HOGESCHOOL ROTTERDAM

Hardware Programming HWP01

capturing a

FPGA

design with

VHDL

Fifth Week: Theory

- Theory:
  - H11: State Machines

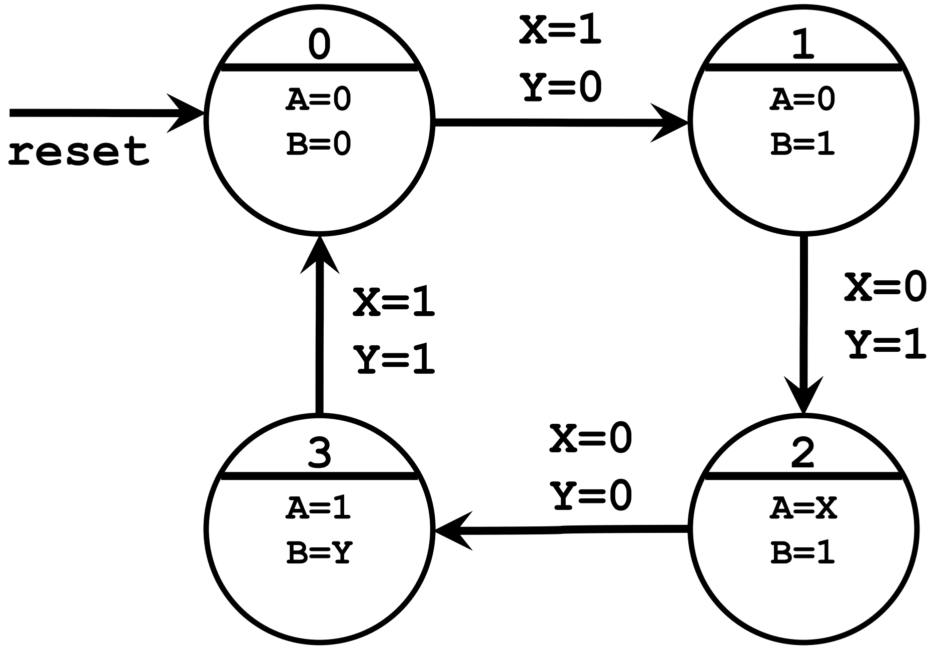
- Goals:
  - Learn how to design and implement a Finite State Machine in a digital circuit using VHDL

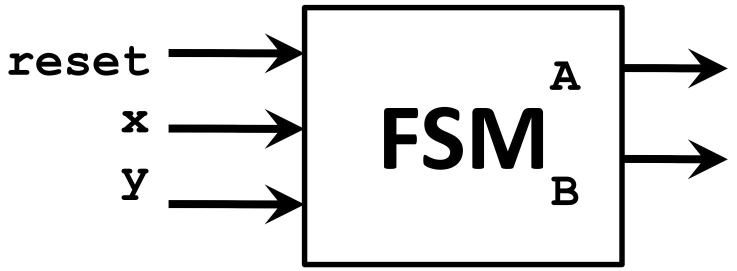
Agenda

- Finite State Machines (for circuits)
- Example in circuits
- Example in VHDL
- Template in VHDL

