	Universidad de Costa Rica Escuela de ingeniería Eléctrica Laboratorio 1	EIE Escuela de Ingeniería Eléctrica
IE0424: Laboratorio de Circuitos Digitales II-2020		

Objetivo General

Introducir las herramientas que serán utilizadas durante el curso mediante una práctica introductoria sobre la síntesis de código HDL en un FPGA.

Objetivos específicos

- Familiarizar al estudiante con el ambiente integrado de desarrollo Vivado de Xilinx.
- Presentar al estudiante una implementación en HDL de un microprocesador de la arquitectura RISC-V.
- Presentar al estudiante un ejemplo sencillo en código C para ser ejecutado en el microprocesador.

Propuesta del problema

Se debe corroborar el funcionamiento correcto de un código HDL que implementa un procesador de la arquitectura RISC-V. Se da también un código en C que debe ser compilado a la arquitectura señalada. Este código es un contador infinito que es desplegado en 8 leds en la tarjeta de desarrollo.

El estudiante debe luego de crear varios códigos, simularlos y justificar los resultados obtenidos de acuerdo con los conceptos de procesadores RISC.

Anteproyecto

- De los conceptos aprendidos en los cursos de Estructuras de Computadoras Digitales, repase y mencione los siguientes conceptos:
  - ¿Qué es pipelining?
  - ¿Cuáles son las etapas clásicas de un pipeline de un procesador RISC?
  - ¿Qué son riesgos que atentan contra el pipeline? Mencione y explique algunos.
  - ¿En qué consiste el riesgo de read after write?
- Investigue acerca de la arquitectura RISC-V. ¿Qué elementos distinguen esta arquitectura de cualquier otra arquitectura RISC? ¿Cuál es el set de instrucciones de una arquitectura RISC-V de 32 bits?
- Investigue qué es compilación cruzada (cross compiling). ¿Para qué es necesario?
- Escriba un código en ensamblador de RISC-V que haga lo siguiente:

- El código debe de escribir uno por uno los números pares de la secuencia de Fibonacci hasta el número 10946. Los números se deben de escribir en la posición de memoria 0x10000000 si solamente si son pares.

Requisitos

- Contar con Vivado previamente instalado.

Arquitectura

Durante el curso se va a utilizar una implementación de un procesador RISC-V que se llama picorv32. Como referencia, este es código fuente abierto y se puede localizar en https://github.com/cliffordwolf/picorv32.

A este procesador, le vamos a cargar código ejecutable al que durante el resto del curso le llamaremos *firmware*. Este firmware son básicamente instrucciones en el formato de la arquitectura RISC-V que le van a indicar al procesador qué hacer.

Procedimiento

Primero, es necesario construir el *toolchain* para la arquitectura RV32I. Para no compilar el toolchain desde cero, se va a utilizar un ambiente de desarrollo basado en contenedores (Investigación extra: Investigar cuáles son la diferencias entre máquinas virtuales y contenedores).

Asimismo, se usará un administrador de ambientes de desarrollo llamado V*agrant*. Para ello, es necesario instalar en su sistema los siguientes paquetes usando:

```
$ sudo apt install vagrant docker.io
$ sudo usermod -a -G docker $USER
```

Nota: El último comando agrega el usuario al grupo *docker*. Para que este efecto, proceda a hacer *logout* y luego a hacer *login* de nuevo. Asimismo, es posible que un logout no sea suficiente y se tenga que reiniciar el OS. No proceda al siguiente paso si no ve una salida al ejecutar:

```
$ groups | grep docker
```

Seguidamente, haga ingrese a su cuenta en GitHub (https://github.com/). Si no tiene cuenta, proceda a crearse una.

Proceda ahora a obtener el código inicial del proyecto. Para ello, siga el siguiente enlace:

```
https://classroom.github.com/g/sTmZjcfy
```

Autorice la aplicación, busque su nombre (si su nombre no aparece, contacte al profesor) y seleccione un equipo (si su equipo no aparece listado, proceda a crear uno). Una vez finalizado un repositorio debió haber sido creado. Este repositorio va a ser el repositorio con el que va a trabajar y va a ser uno de los entregables de este laboratorio.

