Indonesian Journal of Electrical Engineering and Informatics (IJEEI)
Vol. 9, No. 4, December 2021, pp. 811~826
ISSN: 2089-3272, DOI: 10.52549/ijeei.v9i4.3243

The review of heterogeneous design frameworks/Platforms for digital systems embedded in FPGAs, and SoCs

Abdelhakim Alali[1], Hasna Elmaaradi [2], Mohammed Khaldoun[3] and Mohamed SADIK[4]
[1] Faculty of Sciences Ben M'Sick, Hassan II University of Casablanca, Casablanca, Morocco;
abdelhakim.alali@univh2c.ma
[2,3,4] National High School for Electricity and Mechanics, Hassan II University of Casablanca, Casablanca, Morocco.

Article Info

Article history:

Received Jul 31, 2021
Revised Oct 24, 2021
Accepted Nov 23, 2021

Keywords:

Open source
SoC
FPGA
Python
HW/SW codesign

ABSTRACT

Systems-on-a-chip integrates specialized modules to provide well-defined functionality. To guarantee its efficiency, designers are careful to choose high-level electronic components. In particular, FPGAs (field-programmable gate array) have demonstrated their ability to meet the requirements of emerging technology. However, traditional design methods cannot keep up with the speed and efficiency imposed by the embedded systems industry, so several frameworks have been developed to simplify the design process of an electronic system, from its modeling to its physical implementation. This paper illustrates some of them and presents a comparative study between them. Indeed, we have selected design methods of SoC (ESP4ML and HLS4ML, OpenESP, LiteX, RubyRTL, PyMTL, SysPy, PyRTL, DSSoC) and NoC networks on OCN chip (PyOCN) and in general on FPGA (PRGA, OpenFPGA, AnyHLS, PYNQ, and PyLog).

The objective of this article is to analyze each tool at several levels and to discuss the benefit of each in the scientific community. We will analyze several aspects constituting the architecture and the structure of the platforms to make a comparative study of the hardware and software design flows of digital systems.

Corresponding Author:

Hasna Elmaaradi,
National High School for Electricity and Mechanics,
Hassan II University of Casablanca, Casablanca, Morocco;
Hasna.elmaaradi.doc20@ensem.ac.ma

1. INTRODUCTION

The automation of the design flow of electronic systems began in the eighties with the appearance of coding in VHDL and Verilog. EDA (electronic design automation) takes advantage of the different levels of abstraction offered by these two languages to make the engineers' work easier and more flexible. Over the years and the evolution of technology, new challenges and challenges provoke the birth of new HDLs, like SystemC and DSL (Domain Specific Languages). These new generations of co-design tools exploit so-called high-level languages like Scala, C, and Python. By exploiting their resources in terms of libraries and the characteristic of simplified handling, the design flow of electronic components has seen a great leap forward that has marked state-of-the-art technology and contributed to the realization of intelligent systems.

These tools not only improve the design methodology but also contributed to the evolution of the applications and algorithms to be implemented. Indeed, thanks to the exploitation of the parallelism of FPGAs, circuits can make huge calculations and process mega data. The design flow of quite complex systems such as ASICs, and even the reconfiguration and reprogramming of FPGAs, also benefits from the evolution of computer science, especially development and programming. Certainly, many tools are based on high-level computer languages such as Python. The DSLs that we present in this, paper illustrate this

combination, which provides them with the required reliability, flexibility, and performance efficiency. To meet these needs and to make hardware design easy at a high level of abstraction, open-source hardware platforms and frameworks are being developed. We will highlight the common and distinct points of each, by analyzing their architectures and structure. In addition, we present the achievements and experiments demonstrating their effectiveness.

The present work is the first one that reveals such a study. Indeed, there are no similar reviews, hence the constraints of finding references in scientific articles and bibliographic literature. However, there is scientific research, such as an article [63] (published on August 25, 2021) that discusses only SoCs dedicated to medical applications that use the Internet of Things. It discusses current achievements in patient identification and analysis, particularly methods based on the so-called ECG, or electroencephalogram (EEG). The author highlights the combination of the optical aspects in the design of SoCs (HNoCs hybrid network-on-chip and ONoCs optical network on chip) in order to increase the different optimization parameters of throughput, latency, and power/energy.

In the following section, we will present the SLR (Systematic Literature Review) approach as a working method. Then, we will briefly describe each platform, LiteX, RubyRTL, PyMTL, PyOCN, PyRTL, SysPy, OpenESP, PYNQ, PRGA, HSL4ML, ESP4ML, PyLog, OpenFPGA, AnyHLS, and DS3. In the fourth section, a comparative study of these tools is detailed. Section 5 is a discussion and finally, we conclude this work.

2. WORKING METHOD: THE SLR APPROACH

The literature review presented in this paper is structured according to the systematic review method. The figure below illustrates the approach followed in preparing this article.

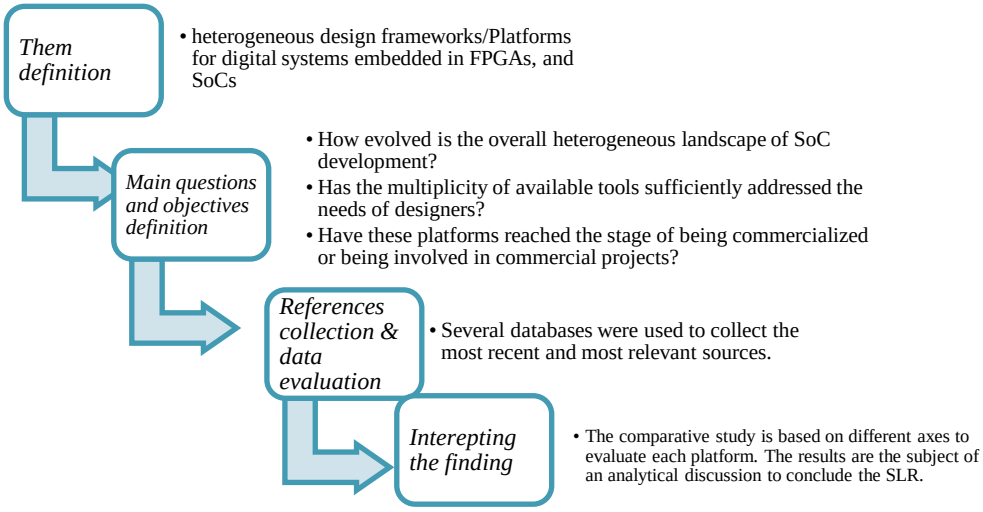

Figure 1. Working method description

3. AN OVERVIEW OF OPEN-SOURCE INTEGRATED CIRCUITS CO-DESIGN

The study concerns many heterogeneous codesign processes of digital systems, which we classified into two categories:

- ✓ Systems on Chip and their derivatives such as MPSOC (Multiprocessor SoCs), DSSoC (Domain Specific SoC), SoC centric and, NOC (Network on Chip).
- ✓ Field programmable gate arrays FPGAs Below is a definition of the main lines of the design flow of each.

3.1 Computing platforms for Building SoCs

3.1.1 LiteX

LiteX [1] [2] is a design tool for SoCs and FPGAs, using libraries developed by Migen/MiSoC. It provides blocks (IP) ready to be used in a circuit. It integrates different infrastructures that the user needs, to define his project. All the elements necessary to create a SoC are taken care of by LiteX, from modeling by a

simple script written in Python, to simulation and synthesis. As a result, the design process becomes easier and does not require another compiler or HDL generator.

3.1.2 RubyRTL

RubyRTL [3], as its name indicates, through the Ruby language (web development language), generates the Register Transfer Language RTL. It is based on the MigenPython toolkit as well as MyHDL, to establish the internal embedded DSL that will lead to the construction of the electronic components.

