Building A RISC Microcontroller in an FPGA

- Name : Yap Zi He
- Course : 4 SEL
- Supervisor : PM Muhammad Mun’im Ahmad Zabidi
Introduction

- **Reduce Instruction Set Computer (RISC)** is a new trend on computer design. The opposite and rather established trend is **Complex Instruction Set Computer (CISC)**, which consist lots of instruction sets, addressing modes, instruction formats and a number of possible instructions length. The complexity of the instruction makes the design more complicated. While RISC tend to reduce the complexity of the instructions set and thus make the design process easier.
Objectives

- The objectives of my project is to design a **RISC Microcontroller** using **VHDL** and implement it in an **FPGA**.
- The instructions set and features are based on ATMEL AVR AT90S1200 RISC Microcontroller.

**AT90S1200 At First Glance**

- 89 Instructions
- 32 General Purpose Registers
- 2 IO Ports (15 pins)
- 1 x 8-bit Timer
- 1 x External Interrupt
- 1 x Analog Comparator
Scopes

- Implement the Complete AT90S1200 Microcontroller in an FPGA
- FPGA unimplementable features (analog comparator and internal pull-up resistors) and unnecessary feature (WatchDog Reset) will be ignored

What is the Challenges?

1. **AREA** – the design must be able to fit into the targeted FPGA device, which is Altera EPF10K20RC240-4.
2. **SPEED** – the microcontroller must be able to run at a minimum clock speed of 10MHz.
Project Background

“FPGA Implementation of a RISC microcontroller” – Wan Mohd Khalid

Problems:
1. Project is poorly documented
2. Design using both VHDL and schematics
3. Only 50% instructions designed (behavioral approach)
4. No peripheral features (port, timer, ext int)
5. Project size is too large, require 3 pieces of EPF10K20!
6. FPGA implementation is not done
Approach

Software (VHDL)

- Architecture & Instructions Set
- VHDL Coding (Text Editor)
- Synthesis (FPGA Express)
- Compilation & Simulation (MaxPlus II)

Program Coding & Assembling (AVR Assembler)

ASM to MIF Converter (Self Written)

Simulation (AVR Studio)

Compilation & Device Programming (MaxPlus 2)

Results Checking

Hardware Implementation

Program

Designed Microcontroller
Architecture Overview

* All IO Ports are 8-bits width (3 ports = 24 pins)
* External Interrupt and Timer external clock source share the same pin with pin D7
Instructions Cycle / Pipelines

<table>
<thead>
<tr>
<th>Clock Cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1\textsuperscript{st} Instruction</td>
<td>Fetch</td>
<td>Execute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2\textsuperscript{nd} instruction</td>
<td>Fetch</td>
<td>Execute</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3\textsuperscript{rd} instruction</td>
<td></td>
<td>Fetch</td>
<td>Execute</td>
<td></td>
</tr>
</tbody>
</table>

The 2 stage pipelines allowed 1 instruction to be executed on every clock cycle.
Designing the control unit is usually the most tedious task in digital system design.

In my design, the control unit is implemented with **Synchronous Mealy Model**.

The advantages are:

1. The control signals are asserted according to clock transition rather than generated by the state as in Moore Model. Thus the control signals appear faster and will speed up the microcontroller.

2. The control unit for the microcontroller has only **8 states**! This is because control signals are lock to transition rather than the states. So all single cycle instructions can share the same state.
What is Synchronous Mealy Model?

