Project Digital Hardware

Contents

Introduction

In the lectures Computer Architecture and Organisation the ARC processor is used. In the lectures Digital Hardware you had an introduction in the hardware description language VHDL, simulation and synthesis tools.
In this project you have to design and realize on the DE1-SoC your own simple processor. For the realization each group will have a DE1-SoC and no additional peripherals are to be used. Therefore also additional debug facility is to be realized in the DE1-SoC.

Requirements

1. You have to design a processor; you can choose: 1-address machine (accumulator) or a 2-address machine. **Not** a 3-address machine.
   See https://www.geeksforgeeks.org/computer-organization-instruction-formats-zero-one-two-three-address-instruction/ (accessed 27 September 2021)
2. Word size; you can define your own word size. What is a proper size for your instruction set?
   a. 16 bit seems sufficient to me, but it is your decision.
3. The processor description:
   a. It must have a separate entity-architecture for the data path.
      A schematic of the data path must be at such a level that it is easy to understand by an outsider (e.g. the teacher)!
   b. It must have a separate entity-architecture for the controller
      For the ARC processor a micro controlled controller is available. You may use a FSM description for the controller.
   c. It must have a separate entity-architecture for the processor
   d. The processor must have a test pin. If the test pin is active the current instruction is finished, but the next instruction is postponed until the test pin is deactivated. (The test pin is used for debugging.)

e. Main memory is byte addressable and aligned addressing is used.
   Note: to keep it simple use word size of 8 or s 16 bits. If 16 bits then the data/instruction is in two consecutive memory addresses. To read/write from main memory the least significant bit is 0.
f. After reset the first instruction is read from main memory address decimal 0.

4. Main memory: we keep it simple. The DE1-SoC has SDRAM, but communication with the SDRAM is not easy. Therefore the main memory is also to be realized in the FPGA.
   Note: this will limit the size of the memory (number of address bits).
   a. A separate entity-architecture for main memory is required.
   b. You may model the main memory such that it fits the word size of the processor. I.e. if word size is 16 bits the main memory consists of two parallel byte addressable memory.
5. Application: you have to come up with a very simple (!!!) application (program) for your processor and based on that define an instruction set.
   Notes:
   a) For the application you have limited I/O (7-segment displays, individual LEDs, switches and buttons)
   b) It must be easy for others to write another application for your processor. Documentation of your instruction set is important.
6. Instruction set: define your own instruction set (~10 instructions). It must include at least a "branch always", "branch on a status bit (one status bit is sufficient)", "load from main memory" and "store to main memory"
   Notes:
   a. A regular instruction set will ease the design!
   b. The ARC has a fixed length instruction, i.e. instruction and data are 32 bits. A variable length instruction is possible, e.g. is word size is 8 bits, then an instruction is a multiple of 8 bits (http://www.mathcs.emory.edu/~cheung/Courses/255/Syl-ARM/6-CPU/risc-cisc.html accessed 27-9-2021)
   c. How to indicate during simulation and synthesis that an application is finished?
7. Debug facility; during the design but also on the FPGA it must be possible to verify the correctness. For this it must be possible to at least read data from main memory from an address.
   Note: Be creative! In test mode the processor must be suspended ("test pin"). Now the debug environment can read the data from main memory. Show address and data on that address on the 7-segment display (not necessarily at the same time). Would be nice that you can easily retrieve the data on the next addresses (without entering these addresses explicitly). Also step mode would ease debugging (one instruction is executed).
   Notes:
   a. you can use the switch buttons to enter an address.
   b. If the word size is large you cannot probably not show the whole data at the same time (e.g. show lower bits, and when a button i
8. For the processor (excluding memory/debug environment) the timing information must be given (INTEL/Quartus timing analyzer).

9. Only upload design document, relevant VHDL files, QSF-filed and SDC files and a scriptfiles for modelsim (see as example the script files used for the model of the ARC processor). It must be possible to reproduce you test with ModelSim-INTEL. Do NOT include the “work” library, I assume your script file will compile everything.

