ESL Laboratorium

Wyniki wykonania tutoriali

Piotr Prusak

Elektronika IV rok

Nr. Indeksu:305098

Introduction to MyHDL

A basic MyHDL simulation

```
from myhdl import block, delay, always, now

@block
def HelloWorld():

    @always(delay(10))
    def say_hello():
        print("%s Hello World!" % now())

    return say_hello


inst = HelloWorld()
inst.run_sim(30)
```

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

Signals and concurrency

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

@block
def HelloWorld():

    clk = Signal(0)

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

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

    return drive_clk, say_hello


inst = HelloWorld()
inst.run_sim(50)
```

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

Parameters, ports and hierarchy

```
from myhdl import block, delay, instance


@block
def ClkDriver(clk, period=20):

    lowTime = int(period / 2)
    highTime = period - lowTime

    @instance
    def drive_clk():
        while True:
            yield delay(lowTime)
            clk.next = 1
            yield delay(highTime)
            clk.next = 0

    return drive_clk
```

```
from myhdl import block, always, now


@block
def Hello(clk, to="World!"):

    @always(clk.posedge)
    def say_hello():
        print("%s Hello %s" % (now(), to))

    return say_hello
```

```
from myhdl import block, Signal
```

```
from Clk import ClkDriver
from hello import Hello


@block
def Greetings():

    clk1 = Signal(0)
    clk2 = Signal(0)

    clkdriver_1 = ClkDriver(clk1)  # positional and default association
    clkdriver_2 = ClkDriver(clk=clk2, period=19)  # named association
    hello_1 = Hello(clk=clk1)  # named and default association
    hello_2 = Hello(to="MyHDL", clk=clk2)  # named association

    return clkdriver_1, clkdriver_2, hello_1, hello_2


inst = Greetings()
inst.run_sim(50)
```

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

Process finished with exit code 0
```

Hardware-oriented types

The intbv class

```
In[6]: from myhdl import intbv
In[7]: >>> a = intbv(24)
In[8]: >>> print(a.min)
None
In[9]: >>> print(a.max)
None
In[10]: >>> len(a)
Out[10]: 0

In[11]:
```

```
In[14]: >>> a = intbv(24, min=0, max=25)
In[15]: >>> print(a.min)
0
In[16]: >>> print(a.max)
25
In[17]: >>> len(a)
Out[17]: 5
```

```
In[18]: >>> a = intbv(6, min=0, max=7)
   ...: >>> len(a)
Out[18]: 3
In[19]: >>> a = intbv(6, min=-3, max=7)
   ...: >>> len(a)
Out[19]: 4
In[20]: >>> a = intbv(6, min=-13, max=7)
   ...: >>> len(a)
Out[20]: 5
```

```
In[21]: >>> from myhdl import bin
   ...: >>> a = intbv(24)
   ...: >>> bin(a)
Out[21]: '11000'
In[22]: >>> int(a[0])
Out[22]: 0
In[23]: >>> int(a[3])
Out[23]: 1
In[24]: >>> b = intbv(-23)
   ...: >>> bin(b)
Out[24]: '101001'
In[25]: >>> int(b[0])
Out[25]: 1
In[26]: >>> int(b[3])
Out[26]: 1
In[27]: >>> int(b[4])
Out[27]: 0
```

Bit slicing

```
In[29]: >>> a = intbv(24)
   ...: >>> bin(a)
   ...:
Out[29]: '11000'
In[30]: >>> a[4:1]
Out[30]: intbv(4)
In[31]: bin(a[4:1])
Out[31]: '100'
In[32]: >>> a[4:1] = 0b001
   ...: >>> bin(a)
Out[32]: '10010'
In[33]: >>> a
Out[33]: intbv(18)
```

```
In[35]: >>> a = intbv(24)
In[36]: >>> bin(a)
Out[36]: '11000'
In[37]: >>> bin(a[4:])
Out[37]: '1000'
In[38]: >>> a[4:] = '0001'
   ...: >>> bin(a)
Out[38]: '10001'
In[39]: >>> a[:] = 0b10101
   ...: >>> bin(a)
Out[39]: '10101'
```

```
In[40]: >>> a = intbv(6, min=-3, max=7)
   ...: >>> len(a)
Out[40]: 4
In[41]: >>> b = a[4:]
   ...: >>> b
Out[41]: intbv(6)
In[42]: >>> len(b)
Out[42]: 4
In[43]: >>> b.min
Out[43]: 0
In[44]: >>> b.max
Out[44]: 16
```

```
In[45]: >>> a = intbv(-3)
   ...: >>> bin(a, width=5)
Out[45]: '11101'
In[46]: >>> b = a[5:]
   ...: >>> b
Out[46]: intbv(29)
In[47]: >>> bin(b)
Out[47]: '11101'
```

```
In[48]: >>> a = intbv(24)[5:]
In[49]: >>> a.min
Out[49]: 0
In[50]: >>> a.max
Out[50]: 32
In[51]: >>> len(a)
Out[51]: 5
```