3.1.3 PyMTL

PyMTL [4] [5]presents a modular methodology for the design of integrated circuits. In fact, by exploiting the Multi-Level Modeling MTL, different phases of abstractions are established independently. PyMTL is a result of the work done in 2014 [6], which proved the importance of HGSF (hardware Generation and Simulation Framework) based on Python, to increase the performance of such a tool.

3.1.4 PYOCN

PYOCN (or PyMTL3-net) [7] [8] is a more complete version of PyMTL and aims at modeling, evaluating, and testing OCNs. It integrates the libraries necessary for the realization of this type of system, which involves the combination of several tools and processes. This Framework provides a synthesizable RTL while ensuring performance in terms of energy, time, and space.

Its principle is to build interconnection networks vertically to optimize the design space. The modular architecture adopted in this platform allows modeling, testing, and simulation in different levels of abstraction.

3.1.5 PyRTL

PyRTL [9] is a tool for describing embedded digital hardware. It exploits the advantages of Python, the high-level language. It does not contain non-synthesizable hardware primitives. In addition, it allows the inclusion of instrumentations tools in the structure to be designed. PyRTL provides a very important feature that was not possible with traditional design methods.

The intermediate representation of PyRTL provides the various tools necessary for simplification, implementation, and efficiency in the modelling of complex digital systems. Indeed, the intermediate representation allows for the creation of an instrumentation platform and to transform better handling. To create these tools, several easy-to-use APIs allow modifying the hardware blocks and providing the necessary information. Binary instrumentation for embedded hardware design is a method developed by PyRTL.

3.1.6 System Python: SysPy

SysPy [10] (released in 2012) is one of the high-level co-design approaches. It provides a platform that allows the synthesis of hardware behaviors, and design SoC centric:

- ✓ Incorporate scripts developed in Python, with existing hardware blocks.
- ✓ Incorporate IP cores into a circuit and ensure the connection between them.

In addition, it performs the main tasks in the design of SoCs and MPSOCs.

- ✓ The hardware description of all components is attached to the processor.
- ✓ Incorporation of the modules into a synthesized design.
- ✓ Establishment of software development scripts for the core processor.

SysPy combines two essential processes for SoC design. Abstract hardware description and FPGA prototyping to synthesize the circuit. It supports the following processor cores: OpenRisc 32bit, AVR ATmega128 8bit, and Leon 3.

3.1.7 OpenESP

OpenESP [11] is an open-source, tile-based platform for implementing SoCs. It is intended for applications that develop and enhance accelerators. It also comes for the deployment of Network of Chips (NOC) and interconnecting their processors and accelerators. Its high-level modular architecture allows for a high degree of abstraction in the heterogeneous chip design flow. It allows automated agile system synthesis, expansion, and flexibility when compared to other SoC co-design tools.

OpenESP integrates RISC-V core and NVIDIA Deep-Learning Accelerator. It encompasses several hardware design methodologies such as Vivado HLS, Catapult HLS, Stratus HLS, HLS4ML, and RTL.

The ESP SoC architecture is a matrix of tiles of different types (processor tile, accelerator tile, memory tile, auxiliary tile). The connection between these different elements is ensured by a multi-plan NOC (Network on Chip).

3.1.8 PYNQ

PYNQ [12] (Python Productivity for Zynq) is an application made by Xilinx to make it easier for programmers and designers to handle the Zynq and Zynq Ultrascale + MPSoC (Multiprocessor SoC) + RFSoC [13] (Radiofrequency SoC). It is the first project that used Python as a high-level description language. In addition, it provides a resource-rich library that the design engineer needs. This library hardware and joint JUPYTER application and web architecture strengthen the structure of the PYNQ Framework. Indeed, it has been exploited to realize different projects in different fields.

3.1.9 DS3

DS3 [14] (system-level domain-specific SoC simulation) is a system-level domain-specific SoC simulation platform. It leverages Dynamic Power and Thermal Management (DTPM) methodologies to provide users with a tool for resource optimization including the SoC design space.

DS3 has implemented wireless and radar communication benchmark processes. It also allows the implementation of heuristic and tabular planning algorithms. A solution aims at improving the energy performance of DSSoCs.

3.1.10 HLS4ML

It is an electronic system-building platform for implementing machine learning algorithms on FPGAs and ASICs. HLS4ML [15] allows the design of low-power components and accelerates decision-making by ensuring the translation and processing of algorithms in real-time.

3.1.11 ESP4ML

ESP4ML [16], is a whole design process for SoCs used in the field of machine learning ML and signal processing. It aims at the hardware acceleration of ML, taking advantage of the work done and the results obtained in HLS4ML and ESP (Embedded Scalable Platforms).

However, it has been necessary to adapt these two systems with the goals underlined for the ESP4ML framework. Indeed, an HW/SW layer is developed to adapt the architecture of the accelerators, in addition, the interface libraries to successfully implement the algorithms realized with HLS4ML in the SoCs.

3.2 Prototyping FPGA's frameworks

3.2.1 Princeton Reconfigurable Gate Array (PRGA)

PRGA [17] is a manufacturer of custom FPGAs and manipulates them in other systems as ASIC. Through Computer-Aided Design tools, it allows the synthesis of the generated RTL. It converted first in a Bitstream series by the PRGA Tool Chain part. The separation between the manufacturer and the toolchain gives the characteristic of modularity and flexibility. Indeed, the user can intervene to modify his system at any phase of the design.

3.2.2 PyLog

PyLog [18] is a focused algorithm that aims to simplify and optimize FPGA configuration. This scalable tool converts the script written in Python, into a synthesizable code via compilers and different passes, which completes the role of intermediate presentations PyLog IR generation and optimization.

3.2.3 AnyHLS

AnyHLS [19] has come to solve the problem that programmers have with hardware description whether it be by VHDL, Verilog, or System Verilog. It requires knowledge of FPGAs. This approach allows the design, by generating HLS synthesizable code.

It takes advantage of the already proven work in AnyDSL [20], as a high design compiler, using the Impala language and partial evaluation. AnyHLS has developed its library in the field of image processing.

3.2.4 OpenFPGA

OpenFPGA [21] is an approach for the design of custom FPGAs. This new methodology for building these components. It reduces the time needed to build their architectures, and even reconfigure them for a specific application.

OpenFPGA provides not only the ready-to-use Verilog coding and manufacturing (production flow) but also the Bitstream structure (end-user flow), which describes the designed FPGA in XML language, for digital use.

4. Comparison study

This section aims to develop a study and analysis of the Frameworks/platforms presented previously. We will dig into their architectures, and their design flows to identify the common and distinct points and the advantages of one over the others. Furthermore, we evaluate each platform, via enumerating hardware achievements, and completed projects. Table 1 below, samurize the highlighted lines of the comparative study.

Table 1. table summarizing the comparative study

Platform/ Criteria	HDL	Synthesis Language	ToolBox	Simulation/ Compilation
RubyRTL	Ruby+ Python	VHDL	Migen+Sexpir Tools	GHDL
AnyHLS	Impala	Altera SDK for OpenCL (AOCL) + Vivado HLS	Created ones Libraries	AnyDSL
ESP4ML	Generic Kernel	Vivado HLS	Runtime system	Vivado Simulator
OpenFPGA	XML	Bitstream+ YOSYS	FPGA-Verilog, FPGA-SPICE and FPGA-Bistream[64]	HDL Simulator
LiteX	Python	Verilog	Migen+MiSoC	LiteXSim
PyMTL		SystemVerilog	Python/C tools	SimJit
PYOCN		Verilog	Python/C tools	PYOCN Simulator
PyRTL		YOSYS	Computer-Aided Design (CAD)	PyRTL Simulator
SysPy		VHDL	Python/C tools	GCC Compiler
OpenESP		VHDL+ SystemVerilog	CAD tools	Vivado Simulator
PYNQ		Python HLS	Python/C tools	PYNQ Emulator
DS3		-------	Job/task generator + Scheduling&DTPM	Kernel Simulator
HLS4MTL		Vivado HLS + C++	Keras+Pytorch	Vivado Simulator
PyLog		Python HLS	Created ones Librairies	Merlin Compiler
PRGA		Bitstream+ YOSYS	CAD	SynoPsys VCS

4.1 HDL describes the structure of the design flow

Most of the design processes presented are based on Python as illustrated in Figure 2, a high-level programming language.

