MEMORIA MICROCONTROLADOR

Realizado por:
Germán Bravo López
Iñaki Ormaetxea Orobengoa

6 DE ENERO DE 2019
LCSE - MÁSTER UNIVERSITARIO EN INGENIERÍA DE SISTEMAS ELECTRÓNICOS

Tabla de contenido

1. Introducción 2
2. Objetivos 2
3. Descripción de la arquitectura 2
4. Bloques del sistema 3
   4.1. Interfaz RS232 3
   4.2. Memoria RAM 3
   4.3. Controlador DMA 4
   4.4. Memoria ROM 5
   4.5. ALU 5
   4.6. CPU 6
   4.7. Fichero ‘top’ 7
5. Bancos de pruebas 7
6. Mejoras 8
   6.1. Paralelismo de la unidad DMA 8

1. Introducción

Tras la realización de la interfaz de comunicación RS232 en la práctica anterior, se ha procedido en esta práctica a la creación de un microcontrolador de propósito específico. Este dispositivo, que se va a implementar en VHDL, consta de una serie de módulos principales para su funcionamiento:

- Interfaz RS232: Para recibir órdenes y poder responder a ellas.
- Memoria RAM: Para almacenar los datos recibidos.
- Controlador DMA: Que se encarga de repartir los buses según la tarea a realizar.
- Memoria ROM: Contiene la memoria del programa (instrucciones a realizar).
- ALU: Para la realización de las operaciones pertinentes.
- CPU: Microprocesador que se encarga de ejecutar las instrucciones de programa.

De esta forma se procede a explicar los objetivos y cada módulo creado.

2. Objetivos

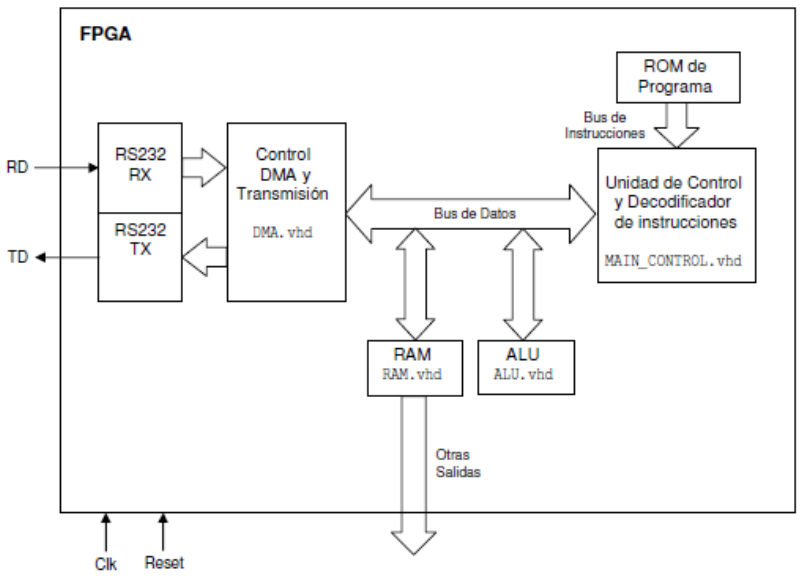
El propósito del microcontrolador es decodificar comandos que se le entregan por puerto serie y actuar según lo que se le solicite.

Para ello, se podrán realizar las siguientes funciones:

- Debe controlar 8 interruptores que encienden o apagan 8 LEDs en la FPGA.
- Cargar un valor de temperatura en un registro dedicado para el control de un termostato, el cual se mostrará por dos displays de 7 segmentos.
- Enviar información sobre el estado de alguno de los periféricos.

3. Descripción de la arquitectura

Este sistema tiene una arquitectura tipo Harvard, donde los buses e instrucciones están separados. Para comprender mejor el funcionamiento explicado, en la siguiente figura se observa un diagrama de bloques a alto nivel que facilita el entendimiento.



