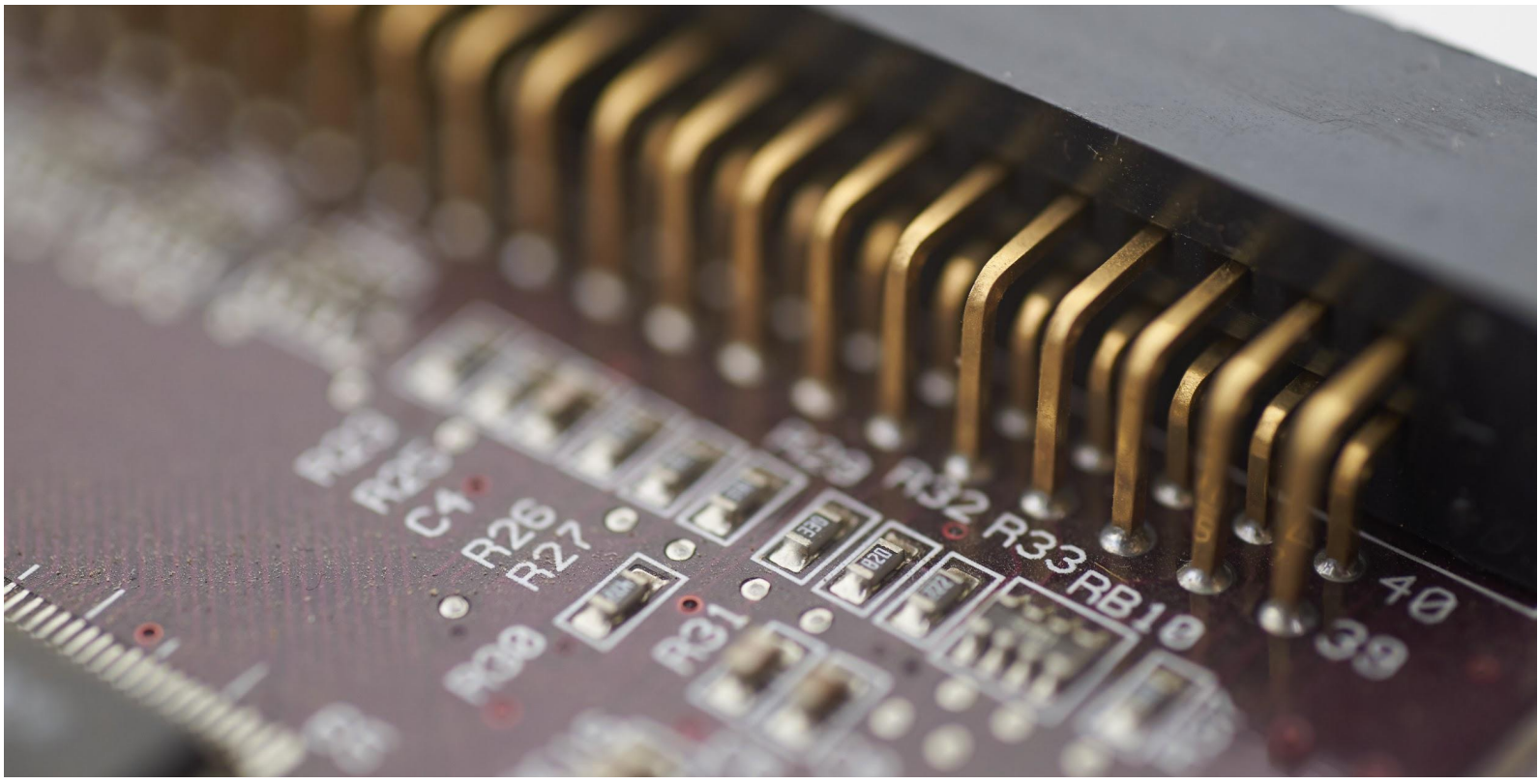
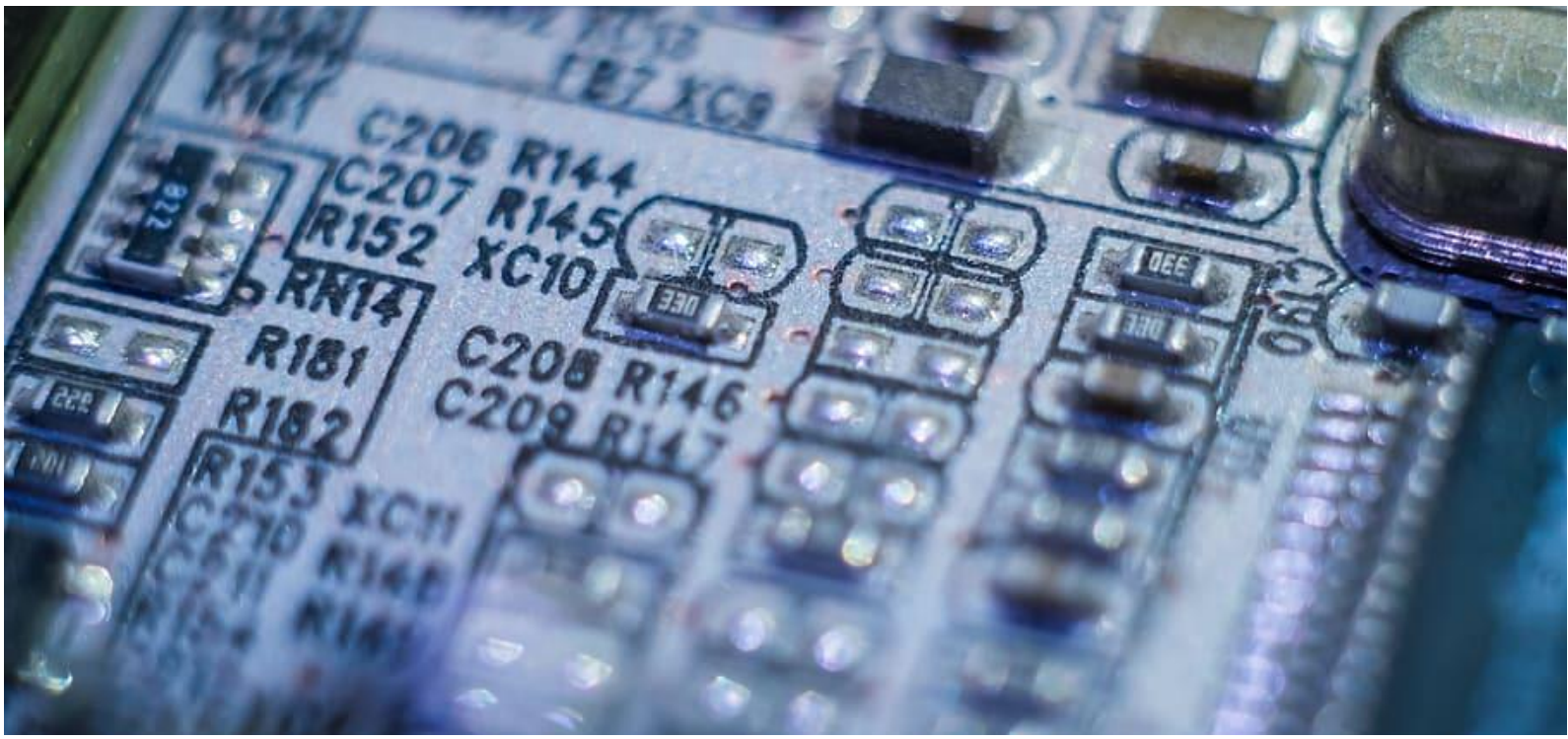
MICROCONTROLADORES

