RUNNING A COMPLETE RTL-TO-LOGIC DESIGN FLOW

In this project I will run a full RTL-to-logic design flow involving synthesis, timing and power analysis and verification. We will use open-source electronic design automation (EDA) tools and an open-source RISC-V microprocessor for this purpose.

TITLE: Running a complete RTL-to-logic design flow

Introduction

In this Project, We will run a full RTL-to-logic design flow involving synthesis, timing and power analysis and verification. We will use open-source electronic design automation (EDA) tools and an open-source RISC-V microprocessor for this purpose.

Digital circuits can be represented at different levels of abstraction. During the design process, a circuit is usually first specified using a higher-level abstraction. Implementation can then be understood as finding a functionally equivalent representation at a lower abstraction level. When this is done automatically using software, the term synthesis is used. In other words, synthesis is the automatic conversion of a high-level representation of a circuit to a functionally equivalent low-level representation of a circuit. In this course we are especially concerned about synthesis from the Register-transfer level (RTL) to a netlist at the logic level (Gates).

Regardless of how a lower-level representation of a circuit is obtained (synthesis or manual design), the lower-level representation is usually verified for functional correctness. While simulation is widely used for verification, in recent years, formal equivalence checking has become a vital verification technique. Once the functional equivalence of the synthesized netlist is established, the timing requirements of the design must be checked. For example, an adder may add, but does it add fast enough? At the logic level the speed metric we are specifically concerned about is the clock period or cycle time which is inversely related to the clock frequency. A timing analyzer is used to verify that a given circuit meets a specific timing constraint, i.e., a target clock period.

Power consumption has become a very important metric in virtually all integrated circuits designed today. Power analysis or estimation is the process of determining how much power is consumed by a given circuit.

It is common to run synthesis, verification, timing, and power analysis in a loop to create an optimized implementation of the design that meets the desired constraints.

This Project will consist of three steps:

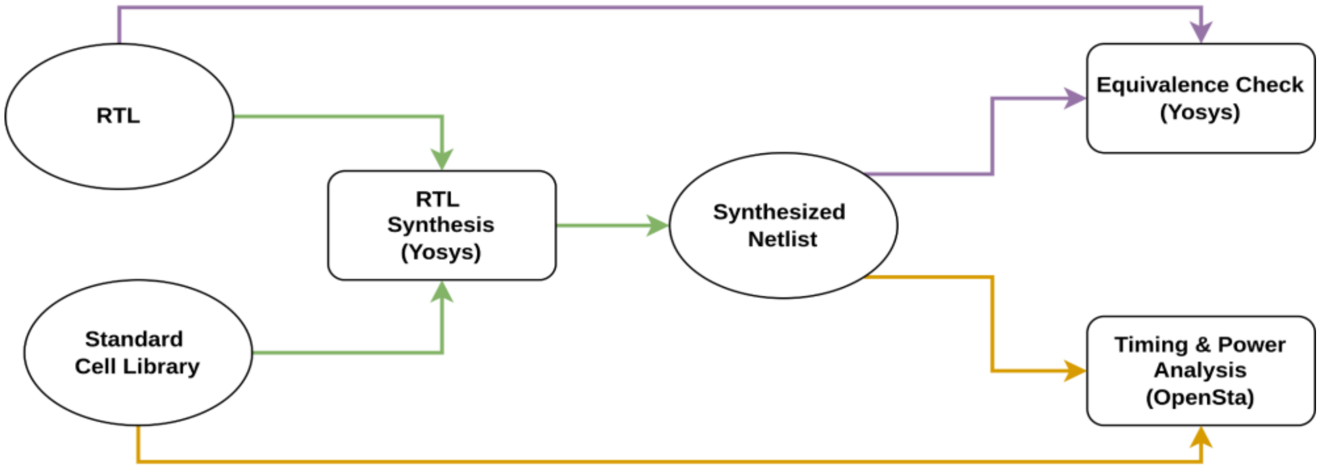
1. RTL Synthesis
2. Functional Equivalence checking
3. Timing and Power Analysis

We will use the following open-source materials:

1. Yosys: Open synthesis suite
2. OpenSta: Gate-level static timing and power analyzer
3. Skywater 130nm: Standard cell library

4. Verilog design files for an open-source 64-bit RISC-V CPU.

The design flow you will step through is illustrated in the following figure.



