STM32CubeIDE の導入（公開資料） (Introduction of the STM32CubeIDE)			作成 2021/9/10 ニューラルソフト有限会社 市来 博記
初版	－	2021/9/10	
B	PDF にメタ データを追加した。	2021/12/25	

概要

本書は、試作品開発時のハードウェアー プラットフォームの選択肢である ST マイクロエレクトロニクス社製の STM32 Nucleo マイコン開発ボードと STM32Cube 統合開発環境の使用方法について調査した結果を記したものです。

調査内容

STM32Cube 統合開発環境のインストールから SW4STM32 用プロジェクトの変換までを調査しました。（SW4STM32 用プロジェクトはモータ制御用シールドのサンプル プロジェクトです。）

調査項目を示します。

1 STM32Cube のインストール
2 新規プロジェクトの作成
3 LED 点滅処理のコード追加
4 プロジェクトのビルド
5 開発用ホスト PC と STM32 Nucleo マイコン開発ボードの接続
6 実行モジュールの転送と起動、及び LED 点滅の動作の確認
7 実行モジュールのデバッグ
  7.1 ブレーク ポイントの設定
  7.2 ステップ実行
  7.3 変数値の確認と変更
  7.4 レジスタ値の確認
  7.5 逆アセンブル
8 SW4STM32 用プロジェクトから STM32Cube 用プロジェクトへの変換
9 FreeRTOS のタスク起動の確認
10 デバッグ ライトの確認
11 モータの挙動の確認

前提条件

調査における前提条件を示します。

- 開発用ホスト マシンは、Windows10（64 ビット）がインストールされた PC（i7-7500U，16GB メモリ搭載のノート PC）を使用します。
- STM32 Nucleo マイコン開発ボードは、STM32 Nucleo-64 - NUCLEO-F411RE を使用します。
- モータ制御用シールドは、学習用 DC モータ制御シールド（スイッチ サイエンスで調達）を使用します。
- 本書の記載内容は Microsoft Visusal Studio での C/C++のデバッグの経験がある方が対象です。

調査結果

〔STM32Cube 統合開発環境インストール〕

1. ST マイクロエレクトロニクス社のホームページにアクセスしてアカウントを作成後、ログインする。（アカウント作成済みの場合は、そのままログインします。）
2. STM32 用統合開発環境のページへ移動する。
3. 概要タブに「ソフトウェア入手」のメニューがあるので、開発環境に合ったものをダウンロードする。（今回は STM32CubeIDE Windows Installer の最新バージョンを選択しました。又、ライセンス規約への同意を求められます。）
4. ダウンロードした ZIP ファイルを展開して、st-stm32cubeide_～_x86_64.exe を起動する。（ユーザ アカウント制御のダイアログが表示されるので、ソフトウェアの発行元を確認後、「はい」を選択します。）
5. インストール画面が表示されるので、ライセンスを確認後、デフォルト設定でインストールする。
6. デバイス インストール（ST Microelectronics ユニバーサル シリアル バス）確認のダイアログが表示されるので、「インストール」を選択する。
7. 「completed」が表示されたら、「next」を選択後、デスクトップのアイコン作成の有無を選択して「Finish」ボタンを押下する。
8. STM32Cube を起動すると、ワーク スペース ディレクトリの選択ダイアログが表示される。表示されているディレクトリで問題がなければ、そのまま「Launch」ボタンを押下する。（別ディレクトリにワーク スペースを配置する場合は、そのディレクトリを指定後に「Launch」ボタンを押下します。）
9. ファイヤー ウォールで stm32cubeide.exe の通信を許可するかどうかの確認ダイアログが表示されるので、セキュリティー環境に応じて許可する。
10. ST マイクロエレクトロニクス社の製品の品質向上に協力するかどうかを確認するダイアログが表示されるので、協力する場合は「Yes」、見合わせる場合は「No Thanks」を選択する。
11. 英語表示となっているので、日本語化する為、STM32Cube を終了する。
12. 統合開発環境 Eclipse 日本語化プロジェクト – Pleiade のサイトへアクセスする。
13. Pleiades プラグインをダウンロードする。（今回は Windows 用を選択しました。）

