VoIP製品におけるZIPCの活用と PC上のプロトタイプの有効性

サクサシステムエンジニアリング株式会社 技術本部 第一プロジェクト 調査役
十日市　勉

●はじめに

当社 サクサシステムエンジニアリング㈱は、電話をはじめとする情報通信機器のソフトウェアを開発しています。年々規模が大きくなるソフトウェアの開発に対して開発効率の向上と品質確保のため、ZIPCを導入しました。

その導入時にZIPC WATCHERSに経緯や導入手順について記事を寄稿しております。(ZIPC WATCHERS Vol.9「情報通信機器におけるZIPCの適用 ～ZIPCリターン～」) 是非、そちらもご参照下さい。

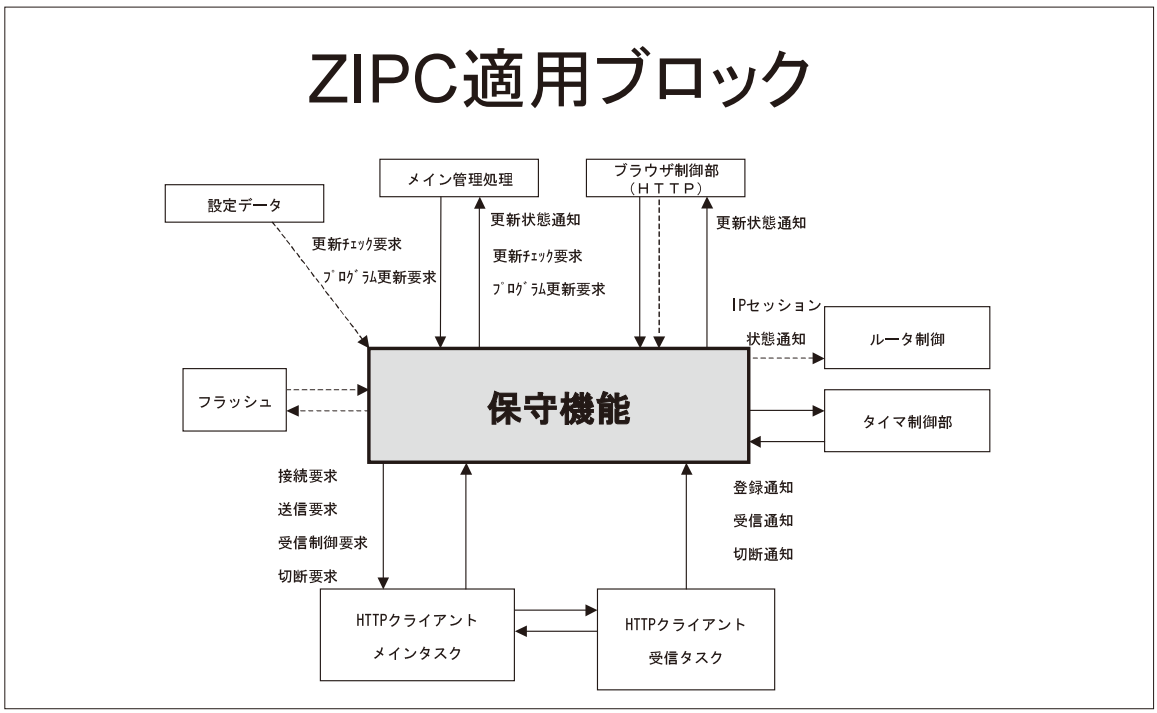
その時の記事では導入経緯とパイロットプロジェクトへの適用が主な内容でしたが、今回は製品プロジェクトへ適用した結果についてご報告いたします。

また弊社の組込み用ソフトウェア開発で品質の向上、開発効率の向上として大きな成果を上げている組込みソフトウェアのPC上プロトタイプの有効性についても紹介いたします。

VoIP製品におけるZIPCの活用

●どこに適用するか？

製品への適用では、VoIP製品の保守機能部分の状態遷移に適用しました。その理由として大きいのは、この部分が新規の機能ブロックになるため既存部分の影響を受けにくく独立した開発が可能なためです。更に、保守機能部分は仕様変更が多く、状態遷移を変更する毎に発生する設計書の変更、プログラム修正、デバッグ作業を効率化したいことも理由でした。



