第12章 アレイ通信の方式とプログラミング

―― 二つの付属基板をケーブルで繋いで同期させる

本章では，複数の付属基板(MB)間でアレイ通信を行う方式について説明します．付属基板(MB)間をアレイ接続ケーブルで繋いで，フル・カラーLEDの点滅を同期させてみます．

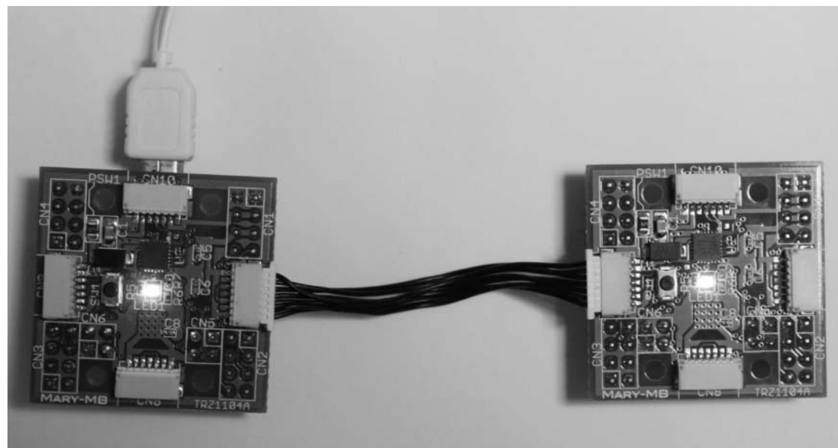


写真1　複数の付属基板(MB)間でLED点滅を同期させる

【用意するもの】
付属基板(MB)：2枚以上(はんだ付けなどの加工は不要)
アレイ接続ケーブル：付属基板(MB)間接続に必要な本数
使用するプロジェクト：PROG02_COLOR_LED_MULTI

● プロジェクトPROG02_COLOR_LED_MULTIの実験

まず付属基板(MB)を2枚用意して，**写真1**のようにアレイ接続ケーブルで繋いでください(第3章を参照)．プロジェクト「PROG02_COLOR_LED_MULTI」をビルドして，完成したバイナリを両方の付属基板(MB)にダウンロードしてください．

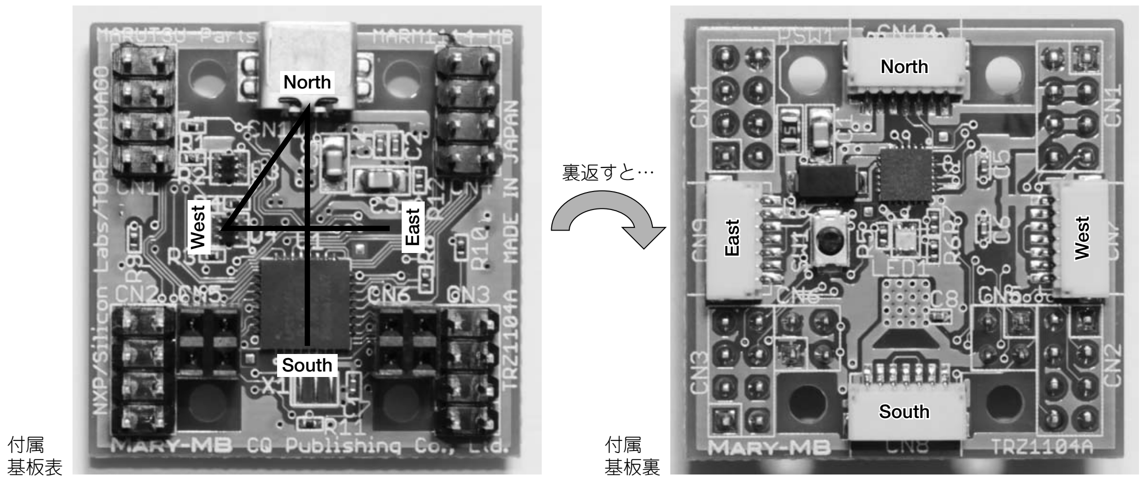
写真1のように付属基板(MB)間で同期を取って，フル・カラーLEDが色を変えながら点滅します．

なお，個々の付属基板(MB)上のLPC1114は内蔵発振器で動作していますので，互いの動作周波数は微妙に異なります．このため，アレイ接続ケーブルなしで2枚の付属基板(MB)を同時に動作開始させると，次第にLEDの点滅タイミングが異なってきます．互いに接続されていれば，タイミングは同期し続けます．ぜひ比較実験してみてください．

● 付属基板(MB)をアレイ接続するときのルール

付属基板(MB)を2枚以上アレイ接続する場合のルールがあります．これは，付属基板(MB)が複数接続されているときに，リセット後に互いの接続関係をチェックして，それぞれの基板に固有のID番号と接続位置を自動的に判断させるルーチンを実行するために必要なものです．

付属基板(MB)がアレイ接続された状態で，**図1**に示すようにNorthコネクタとWestコネクタが同時に



（a）付属基板（MB）アレイ接続コネクタの命名（方角に対応させている）

（b）NorthとWest側のコネクタが同時に開放（通信クローズ状態）になっている付属基板（MB）が2枚以上あってはいけない

図1　付属基板（MB）をアレイ接続するときのルール

付属基板（MB）上のアレイ接続コネクタを表から見たときの方角に対応させて（a）のように命名する．付属基板（MB）をアレイ接続するときは，（b）のように North コネクタと West コネクタが同時に開放（通信クローズ状態）になっているものが 2 枚以上あってはいけない（右下の例）

開放(未接続；通信ポートがクローズ状態)になっているものが必ず1枚だけになるようにしてください．

このルールで接続すれば，3枚以上のMBに「PROG02_COLOR_LED_MULTI」を書き込んでアレイ接続すれば，全基板でLEDの点滅が同期します．なお，ルールに従わずアレイ接続しても，ハードウェア的に壊れる心配はありません．

アレイ接続の最大枚数は，アルゴリズム上は65536×65536です．しかし，アレイ内側の基板への電源供給能力から考えると，現実的には16×16程度と思ったほうがいいでしょう．

プログラムの内容

● メイン・ルーチンの構造

メイン・ルーチン`main()`内のフローを図2に示します．各APIの初期化後，無限ループ内でカラーLED表示しながら，一定時間ごとにSouth側とEast側に向けて`SysTick`値を送り，相手側の中で`SysTick`値を同期(補正)させています．

