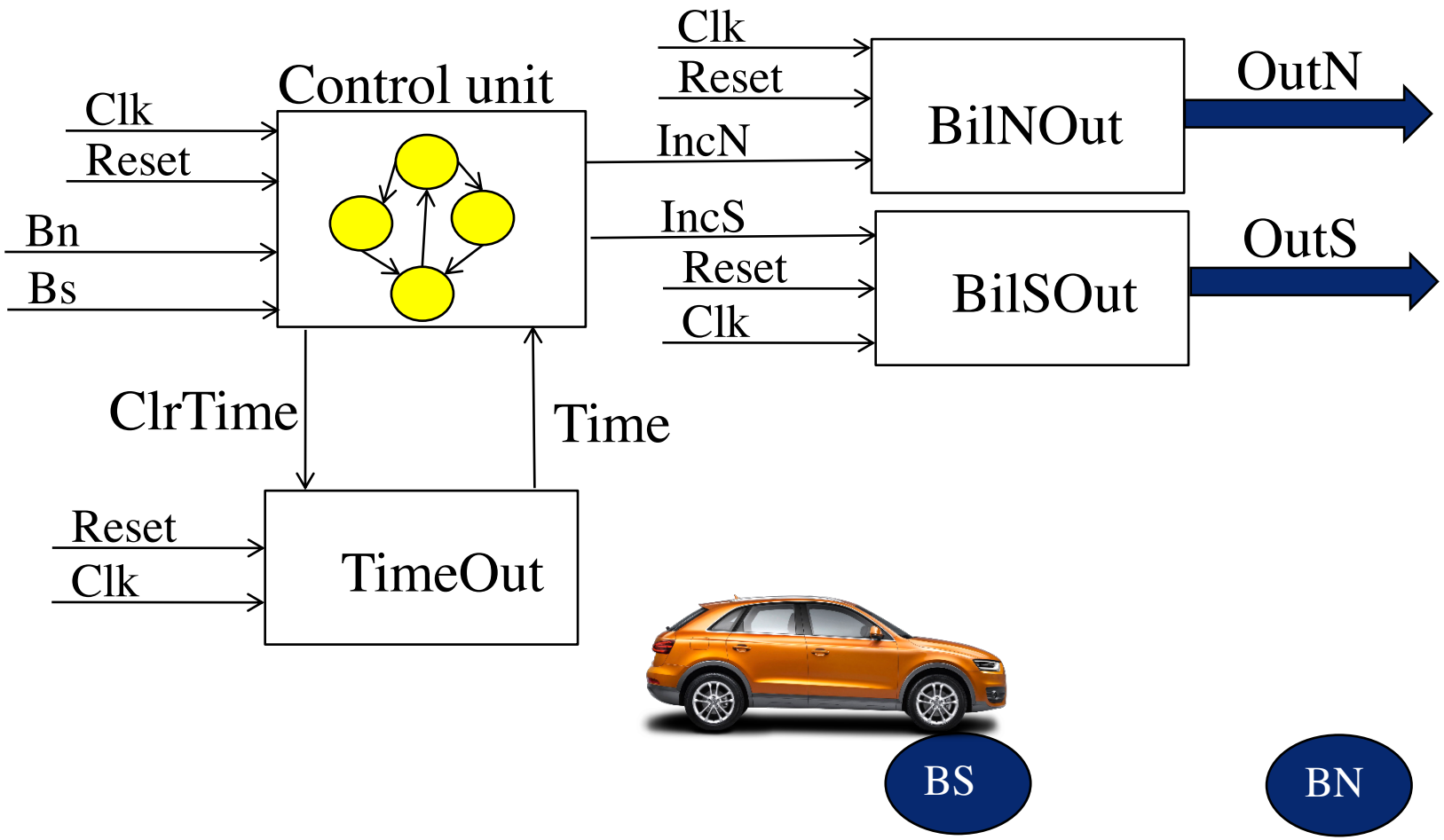
Positively edge triggered `clk` signal

rising_edge(clk)

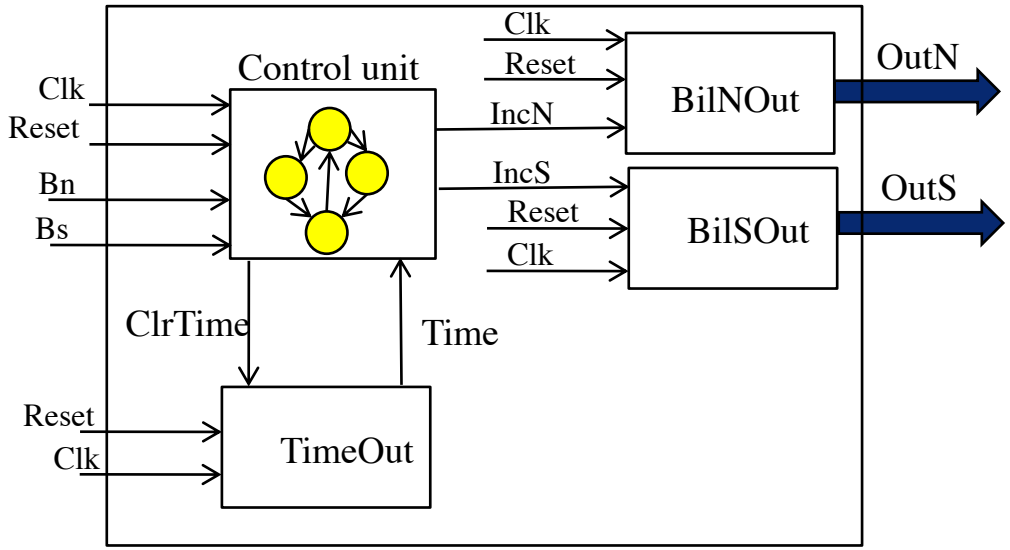
Think in hardware when writing VHDL. Ex

- A traffic counter should be designed.
- It shall calculate the number of wheels traveling north and the number traveling south.
- When a car travels over the sensor, a pulse of *Bn* and *Bs* is generated.
- Should only one pulse occur, *Bn* or *Bs*, the system should return after a timeout and wait for a new pulse.

Traffic counter

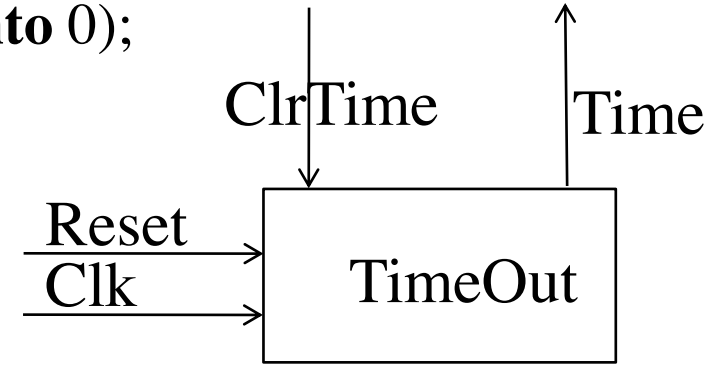


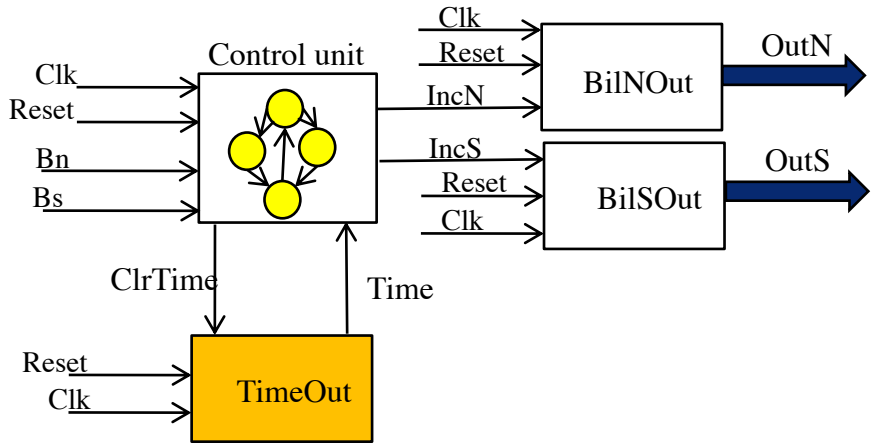
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
use ieee.std_logic_unsigned.all;


ENTITY TrafikM IS
 generic (nc : NATURAL := 8;
       nt : NATURAL := 3);
  PORT(Reset:IN STD_LOGIC;
      Clk:IN STD_LOGIC;
      Bn:IN STD_LOGIC;
      Bs:IN STD_LOGIC;
      Outn : out std_logic_vector (nc-1 downto 0);
      Outs : out std_logic_vector (nc-1 downto 0));
END TrafikM;
```



```
ARCHITECTURE arch_TrafikM OF TrafikM IS
SIGNAL Time : std_logic_vector (nt-1 downto 0);
BEGIN
TimeOut : process (Clk, Reset) is
    variable cnt : std_logic_vector (nt-1 downto 0);
    begin
      if Reset = '1' then
        cnt := (others => '0');
      elsif rising_edge(Clk) then
        if ClrTime='1' then
          cnt := ( others => '0');
        else
          cnt := cnt+1;
        end if; -- ClrTime
      end if;
      Time <= cnt;
end process TimeOut;
```

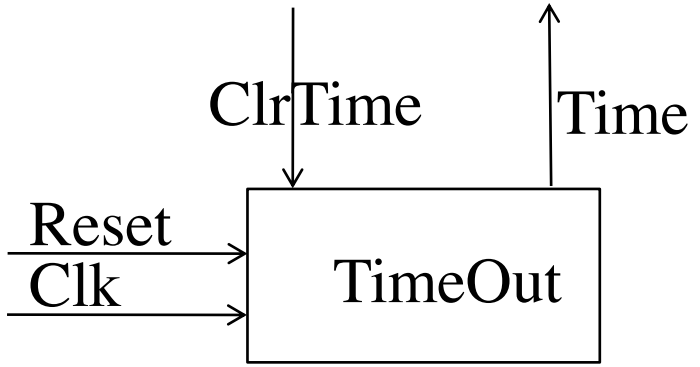
1 process

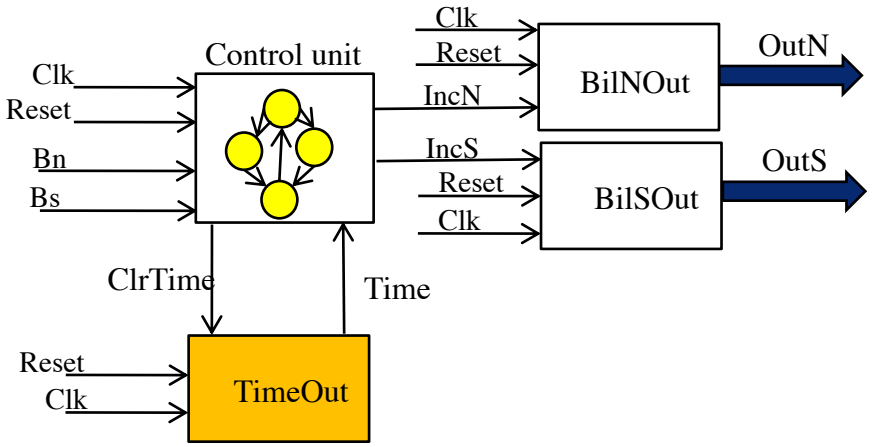




```
CMem : process (Clk, Reset) is
begin
      if Reset = '1' then
        Time <= (others => '0');
      elsif rising_edge(Clk) then
        Time <= Next_TC;
      end if;
end process CMem;

