

Práctica 4

Implementación, programación paralela y evaluación de prestaciones de multiprocesadores Nios II (versión 2022/2023)

Introducción

Los principales objetivos de esta Práctica 4 son los siguientes:

- Implementación de programas paralelos en dos multiprocesadores Nios II de doble núcleo con el propósito de reducir significativamente el tiempo de ejecución de los programas.
- Registrar y comparar los tiempos de ejecución de los programas paralelos respecto a las correspondientes versiones secuenciales utilizando la placa DE0-Nano.
- Evaluar las prestaciones de los multiprocesadores Nios II para distintos volúmenes de datos procesados.
- Comparar las prestaciones de los distintos multiprocesadores Nios II.
- Implementar los programas paralelos en la placa DE0-Nano.

La metodología de programación de los multiprocesadores de doble núcleo que usaremos en esta práctica se denomina *programación multihilos*. Consiste en la realización de dos programas que se sincronizan usando variables compartidas en el espacio de direccionamiento de los procesadores además de un mecanismo hardware de exclusión mutua denominado *semáforo*.

Este guion de práctica consta de cuatro partes que se resumen a continuación.

Parte 1. Se introducirán las herramientas Software Building Tools y Command Shell de Intel-Altera para el desarrollo de programas en lenguaje C que se ejecutan en procesadores Nios II.

Parte 2. Se realizarán dos tutoriales de programación en C usando las herramientas mencionadas en la Parte 1 y se ejecutarán en la placa DE0-Nano que se encuentra en uno de los laboratorios de la ULPGC (ver Figura 1). Para ello, cada estudiante se conectará a un ordenador del laboratorio y ejecutará la máquina virtual “Altera” donde se encuentran las herramientas necesarias para interactuar con la mencionada placa. El primer tutorial se ejecuta en un solo procesador y el segundo en dos procesadores de forma paralela.

Figura 1. Placa DE0-Nano.



Arquitectura de Computadores – Práctica 4

Parte 3. Se realizará un tutorial que consiste en la implementación del algoritmo Matriz \times Vector y, además, se analizarán las prestaciones de este programa en dos multiprocesadores de doble núcleo basados en Nios II: $2 \times$ Nios II/e, $2 \times$ Nios II/s. Por último, se propone un ejercicio para evaluar las prestaciones de un algoritmo de multiplicación de matrices.

Parte 4 (opcional). Se implementarán programas paralelos de los algoritmos Matriz \times Vector y Matriz \times Matriz utilizando procesadores Nios II/f.

El material informático que acompaña a esta práctica se encuentra en las siguientes carpetas (ver la sección Moodle de la UPGC correspondiente a la Práctica 4):

- Tutorial1
- Tutorial2
- Tutorial3
- DualCoreNios2e
- DualCoreNios2s
- DualCoreNios2f
- guion

Parte 1. Infraestructura software-hardware de la práctica

Descripción general: en esta parte de la práctica se describen las herramientas software que se utilizarán para desarrollar programas en el lenguaje C, así como la microarquitectura de los multiprocesadores Nios II que se configurarán en la placa DE0-Nano, en la cual se ejecutarán los programas C.

Software Build Tools (SBT) para Eclipse: desarrollo de programas en C

La herramienta SBT consiste en una interfaz de usuario que automatiza la compilación de programas en C para la arquitectura Nios II (ver Figura 2). Integra un editor de textos, un depurador de programas y un programador de dispositivos FPGA.

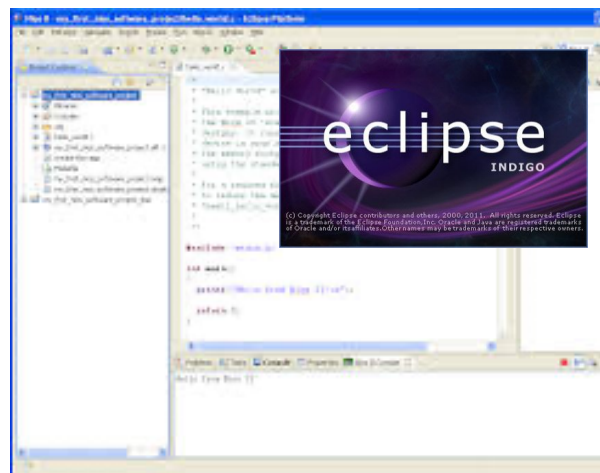


Figura 2. Interfaz de usuario de SBT (Software Build Tools) para Eclipse.

Arquitectura de Computadores – Práctica 4

Nios II Command Shell: configuración de la placa, carga y ejecución de programas

Con la herramienta software *Command Shell* se pueden crear, modificar, compilar y ejecutar programas para procesadores Nios II (ver Figura 3). Para ello, se escriben los comandos a través de teclado en la ventana Command Shell o se insertan en un fichero script y luego se ejecuta el fichero si tiene permisos de ejecución. Los comandos que se utilizarán en esta práctica son los siguientes:

```
$ nios2-configure <fichero>.sof
```

Usa un fichero *sof* para configurar la FPGA de la placa DE0-Nano. Los ficheros *.sof* que se utilizan en esta práctica fueron generados utilizando la herramienta de desarrollo hardware denominada Altera Quartus II [1,3,5,6]. En esta práctica se usan tres configuraciones que permiten disponer de tres multiprocesadores de doble núcleo basados en procesadores Nios II.

```
$ nios2-download -r -g -i <número núcleo> <fichero>.elf
```

Carga un programa compilado *elf* en la memoria principal del procesador Nios II que se encuentra físicamente en la placa DE0-Nano.

```
$ nios2-terminal
```

Visualiza en la ventana los resultados de las sentencias *printf* que se ejecutan en los programas C.

Figura 3. Ventana del Command Shell con la línea de comandos para introducir comandos que permiten interactuar con la placa DE0-Nano.



Multiprocesador Nios II con arquitectura paralela de memoria compartida

Dentro del circuito FPGA de la placa DE0-Nano se configura en esta práctica un multiprocesador integrado por dos núcleos procesadores Nios II que se denominan **CPU** y **CPU2** respectivamente. Adicionalmente, la FPGA integra los siguientes elementos hardware con los que los programas interactuarán:

- Un dispositivo semáforo para operaciones atómicas de 8 bytes (MUTEX). Asegura que una determinada variable es accedida por sólo uno de los procesadores en un instante de tiempo determinado. Por ello, se le denomina dispositivo de *exclusión mutua*. El dispositivo MUTEX se utiliza para sincronizar la ejecución de los varios hilos en los que se puede dividir un programa paralelo.
- Una memoria SRAM para almacenar variables del semáforo cuya capacidad de almacenamiento es de 1 KiB (BUFFER).

Arquitectura de Computadores – Práctica 4

Una característica de este multiprocesador es que ambos núcleos procesadores comparten los mismos dispositivos físicos de memoria RAM y por tanto sus espacios de direccionamiento. Por esta razón, la arquitectura del multiprocesador se denomina de *memoria compartida*. Adicionalmente, el multiprocesador puede utilizarse para ejecutar varios programas en paralelo. Por ello la arquitectura se denomina también *paralela*.

Otro elemento de la placa DE0-Nano que es necesario para el desarrollo de esta práctica es una memoria SDRAM cuya capacidad de almacenamiento es de 32 MiB. En la Figura 4 se muestra un diagrama de la microarquitectura del multiprocesador Nios II. Esta microarquitectura se fundamenta en la microarquitectura *Basic Computer System* que fue utilizada en las Prácticas 1 y 3 [2].

