Laboratorio 06: Control de un motor de pasos

Por:
Ian Gabriel Cañas Fernández, 1092228

.
.

Control de un motor de pasos
Reutilizando el funcionamiento del teclado matricial para controlar un motor de pasos.

Objetivos:

- Diseñar un sistema digital que permita el control de un motor de pasos.

Procedimiento:
En el presente laboratorio estamos reutilizando tanto el funcionamiento del teclado matricial 4x4 como el funcionamiento del display y todo lo que hemos visto con anterioridad. En este preciso caso estamos utilizando el teclado matricial para introducir los valores de velocidad y ángulo de giro.

Al ser el mismo código del teclado utilizado en el laboratorio 05, se obviará la explicación de este.

El control del motor de pasos

Empezamos generando los puertos, funciones y señales necesarias para el laboratorio:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
use IEEE.std_logic_arith.ALL;
use IEEE.std_logic_unsigned.ALL;


entity Lab06 is
  Port (
        clk: in std_logic; -- Reloj
        FILAS: inout std_logic_vector (3 downto 0);
        COLUMNAS: in std_logic_vector (3 downto 0) := "0000"; -- Posici?n pulsada

        Disp: out std_logic_vector (7 downto 0); -- Vector que establecer? cu?les displays encender
        Num7Seg: out std_logic_vector (6 DOWNTO 0); -- C?digo del n?mero correspondiente al 7 Segmentos

        LEDMODO, LEDCARGA, LEDROTA: out std_logic;

        MOTOR: out std_logic_vector (3 downto 0) := "0011"
        );
end Lab06;
```

```
constant NumMax: INTEGER := 49999;
constant NumMax2: INTEGER := 4999999;

signal conteo: INTEGER range 0 to NumMax;
signal conteo2: INTEGER;

signal clck_state: std_logic; -- Permite el cambio de display.
signal clck_state100: std_logic;

------------------------------------------------------------

signal Cambio: std_logic_vector(2 downto 0) := "000";
signal Cambio2: std_logic_vector(2 downto 0) := "000";
signal digit: std_logic_vector (3 downto 0);
signal Velocidad, Angle: integer := 5;

signal unidad, decena, centena, miles, diezmiles, cienmiles, millones, diezmillones: std_logic_vector (3 downto 0) := "0000";
signal DigUnidad, DigDecena, DigCentena, DigMiles, DigDiezmiles, DigCienmiles, DigMillones, DigDiezmillones: integer range 0 to 9;
signal DigUnidad1, DigDecena1, DigCentena1, DigMiles1, DigDiezmiles1, DigCienmiles1, DigMillones1, DigDiezmillones1: integer range 0 to 9;


signal Digitos1, Digitos2, DigitosRes: integer;

signal boton_pres : std_logic_vector(3 downto 0) := (others => '0');
signal IND, IND_S        : std_logic := '0';
signal MODO: STD_LOGIC;
signal carga: std_logic;
signal rotando: std_logic;
signal veces: integer;
```

```
  function BcdTo7Seg (
      BCD: in std_logic_vector(3 downto 0)) -- C?digo b
      return std_logic_vector is

      variable Cod7Seg: std_logic_vector (6 downto 0);

      begin
      with BCD select -- Selecci?n de la combinaci?n de
          Cod7Seg :=    ("0000001") WHEN ("0000"),
              ("1001111") WHEN ("0001"),
              ("0010010") WHEN ("0010"),
              ("0000110") WHEN ("0011"),
              ("1001100") WHEN ("0100"),
              ("0100100") WHEN ("0101"),
              ("0100000") WHEN ("0110"),
              ("0001111") WHEN ("0111"),
              ("0000000") WHEN ("1000"),
              ("0000100") WHEN ("1001"),
              ("0001000") WHEN ("1010"),
              ("1100000") WHEN ("1011"),
              ("0110001") WHEN ("1100"),
              ("1000010") WHEN ("1101"),
              ("0110000") WHEN ("1110"),
              ("1111111") WHEN OTHERS;
      return Cod7Seg; -- Devoluci?n del respectivo valo
  end function;
```

