



Istanbul Medipol University
School of Engineering and Natural Sciences
Graduation Project

2022-2023

PROJECT TITLE
RISC-V Processor
PROJECT ADVISOR
Prof. Selim Akyokuş Prof. Mustafa Aktan
TEAM MEMBERS
Amer Alhamvi İbrahim Yuşa Çetin



Istanbul Medipol University
School of Engineering and Natural Sciences
Graduation Project

BUDGET (TL)	PROPOSED	CONSENTED
IMU FUNDING	8500	8500
SPONSOR COMPANY FUNDING	0	0
TOTAL	8500	8500

PROJECT PLAN	PROPOSED	CONSENTED
PROJECT PLAN Duration in Weeks	28 Weeks	28 Weeks
STARTING DATE	01.11.2022	01.11.2022



Project Code: RISC_V_Processor

PROJECT ADVISOR	PROJECT CO-ADVISOR
Name: Prof. Selim Akyokuş	Name: Prof. Mustafa Aktan
Contact Information: Tel : 216-681-5141 E-mail : sakyokus@medipol.edu.tr	Contact Information: Tel : E-mail : mustafa.aktan@medipol.edu.tr
Signature:	Signature:



Istanbul Medipol University
School of Engineering and Natural Sciences
Graduation Project

DEPARTMENT CHAIR
Name: Prof. Mehmet Kemal Özdemir
Contact Information: Tel : 216-681-5626 E-mail : mkozdemir@medipol.edu.tr
Signature:

TEAM MEMBER	TEAM MEMBER
Name: Amer Alhamvi	Name: İbrahim Yuşa Çetin
Contact Information: Tel : +905312372775 E-mail : amer.alhamvi@std.medipol.edu.tr	Contact Information: Tel : +905458607028 E-mail : ibrahim.cetin@std.medipol.edu.tr
Signature: 	Signature: 



Istanbul Medipol University
School of Engineering and Natural Sciences
Graduation Project

Project Title: RISC-V Processor

Project Advisors: Prof. Selim Akyokuş, Prof. Mustafa Aktan

Team Members: Amer Alhamvi, İbrahim Yuşa Çetin

Project Group Title: MediRISC

PROJECT OVERVIEW/SUMMARY/ABSTRACT

The objective of this project is to develop a System-on-a-Chip (SoC) that houses a 32-bit RISC-V CPU and supports the RISC-V standard extensions RV32IMC and an extra 11 additional custom non-standard instructions for specialized use cases in cryptography and neural networking. The project comes at a time of pushing towards open-source hardware and embodies the idea by building an SoC using an open-source CPU core that runs an open-source ISA. The SoC also houses main memory and L1 cache elements as well as a VGA module and basic SPI, PWM, and UART transmission peripherals.

Keywords: RISC-V, CPU, SoC, chip, FPGA, Verilog



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

1. OBJECTIVE OF THE PROJECT:

The aim of this project is to build an SoC (System on Chip) with a 32-bit extended RISC-V core with custom instructions for specialized use cases for cryptography and neural networking purposes on an FPGA.

The SoC supports RISC-V RV32IMCX instructions and has two caches for fast memory access for data and instructions. Also, an AI accelerator module is included for use for fast 2D convolution calculations.

This SoC is built with competing in the Teknofest chip design competition in mind. Furthermore, the project reached the finals of this year's competition and can be further developed within the next 1 to 2 years with the aim of winning the competition.

Finally, to show off the work done on this project, a demonstration will be done utilizing seven segment displays and the VGA port on the DE10 Lite board.

2. LITERATURE REVIEW:

- RISC-V

RISC-V is an open ISA built around a simple integer core instruction set with support for 32-bit, 64-bit, and 128-bit computing. While it also supports for series of extension sets to support additional functionalities such as multiplication and division, floating-point operations with different precision levels and more instructions (2016).

The widespread adoption of an open ISA like RISC-V would encourage a more cooperative environment for hardware development by the means of open-source cores, which would push down the cost of reuse of existing designs inducing a new market where small companies can develop their own hardware (2014).

- Cache

The idea of small, fast memory level between the main memory and the CPU can be found as back as in (1965) describing a 1-level cache system to exploit the temporal locality of memory access patterns. Modern cache systems build upon the same core principles as in (1965) and share the core elements such as the validity bit, storing only address tag instead of the full memory address, and the idea of write-back cache but in a more advanced manner (Patterson and Hennessy, 2013).

Modern cache systems utilize both spatial and temporal locality of the data in the memory to achieve highest possible hit ratios. Also, in literature can be found the investigation of cache size, cache word size, different replacement algorithms, write-through and write-back, direct, associative, and set-associative caches, multi-level caching systems, separating data cache from instruction cache, cache pipelining, and more(1982)(Stallings, 2015).

3. ORIGINALITY:

RISC-V is an instruction set architecture (ISA) planned with forward thinking in mind, by supporting 32, 64, and 128-bit computing, and has few complete implementations



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

(2014). The rationale behind RISC-V can be considered under two categories: the philosophies of open computing and reduced instruction sets. RISC-V is an open-source instruction set meaning that it can be implemented and extended by anybody without any legal drawbacks. It is also a RISC instruction set meaning that it has a significantly smaller number of instructions compared to CISC ISAs such as x86. Upon the completion of the project, it will be among the few implementations of a RISC-V SoC that are available on the market today.

4. SCOPE OF THE PROJECT AND EXPERIMENTS/METHODS:

The project work packages are the following:

- Development of a primitive 8-bit CPU (for learning purposes only).
- Running the Ibex core, in simulation and on real hardware.
- Testing the Ibex core using simple bare-metal RISC-V programs.
- Implementation of 11 non-standard instructions to the Ibex core.
- Implementation of memory routing unit.
- Building an L1 cache for the main memory.
- Constructing a main memory for programs and data.
- Building peripheral modules.
- Creating some demonstration software using the seven segment display and VGA ports on the DE10-Lite board.

4.1) 8-bit CPU

The first step in our project was to develop a simple, non-standard, 8-bit CPU and use it to gain elementary experience with Intel's FPGA development environment and the process of developing and working with a digital CPU. This CPU was written with Verilog HDL from scratch and loosely followed the architecture of the CPU shown in Chapter 13 of (LaMeres, 2017).

The instructions of this CPU are made of an opcode and an optional operand. Each memory address contains either an opcode or an operand, where the operands correspond to the opcode in the previous memory address. The CPU first decodes the opcode at the PC and increments the PC. If the instruction does not require an operand and is not a branch instruction, the PC points to the next opcode. Otherwise, if the instruction requires an operand the PC points to the operand.

This was a short task and was completed in two weeks.

4.2) Running The Ibex Core on the FPGA

The Ibex Core (lowRISC, 2018) is an open-source configurable RISC-V CPU core written in SystemVerilog. The Ibex Core planned to be used has a two-stage pipeline and supports the RISC-V configuration of RV32IMC (Figure 4-1). This means that along with the standard 32-bit instructions, the CPU Core will support multiplication instructions and the 16bit long compressed instruction sets.

This part includes the conversion of the Ibex Core’s RTL code from SystemVerilog to Verilog using the sv2v library (Snow, 2019). This operation will ensure that the Quartus synthesis tools can read the RTL code as per the Ibex Core documentation.

Also for this part an elementary on-chip memory was programmed to test the main functionalities of the processor. For simulation testing, instructions to be tested were loaded to this memory using the “\$loadmemh” Verilog function. Since implementation on the real hardware does not support this directive, instructions were written as shown in figure in section 11.1 under a reset trigger in order to test instructions on real hardware.

At this stage, we tested simple programs by manually loading the binary machine code into the memory.

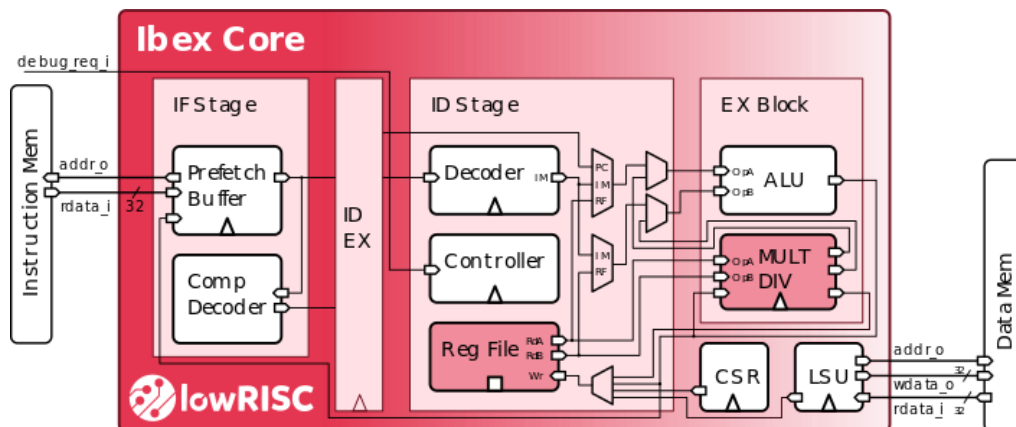


Figure 4-1: The Ibex Core pipeline

4.3) Testing the Standard Ibex Core

To test the hardware, basic RISC-V architecture programs were written in RISC-V assembly. The GNU RISC-V toolchain was used to compile this software. The programs were firstly run on the simulation using the Icarus Verilog software and the results are analyzed using the GTK Wave program. If simulation results were successful, the programs were tested on real hardware. Shell scripts were created to compile the source code and then extract the machine instructions from it using the objcopy command from the GNU RISC-V toolchain. The scripts output a file that contains the instruction code that can be run on a bare-metal RISC-V processor. Several small programs have been tested this way and yielded successful results.



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

The GTK Wave program is used to view simulation results. It allows us to see every input, output, and internal signal as they change with respect to time. To confirm the successful execution of instructions, contents of the register that was the destination of an instruction is checked. If it matches the theoretically correct result that the instruction was expected to produce, it means that the instruction could be executed successfully by the hardware.

To confirm successful execution on the DE10-Lite board, the contents of certain registers were tied up to the seven segment displays. Since registers store 32 bits words, and there are only 6 seven segment displays on the DE10-Lite board, only 24 bits of a single register can be displayed at once. Therefore, unlike in the simulation where any number of instructions can be tested in a single program, in the real hardware only the last instruction's output can be viewed. For the standard instructions that should be working out of the box with Ibex, testing each instruction individually would be unnecessarily time consuming, so two methods were chosen to test multiple instructions in a single program in this environment. In both of them, an arbitrary register's least significant 24 were tied to the seven segment displays. This register would be used as the destination register of the instruction that was being tested. The first method is to slow down clock speed significantly so that the output can be recorded by hand and compared to the simulation results. The second method is to use the output of each instruction as an input of the next instruction so that the only way for the destination register to have the correct value after execution is completed is if every instruction used in the program was executed correctly. Both methods were used throughout the development of the project for testing.

In later stages of the project, the testing method was improved in multiple ways to test other parts of the project. Loading the program to the BRAM based main memory was done by initialization files or through the In-System Memory Content Editor tool in Intel Quartus Prime. The first method allows the program to be uploaded to the FPGA along with the rest of the system while the later method allows for loading programs online and efficiently.

These methods require the code compilation process to generate an Intel hex format file, which is done with a python script incorporated within the shell script. The script also now compiles C code, which is used for preparing the demos.

4.4) Non-Standard Instructions

The RISC-V based CPU supports all base RISC-V instructions, and additionally 11 custom non-standard instructions. 6 of these instructions are related to the field of cryptography, and the other 5 are for performing 2D convolution in a quick, parallel manner. The details of these instructions are given in figures 4-2 and 4-3 below. These bit schemes were provided by the Teknofest Chip Design Competition Commission.



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

31	25 24	20 19	15 14	12 11	7 6	0	
	0000101	rs2	rs1	001	rd	0110011	hmdst
31	25 24	20 19	15 14	12 11	7 6	0	
	0000100	rs2	rs1	100	rd	0110011	pkg
31		20 19	15 14	12 11	7 6	0	
	011010111000	rs1	101	rd	0010011		rvrs
31	25 24	20 19	15 14	12 11	7 6	0	
	0010000	rs2	rs1	010	rd	0110011	sladd
31	25 24	20 19	15 14	12 11	7 6	0	
	0110000	00001	rs1	001	rd	0010011	cntz
31	25 24	20 19	15 14	12 11	7 6	0	
	0110000	00010	rs1	001	rd	0010011	cntp

Figure 4-2: Cryptography related custom instructions

31	30	29	25 24	20 19	15 14	12 11	7 6	0	
	en	00000	rs2	rs1	010	00000	0001011		conv.ld.w
31		25 24	20 19	15 14	12 11	7 6	0		
		0000000	00000	00000	011	00000	0001011		conv.clr.w
31	30	29	25 24	20 19	15 14	12 11	7 6	0	
	en	00000	rs2	rs1	000	00000	0001011		conv.ld.x
31		25 24	20 19	15 14	12 11	7 6	0		
		0000000	00000	00000	001	00000	0001011		conv.clr.x
31		25 24	20 19	15 14	12 11	7 6	0		
		0000000	00000	00000	100	rd	0001011		conv.run

Figure 4-3: Custom instructions for 2D convolution

The functions of these instructions are described in table 4-1 below. In all of these instructions, the result is saved to rd (destination register).

