Professor: John Penvenne

EE 329-05 S'25
MWF 2-3pm

Lab A2
Keypad

Written by:
Justin Rosu & Brayden Daly

This lab is about interfacing a 4x4 matrix keypad with the STM32L4 microcontroller using GPIO polling. The keypad is a passive switch matrix, requiring detection logic to determine which key has been pressed. The task involves designing a keypad module with configuration and polling functions, implementing software debouncing to ensure reliable key detection, and displaying the identified key on a 4-bit LED output. By configuring pull-down resistors and scanning columns and rows appropriately, the system can identify single keypresses accurately.

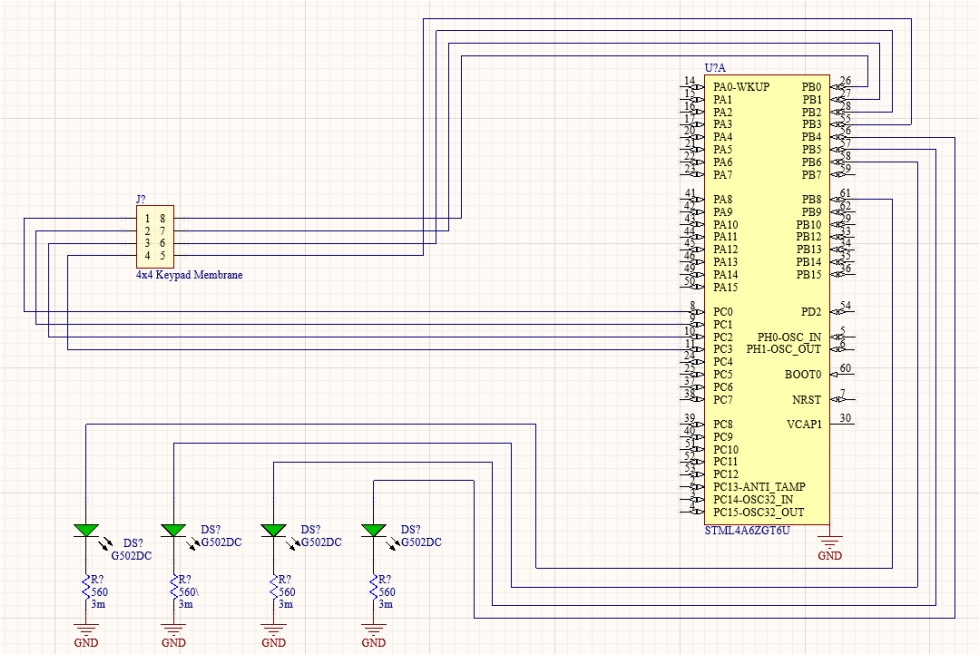

Figure 1. Keypad circuit diagram

The column pins of the keypad are on GPIOB (Outputs) pins 0-3. The row pins of the keypad are GPIOC (Inputs) pins 0-3. The output is mapped onto leds as a 4 bit output using GPIOB(Outputs) pins 4-6,8.

Table 1: L4A6ZGT6 I/O Characteristics

	RESISTANCE (Ω)	PAGE	REF.
PULL UP	40k	176	STM32L4A6 DATASHEET
PULL DOWN	40k	176	STM32L4A6 DATASHEET

Table 2: L4A6ZGT6 I/O Characteristics (from L4A6ZGT6 Datasheet)

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
I_{lkg}(4)	FT_xx input leakage current(3)(5)	$V_{IN} \le Max(V_{DDXXX})$(6)(7)	-	-	±100	nA
		$Max(V_{DDXXX}) \le V_{IN} \le Max(V_{DDXXX})+1$ V(6)(7)	-	-	650	
		$Max(V_{DDXXX})+1$ V < $V_{IN} \le$ 5.5 V(6)(7)	-	-	200	
	FT_lu, FT_u, PB2 and PC3 IO	$V_{IN} \le Max(V_{DDXXX})$ (6)(7)	-	-	±150	
		$Max(V_{DDXXX}) \le V_{IN} \le Max(V_{DDXXX})+1$ V(6)(7)	-	-	2500(3)	
		$Max(V_{DDXXX})+1$ V < $V_{IN} \le$ 5.5 V(6)(7)	-	-	250	
	TT_xx input leakage current	$V_{IN} \le Max(V_{DDXXX})$(6)	-	-	±150	
		$Max(V_{DDXXX}) \le V_{IN}$ < 3.6 V(6)	-	-	2000(3)	
	OPAMPx_VINM (x=1,2) dedicated input leakage current (UFBGA132 only)	-	-	-	(8)	
R_{PU}	Weak pull-up equivalent resistor (9)	$V_{IN} = V_{SS}$	25	40	55	kΩ
R_{PD}	Weak pull-down equivalent resistor(9)	$V_{IN} = V_{DDIOx}$	25	40	55	kΩ
C_{IO}	I/O pin capacitance	-	-	5	-	pF