  TimeOut : process (ClrTime, Time) is
    begin
      if ClrTime='1' then
        Next_TC <= ( others => '0');
      else
        Next_TC <= Time + 1;
      end if; -- ClrTime
end process TimeOut;
```

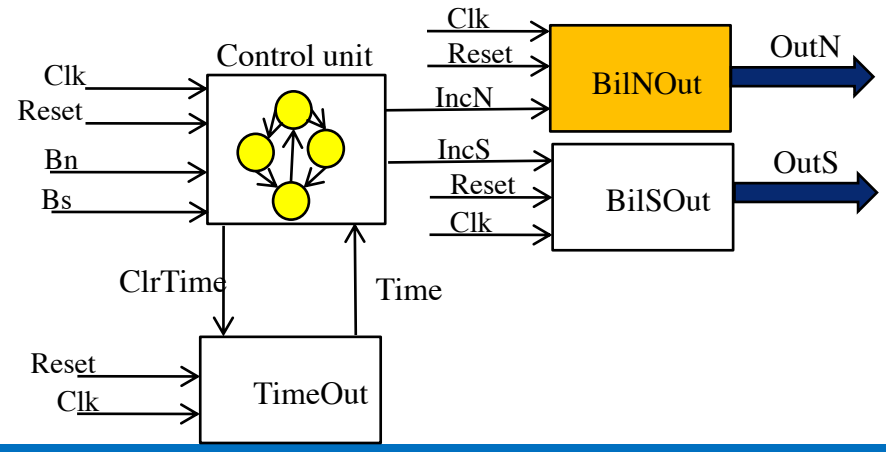
2 processes

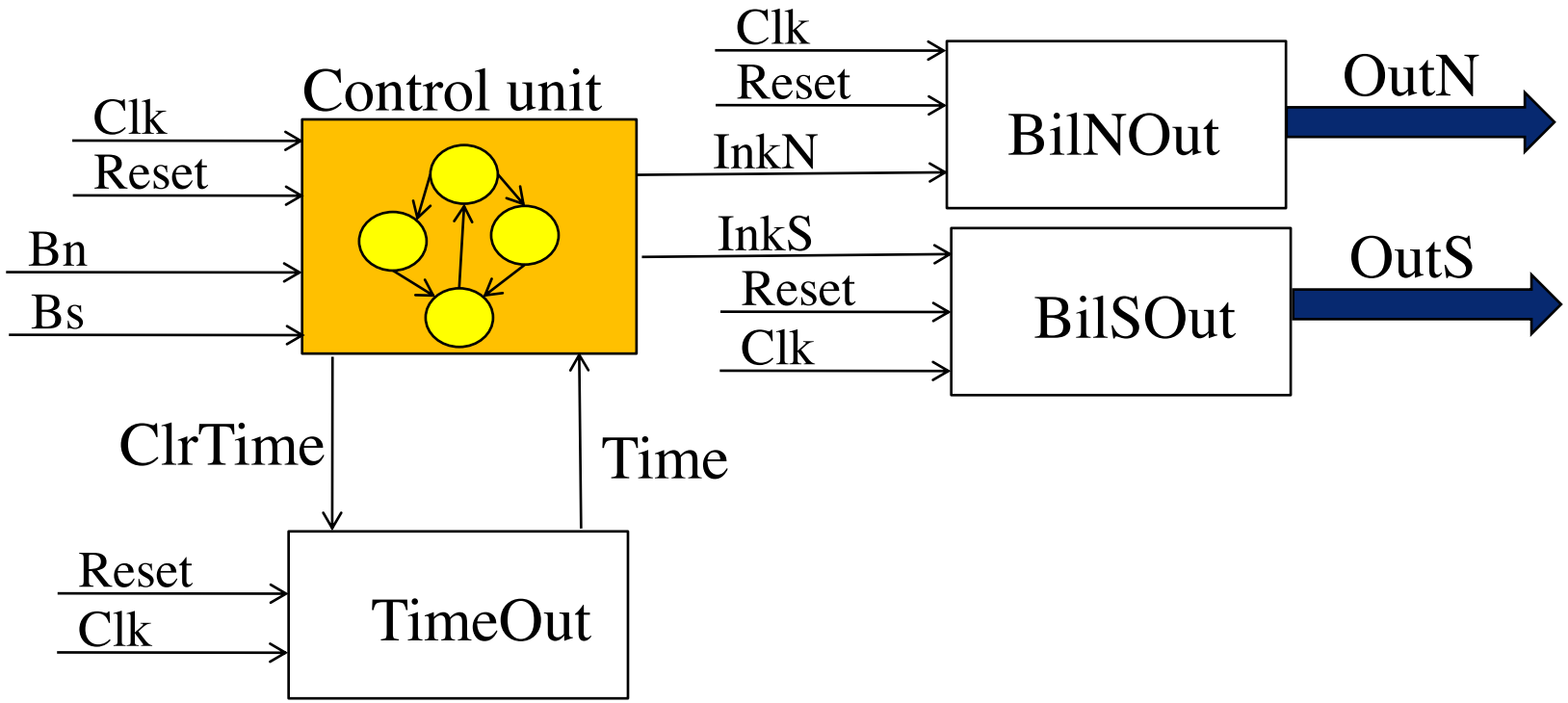




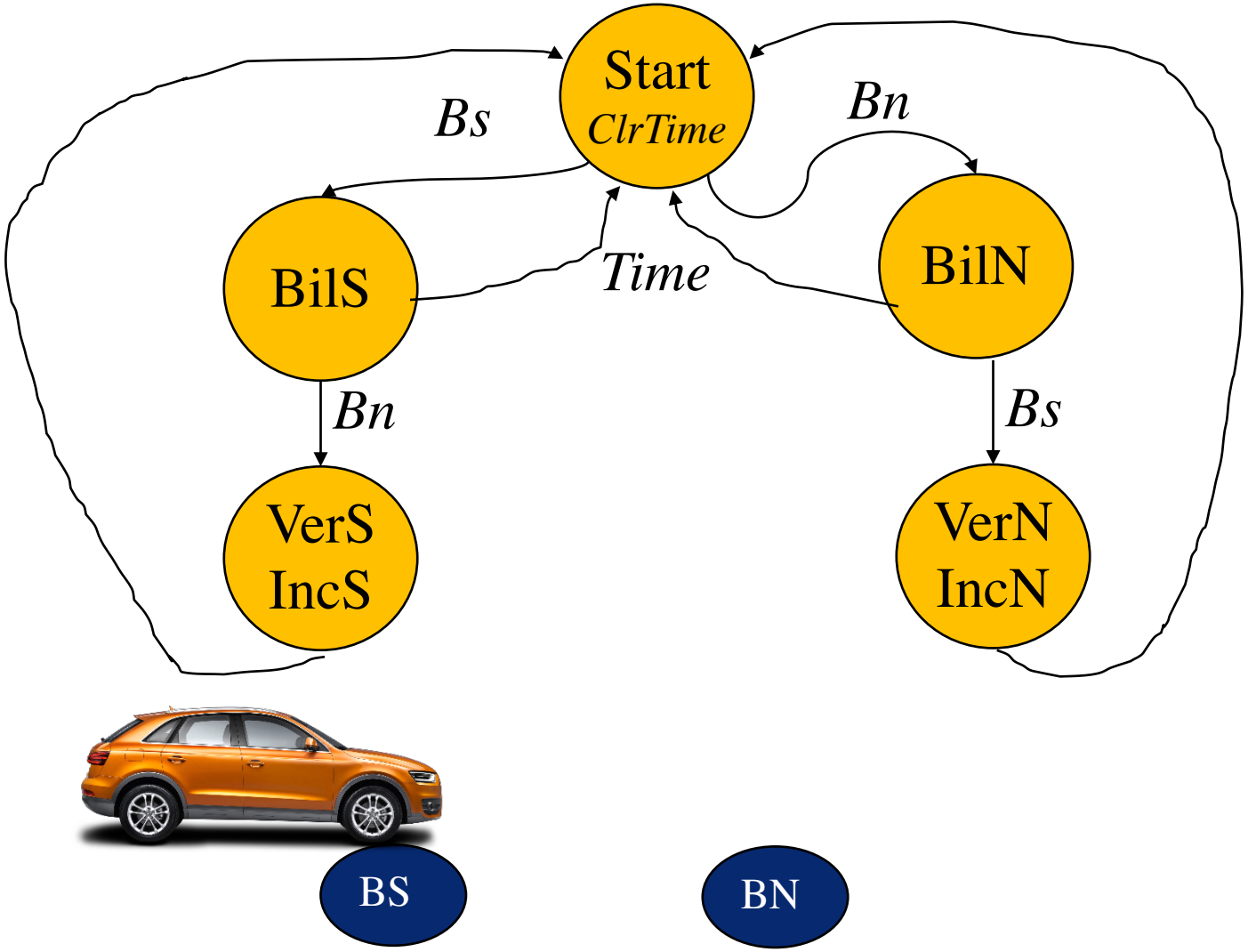
```
ARCHITECTURE arch_TrafikM2 OF TrafikM2 IS
SIGNAL Norr_Cnt, Next_Norr
    : std_logic_vector (nc-1 downto 0);
…
BEGIN
CMem : process (Clk, Reset) is
begin
    if Reset = '1' then
      Time <= (others => '0');
      Norr_Cnt <= (others => '0');
    elsif rising_edge(Clk) then
      Time <= Next_TC;
      Norr_Cnt <= Next_Norr;
    end if;
end process CMem;
```

```
BilNOut : process (IncN, Norr_Cnt) is
begin
    if IncN='1' then
       Next_Norr <= Norr_Cnt+1;
    else
      Next_Norr <= Norr_Cnt;
    end if; -- CnUppS
end process BilNOut;
OutN <= Norr_Cnt; -- OBS!!
```

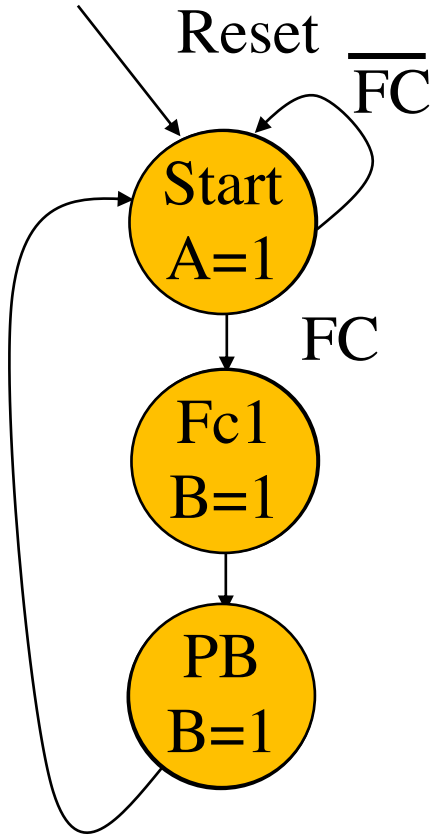





```
ARCHITECTURE arch_TrafikM2 OF TrafikM2 IS
TYPE state_type is (Start, BilS, BilN, VerS, VerN);
SIGNAL State, Next_State : state_type;
SIGNAL Time, Next_TC : std_logic_vector (nt-1 downto 0);
SIGNAL Norr_Cnt, Next_Norr, Sod_Cnt, Next_Sod : std_logic_vector (nc-1 downto 0);
SIGNAL ClrTime, IncS, IncN  : STD_LOGIC;
BEGIN
```



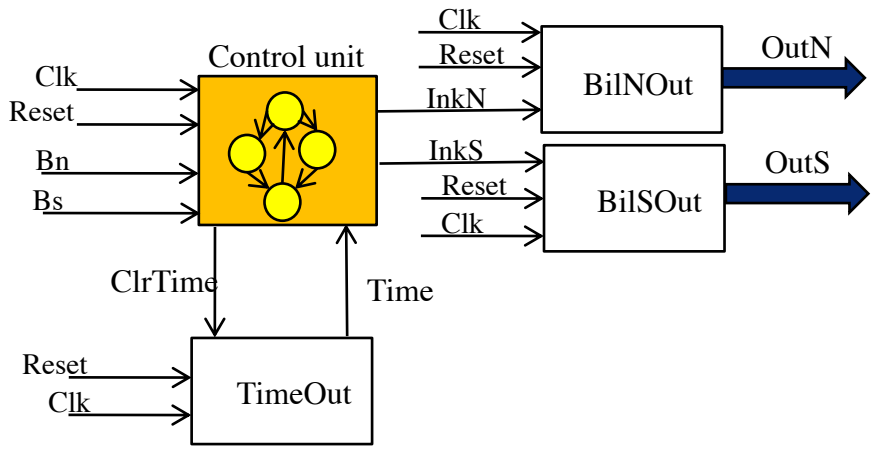
To VHDL



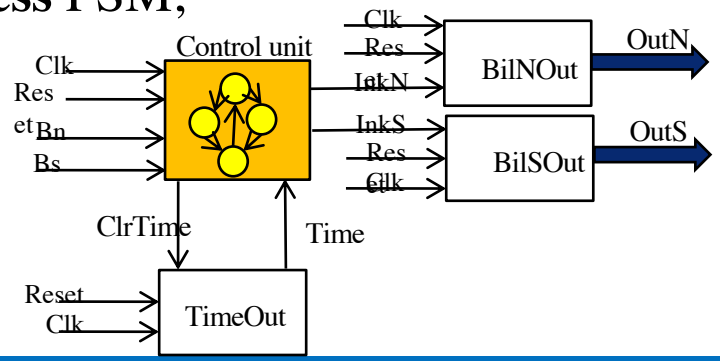

```
Kombinatorisk:process(state,FC)
begin
  A<='0'; B<='0' -- DEFAULT
  case state is
     when Start=>
         A<='1';
         if FC ='1' then next_state<= Fc1;
          else next_state<=Start;
         end if;
      when FC1=>
            B<='1';
            next_state<= PB;
      when PB=>
            B<='1';
            next_state<= Start;
  end case;
end process Kombinatorisk;
```

```
p1:process(Clk)
begin
  if Clk'event and Clk='1' then
    if Reset='1' then
      state<=Start;
    else
      state<=next_state;
  end if;
end process p1;
```

