```
# Instructions:
# 1. Type your response to each question below the question's heading;
#    i.e., write question 1's response below "** Question 1:"
# 2. Do not modify this template;
#    this may result in your submission not being graded and a score of 0.
# 3. Do not change the file type of this file; i.e., do not change to rich
text,
#    a Word document, PDF, etc.  You will submit this text file, as a .txt
file,
#    on Canvas for this homework assignment.

** Question 1:

/* USER CODE BEGIN Header */
/**
******************************************************************************
****
*  @file           : main.c
*  @brief          : EGEC 450 - Traffic Light FSM with LCD
******************************************************************************
****
*  @attention
*
*  Copyright (c) 2025 STMicroelectronics.
*  All rights reserved.
*
*  This software is licensed under terms that can be found in the LICENSE
   file
*  in the root directory of this software component.
*
*  References:
*  [1] IEEE, "Reference Guide," ieee.org, Accessed: Aug. 18, 2025.
   [Online].
*  Available: https://journals.ieeeauthorcenter.ieee.org/wp-
   content/uploads/sites/7/IEEE_Reference_Guide.pdf
*
*  [2] OpenAI ChatGPT, "Assistance in debugging LCD initialization and
   display update issues
*  for STM32F0 and HD44780 interface," conversation with Mohamed Khalifa,
   Oct. 2025.
******************************************************************************
****
*/
/* USER CODE END Header */

/* ---------- Includes ---------- */
#include "main.h"
#include "LCD.h"
#include <stdint.h>
#include <stdio.h>

/* ---------- Prototypes ---------- */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
```

```
/* ---------- Pins ---------- */
#define NORTH_BTN_PIN  GPIO_PIN_0
#define EAST_BTN_PIN   GPIO_PIN_1
#define WALK_BTN_PIN   GPIO_PIN_2

#define NS_GREEN (1u<<0)
#define NS_YELLOW (1u<<1)
#define NS_RED (1u<<2)
#define EW_GREEN (1u<<3)
#define EW_YELLOW (1u<<4)
#define EW_RED (1u<<5)
#define PED_WALK (1u<<6)
#define PED_DONT (1u<<7)

/* ---------- Timing (ms) ---------- */
#define T_YELLOW     1000
#define T_WALK       4000
#define T_FLASH      400
#define WALK_HOLD_MS 2000
#define T_GO         2000

/* ---------- FSM ---------- */
typedef const struct {
  uint32_t Out;
  uint32_t Time;
  uint8_t  Next[8];
  const char *Name;
} ST;

enum {
  goN=0, waitN_toE, waitN_toWalk,
  goE, waitE_toN, waitE_toWalk,
  walkN_on, walkN_b1, walkN_b2, walkN_b3, walkN_b4, walkN_b5, walkN_b6,
walkN_b7, walkN_solid,
  walkE_on, walkE_b1, walkE_b2, walkE_b3, walkE_b4, walkE_b5, walkE_b6,
walkE_b7, walkE_solid
};

#define ALL_TO(s) {s,s,s,s,s,s,s,s}

/* ---------- FSM Table ---------- */
static ST FSM[] = {
  [goN] = { NS_GREEN|EW_RED|PED_DONT,  T_GO,
            { goN, waitN_toE, waitN_toE, waitN_toE, waitN_toWalk,
waitN_toWalk, waitN_toWalk, waitN_toWalk },
            "NORTH GREEN" },

  [waitN_toE]    = { NS_YELLOW|EW_RED|PED_DONT, T_YELLOW, ALL_TO(goE),
"NORTH YELLOW" },
  [waitN_toWalk] = { NS_YELLOW|EW_RED|PED_DONT, T_YELLOW,
ALL_TO(walkN_on), "NORTH->WALK" },

  [goE] = { EW_GREEN|NS_RED|PED_DONT, T_GO,
```

```
            { goE, waitE_toN, waitE_toN, waitE_toN, waitE_toWalk,
waitE_toWalk, waitE_toWalk, waitE_toWalk },
            "EAST GREEN" },

  [waitE_toN]    = { EW_YELLOW|NS_RED|PED_DONT, T_YELLOW, ALL_TO(goN),
"EAST YELLOW" },
  [waitE_toWalk] = { EW_YELLOW|NS_RED|PED_DONT, T_YELLOW,
ALL_TO(walkE_on), "EAST->WALK" },

  /* Walk from N */
  [walkN_on]    = { NS_RED|EW_RED|PED_WALK,  T_WALK,  ALL_TO(walkN_b1),
"WALK N" },
  [walkN_b1]    = { NS_RED|EW_RED|PED_DONT,  T_FLASH, ALL_TO(walkN_b2),
"WALK BLINK" },
  [walkN_b2]    = { NS_RED|EW_RED,           T_FLASH, ALL_TO(walkN_b3),
"WALK BLINK" },
  [walkN_b3]    = { NS_RED|EW_RED|PED_DONT,  T_FLASH, ALL_TO(walkN_b4),
