ETCB Manual

HARDWARE & PROGRAMMING MANUAL

株式会社知能機械研究所

ETCB Manual Version 2.0

http://chinoken.jp support@chinoken.jp

目次

始めに	
注意事項	
免責事項	
同梱品	
別途必要なもの	
サポート	2
ETCB仕様	
107 - 10	
	
刘心RASPBERRY PI 注意事項	
外部接続詳細	6
外観・ポート	6
ETCB接続ポート	7
電源(POW)	7
COMポート (COM1)	
AD変換ポート (ADC)	
UART2 • UART3	
S1~S2	
EXTポート	
UART1 (UART_TX · UART_RX)	
I2C	
その他通信ポート	
SPI	
GPI01~4	
GPIO5 · GPIO6	
PWM(フルカラーLED)	9
開発環境の構築	10
ハードウェアの準備・確認	
ADポート入力電圧の切り替え	

ソフトウェアのインストール	11
Atollic TrueSTUDIOのインストール	12
DFuSe Demonstrationのインストール	12
5TM32CubeMXのインストール	13
ETCBのMCU用ライブラリのインストール	13
STM32CubeMXアップデートとライブラリのアップデート	13

プログラム作成と書き込み・実行......15

USBケーブルで書き込み	15
HEXファイルをDFU形式に変換する	15
DFUファイルの書き込み	15
ソースコード編集・新規プログラム作成	17
STM32CubeMXでソースコード生成	17
TRUESTUDIOを使ったソースコード編集・デバッグ	19
書き込みと実行	24
ST-LINKを使用しない場合(USBケーブル書き込み)	24
ST-LINKを使用する場合	24
デバッグ命令	25
デバッグ中にブレークポイントで止まらない場合	26

準備	
必要なソフトウェア・デバッガ	
RaspBerry Piとハードウェア接続(UART)	
STM32CubeMXでETCBプログラムひな形を作る	
プログラムひな形の内容	
STM32CuBEMXの起動とプロジェクトの新規作成	
PINOUT設定(端子設定)	
RCC設定	
AD変換設定	
GPIO設定	
UART1設定	
UART2、UART3設定	
PWM(フルカラーLED)設定	
タイマー(TIM2)設定	35
デバッガ設定	
サーボON/OFFスイッチGPIO設定	
クロック設定	
CONFIGURATION	

目次

IM1(フルカラーLED)周期設定	39
1M2	40
JART2とUART3の通信プロトコル設定	41
JART1(RX割り込み・TX DMA転送)	41
AD変換設定(変換後DMA転送+DMA転送完了割り込み)	43
割り込み優先順位設定	45
PRIORITY GROUP	46
プログラムの生成	47
プロジェクトの保存	47
コード(プログラムひな形)の生成	49

プロジェクト構成の確認とソースコード・ヘッダファイルの追加	51
例題プログラム概要(内容の確認)	51
USER.H	54
USER.C	
MAIN.C	62
コマンドを送信する	66

始めに

このたびは弊社ロボット制御プログラミングボードETCBをお選びいただきまして誠にありがとうございます。本マニュアルをよくお読みいただいた上でご利用なさいますようお願い申し上げます。

注意事項

- ETCBは電気・電子部品でできています。湿気の多いところやほこりっぽい場所などでのご使用 にならないでください。同様に濡れた手で扱ったり、電気・電子部品を直接触ったりしてはいけ ません。
- 仕様に合わない使い方はしないでください。
- 改造をしないでください。
- 動作がおかしいと感じた場合はすぐに使用を止め、すぐに弊社サポートまでご連絡ください。
- Raspberry Piは英国Raspberry Pi財団の登録商標です。

免責事項

ETCBを使用したときのいかなる結果においても弊社では責任を負いません。

同栶品

ETCBには次の部品が付属しています。

- ETCB 1個
- バインド小ねじ M2.6-12 4本
- バインド小ねじ M2.6-4 4本
- 樹脂スペーサΦ2.6L11 4本
- 樹脂スペーサΦ2.6L6 4本
- 2.54ピッチジャンパ 3個
- 2.54ピッチ、13x2ピンソケット 1個

もし足りない場合は下記サポートまでご連絡ください。

別途必要なもの

- 9~12Vの直流電源またはバッテリー
- シリアル通信用のUSBケーブル(USB-AオスーmicroBオス)
- Raspberry Piと接続する場合は半田と半田ごて

サポート

不具合、修理やご質問は下記メールアドレスへご連絡ください。不具合の詳細(不具合の内容、不具 合発生時の状況など)とご連絡先(氏名・メールアドレス)を忘れずお書き添えください。内容によ りましては弊社より折り返しご質問させていただくことがあります。あらかじめご了解ください。

support@chinoken.jp

ETCB仕様

概要

ETCBは浮動小数点演算回路をもつ最大72MHzで動作するSTM32F302 MCUを搭載したロボット用制御 ボードです。ドーターボードとしてRaspberry Piと接続すると、ETCBボードからRaspberry Pi駆動用の 電源を供給できます。またUART、I2C、SPIなどの通信ポートを共有してRaspberry Piと通信できま す。

外部ポートとしてADC入力ポート、UARTポート、USBポートなどが準備されておりセンサーを使った 自律動作するロボットや遠隔操縦ロボットのプログラミングがすぐに始められます。

市販のサーボモーターが使用できるサーボポートを2系統12ポート用意しています。最大6Aの大電流 を流せるので大型のロボットも作れます。さらにサーボポートへの電源供給ON/OFFを切り替えるこ とができますので、省電力モードも実現できます。

Atollic社のTrueSTUDIO とSTMicroelectornics社のSTM32CubeMX、およびST-LINK/v2デバッガに対応しているので、組み込み初心者でも比較的容易にプログラム・デバッグができます。プログラムファイルはUSB書き込みに対応していますので、ST-LINK/v2を持っていなくても書き込みができます。

項目	サブ項目	内容
電源	推奨入力電圧	6.6V~12V
	最大入力電圧	16V
	出力電圧	内部3.3V
		外部5.0V
	最大出力電流値	最大2.2A
	最大連続出力電流値	24以下
МСИ	型番	STMicroelectronics STM32F302C8
	メモリー	16kB SRAM
		64kB Flash
	CLOCK	72MHz(最大・標準)
IO	ADコンバータ	・12bit ADC入力ポートx4(カットオフ周波数1.6kHzのローパス
		フィルター付き)
		・ADCリファレンス電圧は3.3Vまたは5.0Vのいずれかを選択可
		能。ただしADC入力電圧は3.3Vを超えてはならない
	サーボ	・シリアルサーボ、コマンドサーボに対応
		・PWMサーボは使用できません
		・サーボポート2系統(UART2、UART3)各6ポート
		・信号ライン電圧5.0Vサーボに対応
		・UART2、UART3ポートと排他的利用可能
		・サーボ電圧は入力電源電圧と同じ

仕様

		・PA15ピンにてサーボへの電源供給ON/OFF切り替え可能
		・サーボポートは1系統あたり最大連続電流値3A以下を推奨
	USB	・PC接続用USBポートx1(FTDI FT230XS経由で仮想COMポートと
		して使用)
		・USBポートx1(MCUのUSB DP、DM端子と接続)
	UART1	・外部拡張ポート(EXT)、COM1ポートと共有可
		・Raspberry Piとポート共有可能
	UART2	・UART2ポートおよびサーボポートS7~S12に接続、UART2ポート
		とS1~S6ポートは排他的利用可能
	UART3	・UART3ポートおよびサーボポートS1~S6に接続、UART3ポート
		とS7~S12ポートは排他的利用可能
	12C	・I2C1として利用
		・Raspberry Piとジャンパピンにて共有可能
		・I2C_SDAおよびI2C_SCL端子は10kΩでプルアップ済み
	SPI	・SPI1として使用
		・Raspberry Piと接続
	GPI01~GPI04	・EXTポートに接続
		・MCU GPIOポートまたはTSC_G2_IO1~4ポートとして使用可能
	GPI05	・Raspberry Pi GPI022と接続
	GPI06	・Raspberry Pi GPI023と接続
LED	D1	・仮想COMポートを使って、MCU側より送信時に点灯(変更可
		能)
	D2	・EXTポートに接続
		・フルカラーLED
		・MCU端子PA8、PA9、PA10にLED RED、GREEN、BLUEがそれぞれ接
		続
		・PA8~10はAlternate FunctionとしてTIM_CH1~3に変更できる
		ので、PWM出力可能
書き込み	USB1	・DFUポートをショートしてETCBボードを起動すると、PCからDF
		U形式のプログラムが書き込み可能
	ST-LINK	・EXTポートの3.3V OUT、GND、SWDIO、SWCLK、NRSTポートを使
		用してSTMicroelectornics ST-LINK/v2でプログラム・デバッ
		ク可能

対応Raspberry Pi

ETCBが対応しているRaspberry Piは以下の通りです。ただしいずれのRaspberry PiもETCBが供給可能な 電流値を超えて使用はできません。

- Raspberry Pi Model B
- Raspberry Pi Model B+
- Raspberry Pi 2 Model B

Raspberry Pi 3 Model B

注意事項

- 6.6V以下で使用するとETCBの5V出力端子の電圧が低くなります。そのためアナログセンサに5V を供給する場合は注意してください。またRaspberry Piを取り付けていた場合は画面上に電圧低 下警告が表示されることがあります。
- サーボモーターを取り付けている場合は、サーボモーターが動作するのに十分なバッテリーまた は電源を接続してください。電源の性能により供給電力が不十分だったり急激な電圧低下が起こ ったりするとMCUがリセットされることがあります。
- 一般的にシリアルサーボやコマンドサーボといった、UART (Universal Asynchronous Receiver Tr ansmitter)通信で動作するサーボモーターのみ使用できます。PWMサーボは使用できません。
- STM32CubeMXやTrueSTUDIOでプログラムを作成する際には、サーボモーター電源をON/OFFで きる端子SVO_POW_SW(PA15、「図2 MCUピン配置」を参照)をGPIO Outputポートとして 定義してください。定義しないとサーボモーターに電源が入りません。
- Raspberry Piと接続する13x2ピンソケットはユーザー様が半田付けにて取り付ける必要があります。

外部接続詳細

外観・ポート

下図は基板寸法図とポート位置、MCUピン割り当て、およびEXTポートです。色はポートカラーと合わせています。薄い水色はRaspberry Piの拡張ポートと直接接続されます。







図2 MCUピン配置

		EXT		
*3.3V OUT	1	00	2	3.3V OUT
I2C_SDA	3	00	4	I2C_SDA(Raspi)
I2C_SCL	5	00	6	I2C_SCL(Raspi)
GND	7	00	8	GND
UART1_RX	9	00	10	UART_TX(Raspi)
UART1_TX	11	00	12	UART_RX(Raspi)
GND	13	00	14	GND
GPI04	15	00	16	NRST
GPI03	17	00	18	SWCLK
GPI02	19	00	20	SWDIO
GPI01	21	00	22	3.3V OUT
LED_BLUE(PA10)	23	00	24	LED_GREEN(PA9)
LED_RED(PA8)	25	00	26	GND

図3 EXTポート・端子名称

ETCB接続ポート

.*電源(POW)*

ETCBに電源を供給します。

電源コネクタは日本圧着端子製造社製のVHコネクタ(S2P-VH)を使っています。+とーを間違えない で、使用の範囲内(6.6~12V)の電圧でご利用ください。Li-Fe 2セル、3セルまたはLi-Po2セル、3セ ルバッテリーが使用できます。サーボ端子には電源電圧がかかりますので、入力電圧とサーボ電圧が 正しい組み合わせになっているか確認してください。

.COMポート(COM1)

COM1ポートでPCやその他デバイスとシリアル通信ができます。

COM1ポートはMCUのUART1ポートと接続されています。途中にFTDI社USB-シリアル変換ICが入って いますので、PCとUSBケーブルでつなぐとPC側では仮想COMポートとして認識します。FTDI社のデバ イスドライバ.¹が必要です。

_AD変換ポート(ADC)

ADCポートは12ビットアナログ入力ポートです。

¹ FTDI社ホームページ(<u>http://www.ftdichip.com/Drivers/VCP.htm</u>)よりVCP(Viratual COM Port)ド ライバをダウンロードしてインストールしてください。 切り替えジャンパで3.3Vまたは5.0V電源をセンサー側に供給します。MCUのリファレンス電圧は3.3V で入力最大電圧も3.3Vになっているので、5.0Vのセンサーをつないだ場合でもADC入力端子の電圧は3. 3Vを超えないように注意してください。

AD1~4ポート (ADC1_IN1~ADC1_IN4) ポートは1.6kHzのローパスフィルターが入っています。

ADC1_IN11ポート(PB0)は電源電圧監視専用ADCポートです。電源電圧を10/57に分圧しています。

.UART2·UART3

下記サーボポート(Sポート)と排他的共有ができるシリアルポートです。サーボポートは半二重通信 しかできませんが、全二重通信をする場合にUARTポートを使います。サーボポートと違って、信号の 電圧レベルは3.3Vですので注意してください。

UART2ポートのTX端子はサーボポートS7~S12に接続されています。UART3ポートのTX端子はS1~S6 に接続されています。

.*S1~S2*

市販のシリアルサーボモーターを使うためのポートです。各端子は外側から信号・電源・グランドとなります。

信号レベルは5.0Vです。通信プログラムを作成する場合はMCUポート設定でHalf Duplexに設定してお 使いください。UART2、UART3ポートとは排他的使用となります。

EXTポート

UART1 (UART_TX·UART_RX)

UARTポートはMCUのUART1ポートに接続されています。またCOMポートにも接続されています。EXT ポートのUARTポートとCOMポートは排他的使用となります。

EXTポートのUART1_TXとUART1_RX端子をそのまま使用すると、ETCBのMCUとシリアル通信ができます。Raspberry PiのUART端子もそのまま使えます。

UART1_TXとUART1_RX(Raspi)端子、UART1_RXとUART1_TX(Raspi)端子をジャンパピンでショートするとETCBのMCUとRaspberry Piでシリアル通信ができます。

.12C

MCUのI2C1ポートと接続されています。10kΩでプルアップ済みです。I2CポートとジャンパピンでショートするとRaspberry PiのI2Cポートと接続されます。ショートさせないで外部I2Cデバイスに接続することもできます。外部I2Cデバイスと接続するときはRaspberry Piの3.3V端子は使用せず、EXTポート3.3V端子を使用してください。

その他通信ポート

.SPI

Raspberry PiのSPIと接続済みです。Raspberry PiとMCU間でSPI通信するときに使用します。

.GPI01~4

GPIOポートです。

信号レベルは0~3.3Vになりますので、入力に使う場合は5V信号を入れないでください。Raspberry Pi とETCBの両方から出力しないようにしてください。

.GPI05 • GPI06

GPIO5とGPIO6ポートはRaspberry PiのGPIO2、GPIO3ポートとそれぞれつながっています。Raspberry PiとETCBの両方から出力しないようにしてください。

.PWM(フルカラーLED)

LED_RED、LED_GREEN、LED_BLUE はそれぞれMCUのPA8(TIM1_CH1)、PA9(TIM1_CH2)、PA10 (TIM1_CH3)とつながっています。フルカラーLEDをPWMで表示する場合は、表示間隔(PWM周 期)を10~20msにするときれいに表示されます。

