



# Digital Systems Design

## Assignment 4: NIOS II

### Abstract

This study introduces the integration of various components into Qsys, including a Nios2 processor, memory units, peripherals, and a custom instruction module for leading one computation. Additionally, a PLL is incorporated to delay the SDRAM clock, and pin assignments are finalized. Memory testing is automated through C code generation. Testing reveals successful SDRAM and SRAM functionality. The subsequent phase involves utilizing Qsys for a custom module, particularly for computing leading 1s. Verification occurs through waveform simulation before integration into Qsys. Testing in Eclipse demonstrates the custom module's accuracy and efficiency compared to software execution, showing a 18% speed improvement.

### Declaration

I confirm that I have read and understood the University's definitions of plagiarism and collusion from the Code of Practice on Assessment. I confirm that I have neither committed plagiarism in the completion of this work nor have I colluded with any other party in the preparation and production of this work. The work presented here is my own and in my own words except where I have clearly indicated and acknowledged that I have quoted or used figures from published or unpublished sources (including the web). I understand the consequences of engaging in plagiarism and collusion as described in the Code of Practice on Assessment.

# Contents

1 Part A.....	1
1.1 Block diagram of BDF .....	1
1.2 Memory map .....	2
1.3 Test results for SRAM and SDRAM .....	3
1.4 Explanation of results.....	4
2 Part B .....	5
2.1 Custom instruction (Leading one).....	5
2.1.1 ASM for custom instruction .....	5
2.1.2 Verilog for custom instruction .....	5
2.1.3 Waveform simulation of custom instruction .....	6
2.2 C code for test program .....	7
2.3 Result for custom instruction (Leading 1s).....	7
2.4 Speed comparison (Custom instruction and software implementation) .....	8
2.5 Explanation of result.....	8

