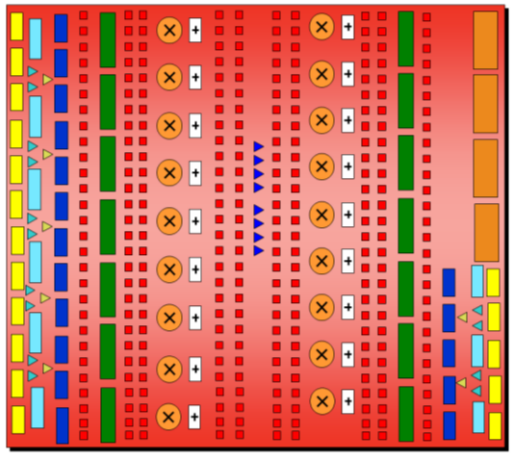
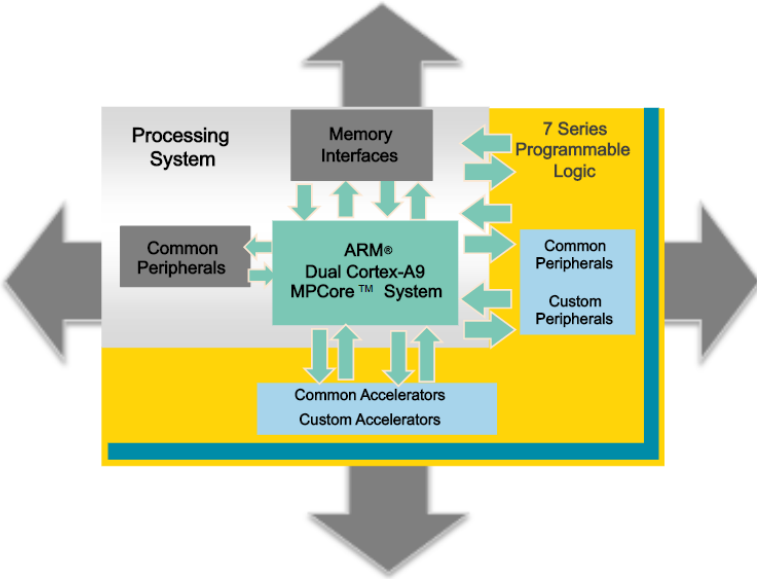
DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

A2A
Algorithms to Architecture

ECE 270: Embedded Logic Design



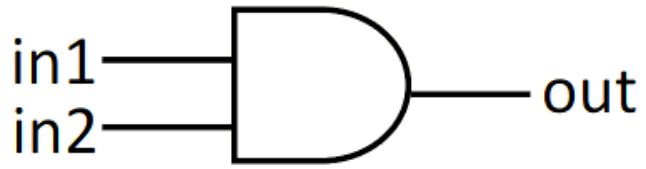


Verilog

Few words of wisdom

- One of the common mistakes for beginners is to think of HDL as a computer program rather than as a shorthand for describing digital hardware.
- If you do not know approximately what hardware your HDL should synthesize into, you probably won't like what you get.
- You might create far more hardware than is necessary or you might write non-synthesizable code
- **THINK** of your system in terms of blocks of combinational logic, registers and FSMs. **SKETCH** these blocks on paper and show how they are connected **BEFORE** you start writing code.
- Describing hardware with a language is similar, however, to writing a parallel program

Verilog: Module (Examples)




```
module AND (out, in1, in2) ; // <module name> <ports list>
    input in1, in2 ;
    output wire out ;
    assign out = in1 & in2 ;    // data flow - continuous Assignment
endmodule
```

```
module AND (out, in1, in2) ; // <module name> <ports list>
    input in1, in2 ;
    output reg out ;
    // must be reg type when used as LHS in an always block
    always @( in1 or in2) // always block (sensitivity list) - behavioral
        out = in1 & in2 ;    /* statements inside always block are
executed only when one or more signals in the list changes value */
endmodule
```

