ソフトウェア工学 80－2
（1991. 7. 18）

マイコン組み込みシステムのためのソフトウェア・アーキテクチャ設計の一手法

井上勝博　　西村朋子

株式会社 東芝　システム・ソフトウェア技術研究所

マイコン組み込みシステムを対象としたソフトウェアの分析、設計の試みについて述べる。ソフトウェアの分析、設計技法については従来からいくつかの方法が提案されているが、この分野において、有効と考えられる技法は見あたらない。その理由の一つとして実現における強い制約がある。本試みでは、実現における問題点の緩和とソフトウェアの再利用性、拡張性向上を目的としてアーキテクチャモデルを介したソフトウェアの分析と設計方法を考えた。このモデルは、並行タスクとデバイス抽象化の概念を用いてソフトウェアのモジュール化をすすめると共に、実現における効率を考慮している。本方法をエアコンのソフトウェアに適用した結果、効率よいソフトウェア構造を設計することができた。

A SOFTWARE DESIGN METHOD FOR EMBEDDED MICRO-COMPUTER SYSTEMS

Katsuhiro INOUE　　Tomoko NISHIMURA

System and Software Engineering Laboratory, TOSHIBA Corporation
70 Yanagi-cho Saiwai-ku Kawasaki, 210 Japan

This paper describes a software design method for embedded micro-computer systems. Some kinds of software design methods have been proposed. But they aren't suitable for the embedded micro-computer systems. One of the reasons is restriction of the hardware resources. To relieve this problem, we propose a software analysis and design method with software architecture model. This model is defined for the purpose of effective software design and implementation, and to increase software reusability and extensibility with concurrent tasks and abstract device interface concepts. This method has been applied to redesign of software of air-conditioners. In result, we have been able to design the effective software structure.

1．はじめに

マイコン組み込み型のリアルタイム制御システムのソフトウェアの設計法について考察する。

近年、家電製品を代表とするマイコン組み込み製品の増加は著しいものがある。ソフトウェアに要求される機能、規模ともに拡大しており、ソフトウェアの機能拡張は、頻繁に繰り返されている。これにともない、この分野においてもソフトウェアの拡張性、再利用等が強く要求されている。

従来より、ソフトウェアの設計技法としてＳＡ／ＳＤ、ＲＴＳＡ、ＪＳＤ等の多くの技法[1-4]が提案されているが、この分野において、有効と考えられる技法は見あたらない。その理由の一つとして、実現における強い制約がある。この分野では、コスト的な問題からハードウェアリソース（ＣＰＵ，メモリ容量等）を必要最小限のものに抑える傾向にあり、ソフトウェアはハードウェアの制約を最大限に意識した最適設計を行う必要がある。このように設計されたソフトウェアは、一般に、再利用性、拡張性の観点で問題となることが多い。

再利用性、拡張性を考慮した専用の高級言語から、直接プログラムに変換する方式も考えられるが、対象とするハードウェアは多種多様であり、標準化されていないことから容易ではない。

本稿では、これらの点を考慮したソフトウェアの設計法について考察する。ここでは、以下のような方法によって、ソフトウェアの再利用、拡張性、実現効率を考慮した設計を行った。

(1)拡張性、再利用を考慮したソフトウェアの枠組み（アーキテクチャモデル）を決める

(2)仕様あるいは既存のソフトウェアを分析する

(3)分析結果をアーキテクチャに当てはめることにより、ソフトウェアを設計する

以下、これらの方法について述べ、エアコンへの適用事例について紹介する。

2．ソフトウェア・アーキテクチャ設計

2．1　アーキテクチャモデル

アーキテクチャモデルは、ソフトウェアの再利用性、拡張性を考慮したソフトウェアの枠組みである。ここでは、対象をマイコン組み込みシステムに限定した範囲で有効と考えられるアーキテクチャを提案する。

マイコン組み込みシステムのソフトウェアの役割は、周辺機器のリアルタイム制御である。リアルタイムな制御システムには、一般に並列性が内在する。並列性の内在したソフトウェアに対して有効な手法としてマルチタスクの考え方がある。タスクは、並列性を含まない実行の単位として高度なモジュール化を可能にする。

