LONシステムとZIPCの開発実例

高橋 保典

システム開発をする時に誰もが思うこと、それは洩れの無い設計と開発効率をどのように上げるか、これを設計前に明確にして工程を引けば開発途中に工程表とにらめっこすることや、進捗の遅れを報告して嫌な思いをすることが無くなるはず。この事はシステム開発をした事のある人、誰もが分かっていることなのですが、ほとんどの人が出来ていない、もしこれが出来ていると言う人は素晴らしい才能をもった人かも知れません。だれもが一度は味わった事があると思いますが製作途中に問題が出て、あの時もっと細かく設計しておけばとか、安易に考えすぎたと思った事があると思います、もし無いと言う人がいたらその人はもっと反省する時間をもった方がいいかも知れません。人間は反省することによって成長していく生き物ですから自分へ投資する時間をつくって、反省したほうが少し説教じみた話しになってしまいましたが、これも今から話しするZIPCの素晴らしさを分かってもらえ、今後の開発の手助けになればと思ってのことです。それでは開発効率を上げるCASE TOOLZIPCを私がOEM開発でどのように使用したかをお話していきます。

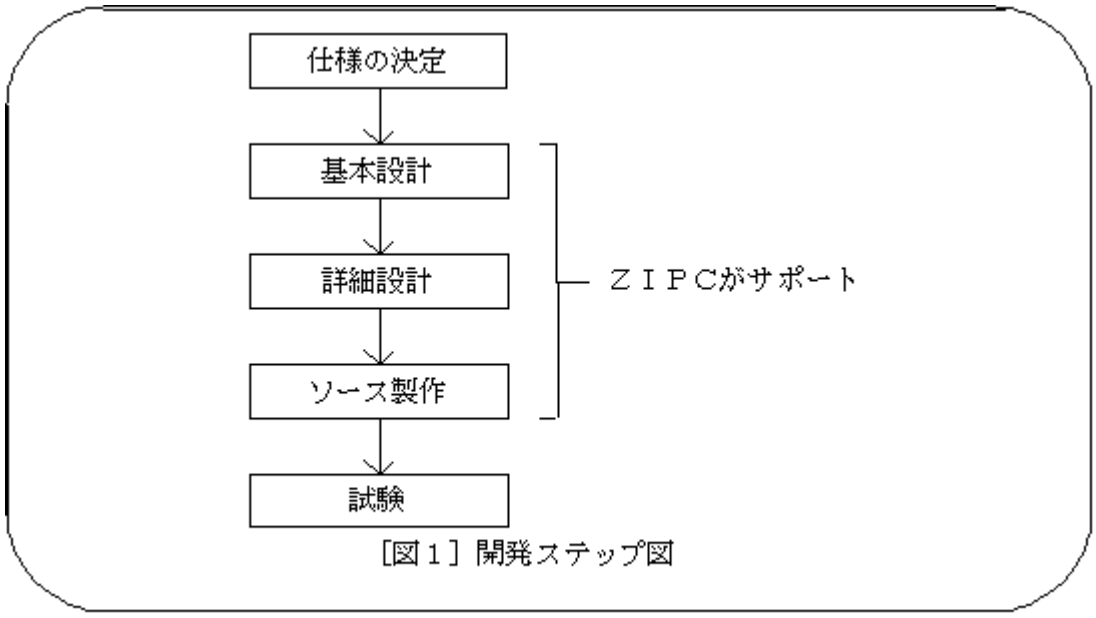
はじめに

ここではZIPCがどのようなシステムに有効かを述べておきます。ZIPCは状態遷移表をベースにしたCASE TOOLで、遷移表を用いて開発するシステムならZIPCを使って開発することが出来ます。つまり、状態が複数存在してイベントドリブン方式のシステムを開発するのであれば使用することが可能と言う事です。
次にZIPCで何をサポートしてくれるかですが、ZIPCを使用してシステム開発を行うと基本設計、詳細設計、ソース製作のステップをサポートをしてくれます。

