概要

前回は、I2C I/Oエクスパンダー MCP23017をバイト単位でアクセスする基本的な アクセス関数を作成しました。 本来、I2Cインタフェースはバイト単位で、I2Cデバイスのアクセスを行います。

よって、ポートにアクセスするバイト単位のデータの 特定の bitだけ書き込んだり、読み出したりする機能は、I2Cインタフェースでは、サポートされていません。

実は マイコン内蔵のポートにおいても CPUコアが、周辺回路のI/Oポートを アクセスする単位は バイトの場合が、殆どです。 それは、CPUコアと 周辺回路を接続する、データバス幅が、バイトまたは ワードが 殆どだからです。 よって I/Oポート内の 特定の 1bit だけをアクセスする機能は バスラインのハードウェアは 持っていません。最低 バイト単位なのです。

でも、例えば R8Cマイコンの C言語でのビット操作の記述で p1_3 = 1; とか あります。これはポート1の bit 3 に 1を 入れるという事です。（ 処理系によっては p1.3 = 1; とか記述する場合も あります。） R8Cマイコンの場合は、この様なビット操作は、CPUの命令で 存在します。

R8Cマイコンで なくても 組み込み用マイコンであれば ビット操作命令は、有ると思います。

これらの命令は、命令実行時 イグゼキュートサイクル時に、3つの処理を連続的に行う物と思います。

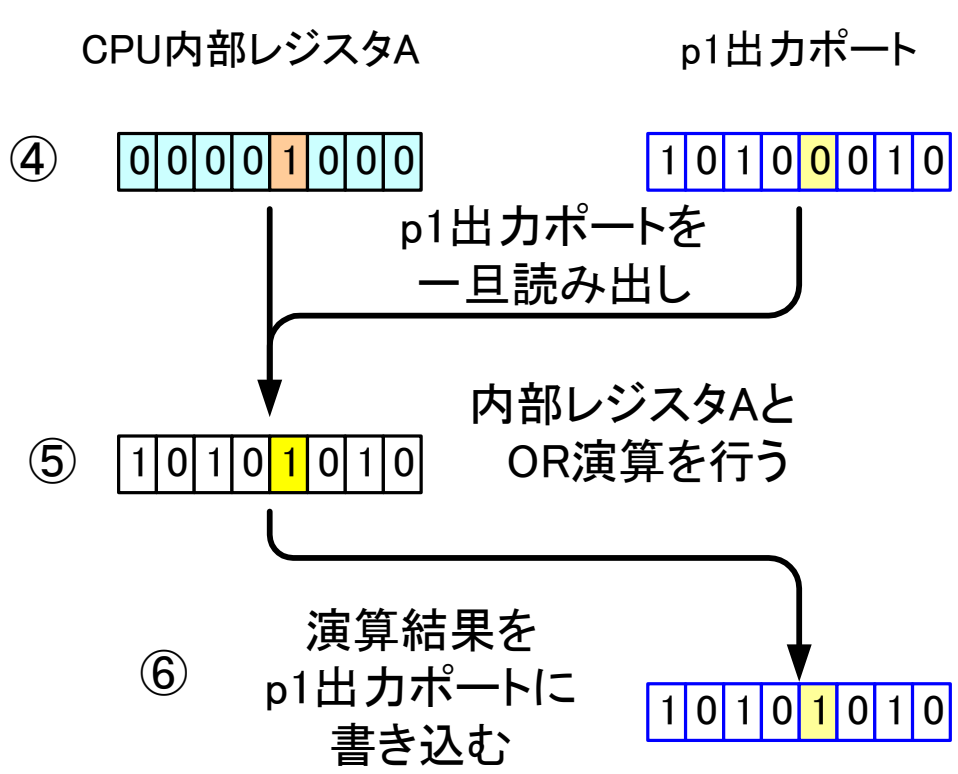
① バイト単位で 出力ポート上の値を 一旦 CPU内部レジスタに 読込みます。

② CPUレジスタ上で 1を立てる場合は OR命令、 0 を書き込む場合は AND命令で ビット操作を行います。

③ ビット演算処理した結果を、バイト単位で、読み出した出力ポートに 書き込みます。

④ 例えば、出力に設定された p1のポートに p1 = 1010 0010 が 設定されていたとします。 で、p1_3 = 1; を 行う場合を考えてみます。 ちなみに p1_3 だけが 1 の 状態を考えると 0000 1000 になります。
これを、仮に CPU内部レジスタAに入れておきます。 そして、p1出力ポートの内容を一旦 読み出します。

