UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Práctica 4

Lenguaje de Descripción de Hardware VHDL

ALUMNOS

Ríos Lira, Gamaliel
Vélez Grande, Cinthya

PROFESOR

Flores Olvera, Vicente

ASIGNATURA

Laboratorio de Diseño Digital Moderno

GRUPO

1

26 de marzo de 2023

Índice

1. Objetivo

El alumno analizará de qué partes esenciales contra un código hecho a través del lenguaje de descripción de hardware *VHDL*, así que implica la implementación de funciones mediante el estilo flujo de datos.

2. Materiales

Equipo del laboratorio

- Computadora

Software

- *Quartus Prime Lite* 18 (o superior)

3. Introducción

El lenguaje *VHDL* (*Very High Speed Integrates Circuit Hardware Description Language*) es un lenguaje de descripción de circuitos electrónicos digitales que utiliza distintos niveles de abstracción. *VHDL* nació como una manera estándar de documentar circuitos; desarrollado por el gobierno de Estados Unidos en la década de los 80's.

De manera particular, es un lenguaje que permite tanto una descripción de la estructura del circuito (descripción a partir de subcircuitos más sencillos), como la especificación de la funcionalidad del circuito utilizando formas familiares a los lenguajes de programación.

Los circuitos descritos en *VHDL* pueden ser simulados utilizando herramientas que reproducen el funcionamiento del circuito descrito; la misión más importante de un lenguaje de descripción HW es que sea capaz de simular perfectamente el comportamiento lógico de un circuito sin que el programador necesite imponer restricciones.

Un circuito o subcircuito descrito mediante *VHDL* se denomina diseño de entidad (design entity). Está compuesto por dos partes: la declaración de la entidad, entity, (donde se declaran las señales de entrada y salida, por lo tanto es el modelo de interfaz con el exterior) y la arquitectura, architecture (donde se definen los detalles del circuito, es decir, es la especificación del funcionamiento de una entidad).

Por otro lado están las bibliotecas, donde se almacenan los elementos de diseño: tipos de datos, operadores, componentes, funciones, etc... Hay dos bibliotecas que siempre son visibles por defecto: std (la estándar) y work (la de trabajo) y que no es necesario declarar.

Los elementos de las bibliotecas se organizan en paquetes o bibliotecas (Packages) y hay que declararlos para poder utilizarlos dentro de la construcción de los circuitos.

A lo largo de la presente práctica se han implementado, a través del lenguaje *VHDL*, diversos circuitos que se diseñaron previamente mediante herramientas gráficas proporcionadas por el entorno de desarrollo; como se verá, es posible obtener equivalencias en los resultados mediante ambas implementaciones.

4. Desarrollo

4.1. Previo

Pregunta 1

Obtener las formas mínimas SOP y POS al igual que las formas canónicas de las funciones combinacional representado por un sistema que tiene cuatro entradas: a, b, c y d; y cuatro salidas: w, x, y y z. La salida representa un número en código exceso-3 cuyo valor es igual al número de unos presentes en la entrada. Por ejemplo, si $abcd = 1101$, entonces la salida debe ser $wxyz = 0110$.

Obteniendo la tabla de verdad del comportamiento del sistema, se tiene que:

a	**b**	**c**	**d**	**w**	**x**	**y**	**z**
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	0
0	0	1	1	0	1	0	1
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	1	0
1	0	0	0	0	1	0	0
1	0	0	1	0	1	1	0
1	0	1	0	0	1	0	1
1	0	1	1	0	1	1	0
1	1	0	0	0	1	0	1
1	1	0	1	0	1	1	0
1	1	1	0	0	1	1	0
1	1	1	1	0	1	1	1

Tabla 1: Comportamieto de $f(abcd)$ (Previo)

De la tabla de verdad anterior se pueden obtener las formas *SOP* o *POS* canónicas.

