FPGA PROTOTYPING SOLUTIONS

Whitepaper

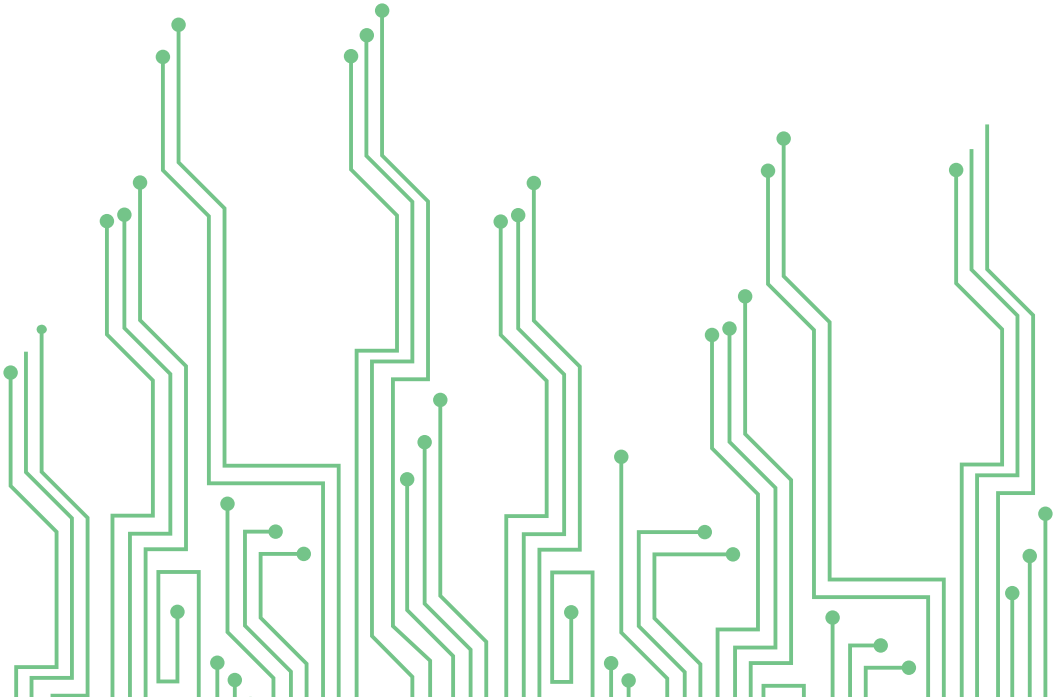
ProtoBridge™

Accelerate Design Verification, Architecture Exploration and Software Development with High-bandwidth PC-to-DUT Connectivity

Author:
Ashok Kulkarni

Release date:
August 2022

Contents



Introduction to ProtoBridge

While FPGA-based prototypes today are predominantly used at late design stages such as system validations and software developments, solutions for earlier design stages are far and few. ProtoBridge AXI from S2C enables designers to maximize the benefits of FPGA-based prototypes early in the design project for algorithm validation, IP design, simulation accelerations, and corner case testing by providing a high bandwidth datapath between software running on a workstation and the FPGA prototype.

ProtoBridge AXI consists of a computer software component and an FPGA design component. The computer software component contains Linux/Windows drivers and a set of C-API/DPI routines to perform AXI transactions. The FPGA design component contains a PCIe core implemented as an end-point device, an interconnection module, and AXI transactors to be instantiated in users' design-under-test (DUT). With these enhanced product features, users can read and write at speeds of up to gigabytes per second through the PCIe Gen3 interface, connect 8 Master devices and 8 Slave devices on the AXI bus, and take advantage of the patent-pending Shared Memory technology that link the FPGA prototype with third-party design tools.

