STM32-NUCLEO向け
オフライン作業ひととおりやってみました

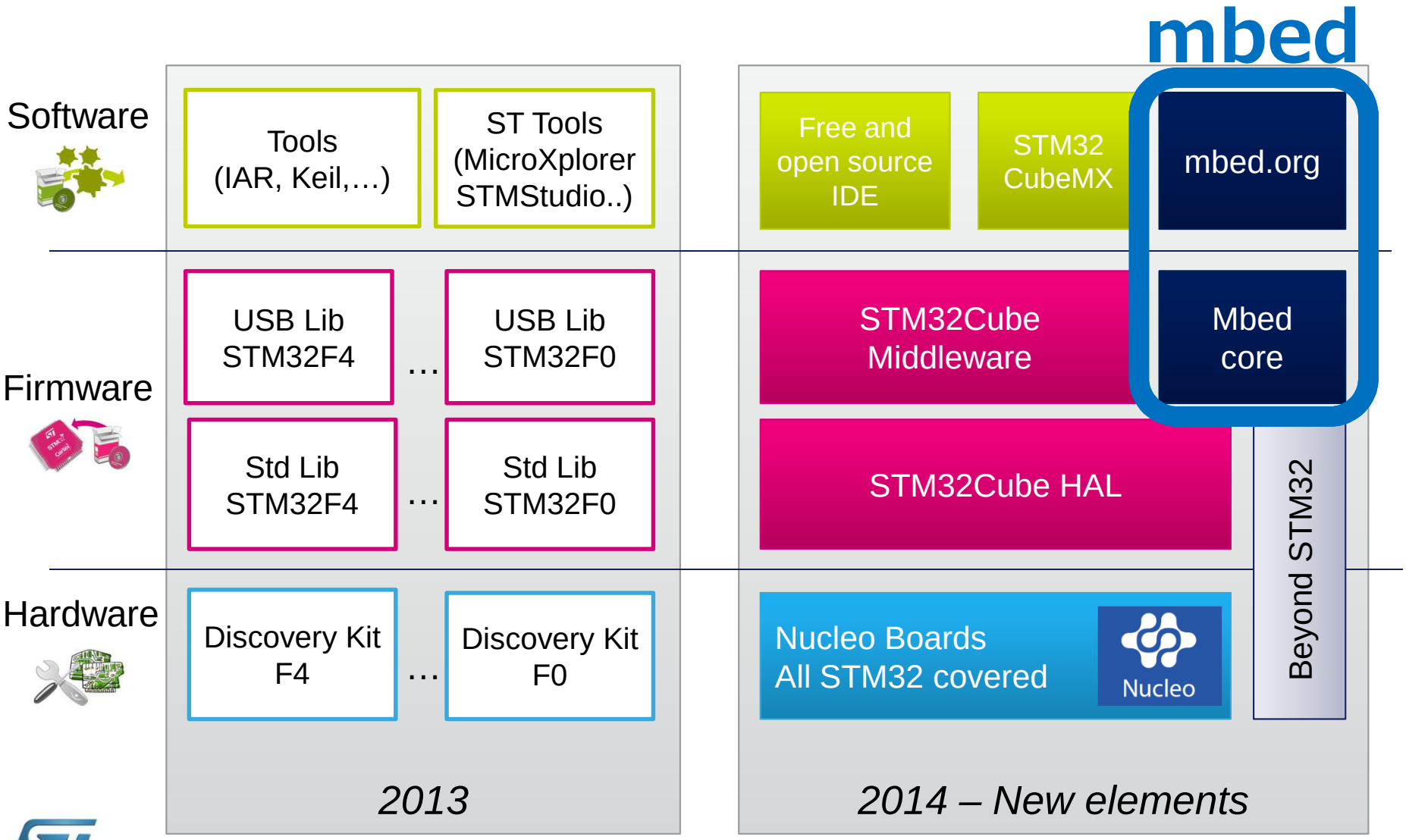
mbed祭り2014@春の大阪

life.augmented

- 重い腰を上げてやっとmbedに対応に乗り出したST。mbed対応は、数年前から水面下で進めていたSTM32マイコンエコシステムのリニューアルの一部という位置付けです。実際に対応されたものが出てくると、どう使っていけるか？というところで悩んでいます。導き出したひとつの答えと、それをもとにした作業の中で、いちユーザーとして体験・会得した事柄をご紹介したいと思います。話のタネは、mbed SDKを使って何かをつくった、というよりも、設定を調整してmbed SDKライブラリをビルドした事例、開発環境エクスポート形式の追加など、STM32-NUCLEOをmbedのプラットフォームとして使ってもらうためのオフライン作業結果です。

- **自己紹介**
  - 矢郷　洋一（やごう　ひろかず）
  - STマイクロエレクトロニクス（株）マイクロコントローラ製品部
  - ST製マイコン製品の技術サポート、邦訳マニュアルの校正をしています。といってもマイコン担当部署に異動してからまだ3年目です。経験豊富な「マイコンマスター」たちに圧倒されながら腕を磨く日々です。以前は、テレビ放送受信機向けアプリケーションプロセッサの分野で似た仕事をしていました。