Instruction Name	Function
hmdst	Calculates the Hamming distance between rs1 and rs2.
pkg	Combines the least significant half of rs2 and rs1 such that the least significant half of rs2 is the most significant half of the result.
rvrs	Reverses the byte order of rs1.
sladd	Left shifts rs1 by one bit and adds it to rs2.
cntz	Counts the number of trailing zeros (i.e. the number of zeros until the first 1 is encountered starting from the least significant bit) at rs1.
cntp	Counts the number of 1's in rs1.
conv.ld.w	Loads rs1 and rs2 as the filter values for the convolution.
conv.clr.w	Clears any existing filter values for the convolution.
conv.ld.x	Loads rs1 and rs2 as the operand values for the convolution.
conv.clr.x	Clears any existing operand values for the convolution.
conv.run	Runs the convolution with the configured filter and operand values.

Table 4-1: Instruction functionality description table

4.4.1) Cryptography instructions



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

These instructions are executed inside the ALU. The reason it was decided to implement them inside the ALU was because the Ibex Core detects the 4 standard types of instructions and routes the necessary signals for their execution to the correct places and subsequently writes the results to the destination register specified in these instructions. All custom instructions were designed to comply with the standard RISC-V instruction types.

Among the custom cryptography instructions *hmdst*, *pkg*, and *sladd* fall into the *R-Type* instruction category, whereas *rvrs*, *cntz*, and *cntp* fall into the *I-Type* category. Since these instructions are not in the standard ISA, the Ibex Core treats them as illegal by default. It uses the *opcode* (bits 31:27), the immediate value (bits 26:20), and the *funct3* parameter (bits 14:12) to identify these instructions. Since the cryptography instructions have the same *opcode* as standard ALU operations, it wasn't necessary to make any changes to *opcode* based identification. However, *R* and *I-Type* instructions were being marked illegal based on their immediate and *funct3* parameters.

There were case statements inside the Ibex decoder to check the *funct3* parameter of *I-Type* instructions and there were already cases for 0b101 which represents *rvrs* and 0b001 which represents *cntz* and *cntp*. For the group of hypothetical instructions whose *funct3* value is 0b001 -which includes the *cntz* and *cntp*-, the marking-as-illegal operation was being performed based on the constant immediate values at bits 26:20. The constant immediate values at this range for *cntz* and *cntp* are 0b0000001 and 0b0000010 respectively, so new cases were added to the case statement checking the immediate values for these two instructions and in order to make sure that the CPU treated them as legal instructions, the *illegal_insn* signal was set to zero.

The other *I-Type* instruction which is *rvrs* has a different *funct3* value than *cntz* and *cntp* so its legalization was done under a different section of the code. Here, the Ibex Core first checked the section of the immediate value at bits 31:27 and then checked some other parameters under certain cases to determine the instruction's legality. The value of 0b01101 present in this parameter of *rvrs* fell under the latter group. To legalize this instruction, an if statement was added to the case for the 31:27 bits value for 0b01101 and the *illegal_insn* signal was set to zero if the remaining parts of the immediate parameter (bits 26:20) matched the value required for *rvrs*.

To legalize the *R-Type* instructions *hmdst*, *pkg*, and *sladd*, in the general case statement that segregated instructions based on their *opcodes* under the case for 0x33 which is the *opcode* for all three of these instructions under the additional case statement which checked for bits 31:25 and 14:12, the options 0b0000101001 (for *hmdst*), 0b0000100100 (for *pkg*), and 0b0010000010 (for *sladd*) were removed. *multdiv_operator_o* and *multdiv_signed_mode_o* signals were also determined under this case statement. Even if an instruction was not explicitly marked as illegal inside one of the cases, any instruction that was not mentioned anywhere got *illegal_insn* assigned to one under the default case. Instead of removing the default case, additional cases were added for *hmdst*, *pkg*, and *sladd*. Under these cases *multdiv_operator_o* and *multdiv_signed_mode_o* were assigned to zero as these instructions are not to be executed inside the multiplication-division unit. Under the last mentioned case statement, the *illegal_insn* assignment was being done based on certain conditions that were exactly the same for all the instructions that fell under the same upper-case, so the conditions were copied from them.



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

Lastly, the signal that was passed through the *alu_operator_o* port had to be given arbitrary but consistent codes for each of the custom instructions to be implemented inside the ALU because the ALU differentiated the operation to be performed based on this information. This assignment had to be done under a different case statement that checked the *opcode_alu* signal, for the case 0x33 for *R-Type* instructions and 0x13 for *I-Type* instructions. These arbitrary operator values were decided as shown in the table below.

Instruction	Operator Value
rvrs	0x7A
cntp	0x7B
cntz	0x7C
sladd	0x7D
pkg	0x7E
hmdst	0x7F

Tabel 4-2: Operator values to identify the cryptography instructions inside the ALU

After the cryptography instructions were legalized and assigned operator values, the Ibex core began sending the requested values from the requested registers as specified in these instructions to the ALU. Likewise, it saves the results after a single cycle (standard for all instructions that execute inside the ALU) to the corresponding destination register.

Buffer signals were created inside the ALU to hold the results of cryptography operations, and the result leaving the ALU was determined based on a multiplexing logic that used the operator value as a select signal.

To determine which operation is needed to be performed, the *operator_i* signal which carried the operator value was checked with a case statement under a new *always* block. Cases were created for all the values given in the operator values table above and the corresponding operations were performed there.

The mathematical expression used to calculate *hmdst* is given in the figure below.

$$\sum_{k=0}^{31} XOR(operand)[k]$$

The Verilog statement used to calculate *pkg* is given in the figure below.

```
pkg_result = {operand_b_i[15:0], operand_a_i[15:0]};
```

The Verilog statements used to calculate *sladd* are given in the figure below.

```
sladd_result = operand_a_i<<1;  
sladd_result = sladd_result + operand_b_i;
```

The Verilog statement used to calculate *rvrs* is given in the figure below.



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

```
rvrs_result = {operand_a_i[7:0], operand_a_i[15:8],  
operand_a_i[23:16], operand_a_i[31:24]};
```

The mathematical expression used to calculate *cntp* is given in the figure below.

$$\sum_{k=0}^{31} operand[k]$$

For the *cntz* instruction, an if statement was created that covered each of the 33 potential cases. The cases can be categorized into 3 broad scenarios. The first scenario is when the first bit of the operand is 1, in which case the result is zero. The second scenario is for when there is at least one 1 inside the operand, but it is not the first bit. For this case there are 31 else-if clauses to detect where the first 1 occurs. The third scenario is for when there are no 1's inside the operand and this scenario is handled in the else block at the end of this if statement. In this case the result is 32.

4.4.2) Custom AI Accelerator Instructions

Among the instructions to be developed for the AI Accelerator (henceforth to be called “accelerator”), 4 out of 5 of them are for loading values to the registers inside the accelerator unit or clearing the values present in those registers, whereas the remaining one is for triggering the execution of the convolution operation using these values. There are two restrictions of the accelerator: both the data and filter matrices must have the same dimensions, and they can only have up to 16 values. The accelerator was designed this way to adhere to the requirements for the Teknofest competition.

The four instructions for loading or clearing data can be executed inside a single cycle and all of them are *R-Type* instructions and they are *conv.ld.w*, *conv.clr.w*, *conv.ld.x*, and *conv.clr.x*. To enable the transmission of data from the registers called inside the instructions to the accelerator, firstly a case for the value 0x0B was added to the case statement inside the decoder that checked the *opcodes*. Another case statement was created under this case to check for the *funct3* values because that's the differentiating factor between these four instructions. Since the cryptography instructions were compatible with ALU instructions, all the necessary data transfers were being handled by the Ibex Core automatically, but this is not the case with the accelerator instructions because they are to be executed in their custom built dedicated modules. Because of this, all the signals that were necessary to execute these instructions had to be carried manually using custom ports. A total of 7 signals were used to achieve this. These signals and their descriptions are given in the table below.

Signal	Description
ai_wfilter	The values in the registers specified in the instructions are to be saved as filter values.
ai_wdata	The values in the registers specified in the instructions are to be saved as data values.



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

ai_operand_a_valid	The value obtained from the rs1 register is valid.
ai_operand_b_valid	The value obtained from the rs2 register is valid.
ai_cfilter	Clear all filter values.
ai_cdata	Clear all data values.
ai_run	Indicates that the convolution operation is currently being performed.

Table 4-3: Signals that were created to relay the information given in convolution instructions to the concerned places inside the CPU.

Since these are *R-Type* instructions, the Ibex Core reads the values at the registers specified in the instructions, and these can be read from the accelerator module. Whether these values are valid or not, and whether they are valid are determined using these custom signals. The route that these signals and operands travel from the decoder to the accelerator is illustrated in the figure below.

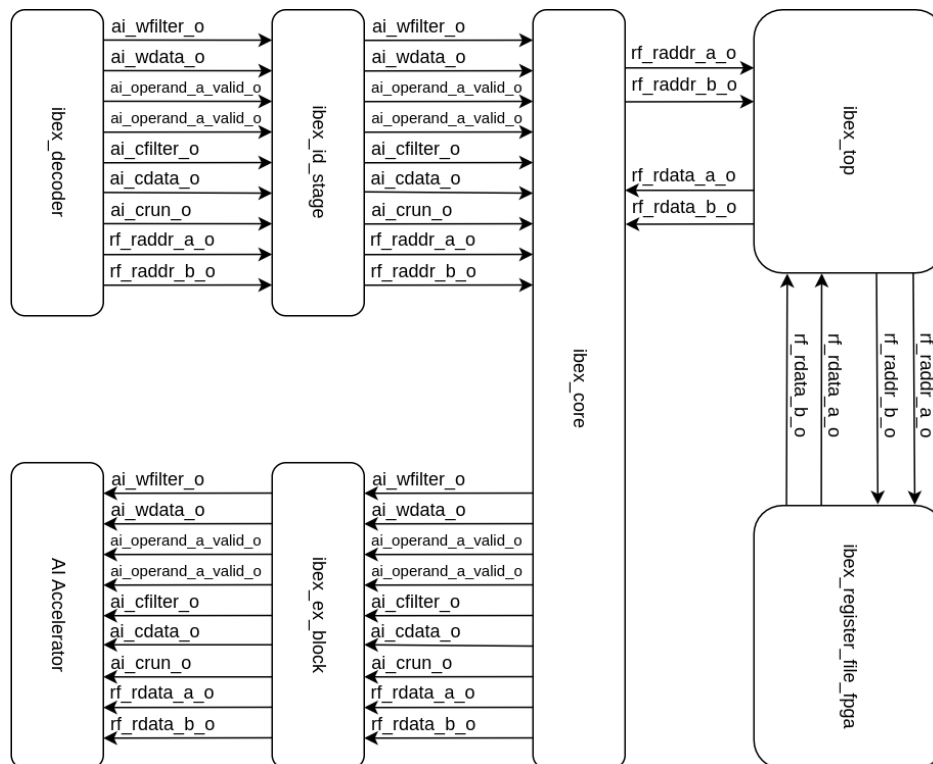


Figure 4-4: The route that signals that enable the communication between the AI Accelerator and the CPU traverse

Inside the accelerator, two separate 32-bit wide 16-item-long arrays were created to hold the data and filter values. Even though these arrays are word-wise single dimensional, they will be treated like 2D matrices that save their values linearly. As long as values are written to the data and filter arrays in the same order, it does not make a difference since all overlapping parts will be multiplied with their corresponding parts. Two 4-bit counters were



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

created to count how many write operations to data and filter arrays have been made since the last reset, taking into consideration that some write operations only write a single operand and others write two operands, indicated by their most significant bit. Upon a clear instruction, the counters are reset as well. If the number of values loaded to the data and filter registers are not the same, the extra values in the more lengthy array are ignored, as the empty spots in both of the arrays are assumed to be zero, these extra values will end up being multiplied with zeros.