以下，メイン・ルーチンで使用している各APIについて説明します．

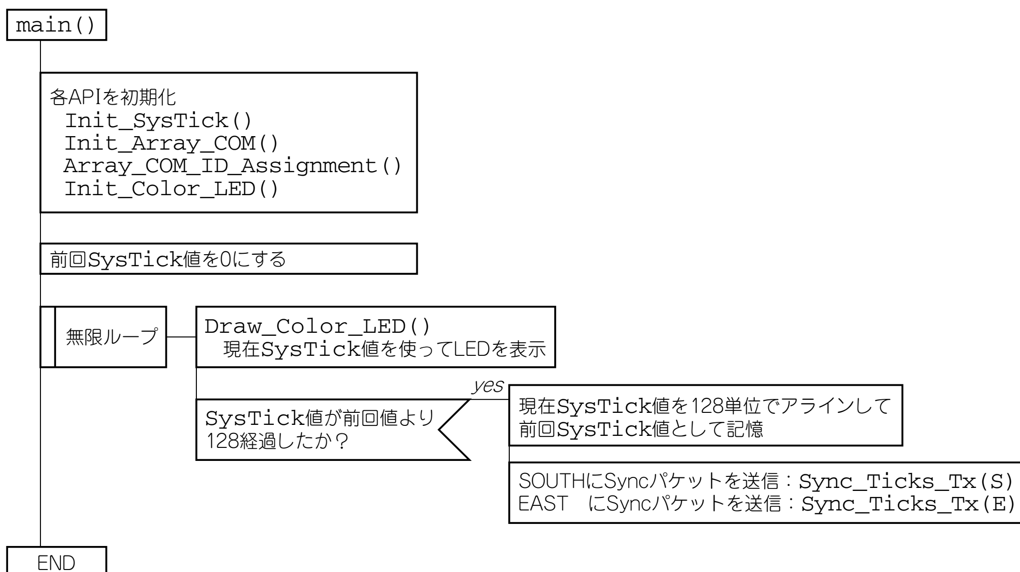


図2　メイン・ルーチンのフロー

表2　array_com.hで使用するリソース

項　目	使用リソース	備　考
周辺機能	PIO3_5	NORTH
	PIO0_3	SOUTH
	PIO3_2	WEST
	PIO3_4	EAST
	TIMER16B1	−
割り込みハンドラ	`PIOINT0_IRQHandler()`	−
	`PIOINT3_IRQHandler()`	−
	`TIMER_16_1_IRQHandler()`	−
	`ARRAY_COM_Rx_N_IRQHandler()`	例外No.13(リザーブ)をソフト起動
	`ARRAY_COM_Rx_S_IRQHandler()`	例外No.14(リザーブ)をソフト起動
	`ARRAY_COM_Rx_W_IRQHandler()`	例外No.22(リザーブ)をソフト起動
	`ARRAY_COM_Rx_E_IRQHandler()`	例外No.23(リザーブ)をソフト起動

● 付属基板間アレイ通信のAPI

付属基板間アレイ通信のAPIを**表1**に，使用するリソースを**表2**に示します．付属基板(MB)間通信は，1-wireシリアル通信をソフトウェアで実現しており制御が複雑ですが，このAPIを使用すれば簡単に実現できます．

初期化ルーチンのInit_Array_COM()は，最初に必ず実行してください．その後，アレイ接続関係の認識をさせるため，Array_COM_ID_Assignment()も必ず実行してください．これにより，東西南北の各アレイ接続ポートに相手ノードが接続されているかどうかや，自分の接続位置，システム全体の固有ID番号，接続されている総ノード数を得ることができます．

付属基板(MB)間のデータの送受信は，データ・パ

表1　付属基板(MB)間のアレイ通信制御のためのAPI

ヘッダ・ファイル	#include "array_com.h"	
ソース	関数名	意　味
初期化		
array_com.c	void Init_Array_COM(void)	アレイ通信用ポートの設定，タイマ(TIMER16B1)の初期設定，割り込み設定など，MB間アレイ通信制御のための初期化
アレイ接続関係の認識		
array_com.c	void Array_COM_ID_Assignment (void)	アレイ間の接続関係を認識する．ポートの開閉状態のチェック，IDアサイン，総ノード数の認識を行う
array_com.c	uint32_t Array_COM_Port_Open (uint32_t port)	ポートportが開いていれば1を，閉じていれば0を返す
array_com.c	uint32_t Array_COM_Get_ID_PosX(void)	IDのposxを返す．アレイ接続状態の自分の位置の*X*座標
array_com.c	uint32_t Array_COM_Get_ID_PosY(void)	IDのposyを返す．アレイ接続状態の自分の位置の*Y*座標
array_com.c	uint32_t Array_COM_Get_ID(void)	IDのid番号を返す．アレイ接続内の固有番号
array_com.c	uint32_t Array_COM_Total_Nodes(void)	アレイ接続されている総ノード数を返す
データ・パケット送受信		
array_com.c	void Array_COM_Tx_Data32 (uint32_t port, uint32_t data32)	32ビット・データdata32をポートportから送信する．ポート名は以下で定義されている enum Array_COM_PORT {N = 0, S, W, E}
array_com.c	uint32_t Array_COM_Rx_Data32 (uint32_t port, uint32_t *data32, uint32_t timeout)	32ビット・データdata32をポートportから受信する．timeoutまでに受信できたら1を，受信できなかったら0を返す．timeoutはCortex-M0のSysTick数で指定する
array_com.c	void Array_COM_Tx_Multi_Data32 (uint32_t port, uint32_t *pMultiData, uint32_t count)	count個の32ビット・データをポートportから送信する．32ビット・データはポインタpMultiDataから格納しておく
array_com.c	uint32_t Array_COM_Rx_Multi_Data32 (uint32_t port, uint32_t *pMultiData, uint32_t timeout)	Array_COM_Tx_Multi_Data32()で送信された32ビット・データをポートportから受信する．32ビット・データはポインタpMultiDataから格納される．timeoutまでに受信できたら受信データ個数を，受信できなかったら0を返す．timeoutはCortex-M0のSysTick数で指定する
array_com.c	void Array_COM_Tx_Multi_Bytes (uint32_t port, uint8_t *pMultiData, uint32_t count)	count個の8ビット・データをポートportから送信する．8ビット・データはポインタpMultiDataから格納しておく
array_com.c	uint32_t Array_COM_Rx_Multi_Bytes (uint32_t port, uint8_t *pMultiData, uint32_t timeout)	Array_COM_Tx_Multi_Bytes()で送信された8ビット・データをポートportから受信する．8ビット・データはポインタpMultiDataから格納される．timeoutまでに受信できたら受信データ個数を，受信できなかったら0を返す．timeoutはCortex-M0のSysTick数で指定する
SysTick同期化		
array_com.c	void Sync_Ticks_Tx(uint32_t port)	SysTick値をポートportからTick値同期用パケットとして送信する．受信した側は自動的に自分のTick値を受信値に更新する

