# Universidad Tecnológica Nacional

# Facultad Regional Buenos Aires

## Asignatura: Robótica

**Prof: Ing. Hernán Giannetta**

# Trabajo Práctico N°2: Análisis Dinámico de un Robot e Implementación sobre FPGA

**Alumnos: Mobrici, Emiliano y Ermida Damian**

**Fecha de Entrega: 10/07/2009**

Aprobado

# INDICE

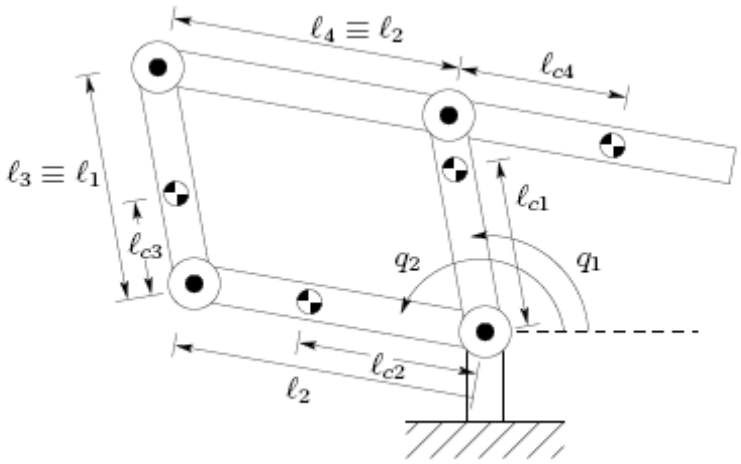
**ANÁLISIS DINÁMICO ..... 3**
**RESOLUCIÓN DE UN ROBOT "FIVE BAR LINKAGE" ..... 3**
**RESOLUCIÓN DE UN CASO PRÁCTICO: ..... 5**
**ELECCIÓN DEL SERVOMOTOR: ..... 10**

**IMPLEMENTACIÓN EN VHDL ..... 11**
**INTRODUCCIÓN AL LENGUAJE VHDL ..... 11**
**ESTRUCTURA DEL LENGUAJE ..... 11**
ENTIDAD - ENTITY ..... 12
ARQUITECTURA - ARCHITECTURE ..... 13
DISEÑO CONCURRENTE: ..... 14
DISEÑO SECUENCIAL: ..... 14
DISEÑO SECUENCIAL: ..... 15
**PLANTEO DEL PROBLEMA ..... 16**
**PLANTEO DE LA SOLUCIÓN ..... 17**
**DESARROLLO DE LA SOLUCIÓN ..... 18**
GENERADOR DE MODULACIÓN PWM ..... 18
GENERADOR SECUENCIA ..... 20
GENERADOR DE SEÑALES DE CONTROL DE TRANSISTORES ..... 21
MODULO DE CONTROL PWM DEL MOTOR ..... 22
**SIMULACIÓN ..... 26**

# Análisis Dinámico

## Resolución de un robot “Five bar linkage”

A continuación se analizará esta estructura clásica en el diseño de robots, cuya particularidad radica en representar una cadena cinemática cerrada.




Para lograr manipular de manera óptima el robot, será necesario que los movimientos q1 y q2 sean independientes entre sí, lo que puede ser logrado únicamente cuidando las relaciones l1=l3 y l2=l4, formando de este modo un paralelogramo. Cabe destacar que las distancias a los centros de masa, lc1, lc3, lc2, lc4 no deberán cumplir necesariamente con esta relación, lo que implica que si bien las barras deberán tener la misma longitud, la distribución de masa podría ser distinta.

Luego planteamos las ecuaciones para los cuatro centros de masa:

$$\begin{bmatrix} xc1 \\ yc1 \end{bmatrix} = \begin{bmatrix} lc1.\cos q1 \\ lc1.senq1 \end{bmatrix}$$

$$\begin{bmatrix} xc2 \\ yc2 \end{bmatrix} = \begin{bmatrix} lc2.\cos q2 \\ lc2.senq2 \end{bmatrix}$$

$$\begin{bmatrix} xc3 \\ yc3 \end{bmatrix} = \begin{bmatrix} lc1.\cos q1 \\ lc1.senq1 \end{bmatrix} + \begin{bmatrix} lc2.\cos q2 \\ lc2.senq2 \end{bmatrix}$$

$$\begin{bmatrix} xc4 \\ yc4 \end{bmatrix} = \begin{bmatrix} l1.\cos q1 \\ l1.senq1 \end{bmatrix} + \begin{bmatrix} lc4.\cos(q2-\Pi) \\ lc4sen(q2-\Pi) \end{bmatrix} \cdot \begin{bmatrix} l1.\cos q1 \\ l1.senq1 \end{bmatrix} - \begin{bmatrix} lc4.\cos q2 \\ lc4.senq2 \end{bmatrix}$$

Ahora estamos en condiciones de obtener las ecuaciones de velocidades de los centros de masa:

$$vc1 = \begin{bmatrix} -lc1.senq1 & 0 \\ lc1.\cos q1 & 0 \end{bmatrix}.q'$$

$$vc2 = \begin{bmatrix} 0 & -lc2.senq2 \\ 0 & lc2.\cos q2 \end{bmatrix}.q'$$

$$vc3 = \begin{bmatrix} -lc3.senq1 & -l2.senq2 \\ lc3.\cos q1 & l2.\cos q2 \end{bmatrix}.q'$$

$$vc4 = \begin{bmatrix} -l1.senq1 & lc4.senq2 \\ l1.\cos q1 & lc4.\cos q2 \end{bmatrix}.q'$$

Ahora vamos a plantear:

$$w1 = w3 = \dot{q}_1.k \qquad w2 = w34 = \dot{q}_2.k$$

Luego hallamos la matriz inercia:

$$D(q) = \sum_{i=1}^{4} miJ_{vc}^{T} J_{vc} + \begin{bmatrix} I1 + I3 & 0 \\ 0 & I2 + I4 \end{bmatrix}$$

Si introducimos las ecuaciones de velocidad en la ecuación anterior:

$$d_{11}(q) = m_1.l_{c1}^2 + m_3.l_{c3}^2 + m_4.l_1^2 + I_1 + I_3$$

$$d_{12}(q) = d_{21}.(q) = (m_3 l_2 l_{c3} + m_4 l_1 l_{c4}) \cos(q_2 - q_1)$$

$$d_{22}(q) = m_2.l_{c2}^2 + m_3.l_2^2 + m_4.l_{c4}^2 + I_2 + I_4$$

Bajo la condición:

$$m_3.l_2.l_{c3} = m_4.l_1 l_{c4}$$

Por cuanto d12 y d21 valen cero, podemos expresar la energía potencial como:

$$P = g\sum_{i=1}^{4} y_{ci} = g.sen_{q1}(m_1 l_{c1} + m_3 l_{c3} + m_4 l_1) + g.sen_{q2}(m_2 l_{c2} + m_3 l_2 - m_4 l_{c4})$$

Por lo tanto:

$$\phi_1 = g.\cos q_1(m_1.lc_1 + m_3 lc_3 + m_4 l_1)$$

$$\phi_2 = g.\cos q_2(m_2.lc_2 + m_3 l_2 - m_4 l_{c4})$$

Como ya mencionamos, □1 depende solo de q1 mientras que □2 depende de q2. Por lo tanto podemos escribir:

$$d_{11}.\ddot{q}_1 + \phi_1(q_1) = \tau_1 \qquad d_{22}.\ddot{q}_2 + \phi_2(q_2) = \tau_2$$

## Resolución de un caso práctico:

Para la resolución se definen los siguientes parámetros:
L3 = L1 = 1m L4=L2=1,5m
m3=m1=1,5Kg m4=1,875Kg m2=1,125Kg
Lc1=0,75m Lc2=0,45m Lc3=0,25m Lc4=0,25m

$$d_{11}(q) = m_1.l_{c1}^2 + m_3.l_{c3}^2 + m_4.l_1^2 + I_1 + I_3 = 2(m_1.l_{c1}^2) + 2.(m_3.l_{c3}^2) + m_4.l_1^2 = 3{,}75$$

$$d_{22}(q) = m_2.l_{c2}^2 + m_3.l_2^2 + m_4.l_{c4}^2 + I_2 + I_4 = 2.(m_2.l_{c2}^2) + 2.(m_4.l_{c4}^2) + m_3.l_2^2 = 4{,}065$$

$$\tau_1 = d_{11}.\ddot{q}_1 + \phi_1(q_1) = 3{,}75\ddot{q}_1 + 9{,}8\cos q_1(3{,}375)$$

$$\tau_2 = d_{22}.\ddot{q}_2 + \phi_2(q_2) = 4{,}065\ddot{q}_2 + 9{,}8\cos q_2(2{,}2875)$$