ZIPCがこのステップをどのようにサポートするのかをこの後、実例を上げて詳しく説明していきます

開発システムの概要

ZIPCを使って開発したシステムは、某電力関係のシステム開発会社から受注した業務で、ダム環境の監視と、動作制御を行うシステムを開発した時のもので



［図１］開発ステップ図

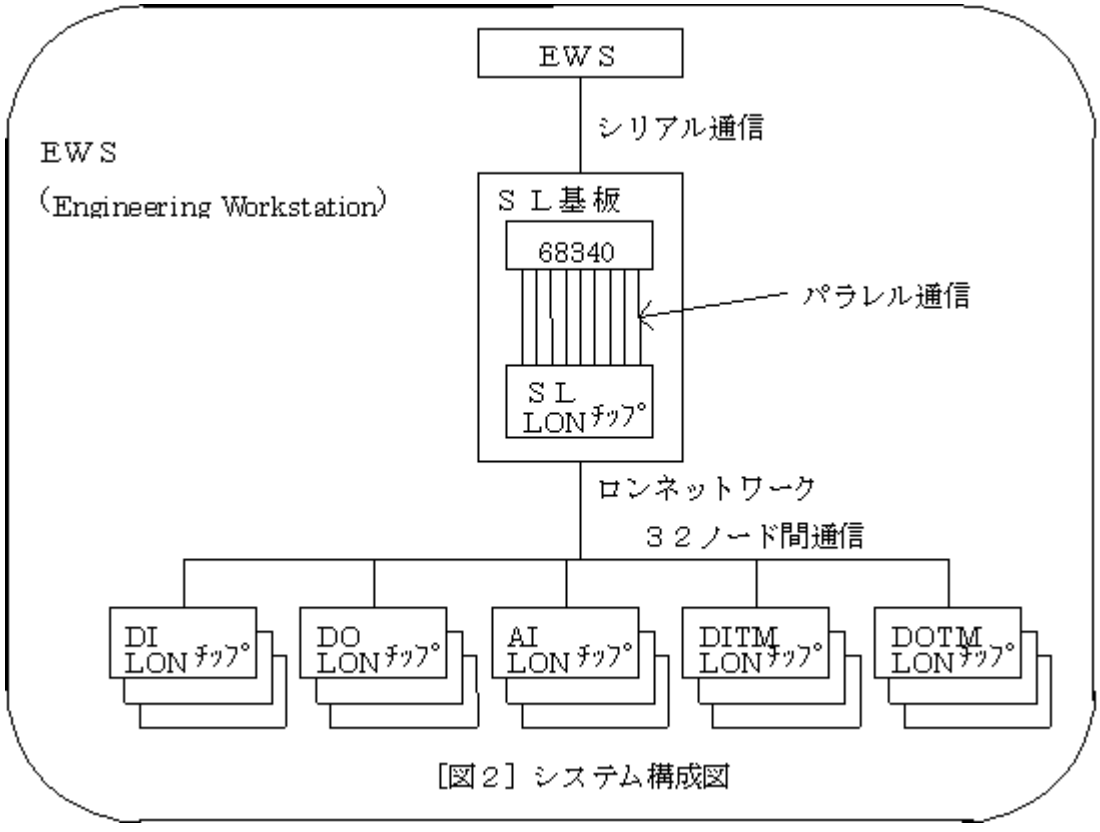
す。そのシステム構成図を［図 2］に記します。
このシステムは、DI(デジタルインプット)、DO(デジタルアウトプット)、AI(アナログインプット)の基板を通して、外部のダム監視又は制御機器とデータの受け渡しをするシステムです。外部から送られて来る監視情報はDI又はAI基板から入力し、SL(リアル変換)基板を通してEWSに伝わります。逆に外部への制御指令は、EWSからSL基板に送られ、DO基板を通して外部装置に送られます。DO基板は設定ピンの設定によりテレメータ(DITM, DOTM)基板として動作するように設計されています。(テレメータは最新の情報を定周期間隔で出力する機能を持っています)
この中で私が開発担当したノードを紹介しておきます。

