

Electrónica Digital III

Trabajo final: SIMON GAME

CASABELLA MARTIN, 39.694.763

ZIMMEL EZEQUIEL JOSÉ, 33.382.573

GRUPO 12

I. Juego de memoria SIMON

SIMON es un juego electrónico que desafía la memoria. El usuario básicamente tiene que repetir la secuencia mostrada en el tablero para progresar al siguiente nivel. En la presente implementación del juego, el usuario puede elegir entre 3 modos de juego:

1. Modo Progresivo o Clásico, donde el juego reutiliza el nivel anterior como inicio del siguiente nivel antes de añadir un paso aleatorio al final;
2. Modo Inverso: se debe proceder a realizar la secuencia generada y mostrada pero al revés
3. Modo Aleatorio, donde cada nivel del juego produce una nueva secuencia aleatoria, cada paso más largo que el anterior;
4. Modo Multijugador, donde cada usuario añade una nota a la secuencia, siendo el primero en equivocarse el perdedor.

A su vez, como mando de juego, el usuario puede optar por manejarse con botones o como alternativa, una palanca de mando o *joystick*.

II. Requerimientos

- Selección de modo de juego.
- Selección de dificultad.
- Selección de cantidad de notas a tocar.
- Generación aleatoria de secuencia.
- Activación sonora y visual de la nota.
- Activación de las notas a través del movimiento de un giróscopo.
- Mantener estado de la secuencia de juego.
- Despliegue de estado del juego por pantalla LCD.
- Comunicación serial para modo multijugador.

III. Especificaciones

- Uso de microcontrolador LPC1769
- Codificación en C.
- Ingreso de valores mediante pulsadores.
- Aviso de nota tocada mediante sonido (Buzzer piezoeléctrico) e iluminación (LED).
- Utilización de LCD 16x2 '1602A'.
- Empleo de joystick, como mando alternativo.
- Comunicación serie mediante pines específicos de la placa de desarrollo.

IV. Esquemático del diseño elaborado

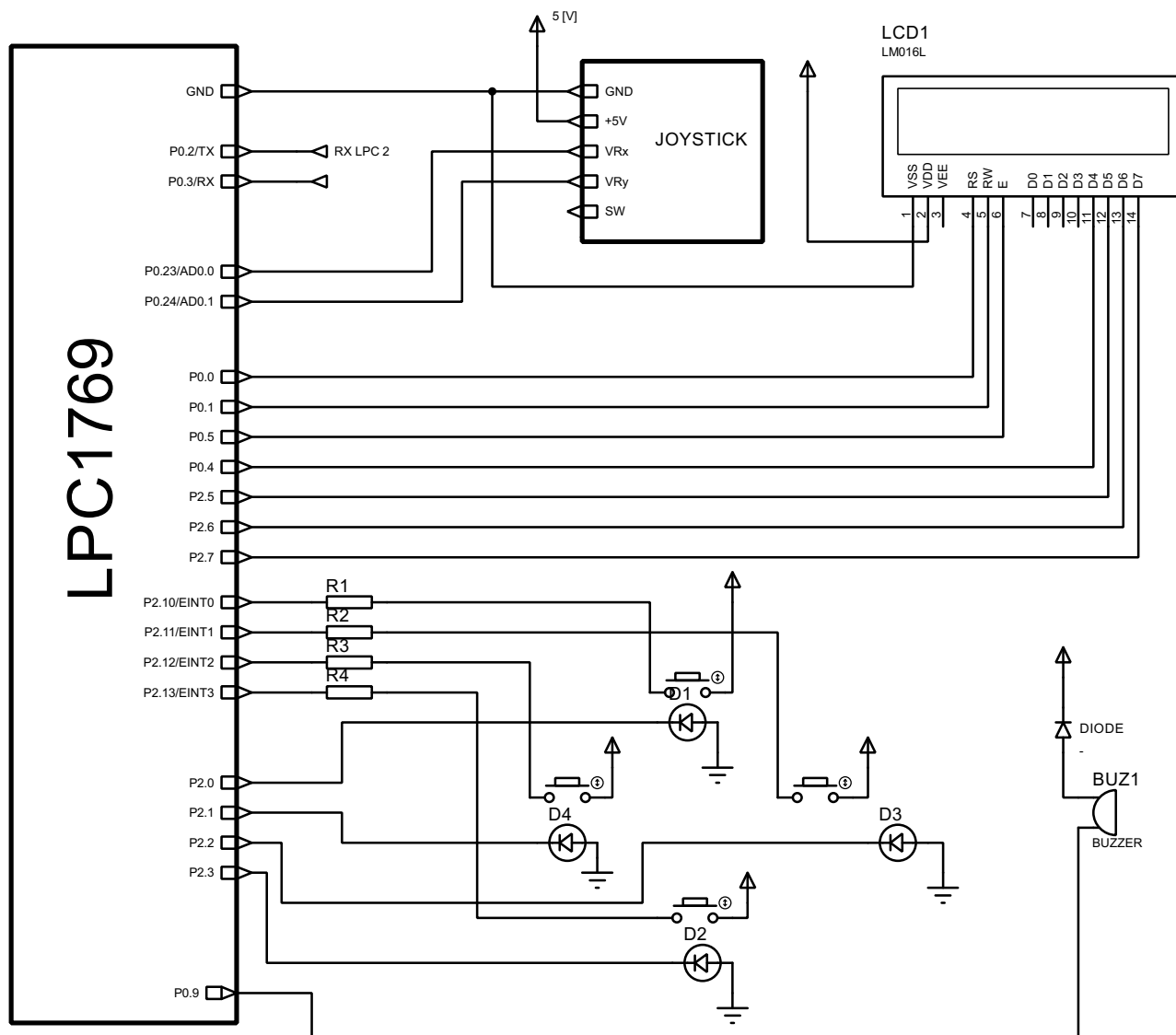


Figure 1: Diagrama circuital

Este esquema se replica en la otra placa utilizada para probar el modo multijugador.

Se ve en el diagrama, que el transmisor del modulo UART, se conecta al receptor de la otra placa, y el receptor al transmisor, para efectuar la comunicacion.

I. Asignación de pines en detalle

Periferico	Pin del periferico	Pin de la placa asignado
LCD	RS	P0.0
	RW	P0.1
	EN	P0.5
	D4	P0.4
	D5	P2.5
	D6	P2.6
	D7	P2.7
Joystick	VRx	P0.23
	VRy	P0.24

Table 1: Asignación de pines para el Joystick y el LCD

Periferico	Pin de la placa asignado
Pulsadores	P2.10/EINT0
	P2.11/EINT1
	P2.12/EINT2
	P2.13/ EINT3
Buzzer	P0.9
LEDS	P2.0
	P2.1
	P2.2
	P2.3
UART	P0.2
	P0.3

Table 2: Asignación de pines para pulsadores, leds, y modulo UART

V. Código empleado

Se elaboraron los siguientes archivos (código fuente en C) y headers:

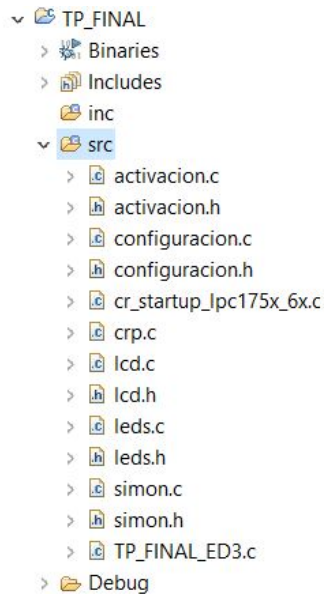


Figure 2: Archivos : códigos fuentes y headers

I. TP_FINAL_ED3.c

Programa principal: arranca el juego mostrando en el LCD al usuario que modos de juego puede elegir, y en función al ingreso del usuario, ejecuta las funciones de un modo u otro.

Al finalizar, le indica al usuario si gana o perdió a través del LCD y una secuencia de luces indicativas. Luego vuelve a mostrar las opciones de modos de juego para que el usuario pueda volver a jugar si así lo desea.

Incluye los siguientes archivos headers:

- *configuracion.h*
- *leds.h*
- *simon.h*
- *lcd.h*

```
1 #ifndef __USE_CMSIS
2 #include "LPC17xx.h"
3 #endif
4
5 #include <cr_section_macros.h>
6
7 #include <stdio.h>
8 #include "configuracion.h"
9 #include "leds.h"
10 #include "simon.h"
11 #include "lcd.h"
12
13 int
14 main (void)
15 {
16     int juego = 0, nivel = 0, exito = 0;
17
18     clearPulsador ();
19     clear_SELMODE ();
20     perifericos_Init ();
21     leds_Init ();
22     Lcd_Init (LCD_RS, LCD_RW, LCD_EN, LCD_D4, LCD_D5, LCD_D6, LCD_D7);
23     Lcd_Show ("SIMON", 1, 7, 0, 0);
24     Lcd_Show ("—EDIII—", 2, 5, 0, 0);
25     simon_Init (1000000);
26
27     while (1)
28     {
29         Lcd_Show ("1) 4", 1, 1, 1, 0);
30         Lcd_Show ("2) 8", 1, 10, 0, 0);
31         Lcd_Show ("3) 12", 2, 1, 0, 0);
32         Lcd_Show ("4) 16", 2, 10, 0, 0);
33         while (!nivel)
34         {
35             nivel = getPulsador ();
36             LPC_SC->EXTINT |= ~0;
37             retardo (500000);
38         }
39         clearPulsador ();
40         clear_SELMODE ();
41         Lcd_Show ("1)Clasi", 1, 1, 1, 0);
42         Lcd_Show ("2)Inver", 1, 10, 0, 0);
43         Lcd_Show ("3)Avanz", 2, 1, 0, 0);
44         Lcd_Show ("4)Multi", 2, 10, 0, 0);
45         juego = 0;
46         while (!juego)
47         {
48             juego = getPulsador ();
49             LPC_SC->EXTINT |= ~0;
50             retardo (500000);
51         }
52
53         switch (juego)
54         {
55             case (1):
56                 Lcd_Show ("MOD0 CLASICO", 1, 4, 1, 2500000);
57                 Lcd_Show ("3", 2, 9, 0, 2500000);
58                 Lcd_Show ("2", 2, 9, 0, 2500000);
59                 Lcd_Show ("1", 2, 9, 0, 2500000);
```

```

60  Lcd_Show ("A JUGAR!", 1, 6, 1, 5000000);
61  modo_1 (nivel, &exito);
62  break;
63  case (2):
64      Lcd_Show ("MODO INVERSO", 1, 4, 1, 2500000);
65      Lcd_Show ("3", 2, 9, 0, 2500000);
66      Lcd_Show ("2", 2, 9, 0, 2500000);
67      Lcd_Show ("1", 2, 9, 0, 2500000);
68      Lcd_Show ("A JUGAR!", 1, 6, 1, 5000000);
69      modo_2 (nivel, &exito);
70      break;
71  case (3):
72      printf("Modo avanzado JOYSTICK: \n");
73      Lcd_Show ("MODO AVANZADO", 1, 3, 1, 2500000);
74      Lcd_Show ("3", 2, 9, 0, 2500000);
75      Lcd_Show ("2", 2, 9, 0, 2500000);
76      Lcd_Show ("1", 2, 9, 0, 2500000);
77      Lcd_Show ("A JUGAR!", 1, 6, 1, 5000000);
78      modo_3 (nivel, &exito);
79      break;
80  case (4):
81      Lcd_Show ("MULTIJUGADOR", 1, 4, 1, 2500000);
82      Lcd_Show ("3", 2, 9, 0, 2500000);
83      Lcd_Show ("2", 2, 9, 0, 2500000);
84      Lcd_Show ("1", 2, 9, 0, 2500000);
85      printf("\n Entre en modo 4 \n");
86      modo_4 (nivel, &exito);
87      break;
88  }
89
90      if (juego != 4)
91      {
92          if (exito)
93          {
94              Lcd_Show ("GANASTE!", 1, 5, 1, 0);
95              simon_SUCCESS ();
96          }
97          else
98          {
99              Lcd_Show ("PERDISTE!", 1, 5, 1, 0);
100              simon_FAIL ();
101          }
102      }
103      clear_SELMODE ();
104      nivel = 0;
105      juego = 0;
106      clearPulsador ();
107
108      Lcd_Show ("A JUGAR!", 1, 6, 1, 0);
109      simon_Init (500000);
110
111  }
112
113 }

```

II. configuracion.c

Este archivo es el encargado de configurar cada periférico utilizado (ADC, timer, systick, UART), junto con las interrupciones y el puerto de propósito general.

