european business press

www.electronics-eetimes.com

June 2012

EE Times europe

For the creators of technology

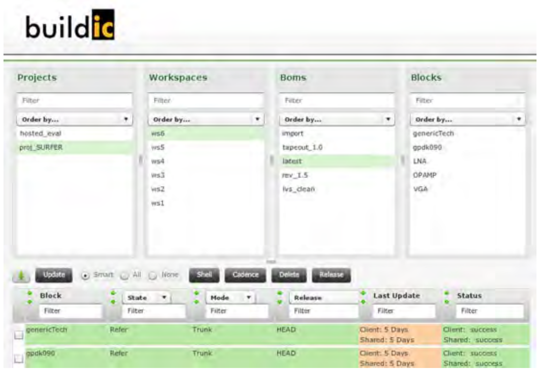
Flexible organic transistors find use in medical implants

Special focus:
EDA & Coding Tools

Special focus:
Sensors & Data Conversion

many customer re-spins. A coherent mechanism for managing configurations is required, allowing designers to revert to a previous configuration at any point in the project lifecycle.

Providing a local workspace assembly client to easily create and maintain configured workspaces based on a defined set of IP versions eases the task of configuration management. In such a system designers are typically given a default BOM to assemble their SoC workspaces.



Block	State	Mode	Release	Last Update	Status
Filter	Filter	Filter	Filter	Filter	Filter
genericTech	Refer	Trunk	HEAD	Client: 5 Days Shared: 5 Days	Client: success Shared: success
gpdk090	Refer	Trunk	HEAD	Client: 5 Days Shared: 5 Days	Client: success Shared: success

Fig. 5: SoC Developer workspace assembly.

The default BOMs are controlled centrally by the project lead and the delegated subsystem owners. For example, PDK entries can be set in the project BOM as a hierarchy include, and the CAD person responsible for that PDK provides the correct BOM definition. Users no longer have to make decisions about which versions of which IPs and their libraries to use - these are configured centrally and the risk of them choosing the wrong version is diminished. Stale (out of date) IP versions can be retired in the BOM management interface and users precluded from assembling their workspaces with these 'old' versions. The correct versions are suggested as part of the assembly process, and BOM owners are forced to update to the correct versions if they intend to make a subsystem release.

IP developers can go "off grid" from the set of IP versions defined in their subsystem BOM when required, but the assembly environment makes it clear they are out of sync with the project BOM. Developers can however derive BOMs from the state of the workspaces and send them back to the centralized cockpit for use by other designers on the project.

Tracking IP usage and quality

The proliferation of IP use introduces issues related to the volume and frequency of use of any particular IP, as well as its quality in relation to the overall design spec. A system that supports the management of the IP release and promotion methodology at the client level is needed. Each IP release and its project and subsystem context should be tracked in an SQL database and, over time, the entire IP fabric of the company can be mapped, including all internally generated IP and external IP purchased.

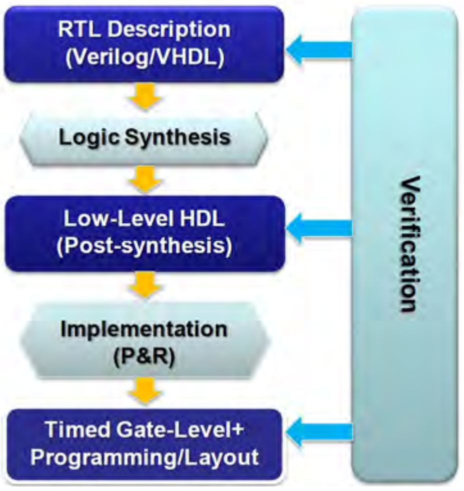
With each release quality measurements, regression results, test coverage, power analysis calculations and other important metrics are collected. This is stored in the SQL database and users can access reporting results via web. Users can measure the quality and progress of the design for each release configuration and this is kept up to date automatically. This lends itself well to IP audits, hierarchical quality checks, cost analysis and other import measurements. With such a system users have access to a real time stream of project IP metrics in an easily configured central portal.

FPGA-based SoC verification challenges

By Ian Gibbins

AS FPGA DESIGNS BECOME increasingly complex, a clear synergy emerges with that of the ASIC (design) world and, more specifically, creating a System-on Chip (SoC).

For example, consider Xilinx's recently announced Zynq-7000 Extensible Processing Platform (EPP). It features an ARM dual-core Cortex-A9 MPCore which has numerous peripherals including memory controllers, CAN, USB, Tri-mode Gigabit Ethernet, SD-SDIO, UARTs and ADCs. Xilinx boasts though that the real advantage (over say a two-chip processor and FPGA solution) is the nine-AXI interfaces and control signals (e.g. DMA and interrupts) between the ARM core and the programmable logic fabric (itself a considerable amount of 'FPGA white space'). Also, there is the ever-present requirement of making increased and better reuse of in-house developed IP. It is not surprising therefore that many FPGA designers are finding themselves no longer working at the chip level - because the chip has now become a system – and this has required making a significant (but not too daunting) adjustment to the design flow. For example, verification now means dealing with different levels of design abstraction; multiple languages used for design description; and assorted verification methodologies. Of these, embracing the verification methodologies are perhaps the most important – yet seem to be overlooked by many FPGA designers (see 'Survey Results'). Some methodologies have their roots in the world of ASIC SoC design, others are more associated with software and embedded system design. As such, they will be mature and have already proven their worth in reducing the amount of time required for design verification – and are therefore well suited to complex FPGA designs.


