ROBOTICA

Análisis Dinámico de un Robot e implementación en FPGA

Año 2012

Integrantes:

Alvarez Nahuel (109712-0)

Kowalczuk Ivan (115576-3)

Introducción general

Realizaremos en el presente documento el análisis y desarrollo de la dinámica de un robot manipulador basado en una configuración propietaria.

El objetivo del análisis dinámico es entender la relación existente entre el movimiento del robot y las fuerzas que actúan sobre el.

Mediante este análisis se determina la relación entre la localización de cada una de sus partes constitutivas (eslabones) o extremos y por lo tanto sus velocidades y aceleraciones instante a instante, también permite conocer las fuerzas y pares que se aplican ante determinados movimientos. Para conocer con precisión estos valores se deben tomar en cuenta las dimensiones, longitudes, masas, inercias, materiales y demás elementos físicos del robot.

Para poder realizar este análisis, al momento de obtener el modelo dinámico se implemento el método de Lagrange-Euler basado sobre la representación obtenida por Denavit-Hartenberg en el trabajo práctico precedente. Si bien este algoritmo es un procedimiento computacional ineficiente, ya que las operaciones que debe realizar una PC es del orden n^4 donde 'n' es el número de grados de libertad del robot, permite obtener las ecuaciones completas de los movimientos de fuerzas, torques y demás magnitudes físicas. En la simulación de MATLAB que se vera mas adelante quedan identificados estos pasos para realizar un rápido seguimiento del algoritmo implementado.

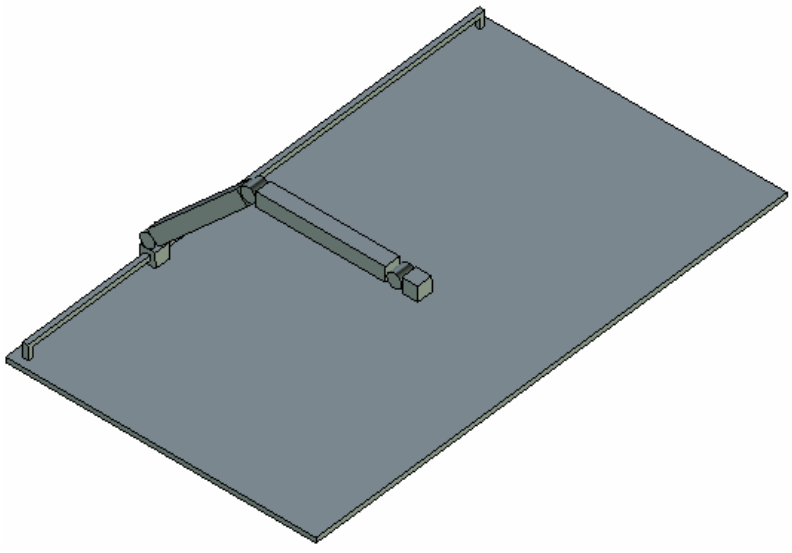


Figura 1. Modelo 3D del manipulador.

Es importante entender que para la obtención de un modelo dinámico completo de un robot, no solo hay que tener en cuenta los eslabones sino que también debe incluirse

los sistemas de transmisión, actuadores y equipos electrónicos de mando que incluyen rozamientos, perturbaciones eléctricas, etc. Dado el alcance de este trabajo practico, estas consideraciones quedan fuera ya que solo realizaremos una modelización y simulación simplificada que considera a los mencionados elementos como ideales.

En la figura 1 se puede apreciar el manipulador del trabajo práctico precedente. Este dispositivo cuenta con 4 grados de libertad de los cuales se puede mover en los tres ejes cartesianos (X; Y; Z) y orientarse en un solo ángulo de giro (Alfa).

En la figura 3, presentada a continuación, se detalla el código MATLAB obtenido aplicando el método de Lagrange-Euler. En dicho código podemos identificar las variables más significativas:

- M_nAm: Matriz de transformación homogénea del sistema de coordenadas “m” al sistema de coordenadas “n”.
- q: Vector que contiene la posición de las articulaciones del manipulador.
- R: Matriz cuyas columnas se encuentran conformadas por los vectores de los centros de masas de cada eslabón.
- M: Matriz de una sola fila cuyas columnas representan la masa de cada eslabón.
- G: Matriz cuyas filas se encuentran conformadas por los vectores que definen como afecta la gravedad al sistema coordenado de cada eslabón.

