タイトル: PCAN-Router FD CANFD_CAN変換

文書バージョン: 1.0

作成日: 2022年8月2日

メーカ: PEAK-System
製品: PCAN-Router FD
OS: Windows 10,11

ガイロジック株式会社
〒180-0004
東京都武蔵野市吉祥寺本町2-5-11
松栄ビル5F
Tel 0422-26-8211　Fax 0422-26-8212
http://WWW.gailogic.co.jp

1　はじめに

本アプリケーションノートでは、PCAN-Router FD を使って CANFD メッセージを CAN メッセージにして転送する方法についてについて説明します。
PCAN-Router FD のサンプルには CAN から CANFD の変換は含まれていますが（サンプル名: 10_CAN_FD)、その逆は含まれていません。
ここでは、PEAK-System の Forum にあるサンプルを使った方法を説明します。

CAN では 1 フレームのデータ長は最大 8byte ですが、CAN FD では最大 64byte に対応しています。そのため、8byte 以上のデータを持つ CAN FD メッセージを CAN メッセージに変換する場合、単純にプロトコルを変換するだけでは全てのデータを変換することができません。
本サンプルでは、CAN FD メッセージを複数の CAN メッセージに分割することでそれを解決しています。8byte 毎に 1 つの CAN メッセージに分割され、CAN ID は最初の CAN ID に 0x001 ずつインクリメントされます。

例： 変換前 CAN FD メッセージ

CANID：0x500

データ： 0x00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

変換後 CAN メッセージ

CANID：0x500

データ：0x00 01 02 03 04 05 06 07

CANID：0x501

データ：0x08 09 0A 0B 0C 0D 0E 0F

2　準備

下記を準備します。

- PCAN-Router FD
- PEAK-Flash（PEAK-System 製品のフラッシュソフトウェア）
- Microsoft Visual Studio Code
- PCAN インターフェイス（例. PCAN-USB）
- 電源（DC 8～26 V）
- CAN ケーブル（終端抵抗付）

2-1 開発環境の準備

下記の日本語資料および PCAN-Router FD マニュアルを参考に、開発環境を準備して下さい。

・PCAN-Router FD 日本語資料: https://www.gailogic.co.jp/pdf/AN_Download_PCAN-Router(_FD)_jpdoc_2_0.pdf

・PCAN-Router FD 日本語マニュアル: https://www.gailogic.co.jp/pdf/PCAN-Router_FD_211_JP_v_2_0.pdf

3 ファームウェア開発

CAN FD メッセージを CAN メッセージにするためのファームウェアを作成します。

開発には 2-1 開発環境の準備 にてインストールした Microsoft Visual Studio Code を使用します。

3-1 プログラミング

CANFD メッセージを CAN メッセージにするコードをプログラミングします。

1. 以下の PEAK-System Forum ページから、サンプルをダウンロードします。
   ファイル名: main.c
   https://forum.peak-system.com/viewtopic.php?f=199&t=5627

2. ダウンロードした main.c を以下のように編集します。
   2 行目
   #include "datatypes.h" -> #include <stdint.h>

