**SEGUNDA SESIÓN DE DISEÑO AVANZADO DE SISTEMAS DIGITALES**

**CONTROL DE MOTORES DE CONTINUA, SERVOS Y PASO A PASO**

Después de conocer la técnica de pwm en diodos, el siguiente paso es usar el mismo enfoque en motores de continua, paso a paso y servos. Los diseños son:

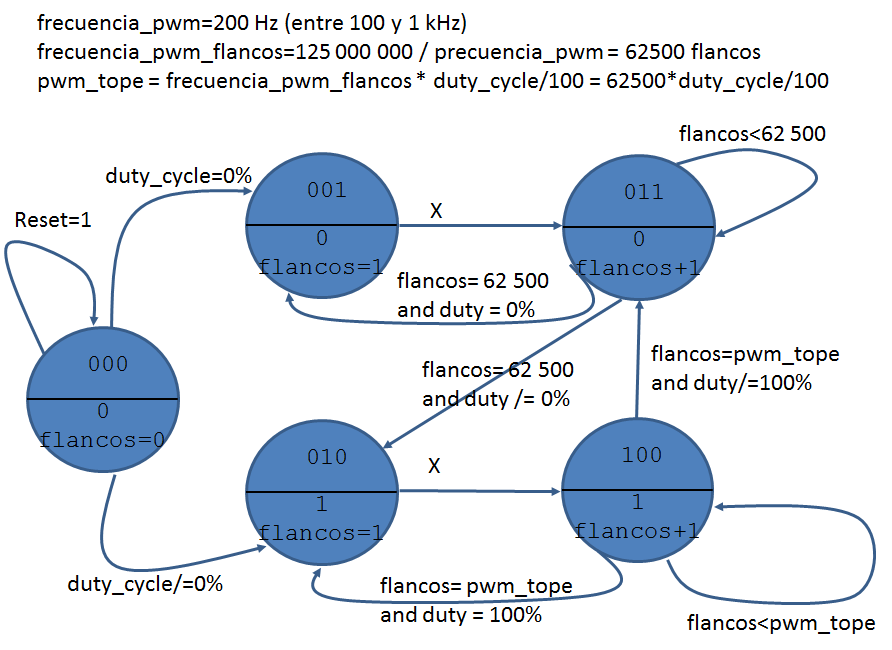
* PWM para un motor de continua (el típico PWM) con una frecuencia de 200 Hz (típica en PWM) y por tanto hay que ver el resultado en el propio motor y sobre todo en un osciloscopio.
* PWM con control “fino” en el cambio de duty cycle para evitar cambios bruscos en el motor.
* PWM para un servo. Se le denomina también PWM, aunque no debería ser así.
* Control de motor paso a paso.
* Opcional. Se ofrece para el trabajo opcional el control “suave” de un PWM de continua con cambio de sentido.

En ambos PWM se describe el comportamiento usando gráficas, texto y el DTE del correspondiente autómata de Moore.

**PWM REAL para MOTORES DE CONTINUA: lab\_22**

El siguiente código es para un PWM de continua normal, en este caso la frecuencia elegida es de 200 Hz (aunque se puede cambiar). La estructura es igual a la anterior, solo que el contador de flancos cambia en su tope y que hay una salida para el osciloscopio que antes no había ya que iba sobre un led.

El siguiente autómata estaba hecho para una ZYBO cuya frecuencia es de 125 MHz, pero para una BASYS la frecuencia es de 100 MHz por tanto donde pone 125 000 000 debe poner 100 000 000 y donde pone 625 000 debe poner 500 000.



Variables:

* frecuencia\_pwm: está entre 100 Hz y 1 kHz, se suele elegir 200 Hz. Este valor se usa como signal.
* frecuencia\_pwm\_flancos: como el reloj del sistema es de 100 MHz, entonces la frecuencia de 200 Hz consiste en 500000 flancos para un periodo de 5 ms (para los 200 Hz)
* pwm\_tope: indica cuántos flancos hay que contar para un determinado duty\_cycle (que va de 0% a 100%), el mínimo es 0 y el máximo es 500000.

Además, y ya que este código es real, hay que tener en cuenta que hay que añadir un driver al motor, ya que la tarjeta Zybo por sí sola no “puede” con el motor.

Para seleccionar el duty\_cycle se usan los 4 interruptores de la BASYS (luego este dato vendrá de otro process) y son porcentuales. Hay que tener en cuenta que de 1011 a 1111 no son combinaciones aceptadas y el duty\_cycle queda a 0%, así que CUIDADO.

ADEMÁS este programa no tiene control del sentido de giro, es decir, siempre gira en el mismo sentido y con los 4 sw solo se controla la velocidad de giro. Lo anterior significa que en el montaje uno de los cables del motor siempre está conectado a tierra (o a Vcc si se quiere que vaya en sentido inverso).

Más adelante habrá otro programa con cambio de sentido.

La velocidad de giro final a la vista depende mucho del motor, de su reductora, de su potencia, etc.

|  |  |
| --- | --- |
| Duty\_cycle | SW3-SW0 |
| 0 % | 0000 |
| 10 % | 0001 |
| 20 % | 0010 |
| 30 % | 0011 |
| 40 % | 0100 |
| 50 % | 0101 |
| 60 % | 0110 |
| 70 % | 0111 |
| 80 % | 1000 |
| 90 % | 1001 |
| 100 % | 1010 |
| 0 % | 1011-1111 |

El resultado que se puede ver en el osciloscopio es:

|  |  |
| --- | --- |
| 10%, periodo de 5 ms y 1 cuadro/ms | 50%, %, periodo de 5 ms y 1 cuadro/ms |
| D:\SCR0010.BMP | D:\SCR0011.BMP |
| 80%, %, periodo de 5 ms y 1 cuadro/ms | 100%, %, periodo de 5 ms y 1 cuadro/ms |
| D:\SCR0012.BMP | D:\SCR0013.BMP |
|  |  |

