Computer Architecture - Lab Assignment 4[*]

Nios II multiprocessor implementation, parallel programming, and performance evaluation

Introduction

The main objective of this laboratory experiment 4 consists on the evaluation of the Nios II dual-core multiprocessor performance and the comparison with a single Nios II processor. For the programming of the multiprocessor, two programs that are synchronized through a mutual exclusion hardware mechanism called semaphore will be carried out. The programs will run on a DE0-Nano board.

This practice contains three parts that are summarized below. For the preparation of this guide, the bibliographical references [1, 5, 2, 4, 3, 6, 8, 7] have been used.

Part 1. Intel-Altera Software Building Tools and Command Shell will be introduced for C language program development.

Part 2. Two C programming tutorials will be carried out using the tools mentioned in Part 1 and they will be executed on the DE0-Nano board found in one of the ULPGC laboratories (see Figure 1). To do this, each student will connect to a computer in the laboratory and will run the "Altera" virtual machine where the necessary tools to interact with the mentioned board are located. The first tutorial runs on a single processor and the second runs on two processors in parallel.

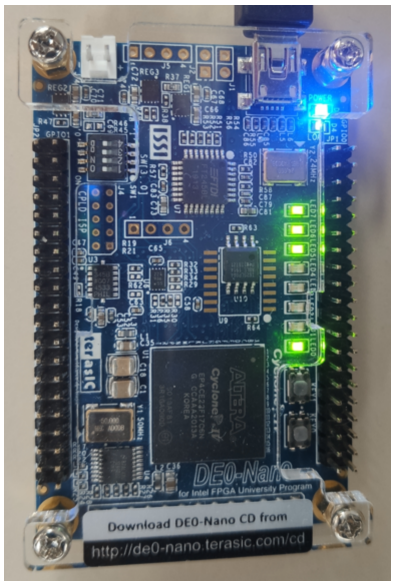


Figure 1: DE0-Nano board

Part 3. A tutorial consisting on the implementation of the Matrix x Vector algorithm will be carried out and, in addition, the performance of this program will be analyzed in two dual-core multiprocessors based

[*]AC 40969 - https://www.english.ulpgc.es/

on Nios II: 2 x Nios II/e, 2 x Nios II/s. Finally, an exercise is proposed to evaluate the performance of a matrix multiplication algorithm.

The computer material that accompanies this practice can be found in the following folders (see the Moodle section of the ULPGC corresponding to Practice 4):

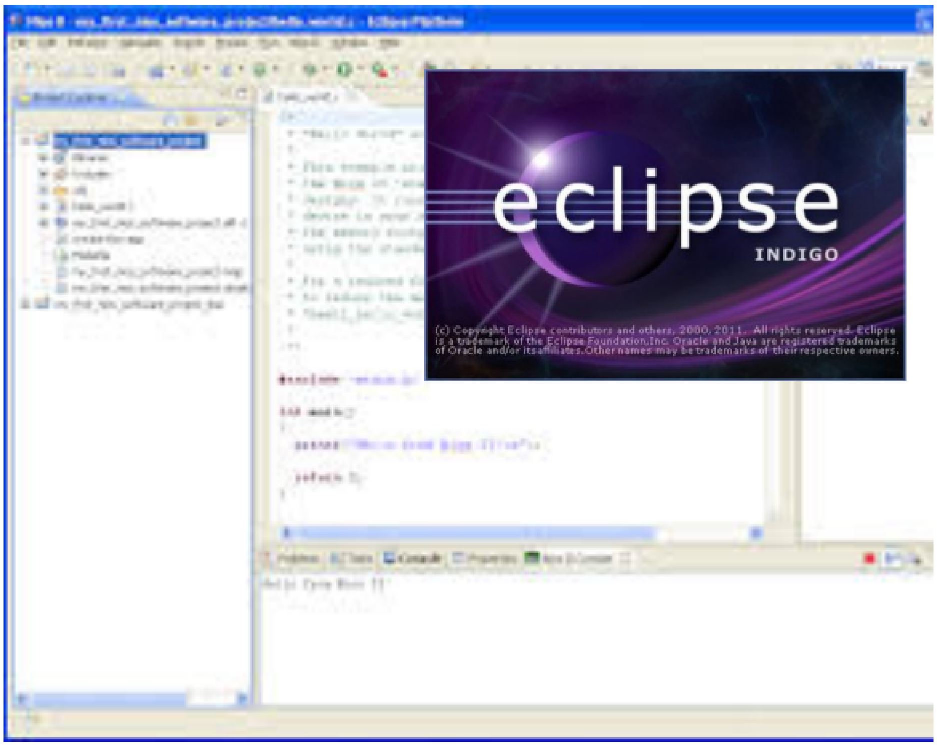
- Tutorial1
- Tutorial2
- Tutorial3
- DualCoreNios2e
- DualCoreNios2s
- Guide

Part 1. Software-hardware infrastructure of the practice

General description: In this part of the practice, the software tools that will be used to develop the C programs and the micro architecture of the Nios II multiprocessors, that will be configured in the DEO-Nano board, in which the C programs will run will be described.

Software Build Tools (SBT) for Eclipse: develop of C programs

The SBT tool consists on an user interface that automatise the compiling of C programs for the Nios II architecture (see Figure 2). It integrates a text editor, a program debugger and a FPGA device programmer.



(c) Copyright Eclipse contributors and others, 2000, 2011. All rights reserved. Eclipse is a trademark of the Eclipse Foundation,Inc. Oracle and Java are registered trademarks of Oracle and/or its affiliates.Other names may be trademarks of their respective owners.

Figure 2: SBT user interface for Eclipse

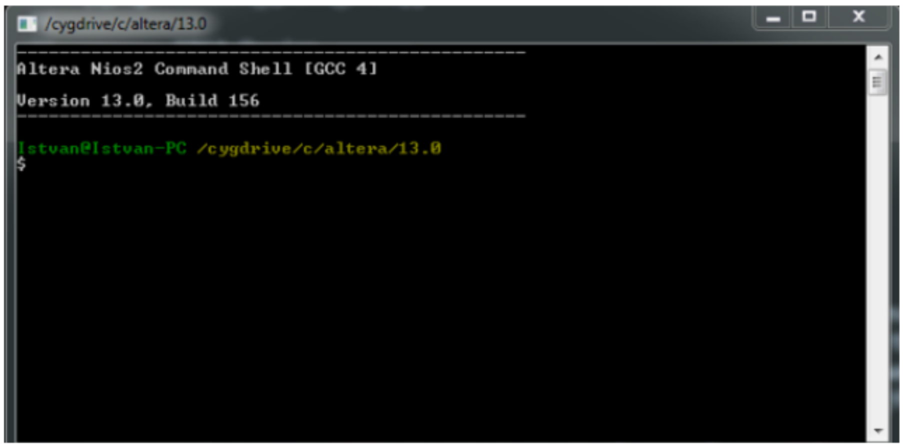
Nios II Command Shell: board setup, loading and running programs

With the Command Shell software tool you can create, modify, compile and run programs for Nios II processors (see Figure 3). To do this, the commands are written through the keyboard in the Command Shell window or inserted into a script file. The commands that will be used in this practice are the following:

$ nios2-configure <file>.sof
Use a sof file to configure the DE0-Nano board FPGA

$ nios2-download -r -g -i <core number> <file>.elf
Load a compiled elf program in the main memory of the Nios II processor that is found physically in the DEO-Nano board.

$ nios2-terminal
Displays in the window the results of printf statements that are executed in C programs.



```
------------------------------------------------
Altera Nios2 Command Shell [GCC 4]
Version 13.0, Build 156
------------------------------------------------
Istvan@Istvan-PC /cygdrive/c/altera/13.0
$
```

Figure 3: Command Shell window with the command line to enter commands of interaction with the DE0-Nano board.

Nios II multiprocessor with shared memory parallel architecture

Within the FPGA circuit of the DE0-Nano board, a multiprocessor integrated by two Nios II processor cores called CPU and CPU2, respectively, is configured in this practice. Additionally, the FPGA integrates the following elements with which it will interact:

- A semaphore device for 8-byte atomic operations (MUTEX). It ensures that a given device is accessed by one of the processors at a given instant of time. For this reason, it is called a mutual exclusion device.
- A 1 KiB SRAM memory to storage variables of the semaphore (BUFFER).

One feature of this multiprocessor is that both processor cores share the same physical RAM memory devices. For this reason, the multiprocessor architecture is called shared memory. Additionally, the multiprocessor can be used to execute the same program in parallel. That is why the architecture is also called parallel. The MUTEX device is used to synchronize the execution of the various threads into which a parallel program can be divided.

Other element of the DE0-Nano board that is necessary for this practice, is a SDRAM memory which storage capacity is 32 MiB. In the figure 4 a diagram of the Nios II multiprocessor micro-architecture is shown.

Two different Nios II multiprocessor configurations that differ in the processor will be handled. One of them, called DualCoreNios2e, integrates two copies of the Nios II/e processor core and the other configuration, called DualCoreNios2s, integrates two copies of the Nios II/s processor core. The Nios II/s processor includes a 1 KiB instruction cache and a 2 KiB data cache, both with 32 byte blocks.

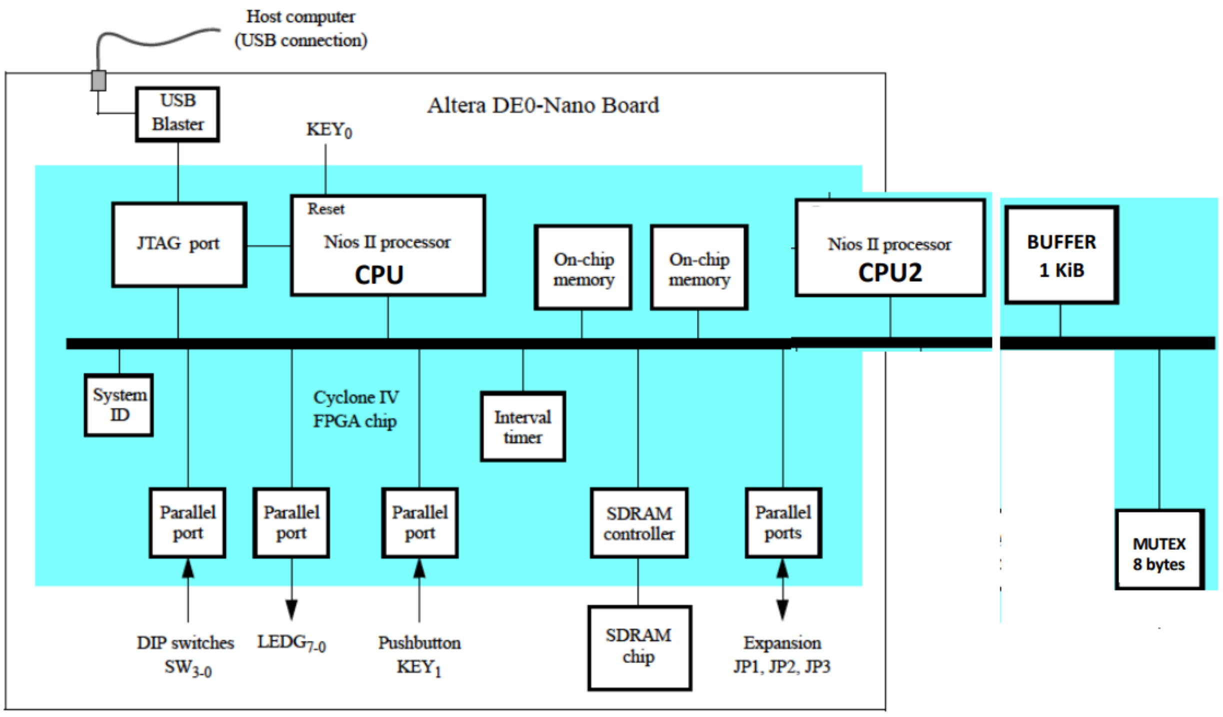


Figure 4: Dual core Nios II multiprocessor with shared memory parallel architecture. In this practice, two FPGA configurations that have a Nios II multiprocessor, called DualCoreNios2e and DualCoreNios2s, are handled.

