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 alternatico.
- Comunicación serie mediantes pines específicos de la placa de desarrollo.

IV. Esquemático del diseño elaborado

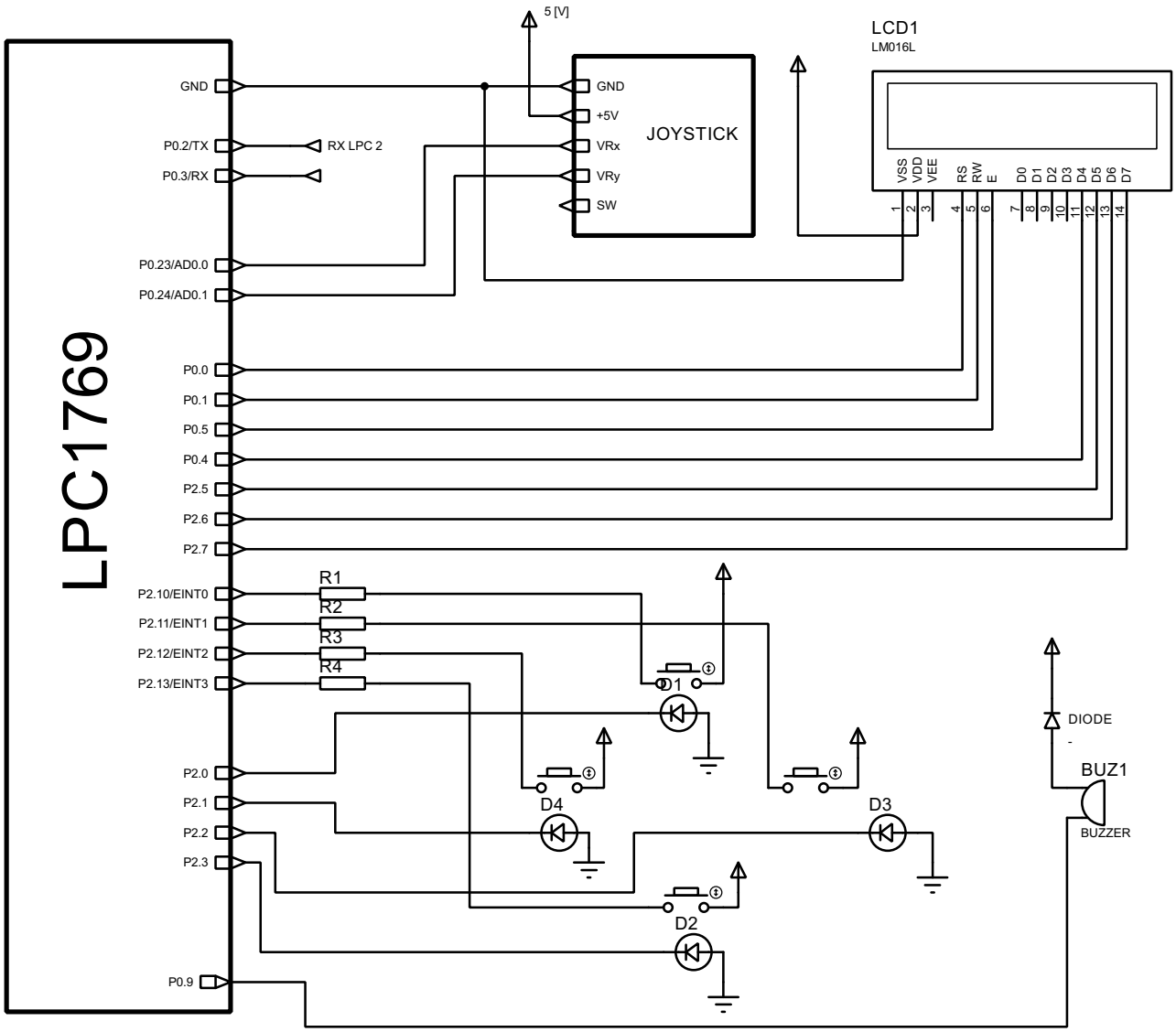

Figure 1: Diagrama circuital

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:

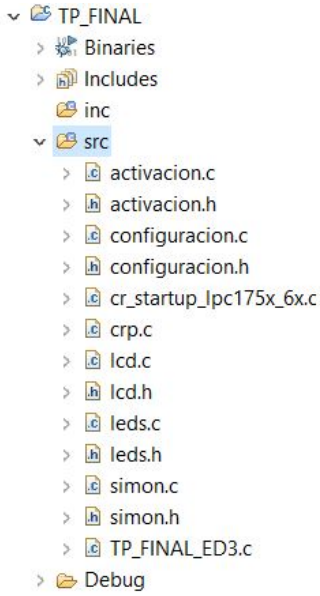

Figure 2: Archivos : códigos fuentes y headers

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 gano 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*

```
#ifdef __USE_CMSIS
#include "LPC17xx.h"
#endif

#include <cr_section_macros.h>

#include <stdio.h>
#include "configuracion.h"
#include "leds.h"
#include "simon.h"
#include "lcd.h"

int
main (void)
{
  int juego = 0, nivel = 0, exito = 0;

  clearPulsador ();
  clear_SELMODE ();
  perifericos_Init ();
  leds_Init ();
  Lcd_Init (LCD_RS, LCD_RW, LCD_EN, LCD_D4, LCD_D5, LCD_D6, LCD_D7);
  Lcd_Show ("SIMON", 1, 7, 0, 0);
  Lcd_Show ("—EDIII—", 2, 5, 0, 0);
  simon_Init (1000000);

  while (1)
    {
      Lcd_Show ("1) 4", 1, 1, 1, 0);
      Lcd_Show ("2) 8", 1, 10, 0, 0);
      Lcd_Show ("3) 12", 2, 1, 0, 0);
      Lcd_Show ("4) 16", 2, 10, 0, 0);
      while (!nivel)
  {
    nivel = getPulsador ();
    LPC_SC->EXTINT |= ~0;
    retardo (500000);
  }
       clearPulsador ();
       clear_SELMODE ();
       Lcd_Show ("1)Clasi", 1, 1, 1, 0);
       Lcd_Show ("2)Inver", 1, 10, 0, 0);
       Lcd_Show ("3)Avanz", 2, 1, 0, 0);
       Lcd_Show ("4)Multi", 2, 10, 0, 0);
       juego = 0;
       while (!juego)
  {
    juego = getPulsador ();
    LPC_SC->EXTINT |= ~0;
    retardo (500000);
  }

       switch (juego)
  {
  case (1):
    Lcd_Show ("MODO CLASICO", 1, 4, 1, 2500000);
    Lcd_Show ("3", 2, 9, 0, 2500000);
    Lcd_Show ("2", 2, 9, 0, 2500000);
    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:

```
#include "configuracion.h"

uint32_t volatile * const STCTRL = (uint32_t *) AddrSTCTRL;
uint32_t volatile * const STRELOAD = (uint32_t *) AddrSTRELOAD;

uint32_t RELOAD[4] =
  { 103370, 167404, 254545, 356759 }; //{Amarillo, Verde, Rojo, Azul}