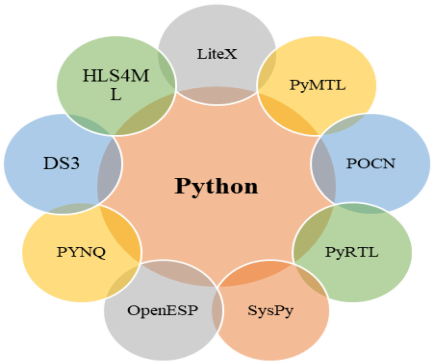

Figure 2. Python as an HDL for description behaviours of digitals circuits

OpenESP indirectly uses Python. It supports PyTorch as a language for designing SoCs. Furthermore, ESP can also be programmed with C/C++, SystemC, Keras Tensor Flow, ONNX, Chisel, SystemVerilog, and VHDL.

Also, HLS4ML [15] is programming with (Q) KERAS, PyTorch and, ONNX. KERAS [22] is an application-programming interface API, based on Python and uses the TensorFlow ML platform as a software interface. Open Neural Network Exchange ONNX [23] provides the developers with a graphical presentation of the modeling of the Machine Learning elements

Python allows both, the description of the behaviour of the system to be designed with a high level of abstraction and to model its hardware architecture using the tools offered by the components to be configured

i.e., FPGAs. Hence, the unification characteristic of the digital platforms of the design flow of embedded systems is endowed.

On the other hand, the programming structure in Python facilitates the work of designers. It provides them with the different tools (functions and commands) they need, and with a syntax that is easy to learn and exploit.

While RubyRTL [3] (Ruby language), AnyHLS [19] (Impala), ESP4ML [16] (Generic Kernel), and OpenFPGA [21] (XML) make use of the features offered by other languages for hardware description.

Ruby as Python is a high-level language, contrary VHDL, Verilog, and SystemVerilog there allow an abstraction low level. Ruby is known as a web environment development language especially with the emergence of Ruby on rails (creation of web applications). It also offers many advantages for programmers from structuring to compilation.

Impala [24] is a programming language that provides as a result the intermediate representation of Thorin. It is based on the continuation-passing style (CPS), and the syntax of Rust. It allows the specific description of such a domain [25]; hence, it does not support standard resources. The advantage of this language is that it can be interfaced with other programming languages like C and C++ [25].

Kernel [26] shared with C many features. Indeed, it is developed with it. The main advantage of the programming language Kernel is the orthogonality of the constructor.

4.2 Synthesis language

As shown in Figure 3 below, apart from the traditional HLS that dominated the automation of the synthesis of electronic components, VHDL, Verilog, and SystemVerilog, there appeared another python-based coding for the physical implementation of an architecture, and digital presentation of description hardware.

Figure 3. Software or HLS generated by platforms

DS3 is a DSSoC simulation/emulation tool without hardware verification, so it does not include an HLS.

- ✓ Verilog, VHDL and, SystemVerilog

Even if they are classified as traditional tools, and long years have passed since their appearance, Verilog, VHDL, and SystemVerilog are still competitive with new languages developed with more sophisticated methodologies of advanced technology. They are still present in the toolchain of many platforms, such as LiteX, PYOCN, SysPy, RubyRTL, OpenESP and, PyMTL [1] [3] [4] [7] [10] [11]. Hence, contributing to the evolution of the automation of the HW/SW design of electronic systems.

- ✓ Synthetized C++

The integration of an ML system in an ASIC chip is more complex than in FPGA. HLS4ML uses C++ to compile the designed model and generates the RTL code that will be physically implemented. This process respects the triple to optimize Power, Performance, and Area PPA.

HLS4ML proceeds to a combination of C++ and RTL to guarantee the synthesis of the modeled architecture [15].

✓ Vivado HLS

Vivado HLS [15] [16] [19] is used by:

- HLS4ML for hardware implementation on Xilinx FPGA.
- ESP4ML relies on it to synthesis ML algorithms on programable systems on Chip.
- AnyHLS for Intel and Xilinx FPGAs.
- PyLog as HLS tools for compilation

Vivado HLS [27] [28] is the platform realized by Xilinx for adapting preliminary code (developed in a familiar C, C++, or SystemC language), and translating it into synthesizable code (VHDL, Verilog, or SystemC RTL). Two necessary steps are performed to properly carry out this task:

- Firstly, model the architecture according to a clock cycle count.
- Secondly, convert the program elements into logic ports.

Vivado HLS also provides tools for the verification and simulation of the generated RTL model, in order to have a first validation of the elaborated system.

✓ AOCL

AOCL (Altera SDK for OpenCL) [19] is leveraged by AnyHLS to ensure the portability of Intel FPGAs. OpenCL [29] is a platform developed in C and C++ that aims at the high-level synthesis of models with more speed while decreasing the Time to Market. AOCL completes the work of OpenCL by ensuring the verification and prototyping of circuits for a good imitation of the real system.

✓ Python HLS

the intermediate representation of Pylog [18] (Pylog IR) allows for the generation of the HLS C code that can be synthesized by the compiler. The latter uses the HLS pragmas to transform the source code written in Python, into an HLS exploitable in the RTL-based design flow.

PYNQ does not generate an HLS [30]. It is necessary to use an FPGA fabric. However, it provides a PNQ Overlays platform which ensures the interfacing of the application developed in Python with the physical inputs and outputs.

These hardware libraries [31] allow the overlay and customization of physical circuits. In addition to API, PYNQ Python provides a tool for the control and configuration of overlays. So, designers are not required to create these libraries, but just write lines in Python to operate the management interface.

✓ Bitstream

OpenFPGA [21] offers hardware engineers the choice of FPGA binary configuration, via FPGA-Bitstream. This feature is only applicable for FPGA architectures supported by Versatile Place and Route VPR [32] (an open-source builder of CAD algorithms and structures for FPGA) whose role is to be a packaging engine [33].

Similarly, PRGA [17] uses the tool named PRGA Builder to generate the .xml extension files that will be processed by VPR to provide at the end of the bitstream.

✓ Yosys

PyRTL [9], PRGA [17], OpenFPGA [21] use an external tool to accomplish the task of establishing the synthesizable RTL model. Yosys [34] is an open source (ISC license) intended for generating synthesizable code via Verilog and implementing it on FPGAs or ASICs. It can be used for Xilinx 7 series, Intel, Lattice iCE40, Lattice ECP5, Silego GreenPAK4, Gowin, and Analogic. Yosys provides a code converter [35] generated in BLIF / EDIF / BTOR / SMT-LIB / simple RTL Verilog / etc. Its use in different projects is due to the synthesis methodologies offered for various applications.

4.3 Tools and libraries

The platforms that are the subject of this study differ in the methodologies and procedures employed in their process of building or simulating/emulating digital systems as shown in Figure 4 below.

