UNIVERSIDAD DE ALCALÁ

Grado en Ingeniería Electrónica y Automática Industrial

DISEÑO ELECTRÓNICO

CONTROL DE UN MOTOR DC DESDE UNA FPGA

Vladislav Kravchenko

Óscar Betancort Hernández

Contenido

1. INTRODUCCIÓN

Con la realización de esta práctica se pretende realizar, mediante el modelado en el lenguaje VHDL, el control de un motor DC desde una FPGA. Para ello se dispone de un motor DC de 6V, de un puente en H (PMOD HB5) que será el encargado de establecer la interconexión entre la placa FPGA, NEXYS DDR, y el motor DC.

Partimos desde los módulos principales (proporcionados por el profesorado) de los que, finalmente, constará todo el proyecto. Estos módulos se han proporcionado para que se sigan los pasos de definición de las señales que vayan a intervenir en el control del motor DC. Durante nuestro desarrollo de la práctica no se han añadido módulos adicionales al conjunto existente.

2. DISEÑO

2.1. MÓDULO GENERACIÓN SEÑAL PWM – M1

Este módulo es el por el cual se ha comenzado el modelado en VHDL de la practica a desarrollar. Esta parte se encargará de la generación de la señal PWM con una frecuencia de 2kHz a partir de la señal diente de sierra. Se ha elegido la señal diente de sierra por ser una señal más fácil de modelar en el lenguaje VHDL, en comparación con la señal triangular que tendría una pendiente de subida y otra de bajada, cada una con sendos contadores.

Para generar la señal PWM, primero se ha realizado la señal de reloj de 2kHz, que será la encargada, con cada flanco de subida, de incrementar la pendiente de la señal diente de sierra hasta un valor máximo de 99, determinado así para un ciclo de trabajo del 100%. Este incremento lo guardaremos en la señal ***cuenta.*** En el momento en que llegue a 99, la señal se reseteará y empezará desde cero la cuenta ascendente. La señal ***cuenta***, se va a ir comparando con la señal de entrada ***PWM_vector***, encargada de proporcionar a este módulo el ciclo de trabajo. En el momento en el cual esta señal, ***PWM_vector***, tenga un valor mayor que el de ***cuenta***, se generará la PWM.

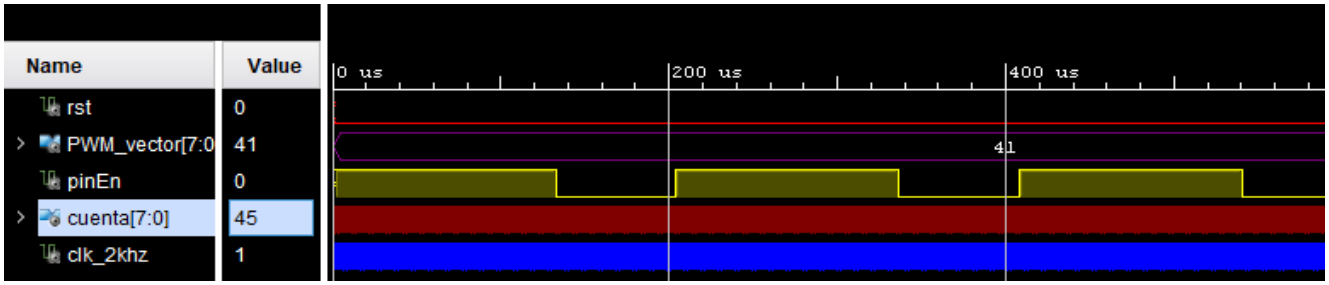

Imagen 1. Ilustración de la generación de la señal PWM con ciclo de trabajo del 65%

En la imagen anterior (Imagen 1), se ilustra la generación de la señal de reloj de 2kHz (***clk_2kHz***), se han establecido unos parámetros, tales que, se pueda generar esta señal a mayor frecuencia; de la señal de cuenta ascendente(***cuenta***) y la propia señal PWM (***pinEn***). En este caso, el ciclo de trabajo es del 65%. Para que se vea mejor tanto la señal de reloj como la señal de cuenta, se hace un zoom en estas señales (imagen 2).

