Hardware Programming
HWP1

capturing a
FPGA
design with
VHDL

Planning: theory

- First week
  - Introduction digital systems
  - Introduction to FPGAs
  - Structured digital Design
  - Modelling concepts in VHDL
- Second week
  - Introduction VHDL
  - Design verification
  - Code structure and data types
- Third week
  - Combinational versus sequential design
  - Concurrent and sequential code
  - Signals and variables
- Fourth week
  - Components
  - Generics
- Fifth week
  - Designing state machines

Fifth Week: Theory

- Theory:
  - Ch15+16: State Machines

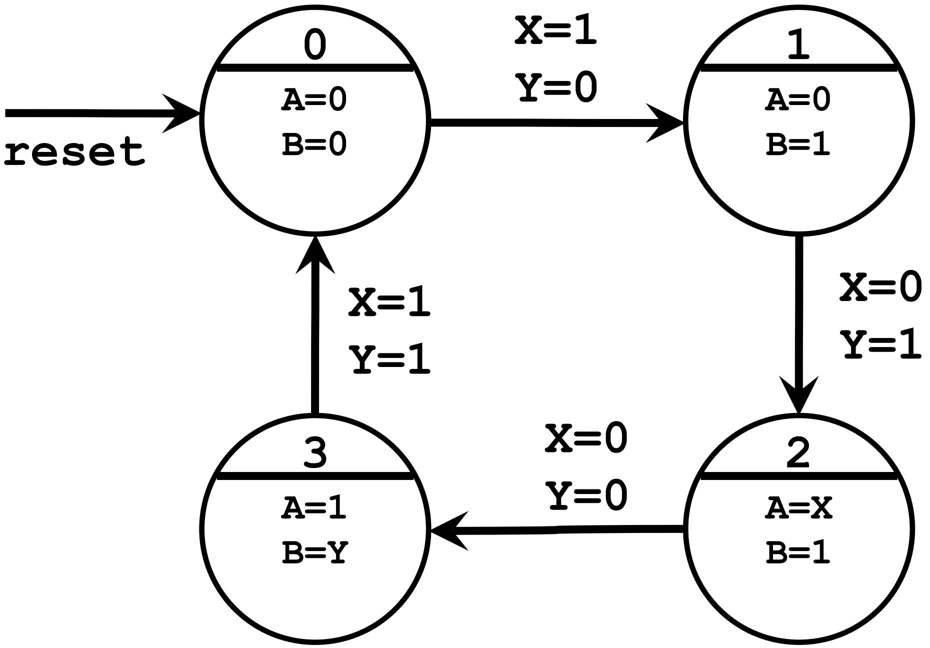
- Goals:
  - Learn how to design and implement a Finite State Machine in a digital circuit using VHDL

