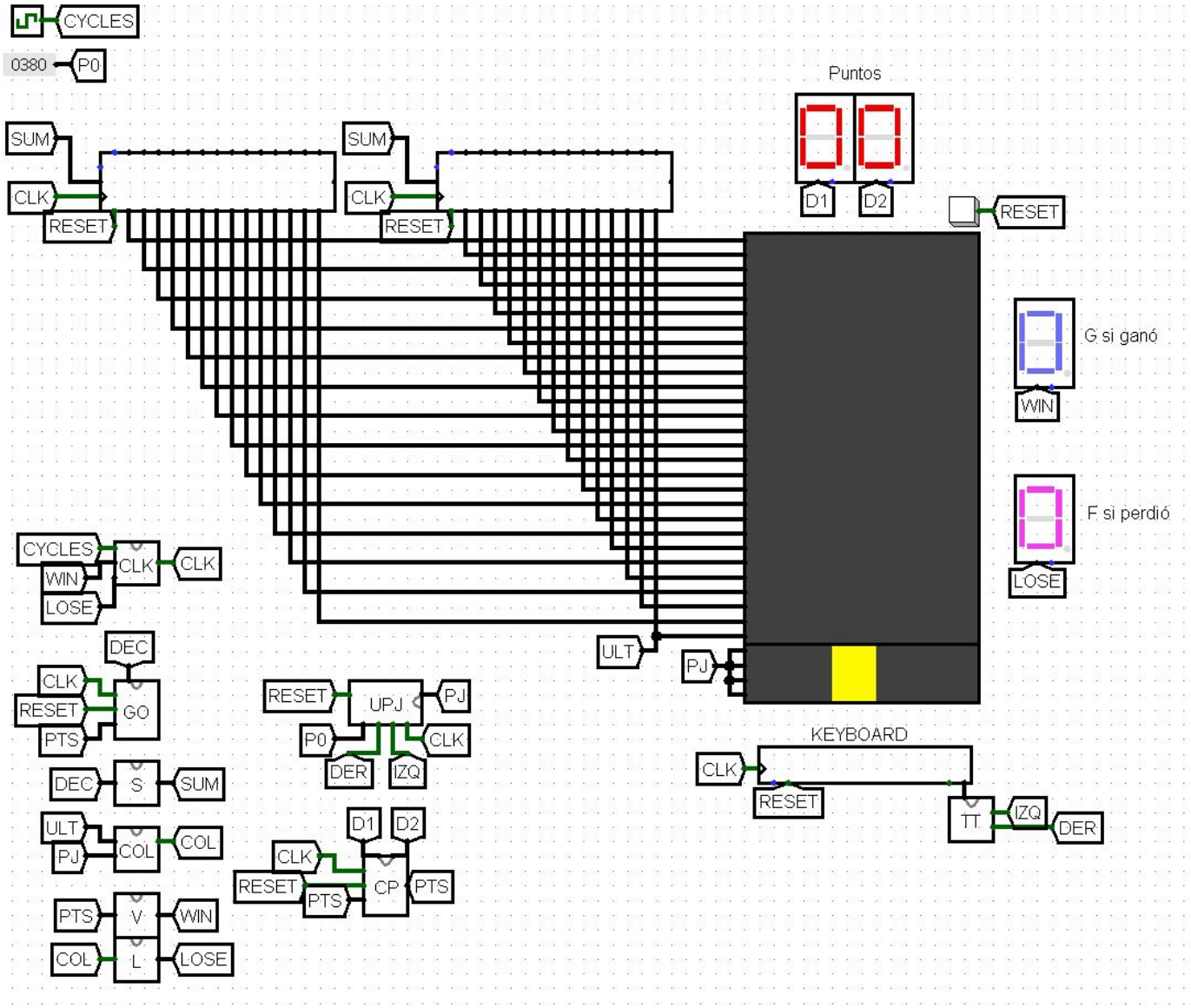
UNIVERSIDAD DEL NORTE
JUAN CARLOS UGARRIZA
JOSE DANIEL FERNANDEZ
STEVEN DAVID SILVA
Proyecto Final
Diseño Digital