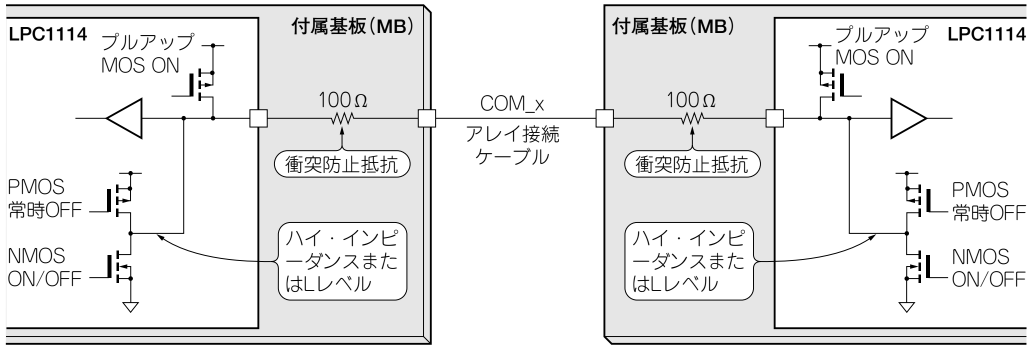


図3　アレイ間の1-wireシリアル通信回路
アレイ間はオープン・ドレイン型のドライブ方式による1-wire非同期シリアル通信で接続している

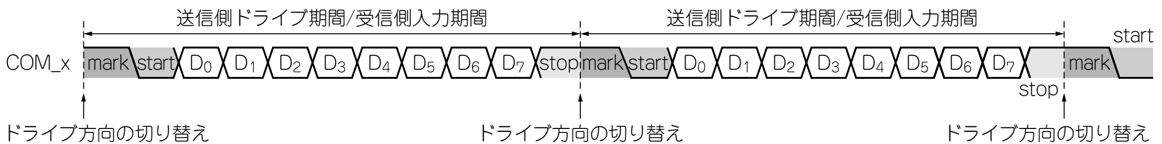


図4　アレイ間の1-wireシリアル通信波形
送受信切り替え時にマーク期間を確保して，受信側がスタート・ビットの立ち下がりを確実に検知できるようにしている

ケット送受信関数を使います．また，相手側の`SysTick`値を自分の`SysTick`値で同期させるには，`Sync_Ticks_Tx()`を実行します．

● アレイ通信の基本方式

以下では，前記のアレイ通信APIがどのように実装されているかを説明します．

付属基板(MB)間のアレイ通信は，**図3**に示す回路による1-wire型の非同期シリアル通信で行います．送信も受信も同じ1本のワイヤを通して行うので，同時送受信はできません(半二重通信)．

双方のノードが同時に送信したときに衝突するのを防ぐため，出力する側のノードはLレベルかハイ・インピーダンスだけを出力します(オープン・ドレイン型)．LPC1114の内蔵プルアップMOSで，ハイ・インピーダンス時にHレベルになります．

1-wire通信時の波形を**図4**に示します．スタート・ビット(start)で始まり，8ビット・データ(D_0～D_7)が続き，最後がストップ・ビット(stop)で終わる通常の非同期式シリアル通信方式です．

ただし，スタート・ビットの前にマーク・ビット(mark；レベルは1)を入れて，送受信のドライブ方向が切り替わる前後に必ず‘1’の期間を確保するようにしています．送受信方向の切り替え時に，すぐにスタート・ビット(レベルは0)から開始すると，受信側が正しくスタート・ビットの立ち下がりを検知できない場合があるので，マーク期間を挿入しました．

● 1-wireシリアル通信フロー

1-wireシリアル通信は，4方向(東西南北)に向けて行うので，非同期シリアル通信機能を4チャネル必要とします．しかし，LPC1114にはUARTが1チャネルしかないので，すべてソフトウェアで実現しています．本システムでは，ボーレートとして10000 bpsを実現しています．

1バイト送信時には，

マーク→スタート→8ビット・データ→ストップ

までの各ビット出力間隔を，タイマによる割り込みで実現しています．

図5に送信時のプログラム，**図6**に受信時のプログラムの状態遷移図を示します．

1バイト受信時には，スタート・ビット(ポート)の立ち下がり検知割り込みで状態遷移を起動し，**図6**に示すようにデータ取得間隔をタイマ割り込みで実現しています．1バイト受信完了時点で，ソフトウェアで別の割り込みを起動し，パケット受信処理をさせています．

送受信パケットの構造を**図7**に示します．

パケットは，Tick値同期用と一般データ用の2種類があります．先頭の1バイトでその種類を示し，2番目のバイトがデータ本体のバイト数を示します．その後に本体データが並びます．パケット受信処理の中で，Tick値同期用パケットを受信したノードは自動的にTick値を同期するようにしてあります．

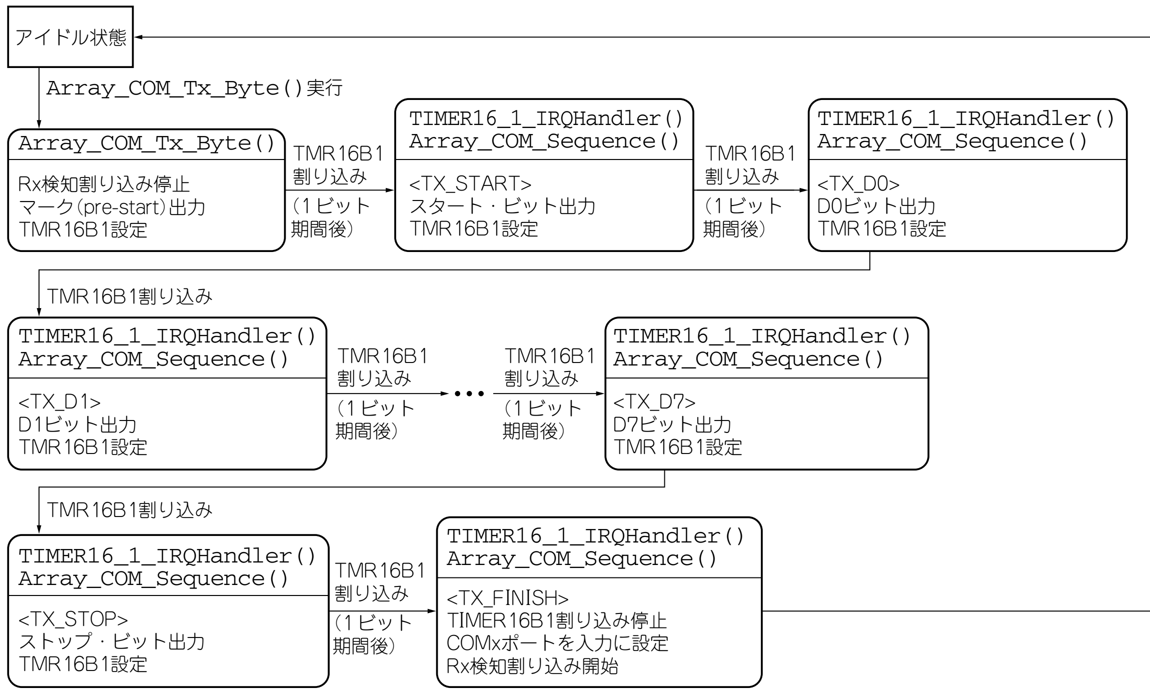
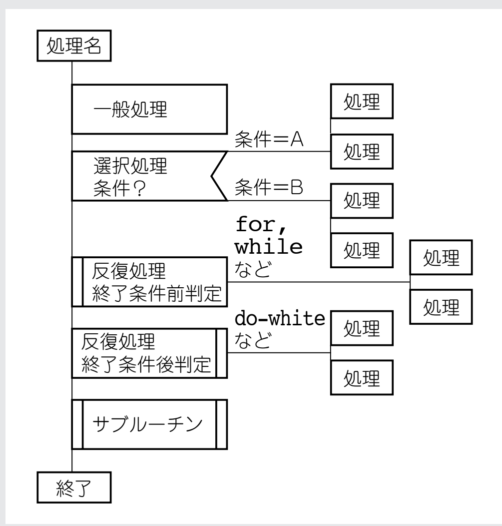


