UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER EN INGENIERÍA DE SISTEMAS ELECTRÓNICOS

Entrega Final

LCSE

Manuel Viqueira Rodríguez

Examen: 1/02/2021

INDICE

1 Introducción:

En esta memoria se va a exponer la realizado e implementación de un microcontrolador tipo PIC por parte mía, el alumno del Master MUISE, Manuel Viqueira Rodríguez para la asignatura Laboratorio de circuitos y sistemas electrónicos.

Este proyecto se ha implementado correctamente en placa la placa Nexis A7 y se ha constatado su correcto funcionamiento. Cabe destacar que no se ha implementado la parte final de interrupciones, pero si todas las demás partes “RS232, RAM, ROM, CPU, ALU, DMA y su conjunto PIC”, funcionando todas ellas y su conjunto correctamente.

2 Bloque RS232 (entrega 1)

La memoria de esta primera entrega se dividió en 4 partes:

- Cálculo del valor de *PulseEndOfCount* "En mi caso ese valor se lo he puesto a *BitCounter* para el bloque receptor"
- Máquina de estados del transmisor *RS232_TX*.
- Máquina de estados del receptor *RS232_RX*.
- Explicación del funcionamiento del bloque *ShiftRegister*.
- Archivos usados para simular

Los modelados que he realizado siguen unas reglas características de las máquinas de estados empleadas en Matlab Stateflow junto con unas cuantas reglas extra para facilitar la lectura. Estas son:

- Las condiciones de cambio de estado se escriben en cursiva, entre corchetes **[]** y se ponen solo en líneas **horizontales**. La prioridad de las salidas "camino a tomar en caso de que 2 sean ciertas" viene dada por un número al lado de estas.
- Las acciones a realizar en caso de que se cumplan las condiciones se colocan en negrilla, entre llaves **{}** y siempre se ponen solo en líneas **verticales**.
- En cada ciclo de reloj, **solo un camino** puede ser tomados.

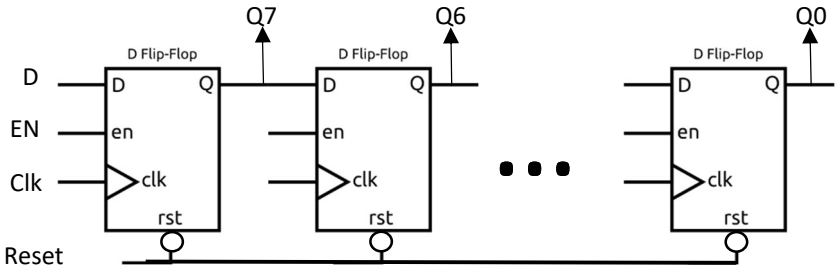
2.1 Cálculo de PulseEndOfCount

Uno de los requisitos a conseguir en este modelo de bus de datos RS232 es que debe funcionar a una tasa binaria 115200 bps, trabajando a su vez con una frecuencia de reloj de 20Mhz "la FPGA cuenta con una entrada a 100Mhz pero esta es bajada internamente a 20".

Para conseguir esta velocidad de transmisión, debemos calcular cuantos ciclos de reloj será necesario esperar para conseguirla (*PulseEndOfCount*), la cuenta sería:

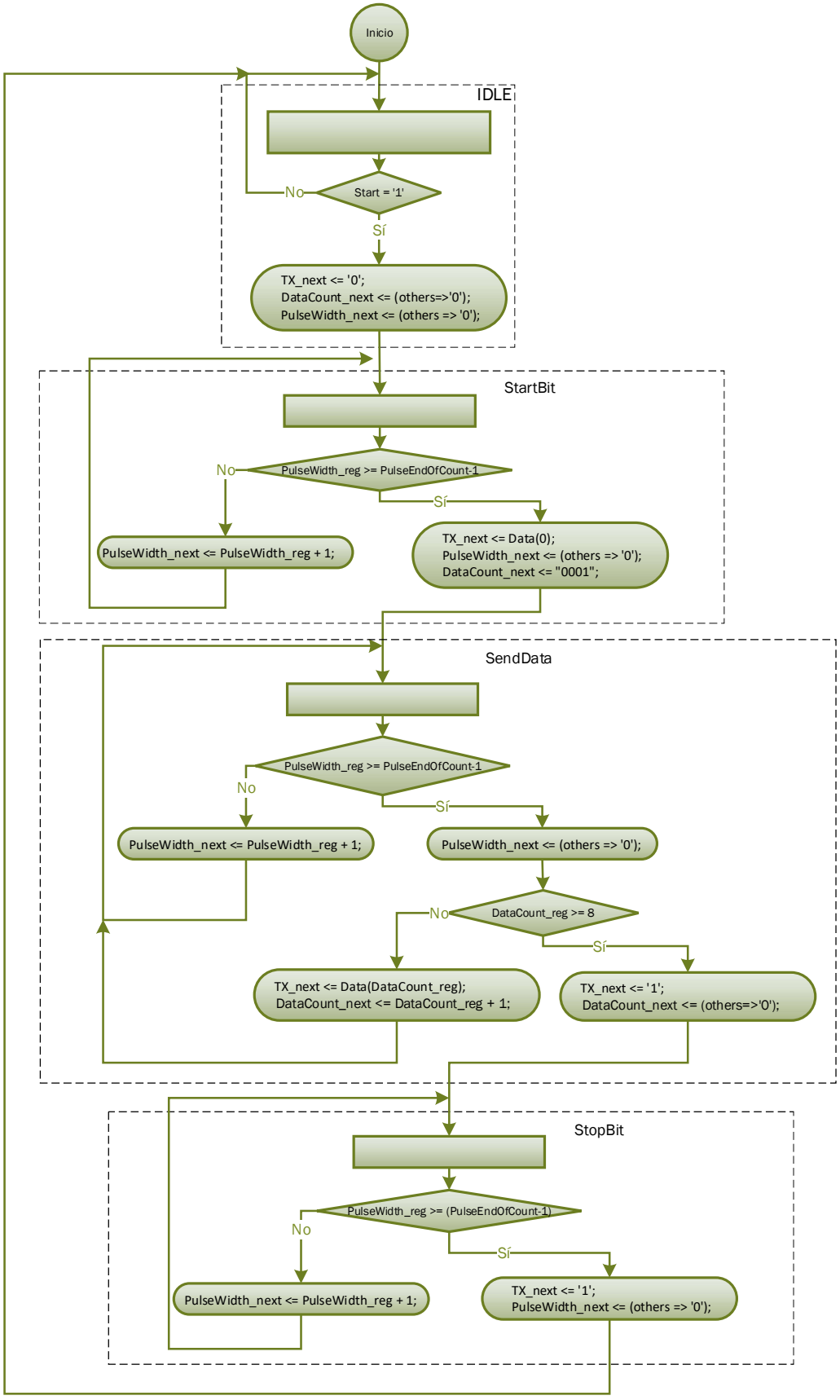
$$PulseEndOfCount = \frac{f_{clock}}{v_{trans}} = \frac{20 \cdot 10^6}{115200} = 173.6\hat{1} \approx 174\ ciclos$$

2.2 ShiftRegister



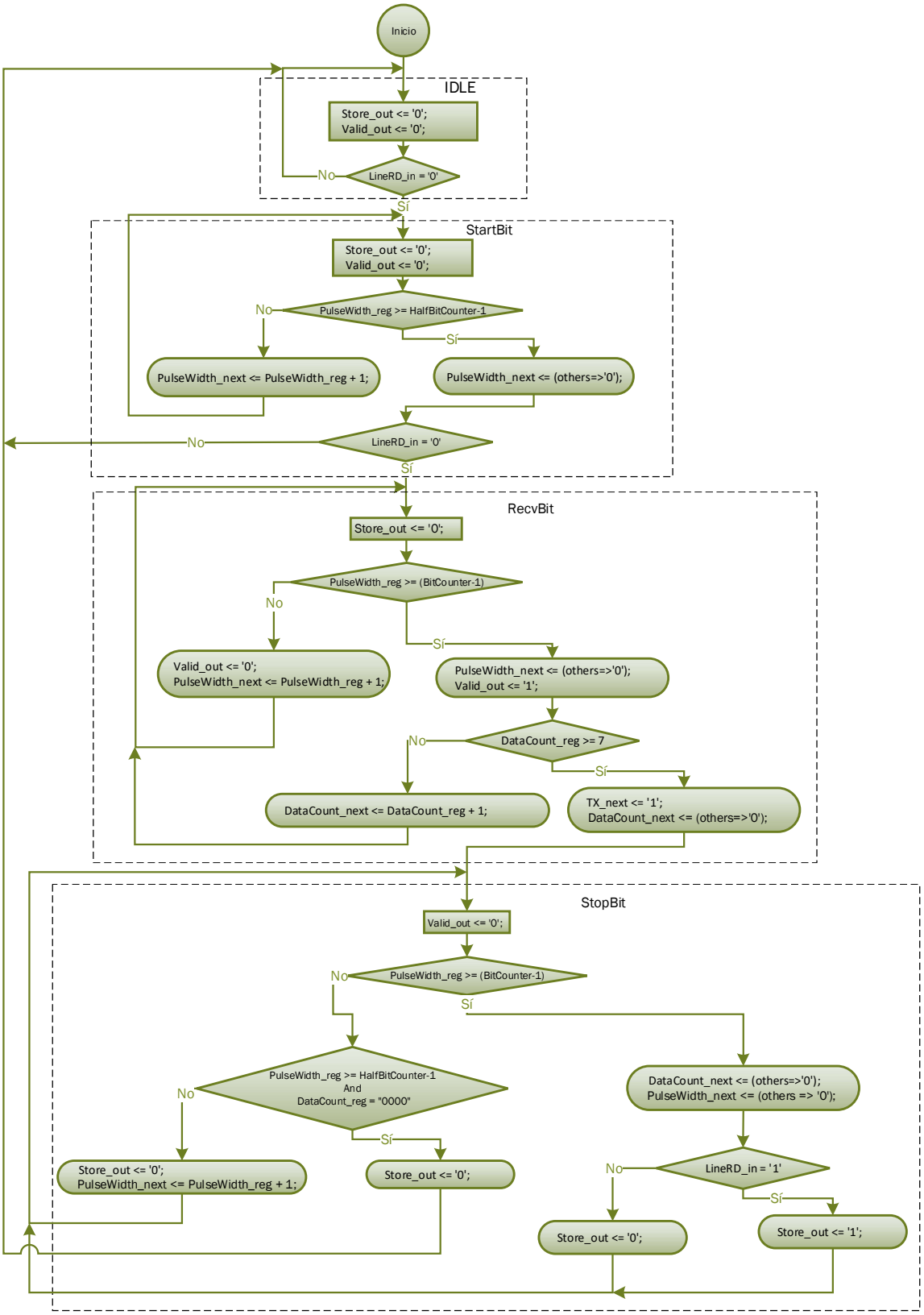
El Shift Register consiste en un bloque con 8 flip-flops seguidores (tipo D) que permite ir desplazando el bit de entrada desde el bit más significativo al menos significativo.