To learn how to run "*yosys*" in the UNIX environment, see the documentation: Yosys

Run the following command on your terminal:

```
git clone https://github.com/sujay-pandit/ece-51216-tutorial.git
cd ece-51216-tutorial;
```

Note: Before proceeding further with this tutorial, make sure your "gcc" is picked from "/bin/gcc". You can check this by executing "which gcc" on your terminal and modifying your PATH variable as necessary to make sure "/bin/gcc" appears first in your PATH.

Part - I: Logic Synthesis

Logic synthesis refers to the process of transforming RTL to a gate-level netlist. As discussed in class, it consists of two important steps:

- Converting RTL into an optimized technology-independent network of gates.
- Mapping those gates to actual technology-dependent logic gates contained in a technology library (standard cell library).

To achieve this, we will run Yosys with the following script: (The script has already been provided to you in the homework directory)

Input: Design RTL files

Synthesis Script:

```
// Read all Verilog design files
read_verilog *.v;
// Set the top module and check for any problems in the design
hierarchy -check -top mkccore_axi4;
// Transform always_blocks to netlists of RTL multiplexer and
// register cells
proc;
// Perform some logic optimizations and clean-ups
opt;
// Perform FSM optimizations
fsm; opt;
// Optimize memory read/write cells
memory; opt;
// Map all RTL cells to a generic library of gates and
// registers
```

```
techmap; opt;

// Map internal register cell types to the register types
// described in the standard cell library
dfflibmap -liberty $LIB_NAME
// Optimize and Map logic cells to the specified standard cell
// library
abc -dff -liberty $LIB_NAME
// Remove unused cells and wires
clean;
// Write synthesized netlist to a specified file
write_verilog -noattr synth_core.yv
```

Output: Synthesized Netlist (synth_core.yv)

Run the following command on your terminal: (This may take a few minutes)

```
./script_synth.ys
```

Part - II: Logic Equivalence Checking (LEC)

LEC is done to ensure that the synthesis optimizations do not alter the functionality of the design. Since it can take a long time to perform a LEC check on large designs, for this homework, we will perform LEC check on a simple combinational circuit, the single-cycle integer ALU of the core.

We will use the following script along with Yosys to perform LEC:

Input: RTL (module_fn_alu.v) and Synthesized Netlist(synth_alu.yv)
Script:

Input: RTL (module_fn_alu.v) and Synthesized Netlist(synth_alu.yv)

Script:

```
// Read golden RTL file
read_verilog module_fn_alu.v
// Prepare for Verification flow
prep -flatten -top module_fn_alu
design -stash gold
// Read synthesized netlist
read_verilog synth_alu.yv
// Read the Standard cell library used for synthesis
read_liberty -ignore_miss_func $LIB_NAME
// Prepare for Verification flow (Replace standard cells with their
// logic function)
prep -flatten -top module_fn_alu
design -stash gate
design -copy-from gold -as gold module_fn_alu
design -copy-from gate -as gate module_fn_alu
// Make equivalence circuit
equiv_make gold gate equiv
hierarchy -top equiv
// Show the equivalence circuit (Commented out for now below)
#show -pause
// Perform equivalence check
equiv_simple -v
// Report if any errors are found
equiv_status -assert
```

Output: Proven or Fails (Below is a sample output)

Run the following command on your terminal:

```
./script_equivalence.ys
```

Part-III: Timing and Power Analysis

Once the synthesized logic is verified, we perform static timing and power analysis to check if the synthesized netlist can meet desired performance and power constraints. For this purpose, we will use OpenSta with the following script:

Input: Standard cell library file and Synthesized Netlist (synth_core.yv)

Script:

```
// Read Standard cell library used for synthesis
read_liberty $LIB_NAME
// Read the synthesized netlist
read_verilog synth_core.yv
// Link top-level module
link_design mkccore_axi4
// Create clock, link it to the clock port in the design and specify the
// clock period constraint (4000ps = 250MHz)
create_clock -name clk -period 4000 {CLK}
// Generate timing report
report_checks
// Generate power report
report_power
```

Output: Timing and Power report (timingpower.log)

Run the following command on your terminal:

```
./sta timingpower.tcl
```

4. [BONUS]

