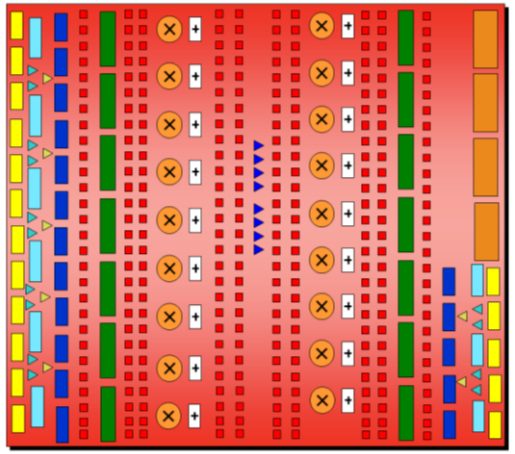
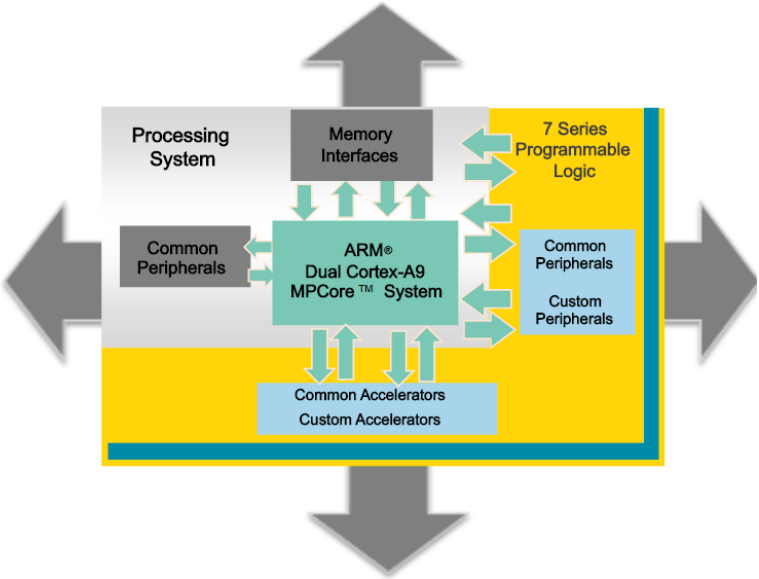
DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