⑤ 内部レジスタAと、読み出した p1ポートの値とで、OR演算を行い、結果を 内部レジスタAに置きます。



⑥ そして、内部レジスタAの 演算結果を p1出力ポートに 書き込みます。
よって、ビットセット、ビットリセット命令は 読込み、演算、書込みの 3ステップが必要という事になります。

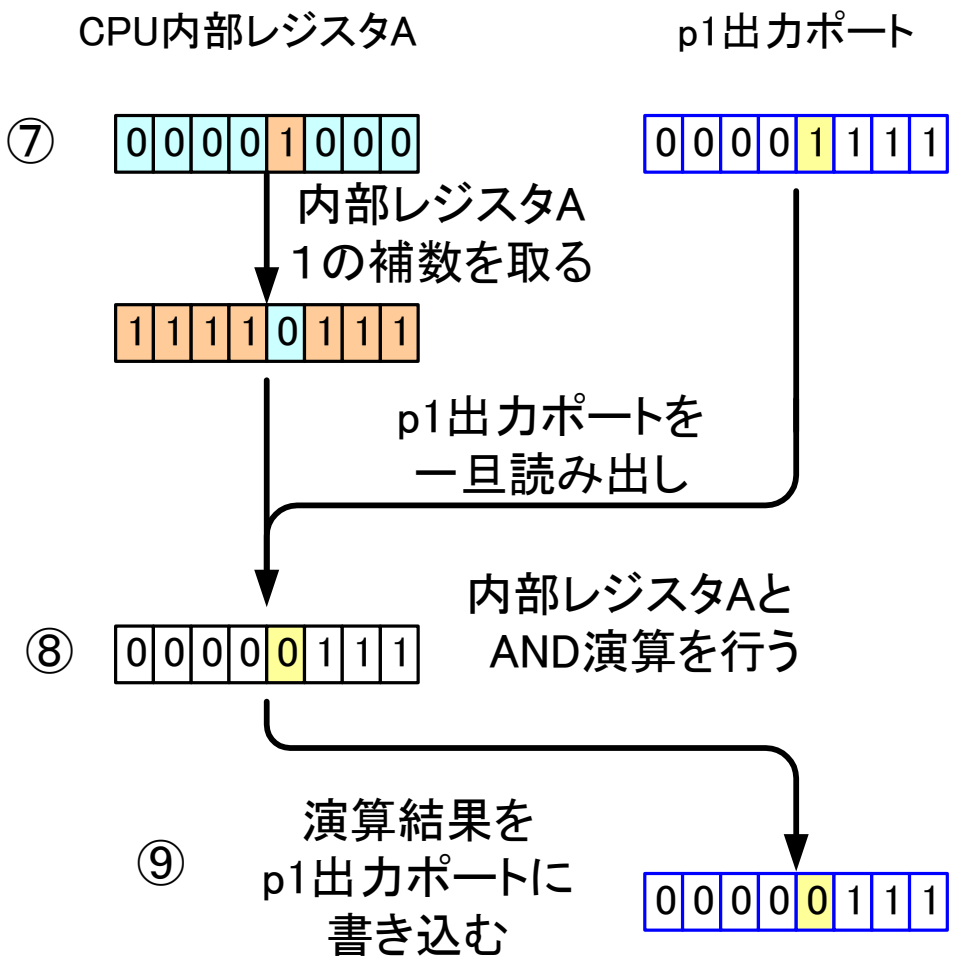
0の ビットを書き込むというか ビットリセットの場合も、一応示します。 今度は、p1ポートに 0000 1111 が設定されていて p1_3に 0 を書込む場合を示します。

⑦ 命令のオペランドからロードされた内部レジスタAの値の 1の歩数を取ります。
同時に、p1出力ポートの値を 一旦読込みます。

⑧ 内部レジスタAと p1の値とで、AND演算を行います。

⑨ 演算結果を p1ポートに書き込みます。

CPUの内部回路は分からないので何とも言えませんが、多分 命令オペランドから 値を取り込むと同時に 1 の歩数を取っているかもしれませんね。ハードで bit反転するのは、NOTゲートで、済みますからね。