So far in the project we have proven equivalence for a combinational ALU circuit and a multiplier circuit. In this part, we are required to prove equivalence for a multi-cycle restoring divider.
Modify the synthesis script and equivalence checking script to make "mkrestoring_div.v" the top module.
Go through the Yosys manual and identify the commands you will need to prove equivalence.
If you are successful, you will see a similar terminal output showing the message as follows

```
153. Executing EQUIV_STATUS pass.
Found __x__ $equiv cells in equiv:
Of those cells __x__ are proven and 0 are unproven.
Equivalence successfully proven!
End of script. Logfile hash: 84be3a9e96
CPU: user 18.24s system 0.04s, MEM: 210.88 MB total, 201.65 MB resident
Yosys 0.9 (git sha1 1979e0b)
Time spent: 63% 1x equiv_induct (11 sec), 11% 147x read_verilog (2 sec), …
```

SUBMISSION:

Question-1:

Part - I: Logic Synthesis

Input: Design RTL files

Synthesis Script: `./script_synth.ys`

Result: last 10 lines……..

```
ABC RESULTS:        internal signals:     2995
ABC RESULTS:           input signals:      332
ABC RESULTS:          output signals:      131
Removing temp directory.
Removed 0 unused cells and 1189 unused wires.

13. Executing Verilog backend.
Dumping module `\module_fn_alu'.

Warnings: 6 unique messages, 54 total
End of script. Logfile hash: 145d5afe77
CPU: user 1.40s system 0.02s, MEM: 33.80 MB total, 27.45 MB resident
Yosys 0.9 (git sha1 1979e0b)
Time spent: 19% 13x opt_merge (0 sec), 15% 2x write_verilog (0 sec), ...
```

Part - II: Logic Equivalence Checking (LEC)

Input: RTL (module_fn_alu.v) and Synthesized Netlist(synth_alu.yv)

Synthesis Script: ./`script_equivalence.ys`

Result: last 10 lines……..

```
Problem size at t=1: 5507 literals, 14061 clauses
Proved equivalence! Marking $equiv cell as proven.
Proved 391 previously unproven $equiv cells.

9. Executing EQUIV_STATUS pass.
Found 391 $equiv cells in equiv:
 Of those cells 391 are proven and 0 are unproven.
 Equivalence successfully proven!

End of script. Logfile hash: 078fc42356
CPU: user 6.95s system 0.05s, MEM: 80.09 MB total, 72.76 MB resident
Yosys 0.9 (git sha1 1979e0b)
```

```
Time spent: 64% 1x equiv_simple (4 sec), 9% 9x opt_clean (0 sec), ...
```

Part-III: Timing and Power Analysis

Input: Standard cell library file and Synthesized Netlist (synth_core.yv)

Script: `./sta timingpower.tcl`

Result: last 10 lines……..

```
4000.00 4000.00   clock clk (rise edge)
  0.00 4000.00   clock network delay (ideal)
  0.00 4000.00   clock reconvergence pessimism
       4000.00 ^ imem/icache/m_data/v_data_0_ram_single_0/_8068_/CLK
(sky130_fd_sc_hvl__dfxtp_1)
 -0.70 3999.30   library setup time
       3999.30   data required time
-----------------------------------------------------------
       3999.30   data required time
       -5703.36   data arrival time
-----------------------------------------------------------
       -1704.07   slack (VIOLATED)
Group                   Internal  Switching    Leakage      Total
                           Power      Power      Power      Power (Watts)
------------------------------------------------------------------
Sequential              1.63e-02   2.86e-04   1.00e-01   1.17e-01  75.2%
Combinational           3.76e-02   8.36e-04   0.00e+00   3.84e-02  24.7%
Macro                   0.00e+00   0.00e+00   0.00e+00   0.00e+00   0.0%
Pad                     0.00e+00   0.00e+00   0.00e+00   0.00e+00   0.0%
------------------------------------------------------------------
Total                   5.40e-02   1.12e-03   1.00e-01   1.56e-01 100.0%
                           34.7%       0.7%      64.6%
```

Question-2:

The library files used, total power consumptions and timing reports have been highlighted in red.

Library1: `sky130_fd_sc_hd__tt_100C_1v80.lib`

```
4000.00 4000.00   clock clk (rise edge)
  0.00 4000.00   clock network delay (ideal)
  0.00 4000.00   clock reconvergence pessimism
       4000.00 ^ imem/icache/m_data/v_data_0_ram_single_0/_10962_/CLK
(sky130_fd_sc_hd__dfxtp_1)
 -0.87 3999.13   library setup time
       3999.13   data required time
```

```
---------------------------------------------------------
       3999.13   data required time
       -3569.62   data arrival time
---------------------------------------------------------
        429.51   slack (MET)

Group                  Internal  Switching    Leakage      Total
                          Power       Power       Power       Power  (Watts)
----------------------------------------------------------------
Sequential             3.42e-03   6.72e-05   8.36e-04   4.33e-03   74.2%
Combinational          6.31e-04   1.68e-04   7.10e-04   1.51e-03   25.9%
Macro                  0.00e+00   0.00e+00   0.00e+00   0.00e+00    0.0%
Pad                    0.00e+00   0.00e+00   0.00e+00   0.00e+00    0.0%
----------------------------------------------------------------
Total                  4.06e-03   2.35e-04   1.54e-03   5.83e-03 100.0%
                          69.6%        4.0%       26.4%
```

