



Práctica II

**Carlos Gómez Osuna
Marisa López Vallejo
Angel Fernández Herrero
Antonio Lomeña Moreno**

INDICE

| | |
|---|-----------|
| 1. Objetivos..... | 3 |
| 2. Contenidos..... | 3 |
| 3. Resumen del estándar RS232 | 4 |
| 4. Descripción de los comandos del terminal | 4 |
| 5. Descripción de la arquitectura y de la interfaz con el exterior | 5 |
| 6. Descripción funcional de los bloques del sistema | 6 |
| 6.1. Interfaz física RS232 | 6 |
| 6.2. Arquitectura de la memoria de datos | 7 |
| 6.3. Controlador DMA | 8 |
| 6.4. Unidad aritmético-lógica (ALU) | 10 |
| 6.5. ROM de programa | 11 |
| 6.6. Control principal..... | 11 |
| 7. Instrucciones del procesador | 12 |
| 7.1. Tipo 1, instrucciones relativas a la ALU | 12 |
| 7.2. Tipo 2, instrucciones de salto | 13 |
| 7.3. Tipo 3, instrucciones de movimiento de datos | 14 |
| 7.4. Tipo 4, instrucciones especiales | 15 |
| 8. Consideraciones adicionales para la implementación..... | 15 |
| 8.1. Memoria RAM | 15 |
| 8.2. ALU | 16 |
| 8.3. Controlador DMA | 17 |
| 8.4. Control principal..... | 18 |
| 8.5. Memoria de programa | 18 |
| 8.6. Fichero ‘top’ del sistema | 19 |
| 8.7. Ficheros de pruebas | 19 |
| 9. Lenguaje ensamblador | 19 |
| 9.1. Formato de los datos..... | 19 |
| 9.2. Definición de constantes..... | 20 |
| 9.3. Definición de etiquetas | 20 |
| 9.4. Registros | 20 |
| 9.5. Comentarios..... | 20 |
| 10. Desarrollo de la práctica y metodología | 20 |
| 11. Posibles mejoras del sistema..... | 21 |
| 12. Bibliografía recomendada..... | 21 |
| 13. Apéndice 1. Recomendaciones finales en el estilo de descripción para síntesis | 22 |
| 14. Apéndice 2. Código completo del programa ensamblador | 23 |

Prototipo de microcontrolador de aplicación específica

1. Objetivos

En su primera fase de prototipado, un microcontrolador puede ser implementado en VHDL [2] 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.). Como consecuencia de la radical simplificación del procesador se ha suprimido el control de interrupciones, por lo que la llegada de nuevos datos por medio de los periféricos debe comprobarse dentro del programa ensamblado.

El prototipo que nos ocupa debe ser capaz de recibir órdenes simples a través de una interfaz serie RS232 [1], 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 sistema de 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. En esta fase preliminar, la interfaz serie estará regida en su otro extremo por un PC con salida serie que ejecute 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

Para la implementación física del sistema desarrollado se dispone de una placa de prototipado XSB-300E de Xess Corp. [5], que incluye entre otros periféricos un adaptador de niveles para RS232.

2. Contenidos

En esta práctica se pretende construir 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 [1]. La velocidad de transmisión puede configurarse desde 9600 bps hasta 115200 bps.

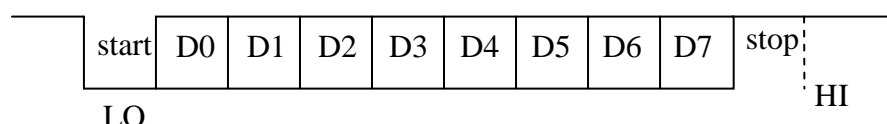


Figura 1. Protocolo de envío RS232

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 115200 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.

La arquitectura específica del controlador del nivel físico se describe con mayor detalle junto con el resto de los bloques que componen el sistema. Más información sobre el protocolo serie RS232 puede encontrarse en la bibliografía complementaria [1].

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 mayúscula, 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.

En el caso de los interruptores puede enviar UP o DW (según el caso), en el de los actuadores su nivel en hexadecimal, y en el del termostato el valor cargado.

Para la decodificación de estos comandos **se proporcionará a los alumnos un programa escrito en el ensamblador del procesador**, y que constituye una versión simple del programa que deberá ejecutar el sistema. Si los alumnos desean modificar y mejorar este programa, se pondrá a su disposición un software que compile el código y lo convierta en una ROM para 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.

