Práctica III

Prototipo de microcontrolador (PIC)

INDICE

Prototipo de microcontrolador de aplicación específica

1. Objetivos

En su primera fase de prototipado, un microcontrolador puede ser implementado en VHDL a fin de comprobar su funcionalidad. **En la práctica que proponemos en este enunciado se lleva a cabo el desarrollo de un microcontrolador de propósito específico.** Este microcontrolador poseerá una arquitectura muy simplificada y un juego de instrucciones reducido, de forma que, mediante un simple programa de código máquina, sea posible controlar algunos elementos (interruptores, actuadores de nivel, termostato, etc.).

El prototipo que nos ocupa debe ser capaz de recibir órdenes simples a través de una interfaz serie RS232, así como enviar información sencilla a través de esta línea. **El control físico de la línea es el realizado en la Práctica II**. Los datos recibidos se almacenarán en una memoria RAM, para lo cual se debe desarrollar un control de acceso directo a memoria que pida los buses al procesador cuando una transferencia sea necesaria. Para la transmisión se habilitará un registro especial cuyo control será compartido por el control de acceso a memoria y el núcleo del procesador. La interfaz serie puede estar regida en su otro extremo por un PC con interfaz serie ejecutando un programa de terminal.

La arquitectura interna del procesador es Harvard, es decir, dispone de memoria de datos y de programa con buses separados. El control dispone de una memoria ROM de programa, una RAM de datos con una estructura específica, una unidad ALU y el ya mencionado controlador RS232.

El propósito del microcontrolador objeto de la práctica es decodificar los comandos llegados por el puerto serie y actuar en consecuencia, bien sea enviando información por el puerto serie, bien actuando sobre una serie de interruptores y otros periféricos. Los periféricos que serán gestionados por el procesador son:

- 8 interruptores ON/OFF
- 10 actuadores de nivel
- Un termostato

2. Contenidos

En esta práctica se construirán algunos de los bloques que constituyen un microcontrolador de aplicación específica. **En este documento se presentan las especificaciones del sistema y se propone una arquitectura determinada.** Además, se indica una metodología para completar la descripción del sistema. En cualquier caso, **el alumno es libre de modificar todos los ficheros que se entregan al principio de la práctica**, según vaya realizando los distintos pasos, o para implementar las mejoras que considere oportunas en el sistema.

3. Resumen del estándar RS232

El estándar de comunicación asíncrona por línea serie RS232 ha sido uno de los más extendidos en el desarrollo de periféricos. Este estándar permite la comunicación bidireccional por dos líneas separadas, e incluye líneas opcionales de control de flujo de datos y negociación de transferencias.

La versión más simple del estándar especifica que la línea de transmisión de datos debe permanecer a nivel alto en reposo, iniciándose la transmisión con un bit que siempre será un nivel bajo. La forma más simple de transmisión envía 8 bits de datos y uno de parada (nivel alto), sin añadir bits de paridad, tal y como se indica en la Figura 1. La velocidad de transmisión puede configurarse desde 9.600 bps hasta 115.200 bps.

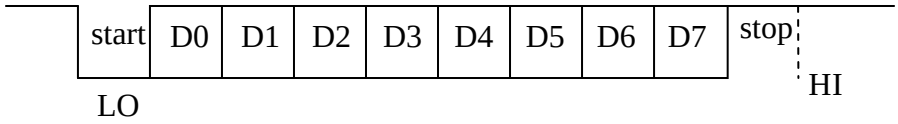

Figura 1. Protocolo de envío RS232

En el sistema propuesto en la práctica se hace uso de un controlador del nivel físico que emplea este modo de transmisión simple a una velocidad de 115.200 bps. Este sistema, al igual que el resto del procesador, será alimentado con una señal de reloj de 20 MHz, una frecuencia suficientemente alta para la funcionalidad del procesador.

4. Descripción de los comandos del terminal

Tal y como se ha explicado, la principal tarea del sistema microcontrolador es decodificar una serie de comandos recibidos a través de una línea serie. De acuerdo con los periféricos que se van a controlar y sus posibles valores, los comandos que puede recibir el procesador por la línea serie son los indicados en la Tabla 1.

Comando	Parámetro 1	Parámetro 2	Descripción
I	0..7	0, 1	Selecciona un interruptor de los 8 existentes y lo enciende (1) o lo apaga (0)
A	0..9	0..9	Selecciona uno de los 10 actuadores y le asigna un valor de apertura de 0 a 9
T	1, 2	0..9	Carga el valor del termostato a una temperatura entre 10 y 29 grados
S	I, A, T	0..9	Solicita información al procesador sobre el estado de alguno de los periféricos

Tabla 1. Comandos del terminal

Nótese que todos estos comandos están escritos en mayúsculas, por lo que el procesador sólo responderá a los códigos ASCII de los caracteres en mayúscula. Asimismo, es de notar que los comandos no se validan con un retorno de carro, lo que se debe a que este procesador no debería ser accedido directamente por un terminal, sino a través de una consola de mandos más elaborada. El sistema que se trata de diseñar en esta práctica es un prototipo, por lo que no nos preocupa acceder con una herramienta más pobre a fin de comprobar su funcionalidad.

El último de los comandos forzará al procesador a enviar a través de la línea serie dos bytes con información acerca del estado del periférico indicado. Este comando incluye dos parámetros: en el primero se indica el tipo de periférico del que se solicita información; en el segundo se indica el número identificador del periférico. Nótese que, en este caso, el parámetro 2 sólo tiene sentido para los interruptores y los actuadores, pudiendo tomar cualquier valor en el resto de casos.

Para la decodificación de estos comandos **se proporcionará a los alumnos un programa escrito en el ensamblador del procesador**, y que constituye una versión simplificada del programa que debería ejecutar un sistema real. Si los alumnos desean modificar y mejorar este programa, está disponible un *software* que compila el código y lo convierta en una ROM escrita en VHDL. Nótese sin embargo que la modificación del programa es, para los objetivos de la práctica, un aspecto opcional.

5. Descripción de la arquitectura y de la interfaz con el exterior

El sistema microprocesador objeto de la práctica tiene una arquitectura Harvard, es decir, los buses de datos e instrucciones están separados. **El bus de instrucciones está disponible únicamente para el decodificador de las mismas,** mientras que **del bus de datos cuelgan, además del control, la memoria RAM, la ALU y el sistema de DMA**. Según lo descrito hasta el momento, la arquitectura puede resumirse en la Figura 2.