Main memory organization of Nios II processors

When creating a multiprocessor system, it would be nice to run the software for multiple processors using the same physical memory device. The software for each processor must be located in a memory region that is not shared by any other processor but resides on the same physical device.

The SBT tool provides memory partitioning that allows multiple processors to run their software from different regions of the same physical memory device. Additionally, SBT ensures that the software of the processors is linked, determining the correct place in memory where the different parts in which this software is divided reside. To do this, SBT uses the exception address to calculate the memory area allocated to each processor.

Each processor has five linking zones, which are the following (see Figure 5):

- .text –where the executable code resides
- .rodata – where any data that is read-only and used in code execution is located
- .rwdata – where read-write variables and memory pointers are located
- .heap – where dynamic memory resides
- .stack – where are the parameters that are handled in calls to functions and subroutines as well as other temporary data

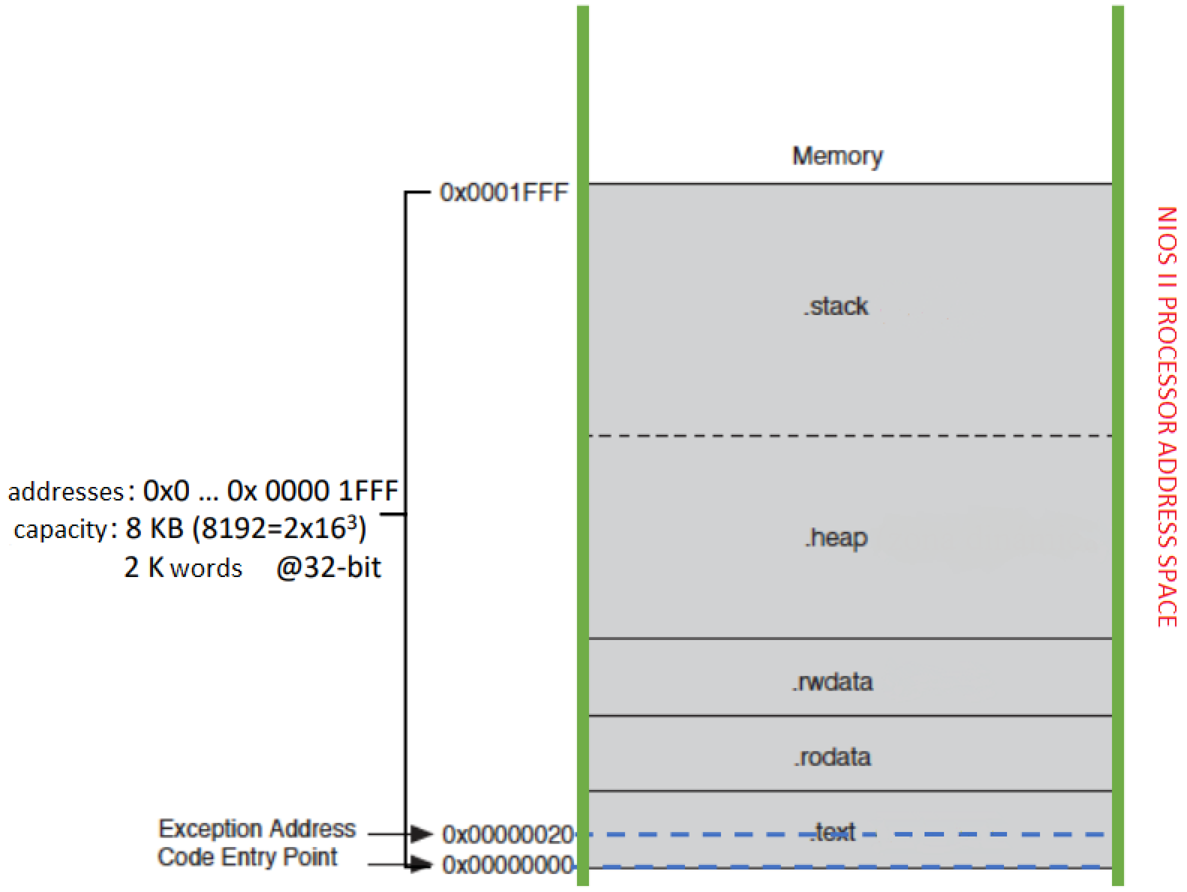


Figure 5: Example of main memory link zones for 1 core.

The SBT tool ensures that these memory sections are located (linked) at fixed memory addresses.

In Figure 5 you can see a memory map that shows how these link sections are assigned to fixed addresses for a single processor. In this figure it has been assumed that the processor software requires eight kilobytes (8 KiB) of memory. Therefore, the processor uses the region between addresses 0x0 and 0x1FFF. Figure 5 also shows the partitioning of the processor's main memory into several linking zones: code (.text), data (.rodata, .rwdata), stack (.stack) and dynamic zone (.heap). The exception address 0x0000 0020 is included in the code area.

In a multiprocessor system, it would be advantageous to use a single memory device to store all sections of code for each processor core. In our case, each processor's exception address is used to define the boundaries between the end of one processor's code sections and the beginning of the next processor core.

For example, in the case of this practice, the SDRAM main memory takes 32 MiB, between addresses 0x0000 0000 and 0x01FF FFFF. Within this range, the address ranges assigned to each of the two Nios II cores, CPU and CPU2, which are arranged in the FPGA configurations mentioned above: DualCoreNios2e and DualCoreNios2s, are established. In Figure 6 it can be seen that the address ranges assigned to the processor cores are:

- 0x0000 0000 a 0x003F FFFF – CPU core
- 0x0040 0000 a 0x007F FFFF – CPU2 core

Each core is allocated 4 MiB of main memory to run its software. The SBT tool automatically partitions main memory by looking at the address of exceptions. For both cores of the multiprocessor, this address has the values 0x0000 0020 for CPU and 0x0040 0020 for CPU2. See Figure 6 where the address assignment of the two cores is shown.

The six least significant bits of the exception address are always 0x20. At the address whose six least significant bits are 0x00 is the called entry point, where the NiosII processor expects to find the reset code. The address offset 0x20 is because the instruction cache line is 32 bytes long (8 32-bit instructions). These 32 bytes correspond to the extension of the reset code, which ends up skipping the exception handling code area to later start executing the main code of a certain processor core.

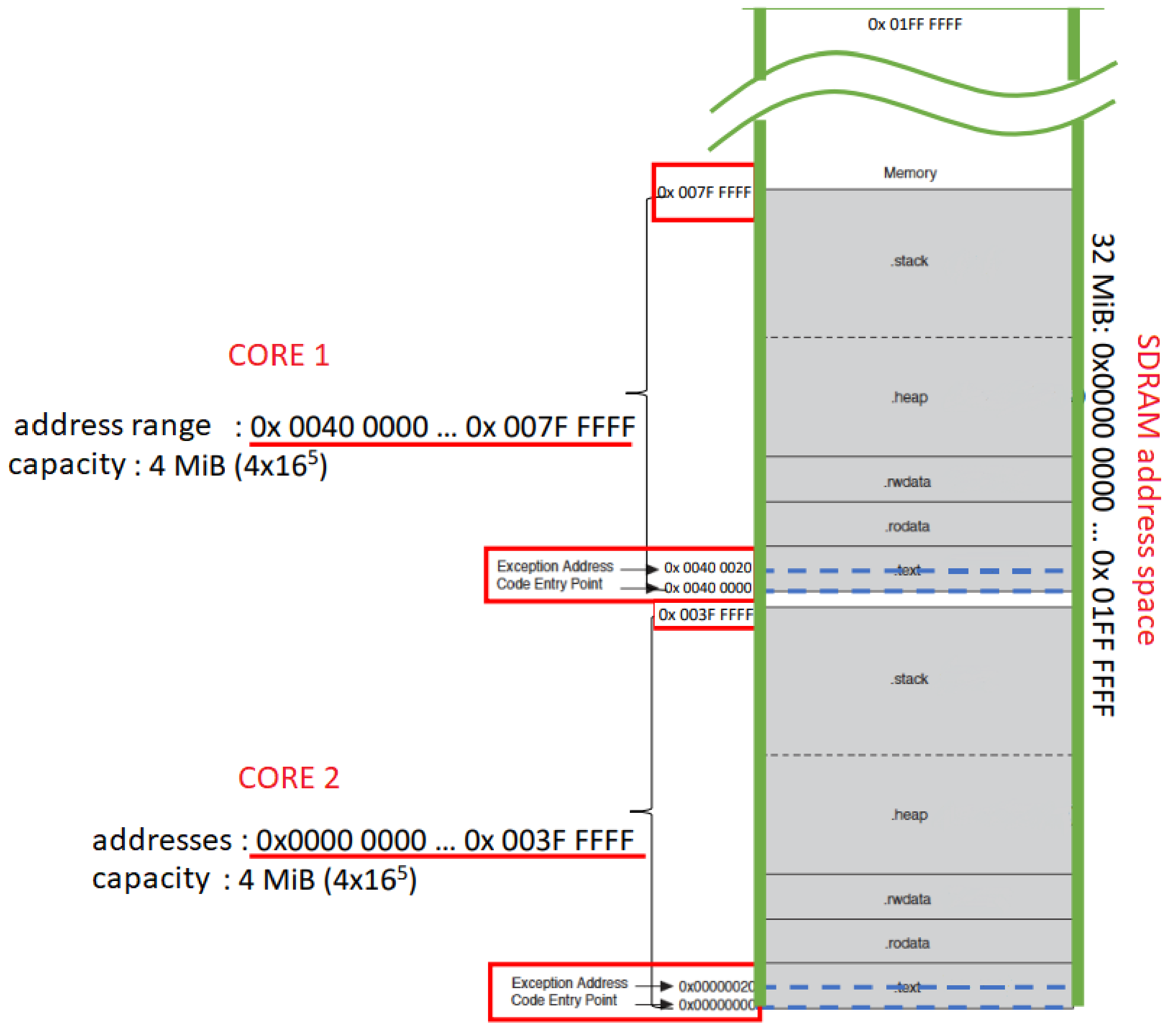


Figure 6: Linking areas of the main memory for the dual-core Nios II multiprocessors that are handled in this practice

Part 2. Tutorials 1 and 2 for programming the Nios II multiprocessor

General description: In this part, two C programs are implemented on the Nios II multiprocessor of the DualCoreNios2e configuration using the SBT and Command Shell tools. These two programs command that the CPU and CPU2 cores access the same main memory location concurrently. This means that the processors never access at the same time, but at different times. The MUTEX device allows coordinating access to the memory area shared by both processor cores.

