New Open Source Design Verification Tools from YosysHQ

N. Engelhardt, C. Wolf

Yosys

Yosys – a swiss army knife for netlists

- Open Source project started in 2012 by Claire Wolf
  - Originally a synthesis tool for an academic CGRA
  - Grew in capabilities and language support
- Now a tool that can be applied in many different contexts, anytime you need to transform netlists
  - Used as "glue" between many third-party tools
  - And for architecture exploration
  - But also a fully-fledged synthesis tool, used e.g. in the OpenLANE ASIC flow and by some FPGA vendors

Yosys – input and output formats

- Input formats:
  - Verilog
  - JSON
  - Aiger
  - Blif
  - Liberty
  - VHDL (GHDL plugin)
- Commercial Edition adds:
  - SystemVerilog
  - SystemVerilog Assertions
  - VHDL

- Output formats
  - Verilog
  - JSON
  - Blif
  - EDIF
  - FIRRTL
  - Aiger
  - SMT2
  - BTOR2
  - C++ (simulation)
  - Truth table

Full List of Commands:
https://yosyshq.readthedocs.io/projects/yosys/en/latest/cmd_ref.html

Yosys – Transformations

- General structure of Yosys-based flows
  - Run commands to read and elaborate the design
  - Run coarse-grain optimization commands
  - (Optional: Map to a fine-grain representation and run fine-grain optimizations)
  - Run back-end command to write design to output file
- Creating custom functionality using existing passes
  - Yosys has a rich set of commands to
    - Elaborate, simplify, infer, synthesize, technology map, simulate, …
    - See https://yosyshq.readthedocs.io/projects/yosys/en/latest/cmd_ref.html
  - One can get very far with creative selections of design elements and combinations of passes
- Creating custom functionality using custom passes
  - Techmap rules (module substitution - verilog file with special names)
  - Plugins (C++) can add custom passes with the same API used by internal passes
  - Pattern Matcher Generator - find subgraphs and modify/replace them

Yosys-based Tools

SBY – formal property checking with Yosys

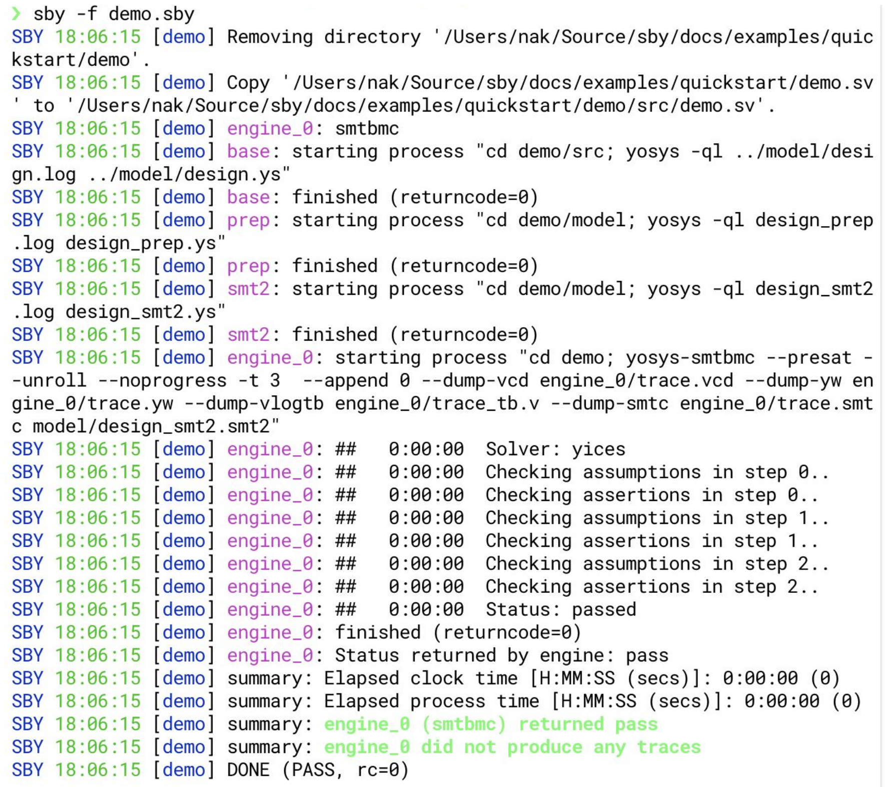
- Frontend for formal flows
  - Allows easy use of SystemVerilog `assume(), assert(), cover()` statements
    - Complex SVA properties/sequences are supported with the commercial version
  - SBY has modes for bounded and unbounded proofs
    - Support for different unbounded proof methods (k-induction, pdr/ic3)
- Automates the steps for running formal proofs with Yosys
  - Yosys translation of design to formal problem formats (SMT2, BTOR2, Aiger…)
  - Running solvers to find a set of signal values responding to the problem (or not)
    - Allows using many solvers being developed by researchers
  - Using Yosys to translate the set of variable assignments back into a VCD trace
- Myriad of different input/output formats “under the hood”
  - SBY provides a uniform interface for a wide range of solvers, hiding those differences.
- Example projects:
  - riscv-formal: formally verify ISA compliance (rv32imc/rv64imc) https://github.com/YosysHQ/riscv-formal/
  - AXI4 formal verification IP (requires SVA support) https://github.com/YosysHQ-GmbH/SVA-AXI4-FVIP

