Gisselquist Technology, LLC

0. Preface

Daniel E. Gisselquist, Ph.D.

Lesson Overview

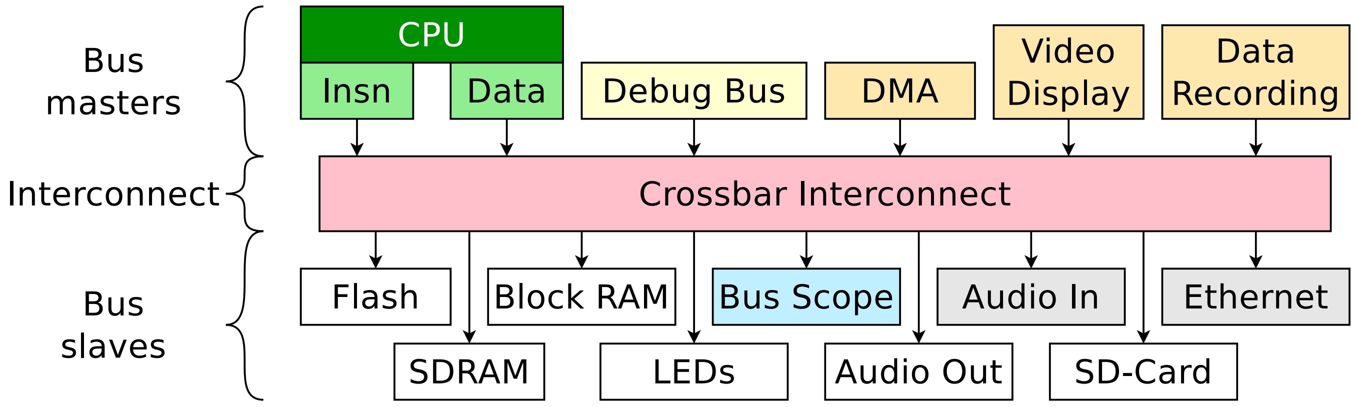
Course Objective: To learn and become familiar with using a bus

- Check the Pre-requisites
- Getting Started

Intermediate Design

Intermediate FPGA design focuses on bus components

- Busses consist of both masters, slaves, and bridges
  - Masters can issue read/write requests of any address
  - Slaves respond to those requests
  - Bridges transform requests received on a slave port into a master port
  - An *interconnect* connects masters and slaves together, so that a master can select a particular slave by its address



Intermediate Design

Intermediate FPGA design focuses on bus components

- Busses consist of both masters, slaves, and bridges
- Examples
  - Bus masters: CPU (instructions, data), DMA, Video, external device command interface (the debug bus master)
  - Bus slaves: Memory (block RAM, flash, SDRAM), SD-Card controllers, serial ports, audio output ports, control registers within a design
    Even a CPU debugging port can be a bus slave in its own right

Intermediate Design

Intermediate FPGA design focuses on bus components

- Busses consist of both masters, slaves, and bridges
- Examples: CPU, DMA, memory, peripherals, etc.
- Bus standards: WB-classic, WB-pipeline, AXI-lite, AXI4 (full), AHB, APB, TileX, etc.
- Actual bus implementations may also depend upon:
  - The width of the data
  - Clock and reset sources and types
  - Big vs Little Endian
  - Optional bus features which may (or may not) be supported

Wishbone bus

This course will focus on the Wishbone (B4) bus with a pipeline implementation

- 32-bit data bus
- Single clock and reset domain
- Pipeline implementation allows one transfer per clock, depending upon the slave and master but not on the interconnect
- Fewest options: no tag lines, no retry signal, etc.

Why? Because it's easy:

- It's easy to learn
- It's easy to formally verify components of this bus
- Pipelined transactions can achieve high performance i.e. 100% throughput

Other busses

I do not wish to be unique to Wishbone

- I anticipate companion courseware/materials for other busses built off of this courses outline
- Feel free to ask for AXI-lite or even AXI course materials

It's just that . . . Wishbone is easy to start with

Why not AXI?

AXI is a very complicated protocol

- Neither Xilinx nor Intel managed to create bug-free AXI training materials
- They're professionals
- If the professionals can't get it right, why should beginner's start there?

Wishbone is much easier to start out with

If you need it

If you need to mix AXI designs with these course materials . . .

