```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"
#include "fatfs.h"

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

#include "fram_func.h"
#include "sd_card_func.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
//#define MEMORY_SIZE   524288
/* USER CODE END PD */

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

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
 DAC_HandleTypeDef hdac1;

I2C_HandleTypeDef hi2c1;
I2C_HandleTypeDef hi2c2;

SD_HandleTypeDef hsd1;

SPI_HandleTypeDef hspi1;
SPI_HandleTypeDef hspi2;
DMA_HandleTypeDef hdma_spi1_rx;
DMA_HandleTypeDef hdma_spi1_tx;

TIM_HandleTypeDef htim16;

UART_HandleTypeDef huart1;

HCD_HandleTypeDef hhcd_USB_OTG_FS;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DAC1_Init(void);
static void MX_SDMMC1_SD_Init(void);
static void MX_SPI1_Init(void);
```

```c
static void MX_USART1_UART_Init(void);
static void MX_I2C1_Init(void);
static void MX_I2C2_Init(void);
static void MX_SPI2_Init(void);
static void MX_DMA_Init(void);
static void MX_USB_OTG_FS_HCD_Init(void);
static void MX_TIM16_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
const uint32_t TAB_SIZE = (MEMORY_SIZE/16);
enum color_Led LED_STATE = OFF;
bool reset = true;
bool start = true;
bool acqu = true;
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
    //FRESULT res; /* FatFs function common result code */
    TCHAR* fileName0 ;
    TCHAR* fileName1 ;
    TCHAR* fileName2 ;
    TCHAR* fileName3 ;
    TCHAR* file_Init = "INIT.txt";
    uint8_t  TX_Data_ADC[2];
    uint8_t  TX_FRAM_TEST_R[10] ;
    uint16_t DATA_SD;
    uint16_t DATA_ind [TAB_SIZE/2];
    uint8_t pretrigValue = 100;


    //uint8_t  RX_DATA [524288];   -> TOO BIG

  /* 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 */
  MX_GPIO_Init();
  MX_DAC1_Init();
  MX_SDMMC1_SD_Init();
  MX_SPI1_Init();
  MX_USART1_UART_Init();
  MX_I2C1_Init();
  MX_I2C2_Init();
  MX_SPI2_Init();
  MX_DMA_Init();
  MX_USB_OTG_FS_HCD_Init();
  MX_FATFS_Init();
  MX_TIM16_Init();
```

```
147     /* USER CODE BEGIN 2 */
148
149     /* USER CODE END 2 */
150
151     /* Infinite loop */
152     /* USER CODE BEGIN WHILE */
153      uint8_t start_Add[4] ;
154      uint32_t start_Add_tot = 0 ;
155      uint32_t Add_= 0;
156      uint8_t TX_32 = 0;
157      uint16_t boucle = 0;
158      uint8_t wtext_0 [100] ;
159      uint8_t wtext_1 [100] ;
160      uint8_t wtext_2 [100] ;
161      uint8_t wtext_3 [100] ;
162      uint8_t* receive;
163      uint8_t chann;
164      double* dataInit_;
165      double triggDacValue;
166      PIN_reset();
167      //FRAM_device(hspi1);
168     while (1)
169     {
170       acqu = true;
171       TX_32 = 0;
172       start = true;
173       HAL_SuspendTick();
174       __WFI();
175
176       //-----------------------------------------------------RESET
          FPGA----------------------------------------
177       HAL_GPIO_WritePin(FPGA_RESET_GPIO_Port, FPGA_RESET_Pin, GPIO_PIN_RESET);
178       HAL_Delay(100);
179       HAL_GPIO_WritePin(FPGA_RESET_GPIO_Port, FPGA_RESET_Pin, GPIO_PIN_SET);
180       //-----------------------------------------------------RESET
          M--------------------------------------
181       PIN_reset();
182
          //-------------------------------------------------------------INIT-----------------
          -------------------------
183       //-------------------init file sd ---------------------------------------
184       sprintf((char*)wtext_0, "FRAM0_%d.bin",boucle);
185       fileName0 = (TCHAR*)wtext_0;
186       SD_create_file(SDFile, fileName0);
187       sprintf((char*)wtext_1, "FRAM1_%d.bin",boucle);
188       fileName1 = (TCHAR*)wtext_1;
189       SD_create_file(SDFile, fileName1);
190       sprintf((char*)wtext_2, "FRAM2_%d.bin",boucle);
191       fileName2 = (TCHAR*)wtext_2;
192       SD_create_file(SDFile, fileName2);
