```c
/**
  ******************************************************************************
  * File Name          : TIM.c
  * Description        : This file provides code for the configuration
  *                      of the TIM instances.
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "tim.h"

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;
TIM_HandleTypeDef htim8;
TIM_HandleTypeDef htim10;
TIM_HandleTypeDef htim14;

/* TIM1 init function */
void MX_TIM1_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};
  TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 41;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 9999;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
```

```
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 5000;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
  sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
  if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
  {
    Error_Handler();
  }
  sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
  sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
  sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
  sBreakDeadTimeConfig.DeadTime = 0;
  sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
  sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
  sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
  if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
  {
    Error_Handler();
  }
  HAL_TIM_MspPostInit(&htim1);

}
/* TIM2 init function */
void MX_TIM2_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 41;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 9999;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
```

```
125   }
126   sConfigOC.OCMode = TIM_OCMODE_PWM1;
127   sConfigOC.Pulse = 5000;
128   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
129   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
130   if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
131   {
132     Error_Handler();
133   }
134   HAL_TIM_MspPostInit(&htim2);
135
136 }
137 /* TIM3 init function */
138 void MX_TIM3_Init(void)
139 {
140   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
141   TIM_MasterConfigTypeDef sMasterConfig = {0};
142   TIM_OC_InitTypeDef sConfigOC = {0};
143
144   htim3.Instance = TIM3;
145   htim3.Init.Prescaler = 41;
146   htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
147   htim3.Init.Period = 9999;
148   htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
149   htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
150   if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
151   {
152     Error_Handler();
153   }
154   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
155   if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
156   {
157     Error_Handler();
158   }
159   if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
160   {
161     Error_Handler();
162   }
163   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
164   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
165   if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
166   {
167     Error_Handler();
168   }
169   sConfigOC.OCMode = TIM_OCMODE_PWM1;
170   sConfigOC.Pulse = 5000;
171   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
172   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
173   if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
174   {
175     Error_Handler();
176   }
177   HAL_TIM_MspPostInit(&htim3);
178
179 }
180 /* TIM4 init function */
181 void MX_TIM4_Init(void)
182 {
183   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
184   TIM_MasterConfigTypeDef sMasterConfig = {0};
