```
/*
 * File:   dial_tone_mian.c
 * Author: Reckless Experimentation Audio
 *
 * Created on February 24, 2018, 9:20 AM
 */

#include <stdio.h>
#include <stdlib.h>

// PIC18F24K22 Configuration Bit Settings

// 'C' source line config statements

// CONFIG1H
#pragma config FOSC = INTIO67   // Oscillator Selection bits (Internal oscillator block)
#pragma config PLLCFG = OFF     // 4X PLL Enable (Oscillator used directly)
#pragma config PRICLKEN = ON    // Primary clock enable bit (Primary clock enabled)
#pragma config FCMEN = OFF      // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor
disabled)
#pragma config IESO = OFF       // Internal/External Oscillator Switchover bit (Oscillator Switchover
mode disabled)
// CONFIG2L
#pragma config PWRTEN = ON      // Power-up Timer Enable bit (Power up timer enabled)
#pragma config BOREN = SBORDIS  // Brown-out Reset Enable bits (Brown-out Reset enabled in hardware
only (SBOREN is disabled))
#pragma config BORV = 190       // Brown Out Reset Voltage bits (VBOR set to 1.90 V nominal)
// CONFIG2H
#pragma config WDTEN = OFF      // Watchdog Timer Enable bits (Watch dog timer is always disabled.
SWDTEN has no effect.)
#pragma config WDTPS = 32768    // Watchdog Timer Postscale Select bits (1:32768)
// CONFIG3H
#pragma config CCP2MX = PORTC1  // 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 CCP3MX = PORTB5  // P3A/CCP3 Mux bit (P3A/CCP3 input/output is multiplexed with RB5)
#pragma config HFOFST = ON      // HFINTOSC Fast Start-up (HFINTOSC output and ready status are not
delayed by the oscillator stable status)
#pragma config T3CMX = PORTC0   // Timer3 Clock input mux bit (T3CKI is on RC0)
#pragma config P2BMX = PORTB5   // ECCP2 B output mux bit (P2B is on RB5)
#pragma config MCLRE = INTMCLR  // MCLR Pin Enable bit (RE3 input pin enabled; MCLR disabled)
// CONFIG4L
#pragma config STVREN = ON      // Stack Full/Underflow Reset Enable bit (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 XINST = OFF      // Extended Instruction Set Enable bit (Instruction set extension and
Indexed Addressing mode disabled (Legacy mode))
// CONFIG5L
#pragma config CP0 = OFF        // Code Protection Block 0 (Block 0 (000800-001FFFh) not code-
protected)
#pragma config CP1 = OFF        // Code Protection Block 1 (Block 1 (002000-003FFFh) not code-
protected)
// CONFIG5H
#pragma config CPB = OFF        // Boot Block Code Protection bit (Boot block (000000-0007FFh) not
code-protected)
#pragma config CPD = OFF        // Data EEPROM Code Protection bit (Data EEPROM not code-protected)
// CONFIG6L
#pragma config WRT0 = OFF       // Write Protection Block 0 (Block 0 (000800-001FFFh) not write-
protected)
#pragma config WRT1 = OFF       // Write Protection Block 1 (Block 1 (002000-003FFFh) not write-
protected)
// CONFIG6H
#pragma config WRTC = OFF       // Configuration Register Write Protection bit (Configuration registers
(300000-3000FFh) not write-protected)
#pragma config WRTB = OFF       // Boot Block Write Protection bit (Boot Block (000000-0007FFh) not
write-protected)
#pragma config WRTD = OFF       // Data EEPROM Write Protection bit (Data EEPROM not write-protected)
// CONFIG7L
#pragma config EBTR0 = OFF      // Table Read Protection Block 0 (Block 0 (000800-001FFFh) not
protected from table reads executed in other blocks)
#pragma config EBTR1 = OFF      // Table Read Protection Block 1 (Block 1 (002000-003FFFh) not
protected from table reads executed in other blocks)
// CONFIG7H
```

```
#pragma config EBTRB = OFF      // Boot Block Table Read Protection bit (Boot Block (000000-0007FFh)
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.

