kVerifier（プログラム検証ライブラリ）

kVerifier(Library to verify program codes)

小林憲次
Kenji Kobayashi

栃木県黒磯市阿波町 117-835　E-mail: kenji@nasuinfo.or.jp

ABSTRACT.　kVerifier is a C++/STL library to verify and debug C/C++ program codes generally. It contain functions of CppUnit which is used in XP(eXtreme Programming).　kVerifier separates test data from program source codes.　The separating makes ease to describe complete test data which measure 100% coverage of the source codes.　kVerifier controls time advances.　It can verify time dependent embedded software behavior.s　It can verify circuit codes modeled by SystemC.　It can describe executable specifications too.

１．はじめに

kVerifier は C/C++ プログラムのテスト・検証・シミュレーション汎用的に行う C++ ライブラリです。XP(eXtreme Programming) の CppUnit を包含します。CppUnit とは異なり、テスト・データをプログラム・コードから独立させます。これによりテストの記述を容易にします。100% カバレッジのテストを実用的に作成できます

kVerifier は時間進行を管理・制御します。時間に依存するプログラム動作のテスト・検証・シミュレーションも可能です。組み込みプログラムを対象とできます。回路記述も対象とします。SytemC を使えば C 言語による回路記述ができるからです。kVerifier のテストは実行可能な仕様記述の道具として使うことさえ可能です。

２．XP のテストと kVerifier のテスト・ベクタ

XP のテストはプログラムの修正・変更に伴う動作確認をコンピュータに代行させます。CppUnit はテスト・プログラム・コードをソース・プログラムにコードに追加します。そこで入力条件を設定します。プログラムの動作結果を assert(.) 構文で確認します。

XP への批判もあると思います。組み込みプログラムなどの時間に従って動作するでは、XP を使えせん。でも XP のテストの考え方は多方面で利用できます。検討に値します。テストを利用することで

- デバッグを再利用可能なソフト資産として残す
- プログラム修正に伴う単純デバッグをコンピュータに代行させる

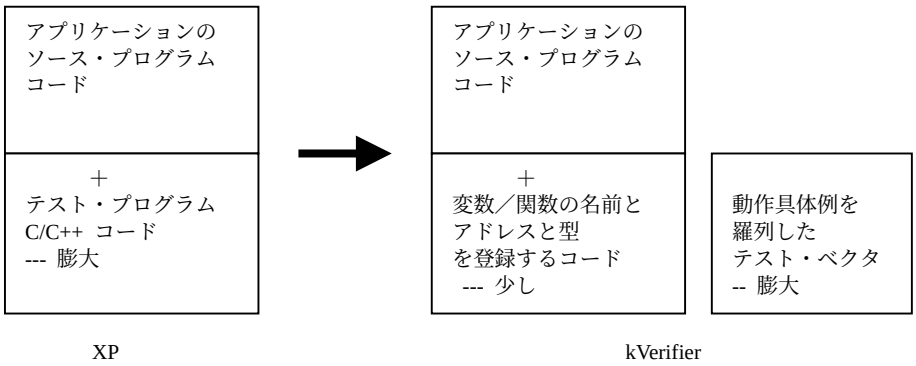

図１ テスト・データをテスト・ベクタに分離します

ことが可能になるからです。XP ではテストを利用することで、プログラム開発期間・規模の増加に伴って、その修正工数が指数関数的に増加することを防ぎます。テストを利用してリファクタリングを行います。プログラム・コードを整理して、プログラム自体を詳細ドキュメントにします。この XP の機能はプログラム一般に有効です。そして、この機能は Verifier にも備わっています。KVerifier は XP を包含します。XP より多くの機能を備えます。

XP では、テストがソース・コードにハード・コーディングされています。テストがプログラム・コードに依存しています。kVerifier はテストを変数／関数の登録とテスト・ベクタに分離します。テストをプログラム・コードに依存する「変数・関数の名前とアドレス登録」プログラムと、プログラム・コードから独立したシーケンス・データの集まりである「テスト・ベクタ」に分けます

３．kVerifier が働く仕組み

kVerifier は登録された変数／関数の名前を使ってテスト・ベクタとアプリケーション・プログラムを協調動作させます。テスト・ベクタより入力変数のシーケンス・データを読んでプログラムを動作させます。動作結果をテスト・ベクタの出力変数の予定値と比較して確認します。

- 入力変数値を、その変数アドレスに設定して、プログラム・コードを動作させます
- アドレスを登録してある関数をテスト･ベクタに指定した引数で呼び出します
- プログラムが入力に応答した出力変数値の結果を予定値と比較・確認させます

この動作原理を表すブロック図を図２に示します。

プログラムの開始時に、変数のアドレスと名前と型、関数のアドレスと名前と関数型を kVerifier に登録しておきます。一方で、テスト・ベクタ・ファイルには、入力変数のシーケンス・データを変数の名前とデータ値を使って記述しておきます。関数の呼び出しシーケンスを関数名と引数値を使って記述しておきます。それらを受けてプログラムが変化させる出力変数のシーケンス・データを記述しておきます。kVerifier　はテスト・ベクタの名前とデータ値を、登録してあった変数／関数と対応させます。入力変数を設定します。関数を呼び出します。出力を確認します

プログラムの開始時に、変数のアドレスと名前と型、関数のアドレスと名前と関数型を kVerifier に登録しておきます。一方で、テスト・ベクタ・ファイルには、入力変数のシーケンス・データを変数の名前とデータ値を使って記述しておきます。関数の呼び出しシーケンスを関数名と引数値を使って記述しておきます。それらを受けてプログラムが変化させる出力変数のシーケンス・データを記述しておきます。kVerifier　はテスト・ベクタの名前とデータ値を、登録してあった変数／関数と対応させます。入力変数を設定します。関数を呼び出します。出力を確認します。

「結果のログ・ファイル」にはテスト・ベクタとアプリケーション・プログラムが動作した結果の全てが記録されます。テストでエラーが検出されたとき、ログ・ファイルをみれば、エラーが発生するまでのシーケンスが記録されています。プログラム開発途中のバグの多くは単純なものです。多くのバグはログファイルを見るだけで原因を推定できます。対策できます。ログファイルはデバッグ時に有効です。特定のデータのみを抜き出してタイム・チャート表示をするときにも使います。

モニタ変数として変数を登録しておくと kVerifier はその変数の変化を監視しつづけます。変化が発生すると、そのタイミングとデータをログ・ファイルに記録します。記録されたモニタ変数のデータはデバッグするときに有効な情報となります。

