Keypad

Source:https://docs.google.com/document/d/1dD15-CHEM5IJJgp71ztsqfjzrRPhDhHK9A905XirVLs/edit?tab=t.0#heading=h.sspkfhxq0hwc

Library Overview

The Keypad Library provides a complete interface for reading input from a 4x4 matrix membrane keypad and implementing secure PIN entry functionality. The library handles all low-level hardware interfacing, including row scanning, column reading, debouncing, and PIN validation.

Key Features

- **Hardware Keypad Interface:** Direct GPIO control of 4x4 matrix keypad
- **PIN Entry System:** Secure 4-digit PIN validation with configurable code
- **Automatic Debouncing:** Built-in hardware and software debouncing (1ms + 50ms)
- **Simple API:** Easy-to-use functions for keypad scanning and PIN management
- **No External Components:** Uses internal pull-up resistors (no external resistors required)

Use Cases

- Security access control systems
- PIN-protected RFID readers

- Password entry interfaces
- Numeric input for embedded systems
- Menu navigation systems

Function Summary

Function	Description
initKeyPad(void)	Configures keypad hardware pins (rows RB15,14,11, and 10 as outputs, columns RA3-0 as inputs with pull-ups) Inputs: none Returns: none Usage: initKeyPad(); // Call once during system initialization

readPins(void)	Scans all keypad rows and returns the ASCII character of the pressed key or '\0' if no key is pressed
	Inputs: none
	Returns: unsigned char the ASCII character on key pad (0-9, A-F), and '\0' if nothing was pressed
	Usage:
	unsigned char key = readPins();
	if (key != '\0') { *// Key was pressed*
	printf("Key pressed: %c\n", key); }

determineKey(void)	Reads column input pins to determine which column (0-3) is pulled LOW by a key press.
	Inputs: none
	Returns: integer column number(0-3) if a key is pressed in that column else return 15 in hexadecimal if no column is low/not pressed
	Usage:
	// typically called internally by readPins()
	int col = determineKey();
	if (col < 0xF) {
	printf("Column %d is pressed\n", col);
	}
	
	Note:This is an internal helper function typically called by readPins(). Only one column should be LOW at a time for proper operation. Multiple simultaneous key presses may produce ambiguous results.

initPinSystem(void)	Initializes the PIN entry system by calling initKeyPad() and setting pinAccepted flag to 0. Inputs: none Returns: none

	Usage:
	int main(void) {
	initPinSystem(); // call once at startup
	
	while(1) {
	// Main program loop
	}
	}

processKeyPress(unsigned char key)	Stores enter digits, validates the 4-digit PIN after completion, and sets pinAccepted flag.
	Inputs: unsigned char key(ASCII characters it accepts only 0-9 it ignores the A-F pads)
	Returns: none
	Usage:
	unsigned char key = readPins();
	if (key != '\0') {
	processKeyPress(key);
	if (getPinAccepted()) {
	// pin was correct
	redLED_OFF();
	}
	}
	
	Note it first checks if the key is digit(0-9)and buffer is not full if so then stores the it into enteredPIN array once the buffer array is full it goes to call validatePIN() if correct change the pin accepted flag to1(correct pin entered) else it stays at 0 and there's a delay of 1 second before clearPIN() is called so can't force it with many entries.

validatePIN(void)	Compares the entered PIN with "1234" and returns 1 if correct, 0 if incorrect
	Inputs: none
	Returns: integer (1) if its correct pin and (0) if its the incorrect pin
	Usage:
	// typically called internally by processKeyPress()
	if (validatePIN()) {
	printf("PIN is correct!\n");
	} else {
	printf("PIN is incorrect!\n");
	}
	Note: it strcmp() function to read the array keys

