MEMORIA FINAL

Miembros del grupo: Rafael Fernández Parra y Álvaro Montesano Martínez.
Fecha: 10/01/2024

1) Introducción.

Partiendo de lo aprendido y desarrollado en la anterior entrega, en la cual hemos desarrollado en VHDL el protocolo RS232, desarrollaremos un sistema que usará este protocolo para la comunicación con dispositivos externos. En esta segunda y última entrega se nos ha propuesto desarrollar un microcontrolador de propósito específico. El cual tendrá una estructura simple y un juego de instrucciones reducido, que será capaz de controlar elementos accionables.

Este procesador tiene una arquitectura Harvard, por lo cual dispone de memoria de programa (ROM), cuya estructura se nos proporciona y otra de datos (RAM), con buses separados. Mediante la ROM y la Unidad de Control y Decodificador de instrucciones, podemos controlar el funcionamiento de la DMA, de la RAM y de la ALU, siendo estas 3 últimas estructuras las que tendremos que desarrollar como parte del trabajo a desarrollar en esta entrega.

Para comprobar el correcto funcionamiento del microcontrolador dispondremos de 8 interruptores, 10 actuadores de nivel y un termostato. Una vez recibidas las instrucciones por el puerto serie y comunicado al microcontrolador a través del RS232, este debe decodificar las instrucciones y tomar las medidas necesarias, comunicándose con los dispositivos externos mediante el puerto transmisor de RS232.

2) Retos técnicos superados

Retos técnicos superados	RS-232 + DMA + RAM	ALU	Sistema completo
Simulación Behavioral funcionando	Si	Si	No
Simulación con retardos funcionando	Si	Si	No
Síntesis sin errores	Si	Si	No
Sistema funcionando en placa	Si		No

Mejoras
Cambiar el tamaño de la palabra de 5 a 8 bits mediante un pulsador

2.1) Descripción del sistema

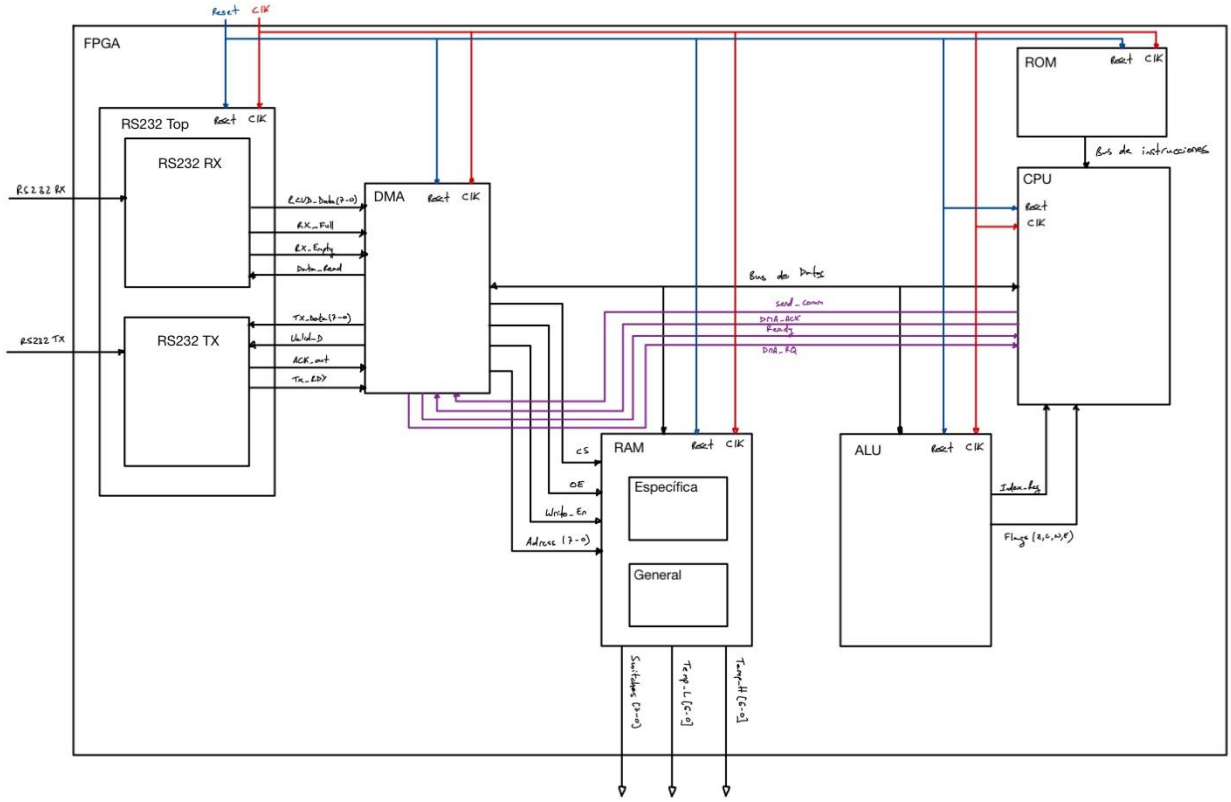

Figura 1: Diagrama de bloques de la terna.

Como puede verse en la Figura 1, este sistema está compuesto por diversos componentes. La parte inicial de este sistema, el RS232 TOP, es la que se encarga de la comunicación con el exterior para recibir y mandar las órdenes a realizar necesarias para el microcontrolador.

El RS232 Top está compuesto por el RS232 RX y el RS232 TX, RS232 RX se encarga de transmitir al módulo de la DMA los datos recibidos por el puerto serie y recibir por parte de la DMA la confinación de la lectura de dichos datos, mientras que el RS232 TX se encarga de recibir los datos a transmitir por el puerto serie, y enviar a la DMA el ACK de confirmación.

