CH ROA平台开发交付

1	**交付时间**	2021年6月底
2	**交付版本**	工作包2, ARS1-Demo-swrelease2.0
3	**交付内容**	详见下述

3.1 平台层面

1. 基于目标板的工程配置	完成
2. 软件烧写方式， 以及A/B面升级的可行性验证	完成
3. 波形配置功能以及数据流验证	完成
4. OS，内存Layout相关系统可行性调试验证	完成
5. 客户软件状态机模块设计的可行性验证	完成

3.2 驱动层面，基于TI XWR6843AOP的底层驱动层包含

1. CAN模块的CAN FD底层驱动，测试程序以及测试方法&报告
2. Debug模块的UART驱动，测试方法以及测试报告
3. sFLash模块的qspi驱动，测试程序以及测试报告
4. PMIC模块的驱动，测试程序，测试方法，以及端口验证的测试报告

（注：目标板上的实际测试依赖于CH 第二版硬件入手时间后另行安排）

3.3 应用层面，ROA系统应用层包括

1. 加载适配TI 呼吸心跳Demo程序
2. 配置CH 发布第二版波形，非第三版新波形的软件工程

3.4 功能层面

1. 波形配置功能
2. CAN/CAN-FD配置选择功能

4	**软件发布**	服务器形式发布

1. CAN FD 驱动（3.2.1）

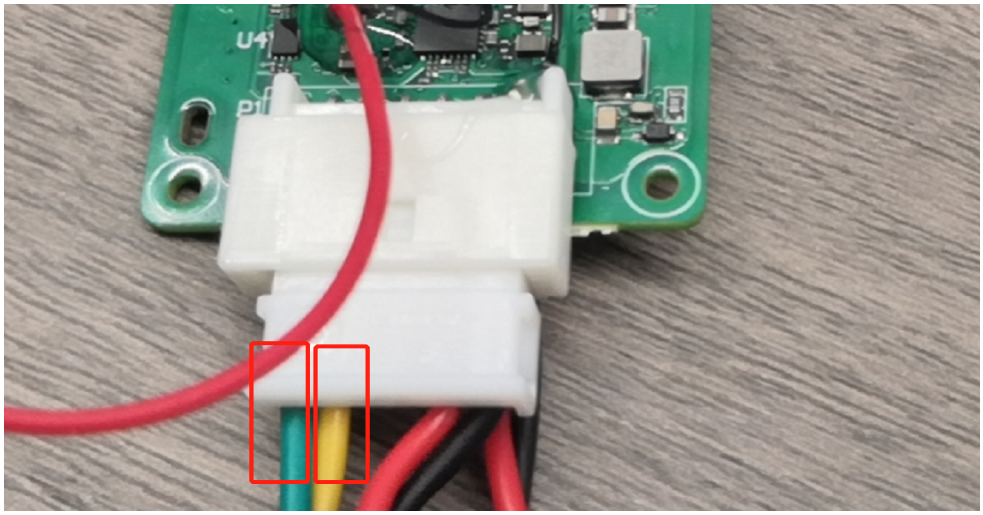
交付内容

1. 本次交付物中包含CANFD的驱动程序
2. CanFD驱动的测试程序
3. 以及本文档中来具体介绍的CAN FD测试方法