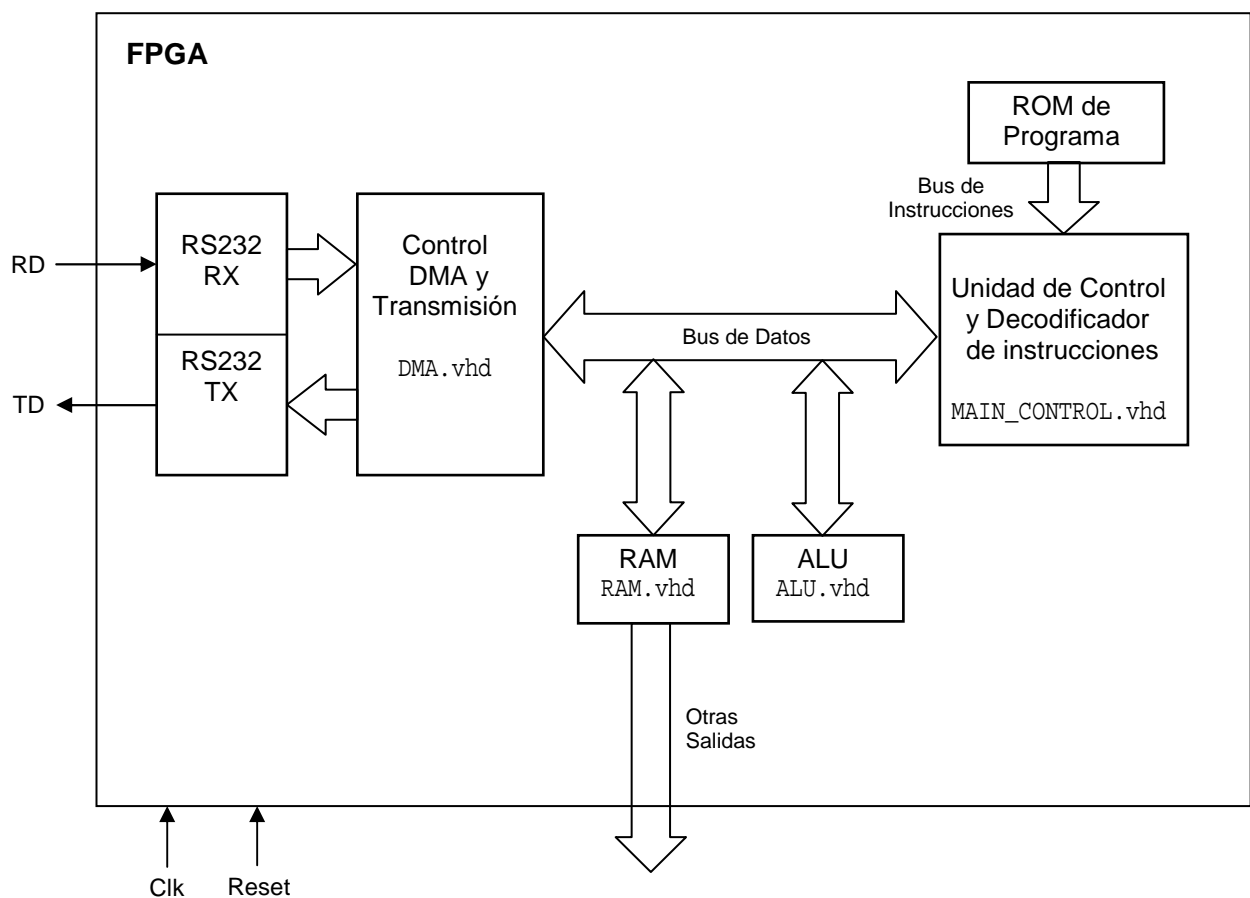


Figura 2. Diagrama de bloques de alto nivel

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 el Ejercicio 7, se caracteriza por:

- Recepción y transmisión de datos a 115200 bps.
- Formato de trama de 8 bits de datos, sin paridad y con un bit de parada.
- Capacidad de almacenamiento de datos recibidos de hasta 64 bytes mediante una memoria FIFO interna.
- Líneas de estado de la memoria interna de recepción: indicadores de memoria llena y memoria vacía.
- Entrega de datos recibidos por una interfaz paralelo y síncrona mediante una línea de Data_read.
- Captura de datos mediante una interfaz asíncrona basada en una línea de validación de datos (Valid_D) y una de reconocimiento de carga (ACK_in).
- Línea de estado del transmisor: 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 con el reloj. |
| 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 que se ponga a nivel bajo la línea ACK_in. La línea TX_RDY se ofrece a la salida como información redundante por si fuera necesario hacer uso de ella para ejecutar otra tarea. La temporización de la entrega de datos del DMA al transmisor RS232 se muestra en la Figura 3.

