

IP Validation for FPGAs using Hardware Object Technology™

Steve Casselman, John Schewel and Christophe Beaumont
Virtual Computer Corporation
6925 Canby Ave #103
Reseda, California, USA 91335
Email: scj|jas|cb|@vcc.com

Abstract

Although verification and simulation tools are always improving, the results they provide remain hard to analyze and interpret. On one hand, verification sticks to the functional description of the circuit, with no timing consideration. On the other hand, simulation runs mainly on subsets of the entire input domain. Furthermore, these tools provide results in a format (e.g. state graphs, bit vectors or signal waves) that remain disconnected from the real output of the application.

We introduce in this paper the process of validation applied to digital designs in FPGAs. It allows the designer the ability to test his/her implementation using the real data of the application and providing real results. With such real data, it becomes easier to identify where the error occurs and then to understand it.

1: Introduction

In the standard design flow used for implementing IP into FPGAs, designers use mainly verification and simulation tools to check their circuits. These tools are always getting more powerful, but they remain generic: the designer has to translate the real input data to a particular format and then interpret results which are not directly in the real output format (e.g. video). Such translations often make the understanding of errors more difficult. One easier way is to feed an operator with the real data it should receive and check if the resulting output matches the expected values. If not, wrong results may be directly interpreted without translation and thus without losing important debugging information. The designer validates his design and implements it with real data from the circuit. Having the real results helps in verifying the design and, once verified, in fine-tuning the implemented circuit.

In the first section, we present the basic design flow used for designs to be implemented into FPGAs. We then introduce validation and the role it takes in this basic flow. In this paper, we will use an implementation of the

Mandelbrot set generator to illustrate the introduced validation concept and its use in error tracking and performance tuning. Section 3 first describes the fractal operator and the choices we made for its implementation in hardware. Then two uses of validation are exposed: help for verification and a tool for precise performance measurement. We then concluded with a summary of these first applications and further directions to be investigated.

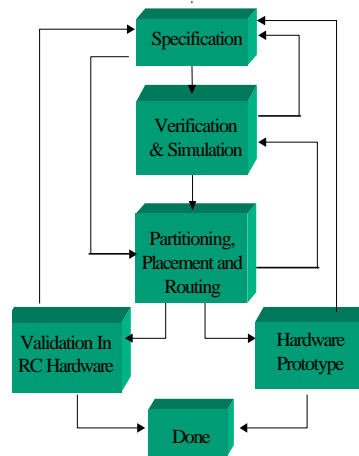


Figure-1 - Design Flow

2: Basic FPGA Design Flow

The most costly pitfalls encountered are the ones that have not been covered by simulation or not detected when analyzing simulation outputs. The principle we introduce is Validation. By this, we mean using real application data to validate the synthesized design. This will provide results in a more understandable form (i.e., the same as the real output provided by the operator). With such a representation of the data, which sticks to the currently developed circuit, analyzing output is much easier than with standard tools. Given real input data, the corresponding real result is calculated with the actual circuit. There is no need to convert input and output to particular formats.

Reconfigurable technology allows the designer to rapidly generate designs at very low cost. These designs may easily be downloaded to the component and exercised to check their behavior. Having a predefined interface helps the designer in focusing only on the operational part of the circuit. Once described and synthesized with the standard design flow, the real circuit is tested with some (or all, depending of the covered data space) of the input data and the output may be then compared to known values.

Let us consider a very basic example to illustrate this. If we want to check if an adder meets some timing constraints, using a timed simulation will require many processor cycles (even with a high level model) for each set of inputs. Validating the design into an FPGA will simply require some tens to a few hundreds of nanoseconds (depending on the circuit frequency). Based on such a simple example, validation provides a speed-up that may range from 10s to 1,000.

3: Uses of Validation

We will use in this section the fractal function computing the well-known Mandelbrot set. This section will briefly introduce some implementation choices we made. Two main uses we have identified for validation in the digital circuit design process are then presented: the first one illustrates the ease in debugging and the other the accurate information it provides for timing and optimization.

