Программа для ЭВМ

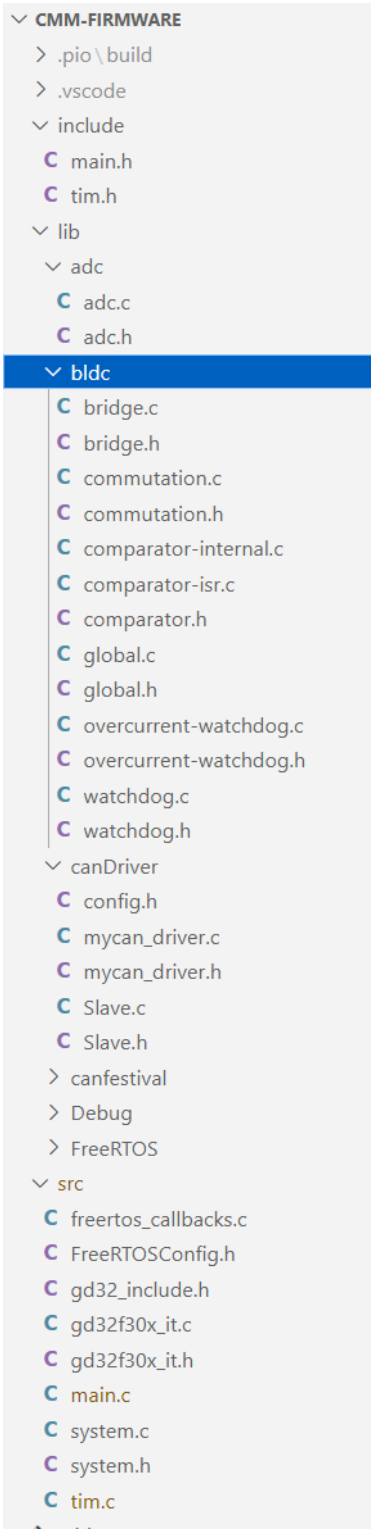
«СММ Комитас»
Депонируемые материалы,
идентифицирующие программу для ЭВМ

Правообладатель:
Общество с ограниченной ответственностью «КОМИТАС»
125212, Российская Федерация, Москва, Головинское шоссе, д. 5, корп. 1, офис 10015/1

Авторы: отказывались быть упомянутыми в качестве таковых

Оглавление

1. **Инициализация** 3
2. **Инициализация периферийного оборудования** 3
3. **Инициализация таймера** 9
4. **Описание функций callback от разной периферии** 10
5. **Обслуживание работы мотора. Телеметрия** 16
6. **Низкоуровневая связь с мотором** 28
7. **Глобальные данные** 31
8. **Защита по току** 34
9. **Сторожевой таймер** 35
10. **Инициализация АЦП** 35
11. **Драйвер CAN** 38


CMM-FIRMWARE
.pio \ build
.vscode
include
main.h
tim.h
lib
adc
adc.c
adc.h
bldc
bridge.c
bridge.h
commutation.c
commutation.h
comparator-internal.c
comparator-isr.c
comparator.h
global.c
global.h
overcurrent-watchdog.c
overcurrent-watchdog.h
watchdog.c
watchdog.h
canDriver
config.h
mycan_driver.c
mycan_driver.h
Slave.c
Slave.h
canfestival
Debug
FreeRTOS
src
freertos_callbacks.c
FreeRTOSConfig.h
gd32_include.h
gd32f30x_it.c
gd32f30x_it.h
main.c
system.c
system.h
tim.c

1. Инициализация

```c
#include <gd32_include.h>
#include <stdio.h>
#include <stdarg.h>
#include <cmsis_os2.h>
#include "mycan_driver.h"
#include "adc.h"
#include "bridge.h"
#include "gd32f30x.h"
#include "tim.h"
#include "system.h"


int main(void)
{
    rcu_config();
    gpio_config();
    nvic_config();
    dac_config();
    uart_config();
    timer_config();
    systick_config();
    //watchdog_start(10); // 10ms watchdog timeout

    /* Initialize CMSIS-RTOS */
    osKernelInitialize();

    BLDC_Task(NULL);

    return 0;
}
```

2. Инициализация периферийного оборудования

```c
#include <system.h>
#include <stdio.h>
#include <can.h>

#define ARRAYNUM(arr_name)      (uint32_t)(sizeof(arr_name) / sizeof(*(arr_name)))
#define USART2_DATA_ADDRESS     ((uint32_t)&USART_DATA(USART2))

/*!
    \brief      configure the different system clocks
    \param[in]  none
    \param[out] none
    \retval     none
```

```
*/
void rcu_config(void)
{
    rcu_periph_clock_enable(RCU_GPIOA);
    rcu_periph_clock_enable(RCU_GPIOB);
    rcu_periph_clock_enable(RCU_GPIOC);
    rcu_periph_clock_enable(RCU_GPIOD);
    rcu_periph_clock_enable(RCU_AF);
    rcu_periph_clock_enable(RCU_DAC);
    rcu_periph_clock_enable(RCU_CAN0);
    rcu_periph_clock_enable(RCU_DMA0);
    rcu_periph_clock_enable(RCU_USART2);

    rcu_periph_clock_enable(RCU_ADC0);
    rcu_adc_clock_config(RCU_CKADC_CKAPB2_DIV6);
}

void systick_config(void)
{
    /* setup systick timer for 1000Hz interrupts */
    if (SysTick_Config(SystemCoreClock / 1000U))
    {
        /* capture error */
        while (1)
        {
        }
    }
    /* configure the systick handler priority */
    NVIC_SetPriority(SysTick_IRQn, 0x00U);
}

/*!
    \brief      configure the GPIO peripheral
    \param[in]  none
    \param[out] none
    \retval     none
*/
void gpio_config(void)
{
    /* config the GPIO as analog mode */
    gpio_init(GPIOA, GPIO_MODE_AIN, GPIO_OSPEED_10MHZ, GPIO_PIN_0);
    gpio_init(GPIOA, GPIO_MODE_AIN, GPIO_OSPEED_10MHZ, GPIO_PIN_1);
    gpio_init(GPIOA, GPIO_MODE_AIN, GPIO_OSPEED_10MHZ, GPIO_PIN_2);
    gpio_init(GPIOA, GPIO_MODE_AIN, GPIO_OSPEED_10MHZ, GPIO_PIN_3);
    gpio_init(GPIOA, GPIO_MODE_AIN, GPIO_OSPEED_10MHZ, GPIO_PIN_4);
    gpio_init(GPIOA, GPIO_MODE_AIN, GPIO_OSPEED_10MHZ, GPIO_PIN_6);
    gpio_init(GPIOA, GPIO_MODE_AIN, GPIO_OSPEED_10MHZ, GPIO_PIN_7);
    gpio_init(GPIOB, GPIO_MODE_AIN, GPIO_OSPEED_10MHZ, GPIO_PIN_0);
    gpio_init(GPIOB, GPIO_MODE_AIN, GPIO_OSPEED_10MHZ, GPIO_PIN_1);
    gpio_init(GPIOC, GPIO_MODE_AIN, GPIO_OSPEED_10MHZ, GPIO_PIN_0);
    gpio_init(GPIOC, GPIO_MODE_AIN, GPIO_OSPEED_10MHZ, GPIO_PIN_1);
```

```
    gpio_init(GPIOC, GPIO_MODE_AIN, GPIO_OSPEED_10MHZ, GPIO_PIN_2);
    gpio_init(GPIOC, GPIO_MODE_AIN, GPIO_OSPEED_10MHZ, GPIO_PIN_3);
    gpio_init(GPIOC, GPIO_MODE_AIN, GPIO_OSPEED_10MHZ, GPIO_PIN_4);
    gpio_init(GPIOC, GPIO_MODE_AIN, GPIO_OSPEED_10MHZ, GPIO_PIN_5);

    /* config the GPIO as analog mode */
    gpio_init(GPIOB,GPIO_MODE_IPU,GPIO_OSPEED_50MHZ,GPIO_PIN_8);
    gpio_init(GPIOB,GPIO_MODE_AF_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_9);
    gpio_pin_remap_config(GPIO_CAN_PARTIAL_REMAP,ENABLE);

    /* config the GPIO as USART mode */
    gpio_init(GPIOC, GPIO_MODE_IN_FLOATING, GPIO_OSPEED_50MHZ, GPIO_PIN_11);
    gpio_pin_remap_config(GPIO_USART2_PARTIAL_REMAP,ENABLE);

    /* config the GPIO as output for led */
    gpio_pin_remap_config(GPIO_SWJ_SWDPENABLE_REMAP,ENABLE);
    gpio_init(GPIOB, GPIO_MODE_OUT_PP, GPIO_OSPEED_50MHZ, GPIO_PIN_4);
    gpio_init(GPIOB, GPIO_MODE_OUT_PP, GPIO_OSPEED_50MHZ, GPIO_PIN_5);
    gpio_init(GPIOA, GPIO_MODE_OUT_PP, GPIO_OSPEED_50MHZ, GPIO_PIN_11);
    gpio_init(GPIOA, GPIO_MODE_OUT_PP, GPIO_OSPEED_50MHZ, GPIO_PIN_12);
    gpio_init(GPIOC, GPIO_MODE_OUT_PP, GPIO_OSPEED_50MHZ, GPIO_PIN_10);
    gpio_init(GPIOC, GPIO_MODE_OUT_PP, GPIO_OSPEED_50MHZ, GPIO_PIN_13);
    gpio_init(GPIOD, GPIO_MODE_OUT_PP, GPIO_OSPEED_50MHZ, GPIO_PIN_2);
    gpio_init(GPIOC, GPIO_MODE_OUT_PP, GPIO_OSPEED_50MHZ, GPIO_PIN_12);
    gpio_init(GPIOC, GPIO_MODE_OUT_PP, GPIO_OSPEED_50MHZ, GPIO_PIN_9);
    gpio_init(GPIOC, GPIO_MODE_OUT_PP, GPIO_OSPEED_50MHZ, GPIO_PIN_8);

    //Set prepare led state
    gpio_bit_set(GPIOB, GPIO_PIN_4);
    gpio_bit_set(GPIOB, GPIO_PIN_5);
    gpio_bit_set(GPIOA, GPIO_PIN_11);
    gpio_bit_set(GPIOA, GPIO_PIN_12);
    gpio_bit_set(GPIOC, GPIO_PIN_10);
    gpio_bit_set(GPIOC, GPIO_PIN_13);
    gpio_bit_set(GPIOD, GPIO_PIN_2);
    gpio_bit_set(GPIOC, GPIO_PIN_12);
    gpio_bit_set(GPIOC, GPIO_PIN_9);
    gpio_bit_set(GPIOC, GPIO_PIN_8);

    /* config the GPIO as input for sensor and aux */
    //gpio_init(GPIOB, GPIO_MODE_IPD, GPIO_OSPEED_50MHZ, GPIO_PIN_1);
    gpio_init(GPIOB, GPIO_MODE_IPD, GPIO_OSPEED_50MHZ, GPIO_PIN_2);

    /* config the GPIO as input/output for CAN addressing */
    gpio_pin_remap_config(GPIO_PD01_REMAP,ENABLE);
    gpio_init(GPIOD, GPIO_MODE_IPD, GPIO_OSPEED_50MHZ, GPIO_PIN_0);
    gpio_init(GPIOD, GPIO_MODE_OUT_PP, GPIO_OSPEED_50MHZ, GPIO_PIN_1);

    //Motor
    gpio_init(GPIOC, GPIO_MODE_OUT_PP, GPIO_OSPEED_50MHZ, GPIO_PIN_6);
//PWRMotorOn
```

```
    gpio_bit_set(GPIOC, GPIO_PIN_6);

    //Interroll Motor
    gpio_init(GPIOC, GPIO_MODE_IPD, GPIO_OSPEED_50MHZ, GPIO_PIN_7);
    gpio_init(GPIOB, GPIO_MODE_OUT_PP, GPIO_OSPEED_50MHZ, GPIO_PIN_11);
//MotorDir
    //gpio_bit_reset(GPIOB, GPIO_PIN_11);

    //Daemon Motor

    /*configure PA8 PA9 PA10(TIMER0 CH0 CH1 CH2) as alternate function*/
    gpio_init(GPIOA,GPIO_MODE_AF_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_8);
    gpio_init(GPIOA,GPIO_MODE_AF_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_9);
    gpio_init(GPIOA,GPIO_MODE_AF_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_10);

    /*configure PB13 PB14 PB15(TIMER0 CH0N CH1N CH2N) as alternate function*/
    gpio_init(GPIOB,GPIO_MODE_AF_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_13);
    gpio_init(GPIOB,GPIO_MODE_AF_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_14);
    gpio_init(GPIOB,GPIO_MODE_AF_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_15);

    /*configure PA8 PA9 PA10(TIMER0 CH0 CH1 CH2) as alternate function*/
    //gpio_init(GPIOA,GPIO_MODE_OUT_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_8);
    //gpio_init(GPIOA,GPIO_MODE_OUT_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_9);
    //gpio_init(GPIOA,GPIO_MODE_OUT_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_10);
//
    ///*configure PB13 PB14 PB15(TIMER0 CH0N CH1N CH2N) as alternate function*/
    //gpio_init(GPIOB,GPIO_MODE_OUT_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_13);
    //gpio_init(GPIOB,GPIO_MODE_OUT_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_14);
    //gpio_init(GPIOB,GPIO_MODE_OUT_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_15);
//

    /* config the GPIO as input for HAL sensor */
    gpio_init(GPIOA, GPIO_MODE_IN_FLOATING, GPIO_OSPEED_50MHZ, GPIO_PIN_15);
    gpio_init(GPIOB, GPIO_MODE_IN_FLOATING, GPIO_OSPEED_50MHZ, GPIO_PIN_3);
    gpio_init(GPIOB, GPIO_MODE_IN_FLOATING, GPIO_OSPEED_50MHZ, GPIO_PIN_10);

    nvic_irq_enable(EXTI3_IRQn,2,0);
    gpio_exti_source_select(GPIOB,GPIO_PIN_3);
    exti_init(EXTI_3,EXTI_INTERRUPT,EXTI_TRIG_BOTH);
    exti_interrupt_flag_clear(EXTI_3);

    nvic_irq_enable(EXTI10_15_IRQn,2,0);
    gpio_exti_source_select(GPIOB,GPIO_PIN_10);
    exti_init(EXTI_10,EXTI_INTERRUPT,EXTI_TRIG_BOTH);
    exti_interrupt_flag_clear(EXTI_10);

    gpio_exti_source_select(GPIOA,GPIO_PIN_15);
    exti_init(EXTI_15,EXTI_INTERRUPT,EXTI_TRIG_BOTH);
    exti_interrupt_flag_clear(EXTI_15);
}
```

```c
void dac_config(void)
{
    dac_deinit();

    dac_trigger_disable(DAC0);
    dac_wave_mode_config(DAC0, DAC_WAVE_DISABLE);
    dac_output_buffer_disable(DAC0);
    dac_disable(DAC0);

    dac_trigger_disable(DAC1);
    dac_wave_mode_config(DAC1, DAC_WAVE_DISABLE);
    dac_output_buffer_enable(DAC1);

    dac_enable(DAC1);
}

/*!
    \brief      configure the nested vectored interrupt controller
    \param[in]  none
    \param[out] none
    \retval     none
*/
void nvic_config(void)
{
    nvic_irq_enable(TIMER0_TRG_CMT_TIMER10_IRQn, 0, 1);
    nvic_irq_enable(USBD_LP_CAN0_RX0_IRQn, 0, 0);
    nvic_irq_enable(CAN0_RX1_IRQn, 0, 0);
    nvic_irq_enable(USART2_IRQn, 0, 1);
    nvic_irq_enable(DMA0_Channel0_IRQn, 0, 1);
}


/*!
    \brief      configure the TIMER peripheral
    \param[in]  none
    \param[out] none
    \retval     none
*/
void timer_config(void)
{
    /* ----------------------------------------------------------------------
    TIMER0 configuration:
    generate 3 complementary PWM signal.
    TIMER0CLK is fixed to systemcoreclock, the TIMER0 prescaler is equal to 119
    so the TIMER0 counter clock used is 1MHz.
    insert a dead time equal to 200/systemcoreclock =1.67us
    configure the break feature, active at low level, and using the automatic
    output enable feature.
    use the locking parameters level 0.
    ---------------------------------------------------------------------- */
    timer_oc_parameter_struct timer_ocintpara;
```

```
timer_parameter_struct timer_initpara;
timer_break_parameter_struct timer_breakpara;

rcu_periph_clock_enable(RCU_TIMER0);

timer_deinit(TIMER0);

/* TIMER0 configuration */
timer_initpara.prescaler         = 7;
timer_initpara.alignedmode       = TIMER_COUNTER_EDGE;
timer_initpara.counterdirection  = TIMER_COUNTER_UP;
timer_initpara.period            = 800;
timer_initpara.clockdivision     = TIMER_CKDIV_DIV1;
timer_initpara.repetitioncounter = 0;
timer_init(TIMER0,&timer_initpara);

 /* CH0/CH0N,CH1/CH1N and CH2/CH2N configuration in timing mode */
timer_ocintpara.outputstate  = TIMER_CCX_ENABLE;
timer_ocintpara.outputnstate = TIMER_CCXN_ENABLE;
timer_ocintpara.ocpolarity   = TIMER_OCN_POLARITY_HIGH;
timer_ocintpara.ocnpolarity  = TIMER_OCN_POLARITY_HIGH;
timer_ocintpara.ocidlestate  = TIMER_OCN_IDLE_STATE_LOW;
timer_ocintpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;

timer_channel_output_config(TIMER0,TIMER_CH_0,&timer_ocintpara);
timer_channel_output_config(TIMER0,TIMER_CH_1,&timer_ocintpara);
timer_channel_output_config(TIMER0,TIMER_CH_2,&timer_ocintpara);

timer_channel_output_pulse_value_config(TIMER0,TIMER_CH_0,0);
timer_channel_output_mode_config(TIMER0,TIMER_CH_0,TIMER_OC_MODE_PWM0);
timer_channel_output_shadow_config(TIMER0,TIMER_CH_0,TIMER_OC_SHADOW_ENABLE);

timer_channel_output_pulse_value_config(TIMER0,TIMER_CH_1,0);
timer_channel_output_mode_config(TIMER0,TIMER_CH_1,TIMER_OC_MODE_PWM0);
timer_channel_output_shadow_config(TIMER0,TIMER_CH_1,TIMER_OC_SHADOW_ENABLE);

timer_channel_output_pulse_value_config(TIMER0,TIMER_CH_2,0);
timer_channel_output_mode_config(TIMER0,TIMER_CH_2,TIMER_OC_MODE_PWM0);
timer_channel_output_shadow_config(TIMER0,TIMER_CH_2,TIMER_OC_SHADOW_ENABLE);


/* automatic output enable, break, dead time and lock configuration*/
timer_breakpara.runoffstate      = TIMER_ROS_STATE_ENABLE;
timer_breakpara.ideloffstate     = TIMER_IOS_STATE_ENABLE ;
timer_breakpara.deadtime         = 300;
timer_breakpara.breakpolarity    = TIMER_BREAK_POLARITY_LOW;
timer_breakpara.outputautostate  = TIMER_OUTAUTO_ENABLE;
timer_breakpara.protectmode      = TIMER_CCHP_PROT_OFF;
timer_breakpara.breakstate       = TIMER_BREAK_DISABLE;
timer_break_config(TIMER0,&timer_breakpara);
```

```
    /* TIMER0 primary output function enable */
    timer_primary_output_config(TIMER0,ENABLE);

    /* TIMER0 channel control update interrupt enable */
    timer_interrupt_enable(TIMER0,TIMER_INT_CMT);
    /* TIMER0 break interrupt disable */
    timer_interrupt_disable(TIMER0,TIMER_INT_BRK);

    /* TIMER0 counter enable */
    timer_enable(TIMER0);
}

uint8_t rxbuffer[10];
uint16_t rxcount;

void uart_config(void)
{
    //dma_parameter_struct dma_init_struct;
    /* USART configure 115200 8N1 */
    usart_deinit(USART2);
    usart_word_length_set(USART2, USART_WL_8BIT);
    usart_stop_bit_set(USART2, USART_STB_1BIT);
    usart_parity_config(USART2, USART_PM_NONE);
    usart_baudrate_set(USART2, 1000000U);
    usart_receive_config(USART2, USART_RECEIVE_ENABLE);
    usart_transmit_config(USART2, USART_TRANSMIT_ENABLE);
    usart_enable(USART2);

    usart_interrupt_enable(USART2, USART_INT_RBNE);
    usart_interrupt_enable(USART2, USART_INT_RT);
    usart_receiver_timeout_enable(USART2);
    //usart_block_length_config(USART2,3);
    usart_receiver_timeout_threshold_config(USART2, 100);

}
```

