designwest

center of the engineering universe

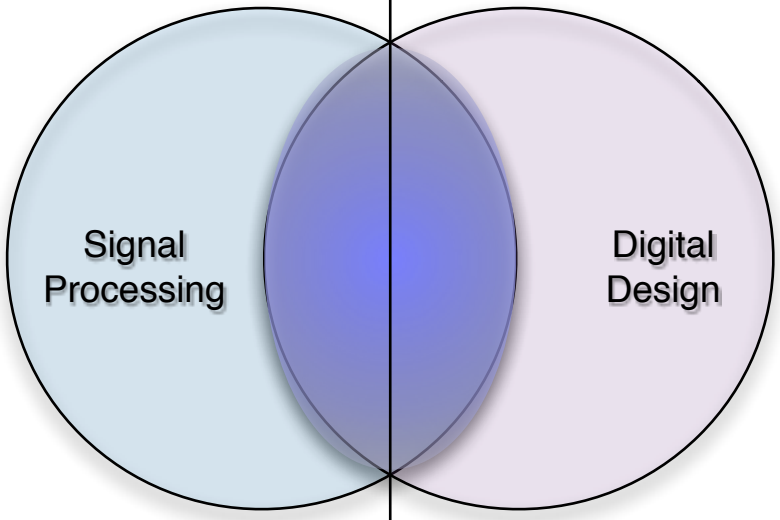
Why You Should be Using Python/MyHDL as Your HDL
ESC-329

April 22-25, 2013
McEnery Convention Center
San Jose, CA
www.ubmdesign.com

UBM Tech

About Me

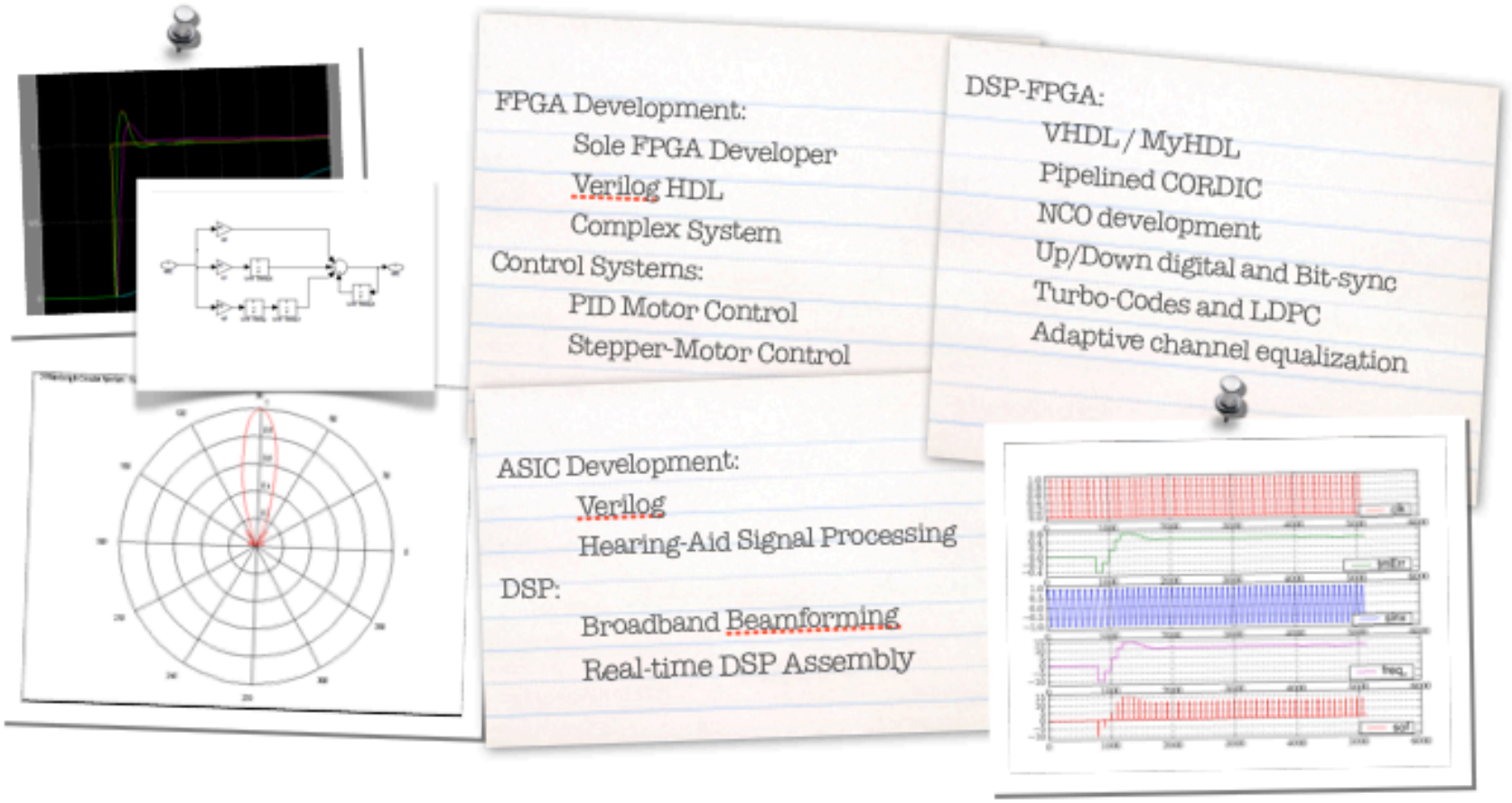
- Self proclaimed applied DSP engineer (DSP-FPGA)
- Using MyHDL 6-8 years more involved the last 4 years