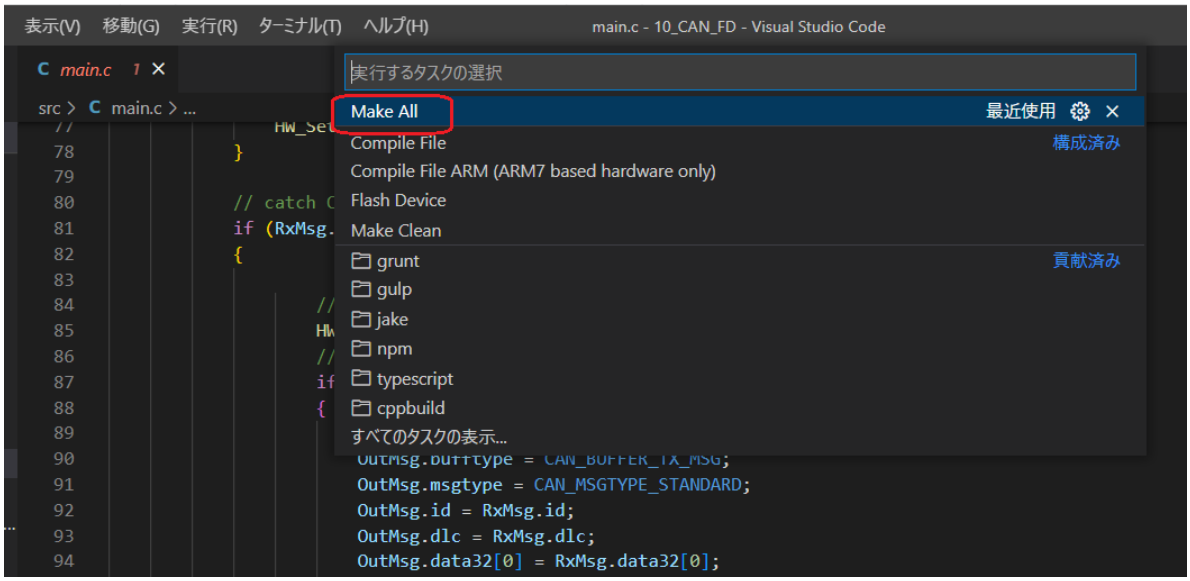
3. 編集した main.c を保存します。

4. PCAN-Router FD の適当なプログラミングサンプルを一つフォルダごとコピーして、名前を変えて複製します。
   （例: PEAK-DevPack¥Hardware¥PCAN-Router_FD¥Examples¥01_ROUTING）

5. 複製したフォルダ内の main.c を、先ほど編集したものに置き換えます。

6. Visual Studio Code を開き、ファイル -> 「フォルダーを開く」から 4.で作成したフォルダを開きます。

7. ターミナル -> タスクの実行から “Make All” を実行します。



8. 「out」フォルダに.bin ファイルが作成されます。

4 接続

図 4-1 のように接続します。

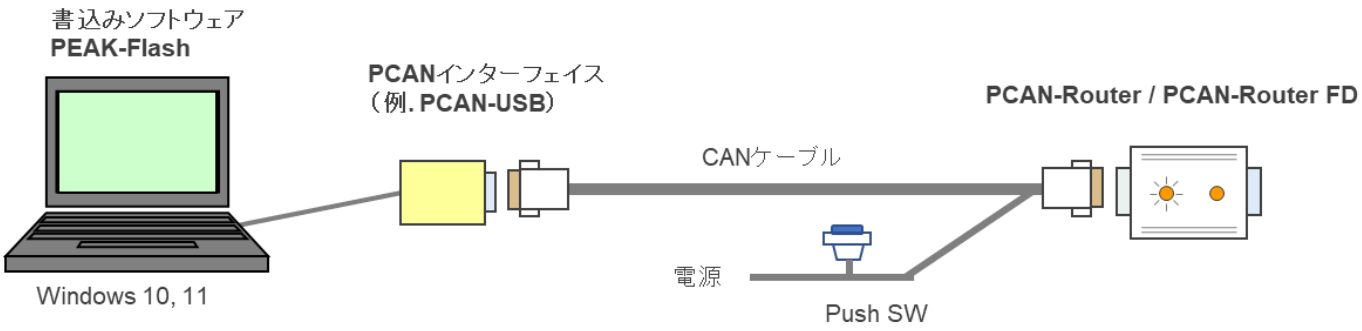


図 4-1

5 ファームウェア書き込み

1. PEAK-FLASH を起動します。

2. Welcome 画面が表示されたら Next > をクリックします。

3. “Modules connected to the CAN-Bus” を選択し、接続している CAN インターフェイスが表示されていることが確認出来たら Detect をクリックします。
   正常に PCAN-Router FD が検出されると図 4-1 のように表示されます。

