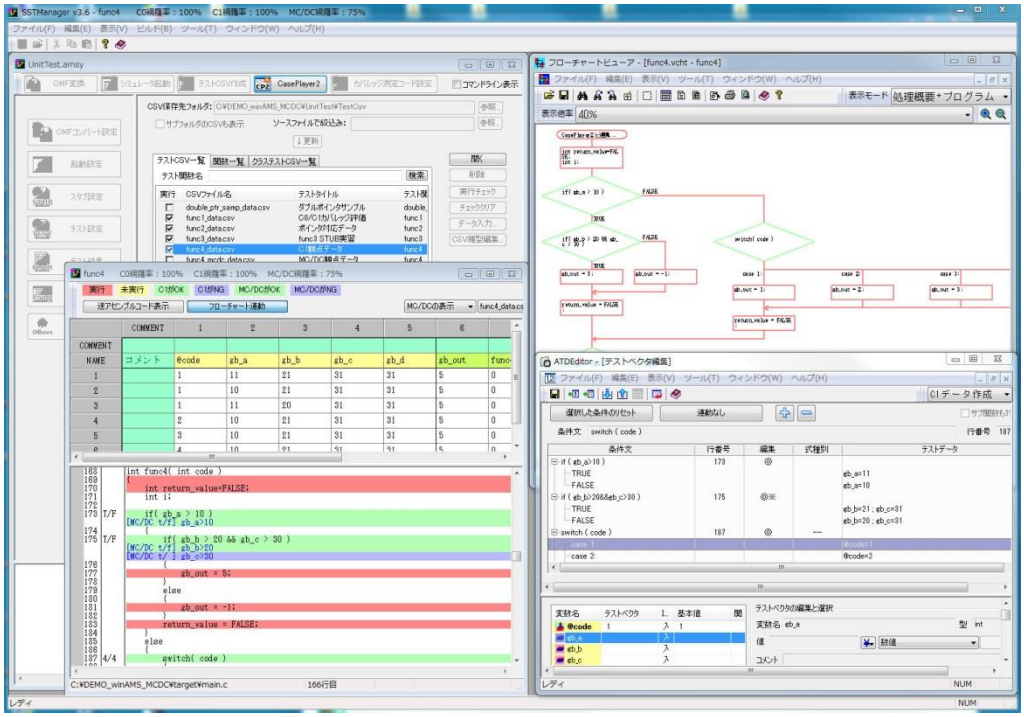


カバレッジマスターwinAMS チュートリアル

2025 年 1 月 10 日
第 6.1.0 版

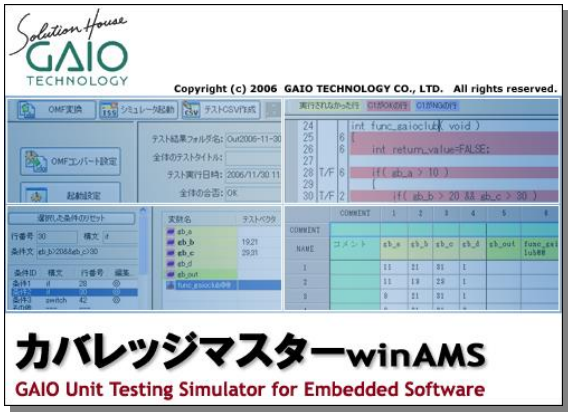
目次

カバレッジマスターwinAMS チュートリアル

はじめに

このたびは、ガイオ単体テストツール「カバレッジマスターwinAMS」をご利用いただきまして、誠にありがとうございます。カバレッジマスターwinAMS は、組込みソフト品質評価・改善のための単体テストを、効率化するための検証用ツールです。

本書は、初めてカバレッジマスターwinAMS をご利用になる方のためのチュートリアルです。カバレッジマスターwinAMS の一般的な使い方や基本機能を、実習を通して学べるように構成されています。



カバレッジマスターwinAMS のツール概要

カバレッジマスターwinAMS をご利用になるために知っておく必要がある、ツール構成や動作概要について説明します。

組込みソフト向けの単体テストツール

カバレッジマスターwinAMS は、「組込みソフト向け」の単体テストツールです。最大の利点は、ソフト(関数)を動作させてテストする環境に、マイコンシミュレータ(System Simulator)を利用している点です。

このマイコンシミュレータ(System Simulator)は、皆様が製品開発に使用するクロスコンパイラで生成した、組込みマイコン用の実行コード(オブジェクト)をそのままテスト実行に使用します。実際に製品に組み込まれるコードと同一の実行コードで単体テストが行えるため、C 言語でコーディングされたプログラムロジックの評価に加え、マイコン演算精度、メモリエラー、クロスコンパイラの最適化による影響など、実装時の問題を含めて検証することができます。

カバレッジマスターwinAMS は、以下のアプリケーションで構成されています。

SSTManager ： 単体テスト管理、実行環境アプリケーション
System-G/SX ： マイコンシミュレータ(System Simulator) － 組込みマイコンの種類毎に提供されます
winAMS ： 単体テストシミュレータ － 単体テスト実行を行う、仮想単体テスト装置に当たります