LABORATORIO

EQUIPO 2

José Manuel Aguilar Zamudio	A01381929
Diego Arnoldo Azuela Rosas	A01208345
Renato Arcos Guzmán	A01067221





CONTENIDO

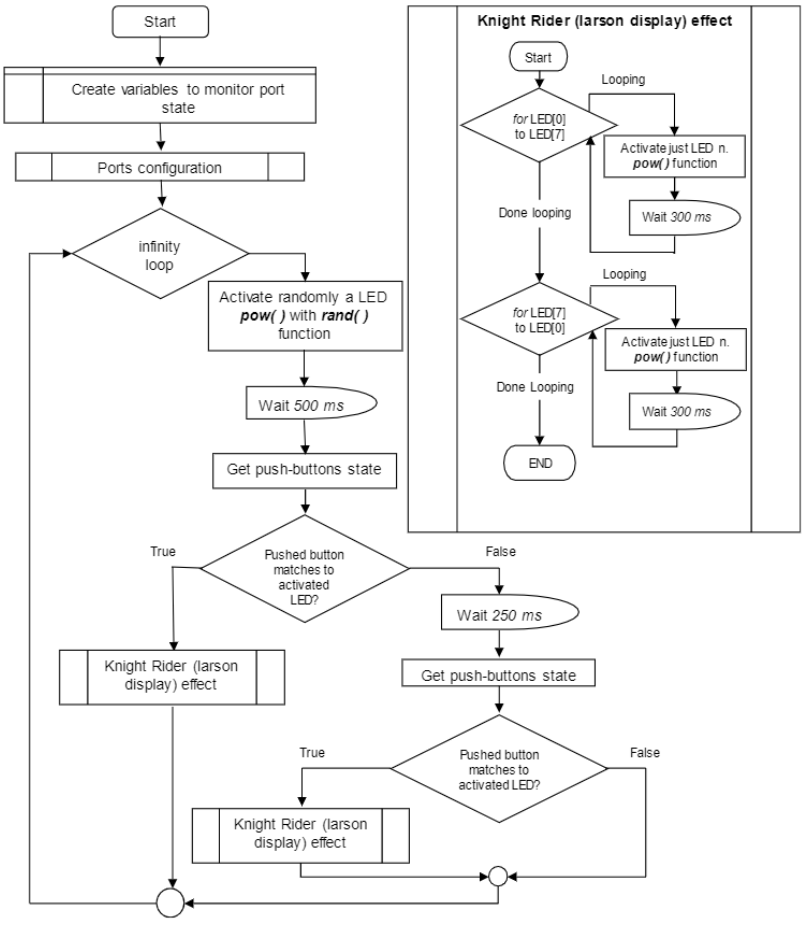
1. INTRODUCCIÓN

Whack-a-mole es un juego en el que los jugadores usan un mazo para golpear lunares de juguete, que aparecen en lugares aleatorios, para lanzarlos de vuelta a sus escondites. Se busca desarrollar el firmware del microcontrolador que ejecutará la lógica operativa básica detrás de este juego. Además, se llevarán a cabo pruebas experimentales de firmware para corregir posibles errores lógicos de su diseño, utilizando hardware de desarrollo, incluyendo la placa Curiosity y componentes adicionales.

2. OBJETIVO

Familiarizarse con las instrucciones básicas para la configuración y operación de un microcontrolador de propósito general de entradas y salidas (GPIO).

Asimismo, se busca que el firmware desarrollado siga la estructura del siguiente diagrama de flujo:



3. DESARROLLO

El firmware desarrollado cumple las funciones básicas del juego Whack-a-mole utilizando LEDs que funcionan como los topos que se encienden aleatoriamente, así como push buttons conectados frente a cada LED para simular el martillazo a cada topo.

El primer paso fue establecer en el ciclo PortsInit () a los puertos A y D como puertos digitales, así como apagar todos los puertos A (puertos a los que están conectados los LEDs) y encender los puertos D, a los cuales están conectados los push buttons.

Posterior a esto, se definieron las variables tipo char *posición_led* y *boton_status* las cuales cumplen la función de almacenar el valor hexadecimal del LED encendido y del botón presionado respectivamente.

Para lograr obtener un valor aleatorio entre el 0 y el 7, se utilizó la función *rand ()* procedente de la librería <math.h>. El valor obtenido de esta función se filtra por medio de una función switch-case en donde en cada uno de los casos, se asigna el valor hexadecimal de la potencialización de 2 al valor del caso en el que se encuentre a la variable *posición_led,* así como un retraso de un segundo para lograr un cambio prudente en el encendido de los LEDs junto con su función *break* para salir de la función switch-case cuando se termine de ejecutar.

Después de este ciclo switch case se iguala el valor de la variable *boton_status* al opuesto del valor lógico del botón, es decir, cuando este sea presionado el valor será dado a esta variable.

Finalmente, se tiene una función condicional *if* con la cual se compara el valor de las dos variables inicialmente declaradas las cuales en caso de resultar iguales, es decir que el valor del botón presionado sea el mismo que el valor del LED se iniciará una secuencia de encendido y apagado lineal continuo de cada LED de izquierda a derecha y de derecha con un tiempo de encendido de una décima de segundo; este es considerado el ciclo de ganador.

4. CÓDIGO

El código desarrollado para esta práctica se encuentra del lado izquierdo de esta tabla. Asimismo, se puede encontrar el archivo del mismo junto con el video demostrativo del juego en el siguiente link de github.