/**
 * @brief  Configuracion de los pines a ser utilizados
 */
void
gpioConfig (void)
{
  LPC_PINCON->PINSEL0 |= (1 << 4);        //Funcion del Pin0.2  TX
  LPC_PINCON->PINSEL0 |= (1 << 6);        //FUncion del Pin0.3  RX
  LPC_PINCON->PINSEL1 |= (1 << 14);     //Funcion del Pin0.23  AD0.0
  LPC_PINCON->PINSEL1 |= (1 << 16);     //Funcion del Pin0.24  AD0.1
  LPC_PINCON->PINSEL1 |= (1 << 18);     //Funcion del Pin0.25  AD0.2
  LPC_PINCON->PINSEL4 |= (0b00 << 0);     //Funcion del Pin2.0 ESTA COMO GPIO, para PWM1
    .1 poner 0b10
  LPC_PINCON->PINSEL4 |= (0b00 << 2);     //Funcion del Pin2.1 ESTA COMO GPIO, para PWM1
    .2 poner 0b10
  LPC_PINCON->PINSEL4 |= (0b00 << 4);                //Funcion del Pin2.2 ESTA COMO GPIO
    , para PWM1.3 poner 0b10
  LPC_PINCON->PINSEL4 |= (0b00 << 6);              //Funcion del Pin2.3 ESTA COMO GPIO,
    para PWM1.4 poner 0b10
  LPC_PINCON->PINSEL4 |= (1 << 20);     //Funcion del Pin2.10: EINT0
  LPC_PINCON->PINSEL4 |= (1 << 22);     //Funcion del Pin2.11: EINT1
  LPC_PINCON->PINSEL4 |= (1 << 24);     //Funcion del Pin2.12: EINT2
  LPC_PINCON->PINSEL4 |= (1 << 26);     //Funcion del Pin2.12: EINT3
  LPC_PINCON->PINMODE1 |= (1 << 15);      //Pin0.23 entrada analogica
  LPC_PINCON->PINMODE1 |= (1 << 17);      //Pin0.24 entrada analogica
  LPC_PINCON->PINMODE1 |= (1 << 19);      //Pin0.25 entrada analogica
  LPC_PINCON->PINMODE4 |= (0b11 << 20);   //Pin2.10 Pull-down
  LPC_PINCON->PINMODE4 |= (0b11 << 22);   //Pin2.11 Pull-down
  LPC_PINCON->PINMODE4 |= (0b11 << 24);   //Pin2.12 Pull-down
  LPC_PINCON->PINMODE4 |= (0b11 << 26);   //Pin2.12 Pull-down
  //LPC_GPIO2->FIOMASK2 |= ();
  LPC_GPIO0->FIODIR |= (1 << 9);            //Pin0.9 como salida
  LPC_GPIO2->FIODIR |= (1 << 0 | 1 << 1 | 1 << 2 | 1 << 3); //Pines 2.0/2.1/2.2/2.3 como
    salida
  LPC_GPIO0->FIOMASK = 0;
  LPC_GPIO0->FIOCLR |= (1 << 9);            //Bajamos salida del tono
}