Incluye solamente el header *configuracion.h*. Se anexan las funciones empleadas:

```

1 #include "configuracion.h"
2
3 uint32_t volatile * const STCTRL = (uint32_t *) AddrSTCTRL;
4 uint32_t volatile * const STRELOAD = (uint32_t *) AddrSTRELOAD;
5
6 uint32_t RELOAD[4] =
7 { 103370, 167404, 254545, 356759 }; //{Amarillo, Verde, Rojo, Azul}
8
9 /**
10  * @brief Configuracion de los pines a ser utilizados
11  */
12 void
13 gpioConfig (void)
14 {
15     LPC_PINCON->PINSEL0 |= (1 << 4); //Funcion del Pin0.2 TX
16     LPC_PINCON->PINSEL0 |= (1 << 6); //Funcion del Pin0.3 RX
17     LPC_PINCON->PINSEL1 |= (1 << 14); //Funcion del Pin0.23 AD0.0
18     LPC_PINCON->PINSEL1 |= (1 << 16); //Funcion del Pin0.24 AD0.1
19     LPC_PINCON->PINSEL1 |= (1 << 18); //Funcion del Pin0.25 AD0.2
20     LPC_PINCON->PINSEL4 |= (0b00 << 0); //Funcion del Pin2.0 ESTA COMO GPIO, para PWM1
21     .1 poner 0b10
22     LPC_PINCON->PINSEL4 |= (0b00 << 2); //Funcion del Pin2.1 ESTA COMO GPIO, para PWM1
23     .2 poner 0b10
24     LPC_PINCON->PINSEL4 |= (0b00 << 4); //Funcion del Pin2.2 ESTA COMO GPIO
25     , para PWM1.3 poner 0b10
26     LPC_PINCON->PINSEL4 |= (0b00 << 6); //Funcion del Pin2.3 ESTA COMO GPIO,
27     para PWM1.4 poner 0b10
28     LPC_PINCON->PINSEL4 |= (1 << 20); //Funcion del Pin2.10: EINT0
29     LPC_PINCON->PINSEL4 |= (1 << 22); //Funcion del Pin2.11: EINT1
30     LPC_PINCON->PINSEL4 |= (1 << 24); //Funcion del Pin2.12: EINT2
31     LPC_PINCON->PINSEL4 |= (1 << 26); //Funcion del Pin2.12: EINT3
32     LPC_PINCON->PINMODE1 |= (1 << 15); //Pin0.23 entrada analogica
33     LPC_PINCON->PINMODE1 |= (1 << 17); //Pin0.24 entrada analogica
34     LPC_PINCON->PINMODE1 |= (1 << 19); //Pin0.25 entrada analogica
35     LPC_PINCON->PINMODE4 |= (0b11 << 20); //Pin2.10 Pull-down
36     LPC_PINCON->PINMODE4 |= (0b11 << 22); //Pin2.11 Pull-down
37     LPC_PINCON->PINMODE4 |= (0b11 << 24); //Pin2.12 Pull-down
38     LPC_PINCON->PINMODE4 |= (0b11 << 26); //Pin2.12 Pull-down
39     //LPC_GPIO2->FIOMASK2 |= ();
40     LPC_GPIO0->FIODIR |= (1 << 9); //Pin0.9 como salida
41     LPC_GPIO2->FIODIR |= (1 << 0 | 1 << 1 | 1 << 2 | 1 << 3); //Pines 2.0/2.1/2.2/2.3 como
42     salida
43     LPC_GPIO0->FIOMASK = 0;
44     LPC_GPIO0->FIOCLR |= (1 << 9); //Bajamos salida del tono
45 }
46
47 /**
48  * @brief Configura los TIMERS
49  */
50 void
51 timerConfig (void)
52 {

```



```

48 //-----TIMER0-----
49 LPC_TIM0->MR0 = 6250000; //Match para interrupcion cada 625ms
50 LPC_TIM0->MCR |= 0b11; //Match0 interrupcion y reinicio del TC
51 LPC_TIM0->TCR &= ~1; //Deshabilitamos el contador
52 LPC_TIM0->TCR |= (1 << 1); //Reseteamos el TC y PC
53 LPC_TIM0->TCR &= ~(1 << 1); //Salimos del estado Reset
54 LPC_TIM0->IR |= ~0; //Bajamos banderas de interrupcion pendientes
55
56 //-----TIMER1-----
57 LPC_TIM1->MR0 = 10000000; //Match para interrupcion cada 1s
58 LPC_TIM1->MCR |= 0b11; //Match0 interrupcion y reinicio del TC
59 LPC_TIM1->TCR &= ~1; //Deshabilitamos el contador
60 LPC_TIM1->TCR |= (1 << 1); //Reseteamos el TC y PC
61 LPC_TIM1->TCR &= ~(1 << 1); //Salimos del estado Reset
62 LPC_TIM1->IR |= ~0; //Bajamos banderas de interrupcion pendientes
63
64 }
65
66 /**
67 * @brief Configura las interrupciones Externas
68 */
69 void
70 eintConfig (void)
71 {
72 //-----EINT0-----
73 LPC_SC->EXTMODE |= 1; //EINT0 sensible por flanco
74 LPC_SC->EXTPOLAR |= 1; //Flanco ascendente
75 //-----EINT1-----
76 LPC_SC->EXTMODE |= (1 << 1); //EINT1 sensible por flanco
77 LPC_SC->EXTPOLAR |= (1 << 1); //Flanco ascendente
78 //-----EINT2-----
79 LPC_SC->EXTMODE |= (1 << 2); //EINT2 sensible por flanco
80 LPC_SC->EXTPOLAR |= (1 << 2); //Flanco ascendente
81 //-----EINT3-----
82 LPC_SC->EXTMODE |= (1 << 3); //EINT3 sensible por flanco
83 LPC_SC->EXTPOLAR |= (1 << 3); //Flanco ascendente
84 }
85
86 /**
87 * @brief Configura la comunicacion Serie UART
88 */
89 void
90 uartConfig (void)
91 {
92 LPC_SC -> PCONP |= 1<<3; //Encendemos periferico
93 LPC_UART0->LCR |= (1 << 7); //On DLAB
94 LPC_UART0->DLL = 130; //PCLK 20Mhz -> 9600BR DLL en 130, DLM en 0
95 LPC_UART0->DLM = 0;
96 LPC_UART0->LCR &= ~(1 << 7); //OFF DLAB, para poder seguir cfg
97 //LPC_UART0->FCR |= 1; // 1 Con fifo, 0 (defecto) Sin fifo
98 LPC_UART0->LCR |= (0b11); //8bits
99 LPC_UART0->IER |= 1; //Interrupcion por recepcion de datos
100 }
101
102
103 /**
104 * @brief Configura el ADC para el uso del STICK
105 */
106 void
107 adcConfig (void)
108 {

```

```

109 LPC_ADC->ADCR |= 0b11;    //Canales a muestrear AD0.0, AD0.1
110 //LPC_ADC->ADCR |= (0b0 << 15); //Divisor para el clock de periferico
111 LPC_ADC->ADCR &= ~(1 << 16); //OFF Burst
112 LPC_ADC->ADCR |= (1 << 21); //ADC operacional
113 LPC_ADC->ADINTEN &= ~(1 << 8); //Deshabilitamos interrupcion por ADC
114 }
115
116
117 /**
118  * @brief Realiza la Habilitacion de los perifericos, la configuracion de su
119  * frecuencia y luego habilita las interrupciones de los perifericos
120  */
121 void
122 nvic_sysConfig (uint8_t i)
123 {
124     if (i == 0)
125     {
126         *STCTRL &= ~(1 | (1 << 1)); //Deshabilitos cuenta e interrupcion de SysTick
127         LPC_SC->PCONP |= (1 << 12); //Activamos periferico ADC
128
129         LPC_SC->PCLKSEL0 |= (0b11 << 24); //ADC clock = 1/8 CPU clock
130         LPC_SC->PCLKSEL0 |= (0b11 << 2); //TIMER0 clock = 1/8 CPU clock
131         LPC_SC->PCLKSEL0 |= (0b11 << 4); //TIMER1 clock = 1/8 CPU clock
132         LPC_SC->PCLKSEL0 |= (0b11 << 12); //PWM clock = 1/8 CPU clock
133         *STCTRL |= (1 << 2);
134         *STRELOAD = RELOAD[0] / 2;
135         i = 1;
136     }
137     else
138     {
139         LPC_SC->EXTINT |= ~0; //Bajo banderas de interrupcion EINT
140         //Habilitamos las interrupciones
141         NVIC_EnableIRQ (EINT0_IRQn);
142         NVIC_EnableIRQ (EINT1_IRQn);
143         NVIC_EnableIRQ (EINT2_IRQn);
144         NVIC_EnableIRQ (EINT3_IRQn);
145         NVIC_EnableIRQ (ADC_IRQn);
146         NVIC_EnableIRQ (TIMER0_IRQn);
147         NVIC_EnableIRQ (TIMER1_IRQn);
148
149         LPC_ADC->ADCR |= (1 << 16); //Conversion Burst ON
150         // *STCTRL |= (1 << 0) | (1 << 1); //Cuenta e interrupcion de SysTick habilitadas
151         i = 0;
152     }
153 }
154
155 /**
156  * Ajusta la carga del SysTick para generar un tono en particular
157  * y habilita su cuenta e interrupcion.
158  */
159 void
160 setTono (uint32_t freq)
161 {
162     {
163         *STRELOAD = freq;
164         *STCTRL |= (1 << 0) | (1 << 1);
165     }
166
167 /**
168  * Deshabilita la cuenta e interrupcion del SysTick.
169  */

```

```
170
171 void
172 sysTick_OFF (void)
173 {
174     *STCTRL &= ~(1 | (1 << 1));
175 }
176
177
178 /*
179 * Setea el tiempo de interrupcion del timer, con el valor pasado como parametro
180 */
181 void
182 setTiempo (uint8_t timer, uint32_t tiempo)
183 {
184     switch (timer)
185     {
186         case (0):
187             LPC_TIM0->MR0 = tiempo;
188             break;
189         case (1):
190             LPC_TIM1->MR0 = tiempo;
191             break;
192     }
193 }
194
195 /*Inicializa perifericos en uso, y habilita por ultimo las interrupciones utilizadas*/
196 void
197 perifericos_Init (void)
198 {
199     nvic_sysConfig (0);
200     gpioConfig ();
201     adcConfig ();
202     timerConfig ();
203     //pwmConfig ();
204     eintConfig ();
205     nvic_sysConfig (1);
206 }
```

III. activacion.c

Este archivo alberga los handlers de las interrupciones configuradas, junto con algunas funciones adicionales utilizadas en diversas etapas (envio por UART, obtencion del boton pulsado, arrancar juego, generacion de diferentes tonos, entre otras).