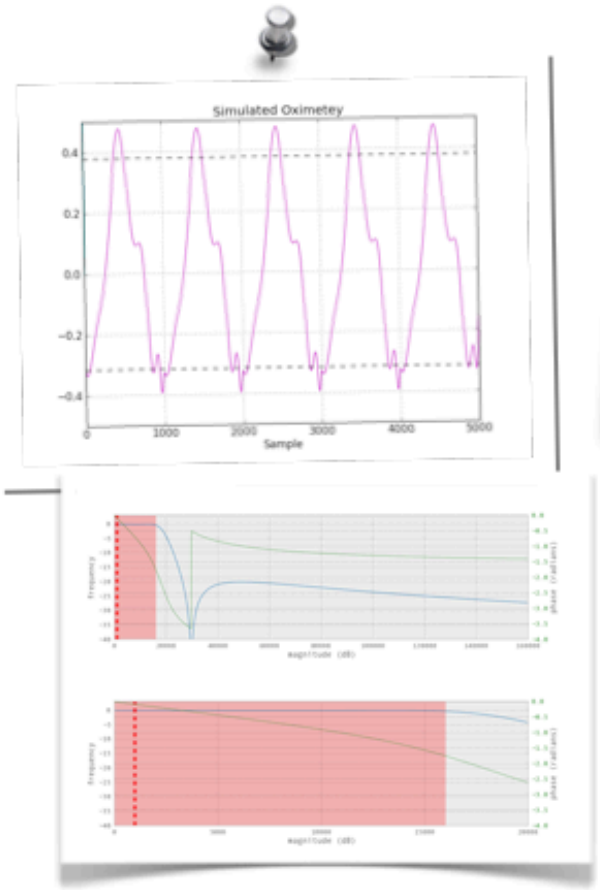
Multirate Signal Processing
Filter Design
Fourier Analysis
FFT Structures
Audio Processing
Comm DSP: Bit Syncs, PLLs, Down/Up conversion
Advanced DSProcessor Architecutres
Matlab
Python/Numpy/Scipy

Verilog / SystemVerilog
VHDL
Advanced Digital Circuit Design
Synthesis Design
Modelsim, VCS, NCsim
Modeling, Matlab, Synplify DSP

Past Work


FPGA Development:
Sole FPGA Developer
Verilog HDL
Complex System
Control Systems:
PID Motor Control
Stepper-Motor Control
DSP-FPGA:
VHDL / MyHDL
Pipelined CORDIC
NCO development
Up/Down digital and Bit-sync
Turbo-Codes and LDPC
Adaptive channel equalization
ASIC Development:
Verilog
Hearing-Aid Signal Processing
DSP:
Broadband Beamforming
Real-time DSP Assembly

Past Work



Signal Processing / Algorithms:

- Complex accelerometer (motion) data analysis
- ECG and Oximetry signal analysis
- Linear and Non-linear signal processing
- Complex signal extraction

Mixed-Signal ASIC Development:

- Verilog / VHDL / MyHDL
- \Delta\Sigma ADC, digital signal processing, decimation filter, compensation filter, low-power
- Design through tape-out
- FPGA Prototype

What is MyHDL

myhdl

Web definitions
MyHDL is a Python based hardware description language (HDL)..
en.wikipedia.org/wiki/MyHDL

- A Python package which enables hardware description
- Open-source project
- Batteries include (more on this later)

What is Python

python programming language

Web definitions

Python is a general-purpose high-level programming language whose design philosophy emphasizes code readability. Python aims to combine...

en.wikipedia.org/wiki/Python_(programming_language)

What is Python

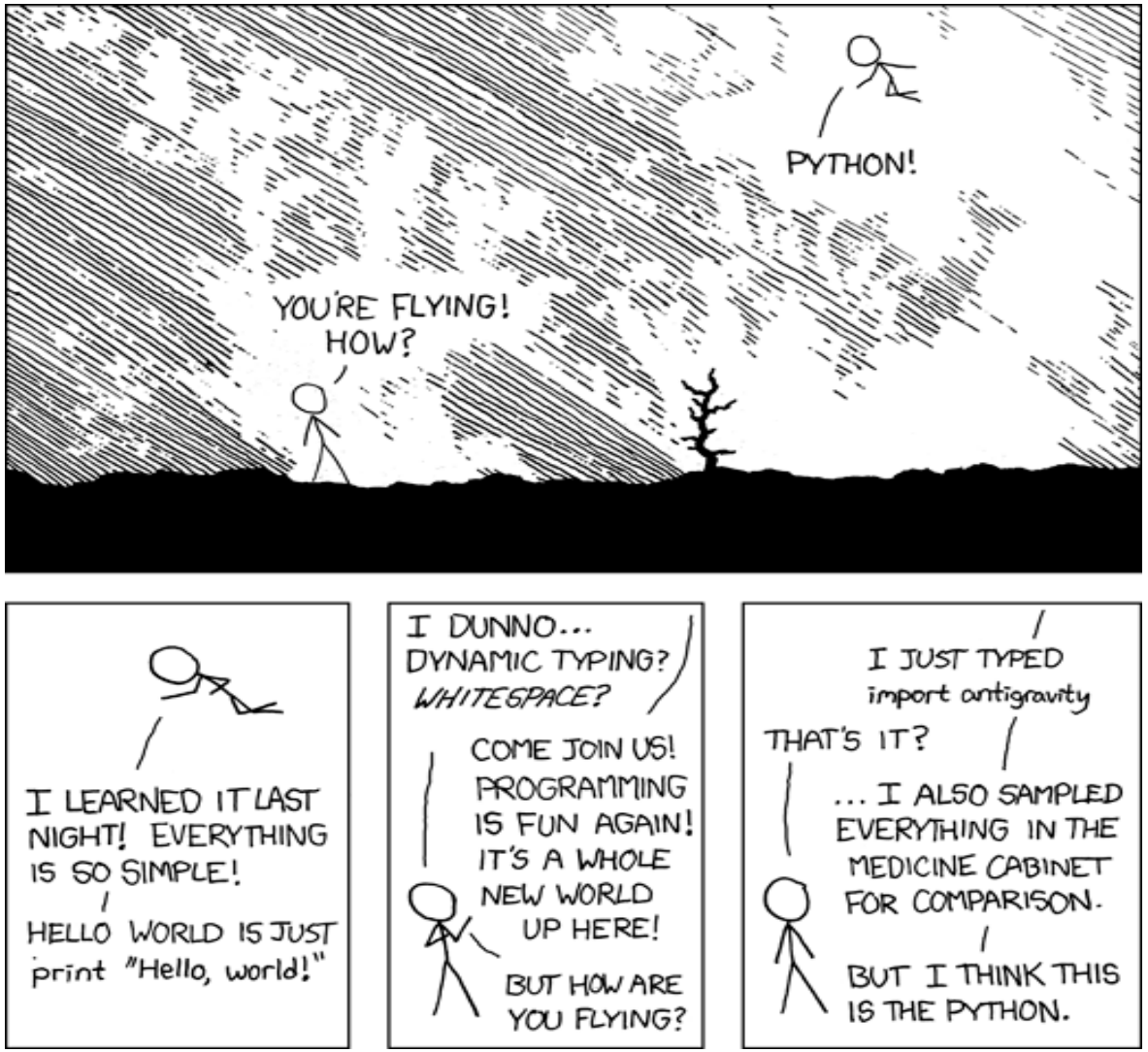
- A general purpose programming language
  - Growing in popularity
  - **Interpreted** language (bytecode-interpretive)
  - **Multi-paradigm**
    - Clean object-oriented
    - Functional – in the LISP tradition
    - Structural (procedural-imperative)
  - Extremely **readable** syntax
  - Very high-level
    - Lists
    - Dictionaries (associative arrays)
  - Extensive documentation