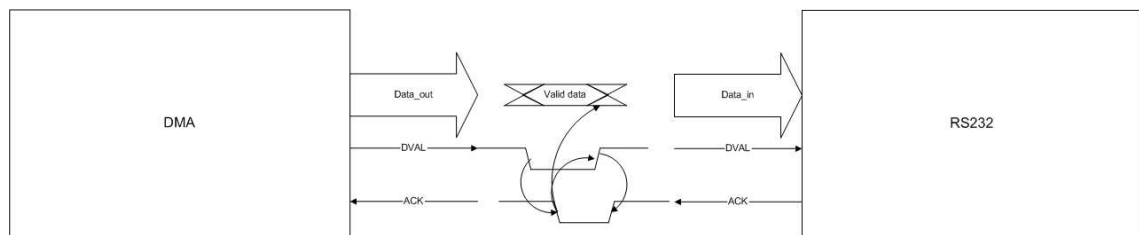


Figura 3. Protocolo de entrega de datos al transmisor RS232

6.2. Arquitectura de la memoria de datos

En este primer prototipo del microcontrolador utilizaremos una entidad RAM que funcionará a modo de “cajón de sastre”. En esta fase preliminar, dicha entidad no sólo implementará la memoria RAM de propósito general, sino también una serie de posiciones de memoria (registros de E/S) que simulan los diferentes periféricos que el sistema completo debe controlar, así como las posiciones de memoria empleadas por el controlador de acceso directo a memoria. No contentos con eso, vamos a añadir una serie de líneas de salida que aprovechan los recursos disponibles en el hardware del laboratorio, a fin de visualizar el estado de alguno de esos 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 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 (20MHz) |
| Databus[7..0] | Bidireccional | Bus de datos del sistema |
| Address[7..0] | Entrada | Direcciones del bus de datos |
| CS | Entrada | 'Chip Select' |
| 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 la temperatura del termostato (formato 7 segmentos) |
| Temp_H[6..0] | Salida | Dígito más alto del valor de la temperatura del termostato (formato 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 del sistema. Switches[7..0] debe recoger el estado de los interruptores mapeados en memoria para su presentación en la barra de LEDs de que dispone el hardware del laboratorio. Del mismo modo, las líneas Temp_X[7..0] deben presentar la conversión del valor guardado en el termostato de forma que sea inteligible 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 esta longitud. En concreto la RAM del procesador reserva los bytes 0x00 a 0x02 a tal efecto. El controlador es responsable de escribir el valor 0xFF en el registro NEW_INST de la RAM cada vez que se haya entregado un comando completo.

- Por otro lado, el controlador se encarga de cargar la interfaz física de transmisión serie con los datos que se han de transmitir de vuelta al PC. El sistema está preparado para que los códigos que se envíen sean de 2 bytes, por lo que el transmisor tendrá que hacer dos accesos a memoria (2 ciclos de petición-entrega de bus). El comienzo de la emisión de una pareja de bytes es iniciado con una instrucción especial del procesador que pone a '1' un registro interno del controlador. Este registro debe volver al estado '0' al finalizar la transmisión.

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 (20MHz) |
| 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 llegada de datos al transmisor RS232 |
| 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 |
| CS | Salida | 'Chip select' de la RAM |
| 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 procesador se encuentre totalmente ocioso |

Tabla 5. Líneas de entrada y salida del controlador DMA

Queda patente que las líneas del controlador están pensadas para servir de separación total del procesador y su memoria respecto de las limitaciones físicas y de temporización de las líneas RS232.

El acceso directo a memoria ocurre del siguiente modo:

1. El controlador coloca un '1' en la línea DMA_RQ y esperar a que el procesador conceda los buses.
2. El procesador concede los buses y coloca un '1' en la línea DMA_ACK.
3. Se realiza la lectura o escritura en la RAM de acuerdo con la temporización que ésta ofrece.
4. Coloca un '0' en la línea DMA_RQ para devolver el control de los buses.
5. El procesador continúa la ejecución y coloca un '0' en la línea DMA_ACK.

