マイコンプログラミング講座

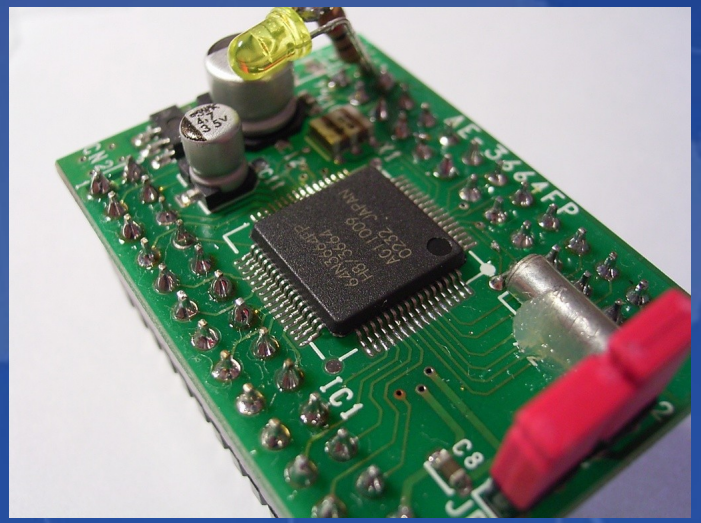
明治大学エレクトロニクス研究部
武山 文信

はじめに

- 回路ができてもプログラムが書けなければ意味がない
- ソフトゼミで触れた部分は、ソフトゼミの資料を見直すなどして
- ちょっと難しめに書いた気がするけど、そこは口頭で補足
- ハードウェアマニュアル嫁
- ググれ

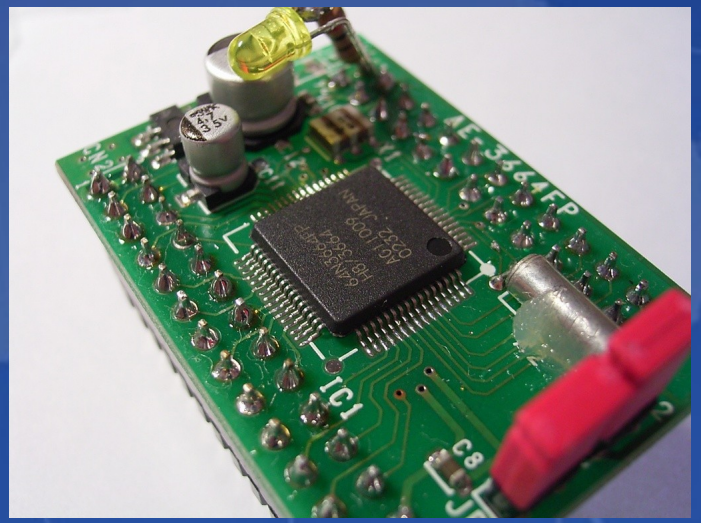
使用するマイコン

- マイコンキット
  - モデル: 秋月電子通商AKI-H8/3664F(QFP)タイニーマイコンキット
  - 価格:¥1600
  - サイズ: 40mm×25mm
  - クロック: 16MHz(メイン) 32.768KHz(サブ)
  - 3端子レギュレーター、シリアル通信用のIC等が実装済み



マイコン

- マイコンキットの中央の黒いチップ
  - Renesas H8/3664F
  - 16bit CPU
  - タイマA、タイマV、タイマW
  - 10bit A/D変換器
  - シリアル通信インターフェース
  - フラッシュROM（プログラム書き込み）32KB
  - RAM（メモリ）2KB



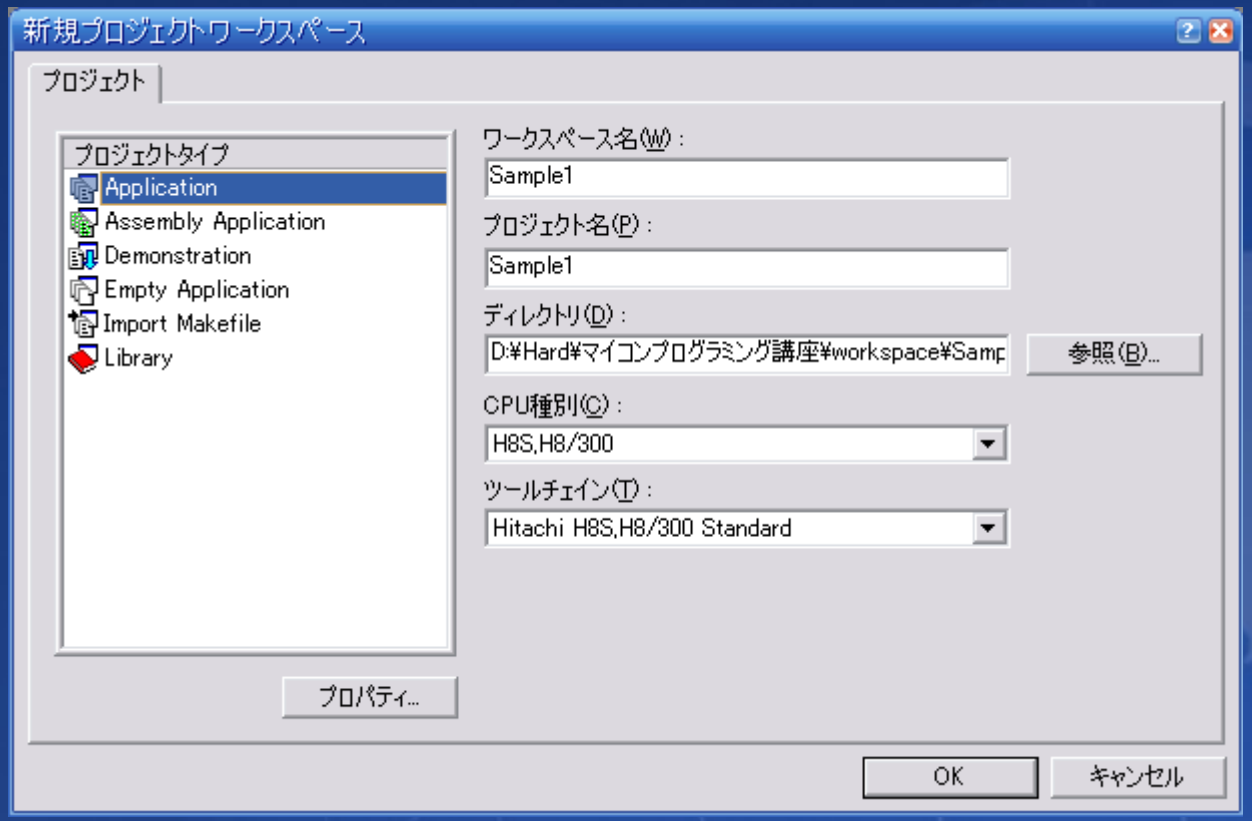
必要なもの

- マイコンキットと説明書
  - 秋月で買ってくる
  - 3664の説明書は秋月のWebサイトに置いてある
- H8/3664 グループ ハードウェアマニュアル
- HEW(コンパイラと開発環境)無償評価版
- FDT(書き込みソフト)無償評価版
  - 3つともRenesasからダウンロード
- USB→RS232C変換ケーブル(ポートが無ければ)
  - 秋月に安いのがある