When the *conv.run* instruction is executed, the *active* signal inside the accelerator is set to 1 which triggers the commencement of the operation. The first step is the multiplication of correspondent values in the data and filter arrays. In order to execute the convolution operation in the minimum number of cycles possible, all multiplications are done in parallel simultaneously. To achieve this, the Ibex multiplication-division unit (henceforth to be called “multdiv unit”) was instantiated 16 times (16 being the highest supported array size) using the *generate* Verilog directive. Two arrays were created to store the inputs to each of the multdiv units, each of their items were combinationally assigned to the corresponding index in the data and filter arrays if the *mult_active* signal was 1, and otherwise to zero. *mult_active* was set to 1 when the *run_i* input brought a value of 1 and got reset to 0 after 3 cycles which is the time it takes for the Ibex multdiv unit to complete one multiplication. The items of these arrays were inputted at each iteration of the multdiv generation loop at indices indicated by the iteration order of the loop. The result of each multiplication was also saved to a 16-item-long array of 32-bit items. This items of this array were given as inputs to a custom made AI Adder module (henceforth to be called “adder”).

The adder module uses a structure similar to a binary tree to perform the addition operations. The operations take 4 cycles in total. The reason all numbers were not added at the same cycle altogether was because that would most likely result in a timing violation on our FPGA. By pipelining the addition operations, in each cycle only pair-of-two’s of numbers are added together. In the fourth cycle of the operation, a single result remains which is the overall result of the convolution operation. This value is outputted to the accelerator which in turn outputs it to the execution block of Ibex. Here, there is a port called *result_ex_o* which contained the overall result of the execution block. Which result among the results produced by various execution block units to pick as the general result is determined by checking the *multdiv_sel* signal and the *ai_result_valid* and *ai_run* signals. The result can be one of the multdiv unit result, ALU result, and AI accelerator result. After the correct result is outputted from the execution block, Ibex normally proceeds to take care of the writeback to the destination register specified in the instruction.

Since convolution takes multiple cycles, the normal operation of the CPU has to be stalled during the duration of the operation. To achieve this, the *stall_alu* signal inside the execution block was set to 1 upon receiving the *ai_run* command, and it was reset back to when an *ai_result_valid* signal was received after that. Even though the unit being stalled was not the ALU, it still yielded the desired outcome.

4.5) Memory Routing Unit

CPU requests are routed to the intended destination through the memory routing unit. This unit receives CPU requests of instruction read and data read and write as inputs and generate response signals to the CPU and deliver read data as outputs. The memory routing



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

unit serves these requests as it has connections to the main memory, data cache, instructions cache, and the ten physical switches on the FPGA board.

In some write-only cases such as writing to the VGA memory, the unit is not responsible for routing the data from CPU to VGA memory, but is responsible for generating response signals to the CPU.

The cache modules are instantiated inside the memory routing unit and output hit signals used by the unit to determine the source of the data (cached or main memory) and also output cached data in the respective address to be read on cache hit. On the other hand, the unit inputs write data and address to the caches for data coming from the main memory in cases of a cache miss.

The memory routing unit is a finite state machine with 16 states detailed in figure 4-4. These states are split into states regarding instruction read, data read, data write, switches read, and operations not handled by the unit.

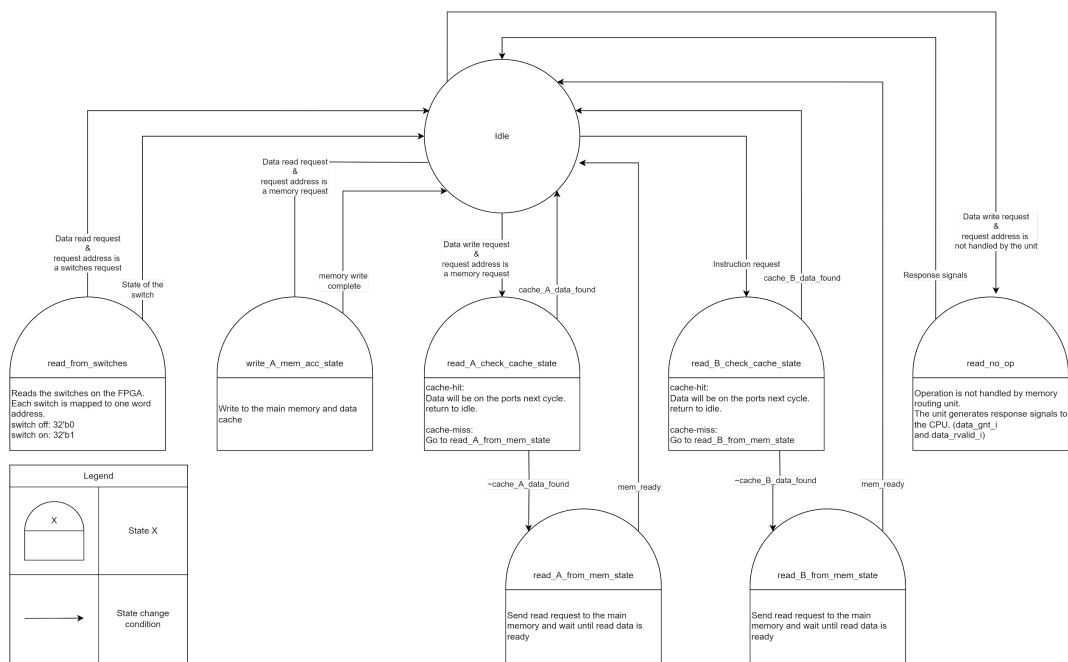


Figure 4-5: The states of the FSM inside of the request routing unit.

4.6) Building an L1 Cache for the Main Memory

Two caches were built for this project. A directly mapped cache for instruction caching, and a set-associative cache for data caching.

For the data storage inside the cache Intel's 1 port RAM IP is used and for every word stored the cache stores data and tag address. Also, for every word stored, one valid bit is stored in the registers in the FPGA. The compiler translates the IP module into M9K memory blocks (BRAM) found on the chip.



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

Valid bits are stored in registers rather than in BRAM to enable reset functionality. Upon pressing the reset button and on startup, all valid bits registers are set to logical 0. Thus, when checking for data in the cache after a reset, all cache addresses show no data. The alternative method was to store the valid bits in the BRAM along with the tag address, but this method would require a reset procedure to set the valid bits to logical 0 one per cycle before resuming operations.

Both caches are set up in a write-through configuration. Which means whenever a write is conducted, the new data is written in the cache and memory at the same time, while the CPU waits for the write to the memory to conclude before resuming operations.

The instructions L1 cache is direct mapped memory with capacity of 1KB or 256 32-bit words.

Since the processor only issues word aligned requests, The lower two bits of the address (the offset bits) are always 00. The next 8 bits, bits 2 to 9 are the index bits which correspond to which line in the cache the address is mapped to. Finally, the remaining bits of the address are the tag bits.

For each data word in the cache, the tag bits and one validity bit are also stored. When a read request comes to the cache, the index bits of the request address are used to point to the validity bit and the tag bits corresponding to the data already in cache from the validity bits table and tag bits table in the cache. The request is called a cache-hit if the validity bit in the table is true and the tag bits in the table match the tag bits of the request address. Otherwise, the request is a cache-miss and data is retrieved from the memory. When a write request comes to the cache, the data, validity bit, and tag bits of the line are all updated according to the request.

The data L1 cache is 2KB of 2-way set-associative memory. During read operations, the cache operates similarly as if there were 2 separate direct-maps connected by comparators. 2 validity bits and 2 tag bits sets are compared to the request address and if one tag bits set matches with the request address, it is determined to be a cache-hit and the data related to this tag bits set are passed to the processor.

Writing to a set-associative cache requires more steps, as first the controller needs to decide in which set the data shall be saved, and what to overwrite. For this purpose a replacement algorithm is used and this project uses the least recently used algorithm replacement for this purpose. The algorithm works by having a vector where each bit indicates which set was most recently accessed for every cache address.

For example, when writing to address $(00000000_00001111_11011000_10110100)_2$ the cache checks the least accessed vector at index of the bits 2 to 9 of the address. In this case $(00101101)_2$, and determines at which set to save the new data.



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project



Figure 4-6: Direct mapped cache (Left) and two-way set-associative cache (right).

4.7) Main Memory

For the main memory a structure of BRAM with delayed response was developed to simulate an SDRAM. This structure was chosen after the team faced challenges accessing SDRAM chips on the DE10-Lite FPGA board and having to work with multiple boards from different vendors which use different software during the project.

Furthermore, this memory structure allows for loading new programs online without needing to re-compile the design.

4.8) Peripherals

The switches, button, LEDs, and Seven Segment Displays on the DE10-Lite board were mapped to addresses to allow access to them using software. In addition, a VGA module, and basic transmission functionality for SPI, PWM, and UART protocols/interfaces were implemented. All peripherals were addressed in the same way as the main memory was addressed. All addressable ranges and their corresponding memory or peripheral is shown in the table below.

Address Range	Corresponding Unit
0x00000000 - 0x00002000	Main Memory
0x20000000 - 0x2000000C	UART
0x20010000 - 0x20010010	SPI
0x20020000 - 0x2002002C	PWM
0x20030000 - 0x20034B00	VGA
0x20040000 - 0x20040024	Switches
0x20050000	Button 1
0x20060000 - 0x20060024	LEDs
0x20070000 - 0x20070030	Seven Segment Displays

Table 4-4: Memory map



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

All memory and peripherals were instantiated inside a module named Memory and Peripherals Manager (shortened to “MPM”), which in turn was instantiated inside the top design module. The Ibex Core and the MPM communicated the address and data values in the top module. Inside the MPM, only module instantiations were done and all modules were given the requested address and data information in all requests, whether to perform an action or not was determined in the individual units based on address information. Any potential outputs from the peripherals are continuously outputted into the MPM, along with a valid signal and in the case of a read operation from the MPM module, the address is used as a select signal to pick among the results sent from the peripherals/memory.

It should be noted that the UART, SPI, and PWM peripherals were developed for the purposes of the Teknofest competition and were not developed any further afterwards, as the team members were not very enthusiastic about them.

4.8.1) UART

UART stands for Universal Asynchronous Receiver Transmitter and is used to send data asynchronously and works as a converter device between serial and parallel interfaces. A bare minimum UART system has ports called TX and RX for enabling transmission and reception respectively on one side, and the same number of ports as the number of bits in the words of data (and any start and/or end bits) being transmitted on the other side. An input clock must also be provided, and if the data transfer is expected to happen both ways, a read/write port should also be present.

The UART module has a programmable baud rate (i.e. clock) and supports only transmission. Two registers are responsible for configuring the transfer: *uart_ctrl* and *uart_wdata*. *uart_ctrl* holds the *baud_div* value used to calculate the baud rate at its most significant half and the *tx_en* signal at the first bit that controls whether data should be transmitted, and *uart_wdata* contains the data to be transferred. There are 32 8-bit wide buffers that hold the data written to *uart_wdata*. Baud rate calculation based on the *baud_div* value is done according to the expression given in the figure below.

$$f_{baud} = \frac{f_{clk}}{baud_div + 1}$$

Each data packet is 10 bits wide; 8 bits for data, one start bit, and one end bit. A 4-state finite state machine (shortened “FSM”) was implemented to manage the transmission. State transitions are performed based on a signal called *tindex* that contains the current index bit index in the data pack that is being transferred. The starting state is called the idle state and the module checks if the buffers contain data at this state and initiates a transmission sequence if there is data in the buffers. In the first transmission state *tindex* is 0xF and the start bit is sent, and *tindex* is incremented by one, which then becomes 0 since it is 4 bits wide. The next state occurs when *tindex* is between 0 and 7 (inclusively). At each baud rising edge the bit at *tindex* is sent and *tindex* is incremented by one. The next state begins when



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

tindex is 8 and in this state the end bit is sent. Then *tindex* is set to 0xF to be ready for another transmission.

4.8.2) SPI

SPI stands for Serial Peripheral Interface and performs serial and synchronous data transection using four ports: SPI clock, MISO (master in slave out), MOSI (master out slave in), and CS (chip select, also called slave select).

The SPI module has a programmable clock and only supports transmission. In order to use the SPI module to transmit data, three registers inside the module need to be written to. Parameters that are set based on the information in these registers are the SPI clock speed and the number of bytes to be transmitted. These registers are called *spi_ctrl*, *spi_wdata*, and *spi_cmd*. The least significant half of *spi_ctrl* is used to set the clock divider limit, *spi_wdata* contains the data to be transmitted, and the least significant 9 bits of *spi_cmd* contains the number of bytes to be transmitted. The number of bytes has to be a multiple of 4 (i.e. integer multiple of 32 bits in length), and there are 8 buffers that can each hold 4 bytes of data. Therefore, at most 32 bytes can be transmitted in one sequence.

