Yosys Basic Usage Guide

Author: Ryan Cramer

Yosys: **Y**osys **O**pen **SY**nthesus **S**uite

Developed by Claire-Xenia Wolfe in 2012, it is a HDL synthesis tool, which is “open-source”, and distributed under an ISC-like license. It converts your HDL to an Abstract Syntax Tree (AST) and RTLIL representation. It can generate HDL gate-level netlists using a specified PDK, as well as generate datapath and netlist visualization, as well as static timing analysis

Tools Used When Using Yosys:

- **ABC**
  - Open-source logic synthesis and optimization engine (Developed at Berkeley). Yosys delegates technology mapping to it (important for ASIC synthesis)
  - Takes generic logic gates $and, $or, $xor, etc, and maps them to real cells described in the Liberty (.lib) file (e.g. sky130_fd_sc_hd_and2_1)
  - Optimizes the logic for delay or area
  - **Yosys turns RTL into the spec, ABC chooses bricks to use, and how to arrange them**
- **dfflibmap**
  - A Yosys pass specifically for flip-flops and latches
  - Generic registers in RTL get turned into $dff cells in Yosys. But each PDK has multiple FF flavors (different set/reset polarity, async vs sync).
  - dfflibmap matches generic $dff cells against the available cells in the Liberty file
  - **ABC handles combinational, dfflibmap handles sequential**
- **sta**
  - Simple Static Timing Analysis pass inside Yosys
    - Reads timing arcs from Liberty files
    - Computes longest topological path in the design
    - Reports estimated critical path delays
  - Limitations
    - No placement/routing delays (only cell delays)
    - Meant for early estimation before PnR (place and route), for full STA use **OpenSTA**

NOTE: The Yosys version used in the asic-tools container is Yosys 0.51

To learn more about Yosys: please visit the v0.51 documentation

Basic Usage:

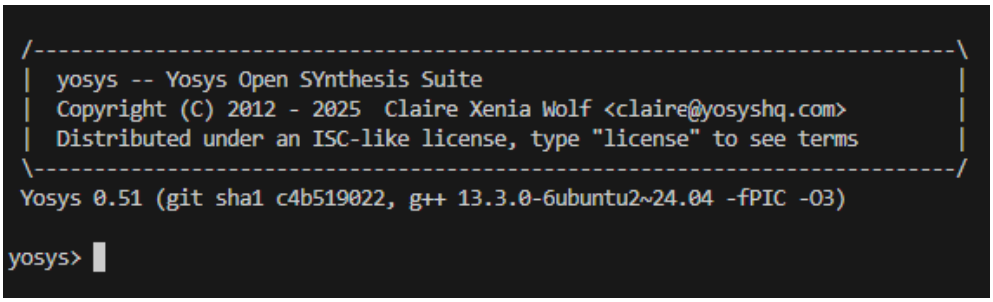
Start with a simple Verilog or SystemVerilog module, such as a 32-bit comparator

```
`timescale 1ns/1ps

module cmp_32(
    input [31:0] A,
    input [31:0] B,
    output logic GT,
    output logic LT,
    output logic EQ
);

    always_comb begin

        GT = 1'b0;
        LT = 1'b0;
        EQ = 1'b0;

        if(A < B) begin
            LT = 1'b1;
        end

        else if(A > B) begin
            GT = 1'b1;
        end

        else begin
            EQ = 1'b1;
        end


    end


endmodule
```

1. **Run Yosys inside the Docker container:**

ubuntu@asic:~/workspace/project$ yosys

...you will then see:



2. **Translate Design: yosys> read_verilog design.v**

The first step is to have Yosys read your RTL. You can read Verilog or SystemVerilog files and parse them into an AST and lowers it into RTLIL form. At this point, the design will be abstracted. It is now behavioral code (not yet the netlist that we want).