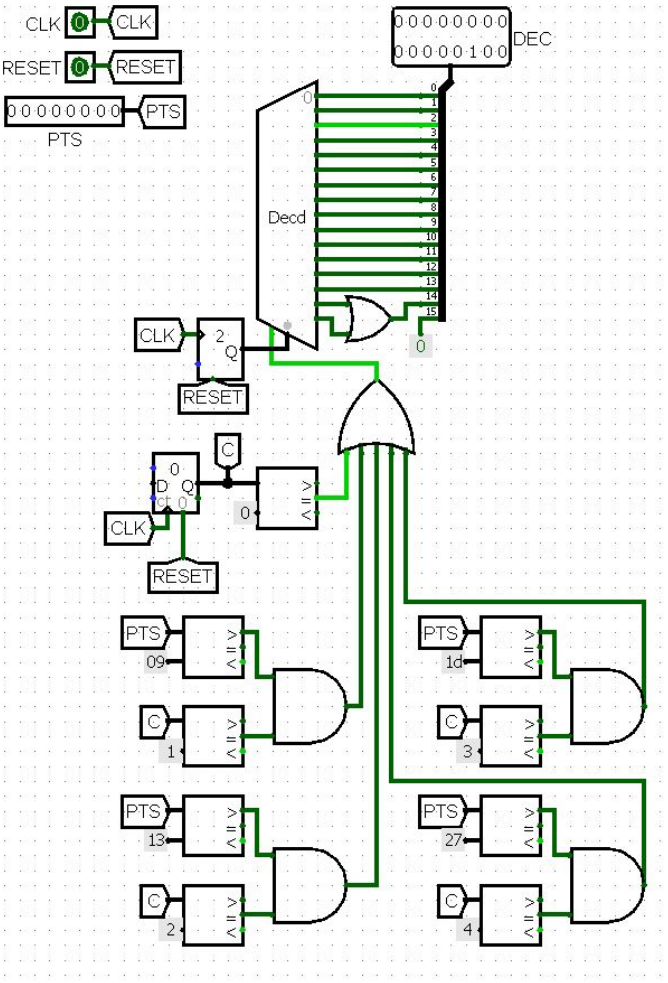
Circuito Main:

- La pantalla del juego es de 28 filas por 16 columnas, y la pantalla del jugador es de 4 filas por 16 columnas y todas las pantallas son controladas por filas.
- La pantalla del juego funciona con dos Shift Register y la pantalla del jugador con un byte de 16 bits que se reparte en las 4 filas.
- Los movimientos del jugador son controlados por un Keyboard que recibe como input cualquier carácter, pero solo se mueve cuando se ingresan las letras A (a la izquierda) y D (a la derecha) tanto minúsculas como mayúsculas.
- Los puntos son contados según los obstáculos evadidos y los puntos son mostrados con un Display.
- El botón de RESET hace “clear” a todos los Shift Registers, Registros y demás que se utilizan en las diferentes clases.
- Cuando se llega a los 50 puntos el jugador gana y se muestra en un Display la letra G (de ganar) y cuando pierde se muestra en otro Display la letra F (to pay respects).



A continuación se explican cada uno de los subcircuitos (desde ahora llamados “clases”).

Generar obstáculo:



1. Inputs
    a. CLK: el valor del reloj del main cuando no se ha ganado ni perdido.
    b. RESET: el valor de el botón en el main para resetear.
    c. PTS: Los puntos que tiene el jugador actualmente.

