# 時計の仕様検討段階におけるVIPの活用

シチズン時計株式会社　時計開発本部　システム開発部　開発技術担当部長
榎田　道弘

## 1．時計の操作・表示手段

操作（入力）手段としては、PB（プッシュボタン）、りゅうずがあります。PBでは短押し、長押し、りゅうずでは0段、1段、2段における右回転、左回転等の入力ができます。表示（出力）手段としては、針（アナログ）、LCD（デジタル）がありますがいずれも通常のIT機器に比べ制約が強く、時計の仕様をどのように操作・表示手段に割り振るかが仕様検討において問題となります。

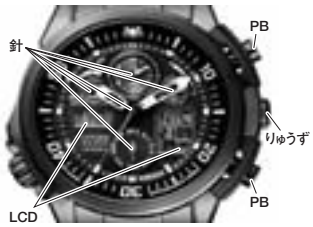


図1　時計の操作・表示手段

## 2．VIPを使う理由

多針で動きが複雑な時計の仕様検討において、動画プログラムで実際に針の動きを表現することによりリアルなイメージに基づいて検討ができるようになります。仕様検討の場で使うので、仕様変更に伴い、その場で動きを変えられる必要があるので、単体で動作するWindowsアプリではなく、VIPでZIPCと連動し状態遷移表を変えると動きが変わるようにするのが良いと考えました。

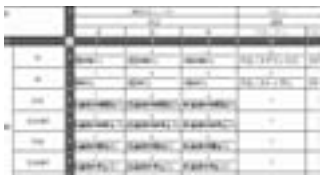

図2　完成イメージ

## 3．簡単なサンプル

状態遷移表を変えることにより動きを変えられる自由度をなるべく大きくし、一方で動画プログラムの作成負荷をなるべく小さくするVIPの使い方及びVC#プログラムの構造について検討しましたので、VC#のコードとZIPCの設定について簡単なサンプルを用いて具体的にご説明します。

サンプルの動画プログラムには、リセットとリバースの2つのボタンがあり、押すとイベントを発生させます。状態遷移表は状態0と状態1の2状態しか存在せず、リセットイベントで状態0に遷移し、リバースイベントで現状と反対の状態に遷移するという単純なモデルを採用しました。単純ですがこのモデルを実現できれば、イベント、状態、アクションを追加することにより実アプリに発展できると考えます。

図3　簡単なサンプル

## 4．基本方針

開発言語はVS2005においては、過去の資産が無いならばどの言語でも同じことができるので、後発で洗練されているVC#を用いることにしました。

プログラムの基本構造としては、動画プログラムが状態遷移表のあやつり人形になるようにしました。すなわち、動画プログラム上でイベントが発生するとVIP経由でZIPCに通知し、状態遷移が発生し、ZIPCからアクションと次の状態をVIP経由で動画プログラムに通知し、動画プログラムが動くようにしました。これにより状態遷移表を修正すれば動画プログラムの動きを変えることができるようになります。

## 5．VC#のコードとZIPCの設定例

以下の例において、状態遷移表名、イベント名、状態名、アクション名、変数名は筆者が適当につけたものです。

### ５．１．動画(イベント)→ ZIPCのVC#スケルトン

名称イベントを使うとZIPCへの設定が不要で簡単です。

VC#のスケルトンは以下のようになります。

引数にはイベントを通知する状態遷移表の名前と通知するイベント名を指定します。

```
private void
button1_Click(object sender, EventArgs e)
{
  axZVipComm1.SendNameEvent(
  “viptest3”,“リセット”);
}
```

図４　動画→ZIPCのVC#スケルトン

### ５．２．ZIPC(アクションと次状態)→ 動画のVC#スケルトン

VC#のスケルトンは以下のようになります。

通知された情報がアクションか状態かをまず判断し、次にその値に従ってしかるべき処理に分岐するようにしました。

```
private void
axZVipComm1_GetVipEvent(
  object sender,
  AxZVIPCOMMLib
    ._DZVipCommEvents_GetVipEventEvent e)
{
  if (e.strEvtName == “actionNoVC”)
  {
    action = Convert.ToByte(e.varEvtData);
    // ここでactionの値によって
    // 該当する処理に分岐し即実行
  }
  else if (e.strEvtName == “stateNoVC”)
  {
    state = Convert.ToByte(e.varEvtData);
  }
}
```

図５　ZIPC→動画のVC#スケルトン

### ５．３．ZIPC(アクションと次状態)→ 動画のZIPCの設定

ZIPCから動画プログラムへの通知については、ZIPC上で様々な設定が必要となります。

システム共通→設計書→メモリ→メモリ設計書作成と登録→IO設計書と辿っていき、ZIPCがアクション番号と状態番号を格納する変数を定義します。(図6)

```
char actionNo;
char stateNo;
```