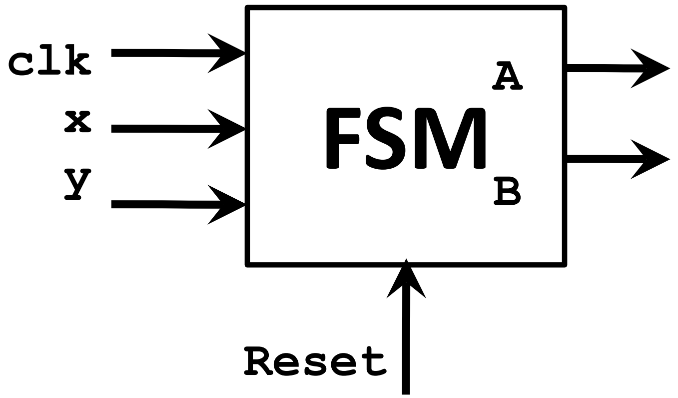
Agenda

- **Finite State Machines**
- Example
- Example in VHDL
- Template in VHDL

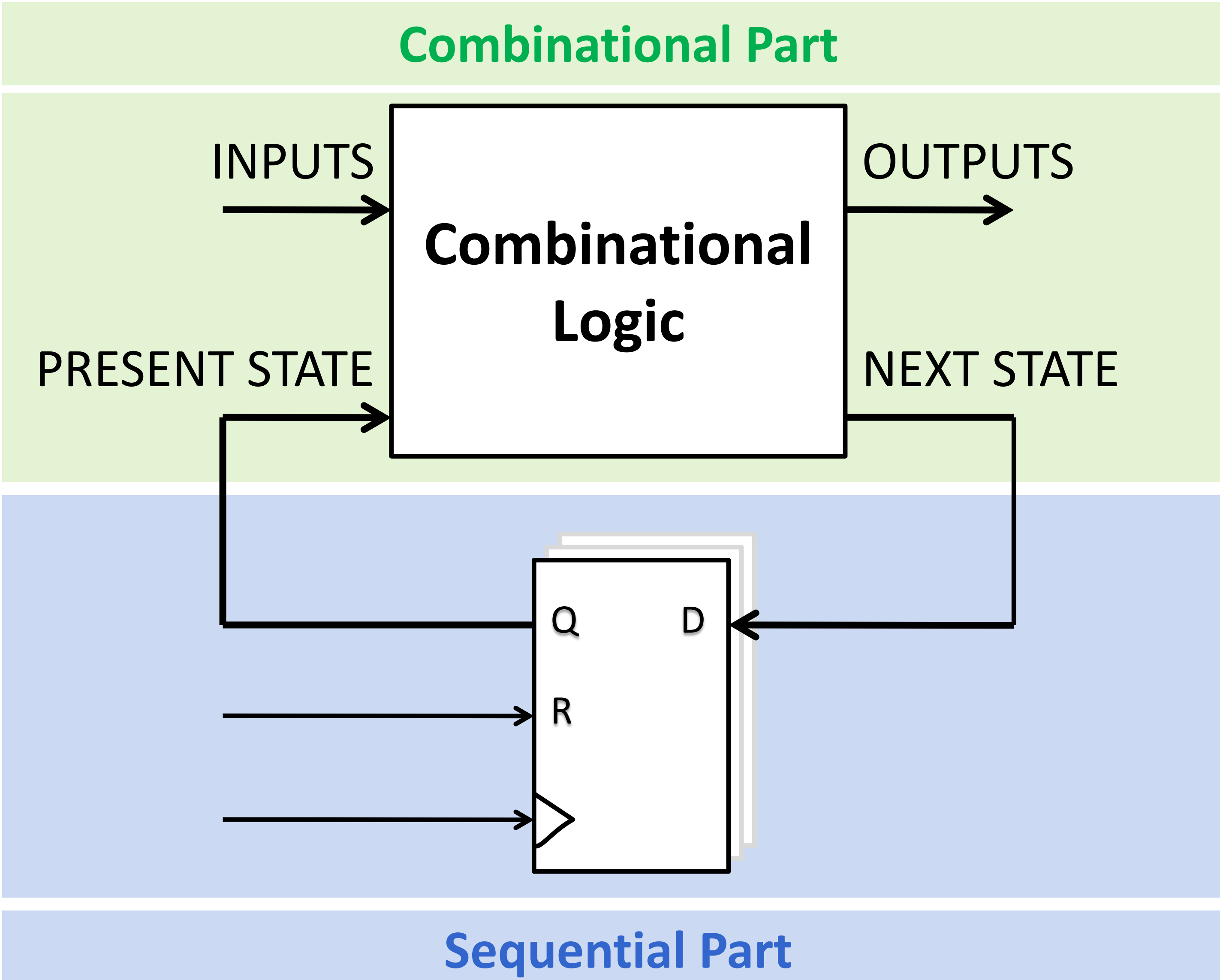
Finite State Machines Design

- FSM diagrams consist of:
  - States
  - Transitions
    - Conditions
  - Inputs
  - Outputs
  - Reset