3.1: Mandelbrot set definition

Named after Benoit Mandelbrot, the Mandelbrot set is one of the most famous fractals in existence. It was born when Mandelbrot was playing with the simple quadratic equation $z=z^2+c$. In this equation, both z and c are complex numbers. In other words, the Mandelbrot set is the set of all complex c such that $z=z^2+c$ does not diverge.

To generate the Mandelbrot set graphically, the computer screen becomes the complex plane. Each point on the plain is tested into the $z=z^2+c$. If the iterated z stayed within a given boundary forever, convergence, the point is inside the set and the point is plotted black. If the

iteration went of control, divergence, the point was plotted in a color with respect to how quickly it escaped. When testing a point in a plane to see if it is part of the set, the initial value of z is always, zero. This is so because zero is the critical point of the equation used to generate the set. For the most part, critical points are a subject better left to a mathematical course. However, they have an application in chaos, and fractals in particular. Critical points are used as the starting value for specific variables in a function while calculating fractal sets. For example, in the Mandelbrot set, the initial value of z in the function $z=z^2+c$, is always zero because zero is the critical point of the function.

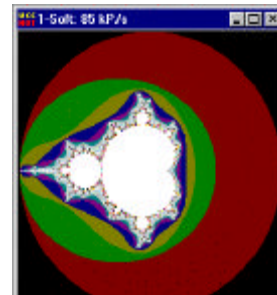


Figure 2. Graphical output of the Mandelbrot set fractal

3.2: Implementation Choices

For each point of the complex plan, the function involves 3 products and 4 additions for each iteration of the function. The calculated function is defined for each point using the C-source code fragment in Figure 3. Integration issues led us to implement these operators on 24 bits with a fixed-point representation:

- 1 bit for the overflow indicator,
- 1 for the sign,
- 2 bits for the integer part of the coordinate,
- the remaining 20 bits for the decimal part (leading to a $2^{-20}=10^{-6}$ precision. Pixelization appears when zooming in, reflecting the lack of accuracy).

The result of the function evaluation is an 8-bit unsigned integer for each point. It represents the drawing color for the corresponding point on the screen. See Figure 2.

```

for (i=0;
(i<=m_iterationCount)&&(distance<m_bailout);
i++)
{
rePart=Zre+x;
imPart=Zim+y;
xSquare=rePart*rePart;
ySquare=imPart*imPart;
Zre=xSquare-ySquare;
Zim=2*rePart*imPart;
distance=xSquare+ySquare;
}
return (byte)(i-1);

```

Figure 3 - C Code for Mandelbrot

With our current implementation of the function into the programmable logic, a single operator fits into the Xilinx Spartan XCS40 and up to 4 operators into the larger Xilinx XC4062 FPGA.

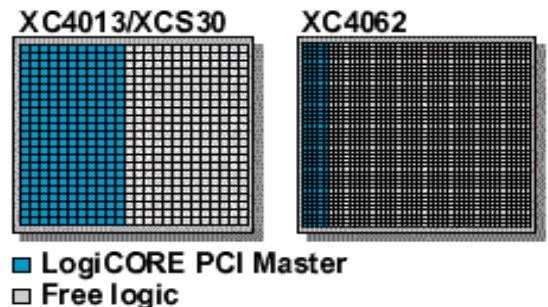


Figure 4 - PCI Core in FPGA

3.3: Validation Reducing Verification

The verification process remains quite easy in the basic flow when the circuit completely crashes and does not give the right result for any value. Nevertheless, it becomes very hard to track every single error if they only appear for inputs with a particular property. Then it may be difficult to define what is responsible for the error (and thus to correct it).

Offering to the designer a way of seeing its design crash with the set of real input data will help him understand what is really happening in the designed circuit.

Figure 4 gives outputs, which expose different errors: in the first one (Fig. 4.a), the output shows an almost working circuit with erroneous values following a particular shape. Having such a pattern led us to consider

an error affecting inputs and results with some particular bit configuration. This error was generated by the wrong management of the overflow bit. In the figure 3.b, one notices that the output has vertical stripes, but a “globally correct” output. The design is replicated into the circuit synthesized and 2 output points are computed at once. One can conclude the design seems to be right but that one of the operators is not operating or is not properly connected to the input/output registers.