A2A
Algorithms to Architecture

ECE 270: Embedded Logic Design

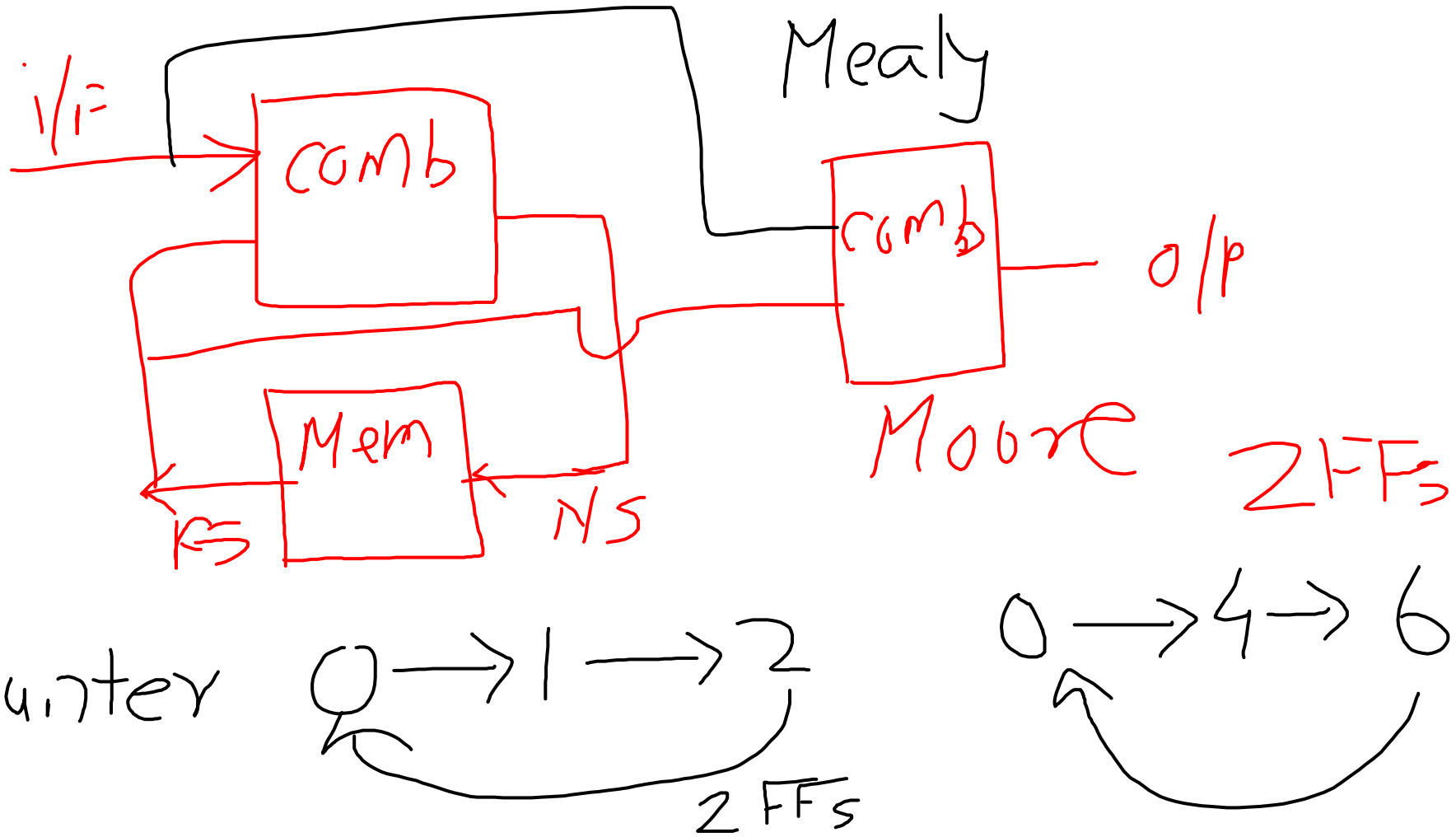




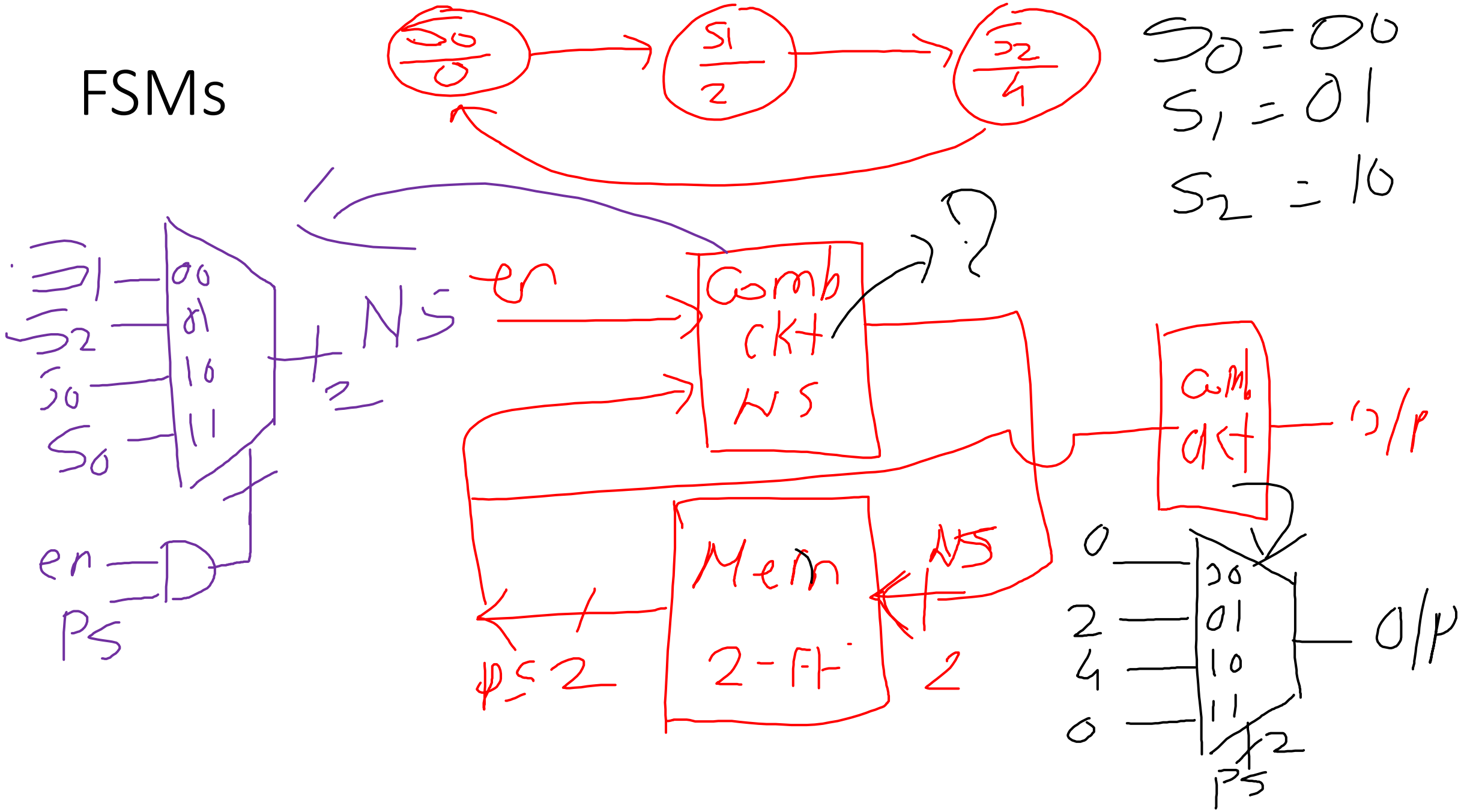
Dr. Sumit J Darak
Algorithms to Architectures Lab
Associate Professor, ECE, IIIT Delhi
http://faculty.iiitd.ac.in/~sumit/