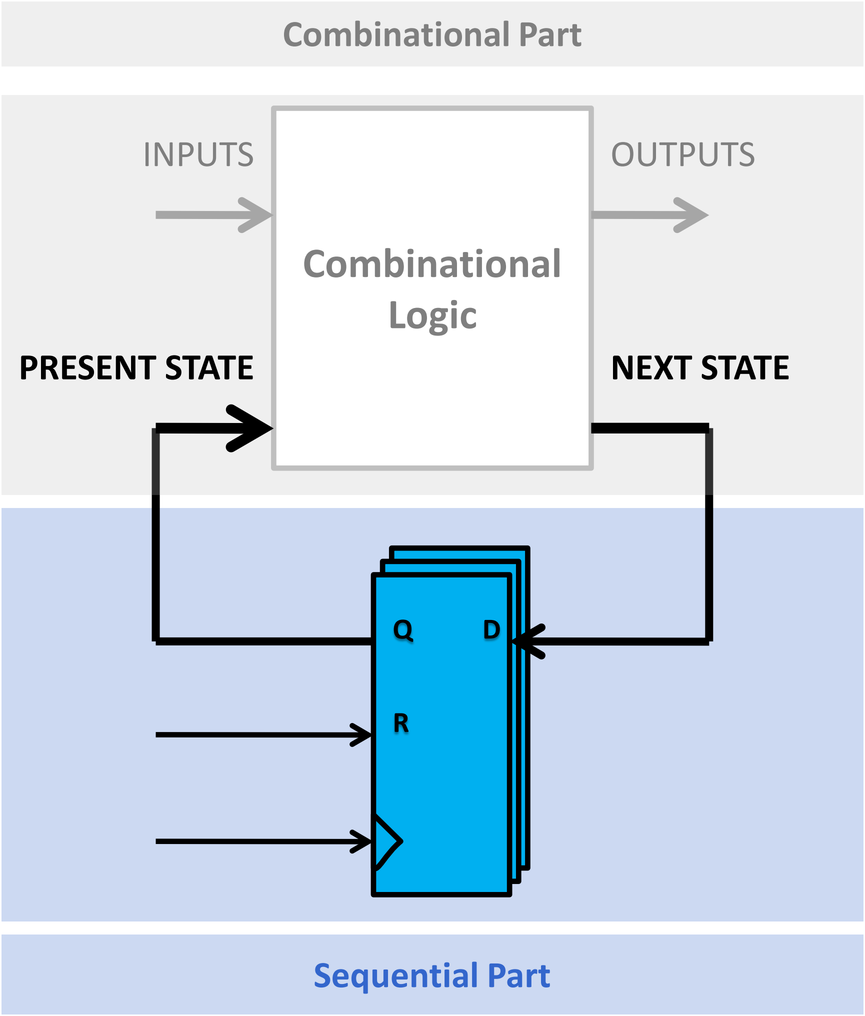




Finite State Machines in Digital Circuits

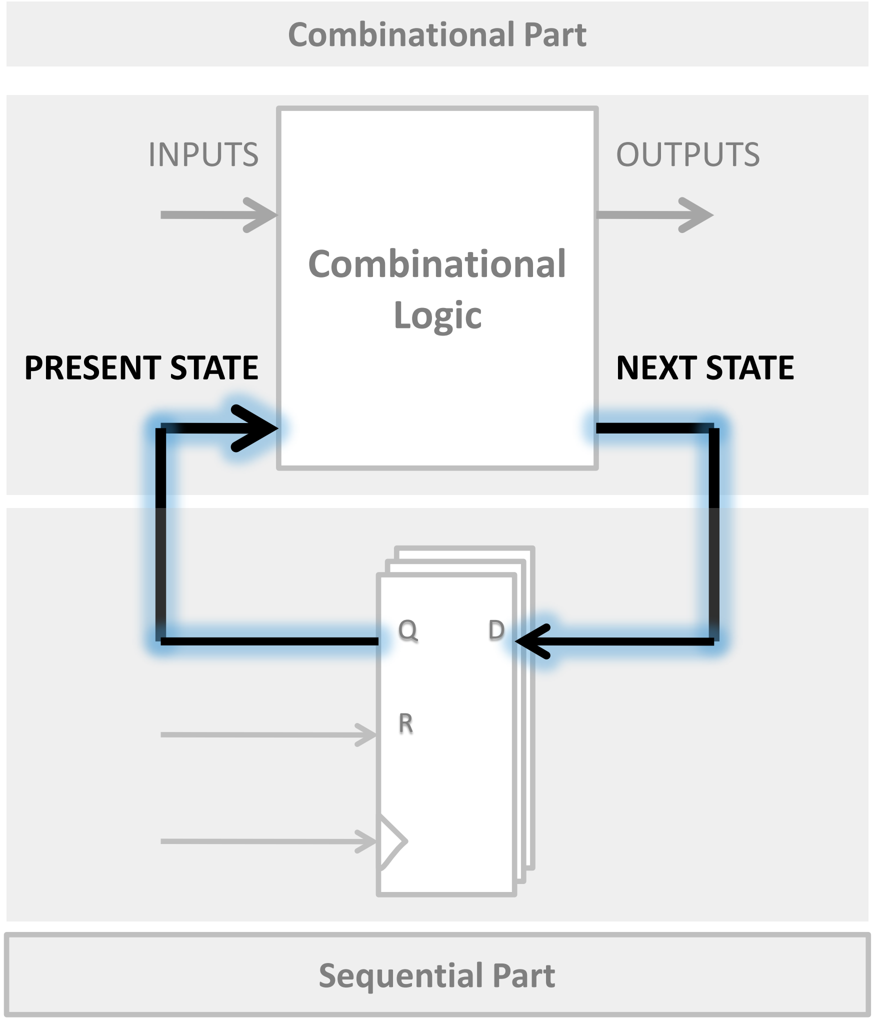


The Sequential Part



- The sequential part consists only of dffs, a clock and a reset.
- The flip-flops hold the present state
- They switch to the next state on the clock-edge
- The reset makes the present state 0000 (initial state).
- The combinational part determines the next state by the inputs and present state