A partir de estos valores se obtienen los arreglos D, h y c que permiten la obtención de ecuaciones de troqué de cada articulación.

```
%% DEFINO LAS VARIABLES VARIABLES DEL ROBOT
clear all;
close all;
syms d1 t3 t4 t5;
a2 = 10;
a3 = 56;
a4 = 56;
a5 = 10;
%% DEFINO LAS MATRICES HOMOGENEAS EN FUNCIÓN DE LAS VARIABLES
% Debo considerar que las transformaciónes se logran como sigue:
% (i-1)A(i) = T(z,tita)T(0,0,di)T(ai,0,0)T(x,alfai);
% DEFINO LA MATRIZ 0A1 - Articulación 1
M_0A1 = ...
        [ 0           1          0          0 ...
        ;-1           0          0          0 ...
        ; 0           0          1          0 ...
        ; 0           0          0          1 ] ...
    * ...
        [ 1           0          0          0   ...
        ; 0           1          0          0   ...
        ; 0           0          1          d1  ...
```

```
        ; 0           0          0          1 ] ;
% DEFINO LA MATRIZ 1A2 - Articulación 2
M_1A2 = ...
        [ 1           0          0          a2  ...
        ; 0           1          0          0   ...
        ; 0           0          1          0   ...
        ; 0           0          0          1 ] ;
% DEFINO LA MATRIZ 2A3 - Articulación 3
M_2A3 = ...
        [ cos(t3)    -sin(t3)    0          0 ...
        ; sin(t3)     cos(t3)    0          0 ...
        ; 0           0          1          0 ...
        ; 0           0          0          1 ] ...
    * ...
        [ 1           0          0          a3  ...
        ; 0           1          0          0   ...
        ; 0           0          1          0   ...
        ; 0           0          0          1 ] ;
% DEFINO LA MATRIZ 3A4 - Articulación 4
M_3A4 = ...
        [ cos(t4)    -sin(t4)    0          0 ...
        ; sin(t4)     cos(t4)    0          0 ...
        ; 0           0          1          0 ...
        ; 0           0          0          1 ] ...
    * ...
        [ 1           0          0          a4  ...
        ; 0           1          0          0   ...
        ; 0           0          1          0   ...
        ; 0           0          0          1 ] ;
% DEFINO LA MATRIZ 4A5 - Articulación 5
M_4A5 = ...
        [ cos(t5)    -sin(t5)    0          0 ...
        ; sin(t5)     cos(t5)    0          0 ...
        ; 0           0          1          0 ...
        ; 0           0          0          1 ] ...
    * ...
        [ 1           0          0          a5  ...
        ; 0           1          0          0   ...
        ; 0           0          1          0   ...
        ; 0           0          0          1 ] ;
% SE COMPONE LA MATRIZ DE TRANSFORMACIÓN HOMOGENEA T.
% Obtengo la matríz de transformación.
A = { M_0A1 ...
    ; M_0A1 * M_1A2 ...
    ; M_0A1 * M_1A2 * M_2A3 ...
    ; M_0A1 * M_1A2 * M_2A3 * M_3A4 ...
    ; M_0A1 * M_1A2 * M_2A3 * M_3A4 * M_4A5 ...
    };
%% APLICAMOS LAGRANGE-EULER
q = [d1; t3; t4; t5];
I = 4;
J = 4;
% Paso L-E 3.
```

```
for i=1:1:size(A,1)
    for j=1:1:size(q,1)
        Uij{i,j} = diff(A{i},q(j));
    end
end
% Paso L-E 4.
for i=1:1:size(Uij,1)
    for j=1:1:size(Uij,2)
        for k=1:1:size(q,1)
            Uijk{i,j,k} = diff(Uij{i,j}, q(k));
        end
    end
end
% Paso L-E 5.
% Defino los centros de masa de los eslabones
R = [   0           25.603      25.603      2.30388 ...
    ;   0           0           0           0       ...
    ;   0           0           0           0       ...
    ;   1           1           1           1       ...
    ];
% Defino las masas de los eslabones
M = [   0.0014976   0.0153544   0.0153544   0.00243763];
% Defino como afecta la gravedad a esos eslabones.
g = 9.8;
G = [   0   -g  0   0   ...
    ;   -g  0   0   0   ...
    ;   -g  0   0   0   ...
    ;   -g  0   0   0   ...
    ];
for i=1:1:size(R,2)
    xi = R(1,i);
    yi = R(2,i);
    zi = R(3,i);
    m = M(i);
    Ji{i} = [ xi^2*m     xi*yi*m     xi*zi*m     xi*m    ...
            ; yi*xi*m    yi^2*m      yi*zi*m     yi*m    ...
            ; zi*xi*m    zi*yi*m     zi^2*m      zi*m    ...
            ; xi*m       yi*m        zi*m        m       ...
            ];
end
% Paso L-E 6.
N = 4; % N grados de libertad
syms D;
for i=1:1:N
    for j=1:1:N
        D(i,j) = 0;
        for k=1:1:size(Ji,2)
            D(i,j) = trace(Uij{k,j} * Ji{k} * Uij{k,i}');
        end
    end
end
% Paso L-E 7.
syms h;
```

```
for i=1:1:N
    for k=1:1:N
        for m=1:1:N
            h(i,k,m) = 0;
            for j=max([i k m]):1:N
                h(i,k,m) = trace(Uijk{j,k,m} * Ji{j} * Uij{j,i}');
            end
        end
    end
end
% Paso L-E 8.
syms H;
syms va1 va3 va4 va5;
qv = [va1; va3; va4; va5];
for i=1:1:N
    H(i) = 0;
    for k=1:1:N
        for m=1:1:N
            H(i) = H(i) + h(i,k,m)*qv(k)*qv(m);
        end
    end
end
H = H';
% Paso L-E 9.
syms c;
for i=1:1:N
    c(i) = 0;
    for j=1:1:N
        c(i) = c(i) - M(j) * G(j) * Uij{j,i} * R(:,j);
    end
end
c = c';
% Paso L-E 10.
syms aa1 aa3 aa4 aa5;
TAU = D * [aa1; aa3; aa4; aa5] + H + c;
%% SIMULACION DEL MOVIMIENTO
t = 1:0.5:20;
t4 = 0;
va4 = 0;
aa4 = 0;
t5 = 0;
va5 = 0;
aa5 = 0;
for i = 1:1:size(t,2);
    d1 = t(i)^2 .* 10 + t(i) .* 0 + 0;
    va1 = 2 .* t(i) .* 10;
    aa1 = 2 .* 10 ;
    t3 = t(i)^2 .* pi/200 - t(i) .* 0 - pi/2;
    va3 = 2 * t(i) .* pi/200  ;
    aa3 = 2 .* pi/200;
    t4 = t(i)^2 .* pi/200 - t(i) .* 0 - pi/2;
    va4 = 2 * t(i) .* pi/200  ;
    aa4 = 2 .* pi/200;
```

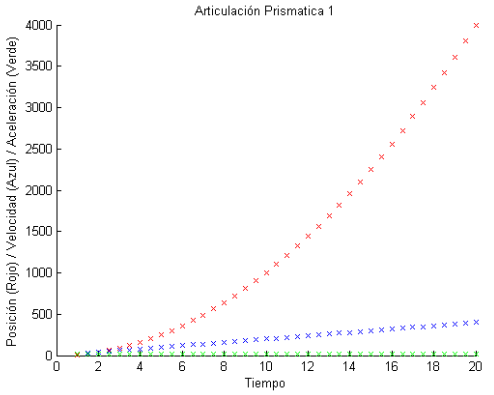
```
    t5 = t(i)^2 .* pi/200 - t(i) .* 0 - pi/2;
    va5 = 2 * t(i) .* pi/200  ;
    aa5 = 2 .* pi/200;
    F(1:N,i) = eval(TAU);
end
figure;
hold on;
plot(t, F(1,:),'x','Color','Red');
plot(t, F(2,:),'x','Color','Blue');
plot(t, F(3,:),'x','Color','Green');
plot(t, F(4,:),'x','Color','Black');
```

Figura 2. Código Fuente para MATLAB que realiza el método de Lagrange-Euler y representa gráficamente las aceleraciones y velocidades de los eslabones del robot diseñado.

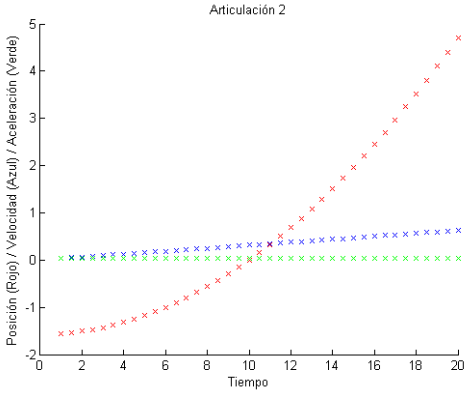
Una facilidad que provee el MATLAB al momento de trabajar con matrices donde cada elemento de la matriz era una nueva matriz son los ARRAYS referenciados por llaves. El uso de esta característica simplifico enormemente la primera versión del SCRIPT.

