From Python to Silicon
python-myhdl

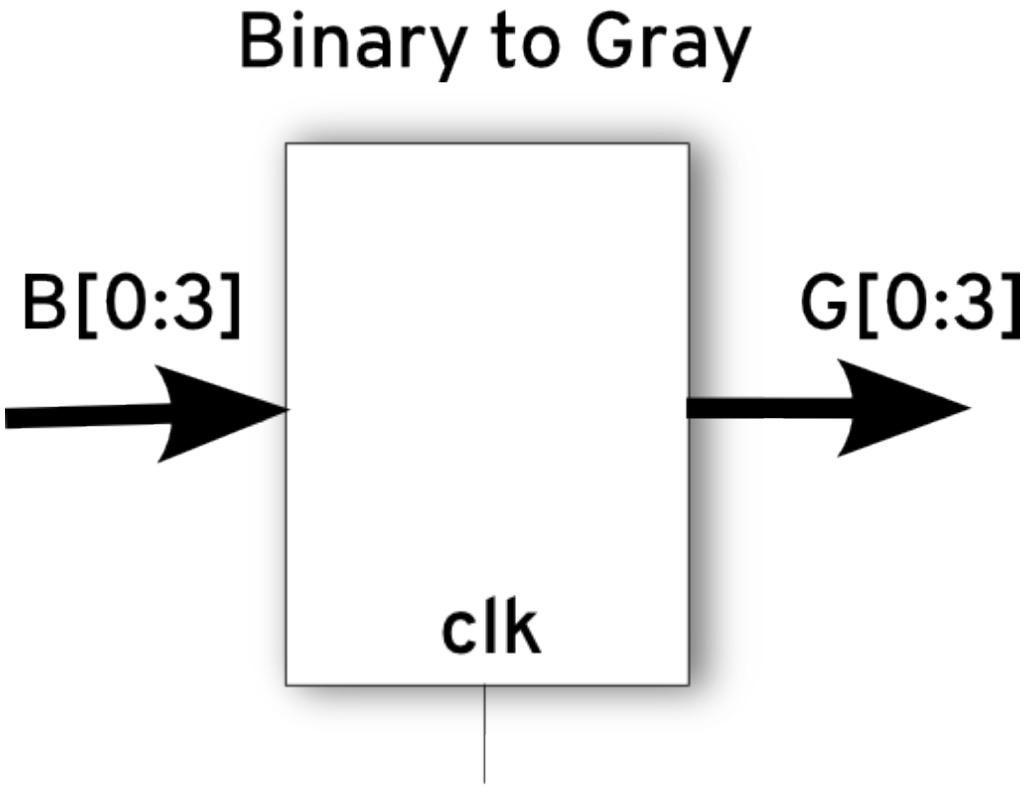
Jan Decaluwe Shakthi Kannan

Version 1.3
September 2011

jan@jandecaluwe.com
@jandecaluwe
http://jandecaluwe.com

shakthimaan@gmail.com
@shakthimaan
http://shakthimaan.com

Hardware Module



Generators

```
def generator():
    for i in range(5):
        yield i
```

```
>>> g = generator()
>>> g.next()
0
>>> g.next()
1
>>> g.next()
2
```

Decorators

```
def func(arg1, arg2, ...):
    <body>
func = deco(func)
```

```
@deco
def func(arg1, arg2, ...):
    <body>
```

Decorators (2)

```
def entryExit(f):
    def new_f():
        print "Entering", f.__name__
        f()
        print "Exited", f.__name__
    return new_f

@entryExit
def func1():
    print "inside func1()"

func1()
```

```
$ python decorator.py

Entering func1
inside func1()
Exited func1
```

Hello World

```
from myhdl import Signal, delay, always, now, Simulation

def HelloWorld():

    interval = delay(10)

    @always(interval)
    def sayHello():
        print "%s Hello World!" % now()

    return sayHello


inst = HelloWorld()
sim = Simulation(inst)
sim.run(30)
```

Hello World (2)

```
$ python hello.py

10 Hello World!
20 Hello World!
30 Hello World!
_SuspendSimulation: Simulated 30 timesteps
```

