Flexible Software to Hardware migration methodology for FPGA design and verification.

Matias Trapaglia[1], Ricardo Cayssials[2,3], Lorenzo De Pasquale[2], Edgardo Ferro[3]

[1]CONICET,

[2]UTN – FRBB,

Department of Electronics Engineering,[3]Universidad Nacional del Sur

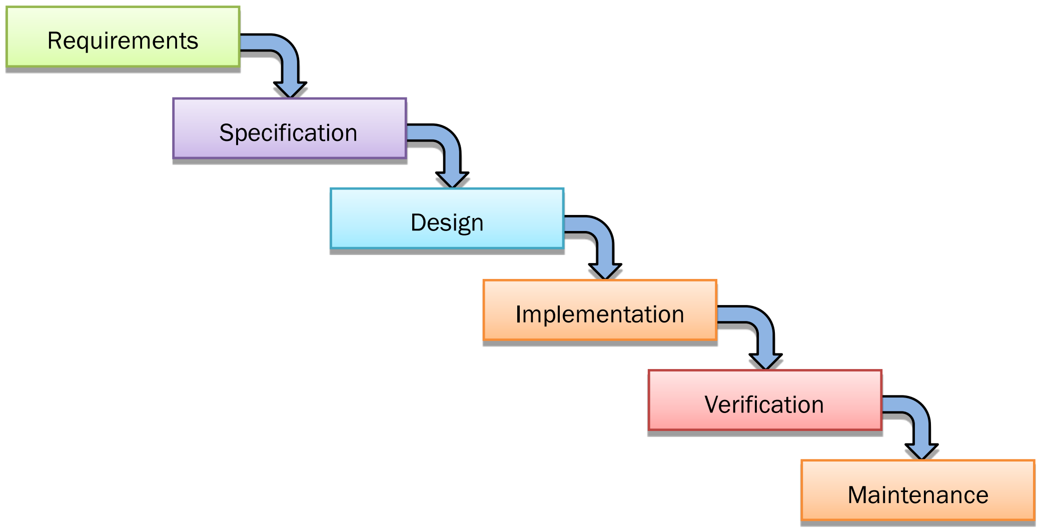
Bahía Blanca - Argentina

Goals

- To propose a Software to Hardware migration methodology for design, testing and verification

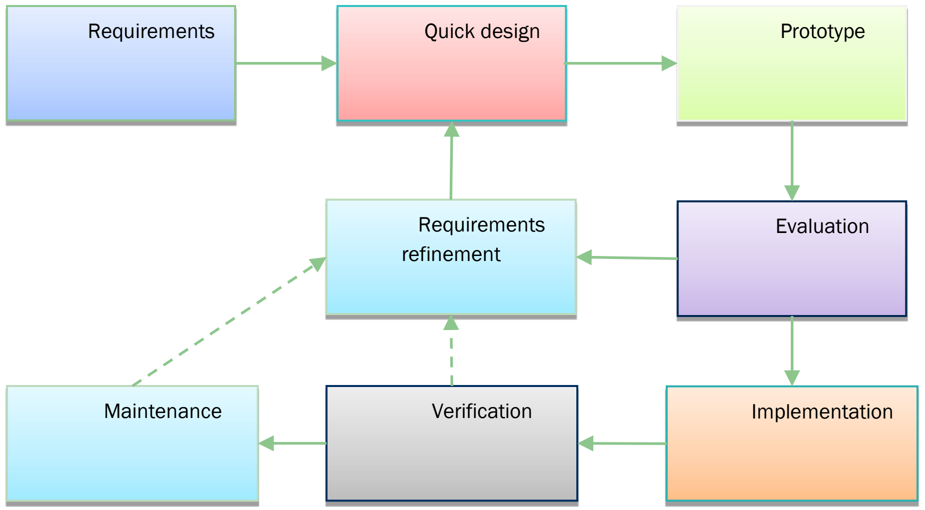
Introduction

- In traditional hardware design methodologies, updating and upgrading hardware functionalities involve expenses difficult to justify in most of the modern developments.
- Waterfall model



Introduction

- FPGA reconfiguration feature allows adopting software techniques to shorten production cycles, time-to-market metrics and increase the life cycle of the design.
- Prototype model



Introduction

- Software development based on Agile approaches allows flexible and cooperative development processes adequate to the complexity of modern designs. Several platforms, software languages, and tools were proposed to support these software methodologies.

