Project Report

Nicolas Blanchard, Axel Davy and Marc Heinrich

September 2012 to January 2013

1 Compiling the simulator

To compile, use ocamlbuild with the following options
-use-menhir -lib unix main.native. This generates the main.native program that will be used later.

1.1 Inner workings

The simulator is organised around a main function that handles the option list (with Arg.parse) and initiates the different independent steps in the following order :

- Netlist Parsing (via the given module)
- Transformation of the Netlist by scheduler
- Conversion into another format (mprogram) to improve later performance and facilitate debugging
- Execution of the mprogram.

1.2 Option list

syntax : main.native (options) filename

- scheduleonly : Stops after printing the result of scheduling
- n *integer*: Number of steps to simulate (default: -1. Does not stop)
- noprint : Does not print the result of scheduling
- debug : prints step by step and waits for user control
- clock : to work with the clock program
- clockfreq *float*: running frequency of the clock
- compare : also displays the computer time
- verbose : print all the outputs
- ramfile *address*: address of the ram file
- romfile *address*: address of the rom file

When launched with incorrect parameters, main automatically displays the command list.

2 Compiling the assembler

Sipmly use : ocamlbuild assembleur.native
To launch : ./assembleur.native -lines 256 *program.asm*

The Assembler language is described in Architecture.txt

3 The microprocessor

It is coded in the proc.mj file and has been precompiled into proc.net. It can be recompiled using MiniJazz

4 Using the clock program

The source code is in horloge.asm and has been compiled into horloge.rom.
A correct ram file (consisting of blank memory) is given : horloge.ram
To use the clock :
./main.native -ramfile horloge.ram -romfile horloge.rom -clock [-compare] [-clockfreq frequency] proc.net
compare shows the difference between the local unix time and the time simulated by our program. There is always a 1 second lapse at first during the launch phase. clockfrequency is by default at 1, and corresponds to the speed at which one wants to simulate the program. Maximum observed speed show a gain of two orders of magnitude. To launch the program at maximum speed one can use 0.0 as a value (some small optimisations have been made for this case).

5 Problems we have encountered

5.1 Part 1 : the simulator

After the first work session we had a scheduler function that did topological sorting and could output a sorted netlist file. However, taking the registers into account proved to be the hard part as it could conflict with the combinatorial loop detection. Also, two interconnected registers caused priority problems, so we chose to remove all links to the registers during the sorting phase, choosing to separate them as input-registers and output-registers, that are handled before and after execution of the netlist. Also, the "programme" type wasn't exactly to our taste and we created the Mprogramme type to add the register list, and to improve the equations by storing their arguments as keys in an array.

5.2 Part 2: Assembler and Processor

The initial idea was to add the division and modulo into the processor (as super-instructions), but as time grew short the changes that needed to be made to add it made the idea too time-expensive. They were however implemented as assembler functions.