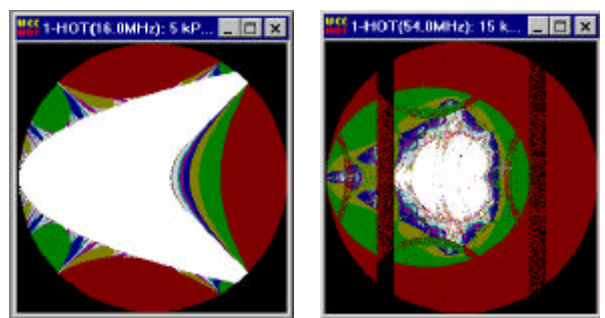


Figure 4a 4b. Outputs of Incorrect Circuits

3.4: Validation for performance tuning

An information difficult to extract from simulation results is the actual and accurate timing of a design. Some delays may be critical in all cases, but some may have only little influence on the overall implementation. Validation will give the user a unique way of timing its circuit: by using it, modifying the clock frequency and seeing the circuit’s results. From a slow frequency at which the circuit operates normally, the frequency is increased until the operator no longer gives the right output. Viewing the operator crash at a certain clock speed and for a certain values gives much more information than the concatenated list of all signal delays. It then becomes easy to optimize and fine tune a design as one can focus on the slow signal.

With the Mandelbrot example used, we could tune the clock frequency to more than 70MHz after 3 successive optimizations (first timings only allowed about 20MHz). More improvement was possible at the cost of an increasing number of errors (compared to software or a slower hardware implementation). If the timing deadline is met before reaching maximum possible frequency, there is then no more need to focus on placement and/or routing. Further optimization may aim to improve the spatial aspect of partitioning (i.e., to reduce the number of used logic elements or to get a more compact design). See Figure 5a @54MHz & 5b @70MHz.

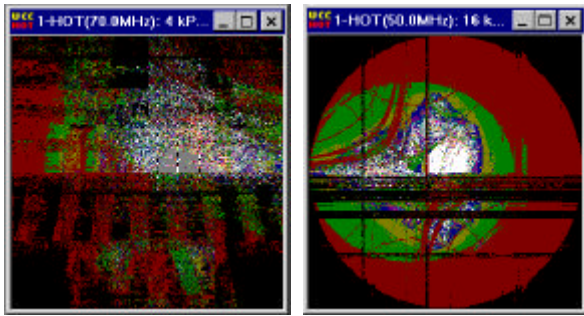


Figure 5a 5b. Over-clocked Mandelbrot circuit

A 70MHz implementation with 2 operators outperforms by 10 to 20% a mid-sized Pentium PC. As no data dependency exists between the computation of any two points, the implementation of 4 operators will at least double these performances (some overhead is then saved as writing a word is less expensive than writing 4 bytes).

If more improved performances are required, one may consider an implementation where the circuit will store all the results in the board memory and transfer them back in burst mode to the main memory of the system. The implementation is totally different and such an optimization, which is beyond the scope of the paper, has to be done at the description level (i.e., the beginning of the entire design flow from figure 1).

The H.O.T. II Development System is a powerful tool for evaluation, customizing and prototyping designs using the Xilinx LogiCORE PCI Interface. It is ideal for validating IP products. The PCI based board includes a bus controller, a compute element (FPGA), on board programmable clock and other system features. The combination of these features enables System Level Validation of both hardware and software components of your design. VCC's unique Run-Time Reconfiguration Programming Method to configure the FPGA on the fly (without re-booting), enables running your designs with real data. See Figure 6.

4.1: Functional Description

On power up the FPGA boots up with the H.O.T. II Interface. The H.O.T. II Interface contains the Xilinx PCI LogiCORE Interface Macro and a VCC custom backed that lets users communicate with two fully independent 32-bit banks of RAM and the Configuration Cache Manager (CCM). The CCM controls the Run-Time Reconfiguration (RTR) and reload behavior of the H.O.T. II System. The CCM can configure the FPGA from two on board sources, the Configuration Flash and the Configuration RAM. During reconfiguration, access to the board is disabled by the driver.

