ARM® Cortex®-M
mbed™ SDK and HDK deep-dive

アーム株式会社
スタッフアプリケーションエンジニア
渡會豊政

内容

- ARM Cortex-M ファミリの概要
- ARM Cortex-M アーキテクチャのポイント
  - 命令セット
  - 浮動小数点
  - 例外、割り込み
  - メモリ管理
  - 低消費電力機能
  - デバッグ機能
- 効率的なプログラミングのアプローチ
  - サイクル数測定方法
  - 応答サイクル数
  - 処理能力のゆとり
- mbed の特徴
- ハードウェアプラットフォームと HDK
- mbed SDK ライブラリ
- クラウド開発環境
- 最近のアップデート
- ポーティング方法

ARM

ARM Cortex-M ファミリの概要

ARM

増え続けるマイコンへの要望

開発期間

32-bit

16-bit

8-bit

```
bit_value[34] = hour10.1;

no_of_ones = 0;
for (i=29; i<35; i++)
   {
   no_of_ones = no_of_ones + bit
   }
bit_value[35] = no_of_ones%2;
bit_value[36] = day1.0;
bit_value[37] = day1.1;
bit_value[38] = day1.2;
bit_value[39] = day1.3;
bit_value[40] = day10.0;
```

通信

Bluetooth®

ZigBee™

安全性

表示

+
-
×
÷

消費電力

$$\sqrt{\log_n}$$

$$\int_x^n \sin$$

効率は重要

制御

ARM

Cortex-Mプロセッサ

Cortex-M0

8/16ビット
アプリケーションに対応

最小コスト

ARMv6-M

0.9 DMIPS/MHz

Cortex-M0+

8/16ビット
アプリケーションに対応

最小電力

卓越したエネルギー効率

ARMv6-M

0.95 DMIPS/MHz

Cortex-M3

16/32ビット
アプリケーションに対応

性能

効率

ARMv7-M

1.25 DMIPS/MHz

Cortex-M4

32ビット/DSC
アプリケーションに対応

MCUにDSP機能を付加
SIMD
浮動小数点演算

ARMv7E-M

1.25 DMIPS/MHz

Cortex-M共通

エネルギー効率　コンフィギュラブル　確定的動作
使いやすさ　小さいシリコンサイズ

市場での実績

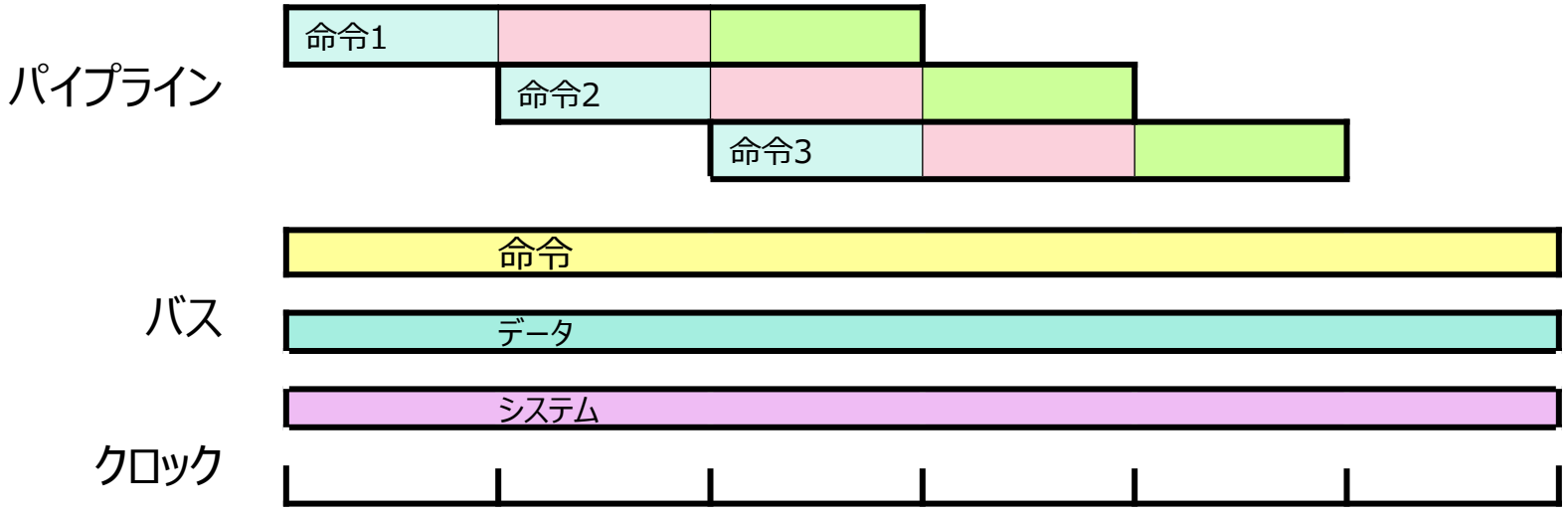
ARM

ARM Cortex-M　アーキテクチャのポイント

ARM

Cortex-M アーキテクチャ

- マイコンで使いやすいアーキテクチャ
  - 32ビットRISC
  - プロセッサアーキテクチャ … ARMv6-M, ARMv7-M (ARMv7E-M)
  - バスアーキテクチャ … AMBA®3 AHB-Lite
  - メモリ保護アーキテクチャ … PMSA
  - デバッグアーキテクチャ … CoreSight™
- 効率的なRISC動作 … パイプライン、バス構成、命令セット



レジスタセット

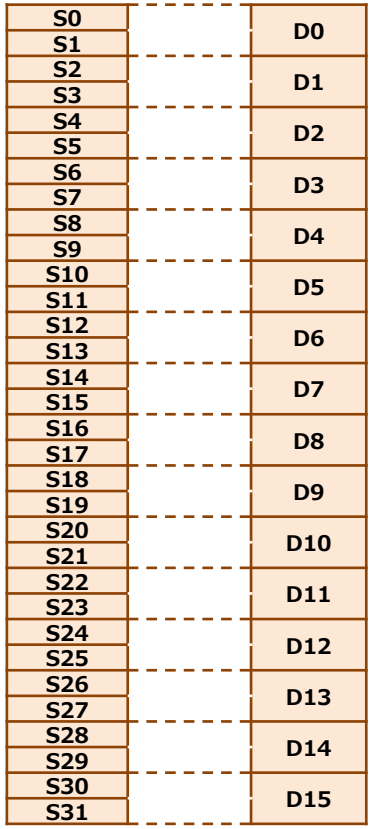
- 汎用レジスタセット

R0	第1引数 / 返却値
R1	第2引数 / 返却値
R2	第3引数
R3	第4引数
R4	変数用
R5	変数用
R6	変数用
R7	変数用
R8	変数用
R9	変数用
R10	変数用
R11	変数用
R12	変数用
R13(SP)	スタックポインタ
R14(LR)	リンクレジスタ
R15(PC)	プログラムカウンタ

xPSR	プログラム・ステータスレジスタ

ARMアーキテクチャプロシージャコール標準(AAPCS)により
レジスタの用途は定義

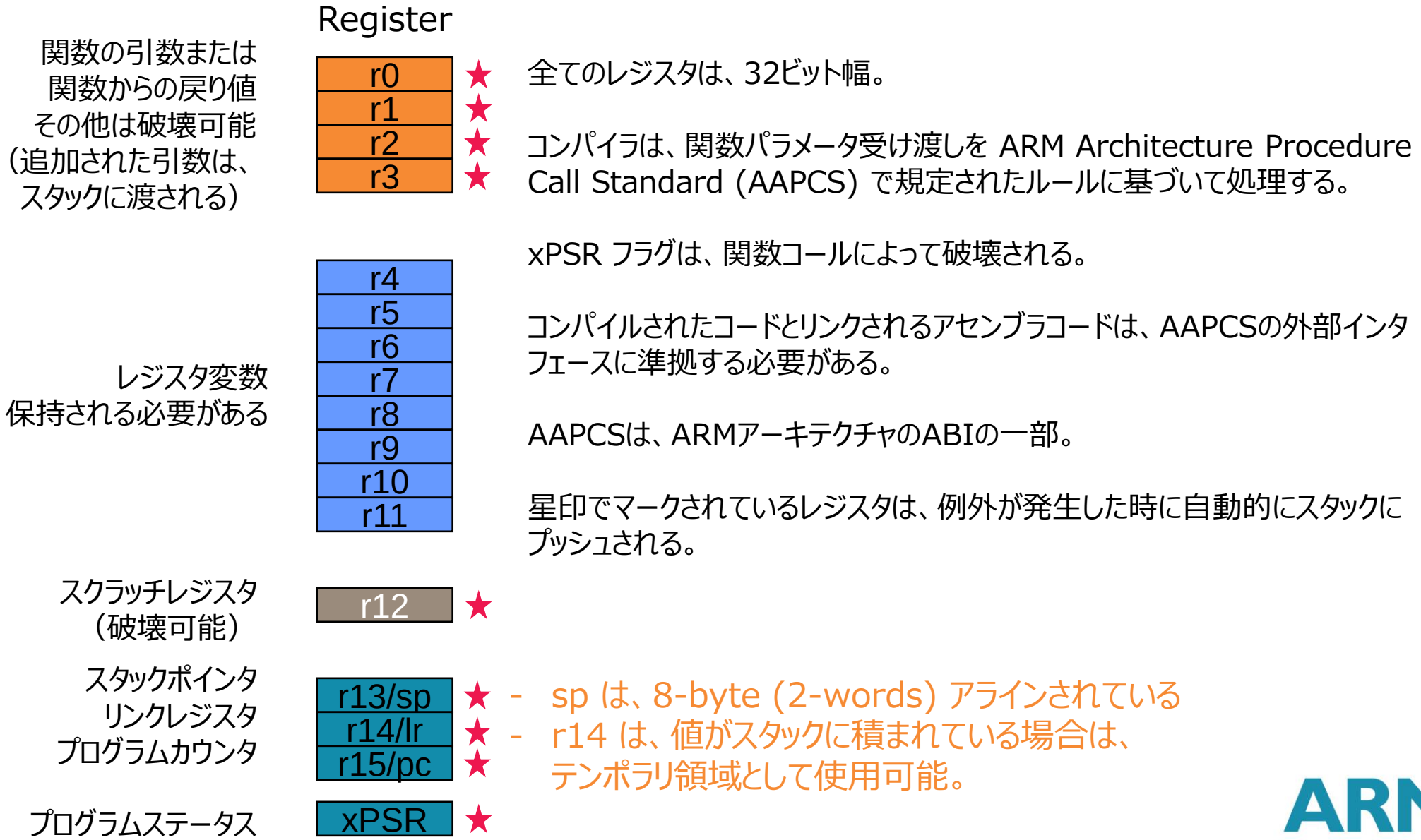
- Cortex-M4 FPU付きの浮動小数点レジスタ



- S0 – S31はそれぞれ32ビット
  <単精度浮動小数データ>
- D0 – D15はそれぞれ64ビット
  <倍精度浮動小数データ>
  S0-S1がD0と見える
- 浮動小数点レジスタは汎用レジスタとスタックの間でデータ転送ができる
※Cortex-M4の命令は単精度のみ

ARM

レジスタの利用方法（１）



全てのレジスタは、32ビット幅。

コンパイラは、関数パラメータ受け渡しを ARM Architecture Procedure Call Standard (AAPCS) で規定されたルールに基づいて処理する。

xPSR フラグは、関数コールによって破壊される。

コンパイルされたコードとリンクされるアセンブラコードは、AAPCSの外部インタフェースに準拠する必要がある。

AAPCSは、ARMアーキテクチャのABIの一部。

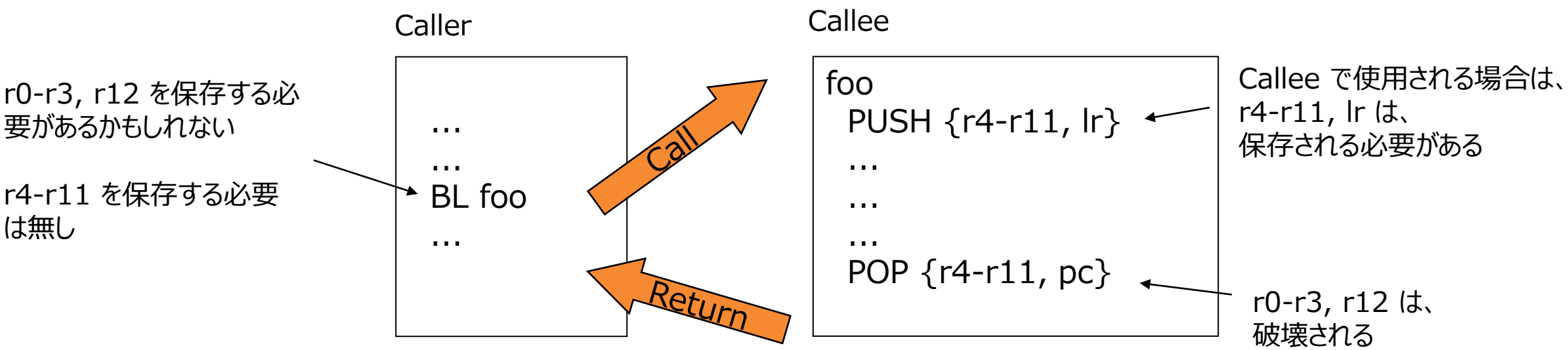
星印でマークされているレジスタは、例外が発生した時に自動的にスタックにプッシュされる。

- sp は、8-byte (2-words) アラインされている
- r14 は、値がスタックに積まれている場合は、テンポラリ領域として使用可能。

ARM

レジスタの利用方法（２）

パラメータは、 r0-r3 で渡される



関数からの戻り値が int/short/char の場合は、r0 が使用され、long/double の場合は、r0 と r1 が使用される

AAPCS – Procedure Call Standard for ARM Architecture

ARM

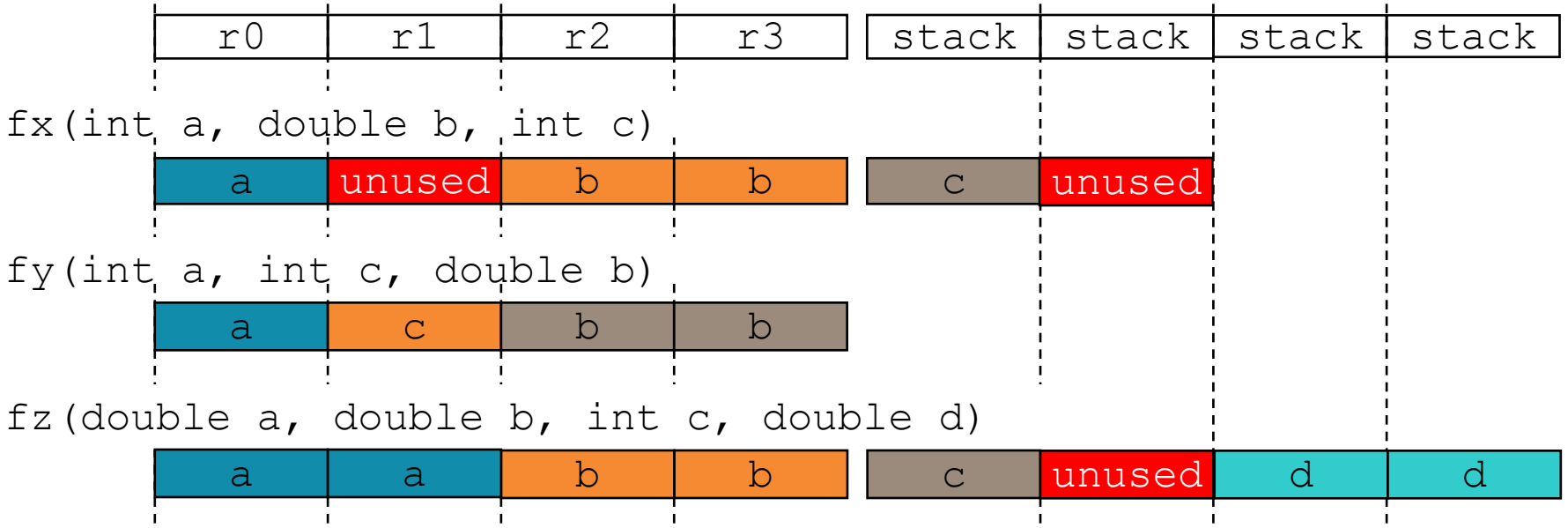
関数パラメータ（引数）について（１）

- 最初の４つのワードサイズ引数は、r0-r3 で渡される（高速で効率的）
  - 小さいサイズの型の引数でも１つのレジスタが使用される（short, char等）
  - ワードサイズより大きな引数では複数のレジスタが使用される（64ビット型については後述）
  - 詳細は、AAPCS を参照
- さらに追加の引数が必用な場合、5番目以降の引数はスタック経由で渡される
  - 追加の命令とメモリアクセスが必用
- …なので、常に関数の引数を4個以下にする事を考えて下さい
  - それが無理であれば、使う頻度の高い引数を最初の4番目以内に配置させる
  - 引数が構造体であれば、実体ではなくてそのポインタを渡す

- C++では1番目の引数は、メンバ関数へのthisポインタが必ず渡されるので、レジスタで渡される引数は3個になる

関数パラメータ（引数）について（２）

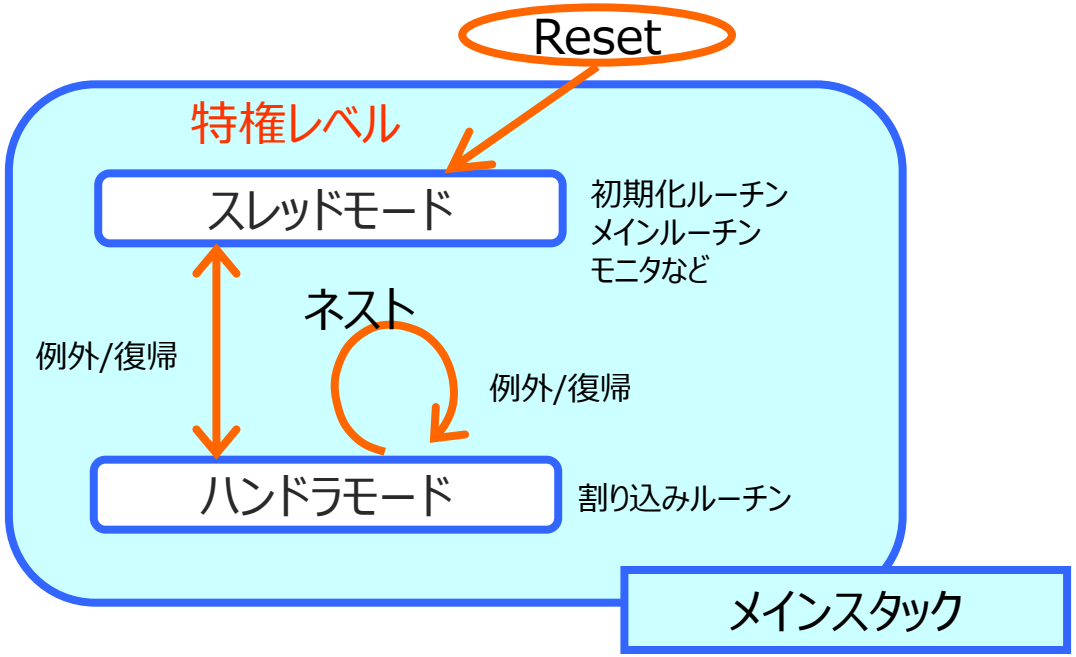
- AAPCSでの64ビット型のルール
  - 64ビット型は、8-byteアラインされる必要があります
  - 関数の64ビット型引数は、偶数＋それに続く奇数レジスタを使って渡される必要があります
  - （例： r0+r1 または r2+r3）または、8バイトアラインされたスタック上の領域
- 引数が最適化されない順番で指定された場合、レジスタとスタックが無駄遣いされます



