Общество с ограниченной ответственностью

«Модульные Системы Торнадо»

Программное обеспечение

Программа «Gridex-microcode» для контроллера IPC GRIDEX II Текст программы (исходный код)

АБНС.57012-01 12 01

Технический директор С.А. Кулагин

2018

Согласовано

Взам. инв. №

Подп. и дата

Инв. № подл.

```
//===================================================================
// main.h
//===================================================================

#ifndef __MAIN_H
#define __MAIN_H

#define GP_LED_G_Pin GPIO_PIN_13
#define GP_LED_G_GPIO_Port GPIOC
#define GP_LED_R_Pin GPIO_PIN_14
#define GP_LED_R_GPIO_Port GPIOC
#define RESET_BTN_Pin GPIO_PIN_15
#define RESET_BTN_GPIO_Port GPIOC
#define PWR_GOOD_Pin GPIO_PIN_0
#define PWR_GOOD_GPIO_Port GPIOA
#define RESET_PCIE_Pin GPIO_PIN_1
#define RESET_PCIE_GPIO_Port GPIOA
#define SYS_S5_Pin GPIO_PIN_9
#define SYS_S5_GPIO_Port GPIOA
#define SYS_STAT_Pin GPIO_PIN_10
#define SYS_STAT_GPIO_Port GPIOA
#define HDD_LED_Pin GPIO_PIN_11
#define HDD_LED_GPIO_Port GPIOA
#define SYS_S3_Pin GPIO_PIN_12
#define SYS_S3_GPIO_Port GPIOA
#define BAT_CONTROL_Pin GPIO_PIN_6
#define BAT_CONTROL_GPIO_Port GPIOF
#define CONFIG_Pin GPIO_PIN_7
#define CONFIG_GPIO_Port GPIOF
#define HDD_LED_R_Pin GPIO_PIN_15
#define HDD_LED_R_GPIO_Port GPIOA
#define HDD_LED_G_Pin GPIO_PIN_3
#define HDD_LED_G_GPIO_Port GPIOB
#define PWR_LED_R_Pin GPIO_PIN_4
#define PWR_LED_R_GPIO_Port GPIOB
#define PWR_LED_G_Pin GPIO_PIN_5
#define PWR_LED_G_GPIO_Port GPIOB
#define WDT_Pin GPIO_PIN_8
#define WDT_GPIO_Port GPIOB
#define PWR_OK_Pin GPIO_PIN_9
#define PWR_OK_GPIO_Port GPIOB

void _Error_Handler(char *, int);

#define Error_Handler() _Error_Handler(__FILE__, __LINE__)

#endif /* __MAIN_H */

//===================================================================
//===================================================================
```

```
//===================================================================
// CBCE.h
//===================================================================

#include "CBCE_DIO.h"


#define RST_BTN_ON  RESET_BTN_GPIO_Port->BSRR = (uint32_t)RESET_BTN_Pin << 16
#define RST_BTN_OFF RESET_BTN_GPIO_Port->BSRR = (uint32_t)RESET_BTN_Pin

#define BAT_CONTROL_ON     BAT_CONTROL_GPIO_Port->BSRR = (uint32_t)BAT_CONTROL_Pin
#define BAT_CONTROL_OFF    BAT_CONTROL_GPIO_Port->BSRR = (uint32_t)BAT_CONTROL_Pin << 16

#define HDD_LR_ON   HDD_LED_R_GPIO_Port->BSRR = (uint32_t)HDD_LED_R_Pin
#define HDD_LR_OFF  HDD_LED_R_GPIO_Port->BSRR = (uint32_t)HDD_LED_R_Pin << 16
#define HDD_LG_ON   HDD_LED_G_GPIO_Port->BSRR = (uint32_t)HDD_LED_G_Pin
#define HDD_LG_OFF  HDD_LED_G_GPIO_Port->BSRR = (uint32_t)HDD_LED_G_Pin << 16

#define PWR_LR_ON   PWR_LED_R_GPIO_Port->BSRR = (uint32_t)PWR_LED_R_Pin
#define PWR_LR_OFF  PWR_LED_R_GPIO_Port->BSRR = (uint32_t)PWR_LED_R_Pin << 16
#define PWR_LG_ON   PWR_LED_G_GPIO_Port->BSRR = (uint32_t)PWR_LED_G_Pin
#define PWR_LG_OFF  PWR_LED_G_GPIO_Port->BSRR = (uint32_t)PWR_LED_G_Pin << 16

#define LED_LR_ON   GP_LED_R_GPIO_Port->BSRR = (uint32_t)GP_LED_R_Pin
#define LED_LR_OFF  GP_LED_R_GPIO_Port->BSRR = (uint32_t)GP_LED_R_Pin << 16
#define LED_LG_ON   GP_LED_G_GPIO_Port->BSRR = (uint32_t)GP_LED_G_Pin
#define LED_LG_OFF  GP_LED_G_GPIO_Port->BSRR = (uint32_t)GP_LED_G_Pin << 16

#define SYS_S3                  (SYS_S3_GPIO_Port->IDR & SYS_S3_Pin) != 0
#define SYS_S5                  (SYS_S5_GPIO_Port->IDR & SYS_S5_Pin) != 0
#define HDD_LED                 (HDD_LED_GPIO_Port->IDR & HDD_LED_Pin) != 0
#define RESET_PCIE              (RESET_PCIE_GPIO_Port->IDR & RESET_PCIE_Pin) != 0

//===================================================================
//===================================================================
```

```
//===================================================================
// CBCE_DIO.h
//===================================================================

