Pametni Parking sa STM32F407

Radili:
Azur Jusić
Rijad Mecić
Senad Efendić

Opis projekta

- Parking sa rampom na ulaz/izlaz -u
- Ako vozilo dođe na ulaz/izlaz sensor to detektuje
- Motor otvori rampu, čeka 5 sekundi,
  zatvori rampu, čeka 3 sekunde
- Dispej pokazuje koliko slobodnih mjesta ima na parkingu
- Trenutno maksimalno 2 mjesta za parking
- Ako vozilo se nalazi na mjestu sensor to detektuje
- Lampice svijetle crveno ako je mjesto zauzeto, a zeleno ako je mjesto slobodno

Korištene komponente:

- STM32F407 board
- HC-SR04 ultrazvučni sensor x3
- 28BYJ-48 Stepper motor
- ULN2003 Driver
- LCD i2C 16x2 display
- 2 Crvene i 2 zelene LED diode
- 150 Ohm x4
- Jumper žice(M-M,F-F)
- Mini breadboard

STM32F407VG ploča

- STM32F407VG je ploča mikrokontrolera zasnovana na STM32F4 seriji mikrokontrolera. Posjeduje 16-bitni Cortex-M4 CPU koji radi na 84 MHz, jedinicu s pomičnim zarezom (FPU) i Arm Cortex-M0+ koprocesor. Ploča također uključuje 1 GB flash memorije, 512 KB SRAM-a i razne interfejse i periferne uređaje.
- STM32F407VG je svestrana i moćna ploča mikrokontrolera koju smo koristiti da bi kontrolisali sve druge komponente u ovom projektu

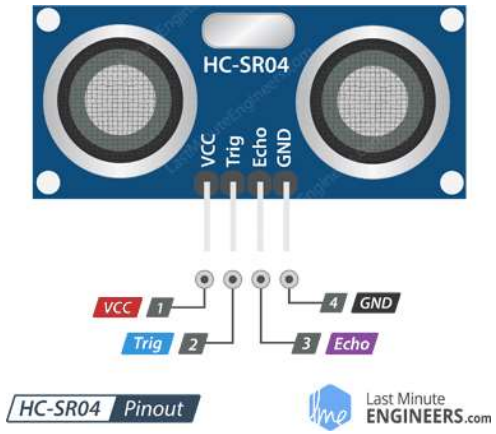
STM32F407VG ploča

- Neke od ključnih karakteristika STM32F407VG ploče uključuju:
- USB 2.0 host and device interfaces
- SD/MMC card interface
- CAN bus interface
- I2C interface
- SPI interface
- UART interface
- PWM generation
- ADC and DAC interfaces
- Timers and counters
- GPIOs

HC-SR04 Ultrazvučni Senzor

- Ultrazvučni senzor udaljenosti HC-SR04 se sastoji od dvije ultrazvučne sonde.
- Ovaj senzor pruža odličnu beskontaktnu detekciju dometa između 2 cm i 400 cm sa tačnošću od 3 mm.
- Trig (Trigger) pin se koristi za aktiviranje ultrazvučnih zvučnih impulsa. Postavljanjem ovog pina na HIGH na 10µs, senzor pokreće ultrazvučni puls.
- Echo pin prelazi na high kada se ultrazvučni puls prenese i ostaje high sve dok senzor ne primi eho, nakon čega opada na low. Mjerenjem vremena kada Echo pin ostaje high, udaljenost se može izračunati.

HC-SR04 Ultrazvučni Senzor



- 3 ova senzora koristimo da bi detektovali prisustvo vozila u različitim pozicijama
- Ima 4 Pina VCC, Trig, Echo i Gnd
- VCC spajamo na 5V napon iz stm32 ploče
- Trig je ulazni pin, na našoj ploči smo ih spojili na izlaze E7, E9 i E11
- Echo izlazni pin na našoj ploči smo ih spojili na ulaze E8, E10 i E12
- Gnd spajamo na GND iz stm32 ploče

LCD i2c 16X2 Displej

- 16x2 LCD I2C displej je mali displej male snage koji se obično koristi u ugrađenim sistemima i drugim malim elektronskim projektima.
- 16x2 znakova znači da može prikazati do 16 znakova u dva reda
- Sardži PCF8574 koji je opće namjenski dvosmjerni 8-bitni I/O port ekspander koji koristi I2C protokol.

LCD i2c 16X2 Displej

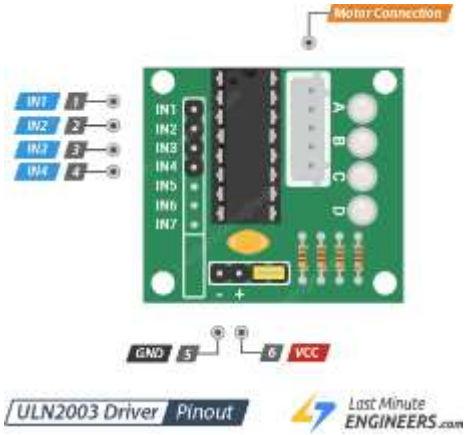
- I2C (Inter-Integrated Circuit) protokol je protokol koji omogućava komunikaciju i kontrolu uređaja povezanih na jednu magistralu. To je multi-master protokol, što znači da bilo koji uređaj na sabirnici može djelovati kao master i inicirati komunikaciju s drugim uređajima.
- Displej je povezan sa mikrokontrolerom hosta preko I2C protokola, što omogućava laku komunikaciju i kontrolu. Podatkovne linije SDA i SCL su spojene na B7 i B6 respektivno.

  VCC je spojen na 5V ploče, a GND na GND ploče.