La transmisión se lleva a cabo siguiendo el siguiente esquema:

1. El procesador pone un '1' en la línea SEND_comm, lo que fuerza que la línea READY se coloque a nivel bajo inmediatamente.
2. El controlador realiza la carga y transmisión de los 2 bytes, uno tras otro. Durante el proceso completo el controlador está en posesión del bus de datos.
3. Una vez terminada la transmisión de los datos, el controlador coloca a nivel alto la línea READY, con lo que 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. 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 para la decodificación de estados. El funcionamiento de dichas señales de estado se especifica en la Tabla 6. La información de su uso y las instrucciones que los modifican se encuentra junto con la especificación de las instrucciones del procesador.

| Flag | Descripción |
|------|--|
| Z | 'Zero', vale '1' si el resultado de la última operación fue cero o la última comparación fue verdadera |
| C | Bit de acarreo de la suma, vale '1' cuando la suma o la resta se desbordan |
| N | Bit de acarreo de 'nibble', vale '1' cuando existe acarreo entre los 'nibbles' (medio byte) |
| E | Error, vale '1' cuando se llega a algún resultado inesperado en la ALU |

Tabla 6. Descripción de los flags del procesador

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

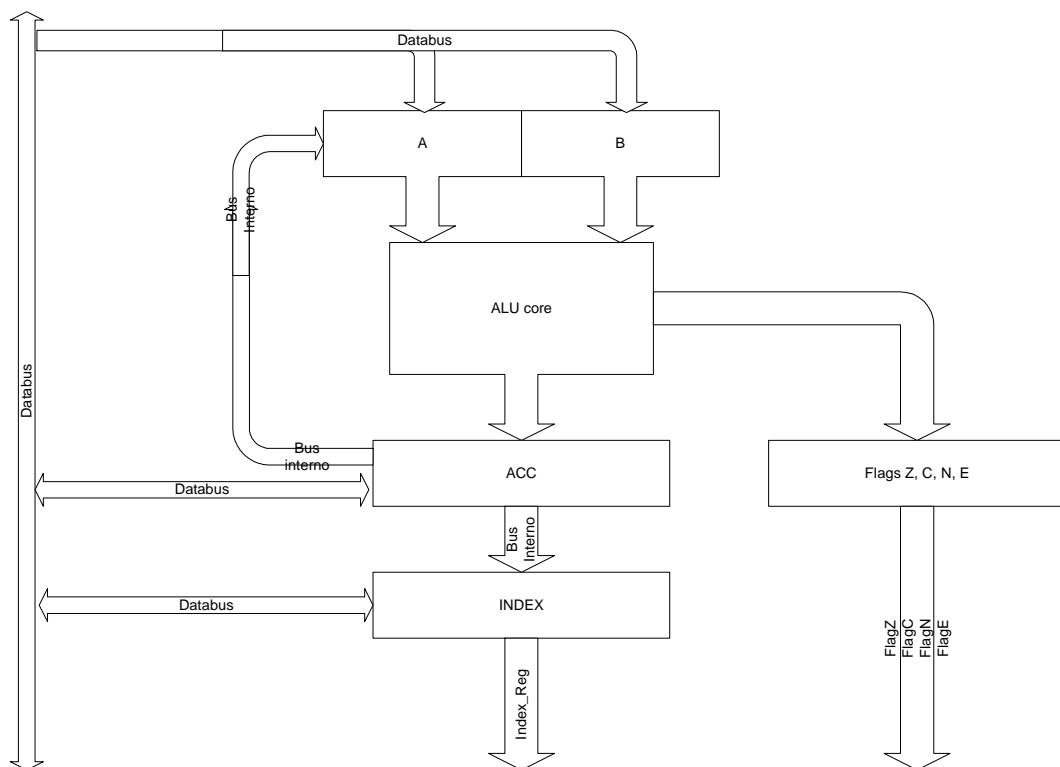


Figura 4. Ruta de datos de la ALU

Las operaciones que debe realizar la ALU se especifican más adelante, junto con el resto 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 todos los registros internos a 0x00 |
| Clk | Entrada | Reloj principal del sistema, 20MHz |
| 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 de índice a la unidad decodificadora 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).

Respecto de 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 grabado 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).

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 |
|-----------------|---------|--|
| INS_Addr[11..0] | Entrada | Bus de direcciones de la ROM de programa |
| INS_Bus[11..0] | Salida | Bus de instrucciones |

Tabla 8. Líneas de entrada y salida de la ROM

Dado que el bus de recogida de instrucciones es de sólo 12 bits, puede ser necesario acceder dos veces a la memoria de programa para la total ejecución de una instrucción. Esta decisión recae en manos del control principal del procesador.

Por último, nótese que sólo es posible direccionar un máximo de 4096 palabras de programa. Esto, aunque pueda parecer en exceso 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 añadirle 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.

