

# 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 Cortex-M ファミリの概要



# Cortex-Mプロセッサ

幅広いアプリケーションに対応

## 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/DSP  
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  
使いやすさ

コンフィギュラブル  
小さいシリコンサイズ

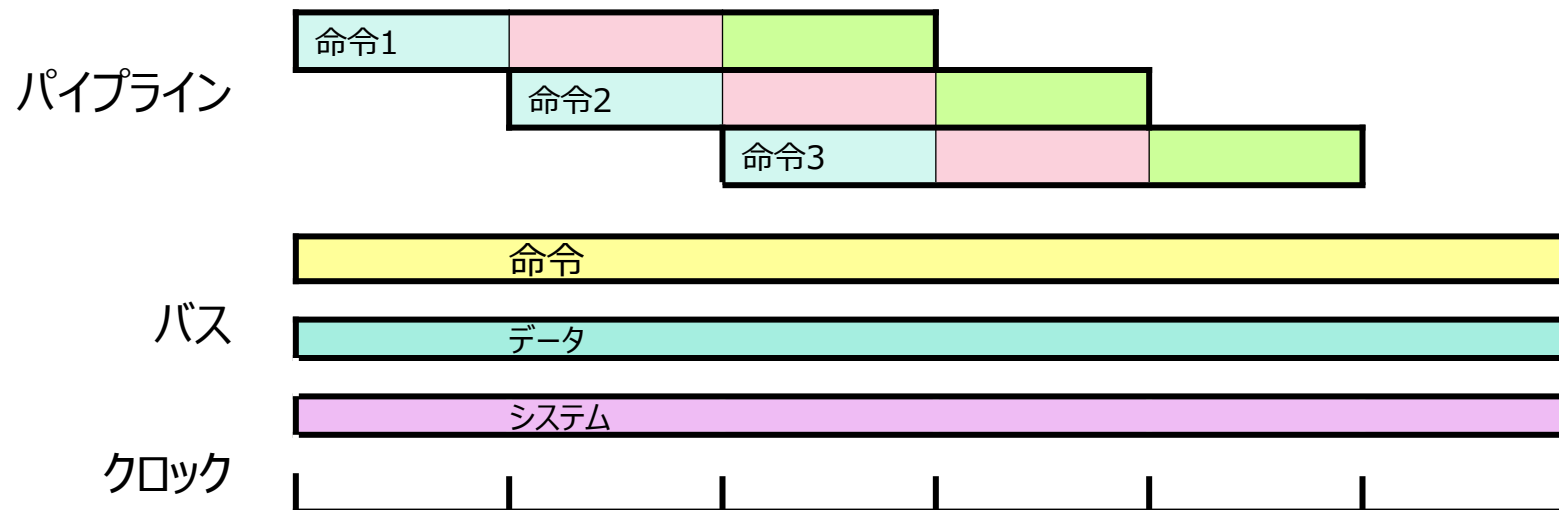
確定的動作

市場での実績

# ARM Cortex-M アーキテクチャのポイント

# 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<sup>®</sup>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<sup>™</sup>
- 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

<b>R0</b>	第1引数 / 返却値
<b>R1</b>	第2引数 / 返却値
<b>R2</b>	第3引数
<b>R3</b>	第4引数
<b>R4</b>	変数用
<b>R5</b>	変数用
<b>R6</b>	変数用
<b>R7</b>	変数用
<b>R8</b>	変数用
<b>R9</b>	変数用
<b>R10</b>	変数用
<b>R11</b>	変数用
<b>R12</b>	変数用
<b>R13(SP)</b>	スタックポインタ
<b>R14(LR)</b>	リンクレジスタ
<b>R15(PC)</b>	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		D0
S1		D1
S2		D2
S3		D3
S4		D4
S5		D5
S6		D6
S7		D7
S8		D8
S9		D9
S10		D10
S11		D11
S12		D12
S13		D13
S14		D14
S15		D15
S16		
S17		
S18		
S19		
S20		
S21		
S22		
S23		
S24		
S25		
S26		
S27		
S28		
S29		
S30		
S31		

- 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



# レジスタの利用方法 (1)

関数の引数または  
関数からの戻り値  
その他は破壊可能  
(追加された引数は、  
スタックに渡される)

## Register

r0	★
r1	★
r2	★
r3	★

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。

レジスタ変数  
保持される必要がある

r4
r5
r6
r7
r8
r9
r10
r11

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。

スクラッチレジスタ  
(破壊可能)

r12	★
-----	---

スタックポインタ  
リンクレジスタ  
プログラムカウンタ

r13/sp	★
r14/lr	★
r15/pc	★

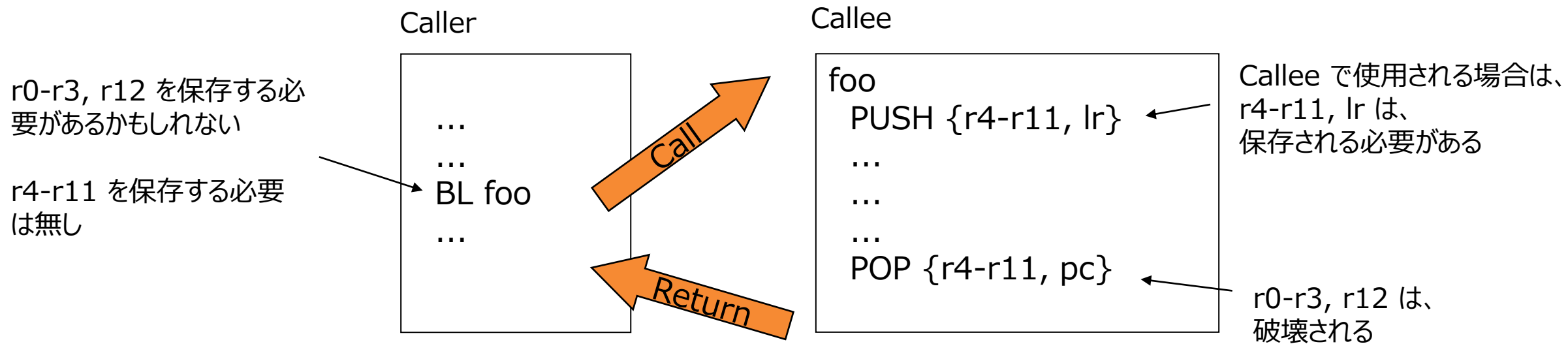
- sp は、8-byte (2-words) アラインされている
- r14 は、値がスタックに積まれている場合は、  
テンポラリ領域として使用可能。

プログラムステータス

xPSR	★
------	---

# 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

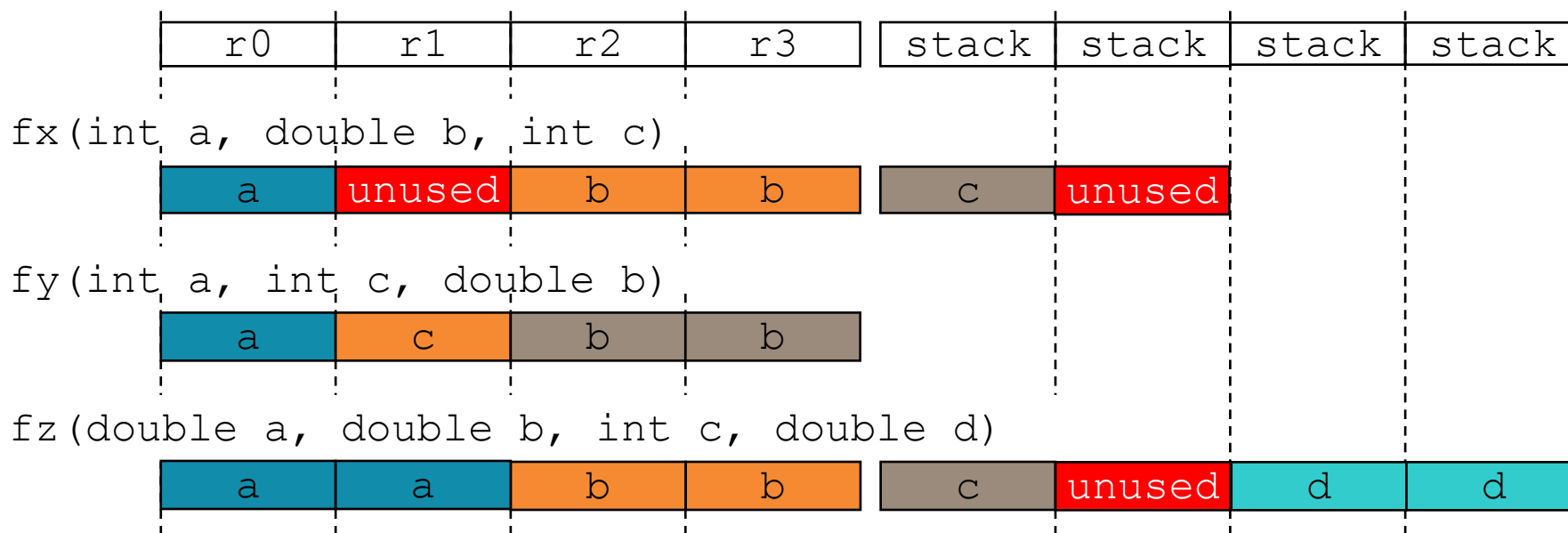
AAPCS – Procedure Call Standard for ARM Architecture

