NYU-6463-RV32I Processor Design Project

by
Aswin Raj K(N121801008, ar7997)
Anish (N12499733, am12553)
Devashish (N19620681, dg4015)

DEPARTMENT OF ELECTRICAL ENGINEERING

Note: Zoom in to see the plots clearly

Contents

1 Complete Design Datapath

The figure shown below represents the datapath of the designed RISC V R32I processor. The major components are Program Counter (PC), Control Unit (CU), Instruction Memory (IM), Data Memory (DM), ALU and Register File (Reg File).

2 Individual Components

2.1 Program Counter

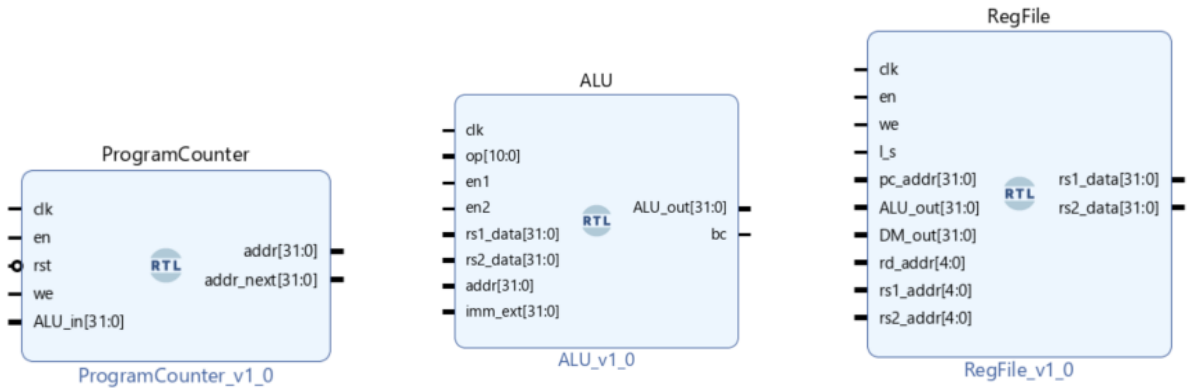
This is a 32-bit register that contains the address of the next instruction to be executed by the processor. Upon reset this should equal the start address of instruction memory (0x01000000). The program counter fetches the next instruction from IM by adding 4 to its current address on clock edge, since we are using byte addressing for this processor model. So, whenever the write enable signal is assigned for the program counter, we can decide which operation can be performed for the Program Counter using the enable signals. When enable is assigned low, the fetch next 32-bit address is performed onto the PC, or when the enable is high, the output data from the ALU is used by the program counter, to update its current address to the desired address provided by the ALU output accordingly.

2.2 ALU

This block performs operations such as addition, subtraction, comparison, rotation, etc. It uses the control signals generated by the Decode Unit, as well as the data from the registers or from the instruction directly. It computes data that can be written into one of the registers (including PC). This block performs operations such as load, store and also executes immediate instructions. All the input and output signals for this block are 32-bit wide. The ALU consists of 2 enable signals, named "en1" and "en2". When enable1 is high, the ALU takes "rs1_data" (output from RegFile) as its first input, else it takes the address delivered by the PC as its first input. Similarly, when enable2 is high, ALU considers immediate extended "imm_ext" data as its second input, else if enable2 is low, it considers "rs2_data" as its second input. We consider a 11-bit control signal for this block named "op" which is assigned by the Control Unit. Based on this control signal "op", the ALU recognizes which operation is to be performed. For branch instructions, the ALU raises a branched condition flag "bc", which is utilized by the CU to direct this flag to the appropriate block.

2.3 Register File

This block contains 32 32-bit registers. The register file supports two independent register reads and one register write in one clock cycle. The RegFile is used for storing the data temporarily. This block has 2 multiplexers at its input. The first multiplexer has a select signal named "l_s", when it is high, the output data from the ALU is feeded into the RegFile at its appropriate address, else when its low, the output data from the DataMemory is feeded into the RegFile at its appropriate address. The next multiplexer chooses between one of the outputs from the first multiplexer and the address provided by the PC. Based on the write enable signal, if it is high, the output from the next multiplexer is feeded into the RegFile through "rd_data_in" at "rd_addr".