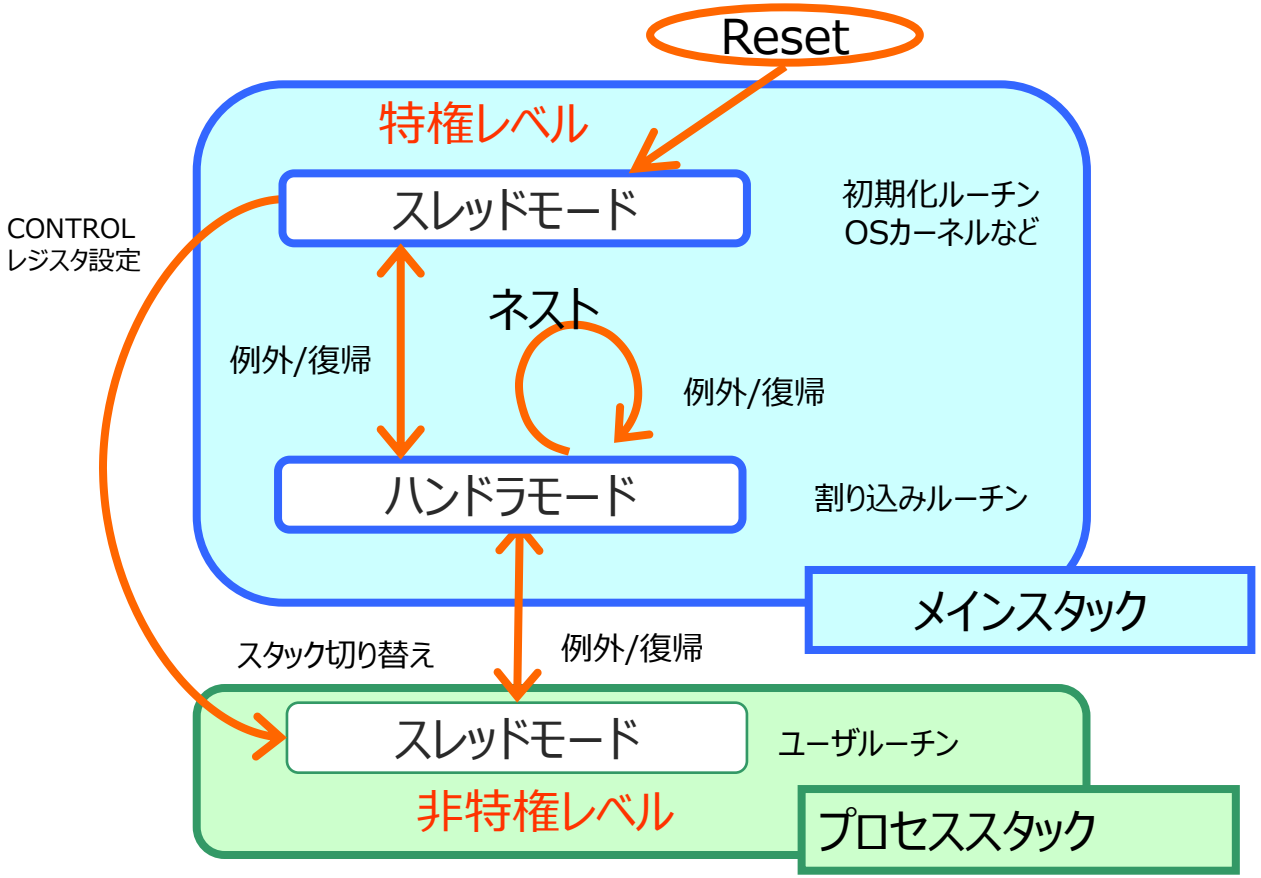
ARM

実行モード

- 基本的なシステム
  - 一つのスタックでシンプルな構成

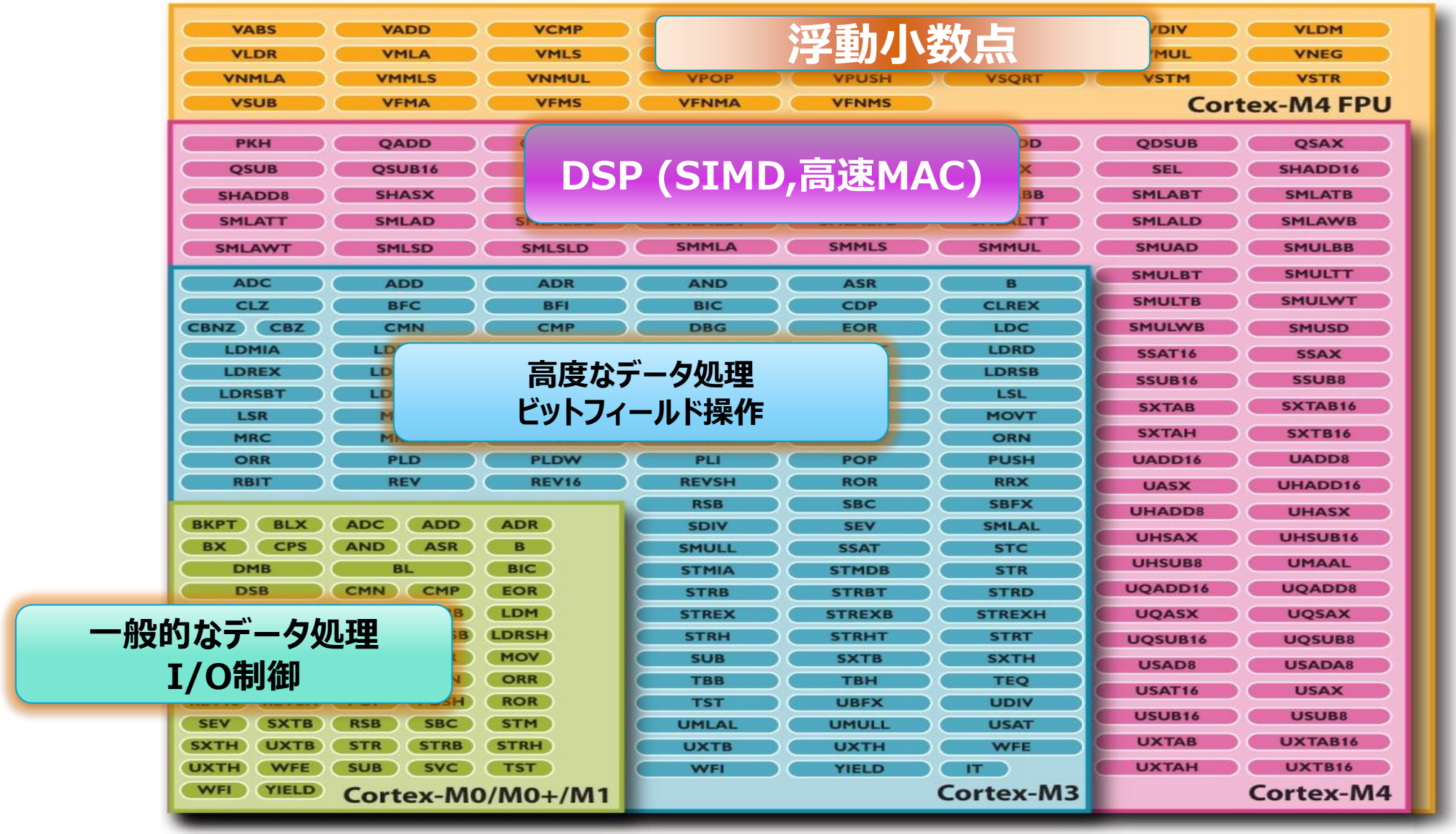


- RTOSを使うシステム
  - OSとユーザルーチンで特権レベルを変え、MPUによりシステムの保護を行うことができる
  - スタックを分離することができる



ARM

スケーラブルな命令セット



ARM

命令セット全体

命令分類	命令
データ処理命令	**ADC**, **ADD**, **ADR**, **AND**, **BIC**, **CMN**, **CMP**, **EOR**, **MOV**, **MVN**, ORN, **ORR**, **RSB**, **SBC**, **SUB**, TEQ, **TST**, ASR, **LSL**, **LSR**, **ROR**, RRX, MLA, MLS, **MUL**, SMLAL, SMULL, UMLAL, UMULL, SSAT, USAT, **SXTB**, **SXTH**, **UXTB**, **UXTH**, SDIV, UDIV, BFC, BFI, CLZ, MOVT, RBIT, **REV**, **REV16**, **REVSH**, SBFX, UBFX
分岐命令、実行制御命令	**B**, **BL**, **BLX**, **BX**, TBB, TBH, BEQ, CBZ, CBNZ, IT
メモリアクセス命令	**LDR**, **STR**, **LDM**, **STM**, **PUSH**, **POP**, LDREX, STREX
例外生成命令	**SVC**, **BKPT**
特殊レジスタアクセス命令	**MRS**, **MSR**, **CPSIE**, **CPSID**
メモリバリア命令	**DMB**, **DSB**, **ISB**
同期命令、その他	**SEV**, **WFI**, **WFE**, **NOP**
DSP命令(Cortex-M4)	PKHBT, PKHTB, QADD, QASX, QDSUB, QSAX, QSUB, SASX, SEL, SHASX, SHSAX, SMLABB, SMLABT, SMLAD, SMLADX, SMLALBB, SMLALBT, SMLALD, SMLALTB, SMLALTT, SMLATB, SMATT, SMLAWB, SMLAWT, SMLSD, SMLSLD, SMMLA, SMMLAR, SMMLS, SMMLSR, SMMUL, SMMULR, SMUAD, SMULBB, SMULBT, SMULTB, SMULTT, SMULWB, SMULWT, SMUSD, SSAX, SXTAB, SXTAH, SXTB, SXTH, UASX, UHASX, UHSAX, UMAAL, UQASX, UQSAX, USAX, UXTAB, UXTAH
浮動小数点命令(Cortex-M4 FPU付き)	VABS,VADD,VCMP,VCMPE,VCMP, VCMPE,VCVT,VCVTR,VCVTB, VCVTT,VFMA,VFNMA,VFMS,VFNMS,VLDM,VLDR,VLMA,VLMS,VMOV,VMRS,VMSR,VMUL,VNEG,VNMLA,VNMLS,VNMUL,VPOP,VPUSH,VSQRT,VSTM,VSTR,VSUB

※**太字**はCortex-M0/M0+の命令

ARM

フレキシブルなオペランド

- イミディエートオフセット
  - 例) LDR r0, [r1, #8]　… 8ビットのオフセットが使える,PCベースのみ12ビットのオフセットが使える
- プリインデックス
  - 例) STR r0, [r1, #12]! … 先にr1に12を加え、r1のアドレスにr0をストア
- ポストインデックス(M3/M4)
  - 例) STR r0, [r1], #12 … r1のアドレスにr0をストアした後、r1に12を加える
- フレキシブルな第2オペランド
  - 定数は8ビットでも特定の32ビット値に拡張できる
  - 以下の定数は0x**XY**のオペランドの8ビットで表現できる
  - 0x00**XY**00**XY**、　0x**XY**00**XY**00、　0x**XYXYXYXY**
  - オペランドのレジスタをシフトして利用できる
  - 例) “r0, {,LSL #4}” … オペランドのr0を4ビット左シフトしたものが演算される

ARM

アドレッシングモード

- アドレスオフセット(例　LDR　Rn, [Rm, #offset])
  - M0/M0+: メモリアクセス命令のオフセットは**5ビット**
  - M3/M4: メモリアクセス命令のオフセットは**5ビット**または**12ビット**

- データアクセスでの複合したアドレスの扱い
  - 例) N番目のワード配列へのアクセス … N番目のアドレスはN x 4+ベースアドレス

C言語	プロセッサ	命令	関数全体サイズ	サイクル数

int func(int *p, int num)
{
　return p[num];
}
	M0/M0+	LSLS　R1, R1, #2
		LDR　R0, [R0, R1]
		BX　LR
			3命令
			6バイト
				3+分岐

	M3/M4	LDR　R0, [R0, R1, LSL #2] BX　LR	2命令 6バイト	2+分岐

ARM

イミディエートデータ

- ARMv7-MではMOVW命令が使える

	命令	説明
M0/M0+	LDR　Rn, =0x123	AHB™上のリテラルデータ読み出しには2サイクル必要。LDR Rn, [PC,#リテラルoffset]と同じ
M3/M4	MOVW　Rn, #0x123	32ビット長命令を1サイクルで実行

- ARMv7-Mではイミディエートデータに対応しているものがある
  - 例) A = B | 0x3C … 8ビットの定数をシフトして得られる32ビットが使える

	命令	説明
M0/M0+	MOVS　Rm, #0x3C ORRS　Rn, Rn, Rm	2命令必要
M3/M4	ORRS　Rn, Rn, #0x3C	32ビット長命令を1サイクルで実行

ARM

分岐 – 効率のためには少ないほど良い

- 条件分岐
  - M0/M0+: +/- 254バイト
  - M3/M4: -1048576 から +1048574 バイト(20ビット) ;32ビット条件分岐命令の場合
- 無条件分岐
  - M0/M0+: +/- 2Kバイト
  - M3/M4: +/- 16MB ; 32ビット分岐命令の場合
- 間接分岐
  - レジスタの値のアドレスに分岐する(4GB空間,0xfffffff0-0xffffffffを除く)
  - 例) BX r0 … r0のアドレスに分岐
- その他
  - M3/M4は条件実行(IT, if-then), 比較分岐(CBZ, CBNZ)、テーブル分岐(TBB,TBH)が使える
- 分岐先は4バイト境界の方が効率が良い
  - 4バイト境界でない場合は32ビットフェッチした下位16ビットが破棄される
  - M0+は4バイト境界でない場合のみ16ビットで命令フェッチする

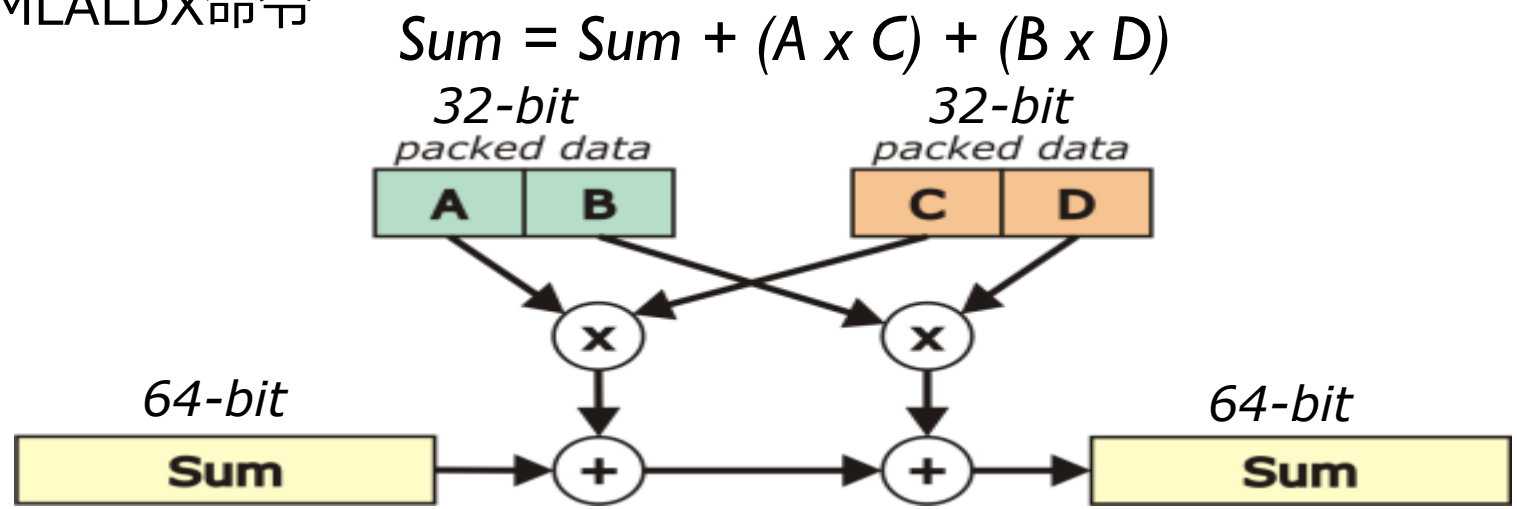
ARM

四則演算

- 加減算 – 1サイクル
- 乗算
  - M0/M0+ は 32b x 32b = 32b
  - M3/M4は 32b x 32b = 32b, 32b x 32b = 64b
- 除算
  - M0/M0+ : ソフトウエア(ライブラリ)で行う
  - M3/M4: 除算命令のハードウエアで行う(2～12サイクル)
- 積和 – 乗算結果を累積する
  - M3/M4: 32b x 32b + 32b = 32b
  - M4: 多くの積和演算の種類を1サイクルで実行
- 飽和演算 – 累積結果を一定の範囲に丸める
  - M3/M4 : SSAT, USAT命令
  - M4 : QADDx, QSUBx, QASX, QDADD, QDSUB, QSAX
  - 値が範囲内にあるかどうかの判定処理を省くことができる

M4: SIMD演算

- SIMD演算は同じ複数の演算を1サイクルで実行
  - SIMD = Single Instruction Multiple Data
  - 例）SMLALDX命令