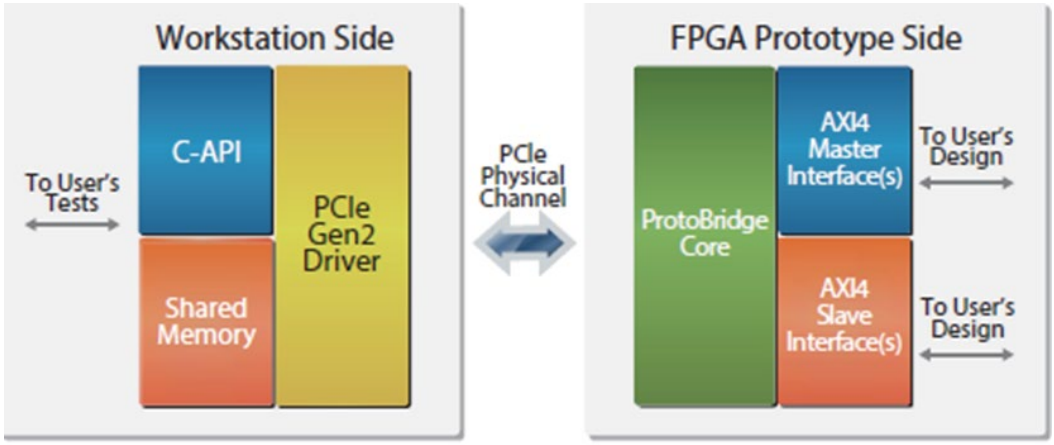


Figure 1. Architecture of ProtoBridge AXI

Before we delve into Proto-Bridge AXI (hereafter referred to simply as ProtoBridge) let's review why we prototype and the benefits of it.

Benefits of Prototyping

The relentless pace in the advancement of silicon process technology, now at sub-10 nanometers, has made it feasible to implement large and highly complex designs as SoC/ASIC/ASSP with several billion transistors [1] that operate in the GHz frequency range. These chips cater to a myriad of applications both in consumer space and commercial usage. Furthermore, many consumer gadgets have a short life cycle (some as low as 3 to 6 months) and are often differentiated by the software content that runs on them. [2][3]

There are two primary requirements for a successful launch of these devices. First, the design must undergo thorough verification before the design tape-out. Second, an affordable hardware platform with high perfor

mance, representing the SoC prototype, for software development must be available before the first batch of silicon.

Silicon re-spin is not an option anymore as the mask cost alone runs into tens of millions of dollars. The lost market opportunity due to delayed product introduction may mean risking the business viability altogether.

Due to ever declining time-to-market (TTM) pressure, chip design teams constantly strive hard to minimize verification costs and avoid schedule slippage. The 2020 Wilson Report found that only 32 percent of today's chip design projects can achieve first silicon success, and 68 percent of IC/ASIC projects were behind schedule. [4]

Various verification strategies such as simulation, acceleration, prototyping, and emulation have been discussed in the papers 'Verifying Complex ASICs through FPGA-based prototyping' [2] and 'Addressing the new challenges of ASIC/SoC prototyping with FPGAs' [3]. Other techniques like formal analysis and virtual prototyping can also play an essential role in specific applications. The papers [2][3] compare the pros and cons of each strategy. FPGA-based prototyping offers some unique benefits in terms of

- Cost (5 to 8 times lower than emulator)
- Performance (~10-60 MHz) - software engineers can run diagnostic, firmware & application software
- Interface with real-time devices that require the DUT (device under test) in FPGA to run a few MHz to several tens of MHz
- Hardware/software co-verification capability
- Replicates – enables multiple engineers and potential customers to verify designs before the arrival of silicon
- Ability to quickly port the flow to prototype systems with the latest FPGAs

Scalability, reliability, access to reliable models and the ability to iterate quickly and seamlessly are all critical and essential in minimizing total verification and pre-silicon software bring-up time. Automated partitioning of large designs across multiple FPGAs is also an increasingly necessary feature. S2C's PlayerPro Compile addresses the partitioning challenge by providing the designer with an easy-to-use GUI to guide the tool for optimal results. Furthermore, the ability to efficiently debug hardware problems on your FPGA platform is critical to uncover bugs deeply buried inside the design or corner cases that get exposed only when the design interfaces with real-time devices or when actual firmware or application is running. This again is addressed by S2C's PlayerPro 'Multi-FPGA Debug Module' (MDM) software and hardware solution. Both 'FPGA partitioning' and 'MDM' for debug solutions are beyond the scope of this paper. If you wish to learn more about these solutions, please go to www.s2cinc.com.

