# Prácticas de Tecnología de Computadores 2010/2011

Este documento recoge el material necesario para la realización de un procesador simple llamado TG00PROC empleando el entorno Xilinx. A continuación se presenta un índice con el contenido del documento.



## 1. Introducción y objetivos

Con esta práctica se pretende que el alumno diseñe, implemente y testee un procesador simple que denominamos TG00PROC. Se proporcionarán cuatro componentes de TG00PROC ya diseñados: la ALU, un banco de registros, un registro y una memoria. Primero, el alumno deberá diseñar el ciclo de instrucción, la unidad de datos y la unidad de control de TG00PROC con lápiz y papel. Segundo, deberá implementar el diseño en Xilinx usando la herramienta *Schematic Editor*. Por último, habrá que testear el funcionamiento del diseño obtenido usando la herramienta *Simulator,* también incluida en Xilinx.

Los objetivos que se persiguen son:

- Diseñar el ciclo de instrucción
- Diseñar la unidad de datos
- Diseñar la unidad de control
- Implementar el diseño en el entorno Xilinx
- Simular o testear el funcionamiento de TG00PROC

## 2. Descarga e instalación del entorno Xilinx

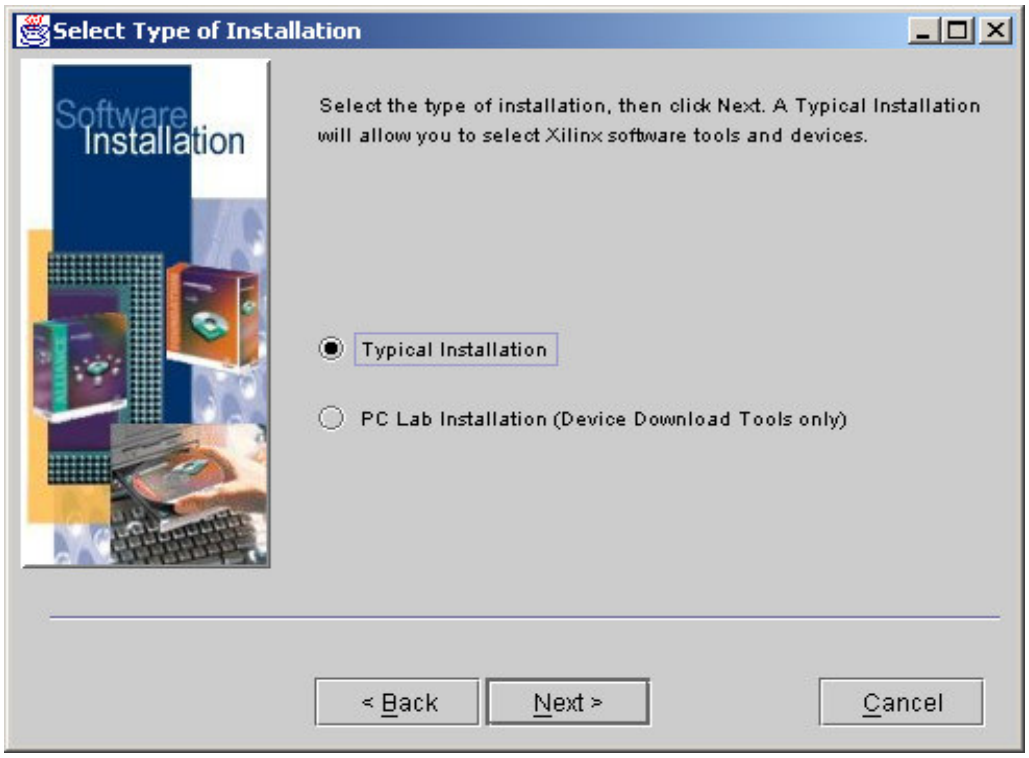
En la industria, antes de tomar la decisión de fabricar un determinado procesador u otro tipo de hardware, previamente se comprueba su buen funcionamiento usando entornos como el de Xilinx. Las herramientas incluidas en dicho software nos van a permitir:

- Crear la circuitería de TG00PROC y
- Realizar simulaciones para testear su buen funcionamiento