The polarity and phase are always assumed to be zero, i.e. the clock is low when idle, data is toggled on the falling edge of the SPI clock, and sampling is done on the rising edge. Five internal signals were created to perform transmission using the parameters provided. These are *sck_counter*, *tindex*, *tactive*, *tbytes*, and *tbytes_counter*. *sck_counter* is the signal used to control the SPI clock. It's incremented in each system clock rising edge until it reaches the limit provided in the *sck_div* section of the *spi_ctrl* register, which is at bits 31:16. The mathematical expression for the SPI clock is given in the figure below.

$$f_{sck} = \frac{f_{clk}}{2(sck_div + 1)}$$

Any write operation to the *spi_cmd* register pulls the *tactive* signal from 0 to 1. On the first SPI clock rising edge where *tactive* is 1, the CS signal which is active-high is buffered to 1 with a delay of one cycle. As long as the *tactive* signal is 1, the index of the bit sent to the MOSI port is incremented on each falling edge of the SPI clock. The slave side is expected to sample the data on the rising edge. At every 8 bits of transmission, the *tbytes_counter* signals is incremented by one, and this signal is used to tell the module when to stop transmitting based on the number of bytes to be transferred specified in the *length* section of the *spi_cmd* register which is found at bits 8:0. After a transmission stops, a new write operation to the *spi_cmd* register must be performed to initiate another transmission sequence.

4.8.3) PWM

Pulse Width Modulation (PWM) is a digital control signal that allows the controller to control the strength of the signal by quickly switching between logic 1 and logic 0 signals. The time of logic 1 signal in each period is called the duty cycle, and is expressed as



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

percentage. The optimal PWM generator would have short periods yet still allow for fine duty cycles changes.

For this project, a PWM module was developed with two operating modes. A standard mode where the duty cycle is set and does not change. The other mode is heartbeat mode where the duty cycle changes from a minimum threshold to a maximum threshold every full cycle by a constant factor.

4.8.4) VGA

VGA stands for Video Graphics Array and is a relatively simple standard for video transmission. There are 5 mandatory signals in VGA, three of which denote red, green, and blue values for the given pixel, and the remaining two are horizontal and vertical synchronization signals that tell the slave when the scanning of a single row and all rows are completed, respectively. A VGA slave (e.g. a monitor) iterates through every single pixel in its display using an internal clock whose frequency is determined by the VGA standard and varies based on the resolution of the display. The VGA module on the DE10-Lite FPGA only supports 640×480 resolution, which needs to run on a clock speed of 25 MHz.

Two counter modules -one for scanning the current row (i.e. horizontally) and one for counting the rows (i.e. vertically)- were created and instantiated inside the VGA module. An additional clock divider was used to half the clock frequency of the FPGA, since the FPGA clock runs at 50 MHz. The signal timing needs to match the timing diagram shown in the figure below.



Figure 4-7: VGA timing diagram

The horizontal timing shown at the top is performed 525 times (once for each row) at each frame. A frame ends when the vertical timer reaches 524. the horizontal synchronization is active in the first 96 cycles of a horizontal scan, and the vertical synchronization signal is active in the first two cycles of a vertical scan. In horizontal scanning, at cycles 96-143 (inclusively) all color signals must be zero. Likewise, in vertical scanning, at cycles 2-34 (inclusively) they must also be zero. Horizontal scanning continues after all addressable pixels in a row have been addressed, until the 800th cycle. During this time all color signals must also be zero. Similarly, vertical scanning continues until the 525th scan cycle and all color signals must be zero after video scanning ends at the 515th cycle. Note that when talking about vertical scanning, “cycle” refers to the time period between the start and end of a horizontal scan, not the VGA clock cycle.



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

All pixel color values are stored in a block RAM -called the video memory-, implemented using the same Intel IP used for main memory. Each word was set to 16 bits even though color signals are 4 bits each, resulting in a total color depth of 12 bits, because the IP only allowed word sizes at powers of two. The most significant 4 bits were ignored, and the following three groups of 4 bits were used as red, green, and blue values in that order. Due to storage limitations of the DE10-Lite board, only a resolution of 160×120 (keeping the original aspect ratio) could be supported with this video memory at most. Each word in the video memory corresponds to a 4×4 group of pixels. The counter values from the horizontal and vertical counter modules are used to calculate the address of the pixel in the video memory. More specifically, the row value is vertical counter minus 35, and the column value is horizontal counter minus 144. Then the corresponding address inside the video memory is calculated as $160 * (\text{row} \gg 2) + (\text{column} \gg 2)$.

The address input of the video memory is connected to the output of a multiplexer system whose inputs are the address for any hypothetical write operation and the regular VGA pixel scanning address explained in the previous paragraph. The select input for this multiplexer system is the *wen* signal which is an input that indicates that the CPU requests a write operation to the video memory. In this case, the pixel in that CPU clock cycle is skipped (the monitor draws a black pixel) and the video memory is written to. Since the display is 60 Hz and the use cases in the scope of this project do not require a pixel perfect and smooth display, this is not considered to be a problem, most of the time the human eye cannot detect a couple of black pixels on a screen that refreshes at 60 Hz. However, if the software makes a very high number of write operations continuously, it can lead to some visual artifacts on the display. The video memory is fed with the system clock which is double the frequency of the VGA clock to mitigate this issue to some extent.

4.8.5) DE10-Lite GPIO

The DE10-Lite has two buttons, ten switches, ten LEDs, and six seven segment displays. All of these have been assigned addresses and they can be read from or written to by programming, except for the top button which has been assigned to system reset.

The address map showing the switches is in table 4-4 above, and an explanation on how the read requests are handled is shown in section 10.5.

4.9) Demo

Multiple small programs will be run on the FPGA in the demo. Memory initialization files in .hex format will be prepared for the demo beforehand. These files will be used to program the FPGA for the different demos.

The main idea of the demo programs is to prove that the SoC does indeed work as intended and verify different features mentioned in this report.

5. PROJECT TARGETS AND SUCCESS CRITERIA:



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

5.1) 8-bit CPU

This work packet's success is decided by testing whether the developed processor can run a program that uses the limited number of implemented instructions and generate the correct result. The program chosen for this test shown in Figure 5-1 includes array operations that will be performed using indirect memory access, and a while loop which will be performed using conditional jumps.

```
int A[5] = {2, 10, 8, 17, 9};
int B[5];
i = 0
while (i <=5)
{
    B[i] = A[i] + 2;
    i++;
}
```

Figure 5-1: The C code that will be tested on the non-standard CPU.

The test was conducted on simulation using Icarus Verilog and on the DE10-Lite FPGA board by displaying the result on the 7-segment displays (see section 4.3 for details on how this is done).

5.2) Running the Ibex Core on the FPGA

The objective of this work packet is to convert the Ibex Core from SystemVerilog to Verilog-2005, compile it, and program the Intel DE10-Lite board with it. The success criteria for this work packet is not getting any error messages in Intel Quartus Prime Lite which is used to compile the design and program the FPGA.

5.3) Testing the Ibex Core

The objective of this work packet is to test the standard RISC-V instructions on the Ibex Core. To achieve this, software was written in assembly to perform the instructions and display the results on the seven segment displays. If the theoretical result matched the value observed on the seven segment displays, it was counted as a success. Not all of the 47 standard instructions were tested as it was deemed unnecessary. Since the core is open source and well trusted in the community, about 10 of the instructions were tested and yielded successful results, then the rest were assumed to be working as well.

5.4) Non-Standard Instructions

The successful execution of these instructions is the success criteria, and it was measured by checking the destination register contents after the execution is complete. The process is very similar to the workflow described in section 4.3. It should be noted that a custom built version of the GNU RISC-V toolchain was used after this point that supports the



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

custom instructions implemented in this project. This custom build was published by Tübitak for the purposes of the Teknofest competition.

5.5) The Memory Routing Unit

Since the most critical objective of the memory routing unit is the correct integration with the L1 caches, the unit was first tested and verified with the instruction and data caches and then the tests were expanded to include reading from switches and generating response signals for operations not handled by the unit.

The success criteria of this part is the correct handling of requests coming for the processor data and instruction request outputs. The requests may be handled differently depending on the request address, and it is essential that all types of requests are handled appropriately.

These tests were done on simulation first and then testing on hardware was conducted.

5.6) Building an L1 Cache for the Main Memory

The success criteria for this part is the ability for the L1 cache to interface with the memory routing unit correctly with hit status and cached data. As such the caches were first tested by running a series of write and read requests and verifying the algorithms such as the replacing algorithm for the data cache and the reset functionality.

Then they were tested with the memory routing unit and finally with the CPU on both simulation and hardware.

The nature of these tests is that they are pass/fail tests, where in the event of failure the cause of this failure is identified and the logic is corrected. As such, it can be said that all tests passed with a 100% success rate.

5.6) Main Memory

The BRAM based main memory was tested by reading and writing to it and testing different delay periods.

5.8) Peripherals

The SPI, UART, and PWM peripherals were tested in simulation only. Since they were only built for the Teknofest competition, there was no incentive for the team members to work on them afterwards. This is why these modules were marked as *optional* in the previous report. The success of these modules are determined by the waveforms in the simulation matching the correct forms for the standard. For SPI, this standard was explained in the Teknofest Chip Design Competition Specifications document.

5.8.1) UART

The success criteria is the successful transmission of an 8-bit data packet in the correct baud rate configured by the programmer (see section 4.8.1).



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

5.8.2) SPI

The success criteria is the successful transmission of a 32-bit data packet in the correct SPI clock configured by the programmer (see section 4.8.2).

5.8.3) PWM

The success criteria is the successful generation of the PWM signal in both standard and heartbeat mode with thresholds and duty cycle correct as configured by the programmer

5.8.4) VGA

The first success criteria is the successful display of an image on a monitor. This is any 160×120 image that is converted to the Intel memory initialization hex format. A Python script using the Python Imaging Library was written to perform this conversion. The second success criteria is the display of a pixel modified through write operations requested by the CPU in the frame following the operations at the correct coordinates on the monitor. This can be directly observed by the human eye.

5.8.5) DE10-Lite GPIO

The success criteria for the switches and the lower button (reminder: the upper button is wired to system reset) is reading the correct values from these inputs using load instructions directed at the addresses they are assigned. The success criteria for the LEDs and GPIO is the correct display of a hexadecimal value that was written to their respective addresses.

5.9) Demo

This work packet's success is determined by the demo to be presented at the end of this project. The demo will consist of multiple small programs and will be prepared in a way such that a non-academic person can comprehend the working status of the FPGA. The demo will be interactive and have display output over the VGA port or the seven segment displays on the DE10-Lite board.

One of the programs to be presented at the demo is completed and tested and it's a program that continuously reads the input from the switches and calculates and displays the hamming distance between the first and the second half of the ten switches on the seven segment display using the custom *hmdst* instruction detailed in section 10.8.5.

6. RISKS AND B PLANS:

Work Package #	Risk	B-Plan
WP2	GNU RISC-V toolchain raising errors	Try to debug simple errors, otherwise research different compilers for RISC-V



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

WP 3	DE10 FPGA not having enough logic units	Ask the department for another FPGA which has more logic units.
WP 6	Intel SDRAM controller IP not working properly with the FPGA	Use a 3rd party SDRAM controller.
WP 6	Can't find working 3rd party SDRAM	Simulate SDRAM with BRAM with added delay.
WP 6	FPGA does not have enough GPIO ports to support all planned peripherals.	Implement only one or two of the peripherals.
WP 6	The demo can not be finished on time.	Switch to a simpler demo. The project's goal is hardware development and demonstration application has lower priority.

7. WORK TIME PLAN OF THE PROJECT:

The first step was the development of the 8-bit CPU because a fully functional complete CPU is an incredibly complicated device that could be difficult to understand if diven straight into. Thus it was reasoned that some preliminary experience with CPU development would be helpful in understanding more complicated CPU structures later on. Even though the Ibex core will be used as the base CPU, it will be extended for custom instructions, so a thorough understanding of the inner workings of a CPU is required in order to complete this project.

The next step was running the Ibex core, because before anything else could be done, the base hardware was necessary to be functional, as it is the foundation for the entire SoC.

Once there is a working base CPU, it needs to be tested to make sure that it is actually working correctly. This is why the third step is the "Testing the Ibex Core" work package. Before moving on to the next steps, it was important that it was confirmed that the foundation of the project is functional, which is explained in section 4.3.

