TrueSTUDIO 用 F401N_ADC_U2 の説明　　V003　　2014／11／03

ADC 入力を行うNUCLEO-F401RE のプロジェクトサンプルです。ADC CH0 ～ ADC CH5 の 6 本の入力が可能です。

NUCLEO-F401RE は STMicroelectronics 社製の Cortex-M4 ARM CPU である STM32F401RET6 を搭載した基板です。

NUCLEO の USART2_TX(PA2)と USART2_RX(PA3)は工場出荷時の状態で ST-LINK 用の CPU(U2)に接続されています。
Mini USB コネクタ CN1 を使用して VCP(仮想 COM ポート)通信を行うためです。

ST-LINK としてデバッグを実行しながら同時に VCP(仮想 COM ポート)通信を行うことができるようになっています。

Mini USB コネクタ CN1 を使用して VCP(仮想 COM ポート)通信を行うにためには、STMicroelectronics 社が提供するドライバ ST-LINK/V2-1 をインストールする必要があります。

PC アプリケーションの Access_SerialPort を使用して UART(非同期シリアル通信)により、ADC 入力値の表示を行うことができます。

試用版の開発ツール Atollic TrueSTUDIO for ARM Lite で作成したプロジェクトです。
ビルド可能なプログラムのコードサイズが 32Kbyte 以内の制限があります。

プログラムの開始番地は 0x08000000 です。デバッグが可能です。

目次

1. ADC 入力について

1. 1. 入力電圧

分解能 12bit の A/D コンバータ入力です。

ADC 入力電圧 と ディジタル値 との対応は理論上、

入力電圧　　0V : 0

入力電圧 +3.3V : 4095

です。

実際はオフセットと傾きがあり、ぴったり この数値にはなりません。通常、ソフトまたはハードで補正を行います。

1. 2. ADC 入力に使用する信号

NUCLEO=F401RE の ADC 入力に使用する信号は以下の ADC_CH0 ～ ADC_CH5 の 6 本です。

ADC 信号表

信号名	CPU 機能名	CPU 信号名	NUCLEO=F401RE コネクタピン番号
ADC_CH0	ADC1_IN10	PC0	CN7-38
ADC_CH1	ADC1_IN11	PC1	CN7-36
ADC_CH2	ADC1_IN12	PC2	CN7-35
ADC_CH3	ADC1_IN13	PC3	CN7-37
ADC_CH4	ADC1_IN14	PC4	CN10-34
ADC_CH5	ADC1_IN15	PC5	CN10-6
AGND	VSSA	VSSA	CN10-32

2. 使用する信号

2. 1. UART 信号

UART 通信に使用する信号と接続相手との接続は以下の通りです。

U5 の USART2_TX(PA2)と USART2_RX(PA3)はそれぞれ solder bridge を介して U2 または CN10 に接続されます。

UART 信号表

番号	CPU 機能名 U5	CPU 信号名	U2 との接続	基板コネクタピン番号	方向	接続相手の 信号名
1	USART2_TX	PA2	SB13-PA3	SB63-CN10-35	--- >	RxD
2	USART2_RX	PA3	SB14-PA2	SB62-CN10-37	< ---	TxD
3	GND	GND		CN10-20	< --- >	GND

2. 2. VCP(仮想 COM ポート)通信に使用する場合

工場出荷時の状態では U5 の PA2 と PA3 は U2 側に接続されています。
PC と Mini USB コネクタ CN1 とを接続して VCP(仮想 COM ポート)通信を行うことができます。

Mini USB コネクタ CN1 を ST-LINK として使用し、デバッグを行いながら同時に VCP(仮想 COM ポート)通信を行うことができます。(STMicroelectronics 社が提供するドライバ ST-LINK/V2-1 のインストールが必要です。)

アプリケーション Access_SerialPort を使用して、VCP(仮想 COM ポート)通信により PC から送信する ADC コマンドを受信して、コマンドに従って ADC 入力値を送信します。

PC と NUCLEO_F401RE との接続は以下のようになります。

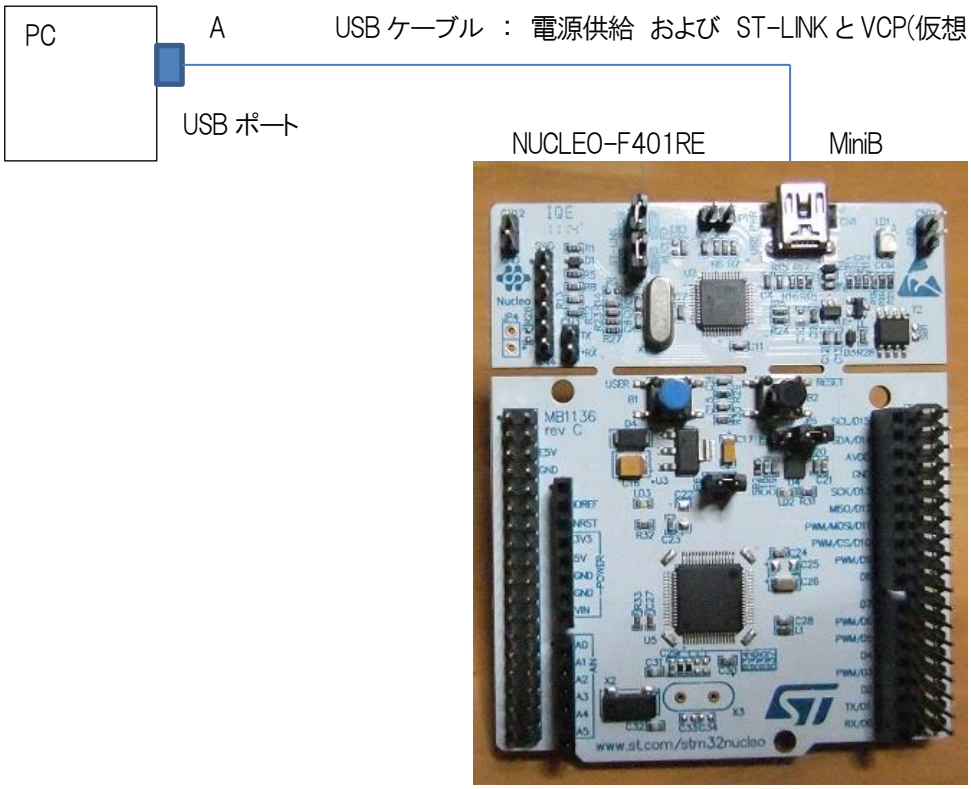


図2. 2.

2. 3. CN10 に接続して使用する場合

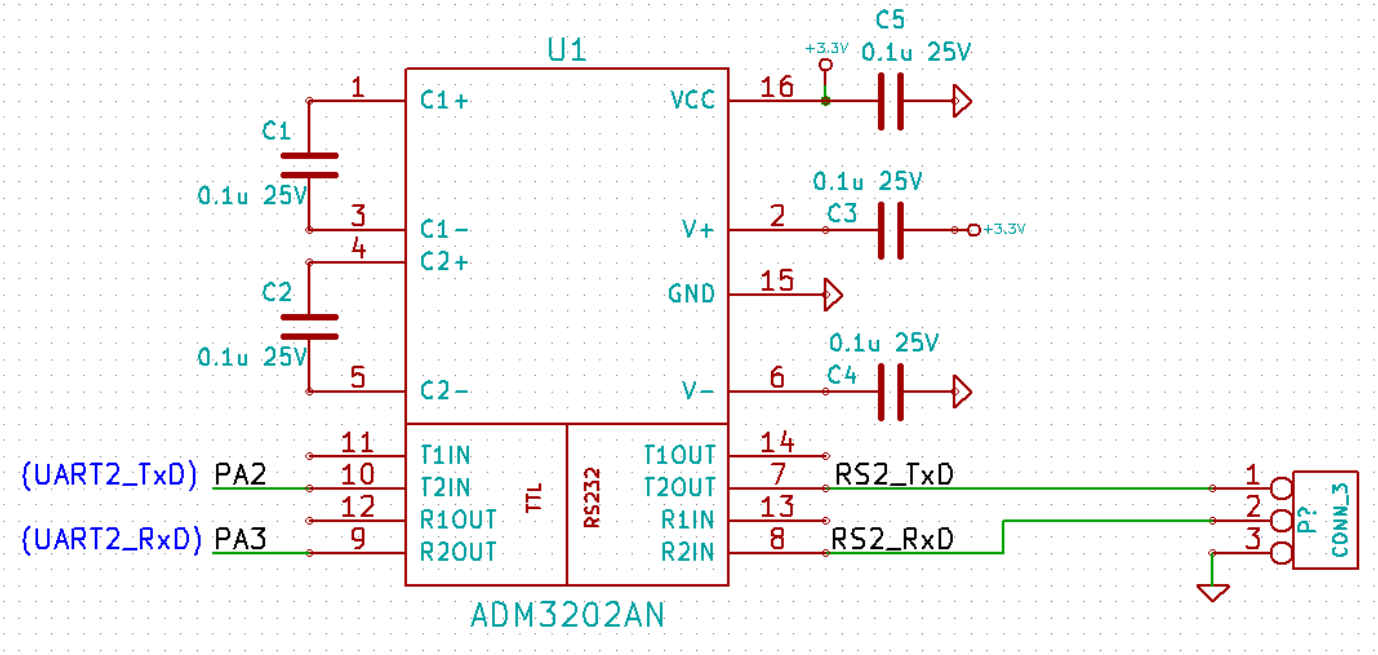
Solder bridge の SB13 と SB14 を解放して、SB63 と SB62 を接続状態にして CN10-35(PA2)と CN10-37(PA3)に接続します。