Library2: sky130_fd_sc_hs__tt_100C_1v80.lib

```
4000.00 4000.00   clock clk (rise edge)
  0.00 4000.00   clock network delay (ideal)
  0.00 4000.00   clock reconvergence pessimism
       4000.00 ^ imem/icache/m_data/v_data_0_ram_single_0/_10936_/CLK
(sky130_fd_sc_hs__dfxtp_1)
 -0.65 3999.35   library setup time
       3999.35   data required time
---------------------------------------------------------
       3999.35   data required time
       -3303.11   data arrival time
---------------------------------------------------------
        696.24   slack (MET)
Group                  Internal  Switching    Leakage      Total
                          Power       Power       Power       Power  (Watts)
----------------------------------------------------------------
Sequential             4.22e-03   8.54e-05   1.16e-02   1.59e-02   56.2%
Combinational          5.71e-03   2.00e-04   6.61e-03   1.25e-02   44.1%
Macro                  0.00e+00   0.00e+00   0.00e+00   0.00e+00    0.0%
Pad                    0.00e+00   0.00e+00   0.00e+00   0.00e+00    0.0%
----------------------------------------------------------------
Total                  9.92e-03   2.86e-04   1.82e-02   2.84e-02 100.0%
                          35.0%        1.0%       64.0%
```

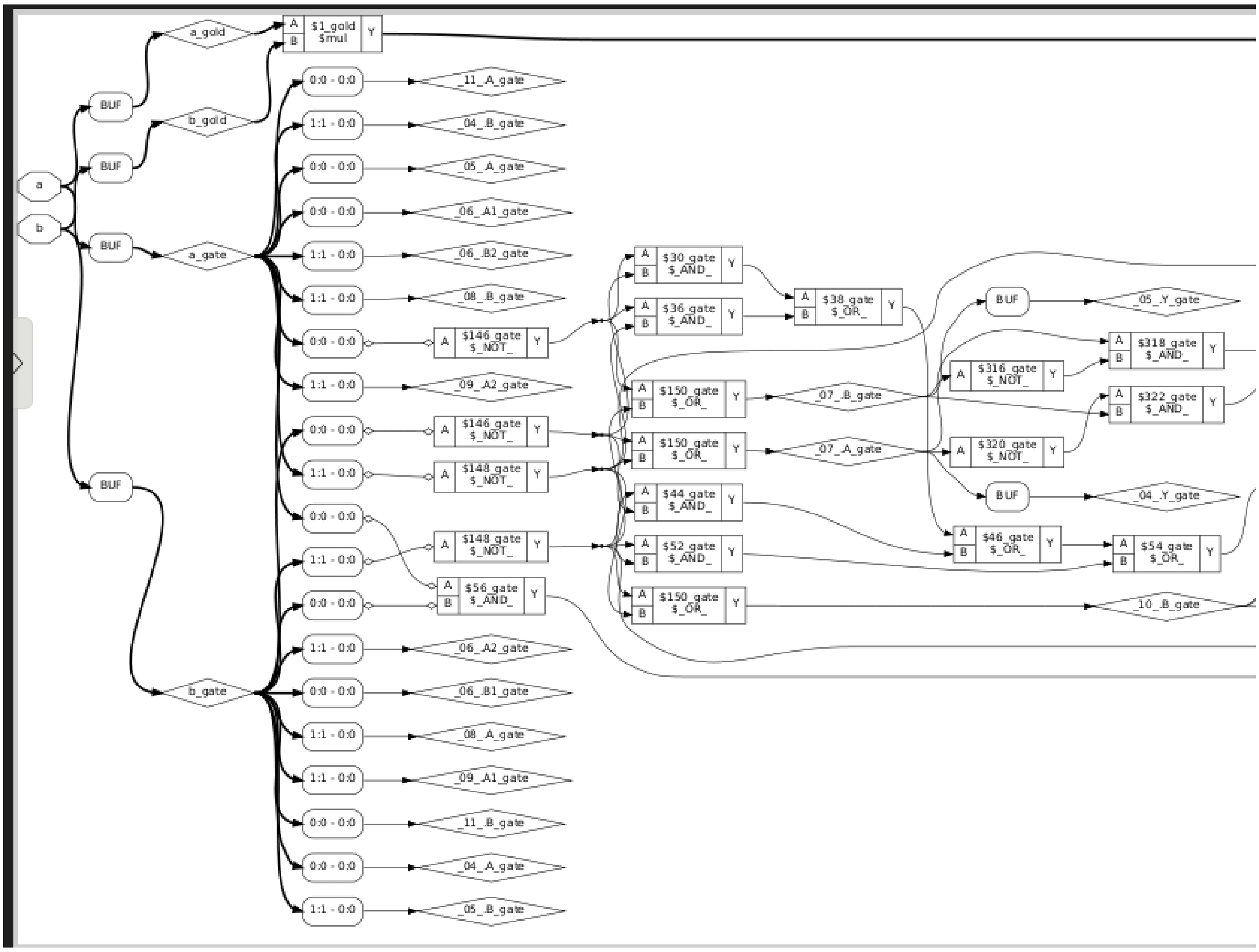
With Library 2 the design has higher slacks even though both the libraries meet timing slack. But, with Library 1 the design has lower power consumptions.