14. ダウンロードしたファイルを解凍して、ルートにある setup.exe(日本語化プラグインのインストーラのショート カット)を起動する。
15. 日本語化するアプリケーションを選択する画面が表示されるので、stm32cubeide.exe を指定する。（デフォルト設定でインストールした場合、stm32cubeide.exe は、C:/ST/STM32CubeIDE_x.x.x/STM32CubeIDE ディレクトリにあります。）
16. 「日本語化」ボタンを押下する。
17. 日本語化した後、このインストーラを削除しても問題がないことを知らせるダイアログが表示されるので、「OK」ボタンを押下後、日本語化プラグインのインストーラを終了する。
18. STM32Cube を起動して、日本語化されていることを確認する。
19. 好みに応じてメニューの「ウィンドウ/設定」の「一般/外観」でルック&フィールを変更する。（今回は「ダーク」に変更しました。）

これで、STM32Cube のインストールは完了です。

〔新規プロジェクトの作成〕

1. 日本語化した STM32Cube を起動する。

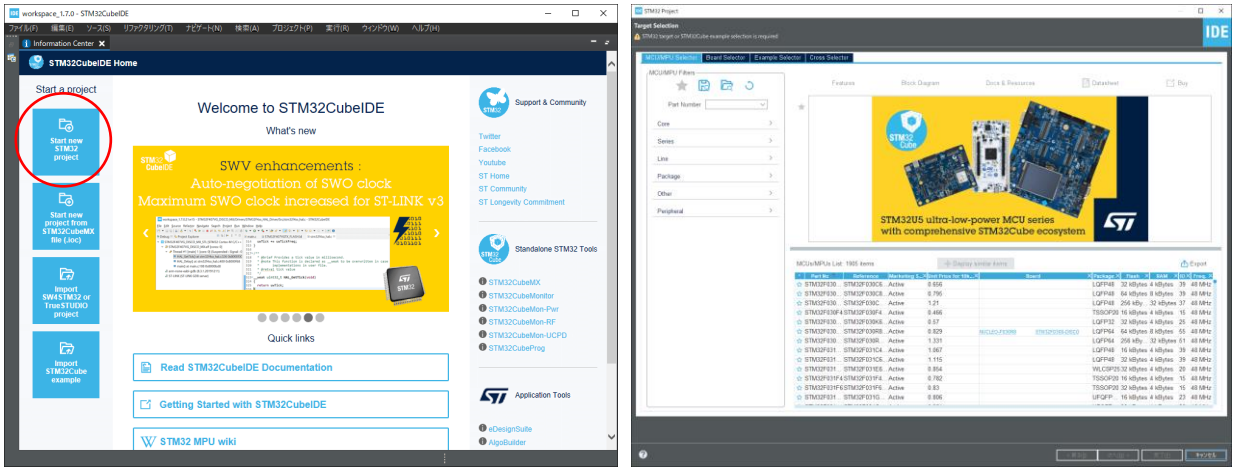

図 1 日本語化した STM32Cube を起動した画面と Target Selector 画面

STM32Cube を起動すると図 1 の左側のような画面が表示されます。この画面はメニューの「ヘルプ/Infomation Center」を選択することで、いつでも表示できます。
この画面の左上に表示されている「Start New STM32 project」ボタンを押下すると、図 1 の右側の画面が表示されます。

2. Board Selector タブを選択して、Nucleo-F411RE を選択して、「次へ」ボタンを押下する。
3. プロジェクト設定画面が表示されるので、プロジェクト名を「Nucleo-F411RE-Test」とし、その他の設定はデフォルトのまま、完了ボタンを押下する。
4. 全てのペリフェラルをデフォルト モードで初期化するかどうかを選択するダイアログが表示されるので、「はい」を選択する。
5. デバイス構成用のパースペクティブ（ビューとエディターの組み合わせ）を切替るかどうかを選択するダイアログが表示されるので、「はい」を選択する。

6. ボード サポート パッケージなどの必要なファイルがダウンロードされ、解凍後に自動的にインストールされる。（かなり時間が掛かります。）
処理が終了すると、図 2 のような表示になります。