4. 测试报告

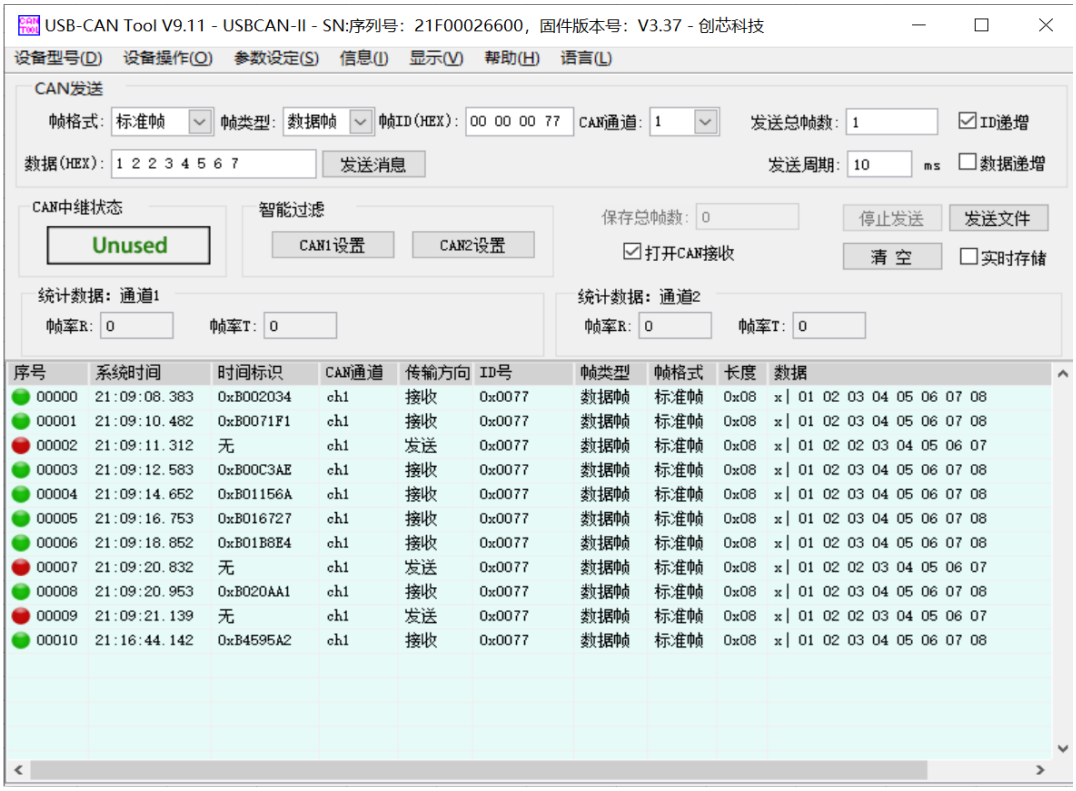
测试方法&报告

1. CAN 外设的连接：

Chuhang的开发板引出来的H，L接线如下图所示，绿色为H，黄色为L



2. pc端使用pcan接收程序，波特率为1000K，打开can信号接收程序



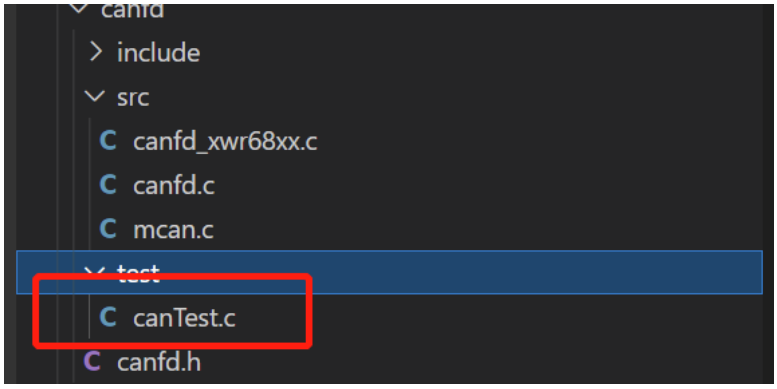
3. 开发板测试程序运行

3.1 关闭sensor，在串口执行sensorStop，提示如下

```
mmwDemo:/>sensorStop
Done
mmwDemo:/>
```

3.2 执行测试程序： canTest

canTest的测试程序默认发送1， 2， 3， 4， 5， 6， 7， 8



4. 测试结果和报告：

4.1 执行测试程序之后会看到如下测试结果：

```
Debug: Single Message Identifier Instance 0
Debug: Start Message Identifier          : 0x77
Debug: End Message Identifier            : 0x77
Debug: Direction                         : Transmit
Debug: Number of interrupts received     : 20
Debug: Number of messages processed      : 20
```