185   TIM_OC_InitTypeDef sConfigOC = {0};
186
```

```
187   htim4.Instance = TIM4;
188   htim4.Init.Prescaler = 41;
189   htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
190   htim4.Init.Period = 9999;
191   htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
192   htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
193   if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
194   {
195     Error_Handler();
196   }
197   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
198   if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
199   {
200     Error_Handler();
201   }
202   if (HAL_TIM_PWM_Init(&htim4) != HAL_OK)
203   {
204     Error_Handler();
205   }
206   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
207   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
208   if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
209   {
210     Error_Handler();
211   }
212   sConfigOC.OCMode = TIM_OCMODE_PWM1;
213   sConfigOC.Pulse = 5000;
214   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
215   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
216   if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
217   {
218     Error_Handler();
219   }
220   HAL_TIM_MspPostInit(&htim4);
221
222 }
223 /* TIM8 init function */
224 void MX_TIM8_Init(void)
225 {
226   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
227   TIM_MasterConfigTypeDef sMasterConfig = {0};
228   TIM_OC_InitTypeDef sConfigOC = {0};
229   TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};
230
231   htim8.Instance = TIM8;
232   htim8.Init.Prescaler = 41;
233   htim8.Init.CounterMode = TIM_COUNTERMODE_UP;
234   htim8.Init.Period = 9999;
235   htim8.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
236   htim8.Init.RepetitionCounter = 0;
237   htim8.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
238   if (HAL_TIM_Base_Init(&htim8) != HAL_OK)
239   {
240     Error_Handler();
241   }
242   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
243   if (HAL_TIM_ConfigClockSource(&htim8, &sClockSourceConfig) != HAL_OK)
244   {
245     Error_Handler();
246   }
247   if (HAL_TIM_PWM_Init(&htim8) != HAL_OK)
248   {
```

```
249     Error_Handler();
250   }
251   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
252   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
253   if (HAL_TIMEx_MasterConfigSynchronization(&htim8, &sMasterConfig) != HAL_OK)
254   {
255     Error_Handler();
256   }
257   sConfigOC.OCMode = TIM_OCMODE_PWM1;
258   sConfigOC.Pulse = 5000;
259   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
260   sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
261   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
262   sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
263   sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
264   if (HAL_TIM_PWM_ConfigChannel(&htim8, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
265   {
266     Error_Handler();
267   }
268   sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
269   sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
270   sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
271   sBreakDeadTimeConfig.DeadTime = 0;
272   sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
273   sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
274   sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
275   if (HAL_TIMEx_ConfigBreakDeadTime(&htim8, &sBreakDeadTimeConfig) != HAL_OK)
276   {
277     Error_Handler();
278   }
279   HAL_TIM_MspPostInit(&htim8);
280
281 }
282 /* TIM10 init function */
283 void MX_TIM10_Init(void)
284 {
285   TIM_OC_InitTypeDef sConfigOC = {0};