マルチタスクを実現するためには、個々のタスクを並行に動作させるスケジューラが必要である。スケジューラの組み込みについては、従来より試みられており[6]、その有効性も実証されている。したがって、ここでもスケジューラの導入を図り、モジュール化を進める。

ただし、タスクをどのように切り分けるかという点に関しては明確な基準がない。タスクの切り分けは、ソフトウェアの拡張性、再利用性に関して重要な要因である。ここでは、周辺機器の制御という観点でタスクを切り分ける。すなわち、マイコンは、周辺に接続された多くの周辺機器（入力デバイス）から信号を受取り、それに対する制御を周辺機器（出力デバイス）に対して行う。したがって、ソフトウェアには、周辺デバイスに強く依存する部分が存在する。ハードウェアの変更は、直接周辺デバイスの変更に関係するため、個々のデバイスに対してタスクを割り当てることにより、ソフトウェアの変更を局所化できる。

また、デバイス制御は低レベルであり、システム全体としての機能とは明確に分けることができる。デバイスに対して割り付けたタスクにより、デバイスを抽象化すれば、システムの機能は、抽象化されたレベルで捉えることができる。このような観点で切り分けたタスクをここではＤＩＴ(Device Interface Task)と呼ぶ。ＤＩＴは、内部に状態を持つ自律的なタスクである。

ソフトウェアをＤＩＴの観点で切り分けると、残りの部分は、個々のＤＩＴを関係づけることによって機能を実現する部分として位置づけられる。この部分を機能部とよぶ。機能部を機能によって分けたタスクをここでは、ＦＴ(Functional Task)とよぶ。組み込みマイコンの場合、一般に、機能部に対してＤＩＴ部の占める割合は大きくこのような分割方式は有効であると考えられる。

以上の観点よりアーキテクチャ・モデルを図１のように捉える。

２．２　タスク内部構造とスケジューラ

マイコン組み込み型ソフトの特徴から、ＤＩＴの内部の構成についてもある程度モデル化することができる。すなわち、ＤＩＴは、デバイスの制御と抽象化の役割を持つ。デバイスの制御は、外部からの入力の有無を調べるためのポーリングあるいは外部機器を駆動するためのパルス出力などの処理があり、msecオーダーの短周期で正確に駆動する必要がある。このため、通常タイマ割り込みを用いてこれらの処理ルーチンを起動する。また、これらの処理によりえられたデータの意味を解析することが必要であるが、この処理はもう少し長い周期での処理が可能であり、要求された時のみ処理すれば良い。これらのことから、ＤＩＴの内部を「タスク」＋「割り込み駆動ルーチン」で構成する。（図２）このようにすることにより、正確な外部機器の駆動が可能となり、個々のデバイス制御は並行処理が可能となる。

マルチタスクを実現する場合、スケジューラが必要となるが、この場合、並行に動作させることのできるタスク数にはハードウェアリソースにより制限される。タスク数を増やせば、それに比例したリソースを必要とする。ここで対象としている分野ではリソースに強い制限がある。特に、メモリがＲＡＭ，ＲＯＭに分かれており、マルチタスク実現において必要とされるＲＡＭは特に制限される傾向にある。したがって、ハードウェア面からみれば、タスク数は少ない方がよい。ただし、拡張性、再利用性からみれば、タスク数に制限があるのは望ましくない。この矛盾への対策として、スケジューラによるタスク制御を以下のような３種類のパターンに分ける。

(A)割り込み起動
(B)周期起動
(C)逐次起動

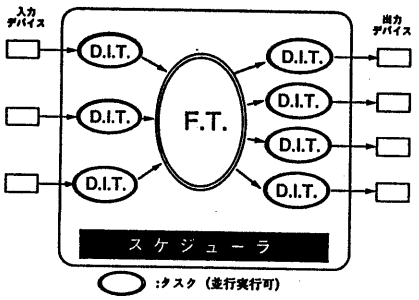

図１　アーキテクチャ・モデル

通常の並行タスクは(B)により実現できる。(A)は(B)の周期駆動では間に合わない処理が必要となる部分の制御に割り当て、(C)は逐次処理で間に合う部分の制御に割り当てる。