### 1.1 Block diagram of BDF

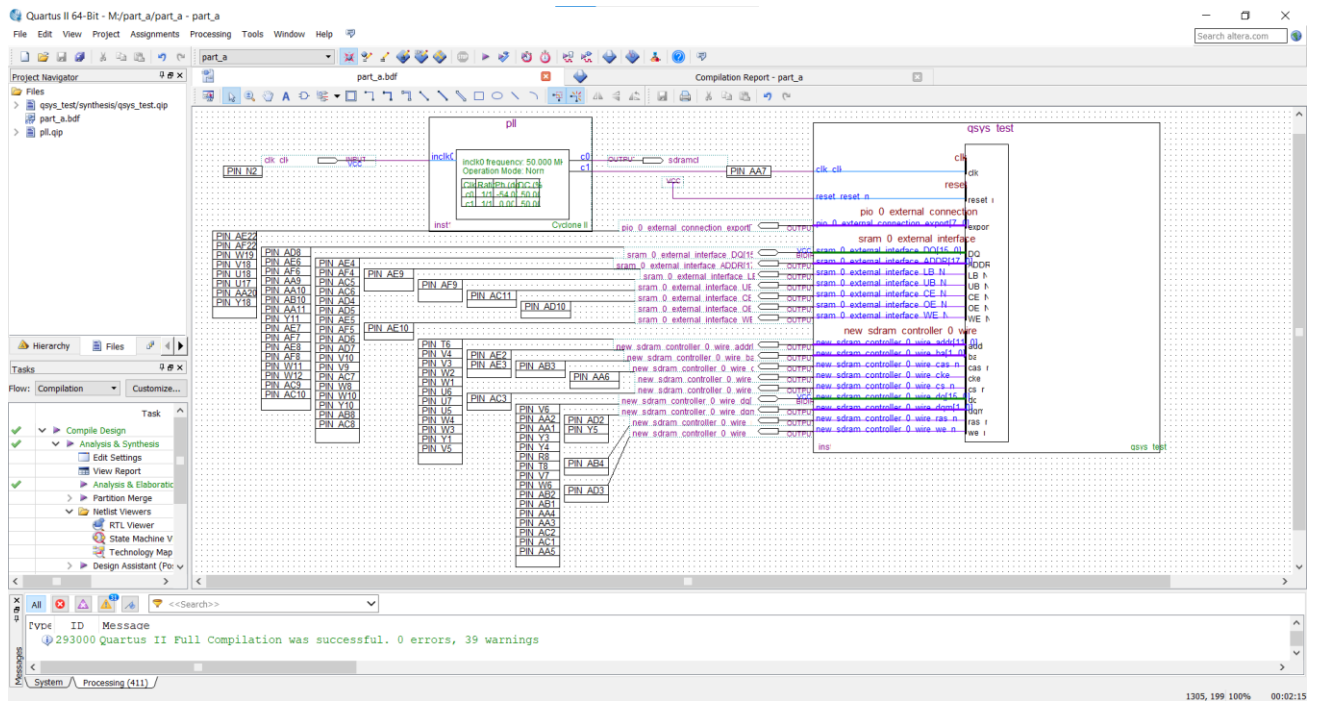


Fig.1. Screenshot of bdf file

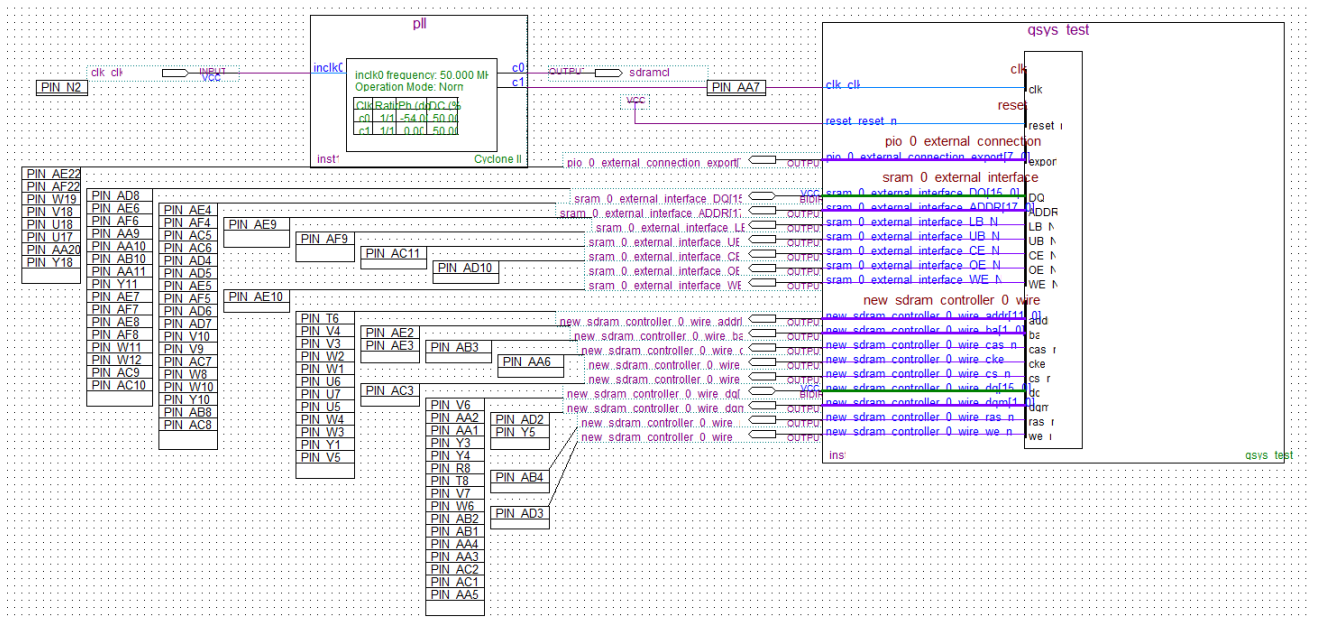


Fig.2. Block diagram for Part A

## 1.2 Memory map

Qsys - qsys\_test.qsys\* (M:\part\_1\qsys\_test.qsys)