286
287   htim10.Instance = TIM10;
288   htim10.Init.Prescaler = 81;
289   htim10.Init.CounterMode = TIM_COUNTERMODE_UP;
290   htim10.Init.Period = 19999;
291   htim10.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
292   htim10.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
293   if (HAL_TIM_Base_Init(&htim10) != HAL_OK)
294   {
295     Error_Handler();
296   }
297   if (HAL_TIM_PWM_Init(&htim10) != HAL_OK)
298   {
299     Error_Handler();
300   }
301   sConfigOC.OCMode = TIM_OCMODE_PWM1;
302   sConfigOC.Pulse = 0;
303   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
304   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
305   if (HAL_TIM_PWM_ConfigChannel(&htim10, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
306   {
307     Error_Handler();
308   }
309   HAL_TIM_MspPostInit(&htim10);
310
```

```
311 }
312 /* TIM14 init function */
313 void MX_TIM14_Init(void)
314 {
315   TIM_OC_InitTypeDef sConfigOC = {0};
316
317   htim14.Instance = TIM14;
318   htim14.Init.Prescaler = 41;
319   htim14.Init.CounterMode = TIM_COUNTERMODE_UP;
320   htim14.Init.Period = 9999;
321   htim14.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
322   htim14.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
323   if (HAL_TIM_Base_Init(&htim14) != HAL_OK)
324   {
325     Error_Handler();
326   }
327   if (HAL_TIM_PWM_Init(&htim14) != HAL_OK)
328   {
329     Error_Handler();
330   }
331   sConfigOC.OCMode = TIM_OCMODE_PWM1;
332   sConfigOC.Pulse = 5000;
333   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
334   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
335   if (HAL_TIM_PWM_ConfigChannel(&htim14, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
336   {
337     Error_Handler();
338   }
339   HAL_TIM_MspPostInit(&htim14);
340
341 }
342
343 void HAL_TIM_Base_MspInit(TIM_HandleTypeDef* tim_baseHandle)
344 {
345
346   if(tim_baseHandle->Instance==TIM1)
347   {
348   /* USER CODE BEGIN TIM1_MspInit 0 */
349
350   /* USER CODE END TIM1_MspInit 0 */
351     /* TIM1 clock enable */
352     __HAL_RCC_TIM1_CLK_ENABLE();
353
354     /* TIM1 interrupt Init */
355     HAL_NVIC_SetPriority(TIM1_UP_TIM10_IRQn, 0, 0);
356     HAL_NVIC_EnableIRQ(TIM1_UP_TIM10_IRQn);
357   /* USER CODE BEGIN TIM1_MspInit 1 */
358
359   /* USER CODE END TIM1_MspInit 1 */
360   }
361   else if(tim_baseHandle->Instance==TIM2)
362   {
363   /* USER CODE BEGIN TIM2_MspInit 0 */
364
365   /* USER CODE END TIM2_MspInit 0 */
366     /* TIM2 clock enable */
367     __HAL_RCC_TIM2_CLK_ENABLE();
368
369     /* TIM2 interrupt Init */
370     HAL_NVIC_SetPriority(TIM2_IRQn, 0, 0);
371     HAL_NVIC_EnableIRQ(TIM2_IRQn);
372   /* USER CODE BEGIN TIM2_MspInit 1 */
```

```
373
374   /* USER CODE END TIM2_MspInit 1 */
375   }
376   else if(tim_baseHandle->Instance==TIM3)
377   {
378   /* USER CODE BEGIN TIM3_MspInit 0 */
379
380   /* USER CODE END TIM3_MspInit 0 */
381     /* TIM3 clock enable */
382     __HAL_RCC_TIM3_CLK_ENABLE();
383
384     /* TIM3 interrupt Init */
385     HAL_NVIC_SetPriority(TIM3_IRQn, 0, 0);
386     HAL_NVIC_EnableIRQ(TIM3_IRQn);
387   /* USER CODE BEGIN TIM3_MspInit 1 */
388
389   /* USER CODE END TIM3_MspInit 1 */
390   }
391   else if(tim_baseHandle->Instance==TIM4)
392   {
393   /* USER CODE BEGIN TIM4_MspInit 0 */
394
395   /* USER CODE END TIM4_MspInit 0 */
396     /* TIM4 clock enable */
397     __HAL_RCC_TIM4_CLK_ENABLE();
398
399     /* TIM4 interrupt Init */
400     HAL_NVIC_SetPriority(TIM4_IRQn, 0, 0);
401     HAL_NVIC_EnableIRQ(TIM4_IRQn);
402   /* USER CODE BEGIN TIM4_MspInit 1 */
403
404   /* USER CODE END TIM4_MspInit 1 */
405   }