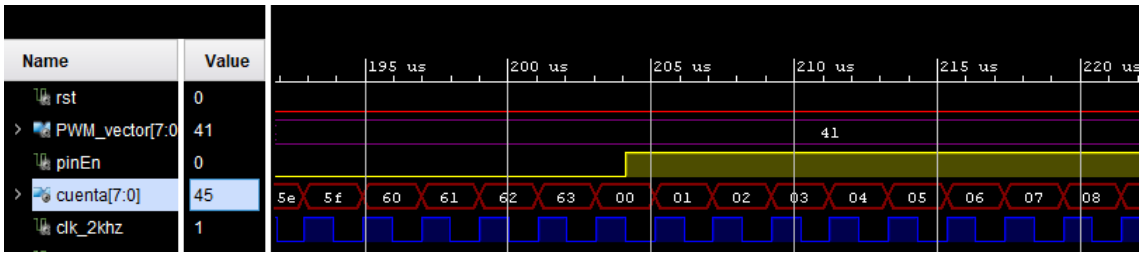

Imagen 3. Ilustración de la señal de reloj de 2kHz y de la señal de cuenta ascendente.

Además, en este módulo, se debe considerar los tiempos muertos en el momento en que se detecte un cambio de giro del motor. El tiempo muerto, es un tiempo en el cual se retrasa el encendido de uno de los "interruptores" de una misma rama, hasta que el otro no se haya apagado; para así no producir un cortocircuito.

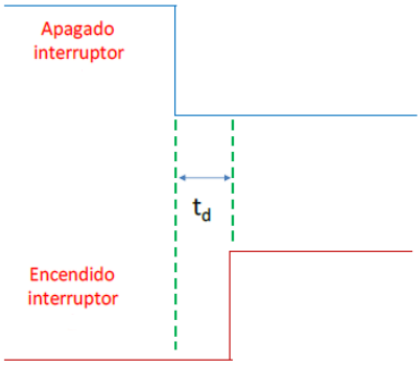

Imagen 2. Ilustración de un tiempo muerto

En el momento que el switch encargado de realizar el cambio de dirección del motor, cambie de posición, de 0 a 1 o viceversa, el motor debe cambiar la dirección de giro a los 10 milisegundos; y la generación de la PWM,

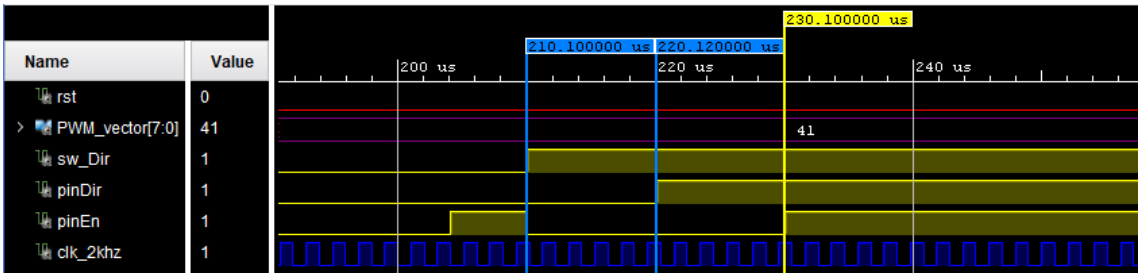

Imagen 4 Ilustración de las señales de clk_2kHz, sw_Dir y pinEn

debe estar en su valor Low, durante 20 milisegundos.

En la imagen 4, se ilustra el tiempo muerto en la simulación del módulo M1. Para una simulación más rápida se han considerado unas constantes de tiempo reducidas en comparación con las reales. Se puede observar que en cuanto se produce un cambio en la señal ***sw_Dir***, el cambio de la dirección de giro en la señal de salida ***pinDir*** (señal de salida encargada de trasmitir el sentido de giro al puente en H) se produce a los 10μs en la simulación (en la realidad deben ser 10ms). Por otro lado, también hay que destacar que en cuanto se detecta un cambio en ***sw_Dir***, la señal PWM cambia su estado y empieza a tener un valor '0' durante 20μs (20ms en la realidad).

2.2. MÓDULO DE SELECCIÓN DE PWM – M2

Este segundo bloque se encarga de leer dos pulsadores que hemos llamado como ***btn_up*** y ***btn_down***, los cuales tendrán la funcionalidad de incrementar o decrementar el ciclo de trabajo de la PWM que se genera en el bloque M1. Se parte de que el ciclo de trabajo inicial es del 0%, aunque en la simulación real al tener un motor de poca potencia es necesario que tenga un ciclo de trabajo mínimo del 60 % para que comience a girar.

La lectura de los pulsadores la realizaremos cada 0.1 segundos, es decir a una frecuencia de 10 Hz, y tendremos que considerar dos formas de actuar sobre ellos. La primera será que cada vez que se pulse se incremente o decremente una unidad del ciclo de trabajo, o que si mantenemos apretado el botón tendremos que ir incrementando o decrementando el valor del ciclo cada 0.1 segundos, de forma que si mantuvieses el pulsador ***btn_up*** durante un 1 segundo se incrementaría en 10 unidades el ciclo de trabajo.

