Lenguajes de descripción de hardware, Diseño Digital 01, 02/03/2020

Marquesina Combinacional

Hernández Castellanos César Uriel

Centro de Investigación en Computación
Instituto Politécnico Nacional, México
uuriel12009u@gmail.com

Resumen: En el presente informe se diseña e implementa un circuito con la capacidad de emular una marquesina combinacional por medio de un conjunto de entradas que representarán los códigos de las letras y una entrada que indica la posición de nuestra marquesina.

Palabras Clave: Circuitos combinacionales, Diseño digital, VHDL, Verilog.

1. Introducción.

1.1. Electrónica digital

El gran desarrollo experimentado por la electrónica en los últimos años ha propiciado que la mayoria de los equipos actuales funcionen con sistemas digitales. Un sistema digital se caracteriza por utilizar señales discretas, es decir, señales que toman un número finito de valores en cierto intervalo de tiempo.

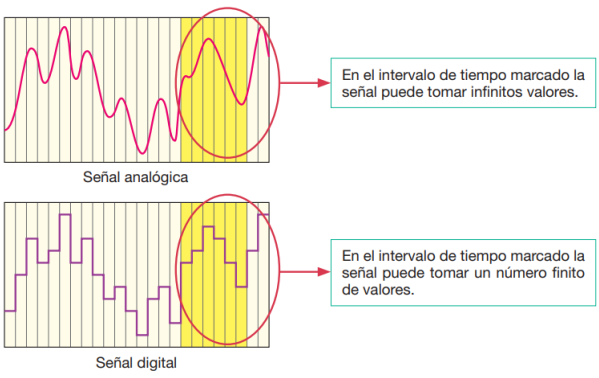

Figura 1: Comparativa gráfica de dos señales

En la figura 1 es posible apreciar que la señal inferior corresponde a la digitalización de la señal analógica, y contiene información suficiente para poder reconstruir la señal digital.

Todas las telecomunicaciones modernas (Internet, telefonía, móvil,etc) están basadas en el uso de este tipo de sistemas.

Son múltiple las razones que han favorecido el uso extensivo de los sistemas digitales, entre ellas: Mayor fiabilidad, disposición de soporte matemático, dominio de las tecnologías de fabricación y una amplia distribución comercial.

1.2. Circuito combinacional

De manera genérica podemos decir que los circuitos combinacionales son aquéllos en los que una serie de variables Xi , definen una serie de funciones Zj . Las principales restricciones que caracterizan el comportamiento funcional son las siguientes:

1. Cada combinación de valores de Xi define un sólo valor de Zj

2. Los valores de Zj sólo pueden cambiar cuando cambian los valores de Xi

La Figura 1 muestra un bloque combinacional genérico.

Figura 2: Circuito combinacional

2. Descripción del problema.

Implementar un diseño digital con la capacidad de desplegar mensajes de un máximo de cinco carácteres en un espacio de ocho displays.

Realizando el control de la posición por medio de una entrada de datos.

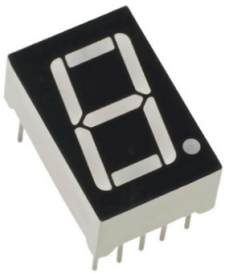

Figura 3: Display de siete segmentos

3. Descripción de la solución.

Con la finalidad de dar solución a la problemática anterior se propuso definir las siguientes entradas y salidas de nuestro diseño digital.

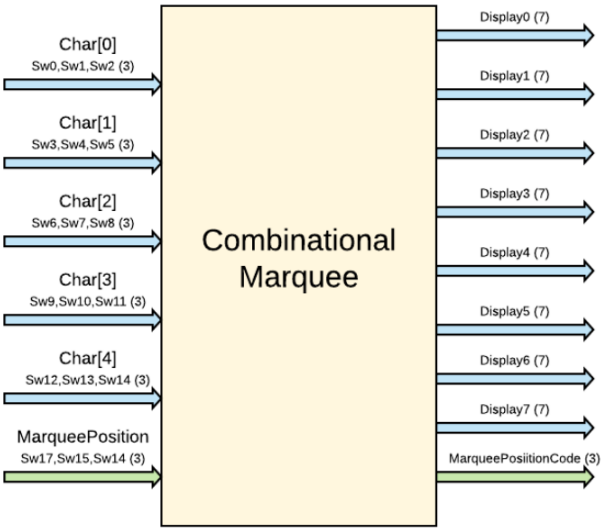

Figura 4: Diagrama a bloques

Nuestro diseño digital cuenta con un conjunto de buses de entrada en los cuales será posible ingresar los códigos de las carácteres a desplegar en los displays los cuales son nuestros buses de salida.