Obteniendo w. Por simple inspección sobre la tabla de verdad, se tiene que:

$$\boxed{\therefore w_{sop}(abcd) = 1}$$

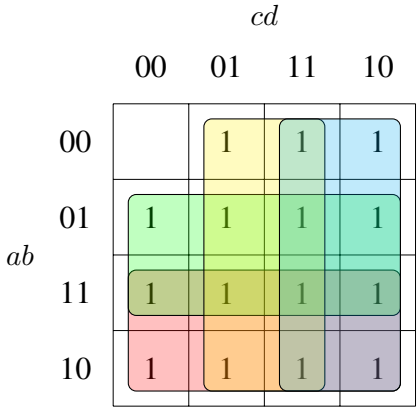
Y como conclusión también se tiene que:

$$\boxed{\therefore w_{pos}(abcd) = 1}$$

Obteniendo x. A través de la inspección a la tabla de verdad se pueden obtener los *mintérminos*.

$$x_{sop}(abcd) = \sum_{m}(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)$$

La simplificación se llevará a cabo con ayuda del método de *Mapas de Karnaugh*.



La lectura del mapa nos indica que:

$$\boxed{\therefore x_{sop}(abcd) = a + b + c + d}$$

Por otra parte, inspeccionando la tabla de verdad y obteniendo los *maxtérminos*, se tiene que:

$$x_{pos}(abcd) = \prod_{M}(0)$$

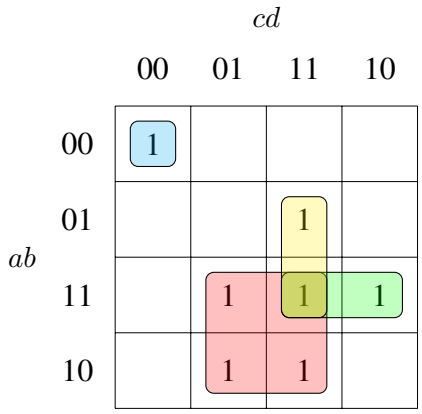
Con lo cual, se tiene que:

$$\boxed{\therefore x_{pos}(abcd) = (a + b + c + d)}$$

Obteniendo y. A través de la inspección a la tabla de verdad se pueden obtener los *mintérminos* y *maxtérminos*.

$$y_{sop}(abcd) = \sum_{m}(0, 7, 9, 11, 13, 14, 15)$$

La simplificación se llevará a cabo con ayuda del método de *Mapas de Karnaugh*.



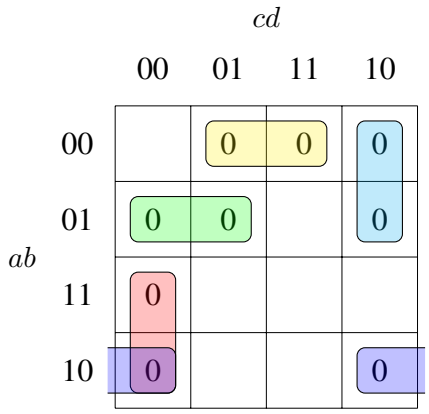
La lectura del mapa resulta en:

$$\boxed{\therefore y_{sop}(abcd) = ad + bcd + abc + \bar{a}\bar{b}\bar{c}\bar{d}}$$

Por otra parte, haciendo uso de los *mastérminos*, se tiene que:

$$y_{pos}(abcd) = \prod_{M}(1, 2, 3, 4, 5, 6, 8, 10, 12)$$

La simplificación se llevará a cabo con ayuda del método de *Mapas de Karnaugh*.



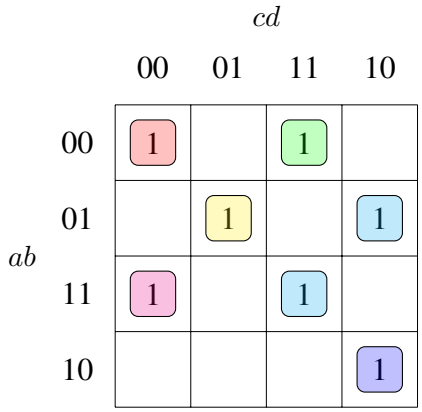
La lectura del mapa resulta en:

$$\boxed{\therefore y_{pos}(abcd) = (\bar{a} + b + d)(\bar{a} + c + d)(a + \bar{b} + c)(a + b + \bar{d})(a + \bar{c} + d)}$$

Obteniendo z. A través de la inspección a la tabla de verdad se pueden obtener los *mintérminos*.

$$z_{sop}(abcd) = \sum_{m}(0, 3, 5, 6, 10, 12, 15)$$

La simplificación se llevará a cabo con ayuda del método de *Mapas de Karnaugh*.