1. SLノード(LONチップ用アプリ)(68340)からのパラレル入力をロン通信用データに変換、ロンネットワークへ送信する、又は、LONからの入力データをパラレルデータに変換して68340に引き渡します。
2. DIノード　(LONチップ用アプリ)外部からの監視情報80ビットを1ビット単位で受け取りEWSへ通知する。設定ピンの設定を変更する事によりテレメータとして動作し、入力ビットをワード単位又は、2ワード連結してEWSへ通知する。
3. DOノード(LONチップ用アプリ)EWSから送られてきたデータをビット単位で出力する。(全80ビット)設定ピンの設定を変更する事によりテレメータとして動作し、EWSからのデータをワード単位で出力する。
4. AIノード(LONチップ用アプリ)ワード　　単位の入力で全6ワードを入力し、定周期間隔でEWSにデータを送信する。

LON概要

LON:　Local Operating Networksの略称でLONチップ上にアプリケーションを作成して、LON専用プロトコルで複数のLONチップと通信しあうネットワークをLONネットワークといいます。LON:集中型制御ネットワークを構築することもできますが、分散型制御ネットワークというシステム信頼性を上げる優れたネットワーク構築を目的として作られたシステムです。

● 集中型制御ネットワークの特徴
1つのCPUで全ての入出力装置を一括制御することが出来る。但しホストCPUに障害が発生した場合システム全体に影響を及ぼすことになる。
● 分散型制御ネットワークの特徴
ノードが故障した時そのノードを交換する事によってシステム全体の運用に影響をあたえる事は無い。各ノード毎にCPUを積んでいる。



［図２］システム構成図

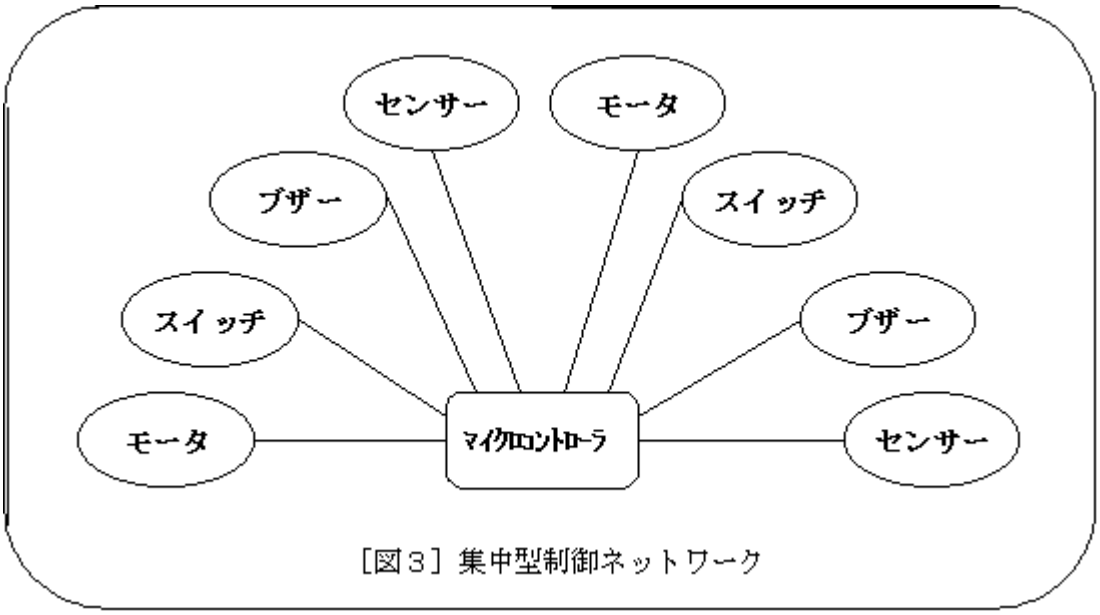
LONシステム開発

LONのシステムを開発するには、LONチップ内部EEPROM（容量1K）もしくは外部ROM（容量42K）にアプリケーションを載せる必要があります。この少ない容量でどのくらいのアプリケーションを載せる事が出来るのかと思うかも知れませんが、LON開発の場合NEURON C（以下ニューロンC）と言われているC言語を使用して容量効率のよいプログラムを組むことができます。ニューロンCとはどのような物かを簡単に説明しておくと、ニューロンCはANSI Cを基本に作られた言語で、分散制御アプリケーション用に拡張機能を加えたC言語です。この拡張機能を使う事により容量効率のよいプログラムを組む事が可能となっています。また、LONはスケジューラをもっていて、イベントの入力があるとLONのスケジューラがイベント内容を解析し、該当するアプリケーションへ処理を移す機能を持っています。