Objective 2-1: Perform the steps mentioned below to run Tutorial-1 in which only one processor core (CPU) is used.

1. Open: Nios II – Software Build Tools - Eclipse (see Figure 7).

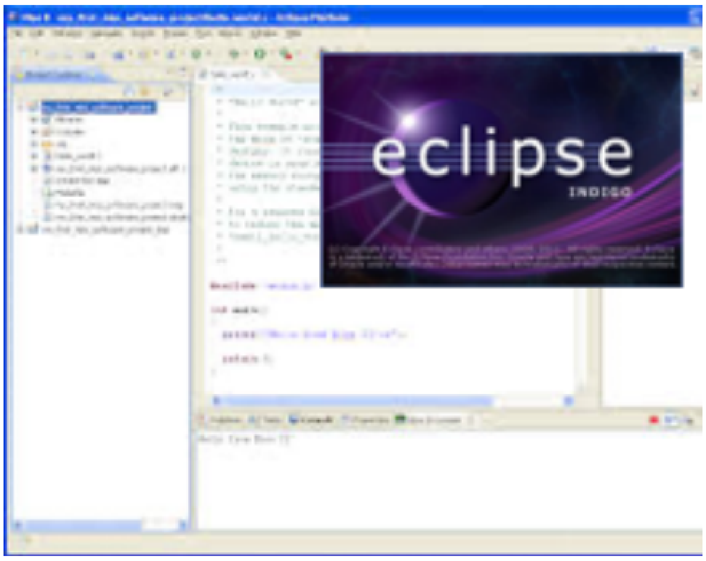
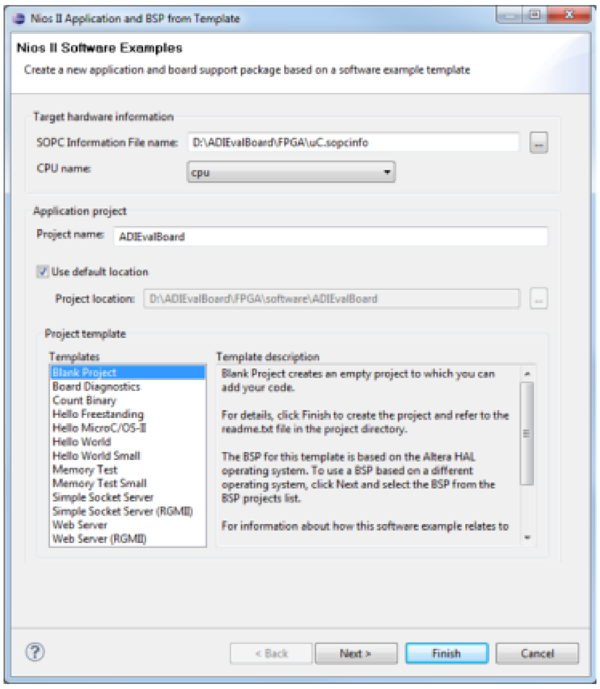


Figure 7: SBT for Eclipse

2. File > New > Nios II app & BSP from Template (see Figure 8).
   - SOPC Information File name: DE0_Nano_DualCoreNios2e.sopcinfo
   - CPU name: CPU (do not select CPU2)
   - Project name: hola_chicos_y_chicas_0
   - Project Location: choose ¡ProjectDir¿
   - Templates: select “Hello World Small”
   - Click “Next >”


Nios II Application and BSP from Template

Nios II Software Examples

Create a new application and board support package based on a software example template

Target hardware information

SOPC Information File name: D:\ADIEvalBoard\FPGA\uC.sopcinfo

CPU name: cpu

Application project

Project name: ADIEvalBoard

Use default location

Project location: D:\ADIEvalBoard\FPGA\software\ADIEvalBoard

Project template

Templates: Blank Project, Board Diagnostics, Count Binary, Hello Freestanding, Hello MicroC/OS-II, Hello World, Hello World Small, Memory Test, Memory Test Small, Simple Socket Server, Simple Socket Server (RGMII), Web Server, Web Server (RGMII)

Template description:
Blank Project creates an empty project to which you can add your code.

For details, click Finish to create the project and refer to the readme.txt file in the project directory.

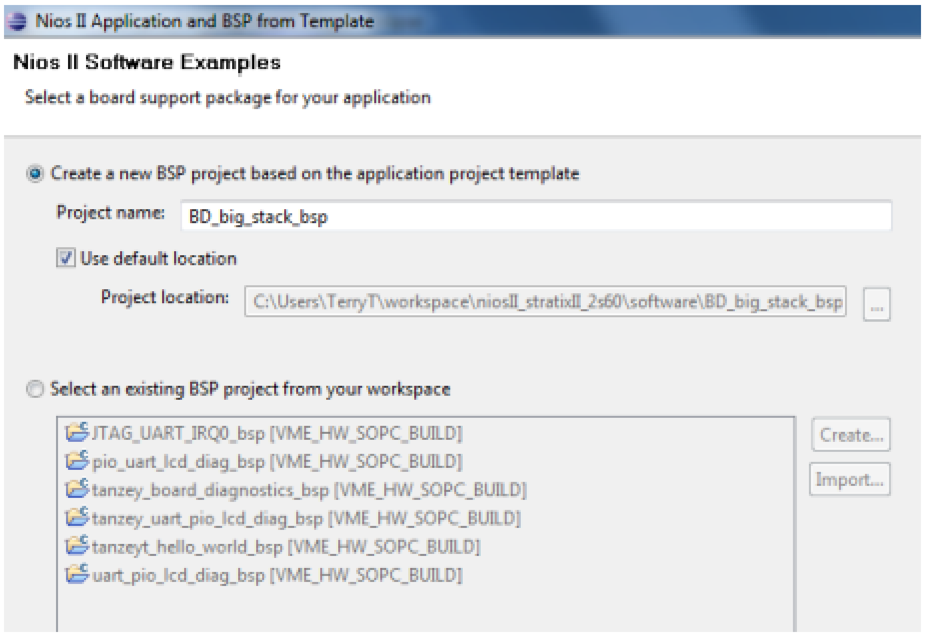
The BSP for this template is based on the Altera HAL operating system. To use a BSP based on a different operating system, click Next and select the BSP from the BSP projects list.

For information about how this software example relates to

Figure 8: Help for the generation of C programs with SBT.

3. Select “Create BSP”, project name: hola_chicos_y_chicas_0.bsp, use default location, click on “Finish” (see Figure 9)

- Result: in the Eclipse “Project Explorer” window, a project of type “app C/C++” and another project “BSP” (Board Support Package) are created



Nios II Application and BSP from Template

Nios II Software Examples

Select a board support package for your application

(●) Create a new BSP project based on the application project template

Project name: BD_big_stack_bsp

[✓] Use default location

Project location: C:\Users\TerryT\workspace\niosII_stratixII_2s60\software\BD_big_stack_bsp

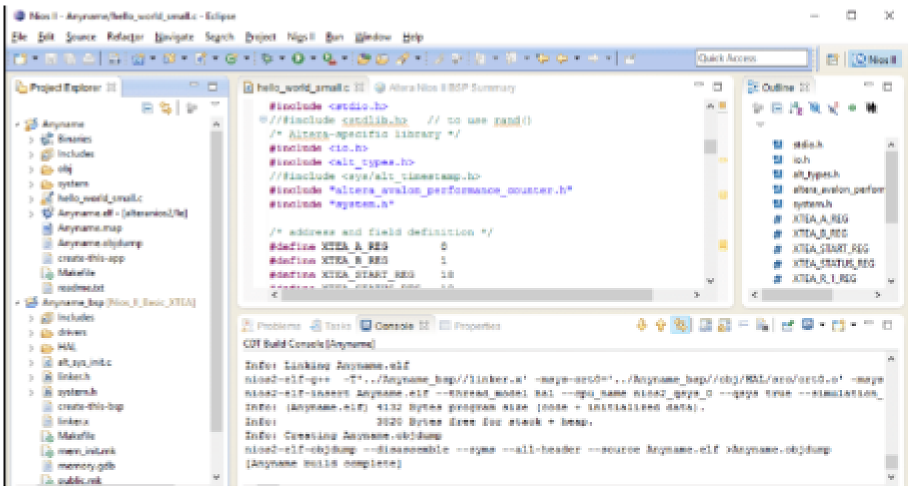
( ) Select an existing BSP project from your workspace

- JTAG_UART_IRQ0_bsp [VME_HW_SOPC_BUILD]
- pio_uart_lcd_diag_bsp [VME_HW_SOPC_BUILD]
- tanzey_board_diagnostics_bsp [VME_HW_SOPC_BUILD]
- tanzey_uart_pio_lcd_diag_bsp [VME_HW_SOPC_BUILD]
- tanzeyt_hello_world_bsp [VME_HW_SOPC_BUILD]
- uart_pio_lcd_diag_bsp [VME_HW_SOPC_BUILD]

Figure 9: Window where the BSP (Board Support Package) project is defined.

4. Display the project hola_chicos_y_chicas_0 in the “Project Explorer” window and click twice on hello_world.c (see Figure 10)

   - Result: a window with the source code is open



```
#include <stdio.h>
//#include <stdlib.h>   // to use rand()
/* Altera-specific library */
#include <io.h>
#include <alt_types.h>
//#include <sys/alt_timestamp.h>
#include "altera_avalon_performance_counter.h"
#include "system.h"

/* address and field definition */
#define XTEA_A_REG          0
#define XTEA_B_REG          1
#define XTEA_START_REG      10
```

```
Info: Linking Anyname.elf
nios2-elf-g++  -T'../Anyname_bsp//linker.x' -msys-crt0='../Anyname_bsp//obj/HAL/src/crt0.o' -msys
nios2-elf-insert Anyname.elf --thread_model hal --cpu_name nios2_gsys_0 --qsys true --simulation_
Info: (Anyname.elf) 4132 Bytes program size (code + initialized data).
Info:               3020 Bytes free for stack + heap.
Info: Creating Anyname.objdump
nios2-elf-objdump --disassemble --syms --all-header --source Anyname.elf >Anyname.objdump
[Anyname build complete]
```

Figure 10: Window where the source code is displayed in the C programming language.

5. Replace the content of the hello_world.c file in the project directory with the content of the file Tutorial1/hola_chicos_y_chicas.c (see Figure 11)

```
#include <stdio.h>

int main()
{
  printf("Hola chicos de AC!\n");
  printf("Hola chicas de AC!\n");

  return 0;
}
```

Figure 11: Source code in C of Tutorial 1. File: hola_chicos_y_chicas.c

6. Compile and link (see Figure 12)

   - Right click on the C/C++ project in the Project Explorer window.
   - Select: Build Project.
   - Result: if the process ends successfully, the message "hola_chicos_y_chicas_0 Build complete" appears in the lower "Console" window. Two projects are created in the Project Explorer window and two folders in <ProjectDir >.

