FPGA Design with Python and MyHDL

Guy Eschemann

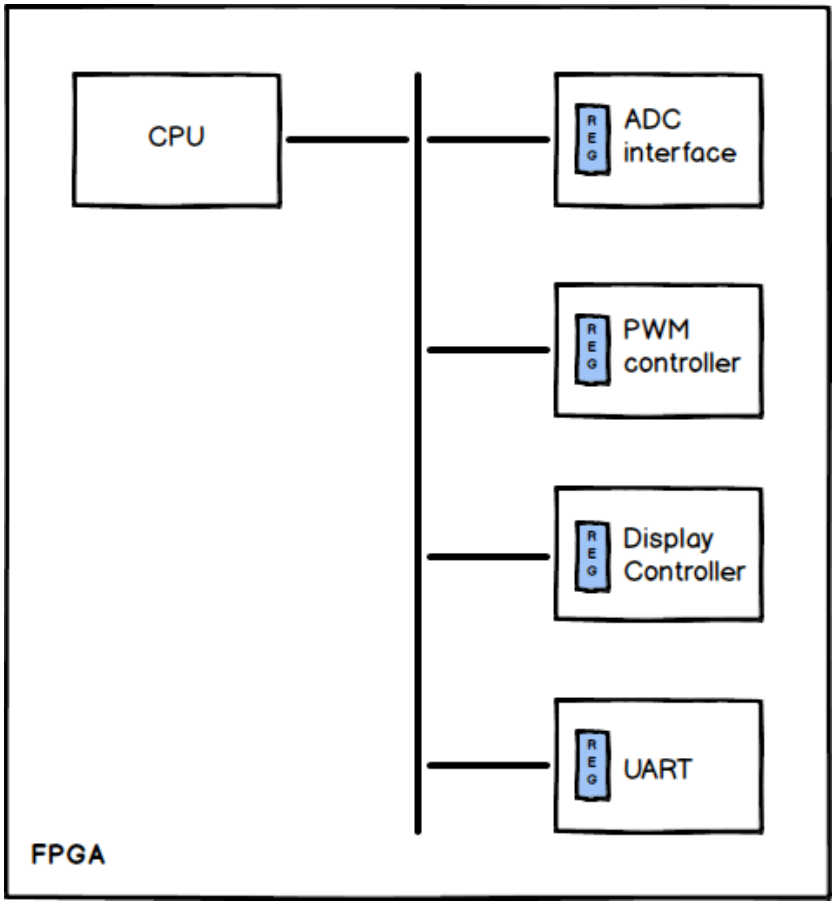
Content

- About us
- A short introduction to Python
- Introduction to MyHDL
- Demo
- Questions