After verifying the core, the development split into custom instructions followed by peripherals and supporting memory elements.

While İbrahim Yuşa Çetin worked on cryptography instruction followed by the AI accelerator and then the peripherals, Amer Alhamvi worked on the caches and the memory routing unit.

During the second half of the project, the focus shifted to working towards Teknofest competition. This included writing the detailed design report for the competition and



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

switching the FPGA being used from Intel Altera FPGA to a Xilinx FPGA. Finally, the competition was held in-person in Istanbul and lasted for 5 days .

After the competition, the project was migrated back to the Intel DE10-Lite board, as both members had one at their disposal as opposed to there being only one Xilinx FPGA. The VGA module was developed in this period, and the GPIO peripherals were also mapped to memory addresses for software access.

The last few weeks were spent on ironing out all the elements and preparing the final demonstration.

8. DEMO PLAN:

The demo plan is detailed in section 4.9, 5.9, and 10.9, while the test shown in part 10.8.5 will be among the programs to be demonstrated.

9. FINANCIAL EVALUATION:

The only expenditure of this project was the acquisition of the Nexys A7 FPGA to participate in the teknofest competition. Otherwise, the school already had the DE10-Lite FPGA boards that were used for this project.

10. RESULTS:

10.1) 8-bit CPU

We have completed the development and testing of a non-standard 8-bit CPU. We tested it both on simulation using Icarus Verilog and on the DE10-Lite FPGA board.

Our processor can do more sophisticated instructions than the one in chapter 13 of (LaMeres, 2017) we used as a reference. Our processor can use both registers as a destination for addition and subtraction operations and also can address the main memory with indirect addressing.

10.2) Running the Ibex Core on the FPGA

The Ibex Core was successfully compiled in Intel Quartus Prime Lite and the DE10-Lite board was programmed without errors. Since the results of the next work packet also indicate the success of this work packet, the results for this work packet are omitted from the report.

10.3) Testing The Ibex Core

Although the project has its focus on hardware design, the most feasible way to verify it is by creating software for it and checking whether it can run successfully. Hardware is a lower level than software in terms of abstraction, and software is only meaningful in the presence of hardware that can interpret it. Therefore, the correct execution of software necessitates the correct execution of hardware, but not the other way around.



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

A build automation script was developed in POSIX shell scripting to compile the software written in RISC-V assembly language and convert it into a format recognized by Icarus Verilog or Quartus Prime. One of the registers in the register file inside the CPU was connected to the seven segment displays. Given in the figures below are the last program that was tested finally conclude this testing, in its C-like pseudocode and the corresponding assembly code, followed by the GTK Wave waveform showing the results, followed by the result being shown on the DE10-Lite board on the seven segment displays. Note that the values shown in the waveforms and on the seven segment displays are in hexadecimal format.

```
sum = 0
x = 14
while (x >=10)
{
    sum = sum + x
    x--
}

lui x4, 1
srli x4, x4, 12
lui x3, 10
srli x3, x3, 12
lui x1, 0
lui x2, 14
srli x2, x2, 12
cond:
bgeu x2, x3, loop
jal x0, finish
loop:
add x1, x1, x2
sub x2, x2, x4
bgeu x2, x3, cond
finish:
jal x0, finish
```

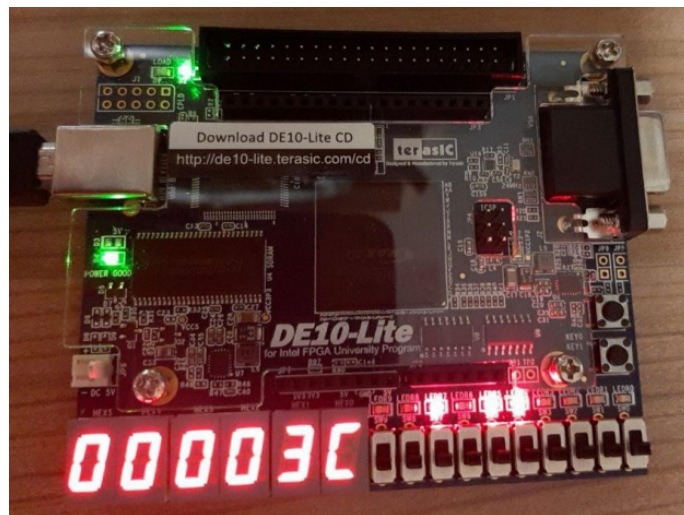


Figure10-1: Sample assembly code (left) and C code (top-right) used for testing the Ibex core, and the result when the program is run on the FPGA (bottom-right).

10.4) Non-Standard Instructions

The non standard instructions were tested in the same manner as the standard instructions as shown in the subsection above. Only this time, a custom built version of the GNU RISC-V Toolchain had to be used that supported these instructions. As mentioned earlier, this custom GCC suite was acquired from the GitHub page of Tübitak Tütel.



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

10.4.1) Cryptography Instructions

See figure 10-13 in the peripherals subsection (10.8.5) for a demonstration of the *hmdst* instruction working.

10.4.2) Convolution Instructions

Figure 10-2 shows the testing code and the resulting waveform. Note that the *.en* suffix at the end of load instructions denote that two load operations (one from each register provided) are to be done. *x18* is the display register.

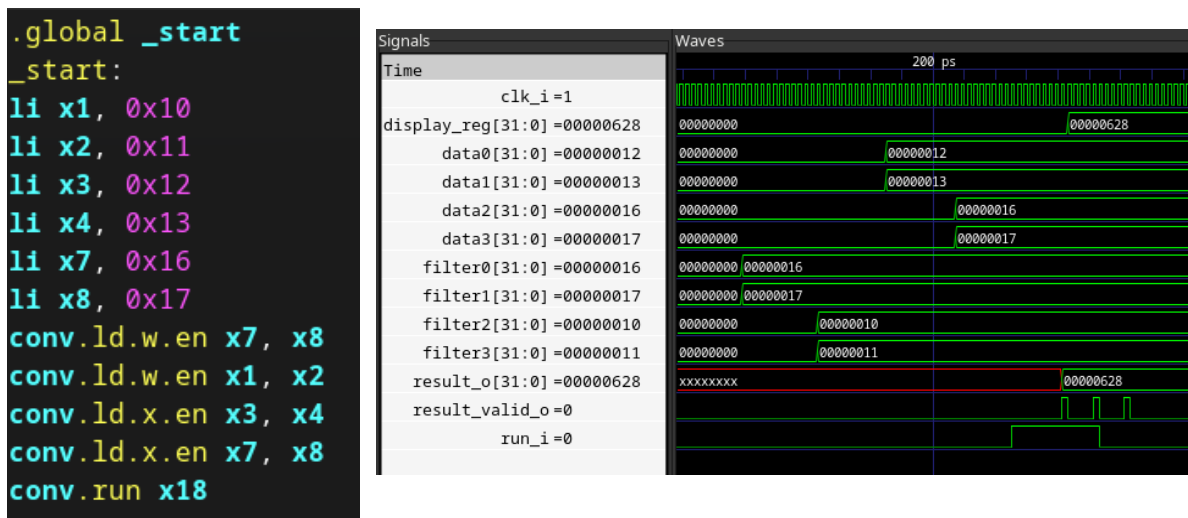


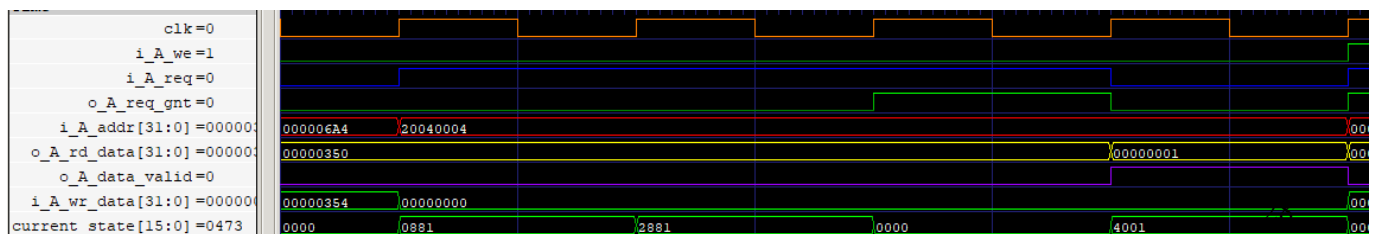
Figure 10-2: A testing of convolution instructions

10.5) The Memory Routing Unit

Figure 10-3 shows how a read from a switch is handled by the memory routing unit. In blue is the request signal input from the CPU and just below that in green is the first response signal called request granted is sent as soon as the unit is not busy. This signal indicates to the CPU that the unit has captured the request and the next request can be sent. In red is the hexadecimal address of the request 0x2004004, which corresponds with the first switch from the right on the board (since each switch is treated as a 4-bit word).

The next signal is the read data and it changes at the same time as the data valid signal. The data valid signal is the second and last response signal to the CPU meaning the read data is ready to be read by the CPU. In this case the switch was set to the upper position meaning logical 1 and returns 1 to the CPU.

After that is the write data (data to be written) signal from the CPU but is irrelevant for this read operation. Finally, the current state of the FSM inside the unit is shown. The request is captured on the state *idle* 0x0 and processed on the state *read_from_switches* 0x4001.





Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

Figure 10-3: Signals inside the memory routing unit during a read from switch operation.

The next figure, figure 10-4 shows how 2 consecutive cache hit read operations are done. When the request granted signal is set to logic 1, the input address changes to the next request's address. 2 cycles later the response signal data valid is set to logic 1 to indicate that the data is ready. Each read operation takes 3 cycles to complete.

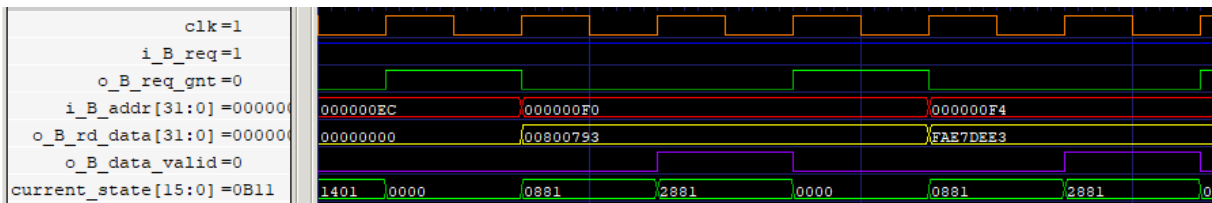


Figure 10-4: Signals inside the memory routing unit during a read with cache hit.

Figure 10-5 shows a write to the VGA memory operation which is an example of an operation not handled by the unit. In this case the unit is only responsible for generating the response signals to the CPU and not actually do read or write operations. The write operations are handled by the receiving unit which is directly connected to the CPU data ports and are done when the correct address is requested.

The memory routing unit generates the request granted signal at the same cycle the request is received and generates the data valid signal (which means the write operation finished) on the second cycle.

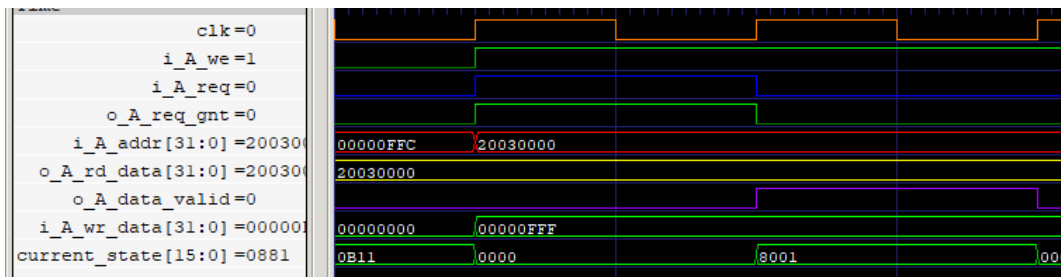


Figure 10-5: Signals inside the memory routing unit during an operation that is not handled by the unit.

10.6) Building an L1 Cache for the Main Memory

Figure 10-6 shows a cache test conducted with memory routing unit integration. This is a test of accessing addresses with similar cache addresses but with different tag addresses. The data output for both addresses comes from cache.

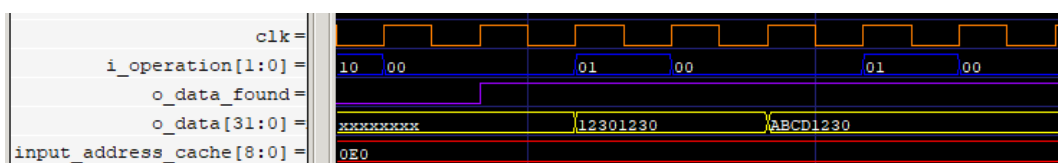


Figure 10-6: Data cache test was conducted successfully.



