Universidad Tecnológica Nacional

Técnicas Digitales IV

Trabajo Práctico 1: Decodificador de encoder y control de lazo cerrado en "posición" de un motor C.C

Autores:
Matías Iván Bucca
Ignacio Rodríguez Grandi
Jonás Gabriel Ramírez Torres
Federico Nicolás Elizondo

Supervisor:
Pablo Cayuela
Sergio Olmedo

June 16, 2021

Contents

1 Introducción

El siguiente informe documenta los procedimientos realizados para el diseño digital de un controlador de un motor mediante la lectura de un encoder. Para esto, se utilizan parte de las prácticas adquiridas en los prácticos de entrenamiento.

2 Primera parte

2.1 Decodificador de encoder visualizando en cuatro display de 7 segmentos

2.1.1 Objetivo

Realizar la descripción de divisores de frecuencia para el barrido de los displays a una frecuencia de 1 kHz utilizando un CLK de 15 kHz.

Describir un contador decimal de 000 a 999. Incorporar el concepto descripción jerárquico utilizando instanciación de componentes. Con una descripción de máquina de estado realizar la decodificación de los flancos de las señales A y B del encoder y transformarlas en pulsos Up/down según el sentido de giro (ej. Horario sea pulsos up y antihorario, pulsos down).

2.1.2 Diagrama en bloques del sistema a realizar

En el siguiente diagrama en bloques se propone una posible solución al ejercicio planteado anteriormente.

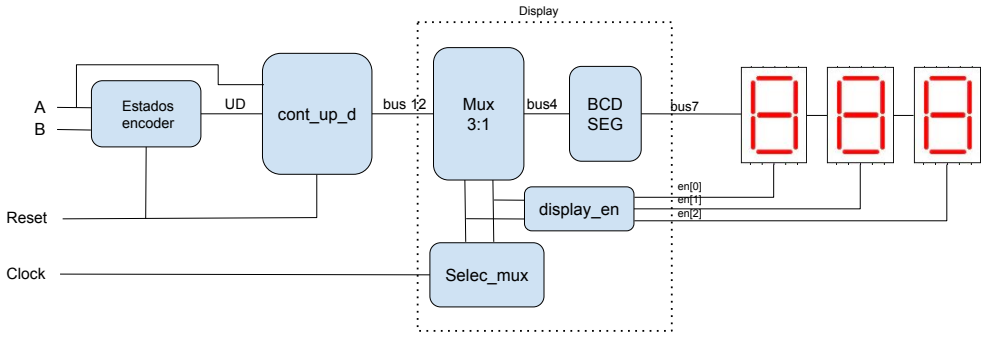

Figure 1: Bloques del decodificador encoder planteado.

2.1.3 Estados encoder

Este bloque tendrá como entradas las señales A y B del encoder y mediante la descripción de una máquina de estados simple, la salida UD será "1" para indicar un incremento positivo y "0" para el caso contrario.

```
library ieee;
use ieee.std_logic_1164.all;

entity estados_encoder is

        port(
                --flanco : in        std_logic;
                A : in        std_logic;
                B : in        std_logic;
                reset         : in        std_logic;
                output         : out        std_logic
        );

end entity;

architecture rtl of estados_encoder is

        -- Build an enumerated type for the state machine
        type state_type is (s0, s1, s2, s3, s4);

        -- Register to hold the current state
        signal state   : state_type;
        signal flanco : std_logic;

begin
    flanco <= A xor B;
    -- Logic to advance to the next state
    process (flanco ,  reset)
    begin
            if reset = '1' then
                    state <= s0;
            --end if;
            elsif (flanco'event and flanco='1') then
                    case state is
                      when s0 =>
                                    if A='1' then
                                            state <= S2;
                                    elsif B='1' then
                                            state <= S1;
                                    end if;
                      when s1 =>
                                    if A='1' then
                                            state <= S4;
                                    else
                                            state <= S1;
```

```
                                        end if;
                                when s2 =>
                                        if B='1' then
                                                state <= S3;
                                        else
                                                state <= S2;
                                        end if;
                                when s3 =>
                                        if A='1' then
                                                state <= S2;
                                        elsif B='1' then
                                                state <= S1;
                                        end if;
                                when s4 =>
                                        if A='1' then
                                                state <= S2;
                                        elsif B='1' then
                                                state <= S1;
                                        end if;
                        end case;
        end if;
        end process;

            -- Output depends solely on the current state
            process (state)
            begin
                    case state is
                            when s0 =>
                                    output <= '1';
                            when s1 =>
                                    output <= '0';
                            when s2 =>
                                    output <= '1';
                            when s3 =>
                                    output <= '1';
                            when s4 =>
                                    output <= '0';
                    end case;
            end process;

    end rtl;
```

Los resultados de la simulación de este bloque se pueden ver en la figura 2.