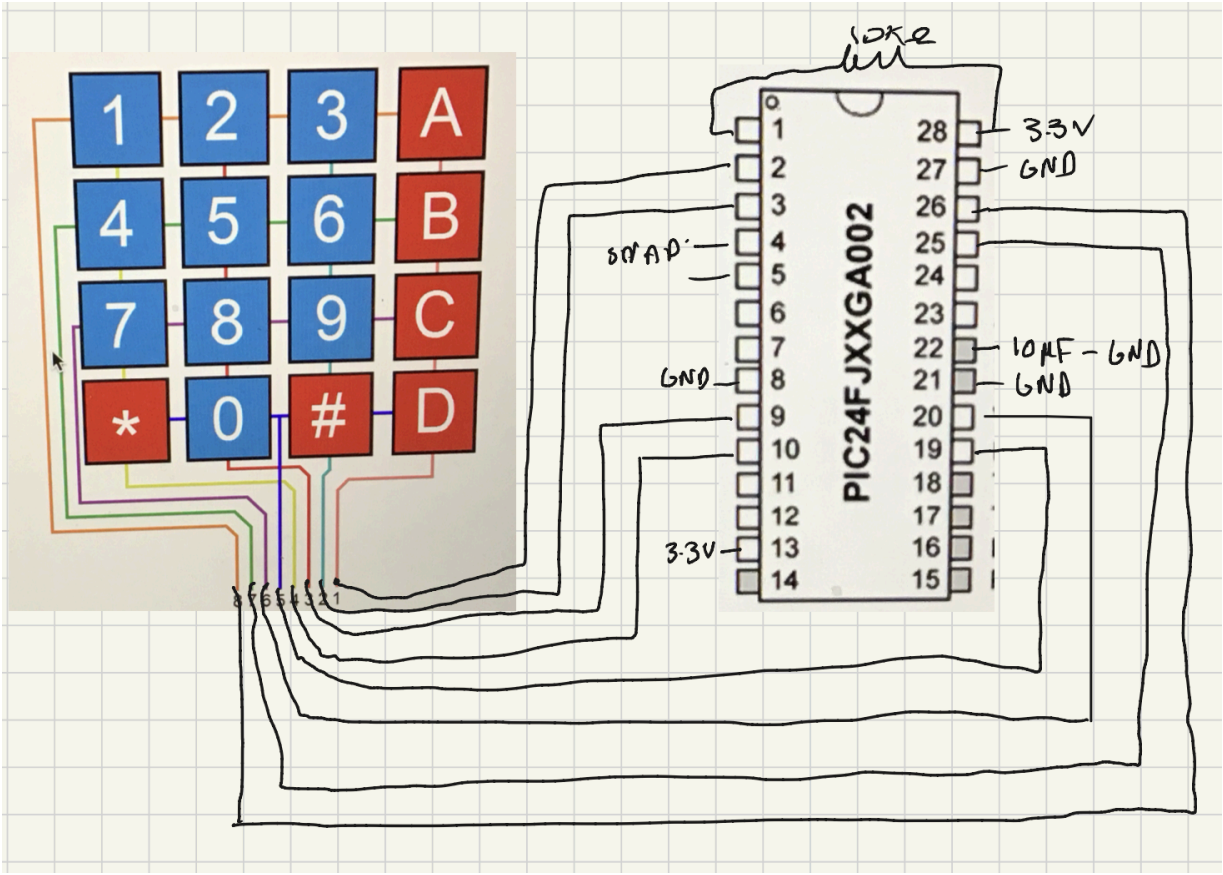
clearPIN(void)	Clears the PIN buffer and resets the digit index counter to 0 for next entry Inputs: none Returns: none Usage: clearPIN(); *// Reset pin buffer for new entry*

	Note: It loops though the array and set each character to '\0' Another thing is this function is automatically called by processKeyPress() after Pin validation

getPinAccepted(void)	Returns the current PIN acceptance status (1 if correct PIN entered, 0 otherwise use this function to gate access to protected features or operations in our case RFID reader
	Input: none
	Returns: if correct pin return integer 1 and rfid reader is enabled if incorrect pin returns 0 and rfidreader is still disabled
	Usage:
	`if (getPinAccepted()) {`
	`// system is unlocked - allow RFID scanning`
	`if (rfid_read_card_uid(cardID) == 0) {`
	`// Process card`
	`}`
	`} else {`
	`// system is locked - display "enter pin"`
	`lcd_printStr("Enter PIN");`
	`}`

resetSystem(void)	Resets the system for the next user by clearing the PIN buffer and setting pinAccepted to 0 its similar to initialization of initPinSystem(void) with addition of the pin accepted flag set back to 0 and delay of 500ms
	
