Computer Architecture - Lab Assignment 5

Nios V processor with customized architecture for a software application

Processors can be optimized for any number of reasons, including combinations of throughput, latency, and power. The goal of processor customization is to change an architecture to benefit a small set of applications, while maintaining the flexibility to run many other applications.

The objectives of this lab assignment are as follows.

- Main goal. Discover and explain the key mechanisms by which soft Nios V processors with a specialized instruction set reduce the execution time of determined programs.
- Goals for the hardware engineering:
  - Student should understand the Verilog hardware description language contained in the `*.v` files: `CRC_Custom_Instruction.v`, `CRC_Component.v`. These files are used to generate the customized hardware module that is directly connected to the arithmetic & logic functional unit (ALU) of the Nios V/g processor.
  - The circuit schematics that are equivalent to these Verilog files should be generated and explained.
- Goals for the computer architecture:
  - How the custom function unit is integrated into the data path of the Nios V/g processor (Intel [2023a,b]).
  - Explain the performance improvement that is achieved when the custom instructions are used. Hint: $t_{CPU} = N \times CPI/f$ (N: number of executed instructions, CPI: cycles per instruction, f:clock speed).
- Goals for the software engineering:
  1. CRC-32 algorithm should be analyzed: operations, data structures, data hazards.
  2. Software profiling is used to discover the most costly operations (see `crc_main.c` and `crc.c` files)
  3. The source codes must be compiled and linked. Then, the executable file must be run using the DE0-Nano board. The execution times of three types of programs are measured: slow, fast and customized software versions of the CRC-32 algorithm.
  4. The performance of the Nios V/g software processor is evaluated using the measurements of execution times. Performance must be evaluated using CPI, t_{CPU}, and speed-up.

The files needed for developing the tasks described in this lab assignment are the followings:

```
SoC_configurations/
-  CA_CI_CRC_NVg.sof
-  CA_CI_CRC_NVg.sopcinfo
code/
-  verilog/
+    CRC_Component.v
+    CRC_Custom_Instruction.v
-  C/
```

```
+    ci_crc.c
+    ci_crc.h
+    crc.c
+    crc.h
+    crc_main.c
```

Part I. CRC-32 Algorithm

The aim of an error detection technique is to enable the receiver of a message transmitted through a noisy (error-introducing) channel to determine whether the message has been corrupted (Williams [2022]). To do this, the transmitter constructs a value (called a checksum) that is a function of the message, and appends it to the message. The receiver can then use the same function to calculate the checksum of the received message and compare it with the appended checksum to see if the message was correctly received. For example, if we chose a checksum function which was simply the sum of the bytes in the message mod 256 (i.e. modulo 256), then it might go something as follows. All numbers are in decimal.

- Message : 6 23 4
- Message with checksum : 6 23 4 33
- Message after transmission : 6 27 4 33

CRC-32 is a checksum/hashing algorithm that is very commonly used in kernels and for Internet checksums (OSDev [2022]). A message composed of multiple bytes is passed to CRC-32 algorithm. It returns the same message that was introduced with a checksum at the end. In order to achieve a better understanding of the CRC-32, the following paragraphs presents a brief explanation of the slow, simple version of the algorithm. The algorithm uses the modulo-2 division and starts with a 32-bit (4 bytes) checksum with all bits set (`0xFFFFFFFF`).

Algorithm 1 Pseudocode for the CRC-32 algorithm.

```
1: for Each byte of the message introduced, repeat the loop until the last byte of the message do
2:     The byte is taken and all its bits are bit-reflected.
3:     The byte is shifted to the upper 8 bits of the current 32-bit chekcsum.
4:     Exclusive-OR operation is executed: checksum = checksum XOR shifted byte.
5:     for Each one of the 8 bits of the byte do
6:         if The sign bit of the checksum is set then
7:             The checksum is shifted up one bit.
8:             XOR operation with the magic value 0x04C11DB7.
9:         else
10:            Checksum is shifted up one bit.
11:        end if
12:    end for
13: end for
```

When the loop finishes, the entire checksum is bit-reflected. This is the final CRC-32 value. Figures 1 and 2 show an execution example of Algorithm 1 for the slow and custom versions.

