ZIPCを用いた家電組込みソフトウェア開発の効率化

九州日立マクセル株式会社
新分野開発PT　ソフトアドバンスGr
安部田　章

１．はじめに

近年家電製品が高度化し、そこに組み込まれるソフトウェアが大規模複雑化してきている[1)]。その一方で、市場の要求に即応することが求められており、開発の効率化と信頼性確保が急務となっている。そこで弊社では、再利用性や保守性に優れ、早期開発に適したコンポーネントベース開発環境の構築をすすめている[3)]。

コンポーネントベース開発では、ソフトウェアフレームワークを基盤として、その上で利用可能なコンポーネントを準備し、それらを呼び出すことでプログラムを作成する。そこで、当社では、自社製品における家電組込みソフトウェアのドメイン分析[6)]を行うことにより、自社ソフトウェアの移植性や再利用性を向上させる多層アーキテクチャフレームワークを導出し、フレームワーク上で動作するコンポーネント群を整備した。

組込みソフトウェアの移植性を考えたとき、まず、マイコン間の移植性と、ハードウェア間の移植性を向上させる必要がある。そこで、我々はソフトウェアの構成をマイコンハードウェアドライバ層、メカニズム層、アプリケーション層の３つのレイヤーに分割し、各階層の特性を考慮した部品化を行うことにより、効果的な再利用を目指した。

まず、マイコンハードウェアドライバ層は、マイコンのハードウェアに依存する部分を抽象化し、マイコン機種に依存しない共通なインターフェースを持つドライバを整備することでマイコン間の移植性を高めた。

メカニズム層は、制御対象のハードウェア毎にその制御メカニズムをコンポーネント化し、ハードウェア部品単位で再利用可能にした。また、ハードウェアの種別や制御構造に依存しない共通なインターフェースを提供することでハードウェア間の移植性を高めた。さらに組込みソフトウェアではハードウェア制御コンポーネントが大規模複雑化する傾向がある[4)]ため、階層化コンポーネントとして構築することにより、メカニズム層の再利用性を高めた[5)]。

家電組込みソフトウェアの場合、アプリケーション層は、製品の振る舞いを実現するため、状態によって挙動が異なる状態遷移機械としてモデル化できる。そこで、アプリケーションをいくつかの状態遷移機構としてモデル化し、アプリケーションが提供する機能を実現するコンポーネント群を準備しておく構成とした。そして、状態遷移機構における状態アクティビティと状態遷移アクションから、機能コンポーネントのサービスを呼び出すことで実装を行う。この状態遷移機械のモデル化および保守管理には、ＺＩＰＣを用いることにより、仕様漏れがなく、仕様変更を効率よく行えるアプリケーション層の開発を実現した。

本稿では、特に、アプリケーション層の構成方法とツールを用いた構築・管理支援について説明し、多層アーキテクチャフレームワークとツール利用により開発効率を向上させた事例について紹介する。

２．多層アーキテクチャフレームワーク

本フレームワークでは、図１に示すように、基本アーキテクチャを共通化し、①マイコンハードウェアドライバ層、②メカニズム層、③アプリケーション層を別レイヤーとして構成する。マイコンのハードウェアに依存する部分（①）と、機構および回路の制御メカニズムに依存している部分（②）を別レイヤとすることにより、マイコン機種変更時は、マイコンハードウェアドライバ部分のみを交換すればよい上に、機構や回路部品の変更時にも、マイコンハードウェアドライバ部分の修正がなく、メカニズム層の制御対象ハードウェアに対応する部分のみを交換すればよい。

図１：フレームワーク構成図

これにより、マイコン機種変更、メカや回路のハードウェア変更時による影響の範囲を局在化させ、移植作業の効率化と信頼性の向上を実現することができた。また、アプリケーション層(#)が分離されているため、これらの変更時には、アプリケーション層の変更が基本的には生じないようにした。

