Universidad Tecnológica Nacional
Facultad Regional Buenos Aires
Dto. Electrónica

Materia: Robótica

Código: 95-0482 **Curso:** R6055

Profesor: Ing. Hernán Giannetta

Trabajo Práctico N°: 2 Fecha: 28/06/2009

Título: Análisis Dinámico de un Robot e implementación en FPGA

Alumnos:

Canale José Leg : 115148-4

Llanos Pablo Leg : 115589-1

Sánchez Martín Leg : 112687-8

Fecha entrega: 15/07/2009

Observaciones:

Análisis Dinámico de Robot Five Bar Linkage e implementación de Driver de control de Motores en FPGA

Índice

1- Introducción

La dinámica se ocupa de las relaciones entre las fuerzas que actúan sobre un cuerpo y el movimiento que en el se origina. Por lo tanto, el modelo dinámico de un robot tiene por objetivo conocer la relación entre el movimiento del robot y las fuerzas implicadas en el mismo.

Esta relación se tiene mediante el denominado **modelo dinámico**, que establece la relación matemática entre:

1. La localización de robot definida por sus variables articulares o por las coordenadas de localización de su extremo, y sus derivadas: velocidad y aceleración.
2. Las fuerzas y pares aplicados en las articulaciones (o en el extremo del robot).
3. Los parámetros dimensionales del robot, como longitud, masas e inercias de sus elementos.

La obtención de este modelo para mecanismos de uno o dos grados de libertad no es excesivamente compleja, pero a medida que el número de grados de libertad aumenta, El planteamiento y obtención del modelo dinámico se complica enormemente. Por este motivo no siempre es posible obtener un modelo dinámico expresado de una forma cerrada, esto es, mediante una serie de ecuaciones, normalmente de tipo diferencial de 2º orden, cuya integración permita conocer qué movimiento surge al aplicar unas fuerzas o qué fuerzas hay que aplicar para obtener un movimiento determinado. El modelo dinámico debe ser resuelto entonces de manera iterativa mediante la utilización de un procedimiento numérico.

El problema de la obtención del modelo dinámico de un robot es, por lo tanto, uno de los aspectos más complejos de la robótica, lo que ha llevado a ser obviado en numerosas ocasiones. Sin embargo el modelo es imprescindible para conseguir los siguientes fines:

1. Simulación del movimiento del robot.
2. Diseño y evaluación de la estructura mecánica del robot.
3. Dimensionamiento de los actuadores.
4. Diseño y evaluación del control dinámico del robot.

Este último fin es evidentemente de gran importancia, pues de la calidad del control dinámico del robot depende la precisión y velocidad de sus movimientos. La gran complejidad ya comentada existente en la obtención del modelo dinámico del robot, ha motivado que se realicen ciertas simplificaciones, de manera que así pueda ser utilizado en el diseño del controlador, sino también en línea con el control, cuando así lo requiera la técnica de control empleada.

Es importante hacer notar que el modelo dinámico completo de un robot debe incluir no sólo la dinámica de sus elementos (barras o eslabones) sino también la propia de sus sistemas de transmisión, de los actuadores y sus equipos electrónicos de mando. Estos elementos incorporan al modelo dinámico nuevas inercias, rozamientos, saturaciones de los circuitos electrónicos, etc. aumentando aún más su complejidad.

Por último, es preciso señalar que si bien en la mayor parte de las aplicaciones reales de la robótica, las cargas e inercias manejadas no son suficientes como para originar deformaciones en los eslabones del robot, en determinadas ocasiones no ocurre así, siendo preciso considerar al robot como un conjunto de eslabones no rígidos. Aplicaciones de este tipo pueden encontrarse en la robótica espacial o en robots de grandes dimensiones, entre otras.

La obtención del modelo dinámico de un robot ha sido y es objeto de estudio e investigación. Numerosos investigadores han desarrollado formulaciones alternativas, basadas fundamentalmente en la mecánica Newtoniana y Lagrangiana, con el objeto de obtener modelos manejables para los sistemas de calculo de una manera mas eficiente. La obtención de modelos mediante formulación Lagrangiana muestra ventajas frente a la formulación Newtoniana a medida que aumenta el número de grados de libertad.

En el presente informe se obtendrá el modelo dinámico siguiendo la formulación lagrangiana.

2- Análisis Dinámico

Como se menciono en párrafos anteriores, el objetivo del análisis dinámico es encontrar la relación entre el movimiento del robot y las fuerzas implicadas en el mismo. La ecuación que se debe obtener es la siguiente:

$$\tau = M \times \ddot{q} + H + C$$

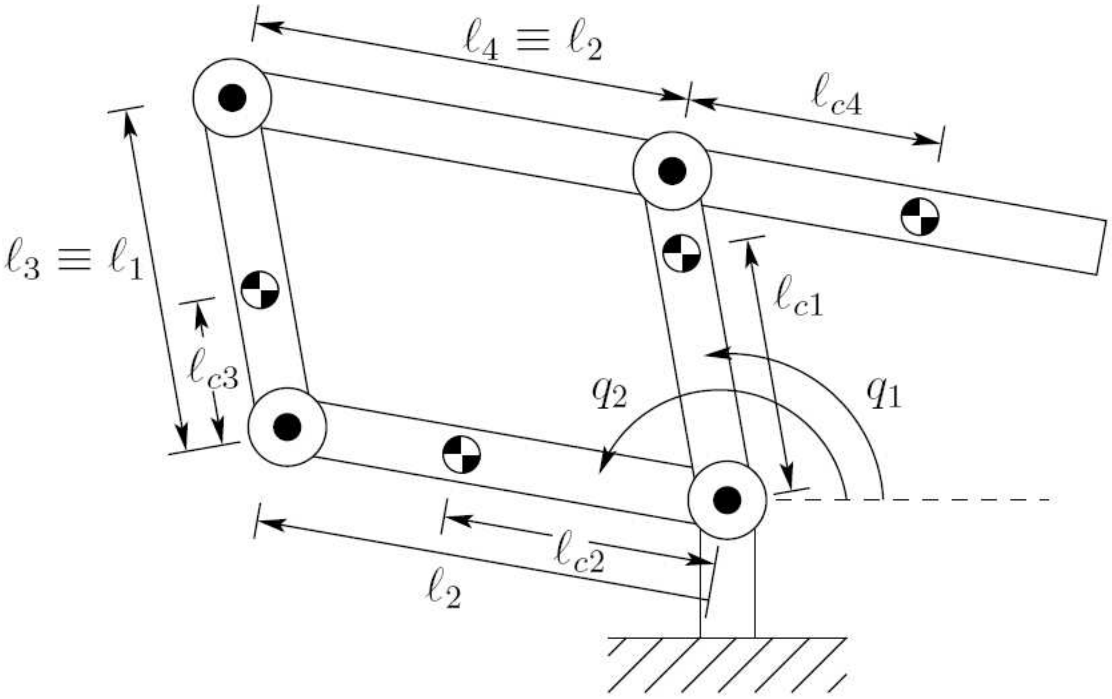
Donde:

τ : Vector de fuerzas y pares motores efectivos aplicados sobre cada coordenada q_i.

M : Matriz de Inercias.

H : Matriz columna de fuerzas de Ciriolis y Centripeta.

C : Matriz columna de fuerzas de gravedad.



Desglosando la ecuación matricial quedan las siguientes 2 ecuaciones:

$$T_1 = (M_{11} + I_{h_1}).\ddot{q}_1 + M_{12}.\ddot{q}_2 + \frac{\partial M_{12}}{\partial q_2}.\ddot{q}_2^2 + g.(m_1 l_{c1} + m_3.l_{c3} + m_4.l_1).\cos q_1 \quad (1)$$

$$T_2 = (M_{22} + I_{h_2}).\ddot{q}_2 + M_{12}.\ddot{q}_1 + \frac{\partial M_{12}}{\partial q_1}.\ddot{q}_1^2 + g.(m_2 l_{c2} + m_3.l_2 - m_4.l_{c4}).\cos q_2 \quad (2)$$

Donde:

I_h : Incluye el momento de inercia y rozamientos del Motor
g : Constante gravitacional

Matriz de inercia:

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

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

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

La ventaja de esta configuración es que si: $m_3.l_2.l_{c3} = m_4.l_1.l_{c4}$ entonces la matriz de inercia es diagonal ($M_{12}=M_{21}=0$) y constante. Como consecuencia las ecuaciones dinámicas no contendrán ni las fuerzas de Coriolis ni las fuerzas centrífugas.

El último termino de las ecuaciones (1) y (2) corresponde a la matriz de gravedad, por lo tanto depende de la constante gravitacional.
De la ecuación (2) se puede notar que si en el diseño se hace cumplir que:
$m_2 l_{c2} + m_3.l_2 = m_4.l_{c4}$ el sistema tendrá la propiedad de que el motor 2 no tendrá carga gravitacional.

Teniendo en cuenta las consideraciones realizadas el Modelo dinámico queda de la siguiente manera:

$$T_1 = (m_1.l_{c1}^2 + m_3.l_{c3}^2 + m_4.l_1^2 + I_{h_1}).\ddot{q}_1 + g.(m_1 l_{c1} + m_3.l_{c3} + m_4.l_1).\cos q_1$$

$$T_2 = (m_2.l_{c2}^2 + m_3.l_2^2 + m_4.l_{c4}^2 + I_{h_2}).\ddot{q}_2 + g.(m_2 l_{c2} + m_3.l_2 - m_4.l_{c4}).\cos q_2$$

Como se puede ver en las ecuaciones del modelo dinámico resulta que q_1 solo depende de T_1 y q_2 solo de T_2 (considerando que los demás parámetros son constantes que surgen de cada diseño en particular). Esta es la razón por la cual la configuración paralelogramo, si se respetan las condiciones mencionadas, es tan popular en los robots industriales. Ya que se puede operar q_1 y q_2 independientemente, sin preocuparse por las interacciones entre los dos ángulos.

3- Sistema de Control para Motor

El la figura siguiente se muestra el diagrama en bloques del sistema que se debe implementar para controlar del movimiento angular de cada articulación del robot.

