

```

1  /* USER CODE BEGIN Header */
2  /**
3   *
4   * @file           : main.c
5   * @brief          : Main program body
6   *
7   * This notice applies to any and all portions of this file
8   * that are not between comment pairs USER CODE BEGIN and
9   * USER CODE END. Other portions of this file, whether
10  * inserted by the user or by software development tools
11  * are owned by their respective copyright owners.
12  *
13  * COPYRIGHT(c) 2019 STMicroelectronics
14  *
15  * Redistribution and use in source and binary forms, with or without
modification,
16  * are permitted provided that the following conditions are met:
17  *   1. Redistributions of source code must retain the above copyright notice,
18  *      this list of conditions and the following disclaimer.
19  *   2. Redistributions in binary form must reproduce the above copyright notice,
20  *      this list of conditions and the following disclaimer in the documentation
21  *      and/or other materials provided with the distribution.
22  *   3. Neither the name of STMicroelectronics nor the names of its contributors
23  *      may be used to endorse or promote products derived from this software
24  *      without specific prior written permission.
25  *
26  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
27  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
28  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
29  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
30  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
31  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
32  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
33  * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
34  * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
35  * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
36  *
37  *
38  */
39  /* USER CODE END Header */
40
41  /* Includes ----- */
42  #include "main.h"
43
44  /* Private includes ----- */
45  /* USER CODE BEGIN Includes */
46  #include <stdio.h>
47  #include "unicon.h"
48  #include "ds18b20.h"
49  #include "motor.h"
50  /* USER CODE END Includes */
51
52  /* Private typedef ----- */
53  /* USER CODE BEGIN PTD */
54
55  /* USER CODE END PTD */
56
57  /* Private define ----- */
58  /* USER CODE BEGIN PD */
59  #define CH_POT_RV1      LL_ADC_CHANNEL_0
60  #define CH_POT_RV2      LL_ADC_CHANNEL_1
61  #define CH_HUMIDITY     LL_ADC_CHANNEL_5
62  #define CH_TEMPERATURE  LL_ADC_CHANNEL_4
63  /* USER CODE END PD */
64
65  /* Private macro ----- */

```

```

66  /* USER CODE BEGIN PM */
67
68  /* USER CODE END PM */
69
70  /* Private variables -----*/
71
72  /* USER CODE BEGIN PV */
73  const char version[] = "Ver. 1.0-20190423";
74  SysData_TypeDef SysData;
75
76  uint8_t ds_status;
77  /* USER CODE END PV */
78
79  /* Private function prototypes -----*/
80  void SystemClock_Config(void);
81  static void MX_GPIO_Init(void);
82  static void MX_ADC_Init(void);
83  static void MX_USART1_UART_Init(void);
84  static void MX_TIM16_Init(void);
85  /* USER CODE BEGIN PFP */
86
87  static void LedHandler(void);
88  static void NewMessageHandler(void);
89  static void SendOk(void);
90  static void TestModeHandler(SysData_TypeDef *self);
91  /* USER CODE END PFP */
92
93  /* Private user code -----*/
94  /* USER CODE BEGIN 0 */
95
96
97
98
99
100 /* USER CODE END 0 */
101
102 /**
103  * @brief The application entry point.
104  * @retval int
105  */
106 int main(void) {
107     /* USER CODE BEGIN 1 */
108     static volatile uint32_t delay = 0;
109     static uint8_t ds_delay = 0;
110
111     /* USER CODE END 1 */
112
113     /* MCU Configuration-----*/
114
115     /* Reset of all peripherals, Initializes the Flash interface and the Systick.
116 */
117
118     LL_APB1_GRP2_EnableClock(LL_APB1_GRP2_PERIPH_SYSCFG);
119     LL_APB1_GRP1_EnableClock(LL_APB1_GRP1_PERIPH_PWR);
120
121     /* System interrupt init*/
122
123     /* USER CODE BEGIN Init */
124
125     /* USER CODE END Init */
126
127     /* Configure the system clock */
128     SystemClock_Config();
129
130     /* USER CODE BEGIN SysInit */

