A Generic FPGA Emulation Framework

Fernando Moraes, Matheus Moreira, Carlo Lucas, Dairan Corrêa, Douglas Cardoso, Maurício Magnaguagno, Guilherme Castilhos, Ney Calazans
PUCRS – FACIN – Av. Ipiranga 6681 – Porto Alegre – 90619-900 – Brazil
{first name.last name}@acad.pucrs.br, {fernando.moraes, ney.calazans}@pucrs.br

Abstract — Verification techniques face growing challenges, as digital system design becomes increasingly complex. Currently, verification is believed to be the main bottleneck for expedite complex designs, consuming at least 70% of the whole system development effort. This paper proposes a new, generic hardware emulation framework to improve the observability of designs as well as reducing emulation-based verification intrusiveness. The proposed emulator provides enhanced observability and controllability of inner workings of the system when compared to commercial FPGA-based emulators and is less intrusive on the design under verification. As FPGA-vendor specific products, the proposed emulator is generic, supporting in principle any digital system design. To enhance flexibility, stimuli generation and response capture is under control of a host computer and communication between the host and the design under verification may occur through an Ethernet interface or through PCIe interfaces in supported platforms. The prototype of the proposed framework is operational and presents promising results in terms of observability and controllability enhancement, although further work is needed to improve the framework emulation performance.

I. Introduction and Related Work

During the process of creating a digital circuit deemed for FPGA implementation, the verification step is fundamental for the correct development of the system. The main verification alternatives include simulation, prototyping and emulation. Simulation of large digital circuits can take a very long time, yet they present the best observability and controllability figures for the design under verification (DUV). Prototyping, on the other hand, provides a much higher verification speed when compared to simulation, but observability is small. The emulation technique combines the qualities of both simulation and prototyping approaches: very good observability and controllability characteristics together with verification speed.

Examples of tools to accelerate the simulation process include Veloce [1] and Palladium [2] products. Both significantly increase simulation speed compared to RTL level software simulation. Their disadvantage is their cost, making acquisition feasible only for large corporations. FPGA vendors offer proprietary tools to reduce the verification time. The Chipscope Analyzer [3] from Xilinx probes internal signals of the design inside an FPGA, enabling the visualization of results as in logic analyzers devices. The advantages of Chipscope are the evaluation of the DUV at-speed and the possibility to observe part of the internal state of the DUV using programmable triggers. On the other hand, Chipscope uses many FPGA resources (RAM memory blocks) to store the observed signals, being the maximum amount of sampled signals a function of available memory blocks. A second tool, iSim [4], integrates simulation with prototyping. The goal is to select one module of the design to be emulated in FPGAs. The advantages of this approach are the reduced verification time and the integration with a logic simulator. The main limitation of iSim is that only one instance of the design can be selected for emulation, and it cannot be the top-level of the DUV. The emulation approach (e.g. iSim), compared to prototyping (e.g. Chipscope), does not enable at-speed verification, since the host alone controls the DUV clock.

Chuang et al. [5] propose an approach for providing full signal visibility for functional debugging on an FPGA. The idea is to record the FPGA internal behavior and replay the period of interest in a software simulator. With this method, the Authors obtain high simulation speed because most of the simulation effort finishes in the FPGA. They also propose an optimization algorithm that minimizes the amount of recorded data, thus reducing hardware overhead. Tan et al. [6] propose a generic design flow for the emulation of networks-on-chip (NoCs) on FPGA platforms. The design flow builds the emulation architecture based on the specific NoC architecture, the type of emulation and the routing algorithm. The whole emulation architecture is designed in a hierarchical synthesizable VHDL description, being FPGA independent. This work enables to emulating a particular type of digital circuit, NoCs, to achieve speedup compared to RTL simulation. Evaluated performance parameters include latency and throughput. The observability of the NoC internal signals is not an issue of the proposed work.

