EE671: VLSI Design
Spring 2024/25

Laxmeesha Somappa

Department of Electrical Engineering

IIT Bombay

laxmeesha@ee.iitb.ac.in

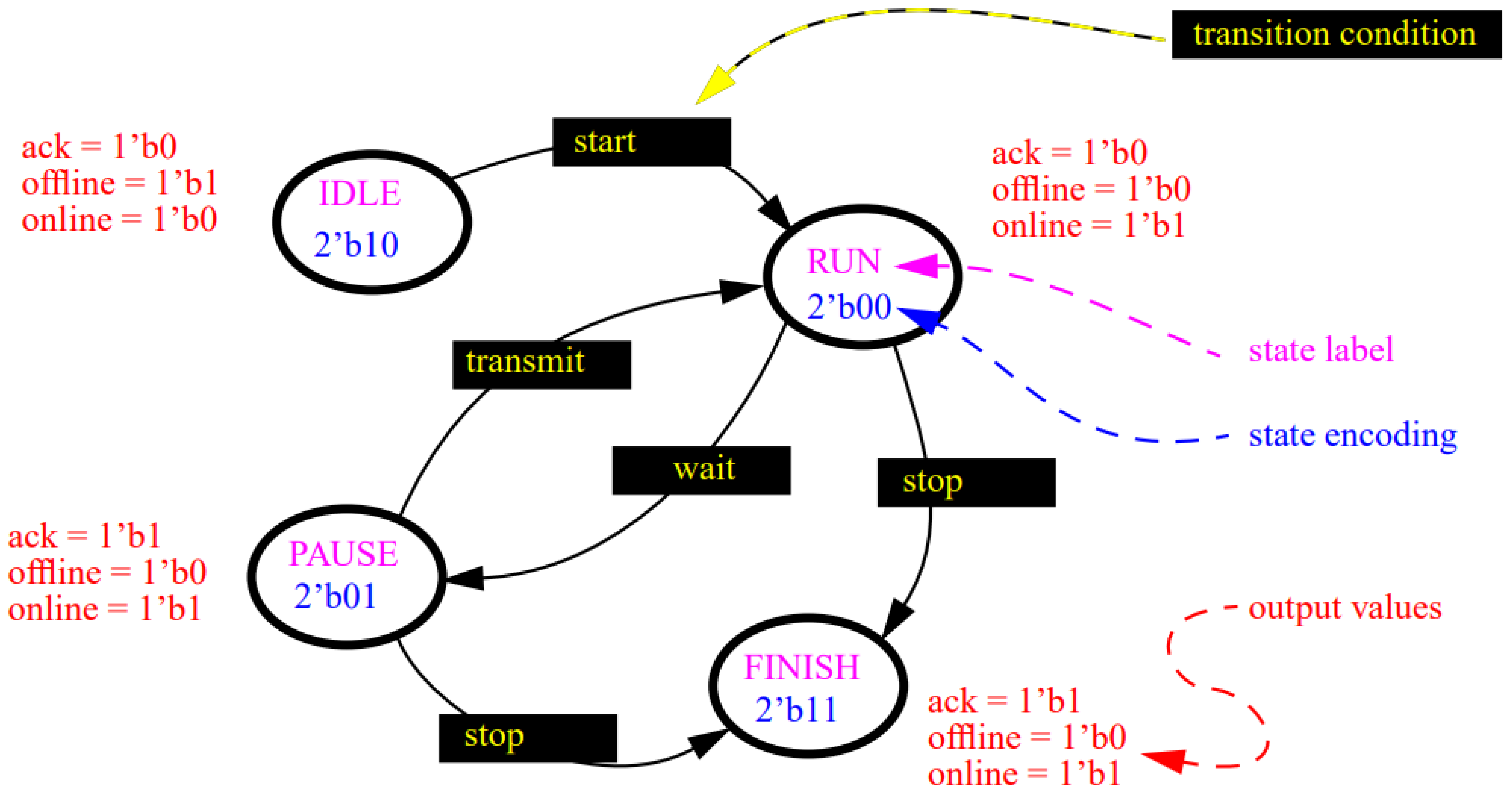
Lecture – 20
Digital Synthesis

Logic Synthesis: State Machine Guidelines

- Draw a state diagram.
- Label the states.
- Allocate state encoding.
- Label the transition conditions.
- Label the output values.
- Use parameters for the state variables.
- Use two procedures (one clocked for the state register and one combinational for the next state logic and the output decode logic).
- Use a case statement in the combinational procedure.
- Have a reset strategy (asynchronous or synchronous).
- Use default assignments and then corner cases.
- Keep state machine code separate from other code (i.e. don't mix other logic in with the state machine clocked and combinational procedures).

LOGIC SYNTHESIS: STATE MACHINE GUIDELINES

State Diagram



Logic Synthesis: State Machine Guidelines

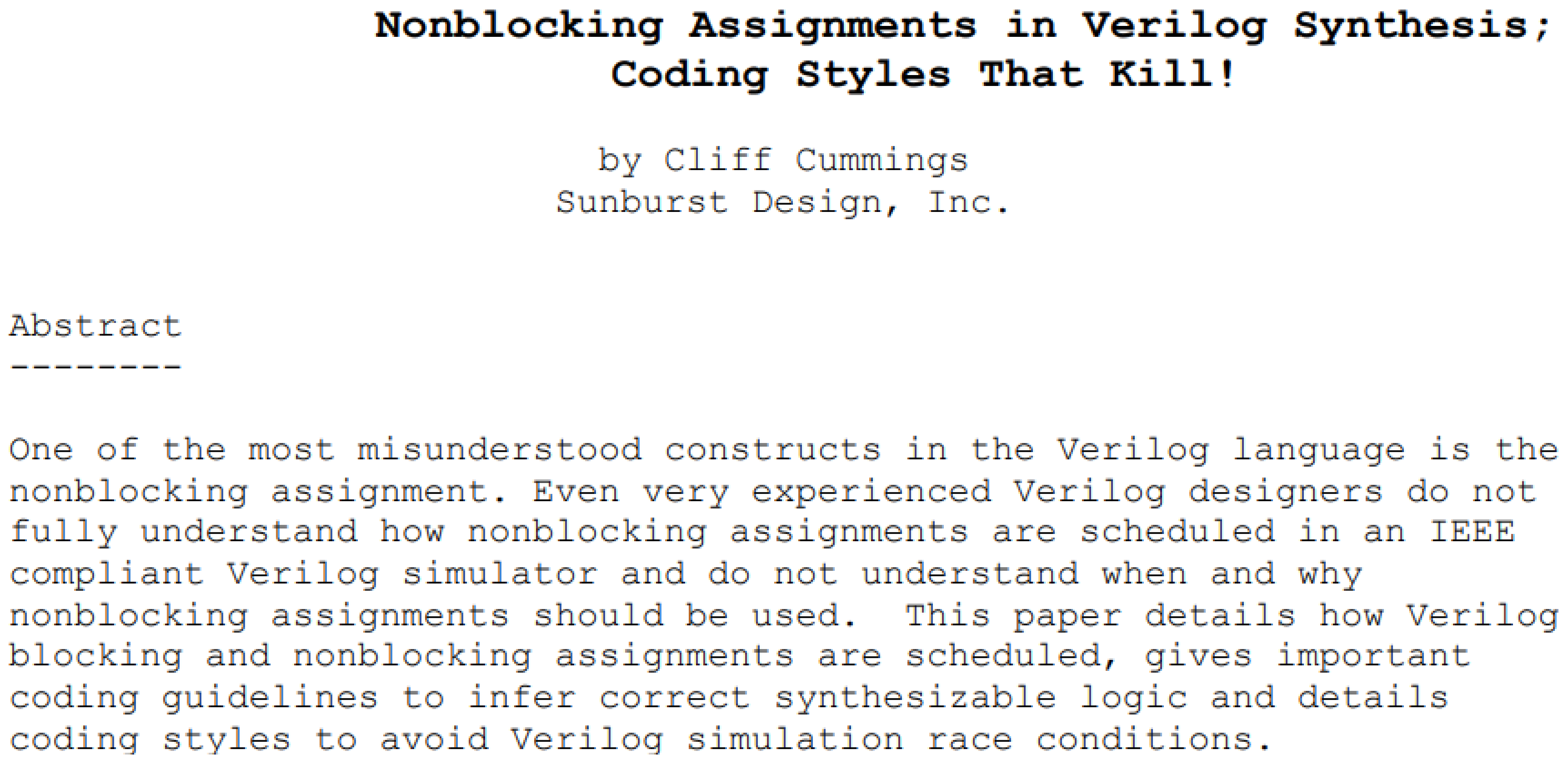
```
module state_machine (clock,reset,start,transmit,wait,stop,ack,offline,online);
// parameter declarations
parameter pIDLE   = 2'b10; // state labels and state encoding
parameter pRUN    = 2'b00;
parameter pPAUSE  = 2'b01;
parameter pFINISH = 2'b11;

// IO declaration section
input clock;
input reset;
input start, transmit, wait, stop;
output ack, offline, online;
// interal variables declaration section
reg [1:0] state, next_state;
reg ack, offline, online;

// clocked procedure with synchronous reset
always @ (posedge clock)
if (reset) // reset strategy
   state <= pIDLE;
else
   state <= next_state;
```

Logic Synthesis: State Machine Guidelines

```
// combinational procedure with case statement and output logic
always @ (start or transmit or stop or wait or state)
   begin
   next_state = state; // default assignment to state and output variables
   ack     = 1’b0;
   offline = 1’b0;
   online  = 1’b1;
   case (state)
      pIDLE:
         begin
         offline = 1’b1;
         online  = 1’b0;
         if (start)
            next_state = pRUN;
         end
      pRUN:
         begin
         if (wait)
            next_state = pPAUSE;
         if (stop)
            next_state = pFINISH; // this has priority over the wait transition
         end
      pPAUSE:
         begin
         ack     = 1’b1;
         if (transmit)
            next_state = pRUN;
         if (stop)
            next_state = pFINISH;
         end
      pFINISH:
         ack = 1’b1;
   endcase
   end

endmodule
```

LOGIC SYNTHESIS: BLOCKING VS NON-BLOCKING

Refer to “Nonblocking Assignments in Verilog Synthesis; Coding Styles That Kill!” by Cliff Cummings – uploaded on Moodle



Nonblocking Assignments in Verilog Synthesis; Coding Styles That Kill!

by Cliff Cummings
Sunburst Design, Inc.

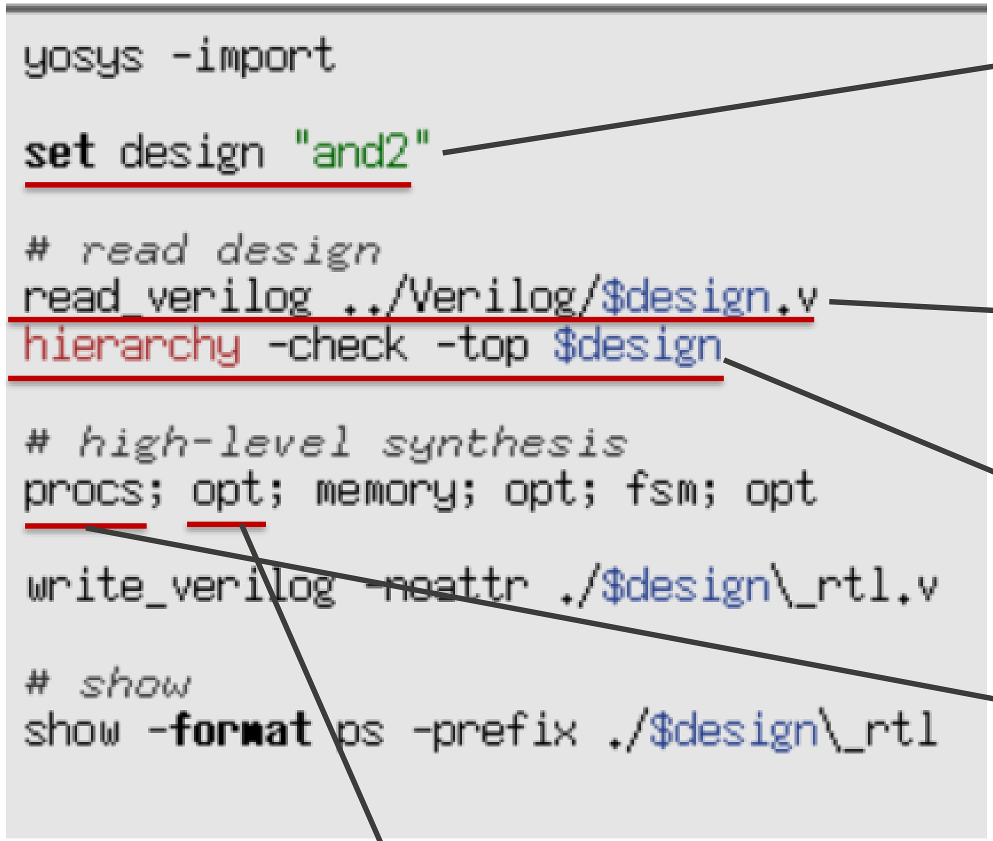
```
Abstract
--------

One of the most misunderstood constructs in the Verilog language is the
nonblocking assignment. Even very experienced Verilog designers do not
fully understand how nonblocking assignments are scheduled in an IEEE
compliant Verilog simulator and do not understand when and why
nonblocking assignments should be used.  This paper details how Verilog
blocking and nonblocking assignments are scheduled, gives important
coding guidelines to infer correct synthesizable logic and details
coding styles to avoid Verilog simulation race conditions.
```