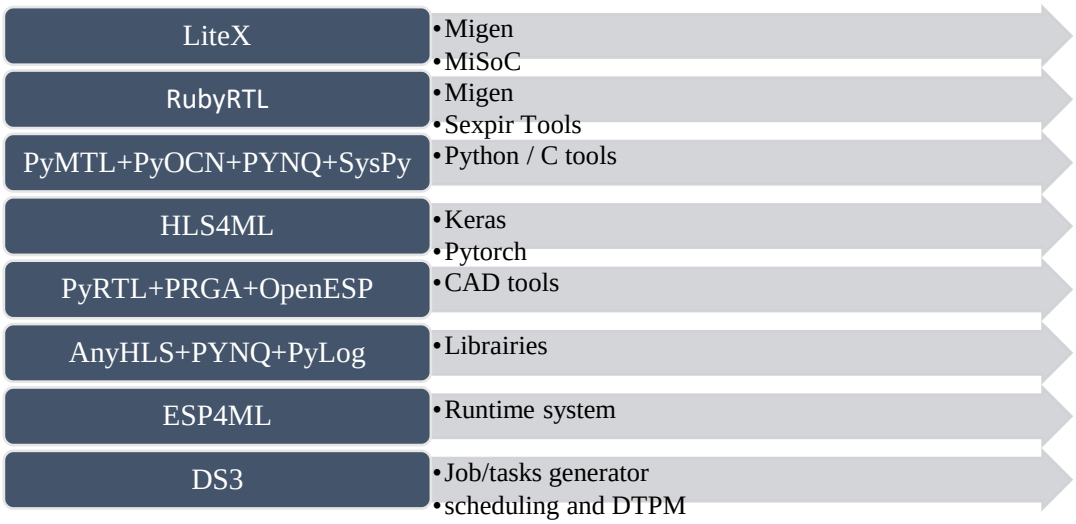

Figure 4. platform's Tools

✓ Migen toolbox

LiteX [1] and RubyRTL [3] use the libraries available in the Migen toolbox. Taking advantage of the benefits and features of the Python programming language, it provides a reliable procedure for designing complex electronic/computer systems [36]. Migen comes to fill the gaps left by software design based on traditional languages (VHDL and Verilog). Migen also uses the FHDL (Fragmented Hardware Description Language) module [37]. It provides a behavioral description with code written in Python so that it differentiates between combinatorial instructions, synchronous instructions, and reset values.

In 2018, an enhanced version of Migen was launched, nMigen [38]. It takes care of all the modeling that Migen supports. In addition, new libraries are integrated. Now it provides communication via I2C and SPI protocol, interconnection with configuration and state register (CSR), and compilation of Rust code on Minerva RISC-V.

✓ MiSoC toolbox:

To be able to synthesize systems on Chip, LiteX uses the MiSoC tool [36]. It is a complementary/extended solution to Migen, to provide the designer with an efficient and easy methodology for building SoCs. it integrates different processors, memory modules, additional peripherals. In addition, it allows resource optimization with high performance.

✓ Sexpir tools:

Sexpir [3] is an intermediate representation developed by the team that designed RubyRTL, but work is in progress for final validation. It is a bidirectional transformation of RTL code from Python (via Migen) to Ruby (via RubyRTL) [39]. It also allows the generation of VHDL code for physical implementation on FPGAs and ASICs.

✓ KERAS

The HLS4ML library [40] relies on two main tools [15], Keras and Pytorch which are developed in the following part. [22] This specific API for deep learning offers speed in obtaining results. It is executable on the end-to-end interface named TensorFlow. It allows to development of solutions for deep learning via different abstractions and the export of graphs to other tools. As a result, [41] the deployment of ML applications becomes easier. TensorFlow aims at accelerating the implementation of neural networks.

✓ PyTorch

Pytorch [XMR like NumPy, SciPy, and Cython. It is portable by Linux, macOS, and Windows operating systems. Pytorch exploits the robustness of GPUs and CPUs for its six main components (torch, torch. autograd, torch.jit, torch.nn, torch.multiprocessing, torch.utils).

✓ Toolbox based on Python / C

The process of PyMTL [6] consists of manipulating the instance model elaborated according to the specifications and configuration described by the programmer, by simulation, translation, and user tools to provide in the end a model that is consistent and faithful to the behavior of the physical circuit.

The translation tools provide the Verilog code after processing PyRTL, analyzing its structure, and its connections. The PYOCN EDA script [7] for use in the EDA flow exploits this result. Then, the simulation tools, analyze the ports and logical blocks, to provide the operator with a powerful python simulator. Finally, the user tools are the custom methodologies created by the operator himself to ensure the scalability and flexibility of the Framework.

In 2020, researchers from Cornell University have released a recent version of PyMTL. Indeed, PyMTL3 [4] comes with new passes that improve the quality and performance of the models developed in the early design phases. The following table 2 illustrates these new features:

Table 2. the different passes integrated into the PyMTL3 toolchain

Analysis Passes	Instrumentation Passes	Transform Passes
Linting: for code verification.	**Simulation:** Simulation of the whole model per cycle.	**Import:** integration of external IP or creation of new components.
Statistics: present the main lines of the design.	**Tracing:** offers the different possible tracings for execution and debugging.	**Prototype Proxy:** provides the wrapped prototype for evaluation testing.
Pre-synthesis: finalize the generated RTL by detecting and correcting errors	**Translation:** traduction of RTL to HDL for the FPGA and ASIC	**Ad-Hoc Transform:** Optimization and simplification of the updates to be made on the code.

As already mentioned, PYOCN [7] is based on PyMTL, being its modular architecture (modeling, testing, and evaluation). The same tools are used in the OCN construction process. Additional tools were needed for modeling the networks. Certainly, libraries specific to this type of chip have been integrated (InputUnit, SwitchUnit, RouteUnit, OutputUnit, Router, Channel, and Network).

The SysPy platform also includes methods written in Python for hardware description. [44] In its structure, we find a synthesizable Python to VHDL converter. Besides a GCC (GNU Compiler Collection) compiler that takes care of the management and control of the processors, there is also an XST synthesizer of the generated code.

Furthermore, SysPy allows synthesizing the CoreLib components describing the SoCs via netlist files, which will be input for the converter.

SysPy provides a communication interface between the SoCs, and the hot computer [10], through a HAL hardware abstraction layer. This gives access to the FPGA communication channels (GPIO, Ethernet), and the RAM.

An interface [10] has been realized, to communicate either with other SoCs or with a host computer. This window allows the controlling and exchange of data via communication protocols. Indeed, there is a channel for intra-chip communication and another for external systems. Then a designer can launch operations and treatments, as a command to be executed by the SoC processor or its peripherals. These features are due to the creation of a hardware abstraction layer HAL and software API. Python is always present in this design. The API is developed with Python classes and functions, which will allow the control of the designed SoC, via transmission and reception frames and data storage in memory.

PYNQ takes advantage of the package structure of Python. The PYNQ Python package [XMR, pynq.bitstream, pynq.devicetree) or those for data manipulation (pynq.mmio, pynq.gpio, pynq.xlnk, pynq.buffer). There are also additional modules and sub-packages to complete the toolbox necessary for the design of the ZyNQ range (pynq.interrupt, pynq.pmbus, pynq.uio, pynq.registers, pynq.utils, pynq.lib, pynq.pl_server).

- ✓ CAD tools

PyRTL [9] provides a very important feature that was not possible with traditional design methods. This is introspection, which allows the hardware behaviour or design pattern to be copied, using pipelines and the next_stage () function. The intermediate representation of PyRTL provides the various tools necessary for simplification, implementation, and efficiency in the modelling of complex digital systems. The logical operations create WireVectors with a bit number as input.

Binary instrumentation for embedded hardware design is a method developed by PyRTL. Indeed, the intermediate representation allows for the creation of an instrumentation platform and to transform better handling. To create these tools, several easy-to-use APIs allow to modify the hardware blocks and provide the necessary information.

The wires used in the circuit do not store the information of the network they belong to. So, to remedy this problem, net connections have been developed. This function establishes a dictionary that gathers all the

information about the wires. That is the source network and the user network. Hence the flexibility in the transformation of the circuits for the instruments.