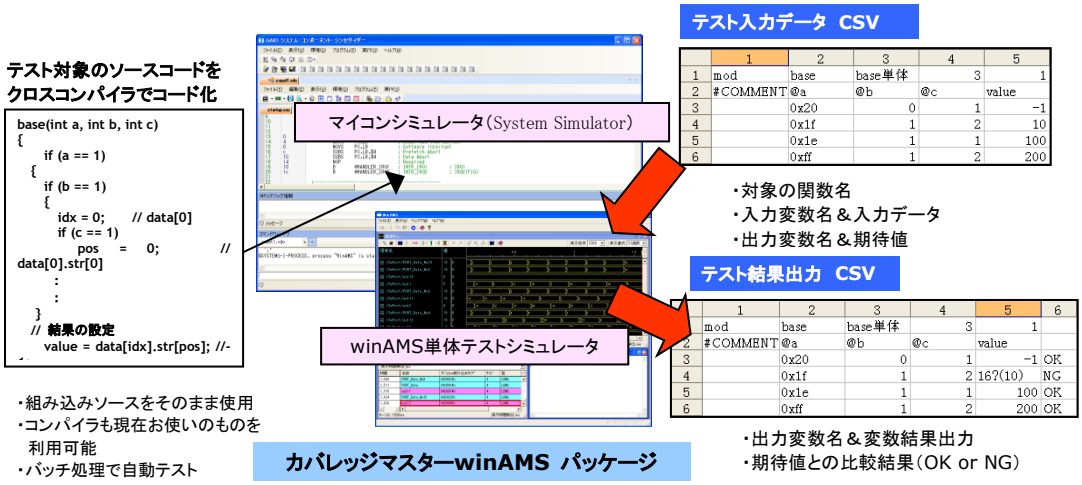
これらの他に、組込みマイコンへの実行コードを生成するクロスコンパイラや統合開発環境(IDE)が必要です。

テスト入出力には汎用的なCSV形式のファイルを使用

カバレッジマスターwinAMS は、単体テストの精度を保ち、さらに単体テストの作業効率を上げるために、テスト対象のソースコードを書き換えることなく、テストが実行できるように工夫されています。

入出力対象の変数指定やスタブ関数の設定など、単体テストの設定項目は、全て CSV ファイルで記述する様になっており、試験対象の関数に特殊なテストコードを追加したり、入力データをソースへ取り込んだりする必要はありません。

以下の図が、カバレッジマスターwinAMS の構成の概要を示しています。



まず、テスト対象の関数を含む組込みソース(全てでも一部でも可)を、クロスコンパイラでコード化したものを、マイコンシミュレータにロードしておきます。この実行コードは、リンカーでリンクを行い、実行可能なオブジェクトである必要があります。(コンパイラがソース毎に生成する、中間オブジェクトファイルは利用できません。)

上の図の関数「base()」をテストするためには、1つの CSV ファイルを作成する必要があります。この CSV ファイルには、対象の関数名、入力データを設定する入力変数シンボル名、関数実行後に値を評価する出力変数シンボル名などのテスト条件と、テストデータ(テストベクター)を記述します。

上の例では、「@a,@b,@c」のセルは、入力条件として、a,b,c の引数を指定するための記述であり、「value」は、関数実行結果として期待値を評価するグルーバル変数「value」を指定する記述となっています。

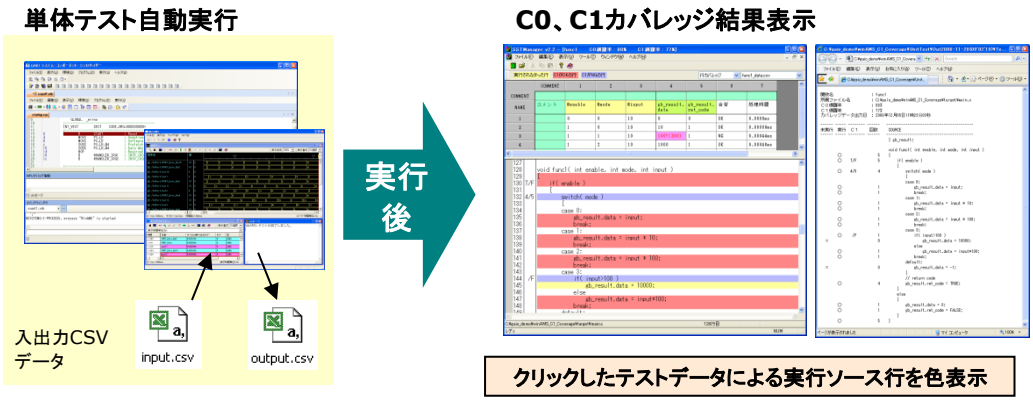
評価対象の出力変数のデータには、期待値を設定することができます。テスト実行後に出力結果を出すだけであれば、期待値欄は空欄のままでも構いません。

期待値が設定されている場合は、テスト実行後に結果と期待値との照合を行い、NG/OK をレポートします。また、期待値と結果が異なるデータを、「16?(10)」(期待値に 10 を設定したが、結果が 16 だった場合)のように示します。