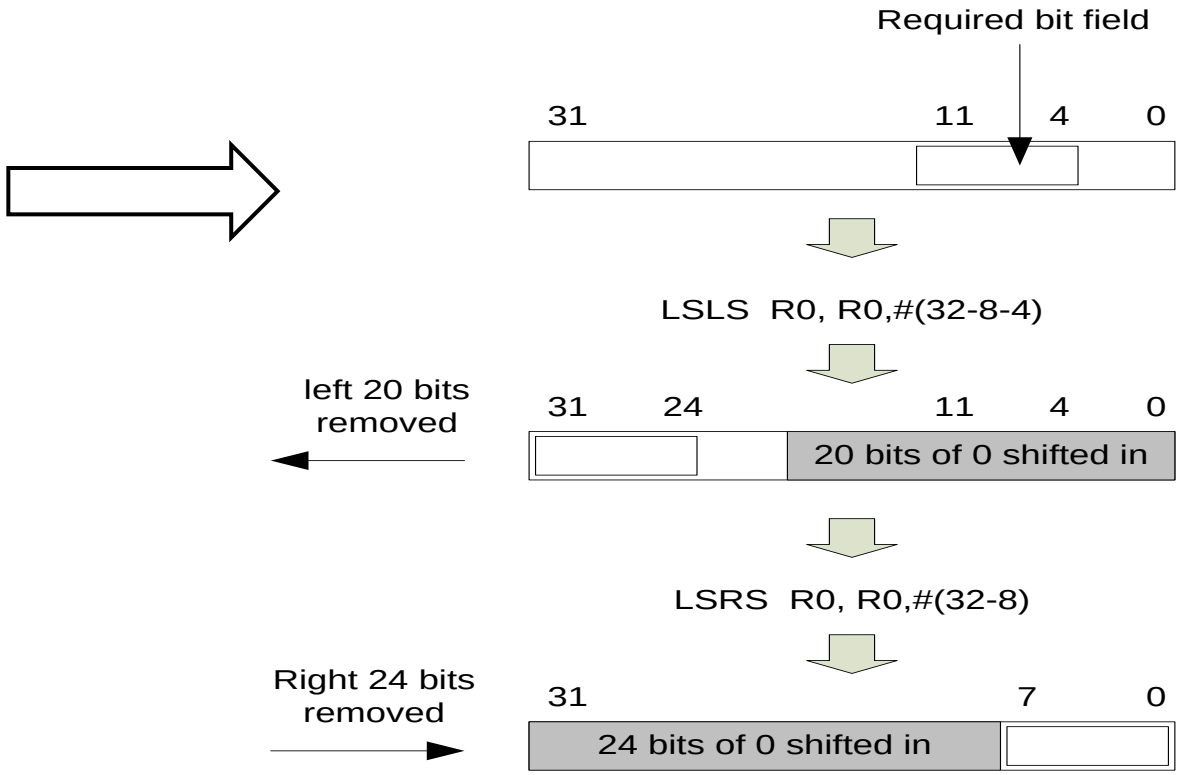


- SIMD演算はレジスタにパックされたデータに対して操作可能
  - メモリ転送命令を削減できる
- 40種類のSIMD演算のバリエーション

ARM

ビットフィールド

- ビットフィールドの抜き出し処理
  - M3/M4は1命令(UBFX, SBFX)
  - M0/M0+は複数命令で操作



- その他M3/M4で使える命令
  - ビットフィールドのクリア
  - ビットフィールドの挿入
  - ビットの反転
  - ゼロビットの数のカウント

ARM

M4: 単精度浮動小数点

- FPv4-SPアーキテクチャ … IEEE 754標準に準拠
- 浮動小数点演算もパイプライン処理される
- 高性能な単精度浮動小数点演算
  - 加算、減算、乗算、除算、積和、平方根
  - フューズドMAC – より高精度に積和演算
- 浮動小数点の効果
  - 浮動小数点FIRフィルタの例 – CMSIS-DSP：arm_fir_example_f32.c の測定結果

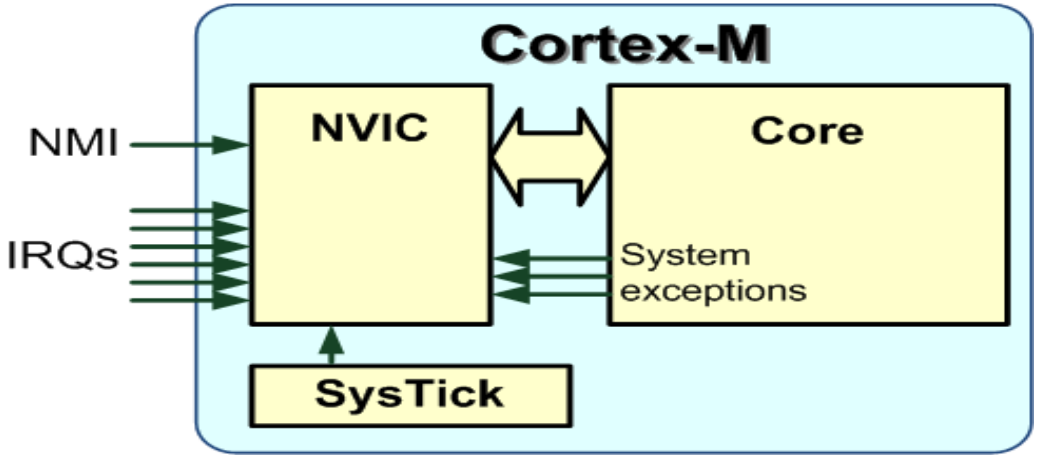
命令	サイクル数
加算/減算	1
除算	14
乗算	1
積和	3
フューズドMAC	3
平方根	14

	M0	M0+	M3	M4 FPU付き
サイクル数(比)	1319666(39.7)	1221087(36.7)	868627(26.1)	33269(1.0)

ARM

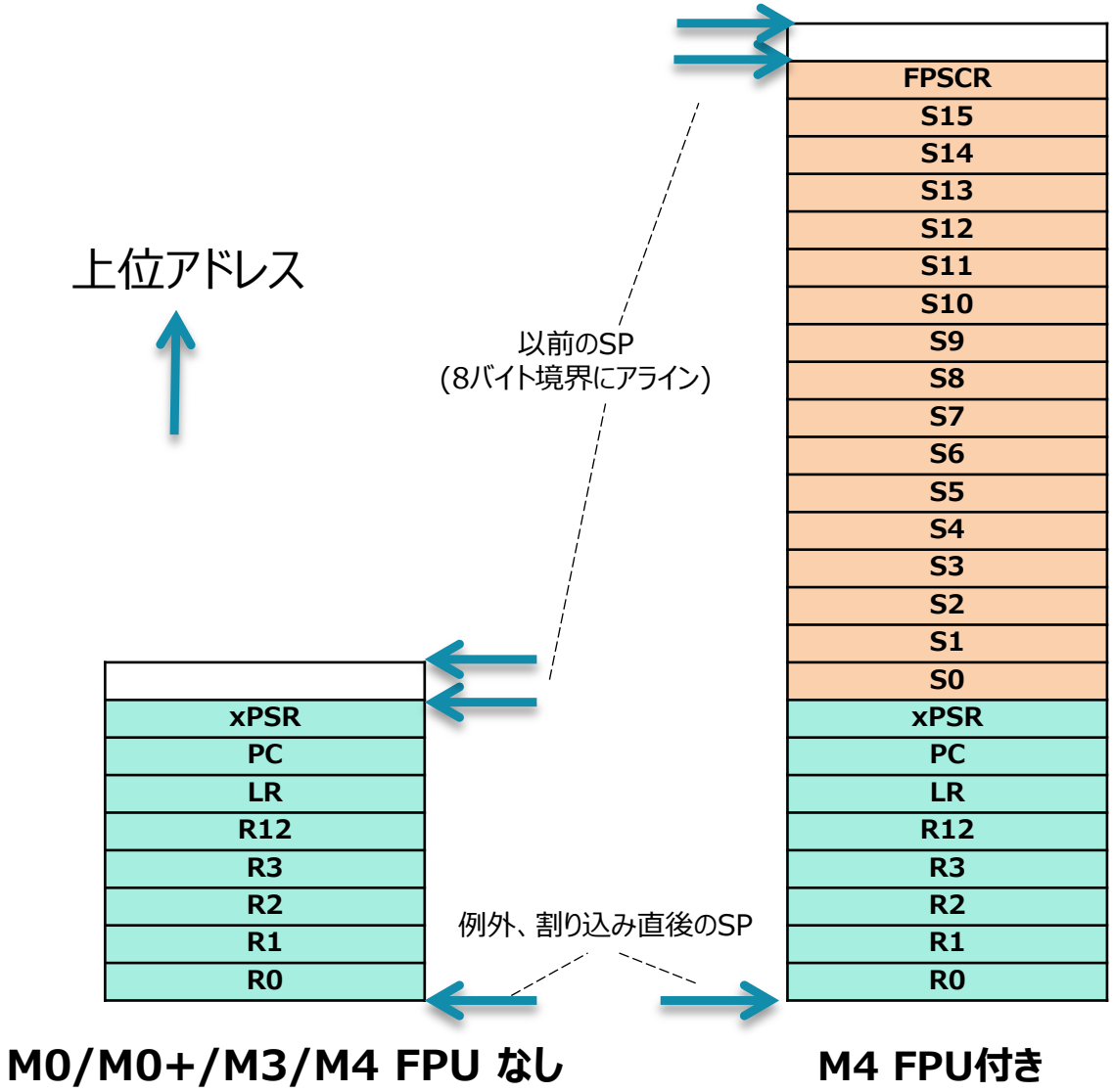
標準化された例外・割り込み管理

- 標準化された割込みコントローラ(NVIC)
  - ネスト可能なベクタ方式の割込みコントローラ
  - 基本的なベクタはCortex-Mシリーズで共通 – ベクタテーブルの内容はハンドラのアドレス
  - システムタイマ自体もコアに装備
- 割込み処理高速化の工夫
  - 割込み応答サイクル数は一定
  - レジスタの待避/復帰が自動的に行われる
  - ネスト時に不要な実行サイクルを動的に削除



ARM

例外・割り込み時に保存されるレジスタ

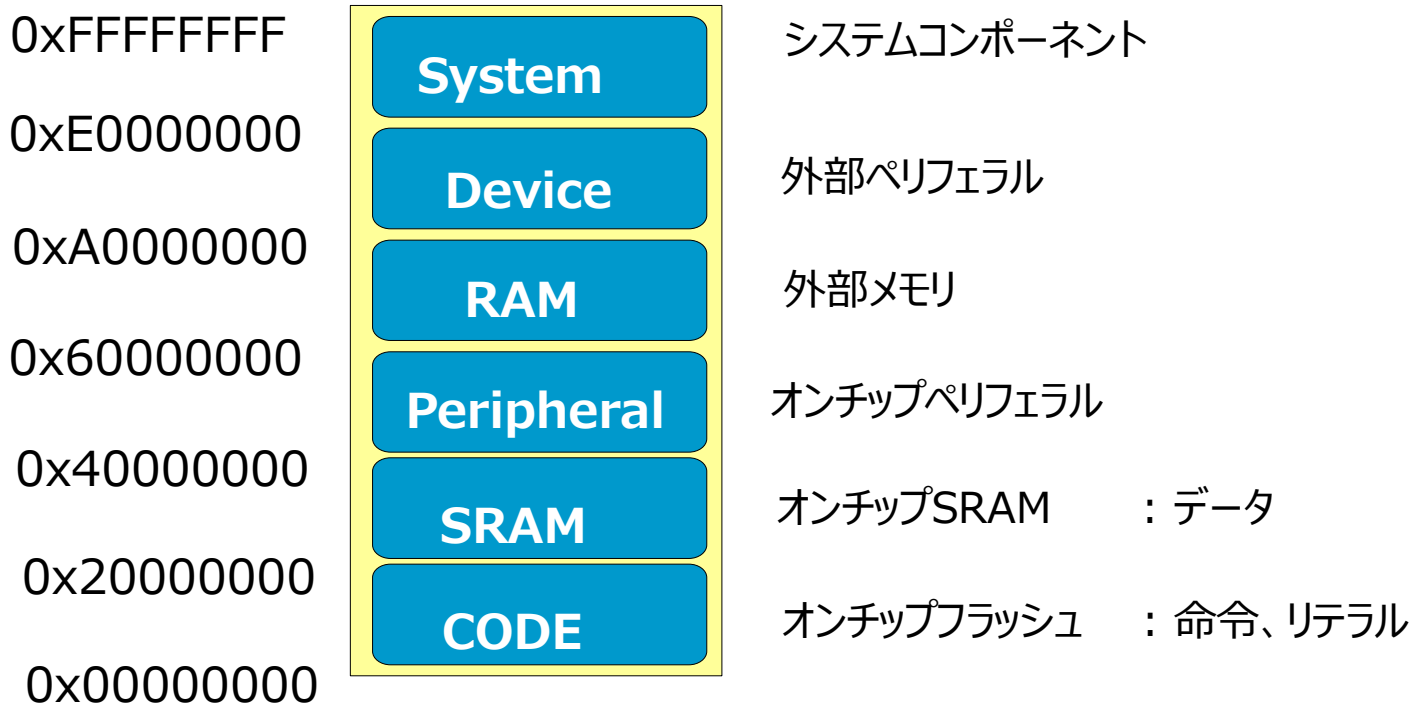


- Cortex-M4 FPU付きの場合の動作
  - CONTROL.FPCAがFPUが使われたかどうかを表す。このビットによって割り込み時のプロセッサの保存内容が異なる。
  - 0: 未使用 – 従来の8本のレジスタをスタックに保存
  - 1: 使用済 – 8本に加えてS0-S15,FPSCRをスタックに保存
  - S16-S32は通常のC言語のレジスタ保存と復帰
- レイジースタック機能
  - 例外・割り込み時にはFPU保存のスタック領域だけ確保し、ハンドラが実際に使う時に初めてFPUレジスタの保存を行う機能
- 例外・割り込みでFPUを使わない場合
  - FPUのレジスタを保存しないように設定できる

ARM

メモリマップ

- 4GB リニアなメモリ空間



- シンプルな物理メモリだけの構成、Cortex-Mで共通
- M3/M4は命令フェッチとデータアクセスを同時に行うことで効率を高める
- 定められた用途を規定することにより移植性が向上

ARM

ビットバンド・エイリアス

- メモリやペリフェラルを効率良くビット操作するメモリシステムの機能

アドレスAにあるビットB(0～7)を操作する場合に対応するビットバンドエイリアスのアドレス：
ペリフェラルの場合: 0x42000000 + (A - 0x40000000) x 32 + B x 4
SRAMの場合: 0x22000000 + (A - 0x20000000) x 32 + B x 4

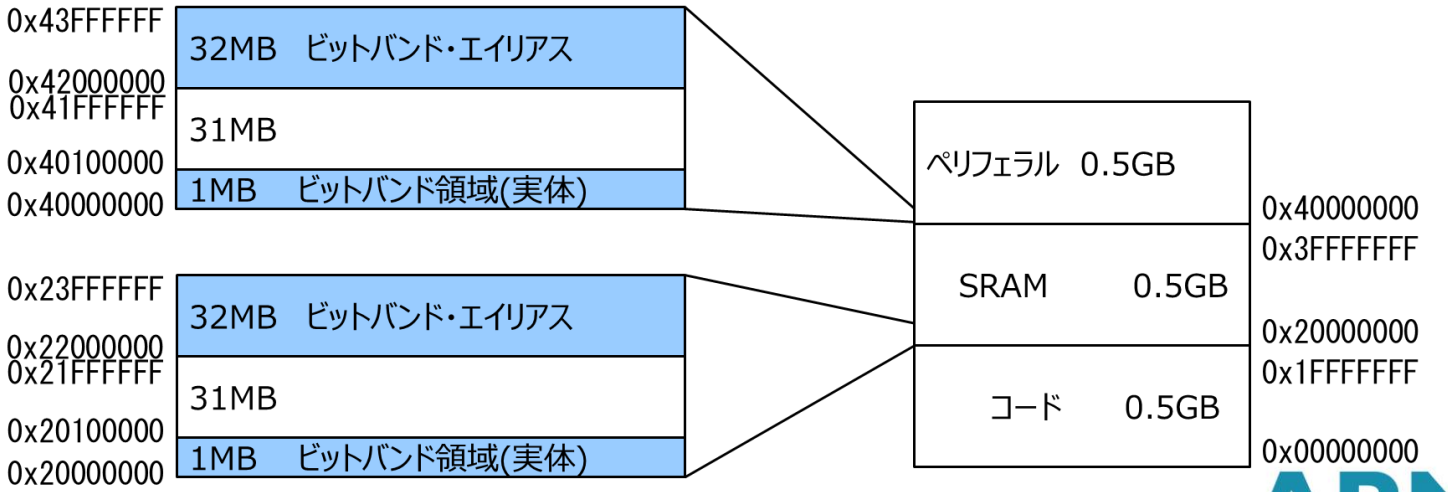
このアドレスにワード単位で0/1を書き込むことで対応するビットが操作される
(ハードウエアでRead-Modify-Writeが行われる)

通常処理：

```
x = x & 0xFFFFFFFE;
x = x | 0x00000001;
```

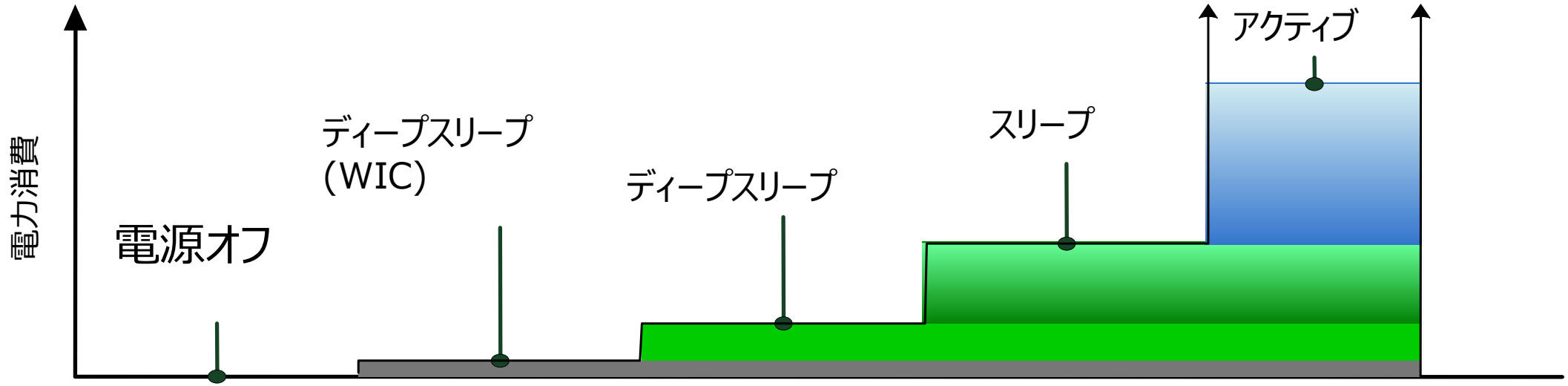
ビットバンド処理:

```
y = 0;
y = 1;
```



Cortex-Mのスリープモード

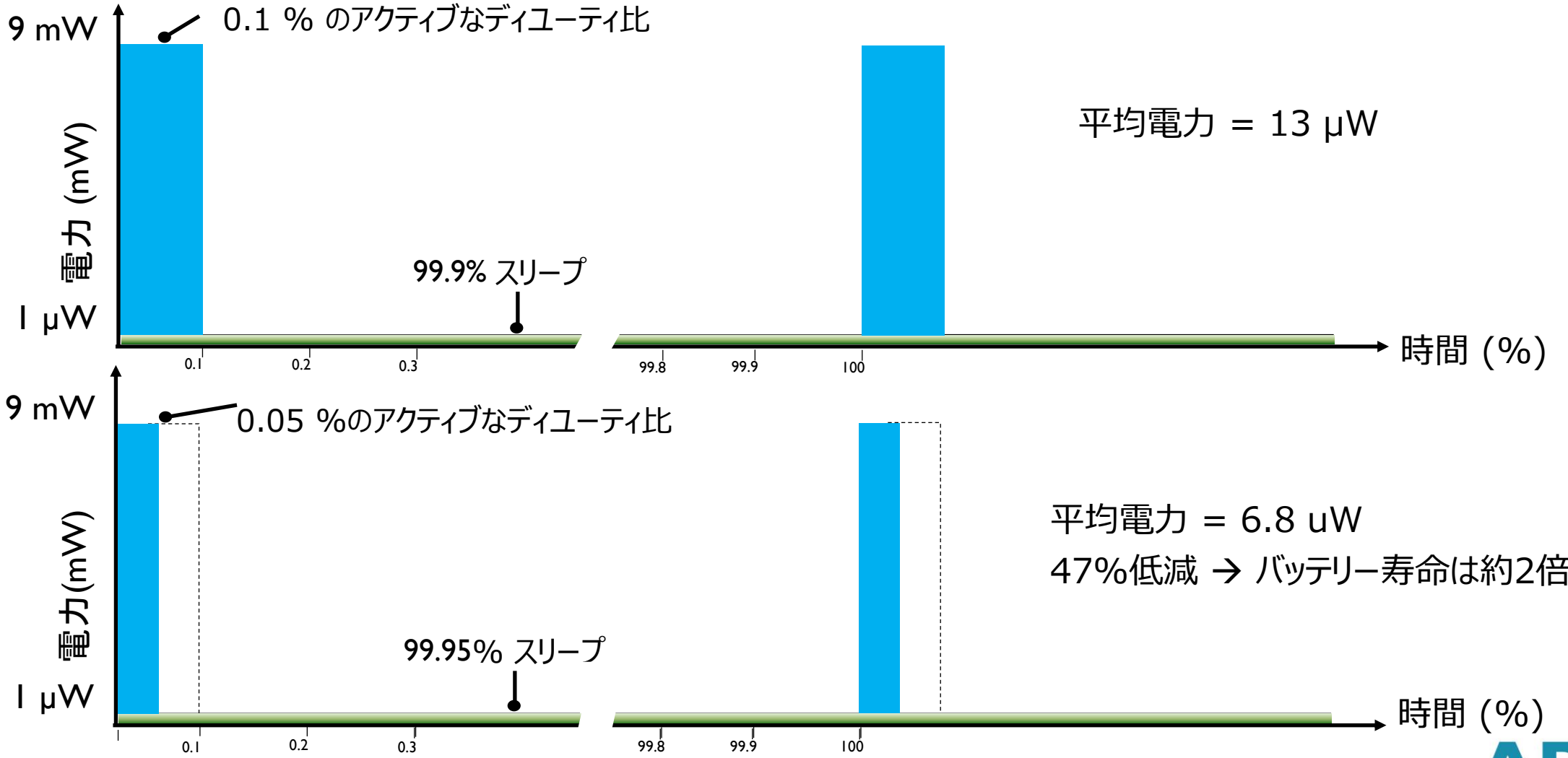
- アーキテクチャで定義されたスリープモード
  - スリープ / ディープスリープ　または WICを使ったディープスリープ(コアも低電圧で停止)
  - 割り込みから戻ったらプログラムに戻らず直接スリープモードに戻る設定も可能
- 低消費電力のために処理を極力短くしてスリープモードで待機させる