YosysHQ

SBY – formal property checking with Yosys

```verilog
module demo (
  input clk,
  output reg [5:0] counter
);
  initial counter = 0;

  always @(posedge clk) begin
    if (counter == 15)
      counter <= 0;
    else
      counter <= counter + 1'b1;
  end

`ifdef FORMAL
  always @(posedge clk) begin
    assert (counter < 32);
  end
`endif
endmodule
```

```
[options]
mode bmc
depth 3

[engines]
smtbmc

[script]
read -formal demo.sv
prep -top demo

[files]
demo.sv
```



```
❯ sby -f demo.sby
SBY 18:06:15 [demo] Removing directory '/Users/nak/Source/sby/docs/examples/quic
kstart/demo'.
SBY 18:06:15 [demo] Copy '/Users/nak/Source/sby/docs/examples/quickstart/demo.sv
' to '/Users/nak/Source/sby/docs/examples/quickstart/demo/src/demo.sv'.
SBY 18:06:15 [demo] engine_0: smtbmc
SBY 18:06:15 [demo] base: starting process "cd demo/src; yosys -ql ../model/desi
gn.log ../model/design.ys"
SBY 18:06:15 [demo] base: finished (returncode=0)
SBY 18:06:15 [demo] prep: starting process "cd demo/model; yosys -ql design_prep
.log design_prep.ys"
SBY 18:06:15 [demo] prep: finished (returncode=0)
SBY 18:06:15 [demo] smt2: starting process "cd demo/model; yosys -ql design_smt2
.log design_smt2.ys"
SBY 18:06:15 [demo] smt2: finished (returncode=0)
SBY 18:06:15 [demo] engine_0: starting process "cd demo; yosys-smtbmc --presat -
-unroll --noprogress -t 3  --append 0 --dump-vcd engine_0/trace.vcd --dump-yw en
gine_0/trace.yw --dump-vlogtb engine_0/trace_tb.v --dump-smtc engine_0/trace.smt
c model/design_smt2.smt2"
SBY 18:06:15 [demo] engine_0: ##   0:00:00  Solver: yices
SBY 18:06:15 [demo] engine_0: ##   0:00:00  Checking assumptions in step 0..
SBY 18:06:15 [demo] engine_0: ##   0:00:00  Checking assertions in step 0..
SBY 18:06:15 [demo] engine_0: ##   0:00:00  Checking assumptions in step 1..
SBY 18:06:15 [demo] engine_0: ##   0:00:00  Checking assertions in step 1..
SBY 18:06:15 [demo] engine_0: ##   0:00:00  Checking assumptions in step 2..
SBY 18:06:15 [demo] engine_0: ##   0:00:00  Checking assertions in step 2..
SBY 18:06:15 [demo] engine_0: ##   0:00:00  Status: passed
SBY 18:06:15 [demo] engine_0: finished (returncode=0)
SBY 18:06:15 [demo] engine_0: Status returned by engine: pass
SBY 18:06:15 [demo] summary: Elapsed clock time [H:MM:SS (secs)]: 0:00:00 (0)
SBY 18:06:15 [demo] summary: Elapsed process time [H:MM:SS (secs)]: 0:00:00 (0)
SBY 18:06:15 [demo] summary: engine_0 (smtbmc) returned pass
SBY 18:06:15 [demo] summary: engine_0 did not produce any traces
SBY 18:06:15 [demo] DONE (PASS, rc=0)
```