Una vez realizada esta actividad fue posible visualizar las distintas aceleraciones que pueden ser alcanzadas por los distintos eslabones de nuestro robot. Las mismas fueron comprobadas por el sentido común, validando así el modelo matemático hallado. Pueden observarse algunas de las simulaciones realizadas en la figura 3. Las gráficas corresponden a la aplicación de movimientos lineales y angulares uniformemente variados con aceleración constante. Esto establece una rampa de velocidad y un crecimiento cuadrático de la posición. Si prestamos atención la posición de la articulación prismática rápidamente supera los límites planteados para el carril por el cual se desplaza. De igual forma las articulaciones rotaciones en la simulación pueden superar los ángulos posibles para las articulaciones observando una variación de los torques en forma seudosenoidal debido a la orientación del brazo respecto de la gravedad. Se puede observar que incluso existen para la articulación 2 torques negativos que se justifican debido a que por momentos es la fuerza de gravedad la que hace el trabajo a favor del movimiento deseado.

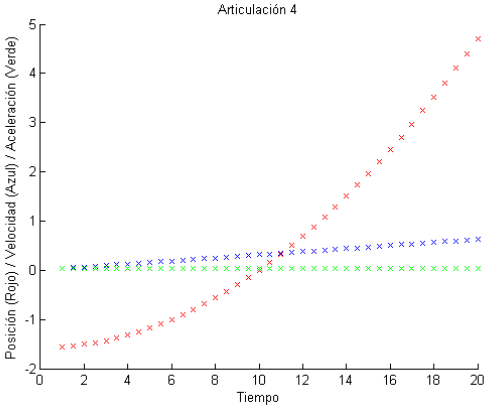
a



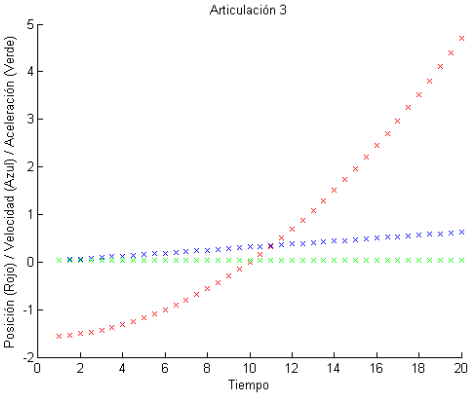
b



c



d



e

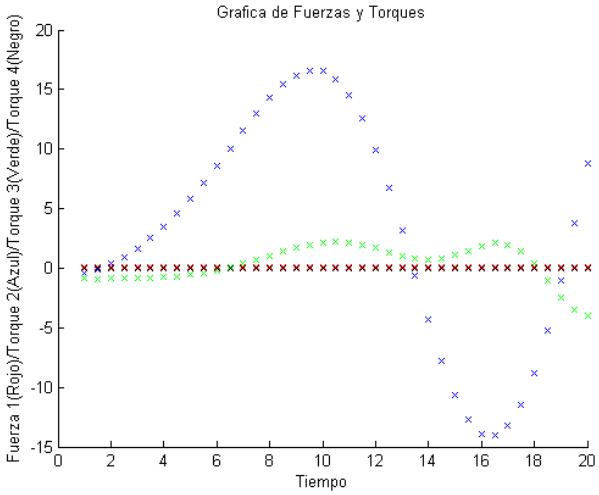


Figura 3. Simulaciones con el modelo obtenido. a) Posición, velocidad y aceleración lineal de la articulación prismática. b) Posición, velocidad y aceleración angular de la articulación 2. c) Posición, velocidad y aceleración angular de la articulación 3. d) Posición, velocidad y aceleración angular de la articulación 4. e) Fuerzas y torques en cada articulación.

Implementación del control de un PWM sobre un FPGA.

A continuación presentamos el esquema en bloques que se pretende implementar sobre el dispositivo FPGA. Utilizamos un código de colores para establecer las diferentes secciones del sistema:

- **VERDE:** Funciones a integrar realmente en un FPGA para implementar el control de un motor.
- **AZUL:** Etapa de potencia media ó alta, integrada en VHDL con el único objetivo de comprobar la resolución realizada.
- **AMARILLO:** Fuente de alimentación, integrada en VHDL con el único objetivo de comprobar la resolución realizada.

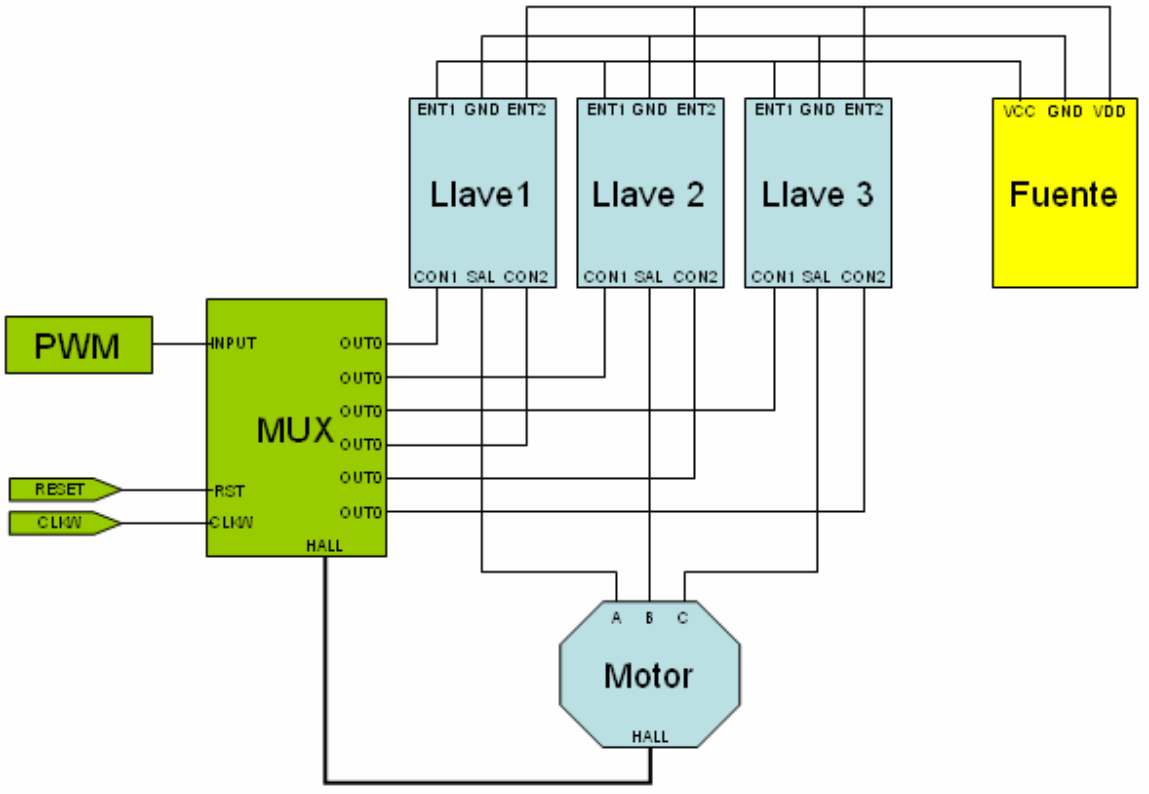


Figura 4. Esquema del sistema simulado con VHDL.