２.１．メカニズム層のコンポーネント化

組込み制御ソフトウェアのメカニズム層をコンポーネント指向で作成する場合、制御対象のハードウェアに一つの制御オブジェクトを割り当て、その制御オブジェクトがいくつかのオブジェクトを統制して一つのまとまりのある制御メカニズムを実現させる。それらの制御オブジェクトを中心としたオブジェクト群を一塊のコンポーネントとして作成し、コンポーネント単位で扱えるようにする。

これにより、ハードウェア部品の交換時に、その影響範囲をコンポーネント内に局在化できるため、メカニズム層の移植性が向上した。また、開発後には、これらのコンポーネントをフレームワークに蓄積していくことにより、次の製品開発ではコンポーネント単位で容易に再利用が可能になった。

２.２．アプリケーション層の再利用

一般に業務系ソフトウェアのアプリケーション層は、図２に示すようにシステムの提供する機能単位に処理を分割することができる。その場合、独立性の高い機能単位で追加・変更・削除を行うことが可能である。

これに対して、制御系ソフトウェアのアプリケーション層は、システムの振る舞いを実現する部分であるため、システムの動作モード単位に処理を分割することになる。

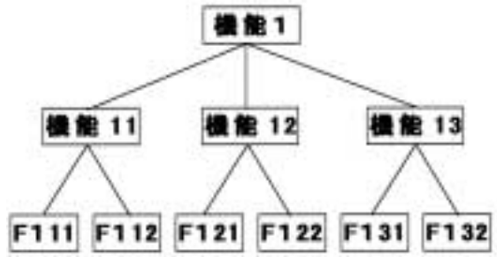


図２：機能分割によるプログラム構成

すなわち、制御系ソフトウェアの場合、システムの振る舞いが動作モードによって変化する特性を持ち、いくつかの動作モードとその遷移構造がシステムの機能を実現している。したがって、動作モードの追加・変更・削除と、そのモード遷移構造の変更を容易に行える必要がある。

そこで当社では、図３に示すように、アプリケーション層の上位層を、状態遷移機械（動作モード、イベント、遷移およびアクション）として定義し、動作モードごとに機能分割する状態遷移フレークワークとして構成する。アプリケーション層の下層には、動作モードを実現するのに必要な機能をコンポーネント化して準備し、これらを状態遷移フレームワークから呼び出すことによってアプリケーション層を構築する。

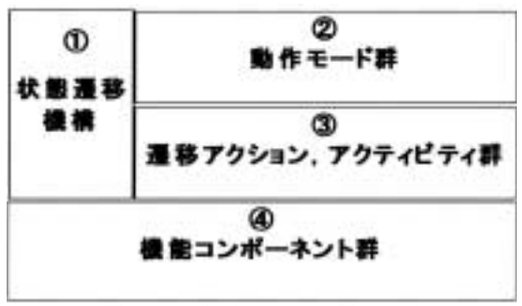


図３：アプリケーション層の構成

本構成にすることにより、製品の振る舞いの部分に変更があった場合、動作モードの追加・変更・削除によって容易に行えるようになる。さらに、提供する機能に変更があった場合には、機能コンポーネントのみを修正すればよく、状態遷移構造には影響を与えず、仕様変更に強いプログラムを作成できるようになった。

３．ZIPCによるアプリケーション層の構築

アプリケーション層の上位層の構造は、図３に示すように、①状態遷移機構を実現する部分、②動作モード群③状態遷移機構から呼び出されるアクションやアクティビティから構成される。アクションは、イベント発生や状態遷移時に実行されるべき機能や処理であり、アクティビティは、特定の状態がアクティブな間中、繰り返し実行される機能や処理である[7)]。

下位層は、これらのアクションやアクティビティから呼び出される機能コンポーネント群（④）から構成される。機能コンポーネントは、モード非依存なコンポーネントとして作成する。遷移アクションやアクティビティからは、機能コンポーネントのインターフェース関数を呼び出すことで、製品の振る舞いを実現する。

上位層の状態遷移フレームワークの部分を下位層の機能コンポーネント群から分離することにより、動作モードの変更の影響を機能コンポーネントから排除した。これにより。機能コンポーネントの独立性を高め、アプリケーション層の機能単位での再利用性を高めることができた。