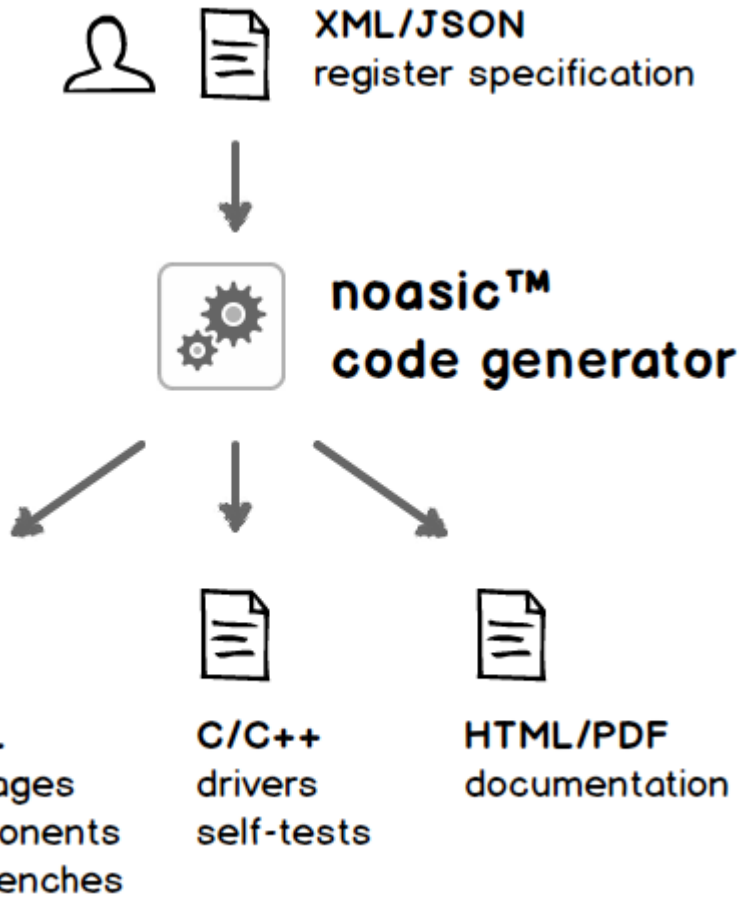
About noasic

- Quick Facts
  - Founded in 2014
  - 14+ years experience in FPGA design
  - Located in Kehl, Germany
- Areas of expertise
  - FPGA & IP core design and verification
  - VHDL code generators

VHDL Code Generators (1)



VHDL Code Generators (2)



FPGA-News.de

Über FPGA-News.de

FPGA-News.de ist eine herstellerunabhängige Informationsseite über Field Programmable Gate Arrays (FPGAs). Hier finden Sie aktuelle Nachrichten und wichtige Termine rund um das Thema FPGA.

Anzeige

Abonnieren

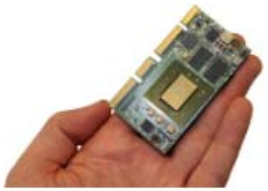
Um automatisch über neue Beiträge informiert zu werden, können Sie FPGA-News.de per E-Mail oder RSS-Feed abonnieren.

Kontakt

noasic e.K.
Sundheimer Feld 6
77694 Kehl
Deutschland

Neues aus der FPGA-Welt

by Guy Eschemann on 3. März 2014



PLDA hat ein neues System-on-Module (SoM) Produkt angekündigt. Das SoMZ-7045 basiert auf einem Xilinx Zynq-7000 XC7Z045 AP Baustein, und verfügt über 512 MB DDR3-RAM für die CPU sowie 256 MB DDR3-RAM für das FPGA. Das Modul hat eine Größe von 76mm x 42mm, und ist ab sofort verfügbar (mehr).

Sigasi hat eine seinen VHDL-Editor aktualisiert. Highlights der Version 2.20 sind der neue Sensitivitätslisten-Check, sowie zwei neue Sigasi Premium Features: State-Machine Diagramme und Net Search (mehr).

Xilinx hat seine Artix-7 FPGA-Familie um neue Automotive-Bausteine erweitert. Die XA Artix-7 Familie besteht aus vier Bausteinen – XA7A35T, XA7A50T, XA7A75T und XA7A100T – mit 30k bis 101k Logikzellen. XA7A100T Bausteine sind ab sofort verfügbar. Die anderen Varianten sollen im Q4/2014 erhältlich sein (mehr).

Nächste FPGA-Termine:

- 03.03.14 – 07.03.14
  Expert VHDL, Doulos, München
- 03.03.14 – 07.03.14
  Comprehensive VHDL Introduction, SynthWorks, Online Class
- 03.03.14 – 07.03.14
  VHDL Testbenches and Verification, SynthWorks, Online Class

FPGA-Termine

06.03.14 - 10.03.14
Advanced Features and Techniques of Embedded Systems Software Design - 5 Tage , PLC2, Stuttgart

10.03.14 - 14.03.14
Professional VIVADO , PLC2, Freiburg

10.03.14 - 14.03.14
Professional FPGA, PLC2, Freiburg

10.03.14 - 14.03.14
Professional DSP, PLC2, Freiburg

10.03.14 - 12.03.14
Video Signal Processing, PLC2, Stuttgart

11.03.14 - 13.03.14
SystemVerilog - Advanced Verification for FPGA Design, Trias, Stuttgart

11.03.14 - 13.03.14
Signal Integrity for Hardware Designers, Trias, Berlin

11.03.14 - 12.03.14

A short introduction to

Python

- Created 1991 by **Guido van Rossum** at CWI (Netherlands)
- General-purpose, high-level programming language
- Emphasizes code readability
- Write less code
- Strongly, dynamically typed
- Object-oriented
- Open-source
- Runs on many platforms