```

```

131 SysTick_Config(SystemCoreClock/1000);
132 /* USER CODE END SysInit */
133
134 /* Initialize all configured peripherals */
135 MX_GPIO_Init();
136 MX_ADC_Init();
137 MX_USART1_UART_Init();
138 MX_TIM16_Init();
139 /* USER CODE BEGIN 2 */
140
141 LL_mDelay(10);
142
143 SysData.TestMode = 0;
144 SysData.HighTemperature = 25;
145 SysData.LowTemperature = 22;
146 SysData.HighHumidity = HUM_LEVEL_DEF;
147 SysData.MotorSpeed = MOTOR_SPEED_DEF;
148 SysData.MotorRunTime = MOTOR_DELAY_TIME_DEF;
149 SysData.MotorDelayTime = MOTOR_PAUSE_TIME_DEF;
150
151 UNI_Start();
152
153 DS18B20_PortInit();
154 ds_status = DS18B20_Init(SKIP_ROM);
155
156 DS18B20_MeasureTemperCmd(SKIP_ROM, 0);
157 Delay_ms(100);
158
159
160 L298_Init(&SysData);
161
162
163 /* USER CODE END 2 */
164
165 /* Infinite loop */
166 /* USER CODE BEGIN WHILE */
167
168 while (1) {
169
170     if(delay <= timestamp) {
171
172         delay = timestamp + 100;
173
174         ADC_Read_VREFINT();
175
176         SysData.ADC_Data.ch0 = ADC_ReadAnalog(CH_POT_RV1);
177         Delay_ms(10);
178
179         SysData.ADC_Data.ch1 = ADC_ReadAnalog(CH_POT_RV2);
180         Delay_ms(10);
181
182         SysData.ADC_Data.ch2 = ADC_ReadAnalog(CH_HUMIDITY);
183         Delay_ms(10);
184
185
186         if(ds_delay == 0) {
187
188             ds_delay = 10;
189
190             DS18B20_ReadStratchpad(SKIP_ROM, 0);
191             SysData.temper = DS18B20_Convert(0);
192
193             Delay_ms(10);
194             DS18B20_MeasureTemperCmd(SKIP_ROM, 0);
195         } else {
196

```

```

197         ds_delay--;
198     }
199
200
201     if(SysData.TestMode == 0) {
202
203         UNI_Process();
204
205         if(SysData.ADC_Data.ch2 >= SysData.HighHumidity) {
206             L298_CloseWindow(&SysData);
207             LED6_ON();
208         } else {
209
210             LED6_OFF();
211
212             if(SysData.temper >= SysData.HighTemperature) {
213                 L298_OpenWindow(&SysData);
214                 LED7_ON();
215             } else {
216
217                 if(SysData.temper < SysData.LowTemperature) {
218                     L298_CloseWindow(&SysData);
219                 }
220
221                 L298_Process(&SysData);
222
223                 LED7_OFF();
224             }
225         }
226     } else {
227         /* testas */
228
229         L298_Process(&SysData);
230
231         TestModeHandler(&SysData);
232     }
233 }
234
235
236 LedHandler();
237
238 if(NewMessageFlag) {
239
240     NewMessageHandler();
241
242     NewMessageFlag = false;
243 }
244
245 /* USER CODE END WHILE */
246
247 /* USER CODE BEGIN 3 */
248
249
250 }
251 /* USER CODE END 3 */
252 }
253
254 /**
255  * @brief System Clock Configuration
256  * @retval None
257  */
258 void SystemClock_Config(void) {
259     LL_FLASH_SetLatency(LL_FLASH_LATENCY_1);
260
261     if(LL_FLASH_GetLatency() != LL_FLASH_LATENCY_1) {
262         Error_Handler();

```

```

263     }
264     LL_RCC_HSI_Enable();
265
266     /* Wait till HSI is ready */
267     while(LL_RCC_HSI_IsReady() != 1) {
268
269     }
270     LL_RCC_HSI_SetCalibTrimming(16);
271     LL_RCC_HSI14_Enable();
272
273     /* Wait till HSI14 is ready */
274     while(LL_RCC_HSI14_IsReady() != 1) {
275
276     }
277     LL_RCC_HSI14_SetCalibTrimming(16);
278     LL_RCC_PLL_ConfigDomain_SYS(LL_RCC_PLLSOURCE_HSI_DIV_2, LL_RCC_PLL_MUL_12);
279     LL_RCC_PLL_Enable();
280
281     /* Wait till PLL is ready */
282     while(LL_RCC_PLL_IsReady() != 1) {
283
284     }
285     LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);
286     LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
287     LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_PLL);
288
289     /* Wait till System clock is ready */
290     while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_PLL) {
291
292     }
293     LL_InitlmsTick(48000000);
294     LL_SYSTICK_SetClkSource(LL_SYSTICK_CLKSOURCE_HCLK);
295     LL_SetSystemCoreClock(48000000);
296     LL_RCC_HSI14_EnableADCControl();
297     LL_RCC_SetUSARTClockSource(LL_RCC_USART1_CLKSOURCE_HSI);
298 }
299
300 /**
301  * @brief ADC Initialization Function
302  * @param None
303  * @retval None
304  */
305 static void MX_ADC_Init(void) {
306
307     /* USER CODE BEGIN ADC_Init 0 */
308
309     /* USER CODE END ADC_Init 0 */
310
311     LL_ADC_InitTypeDef ADC_InitStruct = {0};
312     LL_ADC_REG_InitTypeDef ADC_REG_InitStruct = {0};
313
314     LL_GPIO_InitTypeDef GPIO_InitStruct = {0};
315
316     /* Peripheral clock enable */
317     LL_APB1_GRP2_EnableClock(LL_APB1_GRP2_PERIPH_ADC1);
318
319     LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
320     /**ADC GPIO Configuration
321     PA0 -----> ADC_IN0
322     PA1 -----> ADC_IN1
323     PA2 -----> ADC_IN2
324     PA5 -----> ADC_IN5
325     */
326     GPIO_InitStruct.Pin = AD1_Pin;
327     GPIO_InitStruct.Mode = LL_GPIO_MODE_ANALOG;
328     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;