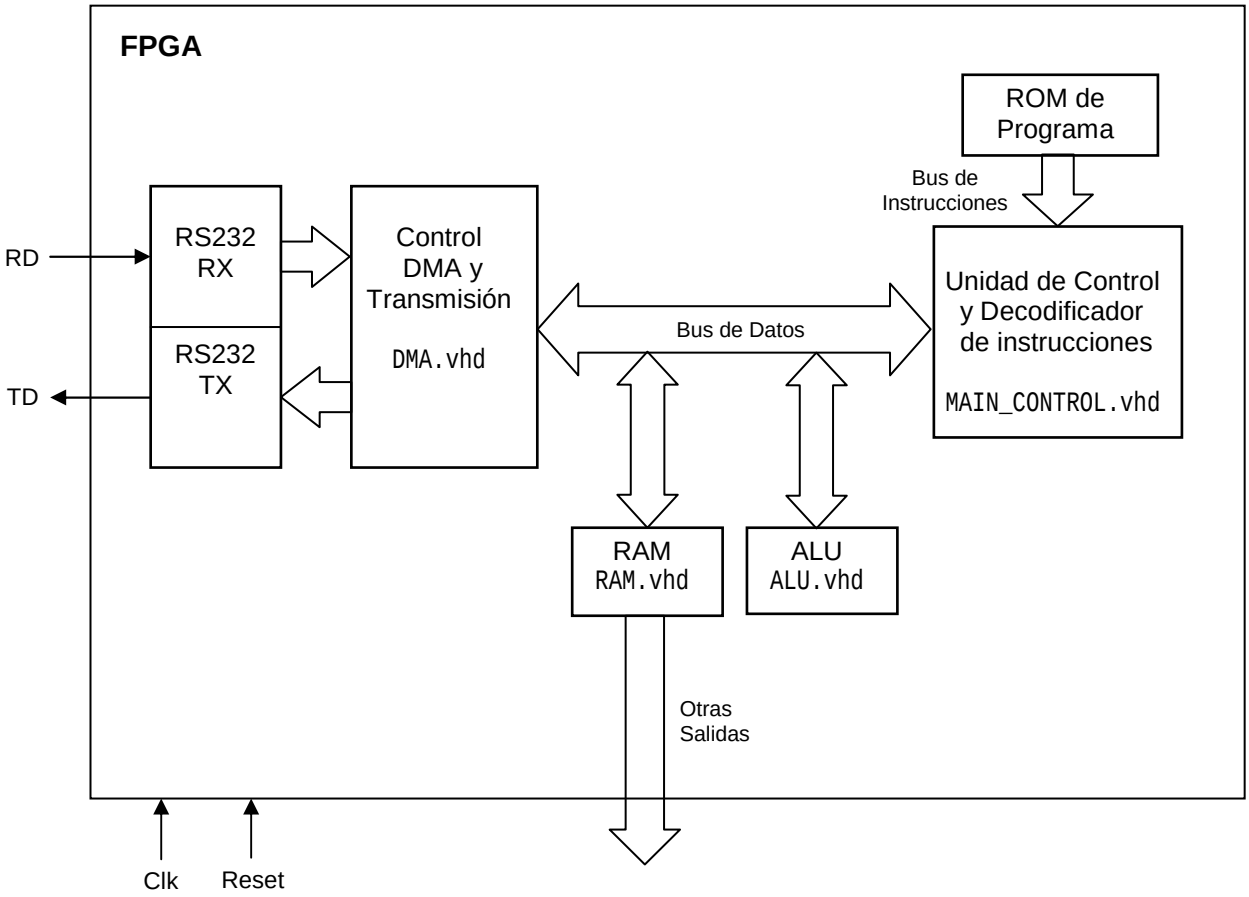

Figura 2. Diagrama de bloques de alto nivel

Como puede verse en la figura, el procesador dispone de apenas un pequeño conjunto de líneas de entrada y salida. Aparte de las líneas de reloj y reset, y de las de transmisión/recepción RS232, el sistema dispondrá de un grupo de líneas desde la RAM que aprovechen el hardware montado en el laboratorio para mostrar información de interés. Esta información estará relacionada con el estado de cada uno de los periféricos que se controlan.

El diagrama se representa a muy alto nivel, y no muestra más que las conexiones lógicas entre los bloques del sistema. La descripción de las entradas, salidas, y funcionalidad de cada bloque se indicará a continuación, cuando se describa cada uno de ellos.

6. Descripción funcional de los bloques del sistema

6.1. Interfaz física RS232

La implementación de la interfaz física de transmisión y recepción por línea RS232, realizada en Práctica II, se caracteriza por:

- Recepción y transmisión de datos a 115.200 bps.
- Formato de trama de 8 bits de datos, sin paridad y con un bit de parada.

- Capacidad para almacenamiento de hasta 16 datos recibidos mediante una memoria FIFO.
- Líneas de estado de la memoria de recepción: indicadores de memoria llena y vacía.
- Entrega de los datos recibidos a la entidad cliente por medio de una interfaz paralela y síncrona mediante la línea Data_read.
- Captura de datos mediante una interfaz asíncrona basada en una línea de validación de datos (Valid_D) y otra de reconocimiento de carga (ACK_in).
- Línea de estado del transmisor (TX_RDY): reposo o enviando.

Las líneas de entrada y salida de la interfaz se indican, junto con su descripción, en la Tabla 2.

LÍNEA	SENTIDO	FUNCIÓN
Reset	Entrada	Reset asíncrono y activo a nivel bajo.
Clk	Entrada	Reloj del sistema.
Data_in [7..0]	Entrada	Entrada de los datos que el sistema cliente desea enviar. Bit más significativo en la posición 7.
Valid_D	Entrada	Validación del dato de entrada por parte del sistema cliente. Activa a nivel bajo.
ACK_in	Salida	Reconocimiento del dato para transmitir. Activa a nivel bajo, se activa al guardar un nuevo dato válido y se desactiva al desactivarse Valid_D.
TX_RDY	Salida	Transmisor está disponible. Activa a nivel alto.
TD	Salida	Línea de transmisión RS232.
RD	Entrada	Línea de recepción RS232.
Data_out [7..0]	Salida	Datos de salida de la FIFO interna del subsistema receptor. Bit más significativo en la posición 7.
Data_read	Entrada	Petición del sistema cliente de lectura de un nuevo dato de los recibidos y almacenados. Activa a nivel alto y síncrona.
Full	Salida	Indica que la memoria interna del subsistema receptor está llena. Activa a nivel alto.
Empty	Salida	Indica que la memoria interna del subsistema receptor está vacía. Activa a nivel alto.

Tabla 2. Líneas de entrada y salida de la interfaz RS232

Como puede inferirse de la tabla anterior, el protocolo de recepción de datos es simple. **El sistema cliente debe muestrear la línea Empty y, cuando esté a nivel bajo, activar la línea Data_read durante un ciclo de reloj para recibir los datos por las líneas Data_out.**

Por otra parte, la carga de datos para su envío debe tener en cuenta que nunca se debe intentar cargar un nuevo dato mientras el transmisor esté ocupado, ya que de ese modo variaríamos los datos que se están enviando. Afortunadamente, esto queda solucionado mediante el protocolo de carga de datos asíncrono de que se dispone. La interfaz ofrecida nunca carga el nuevo dato mientras el transmisor no esté libre. Por ello, **para cargar un dato, se debe colocar éste en el bus Data_in, poner a nivel bajo la línea Valid_D y esperar a que se ponga a nivel bajo la línea ACK_in. La temporización de la entrega de datos del DMA al transmisor RS232 se muestra en la Figura 3.**

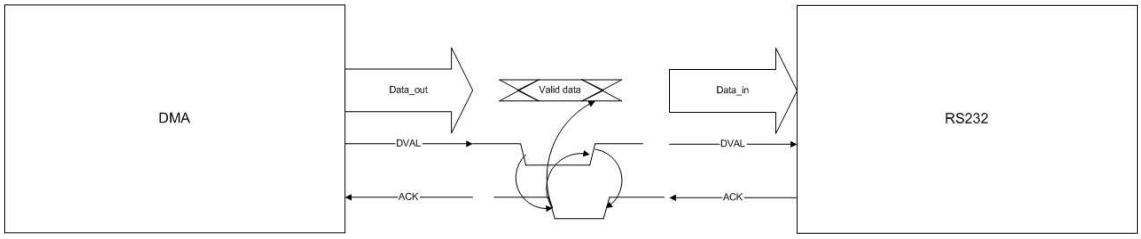

Figura 3. Protocolo de entrega de datos al transmisor RS232

La línea TX_RDY se pone a nivel bajo a la vez que ACK_in, y permanece así hasta que la transmisión iniciada se haya terminado.

6.2. Arquitectura de la memoria de datos

En este prototipo del microcontrolador utilizaremos una entidad RAM que implementará una memoria RAM de propósito general y también una serie de posiciones de memoria (registros de E/S) para acceso a los diferentes periféricos que el sistema completo debe controlar, así como las posiciones de memoria empleadas por el controlador de acceso directo a memoria. La entidad contará con una serie de líneas de salida para control de los algunos de los periféricos.

