Digital System Design ELEC373/473

User Defined Primitives (UDP)

Module Styles

- Modules can be specified in different ways
  1) **Structural**: connect primitives and modules
  2) **RTL** (Register Transfer Level) :
     - use continuous assignments
  3) **Behavioral**: use **initial** and **always** blocks
     - Note that “initial” is primarily for simulation rather than for synthesis.
- A single module can use more than one method.

Truth Table Models of Combinational and Sequential Logic with Verilog

- Verilog has built-in primitives correspond to simple combinational logic gates.
- Verilog has a mechanism for building user-defined primitives (UDPs).
- Verilog uses truth tables to describe sequential behaviour and/or complex combinational circuits.
- UDPs are simulated faster and require less silicon area than modules.
- A UDP has only a single, scalar (one-bit), output port.
- The input ports of a UDP must be scalars.
- UDPs are instantiated just like built-in primitives.

UDP for a Two-input Multiplexer

```
primitive mux_prim (mux_out, select, a ,b);
    output  mux_out;
    input   select, a ,b;

    table
    // select   a   b   :   mux_out
        0       0   0   :   0;  // Order of the table columns follows
        0       0   1   :   0;  // port order of inputs.
        0       0   x   :   0;  // One output, multiple inputs, no inout
        0       1   0   :   1;  // Only 0,1, x on input and output
        0       1   1   :   1;  // A z input in simulation is treated as x
        0       1   x   :   1;  // Last column is the output
    // select   a   b   :   mux_out
        1       0   0   :   0;
        1       1   0   :   0;
        1       x   0   :   0;
        1       0   1   :   1;
        1       1   1   :   1;
        1       x   1   :   1;

        x       0   0   :   0;  // reduce pessimism
        x       1   1   :   1;
    endtable    //Combinations not exactly defined will drive 'x' under simulation
endprimitive
```

Shorthand UDP for a Two-input Multiplexer

```
table
// select    a    b    :    mux_out
     0         0    ?    :    0;   // ? = 0, 1, x
     0         1    ?    :    1;
     1         ?    0    :    0;
     1         ?    1    :    1;
     ?         0    0    :    0;
     ?         1    1    :    1;
endtable
```


The ? Shorthand notation substitutes 0, 1, and x in the table.

UDPs for Level-Sensitive Sequential Logic

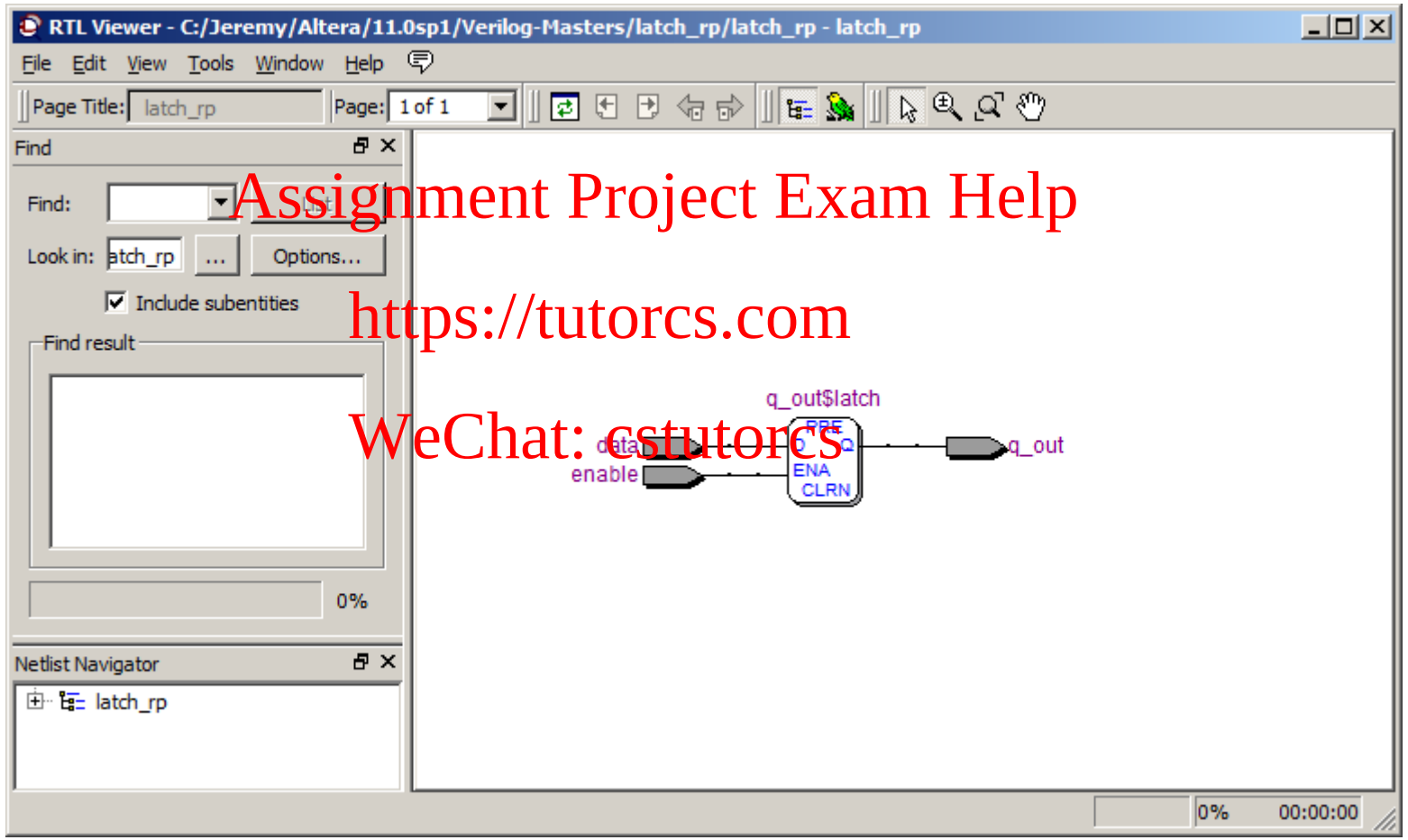
- Level-sensitive behaviours are conditioned by an enabling input signal.
- Level-sensitive devices respond to any change of an input while the enabling input is high.
- The truth table will have two output columns for present state and next state.
- Next state is the state caused by the present inputs when enable is high.
- The output of a sequential UDP must be declared to have type **reg** because it must keep its value when inputs change while the enabling signal is low.

