MNGLUT008

EEE3096S

CBXLIS001

GitHub Link

<https://github.com/LuthoYRN/MNGLUT008_CBXLIS001_EEE3096S/blob/main/Prac3/main.c>

Description of Implementation

In this practical, we further enhanced the initial STM32 firmware by implementing a range of functionalities: SPI-based EEPROM communication, generation of PWM signals, ADC polling, and LCD output.

1. SPI EEPROM Communication: We implemented the functionality to read and write to an external EEPROM using the SPI interface. A predefined array of binary data was written to specific memory addresses in EEPROM using the **write\_to\_address** function. Additionally, the **read\_from\_address** function was used to verify the correctness of the data stored in EEPROM. We compared the read values against the original data, given a mismatch, an error message “SPI ERROR” is displayed on the LCD.

2. PWM Signal Generation and Control: We added PWM control by using the ADC input from a potentiometer. The ADC values were polled periodically, and the result was converted into a PWM duty cycle through the **ADCtoCCR** function. The PWM signal controlled an LED’s brightness by adjusting the duty cycle dynamically based on the potentiometer’s position. The PWM implementation uses TIM3, Channel 3 and the CCR value is updated in each loop iteration to reflect real time changes in the potentiometer’s input.

3. LCD Output: The **writeLCD** function was added to display the relevant information on a connected LCD module. This included the value read from EEPROM memory or an error message if the SPI read was incorrect. The LCD updated every time the EEPROM value was checked, with the display showing both the correct value and the error message.

4. Button Control and Frequency Adjustment: A button interrupt was implemented to toggle the LED blinking frequency between 2 Hz and 1 Hz. We dynamically adjusted this frequency using an EXTI interrupt and debouncing logic to avoid accidental multiple presses. The button-controlled frequency update altered the LED toggling, providing visual feedback based on the selected interval.

5. Timers for periodic tasks: We used TIM6 to handle the periodic toggling of LED7 based on the button-controlled frequency. TIM16 was also used to periodically check the EEPROM values, update the LCD with the current memory content, and ensure data integrity. These timers were configured to run in interrupt mode, ensuring that tasks such as EEPROM checking and LED blinking occurred without blocking other tasks.

Appendix

/\* USER CODE BEGIN Header \*/

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @file : main.c

\* @brief : Main program body

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @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 \*/

/\* Includes ------------------------------------------------------------------\*/

**#include** "main.h"

/\* Private includes ----------------------------------------------------------\*/

/\* USER CODE BEGIN Includes \*/

**#include** <stdio.h>

**#include** "stm32f0xx.h"

**#include** <lcd\_stm32f0.c>

/\* USER CODE END Includes \*/

/\* Private typedef -----------------------------------------------------------\*/

/\* USER CODE BEGIN PTD \*/

/\* USER CODE END PTD \*/

/\* Private define ------------------------------------------------------------\*/

/\* USER CODE BEGIN PD \*/

// Definitions for SPI usage

**#define** **MEM\_SIZE** 8192 // bytes

**#define** **WREN** 0b00000110 // enable writing

**#define** **WRDI** 0b00000100 // disable writing

**#define** **RDSR** 0b00000101 // read status register

**#define** **WRSR** 0b00000001 // write status register

**#define** **READ** 0b00000011

**#define** **WRITE** 0b00000010

/\* USER CODE END PD \*/

/\*our declared variables\*/

**uint32\_t** period = 500; // Initial frequency period (500 ms, 2 Hz)

**uint32\_t** previoustime = 0;

**uint32\_t** adc\_value=0;

**static** **uint8\_t** binaryArray[6] = {

0b10101010, // 170 in decimal

0b01010101, // 85 in decimal

0b11001100, // 204 in decimal

0b00110011, // 51 in decimal

0b11110000, // 240 in decimal

0b00001111 // 15 in decimal

};

**uint16\_t** currentAddress = 0;

/\* Private macro -------------------------------------------------------------\*/