```

```

329     LL_GPIO_Init(AD1_GPIO_Port, &GPIO_InitStruct);
330
331     GPIO_InitStruct.Pin = AD2_Pin;
332     GPIO_InitStruct.Mode = LL_GPIO_MODE_ANALOG;
333     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
334     LL_GPIO_Init(AD2_GPIO_Port, &GPIO_InitStruct);
335
336     GPIO_InitStruct.Pin = ADC_Pin;
337     GPIO_InitStruct.Mode = LL_GPIO_MODE_ANALOG;
338     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
339     LL_GPIO_Init(ADC_GPIO_Port, &GPIO_InitStruct);
340
341     GPIO_InitStruct.Pin = HUMIDITY_Pin;
342     GPIO_InitStruct.Mode = LL_GPIO_MODE_ANALOG;
343     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
344     LL_GPIO_Init(HUMIDITY_GPIO_Port, &GPIO_InitStruct);
345
346     /* USER CODE BEGIN ADC_Init 1 */
347
348     /* USER CODE END ADC_Init 1 */
349     /** Configure Regular Channel
350     */
351     LL_ADC_REG_SetSequencerChAdd(ADC1, LL_ADC_CHANNEL_0);
352     /** Configure Regular Channel
353     */
354     LL_ADC_REG_SetSequencerChAdd(ADC1, LL_ADC_CHANNEL_1);
355     /** Configure Regular Channel
356     */
357     LL_ADC_REG_SetSequencerChAdd(ADC1, LL_ADC_CHANNEL_2);
358     /** Configure Regular Channel
359     */
360     LL_ADC_REG_SetSequencerChAdd(ADC1, LL_ADC_CHANNEL_5);
361     /** Configure Regular Channel
362     */
363     LL_ADC_REG_SetSequencerChAdd(ADC1, LL_ADC_PATH_INTERNAL_VREFINT);
364     /** Configure Internal Channel
365     */
366     LL_ADC_SetCommonPathInternalCh(__LL_ADC_COMMON_INSTANCE(ADC1),
LL_ADC_CHANNEL_VREFINT);
367     /** Configure the global features of the ADC (Clock, Resolution, Data Alignment
and number of conversion)
368     */
369     ADC_InitStruct.Clock = LL_ADC_CLOCK_ASYNC;
370     ADC_InitStruct.Resolution = LL_ADC_RESOLUTION_12B;
371     ADC_InitStruct.DataAlignment = LL_ADC_DATA_ALIGN_RIGHT;
372     ADC_InitStruct.LowPowerMode = LL_ADC_LP_MODE_NONE;
373     LL_ADC_Init(ADC1, &ADC_InitStruct);
374     ADC_REG_InitStruct.TriggerSource = LL_ADC_REG_TRIG_SOFTWARE;
375     ADC_REG_InitStruct.SequencerDiscont = LL_ADC_REG_SEQ_DISCONT_DISABLE;
376     ADC_REG_InitStruct.ContinuousMode = LL_ADC_REG_CONV_SINGLE;
377     ADC_REG_InitStruct.DMATransfer = LL_ADC_REG_DMA_TRANSFER_LIMITED;
378     ADC_REG_InitStruct.Overrun = LL_ADC_REG_OVR_DATA_PRESERVED;
379     LL_ADC_REG_Init(ADC1, &ADC_REG_InitStruct);
380     LL_ADC_REG_SetSequencerScanDirection(ADC1, LL_ADC_REG_SEQ_SCAN_DIR_FORWARD);
381     LL_ADC_SetSamplingTimeCommonChannels(ADC1, LL_ADC_SAMPLINGTIME_1CYCLE_5);
382     LL_ADC_DisableIT_EOC(ADC1);
383     LL_ADC_DisableIT_EOS(ADC1);
384     /* USER CODE BEGIN ADC_Init 2 */
385
386     /* USER CODE END ADC_Init 2 */
387
388 }
389
390 /**
391  * @brief TIM16 Initialization Function
392  * @param None