A continuación debemos definir trayectorias, para lo cual propondremos las siguientes funciones para q1 y q2:

$$q_1 = \frac{t^2}{20}$$

Con t=[seg] y q1= rad para 0<t<5,6seg 0 <q1< □□

De modo que obtendríamos:

$$q_1 = \frac{t^2}{20} \quad \dot{q}_1 = \frac{t}{10} \quad \ddot{q}_1 = \frac{1}{10}$$

Y por otra parte:

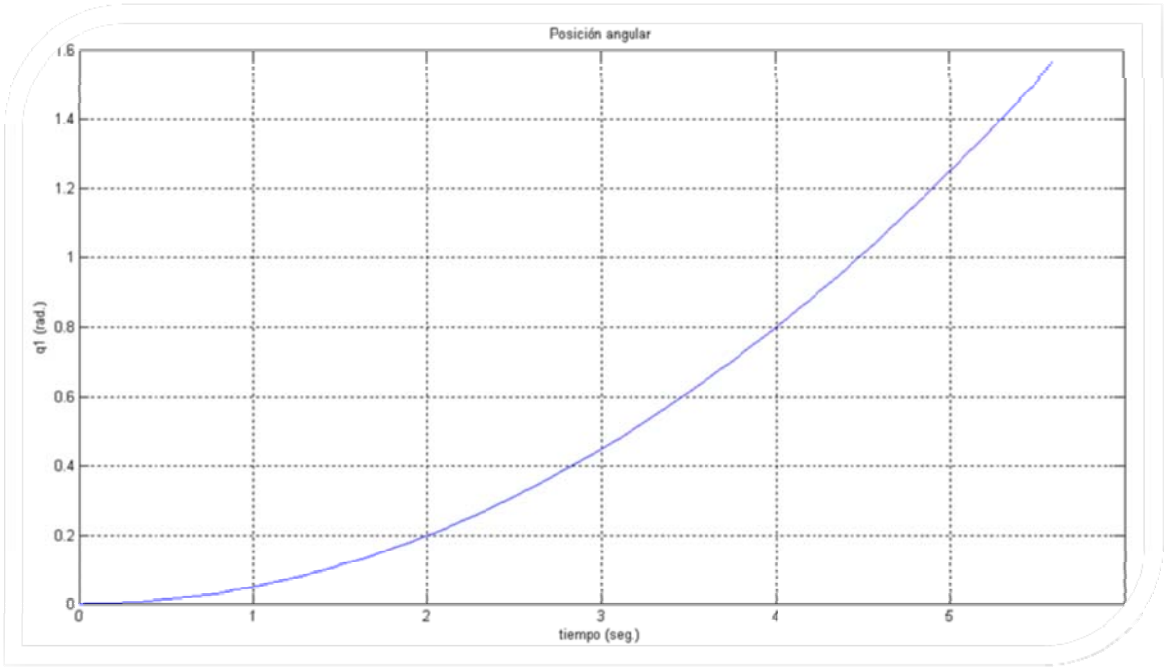
$$q_2 = \frac{t^3 - 4t^2}{20} + 0{,}78$$

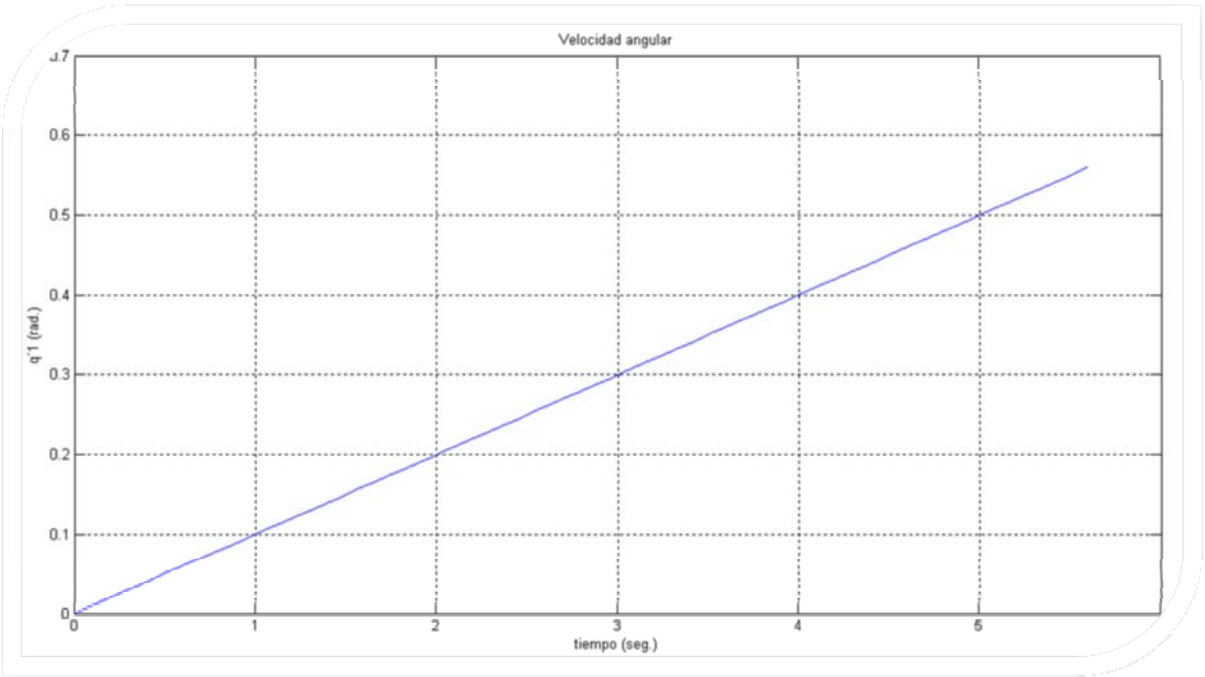
Con t=[seg] y q2= rad para 0<t<5,6seg □□ <q1<

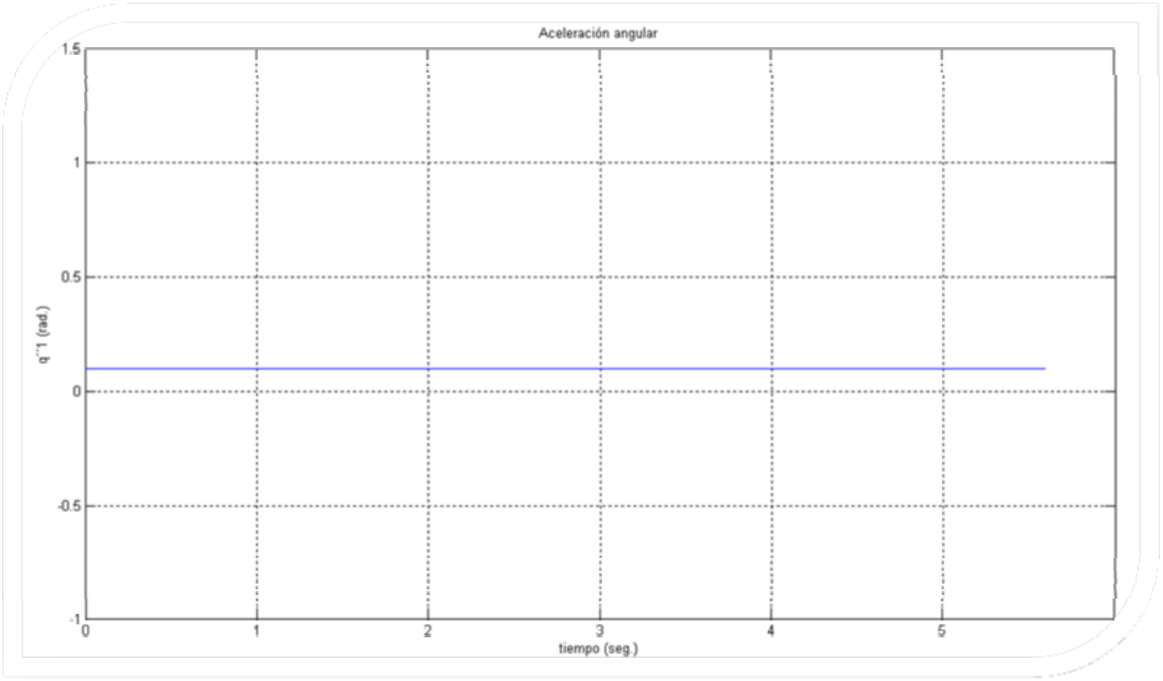
De modo que obtendríamos:

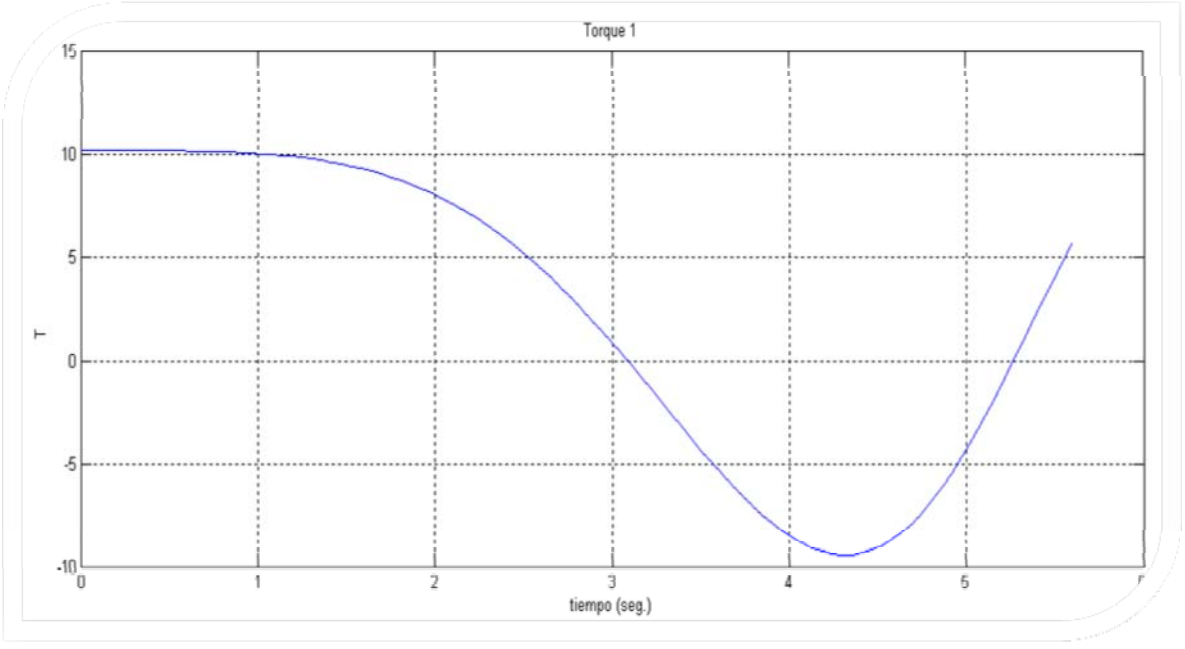
$$q_2 = \frac{t^3 - 4t^2}{10} + 0{,}78 \quad \dot{q}_2 = \frac{3.t^2 - 8t}{10} \quad \ddot{q}_2 = \frac{6.t - 8}{10}$$

Graficando estas expresiones obtenemos:
Q1

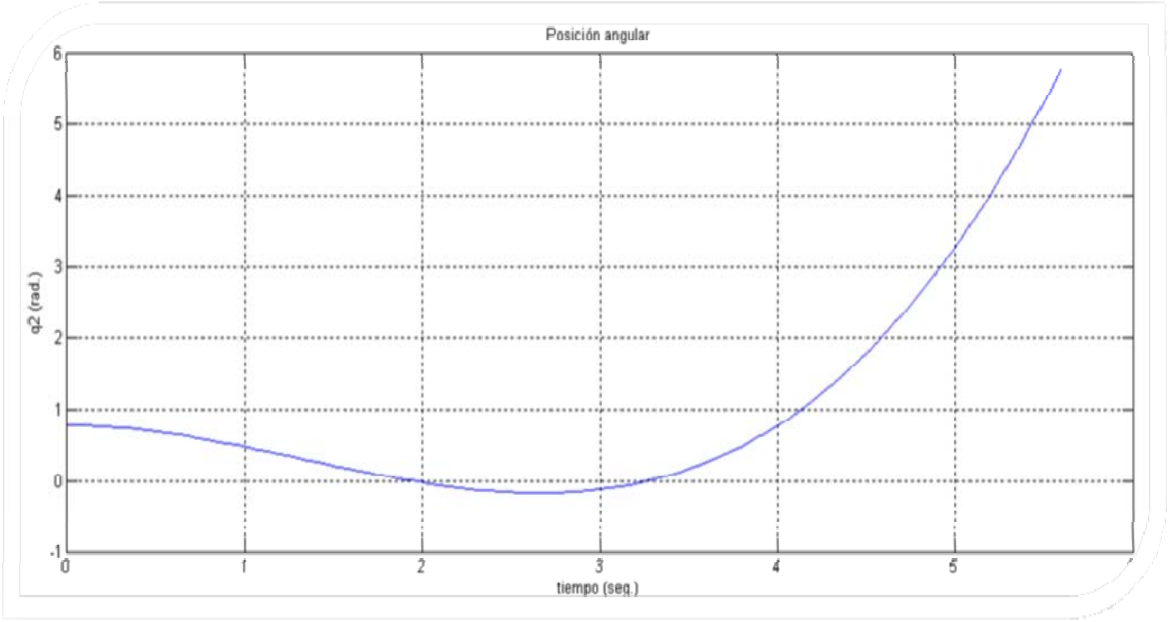


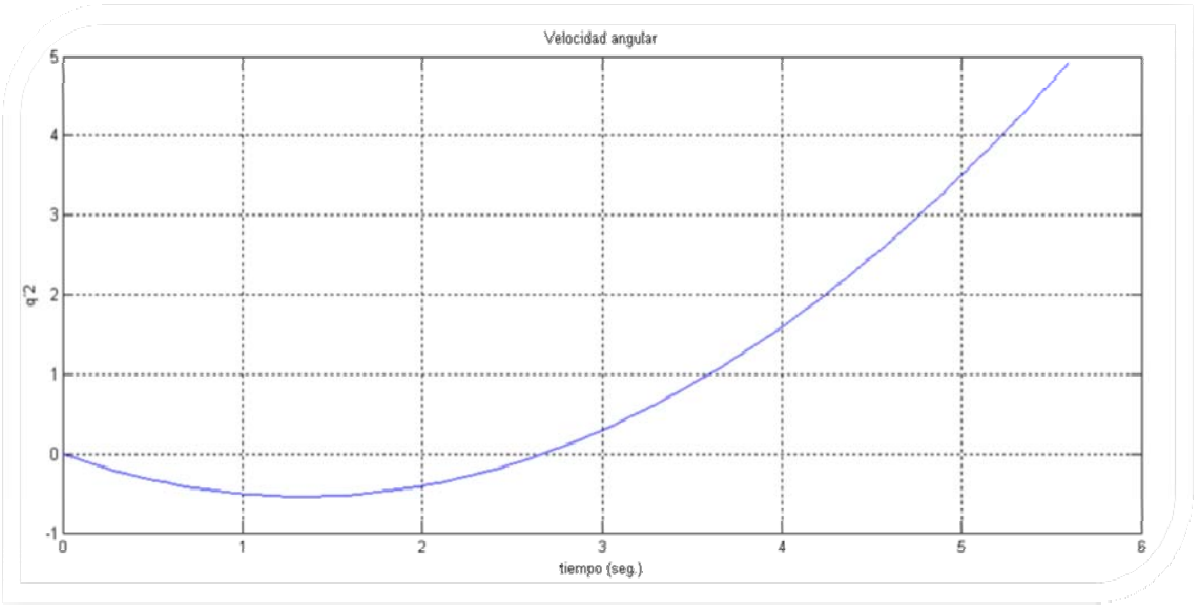



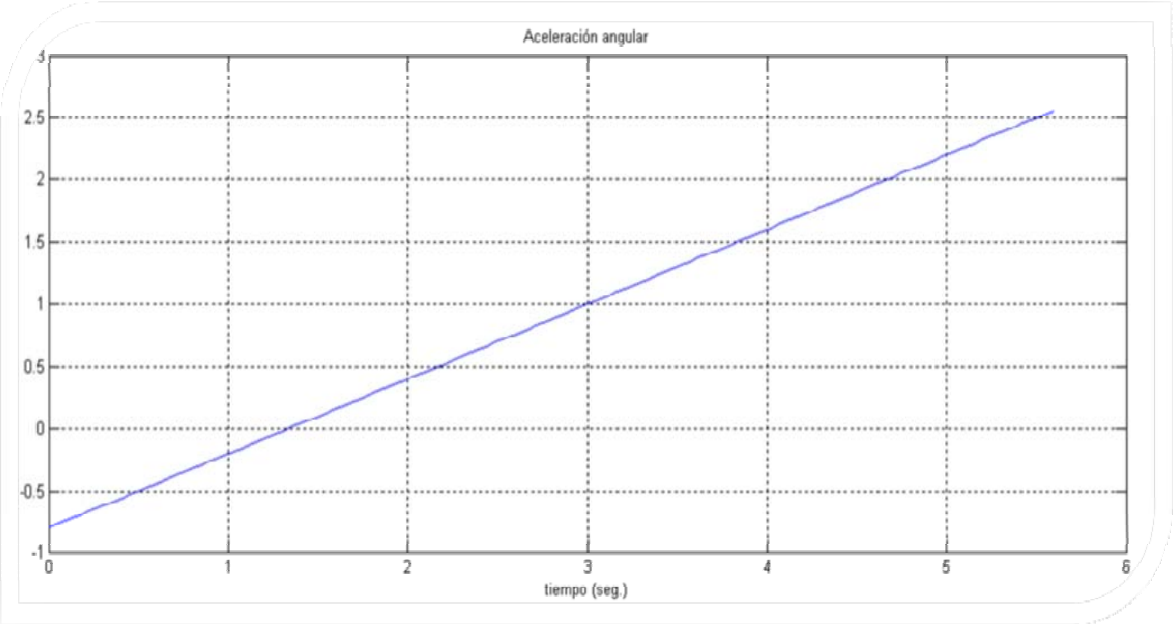

Aceleración angular
q''1 (rad.)
tiempo (seg.)