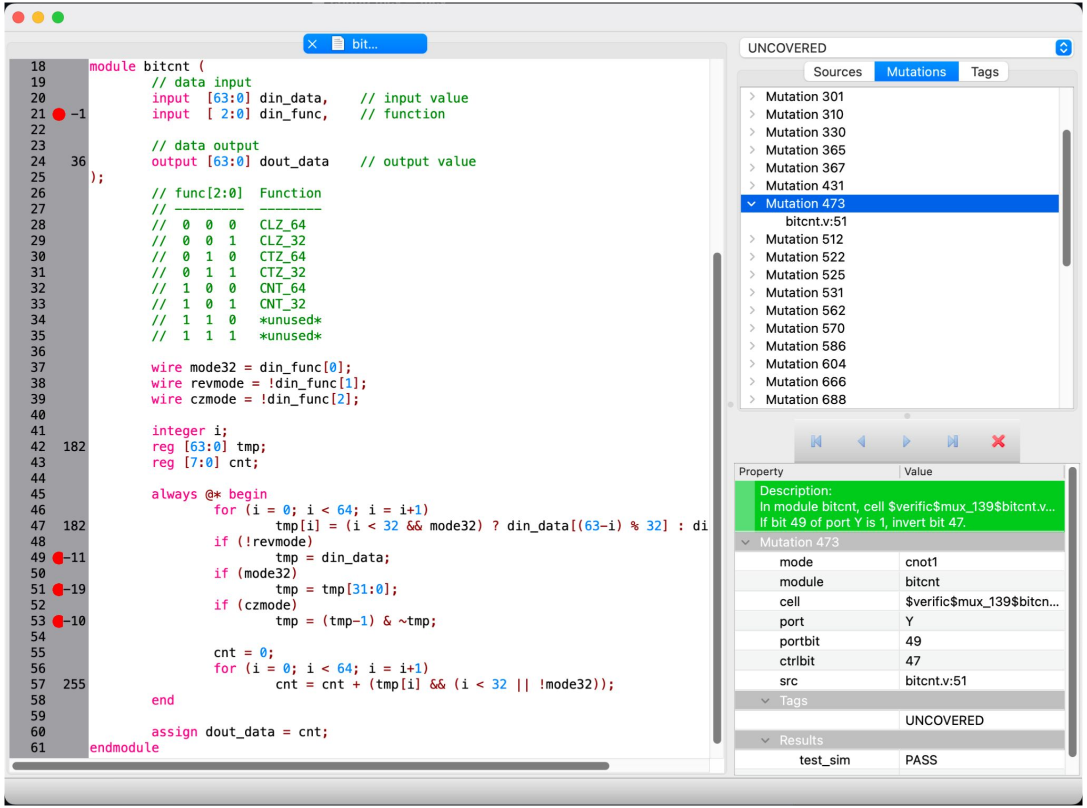
MCY – Mutation Coverage with Yosys

- Mutation coverage is a coverage metric for testbenches
  - Solves the issue of false negatives that is inherent to execution/branch coverage
  - Introduce modifications to the DUT and see if each modification causes the tests to fail
  - Yosys modifies the netlist and outputs a modified module to instantiate in the testbench
  - Works with any self-checking test environment that accepts a synthesized DUT
- Main Problem with Mutation coverage: False Positives
  - Some mutations don't violate the design spec, so it's fine for the test bench not to fail for them
- MCY Solution: Filter False Positives with formal equivalence checks
  - Create a miter circuit with mutated and non-mutated design, to let the formal method investigate the functional change introduced by such a mutation
  - Optional: Write properties taking into account when differences are relevant
    - e.g., only compare data if data_valid is high
    - Much easier than writing formal properties about the expected value of data
  - Mutations that create no relevant functional change are discarded automatically
  - Can run the formal checks in SBY, or interface with formal tools from other vendors