単体テストを自動実行して入出力結果とカバレッジを自動出力

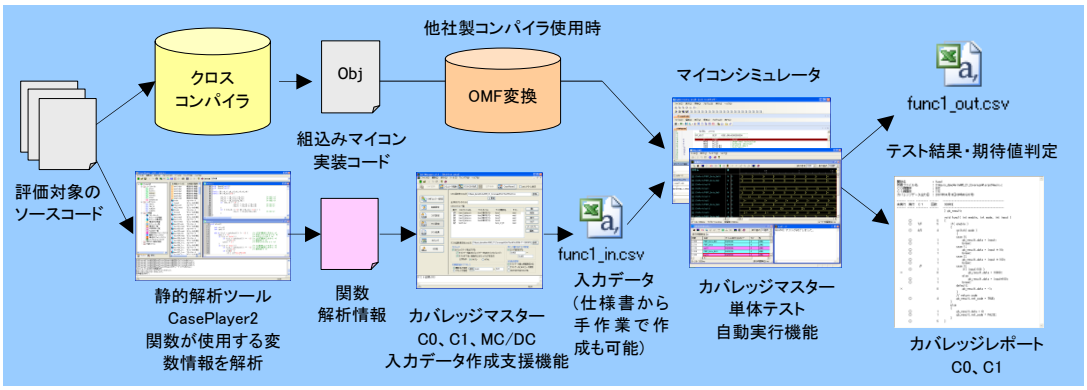
テストデータをCSVファイルで作成するだけで、テスト実行の作業は完全に自動化されます。「シミュレータ起動」のボタンを押すだけで、設定された入出力テストを自動実行し、C0、C1 カバレッジ(注 1)の結果を自動レポートします。カバレッジ結果は、入力データとカバレッジ結果の対応を解析できる専用のカバレッジビューに表示され、テストレポートとしてカバレッジ結果をテキストファイルに出力することも可能です。

(注1):C1 カバレッジは、CasePlayer2 と連携使用時のみ出力可能



CasePlayer2 で評価ソースを解析して カバレッジテストデータを自動生成

カバレッジマスターwinAMS に、別製品のプログラム解析ツール「CasePlayer2」を組み合わせて使用することで、単体テスト入力データ作成を支援する機能が利用できます。この機能を利用する際のツールチェーンは、下図のようになります。



テストデータ作成の支援機能の概要は次の通りです。

入出力変数の自動検索： 評価対象の関数が参照/代入している外部変数を自動検索します
C1、MC/DC カバレッジを満たすテストデータ自動作成、作成支援

この機能は、本チュートリアル後半の実習4で詳しく解説します。

MC/DC 計測機能について

カバレッジマスターwinAMS には、MC/DC のカバレッジ計測を行う機能（オプション機）が用意されています。これは、自動車機能安全の構造カバレッジ計測などにおいて要求される、分岐条件を網羅するテストのための機能です。

MC/DC 計測機能を使用するには、「MC/DC オプション」のライセンスが必要です。環境設定や使用方法については、後の章（【応用編】埋め込みコードによるカバレッジ計測）で解説しています。

関数カバレッジ/コールカバレッジ計測機能について

カバレッジマスターwinAMS V3.6 において、「関数カバレッジ/コールカバレッジ計測機能」（オプション機能）が追加されました。この機能は、単体テストの次のフェーズである結合テストフェーズで使用する機能です。関数単体ではなく、関数を結合した状態で、上位の関数を駆動することで、想定されるサブ関数が網羅して実行されるかを計測します。

この関数カバレッジ/コールカバレッジ計測機能を使用するためには、「関数/コールカバレッジ計測オプション」のライセンスが必要です。環境設定や使用方法については、後の章（【応用編】関数カバレッジ、コールカバレッジ計測）で解説しています。

C++言語に対する単体テストについて

カバレッジマスターwinAMS は、C++言語に対する単体テストを行う機能（オプション機能）がサポートされています。C++特有のクラスに定義されたメソッド（関数）に対するテストや、クラスのコンストラクタ（初期化関数）の動作を考慮したテストなどが行えます。

C++言語に対する単体テスト機能を使用するには、「C++オプション」のライセンスが必要です。C++言語については、本チュートリアルでは扱っておりません。

実習の準備： さあ、はじめましょう！

実習環境の準備

事前に下記の準備が必要です。

1. 教材のインストール
   - サンプルソース「winAMS_CM1.zip」を C ドライブにコピー後、解凍して下さい。
2. ご利用のマイコン用クロスコンパイラ（開発環境）のインストール
   - 別途準備が必要です。（本チュートリアルには含まれておりません）