In this lab assignment, three versions of the CRC-32 algorithm will be studied. One is called *slow version* that is implemented using a modulo-2 division and executed in a Nios V/g soft processor. The second one is called *fast version* that is implemented using a lookup table and also executed with Nios V/g processor. Lastly, the called *custom instruction version* uses a custom instruction created by the hardware designer. This new instruction is executed in a specialized Nios V/g processor that integrates a custom logic module connected to its Arithmetic & Logic Unit (ALU).

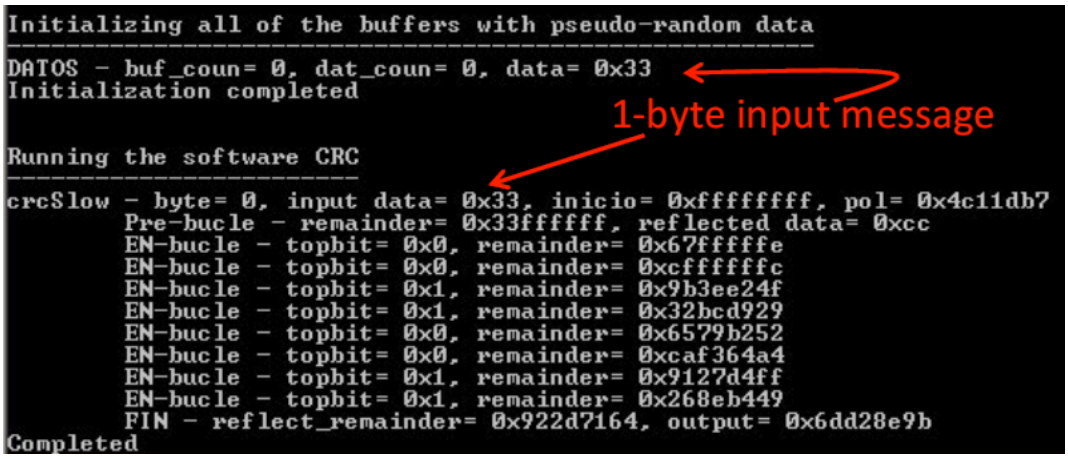

Figure 1: Nios V Command Shell terminal where the results of the slow version of the CRC algorithm is shown. The value of the input message is `0x0000 0033`. The 32-bit result of the CRC-32 algorithm applied to the input message is `0x6DD2 8E9B`.

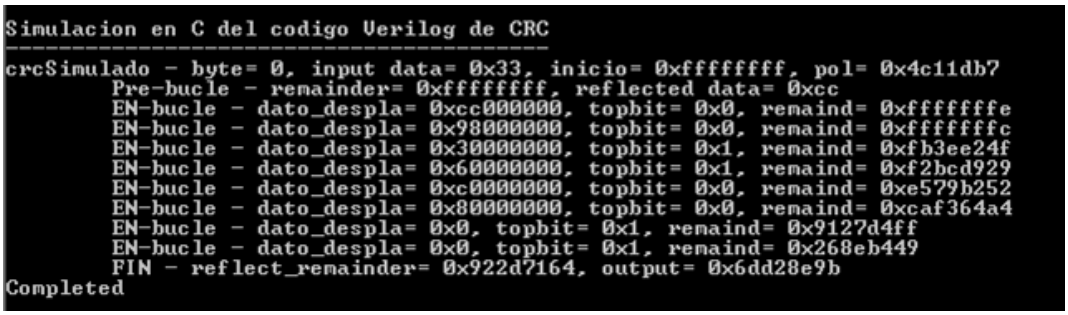

Figure 2: Nios V Command Shell terminal where the results of the version of the CRC algorithm using the Nios V/g custom instruction is shown. As can be seen, the input and output messages are the sale as obtained for the slow versión (see Figure 1).

