Sistemas Ubicuos, Empotrados y Móviles

Práctica 1: Laberinto “two players” (Juego)

Máster Universitario en Ingeniería Informática
junio de 2021

Autor
Miguel Cabezas Puerto
70911497J
Óscar Sánchez Juanes
70918894G

El presente documento recoge el informe desarrollado por los alumnos Miguel Cabezas Puerto y Óscar Sánchez Juanes acerca de la primera práctica de la asignatura Sistemas Ubicuos, Empotrados y Móviles en el seno del Máster en Ingeniería Informática de la Universidad de Salamanca en el curso 2020-2021, consistente en el uso de E/S digitales y analógicas, temporizadores, así como el puerto serie para la implementación de un juego de dos jugadores que buscan escapar de un laberinto.

Tabla de contenido

Tabla de ilustraciones

Introducción

El objetivo de la realización de este trabajo es utilizar E/S digitales y analógicas, temporizadores y puerto serie para la creación de un juego en el que dos jugadores utilizarán un *joystick* para llegar al final de un laberinto antes que su oponente en un determinado tiempo, o bien realizar el mayor número de movimientos por el "mapa".

Descripción del trabajo

Componentes hardware

Para lograr implementar las especificaciones del juego, se han utilizado los siguientes componentes hardware:

- Una placa FRDM-KE06Z: Microcontrolador Kinetis E Series de 5V construido sobre el núcleo ARM® Cortex™-M0+ core. En ella se instala el programa que contiene el juego.
- Dos *joysticks*: Utilizados para que cada jugador mueva su "ficha" por el laberinto.
- Un *buzzer*: Su funcionalidad dentro del juego es emitir sonidos de choque y fin de partida.
- Un cable PL2303HXD: Utilizado para la comunicación serie UART con el equipo en el que se visualizará el juego.

Componentes lógicos Kinetis

Antes de detallar los componentes lógicos utilizados durante la práctica, es necesario incluir la configuración realizada para el procesador de la placa (MKE06Z128VLK4).

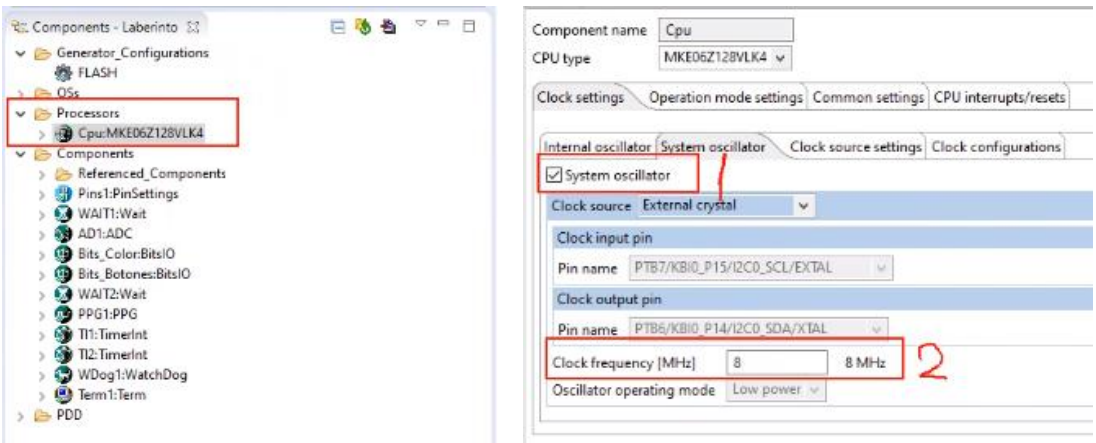

Ilustración 1: Configuración procesador

Tal y como se puede apreciar en la *Ilustración 1*, hay que activar el *checkbox* "*System oscillator*", que se corresponde con el reloj externo. Una vez hecho esto, se establece la frecuencia de dicho reloj a 8MHz.