Las tareas de este bloque pueden resumirse como sigue:

- Si durante el estado inicial de la máquina el controlador DMA está pidiendo los buses, concedérselos inmediatamente y detenerse hasta que el DMA acabe su acceso a memoria.
- Recoger una dirección de memoria y decodificar la instrucción contenida.
- Ejecutar la instrucción, para lo que puede ser necesario generar microinstrucciones, realizar un acceso a la memoria de datos (RAM), o incluso recoger una nueva palabra de la memoria de programa.
- Si la instrucción en concreto es del tipo 4 (SEND), puede ser necesario congelar ('stall') el procesador hasta que la instrucción se lleve a cabo.
- Regresar al estado inicial.

Las líneas de 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, 20MHz |
| ROM_Data[11..0] | Entrada | Bus de datos de la memoria del 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_CS | Salida | 'Chip select' de la RAM |
| RAM_Write | Salida | Microinstrucción para escritura en la RAM |
| RAM_OE | Salida | Microinstrucción para permitir la salida de datos 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 al 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 lenguaje ensamblador del microcontrolador está compuesto por cuatro tipos de instrucciones de longitud variable. A continuación se especifica cada uno de estos tipos y la funcionalidad de las instrucciones. La sintaxis del lenguaje propiamente dicho está desarrollada en el epígrafe 9. Lenguaje ensamblador.

Es de notar que, en las palabras de memoria que definen instrucciones, la información necesaria para decodificarlas se encuentra en los 8 bits menos significativos, siendo el resto cero. Esto es así para permitir la posterior ampliación del juego de instrucciones a nuevos tipos y nemónicos.

Para comprender el formato binario de estas instrucciones es necesario tener en cuenta las constantes definidas en el paquete PIC_pkg (fichero PIC_pkg.vhd). Esas constantes se usan posteriormente en el pequeño compilador asociado al sistema de modo que la ROM generada pueda ser comprendida.

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 | Gira hacia la izquierda el contenido del acumulador, introduciendo un cero | |
| SHIFTR | Gira hacia la izquierda 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 (para números, devuelve FF si hay error) | E |
| BIN2ASCII | Convierte A del 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 germen del procesador, pero se implementan para facilitar su posterior incorporación a las funciones del mismo.

El formato binario de esta instrucción es simple. Los 4 bits más significativos están a 0 y después tenemos 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 Inmediata Etiqueta (#Nombre) | Salto incondicional |
| JMPTrue | Dirección Inmediata Etiqueta (#Nombre) | 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. Las etiquetas empleadas serán traducidas por el compilador en direcciones inmediatas.

El formato binario se define del siguiente modo:

- La primera palabra: los 4 bits más significativos están a 0 y después tenemos 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 2 sería "0000" & TYPE_2 & JMP_COND.
- La segunda palabra es la dirección de salto que ha sido traducida al formato adecuado por el compilador (el compilador las indica 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 In- mediata] | Carga los registros de la ALU |
| LDI | .A .B .INDEX .ACC | [Constante] [Dirección In- mediata] | Carga los registros en la ALU, utiliza el registro .INDEX como índice para acceder a memoria |
| WR | Dirección in- mediata Constante | | Escribe datos en memoria |
| WRI | Dirección in- mediata Constante | | Escribe datos en memoria, utiliza el registro .INDEX como índice para acceder a memoria |

Tabla 12. Instrucciones de tipo 3

Estas instrucciones pueden variar su longitud. 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 sumándolo a la dirección de memoria indicada de modo que, si se va a acceder a la memoria de datos, se puedan leer y escribir tablas de forma sencilla.

A la memoria principal se puede acceder por medio de constantes definidas con anterioridad. Nótese sin embargo que las constantes se pueden definir desde 0 a 4095, pero a la hora de hacer uso de estas 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:

- La 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.
- La 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 código máquina y descripción pueden verse en la Tabla 13.

| Código | Traducción | 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 (8 bits menos significativos) |
| 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 | Para el procesador y ejecuta una petición de envío 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 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. Esta señal se provee como ayuda para agilizar la inicialización del sistema y reducir la memoria de programa, pero podría eliminarse y realizar esta función con software. Para la implementación de este bloque se ofrece como ejemplo el archivo RAM.vhd, que constituye una memoria de propósito general de 16 bytes.

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. La memoria de carácter general deberá construirse en una entidad aparte (basta con modificar adecuadamente el fichero RAM.vhd), siendo luego instanciada en el bloque principal de la memoria.