CMOS レベルどうしの信号を直接接続して通信を行うことができます。

UART 信号に RS232C ドライバを接続して RS232C レベルの信号どうしで接続すれば RS232C 通信を行うことができます。

1） RS232C ドライバ

RS232C ドライバの例を以下に示します。



RS232C ストレートケーブル接続の場合の DSUB-9S への接続例を以下に示します。
この例では DSUB-9S には TxD, RxD, GND の3本の信号以外は接続されません。

番号	RS232C ドライバ	方向	ドライバ側 D-SUB 9S	ストレートケーブル側 D-SUB 9P
1	RS2_TxD	--- >	2	2 RxD
2	RS2_RxD	< ---	3	3 TxD
3	GND	< --- >	5	5 GND

2） RS232C ケーブル接続図

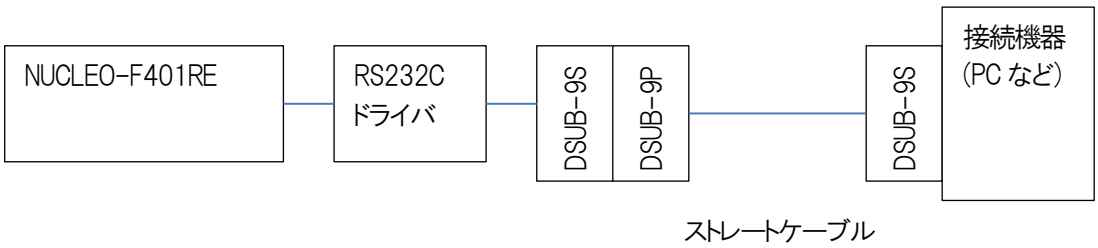


図2. 3.　2）

3） UART 通信確認のための接続例

アプリケーション Access_SerialPort を使用して、RS232C 通信により PC から送信する ADC コマンドを受信して、コマンドに従って ADC 入力値を送信します。