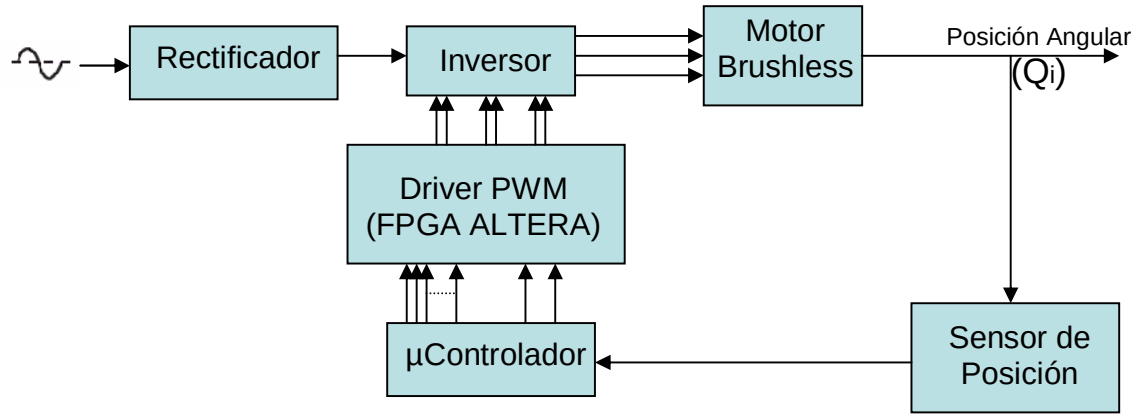
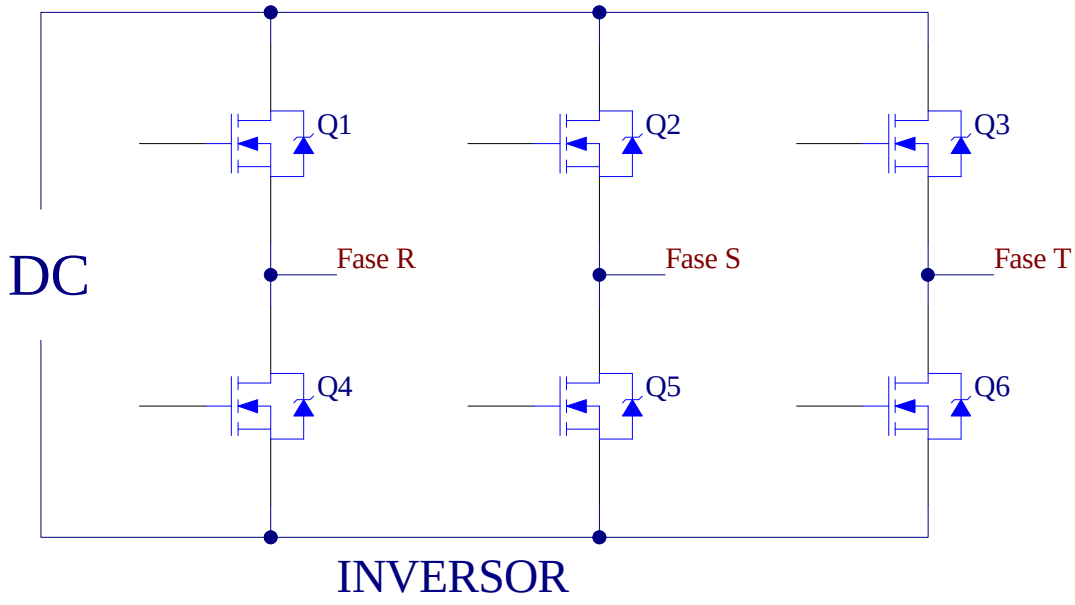


Diagrama en bloques del control de Posición a implementar

El driver PWM que se indica en la figura se implementa con un FPGA del fabricante Altera, el mismo internamente cuenta con 3 moduladores de ancho de pulso (uno por cada fase) que generan las señales de excitación de los 6 transistores de potencia que posee el inversor. En la figura siguiente se muestra la topología del inversor para el cual se diseña el driver.



En driver posee 10 entradas de control: 1- Entrada de Reset 2- Entrada de Inversión de secuencia (inversión de Marcha) y 3- Las 8 entradas restantes conforman un bus por medio del cual se configura el valor del ciclo de actividad de los pulsos.

El FPGA esta programado para generar **3 señales de 50Hz desfasadas 120º** entre si, y a la vez realiza variaciones del ciclo de actividad de forma **trapezoidal**. La pendiente de crecimiento es constante, por lo tanto, si se desea variar el ciclo de actividad desde 0% hasta 50% se tardara la mitad de tiempo de la requerida para llegar hasta 100%. A fines demostrativos en las simulaciones se redujo este tiempo a aproximadamente 120 ms para lograr una mejor visualización. En la práctica estaría en el orden de los segundos.

El µcontrolador utilizado realiza un control dinámico sobre la velocidad del motor, ya que variando el ancho de los pulsos varia el valor eficaz de la armónica fundamental de la señal generada. Las principales ventajas de utilizar este tipo de modulación es que reduce el contenido armónico y a la vez controla el valor eficaz de la señal generada.