La tensión entregada por la fuente de alimentación y el ciclo de actividad de los pulsos entregados por el generador de PWM definen la velocidad de giro del motor. Al mismo tiempo el sentido de giro del motor se controla con la señal CLW (Clockwise – Sentido horario), cuando dicha señal se encuentra activa las fases se secuencian para obtener un giro del motor en sentido horario, en caso contrario las fases se secuencian para obtener

un sentido de giro antihorario. Cabe destacar que el sincronismo de las fases se logra a partir de las señales que entrega el sensor de efecto HALL que realimenta el sistema.

Presentamos a continuación la implementación de cada uno de los componentes.

Componente reloj.vhd

Este componente genera una señal de reloj con un ciclo de actividad del 50%. La frecuencia de la señal de reloj se control a partir de la constante PERIODO que en el ejemplo se encuentra establecida en 3.5ns. Este es un valor figurativo ya que en una implementación real la frecuencia de la señal de control ronda los 25Khz.

```
--------------------------------------------------------------------------------
--  File Name: reloj.vhd
--------------------------------------------------------------------------------
--
--  Generador de la señal de clock del sistema.
--
--------------------------------------------------------------------------------
LIBRARY ieee;

USE ieee.std_logic_1164.all;
--------------------------------------------------------------------------------
-- DECLARACION DE LA ENTIDAD
--------------------------------------------------------------------------------
ENTITY reloj IS
  GENERIC(PERIODO:TIME:=3.5 ns);
  PORT( CLK:OUT STD_LOGIC);
END reloj;
--------------------------------------------------------------------------------
-- DECLARACION DE LA ARQUITECTURA
--------------------------------------------------------------------------------
ARCHITECTURE arq_reloj OF reloj IS
  SHARED VARIABLE ENDSIM: boolean:=false;
BEGIN
  --------------------------------------------------------------------------------
  -- PROCESO PRINCIPAL
  --------------------------------------------------------------------------------
  main:PROCESS
  BEGIN
    IF ENDSIM = FALSE THEN
      CLK <= '1';
      WAIT FOR PERIODO/2;
```

```
      CLK <= '0';
      WAIT FOR PERIODO/2;
    ELSE
      WAIT;
    END IF;
  END PROCESS;
END arq_reloj;
```

Figura 5. Código VHDL del reloj.

Componente contador.vhd

Este componente permite definir un contador que se incremente ó decremente de uno en uno hasta alcanzar el límite superior ó el límite inferior de su acumulador, una vez hecho esto se activara la señal de fin de conteo y se recargará el contador con el valor de recarga definido.

Combinando el reloj con el contador se pueden obtener diferentes ciclos de actividad, para el ejemplo lo mantendremos alimentado directamente con el ciclo de actividad del 50% obviando los contadores.

```
--------------------------------------------------------------------------------
--  File Name: contador.vhd
--------------------------------------------------------------------------------
--
--  Contador que emite la señal de desbordamiento en función de la cantidad de
--  cuentas especificadas en la entrada.
--
--------------------------------------------------------------------------------
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all ;
USE    work.user_pkg.all;
--------------------------------------------------------------------------------
-- DECLARACION DE LA ENTIDAD
--------------------------------------------------------------------------------
ENTITY contador IS
PORT (
  CLK:IN STD_LOGIC;
  RST:IN STD_LOGIC;
  IDC:IN STD_LOGIC;
  RELOAD:IN STD_LOGIC_VECTOR(7 DOWNTO 0);
  TC:BUFFER STD_LOGIC
```

```
);
END contador;
--------------------------------------------------------------------------------
-- DECLARACION DE LA ARQUITECTURA
--------------------------------------------------------------------------------
ARCHITECTURE arq_contador OF contador IS
  SIGNAL REG:STD_LOGIC_VECTOR(7 DOWNTO 0);
  SIGNAL CNT:STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
  --------------------------------------------------------------------------------
  -- PROCESO VALOR DE RECARGA
  --------------------------------------------------------------------------------
  PROCESS(CLK,REG,RST)
  BEGIN
    IF (RST ='1') THEN
      REG <="00000000";
    ELSIF RISING_EDGE(CLK) THEN
      REG <= RELOAD;
    END IF;
  END PROCESS;
  --------------------------------------------------------------------------------
  -- PROCESO CONTADOR
  --------------------------------------------------------------------------------
  PROCESS (CLK,CNT,TC,REG)
  BEGIN
  IF (TC = '1') THEN
    CNT <= REG;
  ELSIF RISING_EDGE(CLK) THEN
    IF (TC = '0' and IDC ='1' and CNT <"11111111") THEN
      CNT <= INC(CNT);
    ELSE
      IF (TC ='0' and IDC ='0' and CNT > "00000000") THEN
        CNT <= DEC(CNT);
      END IF;
    END IF;
  END IF;
  END PROCESS;
  --------------------------------------------------------------------------------
  -- PROCESO FIN DE CUENTA
  --------------------------------------------------------------------------------
  PROCESS(CNT, TC, CLK,RST)
  BEGIN
    IF (RST ='1') THEN
      TC <='1';
    ELSIF RISING_EDGE(CLK) THEN
```

```
      IF ((CNT = "11111111") OR (CNT ="00000000")) THEN
        TC <= '1';
      ELSE
        TC <= '0';
      END IF;
    END IF;
    END PROCESS;
END arq_contador;
```

Figura 6. Código VHDL del contador.

Componente mux.vhd

Este componente se encargará de la sincronización de las fases de alimentación del motor conforme el sentido de giro solicitado y la lectura de posición entregada por el sensor de efecto de campo.

Cada una de sus salidas controlará un “transistor de media/alta potencia” que conectarán las diferentes fases del motor a la fuente de alimentación.

