The Design and Implementation RISC-V based Instruction Set Architecture

By
Kushan Parikh (805654738)
Aditya Tumsare (805643566)

Abstract

Since its inception processors are widely used in each every filed possible. One can say that processors are the heart of any computing device. The working of processor mainly depends on the Central Processing units which in fact consist of different elements such as memory ,register ,control unit, program counter etc. the efficiency of the processor depends on the speed of execution of program which in fact depends on simplicity of programs. RISC-V is a simple yet powerful , open and modular instruction set and to realize ad to fully exploit the features of such instruction set architectures we need fast ,flexible and easily reconfigurable solution . This feature are provided by field Programmable Gate Arrays as compared to Application Specific Gate Array. This project focuses on the design and implementation of multicycle data path for RV32I which is the base instruction set of RISC-V ISA. The processor is developed and simulated using VHDL language. The designed processor will be implemented on Basys 3 FPGA board manufactured by Xilinx. The simulations were carried on Xilinx Vivado Suite. The functional simulation of RISC-V was successfully carried out.

Keywords : Application Specific Gate Array ,Field programmable gate array, Instruction Set Architecture ,Xilinx Vivado , Linux.

I. Introduction

The pace of development in fast processor is slowly going down because as the technology length is shrinking down short channel effects come into play. Nowadays VLSI circuit consist of millions and billions of transistors and therefore any small power drops leads to significant power dissipation. Therefore, there is emphasis on area where to make power efficient system in other words energy efficient circuits. Therefore, development of efficient processing unit is very important and critical.

In this project emphasis is given on designing of CPU based on RISC-V instruction set and implement it on Field Programmable gate array (FPGA). The main motivation behind the designing of CPU is to learn how data path and unit for a processor is built from the scratch using hardware description language. We know that design of real procedure involves complexity and many other factors. The main aim here is to concentrate on basics concept of building processors and not to focus on their applied use. Modern day processors are huge and complicated to study; therefore, such implementation will give us more deep insight about the functionality of the processors.

As RISC-V is booming the architecture world since its inception from 2011becasue it works well with any size of processor from complicated processors to small embedded processors form Field Programmable Gate arrays to Application Specific Gate Array. It is also a stable ISA that is it cannot be discontinued as it is governed by non-profit organization. Due to all this reason the choice of RISC-V instruction set architecture was obvious.

While designing the processor there should not be any constraint on the designer. The designer should also be aware of timing constraints in the design and the availability of the resources while designing the hardware. The circuits can be written into program statement nowadays. This language is known are hardware description language. For this project we are using VHDL which is used to describe structure and behavior of wide variety of digital systems. It is almost same as other programming languages but there is main difference which ih hardware the processes run on flight and not sequentially as compared to software's. But the use of language eases the process as compared to process with schematics and functional block diagrams. The tools used for the simulation of these languages also takes care of routing signals and timing constraints. And hence the use of for these projects.

In the design of processor, the first thing is to define how the instruction set are encoded and executed. Also, there are other factor that needed tpo be addressed such as opcode for every instruction, type of instruction, number of instruction and bit needed to encode these

instructions. After all this data path must be designed which can execute all this instruction along with control unit which controls this data path according to a data path. It also involves to decide on the number of arithmetic logic unit, Program counter, instruction memory and data memory accumulator register, sign extension unit, left shift register.

In the following sections the report describes about the literature survey and background research carried for this project . After that it explains about instruction set architecture followed by the explanation about RISC-V ISA. The RISC-V sections discusses why preference is given to RISC-V over other languages and talks about the 7 basic principles on which RISC-V has been developed. Further the report gives a short review about the RISC-V base ISA. Next the report discusses about the multicycle data path designed and its functional simulations. After that the report describes about the methods and tools used for this project which includes description about simulation tool Xilinx Vivado, Hardware description language VHDL and FPGA board used. Then the results and implementation are explained for building this project. Further the report describes about the timeline of the project and concludes describing future work and conclusion.