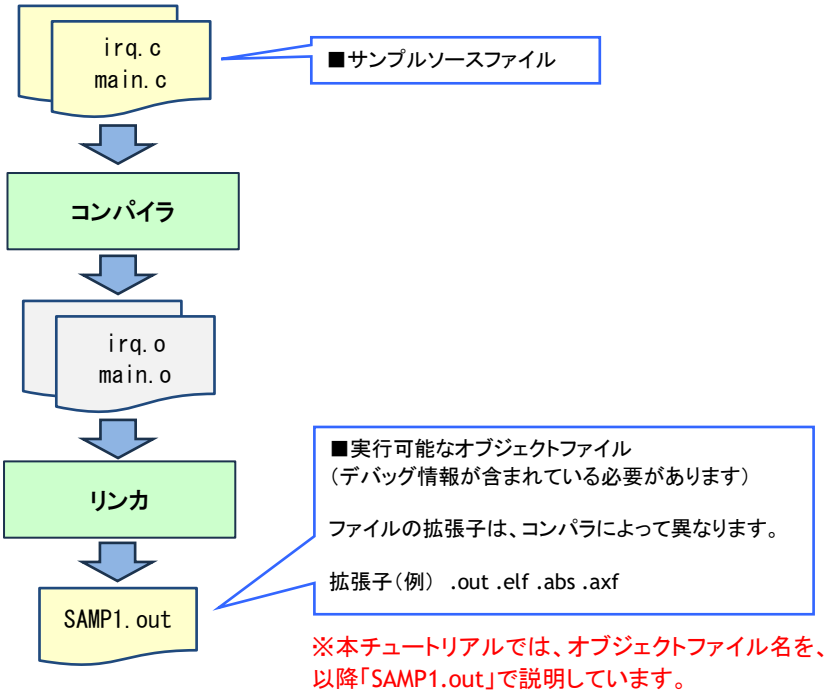
サンプルソースのコンパイル

最初に、実習で使用するサンプルコードを確認します。
以下のソースファイル（2ファイル）が「C:¥winAMS_CM1¥target」フォルダに用意されています。

main.c ： 評価対象の関数 func1()～func5() と main()関数があります。
irq.c ： C 言語で記述された割り込みハンドラ関数があります。（※実習では使用しません）

ソースファイル main.c に、実習で使用するテスト対象の関数が入っています。また、実際はアプリケーションコードが記述される main 関数もありますが、このサンプルコードは空関数になっています。カバレッジマスターwinAMS は、実際のマイコンチップと等価な命令コード実行を行うマイコンシミュレータ（System Simulator）を利用するため、リンカーで実行可能なコードを生成する必要があります。このため、main 関数を記述して、main のエントリを生成しています。

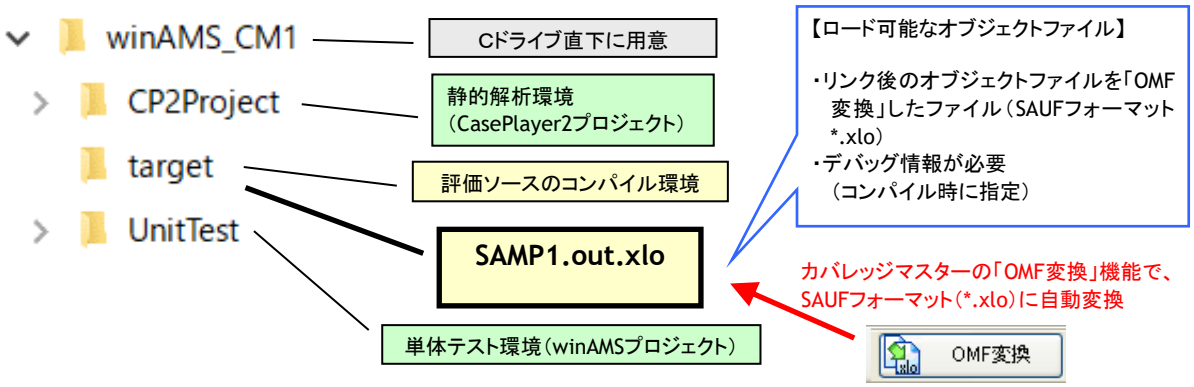
このサンプルソースを、別途ご利用のコンパイラ（開発環境）でビルドを行い、実行可能なオブジェクトを生成してください。※この実行可能なオブジェクトファイルは、「デバッグ情報」を含んでいる必要があります。



実習環境のファイル構成について

本実習環境のファイル構成は以下の通りです。C:¥winAMS_CM1¥target フォルダに、実行可能なオブジェクトファイル（SAMP1.out）を生成してください。
カバレッジマスターwinAMS がテストに使用できるのは、SAUF フォーマット「*.xlo」のオブジェクトだけです。
カバレッジマスターwinAMS で使用するためにはオブジェクト変換機能「OMF変換」を使用して、コンパイラが生成した実行オブジェクトファイルを、「*.xlo」形式に変換する必要があります。この変換作業は、自動化することが可能ですので、通常のテスト作業において、テスト担当者が意識することはありません。

また、この実行可能オブジェクトファイルは、「デバッグ情報」を含んでいる必要があります。カバレッジマスターwinAMS で単体テストを行うためには、このデバッグ情報が必須です。コンパイラに与える最適化などのその他のコンパイルオプションは、そのまま使用することができます。



