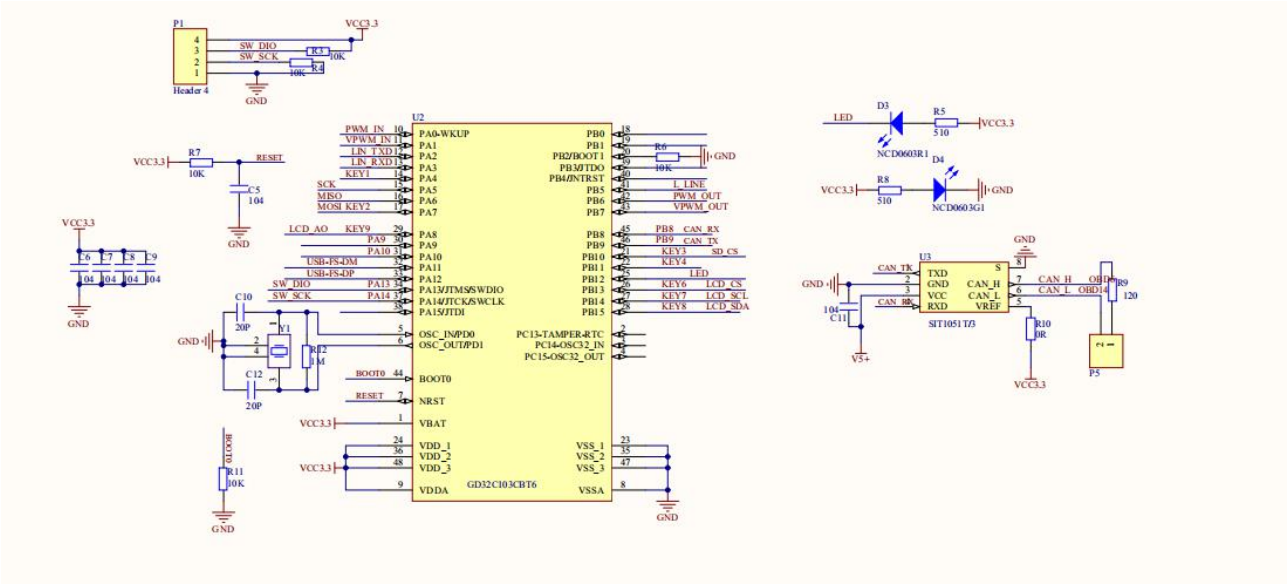
前言

硬件说明:

- MCU: GD32C103 120M,128K,32k RAM.
- 输入:USB 5V.
- OBD 功能口定义:OBD(2,10)VPWM、OBD 7(K 线)、OBD 6(CAN H)、OBD 14(CAN L)、OBD 15(L 线).



软件说明:

一、汽车 CAN2.0(双线 OBD 6、14)

1、支持波特率:1M、800K、500K、250K、125K、100K、62K、50K、33.3K、25K

二、汽车 CAN FD(双线 OBD 6、14)

1、仲裁区波特率:1M、500K

2、数据区波特率 5M 4M 2M 1M

三、汽车 KWP/LIN (OBD 7) 总线数据采集说明

1、波特率:5、4800、9600、10416、57600、115200 BPS

三、SAE J1850 (OBD 2、10)

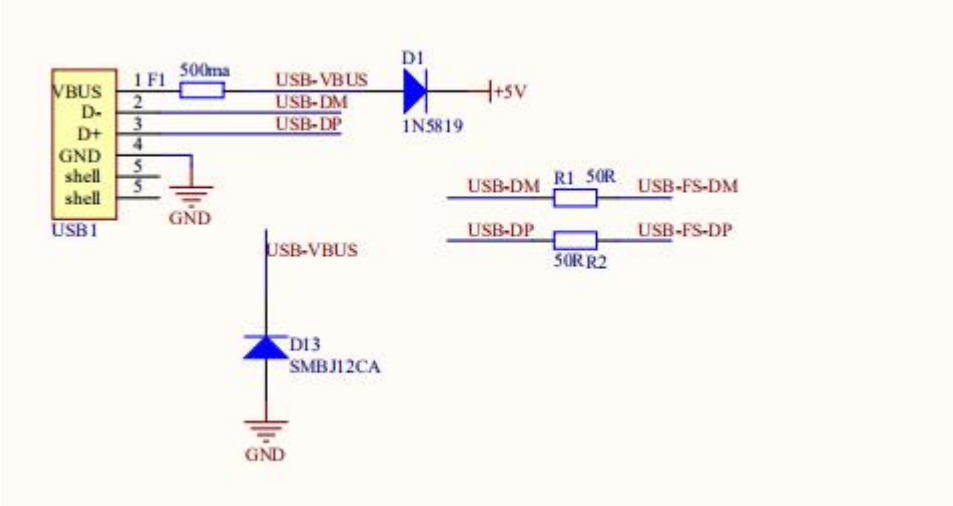
1、PWM 协议发送接收

2、VPW 协议发送接收

目录

第 1 章 GD32 虚拟串口通讯

1. 电路图



2. 例程说明

约定简单通讯协议，测试虚拟串口最大传输速度

(1) 测试平台

1 WIN10 32/64 位（免驱动），WIN7 32 /64 需安装驱动
2 UsbTest.exe 测试工具，传输速度 500K 左右

(2) 通讯协议说明

上位机发送一帧数据格式 55 AA LEN(2BYTE) DATA ...
下位机发送一帧数据格式 55 AA LEN(2BYTE) DATA ...

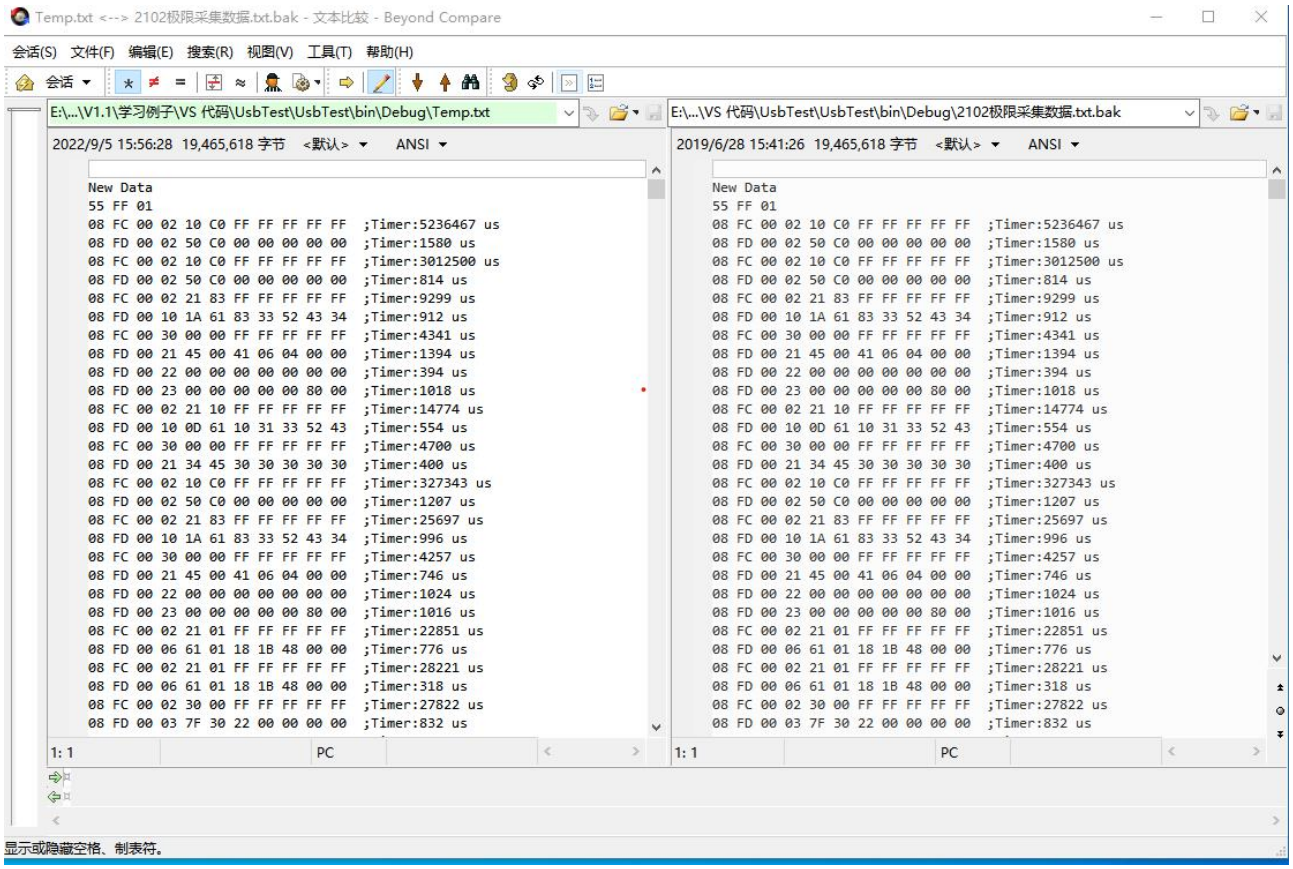
3. 软件设计

UsbTest 上位机软件按 5K 一次循环读取选择的文件内容，
按格式 55 AA LEN(2BYTE) DATA ... 发送给下位机，下位机接收完数据后按
55 AA LEN(2BYTE) DATA ... 格式发回给上位机，上位机根据接收到的 DATA 写进 temp.txt 文件中，当文件发送完成后对比 temp.txt 和选择的文件可知是否丢失数据。

(1) 上位机说明

1. 如下图所示，**Open** 打开串口，**GetFile** 选择文件后开始传输数据

2 传输完成后用 BCompare 软件对比发送和接收的文件，如下图所示 ,18.5M 的文件并未丢失数据



(2)下位机代码说明

1 APP.c 文件代码说明

```
iUsbLen=0;  //接收的数据长度
i UsbFlag=0;//接收完成标记
 iUsbLenPre=0;

/* main loop */
while (1)
  {
      if(iUsbFlag==0x80)//一帧数据 接收完毕
      {
          SendUsbDate(&cdc_acm,iUsbBuf,iUsbLen);//发送数据返回给下位机
          iUsbLenPre=0;
          iUsbLen=0;  //接收的数据长度
          iUsbFlag=0;//接收完成标记
          iCmt++;
          if(iCmt%2) gpio_bit_reset(GPIOB,GPIO_PIN_12);
          else gpio_bit_set(GPIOB,GPIO_PIN_12);
      }
      continue;
  }
```

2 cdc_acm_core.c 文件代码说明

USB 虚拟串口接收函数

```
static uint8_t cdc_acm_out (usb_dev *udev, uint8_t ep_num)
  {
      usb_cdc_handler *cdc = (usb_cdc_handler *)udev->dev.class_data[CDC_COM_INTERFACE];
      cdc->packet_receive = 1U;
      cdc->receive_length = ((usb_core_driver *)udev)->dev.transc_out[ep_num].xfer_count;
      iUsbFlag=0;//接收完成标记
          if(cdc->data[0]==0x55&&cdc->data[1]==0xAA)//帧开头
          {
                  iUsbLen=cdc->data[2]*256+cdc->data[3];
          }
          for(uint8_t i=0;i<cdc->receive_length;i++)
          {
              if(iUsbLenPre>=10239) break;
              iUsbBuf[iUsbLenPre++]=cdc->data[i];
          }

          if(iUsbLenPre>=iUsbLen||iUsbLenPre>10239)
          {
```

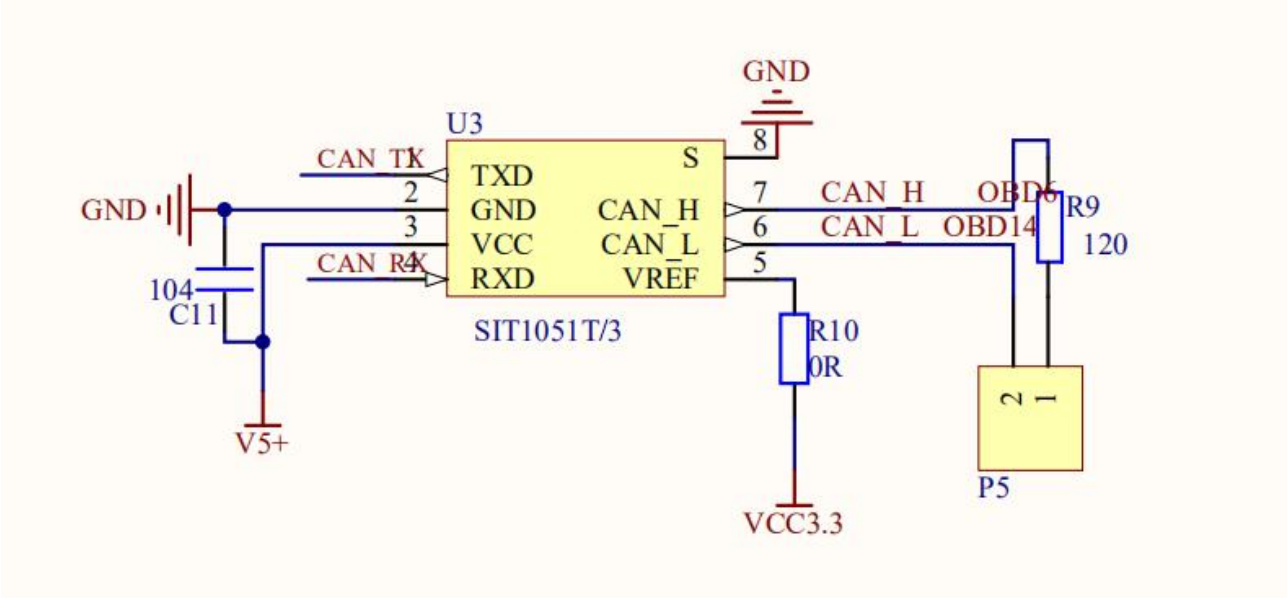
```
            iUsbFlag=0x80;//接收完成标记
        }
        cdc_acm_data_receive(udev);
    return USBD_OK;
}
```

USB 虚拟串口发送函数

```
void SendUsbDate(usb_dev *udev,uint8_t*Buf,uint32_t len)
{
    usb_cdc_handler *cdc = (usb_cdc_handler *)udev->dev.class_data[CDC_COM_INTERFACE];
    if (0U != len)
        {
        cdc->packet_sent = 0U;
        usbd_ep_send (udev, CDC_DATA_IN_EP, (uint8_t*)(Buf), len);
            cdc->receive_length = 0U;
    }
}
```

第 2 章　CAN 2.0 ISO-15765 500K

1. 电路图



2. 例程说明

(1) 硬件连接

用 OBD 一分 2 线接上开发板与 CAN 采集器，采集器设置波特率 500K 不过滤采样。

3. 软件设计

CAN1 接单片机 PB8 PB9，500K 波特率循环发送 CAN 标准帧扩展帧数据

(1) main 主要流程

//初始化 IO 设置波特率 **can_configEx(can_500k);**
//设置过滤器 **CAN_setAllfit();//设置不过滤 ID**
//发送标准帧 **SendISO15765Data(SendData,0xfc00);//15765 STCAN**
//发送扩展帧 **SendISO15765Data(SendData,0x18DB33F1);//15765 EXCAN**

Main() 主要代码

```
can_configEx(can_500k);//500K 波特率
//CAN1_Config16BitFilter(0xFC00,0xFD00);//设置过滤 ID
CAN_setAllfit();//设置不过滤 ID

  /* main loop */
  while (1)
  {
      SendISO15765Data(SendData,0xfc00);//15765 STCAN
      Delay_ms(100);
      SendISO15765Data(SendData,0x18DB33F1);//15765 EXCAN
      Delay_ms(100);
  }
```

(2)can.c 主要函数说明

1 初始化 can_configEx

```
void can_configEx(uint8_t speed)
{
    can_parameter_struct CAN_InitSt;
     can_gpio_config();//IO 初始化   PB8 PB9
    can_struct_para_init(CAN_INIT_STRUCT, &CAN_InitSt);
    /* initialize CAN register */
    can_deinit(CAN0);

    /* initialize CAN parameters */
    CAN_InitSt.time_triggered = DISABLE;
    CAN_InitSt.auto_bus_off_recovery = DISABLE;
    CAN_InitSt.auto_wake_up = DISABLE;

//          0：使能自动重发   ENABLE
//          1：禁用自动重发
```

```
    CAN_InitSt.auto_retrans = ENABLE;//报文自动传输 是否开启 如果是发送的环境 最好设置成自动重发

    CAN_InitSt.rec_fifo_overwrite = DISABLE;
    CAN_InitSt.trans_fifo_order = DISABLE;
    CAN_InitSt.working_mode = CAN_NORMAL_MODE;
     //CAN_SILENT_MODE 静默模式 用于监听总线数据  不发送显性位 当总线有多个节点时 可以选择这个模式
//波特率设置 总线时钟是 60M
//BaudRate = APBCLK/BRP*(1+BS1+BS2)
//SamplePoint = ((1+BS1)/(1+BS1+BS2))*100%

     CAN_InitSt.resync_jump_width=CANBAUD[speed][0];
     CAN_InitSt.time_segment_1=CANBAUD[speed][1];
     CAN_InitSt.time_segment_2=CANBAUD[speed][2];
     CAN_InitSt.prescaler=CANBAUD[speed][3];//BAUD=60m/((1+6+1)*15)   87.5%

          /* initialize CAN */
    can_init(CAN0, &CAN_InitSt);

    /* configure CAN0 NVIC */
    nvic_irq_enable(CAN0_RX0_IRQn, 0, 0);//接收中断 设置
    /* enable can receive FIFO0 not empty interrupt */
    can_interrupt_enable(CAN0, CAN_INTEN_RFNEIE0);//使能接收中断
}
```

2 设置过滤器 CAN_setAllfit

```
//不过滤 ID
void CAN_setAllfit(void)
{
          can_filter_parameter_struct    CAN_FilterInitStructure;
          can_struct_para_init(CAN_FILTER_STRUCT, &CAN_FilterInitStructure);

          CAN_FilterInitStructure.filter_number = 0;
    CAN_FilterInitStructure.filter_mode = CAN_FILTERMODE_MASK;
    CAN_FilterInitStructure.filter_bits = CAN_FILTERBITS_32BIT;

    CAN_FilterInitStructure.filter_enable = ENABLE;
           CAN_FilterInitStructure.filter_fifo_number = CAN_FIFO0;
    /* configure SFID[10:0] */
    CAN_FilterInitStructure.filter_list_high = (uint16_t)0 ;
    CAN_FilterInitStructure.filter_list_low = (uint16_t)0;
    /* configure SFID[10:0] mask */
    CAN_FilterInitStructure.filter_mask_high = (uint16_t)0;
    /* both data and remote frames can be received */
```

```
    CAN_FilterInitStructure.filter_mask_low = (uint16_t)0 ;
     can_filter_init(&CAN_FilterInitStructure);
}
```

