Open Source HDL Synthesis and Verification with Yosys

Clifford Wolf
Abstract

Yosys (Yosys Open Synthesis Suite) is an open source project aiming at creating a fully-featured HDL synthesis tool, and more. Lately a lot of features related to formal verification have been added to Yosys.

Project IceStorm aims at documenting the bit-stream format of Lattice iCE40 FPGAs and providing simple tools for analyzing and creating bit-stream files, including a tool that converts iCE40 bit-stream files into Verilog.

Arachne-PNR is a place&route tool based on the databases provided by Project IceStorm. It converts BLIF files into an ASCII file format that can be turned into a bit-stream by IceStorm tools.

This three projects together implement a complete open source tool-chain for iCE40 FPGAs. It is available now and it is feature complete (with the exception of timing analysis, which is work in progress).

This presentation covers the open source Yosys-IceStorm-Arachne iCE40 flow as well as some other synthesis and verification applications based on Yosys.
Overview

- **Verilog Synthesis** with Yosys
- Project IceStorm: **Reverse-Engineered iCE40 FPGA Bitstream**
- Arachne-PNR: **Open Source Place&Route** for iCE40 (and maybe others in the future)
- **Demo:** **PicoRV32** CPU on iCE40 HX8K FPGA
- Yosys as **formal verification** tool
Quick Intro to Yosys

- Yosys is controlled by synthesis scripts. For example:
  
  ```
  read_verilog top.v
  read_verilog uart.v
  synth_xilinx -top top -edif top.edif
  ```

- Commands like `synth_xilinx` are placeholders for larger scripts. See help `synth_xilinx`:
  
  ```
  http://www.clifford.at/yosys/cmd_synth_xilinx.html
  ```
Synthesis is a reduction of abstraction ...

... by performing a sequence of transformations →
Supported Verilog HDL Constructs

- Almost all of Verilog 2005, for example...
  - Memories (incl. FPGA block-ram mapping)
  - Verilog tasks and functions
  - Real Arithmetic in constant expressions
  - A few things from SystemVerilog (e.g. asserts)
  - Initialized registers (FPGAs, Formal Checks)
  - Various Verilog attributes, e.g. for..
    - FSM encoding schemes
    - full_case and parallel_case
  - DPI-C in constant expressions
  - Behavioral modeling
  - $display and $finish in initial blocks

Open Source CPUs that I have tested with Yosys:
- OpenMSP430
- Amber ARMv2 Clone
- Navré AVR Clone
- OpenRISC 1200
- Rocket (default config)
- PicoRV32
Yosys Front-ends

- Native Verilog 2005 front-end

- Additional native front-ends for:
  - BLIF
  - Liberty File Format
  - Yosys' native ILANG format

- Bindings to commercial Verific library
  - VHDL, Verilog, SystemVerilog
Yosys Back-ends

- Various netlist formats:
  - BLIF, EDIF, InterSynth, Spice, Verilog

- Formats for formal verification:
  - SMT2, SMV, BTOR

- Other back-ends:
  - Yosys' internal ILANG format
  - JSON back-end, 'cause we can
Yosys' Internal Netlist Representation

- The in-memory netlist format used by Yosys is called RTLIL. It is a set of simple C++ structs.
- See `kernel/rtlil.h` in Yosys source code.
- ILANG is a 1:1 text representation of RTLIL.
- Simplified ER diagram:
Getting started with Yosys C++ API

• Recommended reading:
  – The CodingReadme file in the source tree
  – The “Yosys Manual” and “Yosys Presentation”

• Get help and ask questions online:
  – On Reddit: /r/yosys
  – On StackOverflow: yosys tag

• Warning: Correct usage of RTLIL::SigSpec, RTLIL::SigBit, RTLIL::SigChunk, and SigMap(module) is difficult to grasp at first. Ask questions when unsure!
Example ASIC Synthesis Script