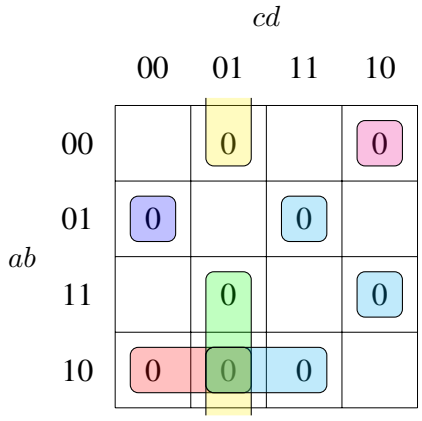
La lectura del mapa resulta en:

$$z_{sop}(abcd) = \bar{a}\bar{b}\bar{c}\bar{d} + \bar{a}\bar{b}cd + \bar{a}b\bar{c}d + \bar{a}bc\bar{d} + a\bar{b}c\bar{d} + ab\bar{c}\bar{d} + abcd$$

Por otra parte, haciendo uso de los *mastérminos*, se tiene que:

$$z_{pos}(abcd) = \prod_{M}(1, 2, 4, 7, 8, 9, 11, 13, 14)$$

La simplificación se llevará a cabo con ayuda del método de *Mapas de Karnaugh*.



La lectura del mapa resulta en:

$$\therefore z_{pos}(abcd) = (\bar{a} + c + \bar{d})(\bar{a} + b + c)(\bar{a} + b + \bar{d})(b + c + \bar{d})(a + \bar{b} + c + d)(a + \bar{b} + \bar{c} + \bar{d})(a + b + \bar{c} + d)(\bar{a} + \bar{b} + \bar{c} + d)$$

Pregunta 2

¿Qué son las bibliotecas, la entidad y la arquitectura en una estructura descrita en VHDL?

Biblioteca. Se trata de un paquete de código pre-programado por alquién más y al cual tenemos acceso para hacer uso de sus tipos de datos, funciones, etc.

Entidad. Una entidad es la abstracción de un circuito, ya sea desde un complejo sistema electrónico o una simple puerta lógica. La entidad únicamente describe la forma externa del circuito, en ella se enumeran las entradas y las salidas del diseño. Una entidad es análoga a un símbolo esquemático en los diagramas electrónicos, el cual describe las conexiones del dispositivo hacia el resto del diseño.

- Define externamente al circuito o subcircuito.
- Nombre y número de puertos, tipos de datos de entrada y salida.
- Tienes toda la información necesaria para conectar tu circuito a otros circuitos.

Además, en la entidad se pueden definir unos valores genéricos (generic) que se utilizarán para declarar propiedades y constantes del circuito, independientemente de cual sea la arquitectura.

Arquitectura. Una arquitectura describe el funcionamiento de la entidad a la que hace referencia, es decir, dentro de architecture tendremos que describir el funcionamiento de la entidad a la que está asociada utilizando las sentencias y expresiones propias de *VHDL*.

- Define internamente el circuito.
- Señales internas, funciones, procedimientos, constantes.
- La descripción de la arquitectura puede ser estructural o por comportamiento.

El código *VHDL* propiamente se escribe dentro del segmento de código *architecture*.

Pregunta 3

Investiga cómo se realiza la asignación de una función o de un valor en VHDL.

Para hacer una asignación a una función (o señal) deberemos usar el operador <=, colocando el identificador de la señal a asignar a la izquierda y el valor que esta debe tomar a la derecha; tal como se muestra en la siguiente expresión:

```
1  signal <= signal1 + signal2;
```

Pregunta 4

¿Qué significa el termino concurrente dentro del lenguaje VHDL?

Las sentencias concurrentes son aquellas que se ejecutan simultáneamente en la simulación, es decir, no existe una prioridad entre unas u otras. Se utilizan para el modelado del hardware porque describen adecuadamente su comportamiento. Deben de formar parte siempre del cuerpo de arquitecturas o de bloques. Las sentencias concurrentes, como su nombre indica, se deben evaluar simultáneamente. Y por lo tanto el orden en que se escriben dentro del módulo que las aloja es totalmente irrelevante. Dicho de otra forma, cualquiera que sea el orden de un mismo conjunto de sentencias concurrentes, estas deben acabar produciendo el mismo resultado.