2.3 Modelado de la Máquina de estados del transmisor RS232_TX



2.4 Modelado de la Máquina de estados del receptor RS232_RX

En el caso del RS23_RX, el contador se realiza mediante la variable *PulseWidth* de un modo similar al transmisor. Sin embargo, en vez de tener como máximo *PulseEndOfCount* tendremos 2 máximos dependiendo de si contamos todos los pulsos necesarios o tan solo la mitad. Estos serán *BitCounter* y *HalfBitCounter* respectivamente.



3 RAM

La RAM se emplea para el almacenamiento de los datos recibidos y a enviar por el módulo RS232, así como para el almacenamiento de los datos de switches, actuadores y temperatura.

Esta RAM cuenta con una parte de la memoria que será reseteada al recibir la señal de *reset* "desde la posición 0 hasta la posición 63, llamada memoria especial" y con una parte de la memoria que no será reseteada "de las posiciones 64 a 255 llamada memoria general. De estas posiciones pertenecientes a la memoria especial, todas (excepto la posición de temperatura) serán reseteadas a 0. En cuanto a la temperatura, esta será reseteada a 25 correspondiendo el 2 a los primeros 4 bits y el 5 a los últimos 4 bits.

Estos datos están almacenados en una sola señal interna denominada *Memoria.* Se ha valorado el uso de dos señales *Memoria_esp* y *Memoria_gen* de 64 y 192 posiciones respectivamente, pero se ha desechado al ocasionar un timing-loop que fue solucionado mediante la fusión de estas en una sola señal.

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

La lectura de la RAM se realizará cuando la señal de entrada *OE* se encuentre a '0' y se realizará de forma asíncrona. Por otra parte, la escritura en RAM será síncrona y se realizará al estar la señal *OE* y la señal *Write_en* ambas a '1', lo cual evitará la lectura y escritura a la vez dando prioridad a la lectura.

Por último, la RAM realizará la conversión de binario a 7segmentos de los valores de temperatura para su posterior salida a un display de 7 segmentos.

4 DMA

La DMA es un bloque que coordina y controla la comunicación entre el módulo RS232 y la RAM, requiriendo por ello el uso de buses comunes gestionados por la CPU. Este bloque será el encargado de realizar las recepciones de los datos y su posterior almacenamiento en RAM, avisando cuando se realice la recepción de 3 bytes a la CPU por medio de la carga del valor x"FF" en la posición 3 de la RAM.

En cuando al envío de datos, este se realizará cuando la CPU lo ordene por medio de la señal "*send_comm*", momento en el cual se cargarán los datos de las posiciones 4 y 5 de la RAM al bloque RS232.

4.1 Consideraciones de diseño

El uso compartido de los buses *Databus*, *Adress*, *OE* y *write_en* por los bloques DMA, CPU y RAM ocasiona diversos problemas que han sido atajados de las siguientes formas:

Los buses *Databus* y *Adress* son empleados por la DMA el menor tiempo posible pasando a estado 'Z' de forma predeterminada permitiendo su uso a la CPU el mayor tiempo posible. Por ejemplo, a la hora de enviar los datos, los buses *Adress* y *Databus* no serán ocupados por la DMA excepto en la carga inicial de los datos en el módulo RS232.

A la hora de sintetizar, se corre el riesgo de que el valor en la simulación Behavioural 'Z' se convierta en '1' o '0' en la síntesis al no ser 'Z' un valor válido, en síntesis. Este comportamiento fue observado en los buses OE y write_en de la RAM, lo cual ocasionaba una constante lectura de posiciones y que tanto la simulación Post-Synthesis como la puesta en placa no se comportaran correctamente.

