INTRODUCTION TO FPGA PROGRAMMING

Lesson 16: Advanced FPGA Topics

Dr. Davide Cieri[1]

[1]Max-Planck-Institut für Physik, Munich

September 2024

MAX-PLANCK-INSTITUT FÜR PHYSIK

INTRODUCTION

- This lecture shows an overview of other advance FPGA topics, that we didn’t have time to cover during the course
- I will not provide too many details
- Starting point for further studies

ADVANCED VERIFICATION METHODOLOGIES

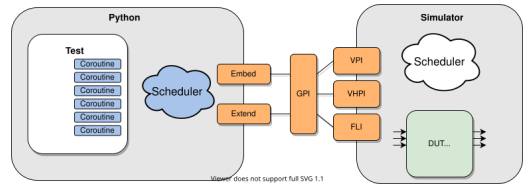
- Verification is crucial for ensuring the correctness of VHDL designs
- VHDL already provides standard features to automatise your test-benches
- Methodologies (Libraries) provide structured approaches to verification.
- UVVM and OSVVM are two popular methodologies in the VHDL community.

UVVM AND OSVVM

- UVVM (Universal VHDL Verification Methodology) and OSVVM are both free, open-source methodology for VHDL verification.
- They aim to simplify and standardize the verification process.
- Provide a set of libraries and tools for writing and managing testbenches
- Both requires simulator with full support to VHDL-2008 (no Vivado)

COCOTB

- *COroutine based COsimulation TestBench* (CocoTB)
- Runs python code concurrently and synchronously to the simulation
- No HDL testbench needed
- All python packages available
- Use cases
  - Python golden model
  - Connect real hardware
- Low simulation performance (longer execution)
- No support to Vivado simulator (yet)

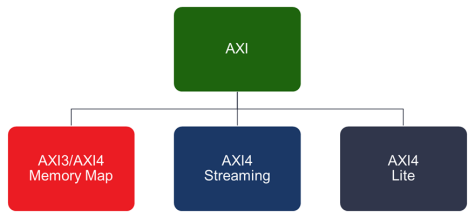


VUNIT

- VUnit – Python based Test Runner
- Open source unit testing framework for VHDL/SystemVerilog
- Workflow:
  1. Scan through all files in a folder and build a dependency tree
  2. Search for testbenches
  3. Compile the sources in the correct order
  4. Start the simulation (in parallel)
  5. Report the test results

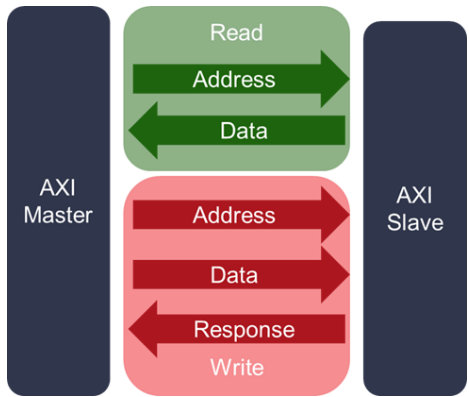
AXI

- AXI (Advanced eXtensible Interface) is an interface protocol defined by ARM.
- Nearly every Xilinx IP uses an AXI Interface
- Three types of AXI4-Interfaces:
  - AXI4 (Full AXI4): For high-performance memory-mapped requirements.
  - AXI4-Lite: For simple, low-throughput memory-mapped communication (for example, to and from control and status registers).
  - AXI4-Stream: For high-speed streaming data



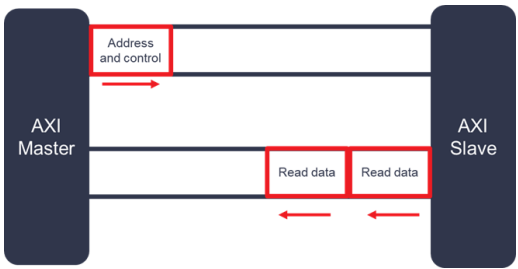
AXI READ AND WRITE CHANNELS

- The AXI protocol defines 5 channels:
- Two for Read transactions:
  - Read Address
  - Read Data
- Three for Write Transactions:
  - Write Address
  - Write Data
  - Write Response