Este es componente clave para la aplicación buscada.

```
--------------------------------------------------------------------------------
--  File Name: mux.vhd
--------------------------------------------------------------------------------
--
--  Multiplexador basado en código GRAY de 1 entrada a 6 salidas.
--
--------------------------------------------------------------------------------
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all ;
--------------------------------------------------------------------------------
-- DECLARACION DE LA ENTIDAD
--------------------------------------------------------------------------------
ENTITY mux IS
PORT (
  INPUT:IN STD_LOGIC;
  CLKW:IN STD_LOGIC;
  HALL:IN STD_LOGIC_VECTOR(2 DOWNTO 0);
  OUTPUT:OUT STD_LOGIC_VECTOR(5 DOWNTO 0)
);
END mux;
--------------------------------------------------------------------------------
-- DECLARACION DE LA ARQUITECTURA
```

```
--------------------------------------------------------------------------------
ARCHITECTURE arq_mux OF mux IS
BEGIN
  --------------------------------------------------------------------------------
  -- PROCESO CONTADOR
  --------------------------------------------------------------------------------
  PROCESS (INPUT,CLKW,HALL)
         BEGIN
           OUTPUT <= "000000";
           CASE HALL IS
         WHEN "100" =>
           -- "001010";
           IF CLKW = '1' THEN
                    OUTPUT(1) <= INPUT;
              OUTPUT(3) <= INPUT;
             ELSE
              OUTPUT(0) <= INPUT;
              OUTPUT(4) <= INPUT;
             END IF;
           WHEN "101" =>
             -- "100010";
             IF CLKW = '1' THEN
              OUTPUT(1) <= INPUT;
              OUTPUT(5) <= INPUT;
             ELSE
              OUTPUT(2) <= INPUT;
              OUTPUT(4) <= INPUT;
             END IF;
           WHEN "001" =>
             -- "100001";
             IF CLKW = '1' THEN
              OUTPUT(0) <= INPUT;
              OUTPUT(5) <= INPUT;
             ELSE
              OUTPUT(2) <= INPUT;
              OUTPUT(3) <= INPUT;
             END IF;
           WHEN "011" =>
             -- "010001";
             IF CLKW = '1' THEN
              OUTPUT(0) <= INPUT;
              OUTPUT(4) <= INPUT;
             ELSE
              OUTPUT(1) <= INPUT;
              OUTPUT(3) <= INPUT;
```

```
              END IF;
            WHEN "010" =>
              -- "010100";
              IF CLKW = '1' THEN
              OUTPUT(2) <= INPUT;
                OUTPUT(4) <= INPUT;
              ELSE
                OUTPUT(1) <= INPUT;
                OUTPUT(5) <= INPUT;
              END IF;
            WHEN "110" =>
              -- "001100";
              IF CLKW = '1' THEN
           OUTPUT(2) <= INPUT;
              OUTPUT(3) <= INPUT;
              ELSE
                OUTPUT(0) <= INPUT;
                OUTPUT(5) <= INPUT;
              END IF;
            WHEN OTHERS =>
              OUTPUT <= "000000";
            END CASE;
          END PROCESS;
END arq_mux;
```

Figura 7. Código VHDL del multiplexor.

Componente llave.vhd

Para simular el puente H de potencia se definieron los componentes “Llave” que conectan según se lo indiquemos en sus entradas cada fase del motor a VCC, VDD ó tierra. Inicialmente se habían definido dos componentes separados para mantener la analogía de los dos transistores que administran cada fase en dicha etapa, pero obtuvimos un mensaje de error al compilar el código VHDL ya que cada fase tenía dos orígenes distintos.

```
--------------------------------------------------------------------------------
--  File Name: llave.vhd
--------------------------------------------------------------------------------
--
--  Simulador de las llaves (posibles transistores de potencia) utilizadas para
--  la etapa de potencia que controla las fases del motor.
--  ENT: Entrada.
```

```
--  SAL: Salida.
--  CON: Control.
--------------------------------------------------------------------------------
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all ;
--------------------------------------------------------------------------------
-- DECLARACION DE LA ENTIDAD
--------------------------------------------------------------------------------
ENTITY llave IS
  PORT (
    ENT1:IN REAL;
    ENT2:IN REAL;
    GND:IN REAL;
    CON1:IN STD_LOGIC;
    CON2:IN STD_LOGIC;
    SAL:OUT REAL
  );
END llave;
--------------------------------------------------------------------------------
-- DECLARACION DE LA ARQUITECTURA
--------------------------------------------------------------------------------
ARCHITECTURE arq_llave OF llave IS
BEGIN
    PROCESS
    BEGIN
      WAIT;
    END PROCESS;
    PROCESS(CON1,CON2,ENT1,ENT2)
    BEGIN
      IF (CON1 = '1') THEN
        SAL <= ENT1;
      ELSIF (CON2 = '1') THEN
        SAL <= ENT2;
      ELSE
        SAL <= GND;
      END IF;
    END PROCESS;
END arq_llave;
```

Figura 8. Código VHDL de la llave.

Componente motor.vhd

Este es el componente al que le dedicamos mayor esfuerzo, no porque fuera un componente clave del desarrollo sobre FPGA, ya que de hecho este será un componente real que se conecte al sistema embebido en el FPGA, sino porque buscamos modelar con cierta veracidad el funcionamiento del motor brushless para comprobar la efectividad de la electrónica de control definida con el dispositivo FPGA.