Algunos ejemplos de sentencias concurrentes se listan a continuación: procesos, asignaciones concurrentes a señal, llamadas concurrentes a procedimientos, bloques, instanciaciones de componentes, sentencias *assert-report* y sentencias *generate*.

4.2. Parte A

4.2.1. Instrucciones

Programar las funciones mostradas, en un formato SOP y POS mínimo (reducido) en forma flujo de datos, para su posterior simulación dentro de la plataforma Quartus II.

- ***Mediante POS***

$$f(xyzt) = (x + \bar{x}y + \overline{yt})(x + \overline{xy} \cdot z)$$

- ***Mediante SOP***

$$f(uvtw) = vt(u + t\bar{w})(u + \bar{w}) + wu(v + t)$$

4.2.2. Análisis

Primero, se desarrollará la función $f(xyzt)$ a través de álgebra booleana para encontrar una forma POS reducida:

$$\begin{aligned} f(xyzt) &= (x + \bar{x}y + \overline{yt})(x + \overline{xy} \cdot z) \\ &= (\cancel{(x + \bar{x})}(x + y) + \overline{yt})(x + (\bar{x} + \bar{y})z) \\ &= \cancel{(x + y + \bar{y} + t)}(x + (\bar{x} + \bar{y})z) \\ &= \cancel{(x + \bar{x} + \bar{y})}(x + z) \\ &= (x + z) \end{aligned}$$

$$\boxed{\therefore f(xyzt) = (x + z)}$$

Posteriormente, se desarrollará la función $f(wvtw)$ a través de álgebra booleana para encontrar una forma SOP reducida:

$$\begin{aligned} f(uvtw) &= vt(u + t\bar{w})(u + \bar{w}) + wu(v + t) \\ &= vt(u + t)(u + \bar{w})\cancel{(u + \bar{w})} + wu(v + t) \\ &= vt(u + t)(u + \bar{w}) + wu(v + t) \\ &= vt(u + t\bar{w}) + wuv + wut \\ &= uvt + vt\bar{w} + uvw + utw \end{aligned}$$

$$\boxed{\therefore f(uvtw) = uvt + vt\bar{w} + uvw + utw}$$

4.2.3. Implementación en Quartus

Con las formas reducidas anteriormente, se implementó el sistema dentro de la plataforma *Quartus II*. Para esto, después de crear el proyecto, se creó un nuevo archivo de tipo **VHDL**, el cual tiene el contenido mostrado a continuación:

```vhdl
-- Implementacion: Parte A

library ieee;
use ieee.std_logic_1164.all;

entity p04a is
  port (
    a, b, c, d    : in    std_logic;
    f1, f2        : out   std_logic
  );
end;

architecture simple of rlg_p04a is
begin
  f1 <= a or c;
  f2 <= (a and b and c) or (b and c and not d)
        or (a and b and d) or (a and c);
end;
```

Listing 1: Archivo VHDL (Parte A)

En este caso, para poder ejecutar ambas funciones se renombraron sus entradas de la siguiente forma:

- Para $f(xyzt)$: $f_1(abcd)$
  - x: a
  - y: b
  - z: c
  - t: d
- Para $f(uvtw)$: $f_2(abcd)$
  - u: a
  - v: b
  - t: c
  - w: d

Que es justo lo que se puede apreciar en el código. Para continuar, se simuló el circuito dentro de la misma plataforma, a través de una simulación funcional. Se obtuvieron las siguientes salidas:

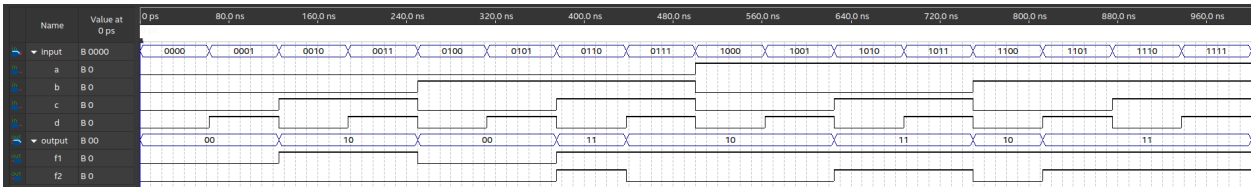


Figura 1: Simulación Funcional (Parte A)

El resultado obtenido coincide con el comportamiento de ambas funciones $f(xyzt)$ y $f(uvtw)$.