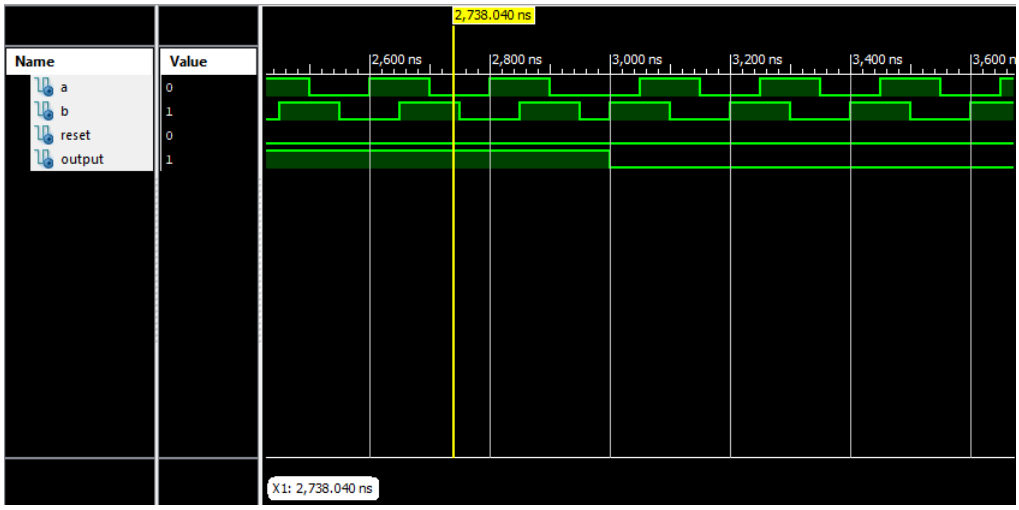

Figure 2: Test Bench estados encoder

2.1.4 Contador up down

Este bloque se encargará de hacer una cuenta incremental o decremental según el nivel de la entrada UD. Nota: en este bloque se instancian tres contadores de cuatro bits.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
    entity cont_up_d is
        Port ( A_B : in  STD_LOGIC;
               U_D : in  STD_LOGIC;
               R : in  STD_LOGIC;
               BCD_OUT : out  STD_LOGIC_VECTOR (11 downto 0));
    end cont_up_d;

    architecture Behavioral of cont_up_d is

    COMPONENT contador_4bits
            Port ( X : in  STD_LOGIC;
                              UD : in  STD_LOGIC;
                              R : in  STD_LOGIC;
                              S : in  STD_LOGIC;
                              Co : out STD_LOGIC;
                              Output : out  STD_LOGIC_VECTOR (3 downto 0));
    END COMPONENT;

    signal sCo : STD_LOGIC_VECTOR (1 downto 0);

    begin
            INST_CONT0: contador_4bits PORT MAP(
                    X => A_B,
                    UD => U_D,
                    R => R,
```

