FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
LUIS LEONARDO RIVERA ABAÚNZA Cod. 24 2004 2025

TAXÍMETRO DIGITAL EN VHDL

OBJETIVOS

- Implementar en VHDL un dispositivo de aplicación o uso cotidiano para observar la potencialidad del lenguaje VHDL y de los dispositivos FPGA's.
- Llevar a su implementación práctica en baquela o protoboard el circuito final del taxímetro mediante el uso de un sistema de desarrollo.

INTRODUCCIÓN

El presente trabajo pretende mostrar una aproximación que se ha hecho al problema en cuestión: una implementación del lenguaje VHDL que permita la aplicación a un problema práctico, tal cual es el de un taxímetro de manera eficiente, es decir utilizando la menor cantidad de recursos computacionales posibles y a la vez obtener la mejor velocidad posible. El uso de hardware para realizar esta tarea nos permite mejorar enormemente la velocidad de procesamiento y a su vez nos da independencia del uso del computador. En este trabajo hemos usado un FPGA (field programmable gate array) de la empresa Altera â Stratix. En esta primera parte llevamos a cabo la primera parte del proceso, la cual es el desarrollo del software para la aplicación.

El presente trabajo trata sobre el desarrollo de una solución al problema. Este trabajo está dividido como se presenta a continuación: la primera parte consta del fundamento teórico necesario para el desarrollo de la aplicación y se desarrollarán los programas que se necesitan para ello. En la segunda parte se explicará las alternativas al diseño y se programará sobre el dispositivo el software y se correrá para ver su funcionamiento.

MARCO TEÓRICO

FPGA

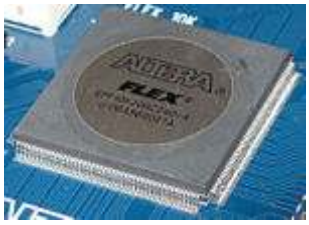


Figura 1. Una FPGA de Altera.

Figura 2. Una Spartan de Xilinx.

Una **FPGA** (del inglés *Field Programmable Gate Array*) es un dispositivo semiconductor que contiene bloques de lógica cuya interconexión y funcionalidad se puede programar. La lógica programable puede reproducir desde funciones tan sencillas como las llevadas a cabo por una puerta lógica o un sistema combinacional hasta complejos sistemas en un chip (w:en:System-on-a-chip).

Las FPGAs se utilizan en aplicaciones similares a los ASICs sin embargo son más lentas, tienen un mayor consumo de potencia y no pueden abarcar sistemas tan complejos como ellos. A pesar de esto, las FPGAs tienen las ventajas de ser reprogramables (lo que añade una enorme flexibilidad al flujo de diseño), sus costes de desarrollo y adquisición son mucho menores para pequeñas cantidades de dispositivos y el tiempo de desarrollo es también menor.

Ciertos fabricantes cuentan con FPGAs que sólo se pueden programar una vez, por lo que sus ventajas e inconvenientes se encuentran a medio camino entre los ASICs y las FPGAs reprogramables.

Históricamente las FPGAs surgen como una evolución de los conceptos desarrollados en las PLAs y los CPLDs

FPGAs vs CPLDs

Las FPGAs fueron inventadas en el año 1984 por Ross Freeman, co-fundador de Xilinx, y surgen como una evolución de los CPLDs.

Tanto los CPLDs como las FPGAs contienen un gran número de elementos lógicos programables. Si medimos la densidad de los elementos lógicos programables en puertas lógicas equivalentes (numero de puertas NAND equivalentes que podríamos programar en un dispositivo) podríamos decir que en un CPLD hallaríamos del orden de decenas de miles de puertas lógicas equivalentes y en una FPGA del orden de cientos de miles hasta millones de ellas.

Aparte de las diferencias en densidad entre ambos tipos de dispositivos, la diferencia fundamental entre las FPGAs y los CPLDs es su arquitectura. La arquitectura de los CPLDs es más rígida y consiste en una o más sumas de productos programables cuyos resultados van a parar a un número reducido de biestables síncronos (también denominados flip-flops). La arquitectura de las FPGAs, por otro lado, se basa en un gran número de pequeños bloques utilizados para reproducir sencillas operaciones

lógicas, que cuentan a su vez con biestables síncronos. La enorme libertad disponible en la interconexion de dichos bloques confiere a las FPGAs una gran flexibilidad.

Otra diferencia importante entre FPGAs y CPLDs es que en la mayoría de las FPGAs se pueden encontrar funciones de alto nivel (como sumadores y multiplicadores) embebidas en la propia matriz de interconexiones, así como bloques de memoria.

CARACTERÍSTICAS

Una jerarquía de interconexiones programables permite a los bloques lógicos de un FPGA ser interconectados según la necesidad del diseñador del sistema, algo parecido a un breadboard programable. Estos bloques lógicos e interconexiones pueden ser programados después del proceso de manufactura por el usuario/diseñador, así que el FPGA puede desempeñar cualquier función lógica necesaria.

Una tendencia reciente ha sido combinar los bloques lógicos e interconexiones de los FPGA con microprocesadores y periféricos relacionados para formar un «Sistema programable en un chip». Ejemplo de tales tecnologías híbridas pueden ser encontradas en los dispositivos Virtex-II PRO y Virtex-4 de Xilinx, los cuales incluyen uno o más procesadores PowerPC embebidos junto con la lógica del FPGA. El FPSLIC de Atmel es otro dispositivo similar, el cual usa un procesador AVR en combinación con la arquitectura lógica programable de Atmel. Otra alternativa es hacer uso de núcleos de procesadores implementados haciendo uso de la lógica del FPGA. Esos núcleos incluyen los procesadores MicroBlaze y PicoBlaze de Xlinx, Nios y Nios II de Altera, y los procesadores de código abierto LatticeMicro32 y LatticeMicro8.

Muchos FPGA modernos soportan la reconfiguración parcial del sistema, permitiendo que una parte del diseño sea reprogramada, mientras las demás partes siguen funcionando. Este es el principio de la idea de la «computación reconfigurable», o los «sistemas reconfigurables».

PROGRAMACIÓN

La tarea del programador es definir la función lógica que realizará cada uno de los **CLB**, seleccionar el modo de trabajo de cada **IOB** e interconectarlos.