```

```

393     * @retval None
394 */
395 static void MX_TIM16_Init(void) {
396     /* USER CODE BEGIN TIM16_Init 0 */
397
398     /* USER CODE END TIM16_Init 0 */
399
400     LL_TIM_InitTypeDef TIM_InitStruct = {0};
401     LL_TIM_OC_InitTypeDef TIM_OC_InitStruct = {0};
402     LL_TIM_BDTR_InitTypeDef TIM_BDTRInitStruct = {0};
403
404     LL_GPIO_InitTypeDef GPIO_InitStruct = {0};
405
406     /* Peripheral clock enable */
407     LL_APB1_GRP2_EnableClock(LL_APB1_GRP2_PERIPH_TIM16);
408
409     /* USER CODE BEGIN TIM16_Init 1 */
410
411     /* USER CODE END TIM16_Init 1 */
412     TIM_InitStruct.Prescaler = 0;
413     TIM_InitStruct.CounterMode = LL_TIM_COUNTERMODE_UP;
414     TIM_InitStruct.Autoreload = 1000;
415     TIM_InitStruct.ClockDivision = LL_TIM_CLOCKDIVISION_DIV1;
416     TIM_InitStruct.RepetitionCounter = 0;
417     LL_TIM_Init(TIM16, &TIM_InitStruct);
418     LL_TIM_DisableARRPreload(TIM16);
419     LL_TIM_OC_EnablePreload(TIM16, LL_TIM_CHANNEL_CH1);
420     TIM_OC_InitStruct.OCMode = LL_TIM_OCMODE_PWM1;
421     TIM_OC_InitStruct.OCState = LL_TIM_OCSTATE_DISABLE;
422     TIM_OC_InitStruct.OCNState = LL_TIM_OCSTATE_DISABLE;
423     TIM_OC_InitStruct.CompareValue = 500;
424     TIM_OC_InitStruct.OCPolarity = LL_TIM_OCPOLARITY_HIGH;
425     TIM_OC_InitStruct.OCNPolarity = LL_TIM_OCPOLARITY_HIGH;
426     TIM_OC_InitStruct.OCIDleState = LL_TIM_OCIDLESTATE_HIGH;
427     TIM_OC_InitStruct.OCNIdleState = LL_TIM_OCIDLESTATE_LOW;
428     LL_TIM_OC_Init(TIM16, LL_TIM_CHANNEL_CH1, &TIM_OC_InitStruct);
429     LL_TIM_OC_DisableFast(TIM16, LL_TIM_CHANNEL_CH1);
430     TIM_BDTRInitStruct.OSSRState = LL_TIM_OSSR_DISABLE;
431     TIM_BDTRInitStruct.OSSIState = LL_TIM_OSSI_DISABLE;
432     TIM_BDTRInitStruct.LockLevel = LL_TIM_LOCKLEVEL_OFF;
433     TIM_BDTRInitStruct.DeadTime = 0;
434     TIM_BDTRInitStruct.BreakState = LL_TIM_BREAK_DISABLE;
435     TIM_BDTRInitStruct.BreakPolarity = LL_TIM_BREAK_POLARITY_HIGH;
436     TIM_BDTRInitStructAutomaticOutput = LL_TIM_AUTOMATICOUTPUT_DISABLE;
437     LL_TIM_BDTR_Init(TIM16, &TIM_BDTRInitStruct);
438     /* USER CODE BEGIN TIM16_Init 2 */
439
440     /* USER CODE END TIM16_Init 2 */
441     LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
442     /**TIM16 GPIO Configuration
443     PB8 -----> TIM16_CH1
444     */
445     GPIO_InitStruct.Pin = L298_IN2_Pin;
446     GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
447     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
448     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_OPENDRAIN;
449     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
450     GPIO_InitStruct.Alternate = LL_GPIO_AF_2;
451     LL_GPIO_Init(L298_IN2_GPIO_Port, &GPIO_InitStruct);
452
453 }
454
455 /**
456 * @brief USART1 Initialization Function
457 * @param None

