robbit
開発マニュアル

吉瀬研究室
制作日：2025年9月

robbit : 概要

- robbitはFPGAを活用する扱いやすいtwo-wheeled self-balancing robotである。

- 開発手順

  robbitの組み立て

  ↓

  Bitstream, バイナリ作成

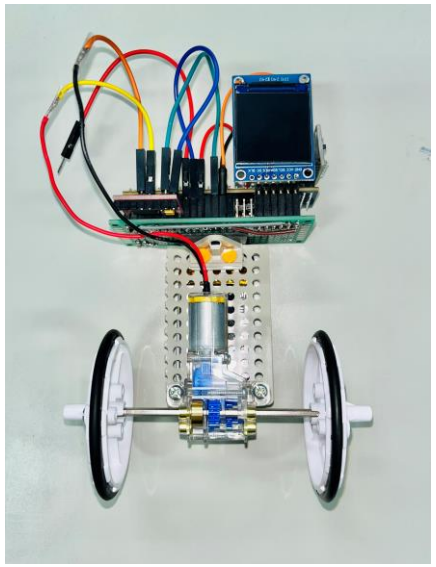
  ↓

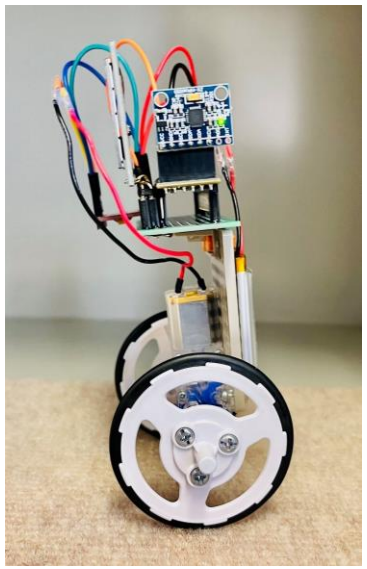
  動作確認

  ↓

  パラメータチューニング

robbit

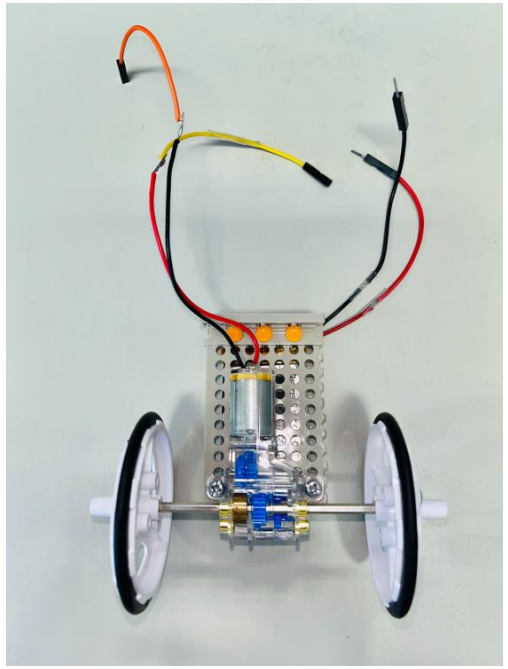


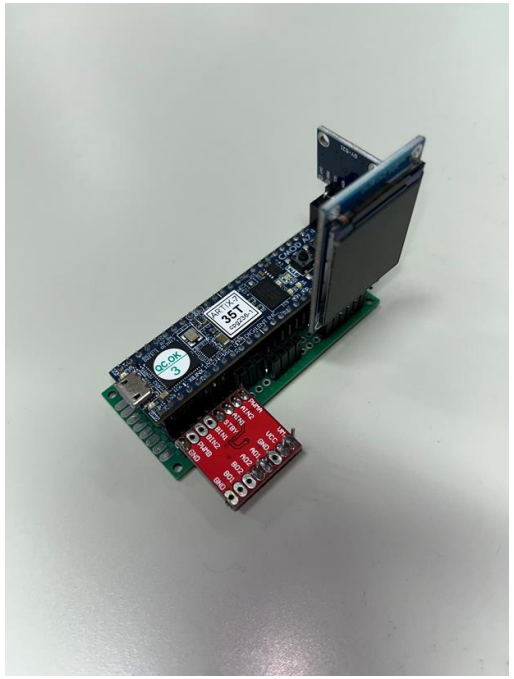


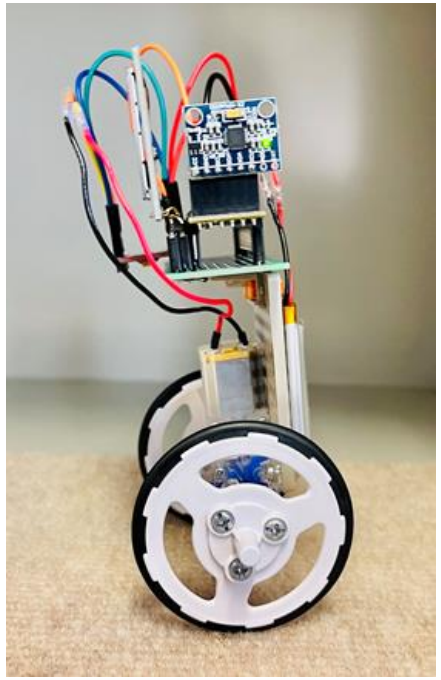
robbitの組み立て：部品購入

■ Robbit-espの組み立ては以下の手順で行う

部品購入 ➡ シャーシ作成 ➡ 制御ユニット作成 ➡ 接着







robbitの組み立て：部品購入

Institute of SCIENCE TOKYO

■ 下記の部品一覧にある部品を購入する

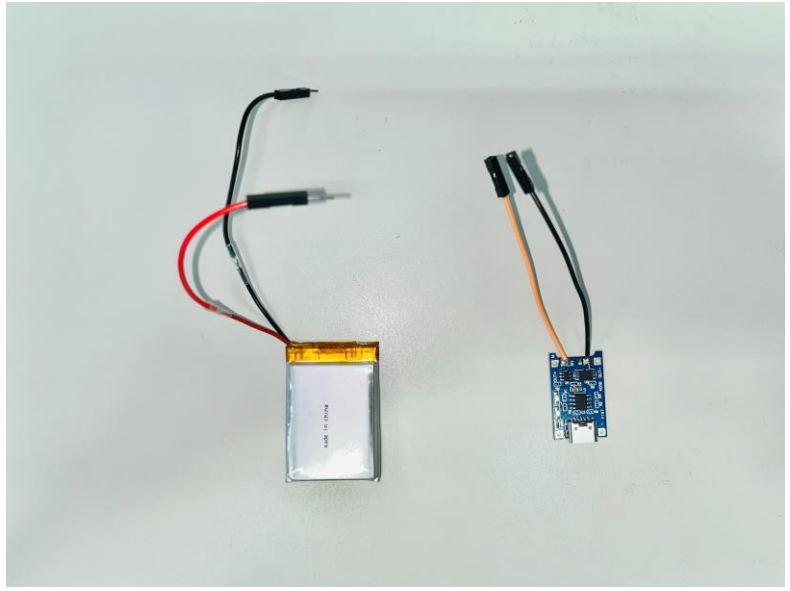
部品	名称	個数
FPGAボード	Cmod A7-35T	1
センサ	MPU-6050	1
モータ	Mini Motor Standard Gearbox 70188	1
タイヤ	Slim Tire Set (55mm Dia.) 70193	1
モータドライバ	TB6612FNG	1
バッテリー	EEMB Lithium-Ion Battery 653042	1
プレート	Universal Plate Set 70157	1

robbitの組み立て：部品購入

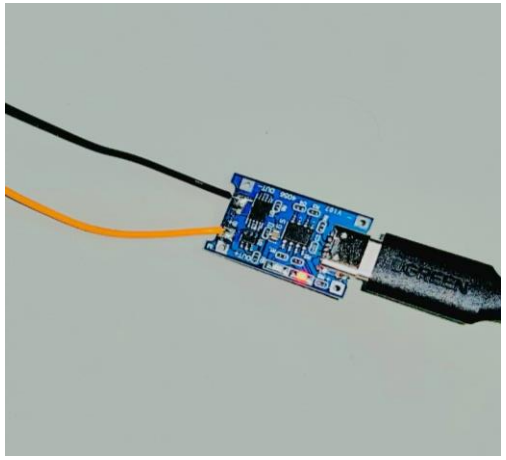
■ 下記の部品があるかを確認する

robbitの組み立て：シャーシ作成

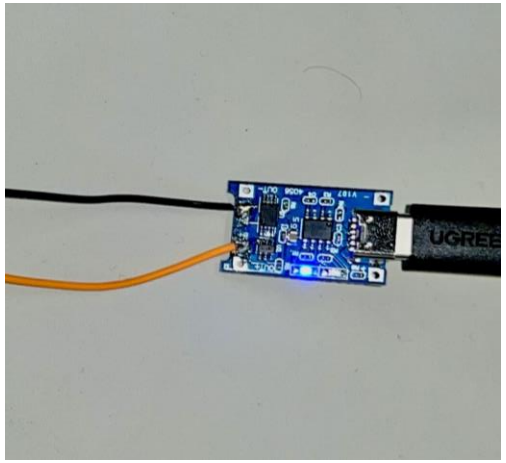
- バッテリーと電源モジュールを図のようにはんだ付けする
- はんだ付け後は、バッテリーを充電モジュールに接続する
- 充電モジュールの色でバッテリーの充電具合がわかる
  （赤: 充電不足, 青: 充電完了）