El diseñador cuenta con la ayuda de entornos de desarrollo especializados en el diseño de sistemas a implementarse en un FPGA. Un diseño puede ser capturado ya sea como esquemático, o haciendo uso de un lenguaje de programación especial. Estos lenguajes de programación especiales son conocidos como HDL o *Hardware Description Language* (lenguajes de descripción de hardware). Los HDLs más utilizados son:

- VHDL
- Verilog
- ABEL
-

En un intento de reducir la complejidad y el tiempo de desarrollo en fases de prototipaje rápido, y para validar un diseño en HDL, existen varias propuestas y niveles de abstracción del diseño. Entre otras, National Instruments LabVIEW FPGA propone un acercamiento de programación gráfica de alto nivel.

APLICACIONES

Cualquier circuito de aplicación específica puede ser implementado en un FPGA, siempre y cuando esta disponga de los recursos necesarios. Las aplicaciones donde más comúnmente se utilizan los FPGA incluyen a los DSP (procesamiento digital de señales), radio definido por software, sistemas aeroespaciales y de defensa, prototipos de ASICs, sistemas de imágenes para medicina, sistemas de visión para computadoras, reconocimiento de voz, bioinformática, emulación de hardware de computadora, entre otras. Cabe notar que su uso en otras áreas es cada vez mayor, sobre todo en aquellas aplicaciones que requieren un alto grado de paralelismo.

Existe código fuente disponible (bajo licencia GNU GPL)[1] de sistemas como microprocesadores, microcontroladores, filtros, módulos de comunicaciones y memorias, entre otros. Estos códigos se llaman **cores**.

FABRICANTES

A principios de 2007, el mercado de los FPGA se ha colocado en un estado donde hay dos productores de FPGA de propósito general que están a la cabeza del mismo, y un conjunto de otros competidores quienes se diferencian por ofrecer dispositivos de capacidades únicas.

- Xilinx es uno de los dos grandes líderes en la fabricación de FPGA.
- Altera es el otro gran líder.
- Lattice Semiconductor lanzó al mercado dispositivos FPGA con tecnología de 90nm. En adición, Lattice es un proveedor líder en tecnología no volátil, FPGA basadas en tecnología Flash, con productos de 90nm y 130nm.
- Actel tiene FPGAs basados en tecnología Flash reprogrammable. También ofrece FPGAs que incluyen mezcladores de señales basados en Flash.
- QuickLogic tiene productos basados en antifusibles (programables una sola vez).
- Atmel es uno de los fabricantes cuyos productos son reconfigurables (el Xilinx XC62xx fue uno de estos, pero no están siendo fabricados actualmente). Ellos se enfocaron en proveer microcontroladores AVR con FPGAs, todo en el mismo encapsulado.
- Achronix Semiconductor tienen en desarrollo FPGAs muy veloces. Planean sacar al mercado a comienzos de 2007 FPGAs con velocidades cercanas a los 2GHz.
- MathStar, Inc. ofrecen FPGA que ellos llaman FPOA (Arreglo de objetos de matriz programable).

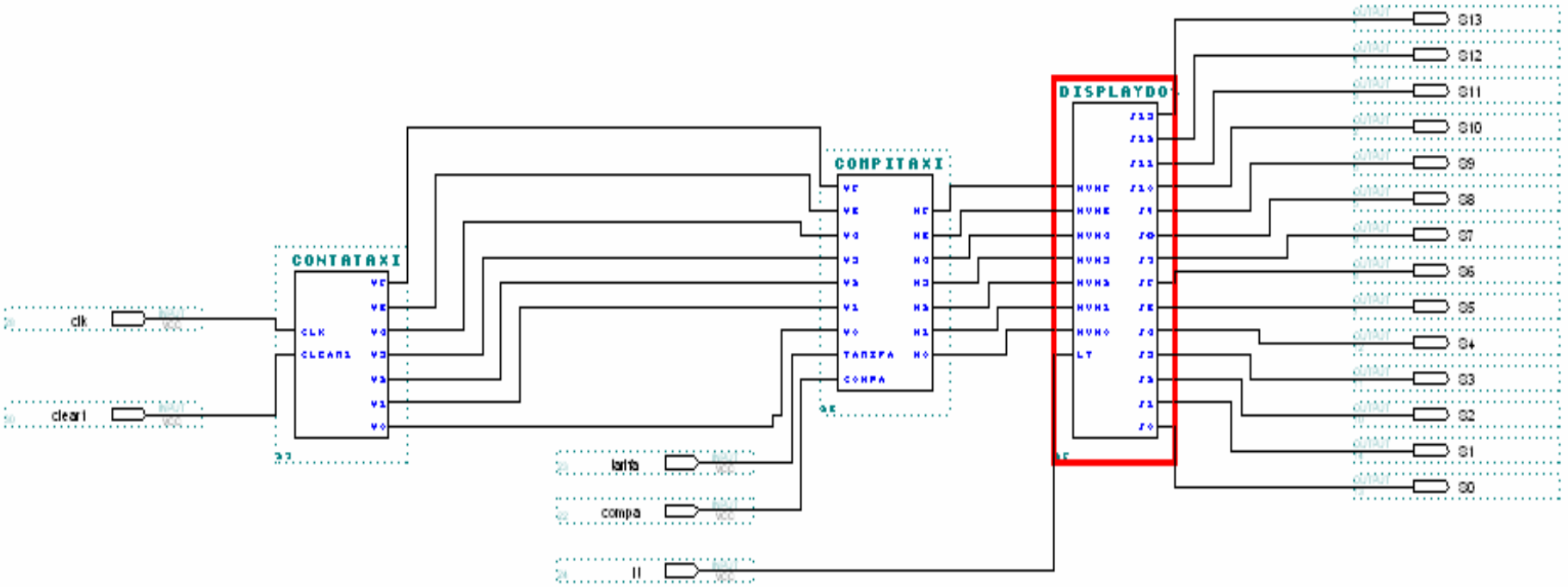
DESCRIPCIÓN DEL PROYECTO

Se pretende desarrollar mediante el uso de lenguaje VHDL sobre FPGA´s, un taxímetro funcional que nos permita dar una tarifa por carrera que dependa tanto del recorrido del viaje como del tiempo que el vehículo esté detenido sea por semáforos o por cualquier otra causa inherente al desempeño de la función.