これらの制御パターンにアーキテクチャモデルの各モジュールを割り付ける。(A)は、正確な起動が要求されるモジュールの起動に用いる。アーキテクチャモデルでは、ＤＩＴの割り込み起動ルーチンがこれにあたる。(B)が並行動作可能なタスクの起動方式である。したがって、並列性を導入することによりモジュールの独立性が得られる部分に用いる。アーキテクチャモデルでは、ＤＩＴのタスク部にこれを割り付ける。(C)は拡張性、再利用性のために設けた起動方式であり、アーキテクチャモデルの機能部にあたる部分のモジュール化に用いる。

このようなパターンに分けることにより、タスク（モジュール）数の制約を受けることなくソフトウェアのモジュール化が可能となる。

なお、本アーキテクチャモデルでは実現言語を考慮してタスク間インタフェースをグローバルデータ領域を介したデータの受け渡しのモデルで考え、(B)タスクに関しては、wait,wake-up等の制御命令をスケジューラが提供するものとする。

２．３　分析／設計手順

以上のような観点で設定したアーキテクチャモデルに対してソフトウェアを当てはめることにより、設計を行う。ここでは、そのための分析方法とその結果から設計に結び付ける方法について述べる。

２．３．１　分析

まず、既存ソフトおよび仕様書よりソフトウェアを分析する。以下にその手順を示す。

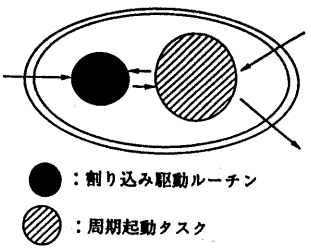

図２　ＤＩＴ内部構成

①システム構成の把握

マイコンにより制御する対象を明確にする。

ＳＡにおけるコンテキスト・ダイアグラムを用いる。

②入出力調査

①で調査した個々の外部エンティティ（デバイス）に対して入出力情報（ここでは、信号、データをまとめて情報とよぶ）およびその方法（割り込み／ポーリング、ポーリングならその周期、デバイス駆動周期）等を調査する。これらの情報は、ＤＩＴに直接反映する。

③内部処理調査

(1)情報を抽出する。

入力と出力の情報は②で明確になっている。その他の情報に関しては、既存ソフトまたは仕様書より抜き出す。既存ソフトの場合、内部で使用されているデータを抜き出す。仕様書の場合は抜き出す情報の選択が問題となる。その方法としては、基本的に、入出力情報より順次辿っていくことにより発見する。全ての単語を抜き出し分類することも考えられる。

(2)情報を関連付ける。

抜き出した情報を関連付ける。

「情報１→○→情報２」の形式で記述し、「情報１から情報２ができる」ことを意味する。情報の変換部には処理が存在することから「○」によって処理を表す。図３に例を示す。この図は、

・［相対温度］は、［設定温度］と［運転モード］と［測定温度］から作られる。

・［相対湿度］は、［運転モード］と［測定湿度］から作られる。

ことを意味する。

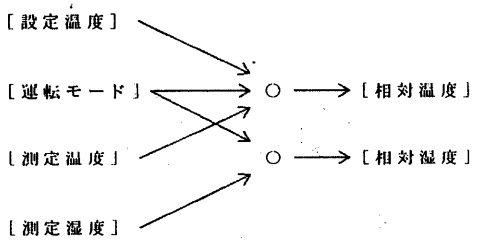

図３　情報関連図

この図は、データを中心に考え、データから処理を導くことを目的とした記述法で、見かけ上はデータフロー図に類似しているが、データフローがどちらかと言えば処理（プロセス）中心に記述を進めるのに対し、この記法は、データを中心に進めていくことが異なっている。理由は、データの方が処理（機能）よりも明確であるからである。