何故、ビットセット命令、ビットクリア命令の動作説明を　行ったかというと、I2Cの I/Oエクスパンダー MCP23017も、先頭ページで説明した通り、ポート内の単独ビットの書き換えは出来ません。 で、それをソフトで、ビットセット、ビットリセットを行う場合、基本的に　前述のビットセット命令、ビットクリア命令と同様に、一旦出力ポートの値を読み込んで、AND、OR命令を行い再書き込みします。 で、当然ですが 400kbps の I2Cシリアル通信でデータをやり取りするので、一旦読んで書き込むという　二度手間的な事を I2Cで行うと処理が遅くなるのは容易に想像が 出来ます。そして、ビット毎に 8bit分 あるいは 16bit分　連続してビット設定を行うと 更に遅くなります。

単純に考えると、1byteまとめてポートに書き込む場合と比べて、ビット毎に 読んで書いてを 8bit 分 行うと、16倍 時間が　かかります。

という事で、I2Cのデータアクセスの頻度を多少でも減らすため、ポートデータの キャッシュメモリというか、各ポートに出力したデータの履歴を保存して、その保存した履歴データを利用してポートからの一旦読み出しを廃止して、代わりに履歴データと ビット操作を行い書き込む、そして書込み時　ポート履歴データの更新を行う という高速化手法を 用いて 今回 MCP23017の ビット単位のアクセス関数を 作ろうと思います。 時間は、約半分早くなると思います。

読み込みに関しては、外部の事象は　いつ更新されるか分かりませんので、毎回 読み込む必要が あります。 関数の関数値としては、指定したポートのビット状態を　1と　0 で返そうと思います。