El sistema debe resultar confiable y constar del mínimo de instrucciones para que sea lo más eficiente posible tanto en costos de materiales como en memoria de programa y desempeño.

En su primera parte, es decir para el primer corte, se desarrollará el software para la aplicación y se elegirá el hardware con el que se trabajará y sobre el que se implementará este software. La elección del software estará dada tanto por la disponibilidad del mismo como por los costos y la disponibilidad del hardware que este requiere para su programación.

CIRCUITO FINAL EN MAXPLUSII



CONVENCIONES DEL DIAGRAMA DE BLOQUES ANTERIOR

- CLEAR: Da inicio al proceso de conteo del taxímetro y habilita los contadores y temporizadores.

- **TARIFA** : Se define qué tipo de carrera se desea cargar, si es de horas extras o de horario normal.

- **CLOCK** : Mediante un potenciómetro o un interruptor se selecciona la velocidad de temporización lo que simula el recorrido normal del taxi. Si se encuentra ante un semáforo o una parada se lleva el potenciómetro a su valor más alto de resistencia lo que simula el conteo en parada del taxi. Cuadrando el ciclo de temporización se puede simular un incremento de 50 pesos cada 50 metros o cada cuanto se desee.

- **CONTADOR** : Inicia el conteo a partir de un banderazo, que en caso de Ibagué es de 1200. Incrementa 50 cada que encuentra un flanco de subida del reloj. Llega hasta 10000 que es la tarifa máxima para la ciudad.

- **COMPARADOR** : Compara si es la tarifa mayor o menor a la mínima y dependiendo de esto da el resultado de la tarifa final. Si es menor a la mínima, por defecto brinda esta tarifa que es de 2600 para carrera NORMAL y de 3000 para EXTRA. Si es mayor a la mínima, 2600 brinda el valor del contador si es carrera NORMAL y lo que de el contador más 400 para EXTRA.También tiene en cuenta, que mientras el reloj esté funcionando el comparador está inactivo y sólo se activa y hace la comparación al final de la carrera

- **BINARIO A 7 SEGEMENTOS**: Se le asigna directamente un código n binario que va al display y va permi9tiendo la visualización del incremento de la carrera en los display’s y al final el valor total de la misma. Algoritmo que convierte de decimal a BCD este mismo código.

- **DISPLAY** : Se visualiza el valor y el incremento constante de la carrera mediante cuatro display´s de 0 a 999

CÓDIGOS EN VHDL DE LOS COMPONENTES DEL PROYECTO

CONTADOR

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

entity contataxi is
  port (clk    : in std_logic; --Reloj
        clear1 : in std_logic;
        y6,y5,y4,y3,y2,y1,y0 : out std_logic); --Salida
end contataxi;

architecture beh of contataxi is
signal cuenta : std_logic_vector (6 downto 0);
signal y :std_logic_vector (6 downto 0);
begin

  output: process (clk,clear1)

   BEGIN

    if (clear1='0') then
      y<="0000000";
      cuenta<="0001100";
                    elsif (clk'event and clk='1') then
       cuenta<=(cuenta+1);
                  y<=cuenta;
    end if;

y6<=cuenta(6);
y5<=cuenta(5);
y4<=cuenta(4);
y3<=cuenta(3);
y2<=cuenta(2);
y1<=cuenta(1);
y0<=cuenta(0);

  end process;

end beh;
```

COMPARADOR

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

entity compitaxi is
        port(Y6,Y5,Y4,Y3,Y2,Y1,Y0: in std_logic;
                tarifa: in std_logic;
                compa: in std_logic;
                n6,n5,n4,n3,n2,n1,n0 : out std_logic);
                end compitaxi;

architecture compconv of compitaxi is

signal bin:std_logic_vector(6 downto 0);
signal entrada:integer range 0 to 64;
signal num1 : std_logic_vector (6 downto 0);

begin

bin<=Y6&Y5&Y4&Y3&Y2&Y1&Y0;entrada <=conv_integer(bin);

process(entrada,tarifa)

begin

if compa='1' then
num1<=bin;

else

if bin>"0011010" and tarifa='0' then
      num1<=(bin);
 elsif bin>"0011010" and tarifa='1' then
      num1<=(bin);
                num1<=bin+4;
 elsif bin<"0011010"  and tarifa='0' then
                num1<="0011010" ;
          --    num1<=num1;
 elsif bin<="0011010" and tarifa='1' then
                num1<="0011110" ;
         --     num1<=(num1+40);

end if;
end if;
n6<=num1(6);
```

```
n5<=num1(5);
n4<=num1(4);
n3<=num1(3);
n2<=num1(2);
n1<=num1(1);
n0<=num1(0);

end process;
end compconv;
```

BINARIO A 7 SEGMENTOS

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity DISPLAYDOS is
        port(num6,num5,num4,num3,num2,num1,num0,lt: in std_logic;
        S13,S12,S11,S10,S9,S8,S7,S6,S5,S4,S3,S2,S1,S0:out std_logic);--a b c d e f g
end DISPLAYDOS;

architecture codificador of DISPLAYDOS is
signal bcd: std_logic_vector(6 downto 0);
signal entrada: integer range 0 to 64;
signal salida:std_logic_vector(13 downto 0);
begin

bcd<=num6&num5&num4&num3&num2&num1&num0;entrada <=conv_integer(bcd);
process(entrada,lt)

begin

        if lt= '1' then
    case entrada is
        when 12=>salida<="01100001101101";
                        when 13=>salida<="01100001111001";
                        when 14=>salida<="01100000110011";
                        when 15=>salida<="01100001011011";
        when 16=>salida<="01100000011111";
                        when 17=>salida<="01100001110000";
                        when 18=>salida<="01100001111111";
                        when 19=>salida<="01100001110011";
                        when 20=>salida<="11011011111110";
        when 21=>salida<="11011010110000";
                        when 22=>salida<="11011011101101";
                        when 23=>salida<="11011011111001";
                        when 24=>salida<="11011010110011";
                        when 25=>salida<="11011011011011";
                        when 26=>salida<="11011010011111";
```