Para implementar la funcionalidad de este módulo hemos creado un proceso que a partir de la señal reloj del FPGA, la señal CLK, obtenemos una señal que tendrá una frecuencia de 10 Hz siendo esta la que usaremos para comprobar el estado de los pulsadores ***btn_up*** y ***btn_down***, esta señal será denominada con el nombre ***clk_10hz.***

Una vez que tenemos el reloj de 0.1 segundos preparado continuamos creando un proceso que sea sensible a los flancos de subida de la señal ***clk_10hz***, cuando haya detectado uno hará dos comparaciones, la primera será comprobar si el pulsador ***btn_up*** está presionado (***btn_up*** = 1) y si el ciclo de trabajo es menor que 100 (***vec_temp*** < 100), esta última comprobación es para evitar que el ciclo se siga incrementando una vez haya alcanzado el 100%, si se cumple las comparaciones sumaremos 1 a la señal que guarda el valor del ciclo de trabajo (***vec_temp***).

La otra comparación que está dentro del proceso es el de comprobar si el pulsador ***btn_down*** está a nivel alto (***btn_down*** = 1) y si el ciclo de trabajo es mayor que 0 (***vec_temp*** > 0), está comprobación es para evitar que introduzca números negativos y

haya problemas con la lectura del ciclo, si se cumplen las comparaciones restaremos 1 a la señal ***vec_temp***.

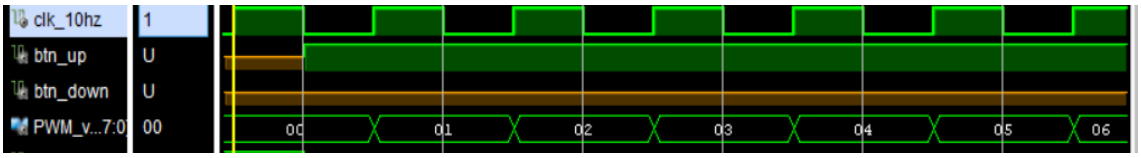

Imagen 5. Ilustración de las señales de clk_10hz, btn_up, btn_down y PWM_vector

En la imagen 5 tenemos un ejemplo de la simulación del módulo 2 donde mantenemos pulsado el botón ***btn_up*** y se observa como la salida ***PWM_vector***, siendo esta la misma que la señal nombrada anteriormente ***vec_temp***, incrementa 1% cada vez que llega un flanco de subida de la señal de reloj ***clk_10hz***.

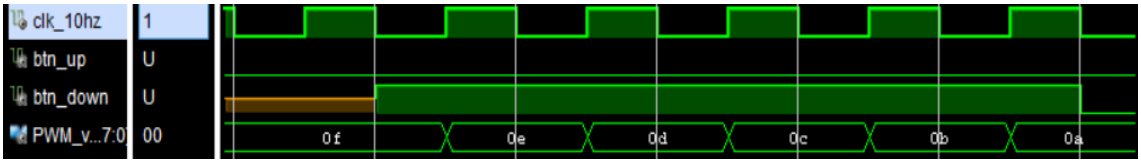

Imagen 6. Ilustración de las señales de clk_10hz, btn_up, btn_down y PWM_vector

En la imagen 6 simulamos que apretamos el botón ***btn_down*** haciendo que la salida ***PWM_vector*** disminuya una unidad por cada ciclo de reloj. De forma que se puede afirmar el correcto funcionamiento del módulo de selección del ciclo de trabajo de PWM.

2.3. MÓDULO DE VISUALIZACIÓN – M3

La función del módulo M3 es la de visualizar en los displays de la FPGA tanto la velocidad de giro del motor, como el ciclo de trabajo que tiene la señal PWM del módulo M1. Solo se puede representar un valor al mismo tiempo así que con la ayuda de un switch elegimos que valor mostramos en las pantallas, el valor del switch lo guardamos en una variable denominada ***sw_sel_disp.***

Otra función es la de indicar la dirección de giro dibujando un signo menos en el display de mayor peso, por lo tanto hay que usar 4 displays de los 8 disponible en la placa, 3 se usarán para mostrar el valor de la velocidad del motor, siendo esta como máximo 150 rpm, y ciclo de trabajo de la señal PWM que como máximo es el 100 % y el cuarto display es el de la dirección, mostrando un signo menos si la señal ***sw_Dir*** es igual a 0 en caso contrario no mostrara nada porque el sentido es positivo.

Para realizar el código tuvimos que crear un reloj de frecuencia 1kHz a partir del reloj de la FPGA, para que con los displays actualizasen los datos cada 1 ms, este tiempo es indetectable para el ojo humano por lo que desde nuestro punto de vista los números se verían de forma constante. Por otra parte, se ha tenido que usar dos componentes de conversión que nos han facilitado los profesores, uno de ellos hace la conversión de número a hexadecimal a BCD debido a que recibimos la información en hexadecimal desde los otros módulos y al representarlo en BCD es más práctico para representar cada digito en los tres displays. El segundo componente nos convierte el valor de BCD a un código de 7 segmentos, cuyo valor es que tenemos que enviar a la pantalla para que nos muestre el número deseado.