Serious, Who Us?

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
…
```

Antigravity

```
>> import antigravity
```



Light-hearted and Quality

- Story from Steven Levy's book *"In the Plex"* while Page and Brin were developing their first web crawler they were working with Scott Hassan. Scott moved the crawler from java to Python because Python was more stable

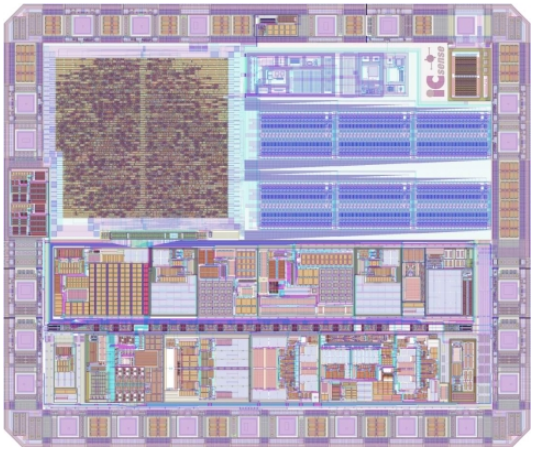
MyHDL Extends Python

- MyHDL is Python
- Using Python constructs to extend
  - Object Oriented
    - `Signal`, `intbv`
  - Generators
    - Microthread like, enables concurrency behavior
    - Resumable function that maintains state
  - Decorators
    - Meta-programming
    - Modifies a function / generator
    - `@always_seq` and `@always_comb`

Short MyHDL History

- Jan Decaluwe
  - Creator of MyHDL
  - Founder & Board Member Easic
  - Created MyHDL between 2002-2003
- First Release on SourceForge Sep 30, 2003

www.programmableplanet.com



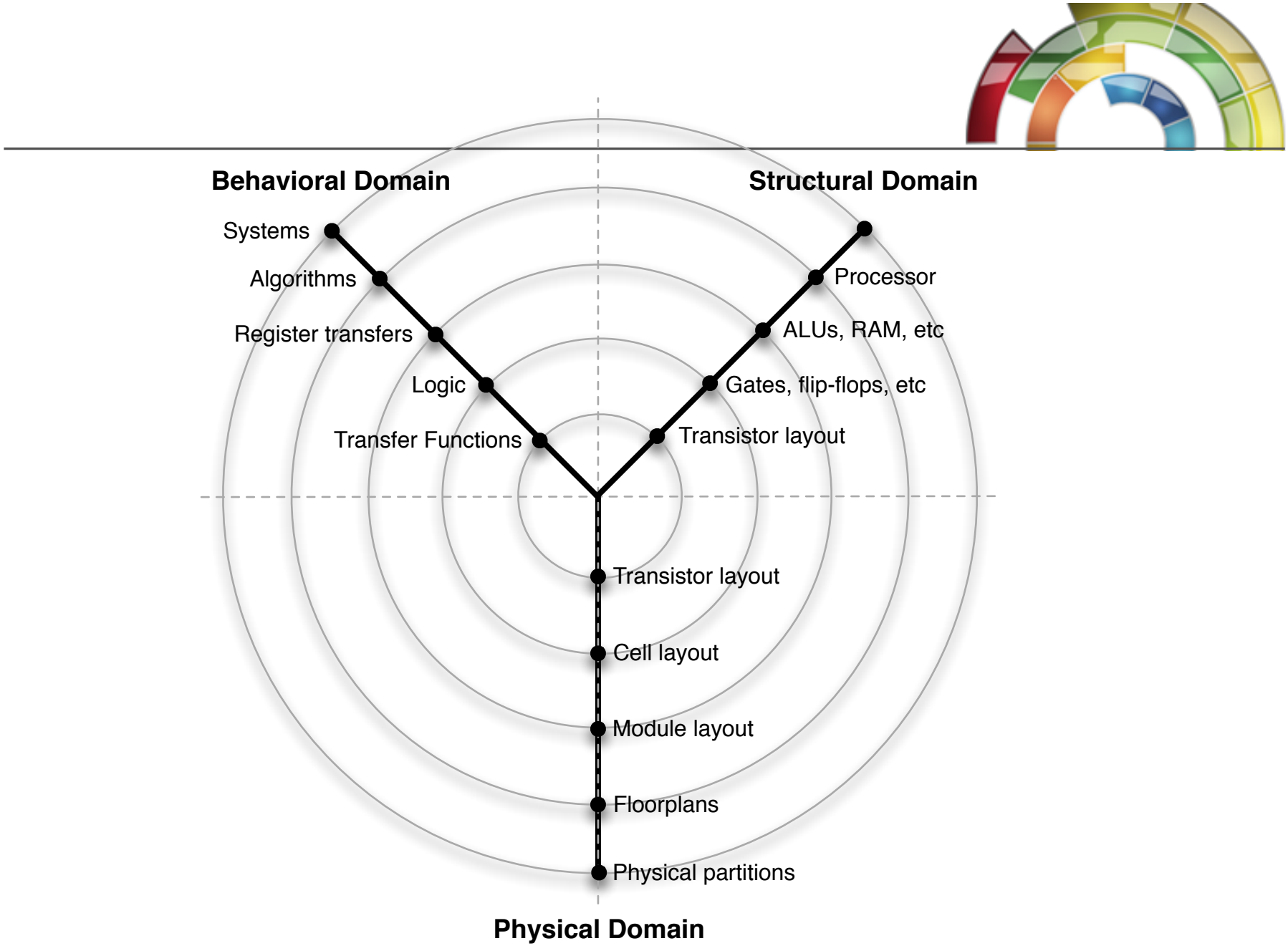
MyHDL ASIC
http://www.jandecaluwe.com/hdldesign/digmac.html

Why use MyHDL

- Manage Complex Designs
- New to Digital Hardware Design
- Scripting Languages Intensively Used
- Modern Software Development Techniques for Hardware Design
- Algorithm Development and HDL Design in a Single Environment
- Require Both Verilog and VHDL
- VHDL Too Verbose
- SystemVerilog Too Complicated
- You Been TCL'd too much

What MyHDL is **NOT**

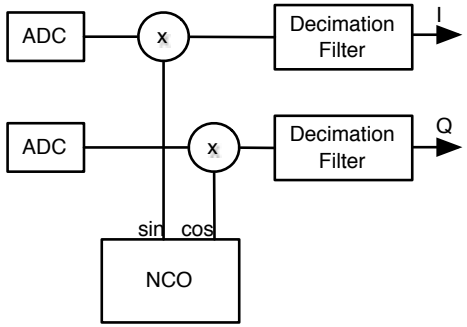
- Not arbitrary Python to silicon
- Not a radically new approach
- Not a synthesis tool
- Not an IP block library
- Not only for implementation
- Not well suited for accurate time simulation



Levels of Abstraction, Gajski and Kuhn Y-Chart

Abstraction Levels

System Level



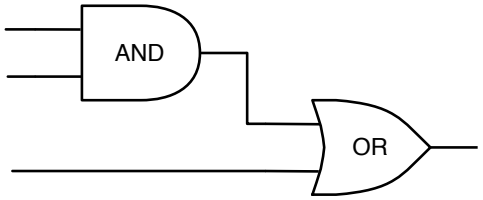
Algorithmic Level