While the benefits of FPGA-based Prototyping have been well established [2],[3],[4], verifying specific designs could be challenging because of the design environment, lack of RTL models, or the need to have a large amount of data available for processing instantaneously and rapidly moving the processed data to and from the memory buffer. S2C's ProtoBridge is one such solution that precisely addresses these challenges. This

paper describes some of the limitations of building a hardware verification environment. Equally important is understanding the unique requirements for an FPGA platform so that the software development team can adopt it. We then focus on solutions from S2C to address these challenges.

FPGA-based Prototyping - Key Design Challenges and Considerations

- Explore different algorithms by mapping them to FPGA and make early architectural design decisions before implementation
- Accelerate prototyping performance - Replace RTL bus-functional model with ESL transaction-level model to accelerate the verification pattern development and execution
- Address non-availability of RTL models - Integrate Virtual and FPGA-based prototypes to partition design to maximize overall simulation performance
- Interface with real-time devices - Achieve high-performance execution of system-level models with real-world interface connectivity of hardware
- Reuse legacy design blocks that are in net-list format by replacing them with bus-functional models or complete re-writing of the block
- Generate and apply massive test patterns quickly to prototype design generated by standard test methodology flows such as OVM (Open Verification Methodology) UVM (Universal Verification Methodology.

In the following several sections, we will elaborate on the above challenges and considerations and see how S2C ProtoBridge helps address them.

Block-based Prototyping and 3rd Party IPs

It would be unrealistic to assume that you would be able to map the entire design, download bitstreams to multiple FPGAs platforms and that it will work. The best strategy is to divide and conquer. So, consider bringing up subsystems of the entire design, block by block, which will allow quick identification of both design issues in a sub-block and any problems related to mapping the design to the FPGA(s). Block-based prototyping works well, especially with designs with many legacies or 3rd party IPs that also need a lot of real-time testing and early software development.

And often, designers don’t have the RTL source code for the IP blocks from 3rd parties (for example, ARM processors) and therefore cannot directly map the IP to the FPGAs. Most often, IP providers have encrypted netlists or behavioral models. Once you have the encrypted netlist from the IP provider, you can synthesize and partition the entire design while treating that IP as a black box. If you specify the correct resources (ALM, LUT, registers, I/Os), the prototype compiles software should take those resources into account when partitioning to multiple FPGAs. You can then integrate the encrypted netlist during the place and route stage. Both

Xilinx and Altera have a well-documented methodology that shows how to incorporate a 3rd party encrypted netlist with your RTL design and stores the decryption key provided by the vendor in a battery-backed RAM (BBR) memory. S2C FPGA platforms support BBR implementation.

ESL and FPGA - Early Architecture Exploration and Algorithm Modeling

Electronic system level (ESL) design and verification is an electronic design methodology focused on modeling designs at higher levels of abstraction. The term Electronic System Level or ESL Design was first defined by Gartner Dataquest, an EDA-industry-analysis firm, in 2001 [6]. It is described in ESL Design and Verification [7] as: "the utilization of appropriate abstractions to increase comprehension about a system, and to enhance the probability of a successful implementation of functionality in a cost-effective manner."

The basic premise is to model the entire system behavior using a high-level language such as C, C++, or graphical "model-based" design tools, such as Mathworks 'M' or National Instruments LabView. So basically, you have abstract models that are used for pre-implementation on the one hand and cycle-accurate RTL representations written in Verilog/System Verilog or SystemC on the other. Newer languages are emerging that enable the creation of a model at an even higher level of abstraction, such as SysML, etc.