La DMA tiene dos modos de funcionamiento, transmisión y recepción. Trabajando en el modo recepción, comunica las órdenes recibidas del exterior a la CPU y escribe los datos requeridos en la RAM. Cuando está funcionando en modo transmisión recibe las instrucciones de la CPU, leyendo los datos deseados de la memoria RAM y trasladándolos al módulo de transmisión del RS232.

La RAM se encarga de almacenar los datos temporales que son enviados y recibidos por la DMA en dos divisiones dos divisiones internas, en la general se almacenará la información entrante y saliente del sistema, mientras que en la específica se encarga de almacenar variables específicas para el funcionamiento del sistema, controlando líneas de salida destinadas a mostrar el estado de registros, ya sea mediante un display o mediante switches.

La CPU se encarga de recoger las instrucciones de la ROM de programa y ejecutarlas, generando los microinstrucciones necesarios, las cuales envía tanto a la DMA para realizar la comunicación necesaria o a la ALU, para procesar las operaciones que sean necesarias realizar.

La ALU se encarga de realizar las operaciones requeridas por la CPU, mediante un juego reducido de instrucciones opera los datos proporcionados y devuelve la solución de estas, junto con diversos flags enviados a la CPU.

La DMA es la encargada tanto de escribir en la memoria los datos que se reciben del RS232, es decir, recepción y a su vez, se encarga de cargar la interfaz física de transmisión serie con las respuestas a los comandos de usuario, es decir transmisión.

En la recepción, la DMA tiene que solicitar los buses, ya que estos son utilizados por el sistema casi completo. Una vez se los han concedido, el controlador realiza la escritura en la RAM con los datos recibidos, y según termina de escribir devuelve los buses y baja un flag para indicar que ha acabado.

La DMA sabrá que tiene que transmitir cuando a través de su entrada “SEND_COM” recibe un 1, haciendo que la DMA baje el flag de READY para indicar que está realizando un proceso de transmisión. Una vez se ha realizado esto, el controlador transmite los 2 datos seguidos, ya que este tiene la posesión de los buses, y cuando termina de transmitir vuelve a poner a alto READY y retira la petición de transmisión.

Para abarcar correctamente este problema de diseño, se optó por hacer una gran máquina de estados, que a su vez se divide en dos, una para transmisión y otra para recepción. Por lo que la máquina se encontrará esperando algún cambio en sus señales para proceder a realizar las operaciones de transmisión o de recepción. Como se ve en la figura de abajo la máquina se encontrará en el estado Idle hasta que send_comm valga 1 o Rx_empty y send_comm valgan 0, donde la máquina pasará a Trasmisión o Recepción respectivamente.

Se ha realizado así el diagrama de estados debido a que sino salía muy grande.

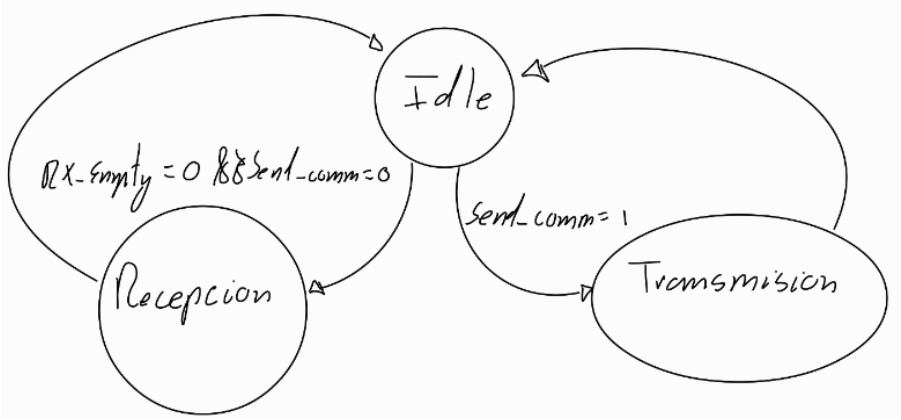

Figura 2: Diagrama de estados de la DMA general.

Siempre que en un estado no haya nada las variables se encontrarán como en el estado anterior, excepto en Idle que estarán de la siguiente manera: Valid_D=1, cnt_r=cnt_r, Data_Read=0, Write_en=0, OE=1, DMA_RQ=0, Ready=1, Databus=ZZZZZZZZ, TX_Data=00000000, Address=00000000.

En las figuras 3 y 4 se encuentran los diagramas de estado de recepción y trasmisión respectivamente.

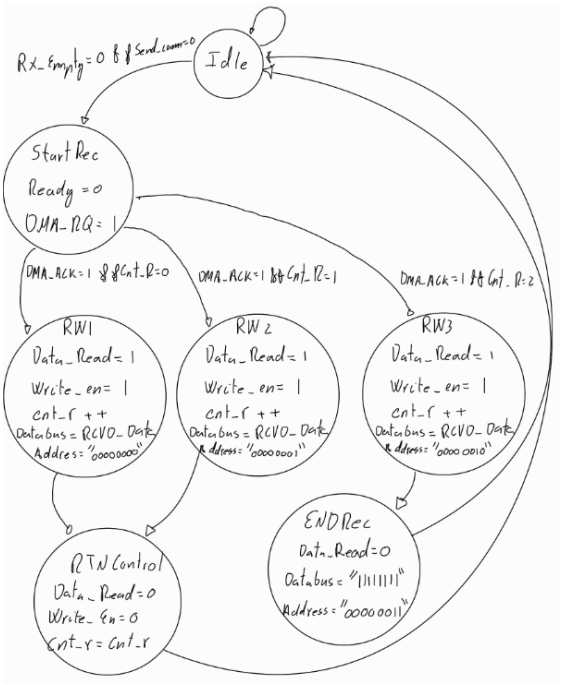

Figura 3: Diagrama de estados de recepción.