はんだ付けの例



赤: 充電不足



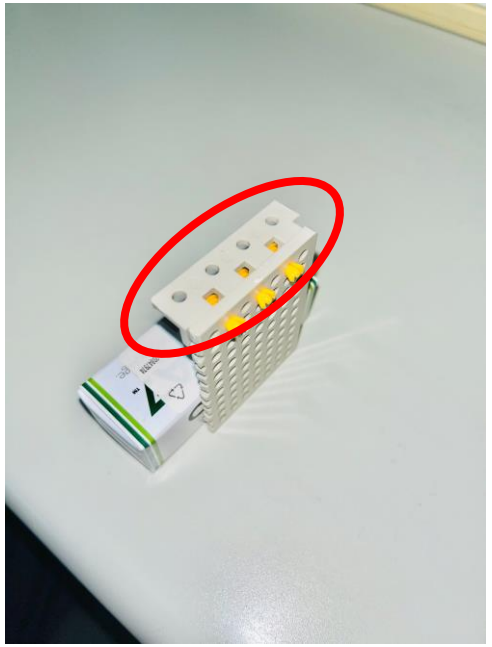
青: 充電完了

充電モジュールのLED点灯

robbitの組み立て：シャーシ作成

- ユニバーサルプレートをはんだ付けする
- ユニバーサルプレートはこの後作る制御ユニットの大きさに合わせるとよい
- またユニバーサルプレートには下図のように、制御ユニットを接着する部分を用意しておく

切断後のユニバーサルプレート



制御ユニットとの接続部分

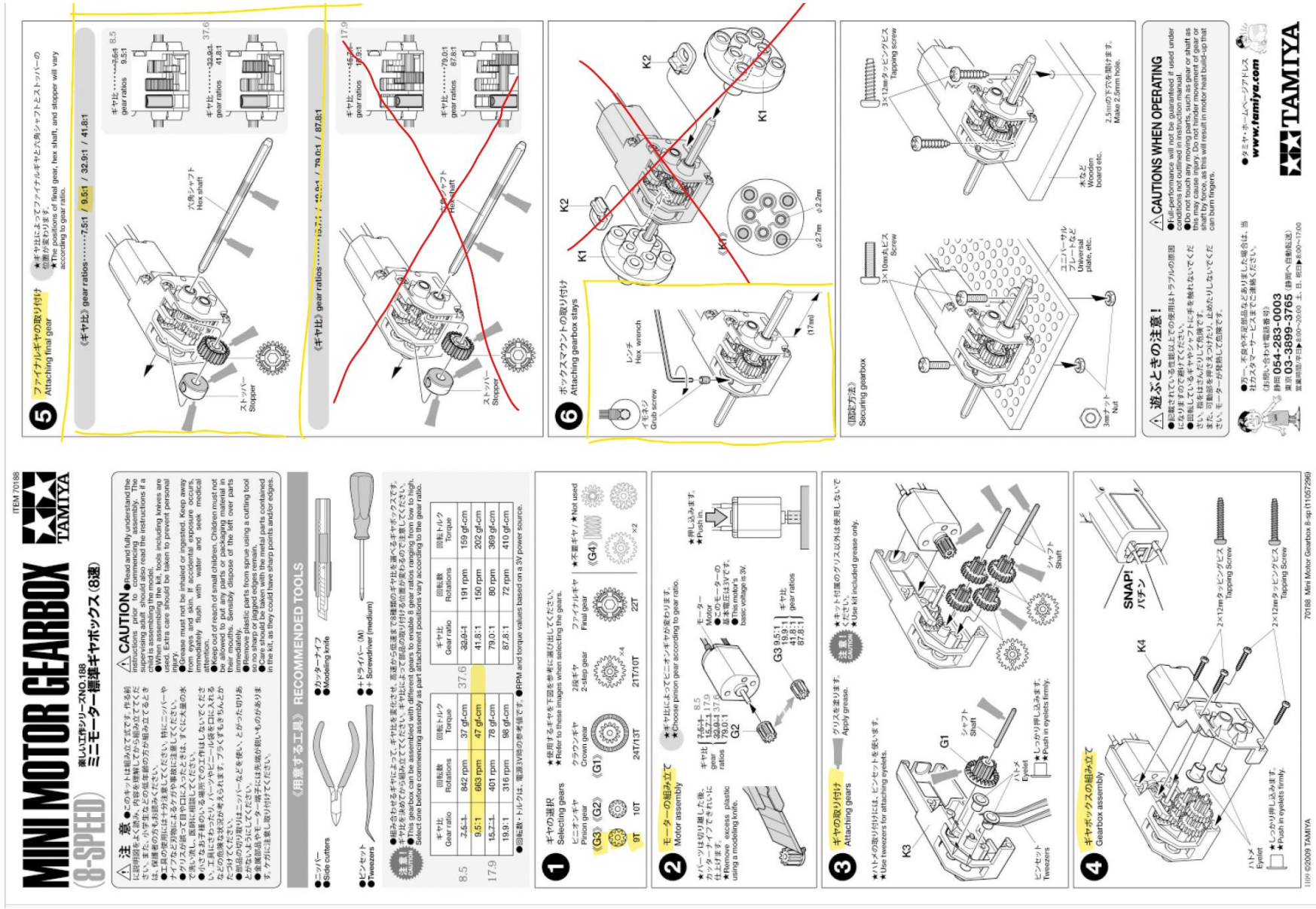
robbitの組み立て：シャーシ作成

- TAMIYAのMini Motor Standard Gearbox 70188とSlim Tire Set (55mm Dia.) 70193を作成する。

- 次ページにギアボックス組み立てマニュアルを載せている。必要な部品や作業には黄色で色付けしている

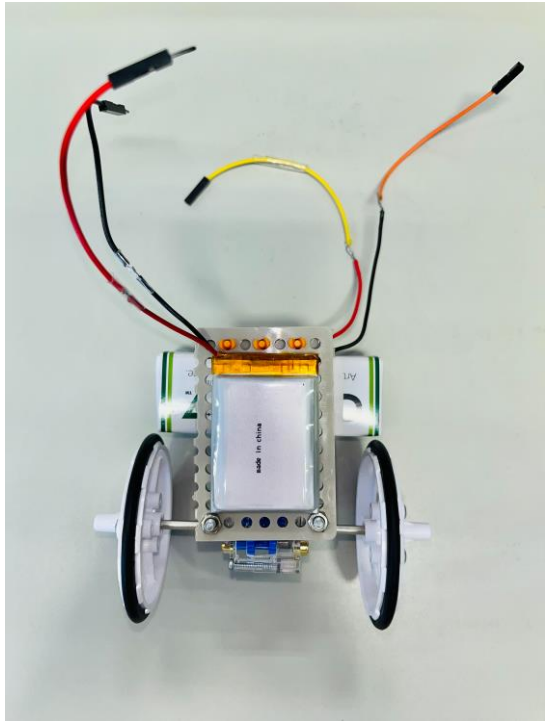
- タイヤに関しては、最初は直径が55mmのタイヤを使用すると良い。
  直径が小さいタイヤも同封されているが、制御が難しくなる場合がある。