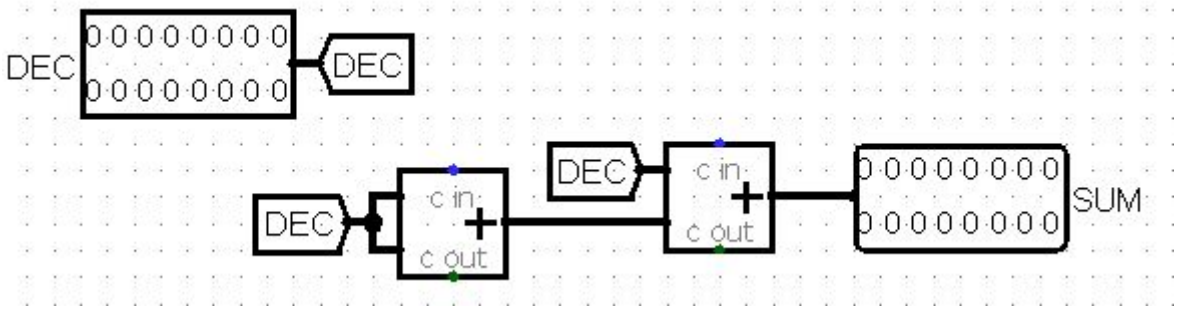
2. Outputs:
    a. DEC: posición del próximo obstáculo a crear.

3. Funcionamiento:
    Se genera un número aleatorio entre 0 y 15 para crear un obstáculo en esa posición. Cuando el jugador tiene más de 9 puntos se realiza este procedimiento dos veces para que se creen dos obstáculos secuencialmente y así con todos los rangos de puntaje por separado.

Suma:

1. Inputs:
    a. DEC: posición del próximo obstáculo a crear. Se trae de la clase Generar obstáculo.
2. Outputs:
    a. SUM: el obstáculo a generar con la cantidad necesaria de columnas (dos).

3. Funcionamiento:
    Se suma DEC consigo mismo 2 veces (se multiplica DEC por 3) para que se genere un 1 en la posición siguiente cosa que el obstáculo se cree con una dimensión de 2 columnas.

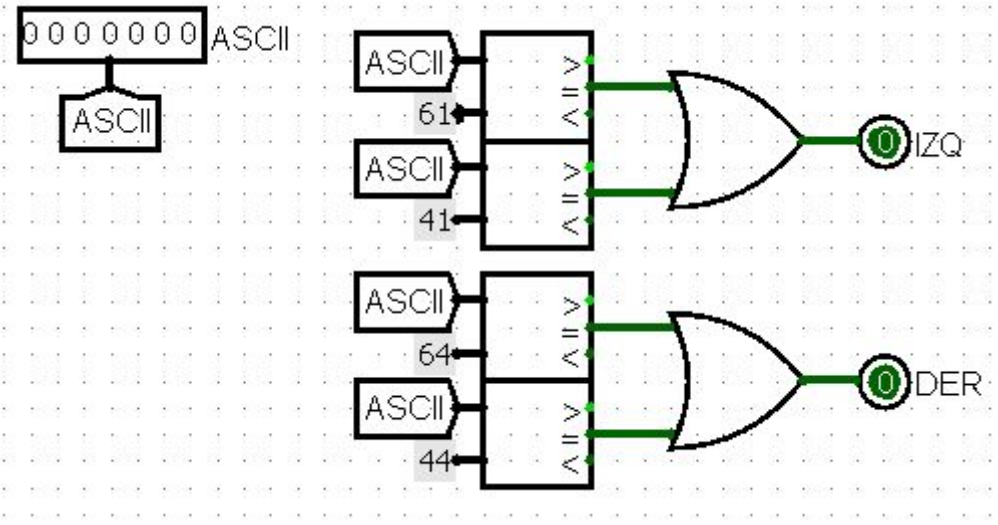


Traducción Teclado:

1. Inputs:
    a. ASCII: código ASCII proveniente del Keyboard en el main.

2. Outputs:
    a. IZQ: si se ingresa A ó a en el Keyboard, el valor de este bit es 1. De lo contrario, es 0.
    b. DER: si se ingresa D ó d en el Keyboard, el valor de este bit es 1. De lo contrario, es 0.

3. Funcionamiento:
    Se compara ASCII con constantes cuyo valor hexadecimal corresponde a las letras A y D, y para cada caso los outputs son diferentes.