```
CMem : process (Clk, Reset) is
   begin
     if Reset = '1' then
       Time <= (others => '0');
       Norr_Cnt <= (others => '0');
       Sod_Cnt <=  (others => '0');
       State <= Start;
     elsif rising_edge(Clk) then
       Time <= Next_TC;
        Norr_Cnt <= Next_Norr;
        Sod_Cnt <= Next_Sod;
        State <= Next_State;
     end if;
end process CMem;
```

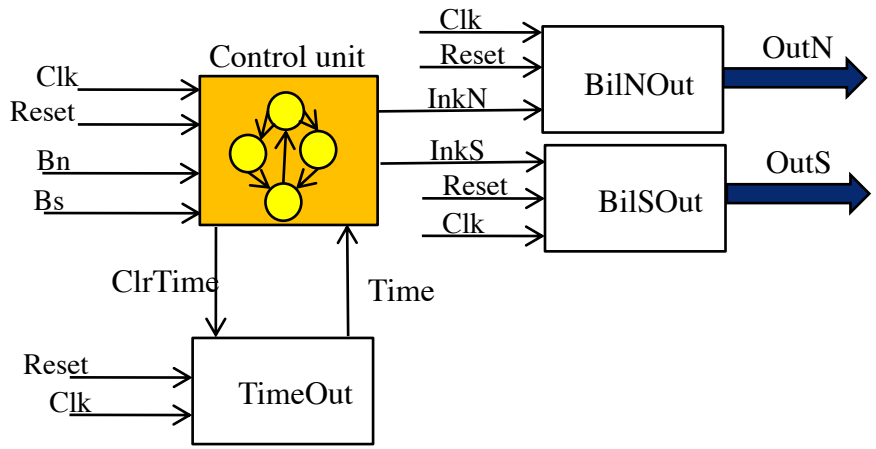


```
FSM : process (State, Bn, Bs, Time) is
 begin
  ClrTime <='0';IncS<='0';IncN<='0';
  case State is
   when Start =>
    ClrTime <='1';
    if Bs='1' then
         Next_State <= BilS;
    elsif BN='1' then
         Next_State <= BilN;
    else
         Next_State <= Start;
    end if;
   when BilS =>
    if Bn='1' then
         Next_State <= VerS;
    elsif Time(nt-1)='1' then
         Next_State <= Start;
    else Next_State <= BilS; end if;
   when BilN =>
    if Bs='1' then Next_State <= VerN;
    elsif Time(nt-1)='1' then
         Next_State <= Start;
    else Next_State <= BilN;
    end if;
   when VerS =>  IncS<='1';
    Next_State <= Start;
   when VerN =>  IncN<='1';
    Next_State <= Start;
   when others =>
    Next_State <= Start;
  end case;
 end process FSM;
```



```
FSM : process (State, Bn, Bs, Time) is
 begin
  ClrTime <='0';IncS<='0';IncN<='0';
  Next_State <= Start;
  case State is
   when Start =>
    ClrTime <='1';
    if Bs='1' then
        Next_State <= BilS;
    elsif BN='1' then
        Next_State <= BilN;
    end if;
   when BilS =>
    if Bn='1' then
        Next_State <= VerS;
    elsif Time(nt-1)='0' then
        Next_State <= BilS;
    end if;
```

```
when BilN =>
    if Bs='1' then Next_State <= VerN;
    elsif Time(nt-1)='0' then
        Next_State <=  BilN;
    end if;
   when VerS =>  IncS<='1';
   when VerN =>  IncN<='1';
   when others =>
  end case;
end process FSM;
```



Package

- To reduce the need for declarations
- To use the same types everywhere
- To use common features

- For the more advanced
- Use records declared in package

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

PACKAGE MyPac IS
COMPONENT full_adder IS
    PORT(    a:IN STD_LOGIC;
             b:IN STD_LOGIC;
           cin:IN STD_LOGIC;
             y:OUT STD_LOGIC
          cout:OUT STD_LOGIC);
END COMPONENT full_adder;
END MyPac;
```

USE work.MyPac.all;

Add:
Generics
Constants
Types
Component
Declarations

HERE!

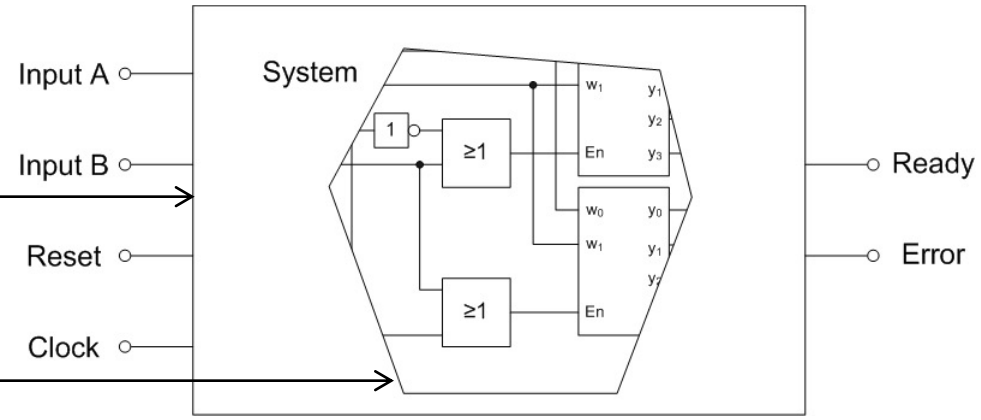
Summary

VHDL is used for

Description / Specification / Documentation of hardware.

- Simulation of hardware.
- Synthesis of hardware.

Entity - Architecture



Summary cont.

- VHDL – A "strongly typed" language
- Library :
  Library IEEE;
  use IEEE.std_logic_1164.all;
  use ieee.std_logic_unsigned.all;

  use ieee.numeric_std.all; ← Newer better
- Signal assignments <= *(At END of PROCESS)*
- Variable assignments := *(Immediately)*
- *It is possible to use variables to simplify for oneself.*

Summary cont.

”Implicit memory”

- The value is retained if no assignment
- D-flip-flop (Register / counter etc.)
- **IF rising_edge(Clk) THEN**
- Asyncronous Reset

```
cnt2 : process (Clk, Reset) is
begin
  if Reset = '1' then <Reset Statements >
 elsif rising_edge(Clk) then
    <Statements>;
 end if;
```

Summary cont.

” Synchronous Reset”

- cnt2 : **process** (Clk) **is**
  begin
  if rising_edge(Clk) then
  if Reset = '1' **then** <Reset Statements >
  else
  <Statements>;
  end if; -- Reset
  end if; -- Clk'EVENT

To do:

- Tutoring times.
- Mo 13-15, Wh 10-12, Fr 10-12 *(week 3)*
- *Select project - as soon as possible*

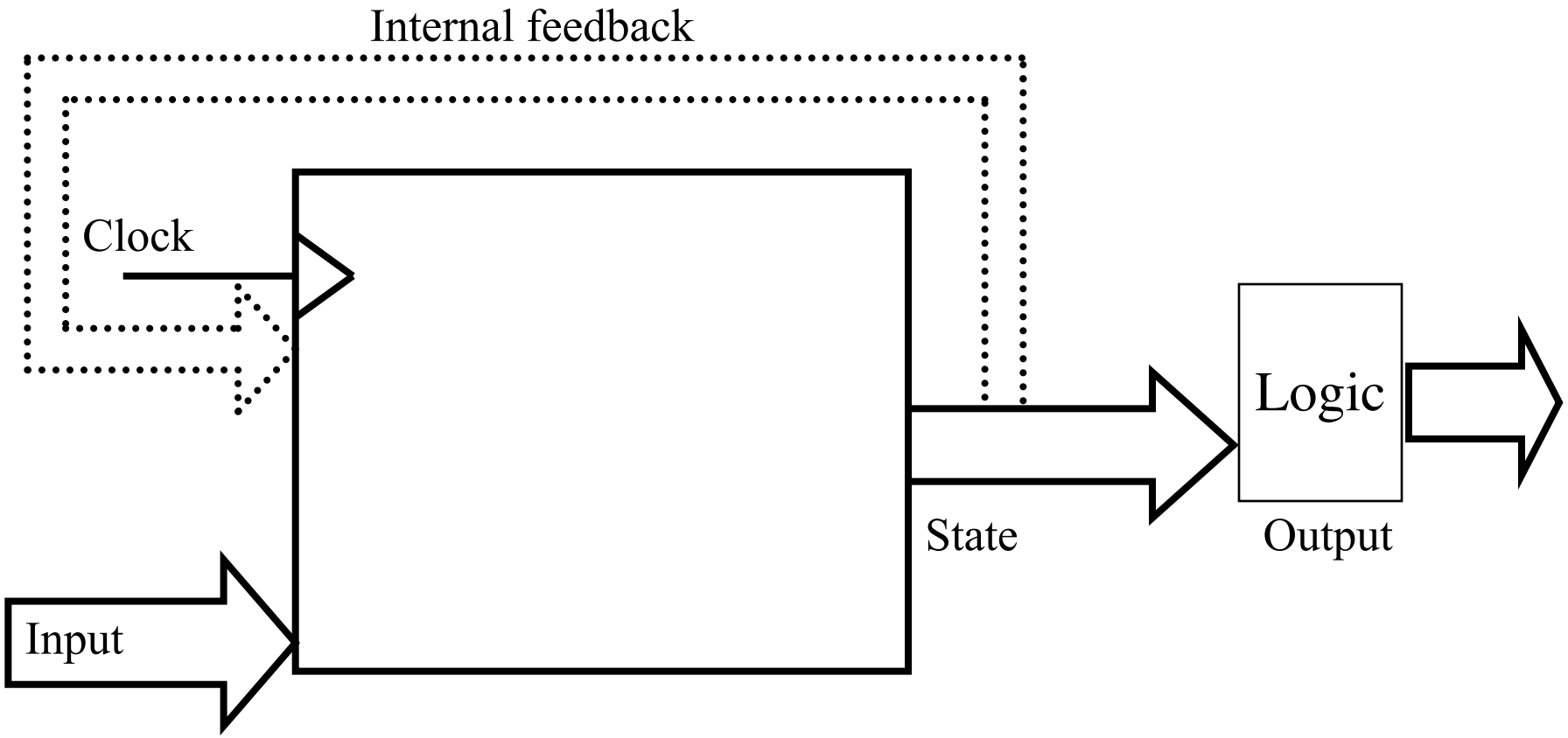
(You who have not done so yet)

- Book time for weekly meeting

Block diagram!