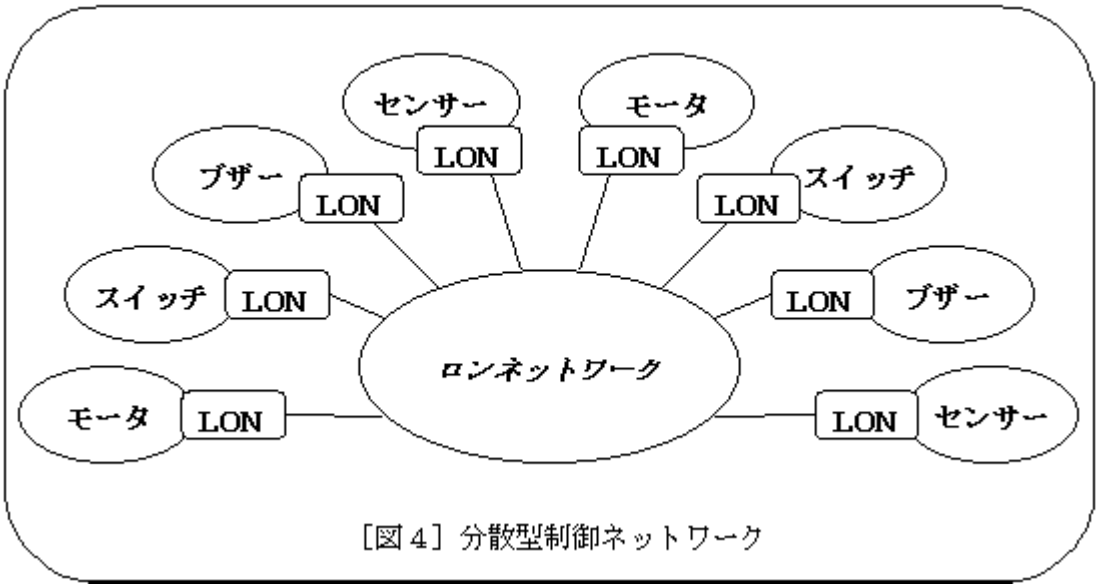
LONには上記図5で示したシーケンスで動作するスケジューラを持っているかわりに、ニューロンCでサポートする関数の中にはメイン関数が存在しませんでは、LONシステム開発でZICを使って開発していくにはどうすればいいのかをこの後の項目で説明していきます。

LONとZIPCのリンク

まずZICを使ってLONシステムの開発が出来るかを考えて見ると、ニューロンCは先程お話したように、ANSI Cをベースにして作られたC言語なので基本的に他のC言語と開発方式が変るということはないのですが、LONの機能としてスケジューラをもっているので、ZICで生成するイベント解析処理とそのスケジューラをどのようにリンクさせるかのほうが問題になりました。



［図３］集中型制御ネットワーク



［図４］分散型制御ネットワーク

しかし、LONのスケジューラ処理が、ZIPCで生成するイベント解析処理に似ていることから、生成したイベント解析処理を多少手直しする事でその問題はクリアされました。

リスト1にLONスケジューラ処理を記述します。ZIPCで生成したイベント解析処理の最後(生成)をwhen(LON用)に変更し、()内のイベント解析条件(ZIPC生成)をイベント条件(LON用)に対応させる事によって、リンクすることができました。

この様に、ZIPCを使用してLONシステム開発を行うことが可能だと言う事が分かっていただけた事として、次の項目ではZIPCを使った開発手順について話していきます。