193       sprintf((char*)wtext_3, "FRAM3_%d.bin",boucle);
194       fileName3 = (TCHAR*)wtext_3;
195       SD_create_file(SDFile, fileName3);
196
197       //-------------------init
          value-------------------------------------------------------------------------
198       dataInit_ = initValue(SDFile, file_Init);
199
200       chann = (uint8_t)dataInit_[0];
201       if(chann >4){
202           chann = 0;
203       }
204       triggDacValue = dataInit_[1]*(0.02)+1;
205       if(triggDacValue <1 || triggDacValue>3.3){
206           triggDacValue = 1;
207       }
208       pretrigValue = (uint8_t)dataInit_[2];
209       if(pretrigValue >100){
210           pretrigValue = 100;
211       }
212       //-----------------------------------------------LED
          CONTROL----------------------------------------
213       if(LED_STATE != RED){
```

```
214         LED_on(BLUE);
215       }
216       else{
217           while(1){
218               //reset the systeme
219           }
220       }
221
222       //-------------------init trigg dc value--------------------------------
223       uint32_t triggDac = (uint32_t)((triggDacValue/VREF_VOLTAGE)*DAC_MAX_VALUE);
224       HAL_DAC_SetValue(&hdac1, DAC1_CHANNEL_2, DAC_ALIGN_12B_R, triggDac);
225       HAL_DAC_Start(&hdac1, DAC1_CHANNEL_2);
226       HAL_DAC_SetValue(&hdac1, DAC1_CHANNEL_1, DAC_ALIGN_12B_R, triggDac);
227       HAL_DAC_Start(&hdac1, DAC1_CHANNEL_1);
228
229       //-------------------init pretrigg value--------------------------------
230       HAL_SPI_Transmit(&hspi1, (uint8_t*)&pretrigValue,1, 100);
231       HAL_GPIO_WritePin(FPGA_PRETRIG_GPIO_Port, FPGA_PRETRIG_Pin, GPIO_PIN_SET);
232       HAL_SPI_Transmit(&hspi1, (uint8_t*)&pretrigValue,1, 100);
233       HAL_GPIO_WritePin(FPGA_PRETRIG_GPIO_Port, FPGA_PRETRIG_Pin, GPIO_PIN_RESET);
234
235       //-------------------init trigg channel ---------------------------------
236       setTriggChannel(chann);
237       //-----------------------------------------------------RESET FRAM
          REG--------------------------------------
238       FRAM_reset_reg(hspi1);
239       //-----------------------------------------------------WRITE FRAM
          REG--------------------------------------
240       FRAM_write_reg(hspi1);
241       //-----------------------------------------------------ADC WRITE
          REG--------------------------------------
242       TX_Data_ADC[0] = 0xA2;
243       TX_Data_ADC[1] = 0x80;
244       HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_SET);
245       HAL_SPI_Transmit(&hspi1, (uint8_t*)TX_Data_ADC,2, 100);
246       HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_RESET);
247       HAL_Delay(100);
248
249       //-----------------------------------------------------WRITE
          FRAM-------------------------------------------
250       PIN_reset();
251       TX_FRAM_TEST_R[0] = 2;
252       for(int i =1; i<5;i++){
253         TX_FRAM_TEST_R[i] = 0;
254       }
255       TX_FRAM_TEST_R[3] = 0;
256       TX_FRAM_TEST_R[4] = 0;
257       TX_FRAM_TEST_R[5] = 56;
258
259       HAL_GPIO_WritePin(SELECTOR_M3_GPIO_Port, SELECTOR_M3_Pin, GPIO_PIN_SET);
260       HAL_SPI_Transmit(&hspi1, (uint8_t*)TX_FRAM_TEST_R,6, 100);
261       HAL_Delay(100);
262
263       //-----------------------------------------------------ADC TO
          FRAM-------------------------------------------
264       HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_SET);
265       //-----------------------------------------------------SLEEP
          MODE-------------------------------------------
266       //HAL_PWR_EnterSLEEPMode(PWR_LOWPOWERREGULATOR_ON, PWR_SLEEPENTRY_WFI);
267       while(acqu){
268           HAL_SuspendTick();
269           __WFI();
270       }
271       //HAL_PWR_ENTER
272       PIN_reset();
273       while(TX_32 <16) {
274             if(TX_32 == 0){
