12 de septiembre de 2024

|  |  |  |  |
| --- | --- | --- | --- |
| **PROGRAMACIÓN DE MICROCONTROLADORES** | |  | TP1 |
| Facultad de Informática UNLP  Sistemas de Tiempo Real | |  |  |
|  | |  |  |
|  | |  |  |
|  |  |  | |
| * Melina Caciani Toniolo   melicaciani@gmail.com | |  | 02866/1 |
| * Joaquín Chanquía   joaquin.chanquia@alu.ing.unlp.edu.ar | |  | 02887/7 |
| * Mateo Emmanuel Larsen   larsenmateo.ml@gmail.com | |  | 02993/7 |
| * Gabriel Ollier   gabyollier@hotmail.com | |  | 02958/4 |
| * Franco Niderhaus   franconiderhaus@gmail.com   * Bruno Zanetti   bzanetti09@gmail.com | |  | 02976/6  02975/5 |

# EJERCICIO 1

* **Objetivo**

Utilice el entorno ISIS para abrir el diseño del circuito que se encuentra en la carpeta “Samples\VSM for PICMICRO\VSM for PIC16\PIC Doorbell”. En este esquema se observa un microcontrolador PIC 16F84; dos pulsadores conectados a los pines RA0 y RA1 como entradas; dos leds conectados a los pines RB4 y RB5 como salidas. Configure el entorno y realice un programa C que permita encender y apagar los leds de manera alternada cada 250 milisegundos utilizando la función “\_\_delay\_ms”. Cuando inicia el programa ambos leds deben estar encendidos. Luego de presionar alguno de los pulsadores deben comenzar a titilar. Nota: Observe en la barra inferior del simulador el tiempo de simulación para confirmar la precisión de la espera.

* **Circuito**

**Gráfico, Gráfico de cajas y bigotes

Descripción generada automáticamente**

* **Resolución**

Primero configuramos RA1 y RA0 como entradas, que corresponden a los pulsadores, y RB4 y RB5 como salidas, correspondientes a los leds.

Los leds debían estar encendidos hasta que se detecte que se pulsó alguno de los dos botones. Estos debían alternar entre prendido y apagado mientras alguno de los dos pulsadores se mantenga presionado. Al soltar, los leds deben quedar en el estado que se encuentren en ese momento.

Para controlar el parpadeo de los leds se utilizó un delay de 250ms.

* **Código**

#include <stdlib.h>

#include <stdio.h>

#include <htc.h>

#define \_XTAL\_FREQ 1000000

void iniPorts(){

TRISA = 0b00000011;

TRISB = 0;

PORTB = 0b00110000;

}

void main(){

iniPorts();

int start=0;

int presionado = 0;

while(1){

if ((!RA1) || (!RA0) && (!presionado)) {

start = 1;

presionado = 1;

}

if ((RA1) && (RA0) && (presionado)) {

start=0;

presionado = 0;

}

if (start == 1){

RB4^=1;

RB5^=1;

}

\_\_delay\_ms(250);

}

}

# EJERCICIO 2

* **Objetivo**

Rehaga el ejercicio anterior conservando la misma funcionalidad, pero modificando la implementación. Utilice el módulo del Timer0 con el prescaler para controlar el tiempo de espera y la interrupción para controlar el estado de los leds.

* **Resolución**

Para este ejercicio, en vez de utilizar la función delay utilizamos un Timer 0. Se utilizó un reloj interno como fuente para el Timer y elegimos un prescaler de 256. Calculamos el valor en el que se debe inicializar el Timer 0 para que ocurra una interrupción cada 250ms:

En la rutina de interrupción, primero verificamos si la interrupción fue causada por el Timer0 y de ser así, reiniciamos el flag correspondiente. Luego, si algún botón está presionado, alternamos el estado de los leds.

* **Código**

#include <stdlib.h>

#include <stdio.h>

#include <htc.h>

#define \_XTAL\_FREQ 1000000 // Definir la frecuencia del oscilador a 1 MHz

// Función para configurar el Timer0

void iniTimer0() {

T0CS = 0; // Seleccionar el reloj interno (Fosc/4) como fuente para el Timer0

PSA = 0; // Asignar el prescaler al Timer0

PS2 = 1; // Configurar el prescaler en 1:256 (PS2:PS1:PS0 = 111)

PS1 = 1;

PS0 = 1;

TMR0 = 12; // Inicializar el registro del Timer0 en 12

T0IE = 1; // Habilitar la interrupción del Timer0

GIE = 1; // Habilitar interrupciones globales (permite que las interrupciones ocurran)

}

// Función para configurar los puertos

void iniPorts() {

TRISA = 0b00000011; // Configurar RA0 y RA1 como entradas (bits 0 y 1 en 1)

TRISB = 0; // Configurar todos los pines de PORTB como salidas

PORTB = 0b00110000; // Inicializar PORTB: RB4 y RB5 en alto (LEDs encendidos)

}

// Variables globales para controlar el estado de los LEDs

volatile int start = 0; // Indica si los LEDs deben parpadear

volatile int presionado = 0; // Indica si un botón fue presionado

void main() {

iniPorts(); // Inicializar los puertos

iniTimer0(); // Inicializar el Timer0

// Bucle principal

while (1) {

// Si alguno de los botones (RA0 o RA1) es presionado y no se ha registrado un presionado previo...

if ((!RA1 || !RA0) && !presionado) {

start = 1; // Habilitar el parpadeo de los LEDs

presionado = 1; // Marcar que un botón fue presionado

}

// Si ambos botones (RA0 y RA1) están sueltos y un botón fue presionado previamente...

if (RA1 && RA0 && presionado) {

start = 0; // Detener el parpadeo de los LEDs

presionado = 0; // Marcar que los botones fueron soltados

}

}

}

// Rutina de Interrupción

void interrupt ISR() {

if (T0IF) { // Verificar si la interrupción fue causada por el Timer0

T0IF = 0; // Limpiar el flag de interrupción del Timer0

if (start == 1) { // Si el parpadeo está habilitado...

