República Bolivariana de Venezuela

Universidad de Carabobo

Facultad de Ingeniería

Escuela de Ingeniería Eléctrica y de Telecomunicaciones

Departamento de Sistemas y Automática

Cátedra de Lógica Digital

**Práctica # 2 – Sesión Única**

**Objetivo: Implementar circuitos combinacionales con salidas múltiples utilizando VHDL y la tarjeta de desarrollo BASYS2**

Sección #05 de Laboratorio

**Integrantes:**

Carlos Hernández

C.I.: 25.829.471

Gianfranco Gasbarri

C.I.: 26.654.860

Fecha de entrega: 12/02/19

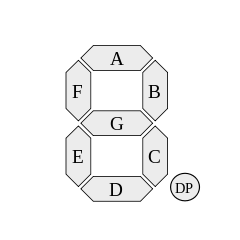
**Pre-Laboratorio**

1. **Diseño del conversor CDB-7Seg:**
   1. **Construya la tabla de la verdad para cada segmento, obtenga la expresión simplificada y su representación con compuertas lógicas:**

Ver **Anexo 1**

**Nota:** se seguirá la convención universal para la numeración de

cada uno de los segmentos, tal como se indica en la siguiente figura



* 1. **Dibuje y simule el diseño del conversor utilizando LogicWorks. Imprima el diseño:**

Ver **Anexo 2**

* 1. **Diseño en Active HDL de los componentes. Imprima el diseño:**

Ver **Anexo 3**

**Post-Laboratorio**

* **Diagramas de bloque, programa VHDL, simulación y tablas de la verdad (Lab#2):**

Ver **Anexo 3 y 4**

* **Diagrama de bloque y programas VHDL (Lab #3 y #4)**

Ver **Anexo 5 y 6**

**Conclusiones**

En la presente práctica, se desarrollaron varios componentes correspondientes a los conversores de números binarios de 4 bits (ya sea correspondiente a una palabra BCD válida o a cualquiera de los símbolos del sistema hexadecimal) a un display 7 segmentos.

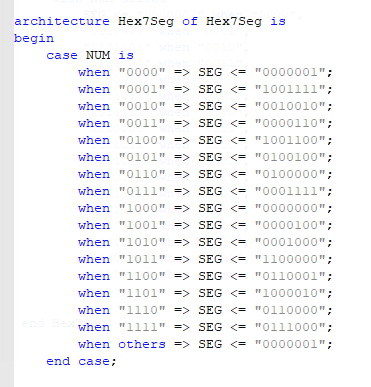
Al diseñar el primero de ellos, pudimos observar que al proporcionar una entrada entre 0 y 9, los segmentos seguían el comportamiento que se definió al armar la función asignando manualmente sus valores en la tabla de la verdad. Pudimos observar que el componente del inversor tenía como salidas 7 funciones lógicas distintas, cada una de ellas dependiendo de las entradas D-C-B-A.

Además, nos percatamos de que en varias de dichas funciones, se podía duplicar alguna expresión, la cual podíamos aprovechar para reutilizar varias veces, simplificando de manera significativa el diseño. Al hacer esto, las combinaciones restantes que no representan una palabra BCD válida (10-15) generaban salidas diferentes a las generadas cuando no se simplificaba. Esto se debe a que fueron definidas como **don’t care** y se adaptaron al proceso de simplificación automático.

El diseño realizado con el software LogicWorks y el descrito con VHDL comparten varias similitudes. En ambos, las entradas y las salidas son las mismas, y al realizar las simulaciones las salidas fueron las esperadas para los valores definidos (del 0 al 9). Sin embargo, para las salidas correspondientes a las combinaciones restantes variaban debido a que en el diseño descrito con VHDL, las mismas se establecieron de manera que pintaran un guión (“-“). Esto quiere decir: los segmentos del A al F en 1 y el G en 0.