開発環境の構築

本節ではETCBでプログラムを作成したり、デバッグを行ったりするための開発環境のインストールや 設定について説明します。

ハードウェアの準備・確認

ADポート入力電圧の切り替え

ETCBのADポート基準(リファレンス)電圧は3.3Vですが、アナログセンサに供給できる電圧は3.3Vま たは5.0Vのどちらか選択できます。お使いのアナログセンサが3.3V系の場合は「外観・ポート」節のV ref-IN端子3本の内、左と中央の端子を付属のジャンパでショートします。5.0Vの場合は右と中央の端 子をショートします。どちらもショートしていない場合、ADポートの中央の端子はオープンポートと なり、センサーに電源を供給しません。別電源を持つセンサーを使う場合はオープンで使用します。



※基準電圧(3.3Vまたは5.0V)にかかわらず、ADC入力ポートの最大入力電圧は3.3Vです

Raspberry Piとドッキング

ETCBをRaspberry Piとドッキングして使用する場合は付属の13x2ピンソケットをEXTポートの隣の未接 続端子に半田付けしてください。

Raspberry Piと通信する場合は、UART通信と12C通信が使えます。I2C通信をする場合は**エラー! ブック** マークが定義されていません。ページエラー! 参照元が見つかりません。を参考にして、I2C_SDA(3) とI2C_SDA(Raspi)(4)を付属のジャンパでショートします。また、I2C_SCL(5)とI2C_SCL(Raspi)(6)もショ ートします。UART通信をする場合は同様にUART1_RXとUART1_RX(Raspi)をショートし、UART1_TXと UART1_TX(Raspi)もショートします。

UART1端子はUSBのCOM1とも接続していますので、EXTポートのUART1端子をショートすると、ETCB のMCU、Raspberry Pi、USBのTX、RXが全てつながりますが、それらは同時に通信することはできま せん。MCU-Raspberry Pi間通信またはMCU-USB間通信のどちらかのみ使用してください。 I2C、UARTを単独(MCU単独またはRaspberry Pi単独)で使用したいときは、EXTポートから直接通信 ラインを引き出して使用してください。

SPIポート、GPIO5・6ポートはRaspberry Piとハードウェア接続していますので、ジャンパは不要です。

別途準備するもの

 6.6~12V電源。2アンペア以上出力できるACアダプタで動作します。電圧は使用するサーボモー ターによって異なります。サーボモーターを使わない場合は12VのACアダプタをおすすめしま す。コネクタは日本圧着端子製造株式会社のVHR-2Nコネクタを取り付けるか、市販のコード変 換アダプタをお使いください。



POW COM1 USB1

- USB1ポートからプログラムを書き込むためにはUSB(AオスーミニBオス)ケーブルが必要です。
- ST-LINKを使ったプログラム書き込みやデバッグを行う場合は、24ページの「図4 ST-LINKとの 接続」図を参考にしてケーブルで接続してください。一般的なジャンパケーブル(メスーメス) が使用できます。

ソフトウェアのインストール

ETCBではプログラム開発のために以下のソフトウェアを使用します。

- Atollic TrueSTUDIO Lite: Atollic社の提供する無料の統合開発環境です。
- DfuSe Demonstration: USBケーブルで(USB1端子)プログラムを書き込むためのツールで す。ST-LINKを使用する場合は不要です。
- DFU File Manager: TrueSTUDIOで作成したプログラム(HEX形式)をDFU形式に変換するツー ルです。DFU形式に変換するとDfuSe DemonstrationツールでUSB端子からプログラムを書き込 めるようになります。
- STM32CubeMX: STMicroelectronics社の提供する、C言語コードジェネレーター(プログラム 生成ソフトウェア)です。マウス操作で簡単にMCUの初期設定やUSB設定コードを生成できま す。コードは、Atollic TrueSTUDIOにインポートして使用できます。生成する内容は、ペリフェ ラル(周辺機器)のポート設定、クロック周波数、タイマー(周期、PWM設定など)、AD変換 (ポート登録、変換時間設定)、UART・I2C・SPIなどの通信設定、DMA転送設定、割り込み設

定などほとんどの初期設定項目をマウスで設定し、C言語コードを生成できます。 設定した項目はプロジェクトファイル(ioc形式)として保存できて、後から編集することも可 能です。ETCBではサンプルプログラムを可能な限りSTM32CubeMXのプロジェクトファイルで提 供する予定です。

Atollic TrueSTUDIOのインストール

開発環境にはAtollic社のTrueSTUDIOを推奨しています。始めにAtollic社サイトからTrueSTUDIO Lite版 をダウンロードしインストールしてください。できるだけ最新のものを使用するようにしてください。

ダウンロードサイト: http://timor.atollic.com/resources/downloads

インストールの途中でデバッグ用デバイスのドライバインストール画面が表示されますので、ST-LINK にチェックマークを入れてインストールしてください。

DfuSe Demonstrationのインストール

USBケーブルを使った書き込みをするには、TrueSTUDIOで作成したELF形式実行ファイルをDfuSe De monstrationソフトウェアを使ってDFU形式(Device Firmware Upgrade)に変更する必要があります。D fuSe Demonstrationソフトウェアは下記手順でダウンロードしてください。ST-LINKを持っている場合 は、本ソフトウェアは不要です。

- 1. STMicroelectronics社サイトヘアクセス http://www.st.com/content/st_com/ja.html
- 2. 右上の検索キーワード入力欄で「DFU」または「STSW-STM32080」と入力し、検索します。
- 3. 検索結果一覧の「製品型番」欄で「STSW-STM32080」を探してください
- 4. 「STSW-STM32080」がリンクになっていますので、クリックして先へ進みます。
- 5. 表示されたページの一番下にSTSW-STM32080の「ソフトウェア入手」ボタンがありますので、 クリックします。
- 6. ソフトウェアのライセンス契約内容が表示されますので、よく読んでライセンスに合意する場合 はACCEPTボタンをクリックします。
- 氏名およびメールアドレス入力欄が表示されますので、全て記入し「提出」ボタンをクリックします。
- ダウンロードリンクが指定したメールアドレスに届きますので、DfuSeソフトウェア(ファイル 名:en.stsw-stm32080.zip)をダウンロードしてインストールしてください。

.STM32CubeMXのインストール

STM32CubeMXはMCUのポート設定やタイマー、割り込み、DMAなどの諸設定をGUIで行うツールで す。ソフトウェア開発用のひな形となるソースコードを出力します。下記の手順でダウンロードして ください。

- STMicroelectronics社サイトへアクセスします。 http://www.st.com/content/st_com/ja.html
- 2. 右上の検索キーワード入力欄で「STM32CubeMX」と入力し、検索します。
- 3. 検索結果一覧の「製品型番」欄で「STM32CubeMX」を探してください。
- 4. 「STM32CubeMX」がリンクになっていますので、クリックして先へ進みます。
- 5. 表示されたページの一番下にSTM32CubeMXの「ソフトウェア入手」ボタンがありますので、ク リックします(画面はバージョン4.15.0のとき)。

/フトウェア入手					
製品型番	▲ s	Software Version	Marketing Status	Supplier 🔶	Order from ST
07142000 1 1414		1 15 0	Activo	ст	いつトウェブ コ チ

- 6. ソフトウェアのライセンス契約内容が表示されますので、よく読んでライセンスに合意する場合 はACCEPTボタンをクリックします。
- 7. 氏名およびメールアドレス入力欄が表示されますので、全て記入し「提出」ボタンをクリックし ます。
- 記入したメールアドレスにダウンロードリンクが届きますので、DfuSeソフトウェア(ファイル 名:en.st32cubemx.zip)をダウンロードしてインストールしてください。

ETCBのMCU用ライブラリのインストール

- 1. インターネットに接続している状態で、STM32CubeMXを起動してください。
- 2. メインメニューから、Help>Install New Libirariesを選択してください。
- 3. STM32CubeF3 Releasesの欄で、一番新しい「Firmware Package for Family STM32F3」を選択して、Install Nowボタンでインストールしてください。

.STM32CubeMXアップデートとライブラリのアップデート

STM32CubeMXのアップデート方法です。STM32CubeMXでは頻繁に本体やライブラリがアップデートされていますので、常に最新版となるようにしてください。

_STM32CubeMXのアップデート

- 1. STM32CubeMXを起動してください。
- 2. メインメニューの「Help」メニューから「Check for Updates」を選んでください。
- 3. 「Check Update Manager」ダイアログが表示されますので、下にある「Check」ボタンで最新版の確認をします。
- アップデートファイルがあった場合は、「Install Now」ボタンでアップデータのダウンロードと インストール準備を行います。
- 5. 「Install Now」ボタンでファイルのダウンロードが完了すると、再起動するように指示が出ま す。いったんSTM32CubeMXを終了してから、管理者権限で起動してください。STM32CubeMX を管理者権限で起動するには、Windowsショートカットアイコンまたはスタートメニューのアイ コンを右クリックして、「管理者として実行」を選択してください。
- 6. 起動時に自動的にアップデートされます。

*.ライブラリのアップデー*ト

- 1. STM32CubeMXを起動してください。
- 2. メインメニューの「Help」メニューから「Check for Updates」を選んでください。
- 3. メインメニューの「Help」メニューから「Install New Libraries」を選んでください。
- ダイアログが表示されますので、「STM32CubeF3 Releases」欄から「Firmware Package For ST M32 F3」の四角マークをクリックしてチェックマークをいれます。最新のバージョンを選択し てください。
- 5. ダイアログ一番下の「Install Now」ボタンでファームウェアパッケージをインストールします。

プログラム作成と書き込み・実行

USBケーブルで書き込み

USBケーブルでプログラムを書き込むには、HEX形式ファイルをDFU形式に変換します。DFU形式のフ ァイルをDfuSe Demoソフトウェアで書き込みます。

ST-LINKでプログラムを書き込む場合は24ページ「ST-LINKを使用する場合」を参照してください。

HEXファイルをDFU形式に変換する

1. Dfu file managerを起動します。I want to GENERATE a DFU file from S19, HEX or BIN filesにチェッ クを入れ、OKボタンを押します。



- 「S19 or Hex...」ボタンを押して、変換対象のファイルを選択します。今回はサンプルプログラム(LED_TEST.elf.hex)を使用します。
- 3. Generateボタンを押して、DFUファイルへの変換を実行します。変換後のファイル名の入力が求められますのでDFUファイルの名前(今回はLED_TEST.dfu)を入力して「保存」ボタンを押します。

B	DFU File Manager (v3.0.5) - Generation 🛛 🛛 🗖 🗙
Device Vendor ID 0x 0483 Product ID 0x 0000 Version 0x 0000	Images Image for Alternate Setting 00 (ST) Injection Target ID: 0 <u>S19 or Hex</u> Multi BIN Target Name: ST Deletion <u>Deletion</u> <u>Deletion</u> <u>Deletion</u>
	<u>G</u> enerate <u>C</u> ancel

4. ETCBボードのDFU端子をジャンパでショートさせた状態で電源を入れます。ジャンパを刺すためのピンヘッダは付属しておりませんので、ピンヘッダをお客様で半田付けしていただくか、ピンセットのようなものでDFU端子をショートさせてください。半田付けや端子のショートは間違えないように十分気をつけて行うようにしてください。

USB1のコネクタにマイクロUSBケーブルを取り付けPCと接続します。初回はデバイスドライバ (STM Device in DFU Mode)のインストールが実行されます。

DFUファイルの書き込み

 DfuSe Demoを起動します。「Available DFU Devices」のプルダウンメニューに「STM Device in DFU Mode」と表示されていることを確認してください。何も表示されていない場合は、4の手 順に戻ってDFUモードで起動しなおしてください。

また、DfuSe Demoウィンドウの真ん中にある、「Select Target(s):」の欄に、「Internal Flash 3 2sectors...」、「Option bytes 1sectors...」とあるか確認してください。表示が違う場合は、DF Uモードで起動していません(MCUを認識していない)。この場合も4の手順に戻ってDFUモー ドで起動しなおしてください。

 「Update or Verify Action」の「Choose…」ボタンを押し、転送対象のDFUファイルを選択しま す。今回は手順3で作成したLED_TEST.dfuを指定します。

	[סfuSe Demo (י	/3.0.5)	- 🗆 🗙
Available DFU Dev STM Device in DF Supports Unlow Can Detach Enter DFU mode/ Actions	ices U Mode ad Manifesta nload Accelera HID detach Leave	✓ ation tolerant ted Upload (ST) a DFU mode	Application Mode: Verdor ID: Procuct ID: Version:	DFU Mode: Vendor ID: 0483 Procuct ID: DF11 Version: 2200
Select <u>I</u> arget(s):	Target Id Name 00 Internal Fi 01 Option By	lash Ites	Available Sect 32 sectors 1 sectors	ors (Double Click for more)
Upload Action File: Choose Upload Transferred data size 0 KB(0 Bytes) of 0 KB(0 Bytes) Operation duration 00:00:00		Upgrade or Ve File: Vendor ID: Procuct ID: Version: Version: Chgose	r download	n file: nove some FFs) le ⊻erify
Abort				Quit

- Updateボタンを押し、ETCBボードへプログラムを転送します。書き込み中はプログレスバーに 進行状況が表示されます。プログレスバーに「Target 00: Upgrade successful!」と表示されたら 書き込み完了です。
- 4. 転送終了後にDFUピンのジャンパを取り外し、電源を入れなおします。

プログラム作成と書き込み・実行

ソースコード編集・新規プログラム作成

本節ではSTM32CubeMXサンプルプロジェクト(テンプレート)を使って、LEDが時間経過に沿って ゆっくりと色が変わっていくプログラムの作成・編集を行います。STM32CubeMXの使い方の詳細は2 9ページからの「STM32CubeMXでETCBプログラムひな形を作る」を参照してください。

_STM32CubeMXでソースコード生成



 ETCB_Turtle_temp.iocをダブルクリックして、「STM32CubeMX」を起動します。
 ETCB_Turtle_tempはETCBの基本的な機能が予め設定されているテンプレートファイルです。CO M1ポートやAD変換は表の構成となっています。

項目	内容	
MPU CLOCK	48MHz	
ADコンバータ	・AD1、AD2、AD3、AD4、バッテリー電圧、MPU内部温度の6チ ャンネルのデータを取得 ・変換後のデータはDMAを用いてメモリーへ転送	
UART1	 RT1 ・全二重通信 ・通信速度:115200[bps]、データ幅:8ビット、パリティ: 無 ・データの送信:DMA転送可、データの受信:割り込み 	
UART2_TX(PB3), UART3_TX(PB10)	 ・半二重通信 ・通信速度:115200[bps]、データ幅:8ビット、パリティ: 無 ストップビット:1ビット ・割り込み、DMA:未使用 	
TIM1	・LED_RED、LED_GREEN、LED_BLUE用のPWM信号を生成 ・PWM周期:15ms ・PWM幅の設定:0~99	
TIM2	・15ms周期のカウンタ	

・定期的な処理を組み込む際に使用

- Projectメニューから「Settings」を選びます。Project Settingsダイアログが表示されますので、P rojectタブを選択してProjectに関する情報を下記のように入力します。「Project Location」の欄 は任意です。通常はTrueSTUDIOを初回起動したときに作成されるWorkspaceフォルダを指定しま す。
 - また、「Toolchain / IDE」はTrueSTUDIOを選択してください。

