```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "usb_device.h"

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

/* 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 ---------------------------------------------------------*/

I2C_HandleTypeDef hi2c1;
UART_HandleTypeDef huart2;
SMBUS_HandleTypeDef hsmbus1;

SPI_HandleTypeDef hspi1;

TIM_HandleTypeDef htim2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
```

```c
static void MX_GPIO_Init(void);
static void MX_I2C1_SMBUS_Init(void);
static void MX_SPI1_Init(void);
static void MX_TIM2_Init(void);
/* USER CODE BEGIN PFP */

/* 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 */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_I2C1_Init();
  /* Initialize all configured peripherals */
  MX_SPI1_Init();
  MX_TIM2_Init();
  MX_USB_DEVICE_Init();
  /* USER CODE BEGIN 2 */

  // CS pin should default high
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);

    // Say something
    uart_buf_len = sprintf(uart_buf, "SPI Test\r\n");
    HAL_UART_Transmit(&huart2, (uint8_t *)uart_buf, uart_buf_len, 100);

    // Enable write enable latch (allow write operations)
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
    HAL_SPI_Transmit(&hspi1, (uint8_t *)&EEPROM_WREN, 1, 100);
```

```
115     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);
116
117     // Read status register
118     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
119     HAL_SPI_Transmit(&hspi1, (uint8_t *)&EEPROM_RDSR, 1, 100);
120     HAL_SPI_Receive(&hspi1, (uint8_t *)spi_buf, 1, 100);
121     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);
122
123     // Print out status register
124     uart_buf_len = sprintf(uart_buf,
125                            "Status: 0x x\r\n",
126                            (unsigned int)spi_buf[0]);
127     HAL_UART_Transmit(&huart2, (uint8_t *)uart_buf, uart_buf_len, 100);
128
129     // Test bytes to write to EEPROM
130     spi_buf[0] = 0xAB;
131     spi_buf[1] = 0xCD;
132     spi_buf[2] = 0xEF;
133
134     // Set starting address
135     addr = 0x05;
136
137     // Write 3 bytes starting at given address
138     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
139     HAL_SPI_Transmit(&hspi1, (uint8_t *)&EEPROM_WRITE, 1, 100);
140     HAL_SPI_Transmit(&hspi1, (uint8_t *)&addr, 1, 100);
141     HAL_SPI_Transmit(&hspi1, (uint8_t *)spi_buf, 3, 100);
142     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);
143
144     // Clear buffer
145     spi_buf[0] = 0;
146     spi_buf[1] = 0;
147     spi_buf[2] = 0;
148
149     // Wait until WIP bit is cleared
150     wip = 1;
151     while (wip)
152     {
153       // Read status register
154       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
155       HAL_SPI_Transmit(&hspi1, (uint8_t *)&EEPROM_RDSR, 1, 100);
156       HAL_SPI_Receive(&hspi1, (uint8_t *)spi_buf, 1, 100);
157       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);
158
159       // Mask out WIP bit
160       wip = spi_buf[0] & 0b00000001;
161     }
162
163     // Read the 3 bytes back
164     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
165     HAL_SPI_Transmit(&hspi1, (uint8_t *)&EEPROM_READ, 1, 100);
166     HAL_SPI_Transmit(&hspi1, (uint8_t *)&addr, 1, 100);
167     HAL_SPI_Receive(&hspi1, (uint8_t *)spi_buf, 3, 100);
168     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);
169
170     // Print out bytes read
171     uart_buf_len = sprintf(uart_buf,
```

```
172                             "0x x 0x x 0x x\r\n",
173                             (unsigned int)spi_buf[0],
174                             (unsigned int)spi_buf[1],
175                             (unsigned int)spi_buf[2]);
176     HAL_UART_Transmit(&huart2, (uint8_t *)uart_buf, uart_buf_len, 100);
177
178     // Read status register
179     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
180     HAL_SPI_Transmit(&hspi1, (uint8_t *)&EEPROM_RDSR, 1, 100);
181     HAL_SPI_Receive(&hspi1, (uint8_t *)spi_buf, 1, 100);
182     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);
183
184     // Print out status register
185     uart_buf_len = sprintf(uart_buf,
186                             "Status: 0x x\r\n",
187                             (unsigned int)spi_buf[0]);
188     HAL_UART_Transmit(&huart2, (uint8_t *)uart_buf, uart_buf_len, 100);
189
190     /* USER CODE END 2 */
191
192     /* Infinite loop */
193     /* USER CODE BEGIN WHILE */
194   while (1)
195   {
196     /* USER CODE END WHILE */
197       // Tell TMP102 that we want to read from the temperature register
198         buf[0] = REG_TEMP;
199         ret = HAL_I2C_Master_Transmit(&hi2c1, TMP102_ADDR, buf, 1, HAL_MAX_DELAY);
200         if ( ret != HAL_OK ) {
201           strcpy((char*)buf, "Error Tx\r\n");
202         } else {
203
204           // Read 2 bytes from the temperature register
205           ret = HAL_I2C_Master_Receive(&hi2c1, TMP102_ADDR, buf, 2, HAL_MAX_DELAY);
206           if ( ret != HAL_OK ) {