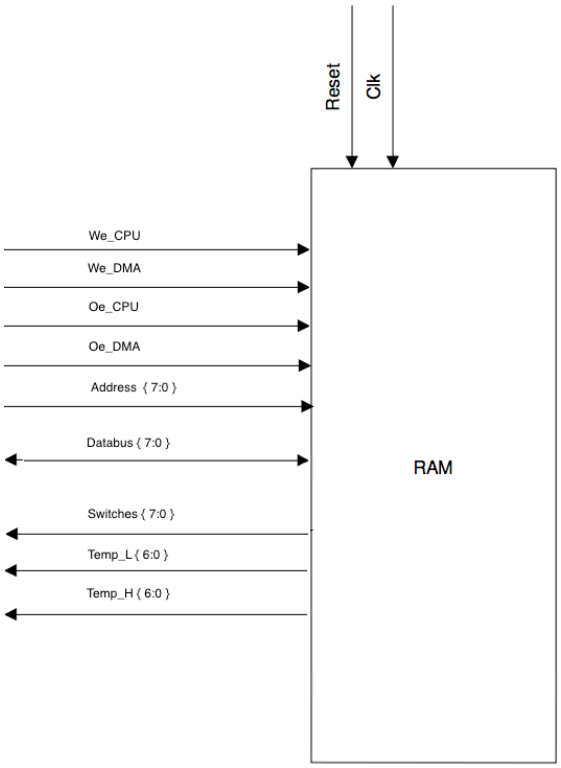
4. Bloques del sistema

4.1. Interfaz RS232

Este módulo (*RS232top.vhd*) se realizó en la práctica anterior y se utiliza en esta como interfaz de comunicación. El único cambio realizado en este es la extracción del convertidor de reloj, ya que este será el encargado de proporcionar la señal de reloj a todos los componentes del sistema.

4.2. Memoria RAM

Se ha diseñado una RAM (*RAM.vhd*) síncrona single-port que servirá como memoria de datos para nuestra CPU global. Esta es la encargada de almacenar temperaturas, estado de los LEDs y es la primera parada para las instrucciones que introduzca el usuario. Tiene un tamaño de 256 palabras de 8 bits cada una y está dividida en una RAM específica y otra genérica en su interior.



La entrada y la salida de los datos se hace mediante el mismo bus de datos, al estar este compartido habrá que tener cuidado con no hacer lecturas y escrituras a la vez. Tendrá una entrada para el *address*, dos entradas de *output_enable*, una proveniente de la DMA y otra de la CPU. En la lógica de los mismos, no ocurrirá en ningún momento que los dos hagan una petición de lectura, cuando uno lo haga, el dato a leer se pondrá en la salida. La escritura se hará desde el *databus* también, con dos señales de *write_enable*, en este caso también empleando la misma lógica.

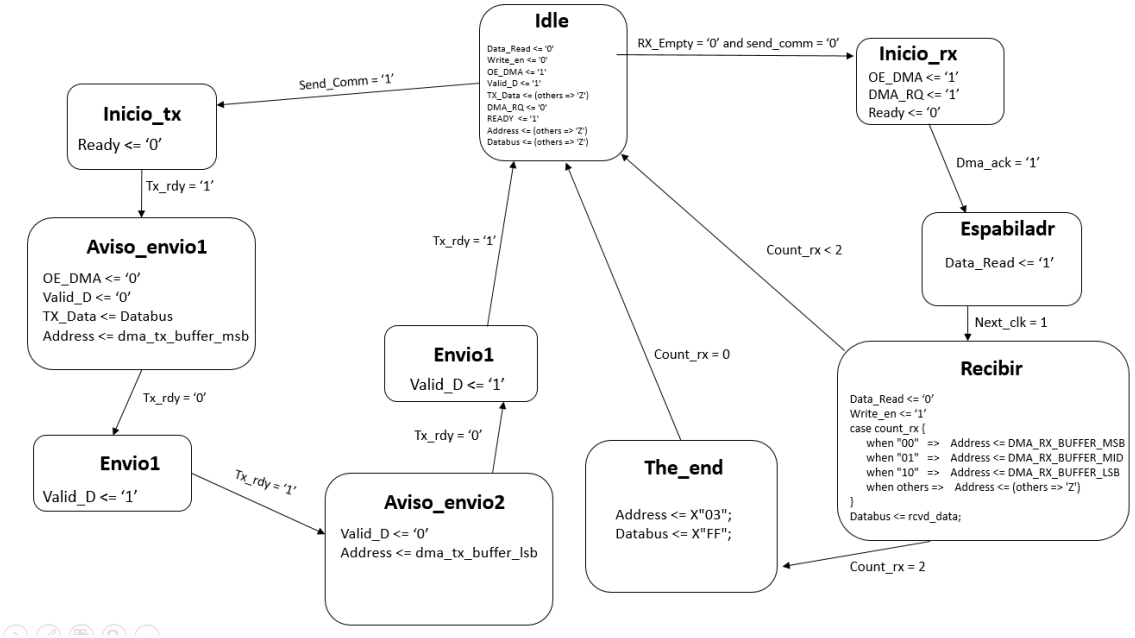
Hay otras salidas auxiliares para conocer el valor de la temperatura y el estado de los LEDs. También tiene una entrada para el reset y otra para el reloj.

