

妥協点を見つめる

高速8ビットマイクロコントローラを使ったアプリケーションの開発

パソコンのソフトウェアの開発者には、組込システムの開発者よりも有利な点がいろいろとあります。ほんの数年前のスーパーコンピュータに匹敵する処理能力とメモリ量を持つシステムのための開発だけでなく、ほとんどの場合既にあるシステム向けの開発も行います。これに対し、組込システムの開発者はより小さいシステムの開発だけでなく、通常はまず最初にシステム自体を設計しなければなりません。

組込システムにするかどうかは、対応する課題に応じて決めます。ユーザとのやり取りがほとんどなく、制御デバイスが少なく、比較的シンプルな場合には、8051や68HC11、Atmel[®]、AVR[®]、PICファミリなど、低電力の8ビットマイクロコントローラで解決できます。これで、処理能力も柔軟性も十分であることが多いのです。これに対し、ユーザとのやり取りが多い場合、イーサネットを使ったやり取りが必要な場合、あるいはデジタルカメラなどの複雑なデバイスとのやり取りが必要な場合には、PC-104やStrongARMなど、「ワンカローパソコン」と呼ばれるものを使います。一般的に高い処理能力と複雑なOS、大量のRAMが使えるからです。

ところが、この2つのアプローチの間にグレーゾーンがあるのです。ジョーが経営するジョーセキュリティサービスという会社が、競争の激しいネットワークドアロック市場に参入するというケースを考えてみましょう。セキュリティ問題に対する関心が高まった結果、ID番号以外の情報を要求するドアロックシステムが求められています。ドアを開けようとするときにその人物の顔写真や指紋を要求する電子ドアロックです。取得した画像はネットワーク経由でセントラルサーバに送られ、そこに記録したり、あるいは画像認識による認証処理などの複雑な処理をします。画像による認証が成功すれば、サーバからネットワークドアロックに応答を送りドアが開きます。ジョーとしては、顧客がなるべく多くのドアロックを設置して欲しいので、製品コストをなるべく低く抑えることが重要です。

ジョーの会社と競合するアレックスのセキュリティセントラル社は、8ビットRISCプロセッサとイーサネットコントローラを使ったネットワークカメラドアロックを開発しています。ジョーはこれでは処理能力が不十分だと考えています。さまざまなプロジェクトで、ハーバードアーキテクチャ型チップとイーサネットコントローラを組み合わせたシステムの開発が進められてきたことは知っ

ています。しかし、いずれも開発初期の段階に留まっており、市場に出回ってはいませんし、このアーキテクチャではTCP/IPスタックに制限があるという問題もあります。ネットワーク経由の通信には問題ないかもしれませんが、デジタルカメラとの通信には大きな問題があります。画像の解像度を十分なレベルとするためには40kBから60kBものメモリが必要で、このメモリはコード用メモリと競合することになります。ハーバードアーキテクチャ以外のプロセッサを使っていると、従来の8ビットマイクロコントローラにとっては処理もデータも多すぎるということになります。

ジョーの宿敵トロイも、トロイエクスティームセキュリティ社で同様のソリューションを開発中です。ジョーがトロイを嫌っている理由は、トロイが組込システムの技術、技巧を尊重していないからです。話を面白くするために、トロイはジョーの元彼女のアミガと付き合っているとします。アミガはジョーがコンピュータにばかり時間を割いている(組込システムの開発者の間では「職業病」と言われています)と言ってジョーの元を去ったのです。トロイのソリューションでは、StrongARMでPocket PCを走らせ、その高速I/Oとネットワーク機能を利用します。ジョーから見ると、これはあまりに過剰な能力を投入したソリューションです。撮影以外の大半の時間は、プロセッサは待機しているだけです。理想的なソリューションは大量のメモリや強力な処理能力など不要であり、組込用LinuxやPocket PCを走らせることは、そのような単純なシステムには無駄な資源投入であり、コストがかかり過ぎます。

ジョーが必要としているのは、ネットワークとカメラを取り扱えるだけの処理能力を持つが、32ビットソリューションよりも安価で機能もシンプルなマイクロコントローラです。アセンブリ言語だけでなく、より高度な言語が利用できれば、開発がシンプルになる点で好ましいと言えます。さて、どうすれば、ジョーはアレックスの価格とトロイの処理能力に打ち勝ち、恋人を取り戻すことができるのでしょうか？