28BYJ-48 Stepper motor i ULN2003 Driver Board

- 28BYJ-48 je koračni motor visoki performansi koji se obično koristi CNC mašinama i drugim industrijskim aplikacijama. To je 4-fazni motor koji se odlikuje velikom gustinom obrtnog momenta i velikom brzinom.
- Motor je dovoljno slab da ne zahtijeva dodatno napajanje izuzev 5V iz ploče
- Koristimo ga za otvaranje i zatvaranje rampe

28BYJ-48 Stepper motor i ULN2003 Driver Board



- ULN2003 modul nam omogućava upravljanje step motorom tj. podešavanje smjera i brzine okreatanja motora, preko pinova IN1,IN2,IN3,IN4
- GND pin spajamo na GND pin stm32 ploče
- VCC spajamo na 5V napon iz stm32 ploče
- Pinove spajamo na sljedeći način
- IN1->B15 , IN2->B14 , IN3->B13 , IN4->B12

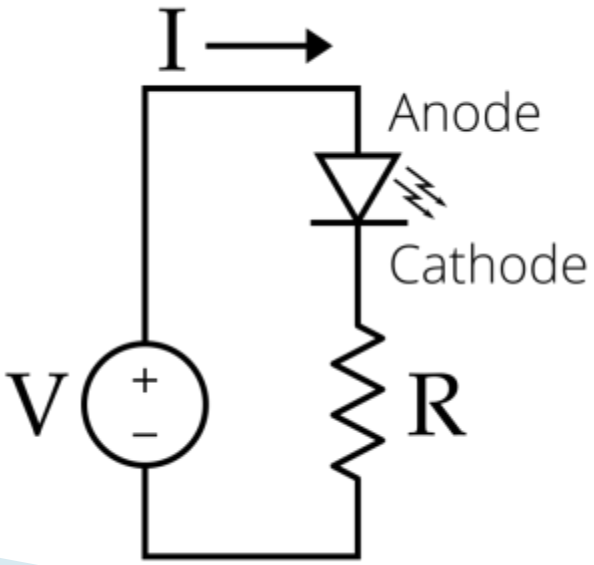
LED diode, otpornici i ostalo

- Za indikaciju okupacije parking mjesta koristimo LED diode
- Za jedno mjesto postoje 2 diode,
- jedna crvena, jedna zelena
- Crvena svijetli kada mjesto nije zauzeto, a zelena kada jeste
- Pinovi za mjesto 1: Pinovi za mjesto 2:
- Zelena->C6 Zelena->C8
- Crvena->C7 Crvena->C9

LED diode, otpornici i ostalo

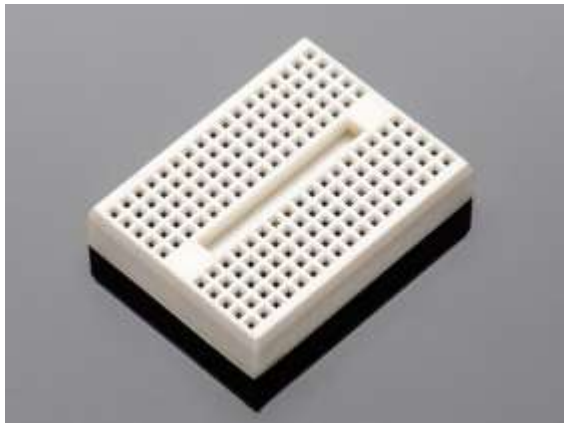
- Diode su spojene u breadboard u seriji sa jednim 150 Ohm otpornikom

- Struja kroz diodu je računata sljedečim izrazom I=(VCC-Vrn)/R Vrn-radni napon
- I=(5V-2.5V)/150 Ω=16.67 mA (MAX 25mA)