とりあえずプログラムを書いてみよう

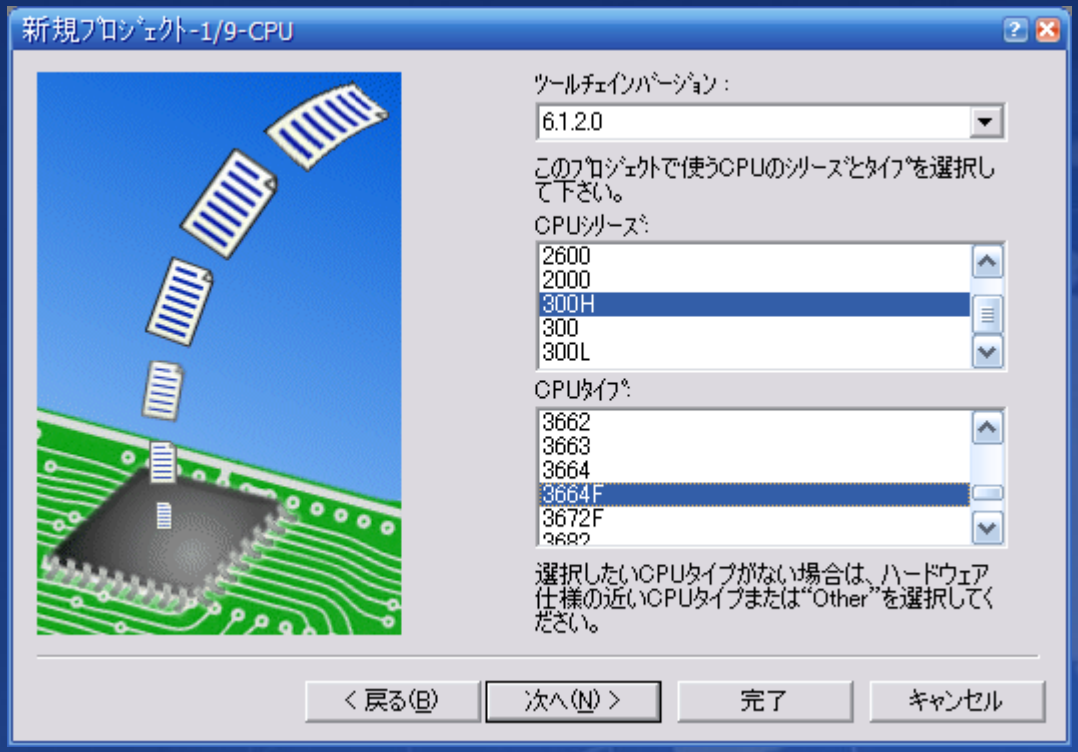
プロジェクトの作成

- プロジェクトとソースコードを保存するディレクトリを設定
- CPUの種類を「H8/300」に設定する



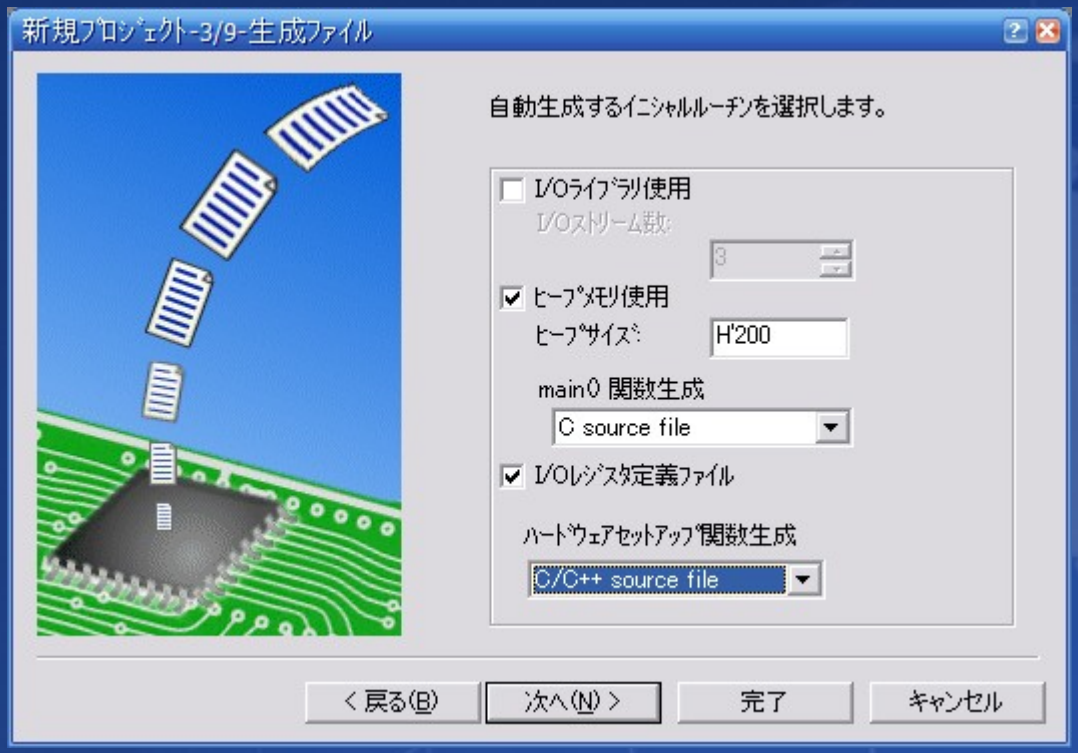
プロジェクトの作成

- ツールチェインバージョンはそのまま
- CPUシリーズは「300H」タイプは「3664F」



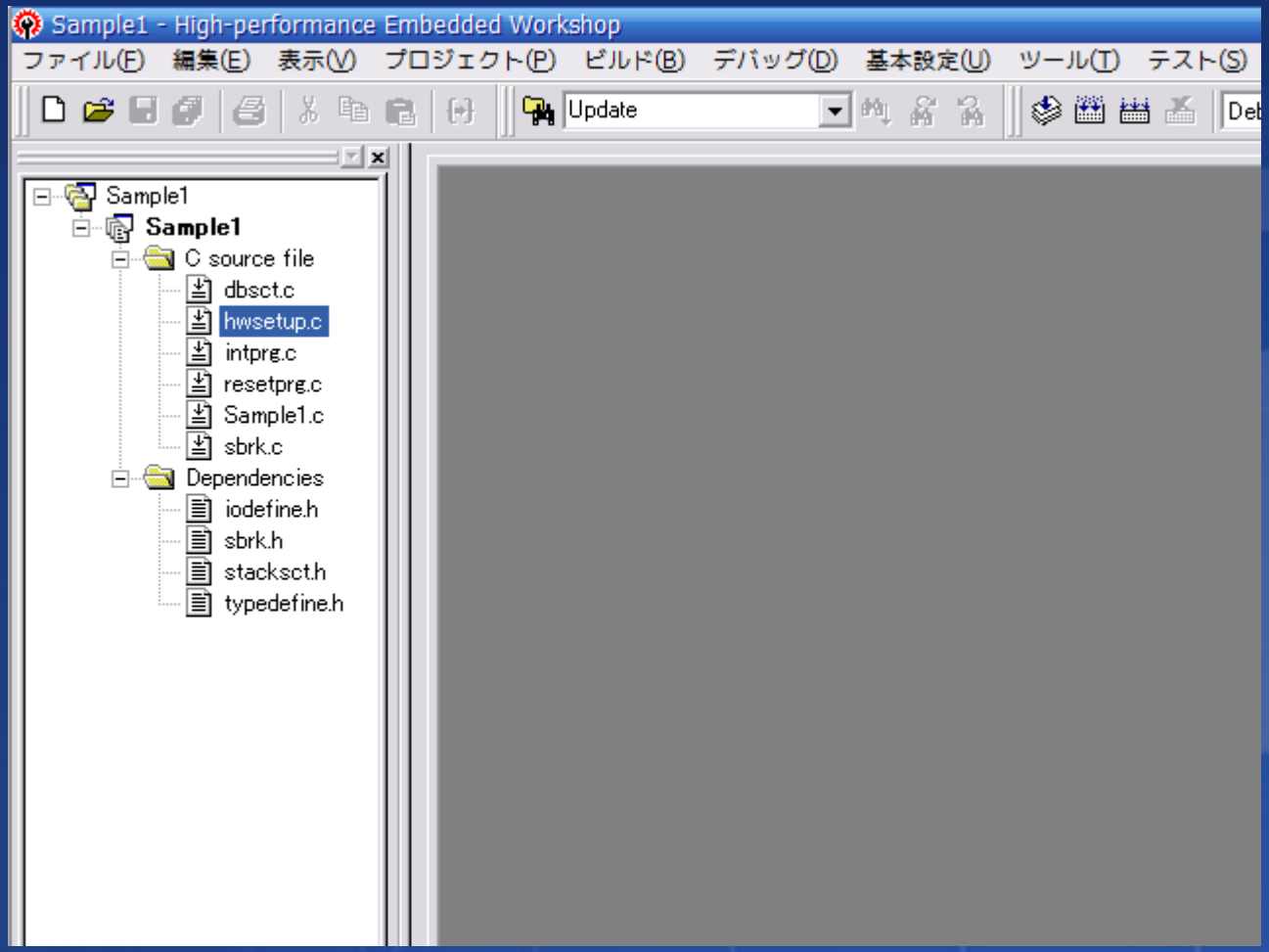
プロジェクトの作成

- 2ページ目は変更せずに次へ
- ハードウェアセットアップ関数生成を C/C++ source fileにする



プロジェクトの作成

- 他のページもそのまま「次へ」をクリック
- 出来上がったワークスペース



プログラム1

- LED点滅プログラム