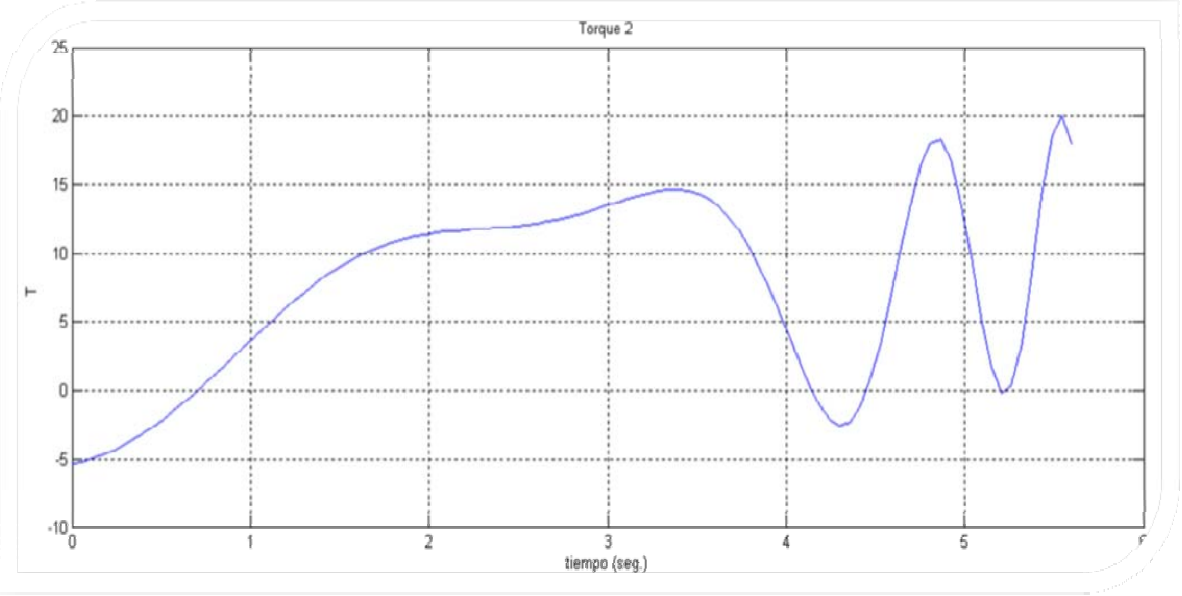

Torque 1
T
tiempo (seg.)

Q2


Posición angular
q2 (rad.)
tiempo (seg.)
6
5
4
3
2
1
0
-1
0
1
2
3
4
5


Velocidad angular
q'2
tiempo (seg.)
5
4
3
2
1
0
-1
0
1
2
3
4
5
6


Aceleración angular
3
2.5
2
1.5
1
0.5
0
-0.5
-1
0
1
2
3
4
5
6
tiempo (seg.)


Torque 2
25
20
15
10
5
0
-5
-10
T
0
1
2
3
4
5
tiempo (seg.)

## Elección del servomotor:

En base a los gráficos que resultan de resolver las ecuaciones de posición, velocidad, aceleraciones y torques podemos plantear los siguientes valores:

$$\dot{q}\max = 5\frac{rad}{seg} = 47{,}74rpm$$
$$\tau\max = 20Nm$$

Dado que los valores de ambos movimientos (q1 y q2) son considerablemente semejantes, se plantan las peores condiciones de modo de utilizar el mismo modelo de motor para ambos.
Ahora plantamos la relación:

$$\tau 1.\omega 1.\eta = \tau 2.\omega 2$$

De modo que partiendo de un motor con velocidad 3000rpm y una caja de reducción de rendimiento del 80%:

$$\tau 1\max. = \frac{\tau 2\max.\omega 2}{\omega 1.\eta} = \frac{20Nm.47{,}74rpm}{3000rpm.0.8} = 0{,}4Nm$$

Por lo que podemos establecer las especificaciones del motor en 3000rpm y un torque máximo de 0,4Nm
Debido a que tenemos requerimientos de potencia relativamente bajos, podríamos utilizar un motor de marca Schneider modelo BRH 0571P combinado con un servo drive LXM 05AD10F1,
Lo que nos brinda un torque nominal de 0,46 Nm.

| BSH (IP 50 or IP 65) | LXM 05AD10F1, BD10F1 Continuous output current: 4 A rms | | | |
|---|---|---|---|---|
| | Nominal operating point | | | Standstill torques |
| | Nominal torque | Nominal speed | Nominal power | $M_0$ / $M_{max}$ (1) |
| | Nm | rpm | W | Nm/Nm |
| BSH 0551T | 0.46 | 3000 | 150 | 0.5/1.4 |

Si bien el motor se encuentra ligeramente sobredimensionado se lo elige de este modo para que posea una baja exigencia.

# Implementación en VHDL

## Introducción al Lenguaje VHDL

El significado de la palabra VHDL es la conjunción de dos acrónimos: "**V"** por *Very High Speed Integrated Circuit* y "**HDL**" por *Hardware Description Lenguage.*

VHDL es un lenguaje diseñado por el IEEE para describir el comportamiento de sistemas de hardware digital. El objetivo de este lenguaje es que se lo utilice para diseñar, modelar y simular circuitos digitales. Es por esta razón que su utilización en síntesis no es inmediata, aunque lo cierto es que la sofisticación de las actuales herramientas de síntesis es tal que permiten implementar diseños especificados en un alto nivel de abstracción.

Ventajas de utilización de VHDL:

- VHDL permite diseñar, modelar, y comprobar un sistema desde un alto nivel de abstracción bajando hasta el nivel de definición estructural de puertas.
- Circuitos descritos utilizando VHDL, siguiendo unas guías para síntesis, pueden ser utilizados por herramientas de síntesis para crear implementaciones de diseños a nivel de puertas.
- Al estar basado en un estándar (IEEE Std 1076-1987) los ingenieros de toda la industria de diseño pueden usar este lenguaje para minimizar errores de comunicación y problemas de compatibilidad.
- Permite diseño Top-Down, esto es, permite describir (modelado) el comportamiento de los bloques de alto nivel, analizándolos (simulación), y refinar la funcionalidad de alto nivel requerida antes de llegar a niveles más bajos de abstracción de la implementación del diseño.
- Modularidad: permite dividir o descomponer un diseño hardware y su descripción VHDL en unidades más pequeñas.

## Estructura del lenguaje

Un circuito electrónico puede ser parte de otro más grande, en este caso el primero sería un subcircuito del segundo.

Un circuito puede estar compuesto por muchos subcircuitos y estos subcircuitos se interconectarían.

Así aparece una jerarquía en el diseño. En la parte alta de la jerarquía aparecerían los circuitos más complejos, que estarían compuestos por subcircuitos y, a su vez, cada uno de estos subcircuitos podría estar compuesto por subcircuitos más sencillos.

Un ejemplo de jerarquía sería un microprocesador. El circuito más complejo y el más alto en la jerarquía sería el propio microprocesador. Éste estaría compuesto por subcircuitos, por ejemplo el de la unidad de control, el de la unidad aritmético-lógica, memorias, registros, etc. Estos subcircuitos estarían conectados por líneas eléctricas, pueden ser simples como un cable o complejas como un bus. Una unidad de control estaría compuesta por más subcircuitos, más registros, más buses, etc.

Cuando se está diseñando en un determinado nivel, seguramente se empleen elementos de niveles más bajos. Para usar estos elementos en un nivel más alto sólo se necesita conocer en primer lugar su comportamiento y por otro lado su interfaz, es decir, sus entradas y salidas.

Del mismo modo en VHDL para definir a un determinado elemento debemos declarar tanto su "**Entidad**" (Entity) que representa a su interfaz como su "**Arquitectura**" (Architecture) que representa al comportamiento que tiene el mismo.

## Entidad - Entity

La entidad sirve precisamente para definir las entradas y salidas que tendrá un determinado circuito.

En principio pudiera parecer que esta definición sea equivalente a la cabecera de una función de un lenguaje cualquiera de programación. En VHDL es más conveniente ver a la entidad como una caja negra con cables para las entradas y salidas. La ventaja de pensar en una entidad como en una caja negra a la que se conectan cables es que es más fácil comprender la ejecución concurrente que ocurrirá en el hardware.