図5　1-wireシリアル送信の状態遷移図
1バイト送信ルーチンが，シリアル送信の状態遷移を起動する．各ビット出力タイミングをタイマによる割り込みで実現している

Column　PAD表記について

本書では，アルゴリズムの図示表現にフローチャートではなく，PAD(Program Analysis Diagram)を使用します．

図Aに示す記号を使うもので，フローチャートよりもコンパクトで見通しのよいアルゴリズム表記が可能です．



図A　PAD表記に使用する記号

アイドル状態

COMxポートのスタート・ビット
立ち下がり検知割り込み

PIOINTn_IRQHandler()
Rx検知割り込み停止
TMR16B1設定(半ビット期間)

TMR16B1
割り込み
(半ビット期
間後)

スタート・ビットで
なかった場合

TIMER16_1_IRQHandler()
Array_COM_Sequence()
＜RX_START＞
スタート・ビット確認
TMR16B1設定

TMR16B1
割り込み
(1ビット
期間後)

TIMER16_1_IRQHandler()
Array_COM_Sequence()
＜RX_D0＞
D0ビット取り込み
TMR16B1設定

TMR16B1割り込み(1ビット期間後)

TIMER16_1_IRQHandler()
Array_COM_Sequence()
＜RX_D1＞
D1ビット取り込み
TMR16B1設定

TMR16B1
割り込み
(1ビット
期間後)

…

TMR16B1
割り込み
(1ビット
期間後)

TIMER16_1_IRQHandler()
Array_COM_Sequence()
＜RX_D7＞
D7ビット取り込み
TMR16B1設定

TMR16B1割り込み(1ビット期間後)

TIMER16_1_IRQHandler()
Array_COM_Sequence()
＜RX_STOP＞
ストップ・ビット確認
TMR16B1設定(半ビット期間)

TMR16B1
割り込み
(半ビット
期間後)

TIMER16_1_IRQHandler()
Array_COM_Sequence()
＜RX_FINISH＞
TIMER16B1割り込み停止
Array_COM_Rx_x_IRQをソフト発生
Rx検知割り込み開始

Array_COM_Rx_x_IRQ割り込み発生

Array_COM_Rx_x_IRQHandler()
Array_COM_Rx_Packet()
受信パケット処理：1バイトずつパケットを受け取る
パケット全体を受信したら
・Tick値同期パケットの場合は自分のSysTick値を補正
・一般データ・パケットの場合は受信フラグgCOMPacketRxFull[]をセット

図6　1-wire シリアル受信の状態遷移図

スタート・ビットの立ち下がり検知割り込みが，シリアル受信の状態遷移を起動する．各ビット入力タイミングをタイマ割り込みで実現している．スタート・ビットの立ち下がり後，半ビット期間待ってから改めてスタート・ビットの中央で0レベルを確認する．その後，1ビット期間間隔で各ビットの中央でデータを取得し，ストップ・レベル(1レベル)を確認したあと，半ビット期間待ってから受信パケット処理を起動するための割り込みをソフトウェアで発生させて終了している