- main関数({$プロジェクト名}.cに作成されている)に次のプログラムを入力
  - 次のページ
- 出来上がったら「ビルド」
- エラーが無ければ、プロジェクトディレクトリのDebugフォルダにmotファイルが作成される

プログラム1

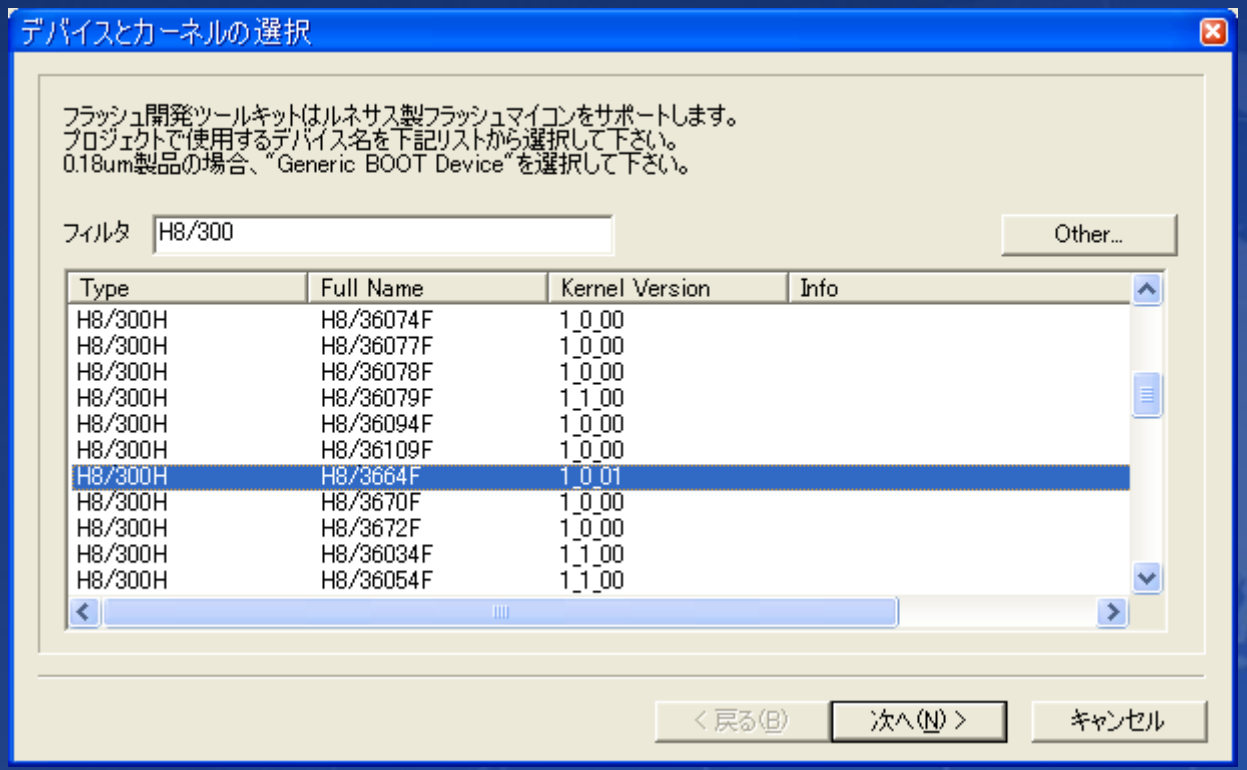
```
#include "iodefine.h"

#ifdef …
中略
#endif

void main(void) {
   unsigned long i;
   IO.PCR8 = 0xff;
   for (;;) {
      IO.PDR8.BYTE = 0xff;
      for (i = 0; i < 500000; i++);
      IO.PDR8.BYTE = 0x00;
      for (i = 0; i < 500000; i++);
   }
}
```

