Implementing a MIPS processor using SME

Carl-Johannes Johnsen

Department of Computer Science
University of Copenhagen

May 19, 2017

Background

The Machine Architecture class at DIKU taught the theory of computer organization and design. However, it did not teach how to construct specialized hardware, as one might implement on an FPGA.

FPGAs are more attractive than general purpose CPUs in some applications, as they do not necessarily have the same overhead in both performance and power usage, as they are not as complex.

However, FPGAs are programmed using Hardware Description Languages, which are very tedious to program. This has changed with SME.

Introduction

The SME programming model is similar to the CSP model, except it is globally synchronous, has broadcasting channels and a hidden clock. This makes it more suitable for generating hardware models than CSP. Additionally, SME can be transpiled into VHDL, which can be written onto an FPGA.

I am implementing a MIPS processor as taught in Machine Architecture by using SME, and documenting the process, so it could be used as teaching material for a course on hardware development.

Basic combinatorial

I started by implementing some basic combinatorial circuits, as this was an simple approach to SME. The first one I made consisted of four processes, which simulate four basic gates: AND, NOT, OR and XOR.

Then I connected these into networks: an decoder, a half adder and a full adder.

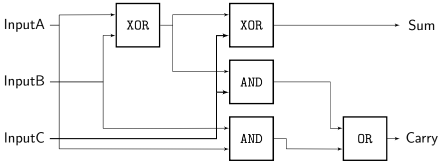
Basic combinatorial - Full adder example

```
public interface InputA : IBus {
    bool bit { get; set; }
}
...
public class AndGate : SimpleProcess {
  [InputBus] InputB input1;
  [InputBus] InputC input2;
  [OutputBus] Internal3 output;

  protected override void OnTick() {
    output.bit = input1.bit && input2.bit;
  }
}

public class OrGate : SimpleProcess {
  ...
  protected override void OnTick() {
    output.bit = input1.bit || input2.bit;
  }
}

public class XorGate : SimpleProcess {
  ...
  protected override void OnTick() {
    output.bit = input1.bit ^ input2.bit;
  }
}
```



Components

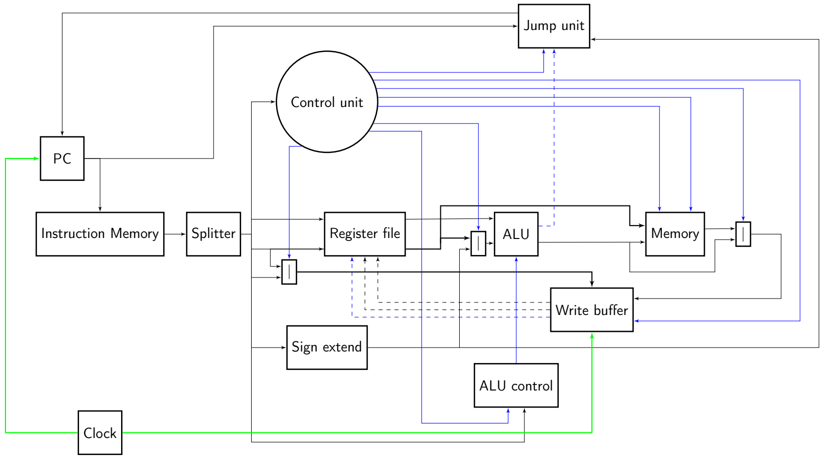
After getting some experience with SME, i started working on the MIPS processor. I started by implementing each of the components as SME processes, as I can then verify them individually.

```
public class Register : SimpleProcess {
  ...
  uint[] data = new uint[32];

  protected override void OnTick() {
    if (write.enabled && write.addr > 0)
      data[write.addr] = write.data;
    outputA.data = data[readA.addr];
    outputB.data = data[readB.addr];
  }
}
```

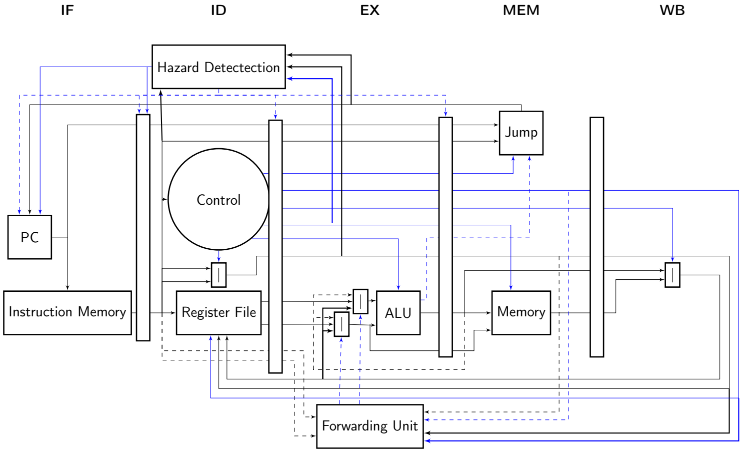
Single Cycle

With all of the components implemented and verified, 'wiring' up the processes is straightforward, as the busses should just be named accordingly.



Pipelining

Following the procedure from the Machine Architecture class, I have pipelined the processor, and handled the hazards introduced by pipelining with an hazard detection unit, and a forwarding unit.



Performance

To test both the single cycle processor and the pipelined processor, I have made some programs in MIPS assembler. To verify both the number of executed clock ticks and the results of the program, I have run them in the MIPS simulation program MARS.

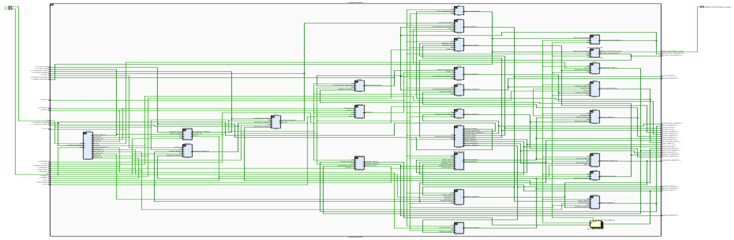
		# CT MARS	# CT	time (ms)	clockrate (hz)
Towers of Hanoi	$n = 5$	719	720 - 1058	516 - 1190	∼1395 - ∼889
Quicksort	$n = 8$	483	484 - 763	375 - 895	∼1290 - ∼852
Fibonacci	$n = 10$	85	86 - 126	113 - 212	∼761 - ∼594

Synthesizing

As mentioned SME can be transpiled into VHDL. Once the VHDL has been transpiled, it can be simulated by `ghdl`, in order to get a quick pointer on the correctness of the transpiled code. Once the code has passed simulation, it can be synthesized and implemented on a FPGA with the use of FPGA tools.

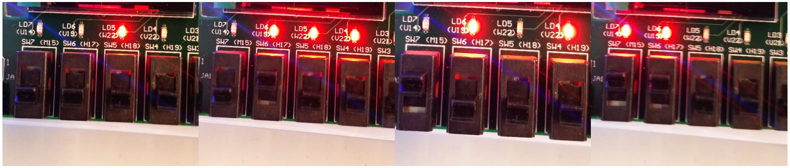
This is as far as I am with the project. So far, I have succesfully generated VHDL for all of the basic combinatorial networks, individual components and the single cycle processor. However, the processor fails synthesization.

Synthesizing - Single cycle processor RTL



Synthesizing - Logic gates

I have only managed to fully implement the first SME network, which consisted of the four basic gates.



Future Work

- Synthesize and write single cycle processor to FPGA
- Synthesize and write the pipelined processor to FPGA
- Fit more cores onto a single FPGA
- Introduce multiple execution paths