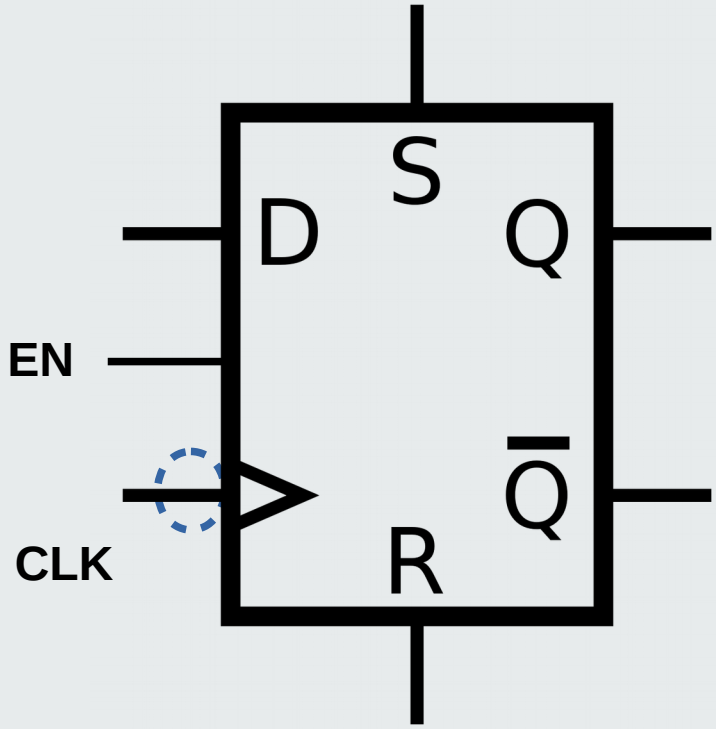
TÉCNICO LISBOA

Computer Electronics

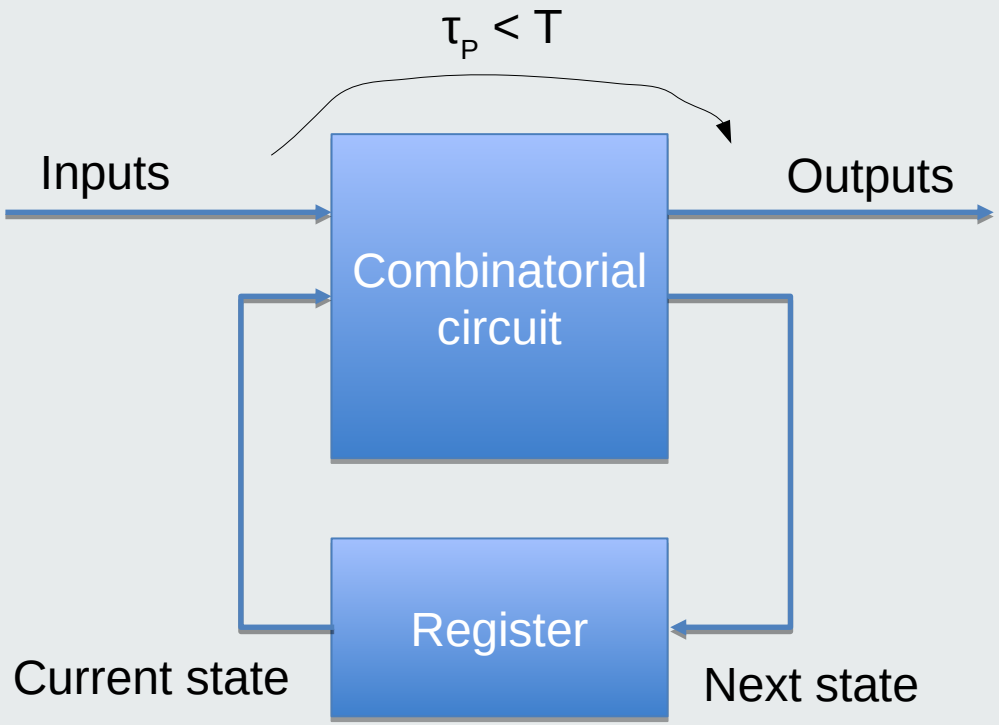
Lecture 7: Digital Circuits and Verilog II

The FLIP-FLOP!