Suggestions

1. Before you write code check if the data path can really do what is must do!
2. For the controller you can use the FSM model used in the lecture (process for the next state and for the output function).
3. For the ARC processor a VHDL model is available (http://iiusatech.com/murdocca/CAO/VHDL.html). This model is closely related to the description in the book. The VHDL description of the ALU in the ARC processor is at the behavioral level.
4. Use enumeration types for readability, e.g. if the ALU can perform operations add, mul, and, or you can model this with resp. 00, 01, 10, 11. But for readability and debugging a enumeration type can be used: `type aluop_tp is (alu_add, alu_mul, alu_and, alu_or);`
   In the control you can write: `aluop <= alu_add;`
   In the data path:
   ```
   case aluop is
     when alu_add => …
   ```

- How is the program stored in main memory? An option is to store the program in a separate entity-architecture, e.g. LoadProgramSyn.vhd. After a reset of the whole system the data is read from this design and stored in main memory. After the program is stored in main memory, the processor reset is released. Think about it! As a suggestion for the program:

```
49  TYPE PrgTpElm IS RECORD
50    address : natural;
51    content : string (1 TO 4); -- hex string
52  END RECORD;
53
54  TYPE PrgTp IS ARRAY (natural RANGE <>) OF PrgTpElm;
55
56  -- Here the program; aligned adressing! Adresses must be multiple of 2 (word size 16 bits)
57  constant prg : PrgTp :=
58    ( 0 =>(0,"8204"), -- here in comment the instruction (I removed mine :))
59      1 =>(2,"1405"), --
60      2 =>(4,"1507"), --
61      3 =>(6,"0405"), --
62      4 =>(8,"9420"), --
63      5 =>(10,"F000")  -- halt
64     );
65  -- end of program
```

`address` is the address in main memory. I used 16 bits for an instruction, therefor two memory addresses for an instruction. In my solution I used hex notation.
Another program? Only change the constant `prg`.
Notes:
a) be sure that the design with the program is synthesizable; because it is also synthesized (otherwise you cannot demonstrate it on the FPGA).

b) if you have time … you also make an assembler instead of writing the program in machine code ☺

Project planning and deliverables

Week 6:	Introduction in the project; composition of teams on Canvas (enroll yourself); max. 5 students in a group; each group 1 DE1-SoC.
Week 7;	Brainstorm sessions of the group. This must result in the instruction set, application, and global data path (schematic) of the processor. And how the debugging approach (text only).
Monday week 8	Upload on Canvas a document with the **project plan**: - Table of contents - Introduction - Application - Instruction set (including the instruction format (machine code)) - Global schematic of the data path of the processor - Test approach; how to add logic on the FPGA to debug you design/program (single step; reading data from main memory (it is not required to read data from register file). I don't think it is necessary to write data form the debug environment in main memory (but it is an option). - Planning and tasks for individual members - Conclusion
Week 9 and Monday in week 10	Upload at the end of each day a short summary of the progress; what went well, what are the problems, .. (it is not necessary to upload draft versions of design documents)
Week 10; Tuesday (end of day)	Design document: Upload a document with the design (incl. relevant Quartus RTL views, etc.). Add a chapter with globally the contribution of each group member. Add a zip with all relevant VHDL, QSF, SDC files and script files. Do not include files generated by the simulator / synthesis tool unless there is good reason to do so. **All documents in ONE zip file.**
Week 10; Wednesday	Short presentation on-campus: - Design process: impression of the design process - Design alternatives: Are design alternatives discussed? - Design decisions: Are decisions well-thought-off? - Quality of the demo
week 10; Wednesday' Evaluation report cooperation (**individual**)	Your individual reflection on the project (short) ~1/2 A4

Grading

When	What	Where	Weight
Monday week 8	Project plan	Canvas	Pass/Fail
Weeks 9 and Monday week 10	Daily reports	Canvas	*n*×0.25 grade penalty for missing *n* deadlines
Tuesday week 10	Design documents	Canvas	70%
Wednesday week 10	Presentation on Campus	Campus	30%
Wednesday week 10	Evaluation report cooperation (**individual**)	Canvas	0.5 grade penalty for missing the deadline

If the design cannot be simulated successfully the **maximum** grade is 7.0
If the design can be simulated but cannot be realized on the FPGA the **maximum** grade is 8.5

Final grade project:
Project plan passed: (0.7×design_doc+ 0.3×presentation) - penalties
Project plan failed: minimum ((0.7×design_doc+ 0.3×presentation) - penalties, 6.5)