```
                    when 27=>salida<="11011011110000";
                    when 28=>salida<="11011011111111";
                    when 29=>salida<="11011011110011";
                    when 30=>salida<="11110011111110";
        when 31=>salida<="11110010110000";
                    when 32=>salida<="11110011101101";
                    when 33=>salida<="11110011111001";
                    when 34=>salida<="11110010110011";
                    when 35=>salida<="11110011011011";
        when 36=>salida<="11110010011111";
                    when 37=>salida<="11110011110000";
                    when 38=>salida<="11110011111111";
                    when 39=>salida<="11110011110011";
                    when 40=>salida<="01100111111110";
        when 41=>salida<="01100110110000";
                    when 42=>salida<="01100111101101";
                    when 43=>salida<="01100111111001";
                    when 44=>salida<="01100110110011";
                    when 45=>salida<="01100111011011";
        when 46=>salida<="01100110011111";
                    when 47=>salida<="01100111110000";
                    when 48=>salida<="01100111111111";
                    when 49=>salida<="01100111110011";
                    when 50=>salida<="10110111111110";

                    when others =>salida<="11111111111110";
            end case;
        end if;

S13<=salida(13);
S12<=salida(12);
S11<=salida(11);
S10<=salida(10);
S9<=salida(9);
S8<=salida(8);
S7<=salida(7);
S6<=salida(6);
S5<=salida(5);
S4<=salida(4);
S3<=salida(3);
S2<=salida(2);
S1<=salida(1);
S0<=salida(0);