Hechas estas consideraciones, el mapa de la memoria de datos del procesador aparece en la Tabla 3.

Dirección	Alias	Función
0x00	DMA_RX_Buffer (MSB)	Byte más significativo de la reserva para el controlador DMA (recepción)
0x01	DMA_RX_Buffer	Byte intermedio de la reserva para el controlador DMA (recepción)
0x02	DMA_RX_Buffer (LSB)	Byte menos significativo de la reserva para el controlador DMA (recepción)
0x03	NEW_INST	Flag que indica la llegada de un nuevo comando por la línea serie
0x04	DMA_TX_Buffer (MSB)	Byte más significativo de la reserva para el controlador DMA (transmisión)
0x05	DMA_TX_Buffer (LSB)	Byte menos significativo de la reserva para el controlador DMA (transmisión)
0x06 … 0x0F	Reservado	Para posterior ampliación
0x10 … 0x17	SWITCH(0..7)	Zona de control de interruptores
0x18 … 0x1F	Reservado	Para posterior ampliación
0x20 … 0x29	LEVER(0..9)	Zona de control de actuadores
0x2A … 0x30	Reservado	Para posterior ampliación
0x31	T_STAT	Temperatura fijada en el termostato
0x32 … 0x3F	Reservado	Para posterior ampliación
0x40 … 0xFF	GP_RAM	Memoria de propósito general

Tabla 3. Mapa de la memoria de datos

La entidad RAM que implementa el mapa de memoria está conectada al bus de datos general del procesador, **por lo que debe disponer de salidas triestado**. Las líneas de acceso al componente se especifican en la Tabla 4.

Línea	Sentido	Descripción
Reset	Entrada	Reset asíncrono a nivel bajo para los registros de E/S
Clk	Entrada	Reloj principal del sistema (20 MHz)
Databus[7..0]	Bidireccional	Bus de datos del sistema

Address[7..0]	Entrada	Direcciones del bus de datos
Write_en	Entrada	Habilitación de escritura
OE	Entrada	Habilitación de lectura
Switches[7..0]	Salida	Estado de los interruptores
Temp_L[6..0]	Salida	Dígito más bajo del valor de temperatura del termostato (7 segmentos)
Temp_H[6..0]	Salida	Dígito más alto del valor de temperatura del termostato (7 segmentos)

Tabla 4. Líneas de entrada y salida de la RAM

La escritura de la RAM se realiza de forma síncrona, y es habilitada mediante la señal Write_en. La señal de Reset existe como forma de inicialización de los registros de carácter específico de la RAM (direcciones 0x00 a la 0x3F). **El Reset debe inicializar dichos registros a un valor conocido, aunque no necesariamente nulo** (carecería de sentido en el caso del termostato).

Las líneas más especiales de la RAM son las de salida al exterior. Switches[7..0] debe recoger el estado de los interruptores mapeados en memoria. Del mismo modo, las líneas Temp_X[7..0] deben presentar la conversión del valor a programar en el termostato, de forma que pueda ser mostrado en dos visualizadores de 7 segmentos.

6.3. Controlador DMA

El siguiente bloque que deben recorrer los caracteres llegados por la línea serie es el controlador DMA. Su propósito es doble:

- Por un lado, volcar a memoria los datos llegados a la interfaz RS232. Para ello, **debe pedir los buses al procesador principal**. Una vez que éste los haya concedido, debe cargar los bytes de que disponga la interfaz física RS232 en las direcciones de memoria habilitadas al efecto. Dado que todos los comandos que se pueden recibir por la línea serie son de 3 bytes, la memoria reservada tiene esa longitud. En concreto, la RAM del procesador reserva los bytes 0x00 a 0x02 a tal efecto. **Este controlador es responsable de escribir el valor 0xFF en el registro NEW_INST de la RAM cuando haya entregado un comando completo**, aunque este requisito se modificará más adelante si se implementa un control por interrupciones.
- Por otro lado, el controlador se encarga de cargar la interfaz física de transmisión serie con las respuestas a los comandos de usuario. El sistema está preparado para que los códigos que se envíen sean de 2 bytes, que es el número de posiciones reservadas en el mapa de memoria. **El comienzo de la emisión de una pareja de bytes se realiza a iniciativa del procesador, que emplea para ello una instrucción especial** que pone a '1' la línea Send_comm del controlador. Adicionalmente, **el procesador cede los buses de forma automática, por lo que en este caso no es necesario pedírselos.**

Las líneas que debe tener este controlador se indican en la Tabla 5.

Línea	**Sentido**	**Descripción**
Reset	Entrada	Reset asíncrono y activo a nivel bajo
Clk	Entrada	Reloj principal del sistema (20 MHz)
RCVD_Data[7..0]	Entrada	Dato recibido por la línea RS232
RX_Full	Entrada	Señal de estado de la memoria interna del receptor
RX_Empty	Entrada	Señal de estado de la memoria interna del receptor
Data_Read	Salida	Petición de lectura de un nuevo dato de los recibidos
ACK_out	Entrada	Señal de reconocimiento de la petición de lectura
TX_RDY	Entrada	Estado de la máquina de transmisión serie
Valid_D	Salida	Validación del dato enviado al transmisor RS232
TX_Data [7..0]	Salida	Dato para enviar por línea serie
Address [7..0]	Salida	Direcciones del bus de datos del sistema
Databus [7..0]	Bidireccional	Bus de datos del sistema
Write_en	Salida	Indicación de escritura para la RAM
OE	Salida	Habilitación de la salida de la RAM

DMA_RQ	Salida	Petición de buses al procesador principal
DMA_ACK	Entrada	Reconocimiento y préstamo de buses por parte del procesador principal
Send_comm	Entrada	Señal de comienzo de envío de datos, controlada por el procesador principal
READY	Salida	Señal a nivel alto únicamente cuando el controlador DMA se encuentre totalmente ocioso

Tabla 5. Líneas de entrada y salida del controlador DMA

El acceso directo a memoria (recepción) ocurre del siguiente modo:

1. El controlador espera a que la línea RX_Empty del RS232 sea cero.
2. Entonces pone a '1' la línea DMA_RQ y espera a que el procesador conceda los buses.
3. El procesador concede los buses y coloca un '1' en la línea DMA_ACK.
4. El controlador realiza la lectura del RS232 y escritura en la RAM.
5. Después pone un '0' en la línea DMA_RQ para devolver el control de los buses.
6. El procesador coloca un '0' en la línea DMA_ACK y continúa la ejecución.

La transmisión se lleva a cabo siguiendo el siguiente esquema:

1. El procesador espera a que la línea READY del DMA sea '1'.
2. Entonces libera los buses y pone un '1' en la línea SEND_comm, lo que fuerza que la línea READY del DMA vaya a nivel bajo inmediatamente.
3. El controlador realiza la carga y transmisión de los 2 bytes previstos, uno tras otro. Durante el proceso completo el controlador está en posesión de los buses de la memoria.
4. Terminada la transmisión de los datos, el controlador vuelve a nivel alto la línea READY, y el procesador puede retirar la petición de transmisión de la línea SEND_comm.

6.4. Unidad aritmético-lógica (ALU)

El subsistema de cálculo del sistema microprocesador es una implementación ligera y algo modificada de una ALU cualquiera de un procesador RISC (*Reduced Ïnstruction Set Computer*). Soporta un juego muy reducido de instrucciones típicas, incluyendo un par de instrucciones específicas para simplificar la implementación de las tareas de control.

La ALU dispone de dos registros de carga para operandos, un acumulador y un registro específico para carga de índices, destinado al acceso indexado a memoria. Además, posee internamente un registro con 4 bits de estado (Z, C, N y E), que se presentan directamente al control central. El significado de estos bits de estado o *banderas* se especifica en la Tabla 6. La información sobre las instrucciones que las modifican se encuentra con la especificación de las instrucciones del procesador.