●ZIPCの利用範囲

ZIPCは以下の機能を利用しました。

・状態遷移表作成（デザインフェーズ）
・状態遷移ＤＲ（シミュレーションフェーズ）
・Ｃコード生成（ジェネレーションフェーズ）

適用部分の機能ブロックは制御処理が主体であり、マンマシンを伴わないためVIP（Visual Interface Prototyping）などのビジュアル表示する機能は使っていません。

●実際に実施した作業

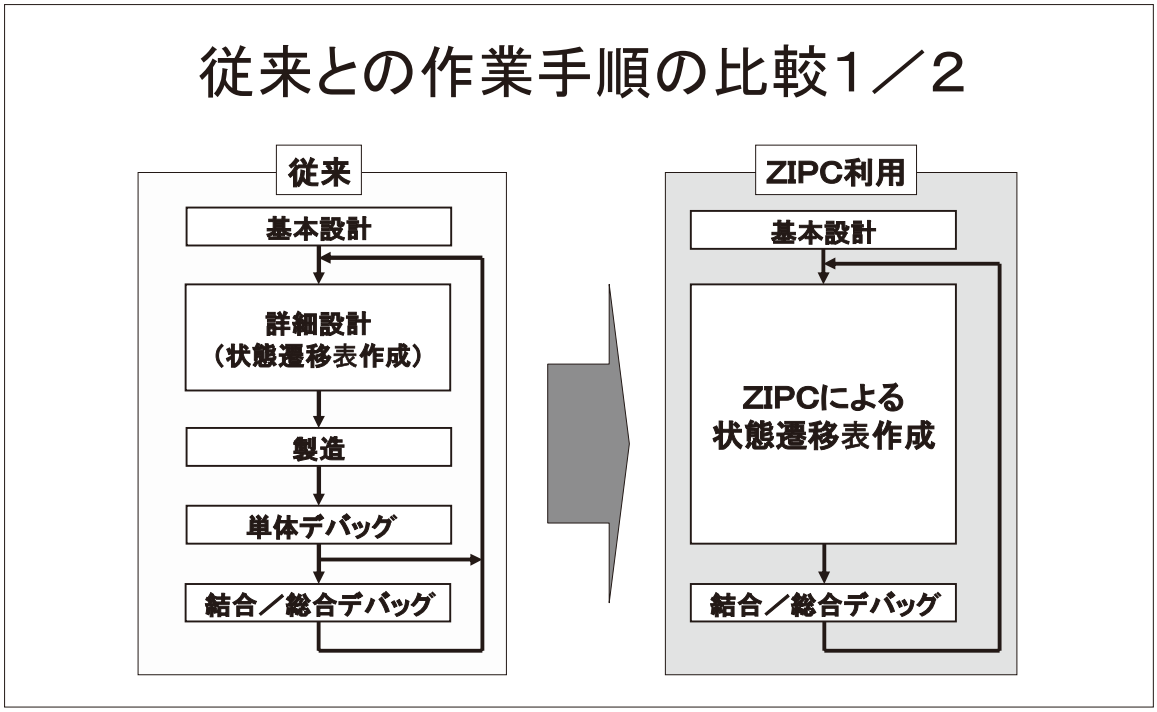
保守機能ブロックにZIPCを適用するために実施した作業は以下の通りです。

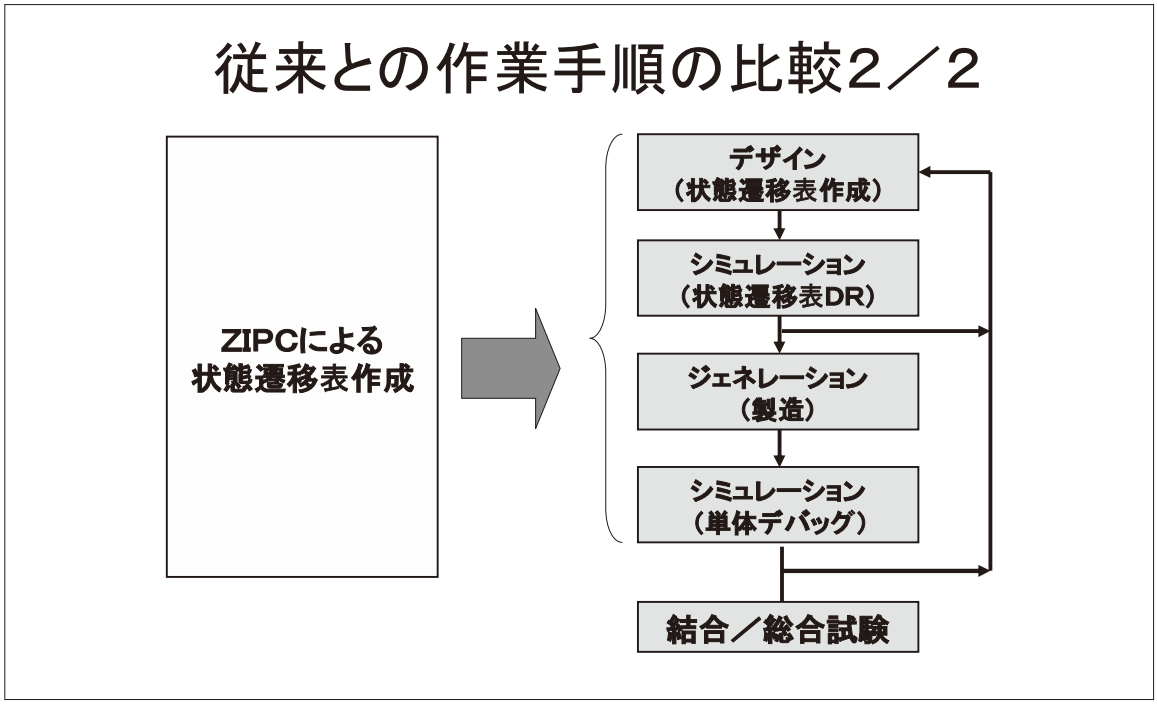
1）状態遷移表の作成
2）各処理のコーディング
　　ここは通常のCコーディングと同じ
3）シミュレーションコンパイル
　①状態遷移表のエラーやワーニング除去
　②システム共通ヘッダの指定（ヘッダのパス指定）
　③外部から提供される関数のダミーを作成
4）シミュレーション
　①イベントを発行して各状態の処理を網羅
5）日本語記述をＣ記述へ変換する変換表の作成
6）Ｃコード生成

これらの作業を行っていましたが、パイロットプロジェクトでノウハウを蓄積してあったので特に大きなトラブルはなく、生成されたCコードをターゲットに実装することができました。

●ZIPCにより変わったこと

このプロジェクトではZIPCを適用することに必死になっていましたが、実際にZIPCのプロジェクトが出来上がると、開発スタイルが大きく変わっているのに気づきました。いままで当たり前のように設計書の修正、コーディング、単体デバッグ、結合／総合試験をしていたものが、ZIPC上で多くの作業を済ませられるようになりました。