(a) Program Counter (b) ALU (c) Reg File

Figure 1: Block Diagram

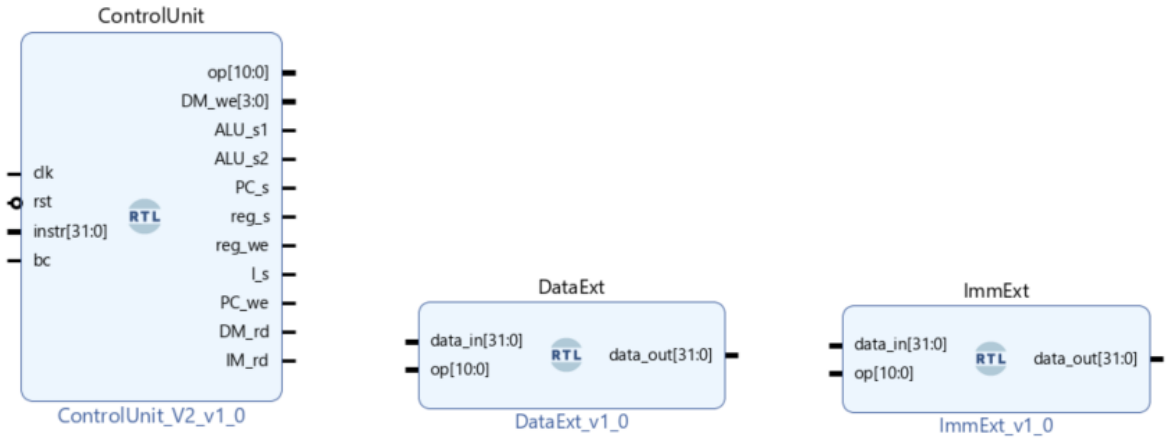
2.4 Control Unit

This block takes as input some or all of the 32 bits of the instruction, and computes the proper control signals that are required for correctly coordinating the other blocks in your design. These signals are generated based on the type and the content of the instruction being executed. This block contains a FSM. It outputs the control signals like "op", write and read enable signals for the memory and regfile, select lines for all the muxes. Based on the 32-bit instruction and the flag "bc", the CU modifies the output control signal suitable for the desired block which is to be executed.

The control unit issues the control signals based on which other modules act. The control unit houses the FSM. Each of the state in the FSM stays for 2 clock cycles. In the first clock cycle the control unit update its control signals and in the next clock cycle the modules performs their task.

2.5 Data Extension

This block extends the data with the most significant bit of the input data, in order to fill the 32-bit array, which is used for arithmetic operations in the ALU (since all inputs are 32-bit wide) or temporarily storing the data in the RegFile. In some cases, we extend the data with zeros depending upon the instruction opcode (control signal), which is then utilized by the ALU for that specific opcode operation. Data Extension is useful for dynamic purposes such as pulling the information or storing it externally, with retaining the data as it is.The data extension is done based on the 3-bit opcode of the instruction from the IM. The output for this block is thus 32-bit wide data.



(a) Control Unit (b) Data Ext. (c) Imm. Extend

Figure 2: Block Diagram

2.6 Immediate Extension

This block extends the data with the most significant bit of the input data, in order to fill the 32-bit array, which is used for arithmetic operations in the ALU (since all inputs are 32-bit wide). In some cases, we extend the data with zeros depending upon the instruction opcode (control signal), which is then utilized by the ALU for that specific opcode operation. Immediate Extension is useful for dynamic purposes such as performing signed and unsigned operations or storing it externally, with retaining the original data as it is.The immediate extension is done based on the 10-bit opcode of the instruction from the IM. The output for this block is thus 32-bit wide data.