RTL Description (Verilog/VHDL)
Logic Synthesis
Low-Level HDL (Post-synthesis)
Implementation (P&R)
Timed Gate-Level+ Programming/Layout
Verification

Levels & languages

While a traditional RTL Design Flow could survive with just one language (VHDL or Verilog), a System Level Design Flow requires languages that can handle significantly higher levels of abstraction. Two popular languages we should mention here are:

- SystemC, which is governed by IEEE Std 1666-2005 and is a library of C++ classes and templates intended for system and hardware descriptions at medium to high levels of abstraction. The C++ origins of SystemC make it very popular among system designers, and using SystemC makes the transition from Algorithm to Architecture very easy.

However, a drawback of SystemC is its lack of native support for Assertion Based Verification (ABV – see later).

Ian Gibbins is an Applications Engineer with Aldec Europe
www.aldec.com

• SystemVerilog is governed by the standard IEEE Std 1800-2009 and is the evolution of earlier Verilog HDL. The main strengths of SystemVerilog are: enhanced RTL coding; its extensive support for ABV; being able to implement Object Oriented Programming (OOP); and the availability of a Direct Programming Interface (DPI) to C/C++.

Functional verification

Over the years the EDA industry has spawned a diverse range of verification techniques such as Design Rule Checking (DRC), Static Timing Analysis (STA), Formal Verification and Simulation. Verification can, and should, be implemented as soon as possible after design capture. DRC, for example, provides an effective and early 'sanity check'. It analyses the design sources, checking if they follow the rules of good design practice; and DRC tools are usually equipped with multiple sets of rules which can be fine-tuned. As a result, you don't have to waste time finding obvious mistakes and can direct your attention to more exacting problems.

STA is a common replacement for timing simulation. Essentially you specify critical paths and timing requirements, and the verification tool lists the paths that meet the requirements and flags those that fail. However, the results from STA only make sense if the correct functionality was verified earlier and that the constraints specified were realistic.

Formal Verification is considered the most scientific of the 'static' verification methods (which include DRC and STA). It uses formal methods to transform the design down to a state machine, a network of processes or similar fundamental form. It then performs checks depending on the class of tool. For example, a Model Checker verifies whether design properties still hold in the transformed design. An Equivalence Checker on the other hand verifies whether the transformed design has the same functionality as the reference design; noting here that your reference design must itself be validated using one of the other verification techniques.

Simulation on the other hand is a dynamic form of verification. It relies on feeding stimulus to the design model (created inside a simulator) and observing the outputs produced. The most popular variety of simulation is event-driven, used for VHDL and Verilog design descriptions. The precise modelling of every single signal change is the source of high accuracy during event-driven simulation but, unfortunately, it slows down the verification of large designs.

One way of addressing this is to implement cycle-based simulation and reporting signal changes not when they happen but rather when the nearest sampling event (e.g. a clock edge) arrives. Simulation can be enhanced by collecting additional data during verification, an approach which has led to some of the newer complementary verification methodologies detailed below.

Methodologies

There are many verification methodologies used in digital design, with two popular being: Assertion Based Verification (ABV) and Open Verification Methodology (OVM).

ABV uses formalised versions of design properties (expressed in dedicated languages instead of plain English) to create an additional layer of verification. Whilst regular HDL code has to be written with hardware implementation in mind, for ABV assertions constantly express the original design intent (i.e. the desired design functionality). OVM on the other hand is more of a generic term, meaning a library of pre-packaged advanced verification blocks (i.e. procedures for stimulus generation, data collection and control of verification). OVM packages typically include Constrained Random Stimulus Generation (CRSG) and Functional Coverage (FC), which are difficult to master and implement if you are a hardware designer and unfamiliar with the more arcane aspects of SystemVerilog or SystemC.

Engineers already using OVM will testify that it improves testbench reuse and makes verification code more portable. However, the close association with SystemVerilog and SystemC often has engineers thinking that the CRSG and FC can only be done in those languages. A recent development on this front has been the launch of the Open Source - VHDL Verification Methodology - www.osvvm.org - which provides advanced features to engineers designing ASICs and FPGA-based applications using VHDL.

If you are a seasoned FPGA designer embracing system-level design for the first time, the above may seem a little daunting. Dealing with the different levels of design abstraction, the multiple languages that may be used for design description and the assorted verification methodologies available – all have, in many respects, complicated your engineering processes. Conversely though, all can be handled by established EDA tools and techniques that have already proven their worth for ASIC design and embedded system development.

SEGGER

1.4 MBytes/sec Download

Emulator (J-Link™)

ARM/Cortex™ and Renesas RX

- Fastest Emulator Available
- Extremely Fast Flash Download
- Intelligence Inside the Emulator for Robust Communication
- Multicore Debugging Support
- Supported by All Popular IDEs Including CodeBench, CodeWarrior, HEW, IAR, Keil, Rowley, and GDB/RDI Compliant Environments

8.16.00 J-Link ULTRA

1.4 MBytes/sec Download Speed
The Fastest J-Link Debug Probe Yet!

The J-Link can be coupled with a number of available software modules to fit your application needs.

J-Flash is a standalone application used with the J-Link to program internal and external flash devices.

J-Link Flash Breakpoint permits you to set an unlimited number of software breakpoints while debugging in flash.

As the original manufacturer of the J-Link and its OEM derivatives, we are happy to inform you that this software also supports the DIGI® JTAG Link, Atmel® SAM-ICE, and IAR® J-Link KS.

49.98€

Special Offer

8.08.90 J-Link EDU
Educational Use J-Link
Includes Flash Breakpoints
Includes J-Link GDB Server
www.segger-us.com/edu.html

J-Link Educational Use (EDU) Bundle

www.segger.com