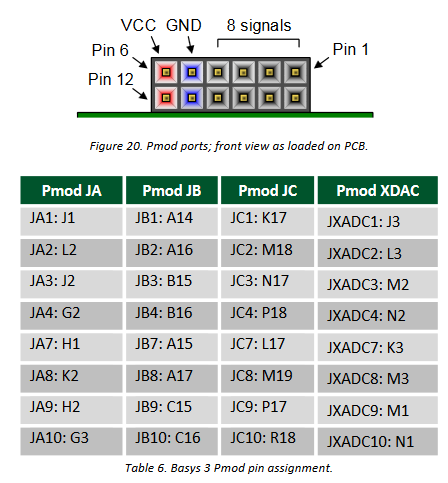
MONTAJE DE LA PRÁCTICA

El montaje tiene dos partes: la conexión del motor a la placa BASYS y la etapa de potencia del motor. Sin la etapa de potencia el motor no se moverá ya que la corriente máxima entregada por la BASYS es muy baja (buscarla).

Hay muchos chips de potencia basados en Darlington. Nosotros usamos uno de MSE que permite controlar hasta cuatro motores de continua o dos paso a paso. Con cualquier otro circuito simplemente hay que seguir las indicaciones.

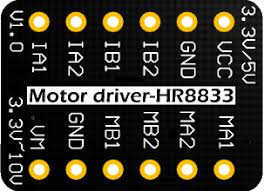
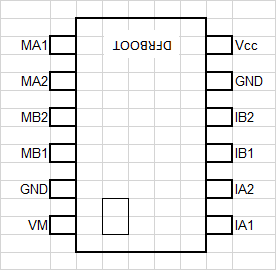
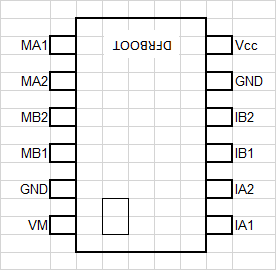
Por un lado se alimenta el chip con +5 V y GND, y en ese mismo conector se conecta la tensión de ataque al motor, se denomina V+ y generalmente se conecta a +5 V, pero se puede llevar a +12 V o lo que sea, según sea el motor. Por otra parte las líneas llamadas S1 y S2 (o S3 y S4) se conectan a los dos cables del motor (es la etapa de potencia), mientras que los dos cables que vienen de la Zybo se conectan en los conectores E1 y E2 (o en E3 y E4). En el caso de un motor de continua se pude considerar irrelevante el orden de los cables, sobre todo en pruebas (lo más que puede pasar es que gire en un sentido en vez de en el otro).

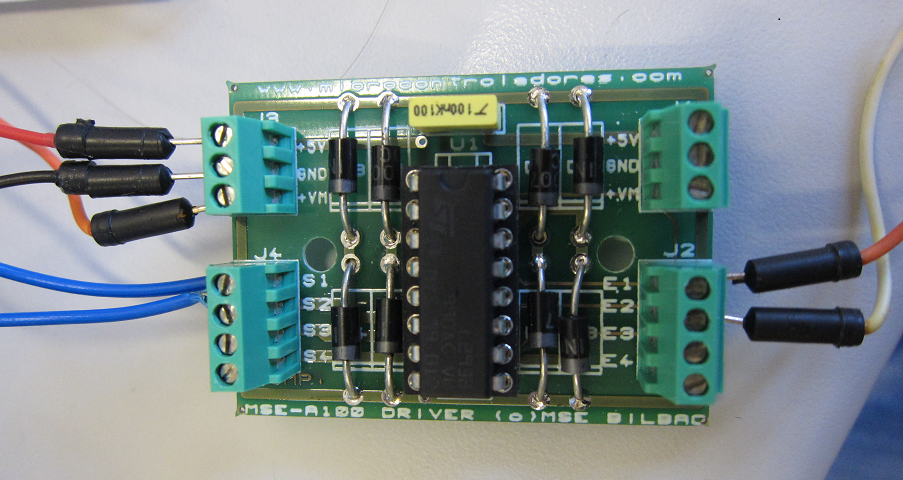
En este diseño además hay que usar un PMOD de la BASYS. Hay cuatro y de ellos hay que usar el A, B o C. La descripción de los pines y la numeración está en la figura siguiente. En J1 se sacará el pwm de control del motor.

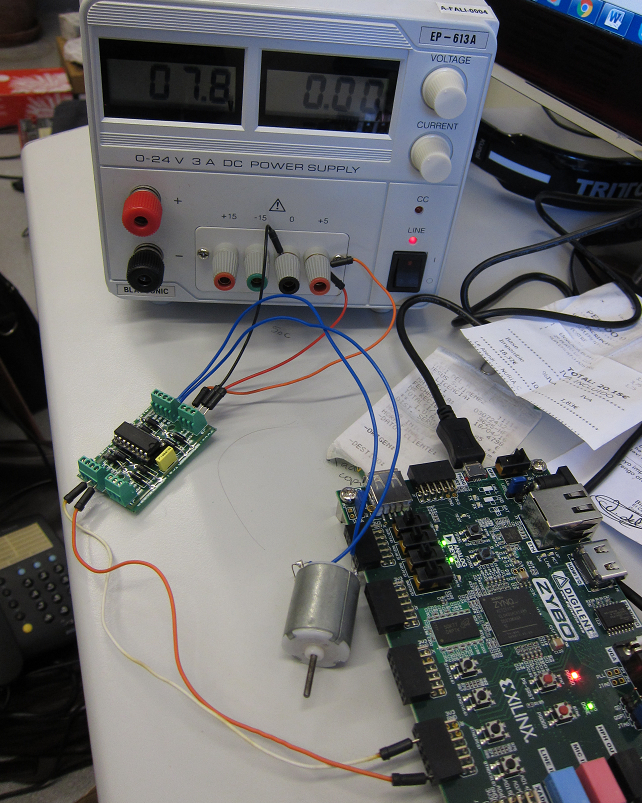


La conexión del motor de continua exige seguir ciertos pasos:

* Disponer de un driver de potencia habiendo leído antes el consumo del motor.
* Si se elige el HR8833 de DFRBOOT este entrega hasta 1,5 A y se alimenta entre 3,3 V y 10 V. Conectarlo con cuidado.
* El HR8833 se alimenta para el motor, VM y GND, desde una fuente de alimentación. VM está entre 3,3 V y 10 V, a más voltaje más velocidad y consumo. Parte de potencia.
* El HR8833 se alimenta desde la BASYS, Vcc y GND, con 3,3 V y GND. Parte lógica.
* La salida de la BASYS (la señal lógica) se conecta a una o varias de IA1, IA2, IB2 e IB1.
* Las salidas de potencia, MA1, MA2, MB1 y MB2 se conectan al motor de forma adecuada.
* Las señales lógicas y las de potencia deben estar conectadas formando parejas: IA1-MA1, IA2-MA2, etc.
* Hay que tener cuidado con cómo se suelda y se conecta el chip a la protoboard.
* Si se enciende la luz roja, quiere decir que se corta la potencia por exceso de consumo.
* Los motores de continua que no son buenos o son viejos suelen estar algo “atascados” e ir a tirones, con lo que el consumo se suele disparar y por tanto se corta la potencia. En este caso la mejor recomendación es bajar el voltaje de alimentación de VM.





Descripción VHDL

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity pwm\_motor\_DC is

port(

clk: in std\_logic;

btn: in std\_logic;

sw: in std\_logic\_vector (3 downto 0);

pwm\_motor\_DC: out std\_logic;

led: out std\_logic\_vector (3 downto 0)

);

end pwm\_motor\_DC;

architecture Behavioral of pwm\_motor\_DC is

signal reset: std\_logic;

signal pwm: std\_logic;

signal selector: std\_logic\_vector (3 downto 0);

signal duty\_cycle: integer range 0 to 100;

signal contador\_base: integer range 0 to 1000000;

signal pwm\_tope: integer range 0 to 10000000;

signal estado: std\_logic\_vector (2 downto 0);

signal frecuencia\_pwm: integer range 0 to 1000;

signal frecuencia\_pwm\_flancos: integer range 0 to 100000000;

begin

reset<=btn;

led(0)<=pwm;

selector<=sw;

led(3 downto 1)<=estado;

pwm\_motor\_DC<=pwm; --para ver la señal en un oscilo

frecuencia\_pwm<=200; --este valor controla la frecuencia del pwm

--duty cycle va de 0 a 100%, con 0 el pwm siempre esta a 0 y con 100 siempre está a 1.

-- el valor es proporcional

process(selector)

begin

case selector is

when "0000" => duty\_cycle<=0;

when "0001" => duty\_cycle<=10;

when "0010" => duty\_cycle<=20;

when "0011" => duty\_cycle<=30;

when "0100" => duty\_cycle<=40;

when "0101" => duty\_cycle<=50;

when "0110" => duty\_cycle<=60;

when "0111" => duty\_cycle<=70;

when "1000" => duty\_cycle<=80;

when "1001" => duty\_cycle<=90;

when "1010" => duty\_cycle<=100;

when others => duty\_cycle<=0;

end case;

end process;

-- frecuencia\_pwm\_flancos indica cuántos flancos del reloj del sistema hacen

-- falta para generar el periodo de PWM contando flancos flancos.

-- Por ejemplo para 100 Hz debe contar 1250000 flancos

process(frecuencia\_pwm)

begin

frecuencia\_pwm\_flancos<=100000000/frecuencia\_pwm;

end process;

-- pwm\_tope expresa en flancos cuántos flancos debe estar la salida a 1

process(duty\_cycle)

begin

pwm\_tope<=100000000/frecuencia\_pwm\*duty\_cycle/100;

end process;

process(clk, reset)

begin

if clk='1' and clk'event then

case estado is

--estado reset

when "000" => contador\_base<=0;

if reset='1' then

estado<="000";

else

if duty\_cycle=0 then

estado<="001";

else

estado<="010";

end if;

end if;

--estado de comienzo pero duty\_cycle es 0%, o sea pwm inactivo todo el rato

when "001" => contador\_base<=1;

if reset='1' then

estado<="000";

else

estado<="011";

end if;

--estado que comienza con el pwm a 1

when "010" => contador\_base<=1;

if reset='1' then

estado<="000";

else

estado<="100";

end if;

--estado que rellena los 0 hasta el final

when "011" => contador\_base<=contador\_base+1;

if reset='1' then

estado<="000";

else

if contador\_base<frecuencia\_pwm\_flancos then

estado<="011";

else

if duty\_cycle=0 then

estado<="001";

else

estado<="010";

end if;

end if;

end if;

--estado que desarrolla el pwm a 1

when "100" => contador\_base<=contador\_base+1;

if reset='1' then

estado<="000";

else

if contador\_base=pwm\_tope then

if duty\_cycle=100 then

estado<="010";

else

estado<="011";

end if;

else

estado<="100";

end if;

end if;

when others => estado<="000";

contador\_base<=0;

end case;

end if;

end process;

process(estado)

begin

case estado is

when "000" => pwm<='0';

when "001" => pwm<='0';

when "010" => pwm<='1';

when "011" => pwm<='0';

when "100" => pwm<='1';

when others => pwm<='0';

end case;

end process;

end Behavioral;

*Constraints:*

## Clock signal

set\_property PACKAGE\_PIN W5 [get\_ports clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports clk]

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports clk]