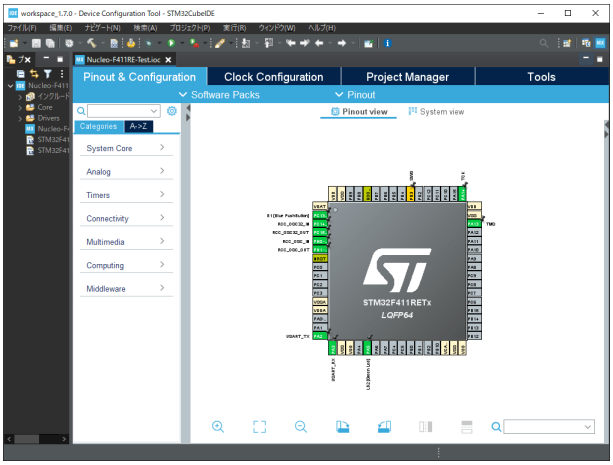

図 2 デバイス構成ツール画面

この画面で入出力ピンの割り当、デバイスの構成、クロックの設定、及びヒープとスタックのサイズ、スレッド セーフ サポートなどの設定を行うことができます。又、この画面はプロジェクト・エクスプローラから ioc ファイルを選択することで、いつでも表示することができます。今回は全てデフォルト設定のままでよいので、画面を閉じます。

これで、新規プロジェクトの作成は完了です。続いて、C 言語ソースに LED 点滅処理のコードを追加します。

〔LED 点滅処理のコード追加とプロジェクトのビルド〕

1. プロジェクト・エクスプローラで “Core/Src/main.c”をダブル クリックして開く。

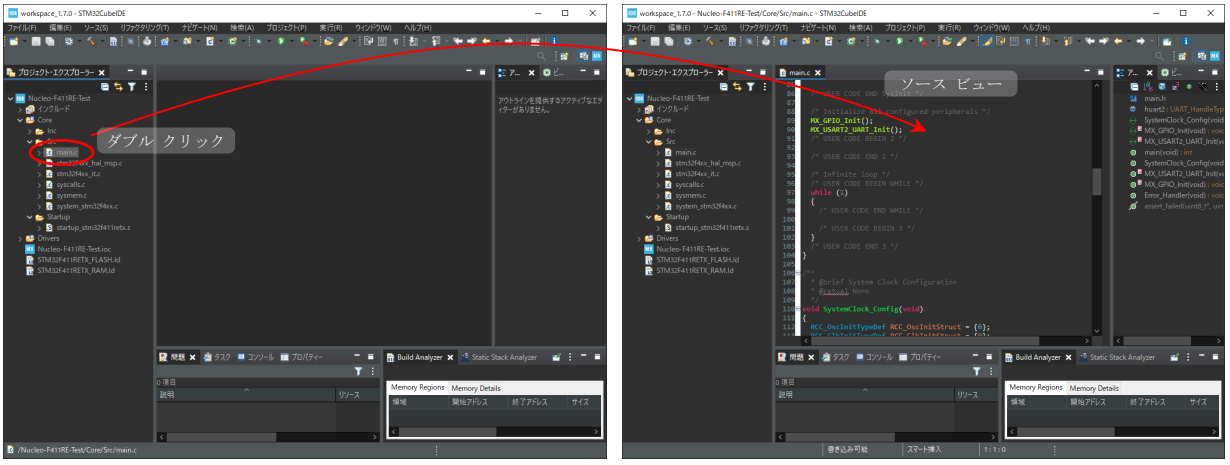

図 3 プロジェクト エクスプローラ操作によるソース コードの表示

main.c 内の main 関数は、一連のブート コードが実行され、
”Core/Startup/startup_stm32f411retx.s”内の初期化処理が実行された後に呼び出されます。

2. main.c の 101 と 102 行目の間に次のコードを追加します。

```
HAL_Delay(500);
HAL_GPIO_TogglePin(LD2_GPIO_Port,LD2_Pin);
```

（本操作の記載は、qiita：@usashirou さんの「STM32CubeIDE を使ってみよう How To STM32CubeIDE 日本語版 L チカ編」の記事を参考にしています。）

HAL と入力して、Ctrl + スペースを押下すると、”HAL”で始まる候補が表示されます。

3. メニューの「ビルド/プロジェクト/プロジェクトのビルド」、又はプロジェクト エクスプローラのプロジェクト（Nucleo-F411RE-Test）を右クリックして表示されるメニューの「プロジェクトのビルド」を選択してビルドする。

タイプミスがなければ、「xx:xx:xx Build Finished. 0 errors, 0 warnings. (経過 x s.xxx ms)」が表示されます。

4. STM32Cube の実行とデバッグの操作を Microsoft Visual Stdio に合わせる為に、キーのショートカットの設定を変更する。

メニューの「ウィンドウ/設定」を選択すると設定画面が表示されます。左側に表示されるリストで、「一般/キー」を選択して、右側に表示されるスキームで、「Microsoft Visual Stdio」を選択して、「適用して閉じる」ボタンを押下します。（以降の記載は、Microsoft Visual Stdio のキー割り当てで記載します。）