A continuación de muestra el codigo en VHDL utilizado para generar el Driver con las características mencionadas, el primer código corresponde al Testbench (archivo utilizado para simular el funcionamiento del FPGA, lo que hace básicamente es generar la señal de clock, carga ciclo de actividad de 100% a secuencia directa y activa el modulo por 100ms, luego resetea por 10ms y vuelve a arrancar el modulo pero en secuencia inversa otros 100ms), 3 códigos idénticos correspondientes a cada Modulador y 1 código que controla el funcionamiento de los Moduladores y a su vez realiza la interfase con el mundo exterior.

4- Implementación de Driver PWM en VHDL

```
--*************************************************************************--
--       Archivo Testbench utilizado para similar el funcionamiento del Modulo PWM      --
--*************************************************************************--

LIBRARY ieee;

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

ENTITY pwm_fpga_test_bench IS
END pwm_fpga_test_bench;


ARCHITECTURE arch_test_bench OF pwm_fpga_test_bench IS

COMPONENT MODULO_PWM
  PORT (  Data_value_M                  :in  std_logic_vector(7 downto 0);
            clock_M,reset_M :in  STD_LOGIC;
            inversion_marcha :in STD_LOGIC;
            pwm_1_M   :out STD_LOGIC;
            pwm_2_M             :out STD_LOGIC;
            pwm_3_M             :out STD_LOGIC;
            out_T4        :out STD_LOGIC;
            out_T5        :out STD_LOGIC;
            out_T6        :out STD_LOGIC;
            clock_motor     :out STD_LOGIC
          );
END COMPONENT;

-- Internal signal declaration

SIGNAL sig_clock        : std_logic;
SIGNAL sig_reset        : std_logic;
SIGNAL pwm_Q1           : std_logic;
SIGNAL pwm_Q2           : std_logic;
SIGNAL pwm_Q3           : std_logic;
SIGNAL out_Q4           : std_logic;
SIGNAL out_Q5           : std_logic;
SIGNAL out_Q6           : std_logic;
SIGNAL clock_motor      : std_logic;
SIGNAL flag_fin_ciclo   : std_logic;
SIGNAL sig_inversion_marcha   : std_logic;
SIGNAL sig_data_value  : std_logic_vector(7 downto 0);
SIGNAL aux          : std_logic_vector(7 downto 0);
```

```
constant clk_period   :TIME:=2600 ns;
shared variable ENDSIM  : boolean:=false;

BEGIN

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;

-- Instantiating top level design Component pwm_fpga1

inst_MODULO_PWM : MODULO_PWM
        PORT MAP(
                clock_M         => sig_clock,
                reset_M         => sig_reset,
                data_value_M    => sig_data_value,
                pwm_1_M         => pwm_Q1,
                pwm_2_M         => pwm_Q2,
                pwm_3_M         => pwm_Q3,
                out_T4 => out_Q4,
                out_T5  => out_Q5,
                out_T6  => out_Q6,
                clock_motor => clock_motor,
                inversion_marcha => sig_inversion_marcha
                );

stimulus_process: PROCESS
  BEGIN
          sig_reset <= '1';
          sig_inversion_marcha <= '0';   -- secuencia directa de fases
          sig_data_value <= "11111111";
          sig_reset <= '0';         -- comienza a funcionar el modulo PWM en secuencia directa
          wait for 100 ms;          -- espera durante 100ms

          sig_reset <= '1';         Mantiene reseteado el modulo
          wait for 10 ms;           -- espera durante 10ms

          sig_inversion_marcha <= '1';  -- secuencia inversa de fases
          sig_data_value <= "11111111";
          sig_reset <= '0';         -- comienza a funcionar el modulo PWM en secuencia inversa
```

```
            wait for 100 ms;        -- espera durante 100ms
            sig_reset <= '1';       resetea y finaliza la simulacion

    wait;
END PROCESS stimulus_process;

END arch_test_bench;

--********************************************************************************--



--********************************************************************************--
--    Código para programar FPGA :                                                --
--                    - controla el funcionamiento de los 3 Moduladores           --
--                    - realiza la interface con el mundo exterior                --
--********************************************************************************--

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

-- Definicion de pin out del Modulo PWM

ENTITY Modulo_PWM IS
PORT (  clock_M,reset_M     :in STD_LOGIC;
            inversion_marcha    :in STD_LOGIC;
            Data_value_M        :in std_logic_vector(7 downto 0);
            pwm_1_M             :out STD_LOGIC;
            pwm_2_M             :out STD_LOGIC;
            pwm_3_M             :out STD_LOGIC;
            out_T4          :out STD_LOGIC;
            out_T5          :out STD_LOGIC;
            out_T6          :out STD_LOGIC;
            clock_motor     :out STD_LOGIC
    );

END MODULO_PWM;

-- Definicion de la arquitectura interna del Modulo PWM

ARCHITECTURE ARCH_PWM OF MODULO_PWM IS
```

```
COMPONENT pwm_fpga1
PORT (
        clock         : in std_logic;
        reset_1       : in std_logic;
        data_value    : in std_logic_vector(7 downto 0);
        pwm_1         : out std_logic;        -- salida del PWM 1
        flag_1        : out std_logic
);
END COMPONENT;

COMPONENT pwm_fpga2
PORT (
        clock         : in std_logic;
        reset_2       : in std_logic;
        data_value    : in std_logic_vector(7 downto 0);
        pwm_2         : out std_logic;        -- salida del PWM 2
        flag_2        : out std_logic
);
END COMPONENT;

COMPONENT pwm_fpga3
PORT (
        clock         : in std_logic;
        reset_3       : in std_logic;
        data_value    : in std_logic_vector(7 downto 0);
        pwm_3         : out std_logic;        -- salida del PWM 3
        flag_3        : out std_logic
);
END COMPONENT;

-- Deinicion de señales internas

SIGNAL señal_reset_1        : std_logic;
SIGNAL señal_reset_2        : std_logic;
SIGNAL señal_reset_3        : std_logic;
SIGNAL aux_reset_2          : std_logic;
SIGNAL aux_reset_3          : std_logic;
SIGNAL salida_T4        : std_logic;
SIGNAL salida_T5        : std_logic;
SIGNAL salida_T6        : std_logic;
SIGNAL señal_flag_1     : std_logic := '0';
SIGNAL señal_flag_2     : std_logic := '0';
SIGNAL señal_flag_3     : std_logic := '0';
SIGNAL flag_fin_ciclo   : std_logic;
```

```
SIGNAL flag_frecuencia  : std_logic;
SIGNAL pwm_int, rco_int    : std_logic;
SIGNAL data_value    : std_logic_vector(7 downto 0);
SIGNAL aux         : std_logic_vector(7 downto 0);
SIGNAL cnt_out_int          : std_logic_vector(7 downto 0);
SIGNAL pend_rampa   : NATURAL:=0;
constant clk_period      :TIME:=100 ns;