Digital Circuits Revisited!

Combinational vs Sequential Circuits



FSMs


S0
0
S1
2
S2
4
S0 = 00
S1 = 01
S2 = 10
en
Comb
ckt
NS
?
Mem
2-FF
PS
2
o/p
00
01
10
11
0
2
4
0

Counter: FSMs

- For a counter with counting sequence of {0,1,2,3,4,5,6,0,1,2,3,4,5,6,0,1,....}, how many flip-flops are needed?

- For a counter with counting sequence of {0,3,5,0,3,5,0,3,....}, how many flip-flops are needed?

Verilog

Verilog

- One of the two major HDLs used by hardware designers in Industry and Academia (Another is VHDL)
- C- based Syntax, easy to master and intensively used by Indian VLSI Industry
- 1983: Introduced by Gateway Design System
- Invented as simulation language. Synthesis was an afterthought
- 1987: Verilog synthesizer by Synopsis
- 1989: Cadence acquired Gateway Design System and became the language owner

Verilog

- Around the same time (1981-1988), the US Department of Defence developed VHDL (VHSIC HDL). Because it was in the public domain it began to grow in popularity.
- Afraid of losing market share, Cadence opened Verilog to the public in 1990.
- 1995: Became IEEE Standard 1364
- 2001 and 2005: New and improved version of Verilog (made life much easier)
- Latest Verilog version is “System Verilog” .
- Ongoing efforts for automating the mapping of the code written in high level language (C, System C, Python) to Verilog/VHDL

Few words of wisdom

- One of the common challenge 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

- Verilog looks like C, but it describes hardware
- First understand the circuit and specifications you want then figure out how to code it in Verilog.
- A large part of ELD (before mid-sem) is knowing how to write Verilog that gets you the desired circuit.
- If you do one of these activities without the other, you will not enjoy the process of algorithms to architecture mapping.
- These two activities will merge at some point for you

Verilog (Three Concepts)

- Difference between Register and Wire (Next two lectures)
- Efficient Behavioral modelling
- Difference between blocking and non-blocking assignments

Verilog:

- Verilog HDL is a case-sensitive language
- All keywords are in lowercase (assign, for, always, fork, if, else, input, output…)
- Statements are terminated by a semicolon (;)
- Two data types: Net (wire) and variable (Reg, Integer, real, time, realtime)
- Primitive Logic Gates and Switch-Level Gates, are built-in (Rarely used in ELD)
- Single line comments begin with the “// “and end with a carriage return.
  // This is one line comment
- Multi Line comments begin with the “/*” and end with the “*/”
  /* This is a multiple
  lines
  comments */

Verilog: Identifiers

- Identifiers are names given to an object, such as a register or a function or a module, so that it can be referenced from other places in a description
- Identifiers must begin with an alphabetic character or the underscore character
- Identifier cannot start with a number or dollar sign
- Identifiers may contain alphabetic characters, numeric characters, underscore, and dollar sign
- Identifiers can be up to 1024 characters long
- Identifiers examples:
  wire outAdd ; // wire is a keyword, outAdd is an identifier
  reg sum ; // reg is a keyword, sum is an identifier