Incluye los siguientes archivos headers:

- *activacion.h*
- *leds.h*
- *simon.h*

```

1 #include "activacion.h"
2 #include "leds.h"
3 #include "simon.h"
4 #include <stdio.h>
5
6 #define TONO_A 51685
7 #define TONO_B 83702
8 #define TONO_C 127272
9 #define TONO_D 178379
10
11 uint8_t SEL_MODE = 0;
12 uint8_t PULSADOR = 0;
13 uint8_t SECUENCIA = 1;
14 uint8_t CONEXION = 1;
15 uint8_t ENABLE = 0;
16 /*Para indicar si ya se inicio comunicacion*/
17 uint8_t START_COM = 0;
18 /*Para indicar si recibí tono del otro jugador*/
19 uint8_t RCV_TONO = 0;
20 /*Para indicar si esta en curso el juego*/
21 uint8_t GAME_ON = 0;
22
23 char byte_to_send = 0;
24 char read_byte = 0;
25
26 char *buffer_recep;
27
28 /*
29  * Empleado para seleccion del nivel de dificultad y el modo
30  * de juego. Lanza el inicio del tono para el LED Verde y
31  * activa el TIMER0 para la duracion del Tono.
32  */
33
34 void
35 EINT0_IRQHandler (void)
36 {
37     NVIC_DisableIRQ (EINT0_IRQn);
38     while (debounce (LPC_GPIO2->FIOPIN >> 10 & 1))
39     {
40         retardo (100000);
41     }
42
43     LPC_SC->EXTINT |= 1; //Bajo banderas de EINT0
44     NVIC_EnableIRQ (EINT0_IRQn);
45     if (SEL_MODE == 0)
46     {
47         PULSADOR = 1;
48         SEL_MODE = 1;
49         return;
50     }
51     if (SEL_MODE == 1)
52     {
53         PULSADOR = 1;
54         SEL_MODE = 2;
55         return;
56     }
57     if (SEL_MODE == 10)
58     {
59         NVIC_DisableIRQ (EINT1_IRQn);

```

```

60     NVIC_DisableIRQ (EINT2_IRQn);
61     NVIC_DisableIRQ (EINT3_IRQn);
62     setTono (TONO_D);
63     LPC_TIM0->TCR |= 1;      //Habilito cuenta del TIMER0
64     //LPC_GPIO2->FIOSET |= (1 << 0);
65 }
66 }
67
68
69 /*
70 * Empleado para seleccion del nivel de dificultad y el modo
71 * de juego. Lanza el inicio del tono para el LED Rojo y
72 * activa el TIMER0 para la duracion del Tono.
73 */
74
75 void
76 EINT1_IRQHandler (void)
77 {
78     NVIC_DisableIRQ (EINT1_IRQn);
79     while (debounce (LPC_GPIO2->FIOPIN >> 11 & 1))
80     {
81         retardo (100000);
82     }
83
84     NVIC_EnableIRQ (EINT1_IRQn);
85     LPC_SC->EXTINT |= (1 << 1);      //Bajo banderas de EINT1
86     if (SEL_MODE == 0)
87     {
88         PULSADOR = 2;
89         SEL_MODE = 1;
90         return;
91     }
92     if (SEL_MODE == 1)
93     {
94         PULSADOR = 2;
95         SEL_MODE = 2;
96         return;
97     }
98
99     if (SEL_MODE == 10)
100    {
101        NVIC_DisableIRQ (EINT0_IRQn);
102        NVIC_DisableIRQ (EINT2_IRQn);
103        NVIC_DisableIRQ (EINT3_IRQn);
104        setTono (TONO_C);
105        LPC_TIM0->TCR |= 1;      //Habilito cuenta del TIMER0
106        //LPC_GPIO2->FIOSET |= (1 << 1);
107    }
108 }
109
110
111 /*
112 * Empleado para seleccion del nivel de dificultad y el modo
113 * de juego. Lanza el inicio del tono para el LED Azul y
114 * activa el TIMER0 para la duracion del Tono.
115 */
116
117 void
118 EINT2_IRQHandler (void)
119 {
120     NVIC_DisableIRQ (EINT2_IRQn);

```

```

121 while (debounce (LPC_GPIO2->FIOPIN >> 12 & 1))
122 {
123     retardo (100000);
124 }
125
126 NVIC_EnableIRQ (EINT2_IRQn);
127 LPC_SC->EXTINT |= (1 << 2);    //Bajo banderas de EINT2
128 if (SEL_MODE == 0)
129 {
130     PULSADOR = 3;
131     SEL_MODE = 1;
132     return;
133 }
134 if (SEL_MODE == 1)
135 {
136     PULSADOR = 3;
137     SEL_MODE = 2;
138     return;
139 }
140
141 if (SEL_MODE == 10)
142 {
143     NVIC_DisableIRQ (EINT0_IRQn);
144     NVIC_DisableIRQ (EINT1_IRQn);
145     NVIC_DisableIRQ (EINT3_IRQn);
146     setTono (TONO_B);
147     LPC_TIM0->TCR |= 1;    //Habilito cuenta del TIMER0
148     //LPC_GPIO2->FIOSET |= (1 << 2);
149 }
150 }
151
152 /*
153  * Empleado para seleccion del nivel de dificultad y el modo
154  * de juego. Lanza el inicio del tono para el LED Amarillo y
155  * activa el TIMER0 para la duracion del Tono.
156  */
157
158 void
159 EINT3_IRQHandler (void)
160 {
161     NVIC_DisableIRQ (EINT3_IRQn);
162     while (debounce (LPC_GPIO2->FIOPIN >> 13 & 1))
163     {
164         retardo (100000);
165     }
166
167     NVIC_EnableIRQ (EINT3_IRQn);
168     LPC_SC->EXTINT |= (1 << 3);    //Bajo banderas de EINT3
169     if (SEL_MODE == 0)
170     {
171         PULSADOR = 4;
172         SEL_MODE = 1;
173         return;
174     }
175     if (SEL_MODE == 1)
176     {
177         PULSADOR = 4;
178         SEL_MODE = 2;
179         return;
180     }
181     if (SEL_MODE == 10)

```

```

182     {
183         NVIC_DisableIRQ (EINT0_IRQn);
184         NVIC_DisableIRQ (EINT1_IRQn);
185         NVIC_DisableIRQ (EINT2_IRQn);
186         setTono (TONO_A);
187         LPC_TIM0->TCR |= 1;      //Habilito cuenta del TIMER0
188         //LPC_GPIO2->FIOSET |= (1 << 3);
189     }
190 }
191
192 /**
193  * Cuando se cumple el tiempo del Tono, pone el bajo la salida y apaga
194  * los Leds. Desactiva el SysTick y el contador del TIMER0.
195  *
196  */
197 void
198 TIMER0_IRQHandler (void)
199 {
200     sysTick_OFF ();
201     LPC_TIM0->IR |= 1;      //Bajo bandera de interrupcion por match0
202     LPC_GPIO0->FIOSET |= 1; //Bajamos salida del tono
203     LPC_TIM0->TCR &= ~1;    //Deshabilitamos el contador
204     LPC_TIM0->TCR |= (1 << 1); //Reseteamos el TC y PC
205     LPC_TIM0->TCR &= ~(1 << 1); //Salimos del estado Reset
206     LPC_TIM0->IR |= ~0;
207     LPC_GPIO2->FIOCLR |= (1 << LED_Ve | 1 << LED_Ro | 1 << LED_Az | 1 << LED_Am);
208     NVIC_EnableIRQ (EINT0_IRQn);
209     NVIC_EnableIRQ (EINT1_IRQn);
210     NVIC_EnableIRQ (EINT2_IRQn);
211     NVIC_EnableIRQ (EINT3_IRQn);
212     SECUENCIA = 0;
213 }
214
215 /**
216  * Se emplea para controlar el tiempo de espera para conexion multijugador (1s).
217  * Desactiva el contador del TIMER1.
218  *
219  */
220
221 void
222 TIMER1_IRQHandler (void)
223 {
224     LPC_TIM1->IR |= 1;      //Bajo bandera de interrupcion por match0
225     LPC_TIM1->TCR &= ~1;    //Deshabilitamos el contador
226     LPC_TIM1->TCR |= (1 << 1); //Reseteamos el TC y PC
227     LPC_TIM1->TCR &= ~(1 << 1); //Salimos del estado Reset
228     LPC_TIM1->IR |= ~0;
229     CONEXION = 0;
230 }
231
232 /*
233  * Para comunicacion en modo multijugador. Un flag se utiliza para el inicio de la
234  * comunicacion, y otro para datos validos.
235  * Si un usuario pierde, el codigo 0xFF es enviado asi se corta la sesion multijugador
236  * iniciada ya que finalizo la misma
237  */
238 void
239 UART0_IRQHandler (void)
240 {

