

Python in the hardware world

MyHDL

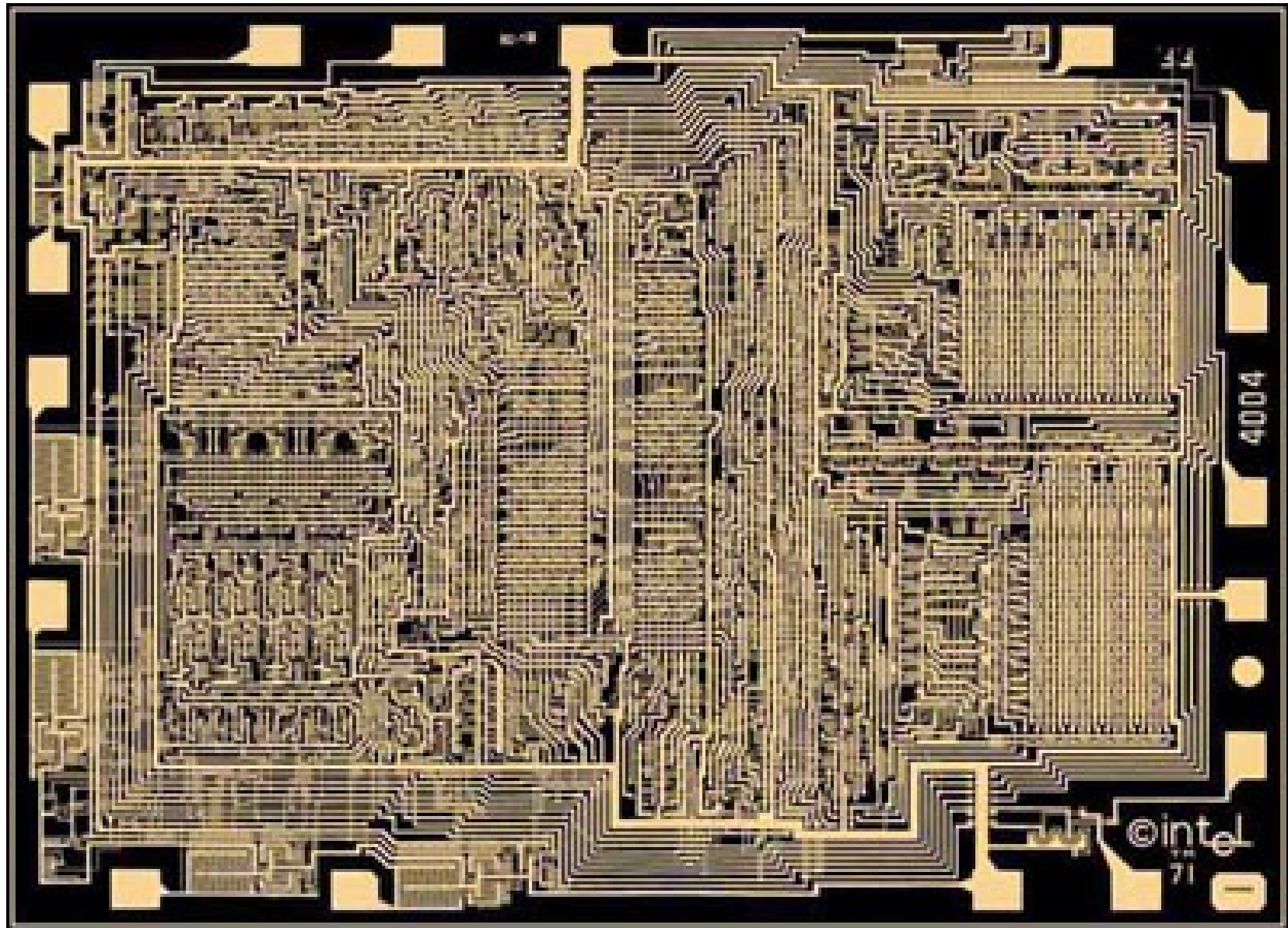
>>> PyCon India '10