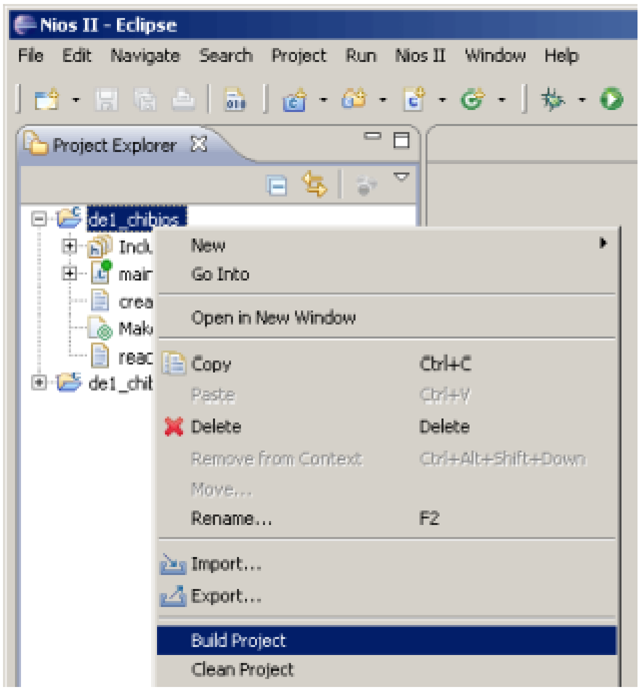
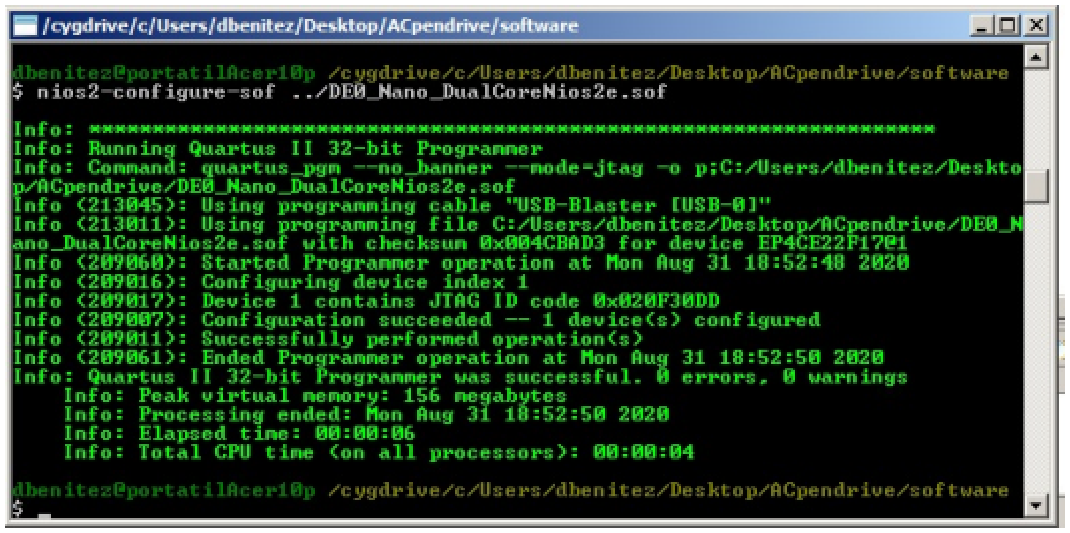


Figure 12: Window where the compilation and linking command of the C code is displayed.

7. DE0-Nano board configuration (see Figure 13):

   - Connect the board to the USB connector (it is already connected in the Laboratory)
   - Open: Start > Altera > NiosII Command Shell
   - File DE0_Nano_Basic_Computer.sof is loaded in the board:
     (a) $ cd <directory with the FPGA configurations from Practice 4>
     (b) $ nios2-configure-sof DE0_Nano_DualCoreNios2e.sof (try it several times if it gives JTAG error)
   - Result: the message "Quartus II 32-bit Programmer was successful" should appear (see Figure 13).

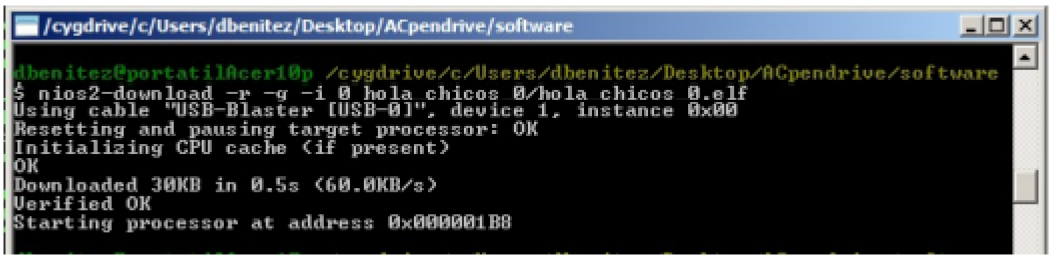


```
/cygdrive/c/Users/dbenitez/Desktop/ACpendrive/software

dbenitez@portatilAcer10p /cygdrive/c/Users/dbenitez/Desktop/ACpendrive/software
$ nios2-configure-sof ../DE0_Nano_DualCoreNios2e.sof

Info: *******************************************************************
Info: Running Quartus II 32-bit Programmer
Info: Command: quartus_pgm --no_banner --mode=jtag -o p;C:/Users/dbenitez/Deskto
p/ACpendrive/DE0_Nano_DualCoreNios2e.sof
Info (213045): Using programming cable "USB-Blaster [USB-0]"
Info (213011): Using programming file C:/Users/dbenitez/Desktop/ACpendrive/DE0_N
ano_DualCoreNios2e.sof with checksum 0x004CBAD3 for device EP4CE22F17@1
Info (209060): Started Programmer operation at Mon Aug 31 18:52:48 2020
Info (209016): Configuring device index 1
Info (209017): Device 1 contains JTAG ID code 0x020F30DD
Info (209007): Configuration succeeded -- 1 device(s) configured
Info (209011): Successfully performed operation(s)
Info (209061): Ended Programmer operation at Mon Aug 31 18:52:50 2020
Info: Quartus II 32-bit Programmer was successful. 0 errors, 0 warnings
    Info: Peak virtual memory: 156 megabytes
    Info: Processing ended: Mon Aug 31 18:52:50 2020
    Info: Elapsed time: 00:00:06
    Info: Total CPU time (on all processors): 00:00:04

dbenitez@portatilAcer10p /cygdrive/c/Users/dbenitez/Desktop/ACpendrive/software
$ _
```

Figure 13: Result of the $ nios2-configure-sof command to configure the DE0-Nano board FPGA.

8. Program execution (see Figure 14):

   - $ cd <project directory hola_chicos_y_chicas_0>
   - $ nios2-download -r -g -i 0 hola_chicos_y_chicas_0.elf

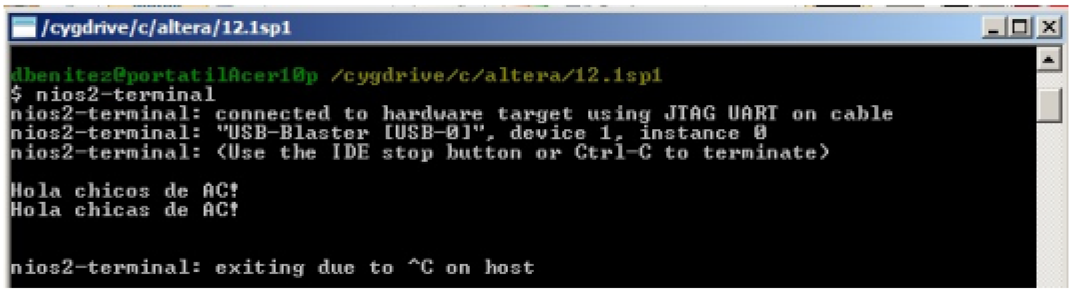


```
/cygdrive/c/Users/dbenitez/Desktop/ACpendrive/software

dbenitez@portatilAcer10p /cygdrive/c/Users/dbenitez/Desktop/ACpendrive/software
$ nios2-download -r -g -i 0 hola_chicos_0/hola_chicos_0.elf
Using cable "USB-Blaster [USB-0]", device 1, instance 0x00
Resetting and pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 30KB in 0.5s (60.0KB/s)
Verified OK
Starting processor at address 0x000001B8
```

Figure 14: Result of the $ nios2-download command to load the executable program in the SDRAM memory of the DE0-Nano board.

9. View results (see Figure 15):

   - $ nios2-terminal
     (a) Result: the corresponding messages to the printf of the source code will appear in the window
   - Close Command Shell window



```
/cygdrive/c/altera/12.1sp1

dbenitez@portatilAcer10p /cygdrive/c/altera/12.1sp1
$ nios2-terminal
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

Hola chicos de AC!
Hola chicas de AC!


nios2-terminal: exiting due to ^C on host
```

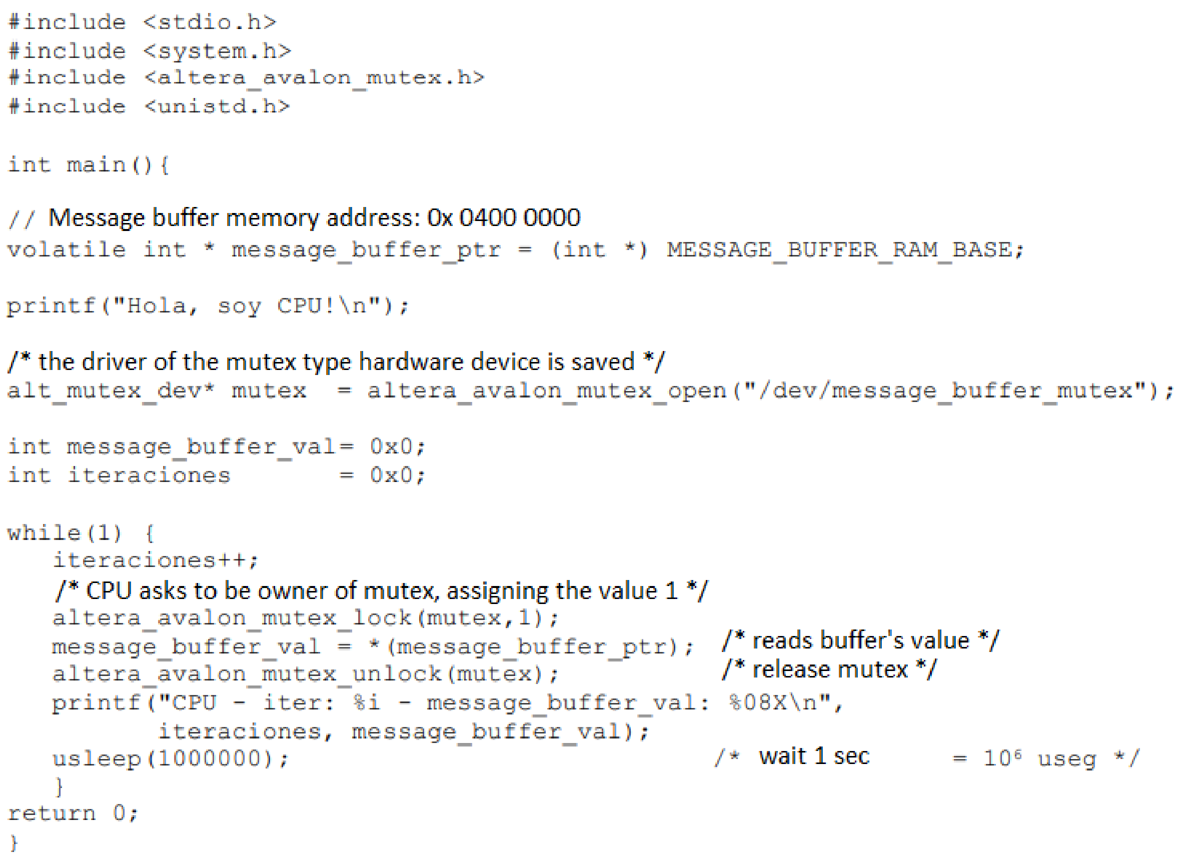
Figure 15: Display of Tutorial-1 messages after running the $ nios2-terminal command in a new Command Shell window.