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

10.7) Main Memory

The main memory was tested for correct signal generation to simulate an SDRAM controller. This is an example of the main memory in operation executing a writing operation. The *o_data_valid* signal in writing operations means that the memory write operation is finished and the memory is ready for next request.

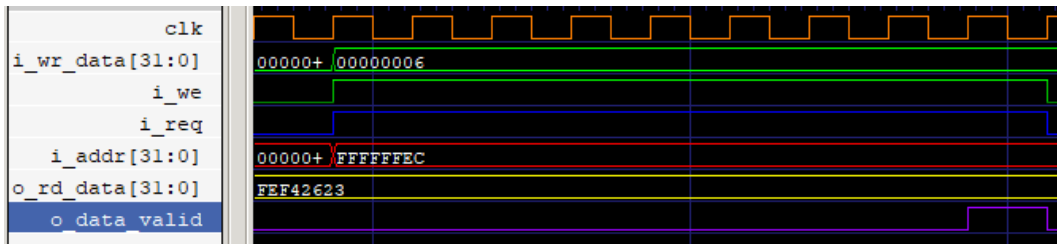


Figure 10-7: A data write operation handled by the main memory.

10.8) Peripherals

10.8.1) UART

The program below produces the waveform given below it.

```
.global _start
_start:
#### STEP 1: setting up register uart_ctrl ###
lui x1, 0x00630; // load the data to register x1 (part 1)
addi x1, x1, 0x001; // load the data to register x1 (part 2)
#### uart_ctrl address: 0x2000000
lui x2, 0x20000; // load the address to register x2 (part 1)
addi x2, x2, 0x000; // load the address to register x2 (part 2)
sw x1, 0(x2); // write the first data to register uart_ctrl
#### STEP 2: filling register uart_wdata with 8 bits of random data ###
#### random data: 0x36
lui x3, 0x00000; // fill register x3 with random data (part 1)
addi x3, x3, 0x036; // fill register x3 with random data (part 2)
#### uart_wdata address: 0x200000C
lui x4, 0x20000; // load address to register x4 (part 1)
addi x4, x4, 0x00C; // load address to register x4 (part 2)
sw x3, 0(x4); // write the random data to register uart_wdata
```

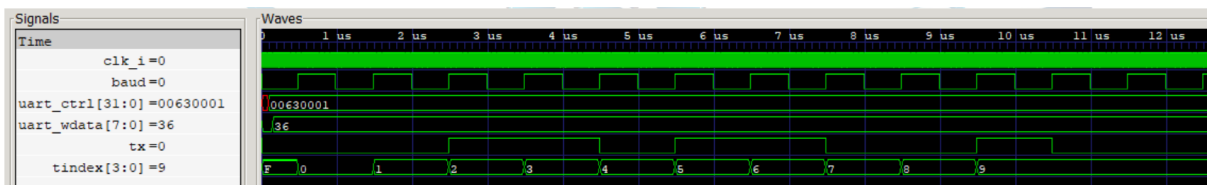


Figure 10-8: The testing of UART.

10.8.2) SPI

The program below produces the waveform given below it.



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

```
.global _start
_start:
#### STEP 1: set up the spi_ctrl register ###
lui x1, 0x00040; // load the first data to register x1 (part 1)
addi x1, x1, 0x003; // load the first data to register x1 (part 2)
lui x2, 0x00040; // load the second data to register x2 (part 1)
addi x2, x2, 0x001; // load the second data to register x2 (part 2)
#### spi_ctrl address: 0x20010000
lui x3, 0x20010; // load address to register x3 (part 1)
addi x3, x3, 0x000; // load address to register x3 (part 2)
sw x1, 0(x3); // write the first data to register spi_ctrl
sw x2, 0(x3); // write the second data to register spi_ctrl
#### STEP 2: filling register spi_wdata with random data ###
lui x4, 0x3b857; // put random data to register x4 (part 1)
addi x4, x4, 0x2a9; // put random data to register x4 (part 2)
#### spi_wdata address: 0x2001000C
lui x5, 0x20010; // load address to register x5 (part 1)
addi x5, x5, 0x00C; // load address to register x5 (part 2)
sw x4, 0(x5); // write random data to register spi_wdata
#### STEP 3: set up the spi_cmd register ###
addi x6, x0, 0x4; // load the data to register x6
#### spi_cmd address: 0x20010010
lui x7, 0x20010; // load address to register x7 (part 1)
addi x7, x7, 0x010; // load address to register x7 (part 2)
sw x6, 0(x7); // write the commands to register spi_cmd
```

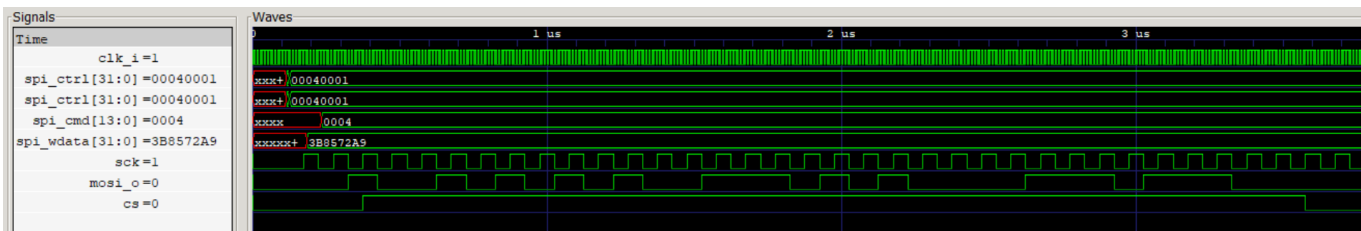


Figure 10-9: The testing of SPI.

10.8.3) PWM

Figure 10-10 shows PWM signal generated in standard mode with duty cycle of 44% with period set to 16 clock cycles. The PWM output signal is gated and as such is shifted one clock cycle and changes at *clk_counter* 2.

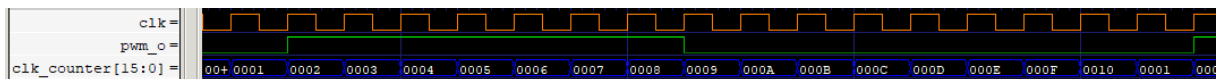


Figure 10-10: PWM signal generated in standard mode

Figure 10-11 shows PWM signals generated in heartbeat mode where the duty cycle starts at 12% and goes to 83% in 20% steps. The period was set to 60 clock cycles.



Figure 10-11: PWM signal generated in standard mode

10.8.4) VGA

Given below is the photograph of a monitor that is connected to the DE10-Lite board through the VGA port. The video memory in the VGA module was initialized with an image



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

of the university logo. The memory initialization file was obtained using the Python script mentioned earlier.

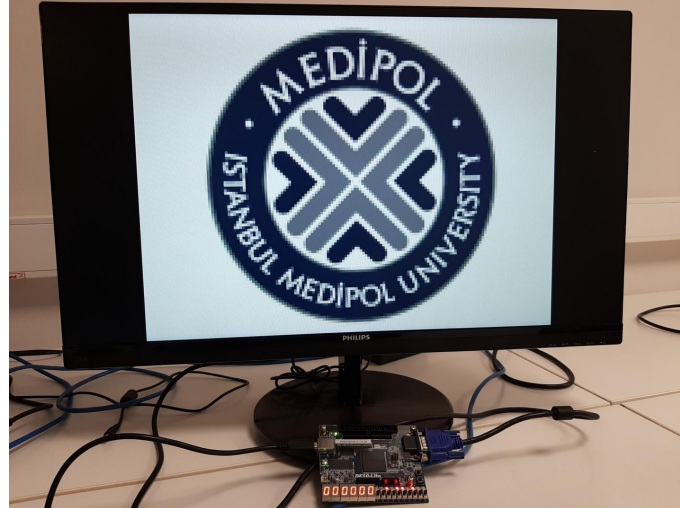


Figure 10-12: VGA output at startup.

10.8.5) DE10-Lite GPIO

Calculating hamming distance between the first five and last five switches and displaying the result on the seven segment display is done using the code below. The results are shown in the image that follows it. Note that this code is running on a testing version of the project where the seven segment display is connected to the contents of the x18 register, not the address 0x200700XX.

```
// Define generic assembly functions
#define STR(i) #i
#define LOAD_IMM(reg, imm) asm("li x" STR(reg) "," STR(imm))
#define LOAD_DIR(dst_reg, addr_reg) asm("lw x" STR(dst_reg) ",0(x" STR(addr_reg) ")")

// Define program constants
#define ADDR_VAR1 0x000004E0 // 0x00000138
#define ADDR_VAR2 0x00000514 // 0x00000145

int _start() {
    while (1==1) { // indefinite loop
        // Define address variable
        int* addr0 = (int*)ADDR_VAR1;
        int* addr1 = (int*)ADDR_VAR2;

        // Read switches into main memory
        int* sw = (int*)0x20040000;
        *addr0 = sw[4]*(8*2) + sw[3]*(4*2) + sw[2]*(2*2) + sw[1]*(1*2) + sw[0]*(1);
        *addr1 = sw[9]*(8*2) + sw[8]*(4*2) + sw[7]*(2*2) + sw[6]*(1*2) + sw[5]*(1);

        // Save addresses of the variables to registers
        LOAD_IMM(3, ADDR_VAR1);
        LOAD_IMM(4, ADDR_VAR2);

        // Load the variables from main memory to variable registers
        LOAD_DIR(1, 3);
        LOAD_DIR(2, 4);

        asm volatile( "hmdst x18, x1, x2;" );
    }
    return 0;
}
```




Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

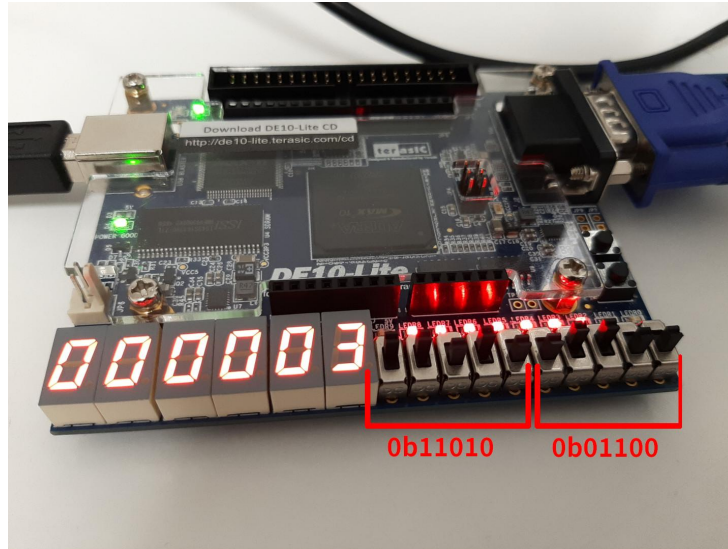


Figure 10-13: The hamming distance calculation demo running on the FPGA

10.9) Demo

See figure 10-13 in subsection 10.8.5 to see how one of the software to be shown in the demo (hamming distance between the first and last 5 switches) was tested and the results were obtained. More software is planned to be presented at the demo, but the programming for them has not been completed as of the writing of this report.

11. DISCUSSION:

It can be said that almost all main goals of this project and proposed methods were successful with no major deviations of the methods discussed during the project proposal. The exceptions are the SDRAM part where it was proposed to use the SDRAM chip on the FPGA board for the main memory, and insufficient testing of the AI Accelerator on the FPGA. It should be noted that even though the peripherals UART, SPI, and PWM have limited features, team members had already clarified that these modules were optional for the purposes of this project in the previous report and in earlier presentations. It could also be argued that the development of software for the demo was left to the last-minute and it had to be rushed a little bit.

The failure in completing the SDRAM comes down to these points:

- The SDRAM controller shipped with Intel Quartus Prime software did not support the memory chip on the FPGA board.
- Their documentation of available open-source SDRAM controllers was lacking.
- It is said that there exist legacy Altera SDRAM controllers that work with this FPGA board, but we could not find them on the internet nor on the Aletra website.

The AI Accelerator has been successfully tested on simulation, but it could not be executed on the hardware even though the Quartus compiler did not produce any errors or



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

critical warnings related to the module. The reasons for this behavior are currently unknown to the team members and this issue will continue to be worked on in the days leading to the presentation.

Beside these, all work packages were developed, tested, and integrated as planned.