実習1： 一連のテスト実行フローを体験する（導入）

課題設定について

まず最初は、単体テストをするためのCSVファイルを作成して、テスト実行後に結果を出力する一連のテスト作業を体験してみましょう。この実習では、テスト入力 CSV ファイルの作成方法と、入出力テスト結果、C0 カバレッジ結果の確認方法が学習できます。

テストプロジェクトを新規作成

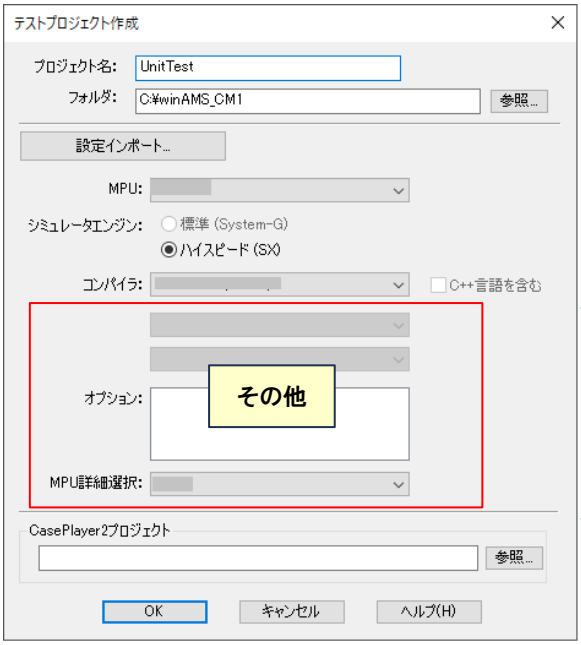
まず、テストプロジェクトを作成します。

1. Windows「スタート」→「GAIO winAMS」→「winAMS(SSTManager)」を起動します。
2. 「ファイル」メニュー→「プロジェクト新規作成」を選択します。

テストプロジェクト作成のダイアログで、以下の様に設定します。

3. プロジェクト名に、「UnitTest」と入力します。（この名称のフォルダが作成されます）
4. 参照ボタンを押して、フォルダに、「C:¥winAMS_CM1」を選択します。（この位置に UnitTest フォルダが保存されます。）
5. 「シミュレータエンジン」では「標準（System-G）又はハイスピード(SX)」を選択します。
6. 「MPU」にマイコン名を選択します。
7. 「コンパイラ」にコンパイラ名を選択します。
8. 「その他」は、選択マイコン／コンパイラによって設定内容が変わります。
   ※詳細については、Windows メニューの「GAIO winAMS」→「MPU 設定マニュアル」から、該当のマイコン／コンパイラの設定情報を参照してください。
9. CasePlayer2 プロジェクトは空欄のままにします。（後の実習4で使用します。）

これにより、C:¥winAMS_CM1 フォルダが作成され、この中にテストに関するデータを格納するための、UnitTest フォルダが作成され、テストプロジェクトが開きます。