207             strcpy((char*)buf, "Error Rx\r\n");
208           } else {
209
210             //Combine the bytes
211             val = ((int16_t)buf[0] << 4) | (buf[1] >> 4);
212
213             // Convert to 2's complement, since temperature can be negative
214             if ( val > 0x7FF ) {
215               val |= 0xF000;
216             }
217
218             // Convert to float temperature value (Celsius)
219             temp_c = val * 0.0625;
220
221             // Convert temperature to decimal format
222             temp_c *= 100;
223             sprintf((char*)buf,
224                   "%u.%u C\r\n",
225                   ((unsigned int)temp_c / 100),
226                   ((unsigned int)temp_c % 100));
227           }
228         }
```

```
229
230         // Send out buffer (temperature or error message)
231         HAL_UART_Transmit(&huart2, buf, strlen((char*)buf), HAL_MAX_DELAY);
232
233         // Wait
234         HAL_Delay(500);
235
236         /* USER CODE END WHILE */
237
238         /* USER CODE BEGIN 3 */
239       }
240       /* USER CODE END 3 */
241   /*USER CODE END 3 */
242 }
243
244 /**
245   * @brief System Clock Configuration
246   * @retval None
247   */
248 void SystemClock_Config(void)
249 {
250   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
251   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
252
253   /** Configure the main internal regulator output voltage
254   */
255   if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
256   {
257     Error_Handler();
258   }
259   /** Initializes the RCC Oscillators according to the specified parameters
260   * in the RCC_OscInitTypeDef structure.
261   */
262   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
263   RCC_OscInitStruct.HSEState = RCC_HSE_BYPASS;
264   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
265   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
266   {
267     Error_Handler();
268   }
269   /** Initializes the CPU, AHB and APB buses clocks
270   */
271   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
272                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
273   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSE;
274   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
275   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
276   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
277
278   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
279   {
280     Error_Handler();
281   }
282 }
283
284 /**
285   * @brief I2C1 Initialization Function
```

```
286   * @param None
287   * @retval None
288   */
289 static void MX_I2C1_SMBUS_Init(void)
290 {
291
292   /* USER CODE BEGIN I2C1_Init 0 */
293
294   /* USER CODE END I2C1_Init 0 */
295
296   /* USER CODE BEGIN I2C1_Init 1 */
297
298   /* USER CODE END I2C1_Init 1 */
299   hsmbus1.Instance = I2C1;
300   hsmbus1.Init.Timing = 0x00303D5B;
301   hsmbus1.Init.AnalogFilter = SMBUS_ANALOGFILTER_ENABLE;
302   hsmbus1.Init.OwnAddress1 = 2;
303   hsmbus1.Init.AddressingMode = SMBUS_ADDRESSINGMODE_7BIT;
304   hsmbus1.Init.DualAddressMode = SMBUS_DUALADDRESS_DISABLE;
305   hsmbus1.Init.OwnAddress2 = 0;
306   hsmbus1.Init.OwnAddress2Masks = SMBUS_OA2_NOMASK;
307   hsmbus1.Init.GeneralCallMode = SMBUS_GENERALCALL_DISABLE;
308   hsmbus1.Init.NoStretchMode = SMBUS_NOSTRETCH_DISABLE;
309   hsmbus1.Init.PacketErrorCheckMode = SMBUS_PEC_DISABLE;
310   hsmbus1.Init.PeripheralMode = SMBUS_PERIPHERAL_MODE_SMBUS_SLAVE;
311   hsmbus1.Init.SMBusTimeout = 0x000080C3;
312   if (HAL_SMBUS_Init(&hsmbus1) != HAL_OK)
313   {
314     Error_Handler();
315   }
316   /* USER CODE BEGIN I2C1_Init 2 */
317
318   /* USER CODE END I2C1_Init 2 */
319
320 }
321
322 /**
323   * @brief SPI1 Initialization Function
324   * @param None
325   * @retval None
326   */
327 static void MX_SPI1_Init(void)
328 {
329
330   /* USER CODE BEGIN SPI1_Init 0 */
331
332   /* USER CODE END SPI1_Init 0 */
333
334   /* USER CODE BEGIN SPI1_Init 1 */
335
336   /* USER CODE END SPI1_Init 1 */
337   /* SPI1 parameter configuration*/
338   hspi1.Instance = SPI1;
339   hspi1.Init.Mode = SPI_MODE_MASTER;
340   hspi1.Init.Direction = SPI_DIRECTION_2LINES;
341   hspi1.Init.DataSize = SPI_DATASIZE_4BIT;
342   hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
```

```
343   hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
344   hspi1.Init.NSS = SPI_NSS_SOFT;
345   hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
346   hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
347   hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
348   hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
349   hspi1.Init.CRCPolynomial = 7;
350   hspi1.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
351   hspi1.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
352   if (HAL_SPI_Init(&hspi1) != HAL_OK)
353   {
354     Error_Handler();
355   }
356   /* USER CODE BEGIN SPI1_Init 2 */
357
358   /* USER CODE END SPI1_Init 2 */
359