MCY – Mutation Coverage with Yosys

```
[options]
size 1000
tags COVERED UNCOVERED NOCHANGE EQGAP FMONLY

[script]
read -sv bitcnt.v
prep -top bitcnt

[files]
bitcnt.v

[logic]
use_formal = False

tb_okay = (result("test_sim") == "PASS")
eq_okay = (result("test_eq") == "PASS")

if tb_okay and use_formal:
    tb_okay = (result("test_fm") == "PASS")
    if not tb_okay:
        tag("FMONLY")

if tb_okay and not eq_okay:
    tag("UNCOVERED")
elif not tb_okay and not eq_okay:
    tag("COVERED")
elif tb_okay and eq_okay:
    tag("NOCHANGE")
elif not tb_okay and eq_okay:
    tag("EQGAP")
else:
    assert 0

[report]
if tags("EQGAP"):
    print("Found %d mutations exposing a formal equivalence gap!" % tags("EQGAP"))
if tags("COVERED")+tags("UNCOVERED"):
    print("Coverage: %.2f%%" % (100.0*tags("COVERED")/(tags("COVERED")+tags("UNCOVERED"))))

[test test_sim]
expect PASS FAIL
run bash $PRJDIR/test_sim.sh

[test test_eq]
expect PASS FAIL
run bash $PRJDIR/test_eq.sh

[test test_fm]
expect PASS FAIL
run bash $PRJDIR/test_fm.sh
```

```
❯ mcy status
Database contains 2000 cached results.
Database contains 938 cached "FAIL" results for "test_eq".
Database contains 62 cached "PASS" results for "test_eq".
Database contains 898 cached "FAIL" results for "test_sim".
Database contains 102 cached "PASS" results for "test_sim".
Tagged 898 mutations as "COVERED".
Tagged 62 mutations as "NOCHANGE".
Tagged 40 mutations as "UNCOVERED".
Coverage: 95.74%
```



```
18          module bitcnt (
19                  // data input
20                  input   [63:0] din_data,    // input value
21  ● -1            input   [ 2:0] din_func,    // function
22
23                  // data output
24     36           output [63:0] dout_data     // output value
25          );
26                  // func[2:0]  Function
27                  // ---------  --------
28                  //  0  0  0   CLZ_64
29                  //  0  0  1   CLZ_32
30                  //  0  1  0   CTZ_64
31                  //  0  1  1   CTZ_32
32                  //  1  0  0   CNT_64
33                  //  1  0  1   CNT_32
34                  //  1  1  0   *unused*
35                  //  1  1  1   *unused*
36
37                  wire mode32 = din_func[0];
38                  wire revmode = !din_func[1];
39                  wire czmode = !din_func[2];
40
41                  integer i;
42    182           reg [63:0] tmp;
43                  reg [7:0] cnt;
44
45                  always @* begin
46                          for (i = 0; i < 64; i = i+1)
47    182                           tmp[i] = (i < 32 && mode32) ? din_data[(63-i) % 32] : di
48                          if (!revmode)
49  ● -11                           tmp = din_data;
50                          if (mode32)
51  ● -19                           tmp = tmp[31:0];
52                          if (czmode)
53  ● -10                           tmp = (tmp-1) & ~tmp;
54
55                          cnt = 0;
56                          for (i = 0; i < 64; i = i+1)
57    255                           cnt = cnt + (tmp[i] && (i < 32 || !mode32));
58                  end
59
60                  assign dout_data = cnt;
61          endmodule
```

