```
# 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          : Main program body
    ******************************************************************************
    * @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.
    * If no LICENSE file comes with this software, it is provided AS-IS.
    *
    ******************************************************************************
    */
    /* USER CODE END Header */
    /* Includes ------------------------------------------------------------------
*/
    /* USER CODE BEGIN Header */
    /**
    ******************************************************************************
    * @file           : main.c
    * @brief          : Main program body
    ******************************************************************************
    */
    /* USER CODE END Header */
    /* Includes ------------------------------------------------------------------
*/
    /* USER CODE BEGIN Header */
    /**
    ******************************************************************************
    * @file           : main.c
    * @brief          : Traffic Light FSM on STM32F0Discovery
    ******************************************************************************
    */
    /* USER CODE END Header */
    /* Includes ------------------------------------------------------------------
*/
    /* USER CODE BEGIN Header */
    /**
    ******************************************************************************
    * @file           : main.c
    * @brief          : Traffic Light FSM with Crosswalk (Phase 2)
    ******************************************************************************
    */
    /* USER CODE END Header */
```

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

void SystemClock_Config(void);
static void MX_GPIO_Init(void);

#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)

#define NORTH_BTN_PIN  GPIO_PIN_0
#define EAST_BTN_PIN   GPIO_PIN_1
#define WALK_BTN_PIN   GPIO_PIN_2

#define T_GREEN   3000
#define T_YELLOW   800
#define T_WALK    3000
#define T_FLASH    400

typedef const struct State {
uint32_t Out;
uint32_t Time;
uint32_t Next[8];
} STyp;

enum {
goN=0, waitN=1, goE=2, waitE=3,
walkGreen=4,
walkBlink1=5, walkBlink2=6, walkBlink3=7,
walkBlink4=8, walkBlink5=9, walkBlink6=10, walkBlink7=11,
walkBlink8=12, walkSolid=13, backToN=14
};

STyp FSM[15] = {
{ NS_GREEN|EW_RED|PED_DONT, T_GREEN,
    {goN, waitN, goN, waitN, walkGreen, walkGreen, walkGreen, walkGreen} },

{ NS_YELLOW|EW_RED|PED_DONT, T_YELLOW,
    {goE, goE, goE, goE, walkGreen, walkGreen, walkGreen, walkGreen} },

{ EW_GREEN|NS_RED|PED_DONT, T_GREEN,
    {goE, goE, waitE, waitE, walkGreen, walkGreen, walkGreen, walkGreen} },

{ EW_YELLOW|NS_RED|PED_DONT, T_YELLOW,
    {goN, goN, goN, goN, walkGreen, walkGreen, walkGreen, walkGreen} },

{ NS_RED|EW_RED|PED_WALK, T_WALK,
    {walkBlink1,walkBlink1,walkBlink1,walkBlink1,
    walkBlink1,walkBlink1,walkBlink1,walkBlink1} },

{ NS_RED|EW_RED, T_FLASH,
    {walkBlink2,walkBlink2,walkBlink2,walkBlink2,
```

```
        walkBlink2,walkBlink2,walkBlink2,walkBlink2} },

    { NS_RED|EW_RED|PED_DONT, T_FLASH,
        {walkBlink3,walkBlink3,walkBlink3,walkBlink3,
        walkBlink3,walkBlink3,walkBlink3,walkBlink3} },

    { NS_RED|EW_RED, T_FLASH,
        {walkBlink4,walkBlink4,walkBlink4,walkBlink4,
        walkBlink4,walkBlink4,walkBlink4,walkBlink4} },

    { NS_RED|EW_RED|PED_DONT, T_FLASH,
        {walkBlink5,walkBlink5,walkBlink5,walkBlink5,
        walkBlink5,walkBlink5,walkBlink5,walkBlink5} },

    { NS_RED|EW_RED, T_FLASH,
        {walkBlink6,walkBlink6,walkBlink6,walkBlink6,
        walkBlink6,walkBlink6,walkBlink6,walkBlink6} },

    { NS_RED|EW_RED|PED_DONT, T_FLASH,
        {walkBlink7,walkBlink7,walkBlink7,walkBlink7,
        walkBlink7,walkBlink7,walkBlink7,walkBlink7} },