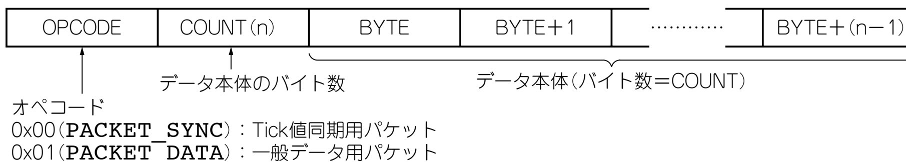


図7　1-wire シリアル送受信パケットの構造

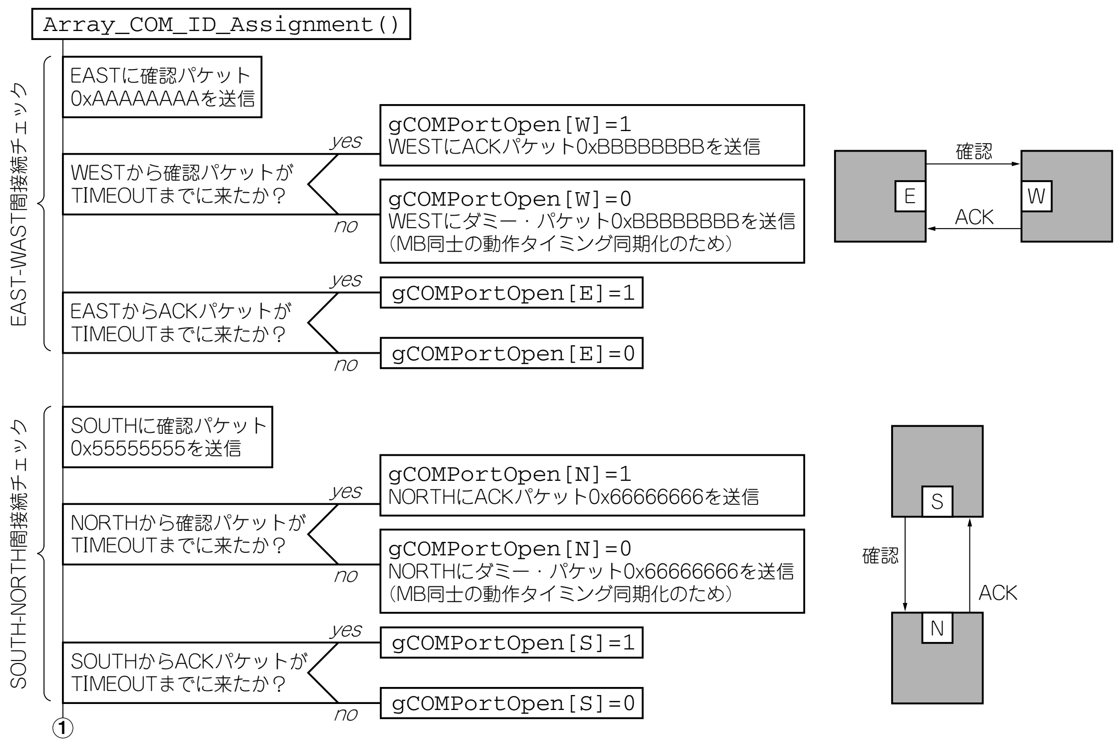


図8　アレイ接続時のポート間接続の確認フロー

● アレイ接続関係のチェック方法

アレイ接続関係を自動チェックするAPI関数 `Array_COM_ID_Assignment()`の内部構造を説明します．

最初に，4方向あるポートが，それぞれ相手側と接続されているかどうかをチェックします．そのフローを**図8**に示します．このチェック結果は，`Array_COM_Port_Open()`で知ることができます．

次に，アレイ接続されたシステム全体の中でのノード(MB)の位置を(x，y)座標として割り当てます．この結果は，`Array_COM_Get_ID_PosX()`，`Array_COM_Get_ID_PosY()`で知ることができます．

また，固有のID番号をそれぞれのノード(MB)に与えます．この結果は，`Array_COM_Get_ID()`で知ることができます．接続位置とIDの割り当てフローを**図9**に示します．

最後に，アレイ接続されている総ノード数を各ノードに知らせます．そのフローを**図10**に示します．この結果は，`Array_COM_Total_Nodes()`で知ることができます．

Column　筆者の独り言③…裏事情

マルチコアの発想に至ったのが2010年の4月．それ以降，ホイホイとアイディアが出て，この「MARYシステム」が誕生しました．一つのマイコンに小さい単位の仕事をやらせ，複数のマイコン同士を緩い通信チャネルで結び，互いに協調/同期動作を行わせることで，全体として大きなシステムを組むことができるものです．

筆者はかつてマルチコア・システムを設計したことがあります．

一つはデータ・フロー型コンピュータで，いわゆる非ノイマン型マシンです．そこに浮動小数点演算器をもつPE(Processing Element)を複数並

➡

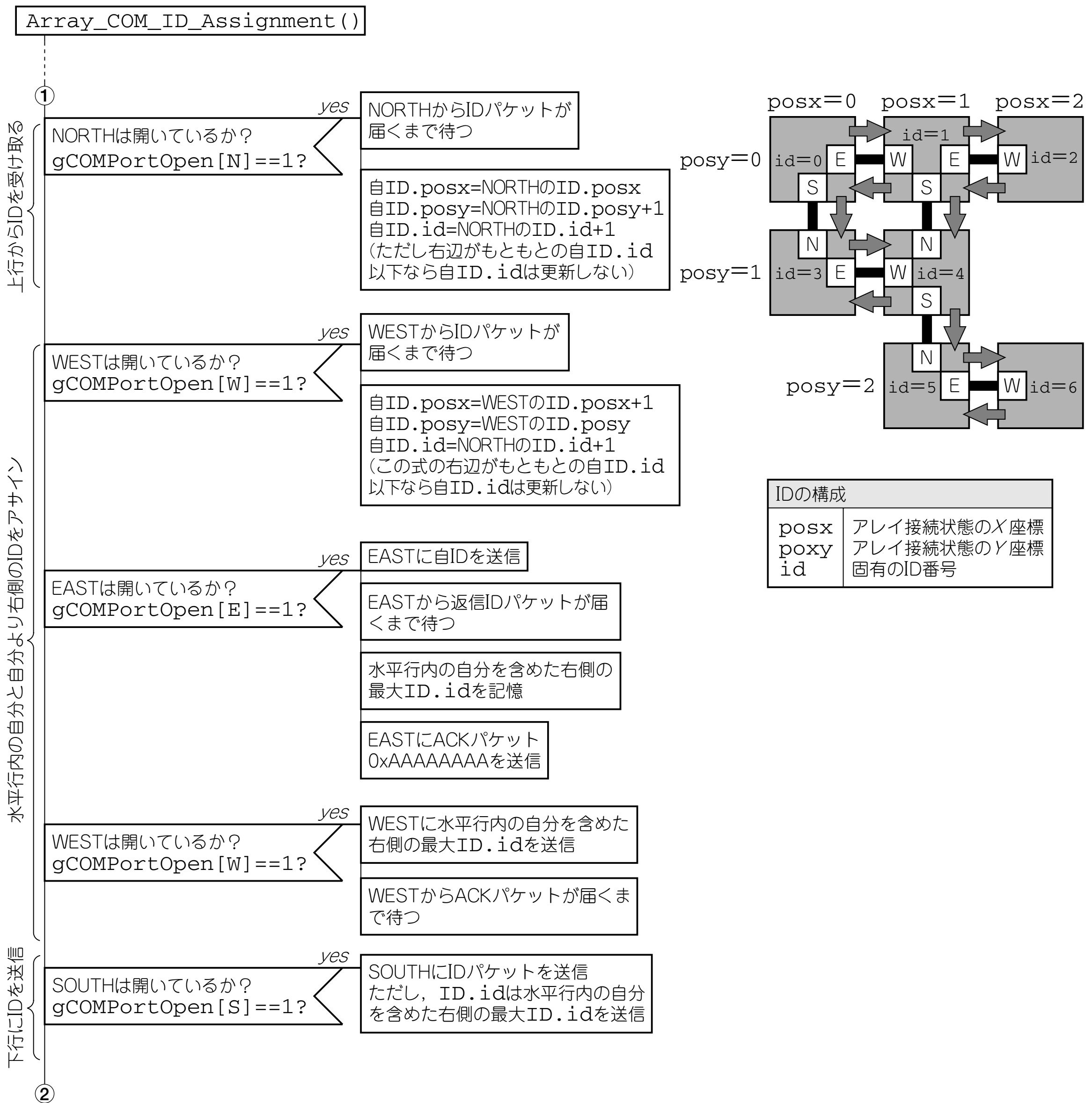


図9　アレイ接続時のIDアサイン・フロー
図8の続きのフロー

べ，計算フロー全体の中で演算できる処理から自動的に空いているPEに処理を割り当てて性能を向上させるものです．

もう一つは汎用プロセサと小型専用RISCコア16個を集積したヘテロジニアス・マルチコアで，動画コーデック処理を高速化するものでした．それらはそれらで非常におもしろい試みだったのですが，その都度，心の中にはもう一つの思いがありました．