4.2 pc端接收到的输出如下：

00005	21:09:16.753	0xB016727	ch1	接收	0x0077	数据帧	标准帧	0x08	x| 01 02 03 04 05 06 07 08
00006	21:09:18.852	0xB01B8E4	ch1	接收	0x0077	数据帧	标准帧	0x08	x| 01 02 03 04 05 06 07 08

4.3 pc端发送数据配置，pc端需要设置发送数据帧ID为0x77 （测试程序中固定为0x77）

4.4 PC端发送数据（1 2 2 3 4 5 6 7），开发板接收到的数据如下：

```
Debug: Single Message Identifier Instance 0
Debug: Start Message Identifier          : 0x77
Debug: End Message Identifier            : 0x77
Debug: Direction                         : Receive
Debug: Number of interrupts received     : 2
Debug: Number of messages processed      : 2


recv:(0x1-0x2-0x2-0x3-0x4-0x5-0x6-0x7)
```

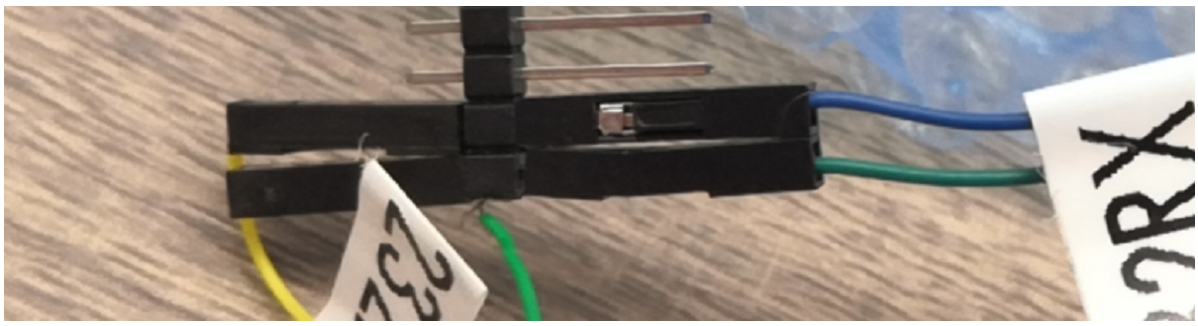
2. UART 驱动（3.2.2）

交付内容

1. 本次交付物中包含UART的驱动程序
2. 以及本文档介绍具体来介绍UART 的测试方法
3. 测试报告

测试方法&报告

1. 串口接线如下图所示，把USB的RX TX和开发板的RX TX互联，USB GROUND接地



2. 开发板上电后会看到类似如下log：

```
? ****************************************
xWR16xx Vital-Signs Monitoring Demo 03.05.00.04
****************************************
mmwDemo:/>Error: Number of averaged chirps is not power of two
Debug: Init Calibration Status = 0xffe
```

