Lab 10.1

Lab 10.1

- In this lab you will:
  - Learn how to infer resetable flip-flops with fan-in logic
- In directory lab10.1 you will find the following files:
  - lr_shift.v Loadable shifter (incomplete)
  - tb.v Self-checking testbench

Lab 10.1

- The 4-bit shifter has the following functions:
  - Reset to "oooo", if "RST" is asserted
  - Shifts right, shifting in 'o', if "RT" is asserted
  - Shifts left, shifting in 'o', if "LF" is asserted
  - Parallel loads all 4 bits in "I", if "LD" is asserted
  - Stays in the same state, if no control is asserted
  - The content of the shifter is available at all times on the output "O"
  - All control inputs are asserted HIGH and are guaranteed to be mutually exclusive (i.e. "RST" and "LD" cannot be asserted at the same time)
  - The clock is active on the rising edge

Lab 10.1

- Add the necessary statements in the module to complete a synthesizable functional description of the shifter.
- Verify the correctness of your description by simulating the decoder model using the provided testbench.

```
% ... tb.v lr_shift.v
```

- Once the functionality is declared correct, synthesize your design.
- Compare the performance of your design with that of others and with the proposed solution.
  - Compare the description of your design with another and try to understand why one is faster and/or smaller.

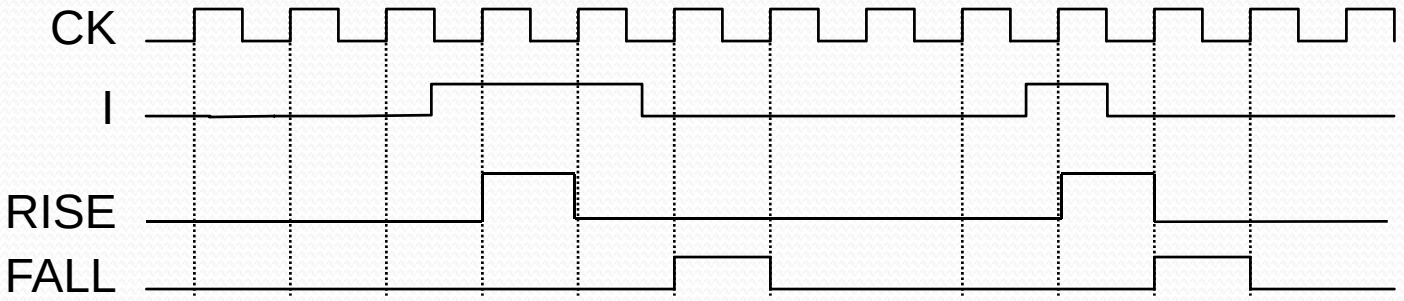
Lab 10.2

Lab 10.2

- In this lab you will:
  - Learn how to allocate fan-in and decode logic in various *always* blocks
- In directory lab10.2 you will find the following files:
  - edge_detect.v Edge detector (incomplete)
  - tb.v Self-checking testbench

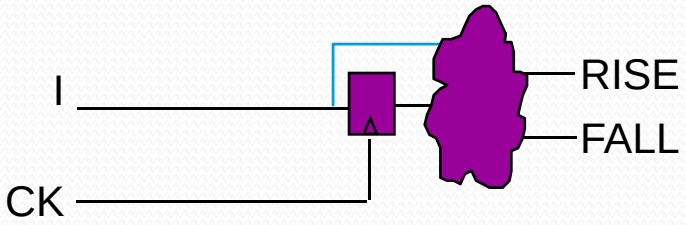
Lab 10.2

- This circuit monitors the input I for rising and falling edges. This circuit asserts the output "RISE" for 1 clock cycle if a rising edge is detected, and the output "FALL" if a falling edge is detected.



Lab 10.2

- Use the following architecture for the circuit.



- Before starting coding your RTL code, circle the logic that can be implemented in a single *always* block, then write the RTL accordingly.

Lab 10.2

- Add the necessary statements in the module to complete a synthesizable functional description of the edge detector.
- Verify the correctness of your description by simulating the decoder model using the provided testbench.

```
% ... tb.v edge_detect.v
```

- Once the functionality is declared correct, synthesize your design.
- Compare the performance of your design with that of others and with the proposed solution.
  - Compare the description of your design with another and try to understand why one is faster and/or smaller.