Transparent Latch

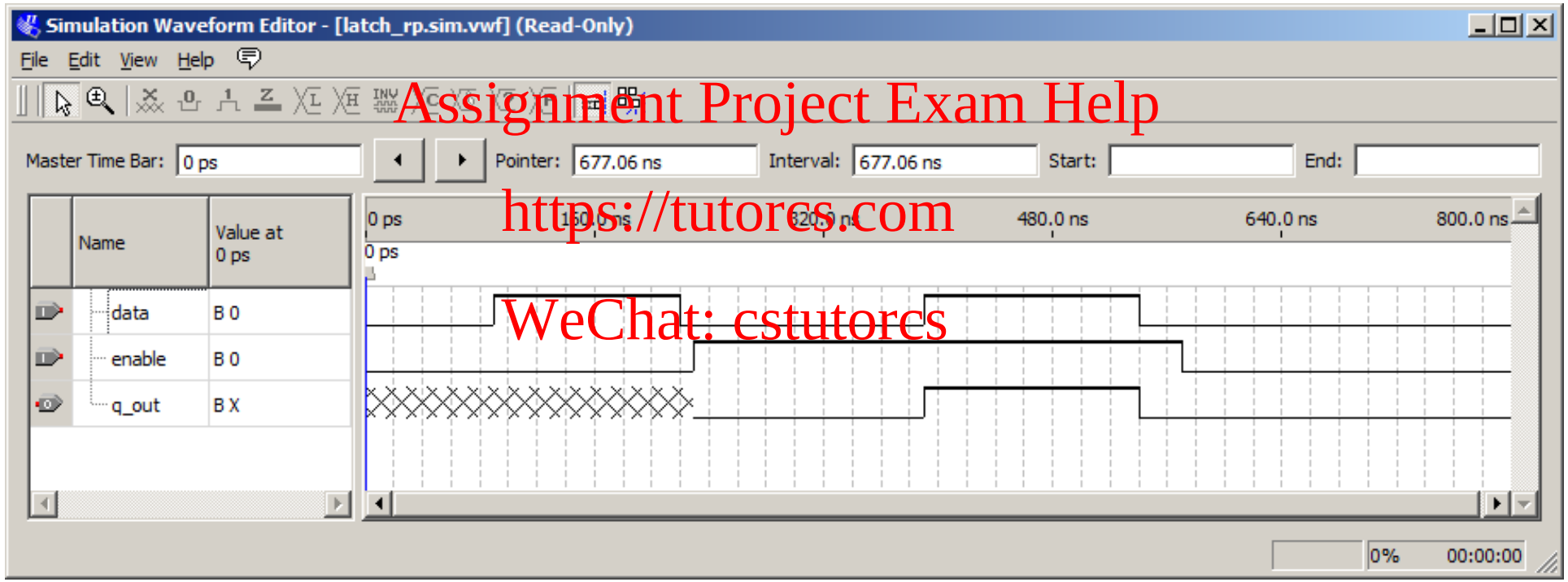
```
primitive latch_rp( q_out, enable, data);
    output  q_out;
    input   enable, data;
    reg     q_out;

    table
    //  enable  data        state       q_out/next_ststae
        1       1       :   ?       :       1;
        1       0       :   ?       :       0;
        0       ?       :   ?       :       -;
        // "-" denotes no change of the output
        x       0       :   0       :       -;
        x       1       :   1       :       -;
    endtable
endprimitive
```

Synthesised Design



Simulation of Transparent Latch



UDP for Edge-Sensitive Behaviours

- A truth table describing edge-sensitive behaviour will be activated whenever an input has an event.
- The output changes depends on whether a synchronising input has made an appropriate transition.
- The synchronising input change is denoted by two bits in parentheses, e.g. (01) denotes a rising edge.
- The synchronising input appears in the first column.
- A truth table can include both level-sensitive behaviour and edge-sensitive behaviour. This is used to model synchronous behaviour with asynchronous set and reset conditions.
- A level-sensitive behaviour should precede the edge-sensitive behaviour in the table.