3. 测试结果&报告

该log 表示串口能正常输出，验证了串口通讯的正确性。

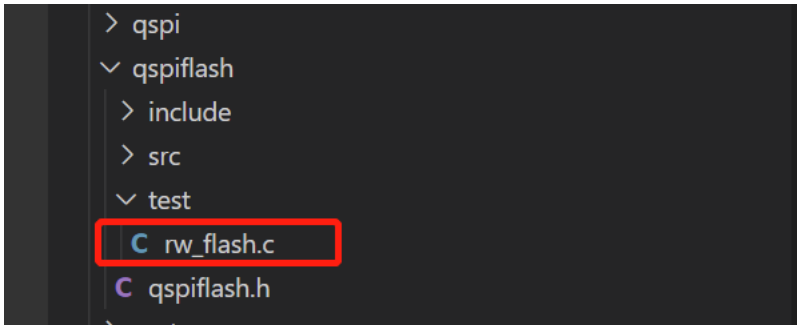
3. QSPI 驱动（3.2.3）

交付内容

1. 本次交付物中包含qspi的驱动程序
2. qspi驱动的测试程序
3. 以及本文档中来具体介绍的qspi 测试方法
4. 测试报告

测试方法&报告

通过 qspi驱动的测试程序，rw_flash.c 在命令行的调用，实现qspi的测试。



测试过程如下，读写都是按照先写后读，再对比的方式完成，
测试了qspi各种读写模式的结果。

测试的命令为：

```
mmwDemo:/>sensorStop
mmwDemo:/>rw_flash 1638400 35
mmwDemo:/>rw_flash 1638400 36
mmwDemo:/>rw_flash 1638400 37
```

向地址qspinorflash地址为0x190000的地方依次写入值为0x23 0x24 0x25
测试log流程

```
QSPIFlash read at address(0x190000) value = 0x21    读取该地址的值
QSPIFlash erase at address(0x190000)   擦除该地址
QSPIFlash write at address(0x190000) value = 0x23   写入该地址得值
QSPIFlash read again at address(0x190000) value = 0x23   读取该地址得值
```

通过测试log中每次第一次读到值均为上次写入的结果，证明该读写流程无错误。
测试log 如下：

```
******************************************
xWR16xx Vital-Signs Monitoring Demo 03.05.00.04
******************************************
mmwDemo:/>Error: Number of averaged chirps is not power of two
Debug: Init Calibration Status = 0xffe
```

```
mmwDemo:/>sensorStop
Done
mmwDemo:/>rw_flash 1638400 35
QSPIFlash Open API OK
QSPIFlash re-Open API OK
QSPIFlash Read Id API: Manufacture MACRONIX(0xc2), device type = 0x28, capacity = 0x15
QSPIFlash read at address(0x190000) value = 0x21
QSPIFlash erase at address(0x190000)
QSPIFlash write at address(0x190000) value = 0x23
QSPIFlash read again at address(0x190000) value = 0x23
QSPIFlash single write /single read API OK
QSPIFlash single write /dual read API OK
QSPIFlash single write /quad read API OK
QSPIFlash single write /mmap read test OK
QSPIFlash single write /mmap(quad read) dma read test OK
QSPIFlash single write /mmap(dual read) dma read test OK
QSPIFlash single write /mmap(single read) dma read test OK
QSPIFlash single write /mmap(auto select read) dma read test OK
QSPIFlash mmap read /write API OK
QSPIFlash mmap write /dma read test OKQSPIFlash dma read /write API OK
QSPIFlash single write /single read API OK
QSPIFlash sector erase API OK
QSPIFlash block erase API OK
Debug: QSPIFlash Instance has been closed successfully
Debug: QSPI has been closed successfully
Debug: DMA has been closed successfully
Debug: QSPIFlash Test is done!
Done
mmwDemo:/>rw_flash 1638400 36
QSPIFlash Open API OK
QSPIFlash re-Open API OK
QSPIFlash Read Id API: Manufacture MACRONIX(0xc2), device type = 0x28, capacity = 0x15
QSPIFlash read at address(0x190000) value = 0x23
QSPIFlash erase at address(0x190000)
QSPIFlash write at address(0x190000) value = 0x24
QSPIFlash read again at address(0x190000) value = 0x24
QSPIFlash single write /single read API OK
QSPIFlash single write /dual read API OK
QSPIFlash single write /quad read API OK
QSPIFlash single write /mmap read test OK
QSPIFlash single write /mmap(quad read) dma read test OK
QSPIFlash single write /mmap(dual read) dma read test OK
QSPIFlash single write /mmap(single read) dma read test OK
QSPIFlash single write /mmap(auto select read) dma read test OK
QSPIFlash mmap read /write API OK
QSPIFlash mmap write /dma read test OKQSPIFlash dma read /write API OK
QSPIFlash single write /single read API OK
QSPIFlash sector erase API OK
QSPIFlash block erase API OK
Debug: QSPIFlash Instance has been closed successfully
Debug: QSPI has been closed successfully
```

```
Debug: DMA has been closed successfully
Debug: QSPIFlash Test is done!
Done
mmwDemo:/>rw_flash 1638400 37
QSPIFlash Open API OK
QSPIFlash re-Open API OK
QSPIFlash Read Id API: Manufacture MACRONIX(0xc2), device type = 0x28, capacity = 0x15
QSPIFlash read at address(0x190000) value = 0x24
QSPIFlash erase at address(0x190000)
QSPIFlash write at address(0x190000) value = 0x25
QSPIFlash read again at address(0x190000) value = 0x25
QSPIFlash single write /single read API OK
QSPIFlash single write /dual read API OK
QSPIFlash single write /quad read API OK
QSPIFlash single write /mmap read test OK
QSPIFlash single write /mmap(quad read) dma read test OK
QSPIFlash single write /mmap(dual read) dma read test OK
QSPIFlash single write /mmap(single read) dma read test OK
QSPIFlash single write /mmap(auto select read) dma read test OK
QSPIFlash mmap read /write API OK
QSPIFlash mmap write /dma read test OKQSPIFlash dma read /write API OK
QSPIFlash single write /single read API OK
QSPIFlash sector erase API OK
QSPIFlash block erase API OK
Debug: QSPIFlash Instance has been closed successfully
Debug: QSPI has been closed successfully
Debug: DMA has been closed successfully
Debug: QSPIFlash Test is done!
Done
```