Otra diferencia notable en el momento de realizar el diseño, es que en el software LogicWorks se debió representar cada uno de los circuitos lógicos para las 7 funciones, lo cual representa una actividad laboriosa. En cambio, la definición por VHDL simplificaba esto, ya que simplemente se debía expresar las 7 salidas concatenadas para cada uno de las posibles entradas.

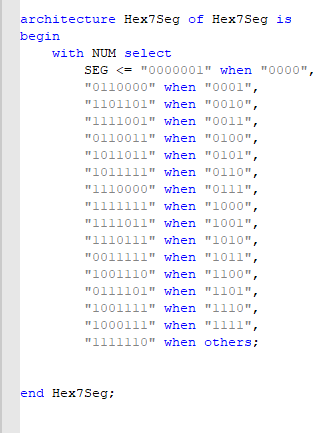
También podemos definir la arquitectura del circuito utilizando las sentencias **case/when**, de la siguiente manera:



Comparando esta manera con la anterior, para este caso es un poco más práctica la anterior ya que simplemente definíamos una a una las salidas. En cambio con ésta, debemos seleccionar una variable o una condición y evaluarla para cada uno de sus posibles valores, pero como debemos ir 1 a 1, no es muy práctico. En el caso en el que con una misma condición se afectasen varias salidas, sí sería más conveniente utilizar la sentencia **case/when.**

Se realizaron dos versiones del componente **Hex7Seg**: en la primera de ellas, mostrábamos las salidas de los segmentos en todos los displays independientemente del uso de los botones de la tarjeta BASYS. En la segunda versión, tomábamos en cuenta los 4 botones de la tarjeta para controlar la visualización de los segmentos en los displays. En ambos diseños, se logró construir los conversores con las salidas correctas para cada una de las combinaciones de la tabla de la verdad, con el debido cuidado de respetar el nombre de las señales de entrada y salida para la segunda versión del componente.

En el caso en que deseáramos diseñar un conversor de hexadecimal a displays 7 segmentos del tipo **cátodo común**, bastaría con intercambiar los diferentes valores que tendrá la salida SEG en la definición de la arquitectura en el código VHDL, intercambiando los 0’s con 1’s. Tal como se muestra a continuación:



Esto se debe a que en los displays 7 segmentos de este tipo, cada uno de los segmentos se enciende con un 1 lógico, a diferencia del tipo **ánodo común**.

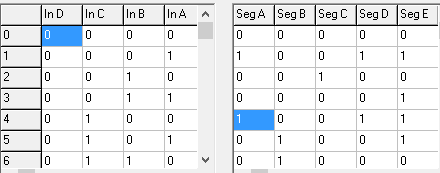
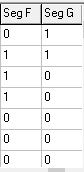
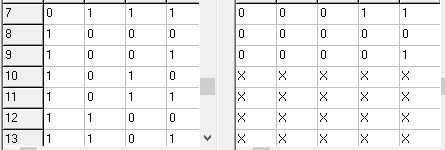
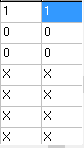
En el diseño con compuertas lógicas, debemos obtener cada una de las funciones lógicas tomando en cuenta los valores de la tabla de la verdad en los que cada uno de los segmentos se enciende o no (tomando en cuenta que esta vez se encienden con un 1 lógico). A partir de dichas funciones lógicas, podremos construir los circuitos para cada uno de los segmentos. Otra solución que se puede tomar es reutilizar el componente construido para un **ánodo común** y simplemente invertir cada una de las salidas con una compuerta **NOT**, para así obtener los resultados deseados.

Pudimos observar, mediante la realización de las múltiples actividades en esta práctica, que definir el comportamiento de un circuito lógico de múltiples salidas de la misma naturaleza mediante el lenguaje VHDL es muy sencillo, puesto que simplemente asignamos valores para cada una de dichas salidas mediante una concatenación para cada una de las entradas. Esto resulta ventajoso cuando manejamos pocas entradas, puesto que a medida de que aumentan las entradas, dichas asignaciones se duplican sucesivamente.

