Real-Time Imaging and Control
Lab

MSCV/ESIREM

Antoine Lavault
antoine.lavault@u-bourgogne.fr

Final Project

The goals of this project are:

- Show basic and advanced usages of VHDL for an embedded system description
- Show basic and advanced knowledge of FPGA development workflow,
- Show intelligent and well-covered descriptions using test benches and simulation
- Show creativity regarding applications and usage of the FPGA given a limited set of instructions.

1 Grading

You are expected to hand over a report where you describe your design process, the relevant result of simulation and physical testing, and a discussion on your choices. You are also expected to provide reproducible VHDL simulations of your results.

The first part is mandatory and aims at evaluating your skills in designing a system given a set of specifications while encouraging exploration through "free thinking" feature additions.

The second part lets you choose your own "project," given limited constraints. This part should be done if and only if the first one is done and completed.

The assignment must be handed over on Teams before the set deadline. No late submissions are accepted. A submission is not final until the deadline.

You are expected to provide a self-contained archive with all the relevant files with a `README` file indicating how to reproduce your results.

2 Designing a simple microprocessor in VHDL.

You will design a simple microprocessor with the following properties:

- Separate 8-bit input and output memory data buses
- 8-bit address bus.
- An accumulator and a program counter

Your CPU will execute an assembly code with a small set of simple instructions. Your implementation needs to include the following five instructions:

- LDA (data): Load the accumulator with "data." The instruction is encoded with two bytes: the first byte, represented as 0x01 in hexadecimal, serves as the opcode, while the second contains the operand data.

- STA (`addr`): Store the contents of the accumulator in memory at address "`addr`." The instruction is encoded with two bytes: the first byte, represented as 0x02 in hexadecimal, serves as the opcode, while the second contains the address data.

- ADD (`addr`): Add the contents at address "`addr`" to the contents of the accumulator. The instruction is encoded with two bytes: the first byte, represented as 0x03 in hexadecimal, serves as the opcode, while the second contains the address data.

- JNC (`addr`): Jump to the address "`addr`" if the "carry" flag is not set. Otherwise, proceed with the next instruction. Coded in two bytes: first byte (opcode): 0x04 and then the address

- JMP (`addr`): Jump to address "`addr`." Coded in two bytes: first byte (opcode): 0x05 and second byte (operand): `addr`.

The CPU is connected to an external memory. The memory interface includes an 8-bit address bus, separate 8-bit read and write data buses, and a write enable line (`wr_en`) for writing to RAM.

The CPU design includes the following internal 8-bit registers:

- The Program Counter (PC) tracks where the next instruction is in the memory.

- The Accumulator stores the result of the last operation

You may need to add additional special-purpose registers to implement some operations.

An asynchronous reset signal initializes all registers. After the reset is released, the program counter (PC) is set to a value corresponding to the default starting address of the program. When reset is released, the CPU reads and begins execution of the opcode stored at that memory address.

CPUs usually read and write to blocks of RAM. To simplify the design, you are provided code that emulates an asynchronous memory in the FPGA logic. You may configure this ROM as you like.

The memory is initially loaded with the following program:

```
0x00: LDA 0x07
0x02: ADD 0x0A
0x04: STA 0x10
0x06: JNC 0x02
0x08: JMP 0x00
0x0a: 0x09 0x00
0x0c: 0x00 0x00
0x0e: 0x00 0x00
```

1. What does the startup program do?

2. Create a CPU capable of running the program above.

3. The processor, as described up to this point, is straightforward, but it can be built out and extended in many different ways. Here is a list of possible processor function extensions, some simpler than others. You can discuss other ideas with the instructor.

    - Additional opcode ideas:

  - Simple arithmetic (subtraction, shift...)
  - Other types of conditional jumps
  - Load a pseudo-random number to the accumulator. The PRNG could be seeded by default, or you could implement a second opcode to load a seed to the generator.
  - Opcodes that let you jump to a subroutine (JSR) and then return to the main program after the subroutine's end. To keep it simple, only allow JSR to be called from the main program so that you only need to store one address to return to.
- Hardware peripheral ideas:
  - Switches to select which information is shown on the hex display of the board (accumulator, memory address, memory contents, etc.)
  - Special registers that allow the CPU to access peripherals. For example, reading switches/buttons, writing to LEDs, or communicating with PMOD peripherals (if we have the money to buy some, which is unlikely).
  - Serial communication with a computer or a device of your choice.

3 Challenge

3.1 Option 1: A real (but old) CPU

Once the first part is completed, you are tasked to choose one "old" CPU, like the Intel 8008, 8051, Intel 8080/5/8, MOS 6502, or Zilog Z80 (sorted in order of "difficulty").

You are free to choose how you design a compatible design with the CPU of your choice. Or even provide a multi-architecture design (e.g., the 8080 and the Z80 are known to be (almost) compatible with each other).

A modern application of these old CPUs with FPGA is the maintenance of old systems due to non-existent replacement, better power management, and potential bug fixes.

Some ideas:

- Reproduce an old computer from the 70s/80s: Commodore 64 or Atari 2600 (6502), Altair 8080 (8080), IBM 5150 (8088). Maybe with some modern capabilities, like Ethernet or USB? Or a less functional demo like running Doom or playing Bad Apple.

- Emulate old application-specific hardware like the Lexicon 224XL (8085).

3.2 Option 2: FPGA in robotics/computer vision

If your final project allows it (see with your supervisors), you can demonstrate FPGA usage with your project. Some applications could be:

- Hardware-accelerated processing (filtering, conversions, etc...). Overall, hardware acceleration for process optimization (offload computationally heavy tasks to the FPGA)

- FPGA for sensor fusion (FPGAs are quite fast and inherently parallel with high throughput capabilities; this can be useful when working with multiple sensors).

4 Project Management

Since you are working in groups, you should separate the workload between each member. However, the overall system design should be straightforward and simplified from the start as much as possible to make for an efficient project workflow. So, first thing first, make a drawing of the system and a drawing of each subsystem.

Also, since you will be working on different components simultaneously (hopefully), using a versioning system like git or subversion could be a massive helper. Or a PITA if you don't know how to use it, which shouldn't be the case.

Another point about test benches and general test-driven development. Since you'll likely be working on different things simultaneously and putting it all together later, having a set of test benches can be a godsend for seeing where a particular bug comes from. And deal with the culprit(s) accordingly.

Lastly, since you are expected to write a report, you better start writing soon. A collaborative environment like Overleaf or Google Docs can be a good idea to allow everyone in the group to write different parts simultaneously. This will avoid an inevitable crunch two hours before the deadline.