```c
/* 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 ------------------------------------------------------------------*/
#include "main.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

#include "OLED.h"

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

//LED
#define R_LED1_PIN GPIO_PIN_6   // PA6
#define G_LED1_PIN GPIO_PIN_7   // PA7
#define R_LED2_PIN GPIO_PIN_8   // PA8
#define G_LED2_PIN GPIO_PIN_9   // PA9
#define R_LED3_PIN GPIO_PIN_10  // PA10
#define G_LED3_PIN GPIO_PIN_11  // PA11
#define R_LED4_PIN GPIO_PIN_12  // PA12
#define G_LED4_PIN GPIO_PIN_15  // PA15
//IR
#define IR_A1_PIN  GPIO_PIN_0  // PC0
#define IR_A2_PIN  GPIO_PIN_1  // PC1
#define IR_B1_PIN  GPIO_PIN_2  // PC2
#define IR_B2_PIN  GPIO_PIN_3  // PC3
#define IR_C1_PIN  GPIO_PIN_4  // PC4
#define IR_C2_PIN  GPIO_PIN_5  // PC5
#define IR_D1_PIN  GPIO_PIN_6  // PC6
#define IR_D2_PIN  GPIO_PIN_7  // PC7


#define ALL_RED_PINS (R_LED1_PIN | R_LED2_PIN | R_LED3_PIN | R_LED4_PIN)
#define ALL_GREEN_PINS (G_LED1_PIN | G_LED2_PIN | G_LED3_PIN | G_LED4_PIN)

#define ALL_LED_MASK (R_LED1_PIN | G_LED1_PIN | R_LED2_PIN | G_LED2_PIN | \
                      R_LED3_PIN | G_LED3_PIN | R_LED4_PIN | G_LED4_PIN)

int state = 0;
int current_group = 1;
int current_state = 0;

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */

volatile uint32_t last_button_press = 0;

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */

```

```
89  //OLED Display
90  void oled_display(void){
91
92      OLED_ShowString(1,1,"Group:");
93      switch(current_group){
94          case 1: OLED_ShowString(1, 7, "A");break;
95          case 2: OLED_ShowString(1, 7, "B");break;
96          case 3: OLED_ShowString(1, 7, "C");break;
97          case 4: OLED_ShowString(1, 7, "D");break;
98          default : OLED_ShowString(1, 7, "A");break;
99        }
100
101     OLED_ShowString(2,1,"State:");
102     OLED_ShowString(2, 7, "        ");
103
104     switch(current_state){
105         case 0: OLED_ShowString(2, 7, "Normal");break;
106         case 1: OLED_ShowString(2, 7, "Busy");break;
107         case 2: OLED_ShowString(2, 7, "Carless");break;
108         default : OLED_ShowString(2, 7, "Normal");break;
109       }
110
111     OLED_ShowString(3,1,"Time:");
112     OLED_ShowString(3, 7, "        ");
113     switch(current_state){
114         case 0: OLED_ShowString(3, 7, "5s");break;
115         case 1: OLED_ShowString(3, 7, "5-15s");break;
116         case 2: OLED_ShowString(3, 7, "3s");break;
117         default : OLED_ShowString(3, 7, "5s");break;
118       }
119
120 }
121
122 //ALL RED ON
123 void ALL_RED_ON(void) {
124
125     GPIOA->ODR &= ~ALL_GREEN_PINS;
126     GPIOA->ODR |= ALL_RED_PINS;
127     HAL_Delay(2000);
128
129 }
130
131 //ALL LED ON for 2s
132 void ALL_LED_ON(void) {
133
134     GPIOA->ODR |= ALL_LED_MASK;
135     HAL_Delay(2000);
136
137 }
138
139
140 // Red-ON Green-OFF
141 void R_LED_ON(int group) {
142     switch(group) {
143         case 1:  // A
144             GPIOA->ODR &= ~G_LED1_PIN;  // G OFF
145             GPIOA->ODR |= R_LED1_PIN;   // R ON
146             break;
147         case 2:  // B
148             GPIOA->ODR &= ~G_LED2_PIN;  // G OFF
149             GPIOA->ODR |= R_LED2_PIN;   // R ON
150             break;
151         case 3:  // C
152             GPIOA->ODR &= ~G_LED3_PIN;  // G OFF
153             GPIOA->ODR |= R_LED3_PIN;   // R ON
154             break;
155         case 4:  // D
156             GPIOA->ODR &= ~G_LED4_PIN;  // G OFF
157             GPIOA->ODR |= R_LED4_PIN;   // R ON
158             break;
159     }
160 }