A continuación, se detallan los componentes lógicos de Kinetis utilizados durante la práctica:

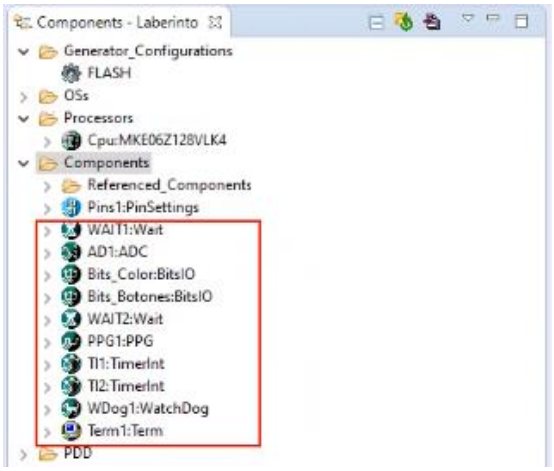

Ilustración 2: Componentes

- Wait: Su función es implementar esperas dentro del programa (a modo de *sleep*).

- ADC: Es un conversor analógico-digital utilizado para la detección del movimiento de los *joysticks*.
  Dentro de su configuración hay que establecer la resolución, el tiempo de conversión y el número de lecturas. Además, es necesario habilitar el servicio de interrupción.

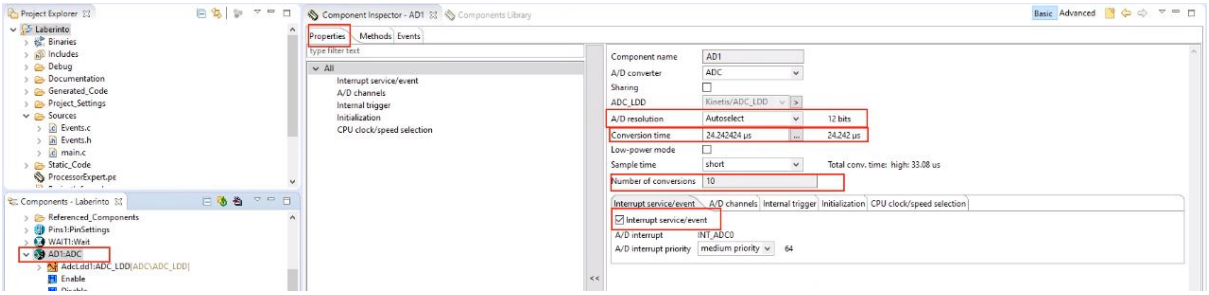

Ilustración 3: Configuración ADC

Aunque haya dos *joysticks*, únicamente es necesario utilizar un componente de este tipo y, para diferenciar entre los *joysticks*, se definen diferentes canales.

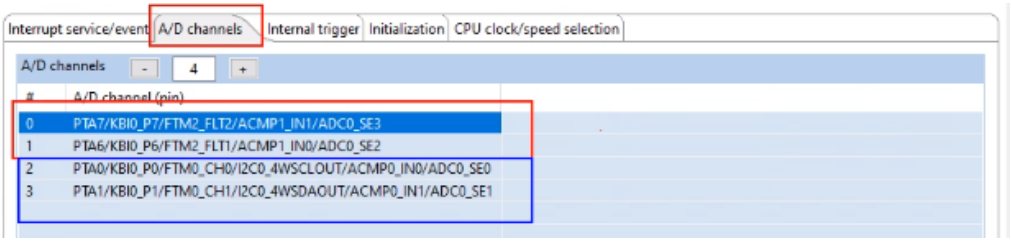

Ilustración 4: Definición de canales ADC

En la *Ilustración 4* se puede apreciar dos grupos de canales: los canales 0 y 1 se corresponden con el *joystick* 1 y los canales 2 y 3 con el *joystick* 2.

- BitsIO – Bits_Color: Utilizado para gestionar los colores del LED de la placa. Para su configuración hay que establecer el puerto, una dirección (salida, concretamente) y un pin.