The main *goal* of the present work is presenting the development of a generic FPGA emulation framework. Its main features comprise: (*i*) *low intrusiveness*, does not requiring to include hard IPs in the DUV; (*ii*) *high*

The main idea behind this work has been suggested by TERACOM Telemática. Also, the same enterprise financially sponsored most of the work. The authors acknowledge the support granted by CNPQ and FAPESP to the INCT-SEC (National Institute of Science and Technology Embedded Critical Systems Brazil), under processes 573963/2008-8 and 08/57870-9.

observability: it is possible to observe any signal of the DUV just as in a software simulation; (*iii*) *controllability*, i.e., stimuli are obtained from the host responsible to control the emulation process, being possible to inject any input pattern; (*iv*) *generic*, any digital circuit may, in principle, be emulated; (*v*) FPGA vendor independent. The limitations of the proposed emulation framework include verification with a clock frequency defined by the host (natural limitation of the emulation technique) and supporting (so far) only one clock domain for the entire DUV. The Authors did not identify in the literature a generic emulation framework as the one proposed herein. Vendor tools, as *iSim*, may present similar features as our work, except for the observability figures.

II. Emulator Framework

The proposed emulation framework contains hardware and software parts, as Figure 1 illustrates. The hardware part contains the emulated DUV and additional communication logic with a host computer. The host computer sends/receives data to/from the DUV, and displays the emulation results in a graphical interface (in our case the freeware GTKwave).

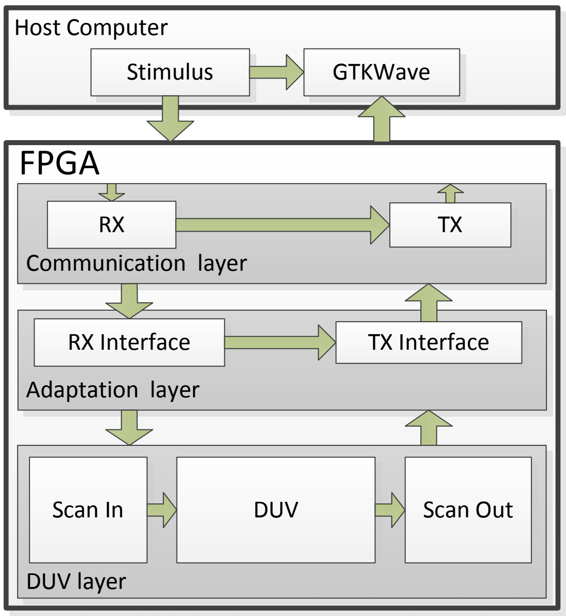

Figure 1 - Software and hardware architecture of the emulator framework.

The software running in the host computer has additional functionalities, as will be explained in Section IV. Relevant functionalities comprise: (*i*) create the hardware description of the system, with the signals to observe, according to the designer specification; (*ii*) generate the hardware bitstream; (*iii*) communicate with the prototyping board. The hardware part contains three layers: (*i*) communication layer; (*ii*) adaptation layer; (*iii*) DUV layer. The *communication layer* (CL) contains an MII Ethernet interface, which communicates with the external world. A MAC hard IP is supposed to exist in the FPGA. As many FPGAs have such modules available, this is not considered a limitation of the framework. The CL is connected to the MAC IP, implementing the link, IP and transport layers. The CL also implements the DHCP protocol that enables to connect the FPGA directly to an Internet network. A modified version of the UDP protocol was developed, including in the packets a sequence number and CRC in the payload, to enhance robustness of the UDP protocol. A previous version of the emulation framework adopted a PCIe interface. Such interface is less flexible, since it requires dedicated platforms, and it is not as universal as Ethernet connections. For such reasons the Ethernet interface was finally adopted. The second layer, named *adaptation layer*, is responsible for: (*i*) adapting the incoming/outcoming data to the format used in the DUV layer (32 bits); (*ii*) synchronizing the data flow between the DUV and communication layers. The first and second layers are DUV independent. Their area is 1,115 Slices, 1,974 FFs, and 9 BRAMs. For the FPGA of our experiments, Xilinx Virtex5 XC5VLX330T, this represents an area overhead of 2.1%.

III. The DUV Layer

The DUV layer contains the instantiation of the DUV itself, and a wrapper logic, responsible for controlling the insertion of stimuli, transmission of the computed data, and for the DUV clock generation. Figure 2 illustrates the DUV layer architecture. Each primary input of the DUV connects to a scan-chain cell, which enables to select a signal incoming from the external world (FPGA pin) or from the adaptation layer. The scan-chain cells are grouped in sets of 32 cells. In this way, during the stimuli load from the adaptation layer, at each clock cycle it is possible to store the value of 32 inputs. Consider for example that the DUV has 420 inputs. It is necessary to insert 14 sets of scan-chain cells, and therefore 14 clock cycles are needed to store all 420 input signal values.