図６　ZIPCでの変数定義

タスク→viptest3→関数→FUNC設計書作成と登録→関数抽出追加生成と辿っていき、FUNC設計書を修正して、

void 処理0{} に「actionNo=0;」を記入します。

void 処理1{} にも同様に「actionNo=1;」を記入します。

プロジェクト→プロジェクト設定→ジェネレータ設定→Cコード生成設定→STM設定→viptest3→viptest3と辿っていき、状態番号取得生成にチェックを入れて状態番号を取得する関数名をcurstateのように指定します。

書式→STM設定→メイン終了アクティビティと辿っていき、状態遷移が終わった時点で実行するコードを

```
stateNo = curstate();
```

のように指定します。

プロジェクト→VIP設定→環境設定→PIO設定と辿っていき、ポート名とアイテム名の対応関係を登録します。

stateNo　と　stateNoVC
actionNo　と　actionNoVC

連動させる動画プログラム（接続外観図）も指定します。

以上の設定が済んだら、シミュレーションモードでCコード生成、シミュレーションコンパイル　をすれば動画プログラムとZIPCが連動するようになります。

## 6．この構造でできたこと

簡単なサンプルを発展させ、時計の実仕様の針の動きをほぼ再現することができました。

またどのイベントによって動くか変更できました。例えば2つPBがある場合、どちらのPBを押したら電波受信を開始するかを変更するような場合です。さらにどのモードの下に機能を入れるかも変更できました。例えば滅多に使わない機能として基準位置修正機能というものがあります。外部からの衝撃力により針がずれてしまった時に正しい位置に戻す機能ですが、通常使う機能ではありません。これをPBを押すたびに、時刻→アラーム→タイマー→クロノグラフ→時刻と通常使うモード間を遷移する中に入れるか、時刻修正モードに入った際にPBを押した時だけ行くようにするかを変更するような場合です。

このように仕様検討の場で仕様変更に追従して動きを変えることができ、ある程度有効性が確認できました。

## 7．この構造の不都合な点

ただし不都合な点もありました。1つのアクションセル内に複数のアクションを記述した場合、この構造ではほぼ同時にアクションが実行されてしまいます。例えば受信モニター機能というものがあります。前回電波を受信した時、受信に成功したのか失敗したのかと、その時の受信局（多局受信が可能な場合。日本と米国等）を別の針で表示する機能ですが、これを受信成否表示処理と受信局表示処理の2つに分けてアクションセルに記述すると2つの針が同時に回転してしまいます。時計では通常重負荷をさけるため同時に複数の針（モーター）は動かしません。

それを避けるために1セル内にアクションを1つに制限（「受信モニター表示処理」1つにまとめる）しましたが、動画プログラムが様々な針の動作シナリオを内蔵しなければならなくなり、プログラム作成負荷が増大してしまいました。

## 8．対策案

そこで以下のように、アクション番号を通知されても直ぐには実行せずキューにためておき、全てのアクション番号の通知が終わり状態番号の通知がされたタイミングでキューにたまっているアクションを順次実行し、その後で状態遷移し、アクションキューを初期化することを考えました。

```
private void
axZVipComm1_GetVipEvent(
  object sender,
  AxZVIPCOMMLib
    ._DZVipCommEvents_GetVipEventEvent e)
{
  if (e.strEvtName == “actionNoVC”)
  {
    action = Convert.ToByte(e.varEvtData);
    acque[qptr++] = action;
     // アクションをキューイングする
  }
  else if (e.strEvtName == “stateNoVC”)
  {
    for (i = 0; i < qptr; i++)
    {
      doaction(acque[i]);
    // キューに溜まっているアクションを
    // 順次実行
    }
    state = Convert.ToByte(e.varEvtData);
    qptr = 0;
  }
}
```

図7　対策案のVC#スケルトン

## 9．対策案の効果

状態遷移表を書き換えることにより動画の動きをより細かく変えられるようになり、仕様検討の場での適用範囲が広がること、動画プログラムが複雑な動作シナリオを持たなくてよくなり、プログラム作成の負荷が下がることが期待できます。

## 10. 今後の課題

今後、対策案に従って時計の動画プログラムを書き直し、効果を確認する予定です。また作成負荷とのバランスという観点から、動画シミュレーションの仕様検討段階における有効性を評価していきたいと考えております。

## ご存知ですか？ EHSTM/ZIPC 豆知識

ＳＴＭの状態（S型）階層化とはなんですか？

状態階層化は上位のSTMで状態を概要としてとらえて、下位のSTMでは状態を詳細にとらえる手法です。状態に関する概要から詳細へのつながりを、解りやすくまとまりのあるSTMで築き上げ、複雑で巨大なシステムを構成できます。
状態階層化STMでの下位STMの呼び出しは状態から行います。しかし、事象階層化のような呼び出しではなく、状態フレームをSTMで切り分けたイメージとなります。状態番号を持っていないフレームに対しては、下位STMを呼び出すことは出来ません。
また、ステートドリブン型で事象階層化STMを呼び出す場合は、呼び出されたSTMがイベントドリブン型として駆動します。これは、事象階層化STMがアクションに記述され、関数と同じような動作をするためです。

図A　状態階層化前の状態遷移表

図B　状態階層化した状態遷移表