	Input: none
	Return: none
	Usage:
	`// after successful RFID card scan`
	`if (rfid_read_card_uid(cardID) == 0) {`
	`// display card info`
	`lcd_printStr("Card Detected");`
	`__delay_ms(2000);`
	
	`// Signal success`
	`greenLED_ON();`
	`__delay_ms(3000);`
	`greenLED_OFF();`
	
	`// Reset for next user`
	`resetSystem();`
	`redLED_ON();  // turn red LED back on`
	`}`
	Note: we only call this function after completing a protected operation to prepare the system for the next user. The 500ms delay provides a brief pause between transactions.

Wiring Diagram



Row Pins (Outputs are active low)

Microcontroller Pin	KeypadPad pin	Row	Keys in Row
RB15	8	Row0	1,2,3,A
RB14	7	Row1	4,5,6,B
RB11	6	Row2	7,8,9,C
RB10	4	Row3	*,0,#,D

Column Pins (Inputs with Internal Pull-ups)

Microcontroller Pin	KeyPad pin	Column	Keys in Column
RA3	4	Column0	1,4,7,*
RA2	3	Column1	2,5,8,0

RA1	2	Column2	3,6,9,#
RA0	1	Column3	A,B,C,D

Figure 1: Configuration for the keypad diagram

Notes: internal interrupts were enabled on column keypad pins the operating voltage was 3.3V and the pull up resistance was ~20kΩ

