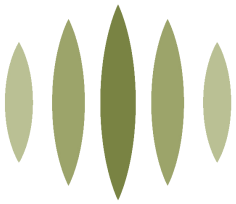
AXI HW/SW VERIFICATION FOR FPGA

Bruno Bratti

Principal Engineer, Wave Semiconductor



wave semi®

Overview

- Our Project
  - Platform
  - FPGA Development Environment
  - Verification Environment
- AXI Background
  - AXI Details
  - AXI Channel Architecture
  - Example Transactions
- DPI
  - Imported and Exported Tasks and Functions
  - Example DPI code
  - Example DPI AXI Master Transactor
- Summary
  - Questions

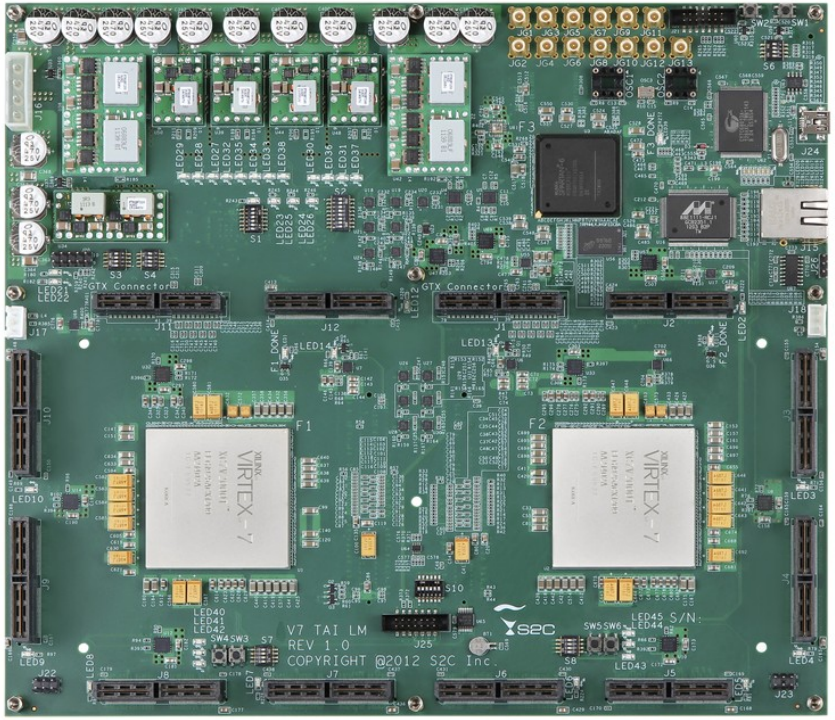
wave semi®

Our Project

- Customer delivery of AXI connected RTL IP delivered on FPGA Model
- IP runs SW which is loaded through AXI
- Dynamic loading and unloading of SW on FGPA model via host PC
- This prevented several verification challenges

Platform

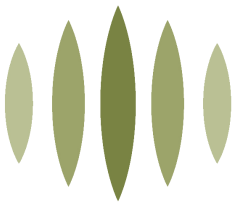
- We used the S2C platform with Xilinx Virtex7 FPGAs



wave semi®

FPGA Development Environment

- Used Xilinx Vivado 2014.2
- Xilinx is pushing hard on AXI as the interconnect of choice for IP on it's FPGAs
- Easy to use and integrate AXI IP in Xilinx FPGA projects
- Used S2C Protobridge to drive AXI on FPGA
  - 3rd Party User level driver connects to Xilinx PCIe IP on FPGA board allowing user level access to AXI master which drives the environment
  - Can read and write to FPGA from PC using simple functions like:
    - AXI_Write128(…, AXI_Read128(…, etc…