**ANEXOS**

**Anexo 1**

**Anexo 1.1**

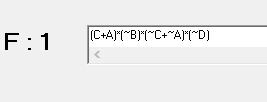
**Tabla de la verdad BCD7Seg**

**Anexo 1.2**

**Segmento A**

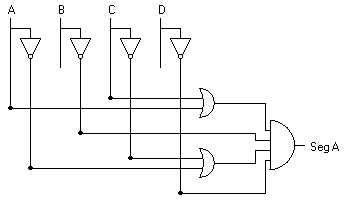
**Anexo 1.2.1**

**Expresión POS simplificada**



**Anexo 1.2.2**

**Representación con compuertas lógicas**

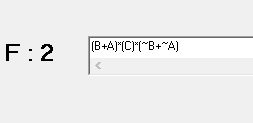


**Anexo 1.3**

**Segmento B**

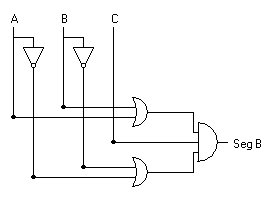
**Anexo 1.3.1**

**Expresión POS simplificada**



**Anexo 1.3.2**

**Representación con compuertas lógicas**

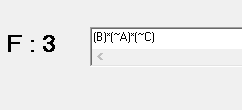


**Anexo 1.4**

**Segmento C**

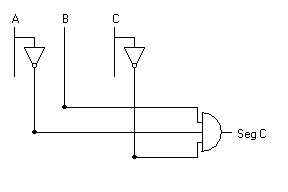
**Anexo 1.4.1**

**Expresión POS simplificada**



**Anexo 1.4.2**

**Representación con compuertas lógicas**

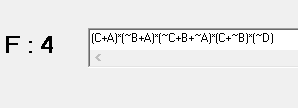


**Anexo 1.2**

**Segmento D**

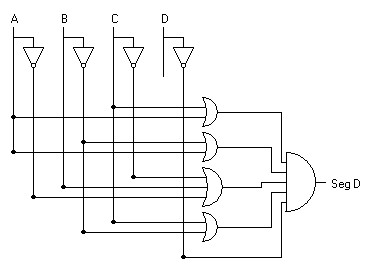
**Anexo 1.5.1**

**Expresión POS simplificada**



**Anexo 1.5.2**

**Representación con compuertas lógicas**

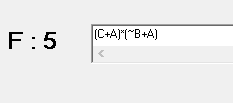


**Anexo 1.6**

**Segmento E**

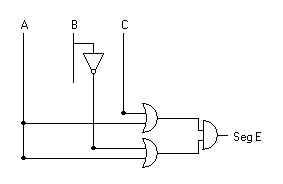
**Anexo 1.6.1**

**Expresión POS simplificada**



**Anexo 1.6.2**

**Representación con compuertas lógicas**

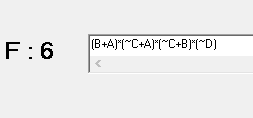


**Anexo 1.7**

**Segmento F**

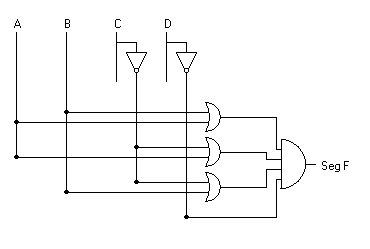
**Anexo 1.7.1**

**Expresión POS simplificada**



**Anexo 1.7.2**

**Representación con compuertas lógicas**

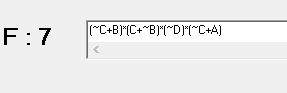


