CSCE330402 - Digital Design II

Project 2 Report

first deliverable

Fall 2020

Submitted By: Nouran Abdelaziz 900170863

Ashrakat Elkhalifa 900170051

Amr Abdelazeem 900172775

Hussam Gawish 900171054

I. Project Description

II. Tools and Sources Used

- IBEX CPU source code:
    - This is a complete open source code implementation of the IBEX core which we use in our SoC.
- Caravel chip source code:
    - This is the source code for the layout of the chip in which we will use the mega project area to put or IBEX based SoC.
- SocGen:
    - This is an open source tool which takes the jason description of our system and automatically generates the verilog files out of it. It also has built in IPs and masters libraries.
- OpenLane:
    - Finally, this is an open source tool which will help us to go through the ASIC flow without human intervention

by taking the verilog implementation of our chip and producing a design clean GDS2 files.

III. IBEX Hardening:

In the hardening process, we cloned a copy of the ibex files from the repo https://github.com/lowRISC/ibex.

We used the makefile and the available scripts to generate one of the possible configurations, in the meanwhile as we do more research we decided to use the “small” configuration with flipflop register files (with an area of 26.06kGE according to yosys), at this moment it is not the smallest but it has the best verification status.

Afterwards we used the provided sv2v scripts to convert the System Verilog files implementing the ibex core to older Verilog files (which is necessary to get them to run through yosys, and thus necessary for the yosys and abc component of openlane).

We then started to test hardening the ibex core using openlane with different configurations to try and reach the best performance/size.

We identified two approaches to this, a) we harden the ibex core separately and then simply link it to the SoC generated from socgen as a hard macro, then rerun this through openlane, or b) we flatten out both parts, by linking the ibex core to the SoC then doing the hardening step once. The second approach should be more efficient in terms of sizing, but could face more congestion problems and would be heavily intensive in terms of computation power, meaning that using available university resources would be essential.

IV. SocGen Test:

In the SocGen test:

We got a copy of the project up and running on our local machine for development and testing purposes using: git clone https://github.com/habibagamal/SoC_automation.

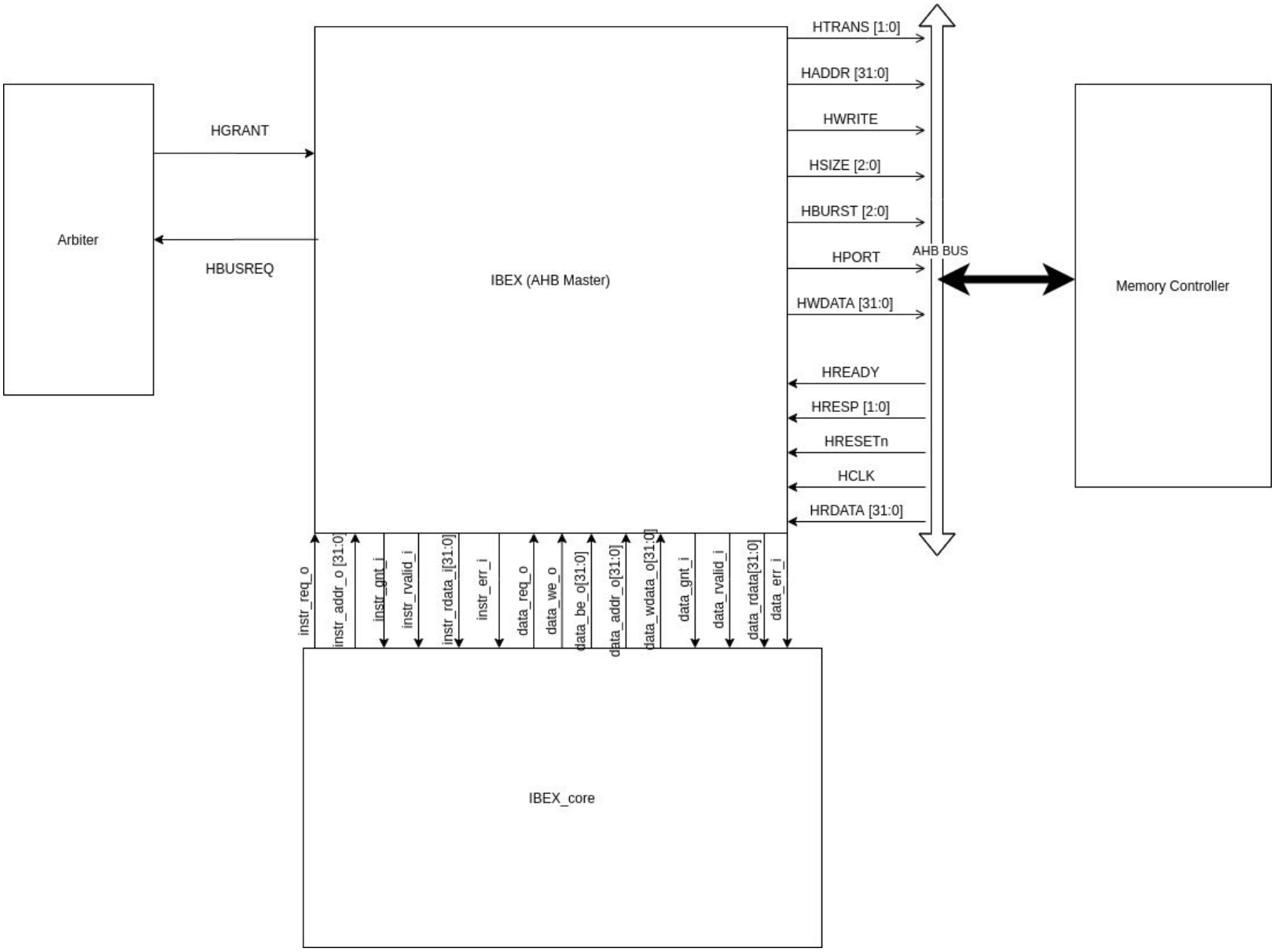
Then,when we tried to run the script compile.sh in SoCGen test, we got the following two errors: one in the build.sh, where a path to the riscv64-gnu-compiler and the other error was a syntax error in the real_dump.vvp. Therefore, we downloaded the riscv64-gnu-compiler using: git clone --recursive https://github.com/riscv/riscv-gnu-toolchain. Then we put the path of the generated riscv gnu compiler in the build.sh to be able to build test cases from c code that can run on the core.We will be using the N5 cpu demo to be able to test the SoCGen.