Circuit changes are handled by two functions: wire_transform() and net_transform(). They support modification of the hardware design, either by adding additional instrumentation or just replacing a logic block. For the transformation to be effective it needs a well-ordered data flow that respects the iterator. This functionality is presented as a default iterator for a block. There are two types of transformation:

i) Operation transformations: The function has an argument, a logical block, and a procedure that will be called on each LogicNet (the tool that stores the information related to a logical block: network and file) of the system. The result it provides is in the form of a Boolean, which will allow the operation transformer to know if the original LogicNet should be kept in the block.

ii) Connection transformations: This time things are more complicated. The connection transformation function uses as input a WireVector which is itself a network, and returns two WireVectors, source, and destination, which correspond to a "slot" connection. So, this transformation is just another function that defines the wire information.

On the other hand, the operation of the PRGA platform requires the application of certain CAD tools [17] :

- an RTL code synthesizer
- port packer
- configurable logic blocks (CLB) locator
- bitstream generator

The production flow includes the different methodical elements that the OpenFPGA framework exploits for the prototyping of custom FPGAs [33]. First, we have the SDC (Synopsys design constraint) which provides us with STA Tools necessary for time management and validation [33]. In addition, with the contribution of Fabric Verilog, SDC allows us to establish GDSII layout via Backend Tool. The HDL Simulator and Format Tool check the format and the functional aspect of the Verilog Testbench model.

✓ Tile's structure

The ESP SoC [11] architecture is a matrix of tiles of different types:

- processor tile: Each tile includes a selected processor that is system independent, and it communicates on a local bus.
- accelerator tile: This tile follows a well-defined architecture. The sockets available on the platform make it easy to create accelerators by exploiting registers, addressing, and protocol.
- memory tile: The memory tile includes an LLC (last-level cache) partition that implements the MESI protocol.
- auxiliary tile: It gathers all the peripherals that complement the work of the processor. Its socket is the most complex, as it manages several services. The tile ensures the communication between its masters (Ethernet), and the slave devices.

The connection between these different elements is ensured by a multi-plan NOC (Network on Chip). The transparent communication layer uses multiple ports to exchange data. They include standard bus ports, bridges, interface adapters, and proxy components. The bus masters are the accelerators and processors, while the slave devices are any other system components such as memories, UART, and Ethernet.

The Application Programming Interface (API) of the ESP platform is a method of managing and executing accelerators from a user application using three programming functions. ESP_run for execution, ESP_alloc, and ESP_free for memory allocation.

For existing accelerators, ESP provides a third-party design flow (TPF) to integrate directly into the SoC. Simply fill in an XML file and provide the source file either coded in Verilog, VHDL, or SystemVerilog.

✓ Libraries

AnyHLS [19] uses AnyDSL as a library that the user uses in his configuration of the system architecture to be designed. It is a compiler based on the THORIN intermediate representation based on continuation-passing style (CPS IR Thorin) and partial online evaluation [20].

AnyHLS [19] also integrates a specific library for image processing. Indeed, the exploitation of the impala language and the architecture of this high programming tool allows generating all the abstractions (in the form of libraries) that the user needs to build his circuit. AnyHLS provides a whole set of tools like:

- The make_local_op generator that ensures decoupling of the algorithm from the scheduling and memory operators

- Optimization functions like REDUCTION
- FACTORIZATION methodology for parallel processing of input pixels.

PYNQ includes libraries that ensure the flexibility of this platform [46]:

- An API that handles IP hardware addresses (Audio, AxiGPIO, AxiIIC, DMA, Logictools, Video).
- Libraries for communication with external devices (Arduino, Grove, Pmod, RPi).
- Libraries for loading or creating applications via Jupyter (MicroBlaze Subsystem, Microblaze RPC, Microblaze Library).
- Libraries for the management of Processing System PS/ Programmable logic PL interfaces (Allocate, Interrupt, MMIO, PS GPIO).
- Libraries for the control of PS/PL interfaces (PMBus, Overlay Device and Bitstream classes, Microblaze Library).
- Hardware libraries [31] that aim to accelerate and/or personalize a digital application. it is simply the construction of programmable logic or a new configuration using a python interface [47].

PyLog [18] incorporates the existing HLS C library as well as additional features. Indeed, the functions offered by NumPy are available on the PyLog platform.

✓ Runtime System:

ESP4ML [16] integrates a software tool for gas pedals. This API allows the configuration of data traffic on NOCs, as well as pipeline elements

✓ Job/Tasks generator:

the task generator [15] used by DS3 allows the establishment of DAG (Directed acyclic graph) specific to a given application.

✓ Scheduling and DTPM Algorithms:

DS3 [14] includes Dynamic Power and Thermal Management policies and scheduling algorithms, which will allow designers to model and implement new algorithmic procedures. It is thanks to the Scheduler class and the constructor functions that the user creates his algorithm.

4.4 Compilation and simulation

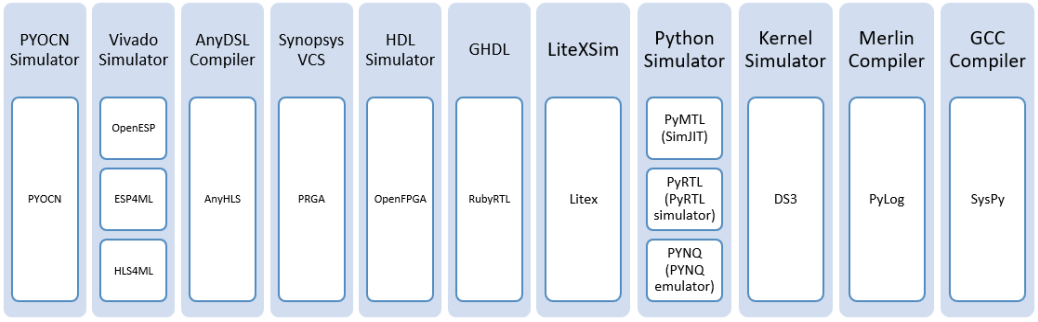

Figure 5. Simulators of each framework

The platforms that are the object of this study have the same intentions of improving the performance of FPGA design whatever the field of application. Hence, the functionality of the simulator used in the design chain determines and evaluates the approach in the first place. Figure 5 above summarizes the different simulators used for each framework.

✓ GHDL

RubyRTL does not include its simulator; however, the results generated by RubyRTL can be simulated on GHDL [3]. It [48] is a VHDL code simulation tool that uses code generators such as LLVM and GCC to ensure a fast simulation cycle.

✓ LiteXSim

The Verilator simulator (released in 1994) inspires LiteXSim SoC Simulator [1]. [49] [50] this one allows converting a synthesizable RTL code developed in Verilog or SystemVerilog, into a script written in C++ or SystemC (.cpp and .h files). The Verilator compiler is characterized by its high performance. The speed,

optimization, and partitioning of the RTL model allow them to outperform traditional compilers (for example, it is 100 times faster than Icarus Verilog).

- ✓ Python simulator

PyMTL [6] [5] unifies the advantages of python via PyPy System and those of HGSF (Hardwar Generation and Simulation Framework) to provide SimJIT (Just in time), a simulation approach that aims at optimizing RTL and CL (cycle level) modeling. The software architecture of SimJIT consists of converting the input code (either RTL or CL) into a wrapped C++ model, using the intermediate representation and the toolbox involved. The simulation passes for the improvement of the established PyMTL model [4] to enhance the performance of this tool.

In the same sense, the PyRTL simulator [9] synchronously interprets the initial C program, so that the states and calculations are returned to the user simultaneously. The simulator has approved its performance not only in terms of emulation cycle time but also in terms of the size of the code to be written.

