| | Carátula para entrega de prácticas |
|---|---|
| Facultad de Ingeniería | Laboratorios de docencia |

# Laboratorio

*Profesor:* ING. JULIO CESAR CRUZ ESTRADA

*Asignatura:* LABORATORIO DE ORGANIZACIÓN Y ARQUITECTURA DE COMPUTADORAS

*Grupo:* 07

*No de Práctica:* 2

*Integrante(s):* Jiménez Treviño Emilio Cristóbal

Martinez Perez Brian Erik

Robles Jiménez Marco Antonio

*No. de Equipo de cómputo empleado:* *s/n*

*Semestre:* *2026-1*

*Fecha de entrega:* *12/09/2025*

*Observaciones:*

CALIFICACIÓN: _____

# Práctica 2. Diseño de máquinas de estado

## Objetivo

Familiarizar al alumno en el conocimiento de los algoritmos de las máquinas de estados utilizando Quartus y el lenguaje VHDL.

## Introducción

En esta práctica de diseño de máquinas de estado, abordaremos el estudio de las máquinas de estados finitos (FSM) y su implementación utilizando el software Quartus junto con el lenguaje de descripción de hardware VHDL. Se explorarán los conceptos fundamentales detrás del diseño de FSM, incluyendo la representación de estados y transiciones a través de cartas ASM (Algorithmic State Machine), así como la derivación en circuitos secuenciales a partir de estas representaciones, utilizando mapas de Karnaugh para la simplificación de la lógica combinacional. Se detallarán los pasos para la creación de un proyecto en Quartus, la implementación de lógica combinacional y secuencial, y la asignación de pines para su posterior simulación y carga en un dispositivo FPGA que en nuestro caso fue la DE10-LITE debido a su capacidad y accesibilidad.

## Desarrollo

Para comenzar primero se propuso una carta ASM con 2 estados y 3 entradas.

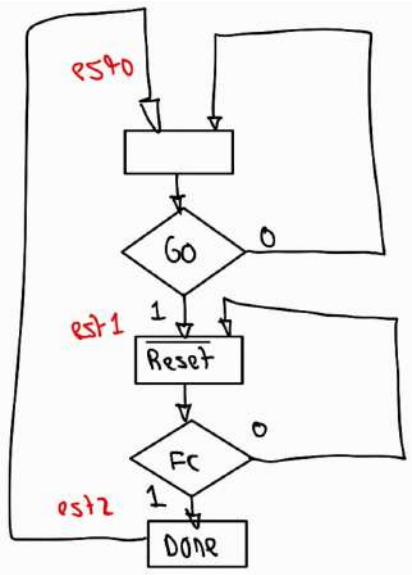



**Figura 1**. Carta ASM

Después de obtener la carta ASM, obtenemos el circuito secuencial de la carta ASM utilizando flip flops tipo D, para ello debemos de obtener los mapas de karnaugh, para reducir el circuito y tener que evitar usar tantas compuertas.

