1.11 LCD-Displays

There are two main types of LCDs: parallel and serial. Parallel LCDs (like the HD44780) connect to a microcontroller via multiple data lines – typically 4 or 8. Four lines save pins but slow down data transfer. Serial LCDs use just one data line (RS232 protocol), making them simpler to use but are generally more expensive. Programming parallel LCDs is complex and requires understanding the controller's timing. The HD44780 is a popular monochrome module, available in various sizes (8, 16, 20, 24, 32 or 40) characters and with 14- or 16-pin connectors.

Table 5: Connection b

LCD Pin	Function	Microcontroller Pin/Connection
VSS (1)	Ground	GND
VDD (2)	Power	+ 5V
VEE (3)	Contrast	10 kΩ potentiometer
R/S (4)	Control	PA6
R/W (5)	Control	PA5
E (6)	Control	PA7
D0 (7)	Data	NC
D1 (8)	Data	NC
D2 (9)	Data	NC
D3 (10)	Data	NC
D4 (11)	Data	PA8
D5 (12)	Data	PA9
D6 (13)	Data	PA10
D7 (14)	Data	PA11
A (15)	Backlight (+)	+5V (via 330 Ω resistor)
K (16)	Backlight (-)	GND

- VSS is the 0 V supply or ground.
- VDD pin should be connected to the positive supply. Although the data sheet specifies 5 VDC supply, the modules will usually work with as low as 3 V or as high as 6 V.
- Pin 3 is named VEE and is the contrast control pin. This is used to control the contrast of the display and should be connected to a variable voltage.
- Pin 4 is the Register Select (RS) pin. When this pin is low, data transferred to the display is interpreted as a command (such as clearing the display or setting the cursor position). When RS is High, the data is treated as character data to be displayed.
- Pin 5 is the Read/Write (R/W) line. This pin is pulled low in order to write commands or character data to the LCD module. When this pin is pulled high, the LCD is in read mode: status information (such as the busy flag) or data can be read from the LCD.
- Pin 6 is the Enable (E) pin which is used to initiate the transfer of commands or data between the module and the microcontroller. When writing to the display, data is transferred only on the HIGH to LOW transition of the line. When reading from the display, data becomes available after the LOW to HIGH transition of the enable pin and this data remains valid as long as the enable pin is at logic HIGH.
- Pins 7 to 14 are the eight data bus lines (D0 to D7). Data can be transferred between the microcontroller and the LCD module using either a single 8-bit byte, or as two 4-bit nibbles. In the latter case only the upper four data lines (D4 to D7) are used. 4-bit

mode has the advantage that four less I/O lines are required to communicate with the LCD.

- Pin 15 is the positive voltage supply for the LCDs backlight LCDs. Connect it with 330 Ω resistor to + 5 V supply voltage.
- Pin16 is the ground for LCDs backlight LCDs.

The circuit diagram is shown Figure 43. The LCD is operated in 4-bit mode; the connections between the LCD and the GPIO pins are listed in Table 5. The LCD is powered by the +5V supply, available on the morpho connectors. A 10 kΩ potentiometer is connected to the VEE pin to adjust the LCD contrast.

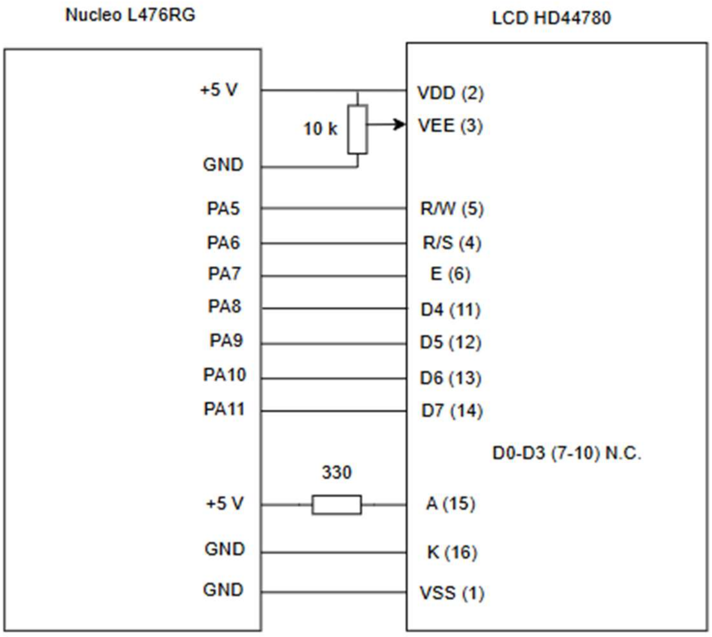
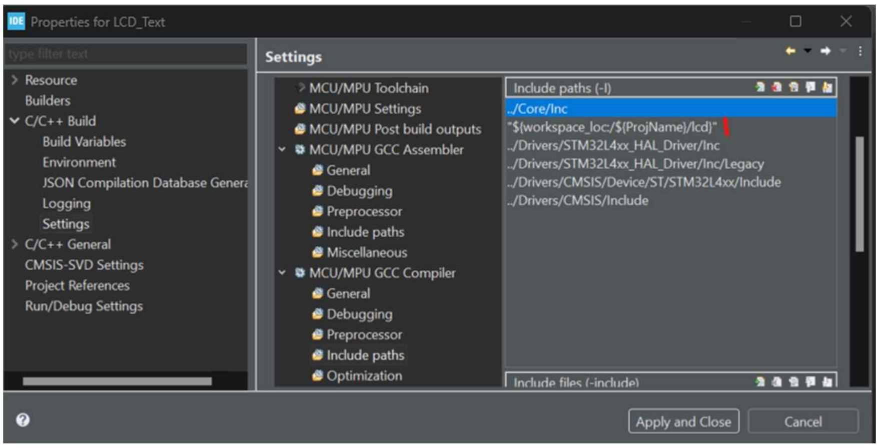


Figure 43: Circuit diagram

The program is configured as follows:

- Create a new workspace in STM32 Cube.
- Click to start a new STM32 project.
- Search for STM32L476RG
- The name of the project is LCD_Text
- The library is set up in a separate folder of the project and named 'lcd'. This folder contains the actual files 'lcd.c' and 'lcd.h'. Set the include path to the files and properties of the project. Thes can be entered in the 'Settings' tab under the menu item 'C/C++ Build'. This is shown in Figure 44 for the file 'lcd.c'.



