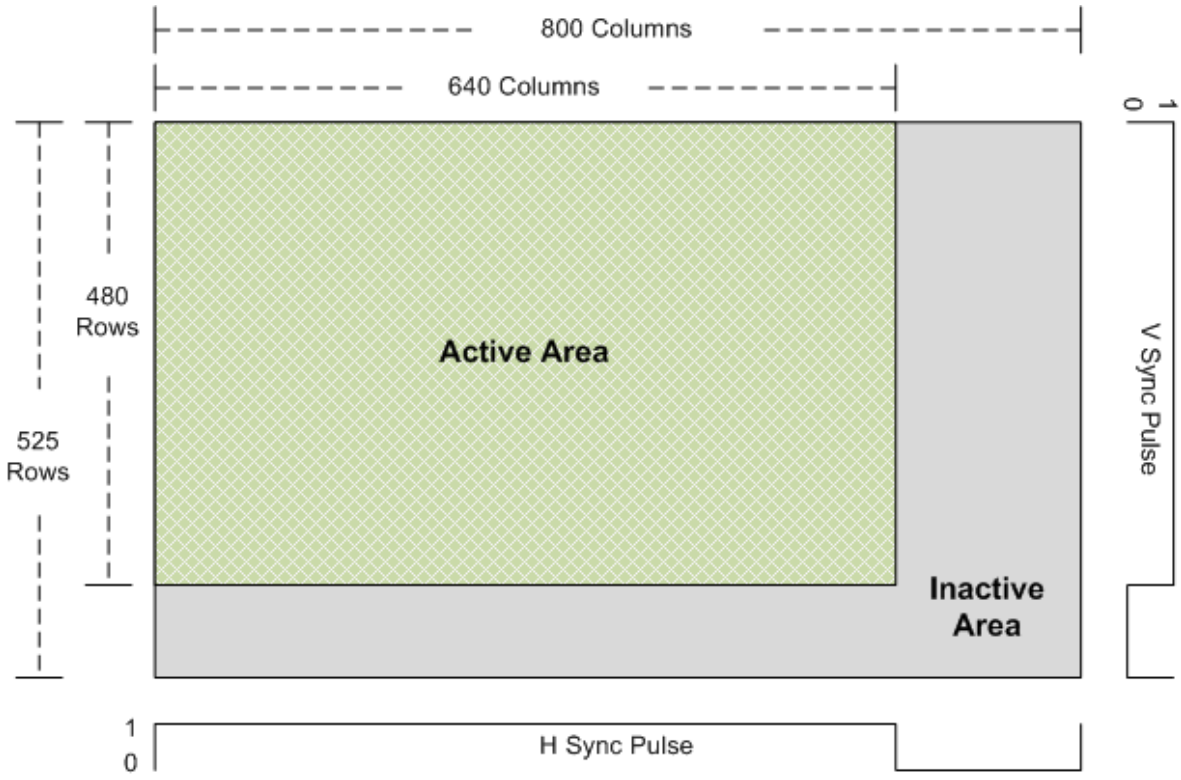
INTRODUCTION:”

DESARROLLO:

Primero describimos el cómo programamos el controlador VGA, que es prácticamente el cimiento del proyecto, tomamos como referencia el siguiente diagrama:



En si el objetivo para que el puerto VGA pueda interpretar la imagen es generar las ondas cuadradas descritas por H_Sync_Pulse y por V_Sync_Pulse y mandarlas al VGA.

En base a este diagrama, y dado que tenemos un oscilador de 50MHZ, podemos aspirar a una mejor resolución: [VESA Signal 800 x 600 @ 72 Hz timing (tinyvga.com)]

VESA Signal 800 x 600 @ 72 Hz timing

General timing

Screen refresh rate	72 Hz
Vertical refresh	48.076923076923 kHz
Pixel freq.	50.0 MHz

Horizontal timing (line)

Polarity of horizontal sync pulse is positive.

