P1 - Digital Lock

Sereen Benchohra
EE 329-03
Fall 2021
Oct 15, 2021
Prof. Hummel

Behavior Description:

The digital lock starts with a default code of “1234,” and prompts “LOCKED” and “ENTER KEY” on the LCD. The user must enter the code with the keypad. If the code is entered incorrectly, or a ‘#’ or ‘*’ key is pressed, the screen clears and provides the locked prompt again. If the code is entered correctly, the LCD prompts “UNLOCKED” and “# TO CHANGE CODE.” If the user presses ‘#,’ the LCD displays “ENTER NEW CODE.” The user may not enter a new 4-digit code. If the ‘#’ or ‘*’ key is pressed while the user enters a new code, the lock reverts to the original code and displays the lock prompt. If the user enters a valid (4-digit numerical) code, this will be the new code and the lock returns to the locked prompt. The LED is turned on when lock is in the LOCKED state, otherwise , the LED is off

System Specification:

Overview:

LCD:

Name	**NHD-0216HZ-FSW-FBW-33V3C**
Description	Character LCD Display Module Transflective 5 x 8 Dots FSTN - Film Super-Twisted Nematic LED - White Parallel 65.50mm x 36.70mm x 15.00mm
Category	**Optoelectronics** **Display Modules - LCD, OLED Character and Numeric**
Mfr	Newhaven Display Intl
Series	-
Package	Bulk
Part Status	Active
Number of Characters	32
Display Format	16 x 2
Character Format	5 x 8 Dots
Display Type	FSTN - Film Super-Twisted Nematic
Display Mode	Transflective
Character Size	4.99mm H x 2.55mm W
Outline L x W x H	65.50mm x 36.70mm x 15.00mm
Viewing Area	54.00mm L x 14.40mm W
Backlight	LED - White
Voltage - Supply	3.3V
Dot Size	0.47mm W x 0.58mm H
Interface	Parallel
Controller Type	SPLC780D OR ST7066U
Operating Temperature	-20°C ~ 70°C
Text Color	Black
Background Color	White
Manufacturer Product Number	NHD-0216HZ-FSW-FBW-33V3C
Manufacturer	Newhaven Display Intl

Table 1 : LCD specifications

Keypad:

Name	**SWITCH KEYPAD 12 KEY NON-ILLUM**
Category	**Switches** **Keypad Switches**
Mfr	Adafruit Industries LLC
Series	-
Package	Bulk
Part Status	Active
Switch Type	Membrane (Snap Dome)
Number of Keys	12
Matrix (Columns x Rows)	3 x 4
Illumination	Non-Illuminated
Legend Type	Fixed
Key Type	Polyester Overlay
Output Type	Matrix
Legend	Telephone Format
Legend Color	White
Key Color	Blue, Red
Mounting Type	Panel Mount, Front
Termination Style	Cable with Connector
Manufacturer	Adafruit Industries LLC
Manufacturer Product Number	419

Table 2 : Keypad specifications

Board:

Name	**STM32L4675RG**
Category	**Development Boards, Kits, Programmers** **Evaluation Boards - Embedded - STM**
Mfr	Newhaven Display Intl
Series	STM32
Package	Bulk
Part Status	Active
Dot Size	0.47mm W x 0.58mm H
Interface	Parallel
Controller Type	SPLC780D OR ST7066U
Operating Temperature	-40°C ~ 85/105/125°C
Voltage - Supply	1.71 V to 3.6V power supply
Core Processor	Arm® Cortex®-M4 32-bit RISC
Operating Frequency	~ < 80 MHz
Flash memory	1 Mbyte
SRAM	128 Kbyte

Table 3 : Board specifications

System Schematic:

The Digital Lockbox is primarily made of three components, the Keypad, LCD, and the STM32 Microcontroller (see System specifications for more details on each of the components)