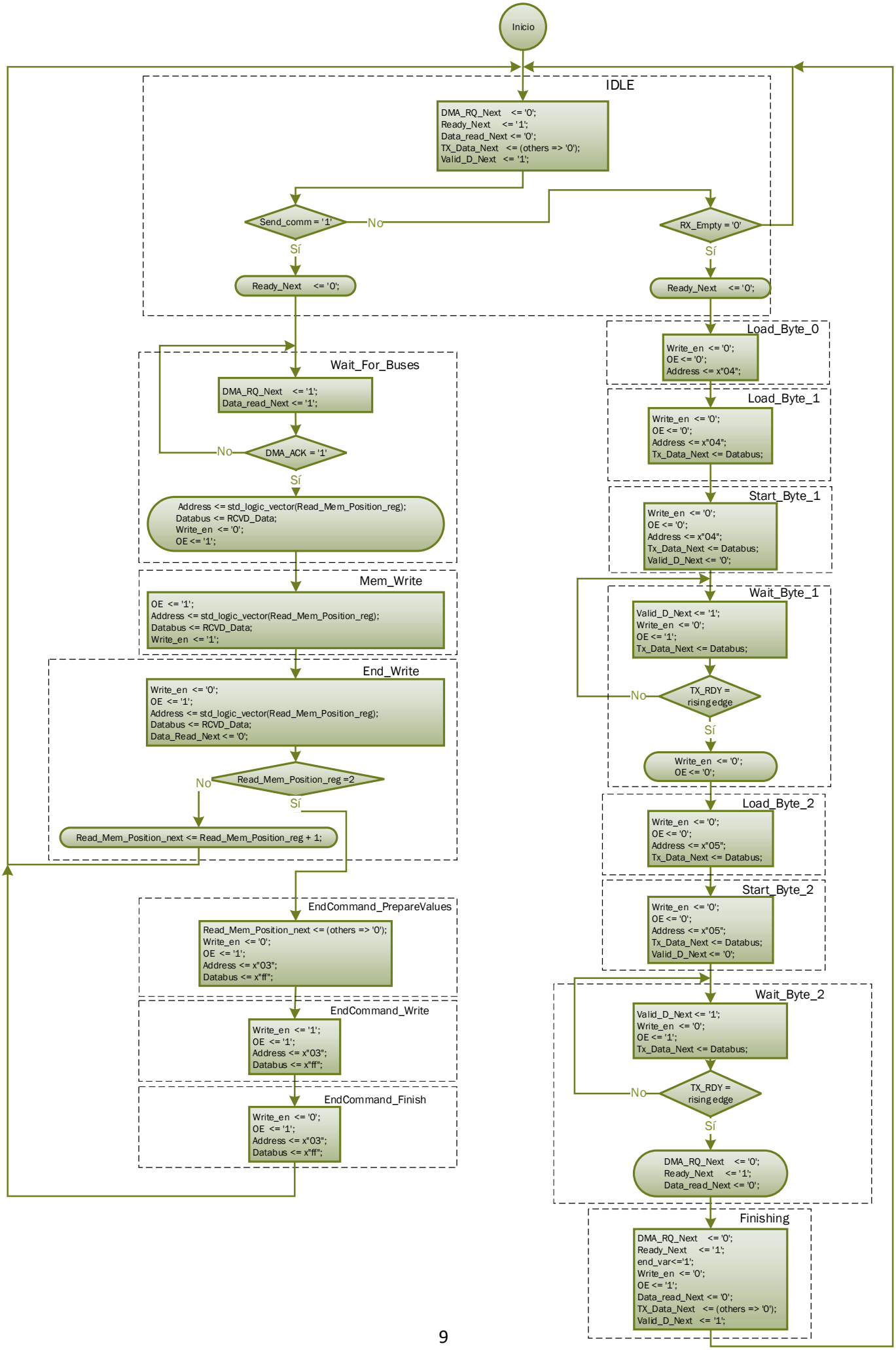
Para solucionar este problema, se ha optado por los valores por defecto *OE* <= '1' y *write_en* <= '0' en lugar de 'Z', evitando así posibles errores de sintetizado.Como es obvio, los valores '0' y '1' por defecto **no permiten el uso compartido de los buses**, por lo cual será necesario el uso de puertas lógicas que combinen las salidas de DMA y CPU para generar una señal apropiada a la RAM. Esto es explicado en el apartado de diseño del conjunto completo

4.2 Máquina de estados

La máquina de estados empleada para el control de la DMA es una máquina que combina Moore y Mealy para reducir el número de los mismos.

Esta máquina sería de la siguiente forma:

"Anotación: la parte izquierda del diagrama corresponde a los estados de **lectura** mientras que los estados de la derecha corresponden a los estados de **escritura**.