161
162 // Red-OFF Green-ON
163 void R_LED_OFF(int group) {
164     switch(group) {
165         case 1:  // A
166             GPIOA->ODR &= ~R_LED1_PIN;  // R OFF
167             GPIOA->ODR |= G_LED1_PIN;   // G ON
168             break;
169         case 2:  // B
170             GPIOA->ODR &= ~R_LED2_PIN;  // R OFF
171             GPIOA->ODR |= G_LED2_PIN;   // G ON
172             break;
173         case 3:  // C
174             GPIOA->ODR &= ~R_LED3_PIN;  // R OFF
175             GPIOA->ODR |= G_LED3_PIN;   // G ON
176             break;
```

```
177             case 4:  // D
178                 GPIOA->ODR &= ~R_LED4_PIN;  // R OFF
179                 GPIOA->ODR |= G_LED4_PIN;   // G ON
180                 break;
181         }
182     }
183 
184     // Green-ON Red-OFF
185     void G_LED_ON(int group) {
186         switch(group) {
187             case 1:  // A
188                 GPIOA->ODR &= ~R_LED1_PIN;  // R OFF
189                 GPIOA->ODR |= G_LED1_PIN;   // G ON
190                 break;
191             case 2:  // B
192                 GPIOA->ODR &= ~R_LED2_PIN;  // R OFF
193                 GPIOA->ODR |= G_LED2_PIN;   // G ON
194                 break;
195             case 3:  // C
196                 GPIOA->ODR &= ~R_LED3_PIN;  // R OFF
197                 GPIOA->ODR |= G_LED3_PIN;   // G ON
198                 break;
199             case 4:  // D
200                 GPIOA->ODR &= ~R_LED4_PIN;  // R OFF
201                 GPIOA->ODR |= G_LED4_PIN;   // G ON
202                 break;
203         }
204     }
205 
206     // Green-OFF Red-ON
207     void G_LED_OFF(int group) {
208         switch(group) {
209             case 1:  // A
210                 GPIOA->ODR &= ~G_LED1_PIN;  // G OFF
211                 GPIOA->ODR |= R_LED1_PIN;   // R ON
212                 break;
213             case 2:  // B
214                 GPIOA->ODR &= ~G_LED2_PIN;  // G OFF
215                 GPIOA->ODR |= R_LED2_PIN;   // R ON
216                 break;
217             case 3:  // C
218                 GPIOA->ODR &= ~G_LED3_PIN;  // G OFF
219                 GPIOA->ODR |= R_LED3_PIN;   // R ON
220                 break;
221             case 4:  // D
222                 GPIOA->ODR &= ~G_LED4_PIN;  // G OFF
223                 GPIOA->ODR |= R_LED4_PIN;   // R ON
224                 break;
225         }
226     }
227 
228     //Blink G LED
229     void blink(int group) {
230         switch(state) {
231             case 0:  // normal:blink once
232                 // G OFF 0.2s
233                 switch(group) {
234                     case 1: GPIOA->ODR &= ~G_LED1_PIN; break;
235                     case 2: GPIOA->ODR &= ~G_LED2_PIN; break;
236                     case 3: GPIOA->ODR &= ~G_LED3_PIN; break;
237                     case 4: GPIOA->ODR &= ~G_LED4_PIN; break;
238                 }
239                 HAL_Delay(200);
240 
241                 // G ON 0.2s
242                 switch(group) {
243                     case 1: GPIOA->ODR |= G_LED1_PIN; break;
244                     case 2: GPIOA->ODR |= G_LED2_PIN; break;
245                     case 3: GPIOA->ODR |= G_LED3_PIN; break;
246                     case 4: GPIOA->ODR |= G_LED4_PIN; break;
247                 }
248                 HAL_Delay(200);
249                 break;
250 
251             case 1:  // busy:blink twice
252                 for(int i = 0; i < 2; i++) {
253                     //G OFF 0.2s
254                     switch(group) {
255                         case 1: GPIOA->ODR &= ~G_LED1_PIN; break;
256                         case 2: GPIOA->ODR &= ~G_LED2_PIN; break;