	Project Settings	•
roject Code Generator	r Advanced Settings	
Project Settings		
Project Name		
LED_TEST		
Project Location		
Z:¥Workspace		Browse
		Lioneo
Toolchain Folder Loca	ition	
Z:¥Workspace¥LED_T	EST¥	
Toolchain / IDE		
TrueSTUDIO	V Generate Under Root	
Linker Settings		
Minimum Heap Size	0×200	
Minimum Stack Size	0×400	
Mcu and Firmware Pa	ckage	
Mcu Reference		
STM32F302C8Tx		
Firmware Package Na	me and Version	
STM32Cube FW_F3 V	/1.5.0	
	Ok	Cancel

項目	内容
Project Name	LED_TEST
Project Location	任意のフォルダ
Toolchain / IDE	TrueSTUDIO
Generate Under Root	チェックを入れます

3. Code Generatorタブに切り替え、「Copy only the necessary library files」を選択します。また「G enerated files」欄で「generate peripheral initialization as a pair of 'C/h' fules per Peripheral」にチ

ェック²を入れます。その他の欄はデフォルトで問題ありません(下記の図を参考にしてください)。

Project Settings	×
Project Code Generator Advanced Settings	
STM32Cube Firmware Library Package	
O Copy all used libraries into the project folder	
Copy only the necessary library files	
O Add necessary library files as reference in the toolchain project configuration file	
Conserved files	
Generated files	
Generate perpireral initialization as a pair of <i>DTI</i> mesperating Backup previously generated files when re-generating	
✓ Keen Liser Code when re-generating	
✓ Delete previously generated files when not re-generated	
HAL Settings	
Set all free pins as analog (to optimize the power consumption)	
Enable Full Assert	
Template Settings	
Select a template to generate customized code	Settings
Ok	Cancel

4. Projectメニューから「Generate Code」を選択しソフトウェア開発用のひな形となるソースコードを生成します。ソースコードは2番で指定した、「Toolchain Folder Location」欄に記載されているフォルダに保存されます。

TrueSTUDIOを使ったソースコード編集・デバッグ

 TrueSTUDIOを起動します。ワークスペース・ランチャー(ダイアログ)が表示されますので、ST M32CubeMXで作成したWorkspaceフォルダを指定してください。

² generate peripheral initialization as a pair of 'C/h' fules per Peripheralにチェックを入れると、AD変換 などの周辺回路のひな形ソースコードも生成されます。

2. 「ファイル」メニューから「インポート」を選択します。インポートダイアログ(選択)が表示 されますので、「一般」から「既存プロジェクトをワークスペースへ」を選択し、「次へ」ボタ ンを押します。

a	インポート	_ 🗆 🗙
選択 アーカイブ・ファイル	しまたはディレクトリーから新規プロジェクトを作成します。	Ľ
 1ンポート・ソース 2/1/2入力 ● 一般 ◎ アー ○ P <li< th=""><th>の選択(<u>5</u>): ・カイブ・ファイル イル・システム デブシェクトをワークスペースへ」 き e ール</th><th></th></li<>	の選択(<u>5</u>): ・カイブ・ファイル イル・システム デブシェクトをワークスペースへ」 き e ール	
?	< 戻る(<u>B</u>) 次へ(<u>N</u>) > 終了(E)	キャンセル

- インポートダイアログ(プロジェクトのインポート)で「ルート・ディレクトリの選択(T):」を選択し、参照ボタンをクリックしてSTM32CubeMXで作成したWorkspaceフォルダを指定してください。
- 4. 「プロジェクト(P):」欄にWorkspaceにあるLED_TESTプロジェクトが表示されていることを確認 して、右下の「終了」ボタンを押します。ボタンを押すとインポートが開始されます。画面上のL

a	インポート	- 🗆 🗙
プロジェクトのインボ 既存の Eclipse プロジ	とート ェクトを検索するディレクトリーを選択します。	
 ルート・ディレクトリーの アーカイブ・ファイルの 	D選択(工): Z:¥Workspace 選択(<u>A</u>):	 ✓ 参照(<u>R</u>) ✓ 参照(<u>R</u>)
プロシェクト(2): ✓ LED_TEST (175/32) 1 ネストしたプロシェク 1 プロシェクトをワーク、 ワーキング・セット(ワーキング・ ワーキング・セット(ワーキング・ ワーキング・セット(ワーキング・ ワーキング ワーキング ワーキング・ ワーキング ワーキング ワーキング ワーキング ワーキング ワーキング・ ワーキング フーキング フーキング ワーキング フーキング ワーキング フーキング フーキン フーキング フーキン フーキング フーキング フーキン フーキング フーキン フーキング フーキン	Z:¥Workspace¥LED_TEST) トを検索(∐) スペースにコピー(<u>C</u>) :プロジェクトを追加(<u>T</u>)	 すべて選択(<u>5</u>) 選択をすべて解除(<u>D</u>) 更新(<u>E</u>) > > > > <!--</td-->
? [< 戻る(<u>B)</u> 次へ(N) > 約	T(E) キャンセル

ED_TESTのチェック欄がGFェックマークでは無くて黒い■マークになっている場合は、オプション欄で「ネストしたプロジェクトを検索」を選択してください。

5. プロジェクト・エクスプローラでLED_TESTの左側の三角マークをクリックしプロジェクトのリストを展開します。Srcの左側の三角マークをクリックしソースファイルのリストを展開します。 「main.c」ファイルをダブルクリックしウィンドウに開きます。

a	C - LED_TEST/Src/main.c - Atollic TrueSTUDIO for ARM	- 🗆 🗙
ファイル(E) 編集(E) ソース(S) リファクタリング	(Ⅰ) ビュー ナビゲート(№) 検索(A) プロジェクト(P) 実行(R) ウィンドウ(W) ヘルプ(H)	
C N C C C R H R C 4	\$ 系 循 \$ 参 番目29 タ ▼1 1 - ◆ ▼ → ▼ ♂ ① ▼ 田 ▼ \$ D	アクセス 🗄 🖻 🗟 С
	_c mainc ☆	≝ / X ¹³ 3 — □
	28 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN A	🐉 🖻 🚏 🖉 🕺 🛛 🗰
LED_TEST	29 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY C 30 *	$\overline{\nabla}$
→ M Includes	31 ************************************	stm32f3xx_hal.h
Drivers	32 */	adc.h
A Src	33 /* Includes	u dma.h
adc.c	34 #include "stm32f3xx_hal.h"	tim.n
⊳ 🕞 dma.c	35 #include "dma.b"	usart.n
▶ i gpio.c	37 #include "tim.h"	SystemClock Confi
⊳ 🖻 main.c	38 #include "usart.h"	++ Error Handler(void
▶ ▲ stm32f3xx_hal_msp.c	39 #include "gpio.h"	 main(void) : int
b time a	40	 SystemClock_Confi
b in usart c	41 /* USER CODE BEGIN Includes */	 Error_Handler(void)
LED_TEST.elf.launch	43 /* USER CODE END Includes */	ø assert_failed(uint8_
LED TEST.ioc	44	
STM32F302C8_FLASH.Id	45 /* Private variables	
_	46	
	47⊖/* USER CODE BEGIN PV */	
	40 /^ Private variables	
	50 /* USER CODE END PV */	
	51	
	52 /* Private function prototypes	
	< >	< >
	の項目 の項目	rn more.
	記述 リソース	
	Memory Regions	
	< > Memory Details	
	書き込み可能 スマート挿入 11:5	

 main.cにLEDの輝度を調整するコードを追記します(赤字の部分)。なおユーザーが書き入れる コードは必ず「USER CODE BEGIN」から「USER CODE END」の間となります。それ以外の場 所にコードを書き入れると、STM32CubeMXでコード生成を行ったときにユーザーコードが消え る場合があります。

先に定義されている関数や変数は、記入途中でSHIFTキーとスペースキーを同時押しすると、名称の補完候補が表示されます。

下記コードをコピー&ペーストすると、全角スペースが挿入されてしまうことがあります。エラ ー表示で「stray '@' in program」や「stray '¥201 ' in program」などと表示された場合は、該 当のプログラムに全角スペースが混じっていますので、半角スペースなどに書き換えてくださ い。

/* Private function prototypes -----*/ void LED(uint8_t , uint8_t , uint8_t); // LEDのPWMを設定する関数のプロトタイプ

```
/* USER CODE END PFP */
  <中略>
 /* USER CODE BEGIN 2 */
  /*
  * LED用PWM出力開始
  */
   HAL_TIM_Base_Start(&htim1); // TIM1スタート
   HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1); // TIM1_CH1(LED赤) PWM出力開始
   HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2); // TIM1_CH2(LED緑) PWM出力開始
   HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3); // TIM1_CH3(LED青) PWM出力開始
   uint8_t count = 0 , red, green;
 /* USER CODE END 2 */
 /* Infinite loop */
 /* USER CODE BEGIN WHILE */
 while (1)
 {
 /* USER CODE END WHILE */
 /* USER CODE BEGIN 3 */
   if( count < 10 ) red = 10*count; // LED(赤)の設定値
   else red = 10*(20-count);
   green = 100 - red; // LED(緑)の設定値
   LED(red, green, 0); // 設定値の反映
   HAL_Delay(100); // 100ms休む
   count = (count+1) % 20; // 変数countの更新
 }
 /* USER CODE END 3 */
                              USER CODE BEGIN 41
}
                              main.cの下の方にあります
<中略>
/* USER CODE BEGIN 4 */
/*
* RGB明度指定でLEDを点灯する
* RGB明度は0~100
*/
void LED(uint8_t r, uint8_t g, uint8_t b)
{
 htim1.Instance->CCR1 = (uint32_t)r * 100;
```

```
htim1.Instance->CCR2 = (uint32_t)g * 100;
htim1.Instance->CCR3 = (uint32_t)b * 100;
}
/* USER CODE END 4 */
```

- 7. DFU形式ファイルをUSBポート経由で書き込みを行う場合は、プロジェクトの設定を変更する必要があります。ST-LINKを使用してプログラムの書き込み・デバッグを行う場合はこの手順を省略してかまいません。
- 8. DFU形式ファイルをUSBポート経由で書き込む場合は、標準の設定で生成されないHexファイルを 生成する必要があります。まずTrueSTUDIOのプロジェクトメニューからプロパティを選択し、 「C/C++ビルド」の左の三角マークをクリックしてビルド設定の項目を展開します。つぎに「設

定」を選び、「Other」の「Output format」を選択します。「Convert build output」にチェック



を入れ、Formatは「Intel Hex」を選びます。設定が終わったらOKボタンを押してプロジェクトの 変更を反映させます。

9. プロジェクトメニューのプロジェクトのビルドを選択し、プログラムのコンパイルを実行しま す。デバッグ用ELF形式ファイルと一緒にHEXファイルが生成されます。

ビルドに失敗した場合は、プログラムが間違っていないか確認・再編集をして再度ビルドしてく ださい。

書き込みと実行

ST-LINKを使用しない場合(USBケーブル書き込み)

「DFU形式ファイルをUSBポート経由で書き込み」で行った手順でビルド後のプログラムをETCBボードへ転送して実行します。

ST-LINKを使用する場合

ST-LINKを使用する場合はデバッグを行うことができます。先ほど作成したプログラムにブレイクポイント(デバッグ時に一時停止する)を設定してみましょう。

1. LED(red, green, 0);が記述されている行を選択し、行番号の左側をダブルクリックします。行番号の左側にブレークポイントが設定されていることを示す〇マークが表示されます。ブレークポイントを取り除く場合は再度ダブルクリックします。

109	/* USER CODE BEGIN 3 */
110	if(count < 10)
111	else red = 10*(20-count);
112	green = 100 - red;
11	
0114	LED(red, green, 0);
115	
116	HAL_Delay(100);
117	count = (count+1) % 20;
118	}
119	/* USER CODE END 3 */

ST-LINKとPCをUSBケーブルで接続し、ETCBボードとST-LINKを接続した状態で、ETCBボードの電源を入れます。EXTポートでのST-LINK差し込み場所を間違えないようにしてください。ST-LINK側とEXTポートのつなぎ方は次のようになります。

EXTポート番号	SWDIOポート名	ST-LINKコネクタ番号
14	GND	3
16	NRST	15
18	SWCLK	9
20	SWDIO	7
22	3.3V	1





図4 ST-LINKとの接続

- 実行メニューのデバッグを選択してデバッグモード(パースペクティブとよぶ)に移行します。
 この時ETCBボードのFlashメモリーへ先ほどビルドしたプルグラムが転送されます。
- 実行メニューの再開(F8)を選択します。プログラムの実行が始まり先ほど設定したブレークポイントの行で実行が一時停止します。
 実行メニューでステップオーバー(F6)を選択します。一時停止していた処理(LEDの輝度調

実行メニューでステッフォーハー(FO)を選択します。一時停止していた処理(LEDの輝度調整)が実行されETCBボード上のLEDが点灯します。

T/\97 - LED_TEST/Src/main.c - Atollic TrueSTUDIO for ARM	
ファイル(E) 編集(E) ビュー 実行(<u>R) ウィンドウ(W) ハル</u> プ(H)	
!!!! ⓑ ゐ 祢 治:८ ▶ !! ■ ♥ ◙ ۞ ↗ ♥ !৫ ৵ ♥ !♡ ♀ ♥ ♥ ४ 0 □ ♥ ⊞ ♥ ೫ ◘ .	クイック・アクセス 11 11 11 11 11 11 11 11 11 11 11 11 11
参デバッグ 🛛 🔥 🔨 🖓 ▽ 🗉 💁 ブ 🛛 🕪 変 盤式 盤 L 🦄 🖓 🗆	- SFRs 1111 レジスター 23 - ロ
🔺 💽 LED_TEST.elf [組みi] C++ アプリケーション] 🛛 🙀 🎇 🍇 🖓 🔍 🖳 🚍 🔩 🌣	約 ↔ 日 📑 🖻 🎽
▲ 🎲 LED_TEST.elf	名前 值 記述/説明
	▷ ## General Re General Pt
再開・一次停止・停止 ステッフィン	
ステップオーバー ,	
ステップリターン	< >
R mainc X	^
115 {	< >
116 red = 10*(20-count);	
11/ } 118 green = 100 - red; // LED(緑)の設定値	アウトライン ※ □ □
119 (J. EAC)/# a Cat	P ⊟ ↓ 2 & x ● # ~
※120 LED(red, green, 0); // 設定値の反映 121 □	adc.h
122 HAL_Delay(100); // 100ms休む	und.n tim.h
123	usart.h
125 }	gpio.h ÷ SystemClock Config(yoid) : yoid
126 /* USER CODE END 3 */	<
128 }	
129 🗸	📃 SWV 🖾 🔍 SWV 🕅 SWV 🗖 🗖
< >	🔀 🗢 🗙 🛼 🚮 🕂
🖳 コンソール 🛿 🕕 メモリー 🤣 FreeRTOS 🥅 SWV トレー 🥅 SWV 例外ト 🖹 問題 🕥 実行可能ファ 🖳 🗖	DEMO MODE: Unlock this feature with a low- cost Pro subscription. Click here to learn
■ ★ 🔆 🚉 📰 📮 🕶 🖬 ▼ 🗂 ▼	more.
LED_TEST.elf [組み込みC/C++ アプリケーション] gdb	ポート 0 ⊠
120 LED(red, green, 0); // ?????1????f	^
< · · · · · · · · · · · · · · · · · · ·	~
書き込み可能 スマート挿入 120:1	1