Part II. Hardware Engineering

The scope of hardware engineering includes the hardware design of computers and their components using a high-level language. We will use the hardware description language called Verilog (Nyasulu and Knight [2003]). This section describes the internal organization of the CRC-32 hardware module.

First of all, the Verilog source code of the custom hardware module is contained in `CRC_Component.v` and `CRC_Custom_Instruction.v` files. The schematic diagram of this hardware module is shown in Figure 3. Figure 4 shows the microarchitecture of the System-on-Chip where the Nios V/g soft processor with customized ALU processing engine is integrated.

Question 1.
Explain and justify the organization and functionality of the CRC-32 hardware component that is coded in `CRC_Component.v` file using Verilog language. Draw schematic diagrams for each one of the submodules.

Part III. Computer Architecture

This section describes the hardware-software interface of Nios V/g custom instructions. Custom instructions allow the programmer to implement an algorithm or a code block in a custom hardware logic block. In the

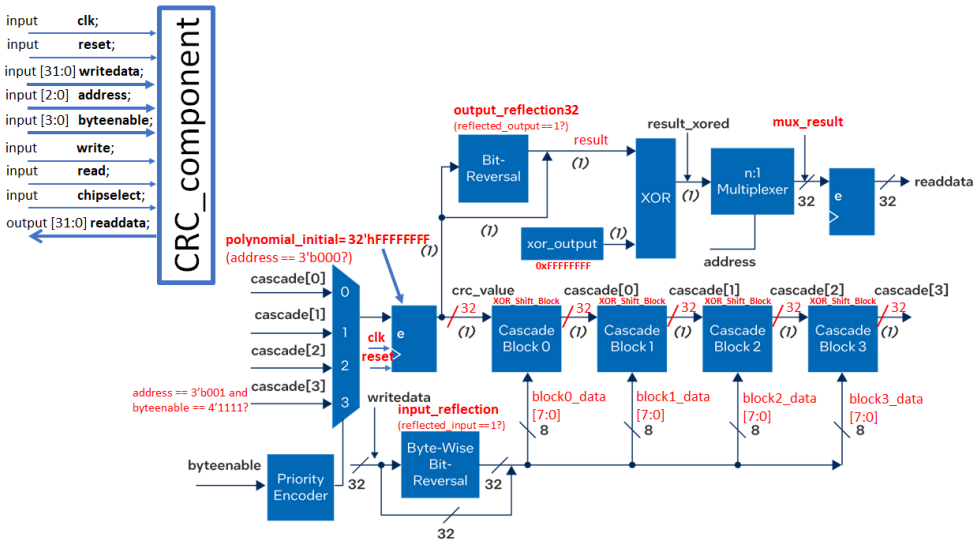

Figure 3: Schematic diagram of the CRC-32 hardware component (Intel [2022]).

Nios V/g processor, the custom instruction's logic circuits connect directly to the Arithmetic & Logic Unit (ALU) in the data path (Intel [2023a]), as shown in Figure 5.

Each custom operation is assigned a single selector index. This index allows software to specify the desired operation. It is determined at the time the hardware is designed using with Platform Designer module of Quartus Prime design suite. Platform Designer exports the selection index value to the `system.h` header file to be used by the Nios V software toolchain. For each custom instruction, the Nios V toolchain generates a macro C function in the `system.h` file. A C program for a software application employs this macro functions to call the custom instructions. Figure 6 shows an example of C macro procedure for eight CRC custom instructions. A specialized Nios V/g processor can have a maximum of 4096 custom instructions connected to the ALU hardware module.

The examples shown in Figures 7 and 9 are two examples of application code that use the CRC custom instruction and includes the `system.h` file.