Anish Mangal
anish@sugarlabs.org

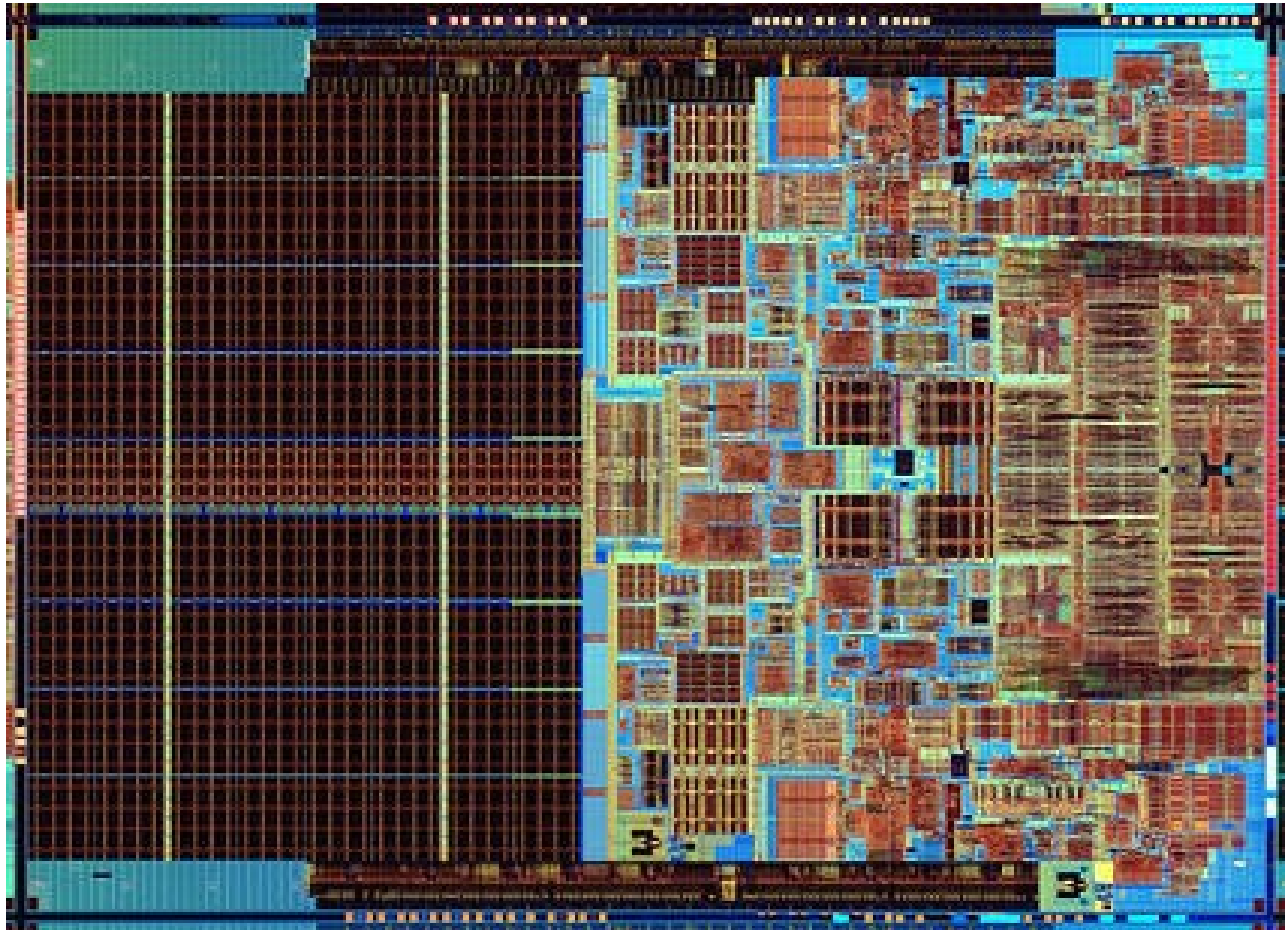
But what is a HDL in the first place?