※電力の大小関係のイメージを表しています

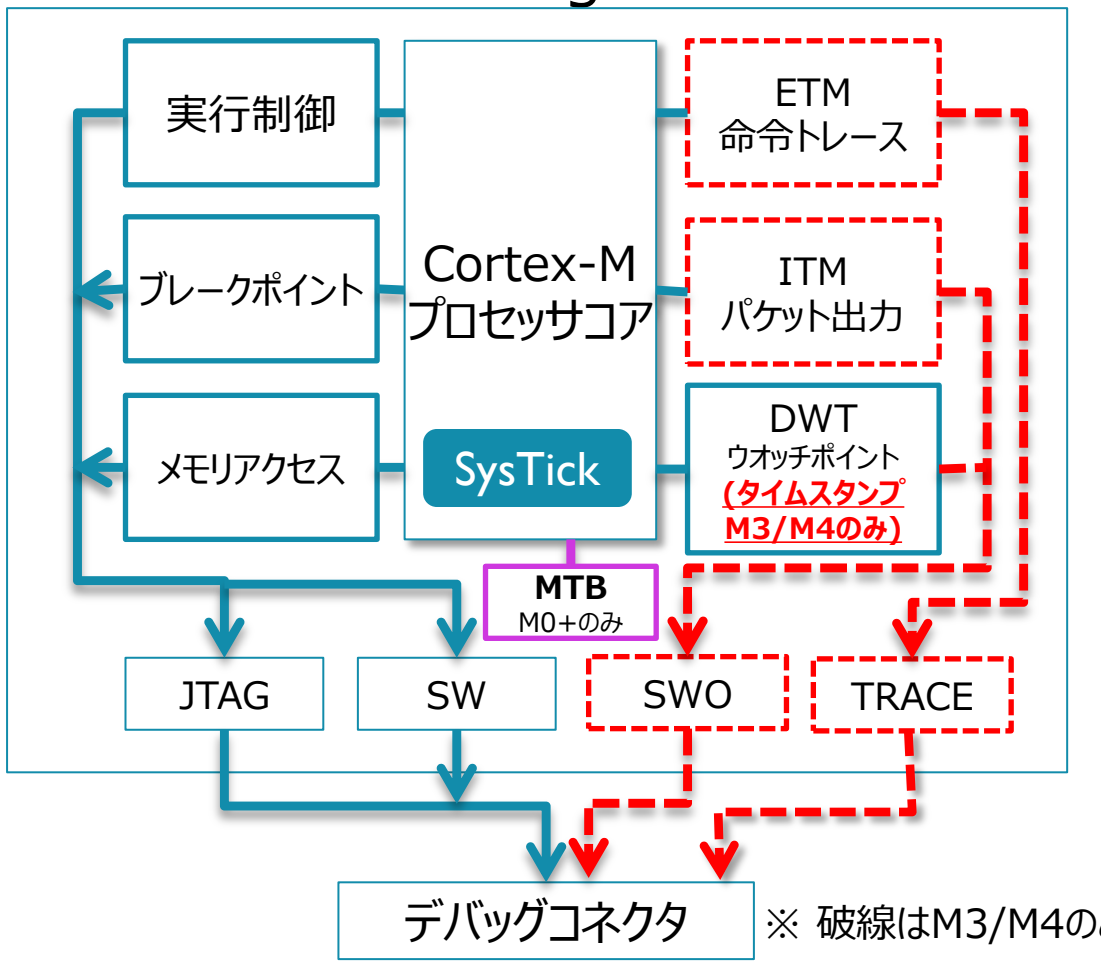
短い実行サイクルほど長いバッテリー寿命

一定周期でアクティブ時: 9mW、スリープ時: 1uWと仮定した場合の考え方　: 処理の実行サイクル数が半分になったとすると …



デバッグ – CoreSight™アーキテクチャ

- デバイス上のCoreSightのコンポーネント



- デバッガで利用できる機能

デバッガの機能	コンポーネント	M0	M0+	M3	M4
ダウンロード/メモリ操作 実行/停止/ステップ実行	実行制御	○	○	○	○
実行ブレーク アクセスブレーク	ブレークポイント DWT	○	○	○	○
ベクタキャプチャ PCサンプリング	DWT	○	○	○	○
データアクセストレース (タイムスタンプ付き)	DWT ITM			○	○
デバッガ経由のprintf出力	ITM			○	○
命令トレース(タイムスタンプ付き) コードカバレッジ、プロファイル	ETM			○	○
MTB命令トレース コードカバレッジ、プロファイル	MTB メモリアクセス		○		

ARM

ARMv6-M と ARMv7-M の違い

- インストラクションセット
  - ARMv6-M は、16-bit Thumb命令と幾つかの追加の命令セット
  - LDREX/STREX 命令やアトミックなスワップ命令は未サポート
  - STM/LDM命令での割り込み継続は未サポート
  - ARMv6-M は、非アラインワードまたはハーフワードアクセスには未対応
- モードと特権
  - 特権はオプションで拡張可能
  - 上記オプションを使用しない場合 ARMv6-M コアは、常に特権で動作する（例：Cortex-M0）
- 例外処理
  - ベクタテーブルのベースアドレスは設定不可能（実装定義）
  - いくつかの優先レベルのみ対応、シンプルな優先度スキーム (FAULTMASK/BASEPRI なし)
  - フォールトの種類は、HardFault のみ
  - 32 要因までの外部割り込み (ARMv7-M では 496 要因まで)
- ペリフェラル
  - SysTick は、オプション
- メモリ
  - スタックアラインメントは必須 (STKALIGN 有効に固定)
  - 縮小されたMPUの機能
- デバッグ
  - デバッグはホールトモードのみ対応（プログラム実行時のメモリ書き換え等は不可）

ARM

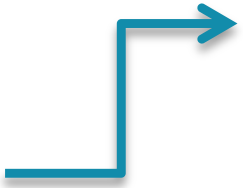
効率的なプログラミングのアプローチ

ARM

目的

- サイクル数からのアプローチ
  - 実行サイクル数をベースに客観的に把握、フィードバック　→ 少ないサイクル数 ＝ 高効率
- 実行サイクル数
  - 対象区間のサイクル数をどうやって測定するか

- 応答サイクル数
  - 割り込み処理の応答に何サイクルかかるか



- システムの余裕
  - システムにどれくらいの処理能力が残っているのか

実行サイクル数の測定方法

- アーキテクチャで用意された測定機能の活用
  - CoreSightアーキテクチャ … タイムスタンプ機能は使いやすい
    - DWT+ITM … データアクセス時のタイムスタンプ
    - ETM … 命令分岐時のタイムスタンプ、ストリーミングトレースにより長時間のトレース情報の蓄積が可能
  - カウンタ … SLEEPCNT(イベントカウンタ)、CYCCNT(タイムスタンプの元) … 32ビット
  - SysTick … 実行サイクル数の測定用にも使える。Cortex-M共通 … 24ビット
- Cortex-M0, M0+
  - SysTickでコアクロック精度での測定が可能
- Cortex-M3, M4
  - タイムスタンプの利用 … デバッガにより容易に利用できる
  - SysTickも使える

ARM

Cortex-M0/M0+の実行サイクル数測定

- SysTickをサイクル数測定に使う。プログラムの改変が必要

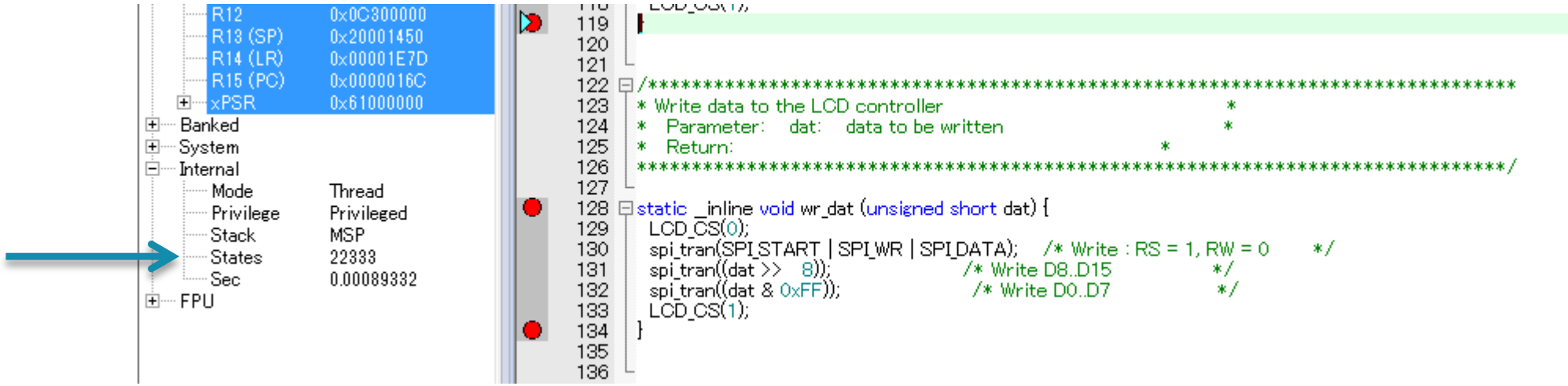
```
 // Systick レジスタ
int *STCSR = (int *)0xE000E010;
int *STRVR = (int *)0xE000E014;
int *STCVR = (int *)0xE000E018;
int  CNT;     // 実行サイクル数

func() {

    // Configure Systick
    *STRVR = 0xFFFFFF;  // リロード値に最大値を設定、Systickは24ビットのカウンタ
    *STCVR = 0;         // カウンタ値にリロード値を設定
    *STCSR = 5;         // コアクロックを選択、割り込みなし

        <測定箇所>

     CNT = *STCVR;          // 現在のカウント値を読み出し
     CNT = 0xFFFFFF – CNT – 4;  // 経過サイクル数演算、4はオーバヘッド、0x1000000 – 4 サイクルまで測定可能
                                                                    ＊コンパイラのコード生成によってオーバヘッドは検証が必要
}
```

- 実行後ブレークしてCNT値を表示する

ARM

Cortex-M3/M4の実行サイクル数測定

- uVisionの場合、Internal -> Statesがコア内部のCYCCNT値を表す
  - 測定開始箇所でブレークしてStatesの値を記録(1)
  - 測定終了箇所でブレークしてStatesの値を記録(2)
  - (2) – (1) で実行サイクル数が求められる



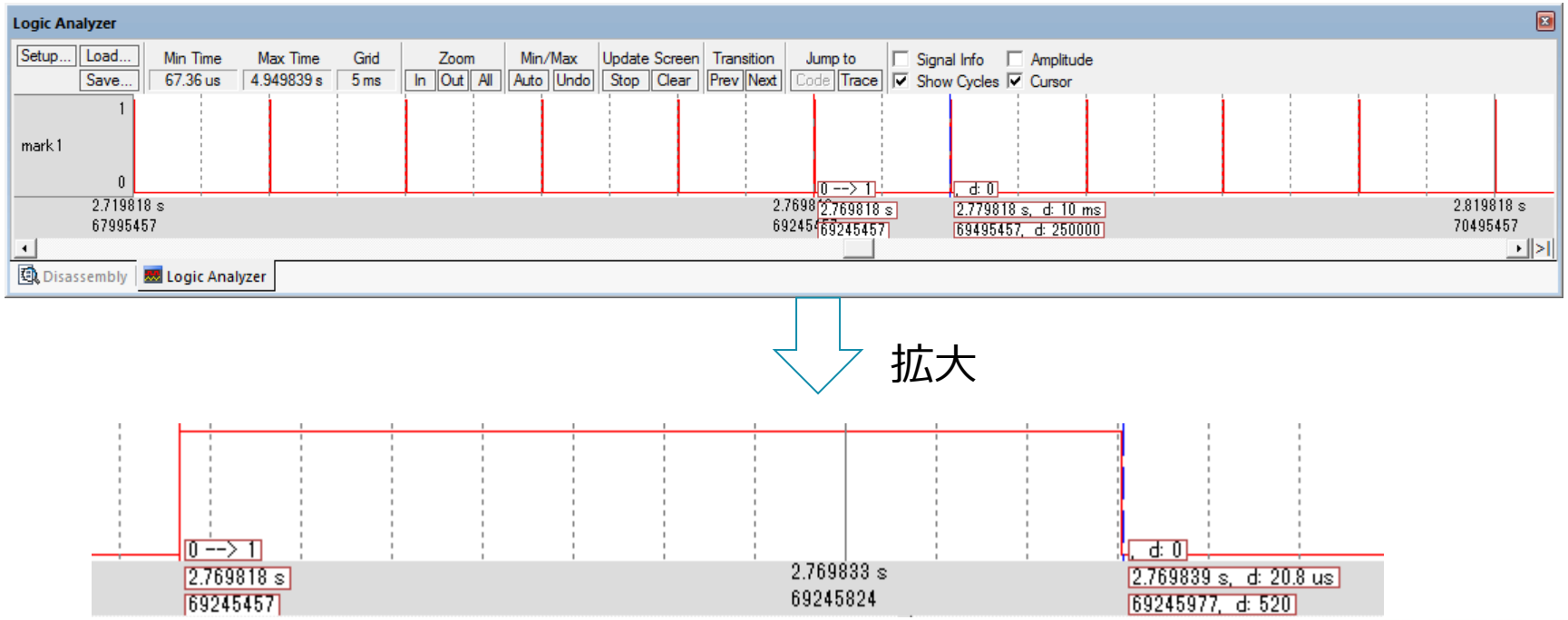
- 命令トレースのタイムスタンプ情報を使う方法も使える
- SysTickを使った測定も使える

ARM

データがアクセスされた時間のロジックアナライザ表示(M3/M4)

- uVisionではデータアクセストレースのタイミングを時間軸でグラフィカルに表示
  - ITM+DWTのタイムスタンプによりコアクロック精度の実行時間の表示ができる
  - 実行停止後も表示画面でカーソルで指定した区間のサイクル数を測定できる
- 特定の変数、ペリフェラル、測定用ダミー変数へのアクセスを波形として表示

```
#define      MARK_ON   1
#define      MARK_OFF  0
 uint32_t mark1=MARK_OFF;

void   SysTick_Handler(void) {
       int     i,s;

       mark1 = MARK_ON;

       <処理>

       mark1 = MARK_OFF;
}
```



応答サイクル数、ハンドラの処理サイクル数の効率的な測定

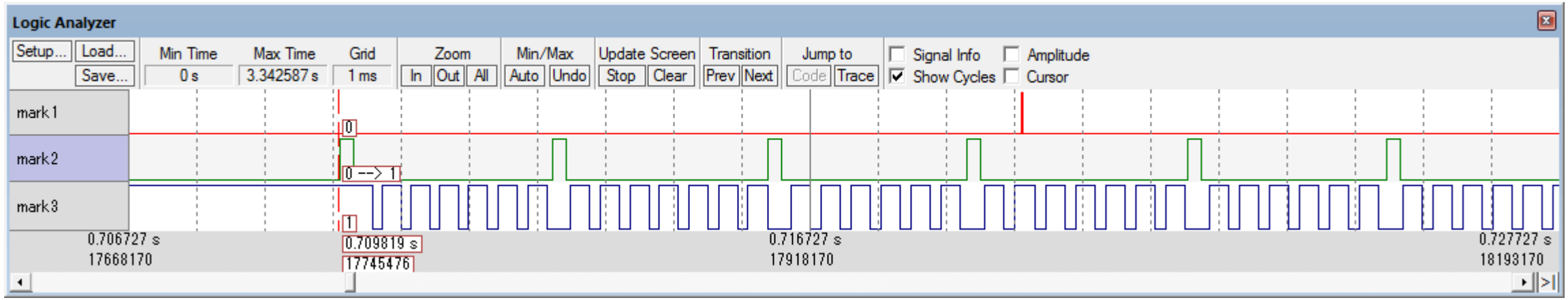
- 測定対象のハンドラをソフトウエア例外で発生するようにベクタテーブルを変更する方法
  - 例外ハンドラのオーバヘッドサイクル数は同じ
  - ソフトウエア例外命令からの必要な測定区間の実行サイクル数を測定する
  - 検証用に割り込みをシーケンシャルに発生させることができる

```
int    input_val;

func() {

    int        i;

    for (i=0; i<100; i++) {
        input_val = i;          // 必要に応じて割り込み時の条件を設定
        __asm { svc 0x00 } // ソフトウエア例外の発生
    }
}

void  Int20_Handler(void) // 本来の例外ハンドラ
{
        …
}
```