TINI[®]を使うのです。

ネットワークブリッジ

TINI (tiny Internet interfaces)は、ガラスセミコンダクタの製品です。TINIプラットフォームは、ネットワークブリッジとして動作するように設計されています。TCP/IP経由でPCはTINIとやり取りすることが可能で、TINIはセンサや従来型のハードウェアなど各種電子機器とやり取りすることができます。TINIには1-Wire[®]、2線、RS-232シリアル、CAN、SPI[™]など、さまざまな外部インタフェースが用意されています。TINIは頑強なネットワークを実現し、IPv4、IPv6、DNS、DHCP、PPP、Telnet、FTPをサポートしています。

AtmelとAVRは、Atmel Corp.の登録商標です。
TINIと1-Wireは、Dallas Semiconductorの登録商標です。
SPIはMotorola, Inc.の商標です。

TINIには2種類のリファレンスデザインがあります。よく使われるのはDS80C390を使ったTINIm390検証モジュールで、72ピンのSIMMに、512kBのフラッシュメモリ、バッテリーバックアップされた512kBまたは1MBのSRAM、イーサネットコントローラ、及びリアルタイムクロックを搭載しています。新しいTINIm400検証モジュールは、DS80C400をベースにした製品で、144ピンのSO DIMMにほぼ同等の機能を搭載しています。ただし、イーサネットMACがDS80C400に内蔵されている点が異なります。

DS80C390もDS80C400も、8ビットのマイクロコントローラです。実際、両製品ともコアに8051マイクロコントローラを使っています。ただし、その機能は大幅に拡張されています。まず、1命令当たりのサイクル数が12から4に高速化されています。これで、同一クロック周波数で標準8051の3倍の処理能力が得られます。また、アドレス能力も大幅に向上しています。DS80C390では4MBのプログラムメモリと4MBのデータメモリが利用可能で、DS80C400では16MBのフラットなアドレス空間が利用可能です。また、最大クロック周波数も大幅に引き上げられています。DS80C390は40MHzまで、DS80C400は75MHzまで動作します。さらに、乗算・除算用の整数演算アクセラレータも搭載しています。この結果、従来の8ビットマイクロコントローラと16ビット/32ビットマイクロコントローラの中間的な処理能力を持つマイクロコントローラに仕上がっています。

TINIプラットフォームが持つユニークな特長の1つに、ガラスセミコンダクタが開発したOSがあります。使用料が無料であるにも関わらず、マルチタスク・マルチスレッド対応で、Java™ランタイム環境が利用できます。入手は無償でダウンロードできます。コアOSとライブラリが512kBのフラッシュに収まるので、最後のフラッシュバンクにアプリケーション用のメモリを64kB確保できます。DS80C400では、C言語とアセンブリ言語用のROMライブラリもあります。

ネットワークカメラ

ネットワークカメラという課題に対するソリューション案として、ウェブカメラを使ったストリーミングをTINIによって実現し、そのベンチマークを測定してみましょう。今回はTINIm400リファレンスデザインではなく、高速DS80C400カスタムボードを使います(図1)。ここでのネットワークカメラは、生の画像を取得し、それをUDP経由でPCホストに送ります。PCとの通信は、ホスト側ソフトウェアで行うか、HTTP上のJavaアプレットで行います。

Javaは、Sun Microsystemsの商標です。
Intelは、Intel Corporationの登録商標です。
Philips社とのI²C特許権契約により、システムがPhilips社のI²C標準規格に合致していることを条件に、Maxim Integrated Products, Inc.または二次ライセンスを受けた関連会社が製造したI²C製品を購入することにより、システムでこれらの製品を使用するライセンスが譲渡されたこととなります。

カメラは、OmniVisionの5017 CMOSチップを搭載したM4088モジュールを使います。これはノンインターレース、白黒のデジタルカメラで、解像度は384 x 288ピクセルです。データラインが8本とアドレスラインが4本、それにチップセレクトがあり、簡単にメモリのマッピングが行えます。画像取得中にアサートされる垂直同期ライン、各スキャンライン中にアサートされる水平同期ライン、及びピクセル入力を知ることができるピクセルクロックがあります。一度カメラを初期化すると、ソフトウェアからアクセスするのは簡単です。5017はクロック分周器を内蔵しており、フレームレートをプログラムで制御することができます。シングルフレームモードも備えており、画像を集録するタイミングをホストデバイスから制御することができます。今回の設計は、プロセッサが非力で、カメラのトップスピードである50フレーム/秒もの処理はできないので、この機能はとてども便利です。

DS80C400バージョンは73MHz(18.4MHz x 4)で駆動するように設計されています。DS80C400はイーサネットMACを内蔵していますが、イーサネットPHYとマグネティックが必要です。PHYについては、この他にもHomePNAやHomePlug PHYなど、さまざまな種類をサポートしています。今回の例では、Intel® LXT972Aを標準MIIインタフェースでDS80C400に直接接続しました(図2)。このPHYには25MHzのクロックが必要です。

カメラには12nsのメモリが必要であり、十分なアクセス速度を持つ日立HM62W8511Hを使用しました。ブート時には、まずフラッシュ内のコードが実行され、実行コードがフラッシュからSRAMにコピーされるとともにクロックを4通倍するフラグがセットされ、TINIの開始アドレスへジャンプします。今回の例では、TINIm400とは異なり、バッテリーバックアップも不揮発化回路も装備していません。高速SRAMを使っており、バッテリーが急速に消耗してしまうためです。これはつまり、TINIOSに不揮発性ファイルシステムがないことを意味します。後述するように、これは大きな問題とはなりません。TINIがスタートアップ時にファイルシステムを構築するからです。

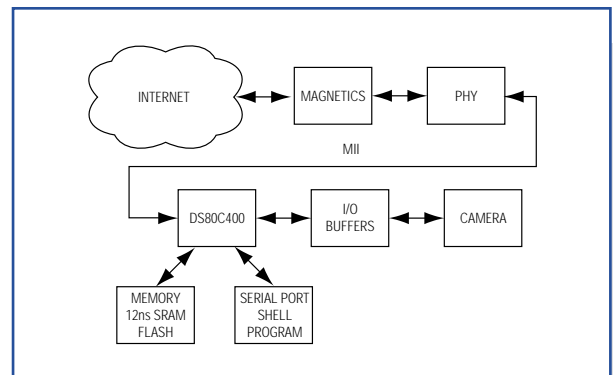


図1. 高速DS80C400を使用した設計の説明図

TINIOSがMACアドレスを保存するには、DS2502が必要です。また、必須ではありませんが、I²C™経由でDS1672リアルタイムクロックを接続しています。この結果、TINIからソフトウェア的にクロックへアクセスできるようになりますが、これ以外にも大きな利点があります。ブート時にリアルタイムクロックを検出すると、TINIOSは自動的にシステムバス速度を計算し、シリアルポートのボーレートやタイマの刻み、ネットワークタイミングなど、内部タイマの調整を行います。

カメラの接続は簡単です。AO-A3とDO-D8、WEBを適切なラインに接続するだけです。CE4はカメラのCSBラインにつなぎます。これにより、カメラは0 x 800000にマッピングされます。カメラのVSYNCラインをデータ読みとりのためにP1.1に接続し、PSENはOEBラインに接続します。HREFラインは反転させてからINT1に接続します。これでレベルトリガ割込が使えるようになります。

ソフトウェアインプリメンテーション

開発は、TINI SDKで行います(ダラスセミコンダクタのウェブサイトから無料でダウンロードできます)。Java仮想マシンでは、カメラの高速データを処理できないと

思うかもしれませんが、しかしTINIでは、アプリケーションとデータのやり取りをするオペレーティングシステムの下に割込サービスルーチンをインストールするネイティブメソッドが用意されています。これはTINIプラットフォームが持つ利点の一つです。HTTPなどの高レベルプロトコルを純粋なJavaに実装できると同時に、高速実行が必要な部分はネイティブ8051アセンブリによって実装できるのです。いろいろな意味で、これは両方の利点です。

TINIでは、JDK 1.1の以下のパッケージをサポートしています。

```
java.io
java.lang
java.net
java.util
javax.comm
```

TINIのJVMとPCのJVMには、若干の違いがあります。第一に、TINIでもガーベジコレクションはサポートされていますが、ファイナライザはサポートされていません。つまりリソースのクローズをシステムにさせるのではなく、プログラム自身が明示的にリソースを閉じなければなりません。このようリソース管理のほうが、組込システムには適しています。また、クラスのサイズにも制限があり

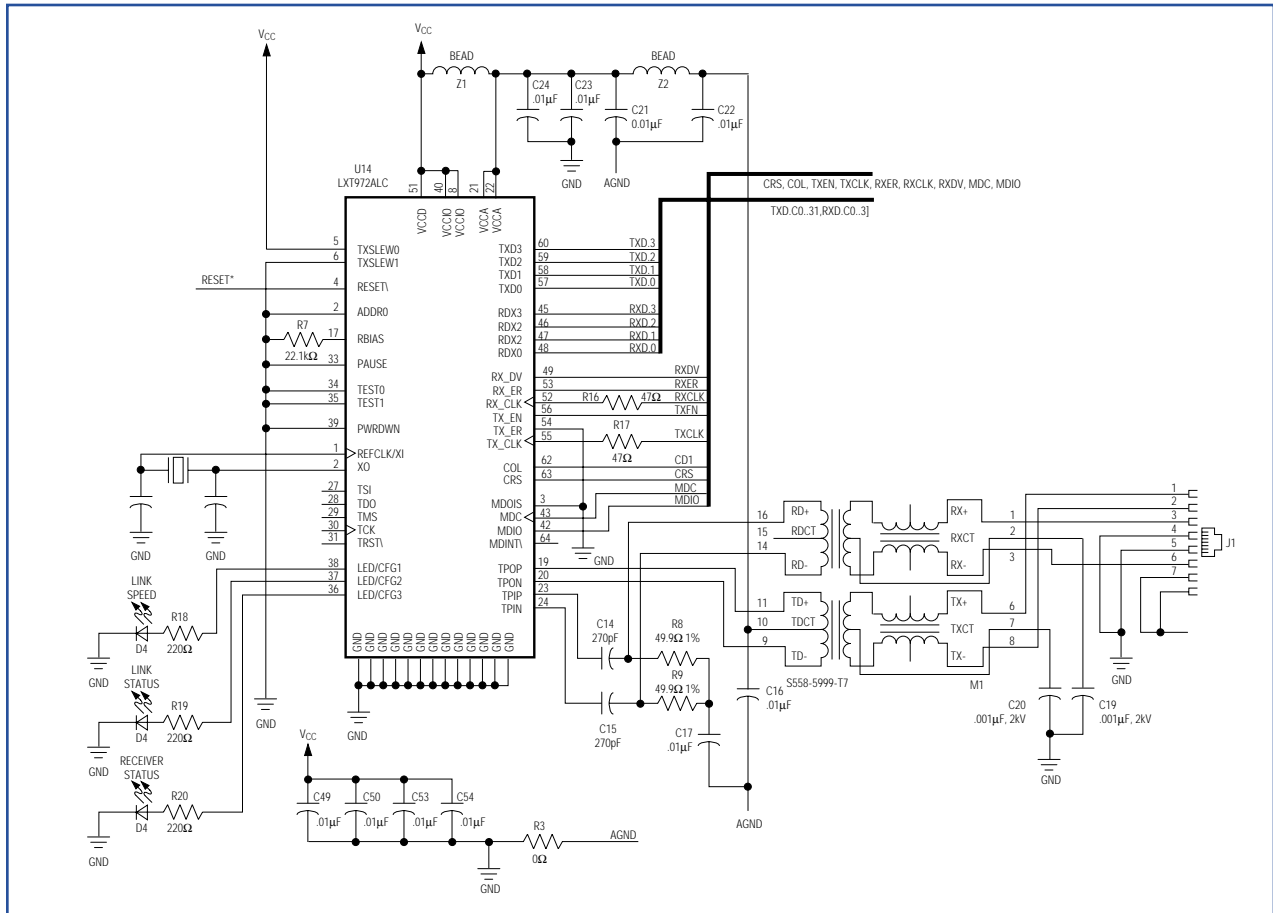


図2. DS80C400とPHYの接続は標準MIIインターフェースで行います。

ます。TINIでは、クラスファイルは64kB以下、メソッドは63ローカルまで、行列サイズも64kBまでしか使えません。システム全体も、プロセス数は8まで、各プロセスのスレッド数は32までです。制限の違いではありません。例えば、JDK 1.1はIPv6をサポートしていないので、TINIはJDK 1.4を採用しています。

TINIでは、標準Javaクラス以外に以下のパッケージが利用できます。

```
com.dalsemi.comm
com.dalsemi.fs
com.dalsemi.system
com.dalsemi.tininet
```

これらのパッケージを使用すると、低レベルのシステムアクセスやCANやI²Cのようなプロトコルのサポートなどが可能になります。また、さまざまな高レベルプロトコルもサポートされています。たとえば、軽量HTTPサーバの例がTINI SDKにあります(クラスファイルは3kBです)。

ソフトウェアの構成概要を図3に示します。最も低層にくるのはカメラ割込サービスハンドラで、間接メモリ内にあるカメラバッファへの定数ポインタを持っています。カメラはシングルフレームモードで使用するので、コマンドを送らなると画像は送られてきません。フラグを変更すると、FRCTLレジスタで指定された速度で画像が同期転送されます。今回のDS80C400では、カメラのフレームレートを10フレーム/秒にセットします。これで、384 x 288ピクセルの画像が0.1秒に1枚、1080kB/秒の速度で転送されます。

HTTP SERVER	CAMERA SERVER	CASH
NATIVE METHODS	JVM	
CAMERA ISR	NETWORK	

図3. TINIストリーミングカメラソフトウェアでは、低レベルプロトコルと高レベルプロトコルの両方をサポートしています。

8051アセンブリ言語によるカメラとの通信は、比較的簡単に実現できます。画像のピクセルは、1つのカメラレジスタから一つずつ、同期読み出しします。カメラはメモリマッピングされているので、データポインタをカメラのアドレスにセットし、movxコードを実行するだけでレジスタデータを読み書きできます。通常の8051アセンブリ言語では、一つのアドレスから別アドレスにデータを移動するだけでも、データポインタをロードし、メモリをアキュムレータに読み込み、データポインタにコピー先アドレスをセットし、アキュムレータの内容をメモリに書き出すという大変な処理が必要でした。

```
mov R0,#LOW(MEMORY_LOW)
mov R1,#HIGH(MEMORY_HIGH)
camera_loop:
;
; Move the camera address into the data pointer.
```

```
;
mov dptr,#CAMERA_ADDRESS
;
; Move the data into the accumulator.
;
movx a,@dptr
;
; Move to the address we will be writing to. Since
; this will increment every time, we will keep
it stored
; in registers. We will also need to move it one
byte at
; a time using the DPL and DPH SFRs.
;
mov dpl,R0

mov dph,R1
;
; Write the accumulator to the address
movx @dptr,a
;
; Increment the data pointer and store back in
R0 and R1.
;
inc dptr
mov R0, dpl
mov R1, dph
; Do the loop again...
```

DS80C400はデータポインタを4つ持っているため、あるアドレスから別アドレスへのデータコピーをすばやく行うことが可能です。アドレススワッピングが不要になるため、コピー操作の処理が速くなるのです。

```
;
; Set up the data pointers. We use the DPS
register to select
; what data pointer we want to use. A data
pointer move allows
; for a 24-bit address to be loaded directly.
;
mov dps,#0
mov dptr,#CAMERA_ADDRESS

mov dps,#1
mov dptr,#MEM_ADDRESS

;
; Set data pointer 0 as the current data pointer.
mov dps,#0
camera_loop:
;
; Read from data pointer zero.
;
movx a,@dptr
;
; Switch to the next data pointer. Note that doing
; an inc on this register only affects the data
; pointer-selection bit. This allows one cycle
toggling
; from one data pointer to another.
;
inc dps
;
; Store the data and increment the address.
;
movx @dptr,a
```

```

inc dptr
;
; Switch back to the first data pointer.
;
inc dps
;
; Do the loop again...
;
このようにアドレス処理の大半がループの外で行われる
ため、ループ実行が速くなります。メモリコピーについ
てもDS80C400では最適化が行われており、高速化に
対応しています。第一に全てのデータポインタのインク
リメントが1サイクルで実行されます。オートインク
リメントモードを有効にすると、データの読み書きを
行うごとにデータポインタのアドレスが自動的にインク
リメントされます。読み書きごとに2種類のデータポ
インタを切り換えるオートセレクションという機能も
あります。この結果、メモリコピーがとても簡単に行う
ことができます。
;
; Set the base address of data pointer zero.
;
mov dps, #0
mov dptr, #ADDRESS1
;
; Set the base address of data pointer one.
mov dps, #1
mov dptr, #ADDRESS2
;
; Enable autoselection and autoincrement.
;
mov dps, #(DPS_AID | DPS_TSL)
memory_loop:
;
; Read from data pointer 0, increment the
; data pointer, and toggle the selection bit
; in one instruction.
;
movx a, @dptr
;
; Write to data pointer 1, increment the data pointer,

```

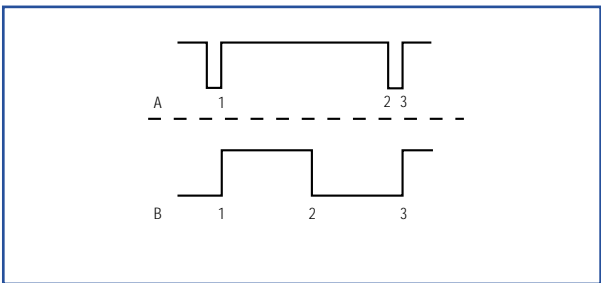


図4. 走査線がスキャンされるごとにHSYNCラインがアサートされます。Aのピクセルウィンドウは全384ピクセル、Bは192ピクセルです。いずれも左端からスタートしています。また、フレームレートはAもBも同じです。1はカメラによってスキャンラインがアサートされるタイミング、2はスキャンとともにアサートが終了するタイミング、3はカメラが次のスキャンを始めるタイミングです。Bは、Aの半分しか実際のスキャン時間はかかりませんが、1スキャンあたりの時間はAと同じです。

```

; and toggle the selection bit in one instruc-
tion.
;
movx @dptr, a
;
; Loop...

```

事例に戻ります。HSYNCラインで割込を発生させます。カメラで走査線スキャンが行われてHSYNCラインがアサートされると、ISRが起動し画像の集録を開始します。性能を高めるため、スケジューラなど、他の割込の優先順位は低くしています。こうすると画質が顕著に向上します。

カメラにはプログラマブルウィンドウというものがあり、384 x 288ピクセルの画像のうち、どのくらい使うのかを設定することが可能です。ただし、画像サイズを適切に設定するというのは、難しい作業です。大きな画像にすると画質が向上しますが、伝送時間が長くなりフレームレートが低下します。今回のアプリケーションでは、240 x 180という解像度を選びました。これがインターネットビデオの標準解像度だからということもありますが、ハードウェアから見ても利点があるのです。図4に示すように、カメラは画像を送出するとき、実際には画像行列に含まれるピクセルすべてを数えませんが、HSYNCラインがアサートされるのは、指定されたウィンドウ内にピクセルがある間だけです。つまり、画像の取り込みは100msの期間にCPUの約3/5を使用することになります。

初期化時にカメラソフトウェアはメモリマネージャから95kBの連続メモリブロックを割り当てし、ダブルバッファとして使用します。Javaでは1スレッドで画像取り込みの処理を行い、別の1スレッドでネットワーク経由の画像伝送を処理します。Javaにより必要なスレッド化機能やロック機能が提供されるため、これを単純に処理できます。

ソフトウェアデザインのISRの上層は、Java仮想マシン用カメラドライバを提供するネイティブメソッドです。TINIプラットフォームにはTINIネイティブインタフェース(TNI)が用意されており、Javaから低レベルコードにアクセスすることが可能です。ネイティブメソッドはTINIOSのネイティブAPIを呼び出し、メモリマネージャやスケジューラなど、オペレーティングシステム内部機能に直接アクセスすることができます。パラメータも渡せるだけでなく、他のJavaメソッドと同じように例外を発生させることも可能です。これらはTINI開発キットを使って構築可能なTINIダイナミックリンクライブラリ(TLIB)を通じて、Java実行モジュールとリンクされます。

このネイティブメソッドを使って、Javaからカメラにコマンドを送ります。カメラのドライバクラスの定義は以下の通りです。

```

public static final int IMAGE_BUFFER_0 = 0;
public static final int IMAGE_BUFFER_1 = 1;

/**
takePhoto takes a photograph and stores it in
the memory.

```

```

@param buffer Species what image buffer to use.
Use IMAGE_BUFFER_0 or IMAGE_BUFFER_1
*/

static native void takePhoto(int buffer);

/**
getScanlines pulls a fixed number of
scanlines out of the memory buffer. This
allows the Java application
the ability to work with fixed pieces of the
image.

@param start first scanline to copy from.
@param end last scanline to copy from
@param offset offset into the data array to
copy to
@param data array to copy into
@param buffer selects the image buffer to read
from. Useful IMAGE_BUFFER_0 or IMAGE_BUFFER_1
*/
public static native int getScanlines(int
start, int end, int offset, byte []data, int
buffer);

```

メインメソッドのtakePhotoは、1枚の画像を画像バッファに取り込みます。まず、割込がイネーブルされ、画像を1枚取り込むためのコマンドがカメラに送られます。実は、ここで小さな問題があります。この時点でJavaスレッドをスリープさせたほうが良いのですが、TINIOS関数が再入可能ではないのでスリープしたJavaスレッドをISRスリープ解除ができません。このため、TINIOSは開発者に4msごとにシステムによって呼ばれるポーリングルーチンを登録することを許可します。このポーリングルーチンで画像取り込みが終了したかどうかを確認し、終了していればスリープしたスレッドを呼び出します。Javaスレッドをスリープさせる直前に、ポーリングルーチンを設定しておきます。呼び出されたスレッドの制御はJavaに戻されます。前述のようにカメラはダブルバッファとなっているので、上書きする画像バッファの指定が必要です。Javaから画像バッファへのアクセスについては、スキャンラインのブロックをJavaバイト行列にコピーするgetScanlinesが用意されています。

記憶の問題も残っています。TINIランタイムが512kBというフラッシュメモリのうち7バンクを占めるので、ユーザアプリケーションでは1バンクしか使用できません。前述のように今回使用した高速設計では不揮発性ファイルシステムが用意されていないので、ファイルシステムを構築する必要があります。HTTPサーバがウェブブラウザに提供するJavaアプレットを含め電源投入から実行する必要のあるものすべてをフラッシュバンクに構築するのが望ましいです。実行モジュールにアプレットを含めるには、アプレットバイナリをアセンブラ互換形式に変換し、ライブラリのデータセグメントに保存します。ライブラリからJavaバイト行列へコピーするネイティブメソッドがあるので、スタートアップ時にJavaコードがアプレットのサイズを確認し、行列を生成してそのアプレットをその行列にコピーし、その内容をファイルシステムに書き出します。若干面倒ではありますが、クリーンブートからスタートでき、

スタートに必要なすべてを持つことができます。このようなタスクを実行するメソッドの例を示します。

```

/**
Extracts the sample jar file from the native
library. The demo application had a jar file
embedded inside the native library.
This allowed the jar file, the application, and
the native library all to be embedded in flash
format.

@param dummy Array to copy jar image into.
Must be of greater size than that specified in
getJarFileSize()
*/
static native void getJarFile(byte []dummy);

/**
Gets the size of the embedded jar file
*/
static native int getJarFileSize();

```

カメラドライバの上位にくるJavaはシンプルです。複数のスレッドが走りますが、その大半は互いに独立しています。スレッドの一つはHTTPサーバです。このコードはTINI SDKにふくまれているHTTPサーバのサンプルによるものです。とても軽量で、サブレットやcgi-bin処理用等に設計されてはいません。ファイルはTINIファイルシステムから読み込みます。このファイルシステムは、TINIの不揮発性メモリに実装されている階層ファイルシステムです。スタートアップ時にフラッシュメモリからカメラアプレットが読み込まれてファイルシステムに書き込まれ、index.htmlページが生成されます。

次にカメラ画像サーバがあります。カメラサーバはメインスレッドが2つあります。1つはTCPサーバソケットをポート42877にオープンにし、アプレットからの接続を待つスレッドです。どうすればサーバソケットを組込システムでオープンできるのでしょうか。実は、以下のようにPCとほとんど同じです。

```
sockpuppet = new ServerSocket(42877);
```

サーバソケットは、IPv4とIPv6の両方にバインドされます。つまり、IPv6準拠のカメラに交換しても、変更は不要なのです。現在、PCや携帯電話など、ネットワークに接続されたさまざまなデバイスとアドレスの取り合いに苦労していますが、IPv6にすればアドレス空間が広がるので、将来のネットワーク家電の取り扱いが容易になります。

プロセスが接続されると、接続時には'A'、切断時には'D'が送出されます。接続コマンドでは接続アドレスの共有ベクトルにIPアドレスが追加され、切断コマンドではIPアドレスが取り除かれます。

```

sock = sockpuppet.accept();
ch = (char)sock.getInputStream().read();

switch (ch)
{
case 'A':

cwt.addAddress(sock.getInetAddress());

```

```
break;
case 'D':

cwt.removeAddress(sock.getInetAddress());
break;
}
```

```
sock.close();
```

もう一つのスレッドは画像トランスミッタです。カメラベクトル内にアドレスがあると、そのアドレスに取り込んだ画像をUDPパケットとして送出します。カメラ側における画像の取り込みと送出はパラレル実行できるように最適化されています。取り込みはダブルバッファとなっており、片方のバッファに画像を取り込んでいる間に、もう一方のバッファから画像を送出できるのです。このような処理が可能なのは、送信ではCPU能力の50%程度しか消費されないからです。非同期ロックはJavaが行います。

```
//
// Notify the camera thread we are ready for
// the next frame.
//
synchronized(stopper)
{
    stopper.notify();
}
```

画像圧縮ハードウェアを持たないので、生の画像データが送られます。パケットレイアウトは非常にシンプルです。2バイトのヘッダの後ろに、スキャンライン5本分のデータが続きます。ヘッダの1バイト目はフレーム番号で、フレームが送信されるたびにインクリメントするロールカウンタです。2バイト目は垂直オフセットを5で割った値です。

最後に、シリアルポートで構成ツールが走ります。CASH(CAMERA SHell)と呼ばれるこのアプリケーションはメニュータイプのユーティリティで、IPアドレスの設定や接続ユーザの確認が行えます。このような機能の大半はTINI SDKに含まれるSlushシェルから取得しています。カメラの構成は、ユーザが電源投入し、シリアルポート経由で通信します。IPアドレスの設定は、CASHのシンプルなユーザインタフェースで行います。

構成が終了したカメラは、ネットワーク上で待機します。ユーザがウェブブラウザでアクセスすると、カメラからアプレットが提供され、カメラサーバとの接続が確立されて画像を表示します。TINIOS自体は同時に24のソケットをオープンできますが、スピードの問題からカメラを同時に使えるユーザ数を4に制限しました。マルチキャストとすれば、この問題をある程度解消できますが、Javaアプレットではマルチキャストをサポートしていません。

このネットワークカメラでは、平均200kB/秒のネットワーク伝送レートで1秒に4.5フレームの画像を取り込み、送出することができます。注目すべき点は、カメラにサポートするハードウェアがほとんど接続されていないと

いうことです。画像取り込みハードウェアや画像エンコードハードウェアはありません。すなわちDS80C400が画像集録から送出、ネットワークトラフィック、ウェブサーバ、シリアルポート経由のユーザとの対話まで、すべてを同時に処理しているのです。

まとめ

ジョーの話に戻りましょう。ジョーは、TINIソリューションが彼のプロジェクトにとって理想的であることを確認し、ネットワークドアロックの開発を進めました。高い能力を持つ低コストソリューションが完成し、ネットワークドアロック市場の覇者になります。トロイのソリューションにはソフトウェアで世界独占を掲げたロゴが付いていましたが、あまりに高価でした。アレックスのソリューションはカメラやネットワーク関連で扱いが難しかったため、市場に受け入れられませんでした。アミガはトロイと別れて、二度と仕事だけの生活にはしないと誓ったジョーのもとに戻ります。ビジネスに成功したジョーとアミガは小さなログハウスを買ってミネソタに引退しました。もちろん、ログハウスのすべてのドアにネットワークドアロックが付いています。

組込デバイスは、課題に合わせて設計する必要があります。そのとき、処理能力とコストのバランスが難しい点です。ネットワーク機能が必要とされる際には、バランスを取るのがさらに難しくなります。方法の一つは、8ビットマイクロコントローラを拡張してネットワークに対応させることです。不可能ではない方法ですが、どうしても遅くなってしまいます。別の方法として、組込LinuxやPC-104、Pocket PCといったデバイスを使う方法があります。このソリューションは高速で柔軟ですが、かなりの不必要な部分が追加されます。小型の32ビットソリューションを構築する方法もありますが、オペレーティングシステムのライセンス、TCP/IPスタックといった問題が発生します。

これらの中間点に位置する優れたソリューションが、DS80C400とTINIの組み合わせです。このシステムは、長い期間をかけて強化された、堅牢で包括的なTCP/IPスタックを持っています。オペレーティングシステムは多重プロセス、Java、スレッド化及び同期をサポートしています。プロセッサはデジタルカメラと通信する重量級タスクを肥大なオペレーティングシステムなしで処理できます。ジョーという凡人にとって優れたシステムだということは、誰にとっても優れたシステムになりうると言えるでしょう。

関連記事が、2003年2月号の*Embedded Control Europe*に掲載されています。