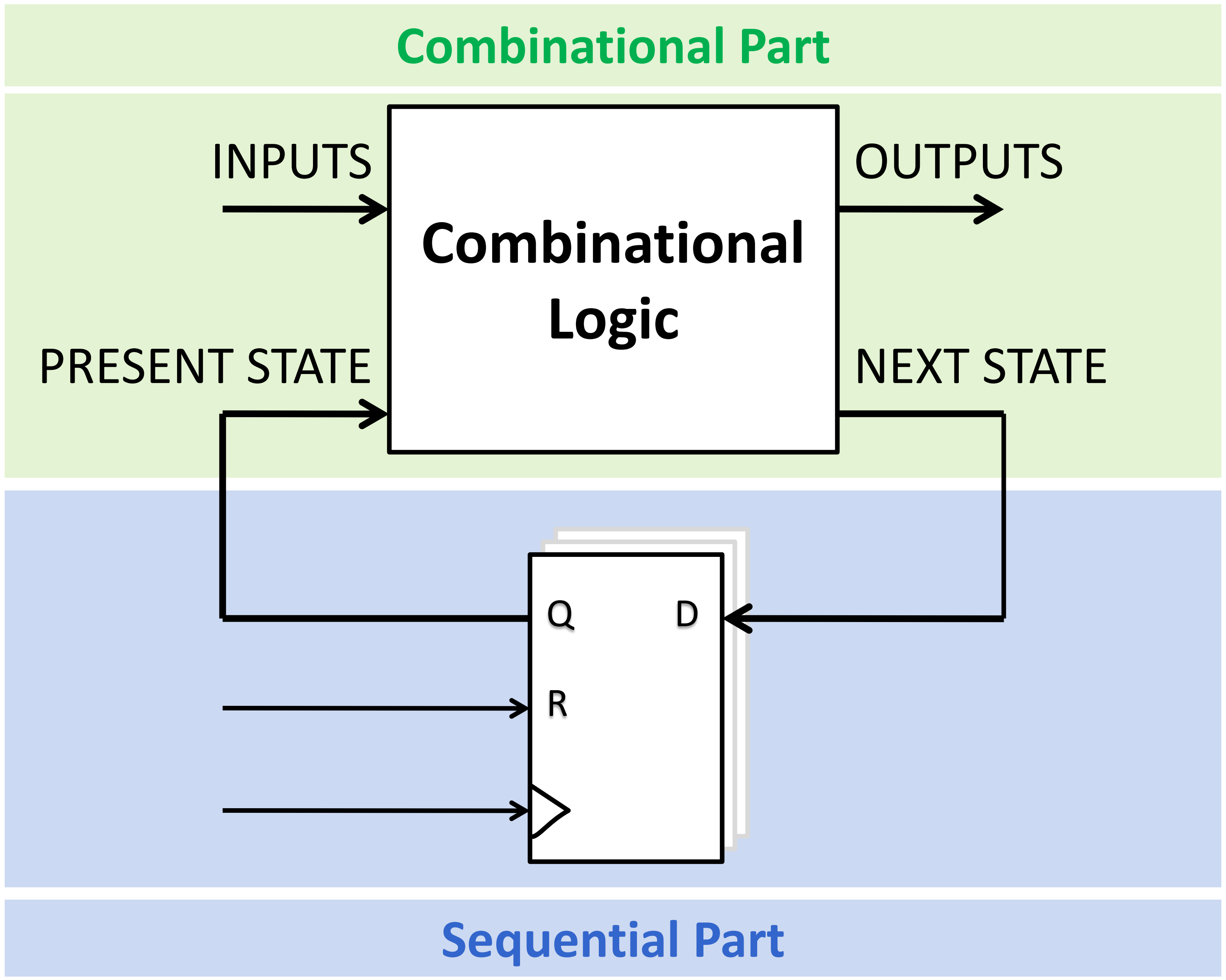
Finite State Machines Design

- In digital circuits FSM diagrams consist of:
  - States
  - Transitions
    - Conditions
  - Outputs
- **There are no actions during events in contrary to software FSMs**
  - **Circuits** provide "functions" and cannot be "executed" sequentially like software functions.
  - We need Enable, Reset and Done signals (handshaking).