La descripción de cómo funciona por dentro esa caja negra, como veremos mas adelante, es la arquitectura y de hecho una misma entidad puede tener distintas arquitecturas, convenientemente que tengan el mismo comportamiento, pero que de acuerdo al problema pueden ajustarse mejor y en consecuencia tener implementaciones físicas diferentes.

Una Entidad se declara como se indica en el código siguiente:

```
ENTITY NombreEntidad
        PORT(Señal: MODO Tipo)
END NombreEntidad;
```

Donde:

- ***NombreEntidad*** representa al identificador de esa entidad.
- ***PORT*** es la instrucción que indica que voy a definir cuales son las entradas/salidas.
- ***Señal*** es el nombre con que voy a identificar a esa entrada/salida en particular.
- ***Modo*** es el comportamiento que tiene dicha Señal, y este puede ser una de las siguientes cuatro posibilidades.
    - **IN:** En este modo las señales solo entran en la entidad
    - **OUT**: Las señales solo salen de la entidad
    - **BUFFER** Este modo se utiliza para las señales que además de salir de la entidad pueden usarse como entradas realimentadas
    - **INOUT**: Este modo se utiliza para señales vi direccionales.
- ***Tipo*** representa al formato que tendrá dicha señal. Existen varios tipos, los más comunes son:
    - **BIT:** En este tipo las señales solo toman los valores de "1" y "0".
    - **Booleana:** En este tipo las señales solo toman los valores de True y False

- **Std_logic:** En este tipo las señales toman 9 valores, entre ellos tenemos: "1", "0", "Z" (para el 3er estado), "-" (para los opcionales).
- **Integer**: En este tipo las señales toman valores enteros.
- **Bit_Vector**. En este tipo los valores de las señales son una cadena de unos y ceros. Ejemplo: “1000”
- **Std_Logic_Vector:** En este tipo los valores de las señales son una cadena de los nueve valores permisibles para el tipo std_logic.
- **Character**: Contiene todos los caracteres ISO de 8 bits, donde los primeros 128 son los caracteres ASCII.

## Arquitectura - Architecture

En la arquitectura es donde se describe el funcionamiento del módulo definido en la entidad. Una arquitectura siempre está referida a una entidad concreta por lo que no tiene sentido hacer declaraciones de arquitectura sin especificar la entidad. Como mencionamos anteriormente una misma entidad puede tener diferentes arquitecturas, es en el momento de la simulación o la síntesis cuando se especifica que arquitectura concreta se quiere simular o sintetizar.

Es importante marcar una diferencia entre el comportamiento que posee un microprocesador o microcontrolador respecto de un dispositivo programado en VHDL. Esta diferencia radica en que en los micros las operaciones del código se ejecutan en forma secuencial, en cambio en VHDL la ejecución de las instrucciones se ejecutan en forma concurrente.

La declaración de una Arquitectura se realiza empleando la palabra ARCHITECTURE y de la forma descripta en el código siguiente.

```
ARCHITECTURE nombre_arch OF la_entidad IS
declaraciones
BEGIN
        Instrucciones
END nombre_arch;
```

Donde:

- ***Nombre_arch***: es el nombre con que se identifica de la arquitectura.
- ***La_entidad***: es la entidad a la que se asocia esta arquitectura.
- En la zona de **Declaraciones** se declaran las señales internas, es decir, a ellas no se puede acceder desde la entidad, por los que los circuitos de nivel superior no podrían acceder a ellas. En un símil con un microprocesador, estas señales podrían ser las líneas que comunican la unidad central con la ALU, a las que no se puede acceder directamente desde el exterior del microprocesador. Obsérvese que en este caso no se indica si son entradas o salidas, puesto que al ser internas pueden ser leídas o escritas sin ningún problema.
- **BEGIN:** indica que a partir de ese punto comienza la descripción del comportamiento que tendrá la arquitectura.

En cuanto a la descripción del comportamiento VHDL admite tres formas bien distinguidas en las que puede realizarse:

- Diseño Concurrente
- Diseño Secuencial
- Diseño Estructural

## Diseño Concurrente:

A la hora de plantearse crear un programa en VHDL no hay que pensar como si fuera un programa típico para micro u ordenador. No hay que olvidar que en VHDL hay que describir un hardware, algo que no se hace en un programa para ordenador. Un circuito electrónico puede tener muchos elementos que estén ejecutando acciones a la vez, por ejemplo en un circuito puede tener una entrada que se aplique a dos puertas lógicas y de cada una obtener una salida, en este caso tendría dos caminos en los que se ejecutarían acciones (las puertas lógicas) de forma paralela. Esto es lo que se llama concurrencia.

VHDL es un lenguaje concurrente, como consecuencia no se seguirá el orden en que están escritas las instrucciones a la hora de ejecutar el código. De hecho, si hay dos instrucciones, no tiene porqué ejecutarse una antes que otra, pueden ejecutarse a la vez.

## Diseño Secuencial:

Si bien VHDL es un lenguaje concurrente también admite que ser secuencial al igual que en un micro. Para ello existe una instrucción (PROCESS) que al invocarla todo lo que el código que se encuentre luego de ella se ejecutará en forma secuencial.
No obstante únicamente se ejecuta en forma secuencial sólo lo que se encuentra contenido dentro del Process, por lo que sí podríamos tener varios procesos corriendo simultáneamente.

La sintaxis a emplear para poder ejecutar código en forma secuencial es la siguiente:

```
PROCESS [lista de sensibilidad]
   [declaración de variables]
BEGIN
   [sentencias secuenciales]
END PROCESS;
```

Como se observa la estructura es similar a la de la Arquitectura excepto en un aspecto y es la denominada “Lista de Sensibilidad”. En ella se enumeran todas las señales que, en caso de que sufran algún cambio, hacen que se ejecute el proceso.

## Diseño Secuencial:

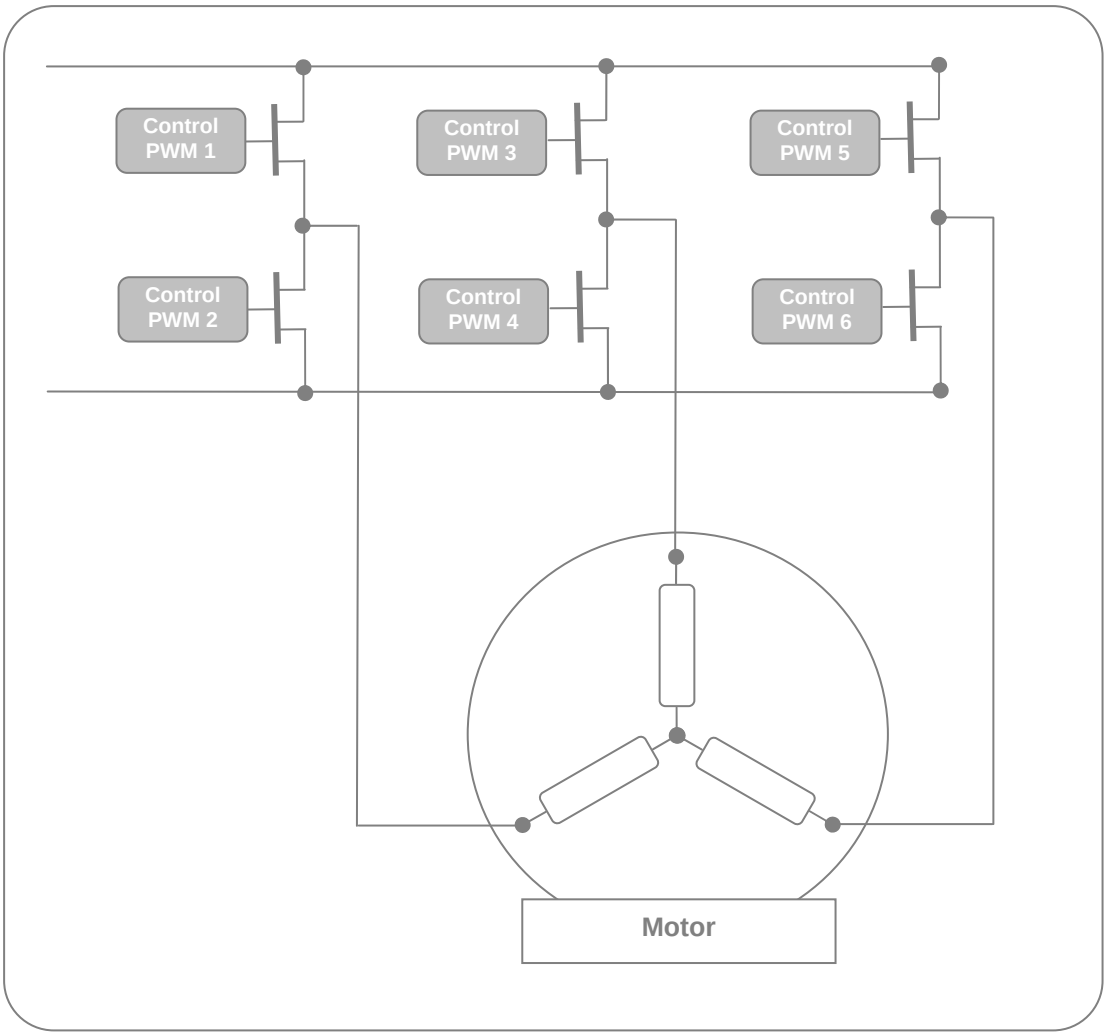
En este método de diseño se utilizan componentes descritos anteriormente y básicamente consiste en interconectar entre si de acuerdo al comportamiento que deseemos.