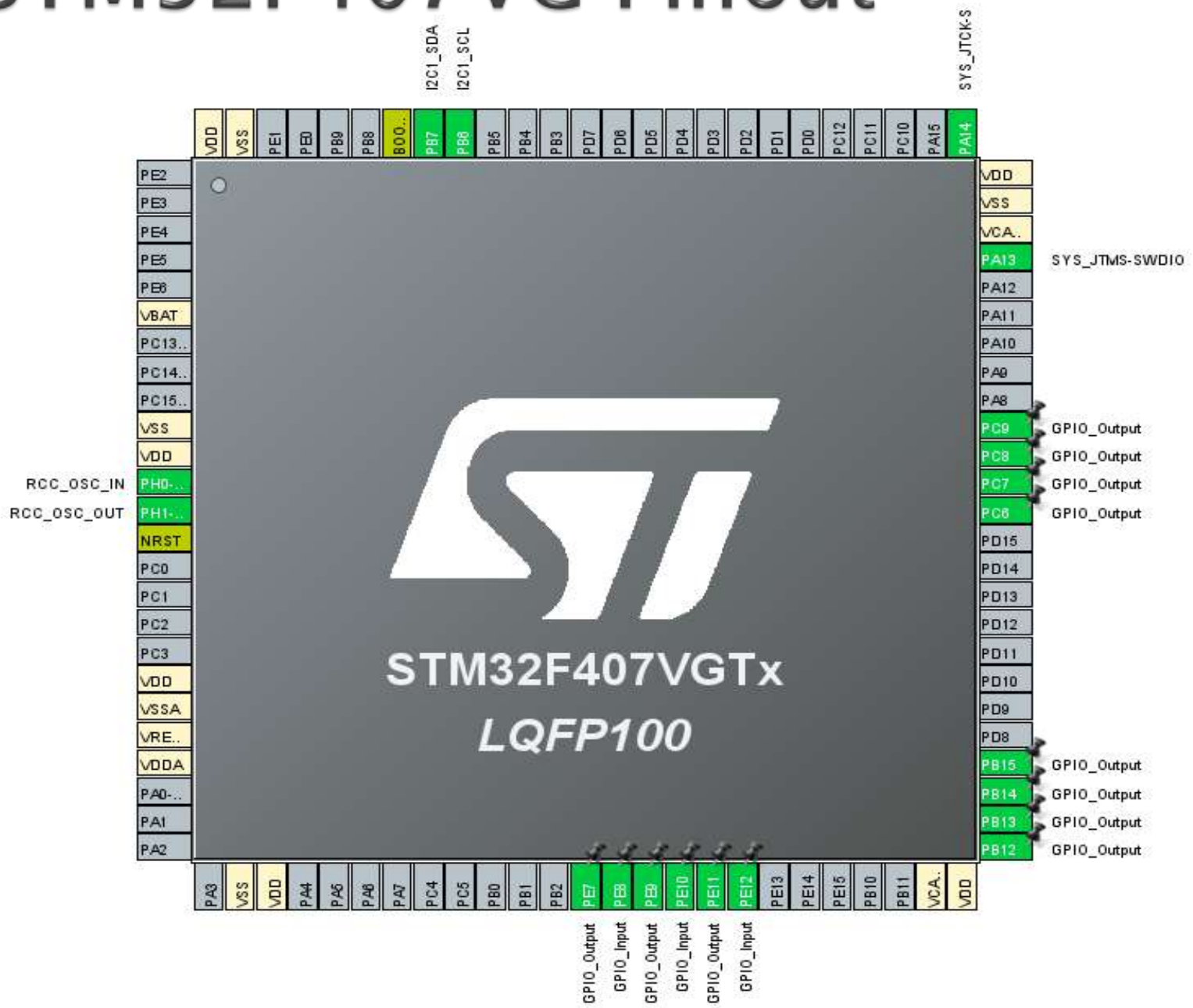


LED diode, otpornici i ostalo

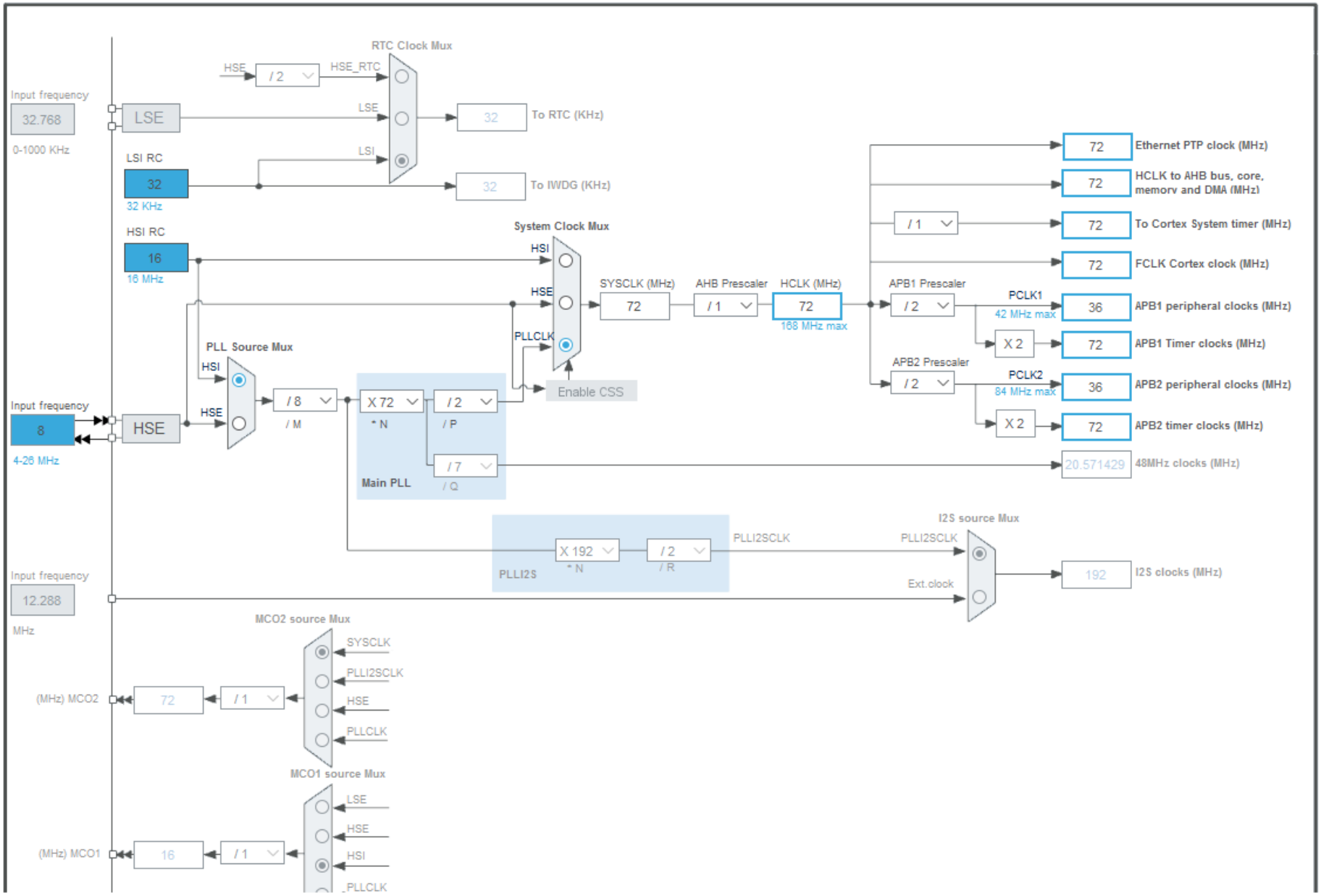
- Za spajanje svih komponenti međusobno koristimo F-F i M-M jumper žice i mini breadboard.Žice su usklađene bojama.



STM32F407VG Pinout



Clock konfiguracija



main.h

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.h
  * @brief          : Header for main.c file.
  *                   This file contains the common defines of the application.
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 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 */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"

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

/* USER CODE END Includes */

/* Exported types ------------------------------------------------------------*/
/* USER CODE BEGIN ET */

/* USER CODE END ET */
```

```
/* Exported constants --------------------------------------------------------*/
/* USER CODE BEGIN EC */

/* USER CODE END EC */

/* Exported macro ------------------------------------------------------------*/
/* USER CODE BEGIN EM */

/* USER CODE END EM */

/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);

/* USER CODE BEGIN EFP */

/* USER CODE END EFP */

/* Private defines -----------------------------------------------------------*/

/* USER CODE BEGIN Private defines */

/* USER CODE END Private defines */

#ifdef __cplusplus
}
#endif

#endif /* __MAIN_H */
```

peripheral_init.h

```
#ifndef INC_PERIPHERAL_INIT_H_

#define INC_PERIPHERAL_INIT_H_

#include "stm32f4xx_hal.h"


extern I2C_HandleTypeDef hi2c1;

extern TIM_HandleTypeDef htim1;

extern TIM_HandleTypeDef htim2;


void peripheral_init();




#endif /* INC_PERIPHERAL_INIT_H_ */
```

peripheral_init.c

```
#include "peripheral_init.h"
#include "main.h"

I2C_HandleTypeDef hi2c1;
TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;



void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM2_Init(void);
static void MX_I2C1_Init(void);

void peripheral_init()
{

HAL_Init();
SystemClock_Config();
MX_GPIO_Init();
MX_TIM1_Init();
MX_TIM2_Init();
MX_I2C1_Init();

}
```

```
/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  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.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 72;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 7;
```

```
  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_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{
```

```
  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}
```

```
/**
  * @brief TIM1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM1_Init(void)
{

  /* USER CODE BEGIN TIM1_Init 0 */

  /* USER CODE END TIM1_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM1_Init 1 */

  /* USER CODE END TIM1_Init 1 */
  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 71;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 65535;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
```

```
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */

}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{
```

```
/* USER CODE BEGIN TIM2_Init 0 */

/* USER CODE END TIM2_Init 0 */

TIM_ClockConfigTypeDef sClockSourceConfig = {0};
TIM_MasterConfigTypeDef sMasterConfig = {0};