Funciona de la siguiente forma:

- Al comienzo de la ejecución toda la RAM será reseteada con valores a '0', menos la temperatura, que se reseteará al valor 20 en decimal.
- Luego, según le vayan llegando órdenes de escritura o lectura, se ejecutarán dichas operaciones síncronamente con el reloj. Habrá un *chip_select* interno que según el *address* activará la parte genérica o especifica del bloque diseñado.

4.3. Controlador DMA

La DMA o Direct Memory Access (*DMA.vhd*) permite a diferentes módulos acceder a la memoria "independientemente" de la CPU. En nuestro caso esto se utilizará para los accesos que necesita el bloque RS232.



Funciona de la siguiente forma, cuando se exija el envío de un dato por parte de la CPU a través del transmisor, la DMA se encargará de gestionar los accesos a memoria necesarios por parte del RS232 para que envíe los datos adecuados. Se puede decir que en este momento la DMA no depende de la CPU para dejar los buses al RS232. Se esperará a que el RS232 envíe los dos bytes, sincronizando las etapas, las direcciones, etc., con el flag de *Tx_Rdy*.

Estando en estado idle, si recibe el aviso de que la FIFO del receptor ya no está vacía (equivalente a que ha llegado algún dato), le concederá los buses a este. Esto se dará si la CPU no está queriendo enviar algo. La DMA hará un "request" de los buses, y una vez recibido que se han concedido (por ello no es totalmente independiente de la CPU, tiene que saber lo que está haciendo la CPU para que no haya dos accesos simultáneos), lo que se haya recibido en el receptor se guardará en la RAM. Esto se hará con los tres datos que se deberían de recibir, dado que una instrucción requiere de tres recepciones. Una vez que la DMA ha visto que los tres han llegado, escribirá un registro de la RAM para que la

CPU sepa que una nueva instrucción ha llegado. Después de este punto el trabajo de la DMA habrá terminado, y volverá al estado de espera.

La ejecución que sigue se muestra en el diagrama previo. Se han hecho mejoras respecto a este diseño que quedarán argumentadas en la sección de mejoras.

4.4. Memoria ROM

El contenido de este bloque (*ROM.vhd*) ha sido proporcionado por el profesor, contiene la memoria de programa en forma de constantes. Tiene dos puertos de comunicación, una entrada y una salida:

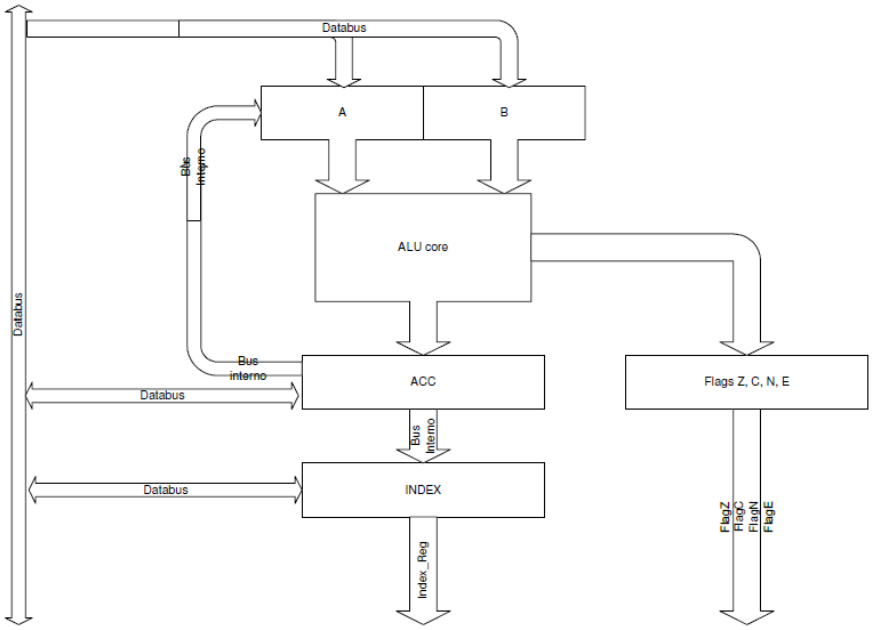
- Entrada (INS_Addr), con la que se solicita por parte de la CPU la dirección de la instrucción.
- Salida (INS_Bus), por el cual se manda el contenido de la instrucción solicitada