Encoding the State Bus

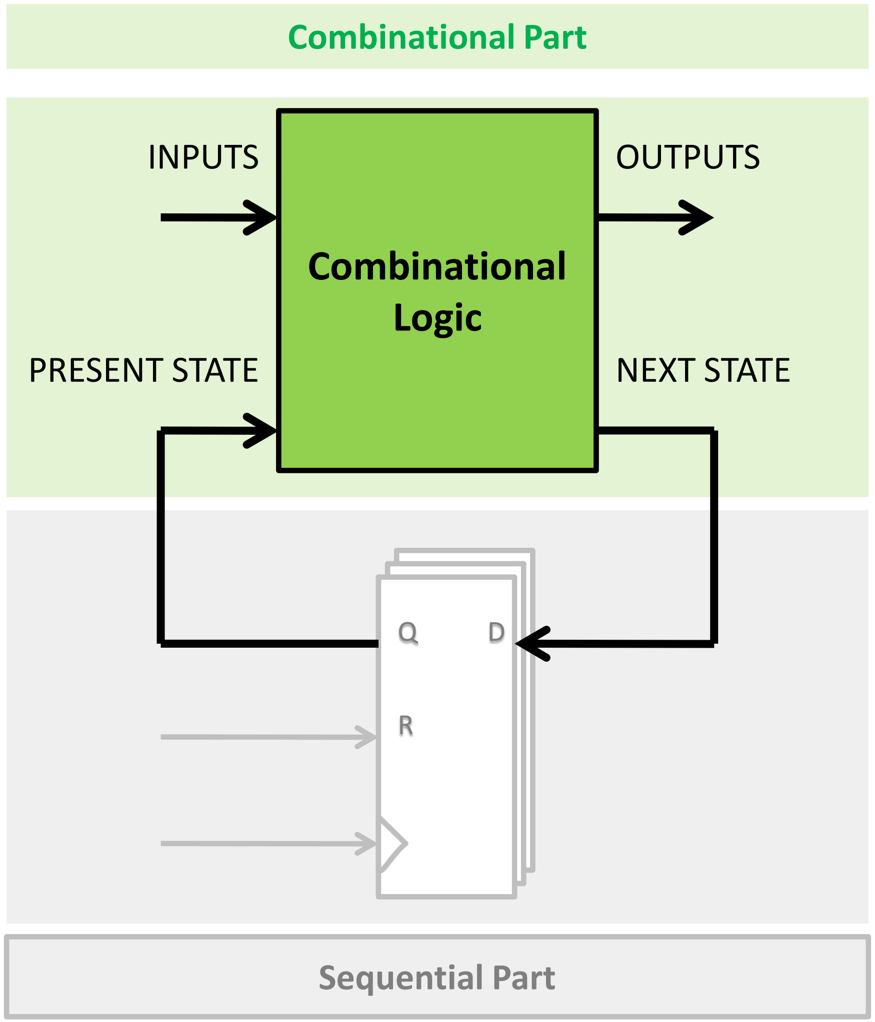


- The states are represented by bits (of course)
- Encoding states can be done in a few ways:

State	BINARY	TWO-HOT	ONE-HOT
0	00	00	0001
1	01	01	0010
2	10	11	0100
3	11	10	1000

- Look in book for “Encoding styles”
- Keep it easy, let Quartus decide the encoding.

