

A Free and Open Source Verilog-to-Bitstream Flow for iCE40 FPGAs

Yosys • Arachne-pnr • Project IceStorm

Clifford Wolf

Overview

Project IceStorm

- Tools and Documentation for the Lattice iCE40 FPGA Bitstream Formats (currently supported: HX1K, HX8K)

Yosys

- A Verilog Synthesis Suite
- For FPGAs and ASICs
- Formal Verification

+ Extra Slides on
Formal Verification

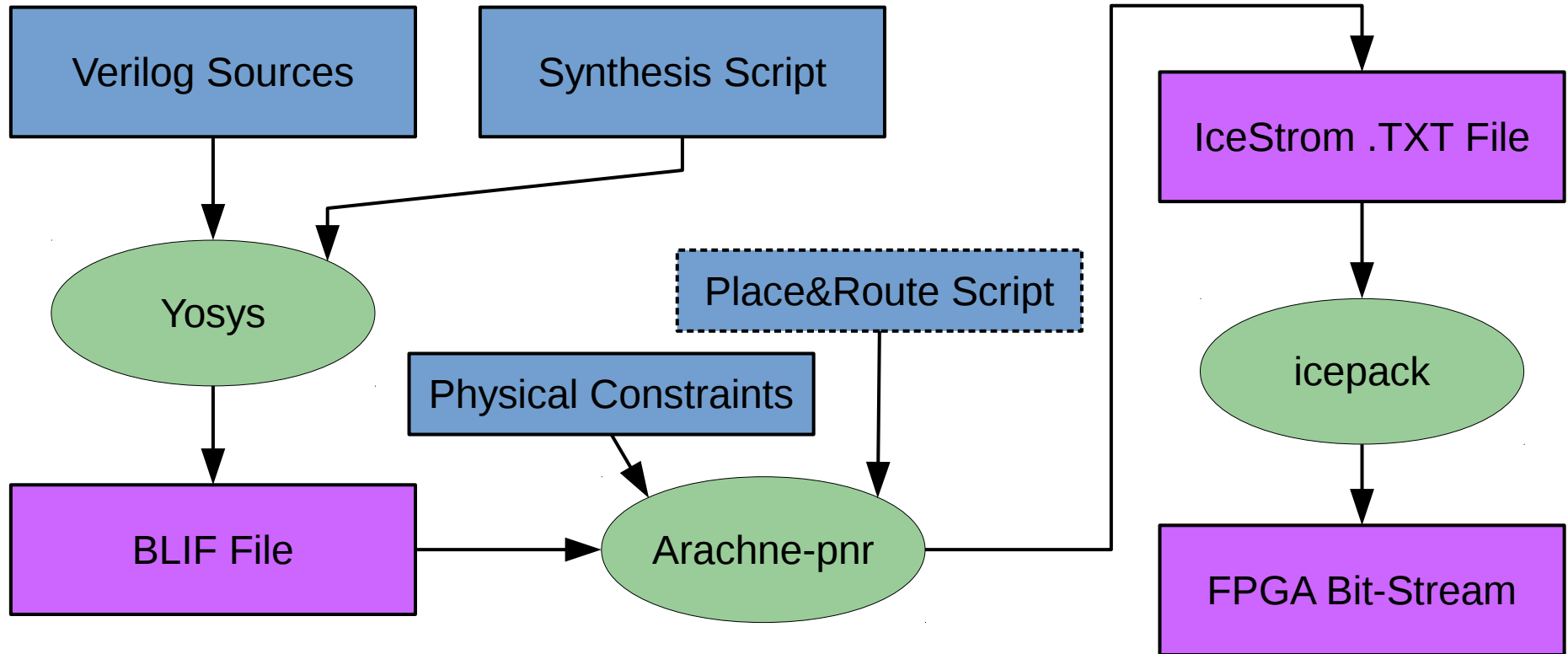
Arachne-pnr

- An FPGA Place-and-Route tool for iCE40 FPGAs
- Based on IceStorm Docs

IcoBoard ~~(Demo)~~

- A Raspberry PI HAT
- Lattice HX8K FPGA
- Up to 20 PMODs for IO
(= about 200 IO pins)