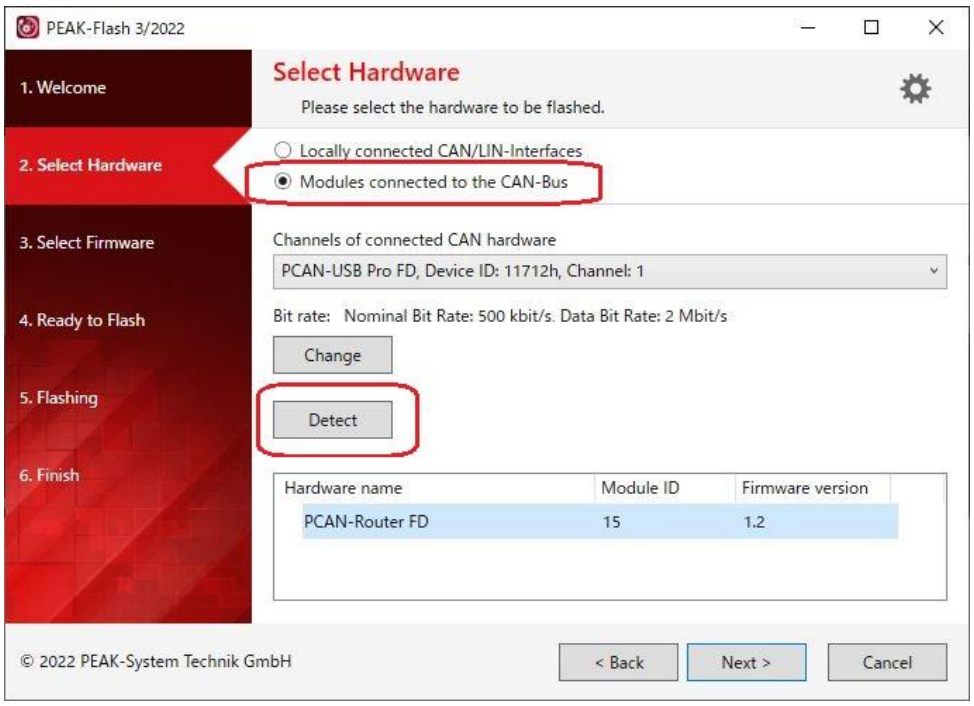


図 5-1

4. Next> をクリックすると、Select Firmware の画面が開きます。
   Firmware File: を選択し、Browse… をクリックして 3 章で作成した .bin ファイルを開きます。

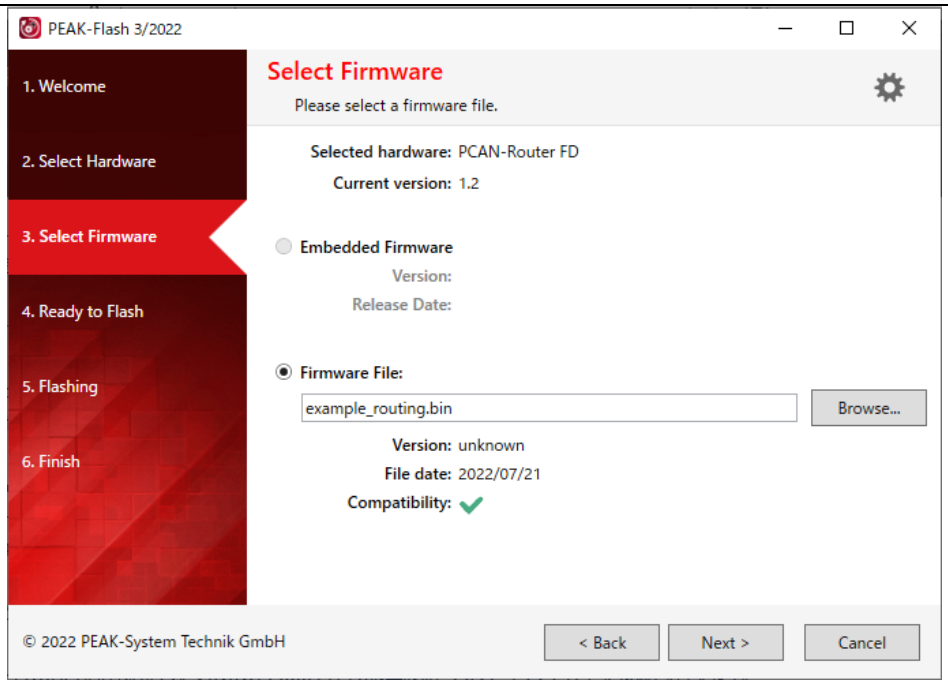


図 5-2

5. Next> をクリックすると、 Ready to Flash の画面に進みます。