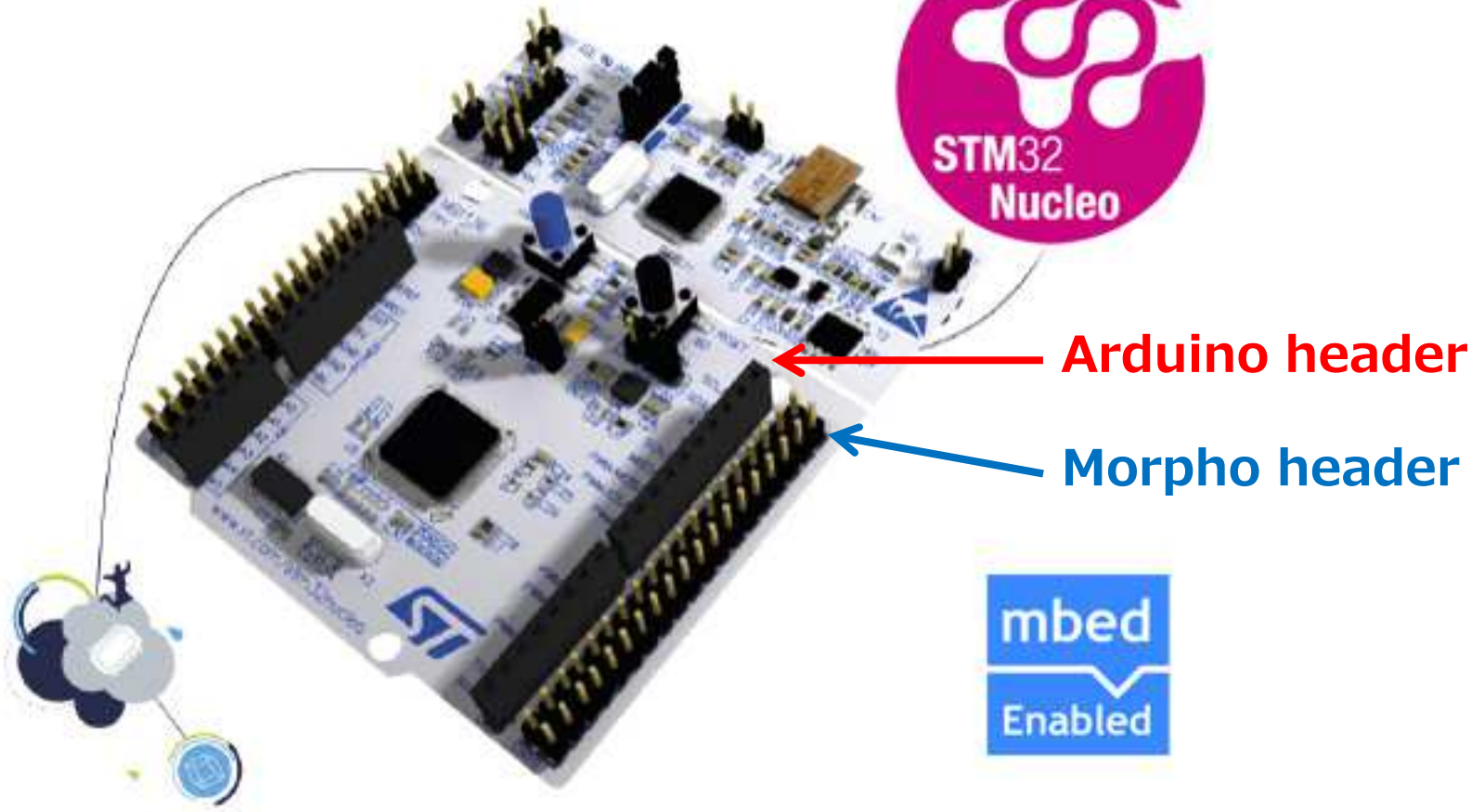
STM32 Ecosystem: News summary



life.augmented

mbed対応プラットフォーム

STM32-NUCLEO基板



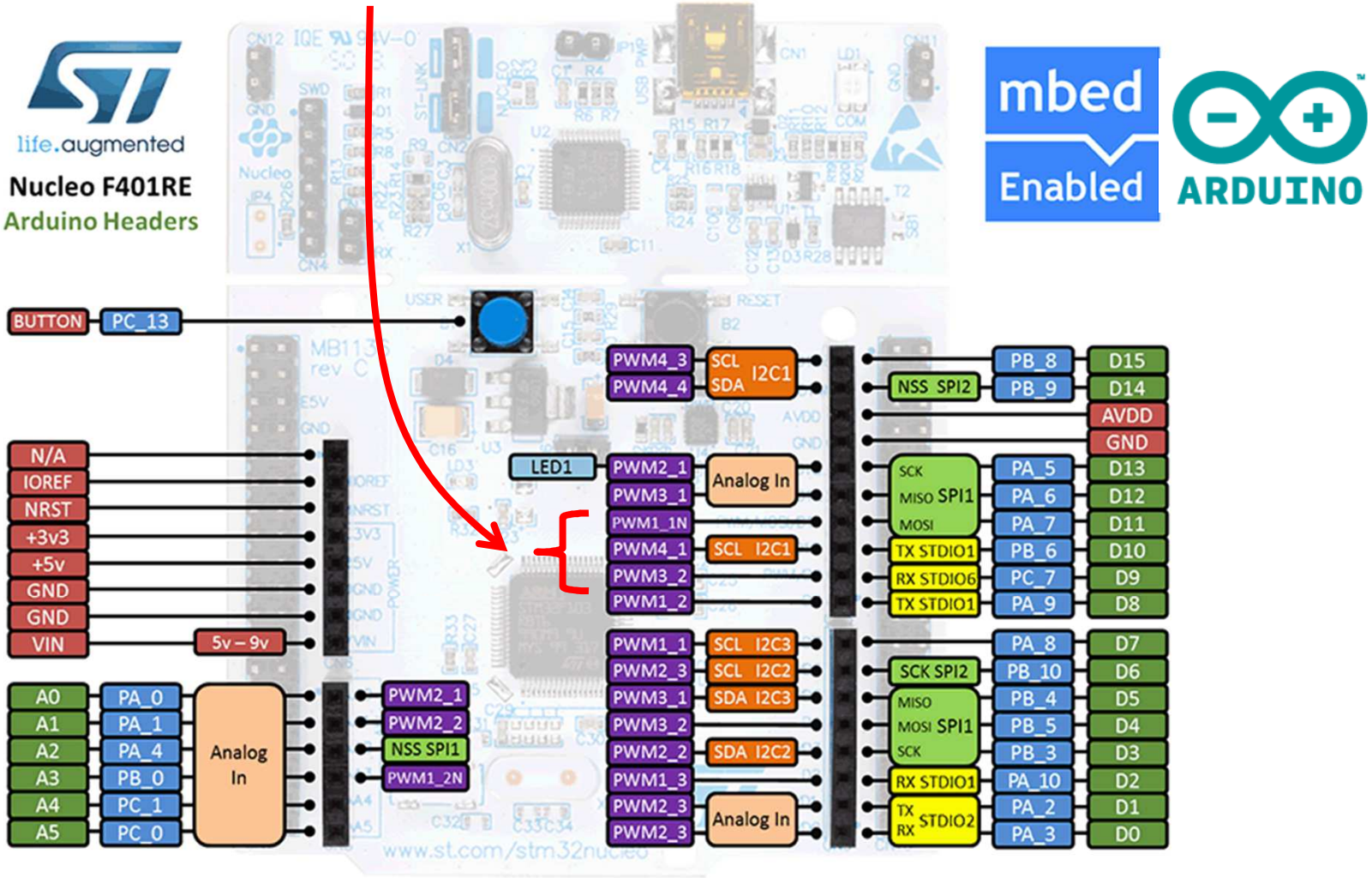
基板ピン互換。

life.augmented

NUCLEO – Arduino Headers

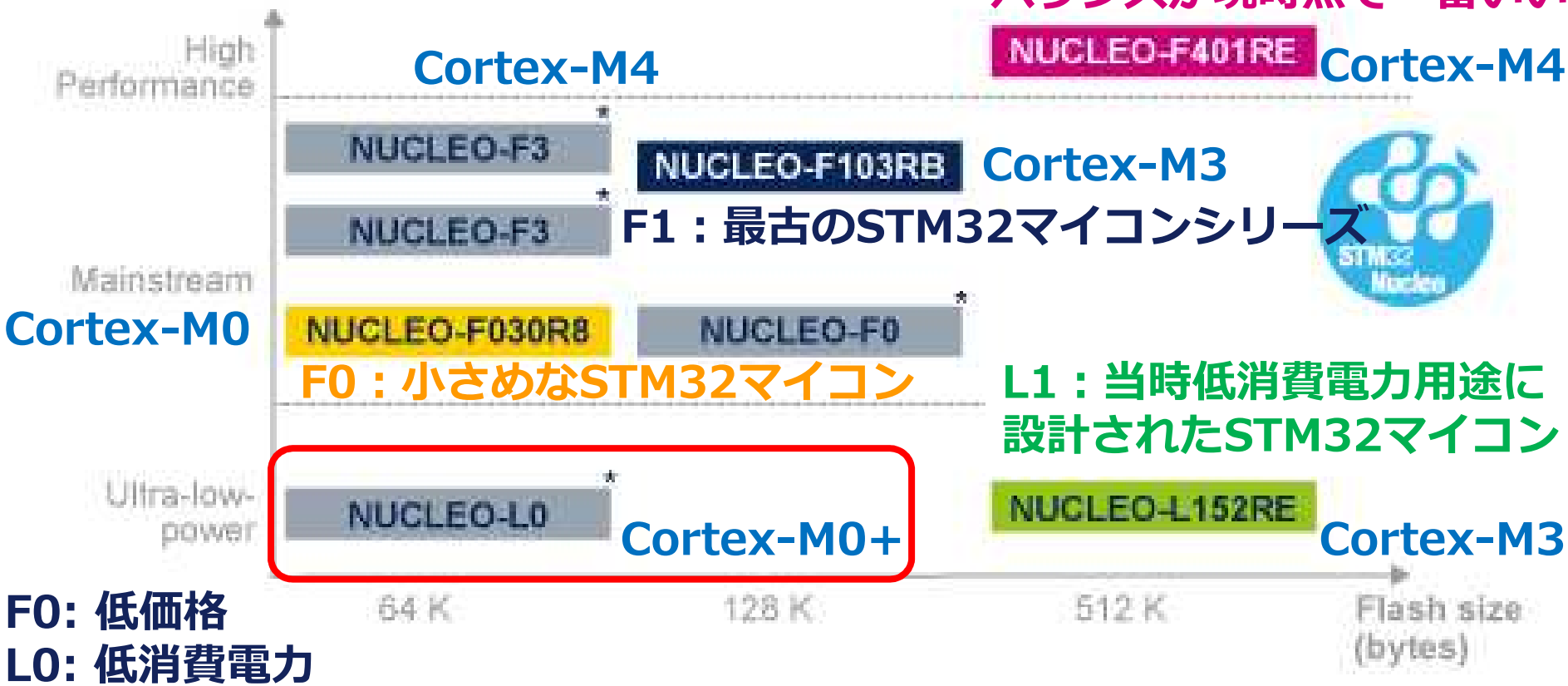
（Arduino UNO R3互換ピンヘッダ）

この3つPWMだけ割り当てられているTimerペリフェラルが基板ごとに異なります。



NUCLEOシリーズ

F401：消費電力と性能のバランスが現時点で一番いい



F0: 低価格

L0: 低消費電力

* Support for STM32 L0, STM32 F3 and more STM32 F0 coming soon

life.augmented