3. Инициализация таймера

```
#include "tim.h"
#include "gd32f30x.h"

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

void MX_TIM2_Init(void)
{
    timer_parameter_struct timer_initpara;
```

```
    rcu_periph_clock_enable(RCU_TIMER7);

    timer_deinit(TIMER7);

    /* TIMER0 configuration */
    timer_initpara.prescaler         = 1200-1;
    timer_initpara.alignedmode       = TIMER_COUNTER_EDGE;
    timer_initpara.counterdirection  = TIMER_COUNTER_UP;
    timer_initpara.period            = 0xFFFF;
    timer_initpara.clockdivision     = TIMER_CKDIV_DIV1;
    timer_initpara.repetitioncounter = 0;
    timer_init(TIMER7,&timer_initpara);
    timer_autoreload_value_config(TIMER7,0xFFFF);
    timer_interrupt_enable(TIMER7,TIMER_INT_UP);
    nvic_irq_enable(TIMER7_UP_TIMER12_IRQn, 0, 1);
    /* TIMER0 counter enable */
    timer_enable(TIMER7);
    // TODO lock registers
}
```

4. Описание функций callback от разной периферии

```
#include "gd32f30x_it.h"
#include "main.h"
#include "global.h"
#include "system.h"
#include "bridge.h"
#include "mycan_driver.h"

__IO uint32_t step = 1;
extern uint32_t globalHeartbeat_50us;

volatile static uint32_t delay;

void delay_1ms(uint32_t count)
{
    delay = count;

    while (0U != delay)
    {
    }
}

void delay_decrement(void)
{
    if (0U != delay)
```

```
    {
        delay--;
    }
}
/*!
    \brief      this function handles NMI exception
    \param[in]  none
    \param[out] none
    \retval     none
*/
void NMI_Handler(void)
{
}

/*!
    \brief      this function handles HardFault exception
    \param[in]  none
    \param[out] none
    \retval     none
*/

void HardFault_Handler(void)
{
    /* if Hard Fault exception occurs, go to infinite loop */
    while (1){
    }
}

/*!
    \brief      this function handles MemManage exception
    \param[in]  none
    \param[out] none
    \retval     none
*/
void MemManage_Handler(void)
{
    /* if Memory Manage exception occurs, go to infinite loop */
    while (1){
    }
}

/*!
    \brief      this function handles BusFault exception
    \param[in]  none
    \param[out] none
    \retval     none
*/
void BusFault_Handler(void)
{
    /* if Bus Fault exception occurs, go to infinite loop */
    while (1){
```

```
    }
}

/*!
    \brief      this function handles UsageFault exception
    \param[in]  none
    \param[out] none
    \retval     none
*/
void UsageFault_Handler(void)
{
    /* if Usage Fault exception occurs, go to infinite loop */
    while (1){
    }
}

/*!
    \brief      this function handles DebugMon exception
    \param[in]  none
    \param[out] none
    \retval     none
*/
void DebugMon_Handler(void)
{
}



/*!
    \brief      this function handles TIMER0 interrupt request
    \param[in]  none
    \param[out] none
    \retval     none
*/

void TIMER0_TRG_CMT_TIMER10_IRQHandler(void)
{
    timer_interrupt_flag_clear(TIMER0,TIMER_INT_FLAG_CMT);

    //bridge_commutate();
    //BLDC_Move();

}
can_receive_message_struct receive_message;

void USBD_LP_CAN0_RX0_IRQHandler(void)
{
    static CanRxMsgTypeDef RxMsg;
    /* check the receive message */
    can_message_receive(CAN0, CAN_FIFO1, &receive_message);
```

```
    if(CAN_FF_STANDARD == receive_message.rx_ff){
      memcpy(RxMsg.Data,receive_message.rx_data, 8);
      RxMsg.StdId = receive_message.rx_sfid;
      CAN_Rcv_DateFromISR(&RxMsg);
    }
    //if(CAN_FF_STANDARD == receive_message.rx_ff){
    //    CAN_Rcv_DateFromISR(&receive_message);
    //}
}

/*!
    \brief      this function handles CAN1 RX0 exception
    \param[in]  none
    \param[out] none
    \retval     none
*/

void CAN0_RX1_IRQHandler(void)
{
    static CanRxMsgTypeDef RxMsg;
    /* check the receive message */
    can_message_receive(CAN0, CAN_FIFO1, &receive_message);

    if(CAN_FF_STANDARD == receive_message.rx_ff){
      memcpy(RxMsg.Data,receive_message.rx_data, 8);
      RxMsg.StdId = receive_message.rx_sfid;
      RxMsg.DLC = receive_message.rx_dlen;
      RxMsg.RTR = receive_message.rx_ft;
      CAN_Rcv_DateFromISR(&RxMsg);
    }
    //if(CAN_FF_STANDARD == receive_message.rx_ff){
    //    CAN_Rcv_DateFromISR(&receive_message);
    //}
}
unsigned char testt=0;
void TIMER7_UP_TIMER12_IRQHandler(void)
{
  // timeout waiting for zero-cross, go to open loop
  // TODO should load commutation timer default values
  if (timer_flag_get(TIMER7, TIMER_FLAG_UP)) {
    TIMx_DispatchFromISR();

    timer_flag_clear(TIMER7, TIMER_FLAG_UP);
  }
}

unsigned char last_pos=0;
unsigned char pos=0;
void USART2_IRQHandler(void)
{
    if(RESET != usart_interrupt_flag_get(USART2, USART_INT_FLAG_RBNE)){
```

```
        /* receive data */
      if (rxcount>5)rxcount = 0;
      rxbuffer[rxcount++] = usart_data_receive(USART2);
      rxbuffer[0] = rxbuffer[0] - 0x80;
      if ((rxbuffer[0] < 7) && (rxbuffer[0] > 0)){
        pos = rxbuffer[0]^0x07;

        if ((pos != last_pos )&&(pos >0)){
          last_pos = pos;
          /*g.Telemetry.Hall_Sensors.Hall_Sensor[0] = (pos>>0)&0x01;
          g.Telemetry.Hall_Sensors.Hall_Sensor[1] = (pos>>2)&0x01;
          g.Telemetry.Hall_Sensors.Hall_Sensor[2] = (pos>>1)&0x01;*/
          g.Telemetry.Hall_Sensors.Hall_Sensor[0] = (pos>>0)&0x01;
          g.Telemetry.Hall_Sensors.Hall_Sensor[1] = (pos>>1)&0x01;
          g.Telemetry.Hall_Sensors.Hall_Sensor[2] = (pos>>2)&0x01;
        }
        gpio_bit_reset(GPIOC, GPIO_PIN_12);
      }

    }
    if(RESET != usart_interrupt_flag_get(USART2, USART_INT_FLAG_RT)){
      rxcount = 0;
      gpio_bit_set(GPIOC, GPIO_PIN_12);
      usart_interrupt_flag_clear(USART2, USART_INT_FLAG_RT);
    }
}

unsigned short currB[1000];
unsigned short currC[1000];
unsigned short voltA[1000];
unsigned short voltB[1000];
unsigned short voltC[1000];
unsigned short adc_count=0;
void DMA0_Channel0_IRQHandler(void)
{
    if(dma_interrupt_flag_get(DMA0, DMA_CH0, DMA_INT_FLAG_FTF)){
        dma_interrupt_flag_clear(DMA0, DMA_CH0, DMA_INT_FLAG_G);
        currB[adc_count]=g.adc_buffer[ADC_IDX_PHASEB_CURRENT];
        currC[adc_count]=g.adc_buffer[ADC_IDX_PHASEC_CURRENT];
        voltA[adc_count]=g.adc_buffer[ADC_IDX_PHASEA_VOLTAGE];
        voltB[adc_count]=g.adc_buffer[ADC_IDX_PHASEB_VOLTAGE];
        voltC[adc_count]=g.adc_buffer[ADC_IDX_PHASEC_VOLTAGE];
        adc_count++;
        if(adc_count>999)adc_count=0;
    }
}

void EXTI3_IRQHandler(void)
{
    if (RESET != exti_interrupt_flag_get(EXTI_3)) {
```

```
        //timer_counter_value_config(COMMUTATION_TIMER,TIMER_CAR(COMMUTATION_TIME
R)-5);
        exti_interrupt_flag_clear(EXTI_3);
    }
}

void EXTI10_15_IRQHandler(void)
{
    if (RESET != exti_interrupt_flag_get(EXTI_10)) {
        //timer_counter_value_config(COMMUTATION_TIMER,TIMER_CAR(COMMUTATION_TIME
R)-5);
        exti_interrupt_flag_clear(EXTI_10);
    }
    if (RESET != exti_interrupt_flag_get(EXTI_15)) {
        //timer_counter_value_config(COMMUTATION_TIMER,TIMER_CAR(COMMUTATION_TIME
R)-5);
        exti_interrupt_flag_clear(EXTI_15);
    }
}

void TIMER5_IRQHandler(void)
{
  // timeout waiting for zero-cross, go to open loop
  // TODO should load commutation timer default values
  if (timer_flag_get(ZC_TIMER, TIMER_FLAG_UP)) {
    //comparator_zc_timeout_isr();
    globalHeartbeat_50us++;
    timer_flag_clear(ZC_TIMER, TIMER_FLAG_UP);
  }
}

void TIMER6_IRQHandler(void)
{
  if (timer_flag_get(COMMUTATION_TIMER, TIMER_FLAG_UP)) {
    commutation_isr();
    timer_flag_clear(COMMUTATION_TIMER, TIMER_FLAG_UP);
  }

  // check cc1 interrupt
  // ccr1 = comparator blanking period
  if (timer_flag_get(COMMUTATION_TIMER, TIMER_FLAG_CH0)) {
    comparator_unblank_isr();
    timer_flag_clear(COMMUTATION_TIMER, TIMER_FLAG_CH0);
  }
}

void TIMER0_Channel_IRQHandler() {
  if (timer_flag_get(TIMER0, TIMER_FLAG_CH3)) {
    adc_start();
    timer_flag_clear(TIMER0, TIMER_FLAG_CH3);
  }
```

```
}
```