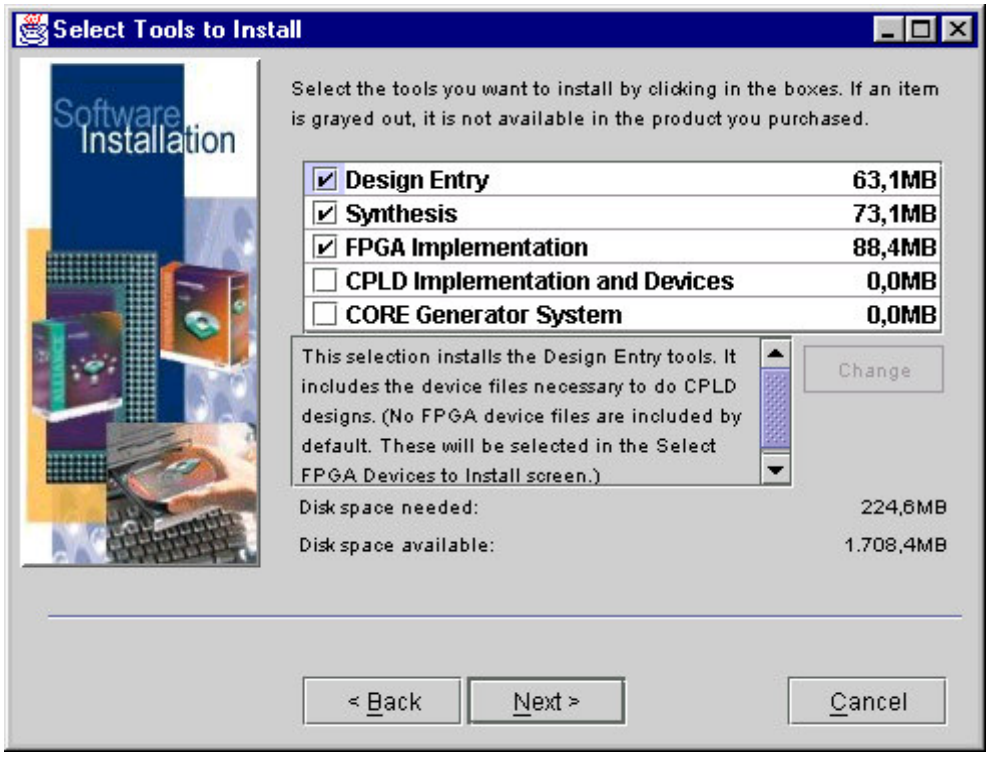
## Descarga e instalación del programa

Para utilizar Xilinx bastará con descargar la imagen ISO disponible en el Campus Virtual. Una vez descargado, habrá que realizar las siguientes etapas:

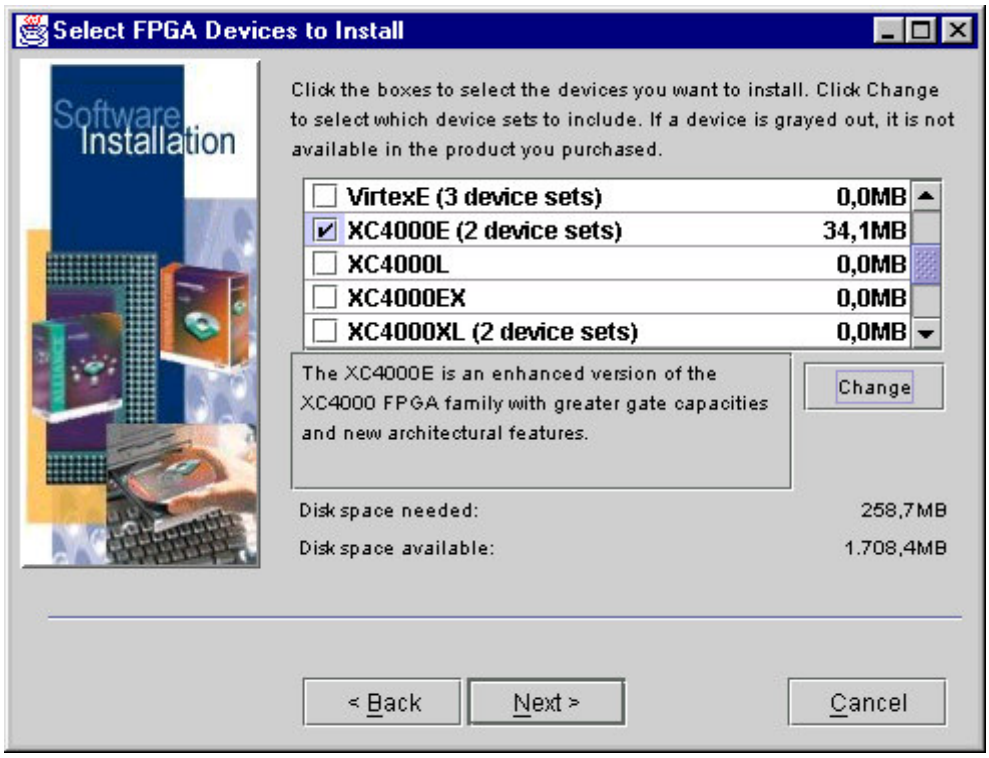
1- Al principio de la instalación deberás mantener las opciones que vienen por defecto, incluida la de la siguiente pantalla:



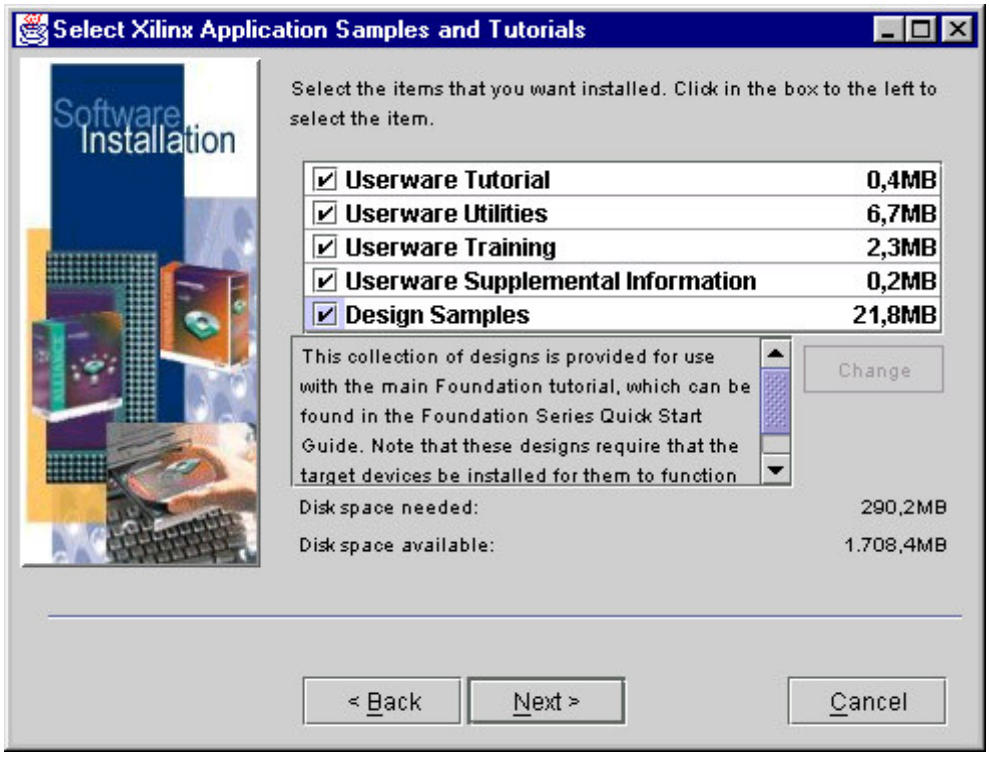

2-Tras introducir el “CD Key” y el número de serie (S/N), proporcionados a través del Campus Virtual, tendréis que seleccionar las herramientas que se necesitan para realizar las prácticas. Las opciones que te recomendamos son:



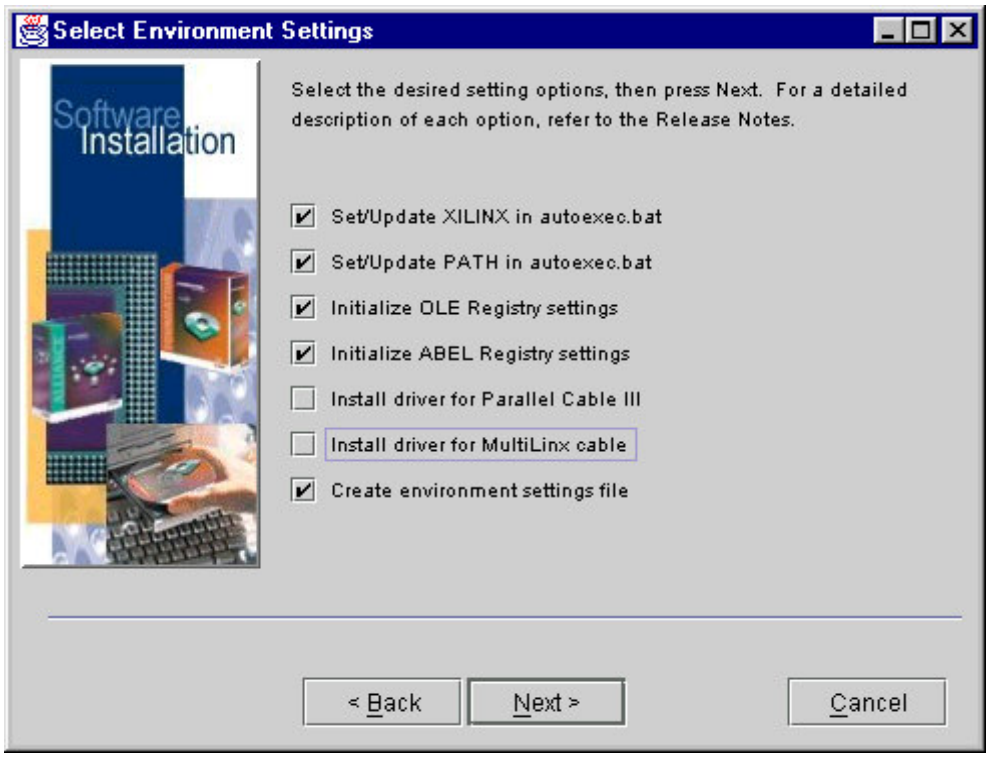

3- En el siguiente punto de la instalación debes seleccionar las librerías. El material que te proporcionamos está desarrollado con la librería XC4000E:

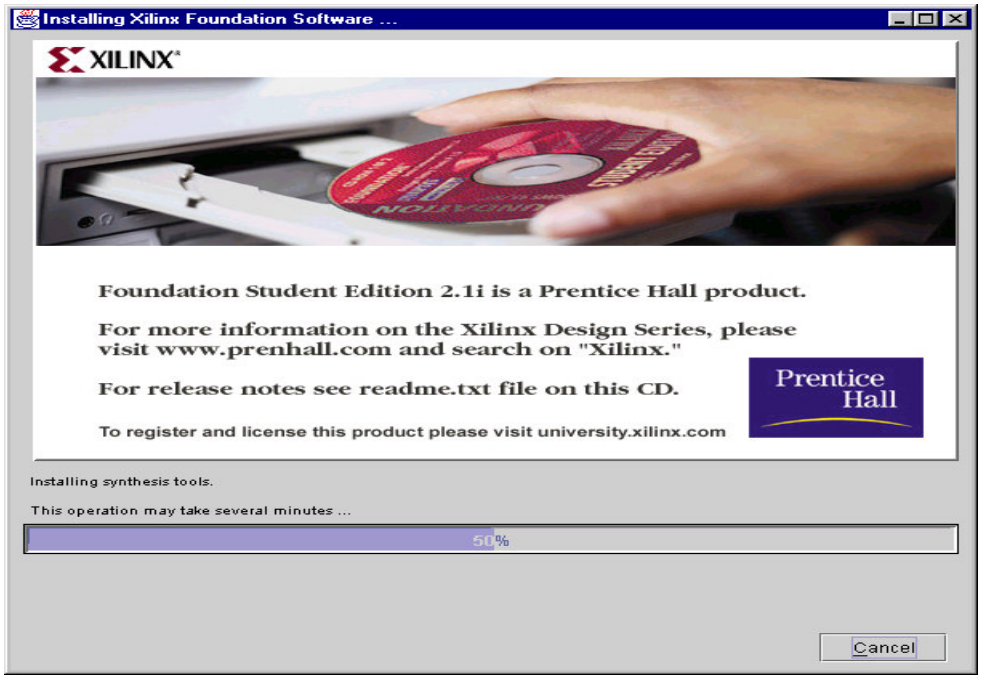


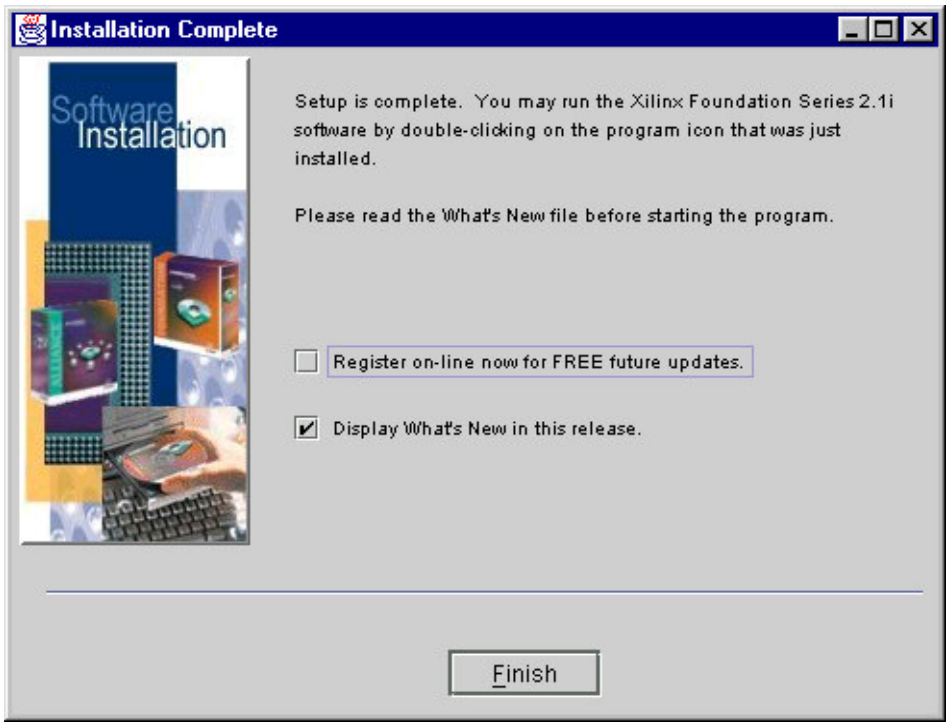

4- Es recomendable que dispongas de mayor selección de material de apoyo, así que selecciónalo todo (tutorial, utilidades, entrenamiento, ...).



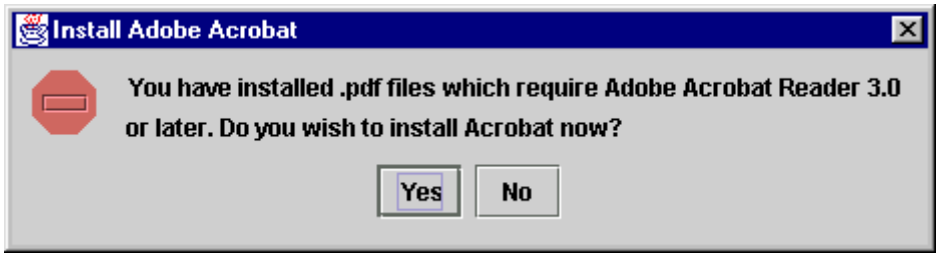

5- Como no vamos a sintetizar los diseños en un chip, eliminamos las opciones de controladores de cable:






La instalación de Xilinx incluye un visualizador para los ficheros .pdf de ayuda.




Si ya tienes instalada la versión 3.0 o superior de Acroread, elige NO.

## 3. Características del procesador TG00PROC

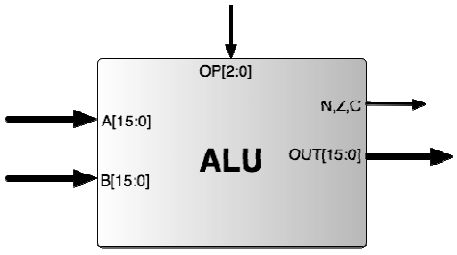
Las características principales del procesador TG00PROC son:

- Palabra de 16 bits.
- Banco de 8 registros de propósito general de 16 bits. **El registro R0 será de solo lectura y contendrá el valor 0**.
- Unidad aritmético-lógica (ALU) de 16 bits con las siguientes características:
  - Suma y resta de operandos enteros representados en convenio complemento a 2 (C2)
  - Operación AND lógica y OR lógica.
  - Desplazamiento lógico a izquierda y derecha.
  - Tres flags de estado: Z (resultado de la ALU igual a 0), N (resultado de la ALU negativo) y C (acarreo en la ALU ).
- Interfaz con la memoria de 8 bits para las direcciones (256 direcciones en total) y 16 para los datos.
- Memoria principal de 256 palabras de 16 bits distribuidas como:
  - 64 posiciones de memoria ROM (posiciones 0:63 de la memoria)
  - 192 posiciones de memoria RAM (a partir de la posición 64 de memoria)

### a. Descripción de los componentes proporcionados

Para el diseño del procesador se proporcionan ya diseñados los siguientes elementos: una ALU, un registro de 16 bits, un banco de 8 registros de 16 bits y la memoria. El alumno podrá utilizarlos y/o modificarlos si lo cree conveniente. En las siguientes figuras y tablas se muestra la forma, interfaz y funcionamiento de los elementos proporcionados.