Flag	Descripción
Z	*Zero*: vale '1' si el resultado de la última operación fue cero
C	Bit de acarreo en el bit más significativo del sumador/restador
N	Bit de acarreo de *nibble* (medio byte), es decir, en el bit 3 del sumador/restador
E	Error: vale '1' cuando se llega a algún resultado inesperado en la ALU

Tabla 6. Descripción de los *flags* (banderas) del procesador

En el desarrollo de la Práctica III sólo es necesario implementar la bandera Z. Todas las demás pueden ser ignoradas.

La ruta de datos de la ALU puede verse en la Figura 4.

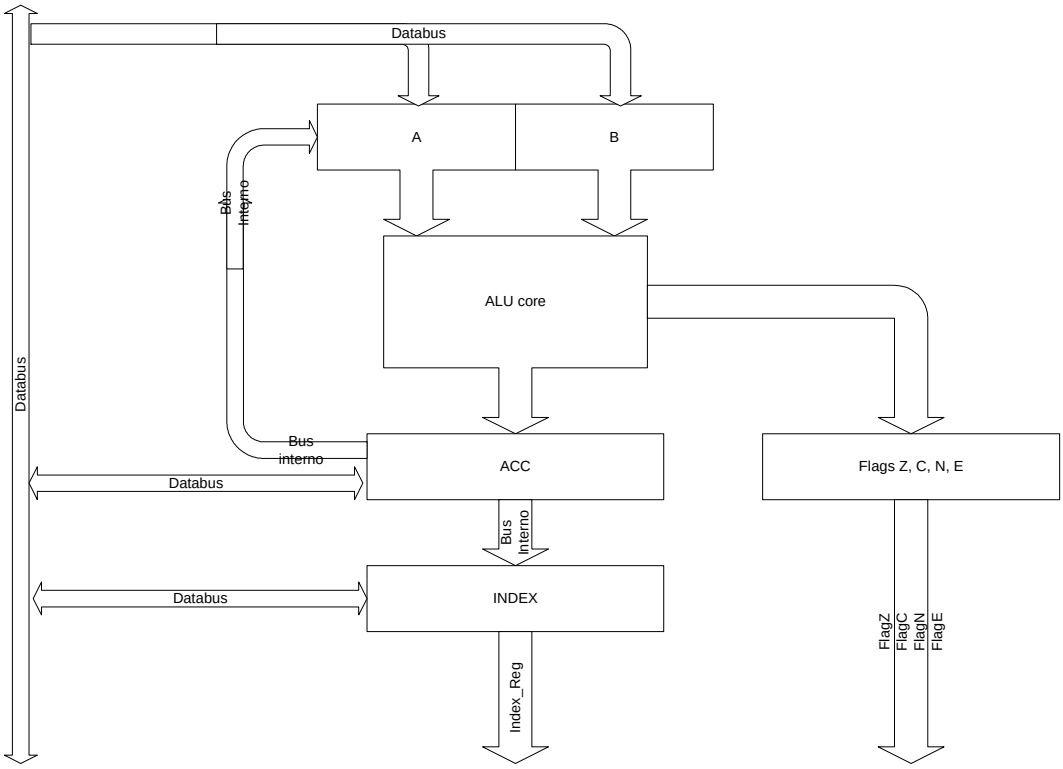

Figura 4. Ruta de datos de la ALU

Las operaciones que debe realizar la ALU se especifican más adelante, junto con el juego de instrucciones del procesador. Los puertos de entrada y salida de la ALU son los indicados en la Tabla 7.

Línea	Sentido	Descripción
Reset	Entrada	Asíncrono y activo a nivel bajo. Inicializa los registros internos a 0x00
Clk	Entrada	Reloj principal del sistema, 20 MHz
Alu_op[5..0]	Entrada	Bus de microinstrucciones del procesador. La ALU permite hasta 64 operaciones, aunque en este prototipo sólo se implementarán algunas
Databus	Bidireccional	Bus de datos del sistema
Index_Reg	Salida	Conexión directa del registro índice al decodificador de instrucciones
FlagZ	Salida	Flag de cero
FlagC	Salida	Flag de acarreo
FlagN	Salida	Flag de acarreo en *nibble*
FlagE	Salida	Flag de error

Tabla 7. Líneas de entrada y salida de la ALU

Entre las microinstrucciones que recibe la ALU han de encontrarse aquéllas que permitan la salida de datos de los registros. Estas microinstrucciones no se corresponden con ninguna instrucción en particular del procesador (más bien con varias de ellas).

En relación con la temporización, **todas las instrucciones de la ALU han de realizarse en un único ciclo de reloj**.

6.5. ROM de programa

El programa del microcontrolador se encuentra en este bloque. Dicho programa se encuentra descrito en una ROM que será accedida según las necesidades del control principal del procesador.

Las instrucciones del procesador son de longitud variable, 1 ó 2 palabras de 12 bits. En general, la primera palabra de la instrucción define la funcionalidad de la misma, reservándose la segunda para

la definición de constantes o direcciones de memoria en aquellas instrucciones que lo requieran (como por ejemplo en una instrucción de salto o de acceso a memoria).

Las líneas del componente son las mínimas necesarias para hacer que éste sea accesible únicamente por el control principal. Estas líneas se indican en la Tabla 8.

Línea	Sentido	Descripción
ROM_Addr[11..0]	Entrada	Bus de direcciones de la ROM de programa
ROM_Data[11..0]	Salida	Bus de instrucciones

Tabla 8. Líneas de entrada y salida de la ROM

Por último, nótese que sólo es posible direccionar un máximo de 4.096 palabras de programa. Esto, aunque pueda parecer limitado, es más que suficiente para un primer prototipado de un sistema. En el momento en el que se desee añadir más memoria de programa, bastará con incluir líneas de dirección al control central y a la ROM, sin variar en absoluto el resto del diseño.

6.6. Control principal

El núcleo de la ejecución del procesador se encuentra en este bloque. Esta máquina de control se encarga de recoger las instrucciones de la ROM de programa y ejecutarlas, generando las microinstrucciones necesarias.

El control actúa de forma cíclica realizando las siguiente tareas en cada iteración:

1. Comprobar si el controlador DMA está pidiendo los buses. Si es así, concedérselos inmediatamente y detenerse hasta que el DMA acabe su acceso a memoria.
2. Leer la dirección de memoria ROM que toque y decodificar la instrucción obtenida.
3. Ejecutar la instrucción. Para ello puede ser necesario generar microinstrucciones, acceder a la memoria de datos (RAM), o recoger una nueva palabra de la memoria de programa.
4. Si la instrucción es de tipo 4 (SEND), puede ser necesario congelar (*stall*) el procesador hasta que la instrucción se lleve a cabo.
5. Regresar a la primera tarea.

Las líneas que este componente debe disponer son las indicadas en la Tabla 9.

Línea	Sentido	Descripción
Reset	Entrada	Asíncrono y activo a nivel bajo
Clk	Entrada	Reloj principal del sistema, 20 MHz
ROM_Data[11..0]	Entrada	Bus de datos de la memoria de programa
ROM_Addr[11..0]	Salida	Bus de direcciones de la memoria de programa
RAM_Addr	Salida	Bus de direcciones de la memoria de datos
RAM_Write	Salida	Microinstrucción para escritura en la RAM
RAM_OE	Salida	Microinstrucción para lectura de la RAM
Databus	Bidireccional	Bus de datos del sistema
DMA_RQ	Entrada	Línea de petición de buses del controlador DMA
DMA_ACK	Salida	Microinstrucción de entrega de los buses al controlador DMA
SEND_comm	Salida	Microinstrucción para iniciar una transmisión por la línea serie
DMA_READY	Entrada	Señal de estado del controlador DMA
ALU_op[5..0]	Salida	Microinstrucciones con la operación parar realizar en la ALU
Index_Reg	Entrada	Conexión directa desde el registro índice de la ALU
FlagZ	Entrada	Flag de cero de la ALU
FlagC	Entrada	Flag de acarreo de la ALU
FlagN	Entrada	Flag de acarreo de *nibble* de la ALU
FlagE	Entrada	Flag de error de la ALU

Tabla 9. Líneas de entrada y salida del control principal

7. Instrucciones del procesador

El juego de instrucciones del microcontrolador está compuesto por cuatro tipos de instrucciones de longitud variable. A continuación se especifica cada uno de estos tipos y su funcionalidad.