さらに、製品の仕様変更による状態遷移機械への動作モードの追加、削除および変更などの操作・管理を専用ツールで行うようにすることで、製品機能の追加、削除および変更作業を大きく効率化することができた。

本状態遷移フレームワークを実装するために、当社ではＺＩＰＣを利用している。ＺＩＰＣは、状態遷移表を作成することにより、動作モードとその遷移構造を容易に実装することができる。さらに、遷移アクション、モードアクティビティなどフレームワークで必要なすべての関数をツール上で定義することができる。作成した状態遷移表からＣ言語のソースを自動生成することができる。自動生成されたソースは実際のアプリケーションの状態遷移フレームワークとしてそのまま利用することができる。

状態遷移フレームワークをＺＩＰＣを用いて実装することによる利点としてさらに、仕様変更が状態遷移表の変更といったモデルベースで行えるようになり、変更に必要な工数や変更ミスを削減できることがあげられる。

以上により、製品のラインアップやモデルチェンジ時は、モデル間の動作モードの追加、削除および変更作業をＺＩＰＣ上で行うことにより効率化することができるとともに、アプリケーション層の機能単位の再利用性を高めることができた。また、マイナーチェンジの場合、ベースモデルのアプリケーション層全体を再利用し、差分となる動作モードをＺＩＰＣ上で追加、削除、変更することにより、最小限のコストで開発することができるようになった。

４．本フレームワークの全体評価

本構成により、実際に健康機器を構築した場合の開発効率を示す。

図４に健康機器の旧型から新型へのモデルチェンジを行った場合の動作モードの追加・変更・削除を一覧にしたものを示した。モデルチェンジ品の開発は、ＺＩＰＣ上で複数の動作モードを追加変更し、遷移先の変更、アクションやアクティビティの変更を行い、Ｃ言語のソースコードを生成することによって行った。これにより、多くの部分を再利用できるとともに、変更もＺＩＰＣ上で簡単に行うことができた。

図４：動作モードの追加・変更

変更手順としては、以下の流れで行った。

①動作モードの追加・変更
②モード遷移アクションおよびアクティビティの追加・変更
③機能コンポーネントの追加・変更
④状態遷移フレームワークとしてＺＩＰＣからＣコード再生成
⑤プログラムのアプリケーション層に状態遷移フレームワークを組み込む

ＺＩＰＣを利用して本フレームワークを適用した場合の開発期間の評価をおこなった。表１に本フレームワークの適用／非適用時の開発期間を比較した。旧型から新型へのモデルチェンジにおいて、適用／非適用を比較したものだが、適用事例と非適用事例で機種が異なるため、厳密な評価はできないが、フレームワークを適用することによって開発期間が少なからず短縮されたことが示唆された。また、ＺＩＰＣ導入による工数削減の評価であるが、これに関しては直接評価できるデータがないことが残念であるが、フレームワーク非適用事例では、ＺＩＰＣを導入しておらず、適用事例ではフレームワーク適用とともに導入している。ＺＩＰＣを導入しなければ、フレークワークの作成と管理が煩雑になるためである。

表1：開発期間の評価

	ステップ数：A	開発期間：B	A/B
フレームワーク非適用	約1.6万	約150日	106.7
フレームワーク適用	約2万	約75日	226.7

５．今後の課題

今後はキャッツ株式会社製のDrawrialを利用してリモコンＬＣＤの画面設計と実装の簡易化を進めることにしている。近年世界的な健康ブームから当社健康関連の開発製品を海外で販売する要求が高まっており、ＬＣＤ表示の多言語化は必須となってきている。Drawrialは多言語開発に対応していることもあり、導入を決定した。また、現在、制御コンポーネントのモデルベースの開発を試行している。状態遷移構造を持つ制御コンポーネントを、ＺＩＰＣあるいはKonesa-RealTimeによる状態遷移機械としてモデル化し、モデルベースでの再利用を行おうというものである。以上の取り組みに関しては、次回以降に報告をしたいと考えている。