Often SoC-based designers model key system capabilities to ensure design specifications match vital parameters such as performance, power, silicon area - in terms of ASIC gates, and functionality – early in the development process. These design abstractions initially modeled at a high level of abstraction are implementation-independent but consider system-level dependencies. This allows them to verify selective and critical blocks of the design. ESL enables quick model iteration allowing multiple scenarios to explore, while the implementation details could be added later in the development process. Many key system parameters could be committed to by early architecture decisions, with little room to modify later in the implementation process. As design development matures, additional implementation details are added to the process as it converges to the final product specification. During this design maturation process, critical sections of the design may be modeled using familiar design verification tools that would not normally be expected to be deployed at a later stage in the development process – such as FPGA prototyping. Algorithm performance, compute-intensive modeling such as in Machine Learning, especially Deep Learning algorithms, video quality analysis, and network throughput are some of the use cases where FPGA prototype modeling can play a vital role in early architecture exploration when used in conjunction with ESL environment.

Thus, the importance of design exploration during early product definition is considered by some to be the most critical stage of any development project – in which optimizations of the architecture are the least costly to realize in time and effort. ESL Design allows system architects to explore ‘what-if’ scenarios with system partitioning and quickly evaluate different implementation alternatives – which design blocks get implemented in hardware and which ones should be implemented in software.

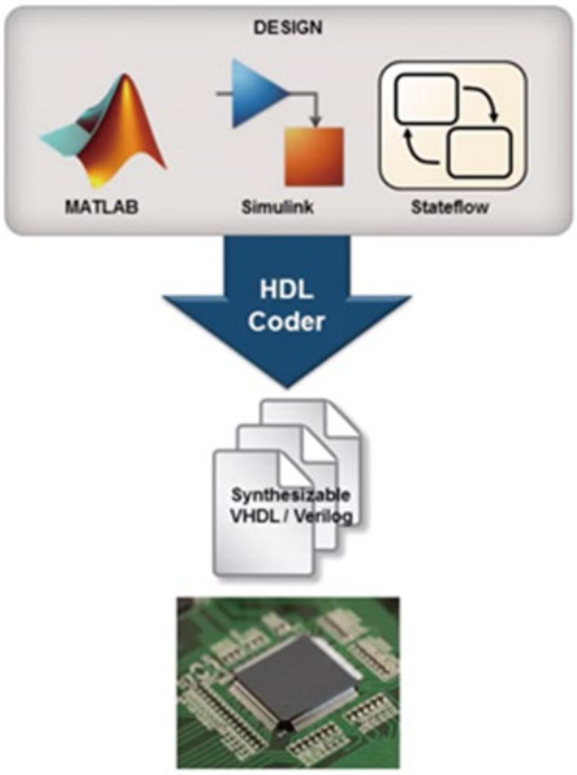


Figure 2. FPGA design using MATLAB, Simulink and HDL Coder

Mathworks' other tools like Simulink and HDL coder can help in migrating from a simulation-only environment to mapping compute-intensive portions, such as DSP blocks or Machine Learning/Deep learning models to the FPGA platform, thus creating a simulation-acceleration verification environment.

So, the mainstream hardware verification tools such as FPGA-based prototyping and emulators, to a lesser extent, together with transaction-level testing, are being deployed into the Design Exploration space for architecture optimization decisions. One can argue that ESL Design builds a verification continuum by simply bridging the classic verification space into the system level. Because the FPGA-based prototyping can run in tens of MHz, running actual boot code from SRAM and flash memory identical to eventual real-world devices is possible with the prototype design. This is not practical in an RTL simulation-only environment – no matter how fast the simulator runs. Typically they run in Hz or a few tens of Hz, and it may take several days or weeks to boot up the OS, which is not practical. Simulation-only performance is explained in detail in the paper [2].

FPGA-based prototypes due to their lower cost and relatively higher performance over emulators, are not only deployed for hardware verification but widely adopted for software development, where multiple replicates are needed at affordable price.

Legacy ASIC netlist and FPGA Implementation

There are instances when designers who want to do an FPGA implementation of their design are reusing some ancient blocks with only the ASIC netlist and without RTL code. Implementing these design blocks

becomes very difficult since the design details are unknown. Often these legacy designs may only be accompanied by a testbench. In this case, the best approach is to convert the ASIC gates to a behavioral model and eventually to an FPGA implementable design code and to use a co-simulation environment (such as S2C's ProtoBridge) to verify if the functionality of the block is correct before integrating it with the entire design. Unfortunately, this is still a painful process, so designers should consider either not using those legacy blocks or, better yet, re-writing them entirely.

