ＺＩＰＣの導入

パイオニアコミュニケーションズ株式会社 技術部 小原 幹彦

■ 導入した経緯

個人的にリアルタイムシステムのソフト設計において状態遷移表は有効であろうとかねがね思っており、ZIPC に出会った時は「これは使えるかな」と思っておりました。しかし、当社において使用する言語はアセンブラであり、単一商品の開発である事から従来のソフト資産の改造設計が主たる作業となっているため、なかなか ZIPC を導入して設計を開始するというチャンスに恵まれませんでした。

ところが、当社の製品であるコードレス電話機においても、世の中の流れに逆らえずソフトウェアの容量が飛躍的に増加したため、従来の 8bit ワンチップマイコンの ROM 容量内に納まらなくなり、16bit ワンチップマイコンへの移行が余儀ない状態となりました。そこで、言語についても将来の作業の効率化を考え、C 言語を採用することとしました。また、元々状態遷移表に当てはめやすい構造をしていた「ダイヤルデータ受信・解析送出処理」の仕様を変更することになり、処理を作り直す必要が生じたため、C ソースコードが自動生成される ZIPC との親和性が良くなったことにより、どうせ作り直すだから試してみようというのがきっかけでありました。

＜利用した ZIPC Ver.4.0w の機能＞

- STM エディタ
- テキストエディタ
- チェッカ
- ジェネレータ

シュミレータや ATV も所有していましたが、設定が少し面倒そうである他、実際のＣコードを生成する記述内容とは違う設定等が必要だったためと、当社の場合実機の方が先に完成していたため、実機デバックの方が効率が良いと判断したため今回は使用しておりません。

■ ZIPC 構成詳細

＜設計書構成＞

作成した設計書の構成は、以下の通りとなります。

- 状態遷移表
  17 ファイル
- IF 設計書（事象分岐）

1 ファイル
・FNC 設計書(関数)
1 ファイル
・TEV 設計書(インメイル送信定義)
1 ファイル
・REV 設計書(インメイル受信定義)
1 ファイル
・EVT 設計書(インメイル構造体定義)
1 ファイル
・日本語変換用ファイル
3 ファイル

＜ソースコードファイル構成＞

上記で作成した設計書により、Cソースコードを自動生成させると主要なものとしては以下のファイルが生成されました。

・STM 操作処理
・イベント解析処理
・インメイル関数処理
・STM 遷移関数処理
・STM 処理関数
・STM テーブル
・ユーザー関数処理
・ベース部メイン関数

上記のファイルで以降のコンパイルでの都合や対象の処理部は、システム全体ではなく部分的な処理であるので1つの関数としなければならないことから、「STM 操作処理」と「ベース部メイン関数」については自動生成後一部手直しをしています。但し、このファイルはデバック過程で設計書が更新されても変更の必要がない部分であるので、手直しは1回のみでした。

＜ROM／RAM 容量＞

ROM 容量　約 20K バイト
RAM 容量　50 バイト未満
（ZIPC 制御上部分のみ）

ROM 容量については、処理内容の変更や機能追加、C 言語による増加も考えられ単純には比較できないが、従来の約倍の容量となってしまった。設計段階で状態遷移表の簡素化を繰り返してきたが、現時点でもまだ工夫できる点があるので、もうすこしコンパクトになる可能性はある。

RAM 容量については、詳細の分析は行なっていないが現状では特に気にならない程度と考えている。

■ 利用結果まとめ

1）利用して満足な点

➔ デバック時、システムの振る舞いが見やすいため問題箇所の発見が早くできたと感じており、一度デバック

した部分の完成度は従来の方法よりも高いのではないかと感じています。

➔ ZIPC のメリットである「状態遷移表の作成が容易」「抜け漏れのない設計」「状態遷移表の階層化」「日本語対応」は充分感じられるものであったと感じています。

2） 利用して不満足な点

ユーザー関数の記述で日本語を使える点は良いのだが、処理自体はいわゆる C 言語の形でほとんど全て作成しなければならないので、効率化や信頼性の点ではさほど従来の方法と大差が無かった。ユーザ関数作成にあたっても何らかのツールがほしかった。

状態遷移表での表現に引数や戻り値が使えなかったため、関数の処理の結果で遷移先を変えたい場合、そのための外部変数の定義が必要。

3） 今後利用する上での注意点

➔ 状態遷移表の枠の数が多ければ多いほど予想以上の容量になってしまう可能性があるので、できるだけコンパクトになる様に状態と事象の数を極力押さえる工夫が必要。

➔ 日本語を使う場合は、あらかじめ日本語の定義を先に明確にした方が良い。後で日本語の定義を行なおうとすると、同じ内容にもかかわらず、微妙に違う表現が多量に発生してしまい、整理整頓がかなり面倒くさいことになってしまう。

➔ 状態遷移表からＣソースコードへの生成構造を十分理解してから状態遷移表を作成しないと、勘違いによるバグが発生する可能性がある。

簡単ですが、以上で今回 ZIPC を導入してのまとめとします。ZIPC ももっと便利になると良いと思いますが、今回ツールを利用した効果はあったと判断します。

（おばら みきひこ）