/\* USER CODE BEGIN PM \*/

/\* USER CODE END PM \*/

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

**ADC\_HandleTypeDef** hadc;

**TIM\_HandleTypeDef** htim3;

**TIM\_HandleTypeDef** htim6;

**TIM\_HandleTypeDef** htim16;

/\* USER CODE BEGIN PV \*/

// **TODO**: Define input variables

/\* USER CODE END PV \*/

/\* Private function prototypes -----------------------------------------------\*/

**void** **SystemClock\_Config**(**void**);

**static** **void** **MX\_GPIO\_Init**(**void**);

**static** **void** **MX\_ADC\_Init**(**void**);

**static** **void** **MX\_TIM3\_Init**(**void**);

**static** **void** **MX\_TIM16\_Init**(**void**);

**static** **void** **MX\_TIM6\_Init**(**void**);

/\* USER CODE BEGIN PFP \*/

**void** **EXTI0\_1\_IRQHandler**(**void**);

**void** **TIM16\_IRQHandler**(**void**);

**void** **writeLCD**(**char** \*char\_in);

// ADC functions

**uint32\_t** **pollADC**(**void**);

**uint32\_t** **ADCtoCCR**(**uint32\_t** adc\_val);

// SPI functions

**static** **void** **init\_spi**(**void**);

**static** **void** **write\_to\_address**(**uint16\_t** address, **uint8\_t** data);

**static** **uint8\_t** **read\_from\_address**(**uint16\_t** address);

**static** **void** **spi\_delay**(**uint32\_t** delay\_in\_us);

/\* USER CODE END PFP \*/

/\* Private user code ---------------------------------------------------------\*/

/\* USER CODE BEGIN 0 \*/

/\* USER CODE END 0 \*/

/\*\*

\* @brief The application entry point.

\* @retval int

\*/

**int** **main**(**void**)

{

/\* USER CODE BEGIN 1 \*/

/\* USER CODE END 1 \*/

/\* MCU Configuration--------------------------------------------------------\*/

/\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/

**HAL\_Init**();

/\* USER CODE BEGIN Init \*/

/\* USER CODE END Init \*/

/\* Configure the system clock \*/

**SystemClock\_Config**();

/\* USER CODE BEGIN SysInit \*/

/\* USER CODE END SysInit \*/

/\* Initialize all configured peripherals \*/

**init\_spi**();

**MX\_GPIO\_Init**();

**MX\_ADC\_Init**();

**MX\_TIM3\_Init**();

**MX\_TIM16\_Init**();

**MX\_TIM6\_Init**();

/\* USER CODE BEGIN 2 \*/

// Initialise LCD

**init\_LCD**();

// Start timers

**HAL\_TIM\_Base\_Start\_IT**(&htim6);

**HAL\_TIM\_Base\_Start\_IT**(&htim16);

// PWM setup

**uint32\_t** CCR = 0;

**HAL\_TIM\_PWM\_Start**(&htim3, TIM\_CHANNEL\_3); // Start PWM on TIM3 Channel 3

// **TODO**: Write all bytes to EEPROM using "write\_to\_address"

**for**(**int** i = 0;i < 6; i++){

**write\_to\_address**(i,binaryArray[i]);

}

/\* USER CODE END 2 \*/

//M16\_IRQHandler();

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

**while** (1)

{

// **TODO**: Poll ADC

adc\_value = **pollADC**(); // Read ADC value from potentiometer

// **TODO**: Get CRR

CCR = **ADCtoCCR**(adc\_value); // Convert ADC value to CCR value

// Update PWM value

\_\_HAL\_TIM\_SetCompare(&htim3, TIM\_CHANNEL\_3, CCR);

