Teclado Matricial con LCD

Contreras R Orlando[1,2*] and Garcia B Iker[1,2*†]

[3]Departamento de Sistemas Electrónicos, UAA, Av. Universidad 940, Aguascalientes, 20131, Aguascalientes, México.

*Corresponding author(s). E-mail(s): {al348390,al307630}@edu.uaa.mx;

†Ambos autores contribuyeron de forma equitativa a este trabajo.

Abstract

Esta practica aborda la lectura de un teclado matricial y la visualización de los datos en una pantalla LCD, mediante la implementación de máscaras lógicas y la correcta configuración de puertos.

Keywords: ASM, ARM, Shift, Registros

1 Introducción

En el mundo de los sistemas embebidos, las interfaces son cruciales para monitorear sensores y controlar operaciones. Entre los medios básicos de transmisión de información destaca la pantalla LCD. Este dispositivo proporciona una visualización clara y directa de datos y estados, facilitando la interacción y la toma de decisiones en aplicaciones embebidas.

2 Objetivos

- Diseñar un código funcional para leer de forma efectiva un teclado matricial y apreciarlo en un display LCD.
- Implementación de máscaras para la correcta lectura y escritura.
- Configuracion de puertos e inicialización de la LCD de manera efectiva.

3 Pseudocódigo

A continuación, se presenta el pseudocódigo que representa la lógica del programa en ensamblador:

Algorithm 1 Keypad Main Scanning Loop

Input: None
Output: Displays a key on the LCD if pressed
Initialize row selector mask: $R12 \leftarrow 1 \ll 8$ **for** $R2 \leftarrow 0$ *to* 3 **do**
    Clear row output bits in GPIOA
Set current row bit using $R12$
Delay for signal stabilization
Read column inputs from GPIOA **if** *any column input is low* **then**
    Store current row number to $R4$
Store column input to $R3$
Call DecodeKey(R3, R4)
Call PrintChar(decoded value)
Call WaitKeyRelease()
break

Shift $R12 \leftarrow R12 \ll 1$ (next row)

Algorithm 2 DecodeKey

procedure DecodeKey($colBits, rowIndex$)
    Input: Column bitmask (`colBits`), row index (`rowIndex`)
    Output: Register with decoded value **for** $colIndex \leftarrow 0$ *to* 3 **do**
    ($colBits$ & $(1 \ll colIndex)) == 0$
    goto matched

    return (no key detected)
    matched:
    keyIndex $\leftarrow 4 \times rowIndex + colIndex$
    Load character from key lookup table at keyIndex
    Return character in R6
end procedure

Algorithm 3 WaitKeyRelease

```
procedure WaitKeyRelease
    Input: None
    Output: Returns only when no key is pressed repeat
| C
 until
; 
 onfigure GPIO rows as outputs, columns as inputs
    Write logic low to all rows
    Delay for stabilization
    Read column pins
    all columns read as high
end procedure
```

Algorithm 4 PrintChar

```
procedure PrintChar(char)
    Input: Character to print in R6
    Output: Sends character to LCD in ASCII
    Set RS (register select) high for data mode
    Call SendByteToLCD(char)
end procedure
```

Algorithm 5 SendByteToLCD

```
procedure SendByteToLCD(byte)
    Input: Byte in R6
    Output: Sends byte to LCD via GPIOB
    Output byte to GPIOB
    Set Enable pin high
    Short delay
    Set Enable pin low
end procedure
```