3 设置标准过滤器 CAN1_Config16BitFilter

//标准帧过滤 ID 设置

```
void CAN1_Config16BitFilter(u16 id1, u16 id2)
{
    can_filter_parameter_struct    CAN_FilterInitStructure;
     can_struct_para_init(CAN_FILTER_STRUCT, &CAN_FilterInitStructure);

          CAN_FilterInitStructure.filter_number = 0;//过滤器组
    CAN_FilterInitStructure.filter_mode = CAN_FILTERMODE_LIST;//列表模式
    CAN_FilterInitStructure.filter_bits = CAN_FILTERBITS_16BIT;//16 位 ID 模式

    CAN_FilterInitStructure.filter_enable = ENABLE;
           CAN_FilterInitStructure.filter_fifo_number = CAN_FIFO0;
    /* configure SFID[10:0] */
    CAN_FilterInitStructure.filter_list_high = (uint16_t)id1 ;//ID
    CAN_FilterInitStructure.filter_list_low = (uint16_t)id2;//ID
    /* configure SFID[10:0] mask */
    CAN_FilterInitStructure.filter_mask_high = (uint16_t)id1;//掩码
    /* both data and remote frames can be received */
    CAN_FilterInitStructure.filter_mask_low = (uint16_t)id2 ; //掩码
          can_filter_init(&CAN_FilterInitStructure);
}
```

4 设置扩展过滤器 CAN1_Config32BitFilterExList

//列表模式下　有 0~13 共 14 组 能过滤 28 个扩展帧 ID

//iBuffer 29 位 ID 缓冲区，iSumFiter ID 个数

```
void CAN1_Config32BitFilterExList(u32 *iBuffer,u8 iSumFiter)
{
    u32 id1=0,id2=0;
          u8 i=0,iCmt=0,iSum=0;
    can_filter_parameter_struct    CAN_FilterInitStructure;
          iSum=iSumFiter/2;
          if(iSumFiter%2)
          {
               iSum+=1;
          }

          for(i=0;i<iSum;i++)
          {
               id1=iBuffer[iCmt++];
               id2=iBuffer[iCmt++];
```

```
            id1=id1>>3;
            id2=id2>>3;

            if(id1==0) id1=0xffffffff;
            if(id2==0) id2=0xffffffff;

            can_struct_para_init(CAN_FILTER_STRUCT, &CAN_FilterInitStructure);
            CAN_FilterInitStructure.filter_number = i;
            CAN_FilterInitStructure.filter_mode = CAN_FILTERMODE_LIST;
            CAN_FilterInitStructure.filter_bits = CAN_FILTERBITS_32BIT;

            CAN_FilterInitStructure.filter_enable = ENABLE;
             CAN_FilterInitStructure.filter_fifo_number = CAN_FIFO0;
            /* configure SFID[10:0] */
            CAN_FilterInitStructure.filter_list_high = (uint32_t)(id1>>13);
            CAN_FilterInitStructure.filter_list_low = (uint32_t)((id1<<3)|4);
            /* configure SFID[10:0] mask */
            CAN_FilterInitStructure.filter_mask_high = (uint32_t)(id2>>13);
            /* both data and remote frames can be received */
            CAN_FilterInitStructure.filter_mask_low = (uint32_t)((id2<<3)|4);
            can_filter_init(&CAN_FilterInitStructure);
        }
}
```

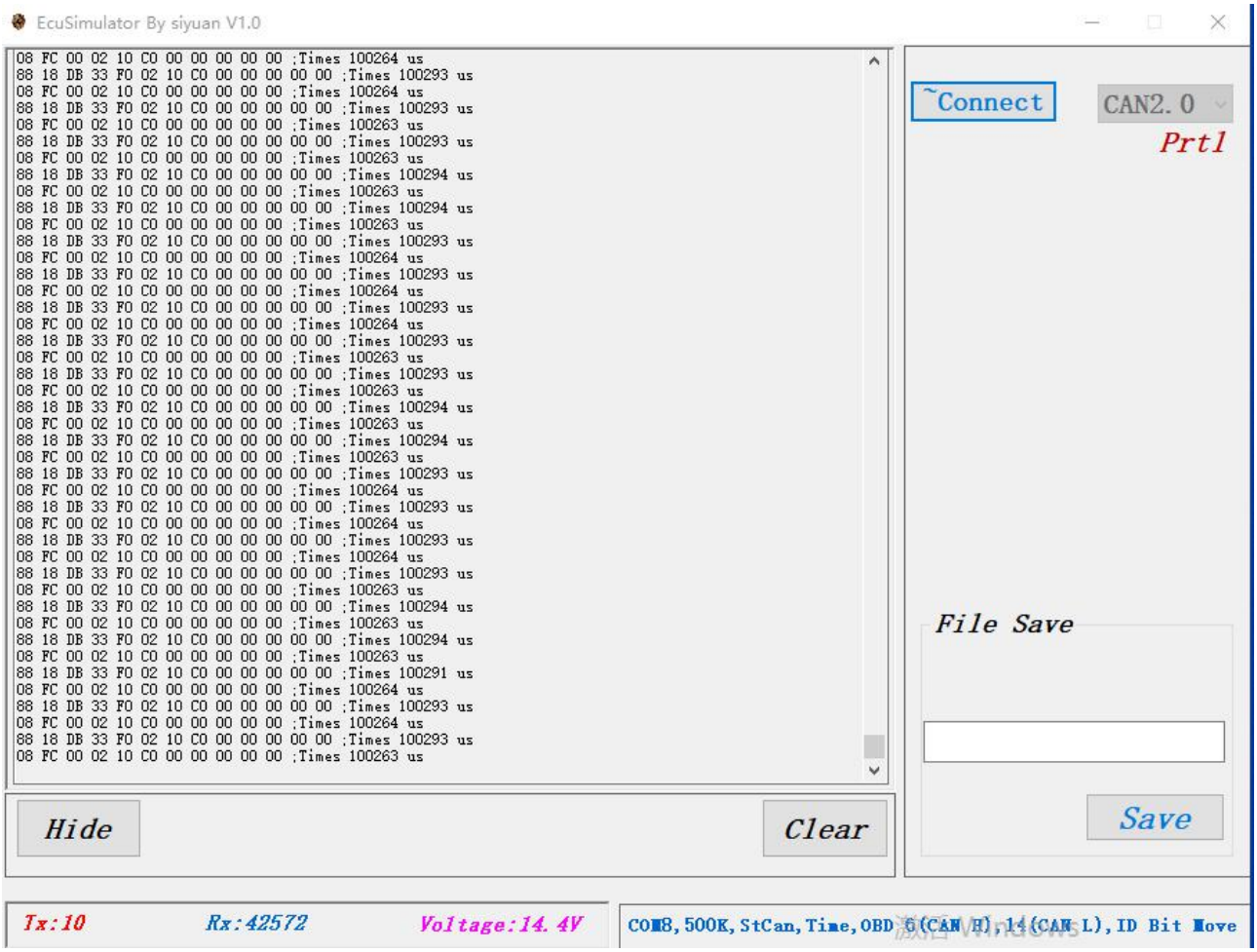
5 CAN2.0 发送命令 SendISO15765Data

```
//iCanId CAN ID   ,已经移位的 FC00(7E0)
// 15765 标准帧
//cmdaddr[0]= 发送长度
//cmdaddr[1...] 发送的数据
//注: iCanId>0xFFFF   时为扩展帧
//  返回 1  成功  返回  0  失败
u8 SendISO15765Data(u8 *cmdaddr,u32 iCanId)     //
{
    u32 i=0,tmp=0;
    uint8_t TransmitMailbox=0;
    u8 Stollen=cmdaddr[0]&0x0F;
    if(Stollen==0) return 1;

    if(Stollen>8)   Stollen=8;
    //初始化参数
    can_trasnmit_message_struct     TbufMege;
    can_struct_para_init(CAN_TX_MESSAGE_STRUCT, &TbufMege);
```

```
    if(iCanId>0Xffff)//扩展帧
    {
        TbufMege.tx_sfid = 0;
        TbufMege.tx_efid = iCanId/0X08;
        TbufMege.tx_ff = CAN_FF_EXTENDED;
    }
    else
    {
        TbufMege.tx_sfid = iCanId/0X20;
        TbufMege.tx_efid = 0x00;
        TbufMege.tx_ff = CAN_FF_STANDARD;
    }
    TbufMege.tx_ft = CAN_FT_DATA;
  TbufMege.tx_dlen = Stollen;
  for(u8 Sidx = 0; Sidx < Stollen; Sidx ++)
  {
            TbufMege.tx_data[Sidx] = cmdaddr[Sidx+1];
  }
    can_transmit_state_enum iFlag;
    TransmitMailbox=can_message_transmit(CAN0, &TbufMege);
//    //判断是否发送成功
    GetTimer6Cnt();//清空
        i=0;
     while(1)
     {
        i++;
        iFlag=can_transmit_states(CAN0,TransmitMailbox);
        if(iFlag==CAN_TRANSMIT_OK) break;//校验是否发送成功
        if(i>0xFFFFFF) break;

        tmp=GetTimer6CntEx();
        if(tmp>1000*500) break;
     }
  return 1;
}
```

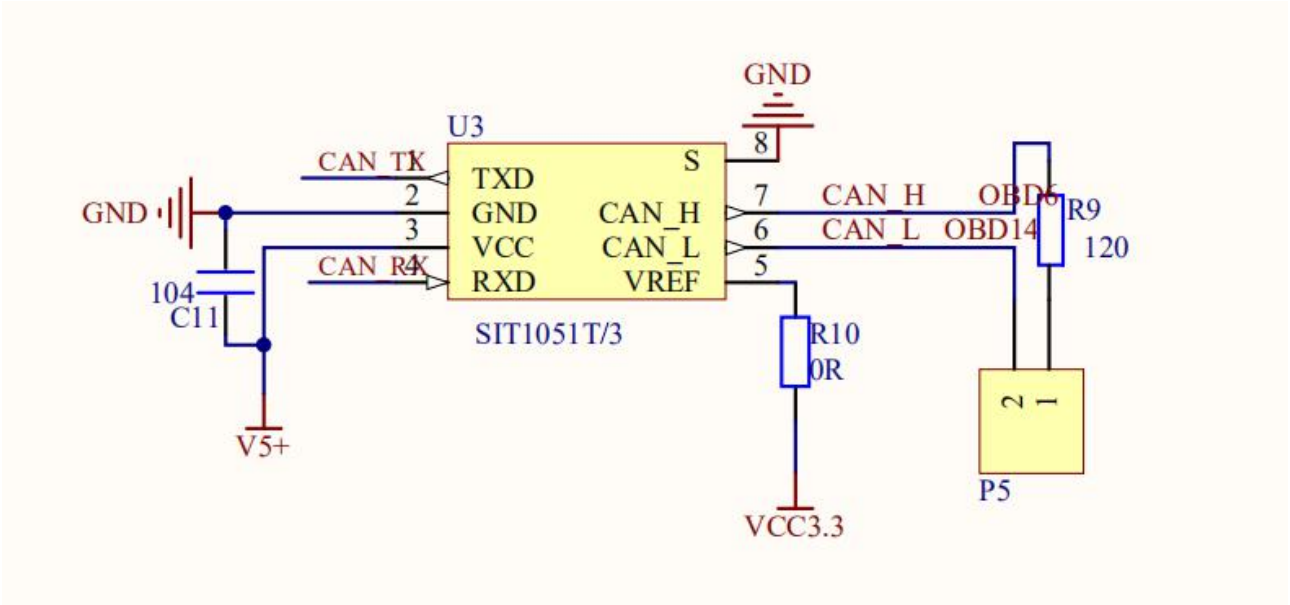
4. PC 平台效果

EcuSimulator 工具设置 CAN2.0 500K 不过滤，显示数据如下图所示



第 3 章　CAN_FD_500K_5M_83.3%

1. 电路图



2. 例程说明

仲裁区波特率 500K,数据区波特率 5M 采样点 83.3%

用 OBD 一分 2 线接上开发板与 CAN 采集器，采集器设置波特率 500K 不过滤采样

3.软件设计

CAN1 接单片机 PB8 PB9，500K_5M 波特率循环发送 CAN 标准帧扩展帧数据

(1)main 主要流程

//初始化 IO 设置波特率 **CanFD_config(can_500k,Data_5M);//CAN FD 500k 5M ;**
//设置过滤器 **CAN_setAllfit();//设置不过滤 ID**
//canfd 发送标准帧 **CanFdSendISO15765Data(SendData,0xfc00);//15765 STCAN**
//canfd 发送扩展帧 **CanFdSendISO15765Data(SendData,0x18DB33F1);//15765 EXCAN**

Main() 主要代码

```
uint8_t SendData[10]={0x08,0X02,0X10,0x03,0x00,0x00,0x00,0x00,0x00};
uint8_t SendData1[100]={0x09,0X02,0X10,0x03,0x00,0x00,0x00,0x00,0x00};
CanFD_config(can_500k,Data_5M);//CAN FD 500k  5M
//CAN1_Config16BitFilter(0xFC00,0xFD00);//设置过滤 ID
CAN_setAllfit();//设置不过滤 ID

  /* main loop */
while (1)
  {

      CanFdSendISO15765Data(SendData,0xfc00);//15765 STCAN 标准帧
      Delay_ms(100);
      CanFdSendISO15765Data(SendData,0x18DB33F1);//15765 EXCAN 扩展帧
      Delay_ms(100);
      for(i=0;i<0x10;i++)//循环发送 长度 0~64 字节的数据
      {
          SendData1[0]=i;//长度改变
          SendData1[1]=i;
          CanFdSendISO15765Data(SendData1,0xfd00);// STCAN 变长度 验证 CAN FD
          Delay_ms(100);
      }
  }
```

(2)can.c 主要函数说明

1 初始化 CanFD_config

```
//speed1 仲裁去波特率
//speed2 数据区波特率
void CanFD_config(uint8_t speed1,uint8_t speed2)
{
    can_parameter_struct CAN_InitSt;
     can_fdframe_struct can_fd_parameter; //CAN FD 参数
```

```
    can_fd_tdc_struct can_fd_tdc_parameter;//
   //GPIO
  can_gpio_config();
  can_struct_para_init(CAN_INIT_STRUCT, &CAN_InitSt);
  /* initialize CAN register */
  can_deinit(CAN0);




  /* initialize CAN parameters */
  CAN_InitSt.time_triggered = DISABLE;
  CAN_InitSt.auto_bus_off_recovery = DISABLE;
  CAN_InitSt.auto_wake_up = DISABLE;

//              0：使能自动重发
//              1：禁用自动重发
  CAN_InitSt.auto_retrans = ENABLE;//报文自动传输 是否开启
  CAN_InitSt.rec_fifo_overwrite = DISABLE;
  CAN_InitSt.trans_fifo_order = DISABLE;
  CAN_InitSt.working_mode = CAN_NORMAL_MODE;

    //speed1 仲裁区波特率
    CAN_InitSt.resync_jump_width=CANBAUD[speed1][0];
    CAN_InitSt.time_segment_1=CANBAUD[speed1][1];
    CAN_InitSt.time_segment_2=CANBAUD[speed1][2];
    CAN_InitSt.prescaler=CANBAUD[speed1][3];//BAUD=60m/((1+6+1)*15)    87.5%

    /* initialize CAN */
  can_init(CAN0, &CAN_InitSt);
    //can_frequency_set(CAN0, DEV_CAN_BAUD_RATE);

  //数据区 初始化
    can_struct_para_init(CAN_FD_FRAME_STRUCT, &can_fd_parameter);
  can_fd_parameter.fd_frame = ENABLE;
  can_fd_parameter.excp_event_detect = ENABLE;
  can_fd_parameter.delay_compensation = ENABLE;

  can_fd_tdc_parameter.tdc_filter = 0x04;
  can_fd_tdc_parameter.tdc_mode = CAN_TDCMOD_CALC_AND_OFFSET;
  can_fd_tdc_parameter.tdc_offset = 0x04;
  can_fd_parameter.p_delay_compensation = &can_fd_tdc_parameter;
  can_fd_parameter.iso_bosch = CAN_FDMOD_ISO;
  can_fd_parameter.esi_mode = CAN_ESIMOD_HARDWARE;
```

```
    //数据区波特率设置
      can_fd_parameter.data_resync_jump_width=CANBAUD_data[speed2][0];
      can_fd_parameter.data_time_segment_1=CANBAUD_data[speed2][1];
      can_fd_parameter.data_time_segment_2=CANBAUD_data[speed2][2];
      can_fd_parameter.data_prescaler=CANBAUD_data[speed2][3];//BAUD=60m/((1+6+1)*15)
      can_fd_init(CAN0, &can_fd_parameter);

  //can_fd_frequency_set(CAN0, 5000000);//1M   自动波特率函数
    /* configure CAN0 NVIC */
    nvic_irq_enable(CAN0_RX0_IRQn, 0, 0);
    /* enable can receive FIFO0 not empty interrupt */
    can_interrupt_enable(CAN0, CAN_INTEN_RFNEIE0);
}
```

2 设置过滤器 CAN_setAllfit

//不过滤 ID

```
void CAN_setAllfit(void)
{
        can_filter_parameter_struct    CAN_FilterInitStructure;
        can_struct_para_init(CAN_FILTER_STRUCT, &CAN_FilterInitStructure);

        CAN_FilterInitStructure.filter_number = 0;
    CAN_FilterInitStructure.filter_mode = CAN_FILTERMODE_MASK;
    CAN_FilterInitStructure.filter_bits = CAN_FILTERBITS_32BIT;

    CAN_FilterInitStructure.filter_enable = ENABLE;
          CAN_FilterInitStructure.filter_fifo_number = CAN_FIFO0;
    /* configure SFID[10:0] */
    CAN_FilterInitStructure.filter_list_high = (uint16_t)0 ;
    CAN_FilterInitStructure.filter_list_low = (uint16_t)0;
    /* configure SFID[10:0] mask */
    CAN_FilterInitStructure.filter_mask_high = (uint16_t)0;
    /* both data and remote frames can be received */
    CAN_FilterInitStructure.filter_mask_low = (uint16_t)0 ;
     can_filter_init(&CAN_FilterInitStructure);
}
```

3 设置标准过滤器 CAN1_Config16BitFilter

//标准帧过滤 ID 设置

```
void CAN1_Config16BitFilter(u16 id1, u16 id2)
{
    can_filter_parameter_struct    CAN_FilterInitStructure;
     can_struct_para_init(CAN_FILTER_STRUCT, &CAN_FilterInitStructure);
```

```
        CAN_FilterInitStructure.filter_number = 0;//过滤器组
    CAN_FilterInitStructure.filter_mode = CAN_FILTERMODE_LIST;//列表模式
    CAN_FilterInitStructure.filter_bits = CAN_FILTERBITS_16BIT;//16 位 ID 模式

    CAN_FilterInitStructure.filter_enable = ENABLE;
          CAN_FilterInitStructure.filter_fifo_number = CAN_FIFO0;
    /* configure SFID[10:0] */
    CAN_FilterInitStructure.filter_list_high = (uint16_t)id1 ;//ID
    CAN_FilterInitStructure.filter_list_low = (uint16_t)id2;//ID
    /* configure SFID[10:0] mask */
    CAN_FilterInitStructure.filter_mask_high = (uint16_t)id1;//掩码
    /* both data and remote frames can be received */
    CAN_FilterInitStructure.filter_mask_low = (uint16_t)id2 ; //掩码
        can_filter_init(&CAN_FilterInitStructure);
}
```

