Tallinn University of Technology
School of Information Technologies
Thomas Johann Seebeck Department of Electronics

Gaspar Karm

Pyha

Master's Thesis

Supervisors:
Muhammad Mahtab Alam
PhD

Yannick Le Moullec
PhD

Tallinn 2017

Contents:

1 Pyha **1**

1.1 Introduction . . . 1
1.2 Model based design . . . 2
1.2.1 Pyha flow . . . 2
1.3 Testing/debugging and verification . . . 3
1.3.1 Simplifying testing . . . 3
1.4 Describing hardware . . . 4
1.4.1 Stateless logic . . . 4
1.4.2 Sequential logic . . . 6
1.4.3 Understanding registers . . . 6
1.5 Fixed-point designs . . . 9
1.6 Extended example . . . 9
1.7 Conclusions . . . 10

Chapter 1

Pyha

Pyha is an tool that allows writing of digital hardware in Python language. Currently it focuses mostly on the DSP applications.

Main features:

- Simulate hardware in Python. Integration to run RTL and GATE simulations.
- Structured, all-sequential and object oriented designs
- Fixed point type support(maps to **'VHDL fixed point library'**_)
- Decent quality VHDL conversion output (get what you write, keeps hierarchy)
- Integration to Intel Quartus (run GATE level simulations)
- Tools to simplify verification

1.1 Introduction

This chapter focuses on the Python side of Pyha, while the next chapter gives details on how Pyha details are converted to VHDL and how they can be synthesised.

A multiply-accumulate(MAC) circuit is used as a demonstration circuit throughout the rest of this chapter. It is a good choice as it is powerful element yet not very complex. Last chapter of this thesis peresents more serious use cases.

Note: The first half of this chapter uses 'integers' as base type in order to keep the examples simple. Second half starts using fixed-point numbers, that ade default for Pyha.

1.2 Model based design

Generally before the hardware system is implemented, it is useful to first experiment with the idea and maybe even do some performance figures like SNR. For this, model is constructed. In general the model is the simplest way to archive the task, it is not optimized.

Model allows to focus on the algorithmical side of things. Also model comes in handy when verifying the operation of the hardware model. Output of the model and hardware can be compared to verify that the hardware is working as expected.

In [blade_adsb], open-sourced a ADS-B decoder, implemented in hardware. In this work the authors first implement the model in MATLAB for rapid prototyping. Next they converted the model into C and implemented it using fixed-point arithmetic. Lastly they converted the C model to VHDL.

More common approach is to use MATLAB stack for also the fixed-point simulations and for conversion to VHDL. Also Simulink can be used.

Simulink based design flow has been reported to be used in Berkeley Wireless Research Center (BWRC) [borph]. Using this design flow, users describe their designs in Simulink using blocks provided by Xilinx System Generator [borph].

The problem with such kind of design flow is that it costs alot. Only the MATLAB based parts can easly cost close to 20000 EUR, as the packages depend on eachother. For example for reasonable flow user must buy the Simulink software but that also requires the MATLAB software, in addtion to do DSP, DSP toolbox is needed.. etc.

Also the FPGA vendor based tools, like Xilinx System Generator are also expensive and billed annually.

While this workflow is powerful indeed.

Model based design, this is also called behavioral model (.. https://books.google.ee/books?hl=en&lr=&id=XbZr8DurZYEC&oi=fnd&pg=PP1&dq=vhdl&ots=PberwiAymP&sig=zqc4BUSmFZaL3hxRilU-J9Pa_5I&redir_esc=y#v=onepage&q=vhdl&f=false)

1.2.1 Pyha flow

Pyha is fully open-source software, meaning it is a free tool to use by anyone. Since Pyha is based on the Python programming language, it gets all the goodness of this environment.

Python is a popular programming language which has lately gained big support in the scientific world, especially in the world of machine learning and data science. It has vast support of scientific packages like Numpy for matrix math or Scipy for scientific computing in addition it has many superb plotting libraries. Many people see Python scientific stack as a better and free MATLAB.