## Switches

set\_property PACKAGE\_PIN V17 [get\_ports {sw[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[0]}]

set\_property PACKAGE\_PIN V16 [get\_ports {sw[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[1]}]

set\_property PACKAGE\_PIN W16 [get\_ports {sw[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[2]}]

set\_property PACKAGE\_PIN W17 [get\_ports {sw[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[3]}]

## LEDs

set\_property PACKAGE\_PIN U16 [get\_ports {led[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[0]}]

set\_property PACKAGE\_PIN E19 [get\_ports {led[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[1]}]

set\_property PACKAGE\_PIN U19 [get\_ports {led[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[2]}]

set\_property PACKAGE\_PIN V19 [get\_ports {led[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[3]}]

## BUTTONs

set\_property PACKAGE\_PIN U18 [get\_ports btn]

set\_property IOSTANDARD LVCMOS33 [get\_ports btn]

##Pmod Header JA

##Sch name = JA1

set\_property PACKAGE\_PIN J1 [get\_ports {pwm\_motor\_DC}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {pwm\_motor\_DC}]

##Sch name = JA2

#set\_property PACKAGE\_PIN L2 [get\_ports {pwm\_motor\_DC}]

#set\_property IOSTANDARD LVCMOS33 [get\_ports {pwm\_motor\_DC}]

**PWM PARA CONTINUA CON SENTIDO DE GIRO**

La idea es similar a la anterior pero en este caso vamos a disponer de sentido de giro.

Para ello simplemente usamos los cuatro interruptores en binario puro en C2, así los valores van de -7 (1001) a +7 (0111) (no usamos el 1000, es decir, el -8). Y además debe haber una salida adicional para controlar el sentido de giro, que antes estaba conectada a GND o Vcc.

Además hay que tener en cuenta una cosa, el motor de continua se mueve por diferencia de potencial entre sus dos cables. Entonces, para cambiar el sentido se puede hacer de varias formas (una es cambiar el sentido de giro de GND a Vcc, y etc.), pero lo más fácil es que cuando cambie el giro intercambiemos las señales. Es decir, que el sentido sea el pwm, y que el pwm sea el sentido (si se piensa es muy sencillo).

En cualquier caso ahora el control del motor tiene dos cables, que van a ser los pines 1 y 2 del PMOD JA.

CUIDADO porque para usar este diseño hay que saber binario puro con signo en C2.

Descripción VHDL:

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

use IEEE.std\_logic\_unsigned.all;

entity pwm\_motor\_DC is

port(

clk: in std\_logic;

btn: in std\_logic;

sw: in std\_logic\_vector (3 downto 0);

pwm\_motor\_DC: out std\_logic;

sentido\_motor\_DC: out std\_logic;

led: out std\_logic\_vector (3 downto 0)

);

end pwm\_motor\_DC;

architecture Behavioral of pwm\_motor\_DC is

signal reset: std\_logic;

signal pwm: std\_logic;

signal selector: std\_logic\_vector (3 downto 0);

signal selector\_sin\_signo: std\_logic\_vector (2 downto 0);

signal duty\_cycle: integer range 0 to 100;

signal contador\_base: integer range 0 to 1000000;

signal pwm\_tope: integer range 0 to 10000000;

signal estado: std\_logic\_vector (2 downto 0);

signal frecuencia\_pwm: integer range 0 to 1000;

signal frecuencia\_pwm\_flancos: integer range 0 to 100000000;

begin

reset<=btn;

led(0)<=pwm;

selector<=sw;

led(3 downto 1)<=estado;

frecuencia\_pwm<=200; --este valor controla la frecuencia del pwm

--duty cycle va de 0 a 100%, con 0 el pwm siempre esta a 0 y con 100 siempre está a 1.

-- el valor es proporcional

process(selector)

begin

if selector(3)='1' then

selector\_sin\_signo<=not(selector(2 downto 0)) + "001";

else

selector\_sin\_signo<=selector (2 downto 0);

end if;

end process;

process(selector, pwm)

begin

if selector(3)='0' then --giro normal

sentido\_motor\_DC<='0';

pwm\_motor\_DC<=pwm; --para llevar la señal al motor

else

sentido\_motor\_DC<=pwm;

pwm\_motor\_DC<='0'; --para llevar la señal al motor

end if;

end process;

process(selector\_sin\_signo)

begin

case selector\_sin\_signo is

when "000" => duty\_cycle<=0;

when "001" => duty\_cycle<=10;

when "010" => duty\_cycle<=25;

when "011" => duty\_cycle<=50;

when "100" => duty\_cycle<=60;

when "101" => duty\_cycle<=75;

when "110" => duty\_cycle<=90;

when "111" => duty\_cycle<=100;

when others => duty\_cycle<=0;

end case;

end process;

-- frecuencia\_pwm\_flancos indica cuántos flancos del reloj del sistema hacen

-- falta para generar el periodo de PWM contando flancos flancos.

-- Por ejemplo para 100 Hz debe contar 1250000 flancos

process(frecuencia\_pwm)

begin

frecuencia\_pwm\_flancos<=100000000/frecuencia\_pwm;

end process;

-- pwm\_tope expresa en flancos cuántos flancos debe estar la salida a 1

process(duty\_cycle)

begin

pwm\_tope<=100000000/frecuencia\_pwm\*duty\_cycle/100;

end process;

process(clk, reset)

begin

if clk='1' and clk'event then

case estado is

--estado reset

when "000" => contador\_base<=0;

if reset='1' then

estado<="000";

else

if duty\_cycle=0 then

estado<="001";

else

estado<="010";

end if;

end if;

--estado de comienzo pero duty\_cycle es 0%, o sea pwm inactivo todo el rato

when "001" => contador\_base<=1;

if reset='1' then

estado<="000";

else

estado<="011";

end if;