4 设置扩展过滤器 CAN1_Config32BitFilterExList

```
//列表模式下   有 0~13 共 14 组 能过滤 28 个扩展帧 ID
//iBuffer 29 位 ID 缓冲区，iSumFiter ID 个数
void CAN1_Config32BitFilterExList(u32 *iBuffer,u8 iSumFiter)
{
    u32 id1=0,id2=0;
        u8 i=0,iCmt=0,iSum=0;
    can_filter_parameter_struct   CAN_FilterInitStructure;
        iSum=iSumFiter/2;
        if(iSumFiter%2)
        {
            iSum+=1;
        }

        for(i=0;i<iSum;i++)
        {
            id1=iBuffer[iCmt++];
            id2=iBuffer[iCmt++];

            id1=id1>>3;
            id2=id2>>3;

            if(id1==0) id1=0xffffffff;
            if(id2==0) id2=0xffffffff;

            can_struct_para_init(CAN_FILTER_STRUCT, &CAN_FilterInitStructure);
            CAN_FilterInitStructure.filter_number = i;
```

```
                CAN_FilterInitStructure.filter_mode = CAN_FILTERMODE_LIST;
                CAN_FilterInitStructure.filter_bits = CAN_FILTERBITS_32BIT;

                CAN_FilterInitStructure.filter_enable = ENABLE;
                 CAN_FilterInitStructure.filter_fifo_number = CAN_FIFO0;
                /* configure SFID[10:0] */
                CAN_FilterInitStructure.filter_list_high = (uint32_t)(id1>>13);
                CAN_FilterInitStructure.filter_list_low = (uint32_t)((id1<<3)|4);
                /* configure SFID[10:0] mask */
                CAN_FilterInitStructure.filter_mask_high = (uint32_t)(id2>>13);
                /* both data and remote frames can be received */
                CAN_FilterInitStructure.filter_mask_low = (uint32_t)((id2<<3)|4);
                can_filter_init(&CAN_FilterInitStructure);
        }
}
```

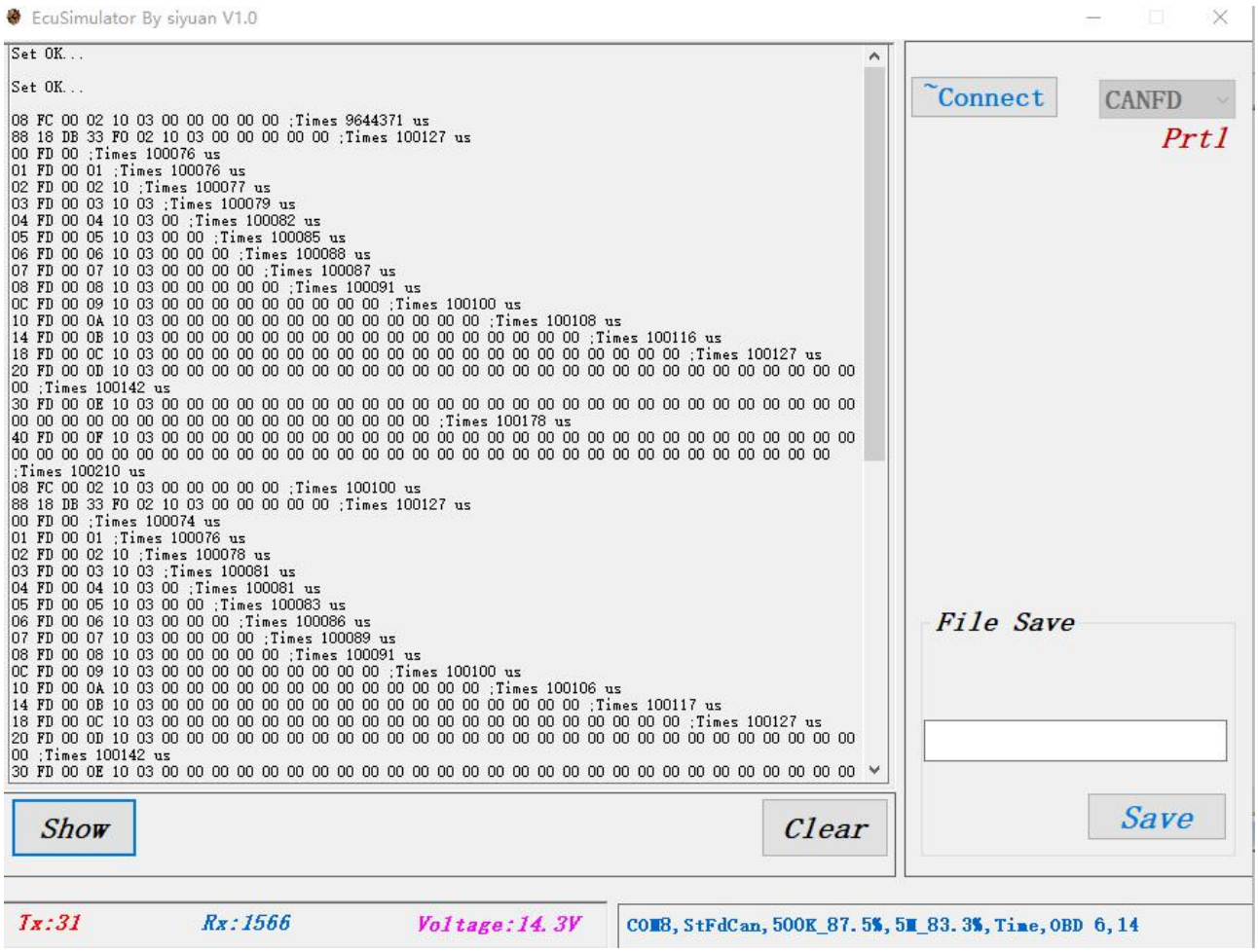
5 CANFD 发送命令 CanFdSendISO15765Data

```
 //iCanId CAN ID   ,已经移位的 FC00(7E0)
// 15765 标准帧
//cmdaddr[0]= 发送长度
//cmdaddr[1...] 发送的数据
//注: iCanId>0xFFFF   时为扩展帧
//CAN FD 协议 数据长度 1~8，12 16，20，24，32，48，64 最大长度是 64，但并不能 1~64 之间的任意长度
// 返回 1 成功 返回 0 失败
u8 CanFdSendISO15765Data(u8 *cmdaddr,u32 iCanId)     //
{
    u32 i=0;
    uint8_t TransmitMailbox=0;
    u8 Stollen=GetFdCanLen(cmdaddr[0]&0x0F);//CAN FD 长度
    //初始化参数
    can_trasnmit_message_struct     TbufMege;
    can_struct_para_init(CAN_TX_MESSAGE_STRUCT, &TbufMege);
    if(iCanId>0xFFFF)//扩展帧
    {
        TbufMege.tx_sfid = 0;
        TbufMege.tx_efid = iCanId/0X08;
        TbufMege.tx_ff = CAN_FF_EXTENDED;//
    }
    else
    {
        TbufMege.tx_sfid = iCanId/0X20;
        TbufMege.tx_efid = 0x00;
```

```
            TbufMege.tx_ff = CAN_FF_STANDARD;
        }

    TbufMege.tx_ft = CAN_FT_DATA;//数据帧
    TbufMege.tx_dlen = Stollen;//帧长度
    TbufMege.fd_flag = CAN_FDF_FDFRAME;//CAN FD
    TbufMege.fd_brs = CAN_BRS_ENABLE;//波特率切换
    TbufMege.fd_esi = CAN_ESI_DOMINANT;//CAN_ESI_DOMINANT;//;

  for(u8 Sidx = 0; Sidx < Stollen; Sidx ++)
  {
                TbufMege.tx_data[Sidx] = cmdaddr[Sidx+1];
  }



      u8 flag=0;
      //can_transmit_state_enum iFlag;
      TransmitMailbox=can_message_transmit(CAN0, &TbufMege);
      while(1)
      {
            flag=can_transmit_states(CAN0,TransmitMailbox);
            if(flag==CAN_TRANSMIT_OK) break;

            if(i>250)
            {//
                  i=0;
                  break;
            }
            Delay_us(1);
            i++;

       }

  return 1;//
}
```

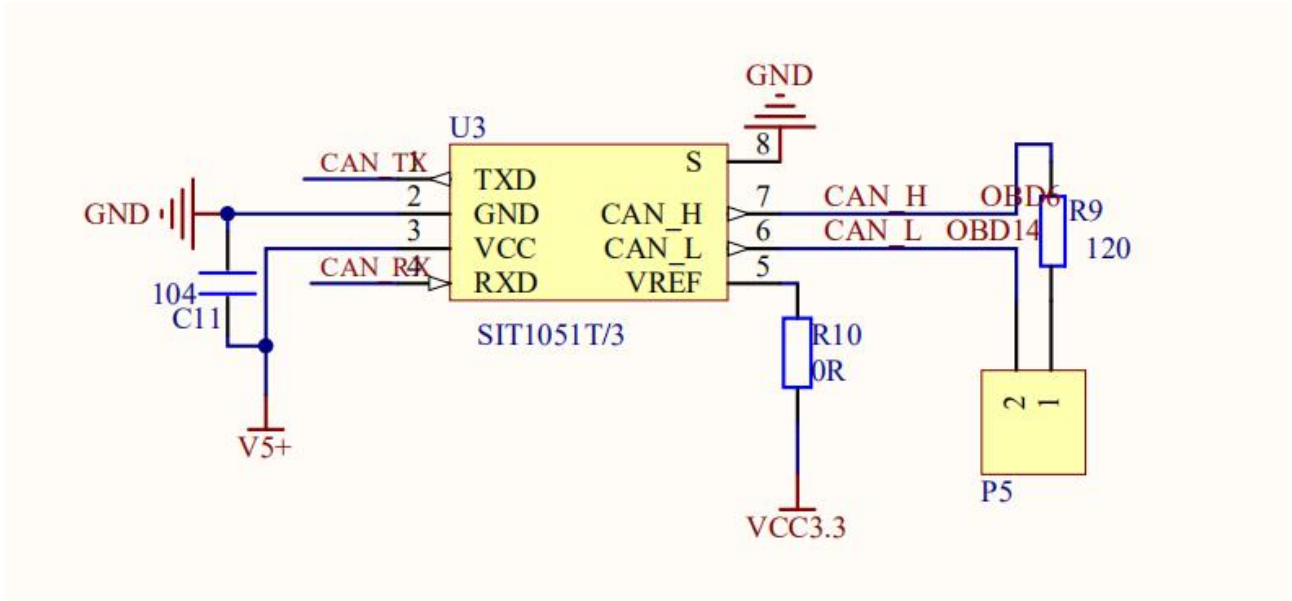
4. PC 平台效果

EcuSimulator 工具设置 CANFD 500K ,5M_83%　不过滤，显示数据如下图所示



第 4 章　CAN_FD_500K_4M_80%

1.　电路图



2. 例程说明

仲裁区波特率 500K,数据区波特率 4M 采样点 80%

用 OBD 一分 2 线接上开发板与 CAN 采集器，采集器设置波特率 500K 不过滤采样

3. 软件设计

CAN1 接单片机 PB8 PB9，500K_4M 波特率循环发送 CAN 标准帧扩展帧数据

(1)main 主要流程

//初始化 IO 设置波特率 **CanFD_config(can_500k,Data_4M);//CAN FD 500k 4M ;**
//设置过滤器 **CAN_setAllfit();//设置不过滤 ID**
//canfd 发送标准帧 **CanFdSendISO15765Data(SendData,0xfc00);//15765 STCAN**
//canfd 发送扩展帧 **CanFdSendISO15765Data(SendData,0x18DB33F1);//15765 EXCAN**

Main() 主要代码

```
uint8_t SendData[10]={0x08,0X02,0X10,0x03,0x00,0x00,0x00,0x00,0x00};
uint8_t SendData1[100]={0x09,0X02,0X10,0x03,0x00,0x00,0x00,0x00,0x00};
CanFD_config(can_500k,Data_4M);//CAN FD 500k 4M
//CAN1_Config16BitFilter(0xFC00,0xFD00);//设置过滤 ID
CAN_setAllfit();//设置不过滤 ID

  /* main loop */
while (1)
  {

      CanFdSendISO15765Data(SendData,0xfc00);//15765 STCAN 标准帧
      Delay_ms(100);
      CanFdSendISO15765Data(SendData,0x18DB33F1);//15765 EXCAN 扩展帧
      Delay_ms(100);
      for(i=0;i<0x10;i++)//循环发送 长度 0~64 字节的数据
      {
          SendData1[0]=i;//长度改变
          SendData1[1]=i;
          CanFdSendISO15765Data(SendData1,0xfd00);// STCAN 变长度 验证 CAN FD
          Delay_ms(100);
      }
  }
```

(2)can.c 主要函数说明

1 初始化 CanFD_config

```
//speed1 仲裁去波特率
//speed2 数据区波特率
void CanFD_config(uint8_t speed1,uint8_t speed2)
{
  can_parameter_struct CAN_InitSt;
```

```
  can_fdframe_struct can_fd_parameter; //CAN FD 参数
  can_fd_tdc_struct can_fd_tdc_parameter;//
 //GPIO
can_gpio_config();
can_struct_para_init(CAN_INIT_STRUCT, &CAN_InitSt);
/* initialize CAN register */
can_deinit(CAN0);



/* initialize CAN parameters */
CAN_InitSt.time_triggered = DISABLE;
CAN_InitSt.auto_bus_off_recovery = DISABLE;
CAN_InitSt.auto_wake_up = DISABLE;

//              0：使能自动重发
//              1：禁用自动重发
CAN_InitSt.auto_retrans = ENABLE;//报文自动传输 是否开启
CAN_InitSt.rec_fifo_overwrite = DISABLE;
CAN_InitSt.trans_fifo_order = DISABLE;
CAN_InitSt.working_mode = CAN_NORMAL_MODE;

  //speed1 仲裁区波特率
  CAN_InitSt.resync_jump_width=CANBAUD[speed1][0];
  CAN_InitSt.time_segment_1=CANBAUD[speed1][1];
  CAN_InitSt.time_segment_2=CANBAUD[speed1][2];
  CAN_InitSt.prescaler=CANBAUD[speed1][3];//BAUD=60m/((1+6+1)*15)    87.5%

  /* initialize CAN */
can_init(CAN0, &CAN_InitSt);
  //can_frequency_set(CAN0, DEV_CAN_BAUD_RATE);

//数据区 初始化
  can_struct_para_init(CAN_FD_FRAME_STRUCT, &can_fd_parameter);
can_fd_parameter.fd_frame = ENABLE;
can_fd_parameter.excp_event_detect = ENABLE;
can_fd_parameter.delay_compensation = ENABLE;

can_fd_tdc_parameter.tdc_filter = 0x04;
can_fd_tdc_parameter.tdc_mode = CAN_TDCMOD_CALC_AND_OFFSET;
can_fd_tdc_parameter.tdc_offset = 0x04;
can_fd_parameter.p_delay_compensation = &can_fd_tdc_parameter;
can_fd_parameter.iso_bosch = CAN_FDMOD_ISO;
```

```
    can_fd_parameter.esi_mode = CAN_ESIMOD_HARDWARE;

    //数据区波特率设置
      can_fd_parameter.data_resync_jump_width=CANBAUD_data[speed2][0];
      can_fd_parameter.data_time_segment_1=CANBAUD_data[speed2][1];
      can_fd_parameter.data_time_segment_2=CANBAUD_data[speed2][2];
      can_fd_parameter.data_prescaler=CANBAUD_data[speed2][3];//BAUD=60m/((1+6+1)*15)
      can_fd_init(CAN0, &can_fd_parameter);

  //can_fd_frequency_set(CAN0, 5000000);//1M   自动波特率函数
    /* configure CAN0 NVIC */
    nvic_irq_enable(CAN0_RX0_IRQn, 0, 0);
    /* enable can receive FIFO0 not empty interrupt */
    can_interrupt_enable(CAN0, CAN_INTEN_RFNEIE0);
}
```

2 设置过滤器 CAN_setAllfit

//不过滤 ID

```
void CAN_setAllfit(void)
{
        can_filter_parameter_struct    CAN_FilterInitStructure;
        can_struct_para_init(CAN_FILTER_STRUCT, &CAN_FilterInitStructure);

        CAN_FilterInitStructure.filter_number = 0;
    CAN_FilterInitStructure.filter_mode = CAN_FILTERMODE_MASK;
    CAN_FilterInitStructure.filter_bits = CAN_FILTERBITS_32BIT;

    CAN_FilterInitStructure.filter_enable = ENABLE;
          CAN_FilterInitStructure.filter_fifo_number = CAN_FIFO0;
    /* configure SFID[10:0] */
    CAN_FilterInitStructure.filter_list_high = (uint16_t)0 ;
    CAN_FilterInitStructure.filter_list_low = (uint16_t)0;
    /* configure SFID[10:0] mask */
    CAN_FilterInitStructure.filter_mask_high = (uint16_t)0;
    /* both data and remote frames can be received */
    CAN_FilterInitStructure.filter_mask_low = (uint16_t)0 ;
     can_filter_init(&CAN_FilterInitStructure);
}
```

3 设置标准过滤器 CAN1_Config16BitFilter

//标准帧过滤 ID 设置

```
void CAN1_Config16BitFilter(u16 id1, u16 id2)
{
    can_filter_parameter_struct    CAN_FilterInitStructure;
```

```
 can_struct_para_init(CAN_FILTER_STRUCT, &CAN_FilterInitStructure);

        CAN_FilterInitStructure.filter_number = 0;//过滤器组
    CAN_FilterInitStructure.filter_mode = CAN_FILTERMODE_LIST;//列表模式
    CAN_FilterInitStructure.filter_bits = CAN_FILTERBITS_16BIT;//16 位 ID 模式

    CAN_FilterInitStructure.filter_enable = ENABLE;
         CAN_FilterInitStructure.filter_fifo_number = CAN_FIFO0;
    /* configure SFID[10:0] */
    CAN_FilterInitStructure.filter_list_high = (uint16_t)id1 ;//ID
    CAN_FilterInitStructure.filter_list_low = (uint16_t)id2;//ID
    /* configure SFID[10:0] mask */
    CAN_FilterInitStructure.filter_mask_high = (uint16_t)id1;//掩码
    /* both data and remote frames can be received */
    CAN_FilterInitStructure.filter_mask_low = (uint16_t)id2 ; //掩码
        can_filter_init(&CAN_FilterInitStructure);
}
```

