LAB 5

FPGA Based Digital Design

Ben Gurion University in the Negev

Central Processing Unit (CPU)

2021 B

Authors:

Ruben Fratty	**340895499**	**rubenf@post.bgu.ac.il**
Yuval Saar	**304999261**	**yuvalsaa@post.bgu.ac.il**

Table of Contents

Assignment Goal

The goal of this lab is to plan, synthesize, and analyze a simple MIPS CPU with Mapped I/O.
To do so we were required to also understand how the memory of Cyclone II FPGA works.

System Design Definition

The CPU we designed is MIPS Single Cycle with standard MIPS register file.
The top entity is part structural and part behavioral.
The ISA commands that were implemented include:

- ADD
- ADDI
- SUB
- ADDU
- AND
- ANDI
- OR
- ORI
- XOR
- XORI
- SLL
- SRL
- LW
- SW
- LUI
- BEQ
- BNE
- SLT
- SLTI
- J
- JR
- JAL
- MUL
- MOVE (Psuedo-Instruction, implemented through other instructions)
- LA (Psuedo-Instruction, implemented through other instructions)

All the commands were implemented using the instruction format described in

Type	-31- format (bits) -0-					
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)
I	opcode (6)	rs (5)	rt (5)	immediate (16)		
J	opcode (6)	address (26)				

Table 1 : MIPS Instruction format

The design matches the block diagram described in Figure 1 and the Mapped I/O linked to the address space described in Figure 2.

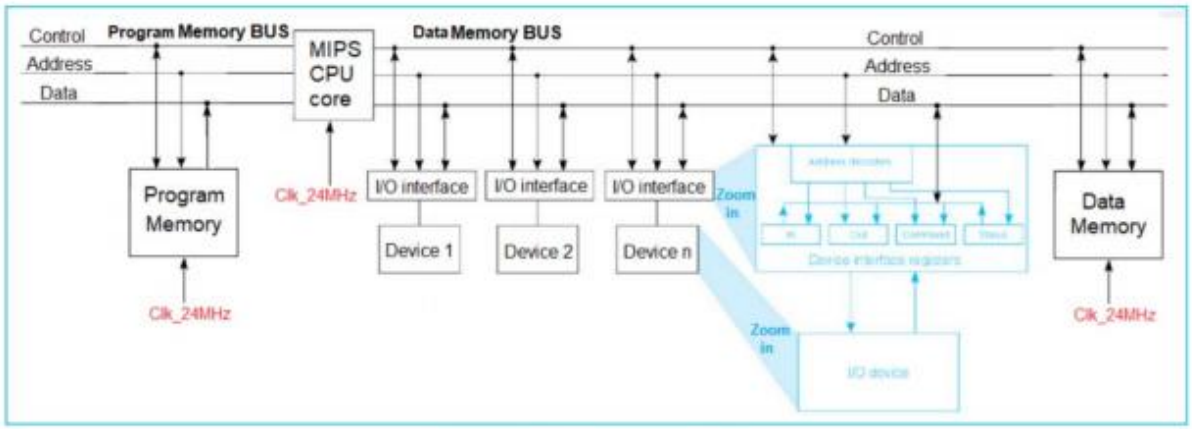


Figure 1 : System architecture

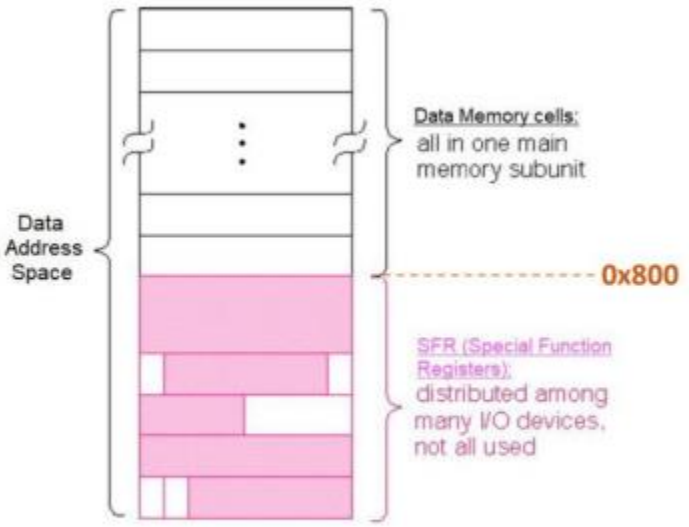


Figure 2: Address Space of a Computer Using Memory Mapped I/O

Performance Test Case

We tested the performance of the system using 5 test benches:

- We wrote Assembly code which sorts a given array of 8 integers stored in the memory, and then shows the sorted array on the LEDs and 7-segment displays.
- 4 test benches were given to us, with codes which count up/down by increments of 1/2 as well as utilize sll/srl, and display the results on the LEDs and 7-segment displays.