257                         case 3: GPIOA->ODR &= ~G_LED3_PIN; break;
258                         case 4: GPIOA->ODR &= ~G_LED4_PIN; break;
259                     }
260                     HAL_Delay(500);
261 
262                     // G ON 0.2s
263                     switch(group) {
264                         case 1: GPIOA->ODR |= G_LED1_PIN; break;
```

```
                    case 2: GPIOA->ODR |= G_LED2_PIN; break;
                    case 3: GPIOA->ODR |= G_LED3_PIN; break;
                    case 4: GPIOA->ODR |= G_LED4_PIN; break;
                }
                HAL_Delay(500);
            }
            break;

        // case 2 (carless) :no blink
    }
}


void emergency(int group)
{
    OLED_ShowString(4, 4, "Acrossing! ");
    R_LED_ON(group);
}


// Read IR1
uint8_t IR1(int group) {
    uint16_t pin;
    switch(group) {
        case 1: pin = IR_A1_PIN; break;  // A
        case 2: pin = IR_B1_PIN; break;  // B
        case 3: pin = IR_C1_PIN; break;  // C
        case 4: pin = IR_D1_PIN; break;  // D
        default: return 0;
    }
    // No car-1; Have car-0
    return (GPIOC->IDR & pin) ? 1 : 0;
}

// Read IR2
uint8_t IR2(int group) {
    uint16_t pin;
    switch(group) {
        case 1: pin = IR_A2_PIN; break;  // A
        case 2: pin = IR_B2_PIN; break;  // B
        case 3: pin = IR_C2_PIN; break;  // C
        case 4: pin = IR_D2_PIN; break;  // D
        default: return 0;
    }
    // No car-1; Have car-0
    return (GPIOC->IDR & pin) ? 1 : 0;
}

// Prediction
void Predict(int group) {

    uint8_t sensor1 = IR1(group);
    uint8_t sensor2 = IR2(group);

    // Both car detected
    if (sensor1 == 0 && sensor2 == 0) {
        state = 1;  // Set busy mode
    }
    else if (sensor1 == 1 && sensor2 == 1) {
        state = 2;  // Set no car mode
    }
    // default
    else {
        state = 0;  // Set normal mode
    }
}


// Normal state
void normal(int group) {
    G_LED_ON(group);      // G ON
    HAL_Delay(200);
    blink(group);
    HAL_Delay(5000);      // 5s
    R_LED_ON(group);      // R ON
}

// Busy state
void busy(int group) {
    uint8_t count = 0;
    uint32_t total_time = 5000;

    G_LED_ON(group);  // G ON
    HAL_Delay(200);
    blink(group);

    // Delay 5s
    HAL_Delay(5000);
```

```
353
354         // cheak IR to expand time
355         while (count < 5) {  // max 5 times
356             if (!IR1(group)) {  // IR=0 car detected
357                 HAL_Delay(200);  // delay 0.2s
358                 if (!IR1(group)) {  // detect IR1 again
359                     HAL_Delay(2000);  // total time +2s
360                     total_time += 2000;
361                     count++;
362
363                     // cheack max time=15s?
364                     if (total_time >= 15000) break;
365                 } else {
366                     break;  // 2nd detect "1" means it's a mistake
367                 }
368             } else {
369                 break;  // 1st detect "1" means not busy
370             }
371         }
372
373         R_LED_ON(group);  // R ON
374     }
375
376     // No Car
377     void carless(int group) {
378
379         G_LED_ON(group);      // G ON
380         //no blink
381         HAL_Delay(3000);      // 3s
382         R_LED_ON(group);      // R ON
383
384     }
385
386
387
388     /* USER CODE END PFP */
389
390     /* Private user code ---------------------------------------------------------*/
391     /* USER CODE BEGIN 0 */
392
393     /* USER CODE END 0 */
394
395     /**
396       * @brief  The application entry point.