- 5. プログラムを再開するたびにLEDの色が変化することが確認できます。またデバッグ画面では変数の値を確認・修正することもできます。
- 6. デバッグを終了するときは、実行メニューの終了を選択します。

.デバッグ命令

- 再生(再開) 次のブレークポイントまで処理を進めます。次のブレークポイントに処理が移 らない場合(ブレークポイントが無い・ブレークポイントを入れた割り込みが 発生しなかったなど)は、プログラムは最後まで実行されます。
- ー次停止 処理を一時的に停止します。一次停止ボタンを押したタイミングで、停止位置 が変わります。
- 停止 プログラムを停止します。

- ステップイン 処理行(ブレーク中の位置)が関数だった場合にステップインを実行すると、 その関数の中に処理が移ります。
- ステップオーバー 次の命令まで処理を進めます。処理行が関数だった場合は、関数を実行してか ら次の処理行まで進みます。
- ステップリターン 処理行が関数の中である場合にステップリターンを実行すると、その関数の処 理を全て実行してから関数の呼び出し位置へ処理が移ります。
- 式ビュー デバッグパースペクティブのときに「式ビュー」を開いておき、モニタリングしたい変数などを記載しておくと、ブレークポイントで一次停止する度に、そのときの変数の中身を更新・表示します。
 デフォルトでは式ビューは開いていますが、ウィンドウメニューのビューから開くこともできます。

デバッグ中にブレークポイントで止まらない場合

プログラムの最適化がセットされている場合、デバッグ中にブレークポイントで止まらないで、その 前後の行でプログラムが停止することがあります。その場合はTrueSTUDIOのプロジェクトメニューか らプロパティを選択し、下図のように「C/C++ビルド」欄の「設定」を開き、「ツール設定」タブか





ら「C Compiler」>「Optimization」を開き「Optimization Level」を「None(-O0)」にしてください。

Raspberry Piへの応用

準備

本マニュアルではETCBとRaspberry Pi間のUART通信プログラムの作成について説明しますが、Raspbe rry Pi側のUART通信プログラムについては扱わないで、代わりにETCBのUSB1端子を使うことにしま す。

通常ETCBとRaspberry PiでUART通信するためには、EXTポートのUART1_TX/RX端子をジャンパで接続 します。同様にETCBのUSB1ポートもMCUのUART1と接続されていますので、USB1ポートから命令を 受け取るプログラムを作成すると、そのままRaspberry Piから命令を受け取るプログラムとなります。 そこで、本マニュアルでは、USB1(UART1)から命令を受け取り、サーボモーターを動かすプログラ ムを作成します。また、プログラム作成環境の整備や、ETCBの機能の標準的な使い方なども説明しま す。

Raspberry Pi側のプログラムについては別紙「ETCB Raspberry Piプログラミングマニュアル」を参照してください。

必要なソフトウェア・デバッガ

- プログラムの作成にはSTM32CubeMX、Atollic TrueSTUDIOが必要です。あらかじめWindows PC ヘインストールしておきます。できるだけ最新版をご用意ください。またSTM32CubeMXでF3フ ァミリーのファームウェアもインストールしておいてください。
- USB1ポートからも書き込みはできますが、デバッグできないので、できるだけST-LINK/v2を使う ようにしてください。USB1ポートから書き込みするときには、DFU端子にピンヘッダを取り付 けるか(半田づけ作業が必要です)、端子をショート出来るものが必要です。
- USB1ポートからサーボモーターのバイナリデータ命令を送るために、CoolTermというターミナ ルソフトウェアを使います。下記サイトからダウンロード・インストールしてください(ダウン ロードしたZIPファイルを適当なフォルダで解凍して、coolterm.exeを実行してください)。 http://freeware.the-meiers.org/

.Raspberry Piとハードウェア接続(UART)

始めにRaspberry PiとETCBを接続する準備をします。本マニュアルでは実際に行う必要はありません。

1. 半田付け

付属の2.54ピッチピンソケットをEXTポート横に半田付けしてください。ピンソケットを半田付け するときにソケットと基板に隙間ができないように注意してください。付属のΦ2.6L11樹脂スペ ーサとネジを使って、あらかじめRaspberry Piに取り付けた状態で半田付けすると失敗が少なくな ります。 2. ジャンパピンのとりつけ

図 3 EXTポート・端子名称を参考にして、UART_RXとUART_TX(Raspi)を付属のジャンパピンで 短絡(ショート)します。同様にUART_TXとUART_RX(Raspi)を短絡します。

Raspberry Piにとりつけ
 Raspberry PiのGPIOピンとずれないように取り付けてください。取り付け時はETCBとRaspberry Pi
 との間に必ずΦ2.6L11をはさみこんでください。取り付けたらネジ止めしてください。

STM32CubeMXでETCBプログラムひな形を作る

ETCBはUART、I2C、AD変換、GPIOなどの機能があらかじめハードウェアで決まっていますが、プロ グラム上で機能の実装を行うと間違えることがあります。このようなPinout設定(端子機能の設定) が簡単に設定できるSTMicroelectronics社のSTM32CubeMXソフトウェアを使用します。STM32CubeM Xは同社の提供しているHALライブラリを使って、MCUのPinout設定をGUIメニューで作って、プログ ラムのひな形を生成することができます。

ここではSTM32CubeMXを使ってETCBの周辺回路(ペリフェラル)設定について、ETCB標準設定例を 使って説明します。実際はユーザーが使用する周辺回路だけを設定してかまいません。

使用するサーボモーターは近藤科学社製のICSシリアルサーボモーターです。

_プログラムひな形の内容

本例題(ETCB_RobotBasic)ではRaspberry Piからサーボの位置命令を受け取り、ETCBへ取り付けられ たサーボモーターへ周期的にコマンドを送信します。その他、AD変換を自動的に行います。また、LE DをPWMで点灯します。また、できるだけDMA転送を使って、MCUへの負担を軽減します

使用するサーボモーターは近藤科学社製のシリアルサーボモーターとします。近藤科学社製のシリア ルサーボモーターはICSコマンドで動作しますので、このICSコマンドをサーボモーターに定期的に送 ります。Futaba社製シリアルサーボを使用するときは、後述のIcsSetPos関数の中身を適当に書き換 えてください。

機能	ポート	内容
AD変換	AD1~AD4ポート	■ AD変換は自動的に繰り返し行うように設定
	MCU温度	■変換後の値はDMA転送を使って指定変数へ自動保存する
	バッテリー電圧監視	■MCU温度とバッテリーの電圧監視
UART1	UART非同期通信でRaspbe	■通信プロトコルは、通信速度115200bps、データ長8ビッ
	rry Piと通信する	ト、1ストップビット、パリティ無し
	※Raspberry Pi側のプロ	■Raspberry Piからの命令をUART1受信割り込みで受信
	グラムはここでは扱いま	■Raspberry Piから命令が3ms来なかったら(タイムアウ
	せん	ト)、その時点まで受け取ったデータを1命令と判断
		■タイムアウト時間はSYSTICKタイマーで判断
		■命令に対する返事はDMA送信するように設定

		■UART1で受け取ったデータを解析して、目標位置をics_s ervo構造体のuint16_t dpos変数に保存
SI0	UART2、UART3 サーボモーターの駆動	 ■uint16_t dposからサーボモーターへの命令(ICSコマンド)を作って、SIOポートから周期的に送信 ■サーボ命令送信周期はTIM2タイマーを使用
LED	PWM(TIM1)を使ったフル カラーLEDの点灯	 PWMを使って、LEDを点灯 点灯(PWM)周期は10ms PWMカウンタ1カウント当たりのPWM出力時間は0.1msとすると100カウントで100%となりますので、フルカラーLEDのRGBはそれぞれ100段階で設定可能となります。 ETCBではPWM端子の極性が逆³なので、CH PoralityをLowに設定
ST-LINK	ST-LINKデバッグ対応	■ST-LINKを使ったデバッグができるように設定
SYSTICK	SYSTICK割り込み	■ SYSTICKをメニューから選ぶ(STM32CubeMXが自動的に1m s間隔のSYSTICKタイマーを生成)
TIM2	サーボモーターへの周期 的命令送信	■TIM2の周期を20msに設定 ■TIM2の周期割り込みが発生したら全サーボへ位置コマン ドを送る(20msごとにサーボモーターへコマンド送信)
RCC	クロック周波数	■72MHz設定

_STM32CubeMXの起動とプロジェクトの新規作成

STM32CubeMXを起動し、初期画面またはFileメニューから「New Project」(プロジェクトの新規作 成)を選択してください。New Projectダイアログが表示されますので、ETCBのMCUに合わせて、各 メニューを下記のように選んでください。または例題からプロジェクトを読み込んでもかまいませ ん。

- Series STM32F
- Lines STM32F302
- Package LQFP48

³端子の電圧レベルをHIGHにすると、LEDが消える

• MCUs List STM32F302C8Tx

New Project								x				
MCU Selector Boa	ard Sele	ector										
MCU Filters Series : STM32F3 Peripheral Selectio	v	Line STI	s: 432F3	002	Package : LQFP48 ns		~		More F	Filters ▼		
Peripherals	Nb	Max		MCU ^	Lines		Package	Flash	Ram	Eeprom	ю	₽
ADC 12-bit	0	11		STM32E302C6Ty	STM32E302		LOEP48	32	16	la	37	
ADC 16-bit	0	0		STM32F302C8Tx	STM32F302	Ν	LQFP48	64	16	0	37	
CAN	0			0TM021 00200TX	0 TM021 002	~~~~	LGTT 40	120	32	0	37	
COMP	0	4		STM32F302CCTx	STM32F302		LQFP48	256	40	0	37	1
DAC 12-bit		1										

図5 MCU選択

選択が完了したら、OKボタンを押してください。そうすると電源などの必要最低限のピン設定が完了 した画面(図6 MCU初期状態)が表示されます。なお同図でPinoutメニューおよびPinout設定画面 の名称は正式名称ではなく、分かりやすく言い換えたものです。正式にはPinoutメニューがPeripheral and Middleware treeで、Pinout設定画面はChip viewと言います。

「Pinout」、「Clock Configuration」、「Configuration」、「Power Consumption Calculator」という4つのタブがありますが、最初はPinoutを選択してください。

Pinout設定(端子設定)



図6 MCU初期状態

Pinout設定画面はマウスホイールで画像の拡大・縮小ができます。またマウスでMCUの位置をドラッ グできます。

RCC設定

RCC設定ではMCUのメインクロックのソースを決定します。

ETCBは8MHzのセラロックが実装されていますので、クロックソースをHSE=Crystal/Ceramic Resonat orにします。画面左側のPinoutメニューからRCCを選択し、High Speed Clock(HSE)を「Crystal/Cer amic Resonator」にします。RCC設定は必ずしも他の周辺回路設定より先にする必要はありません が、後から行う「Clock Configuration」の前に設定しておきます。

⊕- ● IRTIM ⊕- ● IWDG ⊕- ● OPAMP2	PC15 RCC_OSC_IN PF0		PA12 PA11
RCC Hich Speed Clock (HSE) Disable Disable Master Clock Dutnut Audio Clock Input (12S_CKIN) RC RTC Supa	RCC_OSC_OUT PF1 NRST VSSA VDD PA0	STM32F302C8Tx LQFP48	PA10 PA9 PA8 PB15 PB14
AD変換設定

例題では、main関数などで何もしなくても勝手にAD変換をして、指定した変数へAD値を保存してお くように設定します。具体的には、AD変換をしたあとに自動的にDMA転送⁴を行い、DMA転送が完了 すると割り込みが発生するようにセットします。DMA転送完了割り込み関数内では、保存したAD値 を実際の値(AD0ならAD値を電圧値に変換)します。

周辺回路(端子機能)をMCUの端子にセットするには、Pinout設定画面で直接端子を指定するか、Pin outメニューのPeripheralsにある「ADC1」欄から、AD1~AD4を有効にします。ETCBはAD変換ポート が4つあり、AD1~AD4ポートはそれぞれPA0~PA3に対応しています。PB0は電源入力端子(バッテ リー入力)とつながっているので、PB0端子も使えるようにします。また、MCU内部には温度計が備 わっているので、温度計の出力もAD変換して数値化(計測)できるようにします。

PinoutメニューのADC1の各ポートは全て「INx_Single-ended」にしてください。Single-endとはAD 変換器1つを1個ずつ変換するという設定です。



一緒に「Temperature Sensor Channel」にチェックを入れて、温度データもAD変換するように設定し ます。

.GPIO設定

ETCBのMCUは、PA4~PA7がそれぞれGPIO1~GPIO4に、PB1とPB2がそれぞれGPIO5とGPIO6にハード ウェア接続されています。使用する場合はPinout設定画面でPA4~PA7とPB1~PB2をGPIO_Outputま たはGPIO_Inputに設定します。

⁴ Direct Memory Accessのことで、CPUの演算回路を通さずに直接メモリー間でデータ転送をする方法のこと。CPUリソースを使わないでデータの保存ができます。



GPIOのInput/OutputはPinoutメニューで設定する項目はありません。

.UART1設定

UART1はEXTポートのUARTおよびCOM1(USB micro-Bコネクタ)に接続されています。今回はRaspb erry Piから命令を受け取るため、UART1を双方向通信可能にセットして、データを受信したときに受 信割り込みが発生するように設定します。返事(UART TX)はDMA転送で行います。DMA転送ではメ モリーにデータを書き込むだけで、演算回路を介さずにUARTからデータが出力されます。

PinoutメニューのUART1を開き、Modeを「Asynchronous」に設定します。Pinout設定画面ではPA9と PA10が自動的にUART1_TXとUART1_RXにセットされますが、ETCBではUART1をPB6とPB7にハード接 続していますので、あらためてPB6をUART1_TXにして、PB7をUART1_RXに設定して下さい。



.UART2、UART3設定

UART2、UART3はそれぞれS7~12とS1~6にハードウェア接続されています。シリアルサーボを動か す場合はUART2とUART3を「Single Wire(Half Duplex)」設定にして、TXポートだけで送受信できる ように設定します(送信と受信はプログラム内で切り替えます)。

Pinoutメニューを下図のように設定します。Pinout設定画面での端子設定は自動設定のままにしてお きます。



.PWM(フルカラーLED)設定

ETCBのフルカラーLEDは赤・青・緑それぞれにPWM(TIM1出力)端子がハードウェア接続されていま す。PWMのデューティ比を調節することでフルカラー表示ができます。赤・青・緑それぞれはTIM1_ CH1、TIM1_CH2、TIM1_CH3端子に接続されていますので、PinoutメニューのTIM1を開き、Channel1 ~Channel3をそれぞれPWM Generation CH1~3に設定します。PWM出力設定にするとPinout設定画面 では自動的にPA8~PA10がTIM1出力設定になります。Clock Sourceの指定は不要です。