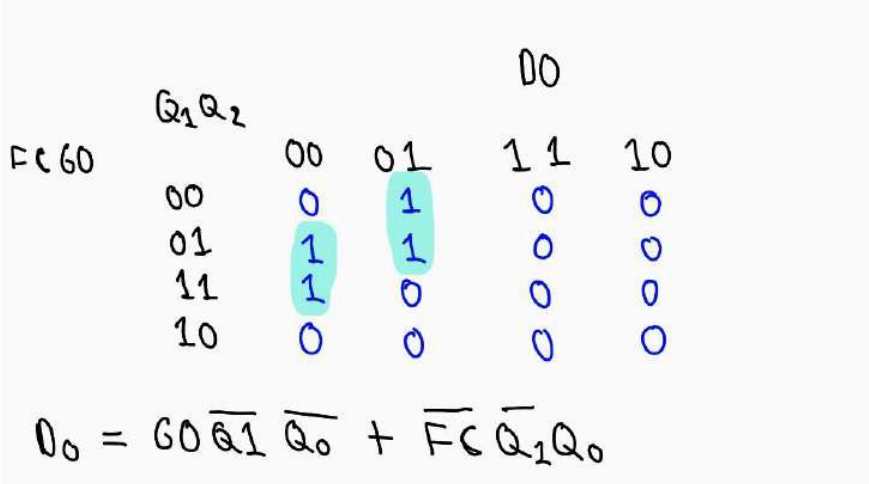



**Figura 2**. Mapa karnaugh del estado 0

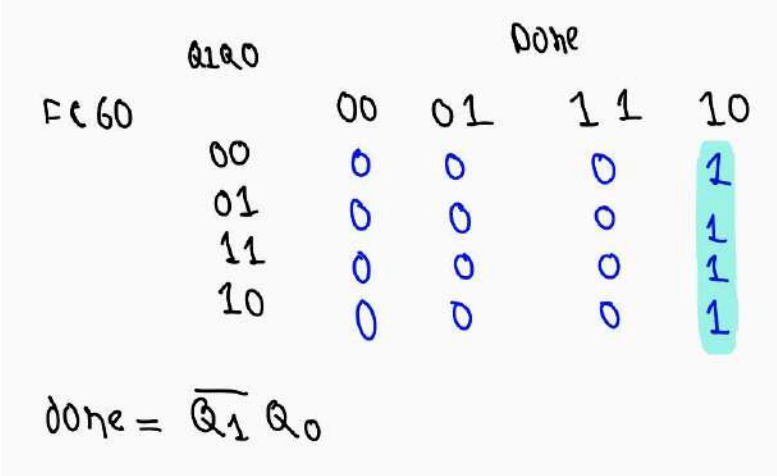



**Figura 3**. Mapa de karnaugh Done

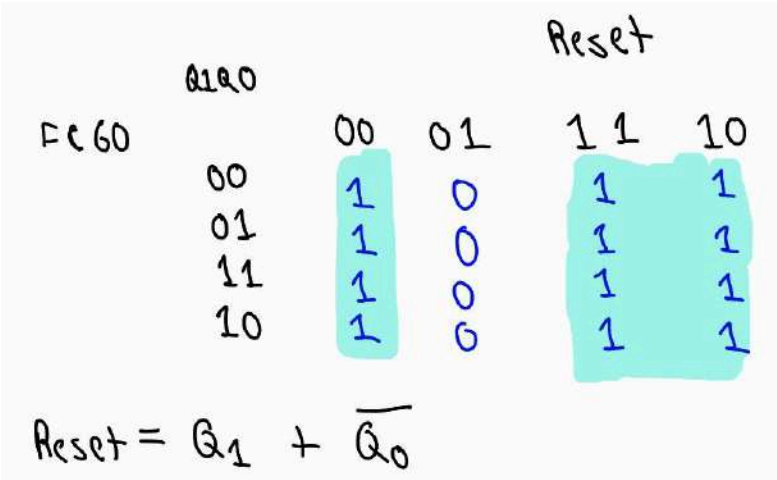



**Figura 4**. Mapa de karnaugh Reset

Cuando ya tenemos los mapas de karnaugh, podemos realizar el circuito secuencial equivalente.

Lo primero que debemos hacer es crear un proyecto en Quartus, seleccionamos la pestaña FILE -> New Project Wizard. En el asistente de creación de nuevos proyectos,en la Fig. 1 podemos observar seleccionado la familia del dispositivo en nuestro caso la MAX 10 y dentro de esta vendría a hacer la 10M50DAF484C7G dentro de esta familia.

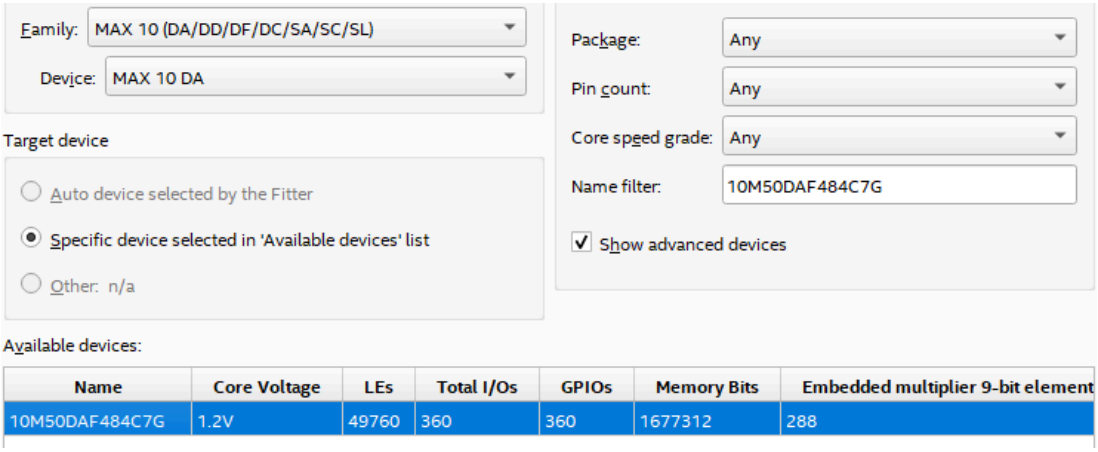



**Figura 4.** Device

Una vez creado nuestro proyecto, creamos un archivo de tipo .bdf dando click en la opción de Block Diagram/Schematic File como se muestra en la Fig. 5.

