EEE3096S PRAC 3 REPORT

Robert Dugmore – *DGMROB001*
Thomas Koen – *KNXTHO003*

19 September 2023

GITHUB LINK

Entire repo: https://github.com/rothdu/EEE3096S
Prac 3 folder: https://github.com/rothdu/EEE3096S/tree/main/Prac3
main.c: https://github.com/rothdu/EEE3096S/blob/main/Prac3/Core/Src/main.c

CODE

```c
/* 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 */

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

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc;
TIM_HandleTypeDef htim3;

/* USER CODE BEGIN PV */
uint32_t prev_millis = 0;
uint32_t curr_millis = 0;
uint32_t delay_t = 500; // Initialise delay to 500ms
uint32_t adc_val;

// TODO: Debounce stuff (added to make this section easier to find)
uint32_t button0_prev_tick = 0; // initiliase "previous button0 tick" to 0
uint8_t debounce_delay = 50;
/* 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);

/* USER CODE BEGIN PFP */
void EXTI0_1_IRQHandler(void);
void writeLCD(char *char_in);
uint32_t pollADC(void);
uint32_t ADCtoCCR(uint32_t adc_val);
/* 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. */
```

```c
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 */
MX_GPIO_Init();
MX_ADC_Init();
MX_TIM3_Init();

/* USER CODE BEGIN 2 */
init_LCD();

// PWM setup
uint32_t CCR = 0;
HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3); // Start PWM on TIM3 Channel 3
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
// Toggle LED0
HAL_GPIO_TogglePin(GPIOB, LED7_Pin);

// ADC to LCD; TODO: Read POT1 value and write to LCD
uint32_t adc_val = pollADC();

char str[5]; // create string to store value
sprintf(str, "%d", adc_val); // convert int to string

writeLCD(str); // write string to LCD


// Update PWM value; TODO: Get CRR
CCR = ADCtoCCR(adc_val);

__HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_3, CCR);

// Wait for delay ms
HAL_Delay (delay_t);
/* 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 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 current_tick = HAL_GetTick();

// check if button press is reasonably long after previous one
if (current_tick - button0_prev_tick > debounce_delay)
{
if (delay_t == 500) // change from 500 to 250
{
delay_t = 250;
}
else if (delay_t == 250) // change from 250 to 500
{
delay_t = 500;
}

button0_prev_tick = current_tick; // set current tick to previous
}
HAL_GPIO_EXTI_IRQHandler(Button0_Pin); // Clear interrupt flags
}

// TODO: Complete the writeLCD function
void writeLCD(char *char_in){
delay(3000);
lcd_command(CLEAR);

lcd_putstring(char_in); // write string to ADC

}

// Get ADC value
uint32_t pollADC(void){
// TODO: Complete function body to get ADC val
HAL_ADC_Start(&hadc); // start ADC conversion
HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY); // wait for ADC to finish converting
uint32_t val = HAL_ADC_GetValue(&hadc); // get ADC value

return val; // return
}
```

```
// Calculate PWM CCR value
uint32_t ADCtoCCR(uint32_t adc_val){
// TODO: Calculate CCR val using an appropriate equation

uint32_t val = adc_val * 47999 / 4095;

return val;
}

void ADC1_COMP_IRQHandler(void)
{
adc_val = HAL_ADC_GetValue(&hadc); // read adc value
HAL_ADC_IRQHandler(&hadc); //Clear flags
}
/* 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 */
```