V. AHB Master Interface for IBEX:

- Problem Statement:
    - SocGen currently supports only masters that are compatible with AHB while IBEX core is not an AHB master, so we need to make the IBEX core compatible with AHB.

- The Ibex core has two interfaces, data interface and instruction interface, so we will need an arbitrary to regulate the transmission between two interfaces

- Black box:



- Black box explanation:

- The IBEX_core module is the top module of the IBEX open source code, and we want to convert this module to an AHB master to be able to generate the SoC through SocGen. That is why we created an IBEX module that acts as an AHB master which takes the exact inputs and outputs of the original IBEX_core but it manipulates the inputs and outputs signals that are related to extraction of instructions in the fetch stage or data in the load-store stage.
- The IBEX core which acts as the AHB master is fed to the SocGen which handles the generating of the verilog code of the AHB bus and other IPs in our system.
- SocGen also should create an arbiter which chooses which transaction to make, whether the fetch of instructions or the load/store of data. Only one will be granted access to the AHB bus.

- IBEX Top module inputs and outputs:

- IBEX Top module is the module that should be the input file in the master JSON
- IBEX module takes all the inputs and outputs of ibex core except for data instruction memory interface and data memory interface. They are replaced by the inputs and outputs of the AHB-master.
- Inputs for AHB-Master are:

-HGRANT_x: This signal indicates if it is high, the bus master has the highest priority. Masters get access to the bus when both HREADY and HGRANT_x are high.

-**HREADY** : when high, it indicates that the bus has finished a transfer. When it is equal to 0, it indicates that a signal is transferred through the bus. The transfer of the signal can be extended by driving HREADY to 0

-HRESP[1:0]: it gives information about the state of the signal. The signal may have four states:

- OKAY: it indicates that the transfer has been done successfully
- ERROR: it indicates that an error has occurred
- REPLY: it indicates that data was not ready

- SPLIT: it indicates that data was not ready

- **HRESET_N:** reset of the master core

- **HCLK:** clock of the master core

- **HRDATA [31:0]:** the data from the read bus that is read from a slave during a read operation

- Outputs for AHB-Master are:

- **HBUSREQ_x:** a signal from the bus master to the bus arbiter to indicate that the bus Master needs to send data through the bus.
- **HBUSREQ_x:** a signal from the bus master to the bus arbiter to indicate that the bus Master needs to send data through the bus.
- **HTRANS [1:0]:** indicates the type of the current transfer. It can be:
  - Non-sequential
  - Sequential
  - IDLE
  - Busy
- **HADDR [31:0]:** 32-bit address
- **HWRITE:** if high, it indicates the write transfer and if low, it indicates a read transfer
- **HSIZE [2:0]:** indicate the size of the transfer
- **HBURST [2:0]:** indicates if the signal is a burst transfer

- **HPROT [3:0]:** provide information about bus access and it is for protection control
- **HWDATA [31:0]:** The write data that will be sent from master to slave through bus in write transfer

- **The signals of Data interface of IBEX:**

Data-Side Memory Interface

Signals that are used by the LSU:

Signal	Direction	Description
`data_req_o`	output	Request valid, must stay high until `data_gnt_i` is high for one cycle
`data_addr_o[31:0]`	output	Address, word aligned
`data_we_o`	output	Write Enable, high for writes, low for reads. Sent together with `data_req_o`
`data_be_o[3:0]`	output	Byte Enable. Is set for the bytes to write/read, sent together with `data_req_o`
`data_wdata_o[31:0]`	output	Data to be written to memory, sent together with `data_req_o`
`data_gnt_i`	input	The other side accepted the request. Outputs may change in the next cycle.
`data_rvalid_i`	input	`data_err_i` and `data_rdata_i` hold valid data when `data_rvalid_i` is high. This signal will be high for exactly one cycle per request.
`data_err_i`	input	Error response from the bus or the memory: request cannot be handled. High in case of an error.
`data_rdata_i[31:0]`	input	Data read from memory

The signals of Instruction Interface in IBEX:

Instruction-Side Memory Interface

The following table describes the signals that are used to fetch instructions. This interface is a simplified version of the interface used on the data interface as described in Load-Store Unit. The main difference is that the instruction interface does not allow for write transactions and thus needs less signals.

Signal	Direction	Description
`instr_req_o`	output	Request valid, must stay high until `instr_gnt_i` is high for one cycle
`instr_addr_o[31:0]`	output	Address, word aligned
`instr_gnt_i`	input	The other side accepted the request. `instr_req_o` may be deasserted in the next cycle.
`instr_rvalid_i`	input	`instr_rdata_i` holds valid data when `instr_rvalid_i` is high. This signal will be high for exactly one cycle per request.
`instr_rdata_i[31:0]`	input	Data read from memory
`instr_err_i`	input	Memory access error