**ALU (TG00ALU)**




| Señales | |
|---|---|
| A[15:0] | Dato de entrada |
| B[15:0] | Dato de entrada |
| OP[2:0] | Operación a realizar por la ALU |
| OUT[15:0] | Dato de salida |
| N | Se activa (1) cuando el resultado es negativo |
| Z | Se activa (1) cuando el resultado es 0 |
| C | Se activa (1) cuando hay acarreo |

| OP[2:0] | Operación ALU | |
|---|---|---|
| 000 | OP = A + B | Suma |
| 001 | OP = A – B | Resta |
| 010 | OP = A & B | And (bit a bit) |
| 011 | OP = A \| B | Or (bit a bit) |
| 100 | OP = LSHIFT(A, B posiciones) | Desplazamiento izquierda |
| 101 | OP = RSHIFT(A, B posiciones) | Desplazamiento derecha |

## REGISTRO (TG00REG16)


| Señales | |
|---|---|
| CLK | Señal de reloj para sincronizar la escritura |
| LOAD | Señal de escritura (1) |
| IN[15:0] | Bus para la entrada del dato a escribir |
| OUT[7:0] | Bus para leer el dato almacenado |

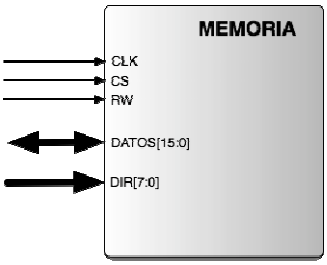
| LOAD | Operación del registro |
|---|---|
| 0 | Lectura: OUT[15:0] = Contenido almacenado en el registro |
| 1 | Escritura por flanco de bajada de CLK: registro ← IN[15:0] |

## BANCO DE REGISTROS (TG00BR)


| Señales | |
|---|---|
| CLK | Señal de reloj para sincronizar la escritura |
| W | Señal de escritura (1) |
| D[2:0] | Bus para seleccionar el registro |
| IN[15:0] | Bus para la entrada del dato a escribir |
| OUT[7:0] | Bus para leer el dato almacenado |

| W | Operación del registro |
|---|---|
| 0 | Lectura: OUT[15:0] = Contenido almacenado en el registro indicado por D[2:0] |
| 1 | Escritura por flanco de bajada de CLK: Registro indicado por D[2:0] se carga con IN[15:0] |

## MEMORIA (TG00MEM)

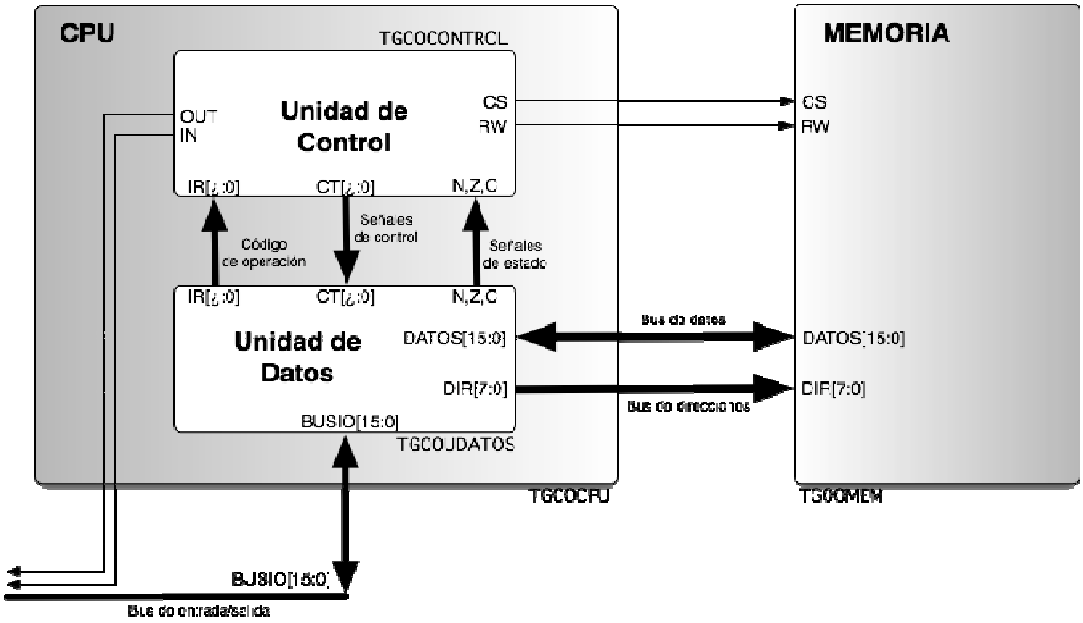



| Señales | |
|---|---|
| CLK | Señal de reloj para sincronizar las escrituras en memoria |
| CS | Señal de habilitación de la memoria (chip select) |
| RW | Señal de lectura (1) o escritura (0) en memoria |
| DATOS[15:0] | Bus de datos de entrada y salida |
| DIR[7:0] | Bus de direcciones |

| CS | RW | Operación de la memoria |
|---|---|---|
| 0 | - | Memoria no operativa, desconectada del bus de DATOS |
| 1 | 1 | Lectura: DATOS[15:0] = M(DIR[7:0]) |
| 1 | 0 | Escritura por flanco de bajada de CLK: M(DIR[7:0]) ← DATOS[15:0] |

A continuación se muestra un esquema del sistema procesador a desarrollar por el alumno.

