simge, sembol, amblem, logo, grafik içeren bir resim

Açıklama otomatik olarak oluşturuldu

FB-CPU RTL DESING

Taha Kerem Eris, Muhammed Yasin Imre,

Ahmet Emir Dagistanli, Ozan Cagilligecit

Fenerbahçe University

Computer Engineering

Istanbul, Turkey

***Abstract*— This project is designed with von neumann architecture and supports 9 commands FB-CPU RTL design is made by state machines method. Various code snippets are written with machine language on the designed FB-CPU. Ram, controllers and storers in a simple processor have been observed to work together and execute machine-language code snippets.**

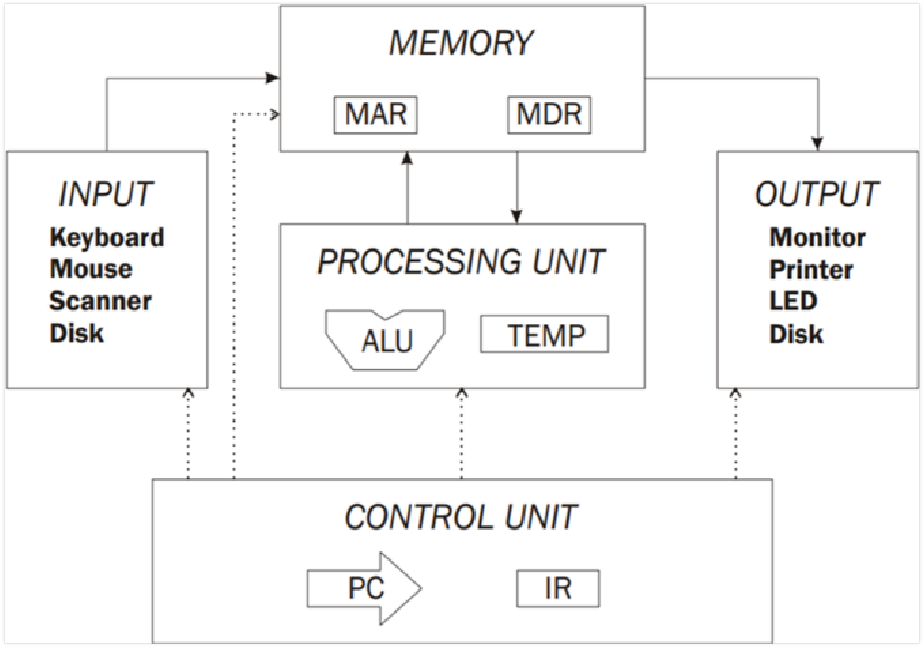
***Keywords — CPU, FPGA.***

# **I. INTRODUCTION**

The FB-CPU design is rooted in the VON-NEUMANN architecture, incorporating a finite state machine. Testing was conducted using assembly language test codes, with the utilization of control units and ALU systems.

## **II. THE SYSTEM ARCHITECTURE**

The FB-CPU has been designed using the Verilog language within the Xilinx Vivado environment. It has been tested in the simulation environment of this application. Additionally, the flow of data during the execution of instructions in the processor can be observed using a Von Neumann machine simulator. The design is implemented with Von Neumann architecture.



CPU basically has 4 elements.

* Registers
* Memory (RAM)
* Processing Unit (ALU)
* Control Unit

The FB-CPU supports 9 instructions:

* LOD ADDR: Takes the value from the specified address in memory and stores it in the ACC register.
* STO ADDR: Takes the value from the ACC register and writes it to the specified address in memory.
* ADD ADDR: Takes the value at the specified address in memory, adds it to ACC, and stores the result in ACC.
* SUB ADDR: Takes the value at the specified address in memory, subtracts it from ACC, and stores the result in ACC.
* MUL ADDR: Takes the value at the specified address in memory, multiplies it with ACC, and stores the result in ACC.
* JMP NUMBER: Sets PC (Program Counter) to the specified number.
* JMZ NUMBER: If the value in ACC is 0, sets PC to the specified number; otherwise, no operation is performed.
* NOP: No Operation; no action is taken.
* HTL: Halts the execution of the program.

metin, yazı tipi, ekran görüntüsü, beyaz içeren bir resim

Açıklama otomatik olarak oluşturulduFigure illustrates the breakdown of FB-CPU's 10-bit command for operations and addresses.

metin, daire, diyagram, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

The graph abow shows the FB-CPU's state diagram. It outlines the tasks that the processor has to complete in detail.

There are 4 registers in the code.

* State: In the state machine, it holds information about the current state.
* PC: It keeps track of the address in RAM where the current instruction is located.
* ACC: Temporary storage area.
* IR: It holds the current instruction being executed.

