マウスのためのマイコン知識

早稲田大学マイクロマウスクラブ
M1

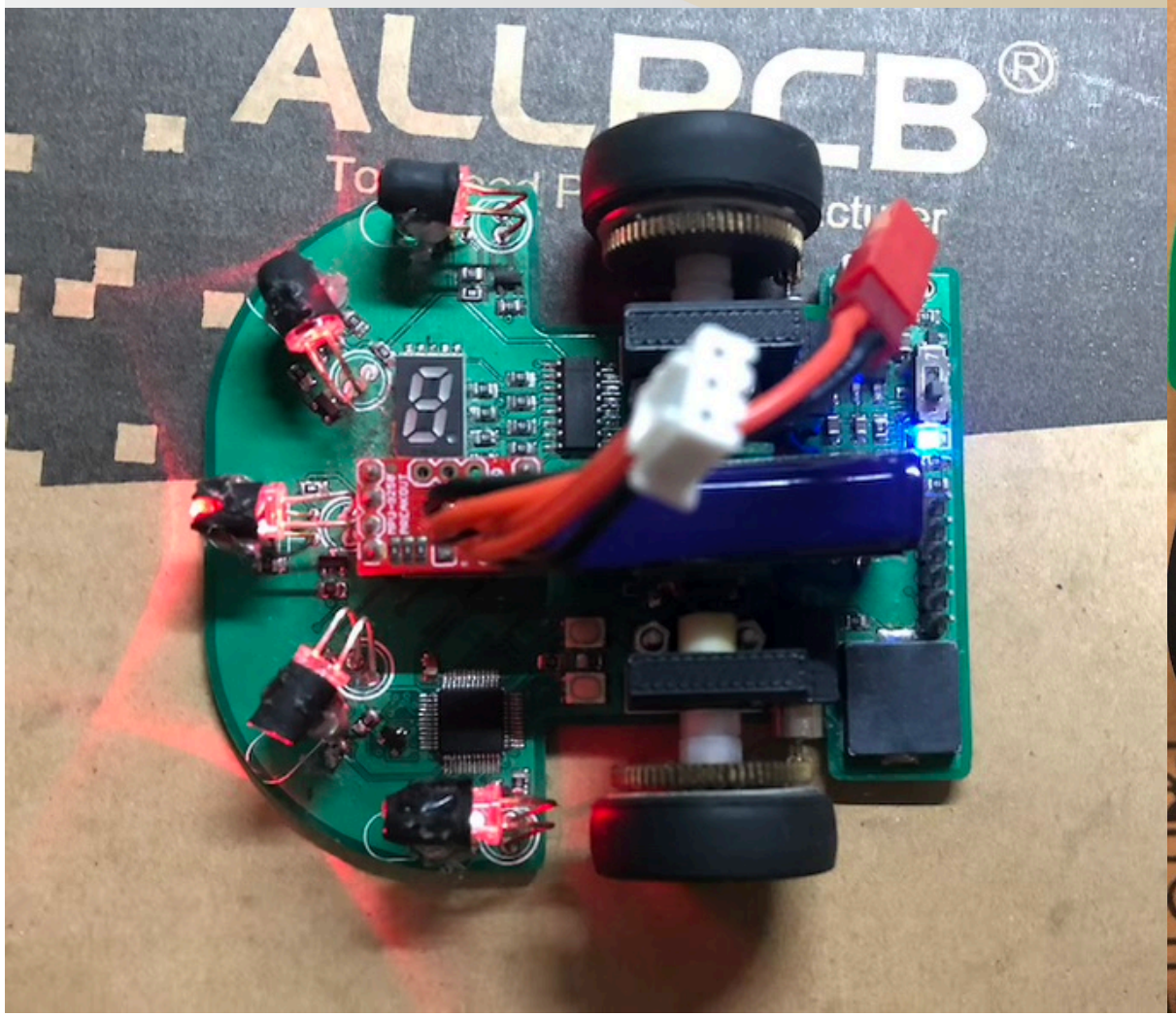
Outline

- **初級編**
  - マイコンとは
  - マウスに必要なペリフェラル
  - メジャーなマイコンの比較
  - マイコン選びのチェック項目
- **上級編**