プログラムを転送

- 出来上がったプログラムをマイコンに転送する
- 転送にはFDT (Flash Development Toolkit)を使う

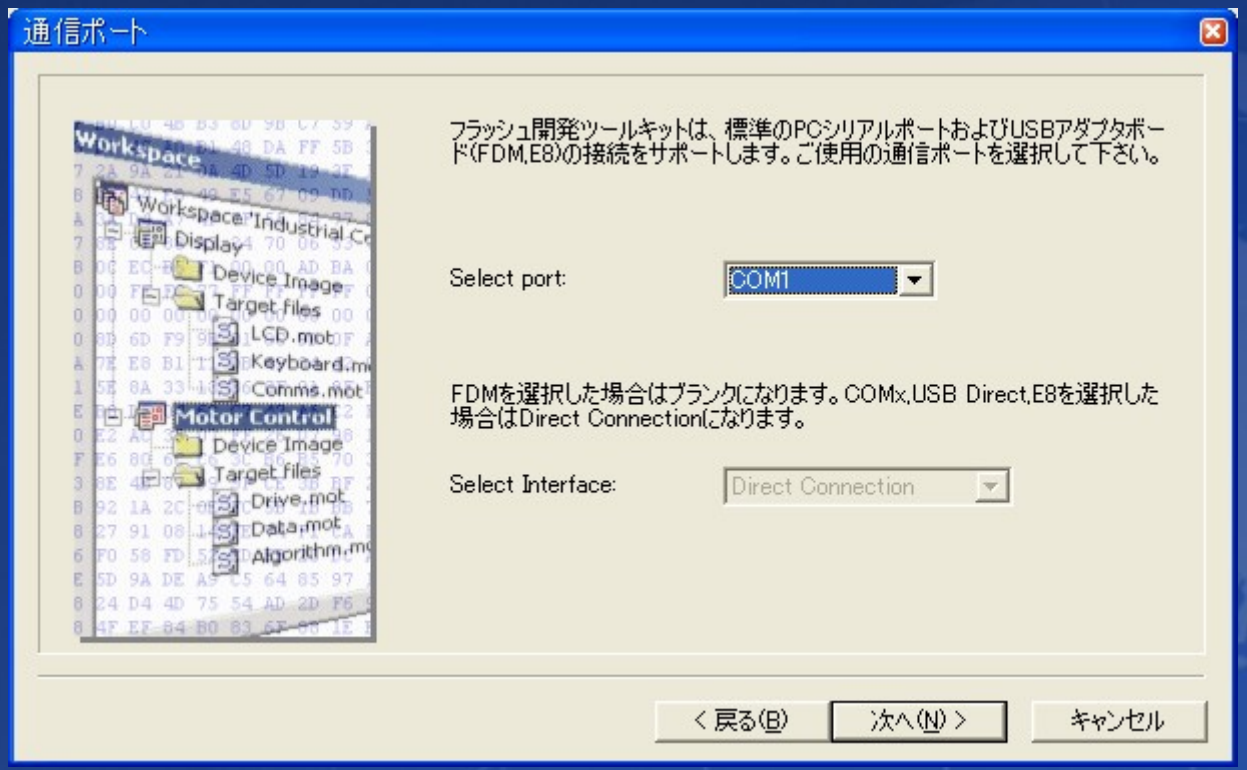
FDTプロジェクトの作成

- H8/300Hを選択



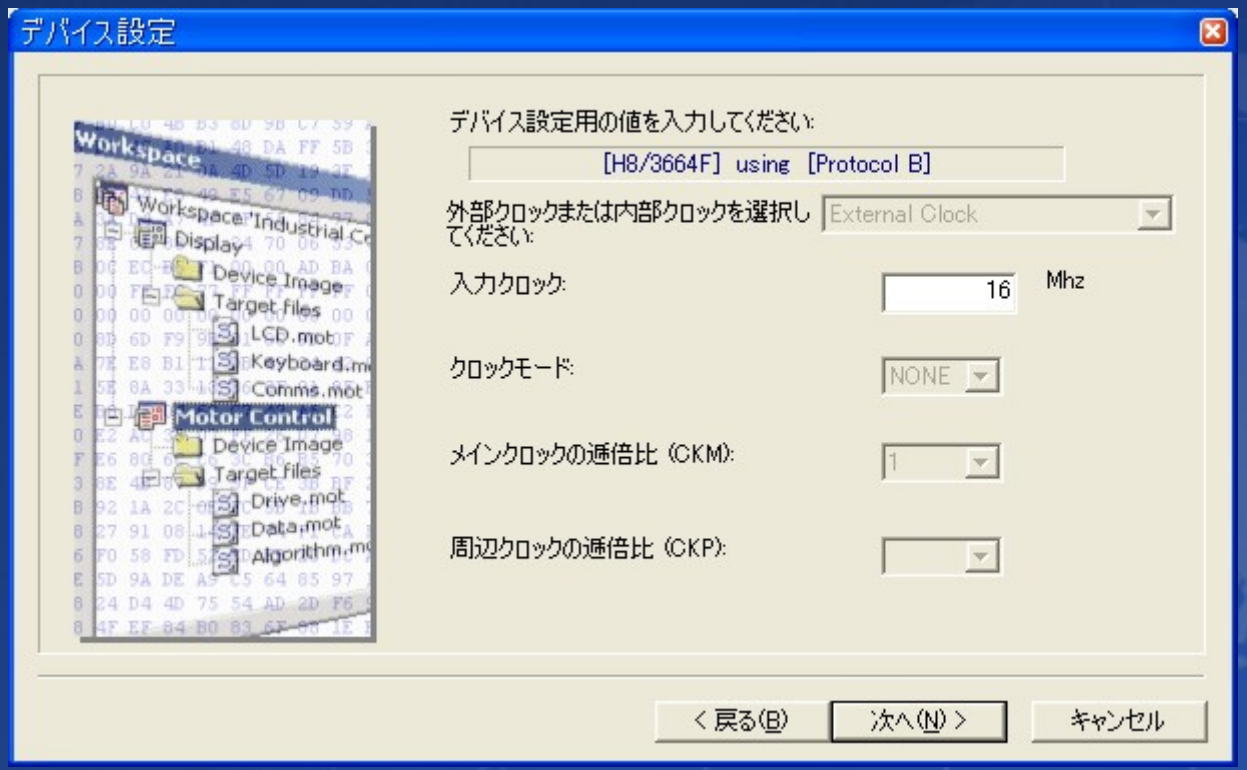
FDTプロジェクトの作成

- ポートはマイコンを接続するポートを選択する
- ポート番号はデバイスマネージャで確認



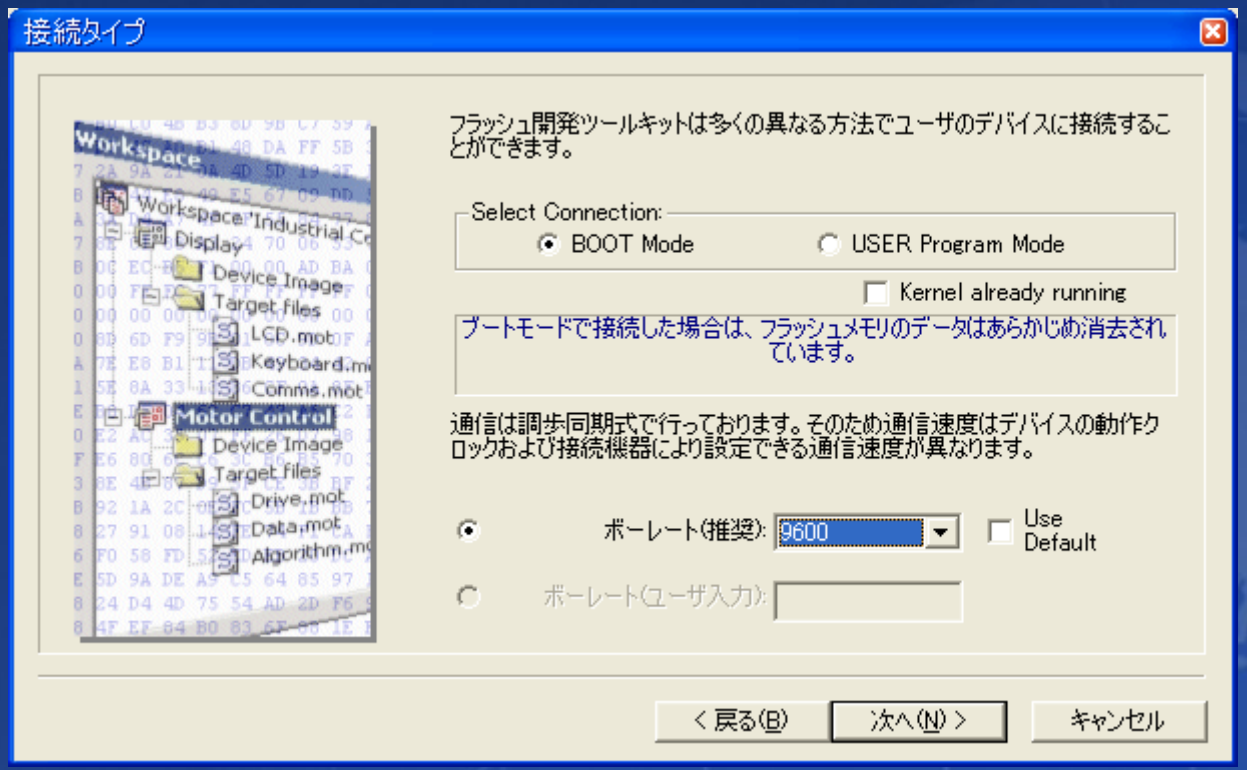
FDTプロジェクトの作成

- 入力クロックを16MHzにする



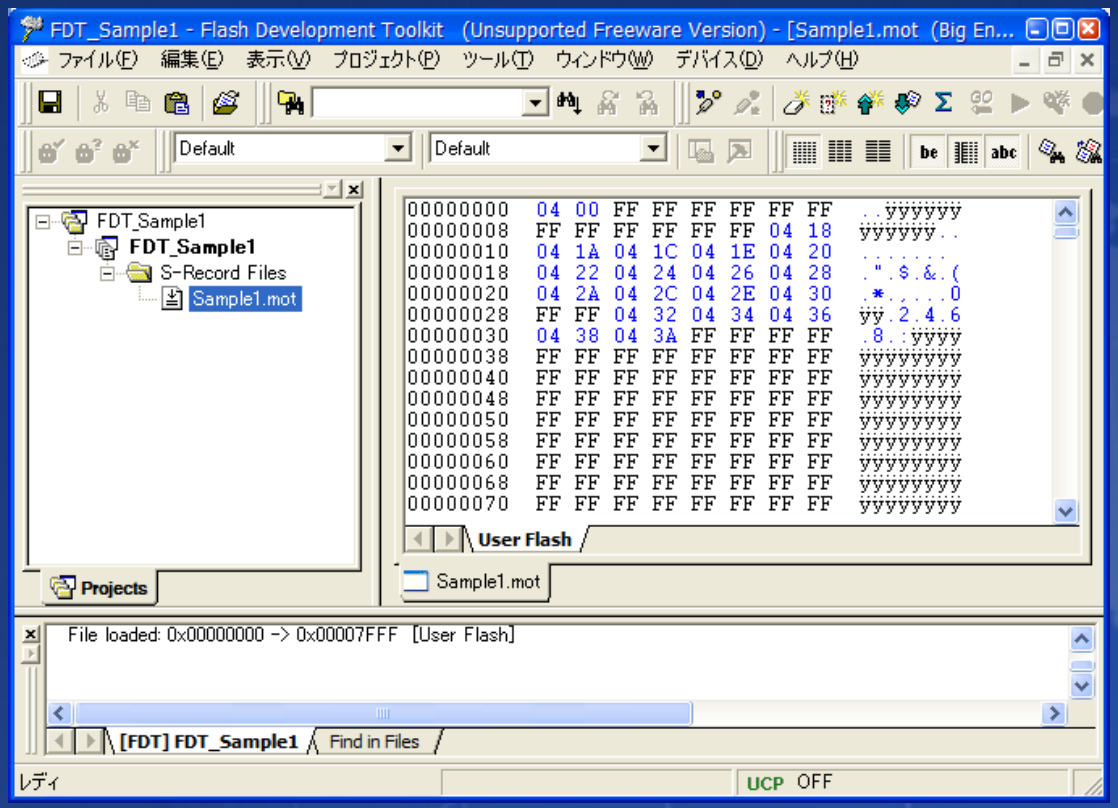
FDTプロジェクトの作成

- 「ボーレート」を9600[bps]に設定
  - 「Use Default」のチェックを外す
- 他のページは初期設定のままでOK



プロジェクトに追加

- 画面左のツリー上の作成されたプロジェクトを右クリックして「ファイルの追加」
- コンパイルして出来上がったmotファイルを選ぶ



マイコンとの接続＆書き込み

- マイコンの電源を切る
- マイコンをブートモードに設定する
  - 今回の場合は、ジャンパーピンを2本挿す
- ケーブルを接続してマイコンの電源を入れる
- motファイルを右クリックして「Download File」をクリック
  - 書き込みが始まる
- 書き込みが終了したら「デバイス」→「デバイスの切断」
- マイコンの電源を切る
- ブートモードを解除する

書き込んだプログラムを実行する

- 通常モードの状態でマイコンの電源を入れると、main関数が実行される

プログラミングの基礎知識

ビット(bit)とバイト(byte)