   Start をクリックしてファームウェアを PCAN-Router FD に書き込みます。

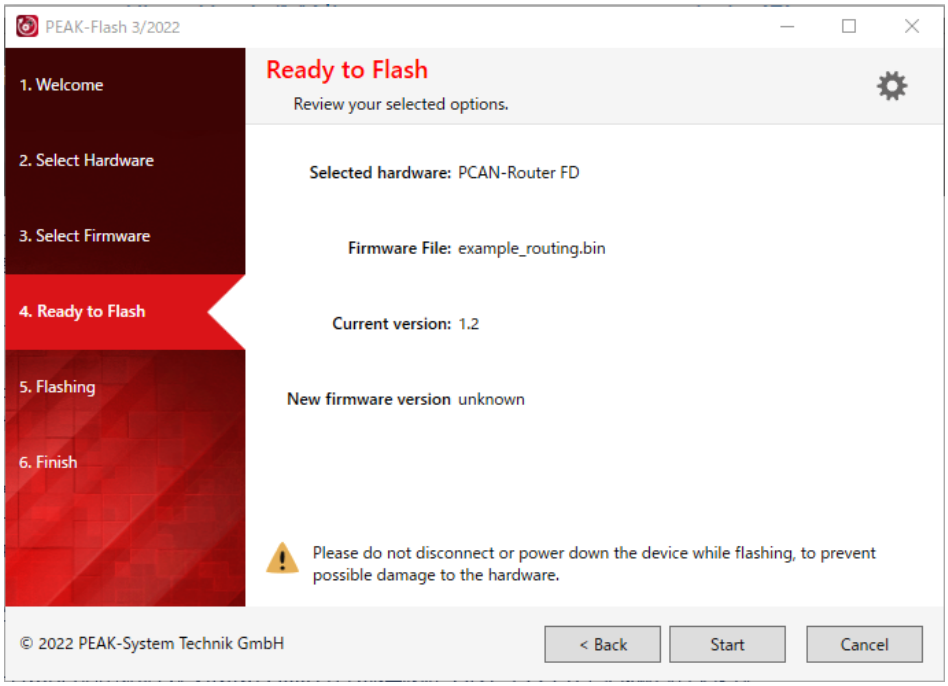


図 5-3

6. Progress: バーが 100%になり、ログに “Please disconnect and reconnect the device” と表示されたら書き込み完了です。
   Next> をクリックし、Exit で PEAK-Flash を終了します。