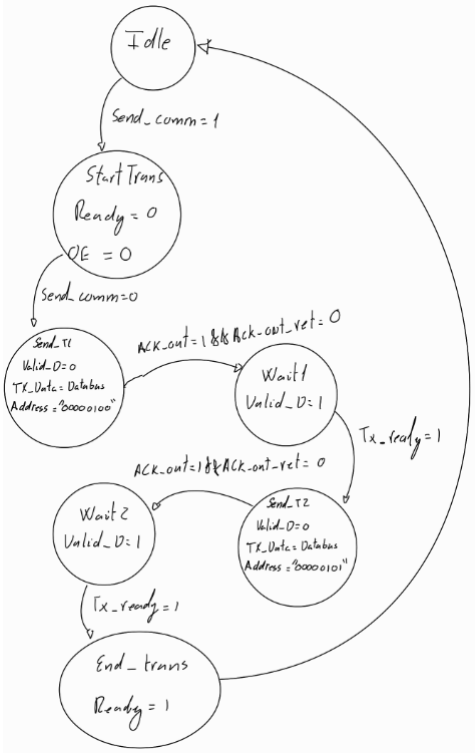

Figura 4: Diagrama de estados de transmisión de la DMA.

Cabe destacar que cada dato recibido vendrá numerado por un contador, y cada vez que se reciba uno se vuelve a Idle, por lo que, si solo han llegado dos bytes, se puede interrumpir la recepción y pasar a transmisión, una vez terminada la transmisión se vuelve a la recepción para recibir ese tercer dato.

En la recepción se tuvo que poner un detector de flancos para la señal ACK_out, debido a su funcionamiento. Para ello se implementó un registro el cual, retrasaba un ciclo de reloj esta señal y comparando ambas señales se obtenía la condición de transición buscada.

La RAM que se ha programado consiste en un bloque que contiene la RAM específica y la RAM general. La de propósito general tendrá los valores de la posición 0x40 hasta la 0xFF, y la de propósito específico va de las direcciones 0x00 a 0x3F. Estas memorias compartirán las señales de Data, Address, WE, OE y Data_out, por esta razón se ha implementado circuitos lógicos para que, en función de las entradas, las señales vayan dirigidas a una memoria o a otra. Esto se puede observar en la figura 5.

Por defecto se pone el valor 23 de temperatura, esto se pone en la dirección 49 de la memoria específica. Cabe destacar que solo la parte específica tiene RESET. Y que cuando no se está entregando un valor a la salida Databus se encontrará en alta impedancia.

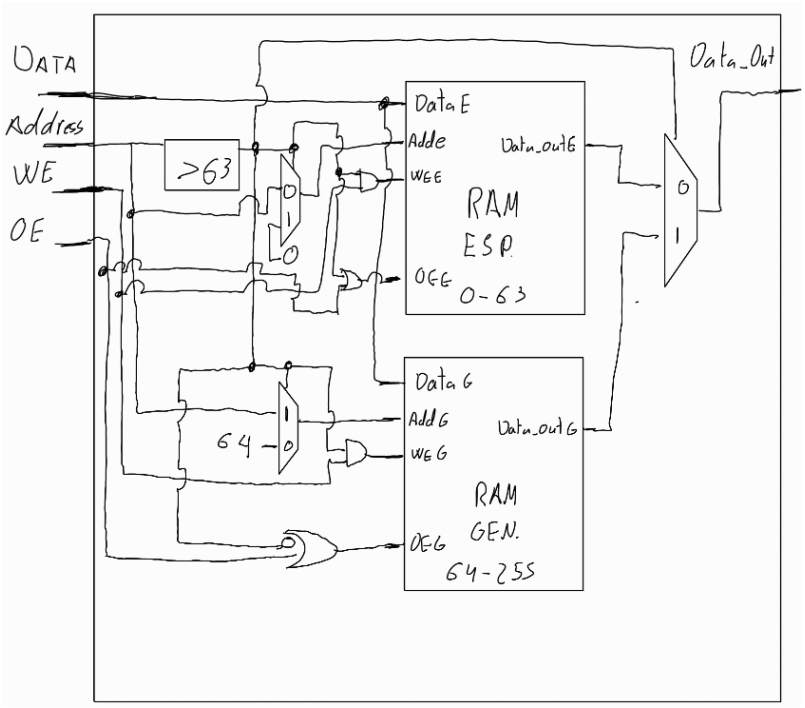

Figura 5: Diagrama de bloques de la memoria RAM.

El módulo de la ALU lo hemos afrontado mediante un proceso secuencial el cual depende de la señal clk y la señal reset, para que sea síncrona y se reinicie al mismo tiempo que el resto del sistema. Su diagrama de bloques se puede ver en la figura 6.

Para realizar este módulo hemos hecho uso del paquete PIC_PKG el cual nos proporciona las distintas operaciones disponibles mediante la variable de tipo Alu_op.