En el siguiente proceso seleccionamos el display y el valor que vamos a representar cada 1 ms, de forma que con cada flanco de subida mostramos la información en el siguiente. Esto lo hacemos que según el valor que tenga la variable ***sel***; entra en un if o en otro, indicando el valor que se convierte a código de 7 segmentos, poniendo a 0 el ánodo del display indicado y asignándole a ***sel*** el siguiente valor. Un ejemplo de simulación es que tras un reset la variable ***sel*** tiene un valor de '00' y entrará en el if correspondiente donde se le asigna a la variable **bcd_code,** las unidades y se

selecciona el display de menor peso, es decir, el de la derecha y se cambia el valor de ***sel*** a '01' para que al producirse un flanco en la señal de reloj creada anteriormente, ***clk_1khz***, entre en la siguiente comparación y así pasando por los cuatros displays de forma cíclica. En la ejecución del display de mayor peso se hace una comparación extra para indicar si se debe mostrar el menos o en caso contrario no mostrar nada.

Seguimos con el proceso de visualización en el cual comprobamos el nivel del switch, ***sw_sel_disp***, para guardar en una señal vectorial denominada ***vector***, el valor de la velocidad si el nivel es bajo y si es alto se guardará el ciclo de trabajo. Ambos en hexadecimal para que solo se tenga que convertir una señal.

El último proceso que se puede encontrar dentro del código VHDL del módulo M3 es el de comprobar si la señal ***sw_Dir*** tiene un nivel bajo lo que indica que el sentido de giro del motor es negativo y se debe mostrar el menos en el cuarto display, en ese caso asignamos a una variable auxiliar, ***aux***, que será comprobado en el proceso de selección el valor 1. Si ***sw_Dir*** está a nivel alto el sentido de giro es positivo y ***aux*** será 0.

Como se indica al principio de este punto, en la FPGA NEXYS4 DDR tiene 8 displays por lo que los 4 de mayor peso serán desactivados poniendo a 1 sus ánodos.

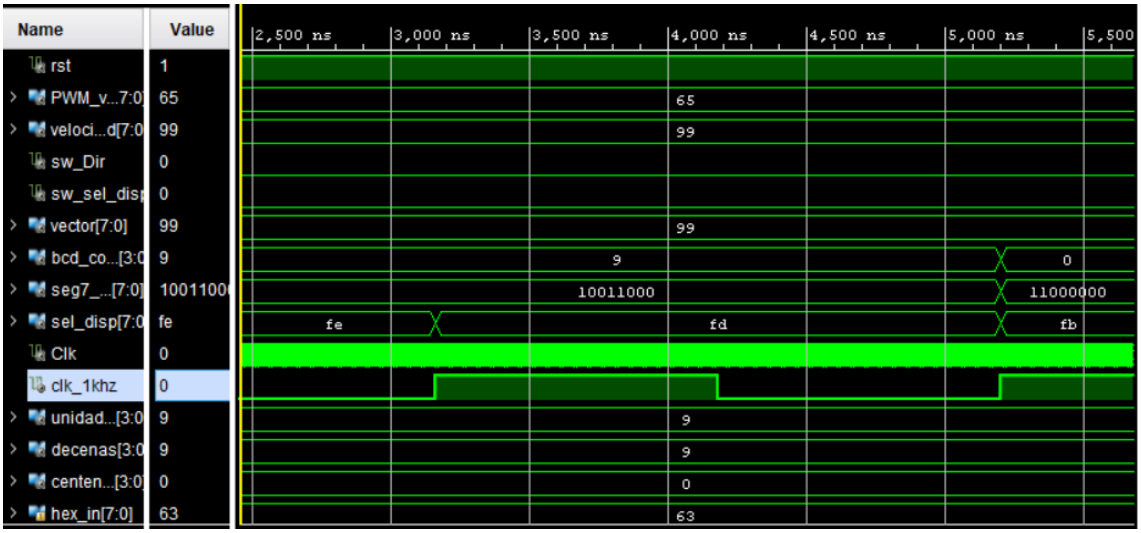

Imagen 7. Simulación del módulo M3

2.4. MÓDULO DE CALCULO DE VELOCIDAD – M4

Este es el último módulo que vamos a crear para esta práctica, en el nos encargamos de calcular la velocidad real del motor en base a la información de las señales ***pinSA*** y ***pinSB*** procedentes del encoder que se encuentra en el motor DC y de la relación del número de vueltas del motor. Las características que sabemos son que el encoder genera 3 pulsos por cada vuelta además que el motor tiene una reductora de 1:53 y su velocidad máxima es de 150 r.p.m.