PYNQ [51] also integrated a Python library for emulation of Xilinx Zynq systems generated by the PYNQ environment. Furthermore, the simulation is available on Vitis [52] (see next paragraph).

- ✓ Kernel simulation

DS3 is a compiler/simulator based on the Kernel language as mentioned above. This simulator [15] allows the implementation of scheduling algorithms and machine synchronization. Thanks to the Kernel simulator [53] structure, it allows visualizing the progress and the future state of the simulated model.

- ✓ Merlin compiler

PyLog compiler [18] is the combination of two compilers Vivado HLS (see the previous section) and Merlin compiler. [54] It is a solution dedicated to hardware acceleration especially for software developers who do not master traditional HLS (VHDL and Verilog). It facilitates computation and processing for cognitive applications such as machine learning and big data. The Merlin compiler from Flacon computing [55] (acquired by Xilinx) provides an executable model on FPGAs from a script written in C with all the design abstractions

- ✓ GCC Compiler

SysPy generates the synthesizable code from the script established by the GCC compiler [44]. The GNU Compiler Collection GCC [56] is an optimized approach developed with multiple programming languages and integrates several hardware architectures.

- ✓ PyOCN Simulator

In addition to PyMTL's optimized approach to model evaluation, PyOCN [7] integrates a simulation process suitable for the OCN target. PyOCN Simulator is used at the cyclic level of network design. It does not apply to synthesizable RTL.

- ✓ Vivado Simulator

HLS4ML is a main component in the structure of OpenESP and ESP4ML [16] [11] [16] . These three platforms provide a code whose simulation is ensured by Vivado Simulator, also we can add in this category OpenFPGA [33] that uses HDL simulator as a tool for verification and evaluation. [57] This general environment reduces the execution time by detecting errors on the object code of behavioral, functional, and temporal simulation. The simulation can be run at any level of abstraction to verify the generated model. [58] The two interfaces DPI (direct programming interface) and XSI (Xilinx proprietary interface) for interaction between the simulation kernel and C scripts or HLS enrich the simulation tools and the co-simulation for better performance.

- ✓ AnyDSL compiler

AnyHLS use AnyDSL for the compilation of image processing via partial evaluation [19]. The partial evaluator [20] works with CPS style intermediate representation-based structures to generate CPU and GPU specific vectorization and parallelization code. This compiler is dedicated to the fields of image processing, ray tracing, and genomics.

- ✓ Synopsys VCS

The experiments carried out on PRGA uses Synopsys VCS as a simulator [17]. It [59] is a simulation engine based on the Fine-Grained Parallelism FGP methodology. This has allowed it to raise the performance in terms of time and bug checking to provide a high-quality design. VCS [60] integrates different technologies

and tools (VCS Xprop for X propagation, VCS native low power NLP, PowerReplay, Certitude, Z01X fault simulation) to cope with the constraints imposed by the complexity of building embedded systems.

4.5 Evaluation, and experimentations

In the field of electronics and especially embedded systems, the physical implementation of the programmed models is an essential step to evaluate the reliability and efficiency of the approach followed for the design of a circuit. Hence the importance of this section, which represents the achievements of each application, either as an experimental verification or even as a contribution to a project with industrial or academic stakeholders. Table 3 below provides more details.

Table 3. The Achievements of each framework

Frameworks/ Platforms	Experimentations	Projects
LiteX Platform [1] [23]	-SoC synthesis using Migen + VivadoSoC -synthesis using Migen+ Trellis +Yosys + Nextpnr	10 open-source IPHDMI2USB Fupy NetV2 Axiom SDI module PCIe Screamer Fomu Betrusted USB3.0 PIPE Chubby75 4 softcores supported (LM32, VesRISCV, PicoRV32, Mor1kx)
RubyRTL [3]	Experimental toolchain for Sexpir using UART IP	----------
PyMTL [5]	-LTAaccelerator, -DAE, ParallelXL and XLOOPS -TSMC 16 -BRGTC1 and 2 Batten Research Group Test Chip 1, and 2	To be a teacher in two universities: Cornell University and Boston University.
PyRTL [9]	-Verification of simulation time -SoC integration using Xilinx PYNQ	----------
PYNQ [24]	Xilinx xc7z020clg400-1	----------
DS3 [15]	-Benchmarking with 6 reference applications -Reference design for each application	----------
PyLog [18]	-Accelerator performance evaluation -Supported FPGA platforms: ZedBoard, PYNQ, Ultra96, Amazon EC2 F1 instance, Alveo series	----------
SysPy [10] [25]	-Processor-centric SoC system to apply Sobel edge detection to a grayscale image. -SoC-centric for the implementation of Gillespie's FRM SSA algorithm.	----------
PyOCN [7]	Verified for different network topologies with 64-terminals	----------
OpenESP [11]	-DFS SoC: dynamic frequency scaling -MC SoC: Multi-core SoC -RISC-V based SoCs for deep learning -RISC-V based SoCs (NVDLA)	----------
HLS4ML[16]	Deploying Neural Networks in Xilinx Virtex UltraScale + VU9P FPGA	----------
ESP4ML[16]	Verified for SVHN Street View House Numbers	----------
AnyHLS [19]	Verified for Gaussian, Harris corner detector, Jacobi, Filter chain, bilateral filter, mean filter, and sobelLuma	----------
PRGA [17]	-BCD2BIN Converter	----------
OpenFPGA [22]	-20x20 homogeneous FPGA -32x32 heterogeneous FPGA	----------

Most (except RubyRTL) of the studied frameworks are experimentally verified. Indeed, each of them proceeded to applications even at a small size to evaluate the developed tool and measure its performances to see the level of its competitiveness compared to the current tools in the field. Thus, highlighting the prospects and future work to be developed or the weak points to be improved, to build a solid approach at the level of architecture, handling, evaluation, and synthesis.

LiteX and PyMTL succeeded in this step and collaborated with other participants interested in the evolution of hardware and software design of systems. LiteX together with Enjoy-Digital [62] [61] realized various projects and provided the industry with high-tech components. Moreover, PyMTL has entered the

academic cursus of two universities as an approach to be taught to students of different cycles. It constitutes a whole module with tutorials and practical works to validate for better learning of the tool.

5. DISCUSSION

In this article, we have tried to present a review of some methodologies for the heterogeneous design of electronic systems, namely FPGAs and SoCs. The different solutions are analyzed in several aspects.

First, the choice of the development language is linked to the objectives to be reached and also to the applications targeted by the manufacturer. The general-purpose platforms exploit the functionalities offered by Python, and Ruby as high-level languages, and yet the others dedicated to the domains of machine learning, image processing, and computer vision go to more specific and specialized languages in an activity like impala and Kernel.

Second, the generated synthesizable code. Each Framework incorporates a tool for generating or converting script initially written in a synthesizable RTL model. Some use traditional languages (VHDL, Verilog, and SystemVerilog) that have proven their reliability over the decades. Others have chosen more recent processes developed with techniques that are more advanced and procedures like Vivado and Yosys. In addition, some of them exploit object-oriented programming languages (Python and C++) to build their converters with additional options such as the digital synthesizer (BitStream).

Afterward, we dissected the toolbox of each approach. Again, the purpose of the work plays a role in the guideline to adopt. However, the libraries integrated into the packages used after LiteX, RubyRTL, AnyHLS, PYNQ, and PyLog are written in Python, and in addition, Pytorch and most of the computer-aided design tools are developed in Python. As a result, most platforms rely on Python to create a toolchain that is appropriate for their architectures. Obviously, due to the nature of the applications they were built for, it was necessary for HLS4ML, ESP4ML, and DS3 to add other additional mechanisms to complete their needs.