5. F5（デバッグの開始）を押下する。（プロジェクト エクスプローラに表示されている Nucleo-F411RE-Test プロジェクトの何れかの要素が選択状態でなければなりません。）

起動構成のプロパティの編集画面が表示されますが、そのまま「OK」ボタンを押下します。

図 4 のエラー ダイアログが表示されます。

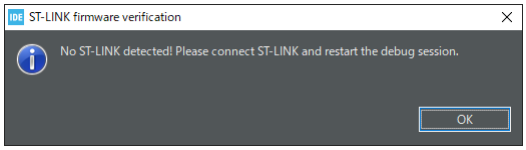

図 4 開発対象のボード間の接続異常を通知するダイアログ

開発対象のボードを開発ホストに接続していないので当然の結果ですが、誤って接続せずに起動してもフリーズするようなことはないことが分かりました。

ここでは、デバッグ モードでビルドしましたが、メニューの「プロジェクト/ビルド構成/アクティブにする」で Debug と Release を切り替えることができます。

これで、LED 点滅処理のコード追加とプロジェクトのビルドは完了です。続いて、開発用ホスト PC と STM32 Nucleo マイコン開発ボードの接続を行います。

〔開発用ホスト PC と STM32 Nucleo マイコン開発ボードの接続〕

1. 開発用ホスト PC と STM32 Nucleo-64 を USB ケーブル（STM32 Nucleo-64 側のコネクターは mini-B）で接続する。

STM32 Nucleo マイコン開発ボードの LED1 が緑色点灯し、LED2 が赤色点灯します。

デバイス マネージャでポート(COM と LPT)を確認すると、「STMicronics STLink Virtual COM Port」があることが分かります。

これで、開発用ホスト PC と STM32 Nucleo マイコン開発ボードの接続は完了です。続いて、実行モジュールの転送と起動、及び LED 点滅動作の確認を行います。

〔実行モジュールの転送と起動、及び LED 点滅動作の確認〕

1. F5（デバッグの開始）を押下する。
   ファイヤー ウォールで st-link_gdbserver.exe の通信を許可するかどうかの確認ダイアログが表示されるので、セキュリティー環境に応じて許可します。パースペクティブ切替の確認ダイアログも表示されますので、「切替」ボタンを押下します。（「常にこの設定を使用する」にチェックを入れてもよいかも知れません。又、ファーム ウェアの更新が必要であることを通知するダイアログが表示される場合は、そのダイアログで「更新」を選択します。）
   STM32 Nucleo マイコン開発ボードの LD1 が緑/赤で点滅し、図 5 のように STM32Cube のソースビューに main.c が表示されます。（main 関数の先頭まで実行され、HAL_INIT 関数の呼び出し前で停止している状態です。）

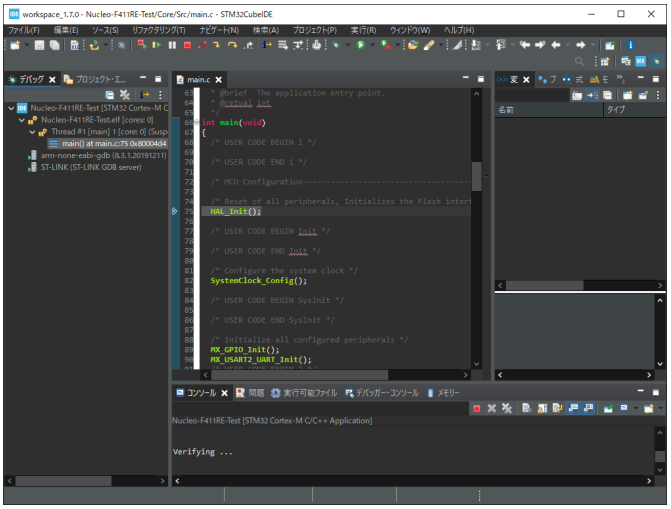

図 5 デフォルトの実行構成で実行した STM32Cube の表示

2. もう一度、F5（続行）を押下する。
   STM32 Nucleo マイコン開発ボードの LD2 が 1 秒周期（500 ミリ秒間点灯、500 ミリ秒消灯の繰り返し）で緑色点滅します。
   終了する場合は、メニューの「実行/終了」を選択します。（ボード上のプログラムは動作したままです。）