LOGIC SYNTHESIS

- Popular logic synthesis tools:
  - Synopsys: Design Compiler (DC)
  - Cadence: Genus
  - Open Source: Yosys (will be used in this course)

- One of the widely used scripting language in the industry:
  - Tcl: Tool Command Language
- Synopsys DC, Cadence Genus and Yosys all support tcl based scripting flow

LOGIC SYNTHESIS: YOSYS EXAMPLE TCL SCRIPT



```
yosys -import

set design "and2"

# read design
read_verilog ../Verilog/$design.v
hierarchy -check -top $design

# high-level synthesis
procs; opt; memory; opt; fsm; opt

write_verilog -noattr ./$design\_rtl.v

# show
show -format ps -prefix ./$design\_rtl
```

Defining a variable called design and assigning it a string value "and2"

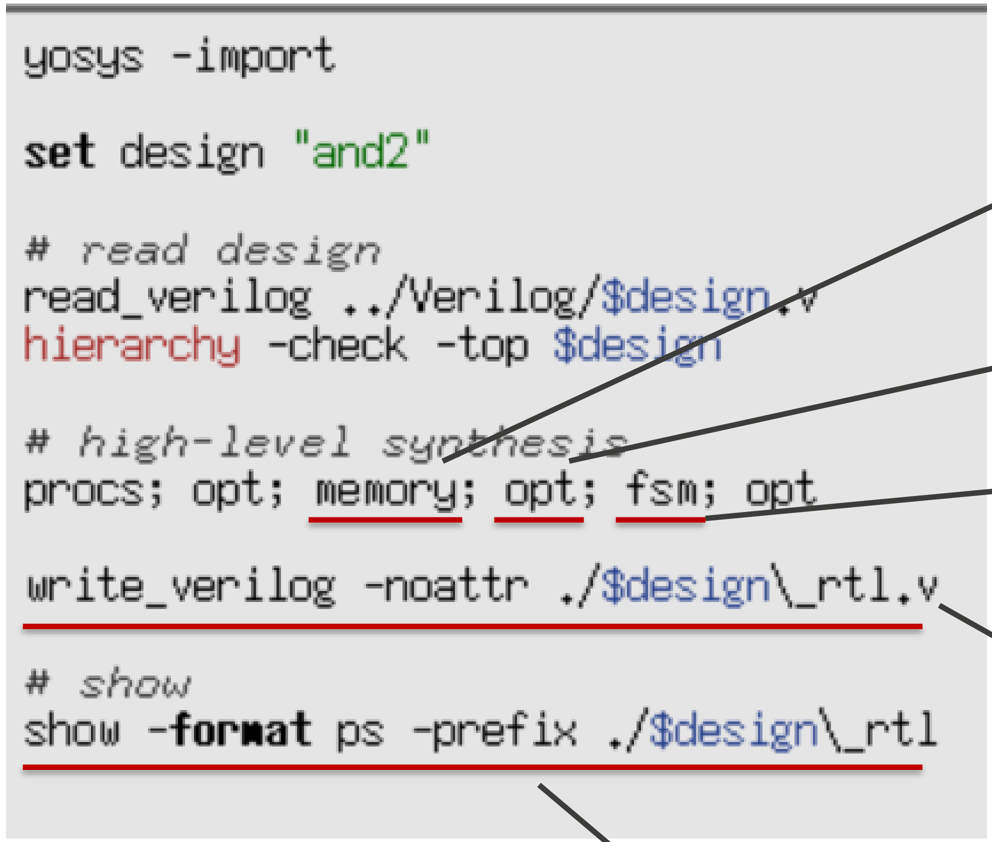
read_verilog <path_to_file>
-- to read-in the design file

Elaborate the design hierarchy with top cell name

Convert processes (internal behavioural structures) into multiplexers and registers

First level of basic optimizations & clean-up

Logic Synthesis: Yosys example TCL script



```
yosys -import

set design "and2"

# read design
read_verilog ../Verilog/$design.v
hierarchy -check -top $design

# high-level synthesis
procs; opt; memory; opt; fsm; opt

write_verilog -noattr ./$design\_rtl.v

# show
show -format ps -prefix ./$design\_rtl
```

Analyze memories and create circuits to implement them (using FFs or other custom IPs)

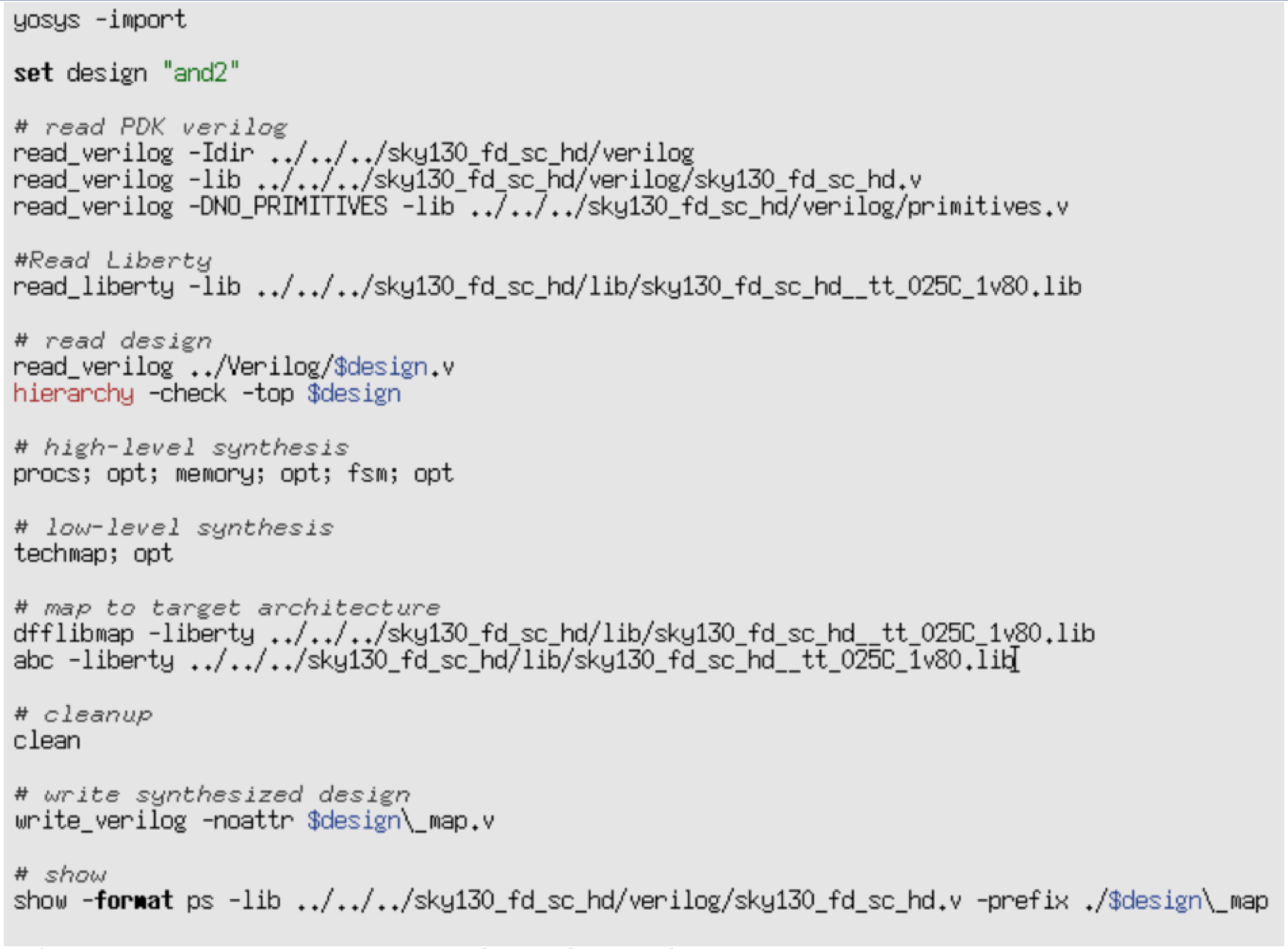
Another round of basic optimization & cleanup

Analyze and optimize finite state machines

Write the output RTL file. This is not yet technology mapped (Std. cell library independent)

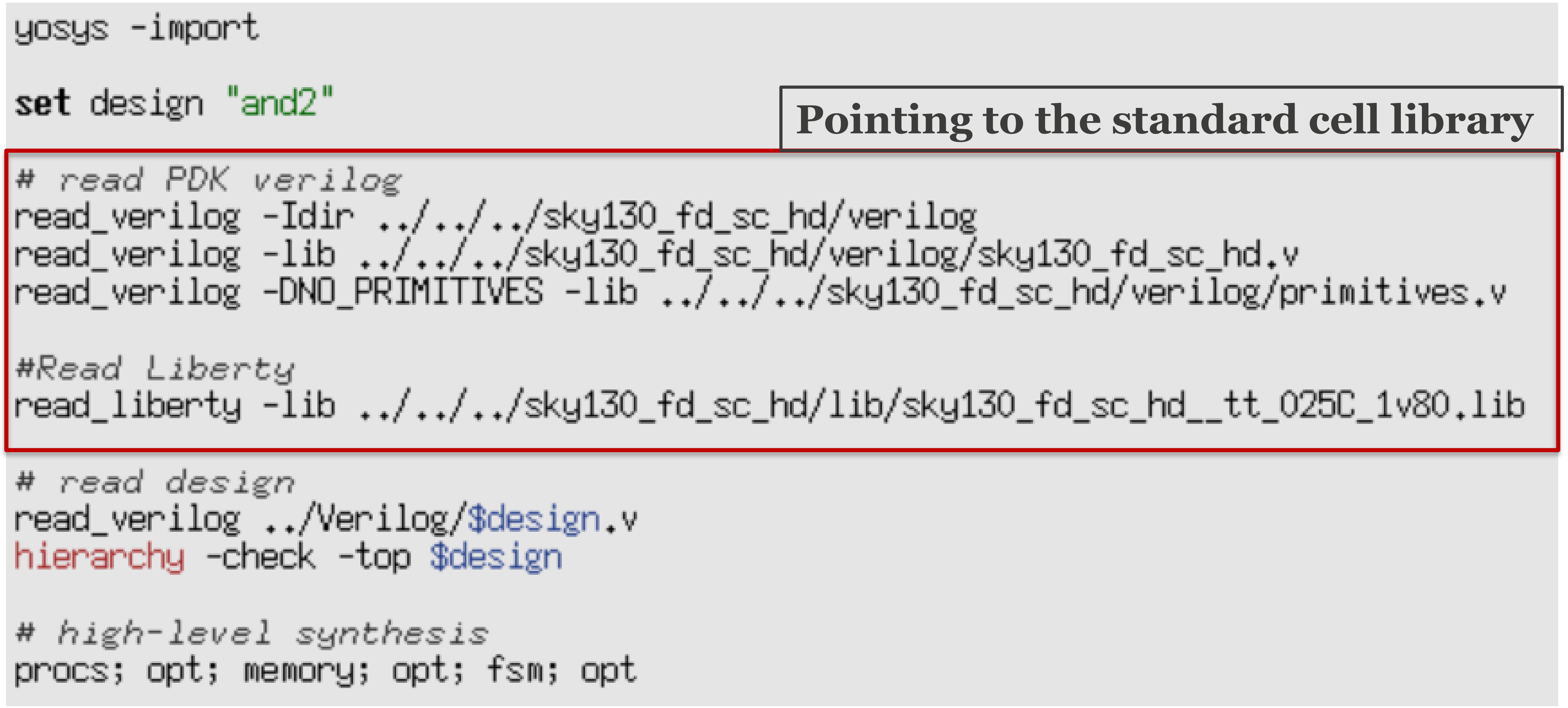
Write a pictorial representation of the logic (can be .ps, .png formats) for user to view