```

```

459     * @retval None
460 */
461 static void MX_USART1_UART_Init(void) {
462     /* USER CODE BEGIN USART1_Init 0 */
463
464     /* USER CODE END USART1_Init 0 */
465
466     LL_USART_InitTypeDef USART_InitStruct = {0};
467
468     LL_GPIO_InitTypeDef GPIO_InitStruct = {0};
469
470     /* Peripheral clock enable */
471     LL_APB1_GRP2_EnableClock(LL_APB1_GRP2_PERIPH_USART1);
472
473     LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
474     /**USART1 GPIO Configuration
475     PA9 -----> USART1_TX
476     PA10 -----> USART1_RX
477 */
478     GPIO_InitStruct.Pin = LL_GPIO_PIN_9;
479     GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
480     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_HIGH;
481     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
482     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
483     GPIO_InitStruct.Alternate = LL_GPIO_AF_1;
484     LL_GPIO_Init(GPIOA, &GPIO_InitStruct);
485
486     GPIO_InitStruct.Pin = LL_GPIO_PIN_10;
487     GPIO_InitStruct.Mode = LL_GPIO_MODE_ALTERNATE;
488     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_HIGH;
489     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
490     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
491     GPIO_InitStruct.Alternate = LL_GPIO_AF_1;
492     LL_GPIO_Init(GPIOA, &GPIO_InitStruct);
493
494     /* USART1 interrupt Init */
495     NVIC_SetPriority(USART1_IRQn, 0);
496     NVIC_EnableIRQ(USART1_IRQn);
497
498     /* USER CODE BEGIN USART1_Init 1 */
499
500     /* USER CODE END USART1_Init 1 */
501     USART_InitStruct.BaudRate = 38400;
502     USART_InitStruct.DataWidth = LL_USART_DATAWIDTH_8B;
503     USART_InitStruct.StopBits = LL_USART_STOPBITS_1;
504     USART_InitStruct.Parity = LL_USART_PARITY_NONE;
505     USART_InitStruct.TransferDirection = LL_USART_DIRECTION_TX_RX;
506     USART_InitStruct.HardwareFlowControl = LL_USART_HWCONTROL_NONE;
507     USART_InitStruct.OverSampling = LL_USART_OVERSAMPLING_16;
508     LL_USART_Init(USART1, &USART_InitStruct);
509     LL_USART_DisableIT_CTS(USART1);
510     LL_USART_DisableOverrunDetect(USART1);
511     LL_USART_ConfigAsyncMode(USART1);
512     LL_USART_Enable(USART1);
513     /* USER CODE BEGIN USART1_Init 2 */
514
515     /* USER CODE END USART1_Init 2 */
516
517 }
518
519 /**
520 * @brief GPIO Initialization Function
521 * @param None
522 * @retval None
523 */
524

```



```

525 static void MX_GPIO_Init(void) {
526     LL_GPIO_InitTypeDef GPIO_InitStruct = {0};
527
528     /* GPIO Ports Clock Enable */
529     LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOA);
530     LL_AHB1_GRP1_EnableClock(LL_AHB1_GRP1_PERIPH_GPIOB);
531
532     /**/
533     LL_GPIO_ResetOutputPin(LED5_GPIO_Port, LED5_Pin);
534
535     /**/
536     LL_GPIO_ResetOutputPin(LED2_GPIO_Port, LED2_Pin);
537
538     /**/
539     LL_GPIO_ResetOutputPin(LED7_GPIO_Port, LED7_Pin);
540
541     /**/
542     LL_GPIO_ResetOutputPin(LED6_GPIO_Port, LED6_Pin);
543
544     /**/
545     LL_GPIO_ResetOutputPin(GPIOB, LL_GPIO_PIN_12);
546
547     /**/
548     LL_GPIO_ResetOutputPin(L298_IN1_GPIO_Port, L298_IN1_Pin);
549
550     /**/
551     LL_GPIO_ResetOutputPin(L298_ENA_GPIO_Port, L298_ENA_Pin);
552
553     /**/
554     GPIO_InitStruct.Pin = LED5_Pin;
555     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
556     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
557     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
558     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
559     LL_GPIO_Init(LED5_GPIO_Port, &GPIO_InitStruct);
560
561     /**/
562     GPIO_InitStruct.Pin = LED2_Pin;
563     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
564     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
565     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
566     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
567     LL_GPIO_Init(LED2_GPIO_Port, &GPIO_InitStruct);
568
569     /**/
570     GPIO_InitStruct.Pin = LED7_Pin;
571     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
572     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
573     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
574     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
575     LL_GPIO_Init(LED7_GPIO_Port, &GPIO_InitStruct);
576
577     /**/
578     GPIO_InitStruct.Pin = LED6_Pin;
579     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
580     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
581     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
582     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
583     LL_GPIO_Init(LED6_GPIO_Port, &GPIO_InitStruct);
584
585     /**/
586     GPIO_InitStruct.Pin = LL_GPIO_PIN_11;
587     GPIO_InitStruct.Mode = LL_GPIO_MODE_ANALOG;
588     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
589     LL_GPIO_Init(GPIOB, &GPIO_InitStruct);
590

