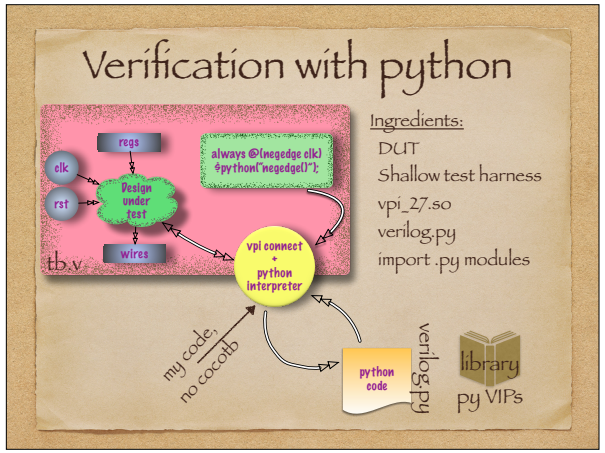
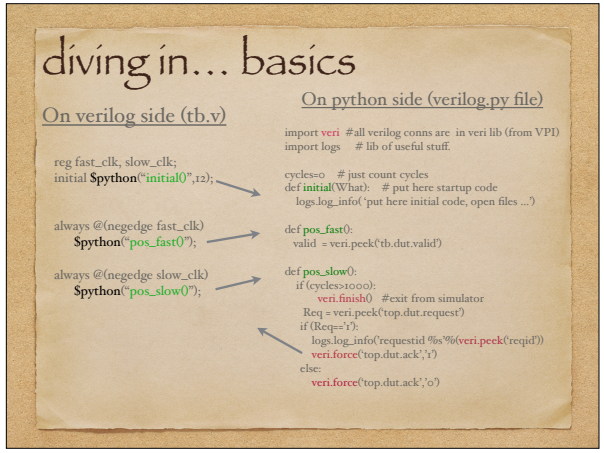
Hop onboard, before Apple starts developing it's own silicon. Oops - it happened.
Don't worry, Apple is a conservative as Your management.

I will try to show that using Python is 3 to 5 times more effective than traditional methods.
The verification code is much shorter and easier to adopt, port and understand.
While legacy will not kill Intel anytime soon, it kills smaller startups. The verification code base becomes un manageable, corners are cut, and silicon arrives crippled.

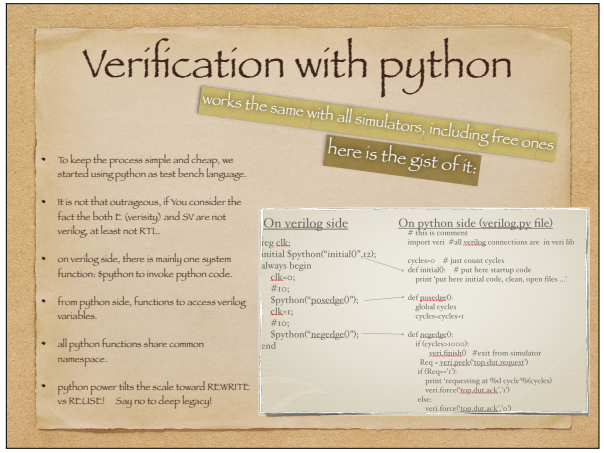
Even if in unlikely case 😜, that Intel will not adopt this methodology, You may adopt some ideas and broaden Your horizons.



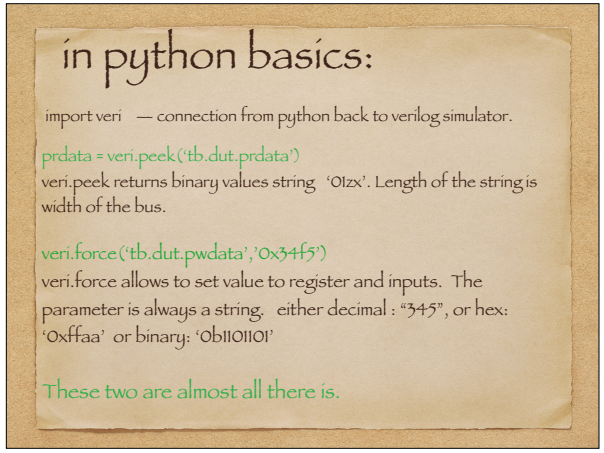
My contribution is "C" code (the yellow blob) that compiles into shared object (in Linux, Mac or Windows) and enables to activate python functions from the running simulation. And allows python code to monitor and manipulate the simulation.
Over time, many modules were written, that play various roles- AXI, Serdes, Interlaken, APB, Jtag, I2C,….
The test harness is very shallow. it includes only regs on inputs, wires on output, clocks and resets. You hang python functions on relevant events. You may hang as may as You wish. In 99% cases it is just one clock.