```
def GMM(clock, reset, stream_in, stream_out):
    cnt = Signal(intbv(0, min=0, max=13))

    @always_seq(clock.posedge, reset=reset)
    def hdl():
        cnt.next = cnt + 1

    @always_streams(clock=clock, reset=reset)
    def hls():
        for f in range(0,39):
            for i in range(0,5300):
                for m in range(0,16):
                    stream_out[i][m] = var[i][f][m] * \
                                       (stream_in[f] - mean[i][f][m])**2
    # convert the hls to hdl
    hdlhls = hlsCompiler(hls)

    return hdlhls, hdl
```

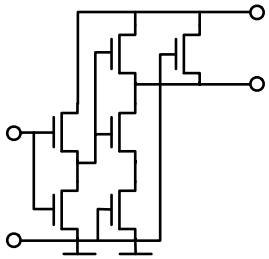
Register Transfer

```
@always_seq(clock.posedge,
            reset=reset)
 def hdl():
     sum.next = a + b
```

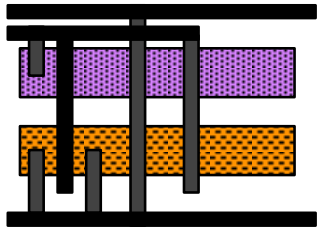
Logic Level



Transistor Level



Geometry Level



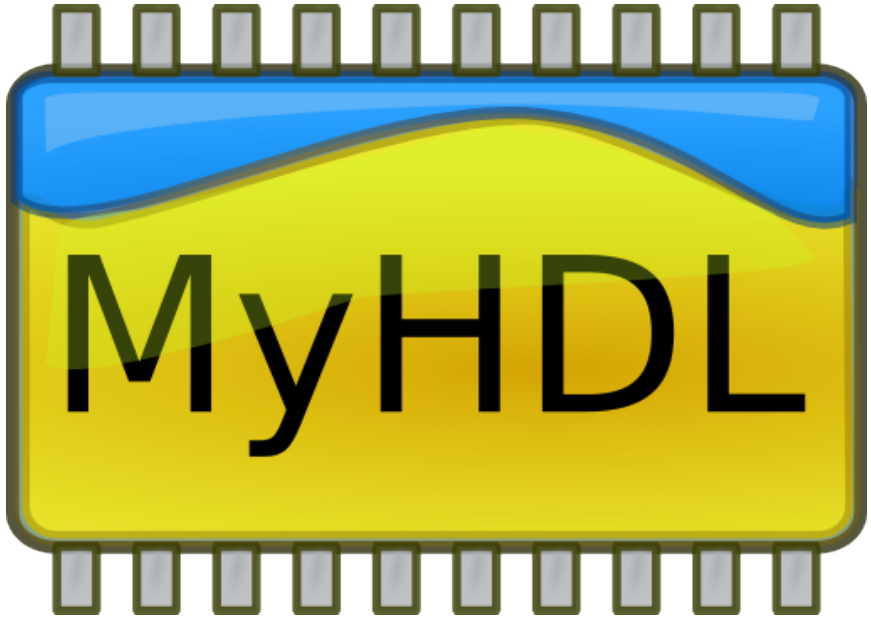
Register Transfer

- Register Transfer Level (RTL) abstraction
  - This is the commonly excepted description of mainstream HDLs: Verilog and VHDL
  - Describes the operations between registers

- MyHDL operates at the Register Transfer Level (RTL)

As Discussed

- MyHDL extends Python for hardware description



MyHDL Types

- intbv
  - Bit vector type
- Signal
  - Deterministic communication
