Complex Digital System design

RISC-V Processor

Prof : Dr Suresh Purini -Sai Manish and Sukumar

Before going into details of the project,One must know what is an ISA and what are different types of ISA's available.

Instruction Set Architecture(ISA) : It is a part of processor visible to programmer.It serves as a boundary between Hardware and Software.It provides the commands to the processor, to tell what it needs to do.The instruction set consists of addressing modes, instructions, native data types, registers, memory architecture, interrupt, and exception handling, and external I/O.

Examples of ISA's are :

- RISC-V (A RISC architecture)
- x86(A CISC architecture)
- MIPS(A RISC architecture)

RISC - reduced instruction set architecture.

CISC - Complex instruction set architecture.

Main difference between RISC and CISC is the number of computing cycles each instruction takes.

Here in our project we are designing a processor which supports RISC-V ISA.

In general,steps in Instruction Cycle(execution of instruction):

- Fetch (to get the instruction from memory)
- Decode (decodes the instruction i.e determines what actions the instruction dictates)
- Execute(carries out those actions)
- Memory Access(if needed memory is accessed)

- Write back(update the results in register file)

There are various ways to design instruction execution.

- Single cycle(executing instruction in a cycle)
- Multi cycle(executing instruction in more than 1 cycle)
- Pipelined(executing multiple parts of instruction cycle in a clock cycle)

In this project we designed Single cycle, Two stage pipeline.

Different Types of Instructions:

- Arithmetic and logic(add,sub,bitwiseAND etc)
- Control transfer(jump,branch)
- Memory(load,store)
- System(CSR i.e to execute syscalls).

In this project we had designed a one cycle and two stage pipelined processors based on RV32I ISA(Instruction Set Architecture) in which we had only included Arithmetic,Control transfer,logical,Memory instructions.We coded this using bluespec.Before trying to design a risc-v processor one must know about bluespec,risc-v ISA (addressing modes etc).References are mentioned at the end.

This processor is deployed on FPGA so we don't have RAM's etc but instead we have BRAM's(refer to FPGA BRAM's) where we can load the file containing instructions.

We used harvard architecture in the design as pipeline is possible and instruction can be completed in single cycle. Harvard architecture consists of data memory and instruction memory separately. Initially a .vmh file (mem file helps in memory mapping) is loaded into Instruction memory (Imem) and Data memory (Dmem) . Imem consists of instruction to be fetched and Dmem is used to retrieve data required and write back once the instruction is done. In Imem and Dmem modules we used Brams instead of Regfile as Regfile uses a huge amount of LUTs (Look up tables,refer to FPGA LUTs) for .vmh file initialisation.

Fetching Instruction:

Each instruction is fetched from Imem and the PC is incremented by 1 as we are taking the whole 32 bit at a time.

Decoding Instruction :

As the fetched instruction is then decoded based on the addressing modes in RV32I (addressing modes are present in the RISC-V manual and follow the link given below)

https://content.riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

We may want to add tagged bit so that we can know whether the corresponding registers are valid for that instruction or not.

Executing Instruction :

In the execution , we get the decoded instruction,allocate the registers from register file and based on that we have to operate i.e if that's an R type instruction then we might want to compute the output by using alu functions.

Memory Access :

If the instruction is of type load or store we need to access memory.Here we accessed memory file using Dmem(which contains data).

Write Back :

Here we updated these values back in the register file.After that incremented program counter to fetch next instruction.

Designing a One Cycle Processor :

As mentioned above What we did was all the above parts in instruction cycle are included in a single cycle.

Designing a Two Stage Pipeline Processor :

The two stages we designed are { fetch+decode , execute+memory access + write back }(please look into pipeline designs for details about stages).First and Second stages are fired in parallel and critical path time is calculated by the maximum time taken in one of the stages.

Later we did a performance analysis on our design to compare onecycle and twostage pipeline processors using vivado. Finally we implemented our design on FPGA .

Verification of a Processor Design:

Given a C code, first we have to compile it to an assembly code and then to an executable with the help of RISC-V toolchain. After that we need to create a .vmh file with the help of elf2hex tool present in the riscv toolchain (required risc-v toolchain commands are given at end of the report) and feed as an input to the fetch module.

we can compare the results obtained from our design to that of an output obtained from the spike tool .