#include "stm32f0xx_hal.h"

typedef struct
{
	uint8_t		RxBuf[100];
	uint8_t		RxFlag;
	uint8_t		RxSize;
	uint16_t	WDT_Counter;
	uint16_t	RST_Counter;
} _WDT;

void HAL_I2C_SlaveTxCpltCallback(I2C_HandleTypeDef *hi2c);
void LED_CTRL(void);
void WDT_CTRL(void);
void I2C_RD(I2C_HandleTypeDef*, uint8_t*);


//===================================================================
//===================================================================
```

```
//===================================================================
// main.c
//===================================================================

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f0xx_hal.h"

/* USER CODE BEGIN Includes */
#include "CBCE.h"

/* USER CODE END Includes */

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

I2C_HandleTypeDef hi2c1;
I2C_HandleTypeDef hi2c2;

IWDG_HandleTypeDef hiwdg;

TIM_HandleTypeDef htim1;

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/

uint8_t RxBuf[100];
_WDT WDT;


/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_ADC_Init(void);
static void MX_I2C2_Init(void);
static void MX_TIM1_Init(void);
static void MX_IWDG_Init(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------*/

/* USER CODE END PFP */

/* USER CODE BEGIN 0 */


/* USER CODE END 0 */

int main(void)
{

  /* USER CODE BEGIN 1 */
      volatile uint32_t i;

  /* 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_I2C1_Init();
  MX_ADC_Init();
  MX_I2C2_Init();
  MX_TIM1_Init();
  MX_IWDG_Init();

  /* USER CODE BEGIN 2 */
       WDT.RxFlag = 0;
       WDT.WDT_Counter = 0;
       WDT.RST_Counter = 0;
       RST_BTN_OFF;
       HDD_LG_OFF;
       PWR_LG_OFF;
       LED_LG_OFF;
       HDD_LR_OFF;
       PWR_LR_OFF;
       LED_LR_OFF;
       BAT_CONTROL_OFF;
       htim1.Instance->CR1 = 1;
       hiwdg.Instance->KR = 0x5555;
       hiwdg.Instance->RLR = 800; // WDT 100ms Timeout


  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
       //1ms Cycle Time
       while((htim1.Instance->SR & 2) == 0){}; htim1.Instance->SR = 0;

              LED_CTRL();
              I2C_RD(&hi2c1, RxBuf);
              WDT_CTRL();
              hiwdg.Instance->KR = 0xaaaa; // RELOAD HW WDT

  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */

  }
  /* USER CODE END 3 */

}

/** System Clock Configuration
*/
void SystemClock_Config(void)
{

  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_PeriphCLKInitTypeDef PeriphClkInit;

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_HSI14
                                |RCC_OSCILLATORTYPE_LSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSI14State = RCC_HSI14_ON;
  RCC_OscInitStruct.HSICalibrationValue = 16;
  RCC_OscInitStruct.HSI14CalibrationValue = 16;
  RCC_OscInitStruct.LSIState = RCC_LSI_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
```

```
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL12;
  RCC_OscInitStruct.PLL.PREDIV = RCC_PREDIV_DIV1;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Initializes the CPU, AHB and APB busses clocks
    */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_I2C1;
  PeriphClkInit.I2c1ClockSelection = RCC_I2C1CLKSOURCE_HSI;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Configure the Systick interrupt time
    */
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

    /**Configure the Systick
    */
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  /* SysTick_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* ADC init function */
static void MX_ADC_Init(void)
{

  ADC_ChannelConfTypeDef sConfig;

    /**Configure the global features of the ADC (Clock, Resolution, Data Alignment and
number of conversion)
    */
  hadc.Instance = ADC1;
  hadc.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
  hadc.Init.Resolution = ADC_RESOLUTION_12B;
  hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc.Init.ScanConvMode = ADC_SCAN_DIRECTION_FORWARD;
  hadc.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  hadc.Init.LowPowerAutoWait = DISABLE;
  hadc.Init.LowPowerAutoPowerOff = DISABLE;
  hadc.Init.ContinuousConvMode = DISABLE;
  hadc.Init.DiscontinuousConvMode = DISABLE;
  hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc.Init.DMAContinuousRequests = DISABLE;
  hadc.Init.Overrun = ADC_OVR_DATA_PRESERVED;
  if (HAL_ADC_Init(&hadc) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Configure for the selected ADC regular channel to be converted.
    */
  sConfig.Channel = ADC_CHANNEL_2;
```

```
  sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

}

/* I2C1 init function */
static void MX_I2C1_Init(void)
{

  hi2c1.Instance = I2C1;
  hi2c1.Init.Timing = 0x2000090E;
  hi2c1.Init.OwnAddress1 = 16;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Configure Analogue filter
    */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Configure Digital filter
    */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

}

/* I2C2 init function */
static void MX_I2C2_Init(void)
{

  hi2c2.Instance = I2C2;
  hi2c2.Init.Timing = 0x20303E5D;
  hi2c2.Init.OwnAddress1 = 0;
  hi2c2.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c2.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c2.Init.OwnAddress2 = 0;
  hi2c2.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c2.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c2.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c2) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Configure Analogue filter
    */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c2, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

    /**Configure Digital filter
    */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c2, 0) != HAL_OK)
```

```
  {
    _Error_Handler(__FILE__, __LINE__);
  }

}

/* IWDG init function */
static void MX_IWDG_Init(void)
{

  hiwdg.Instance = IWDG;
  hiwdg.Init.Prescaler = IWDG_PRESCALER_4;
  hiwdg.Init.Window = 4095;
  hiwdg.Init.Reload = 4095;
  if (HAL_IWDG_Init(&hiwdg) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

}

/* TIM1 init function */
static void MX_TIM1_Init(void)
{

  TIM_MasterConfigTypeDef sMasterConfig;
  TIM_OC_InitTypeDef sConfigOC;
  TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig;

  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 47;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 998;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_OC_Init(&htim1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  sConfigOC.OCMode = TIM_OCMODE_TIMING;
  sConfigOC.Pulse = 0;
  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_OC_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

  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(__FILE__, __LINE__);
  }
```

```
}

/** Configure pins as
        * Analog
        * Input
        * Output
        * EVENT_OUT
        * EXTI
*/
static void MX_GPIO_Init(void)
{

  GPIO_InitTypeDef GPIO_InitStruct;

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOF_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, GP_LED_G_Pin|GP_LED_R_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(RESET_BTN_GPIO_Port, RESET_BTN_Pin, GPIO_PIN_SET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(BAT_CONTROL_GPIO_Port, BAT_CONTROL_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(HDD_LED_R_GPIO_Port, HDD_LED_R_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, HDD_LED_G_Pin|PWR_LED_R_Pin|PWR_LED_G_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(PWR_OK_GPIO_Port, PWR_OK_Pin, GPIO_PIN_SET);

  /*Configure GPIO pins : GP_LED_G_Pin GP_LED_R_Pin */
  GPIO_InitStruct.Pin = GP_LED_G_Pin|GP_LED_R_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pin : RESET_BTN_Pin */