Signals, Ports and Concurrency

```
from myhdl import Signal, delay, always, now, Simulation

def ClkDriver(clk):

    halfPeriod = delay(10)

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

    return driveClk

def HelloWorld(clk):

    @always(clk.posedge)
    def sayHello():
        print "%s Hello World!" % now()

    return sayHello
```

Signals, Ports and Concurrency (2)

```
clk = Signal(0)
clkdriver_inst = ClkDriver(clk)
hello_inst = HelloWorld(clk)
sim = Simulation(clkdriver_inst, hello_inst)
sim.run(50)
```

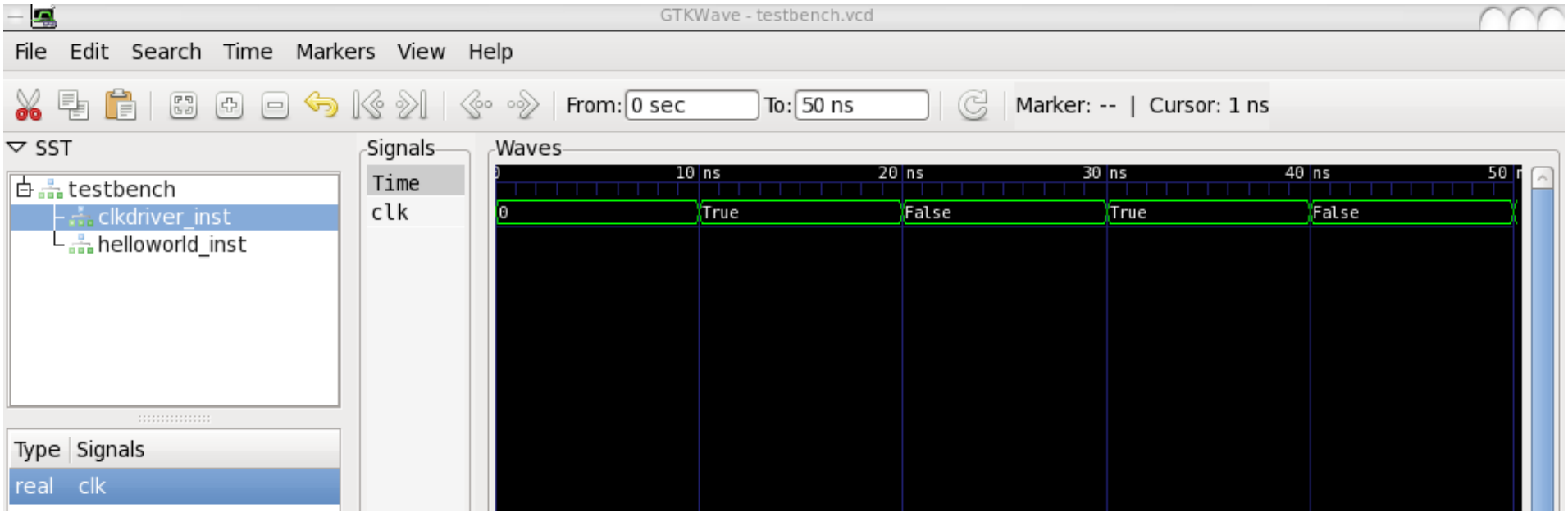
```
$ python hello2.py

10 Hello World!
30 Hello World!
50 Hello World!
_SuspendSimulation: Simulated 50 timesteps
```

GTKWave Analyzer

```
def testbench():
    clk = Signal(0)

    clkdriver_inst = ClkDriver(clk)
    helloworld_inst = HelloWorld(clk)

    return clkdriver_inst, helloworld_inst

tb_fsm = traceSignals(testbench)
sim = Simulation(tb_fsm)
sim.run(50)
```

```
$ python hello3.py

10 Hello World!
30 Hello World!
50 Hello World!
<class 'myhdl._SuspendSimulation'>: Simulated 50 timesteps
```

GTKWave Analyzer (2)



Bit Oriented Operations

```
intbv([val=0] [, min=0]   [, max=0])
```

```
>>> a = intbv(24, min=0, max=25)
```

```
>>> a.min
0
>>> a.max
25
>>> len(a)
5
```

Bit Oriented Operations (2)

```
>>> a = intbv(24)[5:]
```

```
>>> a.min
0
>>> a.max
32
>>> len(a)
5
```

```
>>> a = intbv(6, min=0, max=7)
>>> len(a)
3
>>> a = intbv(6, min=-3, max=7)
>>> len(a)
4
```