**Anexo 1.8**

**Segmento G**

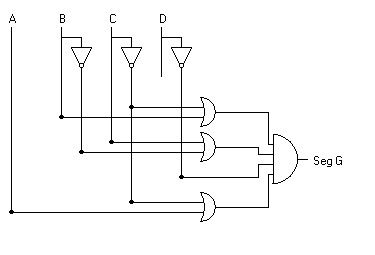
**Anexo 1.8.1**

**Expresión POS simplificada**



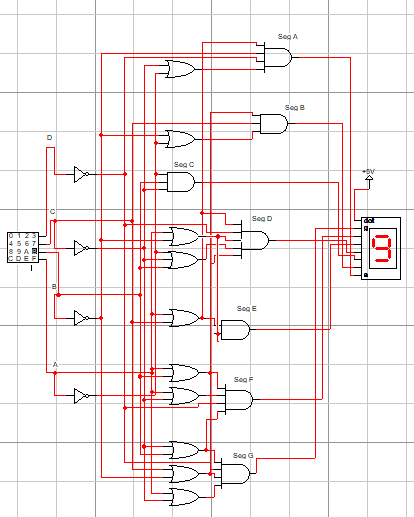
**Anexo 1.2.2**

**Representación con compuertas lógicas**



**Anexo 2**

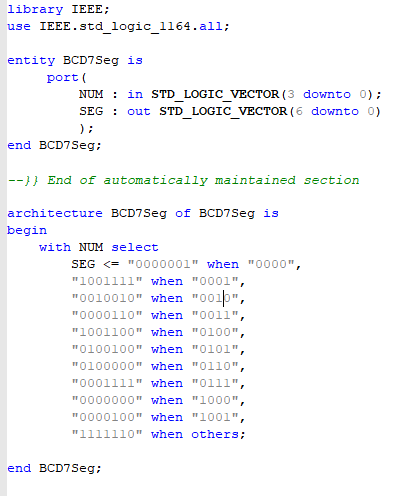
**Diseño del conversor en LogicWorks**

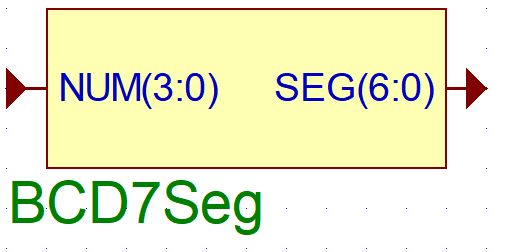


**Anexo 3 BCD7Seg**

**Anexo 3.1**

**Código y símbolo**

****

****

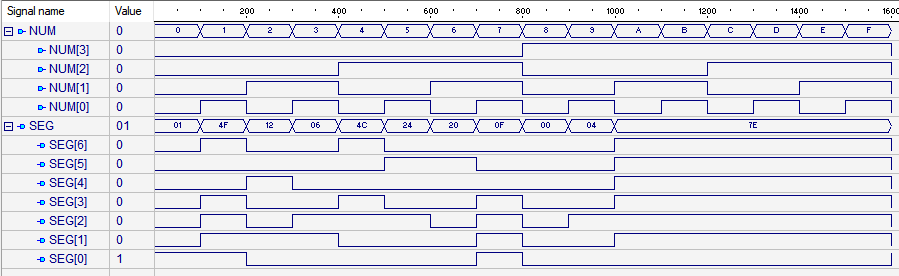
**Anexo 3.2**

**Tabla de resultados**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SW1** | **SW2** | **SW3** | **SW4** | **a** | **b** | **c** | **d** | **e** | **f** | **g** |
| **OFF** | **OFF** | **OFF** | **OFF** | **ON** | **ON** | **ON** | **ON** | **ON** | **ON** | **OFF** |
| **OFF** | **OFF** | **OFF** | **ON** | **OFF** | **ON** | **ON** | **OFF** | **OFF** | **OFF** | **OFF** |
| **OFF** | **OFF** | **ON** | **OFF** | **ON** | **ON** | **OFF** | **ON** | **ON** | **OFF** | **ON** |
| **OFF** | **OFF** | **ON** | **ON** | **ON** | **OFF** | **ON** | **ON** | **OFF** | **ON** | **OFF** |
| **OFF** | **ON** | **OFF** | **OFF** | **OFF** | **ON** | **ON** | **OFF** | **OFF** | **ON** | **ON** |
| **OFF** | **ON** | **OFF** | **ON** | **ON** | **OFF** | **ON** | **ON** | **OFF** | **ON** | **ON** |