--estado que comienza con el pwm a 1

when "010" => contador\_base<=1;

if reset='1' then

estado<="000";

else

estado<="100";

end if;

--estado que rellena los 0 hasta el final

when "011" => contador\_base<=contador\_base+1;

if reset='1' then

estado<="000";

else

if contador\_base<frecuencia\_pwm\_flancos then

estado<="011";

else

if duty\_cycle=0 then

estado<="001";

else

estado<="010";

end if;

end if;

end if;

--estado que desarrolla el pwm a 1

when "100" => contador\_base<=contador\_base+1;

if reset='1' then

estado<="000";

else

if contador\_base=pwm\_tope then

if duty\_cycle=100 then

estado<="010";

else

estado<="011";

end if;

else

estado<="100";

end if;

end if;

when others => estado<="000";

contador\_base<=0;

end case;

end if;

end process;

process(estado)

begin

case estado is

when "000" => pwm<='0';

when "001" => pwm<='0';

when "010" => pwm<='1';

when "011" => pwm<='0';

when "100" => pwm<='1';

when others => pwm<='0';

end case;

end process;

end Behavioral;

Fichero de restricciones:

## Clock signal

set\_property PACKAGE\_PIN W5 [get\_ports clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports clk]

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports clk]

## Switches

set\_property PACKAGE\_PIN V17 [get\_ports {sw[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[0]}]

set\_property PACKAGE\_PIN V16 [get\_ports {sw[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[1]}]

set\_property PACKAGE\_PIN W16 [get\_ports {sw[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[2]}]

set\_property PACKAGE\_PIN W17 [get\_ports {sw[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[3]}]

## LEDs

set\_property PACKAGE\_PIN U16 [get\_ports {led[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[0]}]

set\_property PACKAGE\_PIN E19 [get\_ports {led[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[1]}]

set\_property PACKAGE\_PIN U19 [get\_ports {led[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[2]}]

set\_property PACKAGE\_PIN V19 [get\_ports {led[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[3]}]

## BUTTONs

set\_property PACKAGE\_PIN U18 [get\_ports btn]

set\_property IOSTANDARD LVCMOS33 [get\_ports btn]

##Pmod Header JA

##Sch name = JA1

set\_property PACKAGE\_PIN J1 [get\_ports {pwm\_motor\_DC}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {pwm\_motor\_DC}]

##Sch name = JA2

set\_property PACKAGE\_PIN L2 [get\_ports {sentido\_motor\_DC}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sentido\_motor\_DC}]

**PWM PARA CONTINUA CON SENTIDO DE GIRO Y MEJORA SIN SALTO EN VELOCIDAD DEL MOTOR lab\_24**

La última mejora al control pwm del motor de continua es evitar los saltos bruscos, por ejemplo si se cambia el selector de 10% a 20%, este cambio es brusco, y todavía es mayor cuando se produce un cambio de sentido, por ejemplo solo cambiando un bit en los interruptores se puede pasar de 0001 (+10%) a 1001 (-100%), ), lo que no solo produce un pico de intensidad muy peligroso (no por consumo, sino por ruido) sino también hace que el motor sufra mecánicamente, reduciendo su duración y calidad de uso.

En este caso (y respecto del anterior código) la solución pasa por controlar dos situaciones:

* saber si ha cambiado la selección del usuario (interruptores) y
* si hay cambio, hacerlo de 1% en 1% (se podría hacer en menos) hasta que el nuevo duty cycle alcance el nuevo valor.

La solución pasa entonces por leer selector y con su valor generar duty\_cycle\_aux. El valor de duty\_cycle será el de duty\_cycle\_aux, excepto cuando ambo sean distintos, en cuyo caso duty\_cycle va creciendo/decreciendo de 1% en 1% hasta alcanzar el nuevo valor. Este incremento/decremento se hace síncronamente a una frecuencia de 200 Hz, no tiene sentido hacerlo más rápido (a 100 MHz), porque entonces se volvería loco el pwm y el propio motor. No tiene porqué ser 200 Hz, puede ser otra frecuencia.

El cambio suave de velocidad de giro se ve muy bien con el osciloscopio.

El código incluido a continuación NO INCLUYE el control suave del motor por cambio de signo, solo por cambio de valor.

RETO: modificar el VHDl para gestione el cambio de signo.

Descripción VHDL:

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

use IEEE.std\_logic\_unsigned.all;

entity pwm\_motor\_DC is

port(

clk: in std\_logic;

btn: in std\_logic;

sw: in std\_logic\_vector (3 downto 0);

pwm\_motor\_DC: out std\_logic;

sentido\_motor\_DC: out std\_logic;

led: out std\_logic\_vector (3 downto 0)

);

end pwm\_motor\_DC;

architecture Behavioral of pwm\_motor\_DC is

signal reset: std\_logic;

signal pwm: std\_logic;

signal selector: std\_logic\_vector (3 downto 0);

signal selector\_sin\_signo: std\_logic\_vector (2 downto 0);

signal duty\_cycle: integer range 0 to 100;

signal contador\_base: integer range 0 to 1000000;

signal pwm\_tope: integer range 0 to 10000000;

signal estado: std\_logic\_vector (2 downto 0);

signal frecuencia\_pwm: integer range 0 to 1000;

signal frecuencia\_pwm\_flancos: integer range 0 to 100000000;

signal duty\_cycle\_aux: integer range 0 to 100;

signal contador\_aux\_duty\_cycle: integer range 0 to 500000; -- 200 Hz

begin

reset<=btn;

led(0)<=pwm;

selector<=sw;

led(3 downto 1)<=estado;

frecuencia\_pwm<=200; --este valor controla la frecuencia del pwm

--duty cycle va de 0 a 100%, con 0 el pwm siempre esta a 0 y con 100 siempre está a 1.