Sample Python code

```
def add(a, b):
    """ Returns the sum of 'a' and 'b' """
    c = a + b
    return c

# The main() function
if __name__ == "__main__":
    a = 1
    b = 2
    c = add(a, b)
    print "The sum of %d and %d is %d" % (a, b, c)
```

Python – Syntax

- Blocks are specified by indentation
- No mandatory statement termination
- Comments start with #
- Assign values using =
- Test equality using ==
- Increment/decrement using += or -=

Python – Data Structures

- Lists
  - like one-dimensional arrays
- Tuples
  - immutable, one-dimensional arrays
- Dictionaries
  - associative arrays („hash tables“)

Python – Strings

- Can use either single or double quotation marks
- Multi-line strings enclosed in triple quotes
- Use modulo (%) operator and a tuple to format a string

Python – Flow Control

- Flow control statements:
  - if
  - for
  - while
  - there is no case/switch. Use „if“ instead.

Python – Functions

- Declared with the „def“ keyword
- Optional parameters supported
- Functions can return multiple values

Python – Importing

- External libaries (such as MyHDL) must be imported before they can be used:

  ```
  from myhdl import intbv
  ```

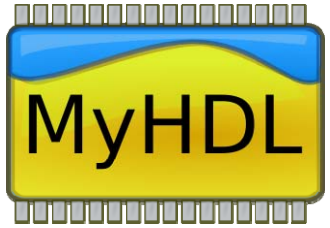
  OR

  ```
  from myhdl import *
  ```

Python Libraries

- String handling
- Regular expressions
- Random number generation
- Unit-test framework
- OS interfacing
- GUI development
- Modules for math, database connections, network interfacing etc.
- **Hardware design (MyHDL)**

Introduction to



MyHDL

- Python-based hardware description language
- Created 2003 by Jan Decaluwe
- Open source

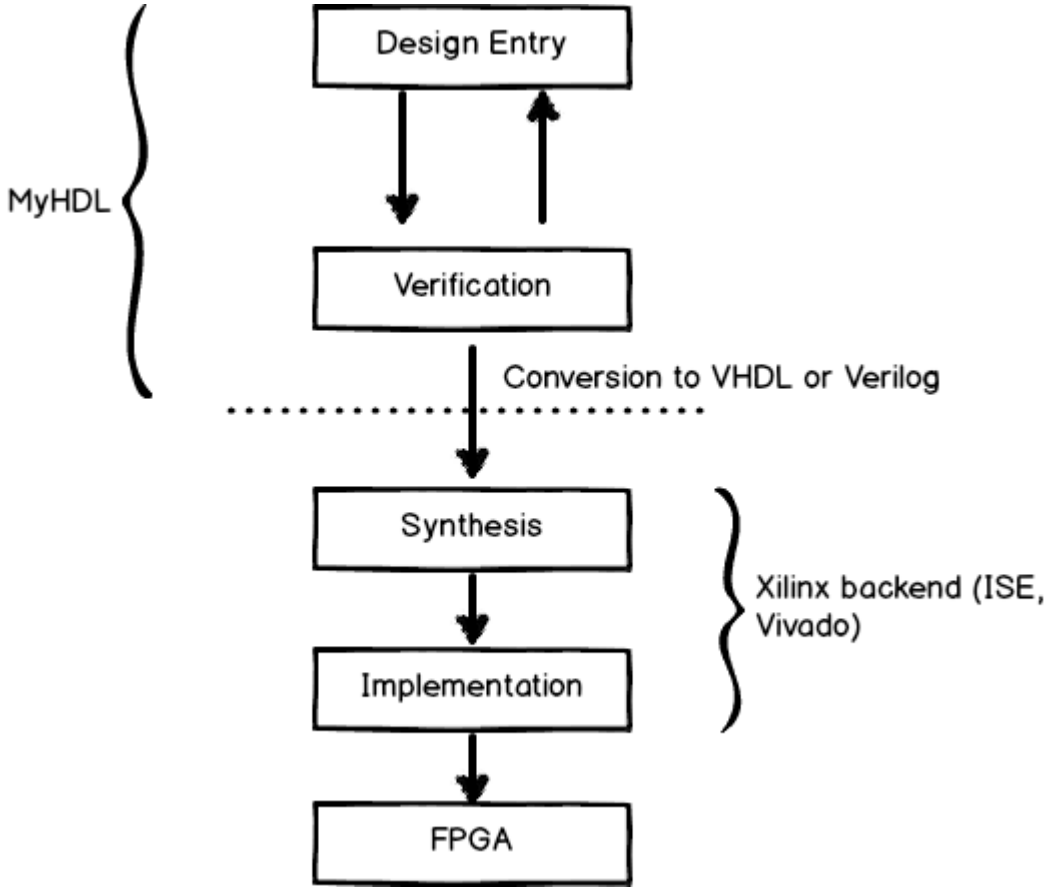
Why MyHDL?