```

```

241  if (START_COM == 0 ){
242
243      read_byte = LPC_UART0->RBR;
244      //printf("buffer: %d",read_byte);
245      //printf("\n");
246
247      //El usuario servidor manda 1s por broadcast, estando a la a la espera de una conexion
248      if( read_byte == 1 ){
249          /*Recibi un 1? arranco/manifiesto mi intencion de conectarme enviando un 2, para
250          arrancar a jugar*/
251          printf("Inicio de comunicacion multijugador \n");
252          byte_to_send = 2;
253          while( ( (LPC_UART0->LSR >> 5) & 1 ) ==0 );
254          LPC_UART0 -> THR = byte_to_send;
255          /*Enable para el juego en modo multijugador, y seteamos el flag que ya estamos
256          conectados (START_COM) y arrancamos a jugar ( GAME_ON)*/
257          ENABLE = 1;
258          START_COM = 1;
259          GAME_ON = 1;
260
261      } else {
262          GAME_ON = 0;
263          ENABLE = 0;
264          START_COM = 0;
265          printf("Aun no se inicio comunicacion con otro jugador\n");
266
267      }
268
269      } else if ( START_COM == 1 ){
270
271          /*Mientras estemos conectados, comprobamos que un dato sea valido, si no recibi el
272          codigo de finalizacion*/
273          read_byte = LPC_UART0->RBR;
274          //printf("Recibi el byte: %d ",read_byte);
275          //printf("\n");
276          if (read_byte != 0xFF){
277
278              /*Lo tomo como dato valido*/
279              RCV_TONO = 1;
280              GAME_ON = 1;
281
282          } else {
283              GAME_ON = 0; //Fin de juego
284
285          }
286
287      }
288
289      }
290
291      }
292
293      }
294
295      }
296
297      }
298
299      }
300
301      }
302
303      }
304
305      }
306
307      }
308
309      }
310
311      }
312
313      }
314
315      }
316
317      }
318
319      }
320
321      }
322
323      }
324
325      }
326
327      }
328
329      }
330
331      }
332
333      }
334
335      }
336
337      }
338
339      }
340
341      }
342
343      }
344
345      }
346
347      }
348
349      }
350
351      }
352
353      }
354
355      }
356
357      }
358
359      }
360
361      }
362
363      }
364
365      }
366
367      }
368
369      }
370
371      }
372
373      }
374
375      }
376
377      }
378
379      }
380
381      }
382
383      }
384
385      }
386
387      }
388
389      }
390
391      }
392
393      }
394
395      }
396
397      }
398
399      }
400
401      }
402
403      }
404
405      }
406
407      }
408
409      }
410
411      }
412
413      }
414
415      }
416
417      }
418
419      }
420
421      }
422
423      }
424
425      }
426
427      }
428
429      }
430
431      }
432
433      }
434
435      }
436
437      }
438
439      }
440
441      }
442
443      }
444
445      }
446
447      }
448
449      }
450
451      }
452
453      }
454
455      }
456
457      }
458
459      }
460
461      }
462
463      }
464
465      }
466
467      }
468
469      }
470
471      }
472
473      }
474
475      }
476
477      }
478
479      }
480
481      }
482
483      }
484
485      }
486
487      }
488
489      }
490
491      }
492
493      }
494
495      }
496
497      }
498
499      }
500
501      }
502
503      }
504
505      }
506
507      }
508
509      }
510
511      }
512
513      }
514
515      }
516
517      }
518
519      }
520
521      }
522
523      }
524
525      }
526
527      }
528
529      }
530
531      }
532
533      }
534
535      }
536
537      }
538
539      }
540
541      }
542
543      }
544
545      }
546
547      }
548
549      }
550
551      }
552
553      }
554
555      }
556
557      }
558
559      }
560
561      }
562
563      }
564
565      }
566
567      }
568
569      }
570
571      }
572
573      }
574
575      }
576
577      }
578
579      }
580
581      }
582
583      }
584
585      }
586
587      }
588
589      }
590
591      }
592
593      }
594
595      }
596
597      }
598
599      }
600
601      }
602
603      }
604
605      }
606
607      }
608
609      }
610
611      }
612
613      }
614
615      }
616
617      }
618
619      }
620
621      }
622
623      }
624
625      }
626
627      }
628
629      }
630
631      }
632
633      }
634
635      }
636
637      }
638
639      }
640
641      }
642
643      }
644
645      }
646
647      }
648
649      }
650
651      }
652
653      }
654
655      }
656
657      }
658
659      }
660
661      }
662
663      }
664
665      }
666
667      }
668
669      }
670
671      }
672
673      }
674
675      }
676
677      }
678
679      }
680
681      }
682
683      }
684
685      }
686
687      }
688
689      }
690
691      }
692
693      }
694
695      }
696
697      }
698
699      }
700
701      }
702
703      }
704
705      }
706
707      }
708
709      }
710
711      }
712
713      }
714
715      }
716
717      }
718
719      }
720
721      }
722
723      }
724
725      }
726
727      }
728
729      }
730
731      }
732
733      }
734
735      }
736
737      }
738
739      }
740
741      }
742
743      }
744
745      }
746
747      }
748
749      }
750
751      }
752
753      }
754
755      }
756
757      }
758
759      }
760
761      }
762
763      }
764
765      }
766
767      }
768
769      }
770
771      }
772
773      }
774
775      }
776
777      }
778
779      }
780
781      }
782
783      }
784
785      }
786
787      }
788
789      }
790
791      }
792
793      }
794
795      }
796
797      }
798
799      }
800
801      }
802
803      }
804
805      }
806
807      }
808
809      }
810
811      }
812
813      }
814
815      }
816
817      }
818
819      }
820
821      }
822
823      }
824
825      }
826
827      }
828
829      }
830
831      }
832
833      }
834
835      }
836
837      }
838
839      }
840
841      }
842
843      }
844
845      }
846
847      }
848
849      }
850
851      }
852
853      }
854
855      }
856
857      }
858
859      }
860
861      }
862
863      }
864
865      }
866
867      }
868
869      }
870
871      }
872
873      }
874
875      }
876
877      }
878
879      }
880
881      }
882
883      }
884
885      }
886
887      }
888
889      }
890
891      }
892
893      }
894
895      }
896
897      }
898
899      }
900
901      }
902
903      }
904
905      }
906
907      }
908
909      }
910
911      }
912
913      }
914
915      }
916
917      }
918
919      }
920
921      }
922
923      }
924
925      }
926
927      }
928
929      }
930
931      }
932
933      }
934
935      }
936
937      }
938
939      }
940
941      }
942
943      }
944
945      }
946
947      }
948
949      }
950
951      }
952
953      }
954
955      }
956
957      }
958
959      }
960
961      }
962
963      }
964
965      }
966
967      }
968
969      }
970
971      }
972
973      }
974
975      }
976
977      }
978
979      }
980
981      }
982
983      }
984
985      }
986
987      }
988
989      }
990
991      }
992
993      }
994
995      }
996
997      }
998
999      }
1000
1001      }
1002
1003      }
1004
1005      }
1006
1007      }
1008
1009      }
1010
1011      }
1012
1013      }
1014
1015      }
1016
1017      }
1018
1019      }
1020
1021      }
1022
1023      }
1024
1025      }
1026
1027      }
1028
1029      }
1030
1031      }
1032
1033      }
1034
1035      }
1036
1037      }
1038
1039      }
1040
1041      }
1042
1043      }
1044
1045      }
1046
1047      }
1048
1049      }
1050
1051      }
1052
1053      }
1054
1055      }
1056
1057      }
1058
1059      }
1060
1061      }
1062
1063      }
1064
1065      }
1066
1067      }
1068
1069      }
1070
1071      }
1072
1073      }
1074
1075      }
1076
1077      }
1078
1079      }
1080
1081      }
1082
1083      }
1084
1085      }
1086
1087      }
1088
1089      }
1090
1091      }
1092
1093      }
1094
1095      }
1096
1097      }
1098
1099      }
1100
1101      }
1102
1103      }
1104
1105      }
1106
1107      }
1108
1109      }
1110
1111      }
1112
1113      }
1114
1115      }
1116
1117      }
1118
1119      }
1120
1121      }
1122
1123      }
1124
1125      }
1126
1127      }
1128
1129      }
1130
1131      }
1132
1133      }
1134
1135      }
1136
1137      }
1138
1139      }
1140
1141      }
1142
1143      }
1144
1145      }
1146
1147      }
1148
1149      }
1150
1151      }
1152
1153      }
1154
1155      }
1156
1157      }
1158
1159      }
1160
1161      }
1162
1163      }
1164
1165      }
1166
1167      }
1168
1169      }
1170
1171      }
1172
1173      }
1174
1175      }
1176
1177      }
1178
1179      }
1180
1181      }
1182
1183      }
1184
1185      }
1186
1187      }
1188
1189      }
1190
1191      }
1192
1193      }
1194
1195      }
1196
1197      }
1198
1199      }
1200
1201      }
1202
1203      }
1204
1205      }
1206
1207      }
1208
1209      }
1210
1211      }
1212
1213      }
1214
1215      }
1216
1217      }
1218
1219      }
1220
1221      }
1222
1223      }
1224
1225      }
1226
1227      }
1228
1229      }
1230
1231      }
1232
1233      }
1234
1235      }
1236
1237      }
1238
1239      }
1240
1241      }
1242
1243      }
1244
1245      }
1246
1247      }
1248
1249      }
1250
1251      }
1252
1253      }
1254
1255      }
1256
1257      }
1258
1259      }
1260
1261      }
1262
1263      }
1264
1265      }
1266
1267      }
1268
1269      }
1270
1271      }
1272
1273      }
1274
1275      }
1276
1277      }
1278
1279      }
1280
1281      }
1282
1283      }
1284
1285      }
1286
1287      }
1288
1289      }
1290
1291      }
1292
1293      }
1294
1295      }
1296
1297      }
1298
1299      }
1300
1301      }
1302
1303      }
1304
1305      }
1306
1307      }
1308
1309      }
1310
1311      }
1312
1313      }
1314
1315      }
1316
1317      }
1318
1319      }
1320
1321      }
1322
1323      }
1324
1325      }
1326
1327      }
1328
1329      }
1330
1331      }
1332
1333      }
1334
1335      }
1336
1337      }
1338
1339      }
1340
1341      }
1342
1343      }
1344
1345      }
1346
1347      }
1348
1349      }
1350
1351      }
1352
1353      }
1354
1355      }
1356
1357      }
1358
1359      }
1360
1361      }
1362
1363      }
1364
1365      }
1366
1367      }
1368
1369      }
1370
1371      }
1372
1373      }
1374
1375      }
1376
1377      }
1378
1379      }
1380
1381      }
1382
1383      }
1384
1385      }
1386
1387      }
1388
1389      }
1390
1391      }
1392
1393      }
1394
1395      }
1396
1397      }
1398
1399      }
1400
1401      }
1402
1403      }
1404
1405      }
1406
1407      }
1408
1409      }
1410
1411      }
1412
1413      }
1414
1415      }
1416
1417      }
1418
1419      }
1420
1421      }
1422
1423      }
1424
1425      }
1426
1427      }
1428
1429      }
1430
1431      }
1432
1433      }
1434
1435      }
1436
1437      }
1438
1439      }
1440
1441      }
1442
1443      }
1444
1445      }
1446
1447      }
1448
1449      }
1450
1451      }
1452
1453      }
1454
1455      }
1456
1457      }
1458
1459      }
1460
1461      }
1462
1463      }
1464
1465      }
1466
1467      }
1468
1469      }
1470
1471      }
1472
1473      }
1474
1475      }
1476
1477      }
1478
1479      }
1480
1481      }
1482
1483      }
1484
1485      }
1486
1487      }
1488
1489      }
1490
1491      }
1492
1493      }
1494
1495      }
1496
1497      }
1498
1499      }
1500
1501      }
1502
1503      }
1504
1505      }
1506
1507      }
1508
1509      }
1510
1511      }
1512
1513      }
1514
1515      }
1516
1517      }
1518
1519      }
1520
1521      }
1522
1523      }
1524
1525      }
1526
1527      }
1528
1529      }
1530
1531      }
1532
1533      }
1534
1535      }
1536
1537      }
1538
1539      }
1540
1541      }
1542
1543      }
1544
1545      }
1546
1547      }
1548
1549      }
1550
1551      }
1552
1553      }
1554
1555      }
1556
1557      }
1558
1559      }
1560
1561      }
1562
1563      }
1564
1565      }
1566
1567      }
1568
1569      }
1570
1571      }
1572
1573      }
1574
1575      }
1576
1577      }
1578
1579      }
1580
1581      }
1582
1583      }
1584
1585      }
1586
1587      }
1588
1589      }
1590
1591      }
1592
1593      }
1594
1595      }
1596
1597      }
1598
1599      }
1600
1601      }
1602
1603      }
1604
1605      }
1606
1607      }
1608
1609      }
1610
1611      }
1612
1613      }
1614
1615      }
1616
1617      }
1618
1619      }
1620
1621      }
1622
1623      }
1624
1625      }
1626
1627      }
1628
1629      }
1630
1631      }
1632
1633      }
1634
1635      }
1636
1637      }
1638
1639      }
1640
1641      }
1642
1643      }
1644
1645      }
1646
1647      }
1648
1649      }
1650
1651      }
1652
1653      }
1654
1655      }
1656
1657      }
1658
1659      }
1660
1661      }
1662
1663      }
1664
1665      }
1666
1667      }
1668
1669      }
1670
1671      }
1672
1673      }
1674
1675      }
1676
1677      }
1678
1679      }
1680
1681      }
1682
1683      }
1684
1685      }
1686
1687      }
1688
1689      }
1690
1691      }
1692
1693      }
1694
1695      }
1696
1697      }
1698
1699      }
1700
1701      }
1702
1703      }
1704
1705      }
1706
1707      }
1708
1709      }
1710
1711      }
1712
1713      }
1714
1715      }
1716
1717      }
1718
1719      }
1720
1721      }
1722
1723      }
1724
1725      }
1726
1727      }
1728
1729      }
1730
1731      }
1732
1733      }
1734
1735      }
1736
1737      }
1738
1739      }
1740
1741      }
1742
1743      }
1744
1745      }
1746
1747      }
1748
1749      }
1750
1751      }
1752
1753      }
1754
1755      }
1756
1757      }
1758
1759      }
1760
1761      }
1762
1763      }
1764
1765      }
1766
1767      }
1768
1769      }
1770
1771      }
1772
1773      }
1774
1775      }
1776
1777      }
1778
1779      }
1780
1781      }
1782
1783      }
1784
1785      }
1786
1787      }
1788
1789      }
1790
1791      }
1792
1793      }
1794
1795      }
1796
1797      }
1798
1799      }
1800
1801      }
1802
1803      }
1804
1805      }
1806
1807      }
1808
1809      }
1810
1811      }
1812
1813      }
1814
1815      }
1816
1817      }
1818
1819      }
1820
1821      }
1822
1823      }
1824
1825      }
1826
1827      }
1828
1829      }
1830
1831      }
1832
1833      }
1834
1835      }
1836
1837      }
1838
1839      }
1840
1841      }
1842
1843      }
1844
1845      }
1846
1847      }
1848
1849      }
1850
1851      }
1852
1853      }
1854
1855      }
1856
1857      }
1858
1859      }
1860
1861      }
1862
1863      }
1864
1865      }
1866
1867      }
1868
1869      }
1870
1871      }
1872
1873      }
1874
1875      }
1876
1877      }
1878
1879      }
1880
1881      }
1882
1883      }
1884
1885      }
1886
1887      }
1888
1889      }
1890
1891      }
1892
1893      }
1894
1895      }
1896
1897      }
1898
1899      }
1900
1901      }
1902
1903      }
1904
1905      }
1906
1907      }
1908
1909      }
1910
1911      }
1912
1913      }
1914
1915      }
1916
1917      }
1918
1919      }
1920
1921      }
1922
1923      }
1924
1925      }
1926
1927      }
1928
1929      }
1929
1930      }
1931
1932      }
1933
1934      }
1935
1936      }
1937
1938      }
1939
1940      }
1940
1941      }
1941
1942      }
1942
1943      }
1943
1944      }
1944
1945      }
1945
1946      }
1946
1947      }
1947
1948      }
1948
1949      }
1949
1950      }
1950
1951      }
1951
1952      }
1952
1953      }
1953
1954      }
1954
1955      }
1955
1956      }
1956
1957      }
1957
1958      }
1958
1959      }
1959
1960      }
1960
1961      }
1961
1962      }
1962
1963      }
1963
1964      }
1964
1965      }
1965
1966      }
1966
1967      }
1967
1968      }
1968
1969      }
1969
1970      }
1970
1971      }
1971
1972      }
1972
1973      }
1973
1974      }
1974
1975      }
1975
1976      }
1976
1977      }
1977
1978      }
1978
1979      }
1979
1980      }
1980
1981      }
1981
1982      }
1982
1983      }
1983
1984      }
1984
1985      }
1985
1986      }
1986
1987      }
1987
1988      }
1988
1989      }
1989
1990      }
1990
1991      }
1991
1992      }
1992
1993      }
1993
1994      }
1994
1995      }
1995
1996      }
1996
1997      }
1997
1998      }
1998
1999      }
1999
2000      }
2000
2001      }
2001
2002      }
2002
2003      }
2003
2004      }
2004
2005      }
2005
2006      }
2006
2007      }
2007
2008      }
2008
2009      }
2009
2010      }
2010
2011      }
2011
2012      }
2012
2013      }
2013
2014      }
2014
2015      }
2015
2016      }
2016
2017      }
2017
2018      }
2018
2019      }
2019
2020      }
2020
2021      }
2021
2022      }
2022
2023      }
2023
2024      }
2024
2025      }
2025
2026      }
2026
2027      }
2027
2028      }
2028
2029      }
2029
2030      }
2030
2031      }
2031
2032      }
2032
2033      }
2033
2034      }
2034
2035      }
2035
2036      }
2036
2037      }
2037
2038      }
2038
2039      }
2039
2040      }
2040
2041      }
2041
2042      }
2042
2043      }
2043
2044      }
2044
2045      }
2045
2046      }
2046
2047      }
2047
2048      }
2048
2049      }
2049
2050      }
2050
2051      }
2051
2052      }
2052
2053      }
2053
2054      }
2054
2055      }
2055
2056      }
2056
2057      }
2057
2058      }
2058
2059      }
2059
2060      }
2060
2061      }
2061
2062      }
2062
2063      }
2063
2064      }
2064
2065      }
2065
2066      }
2066
2067      }
2067
2068      }
2068
2069      }
2069
2070      }
2070
2071      }
2071
2072      }
2072
2073      }
2073
2074      }
2074
2075      }
2075
2076      }
2076
2077      }
2077
2078      }
2078
2079      }
2079
2080      }
2080
2081      }
2081
2082      }
2082
2083      }
2083
2084      }
2084
2085      }
2085
2086      }
2086
2087      }
2087
2088      }
2088
2089      }
2089
2090      }
2090
2091      }
2091
2092      }
2092
2093      }
2093
2094      }
2094
2095      }
2095
2096      }
2096
2097      }
2097
2098      }
2098
2099      }
2099
2100      }
2100
2101      }
2101
2102      }
2102
2103      }
2103
2104      }
2104
2105      }
2105
2106      }
2106
2107      }
2107
2108      }
2108
2109      }
2109
2110      }
2110
2111      }
2111
2112      }
2112
2113      }
2113
2114      }
2114
2115      }
2115
2116      }
2116
2117      }
2117
2118      }
2118
2119      }
2119
2120      }
2120
2121      }
2121
2122      }
2122
2123      }
2123
2124      }
2124
2125      }
2125
2126      }
2126
2127      }
2127
2128      }
2128
2129      }
2129
2130      }
2130
2131      }
2131
2132      }
2132
2133      }
2133
2134      }
2134
2135      }
2135
2136      }
2136
2137      }
2137
2138      }
2138
2139      }
2139
2140      }
2140
2141      }
2141
2142      }
2142
2143      }
2143
2
```



```

299 /*
300 * Comprueba en que posicion se encuentra el Stick y devuelve
301 * el valor correspondiente a la posicion de este:
302 * Superior: - 1
303 * Derecha: - 2
304 * Inferior: - 3
305 * Izquierda: - 4
306 * Reposo: - 0
307 */
308
309 uint8_t
310 movimiento_Stick (void)
311 {
312     uint16_t digital_X, digital_Y;
313     float X_vol, Y_vol;
314     digital_X = (LPC_ADC->ADDR0 >> 4 & 0xFFFF);
315     digital_Y = (LPC_ADC->ADDR1 >> 4 & 0xFFFF);
316     X_vol = (digital_X * 3.3 / 4095.0);
317     Y_vol = (digital_Y * 3.3 / 4095.0);
318     //printf ("X_vol: %f\n", X_vol);
319     //printf ("Y_vol: %f\n", Y_vol);
320
321     if (Y_vol > 2.5)
322     {
323         return 1;
324     }
325     else if (X_vol > 2.5)
326     {
327         return 2;
328     }
329     else if (Y_vol < 0.7)
330     {
331         return 3;
332     }
333     else if (X_vol < 0.7)
334     {
335         return 4;
336     }
337     else return 0;
338 }
339
340 /*
341 * Borra la variable de control SELMODE
342 */
343
344 void
345 clear_SELMODE (void)
346 {
347     SEL_MODE = 0;
348 }
349
350 /*
351 * Devuelve el valor de la variable PULSADOR. Variable asociada
352 * con los botones, los valores son:
353 * Superior: -1
354 * Derecho: -2
355 * Inferior: -3
356 * Izquierda: -4
357 * Default: -0
358 *
359 */