.タイマー(TIM2)設定

TIM2を使って周期的に割り込みを発生させて、サーボモーターへ命令を送信します。TIM2だけ32ビッ トタイマーで、残りは16ビットタイマーですので注意してください。





.デバッガ設定

最後にST-LINKを使ったデバッグができるように端子設定をします。ST-LINKでデバッグしない場合やSWDIOとSWCLKラインをGPIOで使用したいという場合は不要です。

ST-LINKはPinoutメニューのSYSを開き、Debugメニューから「Serial Wire」を選択します。自動的にP A13とPA14がSWDIOとSWCLKにセットされます。

同時にSysTickタイマーを使用できるように、Timebase Sourceメニューを「SysTick」にセットしま す。SysTickタイマーとは、クロック周波数をMCUが数えて、簡単にタイマーとして使う仕組みです。 初期状態では1msタイマーが自動的に設定され、**void HAL_SYSTICK_Callback ()**という割り込み が発生します。SysTickタイマーはその他の割り込みなどに影響され、あまり正確ではありませんの で、制度を要求されないタイマーとしてお使いください。



Raspberry Piへの応用

.サーボON/OFFスイッチGPIO設定

ETCBはサーボ電源をカットすることができます。回路ではPA15端子をGPIO Outputに設定することで PA15をLow(0V)にすると電源OFF、High(3.3V)にすると電源がONになります。

プログラム上でPA15端子からHighを出力しなくてもデフォルトでサーボの電源は入りますが、PA15 端子をGPIO Outputに設定しないと端子電圧が不定となり電源が入りません。Pinout設定画面で直接 端子をクリックし、メニューからGPIO Outputに設定してください。



クロック設定

Pinoutの設定が済んだら、MCUのクロックを「Clock Configuration」タブで設定します。最大72MHz まで設定できます。クロックを初めて設定するときは、クロックの自動設定を行うかどうかを答える ダイアログが表示されますので、Yesボタンを押して先に進みます。

	Clock configuration					
?	Do you want to run automatic clock issues solver ?					
	Otherwise you can do it later by clicking on button "Resolve Clock Issues" 🥑					
	Do not show this message again.					
	Remember my decision for next projects.					
	Yes No					

クロック設定画面が表示されますので、以下の項目通りに画面上のメニューを選んでください。一番 左端にある「Input frequency」のすぐ横に「HSE」と表示されていないときは、Pinoutタブに戻り、R CCの設定をやり直してください。



いずれかのメニューを選んだときに、他の項目が赤く表示された場合は、その選択は動作しない設定ですので、選択をやり直してください(例えばPLLMulをx10以上にした場合など)。

- ① Input frequency HSEとなっていることを確認。そうで無い場合はRCCを再設定
- ② PLL Source Mux HSEを選択(●マークが入る)
- ③ System Clock Mux PLLCLKを選択(●マークが入る)
- ④ PLLMul x9 (8[MHz] x 9=72MHzにセットされます)
- ⑤ APB1 Prescaler
 APB1クロックはメインクロックの半分で動作するため、「/2」を選択します。
- ⑥ ADC1 Prescaler
 AD変換は最長で601.5Cycle(601.5/7200000 = 8.5[us])が選択できま す。ADC1の変換速度が速すぎると、変換後の割り込み発生頻度が高くな り他の処理ができなくなる事があります。ETCBは最大72MHzで動作しま すので、Prescalerで変換時間を1/4くらいにしておいた方が良いでしょ う。

最終的なクロックが72MHzとなるようにしてください。クロックを低く設定すると消費電力が下がり ますが、タイマー設定も変わりますので注意してください。

Configuration

ConfigurationタブではPinout設定画面で設定した各周辺回路の詳細な設定や、割り込みなどの設定を 行います。設定の簡単なものから説明します。 最初にConfigurationタブを選択すると、Multimedia、Connectivity、Analog、System、Controlの 5 つのカテゴリが表示され、それぞれに周辺回路名が表示されたボタンがあります。設定したい機能の ボタンを押すと設定ダイアログが表示されます。



.TIM1 (フルカラーLED) 周期設定

まずLED点灯用にPWM信号をTIM1で周期的に発生させます。ここではPWM周期(TIM1の周期)を10ms、分解能(LEDのRGB光の強弱)を100段階で調整できるようにします。まずTIM1の周期はメイン クロックをPrescalerで割った数値となりますので、10msの周期を100段階で調整するには、タイマーの1カウントが0.1ms(0.1ms×100段階=10ms)にする必要があります。

TIM1 Configuration							
🗹 Parameter Settings 🗹 User Constants 🗹 NV	🖋 Parameter Settings 🚽 User Constants 🖋 NVIC Settings 🚽 DMA Settings 🚽 GPIO Settings						
Configure the below parameters :							
Search : Search (CrtI+F)							
Counter Settings							
Prescaler (PSC - 16 bits value)	7200						
Counter Mode	Up						
Counter Period (AutoReload Register - 16	i b 100 N						
Internal Clock Division (CKD)	No Division 😼						
Repetition Counter (RCR - 16 bits value) 0							
Trigger Output (TRGO) Parameters							

今回はMCUのクロックは72MHzですので、Prescalerが1の場合は1カウント当たりの時間は1/72,000,00 0秒となります。これを1カウントあたり0.1ms(10msの1/100)にするにはPrescalerを7200とする と、7,200/72,000,000秒=0.0001秒=0.1msとなります。

次にCounter Period(周期カウント)を100にして、PWM周期を0.1ms×100=10msとします。

実際のプログラムではRepetition Counterの値をセットしてデューティ(RGB色の明るさ)を調整しま す(後述)が、ここでは0にしておきます(PWM信号を出力しない)。 ETCBのLEDは端子を0V(Low)で点灯しHighで消灯するように、極性が逆になっています。そこでP WM Generation Channel(1~3)の全てのCH Polarityを「Low」にして、PWM出力がONのときはLow 電圧(0V)を出力し、OFFのときはHigh電圧(3.3V)を出力させます。デューティ比を大きくするとL EDが明るく点灯します。タイマーをPWM出力に設定すると周期的に割り込みを発生させなくても自 動的に信号が出力されます。極性を逆にしないとデューティを大きくするとLEDが暗く点灯します。

 TIM1 Configuration 							
🖋 Parameter Settings 🗹 User Constants 🗹 NVIC S	Gettings 🗹 DMA Settings 🗹 GPIO Settings						
Configure the below parameters :	Configure the below parameters :						
Security (Catter)							
PWM Generation Channel 1	Disable A						
Mode	PWM mode 1						
Pulse (16 bits value)							
Fast Mode	Disable						
CH Polarity	Low						
CH Idle State	Reset						
PWM Generation Channel 2							
Mode	PWM mode 1						
Pulse (16 bits value)	0						
Fast Mode	Disable						
CH Polarity	Low						
CH Idle State	Reset						
PWM Generation Channel 3							
Mode	PWM mode 1						
Pulse (16 bits value)	0						
East Mode	Disable						
CH Polarity	Low						
CH Idle State	Reset						

.*TIM2*

TIM2タイマーはサーボに適当な周期で命令を出すために使用します。今回は周期を20msに設定しま すが、あとからタイマーの周期を細かく設定できるように、Prescalerを72、Counter Periodを20000に セットします(周期20ms、分解能0.001ms)。

TIM2タイマーで20ms毎に割り込みを発生させるために、「NVIC Settings」タブを開き、TIM2 Global I nterruptにチェックマークを入れます。

TIM2 Co	nfiguration ×						
🖋 Parameter Settings 🗹 User Constants 🗹 NVIC Se	🖋 Parameter Settings 🚽 User Constants 🚽 NVIC Settings 🚽 DMA Settings						
Configure the below parameters :							
Search : Search (Crtl+F) 🗢 🛧							
😑 Counter Settings							
Prescaler (PSC - 16 bits value)	72						
Counter Mode	Up						
Counter Period (AutoReload Register - 32 bits	10000						
Internal Olock Division (OKD)	No Division						
□ Trigger Output (TRGO) Parameters							
Master/Slave Mode	Disable (no sync between this TIM (Master) and its Slaves						
Trigger Event Selection TRGO	Reset (UG bit from TIMx_EGR)						

TIM2 Configuration						
🖋 Parameter Settings 🗹 User Constants	🖋 DMA Settings					
Interrupt Table	Enabled	Preemption Priority	Sub Priority			
TIM2 global interrupt	v	0	0			
	2					

UART2とUART3の通信プロトコル設定

UART2とUART3を使ってサーボモーターへ命令を送ります。今回は近藤科学社製のICSシリアルサーボ モーターが動くように設定しますので、通信速度115200bps・データ長9ビット・パリティEVEN(偶 数)・1ストップビットにセットします。データ長はパリティビットを含みますので、偶数・奇数パ リティを使う場合はデータ長8ビットに1ビット追加してください。

USART2 Configuration					×	
🖋 Parameter Settings 🚽 User Constants 🖋 NVIC Settings 🗹 DMA Settings 🚽 GPIO Settings						
Configure the below para	meters :					
Search : Search (CrtI+F	Search : Search (CrtI+F)					
🖻 Basic Parameters						
Baud Rate			115200 Bits/s			
Word Length			9 Bits (including Par	ity)		
Parity			Even	N		
Stop Bits			1	45		
Advanced Parameters						

.UART1 (RX割り込み・TX DMA転送)

UART1はRaspberry Piからの通信を受信したときに割り込みを発生させて、コマンド受信完了後、結果 を返すようにします。返事をするときはDMA転送を使って、できるだけMCU演算回路を使わない制御 を行います。本例題では設定のみで実際は返事をしません。

ConfigurationタブでConnectivityカテゴリのUSART1ボタンを押すと、下記のようにUSART1 Configura tionダイアログが表示されますので、UART2、UART3同様に通信プロトコルを設定します。通信速度1 15200bps・データ長8ビット・パリティNone(なし)・ストップビット1にセットします。

USART1 Configuration						
💞 Parameter Settings	🖋 User Constants 🛯 🖋 NVIC Settings 🚽 DMA Settings 🚽 GPIO Settings					
Configure the below para	ameters :					
Search : Search (Crtl+F	Search : Search (OrtI+F) 🗢 🔶					
😑 Basic Parameters						
Baud Rate	115200 Bits/s					
Word Length	8 Bits (including Parity)	~				
Parity	None					
Stop Bits	1					
Advanced Parameters						

つぎにUSART1 ConfigurationダイアログのNVIC Settingsタブを開き、USART1 global interruptの「Ena bled」にチェックマークを入れ、受信割り込みを有効にします。

USART1 Configuration					×
🖋 Parameter Settings 🚀 User Constants	🖋 NVIC Settings	🛷 DMA Sett	tines 🔍	🖉 GPIO Settings	
Interrupt Table		E	nabled	Preemption Priority	Sub Priority
USART1 global interrupt / USART1 wake-up i	interrupt through EXT	Пline 25	<	0	0
			\Im		

つぎにDMA Settingsタブを開き、右下にある「Add」ボタンを押します。画面のDMA Request欄にメ

Image: Object of the second					
🗹 Parameter Settings 🗨	🖉 User Constants 🛛 🎻	NVIC Settings 🗹 DMA Settings 🗸	🖋 GPIO Settings		
DMA Request	Channel	Direction	Priority		
USART1_TX	DMA1 Channel 4	Memory To Peripheral	Low		
			Addボタンを押	して、DM	
			A Requestを注	追加する	
-DMA Request Settings			Add Delete		
		Peri	ipheral Memory		
Mode Normal	¥	Increment Address			
		Data Width Byte	V Byte V		
		Appl	y Ok Cancel		

ニューが1列追加されますので、「Select」欄から「USART1_TX」を選択します。DMAチャンネル (DMA1 Chennel4) やDirection(転送方向・Memory To Peripheral)は自動設定のままで問題ありま せん。MCUが忙しくないときに転送すれば良いので、Priorityは「Low」のままにしておきます。ま た、ダイアログの下にある「Data Width」も、Peripheral=「Byte」、Memory=「Byte」(バイトデ ータをバイトデータのまま送信)にしておきます。

これで、指定する変数にデータを書き込みDMA転送を登録するだけでUART1よりデータが自動的に送 信されます。

AD変換設定(変換後DMA転送+DMA転送完了割り込み)

AD変換は各チャンネルのAD変換終了後に、変換データ(12ビットAD値)をメモリーへDMA転送しま す。全てのチャンネルがDMA転送完了したら転送完了割り込みを発生させて、変換データを割り込み 関数の中で実際の値(電圧、温度、バッテリ電圧)に変更します。

ConfigurationタブのAnalogカテゴリにあるADC1ボタンを押します。つぎにADC_Regular_Conversion Mode欄のNumber Of Conversions(AD変換するポートの数)を6(AD1~4、電源、温度で合計6個) にします。

ADC1 Configuration							
؇ Parameter Settings	🖋 Parameter Settings 🚽 User Constants 📢 NVIC Settings 📢 DMA Settings 🚽 GPIO Settings						
Configure the below para	ameters :						
Search : Search (Ortifie	Sourch (Catter)						
ADC_Settings							
Clock Presc	aler	ADC Asynchronous clock mode					
Resolution		ADC 12-bit resolution					
Data Alignm	ient	Right alignment					
Scan Conve	rsion Mode	Enabled					
Continuous	Conversion Mode	Enabled					
Discontinuo	us Conversion Mode	Disabled					
DMA Contin	uous Requests	Enabled	<u> </u>				
End Of Con	version Selection	End of single conversion					
Overrun beh	aviour	Overrun data overwritten					
	Auto Wait	Disabled					
ADC_Regular_Convert	rsionMode						
Enable Regu	ular Conversions	Enable					
Number Of (Conversion	6					
External Tri	gger Conversion Edge	None					
🗄 Rank		1					
🗄 Rank		2					
🗄 Rank		3					
E Bank		4					
DMA Continuous Re DMAContinuousReques Parameter Descripti Enable/Disable DMA Co	quests ts i on: ontinuous Requests						
		Apply Ok	Cancel				

Number Of Conversions の数だけRankができあがりますので、下記のように設定してください。

- Rank1~4 Rank1=Channel1、Rank2=Channel2、Rank3=Channel3、Rank4=Channel4
- Rank5 Channel11(電源)
- Rank6
 Channel Temperature Sensor
- Sampling Time 61.5Cycle~601.5Cycle⁵ (全てのRank)

ADC_Settings欄は下記のように設定してください。

- Scan Conversion Mode Enabled(自動変換・ソフトウェアトリガー⁶ではない)
- Continuous Conversion Mode Enabled (連続変換)
- End Of Conversion Selection End of sequence conversion(チャンネル毎に変換)

上記設定を行うと、MCUが自動的にAD変換⁷を行い、変換後のデータを指定された変数へ自動的に転送(データの更新)しますので、ユーザーは指定した変数を読み取るだけで良いことになります。

つぎにDMA転送設定を行います。DMA転送タブを開き、「Add」ボタンを押してDMA Request項目を 追加します。Select欄をADC1にセットします。ChannelとDirectionとPriorityは初期状態のままで問題 ありません。