Además se contempla un bus de entrada que indicará la posición de nuestra marquesina, así como un bus de salida de 8 bits el cual activará o desactivará los display pertinentes.

A un nivel más interno se decidió colocar un decodificar por cada bus de entrada de código como se muestra a continuación.

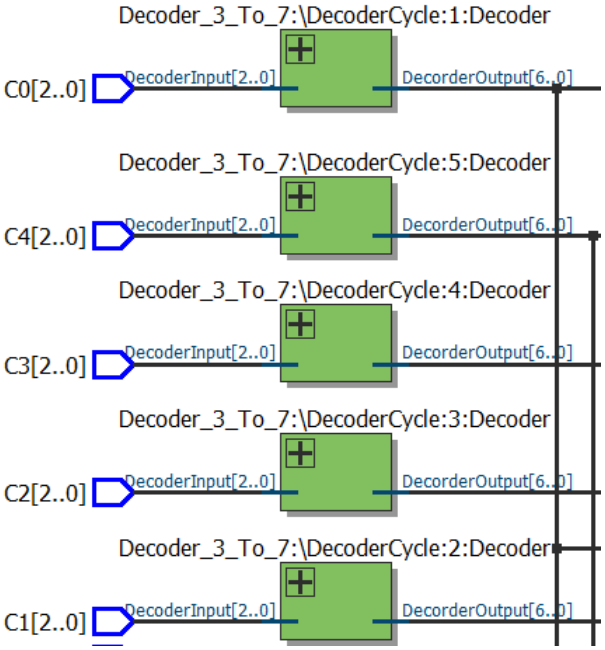

Figura 5: Decodificadores para los códigos

Una vez que se obtuvo la decodificación de los códigos se procedió a implementar diferentes multiplexores que nos auxilian a crear el efecto de marquesina.

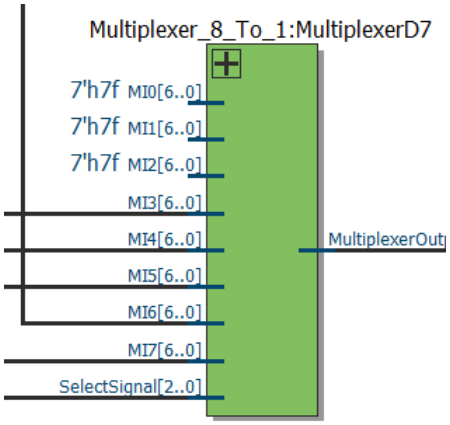

Figura 6: Decodificadores para los códigos

La figura anterior muestra uno de los multiplexores empleados.

Dichos multiplexores reciben como entradas un

conjunto de buses provenientes de la etapa de decodificación.

En el primer decodificador las tres primeras entradas son de un bus que representa carácter nulo.

Los siguientes multiplexores tendrán las mismas entradas pero con un corrimiento de tamaño uno en sus entradas.

4. Implementación.

Para mayores detalles de la implementación se puede consultar en el anexo del mismo documento.

5. Simulación

Con el propósito de tener una mayor legibilidad se procedió a vaciar los resultados de la simulación en la tabla siguiente.

Message	Marquee Position	Output
FPGAA	000	56,24,32,8,8,127,127,127
FPGAA	001	127,56,24,32,8,8,127,127
FPGAA	010	127,127,56,24,32,8,8,127
FPGAA	011	127,127,127,56,24,32,8,8
FPGAA	100	8,127,127,127,56,24,32,8
FPGAA	101	8,8,127,127,127,56,24,32
FPGAA	110	32,8,8,127,127,127,56,24
FPGAA	111	24,32,8,8,127,127,127,56

Figura 7: Tabla de resultados

La simulación puede ser vista a mayor detalle en el anexo del documento.