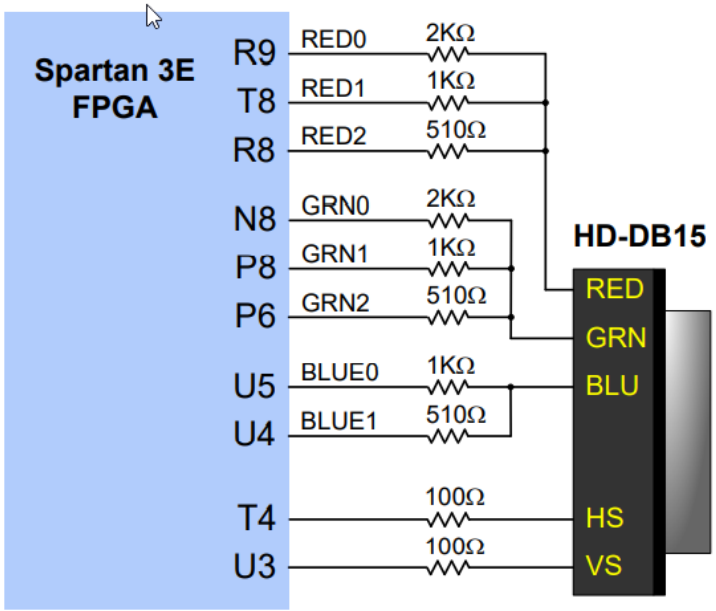
Scanline part	Pixels	Time [µs]
Visible area	800	16
Front porch	56	1.12
Sync pulse	120	2.4
Back porch	64	1.28
Whole line	1040	20.8

Vertical timing (frame)

Polarity of vertical sync pulse is positive.

Frame part	Lines	Time [ms]
Visible area	600	12.48
Front porch	37	0.7696
Sync pulse	6	0.1248
Back porch	23	0.4784
Whole frame	666	13.8528

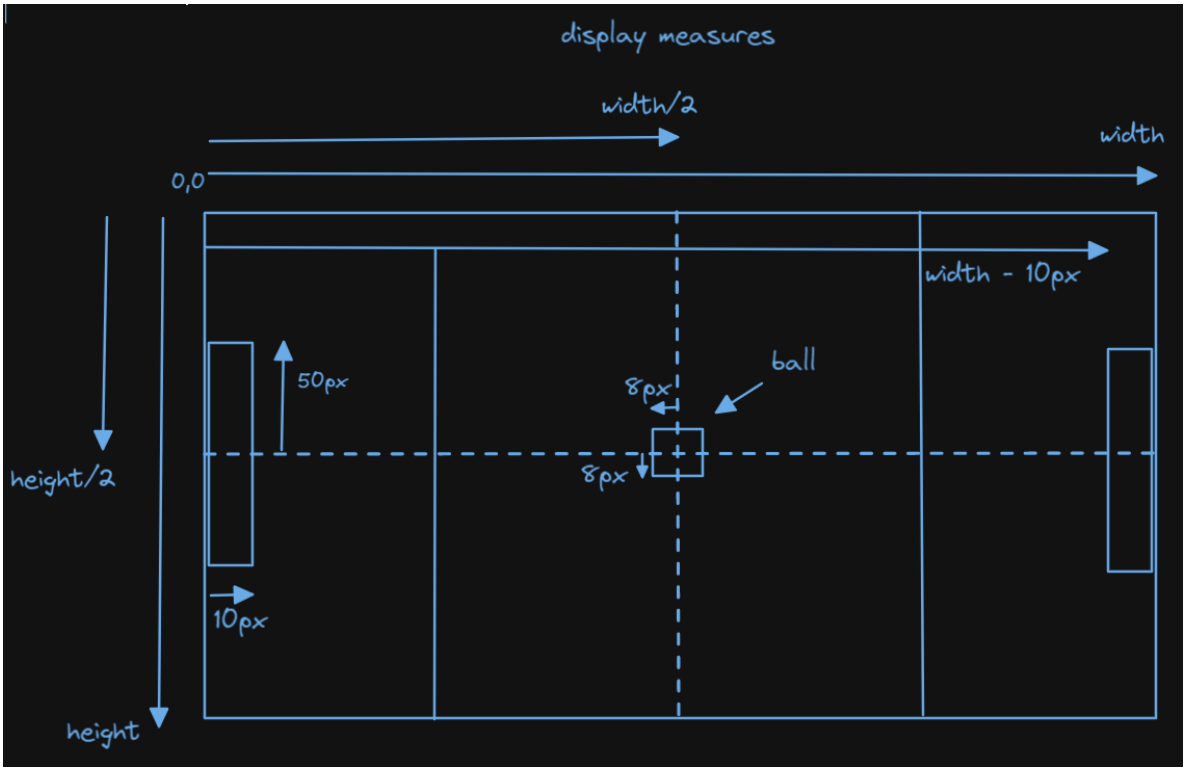
Normalmente para producir los colores tenemos 3 salidas (1 por cada color RGB) que van al VGA y que tendríamos que usar un divisor de voltaje o algo similar a un DAC para generar una variedad de colores, sin embargo, la nexys 2 ya usa un arreglo de resistencias que nos genera hasta 256 colores diferentes