- ビットとバイト
  - 1[byte] = 8[bit]
  - 1[bit]は0か1(2進数)
- int型のサイズはそのコンパイラで汎用的な大きさ
  - ex) PC用のコンパイラ32bit、3664のコンパイラ16bit
- long型
  - 32bit
- char型
  - 8bit
- ここから先はH8/3664の話(int型は16bit)

符号付き整数と符号なし整数

- 符号付き
  - 負の数も扱える
  - int型 (signed int型)　　−32,768～32,767
  - long型(signed long型)　　−2,147,483,648～2,147,483,648
- 符号なし
  - 0以上の数しか扱えない
  - unsigned int型　　0～65,535
  - unsigned long型　　0～4,294,967,296 (32bit: 4G)

ビット演算

- AND
  - 0 & 0 = 0
  - 0 & 1 = 0
  - 1 & 0 = 0
  - 1 & 1 = 1
  - $(0101)_2$ & $(0100)_2$ = $(0100)_2$
- OR
  - 0 | 0 = 0
  - 0 | 1 = 1
  - 1 | 0 = 1
  - 1 | 1 = 1
  - $(0101)_2$ | $(0010)_2$ = $(0111)_2$

ビット演算

- NOT
  - ~0 = 1
  - ~1 = 0
  - $\sim(0101)_2 = (1010)_2$
- XOR
  - 0 ^ 0 = 0
  - 0 ^ 1 = 1
  - 1 ^ 0 = 1
  - 1 ^ 1 = 0
  - $(0101)_2 \mid (0010)_2 = (0111)_2$

ビット演算

- ビットシフト
  - 符号なしの場合
    - $(10000010)_2 << 1 = (00000100)_2$
    - $(00100011)_2 >> 1 = (00010001)_2$
    - $(10010000)_2 >> 2 = (00100100)_2$
  - 符号付きの場合
    - $(00100010)_2 >> 1 = (00010001)_2$
    - $(10001000)_2 >> 2 = (11100010)_2$

C言語でビット演算

- 優先順位
  - 「~」＞「&」＞「^」＞「|」
- C言語で16進数の書き方
  - 0xFF = $(11111111)_2$ = $(255)_{10}$

問題

- 変数xの2bit目が1であるかの条件文は？
  - 0から数える(1から数える場合もある)
  - 1の位…0bit目

```
if (              ) {
     /* xの2bit目は1 */
}
```

問題(回答)

- 変数xの2bit目が1であるかの条件文は？
  - 0から数える(1から数える場合もある)
  - 1の位…0bit目

```
if (x & 4 != 0) {
     /* xの2bit目は1 */
}
```

```
x = 6  → 0110 & 0100 = 0100
x = 10 → 1010 & 0100 = 0000
```

プリプロセッサ

- #define
  - #define NAME TEXT
  - ソースコード中に現れたNAMEを単純にTEXTに置き換える。
  - 定数を宣言するときに使う事が多い
    - 今はconstを使いましょう　const int MAX_SIZE = 10;

```
#define HOGE printf("test\n")
#define N 10

int main(void)
{
    HOGE;
    printf("%d\n", N);
    return 0;
}
```

の実行結果は？

プリプロセッサ

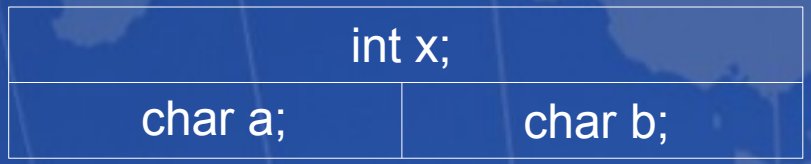
- ヘッダーファイル
  - C言語ファイルと拡張子以外の違いは無い
    - インクルードして使う場合は慣例的に「.h」
  - 主に構造体の宣言や関数のプロトタイプ宣言を複数のソースファイルで共有するときに使う
- #include
  - 指定したファイルの内容を#includeが書かれた位置に単純に挿入する
  - #include <headerfile.h>
    - headerfile.hをインクルードパスに設定されたディレクトリから探す
  - #include "headerfile.h"
    - headerfile.hをインクルードするC言語ファイルと同じディレクトリから探す

構造体と共用体

- 構造体
  - 複数のメンバー変数を1つにまとめる
- 共用体
  - 1つの領域を複数のメンバーで共有する
  - u.x = 0xFF00と代入することは
    u.a = 0xFF, u.b = 0x00と同じ
    - ただしH8(ビッグエンディアン)の場合

```
struct EXAMPLE1 {
    char a;
    char b;
}
```

```
union EXAMPLE2 {
    int x;
    struct EXAMPLE1 ex;
} u;
```

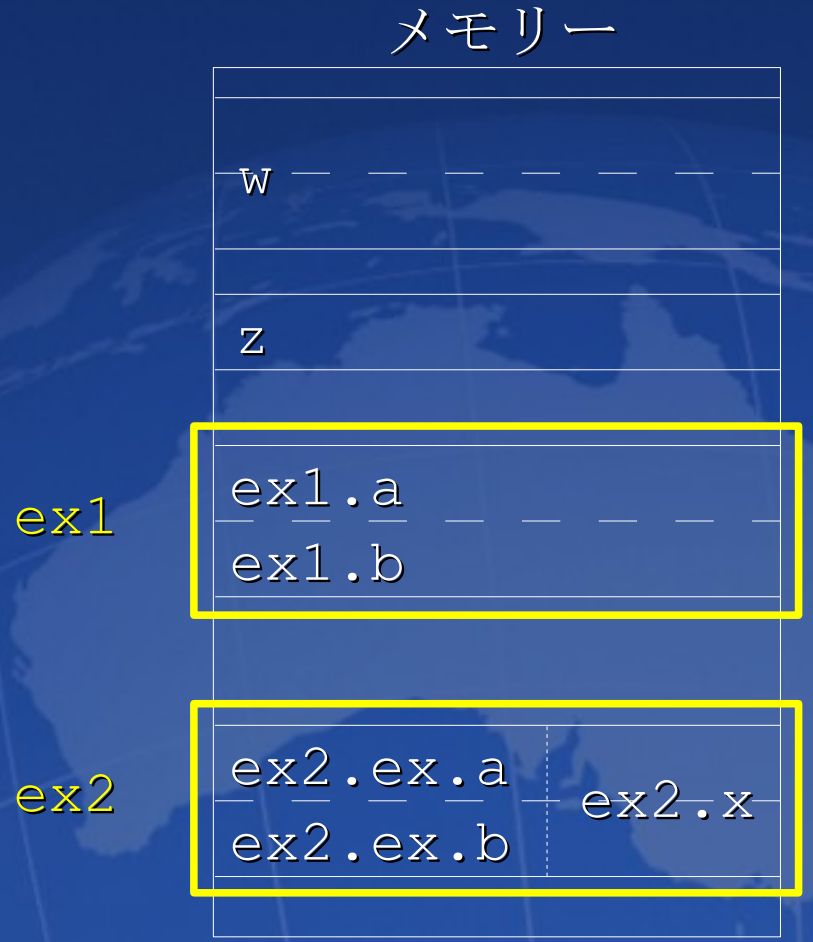


構造体と共用体

- ex2の領域をex (struct EXAMPE1型)としてアクセスするか、x (int型)としてアクセスするか選べる

```
int w;
char z;
struct EXAMPLE1 ex1;
struct EXAMPLE2 ex2;
```

```
ex2.ex.a = 1;
ex2.ex.b = 0;
```
とすると、
ex2.xは256になる