4 设置扩展过滤器 CAN1_Config32BitFilterExList

```
//列表模式下   有 0~13 共 14 组 能过滤 28 个扩展帧 ID
//iBuffer 29 位 ID 缓冲区，iSumFiter ID 个数
void CAN1_Config32BitFilterExList(u32 *iBuffer,u8 iSumFiter)
{
    u32 id1=0,id2=0;
        u8 i=0,iCmt=0,iSum=0;
    can_filter_parameter_struct    CAN_FilterInitStructure;
        iSum=iSumFiter/2;
        if(iSumFiter%2)
        {
            iSum+=1;
        }

        for(i=0;i<iSum;i++)
        {
            id1=iBuffer[iCmt++];
            id2=iBuffer[iCmt++];

            id1=id1>>3;
            id2=id2>>3;

            if(id1==0) id1=0xffffffff;
            if(id2==0) id2=0xffffffff;

            can_struct_para_init(CAN_FILTER_STRUCT, &CAN_FilterInitStructure);
```

```
                CAN_FilterInitStructure.filter_number = i;
                CAN_FilterInitStructure.filter_mode = CAN_FILTERMODE_LIST;
                CAN_FilterInitStructure.filter_bits = CAN_FILTERBITS_32BIT;

                CAN_FilterInitStructure.filter_enable = ENABLE;
                 CAN_FilterInitStructure.filter_fifo_number = CAN_FIFO0;
                /* configure SFID[10:0] */
                CAN_FilterInitStructure.filter_list_high = (uint32_t)(id1>>13);
                CAN_FilterInitStructure.filter_list_low = (uint32_t)((id1<<3)|4);
                /* configure SFID[10:0] mask */
                CAN_FilterInitStructure.filter_mask_high = (uint32_t)(id2>>13);
                /* both data and remote frames can be received */
                CAN_FilterInitStructure.filter_mask_low = (uint32_t)((id2<<3)|4);
                can_filter_init(&CAN_FilterInitStructure);
            }
}
```

5 CANFD 发送命令 CanFdSendISO15765Data

```
 //iCanId CAN ID   ,已经移位的 FC00(7E0)
// 15765 标准帧
//cmdaddr[0]= 发送长度
//cmdaddr[1...] 发送的数据
//注: iCanId>0xFFFF   时为扩展帧
//CAN FD 协议 数据长度 1~8，12 16，20，24，32，48，64 最大长度是 64，但并不能 1~64 之间的任意长度
// 返回 1 成功 返回 0 失败
u8 CanFdSendISO15765Data(u8 *cmdaddr,u32 iCanId)     //
{
    u32 i=0;
    uint8_t TransmitMailbox=0;
    u8 Stollen=GetFdCanLen(cmdaddr[0]&0x0F);//CAN FD 长度
    //初始化参数
    can_trasnmit_message_struct     TbufMege;
    can_struct_para_init(CAN_TX_MESSAGE_STRUCT, &TbufMege);
    if(iCanId>0xFFFF)//扩展帧
    {
        TbufMege.tx_sfid = 0;
        TbufMege.tx_efid = iCanId/0X08;
        TbufMege.tx_ff = CAN_FF_EXTENDED;//
    }
    else
    {
        TbufMege.tx_sfid = iCanId/0X20;
```

```
            TbufMege.tx_efid = 0x00;
            TbufMege.tx_ff = CAN_FF_STANDARD;
        }

    TbufMege.tx_ft = CAN_FT_DATA;//数据帧
    TbufMege.tx_dlen = Stollen;//帧长度
    TbufMege.fd_flag = CAN_FDF_FDFRAME;//CAN FD
    TbufMege.fd_brs = CAN_BRS_ENABLE;//波特率切换
    TbufMege.fd_esi = CAN_ESI_DOMINANT;//CAN_ESI_DOMINANT;//;

   for(u8 Sidx = 0; Sidx < Stollen; Sidx ++)
   {
                TbufMege.tx_data[Sidx] = cmdaddr[Sidx+1];
   }



       u8 flag=0;
       //can_transmit_state_enum iFlag;
       TransmitMailbox=can_message_transmit(CAN0, &TbufMege);
       while(1)
       {
            flag=can_transmit_states(CAN0,TransmitMailbox);
            if(flag==CAN_TRANSMIT_OK) break;

            if(i>250)
            {//
                i=0;
                break;
            }
            Delay_us(1);
            i++;

        }

   return 1;//
}
```

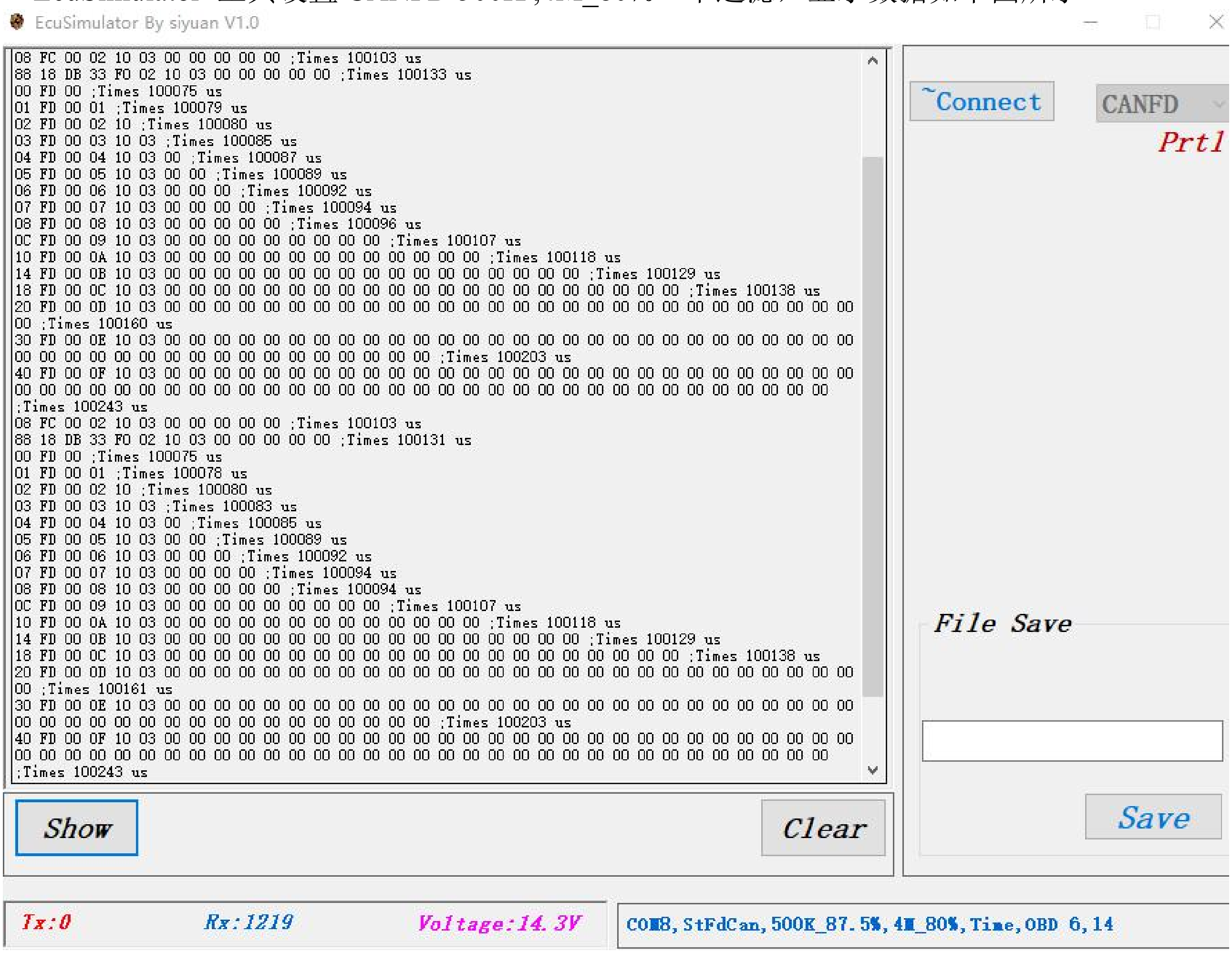
4M

```
      //CAN1_Config16BitFilter(0xFC00,0xFD00);//设置过滤 ID
      CAN_setAllfit();//设置不过滤 ID

     /* main loop */
```

```
while (1)
  {

       CanFdSendISO15765Data(SendData,0xfc00);//15765 STCAN 标准帧
       Delay_ms(100);
       CanFdSendISO15765Data(SendData,0x18DB33F1);//15765 EXCAN 扩展帧
       Delay_ms(100);
       for(i=0;i<0x10;i++)//循环发送 长度 0~64 字节的数据
       {
           SendData1[0]=i;//长度改变
           SendData1[1]=i;
           CanFdSendISO15765Data(SendData1,0xfd00);// STCAN 变长度 验证 CAN FD
           Delay_ms(100);
       }
  }
```

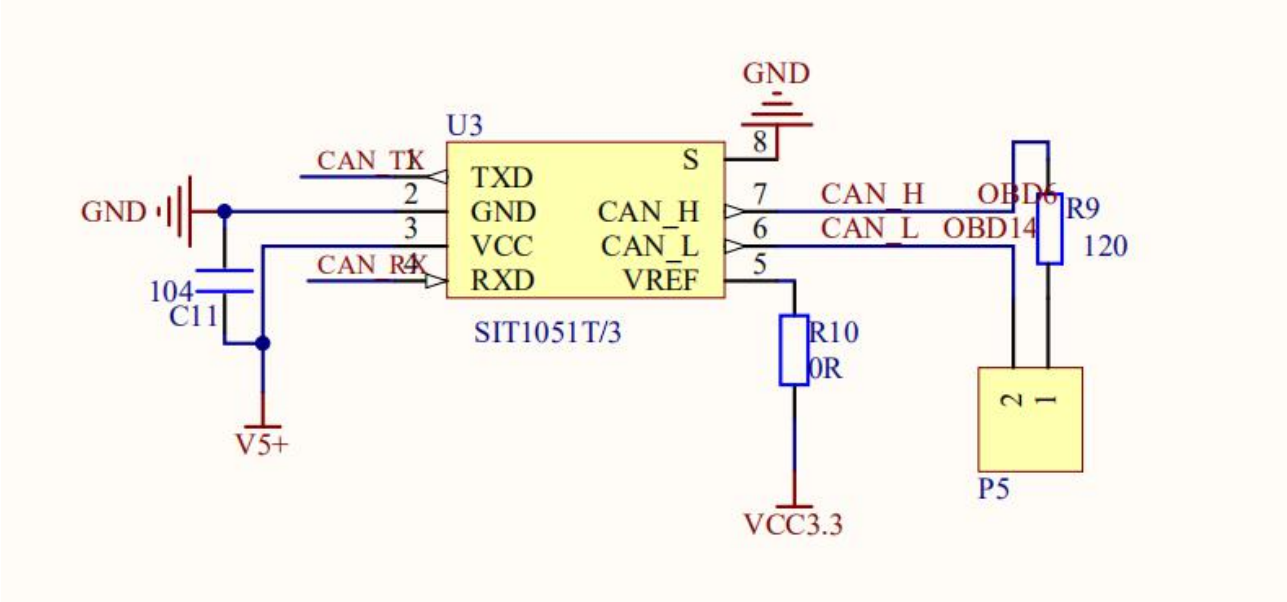
4. PC 平台效果

EcuSimulator 工具设置 CANFD 500K ,4M_80% 不过滤，显示数据如下图所示



第 5 章　CAN_FD_500K_2M_73.3%

1.　电路图



2. 例程说明

仲裁区波特率 500K,数据区波特率 2M 采样点 73.3%
用 OBD 一分 2 线接上开发板与 CAN 采集器，采集器设置波特率 500K 不过滤采样

3. 软件设计

CAN1 接单片机 PB8 PB9，500K_2M 波特率循环发送 CAN 标准帧扩展帧数据

(1)main 主要流程

//初始化 IO 设置波特率 **CanFD_config(can_500k,Data_2M);//CAN FD 500k　2M ;**
//设置过滤器 **CAN_setAllfit();//设置不过滤 ID**
//canfd 发送标准帧 **CanFdSendISO15765Data(SendData,0xfc00);//15765 STCAN**
//canfd 发送扩展帧 **CanFdSendISO15765Data(SendData,0x18DB33F1);//15765 EXCAN**

Main() 主要代码

```
uint8_t SendData[10]={0x08,0X02,0X10,0x03,0x00,0x00,0x00,0x00,0x00};
uint8_t SendData1[100]={0x09,0X02,0X10,0x03,0x00,0x00,0x00,0x00,0x00};
CanFD_config(can_500k,Data_2M);//CAN FD 500k 2M
//CAN1_Config16BitFilter(0xFC00,0xFD00);//设置过滤 ID
CAN_setAllfit();//设置不过滤 ID

  /* main loop */
while (1)
  {
```

```
            CanFdSendISO15765Data(SendData,0xfc00);//15765 STCAN 标准帧
            Delay_ms(100);
            CanFdSendISO15765Data(SendData,0x18DB33F1);//15765 EXCAN 扩展帧
            Delay_ms(100);
            for(i=0;i<0x10;i++)//循环发送 长度 0~64 字节的数据
            {
                SendData1[0]=i;//长度改变
                SendData1[1]=i;
                CanFdSendISO15765Data(SendData1,0xfd00);// STCAN 变长度 验证 CAN FD
                Delay_ms(100);
            }
        }
```

(2)can.c 主要函数说明

1 初始化 CanFD_config

```
//speed1 仲裁去波特率
//speed2 数据区波特率
void CanFD_config(uint8_t speed1,uint8_t speed2)
{
    can_parameter_struct CAN_InitSt;
      can_fdframe_struct can_fd_parameter; //CAN FD 参数
      can_fd_tdc_struct can_fd_tdc_parameter;//
     //GPIO
    can_gpio_config();
    can_struct_para_init(CAN_INIT_STRUCT, &CAN_InitSt);
    /* initialize CAN register */
    can_deinit(CAN0);



    /* initialize CAN parameters */
    CAN_InitSt.time_triggered = DISABLE;
    CAN_InitSt.auto_bus_off_recovery = DISABLE;
    CAN_InitSt.auto_wake_up = DISABLE;

//          0：使能自动重发
//          1：禁用自动重发
    CAN_InitSt.auto_retrans = ENABLE;//报文自动传输 是否开启
    CAN_InitSt.rec_fifo_overwrite = DISABLE;
    CAN_InitSt.trans_fifo_order = DISABLE;
    CAN_InitSt.working_mode = CAN_NORMAL_MODE;
```

```
    //speed1 仲裁区波特率
    CAN_InitSt.resync_jump_width=CANBAUD[speed1][0];
    CAN_InitSt.time_segment_1=CANBAUD[speed1][1];
    CAN_InitSt.time_segment_2=CANBAUD[speed1][2];
    CAN_InitSt.prescaler=CANBAUD[speed1][3];//BAUD=60m/((1+6+1)*15)    87.5%

    /* initialize CAN */
  can_init(CAN0, &CAN_InitSt);
    //can_frequency_set(CAN0, DEV_CAN_BAUD_RATE);

  //数据区 初始化
    can_struct_para_init(CAN_FD_FRAME_STRUCT, &can_fd_parameter);
  can_fd_parameter.fd_frame = ENABLE;
  can_fd_parameter.excp_event_detect = ENABLE;
  can_fd_parameter.delay_compensation = ENABLE;

  can_fd_tdc_parameter.tdc_filter = 0x04;
  can_fd_tdc_parameter.tdc_mode = CAN_TDCMOD_CALC_AND_OFFSET;
  can_fd_tdc_parameter.tdc_offset = 0x04;
  can_fd_parameter.p_delay_compensation = &can_fd_tdc_parameter;
  can_fd_parameter.iso_bosch = CAN_FDMOD_ISO;
  can_fd_parameter.esi_mode = CAN_ESIMOD_HARDWARE;

  //数据区波特率设置
    can_fd_parameter.data_resync_jump_width=CANBAUD_data[speed2][0];
    can_fd_parameter.data_time_segment_1=CANBAUD_data[speed2][1];
    can_fd_parameter.data_time_segment_2=CANBAUD_data[speed2][2];
    can_fd_parameter.data_prescaler=CANBAUD_data[speed2][3];//BAUD=60m/((1+6+1)*15)
    can_fd_init(CAN0, &can_fd_parameter);

 //can_fd_frequency_set(CAN0, 5000000);//1M   自动波特率函数
    /* configure CAN0 NVIC */
    nvic_irq_enable(CAN0_RX0_IRQn, 0, 0);
    /* enable can receive FIFO0 not empty interrupt */
    can_interrupt_enable(CAN0, CAN_INTEN_RFNEIE0);
}
```

2 设置过滤器 CAN_setAllfit

//不过滤 ID

```
void CAN_setAllfit(void)
{
        can_filter_parameter_struct    CAN_FilterInitStructure;
        can_struct_para_init(CAN_FILTER_STRUCT, &CAN_FilterInitStructure);
```

```
        CAN_FilterInitStructure.filter_number = 0;
    CAN_FilterInitStructure.filter_mode = CAN_FILTERMODE_MASK;
    CAN_FilterInitStructure.filter_bits = CAN_FILTERBITS_32BIT;

    CAN_FilterInitStructure.filter_enable = ENABLE;
          CAN_FilterInitStructure.filter_fifo_number = CAN_FIFO0;
    /* configure SFID[10:0] */
    CAN_FilterInitStructure.filter_list_high = (uint16_t)0 ;
    CAN_FilterInitStructure.filter_list_low = (uint16_t)0;
    /* configure SFID[10:0] mask */
    CAN_FilterInitStructure.filter_mask_high = (uint16_t)0;
    /* both data and remote frames can be received */
    CAN_FilterInitStructure.filter_mask_low = (uint16_t)0 ;
     can_filter_init(&CAN_FilterInitStructure);
}
```

3 设置标准过滤器 CAN1_Config16BitFilter

//标准帧过滤 ID 设置

```
void CAN1_Config16BitFilter(u16 id1, u16 id2)
{
    can_filter_parameter_struct    CAN_FilterInitStructure;
     can_struct_para_init(CAN_FILTER_STRUCT, &CAN_FilterInitStructure);

        CAN_FilterInitStructure.filter_number = 0;//过滤器组
    CAN_FilterInitStructure.filter_mode = CAN_FILTERMODE_LIST;//列表模式
    CAN_FilterInitStructure.filter_bits = CAN_FILTERBITS_16BIT;//16 位 ID 模式

    CAN_FilterInitStructure.filter_enable = ENABLE;
          CAN_FilterInitStructure.filter_fifo_number = CAN_FIFO0;
    /* configure SFID[10:0] */
    CAN_FilterInitStructure.filter_list_high = (uint16_t)id1 ;//ID
    CAN_FilterInitStructure.filter_list_low = (uint16_t)id2;//ID
    /* configure SFID[10:0] mask */
    CAN_FilterInitStructure.filter_mask_high = (uint16_t)id1;//掩码
    /* both data and remote frames can be received */
    CAN_FilterInitStructure.filter_mask_low = (uint16_t)id2 ; //掩码
        can_filter_init(&CAN_FilterInitStructure);
}
```