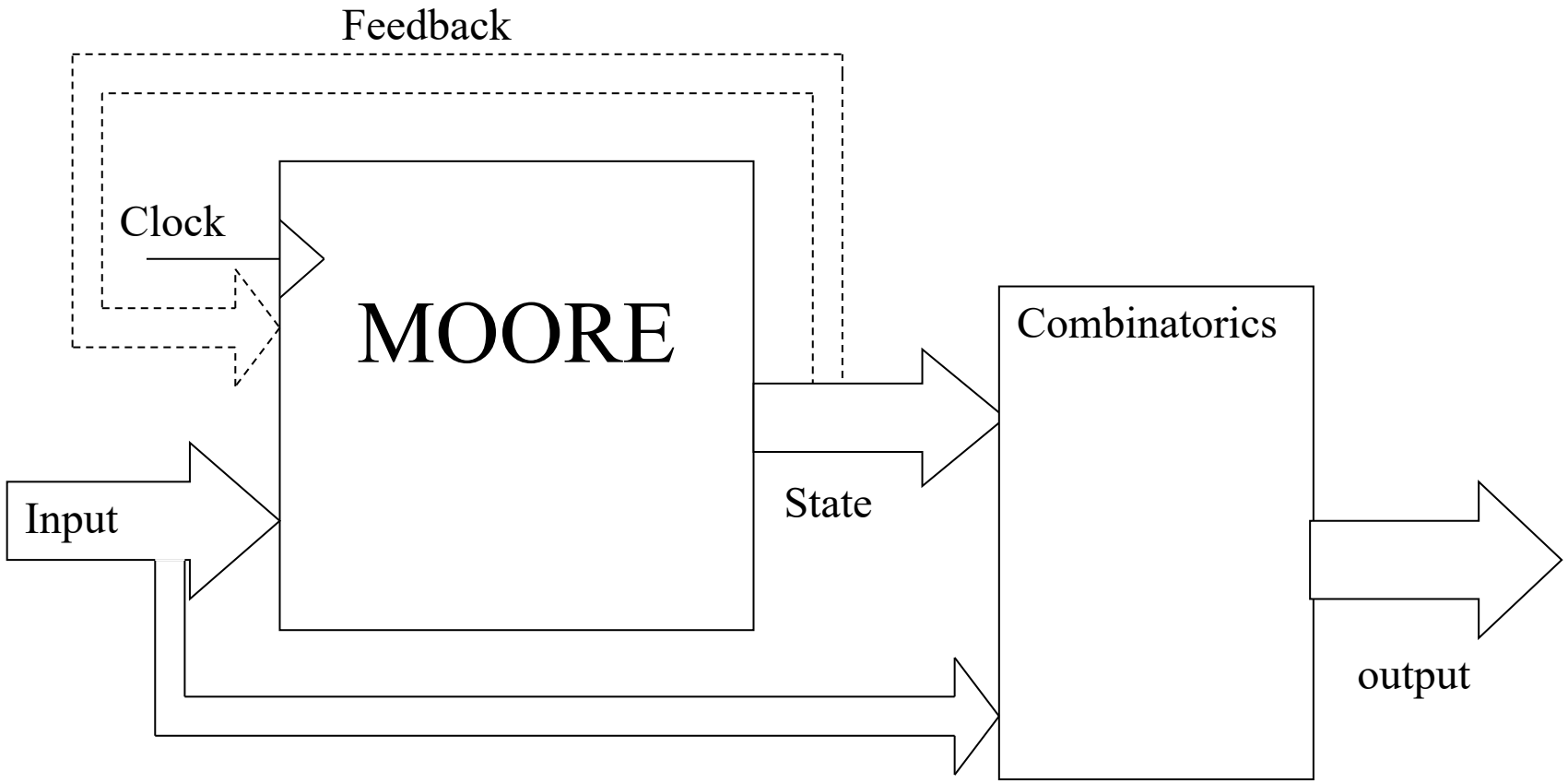
State machine

Moore model

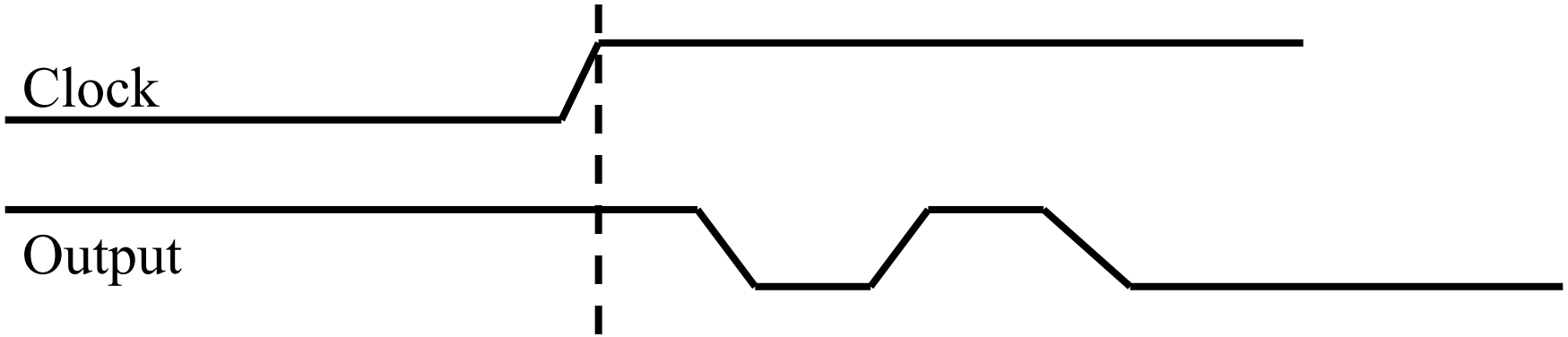


State machine

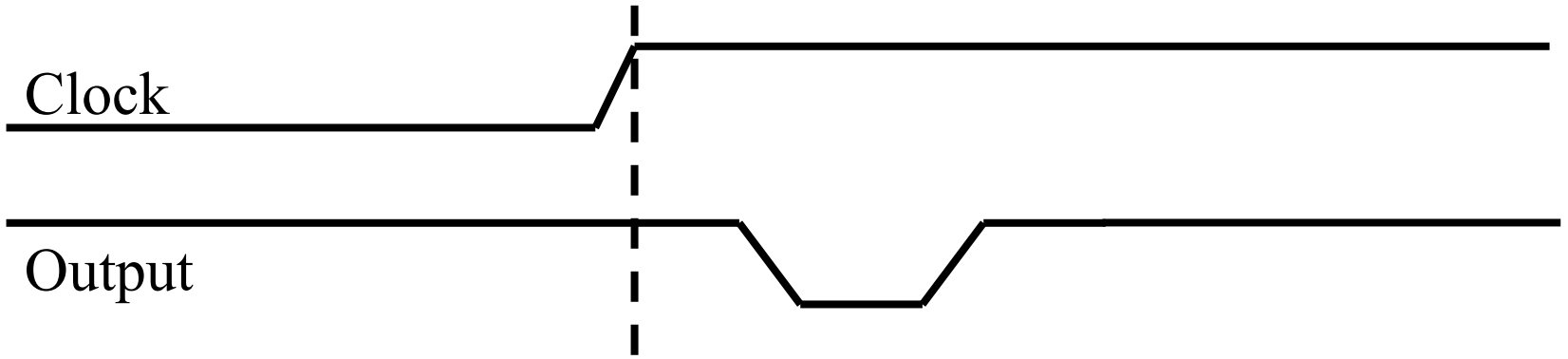
Mealy model



Hazard



Worse, more common

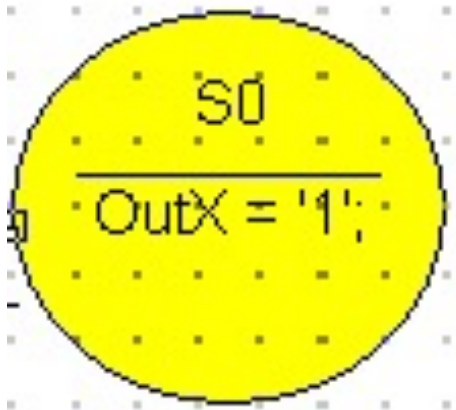


State machine

2-ways to get the output

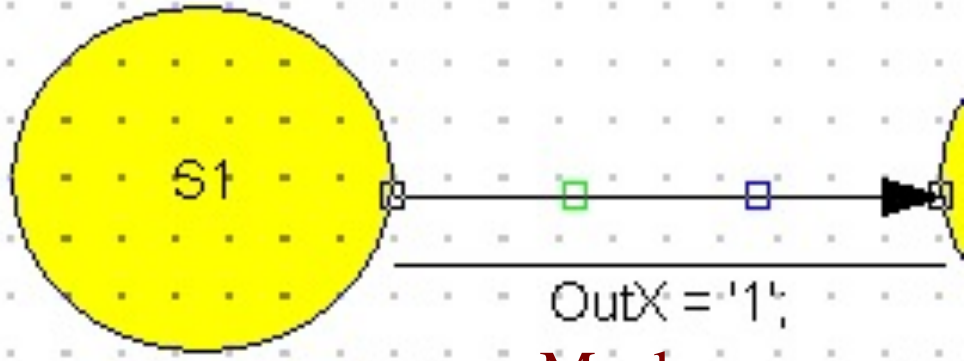
```
WHEN zero=>
   O<="01"; -- Moore output
   IF I='1' THEN
      next_state <=one1;
    O<="10"; -- Mealy output
   END IF;
```