ある世代以上の方は，1980年代に世に出た英国INMOS社の「トランスピュータ（Transputer）」という製品を聞いたことがあるかもしれません．一つの小規模なプロセッサに四つの高速通信チャネルをもたせ，そのプロセッサを複数並べて互いに通信させながらプロセスを並列処理して高性能化するものでした．

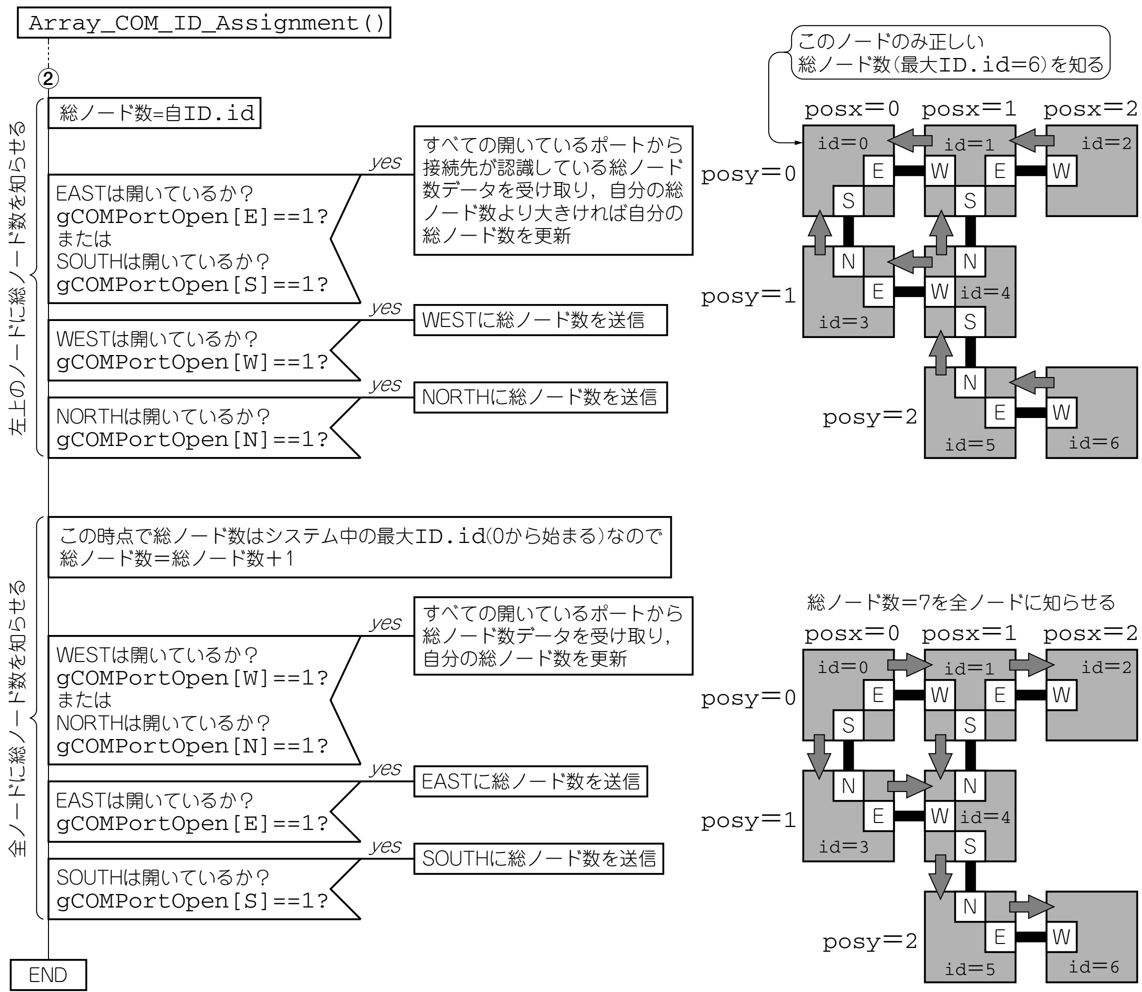


図10　アレイ接続時の総ノード数の検出フロー

図9の続きのフロー

通信チャネルは当時としてはかなり高速なMbps単位のレートを実現でき，並列処理記述言語「オッカム(Occum)」でプログラムできる先駆的でエレガントなものでした．

トランスピュータは商業的には成功しませんでしたが，筆者は常にこのトランスピュータのような美しいアーキテクチャに憧れてきました．少しオーバーな表現かもしれませんが，そのイメージが「MARYシステム」として形作られたように思います．

「MARYシステム」は4方向通信チャネルの速度や並列処理へのインフラなどはトランスピュータにかないませんが，CPUコア自体ははるかに強力であり，コア間通信方式はとても直感的でシンプルなので，多くの方に使いこなしていただけると思っています．

第13章 有機ELディスプレイ基板(OB)上の表示モジュールのコントロール

―― 有機ELディスプレイにビットマップ画像を表示する

本章では，有機ELディスプレイ基板(OB)と付属基板(MB)を使って，OLED(有機ELディスプレイ)へビットマップ画像などを表示する方法について説明します．

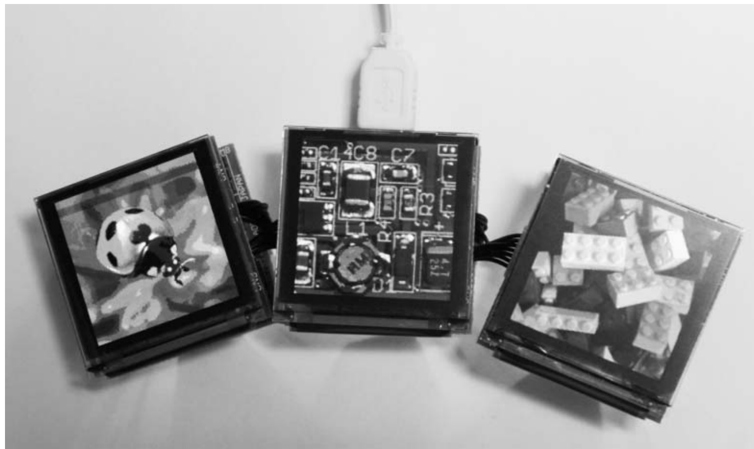


写真1　OLEDモジュールにビットマップ画像を表示

【用意するもの】
付属基板(MB)：1枚(CN_1 ～ CN_4を取り付け済みのもの)
有機ELディスプレイ基板(OB)(別売)：1枚
使用するプロジェクト：PROG03_BMP