275                 PIN_reset();
276                 HAL_GPIO_WritePin(SELECTOR_M0_GPIO_Port, SELECTOR_M0_Pin, GPIO_PIN_SET);
277                 HAL_GPIO_WritePin(SELECTOR_M1_GPIO_Port, SELECTOR_M1_Pin, GPIO_PIN_SET);
278                 HAL_GPIO_WritePin(SELECTOR_M3_GPIO_Port, SELECTOR_M3_Pin, GPIO_PIN_SET);
279                 HAL_SPI_Receive(&hspi1, (uint8_t*)start_Add,4, 100);
280                 HAL_Delay(100);
```

```
281                 start_Add_tot = (((uint32_t)start_Add[0])<<24)+(((uint32_t)start_Add[1
                    ])<<16)
282                                         +(((uint32_t)start_Add[2])<<8)+((uint32_t)
                                            start_Add[3]);
283                 PIN_reset();
284                 start_Add_tot = start_Add_tot>>12;
285                 //start_Add_tot = 0;
286                 HAL_TIM_Base_Start_IT(&htim16);
287             }
288             //-------------------------------------------------READ FRAM
                0-----------------------------------------
289             for(uint32_t i = 0; i<(TAB_SIZE/2) ; i++){
290                 Add_ = i*2+TAB_SIZE*TX_32+(MEMORY_SIZE);
291                 if(Add_> MEMORY_SIZE){
292                     Add_ = Add_ - MEMORY_SIZE;
293                 }
294                 //receive = FRAM_read(FRAM_0,i*2+TAB_SIZE*TX_32, hspi1, 2);
295                receive = FRAM_read(FRAM_0, Add_, hspi1, 2);
296                DATA_SD = (((uint16_t)receive[0])<<8)+(uint16_t)(receive[1]);
297                // DATA_SD = (((uint16_t)RX_FAST[i*2])<<8)+(uint16_t)(RX_FAST[i*2+1]);
298                 DATA_ind[i] = DATA_SD;
299             }
300             SD_write_data(SDFile, fileName0, DATA_ind);
301             //-------------------------------------------------READ FRAM
                1-----------------------------------------
302             for(uint32_t i = 0; i<(TAB_SIZE/2) ; i++){
303                 Add_ = i*2+TAB_SIZE*TX_32+(MEMORY_SIZE);
304                 if(Add_> MEMORY_SIZE){
305                     Add_ = Add_ - MEMORY_SIZE;
306                 }
307                 receive = FRAM_read(FRAM_1, Add_, hspi1, 2);
308                 DATA_SD = (((uint16_t)receive[0])<<8)+(uint16_t)(receive[1]);
309                 //DATA_SD = (short)DATA_SD;
310                 DATA_ind[i] = DATA_SD;
311             }
312             SD_write_data(SDFile, fileName1, DATA_ind);
313             //-------------------------------------------------READ FRAM
                2-----------------------------------------
314             for(uint32_t i = 0; i<TAB_SIZE/2 ; i++){
315                 Add_ = i*2+TAB_SIZE*TX_32+(MEMORY_SIZE-start_Add_tot);
316                 if(Add_> MEMORY_SIZE){
317                     Add_ = Add_ - MEMORY_SIZE;
318                 }
319                 receive = FRAM_read(FRAM_2,Add_, hspi1, 2);
320                 DATA_SD = (((uint16_t)receive[0])<<8)+(uint16_t)(receive[1]);
321                 //DATA_SD = (short)DATA_SD;
322                 DATA_ind[i] = DATA_SD;
323             }
324             SD_write_data(SDFile, fileName2, DATA_ind);
325             //-------------------------------------------------READ FRAM
                3-----------------------------------------
326             for(uint32_t i = 0; i<TAB_SIZE/2 ; i++){
327                 Add_ = i*2+TAB_SIZE*TX_32+(MEMORY_SIZE-start_Add_tot);
328                 if(Add_> MEMORY_SIZE){
329                     Add_ = Add_ - MEMORY_SIZE;
330                 }
331                 receive = FRAM_read(FRAM_3,Add_, hspi1, 2);
332                 DATA_SD = (((uint16_t)receive[0])<<8)+(uint16_t)(receive[1]);
333                 //DATA_SD = (short)DATA_SD;
334                 DATA_ind[i] = DATA_SD;
335             }
336             SD_write_data(SDFile, fileName3, DATA_ind);
337             TX_32++;
338             //receive = FRAM_read(FRAM_3, TX_32, hspi1, 2);