5. Обслуживание работы мотора. Телеметрия

```
#include <bridge.h>
#include "gd32f30x.h"
#include <stdio.h>
#include "FreeRTOS.h" // для отладки
#include "cmsis_os.h" // для отладки
#include "system.h" // для отладки
#include <watchdog.h>
#include <commutation.h>
#include <overcurrent-watchdog.h>
#include <comparator.h>
#include <stdlib.h>
#include <global.h>
#include "mycan_driver.h"
#include "Slave.h"


volatile bridge_state_e g_bridge_state;
volatile bridge_comm_step_e g_bridge_comm_step;

volatile uint16_t g_bridge_run_duty = 300; // run mode duty
volatile uint8_t g_bridge_audio_duty; // audio mode duty
// use this throttle in open loop startup
const uint16_t startup_throttle = 75;

// declared/documented in global.h
const uint16_t slow_run_zc_confirmations_required = 14;
uint16_t run_zc_confirmations_required = slow_run_zc_confirmations_required;
volatile bool starting;
const uint16_t startup_commutation_period_ticks = 7500;
const uint16_t startup_zc_confirmations_required = 15;
volatile uint32_t zc_counter;
uint32_t zc_confirmations_required = startup_zc_confirmations_required;

typedef struct {
    uint32_t CCER; // CCER register value
    volatile uint32_t pinAL; // ccr register address to write zero
    volatile uint32_t pinBL; // ccr register address to write zero
    volatile uint32_t pinCL; // ccr register address to write zero
} commutation_state_t;

#define TIM1_CCR1_ADDR (volatile uint32_t *)(TIMER0 + 0x34)
#define TIM1_CCR2_ADDR (volatile uint32_t *)(TIMER0 + 0x38)
#define TIM1_CCR3_ADDR (volatile uint32_t *)(TIMER0 + 0x3C)
```

```
#define TIM1_CCR4_ADDR (volatile uint32_t *)(TIMER0 + 0x40)

volatile commutation_state_t bridge_step_forward[8] = {
    { 0x00000000, 0, 0, 0 },//0x00000140   0x00000354   1, 0, 0

    { 0x00000050, 0, 0, 1 },//0x00000014   0x000001FC   1, 0, 0
    { 0x00000005, 0, 1, 0 },//0x00000410   0x00000CF1   0, 1, 0
    { 0x00000005, 0, 0, 1 },//0x00000500   0x0000070D   0, 0, 1
    { 0x00000500, 1, 0, 0 },//0x00000005   0x00000D0F   0, 0, 1
    { 0x00000050, 1, 0, 0 },//0x00000140   0x00000354   1, 0, 0
    { 0x00000500, 0, 1, 0 },//0x00000041   0x00000453   0, 1, 0

    { 0x00000000, 0, 0, 0 },//0x00000140   0x00000354   1, 0, 0
};

volatile commutation_state_t bridge_step_reverse[8] = {
    { 0x00000000, 0, 0, 0 },//0x00000140   0x00000354   1, 0, 0

    { 0x00000500, 0, 1, 0 },//0x00000014   0x000001FC   1, 0, 0
    { 0x00000050, 1, 0, 0 },//0x00000410   0x00000CF1   0, 1, 0
    { 0x00000500, 1, 0, 0 },//0x00000500   0x0000070D   0, 0, 1
    { 0x00000005, 0, 0, 1 },//0x00000005   0x00000D0F   0, 0, 1
    { 0x00000005, 0, 1, 0 },//0x00000140   0x00000354   1, 0, 0
    { 0x00000050, 0, 0, 1 },//0x00000041   0x00000453   0, 1, 0

    { 0x00000000, 0, 0, 0 },//0x00000140   0x00000354   1, 0, 0
};

#define holl_table_num 7
#define M_ABC_P 1
#define M_ABC_N 0

typedef struct
{
    unsigned char ha;
    unsigned char hb;
    unsigned char hc;
    unsigned char dir;
}TYPE_POSITION_ABC;
TYPE_POSITION_ABC myholl;

void bridge_initialize()
{
    timer_oc_parameter_struct timer_ocintpara;
    timer_parameter_struct timer_initpara;
    timer_break_parameter_struct timer_breakpara;

    rcu_periph_clock_enable(RCU_TIMER0);

    timer_deinit(TIMER0);
```

```
/* TIMER0 configuration */
timer_initpara.prescaler         = 0;
timer_initpara.alignedmode       = TIMER_COUNTER_EDGE;
timer_initpara.counterdirection  = TIMER_COUNTER_UP;
timer_initpara.period            = 2048;
timer_initpara.clockdivision     = TIMER_CKDIV_DIV1;
timer_initpara.repetitioncounter = 0;
timer_init(TIMER0,&timer_initpara);
*(TIM1_CCR4_ADDR) = 1024;
 /* CH0/CH0N,CH1/CH1N and CH2/CH2N configuration in timing mode */
timer_ocintpara.outputstate  = TIMER_CCX_ENABLE;
timer_ocintpara.outputnstate = TIMER_CCXN_ENABLE;
timer_ocintpara.ocpolarity   = TIMER_OCN_POLARITY_HIGH;
timer_ocintpara.ocnpolarity  = TIMER_OCN_POLARITY_HIGH;
timer_ocintpara.ocidlestate  = TIMER_OCN_IDLE_STATE_LOW;
timer_ocintpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;

timer_channel_output_config(TIMER0,TIMER_CH_0,&timer_ocintpara);
timer_channel_output_config(TIMER0,TIMER_CH_1,&timer_ocintpara);
timer_channel_output_config(TIMER0,TIMER_CH_2,&timer_ocintpara);

timer_channel_output_pulse_value_config(TIMER0,TIMER_CH_0,0);
timer_channel_output_mode_config(TIMER0,TIMER_CH_0,TIMER_OC_MODE_PWM0);
timer_channel_output_shadow_config(TIMER0,TIMER_CH_0,TIMER_OC_SHADOW_ENABLE);

timer_channel_output_pulse_value_config(TIMER0,TIMER_CH_1,0);
timer_channel_output_mode_config(TIMER0,TIMER_CH_1,TIMER_OC_MODE_PWM0);
timer_channel_output_shadow_config(TIMER0,TIMER_CH_1,TIMER_OC_SHADOW_ENABLE);

timer_channel_output_pulse_value_config(TIMER0,TIMER_CH_2,0);
timer_channel_output_mode_config(TIMER0,TIMER_CH_2,TIMER_OC_MODE_PWM0);
timer_channel_output_shadow_config(TIMER0,TIMER_CH_2,TIMER_OC_SHADOW_ENABLE);


/* automatic output enable, break, dead time and lock configuration*/
timer_breakpara.runoffstate      = TIMER_ROS_STATE_ENABLE;
timer_breakpara.ideloffstate     = TIMER_IOS_STATE_ENABLE ;
timer_breakpara.deadtime         = 0x40;
timer_breakpara.breakpolarity    = TIMER_BREAK_POLARITY_LOW;
timer_breakpara.outputautostate  = TIMER_OUTAUTO_ENABLE;
timer_breakpara.protectmode      = TIMER_CCHP_PROT_OFF;
timer_breakpara.breakstate       = TIMER_BREAK_DISABLE;
timer_break_config(TIMER0,&timer_breakpara);

/* TIMER0 primary output function enable */
timer_primary_output_config(TIMER0,ENABLE);

/* TIMER0 channel control update interrupt enable */
timer_interrupt_enable(TIMER0,TIMER_INT_CMT);
/* TIMER0 break interrupt disable */
timer_interrupt_disable(TIMER0,TIMER_INT_BRK);
```

```
    /* TIMER0 counter enable */
    timer_enable(TIMER0);
    // TODO lock registers
}

void bridge_set_state(bridge_state_e new_state)
{

    switch (new_state) {
    case BRIDGE_STATE_DISABLED:
        bridge_disable();
        break;
    case BRIDGE_STATE_RUN:
        bridge_set_run_duty(0);
        TIMER_PSC(TIMER0) = 1;
        TIMER_CAR(TIMER0) = 2400;
        TIMER_CH3CV(TIMER0) = 1500;
        g_bridge_state = BRIDGE_STATE_RUN;
        break;
    default:

        break;
    }
}

void bridge_enable() {
    timer_primary_output_config(TIMER0,ENABLE); // shouldn't do/be part of f0
aapi?
}

void bridge_disable() {
    timer_primary_output_config(TIMER0,DISABLE); // shouldn't do/be part of f0
aapi?
    bridge_set_run_duty(0);
    g_bridge_state = BRIDGE_STATE_DISABLED;
}



// for use in run mode - duty is always 12bit (0-2047)
void bridge_set_run_duty(uint16_t duty)
{

    if (duty > 1500) {
        duty = 1500; // clamp duty to ~25% max for testing
    }

    if (duty > 0x400) {
        bridge_setup_adc_trigger(0x200);
    } else {
```

```
        bridge_setup_adc_trigger(0x600);
    }

    g_bridge_run_duty = duty;
}

void initPhases(bool* Phases){
    for(int i = 0; i < 6; i++){
        *(Phases + i) = false;
    }
}

void initHalls(bool* Halls){
    *(Halls + 0) = true;
    *(Halls + 1) = true;
    *(Halls + 2) = false;
}

bool checkHallSensorMalfunction(uint8_t hallPosition){
    return ((hallPosition > 6) || (hallPosition <1));
}

void readHallSensors(bool* Halls){
    *(Halls + 0) = gpio_input_bit_get(GPIOA, GPIO_PIN_15);
    *(Halls + 1) = gpio_input_bit_get(GPIOB, GPIO_PIN_3);
    *(Halls + 2) = gpio_input_bit_get(GPIOB, GPIO_PIN_10);
}

void getHallPosition(bool Halls[3], uint8_t* hallPosition){
    (*hallPosition) = (Halls[0]<<2) + (Halls[1]<<1) + (Halls[2]);
}

unsigned char _last_step=0;
commutation_state_t cur_step;
void bridge_commutate() {


        if (cur_step.pinAL){
            gpio_init(GPIOB,GPIO_MODE_OUT_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_13);
            gpio_bit_set(GPIOB, GPIO_PIN_13);
        }else{
            gpio_init(GPIOB,GPIO_MODE_AF_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_13);
        }

        if (cur_step.pinBL){
            gpio_init(GPIOB,GPIO_MODE_OUT_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_14);
            gpio_bit_set(GPIOB, GPIO_PIN_14);
        }else{
            gpio_init(GPIOB,GPIO_MODE_AF_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_14);
        }
```

```
        if (cur_step.pinCL){
            gpio_init(GPIOB,GPIO_MODE_OUT_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_15);
            gpio_bit_set(GPIOB, GPIO_PIN_15);
        }else{
            gpio_init(GPIOB,GPIO_MODE_AF_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_15);
        }


        *(TIM1_CCR1_ADDR) = g_bridge_run_duty;
        *(TIM1_CCR2_ADDR) = g_bridge_run_duty;
        *(TIM1_CCR3_ADDR) = g_bridge_run_duty;

        TIMER_CHCTL2(TIMER0) = cur_step.CCER;

}

void bridge_enable_adc_trigger()
{
  // setup adc synchronization
  //nvic_enable_irq(NVIC_TIM1_CC_IRQ);
  //TIM_DIER(TIM1) |= TIM_DIER_CC4IE;

    nvic_irq_enable(TIMER0_TRG_CMT_TIMER10_IRQn, 0, 1);
    timer_interrupt_disable(TIMER0,TIMER_INT_CH3);
}

void bridge_setup_adc_trigger(uint16_t ticks)
{
  //TIM1_CCR4 = ticks;
  *(TIM1_CCR4_ADDR) = g_bridge_run_duty;
}

float koefA_Motor_Current = 0.136496350364963f;
float koefA_Motor_Voltage = 0.08622904031395f;
float koefA_Logic_Voltage = 0.08674655047204f;
float koefA_Source_Voltage =0.0866308243726f;
float koefA_Current_PhaseB =0.0f;
float koefA_Voltage_10V = 0.003421228304406f;
float koefA_Current_PhaseC =0.0f;
float koefA_Voltage_18V = 0.0862735621073f;
float koefA_BEMF_PhaseA = 0.0f;
float koefA_BEMF_PhaseB = 0.0f;
float koefA_BEMF_PhaseC = 0.0f;
float koefA_BEMF_Central = 0.0f;
float koefA_Sensor_Voltage =0.0884487226953f;
float koefA_Hall_Voltage =0.08723404255319f;
float koefA_CPU_Temperature =0.0f;
//Global heartbeats
uint32_t globalHeartbeat_50us = 0, heartbeat_100us = 0, heartbeat_1ms = 0,
heartbeat_10ms = 0;
int measuredSpeed = 0, demandedSpeed = 0, speedError = 0;
```

```
int demandedPWM = 0, controlOutput = 0; //Duty cycle proportional to the control
uint16_t accelPedalValue_scaled = 0;
bool myPhases[6]; //0 for Phase 1 High, 1 for Phase 1 Low, 2 for Phase 2 High,
etc
bool myHalls[3]; // 0 for Hall 1, 1 for Hall 2, 2 for Hall 3
bool xNewStep;
bool xStartStep;
uint32_t last_nodeID=0;
void BLDC_Task(void *pvParameters) {

    initPhases(myPhases);
    initHalls(myHalls);

    //PID
    uint8_t hallPosition = 0; //Stores the position of the hall sensors
    uint8_t node_id;
    uint8_t motor_cmd;
    uint8_t motor_dir;
    uint8_t current_dir=0;
    uint8_t motor_byte;
    uint8_t motor_bit;
    uint8_t lastHallPosition = 0; //Last position of the hall sensors - used to
compute motor velocity
    int encoder_ticks = 0;

    xStartStep = false;

    timer_config();
    overcurrent_watchdog_initialize();