← プロジェクト名のフォルダができます
← プロジェクト名のフォルダが作成されます
「C:¥winAMS_CM1」を選択

← マイコン名を選択

← シミュレータエンジンを選択

← コンパイラ名を選択 ※C++言語を含むはOFF

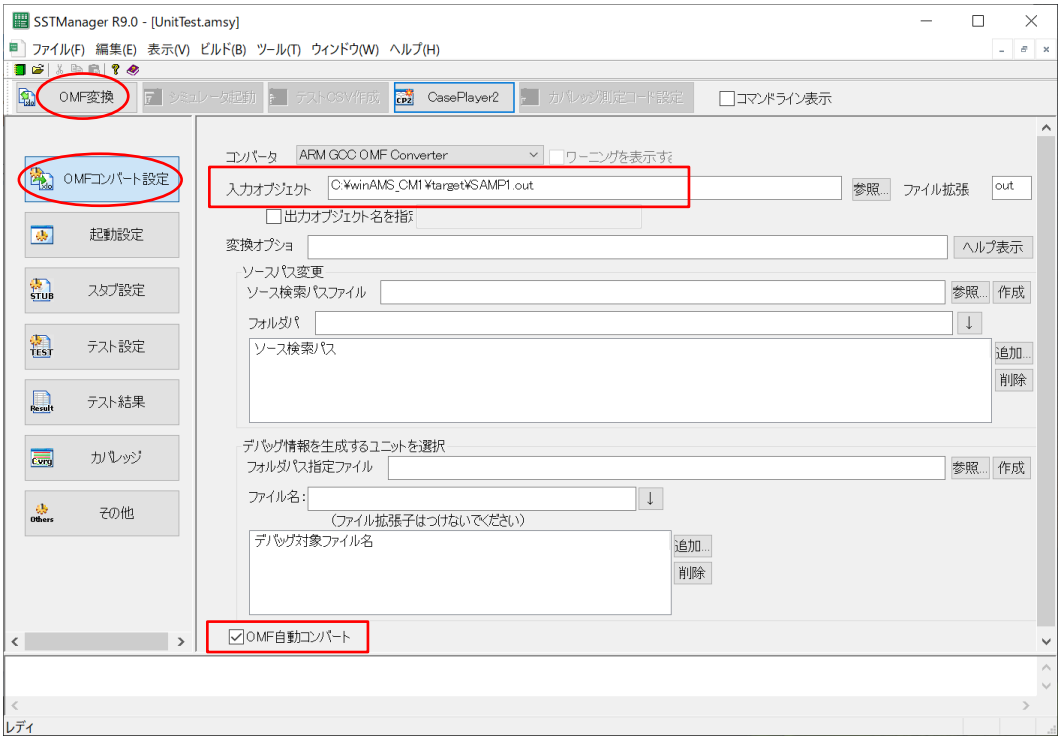
← 選択マイコン／コンパイラによって設定内容が変わります。

← 実習1～3では空欄
※実習4で、テストデータ作成支援機能を利用する際に使用します。

OMF コンバート設定

プロジェクトが開いたら必要な設定を行います。最初に、「OMF変換」を行って、コンパイラが生成した実行オブジェクトファイル(*.out)を、「*.xlo」形式に変換します。

1. 左に並んだ最上部の「OMF コンバート設定」ボタンを押します。
2. 設定項目の上から2行目「入力オブジェクト」へ、コンパイラが生成した実行オブジェクトファイル「C:¥winAMS_CM1¥target¥SAMP1.out」を設定します。
3. 設定項目の下部「OMF 自動コンバート」をチェックします。(デフォルトでチェックが入ります) 本チェックを入れておくことで、以降実行オブジェクトファイルが更新される度に自動的に OMF 変換されます。



次に、OMF 変換を行います。

4. 上部の「OMF 変換」ボタンを押します。
   - 下部のメッセージビューへ変換結果が表示されます。

```
変換を始めます。

  初期化中...

  ヘッダ情報を変換中...

  ユニット情報を変換中.....

  セクション情報を変換中.....

  デバッグ情報を変換中....

  処理を終了しています...

変換が完了しました。 出力した ﾃﾞﾊﾞｯｸﾌｧｲﾙ数(5) ﾃﾞﾊﾞｯｸｼﾝﾎﾞﾙ数(139)
```

その他の設定（EXCEL を CSV エディタに使用するための設定）

プロジェクトが開いたら、最初に CSV 作成のために、Microsoft EXCEL（※使用する PC に EXCEL がインストールされている必要があります。）を使用するための設定を加えます。

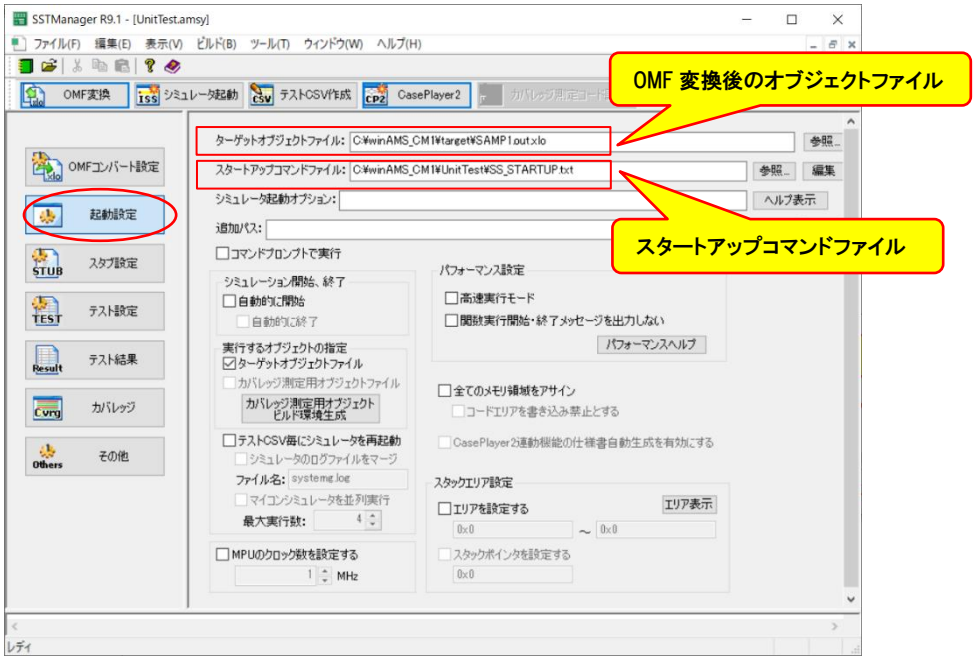
1. 左に並んだ一番下のボタン「その他」を押します。
2. 設定項目の上から2行目「テスト結果 CSV ファイルを外部エディタで開く」がチェックされていることを確認します。 ※チェックされていないときはチェックをしてください。

テスト対象の指定と起動設定

テスト対象の設定を行います。最初に、前章でコンパイルして作成したオブジェクトファイルを OMF 変換した「SAMP1.out.xlo」を指定します。このファイルには、デバッグ情報が含まれており、テスト内容を設定するための変数シンボル名や、ソースファイルパス、ソースコード行と実行コードの対応情報（ライン情報）などが含まれています。テスト対象に指定するファイルは、この.xlo ファイル1つだけです。