Lab 10.2: Optional

- Modify your description to eliminate the output decoder and provide the RISE and FALL output directly from flip-flops.

Lab 10.3

Lab 10.3

- In this lab you will:
  - Learn how to model a Finite State Machine
- In directory lab10.3 you will find the following files:
  - gray.v — Gray counter (incomplete)
  - tb.v — Self-checking testbench

Lab 10.3

- A gray counter cycles to a sequence where only a single bit changes between states.
  - Circuits with gray-coded inputs are glitch-free and consume less power.
- Describe an asynchronously resetable 3-bit gray counter as a state machine.
  - Reset is active high.
  - Clock is active on the rising edge.
- Any gray sequence is acceptable and any state can be the reset state.
  - Example gray sequence:

```
000   110
001   111
011   101
010   100
```

Lab 10.3

- Add the necessary statements in the module to complete a synthesizable functional description of the gray counter.
- Verify the correctness of your description by simulating the decoder model using the provided testbench:

```
% ... tb.v gray.v
```

- Once the functionality is declared correct, synthesize your design.
- Compare the performance of your design with that of others and with the proposed solution.
  - Compare the description of your design with another and try to understand why one is faster and/or smaller.

Lab 10.4

Lab 10.4

- In this lab you will:
  - Learn how to deal with multiple clock domains
- In directory lab10.4 you will find the following files:
  - fifo.v FIFO (incomplete)
  - tb.v Self-checking testbench

Lab 10.4

- Describe a 3-deep FIFO where 8-bit input data is written on the rising edge of CKI, and then if the FIFO is empty, the input data becomes immediately available on the output.
- The output data is consumed and the next available word is presented on the output after a rising edge on CKO.
- Provide two status flags: "MT" and "FL" that indicate if the FIFO is empty or full, respectively.
  - When "MT" is asserted (HIGH), the output value is invalid.
  - Writing a new value when "FL" is asserted (HIGH) yields unpredictable results.
- Reset is asynchronous and active HIGH.

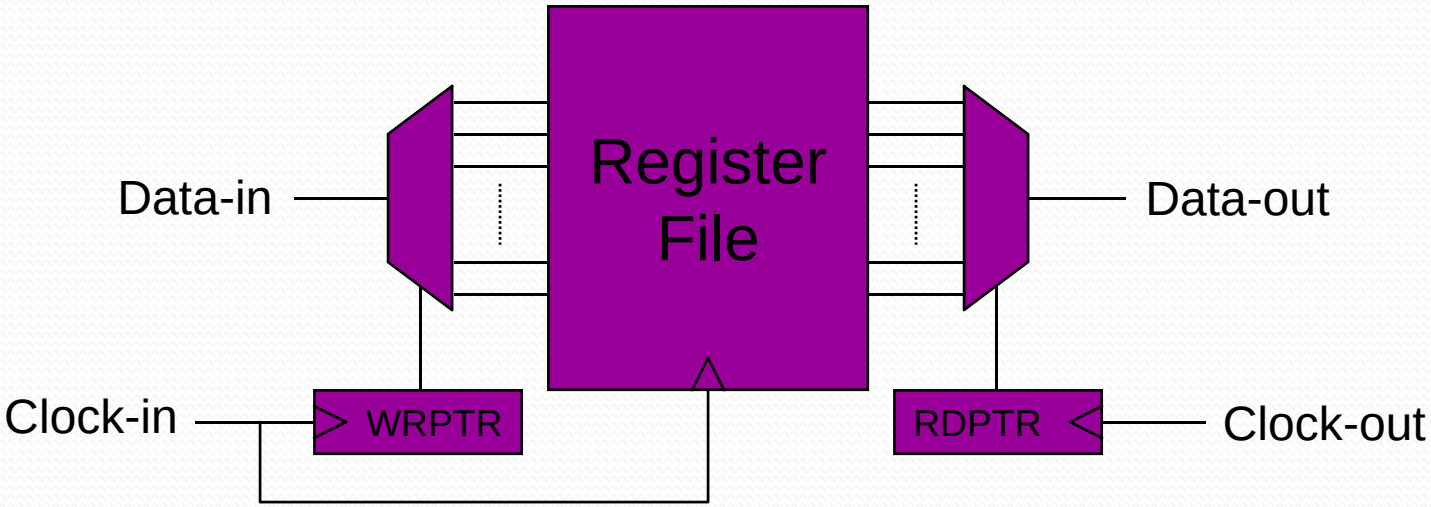
Lab 10.4

- Example of a sequence of operations:

Input	CKI	CKO	Output	MT	FL
-	0/1	0/1	x	1	0
0	^	0/1	0	0	0
1	^	0/1	0	0	0
2	^	0/1	0	0	1
-	0/1	^	1	0	0
3	^	0/1	1	0	1
-	0/1	^	2	0	0
-	0/1	^	3	0	0
-	0/1	^	x	1	0

Lab 10.4

- Use the following architecture:



Lab 10.4

- If you have difficulty figuring out how to implement the "MT" and "FL" flags, read the hint below.