    bridge_initialize();
    bridge_enable_adc_trigger();
    comparator_initialize();
    adc_initialize();
    adc_start();

    zc_timer_initialize();
    CAN_Start_Task(1);
    int32_t iStep = 1;
    bridge_enable();
    bridge_set_state(BRIDGE_STATE_RUN);
    bridge_set_run_duty(200);
    watchdog_reset();
    start_motor();

    g.Configuration.Networking.dwNodeID = 0;
    g.Configuration.Datasheet.ucDriveDirection = 0;
    memset(&g.Control.xDriveCMD,0x00,8);
    g.Configuration.Datasheet.ucMotorType = 1;
    g.Configuration.Datasheet.rSpeed = 750;
    g.Configuration.Datasheet.ucRampDown = 40;
    g.Configuration.Datasheet.ucRampUp = 100;
```

```
    node_id = getNodeId(&_Data);
    motor_byte = node_id / 8;
    motor_bit = node_id % 8;
    unsigned short adc_cnt=0;
    setState(&_Data, Operational);
    g.Telemetry.System_Status = 0;
    for (;;) {
        for(int ii=0;ii<ADC_IDX_TEMPERATURE;ii++){
            g.adc_summ_buffer[ii]+=g.adc_buffer[ii];//(g.adc_buffer[ii]*10+g.adc_
buffer[ii])/11;

            if (adc_cnt >= 1000) {
                g.adc_filter_buffer[ii]= g.adc_summ_buffer[ii]/1000;
                g.adc_summ_buffer[ii]=0;
                adc_cnt=0;
            }
        }
        adc_cnt++;
        //if(g.adc_buffer[ADC_IDX_BUS_CURRENT]>100)g.adc_filter_buffer[ADC_IDX_BU
S_CURRENT]=g.adc_buffer[ADC_IDX_BUS_CURRENT];
        if(g.Control.xDriveCMD[4]<14){
            g.Telemetry.uiCustomADC =
g.adc_buffer[g.Control.xDriveCMD[4]];//g.adc_buffer[g.Control.xDriveCMD];
        }
        if((g.adc_buffer[ADC_IDX_BUS_CURRENT]*koefA_Motor_Current)>254.0f){
            g.Telemetry.ADC.Motor_Current = 0xff;
        }else{
            g.Telemetry.ADC.Motor_Current =
((g.adc_buffer[ADC_IDX_BUS_CURRENT]*koefA_Motor_Current));
        }

        g.Telemetry.ADC.Motor_Voltage =
((g.adc_buffer[ADC_IDX_MOTOR_VOLTAGE]*koefA_Motor_Voltage))-100;
        g.Telemetry.ADC.Logic_Voltage =
((g.adc_buffer[ADC_IDX_LOGIC_VOLTAGE]*koefA_Logic_Voltage))-100;
        g.Telemetry.ADC.Source_Voltage =
((g.adc_buffer[ADC_IDX_SOURCE_VOLTAGE]*koefA_Source_Voltage))-100;
        g.Telemetry.ADC.Current_PhaseB =
((g.adc_buffer[ADC_IDX_PHASEB_CURRENT]*koefA_Current_PhaseB))-100;
        g.Telemetry.ADC.Sens = ((g.adc_buffer[ADC_IDX_SENS]*koefA_Voltage_10V));
        g.Telemetry.ADC.Aux = ((g.adc_buffer[ADC_IDX_AUX]*koefA_Voltage_10V));
        g.Telemetry.ADC.Current_PhaseC =
((g.adc_buffer[ADC_IDX_PHASEC_CURRENT]*koefA_Current_PhaseC))-100;
        g.Telemetry.ADC.Voltage_18V =
((g.adc_buffer[ADC_IDX_18V_CURRENT]*koefA_Voltage_18V));
        g.Telemetry.ADC.BEMF_PhaseA =
((g.adc_buffer[ADC_IDX_PHASEA_VOLTAGE]*koefA_BEMF_PhaseA))-100;
        g.Telemetry.ADC.BEMF_PhaseB =
((g.adc_buffer[ADC_IDX_PHASEB_VOLTAGE]*koefA_BEMF_PhaseB))-100;
        g.Telemetry.ADC.BEMF_PhaseC =
((g.adc_buffer[ADC_IDX_PHASEC_VOLTAGE]*koefA_BEMF_PhaseC))-100;
```

```
        g.Telemetry.ADC.BEMF_Central =
((g.adc_buffer[ADC_IDX_NEUTRAL_VOLTAGE]*koefA_BEMF_Central))-100;
        g.Telemetry.ADC.Sensor_Voltage =
((g.adc_buffer[ADC_IDX_SENS_VOLTAGE]*koefA_Sensor_Voltage))-100;
        g.Telemetry.ADC.Hall_Voltage =
((g.adc_buffer[ADC_IDX_HALL_VOLTAGE]*koefA_Hall_Voltage))-100;
        g.Telemetry.ADC.CPU_Temperature =
((g.adc_buffer[ADC_IDX_TEMPERATURE]*koefA_CPU_Temperature))-100;

        if(g.adc_buffer[ADC_IDX_MOTOR_VOLTAGE] > 2000){
            g.Telemetry.System_Status &=~0x0200;
            gpio_bit_set(GPIOB, GPIO_PIN_4);
            gpio_bit_reset(GPIOB, GPIO_PIN_5);
        }else{
            g.Telemetry.System_Status |=0x0200;
            gpio_bit_reset(GPIOB, GPIO_PIN_4);
            gpio_bit_set(GPIOB, GPIO_PIN_5);
        }

        if(g.adc_buffer[ADC_IDX_LOGIC_VOLTAGE] > 2000){
            g.Telemetry.System_Status &=~0x0100;
        }else{
            g.Telemetry.System_Status |=0x0100;
        }

        if (g.adc_buffer[ADC_IDX_SENS] > 3000){
            g.Telemetry.System_Status |=0x01;
            gpio_bit_reset(GPIOA, GPIO_PIN_12);
        }else{
            g.Telemetry.System_Status &=~0x01;
            gpio_bit_set(GPIOA, GPIO_PIN_12);
        }
        if(g.adc_buffer[ADC_IDX_AUX] > 3000){
            g.Telemetry.System_Status |=0x02;
            gpio_bit_reset(GPIOC, GPIO_PIN_9);
        }else{
            g.Telemetry.System_Status &=~0x02;
            gpio_bit_set(GPIOC, GPIO_PIN_9);
        }


        CAN_Tx_Task(1);
        CAN_Rx_Task(1);
        if(g.Configuration.Datasheet.rSpeed >
1500)g.Configuration.Datasheet.rSpeed = 1500;
        if(g.Configuration.Datasheet.rSpeed <
100)g.Configuration.Datasheet.rSpeed = 100;
        if(g.Configuration.Datasheet.ucRampDown <
10)g.Configuration.Datasheet.ucRampDown = 10;
        if(g.Configuration.Datasheet.ucRampUp <
10)g.Configuration.Datasheet.ucRampUp = 10;
```

```
        watchdog_reset();
        node_id = getNodeId(&_Data);
        if(node_id != g.Configuration.Networking.dwNodeID) {
            setNodeId(&_Data, (UNS8)g.Configuration.Networking.dwNodeID);
        }
        if(node_id>0){
            motor_byte = (node_id-1) / 8;
            motor_bit = (node_id-1) % 8;
            motor_cmd = (g.Control.xDriveCMD[motor_byte] >> motor_bit)&0x01;
            motor_dir = (g.Control.xDriveCMD[motor_byte+4] >> motor_bit)&0x01;
        }else{
            motor_cmd = 0;
        }
        //g.Configuration.Networking.dwNodeID = 1;
        //g.Configuration.Datasheet.ucMotorType = 2;
        //g.Configuration.Datasheet.rSpeed = 100;
        //g.Configuration.Datasheet.ucRampDown = 40;
        //g.Configuration.Datasheet.ucRampUp = 100;
        //motor_dir = 0;
        //motor_cmd = 1;
        if((current_dir!=motor_dir)&&(g.current_speed<100)){
            current_dir=motor_dir;
        }
        if((current_dir!=motor_dir)&&(motor_cmd!=0)){
            motor_cmd=0;
        }

        if(g.Configuration.Datasheet.ucMotorType < 2){
            if(g.Configuration.Datasheet.ucMotorType == 0){
                readHallSensors(g.Telemetry.Hall_Sensors.Hall_Sensor);
            }


            getHallPosition(g.Telemetry.Hall_Sensors.Hall_Sensor,
&g.Telemetry.Hall_Sensor);
            if((g.Telemetry.Hall_Sensors.Hall_Sensor[0] == false)&&
                (g.Telemetry.Hall_Sensors.Hall_Sensor[1] == false)&&
                (g.Telemetry.Hall_Sensors.Hall_Sensor[2] == false))
                g.Telemetry.Hall_Sensors.Hall_Sensor[0] = true;


            if (g.Telemetry.Hall_Sensor != lastHallPosition) { //Compute motor
speed using hall sensors
                g.hall_err = 0;
                xNewStep = 1;
                encoder_ticks++;
                lastHallPosition = g.Telemetry.Hall_Sensor;
            }
```

```
            if
((!checkHallSensorMalfunction(g.Telemetry.Hall_Sensor))&&((g.Telemetry.System_Sta
tus&0xFF00)==0x00)) //Hall sensors work correctly
            {
                if (current_dir){
                    cur_step = bridge_step_forward[g.Telemetry.Hall_Sensor];
                } else {
                    cur_step = bridge_step_reverse[g.Telemetry.Hall_Sensor];
                }

                if ((xNewStep)&&(g.xDriveOn>0)){
                    xNewStep = false;
                }
                if((g.current_speed>0)||(g.xDriveOn)){
                    g.hall_err++;
                    bridge_commutate();
                }
            } else {
                g.hall_err++;
                xNewStep = 0;
            }

            if (g.xDriveOn!=motor_cmd){
                if (motor_cmd&&((g.Telemetry.System_Status&0xFF00)==0x00))
                {
                    xStartStep = true;
                    xNewStep = true;
                    timer_enable(TIMER0);
                    bridge_commutate();
                }
                g.xDriveOn=motor_cmd;
            }

            bridge_set_run_duty(g.current_speed);
        }else{
            //g.current_speed
            if (current_dir){
                gpio_bit_set(GPIOB, GPIO_PIN_11);
            }else{
                gpio_bit_reset(GPIOB, GPIO_PIN_11);
            }

            if(((g.current_speed>0)||(g.xDriveOn))&&((g.Telemetry.System_Status&0
xFF00)==0x00)){
                dac_data_set(DAC1, DAC_ALIGN_12B_R, (unsigned
short)(g.current_speed*2.73));
            }
            if (g.xDriveOn!=motor_cmd){
                if (motor_cmd)
                {
                    xStartStep = true;
```

```
                }
                g.xDriveOn=motor_cmd;
            }
        }

        if(!gpio_input_bit_get(GPIOB,GPIO_PIN_2)&&g.xDriveOn){
            g.Telemetry.System_Status |=0x0400;
        }
        if(!gpio_input_bit_get(GPIOB,GPIO_PIN_2)&&!g.xDriveOn&&((g.Telemetry.Syst
em_Status&0x0400)!=0x0400)){
            g.Telemetry.System_Status |=0x2000;
            gpio_bit_reset(GPIOC, GPIO_PIN_6);
        }

        if(g.xDriveOn&&((g.Telemetry.System_Status&0xFF00)==0x0000)){
            if(g.hall_err>5000){
                g.hall_err=5000;
                g.Telemetry.System_Status |=0x0800;
            }
        }

        if((g.Telemetry.System_Status&0xFF00) == 0x0000){

            gpio_bit_set(GPIOC, GPIO_PIN_13);
        }else{
            gpio_bit_set(GPIOC, GPIO_PIN_10);
            gpio_bit_reset(GPIOC, GPIO_PIN_13);
        }

        if (g.xDriveOn&&((g.Telemetry.System_Status&0xFF00)==0x0000)){
            g.Telemetry.System_Status |=0x04;
            gpio_bit_reset(GPIOC, GPIO_PIN_10);
        }else{
            g.Telemetry.System_Status &=~0x04;
            gpio_bit_set(GPIOC, GPIO_PIN_10);
        }

        if (!g.xDriveOn){
            g.hall_err=0;
            g.Telemetry.System_Status &=~0x0C00;
        }

        if(((g.current_speed==0)&&(!g.xDriveOn))||((g.Telemetry.System_Status&0xF
F00)!=0x00)) {
                g.hall_err=0;
                g.target_speed=0;
                g.current_speed=0;
                *(TIM1_CCR1_ADDR) = 0;
                *(TIM1_CCR2_ADDR) = 0;
                *(TIM1_CCR3_ADDR) = 0;
                TIMER_CHCTL2(TIMER0) = 0;
```

```
                timer_disable(TIMER0);
                dac_data_set(DAC1, DAC_ALIGN_12B_R, 0);

                gpio_init(GPIOB,GPIO_MODE_OUT_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_13);
                gpio_bit_set(GPIOB, GPIO_PIN_13);
                gpio_init(GPIOB,GPIO_MODE_OUT_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_14);
                gpio_bit_set(GPIOB, GPIO_PIN_14);
                gpio_init(GPIOB,GPIO_MODE_OUT_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_15);
                gpio_bit_set(GPIOB, GPIO_PIN_15);
        }
        //vTaskDelay(10);
    }

}
```

