いまさら聞けない組込み入門講座 その①

組込み技術基礎の基礎

中村 憲一
アップウィンドテクノロジー・インコーポレイテッド

100年に1度と呼ばれる世界レベルでの経済不況のさなか、ものづくり日本の再生を望む声が少なくない。ものづくりのための技術は数知れないが、そのなかでも今特に日本の組込み技術が注目を集めている。そこで、今月から数回にわたり、組込み技術の基礎について解説する。

組込み技術と言っても幅広い分野で様々な技術が存在するため、それらを整理するためにまず、独立行政法人情報処理推進機構ソフトウェア・エンジニアリング・センター(IPA/SEC)が定めた「組込みスキル標準」(ETSS:Embedded Technology Skill Standard)を紹介する。IPA/SECによれば、「スキル基準は、組込みソフトウェア開発に必要なスキルを明確化・体系化したものであり、組込みソフトウェア開発者の人材育成・活用に有用な指標(共通基準)を提供するものである。」とのことである。表1にETSSのスキルフレームワークの例を示す。まず、3つのスキルカテゴリが設けられており、「技術要素」には組込みシステム自体に組み込まれシステムの機能を実現する技術項目を、「開発技術」には組込みシステムに各種技術要素を実装するために開発時に使用する技術項目を、「管理技術」には組込みシステム開発を円滑かつ的確に進行させるために使用する技術項目を分類している。そして、分野(ドメイン)毎にスキルを細分化するための複数の階層とその能力を示す4つのレベルが設けられている。

レベル1(初級)は、支援のもとに作業を遂行できることを、
レベル2(中級)は、自律的に作業を遂行できることを、
レベル3(上級)は、作業を分析し改善・改良できることを、
レベル4(最上級)は、新たな技術を開発できることを示す。つまり、技術革新(イノベーション)を推進できる能力があることを示す。

この細分化された分野がドメインであり、半導体・自動車・通信・家電・OA機器など業種や製品によって様々なドメインが存在するため、すべてのドメインでスペシャリストを目指すのは天才でない限り不可能である。よって、ドメイン毎にスキルを「見える化」することが重要である。そうすることにより、自分は何が得意なのか?何を学ぶ必要があるのかを知ることができる。ETSSは、IPA/SECのホームページからダウンロードすることができるので、是非一度目を通しておいていただきたい。

さて、前置きが長くなってしまったが、組込み技術の基礎の基礎とは何だろうか。プロセッサ、メモリ、バス、ハードディスク、OS、ミドルウェアなどコンピュータシステムを構成する様々な技術が思い浮かぶだろう。しかし、筆者は、それらの根幹をなすものは中学校で習った「オームの法則」であると考える。なぜ、電流が流れるのか?電圧や抵抗との関係はどうなのか?第三者に簡単にわかりやすく説明できるだろうか?たとえば、懐中電灯に電池を入れてライトを点灯させるのは誰もが出来るだろう。図1にこの回路を示す。たとえば、1.5Vの電池を2個直列に接続し、0.3Aの電流が流れたとすると、豆電球の抵抗値は何Ωだろうか?抵抗(R)=電圧(E)/電流(I)なので、3/0.3=10Ωとなる。(豆知識:電球が高温になるとオームの法則は成立しなくなる。)

では、秋葉原で購入したLEDを安定して点灯させ続けることのできる技術者はどれだけいるだろうか?もちろん、ハードウェアを専門とする組み込み技術者ができなければ論外だが、組込みソフトウェア技術者の中には出来ない人が多い。というのも、プログラミングさえできれば、いや出来なくても組込みソフトウェア技術者にはなることができるからだ。実際、理系出身者は限られているため、文系出身のプログラマやSEが多いのが現実である。

では、LEDを安定して点灯させるには、どうすればよいのだろうか?それには、仕様書を読みこなすスキル、電源に関する知識、電子回路を設計できるスキル等が必要となる。つまり、LEDの仕様書を読んでその特性を理解し、何アンペアの電流を流すべき

スキルカテゴリ	第1階層	第2階層	第3階層	レベル1 初級	レベル2 中級	レベル3 上級	レベル4 最上級
技術要素	通信	有線	WAN	■			
			LAN	■			
		無線	電気通信事業用無線	■			
			一般業務用無線				
		放送	デジタル放送				
			アナログ放送				
		インターネット	IPv4	■	■		
			IPv6	■			
		・・・	・・・	■			
	情報処理	情報入力	数値入力	■	■	■	
			文字列入力	■			
		セキュリティ	暗号	■			
			著作権保護	■			
		データ処理	数値演算	■	■	■	
			文字列処理	■	■		
		情報出力	数値出力	■	■	■	
			文字列出力	■	■		
		・・・	・・・	■			
	・・・	・・・	・・・	■			
開発技術	ソフトウェア詳細設計	ソフトウェアの詳細設計	・・・	■	■		
		ソフトウェアの詳細設計のレビュー	・・・	■			
		・・・	・・・	■	■		
	・・・	・・・	・・・	■			
管理技術	プロジェクトマネジメント	コストマネジメント	・・・				
		品質マネジメント	・・・	■			
		リスクマネジメント	・・・				
	・・・	・・・	・・・				

表1.ETSSのスキルフレームワークの例

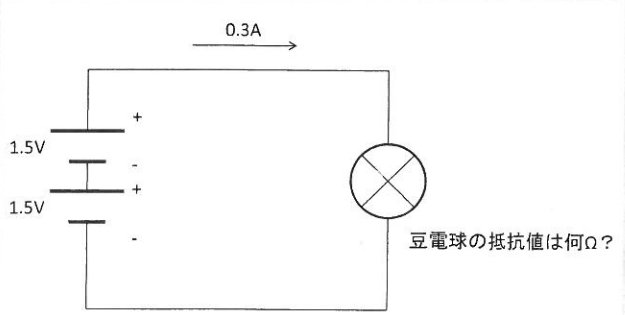


図1.単純な豆電球の回路

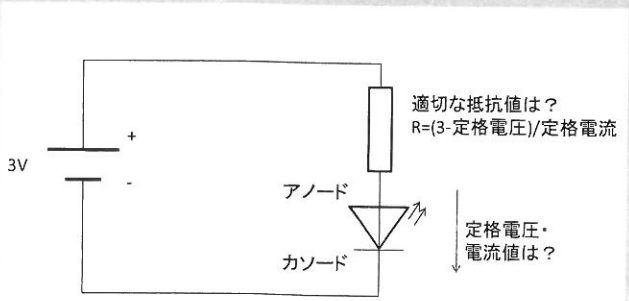


図2.単純なLEDの回路

かを計算する。そして、その電流を流すためには、何オームの抵抗をどこに配置し、何ボルトの安定化電源が必要かを決定しなければいけない。これは、もっとも単純な例であるが、これこそが組み込み技術の基礎の基礎であるといえる。図２に豆電球をLEDに置き換えた回路を示す。豆電球と異なり、LEDでは抵抗が必須であるため、LEDにとって適切な値の抵抗を挿入しなければいけない。つまり、LEDの仕様書から定格電圧と定格電流を読み取り、そのようになる抵抗値を計算する必要がある。たとえば、LEDの定格電圧が2V、定格電流が0.1Aとすると、何Ωの抵抗を挿入すればよいのだろうか？抵抗にかかる電圧は(3-2)=1V、流れる電流は0.1Aなので1/0.1=10Ωとなる。ゆえに、「オームの法則」を知っていれば、回路図から電圧・電流・抵抗の関係を知ることができる。

さらに、図2の回路図では、LEDの上に抵抗を挿入しているが、LEDの下に挿入すると何が違うのだろうか？興味のある読者は是非自分で調べていただきたい。キーワードは、プルアップ抵抗とプルダウン抵抗である。組込みソフトウェア技術者は、ハードウェアを扱うため、時には回路図を見ながらデバッグする機会も多い。そんな時には、きっと「オームの法則」が役に立つはずである。

次に、このLEDを何らかのプロセッサを使用して自由に制御したいとしたら、どうすればよいのだろうか？それには、電子回路のスキルに加えて、組込みソフトウェアのスキルが必要となる。次回は、組込みソフトウェアの基礎の基礎について解説する。

いまさら聞けない組込み入門講座 その②

組込みソフトウェアの基礎の基礎

中村 憲一
アップウィンドテクノロジー・インコーポレイテッド

前回は、ETSSに始まり、オームの法則とLEDの点灯回路について解説した。LEDを制御するためには、電子回路のスキルに加えて組込みソフトウェアのスキルが必要となる。今回は、組込みソフトウェアの基礎の基礎について解説する。

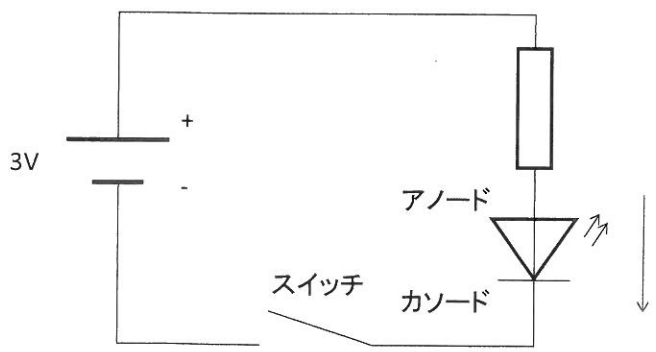
正論理と負論理

前回示したLEDの点灯回路にスイッチを追加すると、スイッチがONのときにLEDが点灯し、OFFのときに消灯する。(図１)このスイッチをプロセッサに置き換えてみよう。(図２)そうすると、LEDに接続される信号線の電圧がLoのときに点灯し、Hiのときに消灯することがわかる。これを負論理という。なお、これとは逆に信号線の電圧がHiのときに対象物が動作する論理を正論理という。負論理回路の場合、制御対象物はプルアップされており、信号線の電圧はHiに保たれる。(前回の宿題:プルアップ、プルダウンについて調べましたか?)つまり、プロセッサの入出力ポート(I/Oポートという)にLoを出力すれば、LEDを点灯させることができる。

論理回路の基礎

論理回路を示すために使われるのが論理ゲートであり、基本は、論理積(AND)、論理和(OR)、否定(NOT)の３つである。また、これらを組み合わせたゲートに、排他的論理和(XOR)、否定論理和(NOR)、否定論理積(NAND)などがある。なお、入力と出力の関係を表にしたものを真理値表という。(表１)そして、これらのゲートをまとめて集積したICを汎用ロジックICと言い、米テキサスインスツルメント社の7400シリーズ(俗に74シリーズと呼ば

図1. 単純なLEDの制御回路



ONのときLEDが点灯
OFFのときLEDが消灯

れる）や米RCA社（当時）の4000シリーズが有名である。これらの論理ゲートの塊を複雑に組み合わせると演算ができるようになる。それがプロセッサである。74シリーズや4000シリーズを組み合わせれば、4bitくらいの単純なプロセッサであれば、個人でも製作することができるので興味のある方はぜひ試していただきたい。（参考文献を参照）ちなみに、ARM社のCortex-M3プロセッサの回路規模は36,000～60,000ゲートであるので同規模のプロセッサを74シリーズだけで製作するのは困難である。そのような場合は、FPGAと呼ばれるデバイスを使用すると解決できるので、興味のある方はFPGAについて調べてみていただきたい。

0と1の世界

これまでの説明で、論理回路では0と1しか表現できないことがわかっただろうか。つまり、組込みソフトウェアも0と1ですべての数を表現する。これを2進数という。しかしながら、プログラミングの世界では便宜上2進数よりも16進数が使われる。（表2）特に組込みソフトウェアでは、アドレスやビットの状態を16進数で表現するため、不慣れな方は早く慣れていただきたい。

リトルエンディアンとビッグエンディアン

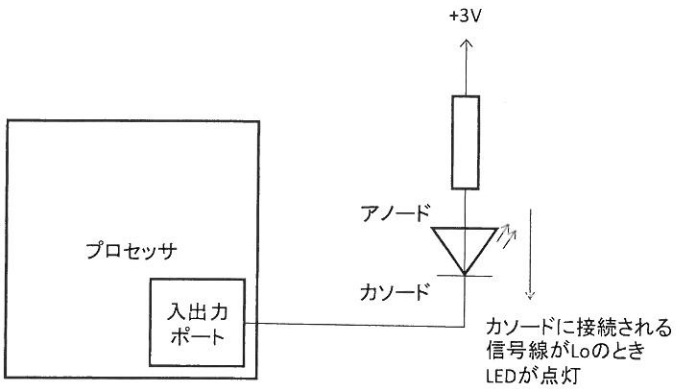
組み込みソフトウェアでは、16進数の値をメモリに格納する方法が2通り存在する。それが、リトルエンディアンとビッグエンディアンである、たとえば、12345678という値は、ビッグエンディアンでは12,34,56,78という順で格納されるが、リトルエンディアンでは78,56,34,12と逆の順で格納される。IA-32などのプロセッサでは、リトルエンディアン形式しかサポートされていないが、組込み用途のプロセッサでは両方の形式がサポートされる。ちなみに、ネットワーク等の通信用途では仕様でビッグエンディアン形式でデータ通信を行うと定められているため、IPアドレスなどのデータがリトルエンディアン形式で格納されている場合、通信時にデータをビッグエンディアンに変換する必要がある。

機械語、アセンブリ言語って何？

では、プログラムをプロセッサに実行してもらうためにはどうすればよいだろうか。プロセッサも論理回路であるため、実際は0と1の羅列である機械語しか理解することが出来ない。たとえば、プロセッサのI/Oアドレス0x40000000番地に接続されているLEDを制御するプログラムをARMプロセッサの機械語で書くとリスト1のようになる。このように、16進数でプログラムを書くのは非常に困難であることがわかる。（大昔は、これが当たり前だったが・・・。）よって、人間が理解できる言語として低級言語であるアセンブリ言語が生まれた。アセンブリ言語で書くとリスト2のようになり、プログラムを書きやすくなることがわかる。

もちろん、このままではプロセッサは理解できないので、アセンブラと呼ばれるプログラムを使用して機械語に翻訳（アセンブルという）するのである。しかし、アセンブリ言語は、プロセッサのアーキテクチャに依存するため移植性がないのが問題である。

図2. プロセッサを使ったLEDの制御回路



--- リスト1、ARMプロセッサの機械語のプログラム例 ---

```
0:  e3a03101
4:  e3a01001
8:  e3a02000
c:  e5831000
10:  e5832000
14:  eafffffe
```

よって、この問題を解決するためにC言語に代表される高級言語が生まれた。つまり、C言語でリスト3のようなプログラムを書けば、コンパイラと呼ばれるプログラムを使用して翻訳(コンパイルという)さえすれば基本的にどのプロセッサ上でも実行できる。基本的にというのは、プロセッサのI/Oアドレスを直接読み書きするようなプログラムの場合はアドレスを書き換えなければいけないためである。

単に一つのLEDの制御だけではなく、複数のスイッチの状態やセンサの値を取得しながら、複数のモータを制御したいというような場合は、どうすればよいのだろうか?それには、OSやソフトウェア設計などのスキルが必要となる。次回は、開発環境、OS、ミドルウェア、プラットフォームなどについて解説する。

表1. 論理ゲートの真理値表

論理積 (AND) 回路記号

入力A	入力B	出力Y
0	0	0
0	1	0
1	0	0
1	1	1

否定論理積 (NAND) 回路記号

入力A	入力B	出力Y
0	0	1
0	1	1
1	0	1
1	1	0

論理和 (OR) 回路記号

入力A	入力B	出力Y
0	0	0
0	1	1
1	0	1
1	1	1

否定論理和 (NOR) 回路記号

入力A	入力B	出力Y
0	0	1
0	1	0
1	0	0
1	1	0

否定 (NOT) 回路記号

入力	出力
0	1
1	0

排他的論理和 (XOR) 回路記号

入力A	入力B	出力Y
0	0	0
0	1	1
1	0	1
1	1	0

表2. 2のべき乗と10,2,16進数

2のべき乗	10進数	2進数	16進数
	0	0	0
0	1	1	1
1	2	10	2
	3	11	3
2	4	100	4
	5	101	5
	6	110	6
	7	111	7
3	8	1000	8
	9	1001	9
	10	1010	A
	11	1011	B
	12	1100	C
	13	1101	D
	14	1110	E
	15	1111	F
4	16	10000	10
	・	・	・
7	128	10000000	80
	・	・	・
8	256	100000000	100
	・	・	・
9	512	1000000000	200
	・	・	・
10	1024	10000000000	400
	・	・	・

---- リスト2. ARMプロセッサのアセンブリ言語のプログラム例 ----

```
.text
.global main
main:
mov   r3, #0x40000000
mov   r1, #1
mov   r2, #0
str   r1, [r3]/* 0x4000000番地のビット0に1を書き込み(LED点灯) */
str   r2, [r3]/* 0x4000000番地のビット0に0を書き込み(LED消灯) */
b     main
```

---- リスト3. C言語のプログラム例 ----

```
# define IO_ADDR (*(volatile unsigned int*)0x40000000)
int main(void){
  volatile int i;
  while(1){
    IO_ADDR = 0x1;  /* 0x4000000番地のビット0に1を書き込み(LED点灯) */
    i=1000000;
    while(i>0) i--; /* 点灯状態を保持するためのループ */
    IO_ADDR = 0x0;  /* 0x4000000番地のビット0に0を書き込み(LED消灯) */
    i=1000000;
    while(i>0) i--; /* 消灯状態を保持するためのループ */
  }
}
```

参考文献:「CPUの創りかた」、渡波 郁 (著)、毎日コミュニケーションズ、2,940円(税込)

いまさら聞けない組込み入門講座 その③

組込みソフトウェアの基礎の基礎

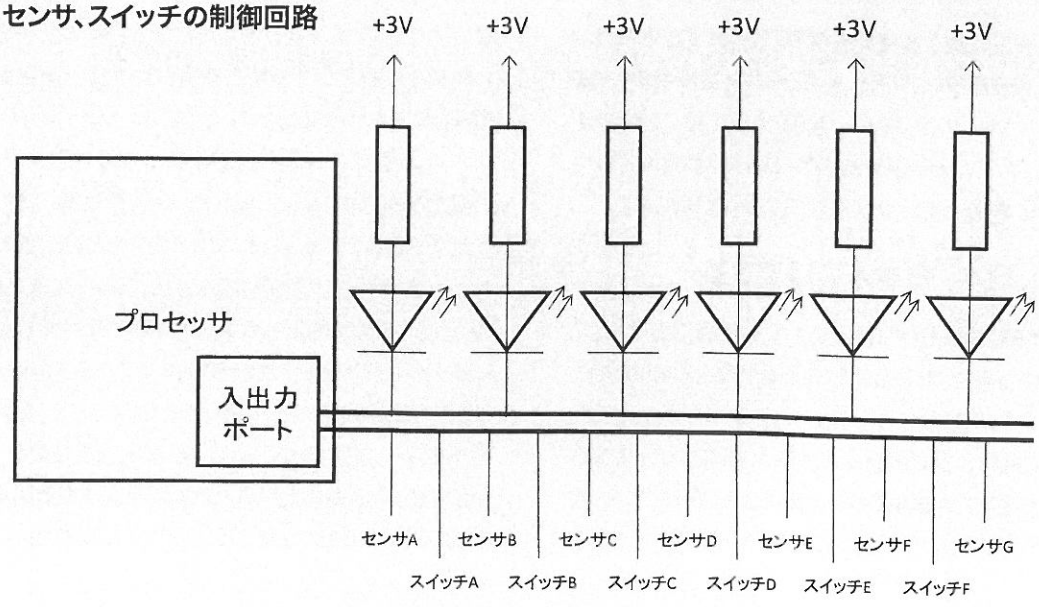
中村 憲一
アップウィンドテクノロジー・インコーポレイテッド

前回は、組込みソフトウェアでLEDを制御する方法について解説した。しかし、単に一つのLEDの制御だけではなく、図1のように複数のスイッチの状態やセンサの値を取得しながらLEDを制御したいというような場合は、OSやソフトウェア設計などのスキルが必要となる。今回は、開発環境、OS、ミドルウェア、プラットフォームなどについて解説する。

クロス開発環境とは

Windows上で動作するアプリケーションソフトウェアを開発するためには、Windows上で動作するWindows用のコードを出力するコンパイラとアセンブラ、そしてリンカを用意すればよいが、Windowsのアプリケーションバイナリインターフェース(ABI)に従ったEXE形式のファイルを出力しなければいけない。これをネイティブ開発環境という。同様に、組込み向けプロセッサ上で動作するソフトウェアを開発するためには、それぞれのプロセッサに適したファイルを出力する必要がある。たとえば、ARMプロセッサの場合は、EABIに従ったELF形式のファイルを出力しなければいけない。しかし、組込み向けプロセッサ上でコンパイラを動作させるのは、使用メモリ等のリソースが乏しく、パフォーマンスが悪いなど非効率なため、ネイティブ開発は事実上困難である。よって、Windows上で動作する組込みプロセッサ向けのコードを出力するクロスコンパイラやクロスアセンブラ等の開発環境が使用される。これをクロス開発環境という。そして、出力されたELFファイルを、S-レコード形式やバイナリ形式に変換し、ターゲットボード上のROMに書きこむ(一般にROMを焼くという)。その後、ターゲットボードをリセットしてプログラムの実行を確認するが、ROMに書き込んで一発で動作することはまずないため、通常、ICEと呼ばれる機器やデバッグ用のソフトウェア(デバッガという)を用いてデバッグ作業を行う。なお、最近では、JTAGポートが用意されているボードが増えたため、これを利用することにより、面倒なファイル形式の変換やROM焼き作業を省略し、ELF形式のファイルをダウンロードして、すぐに実行・デバッグすることが可能になった。

図1 LED、センサ、スイッチの制御回路



リスト1 タスクを使用したプログラム例

```
#include<itron.h>

# define IO_ADDR (*(volatile unsigned int*)0x40000000)

void task_a(VP_INT stacd)
{
  for(;;)
  {
    wai_sem(1);
    IO_ADDR = 0x1;  /* 0x4000000番地のビット0に1を書き込み(LED点灯) */
    sig_sem(1);
  }
}

void task_b(VP_INT stacd)
{
  for(;;)
  {
    wai_sem(1);
    IO_ADDR = 0x0;  /* 0x4000000番地のビット0に0を書き込み(LED消灯) */
    sig_sem(1);
  }
}
```

さらに、Windows等のGUI環境が普及した結果、コマンドラインでの操作を苦手とする技術者も増えたことから、現在ではEclipseのような統合開発環境の利用も当たり前になっている。

リアルタイムOSとは

入出力や制御対象の数が多い場合やリアルタイム処理を必要とする場合、アセンブリ言語やC言語でゼロ(スクラッチという)からプログラムを設計するのは非常に難しい。このようなときに使用するのが、リアルタイムオペレーティングシステム(以下RTOS)である。RTOSは、タスクとして登録された関数を決められた時間内に実行することをほぼ保証するものであり、携帯電話の電話機能のように遅延が許されないシステムやロボットのモータ制御など厳しい制御が要求される分野で広く採用されている。たとえば、LEDの制御を点灯と消灯の2つのタスクに分割するとリスト1のようになる。そして、RTOSのタスクスケジューリングに機能より点灯タスクと消灯タスクが交互に実行される。

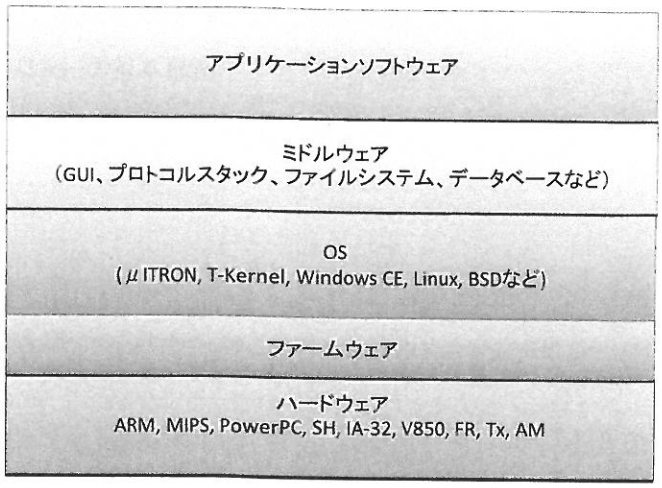
日本では、1984年に東京大学の坂村健博士によって開始されたトロンプロジェクトの成果であるμITRON仕様が事実上の標準として広く採用されている。なお、現在は、μITRONからT-Kernelに移行しつつある。

また、リアルタイム処理が不要なシステムでは、GNU/Linuxシステムを組込みシステムにも適用した組込みLinuxの採用も盛んである。

ミドルウェアとは

計測データをファイルに記録したり、データをネットワーク経由で送受信したりといったことをRTOSだけで行おうとすると、ファイルシステムやネットワークプロトコルスタック、ネットワークサーバーやクライアントソフトウェアなど莫大な量のソフトウェアを作成しなければいけない。このようなときに使用するのがミドルウェアである。ミドルウェアは、RTOSとアプリケーションソフトウェアの中間に位置するもので、アプリケーションプログラムからはミドルウェアのAPIを呼び出せば良いだけであるため、非常に便利である。(図2)

図2 ソフトウェアの構造



プラットフォームとは

最近の組込みシステムでは開発期間の短縮化や品質の確保などを目的に、μITRONやT-Kernel等のRTOSと豊富なミドルウェアを組み合わせたプラットフォームが製品や分野別に構築されている。また、ネットワーク家電などでは、ネットワーク関連の機能が豊富なGNU/Linuxシステムを採用したプラットフォームが主流である。最近では、米Google社からAndroidと呼ばれる携帯電話向けのプラットフォームもオープンソースで公開されている。そして、日本国内においてもこのAndroidを採用した携帯電話が次々と発売されており、話題になっている。また、携帯電話のメーカー以外からも注目され、携帯電話以外での利用を推進する団体も設立されている。このように、現在では製品開発にはプラットフォームの構築と活用が欠かせなくなった。

それでも、品質の高いシステムを短期間で、そして安く開発するためには、効率的な開発手法やプロセスの導入、ソフトウェアの部品化と再利用などソフトウェアエンジニアリングの知識が欠かせない。次回は、ソフトウェアエンジニアリングについて解説する。

参考文献:「μITRON 4.0仕様 Ver. 4.03.03」、坂村健　監修、(社)トロン協会編集/発行、5,000円(税込)

いまさら聞けない組込み入門講座 その④

組込みソフトウェアの基礎の基礎

中村 憲一
アップウィンドテクノロジー・インコーポレイテッド

前回は、開発環境、ミドルウェア、OSについて、それらを活用する方法について解説した。しかし、実際の製品ではLEDを制御するといった単純な回路は少ない。携帯電話、カーナビ、薄型テレビ、ビデオレコーダー、ネット家電など用途に応じて様々なハードウェアが存在する。また、国内の携帯電話だけを見ても各キャリア・各メーカーから様々な機種つまりハードウェアが発売され、春夏秋冬と季節毎に頻繁なモデルチェンジやマイナーチェンジが繰り返されている。このように開発するモデルが増えてくると、ハードウェアが変わるたびに新たに設計・開発を行っていたのでは莫大な数の開発者が必要となり、それに比例してコストがかかるという問題がある。

プラットフォームとは

最近の組込みシステムでは開発期間の短縮化や品質の確保などを目的に、μITRONやT-Kernel等のRTOSと豊富なミドルウェアを組み合わせたプラットフォームが製品や分野別に構築されている。一般的には、OSやミドルウェアのレベルで抽象化し、ハードウェアやOSが異なっても以前開発したアプリケーションを流用できるようになっている。(図1，2)これにより、開発者は新規に開発しなければならないアプリケーションの開発に専念することができる。このように、現在では製品開発

図1
プラットフォームの例（OSによる抽象化）

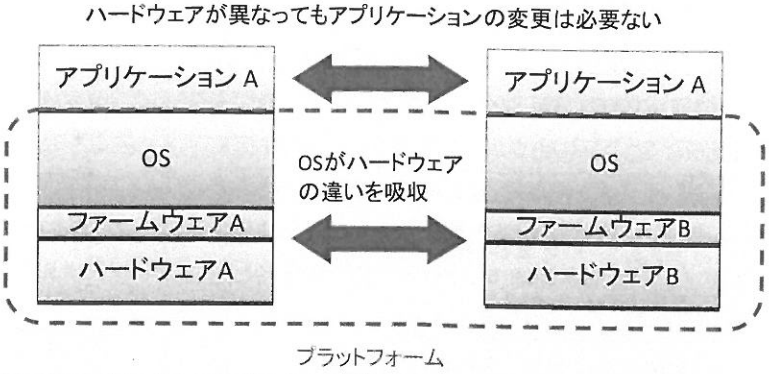


図2
プラットフォームの例（ミドルウェアによる抽象化）

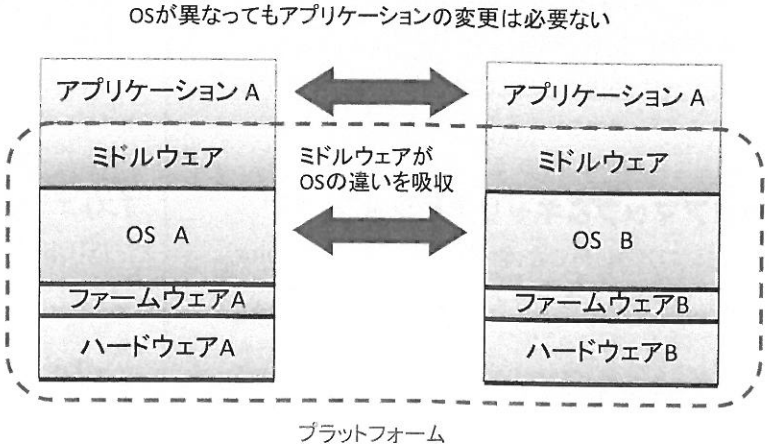
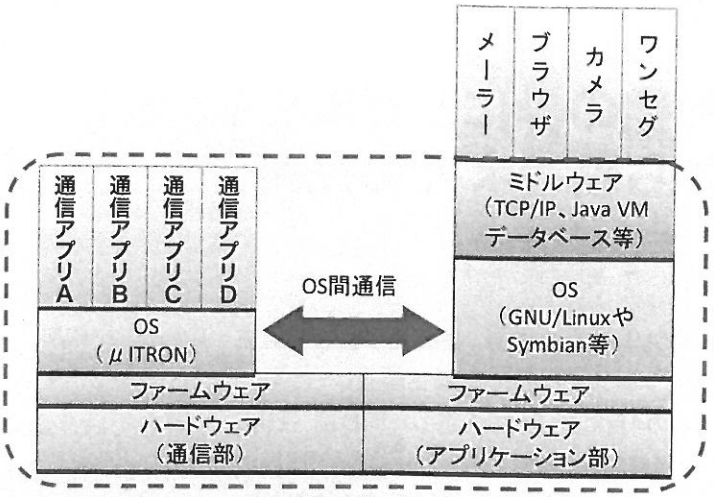


図3
プラットフォームの例（携帯電話の例）



にはプラットフォームの構築と活用が欠かせなくなったが、次に、いくつかの製品のプラットフォームについて解説する。

デジタル家電のプラットフォーム

今やデジタル家電もネットワークに接続できるのが当たり前になった。そのため、ネットワーク関連の機能が豊富なGNU/Linuxシステムを採用したプラットフォームが主流である。これは、GNU/Linuxシステムがあらかじめ TCP/IPやUSB等のプロトコルスタックや、メーラーやブラウザ等を備えており、製品メーカーはほとんどミドルウェアを購入する必要がないことが大きい。また、開発環境もGNUコンパイラコレクション(GCC)等のフリーソフトウェアが用いられるため、テレビのアプリケーションソフトウェアを自分で開発できる開発キットを公開しているメーカーも存在する。

携帯電話のプラットフォーム

一般的に携帯電話は通信部とアプリケーション部から構成されている。通信部は、無線により音声などのデータの送受信を行い、アプリケーション部は、グラフィカルユーザーインターフェース、音楽録音・再生、ビデオ録画・再生、カメラ、ワンセグ機能などを提供する。よって、通信部はハードリアルタイム制御が求められるためμITRON仕様または他社のリアルタイムOSが、アプリケーション部はGNU/LinuxまたはSymbianという構成が典型的である。しかし、さらなる開発期間の短縮化や品質の確保およびコスト削減要求により、各メーカーが独自にプラットフォームを構築し、メンテナンスすることすら難しくなってきた。そこに登場したのがAndroidである。Androidは米Google社が開発した携帯電話向けのプラットフォームであり、オープンソースで公開されている。日本でも、Androidを採用した携帯電話がすでに各キャリアから発売されており人気を集めている。また、携帯電話メーカー以外からも注目され、携帯電話以外での利用を推進するためにOpen Embedded Software Foundationという団体も設立されている。

自動車のプラットフォーム

デジタル家電や携帯電話と同様に、自動車においても機能が増え続けている。エンジン制御、エアバッグ制御、ブレーキ制御等に加え、ミリ波レーダー、カメラ、カーナビ、エアコン、セキュリティ、バッテリー、モーター等が協調制御されるようになった。その結果、ECU(電子制御ユニット)の数もワイヤーハーネスの重量も増え、開発・製造コストも飛躍的に増大した。そこで、ECUやワイヤーハーネスの数を減らすために、パワートレイン系、安全系、セキュリティ系、マルチメディア系など系毎に統合制御を実現するための電子プラットフォームの開発が盛んである。

プラットフォームだけでは不十分

プラットフォームを構築しても、それだけでは、品質の高いシステムを短期間で、そして安く開発することはできない。そのためには、効率的な開発手法やプロセスの導入、ソフトウェアの部品化と再利用などソフトウェアエンジニアリングの知識が必要となる。次回は、ソフトウェアエンジニアリングについて解説する。

参考文献:「エンベデッドプラットホームにおける動向調査」、JASAプラットフォーム研究会、2003年3月、(社)組込みシステム技術協会
「プラットフォーム・リーダーシップ」、アナベル ガワー、マイケル・A.クスマノ、2005年3月、有斐閣

いまさら聞けない組込み入門講座 その⑤

組込みソフトウェアの基礎の基礎

中村 憲一
アップウィンドテクノロジー・インコーポレイテッド

前回は、プラットフォームについて解説したが、品質の高いシステムを短期間で、そして安く開発するためにはそれだけでは足りず、ソフトウェアエンジニアリングの知識が必要であることを指摘した。今回は、ソフトウェアエンジニアリングの一部である効率的な開発手法やプロセスの導入、ソフトウェアの部品化と再利用等について解説する。

ソフトウェアエンジニアリングとは？

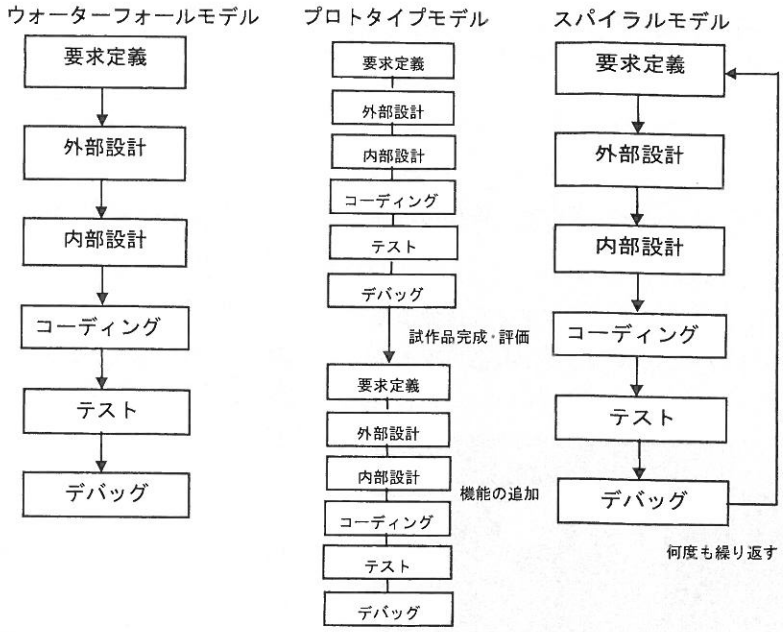
ソフトウェアエンジニアリングとはその名の通りソフトウェアに関する工学であり、ソフトウェアの開発・運用・保守に体系的・学問的・定量的手法を応用する学問である。ソフトウェア工学の歴史は古く、1968年には北大西洋条約機構(NATO)においてソフトウェア工学に関する講演会が開催されている。その後、米国では、1984年にCarnegie Mellon Software Engineering Institute(SEI)が設立され、日本でも経済産業省が2004年10月に(独)情報処理推進機構(IPA)にソフトウェアエンジニアリングセンター(SEC)を設立するなど最近注目されている学問である。しかしながら、ソフトウェア工学が情報工学の一分野であるせいか、残念ながら日本では、機械工学科、電気工学科、電子工学科、情報工学科、建設工学科、土木工学科等を有する大学は多いが、ソフトウェア工学科を有する大学はまだ少ないのが現状である。

表1 ソフトウェアエンジニアリング知識体系SWEBOK(Software Engineering Body Of Knowledge)

ソフトウェア要求	ソフトウェア要求の基礎
	要求プロセス
	要求抽出
	要求分析
	要求の仕様化
	要求の妥当性確認
	実践上考慮すべきことがら
ソフトウェア設計	ソフトウェア設計の基礎
	ソフトウェア設計における主要な問題
	ソフトウェア構造とアーキテクチャ
	ソフトウェア設計品質の分析と評価
	ソフトウェア設計のための表記
	ソフトウェア設計戦略および手法
ソフトウェア構築	ソフトウェア構築の基礎
	ソフトウェア構築のマネジメント
	実践上考慮すべきことがら
ソフトウェアテスティング	ソフトウェアテスティングの基礎
	テストレベル
	テスト技法
	テストに関係した計量尺度
ソフトウェア保守	ソフトウェア保守の基礎
	ソフトウェア保守における主要な課題
	保守プロセス
	保守のための技法

ソフトウェア構成管理	ソフトウェア構成管理のマネジメント
	ソフトウェア構成識別
	ソフトウェア構成コントロール
	ソフトウェア構成実態説明
	ソフトウェア構成監査
	ソフトウェアリリースマネジメントおよび引き渡し
ソフトウェアエンジニアリングマネジメント	始動および適用範囲の定義
	ソフトウェアプロジェクト計画
	ソフトウェアプロジェクト計画実施
	レビューおよび評価
	終結
	ソフトウェアエンジニアリング計量
ソフトウェアエンジニアリングプロセス	プロセスの実現および変更
	プロセス定義
	プロセス査定
	プロセスおよびプロダクト計量
ソフトウェアエンジニアリングのためのツールおよび手法	ソフトウェアエンジニアリングツール
	ソフトウェアエンジニアリング手法
ソフトウェア品質	ソフトウェア品質の基礎
	ソフトウェア品質マネジメントプロセス
	実践上考慮すべきことがら
ソフトウェアエンジニアリングに関連するディシプリン	コンピュータエンジニアリング
	コンピュータサイエンス
	マネジメント
	数学
	プロジェクトマネジメント
	品質マネジメント
	ソフトウェアエルゴノミクス
	システムエンジニアリング

図1
各開発手法の違い



SWEBOK

ソフトウエアの開発・保守に関しては、IEEEが母体のSWECC(SoftWare Engineering Coordinating Committee)が策定したSWEBOK(Software Engineering Body Of Knowledge)と呼ばれるソフトウェアエンジニアリング知識体系が有名であり、「ソフトウェア要求」「ソフトウェア設計」「ソフトウェア構築」「ソフトウェアテスティング」「ソフトウェア保守」の5項目からなる開発・保守の工程に沿った知識と、「ソフトウェア構成管理」「ソフトウェアエンジニアリングマネジメント」「ソフトウェアエンジニアリングプロセス」「ソフトウェアエンジニアリングのためのツールおよび手法」「ソフトウェア品質」「ソフトウェアエンジニアリングに関連するディシプリン」の6項目からなる開発・保守のどの工程にもかかわる横断的な知識の計11の領域からなる。(表1)

ソフトウェアエンジニアリング手法

たとえば、ソフトウェアエンジニアリングのためのツールおよび手法では、ソフトウェアエンジニアリングツールやソフトウェアエンジニアリング手法について整理されている。

ソフトウェアの開発手法で基本となるのが、ウォーターフォールモデル、プロトタイプモデル、スパイラルモデルの3つの手法である。(図1)ウォーターフォールモデルとは、要件定義、外部設計、内部設計、コーディング、テスト、デバッグと順に工程を進める開発手法であり、開発の流れが水が流れ落ちるように見えることからそのように呼ばれる。つまり、開発途中で前の工程に戻ることができないのが特徴であり、開発の途中で仕様変更や設計変更が行われないことが前提条件となる。

プロトタイプモデルとは、開発の初期段階で試作品(プロトタイプ)を開発し、その評価を行いながら開発を進める開発手法であり、開発の途中での仕様変更などにも柔軟に対応できるのが特徴である。

スパイラルモデルとは、ウォーターフォールモデルとプロトタイプモデルの良い点を採用した開発手法である。要件定義、外部設計、内部設計、コーディング、テスト、デバッグと順に工程を進めるのはウォーターフォールモデルと同じであるが、プロトタイプモデルのように小さな規模で何度も各工程を繰り返すのが特徴である。

プロトタイプモデルとスパイラルモデルは、開発の途中で前の工程に戻ることができるため、反復型の開発手法と呼ばれる。

また、最近では開発プロセスを重視し、反復型を前提としたアジャイルソフトウェア開発手法が注目されており、その手法のひとつにXP(eXtreme Programming)と呼ばれる手法がある。XPでは、将来の仕様変更にも対応できるようにソースコードの手直しを行うリファクタリングという方法で再設計を行うことによって、より洗練された設計に近づける。また、従来の開発手法では重要だったドキュメントを重視しない。よって、XPを採用するといかなる工程においても仕様変更に対応することが出来、品質の高いソフトウェアを開発することが出来ると考えられているのが特徴である。次回も引き続き、ソフトウェアエンジニアリングについて解説する。

いまさら聞けない組込み入門講座 その⑥

組込みソフトウェアの基礎の基礎

中村 憲一
アップウィンドテクノロジー・インコーポレイテッド

前回は、ソフトウェアエンジニアリングの一部である効率的な開発手法やプロセスの導入について簡単に説明した。今回も引き続き、開発手法、プロセス、ソフトウェアの部品化と再利用等について解説する。

様々な開発手法

ソフトウェアの開発手法には基本となるウォーターフォールモデル、プロトタイプモデル、スパイラルモデルの３つの手法があることを示したが、この他にも、オブジェクト指向開発(OOD)、モデル駆動開発(MDD)、テスト駆動開発(TDD)、ドキュメント駆動開発(DDD)など様々な開発手法が存在する。ここでは、オブジェクト指向開発について紹介する。

オブジェクト指向開発とは

オブジェクト指向開発ではその名の通り、オブジェクト(物)に着目してシステムのオブジェクト指向分析(OOA)およびオブジェクト指向設計(OOD)を行う。実在する物だけではなく、機能やデータもオブジェクトとして扱うため、開発途中における仕様変更などにも柔軟に対応できるのが特徴である。具体的な手法は、Booch法、OMT(Object Modeling Technique)法、OOSE(Object Oriented Software Engineering)法などがあるが、現在では、ラショナル統一プロセス(RUP)と呼ばれる手法に統合されており、UML(Unified Modeling Language)という記法が使用される。

UMLでは、必要に応じて、クラス図、オブジェクト図、パッケージ図、コンポジット構造図、コンポーネント図、配置図、ユースケース図、アクティビティ図、状態マシン図、シーケンス図、コミュニケーション図、相互作用概要図、タイミング図という13種類の図を作成する。クラス図はクラスの構造とクラス間の静的な関係を、オブジェクト図はシステムのある時点でのスナップショットを、パッケージ図はグループ化したモデルの要素を、コンポジット構造図はクラスの内部構造を、コンポーネント図は実装したコンポーネントの関係を、配置図は構成するハードウェアの関係を、ユースケース図はシステムの役割と外部との関係を、アクティビティ図はシステムの振る舞いを、状態マシン図はオブジェクトの状態遷移を、シーケンス図はオブジェクト間の時系列の相互作用を、コミュニケーション図はオブジェクト間の関係に着目した相互作用

図1 オブジェクト指向開発によるモデリングの例

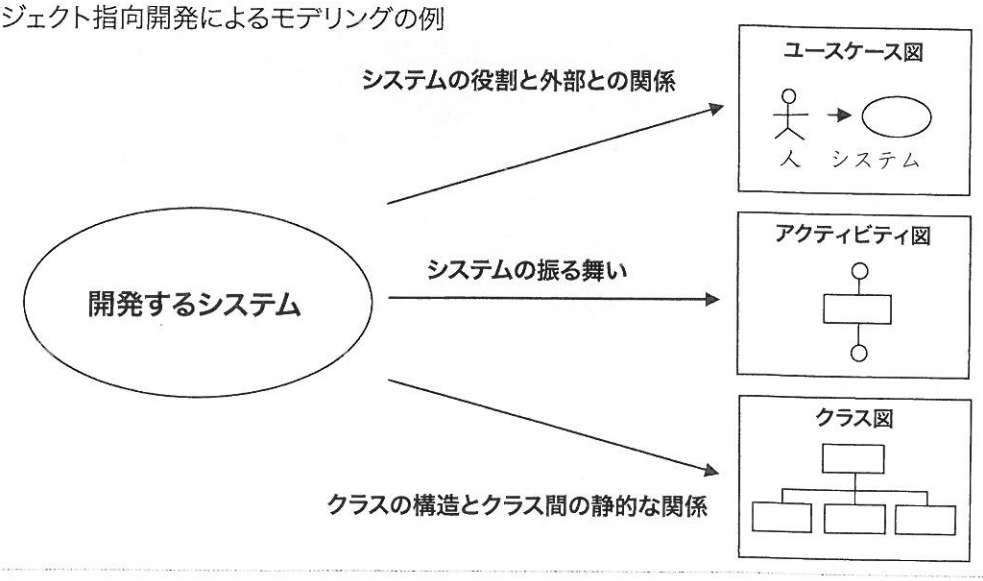
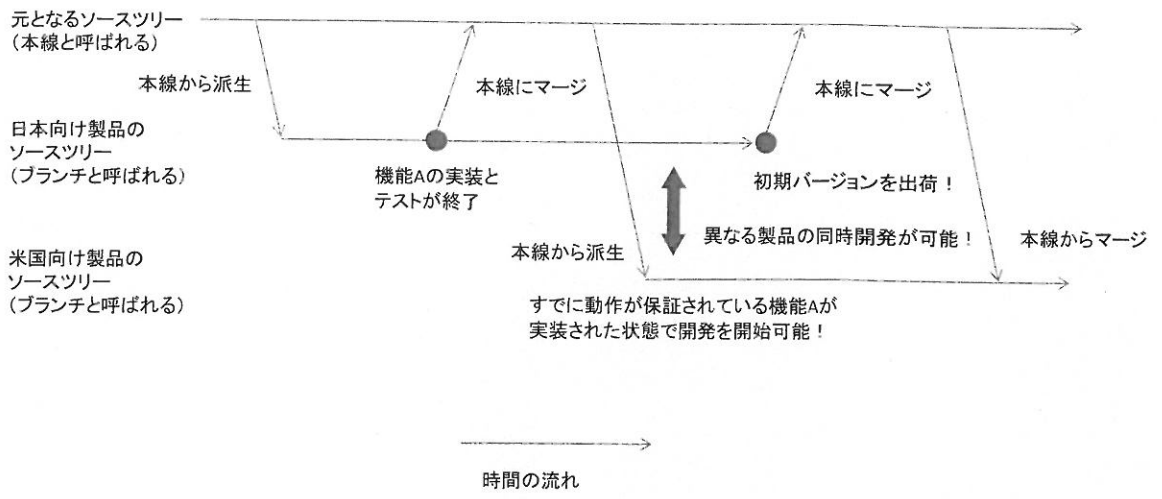


図2 ソースコードのバージョン管理の例



を、相互作用概要図は相互作用の実行順序を、タイミング図は相互作用と状態遷移に関する時間制約を表現する。そして、これらの図を作成することをモデリングという。(図1)

たとえば、携帯電話の場合、カメラ、地磁気センサ、加速度センサなどのように実在するものとしてオブジェクトが抽出できるが、カメラに着目するだけでも、写真のサイズ等のデータオブジェクト、ホワイトバランスの設定等の機能オブジェクトなど、様々なオブジェクトを抽出することができる。この作業をオブジェクト指向分析と言う。

ETロボコンとは？

(社)組込みシステム技術協会では組込みソフトウェア技術教育をテーマとした「ETロボコン」を主催している。ETロボコンでは、走行競技だけでなくモデリングの優劣も競う部門もあるので、新人技術者は腕試しに参加してみるのも良いだろう。そうすると、同じシステムのモデリングでも他のチームとの違いが見えるため、どんなモデリングが優れているのかを身をもって学ぶことができる。

構成管理とは？

品質の高いシステムを短期間で安く開発するためには、ソフトウェアの部品化と再利用を行うのが望ましい。しかし、部品化だけ進めても、部品のバージョン、部品同士の組み合わせの保証などが管理されていなければ意味がない。たとえば、同じ製品でも出荷する国によってはライブラリやフォントなどが異なる場合がある。また、出荷する国が同じでも、時間がたてばモデルチェンジなどでシステムを構成する各ソフトウェアのソースコードのバージョンが異なるのが通常である。よって、様々なバージョンや組み合わせが混在することになり、何のルールも決めずに管理しているとすぐに破たんしてしまうことが容易に予想される。このような問題を解決するのが、構成管理である。具体的な方法としては、用途、規模、開発地域、予算などに応じて、CVS、Subversion、Git、BitKeeper、ClearCase、などのバージョン管理システムや構成管理ツールが使用される。バージョン管理システムといっても単なるバージョン管理ではなく、過去に出荷したソフトウェアの構成を再現できたり、簡単に派生バージョンを作成することができるのが特徴である。(図2)

そして、構成管理を適切に行うことにより、ソフトウェアプロダクトラインやソフトウェアファクトリーの実現につながるのである。ここまでくれば、ソフトウェアエンジニアリングの上級者といえるだろう。

おわりに

以上のように、品質の高い製品を短期間で安く開発するためには、ソフトウェアエンジニアリングを駆使した開発が求められる。しかしながら、これはとても一朝一夕で実現できるものではない。また、ソフトウェアエンジニアリングは常に発展し続けるものである。よって、常に最新のソフトウェアエンジニアリングを実践するためには、ソフトウェアエンジニアリングに対する投資を決して惜しんではいけないだろう。

誌面の都合で十分な説明ができなかった部分もあったが、足りない点は自分で学習して補っていただきたい。本連載が組込みソフトウェア産業に従事する読者の糧になれば幸いである。

参考文献、URL：
・「その場でつかえるしっかり学べるUML2.0」、オージス総研オブジェクトの広場 編集部著、山内亨和監修、秀和システム、2006年2月
・ETロボコンのホームページ(www.etrobo.jp)