After compiling and linking the C source code that uses the CRC custom instructions, an assembler instruction with RISC-V machine code is mapped to the references to the C macro functions shown in Figure 6. This mapping is implemented using the `.4byte` RISC-V assembler directive as can be seen in Figures 8 and 10.

For example, the directive `.4byte 0xe7878b` used for mapping a CRC custom instruction has the following elements:

Elements of a RISC-V machine code: `0x00e7878b`.

- `Binary: 0b 0000 0000 1110 0111 1000 0111 1000 1011`
- `Opcode: 000 1011 (Required selector index:0xB)`
- `rd: 0111 1 (Destination register for result: x15, a5)`
- `funct3[2:0]: 000 (Extension index, ALU switch: 0x0; user-defined immediate)`
- `rs1: 0111 1 (Source register of data1: x15, a5)`
- `rs2: 0 1110 (Source register of data2: x14, a4)`
- `funct7[6:0]: 0000 000 (Optional selector index funct7[6:4] and extension index funct7[3:0]: 0x0; user-defined immediate)`
- `Assembler format of custom instruction: CUSTOM0 x15,x15,x14`

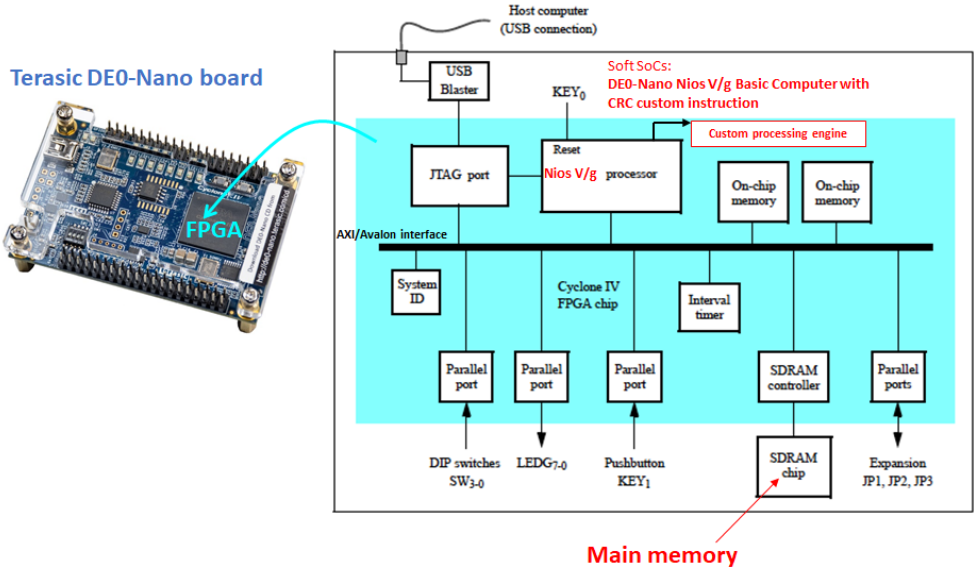

Figure 4: Microarchitecture of the soft System-on-Chip that is configured into the FPGA of the DE0-Nano board.

Question 2.
By analyzing the `lab5_app.elf.objdump` file, discover how the machine code of a CRC custom instruction is formatted in binary syntax. Show an example and explain the meaning of the elements of a custom instruction.

Question 3.
Disassembly the RISC-V machine code `.4byte 0xe7b78b` and write the custom instruction in assembler format.

Part IV. Software Engineering

This section of the lab assignment describes the design of programs that implement **three versions** of the CRC-32 algorithm.

- Slow version. Slow software for the CRC-32 algorithm based on the implementation of the modulo 2 division.
- Fast version. Fast software for the CRC-32 algorithm based on a the implementation of a lookup table.
- Custom instruction version. Very fast software program based on the new Nios V/g custom instruction for the CRC-32 algorithm that was describen in previous sections.