4 Explicación del Código Ensamblador

Este es un fragmento del código original en ensamblador que corresponde al pseudocódigo anterior, aqui se puede aprecir el ciclo principal y algunas subrutinas, se omitió la configuración, el código se encuentra en GitHub Repository

```
__main
    BL      confRCC
    BL      confGPIO
    EOR     R2,R2
    LDR     R0, =GPIOA_ODR
aqui
    EOR     R2,R2
        EOR                 R12,R12
        ORR                 R12, #(1<<8)
loopi
        EOR                 R7,R7
        EOR                 R3,R3
        EOR                 R4,R4
    CMP     R2, #4
    BEQ     aqui

    AND     R7, #0xFFFF00FF
    ORR     R7, R12
    LDR     R0, =GPIOA_ODR
    STR                 R7, [R0]
    LSR     R4,R7,#8

    LDR     R0,=GPIOB_IDR
        LDR                 R3,[R0]
    LSR             R3,#12

    LSR     R5,R3, #1
    ANDS    R5, R3
    BEQ     Well
    MOV     R3, R5
Well
    ADDS    R3,#0
    BLNE    FindCol
        LSL     R12, #1
    ADD     R2,#1

    B      loopi

; ========== Subrutines ==========


FindCol
    EOR     R8,R8
    MOV     R10, #3
    CLZ     R11, R4
        SUB                 R11,#28
    SUB     R10,R11
    LSL     R10, #2
    CMP     R3, #4
    BEQ     is4
    CMP     R3,#8
    BEQ     is8
    SUB                 R8, #1
Done
        PUSH{LR}
        ADD     R3,R8
    ADD     R10,R3
        LDR     R0, =GPIOA_ODR
        AND     R7, #0xFFFFFF00
        STR                 R7, [R0]
    LSL     R10,#4
    ORR     R7, R10 ;

    CMP     R10, #0x90
    bhi     isA
    MOV     R3, #0x30
Ditto
    bl      LCDOUT
        CMP                 R7,R6
        BLNE        previousVal

    BL      Delay_5ms
    BL      Delay_5ms
    BL      Delay_5ms

        POP                 {LR}

    BX      LR

is4
    SUB     R8, #2
    b       Done
is8
    SUB     R8, #5
    b       Done
isA
    MOV     R3, #0x40
    sub     R7,#0x90
    b       Ditto

Delay_5ms
    ldr   r9, =led_delay
delay
    subs    r9,r9,#1
    bne     delay

    BX      LR
```

4.1 Configuración de GPIO

- Se configuran los pines del puerto A como salida.
- Se ajusta la velocidad y se establece un `pull-down` en los pines de entrada del puerto B, ademas se utilizan los primeros dos pines para salida de señal de control a la LCD (RS y Enable).

4.2 LCD Configuracion

4.2.1 ASCII (Data)

El código ascii es un sistema de codificación que asigna un valor numérico único a diferentes caracteres utilizados en la comunicación electrónica. Fue desarrollado en la década de 1960 como un estándar para la transferencia de datos entre dispositivos informáticos.

El código ASCII utiliza 7 bits para representar 128 caracteres diferentes. Entre estos caracteres se incluyen letras mayúsculas y minúsculas, dígitos numéricos, signos de puntuación, símbolos matemáticos y una serie de caracteres de control utilizados para el formateo de texto y el control de dispositivos. Los primeros 32 caracteres son caracteres de control no imprimibles, como el retorno de carro (enter, o salto de línea) y el avance de línea.

4.2.2 Control (RS, RW, Ena)

Para todo esto tiene dos buses, uno de datos (D0-D7) y otro de control (E, R/W y RS), que se comportan de la siguiente manera:
Bus de control:

- E: enable. Sirve como señal de captura, tanto para lectura como escritura.
- R/W: Indica el sentido de los datos: "1" para lectura y "0" para escritura.
- RS Si es igual a "1" indica que se envía o recibe un carácter y a "0" que se envía un comando.

Bus de datos:

- Si RS = 0, contiene el comando a ejecutar (borrar pantalla, poner el cursor al principio, avanzar una posición, mover el cursor, autoincremento de la posición de memoria, etc).
- Si RS=1, el bus de datos contiene el código ASCII del carácter que se quiere leer o escribir en el LCD.

RESUMEN DE COMANDOS

COMANDO	MODO	VALOR HEXADECIMAL
CLEAR DISPLAY		01
RETURN HOME		02
ENTRY MODE SET	INCREMENTA POSICIÓN DERECHA	05
	INCREMENTA POSICIÓN IZQUIERDA	07
CONTROL DISPLAY	DIPSLAY ON, CURSOR OFF, BLINK OFF	0C
	DISPLAY OFF, CURSOR OFF, BLINK OFF	08
	DISPLAY ON, CURSOR ON, BLINK OFF	0E
	DISPLAY ON, CURSOR OFF, BLINK ON	0D
	DISPLAY ON, CURSOR ON, BLINK ON	0F
CURSOR O DISPLAY SHIFT	CURSOR IZQUIERDA	10
	CURSOR DERECHA	14
	SMS IZQUIERDA	18
	SMS DERECHA	1C
SET		38

5 Conclusiones

Durante esta práctica, el principal reto fue lograr un funcionamiento eficiente del teclado matricial, lo cual requirió varios ajustes en el código. En cambio, la implementación de la pantalla LCD en modos de 8 y 4 bits resultó más sencilla gracias a la experiencia previa con el microcontrolador 8515. Además, se reforzaron conocimientos sobre el uso de instrucciones como PUSH y POP, útiles para la gestión del **LR** y subrutinas.