参　考　文　献

1）高田広章：組込みシステム開発技術の現状と展望、情報処理論文誌、Vol.42, No.4, pp.930-938 (2001).
2）渡辺政彦：状態遷移ベースのソフトウェア開発環境の現状と動向,計測と制御．Vol.41, No.2, pp.117-121（2002）.
3）安部田章：組込みソフトウェアの移植性を向上する多層アーキテクチャ，ウィンターワークショップイン神戸論文集，情報処理学会シンポジウムシリーズ，Vol.2003, No.5, pp.9-10（2003）.
4）渡辺博之，渡辺政彦他２名：組み込みUML，翔泳社，pp.129-132.
5）安部田章：多層化による組込みソフトウェアの移植性の向上，ソフトウェア工学研究会資料，02-SE-140-16, pp.117-122（2003）.
6）安部田章，時任正博，島田和明：組込みソフトウェアのドメイン分析に基づく多層アーキテクチャフレームワーク，組込みソフトウェアシンポジウム2003（ESS2003）論文集，情報処理学会シンポジウムシリーズ，Vol.2003, No.13, pp.14-21.
7）渡辺政彦：拡張階層化状態遷移表設計手法Ver.2.0, 東銀座出版

モーター制御プログラム開発へのZIPC適用
－ZIPC導入に際しての評価結果を踏まえて－

中央電子株式会社
制御システム事業部
船田　晋

以下にCASEツール「ZIPC」の評価を報告する。今回は、10軸モータ制御プログラムを参考に、機能ごとに定性的な評価を行った。

１．評価対象

「キャッツ株式会社」製　ZIPC 2001 Version 8.0 Professional Model

・エディタ機能、チェッカー機能、シミュレーション機能、ジェネレータ機能、VIP機能、ATV機能、リバース機能、その他

今回は、主にエディタ、チェッカー、シミュレーション機能についての評価を行った。

【参考文献】渡辺政彦著、拡張階層化状態遷移表設計手法Ver.2.0

２．参考プログラム

以前に製作したプログラムを参考にする。これは、RTOSを使用しない割込み処理主体のモータ制御プログラムであり、10軸のモータを2ch、５軸ずつ制御し、位置合わせを行うプログラムである。

No	項目	実際の処理内容	今回の参考内容
1	ソレノイド制御	上位からの指令により、ソレノイドの制御を行う。	処理内容は省略し、遷移の流れだけを参考にする。
2	モータ回転シーケンス	上位からの指令により、回転パルス数を計算し、モータ停止→回転→停止、また緊急停止のシーケンス処理を行う。	処理内容は省略し、遷移の流れだけを参考にする。
3	125us周期の割込み	ＤＩの読込み処理とモータ駆動処理を行う。	周期ハンドラで DI を監視し、イベントを起こす。

３．評価結果

3.1．所要日数

上記のプログラムを参考にして、状態遷移表（以下、STM）を作成したところ、下表の日数を必要とした。今回、ツールの利用に不慣れなところもあるので、今後の活用時においてはさらなる短縮が可能である。

表1：作成できる設計書

No	ドキュメント種類	作業内容／用途	所要日数
1	状態遷移表（STM）	イベント／状態抽出	２日
		表作成	７日
2	状態遷移図	状態遷移表からコンバートする	
3	シーケンス図	STM 作成時に必要(STM とは独立している)	
4	タイミング図	STM 作成時に必要(STM とは独立している)	

その他のドキュメントとして、関数定義書、変数設計書、定数設計書があるが、これらは今までヘッダーファイルにコーディングしていたものを個別に記述したものと考えてよい。

3.2．エディタ機能

別添資料１は、プログラムのメインフロー～ソレノイド制御～モータ回転シーケンス部分のフローチャートを示している。これをSTMで表すと、別添資料２のようになる。状態を複数のフラグで分けていたものが、遷移表にしたことで、実際の処理内容は変わらないにも関わらず、視覚化、単純化することができる。

実装を意識せずに設計ドキュメント（状態遷移表や状態遷移図など）としてのみ扱うのであれば、ソレノイド制御、割り込みなどを記述するのは容易である。

実際にSTMを作成したところ、エディタ機能の具体的な特徴として以下の点が挙げられる。

（１）STMの表記法、あるいは操作の習得は、ZIPC初心者でも容易である。ただし、状態とイベントの抽出には、要求仕様と十分に照らし合わせて考え、また処理がイベント駆動型であるか状態駆動型のどちらが好ましいかをよく検討することが必要である。