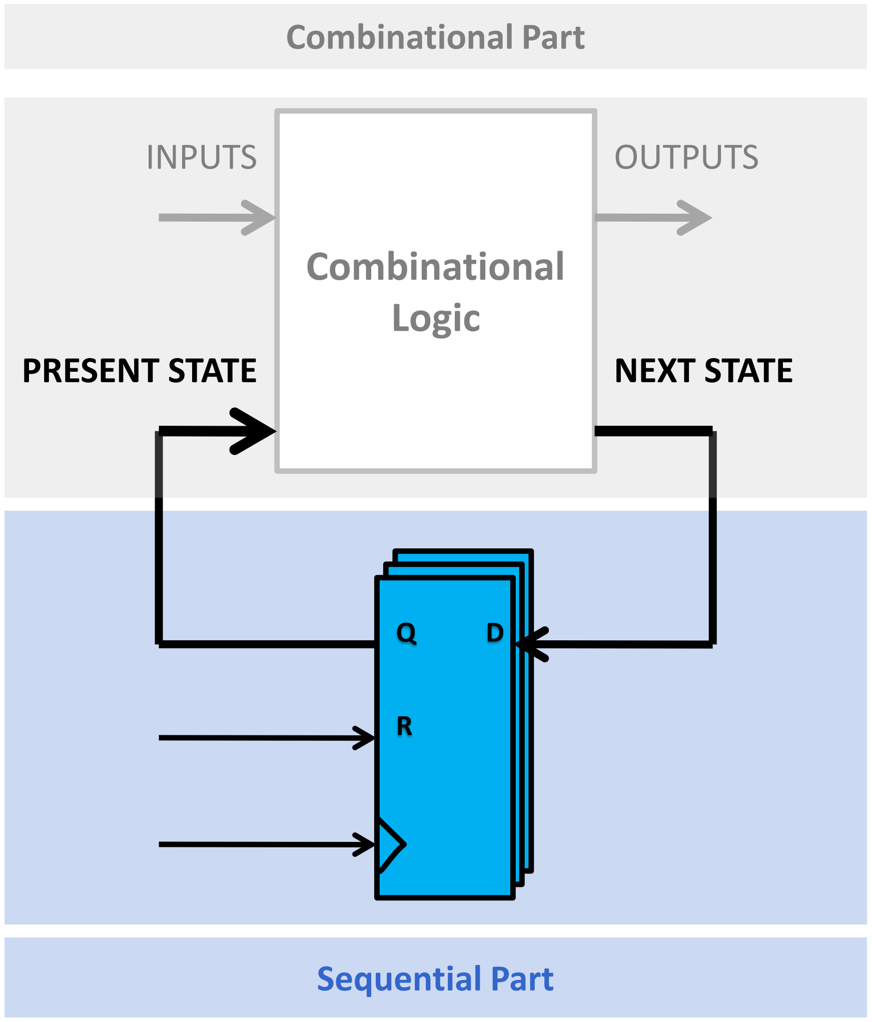




Finite State Machines in Digital Circuits

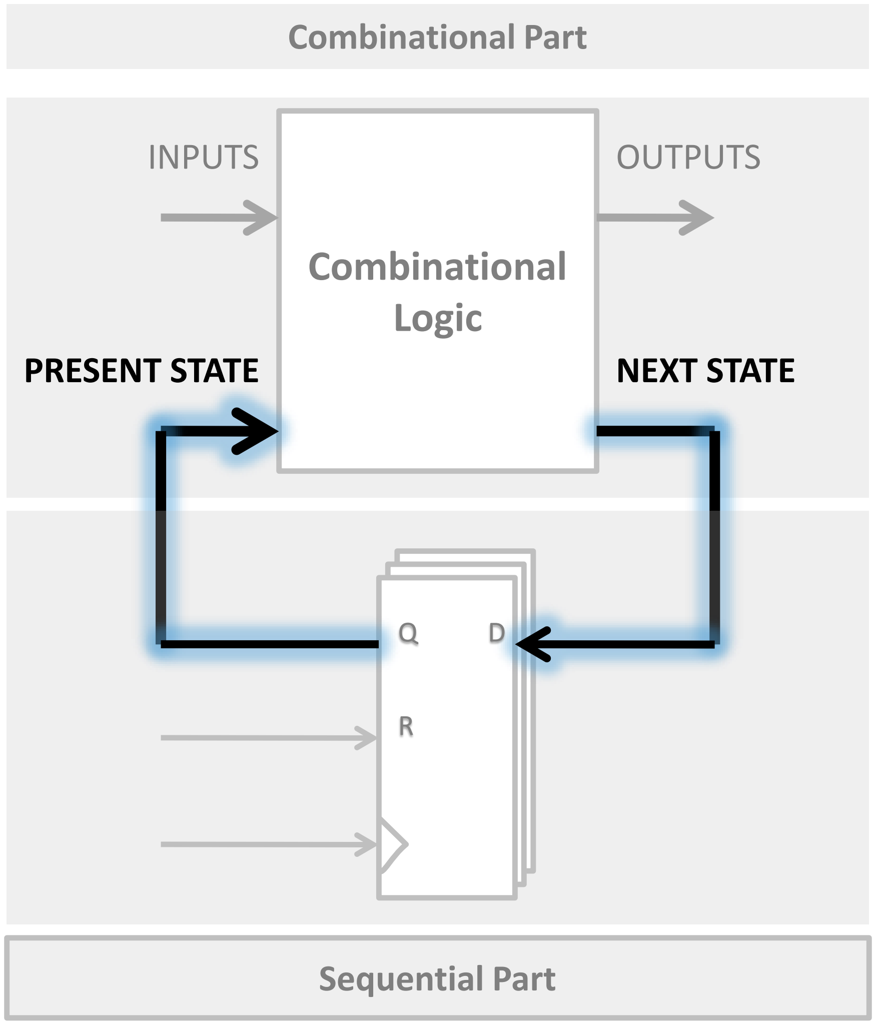


The Sequential Part



- 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