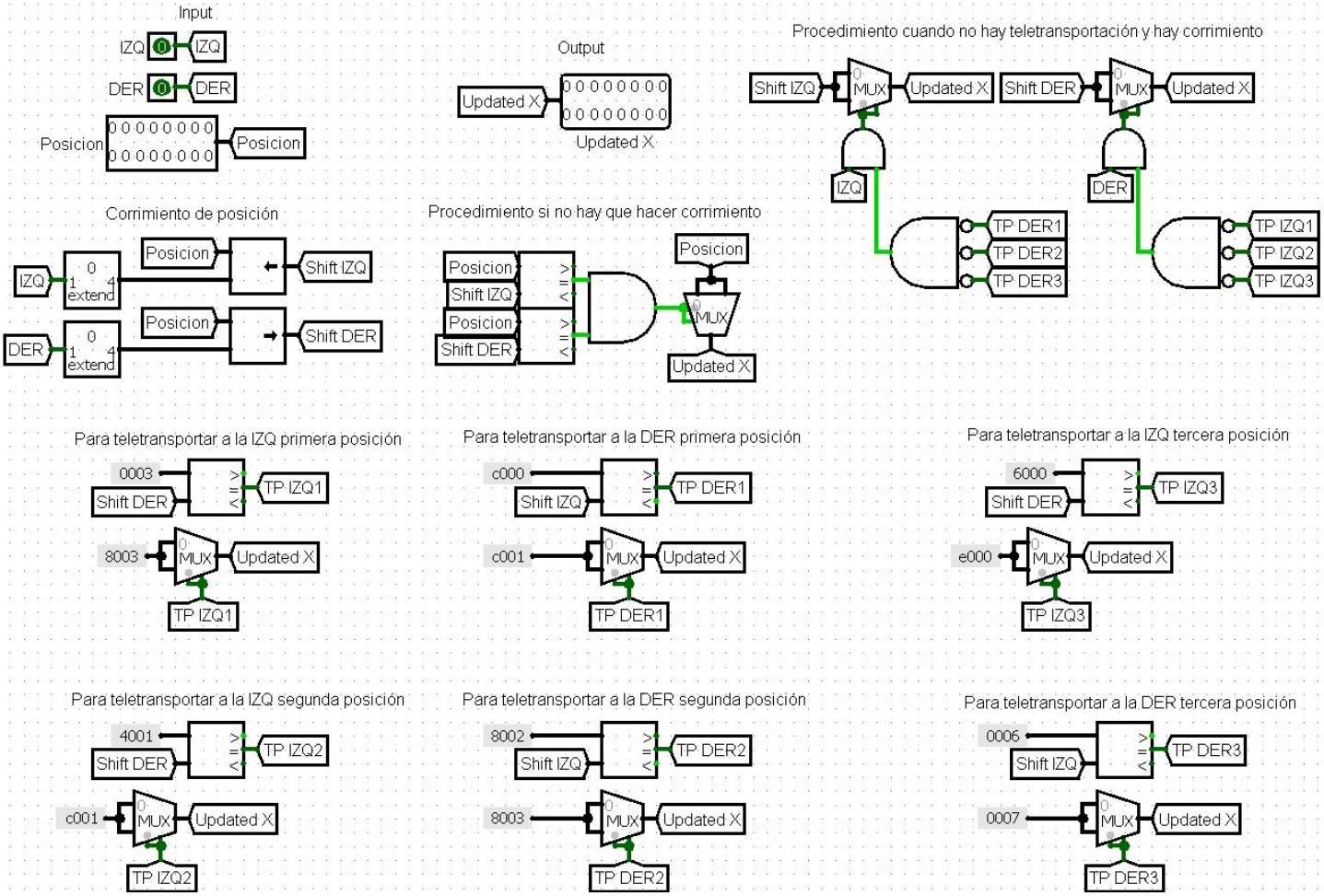
Movimiento:

1. Inputs:
    a. IZQ: proveniente de Traducción Teclado, este bit corresponde a la dirección (izquierda) en que se mueve el personaje.
    b. DER: proveniente de Traducción Teclado, este bit corresponde a la dirección (derecha) en que se mueve el personaje.
    c. Posicion: proveniente de la clase Update PJ, este byte de 16 bits corresponde a la posición actual del personaje.