ZIPCを使った開発手順

本実例開発で使用したZIPCの機能は、STIMエディタ、チェッカ、テキストエディタ、ジェネレータで、これらの機能をどの開発ステップで使用したかを図6の開発手順に記述します。

step1. STIM作成

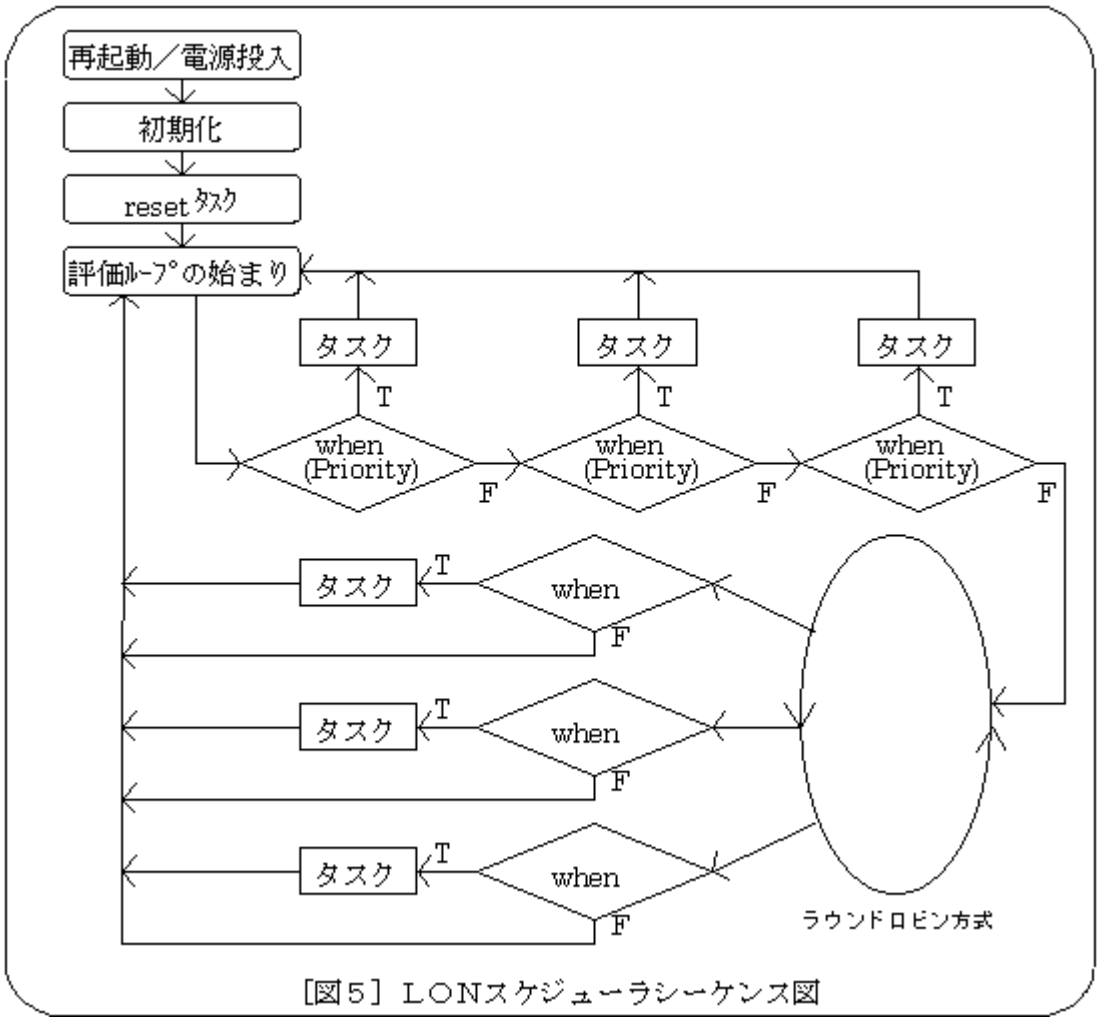
このstep1では、STIMエディタを使用して状態遷移表を作成します。このSTIMエディタを使用して状態遷移表を作成しておく事によって、遷移表の変更が容易にできたり、後のstep3で紹介する機能を使って、遷移処理のソースを生成することができます。実例業務では、イベントの記述方法をLONスケジューラのコーディングに基づいて記述しました。

step2. STIMチェック

このstep2では、先程のstep1で作成した状態遷移表を、チェック機能を使って構文チェックを行います。実例業務では、イベント記述でエラーを検出したが、LONスケジューラコーディング方法で記述している為に出力されたエラーなので無視しています。

step3. 置換情報作成

作成したSTIMの処理内容を、テキストエディタを使ってソースに置き換えるファイル(置換情報ファイ



［図5］ＬＯＮスケジューラシーケンス図

```
when（イベント条件１）
{
        処理１
 }
when（イベント条件２）
{
        処理２
 }
when（イベント条件３）
{
        処理３
 }
          .
          .
          .
when（イベント条件ｎ）
{
        処理ｎ
}
```