Logic Synthesis: Yosys TCL script with std cell lib



```
yosys -import

set design "and2"

# read PDK verilog
read_verilog -Idir ../../../sky130_fd_sc_hd/verilog
read_verilog -lib ../../../sky130_fd_sc_hd/verilog/sky130_fd_sc_hd.v
read_verilog -DNO_PRIMITIVES -lib ../../../sky130_fd_sc_hd/verilog/primitives.v

#Read Liberty
read_liberty -lib ../../../sky130_fd_sc_hd/lib/sky130_fd_sc_hd__tt_025C_1v80.lib

# read design
read_verilog ../Verilog/$design.v
hierarchy -check -top $design

# high-level synthesis
procs; opt; memory; opt; fsm; opt

# low-level synthesis
techmap; opt

# map to target architecture
dfflibmap -liberty ../../../sky130_fd_sc_hd/lib/sky130_fd_sc_hd__tt_025C_1v80.lib
abc -liberty ../../../sky130_fd_sc_hd/lib/sky130_fd_sc_hd__tt_025C_1v80.lib

# cleanup
clean

# write synthesized design
write_verilog -noattr $design\_map.v

# show
show -format ps -lib ../../../sky130_fd_sc_hd/verilog/sky130_fd_sc_hd.v -prefix ./$design\_map
```

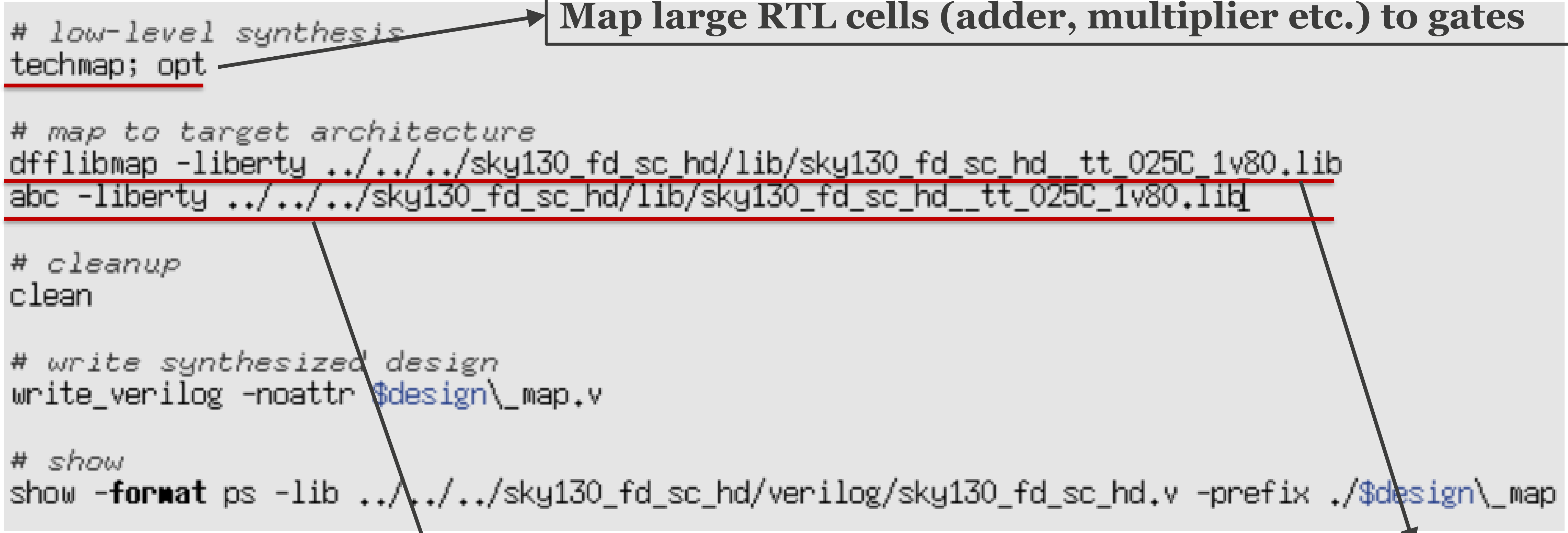
Logic Synthesis: Yosys TCL script with std cell lib



```
yosys -import

set design "and2"

# read PDK verilog
read_verilog -Idir ../../../sky130_fd_sc_hd/verilog
read_verilog -lib ../../../sky130_fd_sc_hd/verilog/sky130_fd_sc_hd.v
read_verilog -DNO_PRIMITIVES -lib ../../../sky130_fd_sc_hd/verilog/primitives.v

#Read Liberty
read_liberty -lib ../../../sky130_fd_sc_hd/lib/sky130_fd_sc_hd__tt_025C_1v80.lib

# read design
read_verilog ../Verilog/$design.v
hierarchy -check -top $design

# high-level synthesis
procs; opt; memory; opt; fsm; opt
```

Pointing to the standard cell library

LOGIC SYNTHESIS: YOSYS TCL SCRIPT WITH STD CELL LIB



```
# low-level synthesis
techmap; opt

# map to target architecture
dfflibmap -liberty ../../../sky130_fd_sc_hd/lib/sky130_fd_sc_hd__tt_025C_1v80.lib
abc -liberty ../../../sky130_fd_sc_hd/lib/sky130_fd_sc_hd__tt_025C_1v80.lib

# cleanup
clean

# write synthesized design
write_verilog -noattr $design\_map.v

# show
show -format ps -lib ../../../sky130_fd_sc_hd/verilog/sky130_fd_sc_hd.v -prefix ./$design\_map
```

Map large RTL cells (adder, multiplier etc.) to gates

Map all registers in the RTL to the Std. Cell Library

Map all logics in the RTL to the Std. Cell Library

Logic Synthesis: Yosys Manual & Demo

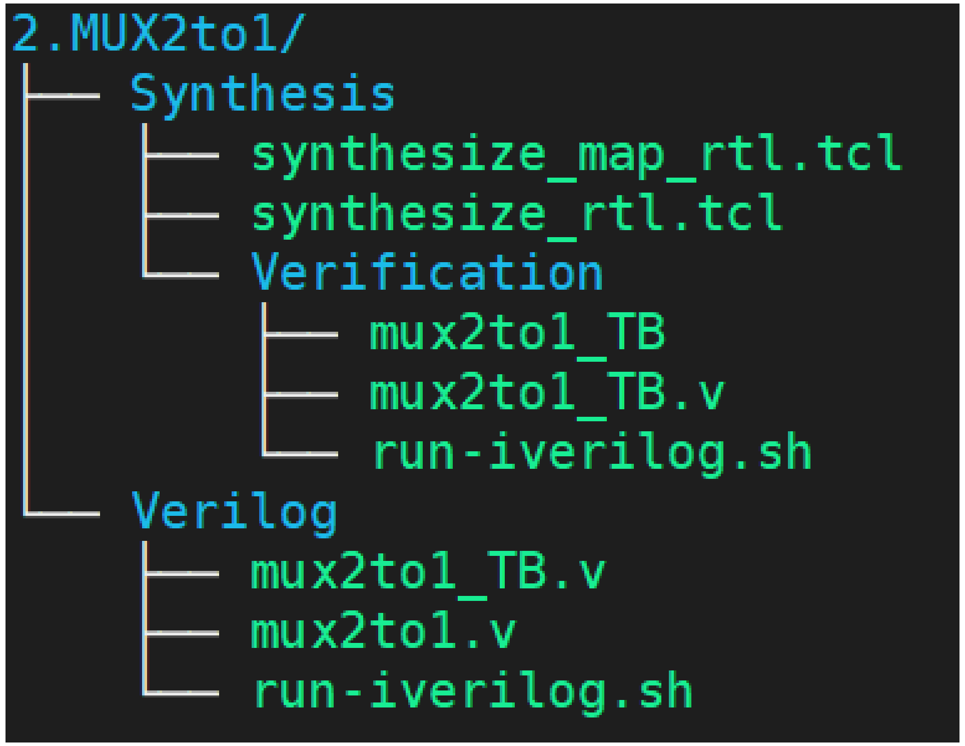
- If you want to read up more on Yosys:
  https://yosyshq.net/yosys/files/yosys_manual.pdf
- You can install Yosys on your machines if you are interested
  https://yosyshq.net/yosys/download.html
- Class Demo on few circuits – Yosys Examples
- You will see later that "Openlane" which is an open source RTL to GDSII flow wraps all open source tools including Yosys with Python wrappers to seamlessly integrate multiple tools under one banner
  - You don't need to learn Tcl scripting – you will use Python to run Yosys later
  - The Tcl scripts provided here are very specific to Yosys and will not work on Synopsys DC and Genus.

Logic Synthesis: Example

- Sequence of steps:
  - Write Verilog Code
  - Write Testbench to test the Verilog code ($dumpvars will dump vcd waveform)
  - Simulate code and tb in iverilog (Icarus Verilog)
  - Open waveform (.vcd) in gtkwave and check waveforms

  - Synthesize Verilog code (Yosys) and check output RTL file (without map to std. cell library – just for your reference)
  - Synthesize Verilog code (Yosys) and check output RTL file (with map to std. cell library – this is the main output RTL)
  - Simulate the output RTL with earlier tb using Iverilog – to check it is functionally still the same

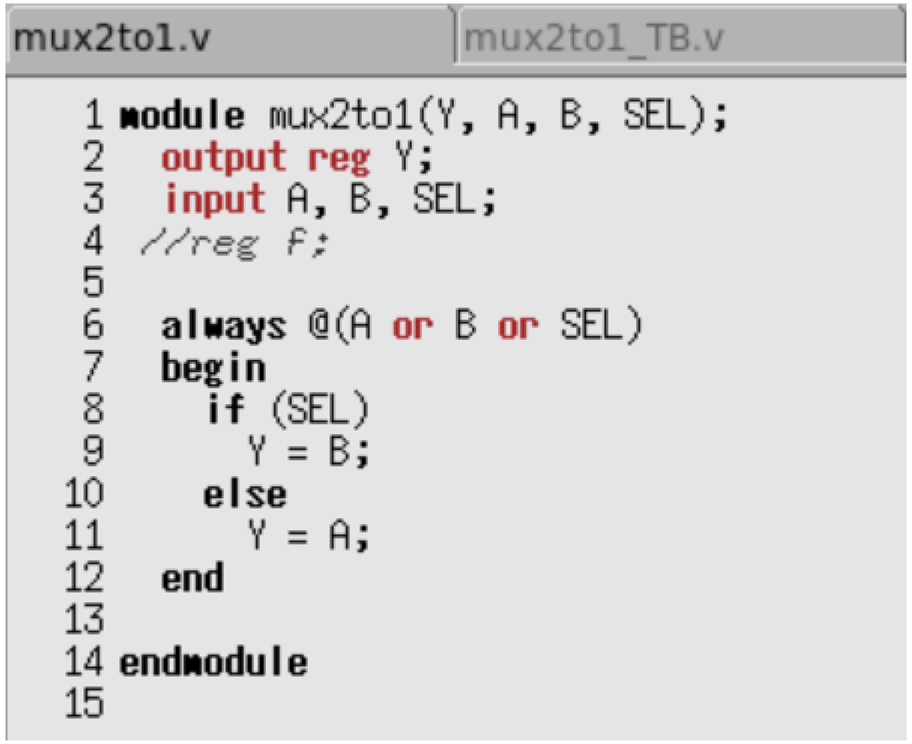
LOGIC SYNTHESIS: EXAMPLE DIRECTORY STRCUTURE