(3)詳細化

情報の関連づけはＳＡ手法と同様に抽象レベルから詳細レベルへと段階的に進める。ただし、本手法ではバブルの詳細化ではなく、情報の詳細化により詳細化される。

④処理の起動周期、起動順序の整理

③において抽出したバブルに関して周期的に起動する必要のあるもの、処理に前後関係があるものを明確にする。

２．３．２　設計

以上分析した結果を、アーキテクチャモデルに当てはめる。以下に、その手順を示す。基本的に、分析結果のバブルをグルーピングすることにより、モジュール化を進める。

①ＤＩＴを決定する。

デバイスを抽象化することを目的としてＤＩＴを決める。ＤＩＴ決定に際する基準として以下のものを考える。

(1)抽象化

・そのデバイスをできるだけ抽象化できるようにグルーピングする。

(2)独立性

・個々のデバイスに対して独立になるようにグルーピングする。

・タスク内に（外に）そのデバイス以外から入る（出る）情報はないようにする。

(3)処理効率

・デバイスの駆動方式および入出力情報が同様なものは一つにまとめる。

②ＦＴを決定する。

ＤＩＴを除いた残りの部分は、機能を実現する部分と考えることができる。この部分に関しても、機能拡

張、仕様変更等の局所化を図るため、モジュール化を進める。ただし、ＦＴに関しては、ＤＩＴほど明確なモジュール化の指針が存在しない。ここでは、以下のような指針によりバブルをグルーピングする。

(1)処理（バブル）の制御関係

起動周期が同じものはまとめる。

起動の前後関係に矛盾がないようにまとめる。

(2)機能の関連性

同じ機能を実現するものはまとめる。

(3)バブル間の情報の関連

情報の関連の少ないところで切る。

(4)処理の分散

グルーピングした処理のサイズをなるべく均等にする。このために、分析の段階で、処理のサイズを見積もることが必要である。

以上の指針を組み合わせることによりグルーピングを行う。

③タスク間インタフェースを明確にする

グルーピングを行った結果、その境界に現れる情報がインタフェースとなる。ここでは、グローバルなデータ領域をインタフェースとする実行モデルを前提にしている。グルーピング内部に現れる情報は、ローカルデータとしてメモリ設計時の参考となる。

なお、並列実行可能なタスクとして実現されるタスクは、制御に関するインタフェースを決定する必要がある。

④スケジューラ設計

ハードウェア資源を調査し、２．２で述べた制御パターンの実現の可能性を調べる。特に、割り込み、レジスタ、スタック等の情報から、並列実行可能なタスク数を決定し、タスク分割の結果からタスク切り替え周期等を決定することが重要で、上記で分割した各モジュールを３つの制御パターンに振り分けを考える。

３．適用事例

上記方法で実際のエアコンのソフトウェアを設計した。ここでは、その適用事例について紹介する。ただし、ここで紹介する事例は、説明のため実際のものより簡略化している。

３．１　分析

①システム構成の把握

図４にエアコンのコンテキストダイアグラムを示す。これは、ハードウェア構成図を抽象化することにより得ることができる。これによりマイコンの制御対象を明確にする。

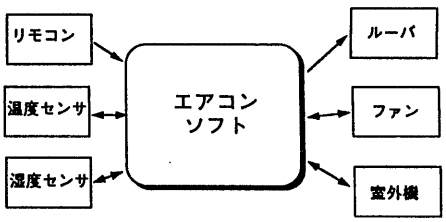

図４　エアコン・システム構成

入力

デバイス名	入力形式	周期	処理時間見積	補足
リモコン	シリアル信号	外部割り込み 4.4ms	60μs/bit	8bit 8 8 8 8 8 8 4.4 4.4 リーダーコード＋３ワード（＝６バイト） 1bit＝ 1:（2ms） 0:（1ms） 信号のエッジで割込発生
温度・湿度センサ	A/D入力	6.14ms	26μs	同一入力２回でＯＫ．センシングに約50ms．センサ駆動に同期．