"WALK BLINK" },
  [walkN_b4]    = { NS_RED|EW_RED,           T_FLASH, ALL_TO(walkN_b5),
"WALK BLINK" },
  [walkN_b5]    = { NS_RED|EW_RED|PED_DONT,  T_FLASH, ALL_TO(walkN_b6),
"WALK BLINK" },
  [walkN_b6]    = { NS_RED|EW_RED,           T_FLASH, ALL_TO(walkN_b7),
"WALK BLINK" },
  [walkN_b7]    = { NS_RED|EW_RED|PED_DONT,  T_FLASH, ALL_TO(walkN_solid),
"WALK BLINK" },
  [walkN_solid] = { NS_RED|EW_RED|PED_DONT,  500,     ALL_TO(goN), "WALK
SOLID" },

  /* Walk from E */
  [walkE_on]    = { NS_RED|EW_RED|PED_WALK,  T_WALK,  ALL_TO(walkE_b1),
"WALK E" },
  [walkE_b1]    = { NS_RED|EW_RED|PED_DONT,  T_FLASH, ALL_TO(walkE_b2),
"WALK BLINK" },
  [walkE_b2]    = { NS_RED|EW_RED,           T_FLASH, ALL_TO(walkE_b3),
"WALK BLINK" },
  [walkE_b3]    = { NS_RED|EW_RED|PED_DONT,  T_FLASH, ALL_TO(walkE_b4),
"WALK BLINK" },
  [walkE_b4]    = { NS_RED|EW_RED,           T_FLASH, ALL_TO(walkE_b5),
"WALK BLINK" },
  [walkE_b5]    = { NS_RED|EW_RED|PED_DONT,  T_FLASH, ALL_TO(walkE_b6),
"WALK BLINK" },
  [walkE_b6]    = { NS_RED|EW_RED,           T_FLASH, ALL_TO(walkE_b7),
"WALK BLINK" },
  [walkE_b7]    = { NS_RED|EW_RED|PED_DONT,  T_FLASH, ALL_TO(walkE_solid),
"WALK BLINK" },
  [walkE_solid] = { NS_RED|EW_RED|PED_DONT,  500,     ALL_TO(goE), "WALK
SOLID" },
};

/* ---------- Globals ---------- */
static uint8_t cur = goN;
static uint8_t prev = 0xFF;
static uint32_t walkHoldStart = 0;
```

```
static uint8_t walkReady = 0;

/* ---------- Helpers ---------- */
static void SetCarLights(uint32_t out) {
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|
                           GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5,
GPIO_PIN_RESET);
  if (out & NS_GREEN)  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0, GPIO_PIN_SET);
  if (out & NS_YELLOW) HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1, GPIO_PIN_SET);
  if (out & NS_RED)    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_SET);
  if (out & EW_GREEN)  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, GPIO_PIN_SET);
  if (out & EW_YELLOW) HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_SET);
  if (out & EW_RED)    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_SET);
}
static void SetPedLights(uint32_t out) {
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0, (out & PED_WALK)?
GPIO_PIN_SET:GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, (out & PED_DONT)?
GPIO_PIN_SET:GPIO_PIN_RESET);
}
static inline void DriveOutputs(uint32_t out) {
  SetCarLights(out);
  SetPedLights(out);
}
static inline uint8_t BtnN(void){ return !HAL_GPIO_ReadPin(GPIOA,
NORTH_BTN_PIN); }
static inline uint8_t BtnE(void){ return !HAL_GPIO_ReadPin(GPIOA,
EAST_BTN_PIN);  }
static inline uint8_t RawWalk(void){ return !HAL_GPIO_ReadPin(GPIOA,
WALK_BTN_PIN); }
static uint8_t ReadIndex(void) {
  uint8_t n = BtnN();
  uint8_t e = BtnE();
  uint8_t w = walkReady ? 1 : 0;
  return (w<<2) | (e<<1) | n;
}

/* ---------- Main ---------- */
int main(void) {
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    LCD_Init();

    LCD_Clear();
    LCD_SetCursor(0, 0);
    LCD_OutString("EGEC 450 LAB 1");
    LCD_SetCursor(1, 0);
    LCD_OutString("PART 2 STARTING");
    HAL_Delay(1500);
    LCD_Clear();

    uint8_t prev = 0xFF;
```

```
    while (1) {
        if (RawWalk()) {
            if (walkHoldStart == 0)
                walkHoldStart = HAL_GetTick();
            if (!walkReady && (HAL_GetTick() - walkHoldStart) >=
WALK_HOLD_MS)
                walkReady = 1;
        } else {
            walkHoldStart = 0;
        }

        if (cur != prev) {
            DriveOutputs(FSM[cur].Out);
            LCD_Clear();

            if (cur == goN) {
                LCD_SetCursor(0, 0);
                LCD_OutString("NORTH GREEN");
                LCD_SetCursor(1, 0);
