****

**CSCE330402 - Digital Design II**

**Project 2 Report**

**Final Deliverable**

**Fall 2020**

Submitted By:   Nouran Abdelaziz   900170863

                          Ashrakat Elkhalifa    900170051

Amr Abdelazeem 900172775

Hussam Gawish 900171054

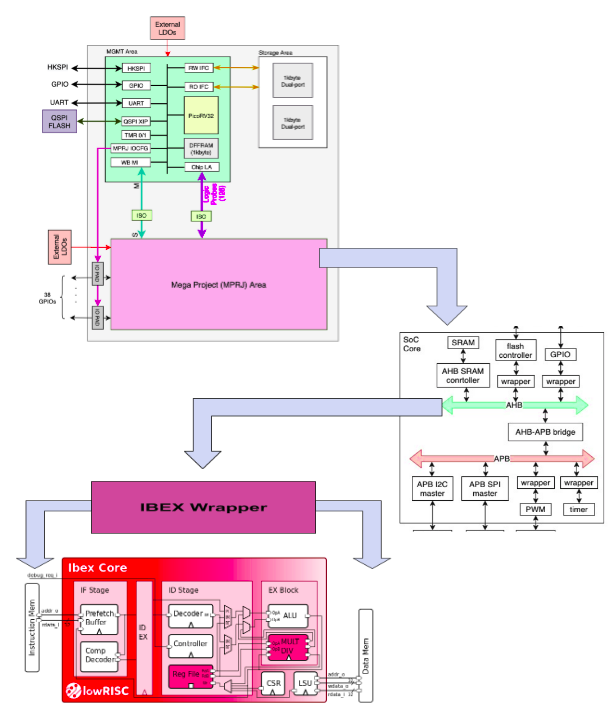
1. **Project Description:**

This project is to design an Ibex based Soc chip using open sources.

1. **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.