ダイアログ右下にあるData Width欄はPeripheral=「Half Word」Memory=「Half Word」にしておき ます。Half Wordは16ビットのことで、ETCBではAD変換値が12ビットの解像度があるため、データを 格納するメモリーは16ビット用意しておくという意味です。データを格納するメモリーはプログラム で設定します。

⁵1Cycleは1/72MHzの時間がかかります。サンプリング周期を短くすると、他の処理に移れなくなる 場合があり、変換精度も悪くなりますので、601.5Cycleを選択した方が良いでしょう。

⁶ソフトウェアトリガー:プログラムで変換を開始すること。

⁷通常はソフトウェアトリガーで繰り返し変換を行う必要があります。

٥	ADC	1 Configuration		×
🖋 Parameter Settings 😽	🕈 User Constants 🛛 🖋 NV.	IC Settings 🗹 DMA Settin	es 🎻 GPIO Se	ttings
DMA Request	Channel	Direction	Priority	
ADC1	DMA1 Channel 1	Peripheral To Memory	y Low	J
			F	Add Delete
DMA Request Settings -			Peripheral	Memory
Mode Circular		Increment Address		
		Data Width Ha	alf Word 🖌 👻	Half Word 🗸
			Apply	Ok Cancel

また、「DMA Request Settings」のModeを「Circular」にセットして、自動的に繰り返しDMA転送が 行われるようにします。

つぎにNVIC Settingsタブを開き、DMA1 channel1 global interruptのEnabled欄にチェックマークが入 れてください。EnabledのときはDMA転送完了時に割り込みが発生します。ADC1 interruptのEnabled 欄にチェックマークを入れるとAD変換が完了する度に割り込みが発生してしまい、多チャンネルを変 換している場合は割り込み発生頻度が高くなり、他の機能が実行されなくなる恐れがあります。今回 はチェックしないでください。

ADC1 Configuration					
🖋 Parameter Settings 🖋 User Constants 🖋 NVIC	Settings 🧹 DM	1A Settings 🗹 GPIO S	ettings		
Interrupt Table	Enabled Preemption P		Sub Priority		
DMA1 channel1 global interrupt	\checkmark	0	0		
ADC1 interrupt		0	0		
	\searrow				

割り込み優先順位設定

割り込み関数の優先順位を設定します。優先順位の高い割り込みを処理している最中は、優先順位の 低い割り込みは実行されません。逆に優先順位の低い割り込みを処理している間でも、優先順位の高 い割り込みが発生すると、いったん優先順位の高い方へ処理が移動し、それが終わると元の割り込み 位置へ処理が戻ります。

UARTの受信割り込みを最優先にしないと、データの取りこぼしが起きることがありますので、UART 受信割り込みは最優先に設定します。SystemカテゴリからNVICボタンを押して、「NVIC Configuratio n」ダイアログを表示します。今回はTIM2のタイマー周期割り込みとUART1受信割り込みを使います ので、USART1受信割り込みの優先度を0、TIM2割り込みを1にしています。

Priority Group

割り込みの優先順位の付け方をグループ(プリエンプションとサブプライオリティの組み合わせ)に よって変えることができます。優先順位は多重割り込みが発生した場合の実行優先順位です。優先順 位の高い処理をしている最中は優先順位の低い割り込みは発生しません。

NVIC Configuration	n			×
✓ NVIC ✓ Code generation				
Priority Group 4 bits for pre-emption priority 0 bits for subpriority 🗸	🗌 Sort b	y Premption Priority	and Sub Pror	ity
Search (CrtI+F)	Show	only enabled interru	ipts	
Interrupt Table	Enabled	Preemption Priori	Sub Priority	
Non maskable interrupt	~	0	0	
Hard fault interrupt	~	0	0	
Memory management fault	\checkmark	0	0	
Pre-fetch fault, memory access fault	\checkmark	0	0	
Undefined instruction or illegal state	~	0	0	
System service call via SWI instruction	~	0	0	
Debug monitor	~	0	0	
Pendable request for system service	~	0	0	
Time base: System tick timer	~	0	0	
PVD interrupt through EXTI line16		0	0	
Flash global interrupt		0	0	
RCC global interrupt		0	0	
DMA1 channel1 global interrupt	\checkmark	0	0	
DMA1 channel4 global interrupt	~	0	0	
ADC1 interrupt		0	0	
TIM1 break and TIM15 interrupts		0	0	
TIM1 update and TIM16 interrupts		0	0	
TIM1 trigger, commutation and TIM17 interrupts		0	0	
TIM1 capture compare interrupt		0	0	
TIM2 global interrupt	✓			
USART1 global interrupt / USART1 wake-up interrupt through EXTI line	-	0	0	
USART2 global interrupt		0	0	
USART3 global interrupt		0	0	
Floating point unit interrupt		0	0	
				\mathbf{v}
☑ Enabled Preemption Priority	Sut	o Priority 0 🗸		
	Ap	oply Ok	Cancel	

優先順位は全部で4ビット(0~15)を設定できます。上の図では4ビット全てをプリエンプションに
 割り当てていますが、例えばプリエンプションに2ビット(0~3)・サブプライオリティに2ビット
 (0~3)を割り当てることもできます。振り分けたときはプリエンプションを同じ優先度にしてサブ
 プライオリティで優先度を変えることもできます。プリエンプションはサブプライオリティよりも優

先されます。また0に近い方の優先度が高くなります。

優先度に同じ数値を割り当てると割り込みがうまく発生しないことがあります。必ずユニークな優先 順位を指定してください。

プログラムの生成

プロジェクトの保存

STM32CubeMXのFileメニューから「Save Project」を選択するか、Projectメニューから「Settings」を 選択します。新規作成時は「Project Settings」ダイアログが表示されますので、下記のように設定し てください。

۲				
File	Proj	ect Pinout Windo	w Help	
Ľ	۵	Generate Code	Ctrl+Shift+G	urrent Signals Placement
Pine	2	Generate Report	Ctrl+R	wer Consumption Calculato
Con	X	Settings	Alt+P	^
	÷	FATFS Sett	tings for softwa	re Project
	<u>ک</u>	FREERTOS		
	₽ ~ (0)	USB_DEVICE		
	Pro	ject Location	True	sTUDIOで使ってい

● Project Name 任意の名前を設定。ここではETCB_RobotBasic

Code Generator Au	vanceu pettings			
Project Settings				
Project Name				
ETCB_RobotBasic				
Project Location				
C:¥Users¥chinoken¥Atollic	¥TrueSTUDIO¥ARM_work	space_6.0		Browse
				3
Toolchain Folder Location				
C:¥Users¥chinoken¥Atollic	¥TrueSTUDIO¥ARM_work	space_6.0¥ETCB_Roboti	Basic¥	
Toolchain / IDE				
TrueSTUDIO		🗸 🗹 Generate Un	der Root	
Minimum Heap Size Minimum Stack Size	0×200 0×400			
Mcu and Firmware Package				
Mcu Reference				
STM32F302C8Tx				
Firmware Package Name a	nd Version			
STM32Cube FW F3 V160				
01110200001 W_1 0 V 1.0.0				

次にProject SettingsダイアログのCode Generatorタブを開きます。各項目のチェック状態を下図のようにします。

• Toolchain/IDE TrueSTUDIO

٢	Project Settings	x
F	roject Code Generator Advanced Settines	
	STM32Cube Firmware Library Package	
	Ocopy all used libraries into the project folder	
	Copy only the necessary library files	
	○ Add necessary library files as reference in the toolchain project configuration file	
	Generated files	
	Generate peripheral initialization as a pair of c/h files per peripherals	
	Backup previously generated files when re-generating	
	✓ Keep User Code when re-generating	
	✓ Delete previously generated files when not re-generated	
	HAL Settings	
	Get an mee pins as analog (to optimize the power consumption)	
	Template Settings	
	Select a template to generate customized code Settings	
	Ok Cancel	

Generated files欄の「Generate periheral initialization as …」にチェックを入れると、AD変換、タイマー、DMA転送など機能ごとにC言語ソースファイルとヘッダファイルが生成されます。大規模プログラムを作成するときはプログラム構成や規模が確認しやすくなる反面、変数や関数の取り扱いがやや複雑になります。本例題ではチェックを入れないで、初期化関数は全てmain.cにまとめて保存されるように設定します。

設定が終わったらOKボタンでダイアログを閉じます。

.コード (プログラムひな形)の生成

STM32CubeMXのメインメニューにある、Projectメニューから「Generate Code」を選択してください。Project Settingsで設定したフォルダにプログラムのひな形が保存されています。

プログラム編集

プログラム編集

まず19ページの「*TrueSTUDIOを使ったソースコード編集・デバッグ*」と同様に、作成したひな形を インポートします。インポート時に「ルート・ディレクトリの選択」をチェックして、ひな形(フォ ルダ)があるディレクトリ(フォルダ)を選択します。そうすると「プロジェクト」欄にインポート 可能なプロジェクトが表示されますので、チェックマークを入れて終了ボタンを押します。

プロジェクトのインボート 既存の Eclipse プロジェクトを検索するティレクトリーを選択します。
 ・ディレクトリーの選択(T): C:¥Users¥chinoken¥Atollic¥TrueS v
○)/ − パイン・パイルの違水(A): プロジェクト(P):
 ✓ ETCB_RobotBasic(C:¥Users¥chinoken¥Atollic¥TrueSTUI
<
⑦ < 戻る(B) 次へ(N) > 終了(F) キャンセル

インポートが正常に実行されると、次のようなプログラム編集画面が表示されます。

a	C - Atollic TrueSTUE	IO for ARM		- • ×
ファイル(F) 編集(E) ソース(S) リファクタリング(T) ビュー ナビゲート(N) 検索(A) プロジェクト(P)	実行(R) ウィンド	ウ(W) ヘルプ(H)	
	6 6 8 ∞ # 🔍 🐂 🖉 🗸 • (+ +	• -> • 🚺 🛛	□ • = • 8 ⊡	<u> ታイック・アクセス</u> 🔡 🗟 C
D JUSZDI-JOZDU-J- ≥ □ □			- 0	BE 7 22 [№] 3 [□] □ 最示するアウトラインはありません。
	🔝 問題 🛿 🧔 タスク 📄 コンソール 🔲 プロパティー		🗟 Build Analyzer 🔀	~
	0 項目	<u>6</u> 9 ⊽	DEMO MODE: Unlock this feat subscription. Click here to lea	cure with a low-cost Pro m more.
	記述/説明	リソース		
			Memory Regions Memory I	Details
			Region Start addr	ess Size Free
	<	>	`	,
0 項目が選択されました。				

プログラム編集

.プロジェクト構成の確認とソースコード・ヘッダファイルの追加

TrueSTUDIOのプロジェクトエクスプローラーウィンドウにいくつかファイルとディレクトリが表示されますが、ユーザーがソースコード(.c)を追加する場合は、「Src」ディレクトリに追加してください。

今回はヘッダファイル「user.h」とソースコードファイル「user.c」をそれぞれIncディレクトリとSrc ディレクトリに追加します。追加するときは追加したいディレクトリのアイコンを右クリックして、 新規(N)メニューからソース・ファイルまたはヘッダー・ファイルを選んでください。

쀁 プロジェクト・エ	クスプローラー 🛛	🖻 😫 💱 ▽ 🗖 🗖	📴 ሥንኮライン 😫 デンプレート 🔘 Make ターゲット 🗐 タスク・リスト	- 0
⊿ 😂 ETCB_F ⊳ 🖑 バイナ	lobotBasic		In user h	# ~
Inclusion	ıdes		 huart1 : UART_HandleTypeDef 	
Driv	ers		huart2 : UART_HandleTypeDef	
	新規(N)	+	プロジェクト(R)	
⊳	次ヘジャンプ(I)		デンプレートからファイル	
	新規ウィンドウで開く(N)		° วิราสม	
a 😕 s	Show In	+	フォルダー vo	oid
2	⊐ピ–(C)	Ctrl+C): void
	貼り付け(P)	Ctrl+V	C ソース・ファイル : V	oid
▶ [🗙	削除(D)	削除		
Þ 🧀 🔍	コンテキストから除去	Ctrl+Alt+シフト+下^	Nyダー・ファイル	
	ソース		C フロジェクト	
2	1分割(∀)… 久前を亦再(M)	E2	C++ プロジェクト	
		12	TrueSTOREからサンプルプロジェクトをダウンロードする。	
	インホート(1)		📑 その他(O) Ctrl+N	
<u>(~)</u>	100m 1(0)			
	クスプローラー 🕱	- \$ 8 ~	P アウトライン 23 日 テンプレート Make ターゲット 目 タスク・リスト	- 8
L D D D D D D D D D D D D D	クスプローラー 🛛	□\$ \$ ▼□□	 	# △
プロジェクト・エ ダ ETCB_F ダ	クスプローラー w kobotBasic +リー		E: アウトライン 23 ≧ テンプレート @ Make ターゲット 員 タスク・リスト ② E: J ⁴ Z ≷ ≷ ● ■ user.h	# △
プロジェクト・エ	クスプローラー 🛛 lobotBasic ドリー udes ers		E: アウトライン 23 E テンプレート Make ターゲット 原 カスク・リスト ジ ロ パート ・ ロuser.h ・ huart1 : UART_HandleTypeDef ・ huart2 : UART_HandleTypeDef	# △
 プロジェクト・エ グロジェクト・エ グロジェクト・エ デロシェクト・エ デロション デロシェクト・エ デロシェクト・エ デロション デロ	クスプローラー 図 lobotBasic ドリー udes ers		E: アウトライン ☆ ≧ テンプレート ● Make ターゲット 目 タスク・リスト Wake ターゲット 目 タスク・リスト User.h huart1: UART_HandleTypeDef huart2: UART_HandleTypeDef huart2: UART_HandleTypeDef	# ⊽
プロジェクト・I ▲ ご ETCB_F ▶ ポ バイフ ▶ ご Incli ▶ ご Driv ▲ ご Inc ▶ ご Incli ▶ ご Incli ▶ ご Inc ▶ ※ Inc	クスプローラー 22 kobotBasic サリー udes ers nxconstants.h	- 	P7トライン ☆ ≦ デンプレート ● Make ターゲット 員 タスク・リスト Warch user.h huart1 : UART_HandleTypeDef huart2 : UART_HandleTypeDef huart3 : UART_HandleTypeDef html : TIM_HandleTypeDef	# ~
↓ ↓	クスプローラー 22 idootBasic */J- ides ers nxconstants.h tm32f3xx_hal_conf.h tm32f3xx_lt.h	-D\$\$ ₽ ▼ = D	P71512 23 S7272-h Make 9-549 F 9239-112h If 1 12 1	# ~
Image: The second se	クスプローラー 22 idootBasic +'J- udes ers nxconstants.h tm32f3xx_hal_conf.h tm32f3xx_it.h ser.h		Image: Second Secon	#
¹ → 1→ 5×50 × 1→ 1→ 1→ 5×50 × 1→ 1→ 1→ 1→ 1→ 1→ 1→ 1→ 1→ 1→ 1→ 1→ 1→	クスプローラー 22 idootBasic +'J- udes ers nxconstants.h tm32f3xx_hal_conf.h tm32f3xx_l.h ser.h 新規(N)		E: 771572 S3 Sign 5272-b ● Make 9-549 b 9239-112b I user.h I user.h I user.h I user.h I user.h I user.h	roid
¹ → 1→ 5×50 + ×1 ¹ → 1→ 5×50 + ×1 ¹ → 1→ 5×50 + ×1→ 1→ 1→ 1→ 1→ 1→ 1→ 1→ 1→ 1→ 1→ 1→ 1→ 1	クスプローラー 22 kobotBasic **/J- udes ers nxconstants.h tm32f3xx_hal_conf.h tm32f3xx_l.h ser.h 新規(N) 次ペジャンプ(1)		Image: Second Secon	roid ₩ ▼
	クスプローラー 22 idootBasic +リー ides ers nxconstants.h tm32f3xx_hal_conf.h tm32f3xx_it.h ser.h 新規(N) 次へジャンプ(I) 新規ウインドウで聞く(N)			roid ₩, ▽
	クスプローラー 22 iobotBasic +リー udes ers nxconstants.h tm32f3xx_hal_conf.h tm32f3xx_it.h ser.h 新規(N) が、シャンプ(I) 新規ウインドウで開く(N) Show In		P7)トライン 23 テンプレート ● Make ターゲット ■ タスク・リスト Wake ターゲット ■ タスク・リスト Wake ターゲット ■ タスク・リスト Wake ターゲット ■ タスク・リスト Wake ターゲット ■ パー・ Wake ターゲット ■ タスク・リスト Wake ターゲット ■ タスク・リスト Wake ターゲット ■ パー・ Wake ターゲット ■ パー・ State State Yake State Stat	roid): void roid
	クスプローラー 22 tobotBasic +リー ddes ers nxconstants.h tm32f3xx_hal_conf.h tm32f3xx_it.h ser.h 新規(N) 次へシャンプ(I) 新規(ウィンドウで開く(N) Show In コピー(C)	Ctrl+Q	Image: Second Secon	roid): void roid
	クスプローラー 22 tobotBasic +リー udes ers nxconstants.h tm32f3xc_ih.h ser.h 新規(N) 新規ウインドヴで開く(N) Show In レビー(C) 齢力付け(P)	Ctrl+C	Image: State Sta	roid): void void
	クズローラー 22 tobotBasic HJー ddes ers nxconstants.h tm32f3xx_hal_conf.h tm32f3xx_ihal_conf.h tm32f3x_ihal_conf.h tm32f	Ctrl+C Ctrl+C	P7トライン 23 Fンプレート Make ターゲット ダスク・リスト wer.h huart1: UART_HandleTypeDef huart2: UART_HandleTypeDef huart3: UART_HandleTypeDef hturt1: TIM_HandleTypeDef LEDColor(uint16_t, uint16_t, uint16_t): void IcsSetPos[IcsServo*]: HAL_StatusTypeDef Parse(): CommandStatus TDジエクトパル フィル アンプレートからフィル アンプレートからフィル アンブルート	roid): void void