constant frecuencia_motor :TIME:=20 ms;

BEGIN

-- Instantiating top level design Component pwm_fpga1

inst_pwm_fpga1 : pwm_fpga1
PORT MAP(
                clock   => clock_M,
                reset_1=> señal_reset_1,
                data_value      => data_value,
                pwm_1           => pwm_1_M,
                flag_1 => señal_flag_1
        );

-- Instantiating top level design Component pwm_fpga2

inst_pwm_fpga2 : pwm_fpga2
PORT MAP(
                clock   => clock_M,
                reset_2=> señal_reset_2,
                data_value      => data_value,
                pwm_2           => pwm_2_M,
                flag_2 => señal_flag_2
        );

-- Instantiating top level design Component pwm_fpga3

inst_pwm_fpga3 : pwm_fpga3
PORT MAP(
                clock   => clock_M,
                reset_3=> señal_reset_3,
                data_value      => data_value,
                pwm_3           => pwm_3_M,
                flag_3 => señal_flag_3
        );
```

```
-- Rutina para generar flag indicador de final de ciclo

PROCESS (flag_fin_ciclo,señal_flag_1,señal_flag_2,señal_flag_3)
  BEGIN
    flag_fin_ciclo <= señal_flag_1 or señal_flag_2 or señal_flag_3;
END PROCESS;


-- rutina de seguimiento del valor de set point (data_value_M) de forma gradual. Ciclo de
actividad = data_value_M / 255
-- El tiempo de crecimiento del ciclo de actividad hasta su valor final es : data_value_M *
Tclock * (N+1)

PROCESS
(reset_M,señal_reset_1,señal_reset_2,señal_reset_3,data_value_M,flag_fin_ciclo)
  BEGIN
    IF(reset_M = '0') THEN
          IF(rising_edge(flag_fin_ciclo)) THEN
                IF(pend_rampa = 0) THEN          -- Cantidad de ciclos (N+1) de clock luego de
los cuales se incrementa el ciclo de actividad en 1/255
                      IF(data_value < data_value_M) THEN
                            data_value <= INC(data_value);
                      ELSIF(data_value > data_value_M) THEN
                            data_value <= DEC(data_value);
                      END IF;
                      pend_rampa <= 0;
                ELSE
                       pend_rampa <= pend_rampa + 1;
                END IF;
          END IF;
    ELSE
          data_value <= "00000001"; --data_value_M;
    END IF;
END PROCESS;


-- Activacion de los distintos modulos PWM con 120º de desfasaje entre si. Frecuencia:
50Hz

PROCESS (flag_frecuencia,reset_M)
  BEGIN
      IF(reset_M = '0') THEN
        IF(rising_edge(flag_frecuencia)) THEN
            IF(señal_reset_1 = '0' and aux_reset_2 = '1' and aux_reset_3 = '1') THEN
```

```
            señal_reset_1 <= '0';
            aux_reset_2 <= '0';
            aux_reset_3 <= '1';
            salida_T4 <= '0';
            salida_T5 <= '0';
            salida_T6 <= '1';
        ELSIF(señal_reset_1 = '0' and aux_reset_2 = '0' and aux_reset_3 = '1') THEN
            señal_reset_1 <= '1';
            aux_reset_2 <= '0';
            aux_reset_3 <= '1';
            salida_T4 <= '1';
            salida_T5 <= '0';
            salida_T6 <= '1';
        ELSIF(señal_reset_1 = '1' and aux_reset_2 = '0' and aux_reset_3 = '1') THEN
            señal_reset_1 <= '1';
            aux_reset_2 <= '0';
            aux_reset_3 <= '0';
            salida_T4 <= '1';
            salida_T5 <= '0';
            salida_T6 <= '0';
        ELSIF(señal_reset_1 = '1' and aux_reset_2 = '0' and aux_reset_3 = '0') THEN
            señal_reset_1 <= '1';
            aux_reset_2 <= '1';
            aux_reset_3 <= '0';
            salida_T4 <= '1';
            salida_T5 <= '1';
            salida_T6 <= '0';
        ELSIF(señal_reset_1 = '1' and aux_reset_2 = '1' and aux_reset_3 = '0') THEN
            señal_reset_1 <= '0';
            aux_reset_2 <= '1';
            aux_reset_3 <= '0';
            salida_T4 <= '0';
            salida_T5 <= '1';
            salida_T6 <= '0';
        ELSIF(señal_reset_1 = '0' and aux_reset_2 = '1' and aux_reset_3 = '0') THEN
            señal_reset_1 <= '0';
            aux_reset_2 <= '1';
            aux_reset_3 <= '1';
            salida_T4 <= '0';
            salida_T5 <= '1';
            salida_T6 <= '1';
        ELSE
            señal_reset_1 <= '0';
            aux_reset_2 <= '1';
```

```
                aux_reset_3 <= '1';
                salida_T4 <= '0';
                salida_T5 <= '1';
                salida_T6 <= '1';
            END IF;
        END IF;
    ELSE
        señal_reset_1 <= '1';
        aux_reset_2 <= '1';
        aux_reset_3 <= '1';
        salida_T4 <= '0';
        salida_T5 <= '0';
        salida_T6 <= '0';
    END IF;
END PROCESS;

-- Generacion de onda cuadrada de 150Hz

PROCESS
  BEGIN
      flag_frecuencia <= '0';
            wait for frecuencia_motor/12;
                flag_frecuencia <= '1';
                wait for frecuencia_motor/12;
END PROCESS;

PROCESS(inversion_marcha, clock_M)
  BEGIN
        IF(inversion_marcha = '0') THEN
                señal_reset_2  <= aux_reset_2;
                señal_reset_3  <= aux_reset_3;
        ELSE
                señal_reset_2  <= aux_reset_3;
                señal_reset_3  <= aux_reset_2;
        END IF;
END PROCESS;

clock_motor <= flag_frecuencia;

out_T4 <= salida_T4;
out_T5 <= salida_T5;
out_T6 <= salida_T6;

END ARCH_PWM;
```

```
--*********************************************************************--
--                    Código de funcionamiento de  Modulador  1                    --
--*********************************************************************--

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


ENTITY pwm_fpga1 IS
  PORT (  clock,reset_1     :in  STD_LOGIC;             -- señales de entrada (1 bit) externa
          Data_valuE :in  std_logic_vector(7 downto 0); -- señal de entrada (1 byte) externa
          pwm_1             :out STD_LOGIC;             -- señal de salida (1 bit) externa
          flag_1            :out STD_LOGIC
     );
END pwm_fpga1;

ARCHITECTURE arch_pwm OF pwm_fpga1 IS

SIGNAL set_point          : std_logic_vector(7 downto 0);   -- variable (1 byte) interna
SIGNAL reg_out            : std_logic_vector(7 downto 0);   -- variable (1 byte) interna
SIGNAL cnt_out_int        : std_logic_vector(7 downto 0);   -- variable (1 byte) interna
SIGNAL rco_int            : STD_LOGIC;                      -- variable (1 bit) interna
SIGNAL pwm_int            : STD_LOGIC;                      -- variable (1 bit) interna
SIGNAL flag               : 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,set_point,reg_out,reset_1)
  BEGIN
        IF (reset_1 ='1') THEN
                set_point <="00000000";
        ELSIF (rising_edge(clock)) THEN
                set_point <= data_value;
        END IF;
END PROCESS;
```

