Vol.2014-SE-183 No.18
2014/3/20

データバインディング機構を利用した C プログラムの再構成手法の提案

谷光 大樹† 荻原 剛志†

オブジェクト指向によるソフトウェア開発において，クラス間の依存関係をコードから取り除く方法として DI（Dependency Injection）が知られている．本稿では，DI の手法を参考に，手続き型言語である C 言語で記述されているプログラムからモジュールの依存性に関わるコード部分を取り除き，実行時に依存関係を設定する手法について提案する．提案手法を用い，マイコンボード上で動作するプログラムの開発も行った．

A reconstruction method of C programs using a data binding mechanism

DAIKI TANIMITSU†1 TAKESHI OGIHARA†2

Dependency injection is known as a means that removes the dependency from classes in object-oriented software development. In this article, we propose a method that removes the dependency from source codes described in C. The dependency is injected into the program at runtime. Additionally, we show a test program using the proposed method, which can run on a single-board microcomputer.

1. はじめに

オブジェクト指向によるソフトウェア開発では，オブジェクト間の依存性を低減し，独立性の高いコンポーネントを作成する手法として DI（Dependency Injection：依存性の注入）[1][2][3]が知られている．クラス名を明示してインスタンス化を行うといったコードは，コンポーネント間の依存性を高めてしまう．DI では，このような依存関係の情報をソースコードとは別の設定ファイルに記述しておき，ソフトウェアの実行時に読み込んで具体的なオブジェクトを作成する．利用するオブジェクトを変更するには設定ファイルを書き換えれば良く，ソースコードを書き換える必要は無い．

本稿では，DI が目的とする「設定を利用から分離する」という原則を，手続き型言語である C 言語で実現するための一手法を示す．特に，C 言語が開発用言語として広く用いられている組込みソフトウェアでも利用可能であることを念頭に提案を行う．

ただし，C 言語は動的にモジュールを結合させる機能が十分ではないため，本手法ではデータバインディング機能を提供する coval[4]ライブラリを使用する．データバインディングは，オブジェクト指向言語で広く使用されており，関連のあるプロパティ同士を相互に結合(バインド)することにより，複数のオブジェクトを動的に関連付ける技術である．

本手法では，プログラムの各モジュールを，coval によって互いに関連づけられて動作するように記述しておく．ただし，どのモジュールのどの coval をバインドするのかという情報は，実装ファイルではなく，別に用意した設定ファイルに記述する．

以下では，オブジェクト指向における DI，データバインディングの概要について述べ，次に提案した手法について論じる．さらに提案手法を実装したソフトウェアの例を示し，提案手法の有効性について論じる．

2. DI（Dependency Injection）の概要

オブジェクト指向において，あるクラスが自身の担当する機能を実現するために他のクラスを利用している場合，両クラスの間には依存性（Dependency）があるという．クラス間の依存性が強いと，あるクラスを別のクラスに変更する場合に，そのクラスに関するコードを全て修正しなければならず，変更に伴うコストが大きいという問題があった．

DI では，利用したいオブジェクトを利用する側が生成するのではなく，外部から注入してもらうことで，この問題に対応している．DI を用いる場合，ソースコードにはオブジェクトを生成するコードの記述は行わず，代わりに設定ファイルを用意し，そこに利用したいオブジェクトについての情報を記述しておく．設定ファイルは，実行時に DI コンテナと呼ばれるコンポーネントに読み込まれ，DI コンテナがこの情報に基づいて生成したオブジェクトを利用する側に注入する．