Code

Source file

```
/*
 * File: keyPad.c
 * Author: Alamin Suliman
 * Project: RFID Security Access System
 * Description: Implementation of 4x4 matrix keypad with PIN entry system
 */

#include "xc.h"
#include "keyPad.h"
#include <string.h>                 // Include string library for strcmp function
#define FCY 16000000UL              // Define instruction cycle frequency
#include <libpic30.h>               // Include for __delay_ms() function
#define CORRECT_PIN "1234"          // Define the correct 4-digit PIN code
#define PIN_LENGTH 4                // Define PIN length as 4 digits

// keypad lookup tables
unsigned char keyTable[4][4] =
{
   // 0col    1col    2col    3col
    {'1', '2', '3', 'A'},  // Row 0 (RB15)
    {'4', '5', '6', 'B'},  // Row 1 (RB14)
    {'7', '8', '9', 'C'},  // Row 2 (RB11)
    {'E', '0', 'F', 'D'}   // Row 3 (RB10)
};

//global variables for PIN entry
char enteredPIN[PIN_LENGTH + 1];     // array to store entered PIN (+1 for null
terminator)
int pinIndex = 0;                   // index tracker for currentss PIN digit position (0-3)
```

```
int pinAccepted = 0;                // flag to track if correct PIN was entered 0=no and
1=yes

//Function: initKeyPad
//Description: Initializes the 4x4 matrix keypad hardware
//Rows: RB15, RB14, RB11, RB10 as outputs
//Columns: RA3, RA2, RA1, RA0 as inputs
void initKeyPad(void)
{
   // Enable internal pull-up resistors on column input pins
   CNPU1bits.CN2PUE = 1;       // Enable pull-up on RA0 (Column 3)
   CNPU1bits.CN3PUE = 1;       // Enable pull-up on RA1 (Column 2)
   CNPU2bits.CN30PUE = 1;      // Enable pull-up on RA2 (Column 1)
   CNPU2bits.CN29PUE = 1;      // Enable pull-up on RA3 (Column 0)

   // Configure column pins as inputs
   TRISA |= 0x000F;            // Set RA<3:0> as inputs (0x000F =
0b0000000000001111)

   // Configure row pins as outputs
   TRISBbits.TRISB15 = 0;      // Set RB15 as output (Row 0)
   TRISBbits.TRISB14 = 0;      // Set RB14 as output (Row 1)
   TRISBbits.TRISB11 = 0;      // Set RB11 as output (Row 2)
   TRISBbits.TRISB10 = 0;      // Set RB10 as output (Row 3)
}

//Function: readPins
//Description: Scans the entire 4x4 keypad matrix to detect key presses
//Sequentially activates each row and checks all columns
//Returns: ASCII character of pressed key (0-9, A-F) or '\0' if nothing pressed
unsigned char readPins(void)
{
   int key = 0;                    // Stores column number of key that was pressed
   unsigned char buttonPressed = '\0'; // Character to return if nothing was pressed

   //loop through each row sequentially (0 to 3)
   for (int row = 0; row < 4; row++) {

      //clear all row outputs (set them HIGH initially)
      LATBbits.LATB15 = 1;       // row 0 output = HIGH (inactive)
      LATBbits.LATB14 = 1;       // row 1 output = HIGH (inactive)
      LATBbits.LATB11 = 1;       // row 2 output = HIGH (inactive)
      LATBbits.LATB10 = 1;       // row 3 output = HIGH (inactive)

      //activate current row by pulling it LOW
      switch(row) {
```

```
            case 0:
                LATBbits.LATB15 = 0;    //pull Row 0 (RB15) LOW
                break;
            case 1:
                LATBbits.LATB14 = 0;    // pull Row 1 (RB14) LOW
                break;
            case 2:
                LATBbits.LATB11 = 0;    // pull Row 2 (RB11) LOW
                break;
            case 3:
                LATBbits.LATB10 = 0;    // pull Row 3 (RB10) LOW
                break;
        }

        __delay_ms(1);              // Wait 1ms for signal to stabilize (debouncing)
        key = determineKey();       // check which column is pulled LOW

        //check if a valid key was detected
        //valid columns are 0-3, 0xF (15) means no key was pressed
        if (key < 0xF)
        {
            buttonPressed = keyTable[row][key];  //look up character in key table
            return buttonPressed;               //return the pressed key character
        }
    }
    return buttonPressed;           // Return '\0' if no key was pressed in any row
}

//Function: determineKey
//Description: Reads column input pins to determine which column is LOW
//Uses internal pull-up resistors: unpressed columns read HIGH (1), pressed columns
read LOW (0)
//Column mapping: RA3=Col0, RA2=Col1, RA1=Col2, RA0=Col3
//Returns: Column number (0-3) or 0xF if no column is LOW
int determineKey(void)
{
    //check for Column 0 pressed (RA3 LOW, others HIGH)
    // binary pattern: RA<3:0> = 0111
    if (PORTAbits.RA0 == 1 && PORTAbits.RA1 == 1 &&
        PORTAbits.RA2 == 1 && PORTAbits.RA3 == 0)
    {
        return 0;       // Return column 0 (leftmost column: keys 1, 4, 7, E)
    }

    // check for Column 1 pressed (RA2 LOW, rest HIGH)
    // binary pattern: RA<3:0> = 1011
```

```
    if (PORTAbits.RA0 == 1 && PORTAbits.RA1 == 1 &&
        PORTAbits.RA2 == 0 && PORTAbits.RA3 == 1)
    {