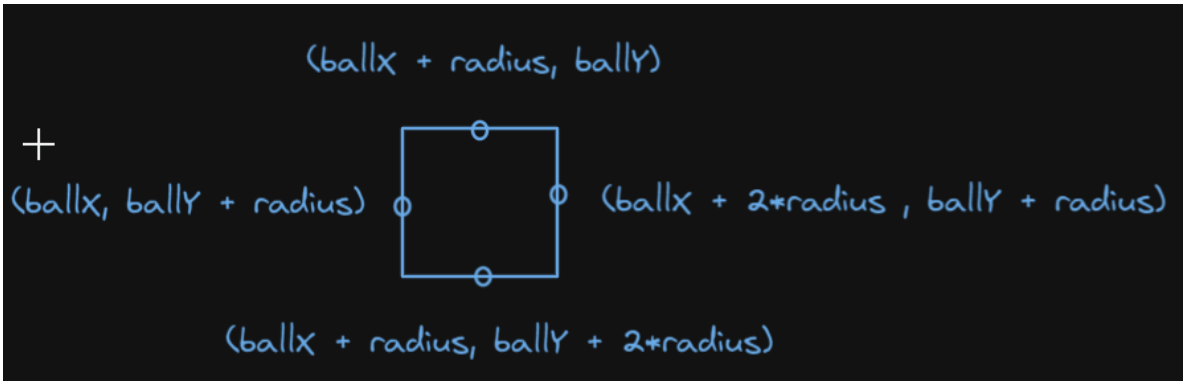
Ya teniendo el controlador VGA, comenzamos a discutir como hacer el tablero del juego, y los distintos elementos, quedando en los siguientes diagramas:

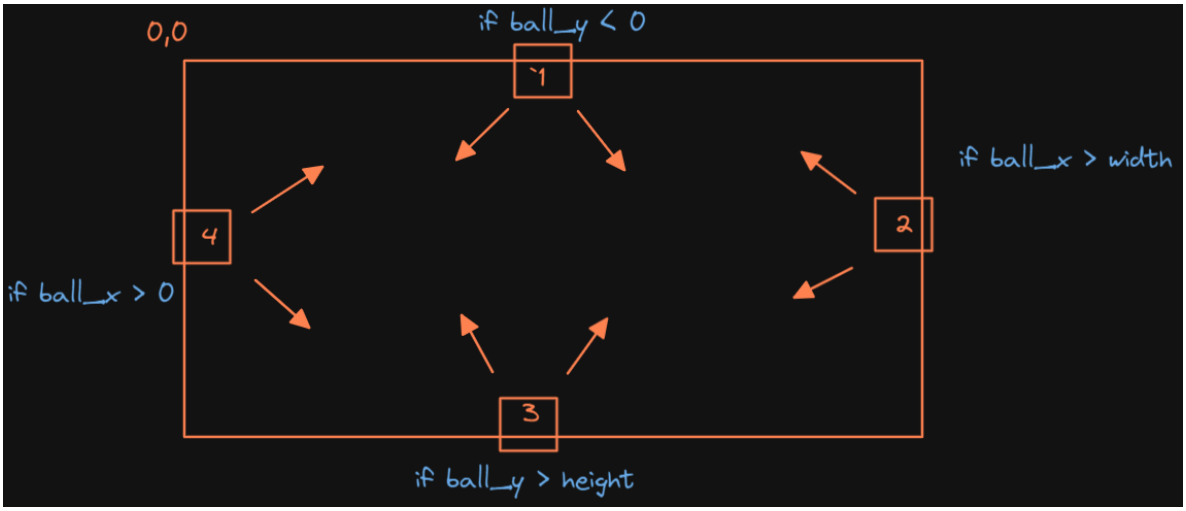
Importante el origen de la pantalla, esta en la parte superior izquierda, esto debido a que el VGA

Dibuja la imagen de izquierda a derecha y de arriba hacia abajo



Las colisiones las detectamos mediante la pelota, debido a que es más fácil verificar a que golpeo la pelota que, quien golpeo a la pelota. En la imagen se especifican los puntos de contacto que detectan las colisiones.