6. Conclusiones.

1. La lógica digital es una herramienta fundamentalpara la solución de problemas diarios, tanto así que nos encontramos rodeados de sus aplicaciones.
2. Una señal digital se caracteriza por utilizar señales discretas.
3. Analizar un circuito combinacional consiste en obtener la función de salida a partir de las entradas y las puertas a las que se encuentran conectadas.
4. Cada combinación de valores nos generará una salida en nuestro circuito combinacional.
5. El diseño digital a nivel de componentes resulta ventajoso cuando se desea implementar circuitos más complejos.

7. Referencias.

[1] Bosque Pérez, G. and Fernández Rodriguez, P. (2016). Principios de Diseño de Sistemas Digitales. 1st ed. Vitoria: Euskal, p.89.

[2] Couch II, L. (2011). Sistemas de comunicacion digitales y analogicos. Pearson Educacion de Mexico S.A. de C.V.

8. Anexo.

8.1. Simulación.

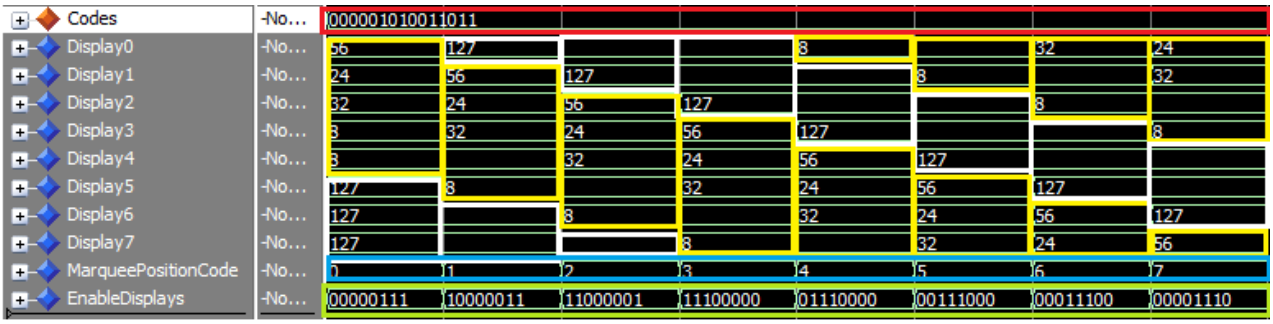

Figura 8: Simulación del circuito final

8.2. Entidad principal [VHDL]

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.NUMERIC_BIT.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY CombinationalMarquee IS

        GENERIC (
                InputCodeSize         : INTEGER := 3;
                InputPositionBusSize : INTEGER := 3;
                OutputSize            : INTEGER := 7;
                NumberOfLetters       : INTEGER := 5;
        );

        PORT (
         C0, C1, C2, C3, C4 : IN STD_LOGIC_VECTOR(InputCodeSize - 1 DOWNTO 0);
         MarqueePosition : IN STD_LOGIC_VECTOR(InputPositionBusSize - 1 DOWNTO 0);
         D0, D1, D2, D3, D4, D5, D6, D7 : OUT STD_LOGIC_VECTOR(OutputSize - 1 DOWNTO 0);
         MarqueePositionOutput : OUT STD_LOGIC_VECTOR(NumberOfDisplays - 1 DOWNTO 0)
        );

END CombinationalMarquee;

ARCHITECTURE CombinationalMarqueeArchitecture OF CombinationalMarquee IS

 SIGNAL EmptySymbol : STD_LOGIC_VECTOR(OutputSize - 1 DOWNTO 0) := (OTHERS => '1');
 SIGNAL BusOfCodes : STD_LOGIC_VECTOR(InputCodeSize * NumberOfLetters - 1 DOWNTO 0);
 SIGNAL BusOfCharacteres : STD_LOGIC_VECTOR(OutputSize * NumberOfLetters - 1 DOWNTO 0);

BEGIN
        BUSOFCODES <= C4 & C3 & C2 & C1 & C0;