# 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要な場合、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

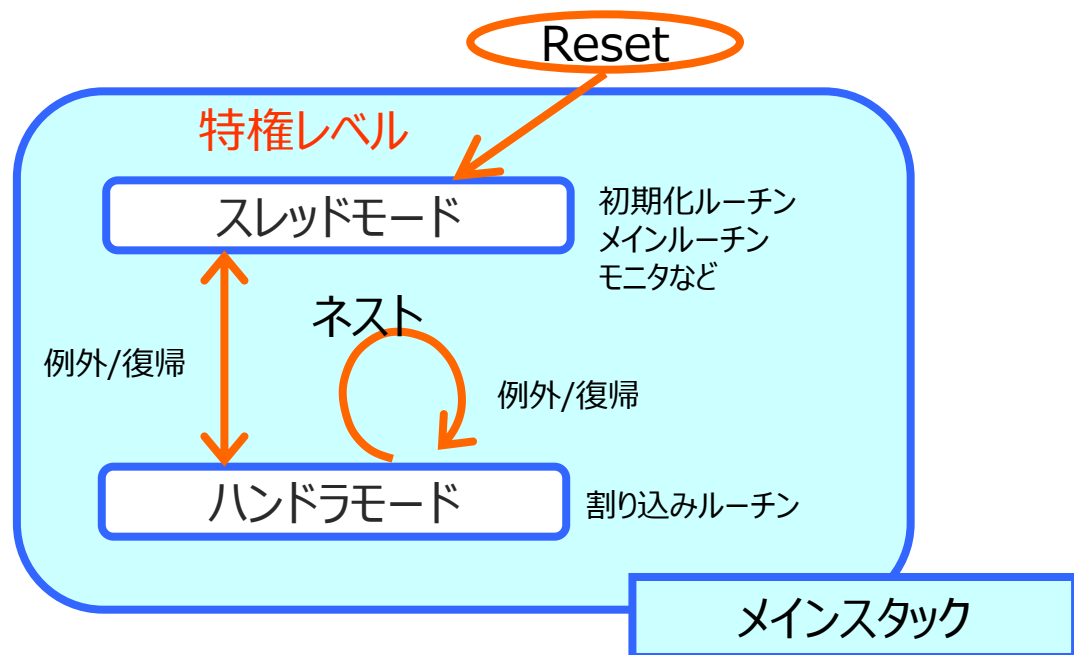
## 関数パラメータ（引数）について（2）

- 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

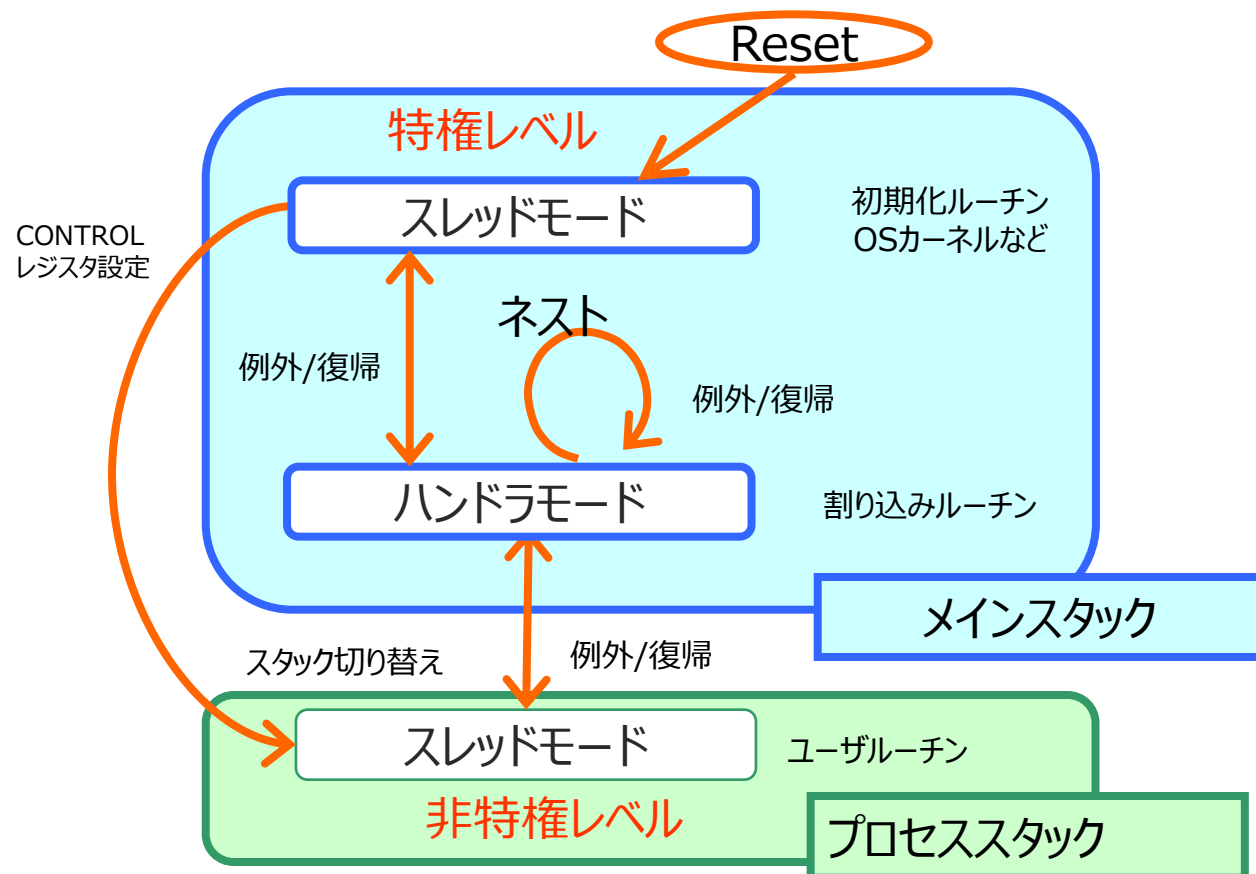


# 実行モード

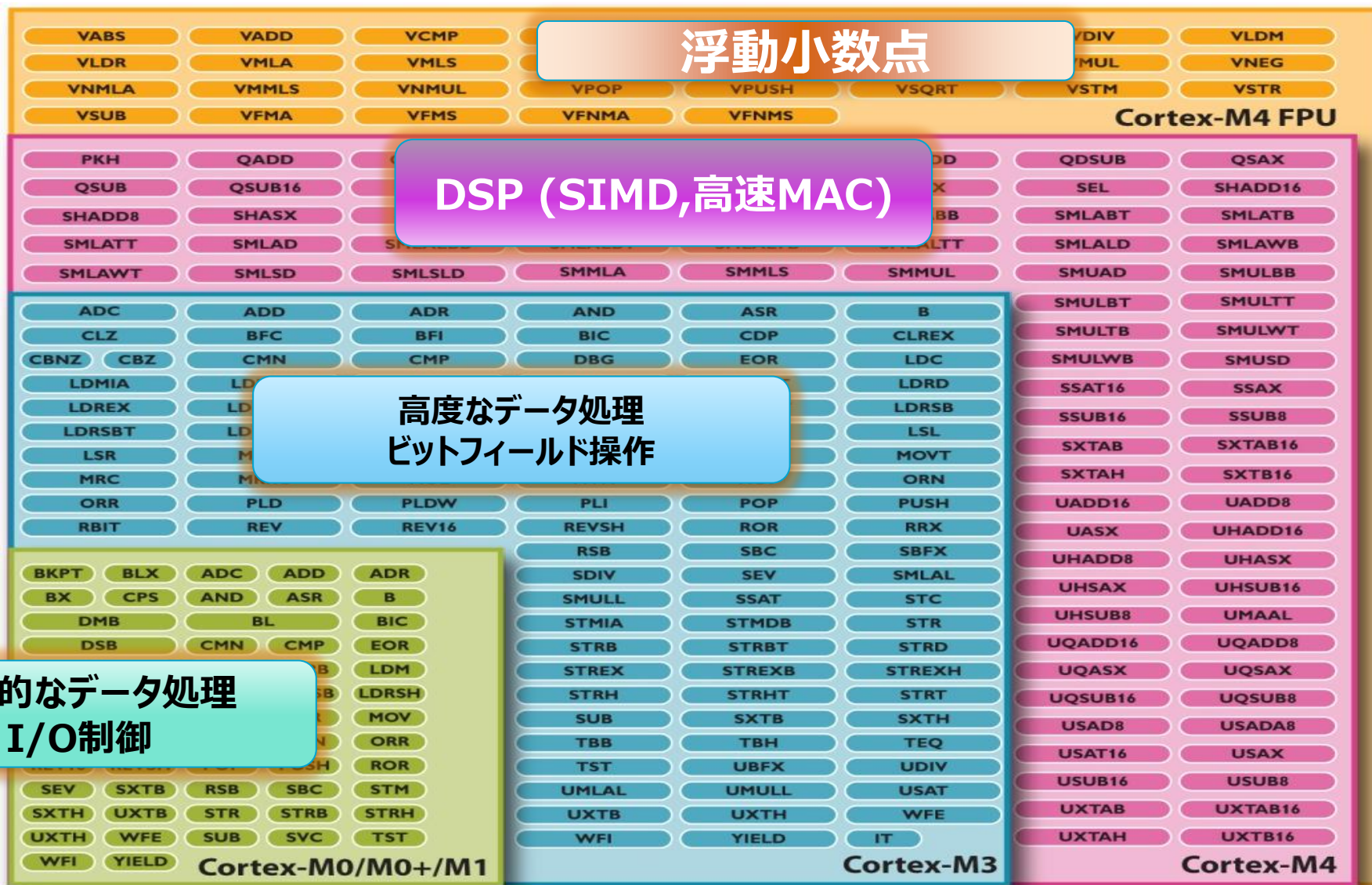
- 基本的なシステム
  - 一つのスタックでシンプルな構成



- 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



# スケーラブルな命令セット



# 命令セット全体

命令分類	命令
データ処理命令	<b>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</b>
分岐命令、実行制御命令	<b>B, BL, BLX, BX, TBB, TBH, BEQ, CBZ, CBNZ, IT</b>
メモリアクセス命令	<b>LDR, STR, LDM, STM, PUSH, POP, LDREX, STREX</b>
例外生成命令	<b>SVC, BKPT</b>
特殊レジスタアクセス命令	<b>MRS, MSR, CPSIE, CPSID</b>
メモリバリア命令	<b>DMB, DSB, ISB</b>
同期命令、その他	<b>SEV, WFI, WFE, NOP</b>
DSP命令(Cortex-M4)	<b>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</b>
浮動小数点命令(Cortex-M4 FPU付き)	<b>VABS, VADD, VCOMP, VCMPE, VCMPE, VCVT, VCVTR, VCVTB, VCVTT, VFMA, VFNMA, VFMS, VFNS, VLDM, VLDR, VLMA, VLMS, VMOV, VMRS, VMSR, VMUL, VNEG, VNMLA, VNMLS, VNMUL, VPOP, VPUSH, VSQRT, VSTM, VSTR, VSUB</b>

※太字はCortex-M0/M0+の命令



# 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0xXYのオペランドの8ビットで表現できる
- 0x00XY00XY、 0xXY00XY00、 0xXYXYXYXY
- 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



# 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
<pre>int func(int *p, int num) {     return p[num]; }</pre>	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

# 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)  $A = B \mid 0x3C \dots$  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

# 分岐 – 効率のためには少ないほど良い

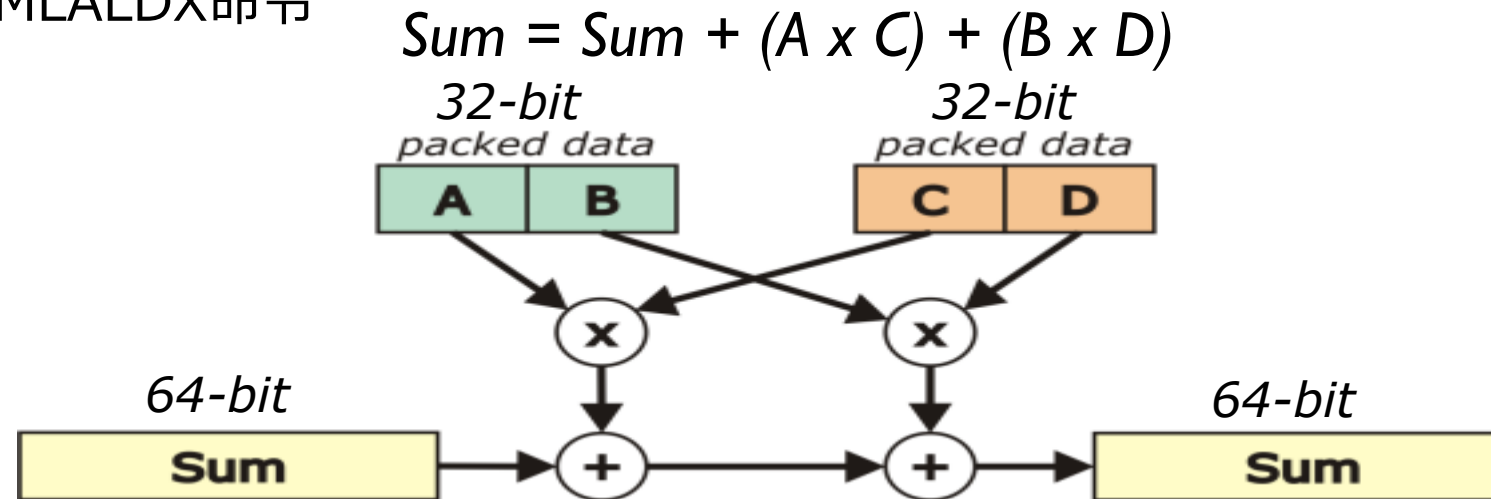
- 条件分岐
  - M0/M0+: +/- 254バイト
  - M3/M4: -1048576 から +1048574 バイト(20ビット) ;32ビット条件分岐命令の場合
- 無条件分岐
  - M0/M0+: +/- 2Kバイト
  - M3/M4: +/- 16MB ; 32ビット分岐命令の場合
- 間接分岐
  - レジスタの値のアドレスに分岐する(4GB空間,0xffffffff0-0xffffffffを除く)
  - 例) BX r0           … r0のアドレスに分岐
- その他
  - M3/M4は条件実行(IT, if-then), 比較分岐(CBZ, CBNZ)、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

# 四則演算

- 加減算 – 1サイクル
- 乗算
  - M0/M0+ は  $32b \times 32b = 32b$
  - M3/M4は  $32b \times 32b = 32b$ ,  $32b \times 32b = 64b$
- 除算
  - M0/M0+ : ソフトウェア(ライブラリ)で行う
  - M3/M4: 除算命令のハードウェアで行う(2~12サイクル)
- 積和 – 乗算結果を累積する
  - M3/M4:  $32b \times 32b + 32b = 32b$
  - 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

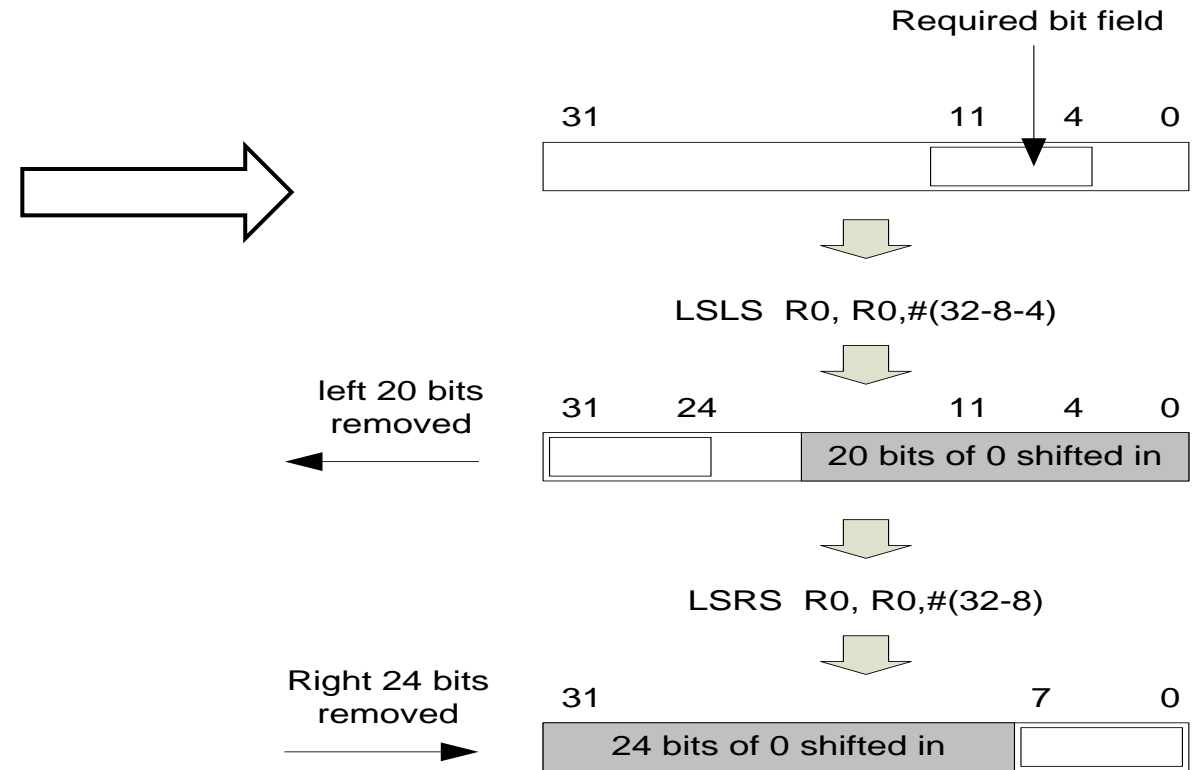
- SIMD演算は同じ複数の演算を1サイクルで実行
  - SIMD = Single Instruction Multiple Data
  - 例) SMLALDX命令