| **OFF** | **ON** | **ON** | **OFF** | **ON** | **OFF** | **ON** | **ON** | **ON** | **ON** | **ON** |
| **OFF** | **ON** | **ON** | **ON** | **ON** | **ON** | **ON** | **OFF** | **OFF** | **OFF** | **OFF** |
| **ON** | **OFF** | **OFF** | **OFF** | **ON** | **ON** | **ON** | **ON** | **ON** | **ON** | **ON** |
| **ON** | **OFF** | **OFF** | **ON** | **ON** | **ON** | **ON** | **OFF** | **OFF** | **ON** | **ON** |
| **ON** | **OFF** | **ON** | **OFF** | **ON** | **OFF** | **OFF** | **OFF** | **OFF** | **OFF** | **OFF** |
| **ON** | **OFF** | **ON** | **ON** | **ON** | **OFF** | **OFF** | **OFF** | **OFF** | **OFF** | **OFF** |
| **ON** | **ON** | **OFF** | **OFF** | **ON** | **OFF** | **OFF** | **OFF** | **OFF** | **OFF** | **OFF** |
| **ON** | **ON** | **OFF** | **ON** | **ON** | **OFF** | **OFF** | **OFF** | **OFF** | **OFF** | **OFF** |
| **ON** | **ON** | **ON** | **OFF** | **ON** | **OFF** | **OFF** | **OFF** | **OFF** | **OFF** | **OFF** |
| **ON** | **ON** | **ON** | **ON** | **ON** | **OFF** | **OFF** | **OFF** | **OFF** | **OFF** | **OFF** |

**Anexo 3.3**

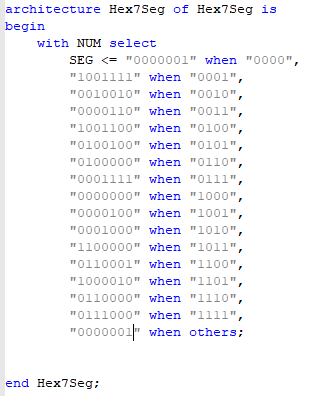
**Simulación**

****

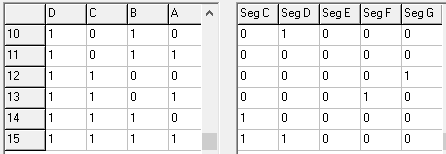
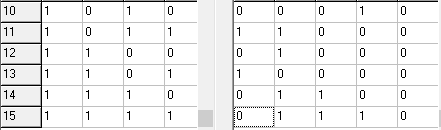
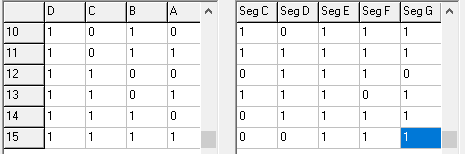
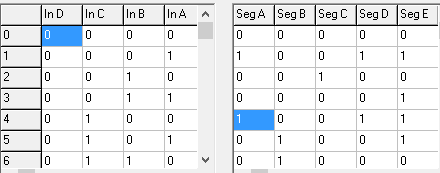
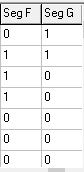
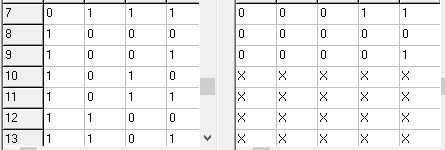
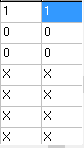
**Anexo 4 Hex7Seg**