Large Scale Test Generation and Fast Data Transfer

In-circuit testing is probably the most critical reason for prototyping the design. But in-circuit tests are usually based on 'Directed testing technique' [5] or 'Unconstrained random tests'. Directed Verification Technique with a set of directed tests is extremely time-consuming to generate and maintain for more complex designs to verify. Directed tests are developed based on the design specification and thus only cover scenarios that have been anticipated. More importantly, it fails to ensure complete test coverage. This can lead to costly silicon re-spins, leading to missing time to market, which can be a disaster for a business. Transactor level interface enables the generation of large suites of test cases in simulation in a constrained random testing environment. Constrained random testing is supported by test bench methodologies such as OVM/UVM to be run directly on the prototype making these tests available to the DUT on the FPGA platform. Furthermore, these tests can be easily extended to large data sets and complex test scenarios, providing coverage for corner cases and hard-to-find bugs. Adding a transactor-level interface to an FPGA-based prototype facilitates the development of new systems faster and with improved productivity. As behavioral models are introduced, architectures become refined and block functionality well defined. These blocks are eventually implemented as part of the new system. Blocks defined and implemented in RTL become IP for designs of successive generations, enabling the cycle of design reuse.

FPGA-based prototyping is well-suited for designs fully implemented in RTL that can be mapped to an FPGA. However, many designs may not be entirely mapped to an FPGA simply because some design blocks may only be available as behavioral models described in C, C++, VHDL or Verilog bus-functional models. In these cases, transaction-level interfaces play a critical role in bridging the abstraction level between behavioral and RTL models running on hardware. These transactors offer a way to communicate between behavioral models running on a host machine and an FPGA-based prototyping platform that often includes memories – including boot ROM, DSP blocks, processors, and high-speed interfaces.

S2C's unique patent-pending Prodigy™ ProtoBridge System is a solution that allows for just this type of high-speed communication. ProtoBridge supplies a transactor interface between a software program, representing a behavioral model and user design implemented as AXI-compliant (ARM's Advanced eXtensible Interface) hardware. There are two critical parts: an AXI-to-PCIe bridge connecting to a host computer and a C-API communicating to the design through the bridge. The software to-AXI transactor communicates through a PCIe interface supporting transfer speeds up to 4000 Mbytes/sec, providing a perfect development platform. This can be extremely useful for data-intensive applications that need to transfer large amounts of data in the

form of video images or clips. The data patterns may reside on the host PC, which can be transferred to the memory - either BRAM of the FPGA or DDR4 on the FPGA-based platform.

Let's look deeper into the S2C ProtoBridge solution.

ProtoBridge Architecture

The ProtoBridge system consists of two parts: software and hardware. There are C programs, a C-API library, and a PCIe device driver in the software part. They are all installed on the host machine. In the hardware part, we adopt the Prodigy Logic System produced by S2C. The Logic System is logically designed by users, PCIe IP, and a core controller connecting to form an FPGA project. After the project is downloaded to the platform, it can be connected to the software. The PCIe hardware suite makes the connection.