```

```

591     /**/
592     GPIO_InitStruct.Pin = LL_GPIO_PIN_12;
593     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
594     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_MEDIUM;
595     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_OPENDRAIN;
596     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
597     LL_GPIO_Init(GPIOB, &GPIO_InitStruct);
598
599     /**/
600     GPIO_InitStruct.Pin = L298_IN1_Pin;
601     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
602     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
603     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_OPENDRAIN;
604     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
605     LL_GPIO_Init(L298_IN1_GPIO_Port, &GPIO_InitStruct);
606
607     /**/
608     GPIO_InitStruct.Pin = L298_ENA_Pin;
609     GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
610     GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_LOW;
611     GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSH_PULL;
612     GPIO_InitStruct.Pull = LL_GPIO_PULL_NO;
613     LL_GPIO_Init(L298_ENA_GPIO_Port, &GPIO_InitStruct);
614
615 }
616
617 /* USER CODE BEGIN 4 */
618
619 static void LedHandler(void) {
620
621     static uint32_t delay = 0;
622     static uint8_t led = 0;
623
624     if(delay < timestamp) {
625
626         delay = timestamp + 100;
627
628         if(SysData.TestMode) {
629
630             LED2_OFF();
631             LED5_OFF();
632             LED6_OFF();
633             LED7_OFF();
634
635             switch(led) {
636             case 0:
637                 LED2_ON();
638                 led = 1;
639                 break;
640             case 1:
641                 LED5_ON();
642                 led = 2;
643                 break;
644             case 2:
645                 LED6_ON();
646                 led = 3;
647                 break;
648             case 3:
649                 LED7_ON();
650                 led = 0;
651                 break;
652             }
653
654         } else {
655
656             if(SysData.WindowState) LED2_ON();

```

```

657         else LED2_OFF();
658
659         if(LL_GPIO_IsInputPinSet(GPIOB, LL_GPIO_PIN_9) ) LED5_ON();
660         else LED5_OFF();
661     }
662 }
663 }
664 }
665
666
667 /* */
668 static void NewMessageHandler(void) {
669
670     if( !strcmp(ptrPrimaryRxBuffer, "AT+HTEMP=", 9 )) {
671
672         uint8_t tmp = atoi(ptrPrimaryRxBuffer+9);
673
674         if(tmp < 22 || tmp > 32) {
675
676             sprintf(ptrPrimaryTxBuffer, "%s%02u%s", "BAD PARAM: HTEMP=", tmp,
677 "\r\n");
678             USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
679             while(RespondWaitingFlag);
680
681             USART_ClearRxBuffer(PRIMARY_PORT);
682             return;
683         }
684
685         SysData.HighTemperature = tmp;
686         USART_ClearRxBuffer(PRIMARY_PORT);
687         SendOk();
688         return;
689     }
690
691     if( !strcmp(ptrPrimaryRxBuffer, "AT+HTEMP?", 9 )) {
692
693         sprintf(ptrPrimaryTxBuffer, "%s%02u%s", "HTEMP=", SysData.HighTemperature,
694 "\r\n");
695         USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
696         while(RespondWaitingFlag);
697
698         USART_ClearRxBuffer(PRIMARY_PORT);
699         return;
700     }
701
702     if( !strcmp(ptrPrimaryRxBuffer, "AT+LTEMP=", 9 )) {
703
704         uint8_t tmp = atoi(ptrPrimaryRxBuffer+9);
705
706         if(tmp < 15 || tmp > 28) {
707
708             sprintf(ptrPrimaryTxBuffer, "%s%02u%s", "BAD PARAM: LTEMP=", tmp,
709 "\r\n");
710             USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
711             while(RespondWaitingFlag);
712
713             USART_ClearRxBuffer(PRIMARY_PORT);
714             return;
715         }
716
717         SysData.LowTemperature = tmp;
718         USART_ClearRxBuffer(PRIMARY_PORT);
719         SendOk();
720         return;
721     }