- Only D type is studied
- Verilog description is directly mapped to a library component
- Set (S) and Reset (R) are optional
- It is rare that S and R are simultaneously supported: no library component
- Responds to the rising (positive) edge of the clock (CLK)
- Responds to the falling (negative) edge of the clock if an inverter is placed at the CLK input (DANGER ZONE!)
- The optional Enable signal (EN) determines whether the flip-flop will respond or not to the clock edge

Recap: just learn 2 things!



- If <u>any</u> digital circuit is just this then you just need to learn to describe:
  - combinatorial circuits
  - registers
- Not bad, is that?

Signal wires

- Verilog describes circuits
- Circuits are blocks connected by wires that carry digital signals (0 or 1)
- So we need to declare and use wires
- Verilog describes two types of wires: *wire* and *reg*
- And the mess begins:
  - A *wire* is a wire but it cannot be used in all situations
  - A *reg* is a wire (not a register as the name suggests) but it cannot be used in all situations
- What a start!!!

Iob-lib: making digital hardware design easy

- https://github.com/IObundle/iob-lib
- The **iob-lib** repo contains
  - Useful Verilog **macros** to avoid writing the same Verilog description over and over again
  - Useful **components** (modules) to avoid writing the same Verilog description over and over again
- Enjoy!

IOb-lib: input and output ports

- To declare a module:

```
module my_module
(
  `INPUT(X,8),   //declare 8-bit input
  `INPUT(Y,8),
  `OUTPUT(Z,16) //declare 16-bit output
  `INOUT(F,32) //declare 32-bit bidirectional port
);
   <body of module>
endmodule
```

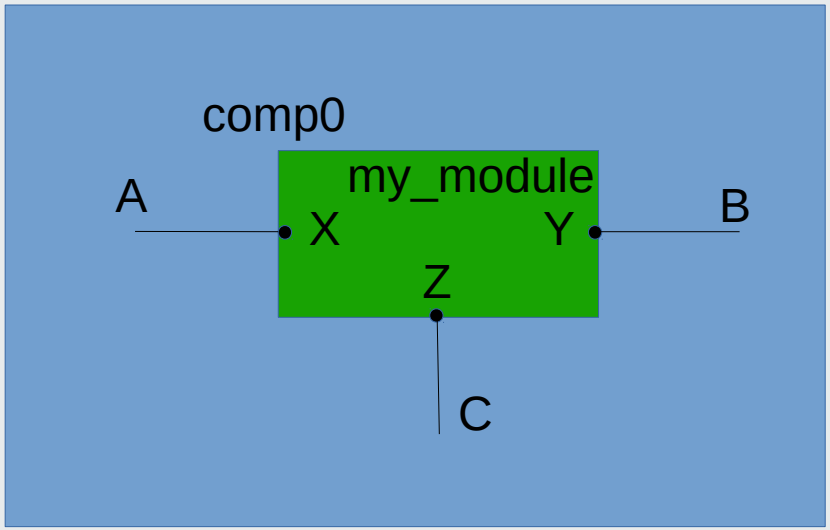
IOb-lib: instantiate components

- To instantiate a component is same as in Verilog:

```
module my_system
(
`OUTPUT( C),
 //interface signals
);
   <body verilog code: preamble>

   my_module comp0 //component instance
   (
      .X(A), //component port
      .Y(B),
      .Z ( C), //is output
      .F (D)
   )

   <body verilog code: trailer>
endmodule
```



IOb-lib: signals

- To declare a signal:

  `SIGNAL(my_wonder_signal, 10)
- With normal Verilog it would be
- wire [9:0] my_wonder_signal;
- reg [9:0] my_wonder_signal;
- wire [0:9] my_wonder_signal;
- Etc...

IOb-lib: signed signals

- To declare a signed signal:

  `SIGNAL_SIGNED(my_wonder_signal, 10)
- With normal Verilog it would be
- wire signed [9:0] my_wonder_signal;
- **reg signed [9:0] my_wonder_signal;**
- wire signed [0:9] my_wonder_signal;
- Etc...