●製品へのZIPC適用成果

実際にVoIP製品の保守機能にZIPCを適用した結果について、生産性と自動生成コードの大きさについて従来手法と比較を行いました。また、その他に担当者が感じた効果についても示します。

1）生産性データ

	従来手法(予測)	ZIPC 使用	差分
状態遷移設計・作成	４０H	６４H	＋２４H
状態遷移レビュー	３４H	１７H	－１７H
製造	３２H	２４H	－８H
単体試験	４０H	２９H	－１１H
仕様変更など修正	１２H	８H	－４H
合計	１５８H	１３２H	－１６H

状態遷移表の作成に時間がかかっていますが、これはツールに慣れる時間も含んでいます。ツールに慣れることでこの時間は短縮することが出来るようになります。また、バージョンアップであれば、コード生成用の環境を作る時間がないのでもっと時間の短縮が可能です。

小さい機能ブロックにZIPCを適用していますが、10%程の工数削減につながっています。ここでポイントになるのが、小規模なプロジェクトではCASEツールを導入しても導入教育やツールに慣れるまでの時間がかかりすぎて成果を望めないケースが多いのですが、ZIPCの簡単さは修得に時間がかからないため小規模な開発でも効果を得ることができています。

2）自動生成コードのサイズ

	従来方法	ZIPC 使用	差分
ステップ数	３．９ＫＳ（見積もり）	３．９ＫＳ（実測）	０

コード生成についてはオプション設定である程度サイズは変わりますが、ZIPCを使ったからといってコードが大きくなることは殆どないと判断できました。これは、組込みで使う場合には特に大きなメリットになります。