II. Related Work and Background Research.

RISC-V is open, and its future does not depend on the single corporation. It is managed by the non-profit organization RISC-V foundation. There are almost 60 organization headquartered in USA and outside USA who works independently on their project using RISC-V.

As RISC-V is a recently developed technology there are a lot of ongoing research and projects going on currently. 7th workshop for RISC-V organized by RISC-V foundation concluded recently in NOV 2017. In that workshop first part consisted of updates on RISC-V. In the second part there were presentations of the companies on their ongoing project consisting of RISC-V. Going through their research work we discovered how different companies are using RISC-V in a different manner on their product.

Further we refereed 2 books to study RISC-V Namely Computer Organization and Design RISC-V Edition: The Hardware Software Interface by David Patterson & John Hennessy and RISC-V an open architecture atlas by David Patterson and Andrew Waterman.

The first book focuses on hardware and software interfaces and give emphasis to he concepts and principle which serves as the basic of current computer working. This book explains computer organization based on realistic approach.

The second book gives short introduction about the RISC- ISA and serves ad reference guide to embedded programmers and students who wants to write code for RISC-V. It introduces RISC-V in less than 100 pages. The book has reference card which summarizes all the instruction and the book assumes that the reader is experienced with at least one instruction set. This book serves as quick guide for RISC-V students.

For the review of VHDL and to design processor using VHDL we referred Digital Systems Design Using VHDL by Charles H. Roth, Jr and Lizzy Kurian John. It served as reference to design RISC-V processor as MIPS processor design was explained in the book.

Along with books we went through several refence paper such as in paper [2] whole RISC-V instruction set architecture is explained. In the paper[4] the author describes RISC-V architecture as modular simple and explains in depth why it is modular and simple and comparing it with ISA. In the paper [5] the authors describe about the integration of reconfigurable fabric with modern processor. The paper describes about soft processor architecture which supports the extension of RISC-V add/multiply/divide and other atomic operations. The processor is designed to support Linux based shared memory systems.

In paper [6] the authors describe prototyping FPGA of a RISC processor for cost reduction in development and for the reduction in duplication of effort. The prototypes in paper [6]

are derived from the common source. In this paper the authors describe about the design RISC processor in several stages and its implementation in detail. Also, we refereed risc-v.org for instruction set manual and various other documentation.

III. Instruction Set Architecture

The instruction Set Architecture is a well-defined interface between hardware and software or in other words it is a part of the CPU which is visible to the programmer or compiler writer. It is a contract between hardware and software. The overall set of instruction a computer executes and dealing with the process that how it executes is known Instruction Set Architecture (ISA) which is a book of law for building any processor. It stores the functional definition of instruction operation, instruction modes and how the instructions are stored on the hardware along with the information to invoke them. However, the instruction set architecture does not give any information about the power and time required for any instruction to execute or how a instruction is implemented. The ISA should not be ambiguous.

There are mainly two types of ISA which are RISC (Reduced Instruction Set Computing) and CISC (Complex Instruction Set Computing). In CISC instructions are of variable length which made it difficult to prefetch and predecode. In RISC ISA the instructions are fixed length which makes prefetching and precoding easier. Hence the word reduced. Also in RISC memory access are restricted to load and store operation and other instruction accesses are registering to register while in CISC most of the instruction are memory access instruction. In RISC instructions only perform elementary operations as compared to elementary and complex operations in CISC. CISC does not have limited addressing modes as compared to RISC.

The factors such as good programmability ,it should implementable and controllability decides whether an ISA is good or bad. Programmability means the compiler should express the code to processor easily . the semantic gap should be as short as possible for this. The ISA should be easy to implement for techniques such as pipelining, out-of-order execution. But some features make it difficult for example in CISC variable length instruction make it difficult to predecode. Also, compatibility old hardware's must be able to run new software's and old software's must able to run on new hardware. This is called as backward and forward compatibility. Also, the main part is ISA are incremental that's why their size incases due to backward compatibility. This issued is removed in RISC-V architecture.