339             //DATA_SD = (((uint16_t)receive[0])<<8)+(uint16_t)(receive[1]);
340             //DATA_SD = (short)DATA_SD;
341             //SD_write_data(SDFile, fileName3, DATA_SD);
342         }
343         if(TX_32==16){
344             HAL_TIM_Base_Stop(&htim16);
345             if(LED_STATE != RED){
346                 LED_off();
347                 LED_on(GREEN); //6
```

```
348             }
349             else{
350                 LED_off();
351                 LED_on(RED);
352             }
353         }
354         boucle++;
355       /* USER CODE END WHILE */
356 
357       /* USER CODE BEGIN 3 */
358     }
359     /* USER CODE END 3 */
360   }
361 
362   /**
363     * @brief System Clock Configuration
364     * @retval None
365     */
366   void SystemClock_Config(void)
367   {
368     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
369     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
370 
371     /** Configure the main internal regulator output voltage
372     */
373     if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
374     {
375       Error_Handler();
376     }
377 
378     /** Initializes the RCC Oscillators according to the specified parameters
379     * in the RCC_OscInitTypeDef structure.
380     */
381     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI48|RCC_OSCILLATORTYPE_MSI;
382     RCC_OscInitStruct.HSI48State = RCC_HSI48_ON;
383     RCC_OscInitStruct.MSIState = RCC_MSI_ON;
384     RCC_OscInitStruct.MSICalibrationValue = 0;
385     RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
386     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
387     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_MSI;
388     RCC_OscInitStruct.PLL.PLLM = 1;
389     RCC_OscInitStruct.PLL.PLLN = 32;
390     RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
391     RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
392     RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
393     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
394     {
395       Error_Handler();
396     }
397 
398     /** Initializes the CPU, AHB and APB buses clocks
399     */
400     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
401                                   |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
402     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
403     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
404     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
405     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
406 
407     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
408     {
409       Error_Handler();
410     }
411   }
412 
413   /**
414     * @brief DAC1 Initialization Function
415     * @param None
416     * @retval None
417     */
418   static void MX_DAC1_Init(void)
419   {
420 
```

```
421     /* USER CODE BEGIN DAC1_Init 0 */
422
423     /* USER CODE END DAC1_Init 0 */
424
425     DAC_ChannelConfTypeDef sConfig = {0};
426
427     /* USER CODE BEGIN DAC1_Init 1 */
428
429     /* USER CODE END DAC1_Init 1 */
430
431     /** DAC Initialization
432     */
433     hdac1.Instance = DAC1;
434     if (HAL_DAC_Init(&hdac1) != HAL_OK)
435     {
436       Error_Handler();
437     }
438
439     /** DAC channel OUT1 config
440     */
441     sConfig.DAC_SampleAndHold = DAC_SAMPLEANDHOLD_DISABLE;
442     sConfig.DAC_Trigger = DAC_TRIGGER_NONE;
443     sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;
444     sConfig.DAC_ConnectOnChipPeripheral = DAC_CHIPCONNECT_DISABLE;
445     sConfig.DAC_UserTrimming = DAC_TRIMMING_FACTORY;
446     if (HAL_DAC_ConfigChannel(&hdac1, &sConfig, DAC_CHANNEL_1) != HAL_OK)
447     {
448       Error_Handler();
449     }
450
451     /** DAC channel OUT2 config
452     */
453     if (HAL_DAC_ConfigChannel(&hdac1, &sConfig, DAC_CHANNEL_2) != HAL_OK)
454     {