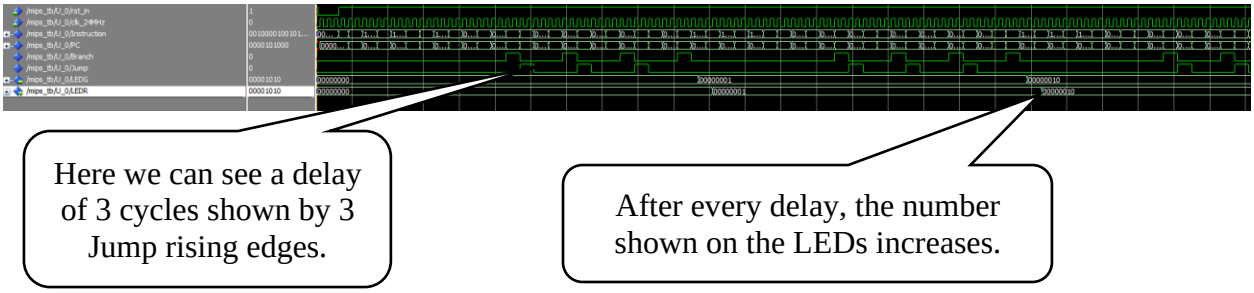
Our Testbench:

```
#-------------------- MEMORY Mapped I/O -----------------------
#define PORT_LEDG[7-0] 0x800 - LSB byte (Output Mode)
#define PORT_LEDR[7-0] 0x804 - LSB byte (Output Mode)
#define PORT_HEX0[7-0] 0x808 - LSB byte (Output Mode)
#define PORT_HEX1[7-0] 0x80C - LSB byte (Output Mode)
#define PORT_HEX2[7-0] 0x810 - LSB byte (Output Mode)
#define PORT_HEX3[7-0] 0x814 - LSB byte (Output Mode)
#define PORT_SW[7-0]   0x818 - LSB byte (Input Mode)
#--------------------------------------------------------------

.data
        ID:             .word 4,3,8,2,7,2,9,3
        sortedID:       .word 0,0,0,0,0,0,0,0
        IDsize:         .word 8
        D:              .word 0x3D0900

.text
        sw   $0,0x800 # write to PORT_LEDG[7-0]
        sw   $0,0x804 # write to PORT_LEDR[7-0]
        sw   $0,0x808 # write to PORT_HEX0[7-0]
        sw   $0,0x80C # write to PORT_HEX1[7-0]
        sw   $0,0x810 # write to PORT_HEX2[7-0]
        sw   $0,0x814 # write to PORT_HEX3[7-0]

################################################################################
### Values Initialization
################################################################################

        la      $s0,ID
        la      $s1,sortedID
        lw      $t1,IDsize                      # $t1 = Value of IDsize
        addi $s3, $zero, 4                      # $s3 = 4
        mul     $t2,$t1,$s3                     # size of the ID in memory
        add $t0,$s0,$t2                         # $t0 holds the address of the end of the ID