Proceda a clonar el repositorio que ha sido creado:

```
$ git clone https://github.com/ucr-ie-0424/...
```

La dirección completa de su repositorio debe aparecer seleccionando el botón verde que dice *Code.*

Compile el código que se correrá en el procesador que se va a sintetizar. Para ello, levante el ambiente de desarrollo (que por debajo levantará un contenedor) y compile el código fuente en *src/firmware* para la arquitectura RISC-V de 32 bits. Esto lo puede hacer ejecutando los siguientes comandos:

```
$ cd <repositorio> # El directorio que contiene el código clonado
# El siguiente comando puede tardar varios minutos
$ docker pull jsdanielh/ie-0424:0.0.2
$ vagrant up # Inicia el contenedor
$ vagrant ssh # Ingresa al contenedor por SSH
$ cd ws/src/firmware/
$ make
```

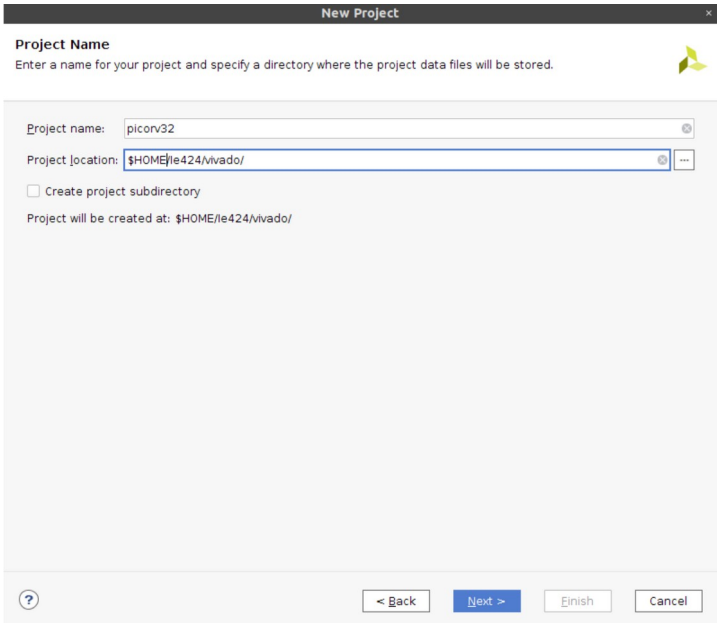
Asegúrese que los comandos anteriores no hayan presentado errores. En caso de éxito, se le debió haber generado el archivo *firmware.hex* dentro del directorio *src/firmware*.

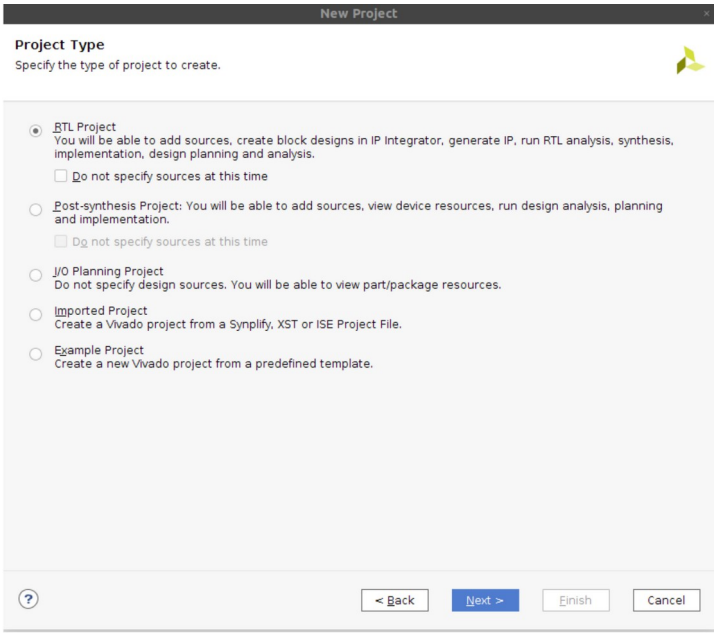
Nota: Siempre que se desee compilar *firmware*, asegúrese de estar dentro del contenedor.