4.3. Parte B

4.3.1. Instrucciones

Programar las funciones expresadas en formato SOP y POS canónicas y mínimas del previo en forma flujo de datos, para su posterior simulación dentro de la plataforma Quartus II.

4.3.2. Análisis

El análisis correspondiente a esta sección se realizó en la sección 4.1 —en la primer pregunta del previo. En ella, se describen las funciones deducidas, las cuales se pueden pasar a código. Los mintérminos y maxtérminos se encuentras en número, pero el paso a forma algebraica es inmediato.

4.3.3. Implementación en Quartus

Con las formas reducidas anteriormente, se implementó el sistema dentro de la plataforma *Quartus II*. Para esto, después de crear un nuevo proyecto, se creó un nuevo archivo de tipo **VHDL**, el cual tiene el contenido mostrado a continuación:

```
-- Implementacion: Parte B
library ieee;
use ieee.std_logic_1164.all;

entity p04b is
  port(
    a,b,c,d   : in    std_logic;
    f_sop,
    f_pos,
    fr_sop,
    fr_pos    : out   std_logic_vector(3 downto 0)
  );
end;

architecture simple of p04b is
begin
  -- Forma SOP Simplificada
  fr_sop <= (
    3 => '0',
    2 => a or b or c or d,
    1 => (a and d) or (b and c and d) or (a and b and c)
        or (not a and not b and not c and not d),
    0 => (not a and not b and not c and not d)
        or (not a and not b  and c and d)
        or (not a and b and not c and d)
        or (not a and b and c and not d)
        or (a and not b and c and not d)
        or (a and b and not c and not d)
        or (a and b and c and d)
```

```
);

-- Forma POS Simplificada
fr_pos <= (
  3 => '0',
  2 => a or b or c or d,
  1 => (not a or b or d) and (not a or c or d)
      and (a or not b or c) and (a or b or not d)
      and (a or not c or d),
  0   => (not a or c or not d)
      and (not a or b or c)
      and (not a or b or not d)
      and (b or c or not d)
      and (a or not b or c or d)
      and (a or not b or not c or not d)
      and (a or b or not c or d)
      and (not a or not b or not c or d)
);

-- Forma SOP Canonica
f_sop <= (
  3 => '0',
  2 => (not a and not b and not c and d)
      or (not a and not b and c and not d)
      or (not a and not b and c and d)
      or (not a and b and not c and not d)
      or (not a and b and not c and d)
      or (not a and b and c and not d)
      or (not a and b and c and d)
      or (a and not b and not c and not d)
      or (a and not b and not c and d)
      or (a and not b and c and not d)
      or (a and not b and c and d)
      or (a and b and not c and not d)
      or (a and b and not c and d)
      or (a and b and c and not d)
      or (a and b and c and d),
  1 => (not a and not b and not c and not d)
      or (not a and b and c and d)
      or (a and not b and not c and d)
      or (a and not b and c and d)
      or (a and b and not c and d)
      or (a and b and c and not d)
      or (a and b and c and d),
  0 => (not a and not b and not c and not d)
      or (not a and not b and c and d)
      or (not a and b and not c and d)
      or (not a and b and c and not d)
      or (a and not b and c and not d)
      or (a and b and not c and not d)
      or (a and b and c and d)
);

-- Forma POS Canonica
```

```
  f_pos <= (
    3 => '0',
    2 => a or b or c or d,
    1 => (not a or not b or c or d)
        and (not a or b or not c or d)
        and (not a or b or c or d)
        and (a or not b or not c or d)
        and (a or not b or c or not d)
        and (a or not b or c or d)
        and (a or b or not c or not d)
        and (a or b or not c or d)
        and (a or b or c or not d),
    0 => (not a or not b or not c or d)
        and (not a or not b or c or not d)
        and (not a or b or not c or not d)
        and (not a or b or c or not d)
        and (not a or b or c or d)
        and (a or not b or not c or not d)
        and (a or not b or c or d)
        and (a or b or not c or d)
        and (a or b or c or not d)
  );
end;
```

Listing 2: Archivo VHDL (Parte B)

Tal como se puede ver, las salidas del sistema están dada en forma de vector, para lo cual se usó el tipo de dato *std_logic_vector* y donde w era el bit más significativo, mientras que z era el bit menos significativo. Tal como se puede ver en el código, se implementaron cuatro funciones diferentes:

- *fr_sop*: Forma SOP reducida
- *fr_pos*: Forma POS reducida
- *f_sop*: Forma SOP canónica
- *f_pos*: Forma POS canónica

A través de uns simulación funcional en *Quartus II*, se obtuvo el siguiente resultado:

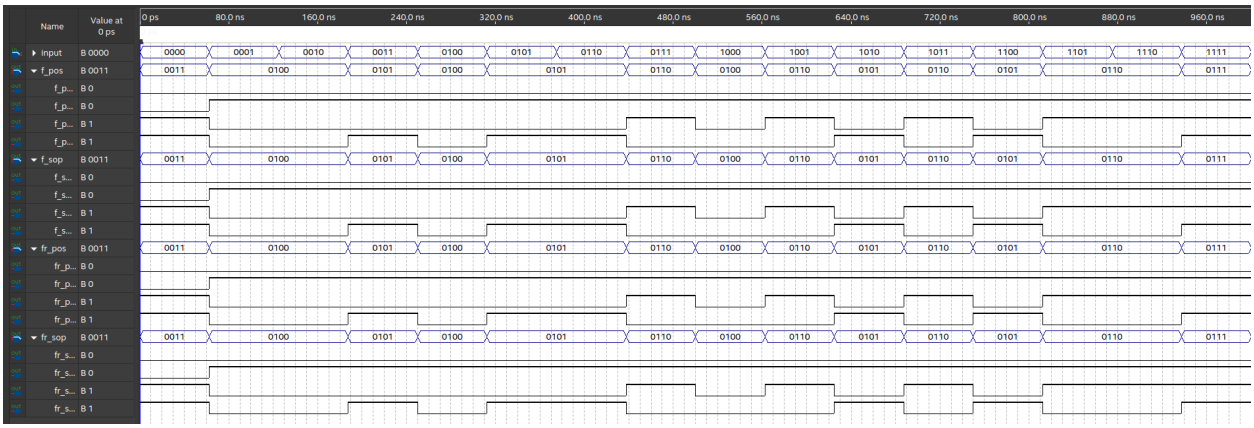


Figura 2: Simulación Funcional (Parte B)

El comportamiento del cronograma es el esperado: Tanto la forma SOP canónica; la forma POS canónica; la forma SOP reducida; y la forma POS reducida se comportan todas iguales.

4.4. Parte C

4.4.1. Propuesta

Nota. Se utilizará el mismo problema propuesto en la clase pasada con la finalidad de implementarlo en *VHDL*.

Luis, Ana y sus padres van al cine bajo ciertas condiciones que son limitadas por la Madre y el Padre. Ellos van al cine si la Madre y el Padre están totalmente de acuerdo, si los dos no lo están, entonces no pueden ir. Ahora, si los dos están en desacuerdo, la salida al cine debe ser resuelta por mayoría absoluta entre todos los integrantes de la familia. Si existe un empate general, la decisión será tomada por la madre. Diseñar un circuito digital que señalice con un diodo LED el momento cuando pueden ir al cine.

4.4.2. Análisis

De forma general, el sistema tiene cuatro entradas: l, a, m Y p (el voto de Luis, Ana, su Madre y su Padre respectivamente); y la salida estará dada por el valor de $c(lamp)$. La Implementación de la tabla de verdad del problema se muestra a continuación:

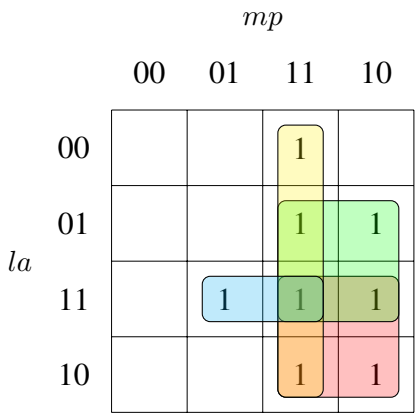
l	a	m	p	c
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Tabla 2: Tabla de verdad del problema (Sección C)

Con lo anterior, se puede deducir la forma canónica *SOP*, tal como se muestra a continuación:

$$c(lamp) = \sum_{m}(3, 6, 7, 10, 11, 13, 14, 15)$$

La simplificación se llevará a cabo con ayuda del método de *Mapas de Karnaugh*.