- Hardware Description Language
- Represents hardware at various abstraction levels

...but why do we need it?

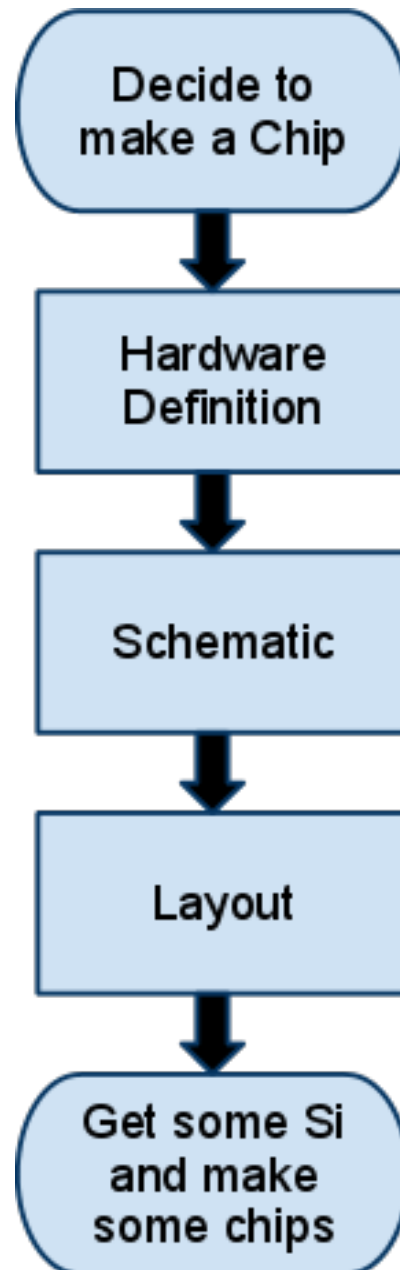


Intel 4004 (1971)
2300 Transistors

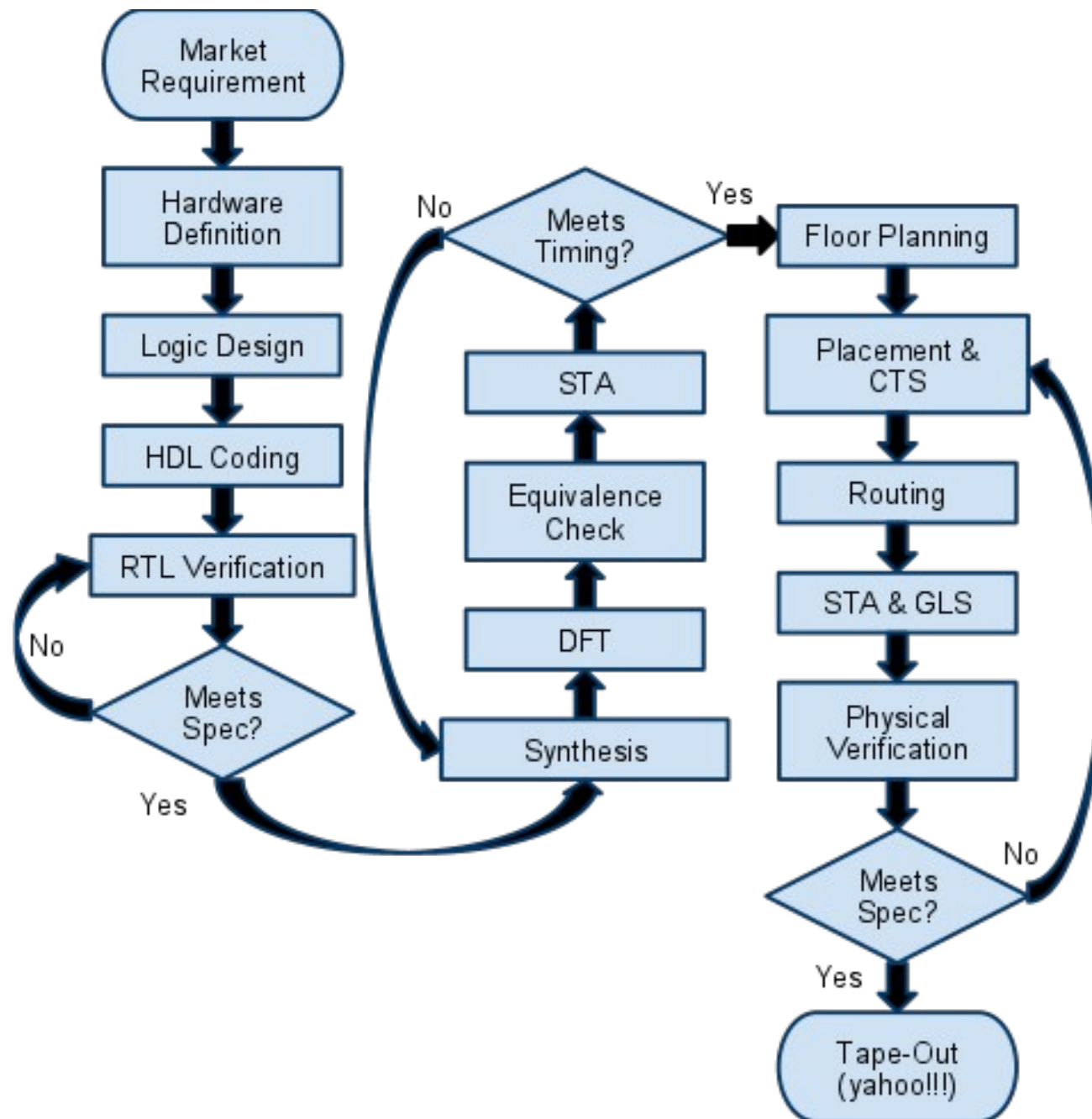


Intel Core 2 Duo (2006)
291,000,000+ Transistors

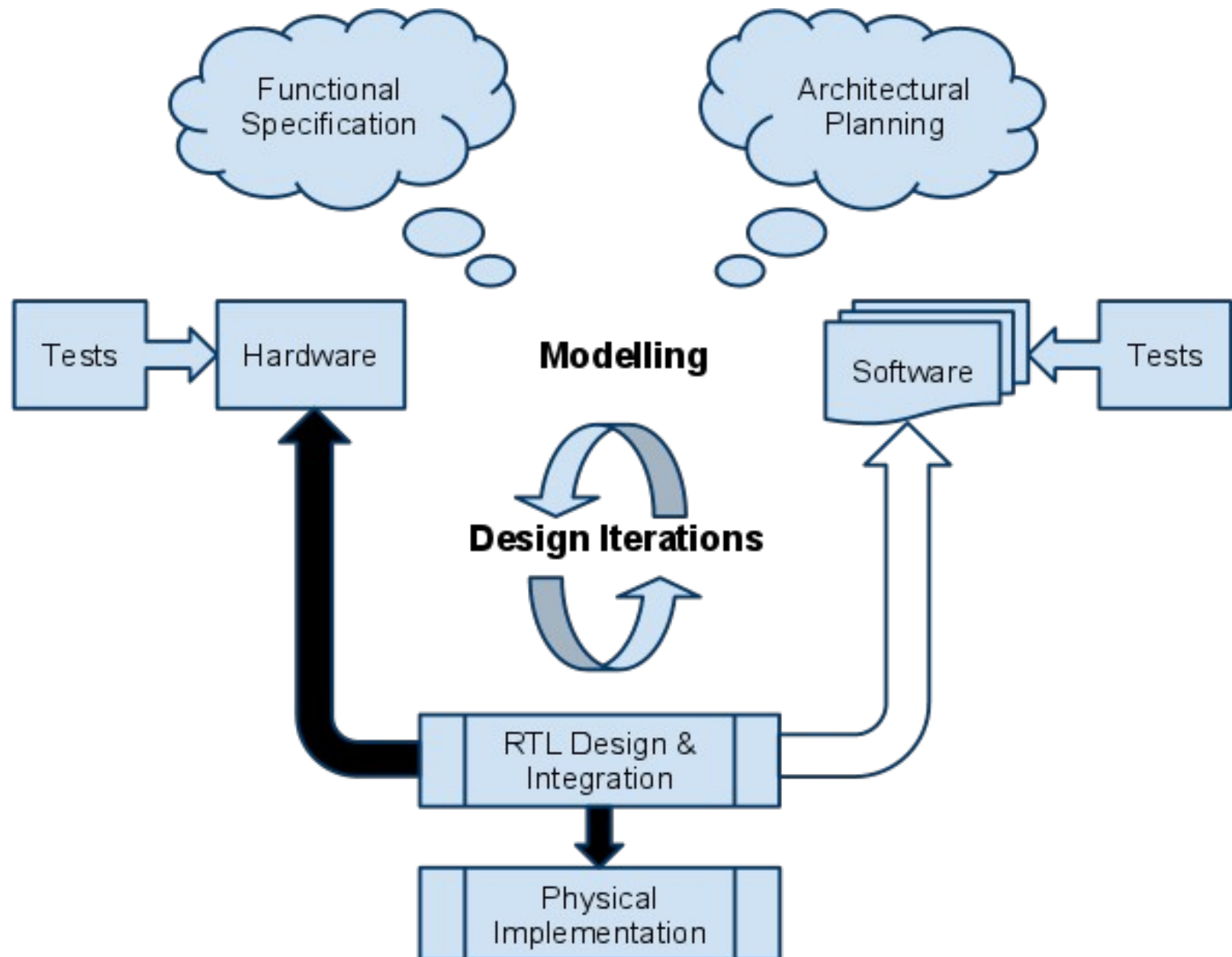
How the times have changed



How the times have changed



How the times have changed



1977 - ISM (Carnegie Mellon University)
- KARL (University of Kaiserslautern)

1985 - Verilog

1987 - VHDL