Practice files for Tutorial-1:

- C source code: hola_chicos_y_chicas.c (folder: Tutorial1).
- FPGA configuration files: DE0_Nano_DualCoreNios2e.sopcinfo, DE0_Nano_DualCoreNios2e.sof (folder: DualCoreNios2e).

Objective 2-2: Perform the steps mentioned below to run Tutorial 2 using two processor cores (CPU and CPU2). Two projects are created following the steps in Tutorial-1.

1. Common elements of both projects:
   - SOPC Information File name: DE0_Nano_DualCoreNios2e.sopcinfo
   - Project Location: choose <ProjectDir>
   - Templates: select "Hello World Small"
2. Project 1:
   - CPU name: CPU (do not select CPU2)
   - Project name: hola_semaforo_0
   - BSP project, project name: hola_chicos_y_chicas_0.bsp
   - Replace the content of the hello_world.c file in the project directory with the content of the file Tutorial2/hola_semaforo_0.c (see Figure 16). Note that the CPU processor core only reads the message_buffer_val variable. The display of results in CPU is done with the printf() function. Only CPU is connected to the JTAG interface so its program is the only one that can have printf.
   - Compile and link

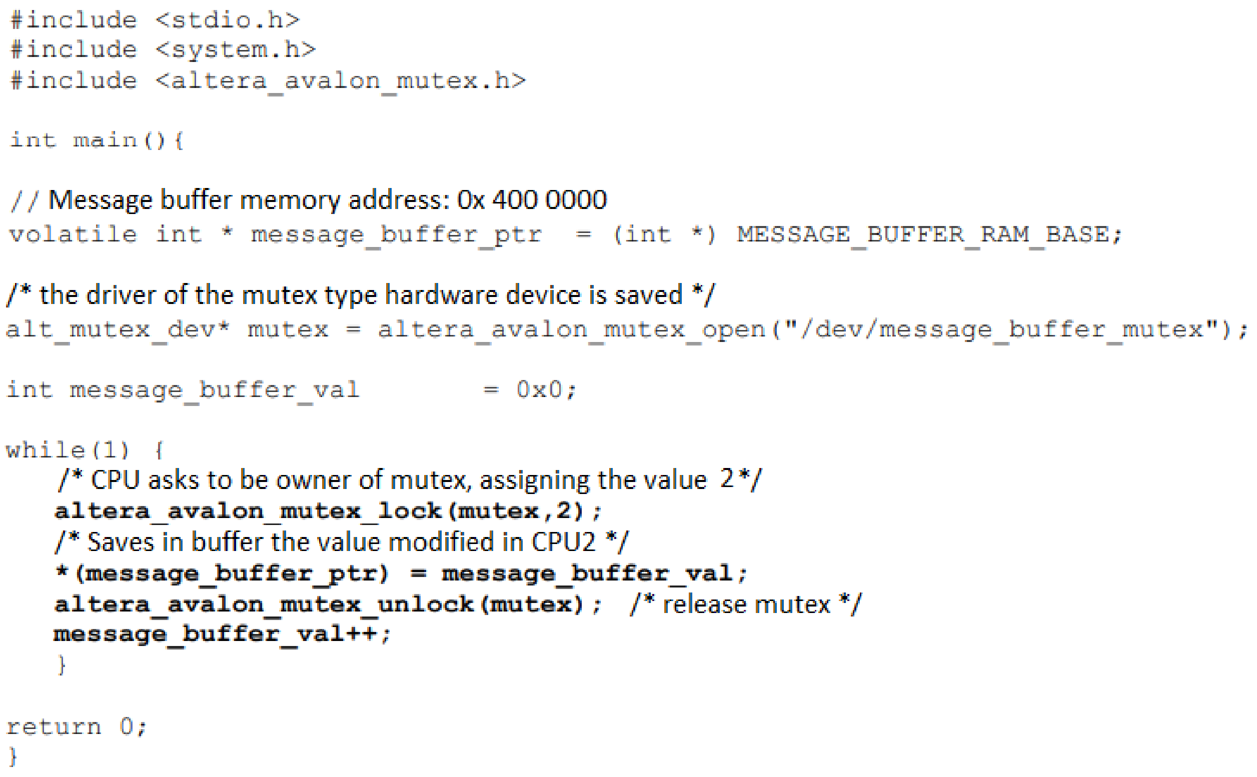


```
#include <stdio.h>
#include <system.h>
#include <altera_avalon_mutex.h>
#include <unistd.h>

int main(){

// Message buffer memory address: 0x 0400 0000
volatile int * message_buffer_ptr = (int *) MESSAGE_BUFFER_RAM_BASE;

printf("Hola, soy CPU!\n");

/* the driver of the mutex type hardware device is saved */
alt_mutex_dev* mutex  = altera_avalon_mutex_open("/dev/message_buffer_mutex");

int message_buffer_val= 0x0;
int iteraciones       = 0x0;

while(1)  {
   iteraciones++;
   /* CPU asks to be owner of mutex, assigning the value 1 */
   altera_avalon_mutex_lock(mutex,1);
   message_buffer_val = *(message_buffer_ptr);  /* reads buffer's value */
   altera_avalon_mutex_unlock(mutex);           /* release mutex */
   printf("CPU - iter: %i - message_buffer_val: %08X\n",
          iteraciones, message_buffer_val);
   usleep(1000000);                            /*  wait 1 sec      = 10^6 useg */
   }
return 0;
}
```

Figure 16: Source code in C (hola_semaforo_0.c) for the processor core 0 (CPU) of Tutorial-2.

3. Project 2:
   - CPU name: CPU2

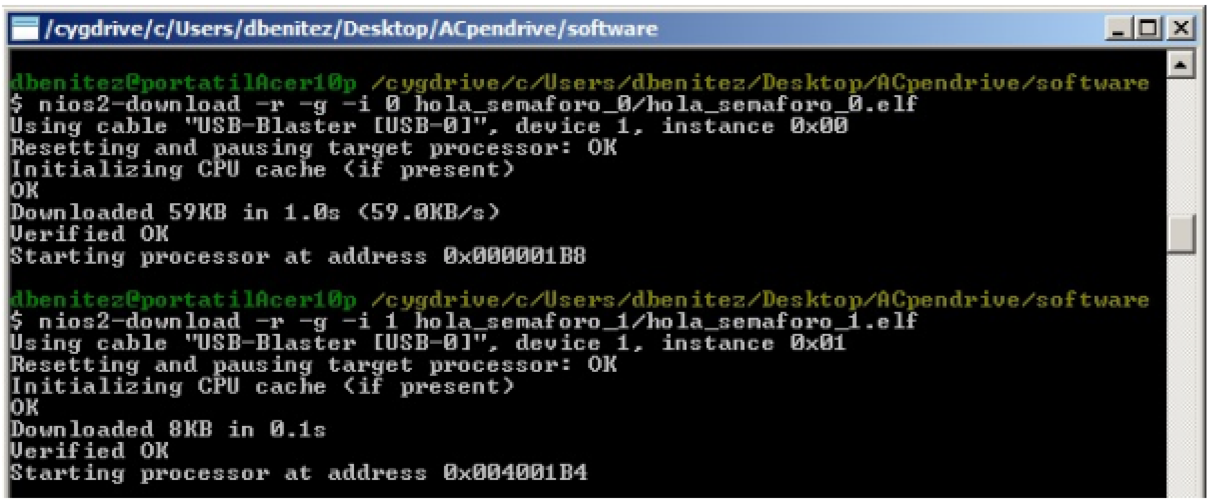
- Project name: hola_semaforo_1
- BSP project, project name: hola_chicos_y_chicas_1.bsp
- Replace the content of the hello_world.c file in the project directory with the content of the file Tutorial2/hola_semaforo_1.c (see Figure 17). Note that the processor core CPU2 modifies the message_buffer_val variable.
- Compile and link



```
#include <stdio.h>
#include <system.h>
#include <altera_avalon_mutex.h>

int main(){

// Message buffer memory address: 0x 400 0000
volatile int * message_buffer_ptr  = (int *) MESSAGE_BUFFER_RAM_BASE;

/* the driver of the mutex type hardware device is saved */
alt_mutex_dev* mutex = altera_avalon_mutex_open("/dev/message_buffer_mutex");

int message_buffer_val        = 0x0;

while(1)  {
    /* CPU asks to be owner of mutex, assigning the value  2*/
    altera_avalon_mutex_lock(mutex,2);
    /* Saves in buffer the value modified in CPU2 */
    *(message_buffer_ptr)  = message_buffer_val;
    altera_avalon_mutex_unlock(mutex);   /* release mutex */
    message_buffer_val++;
    }

return 0;
}
```

Figure 17: Source code in C (hola_semaforo_1.c) for the processor core 1 (CPU2) of Tutorial-2.