- 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

# 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



# 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)

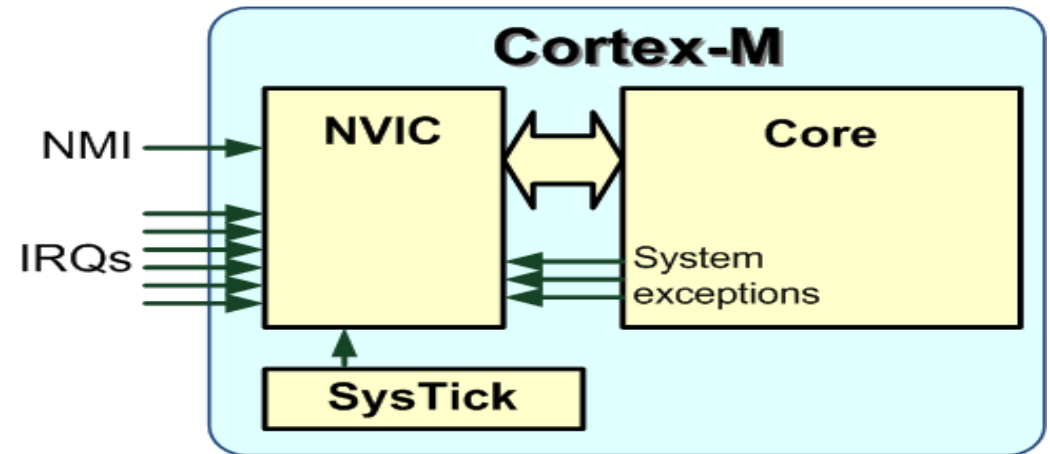
# 標準化された例外・割り込み管理

## ■ 標準化された割り込みコントローラ(NVIC)

- ネスト可能なベクタ方式の割り込みコントローラ
- 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アドレス
- システムタイマ自体もコアに装備

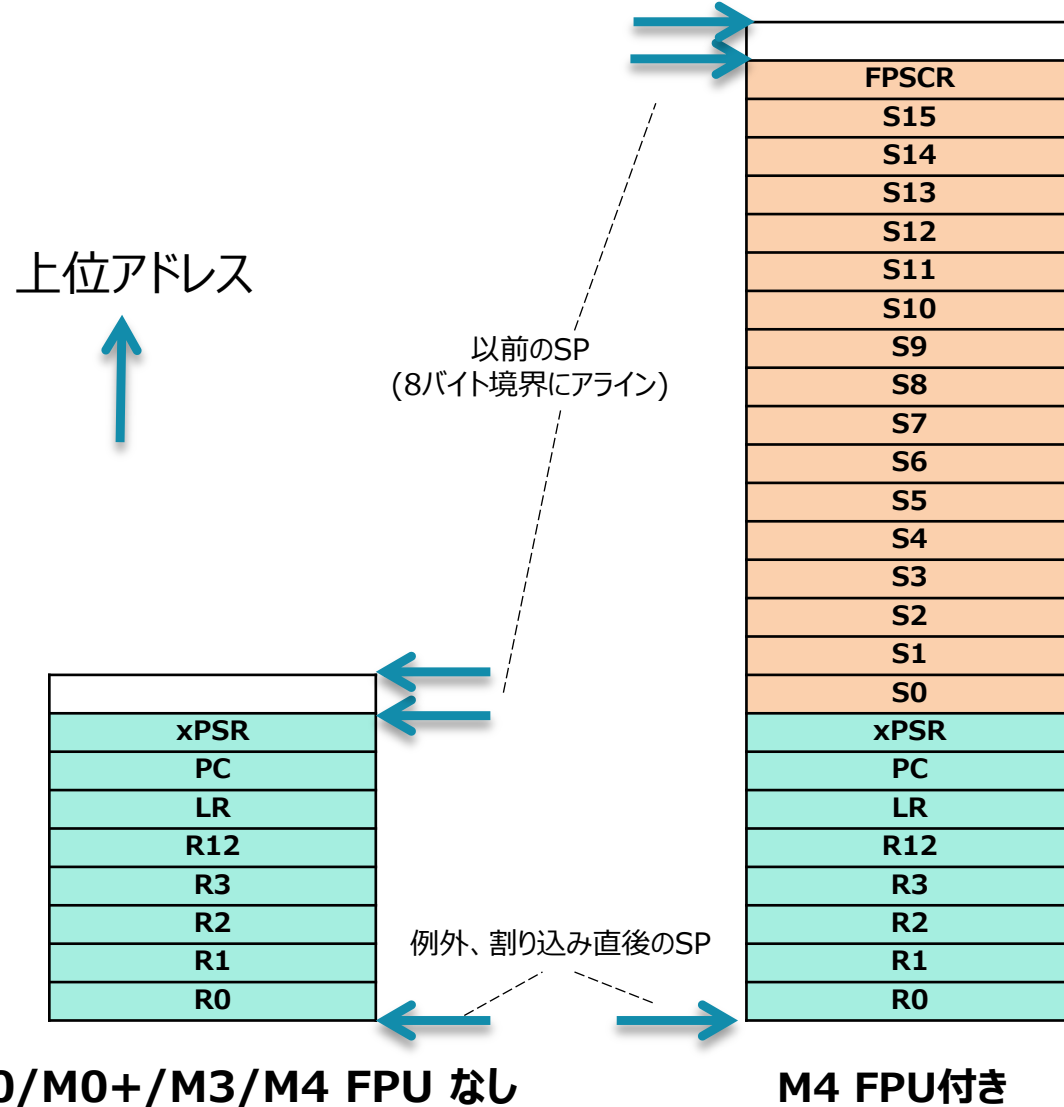
## ■ 割り込み処理高速化の工夫

- 割り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





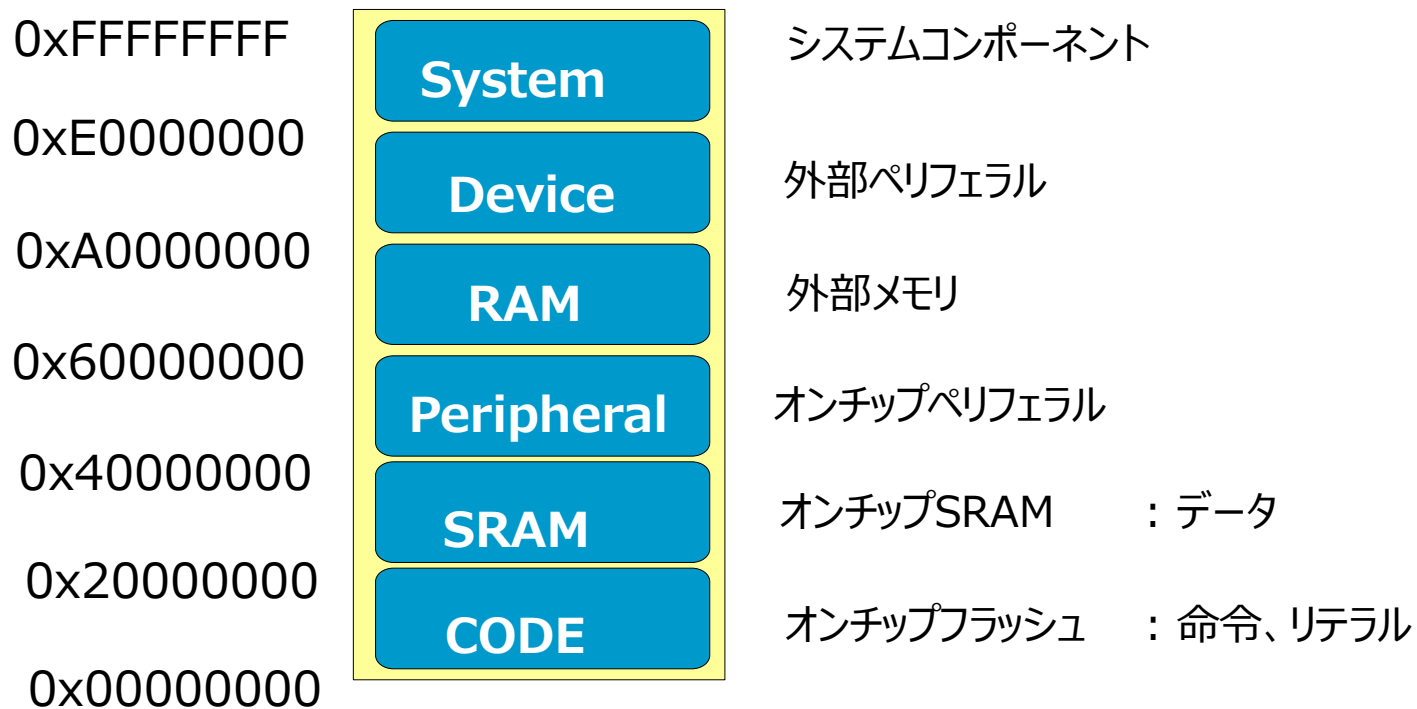
# 例外・割り込み時に保存されるレジスタ



- 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

# 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

# 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： $0x42000000 + (A - 0x40000000) \times 32 + B \times 4$

SRAMの場合： $0x22000000 + (A - 0x20000000) \times 32 + B \times 4$

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  
(ハードウェアでRead-Modify-Writeが行われる)

通常処理：

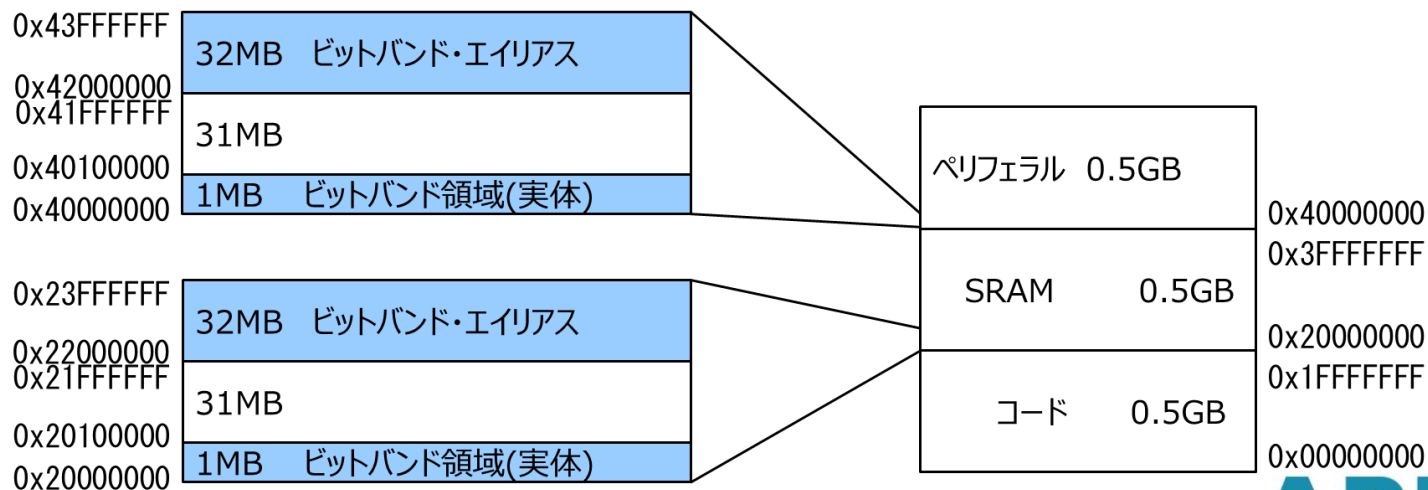
```
x = x & 0xFFFFFFFFE;
```

```
x = x | 0x00000001;
```