4 设置扩展过滤器 CAN1_Config32BitFilterExList

//列表模式下　有 0~13 共 14 组 能过滤 28 个扩展帧 ID

//iBuffer 29 位 ID 缓冲区，iSumFiter ID 个数

```
void CAN1_Config32BitFilterExList(u32 *iBuffer,u8 iSumFiter)
{
```

```
    u32 id1=0,id2=0;
        u8 i=0,iCmt=0,iSum=0;
    can_filter_parameter_struct    CAN_FilterInitStructure;
        iSum=iSumFiter/2;
        if(iSumFiter%2)
        {
            iSum+=1;
        }

        for(i=0;i<iSum;i++)
        {
            id1=iBuffer[iCmt++];
            id2=iBuffer[iCmt++];

            id1=id1>>3;
            id2=id2>>3;

            if(id1==0) id1=0xffffffff;
            if(id2==0) id2=0xffffffff;

            can_struct_para_init(CAN_FILTER_STRUCT, &CAN_FilterInitStructure);
            CAN_FilterInitStructure.filter_number = i;
            CAN_FilterInitStructure.filter_mode = CAN_FILTERMODE_LIST;
            CAN_FilterInitStructure.filter_bits = CAN_FILTERBITS_32BIT;

            CAN_FilterInitStructure.filter_enable = ENABLE;
             CAN_FilterInitStructure.filter_fifo_number = CAN_FIFO0;
            /* configure SFID[10:0] */
            CAN_FilterInitStructure.filter_list_high = (uint32_t)(id1>>13);
            CAN_FilterInitStructure.filter_list_low = (uint32_t)((id1<<3)|4);
            /* configure SFID[10:0] mask */
            CAN_FilterInitStructure.filter_mask_high = (uint32_t)(id2>>13);
            /* both data and remote frames can be received */
            CAN_FilterInitStructure.filter_mask_low = (uint32_t)((id2<<3)|4);
            can_filter_init(&CAN_FilterInitStructure);
        }
}
```

5 CANFD 发送命令 CanFdSendISO15765Data

```
 //iCanId CAN ID   ,已经移位的  FC00(7E0)
// 15765 标准帧
//cmdaddr[0]=  发送长度
```

```
//cmdaddr[1...] 发送的数据
//注: iCanId>0xFFFF  时为扩展帧
//CAN FD 协议 数据长度 1~8，12 16，20，24，32，48，64 最大长度是 64，但并不能 1~64 之间的任意长度
// 返回 1 成功 返回 0 失败
u8 CanFdSendISO15765Data(u8 *cmdaddr,u32 iCanId)     //
{
    u32 i=0;
    uint8_t TransmitMailbox=0;
    u8 Stollen=GetFdCanLen(cmdaddr[0]&0x0F);//CAN FD 长度
    //初始化参数
    can_trasnmit_message_struct     TbufMege;
    can_struct_para_init(CAN_TX_MESSAGE_STRUCT, &TbufMege);
    if(iCanId>0xFFFF)//扩展帧
    {
        TbufMege.tx_sfid = 0;
        TbufMege.tx_efid = iCanId/0X08;
        TbufMege.tx_ff = CAN_FF_EXTENDED;//
    }
    else
    {
        TbufMege.tx_sfid = iCanId/0X20;
        TbufMege.tx_efid = 0x00;
        TbufMege.tx_ff = CAN_FF_STANDARD;
    }

  TbufMege.tx_ft = CAN_FT_DATA;//数据帧
  TbufMege.tx_dlen = Stollen;//帧长度
  TbufMege.fd_flag = CAN_FDF_FDFRAME;//CAN FD
  TbufMege.fd_brs = CAN_BRS_ENABLE;//波特率切换
  TbufMege.fd_esi = CAN_ESI_DOMINANT;//CAN_ESI_DOMINANT;//;

 for(u8 Sidx = 0; Sidx < Stollen; Sidx ++)
 {
            TbufMege.tx_data[Sidx] = cmdaddr[Sidx+1];
 }



    u8 flag=0;
    //can_transmit_state_enum iFlag;
    TransmitMailbox=can_message_transmit(CAN0, &TbufMege);
    while(1)
    {
```

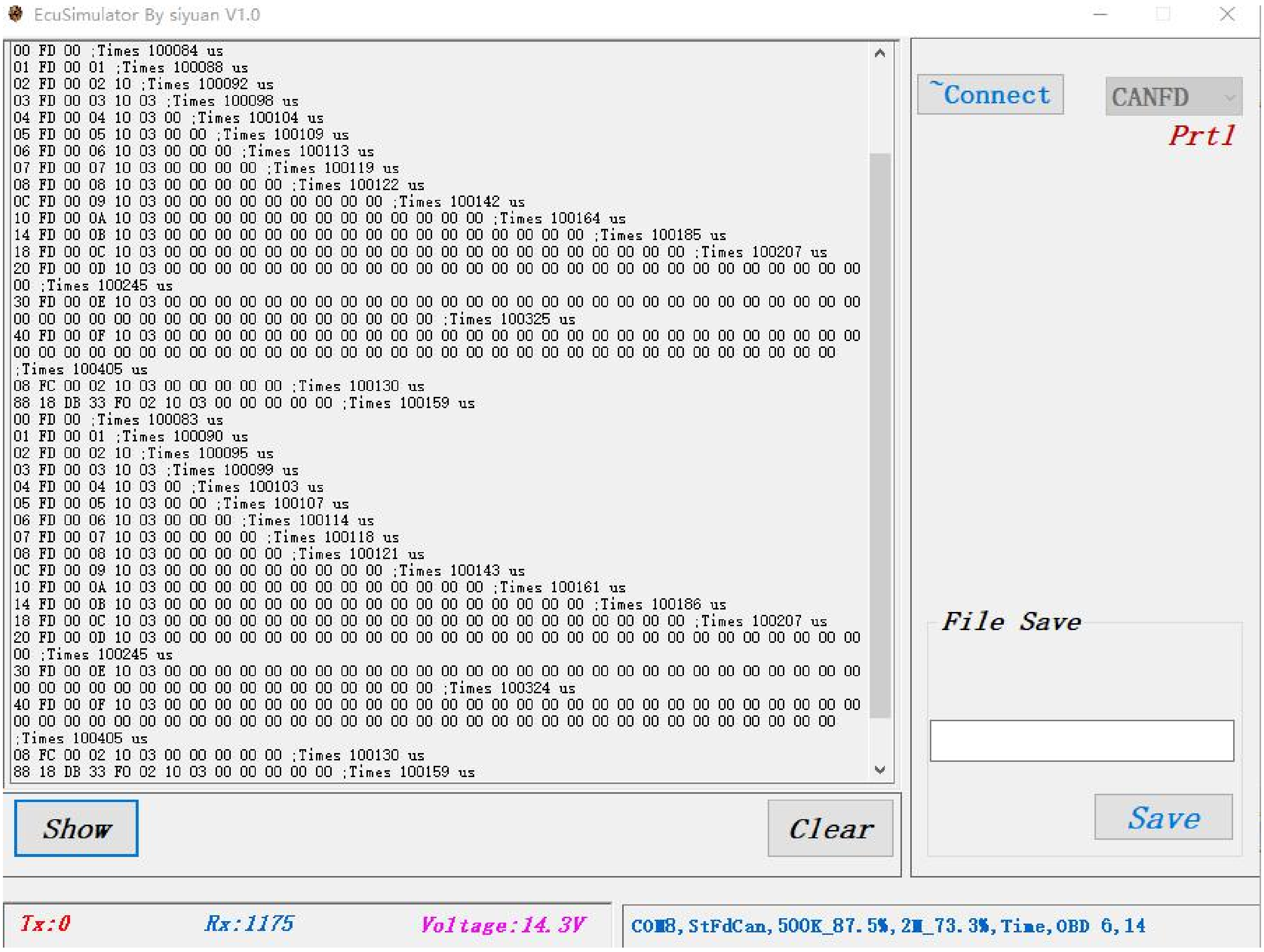
```
            flag=can_transmit_states(CAN0,TransmitMailbox);
            if(flag==CAN_TRANSMIT_OK) break;

            if(i>250)
            {//
                i=0;
                break;
            }
            Delay_us(1);
            i++;

        }

   return 1;//
}
```

4M

```
      //CAN1_Config16BitFilter(0xFC00,0xFD00);//设置过滤 ID
      CAN_setAllfit();//设置不过滤 ID

     /* main loop */
   while (1)
      {

            CanFdSendISO15765Data(SendData,0xfc00);//15765 STCAN 标准帧
            Delay_ms(100);
            CanFdSendISO15765Data(SendData,0x18DB33F1);//15765 EXCAN 扩展帧
            Delay_ms(100);
            for(i=0;i<0x10;i++)//循环发送 长度 0~64 字节的数据
            {
                SendData1[0]=i;//长度改变
                SendData1[1]=i;
                CanFdSendISO15765Data(SendData1,0xfd00);// STCAN 变长度 验证 CAN FD
                Delay_ms(100);
            }
     }
```

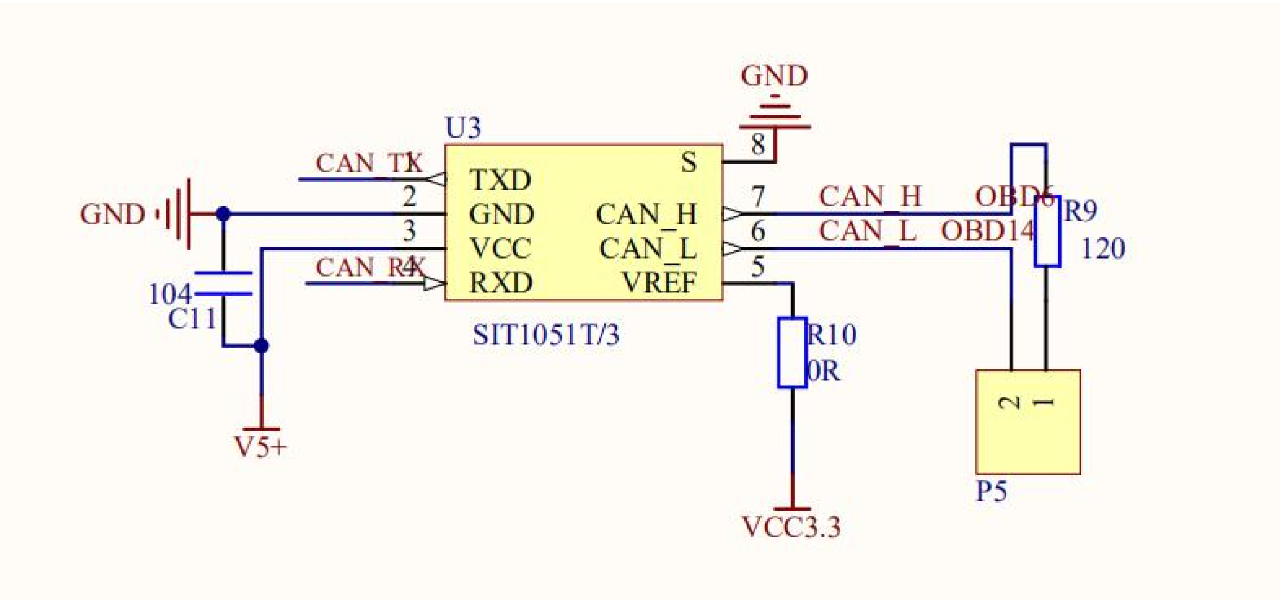
4. PC 平台效果

EcuSimulator 工具设置 CANFD 500K ,2M_73% 不过滤，显示数据如下图所示



第 6 章　CAN_FD_500K_1M_75%

1. 电路图



2. 例程说明

仲裁区波特率 500K,数据区波特率 1M 采样点 75%

用 OBD 一分 2 线接上开发板与 CAN 采集器，采集器设置波特率 500K 不过滤采样

3.软件设计

CAN1 接单片机 PB8 PB9，500K_1M 波特率循环发送 CAN 标准帧扩展帧数据

(1)main 主要流程

//初始化 IO 设置波特率 **CanFD_config(can_500k,Data_1M);//CAN FD 500k 1M ;**
//设置过滤器 **CAN_setAllfit();//设置不过滤 ID**
//canfd 发送标准帧 **CanFdSendISO15765Data(SendData,0xfc00);//15765 STCAN**
//canfd 发送扩展帧 **CanFdSendISO15765Data(SendData,0x18DB33F1);//15765 EXCAN**

Main() 主要代码

```
 uint8_t SendData[10]={0x08,0X02,0X10,0x03,0x00,0x00,0x00,0x00,0x00};
 uint8_t SendData1[100]={0x09,0X02,0X10,0x03,0x00,0x00,0x00,0x00,0x00};
CanFD_config(can_500k,Data_1M);//CAN FD 500k 1M
//CAN1_Config16BitFilter(0xFC00,0xFD00);//设置过滤 ID
CAN_setAllfit();//设置不过滤 ID

  /* main loop */
while (1)
  {

      CanFdSendISO15765Data(SendData,0xfc00);//15765 STCAN 标准帧
      Delay_ms(100);
      CanFdSendISO15765Data(SendData,0x18DB33F1);//15765 EXCAN 扩展帧
      Delay_ms(100);
      for(i=0;i<0x10;i++)//循环发送 长度 0~64 字节的数据
      {
          SendData1[0]=i;//长度改变
          SendData1[1]=i;
          CanFdSendISO15765Data(SendData1,0xfd00);// STCAN 变长度 验证 CAN FD
          Delay_ms(100);
      }
  }
```

(2)can.c 主要函数说明

1 初始化 CanFD_config

```
//speed1 仲裁去波特率
//speed2 数据区波特率
void CanFD_config(uint8_t speed1,uint8_t speed2)
{
    can_parameter_struct CAN_InitSt;
```

```
  can_fdframe_struct can_fd_parameter; //CAN FD 参数
  can_fd_tdc_struct can_fd_tdc_parameter;//
 //GPIO
can_gpio_config();
can_struct_para_init(CAN_INIT_STRUCT, &CAN_InitSt);
/* initialize CAN register */
can_deinit(CAN0);



/* initialize CAN parameters */
CAN_InitSt.time_triggered = DISABLE;
CAN_InitSt.auto_bus_off_recovery = DISABLE;
CAN_InitSt.auto_wake_up = DISABLE;

//              0：使能自动重发
//              1：禁用自动重发
CAN_InitSt.auto_retrans = ENABLE;//报文自动传输 是否开启
CAN_InitSt.rec_fifo_overwrite = DISABLE;
CAN_InitSt.trans_fifo_order = DISABLE;
CAN_InitSt.working_mode = CAN_NORMAL_MODE;

  //speed1 仲裁区波特率
  CAN_InitSt.resync_jump_width=CANBAUD[speed1][0];
  CAN_InitSt.time_segment_1=CANBAUD[speed1][1];
  CAN_InitSt.time_segment_2=CANBAUD[speed1][2];
  CAN_InitSt.prescaler=CANBAUD[speed1][3];//BAUD=60m/((1+6+1)*15)    87.5%

  /* initialize CAN */
can_init(CAN0, &CAN_InitSt);
  //can_frequency_set(CAN0, DEV_CAN_BAUD_RATE);

//数据区 初始化
  can_struct_para_init(CAN_FD_FRAME_STRUCT, &can_fd_parameter);
can_fd_parameter.fd_frame = ENABLE;
can_fd_parameter.excp_event_detect = ENABLE;
can_fd_parameter.delay_compensation = ENABLE;

can_fd_tdc_parameter.tdc_filter = 0x04;
can_fd_tdc_parameter.tdc_mode = CAN_TDCMOD_CALC_AND_OFFSET;
can_fd_tdc_parameter.tdc_offset = 0x04;
can_fd_parameter.p_delay_compensation = &can_fd_tdc_parameter;
can_fd_parameter.iso_bosch = CAN_FDMOD_ISO;
```

```
    can_fd_parameter.esi_mode = CAN_ESIMOD_HARDWARE;

    //数据区波特率设置
      can_fd_parameter.data_resync_jump_width=CANBAUD_data[speed2][0];
      can_fd_parameter.data_time_segment_1=CANBAUD_data[speed2][1];
      can_fd_parameter.data_time_segment_2=CANBAUD_data[speed2][2];
      can_fd_parameter.data_prescaler=CANBAUD_data[speed2][3];//BAUD=60m/((1+6+1)*15)
      can_fd_init(CAN0, &can_fd_parameter);

  //can_fd_frequency_set(CAN0, 5000000);//1M   自动波特率函数
    /* configure CAN0 NVIC */
    nvic_irq_enable(CAN0_RX0_IRQn, 0, 0);
    /* enable can receive FIFO0 not empty interrupt */
    can_interrupt_enable(CAN0, CAN_INTEN_RFNEIE0);
}
```

2 设置过滤器 CAN_setAllfit

//不过滤 ID

```
void CAN_setAllfit(void)
{
        can_filter_parameter_struct    CAN_FilterInitStructure;
        can_struct_para_init(CAN_FILTER_STRUCT, &CAN_FilterInitStructure);

        CAN_FilterInitStructure.filter_number = 0;
    CAN_FilterInitStructure.filter_mode = CAN_FILTERMODE_MASK;
    CAN_FilterInitStructure.filter_bits = CAN_FILTERBITS_32BIT;

    CAN_FilterInitStructure.filter_enable = ENABLE;
          CAN_FilterInitStructure.filter_fifo_number = CAN_FIFO0;
    /* configure SFID[10:0] */
    CAN_FilterInitStructure.filter_list_high = (uint16_t)0 ;
    CAN_FilterInitStructure.filter_list_low = (uint16_t)0;
    /* configure SFID[10:0] mask */
    CAN_FilterInitStructure.filter_mask_high = (uint16_t)0;
    /* both data and remote frames can be received */
    CAN_FilterInitStructure.filter_mask_low = (uint16_t)0 ;
     can_filter_init(&CAN_FilterInitStructure);
}
```