// Wait for delay ms

**HAL\_Delay** (period);

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

**void** **SystemClock\_Config**(**void**)

{

**LL\_FLASH\_SetLatency**(LL\_FLASH\_LATENCY\_0);

**while**(**LL\_FLASH\_GetLatency**() != LL\_FLASH\_LATENCY\_0)

{

}

**LL\_RCC\_HSI\_Enable**();

/\* Wait till HSI is ready \*/

**while**(**LL\_RCC\_HSI\_IsReady**() != 1)

{

}

**LL\_RCC\_HSI\_SetCalibTrimming**(16);

**LL\_RCC\_HSI14\_Enable**();

/\* Wait till HSI14 is ready \*/

**while**(**LL\_RCC\_HSI14\_IsReady**() != 1)

{

}

**LL\_RCC\_HSI14\_SetCalibTrimming**(16);

**LL\_RCC\_SetAHBPrescaler**(LL\_RCC\_SYSCLK\_DIV\_1);

**LL\_RCC\_SetAPB1Prescaler**(LL\_RCC\_APB1\_DIV\_1);

**LL\_RCC\_SetSysClkSource**(LL\_RCC\_SYS\_CLKSOURCE\_HSI);

/\* Wait till System clock is ready \*/

**while**(**LL\_RCC\_GetSysClkSource**() != LL\_RCC\_SYS\_CLKSOURCE\_STATUS\_HSI)

{

}

**LL\_SetSystemCoreClock**(8000000);

/\* Update the time base \*/

**if** (**HAL\_InitTick** (TICK\_INT\_PRIORITY) != *HAL\_OK*)

{

**Error\_Handler**();

}

**LL\_RCC\_HSI14\_EnableADCControl**();

}

/\*\*

\* @brief ADC Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** **MX\_ADC\_Init**(**void**)

{

/\* USER CODE BEGIN ADC\_Init 0 \*/

/\* USER CODE END ADC\_Init 0 \*/

**ADC\_ChannelConfTypeDef** sConfig = {0};

/\* USER CODE BEGIN ADC\_Init 1 \*/

/\* USER CODE END ADC\_Init 1 \*/

/\*\* Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)

\*/

hadc.Instance = ADC1;

hadc.Init.ClockPrescaler = ADC\_CLOCK\_ASYNC\_DIV1;

hadc.Init.Resolution = ADC\_RESOLUTION\_12B;

hadc.Init.DataAlign = ADC\_DATAALIGN\_RIGHT;

hadc.Init.ScanConvMode = ADC\_SCAN\_DIRECTION\_FORWARD;

hadc.Init.EOCSelection = ADC\_EOC\_SINGLE\_CONV;

hadc.Init.LowPowerAutoWait = *DISABLE*;

hadc.Init.LowPowerAutoPowerOff = *DISABLE*;

hadc.Init.ContinuousConvMode = *DISABLE*;

hadc.Init.DiscontinuousConvMode = *DISABLE*;

hadc.Init.ExternalTrigConv = ADC\_SOFTWARE\_START;

hadc.Init.ExternalTrigConvEdge = ADC\_EXTERNALTRIGCONVEDGE\_NONE;

hadc.Init.DMAContinuousRequests = *DISABLE*;

hadc.Init.Overrun = ADC\_OVR\_DATA\_PRESERVED;

**if** (**HAL\_ADC\_Init**(&hadc) != *HAL\_OK*)

{

**Error\_Handler**();

}

/\*\* Configure for the selected ADC regular channel to be converted.

\*/

sConfig.Channel = ADC\_CHANNEL\_6;

sConfig.Rank = ADC\_RANK\_CHANNEL\_NUMBER;

sConfig.SamplingTime = ADC\_SAMPLETIME\_1CYCLE\_5;

**if** (**HAL\_ADC\_ConfigChannel**(&hadc, &sConfig) != *HAL\_OK*)

{

**Error\_Handler**();

}

/\* USER CODE BEGIN ADC\_Init 2 \*/

ADC1->CR |= ADC\_CR\_ADCAL;

**while**(ADC1->CR & ADC\_CR\_ADCAL); // Calibrate the ADC

ADC1->CR |= (1 << 0); // Enable ADC

**while**((ADC1->ISR & (1 << 0)) == 0); // Wait for ADC ready

/\* USER CODE END ADC\_Init 2 \*/

}

/\*\*

\* @brief TIM3 Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** **MX\_TIM3\_Init**(**void**)