```

```

720
721     if( !strcmp(ptrPrimaryRxBuffer, "AT+LTEMP?", 9 )) {
722
723         sprintf(ptrPrimaryTxBuffer, "%s%02u%s", "LTEMP=", SysData.LowTemperature,
"\r\n");
724         USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
725         while(RespondWaitingFlag);
726
727         USART_ClearRxBuffer(PRIMARY_PORT);
728         return;
729     }
730
731
732     if( !strcmp(ptrPrimaryRxBuffer, "AT+HUMLEVEL=", 12 )) {
733
734         uint16_t tmp = atoi(ptrPrimaryRxBuffer+12);
735
736         if(tmp < 200 || tmp > 800) {
737
738             sprintf(ptrPrimaryTxBuffer, "%s%02u%s", "BAD PARAM: HUMLEVEL=", tmp,
"\r\n");
739             USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
740             while(RespondWaitingFlag);
741
742             USART_ClearRxBuffer(PRIMARY_PORT);
743             return;
744         }
745
746         SysData.HighHumidity = tmp;
747         USART_ClearRxBuffer(PRIMARY_PORT);
748         SendOk();
749         return;
750     }
751
752     if( !strcmp(ptrPrimaryRxBuffer, "AT+HUMLEVEL?", 12 )) {
753
754         sprintf(ptrPrimaryTxBuffer, "%s%02u%s", "HUMLEVEL=", SysData.HighHumidity,
"\r\n");
755         USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
756         while(RespondWaitingFlag);
757
758         USART_ClearRxBuffer(PRIMARY_PORT);
759         return;
760     }
761
762
763     if( !strcmp(ptrPrimaryRxBuffer, "AT+SPEED=", 9 )) {
764
765         uint16_t tmp = atoi(ptrPrimaryRxBuffer+9);
766
767         if(tmp < 300 || tmp > 800) {
768
769             sprintf(ptrPrimaryTxBuffer, "%s%02u%s", "BAD PARAM: SPEED=", tmp,
"\r\n");
770             USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
771             while(RespondWaitingFlag);
772
773             USART_ClearRxBuffer(PRIMARY_PORT);
774             return;
775         }
776
777         SysData.MotorSpeed = tmp;
778
779         USART_ClearRxBuffer(PRIMARY_PORT);
780         SendOk();
781

```

```

782     return;
783 }
784
785 if( !strcmp(ptrPrimaryRxBuffer, "AT+SPEED?", 9 )) {
786
787     sprintf(ptrPrimaryTxBuffer, "%s%s", "SPEED=", SysData.MotorSpeed, "\r\n"
);
788     USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
789     while(RespondWaitingFlag);
790
791     USART_ClearRxBuffer(PRIMARY_PORT);
792
793     return;
794 }
795
796
797 if( !strcmp(ptrPrimaryRxBuffer, "AT+RUNTIME=", 11 )) {
798
799     uint8_t tmp = atoi(ptrPrimaryRxBuffer+11);
800
801     if(tmp < 2 || tmp > 10) {
802
803         sprintf(ptrPrimaryTxBuffer, "%s%s", "BAD PARAM: RUNTIME=", tmp,
"\r\n");
804         USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
805         while(RespondWaitingFlag);
806
807         USART_ClearRxBuffer(PRIMARY_PORT);
808         return;
809     }
810
811     SysData.MotorRunTime = tmp;
812     USART_ClearRxBuffer(PRIMARY_PORT);
813     SendOk();
814     return;
815 }
816
817 if( !strcmp(ptrPrimaryRxBuffer, "AT+RUNTIME?", 11 )) {
818
819     sprintf(ptrPrimaryTxBuffer, "%s%s", "RUNTIME=", SysData.MotorRunTime,
"\r\n");
820     USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
821     while(RespondWaitingFlag);
822
823     USART_ClearRxBuffer(PRIMARY_PORT);
824     return;
825 }
826
827
828
829 if( !strcmp(ptrPrimaryRxBuffer, "AT+DELAYTIME=", 13 )) {
830
831     uint8_t tmp = atoi(ptrPrimaryRxBuffer+13);
832
833     if(tmp < 1 || tmp > 3) {
834
835         sprintf(ptrPrimaryTxBuffer, "%s%s", "BAD PARAM: DELAYTIME=", tmp,
"\r\n");
836         USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
837         while(RespondWaitingFlag);
838
839         USART_ClearRxBuffer(PRIMARY_PORT);
840         return;
841     }
842
843     SysData.MotorDelayTime = tmp;