Inicio
IDLE
DMA_RQ_Next <= '0';
Ready_Next <= '1';
Data_read_Next <= '0';
TX_Data_Next <= (others => '0');
Valid_D_Next <= '1';
Send_comm = '1'
No
Sí
Ready_Next <= '0';
RX_Empty = '0'
Sí
Ready_Next <= '0';
Wait_For_Buses
DMA_RQ_Next <= '1';
Data_read_Next <= '1';
No
DMA_ACK = '1'
Sí
Address <= std_logic_vector(Read_Mem_Position_reg);
Databus <= RCVD_Data;
Write_en <= '0';
OE <= '1';
Mem_Write
OE <= '1';
Address <= std_logic_vector(Read_Mem_Position_reg);
Databus <= RCVD_Data;
Write_en <= '1';
End_Write
Write_en <= '0';
OE <= '1';
Address <= std_logic_vector(Read_Mem_Position_reg);
Databus <= RCVD_Data;
Data_Read_Next <= '0';
No
Read_Mem_Position_reg =2
Sí
Read_Mem_Position_next <= Read_Mem_Position_reg + 1;
EndCommand_PrepareValues
Read_Mem_Position_next <= (others => '0');
Write_en <= '0';
OE <= '1';
Address <= x"03";
Databus <= x"ff";
EndCommand_Write
Write_en <= '1';
OE <= '1';
Address <= x"03";
Databus <= x"ff";
EndCommand_Finish
Write_en <= '0';
OE <= '1';
Address <= x"03";
Databus <= x"ff";
Load_Byte_0
Write_en <= '0';
OE <= '0';
Address <= x"04";
Load_Byte_1
Write_en <= '0';
OE <= '0';
Address <= x"04";
Tx_Data_Next <= Databus;
Start_Byte_1
Write_en <= '0';
OE <= '0';
Address <= x"04";
Tx_Data_Next <= Databus;
Valid_D_Next <= '0';
Wait_Byte_1
Valid_D_Next <= '1';
Write_en <= '0';
OE <= '1';
Tx_Data_Next <= Databus;
No
TX_RDY = rising edge
Sí
Write_en <= '0';
OE <= '0';
Load_Byte_2
Write_en <= '0';
OE <= '0';
Address <= x"05";
Tx_Data_Next <= Databus;
Start_Byte_2
Write_en <= '0';
OE <= '0';
Address <= x"05";
Tx_Data_Next <= Databus;
Valid_D_Next <= '0';
Wait_Byte_2
Valid_D_Next <= '1';
Write_en <= '0';
OE <= '1';
Tx_Data_Next <= Databus;
No
TX_RDY = rising edge
Sí
DMA_RQ_Next <= '0';
Ready_Next <= '1';
Data_read_Next <= '0';
Finishing
DMA_RQ_Next <= '0';
Ready_Next <= '1';
end_var<='1';
Write_en <= '0';
OE <= '1';
Data_read_Next <= '0';
TX_Data_Next <= (others => '0');
Valid_D_Next <= '1';

5 ALU

La Unidad Aritmético-Lógico (ALU) del PIC es el componente encargado de la realización de los cálculos matemáticos y lógicos requeridos por el set de instrucciones. La ALU cuenta con dos registros A y B para realizar las operaciones más un registro para guardar el resultado ACC y un registro de salida Index_reg. Además, cuenta con otras 4 salidas que corresponderían a los flags N, C, Z y E.

Para facilitar los cálculos "especialmente los cálculos de los flags" se ha optado por la incorporación de dos variables auxiliares *Aux* y *Aux_Nib* de 8 y 4 bits respectivamente

Los cálculos que realizará serán:

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

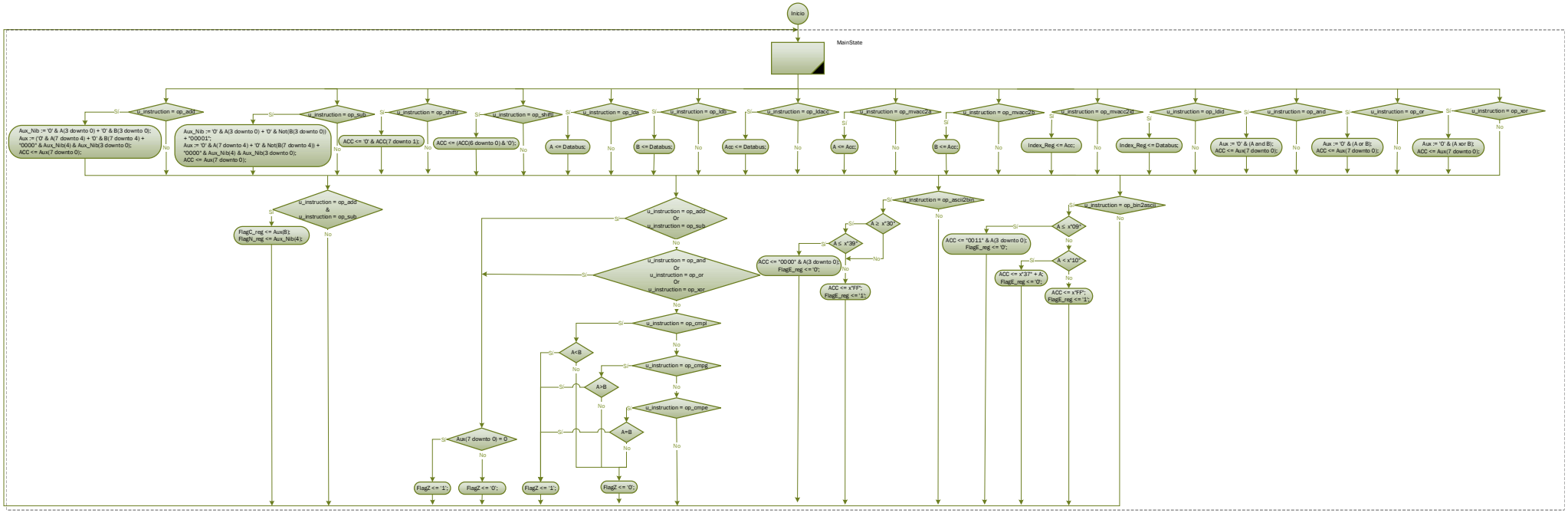
Todas las operaciones excepto la operación de carga del acumulador en el Bus de datos *Databus* se realizan de forma síncrona "cuando llega el flanco positivo del *clk*". La carga del acumulador en el Bus se realiza de forma asíncrona para garantizar que el *Adress* "aportado por la CPU" y el Bus de datos *Databus* lleguen al mismo tiempo a la RAM.