IOb-lib: special signals that can be used as outputs

- To get around the complexities of Verilog a special signal type is created in iob-lib:

  `SIGNAL_OUT(my_special_signal, 10)

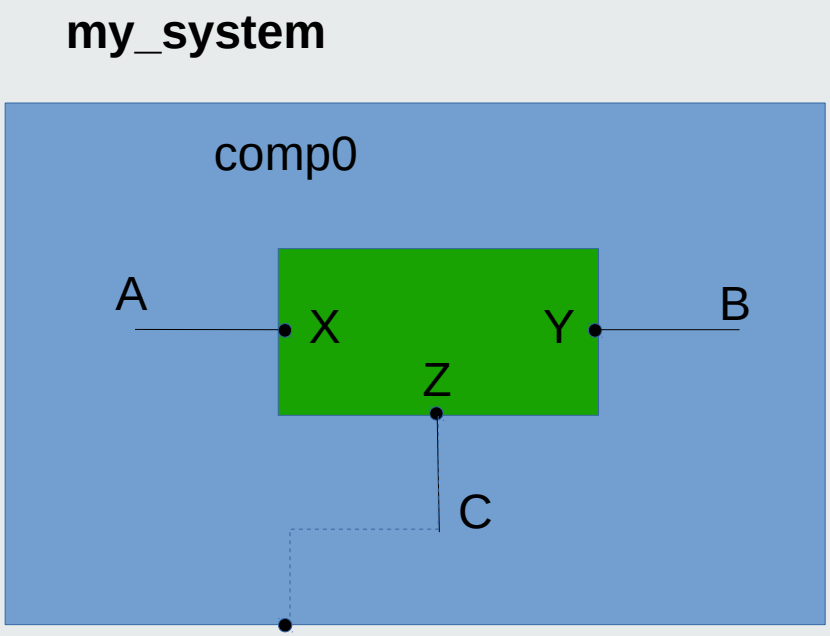
- Used to connect to component outputs
- Connecting a `SIGNAL to a component output is an ERROR
- Cannot be used as the result of computations (to be explained)
- Using `SIGNAL_OUT as the result of a computation is an ERROR
- With normal Verilog it would be
  - You don’t want to know!!

IOb-lib: using `SIGNAL_OUT

```
module my_system
(
 //interface signals
//`OUTPUT(C,16) is also an alternative
);
   <body verilog code: preamble>
   `SIGNAL_OUT(C,16)      //`SIGNAL(C,16) is an ERROR)

   my_module comp0 //component instance
   (
      .X(A), //component port
      .Y(B),
      .Z(C)
  )

   <body verilog code: trailer>
endmodule
```

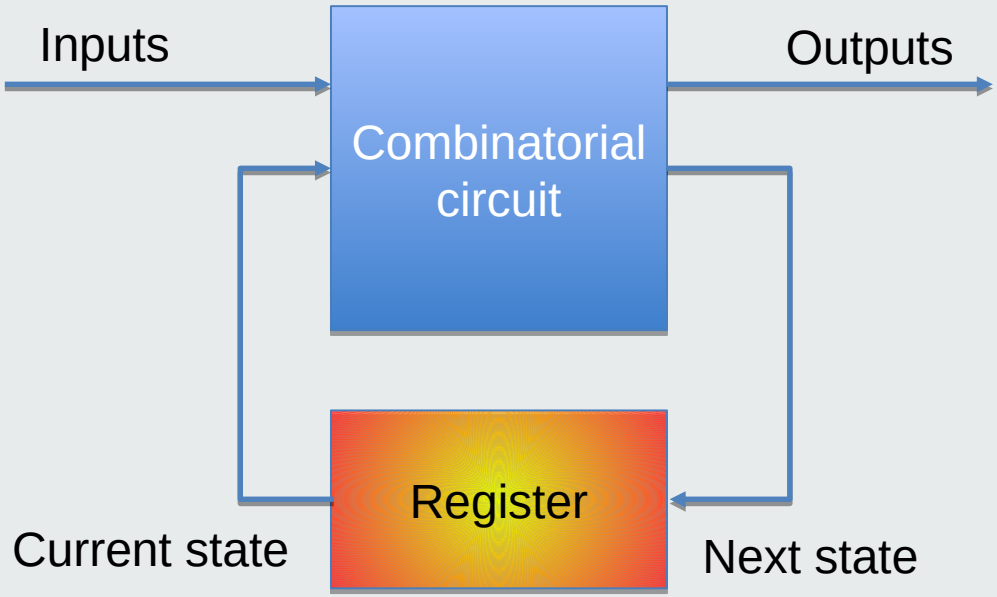


IOb-lib: flip-flops

- Flip-flop: stores 1 bit

```
`SIGNAL(D,1)   //declare input signal D
`SIGNAL(Q,1)   //declare output signal Q
`REG(clk, Q, D) //instantiate one flip-flop
```

IOb-lib: registers!



