

# Management and analysis of physics datasets, Part. 1

## Fifth Laboratory

---

Antonio Bergnoli [bergnoli@pd.infn.it](mailto:bergnoli@pd.infn.it) - Filippo Marini [filippo.marini@pd.infn.it](mailto:filippo.marini@pd.infn.it)

02/12/2020

## Laboratory Introduction

---

- Gain confidence with State Machines
- Use the new VHDL statement **case**

**Doubts?**

---

# Synchronizer (Double FF)

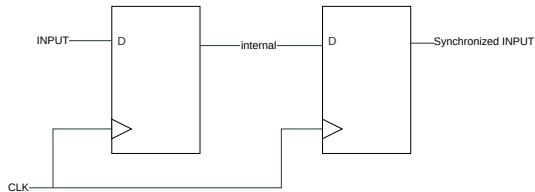


Figure 1: double flop

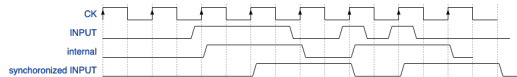


Figure 2: double flop wave

## Exercise

1. write a VHDL source code for the synchronizer
2. write a testbench to validate the input synchronizer

## State Machines

---

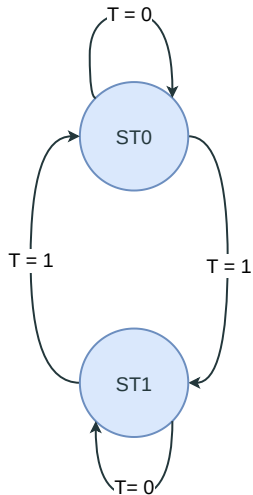


Figure 3: toggle state machine

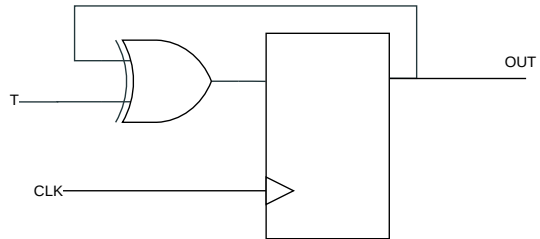


Figure 4: toggle flip flop

## Toggle flip flop as state machine

```
library IEEE;
use IEEE.std_logic_1164.all;           -- entity
entity my_fsm1 is
    port (TOG_EN, CLK, CLR : in  std_logic;
          Z1              : out std_logic);
end my_fsm1;
```

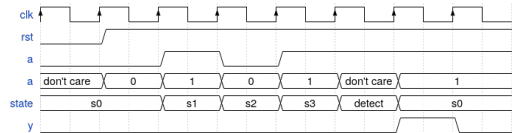
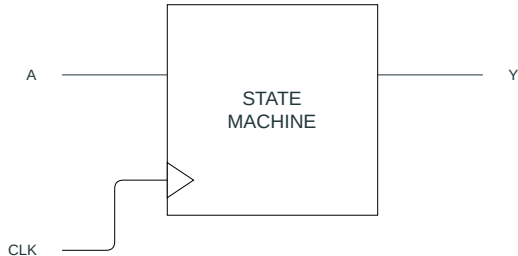
```
architecture fsm1 of my_fsm1 is
    type state_type is (ST0, ST1);
    signal state : state_type;
begin
    sync_proc : process(CLK)
    begin
        if (rising_edge(CLK)) then
            if (CLR = '1') then
                state <= ST0;
                Z1   <= '0';           -- pre-assign
            else
                case state is
                    when ST0 => -- items regarding state ST0 Z1 <= '0'; -- Moore output
                        Z1 <= '0'; -- pre-assign
                        if (TOG_EN = '1') then state <= ST1;
                        end if;
                    when ST1 => -- items regarding state ST1
                        Z1 <= '1'; -- Moore output
                        if (TOG_EN = '1') then state <= ST0;
                        end if;
                    when others => -- the catch-all condition
                        Z1 <= '0'; -- arbitrary; it should never
                        state <= ST0; -- make it to these two statements
                end case;
            end if;
        end if;
    end process sync_proc;
end fsm1;
```