DI を用い，利用したいクラスを選択する例を図 1 に示す．

† 京都産業大学
Kyoto Sangyo University

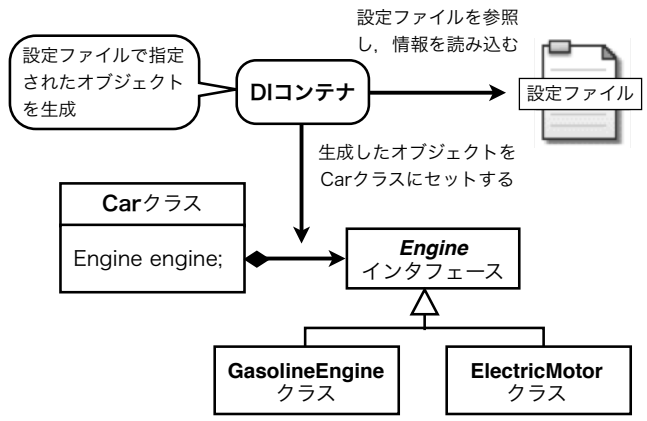

図 1　DI 使用時の依存関係

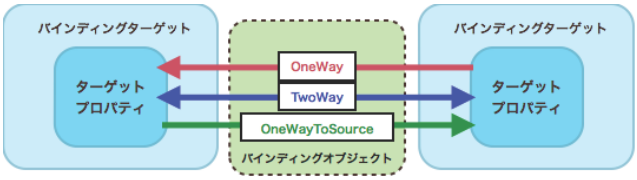

図 2　データバインディングの種類（文献[5]より）

Car クラスは Engine インタフェースを継承したクラスを利用することができ，その候補として GasolineEngine クラスと ElectricMotor クラスが用意されている．設定ファイルでは Car クラスで利用するクラス名を記述しておく．DI コンテナは実行時に設定ファイルを参照し，そこに記述された情報に基づいてオブジェクトを生成し，Car クラスにセットする．Car クラスは与えられたオブジェクトを利用して処理を行うため，自らオブジェクトの生成を行う必要はない．

3.　データバインディングの概要

3.1 オブジェクト指向におけるデータバインディング

データバインディングは C#，JavaFX，Objective-C などのオブジェクト指向言語で利用可能な技術である．ここでは Microsoft 社の Windows Presentation Foundation（WPF）[5]の解説を参照しながらデータバインディングの概要を説明する（図 2）．

データバインディングは，複数のオブジェクトでプロパティを共有しあい，プロパティの変更に伴って自動的に必要な手続きを起動させる仕組みである．バインディングによって共有される値を保持するプロパティをソースプロパティ，ソースプロパティのバインド対象となるプロパティをターゲットプロパティと呼ぶ．WPF には，どちらか一方のプロパティで起きた変更のみをもう一方に反映させる片方向のみのバインディング（OneWay，OneWayToSource）と，どちらのプロパティで起きた変更でも，もう一方へ反映する両方向のバインディング（TwoWay）が存在する．プロパティがアクセサ経由で参照，更新される場合，自動的にこれらの手続きも実行される．

3.2 coval の概要

coval は C 言語において，データバインディングと同様な機能を提供することを目指して開発されたライブラリである．coval は構造体として実装されており，値を格納する変数や，参照するためのポインタを備えている．また，それらを用いて同じ型のデータを保持する coval とバインドし，値を共有することができる．バインドされている全ての coval は共通の変数を参照するため，いずれかの coval が値を変更すると，他の coval の値も更新されたように見える．coval はバインドしていない状態では自身で保持している値を使用し，バインドしている状態では共有している値を優先的に使用する．そのため，普段は通常の変数として使用し，必要に応じて複数のモジュール間で値を共有するための手段として使用することができる．

coval にはコールバック関数を対応させることができ，バインド中に共有している値が更新されたタイミングで自動的に呼び出される．coval にコールバック関数が設定されていなければ，値が更新されても何も行われない．バインドされた coval のどちらか一方のみコールバック関数を設定すれば，3.1 節で述べた OneWay（あるいは OneWayToSource）の動作が実現でき，両方にコールバック関数を設定すれば TwoWay の動作を実現できる．coval 構造体の概念を図 3，4 に示す．