5.1 Máquina de estados

La máquina de estados de la ALU es muy sencilla ya que poidría ser considerada como una máquina de estados Mealy con tan solo un estado ejecutado en bucle y que ejecuta una de las operaciones descritas en el apartado anterior + los cálculos de los flags que esta modificaría cada vez que llega el flanco de reloj.

Fuera de esta máquina queda la carga del *Acc* en el *Databús* al realizarse de forma asíncrona en un proceso separado aún que dentro del mismo componente.

Esta máquina de estados sería:


Inicio
MainState
u_instruction = op_add
u_instruction = op_sub
u_instruction = op_shiftr
u_instruction = op_shiftl
u_instruction = op_ldg
u_instruction = op_ldb
u_instruction = op_ldacc
u_instruction = op_mvaccr2a
u_instruction = op_mvaccr2b
u_instruction = op_mvaccr2d
u_instruction = op_ldid
u_instruction = op_and
u_instruction = op_or
u_instruction = op_xor
Aux_Nib := '0' & A(3 downto 0) + '0' & B(3 downto 0);
Aux := ('0' & A(7 downto 4) + '0' & B(7 downto 4) + "0000" & Aux_Nib(4) & Aux_Nib(3 downto 0);
ACC <= Aux(7 downto 0);
Aux_Nib := '0' & A(3 downto 0) + '0' & Not(B(3 downto 0)) + "00001";
Aux := '0' & A(7 downto 4) + '0' & Not(B(7 downto 4)) + "0000" & Aux_Nib(4) & Aux_Nib(3 downto 0);
ACC <= Aux(7 downto 0);
ACC <= '0' & ACC(7 downto 1);
ACC <= (ACC(6 downto 0) & '0');
A <= Databus;
B <= Databus;
Acc <= Databus;
A <= Acc;
B <= Acc;
Index_Reg <= Acc;
Index_Reg <= Databus;
Aux := '0' & (A and B);
ACC <= Aux(7 downto 0);
Aux := '0' & (A or B);
ACC <= Aux(7 downto 0);
Aux := '0' & (A xor B);
ACC <= Aux(7 downto 0);
u_instruction = op_add
&
u_instruction = op_sub
FlagC_reg <= Aux(8);
FlagN_reg <= Aux_Nib(4);
u_instruction = op_add
Or
u_instruction = op_sub
u_instruction = op_and
Or
u_instruction = op_or
Or
u_instruction = op_xor
u_instruction = op_cmpl
A<B
u_instruction = op_cmpg
A>B
u_instruction = op_cmpe
A=B
Aux(7 downto 0) = 0
FlagZ <= '1';
FlagZ <= '0';
FlagZ <= '1';
FlagZ <= '0';
u_instruction = op_asciiZbin
A ≥ x"30"
A ≤ x"39"
ACC <= "0000" & A(3 downto 0);
FlagE_reg <= '0';
ACC <= x"FF";
FlagE_reg <= '1';
u_instruction = op_bin2ascii
A ≤ x"09"
ACC <= "0011" & A(3 downto 0);
FlagE_reg <= '0';
A < x"10"
ACC <= x"37" + A;
FlagE_reg <= '0';
ACC <= x"FF";
FlagE_reg <= '1';

6 CPU

La CPU (MainControl) es el componente principal que dirige todo el resto de componentes. Recibe las líneas de código de la ROM, las interpreta y las ejecuta mandando a DMA, ALU y RAM todo lo necesario para que puedan actuar.