Mutation 301
Mutation 310
Mutation 330
Mutation 365
Mutation 367
Mutation 431
Mutation 473
    bitcnt.v:51
Mutation 512
Mutation 522
Mutation 525
Mutation 531
Mutation 562
Mutation 570
Mutation 586
Mutation 604
Mutation 666
Mutation 688

Property	Value
Description: In module bitcnt, cell $verific$mux_139$bitcnt.v... If bit 49 of port Y is 1, invert bit 47.	
Mutation 473	
mode	cnot1
module	bitcnt
cell	$verific$mux_139$bitcn...
port	Y
portbit	49
ctrlbit	47
src	bitcnt.v:51
Tags	
	UNCOVERED
Results	
test_sim	PASS

MCY for fault tolerance testing

- MCY test specification is generic - the same infrastructure for testbench tests and formal equivalence check
- The pass/fail logic is also customizable
- Mutations are always single-bit changes

⇒ Can apply the same tool to introduce mutations into fault-tolerant netlists and check that the output does not change after mutation

Demo: https://github.com/nakengelhardt/faultinjection_mcy

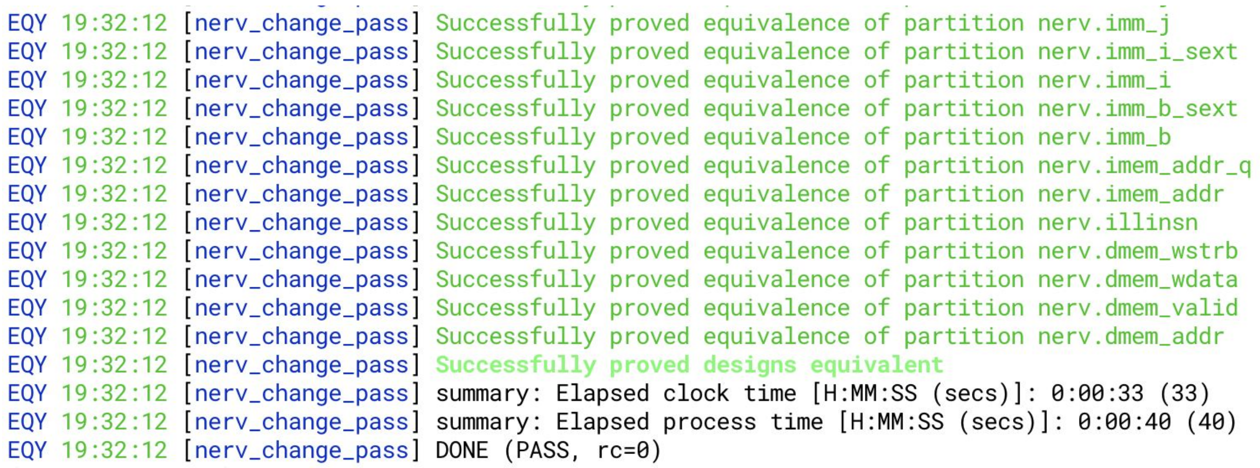
EQY – Equivalence Checking with Yosys

- Initial Release this Month: Our brand new Equivalence Checking Tool \o/
- Identifies matching points in two designs
  - Then partitions the design into smaller pieces that can be checked independently
  - Scales much better than using SBY on a miter circuit
  - Much easier to identify parts of design that cause scaling issues
- Application Domains
  - Ensure post-synthesis netlist is the same as input design
  - Check that a non-functional change does not change the behavior of the design
- Example/Tutorial Projects included with the tool
  - Verification of a design change in ALU/shifter architecture in NERV RISC-V Processor
  - Verification of Xilinx Vivado synthesis output for PicoRV32 processor design