El componente que presentamos computa la alimentación de las tres fases para establecer la proyección de las componentes magnéticas sobre el eje del motor y computar el movimiento que debe realizar el mismo. Una de las variables internas de este componente es la posición angular del eje que variará más ó menos lento con la tensión de alimentación y el ciclo de actividad de la señal PWM entregada por el MUX a través de las llaves antes mencionadas.

```
--------------------------------------------------------------------------------
--  File Name: motor.vhd
--------------------------------------------------------------------------------
--
--  Simulador de motor con sensor por efecto hall.
--  FASEA, FASEB, FASEC: Las tres fases del motor.
--  HALL: Lectura que realiza el sensor por efecto hall a cada momento.
--------------------------------------------------------------------------------
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.math_real.all;
USE ieee.math_complex.all;
--------------------------------------------------------------------------------
-- DECLARACION DE LA ENTIDAD
--------------------------------------------------------------------------------
ENTITY motor IS
  GENERIC (
    ARG_FA:REAL := 1.57;
    ARG_FC:REAL := 3.66;
    ARG_FB:REAL := 5.75;
    muestreo:TIME := 1 ns;
    lsup0:REAL := 0.0;
    lsup1:REAL := 1.05;
    lsup2:REAL := 2.09;
    lsup3:REAL := 3.14;
    lsup4:REAL := 4.19;
    lsup5:REAL := 5.24;
    lsup6:REAL := 6.28
  );
  PORT (
```

```
    FASEA:IN REAL;
    FASEB:IN REAL;
    FASEC:IN REAL;
    HALL:BUFFER STD_LOGIC_VECTOR (2 DOWNTO 0)
  );
END motor;
--------------------------------------------------------------------------------
-- DECLARACION DE LA ARQUITECTURA
--------------------------------------------------------------------------------
ARCHITECTURE arq_motor OF motor IS
  TYPE ESTADO IS (I,A,B,C,D,E,F);
  SIGNAL s_estado:ESTADO;
  SIGNAL ARGANT:REAL;
  SIGNAL PROY:REAL; -- Magnitud proyectada sobre la ortogonal a la orientación del campo.
  SIGNAL ALFA_A:REAL; -- Angulo existente entre el polo del eje y el polo de signo opuesto
en el estator.
  SIGNAL ALFA_B:REAL; -- Angulo existente entre el polo del eje y el polo de signo opuesto
en el estator.
  SIGNAL ALFA_C:REAL; -- Angulo existente entre el polo del eje y el polo de signo opuesto
en el estator.
  SIGNAL PROY_A:REAL;
  SIGNAL PROY_B:REAL;
  SIGNAL PROY_C:REAL;
  SIGNAL ARG:REAL; -- Posición del eje del motor.
  SIGNAL ESCALA:REAL;
BEGIN
    PROCESS
    BEGIN
      IF s_estado = I THEN
        ARG <= 0.0;
        ESCALA <= 100.0;
        PROY <= 0.0;
        PROY_A <= 0.0;
        PROY_B <= 0.0;
        PROY_C <= 0.0;
        s_estado <= A;
      ELSE
        ALFA_A <= ARG_FA - ARG;
        ALFA_B <= ARG_FB - ARG;
        ALFA_C <= ARG_FC - ARG;
        PROY_A <= FASEA * SIN(ALFA_A);
        PROY_B <= FASEB * SIN(ALFA_B);
        PROY_C <= FASEC * SIN(ALFA_C);
        PROY <= PROY_A + PROY_B + PROY_C;
        ARG <= ARG + PROY / ESCALA;
```

```
        CASE s_estado IS
        WHEN A =>
          HALL <= "011";
          IF (ARGANT <= lsup1 AND ARG > lsup1) THEN
            s_estado <= B;
          ELSIF (ARGANT >= lsup0 AND ARG < lsup0) THEN
            s_estado <= F;
            ARG <= ARG  + 6.28;
          END IF;
        WHEN B =>
          HALL <= "010";
          IF (ARGANT <= lsup2 AND ARG > lsup2) THEN
            s_estado <= C;
          ELSIF (ARGANT >= lsup1 AND ARG < lsup1) THEN
            s_estado <= A;
          END IF;
        WHEN C =>
          HALL <= "110";
          IF (ARGANT <= lsup3 AND ARG > lsup3) THEN
            s_estado <= D;
          ELSIF (ARGANT >= lsup2 AND ARG < lsup2) THEN
            s_estado <= B;
          END IF;
        WHEN D =>
          HALL <= "100";
          IF (ARGANT <= lsup4 AND ARG > lsup4) THEN
            s_estado <= E;
          ELSIF (ARGANT >= lsup3 AND ARG < lsup3) THEN
            s_estado <= C;
          END IF;
        WHEN E =>
          HALL <= "101";
          IF (ARGANT <= lsup5 AND ARG > lsup5) THEN
            s_estado <= F;
          ELSIF (ARGANT >= lsup4 AND ARG < lsup4) THEN
            s_estado <= D;
          END IF;
        WHEN F =>
          HALL <= "001";
          IF (ARGANT <= lsup6 AND ARG > lsup6) THEN
            s_estado <= A;
            ARG <= ARG - 6.28;
          ELSIF (ARGANT >= lsup5 AND ARG < lsup5) THEN
            s_estado <= E;
          END IF;
```

```
        WHEN OTHERS =>
          HALL <= "000";
        END CASE;
      END IF;
      ARGANT <= ARG;
      WAIT FOR muestreo;
    END PROCESS;
END arq_motor;
```

Figura 9. Código VHDL del motor.

Componente fuente.vhd

Este componente simplemente simula las salidas de una fuente partida simétrica que alimentará a las diferentes llaves que dominan la polaridad, conexión y desconexión de cada fase.

```
--------------------------------------------------------------------------------
--  File Name: fuente.vhd
--------------------------------------------------------------------------------
--
--  Fuente partida simétrica.
--
--------------------------------------------------------------------------------
LIBRARY IEEE;
--USE ieee.std_logic_1164.all;
--USE ieee.std_logic_arith.all ;
USE ieee.math_real.all;
--USE  work.user_pkg.all;
--------------------------------------------------------------------------------
-- DECLARACION DE LA ENTIDAD
--------------------------------------------------------------------------------
ENTITY fuente IS
GENERIC(
  V:REAL:=0.45e+1
);
PORT (
  VCC:OUT REAL;
  GROUND:OUT REAL;
  VDD:OUT REAL
);
END fuente;
--------------------------------------------------------------------------------
-- DECLARACION DE LA ARQUITECTURA
```

```
--------------------------------------------------------------------------------
ARCHITECTURE arq_fuente OF fuente IS
BEGIN
  --------------------------------------------------------------------------------
  -- PROCESO VALOR DE RECARGA
  --------------------------------------------------------------------------------
  main:PROCESS
              BEGIN
                VCC <= V;
                VDD <= -V;
                GROUND <= 0.0e+0;
                WAIT;
    END PROCESS;
END arq_fuente;
```

Figura 10. Código VHDL de la fuente partida.

Componente TP2.vhd (Testbench)