念願のCortex-M0+搭載STM32マイコン。L1シリーズに対する製品フィードバックをもとにして低消費電力をテーマに設計。

mbed、どう使えるだろうか？

- mbed SDKはマイコンファームウェア開発の敷居を下げるという特長があるので：
- 平常業務でのソフトウェアプラットフォームとして使えないか？
  - デモ環境の構築、トラブルシューティング時の再現環境の構築に役立ちそう・・・
  - ==> ブレークポイントを張りたい。デバッガを使えるようにしたい。
- マスマーケットに使えそう？
  - ==> LPCXpresso IDEのような無償IDEがあるといいなあ・・・。そういえば事業本部が無償IDEとして「**CooCox CoIDE**」というのを紹介してたな・・・。

オフラインコンパイル・デバッグ

- 現時点のmbedクラウド開発環境では、NUCLEO向けにオンラインIDEからKeil-uVisionプロジェクト形式でexport可能。exportするとコンパイル済のmbed SDKライブラリとアプリケーションのソースファイルmain.cppがローカルPCにダウンロードされるので、Nucleo_blink_led/サンプル+NUCLEO-F401REでやってみたところ、見事につまづく。exportされるmbed SDKライブラリはハードウェアFPU設定 = “無効”設定でコンパイル済なのに、exportされるuVisionプロジェクトオプションがハードウェアFPU設定 = "有効"になっている・・・。

- main.cpp: ハードウェアFPU使用でコンパイルされる。

- mbed SDKライブラリ: ハードウェアFPU非使用でコンパイル済。
  - SystemInit()実行後もCPACRレジスタが0のまま。

- main.cppでwait(0.2)のような浮動小数点を取り扱う個所でFloating-point instruction実行時にHard-fault例外が起きる。

対処

- ひとまずuVisionのプロジェクトオプションを調整。
- [Porject] – [Options for Target …] – [Target]タブ
- Floating Point Hardware = Not Used

- 「Hard FPUを使えるようにしたいんですけどー、」と言われたらどうしよう・・・・・・・。

mbed SDKフルビルドにチャレンジ

- 必要なもの
  - Python ==> 2.7.6をインストール。
    - C:¥Python27;を環境変数PATHに追加。
  - mbed masterブランチ ==> .zipスナップショットをダウンロード @ https://github.com/mbedmicro/mbed。
  - mbed-master/workspace_tools/private_settings.py ==> http://d.hatena.ne.jp/va009039/20130809/p1を参考に追加。
- 達成目標
  - ハードウェアFPU = “有効”設定でmbed SDKライブラリをNUCLEO-F401RE向けにビルドする。
- 課題
  - どうやれば「ハードウェアFPU = 有効」設定になるんだろう・・・？ SystemInit()から地道に遡る。