1995 - Vera

2000 - SystemC

2005 - SystemVerilog (IEEE 1800)

Problems

- Everything in the Si world proprietary
 - Little constructive collaboration
 - Little standard-ness of standards
- Essentially hardware languages
- Not built for High Level Design/Modelling

In short, they start to suck as the design/codebase becomes complex

Factors that drive costs

- **Time**
 - **R & D**
 - **Physical Testing**
 - **Manufacturing**
- **Resources**
 - **Manpower**
 - **Computing**
- **Reuse, Reuse, Reuse**
- **Tools & Methodology**
- **Process automation**
- **Abstraction**
- **Pipelining**
- **Quality**

Mantras for efficient chip designing

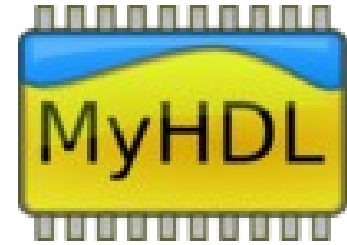
Focus

High level design and modelling



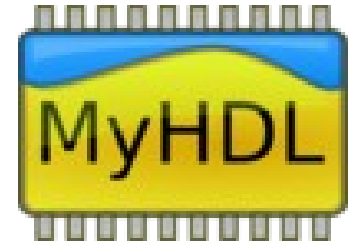
from **Python** to **Silicon**

Modeling



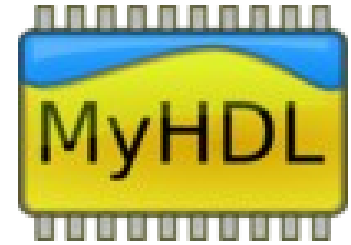
- Python generators \Rightarrow Hardware Concurrency
- Hardware module: Function that returns generators
 - Arbitrary hierarchy
 - Named port associations
 - Arrays of instances
 - Conditional instantiation