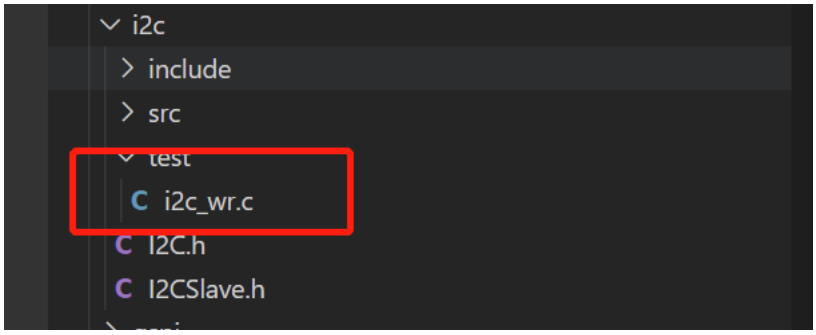
4. PMIC 驱动（3.2.4）

交付内容

1. 本次交付物中包含I2C的驱动程序
2. I2C驱动的测试程序
3. 以及本文档中来具体介绍的I2C 测试方法
4. 测试报告

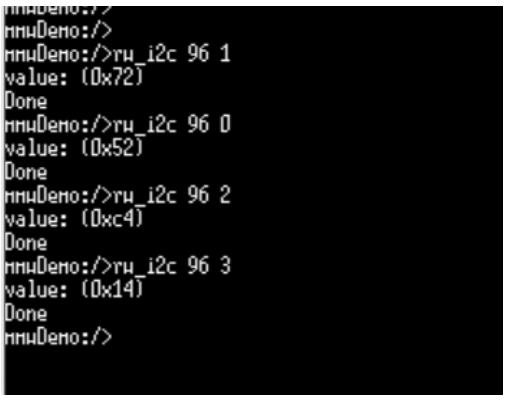
测试方法&报告

1. 添加I2Ctest测试文件，以下介绍I2Ctest测试使用方法和测试报告



2. 目前使用i2ctest 扫描i2c总线上设备，检测出有一个pmic挂载在i2c总线上，
   测试出该pmic slave address：0x60,所以使用该程序读取0x60地址如下：

2.1 开发板上电之后，打开pc端uart调试工具；
2.2 i2c读命令：rw_i2c slave_address slave_reg_address（所有参数为10进制表示）
   slave_address：从设备地址
   slave_reg_address：从设备寄存器地址