La ventaja de trabajar con la memoria RAM genérica por separado es que la herramienta de síntesis la reconocerá como tal, y será capaz de optimizarla haciendo un mejor uso del hardware. Esto no ocurre sin embargo con la zona de memoria con señal de Reset, para la que el sintetizador deberá hacer uso de flip-flops discretos hasta completarla. Evidentemente, el alumno debe ser capaz de decodificar la dirección especificada para acceder a uno u otro banco de memoria.

La RAM dispone de líneas de salida destinadas a mostrar el estado de alguno 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 el ‘nibble’ bajo y alto del registro T_STAT (dirección 0x31) a un formato legible en un display de 7 segmentos. Esto puede conseguirse de forma sencilla construyendo una ROM con una instrucción del tipo with-select (aunque existen otras maneras más o menos elegantes de conseguir el mismo resultado). Para la construcción de estas funciones de conversión, téngase en cuenta que las líneas de los visualizadores están ‘mapeados’ con respecto a las líneas de salida según lo indicado en Figura 5.

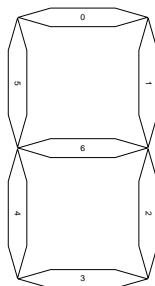


Figura 5. Líneas de los displays de 7 segmentos

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

Como apoyo para la codificación de la RAM se han definido algunos tipos de datos en el paquete PIC_pkg (contenido en el archivo PIC_pkg.vhd). Este paquete contiene, aparte de los ya mencionados tipos, otras funciones y tipos destinados a facilitar el desarrollo de la práctica.

8.2. ALU

La implementación de la ALU resulta bastante inmediata. Basta con seguir la ruta de datos indicada anteriormente. Para desarrollar de una forma cómoda el conjunto de las microinstrucciones que debe recibir la ALU, se ofrece el tipo alu_op como parte del paquete PIC_pkg.vhd. Este tipo no está completo, pero da una idea bastante adecuada de cómo debe procederse para hacer uso de un tipo enumerado. Si se hiciera uso de este tipo, la cabecera de la entidad de la ALU sería:

```
entity ALU is
  port (
    Reset      : in    std_logic;    -- asynnnchronous, active low
    Clk        : in    std_logic;    -- Sys clock, 20MHz, 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 **este bloque 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 control que forman parte del sistema es la encargada de gestionar la carga de los datos recibidos por la línea serie en la memoria. Su tarea básica es “escuchar” las líneas de EMPTY del receptor RS232 y Send_Comm procedente del control principal del procesador.

Para el diseño de este bloque pueden tomarse dos aproximaciones igualmente válidas, bien puede desmembrarse la máquina de control en dos mitades (recepción y transmisión) y añadir algunas señales de negociación entre ellas; o bien se puede optar por la codificación de una única máquina de estados que englobe ambos procesos.

Un esquema de la conexión se muestra en la Figura 6. Las líneas CS, 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.**

El bus de control (CS, 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 quedarse en alta impedancia) únicamente cuando el control DMA así lo requiera.

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

El controlador DMA debe constar de al menos un fichero vhd, el 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 componentes.

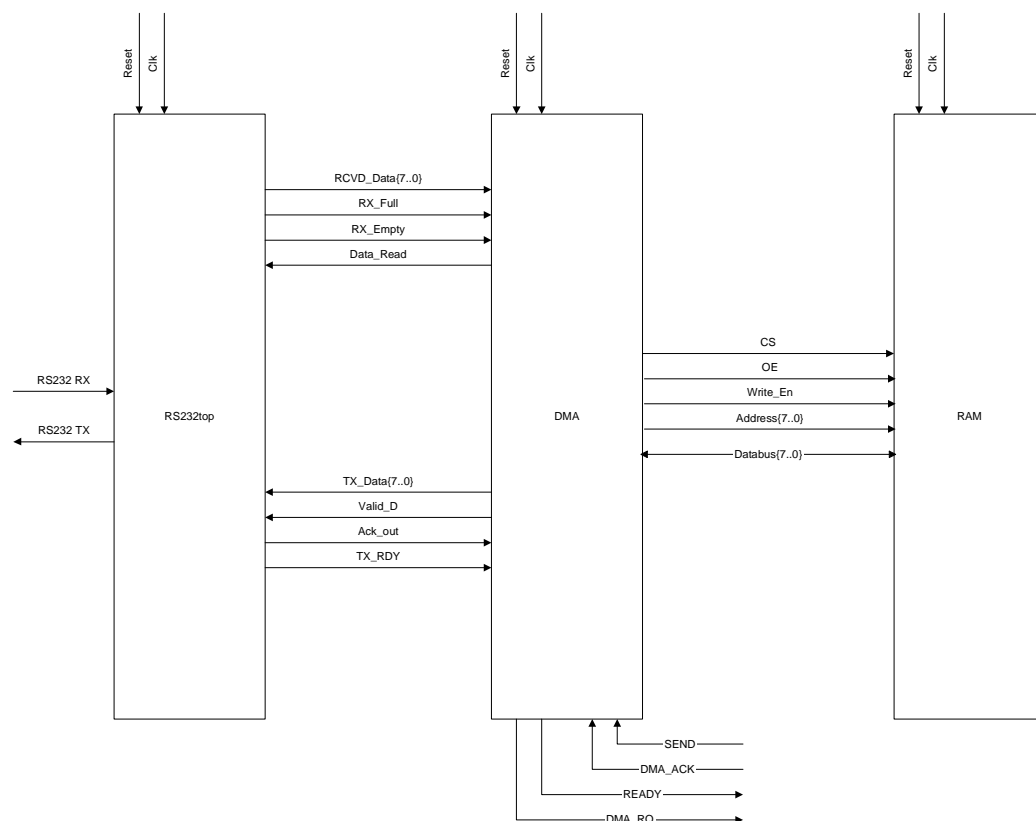


Figura 6. Conexión RS232-DMA-RAM

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 controlar 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 controla el acceso secuencial o los saltos que se produzcan en la lectura del programa.
- 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.
- Un *registro de almacenamiento temporal*: que no resulta imprescindible, pero que facilita las operaciones en las que sea necesario volcar un contenido de la memoria de programa a otro registro, sea éste interno o externo.

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 115200 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 control que reproduce el comportamiento especificado con la ayuda de los registros internos indicados y que haga especial hincapié en el exhaustivo control del bus de control de la memoria de datos.

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 vhd.

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 instrucciones 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. Como ya se comentado, la especificación del lenguaje ensamblador se da en el Apartado 9. Lenguaje ensamblador.

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 por la línea que se especifique en formato RS232.