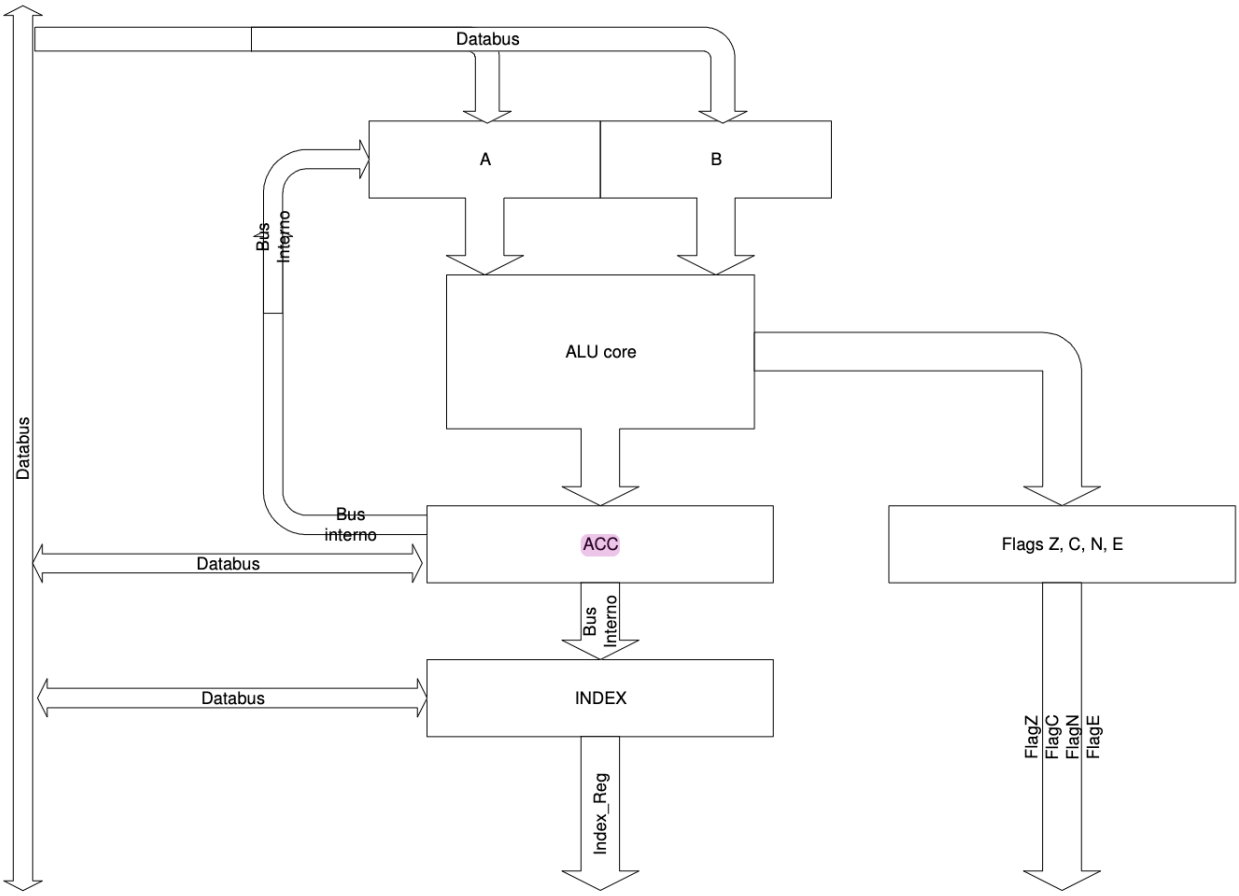

Figura 6: Diagrama de bloques de la ALU.

Para elegir entre las distintas operaciones, la correcta a realizar, hemos optado por la sentencia *case*, la cual nos permite seleccionar la operación en función de la señal u_instruction, la cual es de tipo Alu_op. Siendo los primeros casos a realizar la carga del Databus tanto en el registro A como en el B dependiendo de la instrucción. La operación desplazamiento, ya sea a la izquierda como a la derecha la hemos realizado mediante la concatenación de los 6 inferiores o superiores bits con un 0 a la derecha o a la izquierda, respectivamente.

A destacar sería el método usado para la activación del Flag Z y el Flag E. Hemos declarado que se active el Flag Z cuando el acumulador sea 0 o bien cuando un flag interno que hemos denominado FlagCMP este activo, para que así se active cuando el resultado es 0 o cuando las operaciones de comparación son verdaderas. El valor del flag FlagCMP se pone a 1 cuando cualquiera de las comparaciones realizadas es afirmativa. Para la activación del Flag E, hemos definido que este activo cuando el flag interno FlagCONV tenga valor 1. Hemos definido que el FlagCONV se active cuando se introduzca como datos a convertir de ASCII a Binario o de Binario a ASCII, caracteres fuera del rango permitido. Siendo este rango para ASCII los números en caracteres ASCII y para Binario los números del 0 al 9.

2.2) Simulaciones y prueba en placa

Para comprobar el correcto funcionamiento de la RAM, la DMA y el RS232 se probaron en conjunto. Para ello hay un testbench realizado para que la DMA transmite y envíe datos para que estos se

almacenen en la RAM. Los datos son enviados y recibidos por el RS232 (explicado en la entrega anterior).

Primero se establecen los valores de 0x34 y 0x6D en las posiciones 4 y 5 de la RAM, esto se realiza en el propio testbench. Una vez hecho esto, se comprueba la transmisión poniendo send_comm a 1, con esto se ve que funciona correctamente.