動作確認を行うための接続は以下のようになります。この例は RS232C レベルで UART 信号を接続しています。

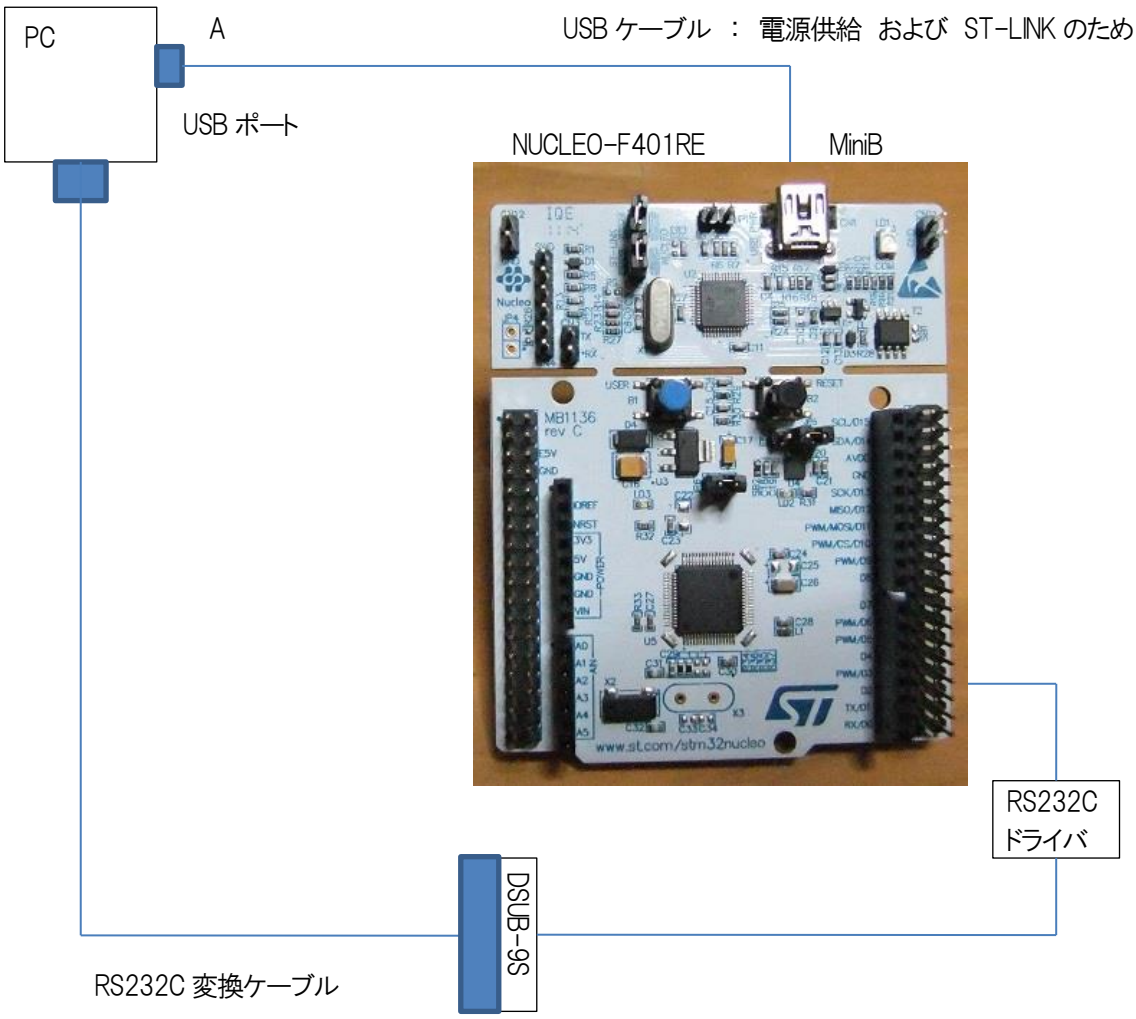


図2. 3.　3）

3.　プログラム実行時の動作

1）プログラムを実行すると基板上の LED　LD2(緑)が　1 秒点灯、2 秒消灯　で点滅します。

2）UART2 のチャンネルでデータ受信待ちを行います。

3）有効なコマンドを受信した場合、以下のように動作します。

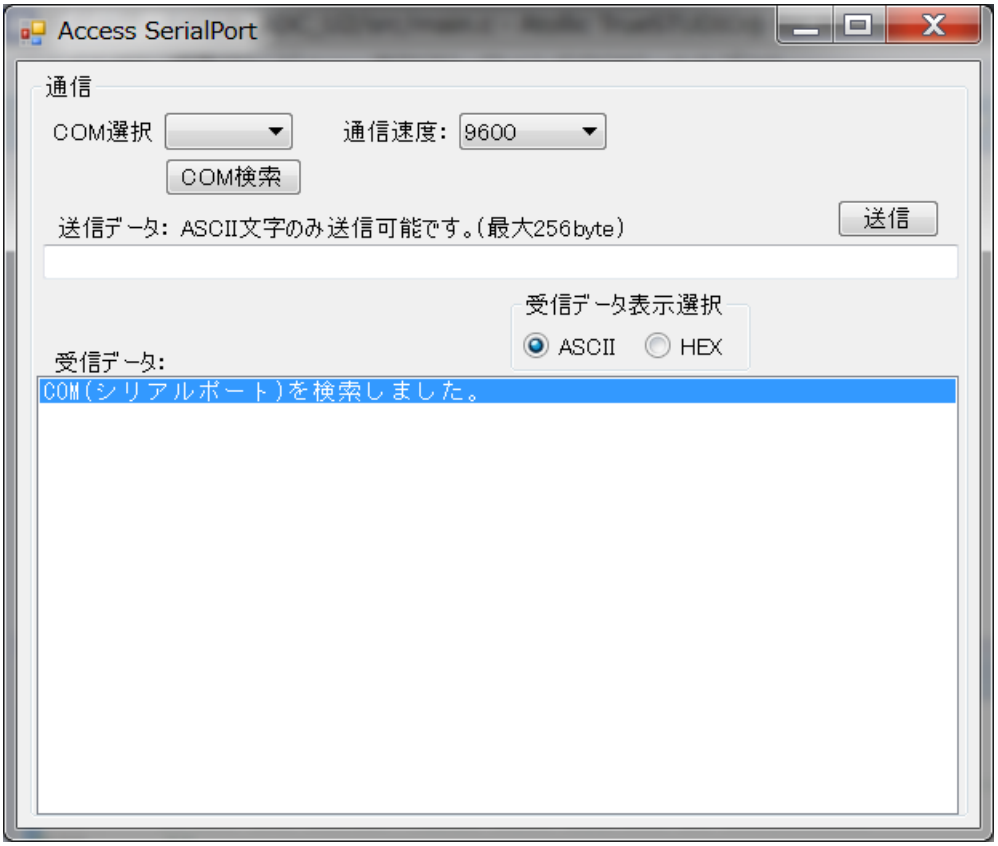
番号	受信コマンド	動作	対応する ADC 入力
1	ADC Input STOP	ADC データ送信停止	
2	ADC Input CH0	ADC CH0 データ送信	PC0 : ADC1_IN10
3	ADC Input CH1	ADC CH1 データ送信	PC1 : ADC1_IN11
4	ADC Input CH2	ADC CH2 データ送信	PC2 : ADC1_IN12
5	ADC Input CH3	ADC CH3 データ送信	PC3 : ADC1_IN13
6	ADC Input CH4	ADC CH4 データ送信	PC4 : ADC1_IN14
7	ADC Input CH5	ADC CH5 データ送信	PC5 : ADC1_IN15
8	ADC Input ALL	ADC　全 CH データ送信	PC0 – PC5 : ADC1_IN10 – ADC1_IN15

4.　ADC 入力のデータ確認

アプリケーション Access_SerialPort を使用して、UART(非同期シリアル通信)により PC からコマンド文字列を送信して NUCLEO-F401RE に入力される ADC データを取得します。

4. 1.　アプリケーション Access_SerialPort の起動画面

まず、PC と NUCLEO-F401RE とを USB ケーブルまたは RS232C インターフェースで接続してください。
アプリケーション Access_SerialPort を起動すると以下のダイアログが表示されます。

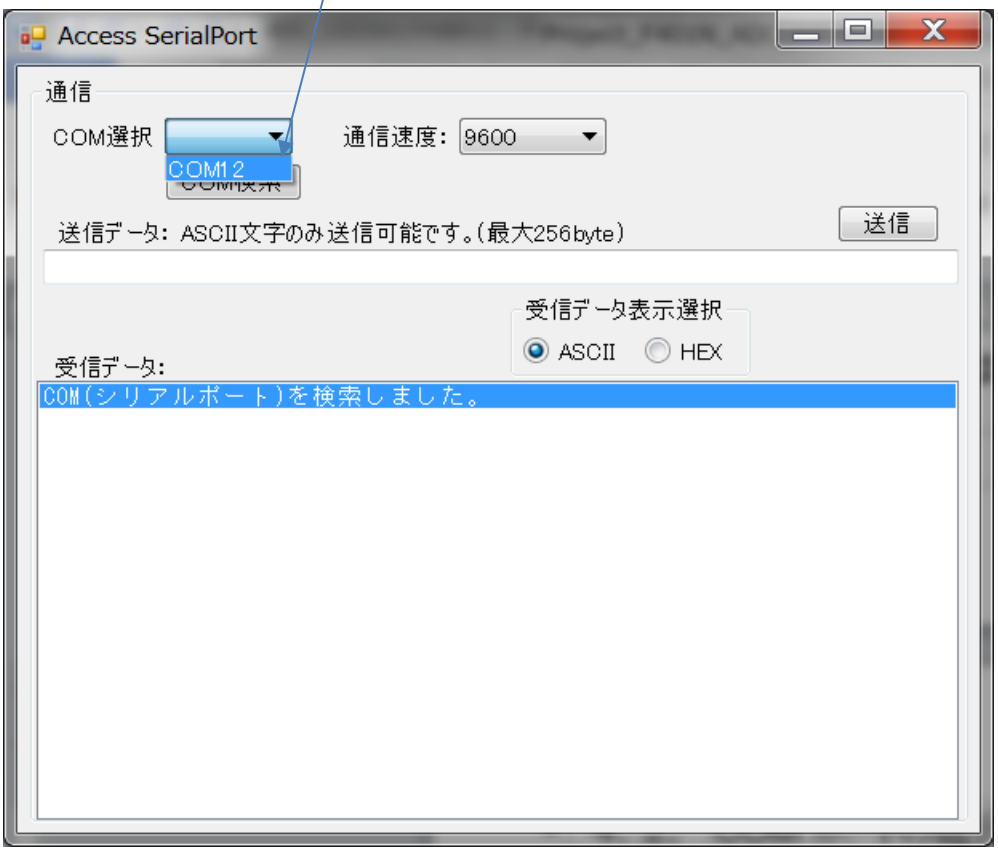


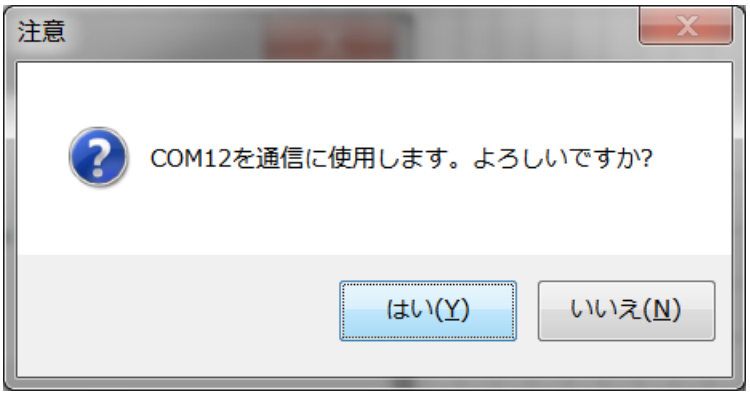
USB ケーブルまたは RS232C ケーブルを接続している場合、”COM(シリアルポート)を検索しました。” と表示されます。