Se manejarán tres configuraciones distintas de multiprocesador Nios II que se diferencian en el procesador. Una de ellas, denominada **DualCoreNios2e**, integra dos copias del núcleo procesador Nios II/e. Otra configuración, denominada **DualCoreNios2s**, integra dos copias del núcleo procesador Nios II/s. El procesador Nios II/s incluye una cache de instrucciones de 1 KiB con bloques de 32 bytes. La tercera configuración se denomina **DualCoreNiosf_dCache4K** e integra dos procesadores Nios II/f que disponen de una cache de instrucciones de 1 KiB y una cache de datos de 4 KiB, ambas con bloques de 32 bytes.

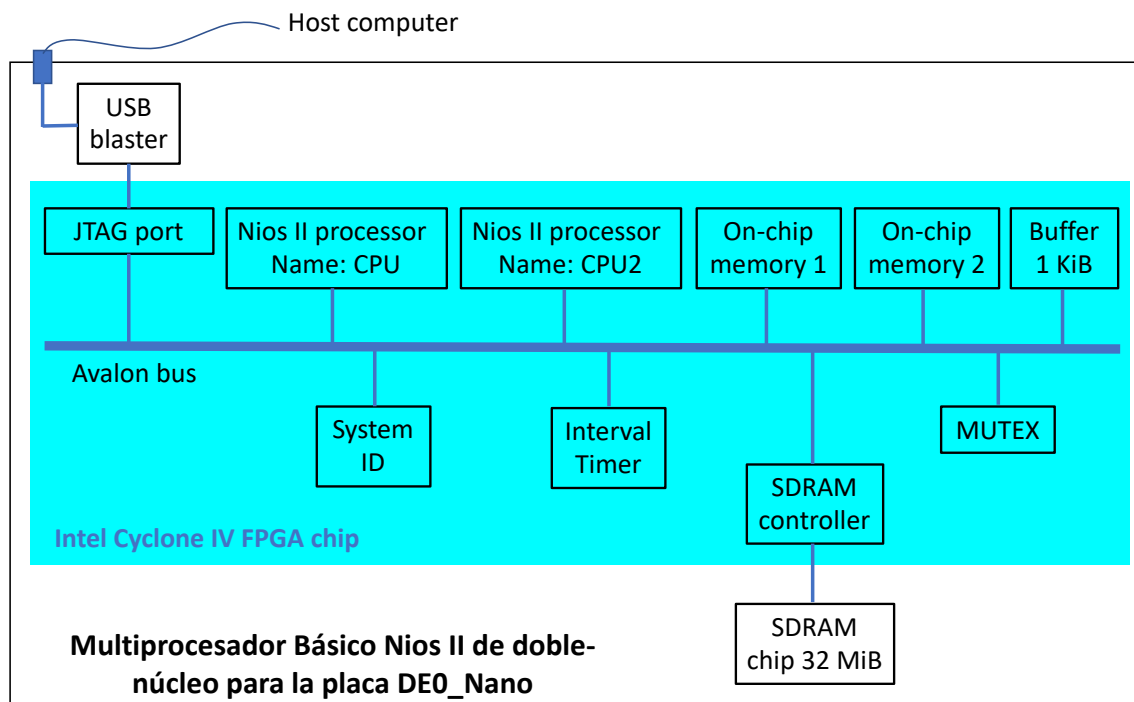


Figura 4. Multiprocesador Nios II de doble núcleo con arquitectura paralela de memoria compartida. En esta práctica se manejan tres configuraciones FPGA que disponen de un multiprocesador Nios II, denominadas DualCoreNios2e, DualCoreNios2s y DualCoreNios2f_dCache4K.

Organización de la memoria principal de los procesadores Nios II

Cuando se crea un sistema multiprocesador, el software de los múltiples procesadores se ejecuta usando el mismo dispositivo físico de memoria SDRAM. Cada procesador tiene

Arquitectura de Computadores – Práctica 4

asociado un software que debe estar localizado en una región de memoria que no es compartida por ningún otro procesador pero que reside en el mismo dispositivo físico y por tanto, el mismo espacio de direccionamiento.

La herramienta de desarrollo software SBT proporciona un particionado de la memoria que permite a varios procesadores ejecutar su software desde diferentes regiones del mismo dispositivo físico de memoria. Adicionalmente, SBT asegura que el software de los procesadores esté **enlazado (linkado)**, determinando el lugar correcto de la memoria donde residen las distintas partes en las que se divide este software. Para ello, SBT utiliza la **dirección de excepciones** para calcular la zona de memoria asignada al programa de cada procesador.

Cada procesador dispone de cinco **zonas de enlazado (linkado)** que son las siguientes (ver Figura 5):

- .text — donde reside el código ejecutable
- .rodata — donde se encuentra cualquier dato que sea de solo lectura y usado en la ejecución del código
- .rwdata — donde se encuentran las variables de lectura-escritura y los punteros de memoria
- .heap — donde reside la memoria dinámica
- .stack — donde se encuentran los parámetros que se manejan en las llamadas a funciones y subrutinas además de otros datos temporales

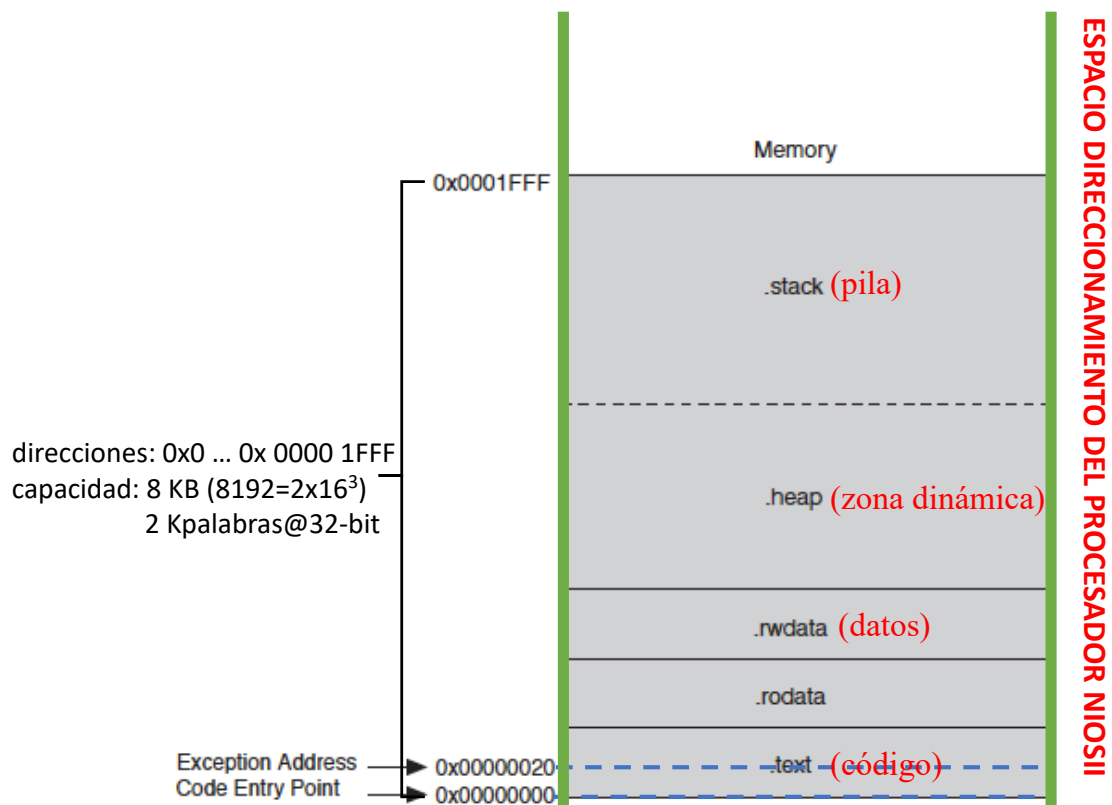


Figura 5. Ejemplo de zonas de enlazado (linkado) de la memoria principal para 1-core.