Properties for LCD_Text

type filter text

- Resource
- Builders
- C/C++ Build
  - Build Variables
  - Environment
  - JSON Compilation Database Genera
  - Logging
  - Settings
- C/C++ General
- CMSIS-SVD Settings
- Project References
- Run/Debug Settings

Settings

- MCU/MPU Toolchain
- MCU/MPU Settings
- MCU/MPU Post build outputs
- MCU/MPU GCC Assembler
  - General
  - Debugging
  - Preprocessor
  - Include paths
  - Miscellaneous
- MCU/MPU GCC Compiler
  - General
  - Debugging
  - Preprocessor
  - Include paths
  - Optimization

Include paths (-I)

```
../Core/Inc
"${workspace_loc:/${ProjName}/lcd}"
../Drivers/STM32L4xx_HAL_Driver/Inc
../Drivers/STM32L4xx_HAL_Driver/Inc/Legacy
../Drivers/CMSIS/Device/ST/STM32L4xx/Include
../Drivers/CMSIS/Include
```

Include files (-include)

Figure 44: Path to the lib file lcd.c

- Configure GPIO pins PA6 to PA11 as digital output (Figure 45).

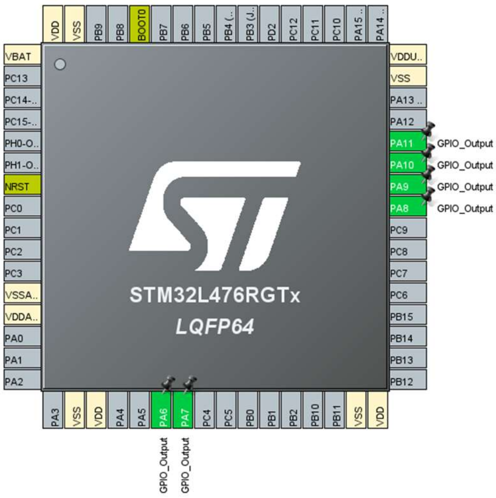


Figure 45: Chip configuration

The following image illustrates the test setup of the project.

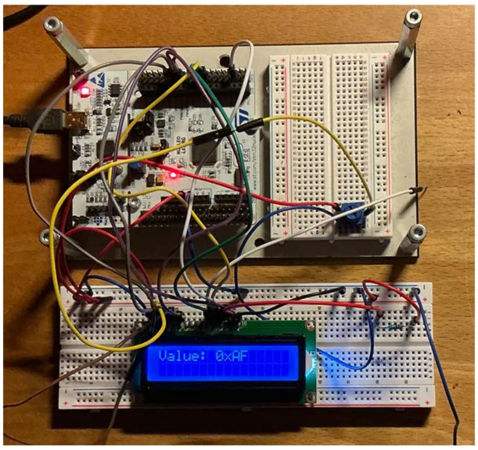
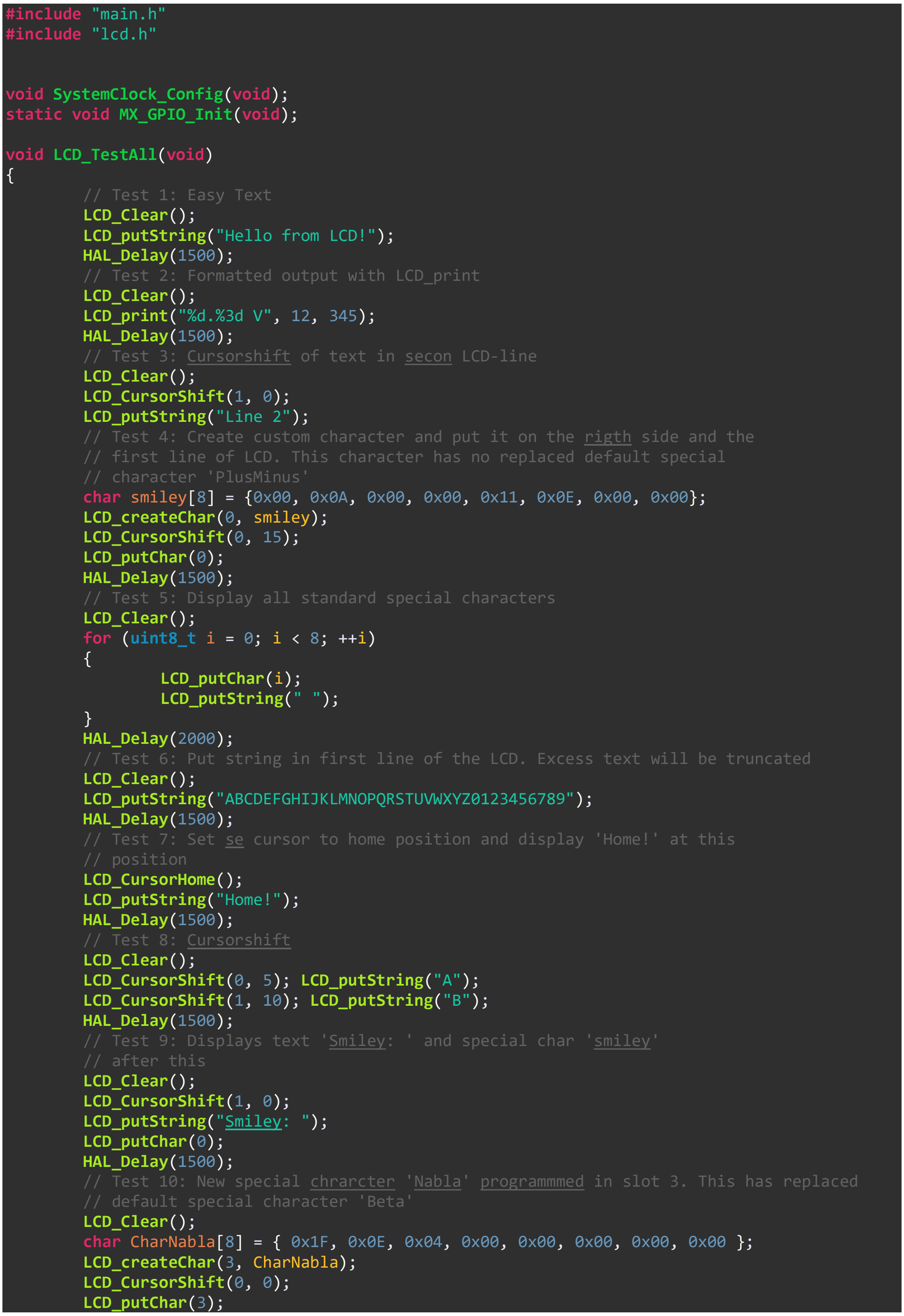