-- el valor es proporcional

process(selector)

begin

if selector(3)='1' then

selector\_sin\_signo<=not(selector(2 downto 0)) + "001";

else

selector\_sin\_signo<=selector (2 downto 0);

end if;

end process;

process(selector, pwm)

begin

if selector(3)='0' then --giro normal

sentido\_motor\_DC<='0';

pwm\_motor\_DC<=pwm; --para llevar la señal al motor

else

sentido\_motor\_DC<=pwm;

pwm\_motor\_DC<='0'; --para llevar la señal al motor

end if;

end process;

process(selector\_sin\_signo)

begin

case selector\_sin\_signo is

when "000" => duty\_cycle\_aux<=0;

when "001" => duty\_cycle\_aux<=10;

when "010" => duty\_cycle\_aux<=25;

when "011" => duty\_cycle\_aux<=50;

when "100" => duty\_cycle\_aux<=60;

when "101" => duty\_cycle\_aux<=75;

when "110" => duty\_cycle\_aux<=90;

when "111" => duty\_cycle\_aux<=100;

when others => duty\_cycle\_aux<=0;

end case;

end process;

-- contador para generar una señal de 200 Hz para incrementar/decrementar el duty cycle

process(clk, reset)

begin

if clk='1' and clk'event then

if reset='1' then

contador\_aux\_duty\_cycle<=0;

else

if contador\_aux\_duty\_cycle=500000 then

contador\_aux\_duty\_cycle<=0;

else

contador\_aux\_duty\_cycle<= contador\_aux\_duty\_cycle+1;

end if;

end if;

end if;

end process;

-- Mientras el duty\_cycle no cambie, mantiene su valor, pero cuando cambie selector y con´el

-- el duty\_cycle, entonces llegará al nuevo valor de 1% en 1%, y no de 10%, lo hace una

-- velocidad de incremento de 200 Hz (como el pwm)

process(clk, reset)

begin

if clk='1' and clk'event then

if reset='1' then

duty\_cycle<=0;

else

if contador\_aux\_duty\_cycle=0 then

if duty\_cycle/=duty\_cycle\_aux then

if duty\_cycle\_aux>duty\_cycle then

duty\_cycle<=duty\_cycle+1;

elsif duty\_cycle\_aux<duty\_cycle then

duty\_cycle<=duty\_cycle-1;

end if;

end if;

end if;

end if;

end if;

end process;

-- frecuencia\_pwm\_flancos indica cuántos flancos del reloj del sistema hacen

-- falta para generar el periodo de PWM contando flancos flancos.

-- Por ejemplo para 100 Hz debe contar 1250000 flancos

process(frecuencia\_pwm)

begin

frecuencia\_pwm\_flancos<=100000000/frecuencia\_pwm;

end process;

-- pwm\_tope expresa en flancos cuántos flancos debe estar la salida a 1

process(duty\_cycle)

begin

pwm\_tope<=100000000/frecuencia\_pwm\*duty\_cycle/100;

end process;

process(clk, reset)

begin

if clk='1' and clk'event then

case estado is

--estado reset

when "000" => contador\_base<=0;

if reset='1' then

estado<="000";

else

if duty\_cycle=0 then

estado<="001";

else

estado<="010";

end if;

end if;

--estado de comienzo pero duty\_cycle es 0%, o sea pwm inactivo todo el rato

when "001" => contador\_base<=1;

if reset='1' then

estado<="000";

else

estado<="011";

end if;

--estado que comienza con el pwm a 1

when "010" => contador\_base<=1;

if reset='1' then

estado<="000";

else

estado<="100";

end if;

--estado que rellena los 0 hasta el final

when "011" => contador\_base<=contador\_base+1;

if reset='1' then

estado<="000";

else

if contador\_base<frecuencia\_pwm\_flancos then

estado<="011";

else

if duty\_cycle=0 then

estado<="001";

else

estado<="010";

end if;

end if;

end if;

--estado que desarrolla el pwm a 1

when "100" => contador\_base<=contador\_base+1;

if reset='1' then

estado<="000";

else

if contador\_base=pwm\_tope then

if duty\_cycle=100 then

estado<="010";

else

estado<="011";

end if;

else

estado<="100";

end if;

end if;

when others => estado<="000";

contador\_base<=0;

end case;

end if;

end process;

process(estado)

begin

case estado is

when "000" => pwm<='0';

when "001" => pwm<='0';

when "010" => pwm<='1';

when "011" => pwm<='0';

when "100" => pwm<='1';

when others => pwm<='0';

end case;

end process;

end Behavioral;

Fichero de restricciones:

## Clock signal

set\_property PACKAGE\_PIN W5 [get\_ports clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports clk]

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports clk]

## Switches

set\_property PACKAGE\_PIN V17 [get\_ports {sw[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[0]}]

set\_property PACKAGE\_PIN V16 [get\_ports {sw[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[1]}]

set\_property PACKAGE\_PIN W16 [get\_ports {sw[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[2]}]

set\_property PACKAGE\_PIN W17 [get\_ports {sw[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[3]}]

## LEDs

set\_property PACKAGE\_PIN U16 [get\_ports {led[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[0]}]

set\_property PACKAGE\_PIN E19 [get\_ports {led[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[1]}]

set\_property PACKAGE\_PIN U19 [get\_ports {led[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[2]}]

set\_property PACKAGE\_PIN V19 [get\_ports {led[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[3]}]

## BUTTONs

set\_property PACKAGE\_PIN U18 [get\_ports btn]

set\_property IOSTANDARD LVCMOS33 [get\_ports btn]

##Pmod Header JA

##Sch name = JA1

set\_property PACKAGE\_PIN J1 [get\_ports {pwm\_motor\_DC}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {pwm\_motor\_DC}]

##Sch name = JA2

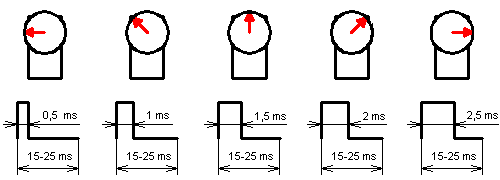
set\_property PACKAGE\_PIN L2 [get\_ports {sentido\_motor\_DC}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sentido\_motor\_DC}]

**PWM PARA SERVO**

La idea es similar pero algo distinta. Para empezar un servo no gira sino que va a una posición expresada en grados, de 0 a 180° (medio giro). Dada la posición grados se debe generar un tren de pulsos que llevará al servo hasta la posición deseada y detenerse allí.