                LCD_OutString("CARS MOVING");
            }
            else if (cur == waitN_toE || cur == waitN_toWalk) {
                LCD_SetCursor(0, 0);
                LCD_OutString("NORTH YELLOW");
                LCD_SetCursor(1, 0);
                LCD_OutString("SLOW DOWN");
            }
            else if (cur == goE) {
                LCD_SetCursor(0, 0);
                LCD_OutString("EAST GREEN");
                LCD_SetCursor(1, 0);
                LCD_OutString("CARS MOVING");
            }
            else if (cur == waitE_toN || cur == waitE_toWalk) {
                LCD_SetCursor(0, 0);
                LCD_OutString("EAST YELLOW");
                LCD_SetCursor(1, 0);
                LCD_OutString("SLOW DOWN");
            }
            else if (cur >= walkN_on && cur <= walkN_b7) {
                LCD_SetCursor(0, 0);
                LCD_OutString("WALK N");
                LCD_SetCursor(1, 0);
                LCD_OutString("PEDESTRIAN WALK");
            }
            else if (cur >= walkE_on && cur <= walkE_b7) {
                LCD_SetCursor(0, 0);
                LCD_OutString("WALK E");
                LCD_SetCursor(1, 0);
                LCD_OutString("PEDESTRIAN WALK");
            }
            else if (cur == walkN_solid || cur == walkE_solid) {
                LCD_SetCursor(0, 0);
                LCD_OutString("WALK SOLID");
```

```
                LCD_SetCursor(1, 0);
                LCD_OutString("STOP WALK");
            }
            else {
                LCD_SetCursor(0, 0);
                LCD_OutString("UNKNOWN STATE");
                LCD_SetCursor(1, 0);
                LCD_OutString("WAIT...");
            }

            prev = cur;
        }

        HAL_Delay(FSM[cur].Time);

        uint8_t idx = ReadIndex();
        uint8_t next = FSM[cur].Next[idx];

        if ((cur == goN && next == waitN_toWalk) || (cur == goE && next ==
waitE_toWalk)) {
        }
        if (next == walkN_on || next == walkE_on)
            walkReady = 0;

        cur = next;
    }
}


/* ---------- System Clock ---------- */
void SystemClock_Config(void) {
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL12;
  RCC_OscInitStruct.PLL.PREDIV = RCC_PREDIV_DIV1;
  HAL_RCC_OscConfig(&RCC_OscInitStruct);

  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK|RCC_CLOCKTYPE_PCLK1;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1);
}

/* ---------- GPIO ---------- */
static void MX_GPIO_Init(void) {
  GPIO_InitTypeDef GPIO_InitStruct = {0};
```

```
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();


  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|
                        GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5;
  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);

  GPIO_InitStruct.Pin =
GPIO_PIN_8|GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_13;
  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);

  GPIO_InitStruct.Pin = GPIO_PIN_0|GPIO_PIN_1;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  GPIO_InitStruct.Pin = NORTH_BTN_PIN|EAST_BTN_PIN|WALK_BTN_PIN;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
}

void Error_Handler(void) {
  __disable_irq();
  while (1) {}
}


** Question 2:

LCD.h:

#ifndef __LCD_H
#define __LCD_H

#include "main.h"
#include <stdint.h>

/* === LCD Pin Definitions ===
   Adjust these to match your CubeMX pinout */
#define LCD_PORT GPIOB
#define LCD_RS_Pin GPIO_PIN_8
#define LCD_E_Pin  GPIO_PIN_9
#define LCD_D4_Pin GPIO_PIN_10
#define LCD_D5_Pin GPIO_PIN_11
#define LCD_D6_Pin GPIO_PIN_12
```

```
#define LCD_D7_Pin GPIO_PIN_13

/* === Public API === */
void LCD_Init(void);
void LCD_Clear(void);
void LCD_SetCursor(uint8_t row, uint8_t col);
void LCD_OutChar(char c);
void LCD_OutString(char *str);

#endif
```