Simulation & Verification



- Record signal changes to a Value Change Dump (vcd) file
 - Viewable in popular FOSS wave viewers such as gtkWave
- Unit testing
- Co-Simulation

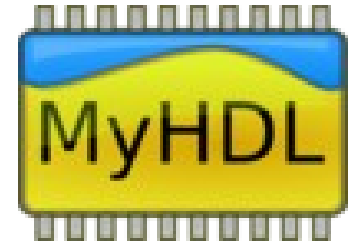
Conversion to Verilog & VHDL



Python → FPGA → Silicon

Introduction

Generators

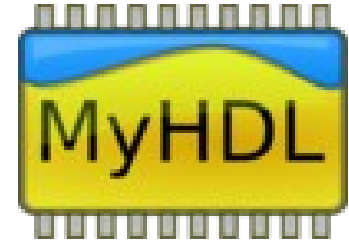


```
def function():  
    for i in range(5):  
        return i
```

```
>>> function()  
0
```


Introduction

Generators

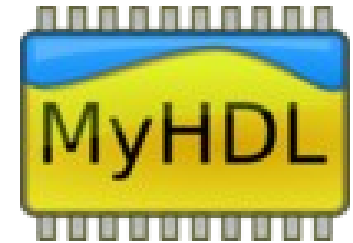


```
def generator():  
    for i in range(5):  
        yield i
```

```
>>> generator()  
<generator object at 0x815d5a8>
```

```
>>> g = generator()  
>>> g.next()  
0  
>>> g.next()  
1  
>>> g.next()  
2  
>>> g.next()  
3  
>>> g.next()  
4  
>>> g.next()  
Traceback (most recent call last): File "<stdin>", line 1, in ?  
StopIteration
```

Introduction



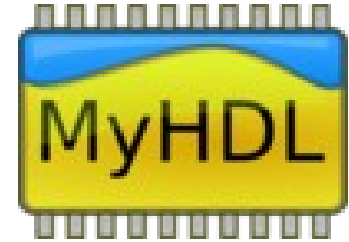
Generators

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

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

```
func = deco(func)
```

A basic simulation



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

def HelloWorld():
    interval = delay(10)

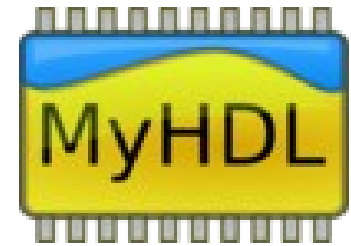
    @always(interval)
    def sayHello():
        print "%s Hello World!" % now()

    return sayHello

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

```
% python hello1.py
10 Hello World!
20 Hello World!
30 Hello World! _
SuspendSimulation: Simulated 30 timesteps
```

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

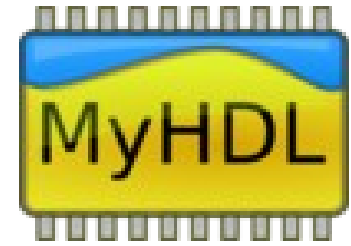
def HelloWorld(clk):

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

    return sayHello

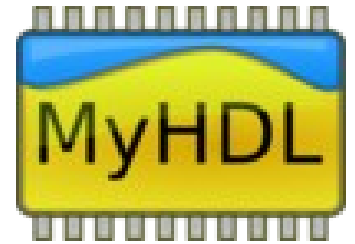
clk = Signal(0)
clkdriver_inst = ClkDriver(clk)
hello_inst = HelloWorld(clk)
sim = Simulation(clkdriver_inst, hello_inst)
sim.run(50)
```

Concurrency