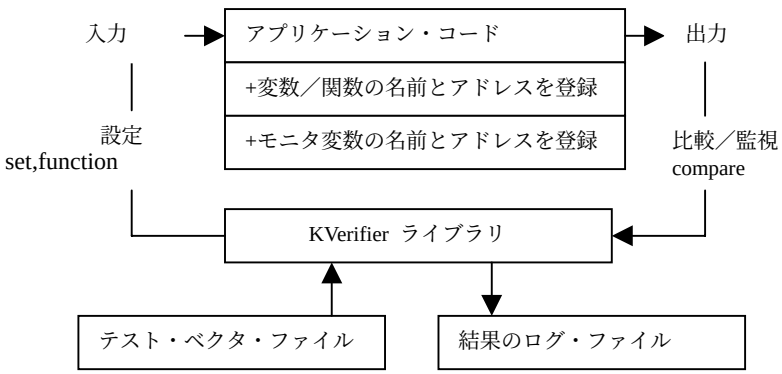

図２　kVerifier の動作原理

```
  1  class ClRegularString : public string {
  2                  ・
        クラス・アプリケーション・プログラム・コード
                   ・
        // クラスのデータ・メンバー、関数メンバーの登録
 15     void Register(ClTestVct* pClAg, const string& crStrAg)
 16     {
 17         tfRgstFnctnStr(pClAg, this, &ClRegularString::CheckMatchedString,
 18                 "checkMatchedString");
 19         tfRgstVrfyUsr(pClAg, *(string*)this, crStrAg+".Matched");              //-0
 20         tfRgstVrfyUsr(pClAg, m_strPostMatched, crStrAg+".PostMatched"); //1
 21         tfRgstVrfyUsr(pClAg, m_strPreMatched, crStrAg+".PreMatched");     //-2
 22         tfRgstVrfyUsr(pClAg, m_strTraversed, crStrAg+".Traversed");          //-3
 23         tfRgstFnctnStr(pClAg, this, &ClRegularString::SetRegularString
 24                 , crStrAg+".SetRegularString");      //-3
 25     }
 26 };
                   ・
        アプリケーション・プログラム・コード
                   ・
120  typedef ClRegularString krgstr;
121  static krgstr krgStrStt("");
122  bool blDbgPrintStt=false;

224  void ClTestVctD::doAtInitial( kc string& crStrAg)
225  {
226                ・
227                ・
228         tfRgstVerified(this, blDbgPrintStt, "dbgPrintFlag", ios::hex);
229         tfRgstFnctnStr(this, setRglExprssnString, "setRglExprssnString");
230         tfRgstFnctnUsr(this, setSentenseString, "setSentenseString");
231         tfRgstFnctnUsr(this, checkRglElementString,"checkRglElementString");
232
233         tfRgstVrfyUsr(this, strSentenseStt, "strSentenseStt");
234         // クラス・メンバーの名前とアドレスの登録
235         krgStrStt.Register(this, "krgStrStt");
236     }
237  }              ・
        ・
```

図３　変数／関数の名前とアドレスと型の登録

4　変数／関数の登録

ここでは、kVerifier の変数／関数登録を実際のコード例を使って説明していきます

「図３ --- 変数／関数の名前とアドレスと型の登録」に正規表現文字列クラス ClRegularString プログラムでのテスト変数／関数の登録コード例を示します。

15 -- 25 行、224 -- 236 行が変数、関数の登録コードです。クラスの最後、プログラム・コードの最後に登録コードを追加記述します。doAtInitial(.) はすでに仮想的に呼び出されており、アプリケーション・コード側から doAtInitial(.) を呼び出す必要はありません。アプリケーション・コードへの影響を最小にするように配慮しています。

228 行目の の tfRgstVerified(.) 関数は "dbgPrintFlag" 文字列引数によって、変数名を登録しています。bool 型変数 blDbgPrintStt を指定することで、そのアドレスと変数の型を kVerifier に登録しています。blDbgPrintStt が参照引数とすることを利用して変数のアドレスを登録しています。

kVerifier がテスト・ベクタとプログラムの変数の間でデータの設定や比較を行うためには変数の型情報が必要になります。このために tfRgstVerified(.) はテンペレート関数として実装してあります。テンペレート関数の性質により、登録変数引数の位置に指定した変数の型に応じた kVerifier への登録コードを、tfRgstVerified(.) は C++ コンパイラに展開させます。tfRgstVerified(.) は bool、int,char,float、complex<.> などの C++/STL が用意している基本型変数を登録するテンペレート関数です。

そのほか

tfRgstVrfyUsr(.) ---- ユーザー定義型の変数を登録します

tfRgstFunction(.) ---- void, または基本型引数を持つ関数を登録します

tfRgstFnctnStr(.) ---- std::string 引数を持つ関数を登録します

のテンペレート関数を用意しています。

235 行目はクラス・インスタンスを指定して、クラス・メンバーの登録関数を呼び出しています。15 -- 25 行がクラス・メンバーの登録コードです。クラス・メンバーの登録関数 Register(.) メンバー関数を追加することで、クラス内のプライベート・データ／関数も登録することが可能になります。

121 行目の「static krgstr　krgStrStt(“”) 」は、空白文字列 “” で正規表現文字列クラスのインスタンス krgStrStt をスタティックに生成しています。235 行目で、krgStrStt インスタンスのメンバー要素を登録する kVerifer に登録する関数を呼び出しています。krgStrStt インスタンスの名前文字列を “krtStrStt” と指定する引数も与えています。

20 行目の引数に指定されている m_strPostMatched は正規表現クラスのプライベートな std:string 型のメンバー変数です。Perl の $' に相当する文字列です。$& マッチ文字列の後ろ側の文字列です。このような ClRegularString 内部変数に crStrAg+".PostMatched" の名前を付けて、テスト・ベクタから参照可能にします。,

5　テスト・ベクタ

「図４ --- 正規表現ライブラリのテスト・ベクタ」に正規表現ライブラリの動作をテストするテスト・ベクタの実際のコードを示します。テスト・ベクタがプログラム動作の具体例の羅列であること、その記述は職業プログラマーでなくても可能なことが判ると思います。

図４で、# 以降の記述はコメントです。2 行目のコメントは "@abc:" の文字列を正規表現文字列 "a(.*):" に対するマッチを行わせたときの (.*) に対応する文字列を調べることを報せています。Perl での $1 に対応する文字列を調べています。なお ClRegularString では正規表現マッチを行う処理を "/" 演算子に割り当てているので "abc:"/"@(.*):.*" のような記述をコメント記述にしています。