Para usar los componentes hay que seguir la siguiente secuencia:

- Declarar de los componentes a utilizar entre ARCHITECTURE y su BEGIN correspondiente. Para ello se emplea la palabra clave COMPONENT y a continuación se lo describe como si fuera una entidad pero reemplazando ENTITY por COMPONENT.
- Utilización de los componentes en el interior de la arquitectura. Es lo que se conoce como Instanciación. Suele realizarse esa operación suele realizarse realizando un PORT MAP donde se relacionan las distintas entradas/salidas con otras señales o entradas/salidas de otros componentes.

## Planteo del Problema

Debemos realizar el control PWM de la etapa de potencia de un motor trifásico. El esquema que se muestra a continuación describe gráficamente el problema a encarar.




El control que a desarrollar debe realizarse sobre los gates de los transistores MOS que controlas las bobinas del Motor.
Como debemos controlar la potencia que le damos al motor es conveniente realizarlo mediante el uso de una señal modulada en PWM.

De acuerdo al funcionamiento del motor solamente debemos aplicar tensión sobre dos de las bobinas, queda clara entonces que sólo tendremos funcionando a dos transistores a la vez. La secuencia en la que tenemos que activarlos para que se produzca el giro del rotor en un sentido es:

1. TR1 con TR6
2. TR5 con TR4
3. TR3 con TR2

Como dijimos, a su vez, debemos controlar la potencia aplicada utilizando modulación PWM, pero no es necesario que apliquemos esta modulación a ambos transistores. Lo

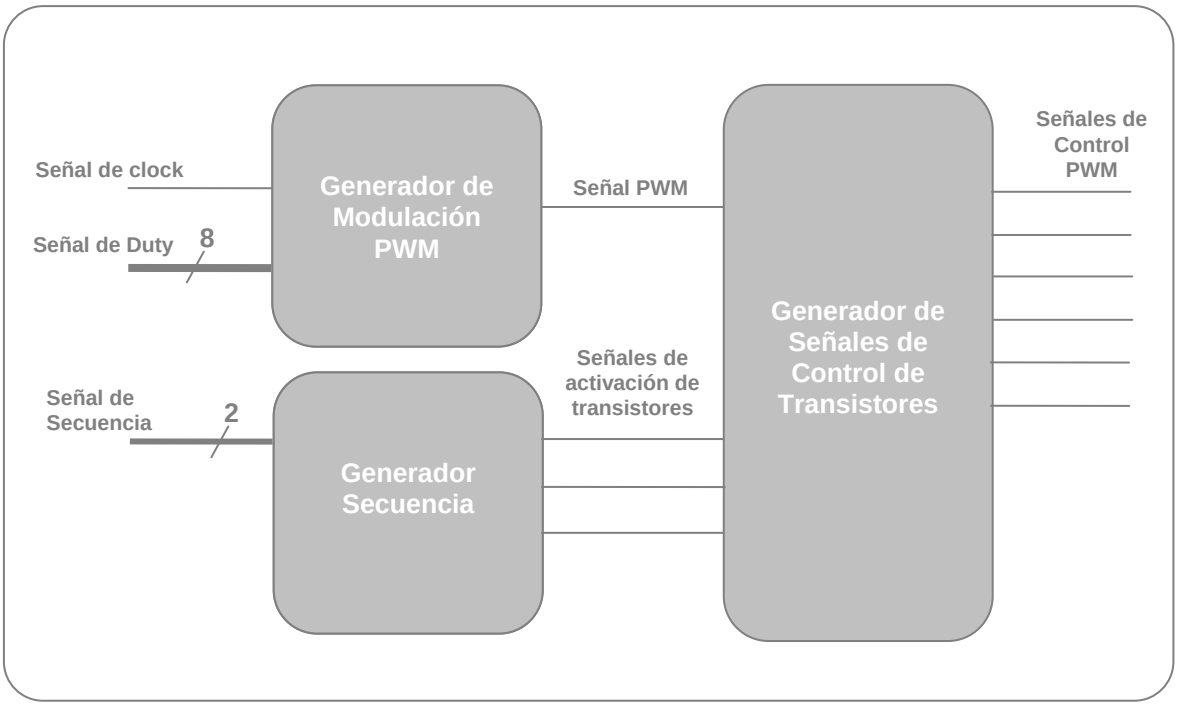
que realizaremos en cambio es activar al 100% los transistores de la rama superior y aplicar la modulación sobre los de la rama inferior.

## Planteo de la Solución

La solución que desarrollaremos se basa en tres bloques que cumplen las funciones de:

- Generar la señal modulada en PWM.
- Generar la secuencia en la deben activarse los transistores.
- Generar las señales que controlan a todos los transistores.

El siguiente diagrama en bloques muestra esta idea con mayor claridad.



# Desarrollo de la Solución

## Generador de Modulación PWM

Este modulo no fue desarrollado por nosotros sino que se empleó uno realizado por la firma ATMEL. El código del mismo se encuentra detallado a continuación:

```
--**********************************************
-- Archivo: pwm_fpga.vhd
-- Vendor : Atmel
-- Created : May 31, 2001 at : 11:45:56 am
--**********************************************

library IEEE;
USE   ieee.std_logic_1164.all;
USE     ieee.std_logic_arith.all ;
USE   work.user_pkg.all;


ENTITY pwm_fpga IS
PORT (  clock,reset                :in  STD_LOGIC;
        Data_value            :in  std_logic_vector(7 downto 0);
        pwm                   :out STD_LOGIC

     );

END pwm_fpga;

ARCHITECTURE arch_pwm OF pwm_fpga IS

SIGNAL reg_out                            : std_logic_vector(7 downto 0);
SIGNAL cnt_out_int                        : std_logic_vector(7 downto 0);
SIGNAL pwm_int, rco_int                   : STD_LOGIC;

BEGIN

 -- 8 BIT DATA REGISTER TO STORE THE MARKING VALUES .
 -- THE MARKING VALUES WILL DETERMINE THE DUTY CYCLE OF PWM OUTPUT

 PROCESS(clock,reg_out,reset)

            BEGIN
                  IF (reset ='1') THEN
                  reg_out <="00000000";
                  ELSIF (rising_edge(clock)) THEN
                        reg_out <= data_value;
                  END IF;
            END PROCESS;

 -- 8 BIT UPDN COUNTER. COUNTS UP OR DOWN BASED ON THE PWM_INT SIGNAL AND
GENERATES
 -- TERMINAL COUNT WHENEVER COUNTER REACHES THE MAXIMUM VALUE OR WHEN IT
TRANSISTS
 -- THROUGH ZERO. THE TERMINAL COUNT WILL BE USED AS INTERRUPT TO AVR FOR
GENERATING
 -- THE LOAD SIGNAL.
 --  INC and DEC are the two functions which are used for up and down
counting. They are defined in sepearate user_pakge library

 PROCESS (clock,cnt_out_int,rco_int,reg_out)

      BEGIN

       IF (rco_int = '1') THEN
```

```
           cnt_out_int <= reg_out;
       ELSIF  rising_edge(clock) THEN
            IF (rco_int = '0' and pwm_int ='1' and cnt_out_int <"11111111")
THEN
             cnt_out_int <= INC(cnt_out_int);
            ELSE
                IF  (rco_int  ='0'  and  pwm_int  ='0'  and  cnt_out_int  >
"00000000") THEN
                cnt_out_int <= DEC(cnt_out_int);
            END IF;
          END IF;
      END IF;
 END PROCESS;


-- Logic to generate RCO signal

 PROCESS(cnt_out_int, rco_int, clock,reset)
      BEGIN

           IF (reset ='1') THEN
             rco_int <='1';
          ELSIF rising_edge(clock) THEN
              IF  ((cnt_out_int  =  "11111111")  or  (cnt_out_int  ="00000000"))
THEN
                 rco_int <= '1';
              ELSE
               rco_int <='0';
            END IF;
         END IF;

END PROCESS;


-- TOGGLE FLIP FLOP TO GENERATE THE PWM OUTPUT.

      PROCESS (clock,rco_int,reset)
      BEGIN
                  IF (reset = '1') THEN
                      pwm_int <='0';
              ELSIF rising_edge(rco_int) THEN
                pwm_int <= NOT(pwm_int);
              ELSE
                pwm_int <= pwm_int;
            END IF;
      END PROCESS;
      pwm <= pwm_int;

END arch_pwm;
```

## Generador Secuencia

El funcionamiento de este módulo es básicamente similar al de un decodificador al que le ingresan dos líneas codificadas en binario y sale un uno por la línea correspondiente a dicho código.