Pseudocode

MAIN FUNCTION
CHECK IF BUTTON IS PRESSED
    IF BUTTON PRESSED: (DEBOUNCE FUNCTION)
        CHECK WHICH BUTTON PRESSED: (WHICHKEYPRESSED)
        OUTPUT BUTTON WHICH IS PRESSED ONTO LEDS

DEBOUNCE
HOLD A INPUT VALUE
DELAY
HOLD ANOTHER INPUT VALUE
DELAY
HOLD ANOTHER INPUT VALUE
DELAY
LOOP UNTIL ALL 3 VALUES ARE EQUAL
RETURN TRUE

WHICHKEYPRESSED
DETECT WHICH ROW IS PRESSED

FOR THE ROW PRESSED
    TURN OFF ALL THE COLUMNS
    TURN ON COLUMNS 1 by 1 UNTIL THE ROW IS DETECTED AGAIN
    DETERMINE IF SWITCH IS HIGH
    OUTPUT VALUE MAPPED TO THAT ROW AND COLUMN

Demonstration:
https://www.youtube.com/watch?v=q5GYBLxljzY

APPENDIX

main.h

```
/******************************************************************************
* EE 329 A2 KEYPAD INTERFACE
******************************************************************************
* @file          : main.c
* @brief         : Runs the functions from Keypad.c and outputs onto LEDs
* project        : EE 329 S'25 Assignment 2
* authors        : Justin Rosu & Brayden Daly  (JRBD)
* version        : 0.1
* date           : 250412
* compiler       : STM32CubeIDE v.1.12.0 Build: 14980_20230301_1550 (UTC)
* target         : NUCLEO-L4A6ZG
* clocks         : 4 MHz MSI to AHB2
* @attention     : (c) 2023 STMicroelectronics.  All rights reserved.
******************************************************************************
* KEYPAD PLAN :
* set columns as outputs, rows as inputs w pulldowns
* loop:
* drive all columns HI read all rows
* if any row N is HI
*    set all columns LO
*    drive each column M HI alone
*    read row N until HI □ pressed key loc'n = N, M
* key value = 3N+M+1 for 1..9,special case for *,0,#
******************************************************************************
* KEYPAD WIRING 4 ROWS 4 COLS (pinout NUCLEO-L4A6ZG = L496ZG)
*      peripheral – Nucleo I/O
* keypad 1  COL 4 - PB0
* keypad 2  COL 3 - PB1
* keypad 3  COL 2 - PB2
* keypad 4  COL 1 - PB3
* keypad 5  ROW 4 - PC2
* keypad 6  ROW 3 - PC2
* keypad 7  ROW 2 - PC1
* keypad 8  ROW 1 - PC0
* OUTPUT LED BIT0 - PB4
* OUTPUT LED BIT1 - PB5
* OUTPUT LED BIT2 - PB6
* OUTPUT LED BIT3 - PB8
******************************************************************************
* REVISION HISTORY
* 0.1 230318 JRBD  created, wires in breadboard, no keypad
* 0.2 230410 JRBD  made code to make keypad operational
* 0.3 230413 JRBD  implemented modularity
******************************************************************************
```

```
******************************************************************************
* 45678-1-2345678-2-2345678-3-2345678-4-2345678-5-2345678-6-2345678-7-234567 */
/* USER CODE END Header */
#ifndef __MAIN_H
#define __MAIN_H
#ifdef __cplusplus
extern "C" {
#endif
#include "stm32l4xx_hal.h"
void Error_Handler(void);
/* Private defines -----------------------------------------------------------*/
#define B1_Pin GPIO_PIN_13
#define B1_GPIO_Port GPIOC
#define LD3_Pin GPIO_PIN_14
#define LD3_GPIO_Port GPIOB
#define USB_OverCurrent_Pin GPIO_PIN_5
#define USB_OverCurrent_GPIO_Port GPIOG
#define USB_PowerSwitchOn_Pin GPIO_PIN_6
#define USB_PowerSwitchOn_GPIO_Port GPIOG
#define STLK_RX_Pin GPIO_PIN_7
#define STLK_RX_GPIO_Port GPIOG
#define STLK_TX_Pin GPIO_PIN_8
#define STLK_TX_GPIO_Port GPIOG
#define USB_SOF_Pin GPIO_PIN_8
#define USB_SOF_GPIO_Port GPIOA
#define USB_VBUS_Pin GPIO_PIN_9
#define USB_VBUS_GPIO_Port GPIOA
#define USB_ID_Pin GPIO_PIN_10
#define USB_ID_GPIO_Port GPIOA
#define USB_DM_Pin GPIO_PIN_11
#define USB_DM_GPIO_Port GPIOA
#define USB_DP_Pin GPIO_PIN_12
#define USB_DP_GPIO_Port GPIOA
#define TMS_Pin GPIO_PIN_13
#define TMS_GPIO_Port GPIOA
#define TCK_Pin GPIO_PIN_14
#define TCK_GPIO_Port GPIOA
#define SWO_Pin GPIO_PIN_3
#define SWO_GPIO_Port GPIOB
#define LD2_Pin GPIO_PIN_7
#define LD2_GPIO_Port GPIOB
/* USER CODE BEGIN Private defines */
/* USER CODE END Private defines */
#ifdef __cplusplus
}
#endif
#endif /* __MAIN_H */
```