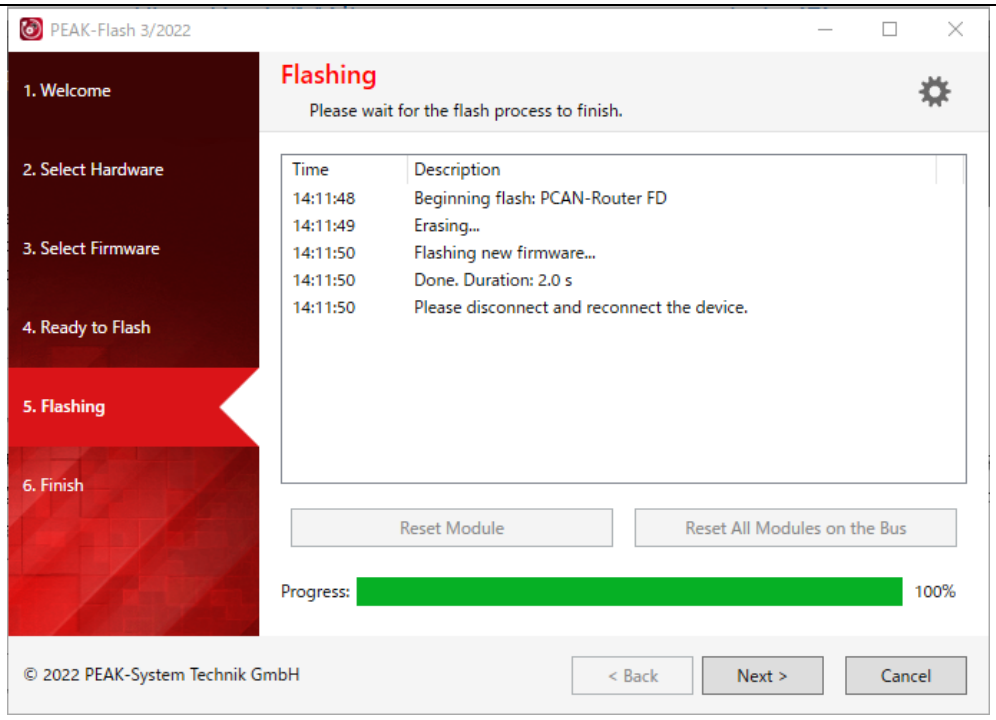


図 5-4

7. PCAN-Router FD の電源を切ります。

   その後、Boot ピンを H レベルにせず電源を投入することで CANFD -> CAN の動作が開始します。

6 動作確認

正常にファームウェアが書き込めていた場合以下の様に動作します。

・CAN1ch に入力される CAN FD メッセージ

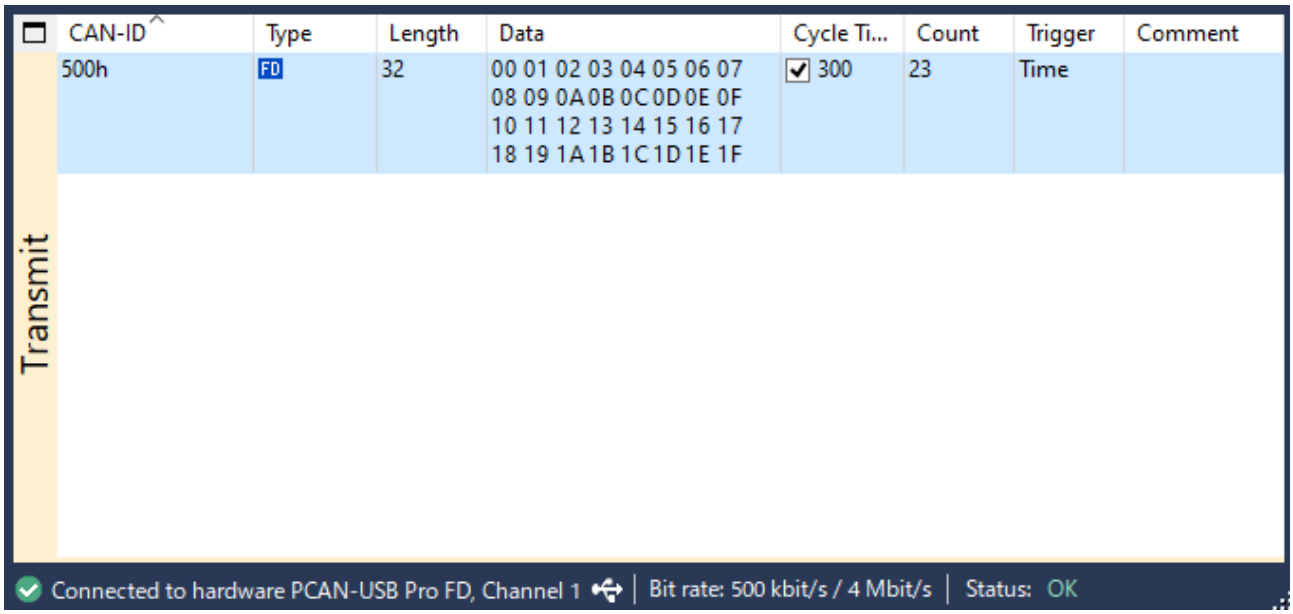


図 6-1

・CAN 2ch から出力される CAN メッセージ

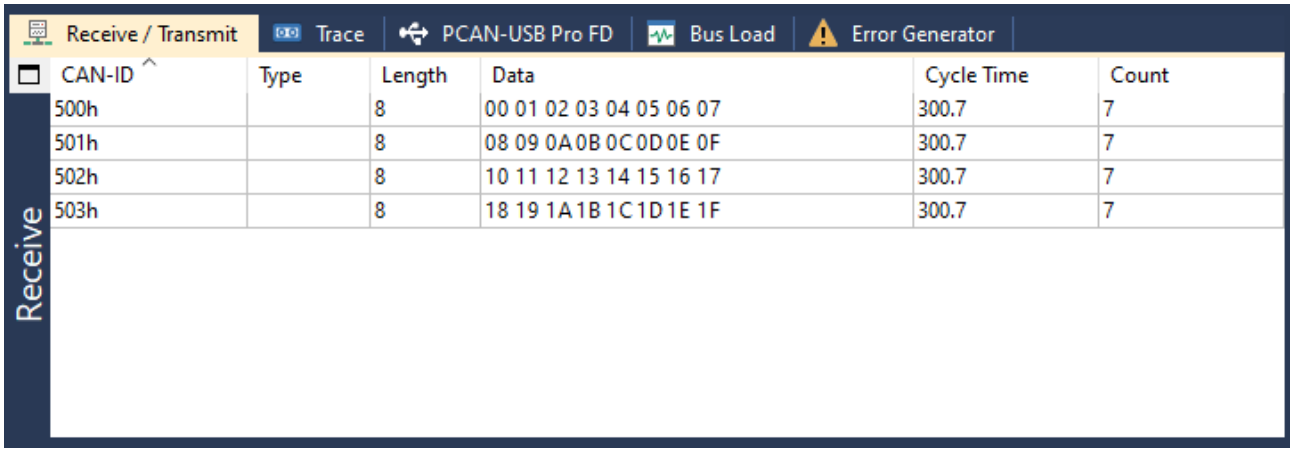


図 6-2

付録 A 編集後サンプルコード

```
#include <stdint.h>
#include "can.h"
#include "can_user.h"
#include "hardware.h"


// This example shows how to conver/split a CAN FD frame into multiple
// CAn2.0 franmes with ascending CAN-IDs


// identifier is needed by PCANFlash.exe -> do not delete
const char Ident[] __attribute__ ((used)) = { "PCAN-Router_FD"};


// variables for LED toggle
static uint8_t LED_toggleCAN1;
static uint8_t LED_toggleCAN2;
//Vriables need for conversion into CAN2.0
static int i;
int MessageLength;
uint32_t DLCValues[16] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 12, 16, 20, 24, 32, 48, 64};

// main_greeting()
// transmit a message at module start
static void   main_greeting ( void)
{
        CANTxMsg_t   Msg;
        Msg.bufftype = CAN_BUFFER_TX_MSG;
        Msg.dlc      = CAN_LEN8_DLC;
        Msg.msgtype   = CAN_MSGTYPE_STANDARD;
        Msg.id       = 0x050;
        Msg.data32[0] = 0x67452301;
        Msg.data32[1] = 0xEFCDAB89;

        // Send Msg
        CAN_Write ( CAN_BUS1, &Msg);
}

// main()
// entry point from cr_startup_lpc40xx.c
int   main ( void)
{
        // init hardware and timer 0. Timer 0 is free running
        // with 1 us resolution without any IRQ.
        HW_Init();

        // init CAN
        CAN_UserInit();

        // set green LEDs for CAN1 and CAN2
        HW_SetLED ( HW_LED_CAN1, HW_LED_GREEN);
        HW_SetLED ( HW_LED_CAN2, HW_LED_GREEN);

        // send the greeting message
        // main_greeting();

        // main loop
        while ( 1)
        {
                CANRxMsg_t   RxMsg;
                CANTxMsg_t   OutMsg;

                // process messages from CAN1
                if ( CAN_UserRead ( CAN_BUS1, &RxMsg) == CAN_ERR_OK)
                {
                        // message received from CAN1
                        LED_toggleCAN1 ^= 1;

                        if ( LED_toggleCAN1)
                        {
                                HW_SetLED ( HW_LED_CAN1, HW_LED_ORANGE);
                        }

                        else
                        {
                                HW_SetLED ( HW_LED_CAN1, HW_LED_GREEN);