2.7 Instruction Decoder

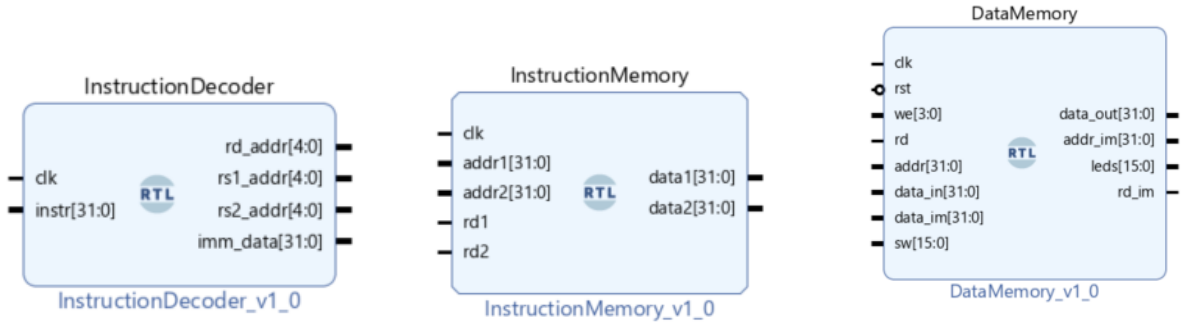
The instruction decoder logic converts the 32-bit instruction into settings for all the internal control lines. The Instruction Decoder reads the instruction from memory, and sets the component pieces of that instruction to the necessary destination blocks. Thus Instruction Decoder outputs the 5-bit address for “rs1_addr”, ”rs2_addr”, ”rd_addr” of RegFile and also outputs the 32-bit immediate data destined for ALU operations. Based on the lower 7 bits (opcode) of the instruction, it segregates the instruction bits and assigns each bit or block of bits to the desired output signals.

2.8 Instruction Memory

The Instruction Memory is used to store the copy of instructions that the processor is currently executing or is about to execute. The IM is of at least 2 KB in size. This IM takes the read signal as its control signal. The 32-bit address “addr” is also the input for this instruction memory to keep fetching the next instruction using the address as its reference. Since we use byte addressing, each 32-bit instruction is stored in the form of 4-bytes, since the width of this IM is 8-bit. Thus current address plus four “addr+4” provides the next address of the next instruction whenever the read signal is asserted high. Thus, these 4-bytes of instruction forms the output for IM for each instruction respectively.

2.9 Data Memory

The Data Memory is used to store the data that the processor is currently working on. Using the control signals like read and write, we can read the data from the DM as well as write the data into the DM. Data memory is an array of 8-bit elements of total size 4KB. Two independent data read and one write operations can be performed in a single clock cycle. The starting addresses of the memory contains the N-numbers of the members, followed by LEDs configuration registers and then switch-input registers.



(a) Instruction Decoder (b) Instruction Memory (c) Data Memory

Figure 3: Block Diagram

3 Finite State Machine

For a processor design, an FSM can be used to model the control logic that coordinates the operation of the various components of the processor, such as the instruction fetch unit, the decoder, and the execution unit. The FSM defines the sequence of steps that the processor follows to fetch, decode, and execute instructions, as well as the conditions under which these steps are performed.

States : The FSM has a total of 8 states that represent the different phases of the processor's operation, such as fetching an instruction, decoding an instruction, executing an instruction, or accessing memory. The FSM diagram is as shown in figure 4.

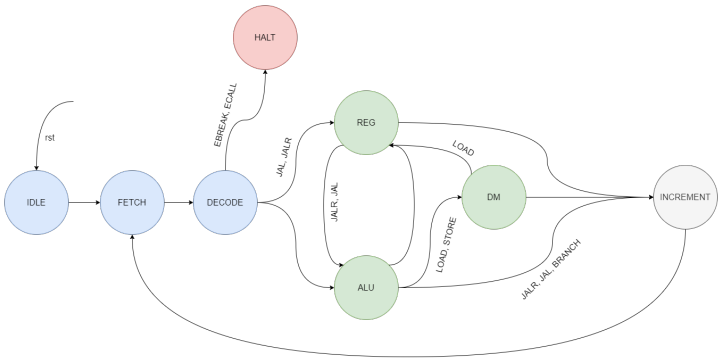


Figure 4: Implemented FSM : The empty path in the FSM shows that it will go to other state for all other instructions.

Inputs : The FSM receives inputs from various sources, such as the instruction memory, the register file, that affect its behavior and the transitions between states.

Outputs : The FSM generates outputs that control the operation of the processor components, such as the address of the next instruction to fetch, the control signals for the decoder, or the data to be written to the register file.