As far as what goes for model writing, Python comes with extensive schinetific stuff. For example Scipy and Numpy. In addition all the GNURadio blocks have Python mappings.

VHDL uuendused? VUNIT VUEM?

Test-driven development / unit-tests

Model based development How MyHDl and other stuffs contribute here?

Since Pyha brings the development into Python domain, it opens this whole ecosystem for writing testing code.

Listing 1.1: Multiply-accumulate written in Python

```
class MAC:
    def __init__(self, coef):
        self.coef = coef

    def model_main(self, sample_in, sum_in):
        import numpy as np

        muls = np.array(sample_in) * self.coef
        sums = muls + sum_in
        return sums
```

Listing 1.1 shows the MAC model written in Python. It uses the Numpy package for numeric calculations.

1.3 Testing/debugging and verification

1.3.1 Simplifying testing

One problem for model based designs is that the model is generally written in some higher level language and so testing the model needs to have different tests than HDL testing. That is one ov the problems with CocoTB.

Pyha simplifies this by providing an one function that can repeat the test on model, hardware-model, RTL and GATE level simulations.

- Siin all ka unit testid?

Python ships with many unit-test libraries, for example PyTest, that is the main one used for Pyha.

Siin peaks olema test funksioonid?

1.4 Describing hardware

Assuming we have now enough knowledge and unit-tests we can start implementing the Hardware model.

Main idea of Pyha is to enable hardware design in Python ecosystem.

Pyha extends the VHDL language by allowing objective-oriented designs. Unit object is Python class as shown on

Listing 1.2: Basic Pyha unit

```
class PyhaUnit(HW):
    def __init__(self, coef):
        pass

    def main(self, input):
        pass

    def model_main(self, input_list):
        pass
```

Listing 1.2 shows the besic design unit of the developend tool, it is a standard Python class, that is derived from a baseclass *HW, purpos of this baseclass is to do some metaclass stuff and register this class as Pyha module.

Metaclass actions:

1.4.1 Stateless logic

Stateless is also called combinatory logic. In the sense of software we could think that a function is stateless if it only uses local variables, has no side effects, returns are based on inputs only. That is, it may use local variables of function but cannot use the class variables, as these are stateful.

Listing 1.3: Stateless MAC implemented in Pyha

```
class MAC(HW):
    def main(self, x, sum_in):
        mul = 123 * x
        y = sum_in + mul
        return y

    def model_main ...
```

Listing 1.3 shows the design of a combinatory logic. In this case it is a simple xor operation between two input operands. It is a standard Python class, that is derived from a baseclass

*HW, purpose of the baseclass is to do some metaclass stuff and register this class as Pyha module.

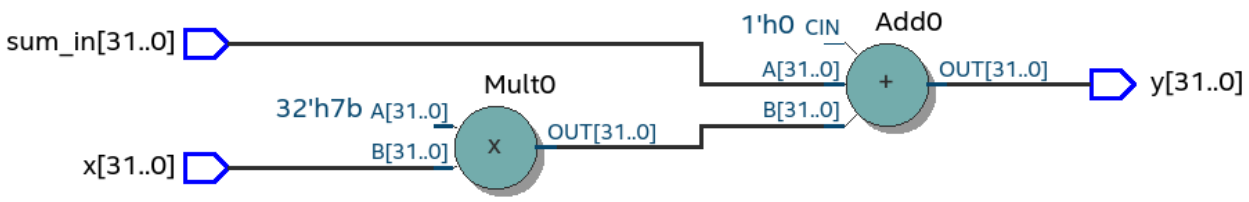

Fig. 1.1: Synthesis result of the revised code (Intel Quartus RTL viewer)

Fig. 1.1 shows the synthesis result of the source code shown in `mac-next-update`. It is clear that this is now equal to the system presented at the start of this chapter.