File Edit System View Tools Help

Component Library

System Contents Address Map Clock Settings Project Settings Instance Parameters System Inspector HDL Example Generation

Use Connections

Name	Description	Export	Clock	Base	End	IRQ	Opcode Name
clk_0	Clock Source	clk	clk_0				
clk_in	Clock Input	reset					
clk_in_reset	Reset Input	Double-click to export					
clk	Clock Output	Double-click to export					
clk_reset	Reset Output	Double-click to export					
onchip_memory2_0	On-Chip Memory (RAM or ROM)	Double-click to export	clk_0	0x0110_8000	0x0110_cfff		
clk1	Clock Input	Double-click to export	[clk1]				
s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]				
reset1	Reset Input	Double-click to export					
nios2_qsys_0	Nios II Processor	Double-click to export	clk_0			IRQ 0	IRQ 31
clk	Clock Input	Double-click to export	[clk]				
reset_n	Reset Input	Double-click to export	[clk]				
data_master	Avalon Memory Mapped Master	Double-click to export	[clk]				
instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]				
jtag_debug_module_re...	Reset Output	Double-click to export	[clk]	0x0111_0800	0x0111_0fff		
jtag_debug_module	Avalon Memory Mapped Slave	Double-click to export	[clk]				
custom_instruction_m...	Custom Instruction Master	Double-click to export					
jtag_uart_0	JTAG UART	Double-click to export	clk_0				
clk	Clock Input	Double-click to export	[clk]				
reset	Reset Input	Double-click to export	[clk]				
avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0111_1030	0x0111_1037		
timer_0	Interval Timer	Double-click to export	clk_0				
clk	Clock Input	Double-click to export	[clk]				
reset	Reset Input	Double-click to export	[clk]				
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0111_1000	0x0111_101f		
sysid_qsys_0	System ID Peripheral	Double-click to export	clk_0				
clk	Clock Input	Double-click to export	[clk]				
reset	Reset Input	Double-click to export	[clk]				
control_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0111_1038	0x0111_103f		
pio_0	PIO (Parallel I/O)	Double-click to export	clk_0				
clk	Clock Input	Double-click to export	[clk]				
reset	Reset Input	Double-click to export	[clk]				
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0111_1020	0x0111_102f		
external_connection	Conduit	pio_0_external_connection					
sram_0	SRAM/SSRAM Controller	Double-click to export	clk_0				
clock_reset	Clock Input	Double-click to export	[clock_reset]				
clock_reset_reset	Reset Input	Double-click to export					
external_interface	Conduit	sram_0_external_interface					
avalon_sram_slave	Avalon Memory Mapped Slave	Double-click to export	[clock_reset]	0x0108_0000	0x010f_ffff		
new_sdram_controll...	SDRAM Controller	Double-click to export	clk_0				
clk	Clock Input	Double-click to export	[clk]				
reset	Reset Input	Double-click to export	[clk]				
s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0080_0000	0x00ff_ffff		
wire	Conduit	new_sdram_controller_0...					
new_component_0	new_component	Double-click to export	Opcode 0				
nios_custom_instruct...	Custom Instruction Slave	Double-click to export					new_component_0

Messages

2 Info Messages

System ID is not assigned automatically. Edit the System ID parameter to provide a unique ID