455       Error_Handler();
456     }
457     /* USER CODE BEGIN DAC1_Init 2 */
458
459     /* USER CODE END DAC1_Init 2 */
460
461   }
462
463   /**
464     * @brief I2C1 Initialization Function
465     * @param None
466     * @retval None
467     */
468   static void MX_I2C1_Init(void)
469   {
470
471     /* USER CODE BEGIN I2C1_Init 0 */
472
473     /* USER CODE END I2C1_Init 0 */
474
475     /* USER CODE BEGIN I2C1_Init 1 */
476
477     /* USER CODE END I2C1_Init 1 */
478     hi2c1.Instance = I2C1;
479     hi2c1.Init.Timing = 0x10707DBC;
480     hi2c1.Init.OwnAddress1 = 0;
481     hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
482     hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
483     hi2c1.Init.OwnAddress2 = 0;
484     hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
485     hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
486     hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
487     if (HAL_I2C_Init(&hi2c1) != HAL_OK)
488     {
489       Error_Handler();
490     }
491
492     /** Configure Analogue filter
493     */
```

```
494     if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
495     {
496       Error_Handler();
497     }
498 
499     /** Configure Digital filter
500     */
501     if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
502     {
503       Error_Handler();
504     }
505     /* USER CODE BEGIN I2C1_Init 2 */
506 
507     /* USER CODE END I2C1_Init 2 */
508 
509   }
510 
511   /**
512     * @brief I2C2 Initialization Function
513     * @param None
514     * @retval None
515     */
516   static void MX_I2C2_Init(void)
517   {
518 
519     /* USER CODE BEGIN I2C2_Init 0 */
520 
521     /* USER CODE END I2C2_Init 0 */
522 
523     /* USER CODE BEGIN I2C2_Init 1 */
524 
525     /* USER CODE END I2C2_Init 1 */
526     hi2c2.Instance = I2C2;
527     hi2c2.Init.Timing = 0x10707DBC;
528     hi2c2.Init.OwnAddress1 = 0;
529     hi2c2.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
530     hi2c2.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
531     hi2c2.Init.OwnAddress2 = 0;
532     hi2c2.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
533     hi2c2.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
534     hi2c2.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
535     if (HAL_I2C_Init(&hi2c2) != HAL_OK)
536     {
537       Error_Handler();
538     }
539 
540     /** Configure Analogue filter
541     */
542     if (HAL_I2CEx_ConfigAnalogFilter(&hi2c2, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
543     {
544       Error_Handler();
545     }
546 
547     /** Configure Digital filter
548     */
549     if (HAL_I2CEx_ConfigDigitalFilter(&hi2c2, 0) != HAL_OK)
550     {
551       Error_Handler();
552     }
553     /* USER CODE BEGIN I2C2_Init 2 */
554 
555     /* USER CODE END I2C2_Init 2 */
556 
557   }
558 
559   /**
560     * @brief SDMMC1 Initialization Function
561     * @param None
562     * @retval None
563     */
564   static void MX_SDMMC1_SD_Init(void)
565   {
566 
```

```
567     /* USER CODE BEGIN SDMMC1_Init 0 */
568
569     /* USER CODE END SDMMC1_Init 0 */
570
571     /* USER CODE BEGIN SDMMC1_Init 1 */
572
573     /* USER CODE END SDMMC1_Init 1 */
574     hsd1.Instance = SDMMC1;
575     hsd1.Init.ClockEdge = SDMMC_CLOCK_EDGE_RISING;
576     hsd1.Init.ClockBypass = SDMMC_CLOCK_BYPASS_DISABLE;
577     hsd1.Init.ClockPowerSave = SDMMC_CLOCK_POWER_SAVE_DISABLE;
578     hsd1.Init.BusWide = SDMMC_BUS_WIDE_1B;
579     hsd1.Init.HardwareFlowControl = SDMMC_HARDWARE_FLOW_CONTROL_DISABLE;
580     hsd1.Init.ClockDiv = 4;
581     /* USER CODE BEGIN SDMMC1_Init 2 */
582
583     /* USER CODE END SDMMC1_Init 2 */
584
585   }
586
587   /**
588     * @brief SPI1 Initialization Function
589     * @param None
590     * @retval None
591     */
592   static void MX_SPI1_Init(void)
593   {
594
595     /* USER CODE BEGIN SPI1_Init 0 */