1. 左に並んだ上から2番目の「起動設定」ボタンを押します。
2. ターゲットオブジェクトファイルに「C:¥winAMS_CM1¥target¥SAMP1.out.xlo」が登録されていることを確認します。

次に、マイコンシミュレータの環境設定のためのファイル「スタートアップコマンドファイル」が設定されていることを確認します。スタートアップコマンドファイルは、マイコンシミュレータ（System Simulator）起動時に実行されるコマンド（スクリプト）ファイルで、毎回のテスト環境を同じ設定で起動するために便利な機能です。

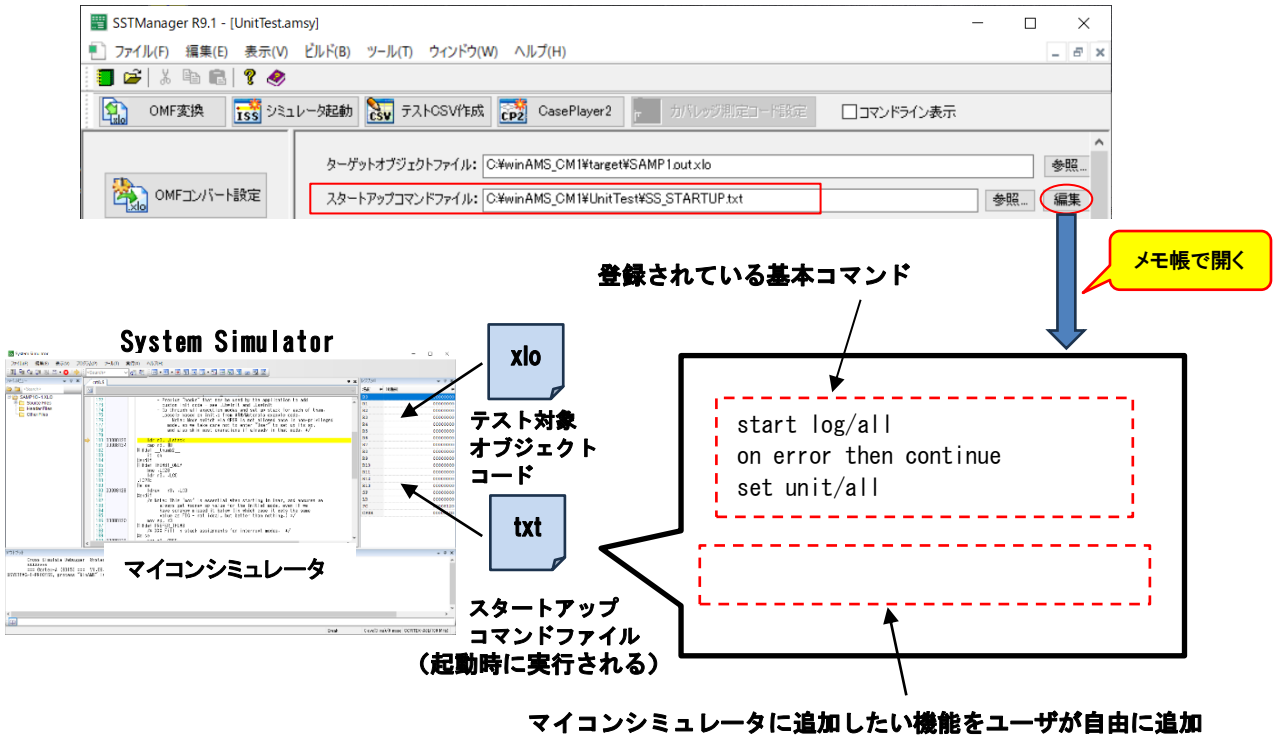


スタートアップコマンドファイルについて

スタートアップコマンドファイルとは、マイコンシミュレータ起動時に読み込まれ、マイコンシミュレータの設定を変更するためのスクリプトファイルです。単体テストの事情に合わせて、マイコンシミュレータの設定や動作を変更するために使用します。例えば、以下の様な場合に使用します。

- ROM 属性(read only)のメモリ領域を、書き換え可能な RAM 領域に変更したい
- マイコン周辺ハードウエア、ペリフェラルの応答を待つループをスキップしたい
- 関数実行中の変数の変化を出力したい
- テスト実行前にメモリ領域をまとめて初期化したい

スタートアップコマンドファイルは、以下のような内容です。マイコンシミュレータ起動時に、上から順に全てのコマンドが実行されます。右にある「編集」ボタンを押すと、メモ帳で内容を確認することができます。



最初の3行は、マイコンシミュレータを単体テストで使用するために必要な基本コマンドです。デフォルトで作成されるファイルには、この3行のみが入っています。

マイコンの設定や動作状態を変える必要のない場合は、デフォルトコマンドのみでテスト可能です。
シミュレーション実行時、メモリ属性に違反した動作が発生した場合は、シミュレーションエラーとなります。