La herramienta SBT asegura que estas secciones de memoria estén localizadas en direcciones fijas de memoria.

En la Figura 5 se puede observar un mapa de memoria que muestra cómo estas secciones de linkado se asignan a direcciones fijas para un único procesador. En esta figura se ha supuesto que el software del procesador requiere de ocho kilobytes (8 KiB) de memoria. Por ello, el procesador usa la región comprendida entre las direcciones 0x0 y 0x1FFF. En la Figura 5 también se puede observar el particionado de la memoria principal del procesador en varias zonas de linkado: código (.text), datos (.rodata, .rdata), pila (.stack) y zona dinámica (.heap). En la zona de código se incluye la dirección de excepciones 0x0000 0020.

En un sistema multiprocesador, sería ventajoso usar un único dispositivo de memoria para almacenar todas las secciones del código de cada núcleo procesador. En nuestro caso, la dirección de excepciones de cada procesador se usa para definir las fronteras entre el final de las secciones de código de un procesador y el principio del siguiente núcleo procesador.

Por ejemplo, en el caso de esta práctica, la memoria principal SDRAM ocupa 32 MiB, entre las direcciones 0x0000 0000 y 0x01FF FFFF. Dentro de este rango se establecen los rangos de direcciones asignados a cada uno de los dos núcleos Nios II, denominados *CPU* y *CPU2*, que se disponen en las configuraciones FPGA mencionadas anteriormente: DualCoreNios2e, DualCoreNios2s y DualCoreNios2f_dCache4K. En la Figura 6 se puede observar que los rangos de direcciones asignados a los núcleos procesadores son:

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

Cada núcleo tiene asignado 4 MiB de memoria principal para ejecutar su software. La herramienta SBT particiona automáticamente la memoria principal fijándose en la dirección de excepciones. Para ambos núcleos del multiprocesador, esta dirección tiene los valores 0x0000 0020 para CPU y 0x0040 0020 para CPU2. Observa la Figura 6 donde se muestra la asignación de direcciones de los dos núcleos.

Los seis bits menos significativos de la dirección de excepciones son siempre 0x20. En la dirección cuyos seis bits menos significativos son 0x00 está el denominado **punto de entrada**, donde el procesador Nios II espera encontrar el **código de reset**. El desplazamiento de dirección 0x20 se debe a que la línea de la cache de instrucciones tiene una extensión es de 32 bytes (8 instrucciones de 32-bits). Estos 32 bytes corresponden a la extensión del código de reset, la cual termina saltando la zona de código de manejo de excepciones para posteriormente empezar a ejecutar el código principal de un determinado núcleo procesador.

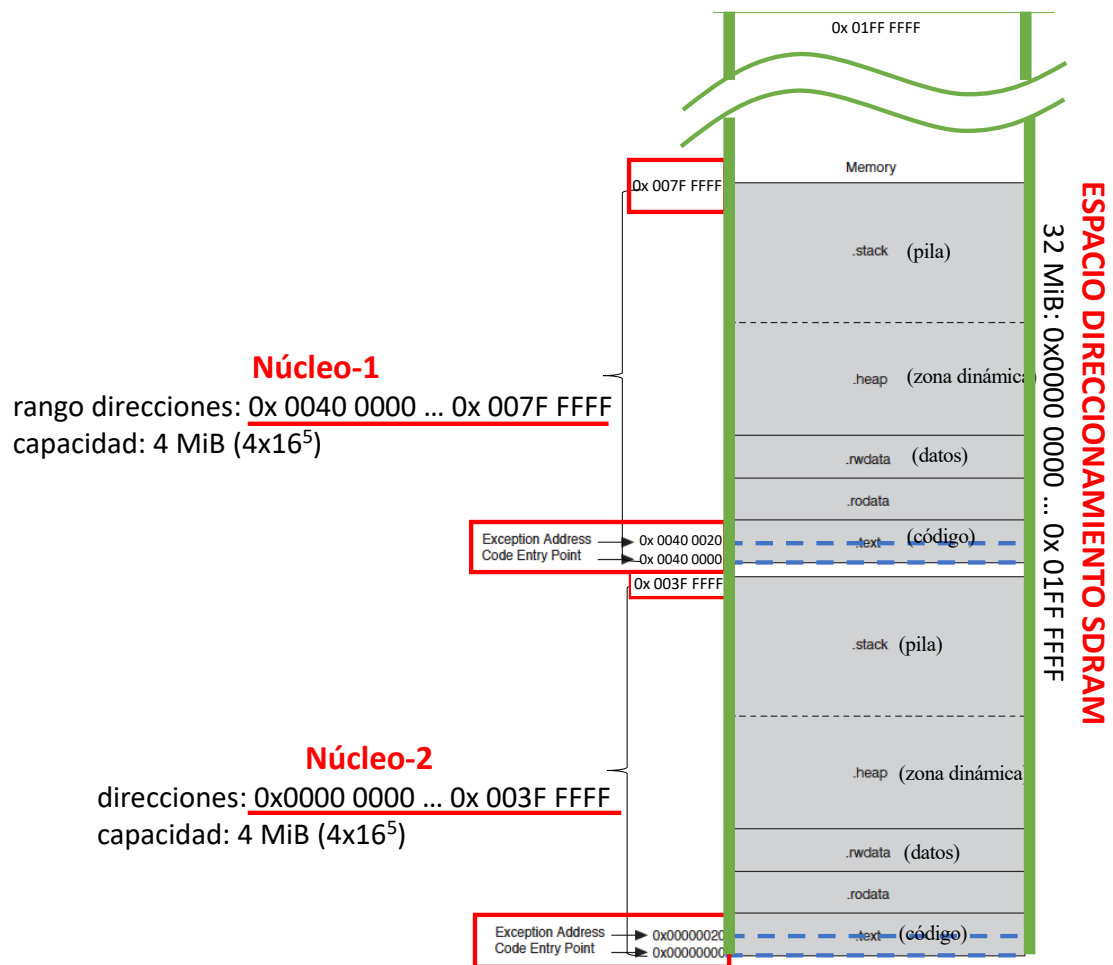


Figura 6. Zonas de linkado de la memoria principal para los multiprocesadores Nios II de doble núcleo que se manejan en esta práctica.

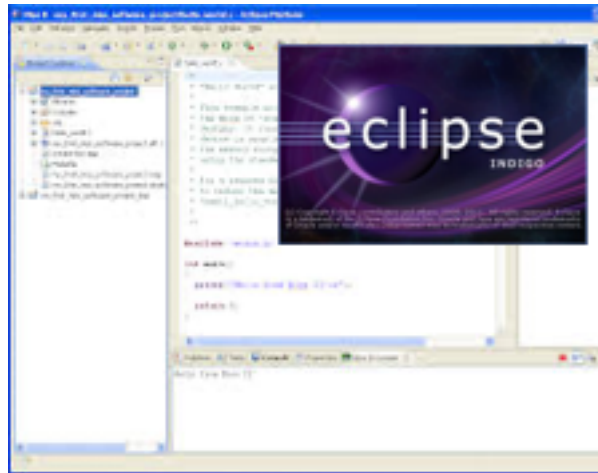
Parte 2. Tutoriales 1 y 2 para la programación multihilo del multiprocesador Nios II

Descripción general. En esta parte se implementan dos programas C (también llamados *hilos*) en el multiprocesador Nios II de la configuración DualCoreNios2e usando las herramientas SBT y Command Shell [4,7]. Estos dos programas o hilos, entre otras tareas, ordenan que los núcleos CPU y CPU2 acceden a una misma posición de memoria principal de forma **concurrente**. Esto significa que los procesadores nunca acceden al mismo tiempo, sino en tiempos distintos. El dispositivo MUTEX permite coordinar el acceso concurrente a la zona de memoria compartida por ambos núcleos procesadores [8].