- Convertible types
  - intbv
  - bool
  - int
  - tuple of int
  - list of bool and list of intbv

MyHDL Generators

- A Python generator is a resumable function

- Generators are the core of MyHDL
  - Provide the similar functionality as a VHDL process or Verilog always block
  - yield in a generator

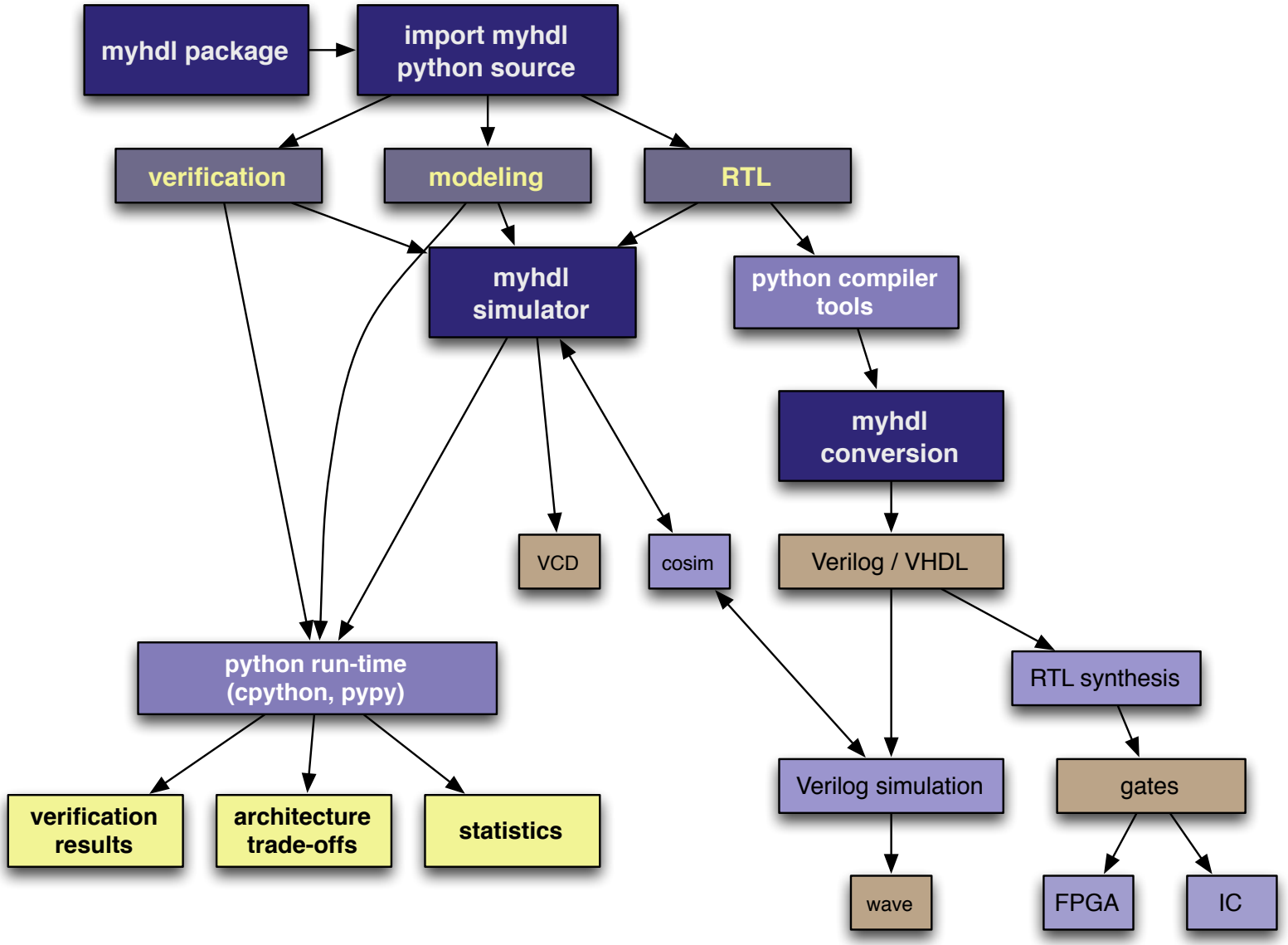
MyHDL Decorators

- MyHDL Decorators

"creates ready-to-simulate generators from local function definitions"

  - @instance
  - @always(sensitivity list)
  - @always_seq(clock,reset)
  - @always_comb

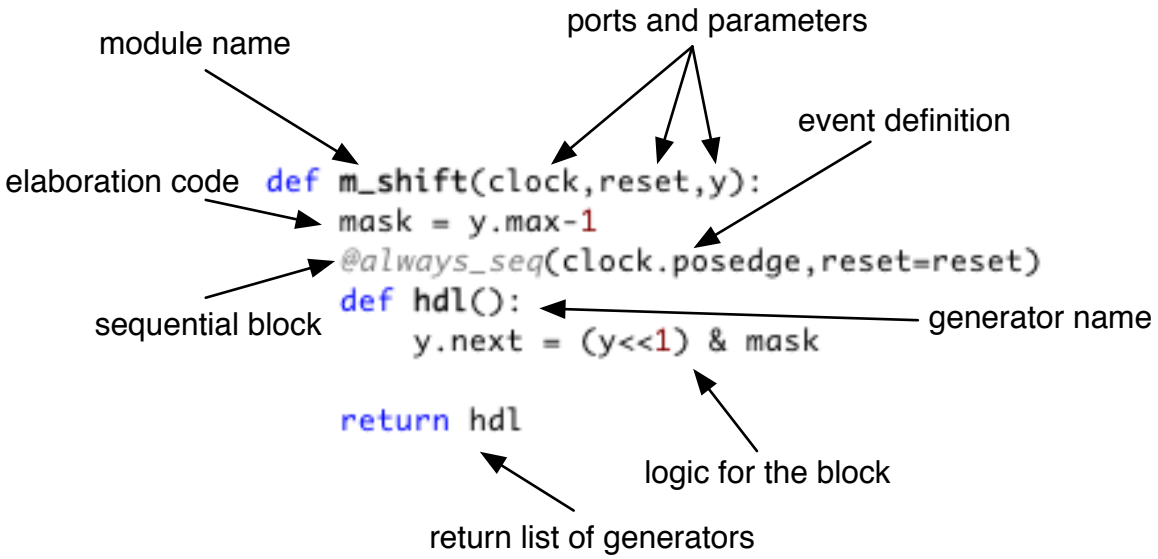
MyHDL Flow


myhdl package
import myhdl python source
verification
modeling
RTL
myhdl simulator
python compiler tools
myhdl conversion
VCD
cosim
Verilog / VHDL
python run-time (cpython, pypy)
RTL synthesis
verification results
architecture trade-offs
statistics
Verilog simulation
gates
wave
FPGA
IC

MyHDL Conversion

- MyHDL has a convertible subset
  - Convert to Verilog
  - Convert to VHDL
- Pragmatic
- Standard FPGA / ASIC flow after conversion

Anatomy of a MyHDL Module



First Example (second?)

- A counter that generates a strobe
- Small but digestible

- Counter with strobe
  - Has a clock, reset, output strobe
  - Generates a strobe every *N* ~milliseconds