3）その他の効果

実際に製品プロジェクトにZIPCを適用した担当者は以下のような効果を感じていました。

①遷移先のない状態や使用していない関数は、シミュレーションコンパイルでZIPCより指摘されるので事前にバグを検出できる。

②状態ロックはシミュレーションにより比較的容易に発見できるので、結合デバッグ以降でのバグの発生を防げる。

③シミュレーションによりすべての状態とイ

ベントの組み合わせを試験できるので、結合デバッグ以降で通りにくいルートのバグ発生を防げる。

④単体デバッグ以降に仕様変更があって状態やイベントを追加する大幅な変更を行ったが、ZIPCの備える機能（状態遷移先を自動補正）によりそれほどのインパクトはなかった。

特に④の仕様変更への対応が楽になることは、現場のエンジニアにはとても助かる要素です。

●メリット・デメリットのまとめ

実際に製品プロジェクトにZIPCを適用したことで感じたメリット、苦労点、改善点を以下にまとめます。

1）良い点

生産性向上の要素	・状態遷移表のDRの代わりに、シミュレーションで状態抜けやリンク切れを確認できるため状態遷移レビューにかかる時間の軽減が可能となる。 ・ソースコードを自動的に生成できるため製造にかかる時間の軽減が可能となる。
品質向上の要素	・視覚的な状態遷移からソースコードが生成されるので状態遷移からソースコードを製造するときのミスをなくすことができる。 ・どの状態でも任意のイベントを発行できるため、すべての状態遷移を通過させながらデバッグできる。
メンテナンス性の向上要素	・状態とイベントの増減は容易に可能で、状態遷移先は自動で補正される。 ・シミュレーションを行うことにより普段見落としがちな変更した箇所以外の部分を含め短時間で再確認できる。 ・状態遷移の内部処理を修正しない場合、コーディング作業は全くいらない。 ・ソースコードが生成されるので、設計書とソースに差異がない。 ・設計書（状態遷移表）は、バージョンアップがしやすい。 ・設計書（状態遷移表）は、類似機能への流用がしやすい。

品質に対して数値的な計測はされていないが、シミュレーションでバグが取り除かれるため、結合試験以降へ持ち込まれるバグは明らかに減少しているとのことでした。

2）苦労点など

苦労した点	・使い慣れないと様々な矛盾を発生させてシミュレーションコンパイルを完了するのに非常に苦労した。 ・システムの一部へ適用している場合、シミュレーションデバッグするために各種ダミー関数を用意しなければならないのが面倒であった。 ・Cソースコード生成では各種設定が必要であり苦労した。 ・大画面のディスプレイでないと、処理の流れを掴みにくい。 ・ZIPCをインストールしたPCでないとバグを修正できない。 ・プロジェクト内のファイル名やディレクトリを後から変更するのは大変。
改善されると良い点	・状態遷移表をExcelに変換できると良いが現状できない。（PDFは可） （Viewerはあるが誰でも分かるExcelで使いたい時がある）
対外的要素	・ZIPCを適用して開発することを発注元へ使用することの確認をとる必要がある。 実質、ZIPCがないと修正が出来ないので、弊社以外でバージョンアップ開発を行う場合に問題になる。

シミュレーションやコード生成は設定などが難しくなり苦労する要素になります。

ZIPCを使用することを発注元に確認する点は導入当時の問題であり、このプロジェクトで問題がなかったことから、今では簡単に許可されるようになっています。

●ZIPCの適用の条件

パイロットプロジェクトや製品プロジェクトへZIPCを適用し、ZIPCの機能や能力が分かったところで、今後どのようにZIPCを適用することが出来るかを分析してみました。

開発ターゲット	適用の可能性
既存システムを置き換え（既に作成されている処理部を変換）	・既存処理部の状態遷移表からZIPCへの移行は**困難**だと思う。 ・実現は可能であるが、詳細設計からテスト工程までは新規開発時と同等の工数が必要になってしまう。
既存システムへの導入（処理部の追加）	・追加処理部をZIPCで作成することにより、追加処理部の状態遷移シミュレーションが可能になるので**有効**である。 ・既存システムの外部提供関数などは、ダミー関数を作成する必要がある。
新規開発システムへの導入（特定処理部への適用）	・特定の処理部をZIPCで作成しても、状態遷移シミュレーションなどは可能なので**有効**である。
新規開発システムへの導入（全処理部への適用）	・全ての処理部をZIPCで作成することにより、システム全体の状態遷移間のシミュレーションが可能になる。 ・ZIPC状態遷移表AとZIPC状態遷移表Bを結合してデバッグが可能となり、結合試験レベルがシミュレーションで確認できるため**最高の効率を得られる**。