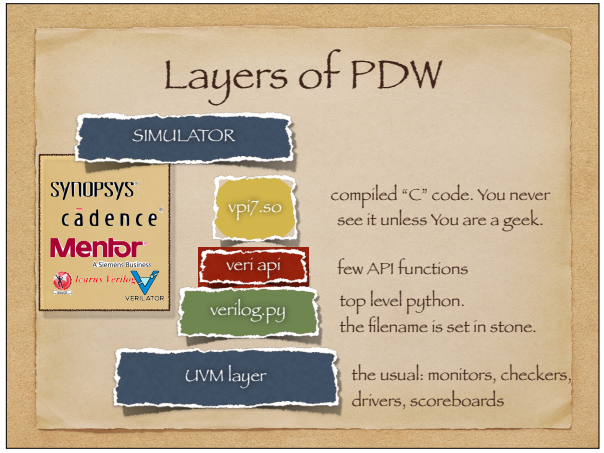
On verilog side single system call “$python” is added. It calls to specific python function. Shared object opens “verilog.py” python module. It must be present locally (usually as link). “verilog.py” is the head of the pyramid, it imports all other support modules.
On python side, “veri” module gives access to verilog simulation. You can peek, force and several more actions. Peek and Force (deposit actually) are 99% of actions used.



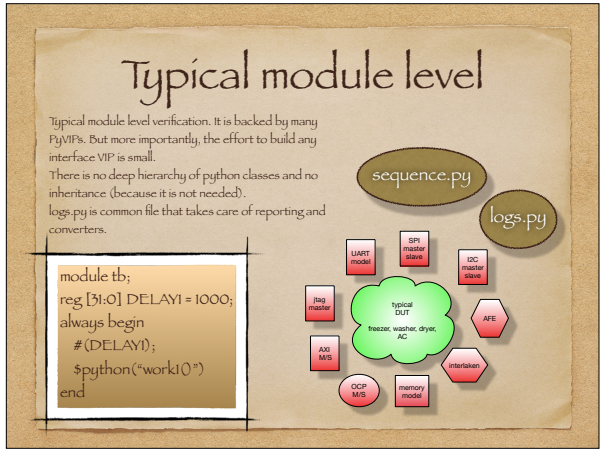
Some people may leave Intel in the distant future. It happens. It is a good idea to know the world around and open horizons.



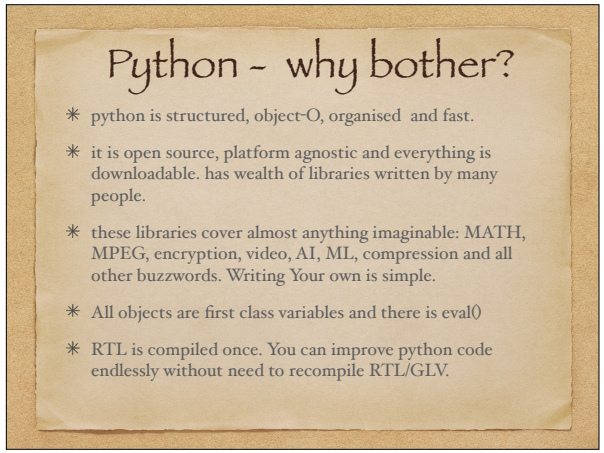
veri.force(Net,Value). Both Net and Value are strings.
veri.peek(Net) returns a string of Ones, Zeros, Xs-es and Z-s. Width of the net is simple len() of the string. Several functions in logs.py can convert this into integers, signed integers, floating point, ascii string and more.



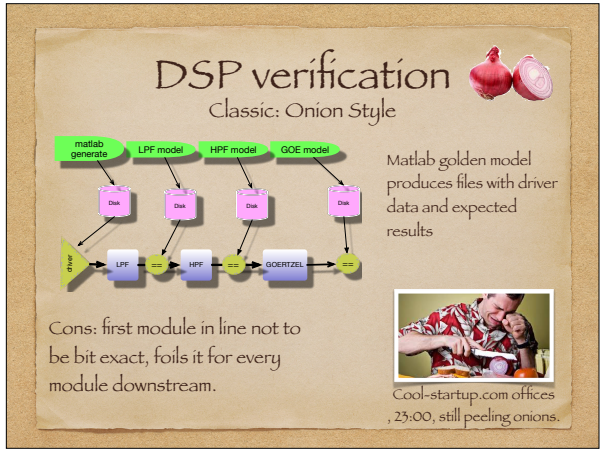
It is enough to connect python to simulation. However to truly uncover this methodology strength whole ecosystem must be adopted. Over the years, i built a good ecosystem. It implements verification methodology, alas uses unstandard names.
UVM expert could help me to present it using more standard naming.
Having said that, it is so easy to develop one Yourself, provided You know verification,