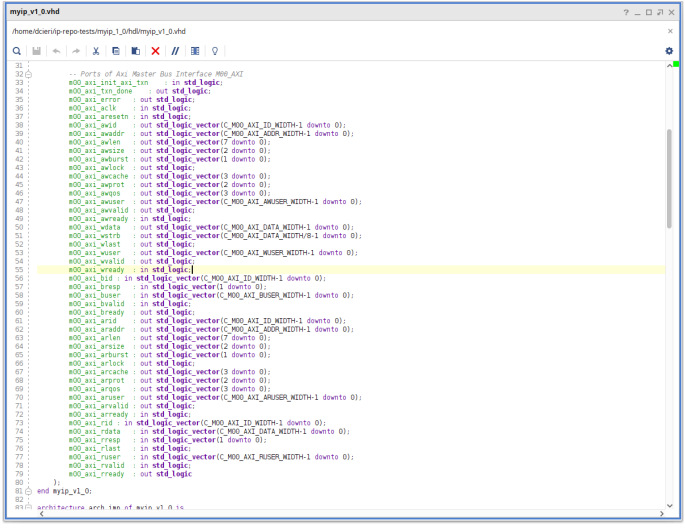


AXI READ AND WRITE TRANSACTIONS

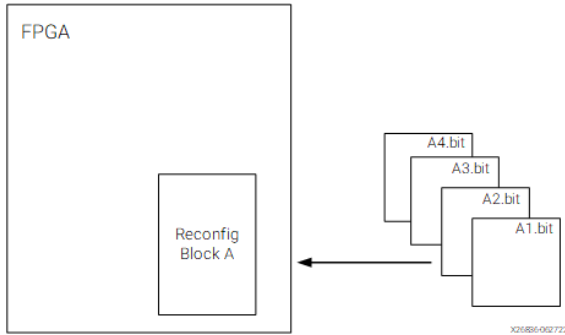
- AXI Master always initiates the transactions
- Multiple data can be transmitted on the same AXI address (burst)



AXI REALITY


myip_v1_0.vhd
/home/dcieri/ip-repo-tests/myip_1.0/hdl/myip_v1_0.vhd
-- Ports of Axi Master Bus Interface M00_AXI
m00_axi_init_axi_txn : in std_logic;
m00_axi_txn_done : out std_logic;
m00_axi_error : out std_logic;
m00_axi_aclk : in std_logic;
m00_axi_aresetn : in std_logic;
m00_axi_awid : out std_logic_vector(C_M00_AXI_ID_WIDTH-1 downto 0);
m00_axi_awaddr : out std_logic_vector(C_M00_AXI_ADDR_WIDTH-1 downto 0);
m00_axi_awlen : out std_logic_vector(7 downto 0);
m00_axi_awsize : out std_logic_vector(2 downto 0);
m00_axi_awburst : out std_logic_vector(1 downto 0);
m00_axi_awlock : out std_logic;
m00_axi_awcache : out std_logic_vector(3 downto 0);
m00_axi_awprot : out std_logic_vector(2 downto 0);
m00_axi_awqos : out std_logic_vector(3 downto 0);
m00_axi_awuser : out std_logic_vector(C_M00_AXI_AWUSER_WIDTH-1 downto 0);
m00_axi_awvalid : out std_logic;
m00_axi_awready : in std_logic;
m00_axi_wdata : out std_logic_vector(C_M00_AXI_DATA_WIDTH-1 downto 0);
m00_axi_wstrb : out std_logic_vector(C_M00_AXI_DATA_WIDTH/8-1 downto 0);
m00_axi_wlast : out std_logic;
m00_axi_wuser : out std_logic_vector(C_M00_AXI_WUSER_WIDTH-1 downto 0);
m00_axi_wvalid : out std_logic;
m00_axi_wready : in std_logic;
m00_axi_bid : in std_logic_vector(C_M00_AXI_ID_WIDTH-1 downto 0);
m00_axi_bresp : in std_logic_vector(1 downto 0);
m00_axi_buser : in std_logic_vector(C_M00_AXI_BUSER_WIDTH-1 downto 0);
m00_axi_bvalid : in std_logic;
m00_axi_bready : out std_logic;
m00_axi_arid : out std_logic_vector(C_M00_AXI_ID_WIDTH-1 downto 0);
m00_axi_araddr : out std_logic_vector(C_M00_AXI_ADDR_WIDTH-1 downto 0);
m00_axi_arlen : out std_logic_vector(7 downto 0);
m00_axi_arsize : out std_logic_vector(2 downto 0);
m00_axi_arburst : out std_logic_vector(1 downto 0);
m00_axi_arlock : out std_logic;
m00_axi_arcache : out std_logic_vector(3 downto 0);
m00_axi_arprot : out std_logic_vector(2 downto 0);
m00_axi_arqos : out std_logic_vector(3 downto 0);
m00_axi_aruser : out std_logic_vector(C_M00_AXI_ARUSER_WIDTH-1 downto 0);
m00_axi_arvalid : out std_logic;
m00_axi_arready : in std_logic;
m00_axi_rid : in std_logic_vector(C_M00_AXI_ID_WIDTH-1 downto 0);
m00_axi_rdata : in std_logic_vector(C_M00_AXI_DATA_WIDTH-1 downto 0);
m00_axi_rresp : in std_logic_vector(1 downto 0);
m00_axi_rlast : in std_logic;
m00_axi_ruser : in std_logic_vector(C_M00_AXI_RUSER_WIDTH-1 downto 0);
m00_axi_rvalid : in std_logic;
m00_axi_rready : out std_logic
);
end myip_v1_0;
architecture arch_imp of myip_v1_0 is