Objetivo 2-1. Realizar los pasos que se mencionan a continuación para ejecutar el Tutorial-1 en el que sólo se usa un núcleo procesador (CPU).

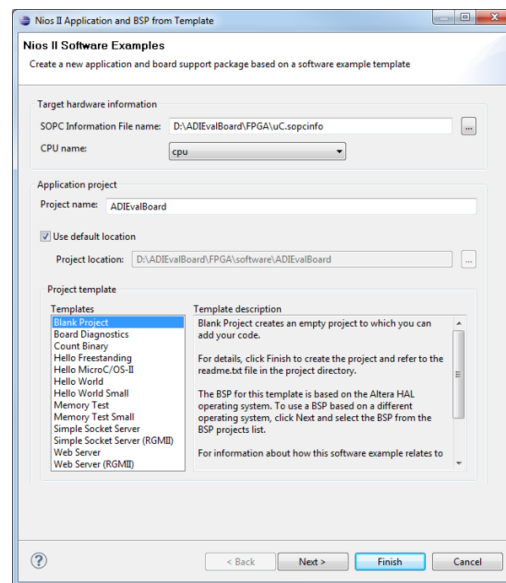
1. Abrir: Nios II – Software Build Tools - Eclipse (ver Figura 7).

Figura 7. SBT para Eclipse.



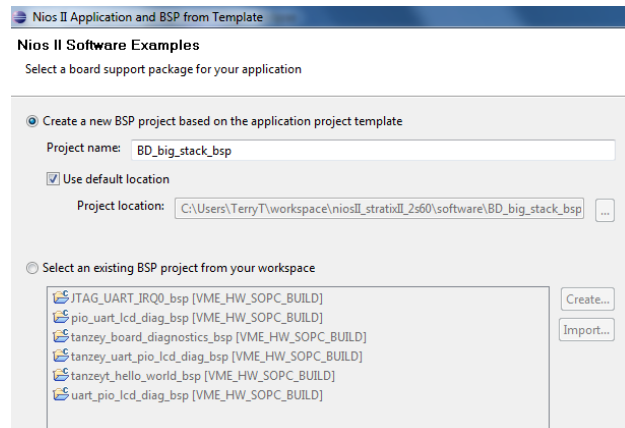
2. File > New > Nios II app & BSP from Template (ver Figura 8).
 - SOPC Information File name: DE0_Nano_DualCoreNios2e.sopcinfo
 - CPU name: CPU (también aparece CPU2 pero no se selecciona)
 - Project name: hola_chicos_y_chicas_0
 - Project Location: elegir <ProjectDir>
 - Templates: **seleccionar “Hello World Small”**
 - Picar en “Next >”

Figura 8. Ayuda para la generación de programas C con SBT.



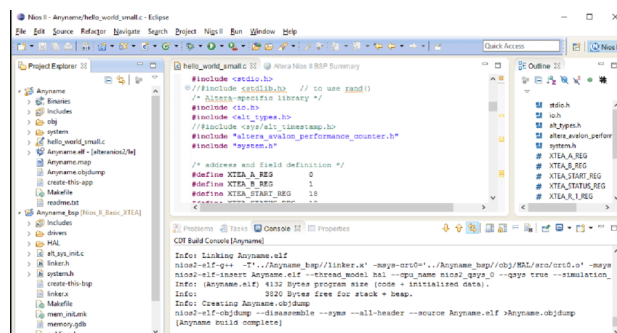
3. Seleccionar “Create BSP”, Project name: hola_chicos_y_chicas_0.bsp, Use default location, picar en “Finish” (ver Figura 9).
 - Resultado: en la ventana Eclipse “Project Explorer” se crean un proyecto de tipo “app C/C++” y otro proyecto “BSP” (Board Support Package).

Figura 9. Ventana donde se define el proyecto BSP (Board Support Package).



4. Desplegar el proyecto hola_chicos_y_chicas_0 en la ventana “Project Explorer” y picar 2 veces en hello_world.c (ver Figura 10).
 - Resultado: se abre una ventana con el código fuente.

Figura 10. Ventana donde se visualiza el código fuente en el lenguaje de programación C.



5. Sustituir el contenido del fichero hello_world.c del directorio del proyecto por el contenido del fichero Tutorial1/hola_chicos_y_chicas.c (ver Figura 11).

Figura 11. Código fuente en C del Tutorial 1. Fichero: hola_chicos_y_chicas.c.

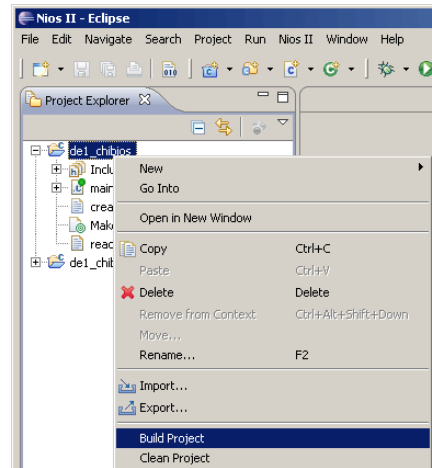
```
#include <stdio.h>

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

    return 0;
}
```

6. Compilar y linkar (ver Figura 12).
 - Picar botón derecho sobre proyecto C/C++ en la ventana Project Explorer.
 - Seleccionar: Build Project.
 - Resultado: si el proceso termina bien, aparece en la ventana inferior “Console” el mensaje: "hola_chicos_y_chicas_0 Build complete". Se crean dos proyectos en la ventana Project Explorer y dos carpetas en <ProjectDir>.

Figura 12. Ventana donde se visualiza el comando de compilación y linkado del código C.



7. Configuración de la placa DE0-Nano (ver Figura 13):

- Conectar placa al conector USB (ya está conectada en el Laboratorio)
- Abrir: Inicio > Altera > Nios II Command Shell
- Se carga fichero DE0_Nano_Basic_Computer.sof en placa :
 1. \$ cd <directorio con las configuraciones FPGA de la Práctica 4>
 2. \$ nios2-configure-sof DE0_Nano_DualCoreNios2e.sof (intentarlo varias veces si da error de JTAG)
- Resultado: tiene que aparecer el mensaje “Quartus II 32-bit Programmer was successful” (ver Figura 13).