I/O Exp MCP23017 周辺回路

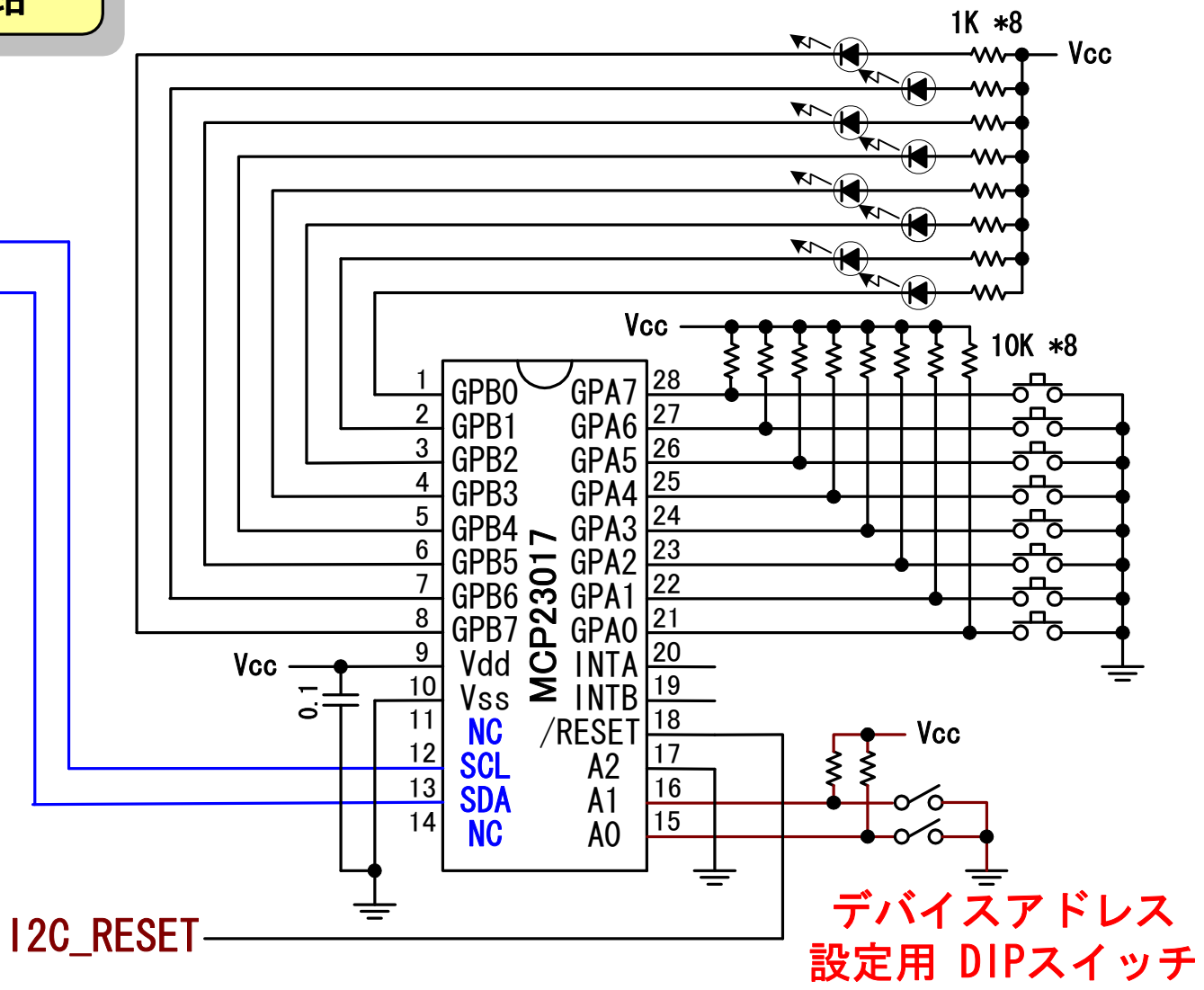
MCP23017(I2C仕様)の ピン アサイン

足ピンの 11 ～ 14 までの 4ピン部分が、SPI仕様と、I2C仕様で異なります。

I2Cでは、11ピンと 14ピンが NC 未接続となっています。
12ピンが SCL、13ピンが SDA です。

デバイスアドレス設定の A2 A1 A0 の A2は、GND接続で、A1と A0 に 0 ～ 3 の アドレスが 設定出来る様に DIPスイッチを 付けました。

それと この MCP23017は 別途 RESET信号が 必要です。



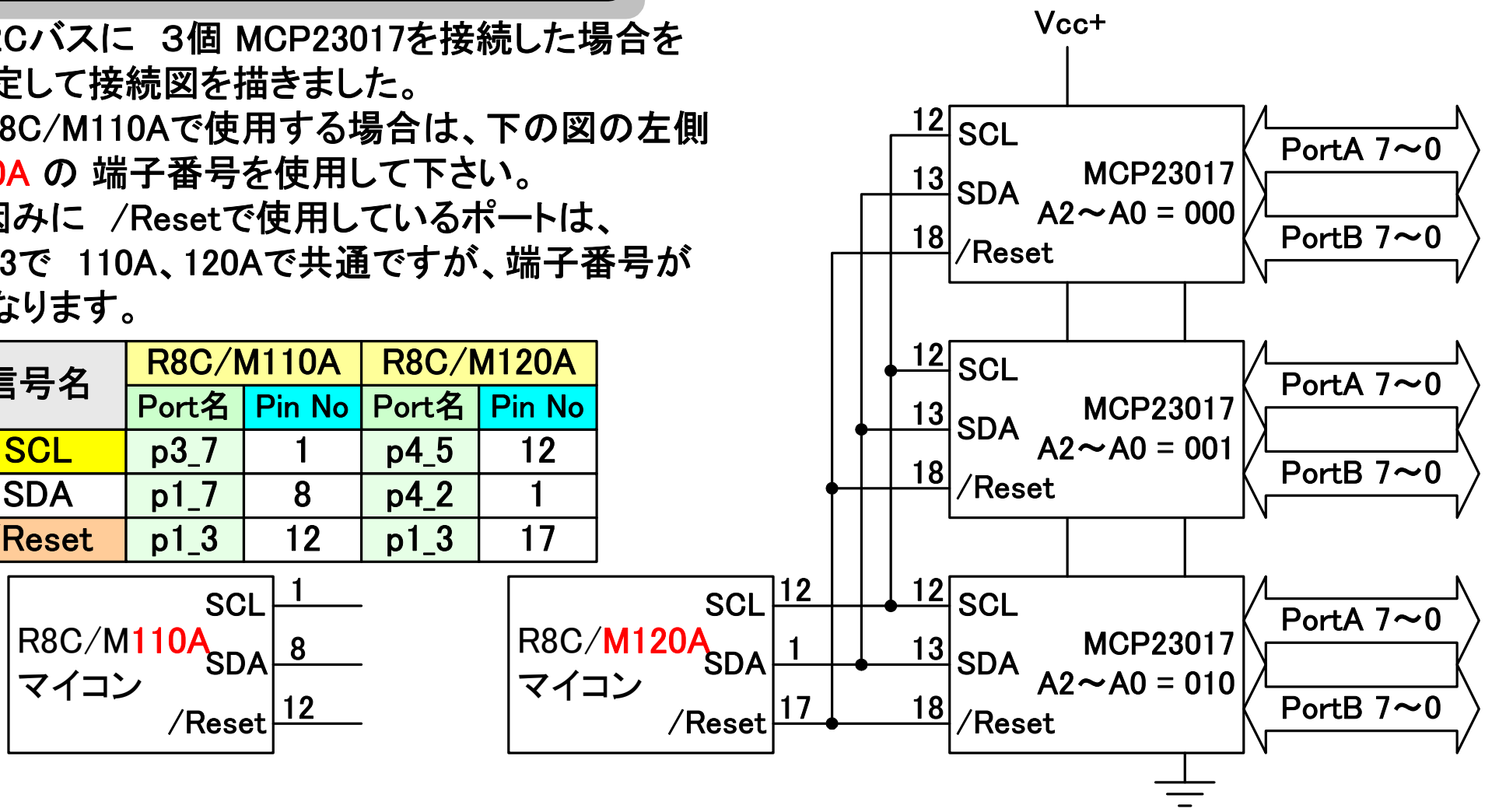
MCP23017 複数接続の場合

I2Cバスに　3個 MCP23017を接続した場合を想定して接続図を描きました。

R8C/M110Aで使用する場合は、下の図の左側110A の 端子番号を使用して下さい。

因みに　/Resetで使用しているポートは、p1_3で　110A、120Aで共通ですが、端子番号が異なります。

信号名	R8C/M110A		R8C/M120A	
	Port名	Pin No	Port名	Pin No
SCL	p3_7	1	p4_5	12
SDA	p1_7	8	p4_2	1
/Reset	p1_3	12	p1_3	17



I2c_Io_Expander.c 内の 関数群

```
void  ioexp_init( BYTE adr );                 // MCP23017 初期化
void  ioexp_pa_dir( BYTE adr, BYTE ptn ); // MCP23017  ポートＡ 入出力方向の設定
void  ioexp_pb_dir( BYTE adr, BYTE ptn ); // MCP23017  ポートＢ 入出力方向の設定