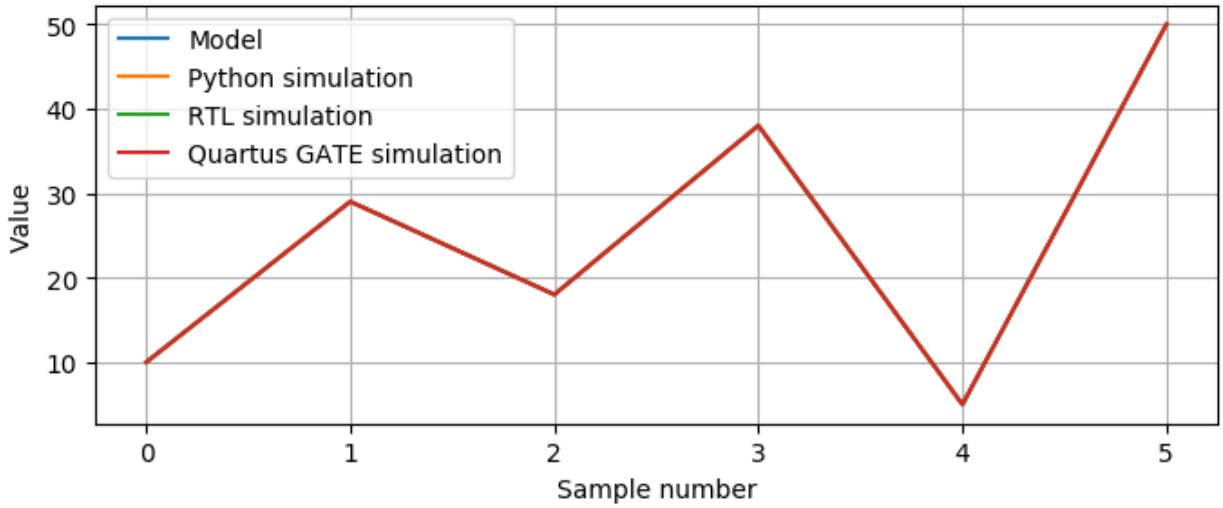

Fig. 1.2: Synthesis result of the revised code (Intel Quartus RTL viewer)

Class contains an function 'main', that is considered as the top level function for all Pyha designs. This function performs the xor between two inputs 'a' and 'b' and then returns the result.

In general all assigments to local variables are interpreted as combinatory logic.

Todo

how this turns to VHDL and RTL picture?

In software operations consume time, but in hardware they consume resources, general rule.

Not clocked...basically useless analog stuff.

1.4.2 Sequential logic

1.4.3 Understanding registers

Clearly the way of defining registers is not working properly. The mistake was to expect that the registers work in the same way as 'class variables' in traditional programming languages.

In traditional programming, class variables are very similar to local variables. The difference is that class variables can 'remember' the value, while local variables exist only during the function execution.

Hardware registers have just one difference to class variables, the value assigned to them does not take effect immediately, but rather on the next clock edge. That is the basic idea of registers, they take a new value on clock edge. When the value is set at **this** clock edge, it will be taken on **next** clock edge.

Trying to stay in the software world, we can abstract away the clock edge by thinking that it denotes the call to the 'main' function. Meaning that registers take the assigned value on the next function call, meaning assignment is delayed by one function call.

VHDL defines a special assignment operator for this kind of delayed assignment, it is called 'signal assignment'. It must be used on VHDL signal objects like `a <= b`.

Jan Decaluwe, the author of MyHDL package, has written a relevant article about the necessity of signal assignment semantics [jan_myhdl_signals].

Using an signal assignment inside a clocked process always infers a register, because it exactly represents the register model.

Registers in hardware have more purposes:

- delay
- max clock speed - how this corresponds to sample rate?

Explain somwhere that each call to function is a clock tick.