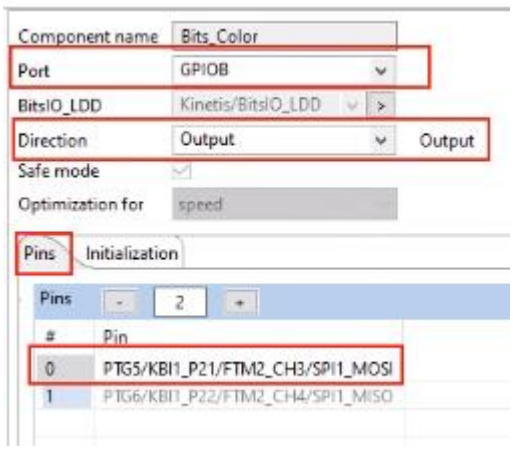

Ilustración 5: Configuración BitsIO – Bits_Color

- BitsIO – Bits_Botones. Utilizado para gestionar la pulsación del botón de comienzo de partida. Al ser del mismo tipo que Bits_Color, hay que configurar los mismos parámetros. En este caso, cabe destacar que la dirección es de entrada.

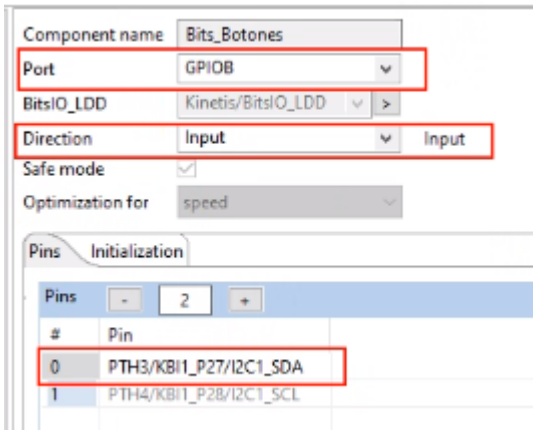

Ilustración 6: Configuración BitsIO - Bits_Botones

- PPG – *Programable Pulse Generator*: Sirve para generar pulso con un determinado periodo y un ancho de pulso. Es utilizado para gestionar el funcionamiento del *buzzer*. Dentro de su configuración hay que establecer el FTM (*FlexTimer Module*) que se va a utilizar, el pin de salida, el periodo en modo intervalo y el ancho de pulso inicial.

Nota: FTM es un temporizador de dos a ocho canales que admite captura de entrada, comparación de salida y generación de señales PWM para controlar aplicaciones de administración de energía y motores eléctricos [1].

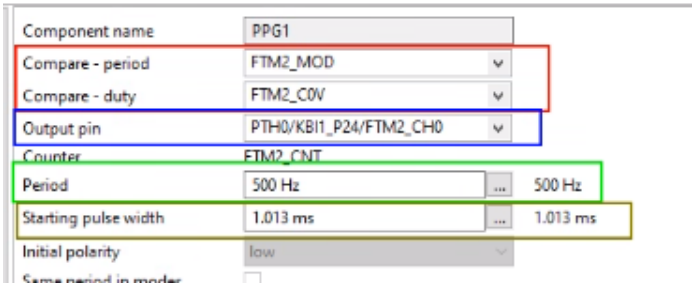

Ilustración 7: Configuración PPG (I)