Figura 13. Resultado del comando \$ nios2-configure-sof para configurar la FPGA de la placa DE0-Nano.

```
/cygdrive/c/Users/dbenitez/Desktop/ACpendrive/software
dbenitez@portatilaceri0p /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/Desktop/ACpendrive/DE0_Nano_DualCoreNios2e.sof
Info (213045): Using programming cable "USB-Blaster [USB-0]"
Info (213011): Programming file C:/Users/dbenitez/Desktop/ACpendrive/DE0_Nano_DualCoreNios2e.sof with checksum 0x004CBAD3 for device EP4CE22F17E1
Info (207060): Started Programmer operation at Mon Aug 31 18:52:48 2020
Info (207016): Configuring device index 1
Info (207017): Device 1 contains JTAG ID code 0x020F30DD
Info (207007): Configuration succeeded -- 1 device(s) configured
Info (207011): Successfully performed operation(s)
Info (207061): 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@portatilaceri0p /cygdrive/c/Users/dbenitez/Desktop/ACpendrive/software
$
```

8. Ejecución del programa (ver Figura 14):

- \$ cd <directorio del proyecto hola_chicos_y_chicas_0>
- \$ nios2-download -r -g -i 0 hola_chicos_y_chicas_0.elf

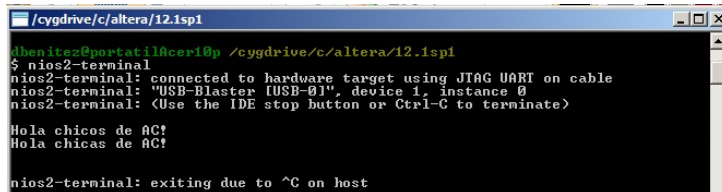
Figura 14. Resultado del comando \$ nios2-download para cargar el programa ejecutable en la memoria SDRAM de la placa DE0-Nano.

```
/cygdrive/c/Users/dbenitez/Desktop/ACpendrive/software
dbenitez@portatilaceri0p /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 0x000001B0
```

9. Visualizar resultados (ver Figura 15):

- \$ nios2-terminal
 1. Resultado: aparecerán los mensajes en la ventana correspondientes a los printf del código fuente
- Cerrar ventana de Command Shell

Figura 15. Visualización de los mensajes del Tutorial-1 después de ejecutar el comando \$ nios2-terminal en una nueva ventana Command Shell.



```
/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
```

Ficheros de la práctica para Tutorial-1:

- Código fuente C:hola_chicos_y_chicas.c (carpeta: Tutorial1).
- Ficheros de configuración de la FPGA: DE0_Nano_DualCoreNios2e.sopcinfo, DE0_Nano_DualCoreNios2e.sof (carpeta: DualCoreNios2e).

Objetivo 2-2. Realizar los pasos que se mencionan a continuación para ejecutar el Tutorial 2 en el que se usan dos núcleos procesadores (CPU y CPU2). Se crean **dos proyectos** siguiendo los pasos del Tutorial-1.

1. Elementos comunes de ambos proyectos:
 - SOPC Information File name: DE0_Nano_DualCoreNios2e.sopcinfo
 - Project Location: elegir <ProjectDir>
 - Templates: **seleccionar “Hello World Small”**
2. Proyecto 1:
 - CPU name: CPU (también aparece CPU2 pero no se selecciona)
 - Project name: hola_semaforo_0
 - Proyecto BSP, Project name: hola_chicos_y_chicas_0.bsp
 - Sustituir el contenido del fichero hello_world.c del directorio del proyecto por el contenido del fichero Tutorial2/hola_semaforo_0.c (ver Figura 16). Observar que el núcleo procesador CPU sólo lee la variable message_buffer_val. La visualización de resultados en CPU se realiza con la función printf(). Solo CPU está conectado de la interfaz JTAG por lo que su programa es el único que puede tener printf.
 - Compilar y linkar.

Figura 16. Código fuente en C (hola_semaforo_0.c) para el núcleo procesador 0 (CPU) del Tutorial-2.

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

int main(){

    // dirección de memoria de message buffer: 0x 0400 0000
    volatile int * message_buffer_ptr = (int *) MESSAGE_BUFFER_RAM_BASE;

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

    /* se guarda el manejador del dispositivo hardware de tipo mutex*/
    alt_mutex_dev* mutex = altera_avalon_mutex_open("/dev/message_buffer_mutex");

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

    while(1) {
        iteraciones++;
        /* CPU pide ser propietario de mutex, asignando el valor 1 */
        altera_avalon_mutex_lock(mutex,1);
        message_buffer_val = *(message_buffer_ptr); /* lee valor guardado en buffer */
        altera_avalon_mutex_unlock(mutex);          /* libera mutex */
        printf("CPU - iter: %i - message_buffer_val: %08X\n",
              iteraciones, message_buffer_val);
        usleep(1000000);                             /* espera 1 seg = 10^6 useg */
    }
    return 0;
}
```

3. Proyecto 2:

- CPU name: CPU2
- Project name: hola_semaforo_1
- Proyecto BSP, Project name: hola_chicos_y_chicas_1.bsp
- Sustituir el contenido del fichero hello_world.c del directorio del proyecto por el contenido del fichero Tutorial2/hola_semaforo_1.c (ver Figura 17). Observar que el núcleo procesador CPU2 modifica la variable message_buffer_val.
- Compilar y linkar

Figura 17. Código fuente en C (hola_semaforo_1.c) para el núcleo procesador 1 (CPU2) del Tutorial-2.

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

int main(){

    // dirección de memoria de message buffer: 0x 400 0000
    volatile int * message_buffer_ptr = (int *) MESSAGE_BUFFER_RAM_BASE;

    /* se guarda el manejador del dispositivo hardware de tipo mutex*/
    alt_mutex_dev* mutex = altera_avalon_mutex_open("/dev/message_buffer_mutex");

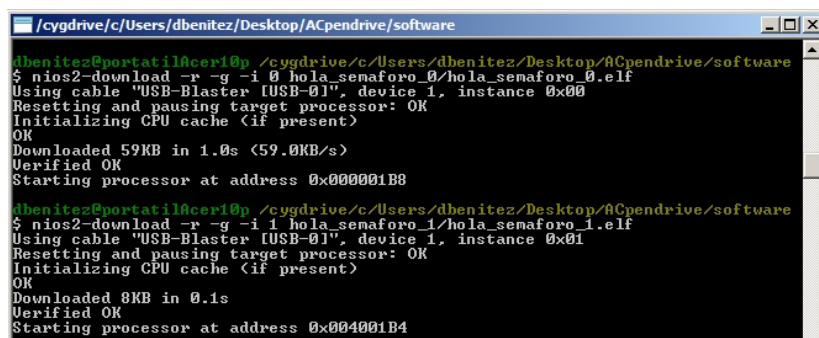
    int message_buffer_val      = 0x0;

    while(1) {
        /* CPU pide ser propietario de mutex, asignando el valor 2 */
        altera_avalon_mutex_lock(mutex,2);
        /* guarda en buffer el valor modificado en CPU2 */
        *(message_buffer_ptr) = message_buffer_val;
        altera_avalon_mutex_unlock(mutex); /* libera mutex */
        message_buffer_val++;
    }

    return 0;
}
```

4. Configuración placa DE0-Nano:
 - Conectar placa a conector USB (ya está conectada en el Laboratorio)
 - Abrir: Inicio > Altera > NiosII Command Shell
 - Se carga fichero DE0_Nano_Basic_Computer.sof en placa:
 - ❑ \$ cd <directorio con las configuraciones FPGA de la Práctica 4>
 - ❑ \$ nios2-configure-sof DE0_Nano_DualCoreNios2e.sof (intentarlo varias veces si da error de JTAG)
 - Resultado: tiene que aparecer el mensaje “Quartus II 32-bit Programmer was successful”.
5. Ejecución del programa (ver Figura 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

Figura 18. Resultado de los comandos \$ nios2-download para cargar los programas ejecutables en la memoria SDRAM de la placa DE0-Nano.