**Input** (Instruction Code, IRQ, Branch Taken? ...)

1. Combinational Logic for **NEXT STATE**
2. Flip Flops
3. Clock
4. Current State

**Next State**

5. Flip Flops
6. Combinational Logic for **OUTPUT**

**Output** (Control Signals)

7. Flip Flops
8. Clock
8 States Control Unit

- **Reset** → **EXE State**
- **No IRQ** → **EXE**
- **IRQ** → **SLEEP**
- **Wait for Interrupt**
- **Load Inst** → **EXE**
- **Store Inst** → **EXE**
- **CBI/SBI** → **EXE**
- **SBIC/SBIS** → **EXE**
- **Conditional branch taken** → **Branch1**
- **Unconditional branch** → **Branch2**
- **Single Cycle Instructions**
- **Branch Instructions (3 cycles)**
- **2 Cycles Instructions**
The designed microcontroller is capable of executing 92 instructions.

**Arithmetic and Logic Instructions (20) :-**
ADD, ADC, SUB, SUBI, SBC, SBCI, AND, ANDI, OR, ORI, EOR, COM, NEG, SBR, CBR, INC, DEC, TST, CLR, SER

**Branch and Skip Instructions (32) :-**
RJMP, RCALL, RET, RETI, CPSE, CP, CPC, CPI, SBRC, SBRS, SBIC, SBIS, BRBS, BRBC, BREQ, BRNE, BRCS, BRCC, BRSH, BRLO, BRMI, BRPL, BRGE, BRLT, BRHS, BRHC, BRTS, BRTC, BRVS, BRVC, BRIE, BRID

**Data Transfer Instructions (10) :-**
LD Z, LD Z+, LD -Z, ST Z, ST Z+, ST -Z, MOV, LDI, IN, OUT

**Bit and Bit Test Instructions (28) :-**
SBI, CBI, LSL, LSR, ROL, ROR, ASR, SWAP, BSET, BCLR, BST, BLD, SEC, CLC, SEN, CLN, SEZ, CLZ, SEI, CLI, SES, CLS, SEV, CLV, SET, CLT, SHE, CLH

**Misc Instructions (2) :-**
NOP, SLEEP

> 20 + 32 + 10 + 28 + 2 = 92 instructions
Results Achieved

1. **VHDL coding** and **simulation** for all the modules in the block diagram has been done successfully.
2. The design has been **programmed into FPGA** successfully.
3. The FPGA is now a microcontroller and is able to **control some real applications** (will be demo).

How about the challenges stated?

**AREA** - The design **fit into the targeted device** successfully (using 92% of EPF10K20RC240-4)

**SPEED** - The final design clock speed is **12 MHz** (target is 10MHz)
## AT90S1200 VS My Design

<table>
<thead>
<tr>
<th></th>
<th>AT90S1200</th>
<th>RISC MCU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instructions</td>
<td>89</td>
<td>92</td>
</tr>
<tr>
<td>G.P Registers</td>
<td>32</td>
<td>16</td>
</tr>
<tr>
<td>Program ROM</td>
<td>512 words</td>
<td>512 words</td>
</tr>
<tr>
<td>IO Ports</td>
<td>2 (15 pins)</td>
<td>3 (24 pins)</td>
</tr>
<tr>
<td>Addressing Modes</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>Speed</td>
<td>4 MHz / 12 MHz</td>
<td>12 MHz*</td>
</tr>
<tr>
<td>8-bit Timer</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>External Interrupt</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Analog Comparator</td>
<td>1</td>
<td>None</td>
</tr>
<tr>
<td>Implementation</td>
<td>CMOS</td>
<td>FPGA</td>
</tr>
<tr>
<td>SRAM</td>
<td>None</td>
<td>128 bytes</td>
</tr>
</tbody>
</table>

* 18 MHz can be achieved when implemented in the fastest device from the same family
Recommendations on Future Works

My design is based on AVR AT90S1200. There are many more members in the AVR family which contribute to more instructions and features. The next step will be adding these instructions and features into my design. Example:

Instructions – MUL, DIV, IJMP ...
Features – UART, SPI, 16-bit Timer (with input capture, output compare & PWM)
Conclusion

- The project has achieve all the objectives and fulfilled all the scope specified
- My main contribution is the VHDL source code, which is an AT90S1200 compatible RISC microcontroller