These registers are located in the fbcpu\_core.v file. All other registers operate based on changes in the state register. In other words, the input signals of all registers change according to the value of the state.

The FB-CPU has a Block RAM mechanism where it reads commands, calculates values, and writes them back. This memory is instantiated in the memory.v file that the memory test code uses. There are 4 registers connected to RAM, along with clock and reset signals.

**MAR:** It is a register called Memory Address Register. This register is connected to the address input of RAM. Since RAM has 2^6 locations, MAR is 6 bits. The saver is inside RAM.

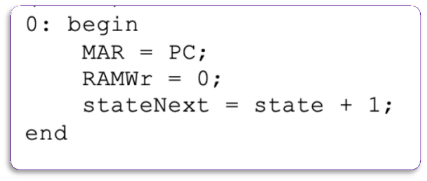
**MDRIn:** Memory Data Register In is the register used when data is written to RAM. Since one location of RAM is 10 bits, the register is 10 bits. The saver is inside RAM.

**RAMWr:** It is activated when data will be written to RAM. If it is not 1, no data is written to RAM. The saver is inside RAM.

**MDROut:** Memory Data Register is the register used when data is read from RAM. Since one location of RAM is 10 bits, the register is 10 bits. The saver is inside RAM.

* The functions of the units: memory unit (RAM) is used by the processor to store the data while the program execution. Registers can store, and manipulate data during the execution instructions and do this through flip-flop circuits. Arithmetic Logic Unit (ALU) is used for all the arithmetic and logical operations. Control Unit (CU) decodes the commands and sends the data to the ALU for decoding.

**III. THE DEVELOPED SOFTWARE**

******

***In state 0:***

MAR = PC;: The address within the Program Counter (PC) is transferred to the Memory Address Register (MAR). This sets the address for the next memory access.

No write operation was performed, so the write signal (RamWr) was set to 0, and the state incremented to 1.

stateNext = state + 1;: The state is incremented to transition to the next state. In this case, the next state facilitates the execution of the next instruction.

The Program Counter (PC) address is transferred to the Memory Address Register (MAR), no RAM write operation occurs, and the processor transitions to the next state. In this state, the processor becomes ready to read the instruction at the next memory address.

metin, yazı tipi, makbuz, beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu***In state 1:***

IRNext = MDROut;: The content of the Memory Data Register Output (MDROut), which holds the instruction read from memory, is assigned to the Instruction Register (IR). This step fetches the instruction from memory and loads it into the IR.

PCNext = PC + 1;: The Program Counter (PC) is incremented by 1, preparing it for the next instruction. This reflects the sequential execution of instructions in program memory.

stateNext = state + 1;: The state is incremented to transition to the next state. This prepares the processor for the next phase of the instruction execution cycle.

In summary, state 1 is responsible for fetching the instruction from memory, loading it into the Instruction Register (IR), incrementing the Program Counter (PC) for the next instruction, and transitioning to the next state to continue the instruction execution cycle.

***In state 2:***

***metin, ekran görüntüsü, yazı tipi, doküman, belge içeren bir resim

Açıklama otomatik olarak oluşturuldu***

If the top 4 bits of the instruction (IR[9:6]) are less than 6, the processor takes the following actions:

1. If IR[9:6] is less than 6, update the Memory Address Register (MAR) with the lower 6 bits of the instruction (IR[5:0]) and transition to state 3 for memory read or write operations.
2. If IR[9:6] is not less than 6, proceed to the next condition:
3. If IR[9:6] is equal to 6, transition to state 0 and update the Program Counter (PC) with the lower 6 bits of IR to execute the next instruction.
4. If IR[9:6] is equal to 7 and the Accumulator (ACC) is 0, update PC with the lower 6 bits of IR and transition to state 0.
5. If IR[9:6] is equal to 8, transition to state 0.
6. If IR[9:6] is equal to 9, transition to state 4 to prepare for jumping to a specific address based on a condition.

These conditions create distinct control flows based on the values of specific bits in IR, allowing the processor to execute specific commands.

***metin, yazı tipi, ekran görüntüsü, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturulduIn state 3:***

Initialize the stateNext variable to 0, marking the transition to the state where the next instruction will be fetched, completing the current instruction execution cycle. Set RAMWr to 0, indicating that no memory write operation will occur in this state. Clear the Memory Address Register (MAR) to 0, as it is not needed beyond this state.