- 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

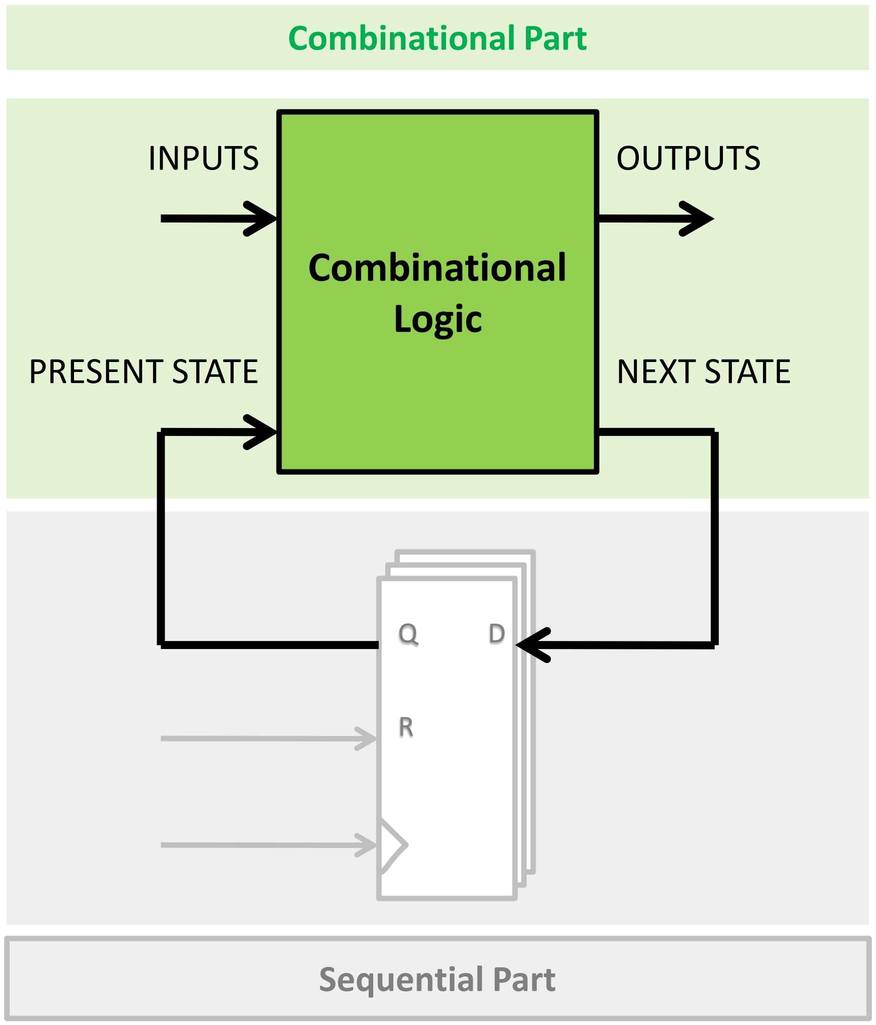


- The states are represented by bits (of course)
- Encoding states can be done in a few ways:

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

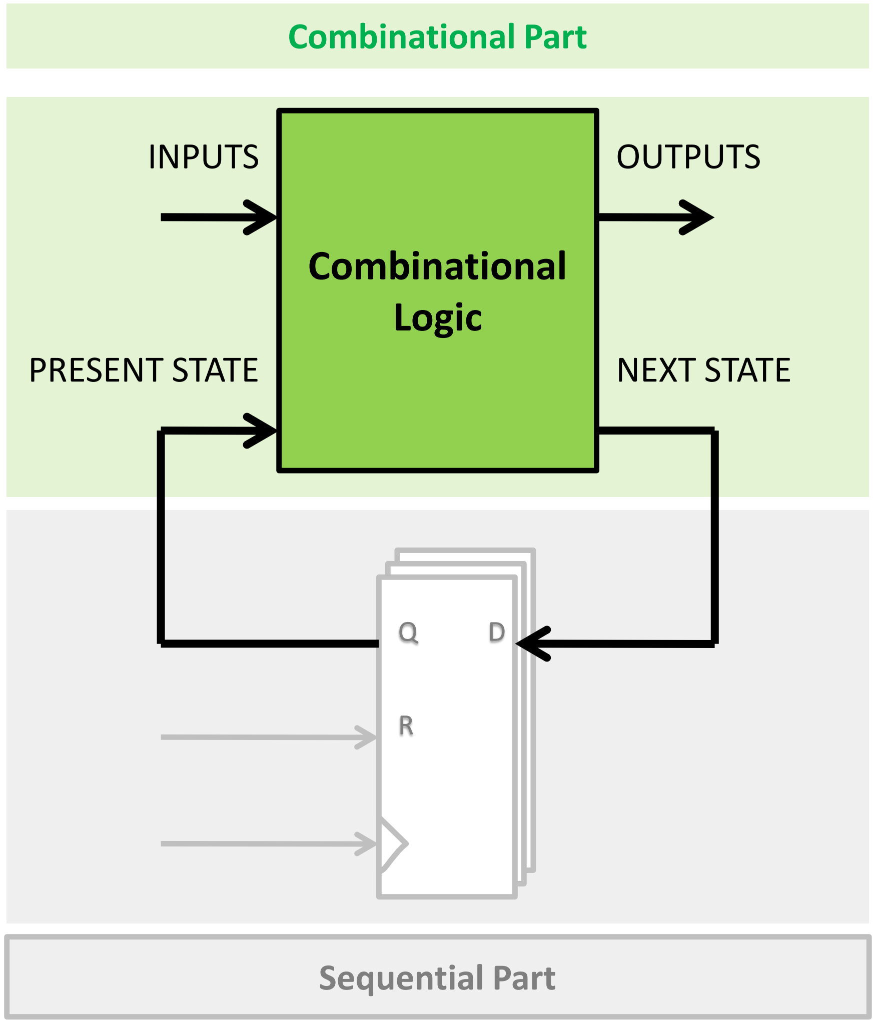
- Normally the 2 log of the number of states is the state bus size (present and next state signals)