6. Низкоуровневая связь с мотором

```
#include <commutation.h>

#include <bridge.h>
#include <comparator.h>
#include <global.h>

#include "gd32f30x.h"

// commutation and zero cross timers must have the same frequency
const uint32_t commutation_zc_timer_frequency = 2000000;

void commutation_timer_enable_interrupts()
{
  timer_interrupt_enable(COMMUTATION_TIMER,TIMER_INT_CH0);
  timer_interrupt_enable(COMMUTATION_TIMER,TIMER_INT_UP);
  nvic_irq_enable(COMMUTATION_TIMER_IRQ, 0, 1);
}

void zc_timer_enable_interrupts()
{
  timer_interrupt_enable(ZC_TIMER,TIMER_INT_UP);
  nvic_irq_enable(ZC_TIMER_IRQ, 0, 1);
}

void commutation_timer_disable_interrupts()
{
  timer_interrupt_disable(COMMUTATION_TIMER,TIMER_INT_CH0);
```

```
  timer_interrupt_disable(COMMUTATION_TIMER,TIMER_INT_UP);
  nvic_irq_disable(COMMUTATION_TIMER_IRQ);
}

void zc_timer_disable_interrupts()
{
  timer_interrupt_disable(ZC_TIMER,TIMER_INT_UP);
  nvic_irq_disable(ZC_TIMER_IRQ);
}

void stop_motor()
{
  commutation_timer_disable_interrupts();
#if defined(FEEDBACK_COMPARATOR)
  comparator_zc_isr_disable();
  zc_timer_disable_interrupts();
#endif
}

void start_motor()
{

  starting = true;
  timer_disable(COMMUTATION_TIMER);
  timer_counter_value_config(COMMUTATION_TIMER,1);
  timer_autoreload_value_config(COMMUTATION_TIMER,startup_commutation_period_tick
s);
  timer_channel_output_pulse_value_config(COMMUTATION_TIMER, TIMER_CH_0,
startup_commutation_period_ticks/16);
  zc_counter = zc_confirmations_required;

  g_bridge_comm_step = BRIDGE_STEP_AH_BL;

#if defined(FEEDBACK_COMPARATOR)
  comparator_set_state((comp_state_e)g_bridge_comm_step);
#endif

  commutation_timer_enable_interrupts();
  zc_timer_enable_interrupts();
  //bridge_set_run_duty(startup_throttle);
  timer_enable(COMMUTATION_TIMER);
  timer_counter_value_config(ZC_TIMER,startup_commutation_period_ticks/2);
  timer_enable(ZC_TIMER);// enable counter
}

void commutation_timer_initialize()
{
  timer_parameter_struct timer_initpara;
  rcu_periph_clock_enable(COMMUTATION_TIMER_RCC);
  timer_deinit(COMMUTATION_TIMER);
  /* TIMER0 configuration */
```

```
  timer_initpara.prescaler         = (SystemCoreClock /
commutation_zc_timer_frequency) - 1;
  timer_initpara.alignedmode       = TIMER_COUNTER_EDGE;
  timer_initpara.counterdirection  = TIMER_COUNTER_UP;
  timer_initpara.period            = 0xff;
  timer_initpara.clockdivision     = TIMER_CKDIV_DIV1;
  timer_initpara.repetitioncounter = 0;
  timer_init(COMMUTATION_TIMER,&timer_initpara);

}

void zc_timer_initialize()
{
  timer_parameter_struct timer_initpara;
  rcu_periph_clock_enable(ZC_TIMER_RCC);
  timer_deinit(ZC_TIMER);
  /* TIMER0 configuration */
  timer_initpara.prescaler         = 10;//(SystemCoreClock /
commutation_zc_timer_frequency) - 1;
  timer_initpara.alignedmode       = TIMER_COUNTER_EDGE;
  timer_initpara.counterdirection  = TIMER_COUNTER_UP;
  timer_initpara.period            = 0xffff;
  timer_initpara.clockdivision     = TIMER_CKDIV_DIV1;
  timer_initpara.repetitioncounter = 0;
  timer_init(ZC_TIMER,&timer_initpara);


}

void commutation_isr()
{
    bridge_commutate();

#if defined(FEEDBACK_COMPARATOR)
    comparator_zc_isr_disable();
#endif

    zc_counter = zc_confirmations_required; // remove

#if defined(FEEDBACK_COMPARATOR)
    // TODO rotate table to get this right
    comparator_set_state((comp_state_e)(g_bridge_comm_step + 2));
#endif

}
```

