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
About Me

- Self proclaimed applied DSP engineer (DSP-FPGA)

- Using MyHDL 6-8 years more involved the last 4 years
Past Work

FPGA Development:
- Sole FPGA Developer
- Verilog HDL
- Complex System

Control Systems:
- PID Motor Control
- Stepper-Motor Control

ASIC Development:
- Verilog
- Hearing-Aid Signal Processing

DSP:
- Broadband Beamforming
- Real-time DSP Assembly

DSP-FPGA:
- VHDL / MyHDL
- Pipelined CORDIC
- NCO development
- Up/Down digital and Bit-sync
- Turbo-Codes and LDPC
- Adaptive channel equalization
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

- 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
>>> 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.
...

Serious, Who Us?
Antigravity

>> import antigravity

I learned it last night! Everything is so simple!
Hello world is just print "Hello, world!"

I dunno...
Dynamic typing?
Whitespace?
Come join us! Programming is fun again!
It's a whole new world up here!
But how are you flying?

I just typed import antigravity
That's it?

... I also sampled everything in the medicine cabinet for comparison.
But I think this is the Python.
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

Behavioral Domain
- Systems
- Algorithms
- Register transfers
- Logic
- Transfer Functions

Structural Domain
- Processor
- ALUs, RAM, etc
- Gates, flip-flops, etc
- Transistor layout

Physical Domain
- Physical partitions
- Floorplans
- Module layout
- Cell layout
- Transistor layout
- Physical partitions
Abstraction Levels

System Level

Algorithmic Level

Register Transfer

Logic Level

Transistor Level

Geometry Level

```python
@always_seq(clock.posedge, reset=reset)
def hdl():
    sum.next = a + b
```
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**
- **myhdl simulator**
- **python run-time (cpython, pypy)**
- **verification results**
- **architecture trade-offs**
- **statistics**
- **modeling**
- **RTL**
- **myhdl conversion**
- **Verilog / VHDL**
- **Verilog simulation**
- **RTL synthesis**
- **gates**
- **FPGA**
- **IC**
- **VCD**
- **cosim**
- **python compiler tools**

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

module name

ports and parameters

event definition

generator name

elaboration code

sequential block

logic for the block

return list of generators

def m_shift(clock, reset, y):
    mask = y.max - 1
    @always_seq(clock.posedge, reset=reset)
    def hdl():
        y.next = (y<<1) & mask
    return hdl
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 $N \sim$milliseconds

```python
1 def m_strober(clock, reset, strobe, ms=33):
```
def test_strober():
    
    """Test the m_strober module"""
    ms = randint(7, 111)
    clock = Clock(0, frequency=1e3)
    reset = Reset(0, active=0, async=False)
    strobe = Signal(bool(0))
    
    tb_clock = clock.gen()
    tb_dut = traceSignals(m_strober, clock, reset, strobe, ms=ms)

@instance
def tb_stim():
    yield reset.pulse(13)
    for ii in range(113):
        yield delay(clock.hticks * 2 * ms)
        assert strobe == True

    yield delay(23)
    # stop simulation
    raise StopSimulation
from myhdl import *

def m_strober(clock, reset, strobe, ms=333):
    """Create a strobe every millisecond (ms)""

    max_cnt = int(round((clock.frequency/1000.)*ms))
    cnt = Signal(intbv(1, min=0, max=max_cnt+1))

    @always_seq(clock.posedge, reset=reset)
    def hdl():
        if cnt >= max_cnt:
            cnt.next = 1
            strobe.next = True
        else:
            cnt.next = cnt + 1
            strobe.next = False

    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
Digital Filter

\[ x[n] \quad b_0 \quad + \quad T \quad y[n] \]

\[ z^{-1} \quad b_1 \quad a_1 \quad z^{-1} \quad b_2 \quad a_2 \quad z^{-1} \]
IIR Type I Digital Filter

```python
def m_iir_type1(clock, reset, x, y, ts, B=None, A=None):
    # make sure B and A are ints and make it a ROM (tuple of ints)
    b0, b1, b2 = map(int, B)
    a0, a1, a2 = map(int, A)

    ffd = [Signal(intbv(0, min=x.min, max=x.max)) for ii in (0, 0)]
    fbd = [Signal(intbv(0, min=x.min, max=x.max)) for ii in (0, 0)]
    # intermediate result, resize from here
    ysop = Signal(intbv(0, min=dmin, max=dmax))

    @always_seq(clock.posedge, reset=reset)
    def hdl():
        if ts:
            ffd[1].next = ffd[0]
            ffd[0].next = x

            fbd[1].next = fbd[0]
            fbd[0].next = ysop // Am  # truncate (>>)

        # extra pipeline on the output at clock
        ysop.next = (b0 * x) + (b1 * ffd[0]) + (b2 * ffd[1]) - 
                    (a1 * fbd[0]) - (a2 * fbd[1])

        # truncate to the output word format
        y.next = ysop // Am  # truncate (>>)

    return hdl
```
Simulation
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

[GitHub Repository](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/