        DecoderCycle : FOR I IN 1 TO NumberOfLetters GENERATE
                Decoder : Decoder_3_To_7
                PORT MAP(
                        DecoderInput => BusOfCodes(3 * I - 1 DOWNTO 3 * (I - 1)),
                        DecorderOutput => BusOfCharacteres(7 * I - 1 DOWNTO 7 * (I - 1))
                );
        END GENERATE;
        MultiplexerD0 : Multiplexer_8_To_1
        PORT MAP(
                MI0 => BusOfCharacteres(6 DOWNTO 0),
                MI1 => EmptySymbol,
                MI2 => EmptySymbol,
                MI3 => EmptySymbol,
                MI4 => BusOfCharacteres(34 DOWNTO 28),
                MI5 => BusOfCharacteres(27 DOWNTO 21),
```

```
        MI6 => BusOfCharacteres(20 DOWNTO 14),
        MI7 => BusOfCharacteres(13 DOWNTO 7),
        MultiplexerOutput => D0,
        SelectSignal => MarqueePosition
);
MultiplexerD1 : Multiplexer_8_To_1
PORT MAP(
        MI0 => BusOfCharacteres(13 DOWNTO 7),
        MI1 => BusOfCharacteres(6 DOWNTO 0),
        MI2 => EmptySymbol,
        MI3 => EmptySymbol,
        MI4 => EmptySymbol,
        MI5 => BusOfCharacteres(34 DOWNTO 28),
        MI6 => BusOfCharacteres(27 DOWNTO 21),
        MI7 => BusOfCharacteres(20 DOWNTO 14),
        MultiplexerOutput => D1,
        SelectSignal => MarqueePosition
);

MultiplexerD2 : Multiplexer_8_To_1
PORT MAP(
        MI0 => BusOfCharacteres(20 DOWNTO 14),
        MI1 => BusOfCharacteres(13 DOWNTO 7),
        MI2 => BusOfCharacteres(6 DOWNTO 0),
        MI3 => EmptySymbol,
        MI4 => EmptySymbol,
        MI5 => EmptySymbol,
        MI6 => BusOfCharacteres(34 DOWNTO 28),
        MI7 => BusOfCharacteres(27 DOWNTO 21),
        MultiplexerOutput => D2,
        SelectSignal => MarqueePosition
);

MultiplexerD3 : Multiplexer_8_To_1
PORT MAP(
        MI0 => BusOfCharacteres(27 DOWNTO 21),
        MI1 => BusOfCharacteres(20 DOWNTO 14),
        MI2 => BusOfCharacteres(13 DOWNTO 7),
        MI3 => BusOfCharacteres(6 DOWNTO 0),
        MI4 => EmptySymbol,
        MI5 => EmptySymbol,
        MI6 => EmptySymbol,
        MI7 => BusOfCharacteres(34 DOWNTO 28),
        MultiplexerOutput => D3,
        SelectSignal => MarqueePosition
);

MultiplexerD4 : Multiplexer_8_To_1
PORT MAP(
```

```
        MI0 => BusOfCharacteres(34 DOWNTO 28),
        MI1 => BusOfCharacteres(27 DOWNTO 21),
        MI2 => BusOfCharacteres(20 DOWNTO 14),
        MI3 => BusOfCharacteres(13 DOWNTO 7),
        MI4 => BusOfCharacteres(6 DOWNTO 0),
        MI5 => EmptySymbol,
        MI6 => EmptySymbol,
        MI7 => EmptySymbol,
        MultiplexerOutput => D4,
        SelectSignal => MarqueePosition
);
MultiplexerD5 : Multiplexer_8_To_1
PORT MAP(
        MI0 => EmptySymbol,
        MI1 => BusOfCharacteres(34 DOWNTO 28),
        MI2 => BusOfCharacteres(27 DOWNTO 21),
        MI3 => BusOfCharacteres(20 DOWNTO 14),
        MI4 => BusOfCharacteres(13 DOWNTO 7),
        MI5 => BusOfCharacteres(6 DOWNTO 0),
        MI6 => EmptySymbol,
        MI7 => EmptySymbol,
        MultiplexerOutput => D5,
        SelectSignal => MarqueePosition
);
MultiplexerD6 : Multiplexer_8_To_1
PORT MAP(
        MI0 => EmptySymbol,
        MI1 => EmptySymbol,
        MI2 => BusOfCharacteres(34 DOWNTO 28),
        MI3 => BusOfCharacteres(27 DOWNTO 21),
        MI4 => BusOfCharacteres(20 DOWNTO 14),
        MI5 => BusOfCharacteres(13 DOWNTO 7),
        MI6 => BusOfCharacteres(6 DOWNTO 0),
        MI7 => EmptySymbol,
        MultiplexerOutput => D6,
        SelectSignal => MarqueePosition
);
MultiplexerD7 : Multiplexer_8_To_1
PORT MAP(
        MI0 => EmptySymbol,
        MI1 => EmptySymbol,
        MI2 => EmptySymbol,
        MI3 => BusOfCharacteres(34 DOWNTO 28),
        MI4 => BusOfCharacteres(27 DOWNTO 21),
        MI5 => BusOfCharacteres(20 DOWNTO 14),
        MI6 => BusOfCharacteres(13 DOWNTO 7),
        MI7 => BusOfCharacteres(6 DOWNTO 0),
        MultiplexerOutput => D7,
        SelectSignal => MarqueePosition
```