```

```
360
361 uint8_t
362 getPulsador (void)
363 {
364     return PULSADOR;
365 }
366
367 /*
368  * Pone a valor por defecto la variable PULSADOR
369  */
370
371 void
372 clearPulsador (void)
373 {
374     PULSADOR = 0;
375 }
376
377 /*
378  * Setea la variable de control SELMODE
379  */
380
381 void
382 set_SELMODE (uint8_t valor)
383 {
384     SEL_MODE = valor;
385 }
386
387 /*
388  * Obtener el valor de la variable de control SELMODE
389  */
390
391 uint8_t
392 get_SELMODE (void)
393 {
394     return SEL_MODE;
395 }
396
397 uint8_t
398 get_SECUENCIA (void)
399 {
400     return SECUENCIA;
401 }
402
403 void
404 set_SECUENCIA (uint8_t valor)
405 {
406     SECUENCIA = valor;
407 }
408
409 uint8_t
410 get_CONEXION (void)
411 {
412     return CONEXION;
413 }
414
415 void
416 set_CONEXION (uint8_t valor)
417 {
418     CONEXION = valor;
419 }
420
```

```
421
422 uint8_t
423 get_ENABLE(void)
424 {
425     return ENABLE;
426 }
427
428 void
429 set_ENABLE (uint8_t valor)
430 {
431     ENABLE= valor;
432 }
433
434
435 /*Para manejo de variables en el modo 4 (comunicacion UART)*/
436 char
437 get_read_byte(void){
438
439     return read_byte;
440 }
441
442 void
443 set_read_byte(char bts ){
444
445     read_byte = bts;
446
447 }
448
449
450
451 /*Para recepcion de tonos en modo multijugador*/
452 uint8_t
453 get_user_tono(void){
454     if (RCV_TONO==1){
455         return read_byte;
456     } else {
457         return 6; //6 indica que no es un dato valido. Mediante el metodo set se usa para
458         inhibir lectura de buffer
459     }
460 }
461
462 /*Para saber si el juego esta en curso*/
463 uint8_t
464 get_GAME_ON(void){
465     return GAME_ON;
466 }
467
468
469 void
470 set_GAME_ON(uint8_t value){
471     GAME_ON = value;
472 }
473
474
475 void
476 set_START_COM(uint8_t value){
477     START_COM = value;
478 }
479
480
```

```
481
482
483 /*
484  * Envia por UART los datos pasados por parametro
485  */
486
487 void
488 Uart_Send (char * dato)
489 {
490     while( ( (LPC_UART0->LSR >> 5) & 1 ) ==0 );
491     LPC_UART0->THR = *dato;
492 }
493
494 /*
495  * Algoritmo Antirrebote
496  */
497
498 int
499 debounce (uint8_t Sample)
500 {
501     unsigned static int Samples[NUM_SAMPLES];
502     unsigned static int index = 0;
503     unsigned static int LastDebounceResult = 0;
504     unsigned int andTerm = Sample;
505     unsigned int orTerm = Sample;
506
507     Samples[index++] = Sample;
508     if (index >= NUM_SAMPLES) index = 0;
509
510     for (int j = 0; j < NUM_SAMPLES; j++)
511     {
512         andTerm &= Samples[j];
513         orTerm |= Samples[j];
514     }
515     LastDebounceResult = ((LastDebounceResult & orTerm) | andTerm);
516     return LastDebounceResult;
517 }
```

IV. **simon.c**

Este archivo contiene el código principal del juego. En el mismo se definen los diferentes modos. Se anexan los comentarios respectivos en el mismo código, a continuación. Incluye los siguientes archivos headers:

- *simon.h*
- *configuracion.h*
- *activacion.h*
- *lcd.h*

```
1
2 #include "simon.h"
3 #include "configuracion.h"
4 #include "activacion.h"
5 #include "time.h"
6 #include "lcd.h"
7 #include <string.h>
8 #include <stdlib.h>
9 #include <stdio.h>
10
11 //Variable empleados en los modos de juego
12 char *mensaje;
13 char buffer[10];
14 uint32_t tiempos[4] =
15 { 6250000, 5000000, 3750000, 2000000 };
16
17
18 /*
19  * Funcion que ejecuta una secuencia de colores para indicar el inicio del juego
20  */
21
22 void
23 simon_Init (uint32_t tiempo)
24 {
25     int i;
26
27     for (i = 0; i < 3; ++i)
28     {
29         leds_ON (1, 0, 0, 0);
30         retardo (tiempo);
31         leds_OFF (1, 0, 0, 0);
32
33         leds_ON (0, 1, 0, 0);
34         retardo (tiempo);
35         leds_OFF (0, 1, 0, 0);
36
37         leds_ON (0, 0, 1, 0);
38         retardo (tiempo);
39         leds_OFF (0, 0, 1, 0);
40
41         leds_ON (0, 0, 0, 1);
42         retardo (tiempo);
43         leds_OFF (0, 0, 0, 1);
44     }
45
46     for (i = 0; i < 2; ++i)
47     {
48         leds_ON (1, 0, 0, 1);
49         retardo (tiempo * 5);
50         leds_OFF (1, 0, 0, 1);
51
52         leds_ON (0, 1, 1, 0);
53         retardo (tiempo * 5);
54         leds_OFF (0, 1, 1, 0);
55     }
56
57     leds_ON (1, 0, 1, 0);
58     retardo (tiempo * 5);
59     leds_OFF (1, 0, 1, 0);
```

```
60
61     leds_ON (0, 1, 0, 1);
62     retardo (tiempo * 5);
63     leds_OFF (0, 1, 0, 1);
64
65     leds_ON (1, 1, 0, 0);
66     retardo (tiempo * 5);
67     leds_OFF (1, 1, 0, 0);
68
69     leds_ON (0, 0, 1, 1);
70     retardo (tiempo * 5);
71     leds_OFF (0, 0, 1, 1);
72
73     leds_ON (1, 1, 1, 1);
74     retardo (tiempo * 5);
75     leds_OFF (1, 1, 1, 1);
76
77 }
78
79 /*
80  * Funcion que ejecuta una secuencia de colores para indicar el exito del jugador
81  * en el juego.
82  */
83
84 void
85 simon_SUCCESS (void)
86 {
87     int i;
88     for (i = 0; i < 4; i++)
89     {
90         set_SELMODE (0);
91         leds_ON (1, 0, 0, 0);
92         retardo (1500000);
93         leds_ON (0, 1, 0, 0);
94         retardo (1500000);
95         leds_ON (0, 0, 0, 1);
96         retardo (1500000);
97         leds_ON (0, 0, 1, 0);
98         retardo (1500000);
99
100        leds_OFF (1, 0, 0, 0);
101        retardo (1500000);
102        leds_OFF (0, 1, 0, 0);
103        retardo (1500000);
104        leds_OFF (0, 0, 0, 1);
105        retardo (1500000);
106        leds_OFF (0, 0, 1, 0);
107        retardo (1500000);
108    }
109 }
110
111 /*Para indicar correcto ingreso de secuencia en modo multijugador (se encienden todos los
112    leds en simultaneo, para evitar confusiones)*/
113 void
114 simon_MultiSUCCESS (void)
115 {
116     int i;
117     for (i = 0; i < 2; i++)
118     {
119         set_SELMODE (0);
120         leds_ON (1, 1, 1, 1);
```

```

120     retardo (2500000);
121     leds_OFF (1, 1, 1, 1);
122
123 }
124 }
125
126
127 /*
128  * Funcion que ejecuta una secuencia de colores para indicar el fracaso del jugador
129  * en el juego
130  */
131 void
132 simon_FAIL (void)
133 {
134     int i;
135     set_SELMODE (0);
136     for (i = 0; i < 3; i++)
137     {
138         leds_ON (1, 1, 1, 1);
139         retardo (3000000);
140         leds_OFF (1, 1, 1, 1);
141         retardo (3000000);
142     }
143 }
144
145 /*
146  * MODO 1: Modo Clasico de Simon. Repetir secuencia hasta un numero determinado
147  * de colores. Numero de turnos/colores en secuencia:
148  *     - Nivel 1: 4 turnos
149  *     - Nivel 2: 8 turnos
150  *     - Nivel 3: 12 turnos
151  *     - Nivel 4: 16 turnos
152  * Cada 4 turnos, disminuye el tiempo que se muestran los colores.
153  * La funcion devuelve si el jugador ha tenido exito o no en *exito.
154  */
155 void
156 modo_1 (int nivel, int* exito)
157 {
158     uint8_t i, turno, fallo, turno_jugador, boton_pulsado, numero_turnos,
159     colores[MAX_TURNOS]; // Declaramos el vector de colores a acertar
160
161     for (int i = 0; i < MAX_TURNOS; i++)
162     {
163         colores[i] = 0;
164     }
165
166     turno = 1;
167     fallo = 0;
168     set_SELMODE (10);
169     srand (time (NULL)); // Inicializamos semilla
170
171     numero_turnos = (nivel * 4);
172
173     // Bucle que hace el juego hasta que se gana o se pierde
174     while (turno <= numero_turnos && fallo == 0)
175     {
176         sprintf (buffer, "%d", turno);
177         Lcd_Show ("Turno:", 1, 1, 1, 0);
178         Lcd_Show (buffer, 0, 0, 0, 0);
179         Lcd_Show ("—", 0, 0, 0, 0);
180         sprintf (buffer, "%d", numero_turnos);

```

```

181     set_SELMODE (10);
182     colores[turno - 1] = rand () % 4; // Nuevo color
183     Lcd_Show (buffer, 0, 0, 0, 2000000);
184
185     for (i = 0; i < turno; i++) // Imprimimos la secuencia de colores
186     {
187         set_SECUENCIA (1);
188         led_color_ON (colores[i]);
189         while (get_SECUENCIA ())
190             {
191             }
192         retardo (500000);
193     }
194
195     turno_jugador = 1;
196     while (turno_jugador <= turno && fallo == 0)
197     {
198         clear_SELMODE ();
199         clearPulsador ();
200         boton_pulsado = getPulsador ();
201         while (!boton_pulsado)
202             {
203                 boton_pulsado = getPulsador ();
204                 retardo (15000);
205             }
206
207         set_SELMODE (10);
208         set_SECUENCIA (1);
209         led_color_ON (colores[turno_jugador - 1]);
210         while (get_SECUENCIA ())
211             {
212             }
213         retardo (500000);
214
215         if ((boton_pulsado - 1) == colores[turno_jugador - 1]) turno_jugador++;
216         else fallo = 1;
217     }
218     turno++;
219
220     if (turno % 4 == 0)
221     {
222         cambiar_Tiempo (turno);
223     }
224
225     } // while
226
227     if (fallo == 0) *exito = 1;
228     else *exito = 0;
229
230     setTiempo (0, tiempos[0]);
231 }
232
233 /*
234 * MODO 2: Modo Inverso de Simon. Se denera repetir la secuencia hasta un numero
235 *   determinado
236 *   de colores de manera inversa. Numero de turnos/colores en secuencia:
237 *       - Nivel 1: 4 turnos
238 *       - Nivel 2: 8 turnos
239 *       - Nivel 3: 12 turnos
240 *       - Nivel 4: 16 turnos
241 *   Cada 4 turnos, disminuye el tiempo que se muestran los colores.

```



```

241  * La funcion devuelve si el jugador ha tenido exito o no en *exito.
242  */
243
244  void
245  modo_2 (int nivel, int* exito)
246  {
247      int i, turno, fallo, turno_jugador, boton_pulsado, numero_turnos,
248          colores[MAX_TURNOS]; // Declaramos el vector de colores a acertar
249
250      for (int i = 0; i < MAX_TURNOS; i++)
251      {
252          colores[i] = 0;
253      }
254
255      turno_jugador = 0;
256      boton_pulsado = 0;
257      numero_turnos = 0;
258      turno = 1;
259      fallo = 0;
260      set_SELMODE (10);
261      srand (time (NULL)); // Inicializamos semilla
262
263      numero_turnos = nivel * 4;
264
265      // Bucle que hace el juego hasta que se gana o se pierde
266      while (turno <= numero_turnos && fallo == 0)
267      {
268          sprintf (buffer, "%d", turno);
269          Lcd_Show ("Turno:", 1, 1, 1, 0);
270          Lcd_Show (buffer, 0, 0, 0, 0);
271          Lcd_Show ("—", 0, 0, 0, 0);
272          sprintf (buffer, "%d", numero_turnos);
273          set_SELMODE (10);
274          colores[turno - 1] = rand () % 4; // Nuevo color
275          Lcd_Show (buffer, 0, 0, 0, 2000000);
276
277          for (i = 0; i < turno; i++) // Imprimimos la secuencia de colores
278          {
279              set_SECUENCIA (1);
280              led_color_ON (colores[i]);
281              while (get_SECUENCIA ())
282              {
283              }
284              retardo (500000);
285          }
286
287          turno_jugador = turno - 1;
288          while (turno_jugador >= 0 && fallo == 0)
289          {
290              clear_SELMODE ();
291              clearPulsador ();
292              boton_pulsado = getPulsador ();
293              while (!boton_pulsado)
294              {
295                  boton_pulsado = getPulsador ();
296                  retardo (15000);
297              }
298              set_SELMODE (10);
299              set_SECUENCIA (1);
300              led_color_ON (colores[turno_jugador]);
301              while (get_SECUENCIA ())

```

```

302     {
303     }
304     retardo (500000);
305
306     if ((boton_pulsado - 1) == colores[turno_jugador]) turno_jugador--;
307     else fallo = 1;
308
309 }
310     turno++;
311
312     if (turno % 4 == 0)
313     {
314         cambiar_Tiempo (turno);
315     }
316
317     } // while
318     if (fallo == 0) *exito = 1;
319     else *exito = 0;
320 }
321
322 /*
323 * MODO 3: Modo Aleatorio de Simon. Se debera repetir la secuencia hasta un numero
324 *   determinado
325 *   de colores. En cada turno se determina de forma aleatoria si el jugado debe
326 *   introducir
327 *   la secuencia de manera clasico o inversa. Se indicara mediante el LCD.
328 *   A su vez, se debe utilizar un Stick para activar la secuencia de colores en vez de
329 *   los
330 *   clasicos botones.
331 *   Numero de turnos/colores en secuencia:
332 *       - Nivel 1: 4 turnos
333 *       - Nivel 2: 8 turnos
334 *       - Nivel 3: 12 turnos
335 *       - Nivel 4: 16 turnos
336 *   Cada 4 turnos, disminuye el tiempo que se muestran los colores.
337 *   La funcion devuelve si el jugador ha tenido exito o no en *exito.
338 *
339 */
340 void
341 modo_3 (int nivel, int* exito)
342 {
343     int i, turno, fallo, turno_jugador, boton_pulsado, sentido, numero_turnos,
344         colores[MAX_TURNOS]; // Declaramos el vector de colores a acertar
345
346     boton_pulsado = 0;
347     turno = 1;
348     fallo = 0;
349     srand (time (NULL)); // Inicializamos semilla
350
351     numero_turnos = nivel * 4;
352
353     // Bucle que hace el juego hasta que se gana o se pierde
354     while (turno <= numero_turnos && fallo == 0)
355     {
356         boton_pulsado = 0;
357         NVIC_DisableIRQ (EINT0_IRQn);
358         NVIC_DisableIRQ (EINT1_IRQn);
359         NVIC_DisableIRQ (EINT2_IRQn);
360         NVIC_DisableIRQ (EINT3_IRQn);