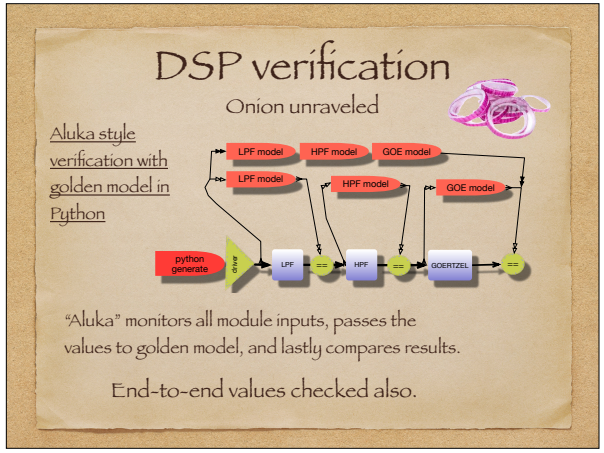
Over time, i accumulated numerous agents, playing roles of different hardware interfaces.
But it is minimal effort to create or recreate more agents. Python language is very powerful.
In many instances, like crc, compression, encryption, math (sinus, tangent,…) code already exists and is open sourced.
Personally, i don't like using inheritance and there is little need for that. I have seen incidents of SV/UVM developers become suicidal because of inheritance. That is not the case if Your last name is Rothschild.



Several reasons to adopt Python (vs PERL, C++,…). It is stable, well documented and understood language. Many times i overheard heated discussions on the meaning of some SV/UVM construct.
It doesn't happen often in Python. Everything is transparent.
And it is fast. The VPI connection is fast. Maybe i will look into DPI, but it is not open-sourced.

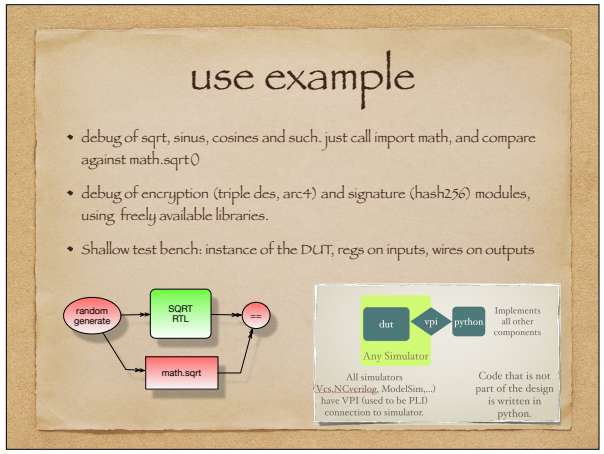


Real story. Some smart companies ditched Matlab and moved to Python as DSP development envir. In that case, no files are needed and simulation can run for weeks, randomising inputs and checking outputs.
In one case i added email send, from time to time, to alert me to the progress or problems.
Using files, is rigid and is very brittle.

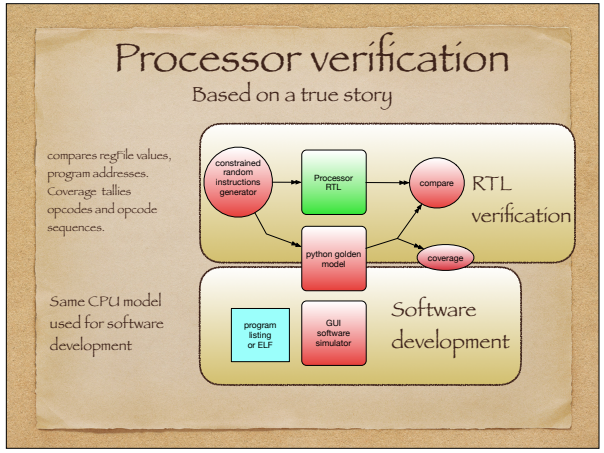


Aluka is an idea, where python code mimics accurately some hardware module on inputs/outputs boundary.
Aluka style is where You have python model for some or all pipeline modules and check each against it's actual inputs. You also should check end-to-end . Even if some module in the middle doesn't produce bit-exact results, All downstream modules are checked.