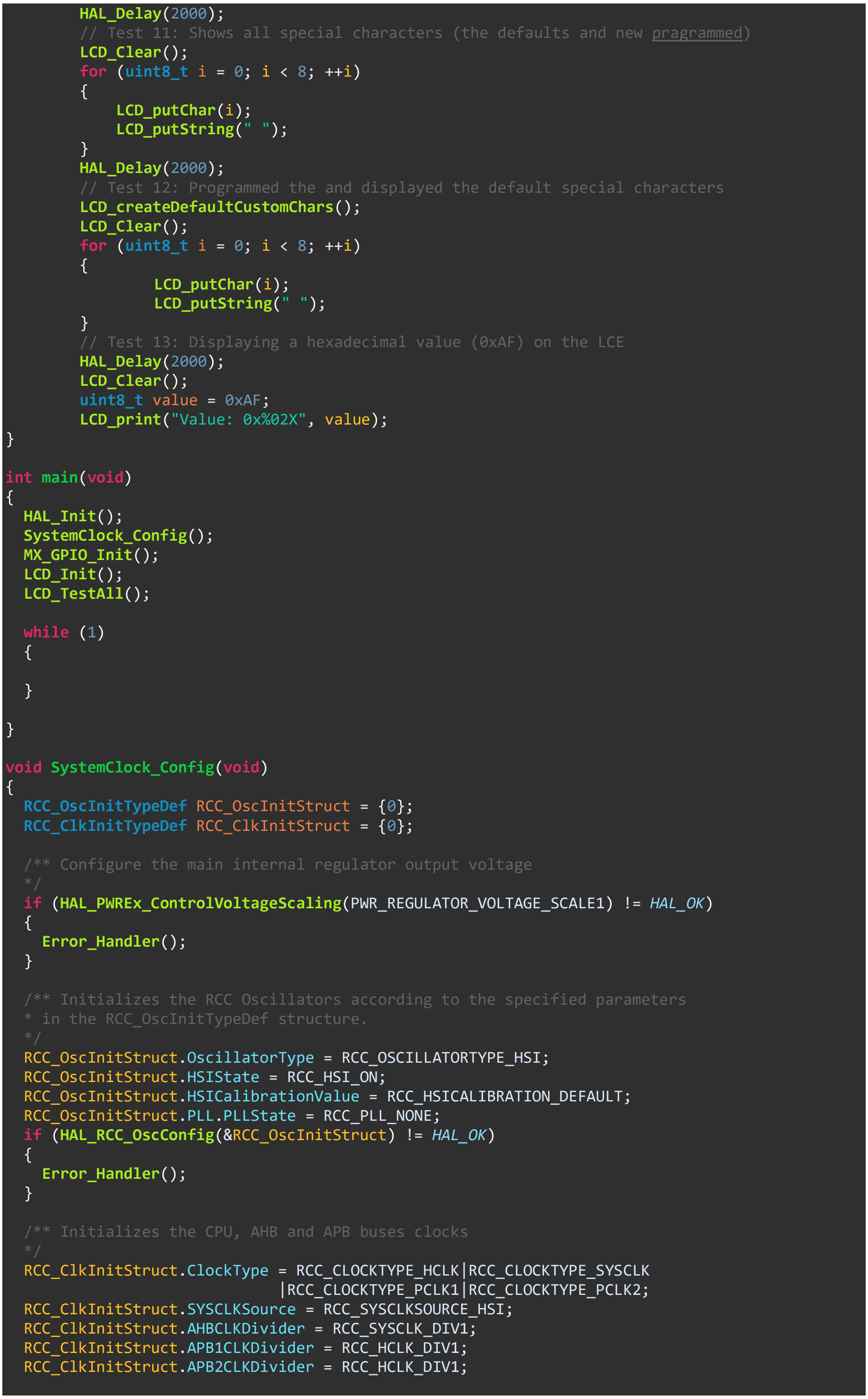
Figure 46: Test setup

The Code is as follows:

The function ‘LCD_TestAll’ (in main.c) is primarily used to test the LCD.