  GPIO_InitStruct.Pin = RESET_BTN_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_OD;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(RESET_BTN_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : PWR_GOOD_Pin RESET_PCIE_Pin SYS_S5_Pin SYS_STAT_Pin
                           HDD_LED_Pin SYS_S3_Pin */
  GPIO_InitStruct.Pin = PWR_GOOD_Pin|RESET_PCIE_Pin|SYS_S5_Pin|SYS_STAT_Pin
                        |HDD_LED_Pin|SYS_S3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : BAT_CONTROL_Pin */
  GPIO_InitStruct.Pin = BAT_CONTROL_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(BAT_CONTROL_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : CONFIG_Pin */
  GPIO_InitStruct.Pin = CONFIG_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
```

```
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(CONFIG_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : HDD_LED_R_Pin */
  GPIO_InitStruct.Pin = HDD_LED_R_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(HDD_LED_R_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : HDD_LED_G_Pin PWR_LED_R_Pin PWR_LED_G_Pin PWR_OK_Pin */
  GPIO_InitStruct.Pin = HDD_LED_G_Pin|PWR_LED_R_Pin|PWR_LED_G_Pin|PWR_OK_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pin : WDT_Pin */
  GPIO_InitStruct.Pin = WDT_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(WDT_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @param  None
  * @retval None
  */
void _Error_Handler(char * file, int line)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  while(1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT

/**
   * @brief Reports the name of the source file and the source line number
   * where the assert_param error has occurred.
   * @param file: pointer to the source file name
   * @param line: assert_param error line source number
   * @retval None
   */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */

}

#endif

//=========================================================================
//=========================================================================
```

```
//=====================================================================
// CBCE_DIO.c
//=====================================================================

#include "stm32f0xx_hal.h"
#include "CBCE.h"


extern IWDG_HandleTypeDef hiwdg;
extern _WDT WDT;
static uint32_t RxStatus = 0;


void WDT_CTRL(void)
{

	if(RESET_PCIE == 0)
	{
		WDT.RxFlag = 0;
		WDT.WDT_Counter = 0;
		WDT.RST_Counter = 0;
		RST_BTN_OFF;
		HDD_LG_OFF;
		PWR_LG_OFF;
		LED_LG_OFF;
		HDD_LR_OFF;
		PWR_LR_OFF;
		LED_LR_OFF;
		BAT_CONTROL_OFF;
	}
	else
	{
		if(WDT.RxFlag != 0)
		{
			WDT.RxFlag = 0;
			if((WDT.RxBuf[2] + (WDT.RxBuf[3] << 8)) == 0x55aa)
				WDT.WDT_Counter = WDT.RxBuf[0] + (WDT.RxBuf[1] << 8);
		}

		if(WDT.WDT_Counter == 1) WDT.RST_Counter = 200;
		if(WDT.WDT_Counter != 0) WDT.WDT_Counter--;

		if(WDT.RST_Counter != 0)
			{
				WDT.RST_Counter--;
				RST_BTN_ON;
			}
			else RST_BTN_OFF;
	}
}

void I2C_RD(I2C_HandleTypeDef *hi2c, uint8_t *RxBuf)
{
	if(HAL_I2C_GetState(hi2c) == HAL_I2C_STATE_READY)
	{
		RxStatus = HAL_I2C_Slave_Receive_IT (hi2c, WDT.RxBuf, 4);
		if(RxStatus != HAL_OK) Error_Handler();
	}
}

void HAL_I2C_SlaveRxCpltCallback(I2C_HandleTypeDef *hi2c)
{
	WDT.RxFlag = 1;
	WDT.RxSize = hi2c->XferCount;
}

void LED_CTRL(void)
{

	// PWR LED Control
	if(SYS_S5 != 0) {PWR_LG_ON;}
```

```
                    else {PWR_LG_OFF;}

        // HDD LED Control
        if((HDD_LED == 0) & (SYS_S5 != 0)) HDD_LG_ON;
                    else HDD_LG_OFF;
}

//=============================================================================
//=============================================================================
```

Таблица регистрации изменений

Изм.	Номера листов (страниц)				Всего листов (страниц) в док.	Номер док.	Подп.	Дата
	изме-ненных	заме-ненных	новых	аннули-рован-ных				