Co-modelling / co-design

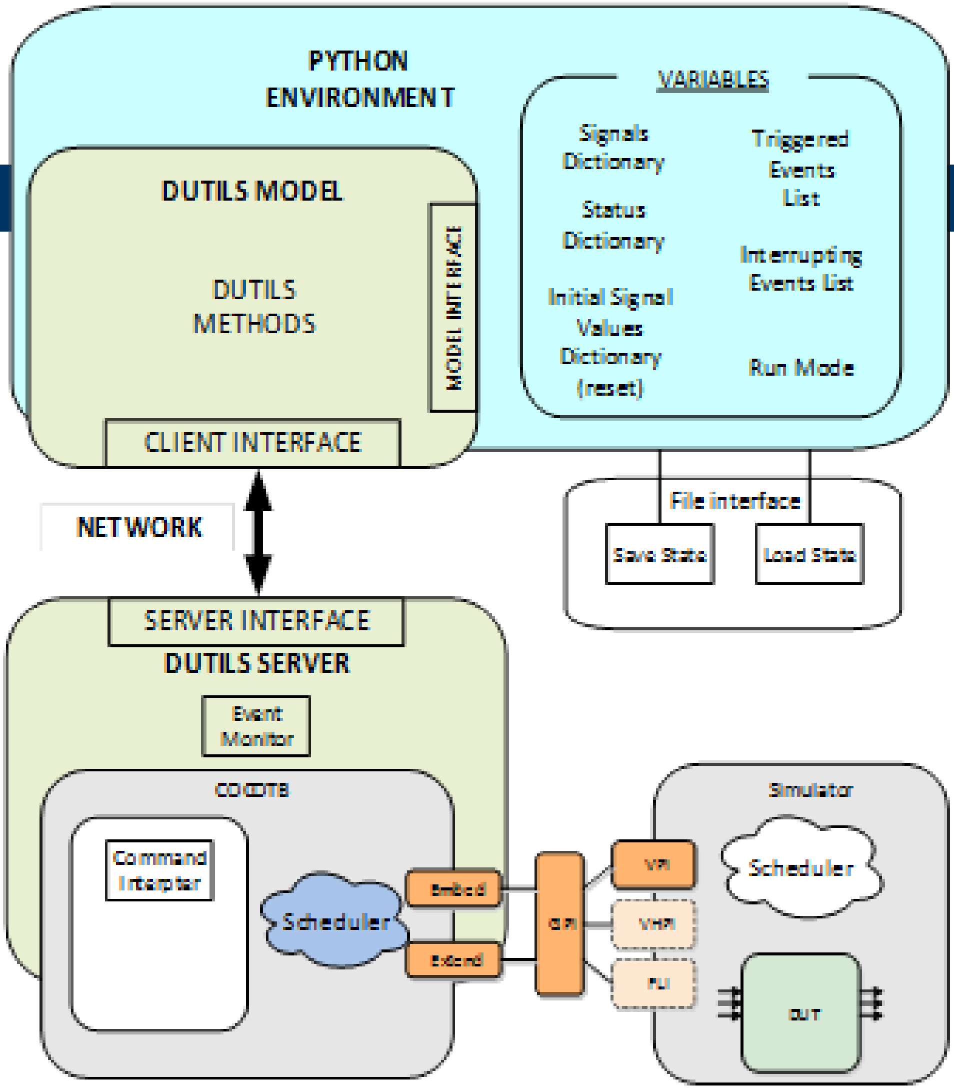
- Co-modelling lets developers investigate and compare systematically different software and hardware partitions to meet systems' constraints earlier in the design process when integration problems are easier and cheaper to resolve.
- The complex interdependencies among software and hardware components aren't often adequately reflected by the functional requirements of an integrated software/hardware system.

Cocotb

- Cocotb is an open source CO-routine-based CO-simulation TestBench environment for verifying VHDL and Verilog RTL using Python [6].
- Cocotb helps to interface different hardware simulation tools with a Python environment where the testbench might be programmed.

DUTILS

- DUTILS [7] extends Cocotb’s functionalities by wrapping the simulation in a class, making the use of the simulated component inside a routine possible.
- This class also allows controlling the execution flow, with conditional stops and step by step debugging, useful in the design stage.