3 行目の「krgStrStt.SetRegularString __function "@(.*):.*"」は krgStrStt.SetRegularString の名前の付いた関数を、正規表現文字列 "@a(.*):.*" の引数文字列で呼び出すことを意味します。

4 行目の「setSentenseString __function "@abc:"」はテストされる文字列を "abc:" 文字列引数にして、"setSentenseString" と名前のついた正規表現マッチを行わせる関数を呼び出します。

6 行目の「krgStrStt.Matched __compare "@abc:"」は "@abc:" が正規表現マッチしたことを確認しています。7 行目の「checkMatchedString __function "1 abc"」は Perl での $1 に対応する括弧で囲まれた部分の文字列を "abc" であることを確認する関数を呼び出しています。

```
1   ##-------------------------------------------------17
2   ## "@abc:" / "@(.*):.*" test $1          01.06.18
3   +0 krgStrStt.SetRegularString __function "@(.*):.*"
4   +0 setSentenseString __function "@abc:"     #test string
5   +0 krgStrStt.Traversed __compare "@abc:"
6   +0 krgStrStt.Matched __compare "@abc:"
7   +0 checkMatchedString __function "1 abc"    # $1
8
9   ##-------------------------------------------------16
10  ## "<ol>"/ "<ol>(<li>)?" == ""
11  +1 krgStrStt.SetRegularString __function "<ol>(<li>)?"
12  +0 setSentenseString __function "<ol>"         #test string
13  +0 checkMatchedString __function "1 "          #check regular matched result
14  +0 checkMatchedString __function "-3 <ol>"#check regular matched result
15  +0 krgStrStt.Matched __compare "<ol>"       # same at upper -3
16
17  ##-------------------------------------------------15
18  ## front caret "abcdefg"/ "^cd" == ""
19  +1 blDbgPrintSttu                    __set 1
20  +0 krgStrStt.SetRegularString __function "^cd"
21  +0 setSentenseString __function "abcdefg"    #test string
22  +0 krgStrStt.Matched __compare ""              #check regular matched result
                .
                .
```

図４　正規表現ライブラリのテスト・ベクタ

1 -- 7 行目のテスト・ベクタは "@abc:" の文字列を "@(.*):.*" の正規表現文字列にマッチさせたときの動作の具体例をを記述したものです。このような動作の具体例を羅列することは C や C++ を知らなくても記述できます。プログラムを作るまえに仕様記述の段階で作成することも可能です。

1 -- 7 行目の正規表現動作例は "@(.)*:.*" と .* の任意文字列のマッチが二回現れても途中の ':' の文字によるマッチングが行われて $1 == "abc" となることを確認するものです。本当は作り上げた正規表現　図４で、# 以降の記述はコメントです。2 行目のコメントは "@abc:" の文字列を正規表現文字列 "a(.*):" に対するマッチを行わせたときの (.*) に対応する文字列を調べることを報せています。Perl での $1 に対応する文字列を調べています。なお ClRegularString では正規表現マッチを行う処理を "/" 演算子に割り当てているので "abc:"/"@(.*):.*" のような記述をコメント記述にしています。

3 行目の「krgStrStt.SetRegularString __function "@(.*):.*"」は krgStrStt.SetRegularString の名前の付いた関数を、正規表現文字列 "@a(.*):.*" の引数文字列で呼び出すことを意味します。

4 行目の「setSentenseString __function "@abc:"」はテストされる文字列を "abc:" 文字列引数にして、"setSentenseString" と名前のついた正規表現マッチを行わせる関数を呼び出します。

6 行目の「krgStrStt.Matched __compare "@abc:"」は "@abc:" が正規表現マッチしたことを確認しています。7 行目の「checkMatchedString __function "1 abc"」は Perl での $1 に対応する括弧で囲まれた部分の文字列を "abc" であることを確認する関数を呼び出しています。

1 -- 7 行目のテスト・ベクタは "@abc:" の文字列を "@(.*):.*" の正規表現文字列にマッチさせたときの動作の具体例をを記述したものです。このような動作の具体例を羅列することは C や C++ を知らなくても記述できます。プログラムを作るまえに仕様記述の段階で作成することも可能です。

1 -- 7 行目の正規表現動作例は "@(.)*:.*" と .* の任意文字列のマッチが二回現れても途中の ':' の文字によるマッチングが行われて $1 == "abc" となることを確認するものです。実際には作り上げた正規表現ライブラリに隠れていたバグを修正したときに追加したテスト・ベクタです。このような、正規表現プログラムで見落としてしまったバグが 17 個あったので 1 行目の 17 の番号となっています。

近いうちに、この正規表現ライブラリのソース自体も公開するつもりです。今回はそのソースを 100% coverage するテスト・ベクタを「付録 1 正規表現ライブラリを 100% coverage するテスト・ベクタ」に示します。このテスト・ベクタは正規表現を記述するプログラム一般に適用できます。DFA アルゴリズムで発生しやすいバグ・パターンを含んでいます。付録に示したテスト・ベクタは正規表現ライブラリのソース・コードと同等以上の価値をもったソフト資産であると言えます。

6　時間を含んだテスト・ベクタ

kVerifer のテスト・ベクタでは時間経過も記述できます。組み込みなどの分野のテストも記述できます。

以下のような機能動作のテスト例を考えてみます。

- 入力ポートＡが 0x03 の時に割り込みＡが発生した時に
- 一秒周期で二秒間ＬＥＤを点滅させた後に
- 処理Ａを関数呼び出しで開始させる

このテストの内容は図５のような入力の設定(set)と出力の確認(compare) のテスト・ベクタで表現できます

```
   0mS   入力ポートＡ      __set          0x03       # port の値を 3 に設定する
 +100mS   割り込みＡ        __set                      # 割り込みＡを発生させる

 +100mS   LED               __compare    1           # LED が On である
 +500mS   LED               __compare    0           # LED が Off である

 +500mS   LED               __compare    1           # LED が On である
 +500mS   LED               __compare    0           # LED が Off である

 +500mS   処理Ａ            __functionn              # 処理Ａ関数を呼び出す

 +100mS   LED               __compare    1           # LED が On である
# 以下に処理Ａの動作例を記述する
          ・
          ・
```