The Combinational Part



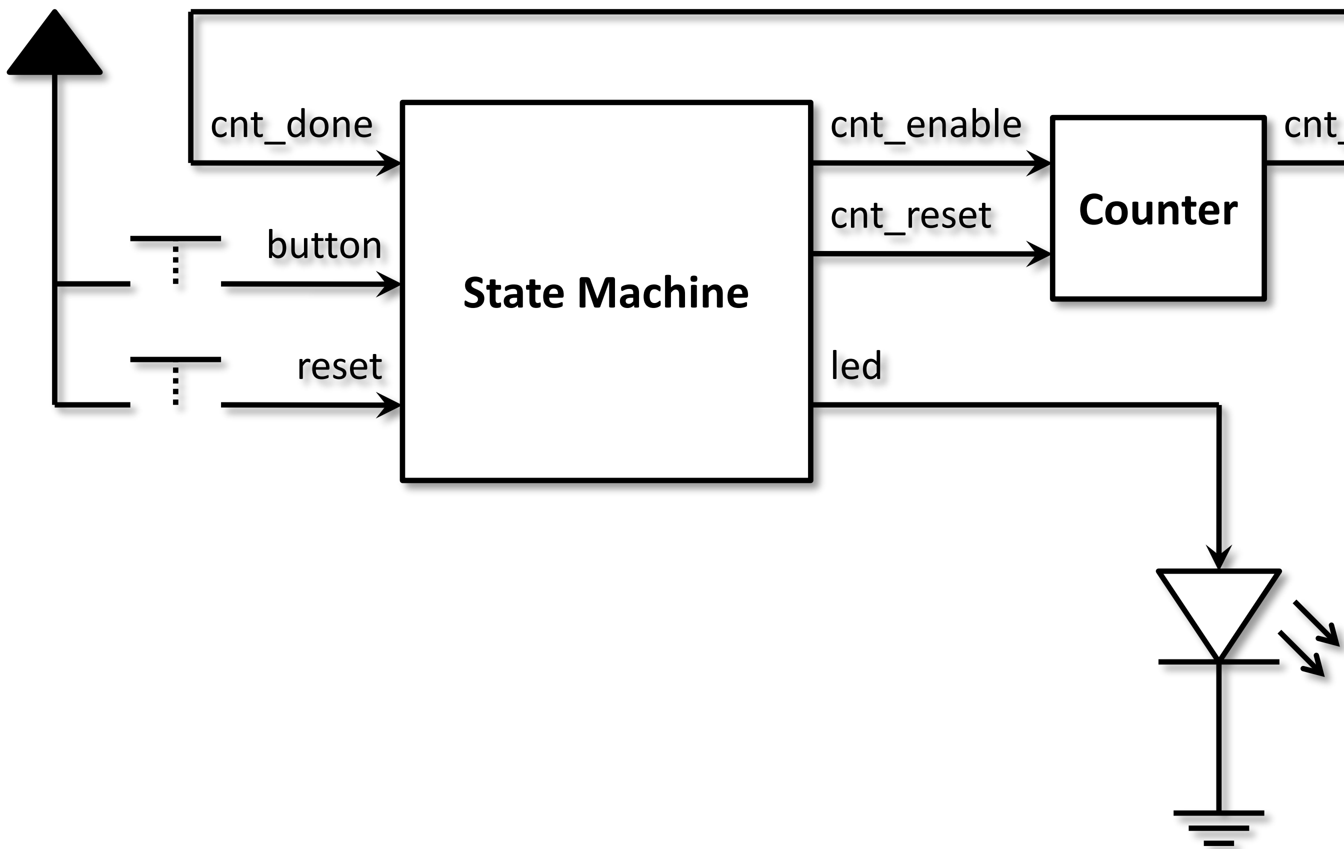
- the combinational part determines the next state
  - it is a function of the inputs and present state
- the combinational part determines the outputs
  - Moore: it is a function of the present state
  - Mealy: it is a function of the present state combined with the inputs
- Always design as Moore and change to Mealy if output needs to react instantly

- Template M1 on page 382
- [Listing]

