Using scripting languages for hardware/software co-design

Evangelos Logaras*

National and Kapodistrian University of Athens,
Department of Informatics and Telecommunications
evlog@di.uoa.gr

Abstract. In this thesis we present a new vertical methodology targeting the hw/sw co-design of embedded SoCs. For the suggested methodology a digital design and verification tool named System Python (SysPy) has been developed, using the strengths of the popular Python scripting language. We exploit the features of the language to boost the productivity of processor-centric SoC designs for Field Programmable Gate Arrays (FPGAs) implementation. In more details we developed methods to: (a) support hardware descriptions using Python syntax and automatically generate synthesizable VHDL code. (b) Support Python descriptions for simulating the behavior of an embedded SoC in algorithmic/functional level or using Register Transfer Level (RTL) descriptions and generate digital simulation plots in a bit-true and cycle-accurate manner. (c) Support the use of C software development tools for the programming of the processor core and (d) automatically generate Tcl scripts to integrate with FPGA implementation tools and ease synthesis and physical implementation steps. Complex SoC's have been designed and implemented in FPGA devices and used as design cases to demonstrate the features of the supported design flow. All designs use a processor IP core as the main programmable system controller, used for data processing. Each design follows the progress that we had in the development of our methodology and highlights certain features of the tool. We believe that with our methodology we cover the lack of existence of tools targeting the hw/sw co-design and prototyping of FPGA based embedded SoCs.

Keywords: Python, Processor-centric SoCs, hw/sw co-design, VHDL, FPGA, SysPy

1 Introduction

Modern Field Programmable Gate Array (FPGA) devices can host very complex digital designs. Most of the implemented System on Chips (SoCs) incorporate at least one programmable microprocessor (uP) unit. The processor's Intellectual Property (IP) core is key elements for the rapid prototyping of new digital systems, but on the other hand its usage raises a lot of design challenges that have to be addressed in the design flow.

* Dissertation Advisor: Elias S. Manolakos, Associate Professor.

The main goal of this dissertation was the development of methods and of a design tool, called System Python (SysPy) targeting the hardware/software co-design and verification, using hight-level abstract descriptions, of processor-centric SoCs implemented using FPGAs. For the needs of the research, we evaluated Python's programming features and especially the combination of scripting capabilities in a Linux shell, combined with Object Oriented Programming (OOP) features. These supported features could be used to:

- Implement high-level abstract models of blocks, e.g. arithmetic, memory and logic blocks, connect them using structural Python descriptions and translate them automatically to FPGA synthesizable Very high speed integrated circuit Hardware Description Language (VHDL), or use them to perform Register Transfer Level (RTL) bit-true simulation of a system. The integration of the SciPy library in Python provides a large number of functions which can be used for modeling arithmetic blocks.
- Build a framework and a design tool that implements the end-to-end design flow of a processor-centric system-on-chip, which invokes/calls other hardware and software related tools, e.g. logic synthesizers, software compilers, simulation tools etc.
- Process the large number of text-based files generated during a hardware design flow. Information extracted from generated text files is used many times as an input for the next design step or can be transformed/parsed to a different format.

While designing a complex digital system cannot be done automatically at a press of a button, we envisioned a design tool that would integrate the majority of the tools needed for an FPGA implementation of an embedded SoC. The first and most difficult task was to build the Python-to-VHDL parser. For this task we needed to define our supported coding style/syntax for the hardware descriptions in Python. The syntax should support a level of abstraction but on the other hand support features that are used in well established Hardware Description Language (HDL) languages, like VHDL and Verilog. A lexical analyzer also needed to recognize and track, in the user supplied Python descriptions, the supported syntax and parse these parts of Python code that later on would be mapped and translated to synthesizable VHDL.

The main contribution of this dissertation is to show that a modern programming language like Python can be used to design, simulate and implement processor-centric embedded SoCs, using high-level, abstract descriptions. This is very useful especially early in the design flow when control and processing logic of a system must be partitioned among software and hardware implementation. Our research work also shows convincingly that Python is a good candidate language to handle the large number of design tools needed to capture and implement a SoC in an FPGA device, in terms of hardware and software development.