- You can still use AutoFPGA
- You'll need a bus bridge
  - To convert from WB to AXI, or
  - To convert from AXI to WB
    - This one can be broken into both a read bridge and a write bridge

Wishbone is much easier to start out with

Vendor Independence

My goal in this intermediate course is vendor independence

- Most bus-based intermediate example designs depend upon proprietary cores and tools
  - Vendor provided interconnect logic
  - Cores: ARM CPU, NiOS2 CPU, MicroBlaze CPU, vendor IP cores, etc.
    - These have proprietary implementations, and
    - Cannot be simulated using Verilator
  - Vendor cores for streaming and copying data
  - They force you to re-learn everything if you ever need to change vendors

This is unacceptable

Vendor Independence

My goal in this intermediate course is vendor independence

- Most bus-based intermediate example designs depend upon proprietary cores and tools
- My intent is openness
  - I'll start with the ZipCPU and its infrastructure
    - The CPU is open source
    - The infrastructure is open source
    - You are free to replace the ZipCPU with any other CPU core you would like
    - I intend to provide an example demonstrating this as part of the course

Vendor Independence

My goal in this intermediate course is vendor independence

- Most bus-based intermediate example designs depend upon proprietary cores and tools
- My intent is openness
  - ZipCPU and its infrastructure
  - AutoFPGA to connect components
    - If you would like to use a non-AutoFPGA generated interconnect, you are free to substitute your own and ignore the AutoFPGA configuration slides
    - AutoFPGA is a fully open source program–you are welcome to modify it for your purposes under the terms of the GPLv3
    - The logic generated by AutoFPGA is your logic. License it as you wish.

AutoFPGA

AutoFPGA composes designs from components

- At it's most basic, it's a copy-paste tool to create design files
  - `rtl/main.v`: Your main project logic
    (Vendor neutral logic, simulatable w/ Verilator)
  - `rtl/toplevel.v`: Your top-level project file
    (Vendor specific logic, not simulatable w/ Verilator)
  - `sim/main_tb.cpp`: Software used to connect emulated external devices to a Verilator based simulation of your design
  - `sw/host/regdefs.h`: Header files for loading and interacting with a design using C++ from an external source
  - `sw/host/regdefs.cpp`: Maps register names to C++ address constants

AutoFPGA

AutoFPGA composes designs from components

- At it's most basic, it's a copy-paste tool to create design files
  - Verilog, simulation, external interface files . . .
  - `sw/board/board.h`: Header files for any embedded core CPU you might have within your design
  - `sw/board/board.ld`: Linker scripts, to tell a CPU where to find its instructions and data in the address space
  - Nothing in a design is "hidden": it's all available for inspection, adjustment, or even fixing

AutoFPGA

AutoFPGA composes designs from components

- At it's most basic, it's a copy-paste tool to create design files
- AutoFPGA also generates interconnect logic
  - Given a set of bus standards, a set of bus masters and bus slaves, AutoFPGA will create an interconnect to connect these together
  - This includes address assignment
  - Eventually, I'd like to be able to automatically insert bus bridges as they are needed–across bus standards, clock rates, reset types, data sizes, etc.
    - AutoFPGA does not (yet) have this capability.
    - Such bridges can easily be inserted manually, just not automatically

AutoFPGA

AutoFPGA composes designs from components

- At it's most basic, it's a copy-paste tool to create design files
- AutoFPGA also generates interconnect logic
- Unlike other "higher-level" language tools, AutoFPGA intends its computer generated design components to remain "human readable"
- If done well, adding or removing a component from a design will require nothing more than adding or removing their configurations from the command line
  - This is the intent
  - There are many ways this can be broken
- Designs composed using AutoFPGA may abandon AutoFPGA at any time and switch to basic (System)Verilog