```
1       def m_strober(clock, reset, strobe, ms=33):
```

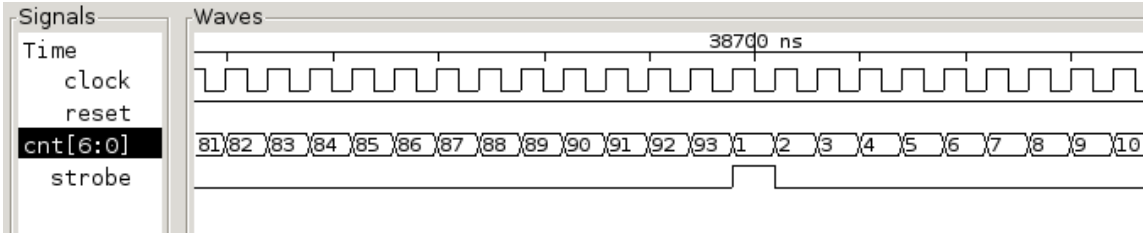
Test Driven Design

```
7       def test_strober():
8       """Test the m_strober module"""
9           ms=randint(7,111)
10          clock = Clock(0, frequency=1e3)
11          reset = Reset(0, active=0, async=False)
12          strobe = Signal(bool(0))
13
14          tb_clock = clock.gen()
15          tb_dut = traceSignals(m_strober, clock, reset, strobe, ms=ms)
16
17          @instance
18          def tb_stim():
19           yield reset.pulse(13)
20              for ii in range(113):
21                  yield delay(clock.hticks*2*ms)
22                  assert strobe == True
23
24              yield delay(23)
25              # stop simulation
26              raise StopSimulation
```

Make the Test Past

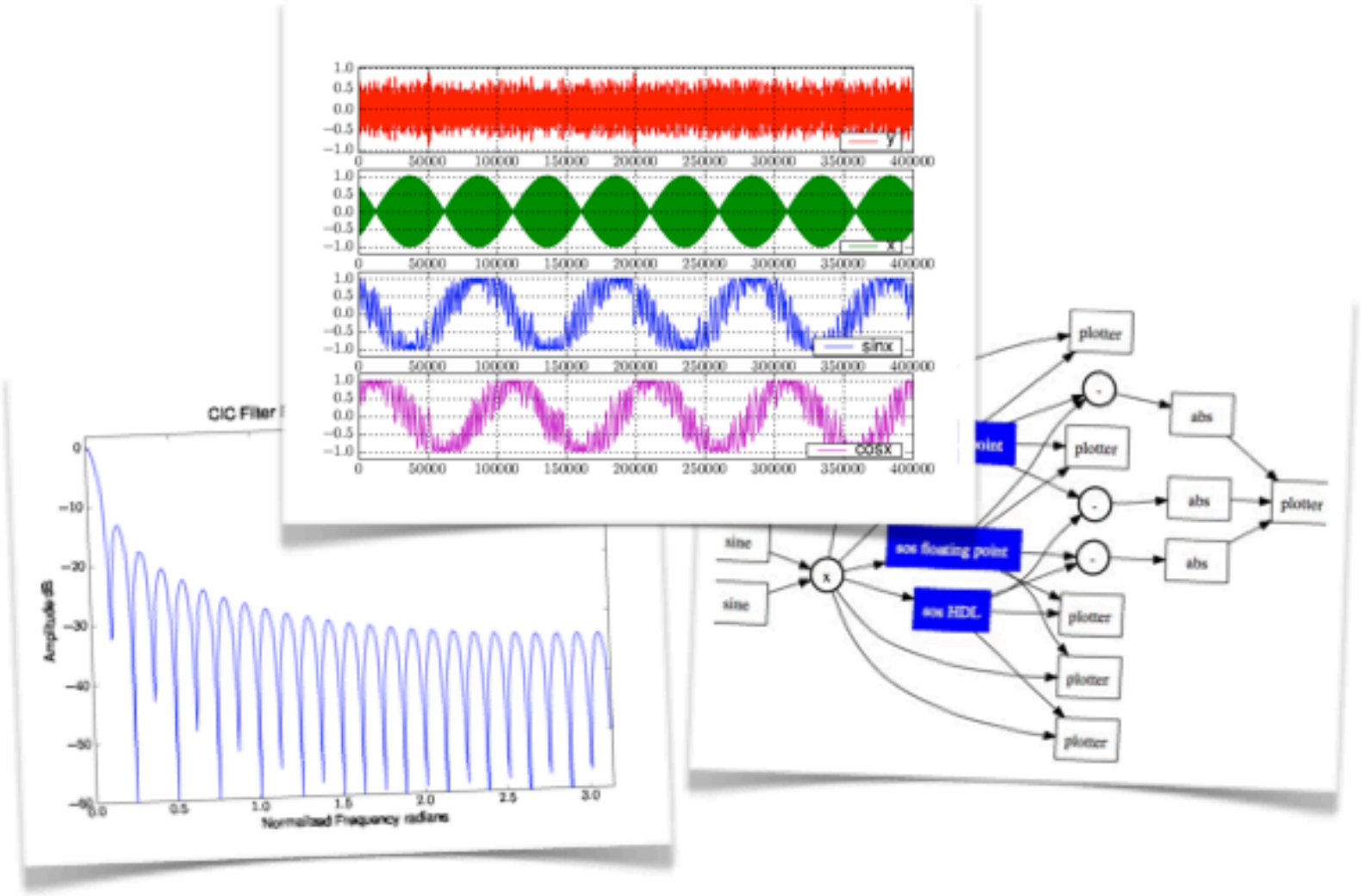
```
1         from myhdl import *
2
3         def m_strober(clock, reset, strobe, ms=333):
4             """Create a strobe every millesecond (ms)"""
5
6             max_cnt = int(round((clock.frequency/1000.)*ms))
7             cnt = Signal(intbv(1, min=0, max=max_cnt+1))
8
9             @always_seq(clock.posedge, reset=reset)
10            def hdl():
11                if cnt >= max_cnt:
12                    cnt.next = 1
13                    strobe.next = True
14                else:
15                    cnt.next = cnt + 1
16                    strobe.next = False
17
18            return hdl
```

First Example Waveforms

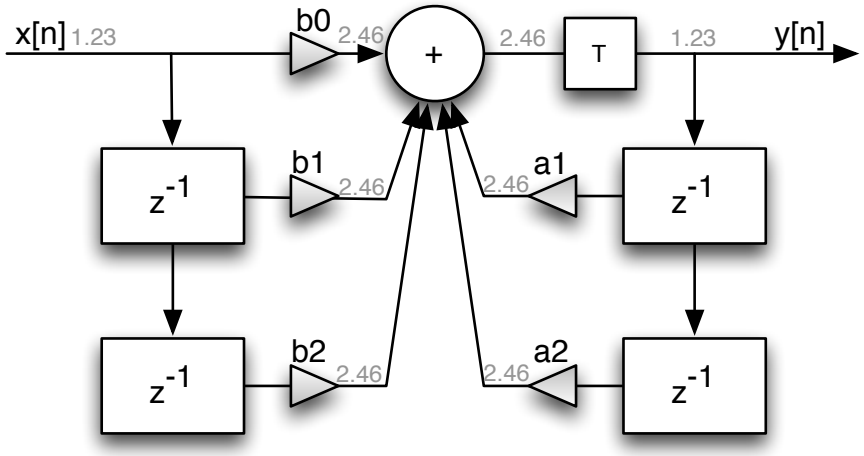