2 Importance of processor-centric Systems-on-Chip

Most of the complex SoC designs implemented nowadays using FPGAs are processor-centric, where the uP is used as a gateway, implementing in software different communication protocols needed to exchange data with other logic devices on the same board or with a connected PC/server. Software executed by the processor is also used to control and exchange data with other custom blocks implemented in the FPGA fabric. Complex control and data processing logic can now be easily partitioned between software and hardware inside the same FPGA device. The latest devices from FPGA vendors, such as the Virtex-7 device from Xilinx or the Arria-V device from Altera, include fast embedded dual-core ARM processors which can communicate with the rest of the FPGA fabric using a very fast data and control bus. New designs can be efficiently prototyped within a few days, while the performance and power consumption of a SoC in an FPGA device can be now compared with that of an ASIC implementation. Of course for mass IC production the ASIC device remains the only option, especially in terms of cost reduction.

2.1 Python's innovative features used for digital design

For describing digital systems, choosing the proper language to develop the tool was of great importance. The special features that are required in a design flow targeting processor-centric designs and hw/sw co-design are:

- Integrate under the same scripting environment all the required tools for hardware/software co-development and co-simulation.
- Support clear and powerful syntax that could be used to describe hardware digital designs using HDL-like syntax or in an abstract way, using functions to auto-generate HDL code.

Figure 1 shows how we have used Python to handle in SysPy the integration of the ready-to-use processor IP-core in a SoC design. FPGA synthesizable VHDL code is generated from Python description for custom blocks along with data/control bus interface/glue logic. SysPy generates Tcl scripts for each one of the supported processor IP cores, which executes along with the FPGA synthesis tools in a command line, in order to incorporate the processor subsystem in the design. The tool also compiles, using the GNU Compiler Collection (GCC) C tools, the processor's control software along with any existing O/S kernel. FPGA bitstream file along with the compiled software binary files are generated, since SysPy makes all the necessary external tool calls (synthesizer, compiler), and can be used for FPGA implementation. In previous work [13] we have presented how SysPy can be used to provide Python level hardware descriptions to ease the design flow of processor-centric SoCs implemented on FPGA devices.

Python has been used for the development of commercial and widely used complex hardware and especially software development tool projects, where the text manipulation and generation features of the language are exercised in the

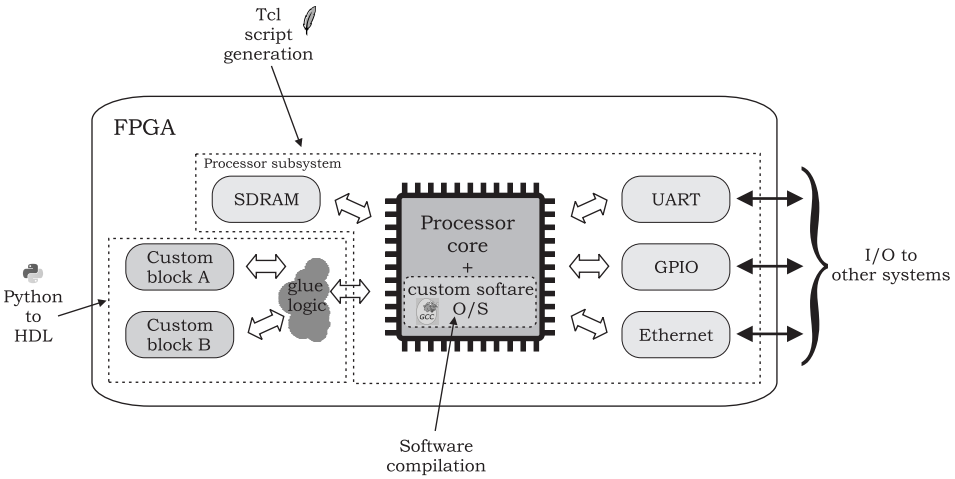

Fig. 1. Generic processor-centric SoC diagram.

best way. Earlier efforts using Python in hardware design include the following: *PyHDL* [1] allowed for structural descriptions which simplified system design using optimized hardware objects. *PHDL* [2] used Python to support a higher level of abstraction for hardware design. A designer can structurally describe a system using components from a Python library and parameters' selection can alter the size of a module, e.g. the bus width. Other tools used Python directly as an HDL. *MyHDL* [3] supports, as SysPy, behavioral, dataflow and structural hardware blocks design capture, and provides behavioral simulation functionalities, presenting text based simulation results. In addition, *MyHDL* can translate Python descriptions to either VHDL or Verilog. However, unlike SysPy, the above methods do not allow mixing Python with VHDL module descriptions in a design (all system components must be described in Python). *PyMTL* [4] targets digital hardware design by unifying functional, cycle-accurate and RTL hardware descriptions. The tool provides HDL generation, by translating Python descriptions to Verilog. Emphasis is placed on accelerating the functional and RTL simulation by converting Python testbenches to C++. However, unlike SysPy, the tool does not generate any files, e.g. Tcl scripts, folder hierarchy etc., to accelerate the FPGA implementation process by synthesis tools. Moreover, the simulation models do not include timing and latency information. Most important of all, none of the above described efforts focuses on handling processor IP cores and their software development environment. None of the above also supports creation of high-level verification models, in the way SysPy does, i.e. by using algorithmic descriptions mixed with RTL level descriptions.