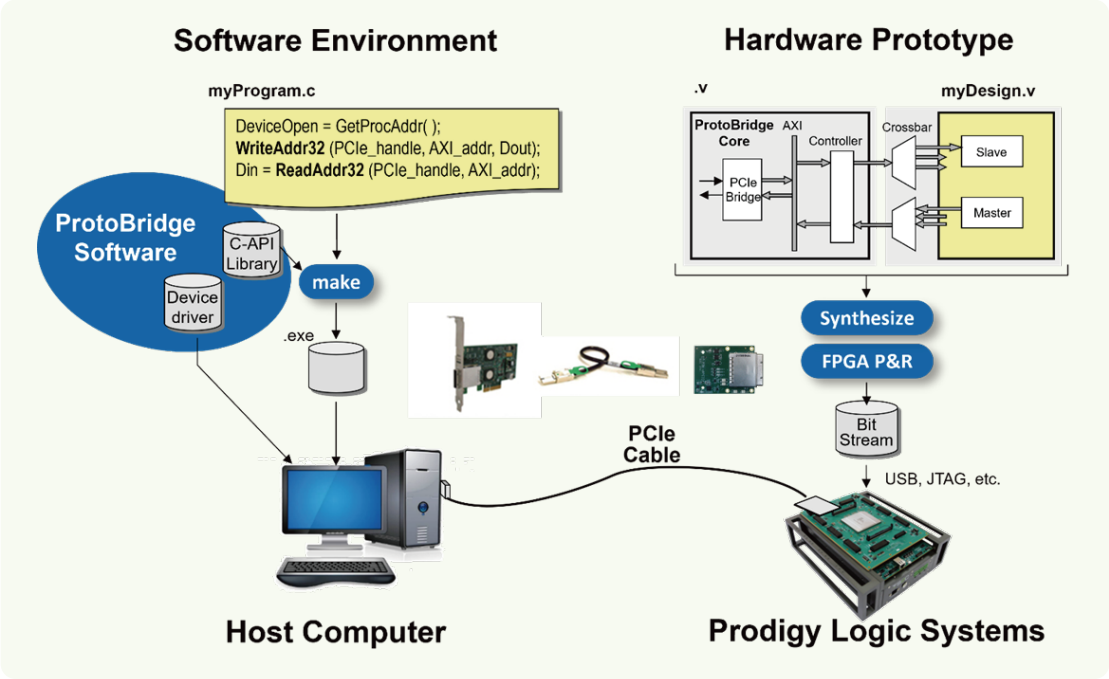


Figure 3. Overview of S2C Prodigy ProtoBridge

As shown in Figure 4, the ProtoBridge system is in the middle of the whole system, connecting software and hardware. In terms of software, there are C programs, while in hardware, there is user-designed logic. PB system includes C-API, PB control core, and interconnecting hardware equipment, including two PCIe adapter cards and one PCIe cable.

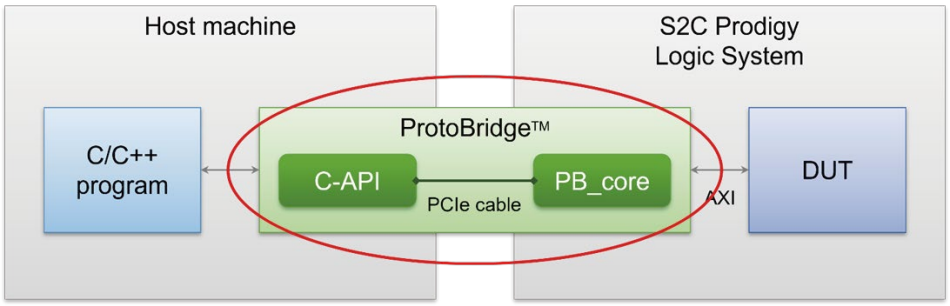


Figure 4. S2C Prodigy ProtoBridge is the key to integrate Software and Hardware

Referring Figure 5 below, we see that,

Hardware side contains

- PCIe Core and Interconnection Module
- 8 AXI-4 Master and 8 AXI-4 Slave devices for users to instantiate

Software side contains

- Corresponding PCIe Gen3 Driver
- C-API to perform AXI bus functions
- Shared Memory Allocation by ProtoBridge Software