- Write less code
- Avoid common VHDL and Verilog pitfalls
- Apply modern SW development techniques to HW design
- Can generate both VHDL and Verilog
- Simulator included
- Open source
- Runs on many platforms (Windows, OSX, Linux)
- Python ecosystem

What MyHDL is not

- Not a way to turn arbitrary Python into silicon
- Not a radically new approach
- Not a synthesis tool
- Not an IP block library
- Not only for implementation
- Not well suited for accurate timing simulations

MyHDL Design Flow



Example – 8 bit counter

```python
from myhdl import *

def counter(i_clk, i_reset, o_count):

    """ A free-running 8-bit counter with a synchronous reset

    s_count = Signal(intbv(0)[8:])

    @always(i_clk.posedge)
    def count():
        if i_reset == 1:
            s_count.next = 0
        else:
            s_count.next = s_count + 1

    @always_comb
    def outputs():
        o_count.next = s_count

    return count, outputs
```

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use std.textio.all;

use work.pck_myhdl_09.all;

entity counter is
    port(
        i_clk   : in  std_logic;
        i_reset : in  std_logic;
        o_count : out unsigned(7 downto 0)
    );
end entity counter;
-- A free-running 8-bit counter with a synchronous reset

architecture MyHDL of counter is
    signal s_count : unsigned(7 downto 0);

begin
    COUNTER_COUNT : process(i_clk) is
    begin
        if rising_edge(i_clk) then
            if (i_reset = '1') then
                s_count <= to_unsigned(0, 8);
            else
                s_count <= (s_count + 1);
            end if;
        end if;
    end process COUNTER_COUNT;

    o_count <= s_count;

end architecture MyHDL;
```

Modeling Components

- Components are modeled using functions
- Function parameters map to ports

```
1  from myhdl import *
2
3  def counter(i_clk, i_reset, o_count):
4
5      """ A free-running 8-bit counter with a synchronous reset """
6
7      s_count = Signal(intbv(0)[8:])
8
9      @always(i_clk.posedge)
10     def count():
11         if i_reset == 1:
12             s_count.next = 0
13         else:
14             s_count.next = s_count + 1
15
16     @always_comb
17     def outputs():
18         o_count.next = s_count
19
20     return count, outputs
```

Modeling Processes

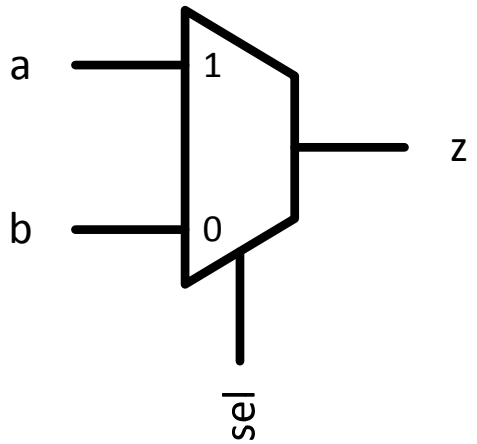
- Processes are modeled using special „generator“ functions
- Decorators are used to create generator functions

