

# Lighting fast introduction to myhdl

[http://github.com/cfetlon/notebook\\_slides](http://github.com/cfetlon/notebook_slides) ([http://github.com/cfetlon/notebook\\_slides](http://github.com/cfetlon/notebook_slides)).

# myhdl.org (http://www.myhdl.org)

MyHDL

Start ▾

Docs ▾

Support ▾

Users

Search

Info

## Design hardware with Python

MyHDL turns Python into a hardware description and verification language, providing hardware engineers with the power of the Python ecosystem.



### Integrates seamlessly

MyHDL designs can be converted to Verilog or VHDL automatically, and implemented using a standard tool flow.



### Silicon proven

Many MyHDL designs have been implemented in ASICs and FPGAs, including some high volume applications.



### Open source

MyHDL is an open source, pure Python package. [Install with pip](#) and enjoy the Python ecosystem immediately. [Visit the project on Github.](#)

#### Release news

09-May-2018 [MyHDL 0.10.0 released](#)

#### General news

#### Tweets



MyHDL Retweeted



**Steven Armour**  
[@ArmourMSG](#)

Finally got the YouTube and Git sorted out and here is a tutorial on Shift Registers in [@myHDL](#) implemented

# myhdl documentation (<http://docs.myhdl.org/en/stable/>)

The MyHDL manual

What's new in MyHDL 0.10

Python 3 Support

What's new in MyHDL 0.9

What's new in MyHDL 0.8

What's new in MyHDL 0.7

What's new in MyHDL 0.6

What's new in MyHDL 0.5

What's new in MyHDL 0.4: Conversion  
to Verilog

What's New in MyHDL 0.3

## Welcome to the MyHDL documentation

- [The MyHDL manual](#)
  - [Overview](#)
  - [Background information](#)
  - [Introduction to MyHDL](#)
  - [Hardware-oriented types](#)
  - [Structural modeling](#)
  - [RTL modeling](#)
  - [High level modeling](#)
  - [Unit testing](#)
  - [Co-simulation with Verilog](#)
  - [Conversion to Verilog and VHDL](#)
  - [Conversion examples](#)
  - [Reference](#)
- [What's new in MyHDL 0.10](#)
  - [The \*block\* decorator](#)
  - [Backwards compatibility issues](#)
- [Python 3 Support](#)

# Core developers



jandecaluwe



jck



josyb



cfelton



hgomersall

# myhdl forum (<http://discourse.myhdl.org/>)

MyHDL Discourse

























all categories ▸

Latest

Top

Categories

+ New Topic

Topic	Category	Users	Replies	Views	Activity
Using standart Python modules	Support	 	2	16	4d
ORConf 2018: need someone to represent myhdl	Meta	  	5	110	7d
— last visit —					
Verilog width expansion and reduction operator equivalence?	Support	  	2	21	8d
Invoke Verilog generate for Python list handling	Support	 	1	402	13d
How to use counters similiar to verilog using for loops?	Support	  	10	74	17d
Variables in VHDL conversion	Bug	   	15	91	Sep 5
I don't get how to convert to VHDL	Support	 	3	58	Sep 3
VHDL constant value overflow	Support	  	9	81	Aug 31

# pyfda and gsoc (gee-sok)

- Python filter, design, and analysis app
- Google summer of code **NOT** system on a chip.