#include <xc.h>
#include <pic18f24k22.h>


#define gate    !PORTAbits.RA2          // define gate input
#define record  !PORTAbits.RA7          // define record input
#define play    !PORTAbits.RA6          // define play input

unsigned char read_keypad(void);        // function for reading the kepad

/*
 *
 */
int main(int argc, char** argv)     // start of main function
{
    //
    //  variable declarations
    //
    unsigned char i,j;              // iterator variables
    unsigned char number[32];       // speed dial number
    unsigned char button;           // button that is pressed
    unsigned char rec_position=0;   // position in array for speed dial
    unsigned char play_position=0;  // position in array for speed dial
    unsigned char marked=0;         // used in recording speed dial

    //
    // chip setup
    //
    ANSELA = 0b00000011;     // set port A to digital expect of 0 and 1
    TRISA  = 0b11000111;     // setup port A IO
    LATA   = 0b00011000;     // turn off tone disable

    ANSELB = 0x00;           // Port B analog inputs off
    TRISB  = 0b00001111;     // setup port B IO for keypad
    LATB   = 0b11110000;     // set all outputs to positive
    INTCON2bits.RBPU = 0;    // Enable weak pull ups on port B
    WPUB   = 0b00001111;     // turn on weak pull ups for inputs

    TRISC  = 0b00000000;     // port C all outputs
    LATC   = 0b11111111;     // all outputs on

    ADCON1 = 0b00000000;     // ADC connected to VDD and VSS
    ADCON2 = 0b00101111;     // left justified, 12 TAD AQ, internal oscillator
    ADCON0 = 0b00000001;     // ADC on

    T2CON  = 0b01111110;     // timer 2 on  1:16 pre-scale, 1:16 post scale

    //
    // main loop
    //
    while(1)                        // start 0f main loop
    {
        button = read_keypad();     // read the keypad

        if(gate)                    // if gate input
        {
            ADCON0 = 0b00000011;    // start a ADC conversion on channel zero
            while(ADCON0bits.GO);   // while the conversion is running, do nothing
            button = ADRESH;        // copy the 8 bit result
            button >>= 4;           // bit shift by 4, to divide by 16
            button++;               // add one
        }

        if(record)                                  // if record input
        {
            if(button !=0)                          // if a button has been pressed
            {
                if(marked == 0)                     // if this button has not been recorded
                {
                    if(rec_position<32)             // if inside the bounds of the speed dial array
                    {
                        if(rec_position == 0)       // if the position is zero
                            for(i=0;i<32;i++)       // for the entire array
                                number[i]=0;        // set its value to zero

                        number[rec_position]= button;   // record the button pushed
                        rec_position++;                 // increment the position in the record array
                    }
                    marked = 1;                     // mark that this input has been recorded
                }                                   // this is needed because the program will loop
            }                                       // multiple times per input
            else                        // no button is pressed
            {
                marked = 0;             // reset to catch the next input
            }
        }
        else                            // if the record input is not present
        {
            marked = 0;                 // reset to catch the next input
            rec_position=0;             // reset record position

            if(play)                                        // if the play input is present
            {
                if((play_position & 0x01) == 0)             // see if play position is even
                {                                           // if it is
                    button = number[play_position>>1];      // set the button pressed to the recorded
value
                }
                else                                        // if the play position is odd
                {
                    button = 0;                             // turn off the sound
                }
                if(T2CONbits.TMR2ON==0)     // if timer 2 is off
                {
                    ADCON0 = 0b00000111;    // start a ADC conversion on channel one
                    while(ADCON0bits.GO);   // while the conversion is running, do nothing
                    PR2 = 255 - ADRESH;     // copy the result to period register two
                    TMR2 = 0;               // reset timer 2
                    PIR1bits.TMR2IF=0;      // reset timer 2 interrupt
                    T2CONbits.TMR2ON=1;     // turn on timer 2
                }
                if(PIR1bits.TMR2IF)         // if timer 2 interrupt
                {
                    T2CONbits.TMR2ON=0;     // turn off timer 2
                    play_position++;        // increment play position
                }
                if(play_position>63)        // if play position has reached the end
                    play_position=63;       // hold position at the end
            }
            else                            // if not play input
            {
                play_position=0;            // reset play position
            }
        }


        switch(button)              // see which button was pressed
        {
            case 0:                 // no button pressed
                LATC = 0b11111111;  // turn off the sound