System.sysid\_qsys\_0

0 Errors, 0 Warnings

Fig.3. Screenshot for Qsys

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Opcode Name
<input checked="" type="checkbox"/>		clk_0	Clock Source	clk	clk_0				
<input checked="" type="checkbox"/>		clk_in	Clock Input	reset					
<input checked="" type="checkbox"/>		clk_in_reset	Reset Input	Double-click to export					
<input checked="" type="checkbox"/>		clk	Clock Output	Double-click to export					
<input checked="" type="checkbox"/>		clk_reset	Reset Output	Double-click to export					
<input checked="" type="checkbox"/>		onchip_memory2_0	On-Chip Memory (RAM or ROM)	Double-click to export	clk_0	0x0110_8000	0x0110_cfff		
<input checked="" type="checkbox"/>		clk1	Clock Input	Double-click to export	[clk1]				
<input checked="" type="checkbox"/>		s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]				
<input checked="" type="checkbox"/>		reset1	Reset Input	Double-click to export					
<input checked="" type="checkbox"/>		nios2_qsys_0	Nios II Processor	Double-click to export	clk_0			IRQ 0	IRQ 31
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to export	[clk]				
<input checked="" type="checkbox"/>		reset_n	Reset Input	Double-click to export	[clk]				
<input checked="" type="checkbox"/>		data_master	Avalon Memory Mapped Master	Double-click to export	[clk]				
<input checked="" type="checkbox"/>		instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]				
<input checked="" type="checkbox"/>		jtag_debug_module_re...	Reset Output	Double-click to export	[clk]	0x0111_0800	0x0111_0fff		
<input checked="" type="checkbox"/>		jtag_debug_module	Avalon Memory Mapped Slave	Double-click to export	[clk]				
<input checked="" type="checkbox"/>		custom_instruction_m...	Custom Instruction Master	Double-click to export					
<input checked="" type="checkbox"/>		jtag_uart_0	JTAG UART	Double-click to export	clk_0				
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to export	[clk]				
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to export	[clk]				
<input checked="" type="checkbox"/>		avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0111_1030	0x0111_1037		
<input checked="" type="checkbox"/>		timer_0	Interval Timer	Double-click to export	clk_0				
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to export	[clk]				
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to export	[clk]				
<input checked="" type="checkbox"/>		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0111_1000	0x0111_101f		
<input checked="" type="checkbox"/>		sysid_qsys_0	System ID Peripheral	Double-click to export	clk_0				
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to export	[clk]				
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to export	[clk]				
<input checked="" type="checkbox"/>		control_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0111_1038	0x0111_103f		
<input checked="" type="checkbox"/>		pio_0	PIO (Parallel I/O)	Double-click to export	clk_0				
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to export	[clk]				
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to export	[clk]				
<input checked="" type="checkbox"/>		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0111_1020	0x0111_102f		
<input checked="" type="checkbox"/>		external_connection	Conduit	pio_0_external_connection					
<input checked="" type="checkbox"/>		sram_0	SRAM/SSRAM Controller	Double-click to export	clk_0				
<input checked="" type="checkbox"/>		clock_reset	Clock Input	Double-click to export	[clock_reset]				
<input checked="" type="checkbox"/>		clock_reset_reset	Reset Input	Double-click to export					
<input checked="" type="checkbox"/>		external_interface	Conduit	sram_0_external_interface					
<input checked="" type="checkbox"/>		avalon_sram_slave	Avalon Memory Mapped Slave	Double-click to export	[clock_reset]	0x0108_0000	0x010f_ffff		
<input checked="" type="checkbox"/>		new_sdram_controll...	SDRAM Controller	Double-click to export	clk_0				
<input checked="" type="checkbox"/>		clk	Clock Input	Double-click to export	[clk]				
<input checked="" type="checkbox"/>		reset	Reset Input	Double-click to export	[clk]				
<input checked="" type="checkbox"/>		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0080_0000	0x00ff_ffff		
<input checked="" type="checkbox"/>		wire	Conduit	new_sdram_controller_0...					
<input checked="" type="checkbox"/>		new_component_0	new_component	Double-click to export	Opcode 0				
<input checked="" type="checkbox"/>		nios_custom_instruct...	Custom Instruction Slave	Double-click to export					new_component_0