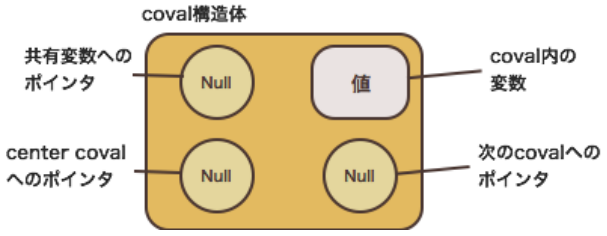

図 3　coval の概念図

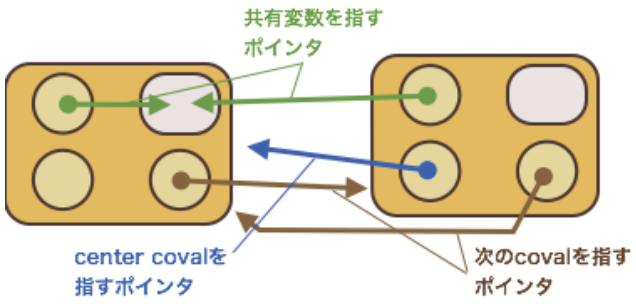

図 4　2 つの coval をバインドした時の概念図

4. 提案手法

4.1 手法の概要

本研究における提案手法は，オブジェクト指向における DI の手法を参考とし，モジュールの実装ファイルから機能の変更時に書き換えられる箇所を抽出し，その情報を実装ファイル外の設定ファイルに記述する．そして，実行時に設定ファイルの情報に基づいて動的にソフトウェアを構成することを目的としている．今回，変更箇所として抽出するのは，モジュールのインタフェースとしての役割を果たす coval をバインドさせるコードである．coval のバインドを行うコードを実装ファイルから取り除くことにより，モジュールの独立性の向上が見込める．

設定ファイルには，どの coval とどの coval をバインドさせるか，というバインドに関する情報を記述する．また，実行時にこの情報を読み込み，coval をバインドさせるルーチンを作成した．本稿では，このルーチンを builder と呼ぶことにする．

4.2 coval の識別方法

coval はプログラム内では，指定された型の値を保持する構造体である．coval を格納した 2 つの変数，a と b があるとき，これらをバインドさせるには次のように CovalBind という手続きでそれぞれへのポインタを指定する．

CovalBind(&a, &b);

C 言語にはリフレクションに相当する機能は存在しないため，設定ファイルの記述から実行時の coval 変数のアドレスを知り，バインドを実行させるための仕組みをあらかじめ用意しておく必要がある．そこで，coval を保持するモジュールが初期化処理を行う際に，自身の持つ coval への参照と識別子（文字列）をテーブルに登録し，builder はそのテーブルから対象となる coval を選択することとする．以下ではこの識別子を coval 名と呼ぶ． 図 5 は提案手法の概念を示している．

coval を実装するモジュールの定義の例を図 6, 7 に示す．図 7 の 11 行目では coval 名を表す char 型の配列の初期値を設定しているが，その文字列の定義は図 6 のヘッダファイルで行われている．coval 名をヘッダファイルで定義することによって，coval 名をモジュールのインタフェースの構成要素として扱うことができる．また，図 7 の 16 行目からはモジュールを初期化するための関数定義である 18, 19 行目はそれぞれ coval の初期化とコールバック関数の設定を行うためのマクロである．20 行目で呼び出されている insertCovalInfo () 関数はテーブルへ coval の情報の登録を行っている．

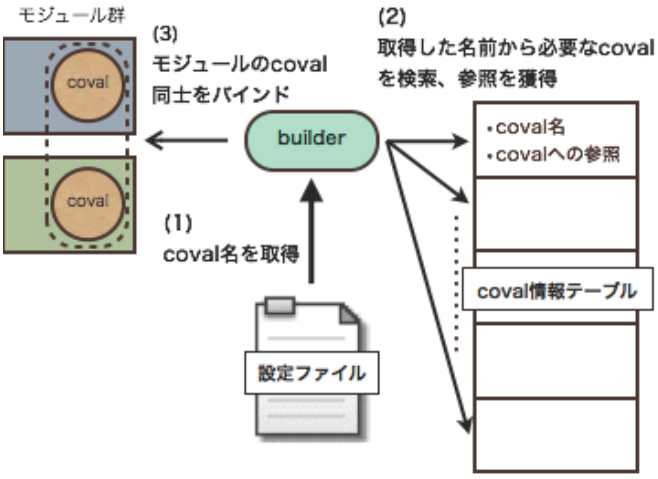

図 5 提案手法の概念図

```
 1 //module.h
 2
 3 #ifndef _MODULE_H_
 4 #define _MODULE_H_
 5
 6 #define COVAL_NAME_FOR_MODULE "cov_module"
 7
 8 void initModule ( void );
 9
10 #endif
```

図 6 モジュールのヘッダファイルの定義例

```
 1 //module.c
 2
 3 #include "module.h"
 4 #include "coval.h"
 5 #include "builder.h"
 6
 7 typedef CovalTypeOf( unsigned int ) uintCoval;
 8
 9 //covalの情報
10 uintCoval cov_module;
11 static const char cov_name[] = COVAL_NAME_FOR_MODULE;
12
13 //内部関数の宣言
14 static int action ( uintCoval *cov, void *arg );
15
16 void initModule ( void )
17 {
18     CovalInit( &cov_module );
19     CovalSetCallback( &cov_module, action, (void *)0 );
20     insertCovalInfo( &cov_module, cov_name );
21 }
22
23 static int action ( uintCoval *cov, void *arg )
24 {
25     //省略
26 }
```