4. 2.　COM ポートの選択と通信速度の設定

1）COM ポートの選択

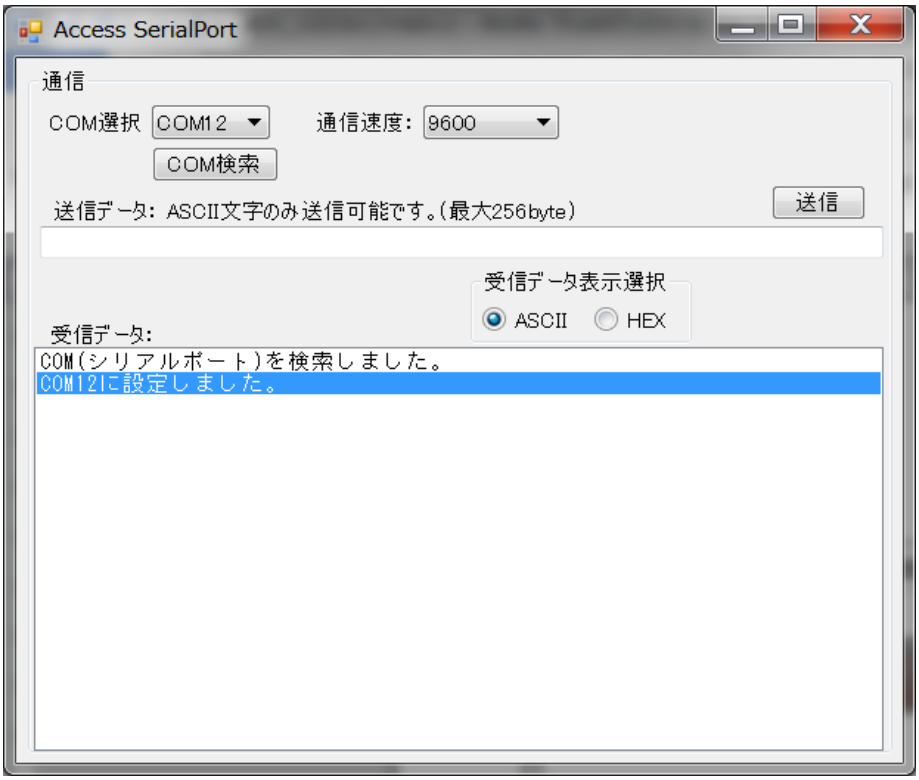
COM 選択の ComboBox で COM を選択します。





確認のメッセージが表示されるので、よい場合は［はい(Y)］ボタンをクリックします。

次ページに続く

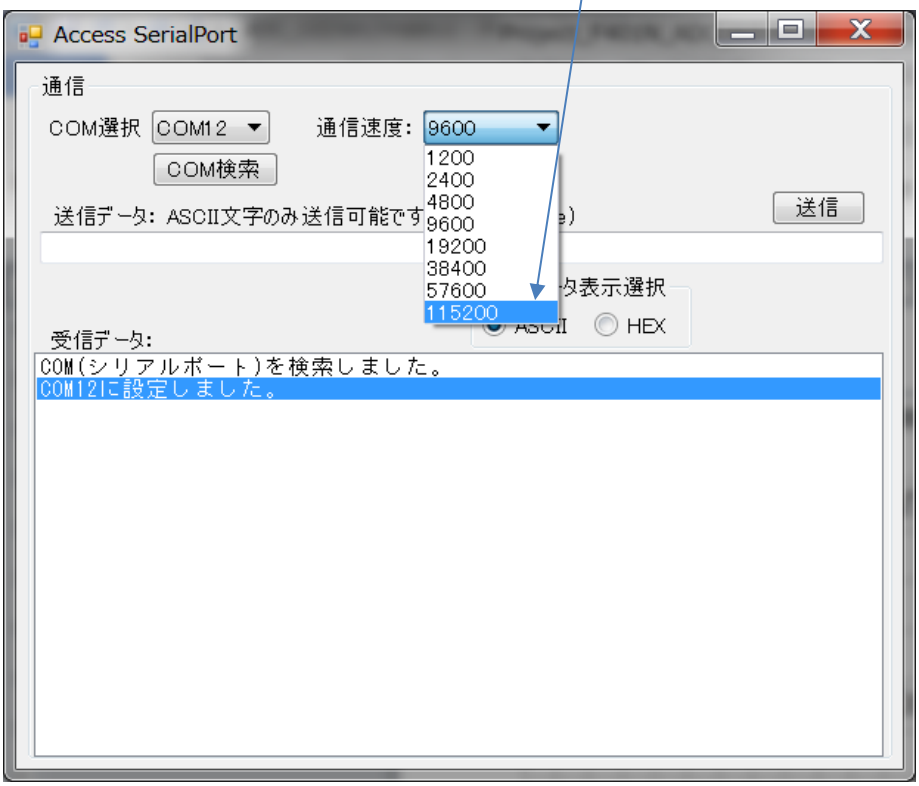


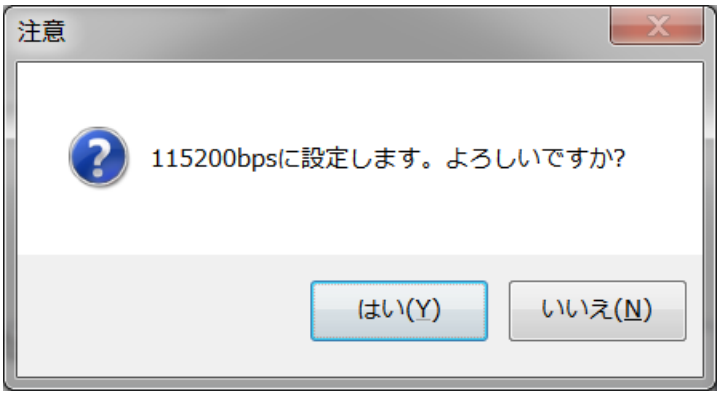
この例では、”COM12 に設定しました。” とメッセージが表示されています。

2） 通信速度の設定

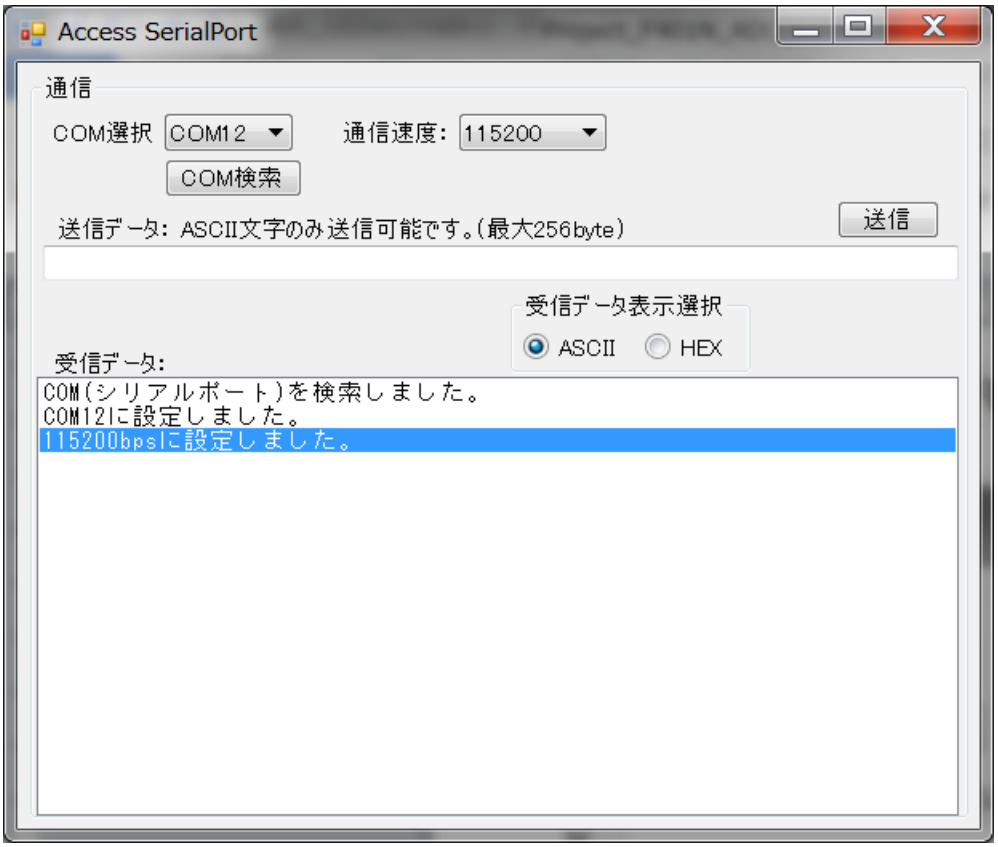
プロジェクトサンプル TrST_F401N_ADC_U2 の通信速度が 115200bps なので通信速度を 115200bps に設定します。