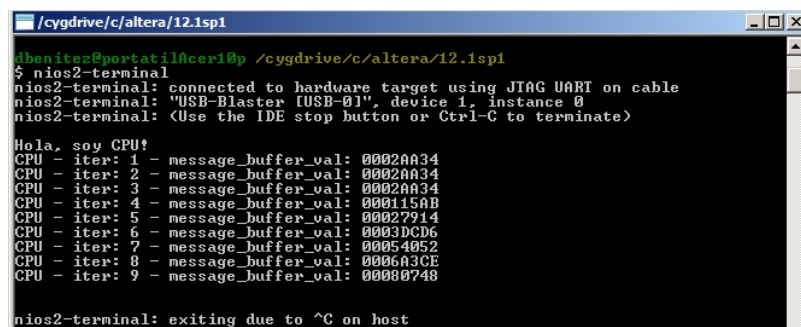


```
/cygdrive/c/Users/dbenitez/Desktop/ACpendrive/software
dbenitez@portatil0ceri0p /cygdrive/c/Users/dbenitez/Desktop/ACpendrive/software
$ nios2-download -r -g -i 0 hola_semaforo_0/hola_semaforo_0.elf
Using cable "USB-Blaster [USB-0]", device 1, instance 0x00
Resetting and pausing target processor: OK
Initializing CPU cache (if present)
OK
Downloaded 59KB in 1.0s (59.0KB/s)
Verified OK
Starting processor at address 0x000001B8

dbenitez@portatil0ceri0p /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
```

6. Visualizar resultados (ver Figura 19):
 - \$ nios2-terminal
 - \$ CTRL-C

Figura 19. Visualización de los mensajes del Tutorial-2 después de ejecutar el comando \$ nios2-terminal en una nueva ventana Command Shell.



```
/cygdrive/c/altera/12.1sp1
dbenitez@portatil0ceri0p /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
```

Ficheros de la práctica para Tutorial-2:

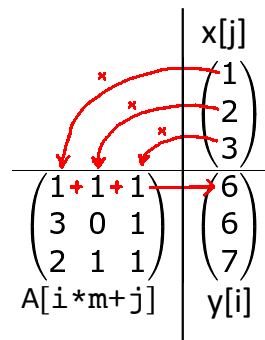
- Código fuente C: hola_semaforo_0.c, hola_semaforo_1.c (carpeta: Tutorial2).
- Ficheros de configuración de la FPGA: DE0_Nano_DualCoreNios2e.sopcinfo, DE0_Nano_DualCoreNios2e.sof (carpeta: DualCoreNios2e).

Parte 3. Programación paralela multihilos y evaluación de prestaciones de multiprocesadores Nios II de doble núcleo

Descripción general: en esta parte de la práctica se realiza el Tutorial-3 donde se implementa el algoritmo de multiplicación Matriz \times Vector. Este algoritmo se ejecutará en un solo núcleo procesador y en dos multiprocesadores basados en Nios II/e y Nios II/s, respectivamente. Finalmente, se propone un ejercicio donde se implementa el algoritmo de multiplicación de matrices.

El código C del Tutorial-3 consiste en un bucle de `Niter` iteraciones en el que cada iteración se multiplica una matriz de $n=16$ filas \times $m=16$ columnas ($A[i*m+j]$) y un vector de 16 componentes ($x[j]$). El resultado consiste en un vector de 16 componentes ($y[i]$): $y[i] = A[i*m+j] \times x[j]$, $i=0, \dots, n-1$. Los elementos de la matriz y de los vectores tienen valores enteros. La operación matemática se representa de la siguiente forma (ver Figura 20): $y = A \times x$.

Figura 20. Algoritmo de multiplicación Matriz \times Vector: $y[i] = A[i*m+j] \times x[j]$, $i=0, \dots, n-1$.