                        }

                        // catch CAN FD and convert it to CAN 2.0
                        if (RxMsg.msgtype == CAN_MSGTYPE_FDF)
                        {
```

```
                                                //Now we have a real FD Frame - set the LED to RED to show it!
                                                HW_SetLED ( HW_LED_CAN1, HW_LED_RED);
                                                //If the message is 8 bytes long or less, it will only be converted.
                                                if(RxMsg.dlc <= 8)
                                                {
                                                        //Send Msg 1:1
                                                        OutMsg.bufftype = CAN_BUFFER_TX_MSG;
                                                        OutMsg.msgtype = CAN_MSGTYPE_STANDARD;
                                                        OutMsg.id = RxMsg.id;
                                                        OutMsg.dlc = RxMsg.dlc;
                                                        OutMsg.data32[0] = RxMsg.data32[0];
                                                        OutMsg.data32[1] = RxMsg.data32[1];
                                                        CAN_Write (CAN_BUS2, &OutMsg);
                                                }
                                                else
                                                {
                                                        //if it exceeds 8 bytes it will divided by 8 and split into multiple messages
                                                        MessageLength = DLCValues[RxMsg.dlc];
                                                        for(i = 0; i < (MessageLength/8); i++)
                                                        {
                                                                OutMsg.bufftype = CAN_BUFFER_TX_MSG;
                                                                OutMsg.msgtype = CAN_MSGTYPE_STANDARD;
                                                                OutMsg.id = RxMsg.id + i;
                                                                OutMsg.dlc = 8;
                                                                OutMsg.data32[0] = RxMsg.data32[(i*2)];
                                                                OutMsg.data32[1] = RxMsg.data32[(i*2+1)];