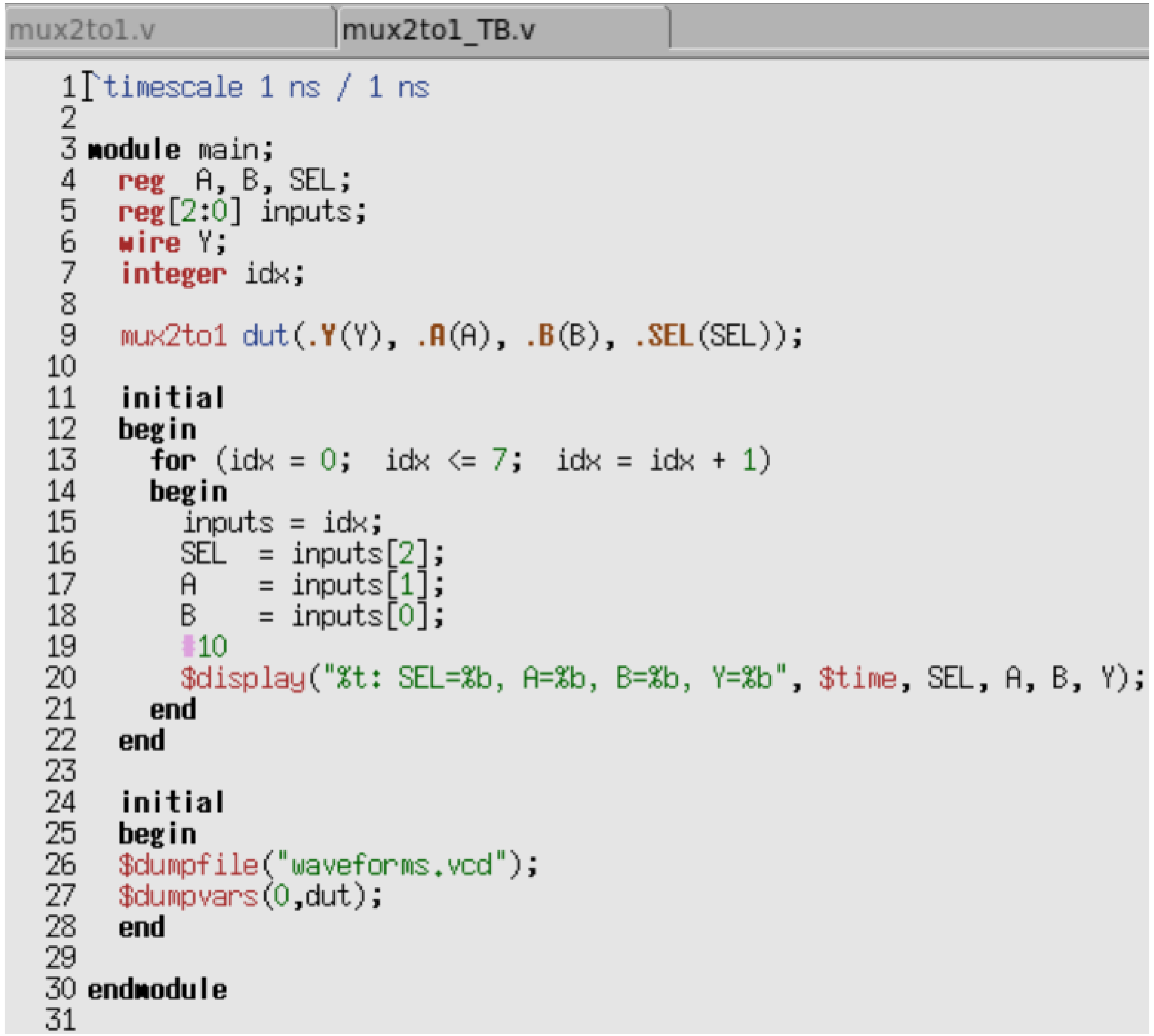
```
2.MUX2to1/
├── Synthesis
│   ├── synthesize_map_rtl.tcl
│   ├── synthesize_rtl.tcl
│   └── Verification
│       ├── mux2to1_TB
│       ├── mux2to1_TB.v
│       └── run-iverilog.sh
└── Verilog
    ├── mux2to1_TB.v
    ├── mux2to1.v
    └── run-iverilog.sh
```

- **2.Mux2to1 is the main folder**
- **In the Verilog folder, we have the main Verilog file, tb and run command for iverilog**
- **Inside Synthesis folder, we have tcl scripts for Yosys**
- **Inside Synthesis folder, we have Verification folder to re-run simulation with RTL netlist**

Logic Synthesis: Example Mux

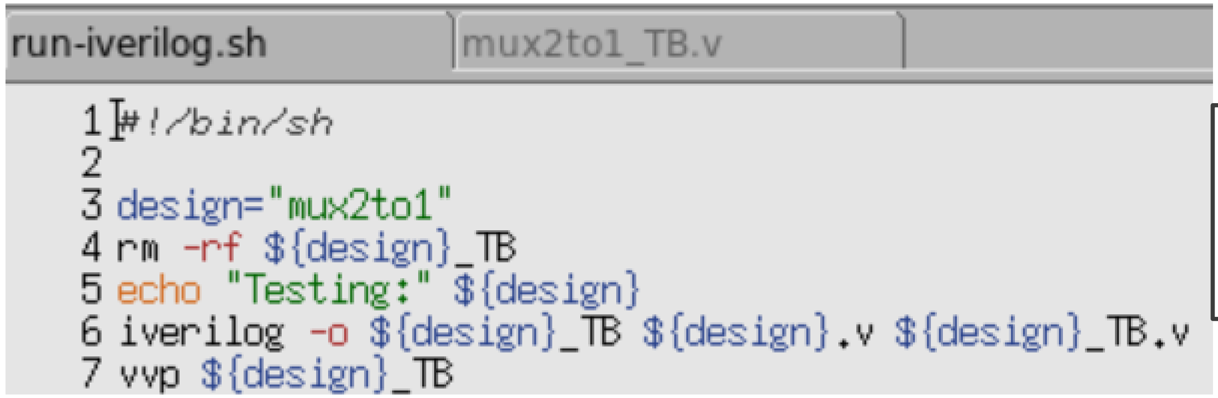


```verilog
module mux2to1(Y, A, B, SEL);
  output reg Y;
  input A, B, SEL;
 //reg f;

  always @(A or B or SEL)
  begin
    if (SEL)
      Y = B;
    else
      Y = A;
  end

endmodule

```



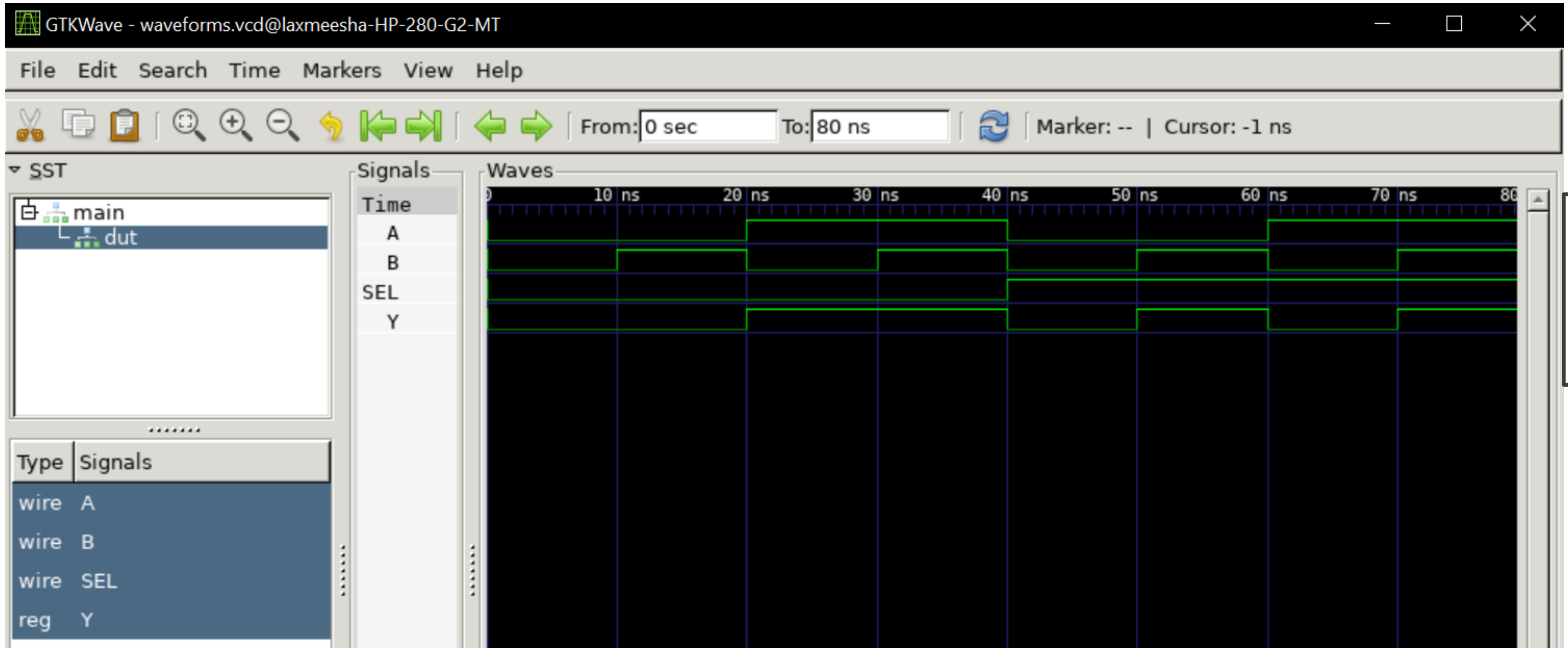
```verilog
`timescale 1 ns / 1 ns

module main;
  reg  A, B, SEL;
  reg[2:0] inputs;
  wire Y;
  integer idx;

  mux2to1 dut(.Y(Y), .A(A), .B(B), .SEL(SEL));

  initial
  begin
    for (idx = 0;  idx <= 7;  idx = idx + 1)
    begin
      inputs = idx;
      SEL   = inputs[2];
      A     = inputs[1];
      B     = inputs[0];
      #10
      $display("%t: SEL=%b, A=%b, B=%b, Y=%b", $time, SEL, A, B, Y);
    end
  end

  initial
  begin
  $dumpfile("waveforms.vcd");
  $dumpvars(0,dut);
  end

endmodule

```

Logic Synthesis: Example Mux sims (Iverilog)



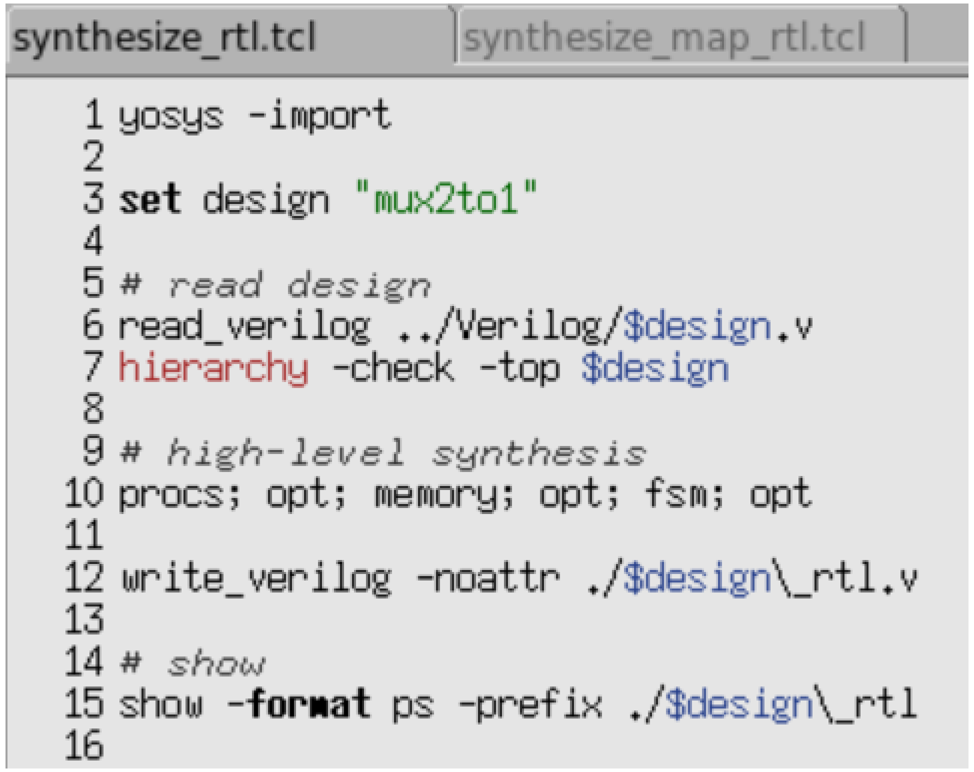
```
#!/bin/sh

design="mux2to1"
rm -rf ${design}_TB
echo "Testing:" ${design}
iverilog -o ${design}_TB ${design}.v ${design}_TB.v
vvp ${design}_TB
```

Shell script to run iverilog: on the terminal execute: ./run-iverilog.sh



Open waveforms using GTKWave. On the terminal gtkwave <path_vcd_file>

LOGIC SYNTHESIS: EXAMPLE (YOSYS)



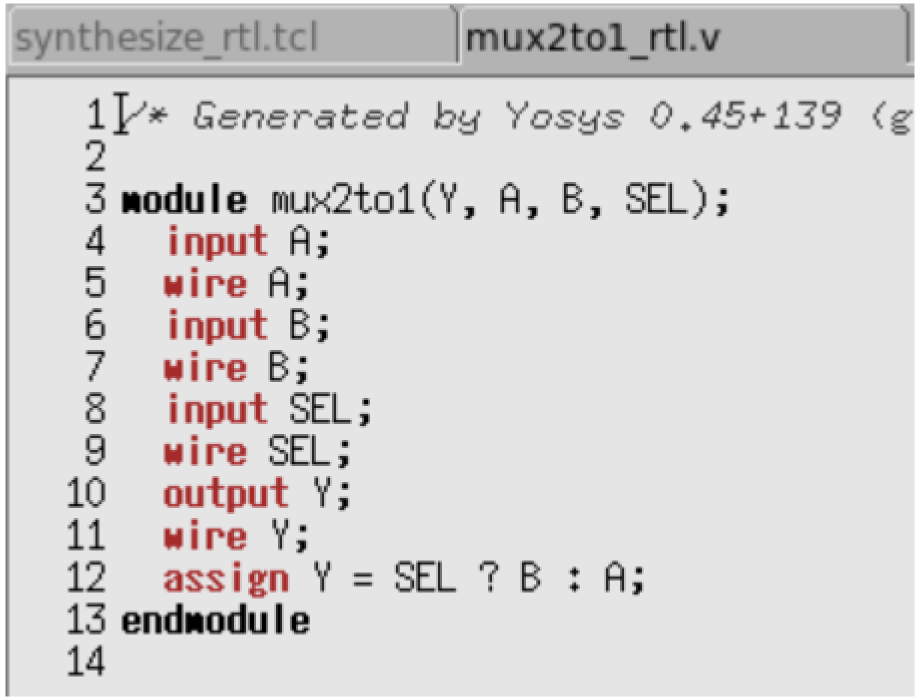
synthesize_rtl.tcl | synthesize_map_rtl.tcl

```
yosys -import

set design "mux2to1"

# read design
read_verilog ../Verilog/$design.v
hierarchy -check -top $design

# high-level synthesis
procs; opt; memory; opt; fsm; opt

write_verilog -noattr ./$design\_rtl.v

# show
show -format ps -prefix ./$design\_rtl