Subsequently, we have delved further into the structure of the studied Frameworks, discussing the methods of verification and evaluation of the functional or behavioral modeling. Except for LiteX, PYOCN, and PyMTL, all the procedures have integrated an existing simulator into their processes. Indeed, instead of building a new kernel, they benefited from the performance demonstrated by high-level simulation engines, for example, Synopsys VCS, Kernel simulator, GCC compiler, Merlin compiler, and Vivado simulator.

As far as PyMTL and PYOCN are concerned, we can say that the simulation between them is similar since the PYOCN simulator neither combines the passes of PyMTL and some libraries to make them suitable for running on-chip networks. However, LiteX takes advantage of Verilator to design and reconfigure a fast simulator of SoCs and IPs.

Finally, the most critical and definitive step in building a digital circuit is synthesis. It identifies the rational and pragmatic potential of such a platform. As we mentioned in the previous section, LiteX marks its strong presence in the industrial market. Not only one or two, but also several projects of highly sophisticated electronic components and boards have been established using this method. This shows the engineers' satisfaction with the features and benefits offered.

6. CONCLUSION

In this article we have illustrated the advancement of integrated circuits design flows, by presenting some platforms, most of which are recent (released in 2020, 2021). Each one has its methodology for the realization of a target component, but they share the concern of performance optimization. According to their application domains, each tool is privileged by one or several characteristics that distinguish it from the others.

The projects carried out and the experiments did show the maturity and the physical and even commercial handling of the platforms. Indeed, some of them are still underdeveloped (RubyRTL, PRGA), and others have already marked their presence in the market via the contribution to the realization of commercialized systems, or they are integrated and approved in an academic cursor (LiteX, PYNQ, PyMTL, SysPy). While OpenESP, ESP4ML, HLS4ML, AnyHLS, PyOCN, PyRTL, DS3, and PyLog have completed evaluation and verification experiments.

SoCs in general are now proposed as emerging and scalable solutions for intelligent applications. Hence, design methodologies must incorporate new functionalities satisfying the performance/power consumption/space/size challenges and dilemmas imposed by the evolution of advanced technology.

REFERENCES

[1] F. Kermarrec, S. Bourdeauducq, J.-C. L. Lann, and H. Badier, "LiteX: an open-source SoC builder and library based on Migen Python DSL," ArXiv200502506 Cs, May 2020, Accessed: Apr. 10, 2021. [Online]. Available: http://arxiv.org/abs/2005.02506

[2] "Litex_introduction," GitHub. https://github.com/enjoy-digital/litex (accessed Apr. 04, 2021).

[3] J.-C. L. Lann, H. Badier, and F. Kermarrec, "Towards a Hardware DSL Ecosystem: RubyRTL and Friends," ArXiv200409858 Cs, Apr. 2020, Accessed: Apr. 10, 2021. [Online]. Available: http://arxiv.org/abs/2004.09858
[4] S. Jiang, P. Pan, Y. Ou, and C. Batten, "PyMTL3: A Python Framework for Open-Source Hardware Modeling, Generation, Simulation, and Verification," IEEE Micro, vol. 40, no. 4, pp. 58–66, Jul. 2020, DOI: 10.1109/MM.2020.2997638.
[5] S. Jiang, C. Torng, and C. Batten, "An Open-Source Python-Based Hardware Generation, Simulation, and Verification Framework," San Diego, p. 5, 2018.
[6] D. Lockhart, G. Zibrat, and C. Batten, "PyMTL: A Unified Framework for Vertically Integrated Computer Architecture Research," in 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, United Kingdom, Dec. 2014, pp. 280–292. DOI: 10.1109/MICRO.2014.50.
[7] C. Tan et al., "PyOCN: A Unified Framework for Modeling, Testing, and Evaluating On-Chip Networks," in 2019 IEEE 37th International Conference on Computer Design (ICCD), Abu Dhabi, United Arab Emirates, Nov. 2019, pp. 437–445. DOI: 10.1109/ICCD46524.2019.00068.
[8] B. R. Group, PyMTL3-NET: PyMTL3-Net: an open-source Python-based framework for modeling, testing, and evaluating on-chip interconnection networks. Accessed: Apr. 07, 2021. [Online]. Available: https://github.com/cornell-brg/pymtl3-net
[9] J. Clow, G. Tzimpragos, D. Dangwal, S. Guo, J. McMahan, and T. Sherwood, "A pythonic approach for rapid hardware prototyping and instrumentation," in 2017 27th International Conference on Field Programmable Logic and Applications (FPL), Ghent, Belgium, Sep. 2017, pp. 1–7. DOI: 10.23919/FPL.2017.8056860.
[10] Evangelos Logaras, Max-Delbrück-Centrum für Molekulare Medizin, and Orsalia Georgia Hazapis, "Python to Accelerate Embedded SoC Design: A Case Study for Systems Biology," ACM Trans. Embed. Comput. Syst., p. Vol. 13, No. 4, Article 84, Feb. 2014, doi: 10.1145/2560032.
[11] P. Mantovani et al., "Agile SoC Development with Open ESP," Proc. 39th Int. Conf. Comput.-Aided Des., pp. 1–9, Nov. 2020, DOI: 10.1145/3400302.3415753.
[12] "PYNQ Introduction — Python productivity for Zynq (Pynq)." https://pynq.readthedocs.io/en/v2.6.1/ (accessed Apr. 06, 2021).
[13] J. Goldsmith, C. Ramsay, D. Northcote, K. Barlee, L. Crockett, and R. Stewart, "Control and Visualisation of a Software Defined Radio System on the Xilinx RFSoC Platform Using the PYNQ Framework," IEEE Access, vol. PP, pp. 1–1, Jul. 2020, DOI: 10.1109/ACCESS.2020.3008954.
[14] S. E. Arda et al., "DS3: A System-Level Domain-Specific System-on-Chip Simulation Framework," ArXiv200309016 Cs, Mar. 2020, Accessed: Apr. 10, 2021. [Online]. Available: http://arxiv.org/abs/2003.09016
[15] F. Fahim et al., "hls4ml: An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices," ArXiv210305579 Phys., Mar. 2021, Accessed: Apr. 10, 2021. [Online]. Available: http://arxiv.org/abs/2103.05579
[16] D. Giri, K.-L. Chiu, G. Di Guglielmo, P. Mantovani, and L. Carloni, "ESP4ML: Platform-Based Design of Systems-on-Chip for Embedded Machine Learning," Feb. 2020. doi: 10.23919/DATE48585.2020.9116317.
[17] A. Li and D. Wentzlaff, "PRGA: An Open-source Framework for Building and Using Custom FPGAs," p. 6.
[18] S. Huang, K. Wu, H. Jeong, C. Wang, D. Chen, and W. Hwu, "PyLog: An Algorithm-Centric Python-Based FPGA Programming and Synthesis Flow," in The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Virtual Event USA, Feb. 2021, pp. 227–228. DOI: 10.1145/3431920.3439478.
[19] M. A. Özkan et al., "AnyHLS: High-Level Synthesis with Partial Evaluation," IEEE Trans. Comput.-Aided Des. Integer. Circuits Syst., vol. 39, no. 11, pp. 3202–3214, Nov. 2020, DOI: 10.1109/TCAD.2020.3012172.
[20] R. Leißa et al., "AnyDSL: a partial evaluation framework for programming high-performance libraries," Proc. ACM Program. Lang., vol. 2, no. OOPSLA, pp. 1–30, Oct. 2018, DOI: 10.1145/3276489.
[21] X. Tang, E. Giacomin, B. Chauviere, A. Alacchi, and P.-E. Gaillardon, "OpenFPGA: An Opensource Framework for Agile Prototyping Customizable FPGAs," IEEE Micro, vol. PP, pp. 1–1, May 2020, DOI: 10.1109/MM.2020.2995854.
[22] K. Team, "Keras documentation: About Keras." https://keras.io/about/ (accessed Apr. 10, 2021).
[23] "ONNX | About." https://onnx.ai/about.html (accessed Apr. 26, 2021).
[24] "Impala_Introduction," AnyDSL. https://anydsl.github.io/Impala (accessed Apr. 10, 2021).
[25] "Tutorial_Impala," AnyDSL. https://anydsl.github.io/Tutorial.html (accessed Apr. 10, 2021).
[26] "Programming Language — The Linux Kernel documentation." https://www.kernel.org/doc/html/latest/process/programming-language.html (accessed Apr. 10, 2021).
[27] F. Callaly, D. O'Loughlin, D. Lyons, A. Coffey, and F. Morgan, "Xilinx Vivado High-Level Synthesis: Case studies," in 25th IET Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communities Technologies (ISSC 2014/CIICT 2014), Limerick, Ireland, 2014, pp. 352–356. DOI: 10.1049/cp.2014.0713.
[28] Z. Zhao and J. C. Hoe, "Using Vivado-HLS for Structural Design: a NoC Case Study," ArXiv171010290 Cs, Aug. 2020, Accessed: Apr. 11, 2021. [Online]. Available: http://arxiv.org/abs/1710.10290
[29] S. O. Ayat, M. Khalil-Hani, and R. Bakhteri, "OpenCL-based hardware-software co-design methodology for image processing implementation on heterogeneous FPGA platform," in 2015 IEEE International Conference on Control System, Computing and Engineering (ICCSCE), Penang, Malaysia, Nov. 2015, pp. 36–41. doi: 10.1109/ICCSCE.2015.7482154.
[30] A. G. Schmidt, G. Weisz, and M. French, "Evaluating Rapid Application Development with Python for Heterogeneous Processor-Based FPGAs," 2017 IEEE 25th Annu. Int. Symp. Field-Program. Cust. Comput. Mach. FCCM, 2017, doi: 10.1109/FCCM.2017.45.