In Table 1 a comparison of the Python related tools targeting digital hardware design is presented. All of the tools support Python code translation to RTL code, but only SysPy and *PHDL* support generation of FPGA synthesizable RTL code. Except from SysPy, only *MyHDL* supports behavioral simulation of

a designed system at the Python level, but no Value Change Dump (VCD) file generation is supported, which is the industry's standard for storing simulation results. Except SysPy, none of the other tools support:

1. processor-centric designs and related C compiler tools handling for software development.
2. RTL code generation using parameterized Python functions.
3. abstract simulation of a design by mapping hardware functionality to Python function and classes.
4. hw/sw co-design using high-level hardware descriptions along with software expressed using C code
5. simulation and generation of VCD files for top-level I/O signal visualization.
6. automatic generation of Tcl scripts for FPGA synthesis tools.

	Support for						
Tools	**Python to RTL**	**FPGA synthesizable code**	**Behavioral simulation**	**Hw/Sw co-design**	**Processor-centric design**	**Synthesis tools integration**	References
PyHDL	**X**	-	-	-	-	-	[1]
PHDL	**X**	**X**	-	-	-	-	[2]
myHDL	**X**	-	**X**	-	-	-	[3]
PyMTL	**X**	**X**	**X**	-	-	-	[4]
SysPy	**X**	**X**	**X**	**X**	**X**	**X**	[5]

Table 1. Python digital hardware related tools comparison.

In all design and simulation steps of our design flow we use Python structures/syntax, and not any custom-defined syntax, to describe the datapath of the SoC and the related simulation models. All supported hardware description and simulation programming structures are compatible with the basic coding style used by the majority of Python programmers. In this way we tried to ease modeling and implementation of a processor-centric SoC even by software programmers with limited experience in hardware design.

3 SysPy digital design and verification features

In SysPy, Python acts as the backbone of a set of tools for hardware description as well as to incorporate other software tools. A typical design cycle starts by providing the simulation models of the desired system and also the timing information, e.g. main clock frequency, duration of the simulation, input data etc. The hardware description can have an HDL-like syntax supported by SysPy or a more abstract algorithmic-style syntax. The first syntax style can later be translated by SysPy to synthesizable VHDL code, while the later one cannot be directly translated to VHDL, but can help a designer to easily verify the functionality of a system.

All the main features of SysPy are presented in the supported design flow shown in Figure 2. The supported flow covers six major tasks related to the design of a processor-centric SoC:

1. Description in HDL of components (modules) that are going to be connected with a processor soft core.
2. Incorporation in a design of ready-to-use components and connection to a processor core.
3. Functional simulation of Python code describing the behavior of hardware blocks and of software executed by the processor
4. Generation of scripts for automating the software development flow for the processor core, e.g. automated calls to C compiler tools, initialization of the processor's program memory in BRAMs etc.
5. Generation and execution of scripts to automate the processes involved in a SoC's design flow, e.g. Tcl scripts for FPGA synthesis tools etc.
6. Generation of meta-data XML description of Python described IP cores, compatible with the IP-XACT standard [6].