ARM

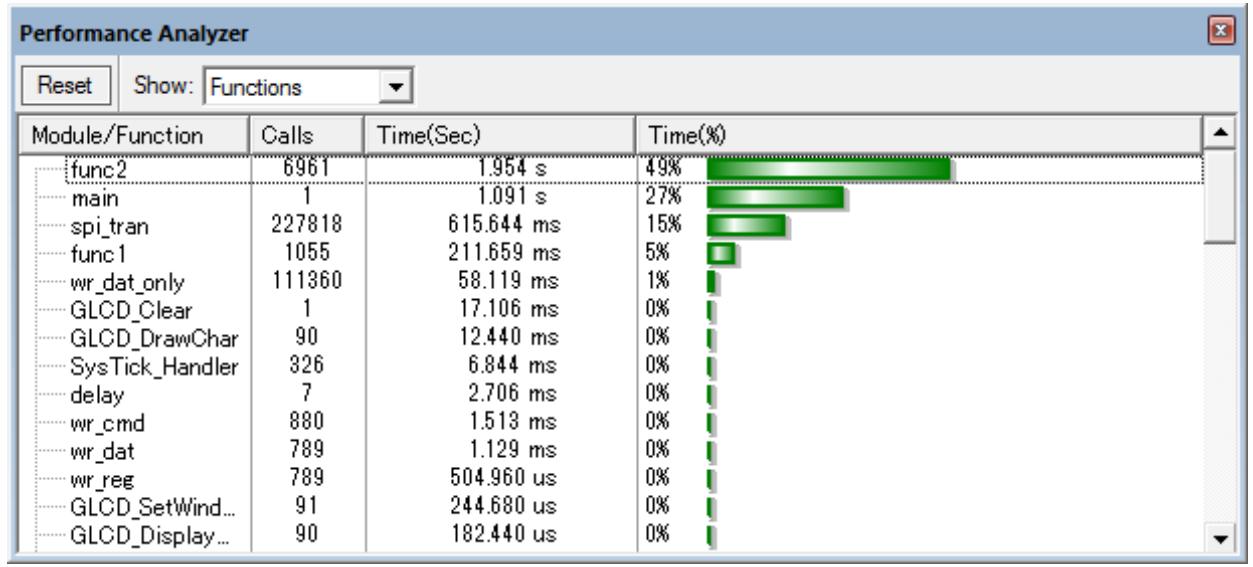
システムのゆとりの測定

- SLEEPCNTを使う方法
  - 測定区間内でイベントカウンタのSLEEPCNT値を測定することにより、SLEEP状態にあったサイクル数の総和を参照できる
  - RTOSのアイドルタスクがSLEEP状態を利用していれば何も実行するタスクがない期間のサイクル数に相当
- マーキングする方法
  - 各処理の開始と終わりにダミー変数への書きこみ(マーキング)を行うようにプログラムを変更することにより、データトレースのロジックアナライザ機能で視覚的に表示、測定することができる(uVisionの場合)



どこから手を付けるか

- プロファイル機能
  - 命令トレース情報からモジュール、関数単位での実行時間、呼び出された回数を表示
  - 使用頻度、実行時間の高いルーチンに対象をフォーカスすることができる
- コードカバレッジ
  - 命令トレース情報から命令レベルで何回実行されたかの情報を表示
  - 未実行箇所、使用頻度の高い箇所を特定できる
- Cortex-M0/M0+のプロファイル、コードカバレッジ
  - PC上のシミュレータにより同様のシミュレーション実行が可能(uVisionの場合)
  - サイクル精度の誤差があるので実測したサイクル数との検証が必要
  - M3/M4の実機環境に移植することができれば、プログラムとしての動作の検証は可能

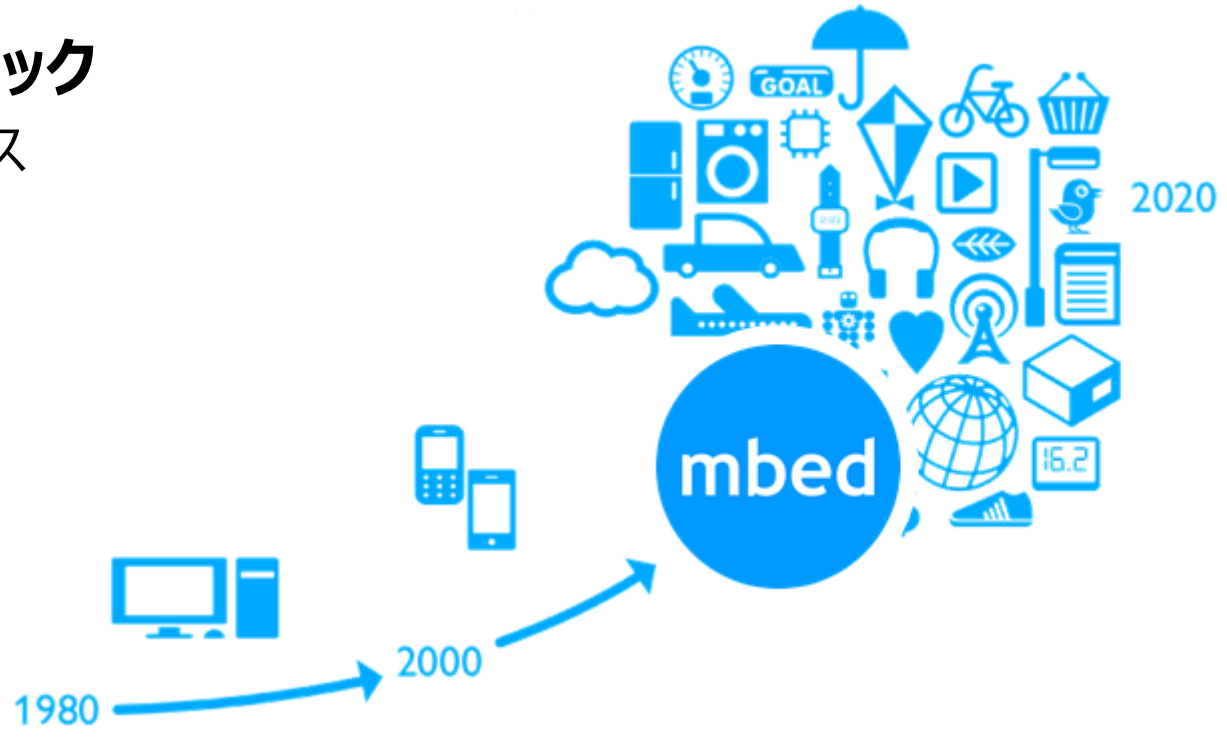


mbed の特徴

ARM

デバイスの開発プラットフォームとしての mbed™

- **優れたプラットフォームとツール**
  - 簡単なセットアップ、高速プロトタイプ、製品として使える十分な性能
- **ソフトウェアとハードウェアのビルディングブロック**
  - ARM® MCU、無線、ペリフェラル、クラウドサービス
- **開発者のためのデザイン**
  - オープンソース、サポート
- **開発者とパートナーのエコシステム**
  - コミュニティとのコラボレーション



ARM

マイコン開発を始めるときの課題

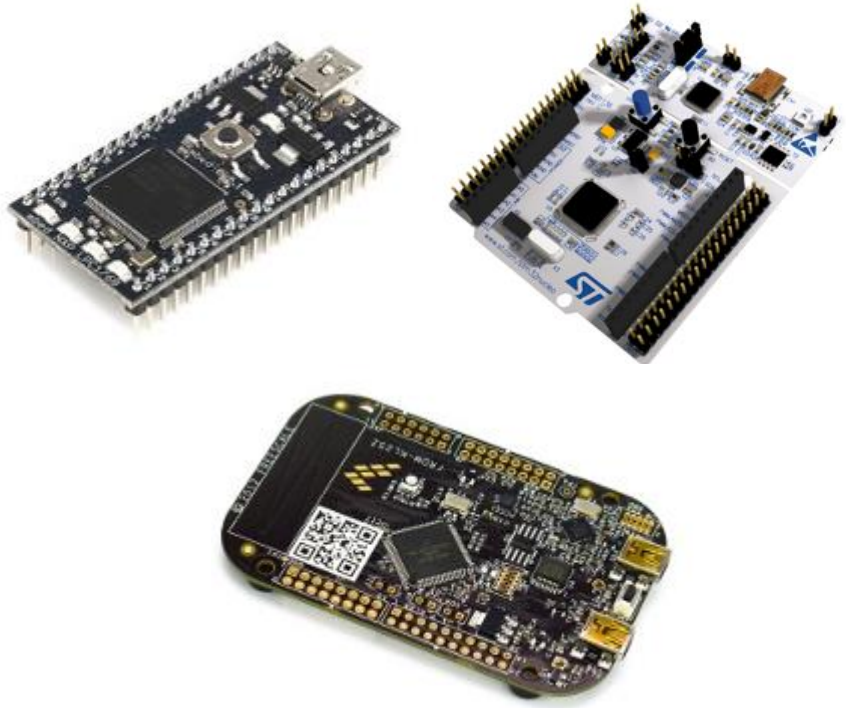
- **多機能なデバイス**
  - ARM コアは知っている・・・だけどペリフェラルが各社独自で複雑
  - 1,000 ページのドキュメントとアプリケーションノート、しかも英語マニュアルのみ？ ！
  - 設定するまでに一苦労。自分で全部コード書くの？
  - 苦労して作ったコードは、別のデバイスを使ったらポーティングやり直し・・・
- **開発ツール**
  - 使える開発ツールはそれなりに高価（購入しても、長期的に使いこなせるの？）
  - 既に自分の PC には色んな開発ツールが入っている！
  - そもそも、使っているのが Mac なんだけど・・・

ARM

mbed の特徴

- **ARM Cortex-M を使用したデバイス開発プラットフォーム**
- **ARM MCU を手軽に始める最短経路**

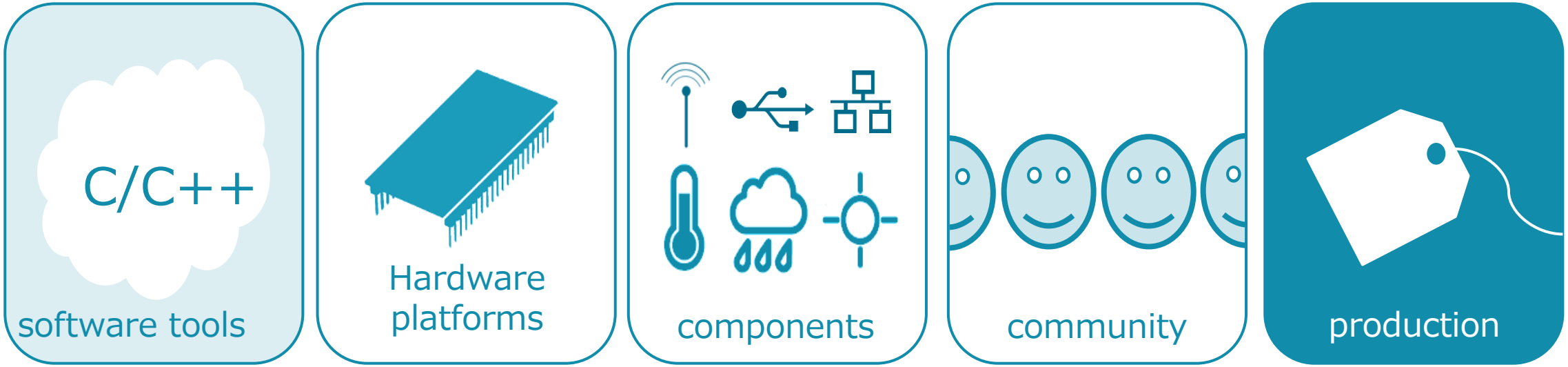
- **クラウド開発環境**
  - オンラインコンパイラ
- **ドラッグ＆ドロップ・プログラミング**
  - CMSIS-DAPデバッグ機能
- **C/C++ APIベース開発**
  - 検証済みの豊富なコンポーネント・ライブラリ



ARM

mbed プラットフォームとエコシステム

Enabling the ubiquitous intelligence and connectivity that underpins the Internet of Things



ARM

ハードウェアプラットフォームと HDK

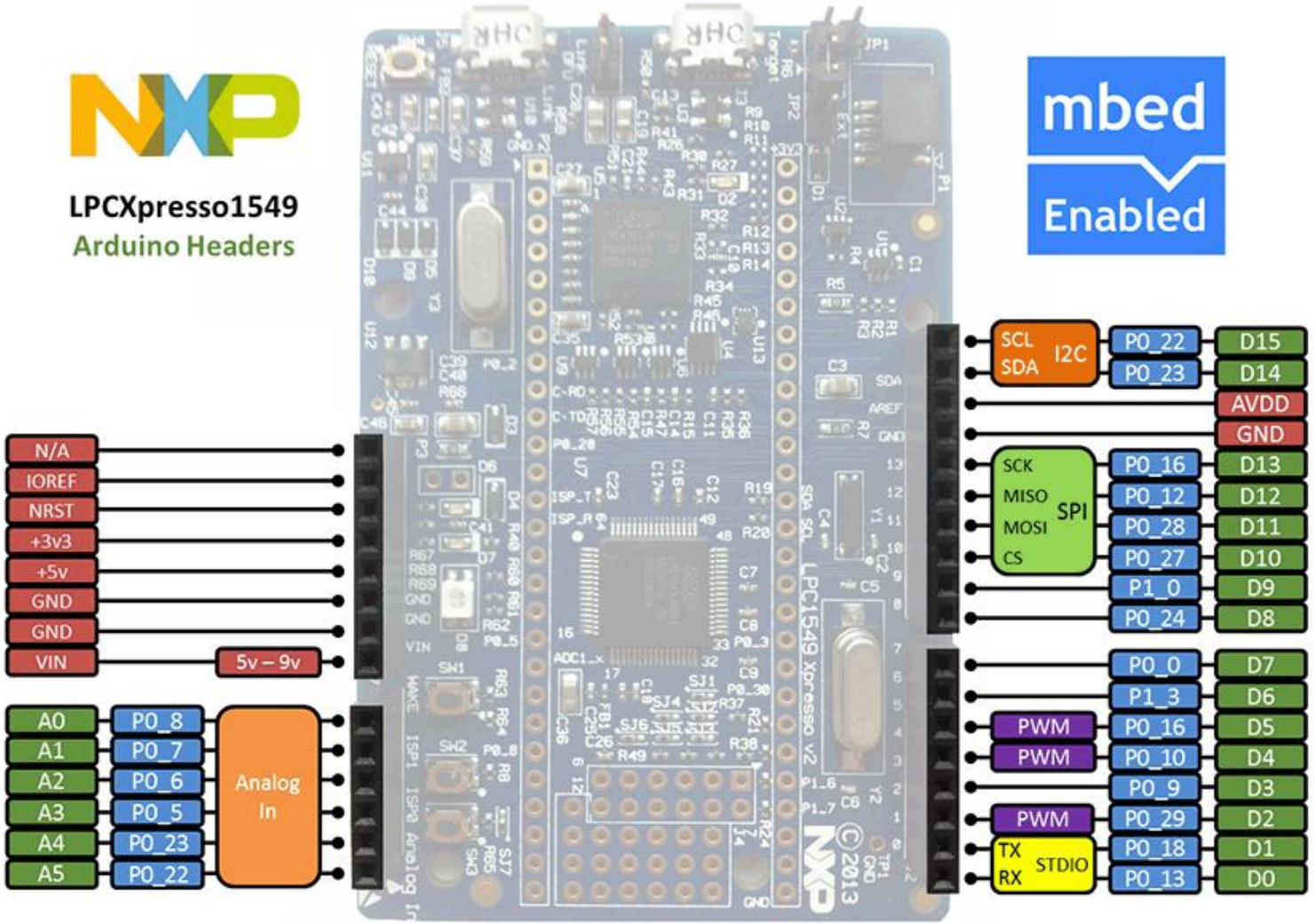
ARM

mbed-enabled プラットフォーム

- **mbed SDK が動作するハードウェア**
- **HDK を使用して mbed-enabled ハードウェアの開発が可能**
- **100,000 台以上の mbed-enabled ボードが出荷済み**
- **26種類のプラットフォーム**
  - https://mbed.org/platforms/
  - NXP, Freescale, STMicroelectronics, Nordic Semiconductor

外観は？

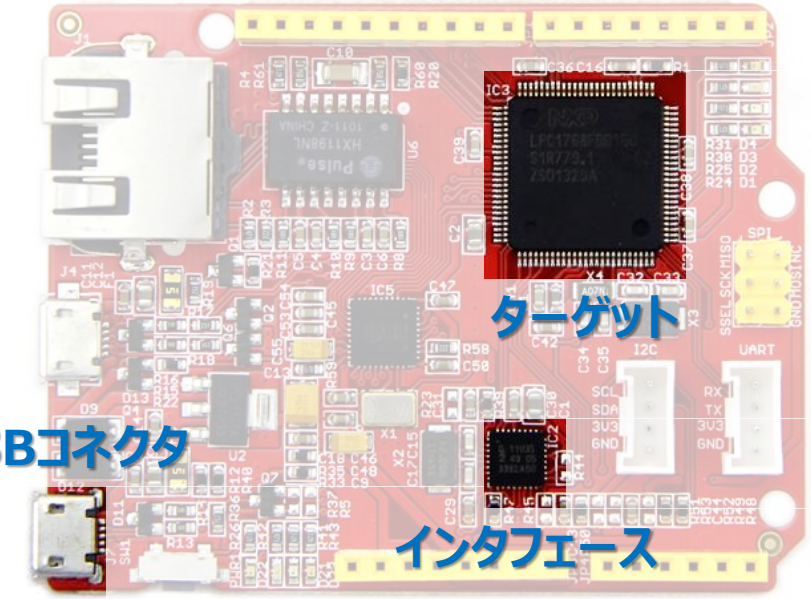
- **DIP 40ピン配列**
  - ブレッドボード対応
- **Arduino フォームファクタ**
  - シールドが使える
- **USBマスストレージドライブとして見える**
- **DIP 28ピンもあります！**



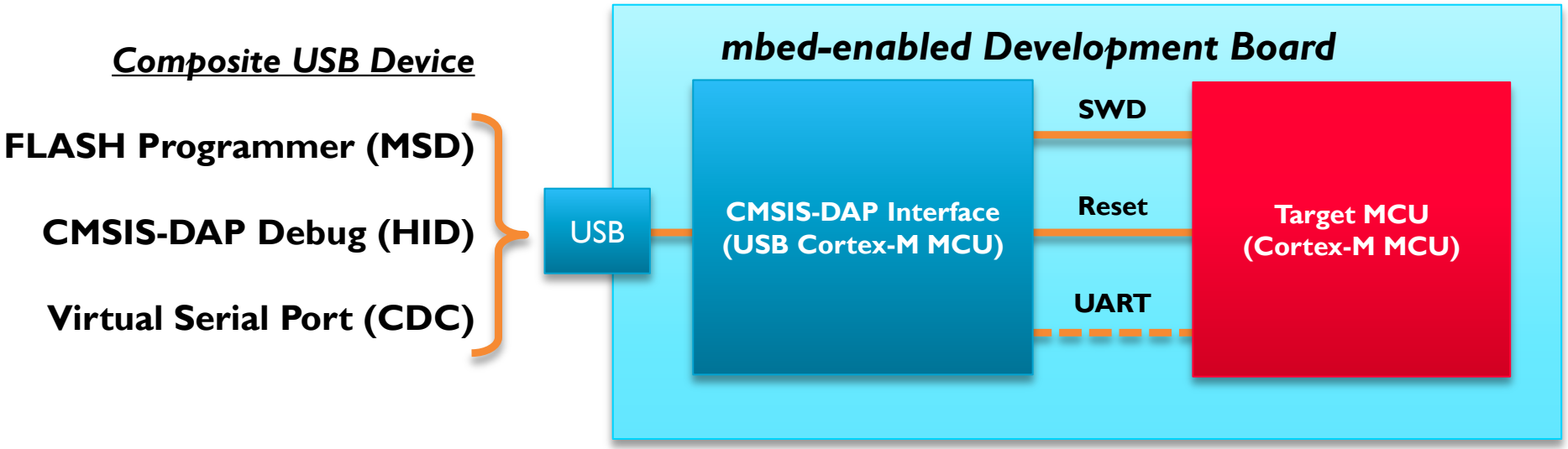
ARM

Target と Interface

- **オンボードUSB interface**
  - MSD 書き込みと、CMSIS-DAP デバッグ
  - ターゲットのUARTは、USB経由でシリアルポートにマッピング