State action

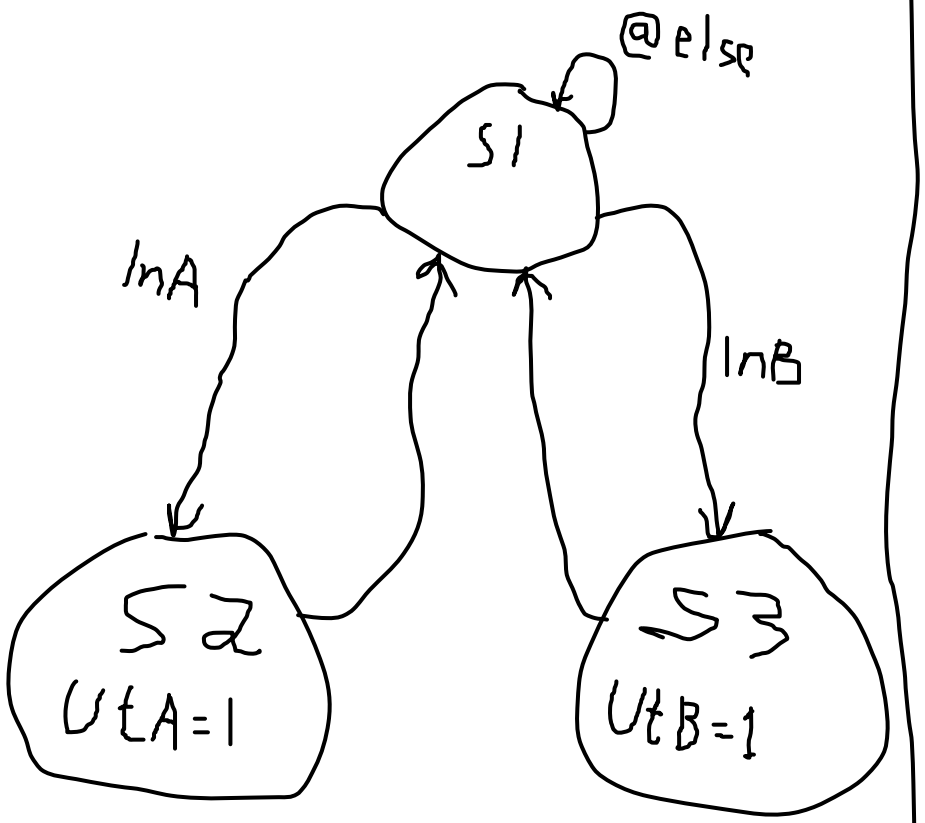


Moore

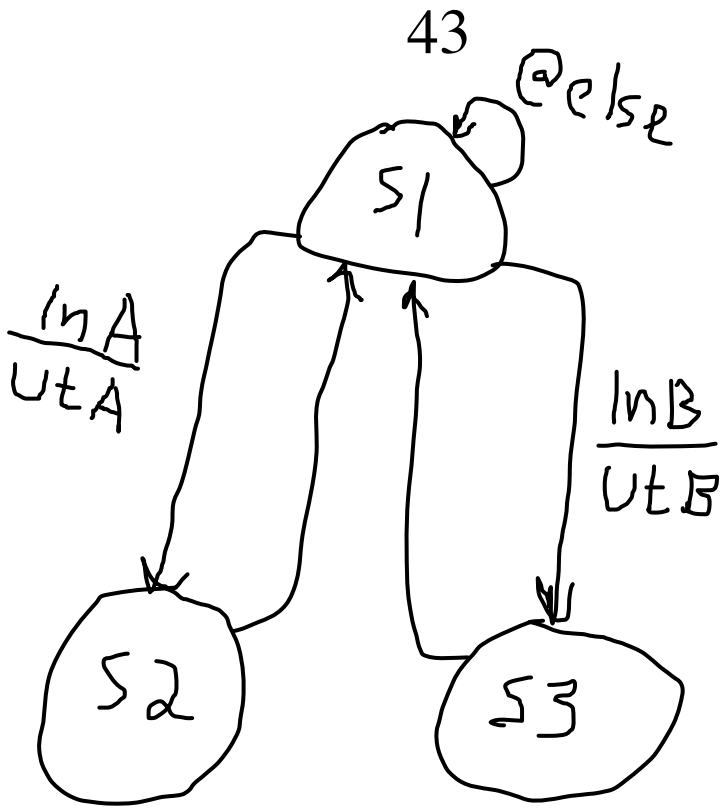
Transaction action



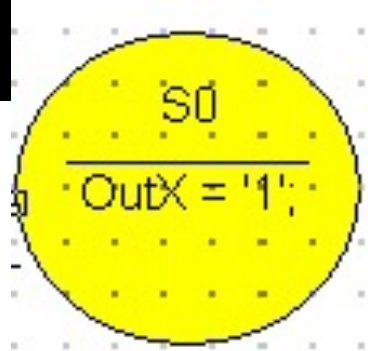
Mealy

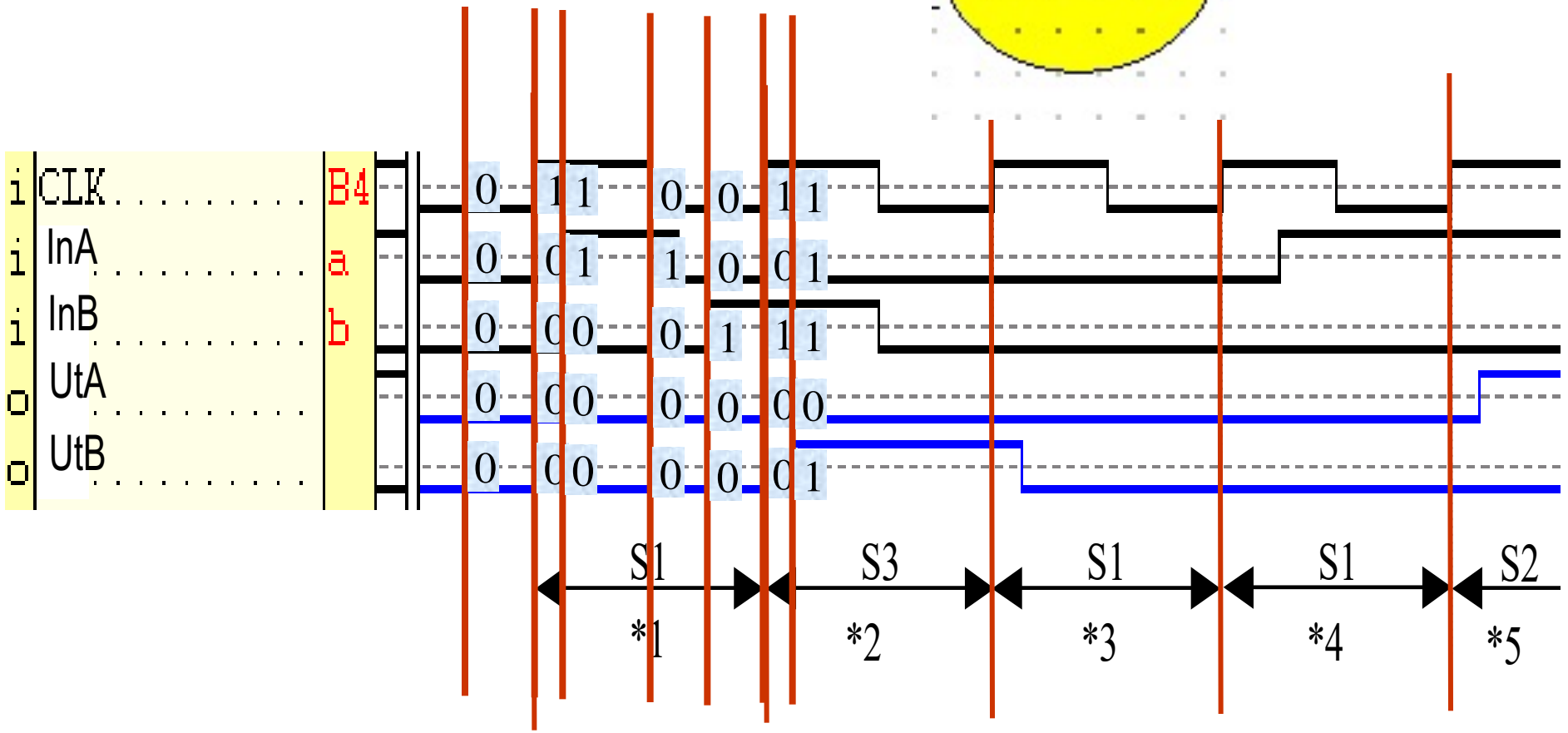


43

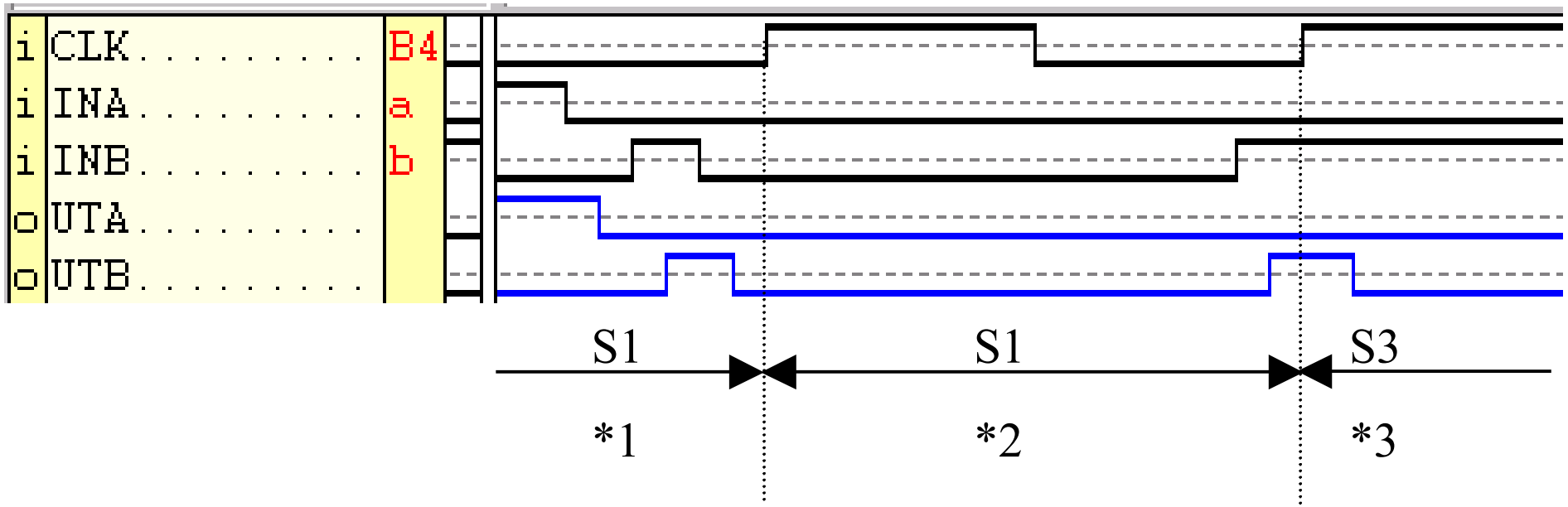


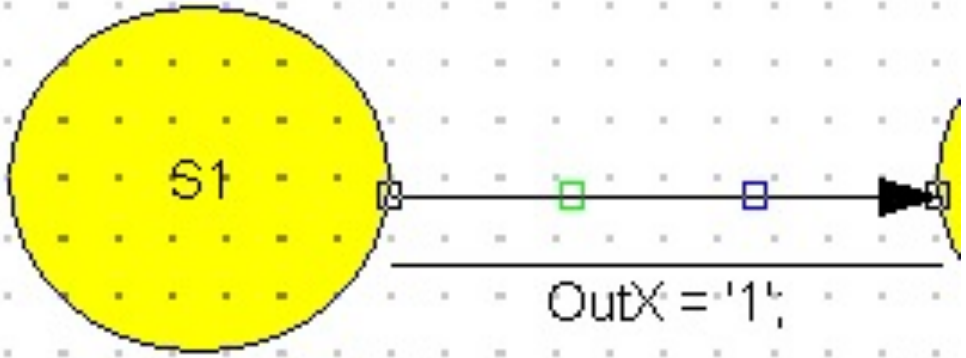
State action (Moore)





Transaction action (Mealy)





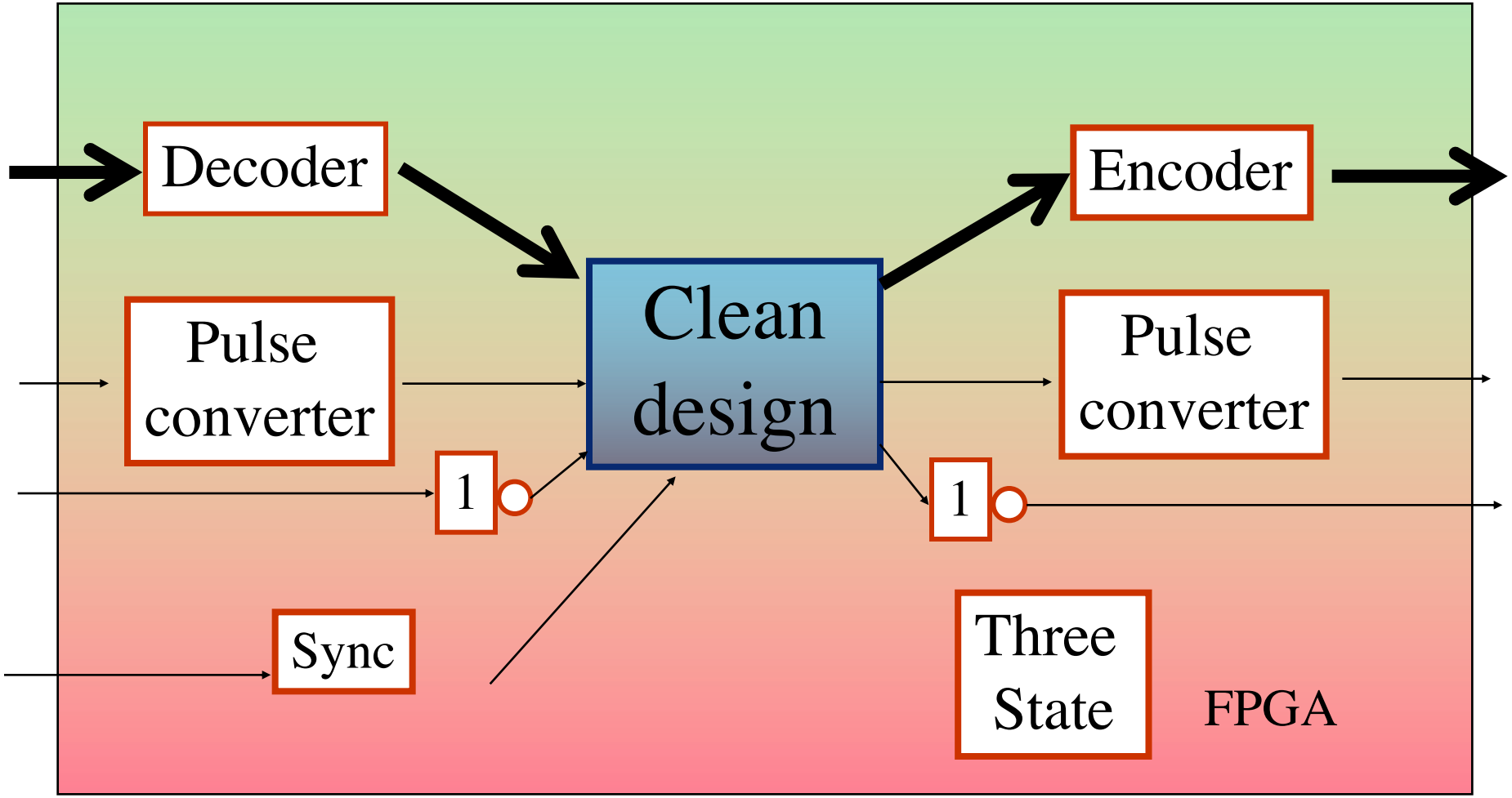
Summary state machines

```
-- Synchronous process
ASM_P: PROCESS (Clk,Reset)
BEGIN
  IF Reset = '1' THEN -- Asynkron
reset
     state<=zero; -- Reset tillstand
  ELSIF rising_edge(CLK) THEN
             state<=next_state;
          SO<= O -- Synkron Mealy output
   END IF;
END PROCESS ASM_P;
```

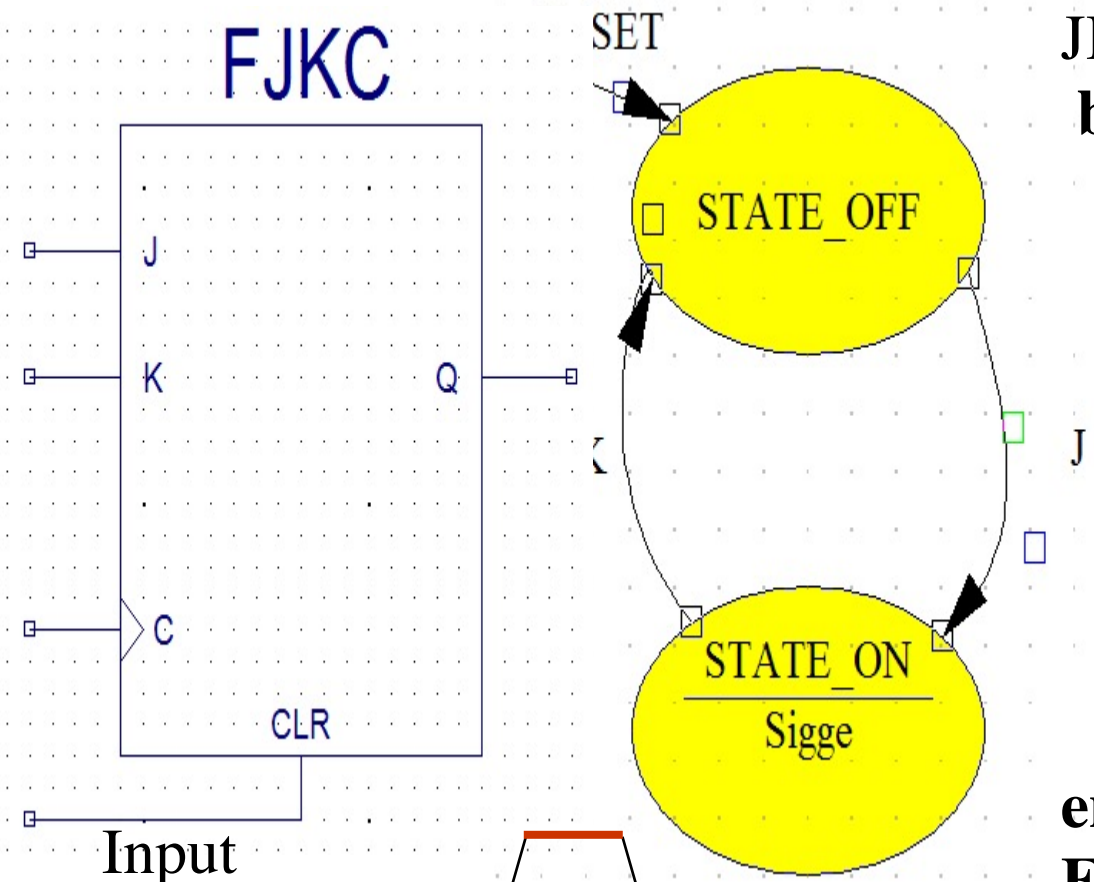
```
next_s:PROCESS (state,I)
BEGIN
next_state <=zero; -- default
O<="00";
  CASE state IS
    WHEN zero=>
       O<="01"; -- Moore output
       IF I='1' THEN
          next_state <=one1;
        O<="10"; -- Mealy output
       END IF;
```