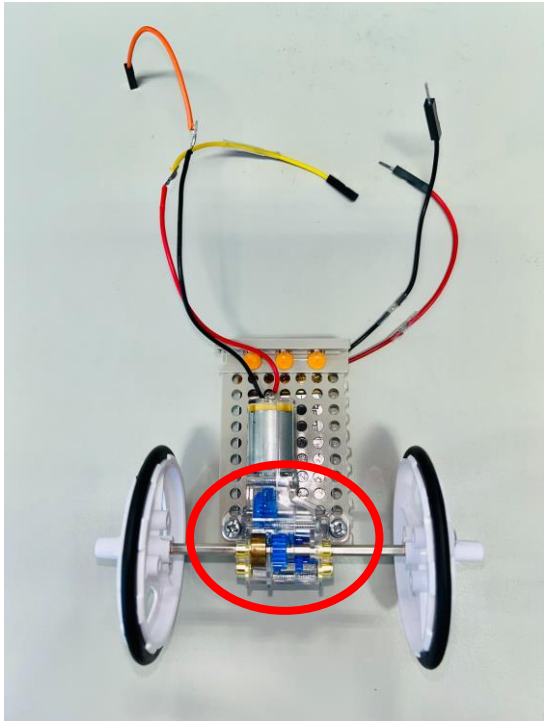
robbitの組み立て：シャーシ作成



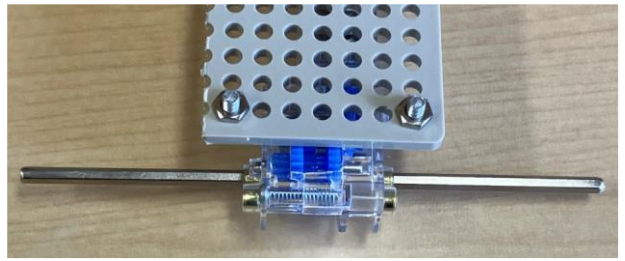
robbitの組み立て：シャーシ作成

- モータが完成したらタイヤを取り付ける
- 最後にユニバーサルプレートにバッテリーとモータを取り付けるとシャーシが完成する。





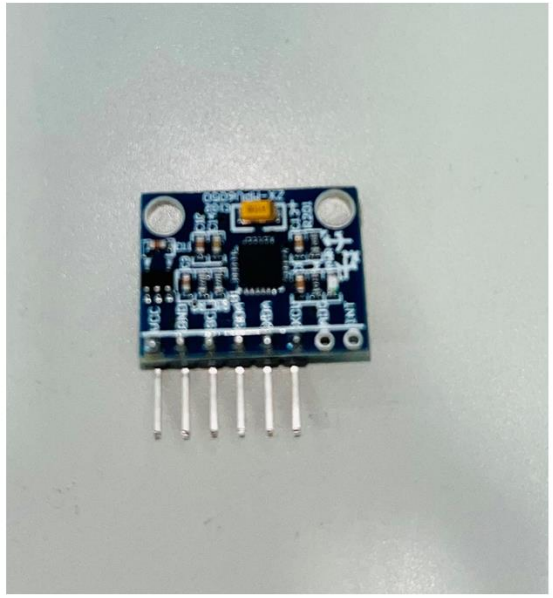
ネジでしっかり固定

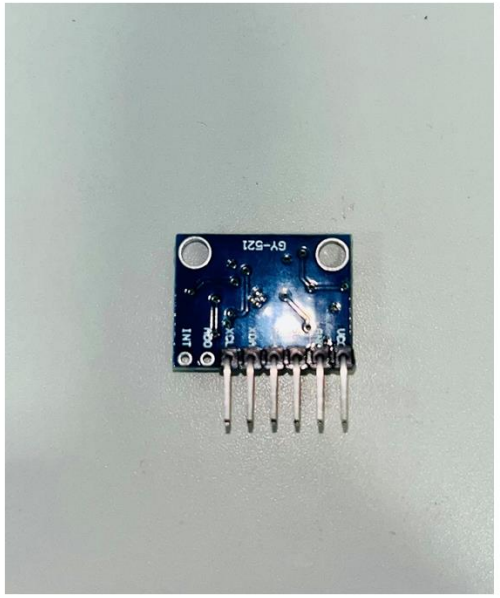


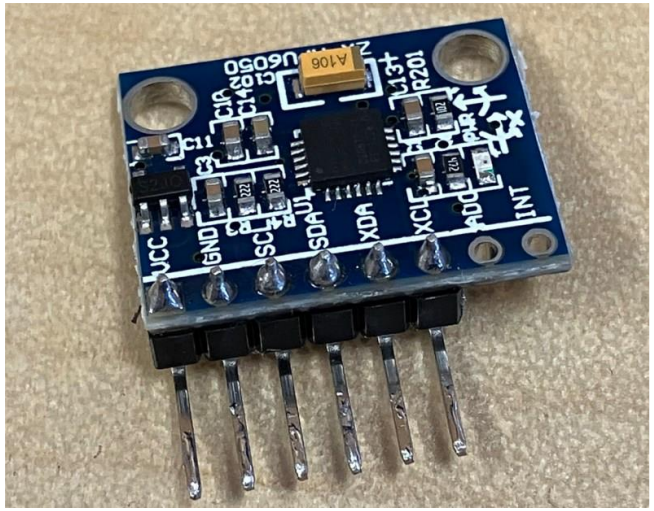
シャーシの完成例

robbitの組み立て：制御ユニット作成

■ センサ(MPU-6050)にピンヘッダをはんだ付けする。 IMUモジュールはVCC, GND, SCL, SDAをつなげば動作するが，安定性向上のため接続部分にはすべてヘッダピンをはんだ付けすることを勧める。



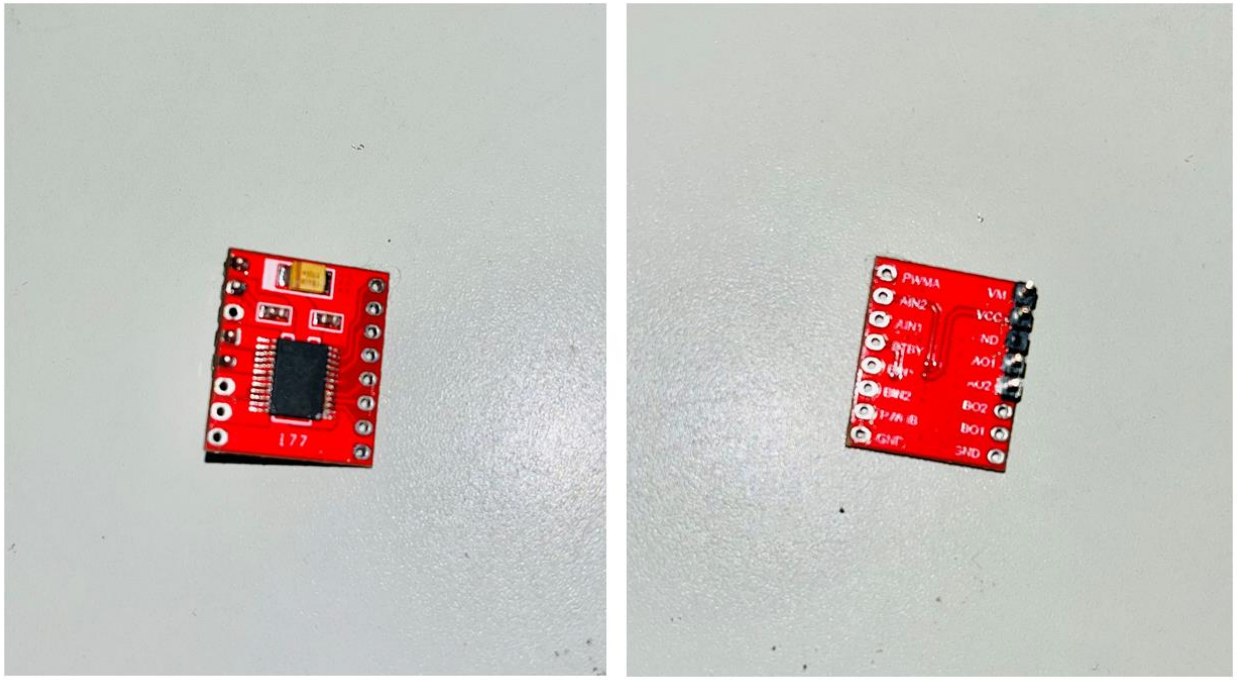




はんだ付けの例

robbitの組み立て：制御ユニット作成

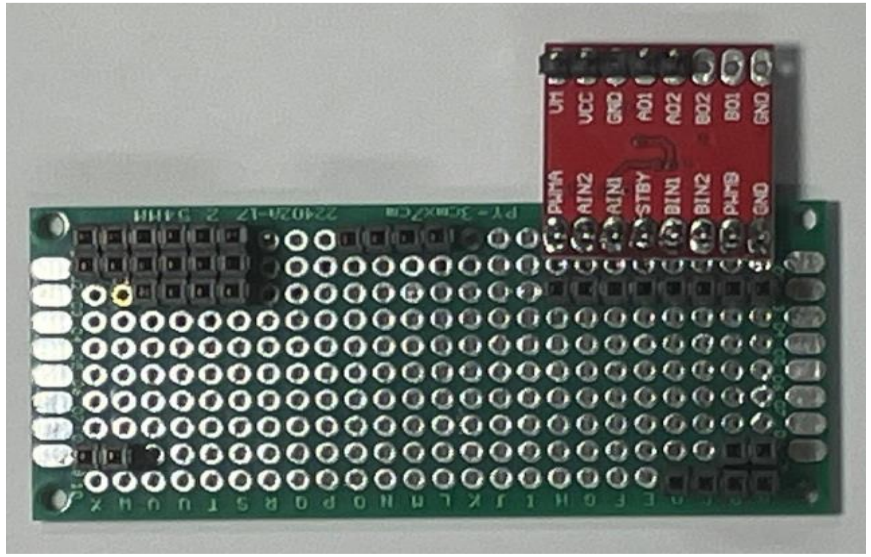
- モータドライバ(TB6612FNG)のVM, VCC, GND, AO1, AO2にはんだ付けをする
- ただし，GNDのピンは使用しないので取り除く

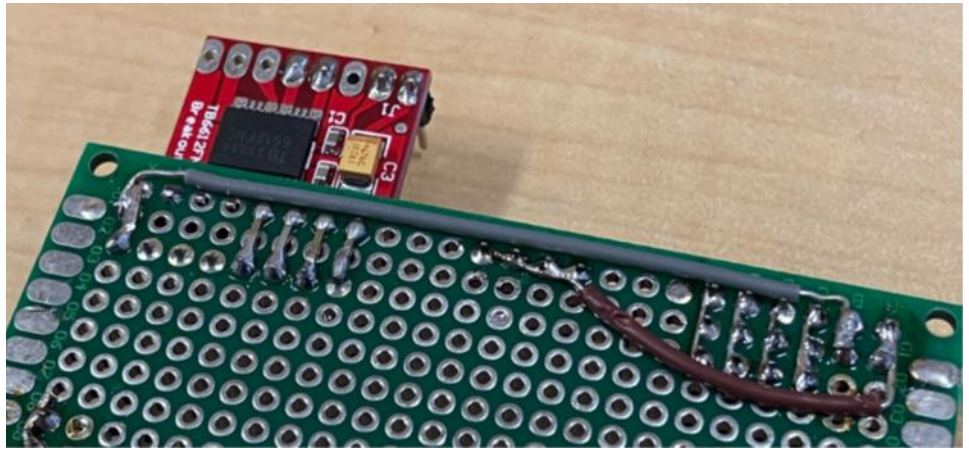


はんだ付けの例

robbitの組み立て：制御ユニット作成

- ピンコネクタを適切にカットして、ユニバーサル基板にはんだ付けする。
- ユニバーサル 基板に、写真の黒色のコネクタを差し込んで、裏面からはんだ付けする。
- モータドライバも基盤にはんだ付けする