El siguiente paso es el de calcular el número de pulsos que se produce en una de las señales del encoder, en nuestro caso hemos cogido ***pinSA,*** para hacer esto programamos 3 registros de forma que cuando se produce un flanco de subida de la señal ***CLK*** el biestable 1 pasa el valor de ***pinSA*** a ***SA1***, el biestable obtiene el valor de ***SA1*** y lo guarda en ***SA2*** y el último registro guarda el valor de ***SA2*** en ***SA3.*** Las condiciones que se aprecia son que solo se incrementa la variable ***npulso*** si las 3 salidas de los registros valen 1, ya que indica que se han producido 3 ciclos de reloj y por tanto un pulso, en caso de que se reciban un nivel alto durante menos de 3 ciclos se considera que es ruido, otro problema es si el pulso dura más de 3 ciclos y para evitar que siga aumentando el número de pulsos pusimos una condición que bloquea la suma (***var_cont*** = '0') de esta forma evitamos que el número de pulsos que obtiene en el procesos sea incorrecto.

Una vez que tenemos que el número de pulsos podemos sacar la velocidad mediante las siguientes relaciones, para facilitar los cálculos el periodo durante el cual estamos obteniendo el número de pulso del encoder es de 0.25 segundos.

$$velocidad_{ext} = \frac{npulsosxbase\_t}{53 * 3} * 60s = npulsos * \frac{4 * 60}{53 * 3} = npulsos * 1.50943$$

$$\approx npulsos * 1.5 \approx npulsos * \frac{3}{2}$$

De esta manera lo que podía ser una formula difícil de implementar, se nos queda como una multiplicación por 3, donde se usa el componente ***multiplicador*** que nos ha proporcionado el profesor y que se puede ejecutar con nuestra placa, y luego una división que suelen ser difícil de programar, pero como en este caso es entre 2 lo que

equivaldría a desplazar el resultado de la multiplicación un bit a la izquierda y añadirle un cero.

Creamos una señal llamada ***clk_4hz*** a partir de la señal de reloj de la FPGA para controlar los 0.25 segundos, periodo que estamos incrementando el número de pulsos, por lo que cuando se produce un nivel alto de esta señal reseteamos los 3 registros y la variable ***npuslos*** además se activa el componente ***multiplicador*** y obtendríamos un nuevo valor de la velocidad.

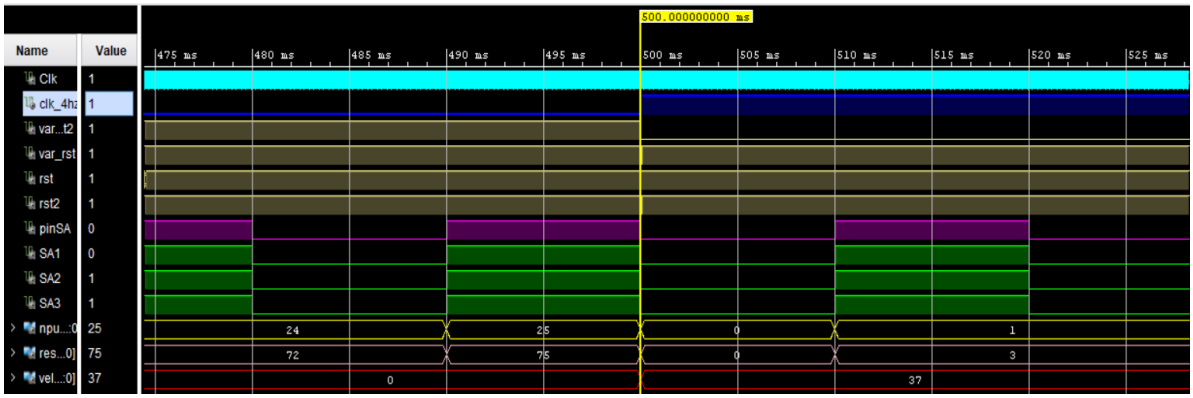

Imagen 8. Simulación del módulo M4

La imagen 8 es una captura de la pantalla del simulador de Vivado en la que podemos ver cómo funciona el módulo M4, para la comprobación indicamos en el testbench que tras un reset se produce una serie de pulsos de la señal ***pinSA*** con una frecuencia de 50 Hz, por lo tanto se incrementa la variable ***npulsos*** como se puede observar, cuyo valor introducimos al componente multiplicador y el valor que se obtiene se muestra en decimal en la señal de color rosa, ***resultado***. También se ve que cuando llega un flanco del reloj ***clk_4hz*** se ponen a 0 las variables ***SA1***, ***SA2***, ***SA3*** y ***npulsos***, de esta señal cogemos el último valor y lo dividimos entre 2 para ello y como la señal ***velocidad*** tiene 8 bits lo que hacemos es coger los bits desde 8 hasta el 1 de la variable ***resultado***. En el caso de la simulación mostrada en la imagen 8 la señal de color rojo representa el valor final de la velocidad de giro del motor.