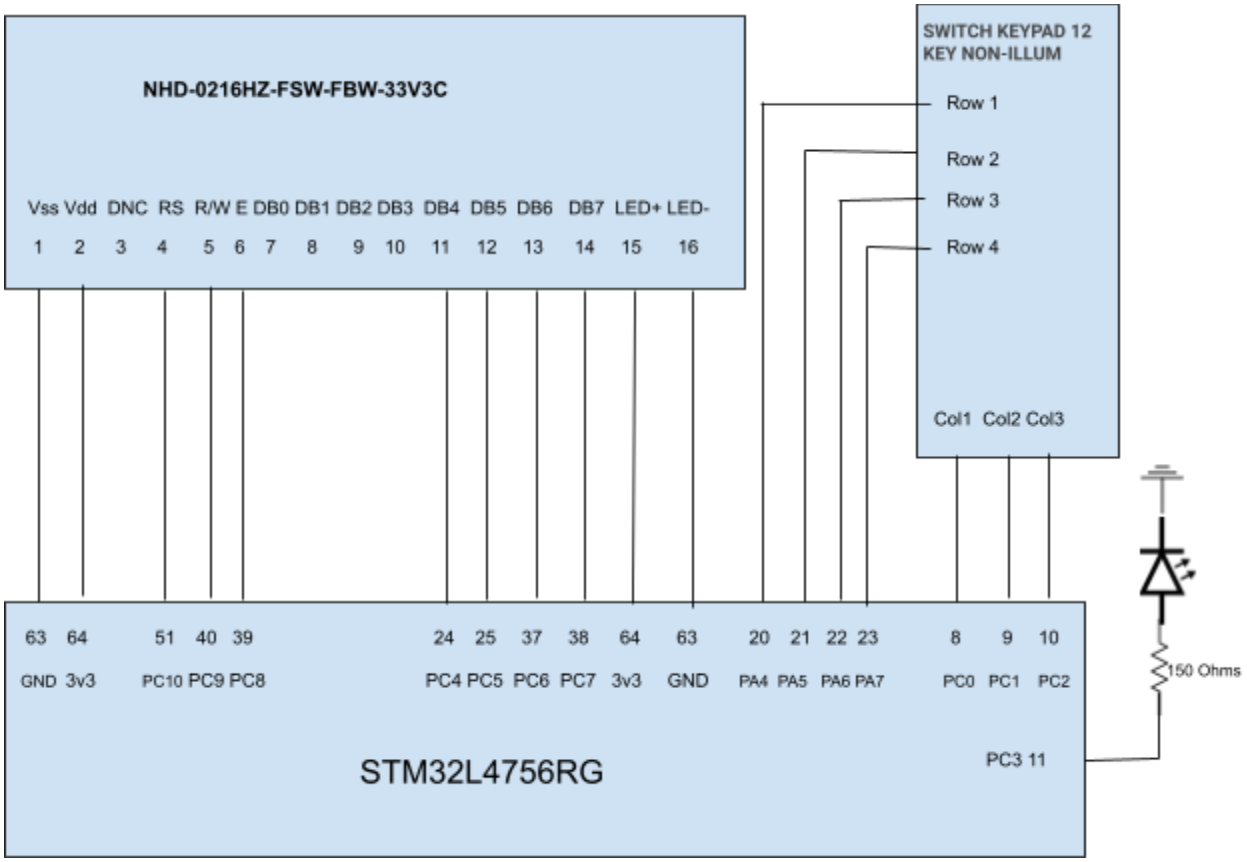

Figure 1: Lockbox Schematic showing how the STM board is connected to the LCD and Keypad and the LED

Software Architecture:

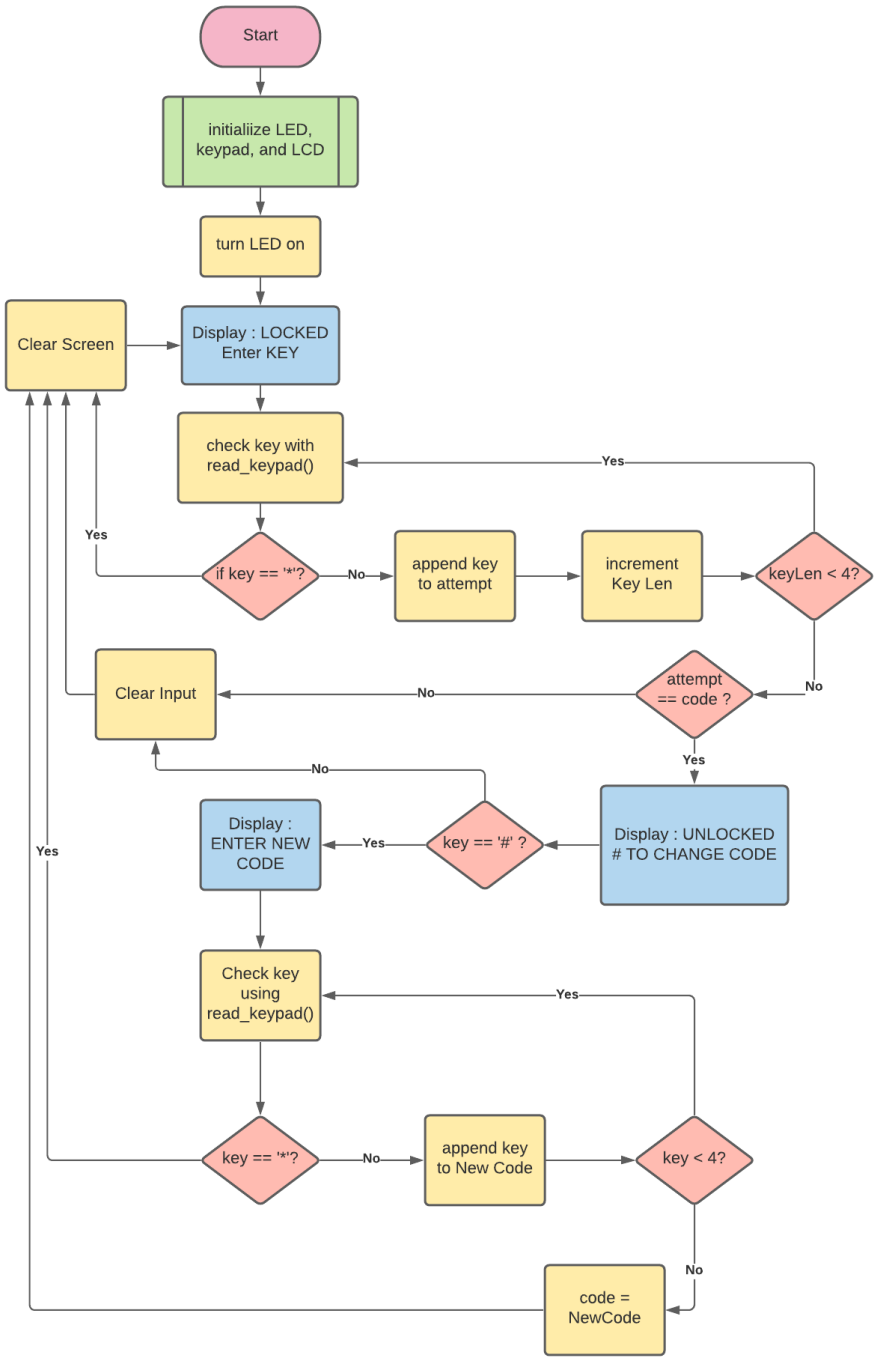

Figure 2: LockBox Flowchart demonstrating the programming process of the Lockbox (see appendix for code details)

Bill of Materials:

Index	Quantity	Part Number	Manufacturer Part Number	Description	Price
1	1	296-39653-ND	MSP-EXP432P401R	LAUNCHPAD MSP432P401R EVAL BRD	23.59
2	1	1568-1513-ND	PRT-12796	JUMPER WIRE F/F 6" 20PCS	1.95
3	1	1528-1136-ND	419	SWITCH KEYPAD 12 KEY NON-ILLUM	3.95
4	1	NHD-C0220AA-FSW-FTW-ND	NHD-C0220AA-FSW-FTW	LCD MOD 40DIG 20X2 TRANSFLCT WHT	10.85
5	1	2057-PH1-16-UA-ND	PH1-16-UA	CONN HEADER VERT 16POS 2.54MM	0.25
Total Price					40.59

Table 4: Bill of Materials chart

Appendices:
C Code:

---

main.c

```
#include "main.h"
#include "keypad.h"
#include "LCD.h"
#include "LED.h"
#include <string.h>

#define LOCKSTR1 "LOCKED"
#define LOCKSTR2 "ENTER KEY "
#define UNLOCKSTR1 "UNLOCKED"
#define UNLOCKSTR2 "# TO CHANGE CODE"
#define CHANGEKEY "ENTER NEW CODE"
#define ENTRY_CURSOR 0xCA
```

```
#define HOME 0x40
#define SECOND_LINE_HOME 0xC0
#define CLEAR_SCREEN 0x01

/* When the lockbox is locked, the onboard LED
 * will be on and when the lockbox is unlocked,
 * the LED will be off. */

void SystemClock_Config(void);
void SysTick_Init(void);
// change to STM board LED

int main(void)
{

  HAL_Init();
  SystemClock_Config();
  SysTick_Init();
  LCD_init();
  set_bit3LED();
  uint8_t key;
  int keyLen;
  char code[] = "1234";
  char newCode[] = "    ";
  char attempt[] = "    ";
  keypad_init();


  while (1)
  {

        GPIOC->ODR |= GPIO_ODR_OD3; // turns LED on when in LOCKED STATE
        attempt[0] = ' ';
      attempt[1] = ' ';
      attempt[2] = ' ';
      attempt[3] = ' ';

      keyLen = 0;
      //delay_us(50);
      LCD_command(CLEAR_SCREEN);     /* clear display */
        /* set cursor at beginning of 1st line */

      delay_us(50);
      LCD_command(HOME);
      delay_us(100);
      LCD_write_string(LOCKSTR1);
      LCD_write_string(LOCKSTR1);
```

```
        delay_us(50);
        LCD_command(SECOND_LINE_HOME);  /* set cursor at beginning of 2nd line */
        delay_us(50);
        LCD_write_string(LOCKSTR2);
        //delay_us(100);
        LCD_command(ENTRY_CURSOR);  /* set cursor at beginning of entry area */



        //wait for 4 characters to be pressed
        while(keyLen < 4)
        {
            key = read_keypad();
            if(key != NO_KEY && key != '#')
            {
                if(key == '*' )
                   break;

                LCD_write_char(key + '0');
                attempt[keyLen] = (key + '0');
                keyLen++;

                while(read_keypad() != NO_KEY);
//                for(int i = 0; i < 1000; i++)
//                      delay_us(1000);
            }
        }

        //compare attempt to code
        if(strcmp(attempt, code) == 0)
        {
           GPIOC->ODR &= ~(GPIO_ODR_OD3); // turns LED off when in UNLOCKED state
           LCD_command(CLEAR_SCREEN);     /* clear display */
           delay_us(100);
            LCD_command(HOME);  /* set cursor at beginning of 1st line */
            //P2->OUT |= (GREEN & LOWER_3BITS);
            for(int i = 0; i < 100; i++)
                 delay_us(1000);
            LCD_write_string(UNLOCKSTR1);
            //LCD_write_string(UNLOCKSTR1);

            LCD_command(SECOND_LINE_HOME);  /* set cursor at beginning of 2nd line */
            LCD_write_string(UNLOCKSTR2);

            //must press any key to continue
            //key = read_keypad();
            //while(read_keypad() != NO_KEY);
```

```
            while((key = read_keypad()) == 0xFF);

            //if # is pressed, enter new code
            if(key == '#')
            {
                keyLen = 0;
                LCD_command(CLEAR_SCREEN);      /* clear display */
                delay_us(100);
                LCD_command(HOME);  /* set cursor at beginning of 1st line */
                delay_us(100);
                LCD_write_string(CHANGEKEY);
                LCD_command(SECOND_LINE_HOME);  /* set cursor at beginning of 2nd
line */
                while(read_keypad() != NO_KEY);
                //wait for 4 keys to be pressed
                while(keyLen < 4)
                {
                    key = read_keypad();
                    //invalid keys
                    //while(read_keypad() != NO_KEY);
                    if(key == '*')
                        break;
                    //numerical keys
                    if(key != 0xFF && key != '#')
                    {
                        LCD_write_char(key + '0');
                        newCode[keyLen] = key + '0';
                        keyLen++;
                        while(read_keypad() != NO_KEY);
                        //to help debounce
                    }
                }
                //set newCode
                if(keyLen == 4)
                {
                    code[0] = newCode[0];
                    code[1] = newCode[1];
                    code[2] = newCode[2];
                    code[3] = newCode[3];
                }
            }
        }

        delay_us(500);
    }
}
```

```c
/**
  * @brief System Clock Configuration
  * @retval None
  */

void SysTick_Init(void){
    SysTick->CTRL |= (SysTick_CTRL_ENABLE_Msk |           // enable SysTick Timer
                      SysTick_CTRL_CLKSOURCE_Msk);     // select CPU clock
    SysTick->CTRL &= ~(SysTick_CTRL_TICKINT_Msk);      // disable interrupt,
                                                       // breaks HAL delay function
}

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
```

```
  }
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

LDC.c

```
#include "LCD.h"
#include <string.h>
#include "msp.h"




void Clear_LCD(){ /* clear display */
    LCD_command(CLEAR);
}
void Home_LCD(){ /* set cursor at beginning of first line */
    LCD_command(HOME);
}

void Second_LCD(){ /* set cursor at beginning of second line */
    LCD_command(SECOND_LINE_HOME);
}

void Write_char_LCD(unsigned char data){ /*Inputs a char into LCD */
    LCD_data(data);
}

void Write_string_LCD(const char* data){ /* Takes string and input each character
individually into the LCD */
```

```
    int i,length;
    length = strlen(data);
    for(i = 0; i < length; i++)
       Write_char_LCD(data[i]);
}
void LCD_init(void) {
    P4->DIR = 0xFF;        /* make P4 pins output for data and controls */
    __delay_us(40000); /* initialization sequence */
    LCD_nibble_write(0x30, 0); /*Send data to 0x3 */
    __delay_us(37000);
    LCD_nibble_write(0x30, 0);
    __delay_us(37);
    LCD_nibble_write(0x30, 0);
    __delay_us(37);
    LCD_nibble_write(0x20, 0);  /* use 4-bit data mode */
    __delay_us(37);