```
function IntToBin (
    Int: in integer) -- C?digo binario del d?gitc
    return std_logic_vector is

    variable Bin: std_logic_vector (3 downto 0);

    begin
    with Int select -- Selecci?n de la combinaci?
        Bin :=    ("0000") WHEN (0),
            ("0001") WHEN (1),
            ("0010") WHEN (2),
            ("0011") WHEN (3),
            ("0100") WHEN (4),
            ("0101") WHEN (5),
            ("0110") WHEN (6),
            ("0111") WHEN (7),
            ("1000") WHEN (8),
            ("1001") WHEN OTHERS;
    return Bin; -- Devoluci?n del respectivo valc
end function;
```

```
function BinToInt (
    Bin2: in std_logic_vector (3 downto 0))
    return integer is

    variable Int2: integer;

    begin
    with Bin2 select -- Selecci?n de la coml
        Int2 :=    (0) WHEN ("0000"),
            (1) WHEN ("0001"),
            (2) WHEN ("0010"),
            (3) WHEN ("0011"),
            (4) WHEN ("0100"),
            (5) WHEN ("0101"),
            (6) WHEN ("0110"),
            (7) WHEN ("0111"),
            (8) WHEN ("1000"),
            (9) WHEN ("1001"),
            (10) WHEN ("1010"),
            (11) WHEN ("1011"),
            (12) WHEN ("1100"),
            (13) WHEN ("1101"),
            (14) WHEN ("1110"),
            (15) WHEN OTHERS;
    return Int2; -- Devoluci?n del respecti
end function;
```

```
digit  <= boton_pres;
ind_s <= ind;
LEDMODO <= MODO;
LEDCARGA <= carga;
LEDROTA <= rotando;
```

Generamos, entonces, una función que genera una frecuencia que depende de la velocidad introducida para no utilizar velocidades predeterminadas, mediante el uso de la siguiente ecuación:

$$f_2 = \frac{f\ Hz}{2} \cdot \frac{360}{200} \cdot \frac{1}{V\ grad/s} = X\ veces/grado$$

```
Revisa: process(CLK, Velocidad) -- Va a "mandar una se?al" a clock_state cada x ms
begin
    if CLK'event and CLK = '1' then
        if Velocidad = 0 then
          null;
        elsif conteo2 < ((50_000_000*360)/(Velocidad*200)) then
            conteo2 <= conteo2 + 1;
        else
            clck_state100 <= not clck_state100;
            conteo2 <= 0;
        end if;
    end if;
end process;
```

No es muy preciso el resultado, pues se está trabajando con enteros, pero se mantiene en un rango aceptable. Hemos de considerar que se está utilizando

frecuencia media porque ha de suceder 2X veces para darse un paso del motor, ya que cada paso se llama cuando clck_state100 llega vuelve a su valor '1'.

```
Motooor: process(cambio2)
begin
    if modo = '0' then
        if Cambio2 = "000" then
            motor <= "0001";
        elsif Cambio2 = "001" then
            motor <= "0011";
        elsif Cambio2 = "010" then
            motor <= "0010";
        elsif Cambio2 = "011" then
            motor <= "0110";
        elsif Cambio2 = "100" then
            motor <= "0100";
        elsif Cambio2 = "101" then
            motor <= "1100";
        elsif Cambio2 = "110" then
            motor <= "1001";
        elsif Cambio2 = "111" then
            motor <= "1000";
        end if;
    else
        if rotando = '1' and veces < (Angle*10/18) then
            if Cambio2 = "000" then
                motor <= "0001";
            elsif Cambio2 = "001" then
                motor <= "0011";
            elsif Cambio2 = "010" then
                motor <= "0010";
            elsif Cambio2 = "011" then
                motor <= "0110";
            elsif Cambio2 = "100" then
                motor <= "0100";
            elsif Cambio2 = "101" then
                motor <= "1100";
            elsif Cambio2 = "110" then
                motor <= "1001";
            elsif Cambio2 = "111" then
                motor <= "1000";
            end if;
            veces <= veces + 1;
        elsif rotando = '0' then
            veces <= 0;
        end if;
    end if;
end process;
```

Luego creamos el proceso que identifica el botón siendo pulsado:

```
ShowsNumber: process(ind_s)
variable aux1, aux2, aux3, aux4, aux5, aux6, aux7, aux8: std_logic_vector (3 downto 0);
begin
        if rising_edge (ind_s) then
            if digit = "1110" then
                if modo = '1' then
                    modo <= '0';
                elsif modo = '0' then
                    modo <= '1';
                end if;
                carga <= '0';
            elsif digit <= "1001" then
                if carga = '1' then
                    carga <= '0';
                    aux6 := "0000";
                    aux7 := "0000";
                    aux8 := digit;
                else
                    aux6 := decena;
                    aux7 := unidad;
                    aux8 := digit;
                end if;

                centena <= aux6;
                decena <= aux7;
                unidad <= aux8;
                carga <= '0';
            elsif digit = "1111" then
                carga <= '1';
            else
                carga <= '0';
            end if;
        end if;
end process;
```

Además, creamos el proceso que recibirá un aviso del proceso anterior de si se cargará el número actual o de un cambio de modo:

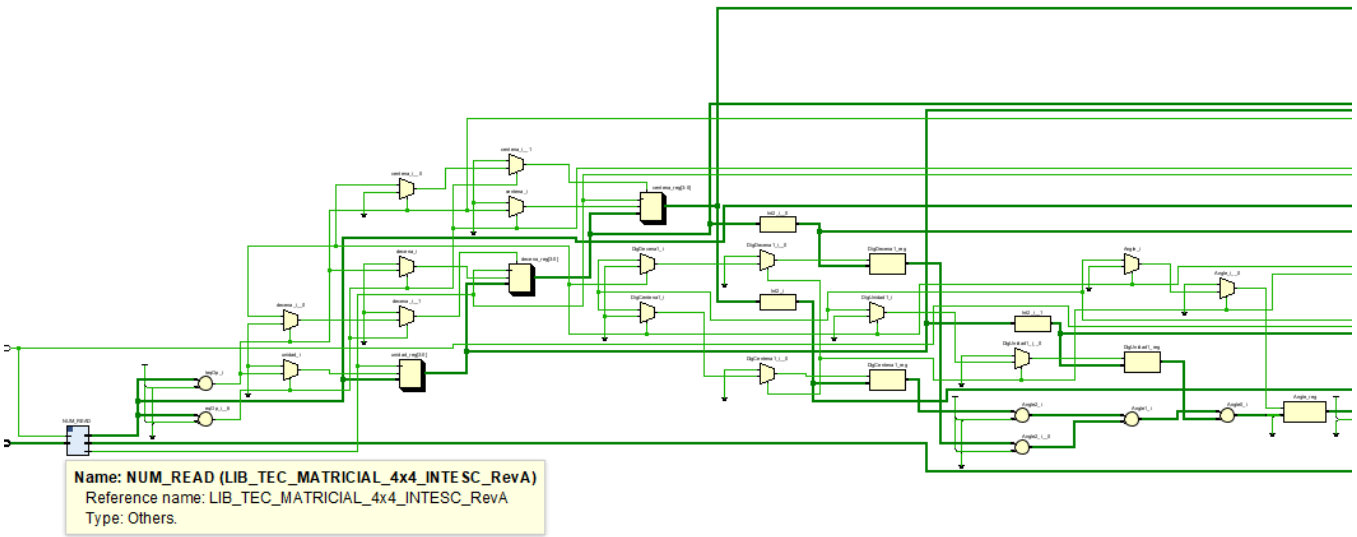
```
ProcesoDeCarga: process(carga, veces)
begin

    if rotando = '1' and veces >= (Angle*10/18) then
        rotando <= '0';
        --Angle <= 0;
    elsif carga = '1' then
         if modo = '0' then

            digcentena       <= bintoint(centena     );
            digdecena        <= bintoint(decena      );
            digunidad        <= bintoint(unidad      );

            Velocidad <= digcentena*100 + digdecena*10 + digunidad;
        elsif modo = '1' then

            digcentenal       <= bintoint(centena     );
            digdecenal        <= bintoint(decena      );
            digunidadl        <= bintoint(unidad      );