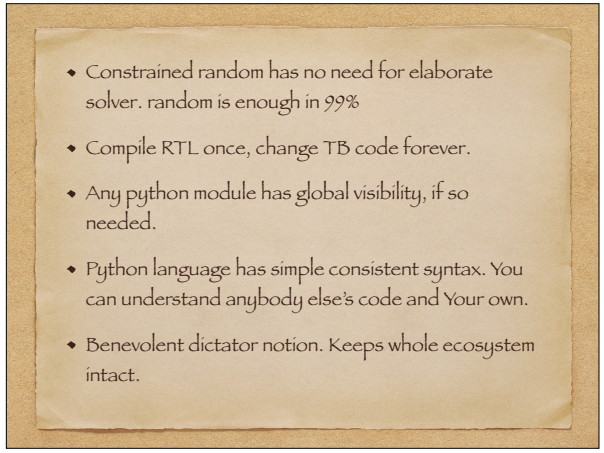
Aluka objects can be called several times and connected to several instances of the same RTL module. Strings based interface makes it trivial.

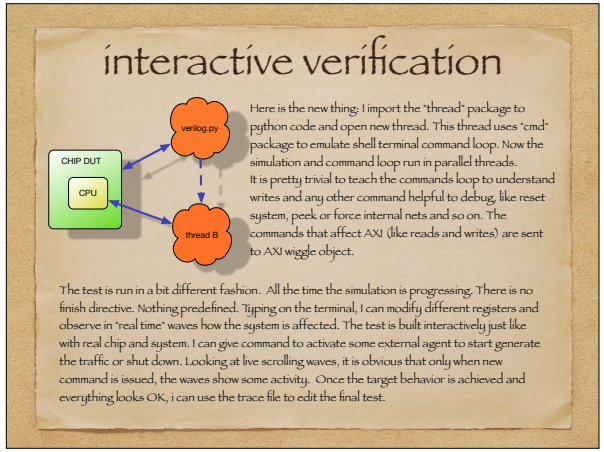


This is more straightforward example. Verification at unit level.



Several embedded CPU were verified this way. Some other tool created accurate python code that mimics the CPU. It is used by software types to develop the code, and by verification team. Coverage of instructions, instruction mixes and jump/nojump conditions are very simple to create.
Playing debug through JTAG is simple too.

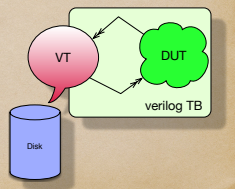




- "python mymodule.py" : simple way to check Your syntax.
- Python error messages are helpful and informative.
- While there are IDE's - i never had a need to use one.
- veri.listing(Path,Depth,FileName) — dump instant values of signals from Path deep (Depth levels) into a file named FileName. Tip: Useful to grep "= z" to catch all un-driven nets.
-

Virtual tester

Files sent to production tester, come in several flavors, most common today is STIL.
Python is used to read STIL file, imitate IC tester, drive and sample DUT pins.
This reduces test house reruns.

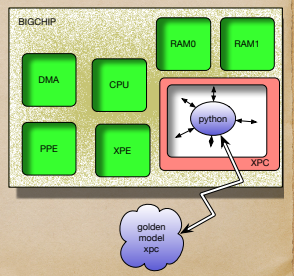


Just another example of versatility. Python, as general purpose programming language, can read any readable format file and do something about it. It can write easily different formats and log files.

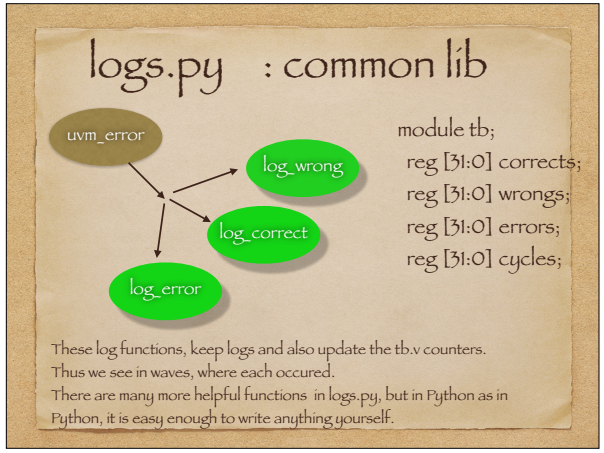
Early module filler

InsideOut Aluka

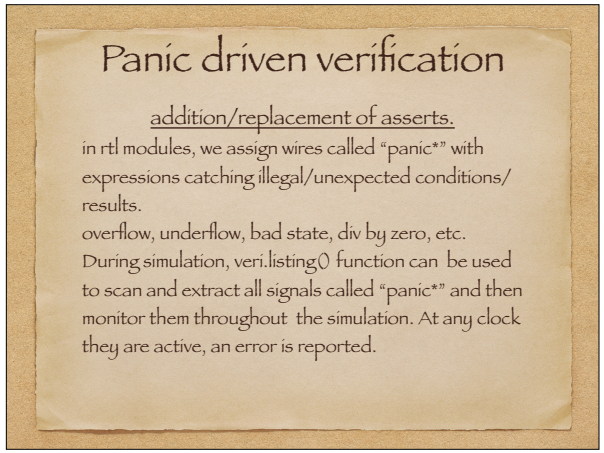
Long before some RTL module is ready or even written, Python golden model can be used as early implementation placeholder.
Create "pretzel" or "beigale" verilog module of the pinout. This module has just inputs and all outputs are regs (can be helped by pyver.py). Wrap around the golden model an adaptor to the actual module pins. This wrapper can model the processing delays. Voilà, system integration can proceed.