```
            S => '0',
            Co => sCo(0),
            Output => BCD_OUT(3 downto 0)
    );
    INST_CONT1: contador_4bits PORT MAP(
            X => sCo(0),
            UD => U_D,
            R => R,
            S => '0',
            Co => sCo(1),
            Output => BCD_OUT(7 downto 4)
    );
    INST_CONT2: contador_4bits PORT MAP(
            X => sCo(1),
            UD => U_D,
            R =>  R,
            --Co => ,
            S => '0',
            Output => BCD_OUT(11 downto 8)
    );
```

La simulación para una cuenta incremental es la de la figura 3 y para la decremental es la de la figura 4.

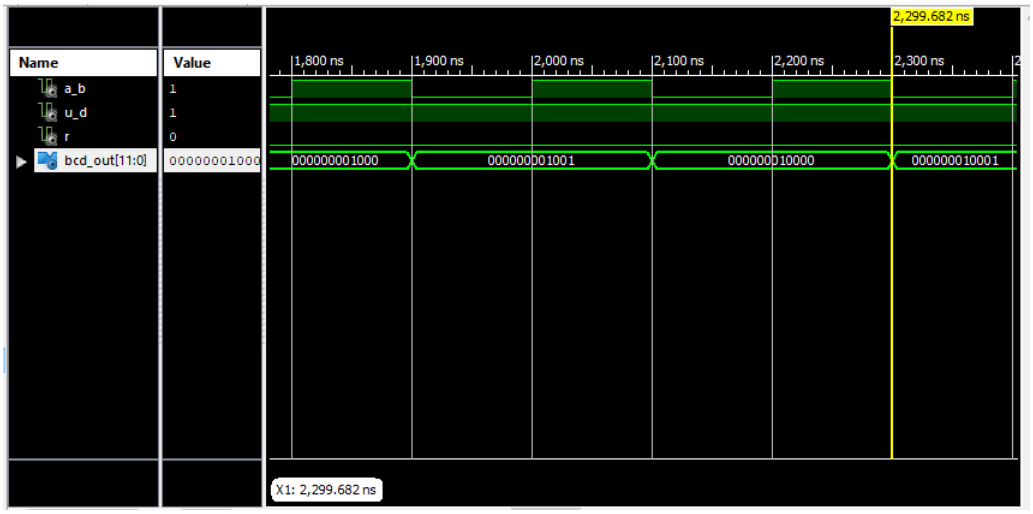

Figure 3: Test Bench cuenta incremental

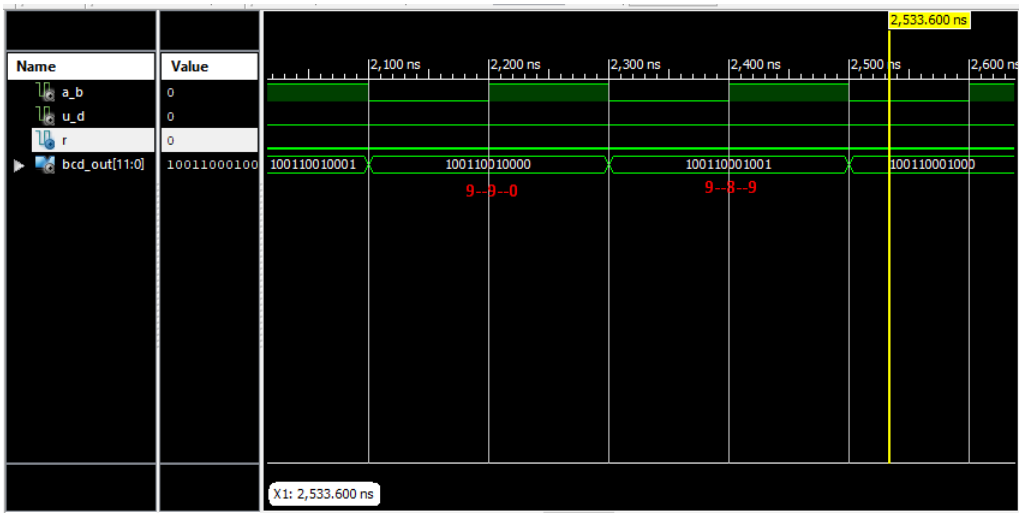

Figure 4: Test Bench cuenta decremental

2.1.5 Bloque Display

En este bloque se instancian cuatro bloques importantes encargados de la visualización de los datos en los displays.

- **Mux 3 a 1**: este será el encargado de multiplexar el bus de 12 bits de la salida del contador en 4 bits.
- **Selec mux**: como tarea principal, este bloque se encarga de dividir la frecuencia de entrada de 15kHz a 1 kHz. Posteriormente, tendrá dos salidas de selección que irán conectadas al bloque anterior.
- **BCD SEG**: realiza la transformación de binario a decimal para poder visualizar los datos en los displays de 7 segmentos.
- **Display enable**: tomando como referencia la salida del bloque selec mux, realizar un barrido en su salida que funcionará como enables de los displays.

A continuación la descripción realizada. Nota: los componentes instanciados se encuentran incluidos en el archivo de entrega.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity display is
    Port ( bcd : in  STD_LOGIC_VECTOR (11 downto 0);
                          input : in  STD_LOGIC;
           output : out  STD_LOGIC_VECTOR (6 downto 0);
           en : out  STD_LOGIC_VECTOR (2 downto 0));
end display;

architecture Behavioral of display is
COMPONENT BCD_SEG
    Port ( BCD_IN : in  STD_LOGIC_VECTOR (3 downto 0);
           SEG_OUT : out  STD_LOGIC_VECTOR (6 downto 0));
END COMPONENT;

COMPONENT mux12_4
    Port ( MUX_IN : in  STD_LOGIC_VECTOR (11 downto 0);
           MUX_OUT : out  STD_LOGIC_VECTOR (3 downto 0);
                          SEL : in  STD_LOGIC_VECTOR (1 downto 0));
END COMPONENT;
COMPONENT slec_mux
                port(
                        input          : in        std_logic;
                        --reset          : in        std_logic;
                        output         : out       std_logic_vector(1 downto 0)
                );
END COMPONENT;
signal sSelec: std_logic_vector(1 downto 0);
signal sSalMux: std_logic_vector(3 downto 0);
begin
        INST_MUX12_4: mux12_4 PORT MAP(
                MUX_IN => bcd,
                SEL => sSelec,
                MUX_OUT => sSalMux
        );
        INST_BCD: BCD_SEG PORT MAP(
                BCD_IN => sSalMux ,
                SEG_OUT => output
        );
         INST_SELMUX: slec_mux PORT MAP(
                input => input,
                output => sSelec
        );
--Display enable
en <= "011" when sSelec = "00" else
          "101" when sSelec = "01" else
      "110" when sSelec = "10" else "111";
end Behavioral;
```