/* USER CODE BEGIN TIM2_Init 1 */

/* USER CODE END TIM2_Init 1 */
htim2.Instance = TIM2;
htim2.Init.Prescaler = 71;
htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
htim2.Init.Period = 4294967295;
htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
{
  Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
{
  Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
```

```
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */

}

/**
  * @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_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
```

```
  /*Configure GPIO pin Output Level */

  HAL_GPIO_WritePin(GPIOE, GPIO_PIN_7|GPIO_PIN_9|GPIO_PIN_11, GPIO_PIN_RESET);


  /*Configure GPIO pin Output Level */

  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15,
GPIO_PIN_RESET);


  /*Configure GPIO pin Output Level */

  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_9,
GPIO_PIN_RESET);


  /*Configure GPIO pins : PE7 PE9 PE11 */

  GPIO_InitStruct.Pin = GPIO_PIN_7|GPIO_PIN_9|GPIO_PIN_11;

  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;

  GPIO_InitStruct.Pull = GPIO_NOPULL;

  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;

  HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);


  /*Configure GPIO pins : PE8 PE10 PE12 */

  GPIO_InitStruct.Pin = GPIO_PIN_8|GPIO_PIN_10|GPIO_PIN_12;

  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;

  GPIO_InitStruct.Pull = GPIO_NOPULL;

  HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);


  /*Configure GPIO pins : PB12 PB13 PB14 PB15 */

  GPIO_InitStruct.Pin = GPIO_PIN_12|GPIO_PIN_13|GPIO_PIN_14|GPIO_PIN_15;

  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 pins : PC6 PC7 PC8 PC9 */
  GPIO_InitStruct.Pin = GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_9;
  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);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}
```

lcd_i2c.h

```
#ifndef LIQUIDCRYSTAL_I2C_H_
#define LIQUIDCRYSTAL_I2C_H_

#include "stm32f4xx_hal.h"

/* Command */
#define LCD_CLEARDISPLAY 0x01
#define LCD_RETURNHOME 0x02
#define LCD_ENTRYMODESET 0x04
#define LCD_DISPLAYCONTROL 0x08
#define LCD_CURSORSHIFT 0x10
#define LCD_FUNCTIONSET 0x20
#define LCD_SETCGRAMADDR 0x40
#define LCD_SETDDRAMADDR 0x80

/* Entry Mode */
#define LCD_ENTRYRIGHT 0x00
#define LCD_ENTRYLEFT 0x02
#define LCD_ENTRYSHIFTINCREMENT 0x01
#define LCD_ENTRYSHIFTDECREMENT 0x00

/* Display On/Off */
#define LCD_DISPLAYON 0x04
#define LCD_DISPLAYOFF 0x00
#define LCD_CURSORON 0x02
#define LCD_CURSOROFF 0x00
#define LCD_BLINKON 0x01
#define LCD_BLINKOFF 0x00

/* Cursor Shift */
#define LCD_DISPLAYMOVE 0x08
#define LCD_CURSORMOVE 0x00
#define LCD_MOVERIGHT 0x04
#define LCD_MOVELEFT 0x00

/* Function Set */
#define LCD_8BITMODE 0x10
#define LCD_4BITMODE 0x00
#define LCD_2LINE 0x08
#define LCD_1LINE 0x00
#define LCD_5x10DOTS 0x04
#define LCD_5x8DOTS 0x00
```

```
/* Backlight */
#define LCD_BACKLIGHT 0x08
#define LCD_NOBACKLIGHT 0x00

/* Enable Bit */
#define ENABLE 0x04

/* Read Write Bit */
#define RW 0x0

/* Register Select Bit */
#define RS 0x01

/* Device I2C Address */
#define DEVICE_ADDR     (0x27 << 1)

void HD44780_Init(uint8_t rows);
void HD44780_Clear();
void HD44780_Home();
void HD44780_NoDisplay();
void HD44780_Display();
void HD44780_NoBlink();
void HD44780_Blink();
void HD44780_NoCursor();
void HD44780_Cursor();
void HD44780_ScrollDisplayLeft();
void HD44780_ScrollDisplayRight();
void HD44780_PrintLeft();
void HD44780_PrintRight();
void HD44780_LeftToRight();
void HD44780_RightToLeft();
void HD44780_ShiftIncrement();
void HD44780_ShiftDecrement();
void HD44780_NoBacklight();
void HD44780_Backlight();
void HD44780_AutoScroll();
void HD44780_NoAutoScroll();
void HD44780_CreateSpecialChar(uint8_t, uint8_t[]);
void HD44780_PrintSpecialChar(uint8_t);
void HD44780_SetCursor(uint8_t, uint8_t);
void HD44780_SetBacklight(uint8_t new_val);
void HD44780_LoadCustomCharacter(uint8_t char_num, uint8_t *rows);
void HD44780_PrintStr(const char[]);

#endif /* LIQUIDCRYSTAL_I2C_H_ */
```

lcd_i2c.c

```
#include "lcd_i2c.h"


extern I2C_HandleTypeDef hi2c1;

uint8_t dpFunction;
uint8_t dpControl;
uint8_t dpMode;
uint8_t dpRows;
uint8_t dpBacklight;

static void SendCommand(uint8_t);
static void SendChar(uint8_t);
static void Send(uint8_t, uint8_t);
static void Write4Bits(uint8_t);
static void ExpanderWrite(uint8_t);
static void PulseEnable(uint8_t);
static void DelayInit(void);
static void DelayUS(uint32_t);

uint8_t special1[8] = {
	0b00000,
	0b11001,
	0b11011,
	0b00110,
	0b01100,
	0b11011,
	0b10011,
	0b00000
};

uint8_t special2[8] = {
	0b11000,
	0b11000,
	0b00110,
	0b01001,
	0b01000,
	0b01001,
	0b00110,
	0b00000
};

void HD44780_Init(uint8_t rows)
```

```
{
 dpRows = rows;

 dpBacklight = LCD_BACKLIGHT;

 dpFunction = LCD_4BITMODE | LCD_1LINE | LCD_5x8DOTS;

 if (dpRows > 1)
 {
  dpFunction |= LCD_2LINE;
 }
 else
 {
  dpFunction |= LCD_5x10DOTS;
 }

 /* Wait for initialization */
 DelayInit();
 HAL_Delay(50);

 ExpanderWrite(dpBacklight);
 HAL_Delay(1000);

 /* 4bit Mode */
 Write4Bits(0x03 << 4);
 DelayUS(4500);