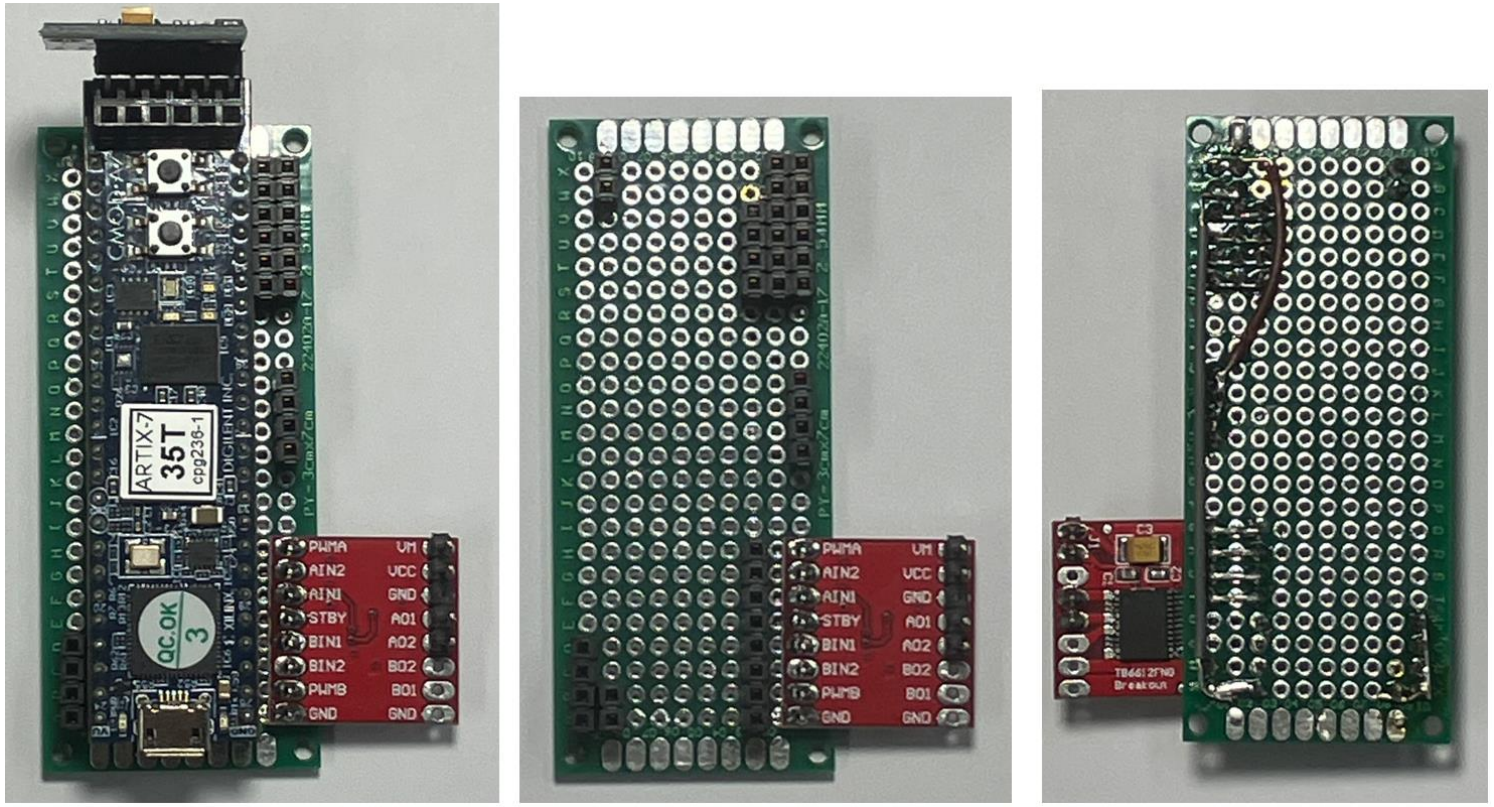




はんだ付けの例

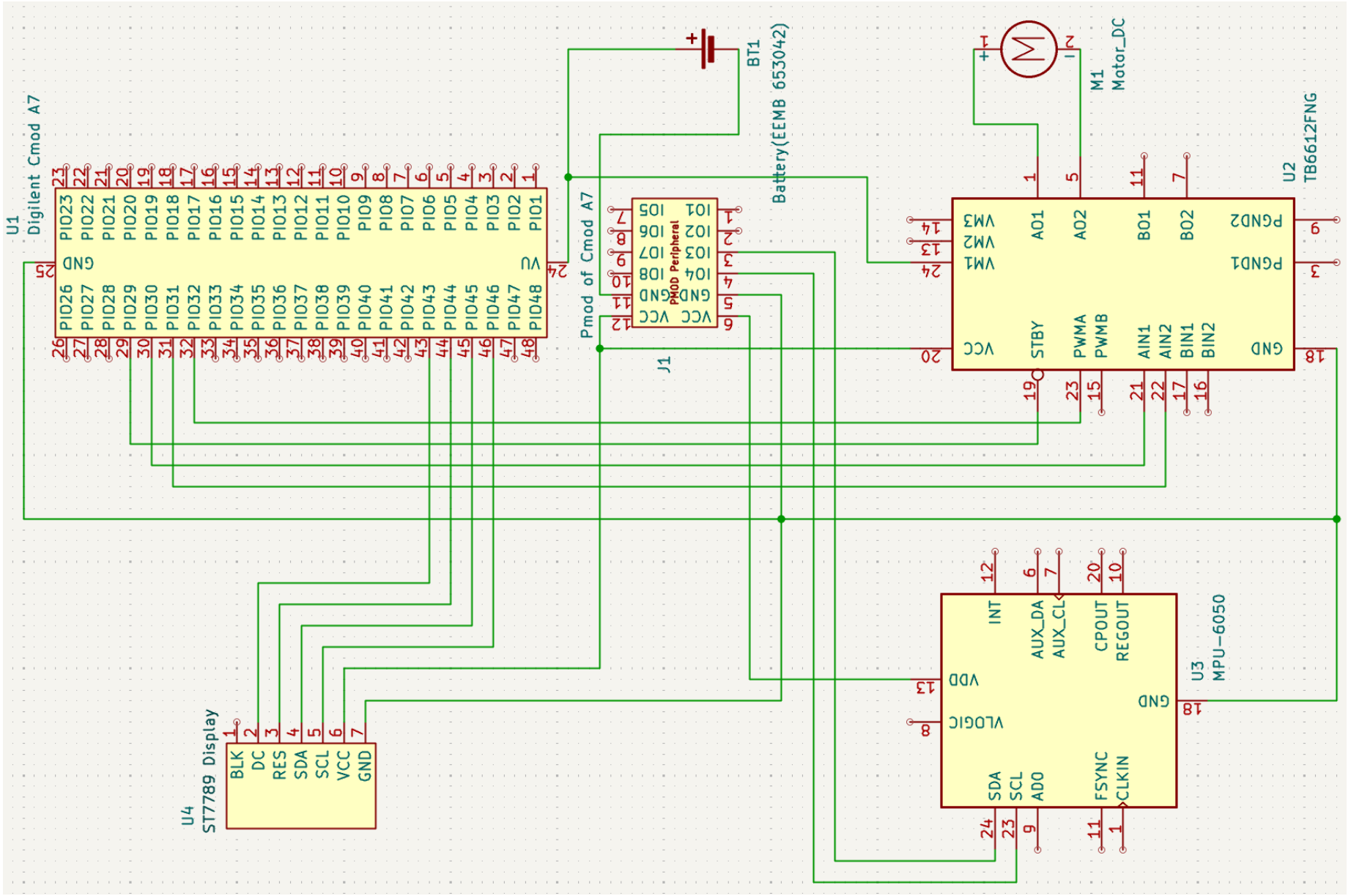
robbitの組み立て：制御ユニット作成

- ブレットボードまたは、ユニバーサル基盤を用意する。
- 用意したブレットボードまたは基盤に回路図や下記の画像を参考に配線、はんだ付けをする。（回路図は次ページに載せている）