Verilog: Module

- Verilog describes a digital system as a set of modules
- Each module has an **interface** and **contents** description
- Modules communicate externally with input, output and bi-directional ports (inout)
- Verilog modules consist of a list of statements declaring relationships between a module and its environment, and between signals within a module

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
```

Verilog: Module (Examples)

```
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

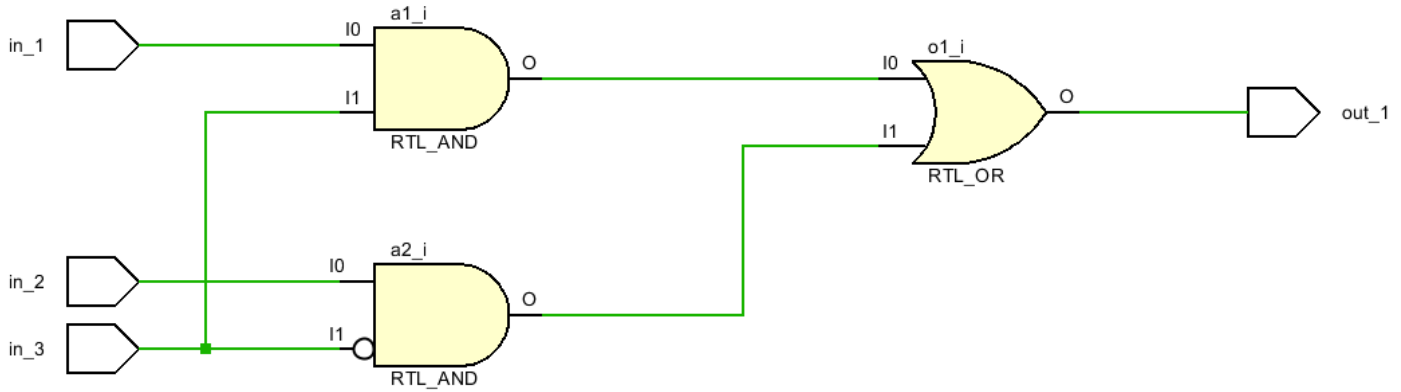
```
module exam_1 (o1,in1,in2,in3); //module
    output o1;
    input in1,in2,in3;
        and a1 (Q, in1, in3); //and instantiation
        not n1 (P,in3);          // not instantiation
        and a2 (R, in2, P);  // and instantiation
        or    o1 (o1, Q, R); // or instantiation
Endmodule
```

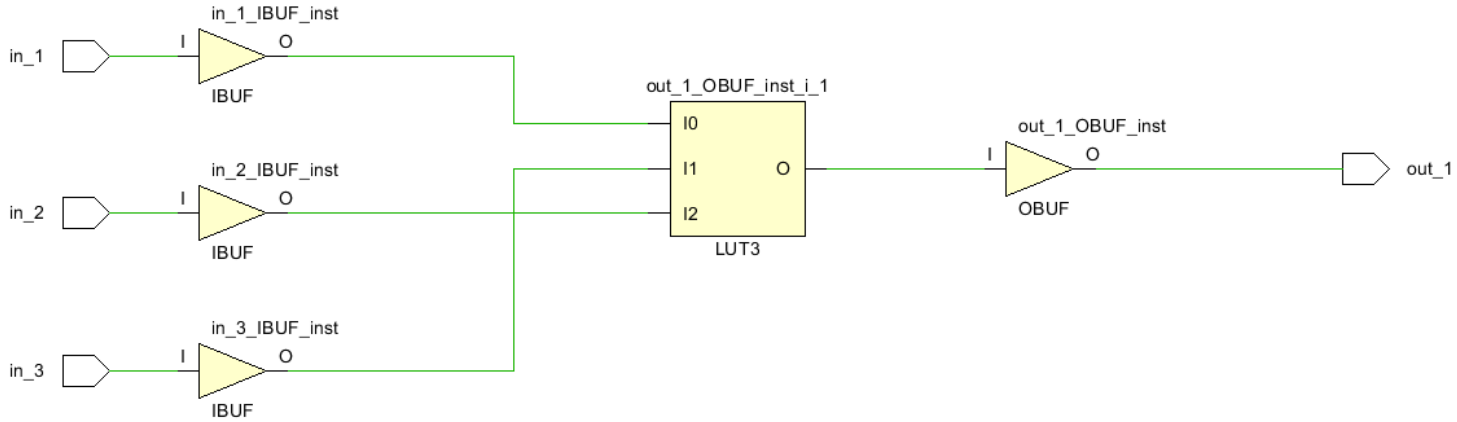
Verilog

```
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
```

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
```