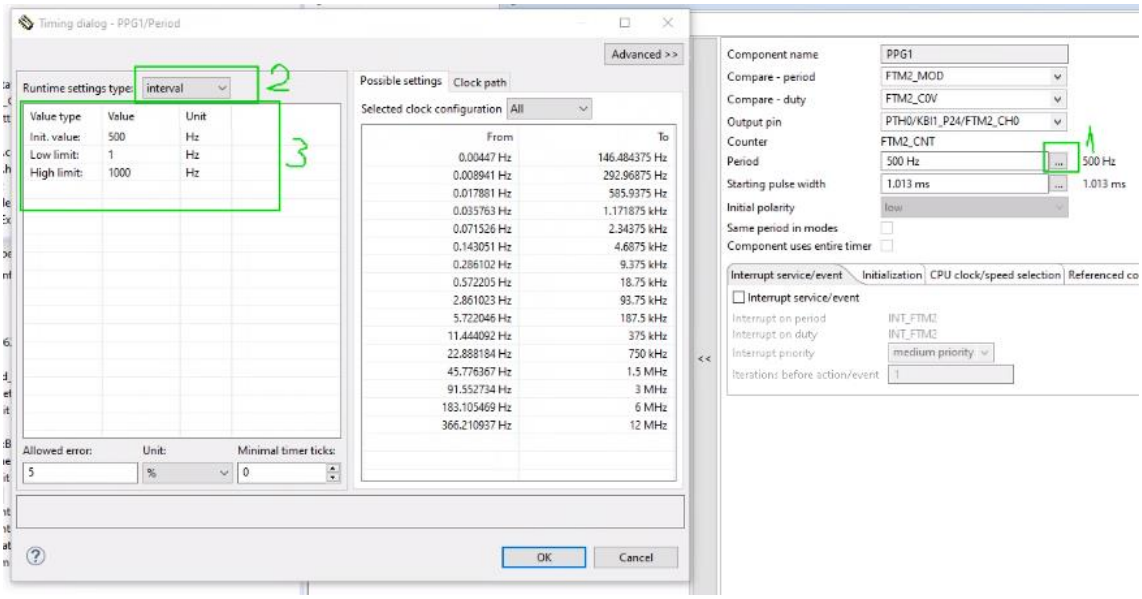

Ilustración 8: Configuración PPG (II)

- *TimerInt*: Es un componente para realizar interrupciones de forma periódica. Se han utilizado dos: uno para actualizar segundo a segundo el tiempo de partida restante y otro para detener el programa una vez finalizado el tiempo establecido.
  La configuración para ambos se realiza de forma análoga. Únicamente hay que configurar el periodo de interrupción.

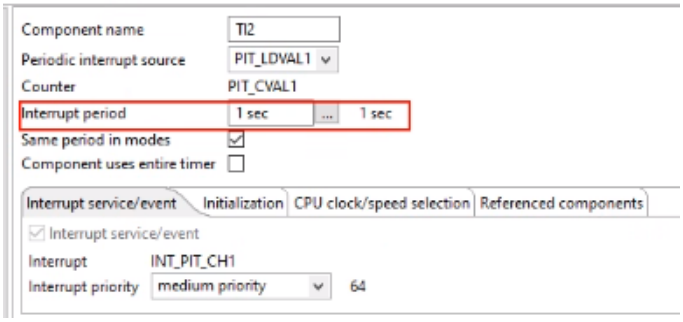

Ilustración 9: Configuración TimerInt

- *WatchDog Timer*: Es un contador que, al llegar a 0, lanza una interrupción y se ejecuta la acción establecida en su configuración. Por tanto, para poder usarlo de manera adecuada hay que determinar tanto el tiempo como la acción a realizar.

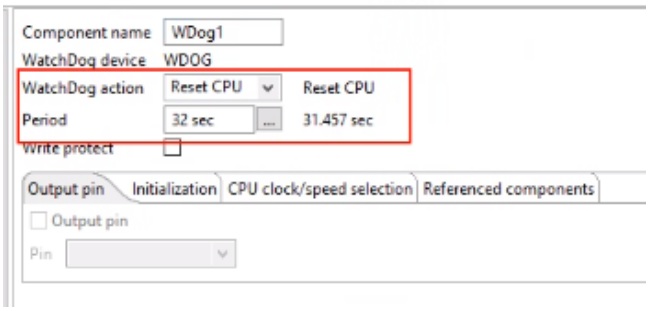

Ilustración 10: Configuración WatchDog Timer

- Term: Componente para la comunicación con el equipo. Consta de un controlador para el SCI/UART, el AsynchroSerial, que permite leer y escribir caracteres de uno en uno o en bloque [2].
  En este caso hay que configurar dicho componente, en lugar del Term como tal.