```
% python hello2.py  
10 Hello World!  
30 Hello World!  
50 Hello World!  
_SuspendSimulation: Simulated 50 timesteps
```

Bit indexing



```
from myhdl import Signal, delay, Simulation, always_comb, instance, intbv, bin

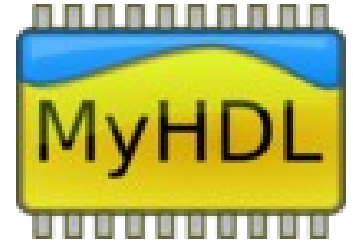
def bin2gray(B, G, width):

    """ Gray encoder. B
    -- input intbv signal, binary encoded G
    -- output intbv signal, gray encoded width
    -- bit width
    """

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

    return logic
```

Bit indexing



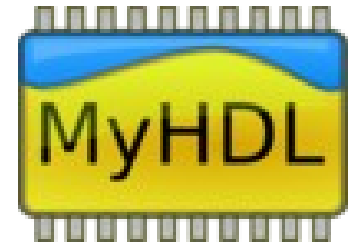
```
def testBench(width):
    B = Signal(intbv(0))
    G = Signal(intbv(0))
    dut = bin2gray(B, G, width)

    @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
```

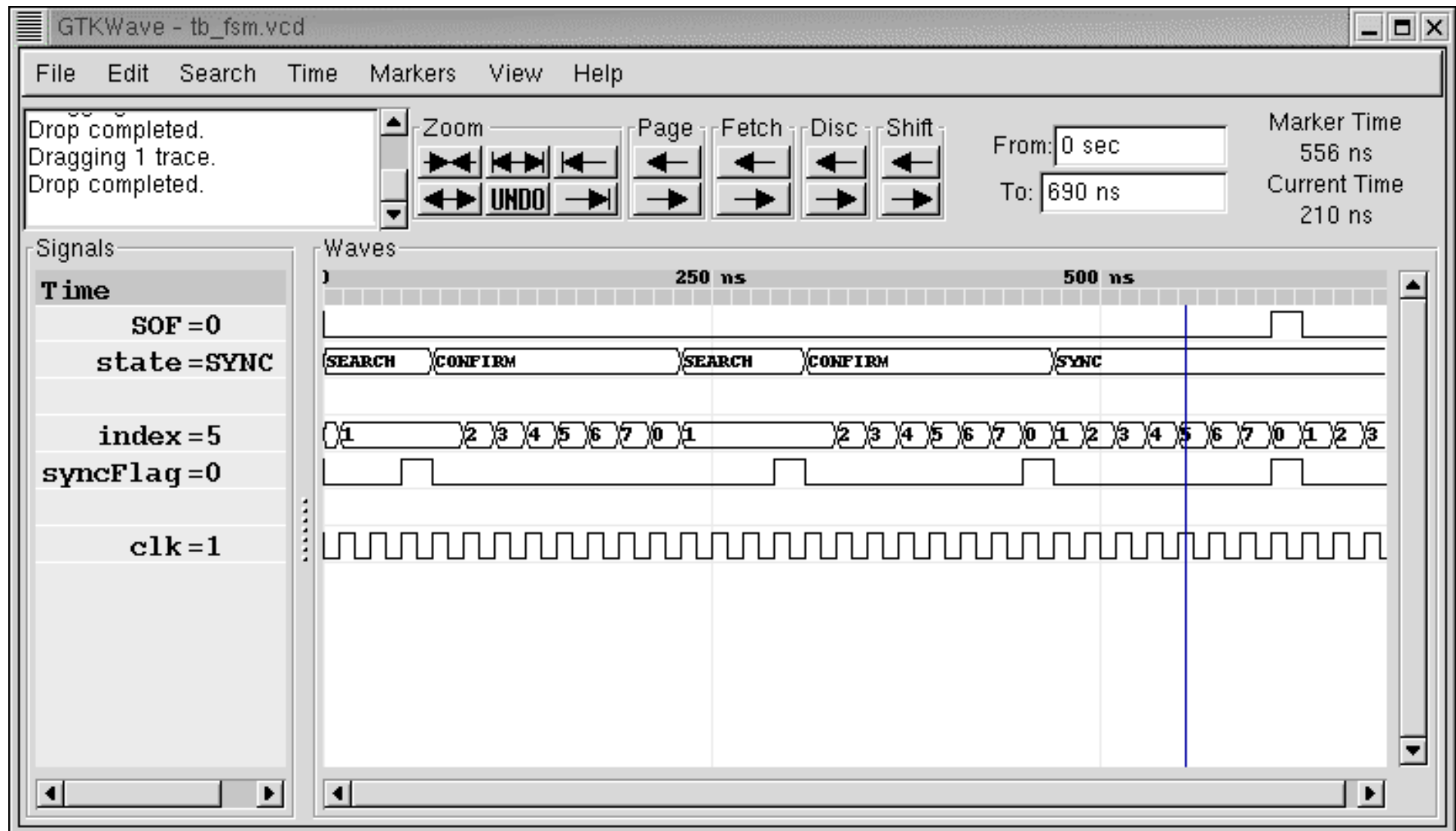
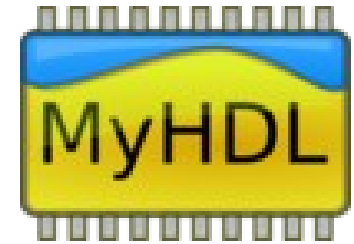
```
sim = Simulation(testBench(width=3))
sim.run()
```

Bit indexing

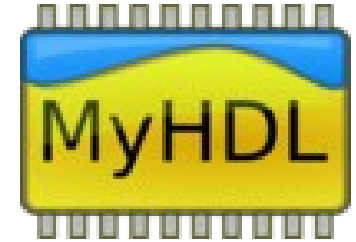