La imagen muestra lo anterior.



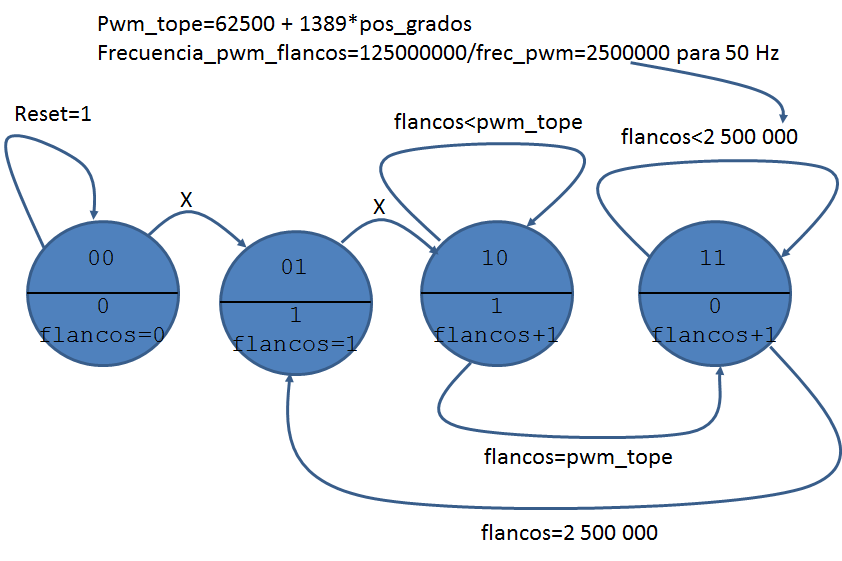
La frecuencia del PWM está entre 66,6 Hz y 40 Hz (de 15 ms a 25 ms), tomaremos el valor central de 50 Hz (20 ms). Este valor es libre y no afecta al funcionamiento dentro de estos límites.

Como la frecuencia del reloj interno es de 100 MHz, entonces cada periodo de PWM consiste de 2.000.000 flancos. Seguidamente se debe dejar a 0 o a 1 parte de esos 2.000.000 flancos según la posición que se quiera alcanzar (en grados) de 0,5 ms (para 0°) a 2,5 ms (para 180°). Lo anterior en flancos es de 50.000 flancos a 250.000 flancos. Es decir, son unos 1.111 flancos/grado (y eso se suma a los 50.000 flancos de “partida” para 0°).

Por tanto el autómata es muy parecido al de antes, solo que hay que reducir el número de estados y ajustar las frecuencias y los flancos.

En el autómata destacan dos elementos: que el autómata arranca siempre con 1 (cosa lógica viendo los dibujos) y que parece que hay un estado que sobra, el “001”. Este estado “001” realmente sobra, pero a mí no me gusta usar el estado de reset para recomenzar de forma “normal” la secuencia del servo: el estado de reset debe ser solo para reset (pero es una opción).

En el autómata siguiente se deben sustituir los valores 62.500 por 50000, el valor 2.500.000 por 2.000.000, y 1389 se sustituye por 1111.



Variables:

* La frecuencia de pwm es de 50 Hz. Este valor es el típico para un servo (debe estar entre 66 y 40 Hz), al igual que lo es 200 Hz para el pwm de continua.
* Para 50 Hz, 2 000 000 flancos dan lugar a un periodo de 20 ms, es decir, cada 2 000 000 flancos se cumple un ciclo de pwm para servo.
* Cada 1 ms son 100 000 flancos, y por tanto 0,5 ms son 50 000 flancos.
* pwm\_tope: como el servo gira un máximo de 180° y por cada grado se necesitan unos 1389 flancos a 1 (como cada 45° son 0,5 ms, entonces 50000/45= 1111,1 flancos), y teniendo en cuenta que hay 0,5 ms siempre a 1 entonces pwm\_tope = 50000 + 1111\*pos\_grados. Para 0°, pwm\_tope=50000 flancos (mínimo) y para 180°, pwm\_tope=250000 flancos (máximo).

Para seleccionar la posición del servo en grados se usan los 4 interruptores (luego este dato vendrá del ARM). CUIDADO de 1101 a 1111 el valor asignado es 0 °. La tabla queda:

|  |  |  |
| --- | --- | --- |
| Servo en grados | SW3\_SW0 | ms a 1 |
| 0 ° | 0000 | 0,5 ms |
| 15 ° | 0001 |  |
| 30 ° | 0010 |  |
| 45 ° | 0011 | 1 ms |
| 60 ° | 0100 |  |
| 75 ° | 0101 |  |
| 90 ° | 0110 | 1,5 ms |
| 105 ° | 0111 |  |
| 120 ° | 1000 |  |
| 135 ° | 1001 | 2 ms |
| 150 ° | 1010 |  |
| 165 ° | 1011 |  |
| 180 ° | 1100 | 2,5 ms |

El sistema implementado da los siguientes resultados: (ESTAS IMÁGENES son de ZYBO)

|  |  |  |
| --- | --- | --- |
| 0°, 0000 y 0,5 ms | 45°, 0011 y 1 ms | 90°, 0110 y 1,5 ms |
|  |  |  |
| 135°, 1001 y 2 ms | 180°, 1100 y 2,5 ms |  |
|  |  |  |