```
        );

        Decoder6 : Decoder_3_To_8
        PORT MAP(
                DecoderInput => MarqueePosition,
                DecorderOutput => MarqueePositionOutput
        );

END CombinationalMarqueeArchitecture;
```

8.3. Decodificador [VHDL]

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.NUMERIC_BIT.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Decoder_3_To_7 IS

        GENERIC (
                InputSize : INTEGER := 3;
                OutputSize : INTEGER := 7
        );

        PORT (
                DecoderInput : IN STD_LOGIC_VECTOR(InputSize - 1 DOWNTO 0);
                DecorderOutput : OUT STD_LOGIC_VECTOR(OutputSize - 1 DOWNTO 0)
        );

END Decoder_3_To_7;

ARCHITECTURE DecoderArchitecture OF Decoder_3_To_7 IS

        CONSTANT SymbolF : STD_LOGIC_VECTOR(OutputSize - 1 DOWNTO 0) := "0111000";
        CONSTANT SymbolP : STD_LOGIC_VECTOR(OutputSize - 1 DOWNTO 0) := "0011000";
        CONSTANT SymbolG : STD_LOGIC_VECTOR(OutputSize - 1 DOWNTO 0) := "0100000";
        CONSTANT SymbolA : STD_LOGIC_VECTOR(OutputSize - 1 DOWNTO 0) := "0001000";

BEGIN
        DECODER : PROCESS (DecoderInput, DecorderOutput) BEGIN
                CASE DecoderInput IS

                        WHEN "000" => DecorderOutput <= SymbolF;
                        WHEN "001" => DecorderOutput <= SymbolP;
                        WHEN "010" => DecorderOutput <= SymbolG;
                        WHEN "011" => DecorderOutput <= SymbolA;
                        WHEN OTHERS => DecorderOutput <= (OTHERS => '1');

                END CASE;
```

```
        END PROCESS DECODER;

END DecoderArchitecture;
```

8.4. Multiplexor [VHDL]

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.NUMERIC_BIT.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY Multiplexer_8_To_1 IS

        GENERIC (
                InputSize : INTEGER := 7;
                OutputSize : INTEGER := 7;
                SelectSignalSize : INTEGER := 3
        );

        PORT (
         MI0, MI1, MI2, MI3, MI4, MI5, MI6, MI7 : IN STD_LOGIC_VECTOR(InputSize - 1 DOWNTO 0);
        MultiplexerOutput : OUT STD_LOGIC_VECTOR(OutputSize - 1 DOWNTO 0);
        SelectSignal : IN STD_LOGIC_VECTOR(SelectSignalSize - 1 DOWNTO 0)
        );

END Multiplexer_8_To_1;

ARCHITECTURE MultiplexerArchitecture OF Multiplexer_8_To_1 IS

        CONSTANT NullSymbol : STD_LOGIC_VECTOR(OutputSize - 1 DOWNTO 0) := "0111000";