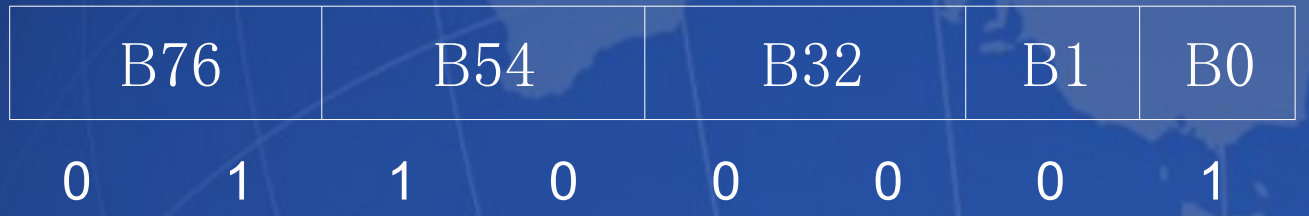
ビットフィールド

- 構造体をビット単位で作ることができる

```
struct TEST {
   unsigned char B76:2;
   unsigned char B54:2;
   unsigned char B32:2;
   unsigned char B1 :1;
   unsigned char B0 :1;
}
```

```
struct TEST t;
t.B76 = 1;
t.B54 = 2;
t.B32 = 0;
t.B1 = 0;
t.B0 = 1;
```

全体は8bit



汎用入出力ポート

汎用入出力ポート

- ポート
  - マイコンの足
  - それぞれのポートにはアドレス(ポインタ)が割り当てられている
  - 1つのポートには複数の機能が割り当てられている
    - ポートコントロールレジスタに値を設定し、どの機能を使うか選択
    - ex) P50とWKP0、PB4とAN0
- そもそもレジスタって何よ
  - CPUに配置された変数
  - ポインタが割り当てられているのでC言語からアクセスできる

汎用入出力ポート

- 汎用入出力ポート
  - P1（ポート1）、P2、P5、P7、P8、PBがある
  - 出力
    - データレジスタに設定した値に応じてポートの電圧が変わる
      0→0V、1→5V（GNDを0Vとした場合）
  - 入力
    - ポートの電圧に応じてデータレジスタの値が変わる（0, 1）
- P8
  - 大電流ポート（他より大きな電流が流れる）
  - 実験用基板でLEDが接続されている
- ハードウェアマニュアルを見てみよう

LEDを点滅させる

- ポート8を汎用出力に設定する
  - PCR8（ポートコントロールレジスタ8）を1にする
- 出力する
  - PDR8（ポートデータレジスタ8）に値を代入すると値に応じた電圧が出力される
- レジスタとポートの対応
  - （PCR8、PDR8）レジスタのビット0 → P80（CN2-9）
  - （PCR8、PDR8）レジスタのビット1 → P81（CN2-10）
  - …
  - （PCR8、PDR8）レジスタのビット7 → P87（CN2-16）

LEDを点滅させる

- C言語で書くと

```
void main(void)
{
    unsigned long i;
    IO.PCR8 = 0xff;                 //P80～P87を出力に設定
    for (;;) {
        IO.PDR8.BIT.B0 = 0x01;   //P81に1を出力
        for (i = 0; i < 100000; i++) {
            ; //何もしない（時間稼ぎ）
        }
        IO.PDR8.BIT.B0 = 0x00;   //P81に0を出力
        for (i = 0; i < 100000; i++) {
            ; //何もしない（時間稼ぎ）
        }
    }
}
```

レジスタへのアクセス

- IO.PCR8って一体どうなってるの？
  - iodefine.hで定義されている
  - IO
    - struct st_io型でレジスタが指定されている
    - st_ioのメンバーにPCR8がある(unsigned char型)
- volatile
  - 最適化禁止
  - 詳しい話は各自調べて

レジスタへのアクセス

- んじゃ、IO.PDR.BIT.B0は？
  - PDR8は共用体 アクセス方法を選べる
    - unsigned char BYTE;
    - struct {}
  - BITは構造体でビットフィールド
    - ビットに直接アクセス
- アクセス方法を選べる
  - IO.PDR.BIT.B0 = 1 と IO.PDR.BYTE = IO.PDR.BYTE | 0x01 は同じ
  - 前者の方が便利

プログラム2

- 8個のLEDのうち4個ずつ交互に点灯するプログラム
  - まずはBIT.B0 = 1を使って書く
  - 次はビット演算を使ってみよう
    - こっちの方が、かなり楽だよね？

プログラム3

- 8つのLEDが1個ずつ順に点灯
  - [○○…○○◎] → [○○…○◎○] → … → [◎○…○○○] →最初に戻る
  - ビットシフトを使ってみよう
    - BITを使うと、かなりメンドクサイ

入力を使ってみる

- P50(CN1-14)に接続されたボタンが押されると…
  - GNDと接続されるのでP50は0Vになる
  - 放された状態だと5V
- ポートを入力に設定すると、PDRのビットに値が格納される
  - P50の状態はPDR5の0ビット目
- P50を汎用入力に設定するには
  - P50は標準状態で汎用入力に設定されている
  - ハードウェアマニュアルを見る

```
if (IO.PDR5.BIT.B0 == 0) {
    //ボタンが押されている
}
```

プログラム4

- 次のプログラムを作ってみよう
  - ボタンが押されている→LED全点灯
  - ボタンが放されている→LED消灯
- ヒント
  - ひたすら、スイッチの状態をチェックする

プログラム5

- 電子ルーレットを作ってみよう

モータードライバーIC

モータードライバーICの使い方

- TA8428K
  - 東芝セミコンダクターの小型モーター用、モータードライバ
  - 信号ピン（IN1とIN2）の値で回転方向をコントロールする優れもの
    - 新月のモータードライバは、高出力で、速度も変えられる
  - データシートを見てみよう
- 入力と動作
  - （IN1, IN2）=（0, 0） ストップ
  - （1, 0） 正転
  - （0, 1） 逆転
  - （1, 1） ブレーキ

プログラム6

- モーターを正転させるプログラムを書いてみよう
  - モータードライバはP80がIN1にP81がIN2に接続されている
- 一定間隔で正転と逆転を切り替えるプログラムを書いてみよう