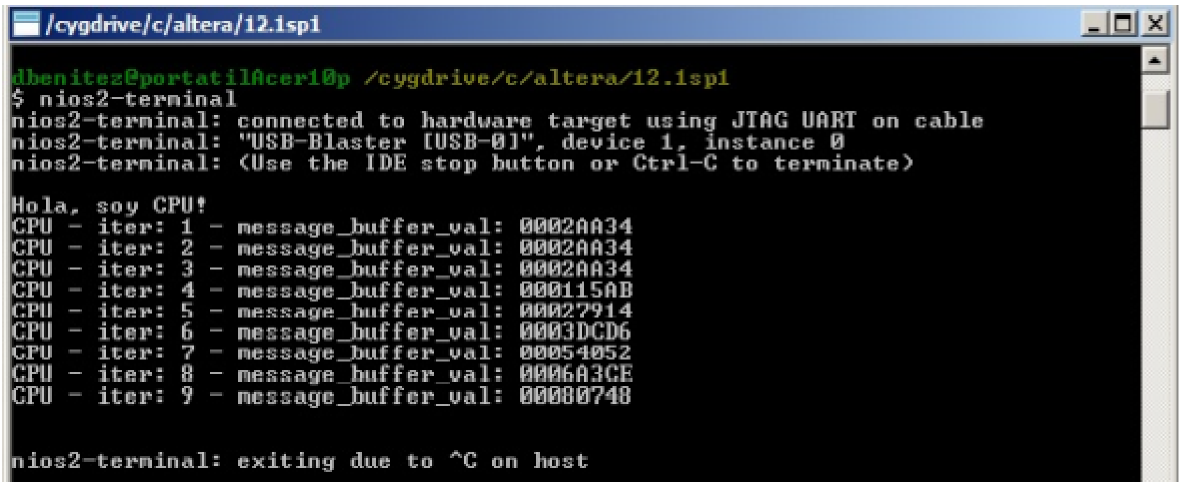
4. DE0-Nano board configuration:
   - Connect the board to the USB connector (it is already connected in the Laboratory)
   - Open: Start > Altera > NiosII Command Shell
   - File DE0_Nano_Basic_Computer.sof is loaded in the board:
     (a) $ cd <directory with the FPGA configurations from Practice 4>
     (b) $ nios2-configure-sof DE0_Nano_DualCoreNios2e.sof (try it several times if it gives JTAG error)
   - Result: the message “Quartus II 32-bit Programmer was successful” should appear
5. Run the program (see Figure 18):
   - $ nios2-download -r -g -i 0 hola_semaforo_0/hola_semaforo_0.elf
   - $ nios2-download -r -g -i 1 hola_semaforo_1/hola_semaforo_1.elf



```
dbenitez@portatilAcer10p /cygdrive/c/Users/dbenitez/Desktop/ACpendrive/software
$ nios2-download -r -g -i 0 hola_semaforo_0/hola_semaforo_0.elf
Using cable "USB-Blaster [USB-0]", device 1, instance 0x00
Resetting and pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 59KB in 1.0s (59.0KB/s)
Verified OK
Starting processor at address 0x000001B8

dbenitez@portatilAcer10p /cygdrive/c/Users/dbenitez/Desktop/ACpendrive/software
$ nios2-download -r -g -i 1 hola_semaforo_1/hola_semaforo_1.elf
Using cable "USB-Blaster [USB-0]", device 1, instance 0x01
Resetting and pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 8KB in 0.1s
Verified OK
Starting processor at address 0x004001B4
```

Figure 18: Result of the $ nios2-download commands to load the executable programsin the SDRAM memory of the DE0-Nano board.

6. View results (see Figure 19):
   - $ nios2-terminal
   - $ CTRL-C



```
dbenitez@portatilAcer10p /cygdrive/c/altera/12.1sp1
$ nios2-terminal
nios2-terminal: connected to hardware target using JTAG UART on cable
nios2-terminal: "USB-Blaster [USB-0]", device 1, instance 0
nios2-terminal: (Use the IDE stop button or Ctrl-C to terminate)

Hola, soy CPU!
CPU - iter: 1 - message_buffer_val: 0002AA34
CPU - iter: 2 - message_buffer_val: 0002AA34
CPU - iter: 3 - message_buffer_val: 0002AA34
CPU - iter: 4 - message_buffer_val: 000115AB
CPU - iter: 5 - message_buffer_val: 00027914
CPU - iter: 6 - message_buffer_val: 0003DCD6
CPU - iter: 7 - message_buffer_val: 00054052
CPU - iter: 8 - message_buffer_val: 0006A3CE
CPU - iter: 9 - message_buffer_val: 00080748

nios2-terminal: exiting due to ^C on host
```

Figure 19: Display of Tutorial-2 messages after executing the command $ nios2-terminal in a new Command Shell window.

<u>**Practice files for Tutorial-2:**</u>

- C source code: hola_semaforo_0.c, hola_semaforo_1.c (folder: Tutorial2).
- FPGA configuration files: DE0_Nano_DualCoreNios2e.sopcinfo, DE0_Nano_DualCoreNios2e.sof (folder: DualCoreNios2e).

Part 3. Multi-threaded parallel programming and performance evaluation of dual core Nios II multiprocessors

General description: In this part of the practice, Tutorial-3 is carried out where the Matrix x Vector multiplication algorithm is implemented. This algorithm will run on a single processor core and two multiprocessors based on Nios II/e and Nios II/s, respectively. Finally, an exercise is proposed where it is proposed to implement the matrix multiplication algorithm.

The C code of Tutorial-3 consists of a loop of Niter iterations in which each iteration multiplies a matrix of n=16 rows x m=16 columns (A[i*m+j]) and a vector of 16 components (x [j]). The result consists of 16 components (y[i]): y[i] = A[i*m+j]. x[j]. Matrix and vectors components are integer values. The mathematical operation is: y = A * x (see Figure 20).

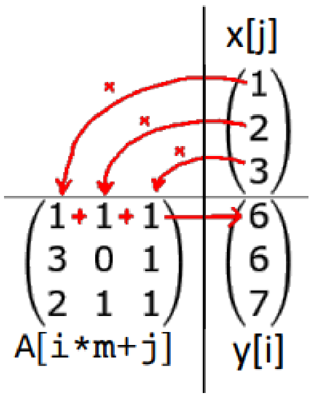
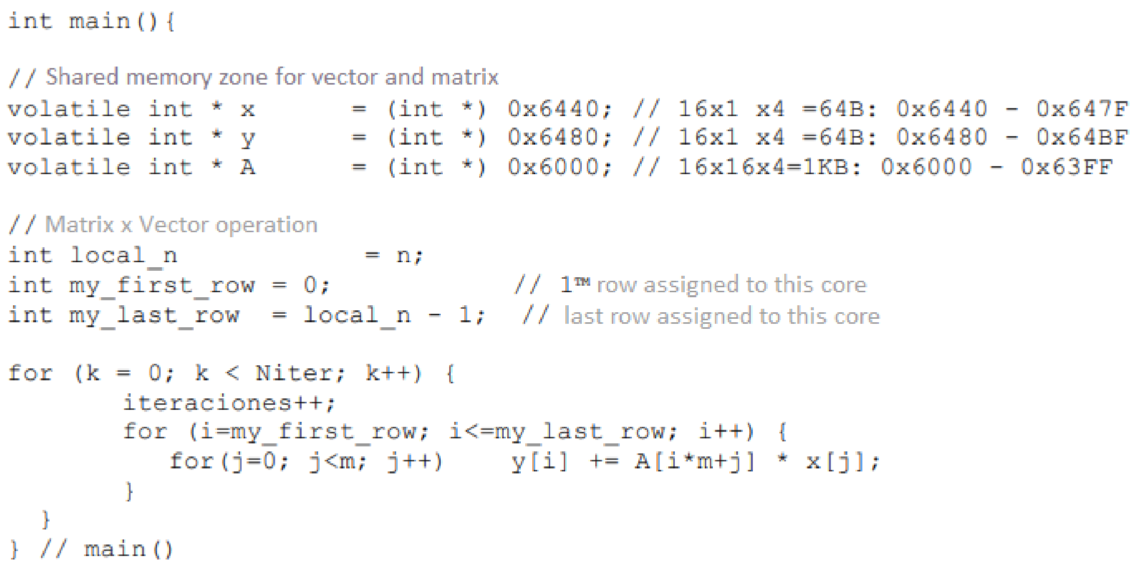


Figure 20: Matrix x Vector algorithm: y[i] = A[i*m+j] * x[j].

The main loop in C of Tutorial-3 is the next, (see Figure 21):

```
for (i=my_first_row; i<=my_last_row; i++) {
  for(j=0; j<m; j++) y[i] += A[i*m+j] * x[j]; }
```



```
int main(){

// Shared memory zone for vector and matrix
volatile int * x      = (int *) 0x6440; // 16x1 x4 =64B: 0x6440 - 0x647F
volatile int * y      = (int *) 0x6480; // 16x1 x4 =64B: 0x6480 - 0x64BF
volatile int * A      = (int *) 0x6000; // 16x16x4=1KB: 0x6000 - 0x63FF

// Matrix x Vector operation
int local_n            = n;
int my_first_row = 0;          // 1™ row assigned to this core
int my_last_row  = local_n - 1;  // last row assigned to this core

for (k = 0; k < Niter; k++) {
       iteraciones++;
       for (i=my_first_row; i<=my_last_row; i++) {
          for(j=0; j<m; j++)     y[i] += A[i*m+j] * x[j];
       }
   }
} // main()
```

Figure 21: MV_serie.c.

In this tutorial, times will be recorded using the CPU processor's Timer and its HAL (Hardware Abstraction Layer) function/driver called alt_timestamp(). To correctly configure the time recording, it is necessary to perform the following steps.

Timer setting of the processor named CPU (first dual core processor):

1. Generate an SBT project with the file DE0_Nano_DualCoreNios2e.sopcinfo for the CPU processor following the same steps as in Tutorial 1. The following project name can be used: MV_serie. The software project includes a BSP project.

2. Edit the configuration of the CPU processor BSP project (MV_serie_bsp). To do this, you need to perform the following actions: (right click on the BSP project) Nios II > BSP editor > timestamp_timer > Value= interval_timer > Save.
3. In the BSP editor and in the "Target BSP Directory" window: click on Generate.
4. Compile + link C program: (right click on C project) > Build Project.

Note: If the execution of the *.elf program indicates that the execution time is 0, it means that the Timer driver is not properly configured. In this case, try to compile the program by executing make in the MV_series project directory

As in Tutorial-2, display of results can only be done through the CPU core using the printf() function.

Objective 3-1: Follow the steps in Tutorial-1 with the file with C code MV_serie.c so that a project is generated in SBT. Next, configure the DE0-Nano board with the FPGA configuration: DualCoreNios2e, and run the sequential program on the CPU core. Next, carry out the following activities:

1. Record execution times in CPU for four workloads.
   - Parameters: Niter = 1000, 2000, 5000, 10000.
2. Performance evaluation: make Tables 1 and 2, whose columns represent the following:
   - Add Noperations: number of add operations in all the Niter iterations
   - Mult Noperations: number of mult operations in all the Niter iterations
   - N loads: number of load instructions
   - N stores: number of store instructions
   - Time: execution time of the whole program

   Table 1. Analysis of the computational and memory accesses charge of the Matrix x Vector algorithm.

Niter	Add Noperations	Mult Noperations	N loads	N stores
1000				
2000				
5000				
10000				

   Table 2. Records of the execution times of the Matrix x Vector sequential algorithm for one of the processor cores (CPU) of two Nios II multiprocessors.

FPGA Cofiguration	Niter	Time (ms)	Speed-up
DualCoreNios2e	1000		1
DualCoreNios2e	2000		1
DualCoreNios2e	5000		1
DualCoreNios2e	10000		1
DualCoreNios2s	1000		
DualCoreNios2s	2000		
DualCoreNios2s	5000		
DualCoreNios2s	10000		

3. Repeat the results for Nios II/s using the processor core called CPU that is integrated in the FPGA configuration: DualCoreNios2s.