596
597     /* USER CODE END SPI1_Init 0 */
598
599     /* USER CODE BEGIN SPI1_Init 1 */
600
601     /* USER CODE END SPI1_Init 1 */
602     /* SPI1 parameter configuration*/
603     hspi1.Instance = SPI1;
604     hspi1.Init.Mode = SPI_MODE_MASTER;
605     hspi1.Init.Direction = SPI_DIRECTION_2LINES;
606     hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
607     hspi1.Init.CLKPolarity = SPI_POLARITY_HIGH;
608     hspi1.Init.CLKPhase = SPI_PHASE_2EDGE;
609     hspi1.Init.NSS = SPI_NSS_SOFT;
610     hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;
611     hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
612     hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
613     hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
614     hspi1.Init.CRCPolynomial = 7;
615     hspi1.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
616     hspi1.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
617     if (HAL_SPI_Init(&hspi1) != HAL_OK)
618     {
619       Error_Handler();
620     }
621     /* USER CODE BEGIN SPI1_Init 2 */
622
623     /* USER CODE END SPI1_Init 2 */
624
625   }
626
627   /**
628     * @brief SPI2 Initialization Function
629     * @param None
630     * @retval None
631     */
632   static void MX_SPI2_Init(void)
633   {
634
635     /* USER CODE BEGIN SPI2_Init 0 */
636
637     /* USER CODE END SPI2_Init 0 */
638
639     /* USER CODE BEGIN SPI2_Init 1 */
```

```
640
641     /* USER CODE END SPI2_Init 1 */
642     /* SPI2 parameter configuration*/
643     hspi2.Instance = SPI2;
644     hspi2.Init.Mode = SPI_MODE_MASTER;
645     hspi2.Init.Direction = SPI_DIRECTION_1LINE;
646     hspi2.Init.DataSize = SPI_DATASIZE_4BIT;
647     hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;
648     hspi2.Init.CLKPhase = SPI_PHASE_1EDGE;
649     hspi2.Init.NSS = SPI_NSS_SOFT;
650     hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
651     hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
652     hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
653     hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
654     hspi2.Init.CRCPolynomial = 7;
655     hspi2.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
656     hspi2.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
657     if (HAL_SPI_Init(&hspi2) != HAL_OK)
658     {
659       Error_Handler();
660     }
661     /* USER CODE BEGIN SPI2_Init 2 */
662
663     /* USER CODE END SPI2_Init 2 */
664
665   }
666
667   /**
668     * @brief TIM16 Initialization Function
669     * @param None
670     * @retval None
671     */
672   static void MX_TIM16_Init(void)
673   {
674
675     /* USER CODE BEGIN TIM16_Init 0 */
676
677     /* USER CODE END TIM16_Init 0 */
678
679     /* USER CODE BEGIN TIM16_Init 1 */
680
681     /* USER CODE END TIM16_Init 1 */
682     htim16.Instance = TIM16;
683     htim16.Init.Prescaler = 6400-1;
684     htim16.Init.CounterMode = TIM_COUNTERMODE_UP;
685     htim16.Init.Period = 1000;
686     htim16.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
687     htim16.Init.RepetitionCounter = 0;
688     htim16.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
689     if (HAL_TIM_Base_Init(&htim16) != HAL_OK)
690     {
691       Error_Handler();
692     }
693     /* USER CODE BEGIN TIM16_Init 2 */
694
695     /* USER CODE END TIM16_Init 2 */
696
697   }
698
699   /**
700     * @brief USART1 Initialization Function
701     * @param None
702     * @retval None
703     */
704   static void MX_USART1_UART_Init(void)
705   {
706
707     /* USER CODE BEGIN USART1_Init 0 */
708
709     /* USER CODE END USART1_Init 0 */
710
711     /* USER CODE BEGIN USART1_Init 1 */
712
```

```
713     /* USER CODE END USART1_Init 1 */
714     huart1.Instance = USART1;
715     huart1.Init.BaudRate = 115200;
716     huart1.Init.WordLength = UART_WORDLENGTH_8B;
717     huart1.Init.StopBits = UART_STOPBITS_1;