Gray Code

Width: 3

Decimal	Binary	Gray
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

XOR Truth Table

A	B	A $\oplus$ B
0	0	0
0	1	1
1	0	1
1	1	0

Gray Code (2)

Width: 3

Decimal	Binary	Gray
0	0000	0

Gray Code (2)

Width: 3

Decimal	Binary	Gray
0	0000	0
0	0000	00

Gray Code (2)

Width: 3

Decimal	Binary	Gray
0	0000	0
0	0000	00
0	0000	000

Gray Code (2)

Width: 3

Decimal	Binary	Gray
0	0000	0
0	0000	00
0	0000	000
1	0001	1

Gray Code (2)

Width: 3

Decimal	Binary	Gray
0	0000	0
0	0000	00
0	0000	000
1	0001	1
1	0001	01

Gray Code (2)

Width: 3

Decimal	Binary	Gray
0	0000	0
0	0000	00
0	0000	000
1	0001	1
1	0001	01
1	0001	001

Gray Code (2)

Width: 3

Decimal	Binary	Gray
0	0000	0
0	0000	00
0	0000	000
1	0001	1
1	0001	01
1	0001	001
2	0010	1

Gray Code (2)

Width: 3

Decimal	Binary	Gray
0	0000	0
0	0000	00
0	0000	000
1	0001	1
1	0001	01
1	0001	001
2	0010	1
2	0010	11

Gray Code (2)

Width: 3

Decimal	Binary	Gray
0	0000	0
0	0000	00
0	0000	000
1	0001	1
1	0001	01
1	0001	001
2	0010	1
2	0010	11
2	0010	011

Bit Indexing

```
from myhdl import *

def bin2gray(B, G, width):
    @always_comb
    def logic():
        for i in range(width):
            G.next[i] = B[i+1] ^ B[i]

    return logic
```

Bit Indexing (2)

```
def testBench(width):

    B = Signal(intbv(0))
    G = Signal(intbv(0))

    dut = bin2gray(B, G, width)

    @instance
    def stimulus():
        for i in range(2**width):
            B.next = intbv(i)
            yield delay(10)
            print "B: " + bin(B, width) +
                        "| G: " + bin(G, width)

    return dut, stimulus
```

Bit Indexing (3)

```
sim = Simulation(testBench(width=3))
sim.run()
```

```
$ python bin2gray.py
B: 000 | G: 000
B: 001 | G: 001
B: 010 | G: 011
B: 011 | G: 010
B: 100 | G: 110
B: 101 | G: 111
B: 110 | G: 101
B: 111 | G: 100
StopSimulation: No more events
```

Bit Slicing

```
>>> a = intbv(-3)
>>> bin(a, width=5)
'11101'
>>> b = a[5:]
>>> b
intbv(29L)
>>> bin(b)
'11101'
```

Bit Unsigned Representation

```
>>> a = intbv(12, min=0, max=16)
>>> bin(a)
'1100'
>>> a.min
0
>>> a.max
16
```

Bit Signed Representation

```
>>> a = intbv(-12, min=0, max=16)

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  ...
ValueError: intbv value -12 < minimum 0
```

```
>>> a = intbv(-12, min=16, max=16)

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  ...
ValueError: intbv value -12 < minimum 16
```

Bit Unsigned and Signed Representation

```
>>> a = intbv(12, min=0, max=16)
>>> bin(a)
'1100'
>>> b = a.signed()
>>> b
-4
>>> bin(b, width=4)
'1100'
```

Unit Tests

```
class TestGrayCodeProperties(unittest.TestCase):

    def testSingleBitChange(self):
        def test(B, G, width):
            B.next = intbv(0)
            yield delay(10)
            for i in range(1, 2**width):
                G_Z.next = G
                B.next = intbv(i)
                yield delay(10)
                diffcode = bin(G ^ G_Z)
                self.assertEqual(diffcode.count('1'), 1)
```