Ahora proceda a iniciar Vivado y cree un nuevo proyecto. Para ello, siga los siguientes pasos:

1. Haga clock en ‘Create project’ y siga las instrucciones que se presentan:


New Project
VIVADO
HLx Editions
Create a New Vivado Project
This wizard will guide you through the creation of a new project.
To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.
XILINX
ALL PROGRAMMABLE.
< Back
Next >
Finish
Cancel


New Project
Project Name
Enter a name for your project and specify a directory where the project data files will be stored.
Project name:
picorv32
Project location:
$HOME/ie424/vivado/
Create project subdirectory
Project will be created at: $HOME/ie424/vivado/
< Back
Next >
Finish
Cancel

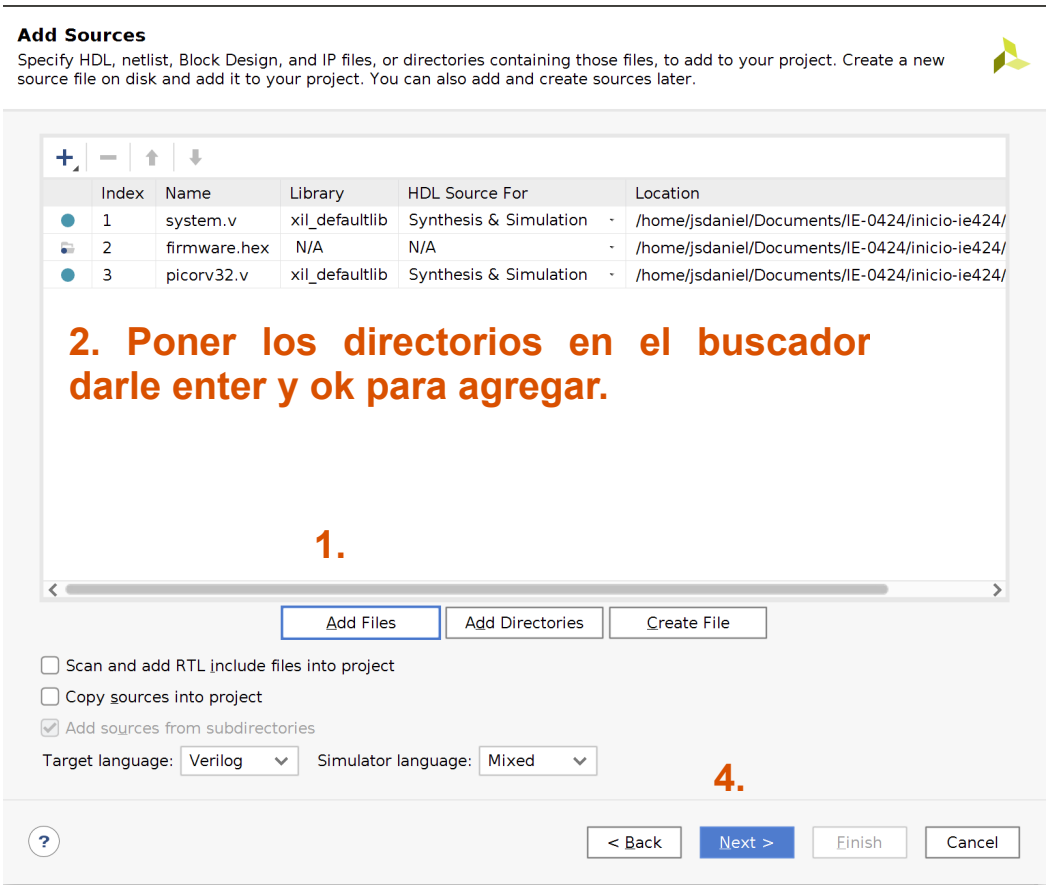


2. Agregue los siguientes archivos como fuentes:

   a) src/picorv32/picorv32.v

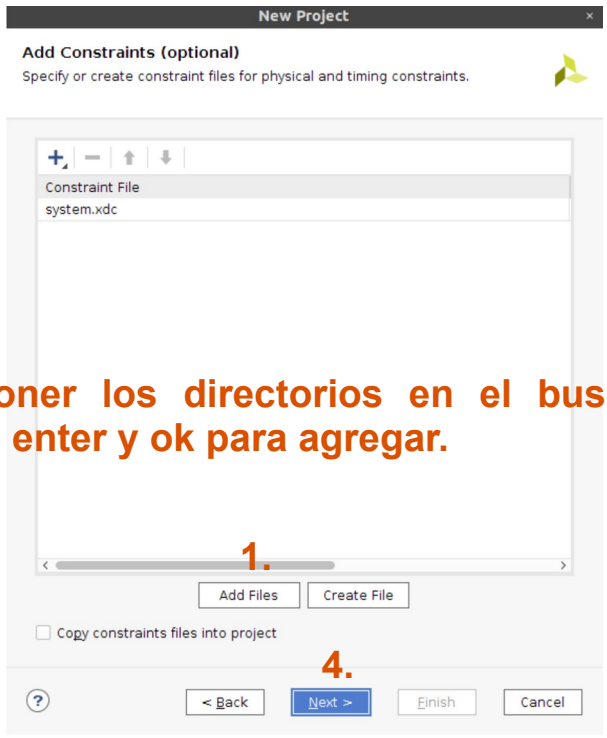
   b) src/verilog/system.v

   c) src/firmware/firmware.hex (asegúrese de tener '*All files*' en el filtro inferior)