Note: Use the "-sv" flag for SystemVerilog

yosys> read_verilog design.v

- or -

yosys> read_verilog -sv design.sv

```
yosys> read_verilog -sv rtl/cmp_32.sv

1. Executing Verilog-2005 frontend: rtl/cmp_32.sv
Parsing SystemVerilog input from `rtl/cmp_32.sv' to AST representation.
Generating RTLIL representation for module `\cmp_32'.
Successfully finished Verilog frontend.
```

3. **Manage Heirarchy: yosys> hierarchy -check -top design**

Ensure that you did not include any unused modules by specifying the top module. It double checks for unused modules and trims them.

```
yosys> hierarchy -check -top cmp_32

2. Executing HIERARCHY pass (managing design hierarchy).

2.1. Analyzing design hierarchy..
Top module:  \cmp_32

2.2. Analyzing design hierarchy..
Top module:  \cmp_32
Removed 0 unused modules.
```

4. **Process Conversion** (proc Pass): **yosys> proc**

Behavioral code becomes structured netlists, and expressions are optimized

PROC_CLEAN	**remove empty switches**
PROC_RMDEAD	**remove dead branches**
PROC_PRUNE	**remove redundant assignments in processes**
PROC_INIT	**extracts init attributes**
PROC_ARST	**detects async resets**
PROC_ROM	**converts switches to ROMs**
PROC_MUX	**covert decision trees to multiplexors**
PROC_DLATCH	**convert process syncs to latches**
PROC_DFF	**convert process syncs to FFs**
PROC_MEMWR	**convert process memory writes to cells**
PROC_CLEAN	**remove empty switches from decision trees**
OPT_EXPR	**optimizes expressions**

```
yosys> proc

3. Executing PROC pass (convert processes to netlists).

3.1. Executing PROC_CLEAN pass (remove empty switches from decision trees).
Cleaned up 0 empty switches.

3.2. Executing PROC_RMDEAD pass (remove dead branches from decision trees).
Marked 2 switch rules as full_case in process $proc$rtl/cmp_32.sv:0$1 in module cmp_32.
Removed a total of 0 dead cases.

3.3. Executing PROC_PRUNE pass (remove redundant assignments in processes).
Removed 0 redundant assignments.
Promoted 3 assignments to connections.

3.4. Executing PROC_INIT pass (extract init attributes).

3.5. Executing PROC_ARST pass (detect async resets in processes).

3.6. Executing PROC_ROM pass (convert switches to ROMs).
Converted 0 switches.
<suppressed ~2 debug messages>
```

5. **Optimization: yosys> opt**

Runs optimization iterations until the program is satisfied.

OPT	**simple optimizations**
OPT_EXPR	**optimize expressions**
OPT_MUXTREE	**detect dead branches in mux trees**
OPT_REDUCE	**consolidate $*mux and $reduce_* inputs**
OPT_MERGE	**detect identical cells**
OPT_DFF	**perform DFF optimizations**
OPT_CLEAN	**remove unused cells and wires**
Rerunning OPT Passes	**reruns OPT passes if optimizations can still be done**

```
yosys> opt

4. Executing OPT pass (performing simple optimizations).

4.1. Executing OPT_EXPR pass (perform const folding).
Optimizing module cmp_32.
```

```
4.8. Executing OPT_EXPR pass (perform const folding).
Optimizing module cmp_32.

4.9. Rerunning OPT passes. (Maybe there is more to do..)

4.10. Executing OPT_MUXTREE pass (detect dead branches in mux trees).
Running muxtree optimizer on module \cmp_32..
  Creating internal representation of mux trees.
  Evaluating internal representation of mux trees.
  Analyzing evaluation results.
Removed 0 multiplexer ports.
<suppressed ~3 debug messages>
```

```
4.14. Executing OPT_CLEAN pass (remove unused cells and wires).
Finding unused cells or wires in module \cmp_32..

4.15. Executing OPT_EXPR pass (perform const folding).
Optimizing module cmp_32.

4.16. Finished OPT passes. (There is nothing left to do.)
```