Fig.4. Qsys with assigned memory map

System Contents	Address Map	Clock Settings	Project Settings	Instance Parameters	System Inspector	HDL Example	Generation
	nios2_qsys_0.data_master				nios2_qsys_0.instruction_master		
onchip_memory2_0.s1	0x0110_8000 - 0x0110_cfff				0x0110_8000 - 0x0110_cfff		
nios2_qsys_0.jtag_debug_module	0x0111_0800 - 0x0111_0fff				0x0111_0800 - 0x0111_0fff		
jtag_uart_0.avalon_jtag_slave	0x0111_1030 - 0x0111_1037						
timer_0.s1	0x0111_1000 - 0x0111_101f						
sysid_qsys_0.control_slave	0x0111_1038 - 0x0111_103f						
pio_0.s1	0x0111_1020 - 0x0111_102f						
sram_0.avalon_sram_slave	0x0108_0000 - 0x010f_ffff				0x0108_0000 - 0x010f_ffff		
new_sdram_controller_0.s1	0x0080_0000 - 0x00ff_ffff				0x0080_0000 - 0x00ff_ffff		

Fig.5. Address Map

### 1.3 Test results for SRAM and SDRAM

```

memtest_small.c
Problems Tasks Console Properties Nios II Console
mem-test Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 1

<----> Nios II Memory Test. <---->

This software example tests the memory in your system to assure it
is working properly. This test is destructive to the contents of
the memory it tests. Assure the memory being tested does not contain
the executable or data sections of this code or the exception address
of the system.

201677461, Jinsong Zhang

Press enter to continue or 'q' to quit.

Base address to start memory test: (i.e. 0x8000000)
>
0x00800000
0x00800000
End Address:
>
0x00ffffff
0x00ffffff

Testing RAM from 0x800000 to 0xffffffff
-Data bus test passed
-Address bus test passed
-Byte and half-word access test passed
-Testing each bit in memory device. . . passed
Memory at 0x800000 Okay

201677461, Jinsong Zhang

Press enter to continue or 'q' to quit.

Base address to start memory test: (i.e. 0x8000000)
>
0x01080000
0x01080000
End Address:
>
0x010ffffff
0x010ffffff

Testing RAM from 0x1080000 to 0x10ffffff
-Data bus test passed
-Address bus test passed
-Byte and half-word access test passed
-Testing each bit in memory device. . . passed
Memory at 0x1080000 Okay

```

Fig.6. Test result for SRAM and SDRAM

## 1.4 Explanation of results

```
int main(void)
{
    int ch;

    /* Print the Header */
    MenuHeader();

    while (1)
    {
        printf("\n201677461, Jinsong Zhang\n");
        printf("\nPress enter to continue or 'q' to quit.\n");
        ch = alt_getchar();
        putchar(ch);
        if(ch == 'q' || ch == 'Q')
        {
            printf( "\nExiting from Memory Test.\n");
            break;
        }
        else if (ch == '\n')
        {
            TestRam();
        }
    }
    return (0);
}
```

Fig.7. Code for memory test

We started by adding to Qsys such things as a Nios2 processor, on-chip memory, a JTAG UART, an Interval Timer, a System ID Peripheral, a PIO, SRAM, SDRAM, and a custom instruction module to compute the leading one. Then a PLL is added to delay the SDRAM clock by 3ns and finally all the pin assignments for this module are done. When we create a new C code project for memory testing, it will automatically generate a code for testing memory. We just need to add our student number and name in the main function to be displayed in each test. In the test results section, I first tested the SDRAM. the test was completed by entering the start and end addresses of the SDRAM memory into the NIOS console. The results show that all the tests passed. Then I entered the start and end addresses of the SRAM into the console and all tests passed.