ビットバンド処理：

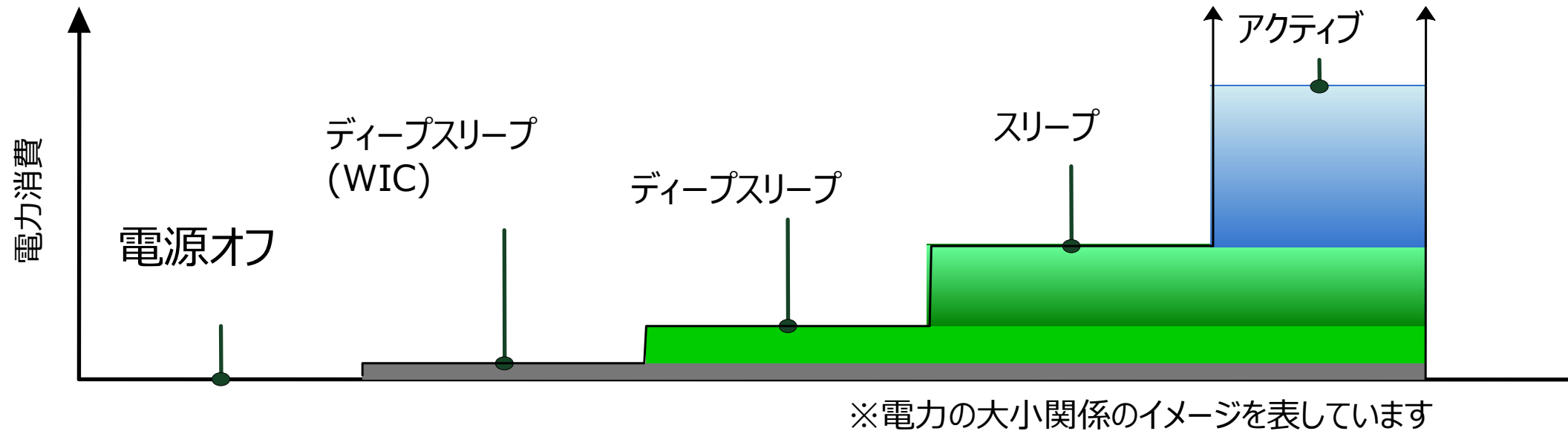
```
y = 0;
```

```
y = 1;
```



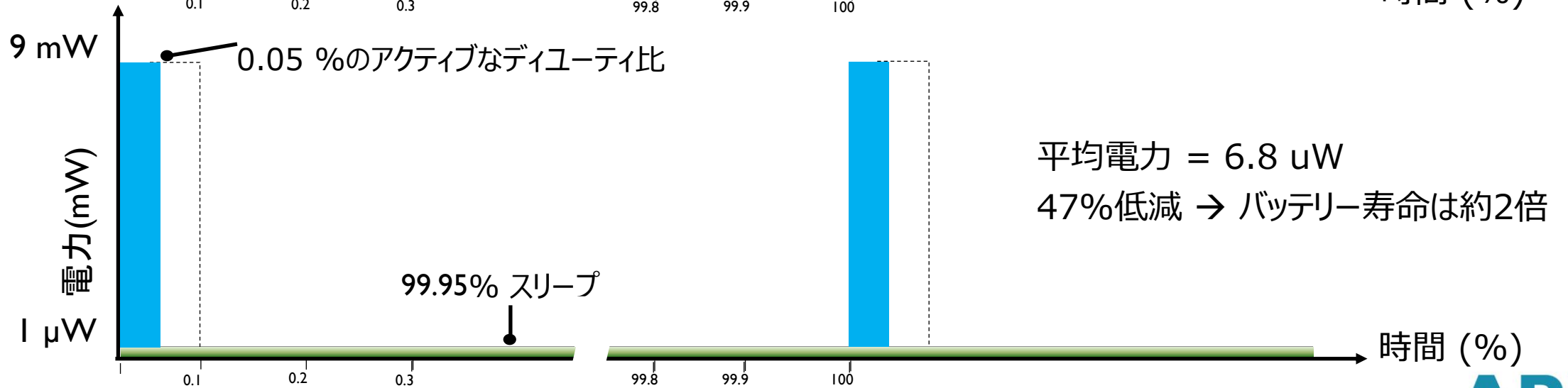
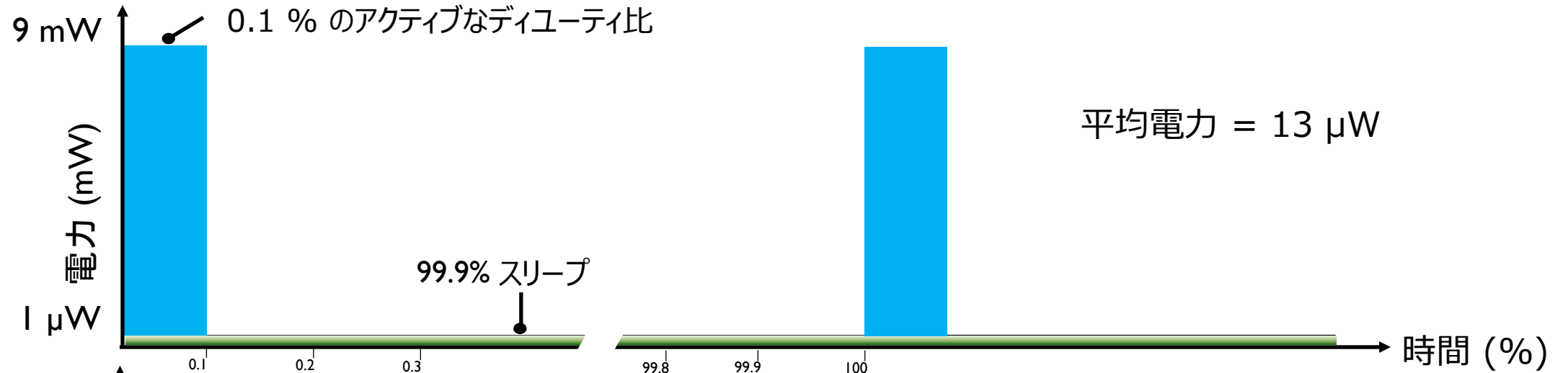
# Cortex-Mのスリープモード

- 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



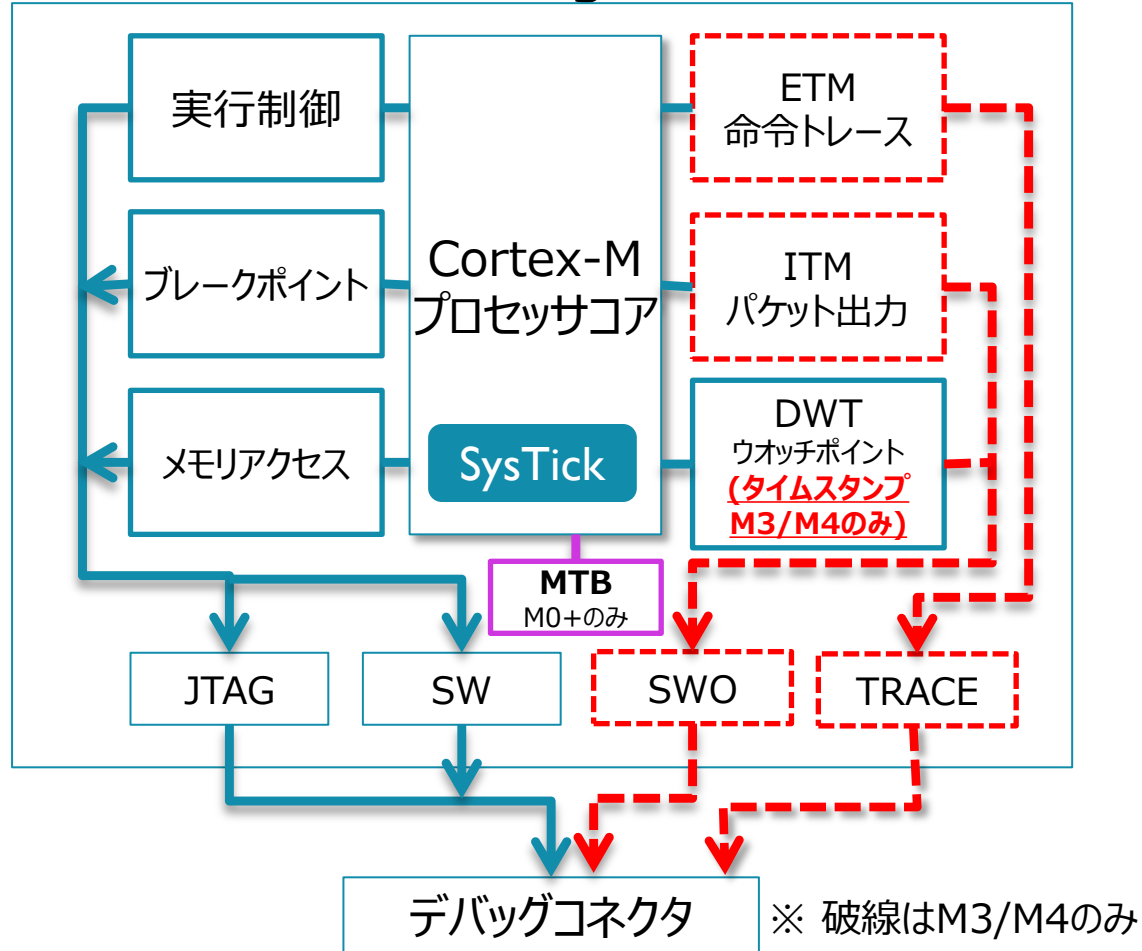
# 短い実行サイクルほど長いバッテリー寿命

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るとすると ...



# 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		○		

# ARMv6-M と ARMv7-M の違い

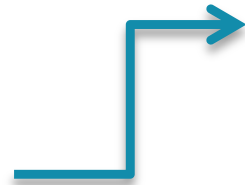
- インストラクションセット
  - ARMv6-M は、16-bit Thumb命令と幾つかの追加の命令セット
  - LDREX/STREX 命令やアトミックなスワップ命令は未サポート
  - STM/LDM命令での割り込み継続は未サポート
  - ARMv6-M は、非アラインワードまたはハーフワードアクセスには未対応
- モードと特権
  - 特権はオプションで拡張可能
  - 上記オプションを使用しない場合 ARMv6-M コアは、常に特権で動作する（例：Cortex-M0）
- 例外処理
  - ベクタテーブルのベースアドレスは設定不可能（実装定義）
  - いくつかの優先レベルのみ対応、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
  - デバッグはホールドモードのみ対応（プログラム実行時のメモリ書き換え等は不可）

# 効率的なプログラミングのアプローチ



# 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

# 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 = 0xFFFFFFFF; // 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; // 現在のカウンタ値を読み出し
    CNT = 0xFFFFFFFF - CNT - 4; // 経過サイクル数演算、4はオーバーヘッド、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ー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

# 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

The screenshot shows the uVision IDE interface. On the left, the 'Internal' register view is expanded, showing the 'States' register with a value of 22333. A blue arrow points to the 'States' register. On the right, the source code is displayed, showing a function `wr_dat` that writes data to the LCD controller. The code includes comments and a function definition. A green highlight is visible in the code area.

```
118  
119  
120  
121  
122 /*****  
123 * Write data to the LCD controller *  
124 * Parameter: dat: data to be written *  
125 * Return: *  
126 *****/  
127  
128 static inline void wr_dat (unsigned short dat) {  
129     LCD_CS(0);  
130     spi_tran(SPI_START | SPI_WR | SPI_DATA); /* Write : RS = 1, RW = 0 */  
131     spi_tran((dat >> 8)); /* Write D8..D15 */  
132     spi_tran((dat & 0xFF)); /* Write D0..D7 */  
133     LCD_CS(1);  
134 }  
135  
136
```

- 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

# 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

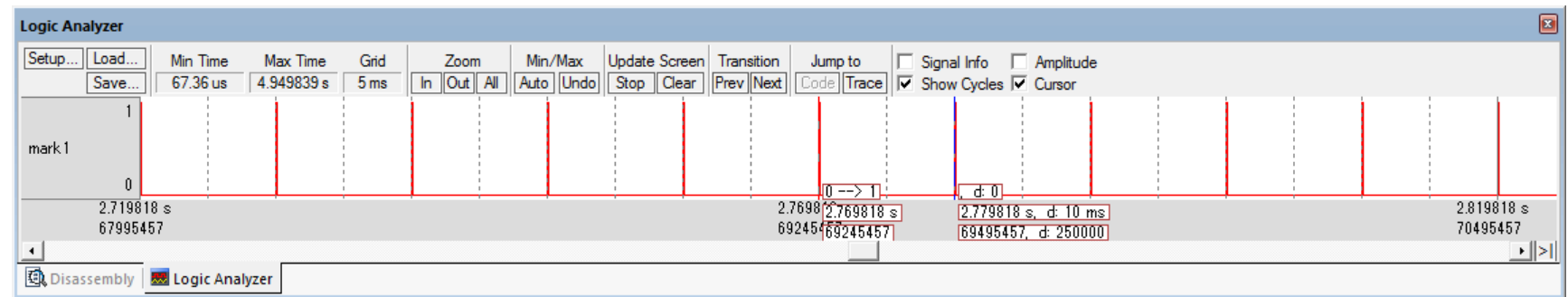
```
#define MARK_ON 1
#define MARK_OFF 0
uint32_t mark1=MARK_OFF;

void SysTick_Handler(void) {
    int i,s;

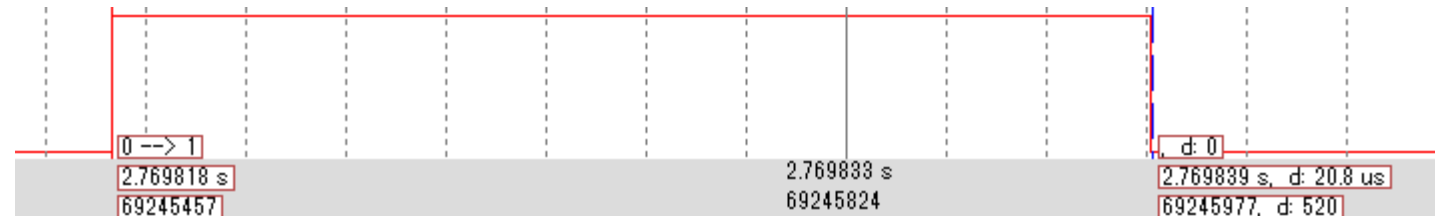
    mark1 = MARK_ON;

    <処理>

    mark1 = MARK_OFF;
}
```