## Esquema general del sistema a implementar




### b. Conjunto de Instrucciones de TG00PROC

El conjunto mínimo de instrucciones que debe soportar el procesador son:

- **Instrucciones de Bifurcación:**

  - Salto incondicional con direccionamiento directo:

    **J dir**

    - **dir** es un operando de 8 bits que representa la dirección de salto.
    - Acción de la instrucción: la siguiente instrucción a ejecutar se encuentra en la posición de memoria **dir**.

    **PC ← dir**

    **Formato de instrucción (1 palabra):**

| OPCODE | | | | | | | | dir | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | x | x | x | | | | | | | | |

  - Salto condicional con direccionamiento relativo al contador de programa (PC):

    **JGT ra, rb, (des)**

    - **ra** y **rb** son operandos con direccionamiento directo a registro. Hacen referencia a alguno de los 8 registros.
    - **des** es un operando de 8 bits de tipo entero (C2) que representa el desplazamiento relativo al PC.
    - Acción de la instrucción: si se cumple la condición especificada por la instrucción, la siguiente instrucción a

ejecutar se encuentra en la posición de memoria **PC+1+des.**

**Si (ra > rb) entonces PC ← PC+1+des**

**Formato de instrucción (2 palabras):**


- **<u>Instrucciones de transformación de información:</u>**

- Operaciones aritméticas con direccionamiento **inmediato**:

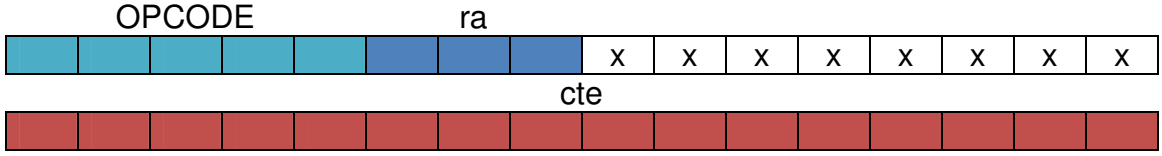
## ADDi ra,#cte  SUBi ra,#cte

  - **ra** es un operando con direccionamiento directo a registro. Hace referencia a uno de los 8 registros. El contenido del registro se debe interpretar como un número entero (C2).
  - **cte** es un operando inmediato de 16 bits representando un número entero (C2).
  - Acción de las instrucciones:

| Instrucción | Acción |
|---|---|
| **ADDi ra,#cte** (suma) | **ra←ra+cte ; carga(Z,N,C)** |
| **SUBi ra,#cte** (resta) | **ra←ra-cte ; carga(Z,N,C)** |

**Formato de instrucción (2 palabras);**

OPCODE ADDi: 00000
OPCODE SUBi: 00001




- Operaciones aritméticas con direccionamiento **directo a registro**:

## ADDd ra,rb  SUBd ra,rb

  - **ra y rb** son operandos con direccionamiento directo a registro. Hacen referencia a uno de los 8 registros. El contenido de los registros se debe interpretar como un número entero (C2).
  - Acción de las instrucciones:

| Instrucción | Acción |
|---|---|
| **ADDd ra,rb** (suma) | **ra←ra+rb ; carga(Z,N,C)** |
| **SUBd ra,rb** (resta) | **ra←ra-rb ; carga(Z,N,C)** |

**Formato de instrucción (1 palabra):**

OPCODE ADDd: 01000
OPCODE SUBd: 01001


- Operaciones aritméticas con direccionamiento **indirecto**:

## ADDn ra,(dir) SUBn ra,(dir)

- **ra** es un operando con direccionamiento directo a registro. Hacen referencia a uno de los 8 registros. El contenido del registro **ra** se debe interpretar como un número entero (C2).
- **dir** es un operando de 8 bits con direccionamiento indirecto.
- Acción de las instrucciones:

| Instrucción | Acción |
|---|---|
| **ADDd ra,(dir)** (suma) | **ra←ra+M[M[dir]] ; carga(Z,N,C)** |
| **SUBd ra,(dir)** (resta) | **ra←ra- M[M[dir]] ; carga(Z,N,C)** |

**Formato de instrucción (1 palabra):**

OPCODE ADDn: 10000
OPCODE SUBn: 10001


- Operaciones de desplazamiento con direccionamiento directo a registro:

## RSH ra LSH ra

- **ra** es un operando con direccionamiento directo a registro. Hace referencia a uno de los 8 registros.
- Acción de las instrucciones:

| Instrucción | Acción |
|---|---|
| **RSH ra** (desplazamiento lógico a la derecha) | **ra←0,ra[7:1]** (desplazamiento del contenido de **ra** un bit a la derecha, insertando 0 en el bit más significativo)**; carga(Z,N)** |
| **LSH ra** (desplazamiento lógico a la izquierda) | **ra←ra[6:0],0** (desplazamiento del contenido de **ra** un bit a la izquierda, insertando 0 en el bit menos significativo); **carga(Z,N)** |

**Formato de instrucción (1 palabra):**

OPCODE RSH: 00101
OPCODE LSH: 00100

- **Instrucciones de transferencia:**
  - Carga/Almacenamiento de registro con direccionamiento directo a memoria:

**LD ra,dir  ST ra,dir**

    - **ra** es un operando con direccionamiento directo a registro. Hace referencia a uno de los 8 registros.
    - **dir** es un operando de 8 bits con direccionamiento directo a memoria.
    - Acción de la instrucción:

| Instrucción | Acción |
|---|---|
| **LD ra,dir** (Carga/Load) | **ra←M[dir]** (llevar el contenido de la posición de memoria indicada por **dir** al registro **ra**) |
| **ST ra,dir** (Almacenamiento/Store) | **M[dir]←ra** (llevar el contenido del registro **ra** a la posición de memoria indicada por **dir**) |