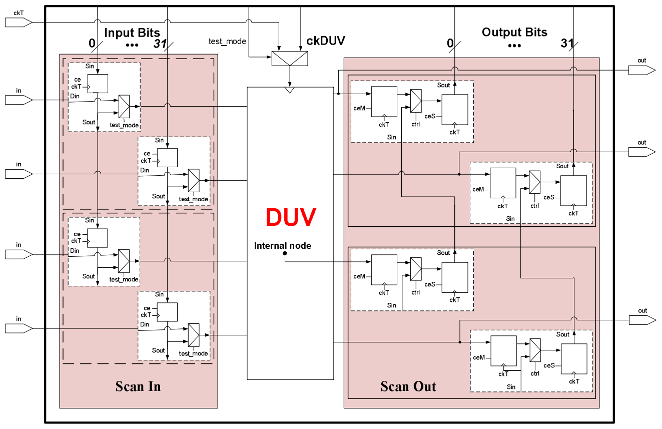

Figure 2 – DUV Layer.

The primary outputs and selected internal nodes are also connected to scan-chain cells, grouped in 32-bit sets. These scan-chains are built using a master-slave architecture. At the end of one DUV clock cycle the computed values are stored in the master flip-flop of each scan-chain cell. Then, in subsequent clock cycles these values are shifted to the adaptation layer through the slave stages.

Figure 3 details the DUV clock generation process. The first event ('1' in the Figure) corresponds to the load of new patterns in the input scan-chains, as well as the storage of the computed values in the output scan-chains. The duration of this step, in system clock (ckT) cycles, is a function of the number of input or output scan-chain sets. Next, the adaptation layer enables the raising edge of the DUV clock (*ckDUV*), event '2'. The DUV clock stays active during a parameterizable number of system clock cycles. This enables to emulate the DUV in a frequency slower than the system clock. The last event, '4', disables the DUV clock. This sequence of events (1-2-3-4) repeats for each input stimulus.

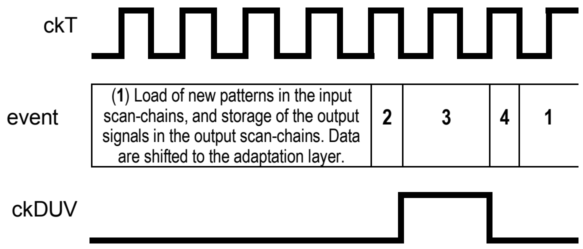
ckT
event
(1) Load of new patterns in the input scan-chains, and storage of the output signals in the output scan-chains. Data are shifted to the adaptation layer.
2
3
4
1
ckDUV

Figure 3 – System (ckT), DUV clocks (ckDUV), and chain of control events.

Such architecture can be applied to any digital circuit (limited to one clock domain), ensuring:

i. *small intrusiveness*, the DUV is not modified, requiring only to "promote" internal signals to the DUV top level (this process is more detailed in the next Section);
ii. *observability*, any internal DUV signal can be observed;
iii. *controllability*, as the emulation can be controlled on a clock by clock basis, it is possible to insert any stimuli sequence in the DUV.

A dedicated scan-chain library (input and output cells) was developed using Xilinx macros (pre-synthesized blocks) to achieve small area overhead. Note that the critical path of the DUV is not modified, since no logic is added.

IV. THE EMULATION FLOW

Figure 4 presents the emulation flow, which can be divided in 4 main steps: (*i*) select the signals to observe; (*ii*) stimuli and bitstream generation; (*iii*) emulation; (*iv*) results verification.

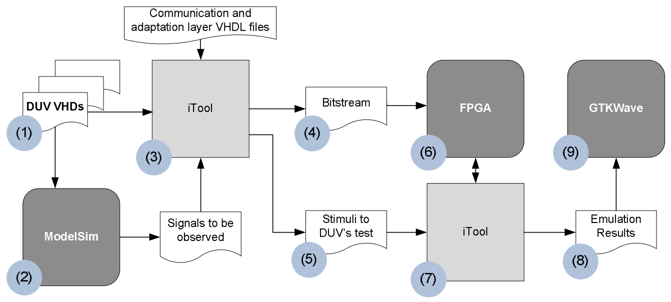

Figure 4 – Emulation flow.