（２）イベント駆動型のSTMの場合、イベントが起こらないと処理に進まないので、シーケンス（自動遷移）処理をする場合の記述には、処理内にイベントをコールする関数を追加するなど、少々の工夫が必要となる。

（３）セル内に処理を記述して、プログラミングすることも可能であるが、それでは実装段階に踏み入ることになるので、なるべく設計をしているという意識で記述するのがよい。（セル内のＩＦ文を減らして、「状態」側を条件分岐するなど）

（４）日本語による表現が可能なため、設計当初のシステム分析やレビューには有効である。また、ハードウェア技術者が容易に理解しやすくなるという利点も考えられる。

（５）イベントや状態の開始時、終了時の処理を、数種類のマークを用いて表しているので、視覚的に遷移の流れを追うことができてよい。

（６）モータの５軸制御の場合は、５軸分のSTMで実現するのが一般的である。また全てのSTMにおいて、一旦イベントを受け付けた後、状態遷移表処理が終了するまでの間、次のイベントを受け付けることはできないようである。数軸制御のサンプルがほしい。

（７）周期ハンドラについては、特別な作りこみが必要となる。割り込みハンドラを仲介してイベントを受け付けるのが一般的である。ハンドラでタイマー割り込みを受け付け、ハンドラからSTMにイベント（メッセージやフラグなどの方法にて）を送る。この辺りの設計のヘルプ機能をもう少し充実させてほしい。

（８）STMからコンバートした状態遷移図は、線が絡み合い見にくく、設計書に添付するドキュメントとして使用するのは難しい。

（９）印刷機能は、STMがすべて印刷範囲に入るようにすると、テキストが重なり、非常に見づらくなる。長い処理文を挿入する際には省略するなどするしかない。（８）と合わせて、何らかの添付ドキュメント用の設定が必要である。

3.3．チェッカー・シミュレーション機能

（１）ZIPC最大の特徴であるモレ・ヌケのチェックであるが、処理がセル単位になっており、すべてのセルをチェックすることで、非常に発見しやすくなっている点はよい。

（２）イベント発行によるシミュレーションの実行が可能である。各イベント、状態、セル内にBREAKを置いて、遷移の流れを追う操作は問題なくできる点は素晴らしい。

（３）子STMを複数作成した場合には、シミュレーション時にアクティブになっているセルも複数あり、把握しづらい面もある。特に、割込み処理を挿入した場合、アクティブ画面が頻繁に移動し、画面が「チカチカ」するので、モード選択等をして改良してほしい。

3.4．ジェネレーション機能

（1）自動生成コードは、状態遷移表の１セルを単位として生成するので、あまり「状態」の数が多いと、生成コードのサイズも大きくなるようである。上限のサイズを選択してSTMを最適化するような処理がほしい。

（2）リバースエンジニアリング機能はあるが、初めにZIPCで生成したコードを編集したものをリバースして状態遷移表やその他のドキュメントに反映するものである。ZIPCによらない既存のコードから状態遷移表を生成するリバース機能があるといい。

3.5．ＶＩＰ機能

（1）VCまたはVBで作成した擬似ターゲットとの関連付けには、手間のかかる設定が必要であるが単純な作業ででき、シミュレーションとの連動を行える点は素晴らしく、また設計・製作を楽しくするという面でも画期的であると感じた。