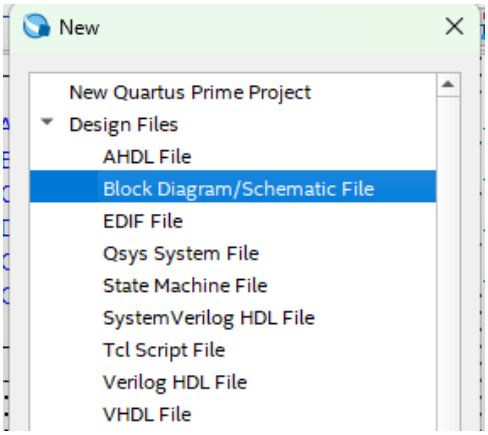



**Figura 5.** New File

En este archivo procedemos a hacer el circuito secuencial, debemos tener en cuenta que usamos 2 entradas “GO” y “CLK” y tenemos 5 salidas “A”, “B”, “C”, ”D” y “DONE”. además podemos observar en la imagen de nuestros diagrama eléctrico que utilizamos el contador “74193” y que se implementó un divisor de frecuencia para poder observar los resultados.

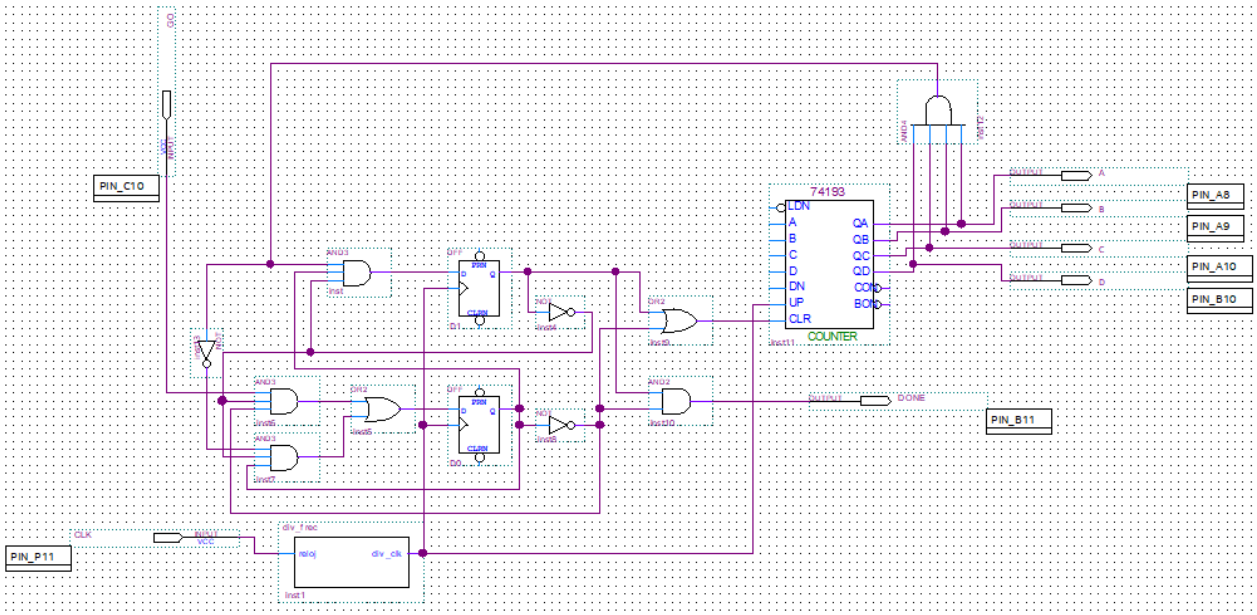



**Figura 6.** Diagrama de conexiones

Para nuestro divisor de frecuencia haremos un programa sencillo, por lo que ahora necesitaremos crear un nuevo archivo pero ahora de tipo .vhdl el código se muestra en la Fig.5 en el cual utilizamos la fórmula $n = \frac{1}{2x(2x10^{-9})}$ obteniendo como resultado 25 millones de pulsos para 1 segundo aproximadamente