PYTHON ENVIRONMENT
VARIABLES
Signals Dictionary
Status Dictionary
Initial Signal Values Dictionary (reset)
Triggered Events List
Interrupting Events List
Run Mode
DUTILS MODEL
DUTILS METHODS
MODEL INTERFACE
CLIENT INTERFACE
NETWORK
File interface
Save State
Load State
SERVER INTERFACE
DUTILS SERVER
Event Monitor
COCOTB
Command Interpter
Scheduler
Embed
Extend
GPI
VPI
VHPI
FLI
Simulator
Scheduler
DUT

Migration Methodology

- Step 1: Software modeling stage

  A model-in-the-loop methodology is used in the early stages of the development process to outstretch the main component and interfaces of the system. Further refinements may detail the method, properties and requirements of each one of the components in the system, usually modeled as object-oriented classes.

Migration Methodology

- Step 2: Partition stage

  The set of system components to be implemented in hardware has to be defined.

  HW/SW implementation of a function is much easier, without the need of changing the software structure

Migration Methodology

- Step 3: Hardware components description

  Hardware components should be described through a hardware description language, synthesizable for FPGA devices.

  With DUTILS flow control, the behavior of the DUT can be traced and compared with the software model; watching the internal registers of the hardware component allows incremental programming from basic functionality to the full-featured component.

Migration Methodology

- Step 4: Software routines integration with DUTILS framework

  The DUTILS model is moved from a stand-alone class inside testing scripts, to form part of an interconnected project, consisting of software/hardware components. With the built-in methods, is an easy task. Only a few additional functions have to be created to adjust software variables to the hardware signals timing.

Migration Methodology

- Step 5: Hardware / Software interface specification

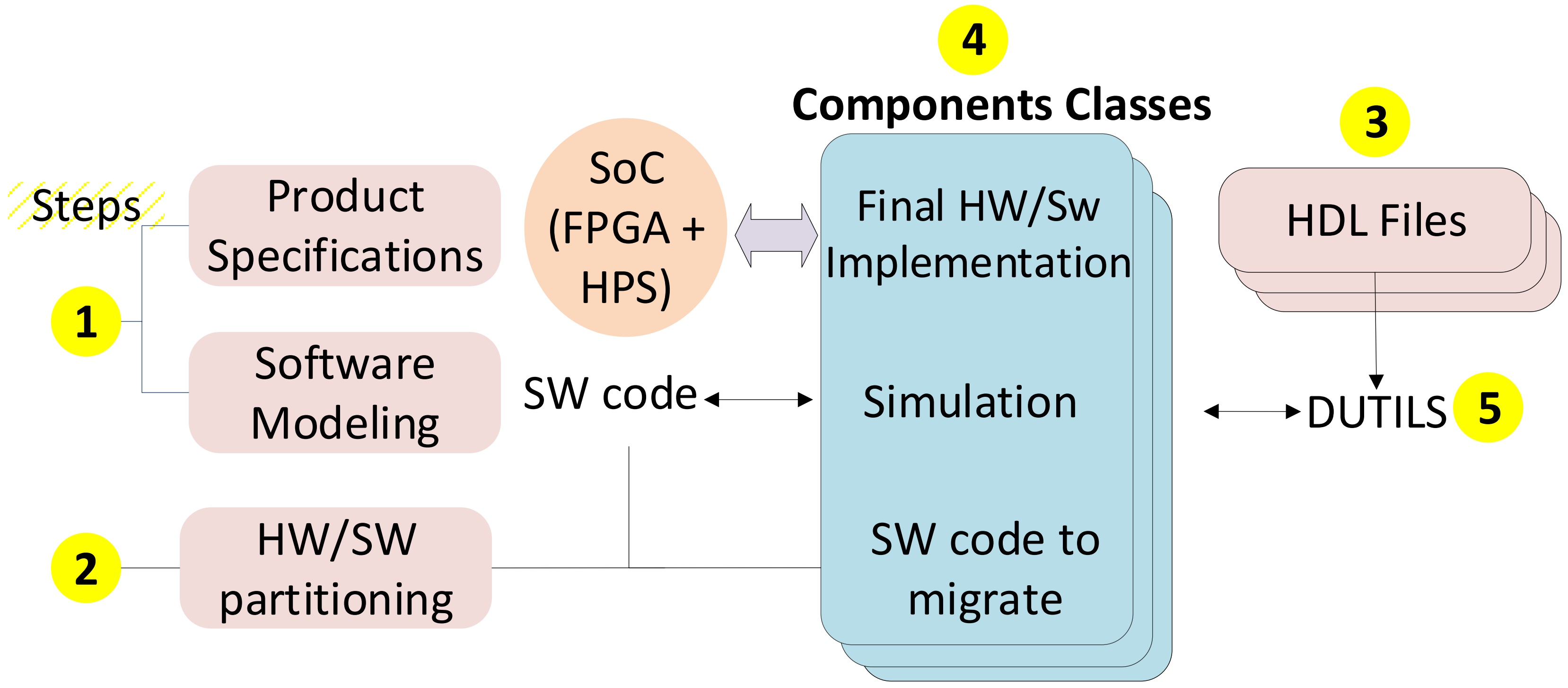
  The hardware component ports and signals are retrieved by the DUTILS framework into a Python dictionary through Cocotb. In this way, no software code modification is needed; the same codification is useful for the simulation as well as for the final implementation, implying a reduction in development times.