```

```

844     USART_ClearRxBuffer(PRIMARY_PORT);
845     SendOk();
846     return;
847 }
848
849 if( !strcmp(ptrPrimaryRxBuffer, "AT+DELAYTIME?", 13 )) {
850
851     sprintf(ptrPrimaryTxBuffer, "%s%s", "DELAYTIME=", SysData.MotorDelayTime,
"\r\n");
852     USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
853     while(RespondWaitingFlag);
854
855     USART_ClearRxBuffer(PRIMARY_PORT);
856     return;
857 }
858
859
860 if( !strcmp(ptrPrimaryRxBuffer, "AT+INF", 6 )) {
861
862     sprintf(ptrPrimaryTxBuffer, "%s%02.02f%s%s", "TEMP=", SysData.temper, "
HUM=", SysData.ADC_Data.ch2, "\r\n");
863     USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
864     while(RespondWaitingFlag);
865
866     USART_ClearRxBuffer(PRIMARY_PORT);
867
868     return;
869 }
870
871
872 if( !strcmp(ptrPrimaryRxBuffer, "AT+VER", 6 )) {
873
874     sprintf(ptrPrimaryTxBuffer, "%s%s", version, "\r\n");
875     USART_SendString(PRIMARY_PORT, ptrPrimaryTxBuffer);
876     while(RespondWaitingFlag);
877
878     USART_ClearRxBuffer(PRIMARY_PORT);
879
880     return;
881 }
882
883
884
885 if( !strcmp(ptrPrimaryRxBuffer, "AT+TEST=", 8 )) {
886
887     SysData.TestMode = atoi(ptrPrimaryRxBuffer+8);
888
889     if(SysData.TestMode) {
890         /* jungiam Testini tezima */
891         L298_CloseWindow(&SysData);
892         SysData.WindowState = Closed;
893     } else {
894
895
896
897     }
898
899     USART_ClearRxBuffer(PRIMARY_PORT);
900     SendOk();
901     return;
902 }
903
904
905 /* papildomos testinio rezimo komandos */
906 if(SysData.TestMode) {
907

```

```

908         if( !strcmp(ptrPrimaryRxBuffer, "AT+OPEN", 7 )) {
909
910             if(SysData.WindowState != Opened){
911
912                 L298_OpenWindow(&SysData);
913                 SysData.WindowState = Opened;
914
915                 SendOk();
916             }else{
917
918                 USART_SendString(PRIMARY_PORT, "Window is opened!\r\n");
919                 while(RespondWaitingFlag);
920             }
921
922             return;
923         }
924
925         if( !strcmp(ptrPrimaryRxBuffer, "AT+CLOSE", 8 )) {
926
927             if(SysData.WindowState != Closed){
928
929                 L298_CloseWindow(&SysData);
930                 SysData.WindowState = Closed;
931
932                 SendOk();
933             }else{
934
935                 USART_SendString(PRIMARY_PORT, "Window is closed!\r\n");
936                 while(RespondWaitingFlag);
937             }
938
939             return;
940         }
941
942     }
943
944
945     USART_SendString(PRIMARY_PORT, "BAD_CMD\r\n");
946     while(RespondWaitingFlag);
947
948     USART_ClearRxBuffer(PRIMARY_PORT);
949 }
950
951
952
953 /* */
954 static void SendOk(void) {
955
956     USART_SendString(PRIMARY_PORT, "OK\r\n");
957     while(RespondWaitingFlag);
958 }
959
960
961 /* */
962 static void TestModeHandler(SysData_TypeDef *self) {
963
964
965
966
967
968 }
969
970
971 /* USER CODE END 4 */
972
973 /**

```

```

974     * @brief This function is executed in case of error occurrence.
975     * @retval None
976     */
977 void Error_Handler(void) {
978     /* USER CODE BEGIN Error_Handler_Debug */
979     /* User can add his own implementation to report the HAL error return state */
980
981     /* USER CODE END Error_Handler_Debug */
982 }
983
984 #ifdef USE_FULL_ASSERT
985 /**
986     * @brief Reports the name of the source file and the source line number
987     * where the assert_param error has occurred.
988     * @param file: pointer to the source file name
989     * @param line: assert_param error line source number
990     * @retval None
991     */
992 void assert_failed(char *file, uint32_t line) {
993     /* USER CODE BEGIN 6 */
994     /* User can add his own implementation to report the file name and line number,
995     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
996     /* USER CODE END 6 */
997 }
998 #endif /* USE_FULL_ASSERT */
999
1000 /***** (C) COPYRIGHT STMicroelectronics *****END OF FILE*****/

```