void  ioexp_pa_out( BYTE adr, BYTE dat ); // MCP23017  ポートＡ バイトデータ出力
void  ioexp_pb_out( BYTE adr, BYTE dat ); // MCP23017  ポートＢ バイトデータ出力
BYTE  ioexp_pa_in( BYTE adr );                // MCP23017  ポートＡ のデータ取り込み
BYTE  ioexp_pb_in( BYTE adr );                // MCP23017  ポートＢ のデータ取り込み

//  I/O エクスパンダー ビット処理関数
void  ioexp_pa_setbit( BYTE adr, BYTE bit_no ); // ポートＡ 特定の bitを 1にする
void  ioexp_pa_clrbit( BYTE adr, BYTE bit_no ); // ポートＡ 特定の bitを 0にする
void  ioexp_pb_setbit( BYTE adr, BYTE bit_no ); // ポートＢ 特定の bitを 1にする
void  ioexp_pb_clrbit( BYTE adr, BYTE bit_no ); // ポートＢ 特定の bitを 0にする
BYTE  ioexp_pa_getbit( BYTE adr, BYTE bit_no ); // ポートＡ 特定の bitを 0 or 1で 取り出す
BYTE  ioexp_pb_getbit( BYTE adr, BYTE bit_no ); // ポートＢ 特定の bitを 0 or 1で 取り出す

void  ioexp_reset( void );// I2Cデバイス用 リセット出力
```