Integrate myhdl and pyfda

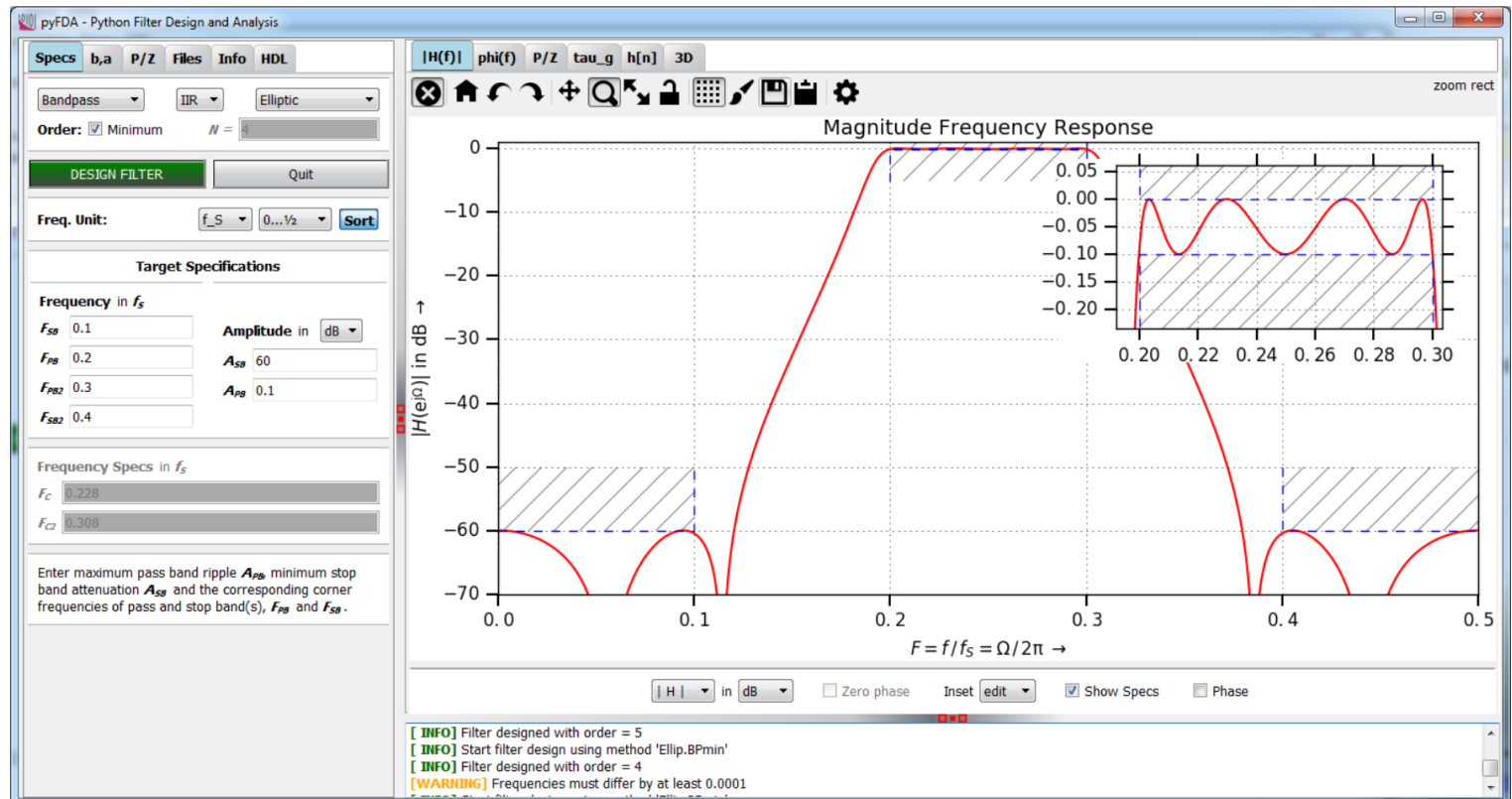
- Define and analyze signal processing elements (mainly filters)
- Digital filters implemented in myhdl and defined / designed in pyfda
- Conversion to verilog and vhdl

# pyfda (<https://github.com/chipmuenk/pyFDA>).

## Python Filter Design Analysis Tool

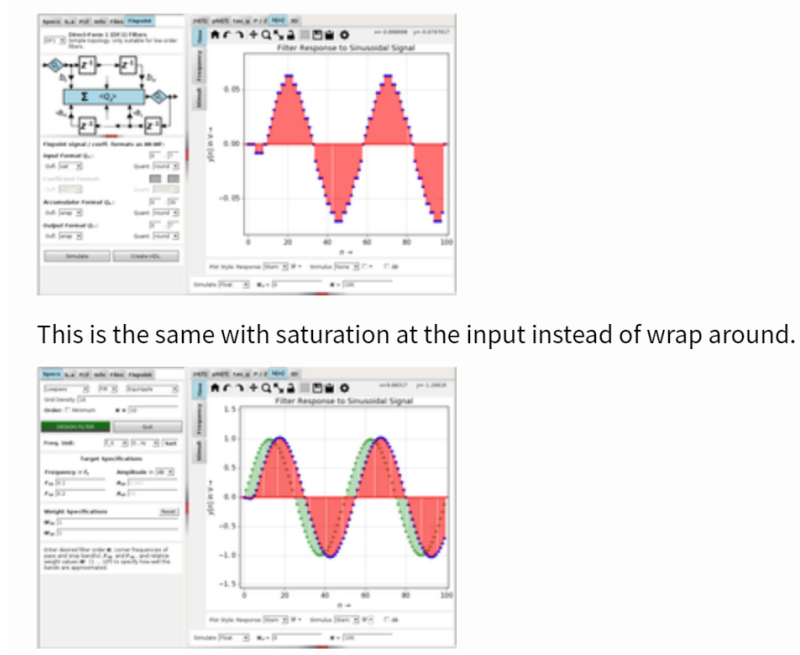
pypi package 0.1.5   anaconda cloud 0.1.5   gitter join chat   license MIT   google group pyFDA   build passing   docs passing

pyFDA is a GUI based tool in Python / Qt for analysing and designing discrete time filters. The capability for generating Verilog and VHDL code for the designed and quantized filters will be added in the next release.