RB4 ^= 1; // Alternar (toggle) el estado del pin RB4

RB5 ^= 1; // Alternar (toggle) el estado del pin RB5

}

}

}

**EJERCICIO 3**

* **Objetivo**

Utilice el entorno ISIS para abrir el diseño del circuito que se encuentra en la carpeta “Samples\VSM for PICMICRO\VSM for PIC16\PIC ADC Example”. En este esquema se observa un microcontrolador PIC 16F877; una resistencia variable (que simula un transductor) conectada a través del pin AN0; dos Latchs 74LS373 (multiplexados) cuyas entradas comparten el puerto B y que son controlados a través de los pines RD6 y RD7 del puerto D; cuatro displays de 7 segmentos hexadecimales conectados a las salidas de los latchs. Configure el entorno y realice un programa en C que permita capturar valores analógicos realizando “polling” y muestre los valores digitales (en hexadecimal) en los displays de 7 segmentos. Tenga en cuenta que: 1. debe configurar el pin AN0 como entrada analógica y como tensión de referencia el mismo del microcontrolador (VCC). 2. los latchs (74LS373) funcionan de forma multiplexada por lo que requieren la activación/desactivación en el pin LE para que los datos de entrada queden en la salida. 3. que los displays muestran los valores en hexadecimal, por ejemplo, si el valor a mostrar es 1510 o 11112 mostrará la letra “F”.

* **Circuito**

Diagrama, Esquemático

Descripción generada automáticamente

* **Resolución**

Para comenzar, definimos RD6 y RD7 como LATCH1 y LATCH2. RD6 controla la activación del primer latch, y este, los displays de los dos dígitos menos significativos. RD7 controla la activación del segundo latch, y este, los displays de los dos dígitos más significativos.

Configuramos el ADC y elegimos la justificación a derecha, por lo que debemos activar el bit ADFM. Luego de esto debemos calcular el valor almacenado en ADRE sumando los dos bytes ADRESH (dezplazado 8 bits) y ADRESL para obtener el valor de 16 bits. El ADC tiene una resolución de 10 bits, por lo que aumentar el potenciómetro al 100%, debería mostrar 3FF en los displays (1023).

Estos 16 bits se dividen en 2 bytes, el byte más significativo se envía al PORTB y se envía un flanco de subida al latch 2. Realizamos lo mismo con el byte menos significativo, pero esta vez activando el latch 1.

* **Código**

#include <htc.h>

#define \_XTAL\_FREQ 4000000 // Frecuencia del oscilador (4 MHz)

#define LATCH1 RD6 // Controla el primer latch (RD6)

#define LATCH2 RD7 // Controla el segundo latch (RD7)

// Función para inicializar el ADC

void initADC() {

ADCON0 = 0x01; // Seleccionar canal AN0 y encender el ADC

ADCON1 = 0x0E; // Configurar AN0 como analógico, VREF = VCC

ADFM = 1; // Configurar justificación derecha

}

// Función para leer el valor analógico desde AN0

unsigned int readADC() {

\_\_delay\_us(20); // Pequeña espera para la adquisición

GO\_nDONE = 1; // Iniciar la conversión ADC

while(GO\_nDONE); // Esperar a que termine

return (ADRESH << 8) | ADRESL; // Devolver el valor de 10 bits

}

// Función para inicializar los puertos

void initPorts() {

TRISA = 0x01; // AN0 como entrada

TRISB = 0x00; // Puerto B como salida (para enviar los datos a los latchs)

TRISD = 0x00; // Puerto D como salida (para controlar los latchs)

PORTB = 0x00; // Limpiar puerto B

PORTD = 0x00; // Limpiar puerto D

}

// Función para actualizar los displays con un valor de 16 bits

void updateDisplay(unsigned int value) {

unsigned char digit;

// Mostrar los dígitos más significativos en el display 1 y 2

digit = (value >> 8) & 0xFF; // Más significativos 8 bits

PORTB = digit; // Enviar el dígito al puerto B

LATCH2 = 1; LATCH2 = 0; // Activar el primer latch

\_\_delay\_ms(1); // Esperar para estabilidad

// Mostrar los dígitos menos significativos en el display 3 y 4

digit = (value) & 0xFF; // Siguiente dígito

PORTB = digit;

LATCH1 = 1; LATCH1 = 0; // Activar el segundo latch

\_\_delay\_ms(1); // Esperar para estabilidad

}

// Programa principal

void main() {

unsigned int adcValue;

initPorts(); // Inicializar puertos

initADC(); // Inicializar ADC

while(1) {

adcValue = readADC(); // Leer valor del ADC

updateDisplay(adcValue); // Mostrar valor en los displays

\_\_delay\_ms(500); // Esperar antes de la siguiente lectura

}

}