## 2 Part B

### 2.1 Custom instruction (Leading one)

#### 2.1.1 ASM for custom instruction

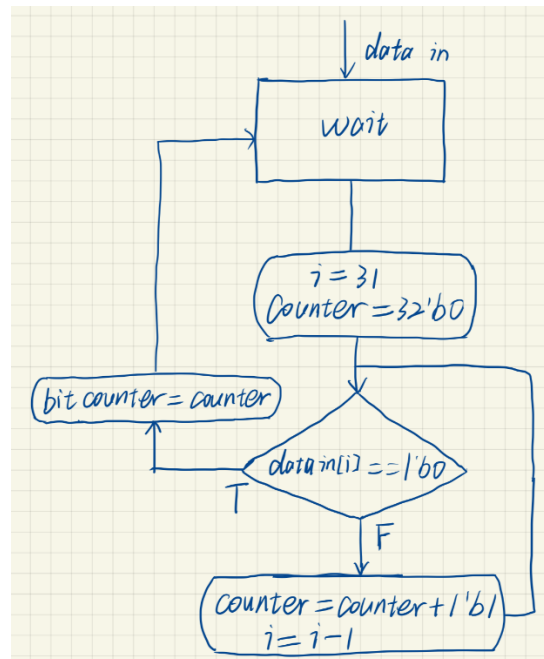


Fig.8. ASM for custom instruction

#### 2.1.2 Verilog for custom instruction

```
1 module lead_one (
2     input wire [31:0] data_in,
3     output reg [31:0] bit_count
4 );
5     reg [31:0] counter;
6
7     integer i;
8
9     always @ (data_in)
10     begin
11         begin:flag
12             counter = 32'b0;
13             for (i=31; i>=0; i=i-1)
14                 begin
15                     if (data_in [i] ==1'b0)
16                         disable flag;
17                     counter = counter + 1'b1;
18                 end
19             end
20             bit_count = counter;
21         end
22     endmodule
23
```

Fig.9. Verilog for custom instruction

### 2.1.3 Waveform simulation of custom instruction

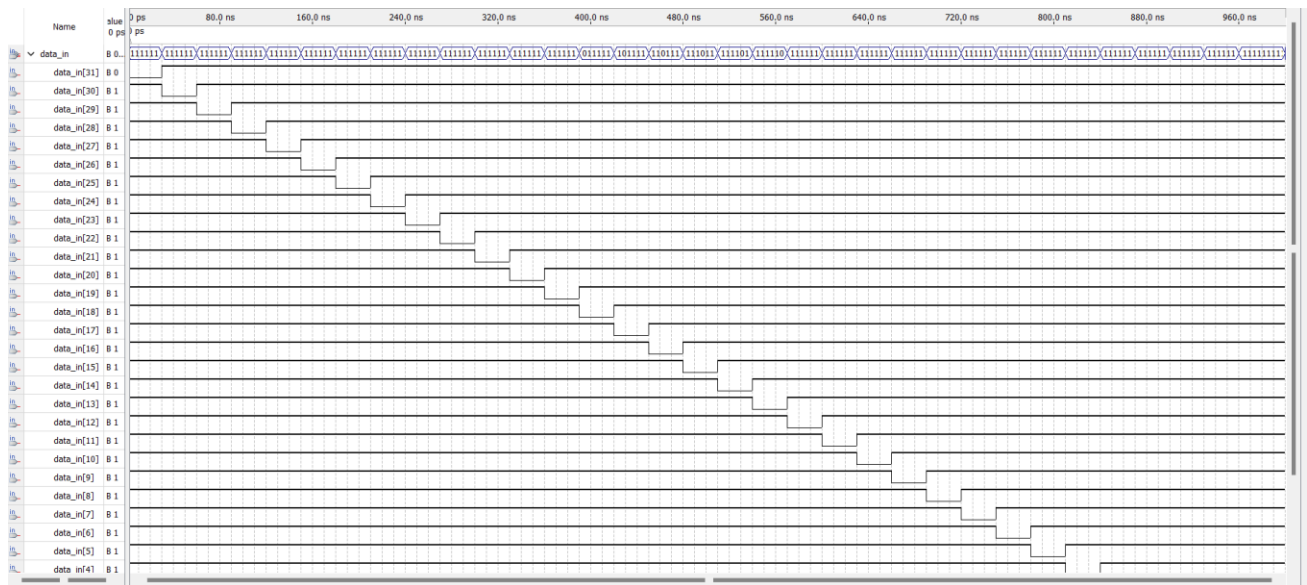


Fig.10. Simulation of custom instruction (Leading one)