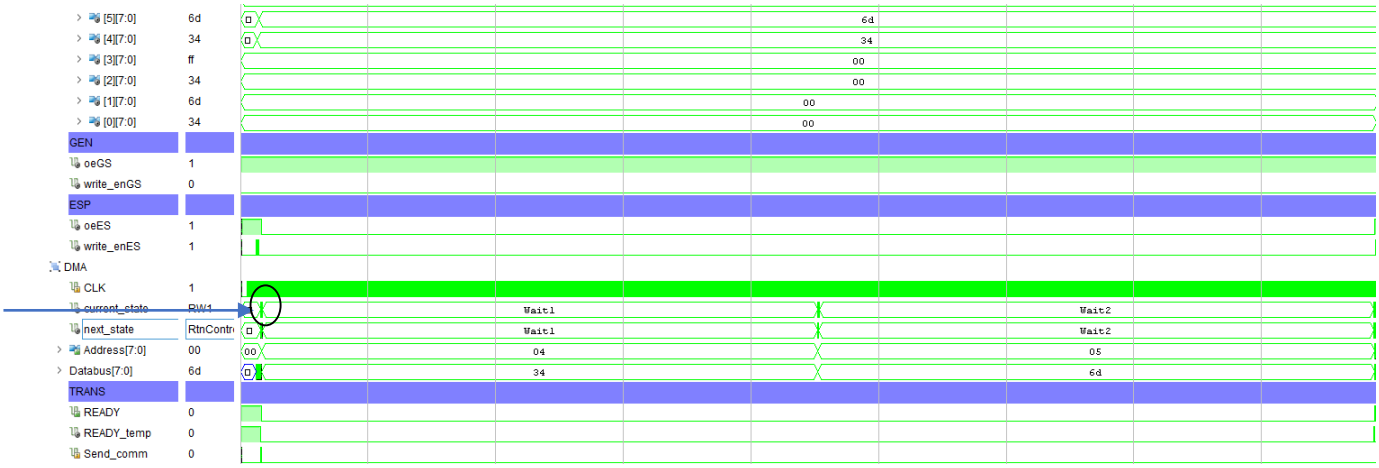

Figura 7: Comprobación de la transmisión en la DMA.

Se ve como cuando envía el primer dato, Address toma el valor de 0x04 y que con el segundo se pone a 0x05. El estado de wait como ya se esperaba dura muchos ciclos de reloj. Se puede apreciar perfectamente como Databus obtiene los datos almacenados en las correspondientes direcciones de memoria. En la siguiente Figura se hará zoom a la zona señalada, ya que al ser tan corta en comparación a Wait no se aprecia.

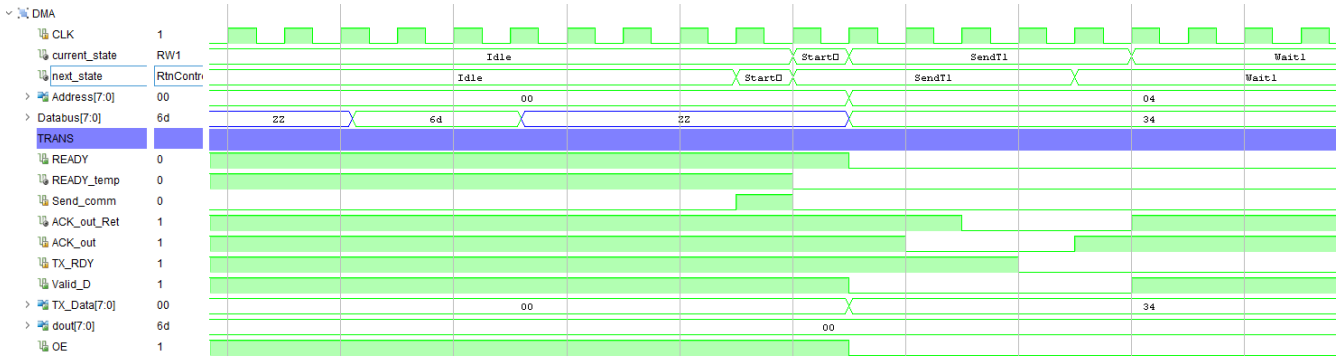

Figura 8: Comprobación de inicio transmisión en la DMA.

A continuación, en la figura de abajo se verá cómo se termina la transmisión y comienza la recepción. En este caso como solo se han enviado dos bytes, solo se recibirán dos datos, es por esto que la máquina de estados se queda esperando ese tercer dato.

Los datos 0x34 y 0x6D se almacenarán en las posiciones 0x00 y 0x01 como estaba previsto. Esto se puede observar en la figura 10. Aquí se ve cómo se habilita la escritura en la memoria que se quería y como en la misma se escribe el dato correspondiente.

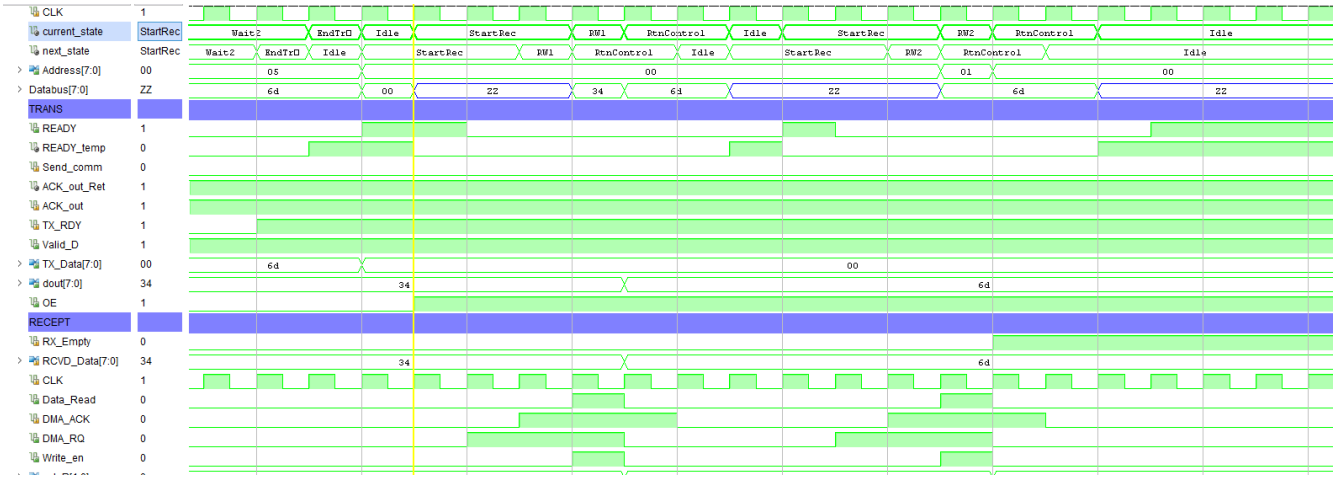

Figura 9: Comprobación de inicio transmisión en la DMA.