                break;              // exit from switch statement
            case 1:                 // if button one pressed
                LATC = 0b01111110;  // set column 1 row 1
                break;              // exit from switch statement
            case 2:                 // if button 2 pressed
                LATC = 0b01111101;  // set column 1 row 2
                break;              // exit from switch statement
            case 3:                 // if button 3 pressed
                LATC = 0b01111011;  // set column 1 row 3
                break;              // exit from switch statement
            case 4:                 // if button 4 pressed
                LATC = 0b01110111;  // set column 1 row 4
                break;              // exit from switch statement
            case 5:                 // if button 5 pressed
                LATC = 0b10111110;  // set column 2 row 1
                break;              // exit from switch statement
            case 6:                 // if button 6 pressed
                LATC = 0b10111101;  // set column 2 row 2
                break;              // exit from switch statement
            case 7:                 // if button 6 pressed
                LATC = 0b10111011;  // set column 2 row 3
                break;              // exit from switch statement
            case 8:                 // if button 8 pressed
                LATC = 0b10110111;  // set column 2 row 4
                break;              // exit from switch statement
            case 9:                 // if button 9 pressed
                LATC = 0b11011110;  // set column 3 row 1
                break;              // exit from switch statement
            case 10:                // if button 10 pressed
                LATC = 0b11011101;  // set column 3 row 2
                break;              // exit from switch statement
            case 11:                // if button 11 pressed
                LATC = 0b11011011;  // set column 3 row 3
                break;              // exit from switch statement
            case 12:                // if button 12 pressed
                LATC = 0b11010111;  // set column 3 row 4
                break;              // exit from switch statement
            case 13:                // if button 13 pressed
                LATC = 0b11101110;  // set column 4 row 1
                break;              // exit from switch statement
            case 14:                // if button 14 pressed
                LATC = 0b11101101;  // set column 4 row 2
                break;              // exit from switch statement
            case 15:                // if button 15 pressed
                LATC = 0b11101011;  // set column 4 row 3
                break;              // exit from switch statement
            case 16:                // if button 16 pressed
                LATC = 0b11100111;  // set column 4 row 4
                break;              // exit from switch statement
        }
    }

    return (EXIT_SUCCESS);      // end of main, we never get here
}

//
//  keypad reading function
//
unsigned char read_keypad(void) // no inputs, returns a char
{
    unsigned char temp;         // temporary storage variable.

    LATB = 0b01110000;          // pull down first column
    asm("NOP");                 // waste some time
    asm("NOP");
    asm("NOP");
    temp = PORTB & 0b00001111;  // copy the lower nibble of port B
    if(temp == 0b00001110)      // if row 1
        return 1;               // return with button 1
    if(temp == 0b00001101)      // if row 2
        return 2;               // return with button 2
    if(temp == 0b00001011)      // if row 3
        return 3;               // return with button 3
    if(temp == 0b00000111)      // if row 4
        return 4;               // return with button 4

    LATB = 0b10110000;          // pull down second column
    asm("NOP");                 // waste some time
    asm("NOP");
    asm("NOP");
    temp = PORTB & 0b00001111;  // copy the lower nibble of port B
    if(temp == 0b00001110)      // if row 1
        return 5;               // return with button 5
    if(temp == 0b00001101)      // if row 2
        return 6;               // return with button 6
    if(temp == 0b00001011)      // if row 3
```

```
        return 7;               // return with button 7
    if(temp == 0b00000111)      // if row 4
        return 8;               // return with button 8

    LATB = 0b11010000;          // pull down third column
    asm("NOP");                 // waste some time
    asm("NOP");
    asm("NOP");
    temp = PORTB & 0b00001111;  // copy the lower nibble of port B
    if(temp == 0b00001110)      // if row 1
        return 9;               // return with button 9
    if(temp == 0b00001101)      // if row 2
        return 10;              // return with button 10
    if(temp == 0b00001011)      // if row 3
        return 11;              // return with button 11
    if(temp == 0b00000111)      // if row 4
        return 12;              // return with button 12

    LATB = 0b11100000;          // pull down 4th column
    asm("NOP");                 // waste some time
    asm("NOP");
    asm("NOP");
    temp = PORTB & 0b00001111;  // copy the lower nibble of port B
    if(temp == 0b00001110)      // if row 1
        return 13;              // return with button 13
    if(temp == 0b00001101)      // if row 2
        return 14;              // return with button 14
    if(temp == 0b00001011)      // if row 3
        return 15;              // return with button 15
    if(temp == 0b00000111)      // if row 4
        return 16;              // return with button 16

    return 0;                   // return with no buttons pressed

}
```