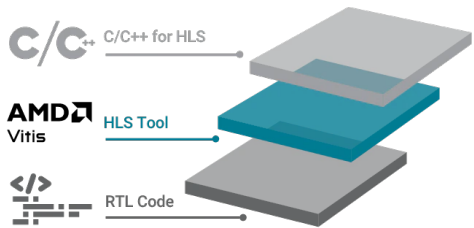
PARTIAL RECONFIGURATION



- Partial Reconfiguration or Dynamic Function eXchange, allows for reconfiguration of modules within an active design
- A full bitstream is loaded on the FPGA, including FPGA regions that can be reconfigured
- Partial bitstream is loaded on the FPGA (dynamic configuration) during operation, to modify reconfigurable regions

HIGH LEVEL SYNTHESIS

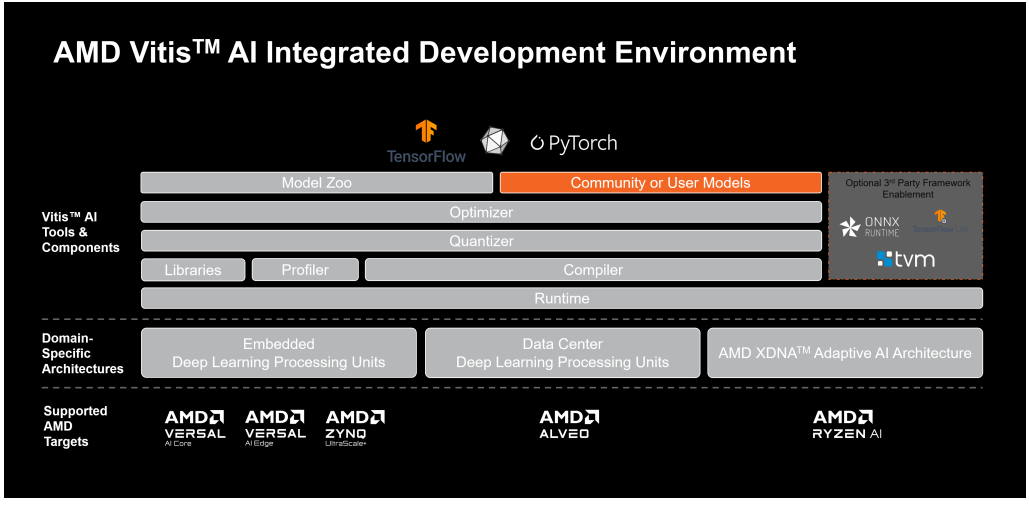
- Major vendor provides tool to translate C/C++ code into RTL
- Xilinx AMD Vitis HLS
  - Write code in C++ / C / System C
  - Code converted by Vitis HLS into RTL
  - C simulation available to validate functionalities
  - Eventually, RTL is packed in an IP, and can be implemented in Vivado



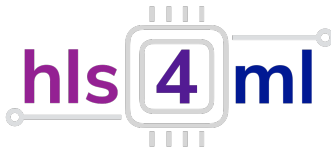
MACHINE LEARNING ON FPGAS

- Due to their ability of processing large chunks of data in parallel, FPGAs are ideal to infe machine learning algorithms
- Different options for the implementations:
  - Write your own VHDL implementation of the algorithm
  - Vitis-AI
  - HLS4ML

VITIS AI


AMD Vitis™ AI Integrated Development Environment
TensorFlow
PyTorch
Vitis™ AI Tools & Components
Model Zoo
Community or User Models
Optional 3rd Party Framework Enablement
Optimizer
ONNX RUNTIME
Quantizer
tvm
Libraries
Profiler
Compiler
Runtime
Domain-Specific Architectures
Embedded Deep Learning Processing Units
Data Center Deep Learning Processing Units
AMD XDNA™ Adaptive AI Architecture
Supported AMD Targets
AMD VERSAL AI Core
AMD VERSAL AI Edge
AMD ZYNQ UltraScale+
AMD ALVEO
AMD RYZEN AI

HLS4ML



- HLS4ML is a python package that translates machine learning models into C++ code for HLS
- Optimizes the algorithm for FPGA implementation
- Executes Vitis-HLS within python, to run synthesis and IP exportation

CONTINUOUS INTEGRATION AND VERSION CONTROL

- Source code control in VHDL project is fundamental, especially in large collaboration
  - Even small changes in the design, can result in large differences in the implementations
- Vivado projects are not git-friendly (long xml files)
- Hog (HDL-on-Git) is an open-source tool, that help maintaining HDL projects with git
  - It also provides templates for Continuous Integration workflow, building and simulating HDL projects

The figures in these slides are taken from:

- Digital Design: Principles and Practices, Fourth Edition, John F. Wakerly, ISBN 0-13- 186389-4.

- allaboutfpga.com
- nandland.com
- docs.amd.com
- https://www.symmetryelectronics.com/
- https://www.edn.com/
- Stephen A. Edwards, Columbia University, Fundamentals of Computer Systems, Spring 2012
- adafruit.com
- www.icdesigntips.com
- techdocs.altium.com
- anysilicon.com
- Yngve Hafting 2021, University of Oslo
- www.myomron.com