INVERSORES DE GIRO PARA MOTORES A PASOS EN DISPOSITIVOS DE LÓGICA PROGRAMABLE

Juan Carlos Herrera Lozada, Juan Carlos González Robles, Agustín Cruz Contreras
Centro de Innovación y Desarrollo Tecnológico en Cómputo (CIDETEC – IPN)
E-mails: {jlozada; jgrobles; acruz}@ipn.mx

El control de motores a pasos (Unipolares o Bipolares) tiene una gran aplicación práctica. Básicamente se busca invertir el sentido del giro en cualquier momento sin caer en condiciones críticas.

Es importante reconocer las diferentes alternativas de implementación; por lo general, se resuelve utilizando algún microcontrolador u otro driver monolítico con funcionamiento específico.
En este trabajo se consideran dispositivos de lógica programable (PLDs), siendo alternativas de bajo costo (como en el caso de un GAL), con magnífico desempeño y velocidad, reconfigurables y adaptables a aplicaciones simples o complejas (utilizando FPGAs o CPLDs).

1. TEORÍA DE MOTORES A PASOS

Un motor a pasos (*stepper motor*) tiene la propiedad de moverse de un paso a otro, por cada pulso de reloj que se le aplique. Así, puede realizar 15 pasos en un mismo sentido si se le aplican 15 pulsos de reloj. Dependiendo de las características del motor, es posible tener pasos muy pequeños (por ejemplo de 1.8º, por lo que después de 200 pulsos completará una vuelta a razón de 1.8º x 200 = 360º) o pasos muy grandes (por ejemplo de 90º, completando una vuelta con 4 pulsos a razón de 90º x 4 = 360º).

Un motor de este tipo es muy socorrido en diseños que requiere posicionar con exactitud el rotor. Lo anterior resulta difícil en los motores de CD, debido a que estos giran libremente al aplicar un voltaje y si se desea detener el rotor es necesario retirar el voltaje de alimentación, lo cual no garantiza que el rotor se detendrá en una posición predefinida. De cualquier modo, existen técnicas que controlan la duración de pulsos de reloj que pueden hacer que un motor a CD se posicione de manera exacta, como sucede con los Servo Motores. En cuanto a su funcionamiento y configuración, los motores a pasos se clasifican en tres tipos: Unipolares, Bipolares y Multifase. Nos enfocamos sólo al motor unipolar, cuyo control es el más simple.

Un motor a pasos consta de dos partes principales: el rotor y el estator (**Figura1**). El rotor es la parte central del motor conformada por un imán permanente que gira debido a que el estator tiene bobinas que cuando se excitan adecuadamente generan un campo electromagnético que produce el movimiento del imán en alguna dirección. Lo anterior indica que para que el motor dé un paso, basta excitar la o las bobinas correspondientes.

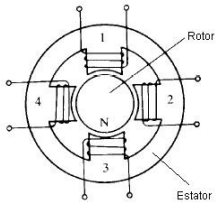

Figura 1. Motor a Pasos de 4 Fases.

El motor unipolar que se utilizó en este trabajo[1] es de 4 fases, o bien, 2 bobinas individuales donde cada una de ellas está separada por un tap central (punto común); de ahí que se consideren 4 bobinas en vez de dos (**Figura 2**). Cada bobina tiene asociados dos cables para la alimentación tanto positiva como negativa; sin embargo, el tap central es común a todas, por lo que físicamente sólo vemos 5 cables: el común y los cables individuales para cerrar la alimentación de las 4 bobinas.

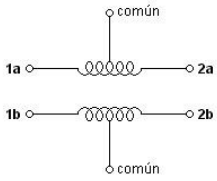

Figura 2. Distribución de las Bobinas de un Motor Unipolar de 4 Fases.