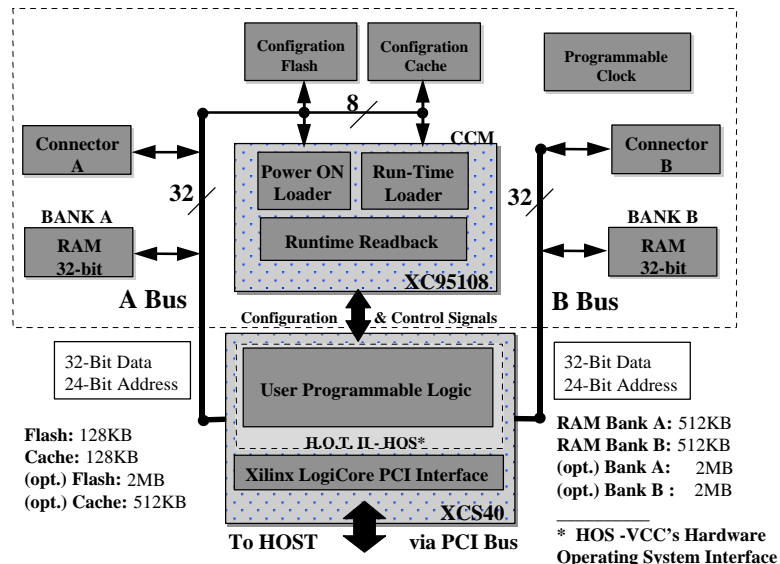


Figure 6 -- H.O.T. II -- Block Diagram

4: H.O.T. II -- IP Validation Platform

When the FPGA comes back on-line, it signals the driver, which reloads the PCI Header information into the LogiCORE PCI Core. A 128KB Configuration Cache RAM can hold 3 XCS40 configurations.

The H.O.T. II Configuration Cache Manager (CCM) allows you to place your hardware design into an application software program and dynamically download the design at program execution time. With the H.O.T. II Development System, you have *Hardware on Demand*™. The user can load the Configuration RAM over the PCI Bus. Easy control and loading of the configurations (IP) is made possible through VCC's Hardware Object Technology-API. The design bitstream is converted into an encrypted program element to be downloaded into the H.O.T. II Board via application program commands. .

The H.O.T. II PCI board has two independent buses, each with 32-bit data and 24-bit address. There is an I/O connector for each of these two buses for daughter boards (HOT2_PDC is a prototyping daughter card available for the HOT II).

4.2: Hardware Components

Features

1. H.O.T. II Standard Version {HOT2}
Ideal for LogiCORE PCI Interface Development
 - PCI Compliant - LogiCORE™ PCI Target & Initiator Interface
 - Single Xilinx Spartan XCS40-4
 - Single Xilinx XC95108-15
 - 1MB of fast SRAM organized as 2 Independent Banks of 32-bit RAM
 - Configuration Flash 128KB
 - Configuration RAM Cache 128KB
 - Programmable Clock Generator Module (360KHz to 100 MHz)
 - Mezzanine Connectors for daughter cards
 - Security Jumper
 - LED's for DONE, 5v and 3.3v
 - Universal 3.3v or 5v PCI board
 - 3 Split Power Planes & 1 Ground Plane
 - 4 Signal Layers
 - Download/Cable98 Module

2. H.O.T. II Expanded Version {HOT2-XL} for CORE Development
 - Single Xilinx XC4062XLT-1 (replaces XCS40-4)
 - 4 MB of fast SRAM organized as 2 Independent Banks of 32-bit RAM
 - Configuration Flash 2MB
 - Configuration RAM Cache 512KB
 - Other features same as Standard Version

4.3: Software Components

With the use of a High Level Hardware Description Languages (HDLs), Hardware Object Technology (H.O.T.) and the HOT II PCI board, the engineer can begin to use Run-Time-Reconfiguration techniques for Validation. The integration of drivers, API and C++ functions in the HOT II DS makes testing designs in Real-Time using Real Data by configuring hardware from executable programs possible.