/**
 * @brief   Configura los TIMERs
 */
void
timerConfig (void)
{
```

```
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
90 void
91 uartConfig (void)
92 {
93   LPC_SC -> PCONP  |= 1<<3;   //Encendemos periferico
94   LPC_UART0->LCR |= (1 << 7);   //On DLAB
95   LPC_UART0->DLL = 130;       //PCLK 20Mhz -> 9600BR DLL en 130, DLM en 0
96   LPC_UART0->DLM = 0;
97   LPC_UART0->LCR &= ~(1 << 7);   //OFF DLAB, para poder seguir cfg
98   //LPC_UART0->FCR |= 1;        // 1 Con fifo, 0 (defecto) Sin fifo
99   LPC_UART0->LCR |= (0b11);   //8bits
100  LPC_UART0->IER |= 1;       //Interrupcion por recepcion de datos
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
160 void
161 setTono (uint32_t freq)
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*

```
#include "activacion.h"
#include "leds.h"
#include "simon.h"
#include <stdio.h>

#define TONO_A  51685
#define TONO_B  83702
#define TONO_C  127272
#define TONO_D  178379

uint8_t SEL_MODE = 0;
uint8_t PULSADOR = 0;
uint8_t SECUENCIA = 1;
uint8_t CONEXION = 1;
uint8_t ENABLE = 0;
/*Para indicar si ya se inicio comunicacion*/
uint8_t START_COM = 0;
/*Para indicar si recibi tono del otro jugador*/
uint8_t RCV_TONO = 0;
/*Para indicar si esta en curso el juego*/
uint8_t GAME_ON = 0;

char byte_to_send = 0;
char read_byte = 0;

char *buffer_recep;

/*
 * Empleado para seleccion del nivel de dificultad y el modo
 * de juego. Lanza el inicio del tono para el LED Verde y
 * activa el TIMER0 para la duracion del Tono.
 */

void
EINT0_IRQHandler (void)
{
  NVIC_DisableIRQ (EINT0_IRQn);
  while (debounce (LPC_GPIO2->FIOPIN >> 10 & 1))
    {
      retardo (100000);
    }

  LPC_SC->EXTINT |= 1;          //Bajo banderas de EINT0
  NVIC_EnableIRQ (EINT0_IRQn);
  if (SEL_MODE == 0)
    {
      PULSADOR = 1;
      SEL_MODE = 1;
      return;
    }
  if (SEL_MODE == 1)
    {
      PULSADOR = 1;
      SEL_MODE = 2;
      return;
    }
  if (SEL_MODE == 10)
    {
      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
        comunicacion, y otro para datos validos.
234  *Si un usuario pierde, el codigo 0xFF es enviado asi se corta la sesion multijugador
        iniciada ya que finalizo la misma
235  */
236
237 void
238 UART0_IRQHandler (void)
239 {
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
      arrancar a jugar*/
250       printf("Inicio de comunicacion multijugador \n");
251         byte_to_send = 2;
252         while( ( (LPC_UART0->LSR >> 5) & 1 ) ==0 );
253         LPC_UART0 -> THR = byte_to_send;
254         /*Enable para el juego en modo multijugador, y seteamos el flag que ya estamos
      conectados (START_COM) y arrancamos a jugar ( GAME_ON)*/
255         ENABLE = 1;
256         START_COM = 1;
257         GAME_ON = 1;
258
259     }else{
260       GAME_ON = 0;
261         ENABLE = 0;
262         START_COM = 0;
263          printf("Aun no se inicio comunicacion con otro jugador\n");
264
265     }
266
267   } else if ( START_COM  == 1 ){
268
269      /*Mientras estemos conectados, comprobamos que un dato sea valido, si no recibi el
      codigo de finalizacion*/
270     read_byte = LPC_UART0->RBR;
271     //printf("Recibi el byte: %d ",read_byte);
272     //printf("\n");
273     if (read_byte != 0xFF){
274
275       /*Lo tomo como dato valido*/
276       RCV_TONO = 1;
277       GAME_ON  = 1;
278
279     } else{
280       GAME_ON = 0; //Fin de juego
281
282     }
283
284   }
285 }
286
287
288 /*
289  * Conmuta el estado del pin asociado al BUZZER para generar
290  * la onda cuadrada del Tono.
291  */
292
293 void
294 SysTick_Handler (void)
295 {
296   LPC_GPIO0->FIOPIN = ~(LPC_GPIO0->FIOPIN >> 9 & 1) << 9;
297 }
298
```

```
/*
 * Comprueba en que posicion se encuentra el Stick y devuelve
 * el valor correspondiente a la posicion de este:
 *  Superior: -1
 *  Derecha:  -2
 *  Inferior:  -3
 *  Izquierda: -4
 *  Reposo:   -0
 */

uint8_t
movimiento_Stick (void)
{
  uint16_t digital_X, digital_Y;
  float X_vol, Y_vol;
  digital_X = (LPC_ADC->ADDR0 >> 4 & 0xFFF);
  digital_Y = (LPC_ADC->ADDR1 >> 4 & 0xFFF);
  X_vol = (digital_X * 3.3 / 4095.0);
  Y_vol = (digital_Y * 3.3 / 4095.0);
  //printf ("X_vol: %f\n", X_vol);
  //printf ("Y_vol: %f\n", Y_vol);

  if (Y_vol > 2.5)
    {
      return 1;
    }
  else if (X_vol > 2.5)
    {
      return 2;
    }
  else if (Y_vol < 0.7)
    {
      return 3;
    }
  else if (X_vol < 0.7)
    {
      return 4;
    }
  else return 0;
}

/*
 * Borra la variable de control SELMODE
 */

void
clear_SELMODE (void)
{
  SEL_MODE = 0;
}

/*
 * Devuelve el valor de la variable PULSADOR. Variable asociada
 * con los botonos, los valores son:
 *  Superior: -1
 *  Derecho:  -2
 *  Inferior: -3
 *  Izquierda:  -4
 *  Default:  -0
 *
 */
```

```
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
      inhibir lectura de buffer
458     }
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 codigo 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
        leds en simultaneo, para evitar confusiones)*/
112 void
113 simon_MultiSUCCESS (void)
114 {
115   int i;
116   for (i = 0; i < 2; i++)
117     {
118       set_SELMODE (0);
119       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