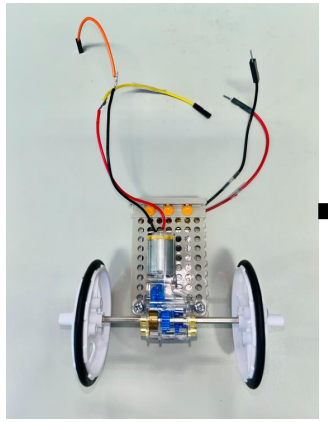
制御ユニットはんだ付け例

robbitの組み立て：制御ユニット作成



robbitの組み立て：制御ユニット作成

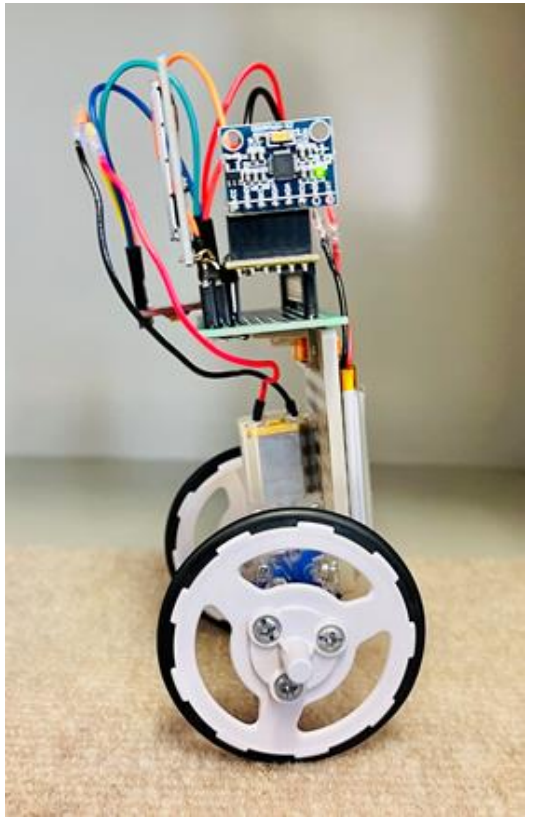
■ 完成したシーシャと制御ユニットを両面テープ等で接着し、制御ユニットの配線を行うと，robbitが完成する。



シャーシ

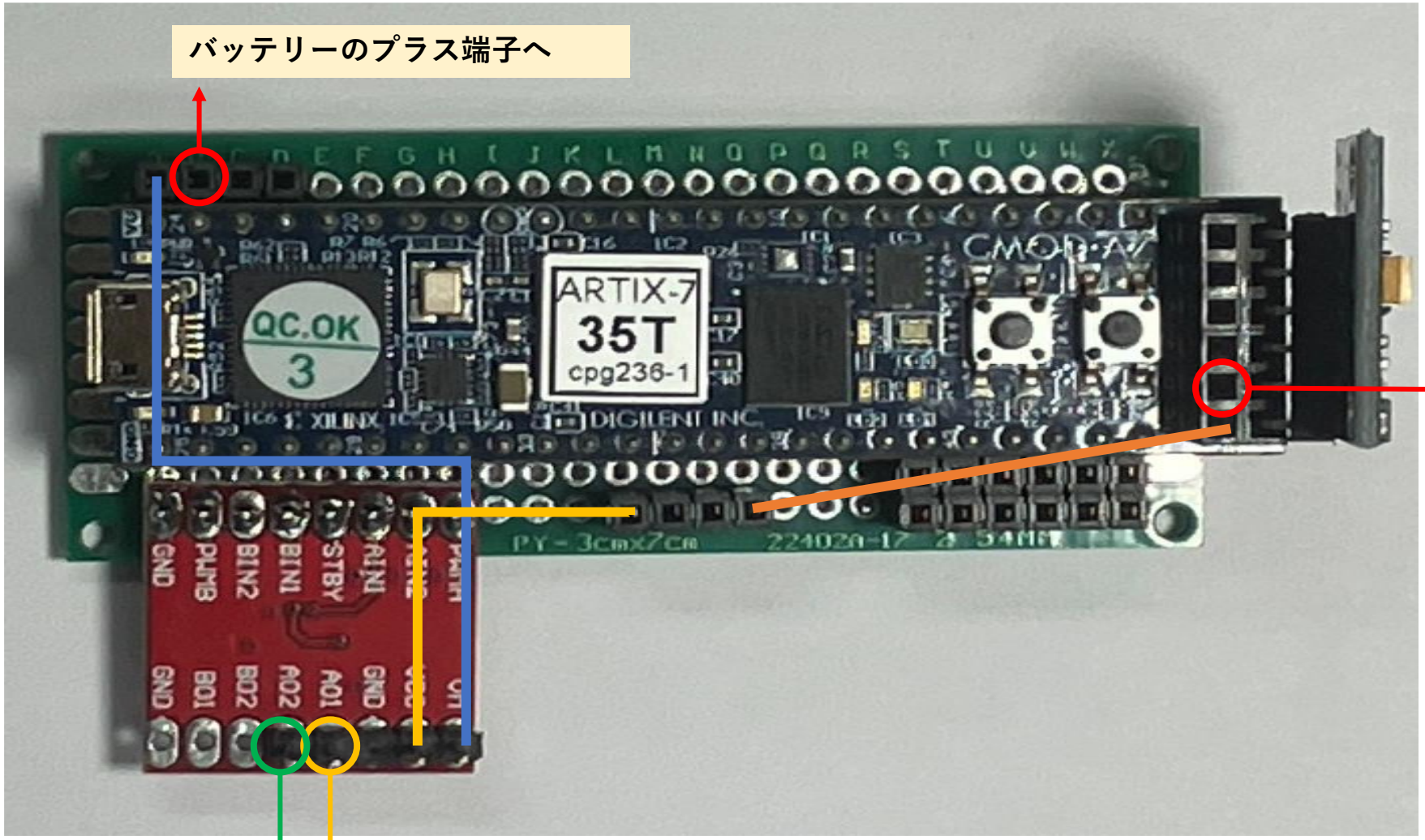
制御ユニット

接着



robbit

robbitの組み立て：制御ユニット作成



制御ユニットの配線例

Bitstream, バイナリ作成: 環境構築

Institute of SCIENCE TOKYO

- 下記のコマンドでrobbitのリポジトリをユーザのローカル環境にクローンする

```
git clone git@github.com:archlab-sciencetokyo/robbit_project.git
```

- CFU Proving Groundがサブモジュールとして存在しているので，
  robbit_projectフォルダに移動し，以下のコマンドで初期化と更新を行う

```
git submodule update --init --recursive
```

Bitstream, バイナリ作成: 環境構築

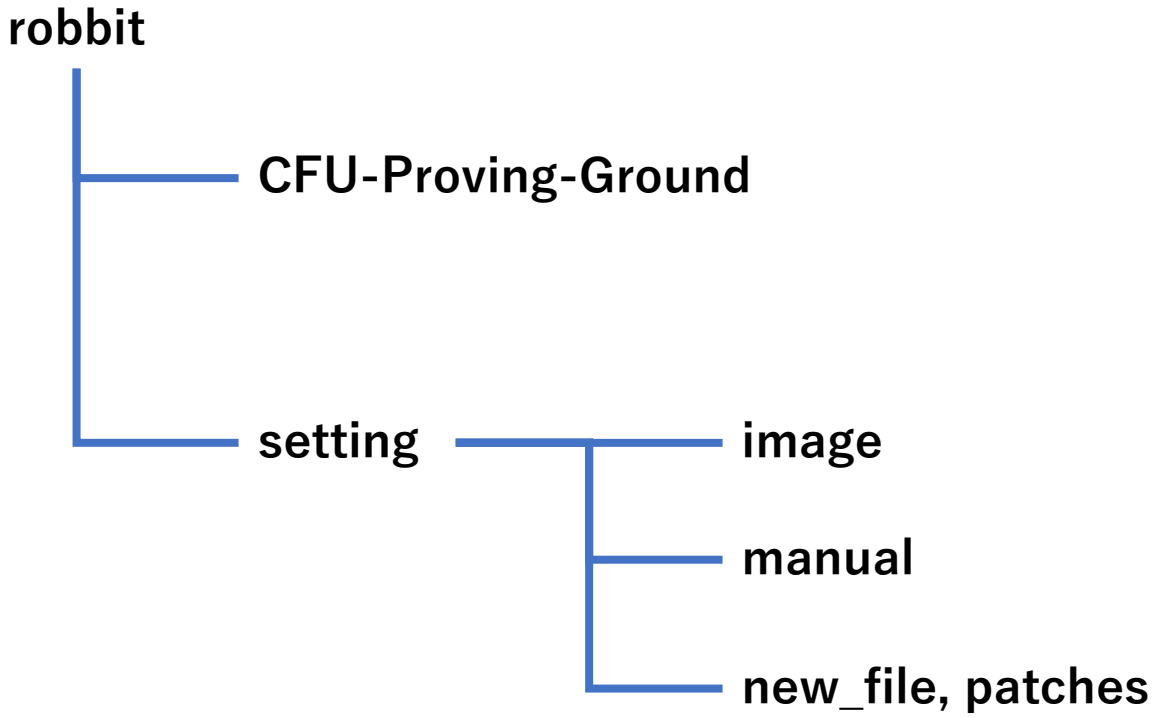
サブモジュール初期化(git submodule update --init --recursive)が失敗する場合は以下のコマンドを試してほしい

- robbit_projectのリポジトリが参照しているCFU Proving Groundのコミットが存在しない場合

  cd robbit_project/robbit/CFU-Proving-Ground (サブモジュールのディレクトリに移動)
  git fetch
  cd ../../ (robbit_projectフォルダに移行)
  git submodule update --init --recursive (再度実行)