Es importante notar que si se decide conectar el común a GND, se excitará las bobinas con Vm

[1] *Fabricado por Howard Ind., distribuido por Jameco con número de parte 105890.*

(voltaje nominal del motor). Si por el contrario, se conecta el común a Vm, se necesitará excitarlas con GND. Todo se resume a polarizar correctamente cada bobina. Consideremos el caso en que se conecta el común a GND y las bobinas se excitan con Vm. De acuerdo a las características eléctricas del motor (12 V, 150 mA, 75 Ω en cada bobina), con el cable negro como **común**, el cable blanco y el verde conformando **1a** y**2a** respectivamente, así como el cable rojo y el cable café como **1b** y **2b**, es posible sugerir la **Figura 3**:

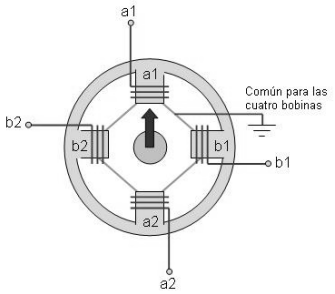

Figura 3. Motor a Pasos de 4 Fases, con Conexiones Propuestas.

En la figura anterior se interpreta que para *a1=1, b1=0, a2=0* y *b2=0*, el rotor apuntará hacia la bobina **a1**, debido a que es la única que está excitada. Si se desea que el rotor aparezca en la posición marcada por la **Figura 4**, es necesario aplicar *a1=0, b1=0, a2=1* y *b2=0*.

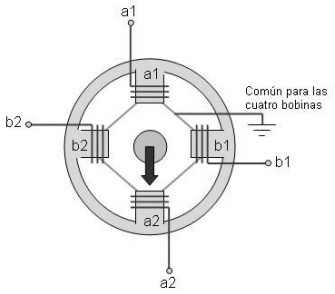

Figura 4. Rotor colocado hacia el sur.

La secuencia debe respetar obligatoriamente un orden; se sobreentiende que el rotor no puede pasar instantáneamente de la posición de la **figura 3** a la posición marcada por la **figura 4** sin haber recorrido las posiciones previas, no importando el sentido del giro. Opcionalmente, el rotor puede adquirir una posición intermedia, si se excitan dos bobinas coincidentes (cercanas) al mismo tiempo (**Figura 5**). Lo anterior implica un mayor torque, al igual que una mayor demanda de corriente. El estado aplicado es *a1=0, b1=1, a2=1* y *b2=0*.

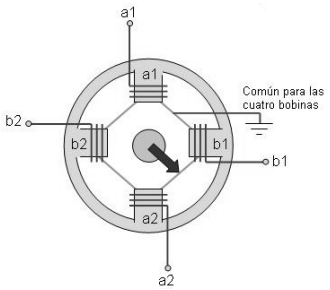

Figura 5. Excitación Simultánea de Dos Bobinas.

Se tienen tres posible secuencias a seguir: *Paso Completo Wave Drive, Paso Completo Normal* y *Medio Paso*. Para la primera secuencia de movimiento se plantea la siguiente tabla, excitando de manera individual cada bobina.

Tabla 1. Wave Drive.

Estado	a1	b1	a2	b2
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1

Se observa que la secuencia implica cuatro estados, donde cada estado entra con un pulso de reloj. El conteo será ascendente para un giro en el sentido de las manecillas del reloj y será descendente en sentido antihorario. La variable que permita cambiar el sentido del giro se declara como parte de una maquina de estados de manera muy similar a un contador UP/DOWN.

Para la segunda secuencia se respetan las condiciones descritas en la **Tabla 2**, excitando dos bobinas a la vez.

Tabla 2. Normal.

Estado	a1	b1	A2	b2
0	1	1	0	0
1	0	1	1	0
2	0	0	1	1
3	1	0	0	1

La tercera secuencia, permite dividir los pasos a la mitad e implica los pasos descritos en la **Tabla 3**.

Tabla 3. Medio Paso.

Estado	a1	b1	a2	b2
0	1	0	0	0
1	1	1	0	0
2	0	1	0	0
3	0	1	1	0
4	0	0	1	0
5	0	0	1	1
6	0	0	0	1
7	1	0	0	1

Obsérvese que se trata de una simple máquina de 8 estados, donde cada estado representa un paso.