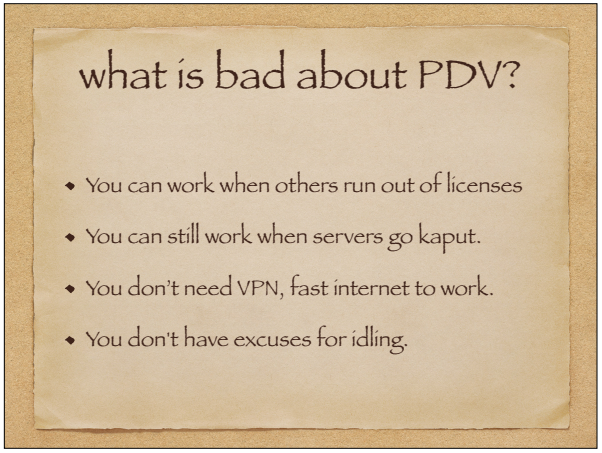
Maybe less relevant to verification team, but was a big help several times for me.



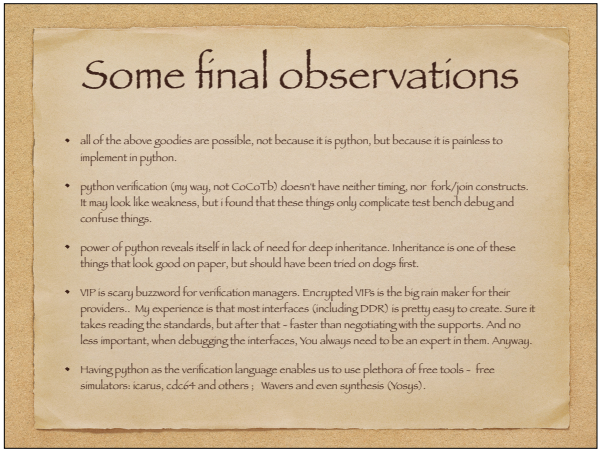
We split the UVM message into error and wrong. wrong is used when comparison to golden model fails. I also print “correct” into log file, to create sense of context when debugging. Is it wrong on all comparisons? or just one, first one, last one and so on.
log_error is reserved for python and system errors (trying to open unexistent file) or unexpected behaviour of the DUT - timeout, reading from empty fifo accessing out-of-bounds address.
Each message is numbered and also increments register in testbench. Thus making it easy to correlate log file with waves.
logs.py includes also numerous conversions functions, like from string to floating, string to int, to signed int and many more.
Test is declared when there are no errors, no wrongs and some amount (>0) of corrects. If there are no corrects, it is possible nothing happened.

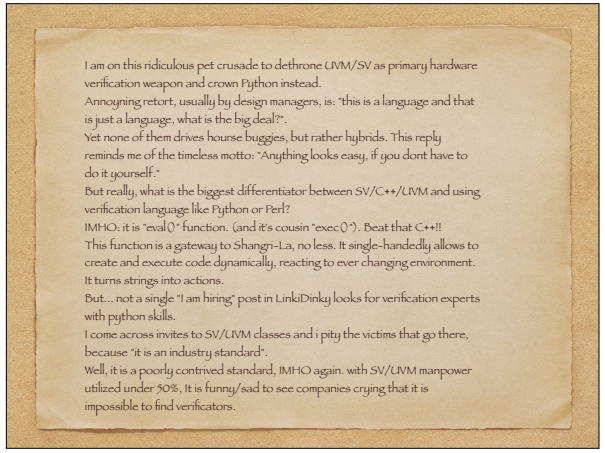


It is useful, when me as a developer receive waves from the verification team. They tell me something went wrong, but not what exactly. First, I check all panics that were inserted. Many times it saved huge amount of time.
Panic wires sit tightly in simulation, unlike assertions and go quietly into the night in synthesis.
Flip side, i also added some “interesting event” equations. Useful for coverage and snooping on quality of Your tests.