Bitstream, バイナリ作成: 環境構築

- CFU-Proving-Groundの初期化が完了するとディレクトリが以下の構成になる
- Settingフォルダ以下のファイルは基本的には編集する必要はない
- まずはrobbitディレクトリで環境を構築していく



Bitstream, バイナリ作成: 環境構築

Institute of SCIENCE TOKYO

- 次にrobbitのゲートウェアとソフトウェアのプログラムをCFU Proving Groundで動かせるように以下のコマンドを入力する

```
make init
```

- コマンドを入力するとCFU-Proving-Groundフォルダにmain.cppやVerilog HDLのファイルがコピーされる
- 以降はCFU-Proving-Groundフォルダで作業を行う

Bitstream, バイナリ作成: 環境構築

Institute of SCIENCE TOKYO

ここまでの流れをターミナルで実行した場合は以下のようになる

```
kumagai@rserv5:~$ git clone git@github.com:archlab-sciencetokyo/robbit_project.git
Cloning into 'robbit_project'...
remote: Enumerating objects: 269, done.
remote: Counting objects: 100% (46/46), done.
remote: Compressing objects: 100% (38/38), done.
remote: Total 269 (delta 13), reused 28 (delta 8), pack-reused 223 (from 1)
Receiving objects: 100% (269/269), 63.30 MiB | 10.43 MiB/s, done.
Resolving deltas: 100% (125/125), done.
kumagai@rserv5:~$ cd robbit_project/
kumagai@rserv5:~/robbit_project$ git submodule update --init --recursive
Submodule 'robbit/setting/CFU-Proving-Ground' (https://github.com/archlab-sciencetokyo/CFU-Proving-Ground.git) registered for path 'robbit/CFU-Proving-Ground'
Cloning into '/home/kumagai/robbit_project/robbit/CFU-Proving-Ground'...
Submodule path 'robbit/CFU-Proving-Ground': checked out 'cdd36452d6aa41d2fe8568df16544d94ad51ca65'
kumagai@rserv5:~/robbit_project$ cd robbit
kumagai@rserv5:~/robbit_project/robbit$ make init
```

```
patching file app/st7789.c
  Applying st7789.h.patch...
patching file app/st7789.h
  Applying top.v.patch...
patching file top.v
Merge complete.
make[1]: Leaving directory '/home/kumagai/Work/bthesis/robbit/refecter/robbit_project/robbit'
kumagai@rserv5:~/Work/bthesis/robbit/refecter/robbit_project/robbit$ cd CFU-Proving-Ground/
```

make initが正常に終了したことを表す

以降はCFU-Proving-Ground配下で作業

Bitstream, バイナリ作成: 環境構築

- CFU Proving Groundを利用するための環境を構築するため、CFU Proving Groundのリポジトリを参照し，READMEのStep. 1に沿って環境を構築していく

- 環境構築にはRISC-Vコンパイラ やXillinx社のVivadoなどが必要になるため、数時間かかる場合がある

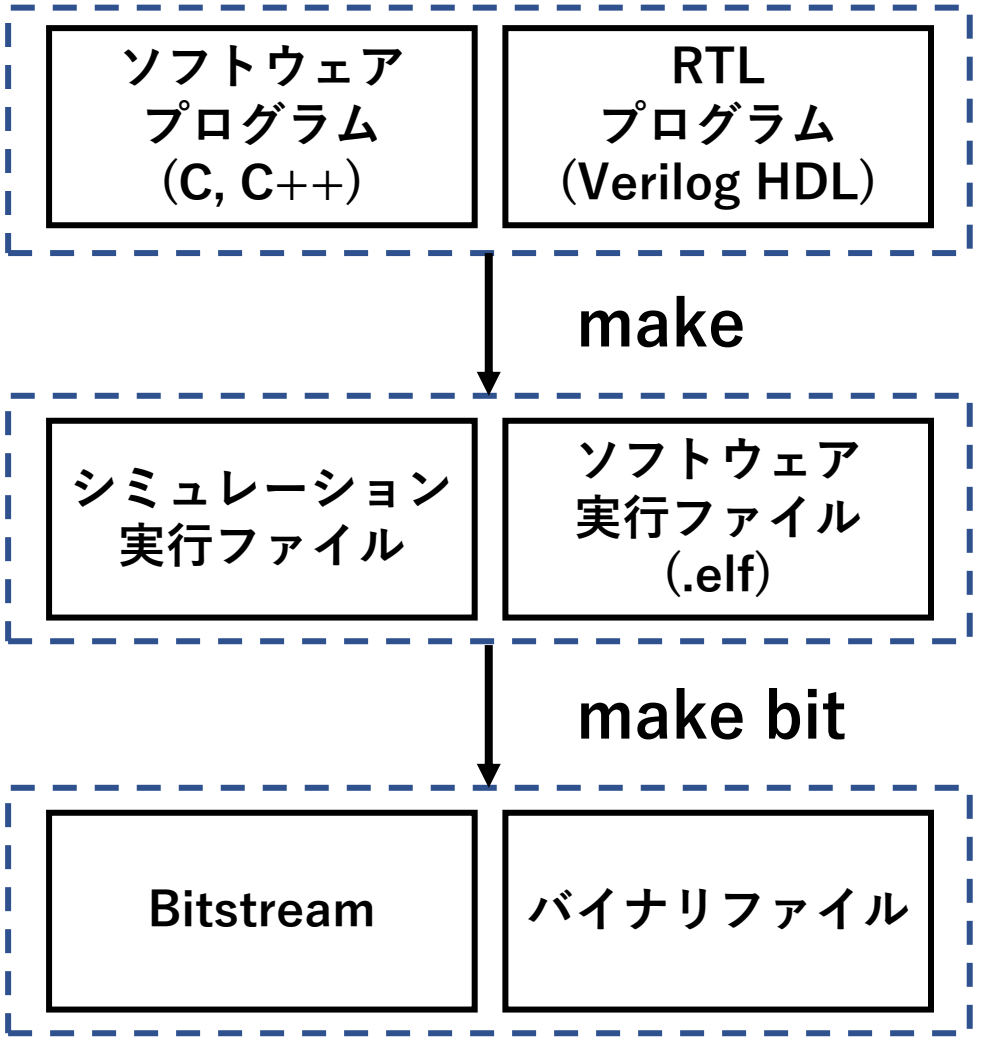
- 環境構築が困難な場合はACRiルームを使用することで、環境構築の手間を省くことができる

Bitstream, バイナリ作成: Bitstream作成

- 環境構築が終了したら，以下の手順でビルドとBitstream, バイナリファイルを生成する

- make bitまで実行すると，Bitstreamファイルとバイナリファイルが同時に生成される

  - make
  - make bit



Bitstream, バイナリ作成: Bitstream作成

- Bitstream生成が終了すると，ターミナルにタイミング制約の結果が出力されるので確認する
- slackが0以上であればタイミング制約を違反していない
- slackが0より小さい負の値の時はタイミング制約に違反している

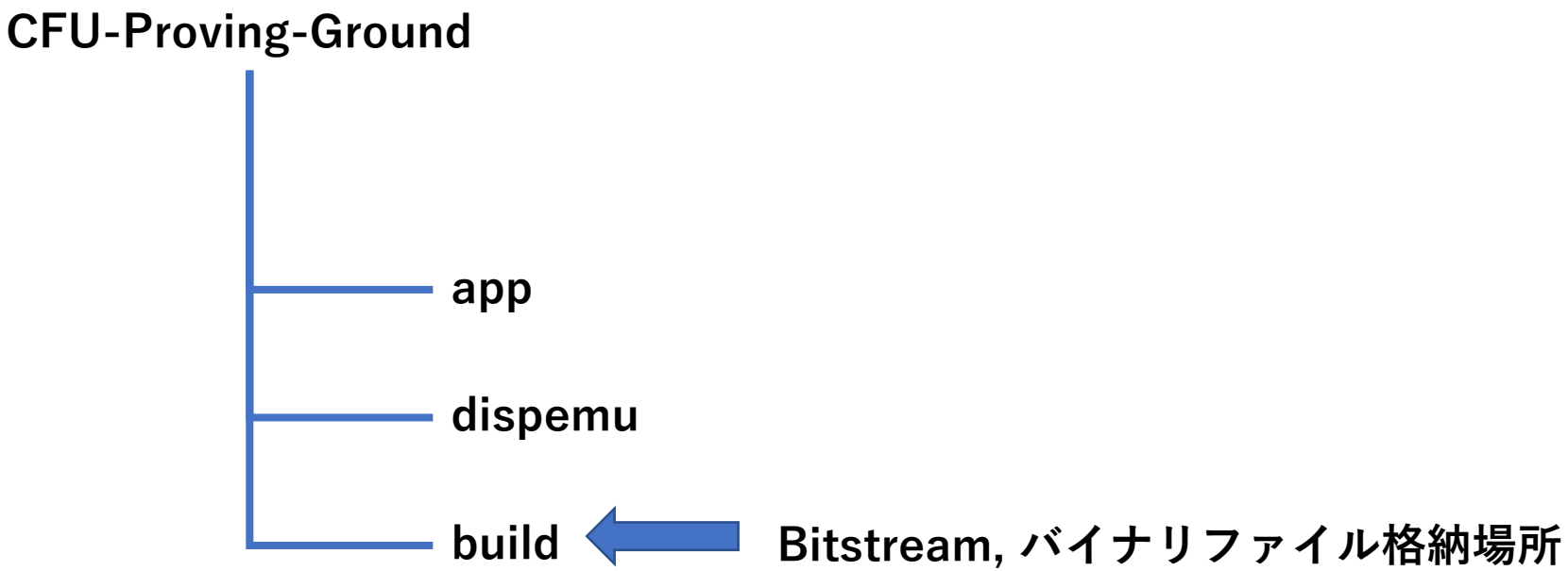
```
-------------------------------------------------------------------
                         required time                          4.516
                         arrival time                          -4.484
-------------------------------------------------------------------
                         slack                                  0.032
```