通信速度の ComboBox で 115200 を選択します。





確認のメッセージが表示されますので［ はい(Y) ］ボタンをクリックしてください。



受信データの欄に“115200bps に設定しました。”とメッセージが表示されます。

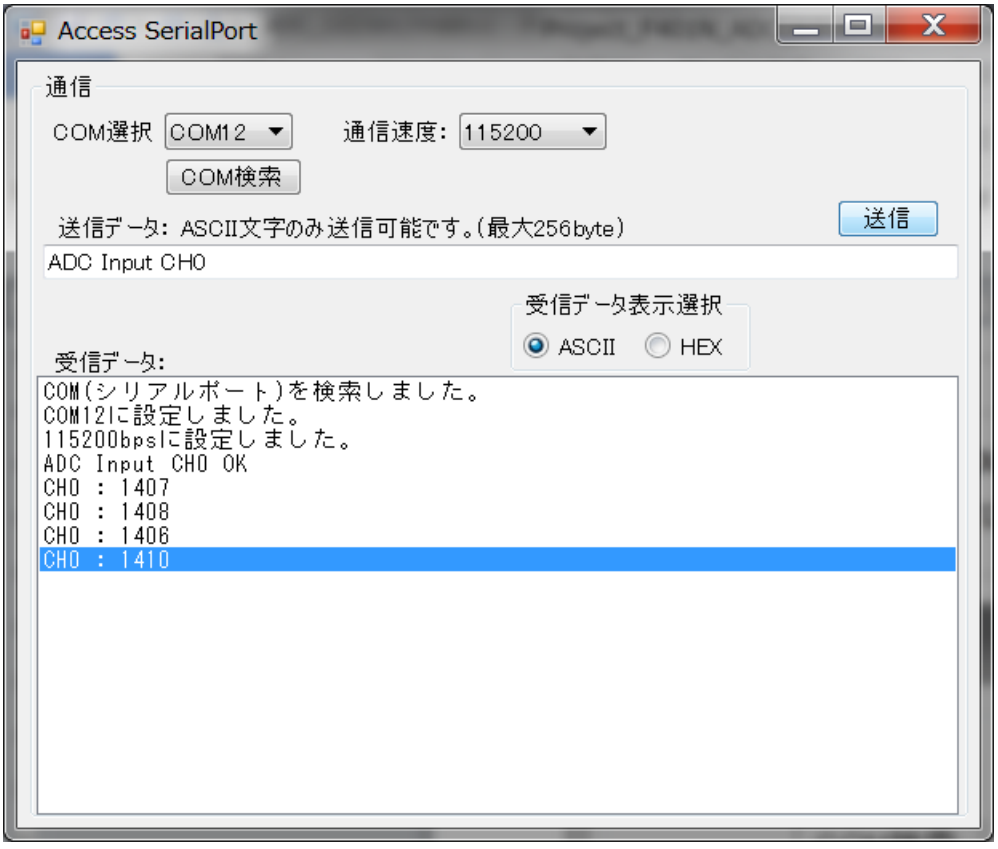
4. 3. ADC 入力コマンドの送信

ADC 入力コマンドは以下の通りです。

番号	受信コマンド	動作	対応する ADC 入力
1	ADC Input STOP	ADC データ送信停止	
2	ADC Input CH0	ADC CH0 データ送信	PC0 : ADC1_IN10
3	ADC Input CH1	ADC CH1 データ送信	PC1 : ADC1_IN11
4	ADC Input CH2	ADC CH2 データ送信	PC2 : ADC1_IN12
5	ADC Input CH3	ADC CH3 データ送信	PC3 : ADC1_IN13
6	ADC Input CH4	ADC CH4 データ送信	PC4 : ADC1_IN14
7	ADC Input CH5	ADC CH5 データ送信	PC5 : ADC1_IN15
8	ADC Input ALL	ADC 全 CH データ送信	PC0 – PC5 : ADC1_IN10 – ADC1_IN15