end process;
end codificador;
```

CONSIDERACIONES DE DISEÑO

Para el diseño del taxímetro en VHDL se tuvieron en cuenta varias cosas muy importantes, a saber:

- El diseño debía corresponder con la realidad de los taxímetros circulantes, esto es, valor del banderazo, incrementos, tarifas, etc.
- Se debía iniciar en 1200 que es valor del banderazo actual y comenzar a contar a partir de este momento hasta máximo 10000 que es el valor máximo de una carrera en Ibagué.
- Se debía tener en cuenta el valor diferencial entre una carrera en horario normal y una carrera en horas extras.
- El incremento cada ciclo de reloj sería igual al de un taxí cada 100 metros que es lo estándar, es decir, 50 pesos cada vez.
- Un comparador debía indicar si la carrera era menor al valor de la mínima, 2600, y era horario normal, que debía cobrarse este valor por defecto y si era horario extra este mismo incrementando 400. Si el valor del taxímetro marcaba más de este mínimo debía mostrar el valor que aparecía en los display's como valor final si era horario normal y este valor más 400 si era extra.
- Debía contar con un botón que hiciera la comparación al final de la carrera para que se pudiera ver el incremento durante el recorrido de la misma.
- El programa debía ser implementado sobre una FPGA de ALTERA, con las compuertas y demás dispositivos necesarios, pero, además debía resultar compacto, de bajo costo y mínimo número de componentes extras.

ANÁLISIS DE RESULTADOS

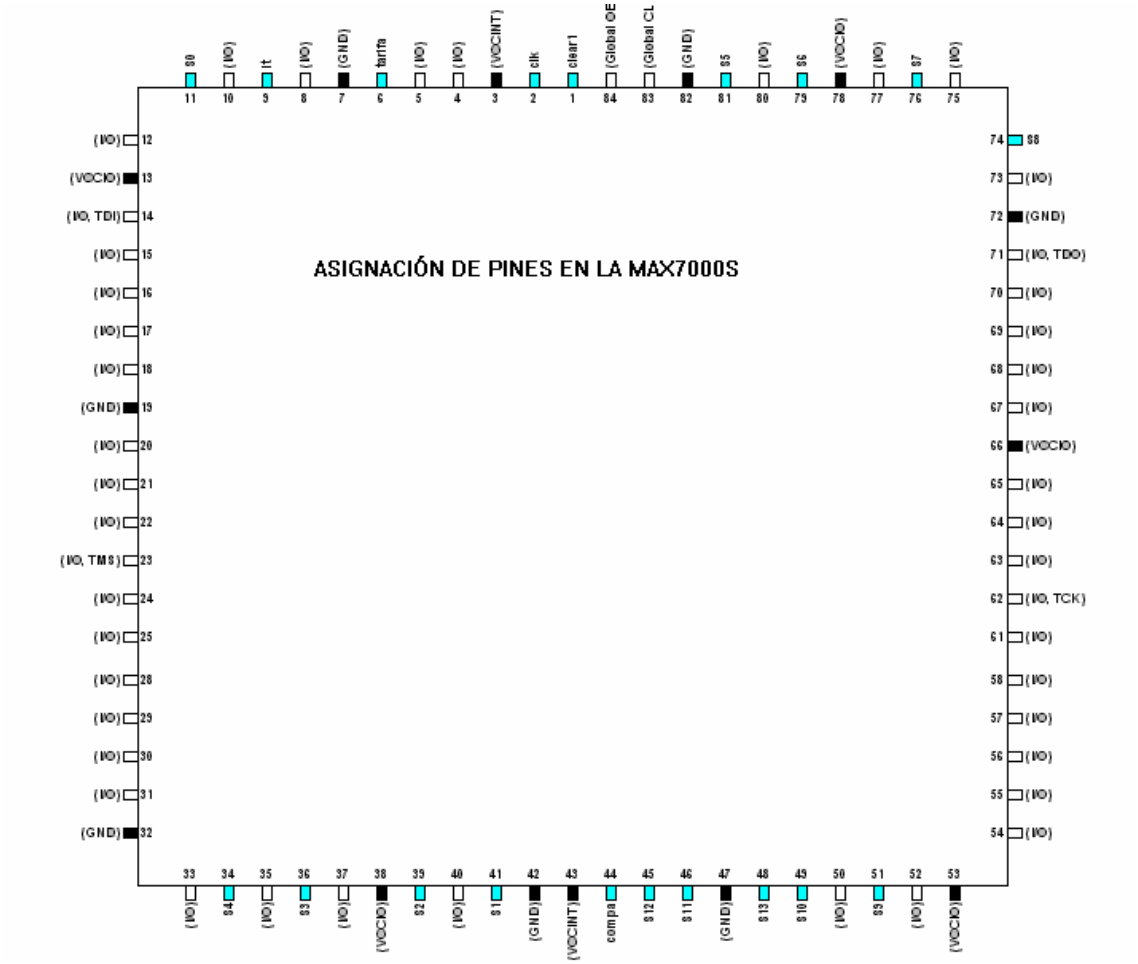
Al llegar a la implementación del programa sobre la FPGA y su sistema de desarrollo, llegamos a las siguientes consideraciones:

- El sistema FPGA, resultó muy frágil y se descompuso al hacer las primeras prácticas y desarrollos sobre él.
- En vista de esto y como no se contaba con la capacidad económica para conseguir otra FPGA, se pensó en la utilización de un CPLD para llevar a cabo el mismo fin.