Read Verilog design mytop, transform it to a generic gate-level netlist and then map to a Liberty cell library, finally write technology netlist to an EDIF file:

```
# read design
read_verilog mydesign.v

# generic synthesis
synth -top mytop

# mapping to mycells.lib
dfflibmap -liberty mycells.lib
abc -liberty mycells.lib
opt_clean

# write synthesized design
write_edif synth.edif
```

Open Source ASIC Flows using Yosys:
Example iCE40 Flow

Synthesis script for PicoRV32 iCE40/IceStorm example (scripts/icestorm/):

1. **Synthesis** (Yosys):
   
yosys -p 'synth_ice40 -top top -blif synth.blif' picorv32.v top.v

2. **Place and Route** (Arachne-PNR):
   
arachne-pnr -d 8k -o synth.txt synth.blif

3. **Create bit-stream** (IceStorm):
   
   icepack synth.txt synth.bin

4. **Upload bit-stream** (IceStorm):
   
   iceprog synth.bin
Project IceStorm

- Reverse-engineered documentation of iCE40 FPGAs bit-stream format.  
  http://www.clifford.at/icestorm/

- Also a few useful tools:
  - `icebox_{vlog,explain,chipdb,...}`
    - Various utilities for analyzing iCE40 bitstreams
  - `icepack / iceunpack`
    - iCE40 Bitstream ⇔ IceStorm ASCII format
  - `iceprog`
    - Programming (icestick, hx8k break-out board, etc.)
  - `icemulti`
    - Packing bitstreams into iCE40 multiboot images
Project IceStorm – History

Mathias Lasser (mostly 2014):
- Reverse-engineered the low-level bitstream format (grouping of bits into tiles, etc)
- Created original iceunpack tool
- Also did some early work on reverse-engineering the iCE40 interconnect

Clifford Wolf (mostly 2015):
- Wrote icefuzz and icebox
- Complete reverse-engineering of all iCE40 tile types (IO, LOGIC, RAMB/RAMT)
- Written documentation on IceStorm web-page
Arachne-PNR

• A place-and-route tool for iCE40 FPGAs written by Cotton Seed.
  https://github.com/cseed/arachne-pnr

• Using the chipdb-* files created by icebox_chipdb.py

• Input format:
  – BLIF files generated by Yosys (using non-standard .param statements for cell parameters)

• Output Format:
  – IceStorm ASCII files as understood by icepack
Yosys synthesis for other proprietary FPGA architectures

- Yosys Xilinx Flow:
  - Yosys has support for Xilinx 7-series synthesis
  - See help synth_xilinx for details
  - Output: EDIF netlist
  - Place and route: Xilinx Vivado

- Yosys Silego GreenPak4 Flow:
  - Going to support synthesis for Silego GreenPak4 Mixed-Signal Matrices
  - Complete documentation available, up to 25 LUTs for logic
  - Support tool-chain by Andrew Zonenberg
  - This is work in progress
ICE40 Demo: PicoRV32 CPU

- **PicoRV32** is a **CPU core** implementing the **RISC-V ISA (RV32I)**
- Optimized for small size, easy integration, and high clock rate, but not high performance
- Optional co-processor interface (incl. example core implementing MUL [H|SU|U] from RV32M)
- 0.309 DMIPS/MHz (4.167 CPI)

- On Xilinx 7-series FPGAs (using Vivado):
  ~1000 LUTs (6-input), up to 476 MHz (Virtex-7T, Speedgrade -3)

- On iCE40 HX8K:
  1521 LUTs (4-input) using Yosys
  1320 LUTs (4-input) using Synplify Pro
  8619 LUTs (4-input) using Lattice Synthesis Engine (LSE)

https://github.com/cliffordwolf/picorv32
IcoBoard – Open Hardware

iCE40 Raspberry Pi Hat

- Applications
  - Raspberry Pi IO Expander
  - Intelligent IO Cores
  - On-demand HDL generation
  - Education

- Hardware
  - iCE40 HX8K FPGA
  - 4 PMOD connectors
  - +16 PMODs on Ico-X Board
  - SPI + GPIOs to Raspberry Pi