            Angle <= digcentenal*100 + digdecenal*10 + digunidadl;
            Velocidad <= 10;
            rotando <= '1';
        end if;
    end if;
end process;
```

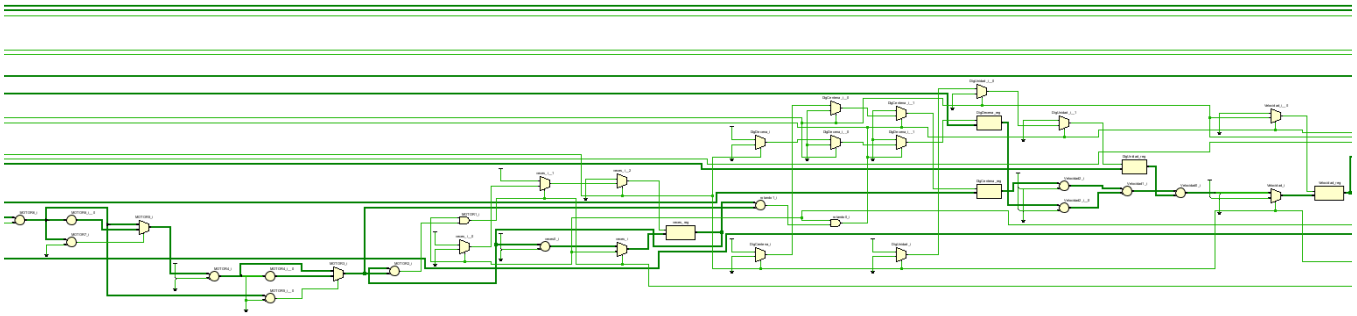
Hemos de notar que aquí, igualmente se cambia el status de rotando a '1' o a '0', esta señal permite la rotación del motor en el proceso donde se dan los pasos en el caso de un ángulo específico. ¿Cómo? Pues en ese proceso se hace un conteo de cantidad de pasos totales a dar, y en caso de dar un paso más al contado inicialmente, en este proceso se comprueba si llegamos a dicha cantidad y la señal rotando se pasa a '0', impidiendo así la reactivación de un paso a menos que intencionalmente se vuelva a activar.

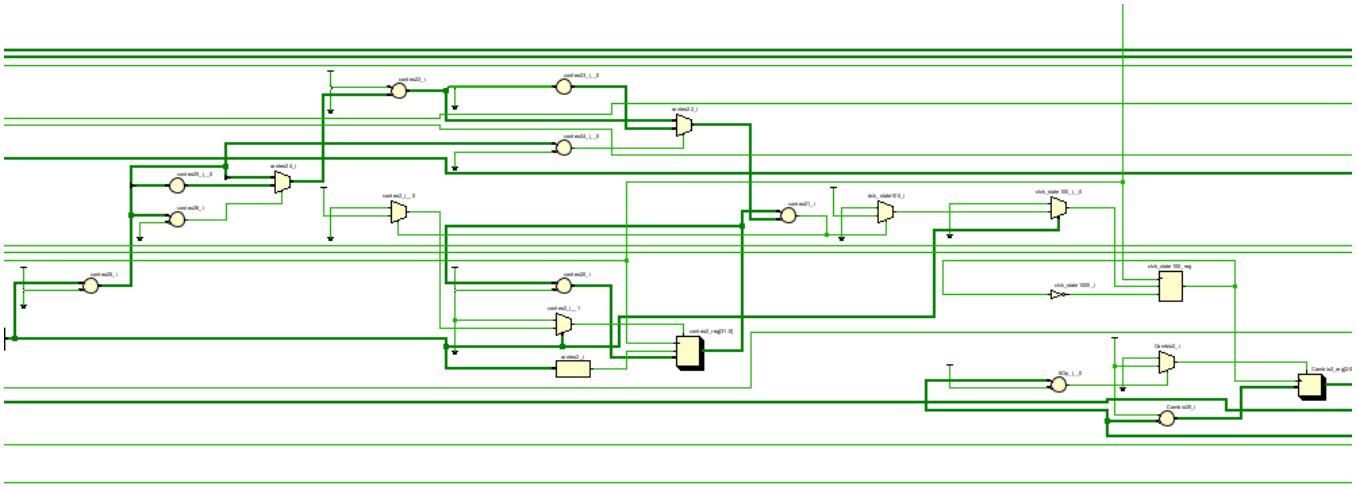
Resultados:

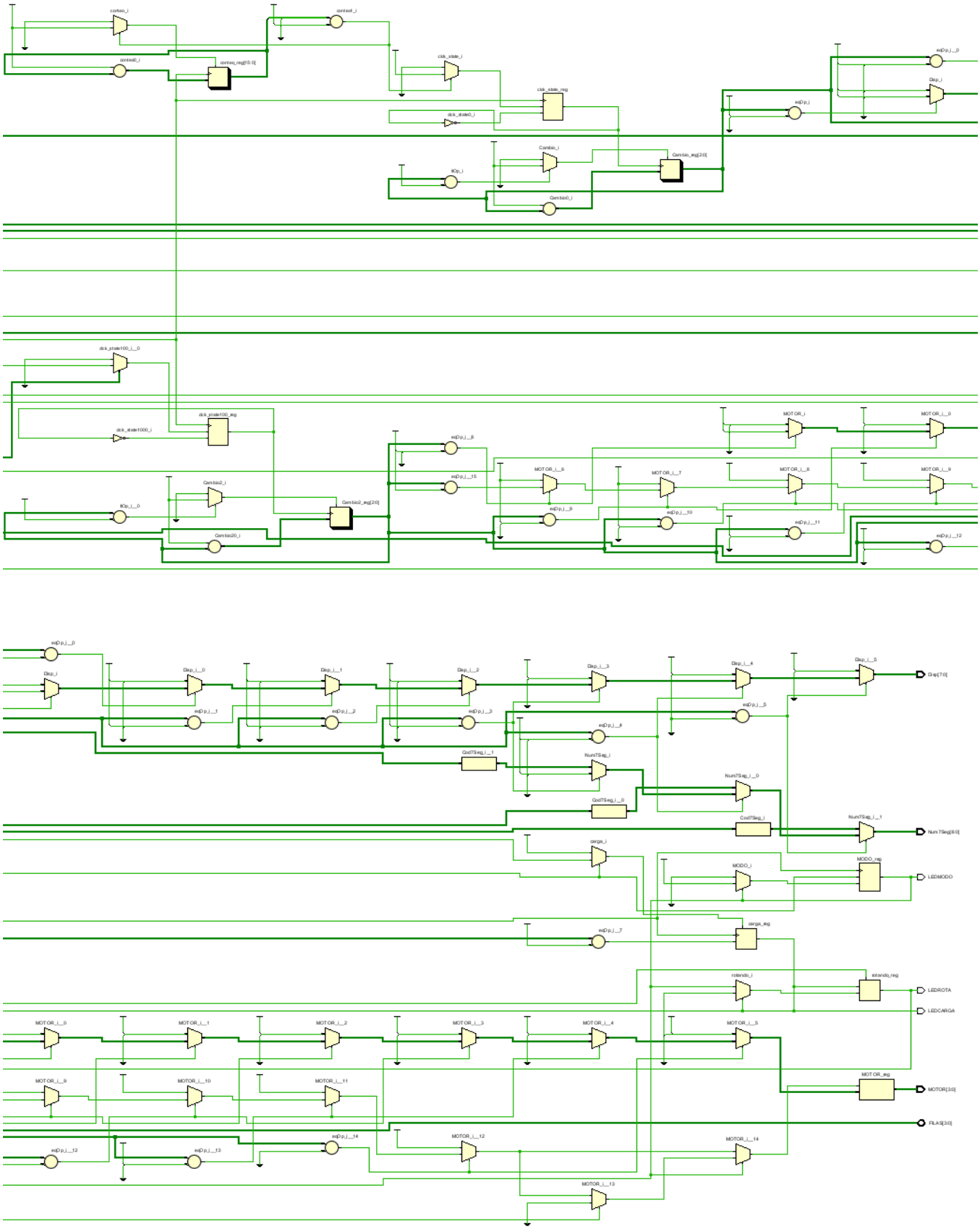