```
1  from myhdl import *
2
3  def counter(i_clk, i_reset, o_count):
4
5      """ A free-running 8-bit counter with a synchronous reset """
6
7      s_count = Signal(intbv(0)[8:])
8
9      @always(i_clk.posedge)
10     def count():
11         if i_reset == 1:
12             s_count.next = 0
13         else:
14             s_count.next = s_count + 1
15
16     @always_comb
17     def outputs():
18         o_count.next = s_count
19
20     return count, outputs
```

The @instance decorator

- Most general, creates generator from a generator function

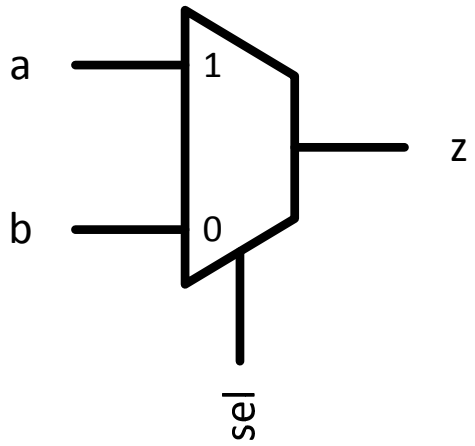
```
@instance # equivalent to mux = mux()
def mux():
    while True:
        yield a, b, sel
        if sel:
            z.next = a
        else:
            z.next = b
```



The @always decorator

- Abstracts an outer „while True“ loop followed by a „yield“ statement

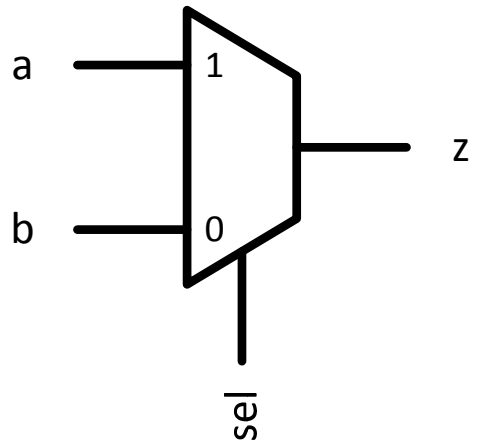
```
@always(a, b, sel)
def mux():
    if sel:
        z.next = a
    else:
        z.next = b
```



Combinational logic: @always_comb

- Automatically infers the sensitivity list

```
@always_comb
def mux():
    if sel:
        z.next = a
    else:
        z.next = b
```



Sequential logic: @always_seq

- Infers the reset functionality

```
reset = ResetSignal(0, active=0, async=True)

@always_seq(clock.posedge, reset=reset)
def dff():
    q.next = d
```

- Equivalent code:

```
@always(clock.posedge, reset=reset)
def dff():
    if reset == 0:
        q.next = 0
    else:
        q.next = d
```

Function Decorators

- **@instance**: most general, multiple yields
- **@always**: single yield, abstracts „while True“ loop
- **@always_comb**: for asynchronous logic, automatically infers sensitivity list
- **@always_seq**: for sequential logic, automatically infers the reset functionality

MyHDL Signals

- Similar to VHDL signals
- Signal declaration:

```
s_empty = Signal(0)
s_count = Signal(intbv(0)[8:])
s_clk = Signal(bool(0))
```

- Signal assignment:

```
s_count.next = 0
```

MyHDL intbv class

- Similiar to standard Python „int“ type with added indexing and slicing capabilities
- intbv = „integer with bit-vector flavor“
- Provides access to underlying two‘s complement representation
- Range of allowed values can be constrained

MyHDL intbv creation

Create an intbv:

```
intbv([val=None] [, min=None]  [, max=None])
```

Example:

```
>>> a = intbv(24, min=0, max=25)
>>> a.min
0
>>> a.max
25
>>> len(a)
5
```

MyHDL intbv indexing and slicing

Indexing:

```
>>> a = intbv(0x12)
>>> bin(a)
'10010'
>>> a[0]
False
>>> a[1]
True
```

Slicing:

```
>>> a = intbv(0x12)
>>> a[4:0]
intbv(2L)
```

MyHDL intbv creation

Create an intbv, specifying its width:

```
>>> a = intbv(24)[5:]
>>> a.min
0
>>> a.max
32
>>> len(a)
5
```

Modeling hierarchy