- Software
  - FPGA SRAM programming tool (bitstream upload)
  - Raspberry Pi port of the entire Yosys / Arachne-PNR / IceStorm tool-chain
  - Python library + FPGA framework for communication with FPGA cores

http://icoboard.org/
Formal Verification with Yosys

- **SAT solving** (built-in MiniSAT-based eager SMT solver, see `help sat`)
- Built-in **equivalence checking** framework (see `help equiv_*`)
- Creating **miter circuits** for equivalence or **property checking** (Verilog `assert`)
  - Either solve with built-in solver or
  - Export as BLIF and solve with e.g. ABC
- Creating **SMT-LIB 2.5** models for circuits and properties that can be used with external SMT solvers
- **Writing Yosys back-ends is easy!** Some future Ideas:
  - Languages like HOL4 or Haskell (on-demand – contact me!)
  - C back-end to be used with something like LLBMC
Verilog asserts

- Yosys supports **SystemVerilog asserts**
  
  In module context and always blocks:
  ```
  module example_assert(A);
  signed input [31:0] A;
  signed wire  [31:0] B;
  assign B = A < 0 ? -A : A;
  assert property (B >= 0);
  endmodule
  ```

- There is also support for the **assume**
  ```
  module example_assert(A);
  signed input [31:0] A;
  signed wire  [31:0] B;
  assign B = A < 0 ? -A : A;
  assume property (B >= 0);
  endmodule
  ```

Example:

```
$ yosys example_assert.ys
...
Solving problem with 845 variables and 2305 clauses.
SAT proof finished - model found: FAIL!
```

```
read_verilog -sv example_assert.v
hierarchy -top example_assert
proc; opt -keepdc
sat -prove-asserts -show-inputs
```

```
<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Dec</th>
<th>Hex</th>
<th>Bin</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbackslash{}A</td>
<td>-2147483648</td>
<td>80000000</td>
<td>10000000000000000000000000000000000000000</td>
</tr>
</tbody>
</table>
```
Miter circuits

- Some tools (e.g. ABC) operate on miter circuits.
  - Miters are circuits with a single output
  - that is asserted when the property is violated

- The miter command can be used to create miters
  - For properties using assert and/or assume statements
  - For equivalence of two circuits

```verilog
module main(input clk, output [63:0] state);
    reg [63:0] state = 123456789;

    function [63:0] xorshift64star;
    input [63:0] current_state;
    begin
        xorshift64star = current_state;
        xorshift64star = xorshift64star ^ (xorshift64star >> 12);
        xorshift64star = xorshift64star ^ (xorshift64star << 25);
        xorshift64star = xorshift64star ^ (xorshift64star >> 27);
        xorshift64star = xorshift64star * 64'd 2685821657736338717;
    end
    endfunction

    always @(posedge clk)
        state <= xorshift64star(state);
    assert property (state != 0);
endmodule
```

```
read_verilog -sv example_miter.v hierarchy; proc; opt; memory; opt miter -assert main; techmap; opt
write_blif example_miter.blif

$ yosys-abc -c 'read_blif example_miter.blif; strash; pdr'
...
Invariant F[3] : 1 clauses with 64 flops (out of 64)
Verification of invariant with 1 clauses was successful. Time = 0.01 sec
Property proved. Time = 0.28 sec
```
SAT solving

- The Yosys sat command provides access to the built-in SAT solver framework
  - Based on MiniSAT SimpSolver
  - Essentially an eager SMT solver

- Bounded Model Checking (BMC):
  - sat -seq 50 -prove-asserts -set-assumes

- Temporal Induction Proofs:
  - sat -tempinduct -prove never_one 0

- Writing traces as VCD files:
  - sat ... -dump_vcd <vcd_filename> ...

- Writing SAT problem in DIMACS format:
  - sat ... -dump_cnf <dimacs_filename> ...