                                                                CAN_Write ( CAN_BUS2, &OutMsg);
                                                        }
                                                        // If the devision by 8 does leave a remainder, the message will be split
                                                        // into multiple messages, where one messages is shorter than 8 bytes, so
all data is
                                                        // transmitted properly.
                                                        if(MessageLength%8)
                                                        {
                                                                OutMsg.bufftype = CAN_BUFFER_TX_MSG;
                                                                OutMsg.msgtype = CAN_MSGTYPE_STANDARD;
                                                                OutMsg.id = RxMsg.id + i;
                                                                OutMsg.dlc = MessageLength%8;
                                                                OutMsg.data32[0] = RxMsg.data32[(i)*2];
                                                                OutMsg.data32[1] = RxMsg.data32[(i)*2+1];
                                                                CAN_Write (CAN_BUS2, &OutMsg);
                                                        }
                                                }
                                        }
                                }

                                // process messages from CAN2
                                if ( CAN_UserRead ( CAN_BUS2, &RxMsg) == CAN_ERR_OK)
                                {
                                        // message received from CAN2
                                        LED_toggleCAN2 ^= 1;

                                        if ( LED_toggleCAN2)
                                        {
                                                HW_SetLED ( HW_LED_CAN2, HW_LED_ORANGE);
                                        }

                                        else
                                        {
                                                HW_SetLED ( HW_LED_CAN2, HW_LED_GREEN);
                                        }
                                        // forward message to CAN1
                                        CAN_Write ( CAN_BUS1, &RxMsg);
                                }
                        }
}
```

以上