IOb-lib: registers

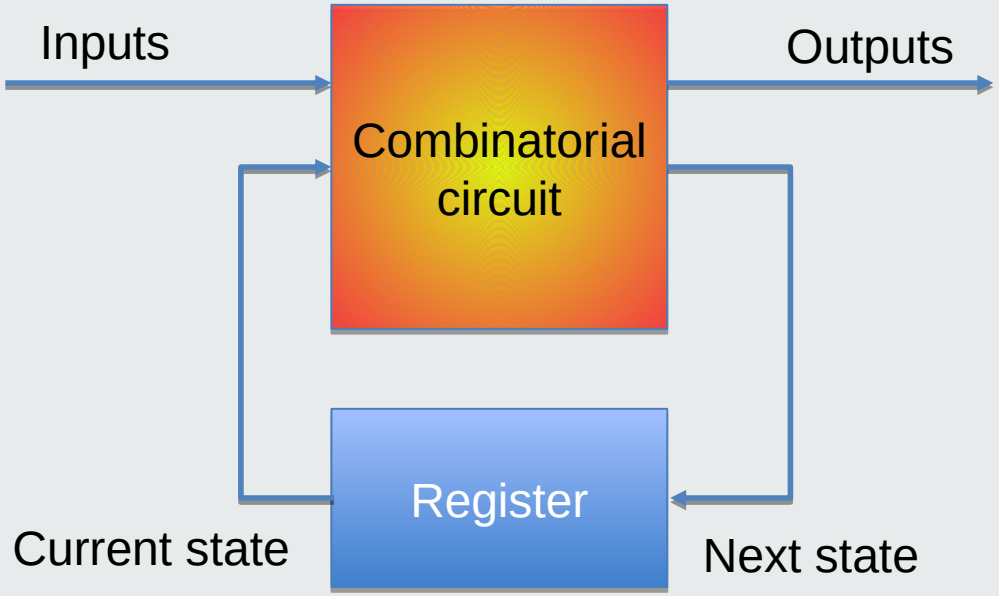
- Register: stores W bits

`SIGNAL(D,**10**) //declare input signal vector D

`SIGNAL(Q,**10**) //declare output signal vector Q

`REG(clk, Q, D) //instantiate register or flip-flop //array

IOb-lib: combinatorial circuits



IOb-lib: combinatorial blocks

```
module my_system
(
 //interface signals
);
...
   `SIGNAL(z,10);
   `SIGNAL(y,10);
   `COMB begin
      //insert C-like expressions here: examples
      z = s? X: Y;
      y = a*(b+c);
      //IMPORTANT: z and y MUST be of type `SIGNAL: type `OUTPUT or `SIGNAL_OUT are ERROR!!!
   end
...
endmodule
```

IOb-lib: combinatorial block result to output problem

- So how to assign an output to the output of a combinatorial circuit?
- Answer: use signal to output conversion function

```
module my_system
(
`OUTPUT(Z, 10);
 //other interface signals
);
…
   `SIGNAL(z,10)
   `COMB begin
       ...
       z = s? X: Y;
       ...
   end

`SIGNAL2OUT(Z,z)
…
endmodule
```

IOb-lib: order of statements

```
module my_system
(
 //interface signals
);
...
   `SIGNAL(z,10); //important to keep here
   `SIGNAL(y,10);

   `SIGNAL2OUT(Z,z) //not important to keep here
   `COMB begin
      //insert C-like expressions here: examples
      z = s? X: Y;
      y = a*(b+c);
      //IMPORTANT: z and y MUST be of type `SIGNAL: type `OUTPUT or `SIGNAL_OUT are ERROR!!!
   end
   //`SIGNAL2OUT(Z,z) //could be here...
endmodule
```

- In Verilog the order of statements is **irrelevant** since we are describing things, not procedures!
- Signal declarations must precede their use
- Inside a combinatorial block order is important because it is ***described procedurally***
- Only the result of the combinatorial block matters and this may be affected by order