2.3 i2c写命令： rw_i2c slave_address slave_reg_address value

3. 连接示波器的测试
   在实验室环境具备示波器设备的环境下，可以把i2c的两根线引出来
   在示波器上看到i2c的波形，进行测试验证。

5. 波形配置功能（3.4.1）

交付内容

1. 本次交付物中包含ROA平台中按CH需求可调整波形信号的程序
2. 可以根据CH算法工程师需求，实时调整波形配置的接口文件
3. 以及本文档中来具体介绍的配置波形文件的具体方法
4. 测试报告

配置方法&报告

1. 客户波形配置需求（例）

xwr6843aop 波形配置需求

```
sensorStop
flushCfg
dfeDataOutputMode 1
channelCfg 15 5 0
adcCfg 2 1
adcbufCfg -1 0 0 1 1
profileCfg 0 60 250 10 40 0 0 98 1 64 2200 0 0 40
chirpCfg 0 0 0 0 0 0 0 1
chirpCfg 1 1 0 0 0 0 0 4
chirpCfg 2 2 0 0 0 0 0 1
chirpCfg 3 3 0 0 0 0 0 4
frameCfg 0 3 128 0 160 1 0
lowPower 0 0
%heatmap size MUST be 32 for data collection mode
guiMonitor 0 1 16
```

2. 波形配置的接口文件

接口文件：mmwave_cfg.h

在header文件里根据波形配置的需求，以结构体的方式对波形参数进行配置。

后续，算法工程师可以根据自己的需求，任意更改波形的配置参数，

实现波形配置与设置的高自由度需求。

3. 结构体规格：

```
struct {
        char cmdline[32];
        int32_t argc;
        char* argv[15];
} cmdline[14] = {
        {"sensorStop", 1, {"se"}},
        {"flushCfg", 1, {"fl"}},
        {"dfeDataOutputMode", 2, {"df", "1"}},
        {"channelCfg", 4, {"ch", "15", "3", "0"}},
        {"adcCfg", 3, {"ad", "2", "1"}},
        {"adcbufCfg", 6, {"ad", "-1", "0", "0", "1", "0"}},
        {"profileCfg", 15, {"pr", "0", "60.25",  "7",  "6",  "57",  "0",  "0",  "65",  "1",  "200",  "4000",  "0",  "0",  "40"}},
        {"chirpCfg", 9, {"ch", "0", "0", "0", "0", "0", "0", "0", "1"}},
        {"frameCfg", 8, {"ch", "0", "0", "2", "0", "50", "1", "0"}},
        {"lowPower", 3, {"lo", "0", "1"}},
        {"guiMonitor", 6, {"gu", "0", "0", "0", "0", "1"}},
        {"calibDcRangeSig", 6, {"ca", "-1", "0", "0", "0", "0"}},
        {"vitalSignsCfg", 10, {"vi", "0.3", "0.9",  "256",  "512",  "4",  "0.1",  "0.05",  "100000",  "300000"}},
        {"motionDetection", 5, {"mo", "1", "20", "2.0", "0"}},
        //{"sensorStart", 1, NULL},
};
```

4. 测试结果&报告

波形配置的功能实现部分，在ROA平台开发阶段已经做了效果验证，在本次交付中，

即在加载TI 呼吸心跳Demo程序后，根据CH发布的第二版波形，进行了配置调试

在TI数据链路通畅的前提下，并且在CH的板子上正常跑通。

6. CAN/CAN-FD配置选择功能（3.4.2）

交付内容

1. 本次交付物中包含ROA平台中按CH需求可选择配置CAN或CAN-FD的功能程序
2. 可以根据CH平台工程师需求，实现选择CAN/CAN-FD的不同配置的接口文件
3. 以及本文档中来具体介绍的CAN/CAN-FD配置相关的具体方法

配置与使用方法说明

1. 对代码进行设置，从而选择使用CAN或CAN-FD帧格式来传输数据的方法

参考代码路径：

vital_signs_demo\ARS1-Demo-swrelease1.0\vital_signs_68xx_mss\drivers\canfd\test\canTest.c