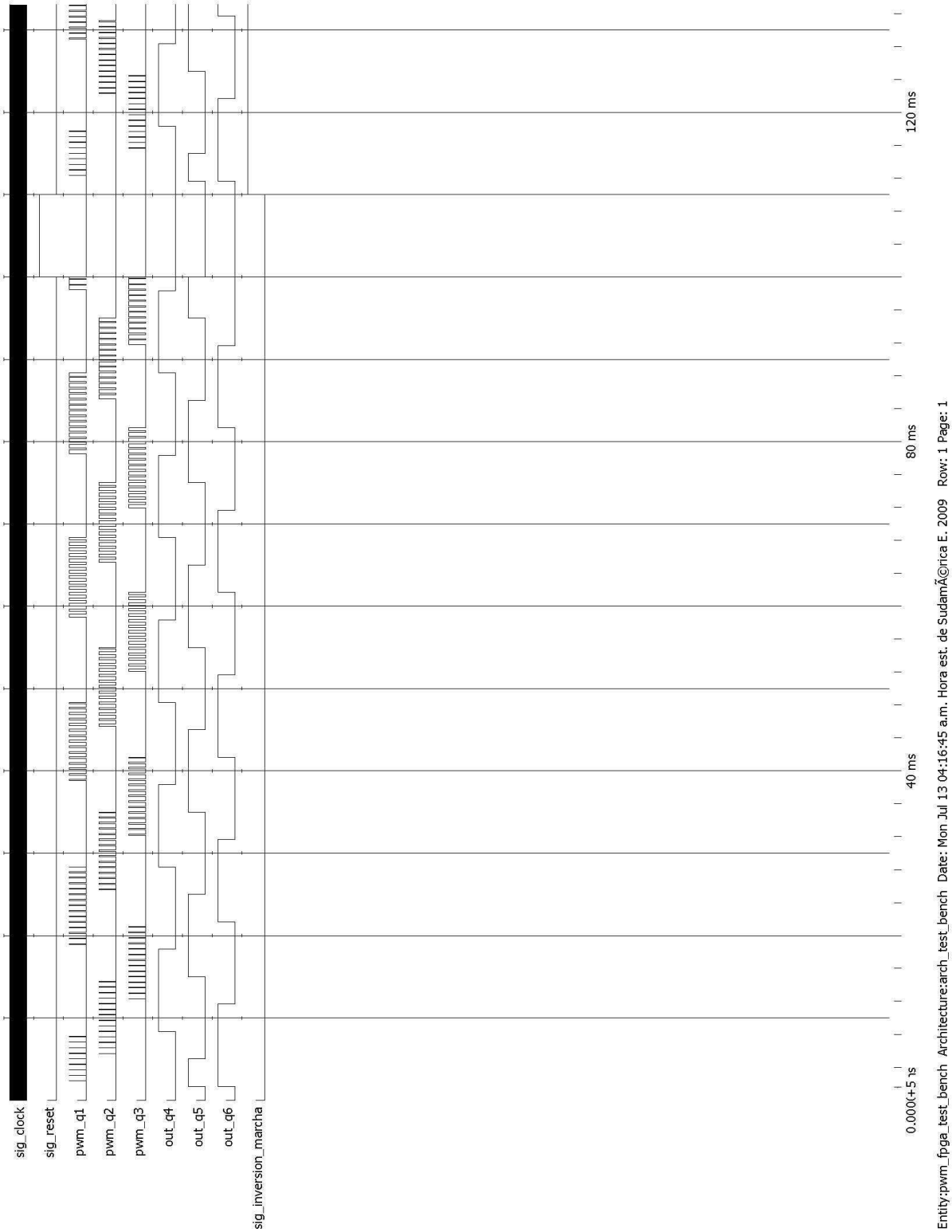
```
 -- 8 BIT UPDN COUNTER. COUNTS UP OR DOWN BASED ON THE PWM_INT
SIGNAL AND GENERATES
 -- TERMINAL COUNT WHENEVER COUNTER REACHES THE MAXIMUM
VALUE OR WHEN IT TRANSISTS
 -- THROUGH ZERO. THE TERMINAL COUNT WILL BE USED AS INTERRUPT TO
AVR FOR GENERATING
 -- THE LOAD SIGNAL.
 -- INC and DEC are the 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,set_point,reg_out)
   BEGIN
        IF(rco_int = '1')  THEN
                cnt_out_int <= set_point;
        ELSIF
                rising_edge(clock) THEN
                IF(rco_int = '0' and pwm_int ='0' and cnt_out_int < "11111111") THEN
                        cnt_out_int <= INC(cnt_out_int);
                ELSE
                     IF (rco_int ='0' and pwm_int ='1' 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_1,set_point)
    BEGIN
        IF(reset_1 ='1') THEN
            rco_int <='1';
            flag <= '0';
        ELSIF(rising_edge(clock)) THEN
                IF ((cnt_out_int = "11111111") or (cnt_out_int ="00000000")) THEN
                        IF(cnt_out_int = "11111111") THEN
                                flag <= '1';
                        END IF;
                        rco_int <= '1';
                ELSE
                        rco_int <= '0';
                        flag <= '0';
                END IF;
```