Figura 10: Comprobación de fin de la transmisión y recepción en la DMA.

A continuación, se va a observar en la siguiente figura como se envían otros dos datos, y al recibir el tercero, automáticamente se pondrá 0xFF en la dirección 0x03 de la memoria y que el cuarto dato remplazará el valor del que había en la dirección 0x00. En este caso en la dirección 0x00 se encontraba el valor de 0x34 y después de recibir el cuarto dato se encontrará el valor de 0x6D.

Todo esto es posible debido a que el contador denominado Cnt_R va a estar manteniendo su valor a no ser que pase a valer 0 o se le actualice su valor sumándole 1. Esto sucede en todos los estados, aunque no se encuentre en transmisión.

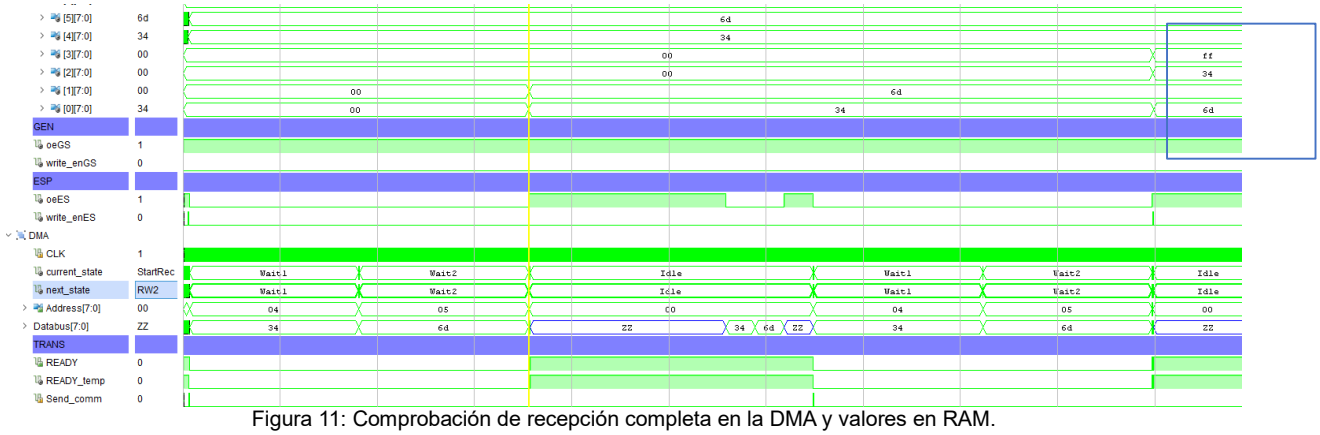

Figura 11: Comprobación de recepción completa en la DMA y valores en RAM.

Con esto se comprobaría el correcto funcionamiento de la terna al completo. Esto también se ha probado en placa y ha funcionado correctamente. Para ello se utilizan los distintos botones, los swicths, los leds y los displays 7 segmentos. En los displays se muestra la temperatura almacenada por defecto en la RAM.

El botón del centro funciona como send_comm, el de la derecha para la lectura de la RAM, el de la izquierda para la escritura de la RAM y el de arriba para el reset. También se conecta un cable entre los pines 1 y 2 del pmodJA para el correcto funcionamiento.

La simulación del módulo de la ALU la hemos realizado mediante un testbench, en el cual probamos todas las instrucciones que nos posibilita la CPU, dejando un intervalo de 2 pulsos de clk por instrucción