2. Outputs:
    a. Updated X: la posición del personaje al hacerle el corrimiento correspondiente (en caso de hacerse).

3. Funcionamiento:

    Esta clase es la responsable de realizar el corrimiento de Posicion, el cual corresponde a la posición actual del personaje. Si tanto IZQ como DER son 0, entonces no se debe realizar un corrimiento y la posición queda igual. Si hay que realizar corrimiento y no hace falta teletransportar el personaje de un lado a otro, se hace output de la posición con el corrimiento. Si hay que hacer teletransportación y (por ende) corrimiento, se hace output de una posición dada en constantes según la dirección del corrimiento. El personaje puede estar dividido en los extremos de la pantalla.



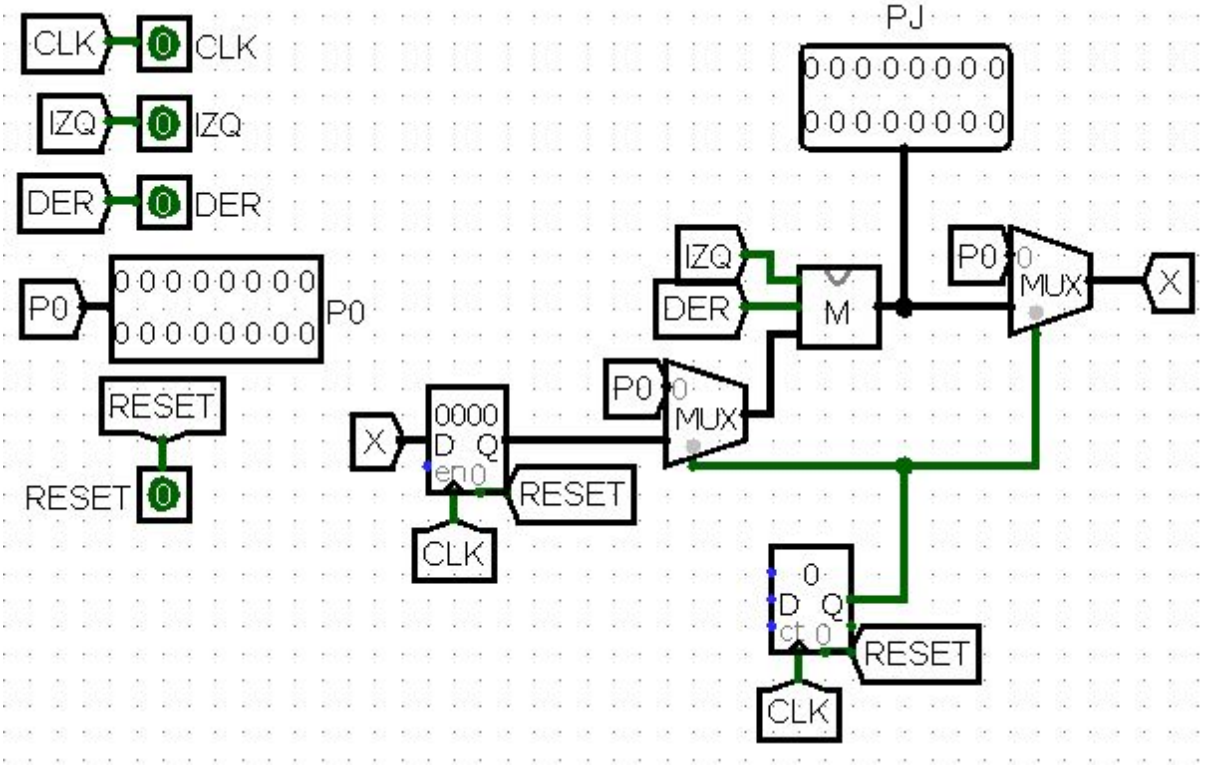
Update PJ:

1. Inputs:
    a. CLK: el valor del reloj del main cuando no se ha ganado ni perdido.
    b. IZQ: proveniente de Traducción Teclado, este bit corresponde a la dirección (izquierda) en que se mueve el personaje.
    c. DER: proveniente de Traducción Teclado, este bit corresponde a la dirección (derecha) en que se mueve el personaje.
    d. P0: posición inicial del personaje denotada como un byte de 16 bits. Este valor corresponde al hexadecimal 0380.
    e. RESET: el valor de el botón en el main para resetear.