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity div_frec is
    Port ( reloj : in std_logic;
    div_clk : out std_logic);
end div_frec;

architecture Behavioral of div_frec is
begin
    process (reloj)
    variable cuenta: std_logic_vector (27 downto 0):=x"0000000";
    begin
        if rising_edge (reloj) then
            if cuenta = x"17D7840" then --25M pulsos
                cuenta:= x"0000000";
            else
                cuenta:= cuenta+1;
            end if;
        end if;
        div_clk <= cuenta(24);
    end process;
end Behavioral;
```

**Figura 7.** Divisor de frecuencia

Cabe recalcar que para que nuestro .vhdl debemos crearlo como símbolo como se muestra en la Fig.8 con esto ya podremos agregarlo al .bdf como cualquier otro símbolo.

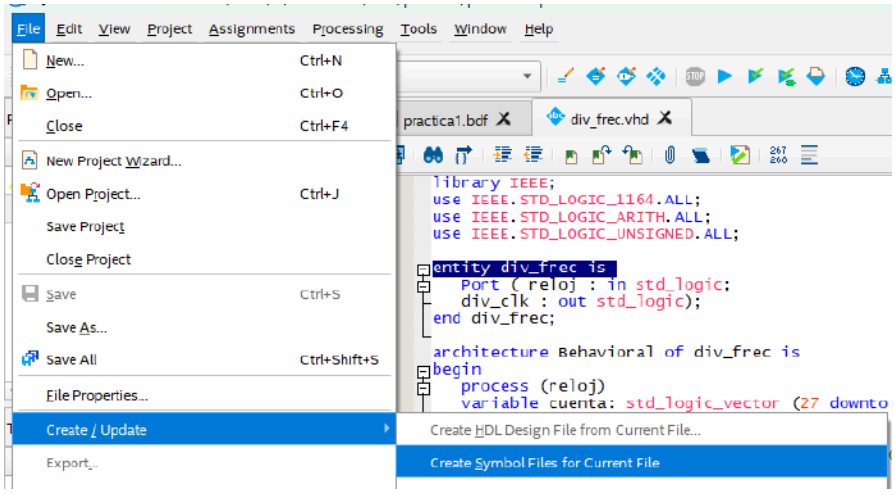



**Figura 8.** Símbolo

Procedemos con nuestro asignación de pines, para esto ocuparemos nuestro data sheet de nuestra FPGA DE-10 Lite, para hacer esto daremos click al icono que dice pin planner o presionando "ctrl+shift+n" los pines se colocan en el apartado de Location en la Fig. 9 podremos ver cuales se asignaron para este caso, recordemos que deben asignarse desde el más significativo al menos, para una correcta secuencia.

| Node Name | Direction | Location |
|---|---|---|
| out A | Output | PIN_A8 |
| out B | Output | PIN_A9 |
| out C | Output | PIN_A10 |
| in CLK | Input | PIN_P11 |
| out D | Output | PIN_B10 |
| out DONE | Output | PIN_B11 |
| in GO | Input | PIN_C10 |

**Figura 9.** Asignación de pines

# Resultados


**Fig 10**. a) Comienza el contador con “GO” en 1.

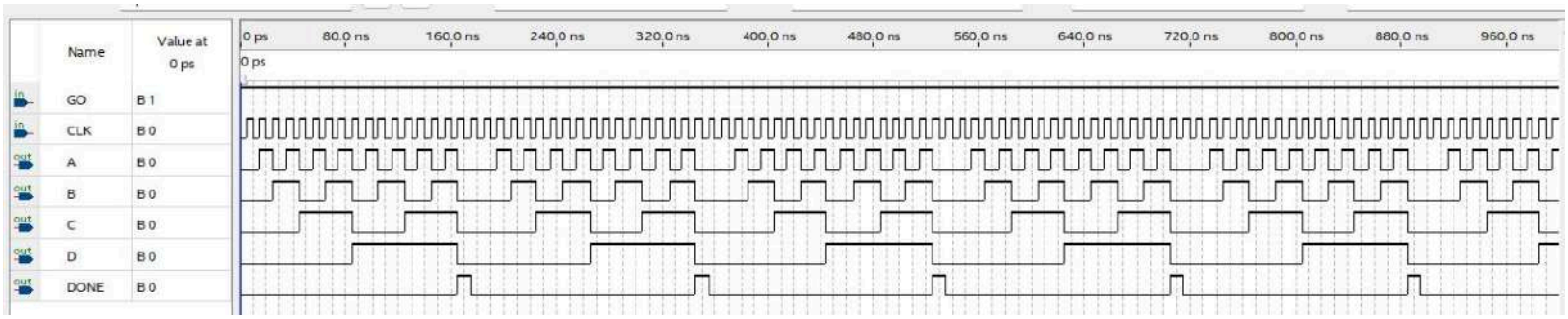

**Fig 10**. b) Continúa la cuenta mientras “GO” esté en 1.


**Fig 10**. c) cuando A, B, C, D son 1, la cuenta termina y “DONE” vale 1.


**Fig 10**. d) continua en un ciclo la cuenta mientras “GO” esté en 1.


**Fig 10**. e) la secuencia se desactiva cuando “GO” vale 0.

## Simulación




En esta simulación se observa que, con la señal GO en nivel alto, el contador empieza a funcionar sincronizado con el CLK. Las salidas A, B, C y D van cambiando de manera binaria, avanzando en secuencia desde 0000 hasta 1111 conforme recibe pulsos de reloj. Al llegar al valor máximo, se activa brevemente la señal DONE, indicando que el conteo terminó. Después, el contador vuelve a iniciar desde cero y el proceso se repite mientras GO permanezca en 1.