A. Selection of signals to observe

Using some design entry tool, the designer creates the DUV description ('1' in Figure 4). A VHDL simulator is used to create the list of signals to be observed ('2'), as shown in Figure 5. This listing can be generated manually or using any other tool. ModelSim was adopted to simplify this step.

```
add wave /datapath/x1
add wave /datapath/x2
add wave /datapath/control/enable
add wave /datapath/output
```

Figure 5 – Example of signals to observe.

B. Stimuli and bitstream generation

The DUV description and the list of signals to observe are inputs to *iTool* (Integration Tool), '3' in Figure 4, developed in the scope of the present work. *iTool* manages a set of tools and scripts. At this step of the flow, *iTool* is responsible for:

1) "Promoting" the internal signals of the DUV to the output scan-chains, and integrating the DUV with the scan-chains, adaptation layer, and communication layer. The "promotion" of internal signals corresponds to add each observed signal as output, up to the DUV top level, adding the suffix *annotated_<instance name>_name* for each observed signal, as shown in Figure 6.

```
entity dpth is
  port (
    annotated_dpth_x1: out std_logic_vector(63 downto 0);
    annotated_dpth_x2: out std_logic_vector(63 downto 0);
    annotated_dpth_control_enable: out std_logic;
    clock    : in  std_logic;
    reset    : in  std_logic;
    input1   : in  std_logic_vector(63 downto 0);
    input2   : in  std_logic_vector(63 downto 0);
    output   : out std_logic_vector(63 downto 0)
  );
end dpth;
```

Figure 6 – Example showing the "promotion" of two internal signals of the DUV.

2) Create a synthesis script and invoke Xilinx synthesis tools. The adaptation layer and communication layer have area and timing constraints to guarantee these pieces of logic do not interfere in the DUV logic. The result is the FPGA bitstream for emulation ('4' in Figure 4).

3) Create a file with the stimuli to send to the FPGA (number '5' in Figure 4). This file contains configuration commands, such as the definition of the length of input and output scan-chains and the number of clock cycles the DUV clock should stay high, as well as payload data. The payload data corresponds to the inputs to send to the DUV at every clock cycle. Such inputs may be obtained from the DUV test bench, or through any other available method of input generation such as the use of dedicated traffic generators. The present implementation adopted the first option, for the sake of simplicity.

C. Emulation

Once the FPGA is configured, *iTool* controls the emulation process. It sends Ethernet packets to the FPGA ('6' and '7' in Figure 4), storing incoming Ethernet packets from the FPGA in the "emulation results" file ('8' in Figure 4). The process works as follows: (*i*) *iTool* reads one line of the stimuli file; (*ii*) transmits the stimulus to the FPGA; (*iii*) waits the processed data, verifying its sequence number and CRC. This process repeats for every stimulus. An ongoing work is to improve the performance of the emulation process, enabling *iTool* to transmit a parameterizable number of stimuli to the FPGA in bursts.

D. Results Verification

The last step is result analysis. Here, observed signals are analyzed as waveforms using GTKwave ('9' in Figure 4). These waveforms are generated converting the "emulation results" file to a VCD file.

V. RESULTS

Figure 7 exhibits some details of the interaction between the adaptation layer and the DUV layer. Event 1 signals a new available stimulus for the DUV. Event 2 corresponds to the reception of 4 8-bit words, which are transferred as a single 32-bit word to the DUV layer. Event 3 signals to the input scan-chain cells the arrival of a new pattern. Event 4 corresponds to the activation of the DUV clock for 8 system clock cycles. Next, Event 5 marks that the computed data is shifted into the adaptation layer. In Event 6 the adaptation layer signals to the communication layer a new result to be transmitted to the host computer.