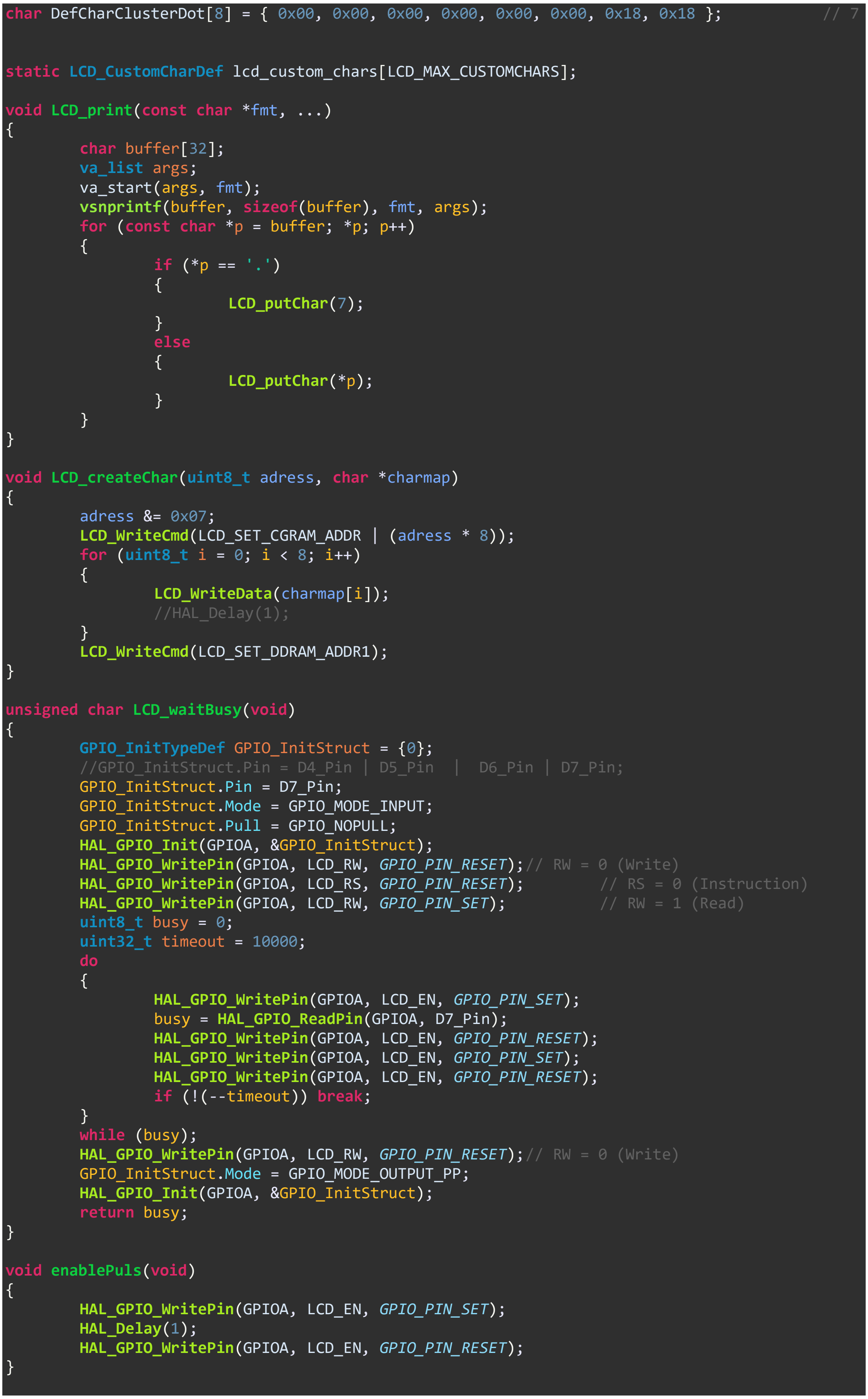
```c
#include "main.h"
#include "lcd.h"


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

void LCD_TestAll(void)
{
        // Test 1: Easy Text
        LCD_Clear();
        LCD_putString("Hello from LCD!");
        HAL_Delay(1500);
        // Test 2: Formatted output with LCD_print
        LCD_Clear();
        LCD_print("%d.%3d V", 12, 345);
        HAL_Delay(1500);
        // Test 3: Cursorshift of text in secon LCD-line
        LCD_Clear();
        LCD_CursorShift(1, 0);
        LCD_putString("Line 2");
        // Test 4: Create custom character and put it on the rigth side and the
        // first line of LCD. This character has no replaced default special
        // character 'PlusMinus'
        char smiley[8] = {0x00, 0x0A, 0x00, 0x00, 0x11, 0x0E, 0x00, 0x00};
        LCD_createChar(0, smiley);
        LCD_CursorShift(0, 15);
        LCD_putChar(0);
        HAL_Delay(1500);
        // Test 5: Display all standard special characters
        LCD_Clear();
        for (uint8_t i = 0; i < 8; ++i)
        {
                LCD_putChar(i);
                LCD_putString(" ");
        }
        HAL_Delay(2000);
        // Test 6: Put string in first line of the LCD. Excess text will be truncated
        LCD_Clear();
        LCD_putString("ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789");
        HAL_Delay(1500);
        // Test 7: Set se cursor to home position and display 'Home!' at this
        // position
        LCD_CursorHome();
        LCD_putString("Home!");
        HAL_Delay(1500);
        // Test 8: Cursorshift
        LCD_Clear();
        LCD_CursorShift(0, 5); LCD_putString("A");
        LCD_CursorShift(1, 10); LCD_putString("B");
        HAL_Delay(1500);
        // Test 9: Displays text 'Smiley: ' and special char 'smiley'
        // after this
        LCD_Clear();
        LCD_CursorShift(1, 0);
        LCD_putString("Smiley: ");
        LCD_putChar(0);
        HAL_Delay(1500);
        // Test 10: New special chrarcter 'Nabla' programmmed in slot 3. This has replaced
        // default special character 'Beta'
        LCD_Clear();
        char CharNabla[8] = { 0x1F, 0x0E, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00 };
        LCD_createChar(3, CharNabla);
        LCD_CursorShift(0, 0);
        LCD_putChar(3);
```