EQY – Equivalence Checking with Yosys

```
// new shifter code
function [31:0] bitreverse(input [31:0] arg);
    for (integer i = 0; i < 32; i++) bitreverse[i] = arg[31-i];
endfunction
wire [32:0] shift_t1 = {insn_funct7[5] && rs1_value[31], insn_funct3 == 3'b001 ? bitreverse(rs1_value) : rs1_value};
wire [31:0] shift_t2 = $signed(shift_t1) >>> (insn_opcode == OPCODE_OP_IMM ? insn[24:20] : rs2_value[4:0]);
wire [31:0] shift_out = insn_funct3 == 3'b001 ? bitreverse(shift_t2) : shift_t2;
```

```
[gold]
read_verilog -sv nerv.sv
prep -top nerv
memory_map

[gate]
read_verilog -sv nerv_change.sv
prep -top nerv
memory_map

[collect *]
group regfile*
join imm_*
join insn*

[strategy sby]
use sby
depth 2
engine smtbmc bitwuzla
```



```
EQY 19:32:12 [nerv_change_pass] Successfully proved equivalence of partition nerv.imm_j
EQY 19:32:12 [nerv_change_pass] Successfully proved equivalence of partition nerv.imm_i_sext
EQY 19:32:12 [nerv_change_pass] Successfully proved equivalence of partition nerv.imm_i
EQY 19:32:12 [nerv_change_pass] Successfully proved equivalence of partition nerv.imm_b_sext
EQY 19:32:12 [nerv_change_pass] Successfully proved equivalence of partition nerv.imm_b
EQY 19:32:12 [nerv_change_pass] Successfully proved equivalence of partition nerv.imem_addr_q
EQY 19:32:12 [nerv_change_pass] Successfully proved equivalence of partition nerv.imem_addr
EQY 19:32:12 [nerv_change_pass] Successfully proved equivalence of partition nerv.illinsn
EQY 19:32:12 [nerv_change_pass] Successfully proved equivalence of partition nerv.dmem_wstrb
EQY 19:32:12 [nerv_change_pass] Successfully proved equivalence of partition nerv.dmem_wdata
EQY 19:32:12 [nerv_change_pass] Successfully proved equivalence of partition nerv.dmem_valid
EQY 19:32:12 [nerv_change_pass] Successfully proved equivalence of partition nerv.dmem_addr
EQY 19:32:12 [nerv_change_pass] Successfully proved designs equivalent
EQY 19:32:12 [nerv_change_pass] summary: Elapsed clock time [H:MM:SS (secs)]: 0:00:33 (33)
EQY 19:32:12 [nerv_change_pass] summary: Elapsed process time [H:MM:SS (secs)]: 0:00:40 (40)
EQY 19:32:12 [nerv_change_pass] DONE (PASS, rc=0)
```

SCY – Sequence of Covers with Yosys

- Sneak peek at our next development:

  A formal methodology and tool for generating long cover traces for large designs, based on “checkpoint” cover properties, that the tool eagerly solves one-by-one, using the final state of one property as the initial state of the next.

- Example Applications:
  - Creating formal cover traces for complex bus interactions on large SoC designs.
  - Using formal tools to create a assembler programs to put a processor in difficult to reach states of its state-space