7. Глобальные данные

```
#pragma once

// todo move this file out of hal and into application src

#include <inttypes.h>
#include <stdbool.h>
#include "data.h"

#define ZC_TIMER TIMER5
#define ZC_TIMER_RCC RCU_TIMER5
#define ZC_TIMER_IRQ TIMER5_IRQn
#define COMMUTATION_TIMER TIMER6
#define COMMUTATION_TIMER_RCC RCU_TIMER6
#define COMMUTATION_TIMER_IRQ TIMER6_IRQn

#define ADC_NUM_CHANNELS 16
#define FEEDBACK_COMPARATOR

#define ADC_CHANNEL_BUS_CURRENT 0
#define ADC_CHANNEL_MOTOR_VOLTAGE 1
#define ADC_CHANNEL_LOGIC_VOLTAGE 2
#define ADC_CHANNEL_SOURCE_VOLTAGE 3
#define ADC_CHANNEL_PHASEB_CURRENT 4
#define ADC_CHANNEL_AUX 6
#define ADC_CHANNEL_PHASEC_CURRENT 7
#define ADC_CHANNEL_18V_CURRENT 8
#define ADC_CHANNEL_SENS 9
#define ADC_CHANNEL_PHASEA_VOLTAGE 10
#define ADC_CHANNEL_PHASEB_VOLTAGE 11
#define ADC_CHANNEL_PHASEC_VOLTAGE 12
#define ADC_CHANNEL_NEUTRAL_VOLTAGE 13
#define ADC_CHANNEL_SENS_VOLTAGE 14
#define ADC_CHANNEL_HALL_VOLTAGE 15
#define ADC_CHANNEL_TEMPERATURE 16

#define ADC_IDX_BUS_CURRENT 0
#define ADC_IDX_MOTOR_VOLTAGE 1
#define ADC_IDX_LOGIC_VOLTAGE 2
#define ADC_IDX_SOURCE_VOLTAGE 3
#define ADC_IDX_PHASEB_CURRENT 4
#define ADC_IDX_AUX 5
#define ADC_IDX_PHASEC_CURRENT 6
#define ADC_IDX_18V_CURRENT 7
#define ADC_IDX_SENS 8
#define ADC_IDX_PHASEA_VOLTAGE 9
#define ADC_IDX_PHASEB_VOLTAGE 10
#define ADC_IDX_PHASEC_VOLTAGE 11
#define ADC_IDX_NEUTRAL_VOLTAGE 12
```

```
#define ADC_IDX_SENS_VOLTAGE 13
#define ADC_IDX_HALL_VOLTAGE 14
#define ADC_IDX_TEMPERATURE 15

// adc threshold for current channel
#define ADC_WWDG_CURRENT_MAX 200

extern uint8_t adc_channels[ADC_NUM_CHANNELS];

typedef struct {
  UNS8 xDriveCMD[8];
} Direct_Drive_Control_t;

typedef struct {
  UNS8 Device_Type;   /* Mapped at index 0x5000, subindex 0x9F */
  UNS32 Device_Serial_Number;   /* Mapped at index 0x5000, subindex 0xA0 */
  UNS32 Firmware_Serial_Number;   /* Mapped at index 0x5000, subindex 0xA1 */
  REAL32 Sampling_Frequency;    /* Mapped at index 0x5000, subindex 0xA2 */
  UNS16 Inverter_PWM_Period;    /* Mapped at index 0x5000, subindex 0xA3 */
  UNS32 ADC_ISR_Execution_Time;   /* Mapped at index 0x5000, subindex 0xA4 */
  UNS32 Timer_ISR_Execution_Time;   /* Mapped at index 0x5000, subindex 0xA5 */
} Device_Information_t;

typedef struct {
  UNS8 Motor_Voltage;   /* Mapped at index 0x5001, subindex 0x03 */
  UNS8 Logic_Voltage;   /* Mapped at index 0x5001, subindex 0x04 */
  UNS8 Source_Voltage;    /* Mapped at index 0x5001, subindex 0x04 */
  UNS8 Sens;    /* Mapped at index 0x5001, subindex 0x04 */
  UNS8 Aux;   /* Mapped at index 0x5001, subindex 0x04 */
  UNS8 Voltage_18V;   /* Mapped at index 0x5001, subindex 0x05 */
  UNS8 BEMF_PhaseA;   /* Mapped at index 0x5001, subindex 0x06 */
  UNS8 BEMF_PhaseB;   /* Mapped at index 0x5001, subindex 0x07 */
  UNS8 BEMF_PhaseC;   /* Mapped at index 0x5001, subindex 0x09 */
  UNS8 BEMF_Central;    /* Mapped at index 0x5001, subindex 0x0A */
  UNS8 Sensor_Voltage;    /* Mapped at index 0x5001, subindex 0x0C */
  UNS8 Hall_Voltage;    /* Mapped at index 0x5001, subindex 0x0C */
  UNS8 Motor_Current;   /* Mapped at index 0x5001, subindex 0x0C */
  UNS8 Current_PhaseB;    /* Mapped at index 0x5001, subindex 0x0C */
  UNS8 Current_PhaseC;    /* Mapped at index 0x5001, subindex 0x0C */
  UNS8 CPU_Temperature;   /* Mapped at index 0x5001, subindex 0x0C */
} ADC_t;

typedef struct {
  UNS8 Hall_Sensors_Fault;    /* Mapped at index 0x5005, subindex 0x02 */
  bool Hall_Sensor[3];    /* Mapped at index 0x5005, subindex 0x03 */
} Hall_Sensors_t;

typedef struct {
} Peripheral_t;
```

```
typedef struct {
  UNS8 ucMotorType;   /* Mapped at index 0x2000, subindex 0x01 */
  UNS8 ucDriveDirection;    /* Mapped at index 0x2000, subindex 0x02 */
  REAL32 rSpeed;    /* Mapped at index 0x2000, subindex 0x04 */
  UNS8 ucRampUp;    /* Mapped at index 0x2000, subindex 0x05 */
  UNS8 ucRampDown;    /* Mapped at index 0x2000, subindex 0x06 */
  REAL32 OvercurrentLimit;    /* Mapped at index 0x2000, subindex 0x07 */
} CDatasheet_t;

typedef struct {
  UNS16 usBitrate;    /* Mapped at index 0x3000, subindex 0x02 */
  UNS8 dwNodeID;    /* Mapped at index 0x3000, subindex 0x03 */
  REAL32 rHeartbeatTimeout;   /* Mapped at index 0x3000, subindex 0x04 */
  REAL32 rTelemetryFrequency;   /* Mapped at index 0x3000, subindex 0x05 */
} CNetworking_t;

typedef struct {
  CDatasheet_t Datasheet;
  CNetworking_t Networking;
} Configuration_t;

typedef struct {
  uint16_t uiSelectADC;
  uint16_t uiCustomADC;
  UNS8 Hall_Sensor;   /* Mapped at index 0x4000, subindex 0x03 */
  ADC_t ADC;
  Hall_Sensors_t Hall_Sensors;
  Device_Information_t ident;
  uint16_t System_Status;
} Telemetry_t;

typedef struct {
  uint16_t adc_buffer[ADC_NUM_CHANNELS];
  uint16_t adc_filter_buffer[ADC_NUM_CHANNELS];
  uint32_t adc_summ_buffer[ADC_NUM_CHANNELS];
  int16_t step;
  uint16_t hall_err;
  bool xDriveOn;
  int16_t target_speed;
  int16_t current_speed;
  // if pwm signal is present at boot, it will be selected as the throttle signal
  // source and throttle commands from the serial interface will be ignored
  // if only serial is present, it will be selected and pwm input will be
disabled.
  Direct_Drive_Control_t Control;
  Telemetry_t Telemetry;
  Configuration_t Configuration;
} global_t;

extern global_t g;
```

```
// the zero crosses required in closed loop mode, this is variable depending on
current speed
extern uint16_t run_zc_confirmations_required;

// if true, we are in open-loop
// we wait for the first zero cross period (2 sequential valid zero crosses)
// timing, then we will enter closed loop
extern volatile bool starting;

extern const uint16_t slow_run_zc_confirmations_required;

// open loop startup commutation timer ARR value
// TODO do this in human-readable time (microseconds)
extern const uint16_t startup_commutation_period_ticks;

// the comparator output must hold for this many checks in a row before we
consider it a valid zero-cross
// this can be descreased as rpm increases
extern const uint16_t startup_zc_confirmations_required;

// REMAINING zero cross checks before we pass
extern volatile uint32_t zc_counter;

// the zero cross confirmations currently required to pass a zero cross check
extern uint32_t zc_confirmations_required;
```

8. Защита по току

```
#include <overcurrent-watchdog.h>
#include <bridge.h>
#include <commutation.h>
#include <global.h>
#include <watchdog.h>
#include "gd32f30x.h"
#define BOARD_ADC_CHANNEL   ADC_CHANNEL_0
#define ADC_WATCHDOG_HT     0x0A00
#define ADC_WATCHDOG_LT     0x0400

void overcurrent_watchdog_initialize()
{
#if defined(ADC_CHANNEL_BUS_CURRENT)

  /* ADC analog watchdog threshold config */
  adc_watchdog_threshold_config(ADC1, ADC_WATCHDOG_LT, ADC_WATCHDOG_HT);
  /* ADC analog watchdog single channel config */
  adc_watchdog_single_channel_enable(ADC1, ADC_CHANNEL_BUS_CURRENT);
```

```
  /* ADC interrupt config */
  adc_interrupt_enable(ADC1, ADC_INT_WDE);
  nvic_irq_enable(ADC0_1_IRQn, 0, 1);
#endif
}

void overcurrent_watchdog_isr()
{
  bridge_disable();
  stop_motor();
  while (1) { watchdog_reset(); }
}
```

9. Сторожевой таймер

```
#include <watchdog.h>
#include "gd32f30x.h"

void watchdog_start(uint32_t period_ms)
{
    fwdgt_config(period_ms*5, FWDGT_PSC_DIV8);
    fwdgt_enable();
}

void watchdog_reset()
{
    fwdgt_counter_reload();
}
```

10. Инициализация АЦП

```
#include <adc.h>


#include "gd32f30x.h"
#include <global.h>
#include "FreeRTOS.h" // для отладки
```

```c
#include "cmsis_os.h" // для отладки

void adc_initialize()
{
    /* ADC_DMA_channel configuration */
    dma_parameter_struct dma_data_parameter;

    /* ADC DMA_channel configuration */
    dma_deinit(DMA0, DMA_CH0);

    /* initialize DMA single data mode */
    dma_data_parameter.periph_addr  = (uint32_t)(&ADC_RDATA(ADC0));
    dma_data_parameter.periph_inc   = DMA_PERIPH_INCREASE_DISABLE;
    dma_data_parameter.memory_addr  = (uint32_t)&g.adc_buffer;
    dma_data_parameter.memory_inc   = DMA_MEMORY_INCREASE_ENABLE;
    dma_data_parameter.periph_width = DMA_PERIPHERAL_WIDTH_16BIT;
    dma_data_parameter.memory_width = DMA_MEMORY_WIDTH_16BIT;
    dma_data_parameter.direction    = DMA_PERIPHERAL_TO_MEMORY;
    dma_data_parameter.number       = sizeof(adc_channels);
    dma_data_parameter.priority     = DMA_PRIORITY_HIGH;
    dma_init(DMA0, DMA_CH0, &dma_data_parameter);
    dma_circulation_enable(DMA0, DMA_CH0);
    /* enable DMA channel */
    dma_channel_enable(DMA0, DMA_CH0);
    dma_interrupt_enable(DMA0,DMA_CH0,DMA_INT_FTF);
    /* ADC mode config */
    adc_mode_config(ADC_MODE_FREE);
    /* ADC continuous function enable */
    adc_special_function_config(ADC0, ADC_CONTINUOUS_MODE, ENABLE);
    /* ADC scan function enable */
    adc_special_function_config(ADC0, ADC_SCAN_MODE, ENABLE);
    /* ADC data alignment config */
    adc_data_alignment_config(ADC0, ADC_DATAALIGN_RIGHT);

    /* ADC channel length config */
    adc_channel_length_config(ADC0, ADC_REGULAR_CHANNEL, sizeof(adc_channels));

    /* ADC regular channel config */
    adc_regular_channel_config(ADC0, 0, ADC_CHANNEL_0, ADC_SAMPLETIME_28POINT5);
    adc_regular_channel_config(ADC0, 1, ADC_CHANNEL_1, ADC_SAMPLETIME_28POINT5);
    adc_regular_channel_config(ADC0, 2, ADC_CHANNEL_2, ADC_SAMPLETIME_28POINT5);
    adc_regular_channel_config(ADC0, 3, ADC_CHANNEL_3, ADC_SAMPLETIME_28POINT5);
    adc_regular_channel_config(ADC0, 4, ADC_CHANNEL_4, ADC_SAMPLETIME_28POINT5);
    adc_regular_channel_config(ADC0, 5, ADC_CHANNEL_6, ADC_SAMPLETIME_28POINT5);
    adc_regular_channel_config(ADC0, 6, ADC_CHANNEL_7, ADC_SAMPLETIME_28POINT5);
    adc_regular_channel_config(ADC0, 7, ADC_CHANNEL_8, ADC_SAMPLETIME_28POINT5);
    adc_regular_channel_config(ADC0, 8, ADC_CHANNEL_9, ADC_SAMPLETIME_28POINT5);
    adc_regular_channel_config(ADC0, 9, ADC_CHANNEL_10, ADC_SAMPLETIME_28POINT5);
    adc_regular_channel_config(ADC0, 10, ADC_CHANNEL_11,
ADC_SAMPLETIME_28POINT5);
```

```
    adc_regular_channel_config(ADC0, 11, ADC_CHANNEL_12,
ADC_SAMPLETIME_28POINT5);
    adc_regular_channel_config(ADC0, 12, ADC_CHANNEL_13,
ADC_SAMPLETIME_28POINT5);
    adc_regular_channel_config(ADC0, 13, ADC_CHANNEL_14,
ADC_SAMPLETIME_28POINT5);
    adc_regular_channel_config(ADC0, 14, ADC_CHANNEL_15,
ADC_SAMPLETIME_28POINT5);
    adc_regular_channel_config(ADC0, 15, ADC_CHANNEL_16,
ADC_SAMPLETIME_28POINT5);

    /* ADC trigger config */
    adc_external_trigger_source_config(ADC0, ADC_REGULAR_CHANNEL,
ADC0_1_2_EXTTRIG_REGULAR_NONE);
    adc_external_trigger_config(ADC0, ADC_REGULAR_CHANNEL, ENABLE);

    /* ADC DMA function enable */
    adc_dma_mode_enable(ADC0);
    /* enable ADC interface */
    adc_enable(ADC0);
    //vTaskDelay(10);
    /* ADC calibration and reset calibration */
    adc_calibration_enable(ADC0);

    /* ADC software trigger enable */
    //adc_software_trigger_enable(ADC0, ADC_REGULAR_CHANNEL);


}

void adc_start()
{
    adc_software_trigger_enable(ADC0, ADC_REGULAR_CHANNEL);
}

volatile uint16_t adc_read_channel(uint8_t channel)
{
    return g.adc_buffer[channel];
}


void ADC_Task(void *pvParameters) {
  adc_initialize();
  adc_start();
   /*  */
  for (;;) {
    vTaskDelay(10);
  }
}
```