Este componente a sido desarrollado de forma externa a este proyecto, por lo que su descripción no es necesaria. La única modificación realizada a la CPU ha sido el cambio de valores predeterminados de las salidas *OE* y *write_en* de 'Z' a '0' y '1' respectivamente. Al igual que en el caso de la DMA, esto se ha realizado para evitar la aparición de errores de sintetizado que ocasionaban el valor por defecto '0' de la línea *OE* el cual generaba una constante lectura y un error grave en la simulación post-synthesis y en la programación de la FPGA.

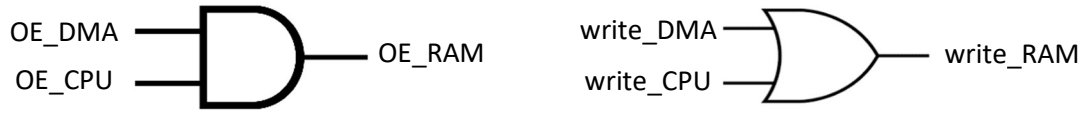
7 PIC completo

En este apartado se explicará la combinación de todos los bloques en uno para su posterior programación en placa así como el constraint “archivo .xdc”, que ha sido ligeramente modificado para alcanzar un correcto funcionamiento en placa.

7.1 Pic_top.vhdl

Aún que este archivo no ha sido hecho íntegramente como parte de este proyecto, si ha requerido dos adiciones importantes para conseguir un correcto funcionamiento en placa. Estas adiciones han sido:

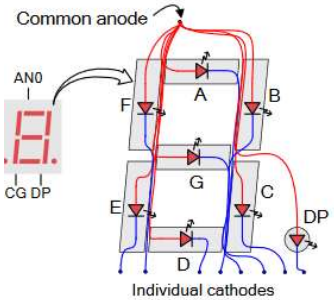
Primero, se ha añadido dos puertas lógicas de control de las señales *OE* y *write_en* procedentes de la DMA y la CPU cuyas salidas se han conectado a la RAM para habilitar la lectura y la escritura. Ambas señales se han modificado para evitar los errores post-synthesis ocasionados por la suposición del valor ‘Z’ como valor ‘0’ cambiando sus valores por defecto a ‘0’ y ‘1’ requiriendo, por ende, dos puertas lógicas con dos entradas procedentes de CPU y DMA y una salida para la RAM. Estas puertas lógicas son:



Además de esas puertas, fue preciso incorporar un sistema síncrono de control de los displays de 7 segmentos ya que los valores procedentes de la FPGA deben de ser direccionados apropiadamente a los diferentes displays por medio de un Bus común, obligando a encender uno de los displays durante 400ms “lo que vienen siendo unos 20.000 ciclos de reloj a 20MHz”, alternando así la visualización de los displays *Temp_H* y *Temp_L* de forma que a ojos de humano parezca que están encendidos todo el tiempo.

Tres apuntes importantes de esta parte es que los displays:

- El sistema cuenta con un reset que resetea el contador y evita que se muestre nada en el display mientras este esté activo.
- Los displays de la placa a utilizar son de ánodo común, por lo que los leds que se iluminarán van a ser los que tengan el valor ‘0’ y no el valor ‘1’ “ver imagen”.
- Los displays de 7 segmentos cuentan en realidad con 8 segmentos ya que incluyen el punto de separación de los números “DP en la imagen inferior”. Para evitar posibles problemas, este dígito extra fue incluido a los valores de *Temp_H* y *Temp_L* como un ‘1’ en la posición 7 para que siempre esté apagado ya que no es requerido en nuestro caso.



7.2 Pictop.xdc

Este es el archivo encargado de acomodar todo el código VHDL al chip y a la placa concreta que vayamos a emplear. En nuestro caso, se trata de una placa Nexis A7 de Diglent, la cual cuenta con un chip XC7A100T-1CSG324C de Xilinx.

Este archivo tampoco a sido realizado al 100% para este proyecto, pero algunas modificaciones fueron necesarias para alcanzar un comportamiento correcto.

Primero, todos los puertos a usar se han unido a las salidas y entradas del bloque PIC_top, utilizando los nombres exactos para evitar fallos.

Algunos puertos extra se han abierto para evitar errores, como es el caso del bit 7 del display de 7 segmentos “el punto de separación de números mencionado en el apartado anterior”, o todos los bits de control de los displays, *disp* “aunque la mayoría de ellos se mantienen siempre a ‘1’ para evitar que muestren nada” siendo solo necesarios los 2 de ellos.

Por último, las líneas de RS232 TX y RX han sido correctamente conectadas a las líneas de RS232-USB.