397       * @retval int
398       */
399     int main(void)
400     {
401
402       /* USER CODE BEGIN 1 */
403
404
405
406       /* USER CODE END 1 */
407
408       /* MCU Configuration--------------------------------------------------------*/
409
410       /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
411       HAL_Init();
412
413       /* USER CODE BEGIN Init */
414
415       __disable_irq();
416
417       // GPIOA, GPIOB, GPIOC, GPIOH clk
418       RCC->AHB1ENR |= (1 << 0) | (1 << 1) | (1 << 2) | (1 << 7);
419
420       //GPIOA: ALL LED output
421       GPIOA->MODER = 1 << (6*2) | 1 << (7*2) | 1 << (8*2) | 1 << (9*2) | 1 << (10*2) | 1 << (11*2) | 1 << (12*2) | 1 <<
    (15*2);
422       GPIOA->OTYPER = 0;
423       GPIOA->PUPDR = 0;
424
425       //GPIOC: ALL IR input
426       GPIOC->MODER = 0;
427       GPIOC->PUPDR = 0;
428
429       //GPIOB: SCL-PB13/SDA-PB14 output, botton(PB4-7) input
430       GPIOB->MODER = 1 << (13*2) | 1 << (14*2);
431       GPIOB->OTYPER = 0;
432       GPIOB->PUPDR = 1 << (13*2) | 1 << (14*2) | 1 << (4*2) | 1 << (5*2) | 1 << (6*2) | 1 << (7*2);
433
434       //Interrupt
435       RCC->APB2ENR |= 0x4000;
436       // EXTI4 (PB4)
437       SYSCFG->EXTICR[1] |= 0x0001;
438       EXTI->IMR |= 0x0010;
439       EXTI->FTSR |= 0x0010;
```

```
440
441     // EXTI5 (PB5)
442     SYSCFG->EXTICR[1] |= 0x0010;
443     EXTI->IMR |= 0x0020;
444     EXTI->FTSR |= 0x0020;
445
446     // EXTI6 (PB6)
447     SYSCFG->EXTICR[1] |= 0x0100;
448     EXTI->IMR |= 0x0040;
449     EXTI->FTSR |= 0x0040;
450
451     // EXTI7 (PB7)
452     SYSCFG->EXTICR[1] |= 0x1000;
453     EXTI->IMR |= 0x0080;
454     EXTI->FTSR |= 0x0080;
455
456     NVIC_EnableIRQ(EXTI4_IRQn);
457     NVIC_EnableIRQ(EXTI9_5_IRQn);
458     __enable_irq();
459
460
461     /* USER CODE END Init */
462
463     /* Configure the system clock */
464     SystemClock_Config();
465
466     /* USER CODE BEGIN SysInit */
467
468     /* USER CODE END SysInit */
469
470     /* Initialize all configured peripherals */
471   //  MX_GPIO_Init();
472     MX_USART2_UART_Init();
473     /* USER CODE BEGIN 2 */
474
475
476     OLED_Init();
477     OLED_ShowString(2,2,"Initializing...");
478     HAL_Delay(1000);
479
480
481     int group = 1;
482
483     ALL_LED_ON();
484     ALL_RED_ON();
485
486     OLED_Clear();
487
488     /* USER CODE END 2 */
489
490     /* Infinite loop */
491     /* USER CODE BEGIN WHILE */
492     while (1)
493     {
494       /* USER CODE END WHILE */
495
496       /* USER CODE BEGIN 3 */
497
498
499       for(;group<=4;group++){
500
501         current_group = group;
502
503         Predict(group);
504
505         current_state = state;
506         oled_display();
507
508         switch(state) {
509           case 0:  // normal
510             normal(group);
511             break;
512           case 1:  // busy
513             busy(group);
514             break;
515           case 2:  // no car
516             carless(group);
517             break;
518           default: // normal
519             normal(group);
520             break;}
521         OLED_Clear();
522       }
523       group = 1;
524
525
526     }
527     /* USER CODE END 3 */
```

```
528   }
529
530   /**
531     * @brief System Clock Configuration
532     * @retval None
533     */
534   void SystemClock_Config(void)
535   {
536     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
537     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
538
539     /** Configure the main internal regulator output voltage
540     */
541     __HAL_RCC_PWR_CLK_ENABLE();
542     __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);
543
544     /** Initializes the RCC Oscillators according to the specified parameters
545     * in the RCC_OscInitTypeDef structure.