BEGIN
        MultiplexerOutput <= MI0 WHEN (SelectSignal = "000") ELSE
                             MI1 WHEN (SelectSignal = "001") ELSE
                             MI2 WHEN (SelectSignal = "010") ELSE
                             MI3 WHEN (SelectSignal = "011") ELSE
                             MI4 WHEN (SelectSignal = "100") ELSE
                             MI5 WHEN (SelectSignal = "101") ELSE
                             MI6 WHEN (SelectSignal = "110") ELSE
                             MI7 WHEN (SelectSignal = "111") ELSE
                             NullSymbol;

END MultiplexerArchitecture;
```

8.5. Entidad principal [Verilog]

```
module CombinationalMarquee(*):

parameter InputCodeSize = 3;
parameter InputPositionBusSize = 3;
parameter OutputSize = 7;
parameter NumberOfLetters = 5;
parameter NumberOfDisplays = 8;

input[InputCodeSize-1:0] C0,C1,2,C3,C4;
input(InputPositionBusSize-1:0) MarqueePosition;
output(OutputSize-1:0) D0,D1,D2,D3,D4,D5,D6,D7;
output(NumberOfDisplays-1:0) MarqueePositionOutput;

wire[OutputSize-1:0] EmptySymbol = 7'1111111 ;
wire[InputCodeSize-1:0] BusOfCodes;
wire[OutputSize*NumberOfLetters-1:0] BusOfCharacteres;

initial
begin

assign BusOfCodes = {C4,C3,C2,C1,C0}

for i in 1 to NumberOfLetters loop

Decoder_3_to_7 Decoder (
    .DecoderInput(BusOfCodes[3*1-1:3*(i-1)]), // input
    .DecoderOutput(BusOfCharacteres[7*i-1:7*(i-1)])
);
end loop;

Multiplexer_8_To_1 MultiplexerD0(
    .mi0(BusOfCharacteres(6:0)),
    .mi1(EmptySymbol),
    .mi2(EmptySymbol),
    .mi3(EmptySymbol),
    .mi4(BusOfCharacteres(34:28)),
    .mi5(BusOfCharacteres(27:21)),
    .mi6(BusOfCharacteres(20:14)),
    .mi7(BusOfCharacteres(13:7)),
    .multiplexeroutput(D0),
    .selectsignal(MarqueePosition)
);

Multiplexer_8_To_1 MultiplexerD1(
    .mi0(BusOfCharacteres(6:0)),
    .mi1(BusOfCharacteres(6:0)),
    .mi2(EmptySymbol),
    .mi3(EmptySymbol),
```

```
    .mi4(EmptySymbol),
    .mi5(BusOfCharacteres(6:0)),
    .mi6(BusOfCharacteres(6:0)),
    .mi7(BusOfCharacteres(6:0)),
    .multiplexeroutput(D1),
    .selectsignal(MarqueePosition)
);

Multiplexer_8_To_1 MultiplexerD2(
    .mi0(BusOfCharacteres(13:7)),
    .mi1(BusOfCharacteres(6:0)),
    .mi2(EmptySymbol),
    .mi3(EmptySymbol),
    .mi4(EmptySymbol),
    .mi5(BusOfCharacteres(34:28)),
    .mi6(BusOfCharacteres(27:21)),
    .mi7(BusOfCharacteres(20:14)),
    .multiplexeroutput(D2),
    .selectsignal(MarqueePosition)
);

Multiplexer_8_To_1 MultiplexerD3(
    .mi0(BusOfCharacteres(6:0)),
    .mi1(BusOfCharacteres(6:0)),
    .mi2(BusOfCharacteres(6:0)),
    .mi3(BusOfCharacteres(6:0)),
    .mi4(EmptySymbol),
    .mi5(EmptySymbol),
    .mi6(EmptySymbol),
    .mi7(BusOfCharacteres(6:0)),
    .multiplexeroutput(D3),
    .selectsignal(MarqueePosition)
);

Multiplexer_8_To_1 MultiplexerD4(
    .mi0(BusOfCharacteres(34:28)),
    .mi1(BusOfCharacteres(27:21)),
    .mi2(BusOfCharacteres(20:14)),
    .mi3(BusOfCharacteres(13:7)),
    .mi4(BusOfCharacteres(6:0)),
    .mi5(EmptySymbol),
    .mi6(EmptySymbol),
    .mi7(EmptySymbol),
    .multiplexeroutput(D4),
    .selectsignal(MarqueePosition)
);