出力

デバイス名	出力形式	周期	処理時間見積	補足
ルーバ	ルーバ駆動	6.14ms	104μs	出力1 2 3 4 T=6.14ms 全体として８Ｔ周期でﾀｲｸﾘｯｸ．各信号は，８Ｔ中　３Ｔがオン．
温度・湿度センサ	センサ駆動	駆動中3.07ms 駆動周期12s	13μs	1s センｼﾝｸﾞ 10s 「ｾﾝｼﾝｸﾞ」時にI-f入力

表１　入出力調査

②入出力調査

入出力情報およびその入出力方法をまとめる。デバイスの制御方式については仕様書に決められている。これらを表にしてまとめる。表１にこの一部を示す。

③内部処理調査

(1)情報を抽出する。

ここでは、既存ソフトおよび仕様書より仕様書で用いられている用語とソフトのデータを関連づけて抜き出した。

(2)情報を関連付ける。

抜き出した情報を関連付ける。関連付けは、抽象レベルで、入力から出力に向かって関連付けた。つぎに、各情報を詳細化することにより実現レベルでの使用データまで詳細化する。

図５に抽象レベルの関連図を示す。

④処理の起動周期、起動順序の整理

②③で抽出した情報を用いて、制御の流れをまとめる。図６にこの図を示す。なお、点線矢印は実行順序を示し、網掛け部は単周期の処理が必要であることを示す。

３．２　設計

①ＤＩＴの決定

デバイスに対して直接関係する処理をまとめる。前節の基準を基に決定したＤＩＴが図７のＤＩＴである。この例では、抽象化、独立性の観点で、リモコン、ルーバ、ファンのＤＩＴを決定し、処理効率の観点を加えて、センサ、室外機のＤＩＴをまとめている。

次に、ＤＩＴの内部の処理を考える。例えば、室外機の場合、室外機の制御のためにシリアル送受信を行い、データへの変換、データの解析という処理を行う。シリアル信号受信およびデータへの変換は、msec単位の正確な処理が必要で、データ解析はそれほど単周期の処理は要求されない、したがって、ＤＩＴの内部をタイマ割り込みを利用した単周期の駆動ルーチンと通常のタスク部より構成する。今回の設計において、全てのＤＩＴは、この構成により実現できることを確認できた。

②ＦＴのモジュール化

ＦＴは、その内部を並行処理を可能とするか逐次処理とするかによって、モジュール化の観点は異なる。ここでは、タスク数に余裕がなく、逐次処理としなければならない場合を考える。逐次処理の場合、処理の前後関係を考慮しなければならない。したがって、制御の流れに重点をおいて、機能的な処理の分割が可能なように考慮する。分割の基準として前節で述べた基準を用いている。その結果を図７のＦＴ部に示す。最終的なタスク構成図を図８に示す。この分割により、ＦＴ内部は、モード計算、相対値計算、基本制御、コンプ保護、ファン保護に分けることができ、この順に実行することによりデータの生成順に問題が発生しないことが図６よりわかる。

③スケジューラ設計

タスクの制御パターンを(A)～(C)の３種類に分けたが、これらの制御パターンに上記タスク群を分類する。図８において、網掛け部が(A)、斜線部が(B)、ＦＴ内の５つのモジュールが(C)に割り付けられる。スケジューラには、これら３種類の制御部を設け、各々のタスクを各々の制御部に登録することにより制御を切り分ける。

４．考察

上記適用事例を基に、本方式を考察する。

(1)分析モデル

ここでは、分析の対象として既存ソフトを挙げ、分析モデルとしてプロセスモデルを用いた。ただし、プロセスよりデータを優先して分析した。既存のソフトは、データが明確であることからこの方法は有効であり、簡潔な方法となった。

ただし、処理方式が既存ソフトのものを流用する形になる可能性は高い。根本的に見直す場合には、より厳密な分析が必要であろう。

(2)アーキテクチャモデル

分析と設計の橋渡しとしてアーキテクチャモデルを考えた。通常の分析技法の問題点は、実現とのギャップであるが、ここでは、アーキテクチャモデルを実現効率を優先して考え、ギャップの緩和を考えた。これにより、実現効率の高い分析、設計が行えた。