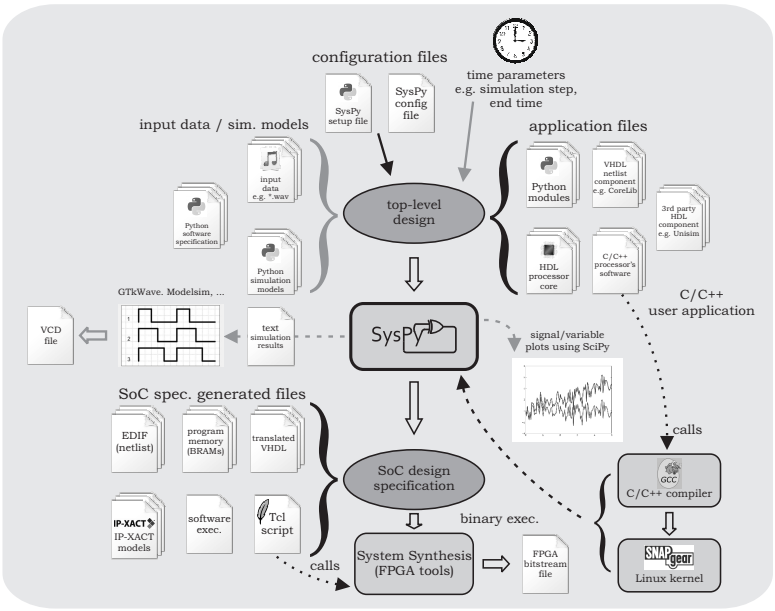

Fig. 2. Processor-centric SoC design flow using SysPy.

To demonstrate the processor-centric design and verification features of our methodology, we present two large SoCs used as design cases to assess and improve our tool's features.

3.1 Biomolecular interaction networks simulation SoC design case

Advanced high performance computational techniques have been used the last decade in several scientific domains to ease and accelerate simulation of complex physical phenomena. Computational and systems biology is a rather new scientific field that takes advantage of computing techniques to describe and simulate complex biological phenomena. Stochastic simulations of biochemical reaction networks, called BioModels [7], [8] can be performed to study the properties of these biological systems. The main idea of the design of our SoC was to accept as an input BioModel files and simulate the chemical reactions of the described system using a stochastic simulation algorithm.

In [9] we have shown how using parallel processing and pipelining it is possible to design a core that performs efficiently, stochastic simulations of biochemical reaction networks (BioModels) based on the First Reaction Method (FRM) of Gillespie's Stochastic Simulation Algorithm (SSA) [10]. Using SysPy's processor-centric design and verification features, in [5],[14] we presented a flexible multi-processor SoC around the Leon3 processor and connected to the SSA core. In this design we proved how using SysPy we can automatically parse a BioModel file in SBML format, which uses XML syntax to define a biochemical reaction network. All the model's important parameters are extracted and used to automatically construct the memory structures and the top-level specification necessary for synthesizing the SoC's FPGA design. With this design approach the SoC can be used to process any BioModel of interest (captured in SBML) without any user intervention or required expertise in FPGA design.

Three different versions of the SSA core implementing the FRM algorithm have been designed, using one (FRM1X), two (FRM2X) or four PEs (FRM4X) operating in parallel (N=1 or 2 or 4). The values of parameters m (number of reactions) and n (number of species) are automatically parsed from the BioModel SBML file, while the rest of the parameters are declared by the user at the top-level Python description. The mode parameter defines the SoC's mode of operation.

SSA core	**Reaction Cycle time (us)**	**MReaction Cycles/sec.**
FRM1X	1.356	0.737
FRM2X	0.93	1.075
FRM4X	0.73	1.37

Table 2. Throughput of the SSA cores at a clock frequency of 160MHz for a network with $m = 136$ reactions and $n = 93$ molecular species.

Using the features of our tool we managed to easily connect the SSA IP core to the Leon3 processor core. Leon is used to connect the SSA core to a host PC and also provides to the core access to a fast SDRAM DDR2 256MB

memory module. The SSA core is connected through an interface FSM which is attached on Leon's AMBA bus, along with all other connected peripheral devices. A top-level schematic of the SSA SoC is shown in Figure 3. For the implementation of the SSA SoC we used an ML509 Xilinx board equipped with a Virtex-5 XC5VLX110T-1 FPGA device. The board has also an SDRAM DDR2 256MB memory module clocked at 190MHz and a PHY Ethernet chip operating at 10/100/1000Mbps.

The implemented SoC delivers performance (0.353MReactions/sec) that is more than an order of magnitude higher compared to a computer cluster [11]. Since performance does not depend heavily on the specific BioModel used but on its complexity we conclude that a well designed FPGA SoC implementation can outperform cluster solutions for complex models. This is also not surprising since the FPGA parallel implementation does not suffer from any costly off-chip communication time overhead for distributing/collecting data during the parallel simulation.