546     */
547     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
548     RCC_OscInitStruct.HSIState = RCC_HSI_ON;
549     RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
550     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
551     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
552     RCC_OscInitStruct.PLL.PLLM = 16;
553     RCC_OscInitStruct.PLL.PLLN = 336;
554     RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
555     RCC_OscInitStruct.PLL.PLLQ = 2;
556     RCC_OscInitStruct.PLL.PLLR = 2;
557     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
558     {
559       Error_Handler();
560     }
561
562     /** Initializes the CPU, AHB and APB buses clocks
563     */
564     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
565                                   |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
566     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
567     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
568     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
569     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
570
571     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
572     {
573       Error_Handler();
574     }
575   }
576
577   /* USER CODE BEGIN 4 */
578
579
580   void EXTI4_IRQHandler(void)
581   {
582
583     if (EXTI->PR & 1<<4) {
584
585       EXTI->PR = 1<<4;
586
587       // shake inspection
588       uint32_t now = HAL_GetTick();
589       if (now - last_button_press < 300) {
590         return;
591       }
592       last_button_press = now;
593
594       // if "R ON", do nothing
595       if (GPIOA->ODR & R_LED1_PIN) {
596         return;
597       }
598
599       // if "G ON", turn to RED and display
600       if (GPIOA->ODR & G_LED1_PIN) {
601         emergency(1);
602       }
603     }
604
605   }
606
607   void EXTI9_5_IRQHandler(void)
608   {
609     uint32_t now = HAL_GetTick();
610     //GroupB
611     if (EXTI->PR & 1<<5) {
612
613       EXTI->PR = 1<<5;
614
615       // shake inspection
```

```
      uint32_t now = HAL_GetTick();
      if (now - last_button_press < 300) {
        return;
      }
      last_button_press = now;

      // if "R ON", do nothing
      if (GPIOA->ODR & R_LED2_PIN) {
        return;
      }

      // if "G ON", turn to RED and display
      if (GPIOA->ODR & G_LED2_PIN) {
        emergency(2);
      }
    }

    //GroupC
    if (EXTI->PR & 1<<6) {

      EXTI->PR = 1<<6;

      // shake inspection
      uint32_t now = HAL_GetTick();
      if (now - last_button_press < 300) {
        return;
      }
      last_button_press = now;

      // if "R ON", do nothing
      if (GPIOA->ODR & R_LED3_PIN) {
        return;
      }

      // if "G ON", turn to RED and display
      if (GPIOA->ODR & G_LED3_PIN) {
        emergency(3);
      }
    }

    //GroupD
    if (EXTI->PR & 1<<7) {

      EXTI->PR = 1<<7;

      // shake inspection
      uint32_t now = HAL_GetTick();
      if (now - last_button_press < 300) {
        return;
      }
      last_button_press = now;

      // if "R ON", do nothing
      if (GPIOA->ODR & R_LED4_PIN) {
        return;
      }

      // if "G ON", turn to RED and display
      if (GPIOA->ODR & G_LED4_PIN) {
        emergency(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
```

```
704     *         where the assert_param error has occurred.
705     * @param  file: pointer to the source file name
706     * @param  line: assert_param error line source number
707     * @retval None
708     */
709   void assert_failed(uint8_t *file, uint32_t line)
710   {
711     /* USER CODE BEGIN 6 */
712     /* User can add his own implementation to report the file name and line number,
713        ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
714     /* USER CODE END 6 */
715   }
716   #endif /* USE_FULL_ASSERT */
717
```