Es de notar que, aunque la ROM tiene un ancho de 12 bits, las palabras que definen las instrucciones ocupan sólo los 8 bits menos significativos, siendo el resto ceros. Esto permite la posterior ampliación del juego de instrucciones a nuevos tipos y nemónicos.

Para comprender el formato binario de las instrucciones es necesario tener en cuenta las constantes definidas en el paquete `PIC_pkg` (fichero `PIC_pkg.vhd`). **Esas constantes se usan también ampliamente en la ROM de programa proporcionada.**

7.1. Tipo 1, instrucciones relativas a la ALU

Nemónico	Descripción	Flags que modifica
ADD	A + B	Z, C, N
SUB	A – B	Z, C, N
SHIFTL	Desplaza hacia la izquierda el contenido del acumulador, introduciendo un cero	
SHIFTR	Desplaza hacia la derecha el contenido del acumulador, introduciendo un cero	
AND	'and' lógico entre A y B	Z
OR	'or' lógico entre A y B	Z
XOR	'xor' lógico entre A y B	Z
CMPE	A = B	Z
CMPG	A > B	Z
CMPL	A < B	Z
ASCII2BIN	Convierte A del formato ASCII al binario (sólo para números, devuelve FF si hay error)	E
BIN2ASCII	Convierte A del formato binario al ASCII (para números menores de 0x10, devuelve FF si hay error)	E

Tabla 10. Instrucciones de tipo 1

Todas estas instrucciones son de 1 palabra de longitud.

En las comparaciones (CMPE, CMPL, CMPG) se varía el bit de cero (Z), **colocando un '1' si la comparación fue verdadera y un '0' en caso contrario.** Por otra parte, el bit E se coloca a '1' cuando se produce un error en las funciones de conversión ASCII2BIN y BIN2ASCII. Los flags C, N y E no se utilizan para nada en este procesador, pero se han incluido para facilitar su posterior incorporación a las funciones del mismo.

Recuerda que, en el desarrollo de la Práctica III, sólo es necesario implementar la bandera Z. Todas las demás pueden ser ignoradas.

El formato binario de estas instrucciones es simple. Los 4 bits más significativos están a 0, después 2 bits que indican el tipo de instrucción y el resto el código de operación. Haciendo uso de las constantes definidas en `PIC_pkg.vhd`, un ejemplo de instrucción de tipo 1 sería "0000" & TYPE_1 & ALU_ADD.

7.2. Tipo 2, instrucciones de salto

Nemónico	Parámetro	Descripción
JMP	Dirección de destino	Salto incondicional
JMPTrue	Dirección de destino	Salto si FlagZ = '1'

Tabla 11. Instrucciones de tipo 2

Todas estas instrucciones son de longitud fija, 2 palabras en este caso. La primera palabra es la instrucción en sí y la segunda la dirección de salto.

El formato binario se define del siguiente modo:

- Primera palabra: los 4 bits más significativos están a 0, seguidos de 2 bits que indican el tipo de instrucción y el resto de bits para el código de operación (en este caso sólo hay dos posibilidades). Haciendo uso de las constantes definidas en `PIC_pkg.vhd`, un ejemplo de instrucción de tipo 2 sería "0000" & TYPE_2 & JMP_COND.
- Segunda palabra: es la dirección de salto, por ejemplo en formato hexadecimal.

7.3. Tipo 3, instrucciones de movimiento de datos

Nemónico	Parámetro Destino	Parámetro Origen	Descripción
LD LDI	.A .B .INDEX	.ACC	Movimiento entre registros
LD	.A .B .INDEX .ACC	Constante [Constante] [Dirección Inmediata]	Carga para los registros de la ALU
LDI	.A .B .INDEX .ACC	[Constante] [Dirección Inmediata]	Carga para los registros en la ALU con acceso indexado a memoria utilizando el registro INDEX
WR	Dirección inmediata Constante		Escribe datos en memoria
WRI	Dirección inmediata Constante		Escribe datos en memoria con acceso indexado mediante INDEX

Tabla 12. Instrucciones de tipo 3

Estas instrucciones pueden tener diferentes longitudes. Si la transferencia es entre registros (LD .A, .ACC) la instrucción tendrá una única palabra. En el resto de casos tendrá dos.

La instrucción LD entre registros puede utilizarse únicamente para cargar ACC en algún otro de los registros de la ALU. Así, instrucciones del tipo LD .ACC, .A son ilegales. Por otra parte, es de notar que no es posible transferir una constante directamente desde la memoria de programa hasta la memoria de datos. Para realizar esta operación son necesarias dos instrucciones del procesador:

```
LD .ACC, 45
WR .ACC, x40
```

Las instrucciones LDI y WRI tienen la particularidad de utilizar el registro INDEX, cuyo valor se suma a la dirección de memoria indicada. De este modo, se pueden leer y escribir tablas en memoria de datos de forma sencilla.

A la memoria principal se puede acceder por medio de constantes predefinidas. Sin embargo, nótese que las constantes se pueden definir desde 0 a 4.095, pero a la hora de hacer uso de éstas para acceder a memoria o cargar los registros, sólo se usarán las 8 líneas más bajas de las 12 posibles.

El formato binario se define del siguiente modo:

- Primera palabra: los 4 bits más significativos están a 0 y después tenemos 2 bits que indican el tipo de instrucción, 1 bit que indica el tipo de operación, 2 que indican la fuente desde la que se carga y el resto (3 bits) el destino.
- Segunda palabra: en los casos en los que sea necesaria, tiene diversos significados según el tipo de instrucción, fuente y/o destino. En general, define todo aquello que no pudo definirse en la primera palabra de la instrucción, esto es, direcciones de memoria o constantes.

Algunos ejemplos de definición de instrucciones, junto con su codificación y descripción, pueden verse en la Tabla 13.

Código	Codificación	Interpretación
LD .A, .ACC	X"0" & TYPE_3 & LD & SRC_ACC & DST_A	Carga el acumulador en A
LD .A, X65	X"0" & TYPE_3 & LD & SRC_CONSTANT & DST_A X"065"	Carga 0x65 en A (8 bits menos significativos)
LDI .A, [X65]	X"0" & TYPE_3 & LD & SRC_INDXD_MEM & DST_A X"065"	Carga el contenido de la dirección (0x65 + INDEX) en A
LD .A, C	X"0" & TYPE_3 & LD & SRC_CONSTANT & DST_A X"000" (según el valor de C)	Carga el valor de la constante C en A
LD .A, [C]	X"0" & TYPE_3 & LD & SRC_MEM & DST_A X"000" (según el valor de C)	Carga el contenido de la dirección C en A
WR X65	X"0" & TYPE_3 & WR & SRC_ACC & DST_MEM X"065"	Guarda el acumulador en la dirección 0x65
WRI C	X"0" & TYPE_3 & WR & SRC_ACC & DST_INDXD_MEM X"000" (según el valor de C)	Guarda el acumulador en la dirección 0x65 + INDEX

Tabla 13. Ejemplos de instrucciones de tipo 3

Es necesario reiterar que, aunque las palabras de la memoria de programa tienen 12 bits, sólo los 8 menos significativos son empleados al hacer uso de constantes y direcciones de memoria.