IV. RISC-V

RISC-V is open source free Instruction set architecture most Importantly is future does not depend on the fate of any company as it belongs to non-profit organization. This is one of the reason that RISC-V is one of the most privileged as compared to its predecessor and competitors and the reason for inception of RISC-V ISA. Also, most of the modern general-purpose instruction set architectures lacks two or more technical features. For example, MIPS [2] is not open source it is proprietary the instruction set architecture is over optimized, branch and jump wastes one clock cycle, wasting the bandwidth and thus making it complicate to implement superscalar.

Traditionally computer architectures are incremental in other words they must maintain backward binary-compatibility [1]. The reason for this is that newly developed processors must run or support old programs in short, they must be able to implement old ISA along with new ISA. This approach leads to not only in increase in size of ISAs but also the cost of expanding ISA as the new ISA contains past and present instructions. This again leads to the fact that if there are mistakes in the past extension it is inherently implemented in the present newly developed ISA. The developer or engineer must

pay for this cost. This can be observed form the graph shown in fig. 1 [1] as it shows the growth of x86 instruction set over the time. This leads to modularity of RISC-V ISA.

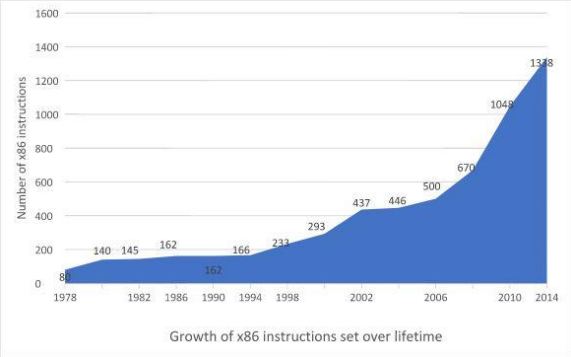

Fig.1 [1] Growth of x86 instruction set.

RISC-V is a modular ISA because the base ISA called RV32I is a full software stack and can never change in the future, it is locked. Depending on the necessity of the application whether it's a low power embedded application or power-hungry application the hardware can include or exclude standard optional extension provided by the RISC-V ISA. This modularity comes in handy such that the compiler generates the best suitable code for hardware depending on options inclusion of extension. This makes RISC-V a stable, modular and attractive target for operating system developers, embedded system developers, compiler developers and assembly language developers.

The underlying principles and tradeoffs in the design RISC-V ISA is discussed in following part.

1] Cost: The cost of integrated circuit depends on the die size and thus on the size of the wafer. It can be concluded that cost is a function of die area and it has linear dependency on cost. The more number of die produced from the wafer smaller the size of die area. The yield gets higher as the size of the die decreases. And lower the number of flaws, smaller the size of die. Thus, to keep the size of the integrated circuit small the engineer will always want to lower the size of processor which in turn drops the size of ISA needed to implement it. For example [1] ARM - 32 vs RISCV ISA, the area RISC-V Rocket chip processor compared to ARM 32 Cortex-A5 is half with same technology and the size of the caches for bot processor remains same. As a result, RISC-V requires less die area thus reducing the cost.

2] Simplicity: As mentioned above that engineer will want to implement simple design to reduce complexity. This saves the time required to design the chip and indirectly saves testing effort. Less complexity leads to decrease in die size and less time for documentation which adds to cost of the chip.

3] Performance: The iron law for performance is

Performance =Instruction count CPI Clock cycle time. [3]

If a complex ISA such executes less instruction than the simple ISA, with the help of this factor simple ISA can compensate for average or faster clocks per instruction. Also, simpler ISA are famous because they result in reduce die size, thus cost and also performance .in this way the cost- performance evaluation will be excellent.

4] Isolation of Architecture from implementation: The inclusion of instruction on ISA to improve cost-performance at the time of implementation affects future implementations. For example, the problems with MIPS-32 was delayed branches. This problem was solved by redefining branches, but it made the life of programmer hard as they must put something meaningful to fill up the delay slots. It not only made the program hard to understand but also due to its backward compatibility it became more complex to program. Thus, engineers must not enhance architecture for a particular implementation.

5] Room For growth: it has become extremely important for ISA to reserve opcode space for future applications such as augmented reality, machine learning and many more.