2. Outputs:
    a. PJ: posición actualizada del personaje luego de hacer el movimiento correspondiente.

3. Funcionamiento:

    En esta clase se hace el movimiento del personaje. P0, que corresponde a la posición inicial del personaje, es tomado y según los valores de IZQ y DER se realiza un corrimiento en la dirección correspondiente haciendo uso de la clase Movimiento. Esta nueva posición con el corrimiento es guardada en un registro para que se “actualice” la posición actual del personaje y se realice este mismo procedimiento las veces que sea requerido.



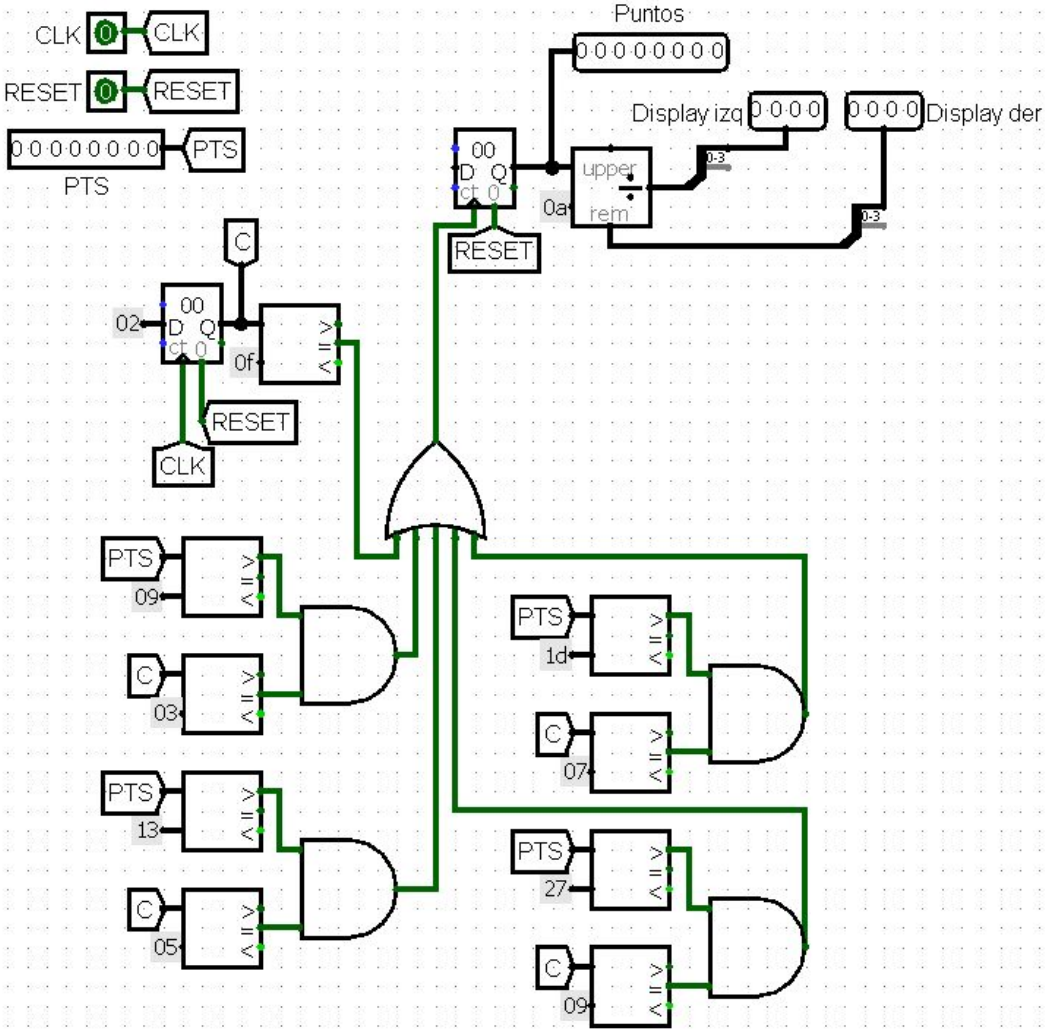
Contador puntos:

1. Inputs:
    a. CLK: el valor del reloj del main cuando no se ha ganado ni perdido.
    b. RESET: el valor de el botón en el main para resetear.
    c. PTS: los puntos que tiene el jugador actualmente.