```

```
36  ###########################################################################
37  ### Sort ID
38  ###########################################################################
39  copy:
40          lw      $t2, 0($s0)
41          sw      $t2, 32($s0)            # copy an int to the sortedID
42          addi    $s0, $s0, 4
43          bne     $s0, $t0, copy          # if $s0 != $t0, we must keep iterating
44
45          addi    $t0,$t0,28
46
47  loop1:
48          add     $t1, $zero, $zero       # $t1 is a flag to know if list sorted (init 0)
49          la      $s0, sortedID           # $s0 is the base address of the sortedID to sort
50
51  loop2:
52          lw      $t2, 0($s0)
53          lw      $t3, 4($s0)
54          slt     $t4, $t3, $t2           # $t4 = 1 if $t3<$t2
55          beq     $t4, $zero, continue    # if $t4 = 1, then swap
56          addi    $t1, $zero, 1           # flag set to 1 means we changed the array
57          sw      $t2, 4($s0)
58          sw      $t3, 0($s0)
59
60  continue:
61          addi    $s0, $s0, 4
62          bne     $s0, $t0, loop2         # if $s0 != $t0, we must keep iterating
63          bne     $t1, $zero, loop1       # if $t4 = 1, we swapped and must go back at the beginning
64
65  ###########################################################################
66  ### Infinite loop for printing contents of array to HEX0.
67  ### $s0: address of sorted array, $t0: i
68  ###########################################################################
69  END:    la      $s0, sortedID
70          add     $t0, $zero, $zero       # $t0 = 0, i = 0
71
72  infinite_loop:
73          lw      $t1, 0x818              # If SW0 = 0 -> RESTART
74          and     $t1, $t1, 1
75          beq     $t1, $zero, END
76
77          and     $t1, $t0, 7             # $t1 = i and 0x7
78          mul     $t1, $t1, $s3           # $t1 = $t1 * 4 (offset)
79          add     $t2, $s0, $t1           # Address of the element in the sorted array
80          lw      $t3, 0($t2)             # Load value of element in the sorted array
81          sw      $t3, 0x808              # write to PORT_HEX0[7-0]
82          addi    $t0, $t0, 1
83
84  delay:
85          move    $t1,$zero       # $t1=0
86          lw      $t3, D          # $t3=0
87
88  L:      addi $t1,$t1,1          # $t1=$t1+1
89          slt  $t2,$t1,$t3    # if $t1<N than $t2=1
90          beq  $t2,$zero,infinite_loop   #if $t1>=N then go to Loop label
91          j    L
```

ModelSim Simulation Results

Shown using test0.asm – a counter that counts and shows outputs on I/Os



We have compiled our code in Quartus:

RTL Viewer Results

Main Logical Unit – MIPS processor, implemented with behavioral and structural design.

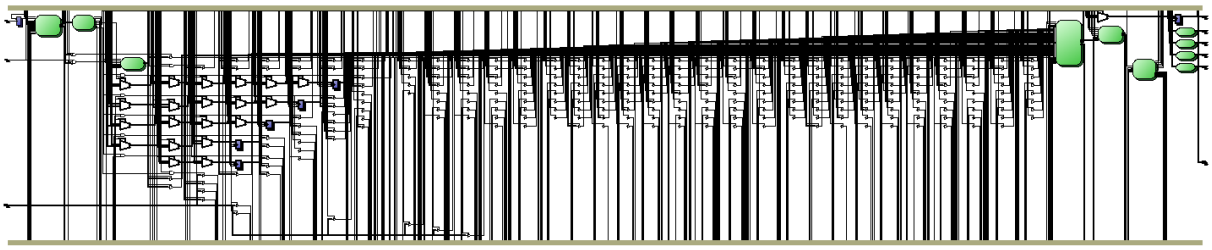
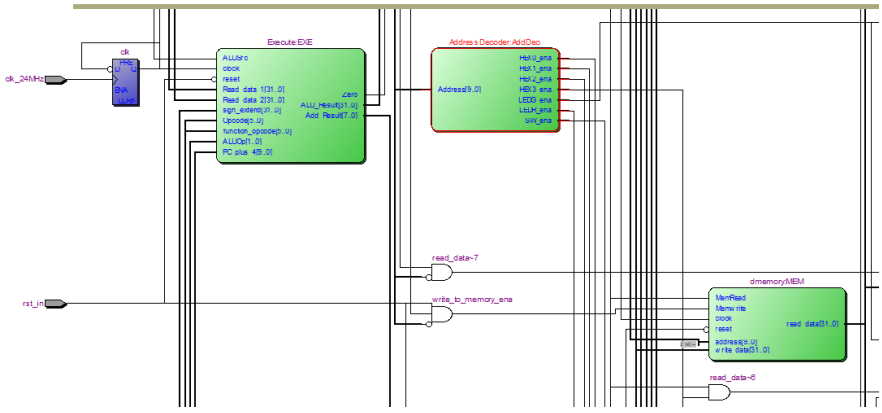