**Formato de instrucción (1 palabra):**

OPCODE LD: 11000
OPCODE ST: 11001


  - Instrucciones de entrada/salida de datos:

**IN ra  OUT ra**

    - **ra** es un operando con direccionamiento directo a registro. Hace referencia a uno de los 8 registros.
    - Acción de la instrucción:

| Instrucción | Acción |
|---|---|
| **IN ra** | **ra←BUSIO** (llevar el contenido del bus de 16 bits de entrada del procesador al registro **ra**)<br>**IN=1** (activar salida IN para que algún dispositivo externo vuelque un dato en BUSIO) |
| **OUT ra** | **BUSIO=ra** (llevar el contenido del registro **ra** al bus de 16 btis de salida del procesador)<br>**OUT=1** (activar salida OUT para que algún dispositivo externo lea el dato de BUSIO) |

**Formato de instrucción (1 palabra):**

OPCODE IN: 11010
OPCODE OUT: 11011

## 4. Tareas a desarrollar por los alumnos

**IMPORTANTE: Respetad escrupulosamente la nomenclatura presentada en este documento (nombres de señales, de macros, de proyectos, etc.) para que la práctica sea evaluada correctamente.**

### 1- Diseño del ciclo de instrucción y de la unidad de datos

**1.1- Ciclo de instrucción y unidad de datos:** realizar un documento (.pdf, .doc, .odt, …) en el que se especifique el diseño del ciclo de instrucción para cada una de las instrucciones, así como un esquema de los componentes que formarán la unidad de datos:

- Expresar ciclo a ciclo, las transferencias que se deben realizar en la unidad de datos para cada una de las instrucciones.
- Realizar un esquema de la unidad de datos donde aparezcan los elementos utilizados así como las conexiones para poder ejecutar sobre ella las transferencias presentadas en el ciclo de instrucción.

**1.2- Cálculo del camino crítico y el tiempo de ciclo:** Calcula el camino crítico del ciclo de instrucción que has diseñado. El tiempo de este camino crítico determinará el tiempo de ciclo de vuestro procesador (periodo mínimo de la señal de reloj a la cual podría funcionar tu procesador).

- Para cada uno de los pasos de vuestro ciclo de instrucción calcula el tiempo necesario para realizar las transferencias que implica dicho paso. Para el cálculo de dicho tiempo tened en cuenta los tiempos necesarios para operar con diversos dispositivos que aparecen en la siguiente tabla.
- El mayor tiempo de los calculados en el apartado anterior determinará vuestro tiempo de ciclo.

Para el cálculo de del tiempo de ciclo considerar que el retardo de las señales de control es nulo.

| Componente | Tiempo | |
|---|---|---|
| Memoria | 50 ns | Acceso de lectura o escritura a memoria |
| Banco de Registros | 5 ns | Acceso de lectura o escritura a un registro del banco de registros |
| Registro/Contador | 3 ns | Escritura en un registro o incremento de un contador. La lectura es inmediata. |
| ALU | 10 ns | Una operación de la ALU |
| MUX 2/1 | 2 ns | Paso por un multiplexor de 2 entradas |
| MUX n/1 | 2 * $Log_2(n)$ ns | Paso por un multiplexor de n entradas |
| Triestado | 2 ns | Paso por un buffer triestado |
| Puerta lógica | 1 ns | Un nivel de puertas lógicas |

**1.3- Cálculo del tiempo de ejecución de un programa:** realizar un programa utilizando las instrucciones que has implementado en tu procesador que calcule el SAD (se explica más adelante) de dos vectores dados.
Una vez realizado el programa, calcular el tiempo de ejecución del mismo a partir del tiempo de ciclo calculado anteriormente y del ciclo de instrucción de cada una de las instrucciones utilizadas.

> **NOTA:** Los pasos 1.1, 1.2 y 1.3 forman parte de un proceso iterativo para la optimización del diseño final. Viendo el tiempo de ejecución del programa y de sus instrucciones quizás nos de ideas de cómo modificar la unidad de datos y el ciclo de instrucción (más o menos ciclos) para reducir el tiempo de ciclo, insertar nuevas instrucciones para facilitar la labor de programación, etc...

## 2- Implementación de la unidad de datos en Xilinx

Para la implementación de la unidad de datos puedes utilizar como bloques básicos de diseño: ALU, banco de registros, registros, sumador/restador, biestables, multiplexores, codificadores, decodificadores y puertas lógicas (todos estos elementos del ancho de bits que necesites).