        determinado
235  *      de colores de manera inversa. Numero de turnos/colores en secuencia:
236  *          - Nivel 1: 4 turnos
237  *              - Nivel 2: 8 turnos
238  *          - Nivel 3: 12 turnos
239  *          - Nivel 4: 16 turnos
240  *   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
        determinado
324  *      de colores. En cada turno se determina de forma aleatoria si el jugado debe
        introducir
325  *      la secuencia de manera clasico o inversa. Se indicara mediante el LCD.
326  *      A su vez, se debe utilziar un Stick para activar la secuencia de colores en vez de
         los
327  *      clasicos botonos.
328  *      Numero de turnos/colores en secuencia:
329  *         - Nivel 1: 4 turnos
330  *             - Nivel 2: 8 turnos
331  *         - Nivel 3: 12 turnos
332  *         - Nivel 4: 16 turnos
333  *   Cada 4 turnos, disminuye el tiempo que se muestran los colores.
334  *   La funcion devuelve si el jugador ha tenido exito o no en *exito.
335  *
336  *
337  */
338
339 void
340 modo_3 (int nivel, int* exito)
341 {
342   int i, turno, fallo, turno_jugador, boton_pulsado, sentido, numero_turnos,
343       colores[MAX_TURNOS]; // Declaramos el vector de colores a acertar
344
345   boton_pulsado = 0;
346   turno = 1;
347   fallo = 0;
348   srand (time (NULL));      // Inicializamos semilla
349
350   numero_turnos = nivel * 4;
351
352   // Bucle que hace el juego hasta que se gana o se pierde
353   while (turno <= numero_turnos && fallo == 0)
354     {
355       boton_pulsado = 0;
356       NVIC_DisableIRQ (EINT0_IRQn);
357       NVIC_DisableIRQ (EINT1_IRQn);
358       NVIC_DisableIRQ (EINT2_IRQn);
359       NVIC_DisableIRQ (EINT3_IRQn);
```

```
360
361       sprintf (buffer, "%d", turno);
362       Lcd_Show ("Turno:", 1, 1, 1, 0);
363       Lcd_Show (buffer, 0, 0, 0, 0);
364       Lcd_Show ("-", 0, 0, 0, 0);
365       sprintf (buffer, "%d", numero_turnos);
366       set_SELMODE (10);
367       colores[turno - 1] = rand () % 4; // Nuevo color
368       Lcd_Show (buffer, 0, 0, 0, 2000000);
369
370       for (i = 0; i < turno; i++) // Imprimimos la secuencia de colores
371   {
372     set_SECUENCIA (1);
373     led_color_ON (colores[i]);
374     while (get_SECUENCIA ())
375       {
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   }
401         retardo (500000);
402
403         if ((boton_pulsado - 1) == colores[turno_jugador - 1]) turno_jugador++;
404         else fallo = 1;
405
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 secuancia 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
     comunicacion)
466  */
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 existosa, 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 recibi 0xFF por algun motivo, mal ingreso o mala replica, el otro usuario perdio
       y me indica que el juego finalizo*/
506     if ((int) get_user_tono() ==255){
507       printf("Usted gano, el player 1 ingreso mal \n");
508       *exito=1;
509       return;
510
511     }
512
513    //Leo byte/tono recibido por el otro ususario
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 iteracion,  en la proxima iteracion
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 dinamico 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 catcheando ingreso de usuario, tono a tono, asi comprobar que primero replique
        bien la secuencia*/
549      while (tono_a_replicar <= num_t){
550
551          clear_SELMODE ();
552         clearPulsador ();
553         boton_pulsado = getPulsador ();
554         while (!boton_pulsado)
555         {
556           boton_pulsado = getPulsador ();
557           retardo (1500);
558         }
559         set_SELMODE (10);
560         set_SECUENCIA (1);
561         led_color_ON (boton_pulsado - 1);
562         while (get_SECUENCIA ())
563         {
564         }
565         if ((boton_pulsado - 1) == colores[tono_a_replicar - 1]) {
566           printf("\n Metiste bien eltono capo \n");
567
568           tono_a_replicar++;
569         } else{
570
571           /*Perdi, envio 0xFF */
572           char error = 0xFF;
573           while( ( (LPC_UART0->LSR >> 5) & 1 ) ==0 );
574           LPC_UART0 -> THR = error;
575           retardo(500000);
576           /*Se setean las correspondientes flags*/
577           set_GAME_ON(0);
578           set_START_COM(0);
579           /*Deshabilito interrupciones*/
580           NVIC_DisableIRQ (UART0_IRQn);
581           /*Chau modo 4*/
582           *exito = 0;
583           return;
584         }
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
598         boton_pulsado = getPulsador ();
599         retardo (1500);
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
      3 (por eso el -1)
611
612       /*Envio tono al otro jugador*/
613      char tono_a_enviar = (char) boton_pulsado-1;
614
615      //printf("\nTe mande : %d ", tono_a_enviar);
616      while( ( (LPC_UART0->LSR >> 5) & 1 ) ==0 );
617      LPC_UART0 -> THR = tono_a_enviar;
618
619      /*Vuelvo a aumentar j, puntero a tonos en secuencia vigente*/
620      j++;
621
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.