Listing 1.4: Basic sequential circuit in Pyha

```
class Reg(HW):
    def __init__(self):
        self.reg = 0

    def main(self, a, b):
        self.next.reg = a + b
        return self.reg
```

Listing 1.4 shows the design of a registered adder.

Fig. 1.3 shows the synthesis result of the source code shown in `mac-next-update`. It is clear that this is now equal to the system presented at the start of this chapter.

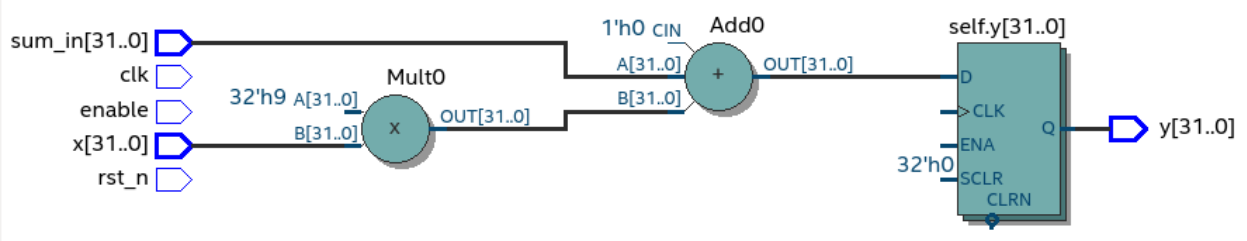

Fig. 1.3: Synthesis result of the revised code (Intel Quartus RTL viewer)

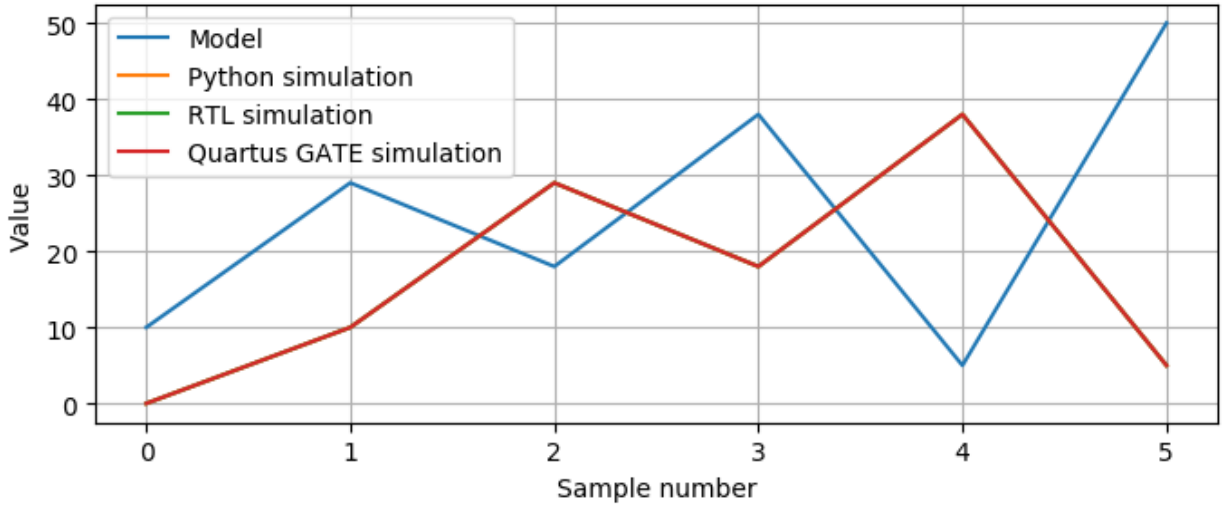

Fig. 1.4: Synthesis result of the revised code (Intel Quartus RTL viewer)

Running the same testing code results in a Fig. 1.4. It shows that while the Python, RTL and GATE simulations are equal, model simulation differs. This is the effect of added register, it adds one delay to the harwdware simulations.