- **シンプルな USB ドラッグ＆ドロップ書き込みと ARM ツールチェインでのフルデバッグ**



ARM

mbed HDK

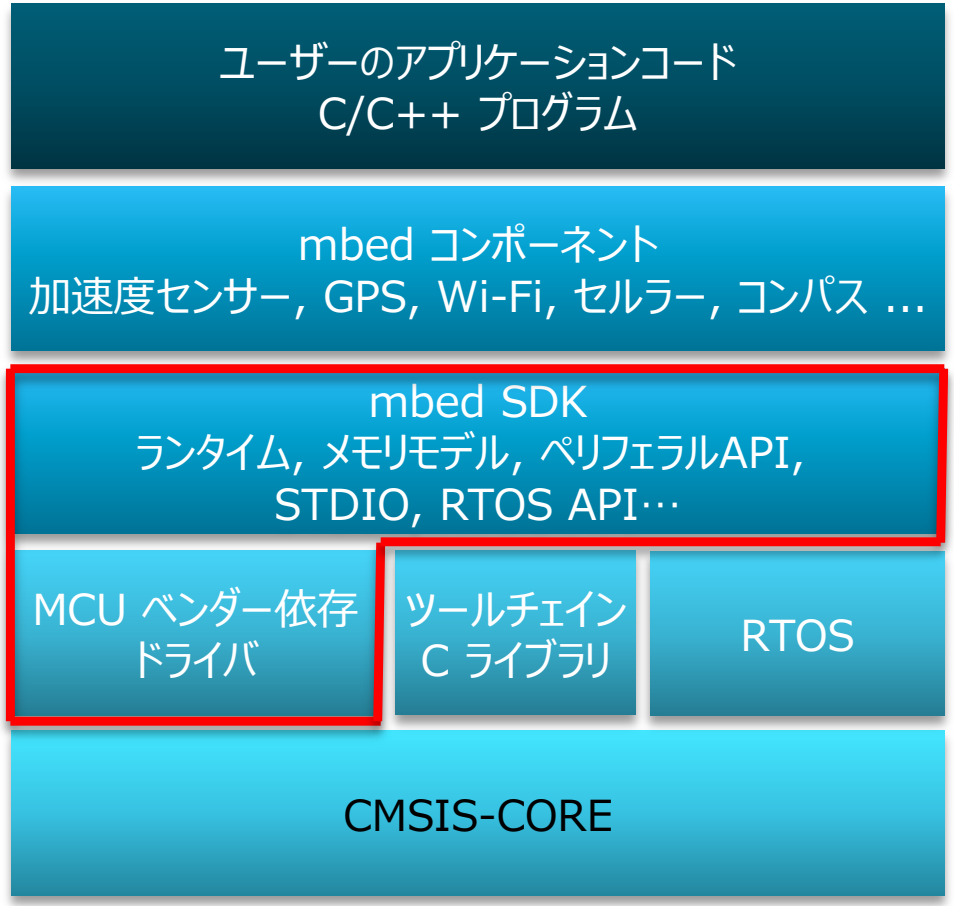
- **mbed プラットフォームを活用するための ハードウェアのリファレンスデザイン**
  - 回路図
  - 完全にオープンソースなファームウェア
  - ドラッグ＆ドロップ・プログラミング
  - シリアルUSB変換
  - デバッガとの接続
- **mbed 互換ボードやカスタムボードを 開発可能**
  - 開発環境、ライブラリの有効利用

mbed SDK ライブラリ

ARM

mbed ソフトウェアブロック図

- **開発者のための設計**
  - High-level API と標準化された環境
  - 異なるベンダーのデバイス間でポータビリティを確保
  - 商用、非商用どちらでも使用可能
  - オープンソース - Apache 2.0
- **堅牢かつスケーラブル**
  - 専門チームによるメンテナンスとサポート
  - 開発コミュニティによる貢献
- **業界標準な開発手法**
  - メジャーなツールチェインに対応
  - CMSIS 準拠



Hello mbed world! … LED を点滅させる

1. USB ケーブルで、mbed と PC を接続
2. 新規プロジェクト作成
3. ビルド
4. バイナリをドラッグ＆ドロップ

DigitalOut クラスのコンストラクタ

指定された GPIO ポートに出力

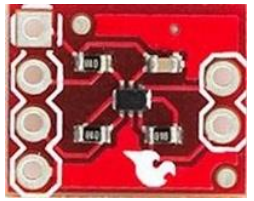
main.cpp

```
1  #include "mbed.h"
2
3  DigitalOut myled(LED1);
4
5  int main() {
6      while(1) {
7          myled = 1;
8          wait(0.2);
9          myled = 0;
10         wait(0.2);
11     }
12 }
13
```


ARM

mbed で何が出来るのか

- **mbed SDK で提供されている基本API（C++クラスライブラリ）**
  - Digital I/O, Analog I/O, Network, Communication interface, Timer and Interrupt, File System, RTOS, USBDevice, USBHost
- **mbed コミュニティが開発したライブラリの再利用（Cookbook）**
  - Display, Audio, SD Card, GPS, Bluetooth, WebSocket
  - 登録ライブラリ数は 2,000 以上（プログラムは、6,000 以上）
- **センサーデバイスやネットワークモジュールが簡単に繋がる**
  - コンポーネントライブラリ



API の例（DigitalOut クラス）

mbed - DigitalOut Class Reference

Public Member Functions

	DigitalOut (PinName pin) Create a **DigitalOut** conn
	DigitalOut (PinName pin, Create a **DigitalOut** conn
void	**write** (int value) Set the output, specified as
int	**read** () Return the output setting, r
DigitalOut &	**operator=** (int value) A shorthand for **write()**
	operator int () A shorthand for **read()**

```
00066     void write(int value) {
67          gpio_write(&gpio, value);
68      }
69
70      /** Return the output setting, represented as 0 or 1 (int)
71       *
72       *  @returns
73       *    an integer representing the output setting of the pin,
74       *    0 for logical 0, 1 for logical 1
75       */
00076     int read() {
77          return gpio_read(&gpio);
78      }
79
80  #ifdef MBED_OPERATORS
81      /** A shorthand for write()
82       */
00083     DigitalOut& operator= (int value) {
84          write(value);
85          return *this;
86      }
87
88      DigitalOut& operator= (DigitalOut& rhs) {
89          write(rhs.read());
90          return *this;
91      }
92
93      /** A shorthand for read()
94       */
00095     operator int() {
96          return read();
97      }
98  #endif
99
```


ARM

DigitalOut クラスの使用例

main.cpp

```
1  #include "mbed.h"
2
3  // DigitalOut クラスインスタンスの定義
4  DigitalOut myled(LED1);
5
6  int main() {
7      // operator= による書き込み処理（write() と等価）
8      myled = 1;
9      myled.write(1);
10     // read() による読み込み処理と書き込み
11     myled = myled.read();
12     // operator int() で読み込んだ値を反転して、operator= で書き込み
13     myled = !myled;
14 }
15
```


ARM

その他の代表的な API

- デジタル入力 **DigitalOut**
- デジタル入出力 **DigitalInOut**
- 割り込み入力 InterruptIn
- アナログ入力 AnalogIn
- アナログ出力 AnalogOut
- PWM出力 PwmOut
- シリアルバス Serial, SPI
- I2C バス I2C
- 周期イベント **Ticker**
- USB USBDevice, USBHost
- 待ち時間処理 **wait**

ARM

mbed SDK の抽象化

```
#include "mbed.h"

DigitalOut myled(LED1);

int main() {
    while(1) {
        myled = 1;
        wait(0.2);
        myled = 0;
        wait(0.2);
    }
}
```

```
#include "LPC17xx.h"

void gpio_init(gpio_t *obj, PinName pin) {
    if(pin == NC) return;
    obj->pin = pin;
    obj->mask = gpio_set(pin);
    LPC_GPIO_TypeDef *port_reg = (LPC_GPIO_TypeDef *) ((int)pin & ~0x1F);
    obj->reg_set = &port_reg->FIOSET;
    obj->reg_clr = &port_reg->FIOCLR;
    obj->reg_in  = &port_reg->FIOPIN;
    obj->reg_dir = &port_reg->FIODIR;
}

int main() {
    PinName pin = 0;
    gpio_t obj;
    gpio_init(&obj, (PinName)(0x2009C000 + 31 + 8)); // P1_18
    unsigned int mask_pin18 = 1 << 18;
    volatile unsigned int *port1_set = (unsigned int *)0x2009C038;
    volatile unsigned int *port1_clr = (unsigned int *)0x2009C03C;
    while (1) {
        *port1_set = (port1_set | mask_pin18);
        wait(0.2);
        *port1_clr = (port1_clr | mask_pin18);
        wait(0.2);
    }
}
```

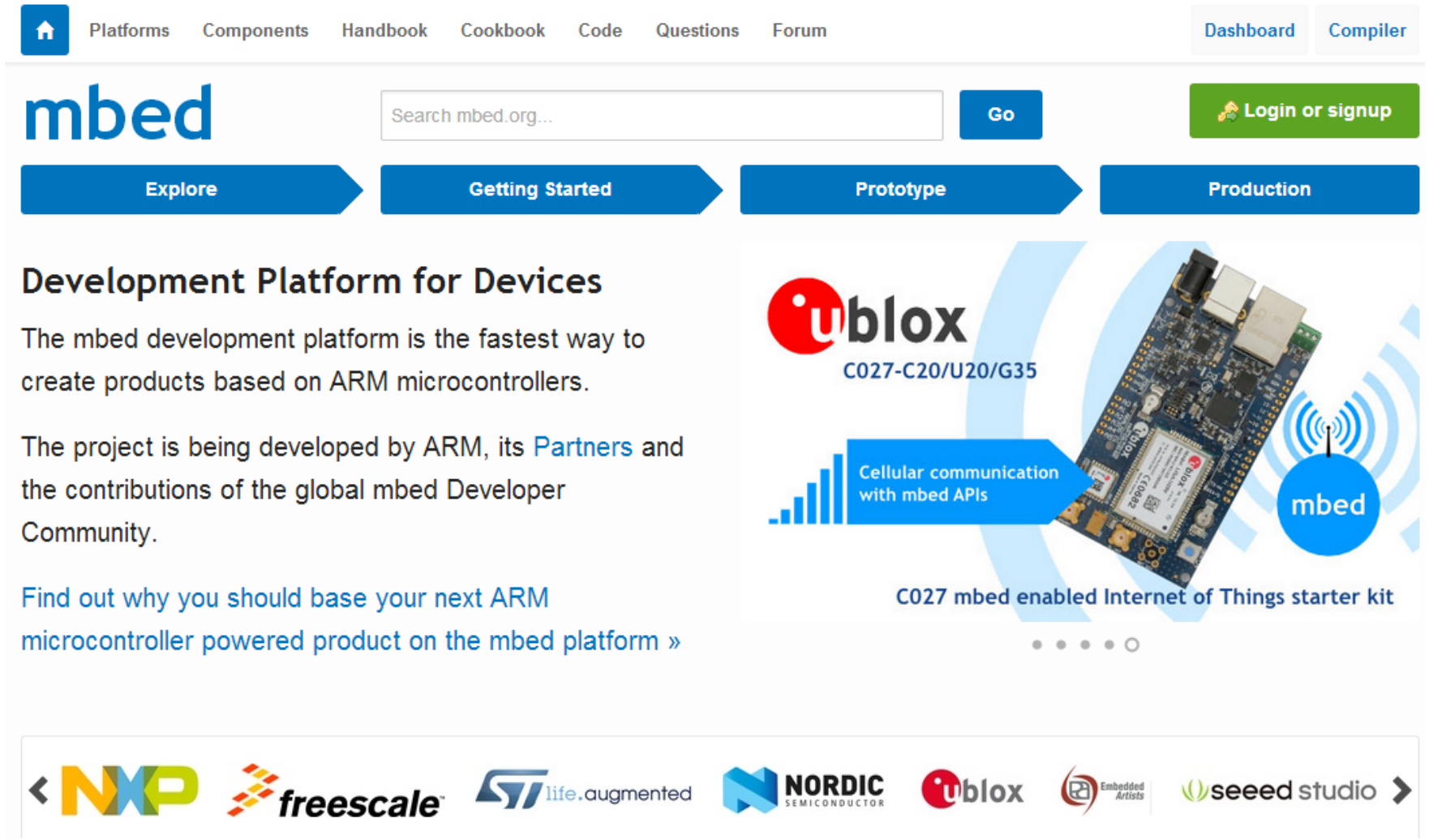

ARM

クラウド開発環境

ARM

mbed.org ホームページ

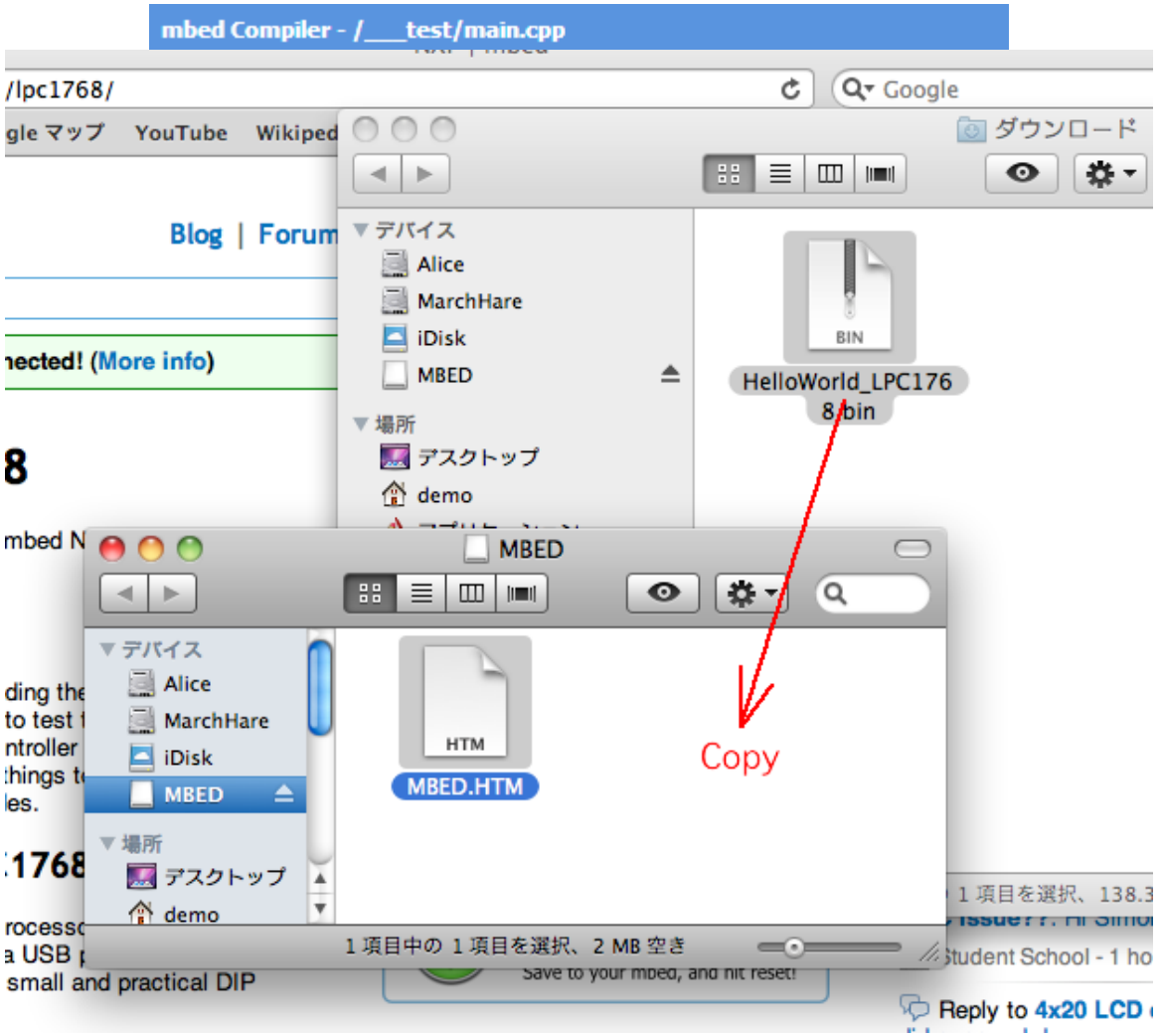
- http://mbed.org
  - 全ての情報を集約
- **Platforms**
- **Components**
- **Handbook**
- **Cookbook**
- **Code**
- **Questions**
- **Forum**
- **Compiler**