406   else if(tim_baseHandle->Instance==TIM8)
407   {
408   /* USER CODE BEGIN TIM8_MspInit 0 */
409
410   /* USER CODE END TIM8_MspInit 0 */
411     /* TIM8 clock enable */
412     __HAL_RCC_TIM8_CLK_ENABLE();
413
414     /* TIM8 interrupt Init */
415     HAL_NVIC_SetPriority(TIM8_UP_TIM13_IRQn, 0, 0);
416     HAL_NVIC_EnableIRQ(TIM8_UP_TIM13_IRQn);
417     HAL_NVIC_SetPriority(TIM8_TRG_COM_TIM14_IRQn, 0, 0);
418     HAL_NVIC_EnableIRQ(TIM8_TRG_COM_TIM14_IRQn);
419   /* USER CODE BEGIN TIM8_MspInit 1 */
420
421   /* USER CODE END TIM8_MspInit 1 */
422   }
423   else if(tim_baseHandle->Instance==TIM10)
424   {
425   /* USER CODE BEGIN TIM10_MspInit 0 */
426
427   /* USER CODE END TIM10_MspInit 0 */
428     /* TIM10 clock enable */
429     __HAL_RCC_TIM10_CLK_ENABLE();
430
431     /* TIM10 interrupt Init */
432     HAL_NVIC_SetPriority(TIM1_UP_TIM10_IRQn, 0, 0);
433     HAL_NVIC_EnableIRQ(TIM1_UP_TIM10_IRQn);
434   /* USER CODE BEGIN TIM10_MspInit 1 */
```

```
435
436   /* USER CODE END TIM10_MspInit 1 */
437   }
438   else if(tim_baseHandle->Instance==TIM14)
439   {
440   /* USER CODE BEGIN TIM14_MspInit 0 */
441
442   /* USER CODE END TIM14_MspInit 0 */
443     /* TIM14 clock enable */
444     __HAL_RCC_TIM14_CLK_ENABLE();
445
446     /* TIM14 interrupt Init */
447     HAL_NVIC_SetPriority(TIM8_TRG_COM_TIM14_IRQn, 0, 0);
448     HAL_NVIC_EnableIRQ(TIM8_TRG_COM_TIM14_IRQn);
449   /* USER CODE BEGIN TIM14_MspInit 1 */
450
451   /* USER CODE END TIM14_MspInit 1 */
452   }
453 }
454 void HAL_TIM_MspPostInit(TIM_HandleTypeDef* timHandle)
455 {
456
457   GPIO_InitTypeDef GPIO_InitStruct = {0};
458   if(timHandle->Instance==TIM1)
459   {
460   /* USER CODE BEGIN TIM1_MspPostInit 0 */
461
462   /* USER CODE END TIM1_MspPostInit 0 */
463     __HAL_RCC_GPIOA_CLK_ENABLE();
464     /**TIM1 GPIO Configuration
465     PA10     ------> TIM1_CH3
466     */
467     GPIO_InitStruct.Pin = GPIO_PIN_10;
468     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
469     GPIO_InitStruct.Pull = GPIO_NOPULL;
470     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
471     GPIO_InitStruct.Alternate = GPIO_AF1_TIM1;
472     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
473
474   /* USER CODE BEGIN TIM1_MspPostInit 1 */
475
476   /* USER CODE END TIM1_MspPostInit 1 */
477   }
478   else if(timHandle->Instance==TIM2)
479   {
480   /* USER CODE BEGIN TIM2_MspPostInit 0 */
481
482   /* USER CODE END TIM2_MspPostInit 0 */
483
484     __HAL_RCC_GPIOB_CLK_ENABLE();
485     /**TIM2 GPIO Configuration
486     PB3     ------> TIM2_CH2
487     */
488     GPIO_InitStruct.Pin = GPIO_PIN_3;
489     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
490     GPIO_InitStruct.Pull = GPIO_NOPULL;
491     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
492     GPIO_InitStruct.Alternate = GPIO_AF1_TIM2;
493     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
494
495   /* USER CODE BEGIN TIM2_MspPostInit 1 */
496
```

```
497   /* USER CODE END TIM2_MspPostInit 1 */
498   }
499   else if(timHandle->Instance==TIM3)
500   {
501   /* USER CODE BEGIN TIM3_MspPostInit 0 */
502
503   /* USER CODE END TIM3_MspPostInit 0 */
504
505     __HAL_RCC_GPIOB_CLK_ENABLE();
506     /**TIM3 GPIO Configuration
507     PB5     ------> TIM3_CH2
508     */
509     GPIO_InitStruct.Pin = GPIO_PIN_5;
510     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
511     GPIO_InitStruct.Pull = GPIO_NOPULL;
512     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
513     GPIO_InitStruct.Alternate = GPIO_AF2_TIM3;
514     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
515
516   /* USER CODE BEGIN TIM3_MspPostInit 1 */