●現在の状況

ここまで、製品プロジェクトへZIPCを適用した最初の例を示しましたが、その後は製品プロジェクトへ順次適用しています。

全てのプロジェクトがCソースコード生成をしている訳ではなく、以下のように使う範囲を使い分けています。

①既存ソースの影響が大きいターゲットではドキュメントとしての利用

②効果を見込めるプロジェクトではCコード生成を利用

最近では、VoIP製品の**中枢機能**である**呼制御**にも適用されるようになり、把握しにくい振る舞いの**見える化**に貢献しています。

ZIPCの導入は、導入推進者がサポートを行いながら進めていましたが、導入推進者が知らないところでも使われておりビックリすることがあります。一度使ったエンジニアは、次の開発でも使いたがることが要因で、以下の理由があります。

①Excelに比べて専用ツールなので気が利いており楽に使える

②状態、イベントの追加が楽なので考えながら使える

③ZIPCを使えると自分の能力評価も良くなる。

この2年の間にZIPCの適用が良い方向で進んできました。今は、ZIPC利用のプロジェクトが同時に複数進行しないかハラハラして見ているところです。ライセンスが不足しそうです。

PC上のプロトタイプの有効性

●組込みソフトの開発は効率が悪い？

時々、組込み開発部門の人が足りなくなるとPC部門から人を借りてきて手伝ってもらうことがあります。そのとき、そのメンバーが必ず漏らす言葉が「効率悪い！」です。特にデバッグの環境がビルド→ローディング→使い勝手の悪いデバッガでデバッグになり、耐えられないと言います。

例えばMicrosoft Visual C++ なら以下のようなメリットがあり、効率よくデバッグが出来ます。

・統合環境が強力でソースのブラウズが早くて簡単

・差分コンパイル、差分リンク機能などがあり大きなシステムでもコンパイル、リンクが早い

・メモリーリークチェックなどのエラーチェック機能がある

・サードパーティ製のサポートツールが安くて豊富

しかし、最初から組込み開発しかやっていないエンジニアにとってはそれが当たり前であり、PCの環境には興味を持つことがなく、自分では効率の悪いことに気が付いていないと言えます。

●弊社のPCプロトタイプの起こり

弊社では組込みソフトウェアをPC上でデバッグする方法を自主開発して対応しました。しかし、これは計画的に開発した物ではなく、必要に迫られて実施したものでした。

始まりは自社ブランドのビジネスホンを開発する際に、ユーザの設定機能のデバッグのために作成しました。

10年間で作り上げたユーザ設定機能を全て捨て、携帯電話のような操作性に作り替える必要がありました。しかも、10年分の内容を2ヶ月で設計から結合デバッグをすまさないといけません。

対応方法としては、同時多人数による設計とデバッグです。設計は何人でも可能ですが、デバッグについては、エミュレータ付きの実機を大量に揃えることは物理的に不可能でした（エミュレータを制御するサーバーが制御出来るデバッグ環境は5台が限度）。

このような状況で**画期的なデバッグ方法を編み出す必要が発生**しました。

悩むことなくデバッグ環境の構築はVC上に構築することを決定しました。VCであればPCグループにライセンスが多数あり、多くの環境が作れます。

しかし、設定機能部分のデバッグ用とはいえ、既存のビジネスホンソフトと結合動作が出来ないとデバッグの効率は悪く、実機に近い動作確認が出来ません。既存のビジネスホンのソフトウェアもPCで動作できるようにする必要があります。

これが一番のハードルです。ビジネスホンのソフトウェア構造を熟知し、且つ、VCの使い方

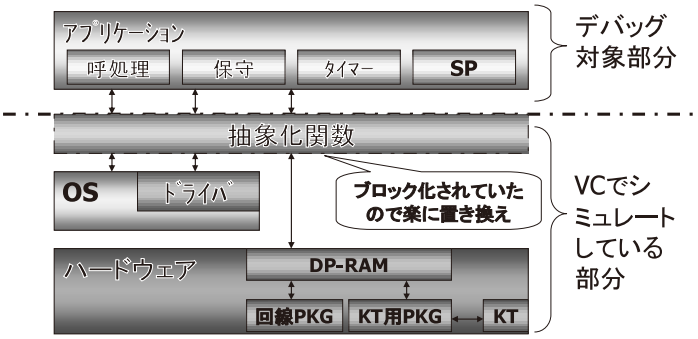
も熟知した担当者が必要になります。普通はこんな担当者はいません。VCが得意でも600KStepを超えるビジネスホンソフトの構造を数日で把握するのは無理でしょう。また、組込み担当者がVCを使いこなすのも時間がかかります。