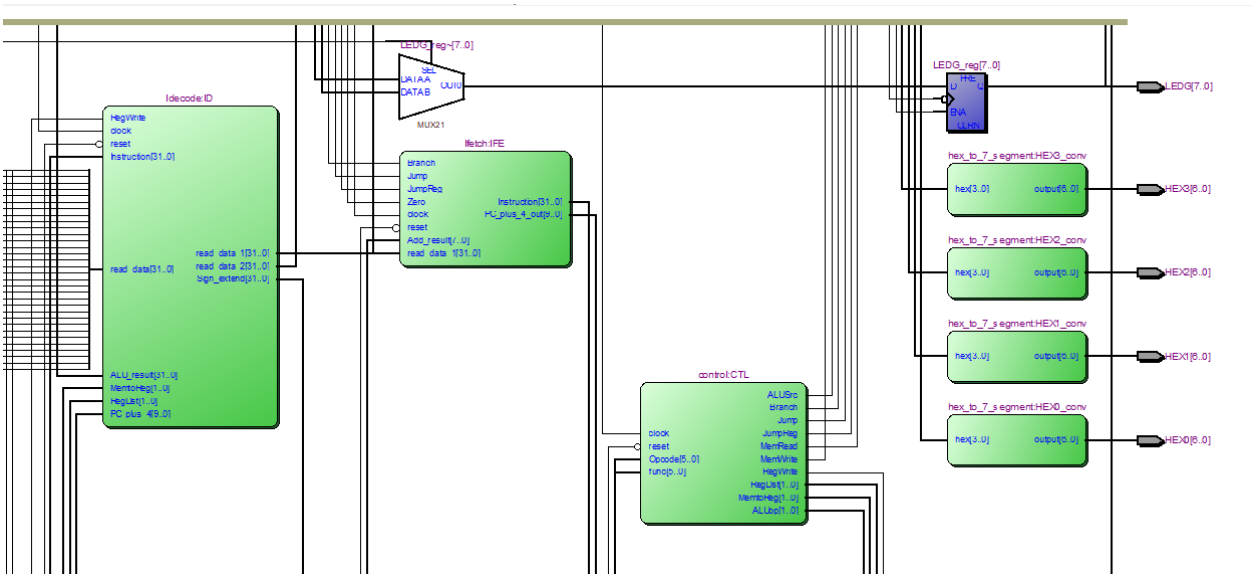
Figure 1

Close ups:

Top Left

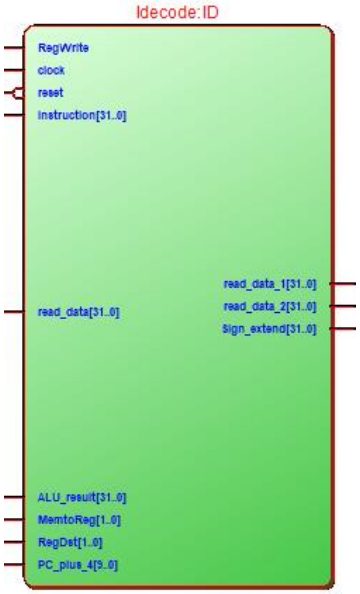


Top Right



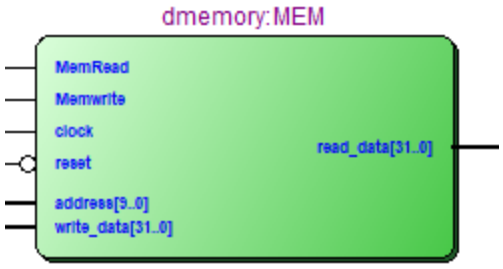
IFetch – Instruction Fetch phase of MIPS architecture

IDecode – Instruction Decode phase of MIPS architecture

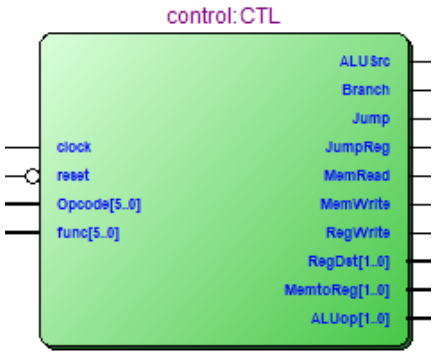


Execute – Execute phase of MIPS architecture

DMemory – Memory Read/Write phase of MIPS architecture, cannot access addresses mapped to I/O



Control – Control phase of MIPS architecture



Port Maps

MIPS			
Port	Direction	Size	Functionality
Rst_in	in	1	
clk_24Mhz	in	1	* We used 12Mhz clock.
Switches	in	8	
LEDG_out	out	8	
LEDR_out	out	8	
HEX0_out	out	7	
HEX1_out	out	7	
HEX2_out	out	7	
HEX3_out	out	7	
HEX4_out	out	7	