I. Design Entry -- Use Standard VHDL/Schematic Tools.

II. Design Implementation -- Use Xilinx Foundation or Alliance Software.

III. Make Bitstream -- Use Xilinx Foundation or Alliance Software.

IV. Convert Bitstream File into Run Time Program Mode. -- The Development System supports two methods for Run-Time Reconfiguration. Both require C++ routines for control of the Board. The first method loads the FPGA from a file on the hard disk via program control. The other method compiles the Hardware Object directly into the executable program, loading the FPGA during application run-time. The necessary API and programming routines are included on the Development System CD.

5: The Application Interface

The H.O.T. II Interface includes the PCI/Target as well as the logic for managing the HOT2 board features. The user connects his/her backend to the HOT II Macro (see Figure 7). All handshaking with the host is automatically handled. The example below shows a simple data register circuit.

6: Conclusions

We have shown in this paper a new supplemental way of verifying the correctness and effectiveness of hardware design. Validation allows the designer to check his design with real input data. As a first advantage, we exposed the resulting improvement over timing simulation. Furthermore, as the operator gets real data as inputs, it will provide result in the same format as the output. Instead of analyzing bit vectors or state graphs, the design can compare directly the output to some expected known results. This comparison is much easier and will help to focus on the exact problem.

This method makes fine-tuning and timing of the circuit extremely easy and straightforward: increase the frequency until the circuit no longer operates correctly. More applications of validation are still to be investigated. We are currently working on a solution to automate as much as possible of the concept (which still requires a lot of user interaction). Another interesting issue is that applications may be less graphical than the exposed example. In such a situation, an immediate visualization of the errors is then more difficult to obtain. A solution may consist in providing different pre-defined graphs with a simple interface (an operator with one input and one output can be represented by different kinds of planar curves,).

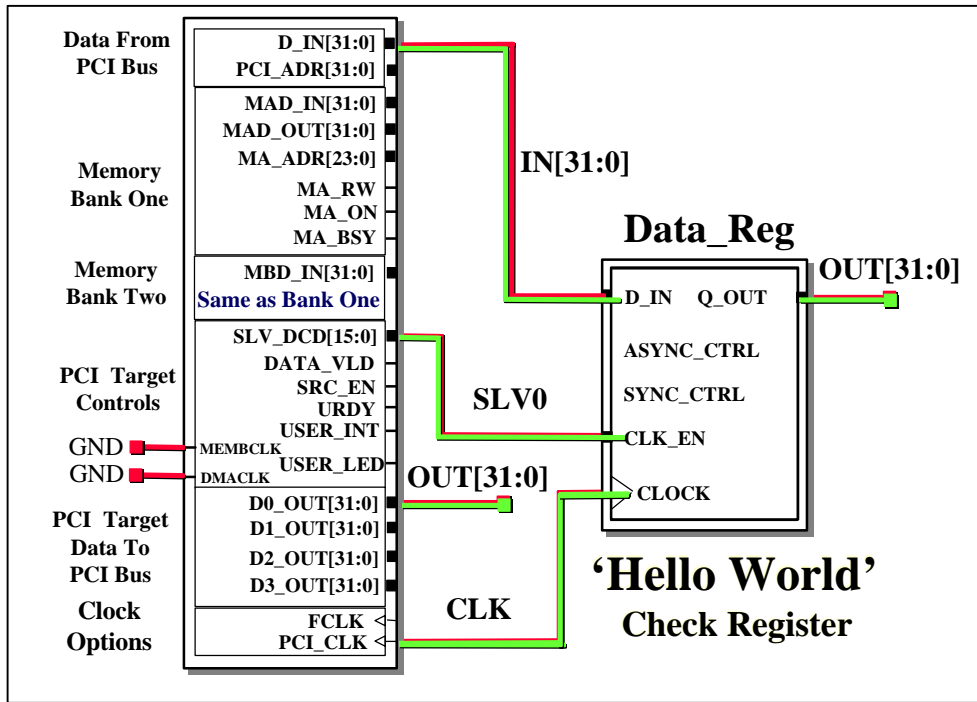


Figure 7 - HOT II Interface