この時は、ちょうど両方をこなせる担当者がおり、およそ１週間で既存のビジネスホンソフトをVC上で動かせるソフトウェアに改造することに成功しました。（簡単ではありません。知識だけでなく最後は根性もいります。）

●既存のソフトウェア構造が良かった

以下はそのときに作成したビジネスホンのソフトウェア構成です。

［ビジネスフォンソフト構成図］



アプリケーション部分がデバッグ対象になる部分です。このビジネスホンソフトの特徴はOS、ドライバなどのハードウェアOSへのアクセスを全て、これが抽象化関数で構成された１つの機能ブロックで吸収していました。また、共通関数はエンディアンの違いまで吸収出来るように考慮されていました。

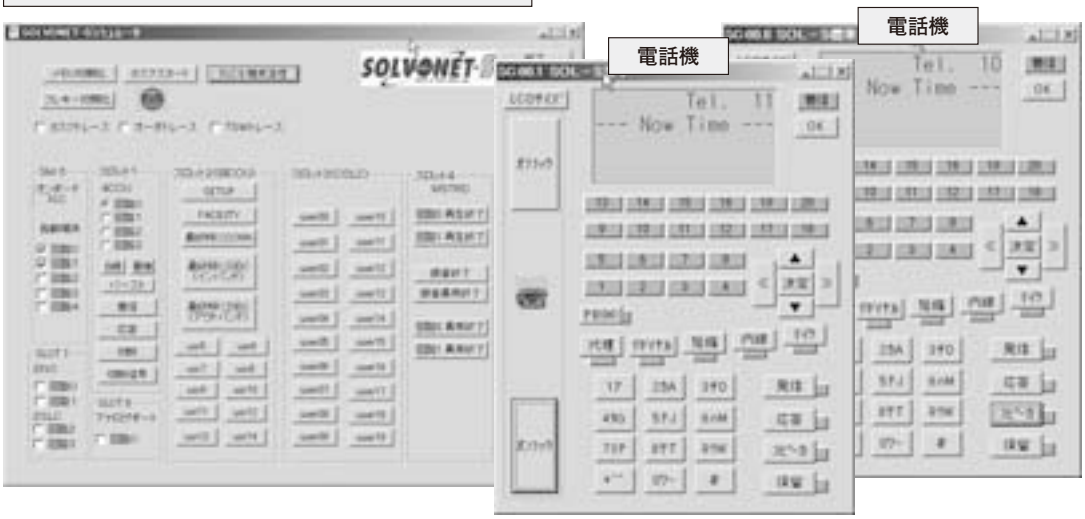
このような構造であったことが功を奏して、この抽象化関数ブロックの関数を、OS用であればWindowsのアクセスに変換、ドライバアクセスをGUIに置き換えることで対応が出来ました。これは運が良かったと思います。

抽象化関数（共通関数）には、一部アセンブラ部分がありますが、これもVCのインラインアセンブラを駆使して対応しました。

●どれだけ実機に近づけるかポイント

以下は、完成したプロトタイプの画面です。

主装置（制御ソフトは全てここにある）



見ての通り、デザインは悪いのですが、電話機も出来るだけ本来のものに近づけてあります。この、マンマシンに関する部分を出来るだけ本物と同じように再現することがプロトタイプでは重要です。

以前、ソフトウェアだけを動作させて、実行結果はメモリダンプとメッセージで確認するような方法を試したことがありますが、実機でデバッグしてから表示のバグが大量に残っておりあまり意味がなかったことがありました。その反省を踏まえてこの部分はこだわって作ったものです。

内部構造も実際のハードウェアの制約に基づいて作ってあります。その結果、LCD表示データの微妙な編集ミスで発生するバグまでこのプロトタイプ上で取り去ることが出来ました。

このようにマンマシン部分だけでも効果が大きいので、是非、マンマシンを伴う開発ではプロトタイプによる開発をお勧めします。組込みとVC両用できるエンジニアがいない場合は、ZIPCのVIP機能を使用する方法もあります。またDrawrialのようなバーチャルモックアップで確認出来る物も有効と思います。設計レベルのツールですが、設計レベルでマンマシンのバグを検出してあれば、ソースコードのレベルでのバグは少ないはずです。