拡大



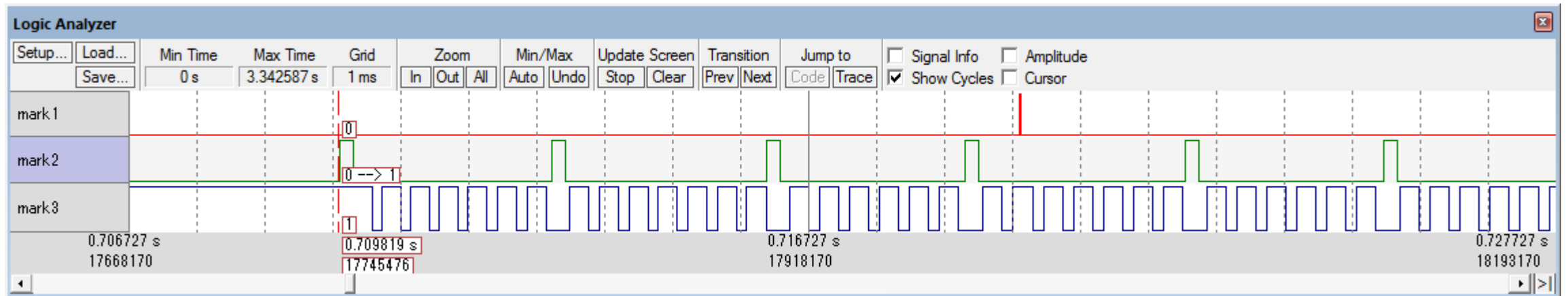
# 応答サイクル数、ハンドラの処理サイクル数の効率的な測定

- 測定対象のハンドラをソフトウェ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
  - ソフトウェ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 } // ソフトウェア例外の発生
    }
}
void Int20_Handler(void) // 本来の例外ハンドラ
{
    ...
}
```

# 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

The screenshot shows the Performance Analyzer window with the 'Functions' view selected. The table below represents the data shown in the window:

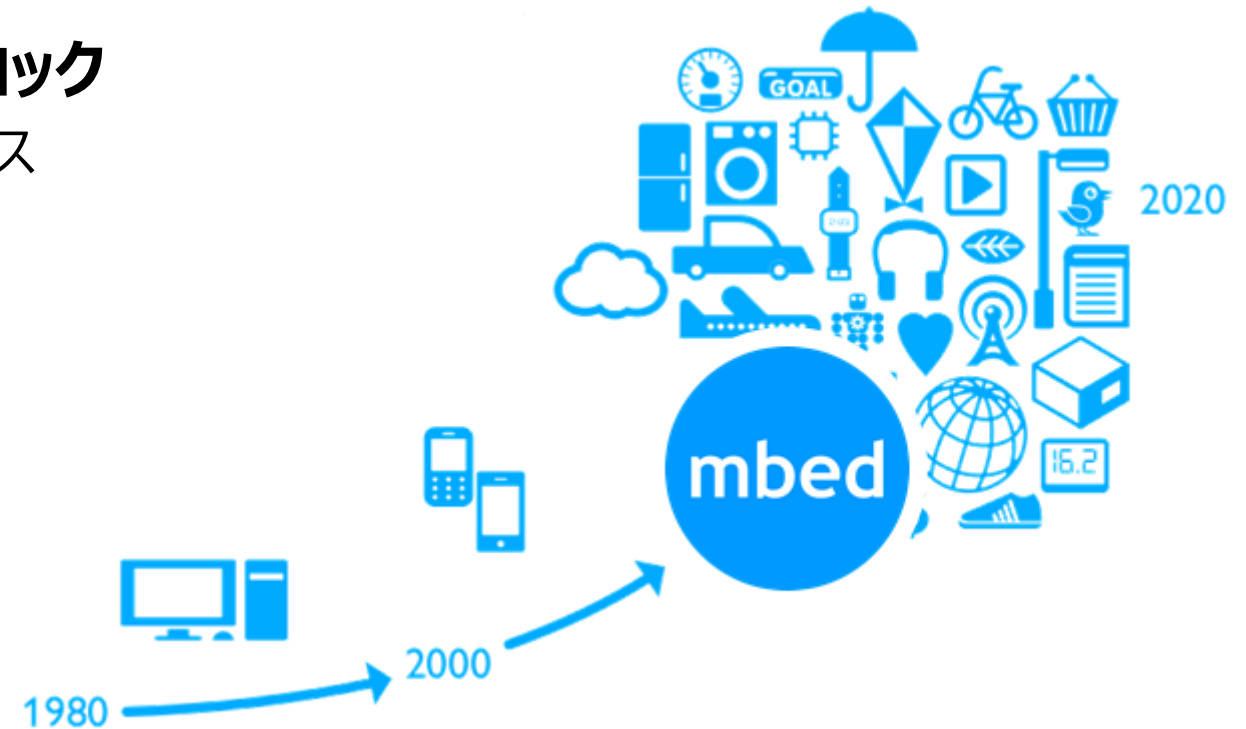
Module/Function	Calls	Time(Sec)	Time(%)
func2	6961	1.954 s	49%
main	1	1.091 s	27%
spi_tran	227818	615.644 ms	15%
func1	1055	211.659 ms	5%
wr_dat_only	111360	58.119 ms	1%
GLCD_Clear	1	17.106 ms	0%
GLCD_DrawChar	90	12.440 ms	0%
SysTick_Handler	326	6.844 ms	0%
delay	7	2.706 ms	0%
wr_cmd	880	1.513 ms	0%
wr_dat	789	1.129 ms	0%
wr_reg	789	504.960 us	0%
GLCD_SetWind...	91	244.680 us	0%
GLCD_Display...	90	182.440 us	0%



# mbed の特徴

# 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



# マイコン開発を始めるときの課題

## ■ 多機能なデバイス

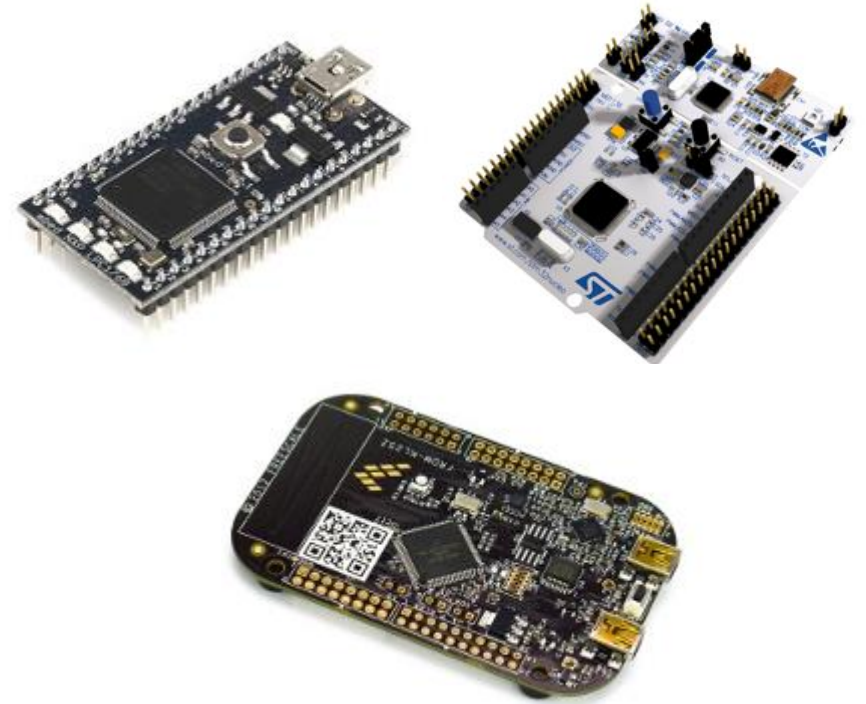
- ARM コアは知っている・・・だけどペリフェラルが各社独自で複雑
- 1,000 ページのドキュメントとアプリケーションノート、しかも英語マニュアルのみ？！
- 設定するまでに一苦労。自分で全部コード書くの？
- 苦労して作ったコードは、別のデバイスを使ったらポーティングやり直し・・・

## ■ 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 には色々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・・・

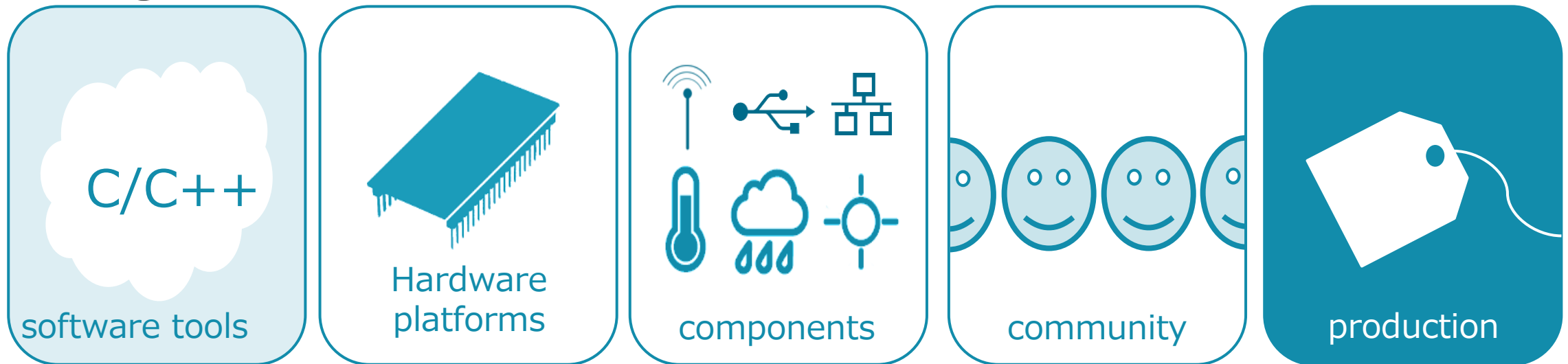
# mbed の特徴

- 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



# 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*



# ハードウェアプラットフォームと HDK

# mbed-enabled プラットフォーム

mbed  
Enabled

- 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



mbed LPC1768



mbed LPC11U24



Freescale KL25Z



NXP LPC800-MAX



EA LPC4088  
QuickStart Board



Seeeduino-Arch



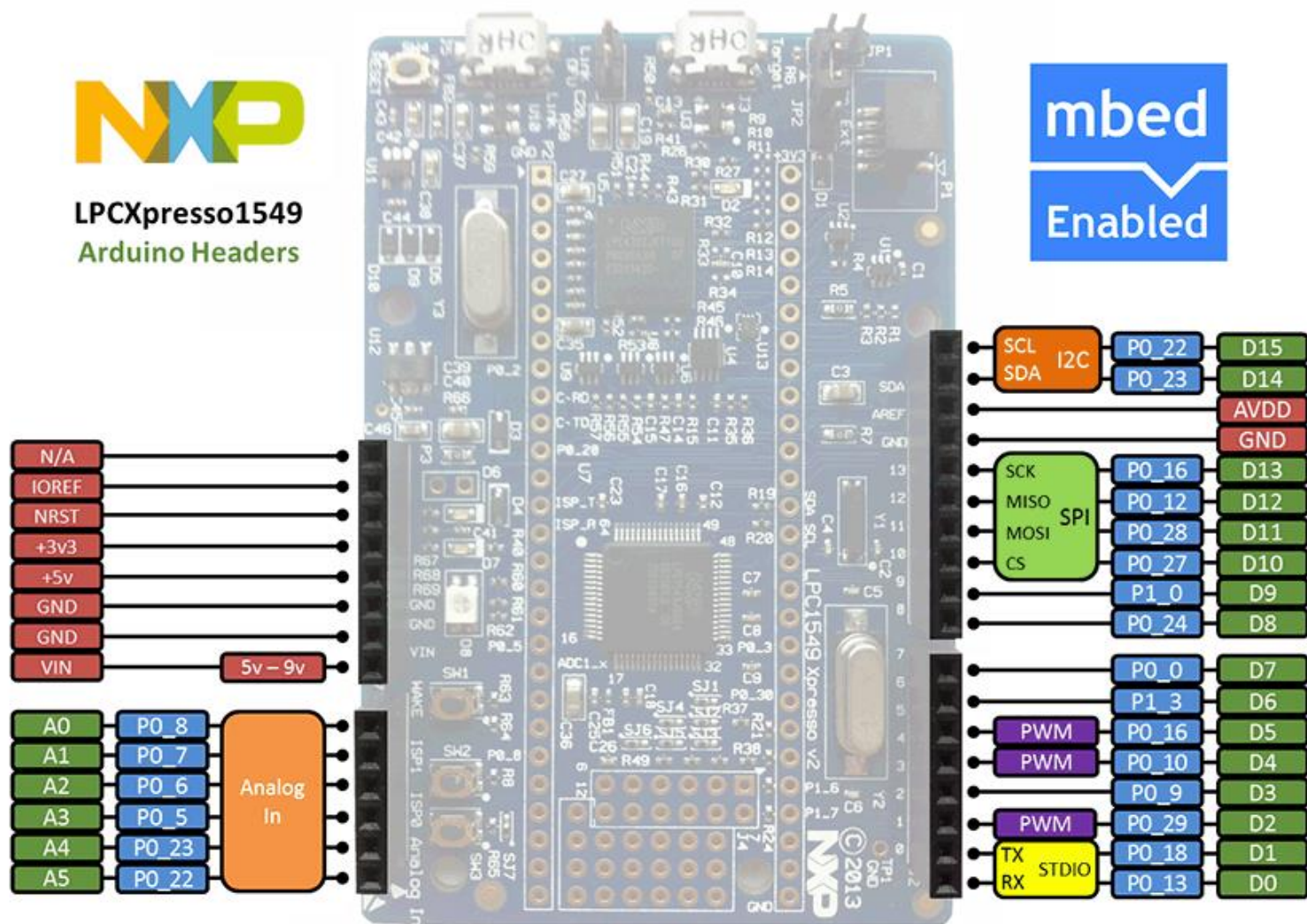
LPC1114FN28



u-blox-C027

# 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マストレージドライブ**  
として見える
- **DIP 28ピンもあります！**

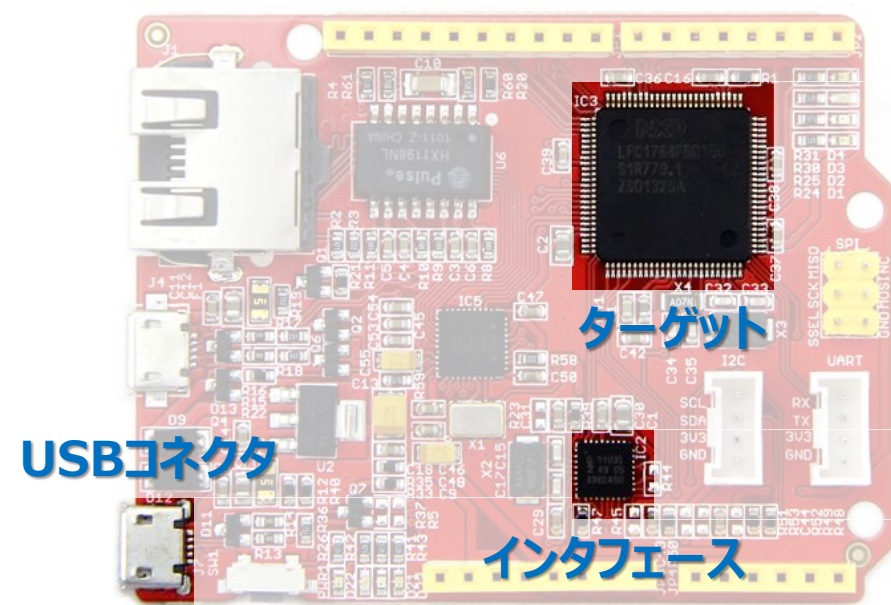




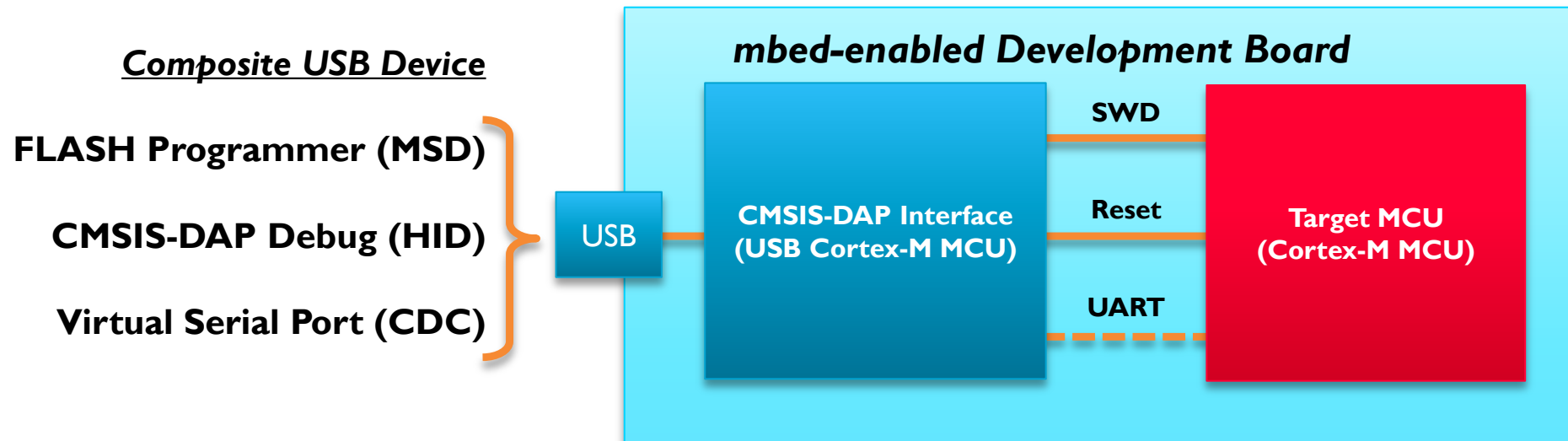
# Target と Interface

- 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 ツールチェーンでのフルデバッグ



# mbed HDK

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

# mbed ソフトウェアブロック図



## ■ 開発者のための設計

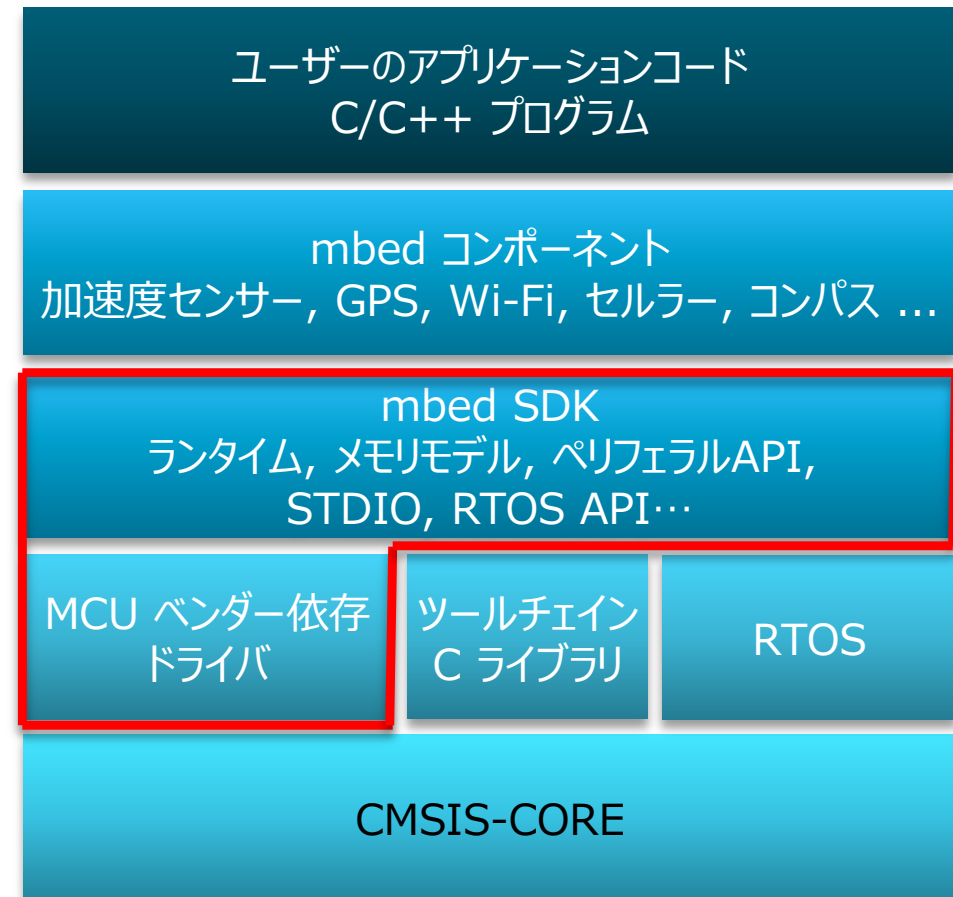
- High-level API と標準化された環境
- 異なるベンダーのデバイス間でポータビリティを確保
- 商用、非商用どちらでも使用可能
- オープンソース - Apache 2.0

## ■ 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ーンに対応
- 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

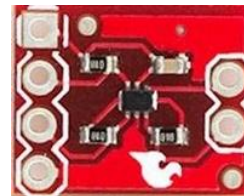
DigitalOut クラスのコンストラクタ

指定された GPIO ポートに出力