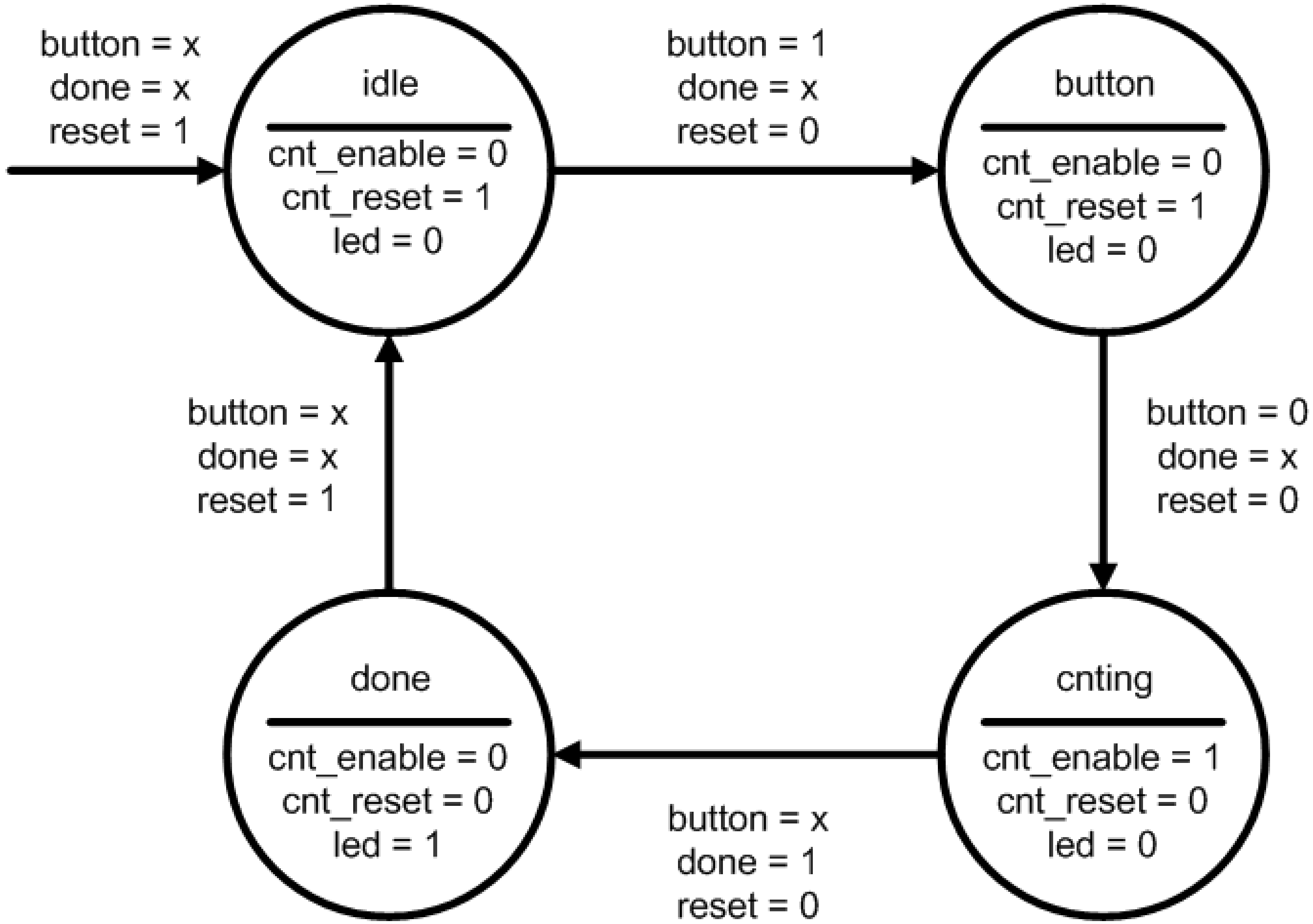
Agenda

- Finite State Machines
- **Example**
- Example in VHDL
- Template in VHDL

Egg Timer: FSM Context



Egg Timer: State Diagram



Bad example in VHDL

- Same Egg Timer: the entity

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

entity fsm_hard is
    port(
        clk          : in    std_logic;
        reset        : in    std_logic;
        button       : in    std_logic;
        cnt_done     : in    std_logic;
        cnt_enable   : out   std_logic;
        cnt_reset    : out   std_logic;
        led          : out   std_logic
    );
 end entity;
```

Bad example in VHDL

- We can do it the hard way (previous example) and then write the code:

```
architecture rtl of fsm_hard is
    signal s: std_logic_vector(1 downto 0) := "00";
    signal n: std_logic_vector(1 downto 0) := "00";
Begin
    -- The flip-flops to hold the current state and switch to next state
    process(clk)
    begin
        if reset = '1' then
            s <= "00";
        elsif rising_edge(clk) then
            s <= n;
        end if;
    end process;
    -- The combinational part
    cnt_enable  <= s(1) and not(s(0));
    cnt_reset   <= not( s(1) and not(s(0)))
    led         <= s(1) and s(0);
    n(1) <= s(1) or (not(button) and s(0));
    n(0) <= (s(0) and s(1)) or (not(s(1)) and button) or (s(1) and cnt_done);
end rtl;
```

Disadvantages?

Time efficiency

Readability

Reusability

Example in VHDL

- To write the circuit in such a low-level way is not recommended.
- **Use the power of abstraction!**
- A state machine template in VHDL can be found in the course folder.

Example in VHDL

- Same Egg Timer: the entity

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

entity fsm_egg_timer is
    port(
        clk, reset, btn, cnt_done : in std_ulogic;
        cnt_enable, cnt_reset, led : out std_ulogic
    );
 end entity;
```