El cuadro de arriba especifica las condiciones y posibilidades de rebote, por ejemplo, las posibilidades de 1 y 3 dependen si la bola se mueve a la izq. O derecha,

Pero es simple en 1 y 3 no nos interesa modificar el movimiento en “x” así que si se cumple “1” solo cambiamos ‘Y’ sumándoles un incremento de forma que se mueva hacia abajo(por como el origen esta configuración)

Si choca en ‘3’ entonces le restamos un incremento de forma que se mueva hacia arriba

Ahora bien, en x, no nos interesa ‘y’ así que si:

Choca en 2: le restamos un incremento para que se mueva a la izq.

Si choca en 4: le sumamos un incremento para que se mueva a la derecha.

```
1013    if w_vel_dir_x = '1' then
1014    |   w_ball_x <= w_ball_x - w_ball_vel_x;
1015    else
1016    |   w_ball_x <= w_ball_x + w_ball_vel_x;
1017    end if;
1018
1019    if w_vel_dir_y = '1' then
1020    |   w_ball_y <= w_ball_y - w_ball_vel_y;
1021    else
1022    |   w_ball_y <= w_ball_y + w_ball_vel_y;
1023    end if;
```

Sumamos el incremento

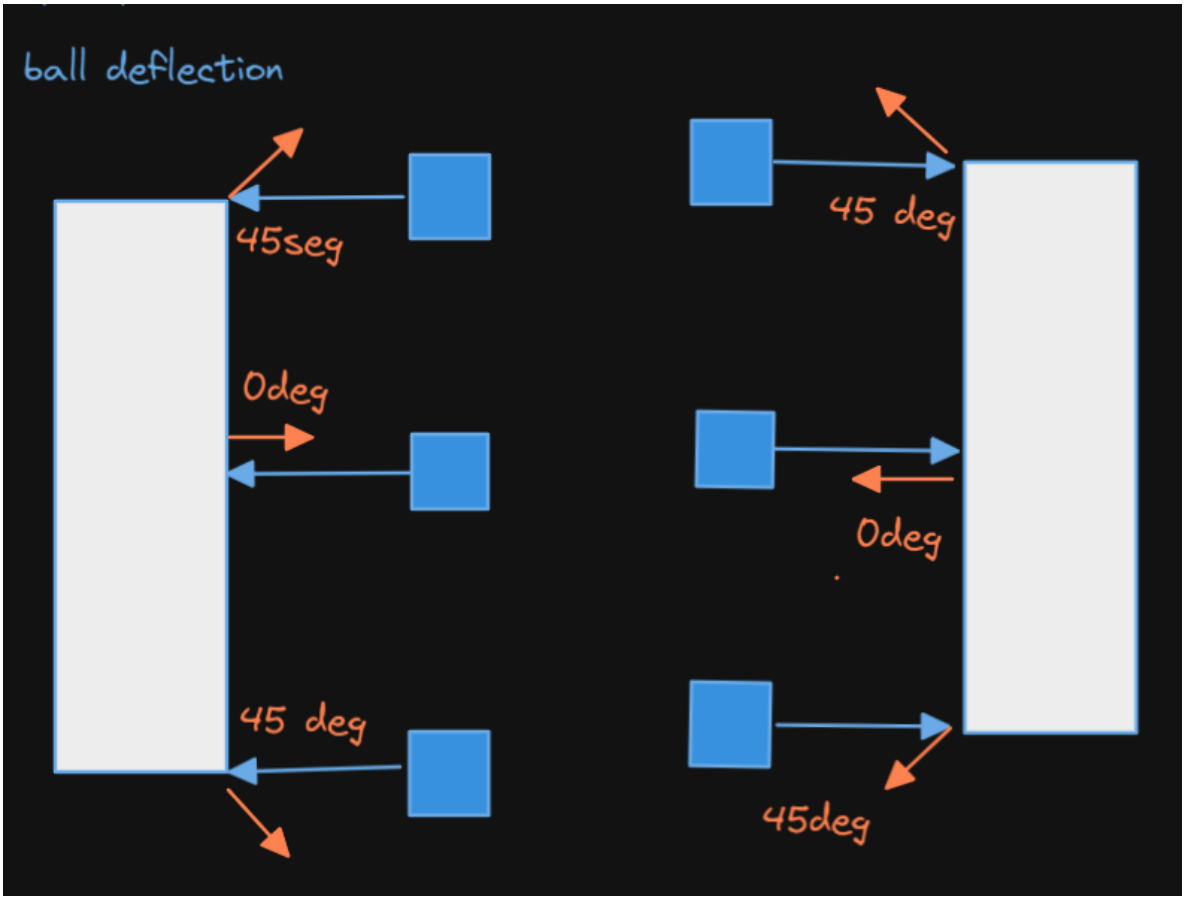
Restamos el incremento

Igual pero en Y

El resto del código modifica, el incremento a realizar en el próximo ciclo de reloj, en base a si ocurrió una colisión, por ejemplo, si ocurre colisión en la izq., entonces modificamos variable de dirección que causara que el código anterior sume el incremento y por tanto se mueva a la derecha, y así con el resto del código.

```
1025    if w_collision_in_left = '1' then
1026      --go to right
1027      w_vel_dir_x <= '0';
1028    elsif w_collision_in_right = '1' then
1029      --go to left
1030      w_vel_dir_x <= '1';
1031    end if;