Transitions : The FSM transitions between states based on the inputs and the current state. For example, the FSM may transition from the fetch state to the decode state when an instruction is fetched, or from the execute state to the fetch state when an instruction is completed.

Next state logic : The FSM includes logic that determines the next state based on the current state and the inputs. This logic can be implemented using combinational logic circuits, such as multiplexers or decoders, or using a programmable logic element, such as a lookup table or a state machine.

4 The Whole Processor

Figure 5 shown below represents the datapath of the designed RISC V R32I processor. The major components are Program Counter (PC), Control Unit (CU), Instruction Memory (IM), Data Memory (DM), ALU and Register File (Reg File).

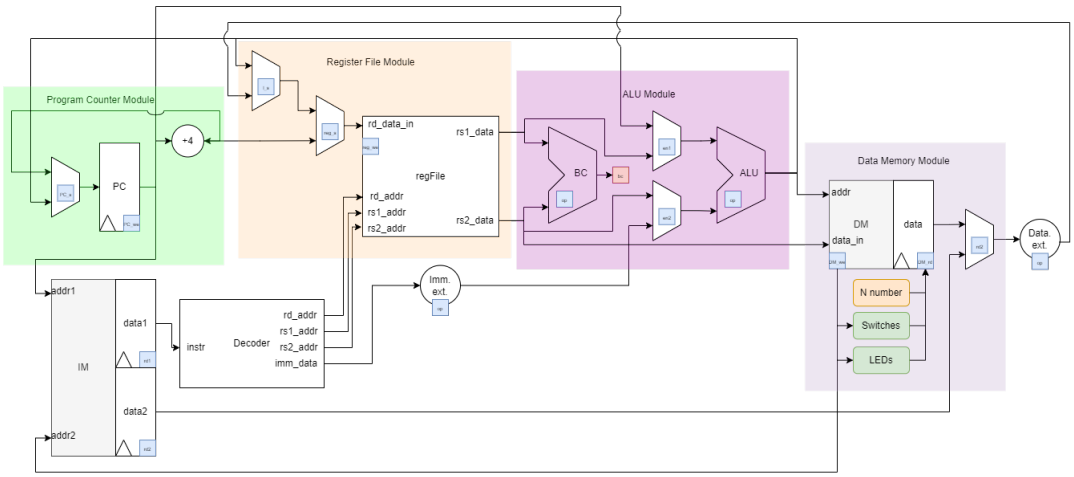


Figure 5: Processor Data Path, The blue squares refers to the control signals issued by the control unit.

5 Design Analysis

Performance and area analysis are important considerations in the design of a processor, as they can impact the overall performance. There are several metrics that can be used to measure the performance and area of a processor, and these metrics can be analyzed and compared to evaluate the design process.

5.1 Performance Analysis

Performance analysis is the process of evaluating the speed and efficiency of a processor in executing instructions and completing tasks. Some common metrics used in performance analysis are:

Clock frequency : The clock frequency is the rate at which the processor's internal clock signal oscillates, and it determines the maximum speed at which the processor can execute instructions. The maximum clock frequency for designed NYU-6463-RV32I is close to 100MHz

Instructions per second (IPS) : The IPS is a measure of the number of instructions that the processor can execute per second. A processor with a higher IPS can complete tasks faster, but it may also consume more power and generate more heat.

CPI : The CPI is a measure of the average number of cycles that the processor requires to execute an instruction. A lower CPI indicates that the processor is more efficient in executing instructions. The CPI for NYU-6463-RV32I is 2.

5.2 Area Analysis

Area analysis is the process of evaluating the size and complexity of a processor in terms of the physical area that it occupies on a chip or printed circuit board. Some common metrics used in area analysis are:

Gate count : The gate count is the total number of logic gates in the processor design, and it is a measure of the complexity and size of the processor.

Power consumption : The power consumption is the amount of power that the processor consumes when it is operating, and it is a function of the clock frequency, the IPS, and the complexity of the processor.