3 设置标准过滤器 CAN1_Config16BitFilter

//标准帧过滤 ID 设置

```
void CAN1_Config16BitFilter(u16 id1, u16 id2)
{
    can_filter_parameter_struct    CAN_FilterInitStructure;
```

```
 can_struct_para_init(CAN_FILTER_STRUCT, &CAN_FilterInitStructure);

        CAN_FilterInitStructure.filter_number = 0;//过滤器组
    CAN_FilterInitStructure.filter_mode = CAN_FILTERMODE_LIST;//列表模式
    CAN_FilterInitStructure.filter_bits = CAN_FILTERBITS_16BIT;//16 位 ID 模式

    CAN_FilterInitStructure.filter_enable = ENABLE;
         CAN_FilterInitStructure.filter_fifo_number = CAN_FIFO0;
    /* configure SFID[10:0] */
    CAN_FilterInitStructure.filter_list_high = (uint16_t)id1 ;//ID
    CAN_FilterInitStructure.filter_list_low = (uint16_t)id2;//ID
    /* configure SFID[10:0] mask */
    CAN_FilterInitStructure.filter_mask_high = (uint16_t)id1;//掩码
    /* both data and remote frames can be received */
    CAN_FilterInitStructure.filter_mask_low = (uint16_t)id2 ; //掩码
        can_filter_init(&CAN_FilterInitStructure);
}
```

4 设置扩展过滤器 CAN1_Config32BitFilterExList

```
//列表模式下   有 0~13 共 14 组 能过滤 28 个扩展帧 ID
//iBuffer 29 位 ID 缓冲区，iSumFiter ID 个数
void CAN1_Config32BitFilterExList(u32 *iBuffer,u8 iSumFiter)
{
    u32 id1=0,id2=0;
        u8 i=0,iCmt=0,iSum=0;
    can_filter_parameter_struct    CAN_FilterInitStructure;
        iSum=iSumFiter/2;
        if(iSumFiter%2)
        {
            iSum+=1;
        }

        for(i=0;i<iSum;i++)
        {
            id1=iBuffer[iCmt++];
            id2=iBuffer[iCmt++];

            id1=id1>>3;
            id2=id2>>3;

            if(id1==0) id1=0xffffffff;
            if(id2==0) id2=0xffffffff;

            can_struct_para_init(CAN_FILTER_STRUCT, &CAN_FilterInitStructure);
```

```
                CAN_FilterInitStructure.filter_number = i;
                CAN_FilterInitStructure.filter_mode = CAN_FILTERMODE_LIST;
                CAN_FilterInitStructure.filter_bits = CAN_FILTERBITS_32BIT;

                CAN_FilterInitStructure.filter_enable = ENABLE;
                 CAN_FilterInitStructure.filter_fifo_number = CAN_FIFO0;
                /* configure SFID[10:0] */
                CAN_FilterInitStructure.filter_list_high = (uint32_t)(id1>>13);
                CAN_FilterInitStructure.filter_list_low = (uint32_t)((id1<<3)|4);
                /* configure SFID[10:0] mask */
                CAN_FilterInitStructure.filter_mask_high = (uint32_t)(id2>>13);
                /* both data and remote frames can be received */
                CAN_FilterInitStructure.filter_mask_low = (uint32_t)((id2<<3)|4);
                can_filter_init(&CAN_FilterInitStructure);
            }
}
```

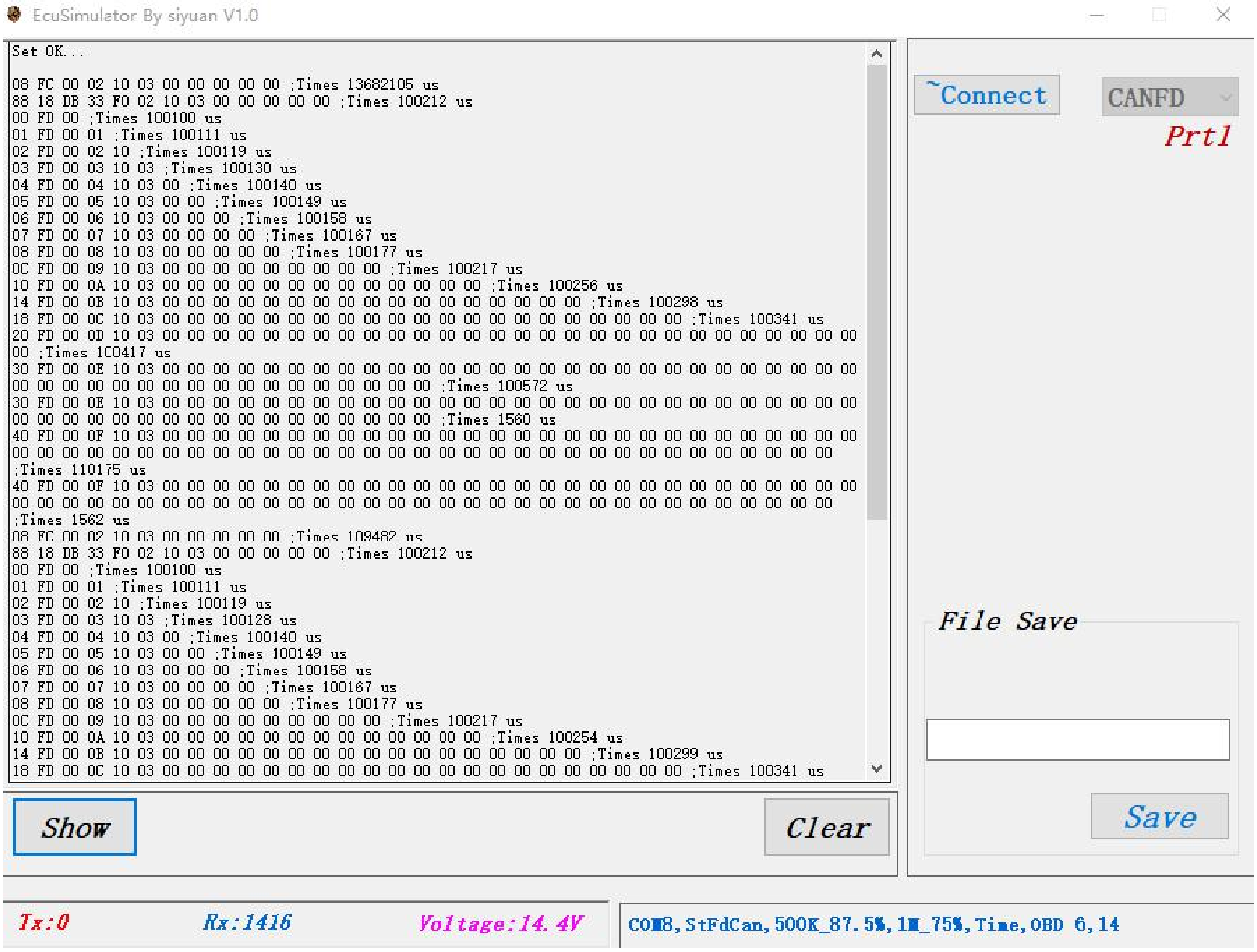
5 CANFD 发送命令 CanFdSendISO15765Data

```
 //iCanId CAN ID   ,已经移位的 FC00(7E0)
// 15765 标准帧
//cmdaddr[0]= 发送长度
//cmdaddr[1...] 发送的数据
//注: iCanId>0xFFFF   时为扩展帧
//CAN FD 协议 数据长度 1~8，12 16，20，24，32，48，64 最大长度是 64，但并不能 1~64 之间的任意长度
// 返回 1 成功 返回 0 失败
u8 CanFdSendISO15765Data(u8 *cmdaddr,u32 iCanId)     //
{
    u32 i=0;
    uint8_t TransmitMailbox=0;
    u8 Stollen=GetFdCanLen(cmdaddr[0]&0x0F);//CAN FD 长度
    //初始化参数
    can_trasnmit_message_struct     TbufMege;
    can_struct_para_init(CAN_TX_MESSAGE_STRUCT, &TbufMege);
    if(iCanId>0xFFFF)//扩展帧
    {
        TbufMege.tx_sfid = 0;
        TbufMege.tx_efid = iCanId/0X08;
        TbufMege.tx_ff = CAN_FF_EXTENDED;//
    }
    else
    {
        TbufMege.tx_sfid = iCanId/0X20;
```

```
            TbufMege.tx_efid = 0x00;
            TbufMege.tx_ff = CAN_FF_STANDARD;
        }

    TbufMege.tx_ft = CAN_FT_DATA;//数据帧
    TbufMege.tx_dlen = Stollen;//帧长度
    TbufMege.fd_flag = CAN_FDF_FDFRAME;//CAN FD
    TbufMege.fd_brs = CAN_BRS_ENABLE;//波特率切换
    TbufMege.fd_esi = CAN_ESI_DOMINANT;//CAN_ESI_DOMINANT;//;

  for(u8 Sidx = 0; Sidx < Stollen; Sidx ++)
  {
                TbufMege.tx_data[Sidx] = cmdaddr[Sidx+1];
  }



      u8 flag=0;
      //can_transmit_state_enum iFlag;
      TransmitMailbox=can_message_transmit(CAN0, &TbufMege);
      while(1)
      {
            flag=can_transmit_states(CAN0,TransmitMailbox);
            if(flag==CAN_TRANSMIT_OK) break;

            if(i>250)
            {//
                  i=0;
                  break;
            }
            Delay_us(1);
            i++;

       }

  return 1;//
}
```

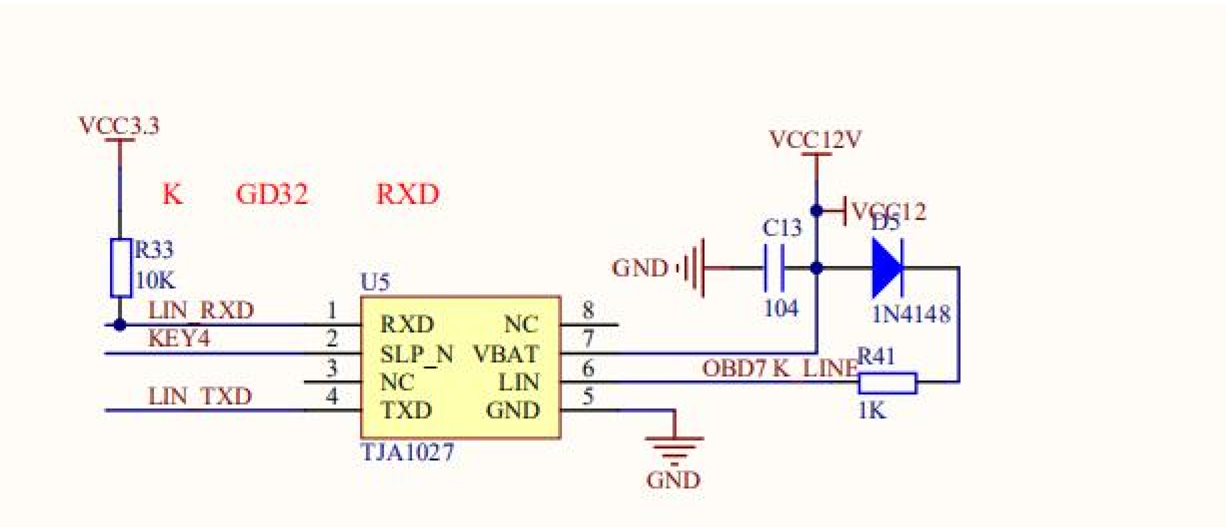
4. PC 平台效果

EcuSimulator 工具设置 CANFD 500K ,1M_75%　不过滤，显示数据如下图所示



第 7 章　KWP ISO14230

1.　电路图



2.　例程说明

物理层特性

空闲电平通常为 12V；

数据位格式为 1+8+1，无校验位；

常用波特率为 10416BPS 和 9600BPS 两种。
通讯方式
① K 线　　② K+L 线

快速模式
诊断仪在 K-线上传送一个唤醒模式(WuP)的信号。该信号在一段空闲时间（300MS）以后，以 25ms 的低电平开始。在 TWuP 的时间后，接着第一个下降沿，诊断仪发送启动通信服务的第一个位。

本例子 KWP 波特率设置为 10416，高低电平激活后，循环发送 0xCX 格式和 0x8X 格式的命令帧
注:用 OBD 一分 2 线接上开发板与 CAN/K 采集器

3.软件设计

(1)main 主要流程

//InitKinSys(0,0,10416);//25/25 拉低拉高激活系统 并初始化 K 线波特率
//SendKwp14230Frame(SendData) 发送命令

Main() 主要代码

```
uint8_t SendData[10]={0xC1,0x33,0xF1,0x81,0x66};
uint8_t SendData1[10]={0x83,0xF1,0x11,0xC1,0xEF,0x8F,0xC4};
gpio_bit_set(GPIOB,GPIO_PIN_11); //PB11=1 开启 1027
InitKinSys(0,0,10416);//25/25 拉低拉高激活系统
    /* main loop */
  while (1)
    {
        SendKwp14230Frame(SendData);//CX 开头的帧
        Delay_ms(500);
        SendKwp14230Frame(SendData1);//8X 开头的帧
        Delay_ms(500);

    }
```

(2)usart.c 主要函数说明

1 高低电平激活 Low_High_ms

```
  //波特率设置
//高低电平激活
void Low_High_ms(u16 low,u16 high)
{
      rcu_periph_clock_enable(RCU_GPIOA);
```

```
    /* configure LED2 GPIO port */
    gpio_init(GPIOA, GPIO_MODE_OUT_PP, GPIO_OSPEED_50MHZ,GPIO_PIN_2);
    /* reset LED2 GPIO pin */
    //tja1027  芯片控制脚接 PB11 ,PB11=1 时使能芯片
    gpio_bit_set(GPIOB,GPIO_PIN_11);   //PB11=1

      gpio_bit_set(GPIOA,GPIO_PIN_2); //PA2=2    拉高
    Delay_ms(300);//300ms  高电平
      //拉低拉高
      gpio_bit_reset(GPIOA,GPIO_PIN_2);
      Delay_ms(low);
     gpio_bit_set(GPIOA,GPIO_PIN_2);
      Delay_ms(high);
}
```

2 初始化 UART1_Init

//波特率设置

```
void UART1_Init(uint32_t bound)
{
      /* enable USART, GPIOA    clock */
    rcu_periph_clock_enable(RCU_GPIOA);
    rcu_periph_clock_enable(RCU_USART1);
    rcu_periph_clock_enable(RCU_AF);

        nvic_priority_group_set(NVIC_PRIGROUP_PRE1_SUB3);
    nvic_irq_enable(USART1_IRQn, 0, 1);


    /* connect port to USART1_Tx */
    gpio_init(GPIOA, GPIO_MODE_AF_PP, GPIO_OSPEED_50MHZ, GPIO_PIN_2);
    /* connect port to USART1_Rx */
    gpio_init(GPIOA, GPIO_MODE_IN_FLOATING, GPIO_OSPEED_50MHZ, GPIO_PIN_3);


    /* USART1 and USART2 baudrate configuration */
    usart_baudrate_set(USART1, bound);//波特率

    /* configure USART word length */
    usart_word_length_set(USART1, USART_WL_8BIT);//8 位数据格式

    /* configure USART stop bits */
    usart_stop_bit_set(USART1, USART_STB_1BIT);//停止位
```

```
    /* configure USART transmitter */
    usart_transmit_config(USART1, USART_TRANSMIT_ENABLE);//使能发送

    /* configure USART receiver */
    usart_receive_config(USART1, USART_RECEIVE_ENABLE);//使能接收

    /* enable USART */
    usart_enable(USART1);

    /* enable the USART interrupt */
    usart_interrupt_enable(USART1, USART_INT_RBNE);//是能接收中断
}
```

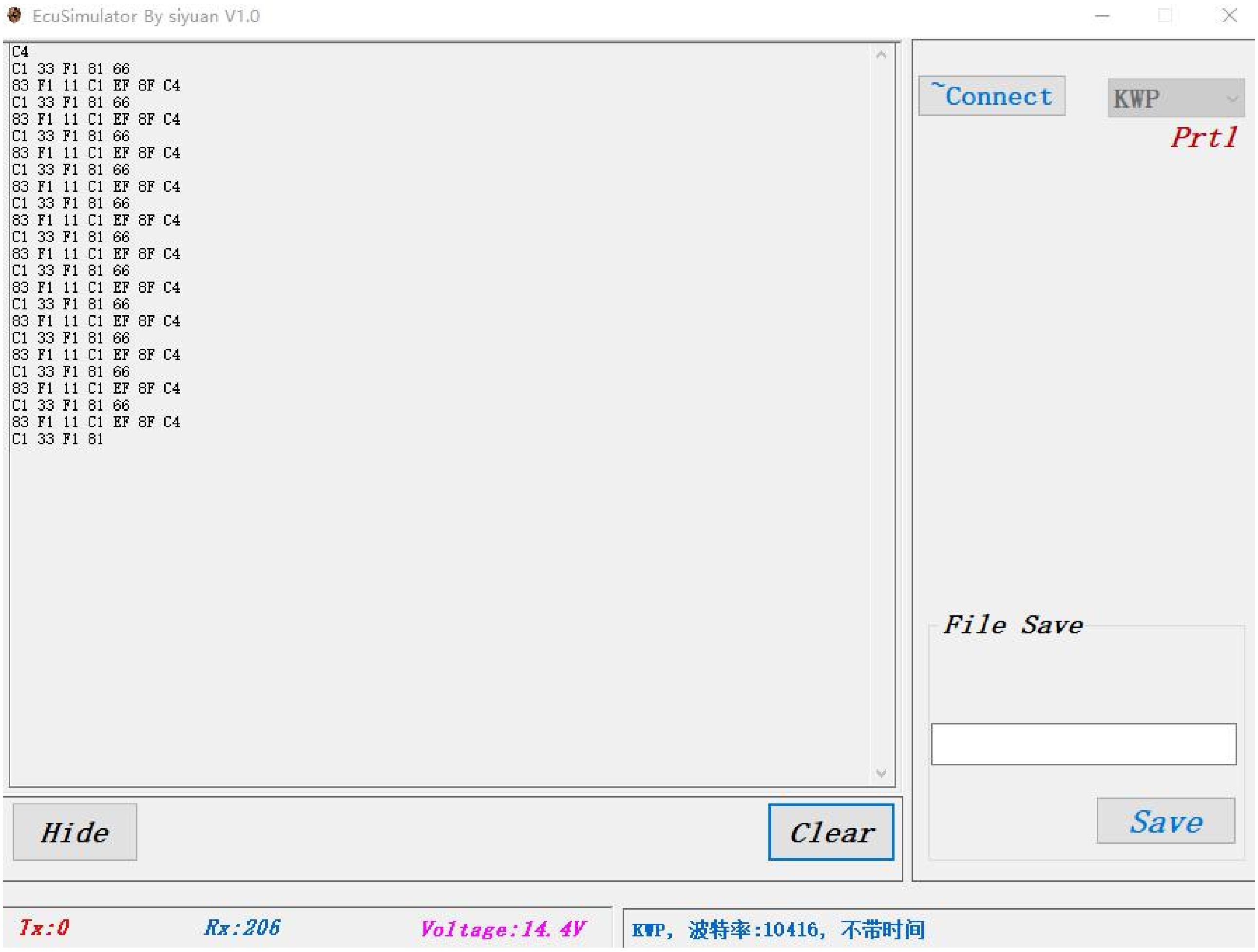
3 发送命令 SendKwp14230Frame

```
/*********************************************

函数:标准  KWP 2000   命令发送函数
参数:cmdaddr = 83 F1 11 C1 EF 8F C4
功能:
返回:
*********************************************/
uint8_t SendKwp14230Frame(uint8_t cmdaddr[])
{
     uint8_t Sidx=0,Slong=0;
     uint16_t i=0;
     uint32_t tmp=0;
     Slong=CountKwpDataLong(cmdaddr);//根据帧头自动计算长度
     cmdaddr[Slong-1]=SumDat(cmdaddr,Slong);
     //cmdaddr[Slong-1]=sum;//累加和最后一个字节
     //USART_ITConfig(USART2, USART_IT_RXNE, DISABLE);//禁中段
          /* enable the USART interrupt */
     usart_interrupt_disable(USART1, USART_INT_RBNE);
     usart_receive_config(USART1, USART_RECEIVE_DISABLE);//关闭接收
     usart_interrupt_flag_clear(USART1, USART_INT_FLAG_RBNE);//清款冲
     Delay_ms(20);
     for(Sidx=0; Sidx <Slong; Sidx ++)
     {
        KLIN_Send_ByteOne(cmdaddr[Sidx]);
           if(Sidx==(Slong-1))
           {
                Delay_us(20);
                usart_data_receive(USART1);
                usart_interrupt_flag_clear(USART1, USART_INT_FLAG_RBNE);//清款冲
```

```
        }
        Delay_ms(5);
    }
    usart_interrupt_enable(USART1, USART_INT_RBNE);
    usart_receive_config(USART1, USART_RECEIVE_ENABLE);//使能接收
    return Slong;
}
```