517
518   /* USER CODE END TIM3_MspPostInit 1 */
519   }
520   else if(timHandle->Instance==TIM4)
521   {
522   /* USER CODE BEGIN TIM4_MspPostInit 0 */
523
524   /* USER CODE END TIM4_MspPostInit 0 */
525
526     __HAL_RCC_GPIOB_CLK_ENABLE();
527     /**TIM4 GPIO Configuration
528     PB6     ------> TIM4_CH1
529     */
530     GPIO_InitStruct.Pin = GPIO_PIN_6;
531     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
532     GPIO_InitStruct.Pull = GPIO_NOPULL;
533     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
534     GPIO_InitStruct.Alternate = GPIO_AF2_TIM4;
535     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
536
537   /* USER CODE BEGIN TIM4_MspPostInit 1 */
538
539   /* USER CODE END TIM4_MspPostInit 1 */
540   }
541   else if(timHandle->Instance==TIM8)
542   {
543   /* USER CODE BEGIN TIM8_MspPostInit 0 */
544
545   /* USER CODE END TIM8_MspPostInit 0 */
546
547     __HAL_RCC_GPIOC_CLK_ENABLE();
548     /**TIM8 GPIO Configuration
549     PC7     ------> TIM8_CH2
550     */
551     GPIO_InitStruct.Pin = GPIO_PIN_7;
552     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
553     GPIO_InitStruct.Pull = GPIO_NOPULL;
554     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
555     GPIO_InitStruct.Alternate = GPIO_AF3_TIM8;
556     HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
557
558   /* USER CODE BEGIN TIM8_MspPostInit 1 */
```

```
559
560   /* USER CODE END TIM8_MspPostInit 1 */
561   }
562   else if(timHandle->Instance==TIM10)
563   {
564   /* USER CODE BEGIN TIM10_MspPostInit 0 */
565
566   /* USER CODE END TIM10_MspPostInit 0 */
567
568     __HAL_RCC_GPIOB_CLK_ENABLE();
569     /**TIM10 GPIO Configuration
570     PB8     ------> TIM10_CH1
571     */
572     GPIO_InitStruct.Pin = GPIO_PIN_8;
573     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
574     GPIO_InitStruct.Pull = GPIO_NOPULL;
575     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
576     GPIO_InitStruct.Alternate = GPIO_AF3_TIM10;
577     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
578
579   /* USER CODE BEGIN TIM10_MspPostInit 1 */
580
581   /* USER CODE END TIM10_MspPostInit 1 */
582   }
583   else if(timHandle->Instance==TIM14)
584   {
585   /* USER CODE BEGIN TIM14_MspPostInit 0 */
586
587   /* USER CODE END TIM14_MspPostInit 0 */
588
589     __HAL_RCC_GPIOA_CLK_ENABLE();
590     /**TIM14 GPIO Configuration
591     PA7     ------> TIM14_CH1
592     */
593     GPIO_InitStruct.Pin = GPIO_PIN_7;
594     GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
595     GPIO_InitStruct.Pull = GPIO_NOPULL;
596     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
597     GPIO_InitStruct.Alternate = GPIO_AF9_TIM14;
598     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
599
600   /* USER CODE BEGIN TIM14_MspPostInit 1 */
601
602   /* USER CODE END TIM14_MspPostInit 1 */
603   }
604
605 }
606
607 void HAL_TIM_Base_MspDeInit(TIM_HandleTypeDef* tim_baseHandle)
608 {
609
610   if(tim_baseHandle->Instance==TIM1)
611   {
612   /* USER CODE BEGIN TIM1_MspDeInit 0 */
613
614   /* USER CODE END TIM1_MspDeInit 0 */
615     /* Peripheral clock disable */
616     __HAL_RCC_TIM1_CLK_DISABLE();
617
618     /* TIM1 interrupt Deinit */
619   /* USER CODE BEGIN TIM1:TIM1_UP_TIM10_IRQn disable */
620     /**
```

```
621     * Uncomment the line below to disable the "TIM1_UP_TIM10_IRQn" interrupt
622     * Be aware, disabling shared interrupt may affect other IPs
623     */
624     /* HAL_NVIC_DisableIRQ(TIM1_UP_TIM10_IRQn); */
625   /* USER CODE END TIM1:TIM1_UP_TIM10_IRQn disable */
626
627   /* USER CODE BEGIN TIM1_MspDeInit 1 */
628