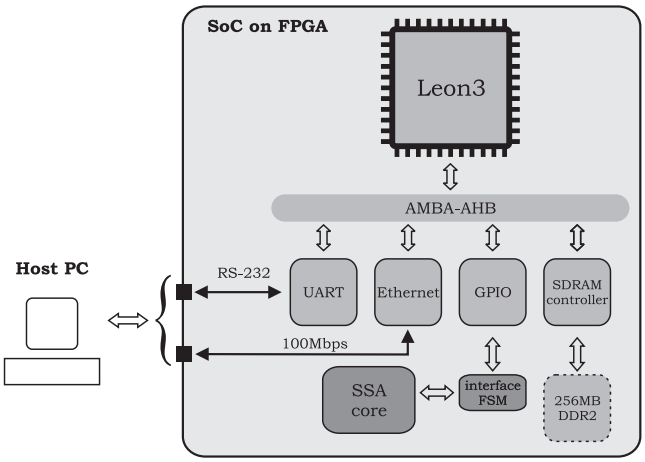

Fig. 3. SSA SoC. Connection of the SSA core to the Leon processor.

3.2 Audio processing SoC design case

Using the design of an audio signal processing SoC [12], we demonstrated the verification and O/S based software development flow supported in SysPy. A processor in an FPGA fetches audio files from a host PC through an FTP connection and analyzes the audio information using a hardwired FIR filter dividing the audible spectrum into four frequency regions. According to the filtered signal, the processor classifies a file into one of four music styles.The goal of SysPy's verification mechanism is to combine in the same testbench abstract algorithmic descriptions, RTL hardware descriptions and embedded software code for the processor core. It is possible to interchangeably use: a) SciPy (Matlab-like) algorithmic descriptions for arithmetic operations mapped in hardware and b) embedded C code executed by a processor core, to build bit-true and cycle-accurate

system-level verification models and generate digital waveforms to assess a SoC's behavior.

In Figure 4, a typical testbench in SysPy i) describes, using HDL-like syntax, the main elements of a pipelined datapath. ii) Python code in SciPy is used to describe in an algorithmic way functionality of hardware blocks not yet defined in HDL. iii) The same functionality can be expressed using C code, in case the required SoC functions need to be ported to software executed by a processor core. iv) Signals plots are generated during simulation in SciPy to observe signals behavior and also SysPy generates VCD files to represent the I/O signals of the system in binary format.

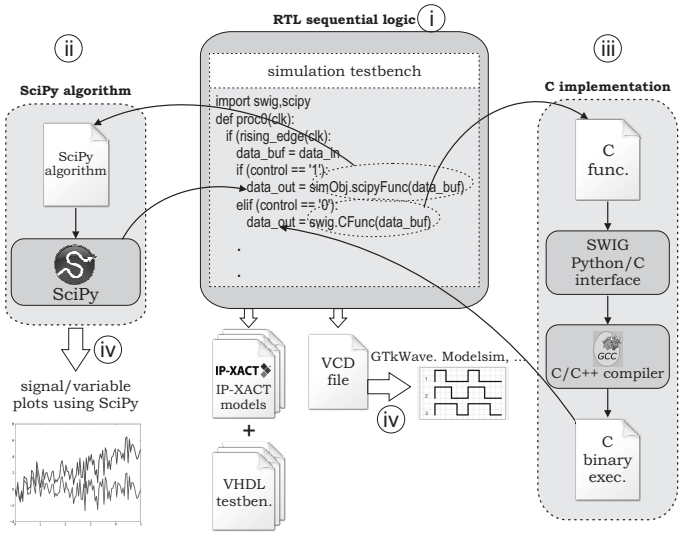

Fig. 4. Simulation flow in SysPy using RTL and algorithmic models.

Using Python simulation models, we were able to make decisions regarding key aspects of the hw/sw design space, such as: a) filter properties e.g. fixed-point notation, number of taps, filter tap value and cutoff frequencies, b) data buffers size and control signals between the processor core and the filter bank and c) software running on the processor in a Linux kernel, that allows data to be handled in a file oriented manner. In Figure 5 a block level schematic of the simulation model is presented that reflects the partitioning of functionality between hardware and software. The text in parentheses designates the type of Python structure used to simulate each block. Software modules are represented using dashed line boxes inside the processors block, while the rest of the blocks correspond to hardware functionality. Software handles music file I/O, transmits the audio samples to the input FIFO, reads back the filtered samples and finally classifies the audio files by analyzing the filtered audio bands. Simulated

hardware functionality in the SoC involves the FIFO memories, the interface FSM handling the data traffic from and to the FIFO memories and the four FIR filters.