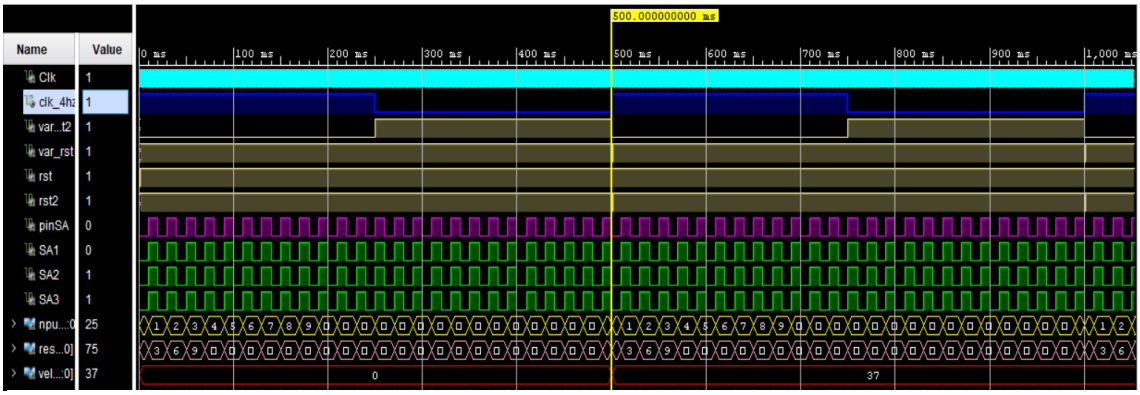

Imagen 9. Simulación del módulo M4

La imagen 9 es una simulación del bloque 4 en la que se muestra como tras un flanco de la señal clk_4hz, se vuelve a iniciar la cuenta de pulsos y se obtiene se actualiza el valor de la velocidad del giro del motor. También se observa como al estar recibiendo una señal pinSA con la misma frecuencia al llegar un flanco el valor sigue siendo el mismo debido a que la velocidad es constante durante toda la simulación.

2.5. MÓDULO TOTAL – M1-M2-M3-M4

A lo largo de la práctica hemos ido formando bloques según se terminaba de comprobar que se ejecutaba correcta el módulo de forma individual, por ejemplo, el bloque M1-M2, pero vamos a explicar el procedimiento para completar el bloque final donde añadimos los 4 módulos ya que la estructura que sigue todos esos bloques es la misma.

A este módulo lo denominamos ***contr_motor*** y para completarlo solo tenemos que hacer dos sencillos pasos, el primero es el de añadir los componentes de cada módulo, especificando el nombre y el tipo de datos que es y el segundo paso es el asignar las señales de cada componente con su correspondiente señal del módulo ***contr_motor*** para que los valores se transmitan y se pueda ejecutar todos los bloques, por lo general tienen el mismo nombre.

Las señales finales del módulo final son las que aparecen en la izquierda tanto de la imagen 10 como la imagen 11, siendo de entrada las señales: ***CLK, rst, btn_up, btn_down, sw_Dir, sw_sel_disp, pinSA*** y ***pinSB***. Por lo tanto, las salidas son ***pinDir, pinEN, seg7_code*** y ***sel_display.***