7.4. Tipo 4, instrucciones especiales

Nemónico	Descripción
SEND	Detiene el procesador y ejecuta una petición de transmisión al controlador DMA

Tabla 14. Instrucciones de tipo 4

Mediante esta instrucción, de una sola palabra, se detiene el procesador y se le da la orden al controlador DMA para que transmita por el puerto serie los datos almacenados en las posiciones de memoria DMA_TX_Buffer (2 bytes). Esta instrucción queda definida en binario del siguiente modo: X"0" & TYPE_4 & "000000".

8. Consideraciones adicionales para la implementación

8.1. Memoria RAM

La memoria RAM del procesador puede dividirse en dos bloques de cara a su implementación. En primer lugar, la batería de posiciones de memoria que poseen algún significado específico (las posiciones 0x00 a 0x3F), y por otro lado la memoria de propósito general (posiciones 0x40 a 0xFF). Atendiendo a su descripción funcional, **la única diferencia apreciable entre una zona de memoria y la otra es la existencia de una señal de Reset para la primera.** Sin embargo, si la descripción es correcta, ambas zonas emplearán recursos de distinto tipo dentro de la FPGA. Para la implementación de este bloque se ofrece como ejemplo el archivo `RAM.vhd`, que proporciona una memoria de propósito general de 16 bytes junto al código de conversión de BCD a 7 segmentos.

La tarea del alumno con este bloque es crear una entidad que contenga una memoria RAM de propósito general como la ofrecida, pero del tamaño adecuado, y una segunda memoria que contenga la señal de Reset.

Si la memoria RAM genérica se describe por separado, la herramienta de síntesis la reconocerá como tal y será capaz de optimizarla haciendo un mejor uso del *hardware*. Esta optimización no es posible para la zona de memoria con señal de Reset, y el sintetizador hará uso de *flip-flops* discretos hasta completarla. Adicionalmente, **el alumno debe diseñar un decodificar de direcciones** para acceder a uno u otro bancos de memoria.

La RAM dispone de líneas de salida destinadas a mostrar el estado de algunos de los registros de propósito específico. Las líneas Switches muestran el estado de los interruptores, para lo que basta que **cada línea esté directamente asociada con el bit menos significativo** de cada registro de estado de interruptor. Las líneas Temp_L y Temp_H deben convertir los *nibbles* bajo y alto del registro T_STAT (dirección 0x31) a un formato legible en un *display* de 7 segmentos. El código de conversión tendrá en cuenta la numeración de los segmentos de los visualizadores mostrada en la Figura 5.

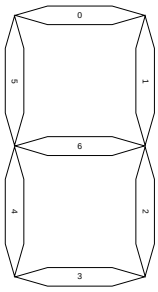

Figura 5. Numeración de los segmentos en un visualizador

Una última consideración a tener en cuenta es que **el bus de datos debe permanecer en estado de alta impedancia mientras que la señal OE esté a nivel alto** (véase la especificación de las señales en la Tabla 4).

Como apoyo para la codificación de la RAM se ha definido un tipo de datos y numerosas constantes en el paquete `PIC_pkg` (contenido en el archivo `PIC_pkg.vhd`). Este paquete contiene también otros recursos destinados a facilitar el desarrollo de otros elementos del microcontrolador.

8.2. ALU

La implementación de la ALU resulta bastante inmediata. Basta con seguir la ruta de datos indicada en la figura correspondiente. **Para desarrollar de una forma cómoda el conjunto de microins-**

trucciones que debe interpretar la ALU, se ofrece el tipo `alu_op` **como parte del paquete** contenido en `PIC_pkg.vhd`. Si se hiciera uso de este tipo, la entidad de la ALU sería:

```
entity ALU is
  port (
    Reset         : in    std_logic;    -- asynnchronous, active low
    Clk           : in    std_logic;    -- System clock, 20 MHz, rising_edge
    u_instruction : in    alu_op;       -- u-instructions from CPU
    FlagZ         : out   std_logic;    -- Zero flag
    FlagC         : out   std_logic;    -- Carry flag
    FlagN         : out   std_logic;    -- Nibble carry bit
    FlagE         : out   std_logic;    -- Error flag
    Index_Reg     : out   std_logic_vector(7 downto 0);   -- Index register
    Databus       : inout std_logic_vector(7 downto 0)    -- System Data bus
  );
end ALU;
```

Nótese que **la ALU debe mantener el bus de datos** (Databus) **en alta impedancia siempre que no le sea imprescindible su uso**.

8.3. Controlador DMA

La primera de las máquinas de estados que forman parte del sistema es la encargada de gestionar la comunicación a través de la línea serie y la interacción necesaria con la RAM. Su tarea básica es "escuchar" las líneas Empty del receptor RS232 y Send_Comm del control principal del procesador. **Para el diseño de este bloque se recomienda construir una única máquina de estados** que englobe ambos procesos, recepción y transmisión.

Un esquema de conexión entre los bloques implicados se muestra en la Figura 6. **Las líneas** OE, Write_En, Address y Databus son **de uso compartido con el procesador principal** (en el caso de Databus también con la ALU), por lo que **deberán mantenerse en alta impedancia siempre que no sea necesario usarlas.**

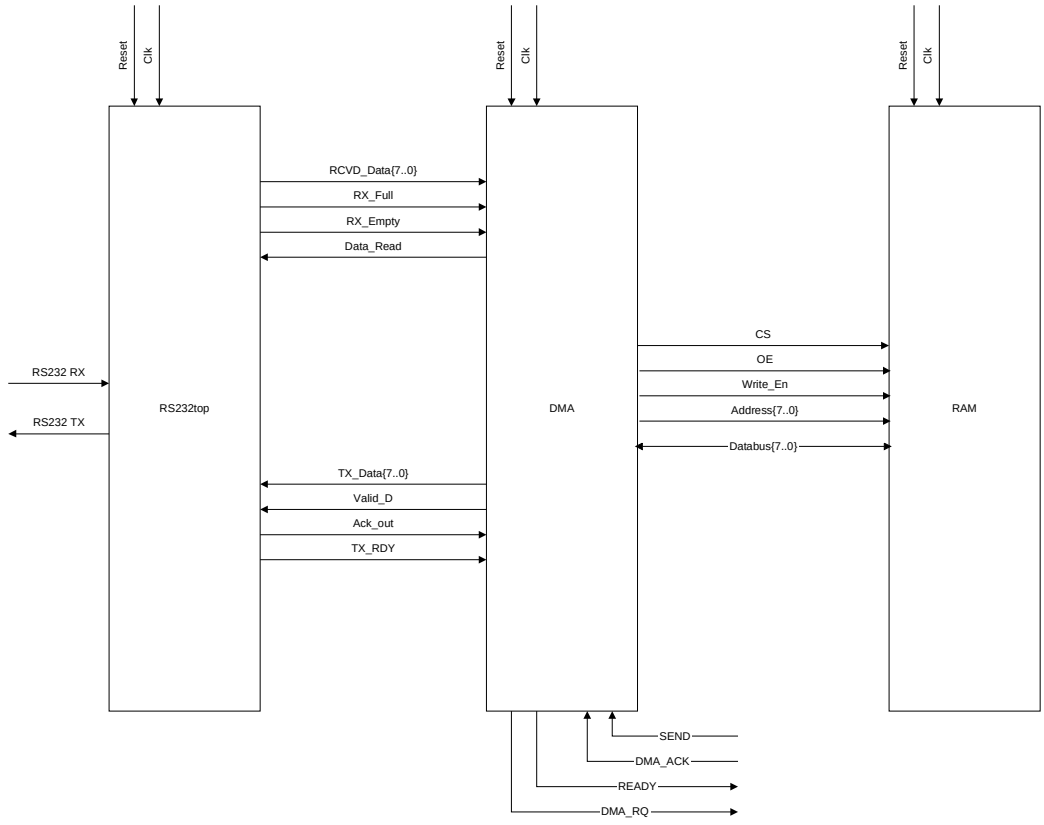

Figura 6. Conexión RS232-DMA-RAM

El bus de control (OE y Write_en) es un aspecto crítico del diseño. El control de estas señales recae principalmente sobre la unidad principal del procesador, debiendo cederlas (y dejarlas en alta impedancia) únicamente cuando el control DMA las requiera.