は、R8CM1_IOCS_I2C_packet.c 内に 入っています。

因みに、赤文字の ADRは MCP23017の デバイスアドレスの 0 ～ 7 です。
bit_no は、8bit データ内の bit指定の番号 0 ～ 7 です。

MCP23017 複数接続時のプログラム

一つ 記載してませんでしたが、I2Cバスに接続するデバイスにおいて、リセット信号を必要とする場合は、マイコンの パワーオン リセット信号を 接続してはいけません。

マイコンの初期化処理において、1bitの出力ポートから I2Cデバイスのリセット信号を 出して下さい。100us程度のパルス幅の Lowアクティブ信号として出力して下さい。

理由:

1. 今回の R8Cマイコンの CPUリセット信号はCRの積分回路で、ドライブ能力が弱いので長く引きまわすと ノイズを拾う恐れが ある事です。
2. 電源ONでは、無いけれど 場合により 再初期化で 周辺機器の リセットを行う事も 考えられるからです。

今回は、p1_3 から出す事にしました。

1.0 初期化

1.1 最初に、I2Cデバイスに リセット信号を 出します。

```
ioexp_reset();   // I2Cデバイス用 リセット出力( 必要のある時のみ使用する )
```

1.2 接続されている I/O エクスパンダ MCP23017の個数分 初期化処理を 行う。

```
ioexp_init( 0 );   // I/Oエクスパンダ 1号 初期化
ioexp_init( 1 );   // I/Oエクスパンダ 2号 初期化
ioexp_init( 2 );   // I/Oエクスパンダ 3号 初期化
```

引数は、MCP23017の デバイスアドレス設定の A2、A1、A0 端子の 設定値です。

1.3 各 I/O エクスパンダの ポート入出力の
設定例を 示します。

```
ioexp_pa_dir( 0, 0xFF ); // デバイス
```
アドレス 0 の ポートAの設定 全 bit 入力

```
ioexp_pb_dir( 0, 0x00 ); // デバイス
```
アドレス 0 の ポートBの設定 全 bit 出力

```
ioexp_pa_dir( 1, 0xF0 ); // デバイス
```
アドレス 1 の ポートAの設定
上位 4 bit 入力、下位 4 bit 出力

```
ioexp_pb_dir( 1, 0x0F ); // デバイス
```
アドレス 1 の ポートBの設定
上位 4 bit 出力、下位 4 bit 入力

```
ioexp_pa_dir( 2, 0xAA ); // デバイス
```
アドレス 2 の ポートAの設定 bit入出力の
設定: in、out、in、out、in、out、in、out

```
ioexp_pb_dir( 2, 0x55 ); // デバイス
```
アドレス 2 の ポートBの設定 bit入出力の
設定: out、in、out、in、out、in、out、in

どうでしょうか。 入出力設定の イメージは、
掴めたでしょうか。?

2.0 次は、デバイス番号 0 の ポートA、ポートB
の データ入出力の例を 示します。

```
sw = ~ioexp_pa_in( 0 ); // データ入力
```
デバイス 0 の ポートAから バイトデータを
読み込み ~ は bit反転を 行い 変数 swに
代入します。
~ は 負論理のデータを 正論理に 変換す
る時に 使用します。 逆も 出来ます。

```
ioexp_pb_out( 0, sw ); // データ出力
```
バイト変数 sw の内容を、デバイス 0 の
ポートBへ 出力します。

一つのポート内に入出力が混在している場合

2.1 一つのポートが、全ビット入力、あるいは全ビット出力の場合は、考えやすいですが一つのポート内に 入力の bit と、出力の bit が、混在している場合は、どうなるのでしょうか。？ 以下の入出力設定の場合で説明します。

```
ioexp_pa_dir( 1, 0xF0 ); // デバイスアドレス 1 の ポートAの設定
上位 4 bit 入力、下位 4 bit 出力
sw = ioexp_pa_in( 1 ); // データ入力
```

で、デバイス1の ポートAを読み込むと上位 4 bit は、入力なので、外部の信号を取り込む事が 出来ます。 下位 4bit は、出力ポートに設定されているので、ポートAの出力レジスタに 設定されている内容を読み込む事に なります。 例えば 0xAAを直前に ポートAに出力している場合は、読み出すと 下位 4bit は 1010です。