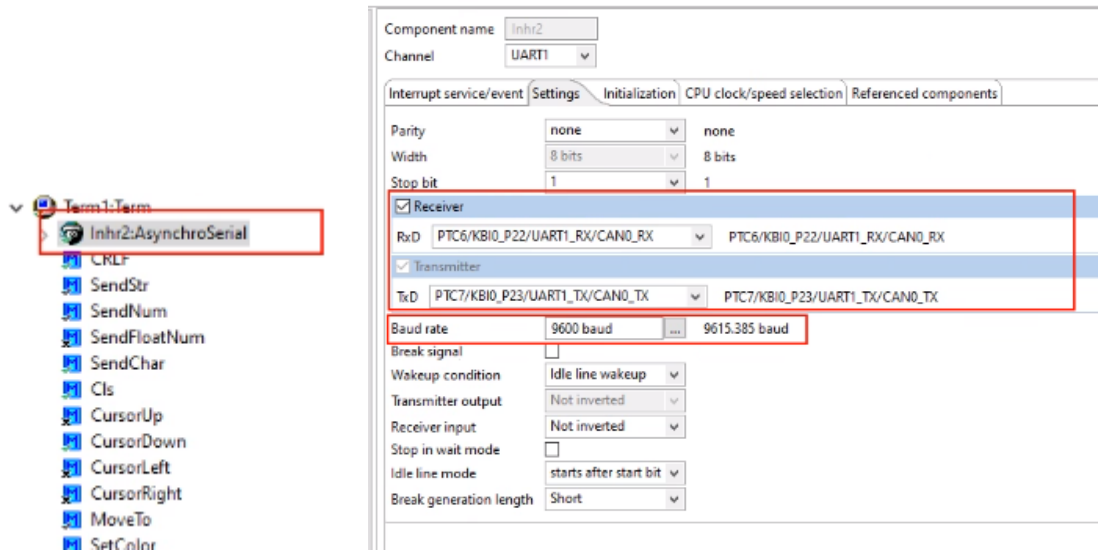

Ilustración 11: Configuración AsynchroSerial

Implementación

A continuación se muestran los aspectos relevantes del código del programa. Dicho programa está compuesto por dos archivos: main.c y Events.c.

Si se desea consultar detalladamente el código, éste se encuentra disponible en el *Anexo 1: Código del programa*.

Main.c

Describe el flujo principal del programa.

Seguidamente, se detallan los aspectos más importantes del mismo:

- Detección de pulsación de botón: Mediante el componente Bits_Botones (con su función *GetVal*) se detecta la pulsación del botón. Cuando esto ocurre, se ilumina el LED de la placa de color verde, usando para ello el componente Bits_Color, a través de su función *PutVal*.

```
222     do {
223         leido = Bits_Botones_GetVal();
224         if (leido == 1) {
225             Bits_Color_PutVal(RGB_GREEN);
226             botonPulsado = 1;
227         }
228         WAIT1_Waitms(500);
229     } while (botonPulsado == 0);
```

Ilustración 12: Detección de pulsación de botón

  Una vez detectado la pulsación del botón, el juego comienza.

- Pintar laberinto: Para dibujar el laberinto en la terminal, se hace uso del componente Term, mediante sus funciones *MoveTo* (movimiento a una coordenada de la pantalla), *SetColor* (establece el color de frente y fondo de la información) y *SendStr* (envía la cadena que se quiere mostrar).

```
134     for (i = 0; i < FILAS; i++) {
135         for (j = 0; j < COLUMNAS; j++) {
136             //printf("%d\t",laberinto[i][j]);
137             Term1_MoveTo(((j + 1) * 2) + 1, ((i + 1) * 2) + 1);
138             if (laberinto[i][j] == 1) {
139                 sprintf(cadena, " ");
140                 Term1_SetColor(clRed, clRed);
141                 Term1_SendStr(cadena); /* writing a string */
142             }
143             if (laberinto[i][j] == 0) {
144                 sprintf(cadena, " ");
145                 Term1_SetColor(clBlack, clBlack);
146                 Term1_SendStr(cadena); /* writing a string */
147             }
148             if (laberinto[i][j] == 2) {
149                 sprintf(cadena, " ");
150                 Term1_SetColor(clGreen, clGreen);
151                 Term1_SendStr(cadena); /* writing a string */
152             }
153             if (laberinto[i][j] == 3) {
154                 sprintf(cadena, " ");
155                 Term1_SetColor(clBlue, clBlue);
156                 Term1_SendStr(cadena); /* writing a string */
157             }
158             if (laberinto[i][j] == 4) {
159                 sprintf(cadena, " ");
160                 Term1_SetColor(clWhite, clWhite);
161                 Term1_SendStr(cadena); /* writing a string */
162             }
163         }
164     }
```