（2）開発の初期段階からシミュレーションが行えるという利点がある。ハードウェア担当者と共に仕様を確認しあうことが一番のメリットである。

（3）「VIP環境設定」の「名称イベント設定」タブの削除ボタンが有効にならないことがあった。サンプルではなく、詳細な説明書がほしいところである。

3.6．その他の機能

今回は評価の対象外であったが、その他の機能を簡単に紹介する。

（1）ATV機能・・・作成したSTMを基に、すべてのイベント、状態の組み合わせについてパターンを作成して、自動試験、検証することができる。

（2）ターゲット・デバッグ機能・・・HEW等のツールと連携し、ターゲットを接続してデバッグをする。

4．まとめ

今回は、ZIPCの機能をSTM作成～シミュレーション実行を中心に評価した。仕様の全体を再現するにはさらなる詳細な設計が必要であるが、様々なシステムに対応する詳細設定が可能となっており、さらに大きなシステムを設計する場合に必要な機能は、十分に揃っている。また、ハードウェアとのインターフェイス部分をどの程度まで表記できるかは、未知である点が多く、今後の試用に任せるが、他社の運用例からしても十分な対応ができると考えられる。

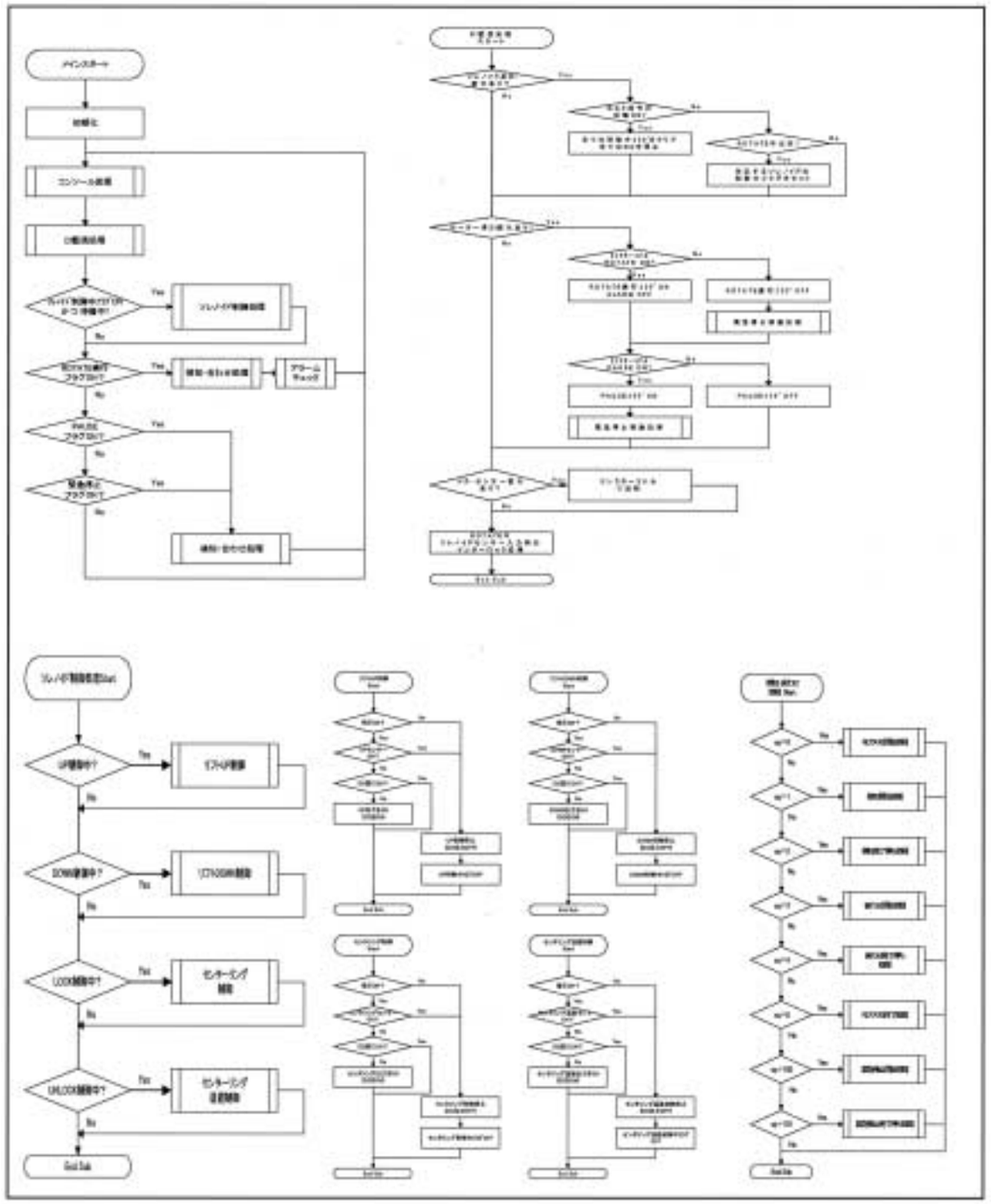
キャッツ（株）のサポート体制はよく整っており、また、ユーザーからの声を取り入れて改良している点も評価でき、実際の運用にはサポートをいただけるものと考える。