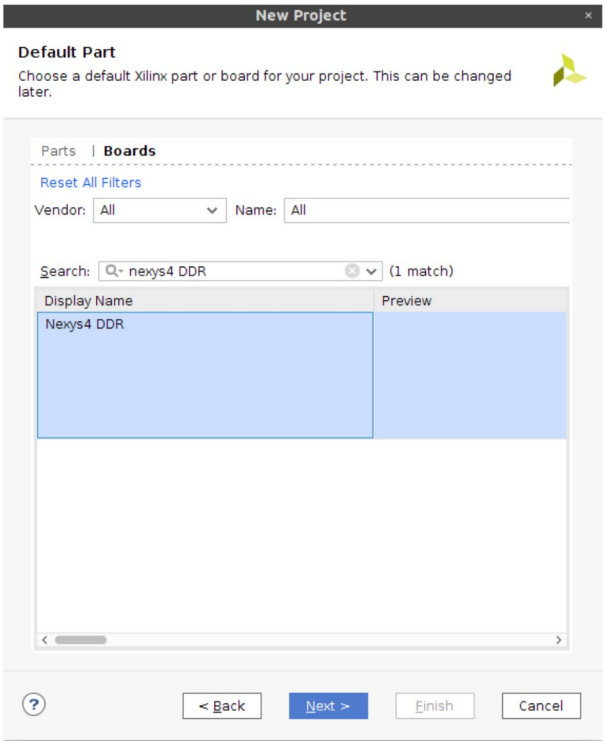


3. Agregar los siguientes archivos como constraints:

   a) src/constraints/system.xdc



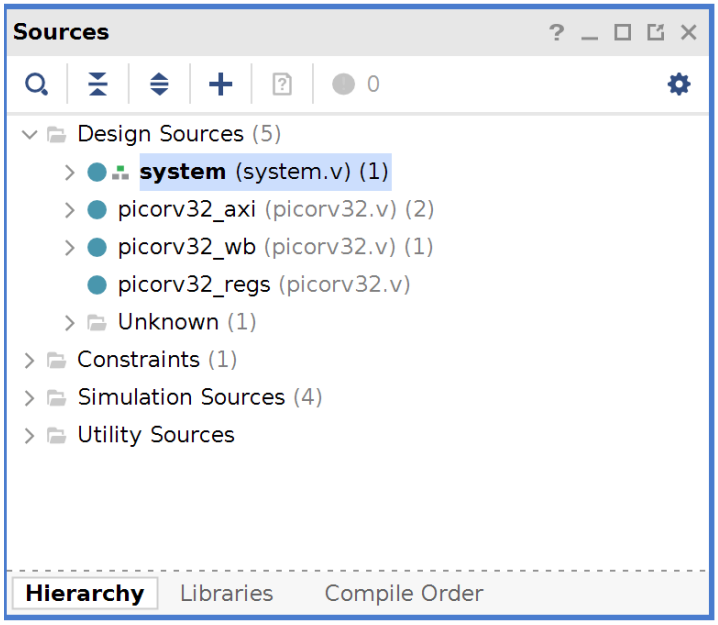
4. Seleccionar la tarjeta Nexys4 DDR como parte por defecto (asegúrese de seleccionar '*Boards*' en la parte superior)



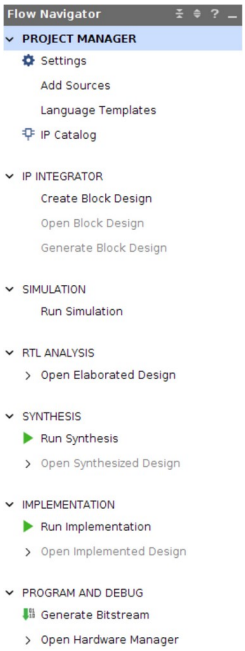
5. Continuar con las instrucciones que se le presenten.

6. Asignar el módulo *system* como top.

a) Para ello, haga click derecho sobre el módulo *system* y seleccione '*Set as Top*'.