### Tabla de Verdad:

| Secuencia IN | Secuencia OUT 0 | Secuencia OUT 1 | Secuencia OUT 2 |
|---|---|---|---|
| 00 | 0 | 0 | 0 |
| 01 | 1 | 0 | 0 |
| 10 | 0 | 1 | 0 |
| 11 | 0 | 0 | 1 |

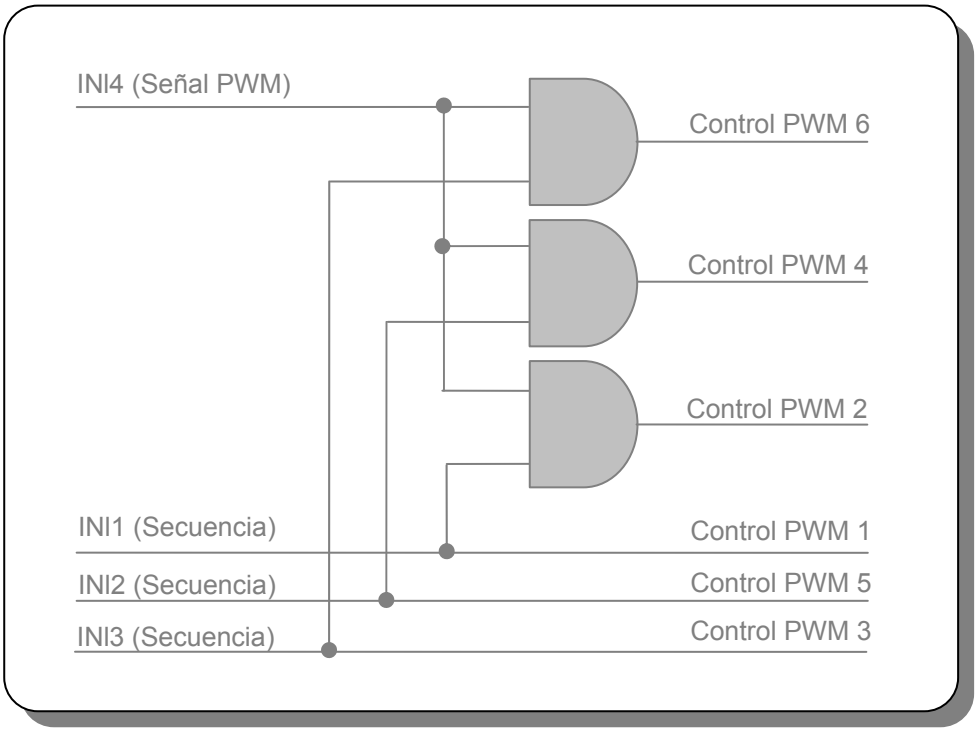
El código de este módulo se detalla a continuación.

```
--************************************************
-- Archivo: Secuenciador.vhd
-- Created : Jul 05, 2009
--************************************************

library IEEE;
USE   ieee.std_logic_1164.all;
USE  ieee.std_logic_arith.all ;
USE   work.user_pkg.all;

ENTITY Secuenciador Is
  PORT (Sec0, Sec1 :in STD_LOGIC;
        SecOut0, SecOut1, SecOut2 :out STD_LOGIC
        );

END Secuenciador;

ARCHITECTURE ArchSecuenciador of Secuenciador IS

BEGIN
  SecOut0 <=  '1' when Sec0='1' and Sec1='0' else '0';
  SecOut1 <=  '1' when Sec0='0' and Sec1='1' else '0';
  SecOut2 <=  '1' when Sec0='1' and Sec1='1' else '0';

END ArchSecuenciador;
```

## Generador de Señales de Control de Transistores

El comportamiento de este módulo es muy sencillo, recibe la señal modulada en PWM y las señales de secuencia generadas por los módulos anteriores. Y saca a la salida las señales necesarias para controlar los transistores.

A continuación se muestra un diagrama que describe el funcionamiento del módulo.




El código de este módulo se describe a continuación.

```
--************************************************
-- Archivo: Logica.vhd
-- Created : Jul 05, 2009
--************************************************

library IEEE;
USE   ieee.std_logic_1164.all;
USE  ieee.std_logic_arith.all ;
USE   work.user_pkg.all;

ENTITY Logica IS
PORT (  INl1,INl2,INl3,INl4                :in  STD_LOGIC;
        OUT1,OUT2,OUT3,OUT4,OUT5,OUT6           :out STD_LOGIC
     );
END Logica;

ARCHITECTURE arch_logica OF Logica IS

BEGIN
OUT1<= INl1;
OUT3<= INl3;
OUT5<= INl2;

OUT2<= INl3 and INl4;
OUT4<= INl2 and INl4;
OUT6<= INl1 and INl4;
END arch_logica;
```

## Modulo de Control Pwm del Motor

Por último se han interconectado los módulos anteriormente descriptos tal como se muestra en el “Planteo de la Solución” formando una única entidad denominada “MotorControl”.

El código descriptivo de esta entidad es el siguiente.

```
--*********************************************
-- Archivo: MotorControl.vhd
-- Created : Jul 05, 2009
--*********************************************

library IEEE;
USE   ieee.std_logic_1164.all;
USE  ieee.std_logic_arith.all ;
USE   work.user_pkg.all;

--************************************
-- Declaro la entidad de Mayor jerarquia
--************************************

ENTITY MotorControl Is
  PORT (DutyControl :in STD_LOGIC_VECTOR (7 downto 0);
        Secuencia0, Secuencia1 :in STD_LOGIC;
        ClockPwm, ResetPwm :in  STD_LOGIC;
        Pwm1,Pwm2,Pwm3,Pwm4,Pwm5,Pwm6 :out STD_LOGIC
        );

END MotorControl;

      --***************************************
      -- Declaro el comportamiento de la entidad
      --***************************************

ARCHITECTURE ArchMotorControl of MotorControl IS

--************************************
-- Defino que componentes voy a utilizar
--************************************
COMPONENT Secuenciador Is
  PORT (Sec0, Sec1 :in STD_LOGIC;
        SecOut0, SecOut1, SecOut2 :out STD_LOGIC
        );
END COMPONENT;

COMPONENT Logica IS
PORT (  INl1,INl2,INl3,INl4               :in  STD_LOGIC;
        OUT1,OUT2,OUT3,OUT4,OUT5,OUT6           :out STD_LOGIC
     );
END COMPONENT;

COMPONENT pwm_fpga
PORT (
      clock: in std_logic;
      reset: in std_logic;
      data_value: in std_logic_vector(7 downto 0);
      pwm: out std_logic
);
END COMPONENT;

--************************************
-- Declaracion de Señales Internas
--************************************
```

```
SIGNAL LinePwm, Secuence0, Secuence1, Secuence2: STD_LOGIC;
SIGNAL SDutyControl : STD_LOGIC_VECTOR (7 downto 0);
SIGNAL SSecuencia : STD_LOGIC_VECTOR (1 downto 0);
SIGNAL SClockPwm, SResetPwm :  STD_LOGIC;
SIGNAL SPwm1,SPwm2,SPwm3,SPwm4,SPwm5,SPwm6 : STD_LOGIC;

BEGIN

--************************************
-- Describo interconexion de los bloques
--************************************
BloquePwm : pwm_fpga
  PORT MAP
    (
    clock => ClockPwm,
    reset => ResetPwm,
    data_value => DutyControl,
    pwm => LinePwm
    );

BloqueSecuenciador : Secuenciador
  PORT MAP
    (
    Sec0 => Secuencia0,
    Sec1 => Secuencia1,
    SecOut0 => Secuence0,
    SecOut1 => Secuence1,
    SecOut2 => Secuence2
    );

BloqueLogica : Logica
  PORT MAP
    (
    INl1 => Secuence0,
    INl2 => Secuence1,
    INl3 => Secuence2,
    INl4 => LinePwm,
    OUT1 => Pwm1,
    OUT2 => Pwm2,
    OUT3 => Pwm3,
    OUT4 => Pwm4,
    OUT5 => Pwm5,
    OUT6 => Pwm6
    );

END ArchMotorControl;
```

Luego se ha desarrollado un Test Bench o Banco de Pruebas para verificar el correcto funcionamiento de esta entidad.

El test bench genera tanto las señales de clock para el módulo pwm como las señales de que gobiernan la duración del ciclo útil de la señal pwm y la secuencia en la que deben activarse los transistores. Tengamos presente que esta última señal usualmente proviene de un sensor vinculado a la posición del eje del rotor.