```
main.cpp X
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
```

# 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

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

```
66 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     */
76 00076     int read() {
77         return gpio_read(&gpio);
78     }
79
80 #ifndef MBED_OPERATORS
81     /** A shorthand for write()
82     */
83 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     */
95 00095     operator int() {
96         return read();
97     }
98 #endif
99
```

# DigitalOut クラスの使用例

```
main.cpp x
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
```



# 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**

# 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);
    }
}
```

# クラウド開発環境

# 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

The screenshot shows the mbed.org homepage with a navigation bar at the top containing links for Platforms, Components, Handbook, Cookbook, Code, Questions, and Forum. On the right side of the navigation bar are links for Dashboard and Compiler. Below the navigation bar is the mbed logo, a search bar with the placeholder text "Search mbed.org...", and a "Go" button. To the right of the search bar is a green button labeled "Login or signup". Below the search bar are four blue arrow-shaped buttons labeled "Explore", "Getting Started", "Prototype", and "Production".

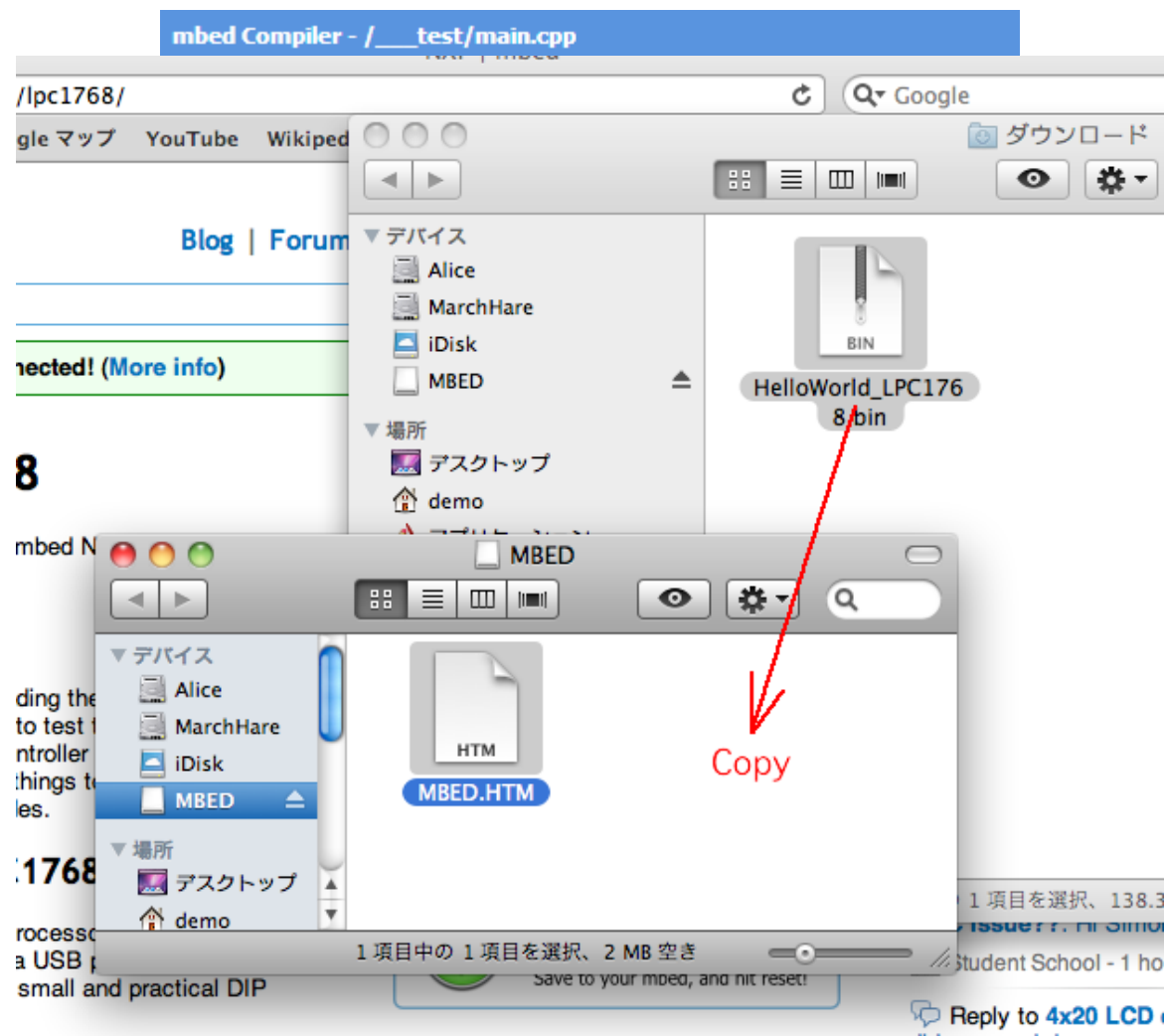
The main content area features the heading "Development Platform for Devices" followed by the text: "The mbed development platform is the fastest way to create products based on ARM microcontrollers." Below this is another paragraph: "The project is being developed by ARM, its Partners and the contributions of the global mbed Developer Community." At the bottom of this section is a link: "Find out why you should base your next ARM microcontroller powered product on the mbed platform »".

A featured product advertisement for the u-blox C027-C20/U20/G35 is shown. It includes the u-blox logo, the product name "C027-C20/U20/G35", and an image of the hardware. A blue callout box points to the hardware with the text "Cellular communication with mbed APIs". A blue circle with the mbed logo and a signal icon is also present. Below the advertisement are four small circles, with the last one being filled, indicating it is the current slide in a carousel.

At the bottom of the page is a row of partner logos: NXP, freescale, ST life.augmented, NORDIC SEMICONDUCTOR, u-blox, Embedded Artists, and seed studio.