Conditional operations based on the value of the top 4 bits (IR[9:6]) of the instruction are as follows:

1. If IR[9:6] is 0, assign the content of the Memory Data Register Output (MDROut) to the Accumulator (ACC), typically representing a load operation.
2. If IR[9:6] is 1, update the Memory Address Register (MAR) with the lower 6 bits of the instruction, set RAMWr to 1, indicating a memory write operation, and write the content of the Accumulator (ACC) to the memory address specified by MAR, typically representing a store operation.
3. If IR[9:6] is 2, update the Accumulator by adding the current value to the content of MDROut, typically representing an addition operation.
4. If IR[9:6] is 3, update the Accumulator by subtracting the content of MDROut from its current value, typically representing a subtraction operation.
5. If IR[9:6] is 4, update the Accumulator by multiplying its current value with the content of MDROut, typically representing a multiplication operation.
6. If IR[9:6] is 5, update the Accumulator by dividing its current value by the content of MDROut, typically representing a division operation. These operations collectively execute arithmetic and data transfer instructions in the processor, with the specific operation determined by the top 4 bits of the instruction.

metin, yazı tipi, ekran görüntüsü, beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu***In state 4:***

* In case 4 nothing is checked, nothing is done it just spins on.

**IV. TEST SOFTWARES**

**Test Software – 1**

**metin, makbuz, yazı tipi, beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu**

The software conducts testing with a series of initial operations, accompanied by specific command sequences.

**Test Software – 2**

**metin, makbuz, yazı tipi, beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu**

Testing software geared towards evaluating arithmetic operations, using the multiplication command in the software environment.

**Test Software - 3**

memory[0]= 10'b0000\_110011; // LOD 51, ACC = \*51, Hex = 33

memory[1]= 10'b0011\_110001; // SUB 49, ACC = ACC - \*49, Hex = F1

memory[2]= 10'b0111\_001010; // JMZ 10

memory[3]= 10'b0000\_110000; // LOD 48

memory[4]= 10'b0010\_110010; // ADD 50

memory[5]= 10'b0001\_110000; // STO 48

memory[6]= 10'b0000\_110001; // LOD 49

memory[7]= 10'b0010\_101110; // ADD 46

memory[8]= 10'b0001\_110001; // STO 49

memory[9]= 10'b0110\_000000; // JMP 0

memory[10]= 10'b0000\_110000; // LOD 48, ACC = temp, Hex = 30

memory[11]= 10'b0001\_110100; // STO 52, \*52 = ACC, Hex = 74

memory[12]= 10'b1001\_000000;// HLT

memory[46]= 10'b1; // 1 sayısı

memory[48]= 10'b0; // Hex = 0, temp

memory[49]= 10'b0; // Hex = 0

memory[50]= 10'b0000000101; // Hex = 5

memory[51]= 10'b0000001010; // Hex = A

The FB-CPU task involves performing a multiplication operation without using the MUL command but rather through loops. The process begins by storing the value at memory address 51 into the ACC register using the LOD command. Subsequently, the ACC value is subtracted by 49 and stored again in the ACC. The JMZ command is employed to check if the result (ACC-49) is equal to 0; if so, the loop ends, and the operation at address 10 is executed. In case JMZ indicates inequality to 0, the loop continues until ACC-49 becomes 0. After the check, the value at address 48 (temp) is loaded into ACC and stored in address 52 using STO, concluding the code with the HLT command. If the result is not 0, the loop persists. The process involves loading temp into ACC, adding the value at address 50, storing the result, loading and adding values from addresses 49 and 46, and storing the outcome in address 49 before looping back to line 0.

## **V. Results**

## The operations supported by the developed processor are derived from the FB-CPU ISA (Instruction Set Architecture) table. The ALU, which is the arithmetic processing unit, is where arithmetic operations are performed. The FB-CPU supports 3 arithmetic operations: addition, subtraction, and multiplication. Test software 1, test software 2, and test software 3 were individually tested and examined using the Vivado simulation. Our algorithmic thinking has improved, and our mastery of the Verilog language has increased.

## **THE PROJECT TEAM**

Taha Kerem Eris 🡪 210304032

Muhammed Yasin Imre 🡪 220304004

Ahmet Emir Dagistanli 🡪 220304501

Ozan Cagilligecit 🡪 220304015

e-mail: { [**taha.eris@stu.fbu.edu.tr**](mailto:taha.eris@stu.fbu.edu.tr),[**muhammed.imre@stu.fbu.edu.tr**](mailto:muhammed.imre@stu.fbu.edu.tr)**,** [**ahmet.dagistanli@stu.fbu.edu.tr**](mailto:ahmet.dagistanli@stu.fbu.edu.tr)**,** [**ozan.cagilligecit@stu.fbu.edu.tr**](mailto:ozan.cagilligecit@stu.fbu.edu.tr),

## **REFERENCE FILES:**

Youtube Video Link: <https://www.youtube.com/watch?v=NsL_BfjbCac>

Github Link:

<https://github.com/TahaKeremEris/FB_CPU>

## **REFERENCES:**

[1]<http://www.levent.tc/files/courses/digital_design/project/BLM201_proje_spesifikasyonlari.pdf>