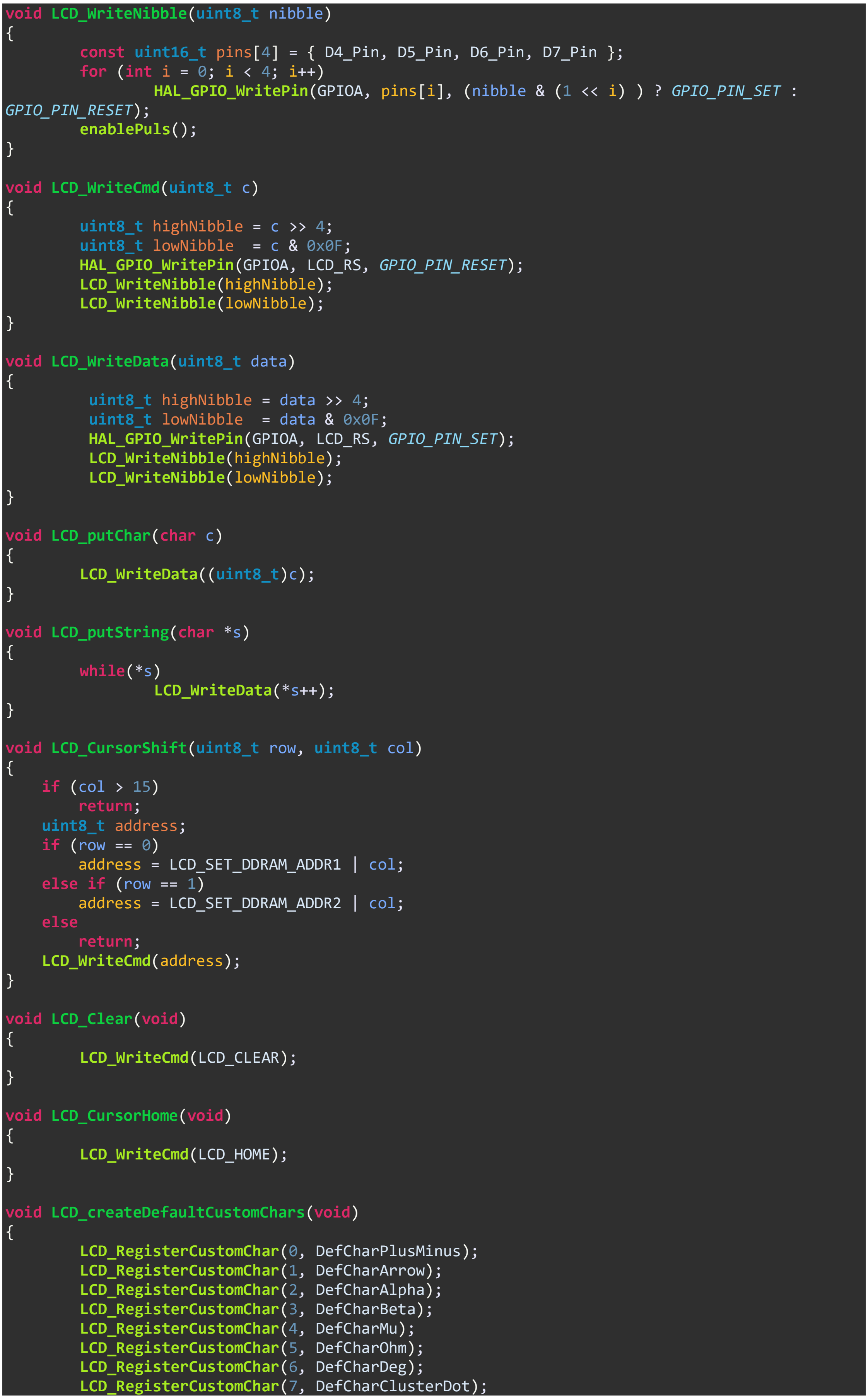
```
        HAL_Delay(2000);
        // Test 11: Shows all special characters (the defaults and new pragrammed)
        LCD_Clear();
        for (uint8_t i = 0; i < 8; ++i)
        {
            LCD_putChar(i);
            LCD_putString(" ");
        }
        HAL_Delay(2000);
        // Test 12: Programmed the and displayed the default special characters
        LCD_createDefaultCustomChars();
        LCD_Clear();
        for (uint8_t i = 0; i < 8; ++i)
        {
                LCD_putChar(i);
                LCD_putString(" ");
        }
        // Test 13: Displaying a hexadecimal value (0xAF) on the LCE
        HAL_Delay(2000);
        LCD_Clear();
        uint8_t value = 0xAF;
        LCD_print("Value: 0x%02X", value);
}

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

  while (1)
  {

  }

}

void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
  {
    Error_Handler();
  }

  /** 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_NONE;
  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_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}


static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8
                          |GPIO_PIN_9|GPIO_PIN_10|GPIO_PIN_11, GPIO_PIN_RESET);

  GPIO_InitStruct.Pin = GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8
                          |GPIO_PIN_9|GPIO_PIN_10|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(GPIOA, &GPIO_InitStruct);
}

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

void assert_failed(uint8_t *file, uint32_t line)
{

}
#endif /* USE_FULL_ASSERT */
```

The files ‘lcd.c’ and ‘lcd.h’ form the actual library.