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

The Combinational Part

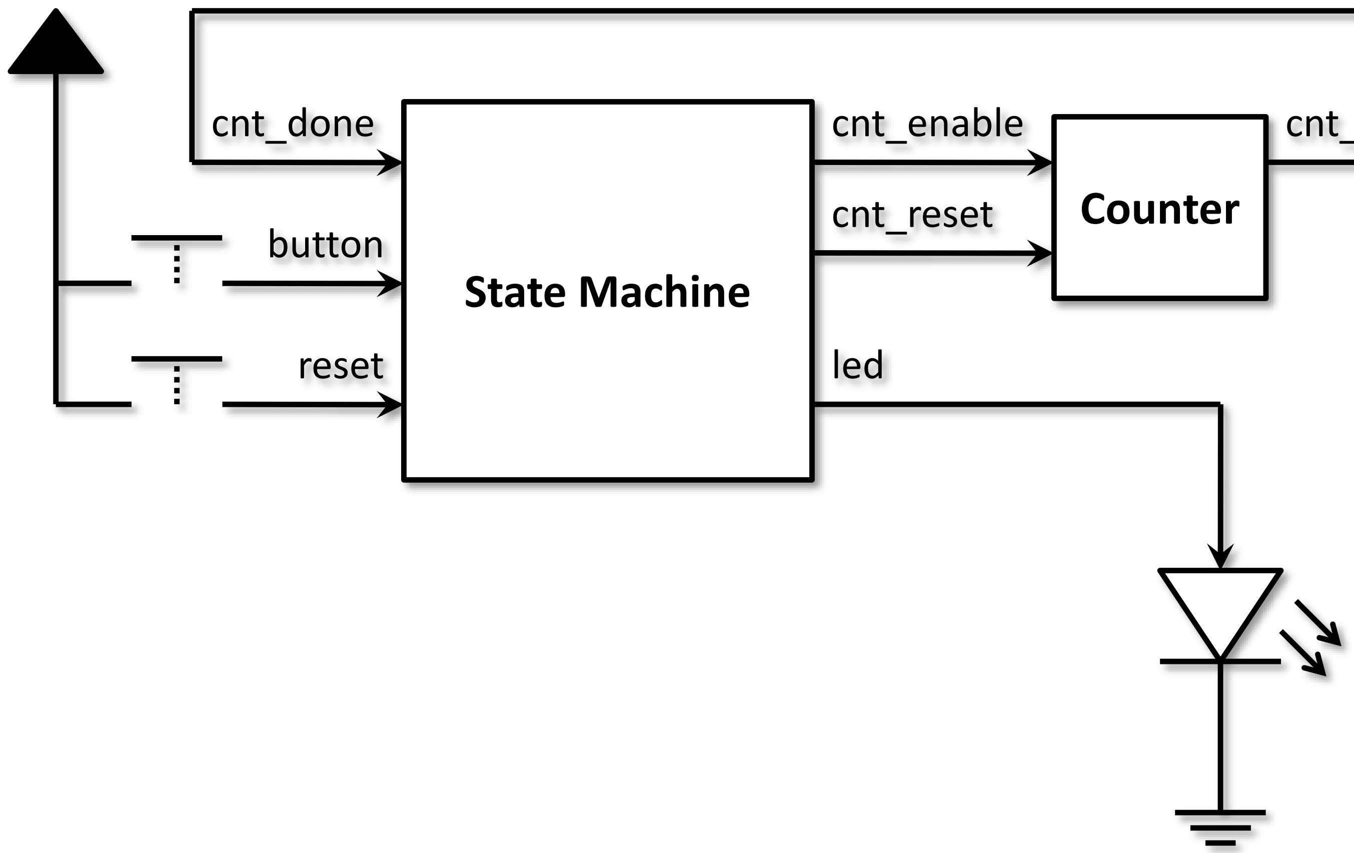


- The Combinational Part is Combinational Logic
- It is a (simple) Boolean Function
- We can draw a truth-table for this
- We can use K-maps to minimize the logic