## Actividad Complementaria

Para la actividad complementaria se utilizó la siguiente carta ASM, donde se realiza el mismo proceso de contador de 0 a 15 en binario, solo que ahora internamente, se añade el bloque de “cuenta”, donde se modela el comportamiento de un controlador para un ciclo de conteo simple, garantizando que el contador externo se reinicie, cuente, y se detenga en el momento preciso según las señales de control.

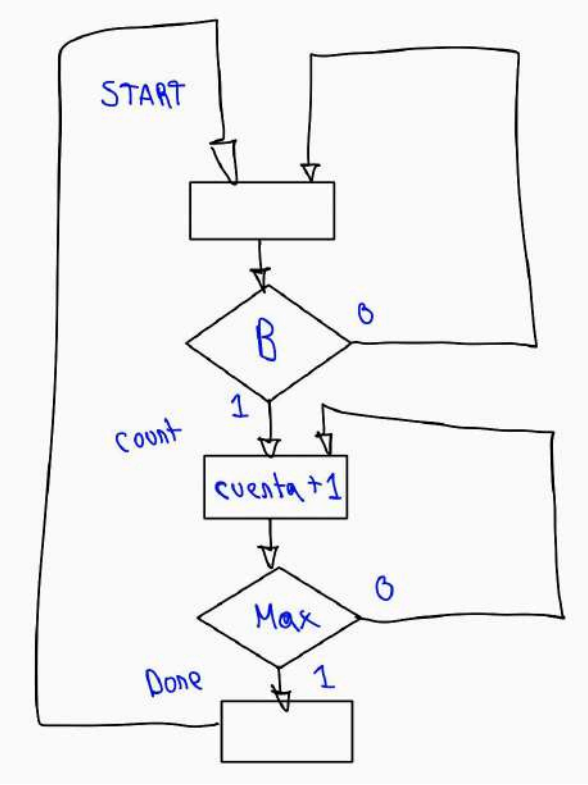



**Figura 11**. Carta ASM de la actividad complementaria

El diagrama de bloques obtenido fue el siguiente, donde se mantienen las salidas que indican los bits del contador, done que indica cuando la cuenta llegó al número 15, lo nuevo en este diagrama sería, que tenemos como salida el estado donde se encuentra la carta ASM, podemos entender aún mejor la secuencia que se sigue.

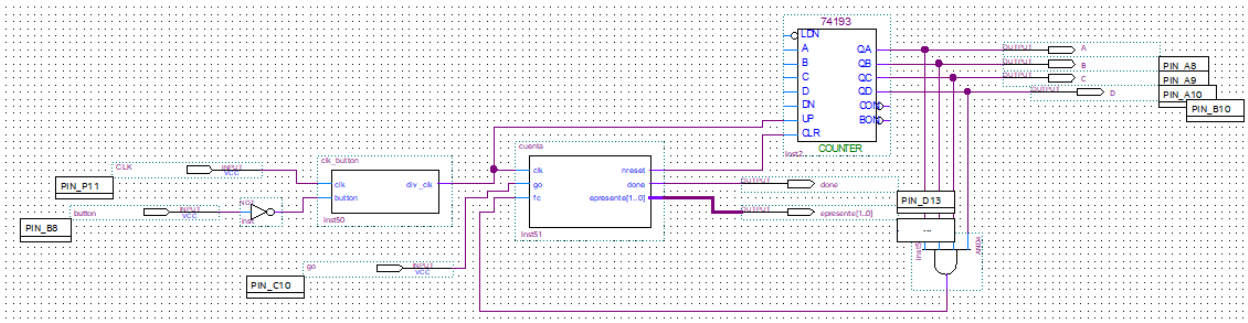



**Figura 12**. Diagrama de conexiones para la actividad complementaria.

El bloque “cuenta” se generó creando un archivo VHDL, El estado START (código "00"), es el estado inicial y de reposo. En este estado, la máquina mantiene la señal de reinicio (nreset) activa para asegurar que el contador externo esté en su valor inicial. El circuito permanece en este estado, esperando que la señal de entrada “GO” cambie a '1', lo que indica que se debe comenzar un nuevo ciclo de conteo.

COUNT (código "01"), cuando la señal “GO” se activa, la máquina pasa a este estado. Aquí, la señal de reinicio (nreset) se desactiva, liberando al contador externo para que comience a contar. El circuito se mantendrá en este estado hasta que la señal de entrada “FC” (final de conteo) se active, lo que significa que el contador ha alcanzado el valor máximo (en este caso, 15).

DONE (código "10"), Cuando el contador termina es decir, “FC” se activa, la máquina de estados pasa a este estado. En este punto, genera un pulso en la señal de salida “DONE” para indicar la finalización del conteo. Inmediatamente después, el circuito regresa al estado “START” para preparar el siguiente ciclo, asegurando que el proceso pueda repetirse.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity cuenta is
    Port(
        clk       : in  std_logic;                      -- viene de clk_b
        go        : in  std_logic;                      -- habilita un ci
        fc        : in  std_logic;                      -- AND(QA..QD), '
        nreset    : out std_logic;                      -- va a CLR del 7
        done      : out std_logic;                      -- pulso de fin d
        epresente : out std_logic_vector (1 downto 0)   -- código del est