The IceStorm Flow

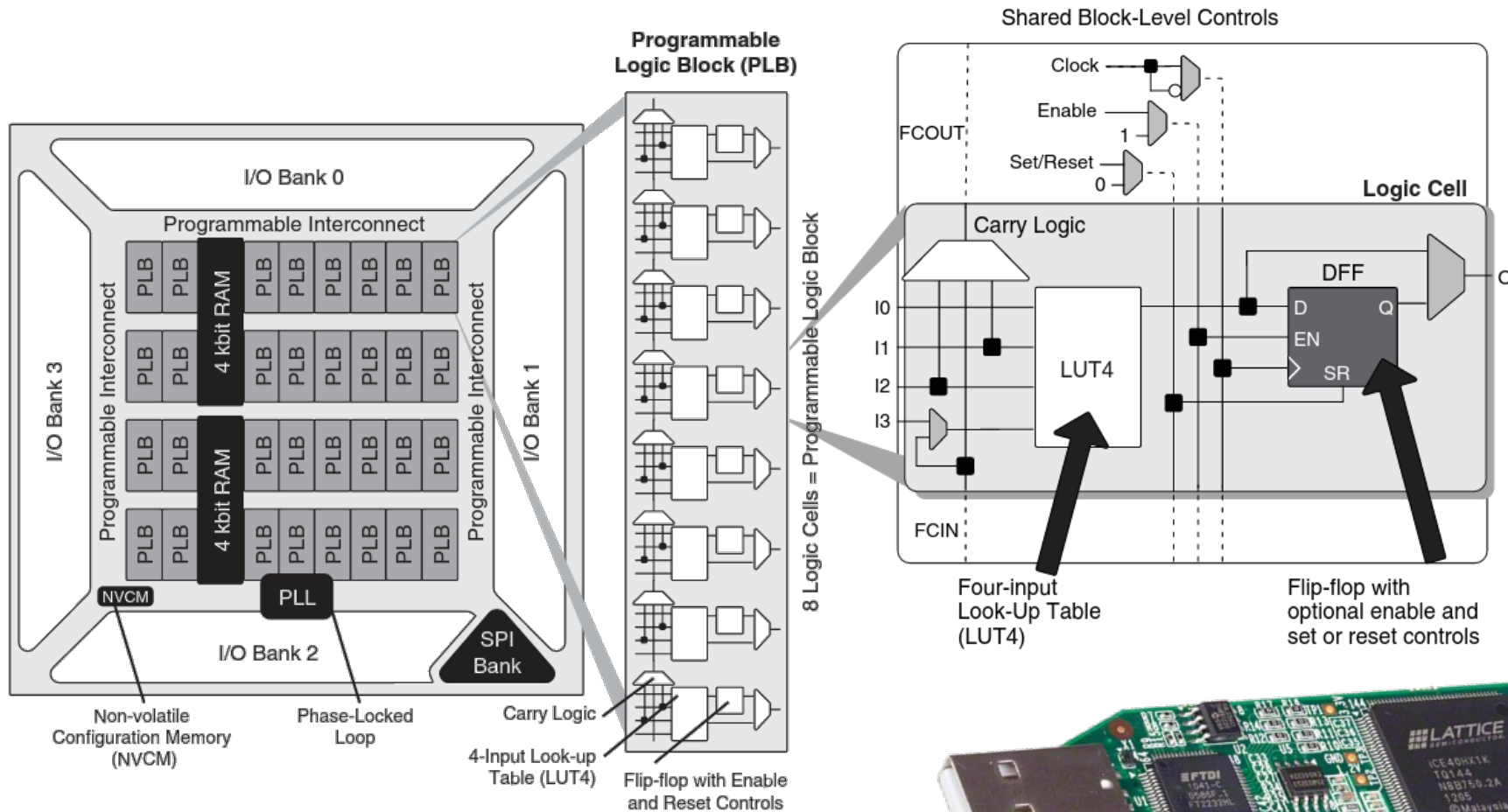


Part 1 of 4:

Project IceStorm

Lattice iCE40 Overview

- Family of **small FPGAs** (up to 7680 4-input LUTs in HX8K)
- Grid of tiles, the following tile types exist:
 - **Logic Tiles**: 8x 4-input LUT with optional FF and carry logic
 - **RAM Tiles**: Each RAMB/RAMT pair implements a 4 kbit SRAM
 - **IO Tiles**: Each IO tile connects to two PIO pins and has one fabric pin that connects to other external blocks (**PLLs**, **global nets**, etc.)
- Available in reasonable packages (e.g. HX1K in 144-pin TQFP)
- Cheapest dev board (Lattice iCEstick) costs under 25 \$.



from iCE40 LP/HX Family Data Sheet



Lattice iCEstick

Programmable Logic IC Development Tools iCE40-HX1K iCEstick Eval Board

Part: ICE40HX1K-STICK-EVN
Category: Programmable Logic IC Development Tools
Stock: No
On Order: Yes
Factory Lead-Time: 21 weeks



Pricing

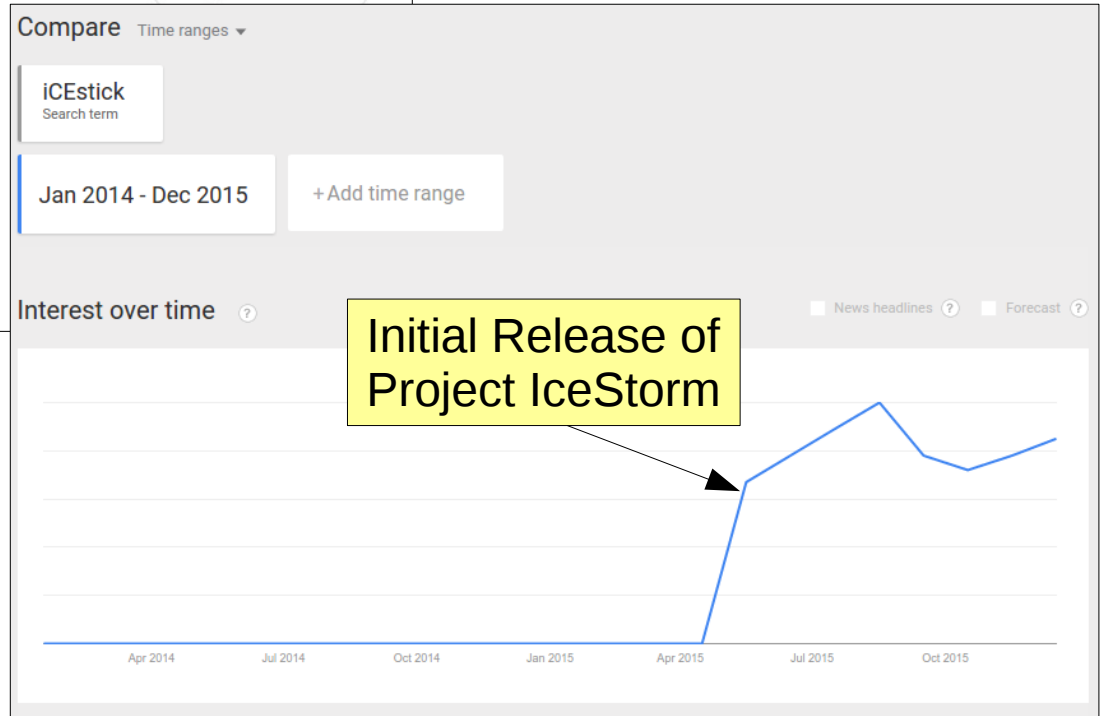
1: \$21.86

Add To Cart

Min: 1 Mult: 1

<http://www.latticestore.com/>
(13th January 2016)