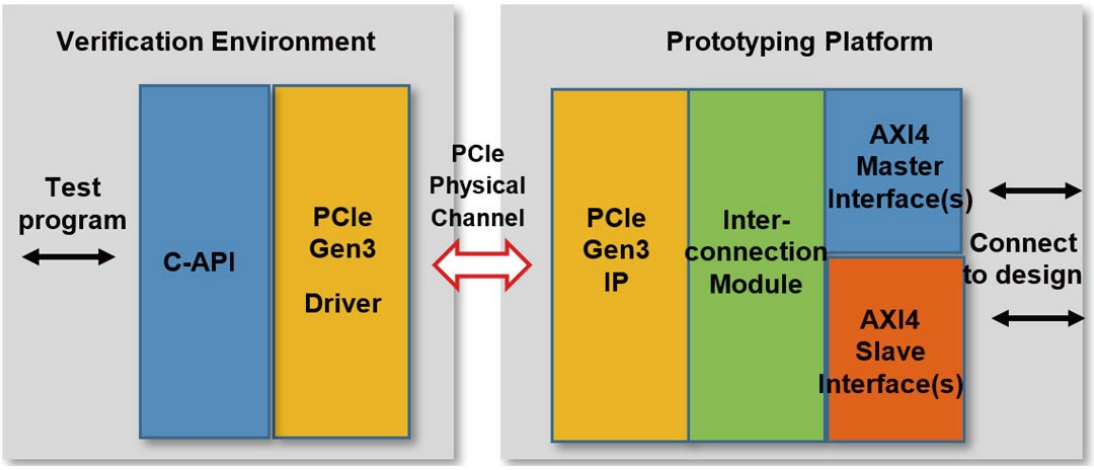


Figure 5. S2C Prodigy ProtoBridge – Software & Hardware components

Besides, ProtoBridge provides a library with a rich set of C-API calls. These C-API calls are used to perform various functions, like system initialization, data read/write, DMA mode, etc.

1. System initialization function calls to manage the tool environment

2. Interrupt control function calls to identify the source of an interrupt signal for C-API's follow-up actions

3. Data read/write function calls to communicate with and operate the FPGA circuit

4. DMA transfer functions calls to perform DMA operations for large amounts of data

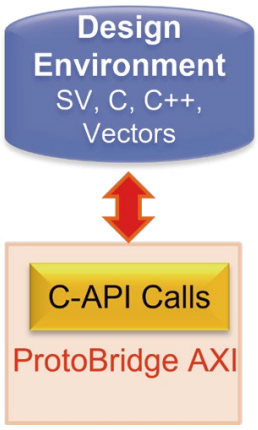


Figure 6. S2C Prodigy ProtoBridge

A system like this allows designers to maximize the benefits of FPGA-based prototypes much earlier in the design project for algorithm validation, IP design, simulation acceleration, and corner case testing. A prototype combined with a transactor interface makes a range of exciting applications possible throughout the design flow.

S2C ProtoBridge enables ESL and FPGA prototyping to coexist

ProtoBridge from S2C provides a high bandwidth data channel between software models running on a host PC and the FPGA prototyping hardware. ProtoBridge consists of C-API for users, software drivers for the host PC, PCIe-based connectivity hardware between the host PC and the Prodigy Logic System, and a PCIe-to-AXI bridge to interface with the user design blocks and provides accurate simulation responses for architecture exploration.

To tap into a wide range of models, S2C has collaborated with Mirabilis Design to deliver a hybrid SoC architecture exploration solution that reuses available RTL-based blocks to accelerate model construction and complex simulations. Mirabilis Design's VisualSim interfaces with S2C's FPGA Prototyping solution, Prodigy Logic System is running ProtoBridge [8], to model a functional block of the design in which the FPGA prototype acts as a sub-model.