1） コマンド “ADC Input CH0” 送信時の動作

コマンド “ADC Input CH0” を送信した場合の動作を以下に示します。

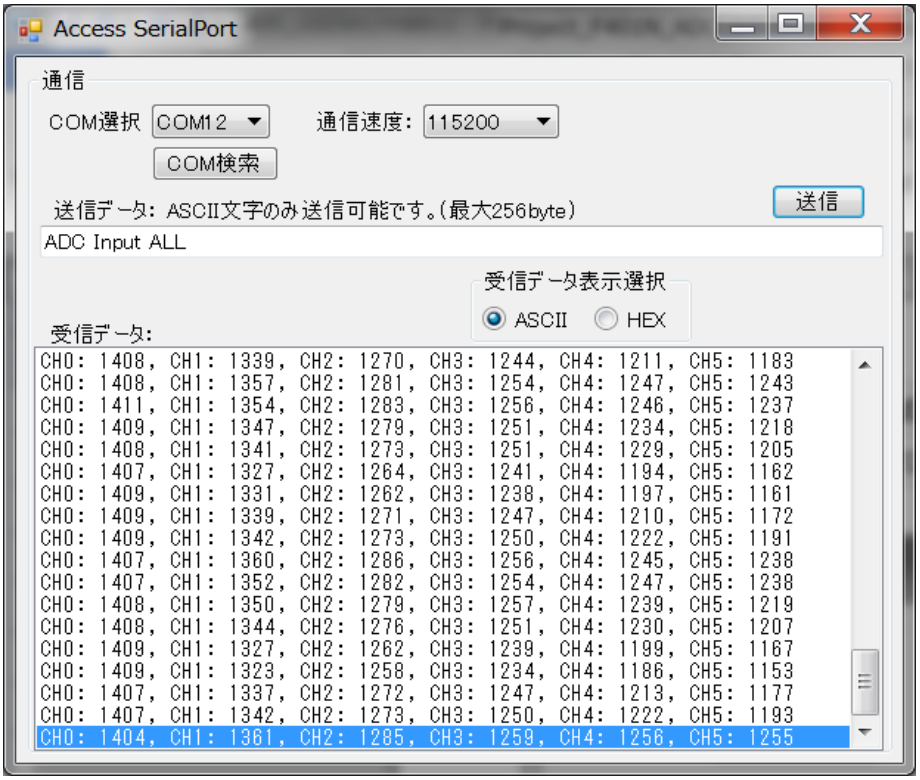


CPU 基板は 0.5 秒に一回 ADC CH0 の入力値を送信します。

次ページに続く

2）コマンド“ADC Input ALL”送信時の動作

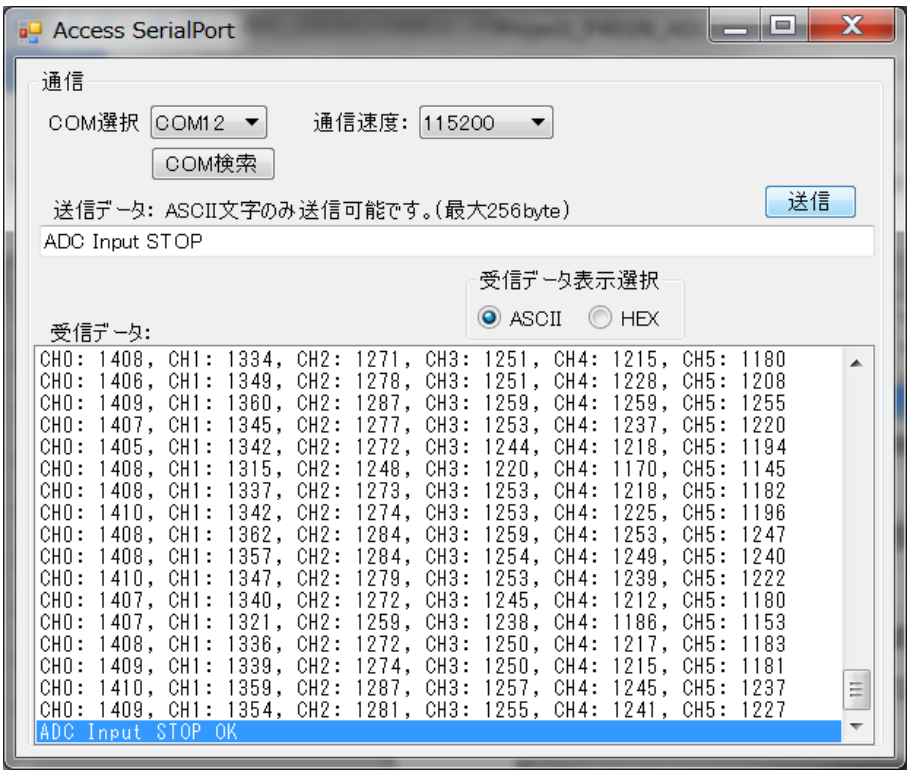
コマンド“ADC Input ALL”を送信した場合の動作を以下に示します。



CPU 基板は 0.5 秒に一回 ADC CH0 ～ ADC CH5 の 6CH の入力データを送信します。

3）コマンド“ADC Input STOP”送信時の動作

コマンド“ADC Input STOP”を送信した場合の動作を以下に示します。

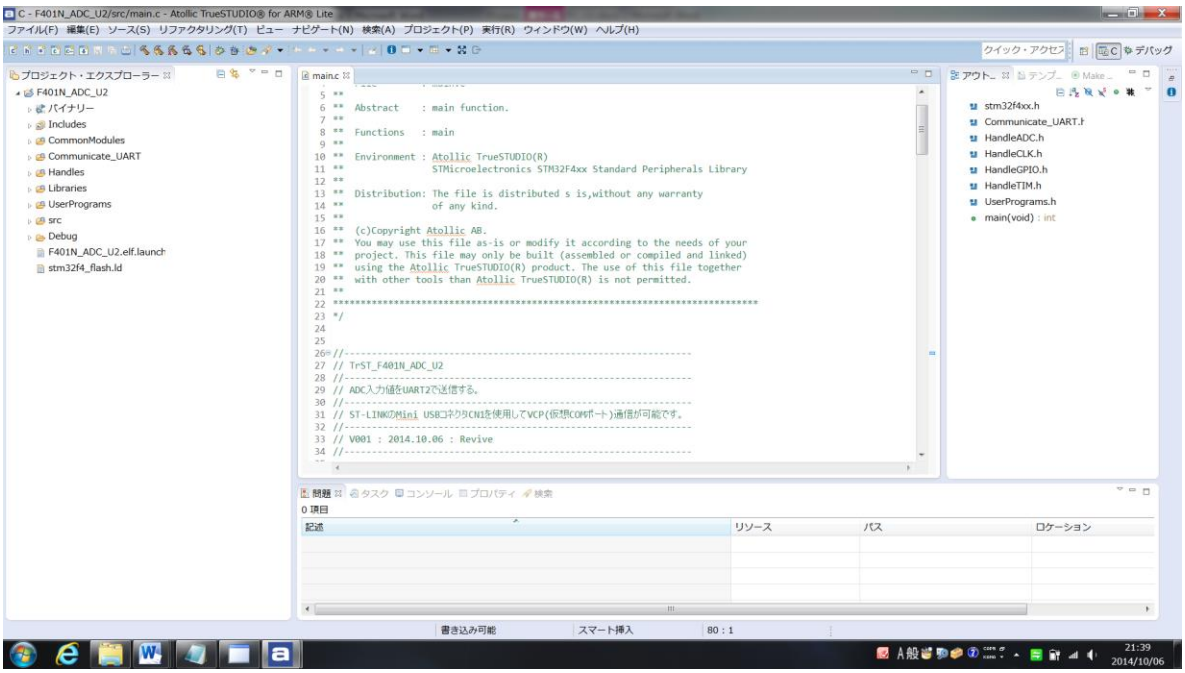


CPU 基板は ADCデータの送信を停止します。

5. プロジェクトの構成

5. 1. プロジェクト F401N_ADC_U2 の起動画面

TrueSTUDIO で作成したプロジェクト F401N_ADC_U2 を開いた状態を以下に示します。
左側のプロジェクト・エクスプローラーの F401N_ADC_U2 を展開した状態です。



5. 2. 追加したソース・フォルダとファイル

追加したソース・フォルダとファイルについて簡単に説明します。

1） CommonModules (ソース・フォルダ)

共通に使用するモジュールを記述してあります。
時間待ち、文字列操作　などの処理を記述しています。

2） Communicate_UART (ソース・フォルダ)

a） Communicate_UART .h　Communicate_UART .c (ファイル)

接続相手との通信処理を記述しています。
データを受信して、ADC コマンドならコマンドに従って ADC 入力データを送信します。

3） Handles (ソース・フォルダ)

Peripheral の設定などを行っています。

a） HandleCLK.h　HandleCLK.c (ファイル)

内部クロック HSI を使用するための設定を記述しています。
HSI(周波数 16MHz)を入力して PLL により 84MHz にしてシステムクロック SYSCLK として使用します。

b） HandleGPIO.h　HandleGPIO.c (ファイル)

GPIO 入出力の初期設定を記述しています。

c） HandleTIM.h　HandleTIM.c（ファイル）
タイマ割り込みを使用するために、タイマの初期設定を記述しています。
1mSec ごとにタイマ割り込みが発生するように設定しています。

d） HandleUART.h　HandleUART.c（ファイル）
UART の初期化と UART 送受信の処理を記述しています。

4） UserPrograms（ソース・フォルダ）
LED のための処理を記述しています。

a） UserPrograms.h　UserPrograms.c（ファイル）
Status LED　：　LD2(緑) に使用している　GPIO の初期設定と点滅処理を記述しています。

6.　主なモジュールの説明

6. 1.　ソース・フォルダ src 内のファイル

ソース・フォルダ src 内のファイルでプログラムを追加した主なファイルについて簡単に説明します。

1） main.c

a） main 関数
プログラムはここから開始します。主に初期化処理関数を呼び出しています。

```
int main(void)
```

b） システムクロックの設定
システムクロック　SYSCLK の設定を行います。
内部クロック HSI を入力に選択し、PLL を使用して 84MHz に設定します。

```
//-----------------------------------------------------------------------------
// HSIを選択して、PLL ClockをSystem Clockとして使用する。 : SYSCLK = 84MHz
//-----------------------------------------------------------------------------
SetHSICLK84MHz();
```

c） 周辺クロックの初期化

```
//----------------------------
//  周辺クロックの初期化
//----------------------------
void RCC_Configuration(void);
```

d） GPIO の初期化

```
//----------------------------
// GPIO初期化
//----------------------------
void Init_GPIOs(void);
```

e） Status LED ポートの初期化

```
//-------------------------------------
// Status LEDポート初期化 : LD2(緑)
//-------------------------------------
InitializePortStatusLED();
```

f） UART の初期化

```
//-----------------------------------------------------------
// UART2通信パラメータ初期化 : 通信速度  9600bps
//-----------------------------------------------------------
InitializeCommunicate_UART2((uint32_t)9600);
```

g） ADC の初期化

```
//------------------------------
// ADC1のWork初期化
//------------------------------
InitializeWork_ADC1();
```

h） TIM11 の初期化

```
//-------------------------------------------------------------------------
// TIM11初期化
//-------------------------------------------------------------------------
InitializeTIMxx(TIM11, RCC_APB2Periph_TIM11, GLB_uint16_vTIM11_CCR1);

EnableIrqTIMxx(TIM11, RCC_APB2Periph_TIM11, TIM1_TRG_COM_TIM11_IRQn);
                                                        // TIM11 Interrupt ON
//-------------------------------------------------------------------------
```

2） stm32f4xx_it.h　stm32f4xx_it.c

このファイルに割り込み処理を記述します。

本プロジェクトサンプルでは ADC 割り込み と TIM11 のタイマ割り込み処理 および UART2 の割り込み処理を記述しています。

6. 2.　HandleADC

1） ADC の初期化

ADC の初期化を行います。

```
//-----------------------------------------------------------------------------
// ADC1の初期化
//-----------------------------------------------------------------------------
void InitializeADC1(void);
```