mbed SDKフルビルドにチャレンジ

- 編集が必要なファイル：
  - mbed-master/workspace_tools/targets.py
- [workspace_tools/targets.py]:
  - `class NUCLEO_F401RE(Targets):`
    - …
    - `self.core = “Cortex-M4” ==> “Cortex-M4F”`に変更
  - この変更がworkspace_tools/toolchains/__init__.pyに影響する。
- [workspace_tools/toolchains/__init__.py]:
  - `class mbedToolchain:`
    - `...`
    - `CORTEX_SYMBOLS = {`
      - …
      - `"Cortex-M4"  : ["__CORTEX_M4", "ARM_MATH_CM4"],`
      - `"Cortex-M4F" : ["__CORTEX_M4", "ARM_MATH_CM4", "__FPU_PRESENT=1"],`
    - `}`

mbed SDKフルビルドにチャレンジ

- ビルド。
  - `> cd …¥mbed-master¥workspace_tools`
  - `> python build.py -t uARM -m NUCLEO_F401RE`
- ビルドしたmbedがmbed-master/build/mbedに出力される。
- オンラインコンパイラからエクスポートしたmbed/をmbed-master/build/mbed/で置き換えて、uVisionのハードウェアFPU設定＝”有効”にしてNucleo_blink_ledプロジェクトをリビルド。
- うまく動いた・・・。

対応エクスポート形式

(platform example)	NUCLEO-F401RE	FRDM-KL05Z	mbed LPC1768
Keil uVision 4	OK	OK	OK
LPCXpresso	N/A	N/A	OK
GCC CodeSourcery	NG	NG	OK
GCC (GNU)	NG	OK	OK
IAR Systems	NG	NG	OK
CooCox IDE	NG	OK	NG
Codered	N/A	N/A?	OK
mbed Online IDE	OK	OK	OK
ZIP Archive	OK	OK	OK

うーん・・・まずいなあ・・・・

対応エクスポート形式

(platform example)	NUCLEO-F401RE	FRDM-KL05Z	mbed LPC1768
Keil uVision 4	OK	OK	OK
LPCXpresso	N/A	N/A	OK
GCC (CodeSourcery)	NG	NG	OK
GCC (GNU)	NG ==> OK	⟷ OK	OK
IAR Systems	NG	NG	OK
CooCox IDE	NG ==> OK	⟷ OK	NG
Codered	N/A	N/A?	OK
mbed Online IDE	OK	OK	OK
ZIP Archive	OK	OK	OK

達成目標

無償IDEが使えた方がいいだろうなあ。ターゲットはここだな・・・。

無償IDE(CoIDE)への対応にチャレンジ

- CoIDEには下記の外部ツールチェーンを使ってみます。
  - GNU Tools for ARM Embedded Processors (GCC_ARM)
    - そのためCoIDEプロジェクトでこのツールチェーンのコンパイル・リンクオプションを設定することになります。Makefileがあれば容易。
- 必要な作業：
  - workspace_tools/export/coide.pyを編集。
    - workspace_tools/export/gccarm.pyも編集。
  - workspace_tools/export/coide_nucleo_f401re.coproj.tmplを追加。
  - gcc_arm_disco_f407vg.tmplが既にあるので、これをもとにgcc_arm_nucleo_f401re.tmplをつくる。
  - GCC_ARM形式でエクスポートしてMakefileを得る。
  - Makefileでコンパイル・リンクオプションを把握する。
  - 既にある.coproj.tmplをベースにして、XMLコードをNUCLEO-F401RE向けに調整する。