6. **Flatten: yosys> flatten**

   Flattens the design (n-bit signals will be separated into n single-bit wires)

```
yosys> flatten

5. Executing FLATTEN pass (flatten design).
```

7. **Techmap: yosys> techmap**

   Maps generic ops to generic target cells ($eq = XOR trees, + = Full Adder)

```
yosys> techmap

6. Executing TECHMAP pass (map to technology primitives).

6.1. Executing Verilog-2005 frontend: /foss/tools/yosys/bin/../share/yosys/techmap.v
Parsing Verilog input from `/foss/tools/yosys/bin/../share/yosys/techmap.v' to AST representation.
Generating RTLIL representation for module `\_90_simplemap_bool_ops'.
Generating RTLIL representation for module `\_90_simplemap_reduce_ops'.
Generating RTLIL representation for module `\_90_simplemap_logic_ops'.
Generating RTLIL representation for module `\_90_simplemap_compare_ops'.
Generating RTLIL representation for module `\_90_simplemap_various'.
Generating RTLIL representation for module `\_90_simplemap_registers'.
Generating RTLIL representation for module `\_90_shift_ops_shr_shl_sshl_sshr'.
Generating RTLIL representation for module `\_90_shift_shiftx'.
Generating RTLIL representation for module `\_90_fa'.
Generating RTLIL representation for module `\_90_lcu_brent_kung'.
Generating RTLIL representation for module `\_90_alu'.
Generating RTLIL representation for module `\_90_macc'.
Generating RTLIL representation for module `\_90_alumacc'.
Generating RTLIL representation for module `\$__div_mod_u'.
Generating RTLIL representation for module `\$__div_mod_trunc'.
Generating RTLIL representation for module `\_90_div'.
Generating RTLIL representation for module `\_90_mod'.
Generating RTLIL representation for module `\$__div_mod_floor'.
Generating RTLIL representation for module `\_90_divfloor'.
Generating RTLIL representation for module `\_90_modfloor'.
Generating RTLIL representation for module `\_90_pow'.
Generating RTLIL representation for module `\_90_pmux'.
Generating RTLIL representation for module `\_90_demux'.
Generating RTLIL representation for module `\_90_lut'.
Successfully finished Verilog frontend.

6.2. Continuing TECHMAP pass.
Running "alumacc" on wrapper $extern:wrap:$lt:A_SIGNED=0:A_WIDTH=32:B_SIGNED=0:B_WIDTH=32:Y_WIDTH=1:
Using template $extern:wrap:$lt:A_SIGNED=0:A_WIDTH=32:B_SIGNED=0:B_WIDTH=32:Y_WIDTH=1:394426c56d1a02
t:A_SIGNED=0:A_WIDTH=32:B_SIGNED=0:B_WIDTH=32:Y_WIDTH=1:394426c56d1a028ba8fdd5469b163e04011def47.
Running "alumacc" on wrapper $extern:wrap:$gt:A_SIGNED=0:A_WIDTH=32:B_SIGNED=0:B_WIDTH=32:Y_WIDTH=1:
Using template $extern:wrap:$gt:A_SIGNED=0:A_WIDTH=32:B_SIGNED=0:B_WIDTH=32:Y_WIDTH=1:394426c56d1a02
t:A_SIGNED=0:A_WIDTH=32:B_SIGNED=0:B_WIDTH=32:Y_WIDTH=1:394426c56d1a028ba8fdd5469b163e04011def47.
Using extmapper simplemap for cells of type $mux.
Using template $paramod$fbc7873bff55778c0b3173955b7e4bce1d9d6834\_90_alu for cells of type $alu.
Using extmapper simplemap for cells of type $not.
Using extmapper simplemap for cells of type $or.
Using extmapper simplemap for cells of type $reduce_and.
Using extmapper simplemap for cells of type $xor.
Using extmapper simplemap for cells of type $and.
Using template $paramod\_90_lcu_brent_kung\WIDTH=32'00000000000000000000000000100000 for cells of ty
Using extmapper simplemap for cells of type $pos.
No more expansions possible.
<suppressed ~574 debug messages>
```

8. **Techmap logic using PDK: yosys> abc -liberty <pdk.lib>**

Uses abc with the pdk library to do technology mapping and optimizations for all combinational and gate logic.