```

```

360     sprintf (buffer, "%d", turno);
361     Lcd_Show ("Turno:", 1, 1, 1, 0);
362     Lcd_Show (buffer, 0, 0, 0, 0);
363     Lcd_Show ("_", 0, 0, 0, 0);
364     sprintf (buffer, "%d", numero_turnos);
365     set_SELMODE (10);
366     colores[turno - 1] = rand () % 4; // Nuevo color
367     Lcd_Show (buffer, 0, 0, 0, 2000000);
368
369     for (i = 0; i < turno; i++) // Imprimimos la secuencia de colores
370     {
371         set_SECUENCIA (1);
372         led_color_ON (colores[i]);
373         while (get_SECUENCIA ())
374         {
375             // Retardo
376         }
377         retardo (500000);
378     }
379
380     sentido = rand () % 2;
381
382     if (sentido == 0)
383     { // Sentido directo
384
385         Lcd_Show ("DIRECTO", 2, 7, 0, 0);
386         turno_jugador = 1;
387         while (turno_jugador <= turno && fallo == 0)
388         {
389             boton_pulsado = 0;
390             while (!boton_pulsado)
391             {
392                 boton_pulsado = movimiento_Stick ();
393                 retardo (750000);
394             }
395             set_SELMODE (10);
396             set_SECUENCIA (1);
397             led_color_ON (colores[turno_jugador - 1]);
398             while (get_SECUENCIA ())
399             {
400                 // Retardo
401             }
402             retardo (500000);
403
404             if ((boton_pulsado - 1) == colores[turno_jugador - 1]) turno_jugador++;
405             else fallo = 1;
406         }
407     }
408     else
409     { // Sentido inverso
410         Lcd_Show ("INVERSO", 2, 7, 0, 0);
411         turno_jugador = turno - 1;
412         while (turno_jugador >= 0 && fallo == 0)
413         {
414             boton_pulsado = 0;
415             while (!boton_pulsado)
416             {
417                 boton_pulsado = movimiento_Stick ();
418                 retardo (750000);
419             }
420             set_SELMODE (10);

```

```

421     set_SECUENCIA (1);
422     led_color_ON (colores[turno_jugador]);
423     while (get_SECUENCIA ())
424     {
425     }
426     retardo (500000);
427
428     if ((boton_pulsado - 1) == colores[turno_jugador]) turno_jugador--;
429     else fallo = 1;
430
431     }
432 }
433     turno++;
434
435     if (turno % 4 == 0)
436     {
437     cambiar_Tiempo (turno);
438     }
439
440     } // while
441     if (fallo == 0) *exito = 1;
442     else *exito = 0;
443
444     NVIC_ClearPendingIRQ (EINT0_IRQn);
445     NVIC_ClearPendingIRQ (EINT1_IRQn);
446     NVIC_ClearPendingIRQ (EINT2_IRQn);
447     NVIC_ClearPendingIRQ (EINT3_IRQn);
448     retardo (15000);
449     NVIC_EnableIRQ (EINT0_IRQn);
450     NVIC_EnableIRQ (EINT1_IRQn);
451     NVIC_EnableIRQ (EINT2_IRQn);
452     NVIC_EnableIRQ (EINT3_IRQn);
453 }
454
455 /*
456 * MODO 4: Modo Multijugador de Simon. El jugador que inicia la partida selecciona
457 * el color de inicio. El segundo jugador debe replicar la secuencia y
458 * si es correcta agregara otro color a la secuencia existente.
459 * Cada jugador debe replicar la secuencia existente y luego agregar
460 * otro color.
461 * El primer jugador que se equivoca en la secuencia, pierde.
462 * Se dispone de un tiempo de 10 segundos para establecer la conexion.
463 * La funcion devuelve si el jugador ha tenido exito o no en *exito.
464 *
465 * Este codigo es el de uno de los usuarios, no el del servidor (placa que inicia
466 * comunicacion)
467 */
468 void
469 modo_4 (int nivel, int* exito)
470 {
471
472     int num_t, tono_a_replicar, boton_pulsado,
473     colores[MAX_TURNOS*4]; // Declaramos el vector de colores a acertar
474
475     int j = 0; //Para almacenamiento de tonos
476
477     /*Arranque y cfg periferico UART (solo en modo 4 es necesario)*/
478     uartConfig ();
479     NVIC_EnableIRQ (UART0_IRQn);
480

```

```

481  /*Espero inicio de comunicacion. get_ENABLE() devuelve el flag ENABLE manejado por el
    handler de UART*/
482  while (get_ENABLE()==0){
483      printf("Esperando a iniciar juego.. \n");
484  }
485
486  printf("Conexion exitosa, inicio SIMON MULTIJUGADOR.. \n");
487
488  /*Inicializamos arreglo*/
489  for (int i = 0; i < MAX_TURNOS*4; i++) {
490      colores[i] = 9; //9 ->significa libre
491  }
492  num_t = 0; //Numero de turnos que transcurrieron
493
494  set_SELMODE (10);
495
496  /*Mientras este jugando (indicado por GAME ON) se ejecuta esta secuencia de operaciones
    */
497  while(get_GAME_ON()==1){
498
499      tono_a_replicar = 1; //Para recorrer arreglo de tonos
500
501      printf("\n Esperando tono player 1.. \n");
502      while(get_user_tono()==6){
503
504      }
505      /*Si recibí 0xFF por algún motivo, mal ingreso o mala replica, el otro usuario perdió
    y me indica que el juego finalizó*/
506      if ((int) get_user_tono() ==255){
507          printf("Usted gana, el player 1 ingreso mal \n");
508          *exito=1;
509          return;
510
511      }
512
513      //Leo byte/tono recibido por el otro usuario
514      colores[j] = (int) get_user_tono();
515
516      //Inhibo variable global de lectura de buffer para no leer basura o el mismo byte
    recibido en esta iteración, en la próxima iteración
517      set_read_byte(6);
518
519      //      printf("Colores:");
520      //      for (int m = 0; m<MAX_TURNOS*4; m++){
521      //
522      //          printf("%d ", colores[m]);
523      //      }
524      //      printf("\n");
525
526      num_t = 0;
527      /*Calculo dinámico de cuantos tonos hay en tonos en curso (cada player agrega uno)*/
528      for(int w = 0; w < MAX_TURNOS * 4; w ++){
529          {
530              if(colores[w] != 9)
531              {
532                  num_t++;
533              }
534          }
535
536
537      for (int pled = 0; pled < num_t; pled++)// Mostramos la secuencia de colores a

```

```

538     replicar
539     {
540         set_SELMODE(10);
541         set_SECUENCIA(1);
542         led_color_ON (colores[pled]);
543         while (get_SECUENCIA())
544         {
545         }
546     }
547
548     /*Voy cacheando ingreso de usuario, tono a tono, asi comprobar que primero replique
549     bien la secuencia*/
550     while (tono_a_replicar <= num_t){
551
552         clear_SELMODE ();
553         clearPulsador ();
554         boton_pulsado = getPulsador ();
555         while (!boton_pulsado)
556         {
557             boton_pulsado = getPulsador ();
558             retardo (1500);
559         }
560         set_SELMODE (10);
561         set_SECUENCIA (1);
562         led_color_ON (boton_pulsado - 1);
563         while (get_SECUENCIA ())
564         {
565         }
566         if ((boton_pulsado - 1) == colores[tono_a_replicar - 1]) {
567             printf("\n Metiste bien el tono capo \n");
568
569             tono_a_replicar++;
570         } else {
571
572             /*Perdi, envio 0xFF */
573             char error = 0xFF;
574             while( ( (LPC_UART0->LSR >> 5) & 1 ) ==0 );
575             LPC_UART0 -> THR = error;
576             retardo(500000);
577             /*Se setean las correspondientes flags*/
578             set_GAME_ON(0);
579             set_START_COM(0);
580             /*Deshabilito interrupciones*/
581             NVIC_DisableIRQ (UART0_IRQn);
582             /*Chau modo 4*/
583             *exito = 0;
584             return;
585         }
586     }
587
588
589     /*Indicamos al usuario que replico bien la secuencia actual*/
590     simon_MultiSUCCESS();
591
592     /*Ahora debe agregar un tono, cualquiera que el quiera*/
593     clear_SELMODE ();
594     clearPulsador ();
595     boton_pulsado = getPulsador ();
596     while (!boton_pulsado){

```

```

597     boton_pulsado = getPulsador ();
598     retardo (1500);
599 }
600
601
602
603     /*Enciendo led del boton que pulso, indicativamente*/
604     set_SELMODE (10);
605     set_SECUENCIA (1);
606     led_color_ON (boton_pulsado-1);
607
608     /*Agrego tono correspondiente al boton que pulso, a la secuencia en cuestion*/
609     j++;
610     colores[j] = boton_pulsado-1; //codigo boton 1 2 3 4 -> codigo leds/colores 0 1 2
611     3 (por eso el -1)
612
613     /*Envio tono al otro jugador*/
614     char tono_a_enviar = (char) boton_pulsado-1;
615
616     //printf("\nTe mande : %d ", tono_a_enviar);
617     while( ( (LPC_UART0->LSR >> 5) & 1 ) ==0 );
618     LPC_UART0 -> THR = tono_a_enviar;
619
620     /*Vuelvo a aumentar j, puntero a tonos en secuencia vigente*/
621     j++;
622
623 }
624 //printf("\n Chau modo4 \n");
625 NVIC_DisableIRQ (UART0_IRQn);
626 }
627
628
629
630 void
631 retardo (uint32_t tiempo)
632 {
633     for (int i = 0; i < tiempo; i++)
634         ;
635 }
636
637 void
638 cambiar_Tiempo (uint8_t turno)
639 {
640
641     if (turno > 12)
642     {
643         setTiempo (0, tiempos[3]);
644     }
645     else if (turno > 8)
646     {
647         setTiempo (0, tiempos[2]);
648     }
649     else if (turno > 4)
650     {
651         setTiempo (0, tiempos[1]);
652     }
653     else setTiempo (0, tiempos[0]);
654 }
655