Line 86 & 87 通过宏控制

CANFD_MCANFrameType frameType = CANFD_MCANFrameType_FD; 选择CAN-FD帧格式

CANFD_MCANFrameType frameType = CANFD_MCANFrameType_CLASSIC; 选择CAN帧格式

```
80 /*************************************************************************
81  *************************** MCAN Global Definitions *********************
82  *************************************************************************
83
84  uint8_t                 rxData[64U];
85 uint32_t                txDataLength, rxDataLength;
86 //CANFD_MCANFrameType     frameType = CANFD_MCANFrameType_FD;
87 CANFD_MCANFrameType     frameType = CANFD_MCANFrameType_CLASSIC;
```

2. 对代码进行设置，从而实现配置传输波特率参数的方法

参考代码路径：

vital_signs_demo\ARS1-Demo-swrelease1.0\vital_signs_68xx_mss\drivers\canfd\test\canTest.c

Line 337 ~ 363: 代码仲裁域预设定了1M和500K俩个选项， CAN-FD模式下数据域预设置了5M和2M俩个选项，可根据下面描述自行修改参数。

```
337 #if 1
338     mcanBitTimingParams.nomBrp      = 0x2U;
339     mcanBitTimingParams.nomPropSeg  = 0x8U;
340     mcanBitTimingParams.nomPseg1    = 0x6U;
341     mcanBitTimingParams.nomPseg2    = 0x5U;
342     mcanBitTimingParams.nomSjw      = 0x1U;
343 #else
344     /*500Kbps NomBitRate: (40)/(((6+5+4)+1)*5)*/
345     mcanBitTimingParams.nomBrp      = 0x5U;
346     mcanBitTimingParams.nomPropSeg  = 0x6U;
347     mcanBitTimingParams.nomPseg1    = 0x5U;
348     mcanBitTimingParams.nomPseg2    = 0x4U;
349     mcanBitTimingParams.nomSjw      = 0x1U;
350 #endif
351
352 #if 0 //5 Mbps
353     mcanBitTimingParams.dataBrp     = 0x1U;
354     mcanBitTimingParams.dataPropSeg = 0x2U;
355     mcanBitTimingParams.dataPseg1   = 0x2U;
```

```
356     mcanBitTimingParams.dataPseg2   = 0x3U;
357     mcanBitTimingParams.dataSjw     = 0x1U;
358 #else //2 Mbps
359     mcanBitTimingParams.dataBrp     = 0x4U;
360     mcanBitTimingParams.dataPropSeg = 01U;
361     mcanBitTimingParams.dataPseg1   = 0x2U;
362     mcanBitTimingParams.dataPseg2   = 0x1U;
363     mcanBitTimingParams.dataSjw     = 0x1U;
```

CAN模式下仲裁域和数据域一致只需要配置，最大1M
nomBrp 波特率预分频
nomPropSeg 传播时间段
nomPseg1 相位缓冲段1
nomPseg2 相位缓冲段2
nomSjw 同步段
计算公式为： (40)/(((nomPropSeg+nomPseg1+nomPseg2)+nomSjw)*nomBrp)

CAN-FD模式下仲裁域和数据域分别配置，仲裁域最大1M，数据域最大8M
仲裁域:
nomBrp 波特率预分频
nomPropSeg 传播时间段
nomPseg1 相位缓冲段1
nomPseg2 相位缓冲段2
nomSjw 同步段
计算公式为： (40)/(((nomPropSeg+nomPseg1+nomPseg2)+nomSjw)*nomBrp)

数据域:
dataBrp 波特率预分频
dataPropSeg 传播时间段
dataPseg1 相位缓冲段1
dataPseg2 相位缓冲段2
dataSjw 同步段
计算公式为： (40)/(((dataPropSeg+dataPseg1+dataPseg2)+dataSjw)*dataBrp)

软件发布

获取项目代码

访问服务器，使用如下命令获取代码

```
git clone git@1.116.243.223:/home/git/vital_signs_demo.git
passwd: ch123456
```