• • •

Structure

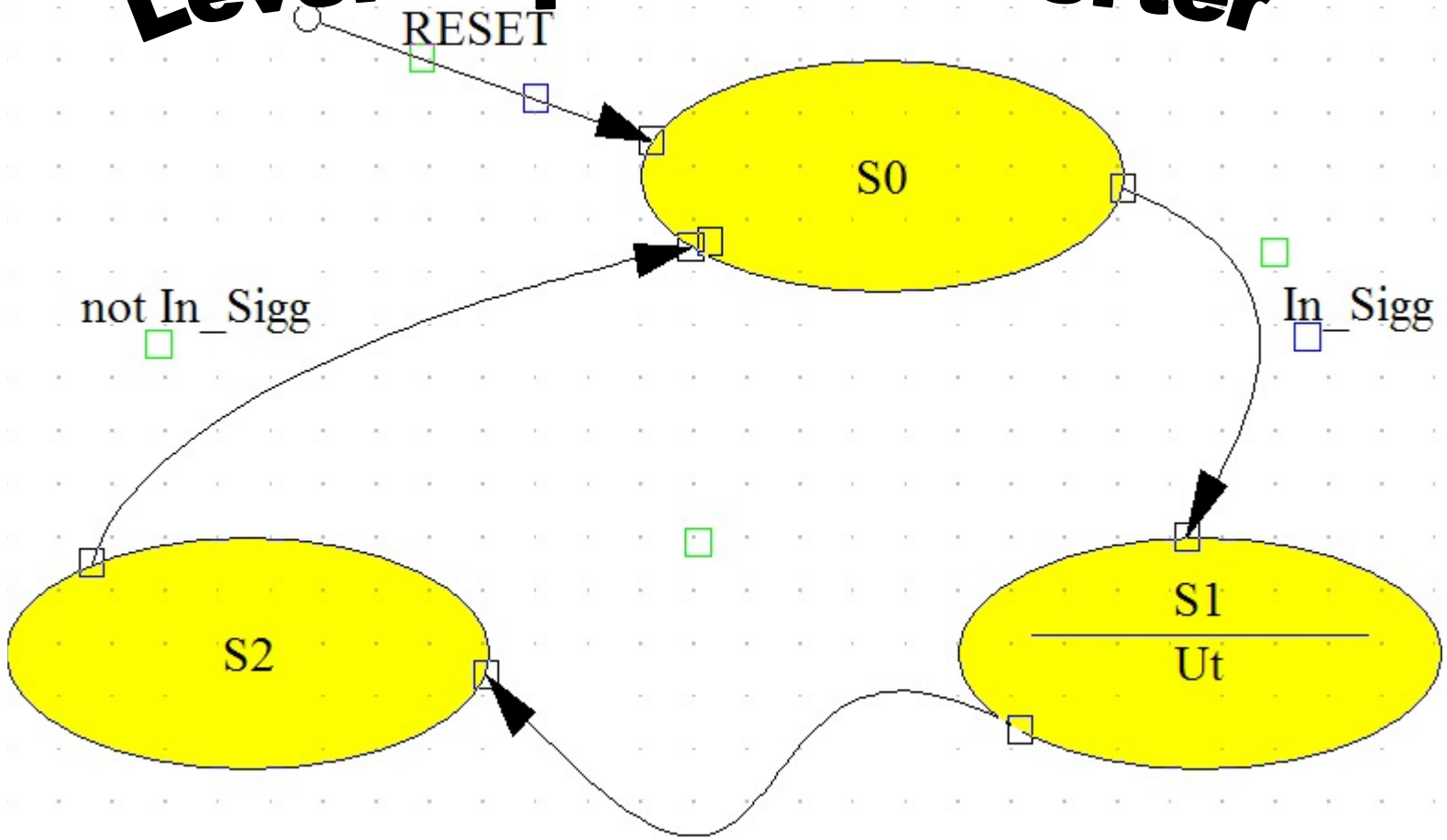


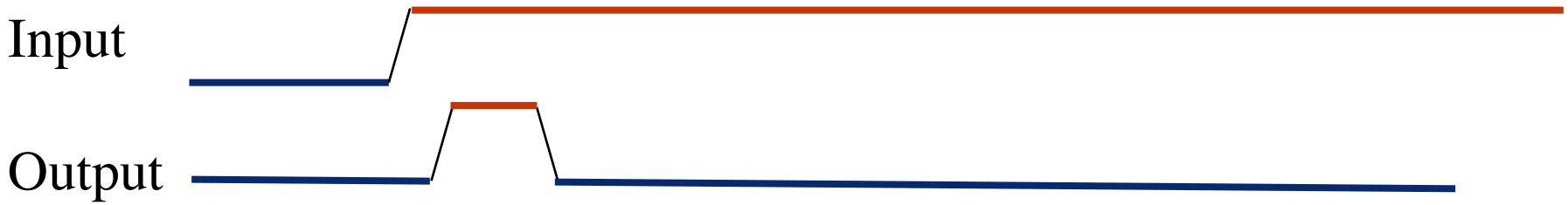
Pulse to level converter

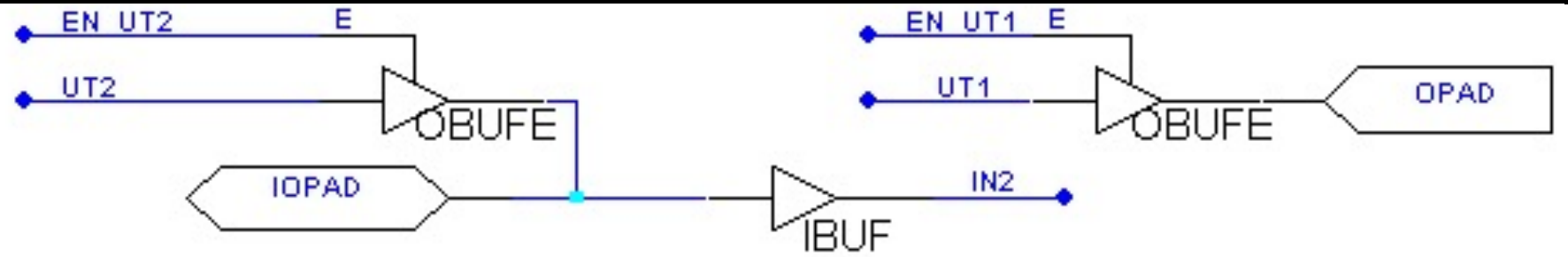



```
JKVippa : process (Clk, Reset) is
 begin
   if Reset = '1' then Y <= '0';
   elsif rising_edge(Clk) then
     if J ='1' and K='1' then
         Q <=not Q;
     elsif J ='1'  then Q <='1';
     elsif K ='1'  then Q <='0';
     end if; -- JK
   end if; -- Reset
end process JKVippa;
END arch_FPGA_D;
```

Level to pulse converter





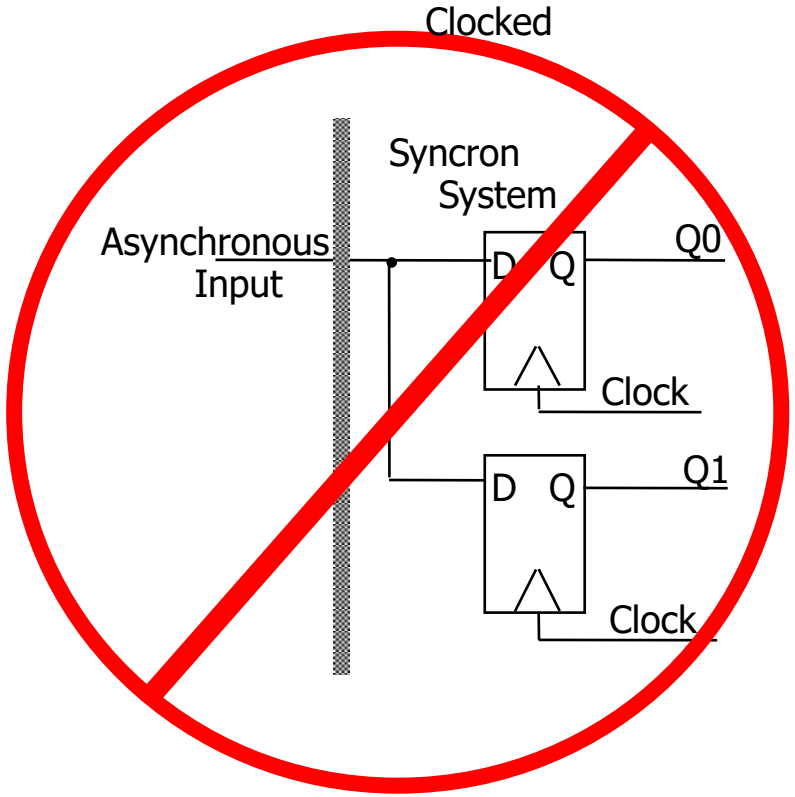


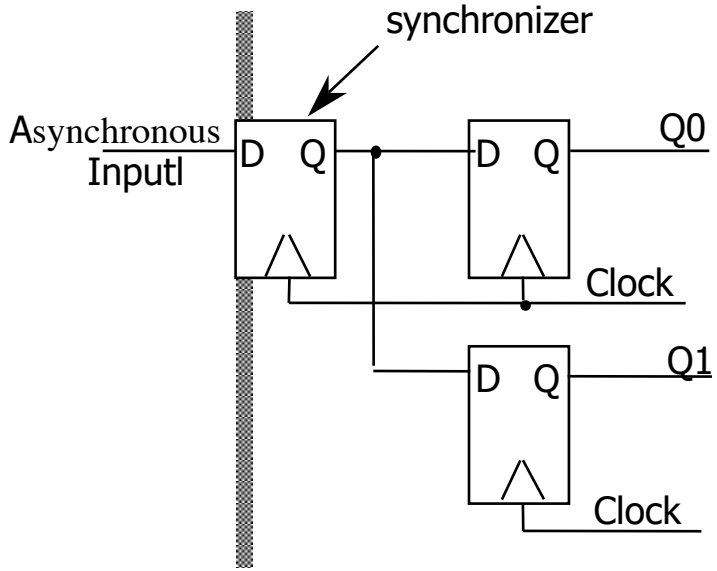

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY pin IS
PORT (
Din, En : IN STD_LOGIC;
OutSig : INOUT STD_LOGIC);
END pin;
```

