Verilog LAB I

Lab 1 – 1 bit full adder

- Use Karnaugh Map to derive logic expression for the addition.
- Create the file OneBitAdder.v
  - Write the Verilog code that implements binary addition operation based on the expression obtained previously
- Use the following interface

```
module onebitadder (
  input  a, b, cin,
  output sum, cout
);
```

a	***b***	***cin***	***sum***	***cout***
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Lab 2 – Basic modules

- This lab will consist on creating some basic designs (modules).
- A description of the design functionality will be provided.
- The objective is to translate the description into a synthesizable Verilog code.
- It is very important to use the module interface that will be provided.
- In order to do a syntax check on the Verilog code written, `vlogan` tool can be used. The following line can be executed on the shell

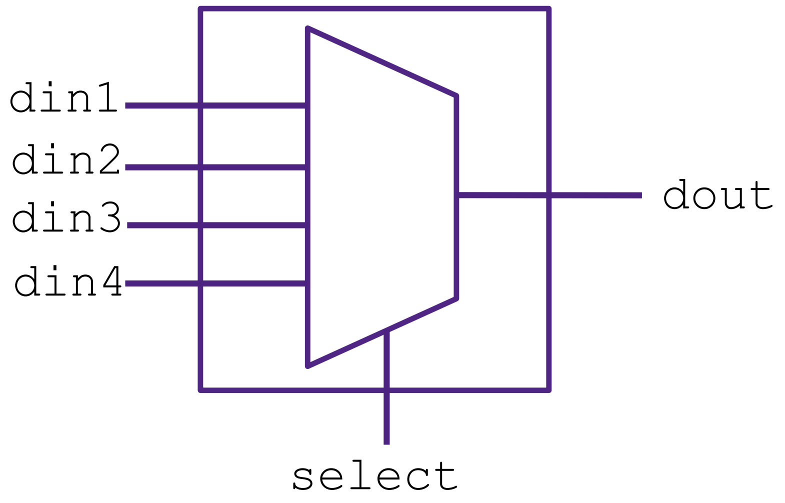
```
vlogan <verilog_file.v> –full64 –sverilog +v2k
```

Lab 2a – Parameterized 4-input MUX

- Write the Verilog code that implements a 4-input multiplexer (mux) with variable (parameterized) input and output bus width.

- Use the following interface

```
module mux4 #(
  parameter WIDTH= 8
  ) (
  input  [WIDTH-1:0] din1, din2, din3, din4,
  input  [1:0]       select,
  output [WIDTH-1:0] dout
);
```

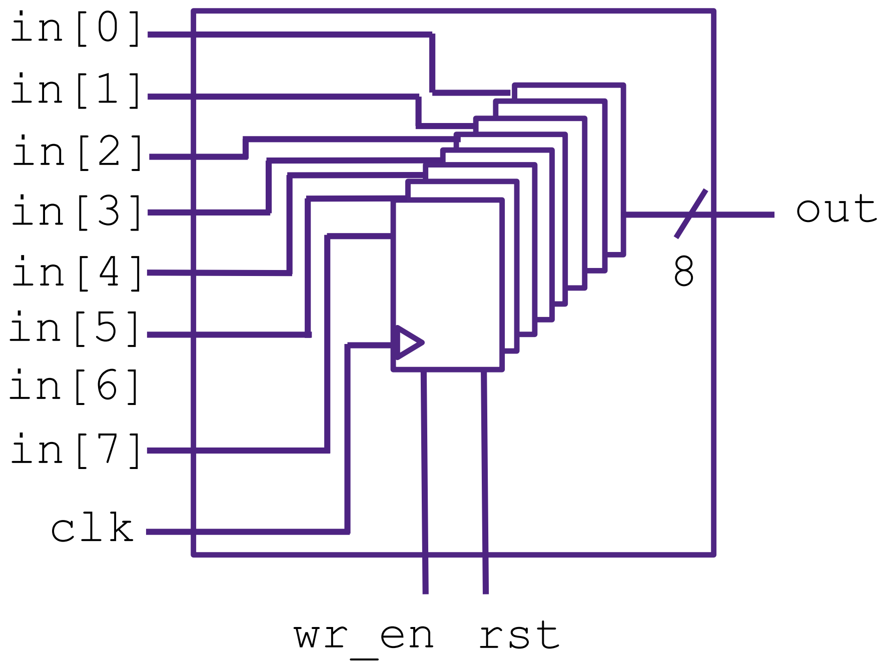


Lab 2b – Parameterized Register Bank

- Write the Verilog code that implements a D-type Register Bank with parameterized depth. It should:
  - Operate on positive edge of the clock (`clk`)
  - Have a Synchronous reset (`rst`)
  - Have an enable signal (`wr_en`) which allows data capturing only when asserted.

- Use the following interface

```
module register_bank #(
  parameter WIDTH = 8
  ) (
  input  [0:0]       clk,
  input  [0:0]       rst,
  input  [0:0]       wr_en,
  input  [WIDTH-1:0] in,
  output [WIDTH-1:0] out
);
```

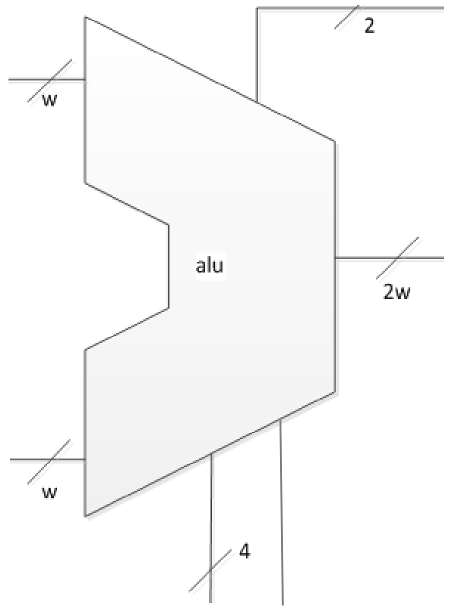


Lab 2c – Basic ALU

- Write the Verilog code for a basic ALU module (Arithmetic Logic Unit).
  - This ALU has to have addition, subtraction, multiplication and division operations.
    - It should have two variable (parameterized) width inputs and output bus in accordance.
    - Signal `zero[0]` should be asserted when the result of the current operation is 0.
    - Signal `error[0]` should be asserted when dividing by 0 is attempted or when input data is **not** valid.
      - On error condition, output must be forced to be `-1`
  - Operations should be described using Verilog arithmetic operators.

- Use the following interface

```
module ALU #(
  parameter WIDTH = 8
  ) (
  input  [WIDTH-1:0]   in1, in2,
  input  [3:0]         op,
  input  [0:0]         nvalid_data,
  output [2*WIDTH-1:0] out,
  output [0:0]         zero,
  output [0:0]         error
);
```



SYNOPSYS®
Silicon to Software™

Thank You