図５　kVerifier のテスト、（テスト・ベクタ）

```
void ClTestDrived::runTest(void* pVdAg)        // task fuction 実装にする
{
      delay(100);
      inPortA = 0x03
      delay(100);
      interruptA();
      delay(100);
      assert(Port0.bit3 == true);
      delay(500);
      assert(Port0.bit3 == false);
            ・
      startProcessA();
            ・
}
```

図６　CppUnit のテスト

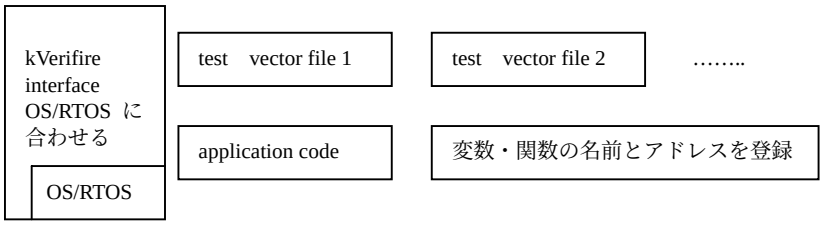

図７　　CppUnit のテスト・コードにはアプリケーシ

このテストを XP の CppUnit で記述すると図６のようになります

CppUnit のテスト・コードにはアプリケーション・コード固有の情報がハード・コーディングされています。LED bit 変数 Port0.bit3 を直接に書いています。RTOS の delay(.) を直接に呼び出しています。コンピュータの機種や RTOS が変わったとき、この CppUnit コードは変更が必要となります。

kVerifier では時間進行の管理を行うインターフェースを設けます。先の正規表現ライブラリのように時間に依存しないときはライブラリ・インターフェースを使います。このときは時間の代わりにループ回数を使います

図７の kVerifire interface 部分がシミュレーション実行を制御・管理します。図１のテスト・ベクタ各行の行頭に記述した時間ぶんだけ送らせたタイミングでテスト・ベクタ・データの設定・確認・関数呼び出しを行います。このインターフェース部分は、OS/RTOS ごとに合わせ込むことが必要になります。

7　テスト・ベクタと仕様記述と仕様記述クラス

テスト・ベクタは具体例を並べたプログラム仕様書と見なせます。プログラム・コードの代用として、ソフトウェアをシミュレーション駆動することも可能です。kVerifer はアルゴリズムを使った仕様記述も用意します。

```
+10mSe      kuOS::interrupt_10mSecInterval   __function
+10mSe      kuOS::interrupt_10mSecInterval   __function
      ・
      ・   100 個ならべる
      ・
+10mSe      kuOS::interrupt_10mSecInterval   __function
```

図８　test vector による 10mSec interval 割り込み

```
static class Cl10mSecIntervalTask: public ClVrfyThrd{
public:
   Cl10mSecIntervalTask(void){ Start();}
protected:
   virtual void main(void)
   {
        for(;;){
             // extern void kuOS::interrupt_10mSecInterval(void);
             Wait(10*mS);       // 10mSec delay
             kuOS::interrupt_10mSecInterval();
        }
   }
}cl10mSecIntervalTaskStt;
```

図９　アルゴリズム・コードによる 10mSec interval 割り込

例えば、図８のテスト・ベクタ行を 100 個並べることで、10 mSec 周期でのタイマー割り込みを発生する回路動作を一秒間だけシミュレーションできます。

このような単純な動作のときはアルゴリズムを使ったほうが単純になります。kVerifier は仕様記述のためのスレッド・クラス ClVrfyThrd を用意しています。ClVrfyThred は C++/STL を使った concurrent 性を持たせた関数記述を可能にします。ClVrfyThrd を使えば、10mSec 周期の割り込み呼び出しは図９のコードで実現できます。

組み込みプログラムをシミュレーション実行させるためには、回路動作をシミュレーション実行させることが必要になることが多くあります。でも全ての回路を SystemC で回路コードを記述する必要はありません。ClVrfyThrd を使えは、簡単に回路動作を記述できます。

ClVrfyThrd は動作する仕様を記述するためにも導入しました。SpecC を使うことも検討しましたが、クラスや STL を使えないため、kVerifeir 側で仕様記述のためのクラスを用意することにしました。STL の記述力の高さを選択しました。

テスト・ベクタはソースから独立しておりポータビリティがあります。これはアルゴリズムよる仕様記述での動作を確認するテスト・ベクタが実装段階でのテスト・ベクタに使えることを意味します。SystemC での回路ビヘービア記述を RTL 記述に変換したとき、動作が変わっていないことをテスト・ベクタが保証できることを意味します。

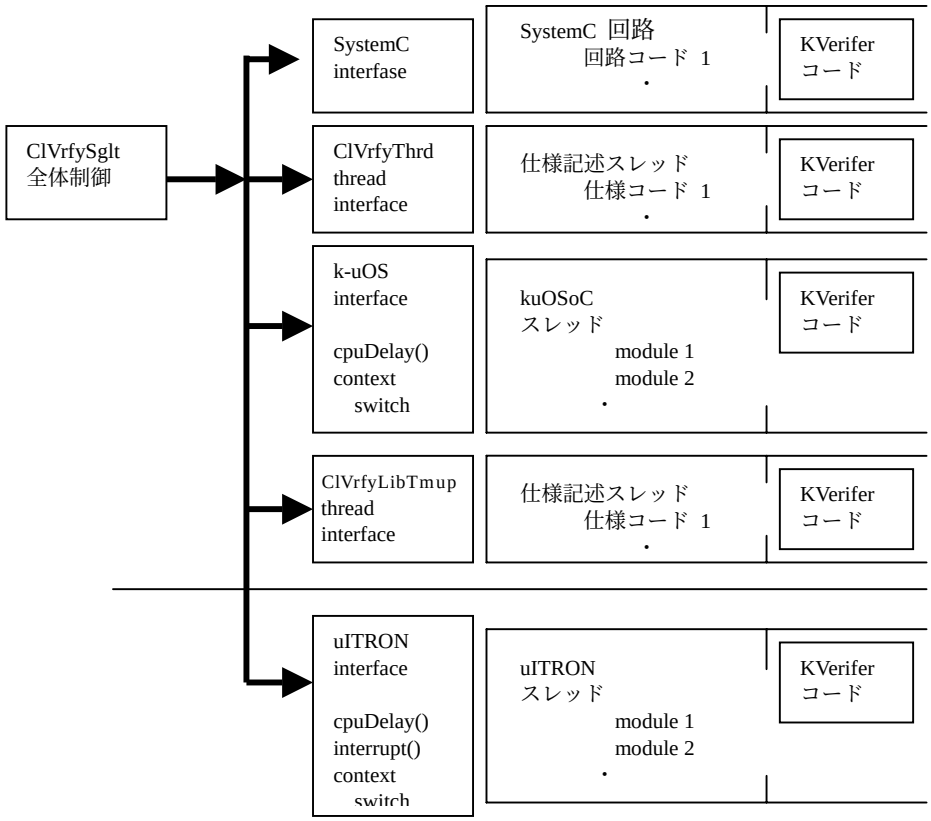

図１０　回路、仕様、タスク全体の連携

8　回路、仕様、タスク記述の協調動作

回路も含めて、組み込みの全体を C/C++ 言語で記述できるようになってきました。kVerifier は回路、仕様、タスクの全体をテスト・ベクタを介在させて、シミュレーション実行・検証します