0以上なのでタイミング制約に違反していない

- タイミング制約を違反している場合の対策
- 動作周波数を下げる(35ページ参照)
- クリティカルパスの解消

Bitstream, バイナリ作成: Bitstream作成

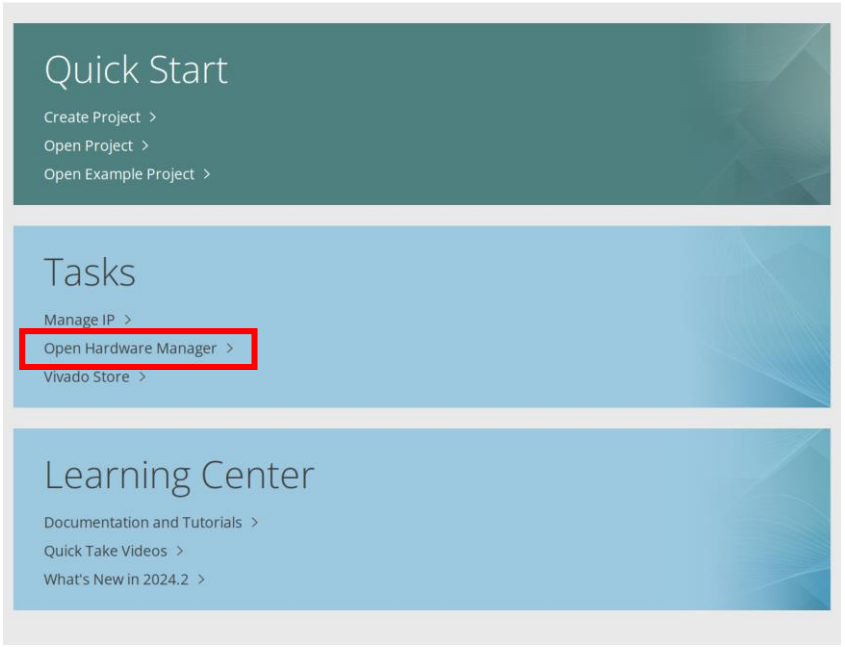
- make bitを実行すると，新たにbuildディレクトリが作成される
- buildディレクトリにはBitstreamファイルとバイナリファイルが格納される



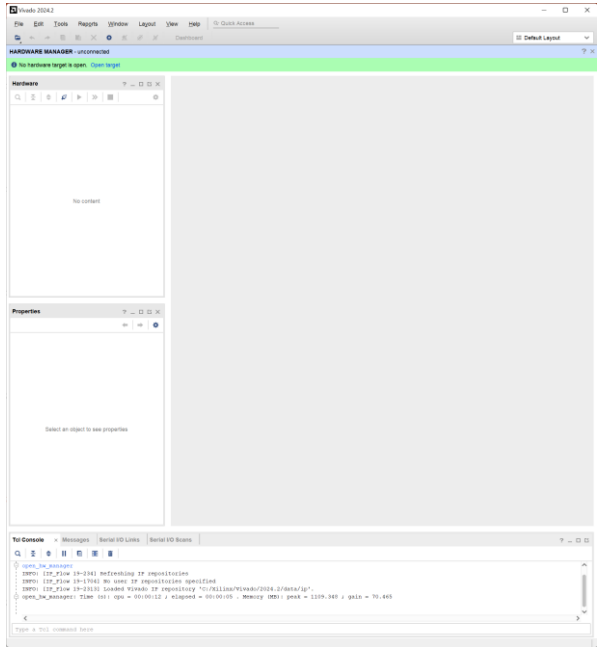
Bitstream, バイナリ作成: 書き込み準備

■ Vivadoを起動し，Open Hardware Managerを選択する

Vivado



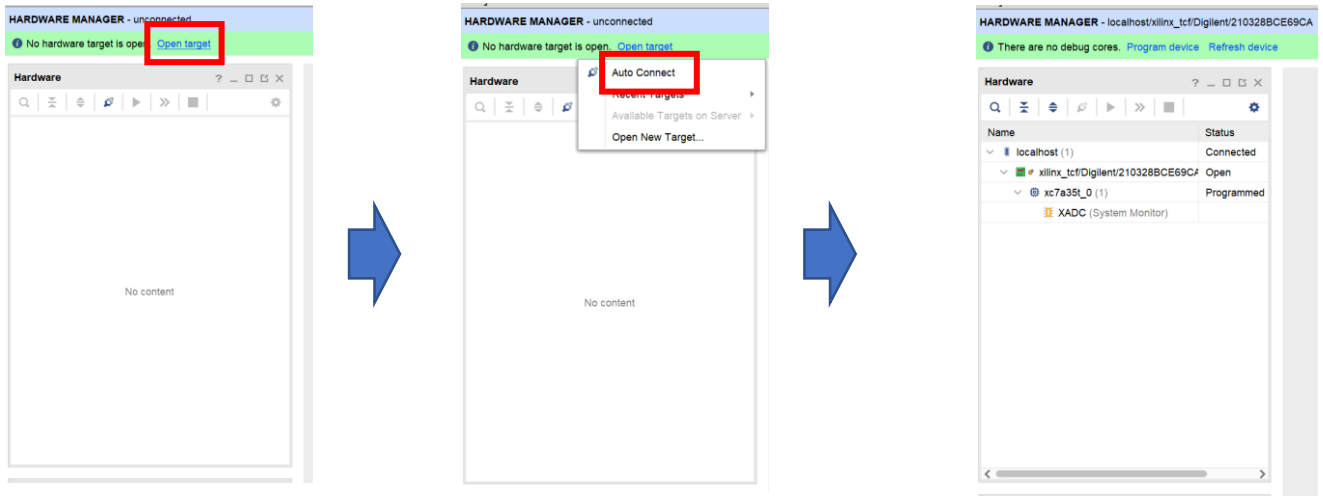
Vivado スタート画面



Hardware Manager

Bitstream, バイナリ作成: 書き込み準備

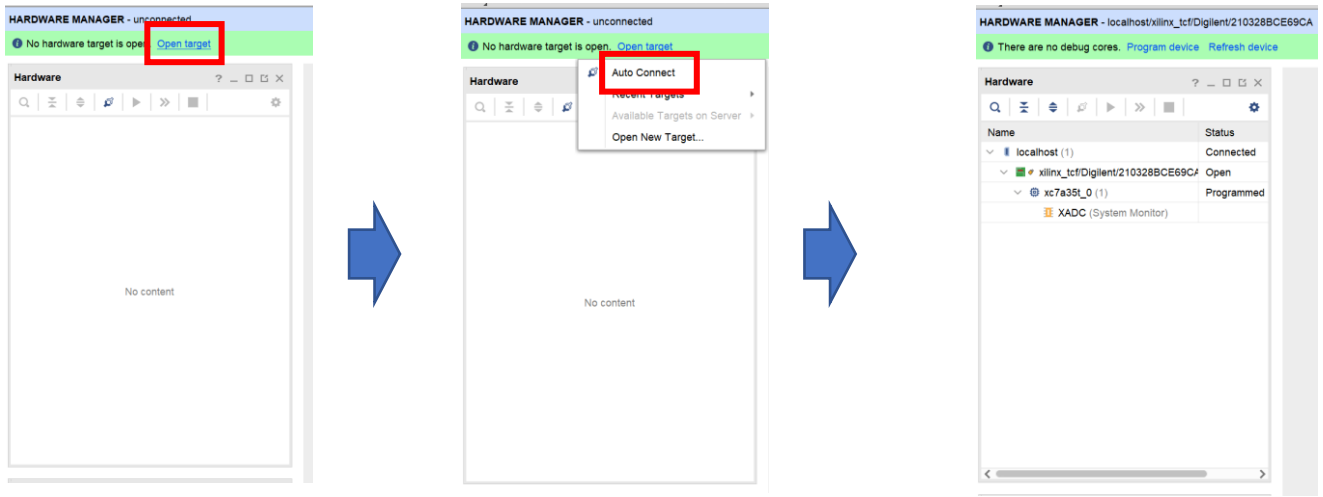
- PCとrobbitのFPGAボードをケーブルで接続する
- 接続したら、Vivadoの画面右上にある`Open target` -> `Auto connect`をクリック
- FPGAボードが認識されると下図の一番右側のように、認識したFPGAが表示される