Verilog

```
module exam_1 (o1,in1,in2,in3); //module
    output o1;
    input in1,in2,in3;
        and a1 (Q, in1, in3); //and instantiation
        not n1 (P,in3);          // not instantiation
        and a2 (R, in2, P);  // and instantiation
        or    o1 (o1, Q, R); // or instantiation
Endmodule
```

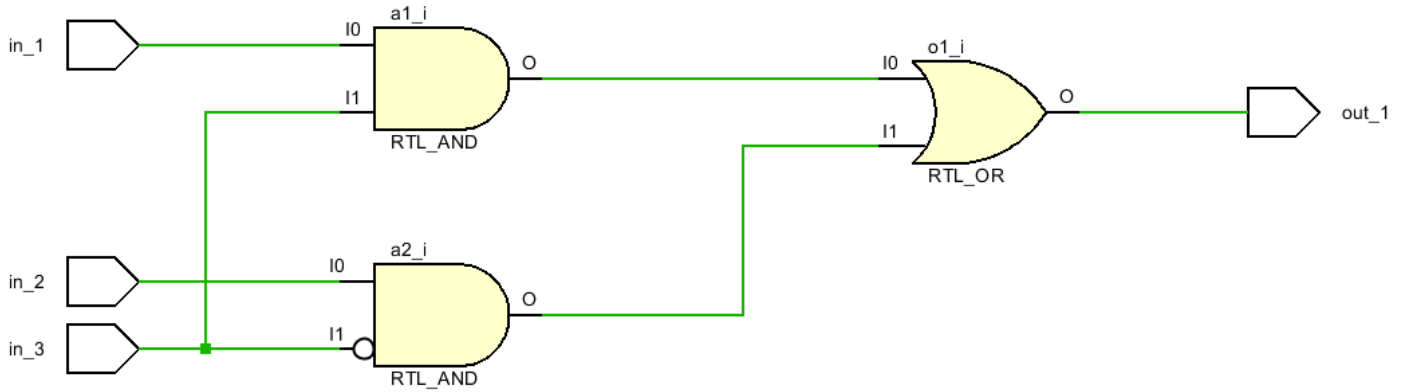
HDL Bits (Lab Homework 1: Due on Thursday)