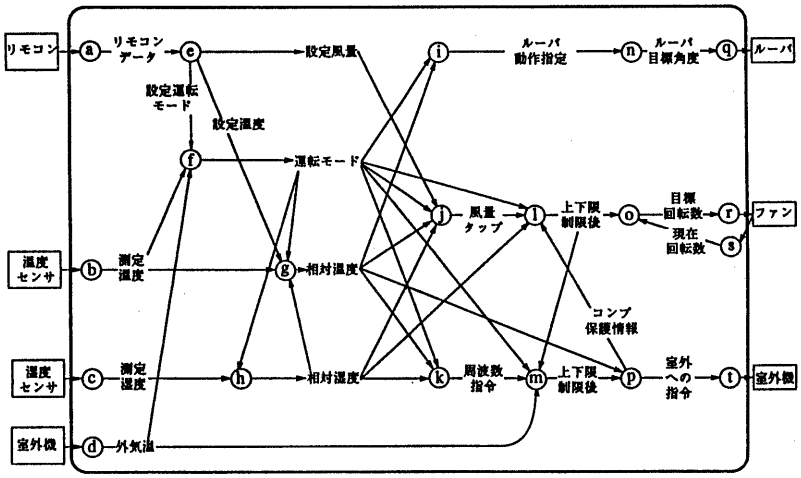

図５　情報の関連

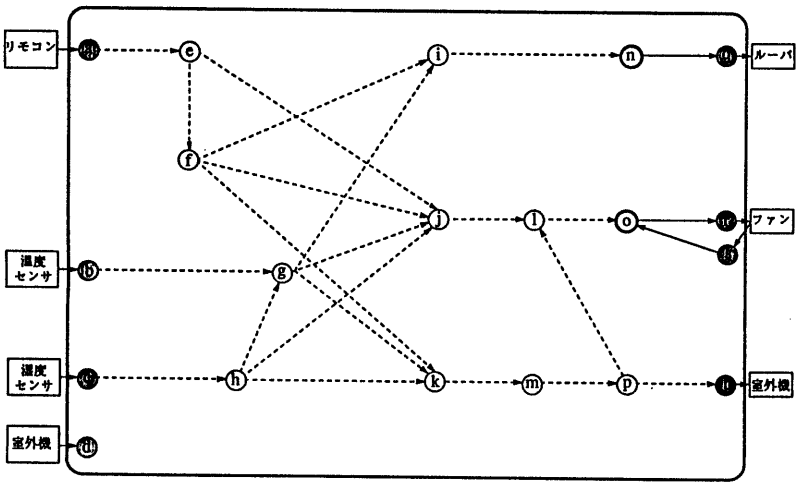

図６　制御の流れ

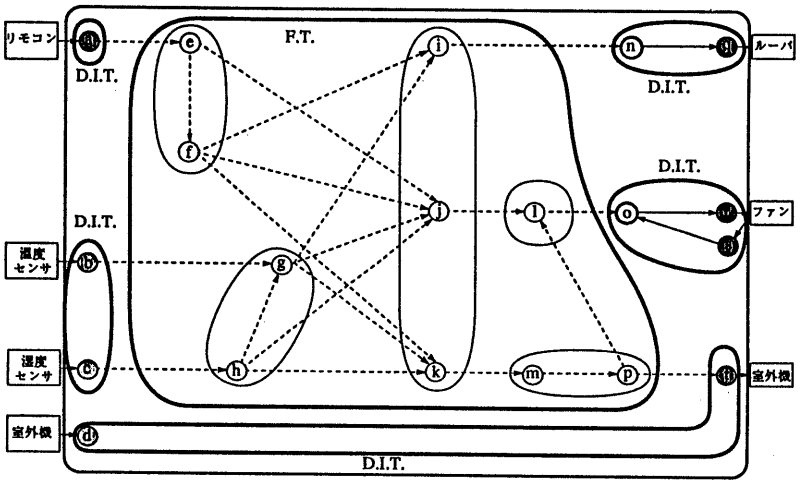

図７　タスク分割

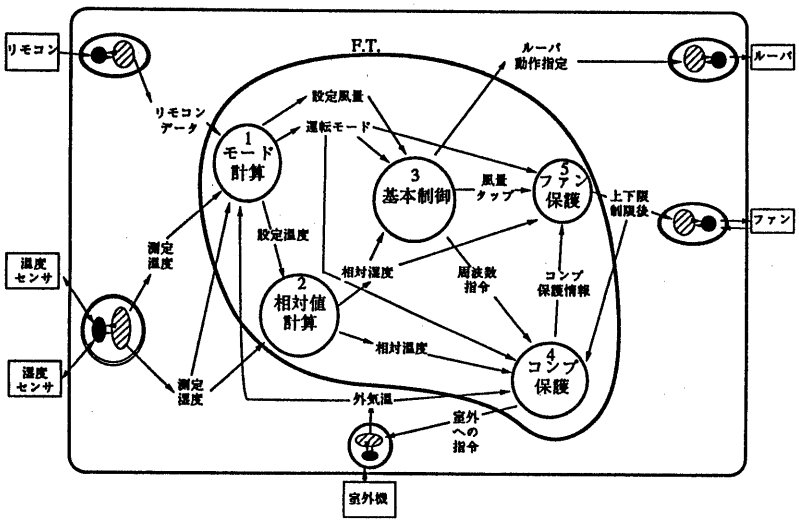

図８　タスク構成

(3)タスク分割

ＤＩＴはデバイスの抽象化という観点で決定要因は比較的明確であるが、ＦＴの決定は難しい。グルーピングによる方法を用いたが、これは、データの依存関係、制御の流れ、機能的関連、実現効率等の要因が複雑に関連して決定する。ここでは４種類のグルーピング基準を挙げて機能的分割を試みた。このような分割により、仕様と機能の関連づけに有効であると考えられる。

(4)仕様変更、再利用への対応

マルチタスクの考え方を用いてプログラムをモジュール化し、スケジューラにより制御構造をメインの処理と切り放した。また、ＤＩＴはデバイスに依存た互いに独立なモジュールであることからこの点に関して有効であると考えられる。この点に関しては、より厳密な評価が必要であり、今後の課題である。

(5)実現効率

エアコンのプログラムについて実現効率を最優先に考えて設計したバージョンと今回、再利用、拡張性を考慮してスケジューラを組み込み、モジュール化したバージョンを比較した結果、プログラムサイズ 3.2%増、データサイズ 6.4%増となり、かなりオーバヘッドを抑えることができた。

５．おわりに

