Antmicro

Guineveer

2025-12-17

CONTENTS

CHAPTER

ONE

INTRODUCTION

Guineveer is a reference SoC disgn based on the Veer EL2 RISC-V core that uses AXI interconnect to connect two CPUs with UART, I3C and memory. Topwrap is used as a generator for the top module, which enables convenient SoC designing and configuration changes.

This documentation is divided into the following chapters:

- *System architecture* describes key components and interconnects in the SoC
- *Testing* describes design simulation and testing with Renode and Verilator
- *User guide* provides build instructions and describes basic usage

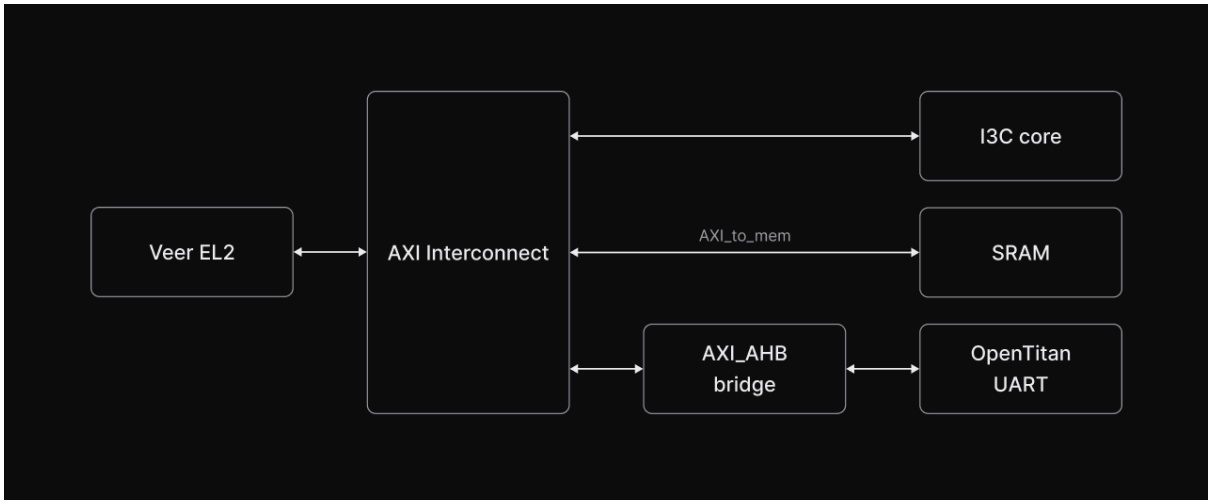
CHAPTER

TWO

SYSTEM ARCHITECTURE

2.1 Block diagram

The following diagram illustrates the key components and interconnections of the Guineveer SoC:



The Guineveer reference System-on-Chip (SoC) design employs the VeeR EL2 core which is a 32-bit CPU supporting RISC-V's extensions for integer (I), compressed instruction (C), multiplication and division (M), and instruction-fetch fence, CSR, and subset of bit manipulation instructions (Z).

The reference design features an AXI system bus which is used to communicate with several peripherals, including an SRAM memory module (accessed via an `AXI_to_mem` interface), an `I3C core`, and an `AXI-to-AHB bridge` (providing access to an `OpenTitan UART` peripheral). There are two VeeR cores, each having its own memory and connection to the AXI interconnect.

The SoC is split into two clock domains: one for the I3C core, and one for the rest of the SoC. On FPGA targets, the I3C core is clocked at 160 MHz, while the rest of the SoC is clocked at 32 MHz.

The entire SoC is described in Topwrap/design.yaml, which contains connections, parameters, and modules used in the top module that will be generated by Topwrap.

2.2 Currently used peripherals and components

Peripheral	Source
VeeR EL2	https://github.com/chipsalliance/Cores-VeeR-EL2
AXI Interconnect	AXI Pulp interconnect wrapper generated by Topwrap
AXI_AHB bridge	https://github.com/antmicro/Cores-VeeR-EL2/blob/main/design/lib/axi4_to_ahb.sv
AXI_to_mem	https://github.com/pulp-platform/axi/blob/master/src/axi_to_mem.sv
AXI_cdc	https://github.com/pulp-platform/axi/blob/master/src/axi_cdc.sv
UART OpenTitan	https://github.com/lowRISC/opentitan/tree/master/hw/ip/uart
I3C core	https://github.com/chipsalliance/i3c-core

2.3 Peripheral and component configuration

2.3.1 VeeR EL2

The VeeR EL2 core is configured with FPGA optimizations enabled; branch predictor, ICCM, DCCM, and I-cache disabled. The full set of configuration options used is found in the root Makefile.

2.3.2 I3C

The I3C core uses mostly its default configuration, with one notable difference: the input sync flip-flops are enabled, which is necessary for FPGAs to prevent glitches.

2.4 Memory map

The table below summarizes the Guineveer memory address map, including the start, end, and size for the various component types.