Una vez generado el proyecto, haga click en '*Generate Bitstream*' para generar el archivo *.bit* a utilizar en la tarjeta de desarrollo.



Si se fuese a cargar el código sintetizado a la tarjeta, finalizada la generación del archivo .bit:

1. Conecte la tarjeta de desarrollo al computador.
2. Abra el 'Hardware Manager'.
3. Seleccione 'autoconectar dispositivo'.

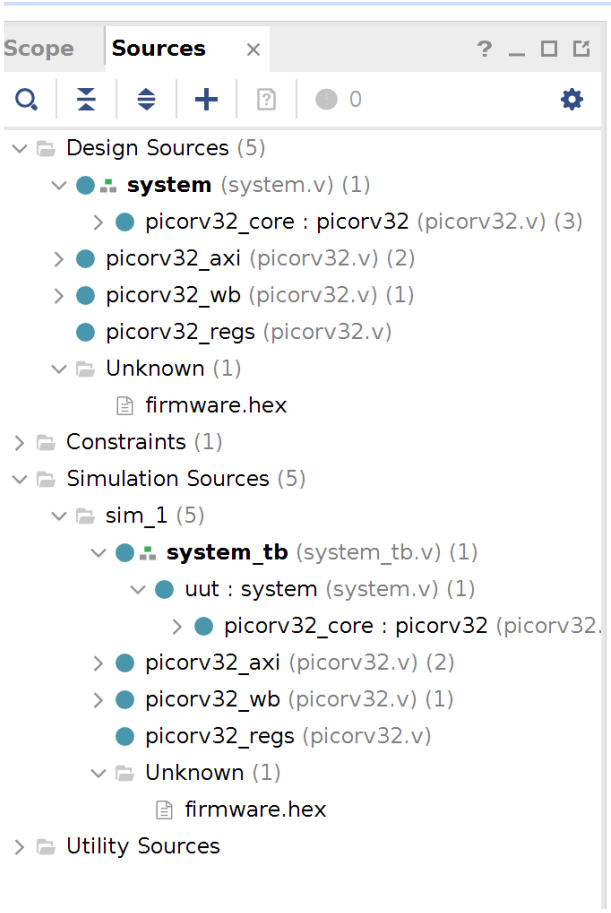
4. Seleccione ‘Programar dispositivo’.
5. Al haberse programado el dispositivo se debe aplicar la señal de reset.
    a) Esto se hace con el switch V10.
6. En caso de éxito, deberá observar un contador en 8 de los LEDs contiguos a los switches.