La funcionalidad de este bloque ha quedado ya suficientemente especificada, por lo que basta codificar la máquina de estados que gestione los procesos que se han explicado.

El controlador DMA debe constar al menos del fichero `DMA.vhd`. De todos modos, el alumno es libre de subdividir las tareas que ejecuta el componente y luego agruparlas en el fichero `DMA.vhd` por medio de subcomponentes.

8.4. Control principal

El control principal es el encargado de interpretar el código del programa guardado en la ROM de instrucciones. De forma resumida su tarea es la del núcleo de todo procesador: recoger la instrucción de memoria (*fetch*), decodificarla (*decode*) y ejecutarla (*execute*). La ejecución de dicha instrucción pasa por la generación de las microinstrucciones necesarias y/o la recogida de una nueva palabra de la memoria de programa (caso de las instrucciones de salto o de movimiento de datos). Además, debe tenerse en cuenta que se debe comprobar el acceso directo a memoria y ofrecer los buses al controlador DMA siempre que no se esté en medio de una ejecución de instrucción.

Para facilitar la implementación de la decodificación de instrucciones, se ofrecen las constantes que definen una instrucción como parte del paquete `PIC_pkg.vhd`. Para la correcta implementación de esta unidad es conveniente recordar cómo es la estructura básica de una CPU. Una CPU contiene internamente diversos registros que le ayudan en su función ejecutora. Algunos de ellos (los mínimos para que la máquina de control sea realmente efectiva) son:

- Un *contador de programa*: que memoriza la dirección de la memoria ROM por la que se va ejecutando el programa, controlando el acceso secuencial o los saltos que se produzcan.
- Un *registro de instrucciones*: recuérdese que la primera fase de ejecución de una instrucción es su lectura de la memoria de programa. Mediante este registro se puede mantener internamente una copia de la instrucción que se está tratando de ejecutar.

El aspecto más especial de esta CPU es que debe detenerse durante los accesos directos a memoria, así como durante los ciclos de envío de datos por la línea serie. Los accesos directos a memoria no pueden limitarse directamente por parte de la CPU, pero siempre es recomendable que el diseñador del *software* limite al máximo el número de envíos por línea serie. Un envío por línea serie a 115.200 bps es siempre mucho más lento que el procesador, cuya frecuencia de funcionamiento hemos fijado en 20 MHz.

Teniendo todo lo anterior en cuenta, **el diseño de la CPU es una máquina de estados** que reproduce el comportamiento especificado con la ayuda de los registros internos indicados.

8.5. Memoria de programa

La memoria de programa definitiva que debe emplearse para comprobar la funcionalidad del sistema se proporcionará en un fichero `ROM.vhd`. Sin embargo, y a fin de facilitar la labor de depurado del código, existe también un pequeño compilador al que se le da como argumento un fichero escrito en el lenguaje ensamblador de este procesador. Este compilador se encarga de generar código binario (o su representación según las constantes definidas en el paquete `PIC_pkg.vhd`) en un fichero intermedio y, si no se han producido errores, se genera una ROM en VHDL.

El compilador en cuestión, `pic_compiler`, acepta uno o dos parámetros. El primero de ellos es el fichero que contiene el código fuente y el segundo el archivo en el que se copiará la ROM generada. Si no se especifica el segundo parámetro, el compilador redirige la salida directamente al flujo de salida estándar (*stdout*). Si se producen errores durante la compilación del código, se puede acceder al fichero `output_tmp_file`, creado automáticamente por el compilador, y que contiene las instruc-

ciones que se han leído hasta el momento. Nótese que el lenguaje ensamblador definido es *case sensitive*, por lo que es necesario tener cuidado con las mayúsculas al escribir un programa.

Para asegurar el correcto funcionamiento del compilador, los ficheros de código deben acabar con una línea en blanco.

8.6. Fichero 'top' del sistema

Como interfaz estándar para todos los grupos de prácticas se ofrece el fichero `PICtop.vhd`, que inicialmente define los puertos de entrada y salida generales del procesador. En esta entidad serán instanciados los diversos componentes del sistema según éstos sean implementados.

```
entity PICtop is
  port (
    Reset    : in  std_logic;           -- Asynchronous, active low
    Clk      : in  std_logic;           -- System clock, 20 MHz, rising_edge
    RS232_RX : in  std_logic;           -- RS232 RX line
    RS232_TX : out std_logic;           -- RS232 TX line
    Switches : out std_logic_vector(7 downto 0);  -- Switch status bargraph
    Temp_L   : out std_logic_vector(6 downto 0);  -- Less significant figure of T_STAT
    Temp_H   : out std_logic_vector(6 downto 0)   -- Most significant figure of T_STAT
  );
end PICtop;
```

8.7. Ficheros de pruebas

A fin de facilitar la construcción de ficheros de pruebas se ofrece un nuevo paquete definido en el fichero `RS232_test.vhd`. Este paquete contiene código VHDL no sintetizable, válido únicamente para simulación. El contenido inicial de este paquete es el procedimiento `Transmit()`, que admite un byte de datos y lo envía en formato RS232 por la línea que se especifique.

```
procedure Transmit (
  signal   TX   : out std_logic;                    -- serial line
  constant DATA : in  std_logic_vector(7 downto 0)  -- byte to send
);
```

Como parte del desarrollo de la práctica **se requiere la implementación de al menos los siguientes bancos de pruebas**:

- Un fichero `tb_DMA.vhd` que contenga pruebas del subistema compuesto por el nivel físico RS232, el controlador de DMA y la RAM.
- Un fichero `tb_ALU.vhd` que contenga pruebas de la unidad aritmético-lógica por separado del resto del procesador.
- Un fichero `tb_PICtop.vhd` que contenga pruebas del procesador completo y que simule el envío de algunos comandos de usuario.

9. Desarrollo de la práctica y metodología

Los grupos de laboratorio deberán entregar un sistema que cumpla los requisitos especificados. Además deberá hacerse entrega del código correspondiente a los bancos de pruebas (*testbenches*) que se hayan desarrollado para verificar el funcionamiento de cada una de las partes que componen el sistema.

A modo de guía se sugiere que se proceda del siguiente modo:

- Escritura del código para la memoria RAM del sistema (fichero `RAM.vhd`) y de un banco de pruebas que verifique su funcionamiento con detalle.
- Escritura del controlador DMA (fichero `DMA.vhd`).

- Desarrollo de un banco de pruebas que haga uso del procedimiento `Transmit()` incluido en el paquete `RS232_test` para comprobar el funcionamiento de la terna RS232-DMA-RAM.
- Implementación de la unidad aritmético-lógica (fichero `ALU.vhd`) y de un banco de pruebas que verifique su funcionamiento por separado del resto del sistema.
- Implementación del control principal del procesador (`MAIN_CONTROL.vhd`).
- Desarrollo de pequeños programas para verificación de que la máquina de control funciona correctamente con las diversas instrucciones.
- Llegados a este punto es posible unir todos los subsistemas hasta ahora generados y realizar una síntesis a fin de comprobar su correcta interconexión.
- Generación de ficheros de pruebas para comprobar el funcionamiento del sistema completo con programas simples.
- Realización de simulaciones adicionales a fin de comprobar el funcionamiento del procesador con el programa completo.
- Repetición de las pruebas anteriores en simulación post-route.
- Síntesis final del sistema y comprobación de funcionalidad sobre la FPGA.