```
1  from myhdl import *
2  from counter import *
3
4  def toplevel(i_clk, i_reset, o_strobe):
5
6      s_count = Signal(intbv(0)[8:])
7
8      # instantiate a counter component
9      inst_counter = counter(i_clk, i_reset, s_count)
10
11     @always_comb
12     def strobe():
13         """ Generate a strobe pulse every time the counter value equals 100 """
14         if s_count == 100:
15             o_strobe.next = 1
16         else:
17             o_strobe.next = 0
18
19     return inst_counter, strobe
```

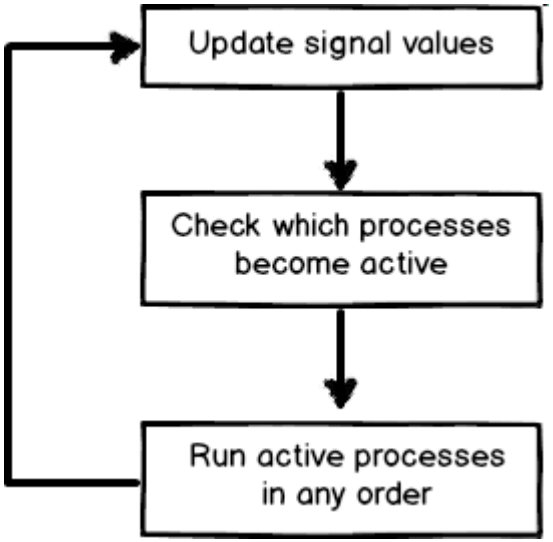
Generated VHDL code

```vhdl
-- File: toplevel.vhd
-- Generated by MyHDL 0.9dev
-- Date: Fri Apr 18 11:21:37 2014


library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use std.textio.all;

use work.pck_myhdl_09.all;

entity toplevel is
    port(
        i_clk    : in  std_logic;
        i_reset  : in  std_logic;
        o_strobe : out std_logic
    );
end entity toplevel;

architecture MyHDL of toplevel is
    signal s_count              : unsigned(7 downto 0);
    signal inst_counter_s_count : unsigned(7 downto 0);

begin
    TOPLEVEL_INST_COUNTER_COUNT : process(i_clk) is
    begin
        if rising_edge(i_clk) then
            if (i_reset = '1') then
                inst_counter_s_count <= to_unsigned(0, 8);
            else
                inst_counter_s_count <= (inst_counter_s_count + 1);
            end if;
        end if;
    end process TOPLEVEL_INST_COUNTER_COUNT;

    s_count <= inst_counter_s_count;

    -- Generate a strobe pulse every time the counter value equals 100
    TOPLEVEL_STROBE : process(s_count) is
    begin
        if (s_count = 100) then
            o_strobe <= '1';
        else
            o_strobe <= '0';
        end if;
    end process TOPLEVEL_STROBE;

end architecture MyHDL;
```

Verification

- Verification is MyHDL´s strongest point
- Arguably the hardest part of hardware design
- There are no restrictions for verification: can use the full power of Python
- Can do „agile“ hardware design
- The Foundation is an event-driven simulator in the MyHDL library

MyHDL Simulator

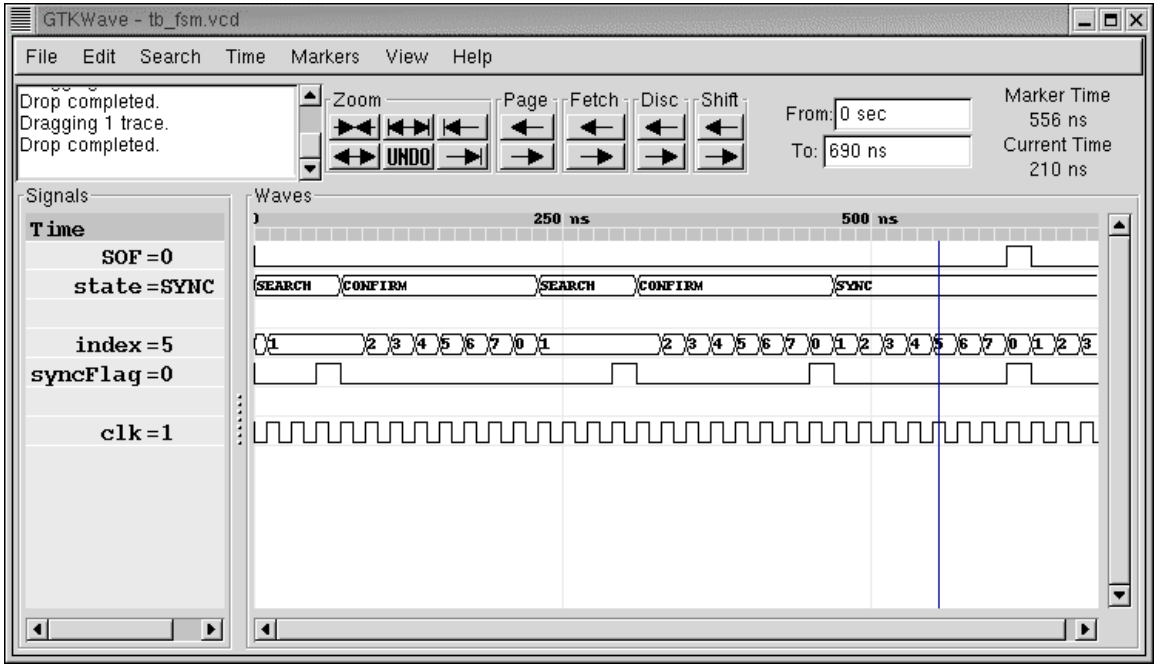


A basic MyHDL simulation