ARM

クラウド開発環境

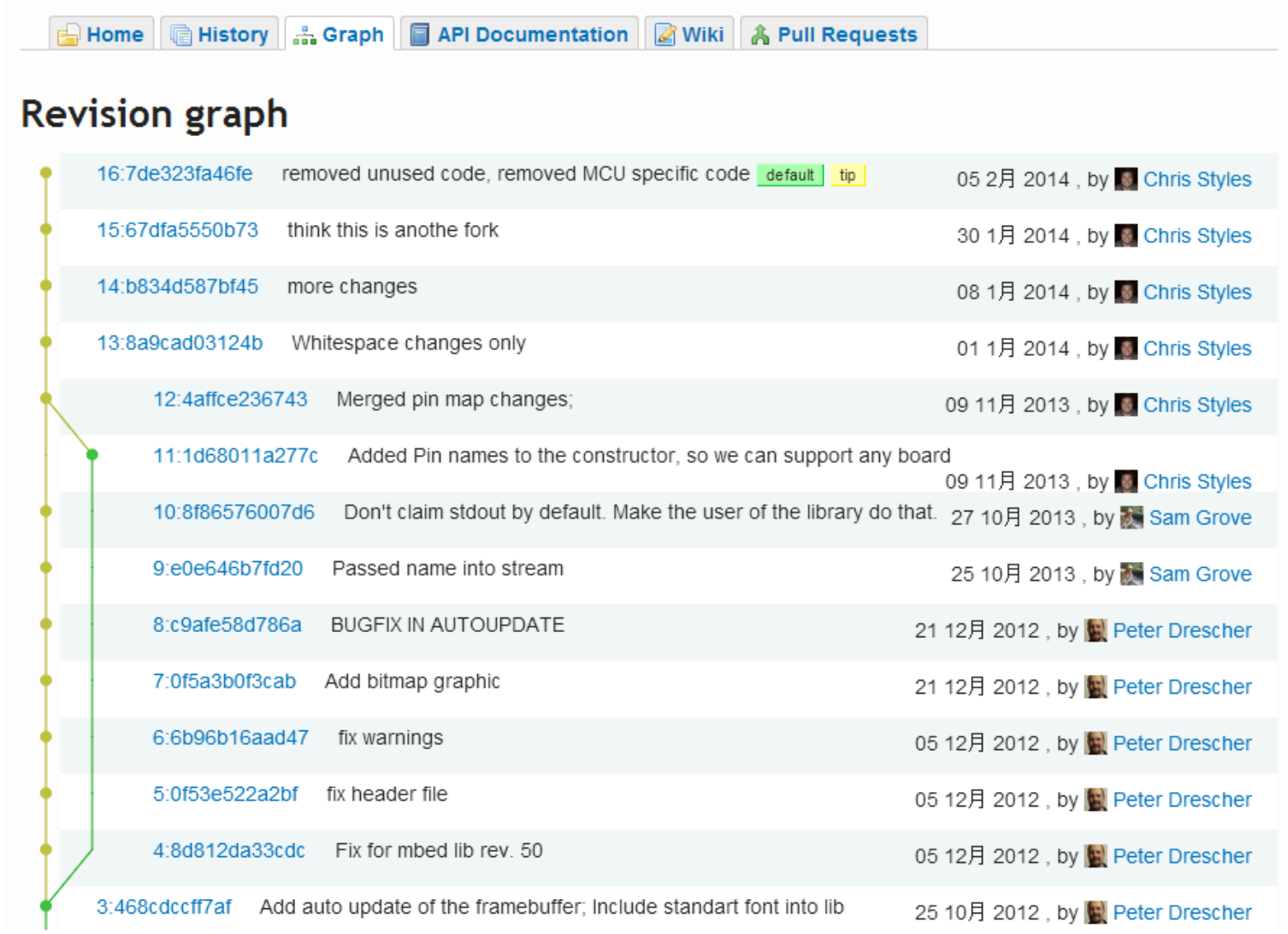
- **インストール不要！**
- **オンライン IDE**
  - プラットフォーム非依存の開発環境
  - ブラウザベース
- **オンラインコンパイラ**
  - ARM純正の最適化コンパイラ
  - [Compile] ボタンを押すと、生成されたバイナリがダウンロードされる
- **ターゲットボードへの書き込み**
  - USB のドライブにドラッグ＆ドロップするだけ



ARM

ソースコード管理

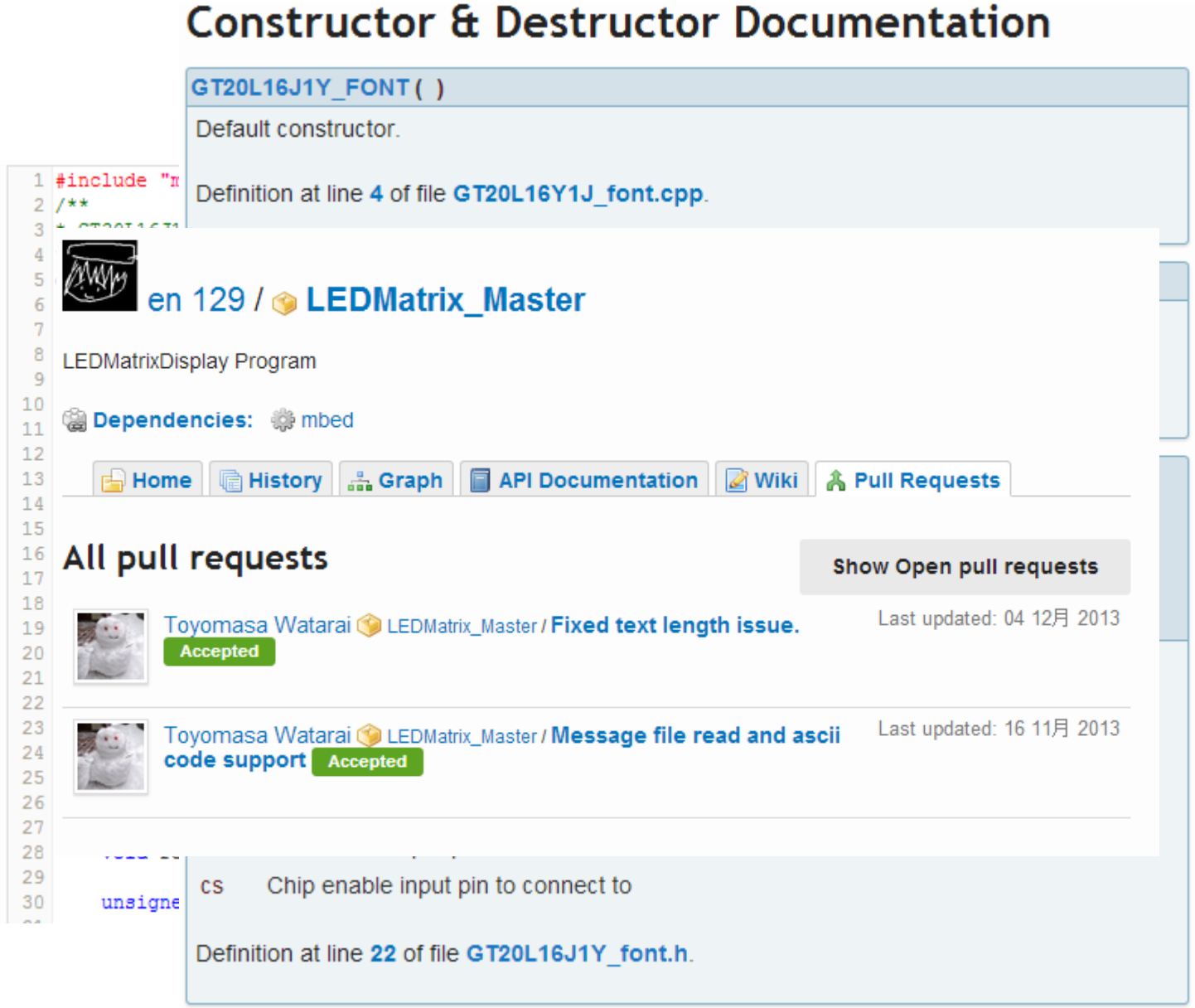
- **コードレポジトリ**
  - コラボレーション・ワークフロー
  - IDEに機能を統合
- **History**
  - 変更履歴
  - コードの diff 表示
- **Graph**
  - レビジョン情報、フォーク、マージ



ARM

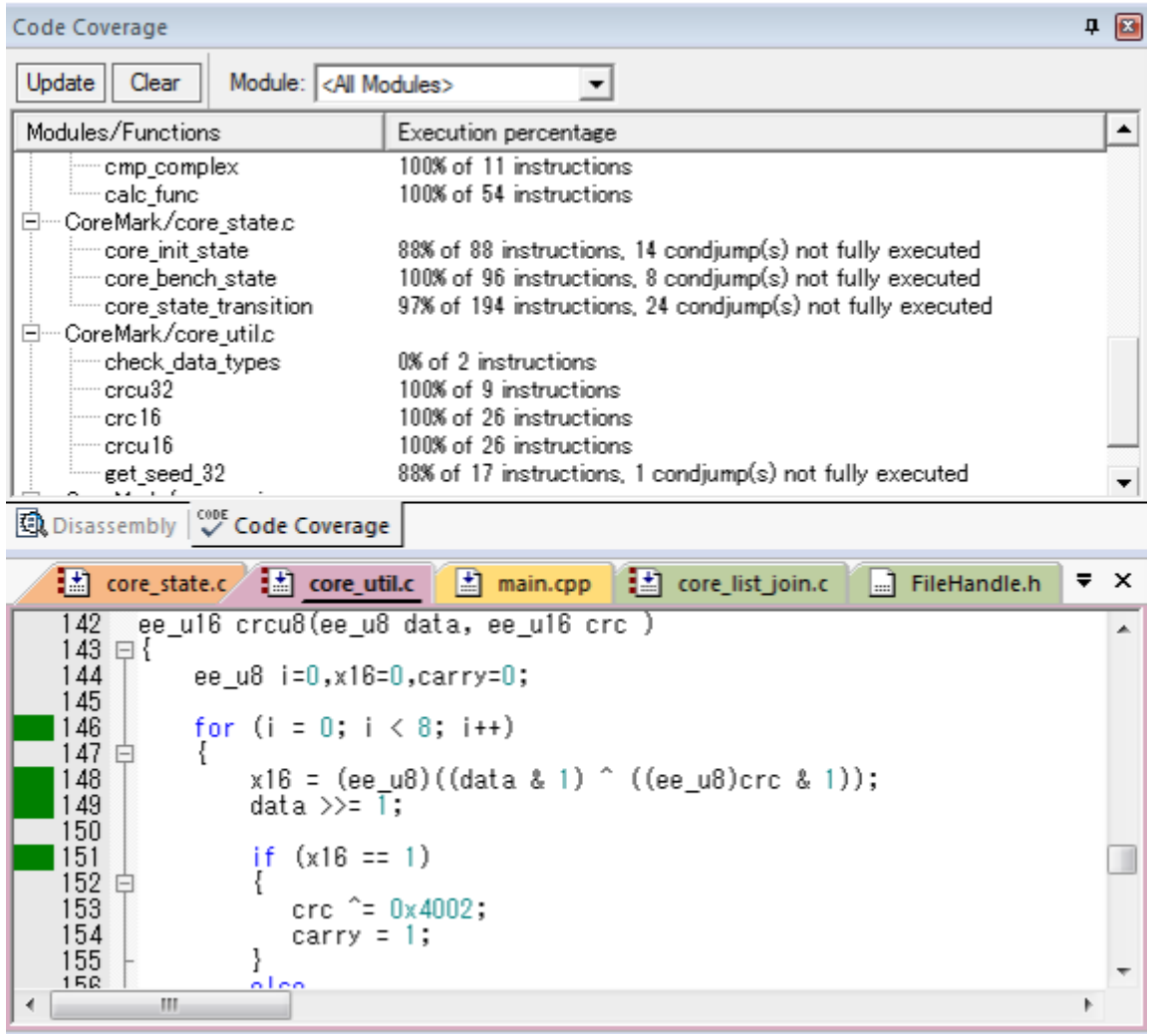
ソースコード管理（続き）

- **API Documentation**
  - Doxygen ドキュメントブロック認識
- **Pull Requests**
  ユーザーからのソース変更リクエスト
- **インポート**
  - コードを自分のオンライン環境で使用
- **コミット**
  - 自分のコードの特定のレビジョンを登録
- **パブリッシュ**
  - 自分のコードを公開



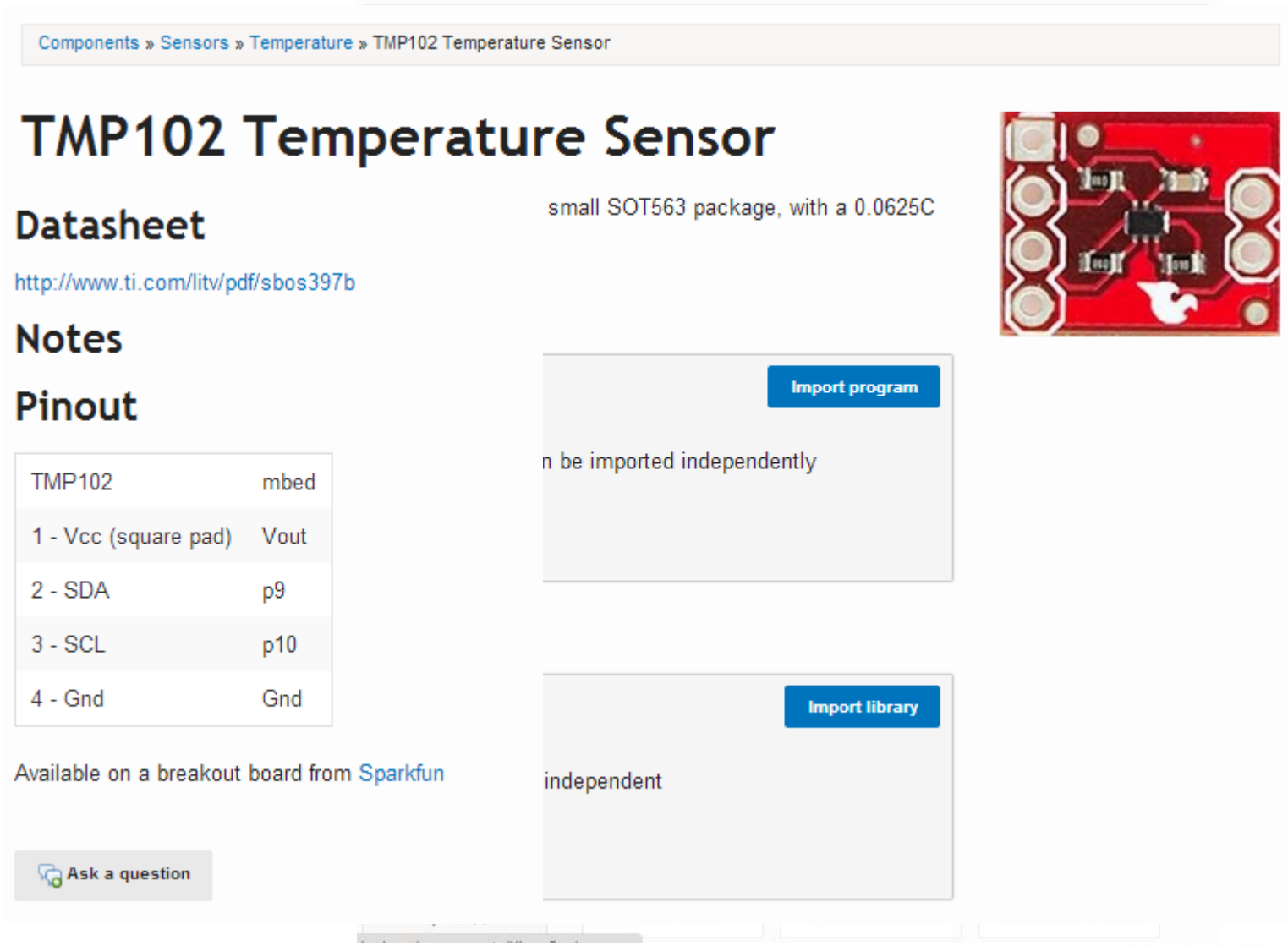
オフライン環境とデバッグ

- **printf() で変数やメッセージなどをシリアルで出力**
- **OpenOCDでGDBserver経由のデバッグ**
- **プロジェクトをエクスポートして、オフライン環境で使用することも可能**
  - Keil™ uVision4 (MDK-ARM), DS-5
  - NXP LPCXpresso IDE
  - IAR EWARM
- **mbed で使用可能なデバッグ機能**
  - C/C++ ソースレベルデバッグ、逆アセンブラ
  - フラッシュメモリへの書き込み
  - ハードウェアブレークポイント（4または8箇所）
  - ウォッチポイント（2または4箇所）
  - CPUレジスタ、ペリフェラルレジスタへのアクセス



コンポーネント・ライブラリ

- **部品単位にライブラリをインポート**
- **コンポーネント**
  - アクチュエータ
  - 通信モジュール
  - ディスプレイ
  - IoT サービス
  - ロボティックス
  - センサー
  - ストレージ



ARM

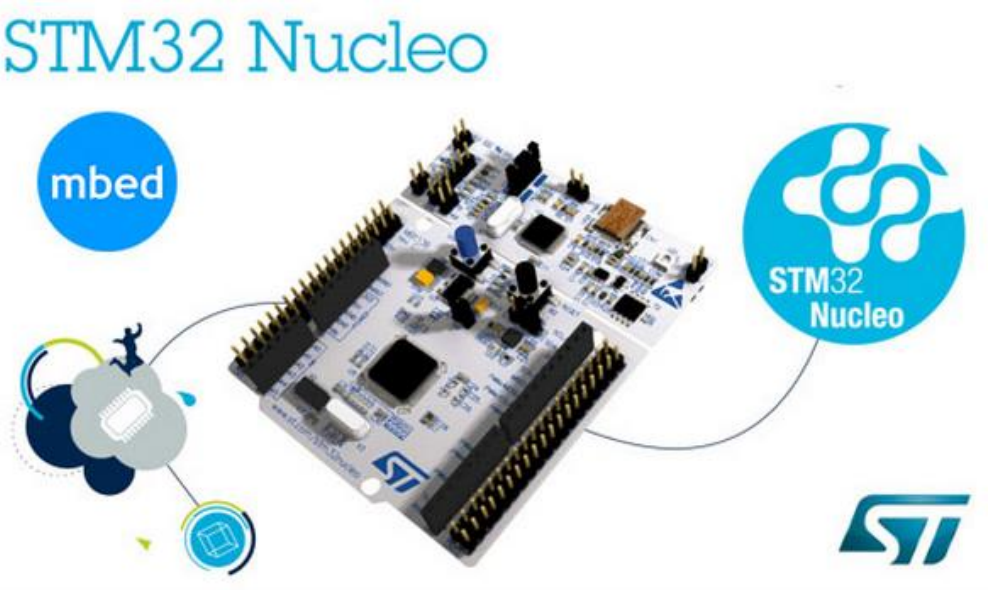
簡単だけど、それだけじゃない

- **簡単につかえる ≠ 単機能**
- **開発環境**
  - mbed SDK の API 以外も当然使える（ハードウェアを直接叩く！）
  - インラインアセンブラ対応
  - エクスポートして、オフライン環境でも使える
  - Keil MDK-ARM や NXP LPCXpresso IDE 等
- **プラットフォーム**
  - 対応プラットフォームの拡充（GitHub を覗くと開発動向が分かる？）
    https://github.com/mbedmicro/mbed/tree/master/libraries/mbed/targets/hal
  - HDK による互換ボード、カスタムボード

ARM

最近のアップデート

- **多数のプラットフォームが追加されました**
  - STマイクロエレクトロニクス STM32 Nucleo（9種類）
  - Nordic nRF51822-mKIT
  - NXP LPCXpresso LPC1549, LPC11U68
  - Freescale FRDM-KL05Z, FRDM-K64F