The software implementation of the CRC-32 algorithm is provided to students in the following files:

- `crc_main.c` - Main program that creates random test data and executes the three versions of CRC-32 algorithm using the C functions named: `crcSlow`, `crcFast` and `crcCI`.
- `crc.c` - Contains the slow and fast versions of the CRC-32 algorithm. The programming methods are called: `crcSlow()` and `crcFast()`.
- `crc.h` - Header file for `crc.c` source code.

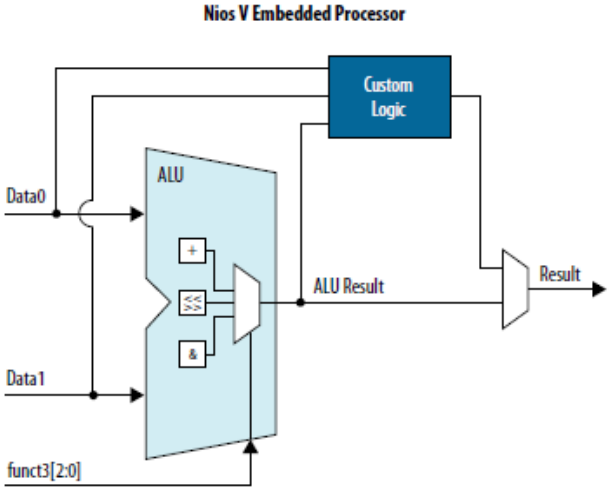

Figure 5: Custom ALU hardware module for the Nios V/g processor.

Custom instruction C macro.

```
#define ci_crc(VAL_1, VAL_2, F3) (
int output;
asm volatile (".insn r 0x0B, %[FUNCT3], 0x0, %[out], %[input1], %[input2]"
:  [out] "=r" (output)
:  [input1] "r" (VAL_1), [input2] "r" (VAL_2), [FUNCT3] "i" (F3));
output;
)
```

Figure 6: C macro procedure for CRC custom instructions that is include in `system.h` file at `lab5_bsp` directory.

- `ci_crc.c` - Source code for the program version that uses the Nios V/g custom instruction.
- `ci_crc.h` - Header file for the source code `ci_crc.c`.

Follow the same method as previously used for Nios V soft processor in other lab assigments but taking the `DE0_NanoBasic_Computer_26nov24.sof` configuration file for the DE0-Nano board and the file named `nios_system_26nov24.sof` for building the software drivers (see Figure 11). The configuration file for the FPGA named `DE0_NanoBasic_Computer_26nov24.sopcinfo` allows the DE0-Nano board to work as a System-on-Chip computer named *DE0-Nano Nios V/g Basic Computer with CRC custom instruction* after chip programming.

Question 4.
The procedure for the slow version is named `crcSlow()` in the `crc_main.c` file. After analyzing this file, do the following exercises:

1. Discover the software algorithm implemented in this procedure and write it in pseudocode language as previously shown in Algorithm 1.
2. Provide the assembler code for this procedure and explain its main steps. For this step, see the `build/lab5_app.elf.objdump` file where tha assembler code is saved after the compile and link processes.
3. Discover the section of this procedure where the largest percentage of instructions are executed for the slow version of the CRC-32 algorithm.

C code.

```
#include "system.h"
#define CRC_CI_MACRO(n, A) ci_crc(A, 0, n)
/* n = 0, Initialize the custom instruction to the initial remainder value */
CRC_CI_MACRO(0,0);
```

Figure 7: Example of C code that uses the custom instruction for CRC-32 algorithm.

Assembly code.

```
/* The custom instruction CRC will initialize to the inital remainder value */
CRC_CI_MACRO(0,0);
2d8:  00000793 li a5,0
2dc:  00000713 li a4,0
2e0:  00e7878b .4byte 0xe7878b
2e4:  fcf42a23 sw a5,-44(s0)
```

Figure 8: Assembly language syntax for the C code shown in Figure 7.