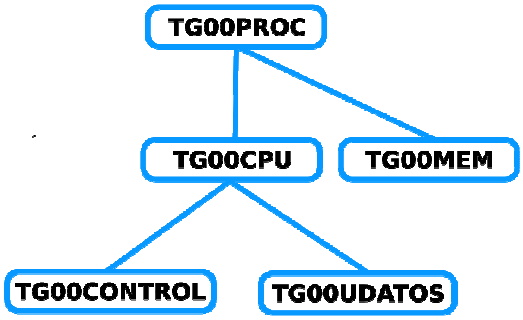
- Interfaz de la unidad de datos:
  - Entradas:
    - **CT[¿:0]**: Bus de control. Señales de control de la unidad de control a la unidad de datos. El número total de señales de control depende de tu implementación.
    - **RESET**: Señal de reset. Inicializa todos los elementos de memoria de la unidad de datos a 0.
    - **CLK**: Señal de reloj. Sincroniza por flanco de bajada todas las transferencias de la unidad de datos.
  - Salidas:
    - **DIR[7:0]**: Bus de direcciones. Especifica la dirección de memoria a la que se quiere acceder.
    - **IR[¿:0]**: Código de instrucción. Indicará a la unidad de control la instrucción que se está ejecutando. El número de bits necesarios depende de tu implementación.
    - **Z**: Señal de estado (cero). Indicará a la unidad de control si el se ha producido un resultado cero en la ALU.
    - **N**: Señal de estado (negativo). Indicará a la unidad de control si el se ha producido un resultado negativo en la ALU.
    - **C**: Señal de estado (acarreo). Indicará a la unidad de control si el se ha producido acarreo en la ALU.
  - Bus Entrada/salida:
    - **DATOS[15:0]**: Bus de datos entre la CPU y la memoria, utilizado para enviar datos a memoria o para leer datos de la misma.
    - **BUSIO[15:0]**: Bus de entrada/salida, utilizado para la comunicación de datos con el exterior a la CPU.

Debes bajarte de la plataforma de enseñanza virtual un prototipo de proyecto sobre el cual vas a diseñar tu procesador. El nombre del prototipo es TG00PROC.ZIP. Dicho nombre deberá ser cambiado antes de empezar a trabajar con él. Deberás cambiar los 4 primeros caracteres (**TG00**) por los que correspondan:

- **T**: Titulación del alumno (**I** Ing. Informática, **S** Ing. del Software, **C** Ing. de Computadores)
- **G**: Grupo (**A, B**)
- **00**: Número de equipo del alumno con dos dígitos (**01, 02, …**)

Así por ejemplo el equipo 13 del grupo B de la ingeniería informática, deberá trabajar sobre el proyecto con nombre: **IB13PROC**.

El proyecto TG00PROC es un proyecto jerárquico, en el cual existen una serie de macros que deberéis ir creando. La jerarquía de macros es la siguiente:




La macro que contendrá la unidad de datos es: **TG00UDATOS**. Recordad, que deberéis sustituir la macro original TG00UDATOS por la vuestra (con vuestro identificador).

## 3- Diseño de la unidad de control

Diseña una unidad de control (por el método que estimes oportuno) que implemente el ciclo de instrucción diseñado para el procesador y active las señales de control que van a la unidad de datos de forma adecuada.
Interfaz de la unidad de control:

- Entradas:
  - **IR[¿:0]**: Código de instrucción. Indicará la instrucción que se está ejecutando, cargada previamente en la unidad de datos.
  - **Z**: Señal de estado (cero). Indicará si el  se ha producido un resultado cero en la ALU de la unidad de datos.
  - **N**: Señal de estado (negativo). Indicará si el  se ha producido un resultado negativo en la ALU de la unidad de datos.
  - **C**: Señal de estado (acarreo). Indicará si el  se ha producido acarreo en la ALU de la unidad de datos.
  - **RESET**: Señal de reset. Inicializa la unidad de control.

        - **CLK**: Señal de reloj. Sincroniza por flanco de bajada los pasos del ciclo de instrucción.
    - Salidas:
        - **CT[¿:0]**: Bus de control. Señales de control para la unidad de dato. El número total de señales de control depende de tu implementación.
        - **CS**: Señal de activación (chip select) de la memoria.
        - **RW**: Señal de lectura (1) o escritura (0) a la memoria.
        - **IN**: Señal de salida que indica a un dispositivo externo que vuelque un dato en el bus de entrada (BUSIN)
        - **OUT**: Señal de salida que indica a un dispositivo externo que hay un dato disponible en el bus de salida (BUSOUT)

## 4- Programación del sistema

Realiza un programa utilizando las instrucciones que has implementado en tu procesador que calcule la suma de las diferencias en valor absoluto de los elementos de dos vectores (SAD), operación que se utiliza para medir cuanto se parecen dos vectores A y B.

$$SAD(A,B) = \sum_i |A[i] - B[i]|$$

Estos dos vectores estarán formados cada uno por un número de valores enteros (16 bits en C2) especificados en la posición $3E_{(16}$ y almacenados consecutivamente a partir de la posición de memoria indicada en la dirección $3F_{(16}$. Se realizará la resta elemento a elemento de los vectores (el primero de A menos el primero de B, el segundo menos el segundo, …) acumulando la suma total del valor absoluto de dichas restas. Debe almacenar el resultado en la posición $40_{(16}$ y **volcarlo en el bus de salida con una instrucción OUT** (no hay que tener en cuenta si se produce desbordamiento al calcular el SAD).

Una vez diseñado el programa utilizando las instrucciones del procesador, pasar a código máquina dichas instrucciones (utilizando el formato de instrucción mostrado al principio de este documento) e introducirlo en el fichero de memoria (ROM.MEM) de la macro TG00ROM que representa la parte ROM de la memoria (posiciones 0 a la 63 de la memoria). Introducirlo a partir de la posición 0 de memoria. En este mismo fichero, introducir los siguientes valores enteros a partir de la posición $36_{(16}$ de memoria: 25, -3, -6, 1, 23, -1, -4,-1, 4, 54. Esto corresponde a un ejemplo de vectores de 4 elementos.

Una vez metido todo el contenido en la ROM, simula el procesador utilizando una señal de reloj de periodo igual al tiempo de ciclo que se calculó en la tarea 1.2. Comprueba que el resultado de la simulación es correcto y que el tiempo que ha tardado en ejecutarse el programa se corresponde con el calculado en la tarea 1.3.

# 5.Descripción y uso de Xilinx