IFETCH			
Port	Direction	Size	Functionality
Instruction	out	32	
PC_plus_4_out	out	10	The next command address
Add_result	in	8	Address value to branch / jump to
Branch	in	1	Control signal indicates BEQ,BNQ command
Jump	in	1	Control signal indicates Jump command
JumpReg	in	1	Control signal indicates JR command
Zero	in	1	
PC_out	out	10	
clock	in	1	
reset	in	1	

IDECODE			
Port	Direction	Size	Functionality
read_data_1	out	32	Data #1 to use in execute
read_data_2	out	32	Data #2 to use in execute
Instruction	in	32	
read_data	in	32	
ALU_result	in	32	
RegWrite	in	1	
MemtoReg	in	2	
Reg_dst	in	1	
Sign_extend	out	32	Sign extended immediate
clock	in	1	
reset	in	1	
PC_plus_4_out	in	10	The next command address

CONTROL			
Port	Direction	Size	Functionality
Opcode	in	6	
Func	in	6	
RegDst	out	2	
ALUSrc	out	1	Binput selector
MemtoReg	out	2	
RegWrite	out	1	
MemRead	out	1	Read control
MemWrite	out	1	Write control
Branch	out	1	
ALUop	out	2	
Jump	out	1	
JumpReg	Out	1	
clock	in	1	
reset	in	1	

Execute			
Port	Direction	Size	Functionality
Read_data_1	in	32	Data #1 to read from
Read_data_2	in	32	Data #2 to read from
Sign_extend	in	32	Sign extended immediate
Function_opcode	in	6	For “0” opcode
opcode	in	6	Opcode number
ALUOp	in	2	With Function_opcode, opcode – decides which ALU operation will be executed
ALUSrc	in	1	
Zero	out	1	
ALU_Result	out	32	The result of the ALU (shift included)
Add_Result	out	8	
PC_plus_4	in	10	
Clock	in	1	
Reset	in	1	

DMEMORY			
Port	Direction	Size	Functionality
read_data	out	32	Memory data read
address	in	10	Address number in
write_data	in	32	Data to write
MemRead	in	1	ONLY for memory read (no I/O)
Memwrite	in	1	ONLY for memory write (no I/O)
clock	in	1	
reset	in	1	

Maximal f_{max}

The maximal f_{max} we got is 16.62 MHz:

Slow Model Fmax Summary

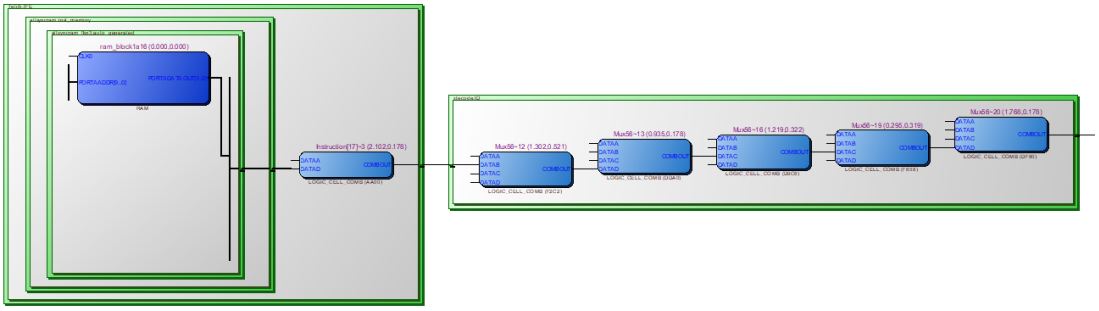
	Fmax	Restricted Fmax	Clock Name	Note
1	16.62 MHz	16.62 MHz	clk_12MHz	

Figure 2

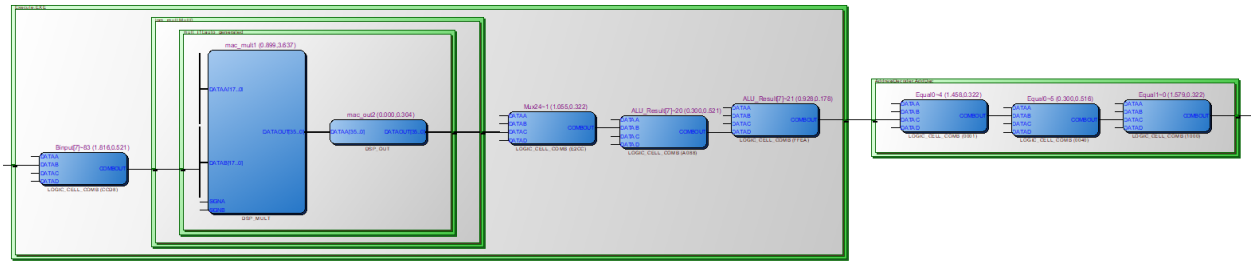
Therefore, we generated a clock of 24MHz and divided it by 2 with a simple frequency divider.