A continuación se describe el código referido al Test Bench.

```
--************************************
--      Test Bench Secuenciador
--************************************

LIBRARY ieee;

use ieee.std_logic_1164.all;

--****************************************
-- Declaracion de la entidad de Test Bench
--****************************************

ENTITY MotorControl_test_bench IS

END MotorControl_test_bench;

--**********************************************
-- Descripcion del comportamiento del test bench
--**********************************************

ARCHITECTURE arch_test_bench OF MotorControl_test_bench IS

COMPONENT MotorControl Is
  PORT (DutyControl :in STD_LOGIC_VECTOR (7 downto 0);
        Secuencia0, Secuencia1 :in STD_LOGIC;
        ClockPwm, ResetPwm :in  STD_LOGIC;
        Pwm1,Pwm2,Pwm3,Pwm4,Pwm5,Pwm6 :out STD_LOGIC
        );

END COMPONENT;

--*********************************
-- Declarcion de señales internas
--*********************************

  signal sig_reset: STD_LOGIC;
  signal sig_Data_value: STD_LOGIC_VECTOR(7 downto 0) := "00011001";
  signal sig_clock: STD_LOGIC;
  signal one : STD_LOGIC := '1';
  signal zero : STD_LOGIC := '0';

  signal SigSec0, SigSec1: STD_LOGIC;
  signal Sig_Pwm1,Sig_Pwm2,Sig_Pwm3,Sig_Pwm4,Sig_Pwm5,Sig_Pwm6 : STD_LOGIC;

  shared variable ENDSIM: boolean:=false;
  constant clk_period:TIME:=200 ns;


BEGIN

--************************************
-- Descripcion de Ports
--************************************

  Bloque : MotorControl
  PORT MAP
  (
  DutyControl => sig_Data_value,
  Secuencia0 => SigSec0,
  Secuencia1 => SigSec1,
  ClockPwm => sig_clock,
  ResetPwm => sig_reset,
  Pwm1 => Sig_Pwm1,
  Pwm2 => Sig_Pwm2,
  Pwm3 => Sig_Pwm3,
  Pwm4 => Sig_Pwm4,
```

```
  Pwm5 => Sig_Pwm5,
  Pwm6 => Sig_Pwm6
  );
--**********************************
-- Generacion de la señal de clock
--**********************************

  clk_gen: PROCESS

      BEGIN

            IF ENDSIM = FALSE THEN
                  sig_clock <= '1';
                  wait for clk_period/2;
                  sig_clock <= '0';
                  wait for clk_period/2;
            ELSE
                  wait;
            END IF;

      END PROCESS;

--**************************************
-- Generacion de Secuencia de conmutacion
--**************************************

  SecuenceGenerator: PROCESS

  BEGIN
    sig_reset <= '1';
    SigSec0 <= '0';
    SigSec1 <= '0';
    wait for 20 us;
    sig_reset <= '0';

  Bucle: LOOP
    SigSec0 <= '1';
    SigSec1 <= '0';
    wait for 150 us;
    SigSec0 <= '0';
    SigSec1 <= '1';
    wait for 150 us;
    SigSec0 <= '1';
    SigSec1 <= '1';
    wait for 150 us;
    SigSec0 <= '0';
    SigSec1 <= '0';

    CASE sig_Data_value IS
      WHEN "00011001" => sig_Data_value<="01000000";
      WHEN "01000000" => sig_Data_value<="10000000";
      WHEN "10000000" => sig_Data_value<="11000000";
      WHEN "11000000" => sig_Data_value<="11100110";
      WHEN "11100110" => sig_Data_value<="00011001";
      WHEN OTHERS     => sig_Data_value<="00011001";
    END CASE;

    --IF (sig_Data_value = "01000000") THEN
      --sig_Data_value <= "11000000";
    --ELSE
      --sig_Data_value <= "01000000";
    --END IF;
  END LOOP Bucle;
    wait;
    END PROCESS SecuenceGenerator;
END arch_test_bench;
```

## Simulación

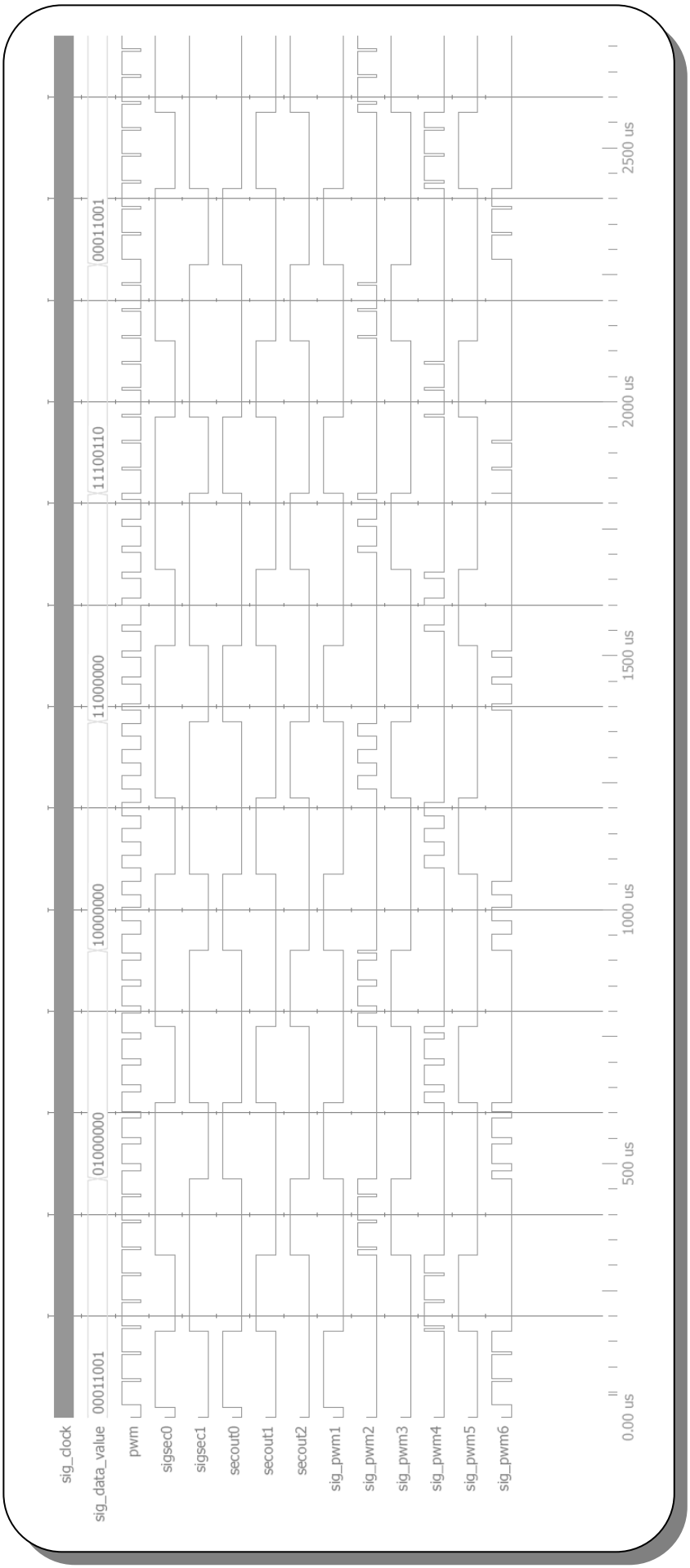
Para realizar la simulación se ha empleado el ModelSim PE Student Edition 6.5b. Esta herramienta de software nos permite visualizar gráficos de las distintas señales en el tiempo.

Para ello hemos generado un nuevo proyecto en el que incluimos cada uno de los archivos desarrolladazos anteriormente incluido el Test Bench.
No lo hemos mencionado antes por este último a además de generar la señales de clock y secuencia hace un barrido de diferentes ciclos de actividad para la señal de pwm. Dicho barrido variando al ciclo de actividad en la siguiente secuencia: 10%, 25%, 50%, 75% y 90%.

Se han configurado para visualizar en las gráficas las siguientes señales:

- Clock (sig_clock,).
- Control del ciclo útil (sid_data_value).
- Pwm (pwm).
- Secuencia de conmutación de entrada (sigsec0 y sigsec1).
- Secuencia de activación de transistores (secout0, secout1 y secout2).
- Salida de control pwm (sig_pwm1 a sig_pwm6).

El resultado de esta simulación se detalla en la figura siguiente.


sig_clock
sig_data_value
00011001
01000000
10000000
11000000
11100110
00011001
pwm
sigsec0
sigsec1
secout0
secout1
secout2
sig_pwm1
sig_pwm2
sig_pwm3
sig_pwm4
sig_pwm5
sig_pwm6
0.00 us
500 us
1000 us
1500 us
2000 us
2500 us