- Al trabajar sobre un CPLD, que tiene mucha menos capacidad que una FPGA hubo que bastantes depuraciones al programa original, entre las que se encuentran principalmente las siguientes:

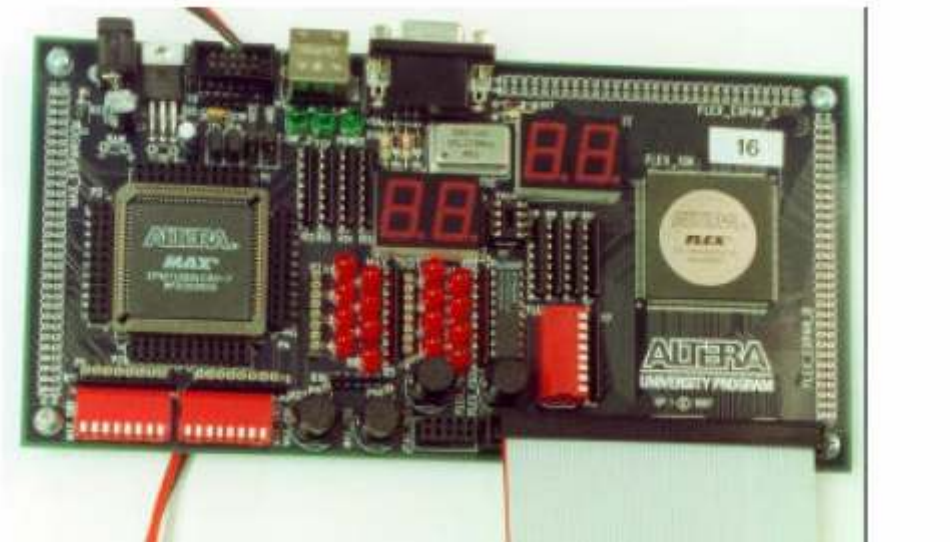
- Se puso el incremento de la tarifa de 100 en 100, con el fin de que de esta manera a partir de 1200 los dos últimos display's no iban a cambiar sino que permanecerían en 0 lo que nos simplificaba líneas de código pues los podíamos mandar directamente a tierra y Vcc y sólo necesitamos un interruptor para controlarlos.
- Se utilizó el temporizador externo, mediante un 555 con el mismo fin, depuración.
- Se eliminaron algunas variables de comparación y solamente se dejaron para el final del programa cuando se hace la comparación antes de dar el resultado del valor total de la carrera.
- Se hizo el incremento del valor de la carrera solamente hasta 8000 con el mismo objetivo, pues de esta manera reduciamos el número de bits necesarios para el conteo.
- Al reducir de esta manera el conteo y la comparación bajamos de 14 bits a 7 bits los que utilizaríamos en el programa, lo cual significa una reducción inmensa en el peso del código total.

- Al llevar a cabo la compilación y programación de la tarjeta CPLD, nos dimos cuenta que había necesidad de hacer primero una asignación de pines para asegurarnos de trabajar con el jumper 1, que es el que se puede utilizar sin activar el puerto JTAG.
- Al momento de montar sobre baquela se presentaron algunos problemas por culpa del cable paralelo que iba del sistema de desarrollo hasta ésta.
- También crearon conflicto los terminales al ser insertados en la baquela o en el protoboard.
- Se presentó un inconveniente al usar el Vcc y la tierra de la tarjeta pues el fan-out de ésta era muy bajo lo que hacía que se nos cayera la corriente en todo nuestro circuito imposibilitando su funcionamiento.
- Al final tocó cambiar el cable y darle alimentación aparte a todos los componentes que no dependieran directamente de la tarjeta.

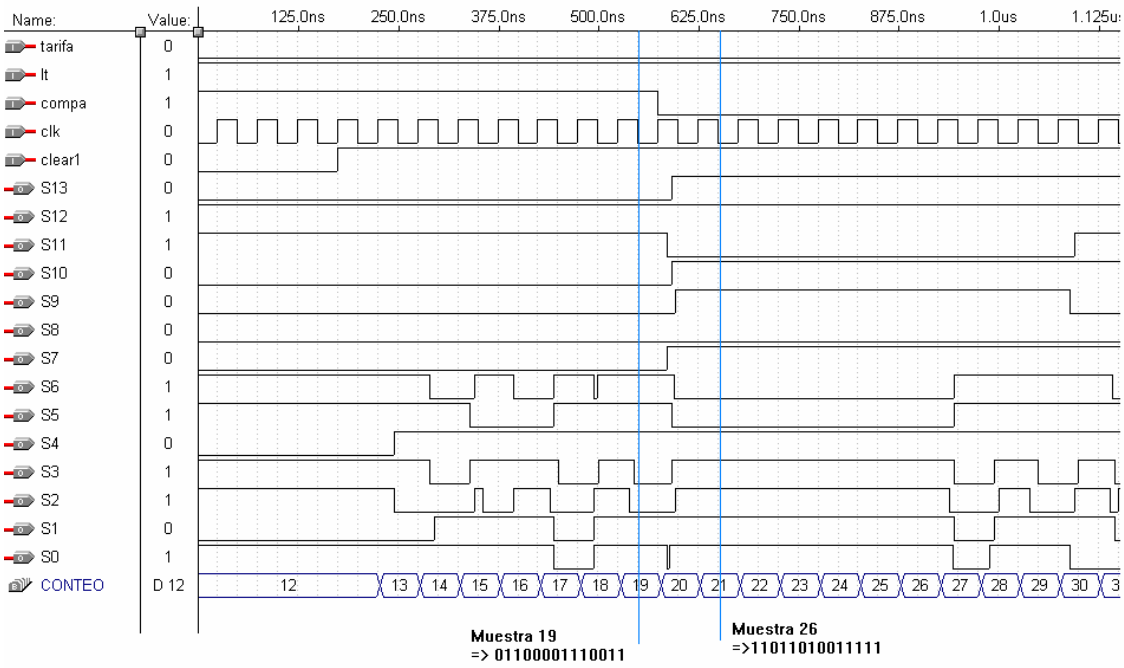