Question 5.
The procedure for the fast version is named `crcFast()` in the `crc_main.c` file. After analizing this file, do the following exercises:

1. Discover the software algorithm implemented in this procedure and write it in pseudocode language.
2. Provide the assembler code for this procedure and explain its main steps.
3. Discover the section of this procedure where the largest percentage of instructions are executed for the fast version of the CRC-32 algorithm.
4. Justify why this procedure executes in less time than the slow version.

Question 6.
The procedure for the version based on custom instructions is named `crcCI()` in the `crc_main.c` file. After analizing this file, do the following exercises:

1. Discover the software algorithm implemented in this procedure and write it in pseudocode language.
2. Provide the assembler code for this procedure and explain its main steps.
3. Discover the section of this procedure where the largest percentage of instructions are executed for the version of the CRC-32 algorithm based on custom instructions.

Part V. Performance Evaluation

Performance evaluation is defined as the process by which a computer system's resources and outputs are evaluated to determine whether the system is performing at an optimal level. This section describes the performance evaluation of a DE0-Nano board whose FPGA is configured to run the CRC-32 algorithm. The above mentioned three types of software implementations are studied: slow, fast, custom instructions. These programs use two types of Nios V/g instructions: standard RISC-V instructions for the slow and fast software implementations and custom instructions for the third software implementation.

C code.

```
#include "system.h"
/* n = 3, Write 32 bits data to custom instruction */
CRC_CI_MACRO(3, *(unsigned long *)input_data_copy);
```

Figure 9: Another example of C code that uses the custom instruction for CRC-32 algorithm.

Assembly code.

```
CRC_CI_MACRO(3, *(unsigned long *)input_data_copy);
2f0:  fd042783 lw a5,-48(s0)
2f4:  0007a783 lw a5,0(a5)
2f8:  00000713 li a4,0
2fc:  00e7b78b .4byte 0xe7b78b
300:  fef42623 sw a5,-20(s0)
```

Figure 10: Assembly language syntax for the C code shown in Figure 9.

Question 7.
After executing the three software versions, you will see on screen multiple messages as depicted in Figure 12. Using the measures shown in the Nios V Command Shell terminal, fill in the Table 1. This table should show the total execution times that were obtained for the three software implementation of the CRC-32 algorithm.

Table 1: Results of the performance evaluation.

Software version	Nios V/g instructions	Execution time	Speed-up
Modulo 2 division implementation (slow)	standard		
Lookup table implementation (fast)	standard		
Using custom instruction	customized		

As shown in the outcome of the program execution, the clock speed is 50 MHz (see Figure 12). This clock speed is not the maximum at which the Nios V/g soft processor can run when configured in the DE0-Nano board. The maximum clock speed is provided by the Quartus Prime software tool. In this case, the maximum clock speed is $86.63MHz$.

Question 8.

1. Assuming that the clock speed of Nios V/g processor is 86.63 MHz, obtain the execution time of the three software versions for the CRC algorithm
2. Obtain the execution times of the slow and fast versions of the CRC algorithm assuming that the Nios V/g processor employed in lab assignment 3 is used.
3. Calculate the speed-up of the software version based on custom instructions respect to the slow and fast versions executed by the Nios V/g processor used in lab assignment 3.
4. Calculate the average number of cycles per instruction (CPI) for all above cases.

Open a Nios V Command Shell terminal and introduce the following commands:

```
$ mkdir lab5_app
$ mkdir lab5_bsp
$ cd lab5_bsp
$ sh
$ niosv-bsp.exe -c -t=hal -s=nios_system_26nov24.sopcinfo settings.bsp
$ cd ../lab5_app
$ niosv-app.exe -a=.  -b=../lab5_bsp -s=.
$ cmake -S . -G "Unix Makefiles" -B build
$ make -C build
$ jtagconfig.exe
$ quartus_pgm -c 1 -m JTAG -o "p;DE0_Nano_Basic_Computer_26nov24.sof@1"
$ niosv-download.exe -g build/lab5_app.elf
$ juart-terminal.exe
```

Figure 11: Method to compile, link, configure the FPGA and download the program into main memory.