**Anexo 4.1**

**Símbolo y código**

****

**Anexo 4.2**

**Tabla de la verdad**

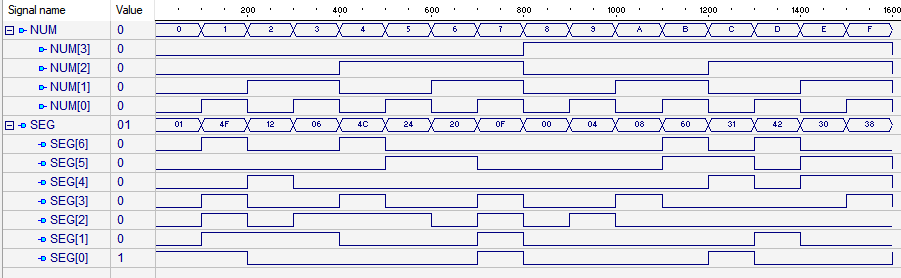
**Anexo 4.3**

**Tabla de resultados**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SW1** | **SW2** | **SW3** | **SW4** | **a** | **b** | **c** | **d** | **e** | **f** | **g** |
| **OFF** | **OFF** | **OFF** | **OFF** | **ON** | **ON** | **ON** | **ON** | **ON** | **ON** | **OFF** |
| **OFF** | **OFF** | **OFF** | **ON** | **OFF** | **ON** | **ON** | **OFF** | **OFF** | **OFF** | **OFF** |
| **OFF** | **OFF** | **ON** | **OFF** | **ON** | **ON** | **OFF** | **ON** | **ON** | **OFF** | **ON** |
| **OFF** | **OFF** | **ON** | **ON** | **ON** | **OFF** | **ON** | **ON** | **OFF** | **ON** | **OFF** |
| **OFF** | **ON** | **OFF** | **OFF** | **OFF** | **ON** | **ON** | **OFF** | **OFF** | **ON** | **ON** |
| **OFF** | **ON** | **OFF** | **ON** | **ON** | **OFF** | **ON** | **ON** | **OFF** | **ON** | **ON** |
| **OFF** | **ON** | **ON** | **OFF** | **ON** | **OFF** | **ON** | **ON** | **ON** | **ON** | **ON** |
| **OFF** | **ON** | **ON** | **ON** | **ON** | **ON** | **ON** | **OFF** | **OFF** | **OFF** | **OFF** |
| **ON** | **OFF** | **OFF** | **OFF** | **ON** | **ON** | **ON** | **ON** | **ON** | **ON** | **ON** |
| **ON** | **OFF** | **OFF** | **ON** | **ON** | **ON** | **ON** | **OFF** | **OFF** | **ON** | **ON** |
| **ON** | **OFF** | **ON** | **OFF** | **ON** | **ON** | **ON** | **OFF** | **ON** | **ON** | **ON** |
| **ON** | **OFF** | **ON** | **ON** | **OFF** | **OFF** | **ON** | **ON** | **ON** | **ON** | **ON** |
| **ON** | **ON** | **OFF** | **OFF** | **ON** | **OFF** | **OFF** | **ON** | **ON** | **ON** | **OFF** |
| **ON** | **ON** | **OFF** | **ON** | **OFF** | **ON** | **ON** | **ON** | **ON** | **OFF** | **ON** |
| **ON** | **ON** | **ON** | **OFF** | **ON** | **OFF** | **OFF** | **ON** | **ON** | **ON** | **ON** |
| **ON** | **ON** | **ON** | **ON** | **ON** | **OFF** | **OFF** | **OFF** | **ON** | **ON** | **ON** |