12. CONCLUSION:

The project has been an incredible journey toward understanding the inner workings of CPUs and computers in general. Team members got to work on FPGA and learn about ASIC implementation flow all the while they participated in the national competition part of Teknofest and met people from the field.

The project aims to create a custom SoC with an extended 32-bit RISC-V CPU, that is capable of generating display output over VGA and has SPI, UART, and PWM peripheral modules. The SoC is equipped with an L1 cache system and supports 11 extra non-standard instructions. Finally, a visual demo was developed for the final presentation

First, a non-standard 8-bit CPU was developed, simulated, and tested. Then an IbexCore RISC-V CPU was tested first with a simple testing method and later with a more sophisticated testing structure on simulation and in the real world on the FPGA board. After that the memory system was built while simultaneously non-standard instructions were being integrated. Then the peripherals were developed.

Then the team focused on preparing for the Teknofest competition including migrating the project from Intel Altera based FPGA to a Xilinx FPGA to adhere to the competition's specifications and then going to the competition.

Finally, the demo was prepared.

13. PLAN FOR FUTURE STUDIES:

Should the DE10-lite FPGA be used in future studies, it is important to be able to utilize the 64MB SDRAM chip on the FPGA board. Even if the development will continue with different FPGA boards, it would be beneficial to be able to utilize the SDRAM chip on said chip.

The Teknofest competition is said to continue running in the next few years where the competition specifications will not see major changes. And since the current form of the project is compatible with the specifications but needs further developments, setting the Teknofest competition as the goal for future works is viable.

Another important thing to do that the team members realized too late into the development is that a GPIO communication module can be used to program the CPU without requiring any intervention from Intel Quartus Prime. This would save a lot of testing time and implementing a module for this purpose should be a priority in the future.

The OpenLane flow could be another point of focus for future studies. If a functional GDS file can be obtained, the design can be sent to a semiconductor fab for manufacturing.

14. ASSESSMENT OF ENGINEERING COURSES:



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

In the Digital Logic Design course, we learned the most basic hardware abstraction element; the gate, and how to use abstraction to build more complex hardware elements with higher abstraction levels. This is a fundamental idea in hardware design found everywhere, for example how we treated The Ibex Core as a logic element with inputs and outputs without considering the logic inside of it.

In the Microprocessors course we had practice in working with microprocessors directly using assembly. And we also took 3 programming courses in Python and C/C++, with our good understanding of them we were able to learn how to use Verilog HDL easily.

In the Introduction to Formal Language and Automata Theory course we saw the theory of finite automaton, pushdown automaton, and Turing Machine which all appear in a processor.

Finally, we studied multiple mathematics and computer science courses which built our knowledge and experience as computer engineers to be able to work on this project.

15. REFERENCES:

- Waterman, A. S. 2016. "Design of the RISC-V instruction set architecture." University of California, Berkeley.
- Asanović, K., Patterson, D. A. 2014. "Instruction Sets Should Be Free: The Case for RISC-V." EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146.
- Wilkes, M. V. 1965. "Slave memories and dynamic storage allocation." IEEE Transactions on Electronic Computers, (2), 270-271.
- Smith, A. J. 1982. "Cache memories". ACM Computing Surveys (CSUR), 14(3), 473-530.
- Patterson, D.A. and Hennessy, J.L. 2013. "Computer Organization and Design, The Hardware/Software Interface" (5th ed.). Elsevier Inc.
- Stallings, W. 2015. "Computer Organization and Architecture Designing for Performance" (10th ed.). Pearson Education.
- LaMeres, B, J. 2017 "Introduction to Logic Circuits & Logic Design with VHDL" (1st ed.) . Springer.
- Snow, Z. 2019, February 8. "sv2v: SystemVerilog to Verilog" Retrieved June 8, 2023, from <https://github.com/zachjs/sv2v>
- lowRISC. 2018. "Ibex RISC-V Core". Retrieved June 8, 2023, from <https://github.com/lowRISC/ibex>



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

16. PROJECT ACTIVITIES AND WORK PLAN

Table 1 The Work-Activity Plan for Project 1

Work and Activity Project 1	Responsible Group Member	Timeline (in week number)													
		1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.
1. 8-bit CPU	Amer - Yuşa														
2. The Ibex-Core	Amer - Yuşa														
3. Testint the Ibex Core	Amer - Yuşa														
4. Non-standard instructions	Yuşa														
5. The request routing unit	Amer														
6. Main Memory	Amer														
7. L1 Cache	Amer														
8. Peripherals	Yuşa														
9. Demo	Amer - Yuşa														

Table 2 The Work-Activity Plan for Project 2

Work and Activity Project 2	Responsible Group Member	Timeline (in week number)													
		1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.
1. 8-bit CPU	Amer - Yuşa														
2. The Ibex-Core	Amer - Yuşa														
3. Test Software	Amer - Yuşa														
4. Non-standard instructions	Yuşa														
5. The request routing unit	Amer														
6. Main Memory	Amer														
7. L1 Cache	Amer														
8. Peripherals	Yuşa														
9. Demo	Amer - Yuşa														
Teknofest	Amer - Yuşa														

Red denotes time spent for or at Teknofest competition.



Istanbul Medipol University
School of Engineering and Natural Sciences
Graduation Project

16.1 LIST OF WORK PACKAGES

Table 3 Detailed Definition of Work and Activity

WP No	Detailed Definition of Work and Activity
1	This work packet consists of the design and development of a non-standard 8-bit CPU and testing it with simulation and on the FPGA.
2	This work packet consists of the compilation, synthesis, programming, and running The Ibex Core on simulation and on the FPGA. Including correctly mapping FPGA pins to the CPU.
3	This work packet contains the successful testing of the Ibex core using a test program.
4	This work packet consists of the integration of 11 non-standard instructions shown in the report above into the ISA. These instructions include 6 instructions for operations related to cryptography and 5 others for performing 2D convolution on data.
5	This work packet consists of developing, testing and integrating a routing unit that will be responsible to handle all the requests from CPU and directing control and data signals to the appropriate interface units. These interface units include the main memory, Cache, and peripherals interface.
6	This work packet consists of developing a main memory system which can simulate SDRAM access delay while also allowing for programs to be uploaded online.
7	This work packet consists of developing an L1 cache and controller and integrating it with The request routing unit to build a memory hierarchy and benefit from high-speed on-chip memory to increase performance by lowering average memory access delay.
8	This work packet consists of developing a VGA module, one or two peripheral interfaces starting with SPI, PWM and if time permits UART. Each developed interface will be connected with the CPU by modifying the request routing unit, and will have a test routine to test its simple functionalities.
9	This work packet consists of developing the final demo of the project to be presented to the faculty at the end of the project. The demo to be developed will be an interactive demo with display output.



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

Table 4 Work package targets, their assessment, and the contribution of each work package to the overall project success.

Work package	Target	Measurable outcome	Contribution to overall success(%)
1. 8-bit CPU	Running a non-standard 8-bit CPU on simulation and FPGA.	The successful execution of a code segment with the nonstandard CPU.	100
2. The Ibex-Core	Running The Ibex Core on simulation and FPGA.	The successful execution of a code segment with the nonstandard CPU.	100
3. Test Software	Writing short programs to test the hardware.	Software executes successfully.	100
4. Non-standard instructions	Integrating 11 non-standard instructions to our CPU.	The successful execution of the 11 non-standard instructions within a test code.	100
5. The request routing unit	Develop a request routing unit to manage all CPU interface requests with memory, cache, and peripherals interface.	The successful routing of all requests of a testing request code per the processor's address map.	100
6. Main Memory	Developing main memory with codes can be loaded easily, and can simulate SDRAM access delay.	The successful implementation of the main memory using BRAM and delay.	-
7. L1 Cache	Implementing an L1 Cache for data stored in the main memory.	The L1 Cache acts as a memory level before the main memory successfully.	100
8. Peripherals	Implementing a number of peripherals interfaces from the list (SPI, PWM, UART). (This is an optional work packet)	UART, SPI, and PWM can transmit data in the correct configuration, VGA can display images, switches and the bottom button can be read from, seven segment displays and LEDs can be controlled with write operations.	optional wp 60
9. Demo	Running a demo on the FPGA that takes inputs from the user and outputs a display signal over VGA port.	The successful presentation of the demo.	100



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

Table 5 The work package distribution to project team members: Who works on which work package? Specify the percentage contributions.

WORK PACKAGE DISTRIBUTION									
Project Member	WP1	WP2	WP3	WP4	WP5	WP6	WP 7	WP 8	WP9
Amer	50	50	10	0	100	100	100	0	30
Yuşa	50	50	90	100	0	0	0	100	70

17. BUDGET

Table 6 Budget in TL

	ITEMS				
	PEOPLE	Instruments	MATERIALS*	SERVICE	TRAVEL
IMU FUND	0	8500	0	0	0
SPONSOR COMPANY FUND	0	0	0	0	0
TOTAL	0	8500	0	0	0

18. SUPPORT LETTERS

2209-A Üniversite Öğrencileri Araştırma Projeleri Destekleme Programı Başvuru Formu			
PROGRAM KODU	BAŞVURU DÖNEMİ	SON BAŞVURU TARİHİ	BAŞVURU NUMARASI
2209 A	2022 / 2	10.11.2022 17:30:00	19198012213757
Kısa Bilgi			
TC Kimlik No/Passport	4171437928		
Ad Soyad	İBRAHİM YUŞA ÇETİN		
Din	ERKEK		
Doğum Tarihi	21.06.2000		
Diyar	T.C.		
İletişim Bilgileri			
Ülke	TÜRKİYE	Şehir	İSTANBUL
İlçe	BEYOĞLU	Posta Kodu	34810
Telefon	(045) 800 70 28	E-posta	prof@im@gmail.com
Adres	Kadıköy Mh. Emirler Cad. No:19 Kavaklı Karayol. 34010 Beşiktaş / İSTANBUL		
Danışmana Ait Bilgiler			
DANİŞMANA AIT BİLGİLER			
TC Kimlik No	25414102788		
Danışman Adı	SELİM ARYOKUŞ		
Akademik Ünvanı	Görev Ünvanı	PROF. DR.	
Çalıştığı Kurum	İSTANBUL MEDİPOL Ü.		
Bütçe			
		Önerilen Bütçe	
Ulaştığı Ödenek (Uçak ve yataklı tesis hariç)	0 TL		
Hizmet Alımı Ödenek	0 TL		
Makine Teçhizat Ödenek	6000 TL		
Sart Malzemesi Ödenek	0 TL		
Genel Toplam	6000 TL		

19198012213757 - İBRAHİM YUŞA ÇETİN - 17/11/2022 00:13

1

Araştırma Projesi Bilgileri			
Proje ve Yürütücüsüne Ait Bilgiler			
Unvan	Lisans		
Sınıf	4. SINIF		
Üniversite	İSTANBUL MEDİPOL ÜNİVERSİTESİ BİLGİSAYAR MÜHENDİSLİĞİ FİL. (INGİLİZCE) (TAM BURSLU)		
Proje Başlığı	RISC-V İşlemi		
Proje Özeti	Projenin amacı RISC-V mimarisini kullanmak ve üzerine bir özel kullanıcının oluşturulmasına göre bir okuma geliştirme ve üzerine RISC-V mimarisinin araştırması amacıyla RISC-V tabanlı bir geliştirme açık kaynaklı bir RISC-V işlemci ile FPGA üzerinde bulunan VISA portu kullanılarak geliştirilecek bir uygulamayı geliştirmektir.		
Anahtar Kelimeler	donanım, işlemci, RISC-V, FPGA, Verilog		
Bilimsel ve Teknolojik Faaliyet Alanları			
Temel Bilimler + Bilgisayar Bilimleri + Donanım + İşlemci Mimarisini			
Proje Konusuyla İlgili Bilimsel/Kültürel Amaçlar (SFA) Başarıldı mı?	Evet		
Bilimsel/Kültürel Kazanma Amaçları			
Borçlu, Verlilik ve Adaylık			
Araştırma Projesi Öğrenci Bilgileri			
TC Kimlik No/Passport No	48204642000	Ad Soyad	AMER ALHAMVI
Üniversitesi	İSTANBUL MEDİPOL ÜNİVERSİTESİ BİLGİSAYAR MÜHENDİSLİĞİ FİL. (INGİLİZCE) (ÜCRETLİ)		
Unvan	Lisans	Sınıf	4. SINIF
E-posta Adresi	amer@im@gmail.com		
Ek Bilgiler			
Proje Özeti	isikim-2209.pdf		
Banka Hesap Bilgileri			
Banka	Halkbank	IBAN Numarası	TR33000120060940001019300

BU FORM BİLDİRİMLİ OLMUP, BİLİR İNSAN DESTEK PROGRAMI BAŞKANLIĞINA GÖNDERİLMESİNE GEREK YOKTUR.
Bu formun kullanılmasında yanlış olarak dolanmış bilgileri ve bilgilerin dışına çıkması beyan edilmektedir.