Unit Tests (2)

```
        for width in range(1, MAX_WIDTH):
            B = Signal(intbv(-1))
            G = Signal(intbv(0))
            G_Z = Signal(intbv(0))
            dut = bin2gray(B, G, width)
            check = test(B, G, width)
            sim = Simulation(dut, check)
            sim.run(quiet=1)
```

```
$ python test_gray.py -v
Check that only one bit changes in successive codewords ... ok

----------------------------------------------------------------------
Ran 1 tests in 3.12s

OK
```

Structural Modelling

```
def top(...):
    ...
    instance_1 = module_1(...)
    instance_2 = module_2(...)
    ...
    instance_n = module_n(...)
    ...
    return instance_1, instance_2, ... , instance_n
```

```
from myhdl import instances

def top(...):
    ...
    instance_1 = module_1(...)
    instance_2 = module_2(...)
    ...
    instance_n = module_n(...)
    ...
    return instances()
```

Conditional Instantiation

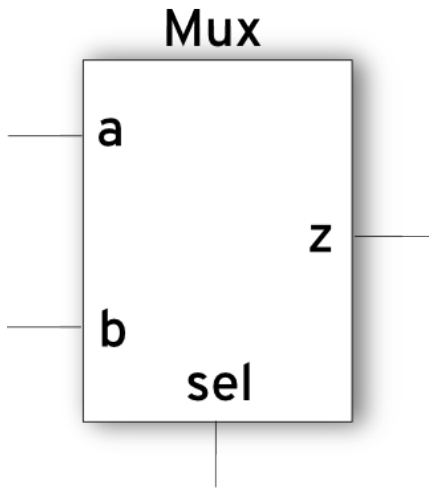
```
SLOW, MEDIUM, FAST = range(3)

def top(..., speed=SLOW):
    ...
    def slowAndSmall():
        ...
    ...
    def fastAndLarge():
        ...
    if speed == SLOW:
        return slowAndSmall()
    elif speed == FAST:
        return fastAndLarge()
    else:
        raise NotImplementedError
```

List of Instances and Signals

```
def top(...):

    din = Signal(0)
    dout = Signal(0)
    clk = Signal(bool(0))
    reset = Signal(bool(0))

    channel_inst = channel(dout, din, clk, reset)

    return channel_inst
```

RTL Modelling: Combinatorial Logic

```
from myhdl import Signal, Simulation, delay, always_comb

def Mux(z, a, b, sel):
    @always_comb
    def muxLogic():
        if sel == 1:
            z.next = a
        else:
            z.next = b

    return muxLogic
```



RTL Modelling: Sequential Logic

```
def top(<parameters>, clock, ..., reset, ...):
    ...
    @always(clock.posedge, reset.negedge)
    def seqLogic():
        if reset == <active level>:
            <reset code>
        else:
            <functional code>
    ...
    return seqLogic, ...
```

RTL Modelling: Finite State Machine

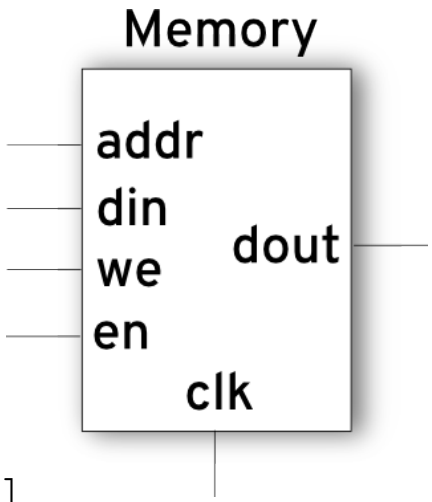
```
>>> SEARCH, CONFIRM, SYNC = range(3)
>>> CONFIRM
1
```

```
>>> from myhdl import enum
>>> t_State = enum('SEARCH', 'CONFIRM', 'SYNC')
>>> t_State
<Enum: SEARCH, CONFIRM, SYNC>
>>> t_State.CONFIRM
CONFIRM
```

```
state = Signal(t_State.SEARCH)
```