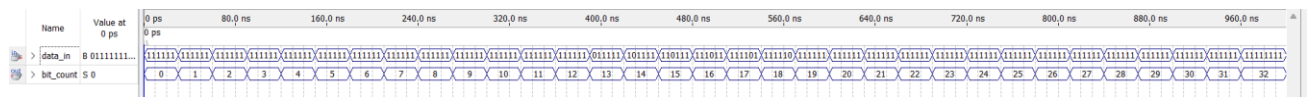


Fig.11. Results of Simulation

Here we tested all the cases of the custom instruction, because my task was to test leading 1s. this means that I need to count how many 1s there are in the highest bit of a 32-bit binary number (until a 0 is encountered or the count reaches 32 bits). I tested all possibilities, i.e. from 0 1s to 32 1s. as you can see from the test results everything is fine.



## 2.2 C code for test program

```
#include "sys/alt_stdio.h"
#include "system.h"
#include "time.h"
#include "sys/alt_timestamp.h"
int main()
{
    int i = 0xF0000000; // 4 leading 1s
    int j = 0xFF000000; // 8 leading 1s
    int k = 0xFFFF0000; // 12 (c in hex) leading 1s
    int time1, time2, time3, time4;
    int counter = 0;
    int msb = 1 << 31;
    alt_putstr("Hello from Nios II!\n");

    // custom instruction
    time1 = alt_timestamp_start();
    counter = ALT_CI_NEW_COMPONENT_0(i,0); // custom instruction to count leading 1s
    time2 = alt_timestamp();

    alt_printf("i = %x, a = %x\n", i, counter);
    alt_printf("Time for custom instruction: %x ticks\n", time2 - time1); //custom instruction time

    //software implementation
    time3 = alt_timestamp_start();
    for (counter = 0; counter < 32; counter++) // software to count leading 1s
    {
        if(((i << counter) & msb) == 0)
        {
            break;
        }
    }
    time4 = alt_timestamp();

    alt_printf("i = %x, a = %x\n", i, counter);
    alt_printf("Time for software implementation: %x ticks\n", time4 - time3); // software implementation time

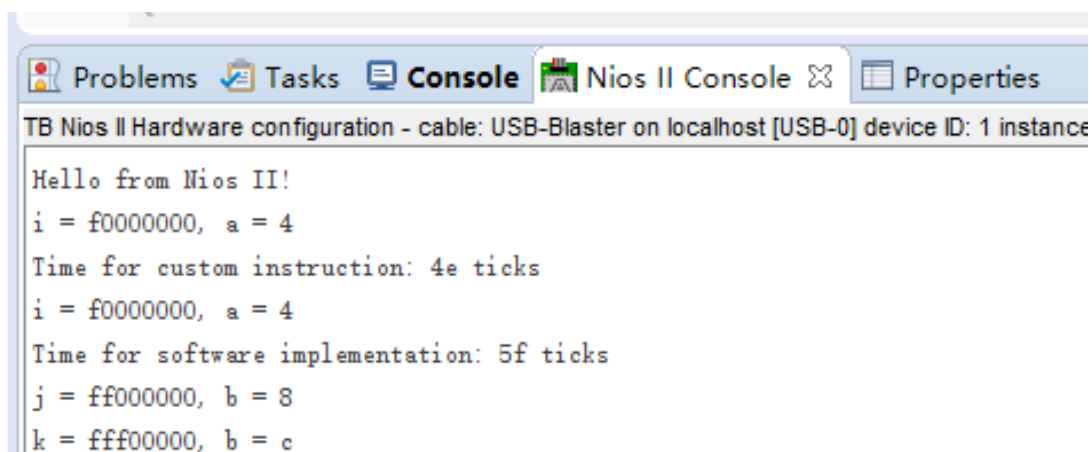
    counter = ALT_CI_NEW_COMPONENT_0(j,0); // custom instruction to count leading 1s
    alt_printf("j = %x, b = %x\n", j, counter);

    counter = ALT_CI_NEW_COMPONENT_0(k,0); // custom instruction to count leading 1s
    alt_printf("k = %x, b = %x\n", k, counter);

    return 0;
}
```