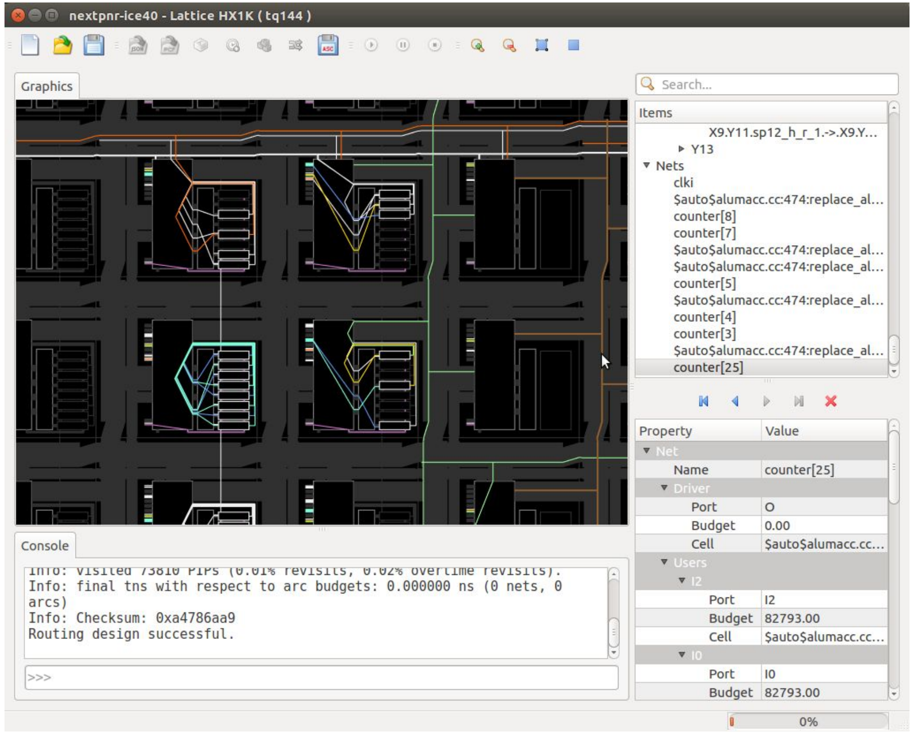
Data-Flow Properties

- With SCY we will also introduce a methodology for formal data-flow properties, for example:
  - Cover a trace that shows `top.Bus.ComponentA.DOUT_VALID`and `top.Bus.ComponentA.DOUT_READY`active in cycle t_1,
  - and `top.Bus.ComponentB.DIN_VALID`and `top.Bus.ComponentB.DIN_READY`active in cycle t_2,
  - and `top.Bus.ComponentB.DIN_DATA`at t_2 is a function of `top.Bus.ComponentA.DOUT_DATA`at t_1.
- Where “is a function of” means we can show data-flow (of a configurable kind) from the input to the output.
- This functionality is especially useful for the kind of properties we are building SCY for, but will be made available in all our formal flows.

nextpnr

nextpnr – Open Source P&R

- API-driven cross-platform P&R tool
  - Each FPGA family can choose different internal data structures as suits the project
- Currently supported Architectures:
  - Lattice iCE40, ECP5, Nexus
  - Gowin LittleBee
  - Experimental: Cyclone V, MachXO2
- https://github.com/YosysHQ/nextpnr


nextpnr-ice40 - Lattice HX1K (tq144)
Graphics
Console
Search...
Items
X9.Y11.sp12_h_r_1.->.X9.Y...
Y13
Nets
clki
$auto$alumacc.cc:474:replace_al...
counter[8]
counter[7]
$auto$alumacc.cc:474:replace_al...
$auto$alumacc.cc:474:replace_al...
counter[5]
$auto$alumacc.cc:474:replace_al...
counter[4]
counter[3]
$auto$alumacc.cc:474:replace_al...
counter[25]
0%

```
Info: visited 73810 PIPs (0.01% revisits, 0.02% overtime revisits).
Info: final tns with respect to arc budgets: 0.000000 ns (0 nets, 0
arcs)
Info: Checksum: 0xa4786aa9
Routing design successful.
>>>
```

Property	Value
Net	
Name	counter[25]
Driver	
Port	O
Budget	0.00
Cell	$auto$alumacc.cc...
Users	
I2	
Port	I2
Budget	82793.00
Cell	$auto$alumacc.cc...
I0	
Port	I0
Budget	82793.00

Get the Tools

Try it out!

- Download nightly builds of the OSS CAD Suite
  - https://github.com/YosysHQ/oss-cad-suite-build/releases/latest
  - Includes Yosys, SBY, MCY, all dependencies, supported solvers, GHDL plugin (linux only)
  - Also nextpnr, Amaranth, cocotb, …
- Documentation: https://yosyshq.readthedocs.io/en/latest/
- Ask me for an evaluation license to the commercial Tabby CAD Suite
  - Email contact@yosyshq.com or tick a box on https://www.yosyshq.com/contact
- Or try our "formal taster" package
  - 2 hours of tailored video support to help you get started
  - Save 33% - just 1800 Euros for 3 months.

YosysHQ

Q&A