 Write4Bits(0x03 << 4);
 DelayUS(4500);

 Write4Bits(0x03 << 4);
 DelayUS(4500);

 Write4Bits(0x02 << 4);
 DelayUS(100);

 /* Display Control */
 SendCommand(LCD_FUNCTIONSET | dpFunction);

 dpControl = LCD_DISPLAYON | LCD_CURSOROFF | LCD_BLINKOFF;
 HD44780_Display();
 HD44780_Clear();

 /* Display Mode */
 dpMode = LCD_ENTRYLEFT | LCD_ENTRYSHIFTDECREMENT;
```

```
 SendCommand(LCD_ENTRYMODESET | dpMode);
 DelayUS(4500);

 HD44780_CreateSpecialChar(0, special1);
 HD44780_CreateSpecialChar(1, special2);

 HD44780_Home();
}

void HD44780_Clear()
{
 SendCommand(LCD_CLEARDISPLAY);
 DelayUS(2000);
}

void HD44780_Home()
{
 SendCommand(LCD_RETURNHOME);
 DelayUS(2000);
}

void HD44780_SetCursor(uint8_t col, uint8_t row)
{
 int row_offsets[] = { 0x00, 0x40, 0x14, 0x54 };
 if (row >= dpRows)
 {
  row = dpRows-1;
 }
 SendCommand(LCD_SETDDRAMADDR | (col + row_offsets[row]));
}

void HD44780_NoDisplay()
{
 dpControl &= ~LCD_DISPLAYON;
 SendCommand(LCD_DISPLAYCONTROL | dpControl);
}

void HD44780_Display()
{
 dpControl |= LCD_DISPLAYON;
 SendCommand(LCD_DISPLAYCONTROL | dpControl);
}

void HD44780_NoCursor()
{
```

```
 dpControl &= ~LCD_CURSORON;
 SendCommand(LCD_DISPLAYCONTROL | dpControl);
}

void HD44780_Cursor()
{
 dpControl |= LCD_CURSORON;
 SendCommand(LCD_DISPLAYCONTROL | dpControl);
}

void HD44780_NoBlink()
{
 dpControl &= ~LCD_BLINKON;
 SendCommand(LCD_DISPLAYCONTROL | dpControl);
}

void HD44780_Blink()
{
 dpControl |= LCD_BLINKON;
 SendCommand(LCD_DISPLAYCONTROL | dpControl);
}

void HD44780_ScrollDisplayLeft(void)
{
 SendCommand(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVELEFT);
}

void HD44780_ScrollDisplayRight(void)
{
 SendCommand(LCD_CURSORSHIFT | LCD_DISPLAYMOVE | LCD_MOVERIGHT);
}

void HD44780_LeftToRight(void)
{
 dpMode |= LCD_ENTRYLEFT;
 SendCommand(LCD_ENTRYMODESET | dpMode);
}

void HD44780_RightToLeft(void)
{
 dpMode &= ~LCD_ENTRYLEFT;
 SendCommand(LCD_ENTRYMODESET | dpMode);
}

void HD44780_AutoScroll(void)
```

```
{
 dpMode |= LCD_ENTRYSHIFTINCREMENT;
 SendCommand(LCD_ENTRYMODESET | dpMode);
}

void HD44780_NoAutoScroll(void)
{
 dpMode &= ~LCD_ENTRYSHIFTINCREMENT;
 SendCommand(LCD_ENTRYMODESET | dpMode);
}

void HD44780_CreateSpecialChar(uint8_t location, uint8_t charmap[])
{
 location &= 0x7;
 SendCommand(LCD_SETCGRAMADDR | (location << 3));
 for (int i=0; i<8; i++)
 {
  SendChar(charmap[i]);
 }
}

void HD44780_PrintSpecialChar(uint8_t index)
{
 SendChar(index);
}

void HD44780_LoadCustomCharacter(uint8_t char_num, uint8_t *rows)
{
 HD44780_CreateSpecialChar(char_num, rows);
}

void HD44780_PrintStr(const char c[])
{
 while(*c) SendChar(*c++);
}

void HD44780_SetBacklight(uint8_t new_val)
{
 if(new_val) HD44780_Backlight();
 else HD44780_NoBacklight();
}

void HD44780_NoBacklight(void)
{
 dpBacklight=LCD_NOBACKLIGHT;
```

```
 ExpanderWrite(0);
}

void HD44780_Backlight(void)
{
 dpBacklight=LCD_BACKLIGHT;
 ExpanderWrite(0);
}

static void SendCommand(uint8_t cmd)
{
 Send(cmd, 0);
}

static void SendChar(uint8_t ch)
{
 Send(ch, RS);
}

static void Send(uint8_t value, uint8_t mode)
{
 uint8_t highnib = value & 0xF0;
 uint8_t lownib = (value<<4) & 0xF0;
 Write4Bits((highnib)|mode);
 Write4Bits((lownib)|mode);
}

static void Write4Bits(uint8_t value)
{
 ExpanderWrite(value);
 PulseEnable(value);
}

static void ExpanderWrite(uint8_t _data)
{
 uint8_t data = _data | dpBacklight;
 HAL_I2C_Master_Transmit(&hi2c1, DEVICE_ADDR, (uint8_t*)&data, 1, 10);
}

static void PulseEnable(uint8_t _data)
{
 ExpanderWrite(_data | ENABLE);
 DelayUS(20);

 ExpanderWrite(_data & ~ENABLE);
```

```
  DelayUS(20);
}

static void DelayInit(void)
{
  CoreDebug->DEMCR &= ~CoreDebug_DEMCR_TRCENA_Msk;
  CoreDebug->DEMCR |=  CoreDebug_DEMCR_TRCENA_Msk;

  DWT->CTRL &= ~DWT_CTRL_CYCCNTENA_Msk; //~0x00000001;
  DWT->CTRL |=  DWT_CTRL_CYCCNTENA_Msk; //0x00000001;

  DWT->CYCCNT = 0;

  /* 3 NO OPERATION instructions */
  __ASM volatile ("NOP");
  __ASM volatile ("NOP");
  __ASM volatile ("NOP");
}

static void DelayUS(uint32_t us) {
  uint32_t cycles = (SystemCoreClock/1000000L)*us;
  uint32_t start = DWT->CYCCNT;
  volatile uint32_t cnt;

  do
  {
    cnt = DWT->CYCCNT - start;
  } while(cnt < cycles);
}
```

step_motor.h

```
#ifndef INC_STEP_MOTOR_H_
#define INC_STEP_MOTOR_H_

#include "main.h"