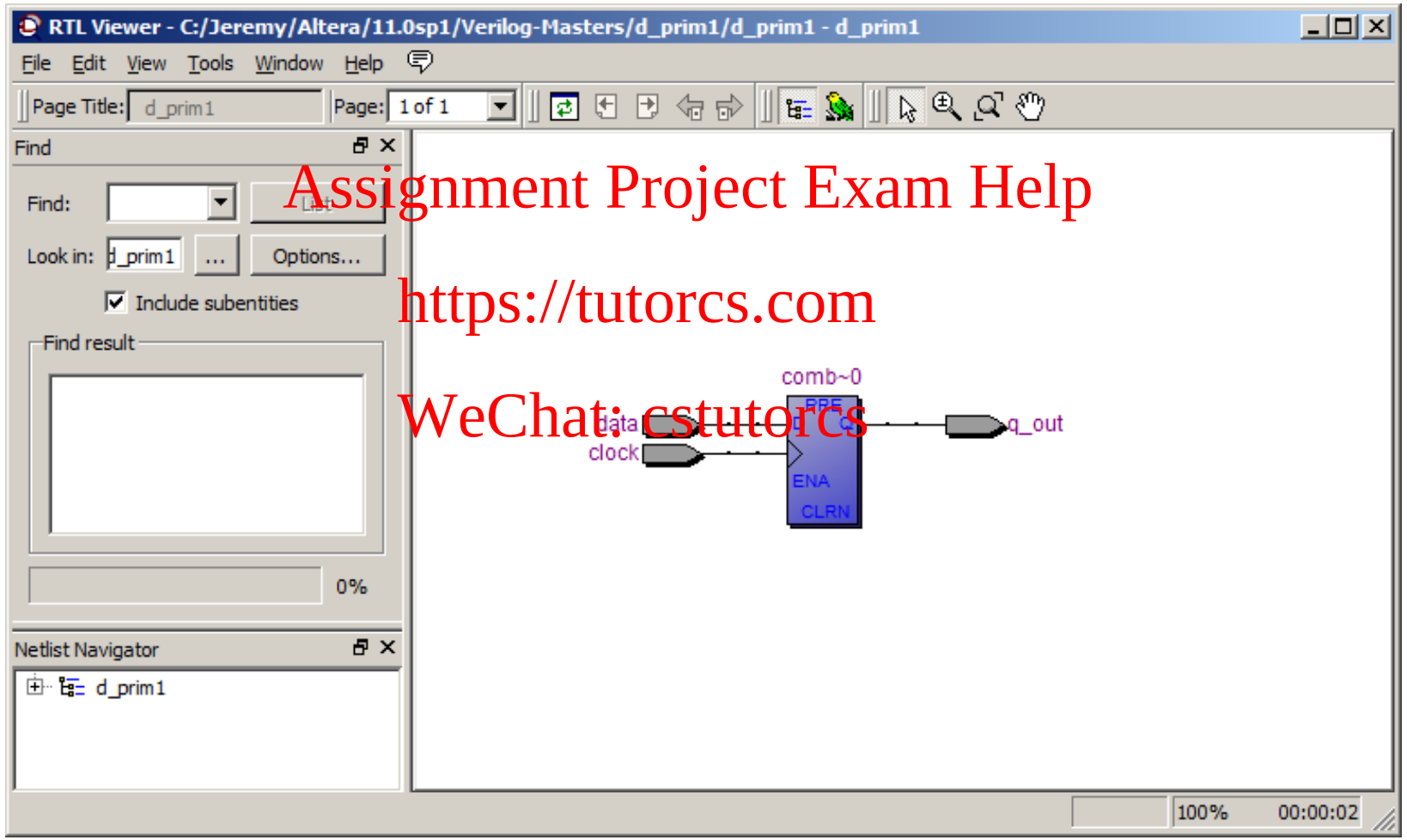
Truth-table model of a D-type FF

```
primitive d_prim1 (q_out, clock, data);
    output  q_out;
    input   clock, data;
    reg     q_out;

    table
    //  clk     data        state       q_out/next_state
        (01)    0       :   ?       :       0;  // rising clock edge
        (01)    1       :   ?       :       1;
        (0x)    1       :   1       :       1;
        (0x)    0       :   0       :       0;
        (?0)    ?       :   ?       :       -;  // falling or steady
                                                // clock edge
        ?       (??)    :   ?       :       -;  // steady clock, ignore
                                                // data transition

    endtable
endprimitive
```

Synthesised design



Functional and Timing Simulation of a UDP D-type Flip-Flop