El bucle principal en C del Tutorial-3 es el siguiente (ver Figura 21):

```
for (i=0; i<=n-1; i++) {
    for(j=0; j<m; j++) y[i] += A[i*m+j] * x[j]; }

int main(){
    // zona de memoria compartida para matriz y vectores
    volatile int * A      = (int *) 0x806000; // 16x16x4=1KiB: 0x806000 - 0x8063FF
    volatile int * x      = (int *) 0x806400; // 16x1 x4=64 B: 0x806400 - 0x80643F
    volatile int * y      = (int *) 0x806800; // 16x1 x4=64 B: 0x806800 - 0x80683F
    // COMPUTO - Operacion Matriz x Vector
    int local_n           = n;
    int my_first_row      = 0; // 1ª fila asignada a este nucleo
    int my_last_row       = local_n - 1; // ultima fila asignada a este nucleo

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

Figura 21.
MV_serie_22-
23.c.

En este tutorial se registrarán tiempos utilizando el Timer del procesador CPU y su función/driver software HAL (Hardware Abstraction Layer) denominada `alt_timestamp()`. Para configurar correctamente el registro de tiempos es necesario realizar los siguientes pasos.

Configuración del Timer del procesador denominado CPU (primer procesador del dual core):

- 1.- Generar un proyecto SBT con el fichero `DE0_Nano_DualCoreNios2e.sopcinfo` para el procesador CPU siguiendo los mismos pasos que el Tutorial 1. Se puede usar el siguiente nombre de proyecto: `MV_serie`. El proyecto software incluye un proyecto BSP.
- 2.- Editar la configuración del proyecto BSP del procesador CPU (`MV_serie_bsp`). Para ello, se necesita realizar las siguientes acciones: (botón derecho sobre el proyecto BSP) Nios II > BSP editor > `timestamp_timer` > `Value= interval_timer` > Save.
- 3.- En el editor BSP y en la pestaña "Target BSP Directory": picar en Generate.
- 4.- Compilar + linkar programa C: (botón derecho sobre proyecto C) > Build Project.

Observación: en caso de que la ejecución del programa `*.elf` indique que el tiempo de ejecución es 0, significa que el driver del Timer no está bien configurado. En este caso, intentar compilar el programa ejecutando `make` en el directorio del proyecto `MV_serie`.

Observación: no configurar el Timer del procesador denominado CPU2 (segundo procesador del dual core).

Al igual que en el Tutorial-2, la visualización de resultados solo se puede hacer a través del núcleo procesador CPU usando la función `printf()`.

Objetivo 3-1: Seguir los pasos realizados en Tutorial-1 para que junto con el fichero con código `C MV_serie_22-23.c` se genere un proyecto en SBT. A continuación, configurar la placa DE0-Nano con la configuración de la FPGA: `DualCoreNios2e.sof`, y ejecutar el **programa secuencial** en el núcleo CPU. Seguidamente, realizar las siguientes actividades:

1. Registrar tiempos de ejecución en CPU para **cuatro cargas de trabajo**.
 - Parámetros: Niter = 1000, 2000, 5000, 10000.
2. Evaluación de prestaciones: rellenar la Tabla 1, cuyas columnas representan lo siguiente:
 - Niter: número de iteraciones que se repite la multiplicación matriz \times vector.
 - Tiempo: tiempo de ejecución del programa secuencial completo.
3. Repetir los resultados para Nios II/s usando el núcleo procesador denominado CPU que está integrado en la configuración de la FPGA: `DualCoreNios2s.sof`.

Arquitectura de Computadores – Práctica 4

Tabla 1. Registros de los tiempos de ejecución del algoritmo secuencial Matriz \times Vector para uno de los núcleos procesador (CPU) de dos multiprocesadores Nios II.

Configuración FPGA / Multiprocesador	Niter	Tiempo total (ms)	Speed-up
DualCoreNios2e	1000		1
DualCoreNios2e	2000		1
DualCoreNios2e	5000		1
DualCoreNios2e	10000		1
DualCoreNios2s	1000		
DualCoreNios2s	2000		
DualCoreNios2s	5000		
DualCoreNios2s	10000		

Preguntas para justificar los resultados:

1. ¿Es razonable que el doble de iteraciones (Niter) ocasione que el tiempo de ejecución de uno de los procesadores Nios II/e del multiprocesador DualCoreNios2e sea el doble? ¿Por qué?
2. ¿Es razonable que los resultados de tiempos de ejecución con Nios II/s sean significativamente inferiores a los obtenidos con Nios II/e? ¿Por qué? ¿Cuál es el speed-up promedio proporcionado por el núcleo Nios II/s respecto al núcleo Nios II/e?

Objetivo 3-2: Seguir los pasos realizados en Tutorial-2 para que junto con los ficheros con código C `MV_paralelo_maestro_22-23.c` (ver Figura 22) y `MV_paralelo_esclavo_22-23.c` (ver Figura 23) se generen dos proyectos en SBT. A continuación, configurar la placa DE0-Nano con la configuración de la FPGA: `DualCoreNios2e.sof`, y ejecutar el programa paralelo multihilos usando los núcleos CPU y CPU2.

Utilización del Timer del procesador denominado CPU (primer procesador del dual core):

- 1.- Generar dos proyectos SBT con el fichero `DE0_Nano_DualCoreNios2e.sopcinfo`, uno para el procesador CPU y el otro para el procesador CPU2. Se pueden usar los siguientes nombres de proyectos: `MV_paralelo_maestro`, `MV_paralelo_esclavo`. Cada proyecto incluye un proyecto BSP.
- 2.- Editar la configuración del proyecto BSP del procesador CPU (`MV_paralelo_maestro_bsp`): (botón derecho sobre el proyecto BSP) Nios II > BSP editor > timestamp_timer > Value= interval_timer > Save.
- 3.- En el editor BSP > en pestaña "Target BSP Directory": Generate.
- 4.- Compilar + linkar programa C: (botón derecho sobre proyecto C) > Build Project.

Observación: en caso de que la ejecución de los programas `*.elf` indique que el tiempo de ejecución es 0, significa que el driver del Timer no está bien configurado. En este caso, intentar compilar los dos programas paralelos ejecutando `make` en cada uno de los directorios de los proyectos `MV_paralelo_maestro` y `MV_paralelo_esclavo`.

Figura 22.
MV_para-
lelo_maes-
tro_22-
23.c. Có-
digo fuente
para generar el
programa eje-
cutable corres-
pondiente al
hilo maestro.

Figura 23.
MV_para-
lelo_es-
clavo_22-
23.c. Código
fuente para ge-
nerar el pro-
grama ejecu-
table correspon-
diente al hilo
esclavo.

1. Configurar la placa DE0-Nano con la configuración `DualCoreNios2e.sof` (doble núcleo: $2 \times$ Nios II/e) utilizando el comando `nios2-configure-sof`.
2. Registrar tiempos activando **un solo hilo**, usando el núcleo CPU del multiprocesador para 4 cargas de trabajo. Para ello, es necesario ejecutar una sola vez el comando `nios2-download`. Los parámetros que cambian para las distintas ejecuciones son los siguientes:
 - `thread_count = 1`
 - `Niter = 1000, 2000, 5000, 10000`.

Arquitectura de Computadores – Práctica 4

3. Registrar tiempos activando **dos hilos**, usando los núcleos CPU y CPU2 del multiprocesador para 4 cargas de trabajo. Para ello, es necesario ejecutar dos veces el comando `nios2-download`, una vez para CPU y la otra para CPU2. Parámetros:
 - `thread_count = 2`
 - `Niter = 1000, 2000, 5000, 10000`.
4. Evaluación de prestaciones: rellenar la Tabla 2 anotando los tiempos de ejecución obtenidos en los pasos 2 y 3 anteriores. Adicionalmente, calcular el speed-up considerando que la configuración de referencia es un núcleo procesador Nios II/e. Las medidas de prestaciones que aparecen en la Tabla 2 son las siguientes:
 - Tiempo Fork: tiempo de la sincronización FORK
 - Tiempo Cómputo: tiempo invertido en el algoritmo Matriz \times Vector
 - Tiempo Join: tiempo de la sincronización JOIN
 - Tiempo Total: tiempo de ejecución del programa completo
 - $\text{Speed-Up}_{\text{Computo}} = \text{Tiempo}_{\text{Computo-1hilo}} / \text{Tiempo}_{\text{Computo-2hilo}}$
 - $\text{Speed-Up}_{\text{Total}} = \text{Tiempo}_{\text{Total-1hilo}} / \text{Tiempo}_{\text{Total-2hilo}}$
 - $\text{EficienciaParalelismo}_{\text{Computo}} = 100,0\% \times \text{Speed-Up}_{\text{Computo}} / 2,0$
 - $\text{EficienciaParalelismo}_{\text{Total}} = 100,0\% \times \text{Speed-Up}_{\text{Total}} / 2,0$
5. Repetir los resultados con la configuración `DualCoreNios2s.sof` (doble núcleo: $2 \times$ Nios II/s). A continuación, rellenar la Tabla 2 con los resultados obtenidos. En el análisis de prestaciones del multiprocesador con dos núcleos de Nios II/s considera que la configuración de referencia es un solo núcleo procesador Nios II/s.

Tabla 2. Evaluación de prestaciones del algoritmo paralelo Matriz \times Vector para 1 y 2 hilos usando los núcleos procesadores CPU y CPU2 de dos multiprocesadores Nios II.

Configuración FPGA / Multiprocesador	Hilos	Niter	Tiempo (ms)				Speed-up		Eficiencia paralelismo	
			Fork	Cómputo	Join	Total	Cómputo	Total	Cómputo	Total
DualCoreNios2e	1	1000					1	1	100%	100%
DualCoreNios2e	1	2000					1	1	100%	100%
DualCoreNios2e	1	5000					1	1	100%	100%
DualCoreNios2e	1	10000					1	1	100%	100%
DualCoreNios2e	2	1000								
DualCoreNios2e	2	2000								
DualCoreNios2e	2	5000								
DualCoreNios2e	2	10000								
DualCoreNios2s	1	1000					1	1	100%	100%
DualCoreNios2s	1	2000					1	1	100%	100%
DualCoreNios2s	1	5000					1	1	100%	100%
DualCoreNios2s	1	10000					1	1	100%	100%
DualCoreNios2s	2	1000								
DualCoreNios2s	2	2000								
DualCoreNios2s	2	5000								
DualCoreNios2s	2	10000								

Preguntas para justificar los resultados:

1. ¿Los resultados del Objetivo 3-2 son similares a los del Objetivo 3-1? ¿Por qué? Justificar los resultados razonadamente, incluyendo el análisis de la eficiencia del paralelismo ($100\% \times \text{speed-up} / \text{número de hilos}$).
2. Compara las prestaciones de los procesadores Nios II/e y Nios II/s en base a la implementación del algoritmo Matriz \times Vector. Utiliza las medidas de tiempos de ejecución, speed-up y eficiencia del paralelismo para realizar la comparación.

Objetivo 3-3: Crear, ejecutar y evaluar prestaciones de un benchmark Matriz \times Matriz ($C[] = A[] \times B[]$) que implementa el algoritmo de la multiplicación de matrices. Los pasos que se deben realizar son los siguientes:

1. Codificar un algoritmo paralelo de la multiplicación de matrices de tamaño n filas \times n columnas con datos enteros de 4 bytes, $n=m=8$. Las matrices de entrada se llaman A y B, y la matriz resultado se llama C: $C = A \times B$ (ver Figura 24). Para ello, se necesitan realizar dos programas fuente que se usarán para generar los programas ejecutables de los hilos maestro y esclavo. Cada programa fuente se compilará en el ordenador sobremesa y luego se ejecutará en un procesador Nios II distinto de la placa DE0-Nano. En cada programa se necesitan codificar las siguientes tareas:
 - Inicializar los punteros del inicio de cada matriz: A, B, C.
 - Codificar los eventos de sincronización FORK y JOIN.
 - Asignar la carga de trabajo a cada núcleo o hilo.

Figura 24.
Procedimiento
para la multi-
plicación Ma-
triz (A) x Ma-
triz (B).