#define IN1_PIN GPIO_PIN_15
#define IN1_PORT GPIOB
#define IN2_PIN GPIO_PIN_14
#define IN2_PORT GPIOB
#define IN3_PIN GPIO_PIN_13
#define IN3_PORT GPIOB
#define IN4_PIN GPIO_PIN_12
#define IN4_PORT GPIOB


void microDelay (uint16_t delay);
void stepCCW (int steps, uint16_t delay);
void stepCW (int steps, uint16_t delay);

void open_gate();
void close_gate();
#endif /* INC_STEP_MOTOR_H_ */
```

step_motor.c

```
#include "step_motor.h"
extern TIM_HandleTypeDef htim2;
void microDelay (uint16_t delay)
{
  __HAL_TIM_SET_COUNTER(&htim2, 0);
  while (__HAL_TIM_GET_COUNTER(&htim2) < delay);
}
```

```
void stepCCW (int steps, uint16_t delay) // CCW - Counter Clockwise
{
 for(int x=0; x<steps; x=x+1)
 {
  HAL_GPIO_WritePin(IN1_PORT, IN1_PIN, GPIO_PIN_SET);   // IN1
  HAL_GPIO_WritePin(IN2_PORT, IN2_PIN, GPIO_PIN_RESET); // IN2
  HAL_GPIO_WritePin(IN3_PORT, IN3_PIN, GPIO_PIN_RESET); // IN3
  HAL_GPIO_WritePin(IN4_PORT, IN4_PIN, GPIO_PIN_RESET); // IN4
  microDelay(delay);
  HAL_GPIO_WritePin(IN1_PORT, IN1_PIN, GPIO_PIN_SET);   // IN1
  HAL_GPIO_WritePin(IN2_PORT, IN2_PIN, GPIO_PIN_SET);   // IN2
  HAL_GPIO_WritePin(IN3_PORT, IN3_PIN, GPIO_PIN_RESET); // IN3
  HAL_GPIO_WritePin(IN4_PORT, IN4_PIN, GPIO_PIN_RESET); // IN4
  microDelay(delay);
  HAL_GPIO_WritePin(IN1_PORT, IN1_PIN, GPIO_PIN_RESET); // IN1
  HAL_GPIO_WritePin(IN2_PORT, IN2_PIN, GPIO_PIN_SET);   // IN2
  HAL_GPIO_WritePin(IN3_PORT, IN3_PIN, GPIO_PIN_RESET); // IN3
  HAL_GPIO_WritePin(IN4_PORT, IN4_PIN, GPIO_PIN_RESET); // IN4
  microDelay(delay);
  HAL_GPIO_WritePin(IN1_PORT, IN1_PIN, GPIO_PIN_RESET); // IN1
  HAL_GPIO_WritePin(IN2_PORT, IN2_PIN, GPIO_PIN_SET);   // IN2
  HAL_GPIO_WritePin(IN3_PORT, IN3_PIN, GPIO_PIN_SET);   // IN3
  HAL_GPIO_WritePin(IN4_PORT, IN4_PIN, GPIO_PIN_RESET); // IN4
  microDelay(delay);
  HAL_GPIO_WritePin(IN1_PORT, IN1_PIN, GPIO_PIN_RESET); // IN1
  HAL_GPIO_WritePin(IN2_PORT, IN2_PIN, GPIO_PIN_RESET); // IN2
  HAL_GPIO_WritePin(IN3_PORT, IN3_PIN, GPIO_PIN_SET);   // IN3
  HAL_GPIO_WritePin(IN4_PORT, IN4_PIN, GPIO_PIN_RESET); // IN4
  microDelay(delay);
  HAL_GPIO_WritePin(IN1_PORT, IN1_PIN, GPIO_PIN_RESET); // IN1
  HAL_GPIO_WritePin(IN2_PORT, IN2_PIN, GPIO_PIN_RESET); // IN2
  HAL_GPIO_WritePin(IN3_PORT, IN3_PIN, GPIO_PIN_SET);   // IN3
  HAL_GPIO_WritePin(IN4_PORT, IN4_PIN, GPIO_PIN_SET);   // IN4
  microDelay(delay);
  HAL_GPIO_WritePin(IN1_PORT, IN1_PIN, GPIO_PIN_RESET); // IN1
  HAL_GPIO_WritePin(IN2_PORT, IN2_PIN, GPIO_PIN_RESET); // IN2
  HAL_GPIO_WritePin(IN3_PORT, IN3_PIN, GPIO_PIN_RESET); // IN3
  HAL_GPIO_WritePin(IN4_PORT, IN4_PIN, GPIO_PIN_SET);   // IN4
  microDelay(delay);
  HAL_GPIO_WritePin(IN1_PORT, IN1_PIN, GPIO_PIN_SET);   // IN1
  HAL_GPIO_WritePin(IN2_PORT, IN2_PIN, GPIO_PIN_RESET); // IN2
  HAL_GPIO_WritePin(IN3_PORT, IN3_PIN, GPIO_PIN_RESET); // IN3
  HAL_GPIO_WritePin(IN4_PORT, IN4_PIN, GPIO_PIN_SET);   // IN4
  microDelay(delay);
 }}
```