The flags are much easier to compute if you implement an extra location in the FIFO even though it is not really “available”.

Lab 10.4

- To complete a synthesizable functional description of the FIFO, add the necessary statements in the module.
- Verify the correctness of your description by simulating the decoder model using the provided testbench:

```
% ... tb.v fifo.v
```

- Once the functionality is declared correct, synthesize your design.
- Compare the performance of your design with that of others and with the proposed solution.
  - Compare the description of your design with another and try to understand why one is faster and/or smaller.

Lab 10.5

Lab 10.5

- In this lab you will:
  - Become familiar with setting constraints
  - See how constraints affect the synthesis outcome
  - Witness how operating conditions affect timing estimates
  - Examine timing reports

Lab 10.5

- In directory ~/lab10.5, you will find the following files:
  - control.v
- Follow the remaining instructions that pertain to the synthesis tool you are using.

Lab 10.5: Synopsys

- Start the command-line interface of Synopsys using the following command:

```
% dc_shell
```

- Read the design using the command:

```
dc_shell> read -format verilog control.v
```

- Save the current state of the design by using the command:

```
dc_shell> write
```

Lab 10.5: Synopsys

- Compile the design with no constraints, then examine the final timing and area results:

```
dc_shell> create_clock -period 10 CLK
dc_shell> compile
dc_shell> report_timing
dc_shell> report_area
```

- Add a model for the clock distribution network, then examine how the timing report changes:

```
dc_shell> set_clock_skew -uncertainty 2 CLK
dc_shell> report_timing
dc_shell> set_clock_skew -delay 1 CLK
dc_shell> report_timing
```

Lab 10.5: Synopsys

- Display the available operating conditions, then select the one you can identify as being the worst case:

```
dc_shell> report_lib qyh500
...
dc_shell> set_operating_conditions name
```

- Look at the timing report and see how it is affected:

```
dc_shell> report_timing
```

- Delete the current design and restart after compilation:

```
dc_shell> remove_design
dc_shell> read CONTROL.db
```

Lab 10.5: Synopsys

- Delete the current design and restart after compilation:

```
dc_shell> remove_design
dc_shell> read CONTROL.db
```

- Constrain the design, using worst-case timing:

```
dc_shell> set_operating_conditions name
dc_shell> create_clock -period 8 CLK
dc_shell> set_input_delay 1.5 -clock CLK all_inputs()
dc_shell> set_output_delay 5 -clock CLK all_outputs()
```

- Compile the design. Did it meet its constraints?

```
dc_shell> compile
dc_shell> report_timing
```

Lab 10.5: Synopsys

- Write out a Verilog netlist:

```
dc_shell> write -format verilog -output gates.v
```

- You are done!

```
dc_shell> quit
```

Lab 10.5: Leonardo

- Start the graphical interface of Leonardo using the following command:

```
% leonardo
```

- Read the design using the command:

```
LEONARDO: load_library flex8
LEONARDO: read control.v
```

- Save the current state of the design by using the command:

```
LEONARDO: write -format verilog control_orig.v
```

Lab 10.5: Leonardo

- Compile the design with no constraints, then examine the final timing and area results:

```
LEONARDO: set_attribute -port clk -name CLOCK_CYCLE -value 10
LEONARDO: optimize -target flex8 -area
LEONARDO: report_delay -critical_paths
LEOANRDO: report_area -cell_usage
```

- Add a model for the clock distribution network, then examine how the timing report changes:

```
LEONARDO: set_attribute -port CLK -name CLOCK_OFFSET -value 2
LEONARDO: report_delay -critical_paths
```

Lab 10.5: Leonardo

- Write out a Verilog netlist:

```
LEONARDO: write -format verilog gates.v
```

- You are done!

```
LEONARDO: quit
```