6] Program Space: Again, the smaller the program the simpler the program and smaller the size of chip. This is an important factor for embedded applications. Further smaller program lower rate of cache miss thus low power consumption as SRAM consumes less power than DRAM.

7] Ease of programming compiling and linking: More number of register makes it easy for the complier writer as register allocation is very important. the reason for this is register are easy and can be accessed faster than memory.

V. RISC V Base ISA

There are three different base instruction architecture of RISC-V which are RV32I, RV32E and RV64I. These ISA are different entities where RV32E is alternative of RV32I with few number of registers for power efficient design and deeply embedded designs. RV32I and RV64 has different register width and different memory address space size.

The RV31I is base 32-bit ISA with 47 instructions. Out of which eight instructions can be implemented as single trapping as they are system instructions thus reducing the size to 40 instructions. Even with small number of instructions it is enough to fulfill modern system requirements and can become compiler target. The remaining 40 instructions are divided in three categories namely memory access, computational and control flow.

RISC-V is a load-store architecture which means only load and store can access memory, other instructions performs operation on registers. It contains 31 general purpose registers. The width of each register is 32-bit. The program counter register holds the address of the current registers. The instructions are 32-bit long. The instructions are stored in little endian byte orders.

RV32I has six basic instruction formats which are R-type for register operations, S-type for store operations, J-type for jumps, B-type for conditional branch operations, I-type for load operation and immediate operations and U-type for long immediate operations. The instruction format can be seen from fig.2.

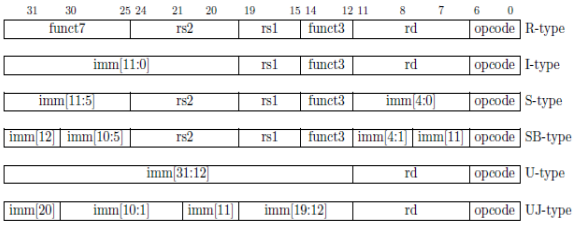

Fig 2 [2] instruction format.

All the instructions are 32 bits long and they offer three register operands as can be observed from the fig 2. Also, it has three register operands filed unlike the one which is shared between source and destination. Further before the decoding start the register can be accessed because the read and write are always in same direction. Furthermore, the immediate field is always sign extended. The advantages of the are decoding is very easy making it cost effective, second there is no need to preserve destination operand, third extra hardware for arithmetic is saved.

VI. Methods and tools

The basic approach in processing of instruction is that first is first is fetched from the instruction memory and decoded. After finishing of the decoding the operand from register to the memory to perform operation on it, which is also called as execution of instruction or execution phase. Further the results from the execution stage are written back into the register. The Instruction set architecture decides what happens to instruction and microarchitecture determines how these steps are going to be implemented by hardware.

There are two main part as discussed above in the designing of a processor which are designing of data path and control logic. The data path specifies the static structure of the structure it specifies the connection between different components which ensures smooth flow of data. The control unit in simple word controls these different components to ensure dynamic flow of the data.

Now based on the basic approach of processing and designed data path and control unit. The processor can process a single instruction or a multicycle processor. Therefore, the processor which processes a single instruction in a single clock cycle is known as single cycle central processing unit and likewise a processor who processes multiple instruction in a multiple cycle is known as multicycle CPU. The clock per instruction ratio for single cycle CPU is 1 and for multiple cycle CPU is greater than 1.

The time required to process an instruction in single cycle processor is same for all type of instruction irrespective of complexity of instruction. To ensure this we must make sure that slowest and longest instruction executes properly and comes out of the cycle. This means that every clock cycle takes equal amount of time. The slowest instruction rubs its speed to other instruction and this is one of the biggest disadvantages of single cycle CPU and is knowns as internal fragmentation. But the upside is is its implement.

The main advantage of a multicycle CPU over single cycle CPU is that we use variable clock cycle to execute every instruction depending on the complexity of instruction. For example, branch example takes only 3 clock cycles to execute as compared to load instruction which takes five cycles to execute. However, this increases complexity in the design of hardware. The control unit which was just a combinational unit now gets converted into a finite state machine.