2）ADC 入力開始

指定されたA/D CHの入力を開始します。

```
//-----------------------------------------------------------------------------------------
// ADCx_INx入力開始
//-----------------------------------------------------------------------------------------
/* ADC1を選択可能 */
//-----------------------------------------------------------------------------------------
//引数　:
// ADC_TypeDef *ADCx : where x can be 1, 2 or 3 to select the ADC peripheral.


// uint16_t uint16_INx : ADCx_INx選択

//      0 : PA0 : ADC1_IN0
//      1 : PA1 : ADC1_IN1
//      2 : PA2 : ADC1_IN2
//      3 : PA3 : ADC1_IN3

//      4 : PA4 : ADC1_IN4
//      5 : PA5 : ADC1_IN5
//      6 : PA6 : ADC1_IN6
//      7 : PA7 : ADC1_IN7

//      8 : PB0 : ADC1_IN8
//      9 : PB1 : ADC1_IN9

//      10 : PC0 : ADC1_IN10
//      11 : PC1 : ADC1_IN11
//      12 : PC2 : ADC1_IN12
//      13 : PC3 : ADC1_IN13

//      14 : PC4 : ADC1_IN14
//      15 : PC5 : ADC1_IN15


// uint8_t uint8_ADC_SampleTime : The sample time value to be set for the selected channel.
//      ADC_SampleTime_3Cycles: Sample time equal to 3 cycles
//      ADC_SampleTime_15Cycles: Sample time equal to 15 cycles
//      ADC_SampleTime_28Cycles: Sample time equal to 28 cycles
//      ADC_SampleTime_56Cycles: Sample time equal to 56 cycles
//      ADC_SampleTime_84Cycles: Sample time equal to 84 cycles
//      ADC_SampleTime_112Cycles: Sample time equal to 112 cycles
//      ADC_SampleTime_144Cycles: Sample time equal to 144 cycles
//      ADC_SampleTime_480Cycles: Sample time equal to 480 cycles
//-----------------------------------------------------------------------------------------
void Start_ADCx_INx(ADC_TypeDef *ADCx, uint16_t uint16_INx, uint8_t uint8_ADC_SampleTime);
```

3） ADC データの移動平均

ADC 入力データの各 CH の移動平均を行います。

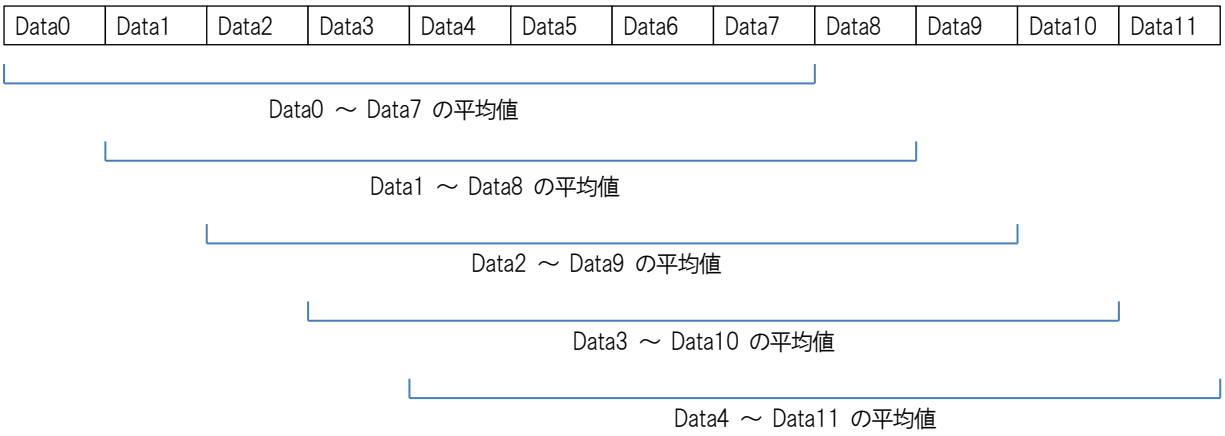
```
//-----------------------------------------------------------
// ADC入力データの移動平均処理
//-----------------------------------------------------------
void MovingAverageADC(void);
```

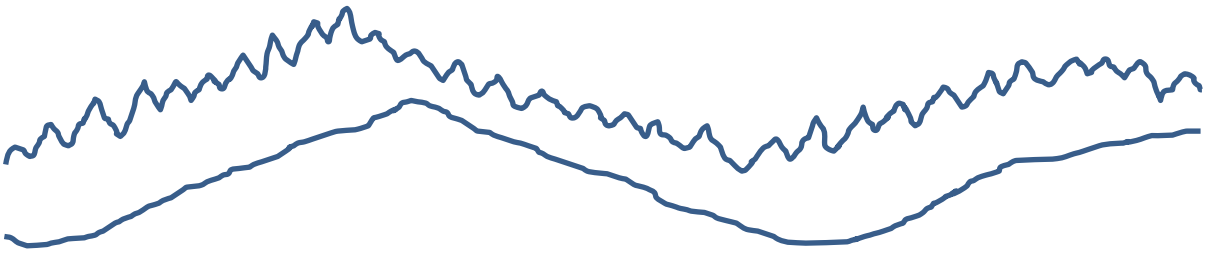
4） 移動平均について

データ 8 個の移動平均について説明します。

データ 8 個の平均を新しいデータが入力される毎に 1 データずつ、ずらして平均していきます。

平均するデータが移動していくので移動平均と呼びます。以下にその様子を示します。





移動平均は上の図のようにでこぼこしたデータを滑らかにする効果があります。平均する個数を多くするほど滑らかになりますが、応答が遅くなります。短い時間での変化が見えなくなってしまいます。

また、平均する個数を多くするほどグラフの形が後ろにずれていきます。

```
//-----------------------------------------------------------
// A/D入力データの移動平均処理
//-----------------------------------------------------------
void MovingAverageADC(void);
```

の処理では、移動平均の個数を 32 個に設定しています。

ヘッダファイル HandleADC.h のなかで **#define** defNumbersAverage 32 を定義しています。

この定義を変更すると、移動平均の個数を変更できます。

6. 3. HandleCLK

動作クロックに内部クロック HSI : 16MHz を選択し、PLL により 84MHz にして使用します。

```
//-------------------------------------------------------------------------------
// HSIを選択して、PLL ClockをSystem Clockとして使用する。 : SYSCLK = 84MHz
//-------------------------------------------------------------------------------
void SetHSICLK84MHz(void);
```

6. 4. HandleGPIO

GPIO を初期化します。

1） 低消費電力モード時の GPIO 初期化

最初は、使用しない GPIO ピンをアナログ入力モードに初期化します。

```
//-------------------------------------------------------------------------------
// 低消費電力モード時のGPIO初期化
//-------------------------------------------------------------------------------
void InitializeGPIOs_LowPower(void);
```

2） GPIO の初期化

必要な GPIO の初期化を行います。

```
//-----------------------------------------
// GPIO初期化
//-----------------------------------------
void  Init_GPIOs(void);
```

6. 5. HandleTIM

1） TIM11 の初期化

タイマ割り込みのために TIM11 を初期化してインターバルをセットします。
1mSec ごとに割り込みがかかるように設定しています。

以下の関数の引数に TIM11 用のパラメータを指定して TIM11 を初期化します。

```
//-------------------------------------------------------------------------------
// TIMxx初期化
//-------------------------------------------------------------------------------
//引数　:
// TIM_TypeDef *TIMxx : TIM選択
// uint32_t RCC_APB1Periph_TIMxx : specifies the APB1 peripheral to gates its clock.
// uint16_t uint16_TIMxx_CCR1 : TiMxx CH1のインターバル
//-------------------------------------------------------------------------------
void InitializeTIMxx(TIM_TypeDef *TIMxx, uint32_t RCC_APB1Periph_TIMxx, uint16_t uint16_TIMxx_CCR1);
```

2） タイマ割り込み許可

以下の関数の引数に希望する TIMxx 用のパラメータを指定して割り込みを許可します。

```
//-----------------------------------------------------------------------------------
// TIMxx割り込み許可
//-----------------------------------------------------------------------------------
//引数　:
// TIM_TypeDef *TIMxx : TIM選択
// uint32_t RCC_APB1Periph_TIMxx : specifies the APB1 peripheral to gates its clock.
// uint8_t TIMxx_IRQn : STM32 specific Interrupt Numbers
//-----------------------------------------------------------------------------------
void EnableIrqTIMxx(TIM_TypeDef *TIMxx, uint32_t RCC_APB1Periph_TIMxx, uint8_t TIMxx_IRQn);
```

6. 6. HandleUART

1）UART の初期化

```
//-----------------------------------------------
// UART2初期化
//-----------------------------------------------
//引数　:
// uint32_t uint32_BaudRate : 通信速度 bps
//-----------------------------------------------
void InitializeUART2(uint32_t uint32_BaudRate);
```