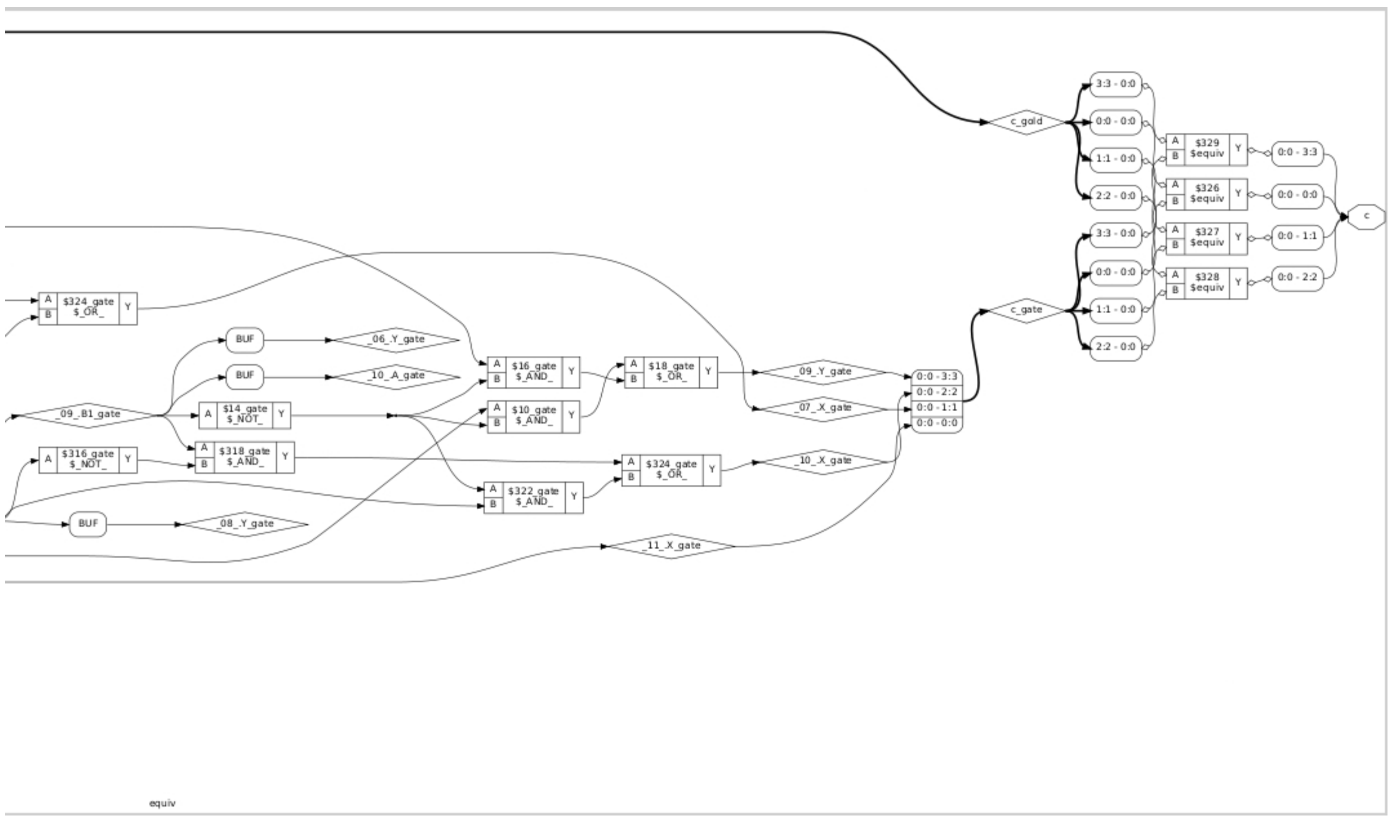
Question-3:

As instructed, I have modified the "script_synth.ys" to perform synthesis on "signedmul.v" and then modified the "script_equivalence.ys" to perform equivalence on it. The resulting dot viewer is shown below. See INDEX for a clearer picture. Here is the generated circuit statistics:

```
=== signedmul ===

   Number of wires:                 50
   Number of wire bits:             55
   Number of public wires:          30
   Number of public wire bits:      35
   Number of memories:               0
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:                 28
     $_AND_                         11
     $_NOT_                          8
     $_OR_                           9
```

The statistics show the no. of gates, and types of gates used, memory bit used. The circuit has two inputs and one output.





Question-4:

File: script_synthesis.ys

```
#!/bin/bash
LIB_NAME=./lib/sky130_fd_sc_hvl__tt_100C_3v30.lib

#
yosys -l synth_alu.log -p "
read_verilog ./verilog/*.v;
hierarchy -check -top mkrestoring_div;
proc; opt; fsm; opt; memory; opt;
techmap; opt;
dfflibmap -liberty $LIB_NAME ;
abc -liberty $LIB_NAME ;
clean;
write_verilog -noattr synth_div.yv
"
```

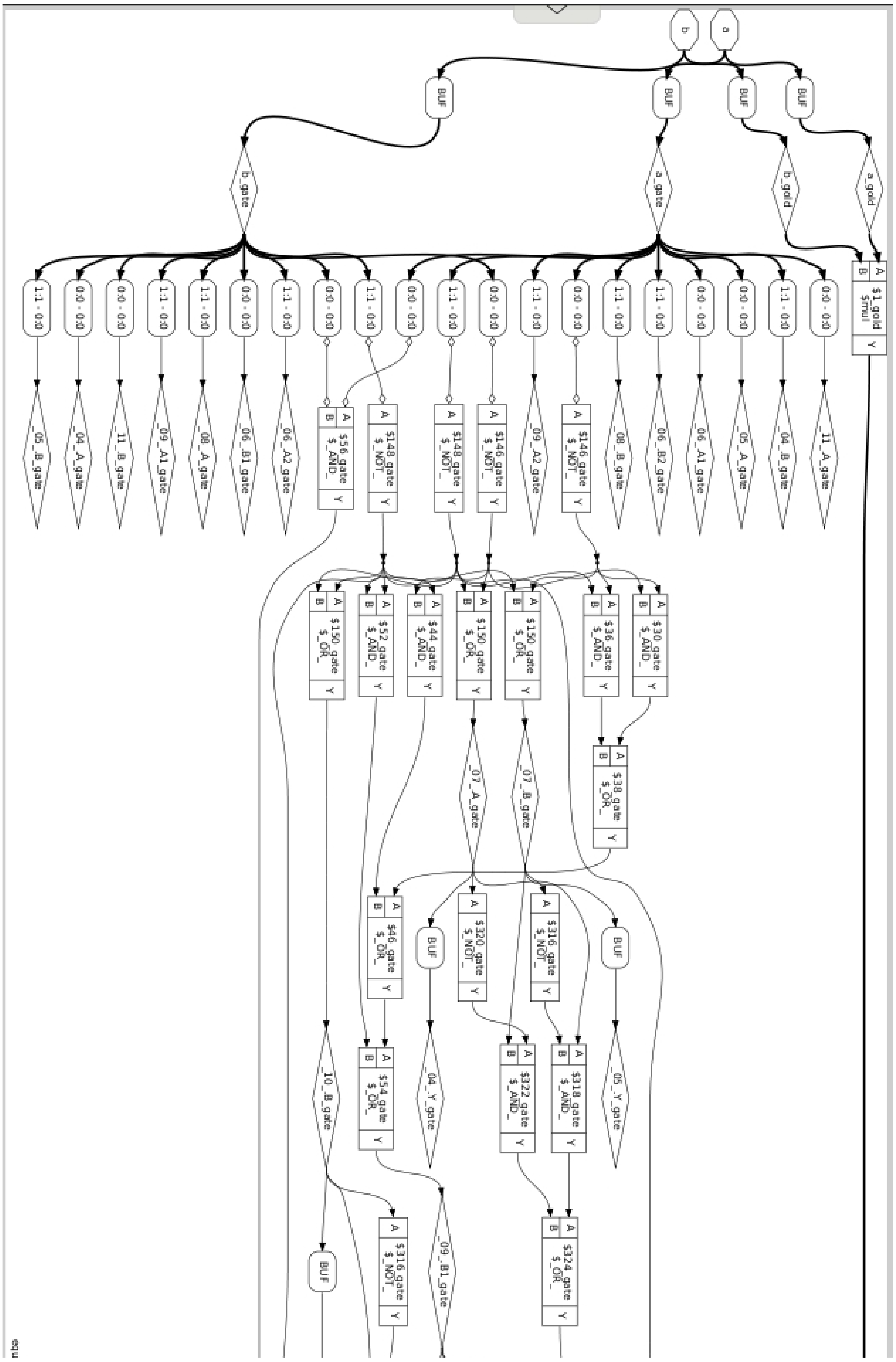
File: script_equivalence.ys

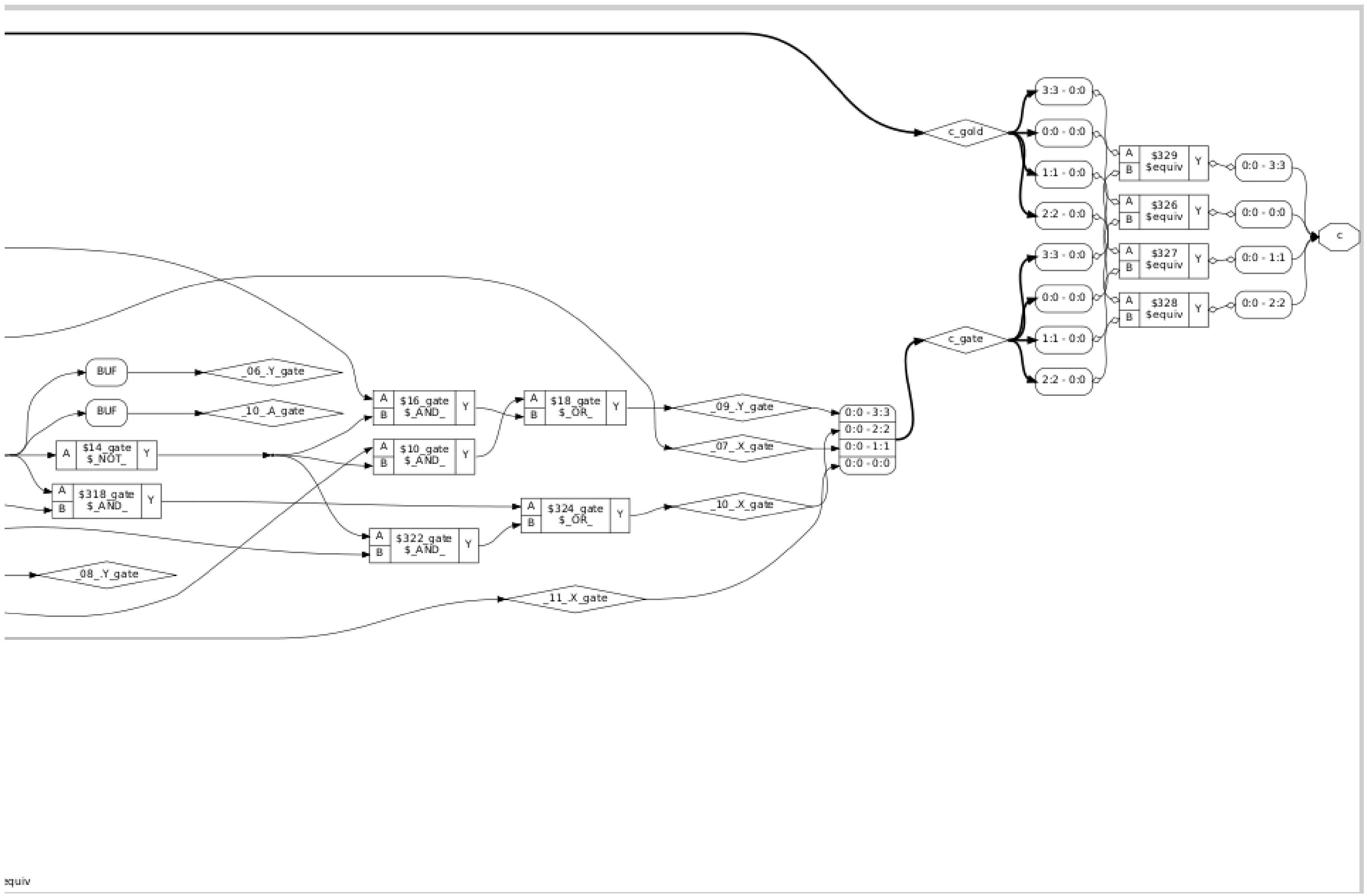
```
 #!/bin/bash
LIB_NAME=./lib/sky130_fd_sc_hvl__tt_100C_3v30.lib

yosys -l check_core.log -p "

    # gold design
    read_verilog ./verilog/*.v
    prep -flatten -top mkrestoring_div
    design -stash gold

    # gate design
    read_verilog synth_div.yv
    read_liberty  -ignore_miss_func $LIB_NAME
    prep -flatten -top mkrestoring_div
    design -stash gate


    # prove equivalence
    design -copy-from gold -as gold mkrestoring_div
    design -copy-from gate -as gate mkrestoring_div
    equiv_make gold gate equiv
    hierarchy -top equiv
    #show -pause
    equiv_induct
    equiv_status -assert
"
```

