RTL design in python:  
porting the mMIPS  

Jos Huiskens  
June 25th, 2013
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
What is MyHDL?

myhdl

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

• A Python package which enables hardware description
• Open-source project
• Batteries included (more on this later)
MyHDL Extends Python

- MyHDL is Python
- Using Python constructs to extend
  - Object Oriented
    - Signal, intbv
  - Generators
    - Micro-thread like, enables concurrency behavior
    - Resumable functions that maintain state
  - Decorators
    - Meta-programming
    - “Macro” mechanism
    - Modifies a function / generator
      - @always_seq and @always_comb
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
Abstraction Levels

System Level

Algorithmic Level

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 expected description of mainstream HDLs: Verilog and VHDL
  • Describes the operations between registers

• MyHDL operates at the Register Transfer Level (RTL)

• MyHDL extends Python for hardware description
MyHDL Types

• **intbv**
  • Bit vector type, an integer with bit vector properties

• **Signal**
  • Deterministic communication, see it as a VHDL signal

• **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
# function version

```python
7 # function version
8 def fibon(n):
9     a = b = 1
10    result = []
11    for i in xrange(n):
12        result.append(a)
13        a, b = b, a + b
14    return result

15 fibon(8)
16 [1, 1, 2, 3, 5, 8, 13, 21]
```

# generator version

```python
7 # generator version
8 def fibon(n):
9     a = b = 1
10    result = []
11    for i in xrange(n):
12        yield a
13        a, b = b, a + b

14 for x in fibon(8):
15    print x,
16 [1, 1, 2, 3, 5, 8, 13, 21]
```
MyHDL Decorators

• MyHDL Decorators
  “creates ready-to-simulate generators from local function definitions”

  • @instance
  • @always(sensitivity list)
  • @always_seq(clock, reset)
  • @always_comb
Python decorator

A decorator is a function (mydecorator) that takes a function object as an argument, and returns a function object as a return value.

@mydecorator: just syntactic sugar
MyHDL Flow

import myhdl
python source

myhdl simulator

myhdl package

verification

modeling

RTL

Verilog / VHDL

myhdl conversion

python compiler tools

VCD

cosim

python run-time (cpython, pypy)

verification results

architecture trade-offs

statistics

Verilog simulation

RTL synthesis

gates

FPGA

IC

wave

modeling

RTV synthesis

gates

FPGA

IC

MyHDL Flow

import myhdl
python source

myhdl simulator

myhdl package

verification

modeling

RTL

Verilog / VHDL

myhdl conversion

python compiler tools

VCD

cosim

python run-time (cpython, pypy)

verification results

architecture trade-offs

statistics

Verilog simulation

RTL synthesis

gates

FPGA

IC

wave
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
- elaboration code
- sequential block
- event definition
- generator name
- logic for the block
- return list of generators

```python
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
```
Simple adder in myhdl

```python
from myhdl import *
from mips import VERBOSE, DWORD

def add(a, b, r):
    """adder which computes r = a + b """
    @always_comb
    def doadd():
        r.next = a + b

    return doadd

def __toV():
    a = Signal(intbv(24)[DWORD:]).next = not clk
    b = Signal(intbv(24)[DWORD:]).next = not clk
    clk = Signal(bool(0)).next = not clk
    r = Signal(intbv(0)[DWORD:]).next = not clk
    aa = toVerilog(add, a, b, r)

if __name__ == "__main__":
    def testbench():
        a = Signal(intbv(24)[8:])
        b = Signal(intbv(24)[8:])
        clk = Signal(bool(0))
        r = Signal(intbv(24)[8:]).next = not clk

        halfPeriod = delay(5)

        @always(halfPeriod)
        def drclk():
            clk.next = not clk

        @always(clk.posedge)
        def stim():
            from random import randrange
            a.next = randrange(15)
            b.next = randrange(15)

            i_add = add(a, b, r)

            return drclk, stim, i_add

    tb = traceSignals(testbench)
    sim = Simulation(tb)
    sim.run(100)
    __toV()
```
from myhdl import *

def register(inp, w, out, rst, clk):
    ''' REGISTER
    '''
    @always(clk.posedge, rst.posedge)
    def do_reg():
        if rst == 1:
            out.next = 0
        elif w == 1:
            out.next = inp
        else:
            pass

    return do_reg

if __name__ == '__main__':
    w = Signal(bool(0))
    clk = Signal(bool(0))
    rst = Signal(bool(0))
    for w in [1, 5, 6, 8, 16]:
        inp = Signal(intbv(0)[w:]
        out = Signal(intbv(0)[w:]
        reg_inst = toVerilog(register, inp, w, out, rst, clk)
        print "register_%d generated" % (w, )
ALU waveform, using gtkwave

```
$ python alu.py
<class 'myhdl.StopSimulation'>: No more events
alu generated
$ gtkwave alu.vcd
```
Verilog co-simulation