```
void stepCW (int steps, uint16_t delay) // CW - Clockwise
{
 for(int x=0; x<steps; x=x+1)
 {
  HAL_GPIO_WritePin(IN1_PORT, IN1_PIN, GPIO_PIN_SET);   // IN1
  HAL_GPIO_WritePin(IN2_PORT, IN2_PIN, GPIO_PIN_RESET); // IN2
  HAL_GPIO_WritePin(IN3_PORT, IN3_PIN, GPIO_PIN_RESET); // IN3
  HAL_GPIO_WritePin(IN4_PORT, IN4_PIN, GPIO_PIN_SET);   // IN4
  microDelay(delay);
  HAL_GPIO_WritePin(IN1_PORT, IN1_PIN, GPIO_PIN_RESET); // IN1
  HAL_GPIO_WritePin(IN2_PORT, IN2_PIN, GPIO_PIN_RESET); // IN2
  HAL_GPIO_WritePin(IN3_PORT, IN3_PIN, GPIO_PIN_RESET); // IN3
  HAL_GPIO_WritePin(IN4_PORT, IN4_PIN, GPIO_PIN_SET);   // IN4
  microDelay(delay);
  HAL_GPIO_WritePin(IN1_PORT, IN1_PIN, GPIO_PIN_RESET); // IN1
  HAL_GPIO_WritePin(IN2_PORT, IN2_PIN, GPIO_PIN_RESET); // IN2
  HAL_GPIO_WritePin(IN3_PORT, IN3_PIN, GPIO_PIN_SET);   // IN3
  HAL_GPIO_WritePin(IN4_PORT, IN4_PIN, GPIO_PIN_SET);   // IN4
  microDelay(delay);
  HAL_GPIO_WritePin(IN1_PORT, IN1_PIN, GPIO_PIN_RESET); // IN1
  HAL_GPIO_WritePin(IN2_PORT, IN2_PIN, GPIO_PIN_RESET); // IN2
  HAL_GPIO_WritePin(IN3_PORT, IN3_PIN, GPIO_PIN_SET);   // IN3
  HAL_GPIO_WritePin(IN4_PORT, IN4_PIN, GPIO_PIN_RESET); // IN4
  microDelay(delay);
  HAL_GPIO_WritePin(IN1_PORT, IN1_PIN, GPIO_PIN_RESET); // IN1
  HAL_GPIO_WritePin(IN2_PORT, IN2_PIN, GPIO_PIN_SET);   // IN2
  HAL_GPIO_WritePin(IN3_PORT, IN3_PIN, GPIO_PIN_SET);   // IN3
  HAL_GPIO_WritePin(IN4_PORT, IN4_PIN, GPIO_PIN_RESET); // IN4
  microDelay(delay);
  HAL_GPIO_WritePin(IN1_PORT, IN1_PIN, GPIO_PIN_RESET); // IN1
  HAL_GPIO_WritePin(IN2_PORT, IN2_PIN, GPIO_PIN_SET);   // IN2
  HAL_GPIO_WritePin(IN3_PORT, IN3_PIN, GPIO_PIN_RESET); // IN3
  HAL_GPIO_WritePin(IN4_PORT, IN4_PIN, GPIO_PIN_RESET); // IN4
  microDelay(delay);
  HAL_GPIO_WritePin(IN1_PORT, IN1_PIN, GPIO_PIN_SET);   // IN1
  HAL_GPIO_WritePin(IN2_PORT, IN2_PIN, GPIO_PIN_SET);   // IN2
  HAL_GPIO_WritePin(IN3_PORT, IN3_PIN, GPIO_PIN_RESET); // IN3
  HAL_GPIO_WritePin(IN4_PORT, IN4_PIN, GPIO_PIN_RESET); // IN4
  microDelay(delay);
  HAL_GPIO_WritePin(IN1_PORT, IN1_PIN, GPIO_PIN_SET);   // IN1
  HAL_GPIO_WritePin(IN2_PORT, IN2_PIN, GPIO_PIN_RESET); // IN2
  HAL_GPIO_WritePin(IN3_PORT, IN3_PIN, GPIO_PIN_RESET); // IN3
  HAL_GPIO_WritePin(IN4_PORT, IN4_PIN, GPIO_PIN_RESET); // IN4
  microDelay(delay);
 }}
```

```
void open_gate(void)
{
	stepCCW(128, 1000);
	HAL_Delay(500);
}


void close_gate(void)
{
	stepCW(128, 1000);
	HAL_Delay(500);

}
```

ultrasonic_sensor_1.h

```
#ifndef INC_ULTRASONIC_SENSOR_1_H_
#define INC_ULTRASONIC_SENSOR_1_H_

#include "main.h"

#define TRIG_PIN_1 GPIO_PIN_7
#define TRIG_PORT_1 GPIOE
#define ECHO_PIN_1 GPIO_PIN_8
#define ECHO_PORT_1 GPIOE

extern uint16_t Distance_1;

void uss_init1();
void uss_dist1();


#endif /* INC_ULTRASONIC_SENSOR_1_H_ */
```

ultrasonic_sensor_1.c

```
#include "ultrasonic_sensor_1.h"

extern TIM_HandleTypeDef htim1;


uint16_t Distance_1  = 0;  // cm

void uss_init1()
{
	HAL_TIM_Base_Start(&htim1);
	HAL_GPIO_WritePin(TRIG_PORT_1, TRIG_PIN_1, GPIO_PIN_RESET);
}
void uss_dist1()
{
	uint32_t pMillis;
	uint32_t Value1 = 0;
	uint32_t Value2 = 0;

  HAL_GPIO_WritePin(TRIG_PORT_1, TRIG_PIN_1, GPIO_PIN_SET);  // pull the TRIG pin HIGH
   __HAL_TIM_SET_COUNTER(&htim1, 0);
   while (__HAL_TIM_GET_COUNTER (&htim1) < 10);  // wait for 10 us
   HAL_GPIO_WritePin(TRIG_PORT_1, TRIG_PIN_1, GPIO_PIN_RESET);  // pull the TRIG pin low

   pMillis = HAL_GetTick(); // used this to avoid infinite while loop  (for timeout)
   // wait for the echo pin to go high
   while (!(HAL_GPIO_ReadPin (ECHO_PORT_1, ECHO_PIN_1)) && pMillis + 10 >  HAL_GetTick());
   Value1 = __HAL_TIM_GET_COUNTER (&htim1);

   pMillis = HAL_GetTick(); // used this to avoid infinite while loop (for timeout)
   // wait for the echo pin to go low
   while ((HAL_GPIO_ReadPin (ECHO_PORT_1, ECHO_PIN_1)) && pMillis + 50 > HAL_GetTick());
   Value2 = __HAL_TIM_GET_COUNTER (&htim1);

   Distance_1 = (Value2-Value1)* 0.034/2;
   HAL_Delay(50);



}
```

ultrasonic_sensor.h scalable modification

```
struct USS{


        uint16_t Distance;  // cm

        uint16_t TRIG_PIN;
        uint16_t ECHO_PIN;
        GPIO_TypeDef *TRIG_PORT;
        GPIO_TypeDef *ECHO_PORT;
};


void USS_init(struct USS* s,GPIO_TypeDef * trig_port, uint16_t trig_pin, GPIO_TypeDef *
echo_port, uint16_t echo_pin);
void USS_dist_calc(struct USS* s);
```

ultrasonic_sensor.c scalable modification

```
void USS_init(struct USS* s,GPIO_TypeDef * trig_port, uint16_t trig_pin, GPIO_TypeDef *
echo_port, uint16_t echo_pin)
{
        s->TRIG_PORT=trig_port;
        s->TRIG_PIN=trig_pin;
        s->ECHO_PORT=echo_port;
        s->ECHO_PIN=echo_pin;
         HAL_TIM_Base_Start(&htim1);
         HAL_GPIO_WritePin(s->TRIG_PORT, s->TRIG_PIN, GPIO_PIN_RESET);