State Bus Encoding in VHDL

- We need to define the states inside our architecture.
- We can use the “TYPE” keyword, it’s like “ENUM” in C. The synthesizer will define what idle, button, cnting, etc… is.
- We need to define signals for the present and next state.

```
architecture rtl of fsm_egg_timer is

    -- Define an enumerated type for the state machine
    type state_type is (idle, button, cnting, done);

    -- Register to hold the current state
    signal present_state, next_state : state_type;

begin
```

Define the flip-flops

```
-- state register
   pr_flipflops : process (clk, reset)
   begin
       if reset then
           present_state <= idle;
       elsif rising_edge(clk) then
           present_state <= next_state;
       end if;
   end process;
```

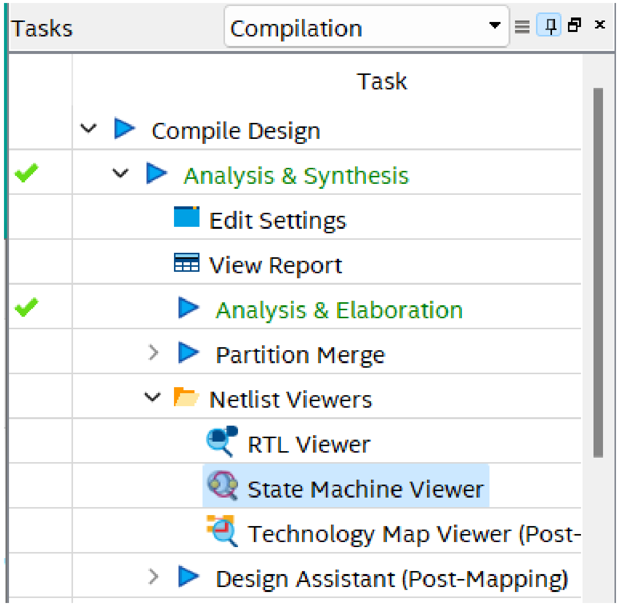
Write the transitions

```
-- logic to determine the next state
   pr_next_state : process (present_state, btn, cnt_done)
   begin
       case present_state is
           when idle =>
               if btn then
                   next_state <= btn;
               else
                   next_state <= idle;
               end if;
           when btn =>
               if not btn then
                   next_state <= cnting;
               else
                   next_state <= btn;
               end if;
           when cnting =>
               if cnt_done then
                   next_state <= done;
               else
                   next_state <= cnting;
               end if;
           when done =>
               next_state <= done;
       end case;
   end process;
```

Write the outputs (Moore)