**Anexo 4.4**

**Simulación**

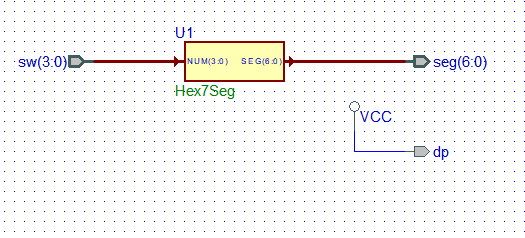


**Anexo 5**

**Hex7Seg\_Top**

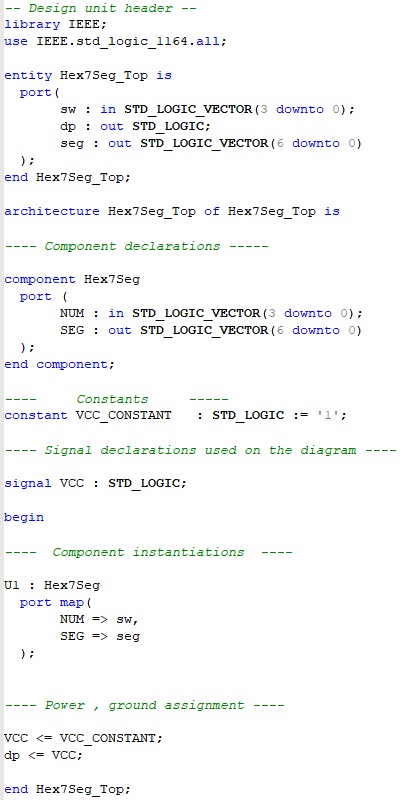
**Anexo 5.1**

**Diagrama de bloques**

****

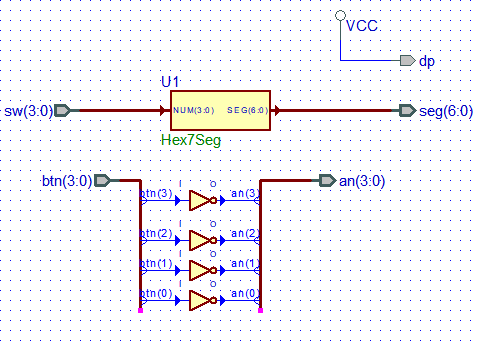
**Anexo 5.2**

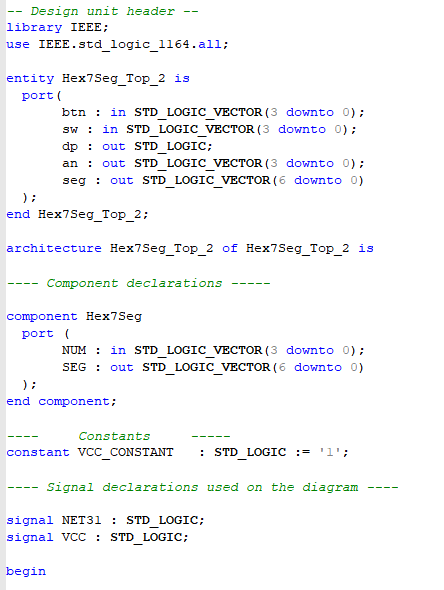
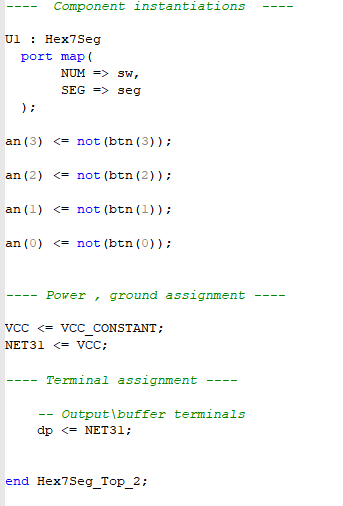
**Código VHDL**

****

**Anexo 6 Hex7Seg\_Top2**

**Anexo 6.1 Diagrama de bloques**

****

**Anexo 6.2 Código VHDL**