AutoFPGA

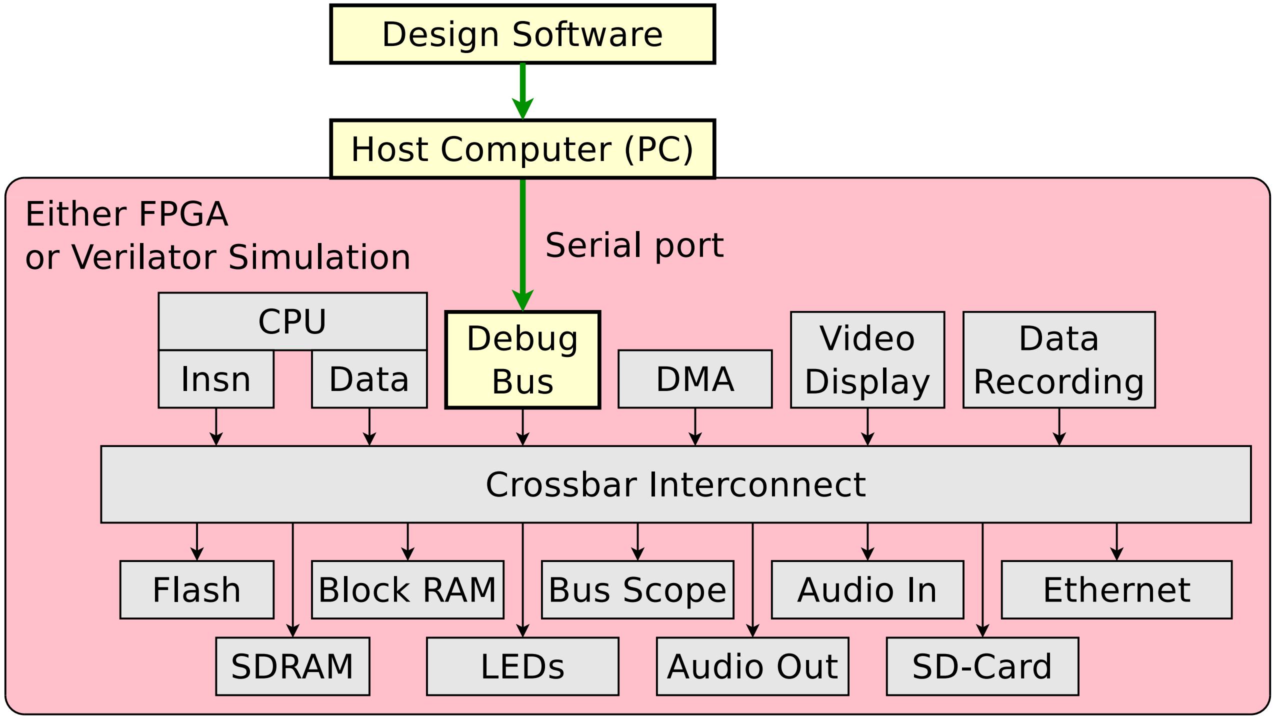
AutoFPGA should help you compose complex designs from components.

- It is a tool
- It does not pretend to build it for you
- The resulting design is still yours to debug

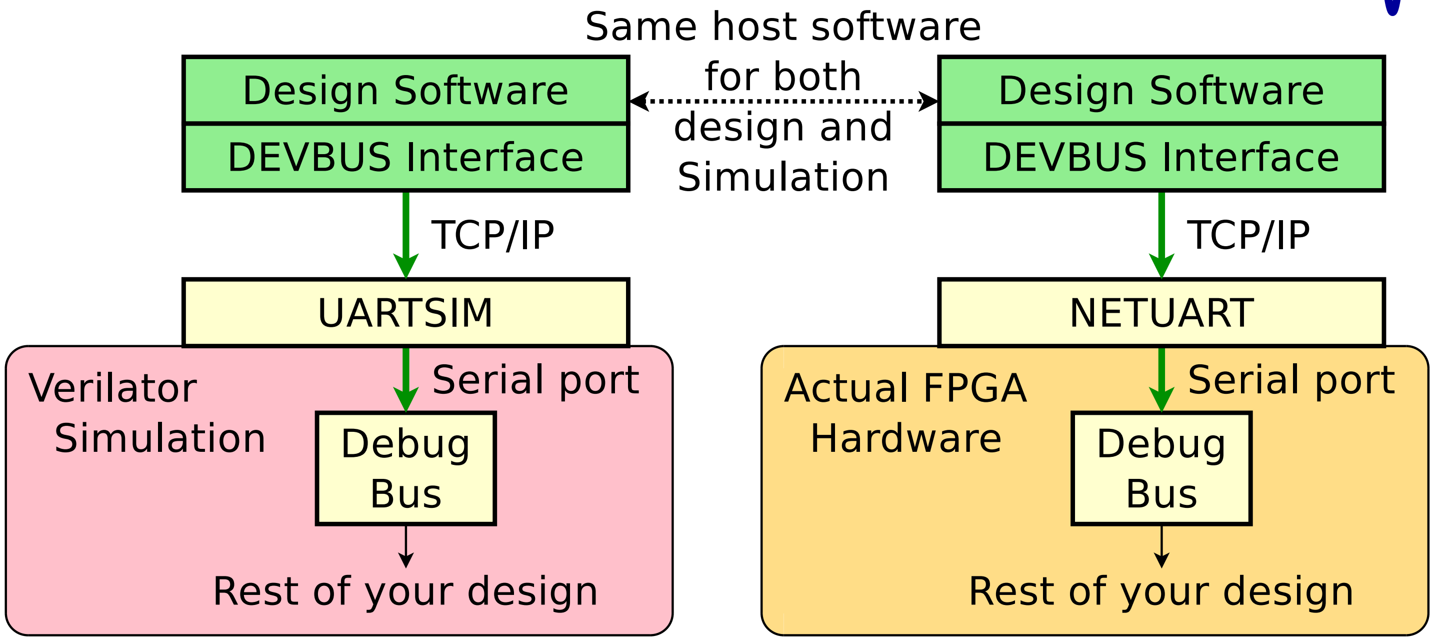
Debugging bus

Our design methodology will center first around a debugging bus

- You can interact with your design from your host CPU
- Interaction is the same both in simulation and hardware
- Requires a serial port to access your design



