SWEST27 s2c

Arduinoから一歩進んでマイコンの仕組み入門

システムアイ
伊藤慎治 (0004)

自己紹介

Twitter：
@windy_pon

- 伊藤 慎治 (システムアイ)
  - 個人事業主(フリーランスではない)
  - 組込みシステムの受託開発
  - 2004年開業、22年目
    - リーマンショックもコロナ禍も乗り越えた
  - 名古屋の大須に事務所があります
  - 組込ファームウェア開発(C/C++)
  - 基板設計・実装
  - FPGAロジック設計（コレが一番好き、VHDL使い）
    - FPGA内蔵CPUのファームウェアとRTLとの協調設計
  - PCアプリケーション（C++Builder使い、まだあるんですよ）

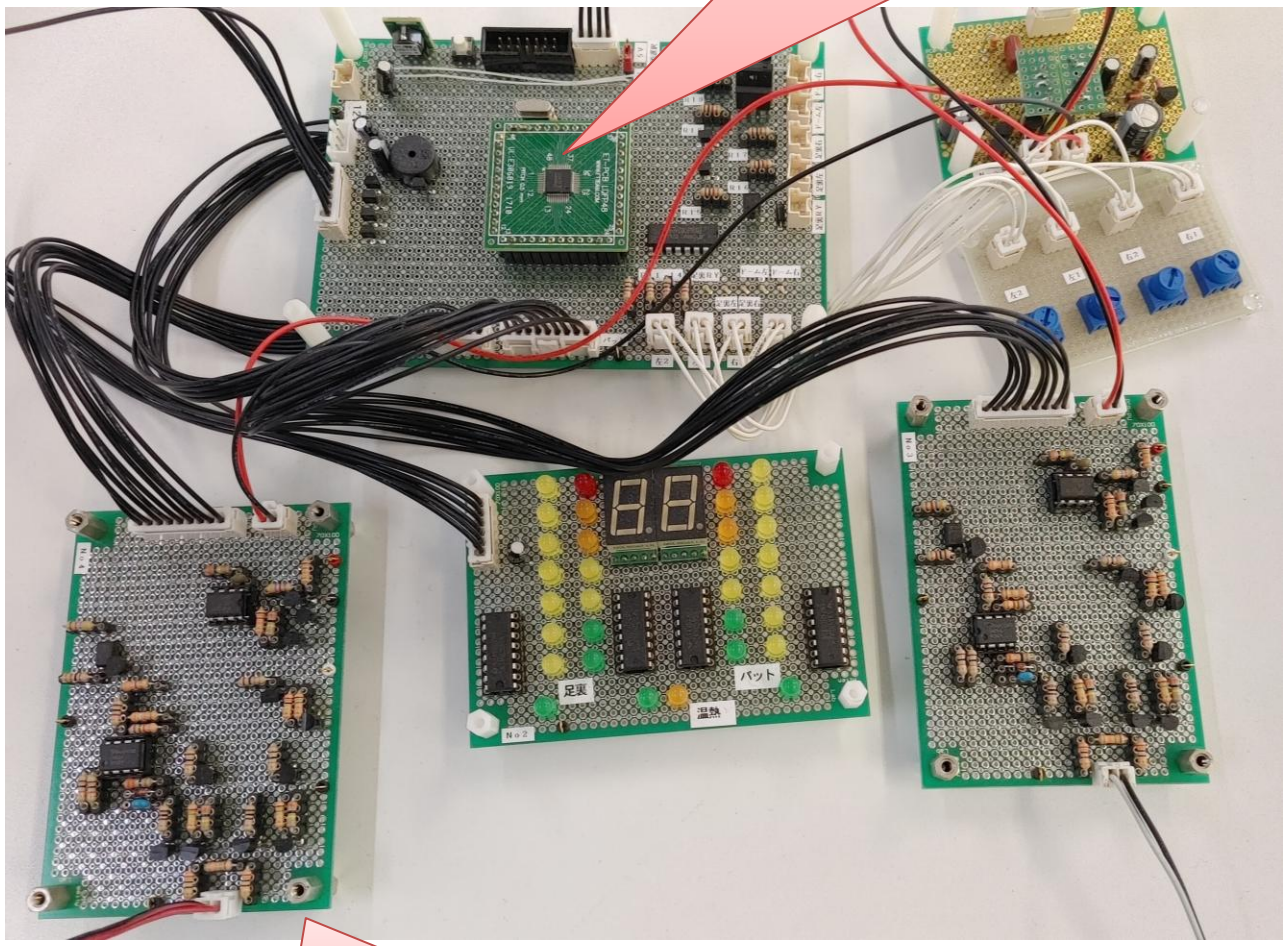
- 出来ない事
  - Web関連・DB関連(SQL知りません)・Linux教えてくれ

低レイヤって、何かね？

- 最近ちょくちょく耳にするこの言葉
- 人によって定義が違うようなので…

「僕の」低レイヤ

これがマイコン
ルネサスRL78 ROM32KB RAM4KB
末端価格￥400/ヶ

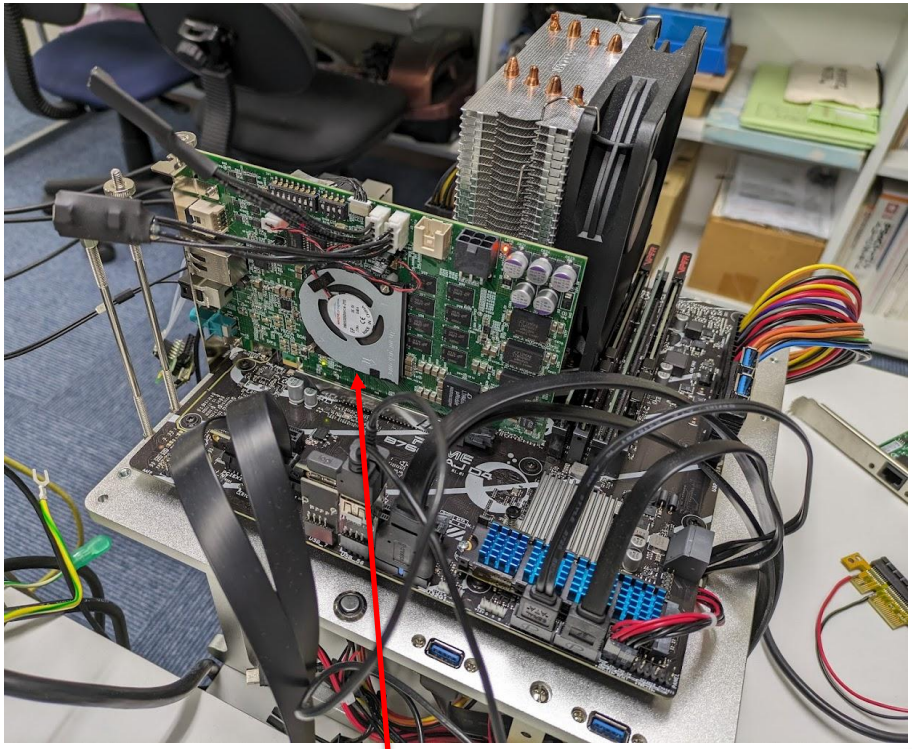


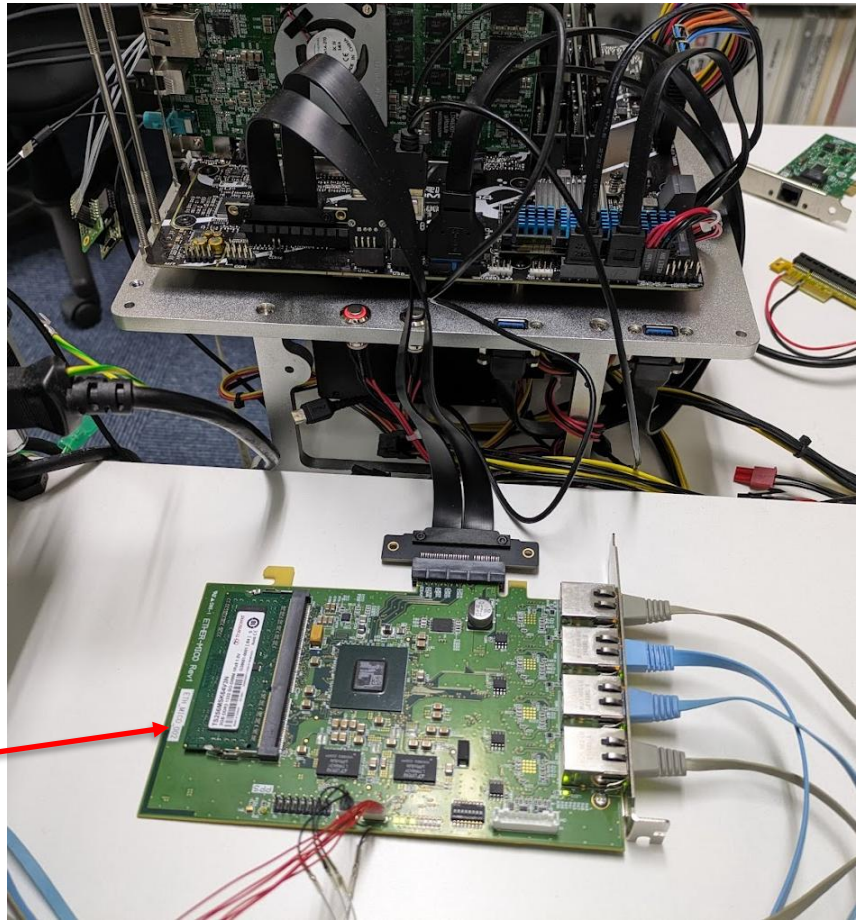
全部ユニバーサル
基板に手組み

この製品の
中身

システムアイ紹介-開発事例

- PCI Express基板のFPGA
  - MIPIカメラ・Ethernetからデータを収集し、PCに保存する
  - FPGAのRTL(VHDL)と内蔵CPUのFW(C++)を同時に開発





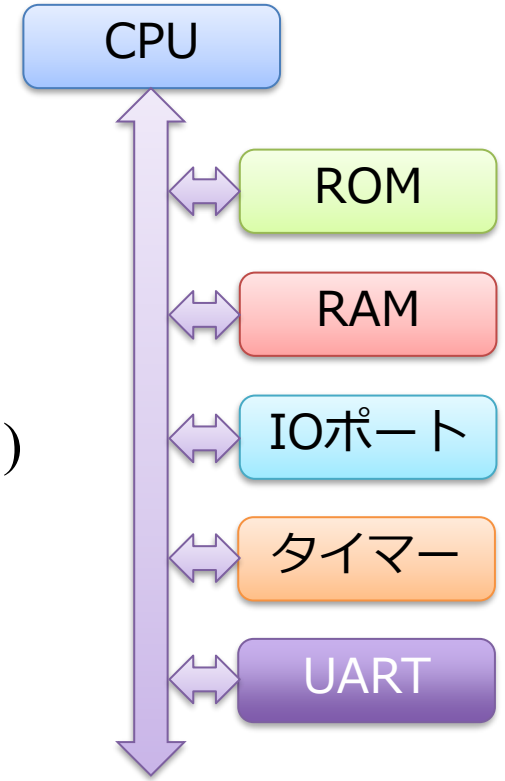
FPGA基板

本セッションの目的

- マイコンの根本的な仕組みを理解する
  - HowTo(使い方)を覚えるのではなく、Why(どうしてそうなっているのか)を理解する
- 根本的な仕組みを理解すれば、マイコンの種類が変わっても対応出来る
- HowToの例
  - Arduinoではsetup()に初期化処理を記述し、loop()に動作中の処理を記述する
  - digitalWrite(pin, val) でIOピン(pin)にH/L(val)が出力される
  - analogRead(pin)でIOピン(pin)の電圧が取得される
- なんでこうなってるの(Why)？

マイコンの中身

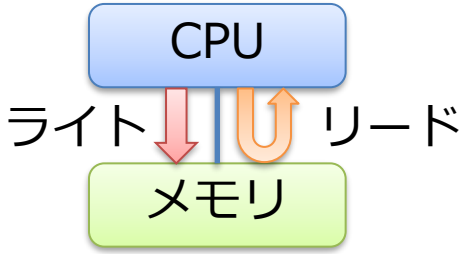
- マイコン(micro controller)とは？
  - 1個のICにプログラム制御に必要な全ての機能が詰まっている
    - CPU
    - メモリ(ROM/RAM)
    - IOポート
    - タイマー
    - 通信機能(UART・I2C・SPI・USB…)
    - Analog to Digital Converter（ADC）
    - Digital to Analog Converter (DAC)
  - 動作周波数は～200MHz



- 対義語はmicro processor
  - MMUが入っていて仮想メモリが使えたり(Linuxが動く)
  - ROMもRAMもなかったり(外部のDRAMを使う)

CPUに出来ること

- CPU (central processing unit) 中央処理装置
- 命令を1個ずつ順に読んで処理していく
- 出来ること
  - 四則演算（数値の演算）
  - 論理演算（ビット演算）
    - NOT・AND・OR・XOR…
  - 比較
    - 一致・不一致・大なり・小なり
  - 分岐
    - 次に読込む命令の位置を変える
  - メモリへのライト/リード



CPUが覚えられるデータ

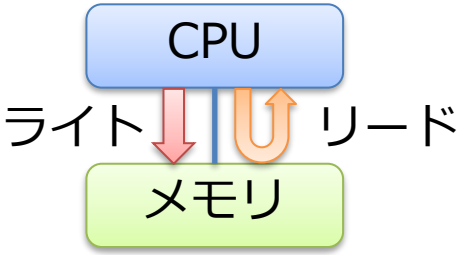
- CPUはレジスタにデータを保持する
- 一般的には32bitのレジスタが16個程度
  - 高機能だと32個だったり
- CPUの演算/比較処理はレジスタ同士で行う
  - R0 ← R1 + R2
  - R3 ← R4 AND R5
  - R14 < R15
  - R6 ← メモリ (リード)
  - メモリ ← R7 (ライト)

R0
R1
R2
R3
R4
…
R14
R15

- これ以上のデータは覚えられない
- CPUにはメモリが必要
  - 少なくともプログラムが格納されているROM
  - 結果を保存するRAM

CPUには入出力は無い

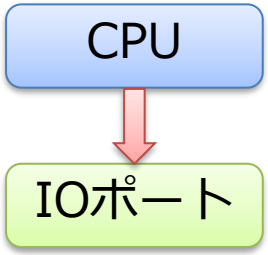
- 手足も無い頭脳だけのような物
  - LEDを点けたり、SW入力を読んだりは出来ない
- どうやって頭脳と手足と繋ぐか？
- CPUはメモリと繋がっている
  - CPUが外部とやりとり出来る唯一の手段



- CPUから見たら全ての入出力はメモリになる

入出力は全てメモリ

- IOポートは特殊なメモリ
  - 書いた値(0/1)がピンに電圧(H/L)で出力される
  - ピンの電圧(H/L)が(0/1)で読める



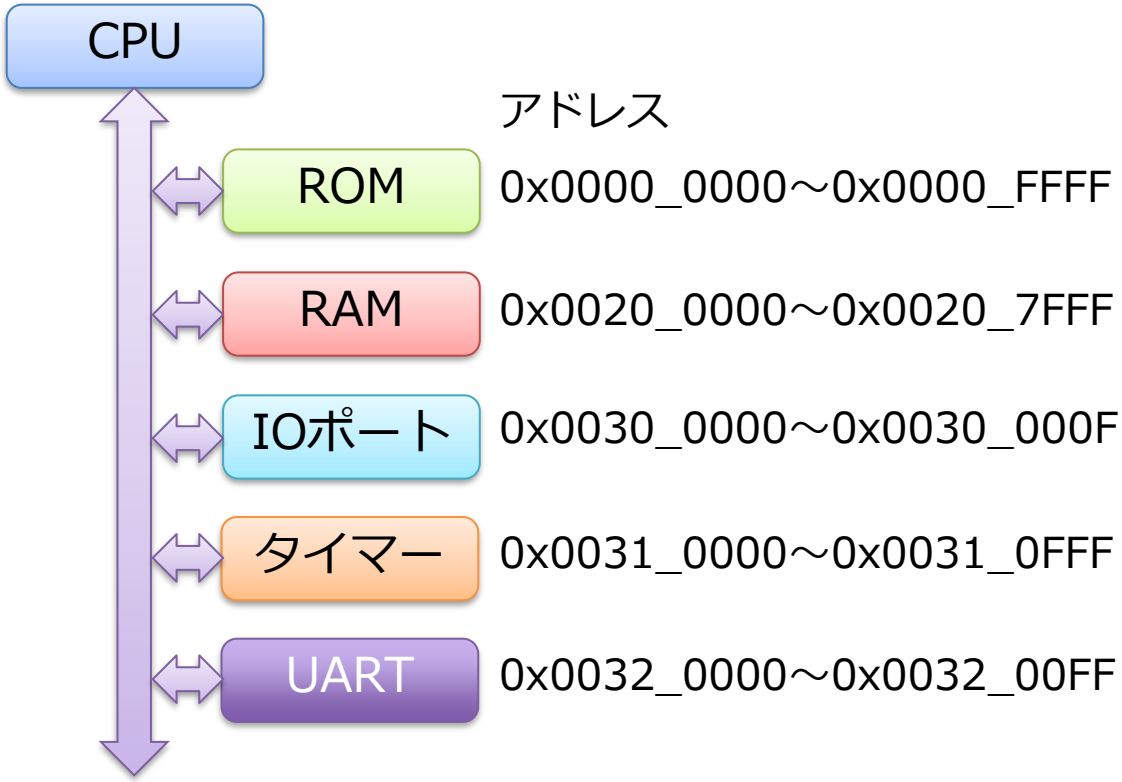
0xA5(1010_0101)
をライト

b7	b6	b5	b4	b3	b2	b1	b0
H	L	H	L	L	H	L	H

b0～b7に対応するピンにH/Lが出力される

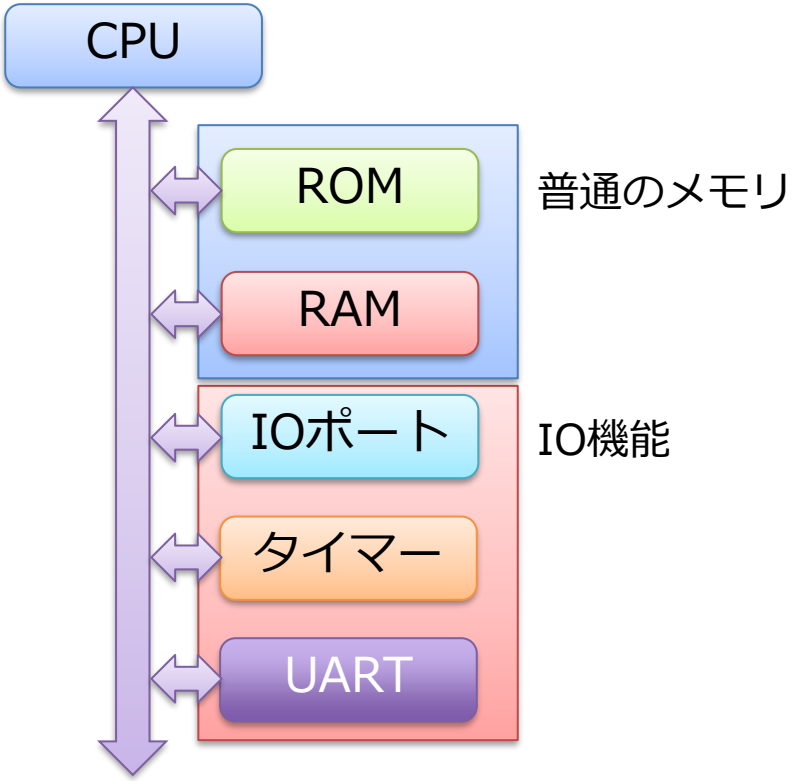
CPUとメモリの接続方法

- バスで繋がっている
- バスとは通信線のようなもの
- アクセス対象は一意のアドレスで識別する



メモリマップドIO

- CPUから見て普通のメモリ(ROM/RAM)と同じ方法でIO機能が割付けられている方式のことを言う
- 現在のマイコンはほぼこの方式と考えて良い



メモリの種類について

- マイコンで使われるメモリは大きく分けて2種類
- ROMとRAM
- ROM（read only memory）
  - 読めるけど変更出来ない(書き込めない)
  - プログラムはROMに格納される
  - マイコンでは特殊な手順で変更出来るFLASH-ROMが使われる
- RAM（random access memory）
  - 読めるし変更も出来る
  - 電源Offするとデータが消える
    - 電源On時にどんなデータになっているか分からない
  - 昔はRWM（read write memory）と呼ばれていた
    - こちらの方が正しく体を表すが母音がないと発音しにくい

C言語のプログラム

- 一般的なC言語の問題

```
int a;
char b =  ‘b’ ;
char c =  ‘c’ ;
const int d = 6;

void main(void)
{
    int e;
    int f = 3;
    //この時点での各変数の値は？
}
```

C言語のプログラム

- 答
  - a：0　　初期値無しグローバル変数
  - b：’b’　初期値有りグローバル変数
  - c：’c’　初期値有りグローバル変数
  - d：6　　グローバル定数
  - e：不定　初期値無しローカル変数
  - f：3　　初期値有りローカル変数

```
int a;
char b =  ‘b’ ;
char c =  ‘c’ ;
const int d = 6;

void main(void)
{
     int e;
     int f = 3;
     //この時点での各変数の値は？
}
```

C言語の変数の種類

- グローバル変数
  - 初期値なし
  - 初期値あり
  - 定数
- ローカル変数
  - 初期値なし
  - 初期値あり

```
int a;
char b =  'b' ;
char c =  'c' ;
const int d = 6;

void main(void)
{
     int e;
     int f = 3;
     //この時点での各変数の値は？
}
```

- これらの変数はどうやって初期化されるのか？
  - 組込みはOSが何かをやってくれることはない
  - 本来は自分で書かないといけない

初期値無しグローバル変数

- main()実行時点で全て値は0
- RAMに領域が確保される
- ROMは使用しない
- main()実行前に0クリアする

```
int a;
char b =  'b' ;
char c =  'c' ;
const int d = 6;

void main(void)
{
     int e;
     int f = 3;
     //この時点での各変数の値は？
}
```

初期値有りグローバル変数

- main()実行時点でそれぞれ指定された値
- RAMに領域が確保される
- ROMに初期値が確保される
- main()実行前にROMからRAMに初期値をコピーする

```
int a;
char b =  'b' ;
char c =  'c' ;
const int d = 6;

void main(void)
{
     int e;
     int f = 3;
     //この時点での各変数の値は？
}
```

グローバル定数

- const指定する事で変更されない
- 変数ではないのでRAMではなくROMに配置される
- 初期化処理は不要

```
int a;
char b =  'b' ;
char c =  'c' ;
const int d = 6;

void main(void)
{
     int e;
     int f = 3;
     //この時点での各変数の値は？
}
```

グローバル変数まとめ

種類	ROM	RAM	初期化
初期値無し	ー	○	0クリア
初期値有り	○	○	ROM→RAMコピー
定数	○	ー	不要

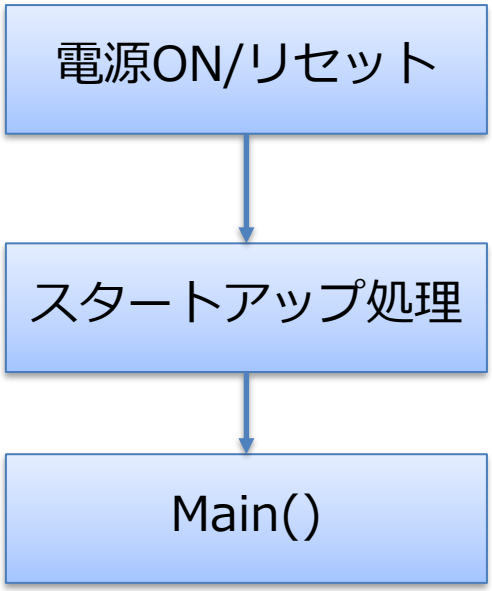
- main()実行前にこれらの処理が実行される

```
int a;
char b =  'b' ;
char c =  'c' ;
const int d = 6;

void main(void)
{
     int e;
     int f = 3;
     //この時点での各変数の値は？
}
```

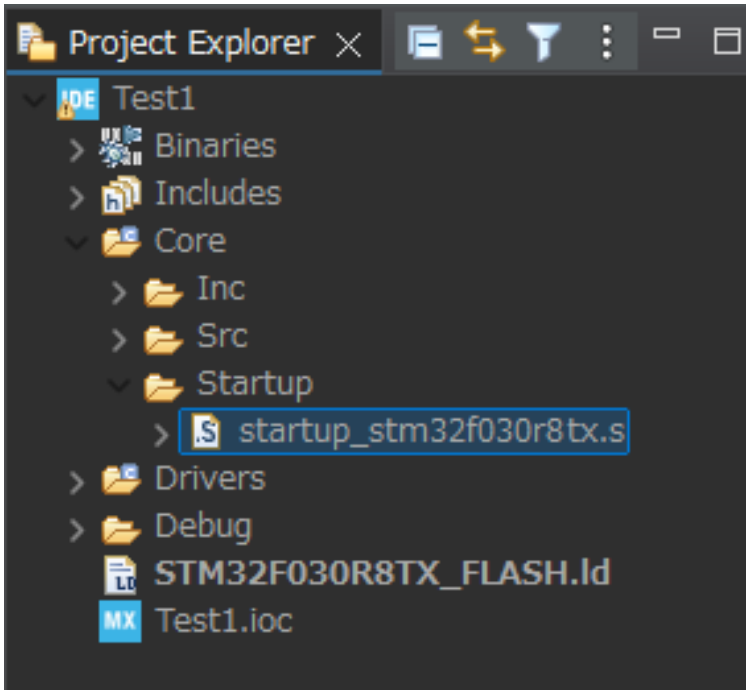
main()が実行されるまで

- いきなりmain()は実行されない
- 電源ON/リセット後には「スタートアップ処理」が実行される
- スタートアップ処理内容
  - 初期値無し変数を0クリア
  - 初期値有り変数にROMの初期値をコピー



STM32のスタートアップ処理

- 「startup_stm32f030r8tx.s」という名前でプロジェクトに生成される
- Cではなくアセンブリ言語
- 生成されるので自分で書く必要はないが、プロジェクトに存在して自分のソースと同じようにコンパイル(アセンブル)されることは知っておく



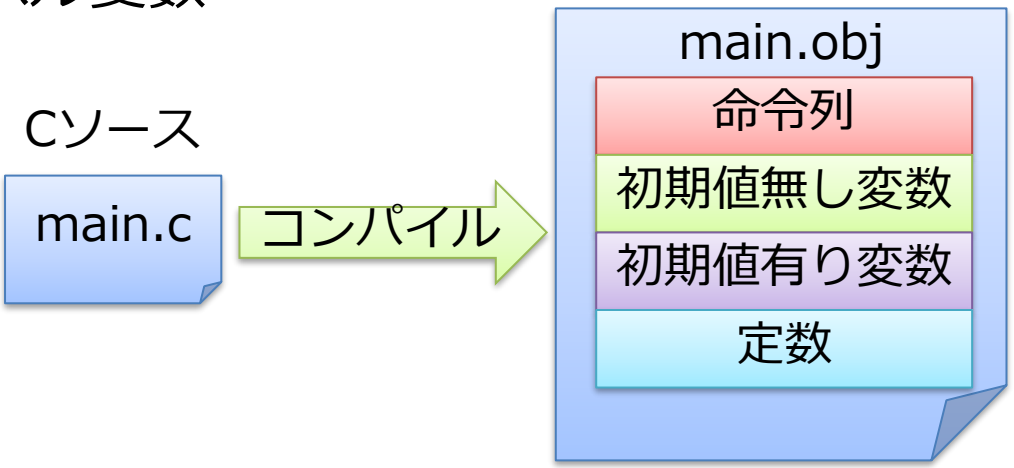
STM32のスタートアップ処理

```
57 /* Copy the data segment
58  initializers from flash to SRAM */
59   ldr r0, =_sdata
60   ldr r1, =_edata
61   ldr r2, =_sidata
62   movs r3, #0
63   b LoopCopyDataInit
64
65 CopyDataInit:
66   ldr r4, [r2, r3]
67   str r4, [r0, r3]
68   adds r3, r3, #4
69
70 LoopCopyDataInit:
71   adds r4, r0, r3
72   cmp r4, r1
73   bcc CopyDataInit
74
75 /* Zero fill the bss segment. */
76   ldr r2, =_sbss
77   ldr r4, =_ebss
78   movs r3, #0
79   b LoopFillZerobss
80
81 FillZerobss:
82   str  r3, [r2]
83   adds r2, r2, #4
84
85 LoopFillZerobss:
86   cmp r2, r4
87   bcc FillZerobss
88
89 /* Call static constructors */
90   bl   libc_init_array
91 /* Call the application's entry point.*/
92   bl main
```

- 初期値有り変数をflash→SRAMにコピー
- _sdata, _edata, _sidata等のシンボルはリンカスクリプトで記述される
- 初期値無し変数を0で埋める
- _sbss, _ebss等のシンボルはリンカスクリプトで記述される
- C++のグローバルクラスオブジェクトの初期化
- main()を呼出す

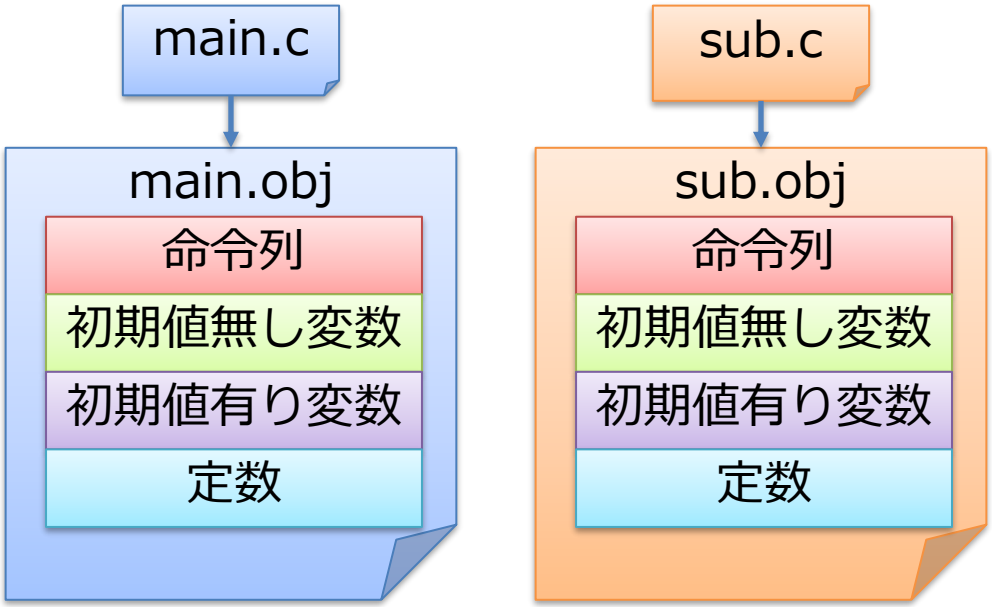
コンパイラとリンカの関係

- コンパイラの仕事
  - テキストで書かれたCソース(.cファイル)をマイコンが実行出来る命令に翻訳する
  - コンパイラが生成するのはオブジェクトファイル(.obj)と呼ばれる
- オブジェクトファイルには以下の情報が含まれる
  - プログラム(命令列)
  - 初期値無しグローバル変数(のサイズ)
  - 初期値有りグローバル変数
  - グローバル定数



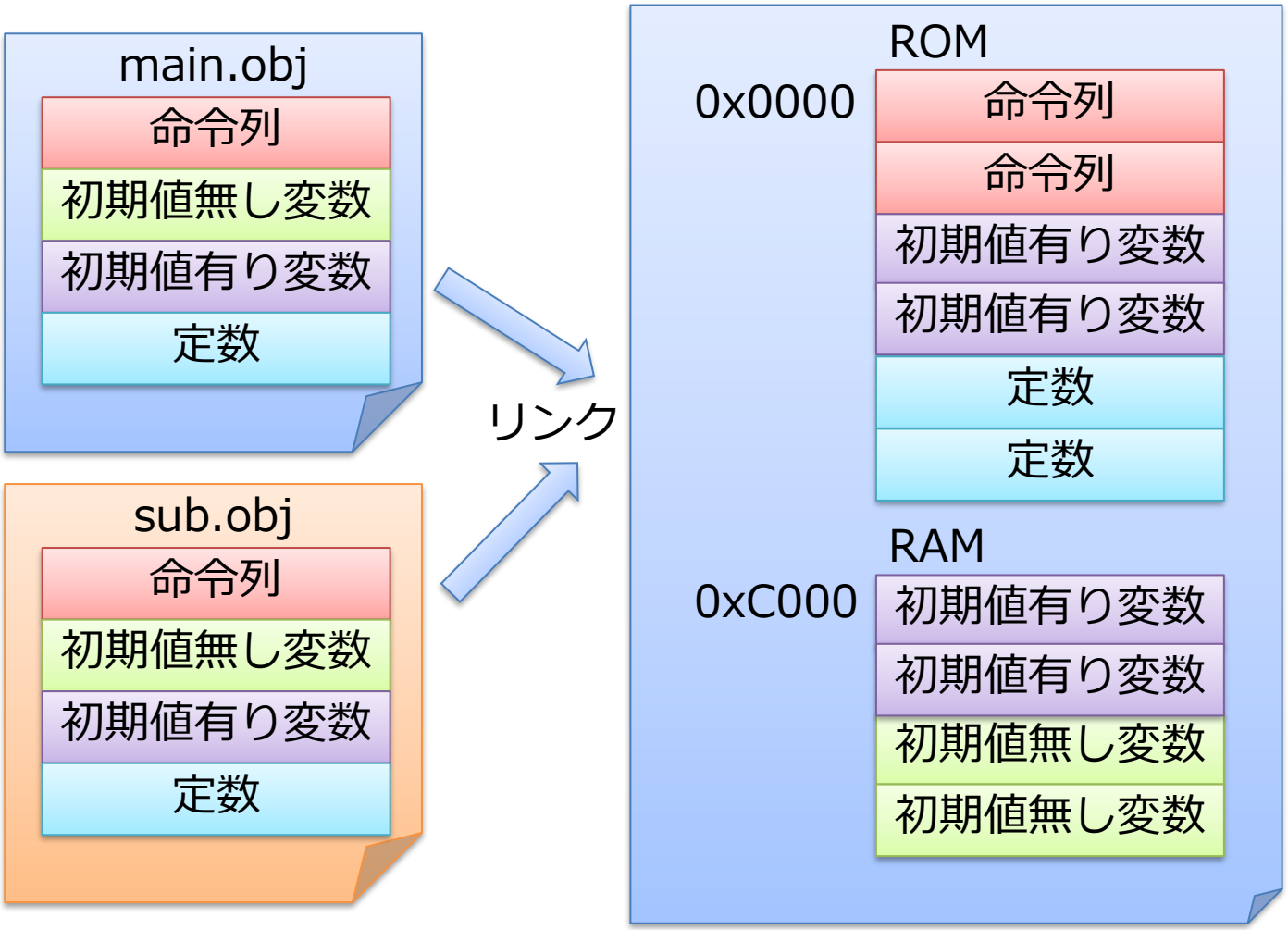
オブジェクトファイル

- オブジェクトファイルは実行出来ない
  - 実行するためにはROM・RAMの具体的なアドレスを割当てる必要がある
  - Cコンパイル時はマイコンのROM・RAMのアドレスを指定出来ない
- コンパイラは単一のCソースだけをコンパイルする
  - 複数ソースがある場合はバラバラのオブジェクトファイルが生成される
- main.objとsub.objは関連性の無いオブジェクトファイル



リンカの仕事

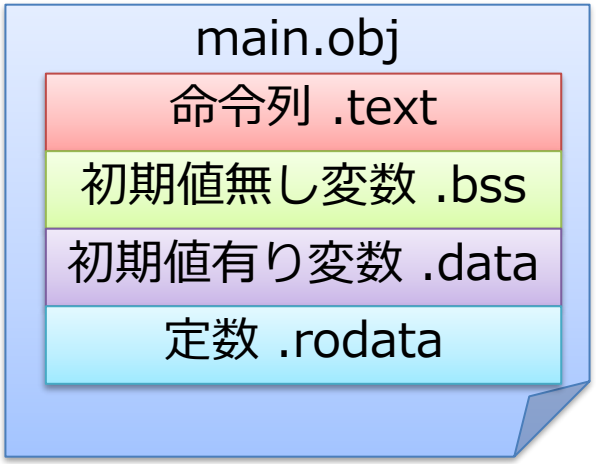
- 複数のオブジェクトファイルを結合する(＝リンク)
- ROM/RAMの具体的なアドレスを割当てる



セクションとは

- オブジェクトファイルに含まれるデータ（プログラム・変数・定数）は「セクション」と呼ばれるデータの塊として分類される
  - STM32 GCCでのセクション名は以下の通り

種類	セクション名
命令列	.text
初期値無し	.bss
初期値有り	.data
定数	.rodata

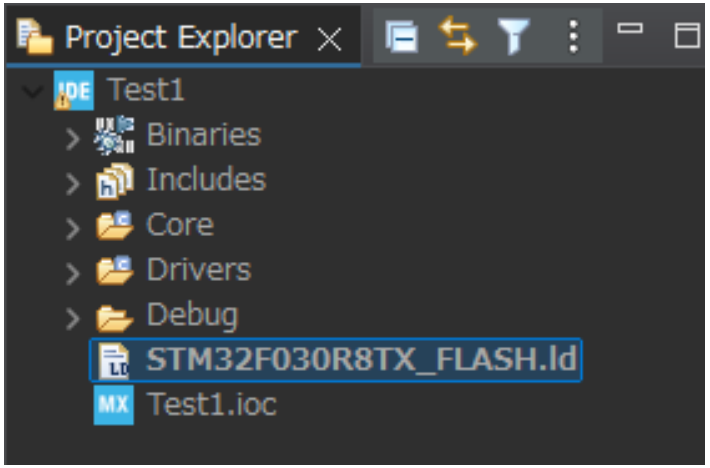


リンカスクリプトとは

- リンカに入力するアドレス情報
  - テキストファイル
  - マイコン固有のROM/RAMのアドレスを記述する
  - 同じ系列のマイコンでもROM/RAMサイズが異なるので、細かい品種毎に内容が異なる
    - サイズを指定することでメモリに納まらない場合にエラーに出来る
- リンカはオブジェクトファイルとリンカスクリプトを入力して実行ファイルを生成する

STM32のリンカスクリプト

- 「STM32F030R8TX_FLASH.ld」という名前でプロジェクトに生成される
- リンカプログラム（この場合はGNU ld）に固有のスクリプト仕様
- C言語のように標準化はされていない
  - ルネサスのRXマイコンではスクリプトではなく、リンカへのコマンドラインでアドレスを指定する



リンカスクリプトの内容

- リンカスクリプトには2種類の情報を記述する
  - メモリのアドレス
    - 使用するマイコンの型番で決まる
  - 各セクションの配置先メモリ
    - 自由に決められるがほぼ定型

STM32のリンカスクリプト (説明のため重要部分のみ抜粋)

STM32F030R8TX_FLASH.ld

```
1  MEMORY
2  {
3    RAM    (xrw)    : ORIGIN = 0x20000000,   LENGTH = 8K
4    FLASH    (rx)    : ORIGIN = 0x8000000,   LENGTH = 64K
5  }
6
7  SECTIONS
8  {
9    .text :
10   {
11     . = ALIGN(4);
12     *(.text)      /* .text sections (code) */
13   } >FLASH
14
15   /* Constant data into "FLASH" Rom type memory */
16   .rodata :
17   {
18     *(.rodata)
19     *(.rodata*)
20   } >FLASH
21
22   /* Initialized data sections into "RAM" Ram type memory */
23   .data :
24   {
25     *(.data)           /* .data sections */
26     *(.data*)          /* .data* sections */
27     *(.RamFunc)        /* .RamFunc sections */
28     *(.RamFunc*)       /* .RamFunc* sections */
29   } >RAM AT> FLASH
30
31   /* Uninitialized data section into "RAM" Ram type memory */
32   .bss :
33   {
34     *(.bss)
35     *(.bss*)
36   } >RAM
37 }
```

- メモリのアドレス指定
  - RAMとFLASH
- .text(命令列)の配置先
  - →FLASH
- .rodata(定数)の配置先
  - →FLASH
- .data(初期値有り変数)
  - →RAMとFLASHの2箇所

  AT> FLASH でデータがFLASHに確保される 重要！
- .bss(初期値無し変数)
  - →RAM

リンカスクリプトを変更する必要性

- 特殊なメモリ配置をしたい時
- 具体例：命令列をRAMに配置したい
  - 一般的にFLASHはRAMより読出し時間がかかるので、速く実行したい関数をRAMに配置する
  - 命令列をRAMに配置する場合は、スタートアップ処理でROM→RAMにコピーする必要がある
    - この処理は初期値あり変数と全く同じ
  - そういう需要は良くあるので、予め書かれている

```
22    /* Initialized data sections into "RAM" Ram type memory */
23    .data :
24    {
25      *(.data)           /* .data sections */
26      *(.data*)          /* .data* sections */
27      *(.RamFunc)        /* .RamFunc sections */
28      *(.RamFunc*)       /* .RamFunc* sections */
29    } >RAM AT> FLASH
```

RAMに関数を配置する

- リンカスクリプトには.RamFuncセクションをRAMに配置するように書かれている
- 実際にRAMに配置したい関数にはソース上で指定が必要
  - 例：main()を.textではなく.RamFuncセクションにする

```
66  __attribute__((section (".RamFunc"))) int main(void)
67  {
```

まとめ

- CPUに出来る事
  - 演算・比較・分岐・メモリアクセスのみ
  - CPUには入出力はない
  - CPUから見て入出力はメモリのみ
  - メモリマップドIO
- C言語の変数の種類
  - 初期値無し変数
  - 初期値有り変数
  - 定数
- main()の前のスタートアップ処理
- コンパイラとリンカの関係
  - リンカスクリプト