```
ARCHITECTURE Bet OF pin IS
BEGIN
  PROCESS (Din,En) IS
  BEGIN
    IF En='1' THEN
      OutSig <= Din;
    ELSE
      OutSig <= 'Z';
    END IF; -- En
  END PROCESS;
END Bet;
```

Handle asynchronous inputs

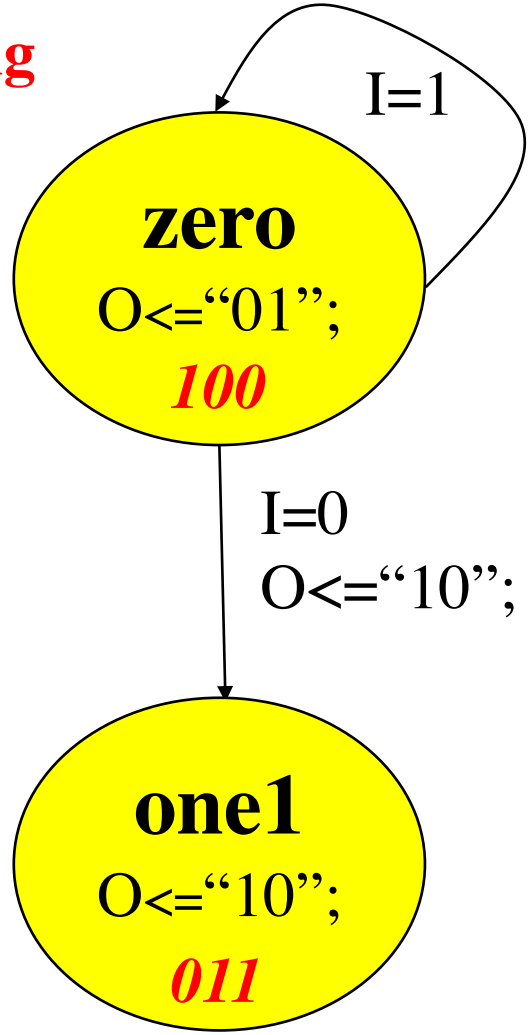
- **Never, Never, Never** allow asynchronous inputs to be connected to more than flip-flop
- synchronize as soon as possible and then process the signal as synchronous

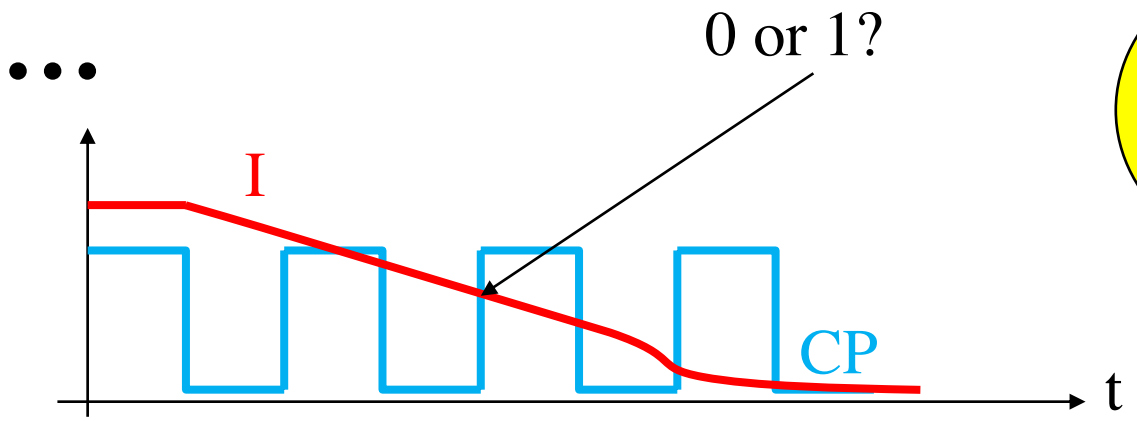




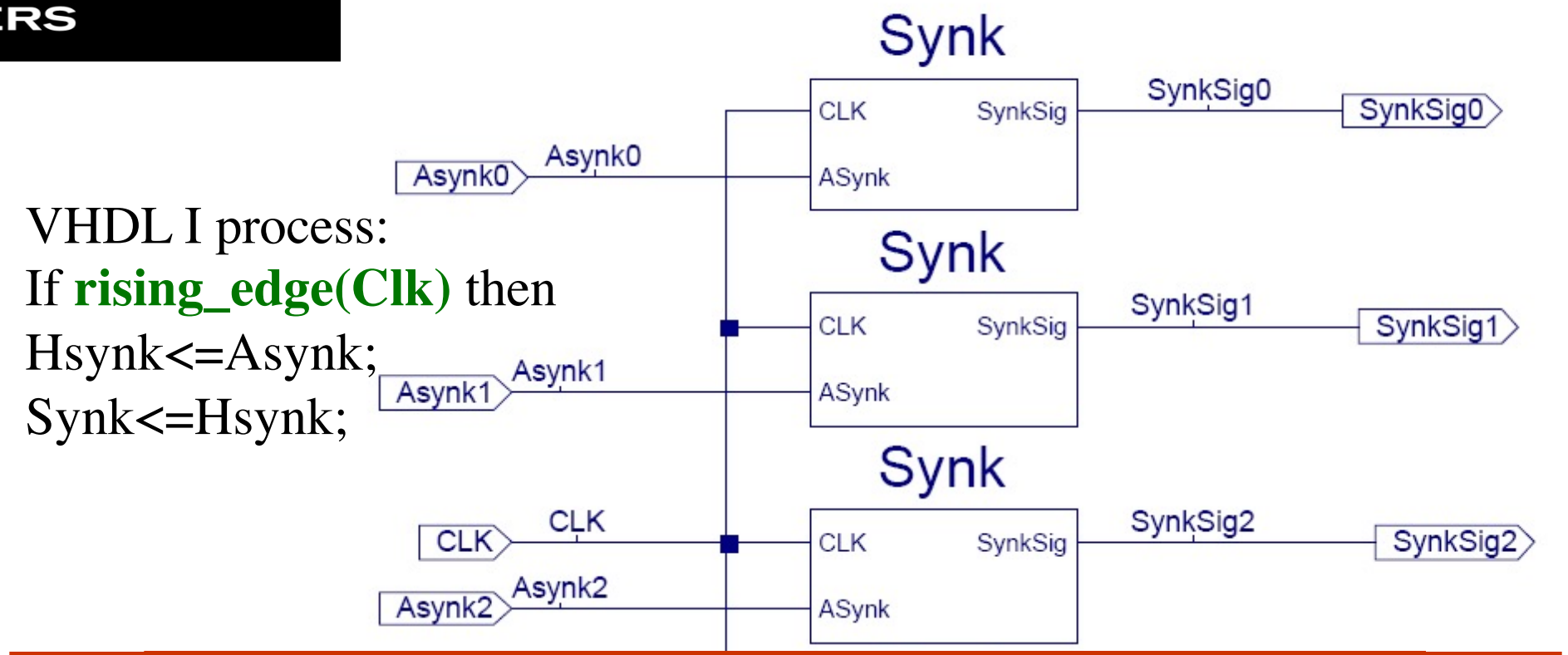
```
next_s:PROCESS (state,I)
BEGIN
 next_state <=zero; -- default
CASE state IS
    WHEN zero=>
        O<="01"; -- Moore output
        IF I='0' THEN
            next_state <=one1;
          O<="10"; -- Mealy output
        END IF;
```

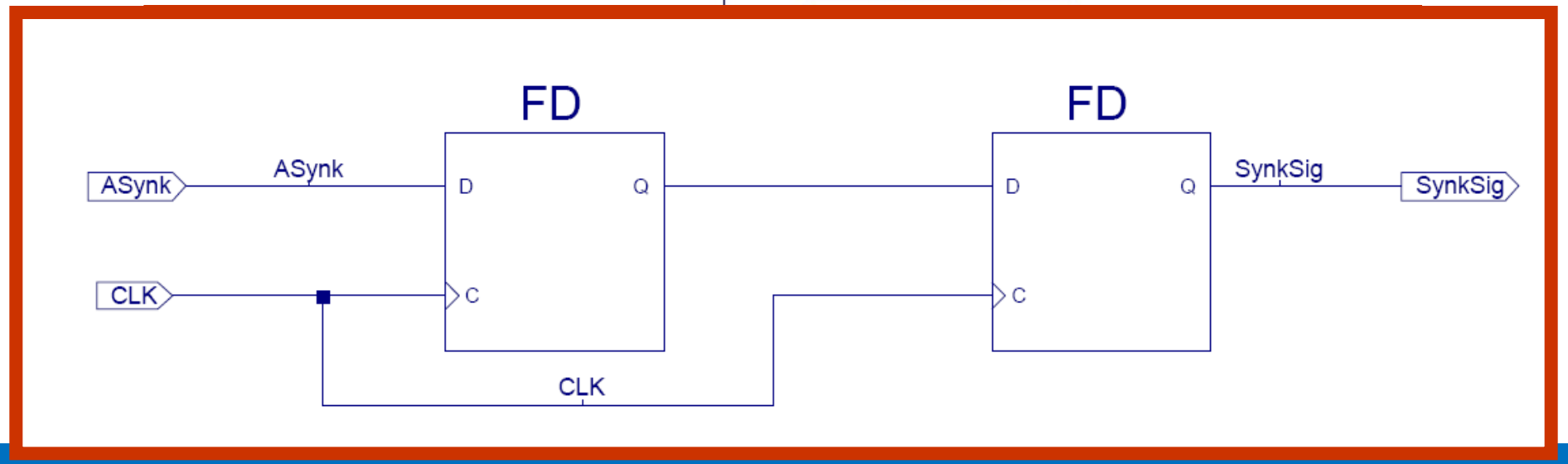
Coding





VHDL I process:
If **rising_edge(Clk)** then
Hsynk<=Asynk;
Synk<=Hsynk;





VHDL – ”for … loop” satsen.

Exempel paritetskontroll.

```
[loop_label:] FOR <identifier> IN <range> LOOP
 statements
END LOOP [loop_label:];
```

```
ARCHITECTURE arch_Par_Check OF Par_Check IS
BEGIN
  p0: PROCESS (a) IS
    VARIABLE even : std_logic;
  BEGIN
    even:='0';
    FOR i IN a'RANGE LOOP
      IF a(i)='1' THEN
        even := not even;
      END IF;
    END LOOP;
    y<= even;
END process p0;
END ARCHITECTURE arch_Par_Check;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY Par_Check IS
 PORT(a:IN STD_LOGIC_vector (7 downto 0);
      y:OUT STD_LOGIC);
END Par_Check;
```

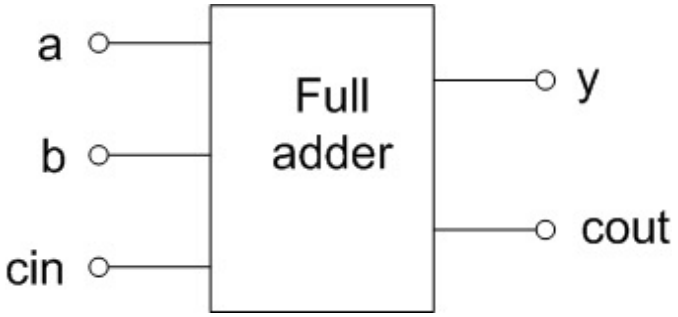
Strukturell VHDL.

Components

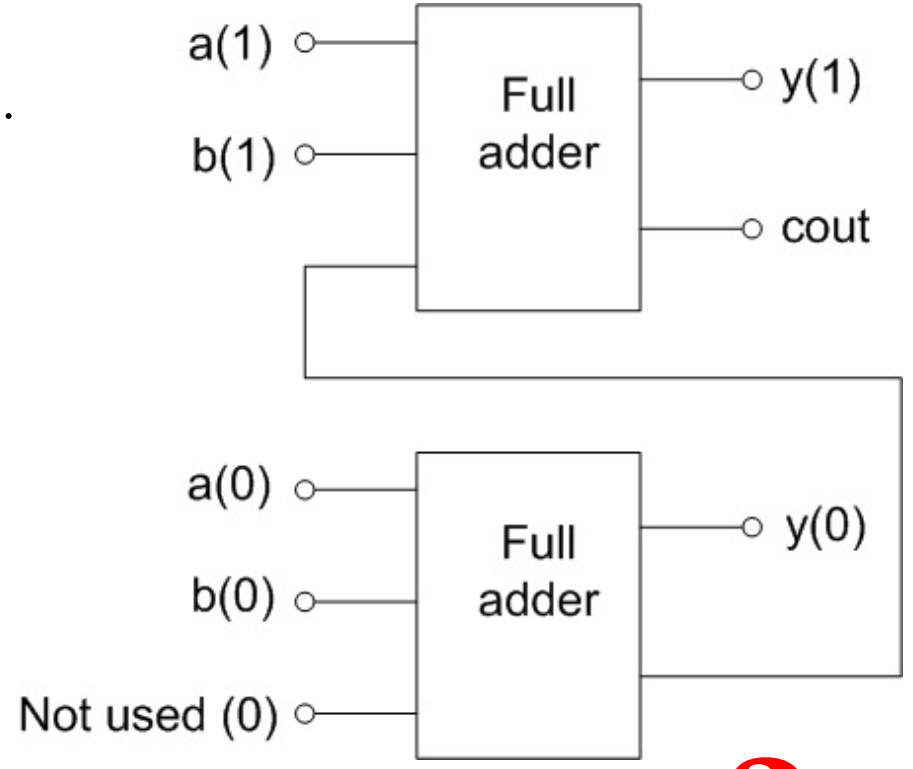
Exempel

Vi vill bygga en två-bitars adderare.

Vi börjar med att bygga en full adderare. Sedan kopplar vi samman två av dessa för att få vår två bitars adderare.



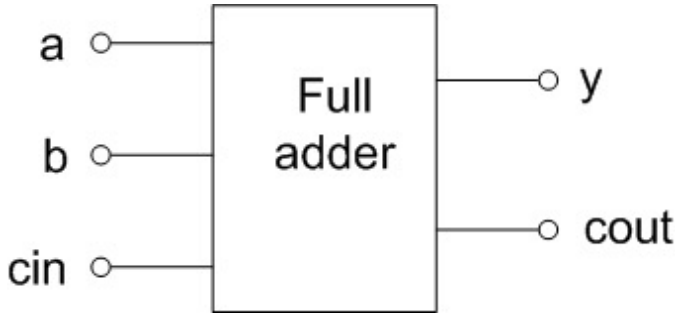
One-bit adder



Two-bit adder ?

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY full_adder IS
    PORT(    a:IN STD_LOGIC;
             b:IN STD_LOGIC;
           cin:IN STD_LOGIC;
             y:OUT STD_LOGIC;
          cout:OUT STD_LOGIC);
END full_adder;

ARCHITECTURE arch_full_adder OF full_adder IS
BEGIN
    y<=a XOR b XOR cin;
    cout<=(a AND b) OR
          (a AND cin) OR
          (b AND cin);
END arch_full_adder;
```

Våran heladderare



Separata rader ökar läsbarheten

?