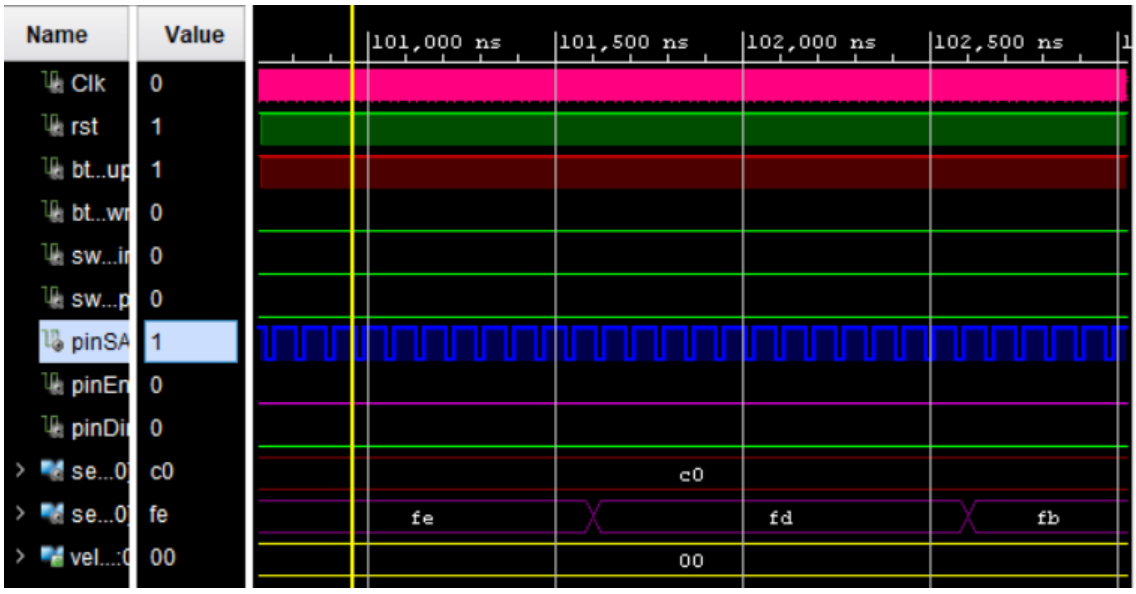

Imagen 10. Captura de la simulación del módulo final

En la imagen 10 se ve un fragmento de la simulación del módulo total para así apreciar bien el funcionamiento de la práctica, en esta parte queríamos que se apreciase como se genera los pulsos de la señal ***pinSA*** y como cambiamos la selección del ánodo de los 8 displays que se encuentran en la FPGA, a pesar de que el número que siempre tiene que representar es el mismos debido a que no se ha producido un flanco del reloj ***clk_4hz*** y la velocidad no haya actualizado su valor siendo esta 0.

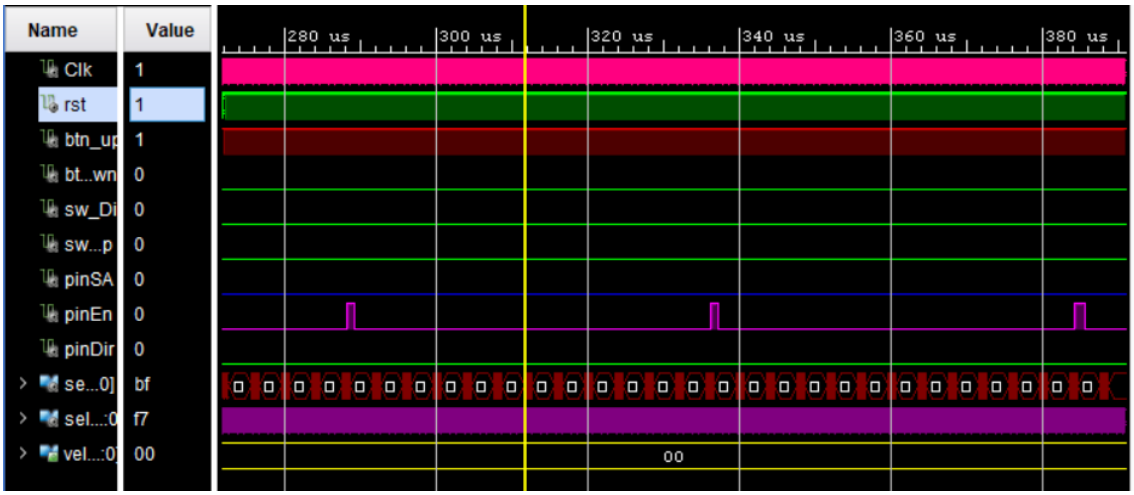

Imagen 11. Captura de la simulación del módulo final

En la imagen 11 se muestra otro fragmento de la simulación del módulo final en la que tras un tiempo con la señal ***btn_up*** activa se empieza a notar el cambio del ciclo de trabajo de la señal en color magenta, ***pinEN***, correspondiente con la señal PWM que introducimos al puente H para controlar el motor DC. También se puede ver que en el testbench le hemos indicado que la señal de control de la dirección de giro del motor, ***sw_Dir***, se encuentra a nivel bajo, al igual que la señal ***sw_sel_disp*** que nos indica que queremos visualizar la velocidad por los displays de la tarjeta NEXYS4 DRR.

2.6. RECURSOS UTILIZADOS

La utilización de slices de la FPGA para que funcionara correctamente el proyecto, fue la que se expone en la imagen 12.