2）UART 送信

送信 Buffer に格納されたデータを指定データ数送信します。

```
//-------------------------------------------------------------------
// UART2 送信処理 : 送信Bufferに送信データがセットされた状態でCallされる。
//-------------------------------------------------------------------
//引数　:
// uint16_t uint16_SendLength : 送信データ数

//戻り値　:
//                0 : 送信終了
//                1 : エラー
//-------------------------------------------------------------------
void SendUART2(uint16_t uint16_SendLength);
```

3）UART 受信

受信待ちを行いデータを受信したら、受信 Buffer に格納します。

```
//-------------------------------------------------------------------
// UART2 受信処理
//-------------------------------------------------------------------
//引数　:
// uint8_t *puint8_ReceiveBuffer : 受信データを格納するBufferのポインタ

//戻り値　:
//               -1 : 受信なし
//                0 : 受信なし
//               1以上 :  受信byte数
//-------------------------------------------------------------------
int16_t ReceiveUART2(uint8_t *puint8_ReceiveBuffer);
```

6. 7. UserPrograms

UserPrograms.h には LED に使用する GPIO に対する定義を記述してあります。

以下に、UaerPrograms.c に記述している関数の説明を記します。

1）LED に使用する GPIO の初期化(共通処理)

GPIO番号とピン番号を指定して I/Oを初期化します。

```
//-----------------------------------------------------------------
// LEDポート初期化
//-----------------------------------------------------------------
//引数　:
// GPIO_TypeDef *GPIOx : GPIOポート指定
// uint16_t GPIO_Pin_x : GPIOピン指定
//-----------------------------------------------------------------
void InitializePortLED(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin_x);
```

2）LED の点滅処理(共通処理)

GPIO番号とピン番号などを指定して 希望のLEDの点滅処理を行います。

```
//-----------------------------------------------------------------
// LED点滅 : 点灯/消灯　切り替え
//-----------------------------------------------------------------
// 点灯/消灯　を切り替えると同時に　点灯時間/消灯時間　をセットする。
//-----------------------------------------------------------------
//引数　:
// GPIO_TypeDef *GPIOx : GPIOポート指定
// uint16_t GPIO_Pin_x : GPIOピン指定

// int16_t *pint16_OnOff : ON/OFF状態
//                         0 : OFF
//                         1 : ON

// uint16_t *puint16_Timer : 点灯時間/消灯時間をセットする変数のポインタ

// uint16_t uint16_TimeON : 点灯時間
// uint16_t uint16_TimeOFF : 消灯時間
//-----------------------------------------------------------------
void BlinkLED(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin_x,
                         int16_t *pint16_OnOff, uint16_t *puint16_Timer,
                         uint16_t uint16_TimeON, uint16_t uint16_TimeOFF);
```

3）LED に使用する I/O の初期化

a） StatusLED : LD2(緑)

```
//-----------------------------------------------------------------
// Status LEDポート初期化 : LD2(緑)
//-----------------------------------------------------------------
void InitializePortStatusLED(void);
```

4） StatusLED : LD2(緑)の点滅

```
//-----------------------------------------------------------------
// Status LED点滅 : LD2(緑) : 点灯/消灯　切り替え
//-----------------------------------------------------------------
//  TIMx割り込み内でGLB_uint16_BlinkTimerStatusLEDをデクリメントする。
// GLB_uint16_BlinkTimerStatusLEDが0になった時、呼び出される。
//-----------------------------------------------------------------
// 点灯/消灯　を切り替えると同時に　点灯時間/消灯時間　をセットする。
//-----------------------------------------------------------------
//引数　:
// uint16_t uint16_TimeON : 点灯時間
// uint16_t uint16_TimeOFF : 消灯時間
//-----------------------------------------------------------------
void BlinkStatusLED(uint16_t uint16_TimeON, uint16_t uint16_TimeOFF);
```

6. 8. Communicate_UART

1）UART の初期化

```
//-----------------------------------------------------------
// UART2通信パラメータ初期化
//-----------------------------------------------------------
//引数　:
// uint32_t uint32_BaudRate : 通信速度
//-----------------------------------------------------------
void InitializeCommunicate_UART2(uint32_t uint32_BaudRate);
```

2）UART 通信処理

UART の受信待ちを行い、受信データを判定して ADC データ送信状態フラグをセットします。

```
//-----------------------------------------------------------
// UART2通信処理
//-----------------------------------------------------------
//戻り値　:
//               -1 : 処理中
//               0 : 終了
//-----------------------------------------------------------
int16_t Communicate_UART2(void);
```

3）受信コマンドの実行

```
//-------------------------------------------------------------------
//受信コマンド判定 および 実行
//-------------------------------------------------------------------
//引数
// uint16_t uint16_ReceiveLength : 受信データ数
// uint8_t *puint8_ReceiveData : 受信データが格納されたBufferのポインタ
// uint8_t *puint8_SendData : 応答送信データを格納するBufferのポインタ

//戻り値：　応答送信データ数
//-------------------------------------------------------------------
uint16_t ExecuteCommandUART(uint16_t uint16_RecieveLength, uint8_t *puint8_ReceiveData,
                                        uint8_t *puint8_SendData);
```

4） ADC入力コマンド処理

```
//---------------------------------------------------------------------
// ADC入力コマンド処理
//---------------------------------------------------------------------
// Command :
//               ADC Input [Parameter]

// [Parameter] :
//               STOP : ADC入力停止
//               CH0 : ADC1_IN10
//               CH1 : ADC1_IN11
//               CH2 : ADC1_IN12
//               CH3 : ADC1_IN13
//               CH4 : ADC1_IN14
//               CH5 : ADC1_IN15
//               ALL : ADC1_IN10 - ADC1_IN15
//---------------------------------------------------------------------
//引数　:
// uint16_t uint16_DataLength : パラメータのデータ数
// uint8_t *puint8_Parameter : ADC入力パラメータのポインタ
//               "CH0" ～ "CH5" : 単独CHのADC入力開始
//               "ALL" : 6CH全てのADC入力開始
//               "STOP" : ADC入力の停止

// uint8_t *puint8_SendData : 応答データを格納するBufferのポインタ

//戻り値　:　応答送信データ数
//---------------------------------------------------------------------
uint16_t ComActInputADC(uint16_t uint16_DataLength, uint8_t *puint8_Parameter,
                                                    uint8_t *puint8_SendData);
```

```
// Command : "ADC Input [Parameter]"
```

の［Parameter］の部分を判定して、ADC データの送信を決定する。

```
//---------------------------------------------------------------------
// [Parameter] を判定してADCデータ送信状態フラグをセットする。
//---------------------------------------------------------------------
//volatile int16_t GLB_int16_vCommunicateADC = -1; // A/Dデータ送信状態フラグ
//---------------------------------------------------------------------
                                          [Parameter]
//      -1 : ADC送信停止                      "STOP"
//      0 : ADC CH0データ送信                 "CH0"
//      1 : ADC CH1データ送信                 "CH1"
//      2 : ADC CH2データ送信                 "CH2"
//      3 : ADC CH3データ送信                 "CH3"
//      4 : ADC CH4データ送信                 "CH4"
//      5 : ADC CH5データ送信                 "CH5"
//      6 : ADC 全CHデータ送信                "ALL"
//---------------------------------------------------------------------
```

5） ADC データ送信

ADC データ送信状態フラグ GLB_int16_vCommunicateADC を参照して、内容に従って送信処理を行います。

```
//---------------------------------------------------------------------
// ADCデータ送信
//---------------------------------------------------------------------
void SendDataADC(void);
```

ADC データ送信状態フラグ GLB_int16_vCommunicateADC の値に対する動作は以下の通りです。

```
//-----------------------------------------------------------------
//       -1 : 送信停止
//       0 : ADC CH0データ送信 : ADC1_IN10
//       1 : ADC CH1データ送信 : ADC1_IN11
//       2 : ADC CH2データ送信 : ADC1_IN12
//       3 : ADC CH3データ送信 : ADC1_IN13
//       4 : ADC CH4データ送信 : ADC1_IN14
//       5 : ADC CH5データ送信 : ADC1_IN15
//       6 : ADC 全CHデータ送信 : ADC1_IN10 - ADC1_IN15
//-----------------------------------------------------------------
```

有限会社りばいぶ

電子工作のための「飛石伝ひ」

改訂履歴

V001　2014／10／06　初版

V002　2014／10／21　誤記訂正

V003　2014／11／03　誤記訂正