Ilustración 13: Pintar laberinto

- Detectar movimiento del *joystick*: La detección del movimiento se realiza de la misma forma tanto para el *joystick* del jugador 1 como para el del jugador 2, con la diferencia de que para el segundo jugador es necesario indicar el canal, mientras que para el primero se utiliza el canal por defecto.

```
446         AD1_Measure(FALSE);
447
448         //AD1_MeasureChan(FALSE,0);
449         // AD1_MeasureChan(FALSE,1);
450         while (!g_Complete)
451             WAIT1_Waitms(1);
452         g_Complete = FALSE;
453         AD1_GetValue16(Measure);
454         //AD1_GetChanValue16(0,Measure);
455         //AD1_GetChanValue16(1,Measure2);
456
457         //DETECTAR ARRIBA Y ABAJO JUGADOR 2
458         AD1_MeasureChan(FALSE, 2);
459         while (!g_Complete)
460             WAIT1_Waitms(1);
461         g_Complete = FALSE;
462         AD1_GetChanValue16(2, Measure2);
463         //DETECTAR IZQUIERDA Y DERECHA JUGADOR 2
464         AD1_MeasureChan(FALSE, 3);
465         while (!g_Complete)
466             WAIT1_Waitms(1);
467         g_Complete = FALSE;
468         AD1_GetChanValue16(3, Measure3);
469
470         v[0] = ((Measure[0] >> 4) * 3.06 / 4096);//+ ((Measure2[0]>>4)*3.06/4096);
471         v[1] = ((Measure[1] >> 4) * 3.06 / 4096);//+ ((Measure2[1]>>4)*3.06/4096);
472
473         v2[0] = ((Measure2[0] >> 4) * 3.06 / 4096); //+ ((Measure2[0]>>4)*3.06/4096);
474         v2[1] = ((Measure3[0] >> 4) * 3.06 / 4096); //+ ((Measure2[1]>>4)*3.06/4096);
```

Ilustración 14: Detectar movimiento joystick

Para el jugador 1: Mediante la línea 446 se ordena la lectura de los canales por defecto (0 y 1) sin esperar a completarla (espera no bloqueante). Seguidamente, en la línea

450, espera a que la lectura se complete y, una vez completada, línea 453, se obtienen las medidas. Posteriormente (líneas 470 y 471), se convierten las medidas a voltaje.