Challenges Faced:

During the synthesis of the design the usage of LUT’s is around 90,000 due to the use of RegFile module for I-Memory and D-Memory in order to reduce the usage of LUT’s we used BRAM for loading a vmh file.

Due to the use of BRAM there has been increase in no of clock cycles per instruction. As in order to access BRAM first we need to put a request and then get the response in the next clock cycle, so there has been an additional delay as a result no of clock cycles per instruction increased.

Implementation of TwoStage Design on FPGA:

Before moving to implementation let us understand how to use vivado, the verilog files of the design are generated using bluespec (how to generate verilog files using bluespec are mentioned at the end of report) and are added along with the mem files to the newly created project file in vivado. Then a constraint file is added to the project which helps us to do the timing analysis in implementation and to map the I/O pins in the top module to the I/O pins present on the FPGA board which is needed to generate the bitstream. After adding the source files and constraint files then we simply need to run “Generate Bitstream” option present in vivado, if only timing analysis and other reports are needed then running implementation option in vivado is enough. Generation of bitstream is needed for FPGA implementation, after bitstream is generated then detect the device (FPGA) using “open target” in hardware management and then program the device as final step to dump the processor design.

We implemented a max function of a given data on FPGA and displayed the max value using the LEDs on the board. In Order to get the output we added an additional interface in the top module and also a constraint file so the output port can be assigned to LED pin on the board. Before implementation and generation of bitstream, we run a simulation of the design using vivado. Then generate bitstream follow the above procedure in order to program it on FPGA.

Performance comparison between OneCycle and TwoStage Pipeline Processor:

We have taken few assembly code like sort, max and sum of a given data and then synthesized and implemented using vivado. We mainly require timing analysis report and the LUTs usage of the design obtained from implementation.

Clock Time Period : 20ns (for both OneCycle and TwoStage)

OneCycle			
Program	**Worst Negative Slack (WNS)**	**Critical Path time**	**Wall Clock time = Clock period X No of Cycles**
Sort	3.323ns	16.677	20 X 3290
Max	3.052ns	16.948	20 X 563
Sum	3.102ns	16.898	20 X 548

TwoStage			
Program	**Worst Negative Slack (WNS)**	**Critical Path time**	**Wall Clock time = Clock period X No of Cycles**
Sort	4.433ns	15.567	20 X 3806
Max	4.547ns	15.453	20 X 719
Sum	3.773ns	16.227	20 X 674

How to Execute ?

After download of files from the github repository (the link is given at the end of the report) , enter the following commands in order to run the program.

1. For OneCycle Processor:

$make onecycle

This generates the verilog files of respective modules and an executable. Then run the executable Onecycle.bsim.

$./Onecycle.bsim

2. For TwoStage Pipeline Processor:

$make twostg

This generates the verilog files of respective modules and an executable. Then run the executable Twostage.bsim

$./Twostage.bsim

Glimpse of RISC-V Toolchain :

Procedure for Installation of risc-v toolchain is described in the link given below,

RISC-V Tools link: https://riscv.org/software-tools/

After installation of risc-v toolchain we can convert an assembly code to machine code or a C-code to a machine code. And even execute it.

let there be a hello world C-code in 'hello.c' file.

$ riscv32-unknown-elf-gcc -o hello hello.c

This command converts the hello.c into a binary file.

$ riscv32-unknown-elf-gcc -S hello hello.c

This command effectively takes the 'hello.c' program and compiles it as 32-bit program into assembly instructions that are saved into the 'hello.s' file.

```
$ riscv32-unknown-elf-gcc -o hello hello.s
```

This command takes the assembly code ‘hello.s’ and compiles it into a binary file.

```
$ spike pk hello
```

This command executes binary file ‘hello’. We can’t run ‘spike hello’, because our“Hello world!” program involves a system call. So we need to use pk - proxy kernel which services system calls.

```
$ riscv32-unknown-elf-objdump -D -S hello > hello.dump
```

This command helps us to generate a dump file for the executable.

```
$ elf2hex 4 65536 hello
```

This helps us to generate a hex file which is used in memory mapping for Imem and Dmem.

GitHub Repository Link : https://github.com/saimanishrao/RISC-V_Processor

References :

1. riscv.org

2. csg.csail.mit.edu

3. github.com/riscv

4. Bluespec Reference gide:
http://csg.csail.mit.edu/6.S078/6_S078_2012_www/resources/reference-guide.pdf