Ejercicio 2. Diseño Hardware DE2-115. Cuestiones Prácticas Co-diseño

Apellidos y Nombre: Coves Puelma, Paola

Ejercicio 2: Diseño de un hardware de referencia para DE2-115

Ejercicio 0: Proyecto software test_mtl

1. ¿Cómo se puede modificar la velocidad de movimiento del gusanito por el display 7-segmentos? ¿Se podría establecer una velocidad exacta de cambio cada 50 ms? ¿Por qué?

   La velocidad de movimiento del gusanillo por el display de 7-segmentos se puede modificar a partir de una sentencia, en el código test_mtl.c. Dicha sentencia, la cual se muestra a continuación, corresponde a un bucle que se ejecutará hasta que la variable entera *delay_count* sea exactamente el valor que le hayamos introducido.

   for(delay_count=200000; delay_count !=0; --delay_count); // retardo

   Para concretar la velocidad se modifica a partir de esta variable *delay_count*, esto produce que cuanto mayor sea el valor que le asignemos, el retardo de la señal aumentará, por lo que el gusanillo ira más despacio. Por otro lado, la posibilidad de modificar la velocidad abre las puertas a que se pueda establecer una velocidad exacta de cambio, por ejemplo, cada 50 ms ya que conocemos la frecuencia del oscilador que posee la placa DE2, f=50 MHz (además de haber configurado el resto de los relojes):

   $$50\ ms * 50000000$$

2. ¿Qué función tiene la línea de código: **`while (*KEY_ptr);`** ? ¿Qué ocurre si la eliminamos?

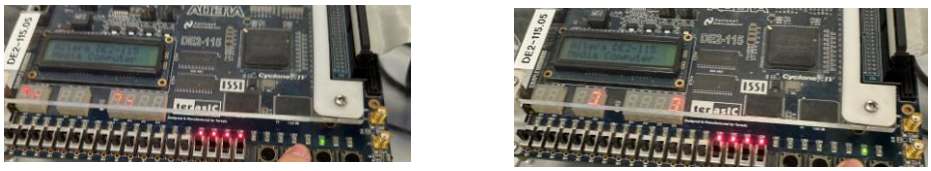

Figura 1. Izquierda: Con la instrucción. Derecha: Sin la instrucción.

Dicha función espera a que se vuelva a pulsar cualquier (de ahí el selector universal *) KEY. Es decir, se entra al while y se detiene el movimiento del

gusanillo en el display 7-segmentos. Por lo que cuando la eliminamos, y seguidamente probamos el funcionamiento siempre se obtiene el mismo resultado en el display 7-segmentos. Las siguientes imágenes representan el resultado con y sin dicha instrucción.

3. En la subrutina de dibujar un cuadrado en la pantalla **`MTL_box`**, aparece la siguiente línea de código: **`offset = (row << 9) + col;`** ¿Que función realiza esta línea de código? ¿A qué se debe ese 9? ¿Se puede modificar el 9 por otro valor? ¿Cuál sería el efecto?

En la línea de código *offset = (row << 9) + col*, offset nos dará el direccionamiento x-y (fila y columna). Esto se debe a que cuando hemos realizado las configuraciones se ha especificado un direccionamiento x-y, en otras palabras, la instrucción sirve para posicionar el punto que deseamos pintar. Por otro lado, el valor de 9 se debe a que tenemos 400 pixeles (para la componente x) y para representarlo como mínimo necesitamos 9 bits ($2^9 = 512$).

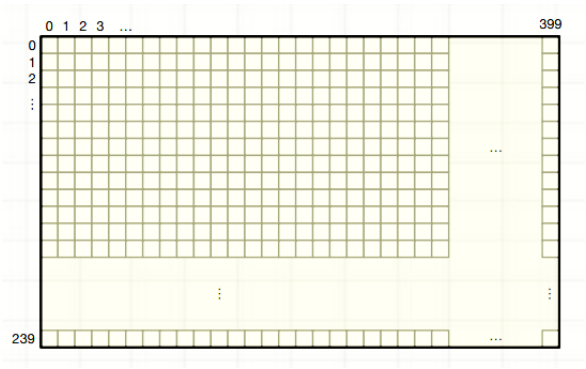

Figura 2: Almacenamiento de un Imagen en Memoria.