- Icarus Verilog
- Cadence ncsim
- Cadence ncsim, CMOS90 netlist

Select icarus/verilog implementation:

```python
def add(aa, b, r):
    path = "/home/huisken/local/iverilog-0.9g/
    cmd = path + 'bin/iverilog -o add add.v tb_add.v'
    print "==", cmd
    os.system(cmd)
    cmd = path + 'bin/vvp -m ./myhdl.vpi -M' + path + 'lib/ivl add'
    print "==", cmd
    return Cosimulation(cmd, a=aa, b=b, r=r)
```
module add (a, b, r);
// adder which computes r = a + b

input [31:0] a;
input [31:0] b;
output [31:0] r;
wire [31:0] r;

assign r = (a + b);
endmodule
mMIPS in MyHDL

Single cycle mini-mini MIPS

```
$ ~/src/mMips/mMips/single_cycle_mMips> wc -l *.py
51 add.py
55 aluctrl.py
89 alu.py
28 and_mips.py
99 ctrl.py
42 decoder.py
20 __init__.py
48 mips.py
41 mux.py
130 ram.py
109 registerfile.py
32 register.py
85 rom.py
204 sc_mMips.py
37 shift.py
48 signextend.py
34 xilinx_bram.py
1152 total

$ ~/src/mMips/mMips/single_cycle_mMips>
```

Multi cycle mini MIPS

```
$ ~/src/mMips/mMips/mMips> wc -l *.py
58 add.py
101 aluCtrl.py
147 alu.py
43 branch.py
255 registerfile16.py
388 ctrl.py
68 decoder.py
53 dev.py
194 hazard.py
62 imm2word.py
29 __init__.py
153 memdev.py
50 mips.py
735 mmips.py
64 mux.py
162 ram.py
39 register.py
123 rom.py
41 shift.py
78 signextend.py
171 testbench.py
147 xlxram.py
3451 total

$ ~/src/mMips/mMips/mMips>
```
Ecosystem

import pylab
import matplotlib
Digital Filter

\[ x[n] \rightarrow b_0 \rightarrow + \rightarrow b_1 \rightarrow b_2 \rightarrow a_1 \rightarrow a_2 \rightarrow y[n] \]

\[ z^{-1} \]

\[ a_1 \]

\[ a_2 \]

\[ b_0 \]

\[ b_1 \]

\[ b_2 \]

\[ T \]

\[ 1.23 \]

\[ 2.46 \]
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

**Time**

<table>
<thead>
<tr>
<th>Time</th>
<th>0.000s</th>
<th>1.000s</th>
<th>2.000s</th>
<th>3.000s</th>
<th>4.000s</th>
<th>5.000s</th>
<th>6.000s</th>
<th>7.000s</th>
<th>8.000s</th>
<th>9.000s</th>
<th>10.000s</th>
<th>11.000s</th>
<th>12.000s</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[9:0]</td>
<td>-103</td>
<td>212</td>
<td>-163</td>
<td>-122</td>
<td>-314</td>
<td>-227</td>
<td>-191</td>
<td>241</td>
<td>274</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fbd(0)[9:0]</td>
<td>33</td>
<td>49</td>
<td>58</td>
<td>73</td>
<td>92</td>
<td>111</td>
<td>122</td>
<td>111</td>
<td>122</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fbd(1)[9:0]</td>
<td>-7</td>
<td>-110</td>
<td>-111</td>
<td>-112</td>
<td>-113</td>
<td>-114</td>
<td>-115</td>
<td>-116</td>
<td>-117</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**IIR Filter Frequency Response**

- HDL direct form I
- direct form I
- direct form II
- Expected Response
- Expected Response, fixed-point

**Python**

```python
import matplotlib.pyplot as plt

# Example code snippet
plt.plot(frequency, magnitude, label='HDL direct form I')
plt.plot(frequency, magnitude2, label='direct form I')
plt.plot(frequency, magnitude3, label='direct form II')
plt.plot(frequency, expected_response, label='Expected Response')
plt.plot(frequency, expected_response_fixed_point, label='Expected Response, fixed-point')
plt.xlabel('Frequency (normalized radians)')
plt.ylabel('Magnitude (dB)')
plt.legend()
plt.show()
```
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
  • Batteries included

• MyHDL
  • Hardware description in Python
  • Powerful environment, ecosystem
  • Verification simplified and fun

• mMIPS
  • Ported from SystemC
  • Simulated and Co-Simulated, not fully verified
  • Synthesized: ISE (untested) and Cadence rc (co-simulated)

• Python: allows (...) matlab style interaction
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
Acknowledgement

• You guys
  • for hosting and providing the mMips

• Christopher Felton
  • Several slides came from him!
Resources

• http://www.myhdl.org

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

• And an invitation to join (still) password protected: https://huisken@bitbucket.org/huisken/mmips

?? Any Further Questions ??