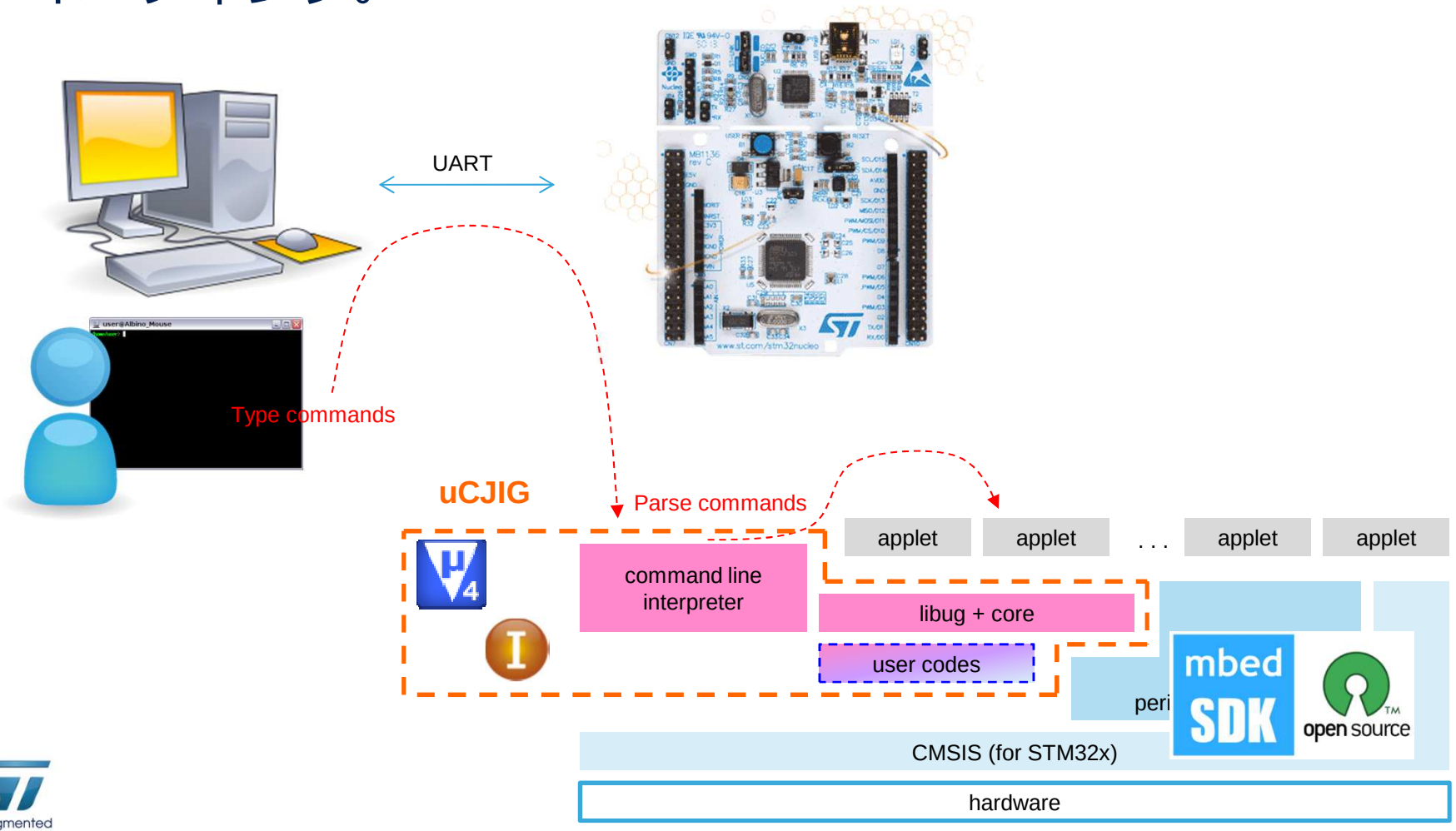
life.augmented

無償IDE(CoIDE)への対応にチャレンジ

- エクスポート。
  - Python jinja2パッケージが必要。インストール後、project.py。
    - `> cd …¥mbed-master¥workspace_tools`
    - `> python project.py -m NUCLEO_F401RE -p 38 -i coide`
  - -p 38: テスト番号38 = [38] MBED_10: Hello World
  - main()実行時に”Hello World”をシリアル出力した後、LED点滅。
- CoIDE v1.7.6をインストール。外部ツールチェーンのbin/ディレクトリパスをツールに理解させる。
- CoIDEからターゲットへの接続時にEclipse環境で必要なオンチップデバッガのOpen-OCD不要。
  - CoIDE内蔵デバッガがST-Link & J-Linkに対応している。

いつかやりたいこと

- サポート業務で使っているソフトウェア治具のmbedへのポーティング。



以上です。