ワンチップ・マイコンの開発言語は C 言語が普通になりました。RTOS も C 言語で記述してあれば、ワンチップ･マイコンの組み込みプログラムを MS Windows などのクロス環境で動作させられます。現在では回路記述も SystemC などを使って C 言語記述ができます。kVerifier は、これらの C 言語で記述してあるプログラム・コードを連携させてシミュレーション実行・検証を行います。

現在の段階では kVerifier は以下のアプリケーションを連携動作させることを想定しています。

- 回路記述を行う SystemC
- RAM の少ないマイコンでのタスク記述を行う k-uOS
- C++/STL を使った並列記述を可能にする仕様記述スレッド ClVrfyThrd
- 代表的な RTOS -- uITRON

SystemC, RTOS タスクなどのアプリケーション・プログラムの動き方はベース・ソフトにより異なります。ベース・ソフトに合わせた kVerifier interface を用意し、アクティブにする kVerifier interface をダイナミックに切り替えることで、全体を協調動作させます。

k-uOS, uITRON interface に設けられている cpuDelay(delayAg) は CPU 消費時間をシミュレートするためのものです。この関数が呼び出されると、それを呼び出したスレッドの実行は中断されて他の kVerifier interface に CPU の使用を許します。シミュレーション時間が delayAg 時間だけ経過したあとに、delayCpu() の次のアドレスの制御が戻ってきます。

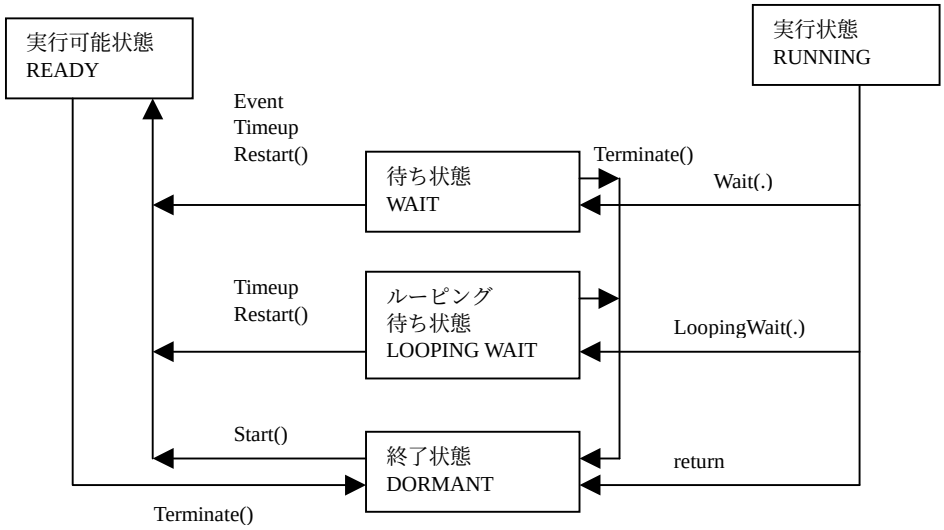

図１１　k-uOS, ClVrfyThrd タスク状態遷移

9　k-uOS と ClVrfyThrd

k-uOS と ClVrfyThrd は両方ともノンプリエンプティブな RTOS です。意図的に似せた機能を実装します。

k-uOS は SoC マイコン、ワンチップ・マイコンなどの極端にラムが少ない環境でもタスク記述を可能にするために設けました。k-uOS はスタックを共用することで、ラムの消費を抑えます。通常はタスクごとに設けるスタック・ラム領域を k-uOS が使うスタック一本で共用します。スタック・ラムの共用により、たとえ 1K byte のラム・サイズであっても実用的なタスク記述を可能にします。

スタックを共用した結果、タスクのプリエンプティブ実行ができません。プリエンプティブ実行を許すと、プリエンプションされたタスクが使っていたスタック・データが壊れてしまうからです。SoC などの小規模なアプリケーションではノンプリエンプティブであることは弊害になりません。逆にタスクの実行タイミングを単純にするメリットのほうが大きいと考えます。ノンプリエンプティブであるため、 k-uOS が用意するタスク制御も単純なものだけに限定できます。

一方で仕様記述スレッド ClVrfyThrd もノンプリエンプティブです。仕様の concurrent 動作を記述するために導入するタスク記述に、プリエンプティブ性は必要ありません。このため ClVrfyThrd と k-uOS は互いに似た構造をもたせることができます。仕様記述から実装記述への変換を容易にするため、意図して似た構造を持たせています。

ClVrfyThrd と k-uOS のタスク状態は図１１に示す共通した遷移を行います

10 k-uOS と ClVrfyThrd

使用する kVerifire interface を全体制御 ClVrfySglt に登録することで、動作させる回路、仕様、RTOS の組み合わせが決まります。各 kVerifier interface は次に実行すべき時刻を保持しています。そして以下の動作を繰り返すことで、全体を連携・協調して動作させます

- 登録された SystemC, ClVrfyThrd, ClKuosIntf のインターフェースの中から、次に実行すべき時刻が最小のものを選択する
- 選択されたインターフェースに登録されたテスト・ベクタ・インスタンスの中から次に実行すべき時刻が最小のテスト・ベクタ・コマンド、またはタスクを実行する
- 以上をくりかえす

より進んだ仕様記述言語や、RTOS,回路言語を使うときには、それぞれに合わせた専用の kVerifier インターフェースを追加します。

11 最後に

kVerifier はテスト・ベクタをプログラム・コードから独立した、ポータビリティのあるものにしました。プログラミングを専門としないユーザーにもテストを書くことを可能にしました。テスト・ベクタ自身を仕様とみなせるようにしました。プログラム・コードの代替に使えるようにしました。C/C++ に限られますが、XP のテストを改善しました。