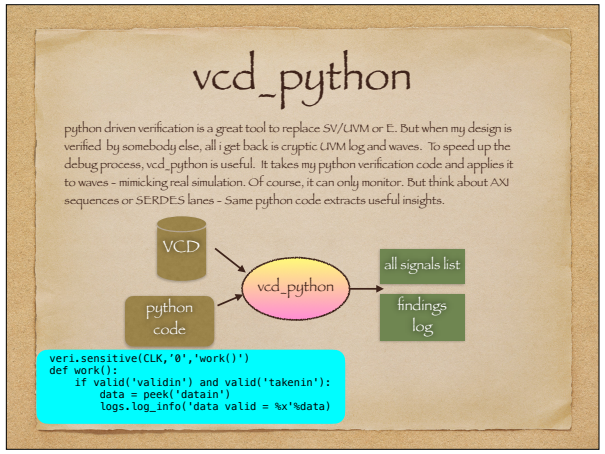


PDV = python driven verification.

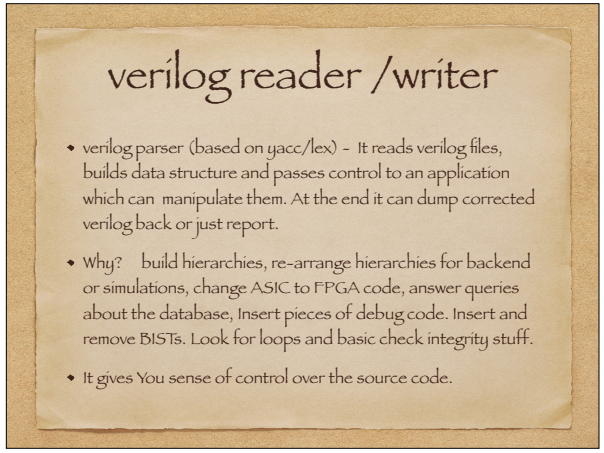


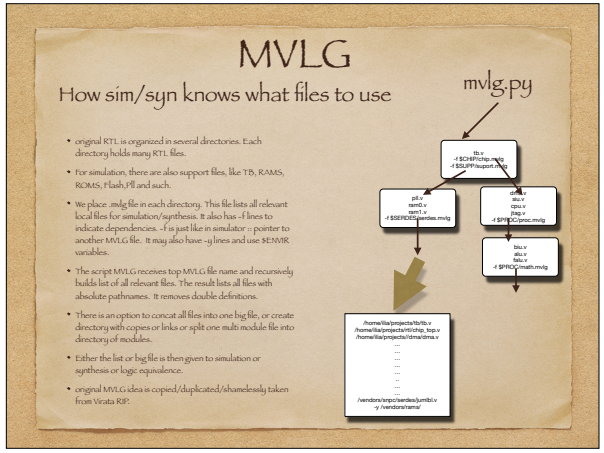


Just something i wrote a while back.

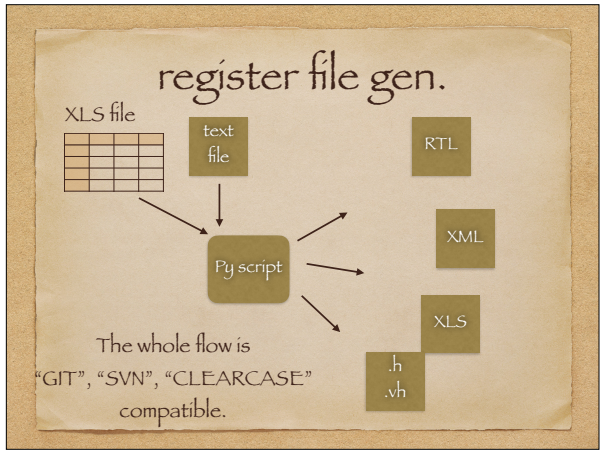


Hope You know what VCD is.
veri.sensitive hangs a python function on posedge or negedge of some signal, usually clock.
The function can query the design, keep context and print findings. The executable is compiled from “C”. It is fast.