# pyfda and myhdl integration

An example (debug during development) using different settings in the hardware filter block.





# Tools

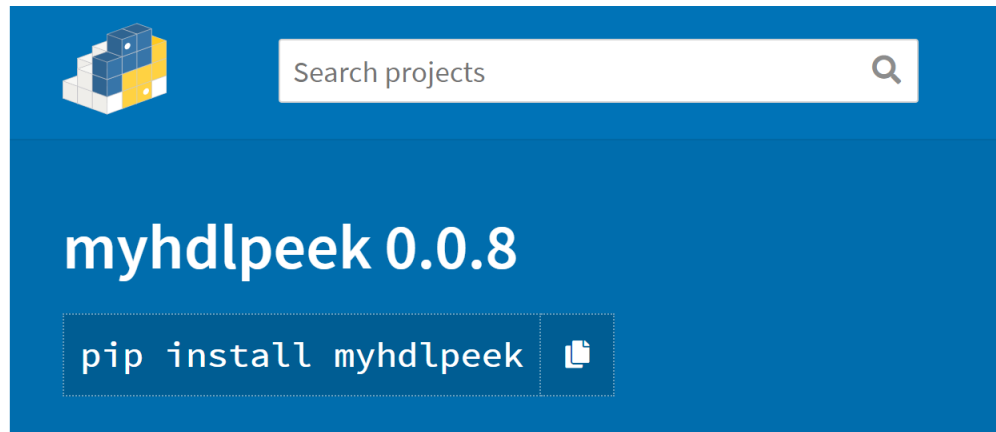


**xesscorp**  
xesscorp

## Maintainers



xesscorp



### myhdlpeek

Monitor and display signal waveforms from your MyHDL digital design in a Jupyter notebook.

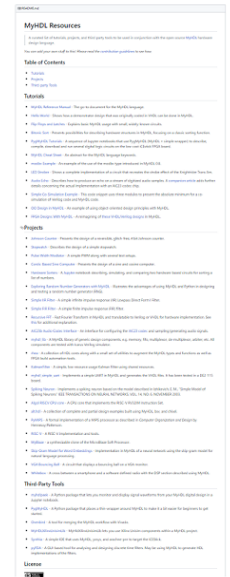
● Python ★ 16 🍴 4

### pygmyhdl

MyHDL hardware design language encased in the tasty PygMyHDL wrapper.

● Python ★ 12 🍴 2

# Resources and projects (<https://github.com/xesscorp/myhdl-resources>)



- [KalmanFilter](#) - A simple, low-resource usage Kalman Filter using shared resources.
- [myhdl\\_simple\\_uart](#) - Implements a simple UART in MyHDL and generates the VHDL files. It has been tested in a DE2-115 board.
- [Spiking Neuron](#) - Implements a spiking neuron based on the model described in Izhikevich, E. M., "Simple Model of Spiking Neurons" IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 14, NO. 6, NOVEMBER 2003.
- [Algol RISC-V CPU core](#) - A CPU core that implements the RISC-V RV32IM Instruction Set.
- [alt.hdl](#) - A collection of complete and partial design examples built using MyHDL, bsv, and chisel.
- [PyMIPS](#) - A formal implementation of a MIPS processor as described in *Computer Organization and Design* by Hennessy/Patterson.
- [RISC-V](#) - A RISC-V implementation and tools.
- [MyBlaze](#) - a synthesizable clone of the MicroBlaze Soft Processor.
- [Skip-Gram Model for Word Embeddings](#) - Implementation in MyHDL of a neural network using the skip-gram model for natural language processing.
- [VGA Bouncing Ball](#) - A circuit that displays a bouncing ball on a VGA monitor.
- [Whitebox](#) - A cross between a smartphone and a software defined radio with the DSP section described using MyHDL.

# myhdl code

Let's get on with it and see some code

```
In [2]: import myhdl as hdl
        from myhdl import Signal, intbv, always_comb

        @hdl.block
        def myadder(a, b, c):
            @always_comb
            def beh():
                c.next = a + b
            return beh
```

```
In [3]: a = Signal(intbv(0, min=-8, max=8))
        b = Signal(intbv(0, min=-8, max=8))
        c = Signal(intbv(0, min=-16, max=16))

        inst = myadder(a, b, c)
        inst.convert('verilog')
        inst.convert('vhdl')

        %less myadder.v
        %less myadder.vhd
```

```
// File: myadder.v  
// Generated by MyHDL 1.0dev  
// Date: Sun Oct 14 06:42:08 2018
```

```
`timescale 1ns/10ps
```

```
module myadder (  
    a,  
    b,  
    c  
);
```

```
input signed [3:0] a;  
input signed [3:0] b;  
output signed [4:0] c;  
wire signed [4:0] c;
```

```
assign c = (a + b);
```

```
endmodule
```

```
-- File: myadder.vhd
-- Generated by MyHDL 1.0dev
-- Date: Sun Oct 14 06:43:01 2018
```