5.3 Timing Analysis

The design passed all timing analysis. The timing analysis summary is as shown in figure 6.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.170 ns	Worst Hold Slack (WHS):	0.096 ns	Worst Pulse Width Slack (WPWS):	3.750 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	5626	Total Number of Endpoints:	5626	Total Number of Endpoints:	733

Figure 6: Timing Analysis Summary for 100MHz clk

5.4 Resource Utilization

The resourse utilization is as shown in figure 7. It is seen that only one block RAM is used. Even though long arrays of 31-bits are used in the design only one of them was inferred as BRAM (i.e regFile) by the synthesis tool. One reason for the synthesis tool not to infer BRAM is multiple read and writes per process. 34 Bonded pins are used which includes 16 pins for leds, 16 pins for switches, 1 pin for rst and finally the clock.

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Block RAM Tile (50)	Bonded IOB (106)	BUFGCTRL (32)
N Processor_Wrapper	1956	212	256	128	552	1444	512	2	34	1
processor (Processor_Top)	1956	212	256	128	552	1444	512	2	0	0
S1 (ALU)	174	33	0	0	140	174	0	0	0	0
S2 (ControlUnit_V2)	416	26	0	0	173	416	0	0	0	0
S3 (DataMemory)	560	80	256	128	194	48	512	0	0	0
S6 (InstructionDecoder)	56	35	0	0	45	56	0	0	0	0
S7 (InstructionMemory)	73	3	0	0	29	73	0	1	0	0
S8 (ProgramCounter)	40	32	0	0	28	40	0	0	0	0
S9 (RegFile)	675	3	0	0	254	675	0	1	0	0

Figure 7: Resource Utilization

6 Testing

6.1 Unit Testing

This program tests all the 40 instructions supported by the RV32I processor in sequence and the testbench checks for the expected outputs of each instruction. Initially data is initialized into the Data Memory and the each instruction is executed separately. Hence, it is flexible enough that the input data can be changed by the user and check for all possible corner cases for every instruction.

As this test includes all the instructions supported by the processor, the successful conclusion of the testbench can be taken as an indicator that the design is accurate and works as expected. Overall, the simulation and testing results obtained were very promising, and they confirmed that the designed processor met the performance and power targets set for the project.

However, we also identified several areas for improvement, such as the need for additional optimization of the control logic and the execution unit, and the possibility of adding support for additional instructions or features.

6.2 High Level Testing

6.2.1 Input Output Testing

Objective

The objective of this simulation is to test the LEDs and switches connected to the processor. A small assembly program is written to control the LEDs with switches.

Results

As you can see in the video the LEDs are controlled with the help of switches.

6.2.2 Generating Fibonacci Series

Objective

The objective of this simulation is to test the performance and correctness of the RISC-V 32I processor in executing a program that generates the Fibonacci series.

Methodology

The test was carried out using a cycle-accurate simulator, with the processor design described in Verilog. The program used to generate the Fibonacci series was compiled using a RISC-V compiler and loaded into the simulated processor's main memory. The simulator was then run for a sufficient number of cycles to allow the program to complete its execution.

Results

The simulation results show that the processor was able to execute the Fibonacci series program correctly.

Conclusion

The simulation results demonstrate the correctness and performance of the RISC-V 32I processor in executing a program that generates the Fibonacci series. The high performance and low CPI achieved in this test suggest that the processor is well-suited for applications that require fast and efficient mathematical computations.

```
==================RESULT====================
         1) reg[3] =          1, exp_reg[3] =          1
         2) reg[3] =          2, exp_reg[3] =          2
         3) reg[3] =          3, exp_reg[3] =          3
         4) reg[3] =          5, exp_reg[3] =          5
         5) reg[3] =          8, exp_reg[3] =          8
         6) reg[3] =         13, exp_reg[3] =         13
         7) reg[3] =         21, exp_reg[3] =         21
         8) reg[3] =         34, exp_reg[3] =         34
         9) reg[3] =         55, exp_reg[3] =         55
        10) reg[3] =         89, exp_reg[3] =         89
```

Figure 8: Fibonacci Testbench output

6.2.3 RC5 Encryption

Objective

The objective of this simulation is to test the performance and correctness of the RISC-V 32I processor in executing a program that implements the RC5 encryption algorithm.

Methodology

The RC5 encryption program was compiled using a RISC-V compiler and loaded into the processor's Instruction Memory. The SKEY array is predefined and loaded into the Data Memory. The simulation was then run for a sufficient number of cycles to allow the program to complete its execution.