main.c

```
#include "main.h"
#include <KEYPAD.h>
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
int main(void)
{
 HAL_Init();
 SystemClock_Config();
 MX_GPIO_Init();
 Keypad_Configuration();
 // 45678-1-2345678-2-2345678-3-2345678-4-2345678-5-2345678-6-2345678-7-2345678-8
 while (1)
 {
        if (Key_Pressed() == 1)
        {
                int i =  Keypad_WhichKeyIsPressed();
                GPIOB->ODR = (GPIOB->ODR & ~(GPIO_PIN_4| GPIO_PIN_5| GPIO_PIN_6| GPIO_PIN_8))
| (i <<4 & 0x070) | (i<<5 & 0x100) ;
                for(int i = 0; i < 10000; i++)
                                {
                                        continue;
                                }
        }
 }
}
/*
* STM32 IDE GENERATED DEFAULT CODE
*/
void SystemClock_Config(void)
{
 RCC_OscInitTypeDef RCC_OscInitStruct = {0};
 RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
 if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
 {
  Error_Handler();
 }
 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();
 }
```

```
 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_DIV2;
 RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
 {
   Error_Handler();
 }
}
/**
 * @brief GPIO Initialization Function
 * @param None
 * @retval None
 */
static void MX_GPIO_Init(void)
{
 GPIO_InitTypeDef GPIO_InitStruct = {0};
 /* USER CODE BEGIN MX_GPIO_Init_1 */
 /* USER CODE END MX_GPIO_Init_1 */
 /* GPIO Ports Clock Enable */
 __HAL_RCC_GPIOC_CLK_ENABLE();
 __HAL_RCC_GPIOH_CLK_ENABLE();
 __HAL_RCC_GPIOB_CLK_ENABLE();
 __HAL_RCC_GPIOG_CLK_ENABLE();
 HAL_PWREx_EnableVddIO2();
 __HAL_RCC_GPIOA_CLK_ENABLE();
 /*Configure GPIO pin Output Level */
 HAL_GPIO_WritePin(GPIOB, LD3_Pin|LD2_Pin, GPIO_PIN_RESET);
 /*Configure GPIO pin Output Level */
 HAL_GPIO_WritePin(USB_PowerSwitchOn_GPIO_Port, USB_PowerSwitchOn_Pin, GPIO_PIN_RESET);
 /*Configure GPIO pin : B1_Pin */
 GPIO_InitStruct.Pin = B1_Pin;
 GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);
 /*Configure GPIO pins : LD3_Pin LD2_Pin */
 GPIO_InitStruct.Pin = LD3_Pin|LD2_Pin;
 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
 HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
 /*Configure GPIO pin : USB_OverCurrent_Pin */
 GPIO_InitStruct.Pin = USB_OverCurrent_Pin;
 GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
 HAL_GPIO_Init(USB_OverCurrent_GPIO_Port, &GPIO_InitStruct);
 /*Configure GPIO pin : USB_PowerSwitchOn_Pin */
 GPIO_InitStruct.Pin = USB_PowerSwitchOn_Pin;
 GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
 HAL_GPIO_Init(USB_PowerSwitchOn_GPIO_Port, &GPIO_InitStruct);
 /*Configure GPIO pins : STLK_RX_Pin STLK_TX_Pin */
 GPIO_InitStruct.Pin = STLK_RX_Pin|STLK_TX_Pin;
 GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
 GPIO_InitStruct.Alternate = GPIO_AF8_LPUART1;
 HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);
 /*Configure GPIO pins : USB_SOF_Pin USB_ID_Pin USB_DM_Pin USB_DP_Pin */
 GPIO_InitStruct.Pin = USB_SOF_Pin|USB_ID_Pin|USB_DM_Pin|USB_DP_Pin;
 GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
 GPIO_InitStruct.Pull = GPIO_NOPULL;
 GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
 GPIO_InitStruct.Alternate = GPIO_AF10_OTG_FS;
 HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}
void Error_Handler(void)
{
 __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 */
```