接続成功時の画面例

Bitstream, バイナリ作成: 書き込み準備

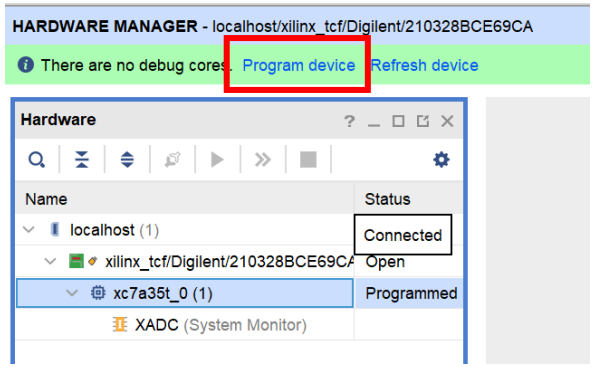
- PCとrobbitのFPGAボードをケーブルで接続する
- 接続したら、Vivadoの画面右上にある`Open target` -> `Auto connect`をクリック
- FPGAボードが認識されると下図の一番右側のように、認識したFPGAが表示される

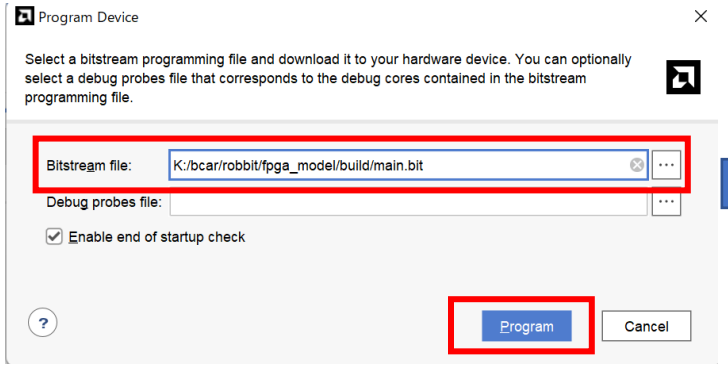


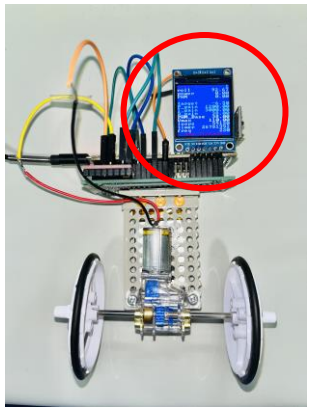
接続成功時の画面例

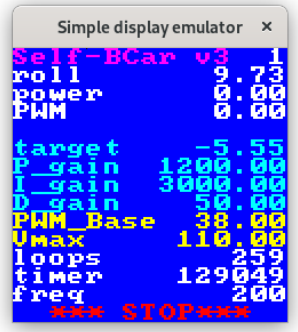
動作確認: Bitstream書き込み

- Vivadoの画面右上にあるProgram deviceを選択する
- Bitstream選択画面になるため，作成したbuildフォルダ内にあるmain.bitを選択しProgramボタンを選択する
- 書き込みが成功するとrobbitのディスプレイにパラメータや角度などの情報が表示される
- Bitstreamの書き込み内容はPCとrobbitとの接続を切ると失われる





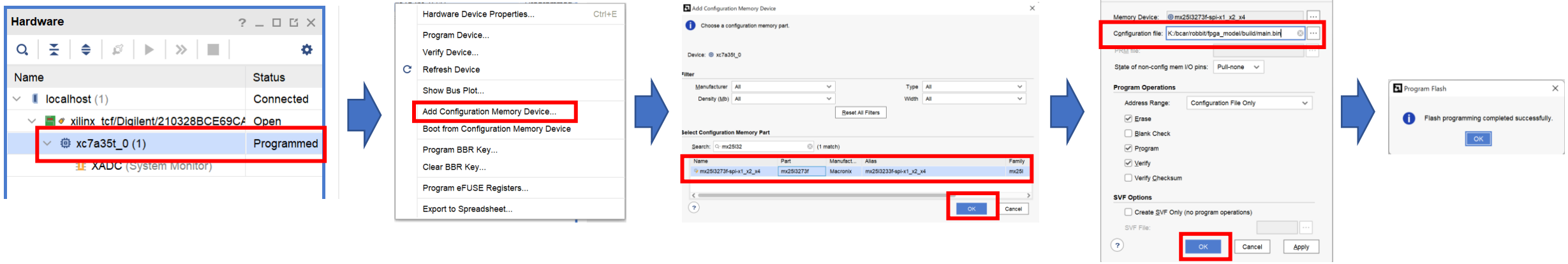




ディスプレイ表示例

動作確認: バイナリファイル書き込み

- PCとの接続を解除してrobbitを動かす場合には，バイナリファイルを書き込む
- FPGAの項目を右クリックし，`Add configuration memory device`を選択
- 検索欄に`mx25l32`と入力するとROMの候補が一つに絞られるので，該当のROMを選択しOKを押す
- Configuration fileを選ぶ画面が出てくるので，buildディレクトリのmain.binを選択し，OKで書き込み



動作確認

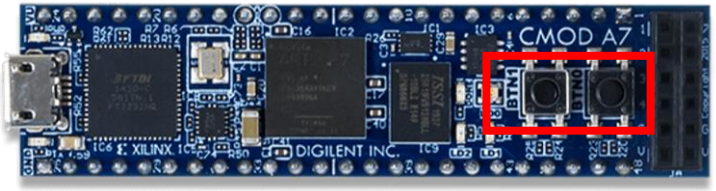
- 書き込みが終了したら、PCとrobbitの接続を解除し、robbitの電源を入れて動作を確認する。
- 動作確認はカーペットのようなある程度摩擦が生じる環境で行うと良い。
- 摩擦のない環境だと、その場で自立することが難しくなる。

動作確認の項目

- 垂直に自立させたときにディスプレイのrollが0度付近の値を示しているか
- 手にもって、robbitを傾けた時に傾けた方向に車輪が回転するか
- 垂直状態から90度傾けた時に車輪の動きが止まるか(プログラムでは40度傾くと止まるようになっている)

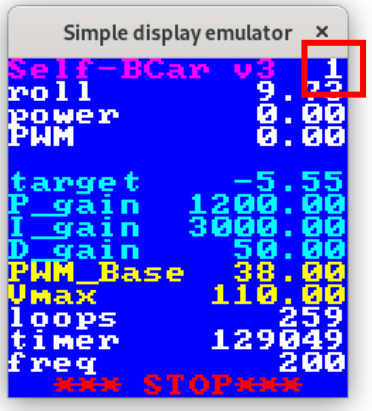
パラメータチューニング: チューニング方法

- FPGAボードに付属している2個のボタンでパラメータチューニングを行える
  BTN1 : パラメータ減少
  BTN0 : パラメータ増加
  BTN1とBTN0同時押し : 対象のパラメータ変更



Cmod A7-35T(FPGAボード) ボタン

- パラメータを複数同時に変更することはできないので，変えたいパラメータを指定する必要がある。変更対象のパラメータは数値としてディスプレイの以下の部分に示される。本資料ではこの数値をパラメータ番号と呼んでいる。



変更対象のパラメータ番号

チューニング可能なパラメータ

パラメータチューニング: パラメータ番号

■ パラメータチューニングの際に利用するパラメータ番号とは、変更可能なパラメータに割り当てられている番号である

パラメータ名	パラメータ番号	役割
Target	1	車体の目標角度
P_gain	2	比例要素のゲイン
I_gain	3	積分要素のゲイン
D_gain	4	微分要素のゲイン
PWM_Base	5	PWM信号の増加値
V_max	6	PWM信号の最大値

パラメータチューニング: その他のパラメータ

Institute of SCIENCE TOKYO

■ 前ページのパラメータ以外にもプログラム中で変更できるパラメータが存在する

- 動作周波数
  CPUの動作周波数も変更することができる。
  config.vhにある`CLK_FREQ_MHZ`が動作周波数を表している
  単位はMHzで記述する

```
// cpu
`define CLK_FREQ_MHZ 150  // operating clock frequency in MHz
```

150MHzに設定した例

ただし，大きくしすぎるとタイミング制約に違反するので，Bitstream生成時のslackをよく確認すること

パラメータチューニング: その他のパラメータ

- ループ周波数
  ループ周波数とはmain.cpp内に記述されているwhileループの周波数を表す
  main.cpp内の``LOOP_HZ`がループ周波数を表す
  値を大きくすればその分whileループ内の処理も高速に行われる
  単位はHzで記述する

```
#define LOOP_HZ      1000  // Hz of main loop
```

1000Hzに設定した例

- 積分要素の上限値
  アンチワインドアップ実装のため，積分要素の上限値を決定する
  main.cppの`I_MAX`に値が設定してある

```
#define I_MAX        0.4   // Anti-windup
```

上限値を0.4に設定した例

ディスプレイ表示

■ ディスプレイに表示されている他の値は以下の通りである

パラメータ名	役割
roll	車体の現在角度
power	PID制御の出力
PWM	PWM信号の値
loops	ソフトウェア内のループ処理の実行数
timer	軌道からのクロック数
freq	ソフトウェア内のループ処理の周波数

問題設定

パラメータチューニングの際にどの程度まで性能向上を目指すか、目標を考える
下記は一例である

■ 難易度：易しい
カーペットなどの摩擦のある領域で10分以上の自立を目指す

■ 難易度：難しい
摩擦の少ない領域でできるだけ長く自立させる
風や傾斜などの外乱を加える