● プロジェクトPROG03_BMPの実験

向きに注意して付属基板(MB)の上に有機ELディスプレイ基板(OB)を取り付けてください．

PROG03_BMPをビルドして，完成したバイナリを付属基板(MB)にダウンロードしてください．

写真1のようにビットマップ画像が表示されます．ビットマップ画像を変える場合は，`main()`内の**リスト1**に示す`#define`を変更して，再ビルドしてダウンロードしてください．

プログラムの内容

● メイン・ルーチンの構造

メイン・ルーチンでは，ビットマップ画像データをインクルードしてROM内に置き，`main()`内で，その画像データの各ピクセルを順次OLEDに描画しているだけです．

ただし，ROM内に置いているビットマップ画像はROMサイズの制限から8ビット・ビットマップにしました．よって，カラー・パレットも一緒にデータとしてROMに格納しています．

ビットマップ画像データの各8ビット・ピクセル値

でカラー・パレットを引いて32ビットのピクセル値を求め，それをOLED用の[RGB：565]フォーマットに変換してOLEDを描画しています．

以下，メイン・ルーチンで使用している各APIについて説明します．

● OLED表示関係のAPI

OLED表示関係のAPIを**表1**に，使用するリソースを**表2**に示します．このうち文字列表示用の`OELD_printf()`は本システム用に専用に作成したコンパクトなもので，ROMサイズをあまり消費しません．この関数は次章のアプリケーションで使用します．

LPC1114とOLEDモジュールの間はSPI通信でインターフェースしますが，これらのAPI関数を使うかぎりはその詳細を意識する必要はありません．

リスト1　ビットマップ画像を変更する場所

```
#define BMP 0   // select which BMP you want to see.
```

0で基板，1でレゴ，2でてんとう虫の画像が出る

ビットマップ・データの作成方法

ROMに格納するビットマップ・データの作成方法を説明します．

表2　oled.hの使用リソース

項　目	使用リソース	備　考
周辺機能	PIO1_4	OLED_VCC_ON
	PIO1_8	OLED_$\overline{\text{RES}}$
	PIO0_2	OLED_$\overline{\text{SCS}}$
	PIO0_6	OLED_SCLK
	PIO0_9	OLED_SDIN
	PIO0_8	(MISO)
	SPI	－
割り込みハンドラ	`SSP_IRQHandler()`	同期シリアル・ポート割り込み

表1　OLED表示を制御するAPI

ヘッダ・ファイル	`#include "oled.h"`	
ソース	関数名	意　味
初期化		
oled.c	`void Init_OLED(void)`	OLEDを制御するポート，SPIを初期化し，OLEDモジュールに初期化コマンドを送る
OLED描画		
oled.c	`void OLED_Draw_Dot(int32_t x, int32_t y,int32_t size, uint32_t color)`	OLED画面上にドットを描画する．座標(x, y)に，`size`ピクセル角サイズのドット(正方形)をピクセル色`color`(下位16ビット有効)で描画する
oled.c	`void OLED_Fill_Rect(int32_t x0, int32_t y0, int32_t xsize, int32_t ysize, uint32_t color)`	OLED画面上に矩形を描画する．座標(x0, y0)を原点(左上)とする幅`xsize`，高さ`ysize`の矩形をピクセル色`color`(下位16ビット有効)で埋めて描画する
oled.c	`void OLED_Clear_Screen (uint32_t color)`	OLED画面をクリアする．埋めるピクセル色は`color`(下位16ビット有効)で指定する
文字表示		
oled.c	`void OLED_printf (const char *format, ...)`	OLED画面上にメッセージを出力する．使いかたは通常の`printf()`と同様．ただし，サポートしているフォーマットは以下のとおり(浮動小数点は未サポート)．各数値は桁指定(`%04d`)も可能である． `%c`：文字，`%s`：文字列， `%d`：符号付き整数，`%u`：符号なし整数， `%x`：16進数，`%b`：2進数

oled.c	`void OLED_printf_Font (uint32_t font)`	`OLED_printf()`で使用するフォント・サイズを指定する．下記の3種類から選択する
		`OLED_FONT_SMALL`(デフォルト)
		`OLED_FONT_MEDIUM`
		`OLED_FONT_LARGE`
		`OLED_printf()`の前に実行すること

oled.c	`void OLED_printf_Color (uint32_t color_f, uint8_t color_b)`	`OLED_printf()`で使用する描画カラーを指定する．文字色を`color_f`で指定し，背景色を`color_b`で指定する．`OLED_printf()`の前に実行すること
oled.c	`void OLED_printf_Position (uint32_t posx,uint32_t posy)`	`OLED_printf()`で文字列表示する開始位置の桁`posx`と行`posy`を指定する．指定した桁と行は`OLED_printf_Font`で指定したフォント・サイズに対応している

リスト3　ビットマップ・データの部分切り出しコマンド
オプション－oで生成する画像本体データのファイル名を指定，－pでパレット・データのファイル名を指定できる．－x，－yオプションで切り出し部分の左上の座標を指定し，－w，－hで切り出し部分の幅と高さを指定する

```
$ ./bmp2c.pl bud.bmp -o bud00.rgb.h -p bud.plt.h -x   0 -y   0 -w 128 -h 128
$ ./bmp2c.pl bud.bmp -o bud01.rgb.h -p bud.plt.h -x 160 -y   0 -w 128 -h 128
$ ./bmp2c.pl bud.bmp -o bud02.rgb.h -p bud.plt.h -x 320 -y   0 -w 128 -h 128
$ ./bmp2c.pl bud.bmp -o bud03.rgb.h -p bud.plt.h -x 480 -y   0 -w 128 -h 128

$ ./bmp2c.pl bud.bmp -o bud10.rgb.h -p bud.plt.h -x   0 -y 160 -w 128 -h 128
$ ./bmp2c.pl bud.bmp -o bud11.rgb.h -p bud.plt.h -x 160 -y 160 -w 128 -h 128
$ ./bmp2c.pl bud.bmp -o bud12.rgb.h -p bud.plt.h -x 320 -y 160 -w 128 -h 128
$ ./bmp2c.pl bud.bmp -o bud13.rgb.h -p bud.plt.h -x 480 -y 160 -w 128 -h 128

$ ./bmp2c.pl bud.bmp -o bud20.rgb.h -p bud.plt.h -x   0 -y 320 -w 128 -h 128
$ ./bmp2c.pl bud.bmp -o bud21.rgb.h -p bud.plt.h -x 160 -y 320 -w 128 -h 128
$ ./bmp2c.pl bud.bmp -o bud22.rgb.h -p bud.plt.h -x 320 -y 320 -w 128 -h 128
$ ./bmp2c.pl bud.bmp -o bud23.rgb.h -p bud.plt.h -x 480 -y 320 -w 128 -h 128

$ ./bmp2c.pl bud.bmp -o bud30.rgb.h -p bud.plt.h -x   0 -y 480 -w 128 -h 128
$ ./bmp2c.pl bud.bmp -o bud31.rgb.h -p bud.plt.h -x 160 -y 480 -w 128 -h 128
$ ./bmp2c.pl bud.bmp -o bud32.rgb.h -p bud.plt.h -x 320 -y 480 -w 128 -h 128
$ ./bmp2c.pl bud.bmp -o bud33.rgb.h -p bud.plt.h -x 480 -y 480 -w 128 -h 128
```