    );
end cuenta;

architecture Behavioral of cuenta is

    -- Codificación de estados
    constant STATE_START : std_logic_vector(1 downto 0) := "00";
    constant STATE_COUNT : std_logic_vector(1 downto 0) := "01";
    constant STATE_DONE  : std_logic_vector(1 downto 0) := "10";

    signal state, next_state : std_logic_vector(1 downto 0) := STATE_START;

begin

    ----------------------------------------------------------------
    -- Registro de estado
    ----------------------------------------------------------------
    process(clk)
    begin
        if rising_edge(clk) then
            state <= next_state;
        end if;
    end process;

    ----------------------------------------------------------------
    -- Lógica combinacional de siguiente-estado y sa
    ----------------------------------------------------------------
    process(state, go, fc)
    begin
        -- valores por defecto
        nreset    <= '1';                   -- por def
        done      <= '0';
        epresente <= state;
        next_state <= state;

        case state is
            ------------------------------------------
            when STATE_START =>
                -- 74193 en reset, esperamos GO
                nreset    <= '1';
                done      <= '0';
                epresente <= STATE_START;

                if go = '1' then
                    next_state <= STATE_COUNT;
                else
                    next_state <= STATE_START;
                end if;

            ------------------------------------------
            when STATE_COUNT =>
                -- Liberamos CLR; el 74193 cuenta
                nreset    <= '0';
                done      <= '0';
                epresente <= STATE_COUNT;

                if fc = '1' then            -- lle
                    next_state <= STATE_DONE;
                else
                    next_state <= STATE_COUNT;
                end if;

            ------------------------------------------
            when STATE_DONE =>
                -- Un pulso de DONE y limpiamos e
                nreset    <= '1';
                done      <= '1';
                epresente <= STATE_DONE;

                next_state <= STATE_START;  -- re

            when others =>
                nreset    <= '1';
                done      <= '0';
                epresente <= STATE_START;
                next_state <= STATE_START;
        end case;
    end process;

end Behavioral;
```