        return 1;       // return column 1 (second column: keys 2, 5, 8, 0)
    }

    // check for Column 2 pressed (RA1 LOW and rest HIGH)
    // binary pattern: RA<3:0> = 1101
    if (PORTAbits.RA0 == 1 && PORTAbits.RA1 == 0 &&
        PORTAbits.RA2 == 1 && PORTAbits.RA3 == 1)
    {
        return 2;       // return column 2 (third column: keys 3, 6, 9, F)
    }

    // check for Colunmn 3 pressed (RA0 LOW rest HIGH)
    // binary pattern: RA<3:0> = 1110
    if (PORTAbits.RA0 == 0 && PORTAbits.RA1 == 1 &&
        PORTAbits.RA2 == 1 && PORTAbits.RA3 == 1)
    {
        return 3;       // return column 3 (rightmost column: keys A, B, C, D)
    }

    //no column is LOWsp no key is pressed in this row
    return 0xF;         // return invalid column indicator 15 in decimal
}

/*
 * Function: initPinSystem
 * Description: Initializes the PIN entry system
 * Calls initKeyPad() to set up keypad hardware
 * Starts with PIN not accepted
 */
void initPinSystem(void)
{
    initKeyPad();                   // Initialize keypad hardware (rows, columns, pull-ups)

    pinAccepted = 0;                // Start with PIN not accepted
                                    // User must enter correct PIN first

    clearPIN();                     // Initialize PIN buffer to empty state
                                    // Sets all characters to '\0' and resets index
}

/*
 * Function: processKeyPress
 * Description: Handles keypad input for PIN entry
```

```
 * Input: key - character from keypad (0-9 accepted, A-F ignored)
 * After 4 digits entered, automatically validates PIN
 * Correct PIN: sets pinAccepted flag to 1
 * Wrong PIN: keeps pinAccepted flag at 0 and adds 1 second delay
 */
void processKeyPress(unsigned char key)
{
   // Check if the pressed key is a digit (0-9)
   if (key >= '0' && key <= '9')    // Compare ASCII values: '0'=48, '9'=57
   {
      // Only accept input if we haven't reached PIN_LENGTH yet
      if (pinIndex < PIN_LENGTH)   // pinIndex ranges from 0 to 3
      {
         enteredPIN[pinIndex] = key;  // Store the digit in PIN array at current position
         pinIndex++;                  // Increment index to move to next position

         // Check if 4 digits have been entered
         if (pinIndex == PIN_LENGTH)   // If we've collected all 4 digits
         {
            enteredPIN[PIN_LENGTH] = '\0';  // Add null terminator to make valid C
string


            if (validatePIN())        // check if entered PIN matches "1234"
            {
               pinAccepted = 1;      // correct pin set flag to 1
            }
            else                     // incorrect pin  entered
            {
               pinAccepted = 0;      // Wrong PIN: keep flag at 0
               __delay_ms(1000);     // Wait 1 second before allowing retry

            }
            clearPIN();              //clear the PIN buffer for next attempt

         }
      }
   }
   // All non-digit keys (A, B, C, D, E, F) are ignored only numeric keys 0-9 are
processed
}

/*
 * Function: validatePIN
 * Description: Compares entered PIN with correct PIN using string comparison
 * Return: 1 if PIN is correct, 0 if incorrect
```

```
 */
int validatePIN(void)
{
    return (strcmp(enteredPIN, CORRECT_PIN) == 0);  // Return 1 for match, 0 for no
match
}

/*
 * Function: clearPIN
 * Description: Clears the entered PIN buffer and resets the index counter
 * Prepares system for next PIN entry attempt
 */
void clearPIN(void)
{
    // Loop through each position in the PIN array
    for (int i = 0; i < PIN_LENGTH; i++)
    {
        enteredPIN[i] = '\0';        // set each position to null character

    }
    pinIndex = 0;                   // reset index back to 0 for next PIN entry

}

/*
 * Function: resetSystem
 * Description: Resets system back to initial state after RFID scan complete
 * Clears PIN buffer and resets pinAccepted flag
 * Prepares system for next user
 * Call this after successfully reading and logging an RFID card
 */
void resetSystem(void)
{
    pinAccepted = 0;
    clearPIN();
    __delay_ms(500);

}

/*
 * Function: getPinAccepted
 * Description: Returns the current state of PIN acceptance
 * Return: 1 if correct PIN was entered, 0 if not
 * Used by main loop to check if RFID scanning should be allowed
 */
int getPinAccepted(void)
```

```
{
    return pinAccepted;
}
```

Header file

```
/*
 * File: keyPad.c
 * Author: Alamin Suliman
 * Project: RFID Security Access System
 * Description: Header file for 4x4 matrix keypad with PIN entry system
 */


#ifndef KEYPAD_H
#define KEYPAD_H

#include <stdint.h>  // Include for uint8_t and other standard integer types


void initKeyPad(void);
unsigned char readPins(void);
int determineKey(void);
void initPinSystem(void);
void processKeyPress(unsigned char key);
int validatePIN(void);
void clearPIN(void);
int getPinAccepted(void);
void resetSystem(void);

#endif // KEYPAD_H
```