Es un bloque totalmente combinacional, proporciona el contenido de la instrucción en el mismo instante solicitado.

4.5. ALU

Este bloque (*ALU.vhd*) es el encargado de realizar las operaciones de suma, resta, comparación, etc., que se puedan solicitar mediante las instrucciones de programa.

Tiene dos registros para carga de operandos, un acumulador, un registro específico para carga de índices destinado al acceso indexado a memoria y un registro con 4 bits de estado (Z, C, N y E).



En esta imagen se observa la funcionalidad y el rutado del interior de la ALU. En ella todas las instrucciones que se le ordenan se ejecutan en un solo ciclo de reloj.

Para su realización nos apoyamos en el paquete de constantes proporcionado por el profesor donde se recogían todas las instrucciones que se podían solicitar.

```
TYPE alu_op IS (
      nop,                                  -- no operation
      op_lda, op_ldb, op_ldacc, op_ldid,    -- external value load
      op_mvacc2id, op_mvacc2a, op_mvacc2b,  -- internal load
      op_add, op_sub, op_shiftl, op_shiftr, -- arithmetic operations
      op_and, op_or, op_xor,                -- logic operations
      op_cmpe, op_cmpl, op_cmpg,            -- compare operations
      op_ascii2bin, op_bin2ascii,           -- conversion operations
      op_oeacc);                            -- output enable
```

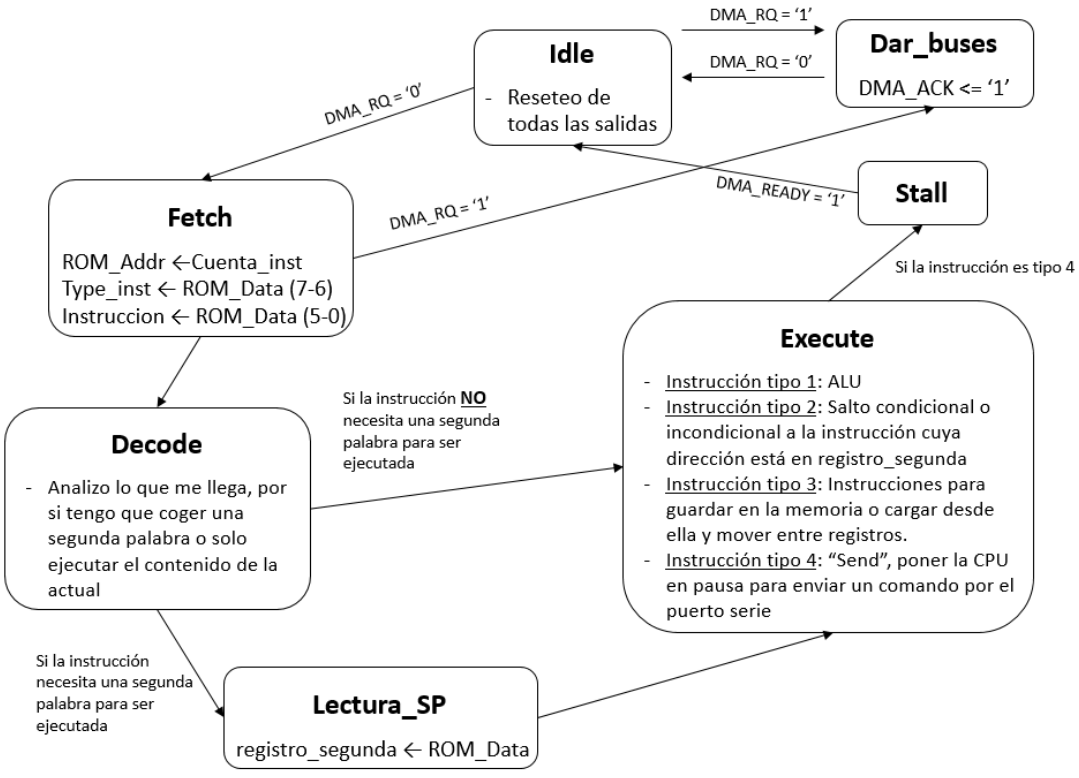
Gracias a esto y al funcionamiento de los flags según la instrucción, se realizó el diseño.