{

/\* USER CODE BEGIN TIM3\_Init 0 \*/

/\* USER CODE END TIM3\_Init 0 \*/

**TIM\_ClockConfigTypeDef** sClockSourceConfig = {0};

**TIM\_MasterConfigTypeDef** sMasterConfig = {0};

**TIM\_OC\_InitTypeDef** sConfigOC = {0};

/\* USER CODE BEGIN TIM3\_Init 1 \*/

/\* USER CODE END TIM3\_Init 1 \*/

htim3.Instance = TIM3;

htim3.Init.Prescaler = 0;

htim3.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim3.Init.Period = 47999;

htim3.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

htim3.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;

**if** (**HAL\_TIM\_Base\_Init**(&htim3) != *HAL\_OK*)

{

**Error\_Handler**();

}

sClockSourceConfig.ClockSource = TIM\_CLOCKSOURCE\_INTERNAL;

**if** (**HAL\_TIM\_ConfigClockSource**(&htim3, &sClockSourceConfig) != *HAL\_OK*)

{

**Error\_Handler**();

}

**if** (**HAL\_TIM\_PWM\_Init**(&htim3) != *HAL\_OK*)

{

**Error\_Handler**();

}

sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;

sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

**if** (**HAL\_TIMEx\_MasterConfigSynchronization**(&htim3, &sMasterConfig) != *HAL\_OK*)

{

**Error\_Handler**();

}

sConfigOC.OCMode = TIM\_OCMODE\_PWM1;

sConfigOC.Pulse = 0;

sConfigOC.OCPolarity = TIM\_OCPOLARITY\_HIGH;

sConfigOC.OCFastMode = TIM\_OCFAST\_DISABLE;

**if** (**HAL\_TIM\_PWM\_ConfigChannel**(&htim3, &sConfigOC, TIM\_CHANNEL\_3) != *HAL\_OK*)

{

**Error\_Handler**();

}

/\* USER CODE BEGIN TIM3\_Init 2 \*/

/\* USER CODE END TIM3\_Init 2 \*/

**HAL\_TIM\_MspPostInit**(&htim3);

}

/\*\*

\* @brief TIM6 Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** **MX\_TIM6\_Init**(**void**)

{

/\* USER CODE BEGIN TIM6\_Init 0 \*/

/\* USER CODE END TIM6\_Init 0 \*/

**TIM\_MasterConfigTypeDef** sMasterConfig = {0};

/\* USER CODE BEGIN TIM6\_Init 1 \*/

/\* USER CODE END TIM6\_Init 1 \*/

htim6.Instance = TIM6;

htim6.Init.Prescaler = 8000-1;

htim6.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim6.Init.Period = 500-1;

htim6.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_ENABLE;

**if** (**HAL\_TIM\_Base\_Init**(&htim6) != *HAL\_OK*)

{

**Error\_Handler**();

}

sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;

sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

**if** (**HAL\_TIMEx\_MasterConfigSynchronization**(&htim6, &sMasterConfig) != *HAL\_OK*)

{

**Error\_Handler**();

}

/\* USER CODE BEGIN TIM6\_Init 2 \*/

**NVIC\_EnableIRQ**(TIM6\_IRQn);

/\* USER CODE END TIM6\_Init 2 \*/

}

/\*\*

\* @brief TIM16 Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** **MX\_TIM16\_Init**(**void**)

{

/\* USER CODE BEGIN TIM16\_Init 0 \*/

/\* USER CODE END TIM16\_Init 0 \*/

/\* USER CODE BEGIN TIM16\_Init 1 \*/

/\* USER CODE END TIM16\_Init 1 \*/

htim16.Instance = TIM16;

htim16.Init.Prescaler = 8000-1;

htim16.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim16.Init.Period = 1000-1;

htim16.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

htim16.Init.RepetitionCounter = 0;

htim16.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_ENABLE;