部内業務については、一つモデルが完成すれば、若干の変更をすることで、他システムに流用しやすいという点もある。

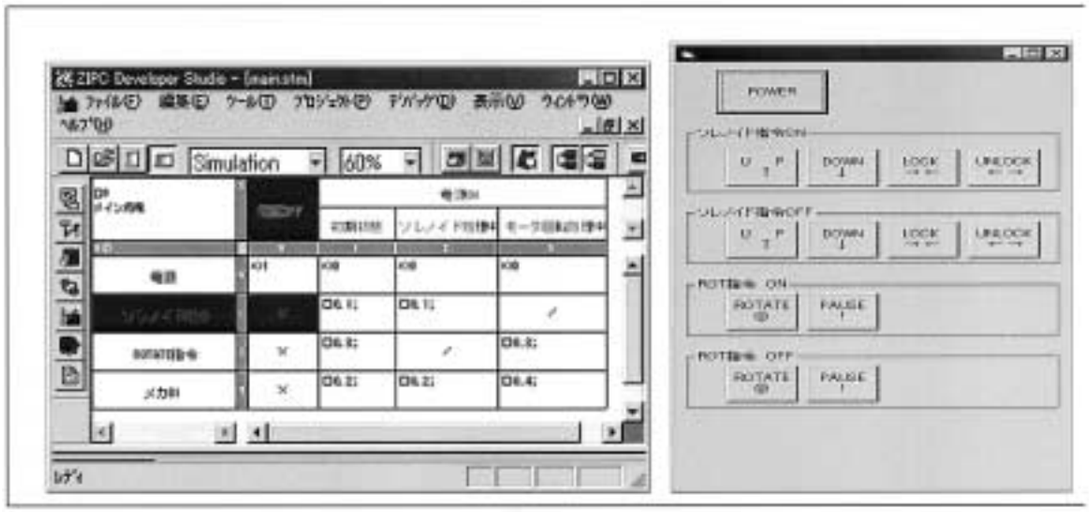
以上の評価結果を考慮すれば、組み込みシステムへのＺＩＰＣの使用は有効である。今後の組込みシステムの複雑化を考えれば、制御システム事業部の業務において、品質向上、開発期間削減のためには、ＺＩＰＣは必要なツールであると考える。