Debugging bus

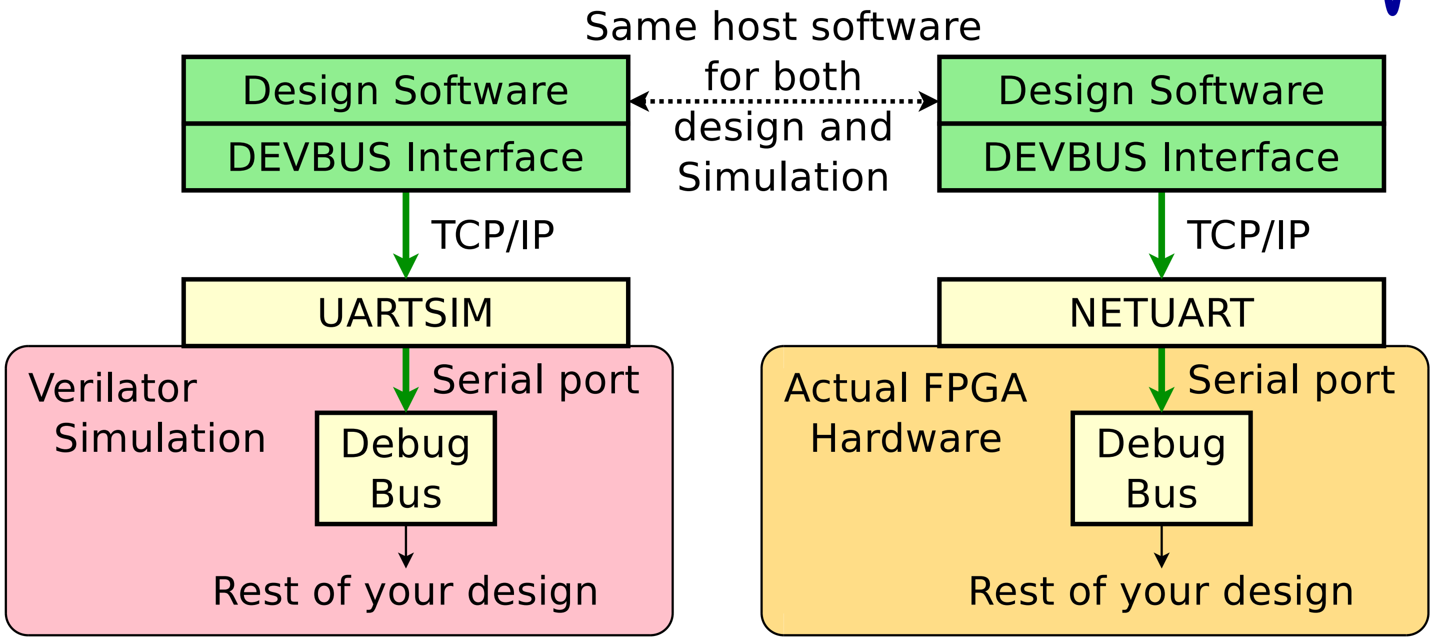


Using the debugging bus you can interact with either

- Your simulation, or
- Your actual design

Using the same host-side software

Debugging bus



The debugging bus makes interacting with your design easy

- Facilitates Ad-hoc or scripted interaction
- Loading CPU software, programming flash ROM
- Capturing data from within a design
- Pre-embedded CPU software testing

Which board?

As with the beginner tutorial

- This course is intended to remain board-agnostic
- We'll use Verilator extensively
- You don't need a board to take this course
- You will likely enjoy the course more with a board

Minimum Board

To do any of the hardware examples you will need a board with (at a minimum):

- Serial port
- Some LEDs and buttons
- 8k LUTs or more

While you may use a board with a SOC-based FPGA on it

- The hard CPU built into your FPGA will likely get in the way (initially)

Minimum Board

Certain lessons of this Intermediate course will require more hardware to test them than the beginners course

▫ Flash memory	(Lesson 5)
▫ 4MB+ External memory is recommended	
– We may use a vendor DDR3 SDRAM controller here	
– A WB to AXI bridge will handle any necessary translation	
▫ Video (VGA or HDMI) output	(Lessons 6-9)
▫ PWM-based Audio output	(Lesson 2 and 12)
▫ Audio (or other) data source	(Lesson 11)
▫ SDCard	(Lesson 20)

Conclusion

- Are you ready to learn?