*Código:*

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity pwm\_servo is

port(

clk: in std\_logic;

btn: in std\_logic;

sw: in std\_logic\_vector (3 downto 0);

pwm\_oscilo: out std\_logic;

led: out std\_logic\_vector (3 downto 0)

);

end pwm\_servo;

architecture Behavioral of pwm\_servo is

signal reset: std\_logic;

signal pwm: std\_logic;

signal selector: std\_logic\_vector (3 downto 0);

signal pos\_grados: integer range 0 to 180;

signal contador\_base: integer range 0 to 2000000;

signal pwm\_tope: integer range 0 to 10000000;

signal estado: std\_logic\_vector (2 downto 0);

signal frecuencia\_pwm: integer range 0 to 1000;

signal frecuencia\_pwm\_flancos: integer range 0 to 100000000;

begin

reset<=btn;

led(0)<=pwm;

selector<=sw;

led(2 downto 1)<=estado(1 downto 0);

pwm\_oscilo<=pwm; --para ver la señal en un oscilo

frecuencia\_pwm<=50; --este valor controla la frecuencia del pwm del servo entre 40 y 66,6 Hz

--la posición en grados va de 0 a 180 grados de la posición final del servo, no gira mas

process(selector)

begin

case selector is

when "0000" => pos\_grados<=0;

when "0001" => pos\_grados<=15;

when "0010" => pos\_grados<=30;

when "0011" => pos\_grados<=45;

when "0100" => pos\_grados<=60;

when "0101" => pos\_grados<=75;

when "0110" => pos\_grados<=90;

when "0111" => pos\_grados<=105;

when "1000" => pos\_grados<=120;

when "1001" => pos\_grados<=135;

when "1010" => pos\_grados<=150;

when "1011" => pos\_grados<=165;

when "1100" => pos\_grados<=180;

when others => pos\_grados<=0;

end case;

end process;

-- frecuencia\_pwm\_flancos indica cuántos flancos del reloj del sistema hacen

-- falta para generar el periodo de PWM contando flancos flancos.

-- Por ejemplo para 50 Hz debe contar 50000 flancos

process(frecuencia\_pwm)

begin

frecuencia\_pwm\_flancos<=100000000/frecuencia\_pwm;

end process;

-- pwm\_tope expresa en flancos cuántos flancos debe estar la salida a 1 para alcanzar la posición.

-- hay que tener en cuenta que la posicion 0 grados debe generar 0,5 ms, es decir, 62500 (es como un offset)

process(pos\_grados)

begin

pwm\_tope<=(50000+1111\*pos\_grados);

end process;

process(pwm\_tope)

begin

if pwm\_tope<312520 and pwm\_tope>300000 then

led(3)<='1';

else

led(3)<='0';

end if;

end process;

process(clk, reset)

begin

if clk='1' and clk'event then

case estado is

--estado reset

when "000" => contador\_base<=0;

if reset='1' then

estado<="000";

else

estado<="001";

end if;

when "001" => contador\_base<=1;

if reset='1' then

estado<="000";

else

estado<="010";

end if;

when "010" => contador\_base<=contador\_base+1;

if reset='1' then

estado<="000";

else

if contador\_base<pwm\_tope then

estado<="010";

else

estado<="011";

end if;

end if;

when "011" => contador\_base<=contador\_base+1;

if reset='1' then

estado<="000";

else

if contador\_base<frecuencia\_pwm\_flancos then --2 500 000 = 125000000/50

estado<="011";

else

estado<="001";

end if;

end if;

when others => estado<="000";

contador\_base<=0;

end case;

end if;

end process;

process(estado)

begin

case estado is

when "000" => pwm<='0';

when "001" => pwm<='1';

when "010" => pwm<='1';

when "011" => pwm<='0';

when others => pwm<='0';

end case;

end process;

end Behavioral;

*Constraints:*

## Clock signal

set\_property PACKAGE\_PIN W5 [get\_ports clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports clk]

create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports clk]

## Switches

set\_property PACKAGE\_PIN V17 [get\_ports {sw[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[0]}]

set\_property PACKAGE\_PIN V16 [get\_ports {sw[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[1]}]

set\_property PACKAGE\_PIN W16 [get\_ports {sw[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[2]}]

set\_property PACKAGE\_PIN W17 [get\_ports {sw[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {sw[3]}]

## LEDs

#set\_property PACKAGE\_PIN U16 [get\_ports {led[0]}]

# set\_property IOSTANDARD LVCMOS33 [get\_ports {led[0]}]

#set\_property PACKAGE\_PIN E19 [get\_ports {led[1]}]

# set\_property IOSTANDARD LVCMOS33 [get\_ports {led[1]}]

#set\_property PACKAGE\_PIN U19 [get\_ports {led[2]}]

# set\_property IOSTANDARD LVCMOS33 [get\_ports {led[2]}]

#set\_property PACKAGE\_PIN V19 [get\_ports {led[3]}]

# set\_property IOSTANDARD LVCMOS33 [get\_ports {led[3]}]

## BUTTONs

set\_property PACKAGE\_PIN U18 [get\_ports btn]

set\_property IOSTANDARD LVCMOS33 [get\_ports btn]

##Pmod Header JA

##Sch name = JA1

set\_property PACKAGE\_PIN J1 [get\_ports {pwm\_oscilo}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {pwm\_oscilo}]

##Sch name = JA2

# set\_property PACKAGE\_PIN L2 [get\_ports {sentido\_motor\_DC}]

# set\_property IOSTANDARD LVCMOS33 [get\_ports {sentido\_motor\_DC}]

MONTAJE

Como es habitual y solo por confundir cada fabricante pone los colores de una manera para GND, Vcc y control. La imagen ayuda.