Hay que recordar que, durante la escritura de cada módulo, debe realizarse el proceso de síntesis periódicamente para asegurar que el código generado es sintetizable.

La secuencia propuesta para la realización de la práctica se indica en la Tabla 15.

	Sistemas desarrollados
Etapa 1	RAM y decodificación de direcciones, pruebas
Etapa 2	DMA integrado con RS232 y RAM, pruebas
Etapa 3	ALU, pruebas
Etapa 4	Máquina de control integrada con el resto, pruebas
Etapa 5	Simulación post-route, pruebas en placa

Tabla 15. Secuencia de desarrollo propuesta

10. Posibles mejoras del sistema

A fin de ofrecer una idea de la clase de mejoras que se pueden realizar en el desarrollo de esta práctica, se subrayan aspectos susceptibles de mejora:

- Experimentación sobre la interfaz RS232: se propone que los alumnos investiguen otros modos de transmisión, como un menor número de bits de datos, bits de paridad, diferentes velocidades de transmisión, etc.
- Investigación de posibles paralelismos en la CPU, es decir, la posibilidad de solapar ciclos de recogida de instrucciones con la ejecución de las anteriores. Se valorará que se estudien con atención los casos de problemas de paralelismo, como son las instrucciones de salto condicional con condición sin resolver.
- Estudio del paralelismo de la unidad DMA, de forma que las transmisiones y las recepciones sean simultáneas. Esto implica dividir la máquina de control DMA y construir un protocolo de negociación para la posesión del bus de datos entre los subsistemas receptor y transmisor.

Nótese que las mejoras aquí propuestas han de servir como ejemplo para que el alumno investigue e implemente cualquier otra mejora que añada una funcionalidad adicional significativa al sistema.

11. Apéndice 1. Código completo del programa ensamblador

```
; ZONA DE CONSTANTES ASCII

I          :       X49 ; CODIGO ASCII PARA LA I
A          :       X41 ; CODIGO ASCII PARA LA A
C          :       X43 ; CODIGO ASCII PARA LA C
T          :       X54 ; CODIGO ASCII PARA LA T
S          :       X53 ; CODIGO ASCII PARA LA S
E          :       X45 ; CODIGO ASCII PARA LA E
R          :       X52 ; CODIGO ASCII PARA LA R
O          :       X4F ; CODIGO ASCII PARA LA O
N          :       X4E ; CODIGO ASCII PARA LA N
K          :       X4B ; CODIGO ASCII PARA LA K
F          :       X46 ; CODIGO ASCII PARA LA F

; ZONA DE CONSTANTES PARA MEMORIA

RCBUF0     :       X00 ; DIRECCIONES BUFFER RECEPCIÓN
RCBUF1     :       X01
RCBUF2     :       X02
NINST      :       X03

TXBUF0     :       X04 ; DIRECCIONES BUFFER TRANSMISIÓN
TXBUF1     :       X05

SWBASE     :       X10 ; BASE PARA LA ZONA DE SWITCHES

LEVBASE    :       X20 ; BASE PARA LA ZONA DE ACTUADORES

GPMEM      :       X40 ; BASE DE LA MEMORIA GENERAL
TMP        :       X41 ; UN BYTE PARA CALCULOS

; COMIENZO DEL CÓDIGO, ESPERA ACTIVA POR
; VIGILANCIA DE NINST

#INIT      LD          .A, [NINST]
           LD          .B, XFF
           CMPL
           JMPT        #INIT

; LLEGA UNA NUEVA INSTRUCCIÓN POR LÍNEA
; SERIE, DECODIFICADOR

           LD          .ACC, X00
           WR          NINST
           LD          .A, [RCBUF0]
           LD          .B, A
           CMPE
           JMPT        #LEVER
           LD          .B, I
           CMPE
           JMPT        #SWITCH
           LD          .B, T
           CMPE
           JMPT        #TEMP
           LD          .B, S
           CMPE
           JMPT        #SND
           JMP         #ERR

; COMPROBACION DEL DECODIFICADOR

#OUT       LD          .ACC, O
           WR          TXBUF0
           LD          .ACC, K
           WR          TXBUF1
           SEND
           JMP         #INIT
```

```
; RUTINA DE ATENCIÓN A LOS SWITCHES

#SWITCH    LD         .A, [RCBUF1]
           ASCII2BIN
           LD         .INDEX, .ACC
           LD         .A, .ACC
           LD         .B, X07
           CMPG
           JMPT       #ERR ; debería saltar a ERR
           LD         .A, [RCBUF2]
           ASCII2BIN
           LD         .A, .ACC
           LD         .B, X01
           CMPG
           JMPT       #ERR ; debería saltar a ERR
           WRI        SWBASE
           JMP        #OUT

; RUTINA DE ATENCION A LOS ACTUADORES

#LEVER     LD         .A, [RCBUF1]
           ASCII2BIN
           LD         .A, .ACC
           LD         .INDEX, .ACC
           LD         .B, XFF
           CMPE
           JMPT       #ERR
           LD         .A, [RCBUF2]
           ASCII2BIN
           LD         .A, .ACC
           LD         .B, X09
           CMPG
           JMPT       #ERR
           WRI        LEVBASE
           JMP        #OUT

; RUTINA DE ATENCION AL TERMOSTATO

#TEMP      LD         .A, [RCBUF1]
           ASCII2BIN
           LD         .A, .ACC
           LD         .B, X02
           CMPG
           JMPT       #ERR
           LD         .B, X00
           ADD
           SHIFTL
           SHIFTL
           SHIFTL
           SHIFTL
           WR         TMP
           LD         .A, [RCBUF2]
           ASCII2BIN
           LD         .A, .ACC
           LD         .B, XFF
           CMPE
           JMPT       #ERR
           LD         .B, [TMP]
           ADD
           WR         TSTAT
           JMP        #OUT

; ENVÍO DE INFORMACIÓN

#SND       LD         .A, [RCBUF1]
           LD         .B, A
           CMPE
           JMPT       #LEVSND
           LD         .B, I
           CMPE
           JMPT       #SWSND
```

```
          LD         .B, T
          CMPE
          JMPT       #TSND
          JMP        #ERR

; ENVIANDO INFO DE UN ACTUADOR

#LEVSND   LD         .A, [RCBUF2]
          ASCII2BIN
          LD         .A, .ACC
          LD         .INDEX, .ACC
          LD         .B, X09
          CMPG
          JMPT       #ERR
          LDI        .A, [LEVBASE]
          BIN2ASCII
          WR         TXBUF1
          LD         .ACC, A
          WR         TXBUF0
          SEND
          JMP        #INIT

; ENVIANDO INFO DE UN SWITCH

#SWSND    LD         .A, [RCBUF2]
          ASCII2BIN
          LD         .A, .ACC
          LD         .INDEX, .ACC
          LD         .B, X07
          CMPG
          JMPT       #ERR
          LDI        .A, [SWBASE]
          BIN2ASCII
          WR         TXBUF1
          LD         .ACC, S
          WR         TXBUF0
          SEND
          JMP        #INIT

; ENVIANDO INFO DEL TERMOSTATO

#TSND     LD         .A, [TSTAT]
          LD         .B, B11110000
          AND
          SHIFTR
          SHIFTR
          SHIFTR
          SHIFTR
          LD         .A, .ACC
          BIN2ASCII
          WR         TXBUF0
          LD         .A, [TSTAT]
          LD         .B, B00001111
          AND
          LD         .A, .ACC
          BIN2ASCII
          WR         TXBUF1
          SEND
          JMP        #INIT

#ERR      LD         .ACC, E
          WR         TXBUF0
          LD         .ACC, R
          WR         TXBUF1
          SEND
          JMP        #INIT
```