```
#include "leds.h"
#include "configuracion.h"
#include "activacion.h"


/*
 *  Funcion que inicializa registros del GPIO2 para el uso de
 *  los leds de los botones
 */
void
leds_Init (void)
{
  LPC_GPIO2->FIODIR |= (1 << LED_Ve) | (1 << LED_Ro) | (1 << LED_Az)
      | (1 << LED_Am);
}

/*
 *  Funcion que enciende el led verde. Si se desea generar tambien el
 *  tono se debe setear el valor de SELMODE a 10
 */
void
led_verde_ON (void)
{
  LPC_GPIO2->FIOSET |= (1 << LED_Ve);
  if (get_SELMODE () == 10)
    {
      EINT0_IRQHandler ();
    }
}

/*
 *  Funcion que apaga el led verde
 */
void
led_verde_OFF (void)
{
  LPC_GPIO2->FIOCLR |= (1 << LED_Ve);
}

/*
 *  Funcion que enciende el led amarillo. Si se desea generar tambien el
 *  tono se debe setear el valor de SELMODE a 10
 */
void
led_amarillo_ON (void)
{
  LPC_GPIO2->FIOSET |= (1 << LED_Am);
  if (get_SELMODE () == 10)
    {
      EINT3_IRQHandler ();
    }
}

```

```
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:

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "LPC17xx.h"
#include "lcd.h"

uint8_t LCD_RS; // LOW: comando.  HIGH: dato.
uint8_t LCD_RW; // LOW: escribir en LCD.  HIGH: leer desde LCD.
uint8_t LCD_EN; // activacion por un pulso en HIGH.

uint8_t LCD_D4;
uint8_t LCD_D5;
uint8_t LCD_D6;
uint8_t LCD_D7;

/*
 * Inicialziacion de LCD. Se admite cualquier pin del GPIO0 para funciones
 * de control y cualquier pin del GPIO2 para datos.
 * */

void
Lcd_Init (uint8_t rs, uint8_t rw, uint8_t enable, uint8_t d4, uint8_t d5,
    uint8_t d6, uint8_t d7)
{
  LCD_RS = rs;
  LCD_RW = rw;
  LCD_EN = enable;

  LCD_D4 = d4;
  LCD_D5 = d5;
  LCD_D6 = d6;
  LCD_D7 = d7;

  // Configura todos los pines del LCD como salida
  LPC_GPIO2->FIODIR |= ((1 << d4) | (1 << d5) | (1 << d6) | (1 << d7));
  LPC_GPIO0->FIODIR |= ((1 << rs) | (1 << rw) | (1 << enable));