```

Call Yosys on the terminal with the tcl script

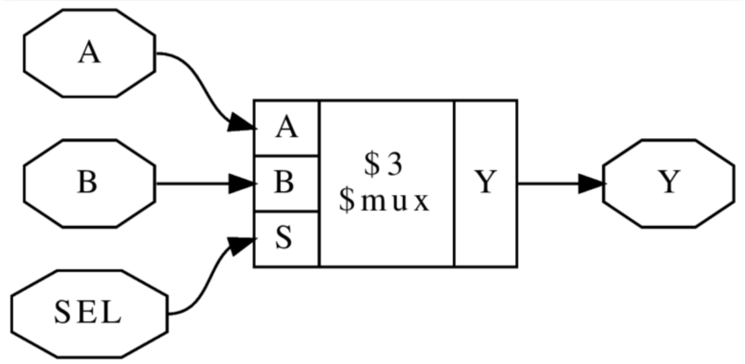
```
yosys synthesize_rtl.tcl
```

Yosys outputs (.v and .ps)



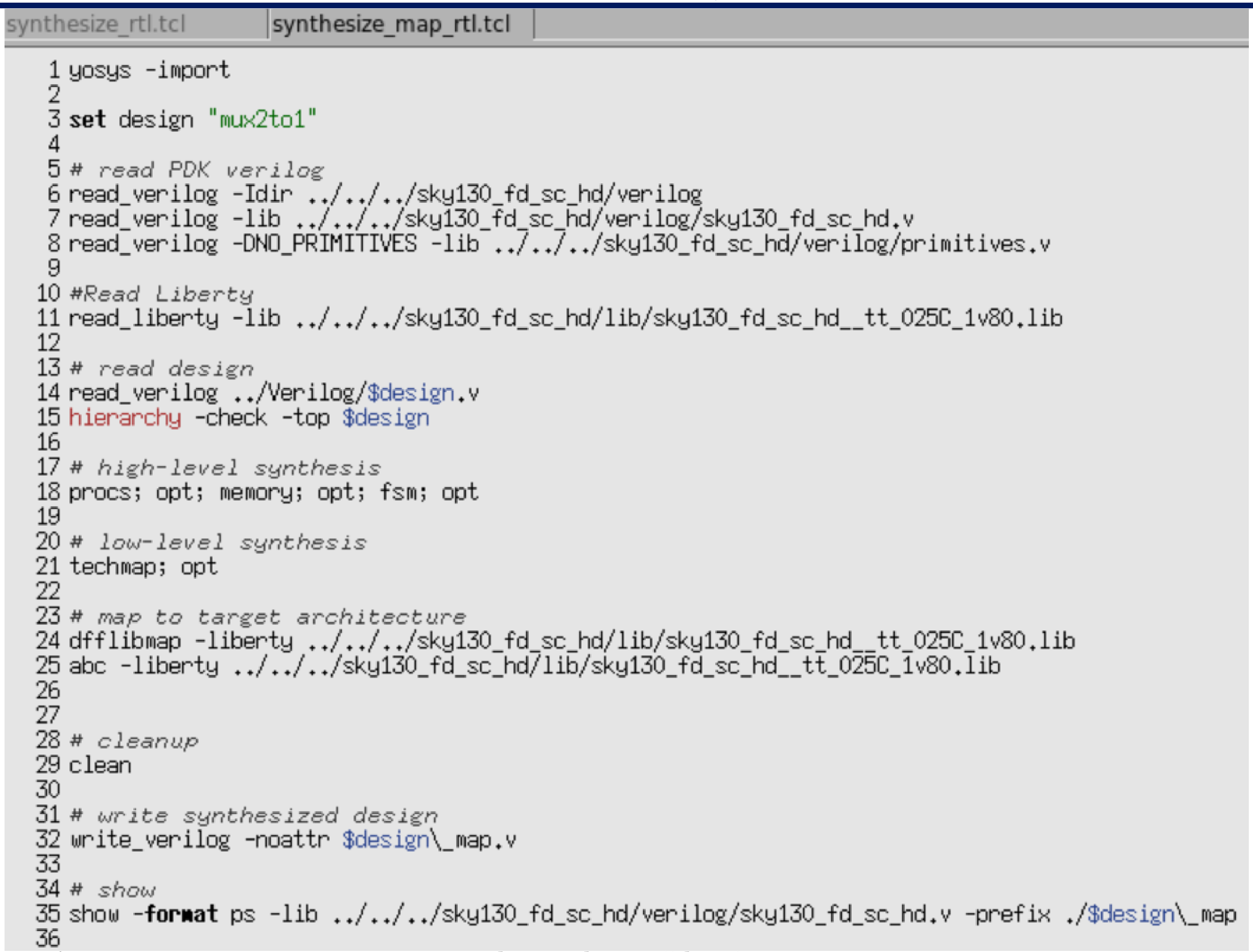
synthesize_rtl.tcl | mux2to1_rtl.v

```
/* Generated by Yosys 0.45+139 (g

module mux2to1(Y, A, B, SEL);
  input A;
  wire A;
  input B;
  wire B;
  input SEL;
  wire SEL;
  output Y;
  wire Y;
  assign Y = SEL ? B : A;
endmodule

```



mux2to1

Logic Synthesis: Example (Yosys map to library)

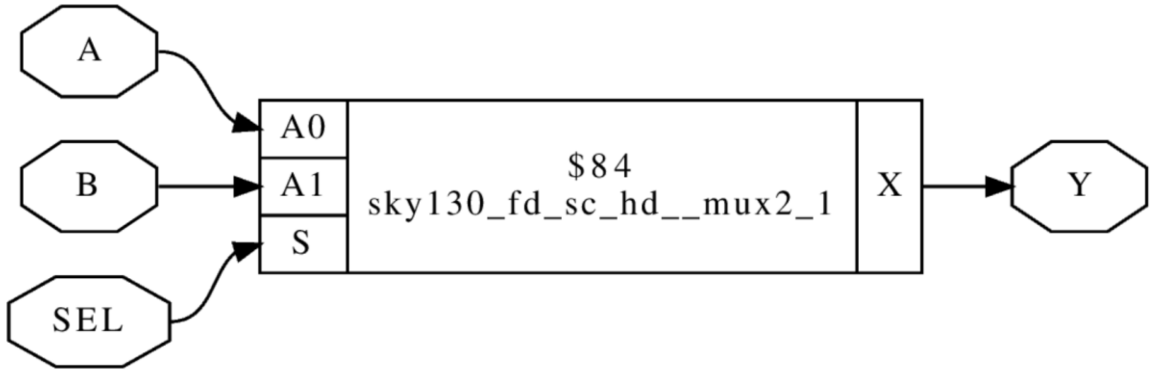


```
yosys -import

set design "mux2to1"

# read PDK verilog
read_verilog -Idir ../../../sky130_fd_sc_hd/verilog
read_verilog -lib ../../../sky130_fd_sc_hd/verilog/sky130_fd_sc_hd.v
read_verilog -DNO_PRIMITIVES -lib ../../../sky130_fd_sc_hd/verilog/primitives.v

#Read Liberty
read_liberty -lib ../../../sky130_fd_sc_hd/lib/sky130_fd_sc_hd__tt_025C_1v80.lib

# read design
read_verilog ../Verilog/$design.v
hierarchy -check -top $design

# high-level synthesis
procs; opt; memory; opt; fsm; opt

# low-level synthesis
techmap; opt

# map to target architecture
dfflibmap -liberty ../../../sky130_fd_sc_hd/lib/sky130_fd_sc_hd__tt_025C_1v80.lib
abc -liberty ../../../sky130_fd_sc_hd/lib/sky130_fd_sc_hd__tt_025C_1v80.lib


# cleanup
clean

# write synthesized design
write_verilog -noattr $design\_map.v

# show
show -format ps -lib ../../../sky130_fd_sc_hd/verilog/sky130_fd_sc_hd.v -prefix ./$design\_map

```

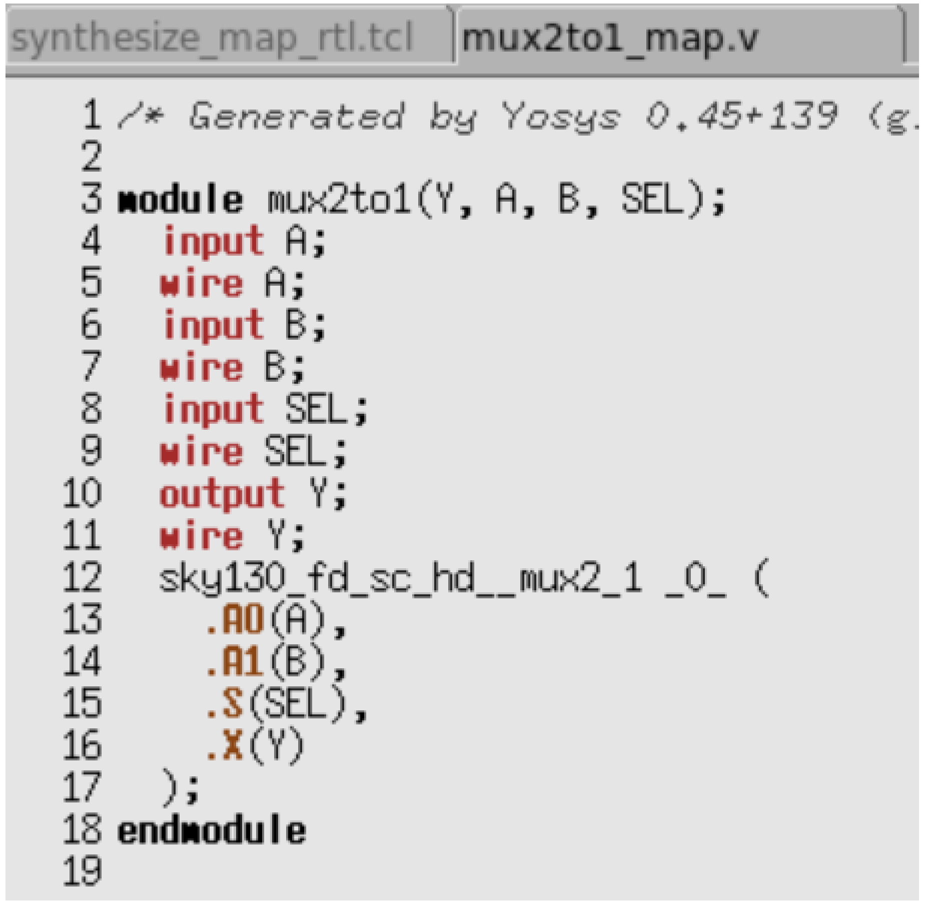
Logic Synthesis: Example (Yosys map to library)

Call Yosys on the terminal with the tcl script

```
yosys synthesize_map_rtl.tcl
```

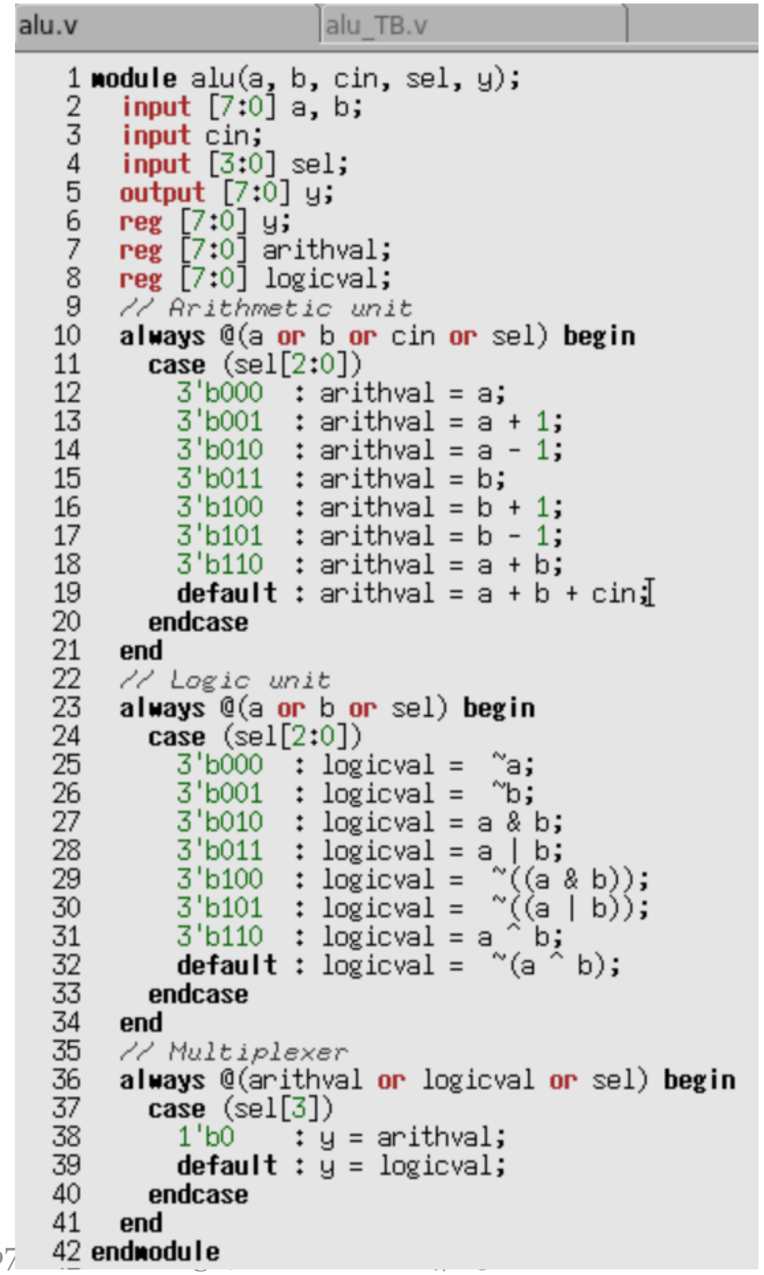


mux2to1



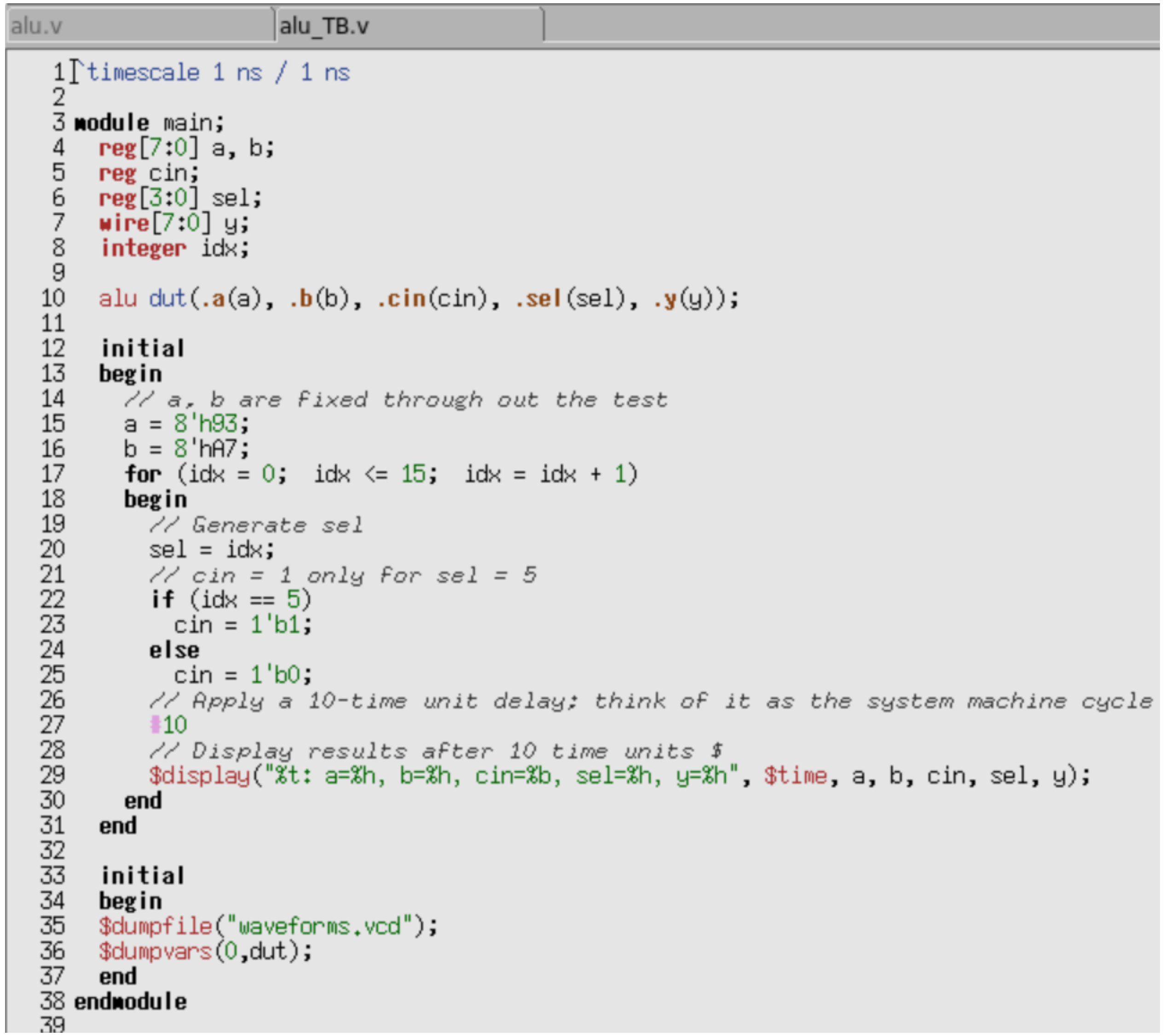
```
/* Generated by Yosys 0.45+139 (g.

module mux2to1(Y, A, B, SEL);
  input A;
  wire A;
  input B;
  wire B;
  input SEL;
  wire SEL;
  output Y;
  wire Y;
  sky130_fd_sc_hd__mux2_1 _0_ (
    .A0(A),
    .A1(B),
    .S(SEL),
    .X(Y)
  );
endmodule

```

Logic Synthesis: Example ALU



alu.v | alu_TB.v

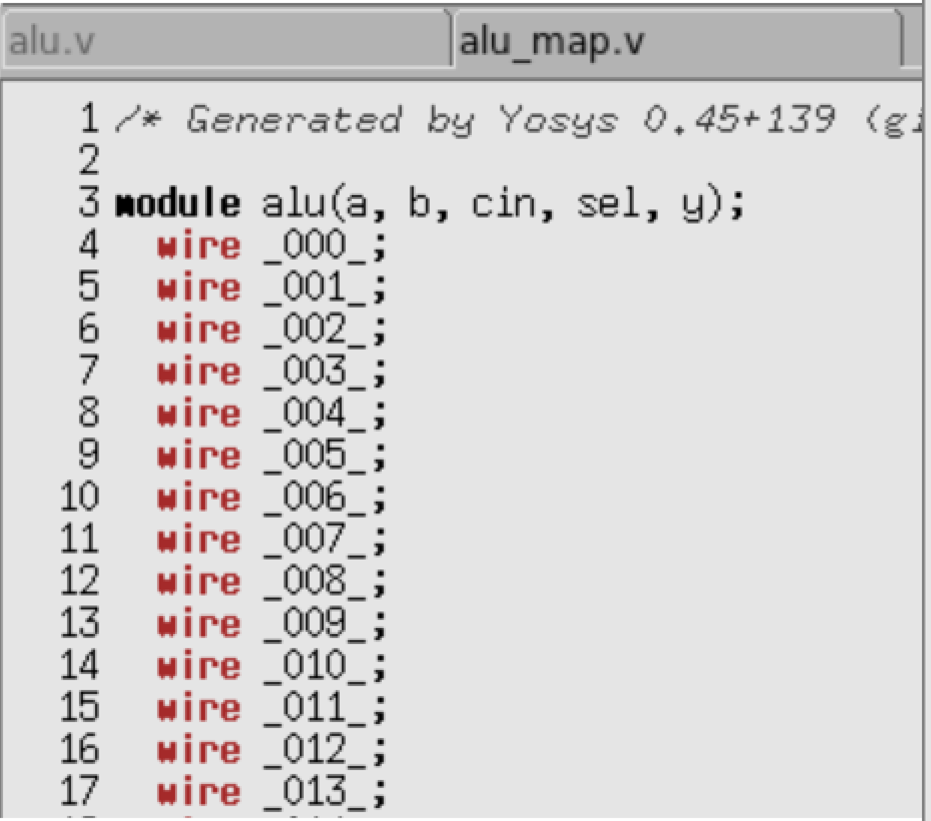
```verilog
module alu(a, b, cin, sel, y);
  input [7:0] a, b;
  input cin;
  input [3:0] sel;
  output [7:0] y;
  reg [7:0] y;
  reg [7:0] arithval;
  reg [7:0] logicval;
  // Arithmetic unit
  always @(a or b or cin or sel) begin
    case (sel[2:0])
      3'b000  : arithval = a;
      3'b001  : arithval = a + 1;
      3'b010  : arithval = a - 1;
      3'b011  : arithval = b;
      3'b100  : arithval = b + 1;
      3'b101  : arithval = b - 1;
      3'b110  : arithval = a + b;
      default : arithval = a + b + cin;
    endcase
  end
  // Logic unit
  always @(a or b or sel) begin
    case (sel[2:0])
      3'b000  : logicval =  ~a;
      3'b001  : logicval =  ~b;
      3'b010  : logicval = a & b;
      3'b011  : logicval = a | b;
      3'b100  : logicval =  ~((a & b));
      3'b101  : logicval =  ~((a | b));
      3'b110  : logicval = a ^ b;
      default : logicval =  ~(a ^ b);
    endcase
  end
  // Multiplexer
  always @(arithval or logicval or sel) begin
    case (sel[3])
      1'b0    : y = arithval;
      default : y = logicval;
    endcase
  end
endmodule
```



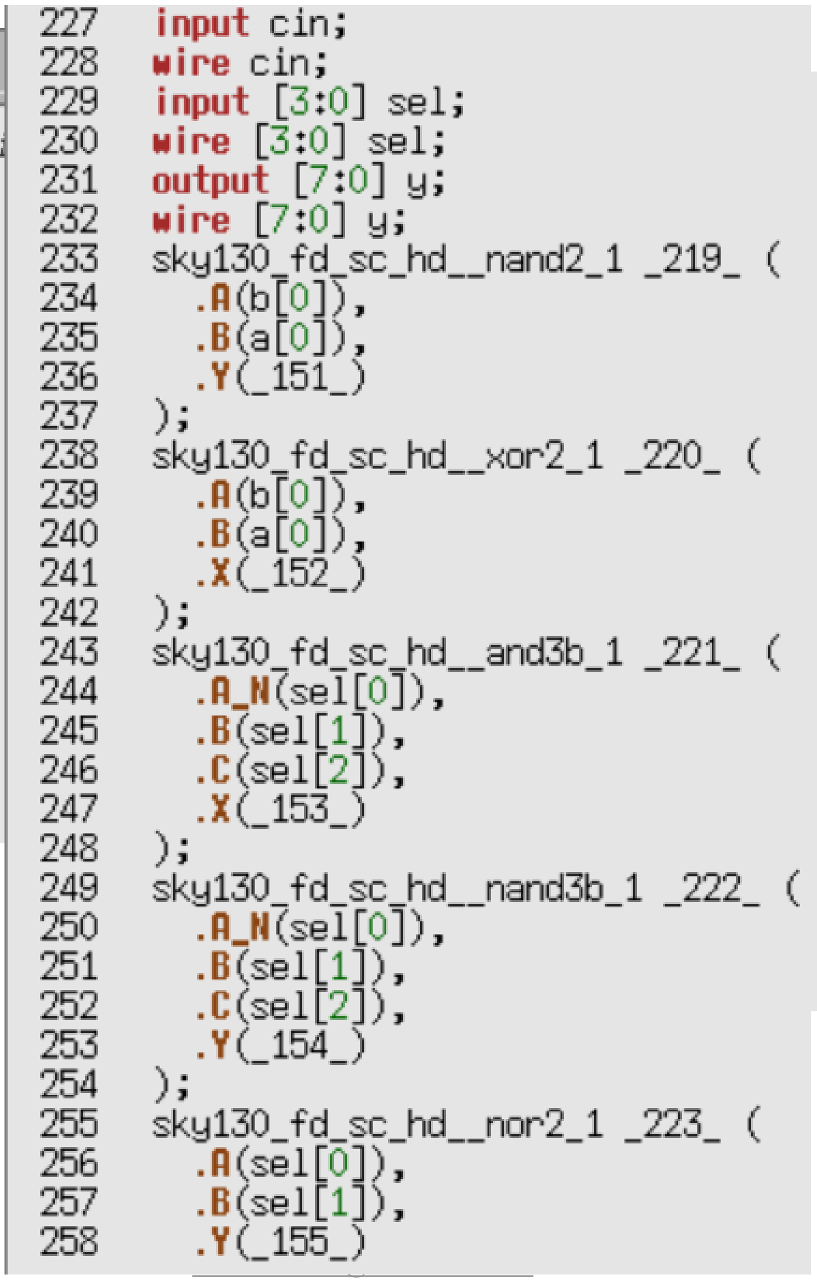
alu.v | alu_TB.v

```verilog
`timescale 1 ns / 1 ns

module main;
  reg[7:0] a, b;
  reg cin;
  reg[3:0] sel;
  wire[7:0] y;
  integer idx;

  alu dut(.a(a), .b(b), .cin(cin), .sel(sel), .y(y));

  initial
  begin
    // a, b are fixed through out the test
    a = 8'h93;
    b = 8'hA7;
    for (idx = 0;  idx <= 15;  idx = idx + 1)
    begin
      // Generate sel
      sel = idx;
      // cin = 1 only for sel = 5
      if (idx == 5)
        cin = 1'b1;
      else
        cin = 1'b0;
      // Apply a 10-time unit delay; think of it as the system machine cycle
      #10
      // Display results after 10 time units $
      $display("%t: a=%h, b=%h, cin=%b, sel=%h, y=%h", $time, a, b, cin, sel, y);
    end
  end

  initial
  begin
  $dumpfile("waveforms.vcd");
  $dumpvars(0,dut);
  end
endmodule
```