2.1.6 Video de funcionamiento

Haciendo clic **aquí** se abrirá un enlace con el video final de funcionamiento.

3 Segunda Parte

3.1 Control de lazo cerrado en "posición" de un motor C.C

3.1.1 Objetivo

Realizar la descripción de un decodificador de encoder incremental visualizando en un display los pulsos en n contador Up Down. Con la cuenta, se debe controlar una señal PWM. Esta señal, el ciclo de trabajo debe variar entre 5% al 95%. La frecuencia debe ser de 400HZ.

El ciclo de trabajo al 50% del PWM, debe suceder cuando la comparación de la cuenta de contador Up Down es igual a la entada de referencia.

La entrada de referencia, nos dice en qué lugar se quiere posicionar el motor, dentro de una revolución indicada por la cuenta desde 0 a 1999 pulsos de encoder.

La entrada debe ser de 11 bit, como también el contador Up Down.

La señal de PWM debe ser máxima para una diferencia +- de 100 entre la cuenta de los pulso (contdor Up Down) y la de referencia.

Se debe simular variaciones de posición del motor en forma aceleración y desaceleración en el punto de referencia para demostrar el funcionamiento del PWM.

3.1.2 Resolución

Para esta parte simplemente se implemento el diseño de un bloque PWM, el cual tendrá como entrada de referencia la salida de 12 bits del contador Up Down y un clock de 150Khz. La salida del bloque permanecerá en alto hasta que la cuenta de un contador interno sea igual a la referencia de la entrada. Posteriormente la salida pasa a un nivel bajo hasta completar la cuenta.

3.1.3 Descripción bloque PWM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity pwm is
    Port ( clk : in  STD_LOGIC;
                          R : in  STD_LOGIC;
           ref : in  STD_LOGIC_VECTOR (11 downto 0);
           Output : out  STD_LOGIC);
end pwm;
```

```
architecture Behavioral of pwm is

COMPONENT contador_4bits
        Port ( X : in  STD_LOGIC;
                        UD : in  STD_LOGIC;
                        R : in  STD_LOGIC;
                        S : in  STD_LOGIC;
                        Co : out STD_LOGIC;
                        Output : out  STD_LOGIC_VECTOR (3 downto 0));
END COMPONENT;

signal sCo : STD_LOGIC_VECTOR (1 downto 0);
signal sCuenta : STD_LOGIC_VECTOR (11 downto 0);


begin
        INST_CONT5: contador_4bits PORT MAP(
                X => clk,
                UD => '1',
                R => R,
                S => '0',
                Co => sCo(0),
                Output => sCuenta(3 downto 0)
        );
        INST_CONT6: contador_4bits PORT MAP(
                X => sCo(0),
                UD => '1',
                R => R,
                S => '0',
                Co => sCo(1),
                Output => sCuenta(7 downto 4)
        );
        INST_CONT7: contador_4bits PORT MAP(
                X => sCo(1),
                UD => '1',
                R =>  R,
                Co => open,
                S => '0',
                Output => sCuenta(11 downto 8)
        );
Output<= '1' when (sCuenta < ref)   else '0';
end Behavioral;
```

3.1.4 Video de funcionamiento

Haciendo clic **aquí** se abrirá un enlace con el video final de funcionamiento.

References

[1] FPGA Prototyping by VHDL Examples (Xilinx Spartan 3 Version) - Pong P. Chu