To find your .libs, please use the **find_libs.sh** script in asic-tools repository

The actual .lib filenames look like this:

- sky130_fd_sc_hd_tt_025c_1v80.lib (tt = typical speed, 25 C, 1.80 V)
- sky130_fd_sc_hd_ss_100c_1v60.lib (ss = slow speed, 25 C, 1.60 V)
- sky130_fd_sc_hd__ff_n40C_1v95.lib (ff = fast speed, -40 C, 1.95 V)

These are the timing/power models that **abc** consumes

```
yosys> abc -liberty /foss/pdks/volare/sky130/versions/0fe599b2afb6708d281543108caf8310912f54af/sky130A/libs.ref/sky130_fd_sc_hd/lib/sky130_fd_sc_h
d__ss_100C_1v60.lib

7. Executing ABC pass (technology mapping using ABC).

7.1. Extracting gate netlist of module `\cmp_32' to `<abc-temp-dir>/input.blif'..
Extracted 646 gates and 712 wires to a netlist network with 64 inputs and 3 outputs.

7.1.1. Executing ABC.
Running ABC command: "<yosys-exe-dir>/yosys-abc" -s -f <abc-temp-dir>/abc.script 2>&1
ABC: ABC command line: "source <abc-temp-dir>/abc.script".
ABC:
ABC: + read_blif <abc-temp-dir>/input.blif
ABC: + read_lib -w /foss/pdks/volare/sky130/versions/0fe599b2afb6708d281543108caf8310912f54af/sky130A/libs.ref/sky130_fd_sc_hd/lib/sky130_fd_sc_hd
__ss_100C_1v60.lib
ABC: Parsing finished successfully.  Parsing time =     0.12 sec
ABC: Scl_LibertyReadGenlib() skipped cell "sky130_fd_sc_hd__decap_12" without logic function.
ABC: Scl_LibertyReadGenlib() skipped cell "sky130_fd_sc_hd__decap_3" without logic function.
ABC: Scl_LibertyReadGenlib() skipped cell "sky130_fd_sc_hd__decap_4" without logic function.
ABC: Scl_LibertyReadGenlib() skipped cell "sky130_fd_sc_hd__decap_6" without logic function.
ABC: Scl_LibertyReadGenlib() skipped cell "sky130_fd_sc_hd__decap_8" without logic function.
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfbbn_1".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfbbn_2".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfbbp_1".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfrbp_1".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfrbp_2".
ABC: Scl_LibertyReadGenlib() skipped sequential cell "sky130_fd_sc_hd__dfrtn_1".
```

```
ABC RESULTS:   sky130_fd_sc_hd__o21ai_0 cells:        3
ABC RESULTS:   sky130_fd_sc_hd__o21ba_1 cells:        1
ABC RESULTS:   sky130_fd_sc_hd__o221ai_1 cells:       1
ABC RESULTS:   sky130_fd_sc_hd__o22ai_1 cells:        5
ABC RESULTS:   sky130_fd_sc_hd__o311ai_0 cells:       1
ABC RESULTS:   sky130_fd_sc_hd__or3b_1 cells:         1
ABC RESULTS:   sky130_fd_sc_hd__xnor2_1 cells:        2
ABC RESULTS:   sky130_fd_sc_hd__xor2_1 cells:         8
ABC RESULTS:        internal signals:      645
ABC RESULTS:           input signals:       64
ABC RESULTS:          output signals:        3
Removing temp directory.
```