Questions to justify the results:

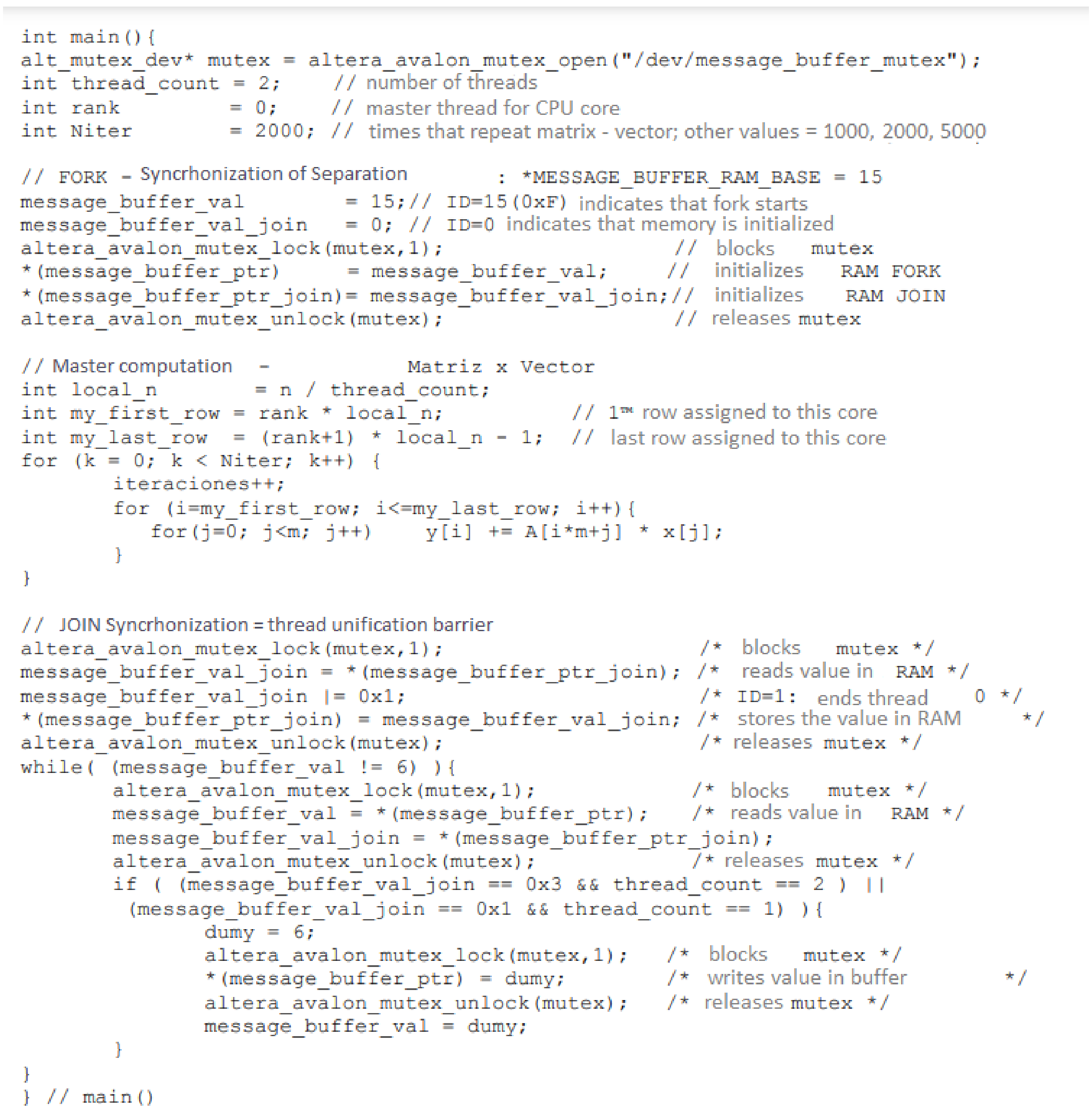
1. Is it reasonable that doubling the number of arithmetic operations and memory accesses would cause the execution time of one of the Nios II/e processors in the DualCoreNios2e multi-processor to double? Why?
2. Is it reasonable that the results of execution times with Nios II/s are significantly lower than those obtained with Nios II/e? Why? What is the average speed-up provided by the Nios II/s core relative to the Nios II/e core?

Objective 3-2: Follow the steps made in Tutorial-2 so that with the files with C code MV_paralelo_maestro.c (see Figure 22) and MV_paralelo_esclavo.c (see Figure 23) two projects are generated in SBT. Next, configure the DE0-Nano board with the FPGA configuration: DualCoreNios2e, and run the parallel program using the CPU and CPU2 cores.

Use of the Timer of the processor named CPU (first processor of the dual core):

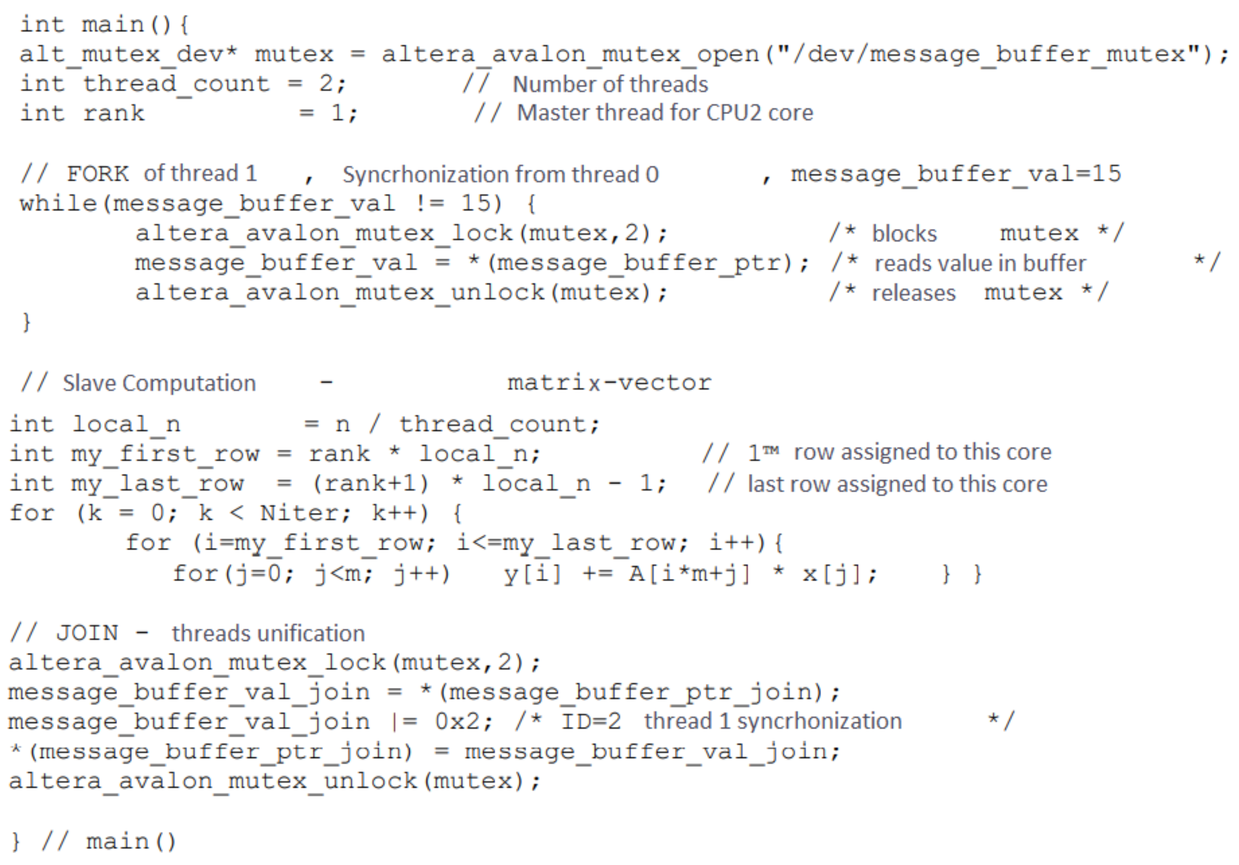
1. Generate two SBT projects with the DE0_Nano_DualCoreNios2e.sopcinfo file, one for the CPU processor and the other for the CPU2 processor. The following project names can be used: MV_paralelo_maestro, MV_paralelo_esclavo. Each project includes a BSP project.
2. Edit the configuration of the CPU processor BSP project (MV_paralelo_maestro_bsp): (right click on the BSP project) Nios II > BSP editor > timestamp_timer > Value = interval_timer > Save.
3. In the BSP editor > in "Target BSP Directory" window: Generate.
4. Compile + link C program: (right click on C project) > Build Project.

Note: If the execution of the *.elf program indicates that the execution time is 0, it means that the Timer driver is not properly configured. In this case, try to compile the two parallel programs by executing make in both projects directories MV_paralelo_maestro and MV_paralelo_esclavo



```
int main(){
alt_mutex_dev* mutex = altera_avalon_mutex_open("/dev/message_buffer_mutex");
int thread_count = 2;      // number of threads
int rank          = 0;      // master thread for CPU core
int Niter         = 2000;  // times that repeat matrix - vector; other values = 1000, 2000, 5000

// FORK - Syncrhonization of Separation          : *MESSAGE_BUFFER_RAM_BASE = 15
message_buffer_val         = 15;// ID=15(0xF) indicates that fork starts
message_buffer_val_join    = 0; // ID=0  indicates that memory is initialized
altera_avalon_mutex_lock(mutex,1);                        // blocks   mutex
*(message_buffer_ptr)      = message_buffer_val;         // initializes   RAM FORK
*(message_buffer_ptr_join)= message_buffer_val_join;// initializes   RAM JOIN
altera_avalon_mutex_unlock(mutex);                        // releases mutex

// Master computation   -            Matriz x Vector
int local_n         = n / thread_count;
int my_first_row = rank * local_n;              // 1ra row assigned to this core
int my_last_row  = (rank+1) * local_n - 1;   // last row assigned to this core
for (k = 0; k < Niter; k++) {
       iteraciones++;
       for (i=my_first_row; i<=my_last_row; i++){
          for(j=0; j<m; j++)     y[i] += A[i*m+j] * x[j];
       }
}

// JOIN Syncrhonization = thread unification barrier
altera_avalon_mutex_lock(mutex,1);                         /* blocks   mutex */
message_buffer_val_join = *(message_buffer_ptr_join); /* reads value in  RAM */
message_buffer_val_join |= 0x1;                            /* ID=1: ends thread    0 */
*(message_buffer_ptr_join) = message_buffer_val_join; /* stores the value in RAM      */
altera_avalon_mutex_unlock(mutex);                         /* releases mutex */
while( (message_buffer_val != 6) ){
       altera_avalon_mutex_lock(mutex,1);                 /* blocks   mutex */
       message_buffer_val = *(message_buffer_ptr);       /* reads value in  RAM */
       message_buffer_val_join = *(message_buffer_ptr_join);
       altera_avalon_mutex_unlock(mutex);                 /* releases mutex */
       if ( (message_buffer_val_join == 0x3 && thread_count == 2 ) ||
        (message_buffer_val_join == 0x1 && thread_count == 1) ){
              dumy = 6;
              altera_avalon_mutex_lock(mutex,1);  /* blocks   mutex */
              *(message_buffer_ptr) = dumy;        /* writes value in buffer         */
              altera_avalon_mutex_unlock(mutex);  /* releases mutex */
              message_buffer_val = dumy;
       }
}
} // main()
```