MAIN.C	DEVICE_CONFIG.H

//START PROJECT
#include "device_config.h"
#include <stdlib.h>
#include <stdio.h> /* printf, scanf, puts, NULL */
#include <math.h>
#include <time.h>
 	
//DIRECTIVES
#define ONE_SECOND 100
#define _XTAL_FREQ 1000000
#define DELAY_SWEEP 1000
//DATA TYPES
enum exponent{bbase = 2, limit = 8};
enum por_dir {output, input}; // output=0, input=1
enum por_ACDC{digital,analog};// digital=0, analog=1
enum resistor_state {set_ON,res_ON};//set_ON=0,resON =1
enum led_state {led_OFF, led_ON};// led_OFF, led_ON=1
enum button_state
	// PIC18F45K50 Configuration Bit Settings
	#include<xc.h>
	// 'C' source line config statements
	
	// CONFIG1L
	#pragma config PLLSEL = PLL4X // PLL Selection (4x clock multiplier)
	#pragma config CFGPLLEN = OFF // PLL Enable Configuration bit (PLL Disabled (firmware controlled))
	#pragma config CPUDIV = NOCLKDIV// CPU System Clock Postscaler (CPU uses system clock (no divide))
	#pragma config LS48MHZ = SYS24X4// Low Speed USB mode with 48 MHz system clock (System clock at 24 MHz, USB clock divider is set to 4)
	