wave semi®

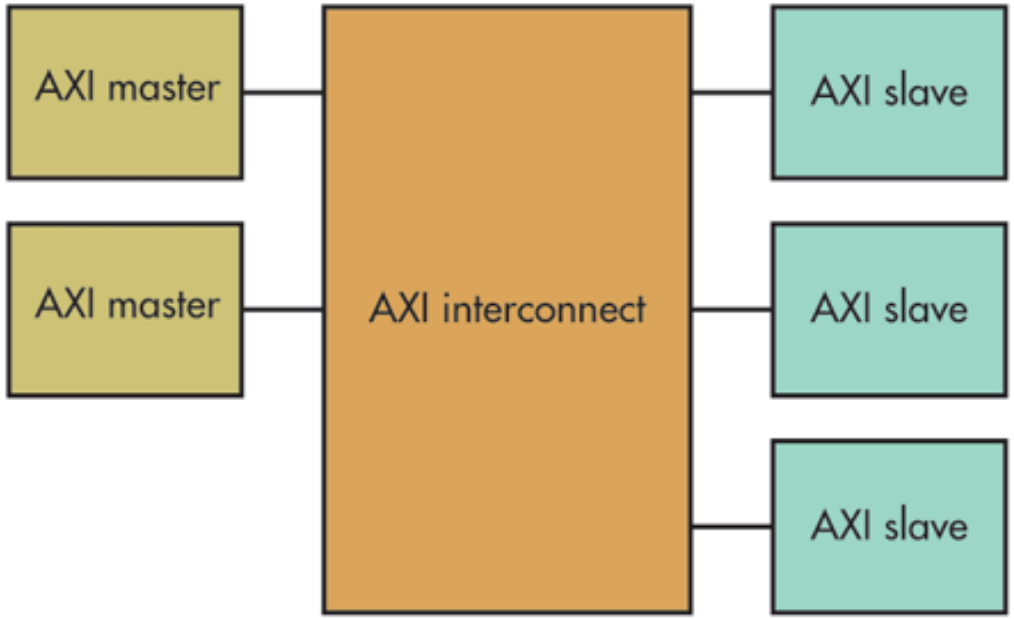
Verification Environment

- RTL simulation with VCS
- We want to use the same C code in VCS simulation to drive FPGA in production
  - AXI_Write128(…, AXI_Read128(…, etc…
- Solution: Use DPI
  - DPI allows C/C++ code to connect with an RTL simulation
  - Create DPI based AXI Master Transactor to drop into FPGA RTL for simulation, replacing connection to Xilinx AXI to PCIe bridge.

wave semi®

AXI Background

- AXI = Advanced eXstensible Interface
- ARM introduced AXI at the Embedded Processor Forum in 2003
- Provides high bandwidth and low latency



wave semi®

AXI Details

- Separate address/control and data phases
- Separate read and write channels (for efficient DMA)
- Unaligned data transfers using separate byte-lane strobes
- Multiple outstanding transactions
- Out of order transaction completion