Figure 22: MV_paralelo_maestro.c



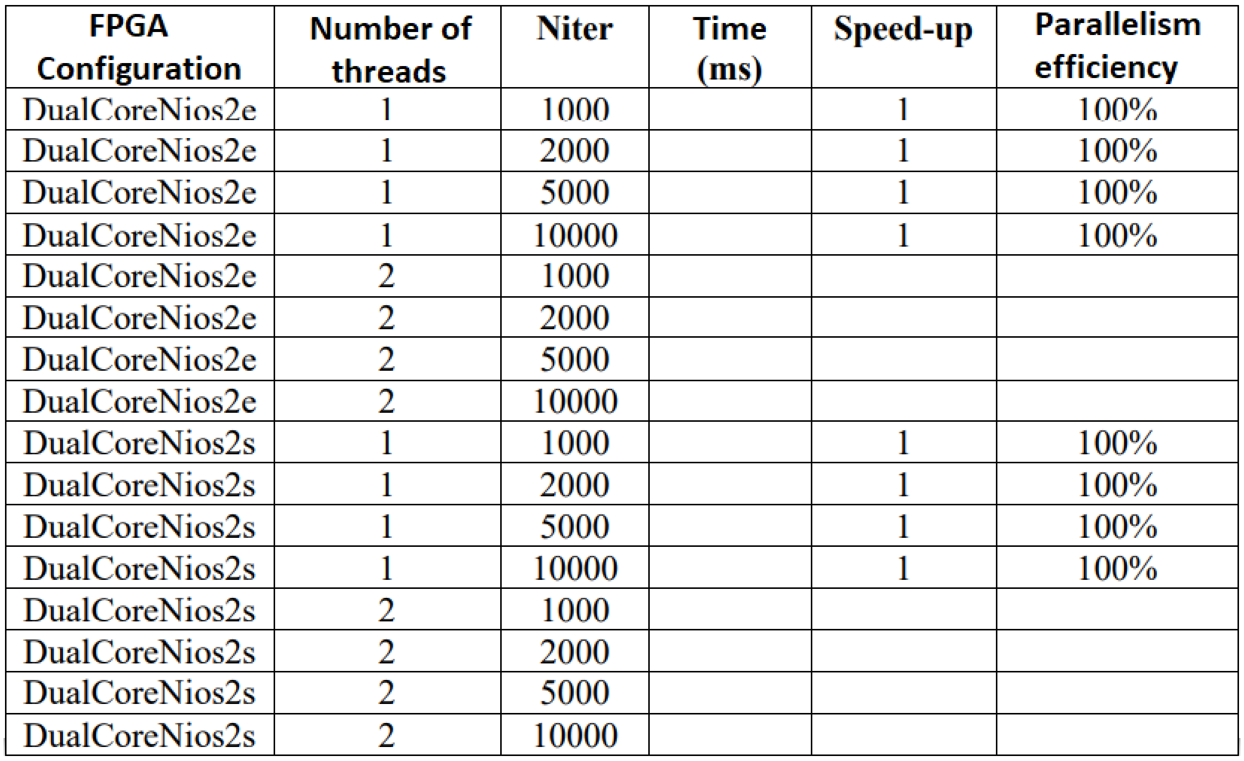
```
int main(){
alt_mutex_dev* mutex = altera_avalon_mutex_open("/dev/message_buffer_mutex");
int thread_count = 2;         // Number of threads
int rank           = 1;         // Master thread for CPU2 core

// FORK of thread 1    ,  Syncrhonization from thread 0        , message_buffer_val=15
while(message_buffer_val != 15) {
        altera_avalon_mutex_lock(mutex,2);              /* blocks      mutex */
        message_buffer_val = *(message_buffer_ptr); /* reads value in buffer          */
        altera_avalon_mutex_unlock(mutex);              /* releases   mutex */
}

// Slave Computation     -             matrix-vector
int local_n         = n / thread_count;
int my_first_row = rank * local_n;           // 1™ row assigned to this core
int my_last_row  = (rank+1) * local_n - 1;   // last row assigned to this core
for (k = 0; k < Niter; k++) {
       for (i=my_first_row; i<=my_last_row; i++){
          for(j=0; j<m; j++)    y[i] += A[i*m+j] * x[j];    } }

// JOIN -  threads unification
altera_avalon_mutex_lock(mutex,2);
message_buffer_val_join = *(message_buffer_ptr_join);
message_buffer_val_join |= 0x2; /* ID=2  thread 1 syncrhonization       */
*(message_buffer_ptr_join) = message_buffer_val_join;
altera_avalon_mutex_unlock(mutex);

} // main()
```

Figure 23: MV_paralelo_esclavo.c

Next, carry out the following activities:

1. Run with DualCoreNios2e (dual core: 2 x Nios II/e)
2. Record times activating a single thread, using the multiprocessor's CPU core for 4 workloads. In this case, it is only necessary to run the nios2-download command once. The parameters that change for the different executions are the following:
    - thread_count = 1
    - Niter = 1000, 2000, 5000, 10000.
3. Record times activating two threads, using the CPU and CPU2 cores of the multiprocessor for 4 workloads. Parameters:
    - thread_count = 2
    - Niter = 1000, 2000, 5000, 10000.
4. Performance evaluation: carry out Table 3, writing down the execution times obtained in steps 2 and 3 above. Additionally, calculate the speed-up considering that the reference configuration is a Nios II/e processor core.

   Table 3. Performance evaluation of the Matrix x Vector algorithm for 1 and 2 threads using the CPU and CPU2 processor cores of two Nios II multiprocessors.



FPGA Configuration	Number of threads	Niter	Time (ms)	Speed-up	Parallelism efficiency
DualCoreNios2e	1	1000		1	100%
DualCoreNios2e	1	2000		1	100%
DualCoreNios2e	1	5000		1	100%
DualCoreNios2e	1	10000		1	100%
DualCoreNios2e	2	1000			
DualCoreNios2e	2	2000			
DualCoreNios2e	2	5000			
DualCoreNios2e	2	10000			
DualCoreNios2s	1	1000		1	100%
DualCoreNios2s	1	2000		1	100%
DualCoreNios2s	1	5000		1	100%
DualCoreNios2s	1	10000		1	100%
DualCoreNios2s	2	1000			
DualCoreNios2s	2	2000			
DualCoreNios2s	2	5000			
DualCoreNios2s	2	10000			

5. Repeat the results with DualCoreNios2s (dual core: 2 x Nios II/s). In the performance analysis, consider the reference configuration to be a Nios II/s processor core.

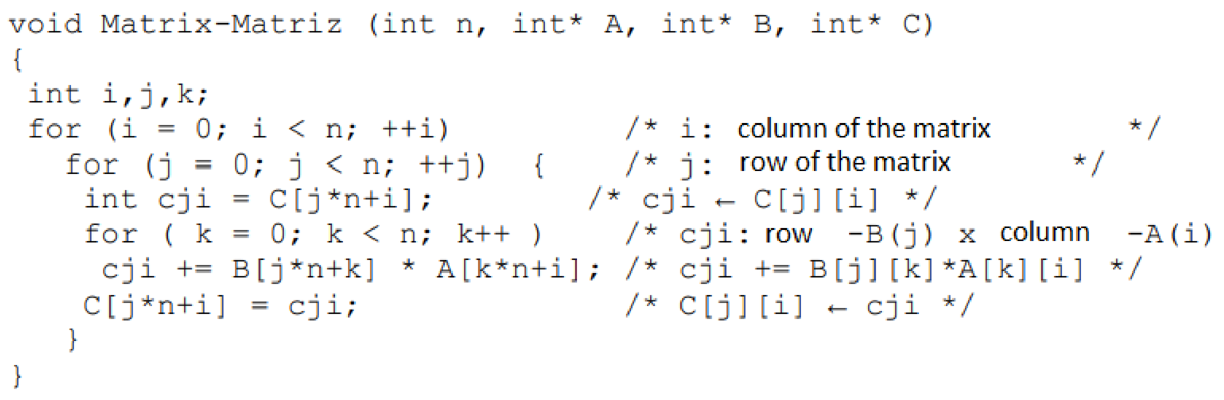
Questions to justify the results:

1. Are the results similar to Objective 3-1? Why? Reasonably justify results, including analysis of parallelism efficiency (100% x speed-up / number of threads)

Objective 3-3:

Create, run, and evaluate performance of a Matrix x Matrix (C[] = B[] . A[]) benchmark that performs matrix multiplication. The steps to be performed are as follows:

1. Encode matrix multiplication of size n rows x n columns with 4-byte integer data. The input matrices are called A and B, and the result matrix is called C: C = B x A (see Figure 24). To do this, you need to make two source programs that will be used to generate the programs for the master and slave threads. In each program you need:
   - Initialize the pointers to the start of each matrix: A, B, C.
   - Assign the workload to each core or thread.
   - Encode FORK and JOIN synchronization events.



```
void Matrix-Matriz (int n, int* A, int* B, int* C)
{
 int i,j,k;
 for (i = 0; i < n; ++i)          /* i: column of the matrix       */
   for (j = 0; j < n; ++j)  {     /* j: row of the matrix        */
    int cji = C[j*n+i];          /* cji ← C[j][i] */
    for ( k = 0; k < n; k++ )     /* cji: row  -B(j) x column  -A(i)
     cji += B[j*n+k] * A[k*n+i]; /* cji += B[j][k]*A[k][i] */
    C[j*n+i] = cji;               /* C[j][i] ← cji */
   }
}
```

Figure 24: Procedure for Matrix (B) x Matrix (A) multiplication.

2. Run with DualCoreNios2e (2 cores: 2 x Nios II/e) and DualCoreNios2s (2 cores: 2 x Nios II/s) multiprocessors. Parameters:
   - thread_count = 1, 2.
   - Niter = 10, 20, 50, 100.
3. Evaluación del desempeño: realice una tabla similar a la Tabla 3 para el algoritmo Matrix x Matrix en la que se muestren los resultados del tiempo de ejecución de los programas, la aceleración y la eficiencia del paralelismo.

Questions to justify the results:

1. Are the results similar to Objective 3-2? Why? Reasonably justify the results, including the analysis of the parallelism efficiency (100% x speed-up / number of threads).
2. Deliver the following material related to Objective 3-3: two C files of the parallel version corresponding to the master and slave threads, a table with performance evaluation results.

Practice Files for Part 3:

- C source code: MV_paralelo_maestro.c, MV_paralelo_esclavo.c (folder: Tutorial3).
- FPGA configuration files: DE0_Nano_DualCoreNios2e.sopcinfo, DE0_Nano_DualCoreNios2e.sof (folder: DualCoreNios2e); DE0_Nano_DualCoreNios2s.sopcinfo, DE0_Nano_DualCoreNios2s.sof (folder: DualCoreNios2s).

Bibliographic references

References

[1] Altera. Quartus ii handbook version 9.0 volume 5: Embedded peripherals, 2009.

[2] Altera. Creating multiprocessor nios ii systems - tutorial. `https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/tt/tt_nios2_multiprocessor_tutorial.pdf`, 2011.

[3] Altera. Nios ii hardware development tutorial, 2011.

[4] Altera. My first nios ii software - tutorial. `https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/tt/tt_my_first_nios_sw.pdf`, 2012.

[5] Altera. Basic computer system for the altera DE0 Nano board. ftp://ftp.intel.com/pub/fpgaup/pub/Intel_Material/12.0/Computer_Systems/DE0/DE0_Basic_Computer.pdf, 2013. Accessed: 2022-01-19.

[6] Intel. Embedded design handbook. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/edh_ed_handbook.pdf, 2020.

[7] Intel. Embedded peripherals ip user guide. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_embedded_ip.pdf, 2021.

[8] Intel. Nios ii software developer's handbook. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/n2sw_nii5v2gen2.pdf, 2021.