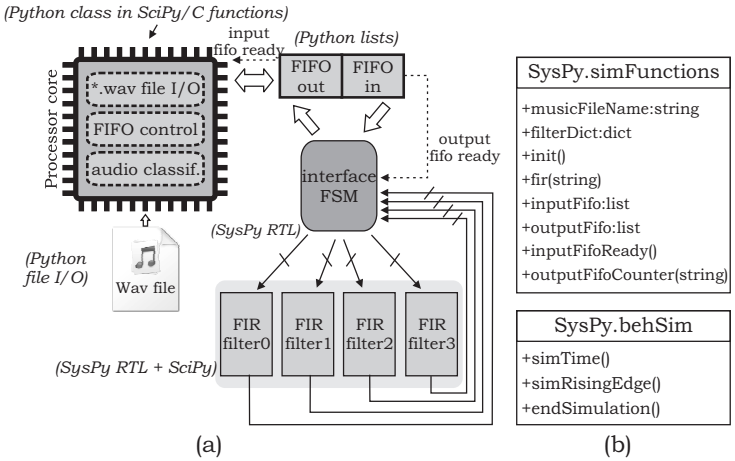

Fig. 5. a) Abstract modeling of the SoC using SysPy and SciPy, b) diagrams of the Python classes used in the SoC's testbench.

Two types of waveforms in SysPy can help a designer decide about system features early in the design flow: a) arithmetic plots in SciPy prove the correctness of an algorithmic model and b) digital signal plots in VCD format, where the variables of an algorithm in SysPy are converted (utilizing ready-to-use functions) to binary fixed-point format (bit-true) and plotted along with time information (cycle-accurate).

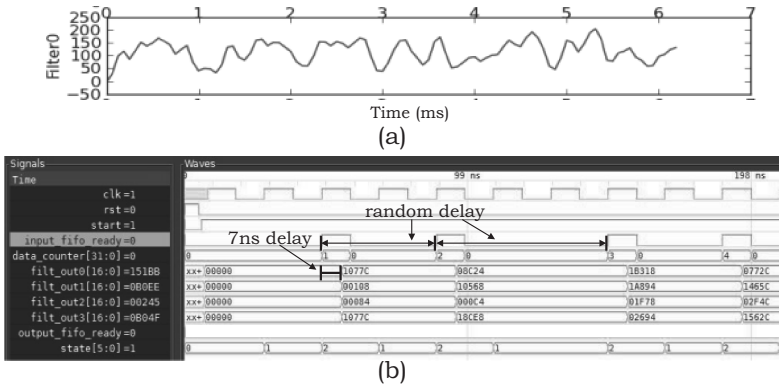

Fig. 6. a) Plots of the filtered audio signals using SciPy, b) waveform of the SoC's I/O signals using GTKWave.

In the waveforms, all the clock, reset, control and data signals, are presented. The filter outputs are presented after a user defined delay in the testbench (7ns). Also the `input_fifo_ready` control signal mimics the way the processor provides

audio samples to the input data buffer asynchronously for realistic simulation. We implemented the system using the ML509 Xilinx board with a medium size Virtex-5 FPGA device (XC5VLX110T-1), 256MB SDRAM DDR2 memory clocked at 190MHz, a PHY Ethernet chip and a serial RS-232 transceiver. Our SoC is a synchronous digital design, using a 90MHz domain for the filters' datapath and a 160MHz domain for the processor and the rest of the blocks. According to the specification, the logic synthesizer was able to synthesize the appropriate clock trees and reset circuitry.

4 Conclusions

In this work we demonstrated how a popular and freely available language like Python, can be used as a unified environment/platform to describe a SoC in a high abstraction level, verify it and deliver RTL FPGA-synthesizable code. Across all design and simulation steps in our flow we use Python structures/syntax, and not any custom-defined syntax, to describe the datapath of the SoC and the related simulation models. This is very important since the main target group of a high-level design tool are engineers and scientists who have little or no experience at all in digital hardware design. The goal in this case is to deliver a tool where a high-level interface can be used to:

- Design a SoC in a block-oriented way, using IP cores in RTL or netlist format and apply minimum effort to include any required digital glue logic between the blocks.
- Support a high-level verification flow, where Python descriptions can be used along with Matlab-like or C descriptions to simulate a digital block in a functional/algorithmic level and also in a cycle-accurate Register-Transfer-Level.
- Ease the use of digital synthesis and physical implementation tools for FPGAs, by auto-generating synthesis and compilation scripts.
- Provide tools to interface a SoC design after its implementation, in the form of software components running in parallel on the processor-core in the SoC and in the host PC connected to the SoC.