Fig.12. C code for custom instruction testing (Leading 1s)

## 2.3 Result for custom instruction (Leading 1s)



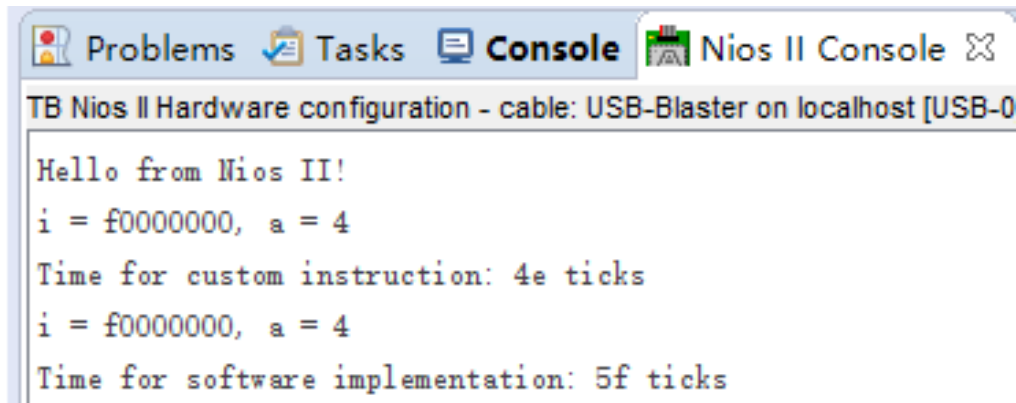
The screenshot shows the Nios II Console window with the following output:

```
TB Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance

Hello from Nios II!
i = f0000000, a = 4
Time for custom instruction: 4e ticks
i = f0000000, a = 4
Time for software implementation: 5f ticks
j = ff000000, b = 8
k = fff00000, b = c
```

Fig.13. Test result for custom instruction implementation and software implementation

## 2.4 Speed comparison (Custom instruction and software implementation)



```
Problems Tasks Console Nios II Console
TB Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0]

Hello from Nios II!
i = f0000000, a = 4
Time for custom instruction: 4e ticks
i = f0000000, a = 4
Time for software implementation: 5f ticks
```

Fig.14. Speed comparison between custom implementation and software implementation

## 2.5 Explanation of result

For the second part we can just use Qsys from the first part. note that we need to add a .v file as a new custom command. The purpose of this directive is to compute a binary number of leading 1s. We have tested our custom module in its entirety in the waveform simulation section. We then added this module to Qsys. Then we create a new C project in Eclipse, the type should be Hello world small. then we can find the functions and the usage of our custom module from the header file of system.h. Then we write C code to test if the custom module is working properly. The test result shows that everything works fine with our custom command module. That is, it calculates leading 1s correctly, and then we use a for loop as a software execution and compare it with our custom instruction. Here we use timestamp to calculate the execution time. We can see that the custom instruction uses 0x4e ticks to calculate 4 leading 1s, while the software execution uses 0x5f ticks. we can conclude that the custom instruction is faster than the software execution.  $0x5f = 95$ ,  $0x4e = 78$ . we can see that the custom instruction is 18% faster than the software execution.