This is an standard hardware behaviour. Pyha provides special variable `self._delay` that specifies the delay of the model, it is useful:

- Document the delay of your blocks
- Upper level blocks can use it to define their own delay
- Pyha simulations will adjust for the delay, so you can easily compare to your model.

Note: Use `self._delay` to match hardware delay against models

After setting the `self._delay = 1` in the __init__, we get:

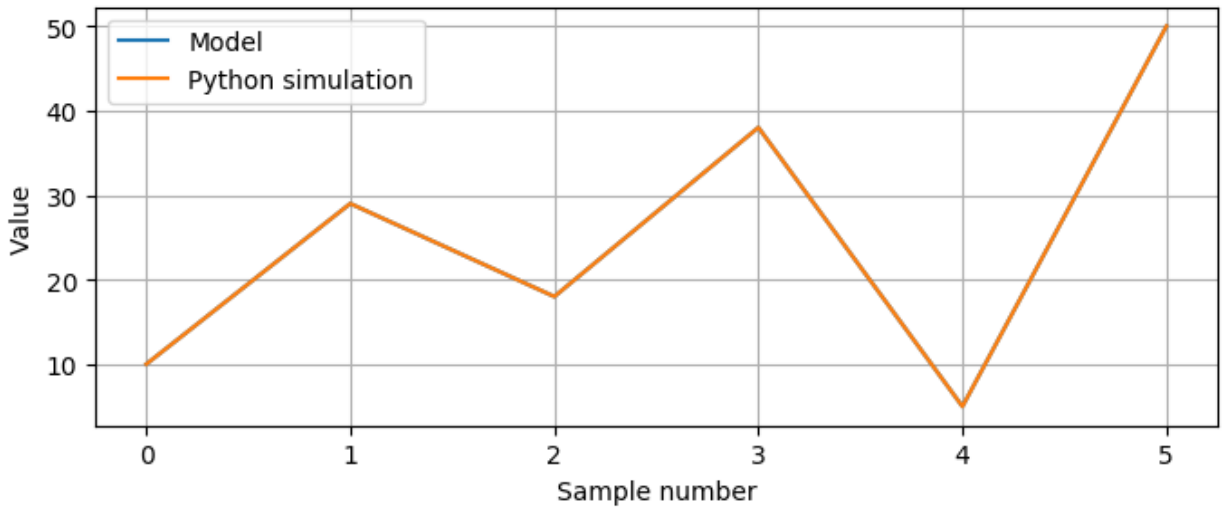

Fig. 1.5: Synthesis result of the revised code (Intel Quartus RTL viewer)

In Pyha, registers are inferred from the ogject storage, that is everything defined in 'self' will be made registers.

The 'main' function performs addition between two inputs 'a' and 'b' and then returns the result. It can be noted that the sum is assigned to 'self.next' indicating that this is the next value register takes on next clock.

Also returned is self.reg, that is the current value of the register.

In general this system is similiar to VHDL signals:

- Reading of the signal returns the old value
- Register takes the next value in next clock cycle (that is self.next.reg becomes self.reg)

- Last value written to register dominates the next value

However there is one huge difference aswell, namely that VHDL signals do not have order, while all Pyha code is stctural.

Todo

how this turns to VHDL and RTL picture?

Pyha way is to register all the outputs, that way i can be assumed that all the inputs are already registered.