1032
1033    if w_collision_in_top = '1' then
1034      --go to down
1035      w_vel_dir_y <= '0';
1036    elsif w_collision_in_bottom = '1' then
1037      --go up to
1038      w_vel_dir_y <= '1';
1039    end if;
```

En cuanto al paddle, una vez la pelota colisiones tenemos que decidir cómo, dirigir la pelota, lo cual se especifica en el siguiente diagrama:



```
if w_collide_mid_to_bottom = '1' and w_collide_in_p1 = '1' then
  --go to right and bottom
  w_vel_dir_y <= '0';
elsif w_collide_in_p1 = '1' and w_collide_mid_to_top = '1' then
  --go to right and top
  w_vel_dir_y <= '1';
end if;
```

El código, va dentro de la colisión en la izquierda por que hay se encuentra el paddle número 1,

Si ocurre buscamos donde la pelota golpeo al paddle y modificamos la variable de dirección de la pelota.

Para el paddle numero 2 buscamos si ocurrió una colisión en la derecha y haríamos lo mismo.

Para poder manipular usamos un par de joysticks, estos varían el voltaje con la ayuda de unos potenciómetros, lo cual puede ser decodificado en numero de hasta 10 bits(0 a 1023), también estos dispositivos son auto centrados(regresan a la mitad del valor automáticamente), por lo que de 512 a 1023 lo podemos interpretar como UP_KEY y de 0 a 512 como DOWN_KEY, sin embargo usamos un Arduino uno para decodificar el valor analógico, y solo disponemos de 13 puertos de salida digital por lo que cada joystick solo puede representar valores de 0 a 64. Dado que estos 6 bits irán al fpga y considerando posibles errores, tenemos la siguiente configuración:

Para paddle 1:

0 – 27 : DOWN KEY

27 – 36 : inactive

36 – 64: UP KEY

Para paddle 2:

0 – 27 : UP KEY

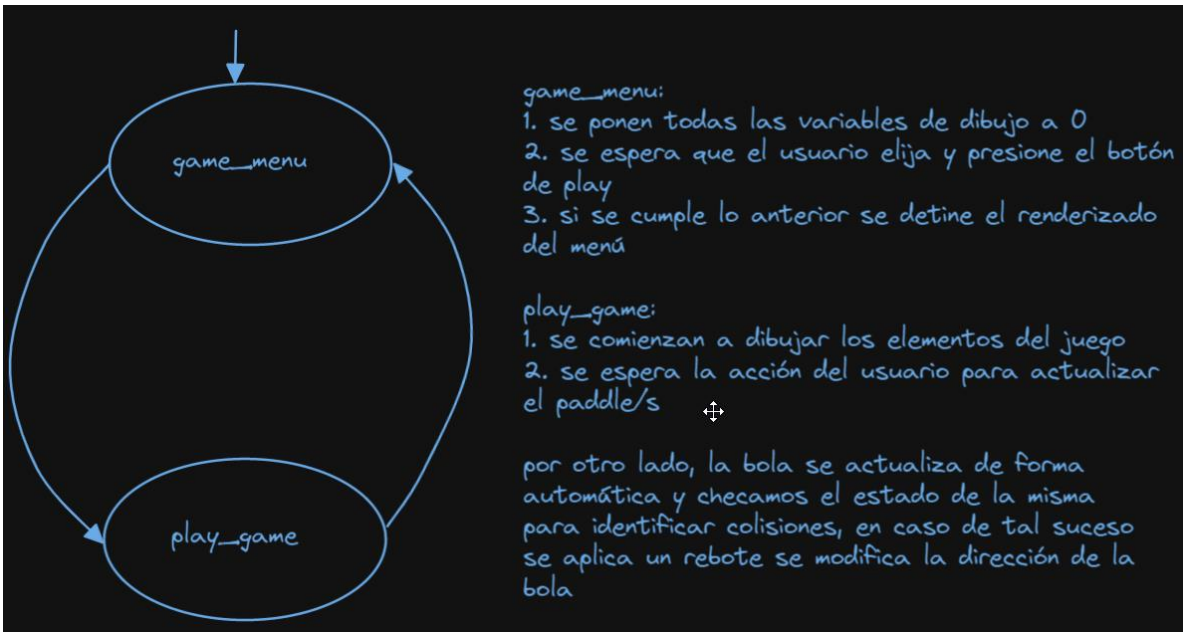
27 – 36 : inactive

36 – 64: DOWN KEY

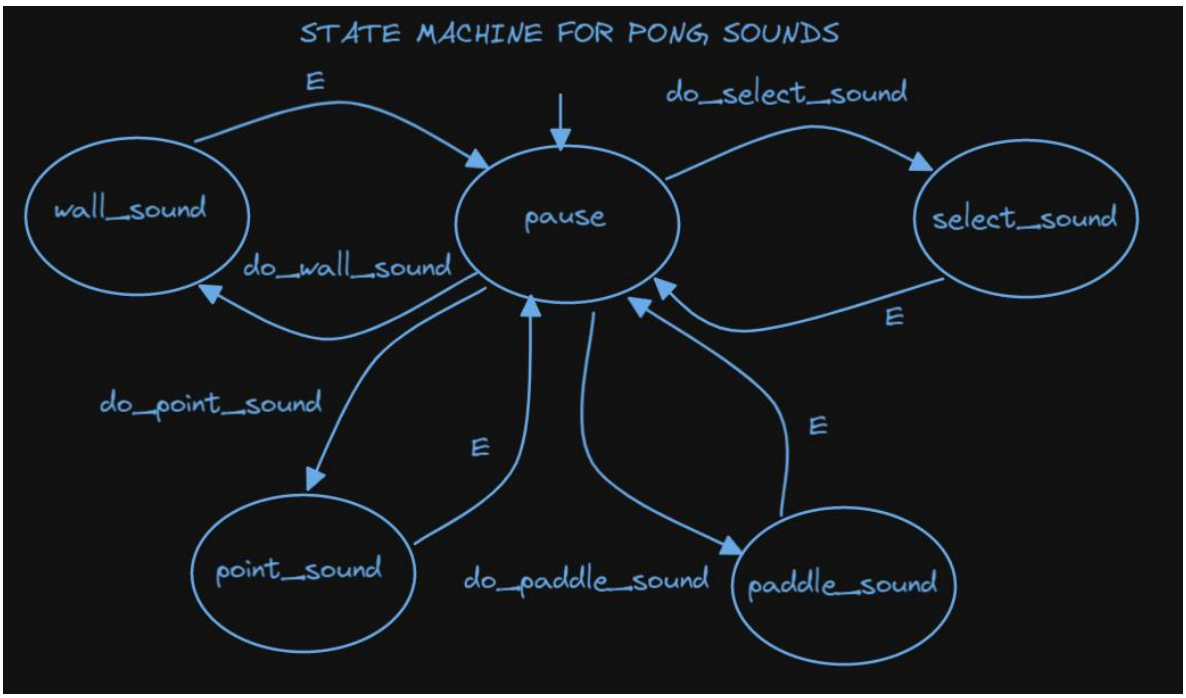
El código, realiza la conversión de analógico a digital

```
int ps1_value;
int ps2_value;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

  DDRD = B11111111; //(0 - 7)
  DDRB = B11110000; //(8 - 13)
}

void loop() {
  ps1_value = analogRead(A0);
  ps2_value = analogRead(A1);

  byte ps1 = ps1_value >> 4 | (ps2_value << 2 & 0x0C0);
  byte ps2 = (ps2_value >> 6);
  PORTD = ps1;
  PORTB = ps2;
  delay(12.5);
}
```