●PC上プロトタイプの効果

このPC上プロタイプの効果は以下のようなものでした。

①多人数、同時デバッグ	・エミュレータによる実機デバッグ環境は5台。その他に12人がプロタイプでデバッグを進行。 ・これにより、工期短縮と設備費用の節約ができ、更に実機デバッグしかできないサブシステムに多くのマシンタイムを提供できた（特にドライバ担当）。
②ハードウエア入手前デバッグ	・ハードウェア入手前に新しいハードウェアの機能試験を実施 ・新端末の試作入手前に、新ハード特有部分のデバッグを行って、ハード待ちのタイムロスを無くした。
③VCによる強力なワーニング、エラー検出	・PC系のコンパイラは組込み系よりチェックが厳しいため、怪しそうなコーディングを発見しやすい。 ・この結果早期に品質向上が出来た。

どれをとっても、開発期間を短縮するのに大きく貢献する内容でした。

また、生産性に関する結果は以下です。

- 設定機能開発メンバーの総工数
  ： 約6047 h
- 設定機能モジュールの総Step数
  ： 約60Kstep
- 開発効率：60Kstep×（160h/6047h）
  ≒1.58 Kstep/160h

通常、組込み新規の場合 0.5〜0.8Kstep程度（当時）ですが、２〜３倍の生産性があったことになります。しかも、これは基本設計〜総合試験の生産性であり、製造〜試験では４倍以上の効率UPになっていました。組込みでありながらPCソフト開発並みの生産性を手に入れることが出来たことになります。

この生産性の良さを考察すると大きな要因は以下でした。

①VCによるデバッグはライフサイクルが非常に短い。
- 実機デバッグでは、バグ修正１回につき修正、コンパイル、リンク、ローディングに最低５分を要す。
- VC環境では、 **10分の１の時間**30秒程度で済む。

②VCの操作性が良い（デバッグが早い）
- ステップ実行、ブレークポイントの設定、変数の参照が簡単。
- プログラムを終わらなくても、実行開始位置を簡単に変更出来るので、異常ルートのデバッグが簡単。

③既存ソースも動作可能としたため、動きがきわめて実機に近い。
- **実機と同じ可視表示が実現されているため、実機で発生する表示のバグを早期に発見できていたことにより、ライフサイクルの短さと相まって早期にバグを収束できた。**

●最近の取り組み

このプロタイプによるデバッグ方法は大きな改善につながることから、現在多くの組込みシステム開発でPC上のプロトタイプを作成して単体デバッグを実施するようになっています。また最初からPC上で動かすことを考慮して、ソフトを製造すると、出来上がった物をPC用に改造するより簡単にプロトタイプ環境を作ることが

出来ます。

ソフトウェアがPC上で動かせるメリットを生かして最近では以下のような取り組みも行っています。

①ＰＣソフト用のエラー検出ツールの利用

・コーディングチェッカー、実行時エラー検出などは、組込み開発用に比べて、ＰＣ用のツールは高機能で価格も安いのがあるので気軽に利用できます。

②ＰＣソフト用のカバレッジツールを利用

・前述のエラー検出ツールとカバレッジを同時に使うことで、あまり動作しないエラールートの、アクセス違反、メモリーリークを発見することが出来ます。

・カバレッジテストは時間もかかり大変ではありますが、検証結果のエビデンスとしては非常に説得力があるものになります。

弊社では検証ツールとして以下のソフトを使用しています。

「日本Compuware　DevPartnerStudio」
実行時エラー、メモリーリーク検出、カバレッジ等の機能を持ちます

●ダマされたと思ってやってみましょう

最近、Linuxの環境が多くなり、組込みの開発環境も進歩してきています、Windowsでのプロトタイプが必ずしも最良と言えないかも知れませんが、PC上プロトタイプの効果は絶大です。キャッツさんからも、これらをサポートするツールがありますので、まだPC上のプロトタイプによるデバッグを行っていない方は是非試して頂きたいと思います。

おわりに

何かある度にキャッツさんからZIPC WATCHERSの寄稿を求められます。いつも断れません。決まり文句になりますが、この程度の内容でも皆さんの役に立てば幸いです。今後もネタが切れないように頑張りたいと思います。