AXI Channel Architecture

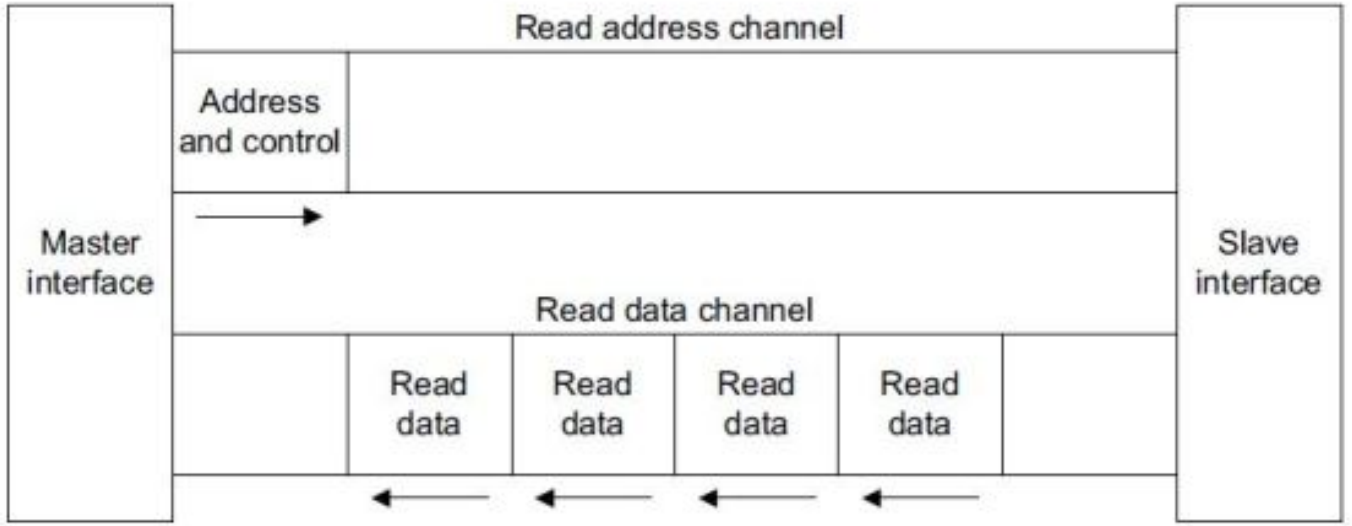
- 5 unidirectional channels
  - Write Address
  - Read Address
  - Write Data
  - Read Data
  - Write Response

Transaction channel handshake pairs

Transaction channel	Handshake pair
• Write address channel	AWVALID, AWREADY
• Write data channel	WVALID, WREADY
• Write response channel	BVALID, BREADY
• Read address channel	ARVALID, ARREADY
• Read data channel	RVALID, RREADY

Example: AXI Read Transaction

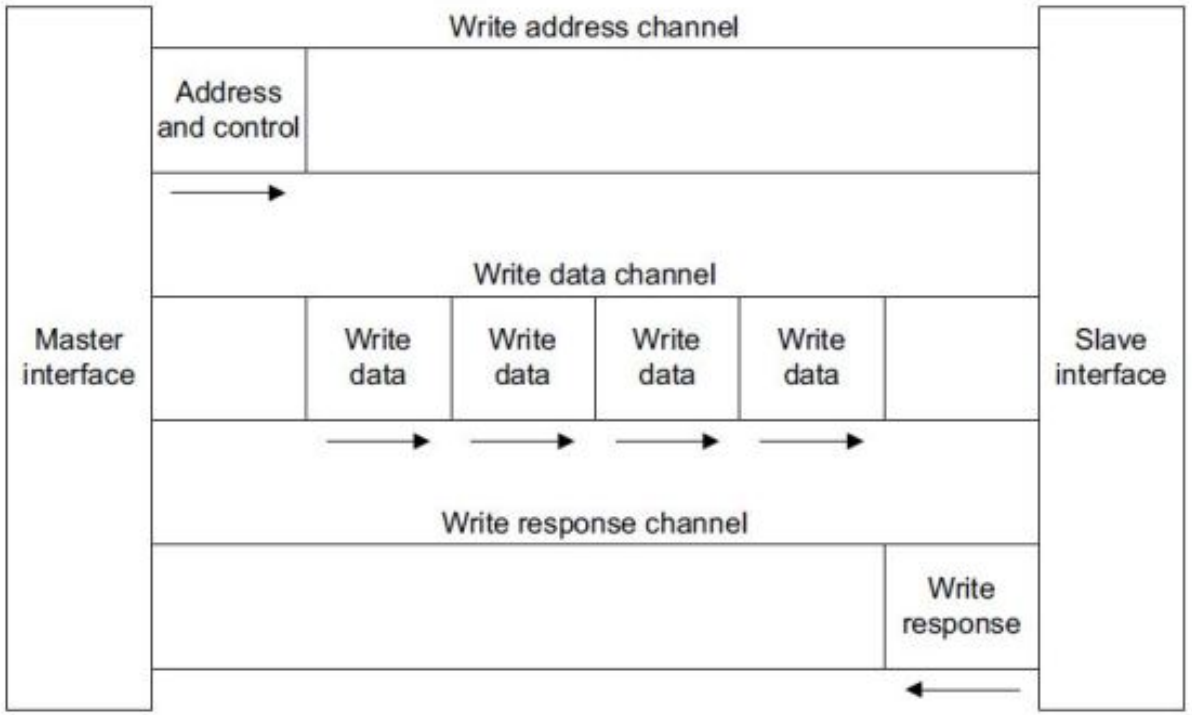
- Master issues address and control
- Slave returns data and response