ＬＯＮスケジューラ処理［リスト１］

ル）を作成します。STM上の置換するべき情報は、テキストエディタの置換情報デフォルト生成を実行する事によって、置換情報を検出してエディタに表示してくれます。

step4. ソース生成
前stepで作成したSTM記述書と置換情報から、遷移処理及びイベント解析処理を自動生成します。

step5. 生成ソースの変更
step4で作成したイベント解析処理をLONスケジューラ用に（LONとZIPCのリンクの項目でお話した用に）変更します。

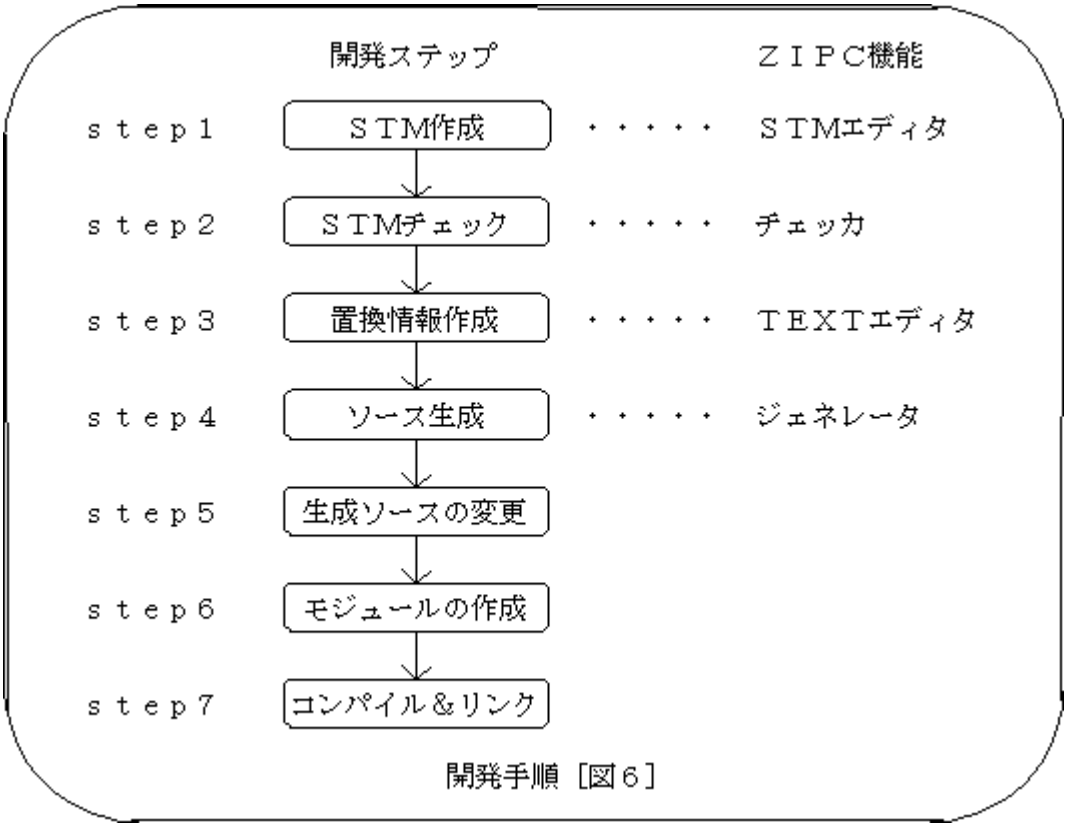
step6. モジュールの作成
このstepでは、遷移処理後にコールするアプリケーションモジュールを、任意のエディタを使用して作成しました。テキストエディタを使ってfm設計書から生成する事もできるのですが、設計時のまとまり具合によっては今回のように直接作成した方が早い場合があります。

step7. コンパイル＆リンク
このstepでは、生成及び作成したソースをコンパイル＆リンクして実行ファイルを作成します。

上記の開発手順は私が実業務で行った手順ですが、業務内容によってはいらないstepや追加しなければならないstepがあるかと思います、そこは臨機応変にZIPC機能を使いこなす事で対応することが出来ます。