KEYPAD.c

```
#include <KEYPAD.h>
/* ---------------------------- (Keypad_Configuration ) ------------------------------
* Initializes all the pins as needed
* Inputs: None
* Outputs: None
* Local vars: None
* -------------------------------------------------------------------------- */
void Keypad_Configuration(void)
{
	//Initialize clock and inputs and outputs and stuff
	//Enable GPIOC clock for peripheral
	RCC->AHB2ENR |= RCC_AHB2ENR_GPIOCEN;
	RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
	//set GPIO to input mode (Reset bits 6 and 7 for input mode) (00)
	GPIOC->MODER = 0;
	//INITIALIZES COLUMNS AND OUTPUTS
	GPIOB->MODER &= ~(GPIO_MODER_MODE0 | GPIO_MODER_MODE1 |
GPIO_MODER_MODE2 | GPIO_MODER_MODE3 |
GPIO_MODER_MODE4|GPIO_MODER_MODE5|GPIO_MODER_MODE6 |GPIO_MODER_MODE8);
	GPIOB->MODER |=  (GPIO_MODER_MODE0_0 | GPIO_MODER_MODE1_0 |
GPIO_MODER_MODE2_0 | GPIO_MODER_MODE3_0 |GPIO_MODER_MODE4_0|
GPIO_MODER_MODE5_0|GPIO_MODER_MODE6_0| GPIO_MODER_MODE8_0);
	//Initializes PUPDR
	GPIOC->PUPDR &= PUPDRST;
	GPIOC->PUPDR |= (GPIO_PUPDR_PUPD0_1 | GPIO_PUPDR_PUPD1_1 | GPIO_PUPDR_PUPD2_1 |
GPIO_PUPDR_PUPD3_1);
	GPIOB->PUPDR |= (GPIO_PUPDR_PUPD0_0 | GPIO_PUPDR_PUPD1_0 | GPIO_PUPDR_PUPD2_0 |
GPIO_PUPDR_PUPD3_0);
	GPIOC->OTYPER &= (GPIO_OTYPER_OT0 | GPIO_OTYPER_OT1 |GPIO_OTYPER_OT2
|GPIO_OTYPER_OT3);
	// Set very high output speed for PC0, PC1, PC2, and PC3
	GPIOB->OSPEEDR |=  (3 << GPIO_OSPEEDR_OSPEED6_Pos);
	GPIOB->OTYPER &= ~(GPIO_OTYPER_OT6);
	GPIOB->BSRR = (GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2| GPIO_PIN_3); // turn off all LEDs
	GPIOB->BRR = OUTPUT_PINS;
}
/* ---------------------------- (Key_Pressed ) ------------------------------
* Detects if a key is pressed with debounce to not bug out
* Inputs: None
* Outputs: TRUE (1) or FALSE (0) depending on if a key is pressed
* Local vars: None
* -------------------------------------------------------------------------- */
uint8_t Key_Pressed(void) {
	 for ( uint16_t idx=0; idx<SETTLE; idx++ ){// let it settle
	 if ((ROWPORTS->IDR != 0 ))  {      // Detects input (keypress)
```

```
            //for(int j = 0; j < 500; j++);   // Debounce
            if ((Debounce()) == TRUE)  return( TRUE );
          }
          else{
            return( FALSE );
          }// nope.
        }
}
/* ---------------------------- (Keypad_WhichKeyIsPressed ) --------------------
* Detects exactly which key was pressed on the keypad and returns a mapped
* value
* Inputs: None
* Outputs: Mapped binary value corresponds to what key is pressed (int)
* Local vars: iCol (int) used as counter
* ------------------------------------------------------------------------- */
int Keypad_WhichKeyIsPressed(void) {
 int iCol;
 COLPORTS->BSRR = COL_PINS;
 if(ROWPORTS->IDR == 0x1){
        for (iCol = 0; iCol < NUM_COLS; iCol++){  //Detects which column in row 1
                COLPORTS->BRR = COL_PINS;
                        if (iCol == 0){
                                COLPORTS->BSRR =  0x1;
                                if(ROWPORTS->IDR != 0){
                                                        COLPORTS->BSRR =
COL_PINS;
                                                        return 10;
                                                }
                        }
                        else if (iCol == 1){
                                COLPORTS->BSRR =  (0x1<<1);
                                if(ROWPORTS->IDR !=0){
                                                        COLPORTS->BSRR =
COL_PINS;
                                                        return 3;
                                                }
                        }
                        else if (iCol == 2){
                                COLPORTS->BSRR =(0x1<<2);
                                if(ROWPORTS->IDR !=0){
                                        COLPORTS->BSRR = COL_PINS;
                                        return 2;
                                }
                        }
                        else if (iCol == 3){
                        COLPORTS->BSRR =  (0x1<<3);
                                if(ROWPORTS->IDR !=0){
```

```
                                        COLPORTS->BSRR = COL_PINS;
                                        return 1;
                                                                }
                                }
                        }
}
else if (ROWPORTS->IDR == 0x2){
        for (iCol = 0; iCol < NUM_COLS; iCol++){ //Detects which column in row 2
        COLPORTS->BRR = COL_PINS;
                        if (iCol == 0){
                                COLPORTS->BSRR =  0x1;
                                if(ROWPORTS->IDR != 0){
                                                                COLPORTS->BSRR = COL_PINS;
                                                                return 11;
                                                        }
                        }
                        else if (iCol == 1){
                                COLPORTS->BSRR =  (0x1<<1);
                                if(ROWPORTS->IDR !=0){
                                                                COLPORTS->BSRR = COL_PINS;
                                                                return 6;
                                                        }
                        }
                        else if (iCol == 2){
                                COLPORTS->BSRR =(0x1<<2);
                                if(ROWPORTS->IDR !=0){