wave semi®

Example: AXI Write Transaction

- Master issues address and control
- Master sends data
- Slave acknowledges

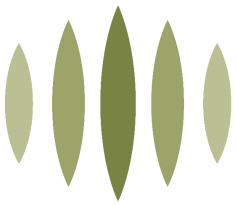


DPI Overview

- The SystemVerilog Direct Programming Interface (DPI) is an interface between SystemVerilog and the C language. It allows the designer to easily call C functions from SystemVerilog and to export SystemVerilog functions, so that they can be called from C.
- The DPI has great advantages: it allows the user to reuse existing C code and also does not require the knowledge of previous mechanisms to connect simulation and C code. The previous mechanisms like VPI, etc. were confusing and cumbersome.

Imported and Exported Tasks and Functions

- Functions implemented in C can be called from SystemVerilog using import "DPI" declarations. We will refer to these functions as imported tasks and functions. All imported tasks and functions must be declared.
- Functions and tasks implemented in SystemVerilog and specified in export "DPI" declarations can be called from C. We will refer to these tasks and functions as exported tasks and functions.



wave semi®

Example System Verilog Code

```
import  "DPI-C" context task wm_main();
export "DPI-C" task waitConfigDone;
  task waitConfigDone;
  begin
    $display("Waiting for config done\n");
     // CODE GOES HERE
$display("Got config done\n");
  end
  endtask
```

Example C Code

```
extern "C" void wm_main()
{
  printf("C CODE - Calling w_main. Wait for config done\n");
  waitConfigDone();
// CODE GOES HERE
  printf(“TEST STARTS\n");
```

DPI Command Lines

- Comple C Code:
  - g++ -fPIC -c -I. -o bruno_dpi.o -DRTL_SIM test_code.cpp -I/tools/synopsys/vcs/J-2014.12-1/include
  - g++ -shared -o libbruno.so bruno_dpi.o
- VCS command line addition:
  - vcs ….. libbruno.so …..
- Run test
  - ./simv |& tee runlog.out

DPI AXI Master Transactor C code

```
void PCIE_Write128_h(UINT32 base, UINT64 dwOffset,
svBitVec32* WdataVec ) {

svBitVec32 addrVec[SV_CANONICAL_SIZE(64)] ;

  addrVec[0] =  dwOffset;

  addrVec[1] =  base;

  printf("WRITE 128 A:0x%x%x D: 0x%x 0x%x 0x%x 0x%x\n",
base,dwOffset,WdataVec[0], WdataVec[1], WdataVec[2],
WdataVec[3]);

  master_xtor_write(addrVec,16,WdataVec);

  return;
```

DPI AXI Master System Verilog Code

```
task master_xtor_write;
input [AXI_AW-1:0] addr;
input bytes_num;
input [127:0] data_in_cpp;

bit  [AXI_AW-1:0] addr;
int bytes_num;
bit  [MAX_WIDTH-1:0] data_in;
bit  [127:0] data_in_cpp;
```

Summary

- Using the approaches described in this presentation we were able to deliver our FPGA IP model to our customer with minimal time spent on debug
- AXI is an easy to use industry standard to connect IP in SOC / FPGA environments
- DPI is easy to use to connect System Verilog simulations with C code