360 }
361
362 /**
363   * @brief TIM2 Initialization Function
364   * @param None
365   * @retval None
366   */
367 static void MX_TIM2_Init(void)
368 {
369
370   /* USER CODE BEGIN TIM2_Init 0 */
371
372   /* USER CODE END TIM2_Init 0 */
373
374   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
375   TIM_MasterConfigTypeDef sMasterConfig = {0};
376   TIM_OC_InitTypeDef sConfigOC = {0};
377
378   /* USER CODE BEGIN TIM2_Init 1 */
379
380   /* USER CODE END TIM2_Init 1 */
381   htim2.Instance = TIM2;
382   htim2.Init.Prescaler = 0;
383   htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
384   htim2.Init.Period = 4294967295;
385   htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
386   htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
387   if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
388   {
389     Error_Handler();
390   }
391   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
392   if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
393   {
394     Error_Handler();
395   }
396   if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
397   {
398     Error_Handler();
399   }
```

```
400   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
401   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
402   if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
403   {
404     Error_Handler();
405   }
406   sConfigOC.OCMode = TIM_OCMODE_PWM1;
407   sConfigOC.Pulse = 0;
408   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
409   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
410   if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
411   {
412     Error_Handler();
413   }
414   if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_4) != HAL_OK)
415   {
416     Error_Handler();
417   }
418   /* USER CODE BEGIN TIM2_Init 2 */
419
420   /* USER CODE END TIM2_Init 2 */
421   HAL_TIM_MspPostInit(&htim2);
422
423 }
424
425 /**
426   * @brief GPIO Initialization Function
427   * @param None
428   * @retval None
429   */
430 static void MX_GPIO_Init(void)
431 {
432   GPIO_InitTypeDef GPIO_InitStruct = {0};
433
434   /* GPIO Ports Clock Enable */
435   __HAL_RCC_GPIOC_CLK_ENABLE();
436   __HAL_RCC_GPIOA_CLK_ENABLE();
437   __HAL_RCC_GPIOB_CLK_ENABLE();
438   __HAL_RCC_GPIOH_CLK_ENABLE();
439
440   /*Configure GPIO pin Output Level */
441   HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_15, GPIO_PIN_RESET);
442
443   /*Configure GPIO pin Output Level */
444   HAL_GPIO_WritePin(GPIOB, GPIO_PIN_1|GPIO_PIN_6|GPIO_PIN_7, GPIO_PIN_RESET);
445
446   /*Configure GPIO pin Output Level */
447   HAL_GPIO_WritePin(GPIOH, GPIO_PIN_3, GPIO_PIN_RESET);
448
449   /*Configure GPIO pins : PA4 PA5 PA15 */
450   GPIO_InitStruct.Pin = GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_15;
451   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
452   GPIO_InitStruct.Pull = GPIO_NOPULL;
453   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
454   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
455
456   /*Configure GPIO pin : PA6 */
```

```
457   GPIO_InitStruct.Pin = GPIO_PIN_6;
458   GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
459   GPIO_InitStruct.Pull = GPIO_NOPULL;
460   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
461
462   /*Configure GPIO pins : PB1 PB6 PB7 */
463   GPIO_InitStruct.Pin = GPIO_PIN_1|GPIO_PIN_6|GPIO_PIN_7;
464   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
465   GPIO_InitStruct.Pull = GPIO_NOPULL;
466   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
467   HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
468
469   /*Configure GPIO pin : PH3 */
470   GPIO_InitStruct.Pin = GPIO_PIN_3;
471   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
472   GPIO_InitStruct.Pull = GPIO_NOPULL;
473   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
474   HAL_GPIO_Init(GPIOH, &GPIO_InitStruct);
475
476 }
477
478 /* USER CODE BEGIN 4 */
479
480 /* USER CODE END 4 */
481
482 /**
483   * @brief  This function is executed in case of error occurrence.
484   * @retval None
485   */
486 void Error_Handler(void)
487 {
488   /* USER CODE BEGIN Error_Handler_Debug */
489   /* User can add his own implementation to report the HAL error return state */
490   __disable_irq();
491   while (1)
492   {
493   }
494   /* USER CODE END Error_Handler_Debug */
495 }
496
497 #ifdef  USE_FULL_ASSERT
498 /**
499   * @brief  Reports the name of the source file and the source line number
500   *         where the assert_param error has occurred.
501   * @param  file: pointer to the source file name
502   * @param  line: assert_param error line source number
503   * @retval None
504   */
505 void assert_failed(uint8_t *file, uint32_t line)
506 {
507   /* USER CODE BEGIN 6 */
508   /* User can add his own implementation to report the file name and line number,
509      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
510   /* USER CODE END 6 */
511 }
512 #endif /* USE_FULL_ASSERT */
513
```

514