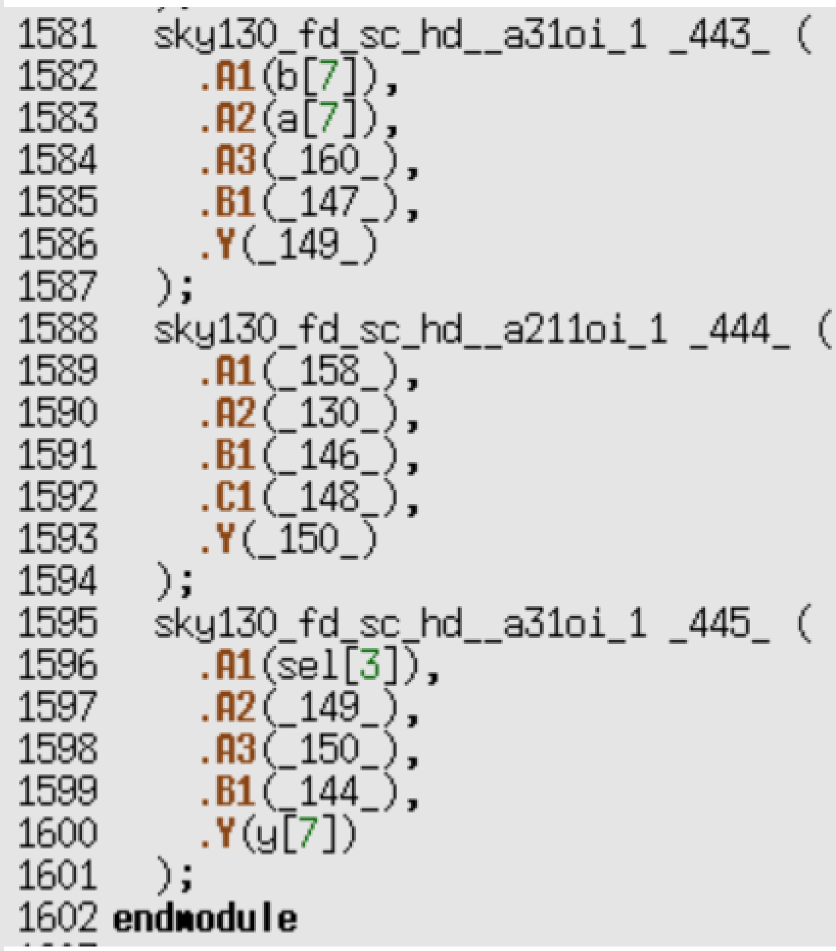
Logic Synthesis: Example ALU Yosys output



```
/* Generated by Yosys 0.45+139 (gi

module alu(a, b, cin, sel, y);
  wire _000_;
  wire _001_;
  wire _002_;
  wire _003_;
  wire _004_;
  wire _005_;
  wire _006_;
  wire _007_;
  wire _008_;
  wire _009_;
  wire _010_;
  wire _011_;
  wire _012_;
  wire _013_;
```



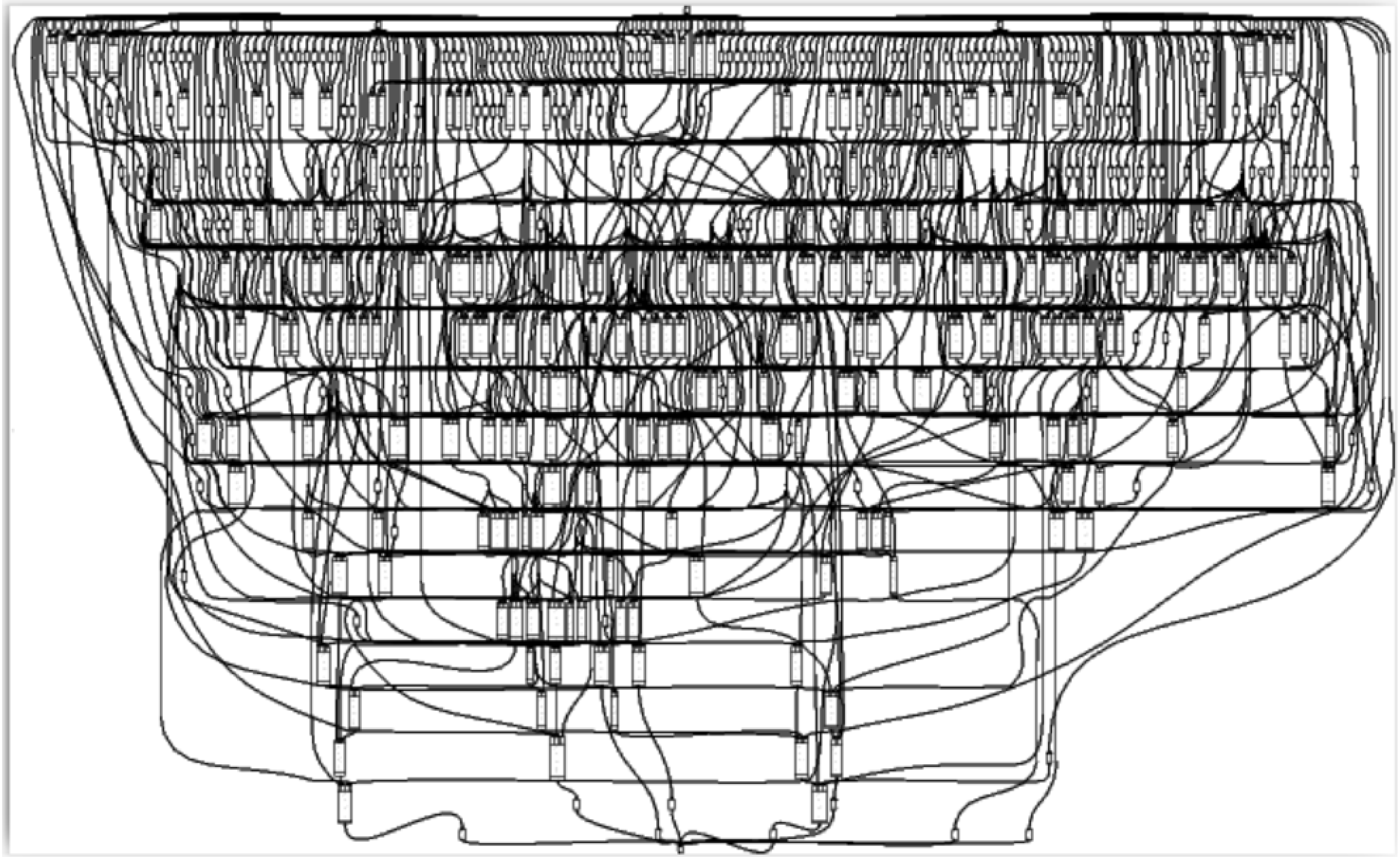
```
  input cin;
  wire cin;
  input [3:0] sel;
  wire [3:0] sel;
  output [7:0] y;
  wire [7:0] y;
  sky130_fd_sc_hd__nand2_1 _219_ (
    .A(b[0]),
    .B(a[0]),
    .Y(_151_)
  );
  sky130_fd_sc_hd__xor2_1 _220_ (
    .A(b[0]),
    .B(a[0]),
    .X(_152_)
  );
  sky130_fd_sc_hd__and3b_1 _221_ (
    .A_N(sel[0]),
    .B(sel[1]),
    .C(sel[2]),
    .X(_153_)
  );
  sky130_fd_sc_hd__nand3b_1 _222_ (
    .A_N(sel[0]),
    .B(sel[1]),
    .C(sel[2]),
    .Y(_154_)
  );
  sky130_fd_sc_hd__nor2_1 _223_ (
    .A(sel[0]),
    .B(sel[1]),
    .Y(_155_)
```



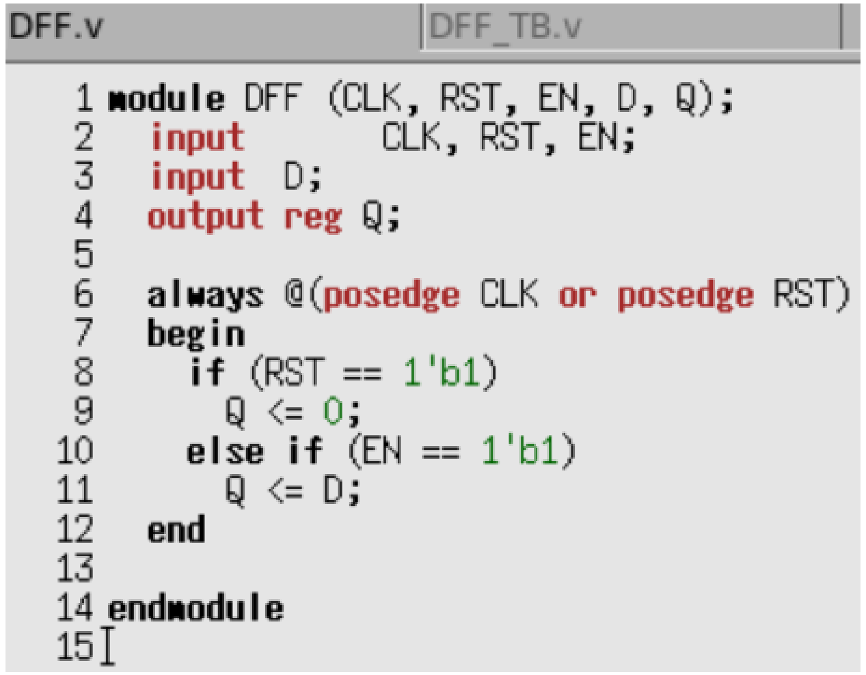
```
  sky130_fd_sc_hd__a31oi_1 _443_ (
    .A1(b[7]),
    .A2(a[7]),
    .A3(_160_),
    .B1(_147_),
    .Y(_149_)
  );
  sky130_fd_sc_hd__a211oi_1 _444_ (
    .A1(_158_),
    .A2(_130_),
    .B1(_146_),
    .C1(_148_),
    .Y(_150_)
  );
  sky130_fd_sc_hd__a31oi_1 _445_ (
    .A1(sel[3]),
    .A2(_149_),
    .A3(_150_),
    .B1(_144_),
    .Y(y[7])
  );
endmodule
```