```
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. Lenguaje ensamblador

Para definir completamente el lenguaje ensamblador es necesario añadir ciertas consideraciones. Para facilitar la comprensión de la sintaxis del lenguaje ensamblador se incluye el código que al finalizar la práctica el procesador debe ejecutar con éxito.

9.1. Formato de los datos

El compilador acepta datos en formato hexadecimal, decimal y binario, siempre y cuando estén dentro de los límites definidos por el ancho de la palabra. Esto limita los valores válidos a números sin signo con un valor máximo de 4095.

- Los valores en hexadecimal van precedidos del carácter 'x' o del carácter 'X'.
- Los valores en decimal se escriben por su valor, sin ningún prefijo.
- Los valores en binario van precedidos del carácter 'b' o del carácter 'B'.

9.2. Definición de constantes

Las constantes se pueden definir en cualquier momento a lo largo del programa y son cadenas de letras y números iniciadas siempre por una letra. Para definir una constante se debe seguir el siguiente formato:

NOMBRECONSTANTE : VALOR

Donde VALOR puede ser un dato en cualquiera de los formatos aceptados.

9.3. Definición de etiquetas

Las instrucciones de salto aceptan únicamente valores en hexadecimal o etiquetas como direcciones válidas de salto.

Una etiqueta puede definirse en cualquier línea del programa excepto en una que contenga una instrucción de salto (para evitar incoherencias en la construcción de un programa). Para ello, basta seguir el siguiente formato:

#NOMBREETIQUETA

Donde NOMBREETIQUETA es una cadena de letras y números.

El compilador se encargará de traducir las etiquetas en direcciones efectivas.

9.4. Registros

Los registros definidos en el procesador pueden accederse según se indica en Tabla 15.

| Registro del procesador | Indicador para el compilador |
|-------------------------|------------------------------|
| A | .A |
| B | .B |
| Acumulador | .ACC |
| Índice | .INDEX |

Tabla 15. Indicadores de registros para el compilador

9.5. Comentarios

Se pueden insertar comentarios en líneas nuevas o al final de una instrucción. Los comentarios van siempre precedidos del carácter ‘;’.

10. Desarrollo de la práctica y metodología

Los grupos de laboratorio deberán entregar un sistema que cumpla los requisitos arriba 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 de 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-PAR.
- 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 16.

| | 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-PAR, pruebas en placa |

Tabla 16. Secuencia de desarrollo propuesta

11. 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.