2. DISEÑO DEL INVERSOR DE GIRO EN UN GAL

Un código aproximado en ABEL HDL, dirigido a un GAL16V8, para resolver la secuencia de *medio paso* se lista a continuación. En la máquina de estados se ha incluido un estado inicial para obligar el inicio del conteo (E0 = 0, 0, 0, 0), por lo que los estados reales de la secuencia se desplazan una posición y después del estado final E8 no se regresa a E0, sino a E1. La variable `dir`, permite cambiar el giro del motor. La variable `stop` tiene lógica negativa y permite detener el motor manteniendo el estado presente de la máquina de estados, lo cual es recomendable para garantizar una secuencia de reestablecimiento lógica. El código se sintetizó con ayuda del software de desarrollo *isp Lever* de *Lattice Semiconductors*.

```
MODULE motor_pasos
"Secuencia de Medio Paso
reloj,dir,stop pin 1,2,3;
"salidas registradas
a1, b1, a2, b2 pin 14,15,16,17 istype 'reg, dc';

"declaración de set
sreg=[a1,b1,a2,b2];
E0=[0,0,0,0];
E1=[1,0,0,0];
E2=[1,1,0,0];
E3=[0,1,0,0];
E4=[0,1,1,0];
E5=[0,0,1,0];
E6=[0,0,1,1];
E7=[0,0,0,1];
E8=[1,0,0,1];

Equations
sreg.clk=reloj;
state_diagram sreg

state E0:
IF dir THEN E1 ELSE E8;

state E1:
IF !stop THEN E1
ELSE IF dir THEN E2 ELSE E0;

state E2:
IF !stop THEN E2
ELSE IF dir THEN E3 ELSE E1;

state E3:
IF !stop THEN E3
ELSE IF dir THEN E4 ELSE E2;

state E4:
IF !stop THEN E4
ELSE IF dir THEN E5 ELSE E3;

state E5:
IF !stop THEN E5
ELSE IF dir THEN E6 ELSE E4;

state E6:
IF !stop THEN E6
ELSE IF dir THEN E7 ELSE E5;

state E7:
IF !stop THEN E7
ELSE IF dir THEN E8 ELSE E6;

state E8:
IF !stop THEN E8
ELSE IF dir THEN E1 ELSE E7;

END
```

El motor necesita un voltaje de 12 Volts y una corriente de 150 mA por bobina. El GAL no entrega estos valores nominales por lo que es necesario agregar una etapa de potencia a su salida, para amplificar la corriente. Así mismo, se debe considerar un voltaje de 5 Volts para el GAL y uno de 12 Volts para el motor. La etapa de potencia (driver) se puede implementar con un integrado monolítico ULN2003 que entrega hasta 500mA y presenta la característica de ser un driver con trabajo invertido, es decir, complementará los datos entrantes. En realidad no existe problema alguno, porque sólo basta con conectar el común a los 12 Volts nominales y conservar la misma tabla de secuencias establecida, ya que un "1" al entrar al driver se convertirá en un "0".

El circuito que se construyó fue el siguiente. El diodo zener es de 12 Volts.

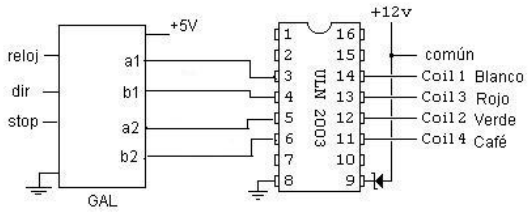

Figura 6. Circuito Controlador en un GAL16V8.

Las **Figuras 7 y 8**, muestran la implementación física. Los Leds se utilizaron para verificar el cambio en las salidas. Para el voltaje nominal del motor (12 Volts) se utilizó un regulador 7812 y para la alimentación del GAL (5 Volts) se utilizó el voltaje de la tarjeta de Desarrollo del FPGA, la cual se comentará posteriormente en este mismo documento.

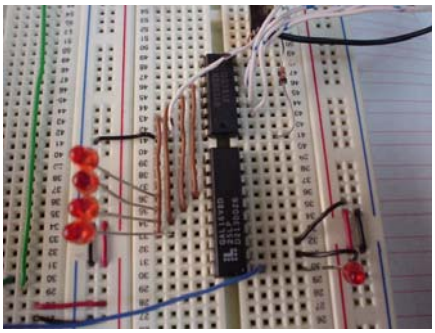

Figura 7. Controlador en un GAL16V8.