718     huart1.Init.Parity = UART_PARITY_NONE;
719     huart1.Init.Mode = UART_MODE_TX_RX;
720     huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
721     huart1.Init.OverSampling = UART_OVERSAMPLING_16;
722     huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
723     huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
724     if (HAL_UART_Init(&huart1) != HAL_OK)
725     {
726       Error_Handler();
727     }
728     /* USER CODE BEGIN USART1_Init 2 */
729
730     /* USER CODE END USART1_Init 2 */
731
732   }
733
734   /**
735     * @brief USB_OTG_FS Initialization Function
736     * @param None
737     * @retval None
738     */
739   static void MX_USB_OTG_FS_HCD_Init(void)
740   {
741
742     /* USER CODE BEGIN USB_OTG_FS_Init 0 */
743
744     /* USER CODE END USB_OTG_FS_Init 0 */
745
746     /* USER CODE BEGIN USB_OTG_FS_Init 1 */
747
748     /* USER CODE END USB_OTG_FS_Init 1 */
749     hhcd_USB_OTG_FS.Instance = USB_OTG_FS;
750     hhcd_USB_OTG_FS.Init.Host_channels = 12;
751     hhcd_USB_OTG_FS.Init.speed = HCD_SPEED_FULL;
752     hhcd_USB_OTG_FS.Init.dma_enable = DISABLE;
753     hhcd_USB_OTG_FS.Init.phy_itface = HCD_PHY_EMBEDDED;
754     hhcd_USB_OTG_FS.Init.Sof_enable = DISABLE;
755     if (HAL_HCD_Init(&hhcd_USB_OTG_FS) != HAL_OK)
756     {
757       Error_Handler();
758     }
759     /* USER CODE BEGIN USB_OTG_FS_Init 2 */
760
761     /* USER CODE END USB_OTG_FS_Init 2 */
762
763   }
764
765   /**
766     * Enable DMA controller clock
767     */
768   static void MX_DMA_Init(void)
769   {
770
771     /* DMA controller clock enable */
772     __HAL_RCC_DMA1_CLK_ENABLE();
773
774     /* DMA interrupt init */
775     /* DMA1_Channel2_IRQn interrupt configuration */
776     HAL_NVIC_SetPriority(DMA1_Channel2_IRQn, 0, 0);
777     HAL_NVIC_EnableIRQ(DMA1_Channel2_IRQn);
778     /* DMA1_Channel3_IRQn interrupt configuration */
779     HAL_NVIC_SetPriority(DMA1_Channel3_IRQn, 0, 0);
780     HAL_NVIC_EnableIRQ(DMA1_Channel3_IRQn);
781
782   }
783
784   /**
785     * @brief GPIO Initialization Function
```

```
786     * @param None
787     * @retval None
788     */
789   static void MX_GPIO_Init(void)
790   {
791     GPIO_InitTypeDef GPIO_InitStruct = {0};
792
793     /* GPIO Ports Clock Enable */
794     __HAL_RCC_GPIOC_CLK_ENABLE();
795     __HAL_RCC_GPIOH_CLK_ENABLE();
796     __HAL_RCC_GPIOA_CLK_ENABLE();
797     __HAL_RCC_GPIOB_CLK_ENABLE();
798     __HAL_RCC_GPIOD_CLK_ENABLE();
799
800     /*Configure GPIO pin Output Level */
801     HAL_GPIO_WritePin(GPIOC, FPGA_RESET_Pin|FPGA_PRETRIG_Pin, GPIO_PIN_RESET);
802
803     /*Configure GPIO pin Output Level */
804     HAL_GPIO_WritePin(GPIOA, SELECTOR_M0_Pin|SELECTOR_M1_Pin|SELECTOR_M2_Pin|
      SELECTOR_M3_Pin, GPIO_PIN_RESET);
805
806     /*Configure GPIO pin Output Level */
807     HAL_GPIO_WritePin(GPIOB, UI_LED_G_Pin|UI_LED_R_Pin|UI_LED_B_Pin|TRIG_SRC0_Pin
808                             |TRIG_SRC1_Pin, GPIO_PIN_RESET);
809
810     /*Configure GPIO pin : WAKE_UP_Pin */
811     GPIO_InitStruct.Pin = WAKE_UP_Pin;
812     GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
813     GPIO_InitStruct.Pull = GPIO_PULLUP;
814     HAL_GPIO_Init(WAKE_UP_GPIO_Port, &GPIO_InitStruct);
815
816     /*Configure GPIO pins : FPGA_RESET_Pin FPGA_PRETRIG_Pin */
817     GPIO_InitStruct.Pin = FPGA_RESET_Pin|FPGA_PRETRIG_Pin;
818     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
819     GPIO_InitStruct.Pull = GPIO_NOPULL;
820     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
821     HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
822
823     /*Configure GPIO pins : SELECTOR_M0_Pin SELECTOR_M1_Pin SELECTOR_M2_Pin
      SELECTOR_M3_Pin */
824     GPIO_InitStruct.Pin = SELECTOR_M0_Pin|SELECTOR_M1_Pin|SELECTOR_M2_Pin|
      SELECTOR_M3_Pin;
825     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