# 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



# ソースコード管理

## ■ コードレポジトリ

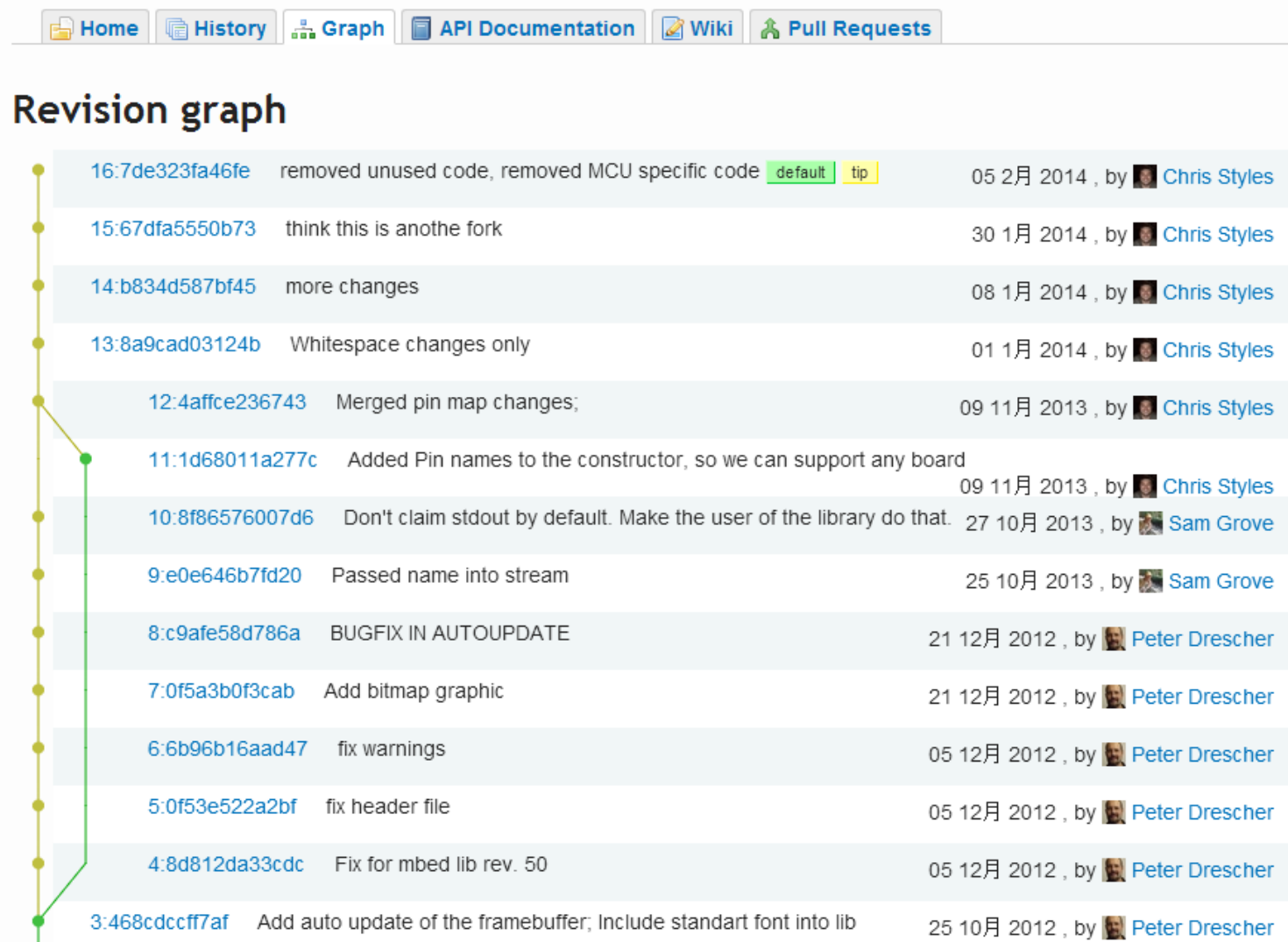
- コラボレーション・ワークフロー
- IDEに機能を統合

## ■ 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



# 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

## Constructor & Destructor Documentation

The screenshot displays a code editor with the following content:

```
1 #include "x
2 /**
3 *
4 *
5 *
6 *
7 *
8 *
9 *
10 *
11 *
12 *
13 *
14 *
15 *
16 *
17 *
18 *
19 *
20 *
21 *
22 *
23 *
24 *
25 *
26 *
27 *
28 *
29 *
30 *
31 *
32 *
33 *
34 *
35 *
36 *
37 *
38 *
39 *
40 *
41 *
42 *
43 *
44 *
45 *
46 *
47 *
48 *
49 *
50 *
51 *
52 *
53 *
54 *
55 *
56 *
57 *
58 *
59 *
60 *
61 *
62 *
63 *
64 *
65 *
66 *
67 *
68 *
69 *
70 *
71 *
72 *
73 *
74 *
75 *
76 *
77 *
78 *
79 *
80 *
81 *
82 *
83 *
84 *
85 *
86 *
87 *
88 *
89 *
90 *
91 *
92 *
93 *
94 *
95 *
96 *
97 *
98 *
99 *
100 *
```

Documentation for `GT20L16J1Y_FONT ( )`:  
Default constructor.  
Definition at line 4 of file `GT20L16Y1J_font.cpp`.

en 129 / **LEDMatrix\_Master**

LEDMatrixDisplay Program

Dependencies: mbed

Home History Graph API Documentation Wiki Pull Requests

### All pull requests

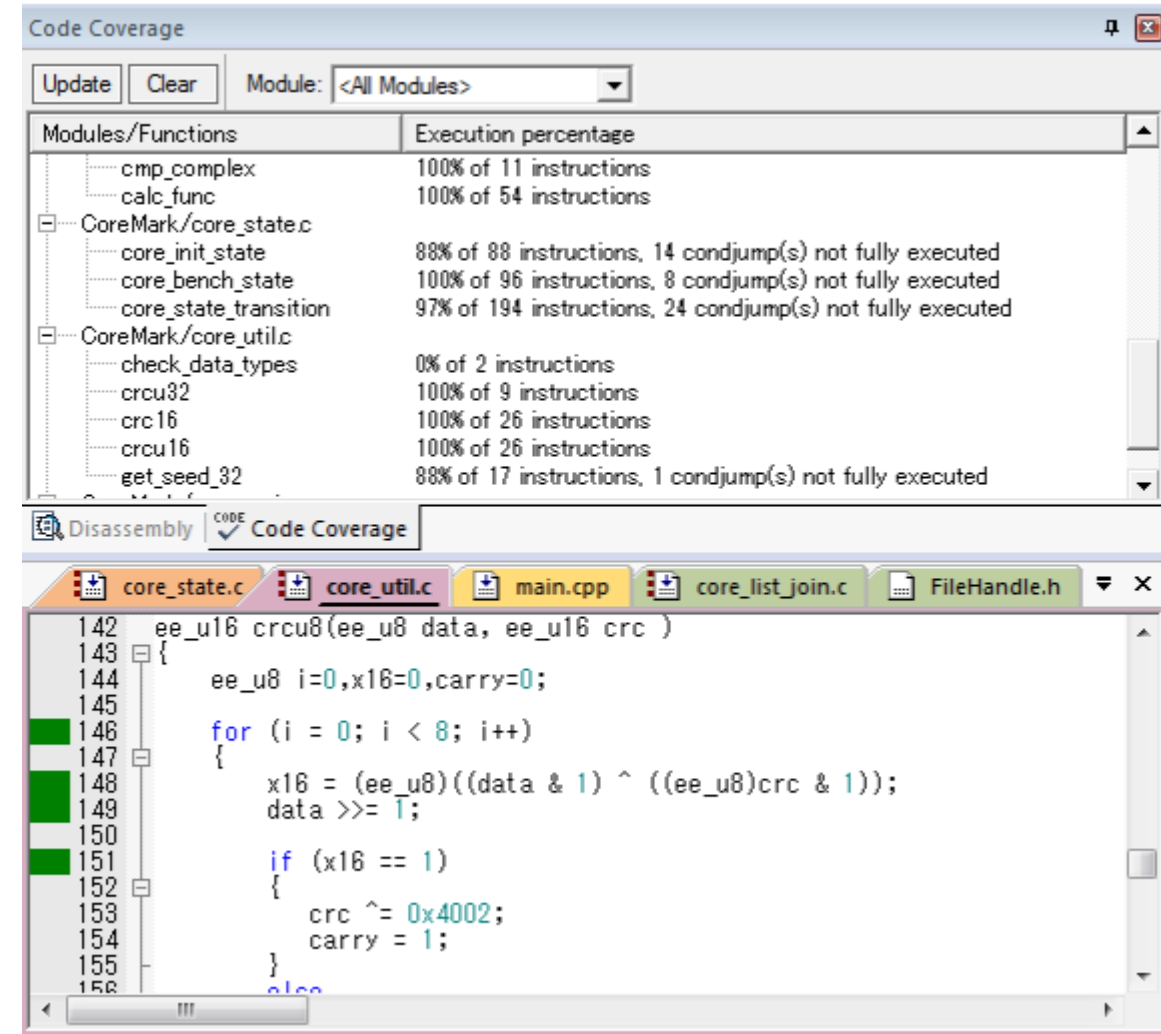
Show Open pull requests

- Toyomasa Watarai LEDMatrix\_Master / **Fixed text length issue.** Last updated: 04 12月 2013  
**Accepted**
- Toyomasa Watarai LEDMatrix\_Master / **Message file read and ascii code support** Last updated: 16 11月 2013  
**Accepted**

-----  
unsigne cs Chip enable input pin to connect to  
Definition at line 22 of file `GT20L16J1Y_font.h`.

# 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



The screenshot displays a 'Code Coverage' window with a table of execution percentages for various modules and functions. The table is as follows:

Modules/Functions	Execution percentage
cmp_complex	100% of 11 instructions
calc_func	100% of 54 instructions
CoreMark/core_state.c	88% of 88 instructions, 14 condjump(s) not fully executed
core_init_state	100% of 96 instructions, 8 condjump(s) not fully executed
core_bench_state	97% of 194 instructions, 24 condjump(s) not fully executed
core_state_transition	
CoreMark/core_util.c	
check_data_types	0% of 2 instructions
crcu32	100% of 9 instructions
crc16	100% of 26 instructions
crcu16	100% of 26 instructions
get_seed_32	88% of 17 instructions, 1 condjump(s) not fully executed

Below the table, the 'Disassembly' and 'Code Coverage' tabs are visible. The 'Code Coverage' tab is active, showing a code editor with the following code snippet:

```
142 ee_u16 crcu8(ee_u8 data, ee_u16 crc )
143 {
144     ee_u8 i=0,x16=0,carry=0;
145
146     for (i = 0; i < 8; i++)
147     {
148         x16 = (ee_u8)((data & 1) ^ ((ee_u8)crc & 1));
149         data >>= 1;
150
151         if (x16 == 1)
152         {
153             crc ^= 0x4002;
154             carry = 1;
155         }
156     }
157 }
```



# 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

Components » Sensors » Temperature » TMP102 Temperature Sensor

## TMP102 Temperature Sensor

small SOT563 package, with a 0.0625C

**Datasheet**  
<http://www.ti.com/litv/pdf/sbos397b>

**Notes**

**Pinout**

TMP102	mbed
1 - Vcc (square pad)	Vout
2 - SDA	p9
3 - SCL	p10
4 - Gnd	Gnd

Available on a breakout board from [Sparkfun](#)

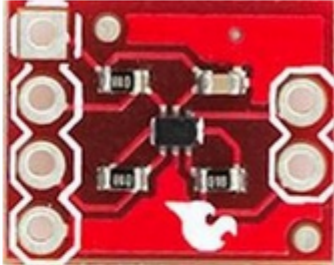
[Ask a question](#)

[Import program](#)

can be imported independently

[Import library](#)

independent



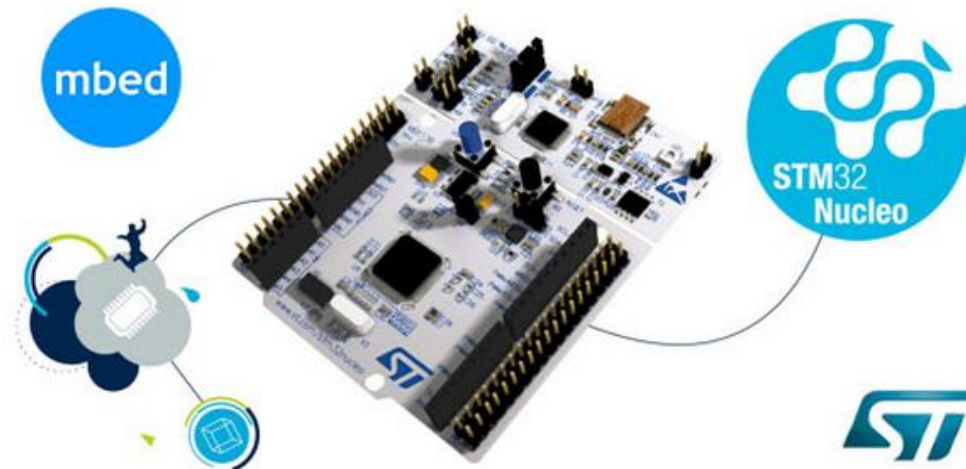
# 簡単だけど、それだけじゃない

- **簡単につかえる ≠ 単機能**
- **開発環境**
  - mbed SDK の API 以外にも当然使える（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