リスト2　ビットマップ・データの作成コマンド
画像本体データboard8.bmp.rgb.hとパレット・データboard8.bmp.plt.hが生成される

```
$ ./bmp2c.pl board8.bmp
```

● perlを利用する

OLEDモジュールの画面サイズが128×128ピクセルなので，元画像データとしては128×128サイズの8ビット・ビットマップ画像を用意してください．これをプロジェクト「PROG03_BMP」フォルダの下の「bmp」フォルダに格納します．

例えば，あらかじめここに格納してある“board8.bmp”という画像からデータを作成する場合の手順を説明します．

まず，perlを実行できる環境を用意してください．最も手軽には，Windows上にcygwinをインストールするのが簡単です．perlパッケージがインストールされるようにしてください．

次に，上記のbmpフォルダにカレント・ディレクトリを移動して，**リスト2**のように“bmp2c.pl”コマンドを実行してください．このコマンドの中身は，まるごとperlプログラムです．これにより，画像本体データ“board8.bmp.rgb.h”とパレット・データ“board8.bmp.plt.h”が生成されます．これを，そのままメイン・ルーチンにインクルードして“PROG03_BMP”のように使ってください．

● 大きいビットマップ画像の部分切り出し表示

元画像データが128×128より大きい場合に，その一部を128×128に切り出す方法を説明します．

例えば，608×608ピクセルの“bud.bmp”という8ビット・ビットマップ画像を例に説明します．これも，bmpディレクトリ内に格納してあります．これを**図1**のように16種の128×128サイズに切り出して，OLEDモジュールに表示させてみるとします．

前記の“bmp2c.pl”に切り出し機能がありますので，**リスト3**のように実行すれば，16個の画像本体データ“budxx.rgb.h”とパレット・データ“bud.plt.h”が生成されます．元画像が1枚なのでパレット・データは16個に共通になるので，毎回同じファイル名“bud.plt.h”で生成させています．これらをメイン・ルーチンにインクルードしてビルドしてください(main.c内では`#define`の変更で済むようになっている)．

生成した16組のデータを，それぞれ16個の有機ELディスプレイ基板(OB)と付属基板(MB)にダウンロードして表示させて並べてみたのが**写真2**です．

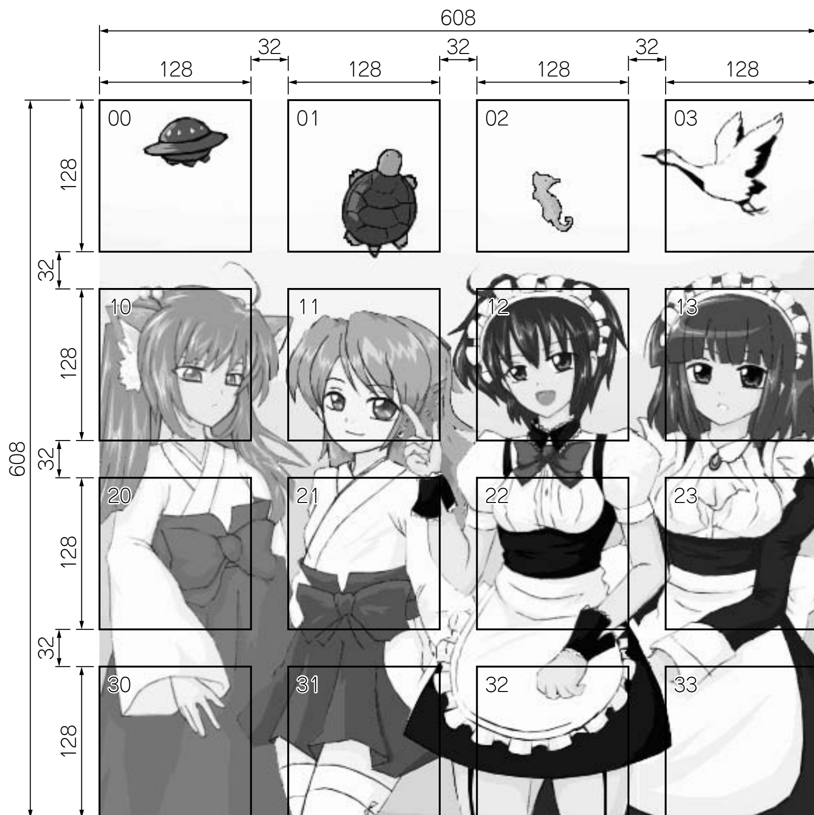


図1　大きいビットマップ画像の部分切り出し

608×608 のサイズのビットマップ画像から 128×128 の小片 16 個を取り出す（作画：©2011 榊）

写真2　16 組の有機 EL ディスプレイ基板（OB）/付属基板（MB）で大きいビットマップ・データを表示

大人買いによるマルチ・ディスプレイもどきである．USB ケーブル 1 本で 1 カ所の付属基板（MB）から給電させるのは最大で 4 ～ 8 組程度の有機 EL ディスプレイ基板（OB）/付属基板（MB）にするのが安心．ここでは 3 カ所から給電している（USB による給電×2 カ所，AC アダプタ接続基板 PB による給電×1 カ所）

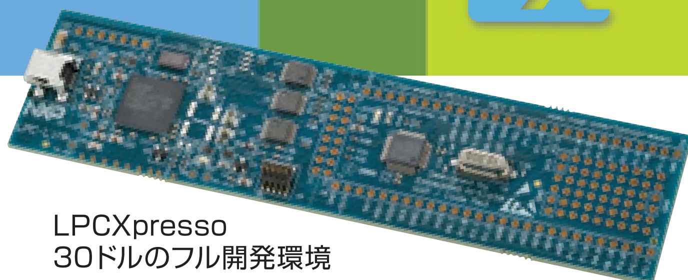
ARM Cortex-Mシリーズ

といえば...

www.nxp-lpc.com

日本語サイトオープン

LPC X PRESSO



LPCXpresso
30ドルのフル開発環境

NXP

NXPセミコンダクターズジャパン株式会社
www.jp.nxp.com

トランジスタ技術 増刊 2枚入り! 組み合わせ自在! 超小型ARMマイコン基板

CQ出版社 〒170-8461 東京都豊島区巣鴨1-14-2 TEL.(03)5395-2141(販売部)

定価3,000円 本体2,857円

雑誌 06664-4
Ⓛ-2011.5/18

4910066640415
02857