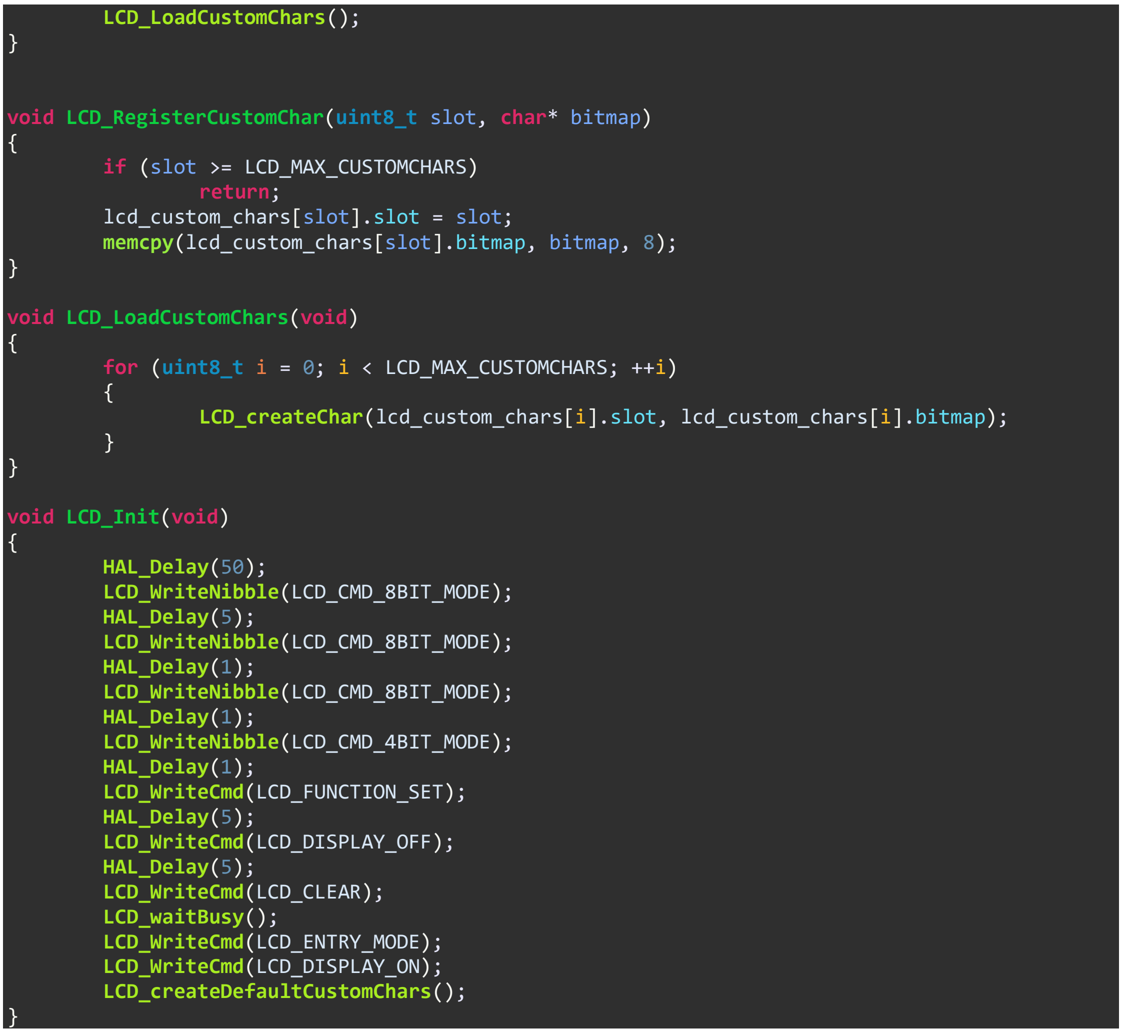
```
/*
 * lcd.c
 *
 *  Created on: Jun 23, 2025
 *      Author: Bernd Seggewiß
 */

#include "main.h"
#include <string.h>
#include "lcd.h"
#include <stdarg.h>
#include <stdio.h>


char DefCharPlusMinus[8] = {0x04, 0x04, 0x1F, 0x04, 0x04, 0x00, 0x1F, 0x00};            // 0
char DefCharArrow[8] = {0x08, 0x0C, 0x0E, 0x0F, 0x0E, 0x0C, 0x08, 0x00};                // 1
char DefCharAlpha[8] = { 0x00, 0x0E, 0x01, 0x0F, 0x11, 0x11, 0x0F, 0x00 };
        // 2
char DefCharBeta[8] = { 0x0E, 0x11, 0x11, 0x0E, 0x11, 0x11, 0x16, 0x10 };               // 3
char DefCharMu[8]    = { 0x00, 0x11, 0x11, 0x11, 0x13, 0x0D, 0x01, 0x00 };
        // 4
char DefCharOhm[8]   = { 0x00, 0x0E, 0x11, 0x11, 0x11, 0x0E, 0x0A, 0x11 };
        // 5
char DefCharDeg[8]   = { 0x06, 0x09, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00 };
        // 6
```