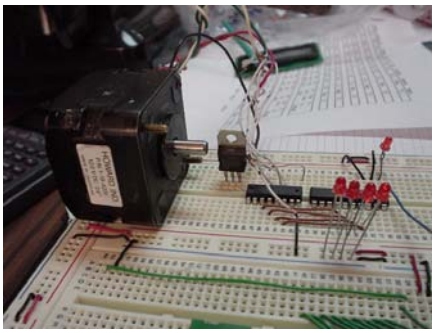

Figura 8. Circuito Controlador con Motor, vista frontal.

3. DISEÑO DEL INVERSOR DE GIRO EN UN FPGA

La arquitectura FPGA tiene diferencias con la propia de los CPLDs; sin embargo, el diseño y la implementación son similares. Recordando que las herramientas de síntesis lógicas estándar soportan descripciones en ABEL, VHDL y Verilog, se realizó una descripción adicional del mismo controlador en VHDL. La intención es proporcionar una idea simple de cómo es posible definir una macro reutilizable para controlar varios motores a la vez, gracias al gran número de pines y recursos lógicos dentro de un dispositivo de este tipo.

El código que resuelve el control a medio paso, se muestra a continuación. La descripción se realizó en VHDL.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

--Secuencia de Medio Paso
entity motor_pasos is
port (
RELOJ, RESET, STOP, DIR: in STD_LOGIC;
DATO_MOTOR: out STD_LOGIC_VECTOR(3 downto 0)
);
end motor_pasos;

architecture motor_arch of motor_pasos is
type state_type is (INICIA, CERO, UNO, DOS, TRES,
CUATRO, CINCO, SEIS, SIETE);
signal estado, estado_siguiente: state_type;
begin
arranque_motor:process (RELOJ, RESET)
begin
if RESET='0' then
estado <= INICIA;
elsif RELOJ='1' and RELOJ'event then
estado <= estado_siguiente;
end if;
end process arranque_motor;

estados_motor:process (estado, DIR, STOP)
begin
case estado IS
when INICIA => if STOP='0' then
estado_siguiente <= INICIA;
elsif DIR='1' then
estado_siguiente <= CERO;
else estado_siguiente <= SIETE;
end if;

when CERO => if STOP='0' then
estado_siguiente <= CERO;
elsif DIR='1' then
estado_siguiente <= UNO;
else estado_siguiente <= SIETE;
end if;

when UNO => if STOP='0' then
estado_siguiente <= UNO;
elsif DIR='1' then
estado_siguiente <= DOS;
else estado_siguiente <= CERO;
end if;

when DOS => if STOP='0' then
estado_siguiente <= DOS;
elsif DIR='1' then
estado_siguiente <= TRES;
else estado_siguiente <= UNO;
end if;

when TRES => if STOP='0' then
estado_siguiente <= TRES;
elsif DIR='1' then
estado_siguiente <= CUATRO;
else estado_siguiente <= DOS;
end if;

when CUATRO => if STOP='0' then
estado_siguiente <= CUATRO;
elsif DIR='1' then
estado_siguiente <= CINCO;
else estado_siguiente <= TRES;
end if;

when CINCO => if STOP='0' then
estado_siguiente <= CINCO;
elsif DIR='1' then
estado_siguiente <= SEIS;
else estado_siguiente <= CUATRO;
end if;

when SEIS => if STOP='0' then
estado_siguiente <= SEIS;
elsif DIR='1' then
estado_siguiente <= SIETE;
else estado_siguiente <= CINCO;
end if;

when SIETE => if STOP='0' then
estado_siguiente <= SIETE;
elsif DIR='1' then
estado_siguiente <= CERO;
else estado_siguiente <= SEIS;
end if;
```