    LCD_command(0x28);     /* set 4-bit data, 2-line, 5x7 font */
    LCD_command(MOVE_RIGHT);     /* move cursor right after each char */
    LCD_command(CLEAR);    /* clear screen, move cursor to home */
    LCD_command(TURN_ON);        /* turn on display, cursor blinking */
}

/* With 4-bit mode, each command or data is sent twice with upper
 * nibble first then lower nibble.
```

```
 */
void LCD_nibble_write(unsigned char data, unsigned char control) {
    data &= 0xF0;          /* clear lower nibble for control */
    control &= 0x0F;       /* clear upper nibble for data */
    P4->OUT = data | control;    /* RS = 0, R/W = 0 */
    P4->OUT = data | control | EN;  /* pulse E */
    __delay_us(0);
    P4->OUT = data;              /* clear E */
    P4->OUT = 0;
}

void LCD_command(unsigned char command) {
    LCD_nibble_write(command & 0xF0, 0);      /* upper nibble first */
    LCD_nibble_write(command << 4, 0);        /* then lower nibble */

    if (command < 4)
       __delay_us(4000);          /* commands 1 and 2 need up to 1.64ms */
    else
       __delay_us(1000);
}

void LCD_data(unsigned char data) {
    LCD_nibble_write(data & 0xF0, RS);    /* upper nibble first */
    LCD_nibble_write(data << 4, RS);    /* then lower nibble  */
```

```
    __delay_us(1000);
}
```

LCD.h

```
 /*
 * LCD.h
 *
 *  Created on: Oct 15, 2021
 *      Author: Sereen
 */

#ifndef LCD_H_
#define LCD_H_

#define RS_bit    (GPIO_ODR_OD10)
#define RW_bit    (GPIO_ODR_OD9)
#define E_bit     (GPIO_ODR_OD8)
#define data_bits (GPIO_ODR_OD4 | GPIO_ODR_OD5 | GPIO_ODR_OD6 |
GPIO_ODR_OD7)

void delay_us(const uint16_t time_us);
void LCD_init (void);
void LCD_command (uint8_t command_byte);
void LCD_write_nib (uint8_t data_nib);  //will only work in the command
subroutine?
void LCD_write_char_nib (uint8_t char_nib);
void LCD_write_char (uint8_t char_byte);
void LCD_write_string(const char *string_in );
void LCD_port_setup(void);


#endif /* LCD_H_ */
```

LCD.c

```
/*
 * LCD.c
 *
 *  Created on: Oct 15, 2021
 *      Author: Sereen
 */
#include "main.h"
#include "LCD.h"

void LCD_init(void)
{
	LCD_port_setup();
	delay_us(50);

	LCD_port_setup();
	delay_us(50);
}

void LCD_port_setup(void)
{

	RCC->AHB2ENR   |=  (RCC_AHB2ENR_GPIOCEN); //enables bus C

	GPIOC->MODER   &= ~(GPIO_MODER_MODE4 | GPIO_MODER_MODE5 |
GPIO_MODER_MODE6 | GPIO_MODER_MODE7
			| GPIO_MODER_MODE8 | GPIO_MODER_MODE9 |
GPIO_MODER_MODE10);

    GPIOC->MODER   |=  (GPIO_MODER_MODE4_0 | GPIO_MODER_MODE5_0 |
GPIO_MODER_MODE6_0 | GPIO_MODER_MODE7_0
		| GPIO_MODER_MODE8_0 | GPIO_MODER_MODE9_0 |
GPIO_MODER_MODE10_0); //set c pins 4-7 and 8-10 to output (for LCD data
then control)

    GPIOC->OTYPER  &= ~(GPIO_OTYPER_OT4 | GPIO_OTYPER_OT5 | GPIO_OTYPER_OT6
```

```
| GPIO_OTYPER_OT7
                      | GPIO_OTYPER_OT8 | GPIO_OTYPER_OT9 |
GPIO_OTYPER_OT10); //pins C0-3 and 8-10 to push pull

    GPIOC->OSPEEDR &= ~(GPIO_OSPEEDR_OSPEED4 | GPIO_OSPEEDR_OSPEED5 |
GPIO_OSPEEDR_OSPEED6 | GPIO_OSPEEDR_OSPEED7
                      | GPIO_OSPEEDR_OSPEED8 | GPIO_OSPEEDR_OSPEED9 |
GPIO_OSPEEDR_OSPEED10);  //slow speed on output pins c 0-3 and 8-10