.. Factory Lead-Time used to be 11 weeks
in first week of January.



<https://www.google.com/trends/explore#q=iCEstick>
via @Sebastian_Boee

Project IceStorm

- [illegible]

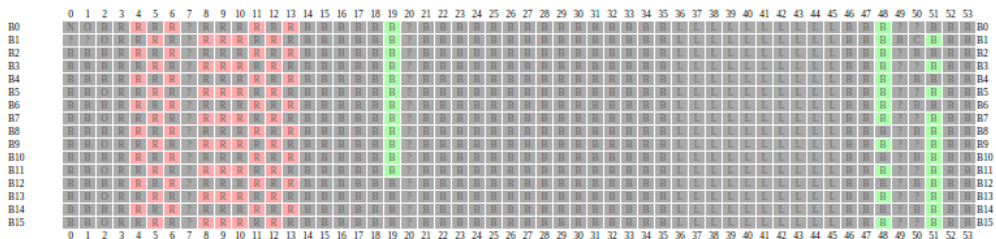
[illegible]

IceStorm Documentation

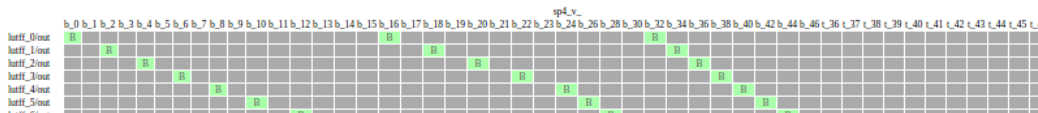
- It's all on teh Interwebs: <http://www.clifford.at/icestorm/>
- **Prerequisites: Basic understanding of how FPGAs work internally.**
This is a reference and not an introductory textbook!
- Also: It's not very well structured, so simply read it all. **It's only a few pages.**
For example: Most of the interconnect is explained in the section on LOGIC Tiles, global nets and PLLs are covered in the section on IO Tiles..
- There are three parts to the documentation:
 - 1) The **written documentation** on how the interconnect and the function blocks work in principle.
 - 2) An **auto-generated HTML reference** for all the configuration bits.
 - 3) An **auto-generated ASCII database** file that can be used by tools to generate or process FPGA bit-streams. Arachne-pnr for example is using this database files.

Some screenshots from IceStrom Docs:

Span-4 Vertical



Connectivity Matrix



Glb Net	0	1	2	3	4	5	6	7
IO Tile	7 0	7 17	13 9	0 9	6 17	6 0	0 8	13 8

Column Buffer Control Bits

Each LOGIC, IO, and RAMB tile has 8 ColBufCtrl bits, one for each global net. In most tiles this bits have no function, but in tiles in rows 4, 5, 12, and 13 (for RAM columns: rows 3, 5, 11, and 13) this bits control which global nets are driven to the column of tiles below and/or above that tile (including that tile), as illustrated in the image to the right (click to enlarge).

In 8k chips the rows 8, 9, 24, and 25 contain the column buffers. 8k RAMB and RAMT tiles can control column buffers, so the pattern looks the same for RAM, LOGIC, and IO columns.

Warmboot

The SB_WARMBOOT primitive in iCE40 FPGAs has three inputs and no outputs. The three inputs of that cell are driven by the fabout signal from three IO tiles. In HX1K chips the tiles connected to the SB_WARMBOOT primitive are:

Warmboot Pin	IO Tile
BOOT	12 0
S0	13 1
S1	13 2

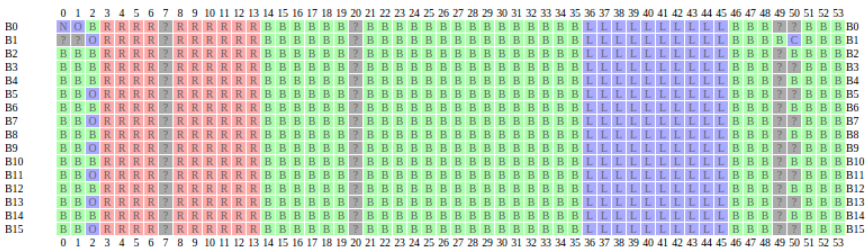
PLL Cores

The PLL primitives in iCE40 FPGAs are configured using the PLLCONFIG_* bits in the IO tiles. The configuration for a single PLL cell is spread out over many IO tiles. For example, the PLL cell in the 1K chip is configured as follows (bits

(2 17)	(3 17)	(4 17)	(5 17)	(6 17)
LOGIC Tile (2 16)	RAMT Tile (3 16)	LOGIC Tile (4 16)	LOGIC Tile (5 16)	LOGIC Tile (6 16)
LOGIC Tile (2 15)	RAMB Tile (3 15)	LOGIC Tile (4 15)	LOGIC Tile (5 15)	LOGIC Tile (6 15)
LOGIC Tile (2 14)	RAMT Tile (3 14)	LOGIC Tile (4 14)	LOGIC Tile (5 14)	LOGIC Tile (6 14)