9. **Techmap Registers using PDK: dfflibmap -liberty <pdk>.lib**

   Same as abc, except for the registers.

   yosys> dfflibmap -liberty sky130_fd_sc_hd__tt_025C_1v80.lib

10. **Statistics: yosys> stat**

    Tells you the resource usage statistics of your synthesized model.

```
yosys> stat

8. Printing statistics.

=== cmp_32 ===

   Number of wires:                473
   Number of wire bits:           4221
   Number of public wires:           5
   Number of public wire bits:      67
   Number of ports:                  5
   Number of port bits:             67
   Number of memories:               0
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:                108
     sky130_fd_sc_hd__a211o_1        1
     sky130_fd_sc_hd__a211oi_1       4
     sky130_fd_sc_hd__a21o_1         1
     sky130_fd_sc_hd__a21oi_1        5
     sky130_fd_sc_hd__a221oi_1       1
     sky130_fd_sc_hd__a222oi_1       1
     sky130_fd_sc_hd__a22o_1         3
     sky130_fd_sc_hd__a2bb2oi_1      1
     sky130_fd_sc_hd__a31oi_1        2
     sky130_fd_sc_hd__and4_1         2
     sky130_fd_sc_hd__clkinv_1      23
     sky130_fd_sc_hd__lpflow_isobufsrc_1     11
     sky130_fd_sc_hd__maj3_1         4
     sky130_fd_sc_hd__nand2_1        2
     sky130_fd_sc_hd__nand2b_1      10
     sky130_fd_sc_hd__nand3_1        3
     sky130_fd_sc_hd__nand4bb_1      1
     sky130_fd_sc_hd__nor2_1         3
     sky130_fd_sc_hd__nor3_1         2
     sky130_fd_sc_hd__nor3b_1        2
     sky130_fd_sc_hd__nor4_1         1
     sky130_fd_sc_hd__nor4b_1        1
     sky130_fd_sc_hd__o2111ai_1      1
     sky130_fd_sc_hd__o211ai_1       1
     sky130_fd_sc_hd__o21ai_0        3
     sky130_fd_sc_hd__o21ba_1        1
     sky130_fd_sc_hd__o221ai_1       1
     sky130_fd_sc_hd__o22ai_1        5
     sky130_fd_sc_hd__o311ai_0       1
     sky130_fd_sc_hd__or3b_1         1
     sky130_fd_sc_hd__xnor2_1        2
     sky130_fd_sc_hd__xor2_1         8
```

11. **Output Synthesized HDL Netlist: yosys> write_verilog -sv design_synth.sv**

Note: Use the "-sv" flag for SystemVerilog

Outputs your synthesized netlist, where every instance has a unique number, regardless of type (wire, reg, module), and instances are PDK standard cells.

```
yosys> write_verilog -sv cmp_32_synth.sv
9. Executing Verilog backend.

9.1. Executing BMUXMAP pass.

9.2. Executing DEMUXMAP pass.
Dumping module `\cmp_32'.
```

```
/* Generated by Yosys 0.51 (git sha1 c4b519022, g++ 13.3.0-6ubuntu2~24.04 -fPIC -O3) */

(* top =  1  *)
(* src = "rtl/cmp_32.sv:3.1-28.10" *)
module cmp_32(A, B, GT, LT, EQ);
  (* src = "rtl/cmp_32.sv:0.0-0.0" *)
  wire _000_;
  (* src = "rtl/cmp_32.sv:0.0-0.0" *)
  wire _001_;
  (* src = "rtl/cmp_32.sv:4.18-4.19" *)
  wire _002_;
  (* src = "rtl/cmp_32.sv:4.18-4.19" *)
  wire _003_;
  (* src = "rtl/cmp_32.sv:4.18-4.19" *)
  wire _004_;
  (* src = "rtl/cmp_32.sv:4.18-4.19" *)
  wire _005_;
  (* src = "rtl/cmp_32.sv:4.18-4.19" *)
  wire _006_;
```

```
sky130_fd_sc_hd__xor2_1 _497_ (
  .A(_056_),
  .B(_024_),
  .X(_115_)
);
sky130_fd_sc_hd__lpflow_isobufsrc_1 _498_ (
  .A(_013_),
  .SLEEP(_045_),
  .X(_116_)
);
sky130_fd_sc_hd__nand2b_1 _499_ (
  .A_N(_002_),
  .B(_034_),
  .Y(_117_)
);
```