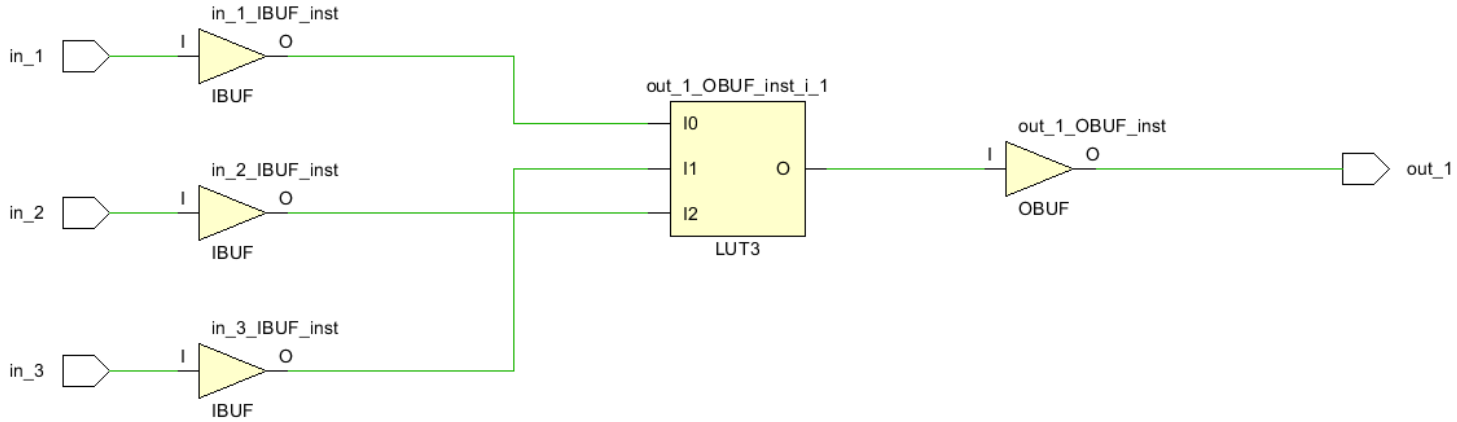
Submit Report from My Stats Page

1. https://hdlbits.01xz.net/wiki/Wire
2. https://hdlbits.01xz.net/wiki/Wire4
3. https://hdlbits.01xz.net/wiki/Notgate
4. https://hdlbits.01xz.net/wiki/Andgate
5. https://hdlbits.01xz.net/wiki/Norgate
6. https://hdlbits.01xz.net/wiki/Xnorgate
7. https://hdlbits.01xz.net/wiki/Wire_decl
8. https://hdlbits.01xz.net/wiki/7458
9. https://hdlbits.01xz.net/wiki/7420
10. https://hdlbits.01xz.net/wiki/Mt2015_q4a
11. https://hdlbits.01xz.net/wiki/Mt2015_q4b
12. https://hdlbits.01xz.net/wiki/Mux9to1v

Verilog

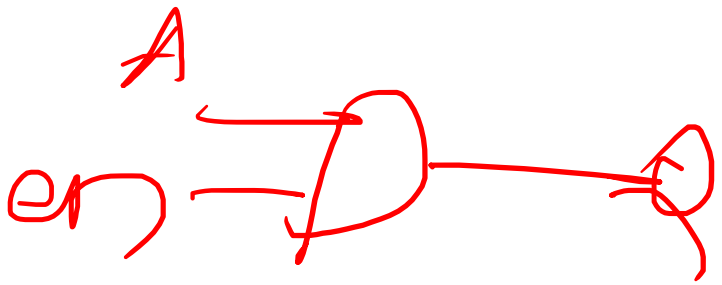
```
module test_1(
    input in_1,
    input in_2,
    input in_3,
    output out_1
    );

    wire Q, P, R;
    and a1(Q, in_1, in_3); //and instantiation
    not n1(P,in_3);        // not instantiation
    and a2(R, in_2, P);  // and instantiation
    or   o1(out_1, Q, R); // or instantiation

endmodule
```





Verilog

```
module test_1(
    input in_1,
    input in_2,
    input en,
    output out_0,out_1,out_2,out_3
    );

   assign out_0 =(en & ~in_2 & ~in_1);
   assign out_1 =(en & ~in_2 & in_1);
   assign out_2 =(en & in_2 & ~in_1);
   assign out_3 =(en & in_2 & in_1);

endmodule
```

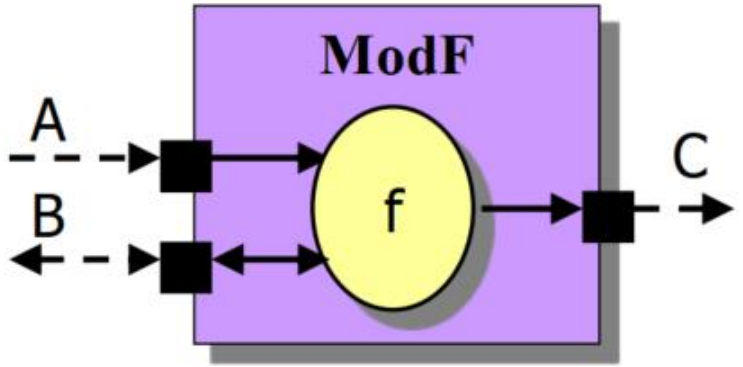


en	in_1	in_2	O_1	O_2	O_3	O_4
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	X	X	0	0	0	0

Verilog

```
module ex1
    // I/O ports
    (
     input wire i0, i1,
     output wire eq
    );

    // signal declaration
    wire p0, p1;

    // body
    // sum of two product terms
    assign eq = p0 | p1;
    // product terms
    assign p0 = ~i0 & ~i1;
    assign p1 = i0 & i1;

endmodule
```