これで、実行モジュールの転送と起動、及び LED 点滅の動作の確認は完了です。

〔実行モジュールのデバッグ〕

使用頻度の高いデバッグ機能と、その操作方法を**表 1** に示します。

表 1 STM32Cube のデバッグ機能と操作方法

機能名	操作方法
ブレイク ポイントの切り替え(設定/解除)	F9 を押下
全てのブレイク ポイントを除去(削除)	メニューの「実行/全てのブレイク ポイントを除去」を選択
デバッグの開始/続行	F5 を押下
ステップ イン	F11 を押下
ステップ オーバー	F10 を押下
ステップ アウト	Shift + F11 を押下
変数の確認	ブレーク ポイントで停止している時に、ソース ビューで確認したい変数にマウス ポインタを近づけると、ポップ アップ ウィンドウが表示され、そこで変数値の確認と変更を行えます。変数ビュー*1 でも同じ操作が可能です。
MPU レジスタ値の確認	レジスタ ビュー*2 でレジスタ値の確認と変更を行えます。
逆アセンブル	逆アセンプル ビュー*3 でアセンブリー コード(デバッグ用モジュールでは C/C++ソースとの混合)を確認できます。 ステップ実行、ブレーク ポイントの設定も可能です。

*1 変数ビューは図 6 を参照のこと。
*2 レジスタ ビューは**図 7** を参照のこと。
*3 レジスタ ビューは**図 8** を参照のこと。

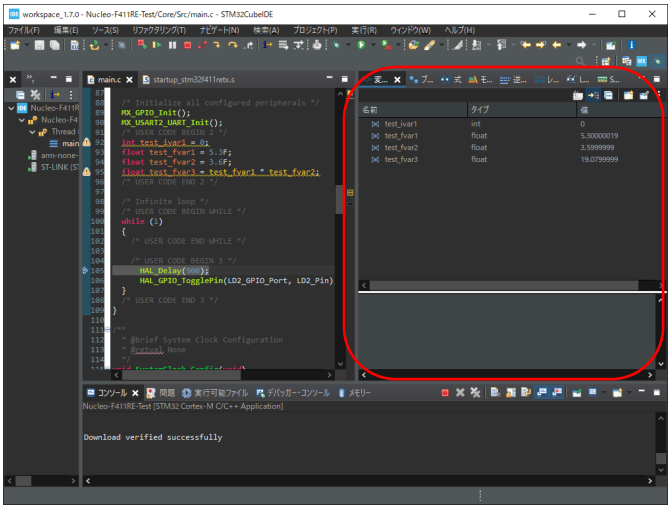

図 6 STM32CubeIDE の変数ビュー

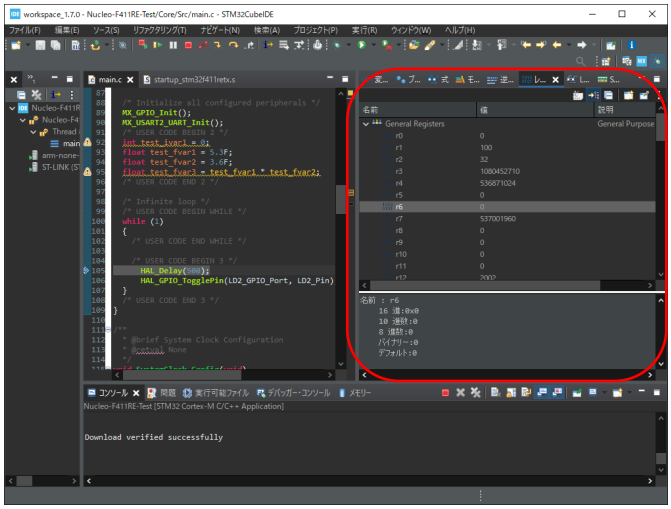

図 7 STM32CubeIDE の変数ビュー

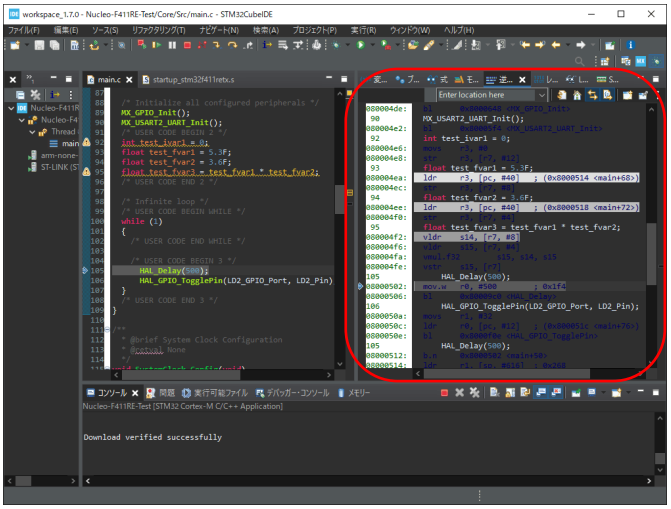

図 8 STM32CubeIDE の逆アセンブル ビュー

〔SW4STM32 用プロジェクトから STM32Cube 用プロジェクトへの変換と動作確認〕

1. STM32CubeIDE を起動後、メニューの「ヘルプ/Infomation Center」を選択する。
2. Infomation Center 画面の左側にある「Import SW4STM32 or True STUDIO project」ボタンを押下する。（インポート設定画面が表示されます。）

図 9 STM32CubeIDE の Infomation Center 画面

3. インポート設定画面でインポートするプロジェクトがあるディレクトリを指定して、「完了」ボタンを押下する。（今回は学習用 DC モータ制御シールドのプロジェクトがあるディレクトリを指定しました。）
   インポートの確認ダイアログが表示されます。プロジェクトの変換が必要であることと、変換前に旧プロジェクトのバックアップが作成さることが表示されます。表示内容を確認してから「OK」ボタンを押下します。

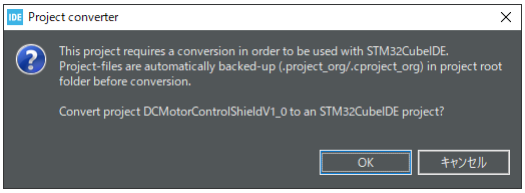

図 10 インポートの確認ダイアログ

問題なく変換が完了したが、手動構成が必要な場合があることを知らせるダイアログが表示されます。（プロジェクト変換のログは、プロジェクト直下の～.log で確認できます。）
表示内容を確認してから「OK」ボタンを押下します。

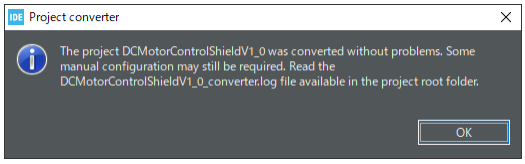

図 11 インポート完了ダイアログ

4. メニューの「プロジェクト/ビルド構成/アクティブにする」で Debug モードであることを確認してからプロジェクトをビルドします。
5. 今回調達した学習用 DC モータ制御シールドの SW4STM32 用サンプル プロジェクトを STM32Cube 用のプロジェクトに変換してビルドすると、幾つかワーニングが出力されました。
ワーニングの内容は以下の通りです。

```
../Src/main.c: In function 'main':
../Src/main.c:111:1: warning: label 'Skip_SystemClock_Config' defined but not used [-Wunused-label]
  111 | Skip_SystemClock_Config:
      | ^~~~~~~~~~~~~~~~~~~~~~~
../Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS/cmsis_os.c: In function 'osDelay':
../Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS/cmsis_os.c:368:31: warning: division by zero [-Wdiv-by-zero]
  368 |   TickType_t ticks = millisec / portTICK_PERIOD_MS;
      |                              ^
../Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS/cmsis_os.c: In function 'osSignalWait':
../Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS/cmsis_os.c:600:22: warning: division by zero [-Wdiv-by-zero]
  600 |     ticks = millisec / portTICK_PERIOD_MS;
      |                      ^
../Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS/cmsis_os.c: In function 'osMutexWait':
../Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS/cmsis_os.c:684:22: warning: division by zero [-Wdiv-by-zero]
  684 |     ticks = millisec / portTICK_PERIOD_MS;
      |                      ^
../Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS/cmsis_os.c: In function 'osSemaphoreWait':
../Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS/cmsis_os.c:837:22: warning: division by zero [-Wdiv-by-zero]
  837 |     ticks = millisec / portTICK_PERIOD_MS;
      |                      ^
../Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS/cmsis_os.c: In function 'osMessagePut':
../Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS/cmsis_os.c:1118:20: warning: division by zero [-Wdiv-by-zero]
 1118 |   ticks = millisec / portTICK_PERIOD_MS;
      |                    ^
../Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS/cmsis_os.c: In function 'osMessageGet':
../Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS/cmsis_os.c:1166:22: warning: division by zero [-Wdiv-by-zero]
 1166 |     ticks = millisec / portTICK_PERIOD_MS;
      |                      ^
../Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS/cmsis_os.c: In function 'osMailGet':
../Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS/cmsis_os.c:1356:22: warning: division by zero [-Wdiv-by-zero]
 1356 |     ticks = millisec / portTICK_PERIOD_MS;
      |                      ^
../Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS/cmsis_os.c: In function 'osDelayUntil':
../Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS/cmsis_os.c:1549:32: warning: division by zero [-Wdiv-by-zero]
 1549 |   TickType_t ticks = (millisec / portTICK_PERIOD_MS);
      |                              ^
../Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS/cmsis_os.c: In function 'osMessagePeek':
../Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS/cmsis_os.c:1618:22: warning: division by zero [-Wdiv-by-zero]
 1618 |     ticks = millisec / portTICK_PERIOD_MS;
      |                      ^
```