1. **Design Architecture:**



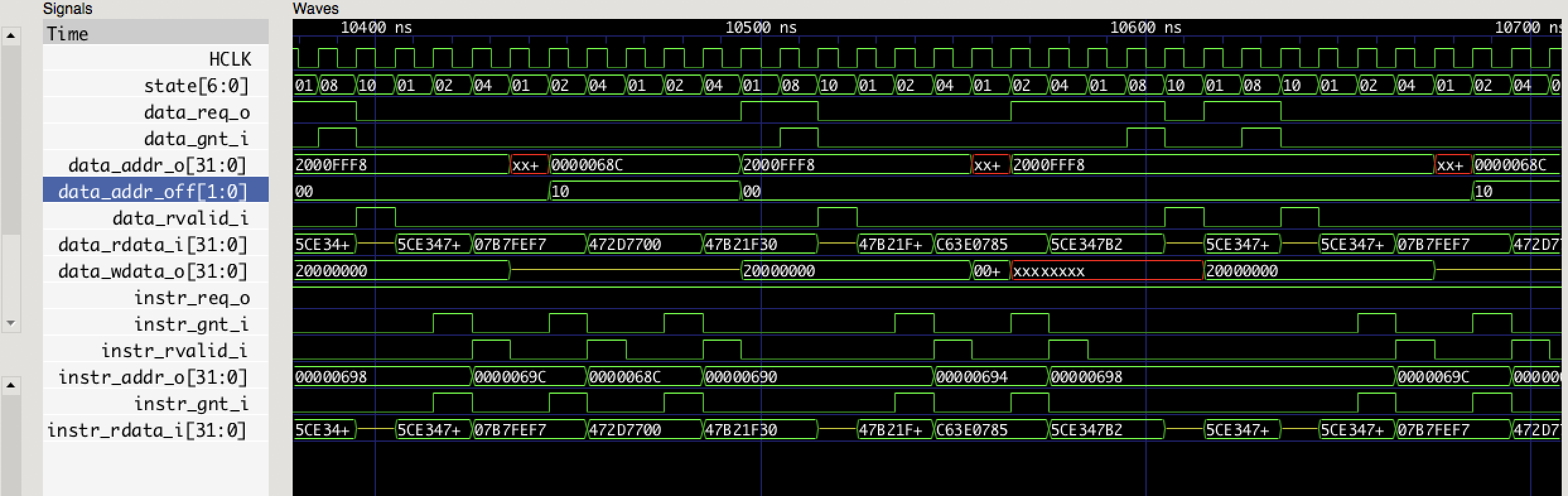
1. **IBEX Hardening:**

* Description:
* 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. We have generated all the three stages and their verilog codes of the IBEX Core.
* Configurations used in OpenLane:
* Synthesis:
* SYNTH\_STRATEGY: This indicates the strategy used in ABC’s logic synthesis and technology mapping. Strategy 1is used to optimize for the delay while strategy 3 to optimize for less area. We set it to 1 to optimize for the delay to have less slack
* SYNTH\_READ\_BLACKBOX\_LIB: This variable is used to allow for reading the full untrimmed liberty file as a blackbox for synthesis. This was used for the liberty file used in prim clock gating to be read as a blackbox. We set it to 1.
* Floorplan:
* FP\_CORE\_UTIL: It’s the core utilization percentage.It determines how much of the area is occupied with cells. It’s default value is 50. We needed to decrease core utilization to 45 to have better placement and routing
* FP\_ASPECT\_RATIO: It’s the core ratio, it’s the ratio between length and width of the die core. It’s default value is 1 and we set it to 1
* DESIGN\_IS\_CORE: It determines whether the design is the core of the chip or just a macro inside the chip in order to control the usage of layers used in power ring. 1 is used to determine that the design is the core of the chip and 0 if the design is a macro. It’s default value is 1, so we set it to 1 as IBEX was hardened to be a macro in our system.
* Placement:
* PL\_TARGET\_DENSITY: It’s used to determine how dense the cells are placed in the die. 1 is used for high dense while 0 is used for low dense. The default value is 0.4 and we set it to 0.4
* Routing:
* GLB\_RT\_MAX\_DIODE\_INS\_ITERS: This variable is used when we use DIODE\_INSERTION\_STRATEGY = 3 for FastRoute Antenna violations. It determines the maximum number of iterations that FastRoute should perform before stopping. In each iteration, the violations are detected and FastRoute fixes them by inserting diodes and producing a new DEF file. It’s compared to the previous iteration. If it has more antenna violation, the DEF file from the previous iteration is used for the rest of the run. If the antenna violations reaches zero, the iterations are stopped and the run is completed with this DEF file. We set it to 3
* Flow Control:
* DIODE\_INSERTION\_STRATEGY: it determines the strategy for diode insertion in design. There are four strategies: 0 which indicates that there are no diodes placed. This strategy yields very high antenna violations. The second strategy (1) which diodes are sprayed. Although this strategy yields the least antenna violations, it produces a huge number of cells. The third strategy(2) is to insert fake diodes and replace them with real ones if needed. Strategy (3) used FastRoute Antenna avoidance flow. Both strategies 2 and 3 yield optimization between the number of cells and antenna violations. We set it to 3.

1. **IBEX Wrapper:**

* This wrapper is used to make IBEX act as an AHB master to be compatible with the system generated by SocGen and be able to communicate with the AHB bus.
* The inputs and outputs of ibex core are mapped to those of AHB bus accordingly.
* To manage the two master interfaces of ibex, the data interface and instruction interface, the data\_grnt and instr\_grnt are set according to data\_req and instr\_req. In this wrapper, the priority is always given to data requests.
* Phases explanation:
* As long as there are no transaction requests from either the data or instructions, this is the idle state (S0) where it keeps checking for requests and sets the next state according to the requests. If both requests are on then the priority is given for the data transaction.
* The first phase is the address phase which is represented by the S1 (instruction) or S3 (data) states according to which request was granted in the idle state. In the address phase, we set the address, transfer type, write enable, and the data size and address offset are set according to the byte enable given by the ibex core.
* The second phase is the data phase which is represented by either S2 (instruction) or S4 (data) which is extended as long as the data is not valid and when is valid and transaction done then it returns back to idle state.

1. **SOC Simulation:**

****

1. **Caravel:**

* Configurations used:
* Most of the configurations in caravel are pre set and cannot be changed. Such as the area configurations, power and pin configurations.
* The only thing we changed is the synthesis strategy which we made 1 to optimize the delay.
* Inputs/Outputs ports:

|  |  |
| --- | --- |
|  | I/O Ports |
| FDO | 0-3 |
| FSCLK | 4 |
| FCEN | 5 |
| GPIO | 6-21 |
| SCL | 26 |
| SDA | 27 |
| MSI | 28 |
| MSO | 29 |
| SSn | 30 |
| SCLK | 31 |
| PWM | 32 |