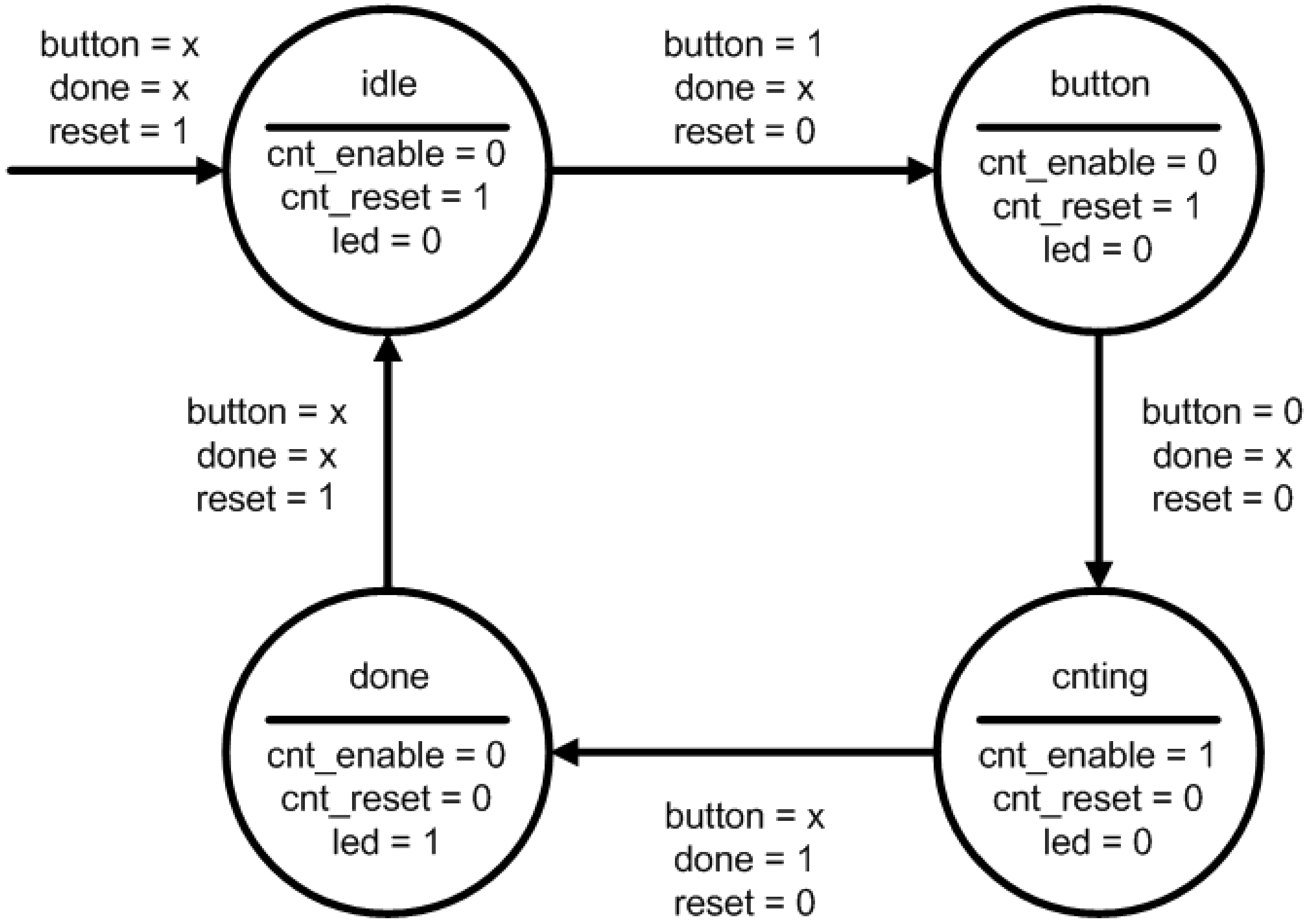
Agenda

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

Egg Timer: FSM Context



Egg Timer: State Diagram



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.

State Bus Encoding in VHDL

- We need to define what a state is (in bits)
- We need to draw the state bus signals 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.

```
architecture rtl of fsm_easy is

    -- Build 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
```

Manual Encoding

- We can also encode the state bus manually using constants
- It's not necessary most of the time, since the synthesizer can also be configured to use gray, one-hot, two-hot, binary, etc…

```
architecture rtl of fsm_easy is

    -- Build an enumerated type for the state machine
    constant idle   : std_logic_vector(1 downto 0) := "00";
    constant button: std_logic_vector(1 downto 0) := "01";
    constant cnting: std_logic_vector(1 downto 0) := "11";
    constant done   : std_logic_vector(1 downto 0) := "10";

    -- Register to hold the current state
    signal present_state,next_state : std_logic_vector(1 downto 0);

begin
```

Draw the flip-flops

```
-- state register
   pr_flipflops: process (clk, reset)
   begin
       if reset = '1' then
           present_state <= idle;
       elsif (rising_edge(clk)) then
           present_state <= next_state;
       end if;
   end process;
```

```
 -- logic to determine the next state
    pr_next_state: process (present_state, btn, cnt_done)
    begin
        case present_state is
            when idle =>
                if btn = '1' then
                    next_state <= button;
                else
                    next_state <= idle;
                end if;
            when button =>
                if btn = '0' then
                    next_state <= cnting;
                else
                    next_state <= button;
                 end if;
        when cnting =>
            if cnt_done = '1' then
                next_state <= done;
            else
                next_state <= cnting;
            end if;
        when done =>
                -- this isn't really necessary
            if reset = '1' then
                 next_state <= idle;
            else
                next_state <= done;
            end if;
     end case;
end process;
```