$ whoami

- **しまじゃき**(@obknt)です
- WMMC老害(0x**18歳**)
  - 大学院で**振動発電回路**の研究をしています
- 趣味: 電子工作、**マイコン**、東武鉄道、島村卯月
- DCマウス製作中(走るとは言っていない)
  - C++わからん

『じゃき太浪 :wq』 くん


ALLPCB®


Jaki-Taro :wq
Version 2.0
Designed on 2018/08
blueeyes.sakura.ne.jp
Vim

$ whoami # 宣伝です

- ブログやってます↓
  - 「**なんとなく活動記録。**」(http://blueeyes.sakura.ne.jp/)
  - マイコン中心。6月は**月間1万PV達成**！

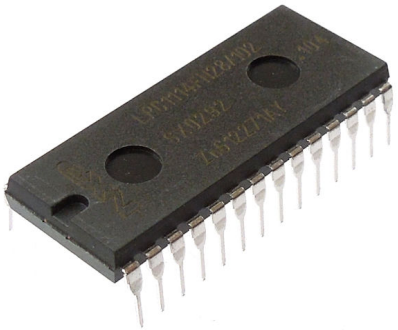
初級編

Outline for 初級編

- マイコンとは
- マウスに必要なペリフェラル
- メジャーなマイコンの比較

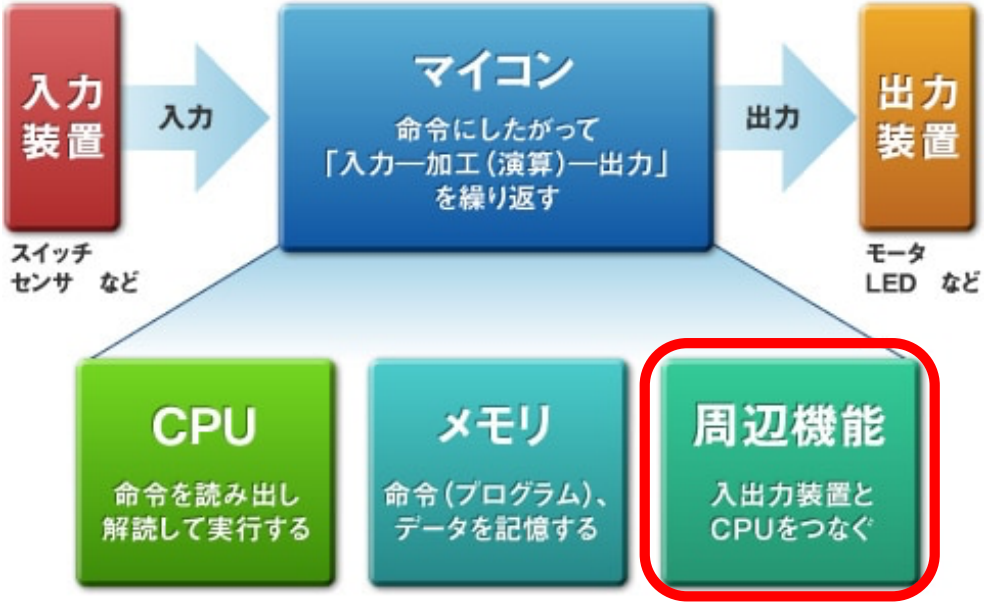
マイコンとは

- モノを制御する小さな計算機(マイクロコンピュータ)
  - **CPU + メモリ + ペリフェラル(周辺機能)**
  - 基本的にOSは存在せず、電源投入時にプログラムがそのまま動く
- CPUが司令塔