	// CONFIG1H
	#pragma config FOSC = INTOSCIO // Oscillator Selection (Internal oscillator)
	#pragma config PCLKEN = ON // Primary Oscillator Shutdown (Primary oscillator enabled)

```
{pushed,no_pushed};//pushed=0, no_pushed=1
//ISR for high-priority
void __interrupt(high_priority) high_isr (void);
//ISR for low-priority
void __interrupt (low_priority) low_isr (void);
//FUNCTIONS DECLARATIONS
void portsInit(void);
typedef enum { F, T } boolean;



//MAIN SECTION
void main(void) {
   //DECLARAR VARIABLES
   portsInit(); //Debe ir antes de iniciar el ciclo
'while' para incializar los puertos correctamente
   char posicion_led;
   char boton_status;
   //CONFIGURATIONS
   while(1){
      //RANDOMIZE VARIABLE
      int numero = rand() %8; //Creamos variable
random para determinar que LED encender
      //CREATE CASES
      switch(numero){        //Función para
asignar la posición de LED por encender
         case 0:
            LATA = 0x01;
            posicion_led=0x01;
            __delay_ms (DELAY_SWEEP);
//DELAY FUNCION XC8 compiler
            break;
         case 1:
            LATA = 0x02;
            posicion_led=0x02;
            __delay_ms (DELAY_SWEEP);
//DELAY FUNCION XC8 compiler
            break;
         case 2:
            LATA = 0x04;
            posicion_led=0x04;
            __delay_ms (DELAY_SWEEP);
//DELAY FUNCION XC8 compiler
            break;
         case 3:
            LATA = 0x08;
            posicion_led=0x08;
            __delay_ms (DELAY_SWEEP);
//DELAY FUNCION XC8 compiler
            break;
         case 4:
            LATA = 0x10;
            posicion_led=0x10;
            __delay_ms (DELAY_SWEEP);
//DELAY FUNCION XC8 compiler
            break;
         case 5:
            LATA = 0x20;
            posicion_led=0x20;
```

```
#pragma config FCMEN = OFF      // Fail-Safe
Clock Monitor (Fail-Safe Clock Monitor disabled)
#pragma config IESO = OFF       //
Internal/External Oscillator Switchover
(Oscillator Switchover mode disabled)

// CONFIG2L
#pragma config nPWRTEN = OFF    // Power-up
Timer Enable (Power up timer disabled)
#pragma config BOREN = SBORDIS  //
Brown-out Reset Enable (BOR enabled in
hardware (SBOREN is ignored))
#pragma config BORV = 190       // Brown-out
Reset Voltage (BOR set to 1.9V nominal)
#pragma config nLPBOR = OFF     //
Low-Power Brown-out Reset (Low-Power
Brown-out Reset disabled)

// CONFIG2H
#pragma config WDTEN = OFF      // Watchdog
Timer Enable bits (WDT disabled in hardware
(SWDTEN ignored))
#pragma config WDTPS = 32768    // Watchdog
Timer Postscaler (1:32768)

// CONFIG3H
#pragma config CCP2MX = RC1     // CCP2
MUX bit (CCP2 input/output is multiplexed with
RC1)
#pragma config PBADEN = ON      // PORTB
A/D Enable bit (PORTB<5:0> pins are
configured as analog input channels on Reset)
#pragma config T3CMX = RC0      // Timer3
Clock Input MUX bit (T3CKI function is on RC0)
#pragma config SDOMX = RB3      // SDO
Output MUX bit (SDO function is on RB3)
#pragma config MCLRE = ON       // Master
Clear Reset Pin Enable (MCLR pin enabled;
RE3 input disabled)

// CONFIG4L
#pragma config STVREN = ON      // Stack
Full/Underflow Reset (Stack full/underflow will
cause Reset)
#pragma config LVP = ON         // Single-Supply
ICSP Enable bit (Single-Supply ICSP enabled if
MCLRE is also 1)
#pragma config ICPRT = OFF      // Dedicated
In-Circuit Debug/Programming Port Enable
(ICPORT disabled)
#pragma config XINST = OFF      // Extended