    { NS_RED|EW_RED, T_FLASH,
        {walkBlink8,walkBlink8,walkBlink8,walkBlink8,
        walkBlink8,walkBlink8,walkBlink8,walkBlink8} },

    { NS_RED|EW_RED|PED_DONT, T_FLASH,
        {walkSolid,walkSolid,walkSolid,walkSolid,
        walkSolid,walkSolid,walkSolid,walkSolid} },

    { NS_RED|EW_RED|PED_DONT, 500,
        {backToN,backToN,backToN,backToN,
        backToN,backToN,backToN,backToN} },

    { NS_RED|EW_RED|PED_DONT, 200,
        {goN,goN,goN,goN,goN,goN,goN,goN} }
    };

    static inline 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 inline 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 SetLights(uint32_t out) {
    SetCarLights(out);
    SetPedLights(out);
```

```
}

static uint32_t ReadInput(void) {
uint32_t idx = 0;
if(HAL_GPIO_ReadPin(GPIOA, NORTH_BTN_PIN)) idx |= 0x1;
if(HAL_GPIO_ReadPin(GPIOA, EAST_BTN_PIN))  idx |= 0x2;
if(HAL_GPIO_ReadPin(GPIOA, WALK_BTN_PIN))  idx |= 0x4;
return idx;
}

static void Delay(uint32_t ms) {
uint32_t end = HAL_GetTick() + ms;
while((int32_t)(end - HAL_GetTick()) > 0) {
    HAL_Delay(10);
}
}

int main(void) {
HAL_Init();
SystemClock_Config();
MX_GPIO_Init();

uint32_t Pt = goN;
while(1) {
    SetLights(FSM[Pt].Out);
    Delay(FSM[Pt].Time);
    uint32_t input = ReadInput();
    Pt = FSM[Pt].Next[input];
}
}

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.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
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);
}

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_0|GPIO_PIN_1;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    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_PULLDOWN;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
    }
```

** Question 2:

Submit image using the separate Canvas submission link

** Question 3:

There are 15 states:

```
    goN=0, waitN=1, goE=2, waitE=3,
    walkGreen=4,
    walkBlink1=5, walkBlink2=6, walkBlink3=7,
    walkBlink4=8, walkBlink5=9, walkBlink6=10, walkBlink7=11,
    walkBlink8=12, walkSolid=13, backToN=14
```

The number of states is defined in two places in your code:

1. The enum { ... } block, which assigns labels to each state.
2. The STyp FSM[15] = { ... }; table, which actually defines outputs, timing, and transitions for each state.

If I wanted to add or remove a state, I would modify the enum and the FSM Table.

** Question 4:

- Each state has 8 possible transitions (one for each 3-bit combination of inputs: [Walk, East, North]).
- This is required by the system specification, since your FSM must respond to every possible button input combination (no undefined cases).
- The number of transitions is defined by the Next[8] array inside the STyp struct.
- Example:

```
    typedef const struct State {
        uint32_t Out;
        uint32_t Time;
        uint32_t Next[8];
    } STyp;
```

- To change the number of transitions per state, you would modify the size of Next[] and update the transition logic accordingly.

** Question 5:

I would prove the system works correctly by collecting experimental data that shows the lights always follow the FSM rules.

What data to collect:

- Timestamped logs of each traffic light (North/East Green/Yellow/Red, Walk/Don’t

Walk).
 - Button press inputs (North, East, Walk) with timestamps.
 - The active FSM state at each point in time.

How to collect it:
 - Use a logic analyzer or oscilloscope to monitor the GPIO pins connected to the LEDs and button inputs.
 - Or, modify the code to print the current state, button input, and outputs over the UART serial port to a computer for logging.

How this proves correctness:
 - Data will show that no two conflicting green lights (North and East) are ever active at the same time.
 - Data will show that the crosswalk (Walk signal) only activates when both North and East are red.
 - Each state transition in the logs will match the designed FSM diagram exactly.

By comparing the logged experimental data with the FSM diagram, I can prove that the system implements the FSM correctly and operates safely. This ensures that the accident could not have been caused by the traffic controller software.