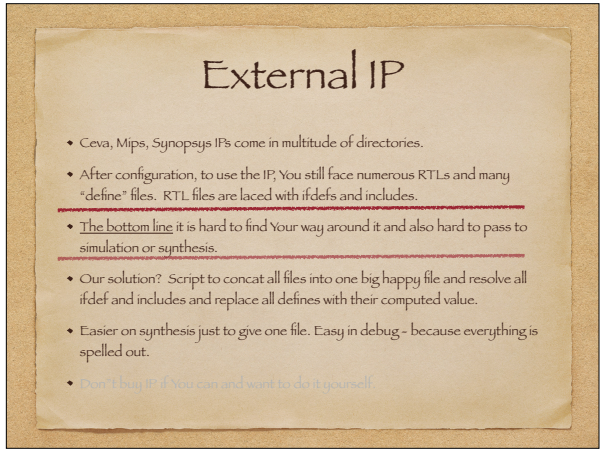


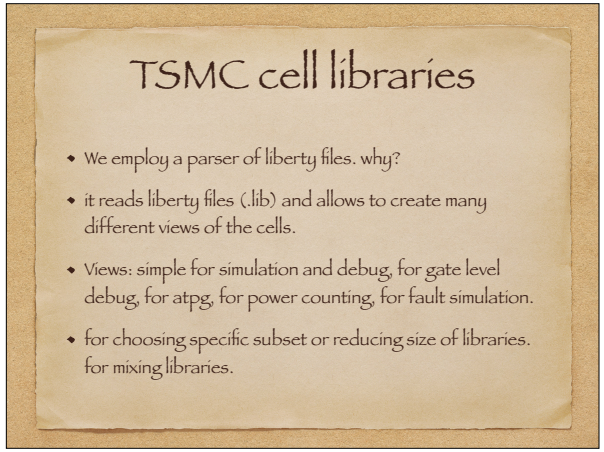


effective file list methodology for simulation and synthesis.

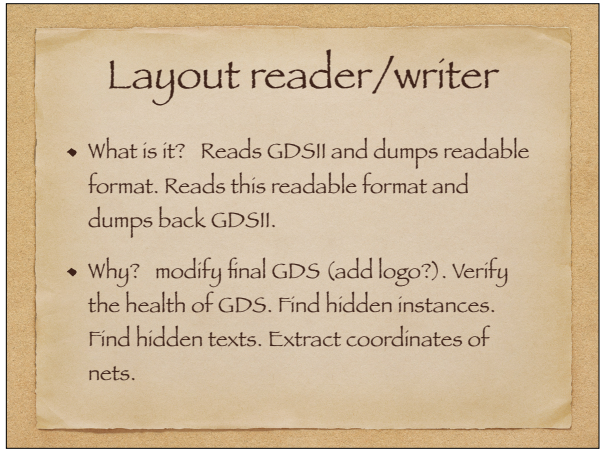


It accepts XLS file or plain text file and produces all needed views. All files are version tools friendly.





Look in official library for humble NAND or INV. It is long and carries a baggage. This way enables to craft libraries for different scenarios: variable delays, technologies.



Genver: replacement of generate.

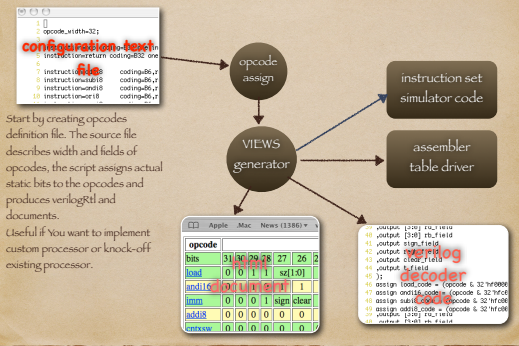
- I hate generate statement in verilog (and instance[0:3] for that matter too).
- It produces strange names and ugly code, You cannot always deduce what is produced and it is limited to whole syntax structures.
- Solution? humble app called genver.py. It is macro preprocessor, where the macro language is python.
- Looks like verilog code spiced with python commands. Then expanded into full verilog file. This file is readable and debuggable.
- The translation is fast and usually done online with simulation or synthesis.

```
#WID = 30
## this will become python comment for yourself
module example (   input clk,input rst
#for ll in range(WID):
  ,output [ll:0] outll
#   ,JJ = WID-ll
  ,input [JJ:0] inll
#
);

#for ll in range(WID):
scm.scmll #(ll,WID) (.in(inll) ,.out(outll) ,.clk(clk) ,.rst(rst));
#

#for ll in range(10):
wire [WID:0]  wwll = 0
#   for JJ in range(5):
    + cucd[JJ]*coeff[ll]
#<
  ;
#
endmodule
```

Fixed and Float math lib generators

- Script accepts either list or single math module name and creates it.
- It can create mul, limited mul, divider, limited divider, square root, distance module for fixed point. Configurable are width of operands and number of stages.
- Cordic and table based sin/cos/atan/asin/acos were added lately.
- The system generates to exact spec, no defines nor parameters. WYSIWIG!!
- and no surprises in logic equivalence. and if better algorithm exists - it can be incorporated in matter of minutes.

Instruction set gen



Start by creating opcodes definition file. The source file describes width and fields of opcodes, the script assigns actual static bits to the opcodes and produces verilog Rtl and documents.
Useful if You want to implement custom processor or knock-off existing processor.

Network on a chip

only messages run around

system built on no-instant gratification (everything takes time)

totality: only rings allowed. there are no wires coming out of modules, except rings.

comprehensive: rings carry control and data: interrupts, status, data, clocking, power save modes, debug and test.