```
void Matriz-Matriz (int n, int* A, int* B, int* C)
{
    int i,j,k,cij;
    for (i = 0; i < n; ++i)
        for (j = 0; j < n; ++j) {
            cij = C[i*n+j];
            for (k = 0; k < n; k++)
                cij += A[i*n+k] * B[k*n+j];
            C[i*n+j] = cij;
        }
}
```

2. Ejecutar para 1 y 2 hilos el programa paralelo con los multiprocesadores DualCoreNios2e (2 núcleos: $2 \times$ Nios II/e) y DualCoreNios2s (2 núcleos: $2 \times$ Nios II/s). Los parámetros que se deben configurar en el programa del hilo maestro son las duplas (thread, Niter) cuyos valores son los siguientes:
 - thread_count = 1, 2.
 - Niter = 1000, 2000, 5000, 10000.
3. Evaluación de prestaciones: realizar una tabla similar a la Tabla 2 para el algoritmo Matriz \times Matriz en la que se muestren los resultados del tiempo de ejecución de los programas, el speed-up y la eficiencia del paralelismo ($100\% \times \text{speed-up} / \text{número de hilos}$).

Preguntas para justificar los resultados:

1. ¿Los resultados del Objetivo 3-3 son similares a los del Objetivo 3-2? ¿Por qué? Justificar los resultados razonadamente, incluyendo el análisis de la eficiencia del paralelismo. Este análisis debe incluir la justificación de los valores obtenidos del

- speed-up y de la eficiencia del paralelismo tanto en la parte exclusivamente de cómputo de la multiplicación de matrices como globalmente.
2. Compara las prestaciones de los procesadores Nios II/e y Nios II/s en base a la implementación del algoritmo Matriz \times Matriz. Utiliza las medidas de tiempos de ejecución, speed-up y eficiencia del paralelismo para realizar la comparación.
 3. Entregar el siguiente material relacionado con el Objetivo 3-3: dos ficheros C de la versión paralela correspondientes a los hilos maestro y esclavo, una tabla con resultados de evaluación de prestaciones y un análisis de prestaciones.

Ficheros de la práctica para la Parte 3 (obtenidos a través de Moodle):

- Código fuente C: MV_serie_22-23.c, MV_paralelo_maestro_22-23.c, MV_paralelo_esclavo_22-23.c (carpeta: Tutorial3).
- Ficheros de configuración de la FPGA: DE0_Nano_DualCoreNios2e.sopcinfo, DE0_Nano_DualCoreNios2e.sof (carpeta: DualCoreNios2e); DE0_Nano_DualCoreNios2s.sopcinfo, DE0_Nano_DualCoreNios2s.sof (carpeta: DualCoreNios2s).

Objetivo 3-4 (opcional): Realizar la implementación de los algoritmos Matriz \times Vector y Matriz \times Matriz usando el multiprocesador basado en Nios II/f. Adicionalmente, evaluar las prestaciones del multiprocesador de doble núcleo Nios II/f usando los tiempos de ejecución, speed-up y eficiencia del paralelismo cuando el número de repeticiones de los algoritmos son: 1000, 2000, 5000, 10000.

Observación sobre el comportamiento del multiprocesador Nios II/f. El procesador Nios II/f incluye una memoria cache de datos de 4 KiB. Este hecho influye en que:

1. Los almacenamientos ejecutados en cualquiera de los hilos del programa paralelo se almacenen automáticamente en la memoria cache de datos del correspondiente núcleo Nios II/f, pero, sin embargo, no se asegura que se guarden los datos en la memoria principal SDRAM. Por ello, las actualizaciones realizadas por uno de los hilos no son observadas por el núcleo que ejecuta el otro hilo.
2. Adicionalmente, si un procesador tiene un valor de una variable compartida en su memoria cache que fue llevada a la cache antes que el otro procesador actualizara la misma variable compartida en la memoria principal SDRAM, los datos almacenados en la cache del otro procesador no están actualizados.

Por ambas circunstancias, es necesario forzar la actualización de las variables compartidas en la memoria SDRAM y en las correspondiente caches de datos para que ambos procesadores del multiprocesador puedan acceder a datos coherentes de dichas variables compartidas.

Procedimiento para forzar la actualización de las variables compartidas en la memoria cache de datos. Simplemente, antes de acceder a la variable compartida por parte de un determinado hilo, se fuerza el acceso a otra variable cuya dirección de memoria se encuentra a una distancia de 4K direcciones respecto a la variable compartida. Observar que la cache de datos de los procesadores Nios II/f es de 4 KiB. Este procedimiento se hace en todos los hilos, tanto del que escribe en la variable compartida como del hilo que lee dicha variable.

Arquitectura de Computadores – Práctica 4

De esta forma se fuerza a que las líneas de cache donde se encuentran los datos de la variable compartida en ambos procesadores sean desalojadas de las memorias cache de datos. Si la variable fue actualizada por un hilo en la memoria cache de datos de un procesador, el dato es actualizado en la memoria principal SDRAM.

Cuando a continuación el otro hilo (el que no ha escrito sobre la variable compartida) accede a la variable compartida, se produce fallo de cache y se copia el dato actualizado de la variable compartida desde la SDRAM a la cache de datos. De esta forma son coherentes los valores de la variable compartida tanto en las caches de datos como en la memoria principal SDRAM.

Ficheros de la práctica para el Objetivo 3-4 (obtenidos a través de Moodle):

- Ficheros de configuración de la FPGA: DualCoreNios2f_dcach4K.sopcinfo, DualCoreNios2f_dcach4K.sof (carpeta: DualCoreNios2f).

Referencias bibliográficas

- [1] Altera. Quartus II Handbook Version 9.0 Volume 5: Embedded Peripherals, Altera, 2009.
- [2] Altera. Basic Computer System for the Altera DE0-Nano Board, Altera, 2013. ftp://ftp.intel.com/pub/fpgaup/pub/Intel_Material/14.1/Computer_Systems/DE0-Nano/DE0-Nano_Basic_Computer.pdf
- [3] Altera. Creating Multiprocessor Nios II Systems - Tutorial, Altera, 2011. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/tt/tt_nios2_multiprocessor_tutorial.pdf
- [4] Altera. My First Nios II Software - Tutorial, Altera, 2012. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/tt/tt_my_first_nios_sw.pdf
- [5] Altera. Nios II Hardware Development Tutorial, Altera, 2011.
- [6] Intel. Embedded Design Handbook, Intel, 2020. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/edh_ed_handbook.pdf
- [7] Intel. Nios II Software Developer's Handbook. Intel, 2021. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/n2sw_nii5v2gen2.pdf
- [8] Intel. Embedded Peripherals IP User Guide. Intel, 2021. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_embedded_ip.pdf