                                        COLPORTS->BSRR = COL_PINS;
                                        return 5;
                                }
                        }
                        else if (iCol == 3){
                        COLPORTS->BSRR =  (0x1<<3);
                                if(ROWPORTS->IDR !=0){
                                COLPORTS->BSRR = COL_PINS;
                                return 4;
                                                        }
                        }
                        else{
                                continue;
                        }
                }
}
else if (ROWPORTS->IDR == 0x4){
        for (iCol = 0; iCol < NUM_COLS; iCol++){ //Detects which column in row 3
                COLPORTS->BRR = COL_PINS;
                                if (iCol == 0){
                                        COLPORTS->BSRR =  0x1;
```

```
                                        if(ROWPORTS->IDR !=0){
                                                                COLPORTS->BSRR =
COL_PINS;
                                                                return 12;
                                                        }
                                }
                                else if (iCol == 1){
                                        COLPORTS->BSRR =  (0x1<<1);
                                        if(ROWPORTS->IDR !=0){
                                                                COLPORTS->BSRR =
COL_PINS;
                                                                return 9;
                                                        }
                                }
                                else if (iCol == 2){
                                        COLPORTS->BSRR =(0x1<<2);
                                        if(ROWPORTS->IDR !=0){
                                                COLPORTS->BSRR = COL_PINS;
                                                return 8;
                                        }
                                }
                                else if (iCol == 3){
                                COLPORTS->BSRR =  (0x1<<3);
                                        if(ROWPORTS->IDR !=0){
                                        COLPORTS->BSRR = COL_PINS;
                                        return 7;
                                                        }
                                }
                                else{
                                        continue;
                                }
                        }
  }
 else if (ROWPORTS->IDR == 0x8){
            for (iCol = 0; iCol < NUM_COLS; iCol++){  //Detects which column in row 4
                    COLPORTS->BRR = COL_PINS;
                                if (iCol == 0){
                                        COLPORTS->BSRR =  0x1;
                                        if(ROWPORTS->IDR != 0){
                                                                COLPORTS->BSRR =
COL_PINS;
                                                                return 13;
                                                        }
                                }
                                else if (iCol == 1){
                                        COLPORTS->BSRR =  (0x1<<1);
                                        if(ROWPORTS->IDR !=0){
```

```
                                                                    COLPORTS->BSRR =
COL_PINS;
                                                                    return 14;
                                                                }
                                    }
                                    else if (iCol == 2){
                                            COLPORTS->BSRR =(0x1<<2);
                                            if(ROWPORTS->IDR !=0){
                                                    COLPORTS->BSRR = COL_PINS;
                                                    return 0;
                                            }
                                    }
                                    else if (iCol == 3){
                                    COLPORTS->BSRR =  (0x1<<3);
                                            if(ROWPORTS->IDR !=0){
                                            COLPORTS->BSRR = COL_PINS;
                                            return 15;
                                                                }
                                    }
                                    else{
                                            continue;
                                    }
                            }
  }
}
/* ----------------------------- (Debounce ) ------------------------------------
* Debounces keypad pressing by making sure output is constant
* Inputs: None
* Outputs: Returns true once input stays contstant
* Local vars: prevalue to keep track if it was previously one
* ---------------------------------------------------------------------------- */
int Debounce(void){
        int preValue1;
        int preValue2;
        int preValue3;
        while ((preValue1 & preValue2 & preValue3) == 0){
                preValue1 = ROWPORTS->IDR;
                for(int i = 0; i < 200; i++);
                preValue2 = ROWPORTS->IDR;
                for(int i = 0; i < 200; i++);
                preValue3 = ROWPORTS->IDR;
                for(int i = 0; i < 200; i++);
        }
        return TRUE;
}
```