**Figura 13.** Código VHDL generado para la cuenta.

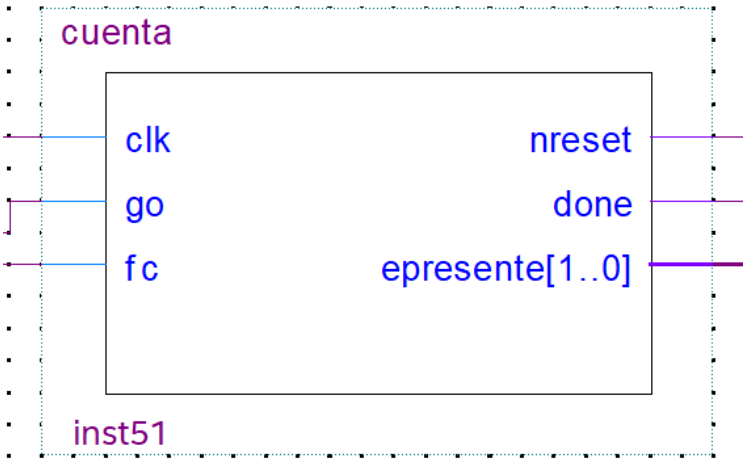


**Figura 15**. Bloque generado a partir del código VHDL.

En la asignación de pines, solo agregamos las 2 salidas adicionales, que indican los estados de la carta ASM en los que nos encontramos.

| Node Name | Direction | Location |
|---|---|---|
| out A | Output | PIN_A8 |
| out B | Output | PIN_A9 |
| in button | Input | PIN_B8 |
| out C | Output | PIN_A10 |
| in CLK | Input | PIN_P11 |
| out D | Output | PIN_B10 |
| out done | Output | PIN_D13 |
| out epresente[1] | Output | PIN_E14 |
| out epresente[0] | Output | PIN_C13 |
| in go | Input | PIN_C10 |

**Figura 16**. Asignación de pines para el proyecto complementario

## Resultados


**Fig 17.** SWITCH (GO) 0

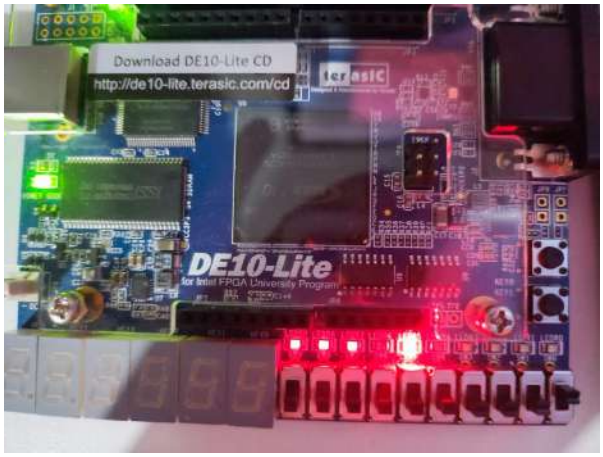



**Fig 17.** ESTADO (COUNT) 01

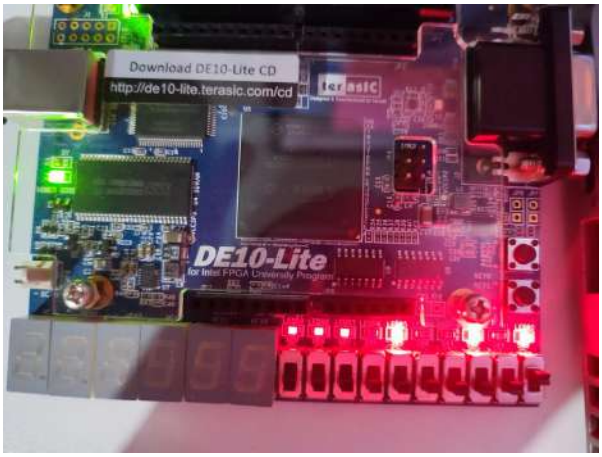
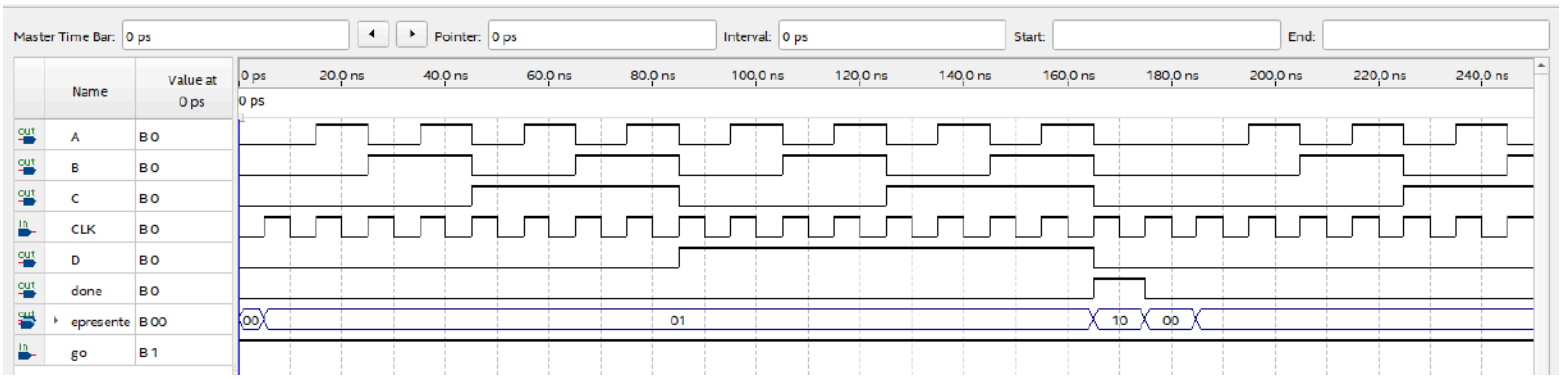