Critical Path Analysis

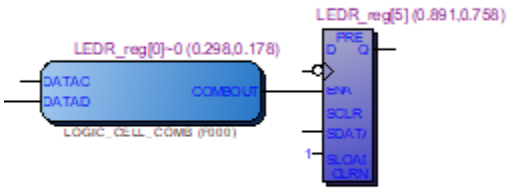
The critical path is between Ifetch and the LEDs output.
First part:

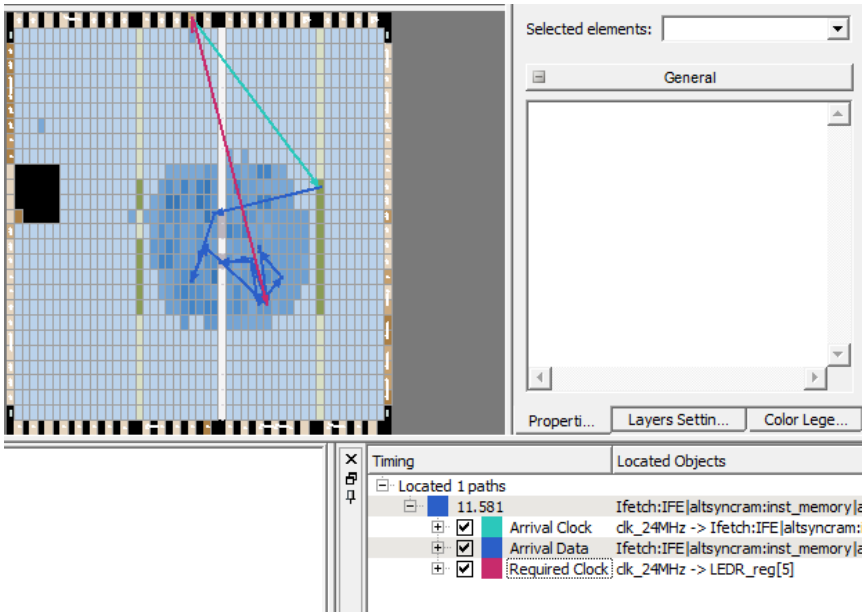


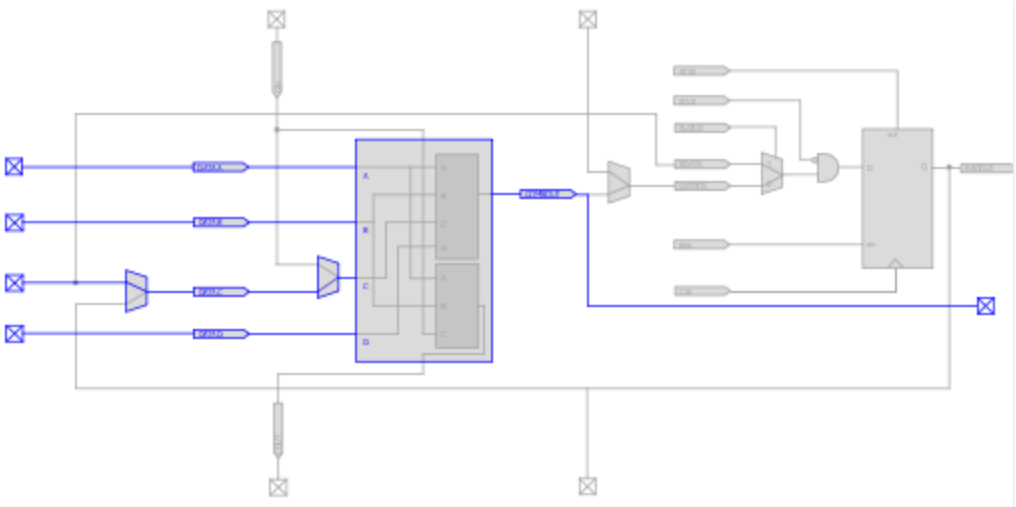
Second part:



Third part:







Shortest Path Analysis

Our guess for the shortest path is one of the bit-wise operations such as OR,XOR or AND, because these operations consist only of logic gates, and they write only to registers and not to the memory.