図 7 モジュールの実装ファイルの定義例

4.3 設定ファイルの書き方

設定ファイルには，coval のバインドに関する情報を記述する．図 8 に設定ファイルの書き方の例を示す．

この例で，cov_A，cov_B，cov_C，cov_D は coval 名である．2 つの coval を相互にバインドさせるには，bind(coval 名 1，coval 名 2)；の様に記述する．例の 1 行目では cov_A と cov_B を coval 名に持つ coval をバインドしており，共有される値は coval_A の持つ値である．

また，bind(coval 名 1, coval 名 2，値);の様に記述することで，バインドした際に coval 間で共有される値を設定することができる．例の 2 行目では，cov_C と cov_D をバインドさせ，その共有値として第 3 引数に与えられた 100 が設定される．

```
bind( coval_A, coval_B );
bind( coval_C, coval_D, 100 );
end
```

図 8　設定ファイルの書き方の例

5. 提案手法の使用例

5.1 SH2 マイコンを使用した例題アプリケーション

提案手法が動作することを確認するため，マイコンボード上で動作する時計アプリケーションを作成し検証を行った．使用したマイコンボードは Sohwa&Sophia Technologies 社からリリースされている SH2-ETB-MEMEs[6]である．このマイコンボードはルネサスエレクトロニクス社製の SH-2 プロセッサ(最高動作周波数 80MHz)を搭載し，キャラクタ LCD，MP3 デコーダ，SD カードソケットなどの周辺装置を備えている．メモリは ROM が 512KB，RAM が 32KB[7]であり，OS は無いがスタートアップルーチンや各周辺機器制御用関数は提供されている．

時計アプリケーションはタイマ機能を持つモジュールと出力を担当するモジュールから構成されている．タイマモジュールは内部に時間を秒単位で表す値を格納するための coval 変数を持っており，マイコンボードのタイマを利用して 1 秒間隔で値を更新する．出力モジュールはマイコンボードのキャラクタ LCD に時刻を表示する機能を備える．出力モジュールの持つ coval は，キャラクタ LCD を再描画する関数がコールバック関数として設定されているため，共有値が変更される度に描画されている文字も更新される．

設定ファイルには，これら 2 つのモジュールをバインドさせる指示を記述する．設定ファイルは図 9 のように記述する．今回は，あらかじめ SD カードに設定ファイルを保存しておき，アプリケーションの実行時にマイコンボードの SD カードソケットに挿入された SD カードから設定ファイルの内容を読み込む．図 10 に動作例を示す．

```
bind( cov_timer, cov_display );
end
```

図 9　時計アプリケーションの設定ファイル

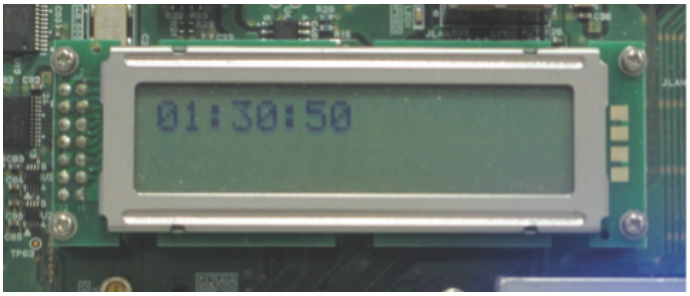

図 10　時計アプリケーションの動作例

5.2 アプリケーションへの機能の追加

例題アプリケーションのモジュール群に新たにアラームモジュールをバインドし，機能の追加を行った．アラームモジュールはあらかじめ設定された値とバインドにより共有される coval の値とを比較し，一致した場合に音楽ファイルの再生を行う機能を有する．図 11 に動作概念を示す．

モジュールの追加には，設定ファイルに新しいモジュールをバインドさせる記述を追加すればよい．変更後の設定ファイルを図 12 に示す．

このように，機能をモジュールを単位としてまとめておき，必要な場合に設定ファイルでバインドを指示するだけで追加が可能になる．モジュールを追加する際に，それぞれの機能を呼び出すコードを記述したり，既存の実装ファイルを書き換えたりする必要はない．