- Gestionar movimiento del *joystick*: La captura que se muestra a continuación se corresponde con la gestión del movimiento hacia arriba del *joystick* del jugador 1. Cabe destacar que, tanto el movimiento en el resto de direcciones como el movimiento del *joystick* del jugador 2, se gestionan de la misma forma (realizando los cambios necesarios).

```
490         /*JUGADOR 1*/
491
492         if (v[0] != 1.54f && v[1] != 1.47f) {
493             // ARRIBA
494             if (v[0] < 1.54f && v[1] >= 1.44f && v[1] <= 1.90f) {
495                 //printf("ARRIBA");
496                 WAIT1_Waitms(500);
497                 v[0] = 1.54f;
498                 v[1] = 1.47f;
499                 if (laberinto[posX - 1][posY] == 1) {
500                     //printf("CHOQUE!\n");
501                     notone();
502                     playtone(g_TonoT, 1);
503                     notone();
504                 } else if (laberinto[posX - 1][posY] == 2) {
505                     // ENCENDER UN LED DE UN COLOR U OTRO, EN FUNCION DEL JUGADOR QUE GANE,  EN PLACA
506                     printf("PREMIO!\n");
507                     laberinto[posX][posY] = 0;
508                     laberinto[posX2][posY2] = 0;
509                     Term1_MoveTo(((posY + 1) * 2) + 1, ((posX + 1) * 2) + 1);
510                     sprintf(cadena1, " ");
511                     Term1_SetColor(clBlack, clBlack); /* red text on black background */
512                     TI2_Disable();
513                     Term1_MoveTo(((posY2 + 1) * 2) + 1, ((posX2 + 1) * 2) + 1);
514                     sprintf(cadena1, " ");
515                     Term1_SetColor(clBlack, clBlack); /* red text on black background */
516                     notone();
517                     playtone(g_TonoT, 3);
518                     notone();
519                     Bits_Color_PutVal(RGB_RED);
520                     Term1_SetColor(clWhite, clBlack); /* red text on black background */
521                     Term1_CRLF();
522                     Term1_Cls();
523                     Term1_MoveTo(1, 1);
524                     Term1_SendStr("¡Ha ganado el jugador A!"); /* writing a string */
525                     Term1_SetColor(clBlack, clBlack); /* red text on black background */
526                     pulsacionBotonComienzoPartida();
527                     break;
528                 } else if (laberinto[posX - 1][posY] == 4) {
529                     printf("CHOQUE OTRO JUGADOR!\n");
530                     notone();
531                     playtone(g_TonoT, 2);
532                     notone();
533                 } else {
534                     movimientosJ1 += 1;
535                     Term1_SetColor(clWhite, clBlack); /* red text on black background */
536                     Term1_MoveTo(24, 1);
537                     Term1_SendNum(movimientosJ1);
538                     posX--;
539                     laberinto[posXAux][posYAux] = 0;
540                     laberinto[posX][posY] = 3;
541                     //clrscr();
542                     //pintarLaberinto(laberinto);
543                     Term1_MoveTo(((posYAux + 1) * 2) + 1,
544                             ((posXAux + 1) * 2) + 1);
545                     sprintf(cadena1, " ");
546                     Term1_SetColor(clBlack, clBlack); /* red text on black background */
547                     Term1_SendStr(cadena1); /* writing a string */
548                     Term1_SetColor(clBlack, clBlack); /* red text on black background */
549                     Term1_SendStr(cadena1); /* writing a string */
550                     Term1_MoveTo(((posY + 1) * 2) + 1, ((posX + 1) * 2) + 1);
551                     sprintf(cadena1, " ");
552                     Term1_SetColor(clBlue, clBlue); /* red text on black background */
553                     Term1_SendStr(cadena1); /* writing a string */
554                     Term1_MoveTo(1, 1);
555                 }
556
557             }
```

Ilustración 15: Gestión movimiento del joystick

En función del voltaje, medido previamente, se determina si el movimiento es hacia arriba (por prueba y error). Una vez determinado, se comprueba si el movimiento es válido, es decir, la casilla a la que se quiere mover el jugador está libre. En este punto existen cuatro supuestos:

a) Choque con una pared (líneas 499-503): Se emite el sonido asociado al choque con una pared.

b) Alcanzar la meta (líneas 504-527): Se emite el sonido de victoria (componente PPG), se establece el color del LED de la placa a rojo (componente Bits_Color), se pinta de negro toda la pantalla de la consola del equipo y se escribe la victoria del jugador (componente Term) y, finalmente, se vuelve a comenzar la partida.

c) Choque con el otro jugador (líneas 528-532): Se emite el sonido asociado al choque con el otro jugador.

d) Movimiento a casilla libre (líneas 533-554): Se suma uno al contador de movimientos del jugador y se actualizan las casillas de origen y fin del movimiento (de forma visual y en la matriz).

- Reproducir sonido a través del *buzzer*: Para reproducir un sonido se llama a la función playtone() que, en función de los parámetros que se le pasan, envía un pulso con una frecuencia diferente. Es en la función tone() en la que se reproduce el sonido propiamente dicho.
  Finalmente, si se quiere detener el sonido, se llama a la función notone().

```
87  void tone(word frecuencia) {
88      PPG1_Enable();
89      PPG1_SetFreqHz(frecuencia);
90  }
91  void notone() {
92      PPG1_Disable();
93      PPG1_ClrValue();
94  }
95  void playtone(word *ptrT, int tono) {
96      int i = tono;
97      if (ptrT[i] != TONE_UNO) {
98          if (tono == 1) { //CHOQUE PARED
99              tone(ptrT[i]);
100             WAIT1_Waitms(100);
101         } else if (tono == 2) { // CHOQUE CON OTRO JUGADOR
102             tone(ptrT[tono]);
103             WAIT1_Waitms(100);
104         } else { // FIN DE JUEGO
105             tone(ptrT[tono]);
106             WAIT1_Waitms(1000);
107         }
108     } else {
109         notone();
110     }
111 }
```

Ilustración 16: Reproducir sonido buzzer

Events.c

Describe los eventos e interrupciones que ocurren a lo largo del programa.