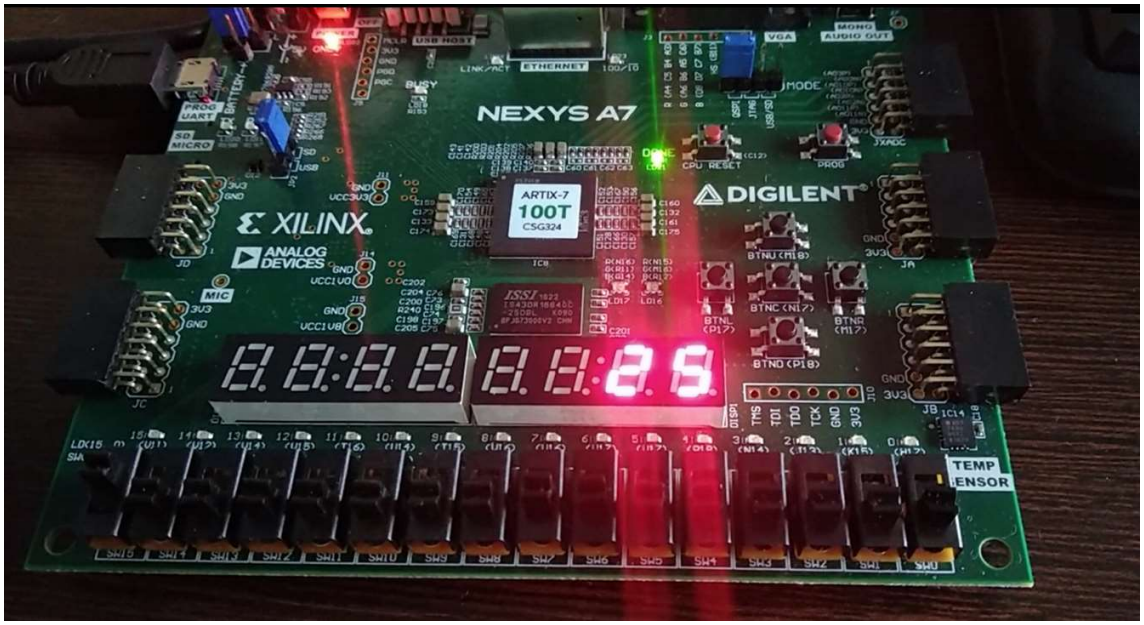
8 Simulaciones

Durante la realización de este proyecto se han realizado, además, multitud de pruebas y simulaciones de los diferentes componentes. Estos testbenches son explicados en la siguiente tabla:

Nombre	Bloques	Comportamiento simulado
RS232_RX_tb.vhd	RS232_RX	Recepción de los Bytes: “01000101” y “10111010”. La simulación debe durar 6.000ns y el valor de *PulseWidth* debe ser 4 para una mayor visibilidad del resultado.
RS232_TX_tb.vhd	RS232_TX	Envio de los Bytes: "01010100" y "10101011". La simulación debe durar 6.000ns y el valor de *BitCounter* debe ser 4, siendo *HalfBitCounter* 2 para una mayor visibilidad del resultado
ShiftRegister_TB.vhd	ShiftRegister	Comprueba desplazamiento
tb_RS232top.vhd	RS232top “completo”	Prueba de recepción del dato “01111001” y envío del dato "11100010" a valores de RS232 normales (*BitCounter* = 174 y *HalfBitCounter* = 82.
ram_tb.vhd	RAM	Pruebas múltiples de escritura y lectura de las posiciones previamente guardadas
RAM_DMA_RS232_tb.vhd	RAM, DMA y RS232 “completo”	Simulación de la recepción por la línea RS232 de los Bytes “10101010”, “11111111” y “11110111” para una respuesta posteriormente que debe ser forzada en la RAM durante la simulación.
tb_ALU.vhd	ALU	Extensa prueba de TODAS las instrucciones posibles a realizar por la ALU. Fuertemente comentada en el archivo .vhdl
PICtop_tb.vhd	Todos los bloques	Simulación del PIC en su conjunto. Comandos simulados por la línea RS232: “I41” -> Encender el switch 4. “A96” -> Poner el actuador 9 a 6. “T12” -> Cambiar la temperatura a 12. “SA9” -> Solicitar información del actuador 9.

9 Resultado y conclusión

Como resultado final, se ha conseguido que funcionen correctamente todos los elementos en placa, prueba de ello es el video indexado con esta memoria en la que se muestra el funcionamiento de este proyecto funcionando en la placa Nexis A7 conectado a un PC mediante la herramienta Putty.



Como conclusión final, este proyecto ha sido muy interesante y he podido aprender muchísimo aún que ha sido extremadamente frustrante en ciertas ocasiones ya que dos errores principalmente, “un timing-loop por culpa de la creación de dos señales de memoria de RAM y no una como finalmente se hizo + errores en la simulación de la síntesis causados por los valores por defecto de ‘0’ en las señales *OE* y *write_en*” me han hecho invertir casi tantas horas como la realización del proyecto completo en sí. No obstante, ha sido muy satisfactorio ver el resultado final funcionando, especialmente teniendo en cuenta mi incorporación tardía con un mes y medio de retraso al master lo cual me ha impedido realizar todas las ampliaciones y modificaciones que me hubiera gustado.