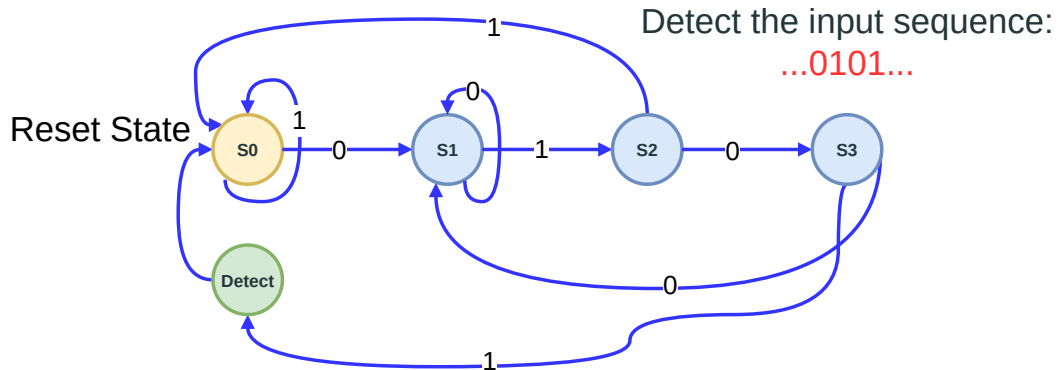
# A Simple State Machine

Pattern recognition machine

- Input is evaluated at the rising edge of the clock signal
- Output goes high only when the sequence “0101” is detected in input



## A Simple State Machine (state diagram )



# A Simple State Machine

```
library ieee;
use ieee.std_logic_1164.all;
entity patterndetect is
    port (
        a : in  std_logic;
        clk : in  std_logic;
        rst : in  std_logic;
        y : out std_logic);
end entity patterndetect;
architecture rtl of patterndetect is
    type state_t is (S0, S1, S2, S3, Detect);
    signal state : state_t := S0;
begin -- architecture rtl
    main : process (clk) is
        begin -- process main
            if rising_edge(clk) then
                if rst = '0' then
                    state <= S0;
                    y <= '0';
                else
                    case state is
                        when S0 =>
                            y <= '0';
                            if a = '0' then
                                state <= S1;
                            end if;
                        when S1 =>
                            y <= '0';
                            if a = '0' then
                                state <= S1;
                            elsif a = '1' then
                                state <= S2;
                            end if;
                        when S2 =>
                            y <= '0';
                            if a = '0' then
                                state <= S3;
                            elsif a = '1' then
                                state <= S0;
                            else
                                null;
                            end if;
                        when S3 =>
                            y <= '0';
                            if a = '0' then
                                state <= S1;
                            elsif a = '1' then
                                state <= Detect;
                            else
                                null;
                            end if;
                        when Detect =>
                            y <= '1';
                            state <= S0;
                    end case;
                end if;
            end if;
        end process main;
    end architecture rtl;
```

```
when S2 =>
    y <= '0';
    if a = '0' then
        state <= S3;
    elsif a = '1' then
        state <= S0;
    else
        null;
    end if;
when S3 =>
    y <= '0';
    if a = '0' then
        state <= S1;
    elsif a = '1' then
        state <= Detect;
    else
        null;
    end if;
when Detect =>
    y <= '1';
    state <= S0;
when others => null;
end case;
end if;
end if;
end process main;
end architecture rtl;
```

# A Simple State Machine ( testbench )

```
library ieee;
use ieee.std_logic_1164.all;
entity patterndetect_tb is
end entity patterndetect_tb;
```

```
architecture test of patterndetect_tb is
    signal a      : std_logic;
    signal clk     : std_logic := '0';
    signal rst     : std_logic;
    signal y      : std_logic;
begin -- architecture test
    DUT : entity work.patterndetect
        port map (
            a  => a,
            clk => clk,
            rst => rst,
            y  => y);
    clk <= not clk after 2 ns;
    WaveGen_Proc : process
    begin
        a  <= '0';
        rst <= '1'; wait for 10 ns; wait until rising_edge(clk);
        rst <= '0'; wait for 10 ns; wait until rising_edge(clk);
        rst <= '1';
        wait until rising_edge(clk);
        a  <= '0';
        wait until rising_edge(clk);
        a  <= '1';
        wait until rising_edge(clk);
        a  <= '0';
        wait until rising_edge(clk);
        a  <= '1';
        wait for 100 ns;
        wait;
    end process WaveGen_Proc;
end architecture test;
```

## Homework

---

- Try to “mess” with the pattern recognition SM, change the detection pattern, add more bits, ...
- Invent and design a new SM. Start from the block diagram and then translate in VHDL
- Build a testbench to check the behaviour of your SM