Simulation a

1.5 Fixed-point designs

1.6 Extended example

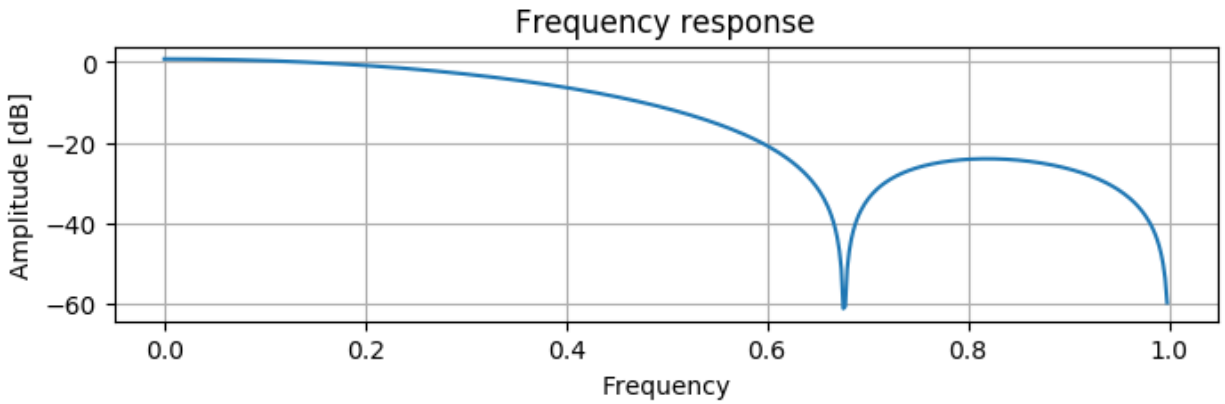

Fig. 1.6: Synthesis result of the revised code (Intel Quartus RTL viewer)

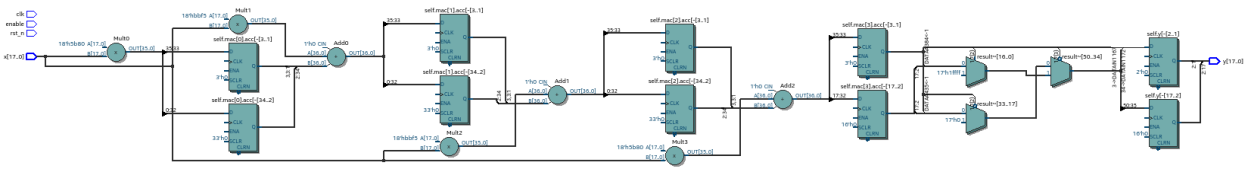

Fig. 1.7: Synthesis result of the revised code (Intel Quartus RTL viewer)

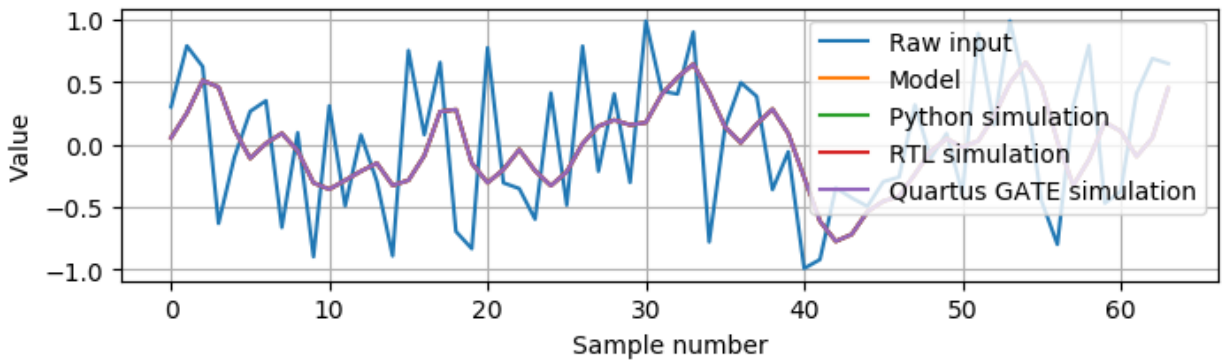

Fig. 1.8: Synthesis result of the revised code (Intel Quartus RTL viewer)

1.7 Conclusions

This chapter showed how Python OOP code can be converted into VHDL OOP code.

It is clear that Pyha provides many conveneince functions to greatly simplyfy the testing of model based designs.