業務ボリュームと開発日数

ここまで話してきたZIPCを使った開発で、実業務でのボリュームと開発日数の割合がどれぐらいであったかを記述します。



開発手順［図６］

SLノード

♦ ROMサイズ
LONファームウェア用ROMサイズ ･･･ 186 byte
アプリケーション用ROMサイズ ･･･ 1098 byte
♦ RAMサイズ
LONファームウェア用RAMサイズ ･･･ 712 byte
アプリケーション用RAMサイズ ･･･ 8652 byte
ネットワークバッファ用RAMサイズ ･･･ 9120 byte
♦ 開発日数
設計期間 ･･･ 4日間
製作期間 ･･･ 2日間
単体試験期間･･･ 2日間

DIノード（Dデレメータ機能含む）

♦ ROMサイズ
LONファームウェア用ROMサイズ ･･･ 186 byte
アプリケーション用ROMサイズ ･･･ 13276 byte
♦ RAMサイズ
LONファームウェア用RAMサイズ ･･･ 744 byte
アプリケーション用RAMサイズ ･･･ 1958 byte
ネットワークバッファ用RAMサイズ ･･･ 5572 byte
♦ 開発日数
設計期間 ･･･ 6日間
製作期間 ･･･ 3日間
単体試験期間･･･ 2日間

DOノード（Dデレメータ機能含む）

♦ ROMサイズ
LONファームウェア用ROMサイズ ･･･ 186 byte
アプリケーション用ROMサイズ ･･･ 5088 byte
♦ RAMサイズ
LONファームウェア用RAMサイズ ･･･ 744 byte
アプリケーション用RAMサイズ ･･･ 796 byte
ネットワークバッファ用RAMサイズ ･･･ 5572 byte
♦ 開発日数
設計期間 ･･･ 4日間
製作期間 ･･･ 1日間
単体試験期間･･･ 2日間

AIノード

♦ ROMサイズ
LONファームウェア用ROMサイズ ･･･ 186 byte
アプリケーション用ROMサイズ ･･･ 49 60 byte
♦ RAMサイズ
LONファームウェア用RAMサイズ ･･･ 747 byte
アプリケーション用RAMサイズ ･･･ 5572 byte
ネットワークバッファ用RAMサイズ ･･･ 284 byte
♦ 開発日数
設計期間 ･･･ 4日間
製作期間 ･･･ 2日間
単体試験期間･･･ 2日間

開発日数を見ると、開発期間が短いことが分かると思います。設計ではSTMエディタを使用して効率よく遷移表を作成して、製作では基本的にコンパイルの通り遷移処理をジェネレータが自動生成してくれるので、単体試験では遷移処理以外をデバッグすればいいのです。このようにZIPCを使って開発日数を短縮することが出来ました。また、ここまではZIPCを使った時のメリットだけをお話してきましたが、この実例の開発で、実際にZIPCを使って私が感じたメリットとデメリットを次の項目に記述しておきます。

ZIPCのメリット、デメリット

メリット

♦ STMの作成及び変更時間が短縮。
♦ STMの記述をチェックすることが出来る。
♦ 遷移処理部分の製作ステップが無くなる。
♦ 生成ソースのコンパイルチェック又は動作確認試験等のステップが無くなる。

デメリット

♦ ＺＩＰＣ機能を使う時の設定が多すぎる。
初めてＺＩＰＣを使う人は、説明書で操作を覚えるより、セミナーを受けて操作を覚えることをお進めします。
◆ 詳細レベルの設計機能が、場合によっては使いにくい。（設計時のまとまり具合による）
ＳＴＭ設計を終了した段階で、アプリモジュールが頭の中でまとまっている時は、ＦＵＮＣ設計書を作成するよりモジュールを直接製作したほうが早い。

最後に

第３回ＺＩＰＣユーザー会の時に、この実例内容を PowerPoint を使って発表する予定になっています、開発でＺＩＰＣを使っている、使おうとしている、興味がある方は、ぜひ参加してください。

高橋 保典（たかはし やすのり）