      GPIOC->PUPDR   &= ~(GPIO_PUPDR_PUPD4 | GPIO_PUPDR_PUPD5 |
GPIO_PUPDR_PUPD6 | GPIO_PUPDR_PUPD7
                           | GPIO_PUPDR_PUPD8 | GPIO_PUPDR_PUPD9 |
GPIO_PUPDR_PUPD10);

//    GPIOC->ODR     |= (RW_bit | RS_bit);


       //we want no pull up or pull down but IDK if this actually needs to
be included, i think it populates to this anyway
      // C pins 0-3 will be for data (d4-7); C pin 10->RS; pin 9->RW pin
8->E

    for (int i=0; i<40;i++){
      delay_us(1000);
    }

      LCD_write_nib(0x3); //4 bit
      delay_us(50);
      LCD_command(0x28); //function set
      delay_us(50);
      LCD_command(0x28); //again for some reason
      delay_us(50);
      LCD_command(0x0F); //turns on cursor
      delay_us(50);
      LCD_command(0x01); //clears screen
      delay_us(1000);
      delay_us(520);

//    delay_us(148);
//    LCD_command (0x80);
//    delay_us(148);
//    LCD_command(0x01); //clear display
```

```
//    delay_us(148);
//    LCD_command (0x02); //return home
//    delay_us(5520);
      LCD_command (0x06);// increments cursor

//    LCD_command (0x0F); //turns on blinker again
      delay_us(50);
}

void LCD_write_nib (uint8_t data_nib){

      GPIOC->ODR &=  ~(RW_bit | RS_bit); //sets pins connected to RS, RW,
and data pins to zero

      GPIOC->ODR &=  ~(data_bits);

      data_nib=data_nib<<4;

      GPIOC->ODR |=  (data_nib);

      GPIOC->ODR |=  (E_bit); //puts the ones corresponding to the data
inputed in the ODR at bits 0-4

      delay_us(1); //this one we need

      GPIOC->ODR &=  ~(E_bit); //sets Enable pin to zero

      return;
}

void LCD_command (uint8_t command_byte){
      GPIOC->ODR &=  ~(RW_bit | RS_bit);  //maybe take out?
      uint8_t ls_nib= 0x0F & command_byte;

      uint8_t ms_nib= 0xF0 & command_byte;

      ms_nib= ms_nib>>4;

      //delay_us(50);  //probably dont need

      LCD_write_nib (ms_nib);
```

```
	delay_us(2);   //this we need

	LCD_write_nib (ls_nib);

//    delay_us(1520);
}

void LCD_write_char_nib (uint8_t char_nib){

	GPIOC->ODR &=  ~(RW_bit); //sets pins connected to RS, and data pins
to zero
	GPIOC->ODR |=   (RS_bit);

//    delay_us(1000);

	GPIOC->ODR &=  ~(data_bits);

	char_nib=char_nib<<4;

	GPIOC->ODR |=  (char_nib);

	GPIOC->ODR |=  (E_bit); //puts the ones corresponding to the data
inputed in the ODR at bits 0-4

	delay_us(2); //this one we need

	GPIOC->ODR &=  ~(E_bit); //sets Enable pin to zero

//    GPIOC->ODR &=  ~(RS_bit); //do i need to be doing this?
//    delay_us(1000);

	return;
}

void LCD_write_char (uint8_t char_byte){
//    GPIOC->ODR &=  ~(RW_bit | RS_bit);  //maybe take out?
	uint8_t ls_nib= 0x0F & char_byte;

	uint8_t ms_nib= 0xF0 & char_byte;

	ms_nib= ms_nib>>4;
```

```
//    delay_us(1520);

      LCD_write_char_nib (ms_nib);

      delay_us(2);   //this we need

      LCD_write_char_nib (ls_nib);
      delay_us(50);   //this we probably dont need

//    delay_us(1520);
}

void LCD_write_string(const char *string_in){
      for(int i = 0; string_in[i]!='\0'; i++){
            LCD_write_char(string_in[i]);
            delay_us(50);
      }
}

void  delay_us(const uint16_t time_us) {
    // set the counts for the specified delay
    SysTick->LOAD = (uint32_t)((time_us * SystemCoreClock / 1000000) - 1);
    SysTick->VAL = 0;                                          // clear the
timer count
    SysTick->CTRL &= ~(SysTick_CTRL_COUNTFLAG_Msk);        // clear the
count flag
    while (!(SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk)); // wait for the
flag
}
```

---

keypad.c

```
#include "main.h"
#include "keypad.h"
#include "LCD.h"
```

```
void keypad_init(void)
{
       RCC->AHB2ENR |= (RCC_AHB2ENR_GPIOAEN); // enable GPIOB clock on bus
       RCC->AHB2ENR |= (RCC_AHB2ENR_GPIOCEN); // enable GPIOC clock on bus