- Interactive Troubleshooting:  
  - sat -seq 15 -set foo 23 -set-at 10 never_one 1 -show bar
Equivalence Checking

- The `equiv_*` commands in Yosys are for equivalence checking.
- This equivalence checker uses hints like net names to partition the circuits into to-be-proved equivalent subcircuits.
- This is extremely helpful for troubleshooting Yosys passes and/or perform pre-vs-post synthesis verification.
- The prover is capable of considering multiple time-steps (`equiv_simple -seq N`) and even perform temporal induction (`equiv_induct`).

```
module gold(input A, B, output Y);
    wire T = -A;
    assign Y = T + B;
endmodule

module gate(input A, B, output Y);
    wire T = --A;
    assign Y = T - B;
endmodule
```

```
equiv_make gold gate equiv hierarchy -top equiv opt -purge; show equiv_simple equiv_status -assert
```

```
Found 2 $equiv cells in equiv:
    Of those cells 2 are proven and 0 are unproven.
    Equivalence successfully proven!
```
SMT-LIB Language

- **SMT-LIB** is the language used in the annual SMT competition.
  - Therefore practically all SMT solvers do support SMT-LIB as input language.
  - SMT-LIB does support incremental problems and many different theories.
  - Most Yosys+SMT-LIB flows are using the QF_AUFBV theory.

- SMT-LIB is designed so that solvers can be “remote-controlled”
  - SMT-LIB is used to communicate with the solver via stdin and stdout
  - A Python library for writing proofs based on Yosys SMT-LIB output is provided
  - This way there is no lock-in on a single SMT solver

- A few SMT-Solvers worth looking at:
  - Z3, CVC4, Yices, MathSAT
module main(input clk, input [3:0] addr, input [7:0] data);
    reg [7:0] memory [15:0]; // zero-initialized in SMT2 template
    
    always @(posedge clk) begin
        assume(data[0] == 0);
        assert(memory[addr][1:0] == 0);
        memory[addr] <= memory[addr] + data*data;
    end
endmodule

read_verilog -formal example_smtlib.v
proc; opt; memory -nomap -nordff; opt
write_smt2 -bv -mem -tpl example_smtlib.tpl example_smtlib.smt2

$ yosys -q example_smtlib.ys
$ z3 -smt2 example_smtlib.smt2
unsat
(set-logic QF_AUFBV)

; declare 21 states
(declare-fun s00 () main_s)
...
(declare-fun s20 () main_s)

; declare 20 state transitions
(assert (main_t s00 s01))
...
(assert (main_t s19 s20))

; s00 is the init state
(assert (main_i s00))

; zero-initialize memory
(assert (= (select (|main_m memory| s00)
    #b0000) #b00000000))
...
(assert (= (select (|main_m memory| s00)
    #b1111) #b00000000))

; assumptions hold in all states
(assert (main_u s00))
...
(assert (main_u s20))

; we are looking for a case with
; violated assertions
(assert (or
    (not (main_a s00))
    ...

; is there such a model?
(check-sat)

(Template file for write_smt -tpl)
SMT-LIB BMC in PicoRV32

- The directory `scripts/smt2-bmc/` in the PicoRV32 sources contains two BMC tests.
- Both tests compare two instances of PicoRV32 in the different configurations.
- Both cores start out with the same memory and register file and with nreset=0.

- The `sync.sh` test (two cores with different ISA):
  - Assumption: Memory requests are synchronized
  - Assumption: The core with smaller ISA never traps
  - Assert: The core with larger ISA never traps
  - Assert: Final register file and memory content are identical
  - About 13 cycles in 10 minutes (with Yices 2.4.0)

- The `async.sh` test (two cores with same ISA):
  - Assumption: At the end of the trace both cores are in trap state
  - Assert: Final register file and memory content are identical
  - About 11 cycles in 15 minutes (with Yices 2.4.0)

- Python library for writing SMT proofs: `scripts/smt2-bmc/smtio.py`
module quine;
parameter t = {"integer i; initial begin $display("module quine;");
","$write("parameter t = {""); for (i = 292*8-1; i > 0; i = i-8)"
","case (t[i-8]) "\n": $write("\\n\",""); "\": $write("\\\\"); default: $write("%s", t[i-8]); endcase
","$display("\"};"); $display("%s", t); end endmodule