```
% python bin2gray.py
B: 000 | G: 000
B: 001 | G: 001
B: 010 | G: 011
B: 011 | G: 010
B: 100 | G: 110
B: 101 | G: 111
B: 110 | G: 101
B: 111 | G: 100
StopSimulation: No more events
```


Recording simulation

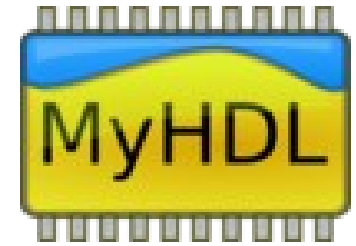


Modeling memories



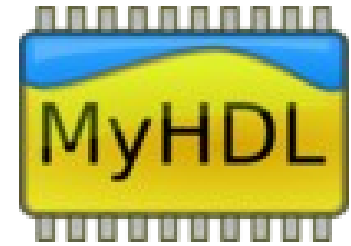
```
def sparseMemory(dout, din, addr, we, en, clk):  
    """ Sparse memory model based on a dictionary.  
  
    Ports:  
    dout -- data out  
    din -- data in  
    addr -- address bus  
    we -- write enable: write if 1, read otherwise  
    en -- interface enable: enabled if 1  
    clk -- clock input  
    """  
  
    memory = {}  
  
    @always(clk.posedge)  
    def access():  
        if en:  
            if we:  
                memory[addr.val] = din.val  
            else:  
                dout.next = memory[addr.val]  
  
    return access
```

Super-cool features



- CoSimulation
 - Only requires a PLI
- Conversion to VHDL and Verilog
 - Some limitations

Try MyHDL today



<http://sourceforge.net/projects/myhdl/files/>

The real power of MyHDL lies in the fact that it can leverage python power and apply it as a pluggable module into an entirely Verilog/VHDL environment

Thank you!