```

V. leds.c

Archivo simple, con las funciones relacionadas a los leds para facilitar exhibición de secuencia, indicativos, entre otros.

```
1 #include "leds.h"
2 #include "configuracion.h"
3 #include "activacion.h"
4
5
6 /*
7  * Funcion que inicializa registros del GPIO2 para el uso de
8  * los leds de los botones
9  */
10 void
11 leds_Init (void)
12 {
13     LPC_GPIO2->FIODIR |= (1 << LED_Ve) | (1 << LED_Ro) | (1 << LED_Az)
14     | (1 << LED_Am);
15 }
16
17 /*
18  * Funcion que enciende el led verde. Si se desea generar tambien el
19  * tono se debe setear el valor de SELMODE a 10
20  */
21 void
22 led_verde_ON (void)
23 {
24     LPC_GPIO2->FIOSET |= (1 << LED_Ve);
25     if (get_SELMODE () == 10)
26     {
27         EINT0_IRQHandler ();
28     }
29 }
30
31 /*
32  * Funcion que apaga el led verde
33  */
34 void
35 led_verde_OFF (void)
36 {
37     LPC_GPIO2->FIOCLR |= (1 << LED_Ve);
38 }
39
40 /*
41  * Funcion que enciende el led amarillo. Si se desea generar tambien el
42  * tono se debe setear el valor de SELMODE a 10
43  */
44 void
45 led_amarillo_ON (void)
46 {
47     LPC_GPIO2->FIOSET |= (1 << LED_Am);
48     if (get_SELMODE () == 10)
49     {
50         EINT3_IRQHandler ();
51     }
52 }
53
```



```
54 /*
55  * Funcion que apaga el led amarillo.
56  */
57 void
58 led_amarillo_OFF (void)
59 {
60     LPC_GPIO2->FIOCLR |= (1 << LED_Am);
61 }
62
63 /*
64  * Funcion que enciende el led rojo. Si se desea generar tambien el
65  * tono se debe setear el valor de SELMODE a 10
66  */
67 void
68 led_rojo_ON (void)
69 {
70     LPC_GPIO2->FIOSET |= (1 << LED_Ro);
71     if (get_SELMODE () == 10)
72     {
73         EINT1_IRQHandler ();
74     }
75 }
76
77 /*
78  * Funcion que apaga el led rojo.
79  */
80 void
81 led_rojo_OFF (void)
82 {
83     LPC_GPIO2->FIOCLR |= (1 << LED_Ro);
84 }
85
86 /*
87  * Funcion que enciende el led azul. Si se desea generar tambien el
88  * tono se debe setear el valor de SELMODE a 10
89  */
90 void
91 led_azul_ON (void)
92 {
93     LPC_GPIO2->FIOSET |= (1 << LED_Az);
94     if (get_SELMODE () == 10)
95     {
96         EINT2_IRQHandler ();
97     }
98 }
99
100 /*
101  * Funcion que apaga el led azul
102  */
103 void
104 led_azul_OFF (void)
105 {
106     LPC_GPIO2->FIOCLR |= (1 << LED_Az);
107 }
108
109 /*
110  * Funcion que enciende leds segun los parametros de entrada ledV, ledA, ledR, ledAz
111  * ledV - Verde
112  * ledA - Amarillo
113  * ledR - Rojo
114  * ledAz - Azul
```

```
115  */
116  void
117  leds_ON (unsigned short ledV, unsigned short ledA, unsigned short ledR, unsigned short
        ledAz)
118  {
119      if (ledV == 1) led_verde_ON ();
120      if (ledA == 1) led_amarillo_ON ();
121      if (ledR == 1) led_rojo_ON ();
122      if (ledAz == 1) led_azul_ON ();
123  }
124
125  /*
126  *  Funcion que apaga leds segun los parametros de entrada ledV, ledA, ledR, ledAz
127  *      ledV  - Verde
128  *      ledA  - Amarillo
129  *      ledR  - Rojo
130  *      ledAz - Azul
131  */
132  void
133  leds_OFF (unsigned short ledV, unsigned short ledA, unsigned short ledR, unsigned short
        ledAz)
134  {
135      if (ledV == 1) led_verde_OFF ();
136      if (ledA == 1) led_amarillo_OFF ();
137      if (ledR == 1) led_rojo_OFF ();
138      if (ledAz == 1) led_azul_OFF ();
139  }
140
141  /*
142  *  Funcion que enciende un led segun el parametro de entrada color
143  *      0 - Verde
144  *      1 - Amarillo
145  *      2 - Rojo
146  *      3 - Azul
147  */
148  void
149  led_color_ON (unsigned short color)
150  {
151      if (color == 0) led_verde_ON ();
152      if (color == 1) led_rojo_ON ();
153      if (color == 2) led_azul_ON ();
154      if (color == 3) led_amarillo_ON ();
155  }
156
157  /*
158  *  Funcion que enciende apaga un led segun el parametro de entrada color
159  *      0 - Verde
160  *      1 - Amarillo
161  *      2 - Rojo
162  *      3 - Azul
163  */
164  void
165  led_color_OFF (unsigned short color)
166  {
167      if (color == 0) led_verde_OFF ();
168      if (color == 1) led_rojo_OFF ();
169      if (color == 2) led_azul_OFF ();
170      if (color == 3) led_amarillo_OFF ();
171  }
```

VI. leds.c

Archivo para utilización del LCD. Se anexa código a continuación:

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <math.h>
4  #include "LPC17xx.h"
5  #include "lcd.h"
6
7  uint8_t LCD_RS; // LOW: comando. HIGH: dato.
8  uint8_t LCD_RW; // LOW: escribir en LCD. HIGH: leer desde LCD.
9  uint8_t LCD_EN; // activacion por un pulso en HIGH.
10
11 uint8_t LCD_D4;
12 uint8_t LCD_D5;
13 uint8_t LCD_D6;
14 uint8_t LCD_D7;
15
16 /*
17  * Inicialziacion de LCD. Se admite cualquier pin del GPIO0 para funciones
18  * de control y cualquier pin del GPIO2 para datos.
19  * */
20
21 void
22 Lcd_Init (uint8_t rs, uint8_t rw, uint8_t enable, uint8_t d4, uint8_t d5,
23          uint8_t d6, uint8_t d7)
24 {
25     LCD_RS = rs;
26     LCD_RW = rw;
27     LCD_EN = enable;
28
29     LCD_D4 = d4;
30     LCD_D5 = d5;
31     LCD_D6 = d6;
32     LCD_D7 = d7;
33
34     // Configura todos los pines del LCD como salida
35     LPC_GPIO2->FIODIR |= ((1 << d4) | (1 << d5) | (1 << d6) | (1 << d7));
36     LPC_GPIO0->FIODIR |= ((1 << rs) | (1 << rw) | (1 << enable));
37
38     Lcd_CmdWrite (CLEAR_DISPLAY); // Borra datos previos del LCD
39     Lcd_CmdWrite (BACK_HOME); // Inicializa el LCD en modo 4-bit
40     Lcd_CmdWrite (_2_LINE_MATRIX); // Habilitado modo 5x7 para caracteres
41     Lcd_CmdWrite (D_ON_C_BLK); // Display OFF, Cursor ON
42     Lcd_CmdWrite (CLEAR_DISPLAY); // Borra datos previos del LCD
43     Lcd_CmdWrite (C_FIRST_LINE); // Fuerza al cursos a la posicion (1,1)
44
45 }
46
47 /*
48  * Envia comandos al LCD para ajustar su funcionamiento
49  * */
50
51 void
52 Lcd_CmdWrite (char cmd)
53 {
54     enviarNibble ((cmd >> 0x04) & 0x0F); // Enviar el nibble mas significativo
55     LPC_GPIO0->FIOPIN &= ~(1 << LCD_RS); // Enviar pulso BAJO al pin RS para seleccion de

```

```

56     MOD0 Comandos
    LPC_GPIO0->FIOPIN &= ~(1 << LCD_RW); // Enviar pulso BAJO al pin RW para establecer
        operacion de escritura
57     LPC_GPIO0->FIOPIN |= (1 << LCD_EN); // Generar un pulso HIGH-LOW en el pin EN
58     delay (10000);
59     LPC_GPIO0->FIOPIN &= ~(1 << LCD_EN);
60
61     delay (100000);
62
63     enviarNibble (cmd & 0x0F); // Enviar el nibble menos significativo
64     LPC_GPIO0->FIOPIN &= ~(1 << LCD_RS); // Enviar pulso BAJO al pin RS para seleccion de
        MOD0 Comandos
65     LPC_GPIO0->FIOPIN &= ~(1 << LCD_RW); // Enviar pulso BAJO al pin RW para establecer
        operacion de escritura
66     LPC_GPIO0->FIOPIN |= (1 << LCD_EN); // Generar un pulso HIGH-LOW en el pin EN
67     delay (10000);
68     LPC_GPIO0->FIOPIN &= ~(1 << LCD_EN);
69
70     delay (100000);
71 }
72
73 /*
74  * Envia los datos a ser mostrados en el LCD
75  */
76
77 void
78 Lcd_DataWrite (char dato)
79 {
80     enviarNibble ((dato >> 0x04) & 0x0F); // Enviar el nibble mas significativo
81     LPC_GPIO0->FIOPIN |= (1 << LCD_RS); // Enviar pulso ALTO al pin RS para seleccion de
        MOD0 Datos
82     LPC_GPIO0->FIOPIN &= ~(1 << LCD_RW); // Enviar pulso BAJO al pin RW para establecer
        operacion de escritura
83     LPC_GPIO0->FIOPIN |= (1 << LCD_EN); // Generar un pulso HIGH-LOW en el pin EN
84     delay (10000);
85     LPC_GPIO0->FIOPIN &= ~(1 << LCD_EN);
86
87     delay (100000);
88
89     enviarNibble (dato & 0x0F); // Enviar el nibble menos significativo
90     LPC_GPIO0->FIOPIN |= (1 << LCD_RS); // Enviar pulso ALTO al pin RS para seleccion de
        MOD0 Datos
91     LPC_GPIO0->FIOPIN &= ~(1 << LCD_RW); // Enviar pulso BAJO al pin RW para establecer
        operacion de escritura
92     LPC_GPIO0->FIOPIN |= (1 << LCD_EN); // Generar un pulso HIGH-LOW en el pin EN
93     delay (10000);
94     LPC_GPIO0->FIOPIN &= ~(1 << LCD_EN);
95
96     delay (100000);
97 }
98
99 /*
100  * Envia por los pines de datos los 4 bits que se envian como parametro
101  */
102
103 void
104 enviarNibble (char nibble)
105 {
106     // Limpiar datos anteriores
107     LPC_GPIO2->FIOPIN &= ~(((1 << LCD_D4) | (1 << LCD_D5) | (1 << LCD_D6)
108         | (1 << LCD_D7)));

```

```

109 LPC_GPIO2->FIOPIN |= (((nibble >> 0x00) & 0x01) << LCD_D4);
110 LPC_GPIO2->FIOPIN |= (((nibble >> 0x01) & 0x01) << LCD_D5);
111 LPC_GPIO2->FIOPIN |= (((nibble >> 0x02) & 0x01) << LCD_D6);
112 LPC_GPIO2->FIOPIN |= (((nibble >> 0x03) & 0x01) << LCD_D7);
113 }
114
115 /*
116  * Posiciona el cursor en la posicion definida por los parametros FILA y COLUMNA
117  */
118
119 void
120 Lcd_posCursor (uint8_t fila , uint8_t columna)
121 {
122     uint8_t i;
123     Lcd_CmdWrite (BACK_HOME); //Posicionamos cursos en el origen
124     switch (fila)
125     {
126         case (1):
127             Lcd_CmdWrite (C_FIRST_LINE); //Cursor en primera fila
128             break;
129         case (2):
130             Lcd_CmdWrite (C_SECOND_LINE); //Cursor en segunda fila
131             break;
132         default:
133             break;
134     }
135     for (i = 0; i < columna - 2; i++)
136     {
137         Lcd_CmdWrite (SHIFT_C_P_RIGHT); //Desplazamos el cursor a la columna indicada
138     }
139     //Deshabilitamos el cursor para evitar su desplazamiento cuando se envian datos al LCD
140     Lcd_CmdWrite (D_ON_C_OFF);
141 }
142
143
144 /*
145  * Transfiere el bloque de informacion que se pasa como parametro al LCD
146  */
147
148 void
149 Lcd_Send (char * texto)
150 {
151     for (int i = 0; texto[i] != 0; i++)
152     {
153         Lcd_DataWrite (texto[i]);
154         Lcd_CmdWrite (SHIFT_C_RIGHT);
155     }
156 }
157
158 /*
159  * Borra toda infomacion existente en el LCD
160  */
161 void
162 Lcd_Clear (void)
163 {
164     Lcd_CmdWrite (CLEAR_DISPLAY);
165 }
166
167 /*
168  * Ajusta el texto en el LCD y alternativamente puede generar un retardo.
169  */

```

```
170 void Lcd_Show (char *mensaje, uint8_t fila, uint8_t columna, uint8_t clear, uint32_t
    tiempo)
171 {
172     if (clear)
173     {
174         Lcd_Clear ();
175     }
176     if (fila != 0 && columna != 0)
177     {
178         Lcd_posCursor (fila, columna);
179     }
180     Lcd_Send (mensaje);
181     if (tiempo != 0)
182     {
183         delay (tiempo);
184     }
185 }
186
187
188 void
189 delay (uint32_t tiempo)
190 {
191     for (uint32_t i = 0; i < tiempo; i++)
192     ;
193 }
```