Es posible modificar el valor de 9 por otro valor, sin embargo, esto provocara que no nos ajustemos al tamaño de la pantalla (o en este caso al tamaño de margen que está guardando en memoria). Por lo que las direcciones no serán correctas y se pintarán puntos en posiciones no deseadas. Por ejemplo, en la siguiente figura se puede apreciar el efecto que produce modificar el valor, en concreto por 8 bits ($2^8 = 256$).

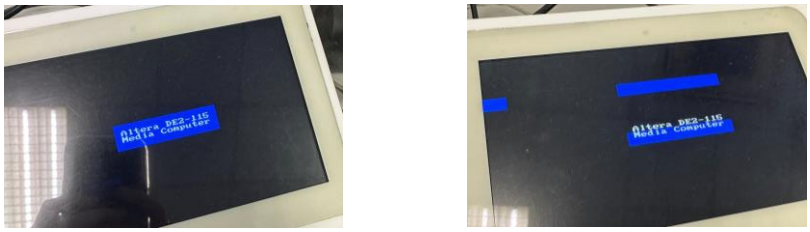

Figura 3. Izquierda: resultado con 9 bits. Derecha: resultado con 8 bits.

Ejercicio 2: El uso del HAL

1. ¿En qué parte del código introduciríamos una instrucción que dibujase una línea horizontal en la pantalla MTL?

La manera correcta de proceder seria colocar la instrucción que dibuja una línea horizontal en la pantalla MTL junto al resto de sentencias de dibujo. Depende de lo que queramos realizar la podemos colocar en el grupo de dibujo que permanece estático en la pantalla o dentro del bucle para dotarla de movimiento. Si optamos por la primera opción el cubo que esta en movimiento borrara los pixeles de nuestra línea al pasar por ella, como se observa en la siguiente figura.

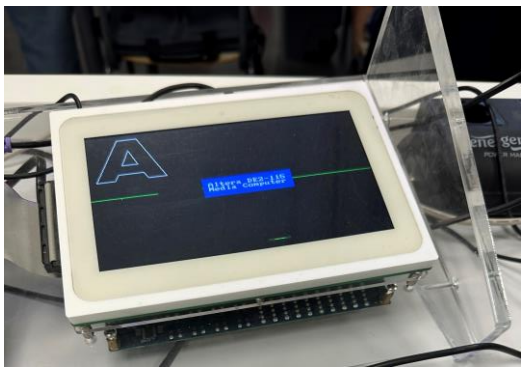

Figura 4: Resultado en la pantalla MTL de dibujar una línea horizontal.

2. ¿Cuál sería la instrucción a colocar?

Para conocer la instrucción a colocar y obtener el resultado mostrado en la figura anterior, debemos de recurrir al manual *Video.pdf*.

alt_up_pixel_buffer_draw_hline

Prototype:	`void alt_up_pixel_buffer_draw_hline(alt_up_pixel_buffer_dev *pixel_buffer, int x0, int x1, int y, int color, int backbuffer)`
Include:	`<altera_up_avalon_pixel_buffer.h>`
Parameters:	`pixel_buffer` – the pointer to the VGA structure `x0`, `x1`, `y` – coordinates of the left (x0,y) and the right (x1,y) end-points of the line `color` – color of the line to be drawn `backbuffer` – set to 1 to select the back buffer, otherwise set to 0 to select the current screen.
Returns:	0 if complete, 1 if still processing
Description:	This function draws a horizontal line of a given color between points (x0,y) and (x1,y).

Figura 5: Sintaxis de la instrucción.

Sin embargo, la instrucción actual que produce una correcta compilación tiene un cambio de nombre en el que se incluye la palabra *dma*. La siguiente sentencia muestra la sintaxis de la instrucción, así como los parámetros a colocar (si se desea que la línea horizontal cubra toda la pantalla MTL).

alt_up_pixel_buffer_dma_draw_hline(pixel_buffer_dev_MTL, 0, 399, 120, 0x07e0, 0);

3. ¿Cómo se podría hacer para que la línea horizontal se desplazase a la misma velocidad que el cubo naranja recorriendo la pantalla de arriba abajo?

Para hacer que la línea horizontal se desplace a la misma velocidad que el cubo naranja recorriendo la pantalla de arriba abajo, deberíamos de introducir la instrucción dentro del bucle, tal y como esta configurado para el cubo.