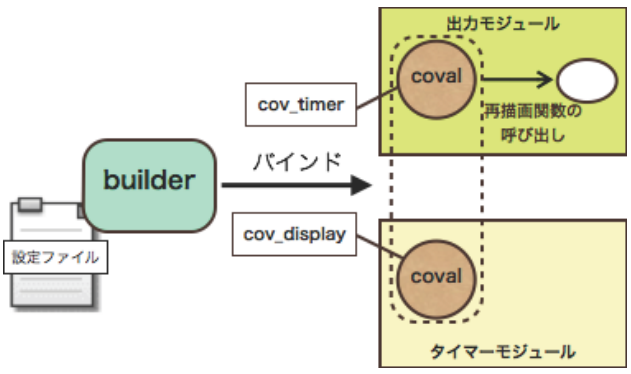

図 11　時計アプリケーションの概念

```
bind( cov_timer, cov_display );
bind( cov_timer, cov_alarm );
end
```

図 12　アラームモジュール追加時の設定ファイル

6. 議論

6.1 提案手法のメリット

(1)モジュールの独立性の向上

提案手法では，coval がモジュール間のインタフェースの役割を果たし，coval のバインドについては設定ファイルで記述を行うことができるため，実装ファイルにモジュール同士の依存性を生み出すコードは含まれない．そのため，モジュール自体の独立性が向上し，再利用しやすい部品としてまとめておくことが容易になる．過去に作成したモジュールを再利用することで，開発の度に同じ機能のモジュールを作成する必要がなくなり，開発のコストを下げることができる．

(2)再コンパイル作業の抑制

ソフトウェアを構成するモジュールを組み替えるのに，実行ファイルを直接書き換える必要がなくなり，それに伴って再コンパイル作業が不要になるという点もメリットとして挙げられる．例えば，クロス開発において，実装されているデバイスや実現すべき機能が少しずつ異なる製品群があるとする．このような場合，それぞれの製品について実行コードを生成し，動作確認を行うのは多くの手間と時間がかかる．提案手法では，利用される可能性のあるモジュールをすべて含めた実行ファイルを作成しておき，各製品ごとに設定ファイルを書き換えるだけでよい．

6.2 提案手法のデメリット

(1)静的解析が行われない

メリットの項でコンパイル作業を行わないことを述べたが，これはデメリットにもなりうる．例えば，設定ファイルに記述した coval 名のスペルが間違っていた場合がそうである．スペルミスによって対象の coval を選択できず，バインドが成り立たなかったことによりソフトウェアが予期もしない動作をすることを避けるため，現在の builder では，設定ファイルに記述された coval 名を coval 情報テーブルから見つけることができなかった場合はエラーを返し，バインド処理を中断するようにしている．

対策としては，スペルミスの発見や誤り補正を行うツールを作成することが考えられる．

(2)型の指定間違いの問題

coval は本来，同じデータ型を持つもの同士がバインドすることを基本としているが，提案手法の設定ファイルにおける記述ミスにより，異なる型を保持する coval 同士がバインドしてしまう可能性がある．これは設定ファイルに記述できる情報が coval を識別するための coval 名のみであり，それぞれの coval がどの型に対応しているかを設計者が把握しておかなければならないためである．誤って異なるデータ型を持つ coval をバインドさせてしまった場合，ソフトウェアがどのような動作を行うかは予想することができない．

この問題を避けるためには，各モジュールの coval が持つデータ型をきちんと把握しておき，誤った記述を行わないように細心の注意を注ぐしかない．

(3)実行時のコスト

提案手法を用いることで，新たにバインドを追加し，coval 経由で新しい機能を呼び出すようにできる．しかし，追加された機能の処理に時間がかかったり，多くの資源を必要とする場合，システム全体の処理が遅くなったり，リアルタイム制約を伴う機能が要求を満たさなくなったりする恐れがある．

この点に関しては，どの機能がどれだけの時間や資源を使うかを事前に把握しておき，バインドによる悪影響が無いことを保証する必要がある．また，今後はそのような設計，および再利用の手法についても検討が必要である．