Verilog: Module (Examples)




```
module module_name (port_list) ;
    declarations:
        port declaration (input, output, inout, …)
        data type declaration (reg, wire, parameter, …)
        task and function declaration
    statements:
        initial block                    }
        always block                     }  Behavioral
        module instantiation             }
        gate instantiation               }  Structural
        UDP instantiation                }
        continuous assignment            }  Data-flow
endmodule
```

```
module ModF ( input wire A
            , input wire [7:0] B
            , output wire [7:0] C);
   // declarations
   // description of 'f'
endmodule
```

```
module ModF (A, B, C);
   input        A;
   inout  [7:0] B;
   output [7:0] C;
   // declarations
   // description of 'f'
endmodule
```

Verilog

```
module test_1(
    input clk,
    input clr,
    input D_in,
    output reg Q_out
    );

   always@(*) begin
       if(clr)
           Q_out=1'b0;
       else if (clk)
           Q_out=D_in;
   end

endmodule
```

always @ (*) — comb.

always @ (posedge clk
negedge clk

+ Seq.

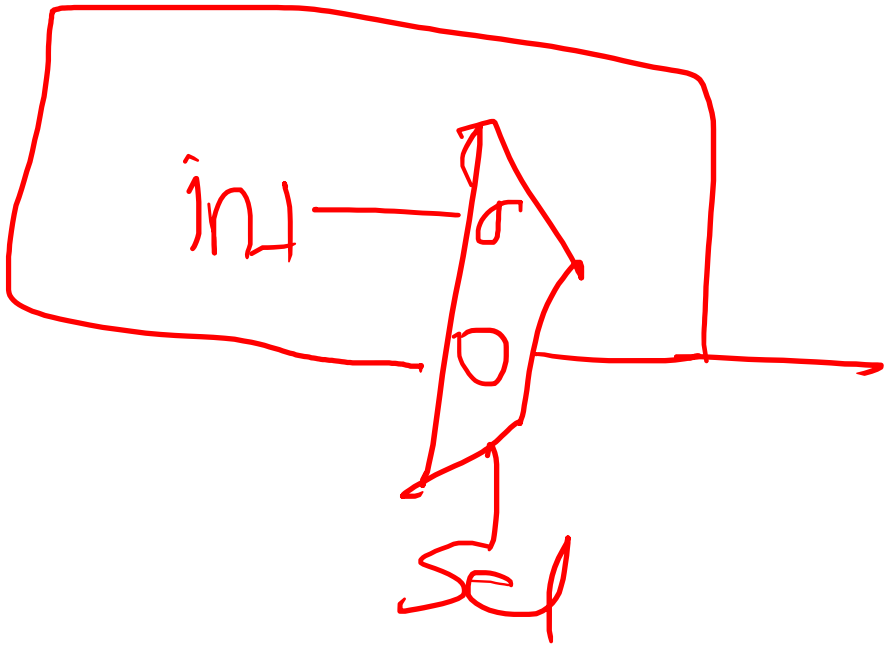
Mem

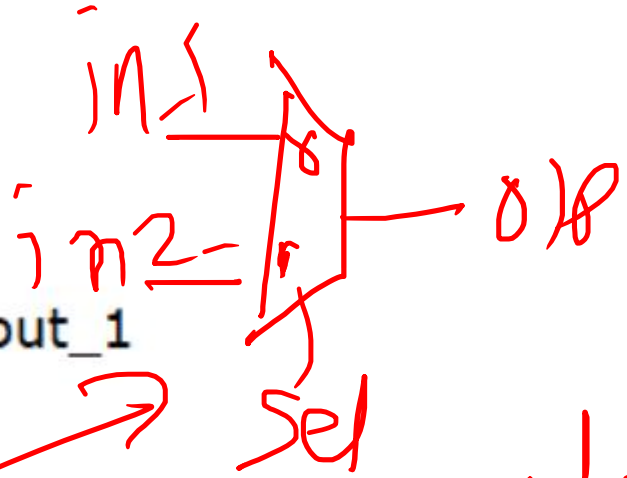
always @ (clr, clk, D_in)

active high Mem

Verilog:

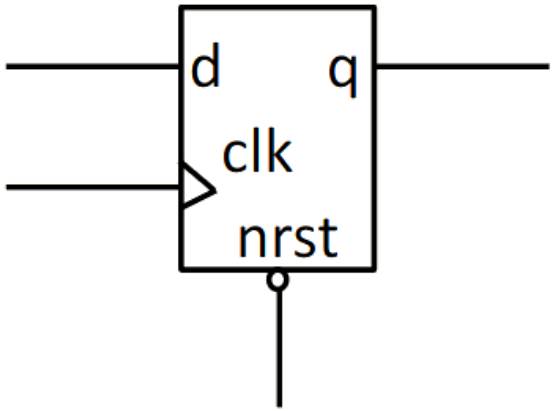
```
module test_1(
    input [2:0] in_1,
    input [2:0] in_2,
    input sel,
    output reg [2:0] out_1
    );

   always@(*) begin
       if(sel==1'b0)
          out_1=in_1;
       else
          out_1=in_2;
   end

endmodule
```





always@(*) begin
if(sel==1'b0)
out_1 = in_1;
end

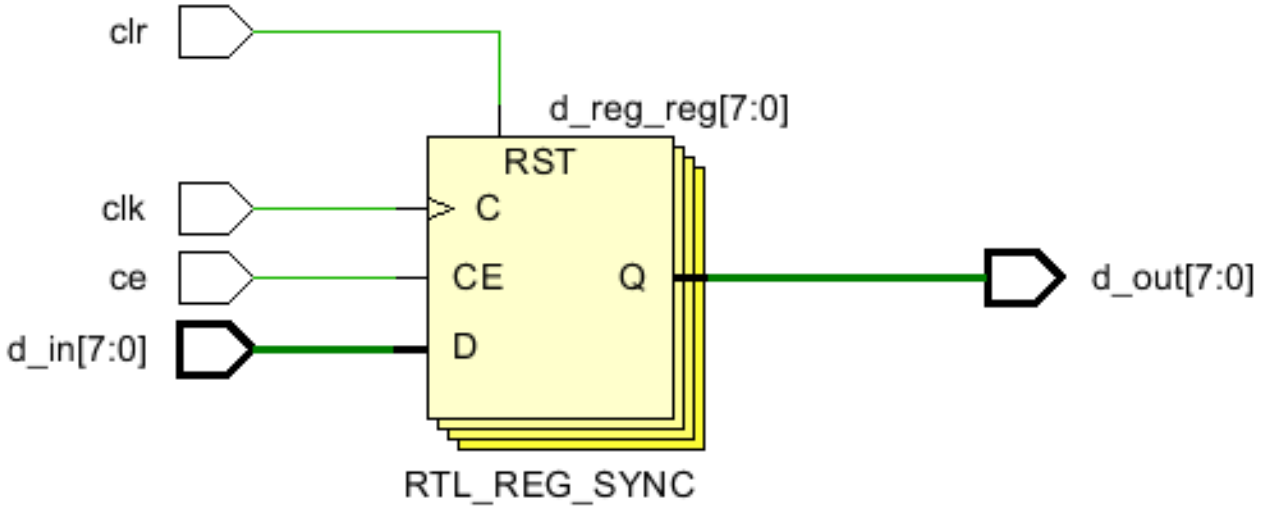
Verilog: Module (Examples)




```
module D_FF(clk, nrst, d, q) ;
    input clk, nrst, d ;
    output reg q ;
    always @(posedge clk or negedge nrst)
                                    // Event-based Timing Control
      if (!nrst)                    // reset state
        q <= 0 ;
      else                          // normal operation
       q <= d ;
endmodule
```

Verilog: Register

```
module test_1(
    input [7:0] d_in,
    input ce,
    input clk,
    input clr,
    output [7:0] d_out
    );

    reg [7:0] d_reg;
    always@(posedge clk)
    begin
        if(clr)
            d_reg <= 8'b00000000;
        else if (ce)
            d_reg <= d_in;
    end
    assign d_out = d_reg;
endmodule
```



Self Study

- Using module for 2:1 mux (data flow level approach), design 8:1 mux via module interconnections
- Design comparator for 2-bit inputs using data flow level approach.
- Using comparator for 1-bit inputs, design comparator for 2-bit inputs via module interconnections