  Lcd_CmdWrite (CLEAR_DISPLAY);   // Borra datos previos del LCD
  Lcd_CmdWrite (BACK_HOME);     // Inicializa el LCD en modo 4-bit
  Lcd_CmdWrite (_2_LINE_MATRIX);        // Habilitado modo 5x7 para caracteres
  Lcd_CmdWrite (D_ON_C_BLK);            // Display OFF, Cursor ON
  Lcd_CmdWrite (CLEAR_DISPLAY);         // Borra datos previos del LCD
  Lcd_CmdWrite (C_FIRST_LINE);      // Fuerza al cursos a la posicion (1,1)

}

/*
 * Envia comandos al LCD para ajustar su funcionamiento
 */

void
Lcd_CmdWrite (char cmd)
{
  enviarNibble ((cmd >> 0x04) & 0x0F);  // Enviar el nibble mas significativo
  LPC_GPIO0->FIOPIN &= ~(1 << LCD_RS);  // Enviar pulso BAJO al pin RS  para seleccion de
```

```
    MODO Comandos
  LPC_GPIO0->FIOPIN &= ~(1 << LCD_RW);  // Enviar pulso BAJO al pin RW para establecer
    operacion de escritura
  LPC_GPIO0->FIOPIN |= (1 << LCD_EN);   // Generar un pulso HIGH-LOW en el pin EN
  delay (10000);
  LPC_GPIO0->FIOPIN &= ~(1 << LCD_EN);

  delay (100000);

  enviarNibble (cmd & 0x0F);            // Enviar el nibble menos significativo
  LPC_GPIO0->FIOPIN &= ~(1 << LCD_RS);  // Enviar pulso BAJO al pin RS  para seleccion de
    MODO Comandos
  LPC_GPIO0->FIOPIN &= ~(1 << LCD_RW);  // Enviar pulso BAJO al pin RW para establecer
    operacion de escritura
  LPC_GPIO0->FIOPIN |= (1 << LCD_EN);   // Generar un pulso HIGH-LOW en el pin EN
  delay (10000);
  LPC_GPIO0->FIOPIN &= ~(1 << LCD_EN);

  delay (100000);
}

/*
 * Envia los datos a ser mostrados en el LCD
 */

void
Lcd_DataWrite (char dato)
{
  enviarNibble ((dato >> 0x04) & 0x0F); // Enviar el nibble mas significativo
  LPC_GPIO0->FIOPIN |= (1 << LCD_RS);   // Enviar pulso ALTO al pin RS  para seleccion de
    MODO Datos
  LPC_GPIO0->FIOPIN &= ~(1 << LCD_RW);  // Enviar pulso BAJO al pin RW para establecer
    operacion de escritura
  LPC_GPIO0->FIOPIN |= (1 << LCD_EN);   // Generar un pulso HIGH-LOW en el pin EN
  delay (10000);
  LPC_GPIO0->FIOPIN &= ~(1 << LCD_EN);

  delay (100000);

  enviarNibble (dato & 0x0F);           // Enviar el nibble menos significativo
  LPC_GPIO0->FIOPIN |= (1 << LCD_RS);   // Enviar pulso ALTO al pin RS  para seleccion de
    MODO Datos
  LPC_GPIO0->FIOPIN &= ~(1 << LCD_RW);  // Enviar pulso BAJO al pin RW para establecer
    operacion de escritura
  LPC_GPIO0->FIOPIN |= (1 << LCD_EN);   // Generar un pulso HIGH-LOW en el pin EN
  delay (10000);
  LPC_GPIO0->FIOPIN &= ~(1 << LCD_EN);

  delay (100000);
}

/*
 * Envia por los pines de datos los 4 bits que se envian como parametro
 */

void
enviarNibble (char nibble)
{
  // Limpiar datos anteriores
  LPC_GPIO2->FIOPIN &= ~(((1 << LCD_D4) | (1 << LCD_D5) | (1 << LCD_D6)
      | (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 }
```