**if** (**HAL\_TIM\_Base\_Init**(&htim16) != *HAL\_OK*)

{

**Error\_Handler**();

}

/\* USER CODE BEGIN TIM16\_Init 2 \*/

**NVIC\_EnableIRQ**(TIM16\_IRQn);

/\* USER CODE END TIM16\_Init 2 \*/

}

/\*\*

\* @brief GPIO Initialization Function

\* @param None

\* @retval None

\*/

**static** **void** **MX\_GPIO\_Init**(**void**)

{

LL\_EXTI\_InitTypeDef EXTI\_InitStruct = {0};

LL\_GPIO\_InitTypeDef GPIO\_InitStruct = {0};

/\* USER CODE BEGIN MX\_GPIO\_Init\_1 \*/

/\* USER CODE END MX\_GPIO\_Init\_1 \*/

/\* GPIO Ports Clock Enable \*/

**LL\_AHB1\_GRP1\_EnableClock**(LL\_AHB1\_GRP1\_PERIPH\_GPIOF);

**LL\_AHB1\_GRP1\_EnableClock**(LL\_AHB1\_GRP1\_PERIPH\_GPIOA);

**LL\_AHB1\_GRP1\_EnableClock**(LL\_AHB1\_GRP1\_PERIPH\_GPIOB);

/\*\*/

**LL\_GPIO\_ResetOutputPin**(LED7\_GPIO\_Port, LED7\_Pin);

/\*\*/

**LL\_SYSCFG\_SetEXTISource**(LL\_SYSCFG\_EXTI\_PORTA, LL\_SYSCFG\_EXTI\_LINE0);

/\*\*/

**LL\_GPIO\_SetPinPull**(Button0\_GPIO\_Port, Button0\_Pin, LL\_GPIO\_PULL\_UP);

/\*\*/

**LL\_GPIO\_SetPinMode**(Button0\_GPIO\_Port, Button0\_Pin, LL\_GPIO\_MODE\_INPUT);

/\*\*/

EXTI\_InitStruct.Line\_0\_31 = LL\_EXTI\_LINE\_0;

EXTI\_InitStruct.LineCommand = *ENABLE*;

EXTI\_InitStruct.Mode = LL\_EXTI\_MODE\_IT;

EXTI\_InitStruct.Trigger = LL\_EXTI\_TRIGGER\_RISING;

**LL\_EXTI\_Init**(&EXTI\_InitStruct);

/\*\*/

GPIO\_InitStruct.Pin = LED7\_Pin;

GPIO\_InitStruct.Mode = LL\_GPIO\_MODE\_OUTPUT;

GPIO\_InitStruct.Speed = LL\_GPIO\_SPEED\_FREQ\_LOW;

GPIO\_InitStruct.OutputType = LL\_GPIO\_OUTPUT\_PUSHPULL;

GPIO\_InitStruct.Pull = LL\_GPIO\_PULL\_NO;

**LL\_GPIO\_Init**(LED7\_GPIO\_Port, &GPIO\_InitStruct);

/\* USER CODE BEGIN MX\_GPIO\_Init\_2 \*/

**HAL\_NVIC\_SetPriority**(EXTI0\_1\_IRQn, 0, 0);

**HAL\_NVIC\_EnableIRQ**(EXTI0\_1\_IRQn);

/\* USER CODE END MX\_GPIO\_Init\_2 \*/

}

/\* USER CODE BEGIN 4 \*/

**void** **EXTI0\_1\_IRQHandler**(**void**)

{

// **TODO**: Add code to switch LED7 delay frequency

**uint32\_t** currentTime = **HAL\_GetTick**();

// Check if button is pressed and debounce time(100ms) has passed

**if** (**LL\_GPIO\_IsInputPinSet**(GPIOA, LL\_GPIO\_PIN\_0) && (currentTime -

previoustime > 500))

{

// Toggle between 500 ms (2 Hz) and 1000 ms (1 Hz) period

**if**( period== 1000-1){

period= 500-1;

}

// Update the previous time for debounce

**else**{

period =1000-1;

}

previoustime = currentTime;

\_\_HAL\_TIM\_SET\_AUTORELOAD(&htim6,period);