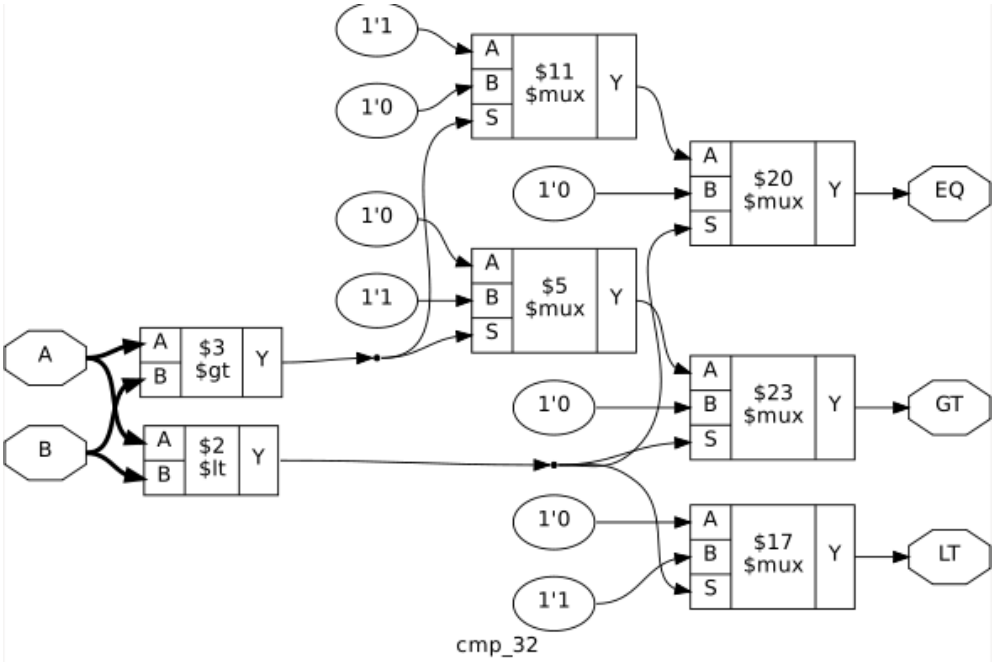
11.1 Visualize Schematic / Netlist Graph

To see the netlist, you must use the following command:

yosys> show -format dot -prefix design_rtl

*****make sure to have xdot installed (sudo apt install xdot)**

Open with: ubuntu@asic$ xdot design_rtl.dot



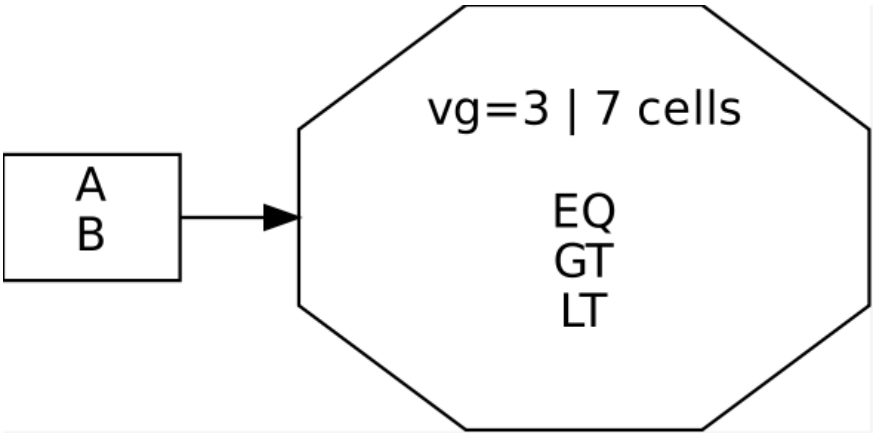
11.2 Visualize the Dataflow Graph

To see the dataflow, you must use the following command:

yosys> viz -format dot -prefix design_viz

*****make sure to have xdot installed (sudo apt install xdot)**

Open with: ubuntu@asic$ xdot design_viz.dot



11.3 Dump RTLIL

To see the AST after its been lowered into RTL, please use

yosys> write_rtlil design.il

cmp_32 > cmp_32.il

```
# Generated by Yosys 0.51 (git sha1 c4b519022, g++ 13.3.0-6ubuntu2~24.04 -fPIC -O3
autoidx 25
attribute \top 1
attribute \src "rtl/cmp_32.sv:3.1-28.10"
module \cmp_32
  attribute \src "rtl/cmp_32.sv:0.0-0.0"
  wire $2\EQ[0:0]
  attribute \src "rtl/cmp_32.sv:0.0-0.0"
  wire $2\GT[0:0]
  attribute \src "rtl/cmp_32.sv:19.17-19.22"
  wire $gt$rtl/cmp_32.sv:19$3_Y
  attribute \src "rtl/cmp_32.sv:16.12-16.17"
  wire $lt$rtl/cmp_32.sv:16$2_Y
  attribute \src "rtl/cmp_32.sv:4.18-4.19"
  wire width 32 input 1 \A
  attribute \src "rtl/cmp_32.sv:5.18-5.19"
  wire width 32 input 2 \B
  attribute \src "rtl/cmp_32.sv:8.18-8.20"
  wire output 5 \EQ
  attribute \src "rtl/cmp_32.sv:6.18-6.20"
  wire output 3 \GT
  attribute \src "rtl/cmp_32.sv:7.18-7.20"
  wire output 4 \LT
  attribute \src "rtl/cmp_32.sv:19.17-19.22"
  cell $gt $gt$rtl/cmp_32.sv:19$3
    parameter \A_SIGNED 0
    parameter \A_WIDTH 32
    parameter \B_SIGNED 0
    parameter \B_WIDTH 32
    parameter \Y_WIDTH 1
    connect \A \A
    connect \B \B
    connect \Y $gt$rtl/cmp_32.sv:19$3_Y
  end
  attribute \src "rtl/cmp_32.sv:16.12-16.17"
  cell $lt $lt$rtl/cmp_32.sv:16$2
    parameter \A_SIGNED 0
    parameter \A_WIDTH 32
```

12. Yosys Scripting

Instead of running the same commands over and over again, you can make a .ys script (such as synth.ys) with a sequence of commands and run it with:

ubuntu@asic:$ yosys -s synth.ys

Note: Yosys cannot create folders, so if you wish to save your results in a folder, it must exist before running the script or else the files will not be generated.

```
read_verilog -sv rtl/cmp_32.sv
hierarchy -top cmp_32
proc
opt
flatten
techmap
abc -liberty
/foss/pdks/volare/sky130/versions/0fe599b2afb6708d281543108caf8310912f54af
/sky130A/libs.ref/sky130_fd_sc_hd/lib/sky130_fd_sc_hd__ss_100C_1v40.lib
stat
show -format dot -prefix synth/cmp_32_graph
viz -format dot -prefix synth/cmp_32_viz
write_rtlil synth/cmp_32_rtl.il
write_verilog -sv synth/cmp_32_synth.sv
```

13. STA (Static Timing Analysis)

Yosys has a built-in sta command, but it's better to use full-fledged tools like OpenSTA. If you need to find critical path information quickly, you can use:

yosys> sta -liberty <pdk>.lib