Migration Methodology

- Step 6: Platform deployment

  If the project platform is a SOC, after the verification succeeded the hardware part can be automatically compiled and configured into the FPGA logic, and the software stored in a memory that the HPS has access to. The communication between both is also automated. This way a final implementation is achieved, making possible to obtain the real metrics.

Process Development flow



Case Study: Fast Fourier transform

- Frequency domain analysis is widely used in diverse engineering areas. They are based on the Fourier transform that determines the frequency components of periodic signals. For sampled signals, the Discrete Fourier transform is expressed as:

$$H_n \equiv \sum_{k=0}^{N-1} h_k \, e^{2\pi i k n/N}$$

FFT implementation

- FFT is a recursive algorithm implemented iteratively in [12].
- The hardware architecture was verified in the DUTILS framework for 16, 32, 64, 128, 256 and 512 samples.
- Data width were modified from 8 to 12 integer bits and 3 to 9 decimal bits.

CS: System design

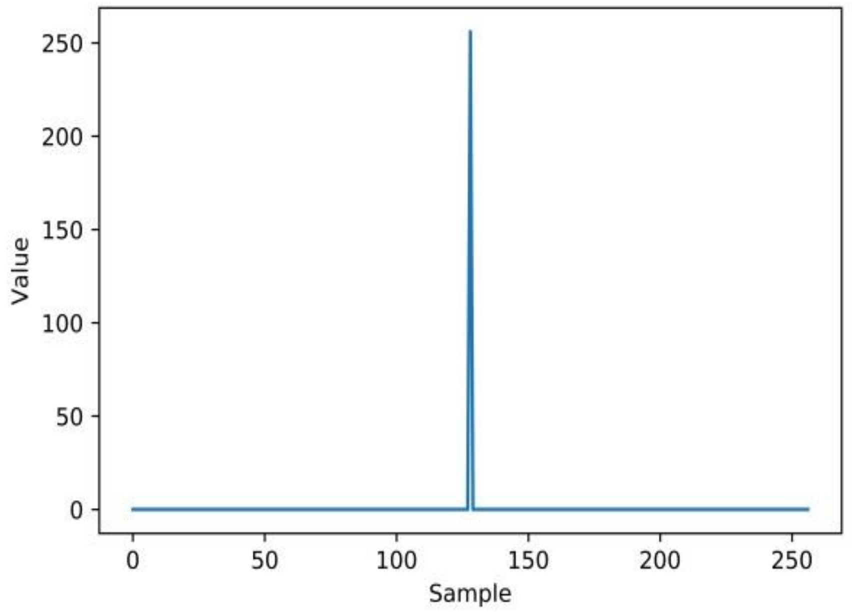
- signal generation class: this class produces an array of the sampled values according to a certain function.
- FFT computation class: returns a complex array of the frequency components. This class is the wrapper of both: (1) the Python imported FFT library and (2) the hardware implementation of the FFT algorithm.
- Control and monitoring class: this class is in charge of transferring the data values produced by the signal generation class to the FFT computation class and check and visualize the results from the FFT computation class.