Unsigned and signed representation

```
In[55]: >>> a = intbv(12, min=0, max=16)
   ...: >>> bin(a)
Out[55]: '1100'
In[56]: >>> b = a.signed()
   ...: >>> b
Out[56]: intbv(-4)
In[57]: >>> bin(b, width=4)
Out[57]: '1100'
```

Structural modeling

Example

```
from myhdl import block, always_comb, Signal

@block
def mux(z, a, b, sel):

    """ Multiplexer.

    z -- mux output
    a, b -- data inputs
    sel -- control input: select a if asserted, otherwise b
```

```
    """

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

    return comb
```

```
import random
from myhdl import block, instance, Signal, intbv, delay
from mux import mux

random.seed(5)
randrange = random.randrange

@block
def test_mux():

    z, a, b, sel = [Signal(intbv(0)) for i in range(4)]

    mux_1 = mux(z, a, b, sel)

    @instance
    def stimulus():
        print("z a b sel")
        for i in range(12):
            a.next, b.next, sel.next = randrange(8), randrange(8),
randrange(2)
            yield delay(10)
            print("%s %s %s %s" % (z, a, b, sel))

    return mux_1, stimulus

tb = test_mux()
tb.run_sim()
```

```
<class 'myhdl.StopSimulation'>: No more events
z a b sel
5 4 5 0
3 7 3 0
2 2 1 1
7 7 3 1
3 1 3 0
3 3 6 1
6 2 6 0
1 1 2 1
2 2 2 0
3 0 3 0
2 2 2 1
3 5 3 0

Process finished with exit code 0
```

Finite State Machine modeling

```
from myhdl import block, always_seq, Signal, intbv, enum

ACTIVE_LOW = 0
FRAME_SIZE = 8
t_state = enum('SEARCH', 'CONFIRM', 'SYNC')

@block
def framer_ctrl(sof, state, sync_flag, clk, reset_n):

    """ Framing control FSM.

    sof -- start-of-frame output bit
    state -- FramerState output
    sync_flag -- sync pattern found indication input
    clk -- clock input
    reset_n -- active low reset

    """

    index = Signal(intbv(0, min=0, max=FRAME_SIZE)) # position in frame

    @always_seq(clk.posedge, reset=reset_n)
    def FSM():
        if reset_n == ACTIVE_LOW:
            sof.next = 0
            index.next = 0
            state.next = t_state.SEARCH

        else:
            index.next = (index + 1) % FRAME_SIZE
```

```
            sof.next = 0

            if state == t_state.SEARCH:
                index.next = 1
                if sync_flag:
                    state.next = t_state.CONFIRM

            elif state == t_state.CONFIRM:
                if index == 0:
                    if sync_flag:
                        state.next = t_state.SYNC
                    else:
                        state.next = t_state.SEARCH

            elif state == t_state.SYNC:
                if index == 0:
                    if not sync_flag:
                        state.next = t_state.SEARCH
                sof.next = (index == FRAME_SIZE-1)

            else:
                raise ValueError("Undefined state")

    return FSM
```

```
import myhdl
from myhdl import block, always, instance, Signal, ResetSignal, delay,
StopSimulation
from fsm import framer_ctrl, t_state
ACTIVE_LOW = 0

@block
def testbench():

    sof = Signal(bool(0))
    sync_flag = Signal(bool(0))
    clk = Signal(bool(0))
    reset_n = ResetSignal(1, active=ACTIVE_LOW, isasync=True)
    state = Signal(t_state.SEARCH)

    frame_ctrl_0 = framer_ctrl(sof, state, sync_flag, clk, reset_n)

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

    @instance
    def stimulus():
        for i in range(3):
            yield clk.negedge
        for n in (70,12, 8, 8, 4):
            sync_flag.next = 1
            yield clk.negedge
            sync_flag.next = 0
            for i in range(n-1):
                yield clk.negedge
        raise StopSimulation()

    return frame_ctrl_0, clkgen, stimulus
```

```
tb = testbench()
tb.config_sim(trace=True)
tb.run_sim()
```