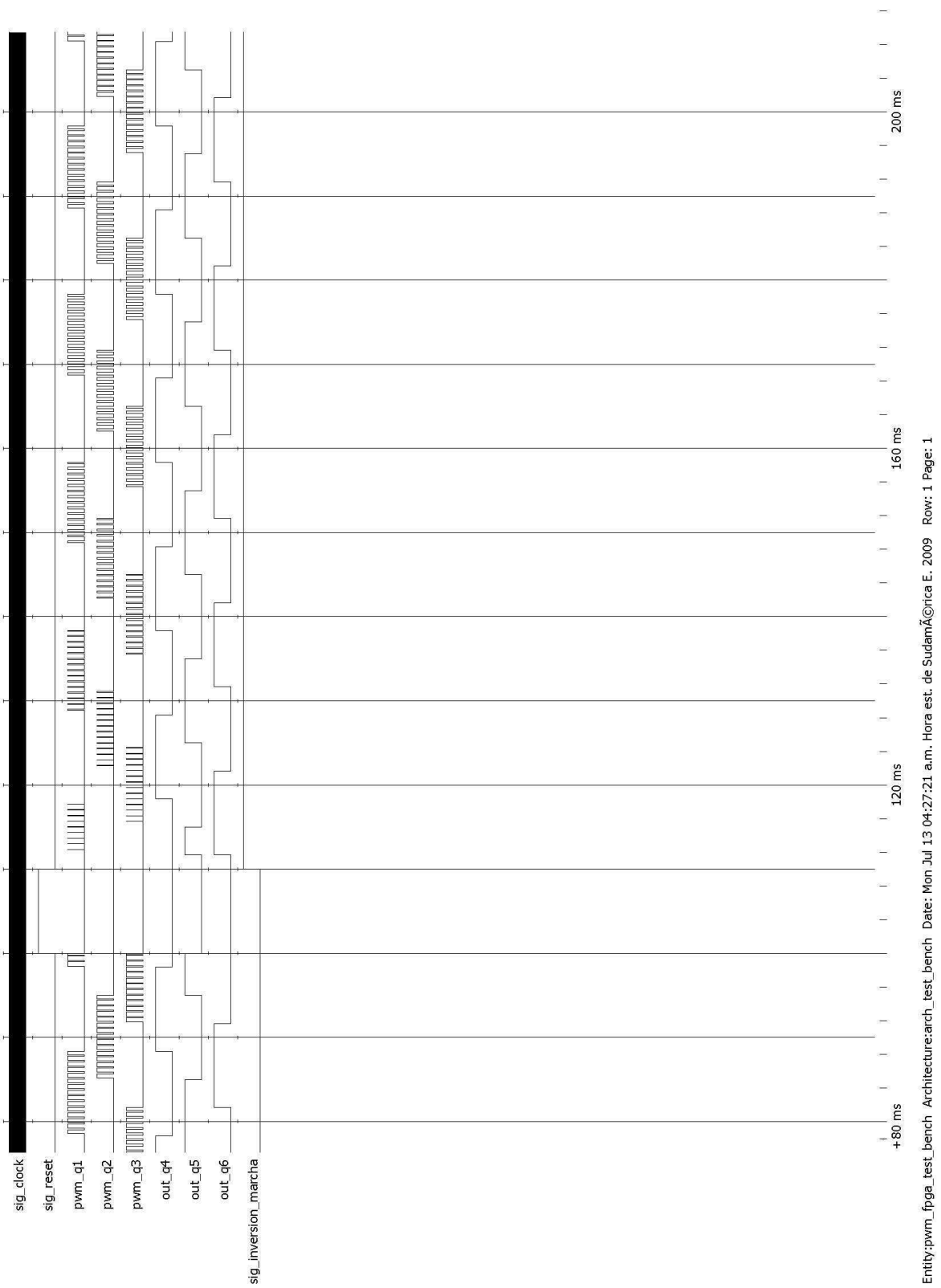
```
        END IF;
END PROCESS;


-- TOGGLE FLIP FLOP TO GENERATE THE PWM OUTPUT.

PROCESS(clock,rco_int,reset_1)
    BEGIN
        IF (reset_1 = '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_1 <= pwm_int;
flag_1 <= flag;

END arch_pwm;
```