Finalmente el proyecto se conforma con este último componente que define la relación entre todos los componentes presentados con anterioridad y pretende reflejar la configuración introducida al inicio de esta sección.

```
--------------------------------------------------------------------------------
--  File Name: TP02.vhd
--------------------------------------------------------------------------------
--
--  Trabajo Práctico N°2
--     Control de motores con FPGA.
--
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.math_real.all;
ENTITY TP02 IS
  GENERIC(
    DATA0: STD_LOGIC_VECTOR := X"04";
    DATA1: STD_LOGIC_VECTOR := X"01"
  );
END TP02;

ARCHITECTURE arq_TP02 OF TP02 IS

COMPONENT reloj
```

```
        PORT (
                CLK :  OUT  STD_LOGIC
        );
END COMPONENT;
COMPONENT contador
  PORT (
    CLK:IN STD_LOGIC;
    RST:IN STD_LOGIC;
    IDC:IN STD_LOGIC;
    RELOAD:IN STD_LOGIC_VECTOR(7 downto 0);
    TC:BUFFER STD_LOGIC
  );
END COMPONENT;
COMPONENT mux
  PORT (
    INPUT:IN STD_LOGIC;
    HALL:IN STD_LOGIC_VECTOR(2 DOWNTO 0);
    CLKW:IN STD_LOGIC;
    OUTPUT:OUT STD_LOGIC_VECTOR(5 DOWNTO 0)
  );
END COMPONENT;
COMPONENT fuente
  PORT (
    VCC:OUT REAL;
    GROUND:OUT REAL;
    VDD:OUT REAL
  );
END COMPONENT;
COMPONENT motor
  PORT (
    FASEA:IN REAL;
    FASEB:IN REAL;
    FASEC:IN REAL;
    HALL:BUFFER STD_LOGIC_VECTOR (2 DOWNTO 0)
  );
END COMPONENT;
COMPONENT llave
  PORT (
    ENT1:IN REAL;
    ENT2:IN REAL;
    GND:IN REAL;
    CON1:IN STD_LOGIC;
    CON2:IN STD_LOGIC;
    SAL:OUT REAL
  );
```

```
END COMPONENT;
-- Internal signal declaration
SIGNAL sig_clock      : std_logic;
SIGNAL sig_reset      : std_logic;
SIGNAL sig_tc: std_logic;
SIGNAL sig_tc2: std_logic;
SIGNAL sig_hall:STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL sig_llaves:STD_LOGIC_VECTOR(5 DOWNTO 0);
SIGNAL sig_vcc:REAL;
SIGNAL sig_ground:REAL;
SIGNAL sig_vdd:REAL;
SIGNAL sig_fase_a:REAL;
SIGNAL sig_fase_b:REAL;
SIGNAL sig_fase_c:REAL;
SIGNAL sig_clkw:STD_LOGIC;

BEGIN
reloj0 : reloj
PORT MAP(
  CLK => sig_clock
);
contador0 : contador
PORT MAP(
  CLK => sig_clock,
  RST => sig_reset,
  IDC => sig_tc,
  RELOAD => DATA0,
  TC => sig_tc
);
contador1 : contador
PORT MAP(
  CLK => sig_tc,
  RST => sig_reset,
  IDC => sig_tc2,
  RELOAD => DATA1,
  TC => sig_tc2
);
fuente1 : fuente
PORT MAP(
    VCC => sig_vcc,
    GROUND => sig_ground,
    VDD => sig_vdd
);
mux1 : mux
PORT MAP(
```

```
    INPUT => sig_clock,
    CLKW => '1',
    HALL => sig_hall,
    OUTPUT => sig_llaves
);
llave1 : llave
PORT MAP(
    ENT1 => sig_vcc,
    ENT2 => sig_vdd,
    GND => sig_ground,
    CON1 => sig_llaves(0),
    CON2 => sig_llaves(3),
    SAL => sig_fase_a
);
llave2 : llave
PORT MAP(
    ENT1 => sig_vcc,
    ENT2 => sig_vdd,
    GND => sig_ground,
    CON1 => sig_llaves(1),
    CON2 => sig_llaves(4),
    SAL => sig_fase_b
);
llave3 : llave
PORT MAP(
    ENT1 => sig_vcc,
    ENT2 => sig_vdd,
    GND => sig_ground,
    CON1 => sig_llaves(2),
    CON2 => sig_llaves(5),
    SAL => sig_fase_c
);
motor1 :motor
PORT MAP(
    FASEA => sig_fase_a,
    FASEB => sig_fase_b,
    FASEC => sig_fase_c,
    HALL => sig_hall
);
main:PROCESS
BEGIN
    sig_reset <= '1';
         WAIT FOR 50ns;
         sig_clkw <= '1';
    sig_reset <= '0';
```

```
    WAIT FOR 500ns;
          sig_clkw <= '0';
    WAIT;
END PROCESS;
END arq_TP02;
```

Figura 11. Código VHDL del proyecto.

Simulación del modelo VHDL

Presentamos a continuación los resultados obtenidos a partir de la simulación del modelo conformado en VHDL.

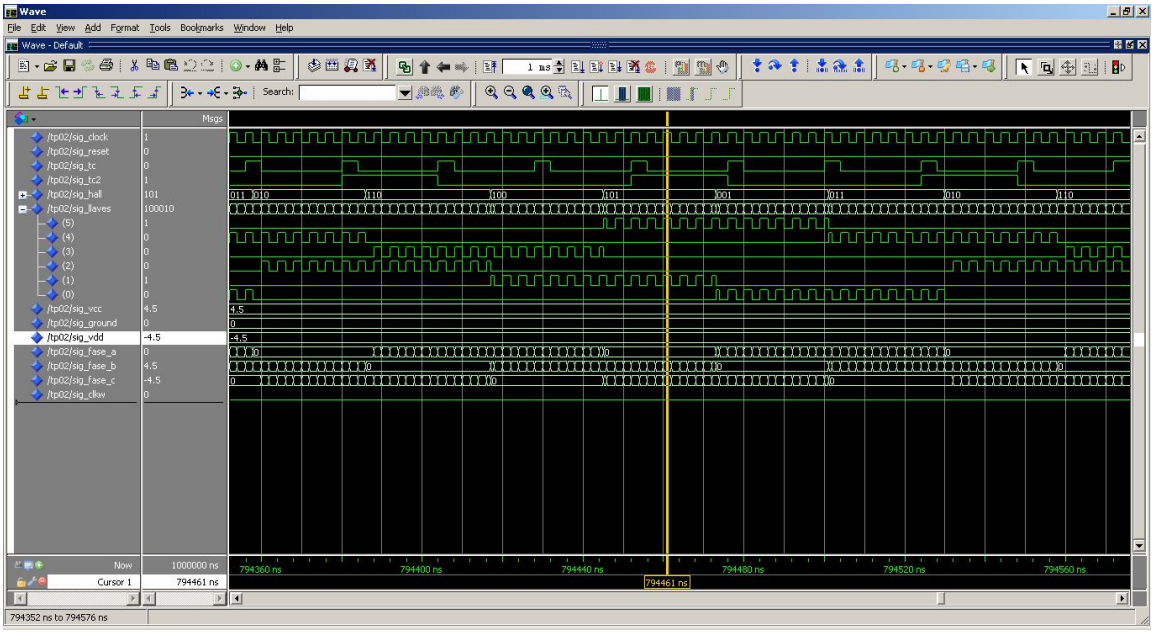


Figura 13. Simulación del sistema completo.