Multiplexer_8_To_1 MultiplexerD5(
    .mi0(EmptySymbol),
```

```
    .mi1(BusOfCharacteres(34:28)),
    .mi2(BusOfCharacteres(27:21)),
    .mi3(BusOfCharacteres(20:14)),
    .mi4(BusOfCharacteres(13:7)),
    .mi5(BusOfCharacteres(6:0)),
    .mi6(EmptySymbol),
    .mi7(EmptySymbol),
    .multiplexeroutput(D5),
    .selectsignal(MarqueePosition)
);


Multiplexer_8_To_1 MultiplexerD6(
    .mi0(EmptySymbol),
    .mi1(EmptySymbol),
    .mi2(BusOfCharacteres(34:28)),
    .mi3(BusOfCharacteres(27:21)),
    .mi4(BusOfCharacteres(20:14)),
    .mi5(BusOfCharacteres(13:7)),
    .mi6(BusOfCharacteres(6:0)),
    .mi7(EmptySymbol),
    .multiplexeroutput(D6),
    .selectsignal(MarqueePosition)
);

Multiplexer_8_To_1 MultiplexerD7(
    .mi0(EmptySymbol),
    .mi1(EmptySymbol),
    .mi2(EmptySymbol),
    .mi3(BusOfCharacteres(34:28)),
    .mi4(BusOfCharacteres(27:21)),
    .mi5(BusOfCharacteres(20:14)),
    .mi6(BusOfCharacteres(13:7)),
    .mi7(BusOfCharacteres(6:0)),
    .multiplexeroutput(D7),
    .selectsignal(MarqueePosition)
);

Decoder_3_To_8 Dec(
    .decoderinput(MarqueePosition),
    .decoderoutput(MarqueePositionOutput)
);

end

end module;
```

8.6. Multiplexor [Verilog]

```
module Multiplexer_8_To_1(MI0,MI1,MI2,MI3,MI4,MI5,MI6,MI7,MultiplexerOutput,SelecSignal):

parameter InputSize = 7;
parameter OutputSize = 7;
parameter SelecSignalSize = 3;

input[InputSize-1:0] MI0,MI1,MI2,MI3,MI4,MI5,MI6,MI7;

output(OutputSize-1:0) MultiplexerOutput;

wire(OutputSize-1:0) MultiplexerOutput;

input(SelecSignalSize-1:0) SelecSignal;

parameter(OutputSize-1:0) NullSymbol = 7'b0111000;

assign MultiplexerOutput = (SelecSignal == 3'b000) ?  MI0 :
                           (SelecSignal == 3'b001) ?  MI1 :
                           (SelecSignal == 3'b010) ?  MI2 :
                           (SelecSignal == 3'b011) ?  MI3 :
                           (SelecSignal == 3'b100) ?  MI4 :
                           (SelecSignal == 3'b101) ?  MI5 :
                           (SelecSignal == 3'b110) ?  MI6 :
                           (SelecSignal == 3'b111) ?  MI7 :
                           (SelecSignal == 3'b000) ?  NullSymbol;

end;
end module;
```

8.7. Decodificador [Verilog]

```
module Decoder_3_To_7(DecoderInput, DecoderOutput);

parameter InputSize = 3;
parameter OutputSize = 7;
input[InputSize-1:0] DecoderInput;
output[OutputSize-1:0] DecoderOutput;
reg[OutputSize-1] DecoderOutput;

parameter[OutputSize-1:0] SymbolF = 7'b0111000;
parameter[OutputSize-1:0] SymbolP = 7'b0011000;
parameter[OutputSize-1:0] SymbolG = 7'b0100000;
parameter[OutputSize-1:0] SymbolA = 7'b0001000;

always@(*)
begin: DECODER
    case(DecoderInput)
        3'b000:
            begin
                DecoderInput<=SymbolF;
            end
        3'b001:
            begin
                DecoderInput<=SymbolF;
            end
        3'b010:
            begin
                DecoderInput<=SymbolF;
            end
        3'b011:
            begin
                DecoderInput<=SymbolF;
            end
        default:
            begin
                DecoderOutput <= 7'b1111111;
            end
    endcase
    end
endmodule
```