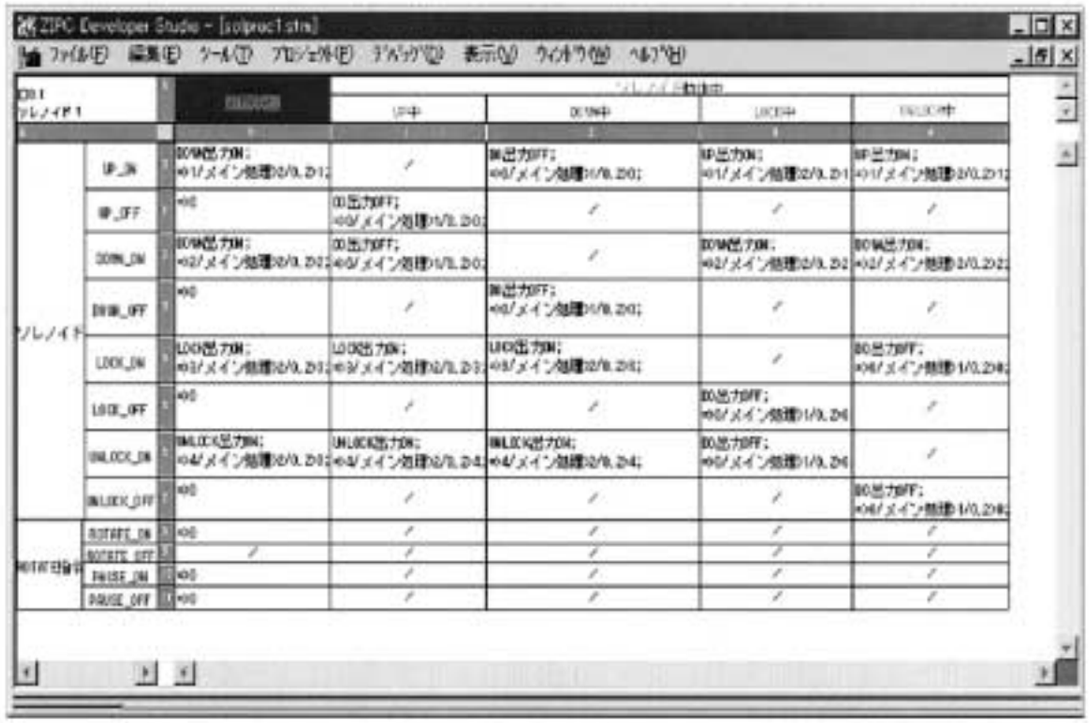
以上

資料１　メインフロー～ソレノイド制御～モータ回転シーケンスのフローチャート



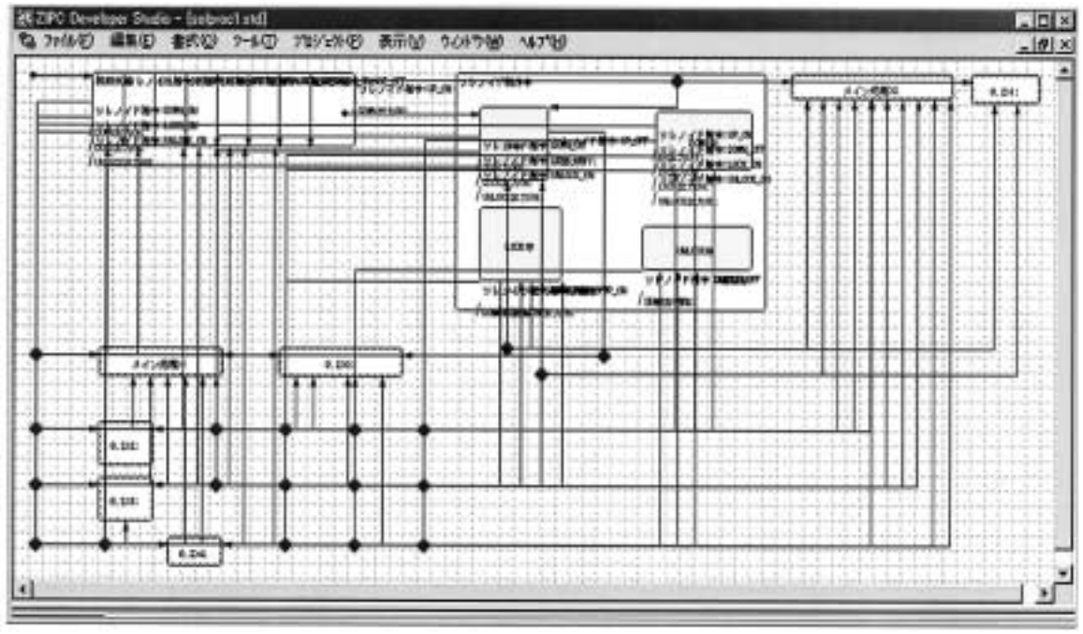
資料２　状態遷移表による設計





ZIPC WATCHERS Vol.8

資料３　STMからコンバートした状態遷移図の例（ソレノイド制御部分）



※状態遷移図のエディタ機能は充実している。ただし、イベントや状態の数が多いと、文字が重なり
　非常に見にくくなるので注意が必要。