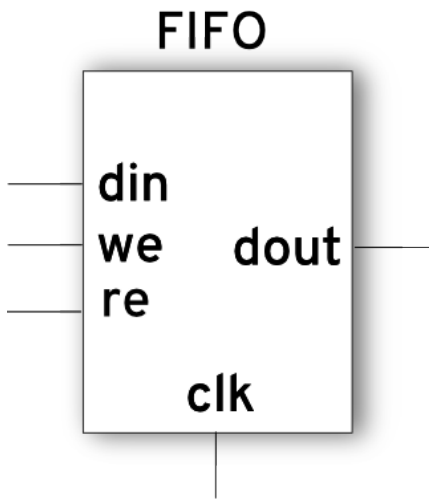
High Level Modelling: Sparse Memory

```
def sparseMemory(dout, din, addr, we, en, clk):

    memory = {}

    @always(clk.posedge)
    def access():
        if en:
            if we:
                memory[addr.val] = din.val
            else:
                dout.next = memory[addr.val]

    return access
```



High Level Modelling: FIFO

```
def fifo(dout, din, re, we, empty, full, clk,
                              maxFilling=sys.maxint):

    memory = []

    @always(clk.posedge)
    def access():
        if we:
            memory.insert(0, din.val)
        if re:
            dout.next = memory.pop()
        filling = len(memory)
        empty.next = (filling == 0)
        full.next = (filling == maxFilling)

    return access
```



Errors using Exceptions: Sparse Memory

```
class Error(Exception):
  pass

def sparseMemory2(dout, din, addr, we, en, clk):
  memory = {}

  @always(clk.posedge)
  def access():
    if en:
      if we:
        memory[addr.val] = din.val
      else:
        try:
          dout.next = memory[addr.val]
        except KeyError:
          raise Error, "Uninitialized address %s" % hex(addr)

    return access
```

Co-simulation with Verilog

```
module bin2gray(B, G);

    parameter width = 8;
    input [width-1:0]  B;
    output [width-1:0] G;
    reg [width-1:0] G;
    integer i;
    wire [width:0] extB;

    assign extB = {1'b0, B}; // zero-extend input

    always @(extB) begin
        for (i=0; i < width; i=i+1)
          G[i] <= extB[i+1] ^ extB[i];
    end

endmodule // bin2gray.v
```

Co-simulation with Verilog (2)

```
module dut_bin2gray;

    reg [`width-1:0] B;
    wire [`width-1:0] G;

    initial begin
        $from_myhdl(B);
        $to_myhdl(G);
    end

    bin2gray dut (.B(B), .G(G));
    defparam dut.width = `width;

endmodule // dut_bin2gray.v
```

Co-simulation with Verilog (3)

```
import os

from myhdl import Cosimulation

cmd="iverilog -o bin2gray -Dwidth=%s bin2gray.v dut_bin2gray.v"

def bin2gray(B, G, width):
    os.system(cmd % width)
    return Cosimulation("vvp -m ./myhdl.vpi bin2gray", B=B, G=G)
```

```
$ python test_gray.py -v
Check that only one bit changes in successive codewords ... ok

----------------------------------------------------------------------
Ran 1 tests in 2.79s

OK
```

Conversion Example

```
width = 8

B = Signal(intbv(0)[width:])
G = Signal(intbv(0)[width:])

bin2gray_inst = toVerilog(bin2gray, B, G, width)
bin2gray_inst = toVHDL(bin2gray, B, G, width)
```

Summary

- Describe the HDL in Python/MyHDL
- Simulate
- Convert to Verilog or VHDL
- Co-simulate to verify conversion routines
- Synthesize to target technology
- Co-simulate with synthesized netlist

Using PyPy

Comparison to VHDL and Verilog (Normalized to PyPy)

MyHDL	MyHDL@pypy	Icarus	GHDL	[Verilog]	[VHDL]
timer	1.00	1.71	2.35	4.19	3.53
lfsr24	1.00	1.20	1.08	4.03	3.64
randgen	1.00	3.18	0.39	1.23	1.08
longdiv	1.00	0.62	3.25	1.39	1.42
findmax	1.00	1.65	26.23	0.24	0.43

PyPy versions (time in seconds)

MyHDL	pypy-1.5.0	pypy-1.6.0
timer	85	62
lfsr24	104	66
randgen	91	62
longdiv	87	69
findmax	112	86

`http://www.myhdl.org/doku.php/performance`

References

MyHDL:
http://www.myhdl.org/doku.php/start

Mailing list:
http://www.myhdl.org/doku.php/mailing_list

Presentation:
http://shakthimaan.com/downloads.html#from-python-to-silicon