Output:

```
150. Executing HIERARCHY pass (managing design hierarchy).

150.1. Analyzing design hierarchy..
Top module:  \equiv

150.2. Analyzing design hierarchy..
Top module:  \equiv
Removing unused module `\gate'.
Removing unused module `\gold'.
Removed 2 unused modules.

151. Executing EQUIV_INDUCT pass.
Found 932 unproven $equiv cells in module equiv:
 Proving existence of base case for step 1. (176038 clauses over 67817
variables)
 Proving induction step 1. (358725 clauses over 137628 variables)
 Proof for induction step holds. Entire workset of 932 cells proven!
Proved 932 previously unproven $equiv cells.

152. Executing EQUIV_STATUS pass.
Found 932 $equiv cells in equiv:
 Of those cells 932 are proven and 0 are unproven.
 Equivalence successfully proven!

Warnings: 5 unique messages, 5 total
End of script. Logfile hash: 45c8269f92
CPU: user 18.67s system 0.15s, MEM: 207.93 MB total, 200.61 MB resident
Yosys 0.9 (git sha1 1979e0b)
Time spent: 63% 1x equiv_induct (11 sec), 11% 147x read_verilog (2 sec),
...
```

INDEX:


b
a
BUF
b_gate
a_gate
b_gold
a_gold
$1_gold
$mul
1:1 - 0:0
0:0 - 0:0
_05_B_gate
_04_A_gate
_11_B_gate
_09_A1_gate
_08_A_gate
_06_B1_gate
_06_A2_gate
$56_gate
$_AND_
$148_gate
$_NOT_
$146_gate
_09_A2_gate
_08_B_gate
_06_B2_gate
_06_A1_gate
_05_A_gate
_04_B_gate
_11_A_gate
$150_gate
$_OR_
$52_gate
$44_gate
$36_gate
$30_gate
$38_gate
_07_A_gate
_07_B_gate
$46_gate
$320_gate
$316_gate
$54_gate
$322_gate
$318_gate
_10_B_gate
_04_Y_gate
_05_Y_gate
_09_B1_gate
$324_gate


BUF
_06_Y_gate
BUF
_10_A_gate
$14_gate $_NOT_
$318_gate $_AND_
_08_Y_gate
$16_gate $_AND_
$10_gate $_AND_
$322_gate $_AND_
$18_gate $_OR_
$324_gate $_OR_
_11_X_gate
_09_Y_gate
_07_X_gate
_10_X_gate
0:0 - 3:3
0:0 - 2:2
0:0 - 1:1
0:0 - 0:0
c_gold
c_gate
3:3 - 0:0
0:0 - 0:0
1:1 - 0:0
2:2 - 0:0
$329 $equiv
$326 $equiv
$327 $equiv
$328 $equiv
c
equiv