シミュレーション実行を含んだ kVerifier は、時間進行を含む組み込みプログラムで XP のテストを行うことを可能にしました。仕様記述、回路記述と RTOS タスク記述 f 全体を組み合わせた状態で、連携動作させることを可能にしました。

付録 1

正規表現ライブラリを 100% カバレッジするテスト・ベクタ

```
##-------------------------------------------------17
## "@abc:" / "@(.*):.*" test $1           01.06.18
+0 krgStrStt.SetRegularString __function "@(.*):.*"
+0 setSentenseString __function "@abc:"      #test string
+0 krgStrStt.Traversed __compare "@abc:"
+0 krgStrStt.Matched __compare "@abc:"
+0 checkMatchedString __function "1 abc"    # $1

##-------------------------------------------------16
## "<ol>"/ "<ol>(<li>)?" == ""
+1 krgStrStt.SetRegularString __function "<ol>(<li>)?"
+0 setSentenseString __function "<ol>"        #test string
+0 checkMatchedString __function "1 "          #check regular matched result
+0 checkMatchedString __function "-3 <ol>"#check regular matched result
+0 krgStrStt.Matched __compare "<ol>"          # same at upper -3

##-------------------------------------------------15
## front caret "abcdefg"/ "^cd" == ""
+1 krgStrStt.SetRegularString __function "^cd"
+0 setSentenseString __function "abcdefg" #test string
+0 krgStrStt.Matched __compare ""                #check regular matched result

##------------------------------------------------- 14
## makeCapatalSmallAtConstruct EnIgnoreSmall "abCdE"/ "cd" == "Cd"
+1 makeCapatalSmallAtConstruct __function "cd"
+0 setSentenseString __function "abCdE"      #test string
+0 krgStrStt.Matched __compare "Cd"             #check regular matched result

##------------------------------------------------- 13
## "hxi38b4y"/ "(i|^d)*" == "xa3"
+1 krgStrStt.SetRegularString __function "(i|¥d)*"
+0 setSentenseString __function "hxi38b4y"#test string
+0 krgStrStt.Matched __compare "hxi38b4y" #check regular matched result
#+0 __setBreakTime

##------------------------------------------------- 11
## "xa38b4y"/ "[¥db]+^d" == "38b4"
+1 setRglExprssnString __function "[¥db]+^d"    #test string
+0 setSentenseString __function "xa38b4y"      #test string
+0 checkMatchedString __function "0 38b4" #check regular matched result

##------------------------------------------------- 11
## "ABCD[XY"/ "D¥x5b" == "D["
+1 setRglExprssnString __function "D¥x5b"      #test string
+0 setSentenseString __function "ABCD[XY"      #test string
+0 krgStrStt.Matched __compare "D["                #check regular matched result
```

```
##-------------------------------------------------10
## "abCde"/"[bCD]+" == "bCd IgnoreCapitalSmall
+1 setRglExprssnString __set "[bCD]+"                #regular string
+0 blIgnoreCapitalSmall __function 1
+0 setSentenseString      __set "abCde"      #test string
+0 krgStrStt.Matched __compare "bCd"               #check regular matched result

##-------------------------------------------------9
## "[C-鵈]+"    range from a Capital character to a Small character
+1 setRglExprssnString __set "[C-鵈]+"               #regular string
+0 setSentenseString      __set "ABCD 小林 zy"      #test string
+0 checkMatchedString __function "0 CD 小林 zy" #check regular matched result

##-------------------------------------------------8
## "[C-c]+"    range from a Capital character to a Small character
+1 setRglExprssnString __set "[C-c]+"                #regular string
+0 setSentenseString __function "ABCD[XYabcd"      #test string
+0 krgStrStt.Matched __compare "CD[XYabc"          #check regular matched result

##-------------------------------------------------7
## "ABCD[XY"/ "[C-¥[]+" == "CD[XY"
+1 setRglExprssnString __function "[C-¥[]+"    #test string
+0 setSentenseString __function "ABCD[XY"      #test string
+0 krgStrStt.Matched __compare "CD[XY"         #check regular matched result

## "xa    by" / "a^s+b" == "a    b" ---------- 6
+1 setRglExprssnString __set "a^s+b"              #regular string '?'==0x3f
+0 setSentenseString      __set    "xa    by"          #sentense
+0 krgStrStt.Matched __compare "a    b"           #check regular matched result

## "XABCDY" / "A[ b-cD]+" == "ABCD"  大文字・小文字を無視してマッチングさせる  -- 5
+1 setRglExprssnString __set "A[b-cD]+"               #regular string '?'==0x3f
+0 blIgnoreCapitalSmall __function 1
+0 setSentenseString __set "XABCDY"                     #sentense
+0 krgStrStt.Matched __compare "ABCD"                  #check regular matched result

## "a?2b" / "a[¥x3f]" == "a?" ---------- 4
+1 setRglExprssnString __set "a[¥x3f^d]+"            #regular string '?'==0x3f
+0 setSentenseString __set "a?2b"                      #sentense
+0 krgStrStt.Matched __compare "a?2"                    #check regular matched result

## "a32", "a.2", "a*2" / "a[¥d^.^*]" ---------- a3 , a., a*
+1 setRglExprssnString __set "a[¥d^.^*]"    #test string
+0 setSentenseString __set "a32"      #test string
+0 krgStrStt.Matched __compare "a3"                    #check regular matched result
+0 setSentenseString __set "a.2"      #test string
+0 krgStrStt.Matched __compare "a."                    # check regular matched result
+0 setSentenseString __set "a*2"      #test string
+0 krgStrStt.Matched __compare "a*"                    # check regular matched result

##-------------------------------------------------
## "//C-- a++"/ "[¥w-¥s]+^ == "C++"
```

```
+1 setRglExprssnString __function "[¥w-¥s]+"    #test string
+0 setSentenseString __function "//C-- a++"      #test string
+0 krgStrStt.Matched __compare "C-- a"               # check regular matched result

##-------------------------------------------------
## "//C++--"/ "[¥w¥+]+^ == "C++"
+1 setRglExprssnString __function "[¥w¥+]+"    #test string
+0 setSentenseString __function "//C++--"      #test string
+0 krgStrStt.Matched __compare "C++" #

##-------------------------------------------------
## "abcdefg" / "cde" test -1, -2
+1 setRglExprssnString __function "cde"
+0 setSentenseString __function "abcdefg"      #test string
+0 krgStrStt.Traversed __compare "abcdefg"
+0 krgStrStt.Matched __compare "cde"
+0 krgStrStt.PreMatched __compare "ab"
+0 krgStrStt.PostMatched __compare "fg"

##-------------------------------------------------
## "a¥.b"/ "a^.b" == a¥.b
+1 setRglExprssnString __function "a^.b"    #test string