}

void USS_dist_calc(struct USS* s)
{
        uint32_t pMillis;
        uint32_t Value1;
        uint32_t Value2;
HAL_GPIO_WritePin(s->TRIG_PORT, s->TRIG_PIN, GPIO_PIN_SET);  // pull the TRIG pin HIGH
__HAL_TIM_SET_COUNTER(&htim1, 0);
while (__HAL_TIM_GET_COUNTER (&htim1) < 10);  // wait for 10 us
 HAL_GPIO_WritePin(s->TRIG_PORT, s->TRIG_PIN, GPIO_PIN_RESET);  // pull the TRIG pin low

  pMillis = HAL_GetTick(); // used this to avoid infinite while loop  (for timeout)
 // wait for the echo pin to go high
while (!(HAL_GPIO_ReadPin (s->ECHO_PORT, s->ECHO_PIN)) && pMillis + 10 >  HAL_GetTick());
 Value1 = __HAL_TIM_GET_COUNTER (&htim1);

 pMillis = HAL_GetTick(); // used this to avoid infinite while loop (for timeout)
  // wait for the echo pin to go low
while ((HAL_GPIO_ReadPin (s->ECHO_PORT, s->ECHO_PIN)) && pMillis + 50 > HAL_GetTick());
Value2 = __HAL_TIM_GET_COUNTER (&htim1);

            s->Distance = (Value2-Value1)* 0.034/2;
            HAL_Delay(50);
}
```

main.c

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "peripheral_init.h"
#include "ultrasonic_sensor_1.h"
#include "ultrasonic_sensor_2.h"
#include "ultrasonic_sensor_3.h"
#include "step_motor.h"
#include "lcd_i2c.h"


#define DISTANCE_FROM_FLOOR 9
#define DISTANCE_FROM_GATE 5
#define MAX_NUMBER_OF_CARS 2


int main(void)
{


  /* Initialize all configured peripherals */

  peripheral_init();



  uss_init1();
  uss_init2();
  uss_init3();

  HD44780_Init(2);
  HD44780_Clear();
  HD44780_SetCursor(0,0);

  HAL_TIM_Base_Start(&htim2);

char num[5];            /* Helper variable for turning numbers into strings using the itoa function */
uint16_t number_of_cars=0; /* Variable to keep track of number of cars in parking lot */
uint16_t state_changed=1;  /* Variable to keep track of changes so that the LCD only clears when
something new happens */
```

```
// If you want to start it with cars already inside:
/*

            uss_dist1();
            uss_dist2();
            uss_dist3();

               if(Distance_2 < DISTANCE_FROM_FLOOR-2)
number_of_cars=number_of_cars+1;
                        if(Distance_3 < DISTANCE_FROM_FLOOR-2)
number_of_cars=number_of_cars+1;

*/


/* Infinite loop */
 while (1)
 {


   /* CALCULATE ALL DISTANCES */
            uss_dist1();
            uss_dist2();
            uss_dist3();


/*  FOR VISUAL DEBUGGING
 *
            HD44780_SetCursor(0,0);
            HAL_Delay(500);
            HD44780_PrintStr("G:");
            itoa(Distance_1,num,10);
            HD44780_PrintStr(num);

            itoa(Distance_2,num,10);
            HD44780_PrintStr("s1:");
            HD44780_PrintStr(num);

            itoa(Distance_3,num,10);
            HD44780_PrintStr("s2:");
            HD44780_PrintStr(num);

            HD44780_SetCursor(0,1);
            itoa(number_of_cars,snum,10);
```

```
            HD44780_PrintStr("Cars:");
            HD44780_PrintStr(snum);
*/

              /* If car is there turn off red light and turn on green light
                if car is not there turn off green light and turn on red light */

              /*CHECK SPOT 1*/

              if(Distance_2 < DISTANCE_FROM_FLOOR-1)
                    {
                          HAL_GPIO_WritePin(GPIOC,GPIO_PIN_7 , 0);
                          HAL_GPIO_WritePin(GPIOC,GPIO_PIN_6 , 1);
                    }
                    else
                    {
                          HAL_GPIO_WritePin(GPIOC,GPIO_PIN_6 , 0);
                          HAL_GPIO_WritePin(GPIOC,GPIO_PIN_7 , 1);
                    }



                    /*CHECK SPOT 2*/

                    if(Distance_3 < DISTANCE_FROM_FLOOR-1)
                    {
                          HAL_GPIO_WritePin(GPIOC,GPIO_PIN_9 , 0);
                          HAL_GPIO_WritePin(GPIOC,GPIO_PIN_8 , 1);
                    }
                    else
                    {
                          HAL_GPIO_WritePin(GPIOC,GPIO_PIN_8 , 0);
                          HAL_GPIO_WritePin(GPIOC,GPIO_PIN_9 , 1);
                    }

/* Update LCD display with message about how many available spots there are */
 if(state_changed)
 {
                  HD44780_Clear();
                  HD44780_SetCursor(0,0);

                   if(number_of_cars == MAX_NUMBER_OF_CARS)
                   {
                          HD44780_PrintStr("Nema slobodnih");
                          HD44780_SetCursor(0,1);
```

```
                        HD44780_PrintStr("mjesta!");
        }
                else
                {

                        itoa((MAX_NUMBER_OF_CARS-number_of_cars),num,10);
                        HD44780_PrintStr(num);
                        HD44780_PrintStr(" mjesta ");
                        HD44780_SetCursor(0,1);
                        HD44780_PrintStr("su slobodna");
                }

state_changed=0;
}

/* Check if car is outside or inside gate. If inside it is leaving , if outside it is coming in.
  Tell LCD display to update. Open gate, wait 5 seconds, close gate wait 3 seconds.
  If parking is full don't open gate to car coming in. */

        /*CHECK GATE*/
                if(Distance_1 < DISTANCE_FROM_GATE + 10)
                {


                        if(Distance_1 <= DISTANCE_FROM_GATE)
                        {
                                if(number_of_cars != 0)
                           number_of_cars=number_of_cars-1;

                        }
                        else
                        {
                                if(number_of_cars == MAX_NUMBER_OF_CARS)
                                        continue;

                                number_of_cars=number_of_cars+1;
                        }
                          state_changed=1;
                                open_gate();
                                HAL_Delay(5000);
                                close_gate();
                                HAL_Delay(3000);

                }
```

```
                        HAL_Delay(500);

  }
}


/* 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 */
```