Hemos obtenido una reacción positiva por parte del motor. Este varía su velocidad con una certeza sorprendente, mientras mayor el valor introducido, mayor la velocidad, tal y como se estimaba. El mismo caso con el ángulo, el motor inicia su recorrido y se detiene en un ángulo cercano al especificado.

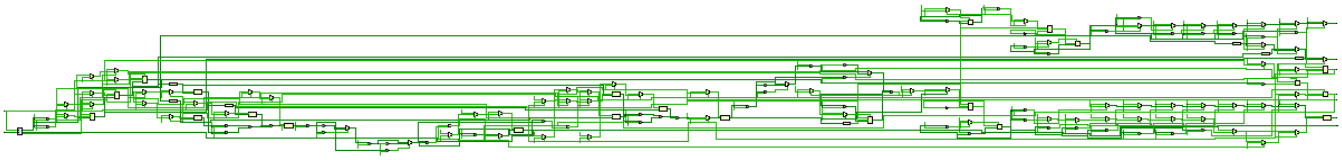
Además, se obtiene el siguiente diseño con la misma complejidad al anterior:


Name: NUM_READ (LIB_TEC_MATRICIAL_4x4_INTESC_RevA)
Reference name: LIB_TEC_MATRICIAL_4x4_INTESC_RevA
Type: Others.

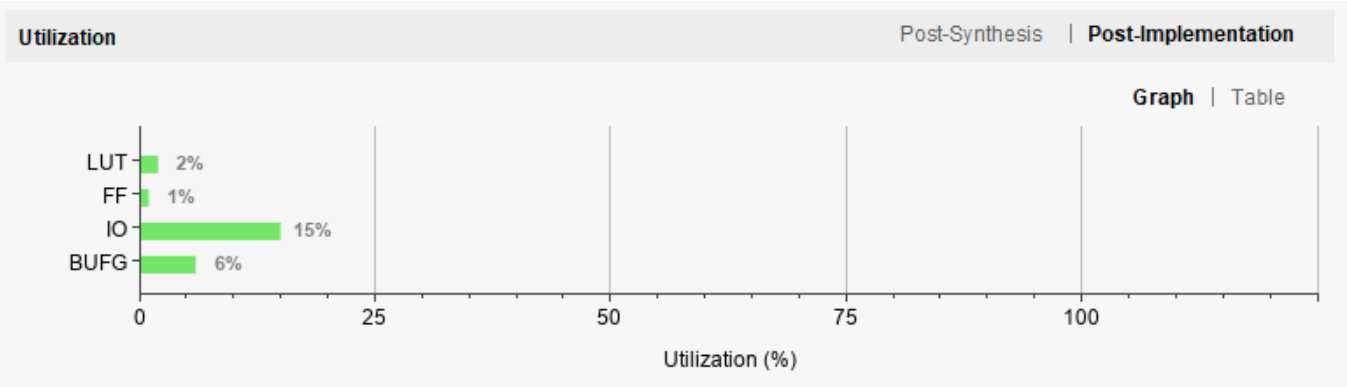




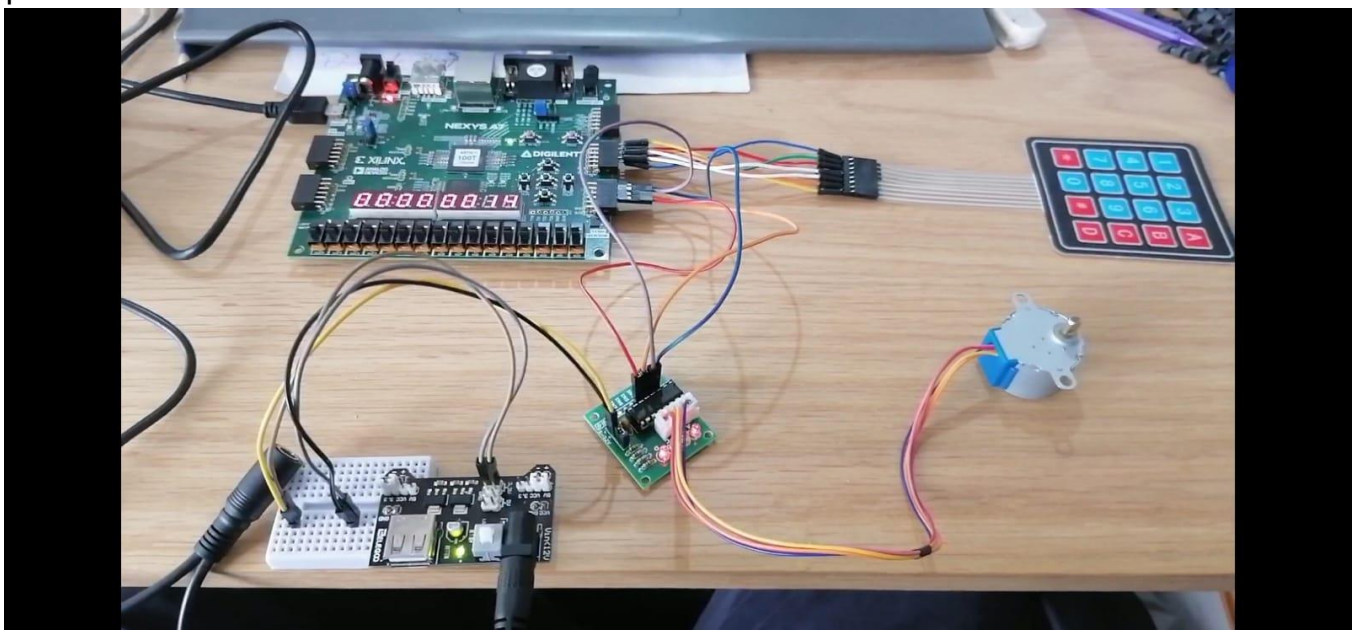

contseo_i
contseo0_i
contseo_reg[15:0]
contseot_i
clck_state_i
clck_state_reg
clck_state0_i
Cambio_i
Cambio_reg[23]
IOp_i
Cambio0_i
exD:p,j__0
Disp_i
exD:p,j
clck_state100_i__0
clck_state100_reg
clck_state1000_i
exD:p,j__8
exD:p,j__15
Cambio2_i
IOp_i__0
Cambio2_reg[20]
Cambio20_i
MOTOR_i__6
exD:p,j__9
MOTOR_i__7
exD:p,j__10
MOTOR_i__8
exD:p,j__11
MOTOR_i
MOTOR_i__0
MOTOR_i__9
exD:p,j__12
exD:p,j__0
Disp_i
Disp_i__0
exD:p,j__1
Disp_i__1
exD:p,j__2
Disp_i__2