+0 setSentenseString __function "a.b"      #test string
+0 krgStrStt.Matched __compare "a.b"                    #check regular matched result

##-------------------------------------------------
## "b" / "(.*)b"    test  行頭、行末
+0 setRglExprssnString __function "^(.*)b"    #test string
+1 setSentenseString __function "b"      #test string
+1 checkMatchedString __function "0 b" #check regular matched result
+0 checkMatchedString __function "1 " #check regular matched result

#---------- Test related to SJIS character Beginning ----------
## "H 多 a" / "H(.+)a"
+0 setRglExprssnString __function "H(.+)a"       #test string
+0 setSentenseString __function "H 多 a"       #test string
+1 checkMatchedString __function "0 H 多 a"    # check regular matched result
+0 checkMatchedString __function "1  多"      # check regular matched result

#---------- Test related to SJIS character Edn ----------


##-------------------------------------------------
## "    abc<br>" / "^¥s*(.+)<br>$"    test  行頭、行末
+0 setRglExprssnString __function "^¥s*(.+)<b>$"    #test string
+1 setSentenseString __function "xaAAB<b>"      #test string
+1 checkMatchedString __function "0 xaAAB<b>" #check regular matched result
+0 checkMatchedString __function "1 xaAAB"          # check regular matched result

## "H::3a" / "¥w$"
+0 setRglExprssnString __function "¥w+$" #test string
+1 setSentenseString __function "H::3a"    #test string
```

```
+1 checkMatchedString __function "0 3a"    # check regular matched result


##-------------------------------------------------
## "¥s*[^<](.*)" FlowNfa(.) で 最初の ¥s* を飛ばして到達可能性をチェックする
+0 setRglExprssnString __function "¥s*[^<](.*)" #test string
+1 setSentenseString __function "Axa"      #test string
+1 checkMatchedString __function "0 Axa" # check regular matched result
+1 checkMatchedString __function "1 xa" # check regular matched result

##-------------------------------------------------
## ¥s+[^<](.*)" #
+0 setRglExprssnString __function "¥s+[^<](.*)" #test string
+1 setSentenseString __function "    <xa"      #test string
#  まだスペースを含むマッチした文字列をチェックできない
+1 checkMatchedString __function "0 ^b^b<xa" # check regular matched result
+0 checkMatchedString __function "1 <xa" # check regular matched result

##-------------------------------------------------
## ¥s*([^<].*)(<br>)?"    #test string
+0 setRglExprssnString __function "^¥s*([^<].*)(<br>)"    #test string
+1 setSentenseString __function "      xaAAB<br>"      #test string
+1 checkMatchedString __function "1 xaAAB"                # check regular matched result

##-------------------------------------------------
+0 setRglExprssnString __function "¥s+(.*)c?"    #test string %1 が NG
+1 setSentenseString __function "      xaAABc"      #test string
+1 checkMatchedString __function "1 xaAABc"      # check regular matched result

##-------------------------------------------------
## a(.+)b*" #test b* は ε 遷移を含む
+0 setRglExprssnString __function "a(.+)b*" #test string
+1 setSentenseString __function "      aABb"      #test string
+1 checkMatchedString __function "1 ABb"       # check regular matched result


##-------------------------------------------------
## <.*>
+0 setRglExprssnString __function "<.*>"    #test string
+1 checkRglElementString __function "0 <.*>            n"         # ClParenticRgl
+0 checkRglElementString __function "1 <          n"               # ClDotPrm
+0 checkRglElementString __function "2 .          *"               # Cl1CharPrm
+0 checkRglElementString __function "3 >          n"               # Cl1CharPrm

+1 setSentenseString __function "<<>x>fgXYZ"      #test string
+1 checkMatchedString __function "0 <<>x>" # check regular matched result

##-------------------------------------------------
## "[a] and "aaaa"    #test string
+0 setRglExprssnString __function "[a]"    #test string
+1 setSentenseString __function "aaaa"      #test string
+1 checkMatchedString __function "0 a"                # check regular matched result
```

```
##-------------------------------------------------
## ¥s*([^<].*)(<br>)?"    #test string
+0 setRglExprssnString __function "[^h]"    #test string
+1 setSentenseString __function "xaA"      #test string
+1 checkMatchedString __function "0 x"                # check regular matched result

##-------------------------------------------------
## [abA]
+0 setRglExprssnString __function "^[aA]+"    #test string
+1 setSentenseString __function "aAAB"      #test string
+1 checkMatchedString __function "0 aAA"                # check regular matched result


##-------------------------------------------------
+0 setRglExprssnString __function "¥s+(.*)c"    #test string %1 が NG
+1 setSentenseString __function "      xaAABc"      #test string
+1 checkMatchedString __function "1 xaAAB"                # check regular matched result

##-------------------------------------------------
+0 setRglExprssnString __function "¥sx(.*)c"    #test string %1 が NG
#  下は同じ"      xaAABc" であり、+0 にするか、コメント・アウトしないと、エラーになる
+0 setSentenseString __function "      xaAABc"      #test string
+1 checkMatchedString __function "0 ^bxaAABc"# check regular matched blank
+1 checkMatchedString __function "1 aAAB"       # check regular matched blank


##-------------------------------------------------
## 最長一致では、もっと長い文字列が、後でたどり着く例である
## <.*>
+0 setRglExprssnString __function "<.*>"    #test string
+1 checkRglElementString __function "0 <.*>           n"         # ClParenticRgl
+0 checkRglElementString __function "1 <          n"               # ClDotPrm
+0 checkRglElementString __function "2 .          *"               # Cl1CharPrm
+0 checkRglElementString __function "3 >          n"               # Cl1CharPrm

+1 setSentenseString __function "<<>x>fgXYZ"      #test string
+1 checkMatchedString __function "0 <<>x>" # check regular matched result

+1 setSentenseString __function "ABC<abcd>fgXYZ"      #test string
+1 checkMatchedString __function "0 <abcd>"       # check regular matched result

##-------------------------------------------------
## [A-F]
+0 setRglExprssnString __function "[A-F]+"    #test string