11. Драйвер CAN

```
#include "mycan_driver.h"
#include "FreeRTOS.h" // для отладки
#include "cmsis_os.h" // для отладки
#include "gd32f30x.h"

#include "global.h"
#include "Slave.h"
#include "data.h" // для callback NMT

/*-----------------------------------------------------------------------*/
static xQueueHandle xCANSendQueue = NULL; //
static xQueueHandle xCANRcvQueue  = NULL; //

TaskHandle_t *CAN_Tx_Task_Header;
TaskHandle_t *CAN_Rx_Task_Header;


uint8_t CAN_Tx_Task_stack[CAN_TX_STACK_SIZE];
uint8_t CAN_Tx_Task_block[100]; //must be >= sizeof(StaticTask_t)
osThreadId_t CAN_Tx_Task_thread;

uint8_t CAN_Rx_Task_stack[CAN_RX_STACK_SIZE];
uint8_t CAN_Rx_Task_block[100]; //must be >= sizeof(StaticTask_t)
osThreadId_t CAN_Rx_Task_thread;


can_receive_message_struct receive_message;
can_trasnmit_message_struct transmit_message;

static TIMEVAL last_counter_val = 0;
static TIMEVAL elapsed_time     = 0;

/*  дельта текущего ID со словарем  и старый ID для отслеживания его смены и
перезагрузки фильтра */
int8_t can_id_delta;
int8_t old_can_id;

// // структуры в соответствии с HAL
// extern TIM_HandleTypeDef hCANOPEN_TIMx;

/*
 ************************************************************************
 *         CAN_Start_Task
 * @brief    Task determinate on CubeMX. Task is initialization can Driver.
 *           After initialization can Driver task deleted. Task calling from
```

```
 *           freertos.c file.
 * @param    - *
 * @return   - none
 * @author   LVA
 ******************************************************************************
 */
void CAN_Start_Task(void *argument) {

  CAN_Configuration();
  //HAL_TIM_Base_Start_IT(&hCAN_TIMx); // включаем таймер  для CAN в режиме
прерывания.

  CAN_Init_Driver();
  CAN_Start_App();

  // DEBUG_NAME_FUNCTION;

 // extern osThreadId_t CAN_TaskHandle;
 // vTaskDelete(CAN_TaskHandle); // Мавр сделал свое дело, его можно убить.
}

/*
 ******************************************************************************
 *         CAN_Configuration
 * @brief   Staring configuratins Can environment.
 *          - initialisation Can_id variables (can_id_delta and old_can_id)
 *          - Setting Canfilter
 *          - Staring CAN on microcontroller
 *          - enable interups
 * @param   - none
 * @return  - none
 * @author  LVA
 ******************************************************************************
 */
void CAN_Configuration(void) {

  // extern uint8_t Slave_bDeviceNodeId; // from Slave.c
  extern uint8_t can_id; // from Slave.c

  /* initialisation Can_id variables */
//  if (can_id > 127) {
//    can_id = Slave_bDeviceNodeId;
//  }
//  can_id_delta = can_id - Slave_bDeviceNodeId;
//  old_can_id   = can_id;
//
//  /* Setting Canfilter */
//  CAN_SetFilter(&_hcan, can_id); /*настраиваем фильтр */
//
//  HAL_CAN_Start(&_hcan); // start CAN
//
```

```
can_parameter_struct            can_parameter;
can_filter_parameter_struct     can_filter;

can_struct_para_init(CAN_INIT_STRUCT, &can_parameter);
can_struct_para_init(CAN_FILTER_STRUCT, &can_filter);

rcu_periph_clock_enable(RCU_GPIOA);
rcu_periph_clock_enable(RCU_GPIOB);
rcu_periph_clock_enable(RCU_GPIOC);
rcu_periph_clock_enable(RCU_GPIOD);
rcu_periph_clock_enable(RCU_AF);
rcu_periph_clock_enable(RCU_DAC);
rcu_periph_clock_enable(RCU_CAN0);
rcu_periph_clock_enable(RCU_DMA0);
gpio_init(GPIOB,GPIO_MODE_IPU,GPIO_OSPEED_50MHZ,GPIO_PIN_8);
gpio_init(GPIOB,GPIO_MODE_AF_PP,GPIO_OSPEED_50MHZ,GPIO_PIN_9);
gpio_pin_remap_config(GPIO_CAN_PARTIAL_REMAP,ENABLE);
/* initialize CAN register */
can_deinit(CAN0);

/* initialize CAN */
can_parameter.time_triggered = DISABLE;
can_parameter.auto_bus_off_recovery = DISABLE;
can_parameter.auto_wake_up = DISABLE;
can_parameter.no_auto_retrans = ENABLE;
can_parameter.rec_fifo_overwrite = DISABLE;
can_parameter.trans_fifo_order = DISABLE;
can_parameter.working_mode = CAN_NORMAL_MODE;
can_parameter.resync_jump_width = CAN_BT_SJW_1TQ;
can_parameter.time_segment_1 = CAN_BT_BS1_5TQ;
can_parameter.time_segment_2 = CAN_BT_BS2_4TQ;
/* baudrate 1Mbps */
can_parameter.prescaler = 6;
can_init(CAN0, &can_parameter);
can_debug_freeze_disable(CAN0);
can_working_mode_set(CAN0,CAN_MODE_NORMAL);
/* initialize filter */
can_filter.filter_number = 0;

/* initialize filter */
can_filter.filter_mode = CAN_FILTERMODE_MASK;
can_filter.filter_bits = CAN_FILTERBITS_32BIT;
can_filter.filter_list_high = 0x0000;
can_filter.filter_list_low = 0x0000;
can_filter.filter_mask_high = 0x0000;
can_filter.filter_mask_low = 0x0000;
can_filter.filter_fifo_number = CAN_FIFO1;
can_filter.filter_enable = ENABLE;
can_filter_init(&can_filter);

can_struct_para_init(CAN_TX_MESSAGE_STRUCT, &transmit_message);
```

```
    can_struct_para_init(CAN_RX_MESSAGE_STRUCT, &receive_message);
    /* enable CAN receive FIFO1 not empty interrupt */
    can_interrupt_enable(CAN0, CAN_INT_RFNE1);
    can_interrupt_enable(CAN0, CAN_INT_RFNE0);



  return;
}

/**
 ****************************************************************************
 *     CAN_Init_Driver
 *  @brief  create Tx and Rx queues create Tx and Rx tasks
 * @author  SWUST Tom
 ****************************************************************************
 */
/*-------------------------------------------------------------------------*/

void CAN_Init_Driver(void) {
  uint8_t Errors = TRUE;
  while (Errors) {
    /* Queues created */
    Errors = FALSE;
    if (xCANSendQueue == NULL) {
      xCANSendQueue = xQueueCreate(CAN_TX_QUEUE_LEN, sizeof(CanTxMsgTypeDef));
      if (xCANSendQueue == NULL) {
        Errors = TRUE;
      }
    }
    if (xCANRcvQueue == NULL) {
      xCANRcvQueue = xQueueCreate(CAN_RX_QUEUE_LEN, sizeof(CanRxMsgTypeDef));
      if (xCANRcvQueue == NULL) {
        Errors = TRUE;
      }
    }

    /* Tasks created */
   /* const osThreadAttr_t CAN_Tx_TaskAttr = {
        .cb_mem = CAN_Tx_Task_block,
        .cb_size = sizeof(CAN_Tx_Task_block),
        .name = "CAN_Tx_Task",
        .stack_mem = CAN_Tx_Task_stack,
        .stack_size = sizeof(CAN_Tx_Task_stack)
    };
    CAN_Tx_Task_thread = osThreadNew(CAN_Tx_Task, NULL, &CAN_Tx_TaskAttr);

    const osThreadAttr_t CAN_Rx_TaskAttr = {
        .cb_mem = CAN_Rx_Task_block,
        .cb_size = sizeof(CAN_Rx_Task_block),
        .name = "CAN_Rx_Task",
```

```
        .stack_mem = CAN_Rx_Task_stack,
        .stack_size = sizeof(CAN_Rx_Task_stack)
    };
    CAN_Rx_Task_thread = osThreadNew(CAN_Rx_Task, NULL, &CAN_Rx_TaskAttr);*/
  }
}

/*
 ******************************************************************************
 *         CAN_Start_App
 * @brief   Staring Canfestival application code.
 *          - initialisation callbask function
 *          - initialisation Canfestival function
 *          - Staring CAN on microcontroller
 * @param   - none
 * @return  - none
 * @author  LVA
 ******************************************************************************
 */
void CAN_Start_App(void) {

  /* register the callbacks for event to other code */
  //RegisterSetODentryCallBack(&_Data, 0x2000, 0, callback_on_position); //
can_id
  //RegisterSetODentryCallBack(&_Data, 0x2004, 0, callback_on_position); // Dac
  _Data.NMT_Slave_Node_Reset_Callback = NMT_Slave_Node_Reset_Callback_Function;

  setNodeId(&_Data, (UNS8)g.Configuration.Networking.dwNodeID);
  setState(&_Data, Initialisation);
  //setState(&_Data, Stopped);
  //setState(&_Data, Operational);
  MX_TIM2_Init();
}

/**
 ******************************************************************************
 *         CAN_SetFilter
 * @brief Initialization hardware Rx Can filter
 * @param  - CAN handle Structure definition
 *         - Can Id
 * @return - standart HAL error code
 * @author     LVA
 ******************************************************************************
 */
void CAN_SetFilter(uint8_t _can_id) {

#define CAN_FILTERS 1

    can_filter_parameter_struct     can_filter;

    can_struct_para_init(CAN_FILTER_STRUCT, &can_filter);
```

```
//  CAN_FilterTypeDef CAN_FilterConfig[CAN_FILTERS] = {
//      {.FilterIdHigh         = (_can_id << 5),
//       .FilterIdLow          = (0x00 << 5), // NMT messages
//       .FilterMaskIdHigh     = (0x7F << 5),
//       .FilterMaskIdLow      = (0x7F << 5),
//       .FilterFIFOAssignment = 0,
//       .FilterBank           = 1,
//       .FilterMode           = CAN_FILTERMODE_IDMASK,
//       .FilterActivation     = ENABLE,
//       .SlaveStartFilterBank = 1},
//
//  };
    /* initialize filter */
    can_filter.filter_number = 0;

    /* initialize filter */
    can_filter.filter_mode = CAN_FILTERMODE_MASK;
    can_filter.filter_bits = CAN_FILTERBITS_32BIT;
    can_filter.filter_list_high = 0x0000;
    can_filter.filter_list_low = 0x0000;
    can_filter.filter_mask_high = (0x7F << 5);
    can_filter.filter_mask_low = (0x7F << 5);
    can_filter.filter_fifo_number = CAN_FIFO1;
    can_filter.filter_enable = ENABLE;
    can_filter_init(&can_filter);

//  if (HAL_CAN_ConfigFilter(_hcan, &CAN_FilterConfig[0]) != HAL_OK) { //
configure main CAN filter
//    can_debug_printf("HAL_CAN_ConfigFilter0 ERROR\n");
//    return HAL_ERROR;
//  }

  return;
}

/**
 ******************************************************************************
 *      CAN_Tx_Task
 * @brief   Read TX queue and Tranmit date to CAN Bus
 * pvParameters - must delete?
 * @author  SWUST Tom updated LVA
 ******************************************************************************
 */
CanTxMsgTypeDef tx_buff[10];
uint16_t tx_buffcount;
uint16_t send_tx_buffcount;
void CAN_Tx_Task(void *pvParameters) {
  static CanTxMsgTypeDef TxMsg;
  static CanTxMsgTypeDef cleanTxMsg;
  cleanTxMsg.StdId = 0;
```

```
  cleanTxMsg.Data[0] = 0;
  cleanTxMsg.Data[1] = 0;
  cleanTxMsg.Data[2] = 0;
  cleanTxMsg.Data[3] = 0;
  cleanTxMsg.Data[4] = 0;
  cleanTxMsg.Data[5] = 0;
  cleanTxMsg.Data[6] = 0;
  cleanTxMsg.Data[7] = 0;
  cleanTxMsg.DLC = 0;
  /*  */
  //for (;;)
  {
    if ((can_flag_get(CAN0,CAN_FLAG_TME0))&&
        (can_flag_get(CAN0,CAN_FLAG_TME1))&&
        (can_flag_get(CAN0,CAN_FLAG_TME2))&&
        (!can_flag_get(CAN0,CAN_FLAG_TS))) {
      if (tx_buffcount!=send_tx_buffcount) {
        TxMsg = tx_buff[send_tx_buffcount];
        tx_buff[send_tx_buffcount++] = cleanTxMsg;
        CAN_Transmit(&TxMsg);
        if(send_tx_buffcount>=9)send_tx_buffcount=0;
      }
    }
  }
}

/**
 ****************************************************************************
 *     CANRcv_Task
 * @brief Main Task Theare work Canfestival code.
 *        Received date from queue an generation other events
 * pvParameters  - must delete ?
 *
 * @author  SWUST Tom. LVA Editing for HAL support
 ****************************************************************************
 */
CanRxMsgTypeDef rx_buff[10];
uint16_t rx_buffcount;
uint16_t rcv_rx_buffcount;
void CAN_Rx_Task(void *pvParameters) {
  static CanRxMsgTypeDef RxMsg;
  static Message msg;

  uint8_t i = 0;

      if (!gpio_input_bit_get(GPIOD,GPIO_PIN_0)&& getNodeId(&_Data) > 0 ){
        gpio_bit_set(GPIOD,GPIO_PIN_1);//signal
      }else{

        gpio_bit_reset(GPIOD,GPIO_PIN_1);//signal
      }
```

```
    if(getNodeId(&_Data) > 0){
      gpio_bit_set(GPIOC, GPIO_PIN_12);