**Fig 17.** ESTADO (COUNT) 01- VALOR 101


**Fig 17.** ESTADO (DONE) 1 Y ESTADO (DONE Epresente) 10

## Simulación




En la simulación se puede ver que al inicio el sistema está en el estado START (epresente=00), y como la señal go está en nivel alto pasa al estado COUNT (epresente=01), donde el contador empieza a incrementar en sus salidas A–D de forma binaria. Una vez que el contador llega al valor máximo, la FSM cambia al estado DONE (epresente=10) y en ese momento se genera un pulso en la señal done indicando que terminó el conteo. Después de esto, el sistema regresa nuevamente a START (epresente=00) quedando listo para repetir el ciclo mientras la señal go siga activa.

## Conclusiones

### Jiménez Treviño Emilio Cristóbal

En esta práctica se logró implementar de manera satisfactoria la interpretación de la carta ASM tanto la de la práctica como la de la actividad complementaria, en esta práctica pudimos aprender a interpretar cada uno de los estados de la carta ASM obteniendo como resultado en el primero un contador en binario del 0 al 15 con estados de inicio y un fin para evitar que la secuencia se repita, por otra parte en la complementaria se implementó el uso de un botón para mostrar paso a paso cada número y cuando se activara el switch ya dejara de hacerlo al terminar la secuencia, esta práctica resultó complicada en la parte de la actividad complementaria por que fue recapitular lo visto en anteriores laboratorios de otras materias por lo que fue algo difícil, esto se pudo solucionar analizando por partes el diagrama dado por el profesor y yendo paso a paso.

### Martinez Perez Brian Erik

La práctica nos ayudó a entender el diseño e implementación de máquinas de estados finitos (FSM). Se logró una comprensión integral de cómo los diagramas de flujo de control, como las cartas ASM, se transforman en hardware digital. El uso de flip-flops tipo D permite visualizar la lógica de transición a nivel de componentes. La implementación de la máquina de estados en VHDL nos ayudo a transformar una secuencia de pasos a un circuito digital.

El código cuenta.vhd sirvió como un ejemplo práctico de cómo una FSM puede coordinar un ciclo de conteo. Esto validó la capacidad de VHDL para crear y simular algoritmos complejos que aseguran un comportamiento correcto del sistema.

### Robles Jimenez Marco Antonio

Esta práctica me permitió cerrar el ciclo completo entre el modelado y la implementación de una máquina de estados: partiendo de la carta ASM, simplifiqué la lógica con mapas de Karnaugh y traducirla a un circuito secuencial, validando en Quartus y en la DE10-LITE el flujo de control con las señales GO, CLK, DONE y el 74193 apoyado por el divisor de frecuencia. En la actividad complementaria, encapsular el control en el bloque cuenta (con los estados START–COUNT–DONE y la indicación explícita del estado presente) confirmó que VHDL no solo describe el algoritmo, sino que coordina de forma fiable la interacción entre la lógica combinacional y secuencial.

## Bibliografía

Laboratorio de Organización y Arquitectura de Computadoras. (2020, September 26). *Práctica No. 2: Diseño de máquinas de estado.*