exD:p,j__3
Disp_i__3
exD:p,j__4
Disp_i__4
exD:p,j__5
Disp_i__5
Disp[7:0]
Cod7Seg_i__1
Num7Seg_i
Cod7Seg_i__0
Num7Seg_i__0
Cod7Seg_i
Num7Seg_i__1
Num7Seg[6:0]
carga_i
MODO_i
MODO_reg
LEDMODO
exD:p,j__7
carga_reg
rotando_i
estando_reg
LEDROTA
LEDCARGA
MOTOR_i__0
MOTOR_i__1
MOTOR_i__2
MOTOR_i__3
MOTOR_i__4
MOTOR_i__5
MOTOR_i__9
MOTOR_i__10
MOTOR_i__11
MOTOR_reg
MOTOR[3:0]
FILAS[3:0]
exD:p,j__12
exD:p,j__13
exD:p,j__14
MOTOR_i__12
MOTOR_i__14
MOTOR_i__13



A pesar de la ”complejidad” presentada, e programa estima el siguiente uso de recursos:



Nos encontramos ante la siguiente configuración física. Hemos de considerar que el motor stepper ha de ser alimentado mediante una fuente externa, pues no es aconsejable alimentar el motor con la misma tarjeta de desarrollo, pues esta no posee tal potencia. Además, siempre se aconseja el uso de un driver para el motor para evitar daños.



Análisis:

Se pudo comprender el uso del motor de pasos (stepper) y su funcionamiento, así como su aprovechamiento físico. Se observa que el funcionamiento correcto de este motor depende de una señal que se le sea enviada dígase con un patrón cíclico o

secuencial. Incluso, mediante el mismo método, su dirección puede ser fácilmente cambiada de sentido horario a antihorario, o viceversa.

En el proceso de diseño, pudimos recalcar sin temor a equivocación, que no es ni fácil ni útil el asignar un valor a una señal en procesos diferentes, aunque estos no se activen al mismo tiempo.

Mediante la aplicación física se pudo observar que el motor tiene una velocidad literal limitada a menos de 90 grados por segundo, esto se debe a que, como es necesario que cruce corriente en ciertos puntos en tiempos distintos para activar un imán, no alcanza el tiempo para manifestarse este efecto físico.

Finalmente, hemos logrado controlar un stepper tal y como se esperaba con la distinción entre velocidad y cantidad de vueltas.