Name	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	LUT Flip Flop Pairs (63400)	DSPs (240)	Bonded IOB (210)	BUFGCTRL (32)
contr_motor	150	161	77	150	90	1	25	1
M1_gen_PWM_1 (M1_...	14	14	7	14	9	0	0	0
M2_sel_PWM_1 (M2_s...	36	21	17	36	14	0	0	0
M3_visualiza_1 (M3_vi...	37	51	35	37	17	0	0	0
cto_convertir (hex2b...	18	10	10	18	6	0	0	0
cto_visualizar (bcd2...	4	0	3	4	0	0	0	0
M4_calc_veloc_1 (M4_...	65	75	29	65	49	1	0	0
cto_multiplicador (...	1	0	1	1	0	1	0	0

Imagen 12. Report utilization de la FPGA

En general, la utilización ha sido menor al 1%. A continuación, se presenta el consumo energético en la FPGA.

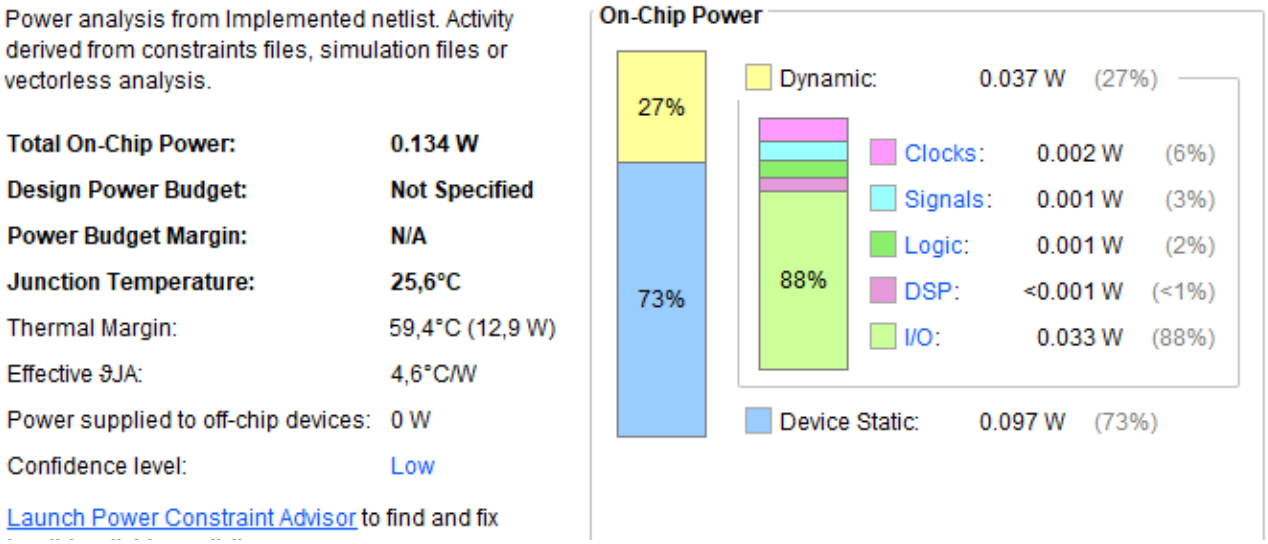

Imagen 13. Report Power de la FPGA Nexys 4 DDR

Se puede observar que el consumo energético predominante ha sido el de las entradas/salidas de la FPGA.

3. CONCLUSIONES

Teniendo el proyecto acabado, y funcionando correctamente, recordamos todos los problemas que hemos tenido durante la realización de esta práctica. Ante todo, hay que mencionar que el lenguaje VHDL es un tipo de lenguaje muy estructurado que en muchas ocasiones no deja margen de maniobra. Por lo que hay que tener cuidado y siempre tener en mente las restricciones que nos depara VHDL.

Los problemas durante la puesta en marcha de la práctica fueron muy variados; desde errores de concepto, hasta errores de no entender cómo debería de funcionar en realidad el motor y el encoder, tirando más hacia el segundo. Los errores de concepto fueron los que nos explicaron desde el primer día, NO se debe utilizar más de un reloj en un proyecto, ya que eso tiende a provocar errores. El módulo 4 en el cual calculamos la velocidad real del motor, nos resultó problemático, ya que no entendíamos como se debía leer el pin del encoder.

Teniendo como herramienta de trabajo Vivado para la simulación de nuestros módulos, nos facilitó mucho el trabajo, viendo lo que realmente hace el código en VHDL que hemos implementado. Aunque en ocasiones, nos confundía. Nos confundía debido a que se pueden simular códigos, que posteriormente no se pueden sintetizar e implementar en la placa FPGA.

En general, ha sido una práctica que nos ha aportado bastante conceptualmente; esto es, hemos entendido la mayoría de las restricciones que tiene VHDL y el beneficio que tiene trabajar con este lenguaje.