```
cfelton$ py.test test_strober.py
============= test session starts ==================
platform darwin -- Python 2.7.3 -- pytest-2.3.4
collected 1 items

test_strober.py .

============= 1 passed in 0.85 seconds =============
cfelton$
```

Ecosystem


y
x
sinx
cosx
CIC Filter
Amplitude dB
Normalized Frequency radians
plotter
abs
sine
x
sos floating point
sos HDL

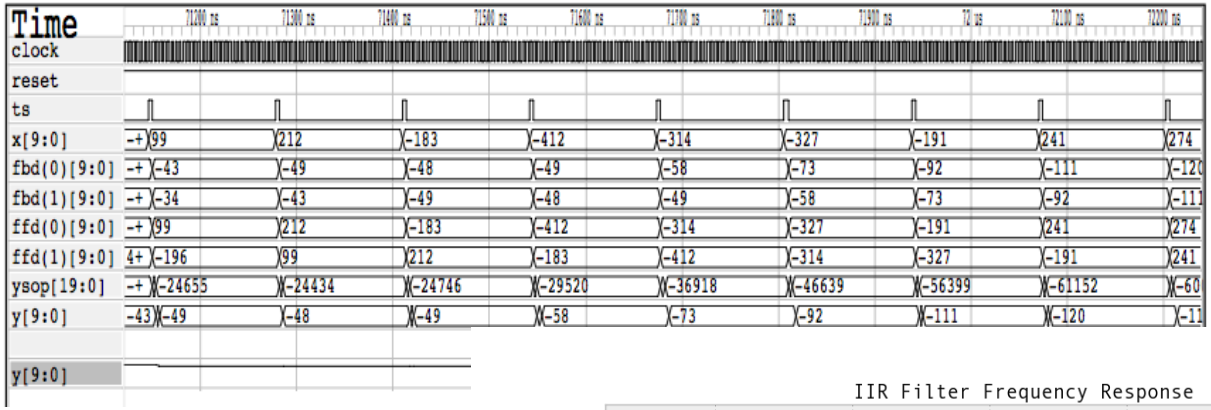
Digital Filter

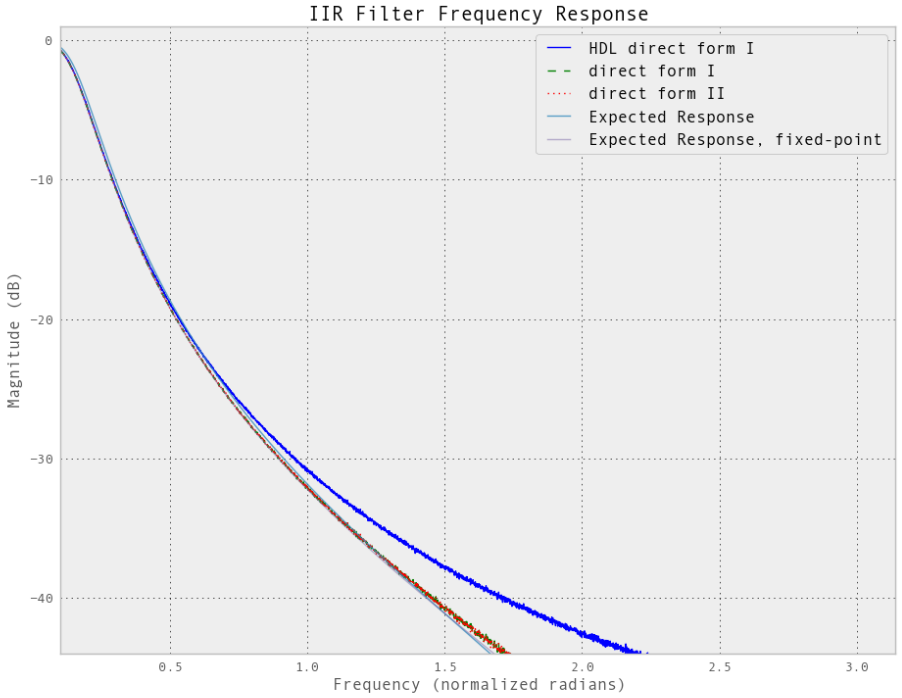


IIR Type I Digital Filter