      gpio_bit_reset(GPIOD, GPIO_PIN_2);  //led
    }else{
      gpio_bit_reset(GPIOC, GPIO_PIN_12);
      gpio_bit_set(GPIOD, GPIO_PIN_2);  //led
    }

  //for (;;)
  {
    if (rx_buffcount!=rcv_rx_buffcount) {
        RxMsg = rx_buff[rcv_rx_buffcount++];
        if(rcv_rx_buffcount>=10)rcv_rx_buffcount=0;

      /*
      т.к. Canfestival не позволяет менять адрес устройства (во всяком случае,
       у меня не получилось) делаем это в драйвере, необходимо чтобы внутри
       функций Canfestival  cob_id всегда =0x00
      */

      if (RxMsg.StdId == 0x00) {
        if (RxMsg.Data[1] == 0x00) {
          ; /* ничего не далаем это широковещательная команда */
        } else if (RxMsg.Data[1] == can_id_delta + (uint8_t)*_Data.bDeviceNodeId)
{
          RxMsg.Data[1] = RxMsg.Data[1] - can_id_delta; /* подменям ID в NMT
сообщении т.к оно персональное */
        } else {
          RxMsg.Data[0] = 0xFF; /* делаем сообщение невалидным чтобы не было
ошибочной отработки*/
        }
        msg.cob_id = 0;
      } else if (RxMsg.StdId == 0x0600){
          if ((RxMsg.Data[1] == 0x00)&&(RxMsg.Data[2] == 0x30)&&(RxMsg.Data[3] ==
0x01)) {
            msg.cob_id = RxMsg.StdId + g.Configuration.Networking.dwNodeID;
          }
      } else if (RxMsg.StdId == 0x0300){
        msg.cob_id = RxMsg.StdId + g.Configuration.Networking.dwNodeID;
      } else msg.cob_id = RxMsg.StdId;

      if (CAN_FT_REMOTE == RxMsg.RTR)
        msg.rtr = 1;
      else
        msg.rtr = 0;

      msg.len = (UNS8)RxMsg.DLC;
      if (getNodeId(&_Data) == 0 ){
        if(gpio_input_bit_get(GPIOD,GPIO_PIN_0))
          return;
```

```
      }else{
        if(!gpio_input_bit_get(GPIOD,GPIO_PIN_0))
          return;
      }
      for (i        = 0; i < RxMsg.DLC; i++)
        msg.data[i] = RxMsg.Data[i];
//      __HAL_TIM_DISABLE_IT(&hCAN_TIMx, TIM_IT_UPDATE);
      canDispatch(&_Data, &msg);
//      __HAL_TIM_ENABLE_IT(&hCAN_TIMx, TIM_IT_UPDATE);

    }
  }
}

/**
 ********************************************************************************
 * canSend
 * @brief  function mediator between Canfestival code and other code
 *         Received date from Canfestival code and insert it to Tx queue
 * @author  SWUST Tom
 ********************************************************************************
 */
unsigned char canSend(CAN_PORT notused, Message *m) {
  uint8_t i;
  static CanTxMsgTypeDef TxMsg;
  portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;
  /*
  т.к. Canfestival не позволяет менять адрес устройства делаем это в драйвере,
  таким образом внутри функций Canfestival  cob_id всегда =0x00
  */
  TxMsg.StdId = m->cob_id;

  if (m->rtr)
    TxMsg.RTR = CAN_FT_REMOTE;
  else
    TxMsg.RTR = CAN_FT_DATA;

  TxMsg.DLC = m->len;
  for (i          = 0; i < m->len; i++)
    TxMsg.Data[i] = m->data[i];

  /*  */
  tx_buff[tx_buffcount++] = TxMsg;
  if(tx_buffcount>=9)tx_buffcount=0;
  /*
  if (0 == __get_CONTROL()) {
    if (xQueueSendFromISR(xCANSendQueue, &TxMsg, &xHigherPriorityTaskWoken) !=
pdPASS) {
      return 0xFF;
    }
    portEND_SWITCHING_ISR(xHigherPriorityTaskWoken);
```

```
  } else {
    if (xQueueSend(xCANSendQueue, &TxMsg, 100) != pdPASS) {
      return 0xFF;
    }
  }*/
  return 0;
}

/*
 ******************************************************************************
 *     CAN_Transmit
 * @brief Received date in short Tx structure from CAN_Tx_Task convert it to
 * long TX HAL structure and sent to bus.
 * @param  - CAN handle Structure definition
 *         - short Tx structure
 * @return - standart HAL error code
 * @author     LVA
 ******************************************************************************
 */
uint32_t TxMailbox;
uint8_t CAN_Transmit(CanTxMsgTypeDef *TxMsg) {
  /* объявляем структуру в соответствии с HAL */
  /*CAN_TxHeaderTypeDef pTXHeader;
  pTXHeader.DLC                = (uint32_t)TxMsg->DLC;
  pTXHeader.ExtId              = 0U;
  pTXHeader.IDE                = CAN_ID_STD;
  pTXHeader.RTR                = (uint32_t)TxMsg->RTR;
  pTXHeader.StdId              = (uint32_t)TxMsg->StdId;
  pTXHeader.TransmitGlobalTime = DISABLE;

  if (HAL_CAN_AddTxMessage(_hcan, &pTXHeader, TxMsg->Data, &TxMailbox) != HAL_OK)
{
    can_debug_printf("%s -> HAL_CAN_AddTxMessage HAL_ERROR\n", __FUNCTION__);
    can_debug_printf("TxMailbox:%d\t", TxMailbox);
    // uint32_t transmitmailbox = (READ_REG(hcan->Instance->TSR) & CAN_TSR_CODE)
>> CAN_TSR_CODE_Pos;
    // читаем состояние ящиков из регистров
    can_debug_printf("transmitmailbox:%d\t%d|%d|%d = %d\n",
                     ((READ_REG(_hcan->Instance->TSR) & CAN_TSR_CODE) >>
CAN_TSR_CODE_Pos),
                     ((READ_REG(_hcan->Instance->TSR) & CAN_TSR_TME0) >>
CAN_TSR_TME0_Pos),
                     ((READ_REG(_hcan->Instance->TSR) & CAN_TSR_TME1) >>
CAN_TSR_TME1_Pos),
                     ((READ_REG(_hcan->Instance->TSR) & CAN_TSR_TME2) >>
CAN_TSR_TME2_Pos),
                     ((READ_REG(_hcan->Instance->TSR) & CAN_TSR_TME) >>
CAN_TSR_TME_Pos));
    osDelay(1000); // для отладки
    return HAL_ERROR;
  }
```

```
  can_debug_printf("StdId:0x%x\tRTR:0x%x\tDLC:0x%x\tData0x:%02x %02x %02x %02x
%02x %02x %02x %02x\n", pTXHeader.StdId,
                   pTXHeader.RTR, pTXHeader.DLC, TxMsg->Data[0], TxMsg->Data[1],
TxMsg->Data[2], TxMsg->Data[3],
                   TxMsg->Data[4], TxMsg->Data[5], TxMsg->Data[6], TxMsg-
>Data[7]);
  can_tasks_debug_printf("TX Task Free Stack:%d\n",
                         uxTaskGetStackHighWaterMark(xTaskGetCurrentTaskHandle())
);
  // Debug_view_int(TxMailbox);
*/
    transmit_message.tx_sfid = (uint32_t)TxMsg->StdId;
    transmit_message.tx_efid = 0x00;
    transmit_message.tx_ft = (uint32_t)TxMsg->RTR;
    transmit_message.tx_ff = CAN_FF_STANDARD;
    transmit_message.tx_dlen = (uint32_t)TxMsg->DLC;
    memcpy(transmit_message.tx_data, TxMsg->Data,8);
    can_message_transmit(CAN0, &transmit_message);



  return 0;
}

/*
 ********************************************************************************
 *     CANRcv_DateFromISR
 * @brief   Received date from Can bus over ISR and штыке it in queue
 *  RxMsg   short RX structure
 *
 * @author  SWUST Tom
 ********************************************************************************
 */
void CAN_Rcv_DateFromISR(CanRxMsgTypeDef *RxMsg) {
  /*static portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;

  if (NULL != xCANRcvQueue) {
    if (xQueueSendToBackFromISR(xCANRcvQueue, RxMsg, &xHigherPriorityTaskWoken)
!= pdPASS) {
      //can_debug_printf("Insert Date to Rx QuxQueue is not OK\n");
    }
    portEND_SWITCHING_ISR(xHigherPriorityTaskWoken);
  }*/
  if(rx_buffcount>=9)rx_buffcount=0;
  rx_buff[rx_buffcount++] = *RxMsg;

}

/*
 ********************************************************************************
 *     setTimer
```

```
 * @brief   Set the timer for the next alarm.
 * @author  SWUST Tom. LVA Editing for HAL support
 ************************************************************************
 */
void setTimer(TIMEVAL value) {
  uint32_t timer = timer_counter_read(TIMER7); // Copy the value of the running
timer
  timer_autoreload_value_config(TIMER7,100);
  //elapsed_time += timer - last_counter_val;
  last_counter_val = value;
}

/*
 ************************************************************************
 *     getElapsedTime
 * @brief Return the elapsed time to tell the Stack how much time is spent
 *        since last call.
 * @author  SWUST Tom. LVA Editing for HAL support
 ************************************************************************
 */
TIMEVAL getElapsedTime(void) {
  uint32_t timer = timer_counter_read(TIMER7); // Copy the value of the running
timer
  if (timer < last_counter_val)
    timer += CAN_OPEN_TIM_PERIOD;
  TIMEVAL elapsed = timer - last_counter_val;// + elapsed_time;
  return 0;
}

/*
 ************************************************************************
 * TIMx_DispatchFromISR
 *
 *  @author SWUST Tom
 ************************************************************************
 */
void TIMx_DispatchFromISR(void) {
  last_counter_val = 0;
  elapsed_time     = 0;
  TimeDispatch();
}

/*
 ************************************************************************
 *         NMT_Slave_Node_Reset_Callback_Function
 * @brief Callback for NMT messages node reset. Reload microcontroller.
 *
 * @param  - Object dictionary structure from Canfestival
 * @return - none
 * @author     LVA
 ************************************************************************
```

```
 */
void NMT_Slave_Node_Reset_Callback_Function(CO_Data *m) {
  //can_debug_printf("NMT_command to Reload\n");
  //HAL_NVIC_SystemReset();
}

/*
 ********************************************************************************
 *         state_change function
 * @brief CA callback called when node state changes
 * @param  - Object dictionary structure from Canfestival
 * @return - none
 * @author     from example Canfestival code
 ********************************************************************************
 */
void state_change(CO_Data *d) {
  if (d->nodeState == Initialisation) {
    //can_debug_printf("Node state is now  : Initialisation\n");
  } else if (d->nodeState == Disconnected) {
    //can_debug_printf("Node state is now  : Disconnected\n");
  } else if (d->nodeState == Connecting) {
    //can_debug_printf("Node state is now  : Connecting\n");
  } else if (d->nodeState == Preparing) {
    //can_debug_printf("Node state is now  : Preparing\n");
  } else if (d->nodeState == Stopped) {
    //can_debug_printf("Node state is now  : Stopped\n");
  } else if (d->nodeState == Operational) {
    //can_debug_printf("Node state is now  : Operational\n");
  } else if (d->nodeState == Pre_operational) {
    //can_debug_printf("Node state is now  : Pre_operational\n");
  } else if (d->nodeState == Unknown_state) {
    //can_debug_printf("Node state is now  : Unknown_state\n");
  } else {
    //can_debug_printf("Error : unexpected node state\n");
  }
}

/*
 ********************************************************************************
*         callback_on_position
* @brief    A callback called when position is written
* @param    - Object dictionary structure from Canfestival
 *          - indextable structure
 *          - subindex
 * @return  - none
 * @author  LVA
 ********************************************************************************
 */
UNS32 callback_on_position(CO_Data *d, const indextable *idxtab, UNS8 bSubindex)
{
```

```
  /* Dac 0x2004sub0x00 */
  if (idxtab->index == 0x2004 && bSubindex == 0x00) {
    //can_debug_printf("Dac have been set to %d (0x%x)\n", Dac, Dac);
  }

  /* can_id 0x2004sub0x00*/
  if (idxtab->index == 0x2000 && bSubindex == 0x00) {
    //can_debug_printf("Can_id have been set to %d (0x%x)\n", can_id, can_id);

    if (can_id != old_can_id) { /* проверяем что can_id не изменился */
      if (can_id > 127) {
        can_id = old_can_id;
        //can_debug_printf("Can_id is not valid. set to %d\n", can_id);
      } else {
        can_id_delta = can_id - _bDeviceNodeId;
        old_can_id   = can_id;
        CAN_SetFilter(can_id);
      }
    }
  }
  return 0;
}
```