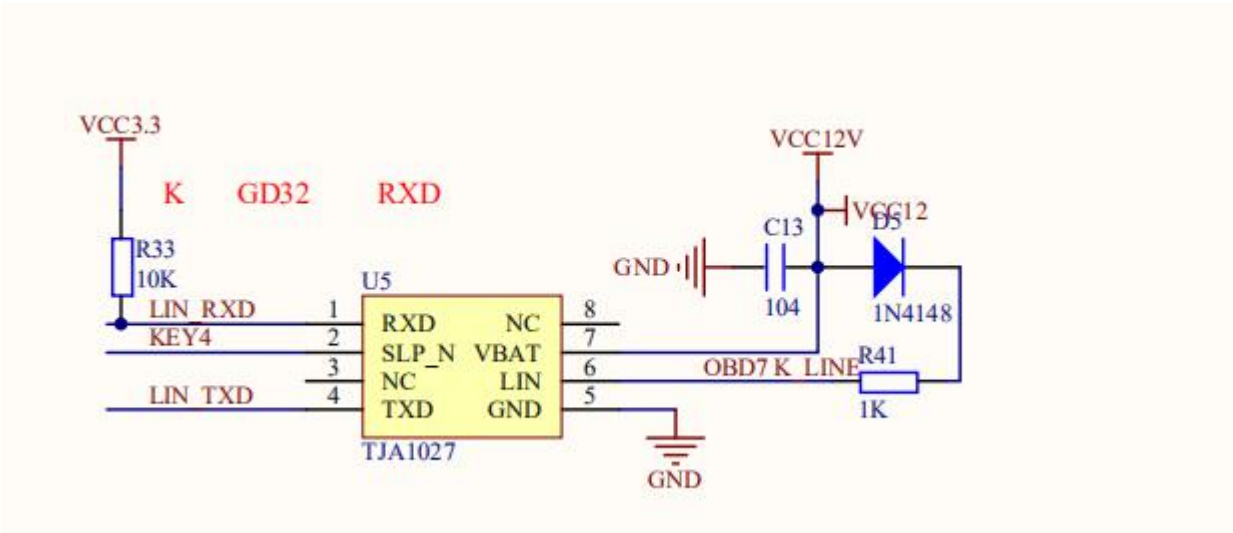
4. PC 平台效果

EcuSimulator 工具设置 KWP 波特率 10416 采集，显示数据如下图所示



第 8 章　ISO-9141-2

1. 电路图



2. 例程说明

物理层特性
空闲电平通常为 12V；
数据位格式为 1+8+1，无校验位；
常用波特率为 10416BPS 和 9600BPS 两种。
通讯方式
② K 线　　② K+L 线

采用地址吗方式激活系统，先用 5BPS 发送地址吗，ECU 会相应 55+KW1+KW2
设备对 KW2 取反发回给 ECU，ECU 对地址码取反发回给设备，完成系统初始化交互。
其中 55H 这个字节用来规定后面的通信波特率

本例子 KWP 波特率设置为 10416，地址码激活后，循环发送命令帧
注:用 OBD 一分 2 线接上开发板与 CAN/K 采集器

3. 软件设计

(1)main 主要流程

//InitKinSys(1,0x33,10416);//地址码 33 激活方式 并初始化 K 线波特率
//SendKwp9141Frame(SendData) 发送命令

Main() 主要代码

```
uint8_t SendData[10]={0x06,0x68,0x6A,0xF1,0x09,0x02,0xCE};
uint8_tSendData1[20]={0x0b,0x48,0x6B,0x10,0x49,0x02,0x01,0x30,0x31,0x32,0x33,0x50};
```

```
gpio_bit_set(GPIOB,GPIO_PIN_11);  //PB11=1  开启  1027
//UART1_Init(10416); //波特率
InitKinSys(1,0X33,10416);//地址吗激活
      /* main loop */
   while (1)
     {
         SendKwp9141Frame(SendData);//CX  开头的帧
         Delay_ms(500);
         SendKwp9141Frame(SendData1);//8X  开头的帧
         Delay_ms(500);

     }
```

(2)usart.c 主要函数说明

1 地址码激活 AddrWakeUpOneEx

```
 //激活系统
// 0  激活失败
// 1  激活成功
u8 AddrWakeUpOneEx(u8 iAddValue,uint16_t iKwpBaudVale)
{
      u8 TimeOutFlag=0;
      u8 err=0;
//    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2,ENABLE);
//    USART_ITConfig(USART2, USART_IT_RXNE, DISABLE);   //禁止接收中断
      usart_interrupt_disable(USART1, USART_INT_RBNE);
      usart_receive_config(USART1, USART_RECEIVE_DISABLE);//关闭接收
      if(SendAddFrame(iAddValue))
      {
          UART1_Init(iKwpBaudVale);
        if(KLIN_Recieve_Byte(&err,1000))//55    300MS 超时       55
        {
            if(KLIN_Recieve_Byte(&err,50))//K1
            {
                    if(KLIN_Recieve_Byte(&err,50))//k2
                    {
                            Delay_ms(30);
                            KLIN_Send_ByteOne(~err);
                            if(KLIN_Recieve_Byte(&err,60 +60)) //接收地址码取反
                            {
                                  Delay_ms(50);
                                 Delay_ms(500);
```

```
                                        TimeOutFlag=1;
                                    }
                                }
                            }
                        }
                    }
                else
                {
                    TimeOutFlag = 0;
                }
                return TimeOutFlag;
}
```

2 初始化 UART1_Init

//波特率设置

```
void UART1_Init(uint32_t bound)
{
    /* enable USART, GPIOA    clock */
    rcu_periph_clock_enable(RCU_GPIOA);
    rcu_periph_clock_enable(RCU_USART1);
    rcu_periph_clock_enable(RCU_AF);

        nvic_priority_group_set(NVIC_PRIGROUP_PRE1_SUB3);
    nvic_irq_enable(USART1_IRQn, 0, 1);

    /* connect port to USART1_Tx */
    gpio_init(GPIOA, GPIO_MODE_AF_PP, GPIO_OSPEED_50MHZ, GPIO_PIN_2);
    /* connect port to USART1_Rx */
    gpio_init(GPIOA, GPIO_MODE_IN_FLOATING, GPIO_OSPEED_50MHZ, GPIO_PIN_3);

    /* USART1 and USART2 baudrate configuration */
    usart_baudrate_set(USART1, bound);//波特率

    /* configure USART word length */
    usart_word_length_set(USART1, USART_WL_8BIT);//8 位数据格式

    /* configure USART stop bits */
    usart_stop_bit_set(USART1, USART_STB_1BIT);//停止位

    /* configure USART transmitter */
    usart_transmit_config(USART1, USART_TRANSMIT_ENABLE);//使能发送
```

```
    /* configure USART receiver */
    usart_receive_config(USART1, USART_RECEIVE_ENABLE);//使能接收

    /* enable USART */
    usart_enable(USART1);

    /* enable the USART interrupt */
    usart_interrupt_enable(USART1, USART_INT_RBNE);//是能接收中断
}
```

3 发送命令 SendKwp9141Frame

```
/*********************************************

函数:标准 9141 命令发送函数
参数:cmdaddr = 06 68 6A F1 09 02 CE
功能:
返回:
*********************************************/
uint8_t SendKwp9141Frame(uint8_t cmdaddr[])
{
    uint8_t Sidx=0,Slong=0;
    uint16_t i=0;
    uint32_t tmp=0;

    Slong=cmdaddr[0];//计算长度
    cmdaddr[Slong]=SumDat(cmdaddr+1,Slong);

    //cmdaddr[Slong-1]=sum;//累加和最后一个字节

    //USART_ITConfig(USART2, USART_IT_RXNE, DISABLE);//禁中段
        /* enable the USART interrupt */
  usart_interrupt_disable(USART1, USART_INT_RBNE);
    usart_receive_config(USART1, USART_RECEIVE_DISABLE);//关闭接收
    usart_interrupt_flag_clear(USART1, USART_INT_FLAG_RBNE);//清款冲

    Delay_ms(20);
    for(Sidx=0; Sidx <Slong; Sidx ++)
    {
      KLIN_Send_ByteOne(cmdaddr[Sidx]);
        if(Sidx==(Slong-1))
        {
            Delay_us(20);
            usart_data_receive(USART1);
```

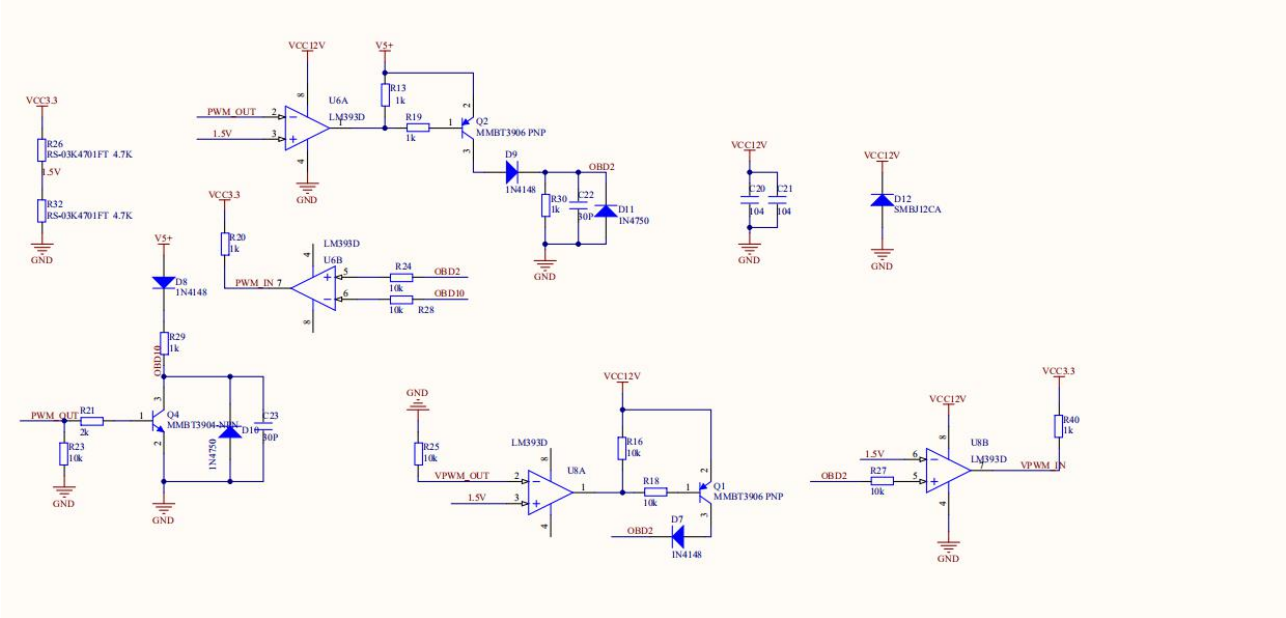
```
            usart_interrupt_flag_clear(USART1, USART_INT_FLAG_RBNE);//清款冲
        }
        Delay_ms(5);
    }
    usart_interrupt_enable(USART1, USART_INT_RBNE);
    usart_receive_config(USART1, USART_RECEIVE_ENABLE);//使能接收
    return Slong;
}
```

4. PC 平台效果

EcuSimulator 工具设置 KWP 波特率 10416 采集，显示数据如下图所示

第 9 章　J1850-VPW

1.　电路图



2.　例程说明

VPW 物理特性

采用 10.4KB/S 的波特率；通讯电平通常为 7.5V；每个字节采用 8 位二进制数形式，没有起始位、停止位和校验位；通讯引脚为 J1850 BUS+，即为 OBD-2PIN。

电平接口

VPW 协议初始电平为 0V，在第 1 帧数据前有一个 163 至 239 微秒（us）的高电平表示 SOF（帧头即数据开始标志），

接下来以不同长短的高低电平表示二进制数据 0 或 1，其中：

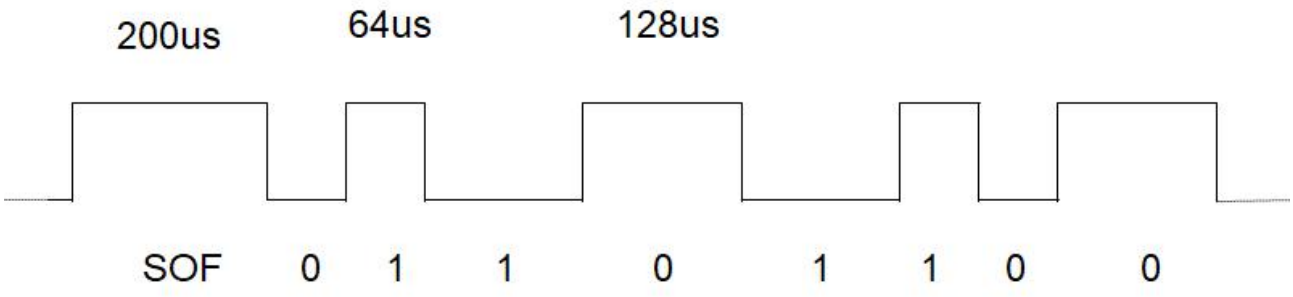
高电平宽度介于 34-96us 表示“1”，高电平宽度介于 96-163us 表示“0”，

低电平宽度介于 34-96us 表示“0”，低电平宽度介于 96-163us 表示“1”，

传输时按字节顺序，且每个字节都是高位在前，低位在后的顺序，高低电平相间用于表示传输的数据，

字节与字节之间没有间隔，传送完一帧数据之后有一个宽度大于 239us 的低电平表示 EOF（帧尾即帧结束标志）

详细如下图



3. 软件设计

(1) main 主要流程

```
//J1850VPW_Init();//vpw 初始化
//J1850_VPW_SendFrame(dat,sizeof(dat)); //发送 VPW 命令
J1850VPW_Init();//vpw 初始化
uint8_t dat[]={0x68,0x6A,0xF1,0x01,0x00,0x17};
uint8_t dat1[]={0x68,0x6A,0xF1,0x01,0x01,0x18};
//uint8_t rxdat[10] = {0};
    /* main loop */
  while (1)
    {

        J1850_VPW_SendFrame(dat,sizeof(dat));//发送一帧 VPW 命令
        Delay_ms(500);
        J1850_VPW_SendFrame(dat1,sizeof(dat1));//发送一帧 VPW 命令
        Delay_ms(500);
     }
```

(2)Vpwm.c 主要函数说明

1 VPW 初始化 J1850VPW_Init

```
//VPW_IN      PA1     TIMER1_CH1   输入
//PB7 OUT PWM 方式
void J1850VPW_Init(void)
{
     rcu_periph_clock_enable(RCU_TIMER3);
     rcu_periph_clock_enable(RCU_GPIOB);
    rcu_periph_clock_enable(RCU_AF);
    gpio_init(GPIOB, GPIO_MODE_AF_PP, GPIO_OSPEED_50MHZ, GPIO_PIN_7);
    gpio_init(GPIOB, GPIO_MODE_IPD, GPIO_OSPEED_50MHZ, GPIO_PIN_6);//PWM 引脚设置成输入 避免影响
     //gpio_bit_reset(GPIOB,GPIO_PIN_6);//0 输出
     TIM1_CH1_Cap_Init();//输入捕获的方式 获取 VPW 数据
}
```

2 VPW 数据转换为电平宽度 DatToVPW

```
uint8_t DatToVPW( VPWM_DATA *vdat, uint8_t *buf, uint8_t len )
{
    uint8_t i, j;
    uint8_t state = true;//HIGH
    uint16_t time = 200;//200;//SOF198;//     帧头
    uint16_t time1;
    vdat->Num = 0;
```

```
    for( i = 0; i < len; i++ )
    {
        for( j = 0; j < 8; j++ )
        {
            if( (buf[i]>>(7-j))&0x1)
            {
                //HIGH
                time1 = (state)?128:64;
            }
            else
            {
                //LOW
                time1 = (state)?64:128;
            }
            state = !state;
            if( !state )//每次都设置 2 个电平  所以要这样操作
            {
                //VPW  协议是一个电平翻转一个 BIT
                ToPWM( vdat, time, time1 );     //time=上升沿  ,time1=下降沿

            }
            time = time1;
        }
    }
    if( !state )
    {
        return false;
    }
    time1 = 160;//EOF
    ToPWM( vdat, time, time1 );
    return true;
}
```