割り込み

割り込み

- 現在実行しているプログラムを中断して、指定された処理を実行する
- この処理が終わったら、元の処理に戻る
- 割り込みの要因に対応した、割り込み処理を定義する

割り込みのメリット

- 今までの方法では、入力待ちの間、無駄なループがあった
  - 他の作業ができない
  - この動作が白線検知であれば、敵を見つけられない

割り込み

- 主な割り込みの種類
  - NMI　NMI端子が0になると発生する
    - 優先度が最も高い＆NMIを止めることはできない（緊急停止など）
  - IRQ端子　WKP端子が指定した条件を満たすと発生する
    - 0から1に変化したとき（立ち上がりエッジ）
    - 1から0に変化したとき（立ち下がりエッジ）
  - タイマ　タイマーカウンタが設定値に達したとき
  - A/D変換やシリアル通信の処理が終了したとき
- 優先順位
  - 高い優先順位の割り込みは、低い優先順位の割り込み処理中にも割り込める

割り込み

- IRQポートとWKPポートの違い
  - IRQポート
    - IRQ0～IRQ3があり、それぞれに対して割り込み処理を定義できる
    - void INT_IRQ0(void)
    - void INT_IRQ1(void)
    - void INT_IRQ2(void)
    - void INT_IRQ3(void)
  - WKPポート
    - WKP0～WKP5があり、共通の割り込み処理を定義する
    - 割り込み発生条件はポート別に設定できる
    - void INT_WKP(void)

割り込み関数の書き方

- 割り込み処理を割り込み関数として書く
- HEWで自動生成されたintprg.cを編集する

```
__interrupt(vect=18) void INT_WKP(void) {
    /*
     * 空の関数が用意されているので、編集する
     * ここに割り込み処理を書く
     */
}
```

割り込みを使ってみる

- スイッチがWKP0端子（CN1-14）に接続されている
  - 実はIRQ0に接続するつもりで間違えた…
- ポートの初期化（ポートモードの設定など）はhwsetup.cのHardwareSetup関数に書く
  - 前半で使用したPCRの設定も
  - 自動生成されたソースコードを追えば分かるはず
  - 割り込み禁止状態で呼び出される

割り込みを使ってみる(準備)

- ポートモードの変更
  - ポートモードレジスタ(PMR)でWKP0を有効にする
- 条件の設定
  - 割り込みエッジセレクトレジスタで条件を設定
- 割り込みの有効
  - 割り込みイネーブルレジスタ(IENR)でWKPによる割り込みを有効にする
- 参照
  - ハードウェアマニュアル
    - 3.例外処理
    - 9.6 ポート5

割り込みを使ってみる(準備)

- ポートを設定する

```
void HardwareSetup(void)
{
    他のポートの設定

    IO.PMR5.BIT.WKP0 = 1; /* WKP0として使う */
    IEGR2.BIT.WPEG0 = 0;  /* 立ち下がりエッジで割り込み */
    IENR1.BIT.IENWP = 1;  /* WKPポートの割り込みを有効化 */
}
```

割り込みを使ってみる(処理)

- 割り込み関数を編集する
  - iodefine.hのインクルードを忘れずに!!
- まず割り込み要求フラグをクリアする
  - IRQポートなど、他の割り込みの場合も、相当する要求フラグをクリアする
    - 割り込み処理完了通知
    - クリアしないと、何度も割り込み関数が実行されてしまう

```
__interrupt(vect=18) void INT_WKP(void) {
    /* WKP0の割り込み要求フラグをクリア */
    IWPR.BIT.IWPF0 = 0;

    ここに、割り込みが発生した時に実行したいプログラムを書く

}
```

割り込み禁止

- 割り込みを禁止すると、割り込みが発生できなくできる
  - 割り込みが発生するとマズイタイミングで使用する
  - 詳細は各自調べて(相撲ロボットのプログラムでは必要)
- 割り込み禁止
  - set_imask_ccr(1);
- 割り込み禁止解除
  - set_imask_ccr(0);
- 割り込み禁止区間に入る際に、状態が分からない場合

```
unsigned char ccr;
ccr = get_imask_ccr();  /* 状態を待避 */
set_imask_ccr(1);
割り込み禁止で実行する処理
set_imask_ccr(ccr);
```

プログラム7

- プログラム6を改造して、ボタンが押されたらモーターを止めるプログラムを書いてみよう
  - main関数は終了してしまうと困るので、今回は無限ループを入れておく

タイマ

タイマ

- 今までは無駄なfor文で時間調整をしていた
  - 正確な時間を計りたい
  - 他の作業ができない→割り込みを使いたい
- タイマ
  - 一定間隔でタイマレジスタをカウントアップ
  - タイマーレジスタの値が条件を満たせば、割り込みを発生

タイマの種類

- タイマの種類(詳しくはハードウェアマニュアル)
  - タイマA → 今回はこれを使う
    - 1～8kHz
    - 時計
    - 広い間隔
    - AKI-H8/3664では32.768KHzのクリスタルが接続されている
  - タイマV
    - 8bitタイマ
  - タイマW
    - 16bitタイマ
    - 短い間隔
    - PWM(最大3出力)など

タイマAの初期化

- 0.5秒タイマに設定し、割り込みを有効にする
  - ハードウェアマニュアルのタイマAを参照

```
void HardwareSetup(void)
{
    TA.TMA.BIT.CKSI = 0x09;  /* (1001)2 0.5秒タイマ */
    IENR1.BIT.IENTA = 1;     /* タイマAの割り込みを有効 */

    /* LED点滅用 */
    IO.PCR8 = 0xff;
}
```

タイマAの割り込み関数

- 他の割り込みと同様にintprg.cを編集する

```
__interrupt(vect=19) void INT_TimerA(void) {
    /* タイマA（TA）の割り込み要求フラグをクリア */
    IRR1.BIT.IRRTA = 0;

    /* ここに割り込み処理を書く */
    /* LED点滅 */
    IO.PDR8.BYTE = ~IO.PDR8.BYTE;
}
```

プログラム8

- 4秒ごとにモーターの回転方向を変えるプログラムを書いてみよう
  - 8回0.5秒の割り込みが発生
  - 大域変数を使う(またはstatic変数)

その他の機能
（相撲ロボットには必要）

シリアル通信

- RS232Cを通して、PCに文字列を送信したりするのに使う
  - デバッグなどに便利
- 新月用に軽量版stdioライブラリがある
  - 流用するなり、参考にするなり…

A/D変換

- アナログ→デジタル変換
- ポートの電圧を測定し10bitの整数として取得する
  - 汎用入力では、ポートの電圧が閾値より高いと1、低いと0
- 新月のPSDセンサーは距離に応じた電圧を出力
  - 電圧を測定して、距離を計算する

その他

- PWM出力(タイマW)
  - サーボモーターの制御やモーターの速度制御に必要
- ポートプルアップ
  - ポートをプルアップする