```
END CASE;
end process estados_motor;

salida:process(estado)
begin
case estado IS
when INICIA => DATO_MOTOR <= "0000";
when CERO   => DATO_MOTOR <= "1000";
when UNO    => DATO_MOTOR <= "1100";
when DOS    => DATO_MOTOR <= "0100";
when TRES   => DATO_MOTOR <= "0110";
when CUATRO => DATO_MOTOR <= "0010";
when CINCO  => DATO_MOTOR <= "0011";
when SEIS   => DATO_MOTOR <= "0001";
when SIETE  => DATO_MOTOR <= "1001";
when others => NULL;
END CASE;
end process salida;

end motor_arch;
```

La implementación del controlador se realizó sobre un FPGA XC4010XL de Xilinx (tarjeta XESS) sintetizando el código con el ambiente de desarrollo *Foundation,* obteniendo resultados satisfactorios.

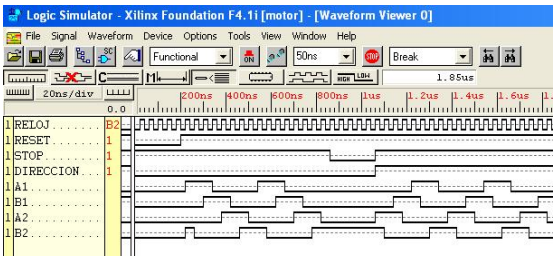

Figura 9. Resultados de Simulación Para el Controlador, en VHDL.

La frecuencia de trabajo es de hasta 6MHz, aunque para elementos de respuesta mecánica son suficientes 10 KHz.

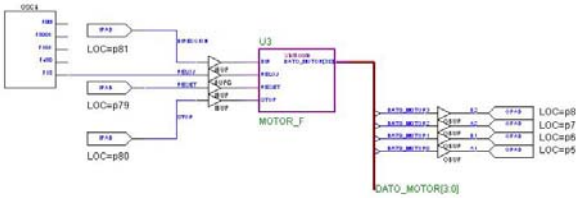

Figura 10. Diagrama Esquemático del Diseño en Foundation.

De manera similar al circuito del GAL, fue necesario el ULN2003 para compensar la demanda de corriente en las bobinas del motor. La **Figura 11**, muestra la tarjeta de desarrollo utilizada para validar los resultados.

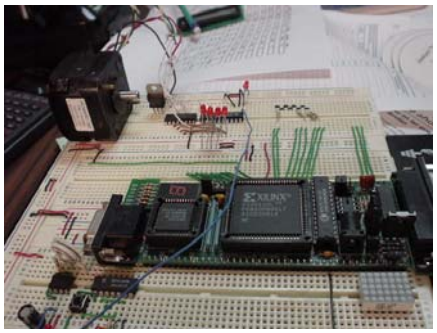

Figura 11. Controlador en FPGA.

4. CONCLUSIONES

El inversor de giro implementado en el GAL16V8 resultó muy económico en comparación a los drivers monolíticos existentes. La lógica descrita es muy simple, por lo que el diseñador puede optar por el HDL de su preferencia sin cambios drásticos en el código. Es posible realizar un controlador para dos motores en un GAL22V10, manteniendo la misma filosofía de diseño.

En código se hace extensivo para dispositivos de arquitectura avanzada, como FPGAs y CPLDs. Tratando los controladores como macros dentro de un esquemático se facilita la implementación de sistemas completos para controlar varios motores de forma paralela. Por tratarse de estándares, los códigos en ABEL y VHDL, pueden sintetizarse sobre dispositivos de otros fabricantes (Actel, Altera, Lattice), vía software de propietario.

Los trabajos planteados a futuro implican una comparación detallada con la alternativa de microcontroladores. La iniciativa presentada por los dispositivos de lógica programable por sí misma deriva en una mayor versatilidad, aunado al diseño contemporáneo de la electrónica digital para actualizar los programas de estudio de la Ingeniería Electrónica y áreas afines.

5. BIBLIOGRAFÍA

[1] Sebastian Michael J., Application - Specific Integrated Circuits, Addison Wesley, 2000.

[2] Synopsis, FPGA Express with Verilog HDL and VHDL, Reference Manual, Xilinx in line, 2002.

[3] John Wakerly, Digital Design, Prentice Hall, 2002.