取り敢えず、F5 で起動すると main 関数の先頭で停止するのですが、C 言語のソースが表示されません。そのまま再実行すると 0 割発生で Infinite Loop に突入します。
プロジェクトの変換ログ、.cproject_org と.cproject の差異、C 言語のソースを調べた結果、以下のことが原因であることが分かりました。

デバッグ起動で C 言語のソースが表示されない原因：

- Debug モードと Release モード用のデバッグ情報出力のオプション設定が逆になっている。

0 割が発生する原因：

- 1 ティックの時間を 1 ミリ秒から 50 マイクロ秒(1000Hz → 20000Hz)に変更している為、portTICK_PERIOD_MS の値が 0 となる。
- SW4STM32 では有効であったビルド ツールの noexceptions の指定が、STM32Cube ではサポートされなくなった。

以下の対応を行った後、デバッグ起動すると C 言語ソースが表示されるようになり、タスクが正常に起動されるようになりました。

- Debug モード用のデバッグ情報出力のオプション設定を g0 から g3 に変更
- Release モード用のデバッグ情報出力のオプション設定を g3 から g0 に変更
- ミリ秒からティック数を求める際、マイクロ秒に換算してから計算するように修正
  Middlewares/Third_Party/FreeRTOS/Source/CMSIS_RTOS/cmsis_os.c
  Middlewares/Third_Party/FreeRTOS/Source/portable/GCC/ARM_CM4F/portmacro.h
  （FreeRTOS へのミリ秒指定は約 1.2 時間に制限されます。）

学習用 DC モータ制御シールドのサンプル プログラムは以下の四つのタスクで構成されています。

- 電流マイナー ループ(50μ秒周期)　Src/control.c：270 MinorLoopTask()
- 加速度メジャー ループ(200μ秒周期)　Src/control.c：181 MajorLoopTask()
- シリアル通信　Src/control.c：132 SerialCommunicationTask()
- アイドル　Src/freertos.c：162 DefaultTask()

電流マイナー ループ タスクと加速度メジャー ループ タスクの処理の流れを見ることで、サンプル プログラムのモータ制御の内容が分かります。

teraterm を起動して、以下の設定を行うと、シリアル通信タスクからの情報を受信できます。
(シリアル通信タスクの通信設定は Src/usart.c：62 MX_USART2_UART_Init 関数内にあります。)

- ポート　STMicroelectronics STLink Virtual COM Port
- スピード　115200
- データ長　8 ビット
- パリティ　none
- ストップ ビット　1 ビット
- フロー制御　なし

目視によるザックリした確認ですが、モータのプーリーにマジックなどで印をつけると、モータの動きを確認できます。

これで、STM32Cube のインストールから旧プロジェクト変換後のコードの動作確認まで完了です。

所感

まず、STM32CubeIDE ですが、開発に必要な機能がオール イン ワンになっており、非常にシンプルです。そして、何より導入時の敷居が低いことが特徴です。（使い込んで行くと何か問題が見えてくるかも知れません。）

ST マイクロエレクトロニクス社の開発用ボードは、とにかく種類が豊富で、低コストです。（逆にどれが適しているのか分かり難いかも知れません。）

今回の調査の一番の収穫は、技術的なことではなく、主力製品の半導体を販売するための ST マイクロエレクトロニクス社の取り組みを垣間見れたことです。