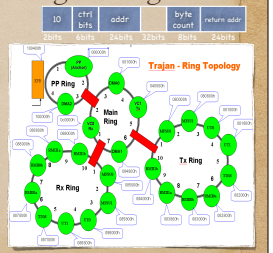
topology doesn't influence functionality, only performance.

no masters / no slaves - all members are producers and consumers.

keep it simple: documentation should be shorter then the code.

ditch signals and busses - define the chip by transactions and flows. This applies to status registers, interrupts and other requests and acknowledges.

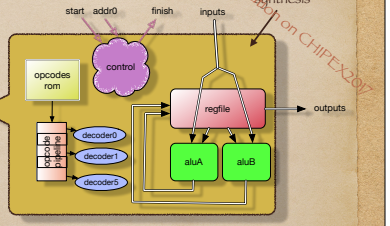
e.g. read msg



FloatProc

add
sub
mul
div
sin
cos
acos
asin
atan
abs
sqrt
select
compare

configuration to reduce synthesis

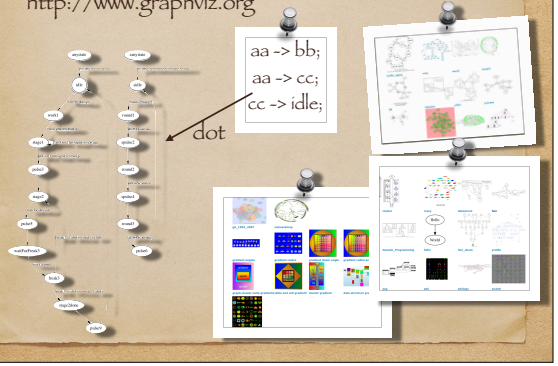


- dual issue
-black box setup
- up to 32 input values
-up to 32 registers
- dflow compiler programming

format

Dot : diagrams the human way

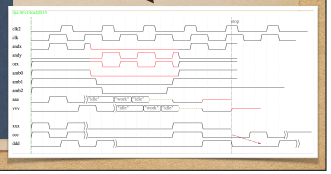
http://www.graphviz.org



waveformer

```
sig clk (repeat 20 (3 0) (3 1))
sig clk2 (repeat 20 (3 0) (2 1))

sig amb0 (10 1) (color red) (5 0) (color black) (10 1)
sig amb1 (8 1) (8 0) (8 1)
sig amb2 (12 1) (8 0) (20 1)

sig aaa   (3 0) (3 1) (3 0) break (3 "idle") (3 "work") (3 "idle") (4 z) (3 0) (3 x)
vgrid 0 2
sig vvv (delay 5 aaa)
sig xxx (insert aaa)

sig ccc xxx (mark x0) (insert xxx)
sig ddd (delay 5 xxx)  (mark x1) (insert xxx)

vmark stop (mark x0)

sig zzzzzzzzzz (3 0) (2 up) (4 1) (2 down) (3 0)
sig andx (and amb0 clk)
sig orx (or amb0 clk)
sig andy (and (not amb0) (not clk))

range 0 (len amb2)
arrow (mark x0) (mark x1)

color vmark 0.5 0.5 0.5
color sig 0 0 0
color arrow 0 0 0
display clk2 clk andx andy orx amb0 amb1 amb2 aaa vvv * xxx ccc ddd *
```

waveformer



Yosys : open synthesis

Yosys runs on linux. It accepts synopsys libraries after some filtering. It reads most of RTL (at least the ones i tried).

Why use it, when there is Synopsys/Cadence?

1. in seconds, You get indication whether RTL is valid.
2. Combi Loops easy to find.
3. Early area / Timing indications.
4. It is fun.

more Free tools
that we use.

- gtkwave and dinotrace for VCD waves viewing.
- icarus and cvc64 verilog simulators.
- zDraw schematic editor.
- mWave analog waves viewer (from .csv files)
- vcd-python player. python extracts events in vcd.

homemade tools examples

- rename module/instance names in the whole verilog hierarchy.
- expand (make explicit) defines/ifdefs/genvers/parameters in verilog code.
- Fubarize code.
- XLS/CSV to RegFile generator.
- lef / def parser and translator.
- VCD analyzers.
- ATPG formats translators.
- Simulink diagram convertor to rtl, processor or assembler.

Summary till now

- Each item above is not earth shattering by itself.
- But their collectivity and change of attitude do carry a big impact.
- Try to move the bar between self-made and purchased stuff higher.
- ריכוך ארטלרי
  All examples before this one were part of "artillery barrage" in the sense to make You less respect the well-trodden paths, open up Your mind to new ideas, especially python verification.