   // clear GPIOA PA4-PA7 (also sets them for input
       GPIOA->MODER &= ~(    GPIO_MODER_MODE4
                                    | GPIO_MODER_MODE5
                                    | GPIO_MODER_MODE6
                                    | GPIO_MODER_MODE7);
       // clear MODE PC5 - PC7  bits for keypad and use as columns
       GPIOC->MODER &= ~(GPIO_MODER_MODE0
                                    | GPIO_MODER_MODE1
                                    | GPIO_MODER_MODE2);

       // set PC5-PC7 as outputs for columns
       GPIOC->MODER |= ( (1 << GPIO_MODER_MODE0_Pos)
                                | (1 << GPIO_MODER_MODE1_Pos)
                                | (1 << GPIO_MODER_MODE2_Pos) );

       // enable pulldown resistor for rows on PA4-7

       // clear pupdr
       GPIOA->PUPDR &=  ~(    GPIO_PUPDR_PUPD4_1
                                    | GPIO_PUPDR_PUPD5_1
                                    | GPIO_PUPDR_PUPD6_1
                                    | GPIO_PUPDR_PUPD7_1);


       GPIOA->PUPDR |=  (    GPIO_PUPDR_PUPD4_1
                                    | GPIO_PUPDR_PUPD5_1
                                    | GPIO_PUPDR_PUPD6_1
                                    | GPIO_PUPDR_PUPD7_1);

       // enable push-pull for columns on PC5-PC7
       // (check later if there is problems)
       GPIOC->OTYPER &= ~( GPIO_OTYPER_OT0
                                    | GPIO_OTYPER_OT1
                                    | GPIO_OTYPER_OT2);


       // set slow speed for columns (PC5-PC7)

       GPIOC->OSPEEDR &= ~(GPIO_OSPEEDR_OSPEED0
                                          | GPIO_OSPEEDR_OSPEED1
                                          | GPIO_OSPEEDR_OSPEED2); // PA0 slow
speed
```

```
	// set columns to high
	COL_PORT |= COL_MASK;

}


uint8_t calculate_key(uint16_t col, uint8_t row)
{
	uint8_t key;
	uint8_t rows;

	rows = row >> 4; // right shift rows 4 places for easiers calc

	if(rows == 4)
		rows = 3;
	if(rows == 8)
		rows = 4;
	// calculate key based on col
	switch(col)
	{
		case 0:
			key = (3 * rows) - 2;
			break;
		case 1:
			key = (3 * rows) - 1;
			break;
		case 2:
			key = (3 * rows);
			break;
	}


	if(key == 10)   // leave as 10 so it can output to LED (change with LCD)
		key  = '*'; // leave it in ASCII for later use
	if(key == 11)
		key = 0;
	if(key == 12)  // leave as 12 so so it can output to LED (change with LCD)
		key = '#';

	return key;
}

void check_columns(uint8_t cur_col)
{
	COL_PORT  &= ~(COL_MASK); // turn all colums off
	switch(cur_col)  // set a col high depending on the row
```

```
	{
		case 0:
			COL_PORT |= COL1;
			break;
		case 1:
			COL_PORT |= COL2;
			break;
		case 2:
			COL_PORT |= COL3;
			break;

	}
}


uint8_t read_keypad(void)
{
	uint8_t rows;
	uint8_t cur_col ;
	uint8_t key;
	// Read the rows PB4-PB7 only
	rows = ROW_PORT & ROW_MASK ;

	if(rows == 0) // check to see if all the rows are low (is so return NO KEY)
		return NO_KEY;

	for(cur_col = 0; cur_col < 3; cur_col++)
	{

		// set current columns high others low
		check_columns(cur_col);
		// read the rows
	    rows = ROW_PORT & ROW_MASK;

		if(rows != 0)
		{
			// calculate button from row and col
			key = calculate_key(cur_col, rows);
			// set columns to high
			COL_PORT |= COL_MASK;
			return key;
		}

	}

	// set columns to high
	COL_PORT |= COL_MASK;
```

```
        return NO_KEY;

}
```

Keypad.h

```
/*
 * keypad.h
 *
 *  Created on: Sep 29, 2021
 *      Author: Sereen
 */

#ifndef SRC_KEYPAD_H_
#define SRC_KEYPAD_H_

void keypad_init(void);
uint8_t read_keypad(void);

#define STAR   10
#define POUND  11
#define NO_KEY 0xFF

// Ports PA4 - PA7
#define ROW1 GPIO_IDR_ID4
#define ROW2 GPIO_IDR_ID5
#define ROW3 GPIO_IDR_ID6
#define ROW4 GPIO_IDR_ID7