```
1           def m_iir_type1(clock,reset,x,y,ts,B=None,A=None):
2           # make sure B and A are ints and make it a ROM (tuple of ints)
3               b0,b1,b2 = map(int,B)
4               a0,a1,a2 = map(int,A)
5
6               ffd = [Signal(intbv(0, min=x.min, max=x.max)) for ii in (0,0)]
7               fbd = [Signal(intbv(0, min=x.min, max=x.max)) for ii in (0,0)]
8               # intermidiate result, resize from here
9               ysop = Signal(intbv(0, min=dmin, max=dmax))
10
11              @always_seq(clock.posedge, reset=reset)
12              def hdl():
13                  if ts:
14                      ffd[1].next = ffd[0]
15                      ffd[0].next = x
16
17                      fbd[1].next = fbd[0]
18                      fbd[0].next = ysop//Am  # truncate (>>)
19
20                  # extra pipeline on the output at clock
21                  ysop.next = (b0*x) + (b1*ffd[0]) + (b2*ffd[1]) - \
22                              (a1*fbd[0]) - (a2*fbd[1])
23
24                  # truncate to the output word format
25                  y.next = ysop//Am  # truncate (>>)
26
27              return hdl
```

Simulation

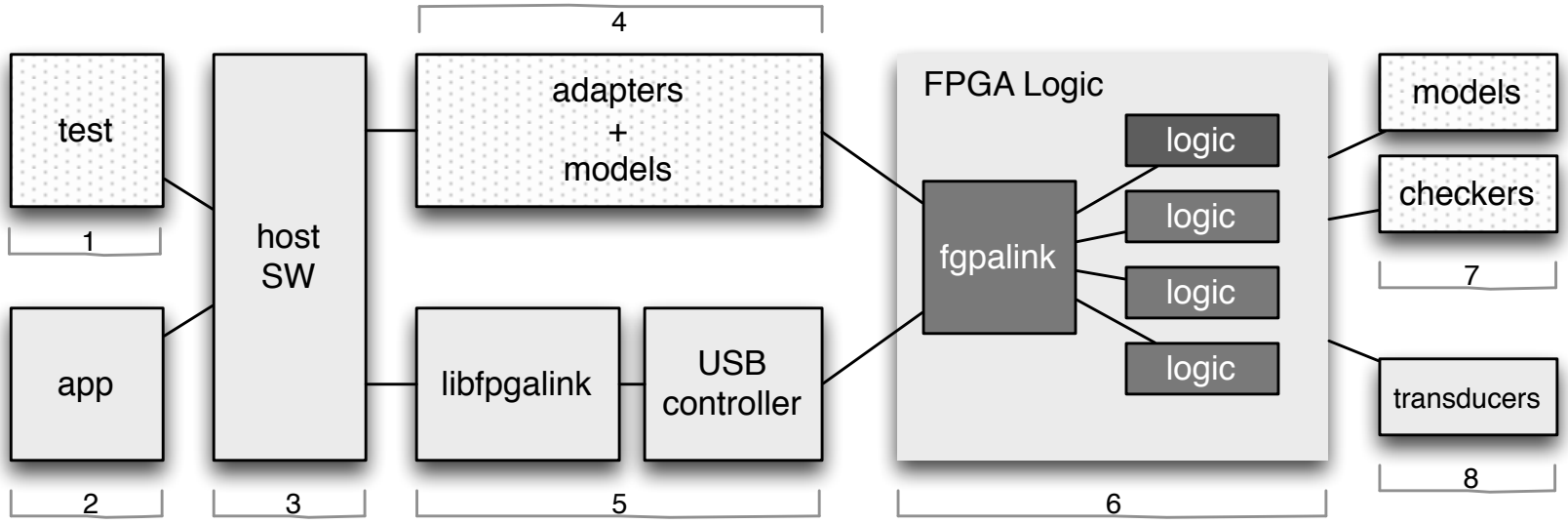

Time
clock
reset
ts
x[9:0]
fbd(0)[9:0]
fbd(1)[9:0]
ffd(0)[9:0]
ffd(1)[9:0]
ysop[19:0]
y[9:0]
y[9:0]
99 212 -183 -412 -314 -327 -191 241 274
-43 -49 -48 -49 -58 -73 -92 -111
-34 -43 -49 -48 -49 -58 -73 -92
99 212 -183 -412 -314 -327 -191 241 274
-196 99 212 -183 -412 -314 -327 -191 241
-24655 -24434 -24746 -29520 -36918 -46639 -56399 -61152
-43 -49 -48 -49 -58 -73 -92 -111 -120


IIR Filter Frequency Response
HDL direct form I
direct form I
direct form II
Expected Response
Expected Response, fixed-point
Magnitude (dB)
0
-10
-20
-30
-40
0.5
1.0
1.5
2.0
2.5
3.0
Frequency (normalized radians)

Test Frameworks

- Test frameworks are easy
- Enables new levels or reuse
- Test Driven Design (TDD)
- Existing test environments
  - py.test
  - nose
  - unittest

Example: *fpgalink*



- Test code
- Application code
- Host interface software
- Connect the host software to the DUT
- Host driver + USB controller
- FPGA logic (HDL)
- External models and checkers
- Physical transducers

https://github.com/cfelton/minnesota

Conclusion

- Python
  - Easy to learn
  - Ugly code matters
  - Batteries included
- MyHDL
  - Hardware description in Python
  - Powerful environment, ecosystem
  - Manage complexity
  - Verification simplified and fun

Resources

- www.myhdl.org 

- http://www.programmableplanet.com/
- http://www.fpgarelated.com/blogs-1/nf/Christopher_Felton.php