Nota: El código para generar los moduladores 2 y 3 es el mismo que el que se utiliza para generar el Modulador 1 con la única diferencia de que las variables están refenciadas con el subíndice 2 y 3 respectivamente.

5- Simulación de Funcionamiento del FPGA

A continuación se muestran los resultados de la simulación, en el primer grafico se muestra una secuencia directa y en el segundo una secuencia inversa. El ciclo de actividad en ambos casos se setea a 100%, debido a la funcionalidad del Modulo PWM el duty va creciendo en forma gradual. Como ya se ha mencionado el tiempo de crecimiento esta seteado para que sea fácil de observar el los gráficos, sin embargo en uso practico ronda en el orden de los segundos.


sig_clock
sig_reset
pwm_q1
pwm_q2
pwm_q3
out_q4
out_q5
out_q6
sig_inversion_marcha
0.000(+5 ns
40 ms
80 ms
120 ms
Entity:pwm_fpga_test_bench Architecture:arch_test_bench Date: Mon Jul 13 04:16:45 a.m. Hora est. de SudamÃ©rica E. 2009 Row: 1 Page: 1


sig_clock
sig_reset
pwm_q1
pwm_q2
pwm_q3
out_q4
out_q5
out_q6
sig_inversion_marcha
+80 ms
120 ms
160 ms
200 ms
Entity:pwm_fpga_test_bench Architecture:arch_test_bench Date: Mon Jul 13 04:27:21 a.m. Hora est. de SudamÃ©rica E. 2009 Row: 1 Page: 1

6- Conclusiones

Los desplazamientos angulares q_1 y q_2 dependen solo de T_1 y T_2 respectivamente, esto hace que el control de movimiento del robot (Five Bar Linkage) pueda ser implementado de manera simplificada una vez que se tiene el Modelo Dinámico ya que ambas variables trabajan de forma independiente. El sistema de control propuesto tiene la posibilidad de controlar el valor eficaz de la señal fundamental variando el duty de los pulsos.

7- Bibliografía y Referencias

Bibliografía:

"Fundamentos de Robótica". 2da Edición ; Barrientos, A.; Peñín,; Balaguer, & Aracil, McGraw Hill, 2007.

Introducción al Lenguaje VHDL ; Miguel Angel Freire Rubio ; Universidad Politécnica de Madrid

Síntesis Cinemática y Dinámica de Mecanismos – Isidro Zabalza Villava – Universidad Publica de Navarra

Referencias:

http://books.google.com.ar/books/Robot analysis

http://www.sciencedirect.com/ ScienceDirect/Mechanism and Machine Theory/Optimized five-bar linkages