#define ROW_PORT (GPIOA->IDR)
#define COL_PORT (GPIOC->ODR)
```

```
// change to PC 0, 1 , 2
#define COL1 GPIO_ODR_OD0
#define COL2 GPIO_ODR_OD1
#define COL3 GPIO_ODR_OD2

#define COL_MASK (COL1|COL2|COL3)
#define ROW_MASK (ROW1|ROW2|ROW3|ROW4)


#endif /* SRC_KEYPAD_H_ */
```

LED.h

```
/*
 * LED.h
 *
 *  Created on: Oct 1, 2021
 *      Author: Sereen
 */

#ifndef SRC_LED_H_
#define SRC_LED_H_

void count(uint8_t num);
void set_4bitLED(void);
void set_bit0LED(void);
void set_bit1LED(void);
void set_bit2LED(void);
void set_bit3LED(void);

#endif /* SRC_LED_H_ */
```

LED.c

```c
/*
 * LED.c
 *
 *  Created on: Oct 1, 2021
 *      Author: Sereen
 */
#include "main.h"
#include "LED.h"

void count(uint8_t num)
{
      GPIOC->ODR &= ~(GPIO_ODR_OD0 | GPIO_ODR_OD1 | GPIO_ODR_OD2 |
GPIO_ODR_OD3); //resets registers in ODR
      GPIOC->ODR |= num; // sets the num in the ODR to displays to 4 LEDs

}

// Initializes the PINS to output as a 4 bit LED
void set_4bitLED(void)
{
   RCC->AHB2ENR |= (RCC_AHB2ENR_GPIOCEN); // enable GPIOA clock on bus
   set_bit0LED();
   set_bit1LED();
   set_bit2LED();
   set_bit3LED();

}
// initialzes the first LED as a 1st bit representation
void set_bit0LED(void)
{
  GPIOC->MODER &= ~(GPIO_MODER_MODE0);   // clear MODE0 bits
  GPIOC->MODER |= (1 << GPIO_MODER_MODE0_Pos); // set PA0 as GPIO output
  GPIOC->OTYPER &= ~(GPIO_OTYPER_OT0);        // PA0 set to push-pull
  GPIOC->OSPEEDR &= ~(GPIO_OSPEEDR_OSPEED0); // PA0 slow speed
  GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPD0);  // PA0 no pullup/pulldown resistor
}
```

```
// initializes the second LED as a 2nd bit representation
void set_bit1LED(void)
{
  GPIOC->MODER &= ~(GPIO_MODER_MODE1); // clear MODE1 bits
  GPIOC->MODER |= (1 << GPIO_MODER_MODE1_Pos); // set PA1 as GPIO output
  GPIOC->OTYPER &= ~(GPIO_OTYPER_OT1); // PA1 set to push-pull
  GPIOC->OSPEEDR &= ~(GPIO_OSPEEDR_OSPEED1); // PA1 slow speed
  GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPD1);  // PA1 no pullup/pulldown resistor
}

// initializes the third LED as a third bit representation
void set_bit2LED(void)
{
  GPIOC->MODER &= ~(GPIO_MODER_MODE2);  // clear MODE2 bits
  GPIOC->MODER |= (1 << GPIO_MODER_MODE2_Pos); // set PA2 as GPIO output
  GPIOC->OTYPER &= ~(GPIO_OTYPER_OT2);  // PA2 set to push-pull
  GPIOC->OSPEEDR &= ~(GPIO_OSPEEDR_OSPEED2);  // PA2 slow speed
  GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPD2);  //PA2 no pullup/pulldown resistor
}

// initializes the fourth  LED as a 4th bit representation
void set_bit3LED(void)
{
  GPIOC->MODER &= ~(GPIO_MODER_MODE3);  // clear MODE3 bits
  GPIOC->MODER |= (1 << GPIO_MODER_MODE3_Pos); // set PA3 as GPIO output
  GPIOC->OTYPER &= ~(GPIO_OTYPER_OT3); // PA3 set to push-pull
  GPIOC->OSPEEDR &= ~(GPIO_OSPEEDR_OSPEED3); // PA3 slow speed
  GPIOC->PUPDR &= ~(GPIO_PUPDR_PUPD3); // PA3 no pullup/pulldown resistor
}
```

Datasheets/Manual/References

Keypad:
https://media.digikey.com/pdf/Data%20Sheets/Adafruit%20PDFs/419_Web.pdf

LCD

https://www.newhavendisplay.com/specs/NHD-0216HZ-FSW-FBW-33V3C.pdf

STM:
https://www.st.com/resource/en/datasheet/stm32l476je.pdf