.例題プログラム概要(内容の確認)

最初にプログラム上の各機能の実装について説明します。

1. 初期設定について

DMA転送設定やUART通信設定など各機能設定は、STM32CubeMXによって初期化関数がmain.c に自動生成されています。自動生成された関数名は最初に「MX_」、最後に「_lnit」とついてい ます。また、これらの初期化関数はすでにmain関数の冒頭で呼び出されていますので、ユーザ ーは初期化(STM32CubeMXで行った設定)について改めてプログラムを書く必要はありませ ん。

初期化が終わった後で、ユーザーはタイマーや割り込みを開始するコードを書きます。

プログラム編集

- main.cでのプログラム作成上の注意 STM32CubeMXで生成したプログラムのmain.cでは、ユーザーが関数を記載して良い場所が決ま っています。「/* USER CODE BEGIN 0 */」から「/* USER CODE END 0 */」などです。同様にin clude定義をする場所や、ローカル変数を記載する場所も決まっています。これらはSTM32Cube MXでPinoutを修正してコードを再度出力するときに、ユーザーが書いたコードを削除してしま わないための目印となっています。自由に関数を作成したいときは、main.c以外のCソースファ イルを作成してください。
- 3. PWM信号でLEDを点灯(カラー調整)する

PWM信号は元となるタイマー(TIM1)をHAL_StatusTypeDef HAL_TIM_Base_Start(TIM_HandleTy peDef *htim)関数で起動します。その後、HAL_StatusTypeDef HAL_TIM_PWM_Start(TIM_HandleTy peDef *htim, uint32_t Channel)関数でPWMを3チャンネル(RGB各カラーの明るさを調節する)起動します。
PWMデューティは各カラーをvoid LEDColor(uint16_t, uint16_t, uint16_t)関数で調整します(100段階)。

4. AD変換とDMA転送

DMA転送開始と転送先変数を登録する関数はHAL_StatusTypeDef HAL_ADC_Start_DMA(ADC_Handl eTypeDef* hadc, uint32_t* pData, uint32_t Length)関数で、pDataにあらかじめ宣言した変数 名を入れます。

次にAD変換がDMA転送完了すると、割り込み関数void HAL_ADC_ConvCpltCallback (ADC_Handle TypeDef *hadc)が自動的に読み出されます。この関数の中でAD値をfloat型の電圧や温度に変換 します。HALライブラリでは割り込み関数はライブラリ内で「__weak」定義⁸していますので、 ユーザーは改めて関数宣言する必要はありません。関数だけ実装してください。

5. MCU温度計算

MCUの温度は下記の式で計算します。V25 やAvg_Slopeはデータシートから代表的な数値を取り 出しています。

Temperature [°C] = { ($V_{25} - V_{TS}$) / Avg_Slope } + 25

- A) $V_{25} = 1.43$ [V], Avg_Slope = 4.3 [mV/°C]
- B) V_{TS} = x * 3.3 / 4096 (xは計測したAD値)

⁸ weak関数属性があると、ユーザーが関数を重複して定義・実装した場合に、関数がオーバーロード され、ユーザー定義関数が優先実行されます。HALライブラリでは割り込み関数は全てweak定義され ています。

プログラム編集

- C) Avg_Slopeの単位はmV/℃なので、単位を合わせるためAvg_Slope=0.0043で計算する
- Raspberry Piからコマンドを送信、ETCBで受信する 6. UART1の受信割り込みで位置命令を受け取ります。割り込みを処理はvoid HAL_UART_RxCpltCall back(UART_HandleTypeDef *huart)の中で実装します。この関数もweak関数属性がついています ので、関数を宣言する必要はありません。関数の実装だけしてください。 UART割り込み処理の実行は、プログラム内でHAL StatusTypeDef HAL UART Receive IT(UART Ha ndleTypeDef *huart, uint8_t *pData, uint16_t Size) を実行しておくと、データ受信待ち状 態となります。データが来ると受信割り込み関数HAL UART RxCpltCallbackが実行されますの で、その関数内で再度HAL_UART_Receive_IT関数を実行し、次の受信待ち状態にすることで、繰 り返し割り込み受信します。pDataに受信バッファ用の配列変数を指定し、Sizeには1を指定しま す。Sizeを例えば3などと指定すると3文字受信するまで割り込みが発生しなくなります。 Raspberry Piからのコマンドは、データ受信間隔が5ms(タイムアウト)程度開いたら、一連の コマンドの区切り(コマンド受信完了)と判断します。HAL_UART_RxCpltCallback関数内でタイ ムアウト変数(ultimeout)を5にセットします。後述のSYSTICKタイマーイベントでultimeout を1ずつ減らし、0になったらコマンド受信完了と判断し、それまで受け取ったデータをParse関 数で解析します。Parse関数では受信したコマンドをID番号と目標位置へ分解して、サーボモー ターのデータ保存用変数(IcsServo構造体)にコピーします。 Raspberry Piから送信する命令は次のような構造にしてください。
 - A) ICS番号は下記の式で決めます。プログラム内ではサーボ構造体変数配列のインデックス番号として使用しています。
 ICS = ID番号×2+Port (Port=0 (S1~S6)、Port=1 (S7~S12))
 - B) 近藤科学製のサーボモーターはポジション(角度)を3500(-135°)~11500(135°)まで指定できます。POS_Lはポジションの下位1バイト、POS_Hは上位1バイトとなります。
 例えば7500(0x1D4C)を指定する場合は、POS_L=0x4C(76)、POS_H=0x1D(29)です。
 - C) SUMはチェックサムです。SUM以外のデータを全て足し合わせた数の下位1バイトのデータ を指定してください。
 SUM=ICS1 + POS1_L + POS1_H + ・・・ + ICSn + POSn_L + POSn_H
 - D) 受け取ったデータはParse関数で内容を確認し、正しいコマンドと判断されたらIcsServo構 造体配列のdpos変数に受け取ったポジションデータを保存します。ポジションデータは下 位バイトを先に指定してください。

バイト数	1	2	3	4	5	6	
内容	ICS1	P0S1_L	POS1_H	ICS2	P0S2_L	POS2_H	
		3n-2	3n-1	3n	3n+1		
		ICSn	P0Sn_L	P0Sn_H	SUM		

プログラム編集

- 20ms周期でサーボにコマンドを送る TIM2を使って20ms周期で割り込みを発生させ、割り込み関数でサーボにコマンドを送ります。 TIM2の周期割り込み(タイマー割り込み)関数はvoid HAL_TIM_PeriodElapsedCallback (TIM_Ha ndleTypeDef *htim)です。割り込み関数内でサーボモーターにポジションコマンドを送ります。 ポジションはIcsServo構造体から読み出します。
- STM32CubeMXのSYSメニューで指定したSYSTICK (36ページ「デバッガ設定」を参照)では、1 ms間隔で割り込みが発生します (void SystemClock_Config(void)内で定義)。割り込み関数はv oid HAL_SYSTICK_Callback ()です。この関数と受信割り込み関数を使って、下記のような手順 でコマンドの区切りを探し、コマンドの実行を行います。
 - A) UART受信割り込みが発生するたびにultimeoutが5にリセットされる
 - B) 受信割り込みが発生しない(コマンドが来ない)と、HAL_SYSTICK_Callback関数でultimeo utが1msあたり1ずつカウントダウンされる
 - C) ultimeoutがOになったら、それまで受信したデータをひとまとめのコマンドとして実行する(Parse関数)
 - D) カウントダウンの最中にUART受信割り込みが発生したら、ultimeoutが再度5にリセットされ、コマンドが続いている物と判断する

SYSTICK割り込みはあまり正確ではないので注意してください。

 今回は設定のみで実際は使用していませんが、UARTからDMA転送でデータを出力するときはHAL _StatusTypeDef HAL_UART_Transmit_DMA(UART_HandleTypeDef *huart, uint8_t *pData, uint16 _t Size)関数を使います。返事をpDataと同じ型の変数に代入し、HAL_UART_Transmit_DMA関数を 実行してください。

ここからは分かりやすくコメントを入れたソースコードを掲載します。

.user.h

user.hはユーザー定義関数の定義や、変数定義などを行っています。詳細はプログラムのコメントを 参照してください。

```
/*

* user.h

*

* Created on: 2016/09/18

* Author: <u>chinoken</u>

*/

#ifndef USER_H_
```

#define USER_H_

```
#include "stm32f3xx_hal.h" // HALライブラリを使うためにインクルードしておくこと
/*
* 共通設定
*/
typedef enum
{
 FALSE = 0, // Bool 変数は定義されていないので、ユーザーが定義する
 TRUE = 1
} Boolean;
/*
* 定義
*/
                                                     受信バッファサイズを
#define UART1_RXBUF_SIZE 32 // 最大受信バッファサイズ
                                                     変更する場合はこの数
#define UART1_RX_TIMEOUT 5 // 受信タイムアウト(5=5ms)
                                                     値を変える
/*
* サーボポート指定(ETCBのS1~S12までのポートを指定するときに使用する)
*/
typedef enum
{
 S1 = 0,
 S2 = 0,
 S3 = 0,
 S4 = 0,
 S5 = 0,
                                        サーボモーターの取付け位置を以下の
 S6 = 0,
                                        式で数値化しておく
 S7 = 1,
                                        ics = id * 2 + port
 S8 = 1,
                                        ics番号をサーボ構造体配列インデッ
 S9 = 1,
                                        クスに使うと、ポートごとに交互にポ
 S10 = 1,
                                        ジション命令が出せるようになる
 S11 = 1,
 S12 = 1
} SI0_Port;
/*
* コマンドエラー
*/
typedef enum
{
 CMD_ERROR_EXCEPTION = -1, // use for HAL_ERROR
 CMD_ERROR_SUCCESS = 0, // succeed to execute command
 CMD_ERROR_TRANSMIT = 1, // fail to transmit data
 CMD_ERROR_RECEIVE = 2, // fail to receive data (enough data length and so on)
 CMD_ERROR_LENGTH = 3, // invalid command length
```



```
CMD_ERROR_INVALID_COMMAND = 4, // invalid command (header)
 CMD ERROR CHECKSUM = 5 // invalid checksum
}
CommandStatus;
/*
* 近藤科学ICSサーボ用のコマンド
*/
                         7500
#define ICS_SERVO_NEUTRAL
#define ICS_SERVO_LOWER_LIMIT 3500
#define ICS_SERVO_UPPER_LIMIT 11500
#define ICS SET POS
                     0x80 // 0b101xxxxx, xはIDが入る
                    8 // サーボの数を最大8個にする(変更時はTIM2タイマー間隔
#define MAX_SERV0_COUNT
を調整すること)
/*
* サーボモーターを管理しやすくするための共用体
* ics番号は配列の添え字で使用し、ics = id * 2 + portの関係になる
*/
typedef union
{
 struct
 {
   char port:1; // SIOポート番号(S1~S12)
            // ID番号
   char id:5;
   char mode:1; // normal=0, rotate=1;
   char enable:1;// サーボ使用bit(使用する=1,使用しない=0)
 } id_bits;
 struct
 {
   char ics:6; // ICS番号:ics = id * 5 + portの計算が成立する
   char b:2;
             // 使用しない
 } ics_bits;
 uint8_t value;
} IcsConfig;
/*
* サーボモーターの位置などを保持するための構造体
*/
typedef struct
{
 IcsConfig config; // サーボID情報
 uint16_t dpos; // ポジション命令を出すときの目標位置
 uint16_t cpos;
                // サーボから取得した現在位置
} IcsServo;
```

```
IcsServo ics_servo[MAX_SERVO_COUNT]; // サーボデータ保存用構造体変数を宣言する
/*
                                                サーボに出力可能な範囲に数
 * 簡単な変数の範囲制限
                                                値を制限する
*/
#define TRIM(a,min,max) ((a)<(min)?(a)=(min):(a)>(max)?(a)=(max):(a))
/*
* AD変換
*/
uint16_t adc_raw_data[5]; // AD1~4と温度データを保存する
float adc_value[5]; // AD返還後のデータを保存する
/*
* UART1受信割り込みバッファ
*/
uint8_t u1buff[1]; // 1文字受信用のバッファ
                                         u1buffは1文字受信割り込み
uint8_t u1cmd[UART1_RXBUF_SIZE];
                                         用バッファに使う。受信した
uint8_t u1index;
                                         データはu1cmdに保存し、u
uint16_t u1timeout;
                                         1indexで受信したデータ数を
                                         数えておく
/*
* 関数定義(オリジナル関数のみ)
*/
void LEDColor(uint16_t, uint16_t, uint16_t);
HAL_StatusTypeDef IcsSetPos (IcsServo *);
CommandStatus Parse();
```

```
#end i f /* USER_H_ */
```