Configuration Bitmap

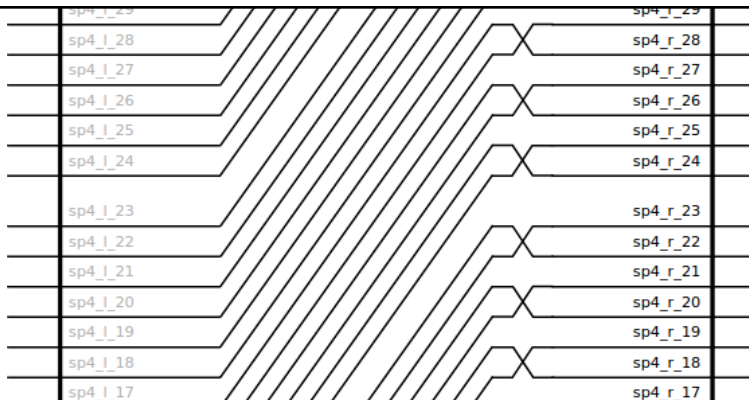
A LOGIC Tile has 864 config bits in 16 groups of 54 bits each:
B0[53:0], B1[53:0], B2[53:0], B3[53:0], B4[53:0], B5[53:0], B6[53:0], B7[53:0],
B8[53:0], B9[53:0], B10[53:0], B11[53:0], B12[53:0], B13[53:0], B14[53:0], B15[53:0]



Nets and Connectivity

This section lists all nets in the tile and how this nets are connected with nets from cells in its neighbourhood.

List of nets in LOGIC Tile (4 16)



Part 2 of 4:

Arachne-pnr

Arachne-pnr

- Arachne-pnr is a place and route tool for iCE40 FPGAs
- Input format: **BLIF Netlist** (Berkeley Logic Interchange Format)
- Output format: **TXT FPGA Config** (IceStorm ASCII Format)
- Performs the following operations (optionally controlled by script):
 - Instantiate IO Cells and Global Clock Buffers
 - Pack LUT, CARRY, FF Instances into iCE40 Logic Cells
 - Place Design (currently only simulated annealing)
 - Route Design
 - Generate FPGA Config

Input Netlist Format

- Cell types compatible with Lattice **iCE40 Technology Library**

SB_IO, SB_GB_IO, SB_GB, SB_LUT4, SB_DFF*,
SB_RAM40_4K*, SB_PLL40*, SB_WARMBOOT

- Using **BLIF** as easy-to-parse **netlist format**
 - Some non-standard extensions for parameters and attributes
 - Simple example:

```
.model top
.inputs a b c d
.outputs y
.gate SB_LUT4 I0=b I1=c I2=d I3=a O=y
.param LUT_INIT 0000011111111000
.end
```

Additional Input Files

- Physical Constraints File (.pcf)
 - Using a similar format as the Lattice Tools
 - Used primarily for **IO pin placement**
- Place-and-route Script (aka “passfile”)
 - Defines which passes to execute
 - And what options to use for this passes
 - E.g. for using Yosys' Analytical Placer output with Arachne-pnr

Output Formats

- Primary Output:
 - FPGA Config in **IceStorm ASCII** Format
 - Can be converted using IceStorm tools to
 - Timing Netlist (under construction)
 - Behavioral Verilog Model
 - FPGA Bit-stream
- Optional Additional Outputs:
 - **BLIF Netlist** of all intermediate steps
 - **PCF File** with assigned placements

Part 3 of 4:

Yosys

Yosys Open SYnthesis Suite

- Yosys can:
 - Read **Verilog**, BLIF, Liberty Cell Libraries, ...
 - Write Verilog, **BLIF**, **EDIF**, SPICE Decks, SMT2, ...
 - Perform RTL **synthesis** and logic **optimization**
 - Map designs to **FPGA and ASIC** cell libraries
 - Perform various **formal verification** tasks

... some say Yosys is “**LLVM for Hardware**”.

Existing Yosys Flows

- Currently there are two **FOSS ASIC Flows** that use Yosys:
 - Qflow: <http://opencircuitdesign.com/qflow/>
 - Coriolis2: <https://soc-extras.lip6.fr/en/coriolis/coriolis2-users-guide/>
 - Multiple successful tape-outs – **People make silicon with this!**
- Synthesis for **iCE40 FPGAs** with Arachne-pnr and IceStorm as place-and-route back-end.
- Synthesis for **Xilinx 7-Series FPGAs** with Xilinx Vivado as place-and-route back-end.
- Yosys-smtbmc is a **formal verification** flow with support for multiple SMT solvers (everything with SMT2 support, tested with: Z3, Yices, CVC4, MathSAT)

Example ASIC Synthesis Script

Yosys is controlled by scripts that execute Yosys passes that operate on the in-memory design.