LCD.c

```
#include "LCD.h"
#include "stm32f0xx_hal.h"
#include "main.h"

/* === Internal Helpers === */
static void LCD_PulseEnable(void);
static void LCD_Send4Bits(uint8_t nibble);
static void LCD_SendCmd(uint8_t cmd);
static void LCD_SendData(uint8_t data);
static void LCD_Delay(uint32_t ms) { HAL_Delay(ms); }

/* === Initialize LCD === */
void LCD_Init(void)
{
    HAL_Delay(50);  // wait >15 ms after power-on (safe for all LCDs)

    // 4-bit init sequence (as per HD44780 datasheet)
    LCD_Send4Bits(0x03); HAL_Delay(5);
    LCD_Send4Bits(0x03); HAL_Delay(5);
    LCD_Send4Bits(0x03); HAL_Delay(5);
    LCD_Send4Bits(0x02);  // switch to 4-bit mode

    // Function set: 4-bit, 2 lines, 5x8 font
    LCD_SendCmd(0x28);
    // Display ON, cursor OFF
    LCD_SendCmd(0x0C);
    // Entry mode set: increment cursor, no shift
    LCD_SendCmd(0x06);
    // Clear display
    LCD_Clear();
}

/* === Clear display === */
void LCD_Clear(void)
{
    LCD_SendCmd(0x01);
    HAL_Delay(3);   // clear needs >1.5 ms
}

/* === Set cursor === */
void LCD_SetCursor(uint8_t row, uint8_t col)
```

```
{
    uint8_t addr = (row == 0) ? 0x00 : 0x40;
    LCD_SendCmd(0x80 | (addr + col));
}

/* === Output single char === */
void LCD_OutChar(char c)
{
    LCD_SendData((uint8_t)c);
}

/* === Output string === */
void LCD_OutString(char *s)
{
    while (*s)
        LCD_OutChar(*s++);
}

/* === Private low-level helpers === */
static void LCD_PulseEnable(void)
{
    HAL_GPIO_WritePin(LCD_PORT, LCD_E_Pin, GPIO_PIN_RESET);
    HAL_Delay(1);
    HAL_GPIO_WritePin(LCD_PORT, LCD_E_Pin, GPIO_PIN_SET);
    HAL_Delay(2);
    HAL_GPIO_WritePin(LCD_PORT, LCD_E_Pin, GPIO_PIN_RESET);
    HAL_Delay(2);
}

static void LCD_Send4Bits(uint8_t nibble)
{
    HAL_GPIO_WritePin(LCD_PORT, LCD_D4_Pin, (nibble & 0x01) ? GPIO_PIN_SET
: GPIO_PIN_RESET);
    HAL_GPIO_WritePin(LCD_PORT, LCD_D5_Pin, (nibble & 0x02) ? GPIO_PIN_SET
: GPIO_PIN_RESET);
    HAL_GPIO_WritePin(LCD_PORT, LCD_D6_Pin, (nibble & 0x04) ? GPIO_PIN_SET
: GPIO_PIN_RESET);
    HAL_GPIO_WritePin(LCD_PORT, LCD_D7_Pin, (nibble & 0x08) ? GPIO_PIN_SET
: GPIO_PIN_RESET);
    LCD_PulseEnable();
}

static void LCD_SendCmd(uint8_t cmd)
{
    HAL_GPIO_WritePin(LCD_PORT, LCD_RS_Pin, GPIO_PIN_RESET);
    LCD_Send4Bits(cmd >> 4);
    LCD_Send4Bits(cmd & 0x0F);
    LCD_Delay(2);
}

static void LCD_SendData(uint8_t data)
{
    HAL_GPIO_WritePin(LCD_PORT, LCD_RS_Pin, GPIO_PIN_SET);
    LCD_Send4Bits(data >> 4);
```

```
    LCD_Send4Bits(data & 0x0F);
    LCD_Delay(2);
}
```

** Question 3:

The shift register (74HC595) is connected to the STM32 using SPI as follows:

- MOSI â□□ SER (pin 14) for serial data,
- SCK â□□ SRCLK (pin 11) for the shift clock, and
- a GPIO pin â□□ RCLK (pin 12) for the latch clock.

OE is tied to ground and MR is tied to VCC.

** Question 4:

A second shift register (74HC595) could be used with the STM32â□□s SPI to control the LCD by daisy-chaining it with the first shift register. The same SPI lines (MOSI and SCK) would send data to both shift registers, but each device would have a separate latch (RCLK) or chip select (CS) line.

This way:

- When the microcontroller toggles the latch for the traffic-light shift register, only the LEDs update.
- When it toggles the latch for the LCD shift register, only the LCD updates.

Thatâ□□s how the STM32 can distinguish whether the SPI data is meant for the traffic lights or for the LCD.