KEYPAD.h

```
/******************************************************************************
* EE 329 A2 KEYPAD INTERFACE
******************************************************************************
* @file          : KEYPAD.c
* @brief         : Runs the functions from Keypad.c and outputs onto LEDs
* project        : EE 329 S'25 Assignment 2
* authors        : Justin Rosu & Brayden Daly  (JRBD)
* version        : 0.1
* date           : 250412
* compiler       : STM32CubeIDE v.1.12.0 Build: 14980_20230301_1550 (UTC)
* target         : NUCLEO-L4A6ZG
* clocks         : 4 MHz MSI to AHB2
* @attention     : (c) 2023 STMicroelectronics.  All rights reserved.
******************************************************************************
* KEYPAD PLAN :
* set columns as outputs, rows as inputs w pulldowns
* loop:
* drive all columns HI read all rows
* if any row N is HI
*    set all columns LO
*    drive each column M HI alone
*    read row N until HI □ pressed key loc'n = N, M
* key value = 3N+M+1 for 1..9,special case for *,0,#
******************************************************************************
* KEYPAD WIRING 4 ROWS 4 COLS (pinout NUCLEO-L4A6ZG = L496ZG)
*      peripheral – Nucleo I/O
* keypad 1  COL 4 - PB0
* keypad 2  COL 3 - PB1
* keypad 3  COL 2 - PB2
* keypad 4  COL 1 - PB3
* keypad 5  ROW 4 - PC2
* keypad 6  ROW 3 - PC2
* keypad 7  ROW 2 - PC1
* keypad 8  ROW 1 - PC0
* OUTPUT LED BIT0 - PB4
* OUTPUT LED BIT1 - PB5
* OUTPUT LED BIT2 - PB6
* OUTPUT LED BIT3 - PB8
******************************************************************************
* REVISION HISTORY
* 0.1 230318 JRBD  created, wires in breadboard, no keypad
* 0.2 230410 JRBD  made code to make keypad operational
* 0.3 230413 JRBD  implemented modularity
******************************************************************************
******************************************************************************
* 45678-1-2345678-2-2345678-3-2345678-4-2345678-5-2345678-6-2345678-7-234567 */
/* USER CODE END Header */
```

```
#ifndef KEYPAD_H_
#define KEYPAD_H_
//define rows and column numbers
#define ROWS 4
#define COLS 4
#define ROWPORTS GPIOC
#define COLPORTS GPIOB
#define ROW_PINS GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3
#define COL_PINS GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3
#define NUM_ROWS 4
#define NUM_COLS 4
#define SETTLE 100
#define FALSE 0
#define TRUE 1
#define BIT0 1
#define KEY_ZERO 0
#define CODE_ZERO 0
#define NO_KEYPRESS 0
#define PUPDRST 0
#define OUTPUT_PINS GPIO_PIN_4| GPIO_PIN_5| GPIO_PIN_6| GPIO_PIN_8
#include "stm32l4xx_hal.h"
//function declaration for key pressed (return 1 for true, 0 for false)
uint8_t Key_Pressed(void);
//function declaration for key pressed (return char,)
uint32_t Which_Key_Pressed(void);
#endif
```