4.6. CPU

La CPU (*MAIN_CONTROL.vhd*) es el núcleo del sistema, ya que está máquina es la que se encarga de coger una instrucción, entender lo que significa y ejecutar su orden. Principalmente este bloque es capaz de realizar lo siguiente:

- Cuando la DMA solicite los buses, y en ese momento no se esté realizando alguna tarea crítica o que deba ser terminada, se le conceden hasta que termine con ellos.
- Recoger una instrucción de la dirección de memoria solicitada.
- Decodificar esta instrucción para saber qué hay que realizar.
- Ejecutarla, es decir general las microinstrucciones necesarias para resolver lo que se solicite en la instrucción recogida.
- Volver al estado inicial.

Para la construcción de este componente se ha realizado una máquina de estados que se muestra en la figura siguiente.



Acompañando a esta máquina de estados se ha incluido un contador de instrucciones para gestionar la lectura de las instrucciones y además poder realizar la lectura de la segunda palabra sin tener que volver a pasar por *Idle*.

Hay que tener en cuenta que el bus de datos se debe dejar en alta impedancia cuando no se utilice ya que esta señal se comparte con otros módulos que pueden cargar datos en él.

4.7. Fichero 'top'

Finalmente en este fichero (*PICtop.vhd*) se realizan las instanciaciones y los enrutamientos de los distintos módulos con la finalidad de conectar todas las partes del sistema completo. Para ello se definen señales para cada entrada y salida de los módulos independientes, de forma que queden conectados.

También se ha incluido en este fichero un proceso para poder pasar por los displays los valores de la temperatura que salen cableados directamente de la RAM. Como existen 8 displays en la FPGA, se deben apagar todos excepto el que se desee utilizar para mostrar un dígito u otro.

Mediante estas instrucciones se manda el dato del dígito de las unidades:

```
temp <= '1' & temp_l;
disp <= "11111110";
```

Mediante estas otras se manda el dato del dígito de las decenas:

```
temp <= '1' & temp_h;
disp <= "11111101";
```

Cada conjunto de instrucciones se ejecuta cada 50 microsegundos ya que se comparten salidas en la entidad.

5. Bancos de pruebas

En esta sección se resumen los bancos de pruebas realizados para comprobar el correcto funcionamiento del sistema completo.

- Fichero *tb_RAM.vhd*
- Fichero *tb_DMA.vhd*
- Fichero *tb_ALU.vhd*
- Fichero *tb_RS232_DMA_RAM.vhd*
- Fichero *tb_PICtop.vhd*

Con ayuda del fichero *RS232_test.vhd* se ha podido realizar la simulación del envío de comandos por línea serie a la interfaz de entrada al microcontrolador, en los dos últimos bancos de pruebas.

6. Mejoras

Tras la comprobación del *tb_PICtop.vhd* en la simulación "post-implementation timing" y su correcto funcionamiento en la FPGA se procedió a realizar una mejoras propuesta por el profesor.

6.1. Paralelismo de la unidad DMA

Esta modificación se realizó con la finalidad de dividir la carga de trabajo del bloque DMA en dos, es decir, que un solo bloque (*DMA_R*) se encargue de las recepciones y avise de que el bus lo tiene ocupado, para que la otra parte (*DMA_T*) no pueda usarlo para enviar.

Para ello se han incluido dos señales nuevas en cada módulo:

- *Flag_DMA_R*. Esta señal sale del módulo de control de la recepción y va al módulo de control de la transmisión. Cuando la recepción se encuentra ocupada, esta señal se pone a '1' y bloquea la *DMA_T* para que no pueda utilizar los buses de datos.
- *Flag_DMA_T*. Esta señal tiene exactamente la misma función que la anterior pero desde el punto de vista de la transmisión.

De esta forma el resultado son dos módulos independientes que se conectan en el fichero *PICtop.vhd* para que cada uno pueda desarrollar su función.