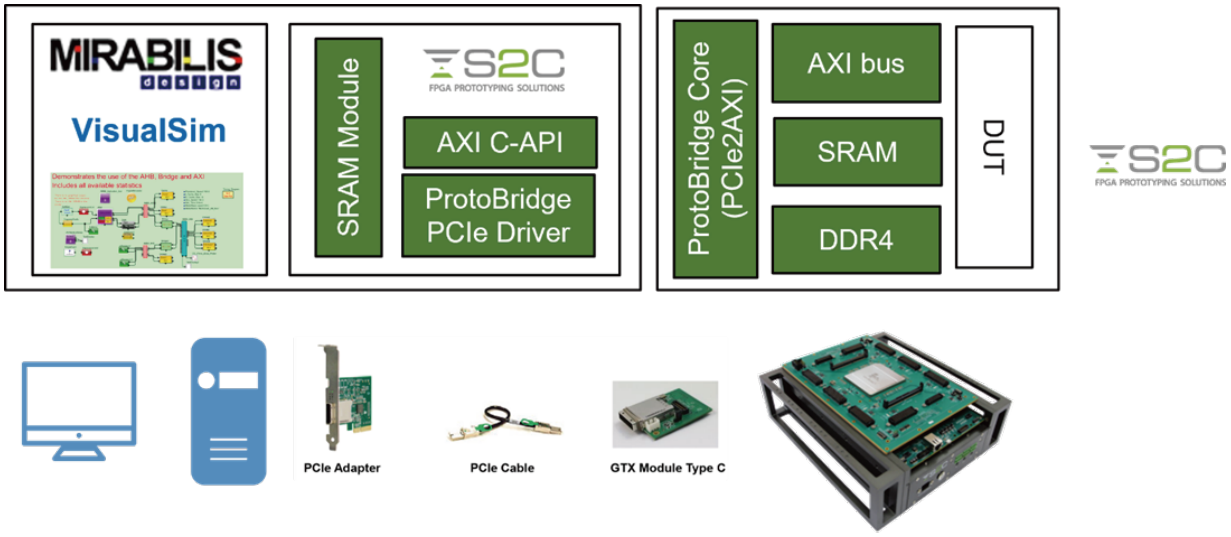


Figure 7. S2C Prodigy ProtoBridge and Mirabilis environment

The collaboration enables the RTL behavior modeled through FPGA prototyping to be easily integrated into an ESL model to create a virtual platform. The model can be simulated to gather metrics on response times, throughput, power consumption, and correctness of data values.

Summary

Creating and comprehensively verifying an ASIC/SoC prototype before silicon implementation is an absolute necessity. But these prototypes need to be able to run with high performance - 10s of MHz so that you can (i) run massive test suites, (ii) interface with real-time devices, and most importantly, (iii) run initial firmware and application software. FPGA-based prototyping system perfectly fits the bill. But creating a prototype comes with a unique set of challenges – lack of RTL models, legacy netlist only blocks, and the need to transfer data at extreme speeds because of the nature of the application (e.g., video clip analysis) or model being devel

oped (e.g., Deep Learning models that require intense DSP computation). S2C ProtoBridge helps build an infrastructure that consists of a hardware and software environment. This, in turn, enables models at different abstraction levels – RTL, ESL, and behavioral code to communicate seamlessly. ProtoBridge, with its ability to transfer data bidirectionally at the rate of 4 GB/s, makes it possible to have large amounts of data available instantly for processing and empty the processed data from the buffer memory - FPGA BRAM or DDR4 on the FPGA platform, equally fast for further analysis.

REFERENCES

1. Mukesh Khare, https://www.ibm.com/thought-leadership/innovation-explanations/mukesh-khare-on-smaller-transistors-analytics

2. Ashok Kulkarni, “Addressing the new challenges of ASIC/SoC prototyping with FPGAs”, https://www.eetimes.com/addressing-the-new-challenges-of-asic-soc-prototyping-with-fpgas/

3. Ashok Kulkarni, “Verifying Complex ASICs through FPGA-based Prototyping”, https://s2c.sharepoint.cn/Product%20Documents/S2C%20Tools/Archives/Competitive%20Study/Synplicity/verifying_complex_asics.pdf#search=white%20paper

4. Harry Foster, "The 2020 Wilson Research Group Functional Verification Study- Part 1," Siemens, 5 11 2020. [Online]. https://blogs.sw.siemens.com/verificationhorizons/2020/11/05/-part-1-the-2020-wilson-research-group-functional-verification-study/.

5. ‘Art of Verification’ - https://www.theartofverification.com/directed-testing-vs-constraint-random-verification/

6. Wikipedia, "Electronic system-level design and verification," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Electronic_systemlevel_design_and_verification.

7. Brian Bailey, Grant Martin and Andrew Piziali, ESL Design and Verification: A Prescription for Electronic System Level Methodology. Morgan Kaufmann/Elsevier, 2007.

8. S2C and Mirabilis Design Team up to Deliver a Heterogeneous Solution for SoC Architecture Exploration and Verification https://www.s2ceda.com/en/info-media-163