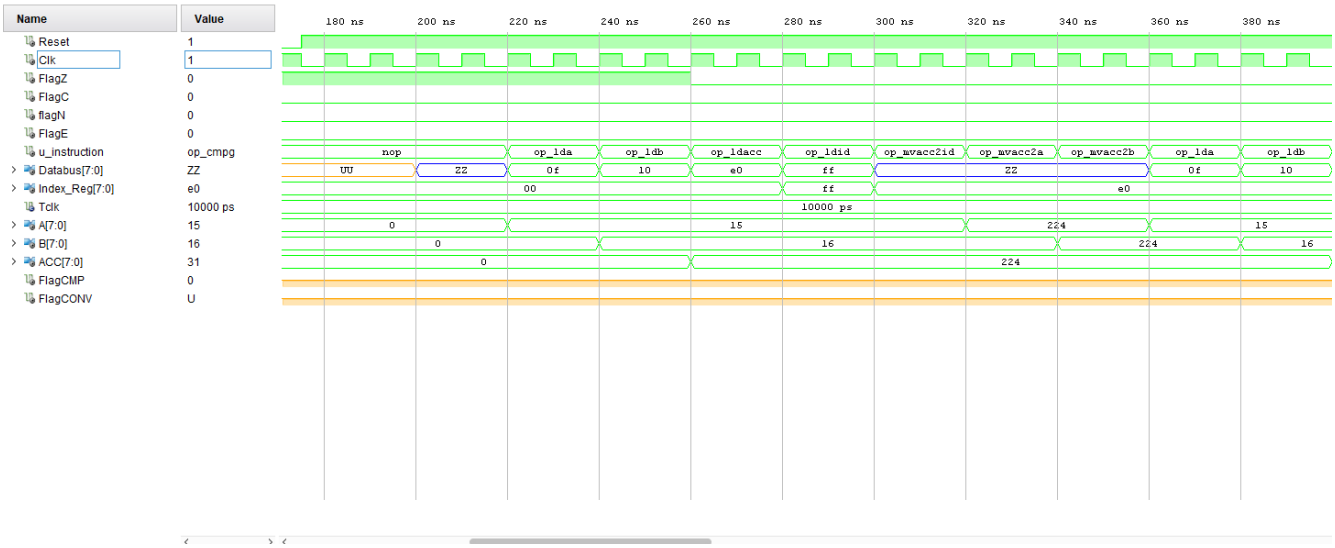

Figura 12: Comprobación de la ALU.

Como se puede ver en la imagen superior en los instantes 220 ns y 240 ns los valores del Databus se cargan correctamente en la variable A y B respectivamente, también se puede observar que desde el instante 300 ns hasta el instante 360 ns se hace la carga interna de valores.

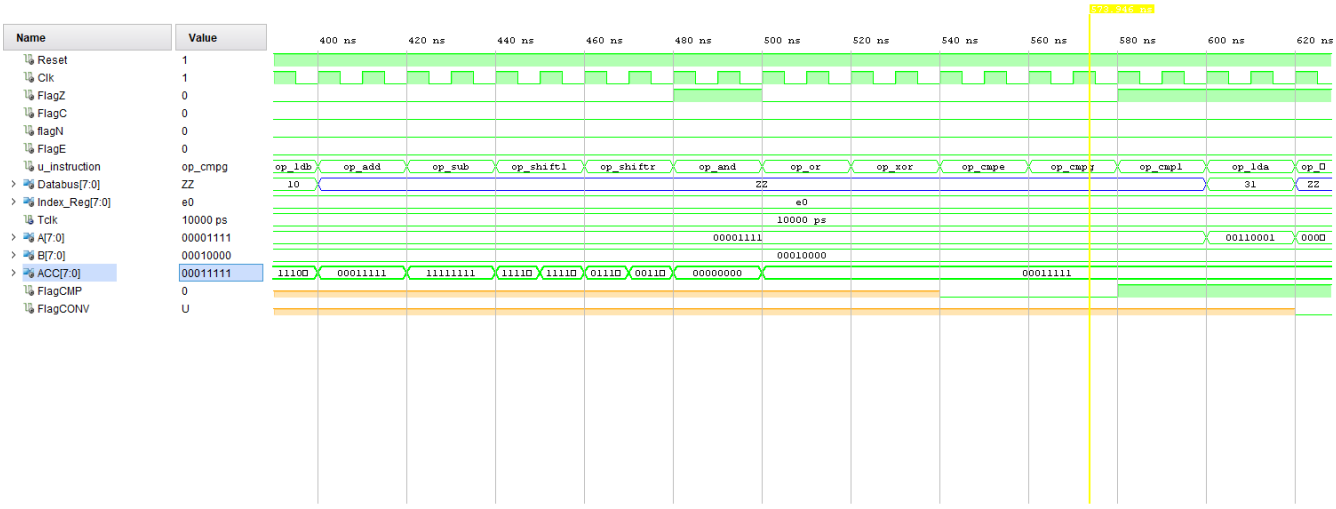

Figura 13: Comprobación de la ALU.

En la imagen superior se puede observar que cuando se realizan las operaciones binarias, el valor de ACC se actualiza correctamente.

También se puede ver en esta imagen que al realizar las operaciones de desplazamiento el resultado es el esperado.

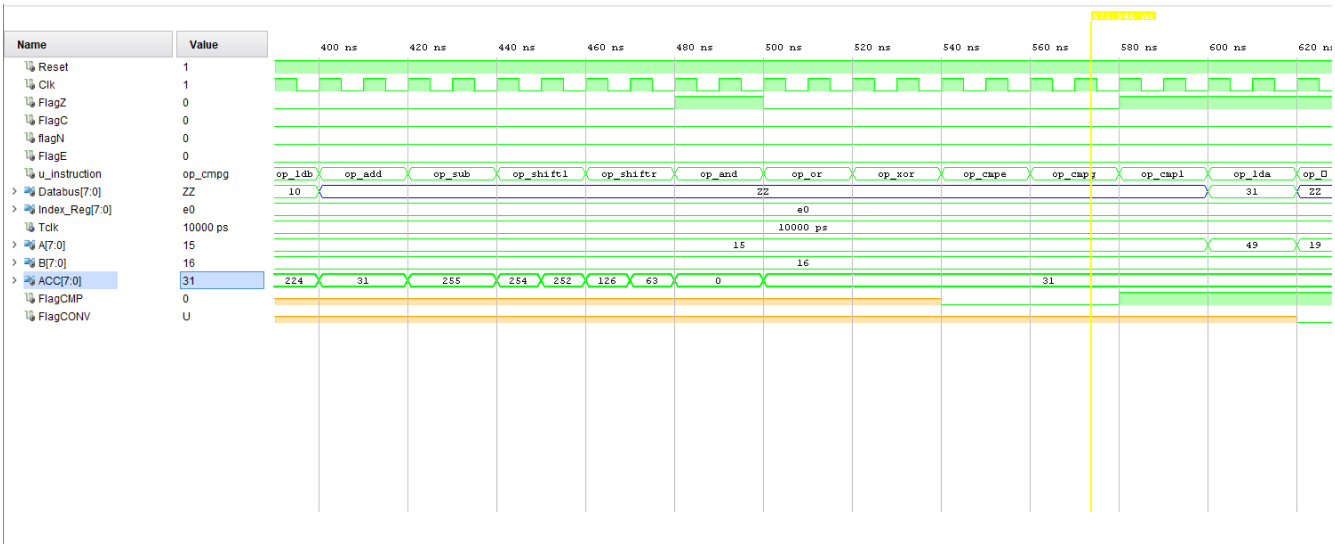

Figura 14: Comprobación de la ALU.