- AD1_OnEnd(): Indica que la lectura de la detección de *joysticks* se ha completado.

```
77 void AD1_OnEnd(void)
78 {
79   /* Write your code here ... */
80     g_Complete = TRUE;
81     //g_Complete2 = TRUE;
82 }
```

Ilustración 17: AD1_OnEnd()

- TI1_OnInterrupt(): Interrupción que se lanza cuando se acaba el tiempo de partida. En ella, se comprueba el número de movimientos de cada jugador para determinar quién es el ganador (o si hay empate). Además, se cambia el color del LED de la placa a rojo.

```
 98 extern volatile int tiempoJuego;
 99 extern volatile int movimientosJ1,movimientosJ2;
100 void TI1_OnInterrupt(void)
101 {
102     TI2_Disable();
103   //printf("\nTIEMPO AGOTADO!!!\n");
104   int j1,j2;
105   j1=movimientosJ1;
106   j2=movimientosJ2;
107   Bits_Color_PutVal(RGB_RED);
108   if(j1>j2){
109       Term1_SetColor(clWhite, clBlack); /* red text on black background */
110       Term1_CRLF();
111       Term1_Cls();
112       Term1_MoveTo(1,1);
113       Term1_SendStr(";Ha ganado el jugador A!"); /* writing a string */
114   }else if(j1<j2){
115       Term1_SetColor(clWhite, clBlack); /* red text on black background */
116       Term1_CRLF();
117       Term1_Cls();
118       Term1_MoveTo(1,1);
119       Term1_SendStr(";Ha ganado el jugador B!"); /* writing a string */
120   }else{
121       Term1_SetColor(clWhite, clBlack); /* red text on black background */
122       Term1_CRLF();
123       Term1_Cls();
124       Term1_MoveTo(1,1);
125       Term1_SendStr(";Empate!"); /* writing a string */
126   }
127   Term1_SetColor(clBlack, clBlack);
128   Term1_MoveTo(1,2);
129   Term1_SendStr("    ");
130   Term1_SetColor(clBlack, clBlack);
131    Term1_MoveTo(1,3);
132    Term1_SendStr("    ");
133   Term1_SetColor(clBlack, clBlack);
134   tiempoJuego=30;
135   //return 0;
136  // pulsacionBotonComienzoPartida();
137 }
```

Ilustración 18: TI1_OnInterrupt()

- TI2_OnInterrupt(): Actualiza el contador descendente que muestra el tiempo restante de partida. El valor correspondiente lo va mostrando por pantalla.

```
153 extern volatile bool empiezaPintadoTiempo;
154 extern volatile bool pintarJugadores;
155 void TI2_OnInterrupt(void)
156 {
157   int i;
158   tiempoJuego--;
159 
160     for(i=8;i<15;i++){
161         Term1_SetColor(clBlack, clBlack);
162         Term1_MoveTo(1,i);
163         Term1_SendStr(" ");
164     }
165   if(empiezaPintadoTiempo){
166       pintarJugadores=FALSE;
167       Term1_SetColor(clWhite, clBlack);
168         Term1_MoveTo(8,1);
169         Term1_SendNum(tiempoJuego);
170          Term1_SetColor(clWhite, clBlack);
171          Term1_MoveTo(8,1);
172          Term1_SendNum(tiempoJuego);
173   }
174   pintarJugadores=TRUE;
```

Ilustración 19: TI2_OnInterrupt()

Bibliografía

[1] M. Plachy, «Configuring the FlexTimer for Position and Speed Measurement with an Encoder,» Diciembre 2011. [En línea]. Available: https://os.mbed.com/media/uploads/GregC/an4381-flextimer_configuration.pdf.

[2] E. Styger, «Tutorial: Using a Terminal Input and Output; *without* printf() and scanf(),» 16 Noviembre 2013. [En línea]. Available: https://mcuoneclipse.com/2013/11/16/tutorial-using-a-terminal-input-and-output-without-printf-and-scanf/.