One more major difference between single cycle CPU and multicycle CPU is length of clock cycle. As the length of clock cycle in single Cycle CPU is determined by the slowest instruction in multi-cycle CPU it is determined by the slowest functional unit which significantly reduces cycle time.

In single cycle CPU for a 5-stage pipeline in the first stage the processor fetches the instruction and updates the PC counter to fetch next consecutive instruction. In second clock cycle the instruction is decoded. In the next consecutive clock cycle execution of instruction is performed. I the fourth and fifth stage the memory is accessed, and results are written back. Therefore total 5 clock cycles are taken to execute any instruction. In parallel pipeline in the first stage the instruction is fetched. in the next clock cycle if there are no dependencies next instruction is fetched as soon as the previous instruction go into decode stage. In the next consecutive cycle first instruction enters execute sage and second instruction enters decode stage and third instruction is fetched. In this way, ideal pipeline processors processes instruction very clock cycle thus reducing the throughput. However, the latency remains same for both types of CPU and pipelined CPU achieves maximum throughput when all the stages of pipeline is filled. However, there are inconsistencies and complexities in implementation and design of multicycle CPU. The data hazards must be completely solved. Also, structural and control hazards should also be resolved.

a. Instruction Implementation

The processor we built in this project has certain architectural assumptions. The processor is 5-stage multi-cycle implementation. The data-path is designed for 5-stage pipeline processor. Also, this pipeline is ideal scalar pipeline, meaning the computations are assumed to be uniform.

The program counter (PC) fetches a new instruction every cycle by pointing to new location in memory. Which is calculated in the fetch stage of the processor, constructing an effective use of arithmetic logic unit. Furthermore, a 32-bit architecture of the RV32I ISA and byte-wise design allow the 4 to be added into PC for next instruction.

From the decode stage instructions gets their independent behavior.

1. R type Instruction

To do an arithmetic operation in the RISC – V, the base line architecture has R-type instructions. In the decode stage, the processor makes the decision about the type of instruction using the opcode.

Bits	31-25	24-20	19-15	14-12	11-7	6-0
Field	func7	rs2	rs1	func3	rd	opcode

Table 1. R - type Instruction Format

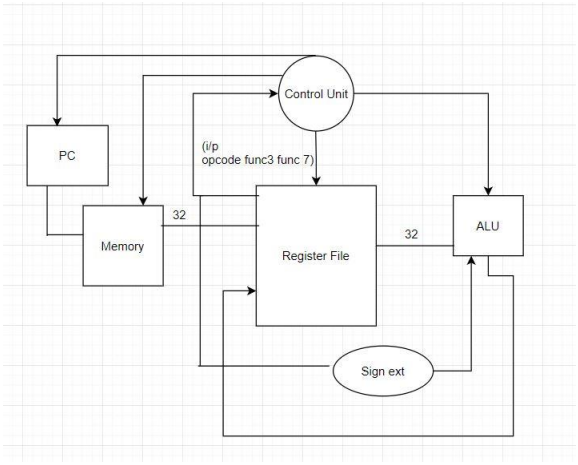

Fig3. R-type Instruction Implementation

Opcode for all the R -type instructions will be same. The func3 and func7 are the fields that will be used by the to differentiate between them. Either combination of both or single fields can determine the operation to be performed in the CPU. Other 3 fields rs1, rs2 and rd are source and destination registers. They all are the 5-bits long and contains the address of the register whose data is supposed to be used.

As shown in the figure, two 32-bit data values are being fetched from the register file to be computed in the arithmetic logic unit (ALU). Therefore the decode stage is also called the register fetch stage.

Next comes the execution stage, in which both source operands are computed depending on the input of the ALU control signal. That's where the fields func7 and func3 comes in play. Depending on them the ALU control signal has a value. In our case it is 2-bit long signal with possible operation addition, subtraction, "logical or" and "logical and". This operation then generates an output signal called ALU out.

Once the output is available, the value needs to be updated into the destination register. The write enable of the register file at a new clock will updated the ALU out value into the register file ate the location pointed by the rd field. This stage is called the write back stage. As there is no memory stage in the R – type instructions.

The R-type instructions are written as:
Opcode rd, rs1, rs2 /* format of the instruction
add r0, r1, r2 /* r0 ← r1 + r2
sub r3, r4, r5 /* r4 ← r4 – r5
and r6, r7, r8 /* r6 ← r7 (logical and) r8

2. I type Instruction

For all the instructions the fetch stage is going to be same irrespective of the type. From the decode stage will have different behavior. Difference in the I type instruction is that there is only 1 source register and 1 destination register. Where the immediate field is used along with the source to calculate the effective address for witch the operations needs to be performed.

Bits	31-20	19-15	14-12	11-7	6-0
Field	imm	rs1	func3	rd	opcode

Table 2. I - type Instruction Format

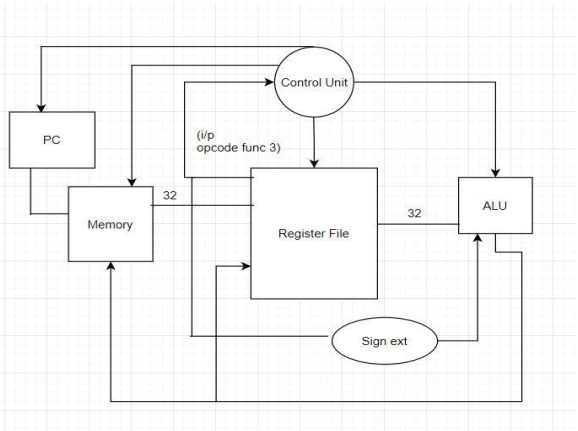

Fig 4. I - type Instruction Implementation

The decision about which bits of the instruction to be send as source and destination addresses in the register file is made by the opcode.

Once the operand data is available the ALU does the address calculation, for which the ALU control signal will always be the addition. However, once the location is calculated the ALU out will have an address, data of whom to be received by the destination register.

In this type of instruction, the memory stage do exist. The data will be fetched from the data memory and in the write-back stage that same data is written into the destination register assuming it is the load instruction. Otherwise, arithmetically computed data will be written in instructions like addi, xori etc.

The R-type instructions are written as:
1) Opcode rd, rs1, Imm /* instruction format
2) Opcode rd, Imm(rs1) /* instruction format
lw r0, 10(r1) /* r0 ← data(r1+10)
addi r3, r4, r5 /* r4 ← r4 + r5
xori r6, r7, 1 /* r6 ← r7 (logical xor) 1

3. U type Instruction

In the decode stage, destination register, and immediate values are selected. Thus, the new program counter is counted, which then is going to be used to fetch the new instruction. Depending upon the instruction the output is calculated may be through adding or shifting the immediate fields for which the opcode will be provide ALU control Signals.

There is no use of the memory stage in this type of instruction. The data will be directly used by the program counter or might be held into register file before being used by the program counter. Thus, the writeback stage will facilitates the writing of the calculated address into the destination register.

The U-type instructions are written as:

Opcode rd, Imm /* instruction format
LUI a4, 0x1234 # a4 ← (0x1234 << 12)

Bits	31-12	11-7	6-0
Field	imm	rd	opcode

Table 3. U - type Instruction Format

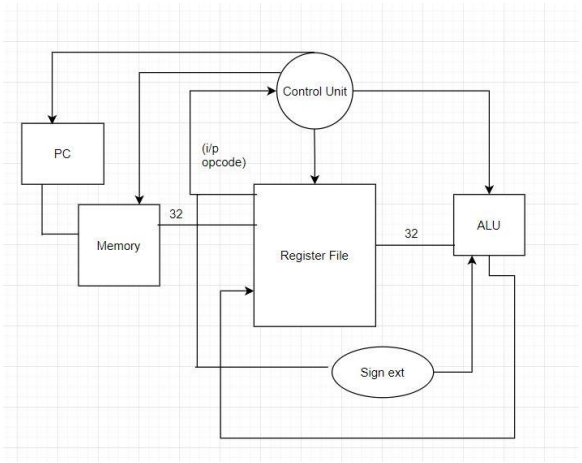

Fig. 5 U - type Instruction Implementation

The control unit shown in all the types of the instructions plays a major role in all the stages of the pipeline. Along with the clock every cycle has its own control bit signals which facilitates the operations achieved by the CPU.