La lectura del mapa nos indica que:

$$c(lamp) = mp + am + lm + lap$$

Simplificando un poco la expresión, se tiene que:

$$\boxed{\therefore c(lamp) = m(p + a + l) + lap}$$

De forma general, lo anterior nos indica que si la Mamá quiere ir al cine y al menos alguien más también quiere ir, entonces irán al cine; la única otra forma de ir al cine es cuando tanto Luis, Ana y el Papá quieren ir.

4.4.3. Implementación en Quartus

Con la reducción anterior, se implementó el sistema dentro de la plataforma *Quartus II*. Para esto, después de crear el proyecto, se creó un nuevo archivo de tipo **VHDL**, el cual tiene el contenido mostrado a continuación:

```
-- Implementacion: Parte C

library ieee;
use ieee.std_logic_1164.all;

entity p04c is
  port(
    l,a,m,p   :in std_logic;
    c       :out  std_logic
  );
end;

architecture simple of p04c is
begin
  c <= (m and (l or a or p)) or (l and a and p);
```

```
16 end;
```

Listing 3: Archivo VHDL (Parte C)

Con lo anteiror, se ejecutó una simulación funcional dentro de *Quartus II*, con lo cual se obtuvieron las siguientes salidas:

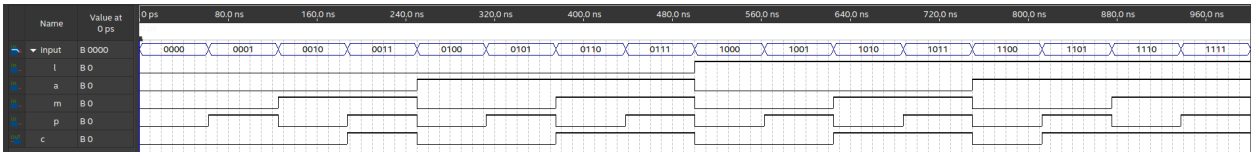


Figura 3: Simulación Funcional (Parte C)

Esto coincide con los valores establecidos en la Tabla 2. Con lo cual, el sistema se implementó de forma adecuada.

5. Conclusiones

Ríos Lira, Gamaliel. Con la realización de la práctica, el objetivo de la misma se cumplió de forma satisfactoria. Por un lado, se tuvo un acercamiento a las ventajas que trae consigo la utilización de un lenguaje de descripción de *hardware* —tal como *VHDL*— al comparar la velocidad con la que se implementan circuitos con respecto al modo gráfico. Y por otro lado, se tuvo una introducción a la forma en la que funciona, describiendo sus partes más esenciales: Bibliotecas, Entidades y Arquitecturas. De igual forma, todo esto se complementó con la implementación en código de algunos ejemplos. La diferencia entre el modelo teórico al que se llega a través de álgebra booleana o *Mapas de Kanaugh* es mínimo ya que el código que se utiliza es muy similar a la notación algábraica.

Vélez Grande, Cinthya. Mediante el desarrollo de la práctica se realizó la implementación de algunos circuitos a través del lenguaje VHDL; a partir de ello fue posible comprender de qué partes se conforma un código desarrollado en este lenguaje. Aprendimos la manera en la que se realiza tanto la declaración de las entidades a emplear, como la asignación de las funciones correspondientes a las formas POS y SOP, dentro del bloque de arquitectura, de las funciones lógicas de los problemas a tratar. Fue posible visualizar la forma en la que se vincula el desarrollo teórico del diseño de los circuitos con su representación en el lenguaje VHDL; obteniendo resultados equivalentes tanto en las implementaciones gráficas realizadas de manera previa en prácticas anteriores, como en su representación con el lenguaje utilizado.

6. Bibliografía

[1] M. Mano, *Digital Design*. Upper Saddle River, NJ, United States: Prentice Hall, 2002.

[2] A. Mora Campos, *GUÍA BÁSICA DEL VHDL*. Instituto Tecnológico de Querétaro, 2016. dirección: `http://www.itq.edu.mx/carreras/IngElectronica/archivos_contenido/Apuntes%20de%20materias/Apuntes_VHDL_2016.pdf`.

[3] F. Nuño García, *Introducción al lenguaje VHDL*. Área de Tecnología Electrónica Universidad de Oviedo. dirección: `https://www.geocities.ws/curso_tecnologia_electronica/VHDL/VHDL-OVI.pdf`.