```
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)
```

Debugging in MyHDL

- Regular Python debugging
- Waveform viewer output



Conversion

- A subset of MyHDL can be automatically converted to VHDL and Verilog
- The converter maintains the abstraction level
- Supports some unique MyHDL features
- Creates readable VHDL/Verilog
- Convertible subset much broader than synthesizable subset

Conversion example

- MyHDL Code

```
a = Signal(intbv(0, min=0, max=8))
b = Signal(intbv(0, min=-8, max=8))
c = Signal(intbv(0, min=-8, max=15))
c.next = a + b
```

- Conversion to Verilog

```
assign c = ($signed({1'b0, a}) + b);
```

- Conversion to VHDL

```
c <= (signed(resize(a, 5)) + b);
```

- The converter does the casts and resizings automatically

User-defined code

```python
from myhdl import *

def ibufds(I, IB, O):
    """ Xilinx Differential Signaling Input Buffer"""
    @always(I, IB)
    def output():
        O.next = I
    return instances()

ibufds.vhdl_code = """
IBUFDS_inst : IBUFDS
generic map (
    DIFF_TERM => FALSE,
    IBUF_LOW_PWR => TRUE,
    IOSTANDARD => "DEFAULT")
port map (
    O => O,
    I => I,
    IB => IB
);"""

if __name__ == '__main__':
    I = Signal(bool(0))
    IB = Signal(bool(0))
    O = Signal(bool(0))
    toVHDL.use_clauses = "library unisim;\nuse unisim.vcomponents.all;"
    inst_ibufds = toVHDL(ibufds,I, IB, O)
```

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use std.textio.all;

library unisim;
use unisim.vcomponents.all;

entity ibufds is
    port(
        I  : in std_logic;
        IB : in std_logic;
        O  : in std_logic
    );
end entity ibufds;
-- Xilinx Differential Signaling Input Buffer

architecture MyHDL of ibufds is
begin
    IBUFDS_inst : IBUFDS
        generic map(
            DIFF_TERM    => FALSE,
            IBUF_LOW_PWR => TRUE,
            IOSTANDARD   => "DEFAULT")
        port map(
            O  => O,
            I  => I,
            IB => IB
        );
end architecture MyHDL;
```

Demo

Caveats

- Dynamic nature of Python needs some getting used to
- Error messages can be cryptic
- Cannot import legacy VHDL/Verilog code
  - Write own behaviorals models for simulation
  - Use user-defined code for conversion
- Cannot generate parameterizable HDL code

MyHDL future

- Fixed-point support in the converter
- Interfaces
- Python 3.0 support

Resources

- http://python.org
- http://myhdl.org
  - Manual & examples
  - Download & installation instructions
  - Mailing list
  - Tutorials & publications
- @MyHDL on twitter
- Development with mercurial on Bitbucket

Contact

Guy Eschemann
noasic GmbH
Sundheimer Feld 6
77694 Kehl

guy@noasic.com
http://noasic.com
Twitter: @geschema