629   /* USER CODE END TIM1_MspDeInit 1 */
630   }
631   else if(tim_baseHandle->Instance==TIM2)
632   {
633   /* USER CODE BEGIN TIM2_MspDeInit 0 */
634
635   /* USER CODE END TIM2_MspDeInit 0 */
636     /* Peripheral clock disable */
637     __HAL_RCC_TIM2_CLK_DISABLE();
638
639     /* TIM2 interrupt Deinit */
640     HAL_NVIC_DisableIRQ(TIM2_IRQn);
641   /* USER CODE BEGIN TIM2_MspDeInit 1 */
642
643   /* USER CODE END TIM2_MspDeInit 1 */
644   }
645   else if(tim_baseHandle->Instance==TIM3)
646   {
647   /* USER CODE BEGIN TIM3_MspDeInit 0 */
648
649   /* USER CODE END TIM3_MspDeInit 0 */
650     /* Peripheral clock disable */
651     __HAL_RCC_TIM3_CLK_DISABLE();
652
653     /* TIM3 interrupt Deinit */
654     HAL_NVIC_DisableIRQ(TIM3_IRQn);
655   /* USER CODE BEGIN TIM3_MspDeInit 1 */
656
657   /* USER CODE END TIM3_MspDeInit 1 */
658   }
659   else if(tim_baseHandle->Instance==TIM4)
660   {
661   /* USER CODE BEGIN TIM4_MspDeInit 0 */
662
663   /* USER CODE END TIM4_MspDeInit 0 */
664     /* Peripheral clock disable */
665     __HAL_RCC_TIM4_CLK_DISABLE();
666
667     /* TIM4 interrupt Deinit */
668     HAL_NVIC_DisableIRQ(TIM4_IRQn);
669   /* USER CODE BEGIN TIM4_MspDeInit 1 */
670
671   /* USER CODE END TIM4_MspDeInit 1 */
672   }
673   else if(tim_baseHandle->Instance==TIM8)
674   {
675   /* USER CODE BEGIN TIM8_MspDeInit 0 */
676
677   /* USER CODE END TIM8_MspDeInit 0 */
678     /* Peripheral clock disable */
679     __HAL_RCC_TIM8_CLK_DISABLE();
680
681     /* TIM8 interrupt Deinit */
682     HAL_NVIC_DisableIRQ(TIM8_UP_TIM13_IRQn);
```

```
683   /* USER CODE BEGIN TIM8:TIM8_TRG_COM_TIM14_IRQn disable */
684     /**
685     * Uncomment the line below to disable the "TIM8_TRG_COM_TIM14_IRQn" interrupt
686     * Be aware, disabling shared interrupt may affect other IPs
687     */
688     /* HAL_NVIC_DisableIRQ(TIM8_TRG_COM_TIM14_IRQn); */
689   /* USER CODE END TIM8:TIM8_TRG_COM_TIM14_IRQn disable */
690
691   /* USER CODE BEGIN TIM8_MspDeInit 1 */
692
693   /* USER CODE END TIM8_MspDeInit 1 */
694   }
695   else if(tim_baseHandle->Instance==TIM10)
696   {
697   /* USER CODE BEGIN TIM10_MspDeInit 0 */
698
699   /* USER CODE END TIM10_MspDeInit 0 */
700     /* Peripheral clock disable */
701     __HAL_RCC_TIM10_CLK_DISABLE();
702
703     /* TIM10 interrupt Deinit */
704   /* USER CODE BEGIN TIM10:TIM1_UP_TIM10_IRQn disable */
705     /**
706     * Uncomment the line below to disable the "TIM1_UP_TIM10_IRQn" interrupt
707     * Be aware, disabling shared interrupt may affect other IPs
708     */
709     /* HAL_NVIC_DisableIRQ(TIM1_UP_TIM10_IRQn); */
710   /* USER CODE END TIM10:TIM1_UP_TIM10_IRQn disable */
711
712   /* USER CODE BEGIN TIM10_MspDeInit 1 */
713
714   /* USER CODE END TIM10_MspDeInit 1 */
715   }
716   else if(tim_baseHandle->Instance==TIM14)
717   {
718   /* USER CODE BEGIN TIM14_MspDeInit 0 */
719
720   /* USER CODE END TIM14_MspDeInit 0 */
721     /* Peripheral clock disable */
722     __HAL_RCC_TIM14_CLK_DISABLE();
723
724     /* TIM14 interrupt Deinit */
725   /* USER CODE BEGIN TIM14:TIM8_TRG_COM_TIM14_IRQn disable */
726     /**
727     * Uncomment the line below to disable the "TIM8_TRG_COM_TIM14_IRQn" interrupt
728     * Be aware, disabling shared interrupt may affect other IPs
729     */
730     /* HAL_NVIC_DisableIRQ(TIM8_TRG_COM_TIM14_IRQn); */
731   /* USER CODE END TIM14:TIM8_TRG_COM_TIM14_IRQn disable */
732
733   /* USER CODE BEGIN TIM14_MspDeInit 1 */
734
735   /* USER CODE END TIM14_MspDeInit 1 */
736   }
737 }
738
739 /* USER CODE BEGIN 1 */
740
741 /* USER CODE END 1 */
742
743 /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
744
```