LPC1114FN28

STM32F303K8T6



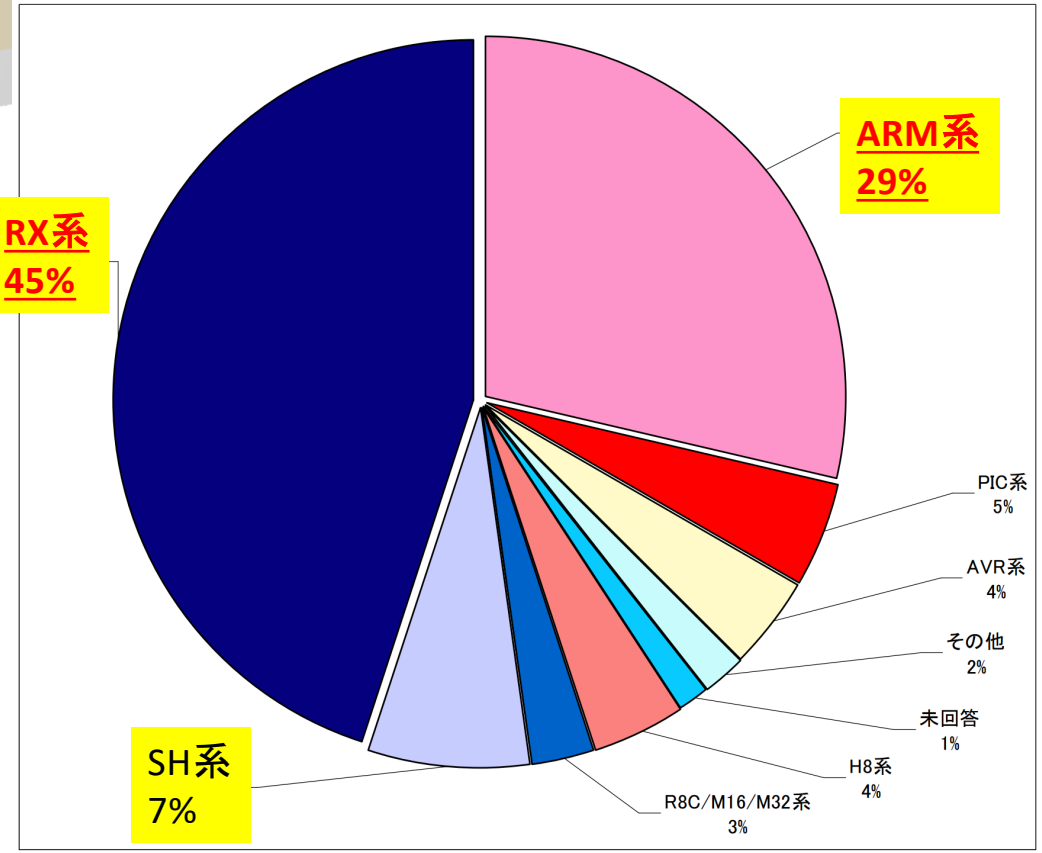
マウスに必要なペリフェラル(一例)

- **タイマー**
  - モーターへのPWM波形出力 → モーター回転速度
  - 周期割り込み → 周期的な制御
  - 2相エンコーダ読み取り(主にDCマウスでの使用)
  - マイコンによってタイマーの機能が異なる
- **ADコンバータ(ADC)**
  - フォトトラからのアナログ電圧を数値として表現 → センサ値取得
  - マイコンによって精度が異なる
- **UART**
  - PCとの通信(デバッグやデータ出力用)

→ **マイコンはペリフェラルで選ぶことが重要**

メジャーなマイコンの比較

- ARMとRXがメジャー
  - SHは過去の遺産を使うための利用？
  - SHは古いので、おすすめしない
- RXとARMで以下の項目を比較
  - 対応するPCのOS
  - 開発環境の違い
  - 調べやすさ(ググリティ)