Verification: data generation

- Python environment offers a simple way to produce complex analysis.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.fftpack
N = 256
signal = np.zeros(N + 1)
for k in range(1, N + 1):
    x = np.linspace(-np.pi, np.pi, N + 1)
    signal = signal + np.cos(k * x)

plt.plot(np.linspace(0, N, N+1), signal)
plt.xlabel('Sample')
plt.ylabel('Value')
plt.show()
```



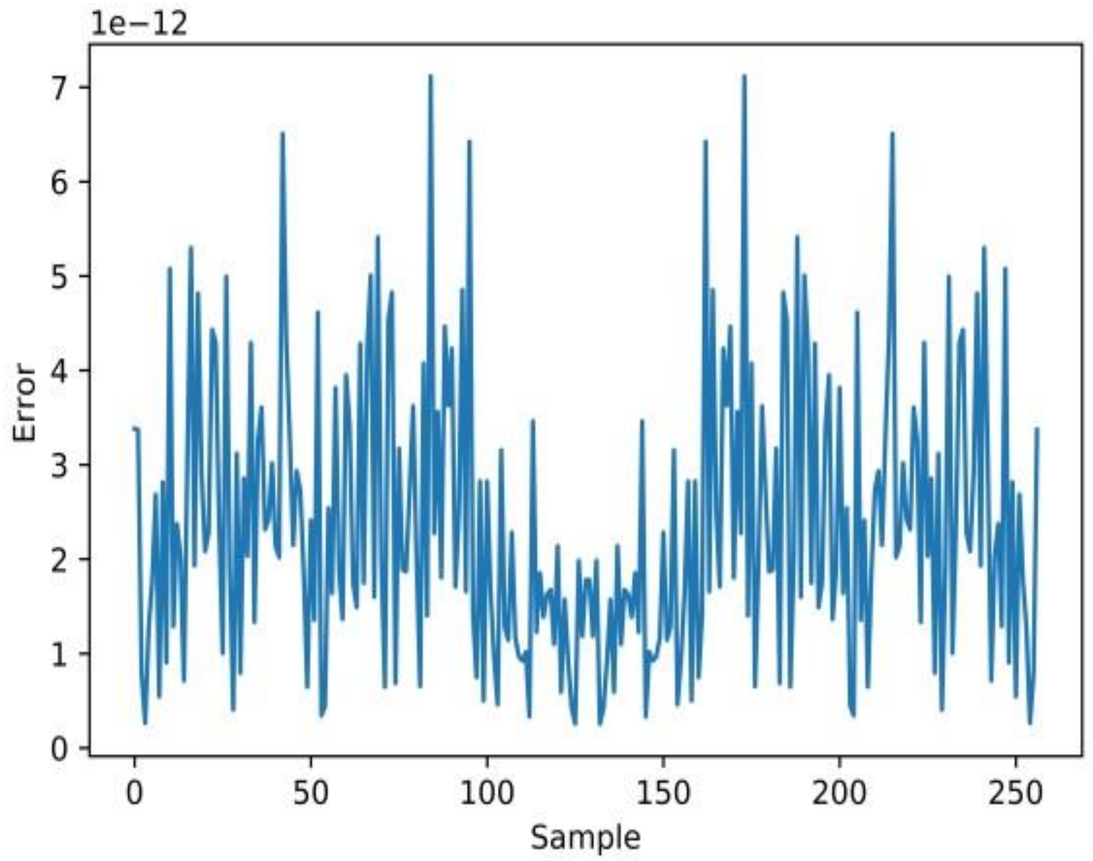
Verification: data verification

```
FFT_soft = scipy.fftpack.fft(signal)
DUT_FFT  = FFT_hard(signal)