The control unit will take an opcode, func3 and func7 as the input values. From this it is coded to give a certain value to all the control signals being fetched to the specific unit of the CPU like register file, memory, and ALU. Along with that other electronic components like multiplexer, non-architected registers, and logic gates.

b. BASYS 3 FPGA

Basys 3 is a field programmable gate array with Xilinx Artix-7 architecture with Xilinx Vivado support. The main reason for selecting this board is Artix-7 technology which supports implementation of RISC-V architectures. Si-Five announced three platforms for RISC-V ISA out of which Artix-7 was one. Another reason for choosing the board is that in spite entry level it supports most of the peripherals and is in expensive.

c. VHDL model

The VHDL model for the processor is shown in the figure 6. The components created were with architecture and entity description are instruction memory, main memory and register. In this we have two different data buses for fetching the data from memory and for writing the data into memory.

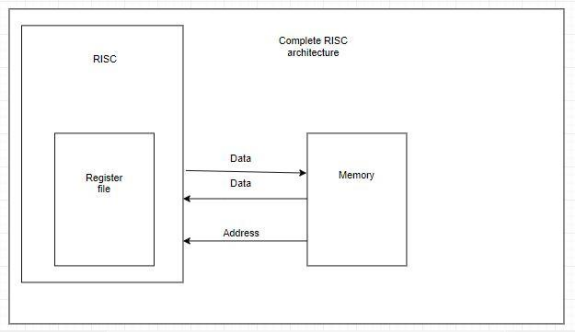

Fig 6 VHDL model of Processor

d. Xilinx Vivado Suite

For the synthesis, simulation and analysis of hardware description language Xilinx provides software known as Xilinx Vivado suite. Xilinx Vivado is based on Tool Command Language. The technology supported by Xilinx Vivado suite are Ultrascale, Virtex-7, Kintex-7, Artix-7, and Zynq-7000. The technology support and functionality of the suite was the reason behind choosing it.

VII. Result and Discussion

From the data-path developed for the 5-stage multi-cycle processor, the implementation is done in VHDL language using Vivado 2016.2 web pack edition. In this total three files have been built which incorporates register file, memory, and CPU. Along with that a source to implements the correlation between them is also designed in terms of Complete RISC file as shown in the figure 6.

The register file contains the list of registers and its external ports to communicate with the brain of the CPU along with the clock – frequency at which processor runs.

The memory file includes the instruction memory and data memory. Once received address it fetches 32-bit data or writes it at the pointed location on clock via two data buses shown in the figure.

In the processor we have all the hardwired connections to implement the logic as per the data path and instruction architecture. In this, register file is placed as its copy. Processor logic depends basically on two main processes. One is clock and other is its state. At every clock control signal, memory and register file given a task to implement. Because of the multi cycle development, in each cycle every unit will be doing certain job and never stays idle. This is true in a nutshell. However, external threats like dependencies in code or some requiring smaller than a 5-satge as explained in section VI still exist. Another process vital to the design is depends on state. Every state, fetch, decode, execute, memory, and writeback does a job which will include the logic for all the instruction implemented. Depending upon the opcode, instruction type and control signal it is coded to reflect a output for the next state.

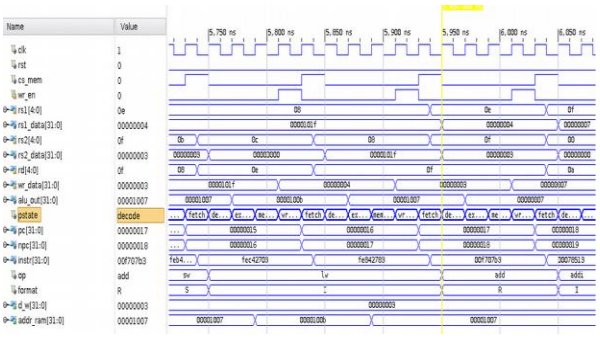

Fig 7 Result of simulation

The last but the most important source file contains copies of all of these to reflect the correlation between them, actually aligning their execution for the final implementation.