6.3 DI コンテナの設定ファイルとの比較

DI を実装している代表的なフレームワークとして挙げられる Spring Framework では，オブジェクトの依存関係を記述した設定ファイルは XML 形式で記述されている．これは，生成するオブジェクトの初期化に必要なパラメータを，XML に存在するタグを使用して渡すためである．タグの指定の仕方によって，整数，文字列，生成するオブジェクトが使用したい別のオブジェクトを選択し，生成するオブジェクトに渡すことができる．また，1 つのオブジェクトに対してタグを複数記述することもでき，初期化に必要なパラメータが 1 つ以上必要であっても対応することができる．タグはそれぞれ独立しているため，1 つのパラメータだけを変更し，状況に応じて必要なオブジェクトを生成することも可能である．

本研究における設定ファイルは，coval のバインドを設定するためのものであるため，記述される情報はどの coval をバインドさせるかという情報とバインドした際に設定される初期値のみである．本研究では現在，初期値として設定できる値は限られているが，本来 coval は共有できる値として整数型，浮動小数点数型，構造体を指定することができ，これらのデータを保持する coval に対しても値を設定できるようにすることが今後の課題である．

6.4 設定ファイルの可読性の問題

Spring Framework の設定ファイルのように，XML 形式で記述することのメリットとして，可読性の向上が挙げられる．XML は階層構造をとることができ，適切にインデントを使用することによって，設定ファイルの内容を直感的に把握しやすくすることができる．また，タグの種類によって，どのようなデータを設定すれば良いか判断することができるため，複数のパラメータを必要とする場合でも分かりやすい．このように XML 形式で記述することにより，設定ファイルの可読性を向上させる工夫を施している．

本研究における設定ファイルは，可読性が良いとは言い難い．これは，coval をバインドさせる記述の書式が 1 通りしかないため，似たような記述が続くことになるためである．そのため，多くのモジュールから構成されるソフトウェアのバインド関係を記述する場合など，記述量が増大するほど設定ファイルは複雑化すると考えられる．この場合，注目したいバインド関係がどこに

記述されているのか，必要なバインドが既に記述されているかなどを知ることが難しくなる．また，提案手法のデメリットの項目でも述べた，スペルミスやcovalの所属するグループの把握にも支障を来すと考えられる．

本研究における設定ファイルでは，#を付けた箇所から行末まではコメントとして扱われるようになっている．そのため，バインド関係に対し適宜コメントを付けたり，バインド関係を続けて記述する場合は，空白行をはさむなどして，少しでも読みやすくなるように心掛けなければならない．

7. おわりに

本稿では，オブジェクト指向で広く使用されている DI 技術を参考に，coval のバインドを行うコードをモジュールの実装ファイルから切り離し，設定ファイルに記述できるようにした．また，実行時に設定ファイルに基づいてモジュールの coval を相互にバインドする機構の作成を行い，マイコンボード上で動作の確認を行った．

提案手法は現在，限られた範囲でしか使用できないため，さらに広い範囲で使用するために，問題点の改善と機能の拡張を行う必要がある．

謝辞　組込みボードの技術的な問題についてアドバイスを頂きました、株式会社 Sohwa & Sophia Technologies の湯瀬卓見氏に感謝致します．

参考文献

1)　Fowler, M.: Inversion of control containers and the dependency injection pattern (online), available from <http://www.martinfowler.com/articles/injection.html>, 2004.
2)　阪田浩一：Spring による Web アプリケーション スーパーサンプル 第 2 版，ソフトバンククリエイティブ株式会社, Nov.2010.Word 2007
3)　長谷川裕一，麻野耕一，伊藤清人，岩永寿来，大野渉：Spring 2.0 入門 Java・オープンソース・Web 開発自由自在, 技術評論社, Jan.2007.
4)　荻原剛志：手続き型言語におけるデータバインディング機構の提案と構造化設計への適用,情報処理学会論文誌, 54 巻, 4 号, pp.1573-1580 (平 25-4).
5) Microsoft：Data Binding Overview (.NET Framework 4.5) (online), available from <http://msdn.microsoft.com/en-us/library/ms752347(v=vs.110).aspx>.
6) Sohwa&Sophia Technologies：K-Skill プロジェクト (online), available from <http://www.ss-technologies.co.jp/service/mono/product/k-skill/index.html>.
7) RENESAS：製品情報　SH7083, SH7084, SH7085, SH7086 (online), available from <http://japan.renesas.com/products/mpumcu/superh/sh7080/sh7080/index.jsp>.