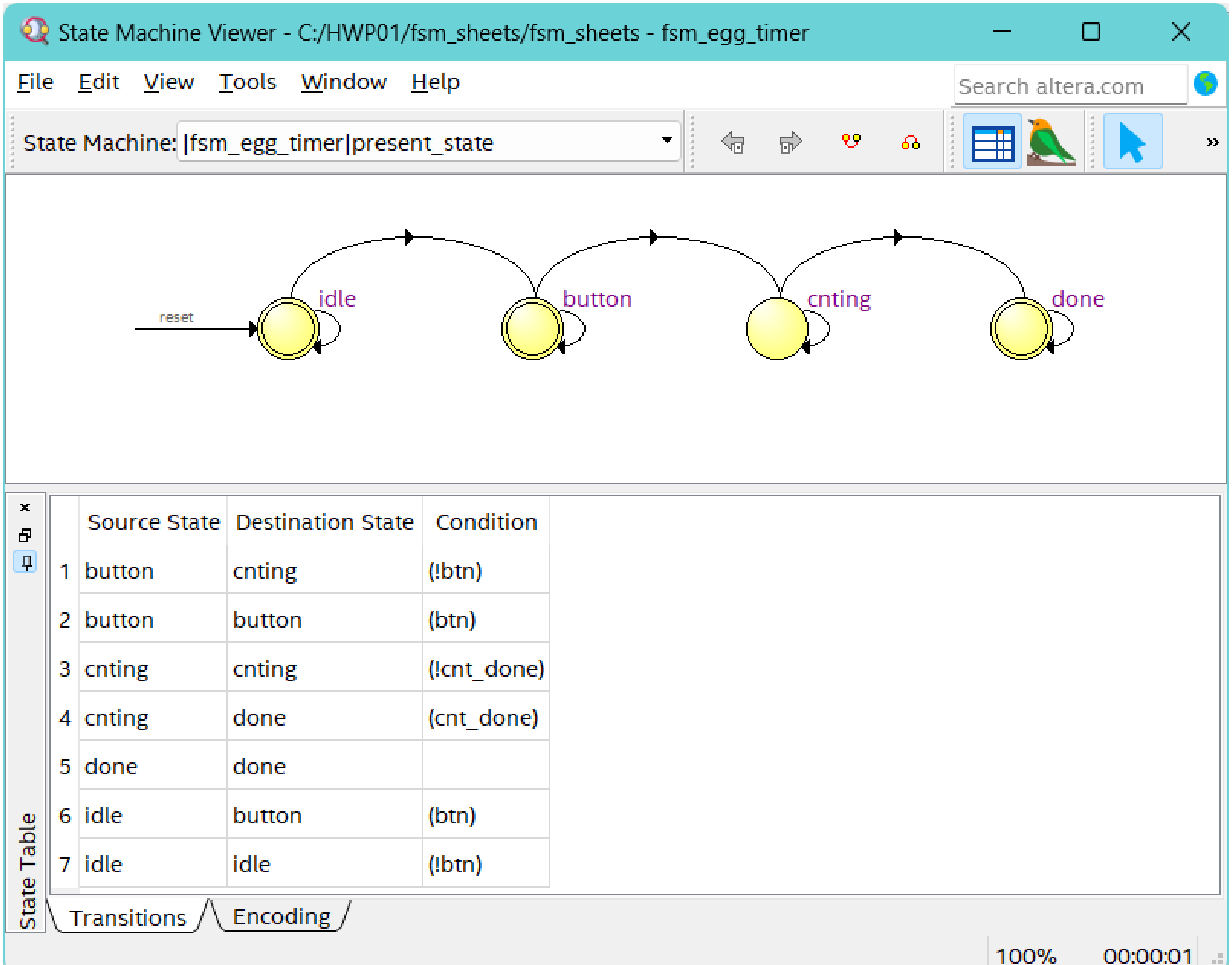
```
-- Logic to determine the outputs
   pr_outputs: process (present_state)
   begin
       case present_state is
           when idle =>
               cnt_enable  <= '0';
               cnt_reset   <= '1';
               led         <= '0';
           when btn =>
               cnt_enable  <= '0';
               cnt_reset   <= '1';
               led         <= '0';
           when cnting =>
               cnt_enable  <= '0';
               cnt_reset   <= '0';
               led         <= '0';
           when done =>
               cnt_enable  <= '1';
               cnt_reset   <= '0';
               led         <= '1';
       end case;
   end process;
end architecture;
```

State Machine Viewer



- Quartus can detect a state machine in your code!

State Machine Viewer



	Source State	Destination State	Condition
1	button	cnting	(!btn)
2	button	button	(btn)
3	cnting	cnting	(!cnt_done)
4	cnting	done	(cnt_done)
5	done	done	
6	idle	button	(btn)
7	idle	idle	(!btn)

Agenda

- Finite State Machines
- Example
- Example in VHDL
- **Template in VHDL**

Write the outputs (Moore)

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

entity fsm is
            port (
        clk, rst: in std_ulogic;
        input1, input2, ...: in std_ulogic;
        output1, output2, ...: out std_ulogic
    );
end fsm;

architecture behavior of fsm is
            type state_type is (idle, state1, state2,
...);
            signal pr_state, nx_state: state_type;
begin

    process(clk, rst)
    begin
        if rst then
            pr_state <= idle;
        elsif rising_edge(clk) then
            pr_state <= nx_state;
        end if;
    end process;
```

```vhdl
    process(pr_state, input1, input2, ...)
    begin
        case pr_state is
          when idle =>
              if input1 then
                nx_state <= state1;
              else
                nx_state <= state2;
              end if;
            when state1 =>
                nx_state <= state2;
            when state2 =>
                ...;
            when ... =>
                ...;
        end case;
    end process;

    process(pr_state)
    begin
        case pr_state is
            when idle =>
                output1 <= '0';
                output2 <= '0';
                ...
            when state1 =>
                output1 <= '0';
                output2 <= '1';
                ...
            when state2 =>
                ...;
            when ... =>
                ...;
        end case;
    end process;

end architecture;
```

- Template M1 in the book
- [Listing]()

HOGESCHOOL ROTTERDAM

Advantages of using the template

- Readability
- Reusability
- Adaptability
- Quicker (especially for bigger designs)

- Quartus also has built-in templates (right-click in the editor)