3 VPW 发送发送命令 J1850_VPWM_SendAndGet

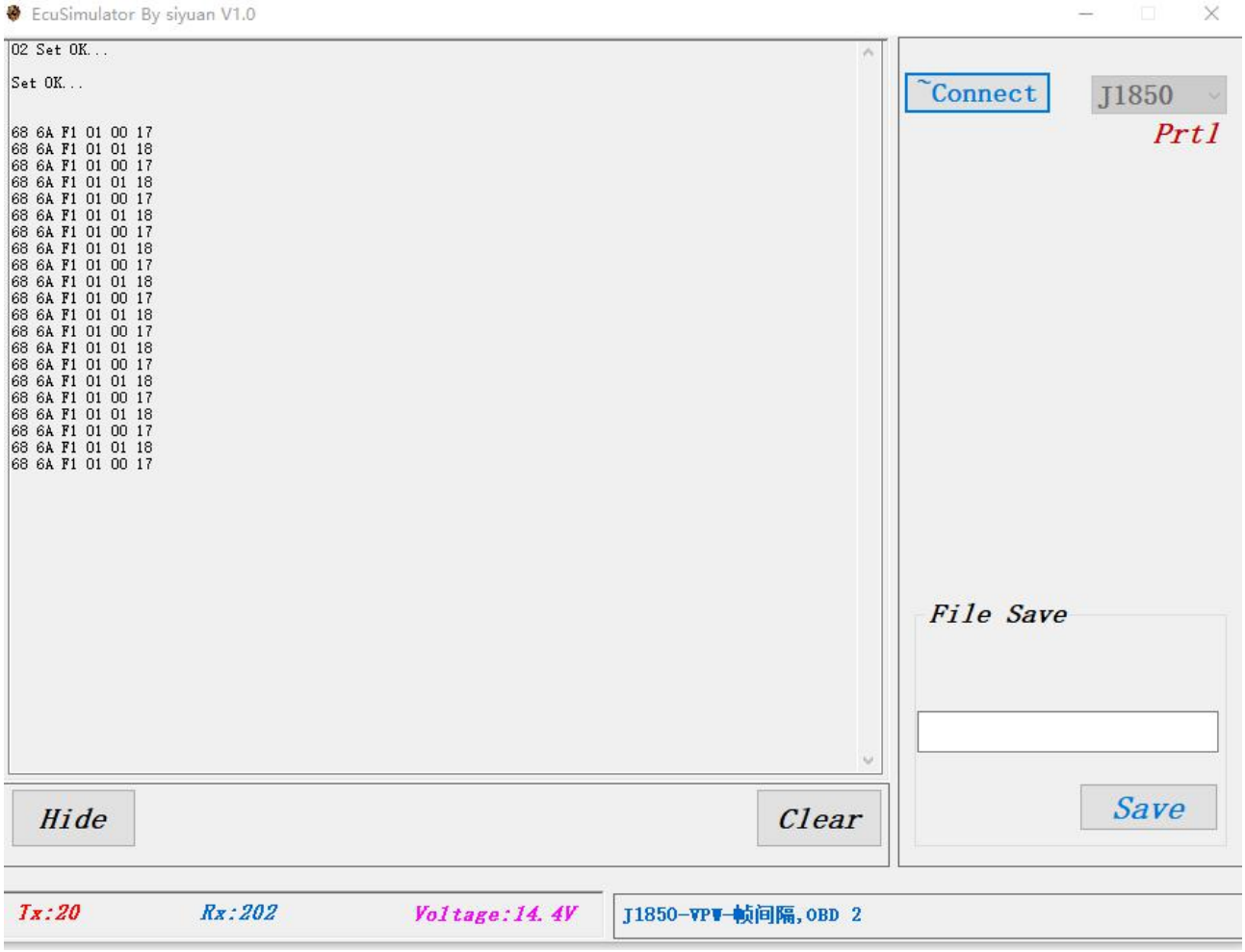
```
//-----------------------------------------------------------------------------
// Funtion: VPW PWM  脉宽输出
// Input    : vpwm  硬件对象
//                vdat  脉冲时间及占空比数据
// Output : none
// Return : none
// Info     : none
//-----------------------------------------------------------------------------
u16 J1850_VPWM_SendAndGet(VPWM_DATA *vdat )
{
```

```
 uint16_t iRet=0;
if( vdat->Num == 0 )
{
      return 0;
}
vdat->Index = 0;
 //vdat->State=VPWM_OUTPUT;
uint16_t Period = vdat->Period[vdat->Index];
 uint16_t CCR_Val = vdat->Duty[vdat->Index];
vdat->Index++;

timer_oc_parameter_struct timer_ocinitpara;
timer_parameter_struct timer_initpara;
timer_deinit(TIMER3);
/* initialize TIMER init parameter struct */
timer_struct_para_init(&timer_initpara);



/* initialize TIMER channel output parameter struct */
timer_channel_output_struct_para_init(&timer_ocinitpara);
/* CH0, CH1 and CH2 configuration in PWM mode */
timer_ocinitpara.outputstate    = TIMER_CCX_ENABLE;//是否使能输出
timer_ocinitpara.outputnstate = TIMER_CCXN_DISABLE;//
timer_ocinitpara.ocpolarity     = TIMER_OC_POLARITY_HIGH;//输出极性
timer_ocinitpara.ocnpolarity    = TIMER_OCN_POLARITY_HIGH;//输出死区延时的极性
timer_ocinitpara.ocidlestate    = TIMER_OC_IDLE_STATE_LOW;//空闲状态  输出极性
timer_ocinitpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;//空闲状态  互补输出极性
timer_channel_output_config(TIMER3, TIMER_CH_1, &timer_ocinitpara);



/* TIMER1 configuration */
          timer_initpara.period                    = Period;//周期
timer_initpara.prescaler               = 120-1;//1us  精度

timer_initpara.alignedmode             = TIMER_COUNTER_EDGE;
timer_initpara.counterdirection    = TIMER_COUNTER_UP;

timer_initpara.clockdivision         = TIMER_CKDIV_DIV1;
timer_initpara.repetitioncounter = 0;
timer_init(TIMER3, &timer_initpara);

/* CH2 configuration in PWM mode0, duty cycle 75% */
timer_channel_output_pulse_value_config(TIMER3, TIMER_CH_1, CCR_Val);//占空比    即高电平时间
```

```
    //timer_channel_output_mode_config(TIMER3, TIMER_CH_1, TIMER_OC_MODE_PWM0);
    timer_channel_output_shadow_config(TIMER3, TIMER_CH_1, TIMER_OC_SHADOW_ENABLE);//必须 ENABLE
    /* auto-reload preload enable */
    timer_auto_reload_shadow_enable(TIMER3);
     //必须接收关闭中断
     //timer_interrupt_disable(TIMER1,TIMER_INT_CH1|TIMER_INT_UP); //TIME1  接收中断关闭
     timer_disable(TIMER1);//必须关闭接收
     timer_channel_output_mode_config(TIMER3, TIMER_CH_1, TIMER_OC_MODE_PWM0);
    /* auto-reload preload enable */
    timer_enable(TIMER3);
     timer_flag_clear(TIMER3, TIMER_FLAG_CH1);//清标志
     while(vdat->State!=VPWM_IDLE)
    {
        //if( TIM_GetFlagStatus(vpwm->TxPPin.Tim,vpwm->TxPPin.TimFlagCc) == RESET )
     if(RESET == timer_flag_get(TIMER3, TIMER_FLAG_CH1)) //TIMER_INT_FLAG_CH1    TIMER_FLAG_CH1
        {
          //TIM_IT_CC3
            continue;
        }

          timer_flag_clear(TIMER3, TIMER_FLAG_CH1);//清标志
        //TIM_ClearFlag(vpwm->TxPPin.Tim,vpwm->TxPPin.TimFlagCc);
        if( !VPWM_Update(vdat) )//更新占空比
        {
                timer_disable(TIMER3);//
//              gpio_init(GPIOB, GPIO_MODE_OUT_PP, GPIO_OSPEED_50MHZ,GPIO_PIN_7);
//          gpio_bit_reset(GPIOB,GPIO_PIN_7);
            vdat->State = VPWM_IDLE;
        }
    }

          //等待接收
          //必须接收关闭中断
          iJ1850EofFlag=0;
          iReCount=0;//

          //timer_interrupt_enable(TIMER1,TIMER_INT_CH1|TIMER_INT_UP);
          timer_enable(TIMER1);//开启接收
          GetTimer6Cnt();
          return iRet;
}
```

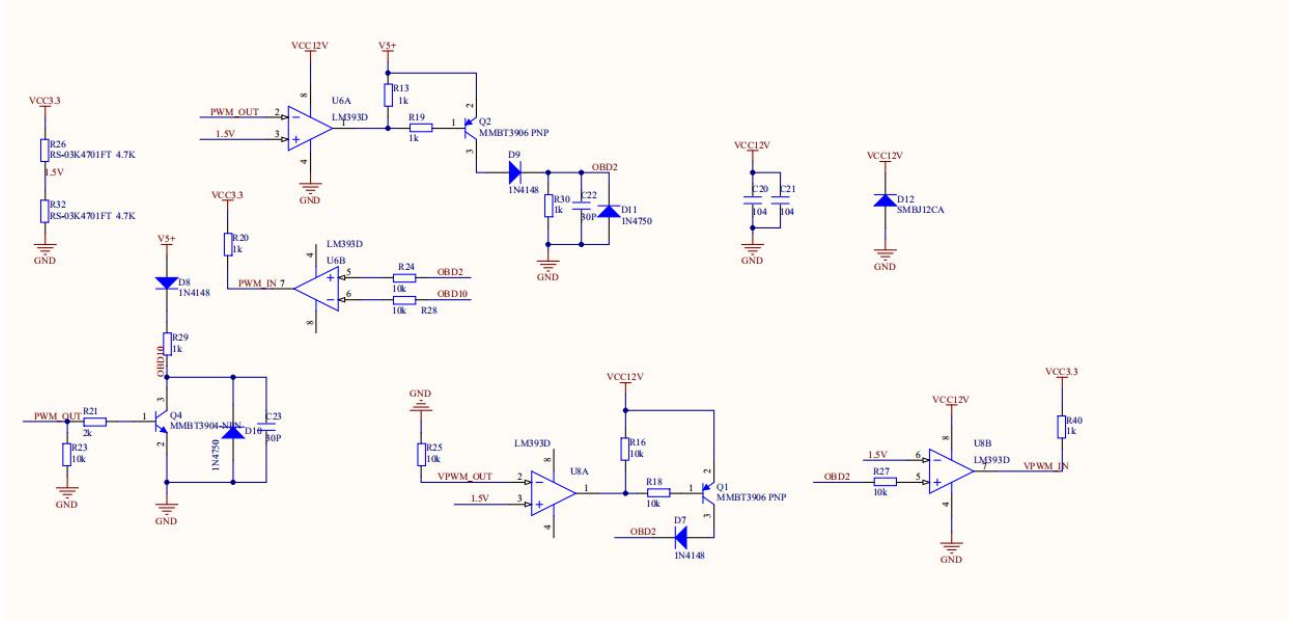
4. PC 平台效果

EcuSimulator 工具设置 J1850 VPW，显示数据如下图所示



第 10 章　J1850-PWM

1. 电路图



2. 例程说明

电平

- 5V

初始是低电平

PWM 是高低电平组合 成 BIT0 1,高位在前 低位在后

交互

- PWM 协议支持的数据交互模式是双线双向半双工方式。

位格式

- 数据位: 8, 无数据起始位和结束位 ，无校验位。

逻辑

- PWM 是 OBD 接头中的 2 和 10 号脚通信，下面我们以 2 号脚的电平信号为例来认识 bit 1 和 bit 0 。
  以 2 号脚位的电平信号为正逻辑，10 号脚电平信号为负逻辑。

帧格式:SOF+DATA+EOF

SOF 常规为 48us 高电平

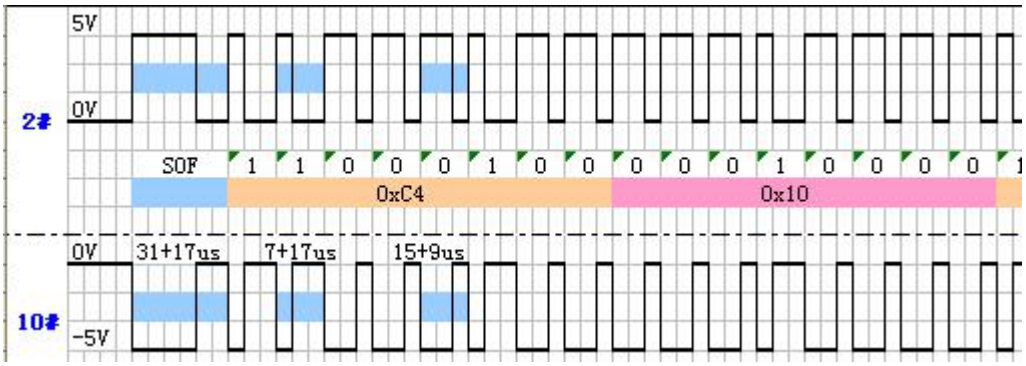
EOF 72us 低电平

DATA 格式如下，以 2 号脚电平为例子

一个周期固定 24us 表示一个 BIT

上升沿 8us,下降沿=16us 时 BIT=1

上升沿 16us,下降沿=8us 时 BIT=0



注:用 OBD 一分 2 线接上开发板与 CAN/K 采集器

3. 软件设计

(1)main 主要流程

```
//J1850PWM_Init();//PWM 初始化
//J1850_PWM_SendFrame(dat,sizeof(dat));//发送 PWM 命令
J1850PWM_Init();//PWM 初始化
uint8_t dat[]={0x61,0x6A,0xF1,0x01,0x00,0x0a};
uint8_t dat1[]={0x41,0x6B,0x10,0x41,0x00,0x56,0xF9,0x0A,0x99,0xF9};
    /* main loop */
```

```
    while (1)
     {
            J1850_PWM_SendFrame(dat,sizeof(dat));
            Delay_ms(500);
            J1850_PWM_SendFrame(dat1,sizeof(dat1));
            Delay_ms(500);
    }
```

(2) vpwm.c 函数说明

1 PWM 初始化 J1850PWM_Init

```
void J1850PWM_Init(void)
{

//       timer_oc_parameter_struct timer_ocinitpara;
//      timer_parameter_struct timer_initpara;

            rcu_periph_clock_enable(RCU_TIMER3);
            rcu_periph_clock_enable(RCU_GPIOB);
    rcu_periph_clock_enable(RCU_AF);
     //gpio_pin_remap_config(GPIO_TIMER1_PARTIAL_REMAP0,ENABLE);//
    gpio_init(GPIOB, GPIO_MODE_AF_PP, GPIO_OSPEED_50MHZ, GPIO_PIN_6);

            gpio_init(GPIOB, GPIO_MODE_IPD, GPIO_OSPEED_50MHZ, GPIO_PIN_7);//VPW 引脚设置成输入
            //gpio_bit_reset(GPIOB,GPIO_PIN_7);//0 输出
            TIM1_CH0_Cap_Init();//输入
}
```

2 PWM 数据转为电平周期 DatToPWM

3 PWM 发送发送命令 J1850_PWM_SendAndGet

```
u16 J1850_PWM_SendAndGet(VPWM_DATA *vdat )
{
     uint16_t iRet=0;
    if( vdat->Num == 0 )
    {
        return 0;
    }
    vdat->Index = 0;
            vdat->State=PWM_OUTPUT;
    uint16_t Period = vdat->Period[vdat->Index]+0;
    uint16_t CCR_Val = vdat->Duty[vdat->Index];
    vdat->Index++;
    timer_oc_parameter_struct timer_ocinitpara;
```

```
    timer_parameter_struct timer_initpara;
    timer_deinit(TIMER3);
    /* initialize TIMER init parameter struct */
    timer_struct_para_init(&timer_initpara);


    /* initialize TIMER channel output parameter struct */
    timer_channel_output_struct_para_init(&timer_ocinitpara);
    /* CH0, CH1 and CH2 configuration in PWM mode */
    timer_ocinitpara.outputstate    = TIMER_CCX_ENABLE;//是否使能输出
    timer_ocinitpara.outputnstate = TIMER_CCXN_DISABLE;//
    timer_ocinitpara.ocpolarity     = TIMER_OC_POLARITY_HIGH;//输出极性
    timer_ocinitpara.ocnpolarity    = TIMER_OCN_POLARITY_HIGH;//输出死区延时的极性
    timer_ocinitpara.ocidlestate    = TIMER_OC_IDLE_STATE_LOW;//空闲状态  输出极性
    timer_ocinitpara.ocnidlestate = TIMER_OCN_IDLE_STATE_LOW;//空闲状态  互补输出极性
    timer_channel_output_config(TIMER3, TIMER_CH_0, &timer_ocinitpara);


/* TIMER1 configuration */
    timer_initpara.period                    = Period;//周期
    timer_initpara.prescaler                 = 120-1;//1us

    timer_initpara.alignedmode             = TIMER_COUNTER_EDGE;
    timer_initpara.counterdirection    = TIMER_COUNTER_UP;

    timer_initpara.clockdivision         = TIMER_CKDIV_DIV1;
    timer_initpara.repetitioncounter = 0;
    timer_init(TIMER3, &timer_initpara);

    /* CH2 configuration in PWM mode0, duty cycle 75% */
    timer_channel_output_pulse_value_config(TIMER3, TIMER_CH_0, CCR_Val);//占空比    即高电平时间
    timer_channel_output_mode_config(TIMER3, TIMER_CH_0, TIMER_OC_MODE_PWM0);
    timer_channel_output_shadow_config(TIMER3, TIMER_CH_0, TIMER_OC_SHADOW_ENABLE);//必须  ENABLE


    /* auto-reload preload enable */
    timer_auto_reload_shadow_enable(TIMER3);

        //必须接收关闭中断
        //timer_interrupt_disable(TIMER1,TIMER_INT_CH1|TIMER_INT_UP); //TIME1  接收中断关闭
        timer_disable(TIMER1);//开启
```

```
    /* auto-reload preload enable */
    timer_enable(TIMER3);
          Delay_us(10);
          timer_flag_clear(TIMER3, TIMER_FLAG_CH0);//清标志
          while(vdat->State!=VPWM_IDLE)
    {
        //if( TIM_GetFlagStatus(vpwm->TxPPin.Tim,vpwm->TxPPin.TimFlagCc) == RESET )
       if(RESET == timer_flag_get(TIMER3, TIMER_FLAG_CH0))      //TIMER_INT_FLAG_CH1    TIMER_FLAG_CH1
        {
                          //TIM_IT_CC3
            continue;
        }
          Delay_us(1);
          timer_flag_clear(TIMER3, TIMER_FLAG_CH0);//清标志

        //TIM_ClearFlag(vpwm->TxPPin.Tim,vpwm->TxPPin.TimFlagCc);
        if( !PWM_Update(vdat) )//更新占空比
        {
                              timer_disable(TIMER3);
//                            gpio_init(GPIOB, GPIO_MODE_OUT_PP, GPIO_OSPEED_50MHZ,GPIO_PIN_7);
//                            gpio_bit_reset(GPIOB,GPIO_PIN_7);
            vdat->State = VPWM_IDLE;
        }
    }

          //等待接收
          //必须接收关闭中断
          iJ1850EofFlag=0;
          iReCount=0;//
          iHBitSum=0;
          iLBitSum=0;

          //timer_interrupt_enable(TIMER1,TIMER_INT_CH1|TIMER_INT_UP);
          timer_enable(TIMER1);//开启
          GetTimer6Cnt();

          //timer_disable(TIMER1);//开启
          //timer_interrupt_disable(TIMER1,TIMER_INT_CH1|TIMER_INT_UP); //TIME1  接收中断关闭
          return iRet;
}
```

4. PC 平台效果

EcuSimulator 工具设置 J1850 PWM，显示数据如下图所示