ARM

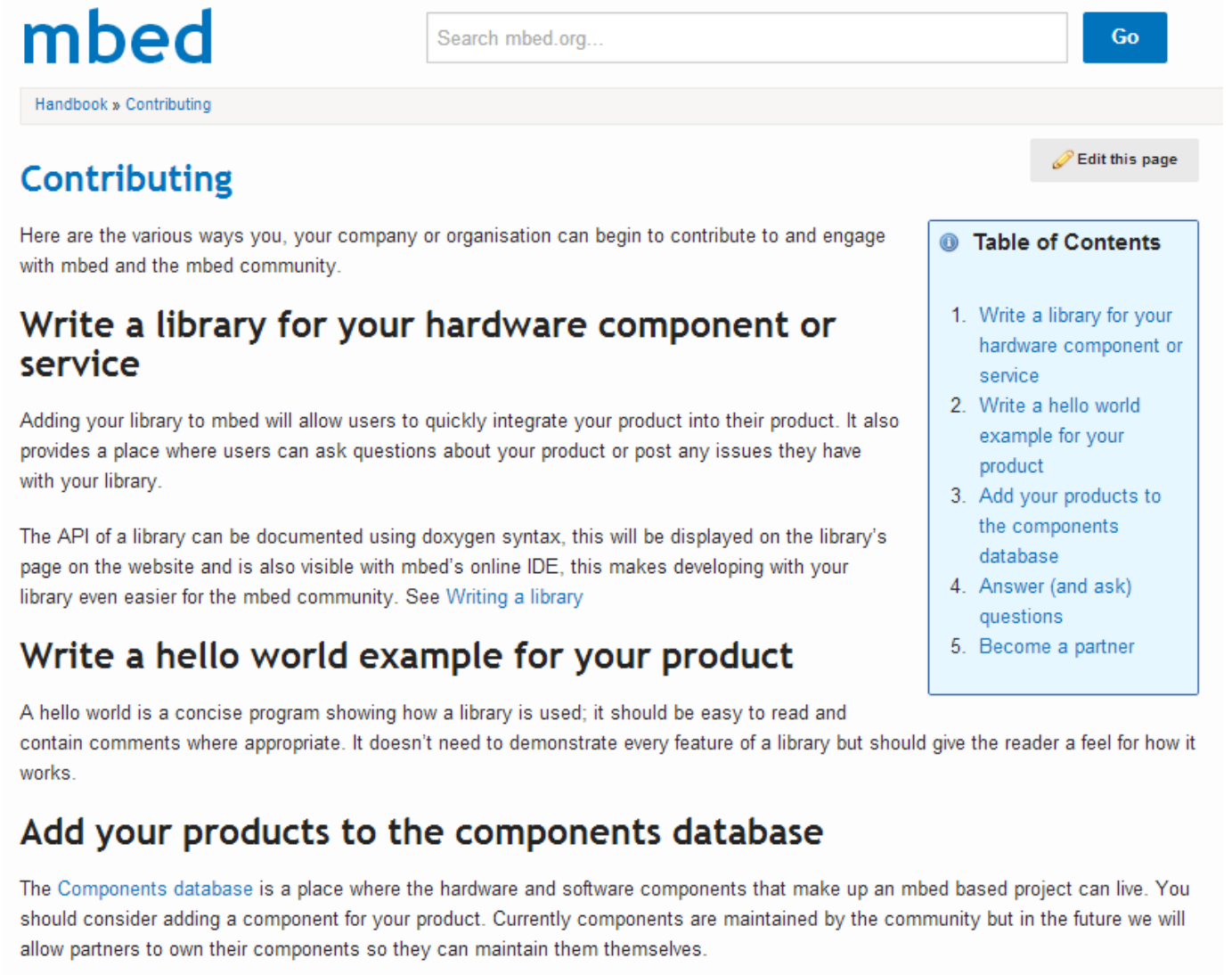
最近のアップデート（続き）

- **mbed LPC1114FN28 リリース**
  - 日本初のmbed製品
- **mbed Application Shield**
  - Component ページに登録済み
  - ほぼ、Application Board 仕様

ARM

mbed プロジェクトに参加するには

- **プラットフォームパートナー**
  - mbed-enabled　プラットフォーム
- **コンポーネントパートナー**
  - センサー、通信、ソフトウェア、ツール
  - オープンソース・ライブラリへの移植
- **開発コミュニティ**
  - 参加、協業、共有
  - グローバルコミュニティによる開発



ARM

mbed パートナー

- シリコンパートナー
  - NXPセミコンダクターズ
  - Freescaleセミコンダクター
  - STマイクロエレクトロニクス
  - Nordicセミコンダクター
- プラットフォームパートナー
  - U-blox
  - Embedded Artist
  - Seeed Studio
  - スイッチサイエンス
  - その他

ARM

他の開発プラットフォームとの違い

	mbed NXP LPC1768	Arduino UNO	Raspberry-Pi
CPU	Cortex-M3 @96MHz	ATmega328 @16MHz	ARM1176JZF @700MHz
Flash	512KB	32KB	-
RAM	32KB	2KB	512MB
Operation voltage	3.3V	5.0V	3.3V
Power	135mA	46.5mA	700mA
OS	mbed-RTOS (option)	-	Linux
開発言語	C/C++	C言語風	Python 他
特徴	オンライン環境 検証済みのライブラリ 高速プロトタイピング 高いポータビリティ	多数のShieldボード 豊富な互換ボードと情報量 ホビーストに人気	Linux ベースなので、豊富な ソフトウェアが使用可能

mbed は、開発作業に素早く着手する事が出来る近道

ARM

良くある質問と回答

- **ターゲット上で RTOS は動くの？**
  - mbed-RTOS
  - Linux は動きません
- **どこまでが無償（サービス、ソフトウェア）なのか？**
  - mbed.org, HDK, SDK, オンラインコンパイラの使用, ライブラリ登録・再利用
  - MDK-ARM 等のツールはライセンス製品
- **どんなマイコンでも mbed が動くのか？**
  - ARM Cortex-M, CMSIS-CORE 実装が必要
- **日本語対応は？**
  - フォーラムやノートブックは対応、エディタは未対応

ARM

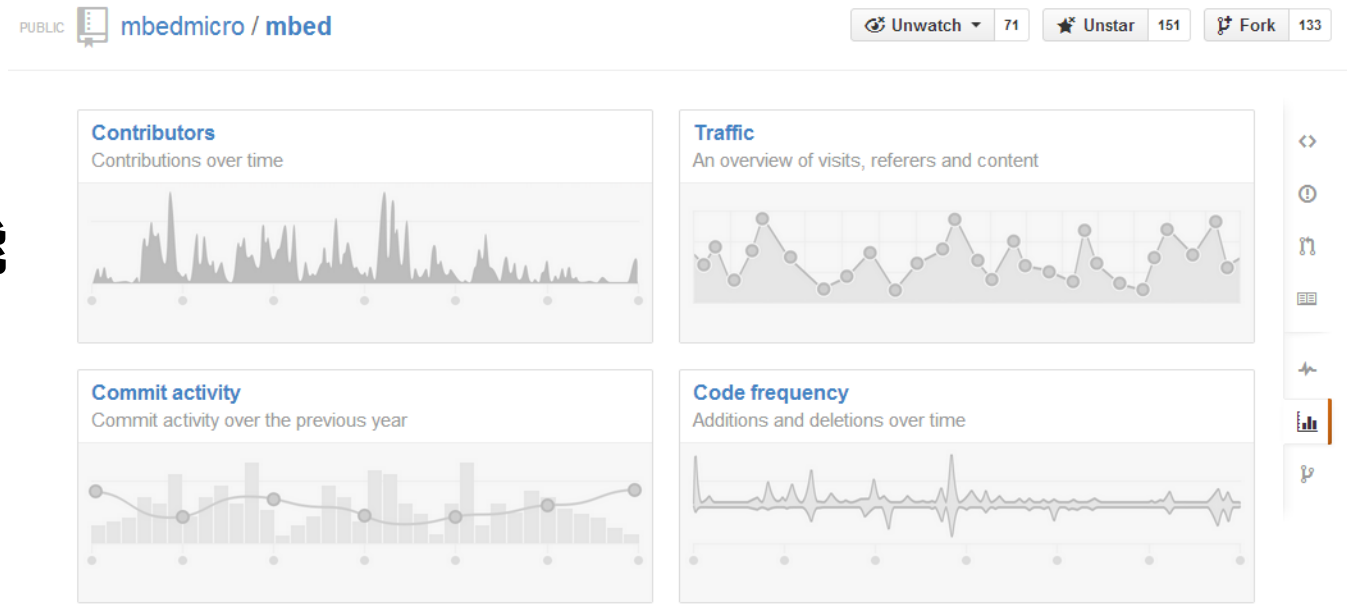
ユーザーミートアップイベント「mbed祭り」

- **ユーザが企画、運営している mbed ユーザのためのイベント**
  - 主催：mbed 祭り実行委員会
- **開催時期は不定期（年に数回開催）**
  - 東京、横浜、名古屋、大阪、札幌
- **内容**
  - プレゼン
  - デモ（mbedを使った作品紹介）
  - 抽選会
  - 親睦会
- **http://mbed.doorkeeper.jp/**

ARM

オープンソースプロジェクト

- **GitHubコードレポジトリ**
  - https://github.com/mbedmicro/
- **mbed SDK/HDKは誰でも参照可能**
  - mbed (mbed SDK)
  - CMSIS-DAP (mbed HDK firmware)
  - PyOCD (Python library for HDK)
- **mbed SDK レポジトリ**
  - 1,800以上のコミット
  - 50人以上のコントリビュータ
  - 170以上のフォーク



mbed SDK ポーティング

- **コードレポジトリ**
  - https://github.com/mbedmicro/mbed
- **mbed SDK 内部構造の解説**
  - https://mbed.org/handbook/mbed-library-internals
  - https://mbed.org/users/MACRUM/notebook/mbed-library-internals/ （日本語）
- **ポーティング方法の解説**
  - https://mbed.org/handbook/mbed-SDK-porting
  - https://mbed.org/users/MACRUM/notebook/mbed-sdk-porting-jp/ （日本語）
- **ポーティング環境の構築（日本語版）**
  - https://mbed.org/users/ytsuboi/notebook/getting-started-mbed-porting-ja/
- **mbed ツール**
  - https://mbed.org/handbook/mbed-tools
  - http://mbed.org/users/MACRUM/notebook/mbed-tools/ （日本語）

ARM

mbed SDK/HDK開発に必要なソフトウェア

- **Keil MDK-ARM v4.74**
  - インタフェースファームウェアのビルドに必要（uVision 4）
  - mbed ライブラリ及びテストコードのビルドに必要（コマンドラインコンパイラ）
  - テスト、デバッグ
- **Python v2.7**
  - mbed ライブラリ及びテストコードのビルドに必要
  - private_setting.py にローカル設定を記述
- **GitHub 最新版**
  - コードリポジトリでの開発用

ARM

GitHub を使用して mbed SDK のソースコードを取得する

- GitHub のアカウントを作成する
- https://github.com/mbedmicro/mbed
  - 画面右上の “Fork” ボタンをクリックし、自分のアカウントにフォークを作成する
- GitHub アプリを立ち上げる
  - “+” ボタンから、”Clone” を選択する
  - 自分のアカウント内の “mbed” を選択する
  - “Clone mbed” を選択する
  - フォルダを選択し、OK ボタンを押す
  - → クローンが作成され、ローカルにソースコードがコピーされる

ARM

mbed SDK ライブラリをビルドする

- workspace_tools/private_settings.py を作成する
  - setting.py をコピーし、ローカルのディレクトリ設定に変更する
- コマンドプロンプトを開き、コマンドラインでビルドする
  - >cd C:¥Users¥xxxxxx¥Documents¥GitHub¥mbed
  - >workspace_tools¥build.py –m LPC1768 –t ARM

mbed SDK テストケースをビルドする

- コマンドプロンプトを開き、コマンドラインでビルドする
  - >cd C:¥Users¥xxxxxx¥Documents¥GitHub¥mbed
  - >workspace_tools¥make.py –m LPC1768 –t ARM –p xxx
    -p オプションを付けないと、テストケースの一覧が表示される

ARM

mbed SDK 新しいターゲットの追加方法

- **ビルドスクリプトにターゲットを追加**
  - {GitHub}¥mbed¥workspace_tools¥targets.py　（必須）
  - {GitHub}¥mbed¥workspace_tools¥build_release.py
  - {GitHub}¥mbed¥workspace_tools¥export_test.py
  - {GitHub}¥mbed¥workspace_tools¥¥export¥ds5.py
- **CMSIS-CORE, スタートアップコード関連を追加**
  - {GitHub}¥mbed¥libraries¥mbed¥targets¥cmsis¥TARGET_XXX
- **ターゲット依存部分のコードを追加**
  - {GitHub}¥mbed¥libraries¥mbed¥targets¥hal¥TARGET_XXX
- **テストコードの修正**
  - {GitHub}¥mbed¥libraries¥tests

ARM

targets.py の修正

- ターゲットの class を追加

```
class LPC4337_M4(Target):
    def __init__(self):
        Target.__init__(self)
        self.core = "Cortex-M4F"
        self.extra_labels = ['NXP', 'LPC43XX', 'LPC4337']
        self.supported_toolchains = ["ARM", "GCC_CR", "IAR", "GCC_ARM"]
```

- TARGETS リストにターゲットを追加
- ビルドの例

```
> workspace_tools¥build.py –m LPC4337_M4 –t ARM
```


ARM

デバイス定義ファイル device.h の作成

- **mbed SDK ポーティングで対応する機能の定義**
  - {GitHub}¥mbed¥libraries¥mbed¥targets¥hal¥TARGET_NXP¥TARGET_LPC15XX¥device.h
  - 必用な部分を 1 に変更してから開発を進める
- **【注意】TARGET_XXXX 配下のソースコードは必ずビルド対象になる**
  - #define DEVICE_PWMOUT 0
  - TARGET_XXXX¥pwmout_api.c は必ずビルドされる
  - **#if DEVICE_PWMOUT** でガードする（LPC11U68のポーティング参照）
- **ここでハマりやすいコンパイルエラー（エラーの場所から原因が推測できないケース）**

```
[Error] CAN.h@235:  #70: incomplete type is not allowed
"no source": Warning:  #3036-D: "C:/Keil/ARM/armcc¥include" was specified as both a system and
non-system include directory -- the non-system entry will be ignored
"C:¥Users¥toywat01¥Documents¥GitHub¥mbed¥build¥mbed¥CAN.h", line 235: Error:  #7
0: incomplete type is not allowed
C:¥Users¥toywat01¥Documents¥GitHub¥mbed¥libraries¥mbed¥common¥CAN.cpp: 1 warning, 1 error
```

  - ターゲット部分に構造体の宣言を追加する必要がある（この場合は、can_s 構造体）

ARM

PinNames.h の作成

- **Pinmap の定義**
  - IOポートの定義：PinName 型
  - 最近のプラットフォームでは、Arduino Shield のピン名の定義
  - 定義の中身は、ターゲット依存（PinName 型の中身はターゲットで閉じるので自由に定義して良い）
- **一般的な例（LPC1549 ポーティング）**

```
typedef enum {
    // LPC Pin Names
    P0_0 = 0,
    P0_1, P0_2, P0_3, P0_4, P0_5, P0_6, P0_7, P0_8, P0_9, P0_10, P0_11, P0_12, P0_13 …
```

- **特殊な例（LPC11U68ポーティング）**

```
typedef enum {
    // LPC11U68 Pin Names (PORT[19:16] + PIN[15:9] + IOCON offset[8:0])
    P0_0 = (0 << PORT_SHIFT) | (0 << PIN_SHIFT) | 0x000,
    P0_1 = (0 << PORT_SHIFT) | (1 << PIN_SHIFT) | 0x004,
    P0_2 = (0 << PORT_SHIFT) | (2 << PIN_SHIFT) | 0x008,
```


ARM

スタートアップコードの作成

- **Startup_XXXXXX.s**
- **開発ツールや半導体ベンダーから提供される CMSIS コードを再利用する**
  - MDK-ARM では、ARM Compiler (-t ARM or –t uARM)
  - LPCXpresso では、GCC CodeRed (-t GCC_CR)
- **スタックポインタの初期値は変更しましょう**
  - 内蔵RAMのトップに設定する
- **ARM, uARM 用の設定変更**
  - 別ファイルにする場合
  - 同じファイルにしてマクロで切り替える場合

ARM

スタートアップコードの作成（続き）

- **System_XXXXXX.c, System_XXXXXX.h**
  - クロックモジュールの設定（メイン、ペリフェラル、USB等）
  - 電源系の設定
  - CMSIS のコードなので、変更は最小限に
- **配布されているコードが常に正しいというわけではない…**
  - 半導体ベンダーが提供しているコードも参考にしましょう（LPCOpenとか）

ARM

必要なターゲット依存部分の実装

- **ターゲット依存部分で閉じるので独自実装**
  - 必須の関数は {GitHub}¥mbed¥libraries¥mbed¥api 配下のヘッダファイル参照
  - それ以外は独自に実装・拡張しても可
  - とは言っても、組み込みソフトなのでコードは小さく、シンプルに
- **ターゲット依存部分の実装例を見てみましょう**

テスト、テスト、テスト…

- **マニュアルテスト**
  - テストアプリを作成して、マニュアルでテストを行う
  - uVision4のプロジェクトを作成すると便利

  https://mbed.org/users/MACRUM/notebook/how-to-setup-mdk-arm-project-to-flashdebug-mbed-sd/
  - デバッグ情報を生成するようにすると更に便利

  http://mbed.org/users/ytsuboi/notebook/getting-started-mbed-porting-ja/

- **mbed SDK 標準のテストスイート**
  - > workspace_tools¥make.py –m LPC4337_M4 –t ARM –p XXX
  - テストコードを読まないと駄目なケースが多い（ループバックで使用するピンとか）
  - ビルドとテストの自動化

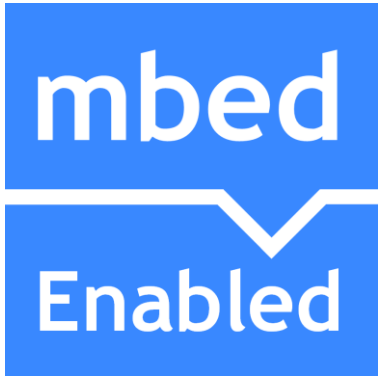
ARM

（私が経験した）SDK portingで発生したバグなど

- **ポートを変えると動かなくなる**
  - 別の機能で使用されていた（tickerとか）
- **チャンネルが異なると動かなくなる**
  - チャンネル切り替え判定ロジックの不具合
- **テスト大事！**
  - チップのマニュアルをよく読もう（たまに間違っているけど）
  - コーナーケースのテストも忘れずに
  - 標準のテストケースだけでは検出できないバグもある

CMSIS-DAP インタフェースのポーティング

- **コードレポジトリ**
  - https://github.com/mbedmicro/cmsis-dap
- **mbed HDK firmware 内部構造とポーティング方法の解説**
  - http://mbed.org/handbook/cmsis-dap-interface-firmware
  - https://mbed.org/users/MACRUM/notebook/cmsis-dap-interface-firmware/ （日本語版）
- **mbed HDK リファレンス回路図、ロゴ**
  - http://mbed.org/users/mbed_official/code/mbed-HDK/
  - Download repository から、.zip ファイルをダウンロード
  - または、http://mbed.org/users/mbed_official/code/mbed-HDK/file/bfc970cfffa6/mbed-HDK.lbr （Eagle 用ベクタデータ）



ARM

CMSIS-DAP ファームウェアのビルド

- **Flashアルゴリズムのプロジェクトファイル**
  - {GitHub}¥CMSIS-DAP¥interface¥flash_algo_mdk¥LPC_IAP¥LPC_IAP.uvproj
  - ターゲットデバイスのRAMにダウンロードされて実行されるRWPI, ROPIコードが生成される
  - ビルドはMDK-ARM pro版が必用
- **Flashアルゴリズムの命令ストリームをデータ化する**
  - flash_algo.axf を {GitHub}¥CMSIS-DAP¥tools¥tmp にコピー
  - {GitHub}¥CMSIS-DAP¥tools¥flash_algo_gen.py を実行
  - flash_algo.txt が生成される
- **CMSIS-DAP ファームウェアのプロジェクトファイル**
  - {GitHub}¥CMSIS-DAP¥interface¥mdk¥lpc11u35¥lpc11u35_interface.uvproj
  - flash_algo.txt の内容を、target_flash.h にコピーする
  - 必用な部分を追記する（ベースポインタ、スタックポインタ初期値やバッファ位置）

ARM