Exempel forts

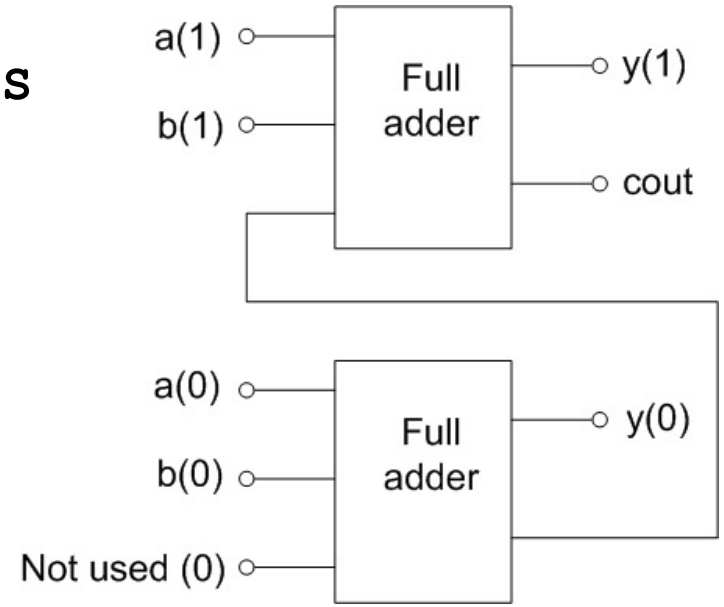
Vi går vidare till två bits adderaren.

```
-- Vi har entiteten

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY adder_2_bit IS
    PORT (    a:IN STD_LOGIC_VECTOR(1 DOWNTO 0);
              b:IN STD_LOGIC_VECTOR(1 DOWNTO 0);
              y:OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
           cout:OUT STD_LOGIC);
END adder_2_bit;
```

?

```
ARCHITECTURE arch_adder_2 OF adder_2 IS
COMPONENT full_adder IS
    PORT(    a:IN STD_LOGIC;
             b:IN STD_LOGIC;
           cin:IN STD_LOGIC;
             y:OUT STD_LOGIC;
          cout:OUT STD_LOGIC);
END COMPONENT full_adder;
    SIGNAL cint:STD_LOGIC;
BEGIN
    U0:COMPONENT full_adder
          PORT MAP(a=>a(0),b=>b(0),
                   cin=>'0',y=>y(0),cout=>cint);
    U1:COMPONENT full_adder
          PORT MAP(a=>a(1),b=>b(1),
                   cin=>cint,y=>y(1),cout=>cout);
END arch_adder_2;
```

Komponent deklaration



Komponent instanser

?

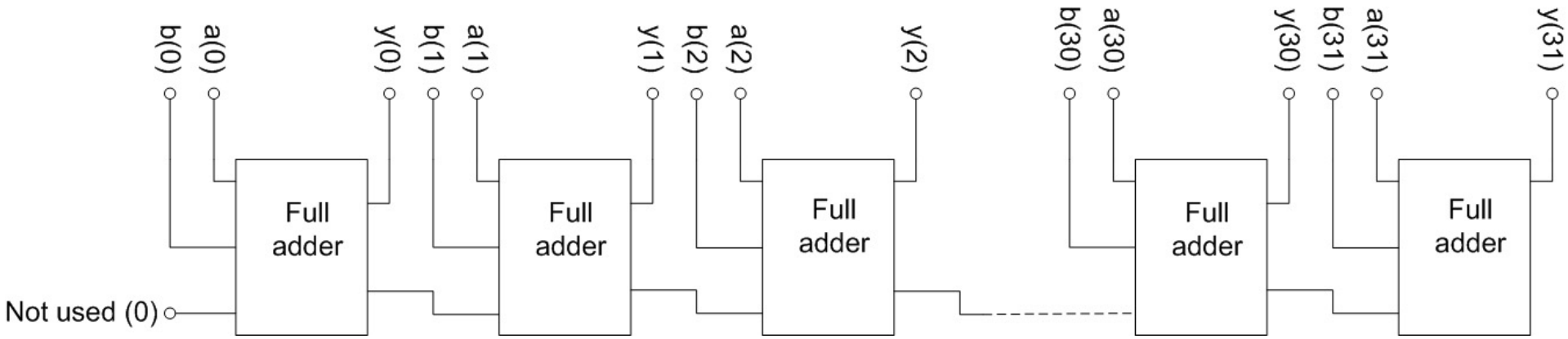
Exempel forts. 32 bitars adderare.

Komponent deklarationen är identisk med 2-bitars fallet
(Eftersom det är samma komponent)

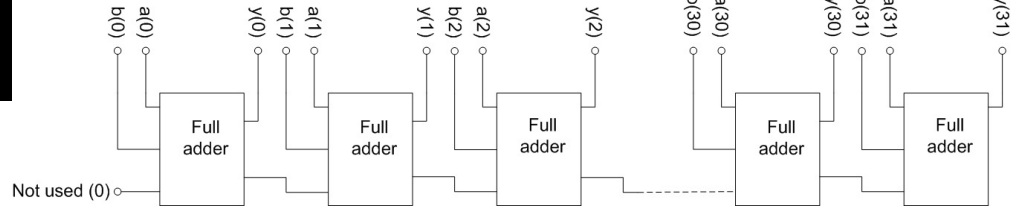
```
COMPONENT full_adder IS
    PORT(    a:IN STD_LOGIC;
             b:IN STD_LOGIC;
           cin:IN STD_LOGIC;
             y:OUT STD_LOGIC
          cout:OUT STD_LOGIC);
END COMPONENT full_adder;
```

?

Vi måste ändra i arkitekturen där komponenten instantieras.



Exempel forts




```
ARCHITECTURE arch_adder_32_bit OF adder_32_bit IS
{declaration of component full_adder}
SIGNAL cint:STD_LOGIC_VECTOR(30 DOWNTO 0);
BEGIN
   U0:COMPONENT full_adder
         PORT MAP(a=>a(0),b=>b(0),cin=>'0',
                                  y=>y(0),cout=>cint(0));
   G:FOR i IN 1 TO 30 GENERATE
      Ui:COMPONENT full_adder
            PORT MAP(a=>a(i),b=>b(i),cin=>cint(i-1),
                                     y=>y(i),cout=>cint(i));
   END GENERATE;
   U31:COMPONENT full_adder
          PORT MAP(a=>a(31),b=>b(31),cin=>cint(30),
                                     y=>y(31));
END arch_adder_32_bit;
```

Signal för överföring av ”carry”

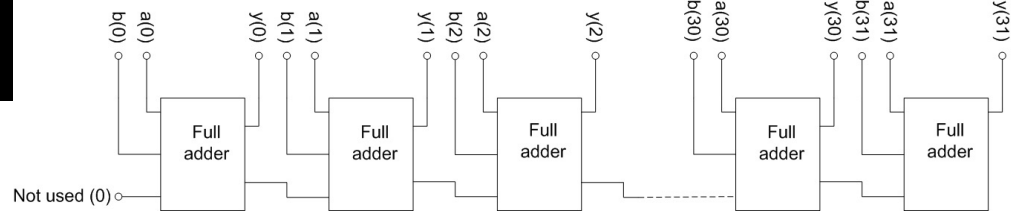
Ingen cin

Component instantiations

Ripple carry

MSB och LSB måste instantieras separat!

`cout` är inte ansluten

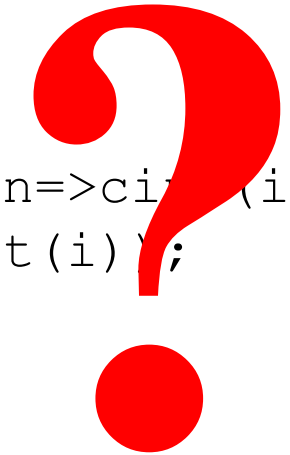


Låt oss titta närmare på genereringen av bit 1 till 30

Generate statement måste ha en label

Index variabeln behöver inte deklareras

```
G:FOR i IN 1 TO 30 GENERATE
      Ui:COMPONENT full_adder
         PORT MAP(a=>a(i),b=>b(i),cin=>cint(i-1),
                  y=>y(i),cout=>cint(i));
   END GENERATE;
```



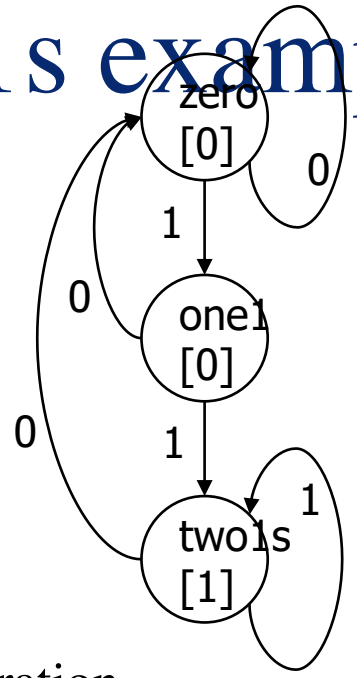
Vi behöver inte separata namn för de olika komponenterna

Vi kan se här att vi har en loop som kommer att vecklas ut till 30 en bitars adderare.

VHDL FSM - Reduce 1s example

- Moore machine

```
entity Reduce1Moor is
      port ( Clk   : in    std_logic;
                I    : in    std_logic;
            Reset : in    std_logic;
                O    : out   std_logic);
end Reduce1Moor;

architecture BEHAVIORAL of Reduce1Moor is
  type state_type is (zero,one1,two1s); --Tillståndsdeklaration
  signal state:state_type;

begin
  ----
  ----
end BEHAVIORAL;
```

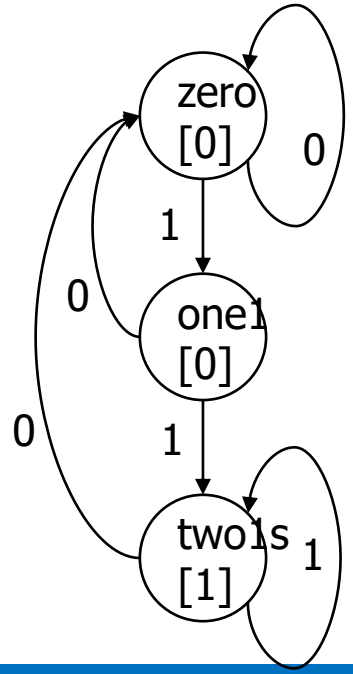


Tillståndskodning (på ett ställe, enkelt att ändra)

```
architecture BEHAVIORAL of Reduce1Moor is
  constant zero : std_logic_vector (1 downto 0) :="00";
  constant one1 : std_logic_vector (1 downto 0) :="01";
  constant two1s : std_logic_vector (1 downto 0) :="10";
  signal state : std_logic_vector (1 downto 0);
```

Moore VHDL FSM 2-process lösning (forts)

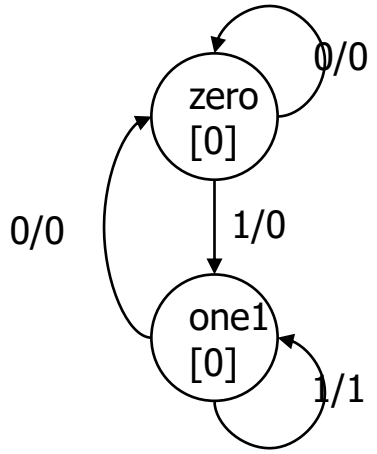

```
ASM_P: process(Clk,Reset) -- Synkron process
begin
  if Reset = '1' then -- Asynkron reset
     state<=zero; -- Reset tillstånd
   elsif Clk'event and Clk='1' then
       state<=next_state
  end if;
end process ASM_P;
```




```
-- Kombinatorisk process
next_s:process(state,I)
begin
 next_state <=zero
  case state is
       when zero=> O<='0';
            if I='1' then next_state <=one1;
            end if;
       when one1=> O<='0';
            if I='1' then next_state <=two1s;
            end if;
       when two1s=> O<='1';
            if I=‘1' then
                next_state <=two1s;
            end if;
     end case;
end process next_s;
```

Mealy VHDL FSM


```
architecture BEHAVIORAL of Reduce1Mealy is
type state_type is (zero,one1); --Tillståndsdeklaration
signal state:state_type;
signal next_state:state_type;
```




```
begin
 p1:process(Clk, Reset, next_state)
 begin
     if Clk'event and Clk='1' then
      if Reset = '1' then state<=zero;
           else state<=next_state;
     end if; end if;
 end process p1;
```

```
 p2:process(state,I)
  begin
       case state is
        when zero=>
            if I='1' then next_state<=one1; O<='0';
                 else next_state<=zero; O<='0';
            end if;
        when one1=>
            if I='1' then next_state<=one1; O<='1';
                 else next_state<=zero; O<='0';
            end if;
        when others=> next_state<=zero;
       end case;
  end process p2;
end BEHAVIORAL;
```