# 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

## STM32 Nucleo



## LPCXpresso1549

Arduino Headers  
LPC-Link 2

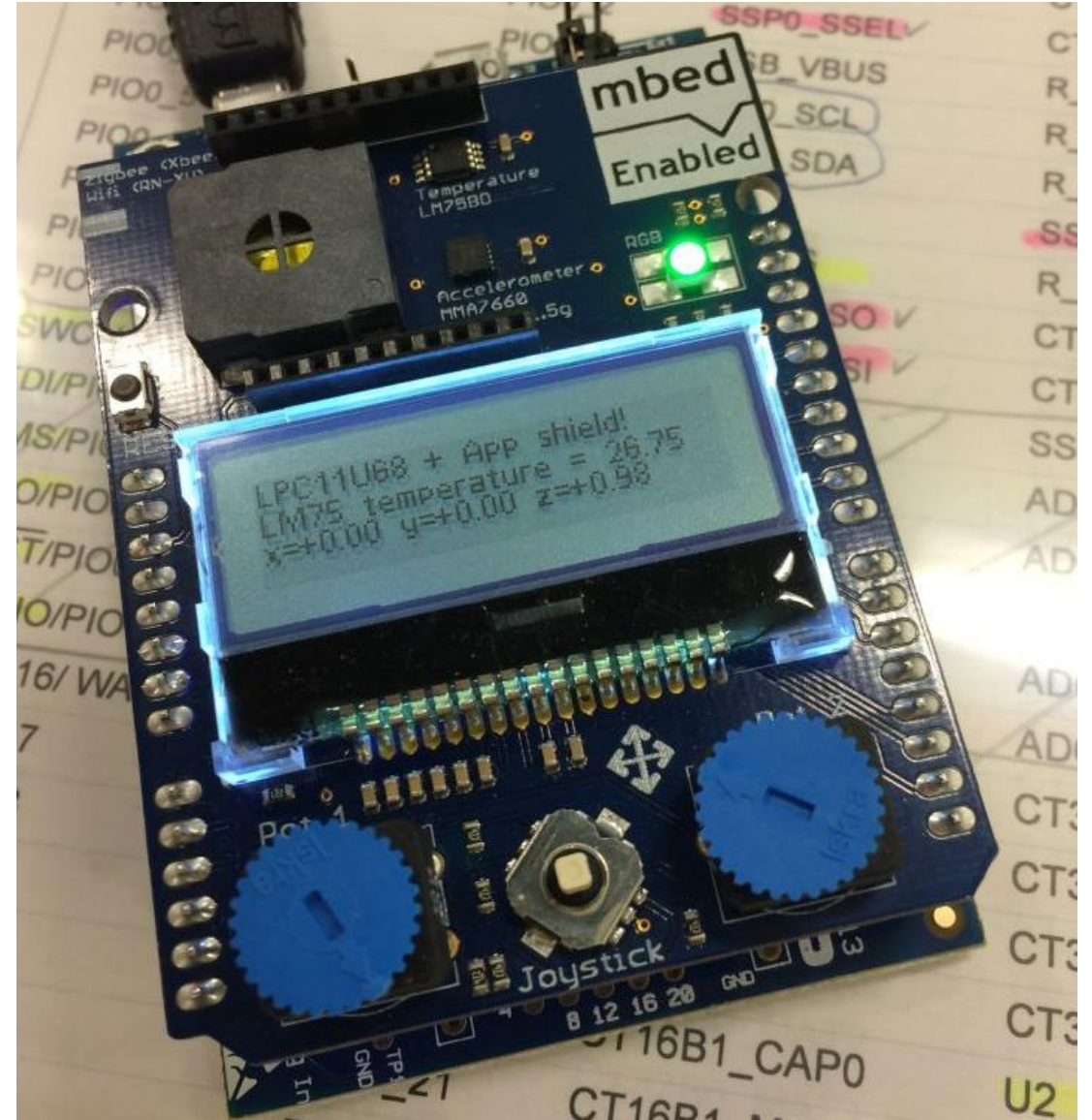


Available now!



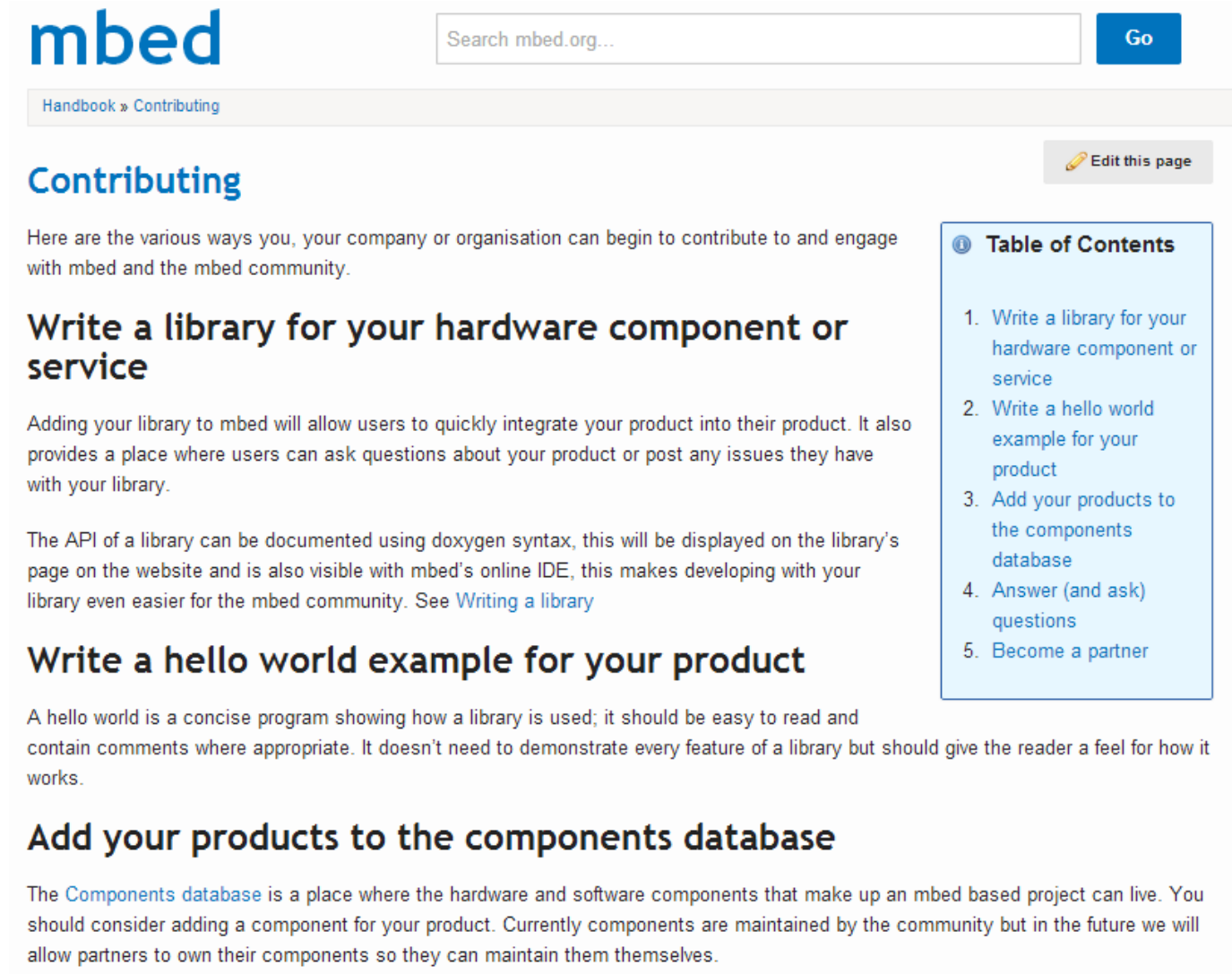
# 最近のアップデート (続き)

- **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



# 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



mbed

Search mbed.org... Go

Handbook » Contributing

## Contributing

Here are the various ways you, your company or organisation can begin to contribute to and engage with mbed and the mbed community.

### Write a library for your hardware component or service

Adding your library to mbed will allow users to quickly integrate your product into their product. It also provides a place where users can ask questions about your product or post any issues they have with your library.

The API of a library can be documented using doxygen syntax, this will be displayed on the library's page on the website and is also visible with mbed's online IDE, this makes developing with your library even easier for the mbed community. See [Writing a library](#)

### Write a hello world example for your product

A hello world is a concise program showing how a library is used; it should be easy to read and contain comments where appropriate. It doesn't need to demonstrate every feature of a library but should give the reader a feel for how it works.

### Add your products to the components database

The [Components database](#) is a place where the hardware and software components that make up an mbed based project can live. You should consider adding a component for your product. Currently components are maintained by the community but in the future we will allow partners to own their components so they can maintain them themselves.

Edit this page

#### Table of Contents

1. Write a library for your hardware component or service
2. Write a hello world example for your product
3. Add your products to the components database
4. Answer (and ask) questions
5. Become a partner

# 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

# 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**

# 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



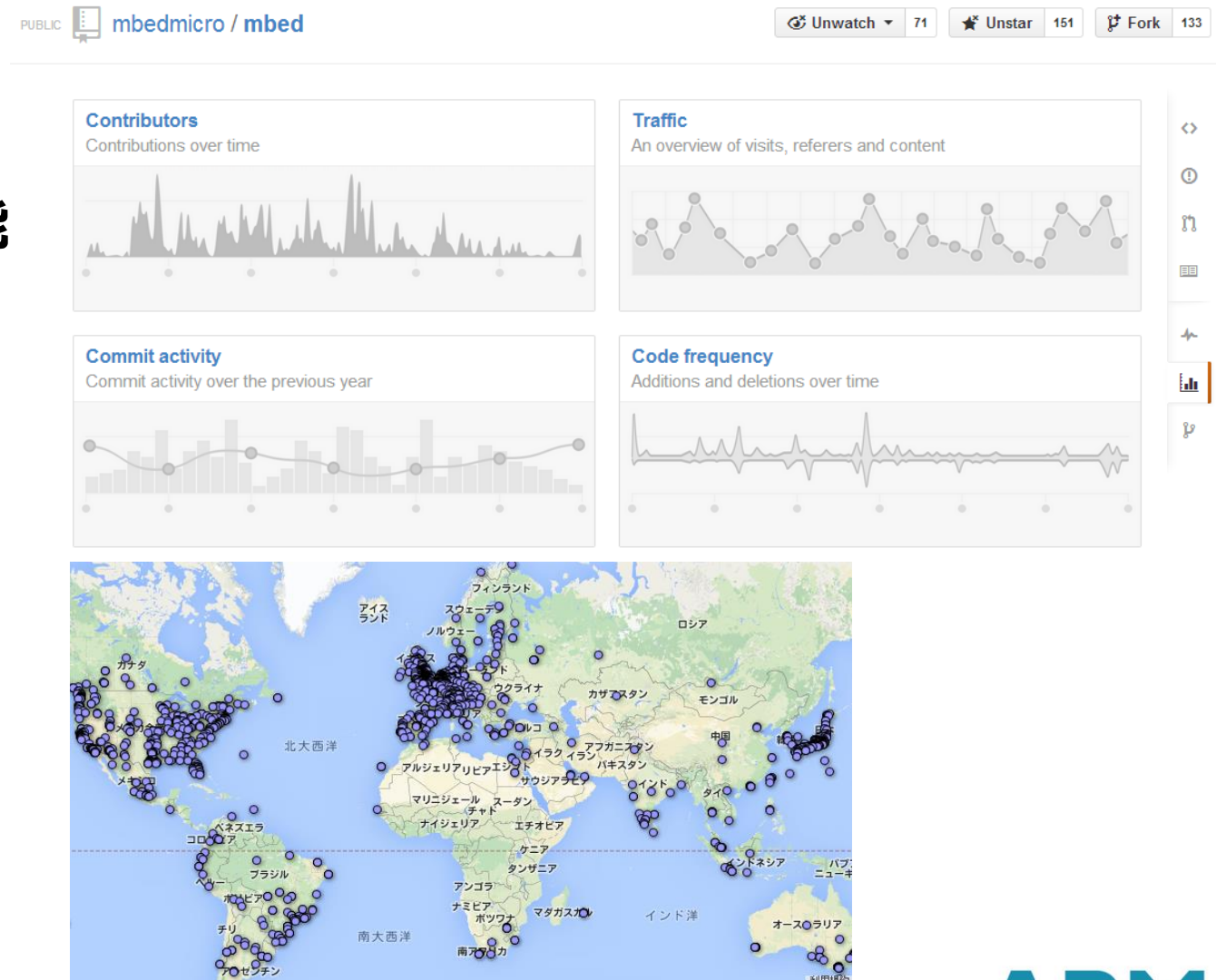
# ユーザーミートアップイベント「mbed祭り」

- ユーザーが企画、運営している mbed ユーザーのためのイベント
  - 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/>



# 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)

# 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

# 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

# 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

# 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`



# 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**

# 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
- 必要な部分を 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)

# 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,  
}
```

# 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

# 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）

# 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で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

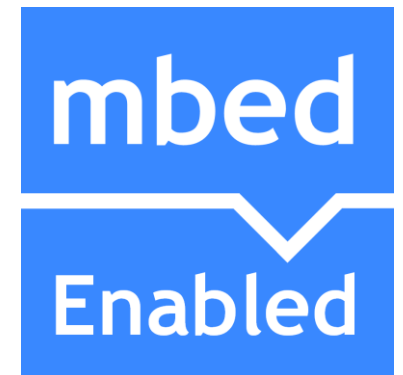
# (私が経験した) SDK portingで発生したバグなど

- **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/](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/bfc970cffa6/mbed-HDK.lbr](http://mbed.org/users/mbed_official/code/mbed-HDK/file/bfc970cffa6/mbed-HDK.lbr) (Eagle 用ベクタデータ)



# 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
- 必要な部分を追記する（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）