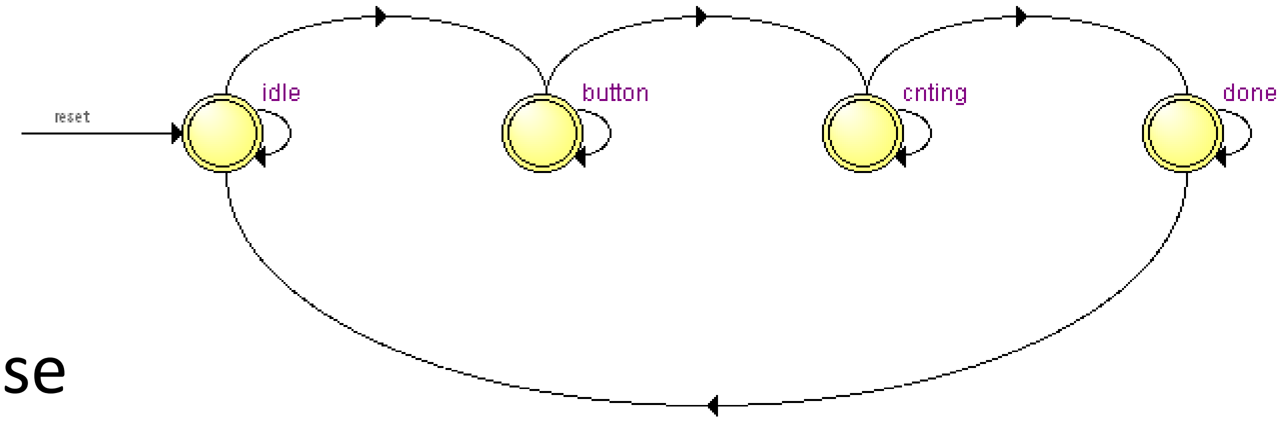
Write the transitions

Write the outputs (Moore)

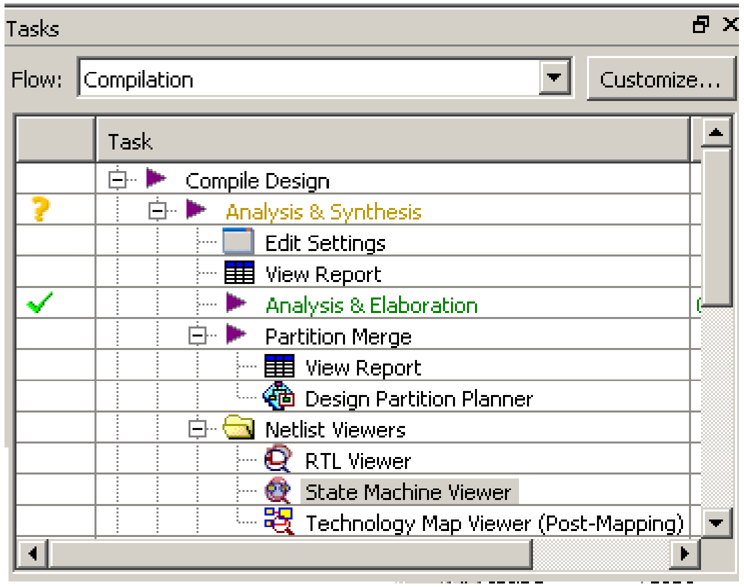
```vhdl
-- Logic to determine the outputs
   pr_outputs: process (present_state)
   begin
       case present_state is
           when idle =>
               cnt_enable  <= '0';
               cnt_reset   <= '1';
               led         <= '0';
           when button =>
               cnt_enable  <= '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 rtl;
```

Or combine next state and outputs!

VHDL Example



- When you use non-manual encoding, Quartus can detect a state machine in your code!



Advantages of using the template

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

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

Homework

- Covered today:
  - Book: Chapter 7 & Chapter 11
- Solve the following problems (book):
  - Problems 11.1, 11.8
- Next week:
  - Last presentation about some tips and tricks in VHDL