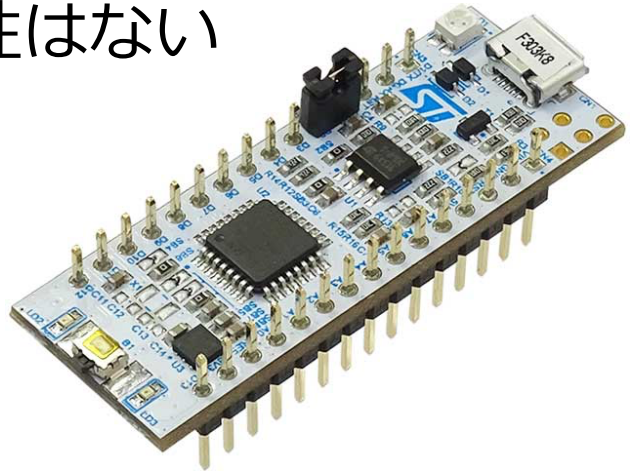


「MM2018第39回全日本マイクロマウス大会記録集」より

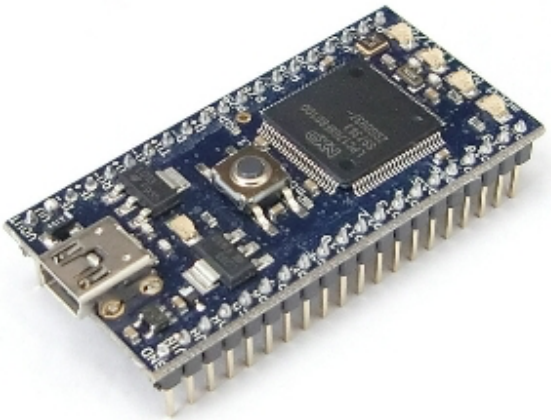
ARMマイコン – ARM Cortex-M シリーズ

- ARMとはCPUコアの名称。様々なマイコンに使用されている
- マイコン採用メーカー：

  Renesas(**RZ**シリーズ), STMicro(**STM32**シリーズ), NXP(**LPC**, **Kinetis**シリーズ), Cypress(**PSoC**シリーズ), and so on…
- **ペリフェラルの構成はメーカー独自**なので、プログラムの互換性はない



STM32F303K8T6 Nucleo Board



mbed LPC1768評価ボード

ARMマイコン – ARM Cortex-M シリーズ

- ARM用のgccを使えば**豊富な環境で開発**できる
  - ARM社がコンパイラー式を公式で配布
  - C++17対応
- マイクロマウス業界では、以下がよく使われているみたい
  - STM32F3, **STM32F4**シリーズ
  - LPC1768

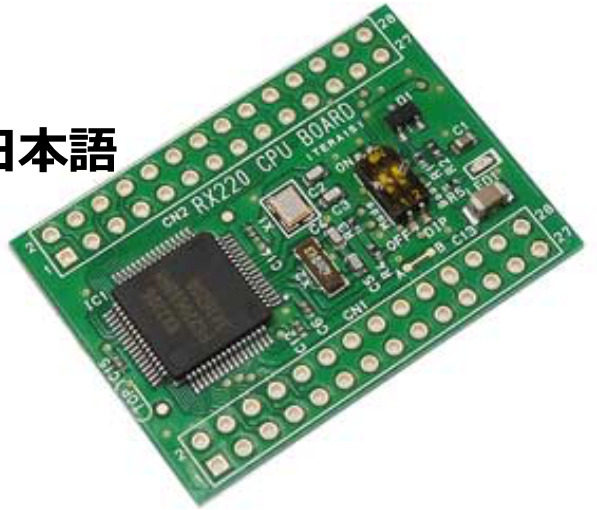
項目	
対応するPCのOS	Windows, macOS, Linux
開発環境	メーカー毎に異なる(Eclipse系が多い)
調べやすさ	メーカー毎に異なる STM32はマウサーによる記事が多い
その他	日本語情報は多くはない STM32はコード自動生成が可能(CubeIDE)

ARMマイコンの代表例

マイコン名	製造メーカー	開発環境	対応OS	その他
LPC	NXP	MCUXpresso, mbed	Windows, macOS, Linux	-
STM32	STMicroelectronics	CubeIDE, mbed	Windows, macOS, Linux	CubeIDEにコード生成機能あり

