eP16MyHDL

July 24, 2016

1 eP16

This file and it's support can be found @ https://github.com/DRuffer/eP16MyHDL

To run this in the interactive editor, type: **jupyter notebook eP16MyHDL.ipynb**.

To just run as a script, type: **runipy eP16MyHDL.ipynb**. This is not working at the moment.

To get PDF output, type: **jupyter nbconvert –to pdf eP16MyHDL.ipynb**. However, you have to setup Pandoc and LaTeK.

After runipy, **echo Exit Code is %ERRORLEVEL%** displays: **True** if the assert passes and **False** if it does not.

Note that it's called `$LastExitCode` in PowerShell.

```
In [1]: import re
        import sys
        import time
        import unittest
        import subprocess
        from myhdl import *
        from random import randrange

        print ("Current date and time %s"  % time.strftime("%c") )

        print ("Python version %s" % sys.version.replace("|","\n\r"))
```

```
Current date and time 07/23/16 14:54:01
Python version 2.7.12
Continuum Analytics, Inc.
 (default, Jun 29 2016, 11:07:13) [MSC v.1500 64 bit (AMD64)]
```

1.1 DFF

Starting from the MyHDL D flip-flop example at http://www.myhdl.org/docs/examples/flipflops.html

```
In [2]: def dff(led_0, sw_7, clk):

            @always(clk.posedge)
            def logic():
                led_0.next = sw_7

            return logic

        def test_dff():

            q, d, clk = [Signal(bool(0)) for i in range(3)]
```

```
            dff_inst = dff(q, d, clk)

            @always(delay(10))
            def clkgen():
                clk.next = not clk

            @always(clk.negedge)
            def stimulus():
                d.next = randrange(2)

            return dff_inst, clkgen, stimulus


        def simulate(timesteps):
            tb = traceSignals(test_dff)
            sim = Simulation(tb)
            sim.run(timesteps)

        simulate(2000)
```

```
<class 'myhdl._SuspendSimulation'>: Simulated 2000 timesteps
```

I'm going to skip the simulation part, for the moment, because I'm more interested in the VHDL output

```
In [3]: q, d, clk = [Signal(bool(1)) for i in range(3)]

        for f in (toVHDL, toVerilog):
            f(dff, q, d, clk)
```

1.2 UART

To start this process off, let's start with the UART, so I can get feedback that can actually be tested. The **eP16.lpf** file gives the key to find: `uart_o`. That leads us to **ep16_chip.vhd**, which gives us the connection to `txd_o` that is defined in **uart16.vhd**.

1.3 Output

Now, we can tell Lattice Diamond to do its thing.

```
In [4]: p = subprocess.Popen('pnmainc ep16MyHDL.tcl', shell=True, \
                             stdout=subprocess.PIPE, \
                             stderr=subprocess.STDOUT)
        for line in p.stdout.readlines():
            line = re.sub(r'[^ -}]', '', line.strip())
            if ((-1 != line.find("WARNING")) or (-1 != line.find("ERROR"))):
                print (line)
        p.wait()
```

```
WARNING - par: The following clock signals will be routed by using generic routing resource and may suf
WARNING - par: The following clock signals will be routed by using generic routing resource and may suf
```

```
Out[4]: 0
```

At this point, we should have some results on the serial port, but not until we get there.
Instead, led_0 turns on when sw_7 is pressed.
Good 1st test.