References

Intel. AN 977: Nios V Processor Custom Instruction. https://cdrdv2-public.intel.com/776470/ug-683632-776470.pdf, 2023a. Accessed: 2024-11-26.

Intel. Intel Agilex 7 FPGA Custom Instruction Design on Nios V/g processor - CRC. https://www.intel.com/content/www/us/en/design-example/789503/agilex-7-crc-custom-instruction-design-on-nios-v-g-processor.html, 2023b. Accessed: 2024-11-26.

Ross N. Williams. A painless guide to CRC error detections algorithms. http://chrisballance.com/wp-content/uploads/2015/10/CRC-Primer.html, 2022. Accessed: 2022-04-19.

OSDev. CRC32. https://wiki.osdev.org/CRC32, 2022. Accessed: 2022-04-19.

P. M. Nyasulu and J. Knight. *Introduction to Verilog*. Carleton University, https://www.cs.upc.edu/~jordicf/Teaching/secretsofhardware/VerilogIntroduction_Nyasulu.pdf, 2003. Accessed: 2022-04-19.

Intel. Nios II crc acceleration design example. https://www.intel.com/content/www/us/en/support/programmable/support-resources/design-examples/horizontal/exm-crc-acceleration.html, 2022. Accessed: 2024-1-28.

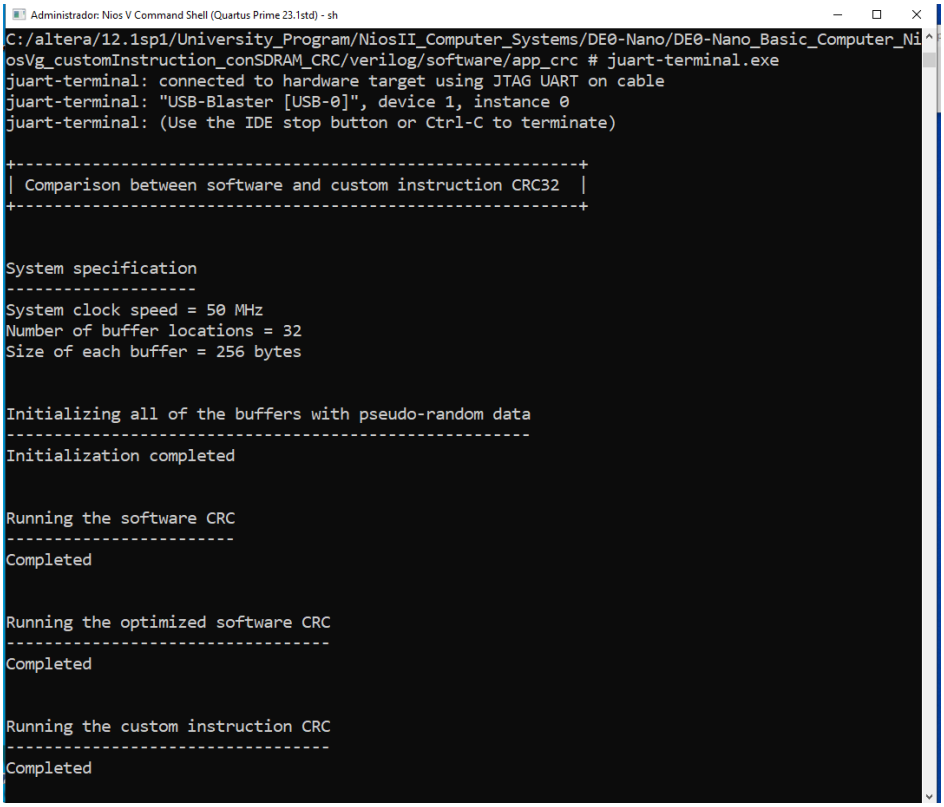

Figure 12: Messages shown on the Nios V Command Shell terminal after downloading the `lab5_app.elf` program into the DE0-Nano board.