}

**HAL\_GPIO\_EXTI\_IRQHandler**(Button0\_Pin); // Clear interrupt flags

}

**void** **TIM6\_IRQHandler**(**void**)

{

// Acknowledge interrupt

**HAL\_TIM\_IRQHandler**(&htim6);

// Toggle LED7

**HAL\_GPIO\_TogglePin**(GPIOB, LED7\_Pin);

}

**void** **TIM16\_IRQHandler**(**void**)

{

// Acknowledge interrupt

**HAL\_TIM\_IRQHandler**(&htim16);

// **TODO**: Initialise a string to output second line on LCD

// **TODO**: Change LED pattern; output 0x01 if the read SPI data is incorrect

// Read the value from EEPROM at the current address

**uint8\_t** eepromValue = **read\_from\_address**(currentAddress);

// Check if the read value matches the expected value from binaryArray

**if** (eepromValue == binaryArray[currentAddress])

{

// Format and print the correct value to LCD

**char** lcdLine[16];

//f(lcdLine, sizeof(lcdLine), "EEPROM byte:\n%d", eepromValue);

**sprintf**(lcdLine, "%d", eepromValue);

**writeLCD**(lcdLine);

}

**else**

{

// Print SPI error message to LCD

**writeLCD**("SPI ERROR!");

}

// Update the current address and wrap around if needed

currentAddress = (currentAddress + 1) % 6;

}

// **TODO**: Complete the writeLCD function

**void** **writeLCD**(**char** \*char\_in){

**delay**(3000);

**lcd\_command**(CLEAR);

**lcd\_command**(CURSOR\_HOME);

**lcd\_command**(TWOLINE\_MODE);

**lcd\_putstring**("EEPROM byte");

**lcd\_command**(LINE\_TWO);

**lcd\_putstring**(char\_in);

}

// Get ADC value

**uint32\_t** **pollADC**(**void**){

**HAL\_ADC\_Start**(&hadc); // start the adc

**HAL\_ADC\_PollForConversion**(&hadc, 100); // poll for conversion

**uint32\_t** val = **HAL\_ADC\_GetValue**(&hadc); // get the adc value

**HAL\_ADC\_Stop**(&hadc); // stop adc

**return** val;

}

// Calculate PWM CCR value

**uint32\_t** **ADCtoCCR**(**uint32\_t** adc\_val){

// **TODO**: Calculate CCR valUE using an equation

**uint32\_t** value = (adc\_value \* 47999) / 4095;

**return** value;

}

**void** **ADC1\_COMP\_IRQHandler**(**void**)

{

//adc\_val = HAL\_ADC\_GetValue(&hadc); // read adc value

**HAL\_ADC\_IRQHandler**(&hadc); //Clear flags

}

// Initialise SPI

**static** **void** **init\_spi**(**void**) {

// Clock to PB

RCC->AHBENR |= RCC\_AHBENR\_GPIOBEN; // Enable clock for SPI port

// Set pin modes

GPIOB->MODER |= GPIO\_MODER\_MODER13\_1; // Set pin SCK (PB13) to Alternate Function

GPIOB->MODER |= GPIO\_MODER\_MODER14\_1; // Set pin MISO (PB14) to Alternate Function

GPIOB->MODER |= GPIO\_MODER\_MODER15\_1; // Set pin MOSI (PB15) to Alternate Function

GPIOB->MODER |= GPIO\_MODER\_MODER12\_0; // Set pin CS (PB12) to output push-pull

GPIOB->BSRR |= GPIO\_BSRR\_BS\_12; // Pull CS high

// Clock enable to SPI

RCC->APB1ENR |= RCC\_APB1ENR\_SPI2EN;

SPI2->CR1 |= SPI\_CR1\_BIDIOE; // Enable output

SPI2->CR1 |= (SPI\_CR1\_BR\_0 | SPI\_CR1\_BR\_1); // Set Baud to fpclk / 16

SPI2->CR1 |= SPI\_CR1\_MSTR; // Set to master mode

SPI2->CR2 |= SPI\_CR2\_FRXTH; // Set RX threshold to be 8 bits

SPI2->CR2 |= SPI\_CR2\_SSOE; // Enable slave output to work in master mode

SPI2->CR2 |= (SPI\_CR2\_DS\_0 | SPI\_CR2\_DS\_1 | SPI\_CR2\_DS\_2); // Set to 8-bit mode