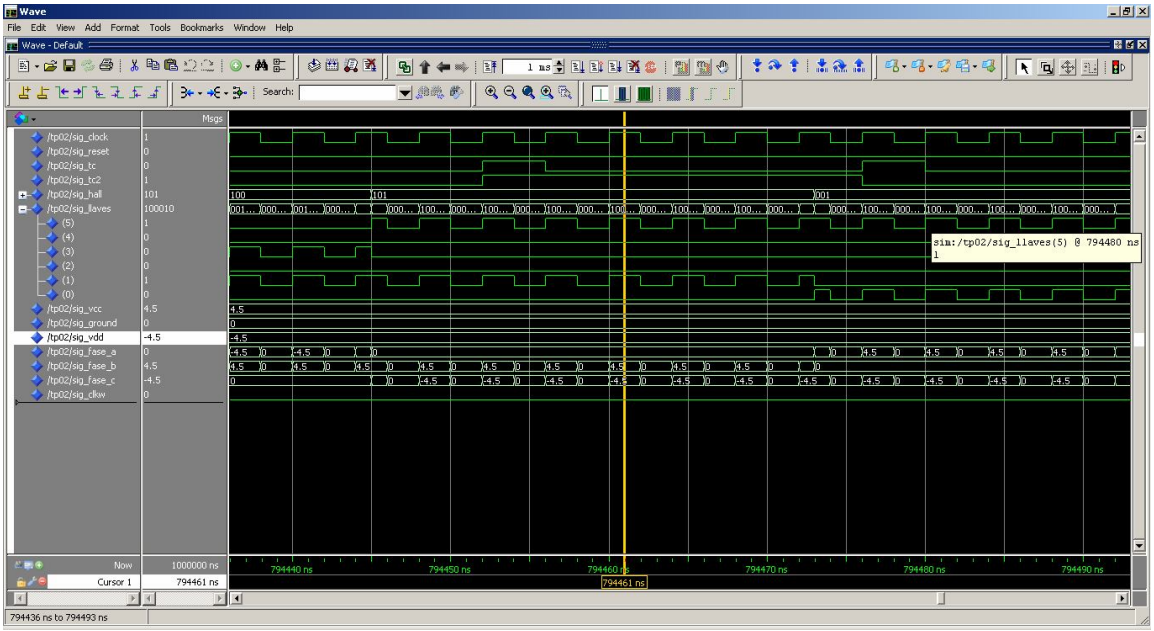


Figura 14. Simulación del sistema completo (Visualización de las tensiones de alimentación de las fases).

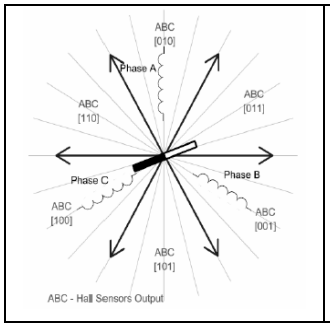
En las imágenes anteriores se discriminan las siguientes señales:

SEÑAL	DESCRIPCIÓN
sig_clock	Señal con un ciclo de actividad del 50%
sig_reset	Señal de reset
sig_tc1	Señal de fin de conteo del contador 1.
sig_tc2	Señal de fin de conteo del contador 2, en cascada con el contador 1.
sig_hall	Bus de comunicación entre los sensores HALL en el motor y el MUX. La información de este bus corresponde a las fases CBA, presentando diferencias con las tablas que presentan las fases ABC.
sig_llaves	Señales enviadas desde el MUX para activar los “transistores de potencia”.
sig_vcc	Tensión positiva de la fuente partida.
sig_vdd	Tensión negativa de la fuente partida.
sig_ground	Conexión a tierra de la fuente partida.
sig_fase_a	Fase A del motor brushless.
sig_fase_b	Fase B del motor brushless.
sig_fase_c	Fase C del motor brushless.

sig_clkw	Sentido de giro. (1) En sentido horario. (0) En sentido antihorario.

Figura 15. Descripción de las señales utilizadas.

Se puede verificar en la simulación, que para el sentido establecido por el valor de la señal sig_clkw (Sentido antihorario) la secuencia de los valores recuperados por el sensor de efecto HALL, y por ende, la de activación de las fases del motor se fijan de acuerdo al siguiente cuadro.



Hall Sensor A	Hall Sensor B	Hall Sensor C	Phase A	Phase B	Phase C
1	0	0	$+V_{DCB}$	$-V_{DCB}$	NC
1	1	0	$+V_{DCB}$	NC	$-V_{DCB}$
0	1	0	NC	$+V_{DCB}$	$-V_{DCB}$
0	1	1	$-V_{DCB}$	$+V_{DCB}$	NC
0	0	1	$-V_{DCB}$	NC	$+V_{DCB}$
1	0	1	NC	$-V_{DCB}$	$+V_{DCB}$

Figura 16. Progresión de las lecturas del sensor de efecto Hall para el eje girando en sentido horario.

Se debe tener en consideración que la información de la señal sig_hall corresponde a las fases CBA, visualizando a la inversa del valor presentado en la tabla.

Conclusiones

Al momento de realizar las graficas en el MATLAB para verificar las respuestas obtenidas con el modelo en función de diferentes entradas, fue fundamental basarnos en la física aplicada para entender si la simulación era correcta o no, al principio tuvimos que corregir y simular varias veces el script hasta entender donde se alojaban los errores cuando se escribió el código.

Observamos que la fuerza que se debe ejercer para mover el brazo robótico a lo largo de la articulación prismática no se ve afectada por las posiciones las otras articulaciones. Esto se debe a que las componentes de fuerza que aplican sobre los eslabones son en su totalidad ortogonales al desplazamiento de la articulación prismática.

Al estudiar la dinámica de nuestro robot obtuvimos el conocimiento necesario para definir la capacidad mecánica que debemos proveer con los motores que movilizan a las diferentes articulaciones. Esto define uno de los parámetros más importantes para seleccionar los motores adecuados para cada articulación, evitando así sobredimensionar excesivamente los motores, ó peor aún subdimensionarlos.

Se identifico el VHDL como una herramienta universal para definir el comportamiento tanto de circuitos electrónicos como de modelos físicos - matemáticos que reflejen el comportamiento de los dispositivos a controlar, facilitando la comprobación de las funciones esperadas del circuito electrónico a integrar en el FPGA dentro de la una herramienta de simulación única. De la misma manera que modelamos el motor brushless, se podría haber conformado un modelo que refleje el comportamiento de los transistores de potencia y reemplazar los componentes “llave” definidos en el presente trabajo.

En este TP asentamos conocimientos sobre las herramientas de modelado dinámico de una configuración robótica. Los mismos comenzaron como entes abstractos, ajenos a todo tipo de aplicación y difíciles de comprender, pero con el uso hicieron posible obtener de forma sistemática y concreta las representaciones matemáticas requeridas para definir las características físicas del robot bajo estudio.