ASIGNACIÓN DE PINES EN LA TARJETA MAX7000S



SISTEMA DE DESARROLLO ANTERIORMENTE A UTILIZAR



SIMULACIONES DEL TAXÍMETRO DIGITAL

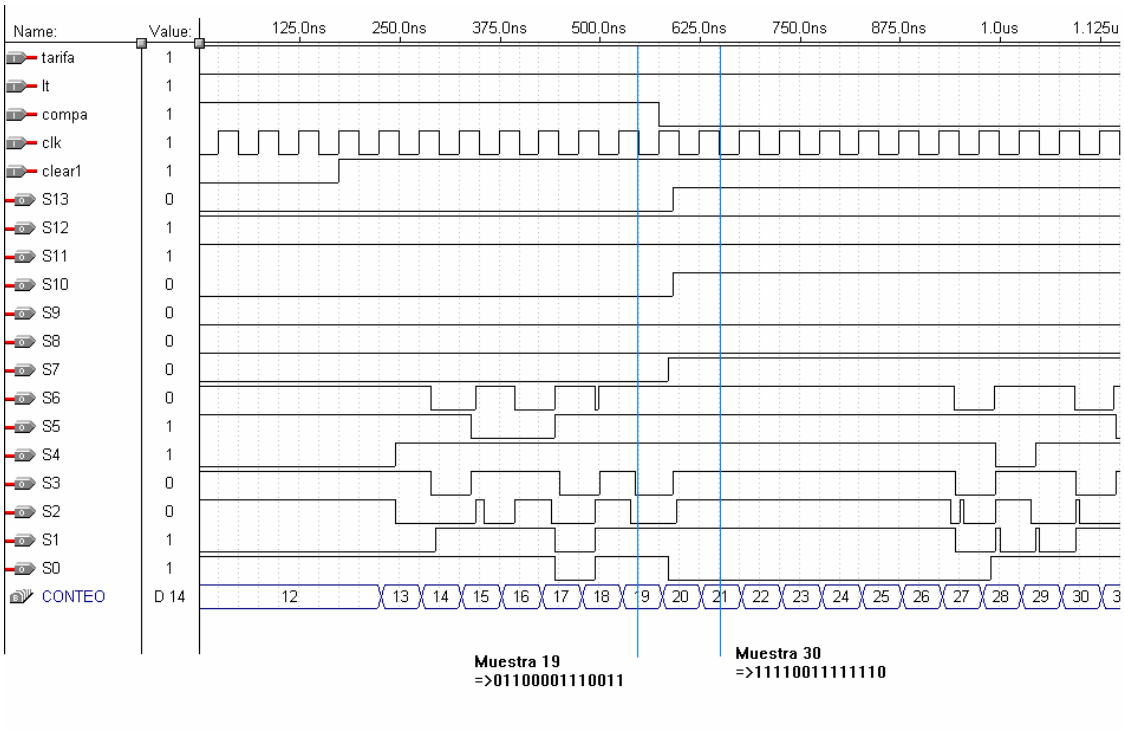


lt : activo en **alto** activa los displays
tarifa : activa en **bajo** cobra carrera **normal**
compa : activo en **alto** no hace comparación hasta que se pone en **baj**o.
Clear1 : mientras esté en **bajo** el **CONTEO** estará en **12** que es el valor del banderazo. Cuando se ponga en alto comenzará a contar a partir de **12**.

Los segmentos del display se activan en **alto** por lo que podemos ver cómo cambian de acuerdo a cada una de las condiciones anteriores.

Mientras esté en **bajo tarifa**, **compa** esté en **alto** y **CONTEO** sea **menor** a **26** => va a mostrar **26** en los displays.

Este valor se multiplica por **100** ya que, como el conteo incrementa de a 100 los dos últimos displays siempre mostrarán 00, entonces se mandan directamente a Vcc, excepto el segmento G que se deja N.C. Esto con el fin de depurar el código del programa.

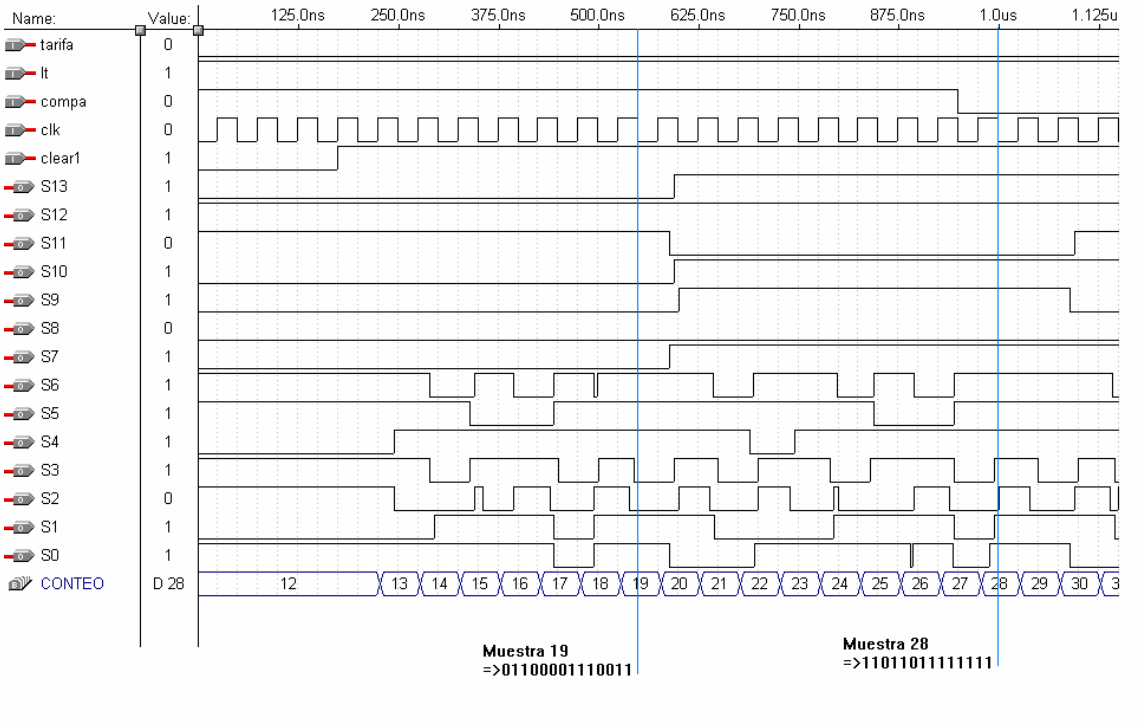


lt : activo en **alto** activa los displays
tarifa : activa en **alto** cobra carrera **extra**
compa : activo en **alto** no hace comparación hasta que se pone en **bajo**.
Clear1 : mientras esté en **bajo** el **CONTEO** estará en **12** que es el valor del banderazo. Cuando se ponga en alto comenzará a contar a partir de **12**.

Los segmentos del display se activan en **alto** por lo que podemos ver cómo cambian de acuerdo a cada una de las condiciones anteriores.

Mientras esté en **alto tarifa**, **compa** esté en **alto** y **CONTEO** sea **menor** a **26** => va a mostrar **30** en los displays.

Este valor se multiplica por **100** ya que, como el conteo incrementa de a 100 los dos últimos displays siempre mostrarán 00, entonces se mandan directamente a Vcc, excepto el segmento G que se deja N.C. Esto con el fin de depurar el código del programa.

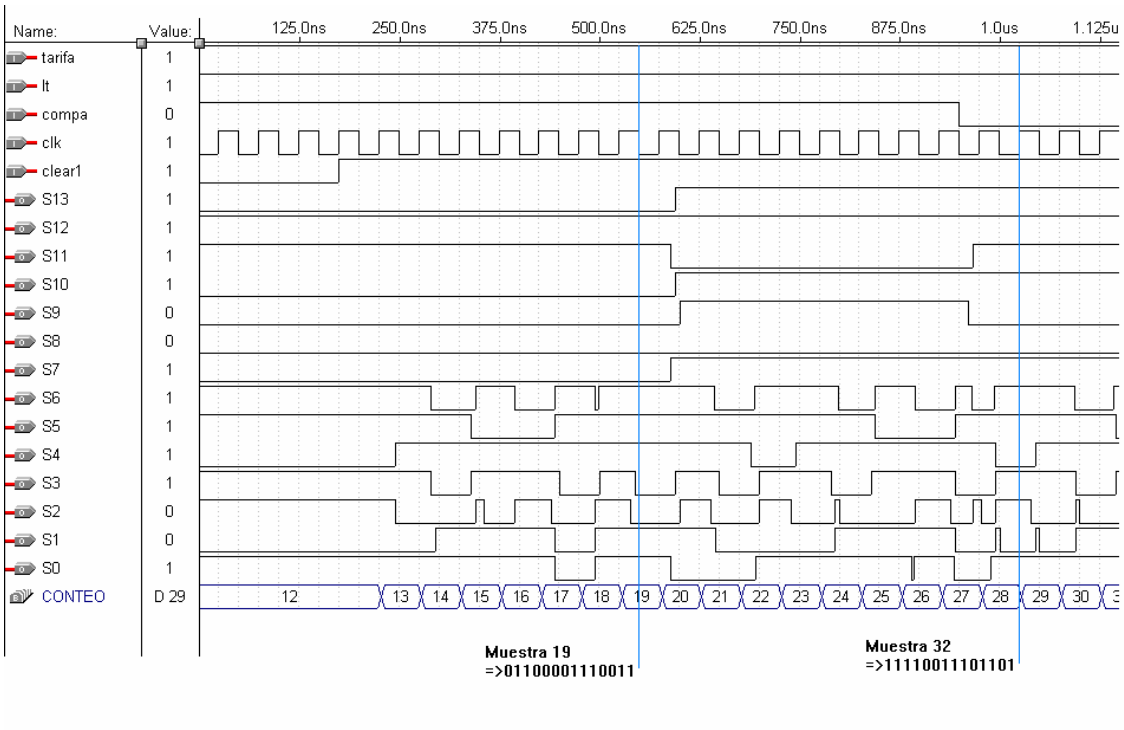