2.2 左の デバイス1 ポートAの設定で、出力を行うと どうなるでしょうか。？

```
ioexp_pa_out( 1, sw ); //デバイス 1 の ポートAへの出力
```

sw の 下位 4bit が、ポートAへ出力されます。 では、上位 4bit は どうなるのでしょうか。？
変数 sw に何らかのデータが 設定されていても 信号は出力されません。
上位 4bit は 入力ポートに設定されているので、Hiでも Low でもない状態 ハイ インピーダンス状態になります。

ハイインピーダンス状態になる可能性の有る端子は、10kΩ ぐらいの抵抗で Vccに プルアップしておく方が 安全です。

bit 単位でアクセスする関数の使い方

その前に、I/Oポートアクセスを イメージしやすいように、I/Oポートに接続する物を 仮に設定しておきます。

I/O Exp Adr=0 Port A	b7	出力	RELAY_1	High Active
	b6	出力		
	b5	出力	LED_2	Low Active
	b4	出力	LED_1	Low Active
	b3	入力		
	b2	入力		
	b1	入力	H_LIM_SW	Low Active
	b0	入力	L_LIM_SW	Low Active

I/Oポートの初期化：

```
ioexp_pa_out( 0, 0x30 ); // ポート出力
      仮データ ( 0011 0000 )
ioexp_pa_dir( 0, 0x0F ); // 入出力設定
      上位 4bit 出力、下位 4bit 入力
```

信号名で bit番号を 設定出来るように #define で 宣言する。

```
#define  RELAY_1  7  // RELAY_1の bit_no
#define  LED_2    5  // LED_2の bit_no
#define  LED_1    4  // LED_1の bit_no
#define  H_LIM_SW 1  // H_LIM_SWのbit_no
#define  L_LIM_SW 0  // L_LIM_SWのbit_no
```

RELAY_1、LED_2、LED_1をアクティブ化する。

```
ioexp_pa_setbit( 0, RELAY_1 );// Hi出力
ioexp_pa_clrbit( 0, LED_2 );  // Low出力
ioexp_pa_clrbit( 0, LED_1 );  // Low出力
```

RELAY_1、LED_2、LED_1を ノーマル化する。

```
ioexp_pa_clrbit( 0, RELAY_1 );// Low出力
ioexp_pa_setbit( 0, LED_2 );  // Hi出力
ioexp_pa_setbit( 0, LED_1 );  // Hi出力
```

上限、下限の リミットスイッチの読み出し

```
sts = ioexp_pa_getbit( 0, H_LIM_SW );
sts = ioexp_pa_getbit( 0, L_LIM_SW );
```

bit 単位でアクセスする 別の方法

実は、終り近くになって、別の方法を 思いつきました。 これも I/Oポートの キャッシュというか、マイコンの RAM上に ビットフィールドの構造体を用意して、各ビットに分かりやすい名前を付けて、RAM上のポートキャッシュをビット単位で設定してから、バイト単位で 一発でポートに出力する方法です。
ビットフィールドの構造体は、以下のように宣言します。

```
typedef struct {
      unsigned char  relay_1:  1;
      unsigned char  pad1:     1;
      unsigned char  led_2:    1;
      unsigned char  led_1:    1;
      unsigned char  pad2:     2;
      unsigned char  h_lim_sw: 1;
      unsigned char  l_lim_sw: 1;
} IoX0_PA;      // データ型の名前
```

使う時は、まず ビットフィールド変数の宣言です。

```
IoX0_PA   iox0_pa;//  変数の宣言

// 各出力項目を設定する
  iox0_pa.relay_1 = 1;
  iox0_pa.led_2 = 0;
  iox0_pa.led_1 = 0;

// バイト単位で ポートAへ書き込む
  ioexp_pa_out( 0,  (BYTE)iox0_pa );
```

ビットフィールドは、各Cコンパイラ処理系に於いて、型宣言のビットフィールド行の 上からの並びが、b7 b6 b5 となっているのか、b0 b1 b2 となっているのか 方言がある様なので、実際にコンパイラで確認する必要が あります。

逆に、1バイト読み出して 各ビットを RAM上で調べる事も可能と思われます。 これが、うまく行くならば 可視性のいいコーディングで、かなり実行速度アップが 図れます。