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

```
use work.pck_myhdl_10.all;
```

```
entity myadder is
    port (
        a: in signed (3 downto 0);
        b: in signed (3 downto 0);
        c: out signed (4 downto 0)
    );
end entity myadder;
```

```
architecture MyHDL of myadder is
```

```
begin
```

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

```
end architecture MyHDL;
```

```

In [4]: from myhdl import Simulation
        from myhdlpeek import Peeker, PeekerGroup

        Peeker.clear()
        PeekerGroup(a=a, b=b, c=c)

        @hdl.block
        def bench():
            tbdut = myadder(a, b, c)

            @hdl.instance
            def tbstim():
                a.next = 1
                b.next = 2
                yield hdl.delay(10)
                print("a: {}, b: {}, c: {}".format(int(a), int(b), int(c)))
                b.next = -4
                yield hdl.delay(10)
                print("a: {}, b: {}, c: {}".format(int(a), int(b), int(c)))
                yield hdl.delay(10)
                a.next = 0
                b.next = 0

            return tbdut, tbstim

        sim = Simulation(bench(), *Peeker.instances()).run()

```

```

a: 1, b: 2, c: 3
a: 1, b: -4, c: -3

```

```

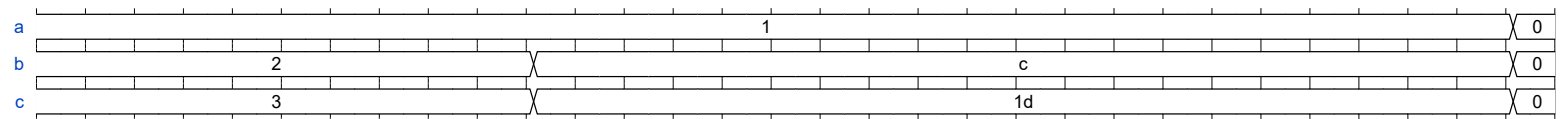
<class 'myhdl.StopSimulation'>: No more events

```



In [5]: `Peeker.to_wavedrom()`

Out[5]:



In [6]: `Peeker.to_html_table()`

Time	a	b	c
0	1	2	3
10	1	-4	-3
30	0	0	0

# Interface driven design

- Create interfaces: logical grouping of signals
- use methods in the interface to simulate transactions

## A block using ports

```
@hdl.block
def myblock(data_in, data_in_valid, data_in_ready,
            data_out, data_out_valid, data_out_ready,
            enable, clock, reset):
    # block logic behavior ...
```

## A block using interfaces

```
@hdl.block
def myblock(glbl: Global, sig_i: Samples, sig_o: Samples):
    # block logic behavior ...
```

## An interface definition

```
import myhdl as hdl
from myhdl import Signal, intbv

class Samples(object):
    def __init__(self):
        self.data = Signal(intbv(0, min=-8, max=8))
        self.valid = Signal(bool(0))
        self.ready = Signal(bool(0))

    def write(self, val):
        pass

    def read(self, val):
        pass
```

# Interfaces simple class definition

- Conversion name expansion
- IDE easy access to attributes

```
In [7]: import myhdl as hdl
        from myhdl import Signal, ResetSignal, intbv

        class Samples(object):
            def __init__(self):
                self.data = Signal(intbv(0, min=-8, max=8))
                self.valid = Signal(bool(0))
                self.ready = Signal(bool(0))

            def write(self, val):
                pass

            def read(self, val):
                pass

        class Global(object):
            def __init__(self):
                self.clock = Signal(bool(0))
                self.reset = ResetSignal(0, active=1, async=False)
                self.enable = Signal(bool(0))
```

In a testbench, use the transactor methods ...

```
In [8]: @hdl.block
def bench():
    sig_i, sig_o = Samples(), Samples()
    tbdut = myblock()

    @hdl.instance
    def tbwrite():
        yield sig_i.write(0x55)

    @hdl.instance
    def tbread():
        yield sig_o.read()

    return tbdut, tbstim
```



# Scale to manage system complexity

With the interface and fully executable elaboration

```
# Coalesced, output, data stream interface
data_out = DataStream()

# A list of data packet interface objects
packets = [DataPackets(data=dd)
            for dd in data_streams]

# Empty block instance list
ps_insts = []
for dd, bb in zip(data_streams, packets):
    # Instantiate a data packet stuffer for each
    # DataPacket interface defined in the design.
    ps_insts.append(
        data_packet_stuffer(
            glbl, dd, timestamp, bb
        ))

route_inst = coalesce_packets(glbl, *packets, data_out)
```

**The end**