The input data for the encryption was a block of random bytes, and the output data was compared to the expected result obtained using a reference implementation of the RC5 algorithm.

Results

The simulation results are summarized in the table 1.

Metric	Value
Clock Frequency	100MHz
Instructions Executed	112
Execution time	12.45μs
CPI (cycles per instruction)	11.12

Table 1: Analysis for RC5 Encryption

The simulation results show that the processor was able to execute the RC5 encryption program correctly and achieve a high performance, with an execution time of 500 microseconds and a CPI of 2.00. This indicates that the processor was able to execute the program efficiently, with minimal pipeline stalls or other bottlenecks.

```
==================RESULT====================
Code to encrypt is d73ca68e21fee44f
Encrypted Code is 91a72d5ae1d1240a
============================================
```

Figure 9: RC5 Testbench output

7 Support For C

To facilitate the development of C programs on the processor, it is important to integrate the GCC compiler and the runtime libraries into a complete toolchain that includes all the necessary tools and libraries. We made use of the resources provided in the class; installed the GCC compiler and the runtime libraries, and set up the necessary build and run scripts or Makefiles.

The GCC toolchain is set up, it is important to test and validate it by building and running a variety of C programs on the processor. This can help to identify any issues or problems with the toolchain, and to ensure that it is working correctly and efficiently on the processor.

We were able to access the data from the Instruction memory using the load commands. We couldn't finish the C part because of the time constraint.

8 Future Scope

The NYU-6463-RV32I currently is a 32-bit processor and supports a subset of complete RISC-V instructions. Each component of the processor has been tested enough to ensure proper operation for every test case. The overall processor has been tested with few high level tests that check for all the instructions. There are several directions in which the designed processor could be extended and improved in the future. Some possible areas for further work are:

- **Optimization :** The processor design could be further optimized to improve its performance and reduce its power consumption. This could involve techniques such as pipeline optimization, register renaming, and out-of-order execution. As of now, every instruction takes two clock cycles to get executed completely. For a major portion of the instructions, it can be optimized to be executed in one clock cycle only. This step could improve the overall performance by 40% (approx.).
- **Instruction set extension :** The processor could be extended to support additional instructions or features from the RISC-V ISA, such as floating-point operations, or vector instructions.
- textbfMulticore support : The processor could be extended to support multicore architectures, with the addition of inter-core communication mechanisms such as message passing or shared memory.
- **Hardware acceleration :** Special-purpose hardware acceleration units could be added to the processor to accelerate the execution of specific types of instructions or applications. For example, the processor could include a dedicated unit for cryptographic operations or machine learning algorithms.
- **Embedded systems :** The processor could be used as the central processing unit (CPU) in an embedded system, such as a sensor, a control system, or a wearable device. This would require the design of additional hardware and software components, as well as the integration of the processor with the rest of the system.

Overall, the future scope and extension of the project will depend on the specific goals and needs of the application or system in which the processor is used. However, the flexibility and modularity of the RISC-V architecture make it well-suited for a wide range of applications and scenarios, and there are many opportunities for further research and development in this field.

9 Conclusion

The RISC-V processor project aimed to design and implement a 32-bit processor that is compliant with the RISC-V instruction set architecture (ISA). The processor was designed using a cycle-accurate model, and it was implemented and tested using a simulation environment.

The project achieved its main goals, which were to design and implement a functional RISC-V processor that is able to execute a wide range of instructions and programs, and to evaluate the performance and correctness of the processor using a variety of tests. The processor achieved a high performance and low CPI (cycles per instruction), indicating that it is well-suited for a wide range of applications that require fast and efficient processing. It has also been implemented on the Basys-3 FPGA, to validate its behavior in real hardware.

The project also encountered a number of challenges and obstacles, such as debugging and optimizing the processor design, and integrating the Data Memory into the processor. Some of these challenges were addressed seeking help and guidance from the Teaching Assistants and most of them were addressed exploring the internet.

Based on the results of the project, it is recommended to further test and validate the processor with a wider range of programs and benchmarks, and to optimize the processor design for specific applications

or targets. Finally, it is recommended to explore the potential of the RISC-V ISA for other domains or platforms, and to continue developing and improving the RISC-V ecosystem to support a wide range of applications and use cases.

10 Resources

The link to the final project files and videos.