2. Outputs:
    a. Puntos: los puntos actuales del jugador.
    b. Display izq: valor hexadecimal de las decenas del puntaje.
    c. Display der: valor hexadecimal de las unidades del puntaje.

3. Funcionamiento:

    Se cuentan los puntos del jugador según el número de ciclos de CLK que han transcurrido y según el número de obstáculos que se evadieron (lo cual es medido por la cantidad de ciclos de CLK que han transcurrido sin que el jugador pierda o gane).



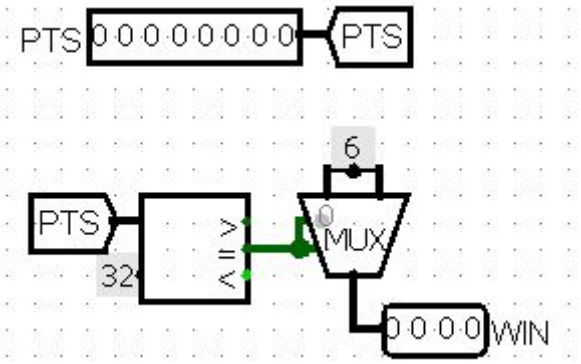
Victoria:

1. Inputs:
    a. PTS: los puntos que tiene el jugador actualmente.

2. Outputs:
    a. WIN: el valor hexadecimal de la letra impresa en el Display del main al ganar.

3. Funcionamiento:

    Se compara que los puntos actuales del jugador coincidan con la condición de victoria (50 puntos). Si coinciden, retorna el valor hexadecimal 6, si no, retorna 0.



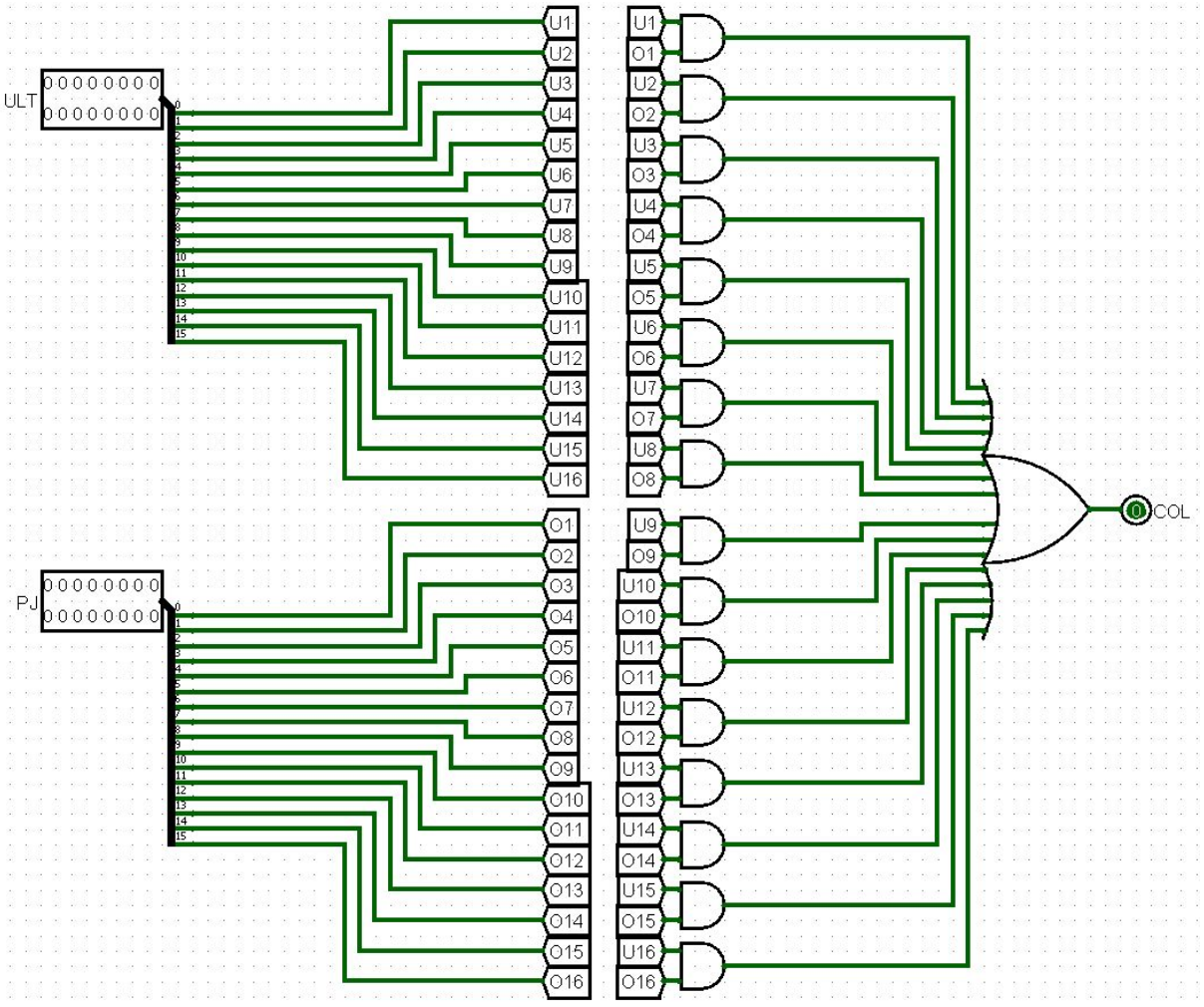
Colisión:

1. Inputs:
    a. ULT: byte que contiene el valor de la última fila de la pantalla de juego.
    b. PJ: byte que contiene el valor de las filas de la pantalla del jugador.

2. Outputs:
    a. COL: es un bit que muestra 1 si hubo colisión entre jugador y obstáculo y 0 si no.

3. Funcionamiento:

    Se separan los bits de ULT y de PJ utilizando un Splitter para comparar si hay bits comunes en 1 en cada índice de los bytes. Si se cumple en siquiera un índice, retorna COL como 1, si no, se retorna COL como 0.



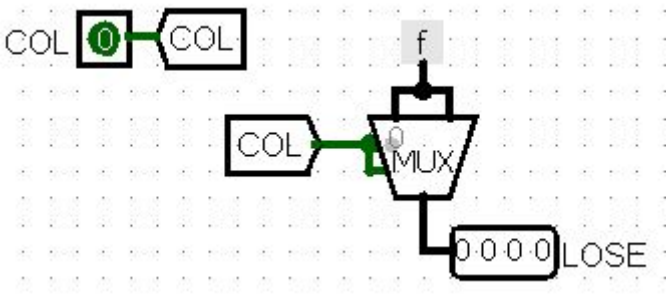
Lose:

1. Inputs:
    a. COL: es un bit que muestra 1 si hubo colisión entre jugador y obstáculo y 0 si no.

2. Outputs:
    a. LOSE: el valor hexadecimal de la letra impresa en el Display del main al perder.

3. Funcionamiento:

    El circuito utiliza un multiplexor para preguntar si hubo una colisión, y si la hubo, retorna el valor hexadecimal de la letra F para indicar que el jugador ha perdido.



Reloj:

1. Inputs:
    a. WIN: el valor hexadecimal de la letra impresa en el Display del main al ganar.
    b. LOSE: el valor hexadecimal de la letra impresa en el Display del main al perder.
    c. CYCLES: Valores del reloj en la simulación.

2. Outputs:
    a. CLK: el valor del reloj del main cuando no se ha ganado ni perdido.

3. Funcionamiento:

    La función de este circuito es preguntar si el jugador ganó o perdió. Si alguna de las dos condiciones se cumple, CLK se debe detener para así detener todo el juego.