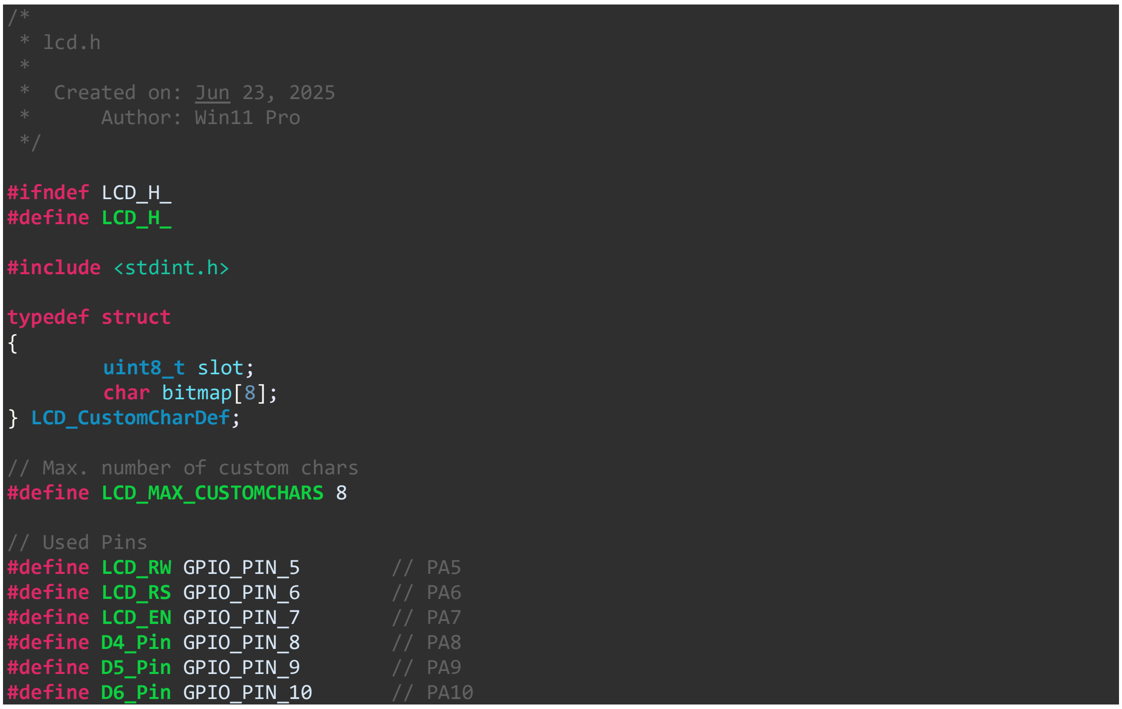
```c
char DefCharClusterDot[8] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18 };          // 7


static LCD_CustomCharDef lcd_custom_chars[LCD_MAX_CUSTOMCHARS];

void LCD_print(const char *fmt, ...)
{
        char buffer[32];
        va_list args;
        va_start(args, fmt);
        vsnprintf(buffer, sizeof(buffer), fmt, args);
        for (const char *p = buffer; *p; p++)
        {
                if (*p == '.')
                {
                        LCD_putChar(7);
                }
                else
                {
                        LCD_putChar(*p);
                }
        }
}

void LCD_createChar(uint8_t adress, char *charmap)
{
        adress &= 0x07;
        LCD_WriteCmd(LCD_SET_CGRAM_ADDR | (adress * 8));
        for (uint8_t i = 0; i < 8; i++)
        {
                LCD_WriteData(charmap[i]);
                //HAL_Delay(1);
        }
        LCD_WriteCmd(LCD_SET_DDRAM_ADDR1);
}

unsigned char LCD_waitBusy(void)
{
        GPIO_InitTypeDef GPIO_InitStruct = {0};
        //GPIO_InitStruct.Pin = D4_Pin | D5_Pin  |  D6_Pin | D7_Pin;
        GPIO_InitStruct.Pin = D7_Pin;
        GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
        HAL_GPIO_WritePin(GPIOA, LCD_RW, GPIO_PIN_RESET);// RW = 0 (Write)
        HAL_GPIO_WritePin(GPIOA, LCD_RS, GPIO_PIN_RESET);        // RS = 0 (Instruction)
        HAL_GPIO_WritePin(GPIOA, LCD_RW, GPIO_PIN_SET);          // RW = 1 (Read)
        uint8_t busy = 0;
        uint32_t timeout = 10000;
        do
        {
                HAL_GPIO_WritePin(GPIOA, LCD_EN, GPIO_PIN_SET);
                busy = HAL_GPIO_ReadPin(GPIOA, D7_Pin);
                HAL_GPIO_WritePin(GPIOA, LCD_EN, GPIO_PIN_RESET);
                HAL_GPIO_WritePin(GPIOA, LCD_EN, GPIO_PIN_SET);
                HAL_GPIO_WritePin(GPIOA, LCD_EN, GPIO_PIN_RESET);
                if (!(--timeout)) break;
        }
        while (busy);
        HAL_GPIO_WritePin(GPIOA, LCD_RW, GPIO_PIN_RESET);// RW = 0 (Write)
        GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
        HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
        return busy;
}

void enablePuls(void)
{
        HAL_GPIO_WritePin(GPIOA, LCD_EN, GPIO_PIN_SET);
        HAL_Delay(1);
        HAL_GPIO_WritePin(GPIOA, LCD_EN, GPIO_PIN_RESET);
}
```



```
void LCD_WriteNibble(uint8_t nibble)
{
        const uint16_t pins[4] = { D4_Pin, D5_Pin, D6_Pin, D7_Pin };
        for (int i = 0; i < 4; i++)
                HAL_GPIO_WritePin(GPIOA, pins[i], (nibble & (1 << i) ) ? GPIO_PIN_SET :
GPIO_PIN_RESET);
        enablePuls();
}

void LCD_WriteCmd(uint8_t c)
{
        uint8_t highNibble = c >> 4;
        uint8_t lowNibble  = c & 0x0F;
        HAL_GPIO_WritePin(GPIOA, LCD_RS, GPIO_PIN_RESET);
        LCD_WriteNibble(highNibble);
        LCD_WriteNibble(lowNibble);
}

void LCD_WriteData(uint8_t data)
{
         uint8_t highNibble = data >> 4;
         uint8_t lowNibble  = data & 0x0F;
         HAL_GPIO_WritePin(GPIOA, LCD_RS, GPIO_PIN_SET);
         LCD_WriteNibble(highNibble);
         LCD_WriteNibble(lowNibble);
}

void LCD_putChar(char c)
{
        LCD_WriteData((uint8_t)c);
}

void LCD_putString(char *s)
{
        while(*s)
                LCD_WriteData(*s++);
}

void LCD_CursorShift(uint8_t row, uint8_t col)
{
    if (col > 15)
        return;
    uint8_t address;
    if (row == 0)
        address = LCD_SET_DDRAM_ADDR1 | col;
    else if (row == 1)
        address = LCD_SET_DDRAM_ADDR2 | col;
    else
        return;
    LCD_WriteCmd(address);
}

void LCD_Clear(void)
{
        LCD_WriteCmd(LCD_CLEAR);
}

void LCD_CursorHome(void)
{
        LCD_WriteCmd(LCD_HOME);
}

void LCD_createDefaultCustomChars(void)
{
        LCD_RegisterCustomChar(0, DefCharPlusMinus);
        LCD_RegisterCustomChar(1, DefCharArrow);
        LCD_RegisterCustomChar(2, DefCharAlpha);