マイコン組み込みシステムを対象としたソフトウェアの分析、設計の試みについて述べた。ソフトウェアの分析、設計技法については従来から多くの方法が提案されているが、実現において、この分野に適した方法は見あたらない。本稿では、実現におけるギャップを緩和するため、アーキテクチャモデルを設定し、それを核とした分析設計法を提案した。この方法をエアコンに適用した結果、効率よく実現することができた。今後、適用範囲を広げることにより、本手法の汎用化を行う予定である。

参考文献

[1]DeMarco T:Structured Analysis and System Specification,Prentice-Hall,1981(高梨他訳"構造化分析とシステム仕様",日経ﾏｸﾞﾛｳﾋﾙ社,1986)

[2]Hatley D.J.,Pirbhai I.A.:Strategies for Real-Time System Specification,Dorset House Publishing,1987-88(立田訳,"リアルタイムシステムの構造化分析",日経ＢＰ社,1989)

[3]M.A.Jackson:System Development,Prentice Hall,1982

[4]H.Gomaa:A Software Design Method for Real-Time systems, Communication of the ACM, Vol.27, No.9, September 1984

[5]K.H.Britton,R.A.Parker,D.L.Parnas:A Procedure for Designing Abstract Interfaces for Device Interface Modules,Proc. of 5th ICSE,pp.195-204,March,1981

[6]西村他：シングルチップマイコンソフトへの簡易スケジューラの導入、情報処理学会研究会報告、Vol.90,No.10, 90-SE-71-3, 1990