.user.c

user.cではuser.hで定義したユーザー関数の実装と割り込み関数を実装しています。

```
/*
* user.c
*
* Created on: 2016/09/18
* Author: chinoken
*/
#include "user.h" // 定義や宣言を読み込むため、user.hをインクルードする
/*
* main.cで定義されている変数をuser.cで使うためextern宣言する
*/
```

```
extern UART_HandleTypeDef huart1;
extern UART HandleTypeDef huart2;
extern UART_HandleTypeDef huart3;
extern TIM_HandleTypeDef htim1;
/*
 * PWM出力(LEDのカラー)を変更
 */
void LEDColor (uint16_t r, uint16_t g, uint16_t b)
{
 htim1.Instance->CCR1 = (uint32_t)TRIM(r,0,100); // 最大値は100=10ms
 htim1.Instance->CCR2 = (uint32_t)TRIM(g,0,100);
 htim1.Instance->CCR3 = (uint32_t)TRIM(b,0,100);
}
/*
 * ICSサーボにポジションコマンドを送る
*/
HAL_StatusTypeDef IcsSetPos (IcsServo *servo)
{
 HAL_StatusTypeDef hStat; // HAL Library関数返値の受け取り
 uint8_t cmd[3], rx[3];
 uint16_t pos = 0;
 UART_HandleTypeDef *huart;
                                                    近藤科学ICSサーボコマ
                                                    ンドに準拠した処理
 cmd[0] = ICS_SET_POS | servo->config.id_bits.id; 
 cmd[1] = (servo->dpos >> 7) & 0x7F; // 上位7bit
 cmd[2] = (servo->dpos) & 0x7F; // 下位7bit
 // ポートによってUSARTを切り替える
 huart = (servo->config.id_bits.port == 0 ? &huart3 : &huart2);
 // コマンドを送信する(受信タイムアウトは2ms)
 // 半二重通信なので、送信と受信の切り替えが必要です
 HAL_HalfDuplex_EnableTransmitter(huart); // 送信を有効にする
 hStat = HAL_UART_Transmit(huart, cmd, 3, 2); // cmd変数3byteを2msのタイムアウトで送信
 if (hStat == HAL OK)
 {
   // 正常に送信できたのでサーボからの返事を受信する
   HAL HalfDuplex EnableReceiver(huart); // 受信を有効にする
   hStat = HAL_UART_Receive(huart, rx, 3, 2); // rx変数に3byteを2msのタイムアウトで受信
   if (hStat == HAL_OK) // 受信成功の場合
   {
     if ((rx[0] & 0x3F) == servo->config.id_bits.id) // 受信データのidを確認
     {
       pos = (rx[1] << 7); // 上位7bit
```

```
pos += (rx[2] & 0x7F); // 下位7bit分を追加
       servo->cpos = TRIM(pos, 3500, 11500); // 返ってきた値を範囲制限して保存する
     }
   }
   else
   {
     return HAL_TIMEOUT;
   }
 }
 else
 {
   return HAL_TIMEOUT;
 }
 return HAL_OK;
}
/*
* Rasberry Piから受信したコマンドを解析する
*/
CommandStatus Parse()
{
 /*
  * コマンドの内容(各1バイト)
  * ICS1 POS1_L POS1_H ICS2 POS2_L POS2_H ,..., ICSn POSn_L POSn_H SUM
  * ICS = ID x2 + Port (Port = 0(S1~S6), Port = 1(S7~S12))
  * SUMはそれ以外のデータの総和の下位1バイト
  */
 if (ulindex % 3 != 1) // コマンドの数は3の倍数+1
 {
   return CMD_ERROR_LENGTH;
 }
 // チェックサム
 uint8_t sum = 0;
 uint16_t i;
 for (i = 0; i < u1index - 1; i++)</pre>
 {
   sum += u1cmd[i];
 }
 if (sum != u1cmd[u1index - 1])
 {
   return CMD_ERROR_CHECKSUM;
 }
```

```
// 解析
 uint16_t pos;
 uint8_t ics;
 for (i = 0; i < u1index - 1; i+=3) // 0~23
 {
   ics = u1cmd[i] & 0x3F; // 下位6bit
   pos = ((uint16_t)u1cmd[i + 2]) << 8; // 上位8bit
   pos += u1cmd[i + 1]; // 下位8bit
   pos = TRIM(pos,ICS_SERV0_LOWER_LIMIT,ICS_SERV0_UPPER_LIMIT);
   ics_servo[ics].dpos = pos; // dposに受け取った目標位置を保存する
 }
                                               TIM2割り込み関数でdpos変数を
 return CMD_ERROR_SUCCESS;
                                               使ってポジションコマンドを実
}
                                               行する
* 以下は割り込みの処理
*/
/*
* HAL_ADC_ConvCpltCallback関数はDMA転送完了後に自動的に実行される
*/
void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef *hadc)
{
                                                         割り込み発生周期
 // AD値を元の単位へ変換する
                                                         はAD変換周期と
 adc_value[0] = ((float)adc_raw_data[0])*3.3/4096.0;
                                                         ほぼ同じ
 adc_value[1] = ((float)adc_raw_data[1])*3.3/4096.0;
 adc_value[2] = ((float)adc_raw_data[2])*3.3/4096.0;
 adc_value[3] = ((float)adc_raw_data[3])*3.3/4096.0;
 // 温度計算
 float Vts = ((float)adc_raw_data[4])*3.3/4096.0; // AD値を電圧に変換
 adc_value[4] = ((1.43 - Vts) / 0.0043) + 25; // 温度
}
* タイマー割り込み
*/
void HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef *htim)
{
 if (htim->Instance == TIM2) // 割り込みの種類がTIM2だったとき
 {
   HAL_StatusTypeDef hStat;
   uint16_t i, j;
                                                   タイマー割り込みは割り込み
                                                   の発生元を確認すること
                                                   割り込みタイマーが複数の場
                                                   合はifで分岐する
```

```
プログラム編集
```

```
// サーボに命令を送る
   for (i = 0; i < MAX SERVO COUNT; i++)</pre>
   {
     if (ics_servo[i].config.id_bits.enable == TRUE)
     {
      // サーボにポジションコマンドを送信する
      hStat = IcsSetPos(&ics_servo[i]);
                                                ics_servo配列のインデックス
                                                はics番号なので、インクリメ
      if (hStat != HAL_OK)
                                                ントするだけでIcsSetPos命令
      {
                                                をUART2,3ポートに交互に出
        // エラー処理をする場合
                                                せる
      }
    }
     // ちょっと間を空ける
     for (j = 0; j < 100; j++) {}
   }
 }
}
                                                UARTも割り込み発生元を確
                                                認すること
 * UART1受信完了割り込み
 */
void HAL_UART_RxCpltCallback(UART HandleTypeDef *huart)
{
  if (huart->Instance == USART1) // 割り込み条件がUART1だった場合
  {
   u1cmd[u1index] = u1buff[0]; // 受信した1文字を配列に追加
   HAL_UART_Receive_IT(&huart1, u1buff, 1); // 1文字受信待ちを再開
   ulindex ++; // カウンタ(配列のインデックス)を進める
   if (u1index > UART1_RXBUF_SIZE)
   {
    u1index = 0; // 受信バッファが変数の最大サイズを超えたらインデックスを最初に戻す
   }
   ultimeout = UART1_RX_TIMEOUT; // 受/
 }
                                 SYSTICK割り込み周期はmain.cのSystemClock_Config関
}
                                 数で
                                 HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);
                                 と定義している。クロックカウント=1秒なので、100
 * SYSTICK割り込み(システムクロックに
                                 0で割ることで1msの割り込みを設定している。1000
 */
                                 を10000とすると0.1msの割り込みになる
void HAL_SYSTICK_Callback ()
{
 // 受信タイムアウトが発生したらそれまで受け取った命令を解析する
```

```
if (ultimeout != 0)
{
    if (--ultimeout == 0) // 受信タイムアウトイベントが発生
    {
        CommandStatus c_stat = Parse(); //コマンド(受信データ)を実行
        if (c_stat != CMD_ERROR_SUCCESS)
        {
            // エラーがあった場合の処理
        }
        ulindex = 0; // インデックスをリセット
    }
}
```

.main.c

本例題では、位置命令の受信やサーボへの命令の送受信、AD変換など全て処理は割り込み関数の中で 行っているので、main.cでは特にプログラム処理命令はありません。変数初期化やETCBの登録のみ行 っています。







void HAL_TIM_MspPostInit(TIM_HandleTypeDef *htim);





```
プログラム編集
```

```
/* Initialize all configured peripherals */
 MX_GPI0_Init();
 MX_DMA_Init();
                                            自動生成されたGPIO、ADなど
 MX_ADC1_Init();
                                            諸設定関数のプロトタイプ宣言
 MX_TIM1_Init();
                                            中身はmain関数の後に記載し
 MX_TIM2_Init();
                                            ている
 MX_USART1_UART_Init();
 MX_USART2_UART_Init();
 MX_USART3_UART_Init();
 /* USER CODE BEGIN 2 */
 /*
  * PWM出力(LED点灯)
  */
 HAL_TIM_Base_Start(&htim1); // TIM1を開始
 HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1); // PWMタイマーを開始
 HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
 HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_3);
 LEDColor(100,0,0); // 赤を点灯
 /*
  * AD変換を開始する
  */
 HAL_ADC_Start_DMA(&hadc1, (uint32_t *)adc_raw_data, 5); // AD返還後にデータを6個DMA転送す
る
 /*
  * UART受信割り込み
  */
 // 受信文字数カウンタをリセット
 ulindex = 0;
 // UARTの受信割り込みは1文字受信待ちを実行すると開始となる
 HAL_UART_Receive_IT(&huart1, (uint8_t *)u1buff, 1);
 // 受信タイムアウト時間をリセット
 ultimeout = 0;
 /*
  * サーボモーターのデータを初期化する
                                                  サーボのenableビットをいっ
  */
                                                  たん0にする
 for (i = 0; i < MAX_SERVO_COUNT; i++)</pre>
 {
   ics_servo[i].config.value = 0; // Port=0,ID=0,Enable=FALSE(0)
   ics_servo[i].dpos = ICS_SERVO_NEUTRAL;
   ics_servo[i].cpos = ICS_SERVO_NEUTRAL;
 }
  * ここから使用するサーボだけID(例として4個分)を登録する
```

プログラム編集

```
* S1~S6ポートにID=0,1のサーボを2個、S7~S12ポートにID=0,1のサーボを2個、合計4個登録する
* index[0]: id=0,port=S1
* index[1]: id=0,port=S7
* index[2]: id=1,port=S2
* index[3]: id=1,port=S8
*/
ics_servo[0].config.id_bits.port = S1; // S1ポートにID=0のサーボを取り付ける
ics_servo[0].config.id_bits.id = 0;
ics_servo[0].config.id_bits.enable = TRUE; // サーボを有効にする
ics_servo[1].config.id_bits.port = S7; // S7ポートにID=0のサーボを取り付ける
ics_servo[1].config.id_bits.id = 0;
ics_servo[1].config.id_bits.enable = TRUE; // サーボを有効にする
ics_servo[2].config.id_bits.port = S2; // S2ポートにID=1のサーボを取り付ける
ics_servo[2].config.id_bits.id = 1;
ics servo[2].config.id bits.enable = TRUE; // サーボを有効にする
ics_servo[3].config.id_bits.port = S8; // S8ポートにID=1のサーボを取り付ける
ics_servo[3].config.id_bits.id = 1;
ics_servo[3].config.id_bits.enable = TRUE; // サーボを有効にする
/*
* TIM2タイマー割り込み開始
*/
HAL_TIM_Base_Start_IT(&htim2);
/*
* 準備完了したのでLEDを緑にする
*/
LEDColor(0,100,0); // 緑を点灯
/* USER CODE END 2 */
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
/* USER CODE END WHILE */
/* USER CODE BEGIN 3 */
 HAL_Delay(50);
                                 これ以降はSTM32CubeMXが自動
                                 生成した関数(MX_)なので省略
                           一支了
 LEDColor (0,i,0); // 適当レ
 i = (i + 1) % 100;
```

プログラム編集

```
}
/* USER CODE END 3 */
}
```

コマンドを送信する

53ページの「5(エ)」で説明した命令を、USB1ポートから送る手順について説明します。

- 1. CoolTermを起動します。
- ツールバーのOptionsボタンを押すか、メインメニューからConnection>Optionsを選択して、C onnection Optionダイアログを表示します。
- 3. Connection Optionダイアログで、下記のように設定します。
 - (ア) Port: ETCBのUSB仮想COMポートを指定します
 - (イ) Baudrate: 115200
 - (ウ) Data Bits: 8
 - (⊥) Parity: none
 - (才) Stop Bits: 1
 - (カ) Flow Controlは全てチェックオフ
 - (キ) OKボタンを押して閉じます。

Cor	nection Options	(CoolTerm_0)	
Serial Port Terminal Receive Transmit Miscellaneous	Serial Port Optio Port: Baudrate: Data Bits: Parity: Stop Bits: Flow Control:	COM3 115200 8 none 1 CTS DTR XON	 . .<
	Initial Line State	es when Port opens: DTR Off RTS Off Re-Scan Serial Ports	0K

プログラム編集

- ツールバーのConnectボタンを押すか、メインメニューからConnection>Connectを選択して、 仮想COMポートを開きます。
- 5. ツールバーのView Hexボタンを押すか、メインメニューからView>View Hexを選択して、画面 を16進数表示に変更します。
- 6. メインメニューからConnection>Send Stringを選択して、データ入力用のSend Stringダイアロ グを開きます。
- サーボモーターへの命令を入力します。下記の(ア)はプログラムで登録した全てのサーボモータ ー(ID=0,Port=S1/ID=1,Port=S2/ID=0,Port=S6/ID=1,Port=S7)がニュートラル位置(7500=0x1 D4C)へ移動するコマンドです。(イ)は全てのサーボが6000(=0x1770)、(ウ)は全てのサーボが 9000(=0x2328)に移動するコマンドになります。リトルエンディアンで受け取るので、数値は 下位のバイトを先にします。青い文字はICS番号で赤い文字はCHECKSUMになります。 下図は(ア)を入力した、SendString画面です。SendString画面で文字列の最後を改行してしまう と、改行コードも送信されてETCBでエラーが出るので、改行を入れないでください。
 - (ア) 00 4C 1D 01 4C 1D 02 4C 1D 03 4C 1D AA

 - (ウ) 00 28 23 01 28 23 02 28 23 03 28 23 32

*	Send String (CoolTerm_0)	
Send Str	ng	
	II	Send
00 40	1D 01 4C 1D 02 4C 1D 03 4C 1D AA	

- 8. 上記の(ア)~(ウ)を入力してから、「Send」ボタンを押してコマンドを送信します。正しくサー ボモーターが動作するか確認します。
- 9. 数値を変えてみて動作検証もしてみてください。もしうまく動かない場合は(1)ポートを開いていない、(2)HEXで入力していない、(3)CHECKSUMの計算が間違っているなどの問題がないか再確認してください。