All four items in the previous list are critical in modern SoC designs. We believe that SysPy comes as an integrated environment and utilizes Python best programming practices like object oriented programming, text processing features, associative lists and ready-to-use numerical libraries, in order to design, verify, implement and test a processor-centric SoC. SysPy supports the most common and basic Python syntax and also any third-party tool or file format used or called within SysPy is adopted by the EDA industry and the software community tools, like Tcl, VCD, Linux OS, SciPy and gcc compiler. In this way our tool was implemented on top of already existing, popular and standard tools used in a hw/sw co-design flow and we do not introduce any new, custom defined and "exotic" practices that would be valid only in SysPy.

To get feedback from the community we provide SysPy as an open source tool through a public code repository [15]. Many code examples are provided along with information on how to setup the tool. This work has been supported by the Greek State Scholarships Foundation (IKY) under grant 2008-5530.

References

1. P. Haglund, O. Mencer, W. Luk, I., B. Tai.: PyHDL: Hardware Scripting with Python. In: 13th International Conference on Field Programmable Logic (FPL), pp. 1040–1043. IEEE Press, New York (2003)
2. Mashtizadeh, A.: PHDL: A Python hardware design framework. ECE Dept. MIT, (2007)
3. Decaluwe, J.: MyHDL: a Python-Based Hardware Description Language. Linux Journal. 127, 5–9 (2004)
4. Lockhart, D., Zibrat, G., Batten, C.: PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research. In: 47th IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 280–292. IEEE Press, New York (2014)
5. Logaras, E., Hazapis, O.G., Manolakos, E.S.: Python to Accelerate Embedded SoC Design: A Case Study for Systems Biology. ACM Transactions on Embedded Computing Systems (TECS). vol. 13, no. 4, 84:1–84:25 (2014)
6. Berman, V.: Standards: the P1685 IP-XACT IP metadata standard. Design & Test of Computers. vol. 23, no. 4, 316–317 (2006)
7. Li, C., Donizelli, M., Rodriguez, N., Dharuri, H. Endler, L., Chelliah, V., Li, L., He, E., Henry, A. Stefan, M. and others: BioModels Database: An enhanced, curated and annotated resource for published quantitative kinetic models. BMC systems biology. vol. 4, no. 1, p. 92 (2010)
8. Breitling, R., Donaldson, R., Gilbert, D., Heiner, M.: Biomodel engineering–from structure to behavior. Transactions on Computational Systems Biology XII., 1–12 (2010)
9. Hazapis, O.G., Manolakos, E.S.: Scalable FRM-SSA SOC Design for the simulation of networks with thousands of biochemical reactions in real time. In: 21st International Conference on Field Programmable Logic (FPL), pp. 459–463. IEEE Press, New York (2011)
10. Gillespie, D.T.: Stochastic Simulation of Chemical Kinetics. Annu. Rev. Phys. Chem. vol. 58, pp. 33–55 (2007)
11. Pineda-Krch, M.: GillespieSSA: Implementing the Gillespie Stochastic Simulation Algorithm in R. Journal of Statistical Software. vol. 25, no. 12, 1–18 (2008)
12. Logaras, E., Koutsouradis, E., Manolakos, E.S.: Python facilitates the rapid prototyping and hw/sw verification of processor centric SoCs for FPGAs. In: IEEE International Symposium on Circuits and Systems (ISCAS), accepted for lecture presentation. IEEE Press, New York (2016)
13. Logaras, E., Manolakos, E.S.: SysPy: using Python for processor-centric SoC design. In: 17th IEEE International Conference on Electronics, Circuits and Systems (ICECS), pp. 762–765. IEEE Press, New York (2010)
14. Hazapis, O.G., Logaras, E., Manolakos, E.S.: A soft IP core generating SoCs for the efficient stochastic simulation of large Biomolecular Networks using FPGAs. In: 19th IEEE International Conference on Electronics, Circuits and Systems (ICECS), pp. 77–80. IEEE Press, New York (2012)
15. SysPy Git code repository, `https://github.com/evlog/SysPy`