Start Address	End Address	Size	Type
0x0000_0000	0x1FFF_FFFF	512 MB	VeeR EL2 reserved space
0x3000_0000	0x3000_1000	4 KB	Uart
0x3000_1000	0x3000_2000	4 KB	I3c
0x8000_0000	0x8FFF_FFFF	256 MB	Mem

CHAPTER

THREE

TESTING

Guineveer SoC support the following targets:

- HDL simulation
- Renode simulation and cosimulation
- FPGA
- ASIC (in progress)

3.1 Software tests

Currently, three software test samples are available:

- `uart` - example initializes and transmits `Hello UART` over UART
- `i3c` - example verifies the correctness of some basic operations on the `i3c` device, including:
  - verifying if registers contain expected values after reset
  - verifying if read-only registers are not writeable and if read-write registers are writable
  - verifying if status bits change values after an interrupt condition in forced
- `i3c-cocotb` - test application dedicated for use with the Cocotb I3C tests, which cover:
  - waiting for dynamic address assignment and observing the register changes,
  - performing I3C private writes and reads to the device,
  - performing various directed CCC transactions,
  - performing a streaming boot via the recovery I3C target,
  - performing a streaming boot using the AXI bypass functionality.
- `axi-streaming-boot-dualcore` - example tests AXI streaming boot feature of `i3c-core` using two cores
  - core 0 is waiting for payload from `i3c-core` via registers
  - core 1 is sending payload to `i3c-core` via registers
  - payload can be modified and rebuild, source files are located in `tests/sw/axi-streaming-boot-dualcore/core1/payload`

  - payload size is limited in core 0 software, to incrase limit change `MAX_STREAMING_BOOT_SIZE`

Building software examples is described in the *User guide*.

3.1.1 Running software tests

Software test can be run in multiple ways:

- in Verilator
- in Renode with a behavioral model of the I3C device
- in Renode with a cosimulated model of the I3C device, generated from the original HDL sources.

Running the test software using each of the available methods is described in the *User guide*

3.2 FPGA tests

3.3 Supported FPGA boards

- Arty A7-100T is an FPGA development board based on the Xilinx Artix-7 FPGA.

CHAPTER

FOUR

USER GUIDE

4.1 Building SoC sources

Use the `make hw` command to generate the top module in `hw/guineveer.sv`, using Topwrap and the neccesary files, in the `build/` folder. This command also creates an AXI-based SoC network (`axi_intercon.sv`) source file, in the `hw/` folder.

4.2 Building software examples

Run `TEST=software_example_name make build_test` to compile one of the provided software samples. As of now, the available samples are:

- `uart` - initializes and transfers a "Hello UART" string over UART,
- `i3c` - checks the I3C register values after reset and initializes the peripheral in device mode.
- `i3c-cocotb` - checks communication over I3C; intended to be used with the I3C Cocotb tests.
- `axi-streaming-boot-dualcore` - uses `i3c-core`'s streaming boot capabilites.

4.3 Building testbench simulation

Run the `make testbench` command to generate the simulation testbench executable using `Verilator`. The program is placed in the `build/obj_dir/Vguineveer_tb` file. It can be launched with `+firmware0=/path/to/the/core0.hex +firmware1=/path/to/the/core1.hex` (with the selected firmware).

Note

Please note that Guineveer's RAM is placed at the offset `0x80000000` and the firmware is loaded using the \$readmemh task. If the hex file was created using `objdump -O verilog`, it will have addresses starting at offset `0x80000000`, but \$readmemh expects the starting address `0x0`. Remember to change the addresses to comply with the address range required by \$readmemh.

4.4 Building design for FPGA

- The defined `HEX_FILE0` and `HEX_FILE1` need to point to the path where firmware for each core is stored - you can update this in `guineveer.tcl`. The HEX files are set automatically in the Makefile; using the `TEST` variable, you can select the files for a test, the same way as with building the software (`make build test`).
- To generate the `tcl` file, run `BOARD=<board> make generate_block_design`, in the `fpga/` folder. The supported boards are `NexysVideo-A7-200T` or `Arty-A7-100T`.
- To synthesize Guineveer using Vivado, run `vivado -mode batch -script guineveer.tcl`.

4.5 Running example SW using the testbench

Run `TEST=software_example_name make sim` to launch the testbench executable with the provided software. The log of all register values and their changes throughout the simulation will be written to the `build/exec.log` file.

4.6 Running example SW using Renode Robot Framework

Run `TEST=software_example_name make renode_test` to launch the Renode simulation with the provided software. It will run the software and compare its output with the expected values.

4.7 Running example SW using Renode Robot Framework with cosimulation

Run `make` in the `sw/renode_i3c_cosim` directory to build the cosimulation binary of the I3C device, form the HDL sources. After building the cosimulation binary and the `i3c` test software, `renode-test` can be used to to run the test available in the `sw/guineveer_i3c_cosim.robot` file. Refer to the following chapters in Renode's documentation for a detailed overview of the required dependencies and available options:

- Testing with Renode
- Co-simulating with an HDL simulator