Flow Summary – MIPS

Flow Summary	
Flow Status	Successful - Sat Jun 26 19:27:28 2021
Quartus II 32-bit Version	12.1 Build 177 11/07/2012 SJ Web Edition
Revision Name	LAB5
Top-level Entity Name	MIPS
Family	Cyclone II
Device	EP2C20F484C7
Timing Models	Final
Total logic elements	2,489 / 18,752 (13 %)
Total combinational functions	2,330 / 18,752 (12 %)
Dedicated logic registers	1,033 / 18,752 (6 %)
Total registers	1033
Total pins	54 / 315 (17 %)
Total virtual pins	0
Total memory bits	65,536 / 239,616 (27 %)
Embedded Multiplier 9-bit elements	6 / 52 (12 %)
Total PLLs	0 / 4 (0 %)

Amount of registers and logic gates

Entity	Logic Cells	Dedicated Logic Registers	I/O Registers	Memory Bits	M4Ks	DSP Elements	DSP 9x9	DSP 18x18	Pins	Virtual Pins	LUT-Only LCs	Register-Only LCs	LUT/Register LCs
Cyclone II: EP2C20F484C7													
MIPS	6112 (82)	4404 (33)	0 (0)	194560	44	6	0	3	86	0	1708 (49)	2524 (1)	1880 (17)
AddressDecoder:AddDec	7 (7)	0 (0)	0 (0)	0	0	0	0	0	0	0	7 (7)	0 (0)	0 (0)
sld_hub:auto_hub	156 (1)	94 (0)	0 (0)	0	0	0	0	0	0	0	62 (1)	14 (0)	80 (0)
sld_signaltap:auto_signaltap_0	3434 (505)	3277 (504)	0 (0)	129024	28	0	0	0	0	0	157 (1)	2358 (252)	919 (0)
control:CTL	22 (22)	0 (0)	0 (0)	0	0	0	0	0	0	0	22 (22)	0 (0)	0 (0)
Execute:EXE	652 (349)	0 (0)	0 (0)	0	0	6	0	3	0	0	649 (346)	0 (0)	3 (3)
hex_to_7_segment:HEX0_conv	7 (7)	0 (0)	0 (0)	0	0	0	0	0	0	0	5 (5)	0 (0)	2 (2)
hex_to_7_segment:HEX1_conv	7 (7)	0 (0)	0 (0)	0	0	0	0	0	0	0	4 (4)	0 (0)	3 (3)
hex_to_7_segment:HEX2_conv	7 (7)	0 (0)	0 (0)	0	0	0	0	0	0	0	7 (7)	0 (0)	0 (0)
hex_to_7_segment:HEX3_conv	7 (7)	0 (0)	0 (0)	0	0	0	0	0	0	0	5 (5)	0 (0)	2 (2)
Idecode:ID	1678 (1678)	992 (992)	0 (0)	0	0	0	0	0	0	0	678 (678)	151 (151)	849 (849)
Ifetch:IFE	72 (72)	8 (8)	0 (0)	32768	8	0	0	0	0	0	63 (63)	0 (0)	9 (9)
dmemory:MEM	0 (0)	0 (0)	0 (0)	32768	8	0	0	0	0	0	0 (0)	0 (0)	0 (0)

Proof Of Work

Shown for the assembly code we wrote.

Optimizations we have done to the code for it to work on the FPGA include adding output signals for the HEX and implementing a frequency divider, as well as implementing 4-bit-to-7-segment converters.

Signal Tap

rst_in – System reset
Jump/JumpReg/RegWrite/MemWrite/MemRead/ALUop/RegDst/MemtoReg – Control signals
Instruction – Current instruction
ALU_Result – Current ALU result
Read_data_1/Read_data_2 – Data currently read from registers
PC – Program Counter
Read_data – Data currently read from memory
Write_data – Data currently written to memory
HEX0_reg – The register of HEX0
HEX0_ena – Enable line for HEX0

By this points the array is sorted and the numbers begin to show up on HEX0.

PC always goes to PC+4 unless a Jump or Branch is given.