RXマイコン

- 車載用としても用いられるRenesasの**純国産マイコン**
- コストパフォーマンス良し(私見)
  - 安い割にROM容量、ペリフェラル数が多い
- 開発環境が複数あるので、好きなものを選べる
- **日本語情報豊富**！
  - データシート、ユーザーマニュアル、フォーラムが**フルで日本語**
- 公式コンパイラCC-RXとgccがある
  - CC-RX: C99 or C++98での開発が可能
  - rx-elf-gcc: C11 or C++11での開発が可能



秋月電子通商で売られている
RX220マイコンボード

RXマイコン

- マイクロマウス業界では、以下がよく使われているみたい
  - RX220, RX631
- 若干ハマりどころがある
  - *e.g.)* PC7端子、EMLE端子のプルダウン、レジスタライトプロテクション
  - **RX631は外部クリスタルを付けないと書き込めない**ので注意
  - 先人の知識を基にすれば回避可能

項目	
対応するPCのOS	Windows
開発環境	CS+, e^{2}studio, HEW と豊富
調べやすさ	◎
その他	日本語情報が多い PDG2によるコード生成が可能

ARM or RX

- 絶対に英語は嫌だ
  → RXマイコン
- 国産が安心できる
  → RXマイコン
- Mac使い、Linux使いである
  → ARMマイコン
- C++17を使いたい
  → ARMマイコン
- 周りが○○マイコン使ってるんだけど…
  → じゃあそれで(**先人の知識は偉大**なので)

マイコン選びのチェック項目

- ✓ 開発環境は使用するPCのOSに対応しているか？
- ✓ 電源電圧は3.3V？5V？
- ✓ ペリフェラル数は希望数以上あるか？
  - UART
  - タイマーの数
  - ADCの数
- ✓ 十分なROMを積んでいるか？
  - 最低でも32kByte、64kByte以上あると余裕がある
- ✓ (Optional) FPUを積んでいるか？
  - あると浮動小数点演算が高速になる

閑話休題

- 我々はマウスを通して何を学ぶのか？
  - マウスだ
  - 制御だ
  - 電子回路だ
  - プログラムだ
  - **マイコンの気持ちだ**

→ マイコンの気持ちになろう！！

というわけで

上級編 ~ マイコンの気持ちになるために ~

Outline for 上級編

マイコン狂に贈る…

- whoami --verbose
- 目的・注意点
- 準備
- Lチカプログラム
  - in C language
  - in Assembly
- まとめ

$ whoami --verbose

- **しまじゃき**(@obknt)です
- 趣味: **マイコン**、東武鉄道、島村卯月
  - 将来の夢：マイコンを弄ってお給料が発生して欲しい
- 自称 マイコンの人
  - WMMCに入って、マイコン沼に(一人で勝手に)ズブズブ
  - 新しいマイコンを見つけては脊髄反射で購入 → **レジスタ手打ち**や**アセンブリ**で征服するのが趣味であり生きがい
  - アンチライブラリ
  - C++は好きだし、嫌い
  - FPGAで(Verilogで)オレオレアーキテクチャ作ろうとしたが、よく分からず挫折
  - 11個の組込み用gccをMac内に持つ男

屍と化したマイコンたち… (as of 2018/5)

屍と化したマイコンたち… (as of 2019/7)

- STM32
  - L052
  - F042K6T6
  - F103CBT6
  - F103RCT6
  - F103RBT6
  - F303K8T6
  - F303VCT6
  - F303C8T6
  - F303CCT6
  - F334C4T6
  - F405RGT6
  - F415RGT6
- STM8S
  - 105K4B6
  - 105K4T3C
- LPC
  - 1114F/302
  - 1114F/102
  - 11U35
  - 1768
  - 810
  - 812
- Renesas
  - H8/3069
  - SH7125
  - RX220
  - uPD78F
  - R8C AN120
  - RL78/G10
  - RL78/G1C
- ATmel
  - ATmega328P
  - ATtiny85
  - ATtiny86
  - SAMD21G18A
- PIC
  - 16F84A
  - 16F887
  - 18F14K22
  - 18F2420
  - 18F2550
  - 24FJ64GA004
  - 32MX270F
- Z80シリーズ
  - TMPZ84C015
- その他
  - MSP430G2553
  - MSP430FR4133
  - CC2610
  - ESP-WROOM-02
  - ESP-WROOM-32
  - PSoC 5LP
  - EFM32
  - Propeller(DIP)
  - PICAXE-18M2
  - PICAXE-08M2
  - TWELITE BLUE
  - STC89

所有しているものを掲示
赤色がARMマイコン

おすすめ商品

ZK-80ボード

- Intel 8080 CPUをベースにした TK-80ボードをエミュレートしたキット
- PIC32MXマイコンがボタン部制御、CPU動作、モニタプログラムを担う
  - Z80仕様への改造も可能
- aitendoにて販売 **3,500円**
- マイコンの気持ちになれる

目的・注意点

- 本発表ではSTM32F303マイコンを対象にする
  - ARM Cortex-M アーキテクチャCPUを積んだマイコン
- **マイコンの気持ちになる**することが目標

  → **アセンブリ**によって理解する

Why Assembly?

- C言語で書かれたプログラムは**機械語**に翻訳される
  - コンパイラが担当
  - そのCPU専用の命令を記述したバイナリデータ
- **マイコンはそのプログラムがC言語で書かれたことを知らない**
  - 我々もアセンブリを知らない

- プログラム環境の抽象化

→ マイコン-人間 間の相互理解を喪失

- IoT時代到来に向けて、アセンブリをすることで理解を図る
- 順にアセンブリ化して解脱

レジスタ(Register)

- CPU内部に持つメモリ回路
  - 「GPIOAのODR**レジスタ**」のそれとは異なる(これはペリフェラルのメモリであって、CPUの外部にある)
  - RAMとは違って、内部にあるので高速にアクセス可能
  - アセンブリ命令はレジスタを用いる

- CPUのアーキテクチャによって搭載している個数が異なる
  - ARM Cortex-M は R0 から R15 の16個

準備 -アセンブラの用意

- binutilsを用いることで**容易に**環境構築できる

```
$ ../configure --target=arm-none-eabi --prefix=/usr/local/cross/arm-none-eabi --disable-werror --disable-nls --disable-bootstrap
$ make -j16 && make install
```

- 後はパスを通すだけ

```
$ export PATH=/usr/local/cross/bin:${PATH}
```

- 試しに虚無をするプログラムを書いてみる

```
$ echo nop >test.s # nopは「何もしない」という命令
$ arm-none-eabi-as -mthumb -mcpu=cortex-m4 -o test.o test.s
```

- エラーが出なければアセンブリ成功

Lチカプログラム in C言語

- HALライブラリの一部(レジスタマクロ)を用いた記述

```
#include “stm32f3xx.h”

int main(void) {
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN; // IO port A clock enable

    GPIOA->MODER &= ~GPIO_MODER_MODER0_Msk;
    GPIOA->MODER |= (GPIO_MODE_OUTPUT_PP << GPIO_MODER_MODER0_Pos);

    while(1) {
        GPIOA->ODR ^= (1 << 0);
        for(volatile unsigned i=0; i<1000000; i++);
    }
    return 0;
}
```

Lチカプログラム in C言語

- データシート通りに普通に書くとこうなる

```
int main(void) {
    *(volatile unsigned *)0x40021014 |= (1 << 17); // IO port A clock enable

    *(volatile unsigned *)0x48000000 &= ~0b11;
    *(volatile unsigned *)0x48000000 |= (0b01 << 0);

    while(114514) {
        *(volatile unsigned *)0x48000014 ^= (1 << 0);
        for(volatile unsigned i=0; i<1000000; i++);
    }
    return 0;
}
```

Table 16. STM32F303x6/8 peripheral register boundary addresses

	Boundary address	Size (bytes)	Peripheral
3	0x5000 0000 - 0x5000 03FF	1 K	ADC1 - ADC2
	0x4800 1800 - 0x4FFF FFFF	~132 M	Reserved
2	0x4800 1400 - 0x4800 17FF	1 K	GPIOF
	0x4800 1000 - 0x4800 13FF	1 K	Reserved
	0x4800 0C00 - 0x4800 0FFF	1 K	GPIOD
	0x4800 0800 - 0x4800 0BFF	1 K	GPIOC
2	0x4800 0400 - 0x4800 07FF	1 K	GPIOB
	0x4800 0000 - 0x4800 03FF	1 K	GPIOA
	0x4002 4400 - 0x47FF FFFF	~128 M	Reserved

データシートより

11.4.6 GPIO ポート出力データレジスタ（GPIOx_ODR）（x =

アドレスオフセット：0x14

リセット値：0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

15	14	13	12	11	10	9	8	7	6	5	4
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

リファレンスマニュアルより

ペリフェラル先頭アドレス　オフセット

GPIOA->ODR ➔ 0x4800 0000 + 0x14 == 0x4800 0014

Lチカプログラム in アセンブリ

- C言語で書かれたプログラムを再現するようにアセンブリ

```
.text
.code 16
.syntax unified

.global _start
_start:
        .org 0x00000000
        .word STK_TOP
        .word init
        .type init, %function

init: @ プログラムはここからスタート
        @@ GPIOAを有効化
        LDR r0, =RCC_AHBENR
        ldr r1, [r0]
        orr r1, r1, #RCC_AHBENR_IOAEN
        str r1, [r0]

        @@ GPIOB_7を出力に設定
        LDR r0, =GPIOA_MODER
        ldr r1, [r0]
        orr r1, r1, #0b01
        bic r1, r1, #0b10
        str r1, [r0]
```

プログラムサイズ：
72バイト

```
LEDToggle:
        @@ toggle GPIOB_7
        LDR r0, =GPIOA_ODR
        ldr r1, [r0]
        eor r1, r1, #1 @ r1 <- r1 XOR 1
        str r1, [r0]
        bl wait
        b LEDToggle


wait:
        LDR r0, =1000000
wait_loop:
        subs r0, #1
        bne wait_loop

        bx lr

.end
```

```
.text
.code 16
.syntax unified

.global _start
_start:
        .org 0x00000000
        .word 0x20003000
        .word init
        .type init, %function

init: @ プログラムはここからスタート
        @@ GPIOAを有効化
        LDR r0, =0x40021014
        ldr r1, [r0]
        orr r1, r1, #0x20000
        str r1, [r0]

        @@ GPIOB_7を出力に設定
        LDR r0, =0x48000000
        ldr r1, [r0]
        orr r1, r1, #0b01
        bic r1, r1, #0b10
        str r1, [r0]
```

マクロを全て
展開した

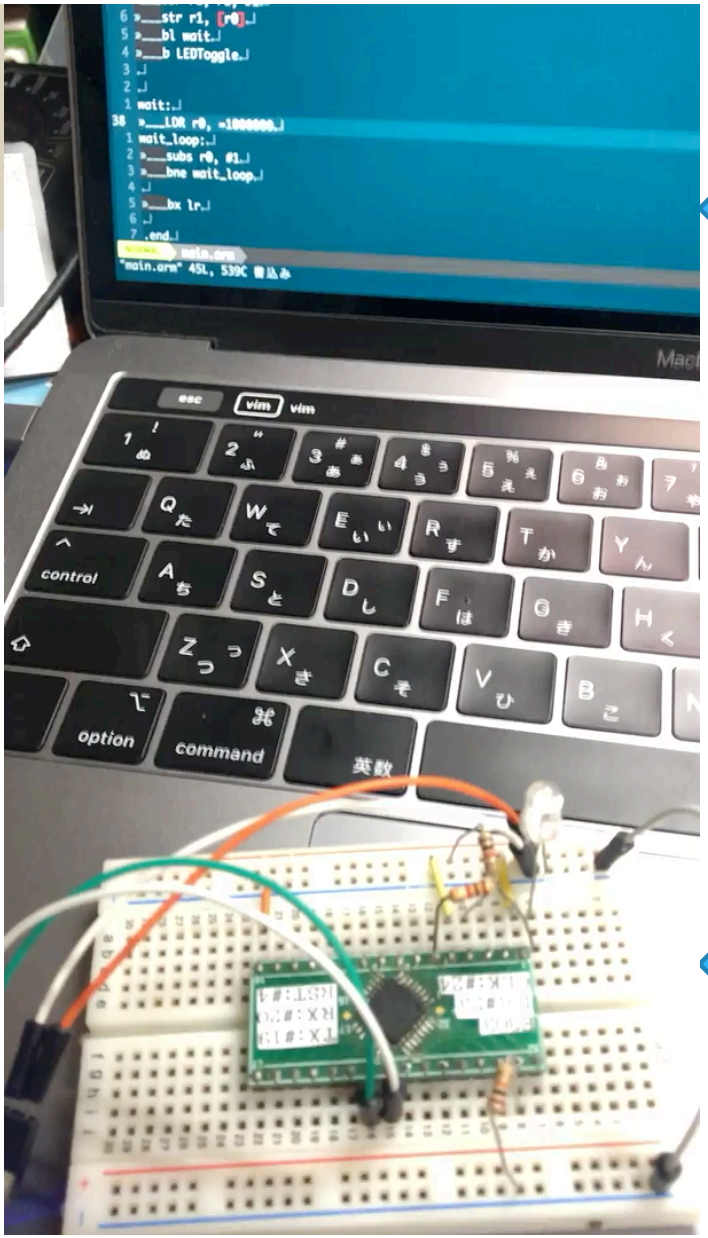
```
LEDToggle:
        @@ toggle GPIOB_7
        LDR r0, =0x48000014
        ldr r1, [r0]
        eor r1, r1, #1 @ r1 <- r1 XOR 1
        str r1, [r0]
        bl wait
        b LEDToggle


wait:
        LDR r0, =1000000
wait_loop:
        subs r0, #1
        bne wait_loop

        bx lr

.end
```

動く・・・動くぞ



STM32マイコン
with 赤色LED

これをやって何が嬉しいのか

- 楽しい
- マイコンの気持ちになれる
  - 一方でC言語は高級アセンブリ言語だと認識されるでしょう
- ポインタが理解出来る
  - [r0] にあるカッコがそう
- 自分の環境に対応していないマイコンを開発できるようになる
  - アセンブリに対する拒絶反応がなくなる

  → スタートアップコードが書ける

  → **圧倒的スキルアップ**

Q&A

- 私もやってみたい！
  - イチからARMに触れるのはおすすめしない(面倒くさい)
  - AVRが一番シンプルでいいかも
- アセンブリは甘えでは？
  - 確かにマイコンはアセンブリ文字列を認識していない(気持ちになりきれない)
  - 分かるのは機械語である

PF0がLチカ出来る秘密のシェル芸

```
$ echo 00 30 00 20 09 00 00 08 17 48 40 F2 00 01 C0 F2 40
01 01 60 15 48 40 F2 01 01 01 60 40 F2 F4 12 00 F0 06 F8
12 48 01 68 81 F0 01 01 01 60 F5 E7 10 48 40 F2 E8 31 01
60 0F 48 00 21 01 60 0F 48 01 21 01 60 00 2A 0A D0 03 68
40 F2 00 01 C0 F2 01 01 0B 40 00 2B F7 D0 A2 F2 01 02 F2
E7 01 68 81 F0 01 01 01 60 70 47 14 10 02 40 00 14 00 48
14 14 00 48 14 E0 00 E0 18 E0 00 E0 10 E0 00 E0 | xxd -r
-p >blink.bin
```

- これで完全にマイコンの気持ちになれるんですよ

参考にしたものたち

- MM2018第39回全日本マイクロマウス大会記録集
- idさんの記事(http://idken.net/posts/2016-12-25-arm_asm1/)
- STM32F303関連：
  - データシート(ピンアサイン・ペリフェラルのメモリマップ)
  - リファレンスマニュアル(レジスタの機能・オフセット値)
  - プログラミングマニュアル(アセンブリのやり方)
- 書籍：
  - ARMマイコン Cortex-M教科書(CQ出版)
  - 大熱血！アセンブラ入門(秀和システム、坂井先生 Kindle版もある)

ご清聴ありがとうございました…