        LCD_RegisterCustomChar(3, DefCharBeta);
        LCD_RegisterCustomChar(4, DefCharMu);
        LCD_RegisterCustomChar(5, DefCharOhm);
        LCD_RegisterCustomChar(6, DefCharDeg);
        LCD_RegisterCustomChar(7, DefCharClusterDot);
```



```
        LCD_LoadCustomChars();
}


void LCD_RegisterCustomChar(uint8_t slot, char* bitmap)
{
        if (slot >= LCD_MAX_CUSTOMCHARS)
                return;
        lcd_custom_chars[slot].slot = slot;
        memcpy(lcd_custom_chars[slot].bitmap, bitmap, 8);
}

void LCD_LoadCustomChars(void)
{
        for (uint8_t i = 0; i < LCD_MAX_CUSTOMCHARS; ++i)
        {
                LCD_createChar(lcd_custom_chars[i].slot, lcd_custom_chars[i].bitmap);
        }
}

void LCD_Init(void)
{
        HAL_Delay(50);
        LCD_WriteNibble(LCD_CMD_8BIT_MODE);
        HAL_Delay(5);
        LCD_WriteNibble(LCD_CMD_8BIT_MODE);
        HAL_Delay(1);
        LCD_WriteNibble(LCD_CMD_8BIT_MODE);
        HAL_Delay(1);
        LCD_WriteNibble(LCD_CMD_4BIT_MODE);
        HAL_Delay(1);
        LCD_WriteCmd(LCD_FUNCTION_SET);
        HAL_Delay(5);
        LCD_WriteCmd(LCD_DISPLAY_OFF);
        HAL_Delay(5);
        LCD_WriteCmd(LCD_CLEAR);
        LCD_waitBusy();
        LCD_WriteCmd(LCD_ENTRY_MODE);
        LCD_WriteCmd(LCD_DISPLAY_ON);
        LCD_createDefaultCustomChars();
}
```



```
/*
 * lcd.h
 *
 *  Created on: Jun 23, 2025
 *      Author: Win11 Pro
 */

#ifndef LCD_H_
#define LCD_H_

#include <stdint.h>

typedef struct
{
        uint8_t slot;
        char bitmap[8];
} LCD_CustomCharDef;

// Max. number of custom chars
#define LCD_MAX_CUSTOMCHARS 8

// Used Pins
#define LCD_RW GPIO_PIN_5       // PA5
#define LCD_RS GPIO_PIN_6       // PA6
#define LCD_EN GPIO_PIN_7       // PA7
#define D4_Pin GPIO_PIN_8       // PA8
#define D5_Pin GPIO_PIN_9       // PA9
#define D6_Pin GPIO_PIN_10      // PA10
```

```
#define D7_Pin GPIO_PIN_11       // PA11

// Custom-Char slot definitions
#define LCD_CGRAM_SLOT_PLUSMINUS 0
#define LCD_CGRAM_SLOT_ARROW            1
#define LCD_CGRAM_SLOT_CLUSTERDOT       2
#define LCD_CGRAM_SLOT_ALPHA            3
#define LCD_CGRAM_SLOT_BETA                     4
#define LCD_CGRAM_SLOT_MU                       5
#define LCD_CGRAM_SLOT_OHM                      6
#define LCD_CGRAM_SLOT_DEG                      7

extern char CharPlusMinus[8];
extern char CharArrow[8];
extern char CharClusterDot[8];
extern char CharAlpha[8];
extern char CharBeta[8];
extern char CharMu[8];
extern char CharOhm[8];
extern char CharDeg[8];

#define LCD_CMD_8BIT_MODE      0x3   // Initialisierungsschritt: 8-Bit-Modus (nur Nibble!)
#define LCD_CMD_4BIT_MODE      0x2   // Initialisierungsschritt: Umschalten auf 4-Bit-Modus (nur Nibble!)
#define LCD_FUNCTION_SET       0x28  // 4-Bit, 2 Zeilen, 5x8 Dots
#define LCD_DISPLAY_OFF        0x08  // Display aus
#define LCD_DISPLAY_ON         0x0C  // Display an, Cursor aus, Blinken aus
#define LCD_CLEAR              0x01  // Display löschen
#define LCD_HOME               0x02  // Cursor Home
#define LCD_ENTRY_MODE         0x06  // Cursor nach rechts, kein Display-Shift
#define LCD_SET_CGRAM_ADDR     0x40  // Setze CGRAM-Adresse (für Custom Characters)
#define LCD_SET_DDRAM_ADDR1    0x80  // Setze Cursor an Anfang erste Zeile
#define LCD_SET_DDRAM_ADDR2    0xC0  // Setze Cursor an Anfang zweite Zeile

unsigned char LCD_waitBusy(void);
void enablePuls(void);
void LCD_WriteNibble(uint8_t);
void LCD_WriteCmd(uint8_t);
void LCD_WriteData(uint8_t);
void LCD_putChar(char c);
void LCD_putString(char *s);
void LCD_CursorShift(uint8_t, uint8_t);
void LCD_Clear(void);
void LCD_CursorHome(void);
void LCD_createChar(uint8_t, char*);
void LCD_print(const char *fmt, ...);
void LCD_createDefaultCustomChars(void);

void LCD_LoadCustomChars(void);

void LCD_RegisterCustomChar(uint8_t slot, char* bitmap);

void LCD_Init(void);

#endif /* LCD_H_ */
```