lt : activo en **alto** activa los displays
tarifa : activa en **bajo** cobra carrera **normal**
compa : activo en **alto** no hace comparación hasta que se pone en **bajo**.
Clear1 : mientras esté en **bajo** el **CONTEO** estará en **12** que es el valor del banderazo. Cuando se ponga en alto comenzará a contar a partir de **12.**

Los segmentos del display se activan en **alto** por lo que podemos ver cómo cambian de acuerdo a cada una de las condiciones anteriores.

Mientras esté en **bajo tarifa, compa** esté en **alto** y **CONTEO** sea **mayor** a **26** => va a mostrar el valor de **CONTEO** en los displays.

Este valor se multiplica por **100** ya que, como el conteo incrementa de a 100 los dos últimos displays siempre mostrarán 00, entonces se mandan directamente a Vcc, excepto el segmento G que se deja N.C. Esto con el fin de depurar el código del programa.



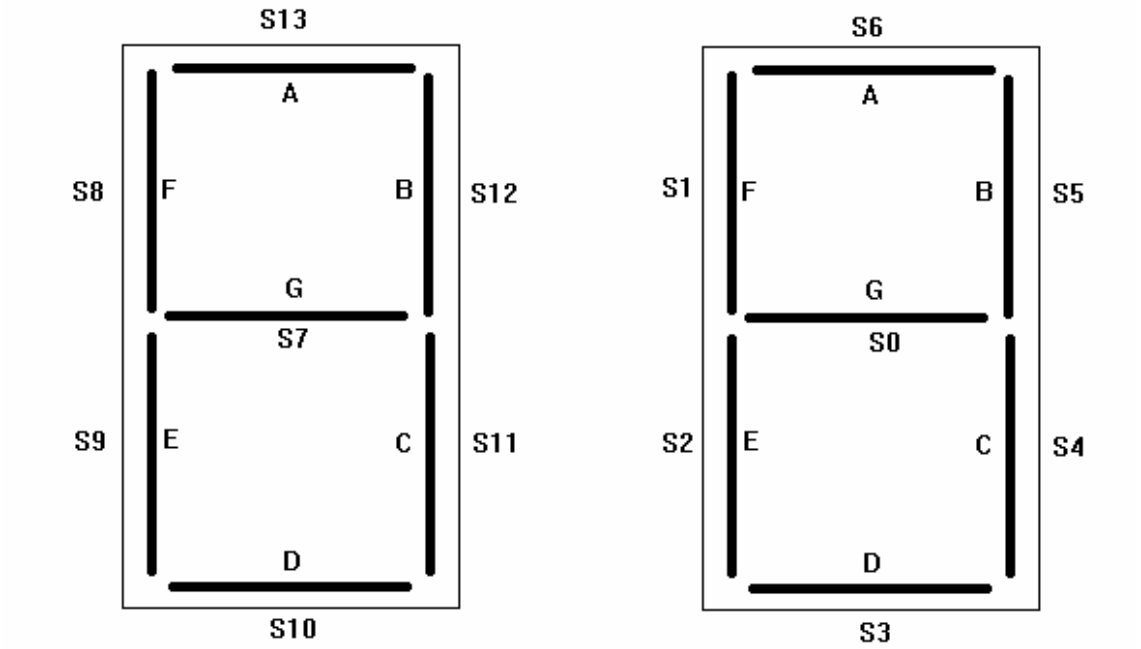
lt : activo en **alto** activa los displays
tarifa : activa en **alto** cobra carrera **extra**
compa : activo en **alto** no hace comparación hasta que se pone en **bajo**.
Clear1 : mientras esté en **bajo** el **CONTEO** estará en **12** que es el valor del banderazo. Cuando se ponga en alto comenzará a contar a partir de **12**.

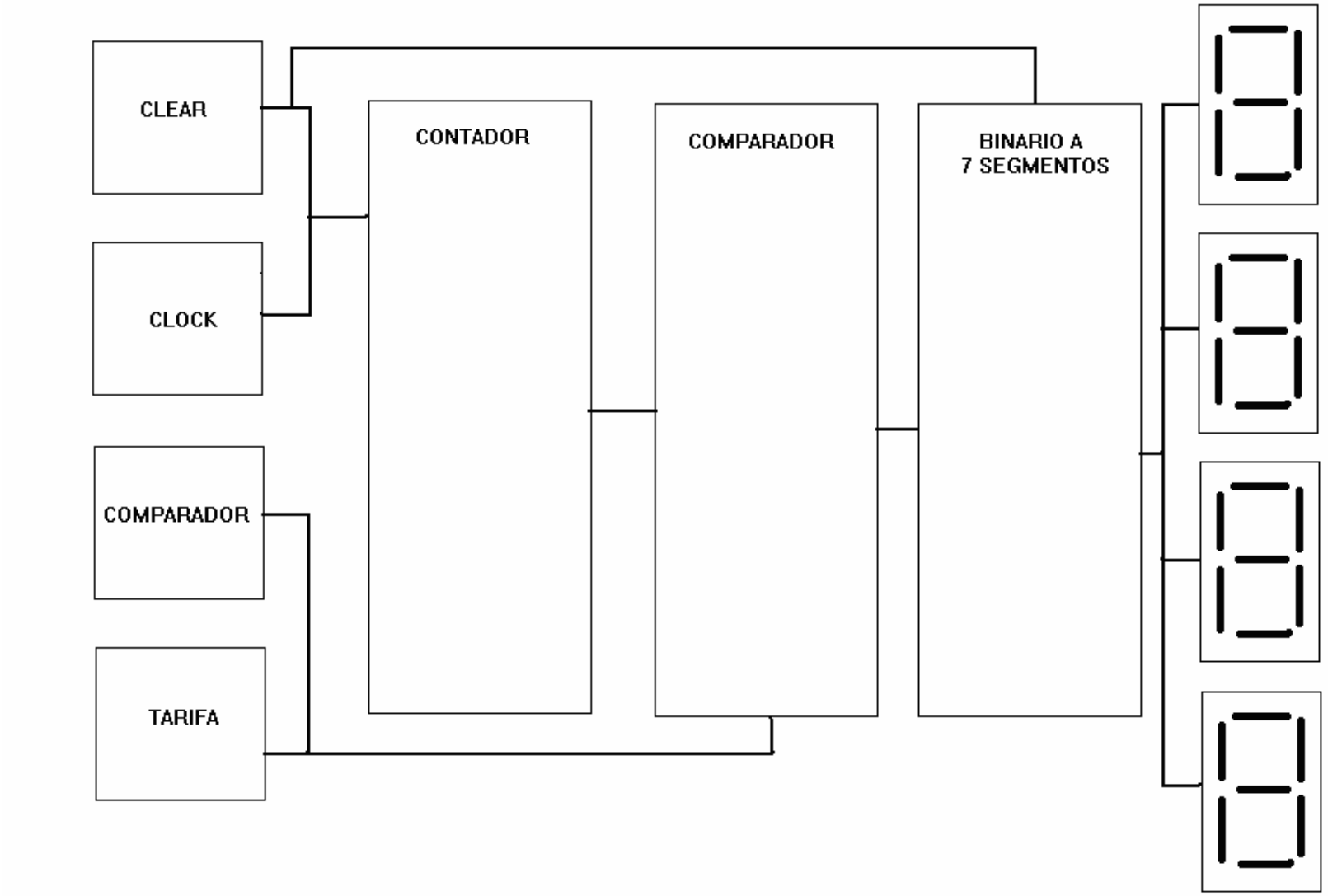
Los segmentos del display se activan en **alto** por lo que podemos ver cómo cambian de acuerdo a cada una de las condiciones anteriores.

Mientras esté en **alto tarifa, compa** esté en **alto** y **CONTEO** sea mayor a **26** => va a mostrar el valor de **CONTEO + 4** en los displays.

Este valor se multiplica por **100** ya que, como el conteo incrementa de a 100 los dos últimos displays siempre mostrarán 00, entonces se mandan directamente a Vcc, excepto el segmento G que se deja N.C. Esto con el fin de depurar el código del programa.

ASIGNACIÓN DE BITS A CADA UNO DE LOS SEGMENTOS DE LOS DISPLAYS CONTROLADOS MEDIANTE EL CPLD




CLEAR
CLOCK
COMPARADOR
TARIFA
CONTADOR
COMPARADOR
BINARIO A
7 SEGMENTOS