12. Bibliografía recomendada

- [1] Christopher E. Strangio, *The RS232 Standard. A Tutorial with Signal Names and Definitions*, CAMI Research Inc, 1993.
- [2] Sundar Rajan, *Essential VHDL – RTL Synthesis Done Right*, S&G Publishing, 1998.
- [3] Xilinx, www.xilinx.com
- [4] Xilinx, *Spartan-II 1.8V FPGA Family*, 2003.
- [5] Xess Corporation, *XSB Board V1.0 Manual*, 2003.

13. Apéndice 1. Recomendaciones finales

Finalmente, es importante recordar que en la realización de todo diseño existen una serie de recomendaciones en el estilo de descripción, provenientes tanto del propio grupo de trabajo como de los fabricantes de las herramientas de diseño. En este apartado se incluyen algunas de las recomendaciones típicas que deben ser tenidas en cuenta por cualquier diseñador de circuitos:

- Recomendaciones referentes a la determinación y evaluación de la arquitectura:
 - Cualquier diseño está compuesto por un conjunto de elementos interconectados, los cuales han de realizar la funcionalidad requerida por la entidad (especificaciones de funcionamiento). La cantidad de elementos que sea posible utilizar estará relacionada con las restricciones de área del sistema.
 - Para cada uno de esos elementos, el diseñador debe especificar el momento en que se han de realizar las operaciones (especificaciones de velocidad y temporización de cada uno de los componentes).
 - **Es importante definir cómo realizar la comunicación entre los distintos componentes o bloques del sistema** (especificaciones de interfaz). Dentro de este grupo de especificaciones debe quedar completamente detallado el formato que utiliza cada una de las señales: si es activa nivel alto o bajo, el número de bits enteros y totales, etc., y en qué momento su valor es válido.
 - Es importante recordar también que el sistema ha de superar una serie de pasos de verificación (especificaciones relativas al test del circuito), por lo que éste debe ser fácil de entender, testear y modificar en el momento que se necesite.
- Recomendaciones referentes a la descripción VHDL del circuito:
 - El estilo de descripción debe ser fácilmente legible y mantenible. En este sentido, en la escritura de los ficheros VHDL se recomienda, entre otras consideraciones, que se distinga la utilización de palabras clave de los nombres definidos por el usuario mediante el uso de mayúsculas y minúsculas, que se utilicen nombres significativos según la función que realicen los distintos elementos, que los ficheros contengan una correcta estructura y comentarios, y que el número de caracteres utilizados en cada línea no exceda de 80.
 - Utilización de las funciones estándar de librería. Además de hacer más simple la lectura y mantenimiento del código, contienen elementos estructurados de uso común que han sido ya verificados.
 - En diseños en los que tanto el tiempo como el área son parámetros críticos es necesario recurrir a los componentes del fabricante. La descripción inicial requiere algo más de tiempo, pero los elementos están ya optimizados y verificados, por lo que las prestaciones obtenidas son bastante mayores.
- Recomendaciones referentes al proceso de síntesis:
 - Es importante conocer y distinguir cuándo utilizar señales y cuándo variables. Las primeras requieren mayor cantidad de memoria, y por tanto ralentizan las simulaciones. Las segundas, por el contrario, dejan una mayor libertad a la herramienta de desarrollo, pero son menos cercanas a la síntesis, y por tanto suelen producir más problemas al utilizar estas herramientas.
 - **Genera todos los componentes del sistema utilizando la misma señal y flanco de reloj.** De esta forma se hace más fácil la verificación por parte de la herramienta de desarrollo y se evitan posibles fallos en la temporización de los componentes en las etapas finales, más complejas, de la síntesis.
 - Para concluir, recuerda que aunque las herramientas CAD facilitan y reducen el tiempo de diseño de los distintos componentes, lo hacen a costa de cierta cantidad de área. Por esta razón, **es importante tener siempre en mente la arquitectura que hay detrás de cada descripción, y comprobar que las inferencias que realiza la herramienta de diseño corresponden con las deseadas.**

14. Apéndice 2. 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, 0
        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
           CMPL
           JMPT    #ERR ; debería saltar a ERR
           LD      .A, [RCBUF2]
           ASCII2BIN
           LD      .A, .ACC
           LD      .B, X01
           CMPL
           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
           CMPL
           JMPT    #ERR
           WRI     LEVBASE
           JMP     #OUT
```

; RUTINA DE ATENCION AL TERMOSTATO

```
#TEMP      LD      .A, [RCBUF1]
           ASCII2BIN
           LD      .A, .ACC
           LD      .B, X02
           CMPL
           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
        CMPE
        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
        CMPE
        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

```