For example:

Generic part →

```
# read design
read_verilog mydesign.v

# generic synthesis
synth -top mytop
```

Target-specific part →

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

# write synthesized design
write_edif synth.edif
```

Details of “synth” command

Many Yosys commands are like scripts on their own: All they do is run a sequence of other commands. For example the synth command is just an alias for (see also `help synth`):

```
begin:  
    hierarchy -check [-top <top>]
```

```
coarse:  
    proc  
    opt_clean  
    check  
    opt  
    wreduce  
    alumacc  
    share  
    opt  
    fsm  
    opt -fast  
    memory -nomap  
    opt_clean
```

```
fine:  
    opt -fast -full  
    memory_map  
    opt -full  
    techmap  
    opt -fast  
    abc -fast  
    opt -fast
```

```
check:  
    hierarchy -check  
    stat  
    check
```

Methods of Formal Verification in Yosys

(Off-topic here, but an important part of my work. :)

- **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. This is what `yosys-smtbmc` does.

Part 4 of 4:

IcoBoard + Demo SoC

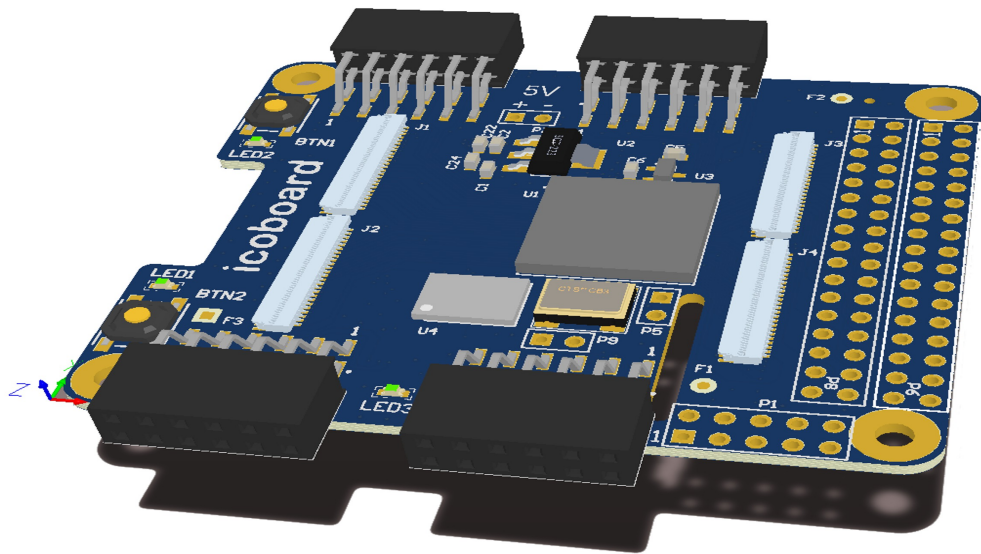
iCE40 Development Boards

- Nandland Go Board
 - <https://www.kickstarter.com/projects/1531311296/nandland-go-board-your-fpga-playground>
- ICed = an Arduino Style Board, with ICE FPGA
 - <https://hackaday.io/project/6636-iced-an-arduino-style-board-with-ice-fpga>
- Wiggleport
 - <https://github.com/scanlime/wiggleport>
- eCow-Logic pico-ITX Lattice ICE40
 - <http://opencores.org/project,ecowlogic-pico>
- CAT Board
 - <https://hackaday.io/project/7982-cat-board>
- IcoBoard
 - <http://icoboard.org/>
- Lattice Dev Boards
 - iCEstick Evaluation Kit
 - iCE40-HX8K Breakout Board
 - iCEblink40HX1K Evaluation Kit
 - <http://www.latticesemi.com/>



IcoBoard – Open Hardware iCE40 HX8K Raspberry Pi Hat

- Up to 20 PMOD ports
 - 4 PMODs directly on board
 - 16 PMODs via IcoX extension boards
 - Almost 200 IO pins in total
- Possible Applications
 - Intelligent Raspberry IO Expander
 - Raspberry Pi as network enabled programmer/debugger
 - On-demand HDL generation and bit-stream synthesis

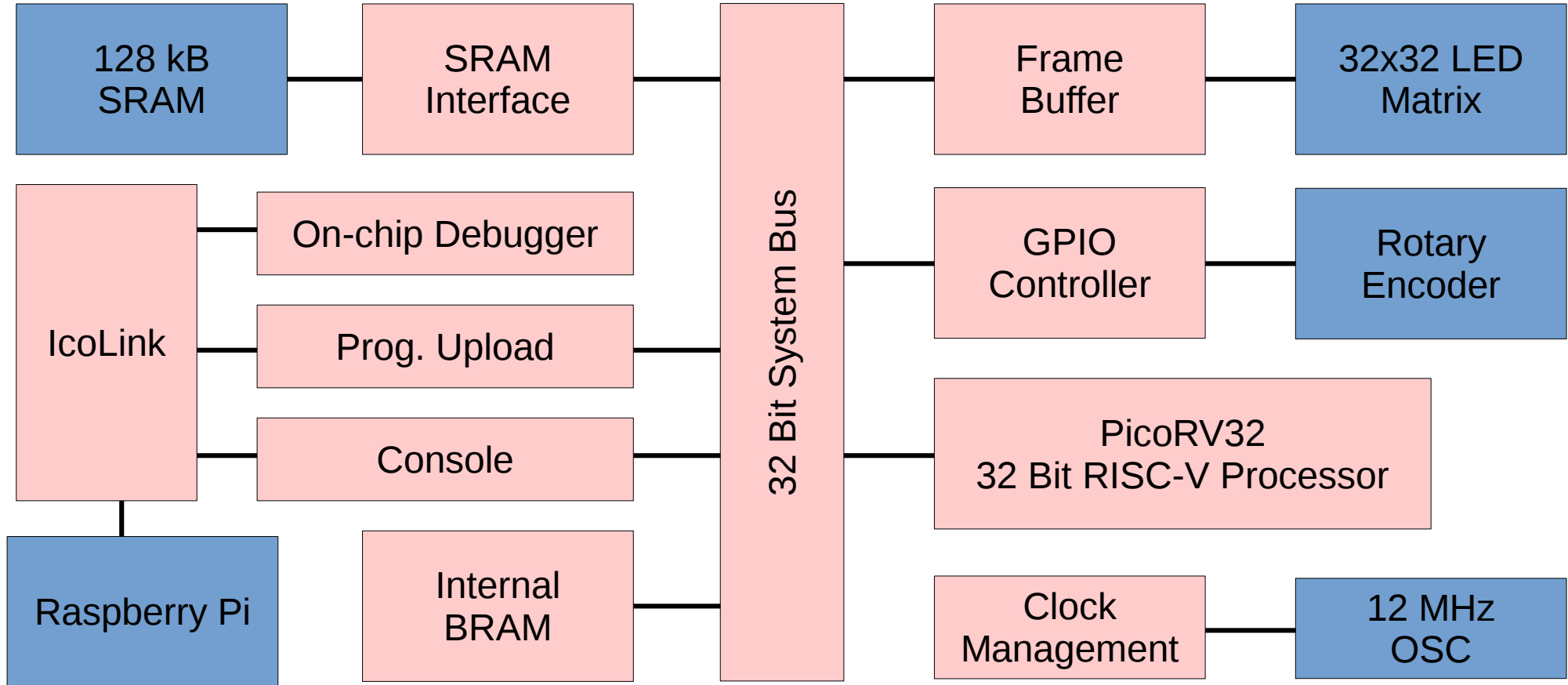


- <http://icoboard.org>

Demo SoC – Motivation – Role of Raspberry Pi

- We built a small Demo SOC
 - Uses about 50% of the HX8K logic resources
 - Includes a 32 Bit Processor (RISC-V Compatible, GCC Toolchain)
- Our motivation is to demonstrate
 - That our flow can handle nontrivial real-world designs
 - That even a small 8k LUT FPGA can do big things
- In this Demo the Raspberry Pi is exclusively used as
 - network-enabled programming and debug probe
 - ssh-gateway for the SoC text console

Demo SoC – Simplified Block Diagram



Synthesis Script for Demo SoC

- The Demo SoC FPGA design is built using a Makefile:

```
c3demo.blif: c3demo.v ledpanel.v picorv32.v firmware.hex
    yosys -v2 -p 'synth_ice40 -abc2 -top c3demo -blif c3demo.blif' c3demo.v ledpanel.v picorv32.v

c3demo.txt: c3demo.pcf c3demo.blif
    arachne-pnr -s 1 -d 8k -p c3demo.pcf -o c3demo.txt c3demo.blif

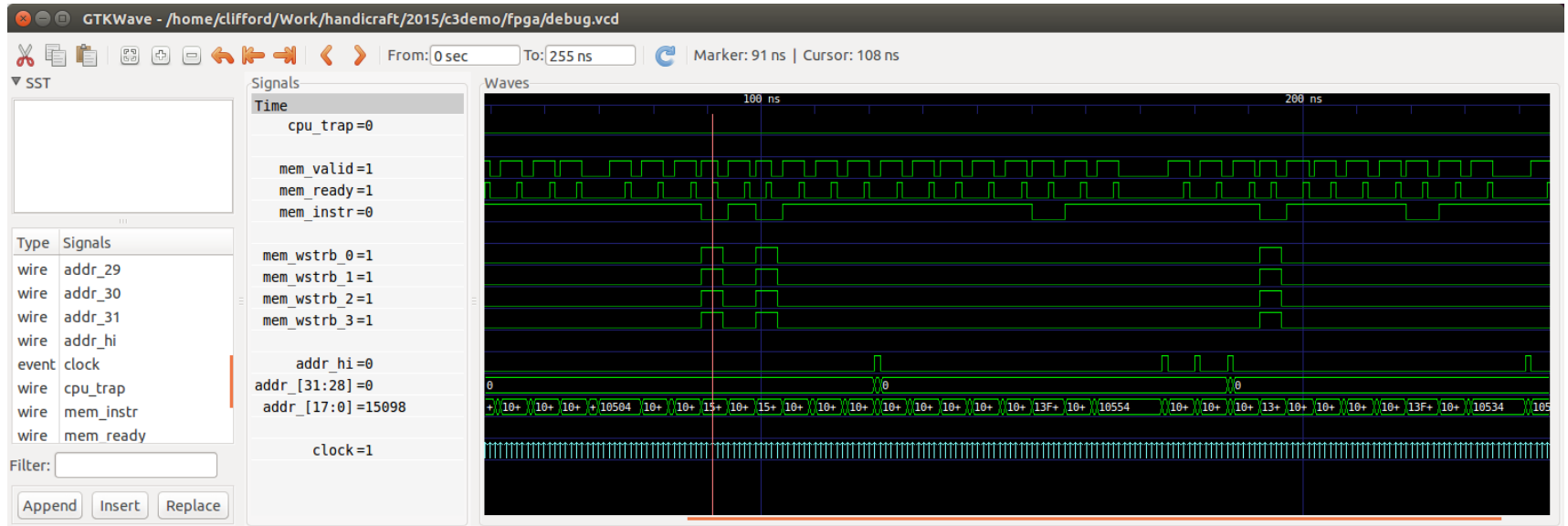
c3demo.bin: c3demo.txt
    icepack c3demo.txt c3demo.bin
```

- The `firmware.hex` file is built by other make rules using the RISC-V Compiler Toolchain (GCC and GNU Binutils).
- Additional Make rules for programming:

```
prog_flash: c3demo.bin
    $(SSH_RASPI) 'icoprogram -f' < c3demo.bin
...
```

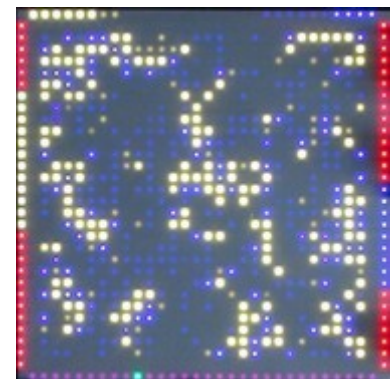
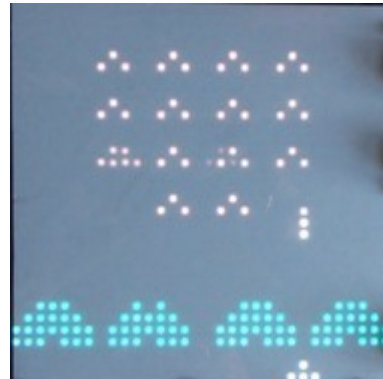
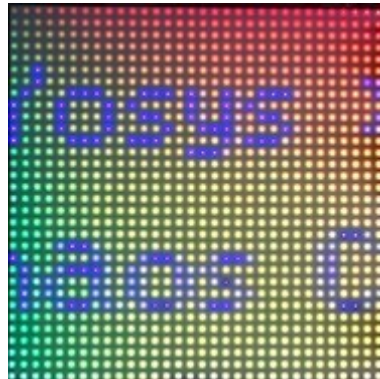
Using the On-Chip Debugger

- The on-chip debugger can be connected to any number of nets, use any trigger- and/or enable-condition.
- Rerunning synthesis is necessary after changes to the debugger.
- Run `make debug` to download a dump and store as VCD:



Running Applications on the SoC

- The demo SoC project comes with a few example apps.
- The base FPGA design has a boot loader stored in block RAM that is executed at boot-up.
- The `make run` target in the `app_*` directory reboots the FPGA from its serial flash, and sends the `.hex` of the application image to the boot loader via the console interface.



Comparison with Lattice iCEcube2 Flow

	Yosys Arachne-pnr	Synplify Pro SBT Backend	Lattice LSE SBT Backend
Packed LCs	2996	2647	2533
LUT4	2417	2147	2342
DFF	1005	1072	945
CARRY	497	372	372
RAM4K	8	7	8
Synthesis Time	30 seconds	30 seconds	21 seconds
Implementation Time	81 seconds	405 seconds	415 seconds

Notes:

- 1) Timings for Intel Core2 Duo 6300 at 1860 MHz running Ubuntu 15.04.
- 2) Using iCEcube2.2014.12 because I had troubles activating the license on newer versions.
- 3) SoC without internal boot memory and frame buffer because Synplify Pro and LSE both could not infer implementations using iCE40 block RAM resources from the behavioral Verilog code.

Timing Analysis Comparison

Design	Timing Tool	Yosys Arachne-pnr (unconstrained)	Lattice LSE SBT Backend (constr. 100 MHz)
PicoRV32_AXI (w/ reduced pin count)	sbtime	N/A	41.74 MHz
	icetime -i	54.33 MHz	41.75 MHz
	icetime -im	53.02 MHz	41.40 MHz
Navre AVR Clone (from Milkymist SoC)	sbtime	N/A	45.82 MHz
	icetime -i	29.89 MHz	45.59 MHz
	icetime -im	27.61 MHz	44.90 MHz
Whishbone SPI Core (from OpenCores)	sbtime	N/A	62.13 MHz
	icetime -i	42.62 MHz	62.23 MHz
	icetime -im	38.89 MHz	61.14 MHz

Current Limitations of IceTime: No STA (purely topological), No multiple clock domains, Pessimistic (-m) or estimated interconnect model, No modeling of launch/capture clock paths

Extra Part:

Formal Verification with Yosys

Formal Verification with Yosys

- Built-in **equivalence checking** framework (see `help equiv_*`)
- Support for SystemVerilog **assert and assume statements**
- Creating **miter circuits** for equivalence or property checking:
 - Either solve with built-in solver or
 - Export as BLIF and solve with e.g. ABC
- **SAT solving** (built-in MiniSAT-based eager SMT solver, see `help sat`)
- Creating SMT-LIB 2.5 models for circuits and properties that can be used with **external SMT solvers**
 - **BMC and temporal induction** with `yosys-smtbmc`

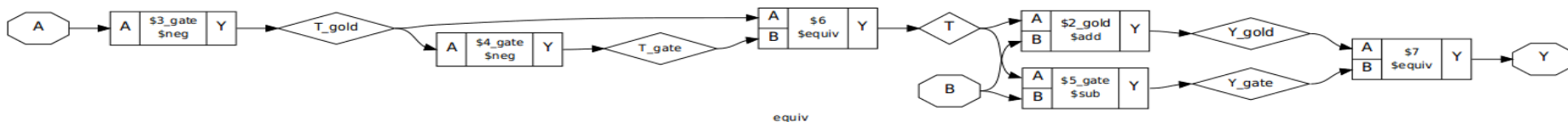
Equivalence Checking

- The **equiv_* commands** in Yosys are for equivalence checking.
- This equivalence checker uses **hints like net names to partition the circuit** 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!



Verilog asserts

- Yosys supports SystemVerilog assert and assume:

In module context:

```
assert property (<expression>);  
assume property (<expression>);
```

In always-block context:

```
assert (<expression>);  
assume (<expression>);
```

```
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
```

```
$ yosys example_assert.ys
```

```
...
```

```
Solving problem with 845 variables and 2305 clauses..
```

```
SAT proof finished - model found: FAIL!
```

```
-----  
(____ \           /__ )   /__ )   ( ) |           || |  
_____) )____ _   _ _ | | _   | | _ _ _ _ _ | | _ _ _ | | |  
| ____/ ____ ) _ \ / _ ( ____ ) ( _ _ _ _ _ | | | _ _ _ | / _ | | | | | | | | | | | | | | | | | | |
| | | | | | _ | | | | | | | | | | | | / _ _ | | | | _ _ _ ( ( | |  
| | | | | \ _ / \ _ / | | | | | \ _ _ | | \ ) _ _ _ ) \ _ _ | |
```

Signal Name	Dec	Hex	Bin
\A	-2147483648	80000000	10000000000000000000000000000000

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

Miter circuits

- Some tools (e.g. ABC) operate on **miter circuits**.
 - Mitters are circuits with a **single output**
 - that is **asserted when** the **property is violated**
- The miter command can be used to create mitters
 - For **properties** using assert and/or assume statements
 - For **equivalence** of two circuits

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

```
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
```

```
$ 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

Yosys sat command

- 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`

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

Formal Verification with yosys-smtbmc

- SMT-LIB2 is a language for describing SMT problems
- Practically all SMT solvers support it, because SMT competition is using it as language for test problems
- Yosys can convert Verilog designs to SMT-LIB2 code snippets
- Yosys-smtbmc is a Python script that can run various formal methods on those code snippets.
- Comes with a Python library for writing custom proofs. (smtio.py)

Simple SMT-LIB2 Example Code

```
(set-option :produce-models true)
(set-logic QF_LIA)

(declare-fun s () Int)
(declare-fun e () Int)
(declare-fun n () Int)
(declare-fun d () Int)
(declare-fun m () Int)
(declare-fun o () Int)
(declare-fun r () Int)
(declare-fun y () Int)

(assert (distinct s e n d m o r y))
(assert (distinct s m 0))

<...>
```

```
<...>

(define-fun is_digit ((i Int)) Bool (and (<= 0 i) (>= 9 i)))
(assert (and (is_digit s) (is_digit e) (is_digit n) (is_digit d)
             (is_digit m) (is_digit o) (is_digit r) (is_digit y)))

(define-fun four_digits ((d1 Int) (d2 Int) (d3 Int) (d4 Int)) Int
  (+ (* 1000 d1) (* 100 d2) (* 10 d3) d4))
(define-fun five_digits ((d1 Int) (d2 Int) (d3 Int) (d4 Int) (d5 Int)) Int
  (+ (* 10000 d1) (four_digits d2 d3 d4 d5)))

(assert (= (+ (four_digits s e n d) (four_digits m o r e)) (five_digits m o n e y)))

(check-sat)
(get-value ((four_digits s e n d) (four_digits m o r e) (five_digits m o n e y)))
```

* This example uses the QF_LIA logic
(Quantifier Free Linear Integer Arithmetic)

Yosys-smtbmc uses QF_AUFBV
(Quantifier Free, Arrays, Uninterpreted
Functions, and Bit Vectors)

send	9567
+ more	+ 1085
-----	-----
money	10652



```
$ z3 -smt2 smtdemo.smt2
sat
(((four_digits s e n d) 9567)
 ((four_digits m o r e) 1085)
 ((five_digits m o n e y) 10652))
```

Yosys-smtbmc example

```
module example(input clk);
    reg [3:0] cnt_a = 1, cnt_b = 0;
    reg [7:0] counter = 150;

    always @(posedge clk) begin
        if (cnt_a) begin
            if (cnt_a == 10) begin
                cnt_a <= 0;
                cnt_b <= 1;
            end else
                cnt_a <= cnt_a + 1;
            counter <= counter + 1;
        end else begin
            if (cnt_b == 10) begin
                cnt_b <= 0;
                cnt_a <= 1;
            end else
                cnt_b <= cnt_b + 1;
            counter <= counter - 1;
        end
    end
end

<...>
```

<...>

```
    assert property (100 < counter && counter < 200);
    assert property (counter == cnt_a + 149 || counter == 161 - cnt_b);
    assert property ((cnt_a == 0) != (cnt_b == 0));
    assert property (cnt_a <= 10);
    assert property (cnt_b <= 10);

    always @* begin
        if (cnt_a)
            assert (counter == cnt_a + 149);
        else
            assert (counter == 161 - cnt_b);
    end
endmodule
```

```
yosys -q - <<EOT
read_verilog -formal example.v
hierarchy; proc; opt
write_smt2 -mem -bv -wires example.smt2
EOT
```

```
yosys-smtbmc example.smt2
yosys-smtbmc -i example.smt2
```


Links

- Yosys
 - <http://www.clifford.at/yosys/>
- Project IceStorm
 - <http://www.clifford.at/icestorm/>
- Arachne-pnr
 - <https://github.com/cseed/arachne-pnr>
- PicoRV32
 - <https://github.com/cliffordwolf/picorv32>
- IcoBoard
 - <http://icoboard.org/>
- IcoProg (not final location!)
 - <http://svn.clifford.at/handicraft/2015/icoprogram>
- Demo SoC (not final location!)
 - <http://svn.clifford.at/handicraft/2015/c3demo>
- This presentation
 - <http://www.clifford.at/papers/2015/icestorm-flow/>