SPI2->CR1 |= SPI\_CR1\_SPE; // Enable the SPI peripheral

}

// Implements a delay in microseconds

**static** **void** **spi\_delay**(**uint32\_t** delay\_in\_us) {

**volatile** **uint32\_t** counter = 0;

delay\_in\_us \*= 3;

**for**(; counter < delay\_in\_us; counter++) {

**\_\_asm**("nop");

**\_\_asm**("nop");

}

}

// Write to EEPROM address using SPI

**static** **void** **write\_to\_address**(**uint16\_t** address, **uint8\_t** data) {

**uint8\_t** dummy; // Junk from the DR

// Set the Write Enable latch

GPIOB->BSRR |= GPIO\_BSRR\_BR\_12; // Pull CS low

**spi\_delay**(1);

\*((**uint8\_t**\*)(&SPI2->DR)) = WREN;

**while** ((SPI2->SR & SPI\_SR\_RXNE) == 0); // Hang while RX is empty

dummy = SPI2->DR;

GPIOB->BSRR |= GPIO\_BSRR\_BS\_12; // Pull CS high

**spi\_delay**(5000);

// Send write instruction

GPIOB->BSRR |= GPIO\_BSRR\_BR\_12; // Pull CS low

**spi\_delay**(1);

\*((**uint8\_t**\*)(&SPI2->DR)) = WRITE;

**while** ((SPI2->SR & SPI\_SR\_RXNE) == 0); // Hang while RX is empty

dummy = SPI2->DR;

// Send 16-bit address

\*((**uint8\_t**\*)(&SPI2->DR)) = (address >> 8); // Address MSB

**while** ((SPI2->SR & SPI\_SR\_RXNE) == 0); // Hang while RX is empty

dummy = SPI2->DR;

\*((**uint8\_t**\*)(&SPI2->DR)) = (address); // Address LSB

**while** ((SPI2->SR & SPI\_SR\_RXNE) == 0); // Hang while RX is empty

dummy = SPI2->DR;

// Send the data

\*((**uint8\_t**\*)(&SPI2->DR)) = data;

**while** ((SPI2->SR & SPI\_SR\_RXNE) == 0); // Hang while RX is empty

dummy = SPI2->DR;

GPIOB->BSRR |= GPIO\_BSRR\_BS\_12; // Pull CS high

**spi\_delay**(5000);

}

// Read from EEPROM address using SPI

**static** **uint8\_t** **read\_from\_address**(**uint16\_t** address) {

**uint8\_t** dummy; // Junk from the DR

// Send the read instruction

GPIOB->BSRR |= GPIO\_BSRR\_BR\_12; // Pull CS low

**spi\_delay**(1);

\*((**uint8\_t**\*)(&SPI2->DR)) = READ;

**while** ((SPI2->SR & SPI\_SR\_RXNE) == 0); // Hang while RX is empty

dummy = SPI2->DR;

// Send 16-bit address

\*((**uint8\_t**\*)(&SPI2->DR)) = (address >> 8); // Address MSB

**while** ((SPI2->SR & SPI\_SR\_RXNE) == 0); // Hang while RX is empty

dummy = SPI2->DR;

\*((**uint8\_t**\*)(&SPI2->DR)) = (address); // Address LSB

**while** ((SPI2->SR & SPI\_SR\_RXNE) == 0); // Hang while RX is empty

dummy = SPI2->DR;

// Clock in the data

\*((**uint8\_t**\*)(&SPI2->DR)) = 0x42; // Clock out some junk data

**while** ((SPI2->SR & SPI\_SR\_RXNE) == 0); // Hang while RX is empty

dummy = SPI2->DR;

GPIOB->BSRR |= GPIO\_BSRR\_BS\_12; // Pull CS high

**spi\_delay**(5000);

**return** dummy; // Return read data

}

/\* 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 \*/