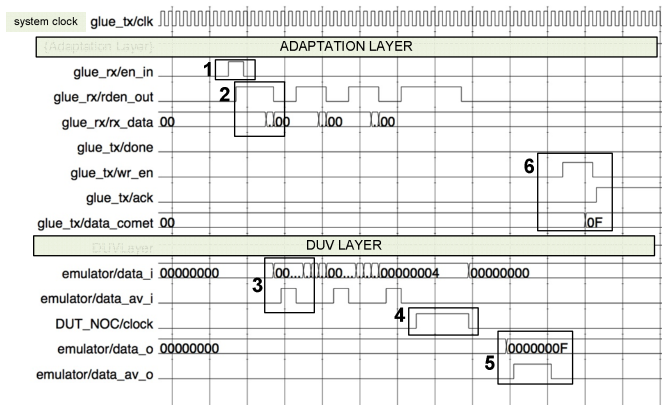

Figure 7 – Simulation showing the interaction between the adaptation layer and the DUV layer.

The emulator framework was validated using small (adders, multipliers, square root) to medium size DUVs. As an example of emulated medium size DUV is a 2x2 mesh topology network-on-chip. This DUV occupies 394 slices and 508 flip-flops. Table I compares the simulation and emulation time for the 2x2 mesh network-on-chip. For a small number of stimuli (up to 40,000) emulation is faster than simulation. However, for a large evaluation time, the emulation is slower than the simulation. This fact is due to the communication protocol adopted in the present implementation: the host sends one stimulus and waits for the answer from the FPGA. This issue is under optimization, with the host computer sending a set of stimuli at each data transmission. Note that it is not included in the emulation time the logical and physical synthesis, which can also be time-consuming. As this is done just once, and next different stimuli sets may be applied to validate the DUV, this time was not included.

Despite the fact that it is generally expected that an emulation environment accelerate the verification process, other advantages of an emulation environment need to be considered. First, emulation is definitely effective to detect errors impossible to uncover during the simulation. For example, incomplete sensitivity lists in processes, inferred latch that cause hardware mal-function, uninitialized signals, etc. The Authors consider that the current state of the proposed emulation framework is useful to debug several DUVs, simply by the fact that verification takes place directly in hardware.

TABLE I. SIMULATION AND EMULATION TIME FOR A 2X2 MESH NETWORK-ON-CHIP.

Evaluation time (µs)	Simulation (s)	Emulation (s)	Speedup
50	6.7600	0.6936	9.7463
150	6.7670	1.4703	4.6025
300	6.8310	2.6670	2.5613
600	6.8540	4.8353	1.4175
1200	7.1830	11.7243	0.6127

VI. CONCLUSIONS

This work described a new emulation framework for the verification of digital systems in FPGAs. The prototype of the system is operational and provides enhanced controllability and observability figures compared to traditional approaches to FPGA emulation, like Xilinx Chipscope. The framework enables step by step (i.e. clock-by-clock) emulation and results are available immediately in the host control software, right after hardware execution of steps. The first prototype of the system has as main limitations: (*i*) the support to a single clock domain for the whole system; and (*ii*) low performance. These drawbacks were expected and derived from the initial framework specification, deemed as a proof of concept implementation. The framework is currently evolving to support any number of clock domains, which will become the end of the road to a truly generic framework to tackle the emulation of digital systems. Further work comprises the enhancement of the host-DUV interface to provide enhanced emulation speeds, by removing the limitation of sending a single set of input stimuli to the DUV at a time. The system is open-source and is available to non-profit use.

REFERENCES

[1] Mentor Graphics. *Veloce2*. Available at: http://www.mentor.com/products/fv/emulation-systems/veloce.

[2] Cadence. *System Design and Verification*. Available at: http://www.cadence.com/products/sd/pages/default.aspx.

[3] Xilinx. *ChipScope Pro and the Serial I/O Toolkit*. Available at: http://www.xilinx.com/tools/cspro.htm.

[4] Xilinx. *ISE Simulator (ISim)*. Available at: http://www.xilinx.com/tools/isim.htm.

[5] Chuang, C-L; Cheng, W-H.; Liu, C-N.; Lu, D-J. *Hybrid Approach to Faster Functional Verification with Full Visibility*. IEEE Design & Test of Computers, vol. 24(2), 2007, pp. 154-162.

[6] Tan, J.; Fresse, V.; Rousseau, F. *Generation of emulation platforms for NoC exploration on FPGA*. In: RSP, 2011, pp. 186-192.