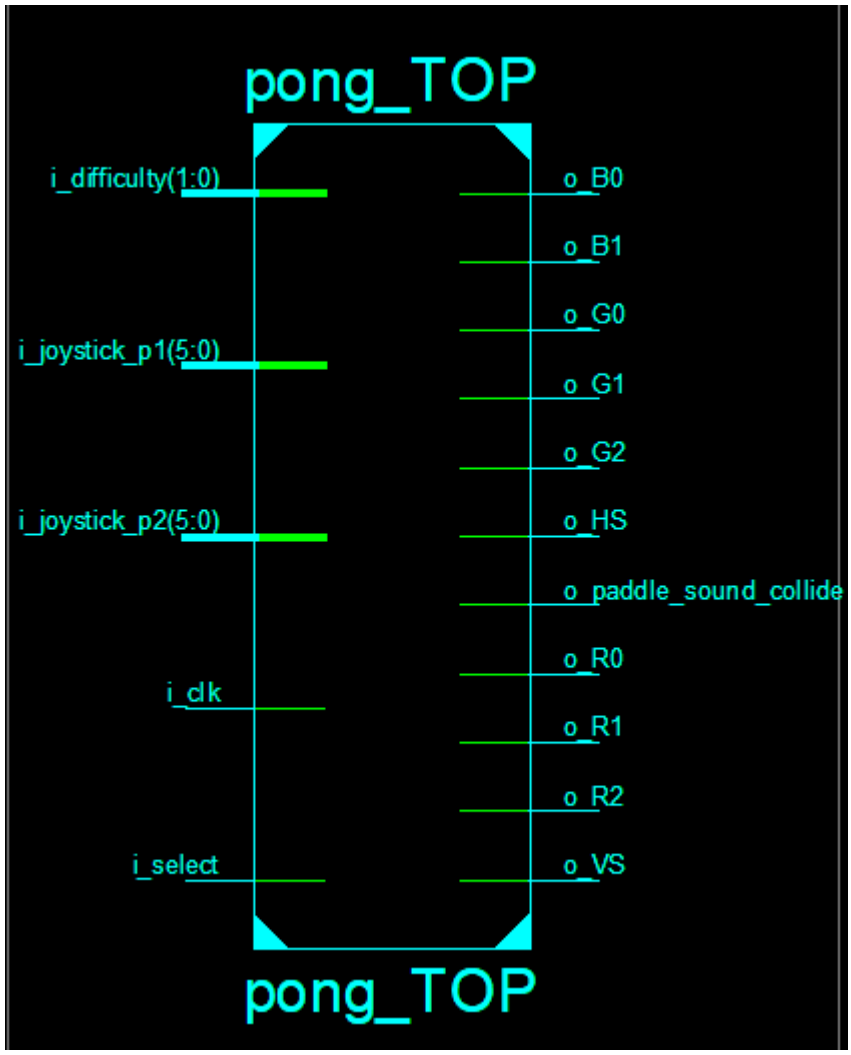
El flujo del juego sería el siguiente



El modulo de sonido funciona en base a una maquina de estados que en base a una señal de entrada reproduce el sonido una sola vez y regresa automáticamente al estado de pausa, como un escucha de evento.



El esquema del integrado, producido por ISE Xilinx:



CONCLUSION: