組み込み機器のソフトウェア開発の実際

第1章 クロス開発とは何か

山際 伸一

本章では，組み込み機器とは何か，そしてその実現方法について簡単に説明した後，その上で動作する組み込みプログラムの開発が，Windows上で動作するアプリケーション・プログラム開発とどのように異なるかについて解説する．さらに，組み込み機器開発で特有なデバッグ手法についても，さまざまな形態のデバッガを例にあげて解説する．（編集部）

1 組み込みシステムとは

● 組み込みシステムって何？

「組み込みシステム（Embedded System）」とは，いったい何でしょうか．組み込みシステムとは非常に簡単にいうと，ある特定の処理を行うために専用に設計され，組み込まれたシステムであるといえます．

組み込みシステムにはさまざまなものがあります．たとえば，電子炊飯器は組み込みシステムの例としてふさわしいでしょう．炊飯器を使うとき，炊飯モード（最近だと玄米モードとか早炊きモードなどがある）を選択し，スタート・ボタンを押すといった操作を行います．この操作の順序やモードの判別，またモードごとの温度や時間の制御は，炊飯器に“組み込まれた”専用システムが行っているわけです（**イラスト1**）．

この定義からいうと，パソコンは組み込みシステムとはいいがたいところがあります．パソコンはインストールするソフトウェアによって，さまざまな処理を行わせることができるからです．

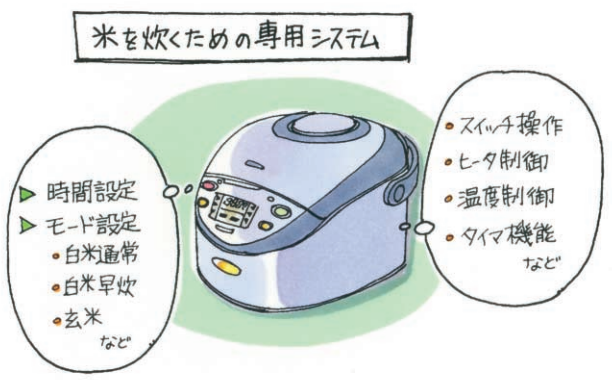


イラスト1　電子炊飯器はれっきとした組み込みシステム

● 組み込みシステムの実現方法

組み込みシステムを実現するためには，大きく分けて次の二つの方法があります．

(1) すべてハードウェアで制御している場合

まず一つは，**図1(a)**のようにハードウェアだけでシステムを実現する方法です．たとえば，選択したモードをフリップフロップで保持し，それにより選択された回路が動作して機器のさまざまな状態を制御します．

図1(a)の回路は標準ロジックの74シリーズのICが使われていますが，最近ではより高度で規模の大きな回路を必要とすることから，PLD（Programmable Logic Device：プログラミング可能なロジック・デバイス）やASIC（Application Specific Integrated Circuit：アプリケーションのために専用に設計されたIC）が使われています．

(2) CPU＋ソフトウェアで処理している場合

この方法は，汎用のCPU（Central Processing Unit）とROM（Read Only Memory）やRAM（Random Access Memory）などのメモリを接続したハードウェア（CPUボード）を必要とするので，(1)の延長線上にあるといえばそうなのですが，それだけではシステムは動作しません．CPUを動かすためのプログラムを作成して，メモリに格納して初めてシステムとして完成します〔**図1(b)**〕．つまり，まったく同じCPUボードでも，メモリに格納するプログラムを入れ替えると，別の動作を行わせることも可能です．

組み込みソフトウェアとは，CPUボードのメモリに格納し，CPUボード上で動作させるソフトウェアのことを指します．

本特集では，(2)の実装を主眼において，組み込みソフトウェアを作成していく方法を解説します．

● ソフトウェアのメリット

先ほど説明した(1)の方法では，回路図を作成すればよいわけですが，(2)の方法では，CPUボードの回路図と，その上で走らせるソフトウェアの両方を開発する必要があります．一見，

(2)の方法はハードウェアとソフトウェアの両方が必要となるので，設計がたいへんだと思うかもしれません．しかし，(2)の方法には次のような大きな利点があります．

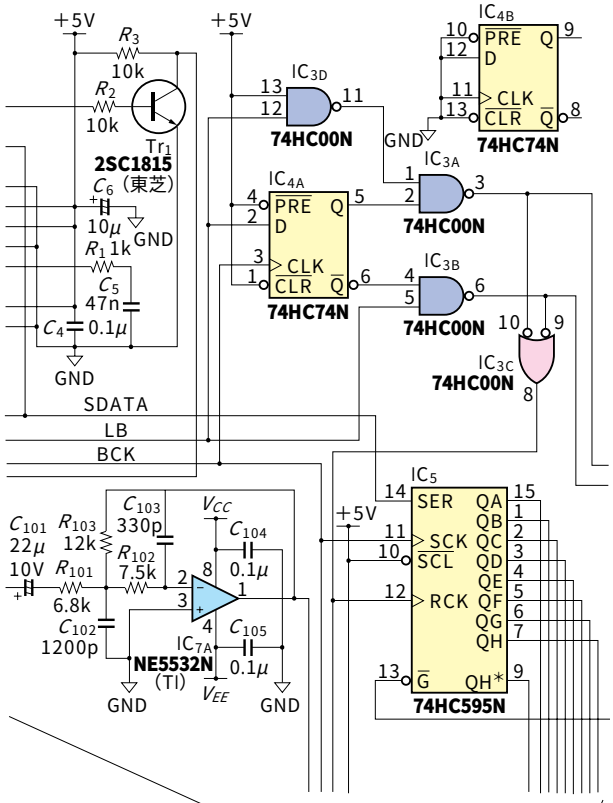
一般的にソフトウェア開発では，開発ツールを安価でそろえることができます．すでに説明したように，同じCPUボードでもソフトウェアを入れ替えることで別の処理を行わせることができます．また，メモリの容量に空きがあれば，機能を追加することも可能です．さらにプログラムに不具合(バグ)が見つかっても，即座に修正することができます．

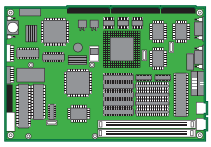
一方，ASICなどのハードウェア開発では，一般的に高価な開発ツールを必要とするので，開発には大きな投資や時間が必要になります．また別の処理を行わせたり機能を追加するには，そのための回路が必要になります．また回路にまちがいが見つかったときに，それを修正するために多くの手間やコストがかかります．

● ソフトウェアのデメリット

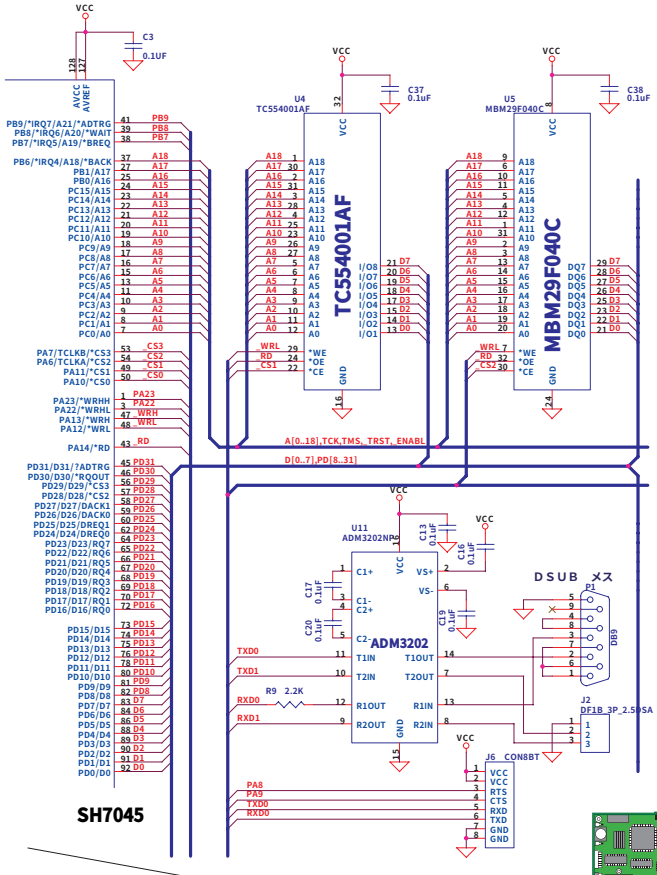
ソフトウェアで開発する際にもいくつかデメリットはあります．

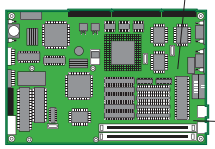
まず，ソフトウェアを動作させるには，CPU＋メモリというハードウェアが必要だということです．とはいえ，わざわざ





(a) すべてハードウェアで実現する





(b) CPU＋ソフトウェアで実現する

```
#define SIG_IRQ 0x18
#define SIG_FIQ 0x1C

typedef struct
  {
    short *memAddr;
    short oldInstr;
  }
stepData;

int registers[NUMREGBYTES / 4];
stepData instrBuffer;
static const char hexchars[] = "0123456789abcdef";
static char remcomInBuffer[BUFMAX];

extern _breakpoint_address;
extern _ilegalinst_address;

char * strcpy(char *dest, const char *src){

  while(*src != '¥0'){
    *dest = *src;
    src ++;
    dest ++;
  }
  *dest = 0;
  return dest;
}
```

ソース・リスト

図1　組み込みシステムの実現方法

ASICを開発するよりは，はるかに安い価格でCPUボードを実現することができます．

もう一つは処理速度です．ハードウェアだけで処理する場合とソフトウェアで処理する場合を比較すると，一般的にはハードウェアでの処理のほうが高速です．人間がボタンを押してLEDを点灯させる…といったようなそれほど高速でない処理は，ソフトウェアでも十分に処理できますが，nsオーダの性能が要求されるような処理は，ハードウェアで行わせるべきでしょう．

もちろん，ソフトウェアでの処理性能を上げるために，より高速なCPUを使うという方法もあります．しかし高性能なCPUはコストも高く，それが製品に跳ね返ってくるので，どれくらいの性能のCPUを採用するかといった問題は，実は非常に重要な問題でもあります．

● 組み込みシステム向けCPU

組み込みソフトウェアを動作させるために欠かせないのがCPUです．多くのベンダから「組み込みシステム向けCPU」なるものがリリースされているということは読者の皆さんもご存じでしょう．この「組み込み向け」という意味は何でしょうか．

組み込みシステムにおいては，次のような事がらに配慮しなければなりません．

(1) コスト

基板面積，部品価格など，物理的なコスト．大きなチップは組み込みシステムに適さない．

(2) 消費電力

バッテリ駆動の製品は当然ながら，据え置き型でも低消費電力が望ましい(冷却ファンなどは故障の原因にもなる)．

◆COLUMN1 HDLを使ったハードウェア開発

ソフトウェアとハードウェアの比較で，ハードウェアの設計には回路図を，ソフトウェアではソース・リストを使うと説明しました．しかし最近のハードウェア開発では，ソース・リストで入力するという設計手法が普及しています．

ハードウェアを記述するソース・リストを，HDL(Hardware description language)と呼びます．現在ではHDLとして一般的なものに，VHDLやVerilog-HDLがあります．また，最近ではC言語でハードウェアを開発することもできるようになってきました．

HDLによるハードウェア開発は，**リストA**のようなソース・リストを入力し，これを論理合成してハードウェアの回路を生成します．回路がまちがっていれば，テキスト・エディタ上でソース・リストを修正して再度論理合成すればよいわけです．回路図を使わず，大規模な論理回路を設計できるので，ASIC開発などでは必須の設計手法となっています．

リストA　HDLソース・リストの例(VHDL)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity crc16_general_multi is
    generic (
        ROUND_NUM   : integer   := 8
    );
    port (
        data_in     : in  std_logic_vector(
                              ROUND_NUM - 1 downto 0);
        shift_in    : in  std_logic;
        init_in     : in  std_logic;
        init_val    : in  std_logic_vector(15 downto 0);
        final_val   : in  std_logic_vector(15 downto 0);
        genpoly_val : in  std_logic_vector(15 downto 0);
        reset_n     : in  std_logic;
        clk         : in  std_logic;
        data_out    : out std_logic_vector(15 downto 0)
    );
end crc16_general_multi;

architecture RTL of crc16_general_multi is

    function fast_round(prev_parity : std_logic_vector
                                          (15 downto 0);
                        data    : std_logic_vector
                                    (ROUND_NUM - 1 downto 0);
                        genpoly : std_logic_vector
                                          (15 downto 0))
            return std_logic_vector is
        variable tmp_result : std_logic_vector(15 downto 0);
        variable d_bit      : std_logic;
        variable d_val      : std_logic_vector(15 downto 0);
    begin
        tmp_result  := prev_parity;
        for I in ROUND_NUM - 1 downto 0 loop
            d_bit   := tmp_result(15) xor data(I);
            for J in 0 to 15 loop
                d_val(J)    := d_bit;
            end loop;
            tmp_result  := ((genpoly and d_val) xor
                           (tmp_result(14 downto 0) & '0'));
        end loop;
        return tmp_result;
    end fast_round;

    signal  parity_reg  : std_logic_vector(15 downto 0);

begin

    u0: process (clk, reset_n) begin
        if (reset_n = '0') then
            parity_reg  <= (others => '0');
        elsif (clk'event and clk = '1') then
            if (init_in = '1') then
                parity_reg  <= init_val;
            elsif (shift_in = '1') then
                parity_reg  <= fast_round(parity_reg,
                                    data_in, genpoly_val);
            end if;
        end if;
    end process;

    data_out    <= parity_reg xor final_val;

end RTL;
```


News Flash ――アイワイバンク銀行で，平成18年よりWindowsServerベースの勘定系システムが稼動
(株)アイワイバンク銀行は，平成18年を目途に，現行の勘定系システムをWindowsServerベースのシステムに移行することを決定した．システム構築を日本ユニシス(株)が，システム・インテグレーションを(株)野村総合研究所が担当する．

(3) 性能

処理対象に対して十分に速い(とくに人間の操作に十分追いつける程度).

(4) 供給の安定性

工場の制御システムなどでは,5年や10年といった長期間使われる場合が多い.修理しようとしたときにすでに部品がないようでは困る.

これら要求を十分に満たすものが「組み込み向けCPU」なのです.

表1に,組み込み向けと呼ばれるCPUの主なものを示します.組み込みシステムとは,最初に説明した電子炊飯器のような簡単なものから,DVD/HDDレコーダといった高機能なものまで,さまざまなものがあります.そのため性能や消費電力,そしてコストに応じて,いろいろなCPUが各社からリリースされているのです.

また,組み込みCPUではないものの,いちばん下に,最近のパソコンに用いられるCPUを挙げてみました.とくに,消費電力では,組み込み向けプロセッサは非常に有利であることがわかると思います.しかし,プロセッサとしての処理能力は,パソコン向けのプロセッサよりも劣っています.これは,チップの大きさを小さくすることでコストダウンを図るためや,消費電力の低減にともなうトレードオフによるものです.

組み込みシステムは特殊で,たいてい,プロセッサの周辺チップはあらかじめ決まっていることが多いため,たとえば,DRAMコントローラや,グラフィック・コントローラといったハードウェア・ロジックを搭載しているものが多く見うけられます.このような,「アプリケーションを見据えた」プロセッサ・アーキテクチャをとるのも組み込みプロセッサの特徴です.

表1 組み込み向けCPUのいろいろ

プロセッサ名	消費電流/電力	クロック周波数(Hz)
PICマイコン		数～十数M
Z80		数～十数M
H8	60mA	十数M
SH-2(SH7145)	200mA	50M
SH-4(SH7750)	500mA	200M
MIPS(V_R7701)	6W	400M
ARM(XScale PXA250)	500mW	400M
PowerQUICC II(MPC8280)	2W	500M
参考:Pentium4	115W	3G

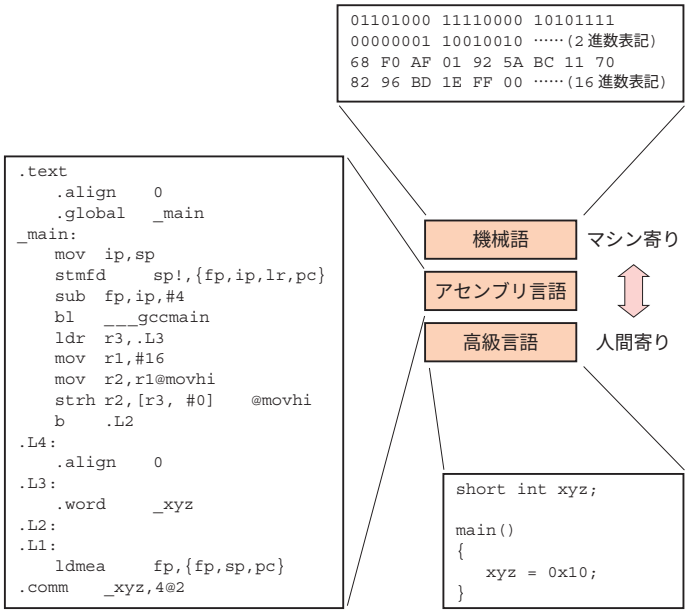


図2 機械語/アセンブリ言語/高級言語

2 組み込みソフトウェア開発方法

● 機械語/アセンブリ言語/高級言語

ソフトウェアとは,CPUを動かすための命令の集まりです.**図1**(**b**)では英語に似たアルファベットや数字が並んだ文字列が示されていますが,CPUが実際に理解するのは'0'と'1'を組み合わせた数値でしかありません.これを機械語と呼びます.機械語のままでは人間が理解できないので,それを人間が理解できるような文字列に置き換え,ソフトウェアを開発します.これをアセンブリ言語と呼びます.

アセンブリ言語は,加算ならADD,引き算ならSUBといった,CPUの一つ一つの動作を記号に置き換えたものです.機械語よりは人間が理解しやすいものの,処理単位があまりにも細かすぎて,大規模なプログラムを作成するのには向きません.そこで,より人間の考える処理内容を記述できような言語が考え出されました.これを高級言語と呼び,現在の組み込みソフトウェアでは一般的に,C言語が使われています(**図2**).

ちなみに,**図1**(**b**)で示したソフトウェアは,C言語で書かれたリストです.

● PC上でのWindowsプログラミング

読者がWindowsプログラムをPCで開発する場合を考えてみてください.つまり,PC上でのプログラミングということになりますが,PCはたいていの場合,ディスプレイ,キーボード,マウスといった開発者に優しいインターフェースがあり,それらを操作することでプログラムを作成していくことになります.また,Windowsでは,**図3**のようなインタラクティブな開発環境が提供されているので,作ったその場で動作確認を,ディスプレイを見ながら可能で,さらに,作業状態をHDDに保存しておき,あとで作業を再開することができます.

このような,ソフトウェアを動作させる環境上でプログラム開発も行うことを「ネイティブ開発環境」といいます.また自分自身で自分用のプログラムを開発することになるので,セルフ開発とも呼びます.

● 組み込み機器のソフトウェア開発

組み込みシステムの場合,必ずしも,ディスプレイを接続で

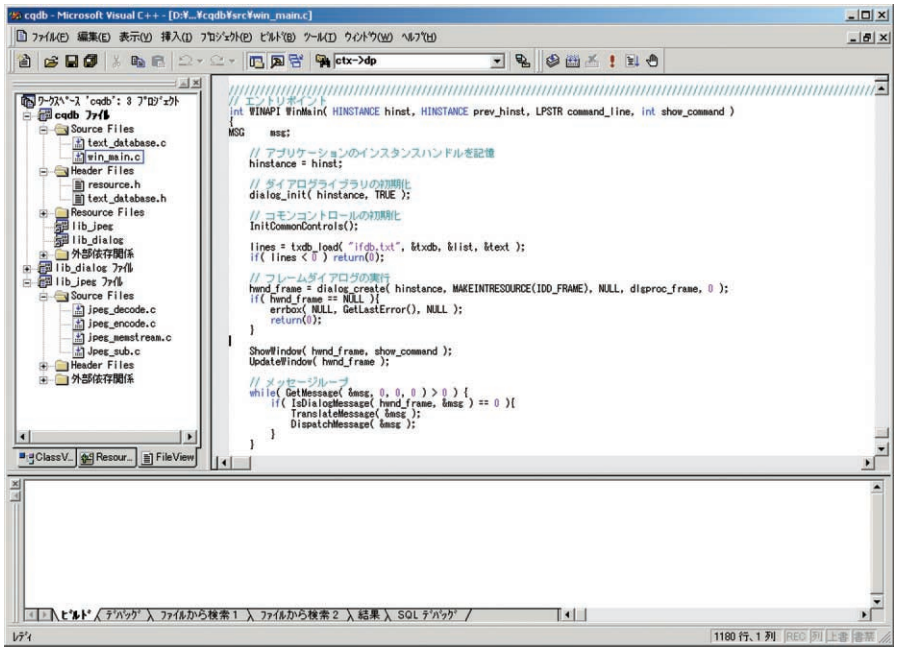


図3
ネイティブ開発環境の例（Visual C++）

きるRGBコネクタや，文字を自在に入力するためのキーボード接続コネクタが存在するわけではありません．それらのコネクタをもっていない場合も多くあります．たとえば，最初に説明した電子炊飯器では，状態出力としてはLED表示や，せいぜいLCDで数桁の数字を表示する程度しかできません．もちろんキーボードも接続することはできません（**イラスト2**）．

しかし，C言語を代表とする高級言語でソフトウェアを開発するには，英語のようなアルファベットの文字列や数字をたくさん入力しなければなりません．また，正しく文字が入力されたことを画面表示などで確認しなければなりません．

● クロス開発環境とは

そこで，画面やキーボードやストレージを持っているPC上で組み込み機器用のソフトウェアを開発し，作成したソフトウェアを組み込み機器に転送して実行するという手法が採用されています．このように，プログラムを開発する環境と実行する環境が異なる場合を，「クロス開発環境」と呼びます（**図4**）．

ネイティブ開発環境では「開発しているマシンのプロセッサ＝アプリケーションが実行されるマシンのプロセッサ」となりますが，組み込みシステムの場合，このイコールの関係が成り立ちません．そのためクロスと呼ばれているようです．

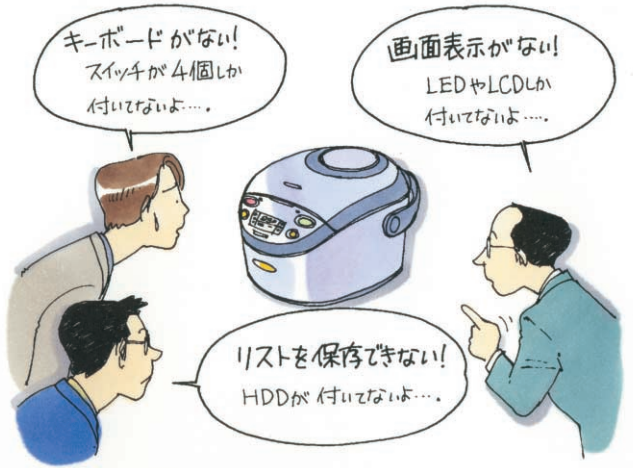


イラスト2　キーボード入力や画面表示の無い組み込み機器で，どうやってプログラムを開発する？

◆COLUMN2　PLDを使ったハードウェア開発

ソフトウェアとハードウェアの比較で，ハードウェアは修正がたいへんだということを説明しました．しかし，現在ではハードウェアの修正も簡単にできるPLD（Programmable Logic Device）と呼ばれる便利なデバイスも存在します．

74TTLによる回路では，回路を修正するにははんだごてなどを使って端子の接続の状態を変更する必要がありました．しかしPLDは，端子の接続状態や回路情報をデバイス内部に保持して動作するので，はんだごてなどを使わなくてもPLDの中身を書き換えれば，回路状態などを変更することができるのです．

現在では，より大容量のPLDとしてFPGA（Field Programmable Gate Array）と呼ばれるデバイスも存在しています．ASICの開発は高額な開発費や開発期間が必要ですが，FPGAは開発者の手元でいつでも回路を開発できるので，ASIC開発のプロトタイプとしてもよく使われます．最近では大容量のFPGAも安くなってきているので，量産品にそのままFPGAを採用する事例も増えています．

News Flash ―富士ゼロックス，上海工場の生産ラインを拡張し，ディジタル複写機と複合機の生産を拡大へ
富士ゼロックス（株）は，上海工場の生産ラインを拡張し，11月1日より本格的に稼動を開始した．2005年度には，ディジタル複写機と複合機の生産能力を3倍に拡大する見込み．

図5に代表的な組み込みクロス開発環境を示します．開発するCPUボードは，クロス開発環境では「ターゲット・システム」と呼びます．そして開発のために使うマシン（組み込みシステムのプログラムを実際に入力するマシン）を，「ホスト」と呼ぶのが一般的です．

ホストには，WindowsやLinuxなどを走らせたPCを使います．ターゲット・システムはホストPCとケーブルなどで接続します．そしてホスト上で組み込みシステム用のプログラムを作成し，ターゲット・システムにダウンロードして動作確認をしていくことになります．

● ソフトウェア開発ツール

C言語に代表される高級言語で作成したプログラムは，コンパイラと呼ばれるツールでアセンブリ言語に変換されます．アセンブリ言語は，アセンブラと呼ばれるツールで機械語に変換されます．また，大規模プログラムの開発では，機械語に変換されたサブルーチンなどの塊を連結して，一つのプログラムを作成します．この機械語に変換されたサブルーチンの塊をオブジェクト・ファイルと呼び，これを連結して最終的な実行形式ファイルを生成するツールをリンカと呼びます．これら3種類が，ソフトウェア開発で重要なツールです．

ネイティブ開発環境でもクロス開発環境でも，これらソフトウェア開発ツールの基本的な構成は同じです．

(1) クロス・コンパイラ
(2) クロス・アセンブラ
(3) リンカ

クロス開発用のコンパイラやアセンブラは，クロス・コンパイラやクロス・アセンブラと呼びますが，リンカの場合はクロス・リンカとはあまり呼びません．

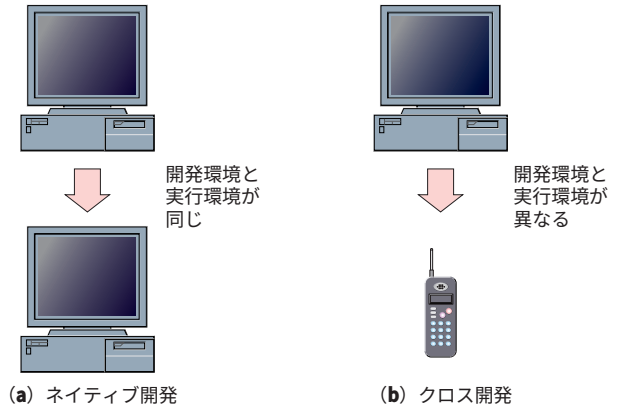


図4　ネイティブ開発とクロス開発

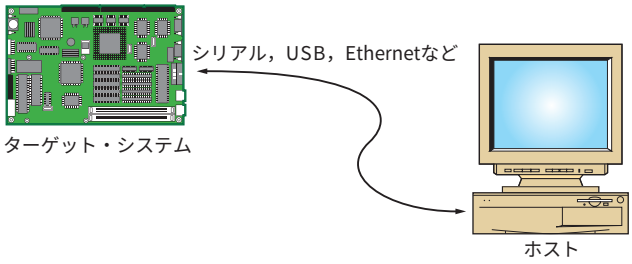


図5　クロス開発環境の例

3 クロス開発環境向けデバッガ

● デバッグとは

ソフトウェアを作成するうえで，必ず発生するのはバグです．当然ながら，細心の注意を払ってプログラムしていくことが重要ですが，人間が開発している以上，プログラムのミスはかならず発生すると考えるべきです．

高級言語の構文記述まちがいなどは，コンパイルの段階でエラーが発生するので，プログラムの入力ミスはすぐに気づきます．しかし，プログラムの構造上のまちがいなどは，コンパイルの段階ではエラーが発生しないので，プログラムのミスには気が付きません．ターゲット・システムにダウンロードしても，意図したとおりに動作しないという状況に陥り，どこに問題があるのかすぐにはわかりません．

デバッグとは，このようなバグを発見して修正する作業のことで，そのときに使用するツールをデバッガと呼びます．

◆ COLUMN3　デバッガとモニタの違い

デバッガはステップ実行やブレーク機能など，デバッグに必要な各種機能を備えたツールを指します．

もう一つ，組み込み開発の世界では，モニタということばを使う場合もあります，モニタとは，基本的にはことばの意味するとおり，値や状態を表示するだけに機能を絞ったツールです．

本来，組み込みシステムは，特定の用途に特化したシステムなので，電源を投入してすぐにその処理が始まります．しかし，開発用に使う評価ボードと呼ばれるCPUボードは，それ自体ではとくに明確化された処理内容があるわけではありません．そこで，このような評価ボード上のROMには，組み込みプログラム開発者が作成したプログラムを，ホストからダウンロードできるような機能だけを書き込んで出荷されるのが一般的です．このROMに書き込まれたプログラムをROMモニタと呼びます．

ROMモニタといった場合には，プログラムのダウンロード機能や，メモリやレジスタのダンプ機能，そして指定したアドレスからのプログラムの実行機能程度を備えているのが一般的です．ステップ実行やブレーク機能など，本格的なデバッグ機能をもたないものがモニタと呼ばれます．

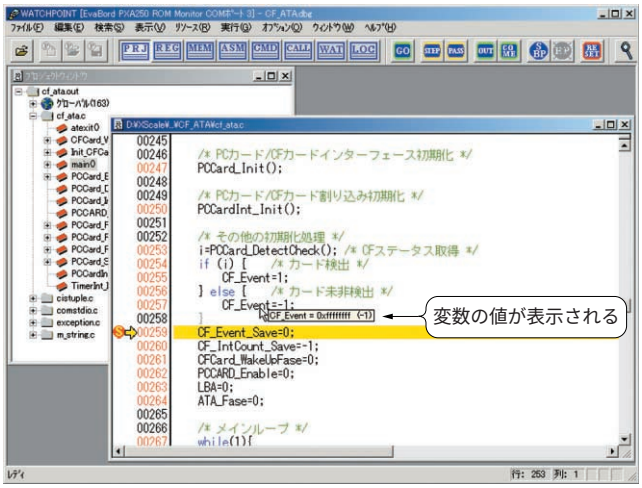


図6　カーソルを持っていくだけで変数の値が表示されるデバッガの例

● デバッガの基本機能

一般的に使われる組み込み向けCPUは，どんなに遅くても数MHzのクロック周波数で動作します．機械語レベルで1秒間に数万命令は実行してしまいます．「あれ？　いま何かおかしな動作をしなかったか？」と思っても，あっという間に実行が過ぎてしまいます．そこでデバッガには，命令を1命令ずつ実行するステップ実行機能が用意されています．ステップ実行をすることで，プログラムがどのような順番で実行されるかを手に取るように知ることができます．

また，起動してから最後のほうに実行するルーチンをデバッグする場合，すべてのプログラムを頭からステップ実行していったのでは時間がかかりすぎるため，ある地点までは一気にプログラムを実行し，指定した地点でプログラムの実行を止め，そこからステップ実行をして動作を確認するという方法もあります．このある地点（ブレーク・ポイント）でプログラムを止める機能を，ブレーク機能と呼びます．また，ブレーク・ポイントを指定せずにプログラムを実行させ，強制的にプログラムの実行を止める機能として，強制ブレーク機能も使われます．

プログラムの動作確認では，プログラムがどのような順番で実行されていったかを確認する以外に，変数がいまどのような値を保持しているかを確認する場合もあります．強制ブレークなどでプログラムの実行を止め，値を確認したい変数を指定すると，その値を表示する機能をウォッチ機能などと呼びます．グラフィカル・ユーザ・インターフェース（GUI）対応のデバッガでは，カーソルを変数のところに持っていくと，チップ・ウィンドウなどで変数の内容が表示されるものもあります（**図6**）．さらにその変数の内容を変更することもできます．

これらの機能はWindows上のアプリケーションを作成するためのデバッガでも，同様の機能をもっています．いわゆるデバッガとしてもっとも基本的な機能といえるでしょう．

● クロス開発環境向けデバッガ

ネイティブ環境では，作成したプログラムはその環境で実行することができます．しかしクロス開発環境では，作成したプログラムはターゲット・システムに転送する必要があります．

一般に，クロス開発用のデバッガにはホストとターゲット・システムを，**図5**のように何らかの通信インターフェースで接続し，そのインターフェースを使って，ホスト環境で作成したプログラムをターゲット・システムにダウンロードする機能をもっています．

また，デバッガとしての基本機能として，先ほど説明したステップ実行やブレーク機能，変数のウォッチや変更機能などを実装しています．

さらに，組み込み機器では物理的なメモリの状態やレジスタの状態を把握して，より細部の動作をチェックすることもできるようになっています．そのため，メモリやレジスタの内容を16進数表示（ダンプ機能）したり，その内容を書き換える機能（メモリ・エディット機能）などももっています．

● 市販の組み込み機器開発用デバッガのいろいろ

市販されている代表的なデバッガを，種類別に分類してみました．おおよそのところ，世の中にはこれらの分類か，またはこれらの組み合わせが商用デバッガとして用いられています．

(1) フルICE（In Circuit Emulator）デバッガ

CPUそのものを置き換えて，CPUの動作をまねて（エミュレーション）プログラムの動作を調べることができるツールです．ICE本体とホストの間は，専用インターフェースやEthernet，最近ではUSBなどを使って接続します．

また，CPUの動作をまねるだけでなく，ターゲットCPUボード上のROMやRAMもICE本体に内蔵（エミュレーション・メモリ）したり，CPUが命令を実行していったアドレスを保持して（トレース・メモリ），それを表示する機能などを実装したICEを，CPU動作のあらゆる機能をサポートできるICEという

◆COLUMN4　デバッガ本体とユーザ・インターフェース

その昔のICEは，ICE本体そのものに画面やキーボードなどユーザ・インターフェース（UI）まで持ったものもありました．しかし画面やキーボードの機能はPCそのものなので，ICE本体にはUI機能をもたせずに，それをPCを使って肩代わりさせる方式が一般化しています．よって現在では，ICE本体以外にホストとなるPCにUIを制御するソフトウェアをインストールすることになります．

ここで気をつけなければならないのは，単にデバッガといった場合に，そのことばが指し示す範囲です．Windowsアプリケーション開発などで使うVisual C++などにもデバッガ機能がありますが，これはソフトウェアのみで構成されています．このことからクロス開発でのデバッガを，UI部分のみを指して使う場合もあるので注意してください．

News Flash ―積水化学工業とNTTデータが資本提携
積水化学工業（株）と（株）NTTデータは11月1日，積水化学工業の100%子会社である（株）セキスイ・システム・センターの発行済み株式の60%をNTTデータが譲り受け，資本提携することで合意した．

写真1　フルICEデバッガの例〔(株)ソフィアシステムズ　UniSTAC http://www.sophia-systems.co.jp/〕

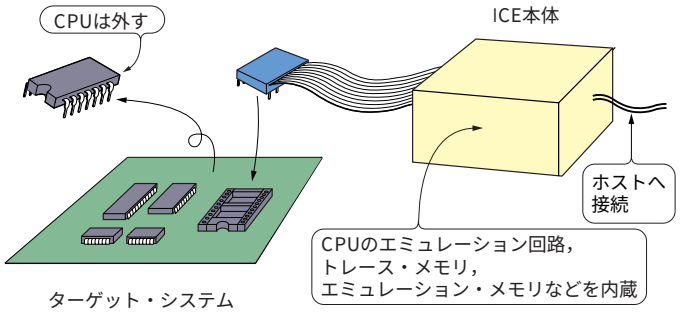


図7　ICEの構成

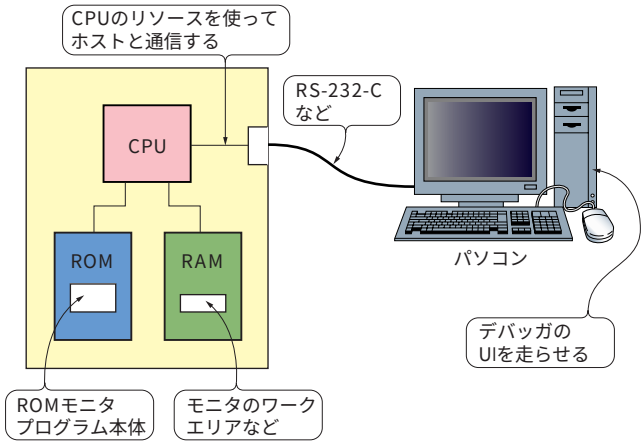


図8　ROMモニタ型デバッガの構成

意味でフルICEと呼ぶこともあります．**図7**にICEの構成を，**写真1**にフルICEデバッガの例を示します．

フルICEは，エミュレーション・メモリなどを内蔵しているので，ターゲットCPUボードが完全には動作しない状態でも，エミュレーション・メモリ上でプログラムの開発を進めることができます．つまりターゲット・システムのCPUとROMやRAMなどローカル・バス接続に問題があっても，ICEを使ってローカル・バス上のどこに問題があるかデバッグを進めることができるのです．

もちろん，ターゲット・システムのメモリやI/Oを，デバッガが占有してしまうこともありません．すべてのリソースをアプリケーションが使うことができます．

(2) ROMモニタ型デバッガ

コラム3でROMモニタについて説明しましたが，このROMモニタに，より本格的なデバッグ機能を実装してデバッガとして使えるようにしたものが，ROMモニタ型デバッガと呼ばれるものです．

図8にROMモニタ型デバッガの構成を示します．ホストとの接続には，簡易的なものではシリアル・インターフェースを使うのが一般的です．そのため，ターゲット・システム上にシリアル・インターフェースが必要です．このシリアル・インターフェースはデバッガがホストと通信するために使用するので，アプリケーションが使うことはできません．また，ROM

◆COLUMN5　ROMエミュレータ

その昔のCPUボードでは，紫外線でメモリ内容を消去し，専用のROMライタでメモリ内容を書き込むUV-EPROMと呼ばれるROMを使っていました．メモリの消去や再書き込みには，数分から数十分という時間が必要なので，一部をちょっとだけ修正する場合でも非常に時間がかかりました．また，UV-EPROMには書き換え寿命があるので，何度も消去/再書き込みをしていると，次第にメモリ内容を消去できなくなったり，値を正しく書き込めていないといった問題も発生するようになります．

そこでプログラムの開発中は，消去や再書き込みの時間の短縮や，UV-EPROMの寿命をむだに減らすことを避けるため，ROMの動作をまねるツールを用いてデバッグを進めることがありました．これがROMエミュレータです．

ROMエミュレータの中は，実際にはSRAMが使われ，ホストからダウンロードするとそのSRAMには書き込めるが，ターゲットCPUからは書き込み信号が届かないようになっているため，ROMに見えるという回路構成をしています．

現在では，ROMにはフラッシュ・メモリを使うのが一般的です．フラッシュ・メモリはCPUボード上に実装したままでも消去や書き換えができるので，UV-EPROMで必要だった紫外線を使うイレーサやROMライタが不要です．とはいえ，消去や書き換えには数十秒から1分程度の時間がかかったり，UV-EPROM同様書き換え寿命もあるので，頻繁に書き換えるには向きません．

よって現在でも，ROMエミュレータが使われる場合もあります．

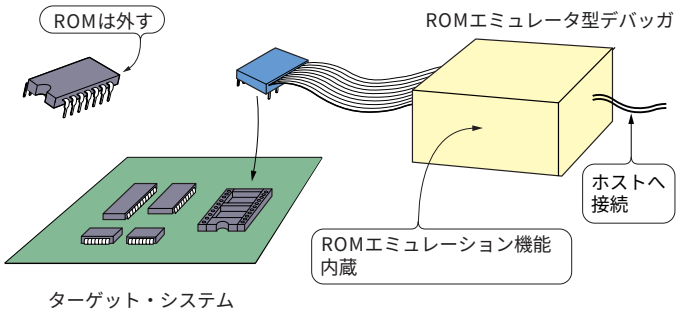


図9　ROMエミュレータ型デバッガの構成

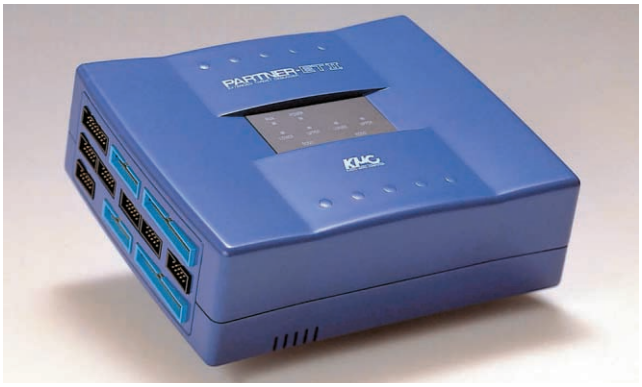


写真2　ROMエミュレータ型デバッガの例〔京都マイクロコンピュータ(株)　PARTNER-ET II　http://www.kmckk.co.jp/〕

上にはホストからの指示に従い，メモリやレジスタの内容を読み書きする通信プログラムを書き込んでおき，ターゲット・システムのCPUにそれを実行させます．

このようにROMモニタ型デバッガは，ターゲット・システムに実装されたインターフェースやメモリの一部を，デバッグ機能のために使ってしまうことや，そもそもホストとの通信プログラムをターゲット・システム上でプログラムとして実行させるので，ローカル・バスに問題があってROMを読み出せないような場合は，デバッガ自体が動作しません．ここがICEと大きく異なるところです．

また，一般的なROMモニタ型デバッガでは，ROM領域にプログラムをダウンロードすることはできません．ただしROMとしてフラッシュ・メモリが搭載されたシステムでは，フラッシュ・メモリへの書き込みアルゴリズムに対応すれば，直接ダウンロードすることも可能です．

また，たとえフラッシュ・メモリにダウンロードできたとしても，ROMモニタ型デバッガはROM領域にブレーク・ポイントを設定することはできません(一部のROMモニタ型デバッガでは，CPU内蔵のハードウェア・ブレーク機能を使って，最大2か所程度のブレーク・ポイントを設定できるものもある)．

このように，ROM領域の扱いに制限がある場合が多いので，ROM化を想定したプログラムのデバッグには使いにくい場面もあります．

なお，第5章で解説するGDBスタブは，ROMモニタ型デバッガに分類されます．

(3) ROMエミュレータ型デバッガ

ROMエミュレータ型デバッガは，基本的にはROMモニタ型デバッガに近い構造をしていますが，ROMエミュレーション機能をもっていて，ROM上で動作させるプログラムをダウンロードしてデバッグすることもできます．もちろんROM上に自由にブレーク・ポイントを設定することもできます．

図9にROMエミュレータ型デバッガの構成を，**写真2**にROMエミュレータ型デバッガの例を示します．ICEではCPUを外してCPUソケットに差し込みましたが，ROMエミュレータ型デバッガではROMをソケットから抜いて，ROMソケットにデバッガを接続します．ホストとの接続にはデバッガ本体に用意されたインターフェースを使うため，ターゲット・システムのリソースを使うことはありません．また，エミュレーショ

◆COLUMN6　高密度実装＆SoC対応＝JTAGデバッガ？

CPUがDIPパッケージで取り外しができた時代は問題がなかったのですが，高密度実装時代になり，ICのパッケージがQFPパッケージになると，CPUを取り外すことができなくなってきました．そこでICE用に隣にDIP用のソケットを用意して，ICEを使う場合にはQFPのCPUを非動作状態にして開発を行うこともありました．また，高価ですがQFP用のコネクタも存在するので，開発用にはそれを使うこともあります．

いずれにせよ，CPUを取り外して接続するという構成を採るICEは，実現しにくい状況になってきています．

同じことがROMエミュレータ型デバッガにもいえます．昔はUV-EPROMが一般的だったため，28ピンや32ピンのDIPパッケージのROMソケットが必ず使われていましたが，最近ではフラッシュ・メモリに置き換わり，ROMソケットがありません．

そしてさらに現在では，ボード上に構成していた回路のほとんどをLSIの中に実装してしまう，SoC(System On Chip)が採用されるようになってきました．CPUやROMが取り外しできないどころか，LSIの中にいっしょに入ってしまうという事態になったのです．

このような事態に対応すべく，CPUのデバッグ専用の信号線を用意して，それを外に引き出してホストと通信させるという手法が考え出されました．ここでは詳細は解説しませんが，現在ではJTAG端子にCPUデバッグのための機能を割り当て，JTAG端子からCPUの状態を取得したり，状態を変更することができるようになりました．

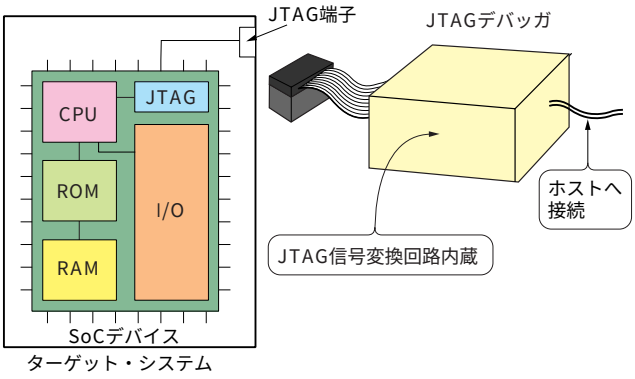


図10　JTAGデバッガの構成

写真3　JTAGデバッガの例〔(株)コンピューテックス　PALMiCE2 http://www.computex.co.jp/〕

ン・メモリを内蔵している点もフルICEに近い特徴です．

しかし，内部構造はROMモニタ型デバッガに近い構造をしているため，ローカル・バスに問題があるターゲット・システムでは，デバッガがうまく動作しない場合があります．

(4) JTAGデバッガ

JTAG端子を使ってCPUをデバッグするものです．**コラム6**に説明するような状況から，最近になって登場してきたデバッガです．

図10にJTAGデバッガの構成を，**写真3**にJTAGデバッガの例を示します．JTAG端子の信号はそのままではホストに接続できないので，一般的にはJTAG信号をEthernetやUSBなどに変換する回路が必要になるので，ICEやROMエミュレータ型デバッガと同じように，ターゲット・システムの外部に箱を置く形になります．

一般的にJTAGデバッガは，CPUの動作を直接内部から制御するので，ICEと同様にローカル・バスに問題があるターゲット・システムでも，メモリやI/Oを読み書きして，その原因を突き止めることができます．しかし，エミュレーション用のメモリは内蔵していないので，プログラムをダウンロードするには，ターゲット・システム上のメモリが正常に読み書きできている必要があります．

ROM領域の扱いはROMモニタ型デバッガに近いところがあります．フラッシュ・メモリに対応していれば，ROM領域にダウンロードすることも可能です．ROM領域へのブレーク・ポイントも，CPUが内蔵するハードウェア・ブレーク機能を使って実現しているJTAGデバッガが多いようです．

● 最強デバッガ＝エミュレーション・メモリ＆トレース・メモリ内蔵JTAGデバッガ？

ICEやROMエミュレータ型デバッガ，JTAGデバッガは，いずれもターゲット・システムの外に変換回路などのための箱を置き，ホストと接続する必要があります．そのため，どうしてもコストがかかります．一般的なコスト比較では，エミュレーション・メモリやCPU機能をまるまる内蔵したICEがもっとも高く，次にエミュレーション・メモリを内蔵したROMエミュレータ型デバッガになります．この中で比較的安価なのはJTAGデバッガといえます．

ROMモニタ型デバッガは，ターゲット・システム上のリソースを使ってホストと通信するので，デバッガとして必要な部品代はかかりません．しかし，デバッガとしての機能は，ほかの3種類のデバッガの一歩も二歩も後を行きます．

最近では，それぞれのデバッガの良いところを組み合わせたデバッガとして，ROMエミュレーションのためのエミュレーション・メモリや，トレース機能のためのトレース・メモリを内蔵したJTAGデバッガが登場しています．

やまぎわ・しんいち

◆COLUMN7　ICEとエミュレータ

組み込みシステムの開発環境を語るうえで，ICEやエミュレータ，デバッガなどの用語は，多少混乱した使われ方がされているようです．

メーカによっては，フルICEデバッガもJTAGデバッガも，使用目的やできる機能が同じということで，広義の意味で“ICE”とひとくくりにしているところもあります．またICEのE(Emulator)だけを取り出して，昔からICEのことを“エミュレータ”と呼ぶメーカもあります．そのため，JTAGデバッガが登場した現在でも，JTAGデバッガをエミュレータと呼んだり，場合によってはROMモニタ型デバッガまでエミュレータに分類していることもあります．

デバッガとしてのエミュレータという呼び方は，人によってイメージが異なるので，注意が必要です．