error = np.absolute(FFT_soft - DUT_FFT)
plt.plot(np.linspace(0, N, N + 1),error)
plt.xlabel('Sample')
plt.ylabel('Error')
plt.axis('tight')
plt.show()
```

Verification: SF/HW error

- For white noise:



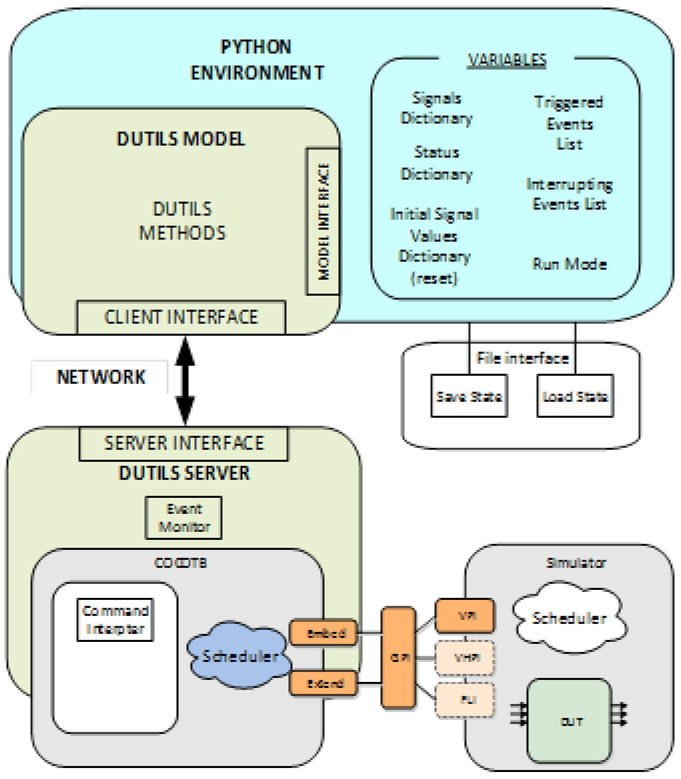
FFT hardware implementation

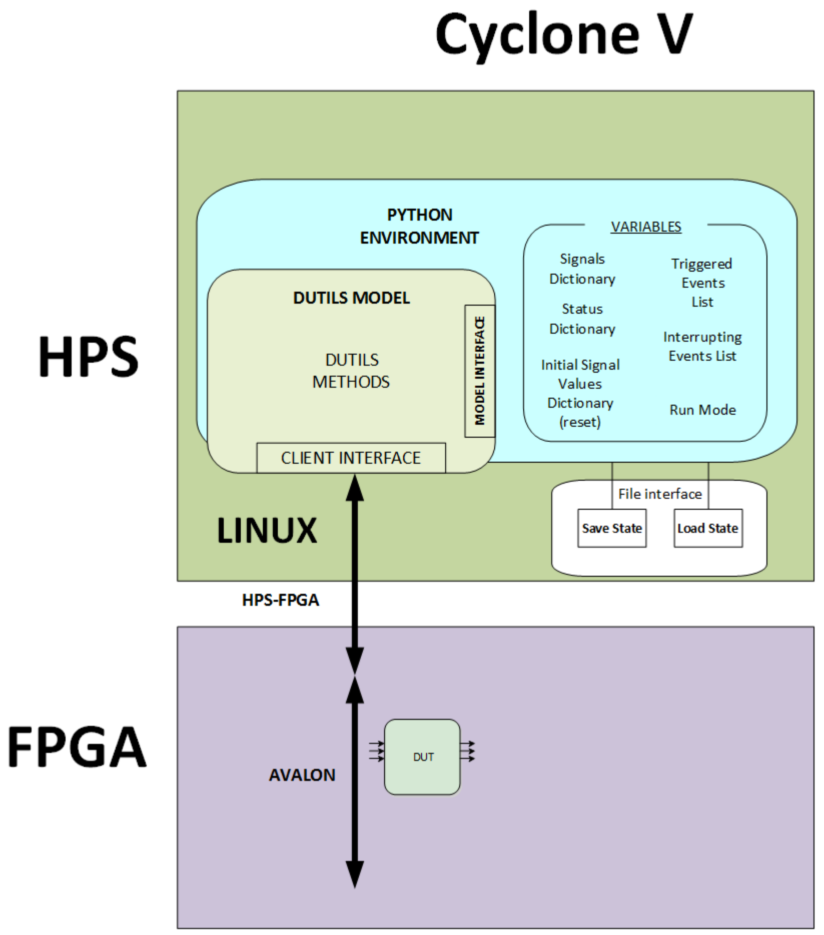
- A concurrent FFT implementation for 512 samples of 16 bits may take around 10,000 logic elements for a 10MHz sampling time in a Cyclone IV E Intel FPGA device.

Methodology Analisys

- The FFT transform proposed as a case study was easily implemented and tested with data structures supported by Python language.
- Migration from Python to HDL could be performed gradually, verifying automatically the consistency between Python variables and hardware signals.
- Wide range of data structures in Python reduces the complexity of producing efficient testbenches.

System implementation





Conclusions

- Flexible hardware design techniques has to be proposed.
- The DUTILS framework helps to achieve a fast migration from software to hardware.
- Migration methodology allows using the same modeling environment for software developments, hardware and software verification, and final deployment.
- A digital signal processing FFT was proposed to highlight the flexibility of the migration methodology to develop and verify complex algorithms in digital hardware.
- A complex iterative algorithm with an array of complex values was easily migrated and verified using the highly efficient data structures and imported functions of Python

Thanks

- Thanks for your attention.

rcayssials@frbb.utn.edu.ar