[31] "PYNQ Overlays — Python productivity for Zynq (Pynq)." https://pynq.readthedocs.io/en/v2.6.1/pynq_overlays.html (accessed Apr. 11, 2021).
[32] "VPR — Verilog-to-Routing 8.1.0-dev documentation." https://docs.verilogtorouting.org/en/latest/vpr/ (accessed Apr. 08, 2021).
[33] X. Tang, E. Giacomin, A. Alacchi, B. Chauviere, and P.-E. Gaillardon, "OpenFPGA: An Opensource Framework Enabling Rapid Prototyping of Customizable FPGAs," in 2019 29th International Conference on Field Programmable Logic and Applications (FPL), Barcelona, Spain, Sep. 2019, pp. 367–374. doi: 10.1109/FPL.2019.00065.
[34] D. Shah, E. Hung, C. Wolf, S. Bazanski, D. Gisselquist, and M. Milanovic, "Yosys+nextpnr: An Open-Source Framework from Verilog to Bitstream for Commercial FPGAs," in 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), San Diego, CA, USA, Apr. 2019, pp. 1–4. doi: 10.1109/FCCM.2019.00010.
[35] "Yosys Open SYnthesis Suite: About." http://www.clifford.at/yosys/about.html (accessed Apr. 11, 2021).
[36] m-labs/migen. M-Labs, 2021. Accessed: Apr. 14, 2021. [Online]. Available: https://github.com/m-labs/migen
[37] "The FHDL domain-specific language — Migen 0.8. dev0 documentation." https://m-labs.hk/migen/manual/fhdl.html (accessed Apr. 14, 2021).
[38] "M-Labs_nMigen," M-Labs. / (accessed Apr. 14, 2021).
[39] "doc · JC-LL/sexpir@dc21c58," GitHub. /JC-LL/sexpir/commit/dc21c580bc31d7b27e492c70ee25c7b25819db9b (accessed Apr. 14, 2021).
[40] T. Aarrestad et al., "Fast convolutional neural networks on FPGAs with hls4ml," ArXiv210105108 Hep-Ex Physicsphysics Stat, Jan. 2021, Accessed: Apr. 22, 2021. [Online]. Available: http://arxiv.org/abs/2101.05108
[41] tensorflow/tensorflow. tensorflow, 2021. Accessed: Apr. 23, 2021. [Online]. Available: https://github.com/tensorflow/tensorflow
[42] pytorch/PyTorch. pytorch, 2021. Accessed: Apr. 23, 2021. [Online]. Available: https://github.com/pytorch/pytorch
[43] pytorch/PyTorch. pytorch, 2021. Accessed: Apr. 26, 2021. [Online]. Available: https://github.com/pytorch/pytorch
[44] XMR2010 17th IEEE International Conference on Electronics, Circuits, and Systems, Dec. 2010, pp. 762–765. doi: 10.1109/ICECS.2010.5724624.
[45] XMR/pynq_package.html (accessed Apr. 22, 2021).
[46] XMR.1/pynq_libraries.html (accessed Apr. 22, 2021).
[47] "Overlay Design Methodology — Python productivity for Zynq (Pynq)." https://pynq.readthedocs.io/en/v2.6.1/overlay_design_methodology.html (accessed Apr. 22, 2021).
[48] XMR/ghdl
[49] W. Snyder, "Verilator: Fast, Free, But for Me?" p. 40.
[50] "Veripool." https://www.veripool.org/verilator/ (accessed Apr. 28, 2021).
[51] "PYNQ emulator." https://riku-s.github.io/led-matrix/ (accessed May 02, 2021).
[52] "PYNQ - Python productivity for Zynq," PYNQ - Python productivity for Zynq. https://www.pynq.io/ (accessed May 02, 2021).
[53] "We wanted to maintain the efficiency of NeutralizePath() · EoflaOE/Kernel-Simulator@dd2b8ad," GitHub. /EoflaOE/Kernel-Simulator/commit/dd2b8adbb860ec8aa707c3d13f5493e0a1614d37 (accessed May 02, 2021).
[54] falconcomputing/merlin-compiler. Falcon Computing Solutions, Inc., 2021. Accessed: May 02, 2021. [Online]. Available: https://github.com/falconcomputing/merlin-compiler
[55] "Falcon Computing," Xilinx. https://www.xilinx.com/about/xilinx-ventures/falcon-computing.html (accessed May 02, 2021).
[56] "GCC, the GNU Compiler Collection - GNU Project - Free Software Foundation (FSF)." https://gcc.gnu.org/ (accessed May 02, 2021).
[57] "Vivado Simulation Flow," Xilinx. https://www.xilinx.com/products/design-tools/vivado/simulation.html (accessed May 02, 2021).
[58] "Vivado Simulator," Xilinx. https://www.xilinx.com/products/design-tools/vivado/simulator.html (accessed May 02, 2021).
[59] "VCS Functional Verification Solution." https://www.synopsys.com/verification/simulation/vcs.html (accessed May 02, 2021).
[60] "High-Performance Simulation." https://www.synopsys.com/verification/simulation.html (accessed May 02, 2021).
[61] "enjoy-digital/litex," GitHub. https://github.com/enjoy-digital/litex (accessed May 03, 2021).
[62] "Enjoy-Digital » FPGA-based design services and Open-Source." http://enjoy-digital.fr/ (accessed May 03, 2021).
[63] M. U. Khan, A. Ishtiaq, S. Ali, K. Habib, S. Samer, and E. Hafeez, A Review of System on Chip (SOC) Applications in Internet of Things (IOT) and Medical. 2021.
[64] X. Tang, E. Giacomin, A. Alacchi, B. Chauviere, and P.-E. Gaillardon, "OpenFPGA: An Opensource Framework Enabling Rapid Prototyping of Customizable FPGAs," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL) (pp. 367-374)*