Instruction Set Enable bit (Instruction set
extension and Indexed Addressing mode
disabled)

// CONFIG5L
#pragma config CP0 = OFF        // Block 0 Code
Protect (Block 0 is not code-protected)
#pragma config CP1 = OFF        // Block 1 Code
```

```
            __delay_ms (DELAY_SWEEP);
//DELAY FUNCION XC8 compiler
            break;
         case 6:
            LATA = 0x40;
            posicion_led=0x40;
            __delay_ms (DELAY_SWEEP);
//DELAY FUNCION XC8 compiler
            break;
         case 7:
            LATA = 0x80;
            posicion_led=0x80;
            __delay_ms (DELAY_SWEEP);
//DELAY FUNCION XC8 compiler
            break;
      }
      boton_status = ~PORTD;
      //posicion_led = LATA;

      if (posicion_led == boton_status){
         for (int x=0;x<8;x++){
      //TRISAbits.RA0=led_ON;
      LATA = pow(2, x); // 2^0=1, 2, 4, 8, 16...
      __delay_ms (ONE_SECOND);
      //TRISAbits.TRISAx=led_OFF;;
         }
         for (int x=7;x>=0;x--){
            //TRISAbits.TRISBx=led_ON;
            LATA = pow(2, x); // 2^0=1, 2, 4, 8,
16...
            __delay_ms (ONE_SECOND);
            //TRISAbits.TRISBx=led_OFF;;
         }
      }
            // LATA = posicion_led;
   }
}


//PSEUDO CÓDIGO
void portsInit (void){//ports configuration
   //DECLARAR PUERTOS DIGITALES
   ANSELA = digital;
   ANSELD = digital;
   //DECLARAR LEDS
   TRISA=0x00; //
   TRISD=0xFF; //
}
```

```
Protect (Block 1 is not code-protected)
#pragma config CP2 = OFF        // Block 2 Code
Protect (Block 2 is not code-protected)
#pragma config CP3 = OFF        // Block 3 Code
Protect (Block 3 is not code-protected)

// CONFIG5H
#pragma config CPB = OFF        // Boot Block
Code Protect (Boot block is not code-protected)
#pragma config CPD = OFF        // Data
EEPROM Code Protect (Data EEPROM is not
code-protected)

// CONFIG6L
#pragma config WRT0 = OFF       // Block 0
Write Protect (Block 0 (0800-1FFFh) is not
write-protected)
#pragma config WRT1 = OFF       // Block 1
Write Protect (Block 1 (2000-3FFFh) is not
write-protected)
#pragma config WRT2 = OFF       // Block 2
Write Protect (Block 2 (04000-5FFFh) is not
write-protected)
#pragma config WRT3 = OFF       // Block 3
Write Protect (Block 3 (06000-7FFFh) is not
write-protected)

// CONFIG6H
#pragma config WRTC = OFF       //
Configuration Registers Write Protect
(Configuration registers (300000-3000FFh) are
not write-protected)
#pragma config WRTB = OFF       // Boot Block
Write Protect (Boot block (0000-7FFh) is not
write-protected)
#pragma config WRTD = OFF       // Data
EEPROM Write Protect (Data EEPROM is not
write-protected)

// CONFIG7L
#pragma config EBTR0 = OFF      // Block 0
Table Read Protect (Block 0 is not protected
from table reads executed in other blocks)
#pragma config EBTR1 = OFF      // Block 1
Table Read Protect (Block 1 is not protected
from table reads executed in other blocks)
#pragma config EBTR2 = OFF      // Block 2
Table Read Protect (Block 2 is not protected
from table reads executed in other blocks)
#pragma config EBTR3 = OFF      // Block 3
Table Read Protect (Block 3 is not protected
from table reads executed in other blocks)

// CONFIG7H
#pragma config EBTRB = OFF      // Boot Block
Table Read Protect (Boot block is not protected
from table reads executed in other blocks)

// #pragma config statements should precede
```

	project file includes. // Use project enums instead of #define for ON and OFF.

5. CAPTURAS