Note que cada vez que realiza alguna acción dentro de Vivado, en la ventana ‘Tcl Console’ se imprimen ciertos comandos y ciertas salidas. El programa se puede manejar utilizando esta consola y es la base para permitir crear el proyecto con un script (opcional).

Para hacer simulaciones, realice los siguientes pasos:

1. Agregue los siguientes archivos como archivos de simulación:

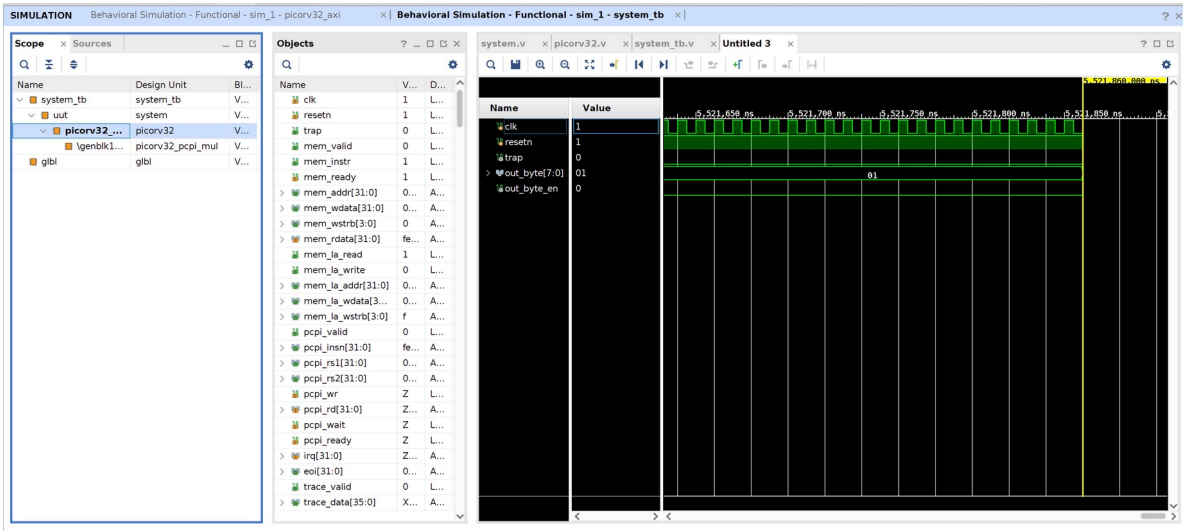
    a) src/verilog/system_tb.v



2. Procure que todos los archivos se muestren como en la imagen anterior. Si *firmware.hex* no aparece como archivo de simulación, agréguelo como archivo de simulación.
3. Luego de esto, proceda a lanzar una simulación por comportamiento.
    a) Para ello vaya al panel izquierdo bajo las opciones de ‘Simulación’.

b) Dé click en ‘Correr simulación’ y luego escoja la opción de ‘Simulación por comportamiento’.

4. En caso de éxito, la simulación se levantará y deberá ver una ventana como esta:



5. Para correr la simulación utilice los botones de control en la parte superior. Luego de correr la simulación utilice el botón de ‘Reiniciar’ y luego proceda a correr la simulación por algún tiempo. Puede usar el botón de ‘Correr’ y luego pararlo después de un tiempo o el botón de ‘Correr por X tiempo’.

6. Utilice el panel de ‘Scope’ y ‘Objects’ para navegar los diferentes módulos, registros y cables del diseño. Por ejemplo, navegue el procesador Picorv32 y agregue cables o registros de forma tal que los pueda ver en el visualizador de ondas.