En la imagen superior se puede observar que en los instantes 400ns, 420ns que es cuando realizamos las operaciones aritméticas, los valores en ACC son los correctos.

Por todo esto anterior, la ALU hace su función, pero para comprobar su correcto funcionamiento miraremos la correcta activación de los flags.

En la imagen superior se puede observar que el Flag Z se activa cuando ACC es 0 o cuando alguna de las comparaciones es cierta. Mientras que podemos observar en la imagen inferior que el Flag E se activa al intentar hacer una conversión ASCII a Binario y al no estar en el rango el valor de Databus el flag se activa.

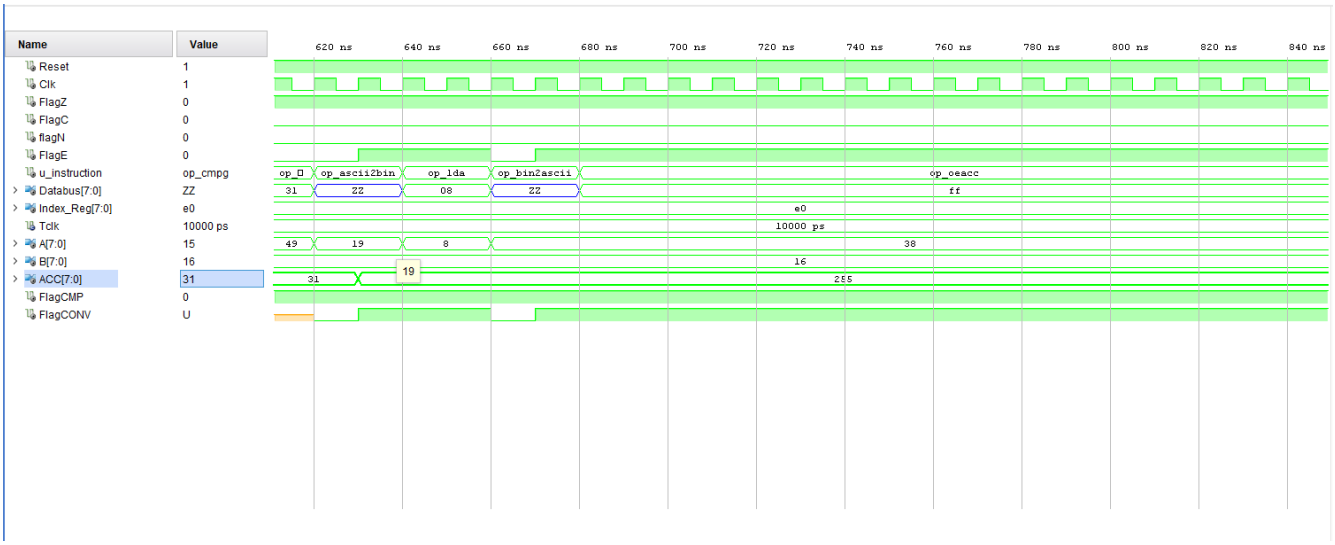

Figura 15: Comprobación de la ALU.

2.3) Resultados de la síntesis

Resource	Utilization	Available	Utilization %
LUT	460	63400	0.73
LUTRAM	24	19000	0.13
FF	666	126800	0.53
BRAM	0.50	135	0.37
IO	55	210	26.19
BUFG	2	32	6.25
MMCM	1	6	16.67

Tabla 1: Resultados de la síntesis para la terna.

En la tabla 1 se puede observar la utilización de los distintos componentes de la FPGA, esto se obtiene tras realizar la síntesis. Sin duda alguna es un porcentaje muy pequeño para la gran utilización que tiene este circuito, por lo que los valores de LUT y FF no parecen valores ilógicos.
BUFG y MMCM se utilizan para el reloj de 20 MHz así que esos valores tienen sentido. BRAM se utiliza para la FIFO, y los IO son los pines que se utilizan en la placa, siendo los switches, leds, displays, botones, etc. por lo que también tienen sentido estos valores.

Resource	Estimation	Available	Utilization %
LUT	97	63400	0.15
FF	43	126800	0.03
IO	27	210	12.86
BUFG	1	32	3.13

Tabla 2: Resultados de la síntesis para la ALU.

En la tabla 2 se muestran los resultados de realizar la síntesis en ALU. Como es lógico usa menos LUT y FF que en el caso anterior, ya que es menos compleja. A su vez, es de apreciar que MMCM, BRAM y LUTRAM desaparecen, esto es debido a que no es necesario aplicar nada de esto en la ALU, ya que no se utiliza ni FIFO ni reloj ni nada por el estilo.

2.4) Mejoras

La mejora que hemos decidido realizar, es el cambio del tamaño de la palabra a enviar mediante el protocolo RS232. Pudiendo utilizar mediante la activación o desactivación de uno de los Switches disponibles en la FPGA, para cambiar de los 8 bits/palabra a 5 bits/palabra.
Esto lo conseguimos mediante un nuevo caso dentro del proceso combinacional, incluyendo un flag interno que nos permite cambiar la longitud del vector a transmitir, tanto en el transmisor como en el receptor del protocolo RS232, este flag se activa cuando el switch está activado, cambiando así su longitud a 5 bits.