VIII. Timeline

- 4th week form the commencement of semester we started literature review of RISC processor vs other processors.
- At the end of 5th week we decided to implement RISC-V on FPGA. Also in the same week we came to conclusion we will use VHDL along with Xilinx Vivado.
- At the start of 6th week we selected BASYS 3 board for implementation .
- At the end of 6th week availability and compatibility of BASYS 3 was checked.
- On the 8th week we submitted proposal of this project for approval.
- After it took us 3 weeks to study RISC-V ISA thoroughly along with its latest development. We took the decision to implement RV32I instruction set.
- Further it took us 2 weeks to create and implement single-cycle and multicycle data path and control unit for RISC-V by using the concepts of MIPS architecture.
- For the next 2 weeks we developed VHDL code simulated it and tackled with the errors.
- Last week was occupied with documentation.

IX. Individual contribution.

After getting the approval, we divided the task in terms of hardware and softwires.

- Kushan Parikh :

  1] Review at processor implementation in VHDL.
  2]Studied the requirement required to run BASYS 3 and its compatibility with Vivado and additional drivers needed.
  3] Created VHDL code for the data path and rectified errors.

- Aditya Tumsare:

  1]Studied RISC-V architecture thoroughly.

  2]Comparing it with the MIPS developed data path and control unit for RISC-V ISA.

  3]Enhanced the data path for other instruction format.

X. Future work

The multicycle implementation of the data path can be improved and be extended to implement parallel pipelining. Thus, can be extended for superscalar processor. Also, the data path can be enhanced for implementation of advance instruction format. The data path can be shaped to work on different FPGA boards and with different peripherals to let the CPU communicate with external world.

XI. Conclusion

RISC-V RV32I is very similar to the old reduced instruction set architectures. Yet it is important for the study of implementation of micro-processors. RV32I is simple, modular, open ISA and adequate to cover basic instruction. Also, elementary in building RISC based architectures. The report explains implementation of multicycle data path for RV32I for different types of instructions. Also, synthesis is also done on XILINX Basys 3 using Vivado design suite.

References

1] David Patterson & Andrew Waterman, “The RISC-V: An Open Architecture Atlas”, September 10, 2017 beta version 0.01

2] Andrew Waterman, “Design of the RISC-V Instruction Set Architecture”, January 3, 2016

3] Computer Organization and Design RISC-V Edition: The Hardware Software Interface by David Patterson & John Hennessy

4] **“RISC-V Offers Simple, Modular ISA”**, D. Kanter, *The Linley Group MICROPROCESSOR report*, March 2016

5] E. Matthews and L. Shannon, "TAIGA: A new RISC-V soft-processor framework enabling high performance CPU architectural features," *2017 27th International Conference on Field*

Programmable Logic and Applications (FPL), Ghent, 2017, pp. 1-4

6] M. Gschwind, V. Salapura and D. Maurer, "FPGA prototyping of a RISC processor core for embedded applications," in *IEEE Transactions on Very Large-Scale Integration (VLSI) Systems*, vol. 9, no. 2, pp. 241-250, April 2001.

7] Wirth N. The Design of a RISC Architecture and its Implementation with an FPGA

8] Charles H. Roth, Jr and Lizy Kurian John, “Digital Systems Design Using VHDL”, by Thomson, second edition

9] Andrew Waterman, Yunsup Lee, Riscv-isa-simulator, 2011, https://riscv.org/software-tools/risc-v-isa-simulator/

10] Tim Newsome, RISC-V sw dev, google group forum, 2016, https://groups.google.com/a/groups.riscv.org/forum/#!topic/sw-dev/ebYQGbU8oUo