Ejercicio 1 (Obligatorio):

Lance una simulación y analice lo que presenta la señal *out_byte*. Responda las siguientes preguntas:

- ¿Qué presenta la señal?
- ¿Cada cuántos ciclos de reloj cambia?
- ¿Cuál es el valor máximo que llega a tener la señal? Dé una posible razón de que ese sea el valor máximo.
- Analice el código fuente de firmware y lo que se escribe en la dirección 0x10000000 y busque similitudes con lo que ve en *out_byte*. Anote las similitudes.
    - Examine el código en system.v y anote cómo es que out_byte adquiere sus valores.

Ejercicio 2 (Obligatorio):

Escriba un segundo firmware que sea igual al del Ejercicio 1 pero que no esté enciclado por siempre (es decir, remueva la línea que dice: “*while (1) {*“ y la línea correspondiente donde se cierra el ciclo con “*}*”). Llame a este nuevo firmware *firmware_lab1_part2.c*

Compile el nuevo firmware. Para ello, refresque el enlace simbólico de *firmware.c* y compile el código de nuevo:

```
$ src/firmware
$ ln -sf firmware_lab1_part2.c firmware.c
$ make
```

Proceda a lanzar una nueva simulación con este nuevo firmware y responda las siguientes preguntas:

- ¿Qué hace terminar la simulación?
- Relacione de acuerdo con lo que ve en la señal out_byte, la parte de firmware luego de la cual se termina la simulación.
- Dé una posible razón por la cual se pueda explicar que la cause de la finalización de la simulación se haya dado.

Ejercicio 3 (Obligatorio):

Escriba un segundo firmware. Llame a este nuevo firmware *firmware_lab1_part3.c*. Este firmware deberá de escribir uno por uno los números pares de la secuencia de Fibonacci hasta el número 10946. Los números se deben de escribir en la posición de memoria 0x10000000 si solamente si son pares.

El código puede quedarse en un ciclo infinito imprimiendo los mismos números una y otra vez para que no suceda lo que vio en el ejercicio pasado.

Compile el nuevo firmware. Para ello, refresque el enlace simbólico de *firmware.c* y compile el código de nuevo:

```
$ src/firmware
$ ln -sf firmware_lab1_part3.c firmware.c
$ make
```

En este ejercicio vamos a proceder a desensamblar el firmware. Para ello podemos usar dentro del contenedor:

```
$ src/firmware
$ riscv32-unknown-elf-objdump -D firmware.elf
```

Compare el código obtenido con el que escribió en el anteproyecto:

- Compare y anote las diferencias.
- Justifique las diferencias con base en los requrimentos de este firmware.

Ejercicio 4 (Obligatorio):

Utilizando el firmware del ejercicio anterior, proceda a lanzar una simulación. Ahora agregue a su simulación las siguientes señales del CPU picorv32:

- reg_pc

- next_pc
- dbg_ascii_instr (utilice Radix en ASCII para esta señal).
- dgb_insn_opcode
- dbg_insn_rs1
- dbg_insn_rs2
- dbg_insn_rd
- cpuregs_wrdata
- cpuregs

1. Con el nombre de las señales y haciendo una comparación con el código que desensambló en el ejericio anterior, explique qué significado tienen las señales (también puede ayudarse viendo el código fuente de *picorv32)*.
2. Busque en la simulación los siguientes momentos en la simulación y justifique el valor de las señales que se especificaron anteriormente:
    1. El momento en el que se obtiene el número 144 de la secuencia de Fibonacci producto de la suma de los dos anteriores.
    2. El momento donde se escribe 144 en la dirección de memoria 0x10000000.
3. Basándose en la simulación, demuestre si la implementación del procesador sigue un diseño de pipelining o no. Justifíquelo comentando la simulación.
    1. ¿Cómo cree usted que el diseñador del CPU está evitando el riesgo de *read after write*?

Referencia

Rojas, E. (2019) *Síntesis de un procesador RISC-V en tarjetas de desarrollo Nexys 4 DDR*. San José, Costa Rica: Escuela de Ingeniería Eléctrica, Universidad de Costa Rica.