826     GPIO_InitStruct.Pull = GPIO_NOPULL;
827     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
828     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
829
830     /*Configure GPIO pins : UI_LED_G_Pin UI_LED_R_Pin UI_LED_B_Pin TRIG_SRC0_Pin
831                              TRIG_SRC1_Pin */
832     GPIO_InitStruct.Pin = UI_LED_G_Pin|UI_LED_R_Pin|UI_LED_B_Pin|TRIG_SRC0_Pin
833                             |TRIG_SRC1_Pin;
834     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
835     GPIO_InitStruct.Pull = GPIO_NOPULL;
836     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
837     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
838
839     /*Configure GPIO pin : FPGA_DONE_Pin */
840     GPIO_InitStruct.Pin = FPGA_DONE_Pin;
841     GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING_FALLING;
842     GPIO_InitStruct.Pull = GPIO_NOPULL;
843     HAL_GPIO_Init(FPGA_DONE_GPIO_Port, &GPIO_InitStruct);
844
845     /*Configure GPIO pin : SD_DETECT_INT_Pin */
846     GPIO_InitStruct.Pin = SD_DETECT_INT_Pin;
847     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
848     GPIO_InitStruct.Pull = GPIO_NOPULL;
849     HAL_GPIO_Init(SD_DETECT_INT_GPIO_Port, &GPIO_InitStruct);
850
851     /*Configure GPIO pin : USB_VBUS_INT_Pin */
852     GPIO_InitStruct.Pin = USB_VBUS_INT_Pin;
853     GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
854     GPIO_InitStruct.Pull = GPIO_NOPULL;
855     HAL_GPIO_Init(USB_VBUS_INT_GPIO_Port, &GPIO_InitStruct);
```

```
856
857     /*Configure GPIO pin : ARM_BUTTON_Pin */
858     GPIO_InitStruct.Pin = ARM_BUTTON_Pin;
859     GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
860     GPIO_InitStruct.Pull = GPIO_PULLUP;
861     HAL_GPIO_Init(ARM_BUTTON_GPIO_Port, &GPIO_InitStruct);
862
863     /* EXTI interrupt init*/
864     HAL_NVIC_SetPriority(EXTI4_IRQn, 5, 0);
865     HAL_NVIC_EnableIRQ(EXTI4_IRQn);
866
867     HAL_NVIC_SetPriority(EXTI15_10_IRQn, 5, 0);
868     HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
869
870   }
871
872   /* USER CODE BEGIN 4 */
873   #ifdef __GNUC__
874     /* With GCC, small printf (option LD Linker->Libraries->Small printf
875        set to 'Yes') calls __io_putchar() */
876   int __io_putchar(int ch)
877   #else
878   int fputc(int ch, FILE *f)
879   #endif /* __GNUC__ */
880   {
881     /* Place your implementation of fputc here */
882     /* e.g. write a character to the UART3 and Loop until the end of transmission */
883     HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, HAL_MAX_DELAY);
884     return ch;
885   }
886   void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim){
887       if(htim== &htim16){
888           HAL_GPIO_TogglePin(UI_LED_B_GPIO_Port, UI_LED_B_Pin);
889       }
890   }
891   /* USER CODE END 4 */
892
893   /**
894     * @brief  This function is executed in case of error occurrence.
895     * @retval None
896     */
897   void Error_Handler(void)
898   {
899     /* USER CODE BEGIN Error_Handler_Debug */
900     /* User can add his own implementation to report the HAL error return state */
901     __disable_irq();
902     while (1)
903     {
904     }
905     /* USER CODE END Error_Handler_Debug */
906   }
907
908   #ifdef  USE_FULL_ASSERT
909   /**
910     * @brief  Reports the name of the source file and the source line number
911     *         where the assert_param error has occurred.
912     * @param  file: pointer to the source file name
913     * @param  line: assert_param error line source number
914     * @retval None
915     */
916   void assert_failed(uint8_t *file, uint32_t line)
917   {
918     /* USER CODE BEGIN 6 */
919     /* User can add his own implementation to report the file name and line number,
920        ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
921     /* USER CODE END 6 */
922   }
923   #endif /* USE_FULL_ASSERT */
924
```