+1 checkRglElementString __function "0 [A-F]+ n"         # ClParenticRgl
+0 checkRglElementString __function "1 A-F        +" # ClRangePrm
+1 setSentenseString __function "xaAABFxabcb"      #test string
#+0 __setBreakTime
+1 checkMatchedString __function "0 AABF"                # check regular matched result
```

```
##-------------------------------------------------
## [a-d]* | [b-e]* regular string
+0 setRglExprssnString __function "a[A-F]*x"        #test string
+1 checkRglElementString __function "0 a[A-F]*x n" # ClParenticRgl
+1 checkRglElementString __function "1 a             n"# ClStringPrm no repeator
+0 checkRglElementString __function "2      A-F      *"# ClRangePrm no repeator
+0 checkRglElementString __function "3 x             n"# Cl1CharPrm no repeator
#+0 debugFlag __set 1
#+0 __setBreakTime
+1 setSentenseString __function "xaAABFxabcb"      #test string
+1 checkMatchedString __function "0 aAABFx"      # check regular matched result
+1 setSentenseString __function "aAABFx"        #test string
+1 checkMatchedString __function "0 aAABFx"      # check regular matched result
#+2 __end


## regular expression 動作をチェックする
+1 setRglExprssnString __function "[a-d]*"    #test string
#+0 __setBreakTime
+1 checkRglElementString __function "0 [a-d]* n" # ClParenticRgl
+0 checkRglElementString __function "1 a-d *"      # ClRangePrm no repeator

+1 setRglExprssnString __function "abc"
#+1 setRglExprssnString __function "ac"
#+1 setSentenseString __function "abc"    #test string
+1 setSentenseString __function "ab"        #test string
#+1 __end

+2 setRglExprssnString __function "abc+"
+1 checkRglElementString __function "0 abc+ n"      #ClParenctic
+0 checkRglElementString __function "1 a n" #ClCharPrm
+0 checkRglElementString __function "2 b n" #ClCharPrm
+0 checkRglElementString __function "3 c +" #ClCharPrm
+1 setSentenseString __function "abccc" #test string
#+0 __setBreakTime
+1 checkMatchedString __function "0 abccc"      # check regular matched result

+1 setSentenseString __function "xxabccabc" #test string
+1 checkMatchedString __function "0 abcc"      # check regular matched result


+2 setRglExprssnString __function "ab+c"
+1 checkRglElementString __function "0 ab+c n"      #ClParenctic
+0 checkRglElementString __function "1 a n" #ClCharPrm
+0 checkRglElementString __function "2 b +" #ClCharPrm
+0 checkRglElementString __function "3 c n" #ClCharPrm
+1 setSentenseString __function "abccc" #test string
#+0 __setBreakTime
+1 checkMatchedString __function "0 abc"      # check regular matched result

+1 setSentenseString __function "xxabbbccabc"      #test string
+1 checkMatchedString __function "0 abbbc"      # check regular matched result
```

```
+1 setSentenseString __function "xxaccac"      #test string
+1 checkMatchedString __function "0 "      # check regular matched result


##---(a^d*)    (^d+)---------------------------------
+0 setRglExprssnString __function "(a^d*)xb(^d+)y"       #test string
+1 checkRglElementString __function "0 (a^d*)xb(^d+)y n" # Base ClParenticRgl
+0 checkRglElementString __function "1 a^d*             n"# ClParenticRgl
+0 checkRglElementString __function "2    a n"             # Cl1CharPrm
+0 checkRglElementString __function "3 ^d *"             # ClNumberPrmtv
+0 checkRglElementString __function "4    x n"             # Cl1CharPrm
+0 checkRglElementString __function "5    b n"             # Cl1CharPrm
+0 checkRglElementString __function "6 ^d+              n"# ClParenticRgl
+0 checkRglElementString __function "7 ^d +"             # ClNumberPrmtv

+1 setSentenseString __function "XXyya324xb54yZZZ"       #test string
+1 checkMatchedString __function "0 a324xb54y"      # check regular matched result
+0 checkMatchedString __function "1 a324"             # check regular matched result
+0 checkMatchedString __function "2 54"               # check regular matched result

##-- (^d)+ ----------------------------------------
+0 setRglExprssnString __function "xb(^d)+y"       #test string
+1 checkRglElementString __function "0 xb(^d)+y n" # Base ClParenticRgl
+0 checkRglElementString __function "1 x            n"# ClParenticRgl
+0 checkRglElementString __function "2 b            n"# Cl1CharPrm
+0 checkRglElementString __function "3 ^d           +"# ClNumberPrmtv
+0 checkRglElementString __function "4    ^d         n"# Cl1CharPrm
+0 checkRglElementString __function "5    y          n"# Cl1CharPrm

+1 setSentenseString __function "XXyya324xb57yZZZ"       #test string
#+1 setSentenseString __function "xb54yZ"       #test string
+1 checkMatchedString __function "0 xb57y"      # check regular matched result
+1 checkMatchedString __function "1 7"             # check $1 matched result

##-------------------------------------------------
## (ab)(cd)e  二回目の  NfaFlow()  が実行されなかったときのシミュレーション
+0 setRglExprssnString __function "(ab)(cd)e"    #test string
+1 checkRglElementString __function "0 (ab)(cd)e n"         # ClParenticRgl
+0 checkRglElementString __function "1 ab       n"              # ClParentic
+0 checkRglElementString __function "2 a        n"              # Cl1CharPrm
+0 checkRglElementString __function "3 b        n"              # Cl1CharPrm
+0 checkRglElementString __function "4 cd       n"              # ClParentic
+0 checkRglElementString __function "5 c        n"              # Cl1CharPrm
+0 checkRglElementString __function "6 d        n"              # Cl1CharPrm
+0 checkRglElementString __function "7 e        n"              # Cl1CharPrm

+1 setSentenseString __function "ABCDabcdefgXYZ"      #test string
+1 checkMatchedString __function "0 abcde"          # check regular matched result
+0 checkMatchedString __function "1 ab"             # check regular matched result
+0 checkMatchedString __function "2 cd"             # check regular matched result
```

```
+1 setSentenseString __function "xyzabcdefg"      #test string
+1 checkMatchedString __function "0 abcde"          # check regular matched result
+0 checkMatchedString __function "1 ab"             # check regular matched result
+0 checkMatchedString __function "2 cd"             # check regular matched result


+2 __end
```