19198012213757 - İBRAHİM YUŞA ÇETİN - 17/11/2022 00:13

2



Istanbul Medipol University

School of Engineering and Natural Sciences

Graduation Project

CURRICULUM VITAE

<p style="text-align: center;">İbrahim Yuşa Çetin</p> <p>Email: yusha493@gmail.com Website: yusacetin.org GitHub: yusacetin</p> <p>EDUCATION</p> <p>Bachelor of Computer Engineering 2019 - 2023 Istanbul Medipol University Main courses: Data Structures, Digital Logic Design, Signals and Systems, Digital Signal Processing</p> <p>EXPERIENCE</p> <p>Part Time Digital Hardware Engineer January 2022 - Present Tübitak Bilgem Gebze</p> <ul style="list-style-type: none"> I've been working part-time at the Embedded Systems and Digital Design subdivision of Tübitak Bilgem since January 2022. I've worked with data converters and designed SPI systems to configure them. I've also done embedded programming on Xilinx MicroBlaze soft processor running on a Xilinx FPGA. <p>Teknofest Chip Design Competition October 2022 - April 2023</p> <ul style="list-style-type: none"> I participated in the Chip Design Competition organized by Teknofest and Tübitak with a new team of two people (one person other than myself). The goal of the competition was to create a System-on-a-Chip with a 32-bit RISC-V CPU with the MCX extensions and SPI, UART, and PWM peripheral modules in Verilog for the Digilent Nexys A7 board which contains a Xilinx Artix 7 FPGA. We were required to implement 11 non-standard instructions; 6 of which were related to the field of cryptography and the remaining 5 were for performing 2D convolution using equal-sized and square-shaped data and filter matrices of size up to 4 x 4. We used the open source Ibc Core (first converting it from SystemVerilog to Verilog-2005) and extended it to implement the non-standard instructions. We ranked 6th place in the competition, among the 10 finalists in our category that were selected from 34 teams that passed the preliminary design stage, who were in turn selected from among an undisclosed number of applicants. <p>Internship on Flight Simulation Integration March 2022 - September 2022 Quantum3D Remote</p> <ul style="list-style-type: none"> I created a simplistic host in C++ for the company's image generator software to be provided to the company's customers as an example and/or template for how they can integrate their own hosts to work with the company's image generator. The host and image generator communicate using CCL (CIGI Class Library) which is an industry standard open source library largely created by Boeing. <p>Teknofest International UAV Competition January 2021 - September 2021</p> <ul style="list-style-type: none"> My team also attended the International Unmanned Aerial Vehicles Competition organized by Teknofest and Tübitak in 2021. The goal of the competition was to detect a pool of water using image processing, pick up some water from the pool, and release the water on a collection point. I had two roles: creating a ground control station software, and creating the flight software for the drone; similar to my roles in the CanSat competition. I developed the ground control station with Windows Forms in C# first using Microsoft Visual Studio on Windows, and later MonoDevelop IDE on Ubuntu, due to the ease of using ROS also to synchronize the communication between the drone and the ground station. My ground control software ranked in the top 5 in the competition among the few dozen teams that attempted the optional ground control station challenge from among over a hundred total finalists. I made the flight software using ROS and MAVROS in C++. The software was responsible for controlling the drone based on the mission state and the feedback from the image processing section, as well as the commands sent from the ground control station. It ran on the NVIDIA Jetson Nano that was placed on the drone and cooperated with the PX4 Cube Orange flight controller. 	<div style="text-align: center;">  <p>Amer Alhamvi FPGA - AI - COMPUTER VISION - ROBOTICS alhamvi.amer@gmail.com +99 539 372 9779 @AmerAlhamvi AmerAlhamvi</p> </div> <p>ABOUT</p> <p>I am a computer engineer graduating this semester. My goal during my university years was to explore multiple fields of my major. I joined many projects as a student which gave me practical experience, from building autonomous vehicles to deep learning research, and now working on hardware design for graduation project.</p> <p>Keywords: Hardware design - FPGA - RISC-V - Deep Learning - Computer Vision - Robotics</p> <hr/> <p>EDUCATION</p> <p>2019 - June 2023 BSc in Computer Engineering ISTANBUL MEDİPOL UNIVERSITY</p> <hr/> <p>PROJECTS & EXPERIENCE</p> <p>2022 - 2023 RISC-V Processor on FPGA for graduation project.</p> <ul style="list-style-type: none"> Used Vivado and Quartus prim. It is my graduation project with one partner. We are using Ibc core and aiming for a national competition. Building a hardware AI accelerator and integrating special instructions for it. We are connecting to off-board RAM. Building cache and cache controller. Using RISC-V GNU toolchain to compile test codes. Using VGA port to develop a visual demo. <p>2021 - 2022 TÜBİTAK STAR (2247-C) & 1001 researcher programs</p> <p>Advisor: Prof. Hasan Fehmi Ateç Project title: Iterative Kernel Reconnection for Deep Learning Based Blind Image Super-Resolution This is a research program funded by Scientific and Technological Research Council of Türkiye (TÜBİTAK).</p> <ul style="list-style-type: none"> My task was to compare developed model with open-source counterparts. Used 12 different motion and noise degradation kernels to produce LR images. Compared PSNR and MSE values and corrected sub-pixel shift in the images. Specifically, compared with SRCNN, SRGAN and USRNet models. The work was done on PyTorch. And the results were used in the published paper. <p>2022 - 2023 Undergraduate Teaching Assistance ISTANBUL MEDİPOL UNIVERSITY</p> <ul style="list-style-type: none"> Worked for 3 courses and currently working for 2 others (until June). I have the responsibility of grading homework, labs, and quizzes. <p>2021 Teknofest Rotary wing Fighter-UAV Competition SCORED 11TH PLACE.</p> <ul style="list-style-type: none"> We built an autonomous drone that uses computer vision and AI to detect, follow, and lock-on other competitors' drone, while escaping being locked-on by them. I acted as team Co-leader. I worked on control algorithms, simulation, and computer vision portions. The drone had Nvidia Xavier NX on-board. I also worked on Gazebo simulation. 																																		
<p>Internship on Web Development August 2021 State Meteorological Institute of Türkiye Ankara</p> <ul style="list-style-type: none"> I made a few sample projects on both front-end and back-end for learning purposes. I used React.js for front-end, and Node.js and MongoDB for back-end development. <p>BROCUP Competition May 2021</p> <ul style="list-style-type: none"> In 2021, I attended the BROCUP competition in the drone category, organized by the Electrotechnology Club of Boğaziçi University, with a different team. The competition took place in the online simulation platform riders.ai due to COVID-19 restrictions. The goal of the competition was to create a simple flight software using MAVROS in Python in one weekend to guide a simulated drone through a course filled with various contours that the drone was supposed to pass through to complete the course. Frames of the contours were colored differently depending on the action to be performed after they were passed (moving forward, waiting for a set amount of time, turning right or left, increasing altitude, etc.). Team members were divided into two groups, those responsible for the controls and those responsible for image processing, and I was in the group responsible for the controls. Ranking was determined based on a grading algorithm that mainly took course completion time into account. My team won first place by a significant margin. <p>CanSat Competition November 2019 - May 2020</p> <ul style="list-style-type: none"> In 2020, I attended the CanSat competition organized by American Astronautical Society as a member of the only applying team from my school. The objective of the competition was to build a can-sized device that would be launched using a rocket propulsion mechanism provided by the organization. The device was to perform tasks such as collecting sensory data and sending it as telemetry to be picked up by the ground station software, deploying a payload at a certain altitude, deploying a gliding mechanism at a certain altitude, and deploying a parachute at a certain altitude to ensure safe landing. My roles were creating the flight software, responsible for keeping track of mission states and executing the appropriate actions at each state; and creating a ground control station software. Due to the global COVID-19 pandemic, the final stage of the competition -launching the device and carrying out the mission- was cancelled and ranking was determined based on the preliminary and critical design reports. My team ranked number 8 among 40 finalists that were selected from among an undisclosed number of applicants. <p>GRADUATION PROJECT</p> <p>Custom 32-bit RISC-V SoC Design October 2022 - Present</p> <ul style="list-style-type: none"> My graduation project is mostly the same as the project I've done for the Teknofest Chip Design Competition, and I'm also teamed with the same teammate from there. In addition to the design from the competition, we are also working on implementing a VGA module to program a visual demonstration to be shown in our final presentations. The demo will be written in a mixture of assembly and C. We haven't decided on the specifics of the demo program yet and we will begin working on it after we complete the VGA module. We will run the project on an Intel DE10 Lite board. <p>SKILLS</p> <p>Programming C, C++, Python, Bash, Java, JavaScript</p> <p>Hardware Design Verilog, VHDL, Xilinx Vivado, Intel Quartus Prime Lite</p> <p>Languages Turkish (native), English (proficient), Japanese (advanced reading, speaking needs practice), Mandarin (beginner)</p> <p>Other I am comfortable with working in a *nix environment and I typically use Arch Linux on my personal computer as a daily driver.</p>	<p>Amer Alhamvi</p> <p>2022 TÜBİTAK 2209A research project</p> <p>More precise and faster map merging using visual inertial odometry and RSSI on multi-robots.</p> <ul style="list-style-type: none"> Integrated RSSI signal to a pre-existing SLAM algorithm. C++ Part of a 4 people group. We also investigated different frame and key point matching techniques and worked on their implementation. Some of the algorithms I work on: ORB, SIFT, RANSAC, Bag of Words. <p>OTHER PROJECTS</p> <ul style="list-style-type: none"> Fixed-wing UAV - Aerodynamic and Mechanical. Autonomous UGV - Computer Vision and Navigation. Stereo Camera (Stereo Labs ZED2) obstacle avoidance. SLAM applications - ORB-SLAM, VINS-Fusion, Open-Vins, ccm_slam, Hector_SLAM. Robotics and Perception Team in Youtube Channel. Matlab Neural Network from scratch Code. <p>SKILLS</p> <table border="0"> <tr> <td>Soft skills</td> <td>Languages</td> </tr> <tr> <td>Leadership</td> <td>English - Advanced</td> </tr> <tr> <td>Team-player</td> <td>Arabic - Native</td> </tr> <tr> <td>Adaptability</td> <td>Turkish - Intermediate</td> </tr> <tr> <td>Problem solving</td> <td></td> </tr> <tr> <td>Communication</td> <td></td> </tr> <tr> <td>FPGA</td> <td>Programming Languages</td> </tr> <tr> <td>Verilog</td> <td>Python C/C++</td> </tr> <tr> <td>Quartus Prime</td> <td>PyTorch Cmake</td> </tr> <tr> <td>Vivado</td> <td>OpenCV</td> </tr> <tr> <td>RISC-V</td> <td></td> </tr> <tr> <td>Computer Skills</td> <td></td> </tr> <tr> <td>Linux</td> <td></td> </tr> <tr> <td>ROS</td> <td></td> </tr> <tr> <td>SLAM</td> <td></td> </tr> <tr> <td>Microsoft Office</td> <td></td> </tr> <tr> <td>LaTeX</td> <td></td> </tr> </table> <p>REFERENCES</p> <p>Lect. Mustafa AKTAN</p> <ul style="list-style-type: none"> Istanbul Medipol University School of Engineering and Natural Sciences mustafa.aktan@medipol.edu.tr <p>Prof. Hasan Fehmi Ateç</p> <ul style="list-style-type: none"> Özyeğin University Engineering Faculty hasan.atec@ozyegin.edu.tr 	Soft skills	Languages	Leadership	English - Advanced	Team-player	Arabic - Native	Adaptability	Turkish - Intermediate	Problem solving		Communication		FPGA	Programming Languages	Verilog	Python C/C++	Quartus Prime	PyTorch Cmake	Vivado	OpenCV	RISC-V		Computer Skills		Linux		ROS		SLAM		Microsoft Office		LaTeX	
Soft skills	Languages																																		
Leadership	English - Advanced																																		
Team-player	Arabic - Native																																		
Adaptability	Turkish - Intermediate																																		
Problem solving																																			
Communication																																			
FPGA	Programming Languages																																		
Verilog	Python C/C++																																		
Quartus Prime	PyTorch Cmake																																		
Vivado	OpenCV																																		
RISC-V																																			
Computer Skills																																			
Linux																																			
ROS																																			
SLAM																																			
Microsoft Office																																			
LaTeX																																			