Only snippets are shown – full file is uploaded on Moodle

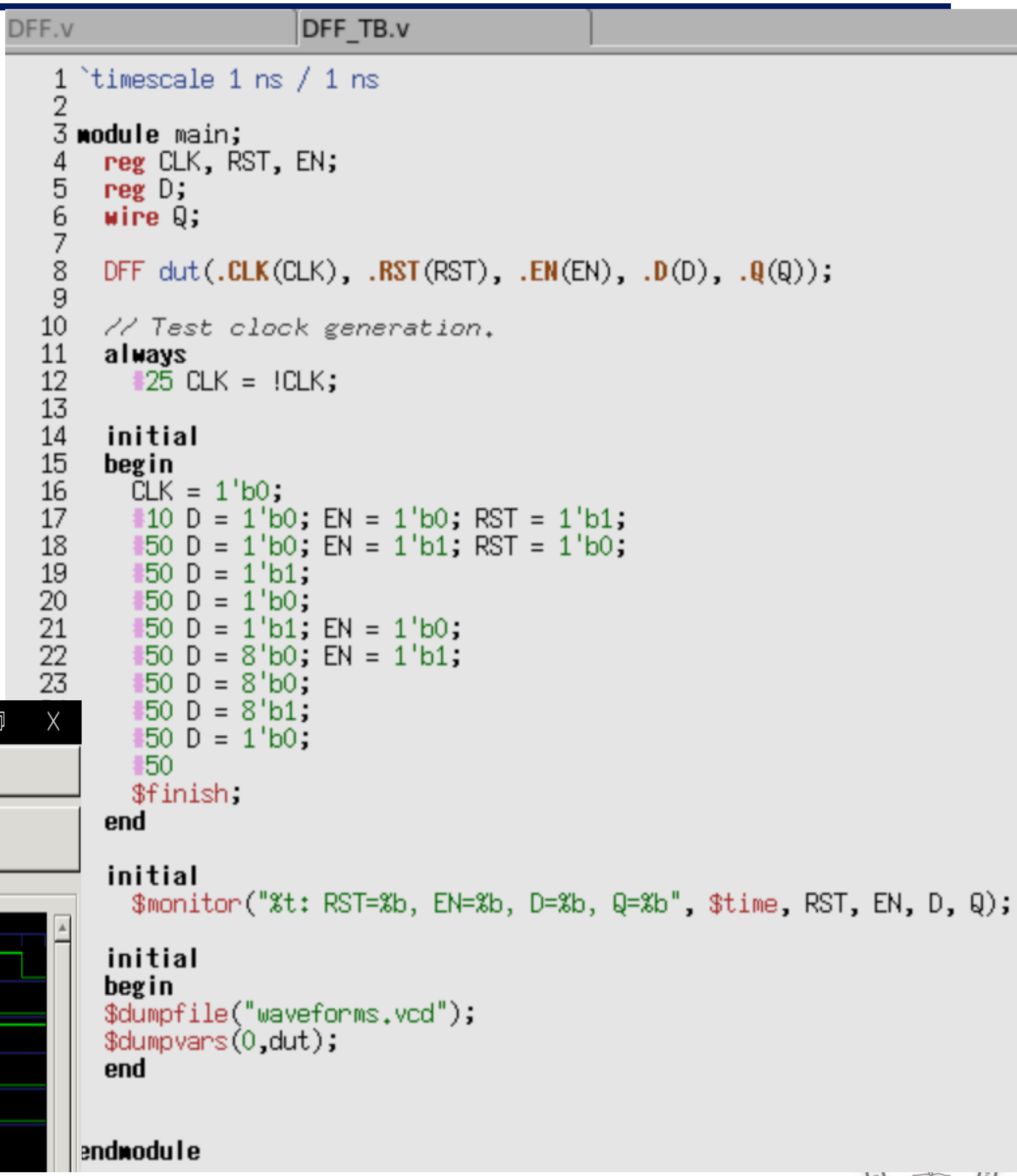
Logic Synthesis: Example ALU Yosys output



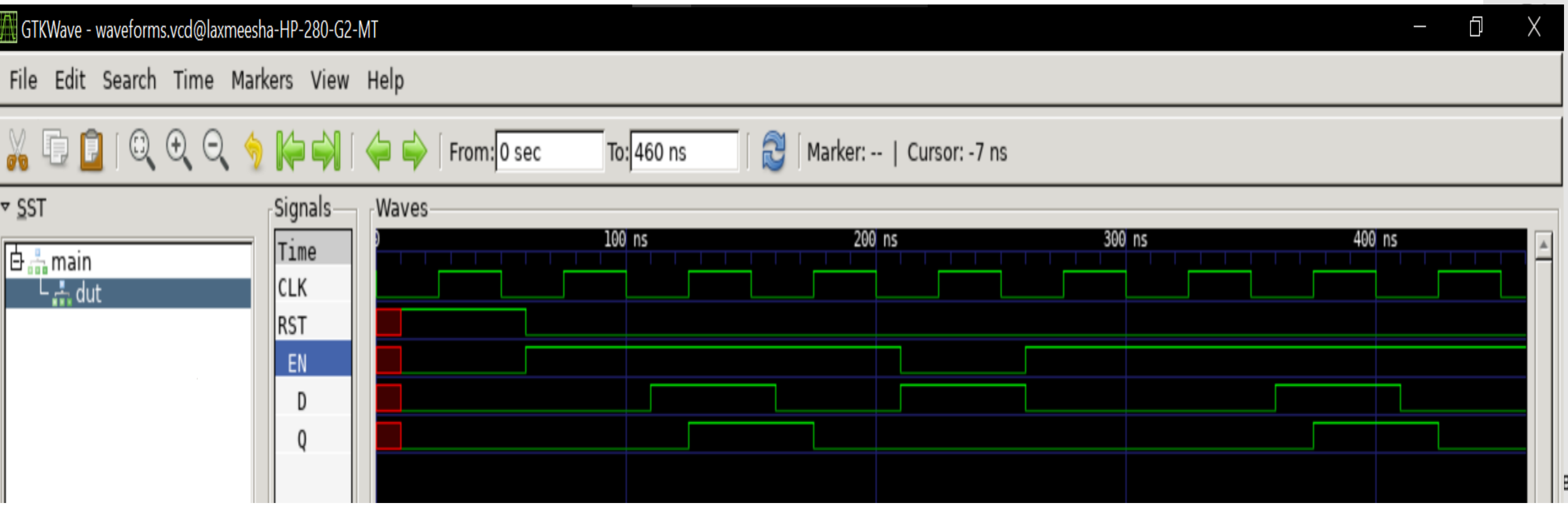
LOGIC SYNTHESIS: EXAMPLE SEQUENTIAL



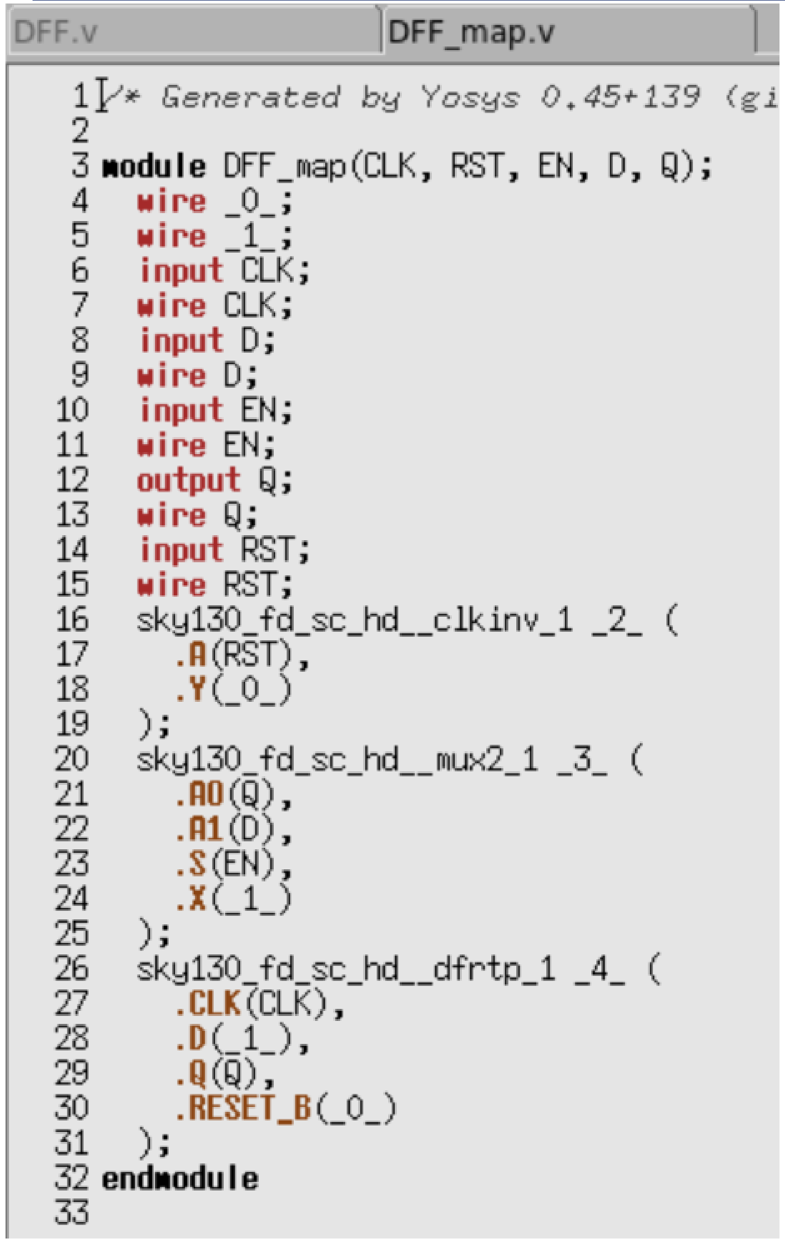
```verilog
module DFF (CLK, RST, EN, D, Q);
  input        CLK, RST, EN;
  input   D;
  output reg Q;

  always @(posedge CLK or posedge RST)
  begin
    if (RST == 1'b1)
      Q <= 0;
    else if (EN == 1'b1)
      Q <= D;
  end

endmodule
```



```verilog
`timescale 1 ns / 1 ns

module main;
  reg CLK, RST, EN;
  reg D;
  wire Q;

  DFF dut(.CLK(CLK), .RST(RST), .EN(EN), .D(D), .Q(Q));

  // Test clock generation.
  always
    #25 CLK = !CLK;

  initial
  begin
    CLK = 1'b0;
    #10 D = 1'b0; EN = 1'b0; RST = 1'b1;
    #50 D = 1'b0; EN = 1'b1; RST = 1'b0;
    #50 D = 1'b1;
    #50 D = 1'b0;
    #50 D = 1'b1; EN = 1'b0;
    #50 D = 8'b0; EN = 1'b1;
    #50 D = 8'b0;
    #50 D = 8'b1;
    #50 D = 1'b0;
    #50
    $finish;
  end

  initial
    $monitor("%t: RST=%b, EN=%b, D=%b, Q=%b", $time, RST, EN, D, Q);

  initial
  begin
  $dumpfile("waveforms.vcd");
  $dumpvars(0,dut);
  end

endmodule
```



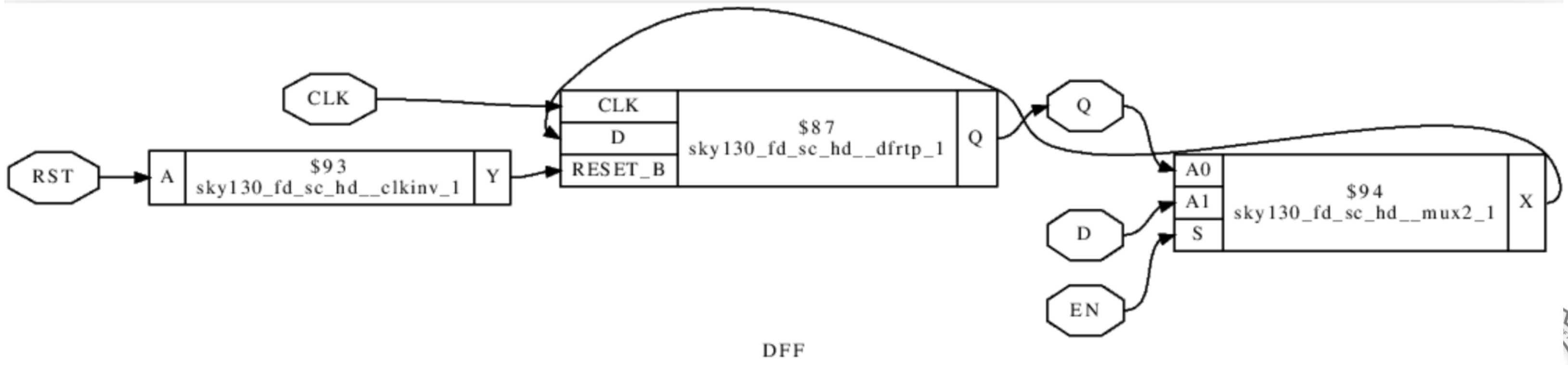
LOGIC SYNTHESIS: EXAMPLE SEQUENTIAL



DFF.v | DFF_map.v

```
/* Generated by Yosys 0.45+139 (gi

module DFF_map(CLK, RST, EN, D, Q);
  wire _0_;
  wire _1_;
  input CLK;
  wire CLK;
  input D;
  wire D;
  input EN;
  wire EN;
  output Q;
  wire Q;
  input RST;
  wire RST;
  sky130_fd_sc_hd__clkinv_1 _2_ (
    .A(RST),
    .Y(_0_)
  );
  sky130_fd_sc_hd__mux2_1 _3_ (
    .A0(Q),
    .A1(D),
    .S(EN),
    .X(_1_)
  );
  sky130_fd_sc_hd__dfrtp_1 _4_ (
    .CLK(CLK),
    .D(_1_),
    .Q(Q),
    .RESET_B(_0_)
  );
endmodule

```

Yosys synthesized a DFF from the library: However, the reset is logic_low for the DFF in the library. So, it inserted an inverter for the reset.
The enable is handled by adding a mux at the input of the DFF


CLK
RST
A
$93
sky130_fd_sc_hd__clkinv_1
Y
CLK
D
RESET_B
$87
sky130_fd_sc_hd__dfrtp_1
Q
Q
D
EN
A0
A1
S
$94
sky130_fd_sc_hd__mux2_1
X
DFF