スタートアップコマンドファイルに記載するマクロコマンドの例は、ガイオ WEB サイトのユーザー向け技術サポート情報に掲載されています。

▼ユーザー向け技術サポート情報
http://www.gaio.co.jp/support/user/tech_paper.html

「マクロ(シミュレータコマンド)の使い方 ※上級者ユーザ向け」を参照してください。
サンプルファイルをダウンロードできます。

（参考）マイコンリセットから単体テストへの動作

ここで、カバレッジマスターwinAMS で単体テストを行う際の動作フローについて説明しておきます。テスト環境を構築する上で、参考にして頂ける情報です。

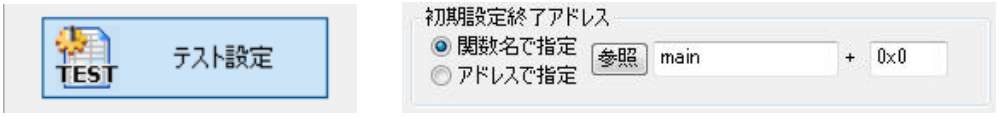
カバレッジマスターwinAMSは、組込みマイコンの命令コードを実行するシミュレータ（System Simulator）を使用して、テスト対象の関数を実行します。この動作は、評価ボード上に実装したマイコンチップで動作させるのと同じ原理です。

通常、マイコンシステムは、電源投入時にハードウエアリセットがかかると、リセットベクタ（開始アドレス）にプログラムカウンタ（PC）がセットされ、動作が始まります。プログラムの先頭には、「スタートアップルーチン」と呼ばれるマイコン機能の初期化プログラムがあり、ここを実行して、スタックポインタ設定やマイコンレジスタの初期化など、マイコンが動作するのに必要な設定が行われます。

スタートアップルーチンの実行の最後に、C言語のmain()関数へのジャンプ命令が実行され、アプリケーション実行のフェーズに移るのが、一般的なマイコンシステムです。

カバレッジマスターwinAMS で単体テストを行う際は、まず最初に、このスタートアップルーチンを実行してシミュレータのスタックポインタやレジスタを設定します。スタートアップルーチンが終了して、main()関数へ移動した際に、アプリケーション実行のフェーズには移らず、シミュレータが強制的にプログラムカウンタを評価対象の関数へセットして、単体テストのフェーズへ移るようになっています。

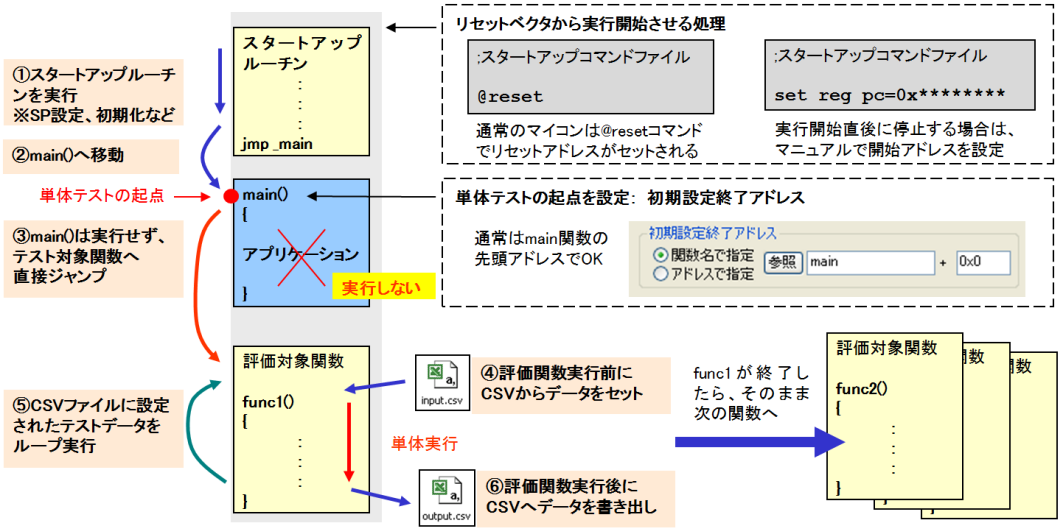
「テスト設定」の項目にある、「初期設定終了アドレス」は、単体テストへの起点となるアドレスを設定するためのものです。通常は main()関数のアドレスを指定しますが、スタートアップルーチンの終了を他のアドレスに設定したい場合は、ここで設定できます。



マイコンシミュレータ起動時に、正しくスタートアップルーチンを起動するためには、リセット時の開始アドレスが PC（プログラムカウンタ）に設定される必要があります。各マイコンシミュレータには、リセット時の開始アドレスを自動設定する「@reset」コマンドがあり、これがスタートアップコマンドファイルに記載されていますが、マイコンの型番などによっては、リセットアドレスが変更されている場合があります。このときは、@reset の代わりに、

```
set reg pc = 0x********
```

のコマンドをスタートアップコマンドファイルに記載することで、リセット時の PC 値を指定できます。リセット時のアドレス値は、マイコンのハードウエアマニュアルを参照して指定してください。



モジュールテスト用 CSV ファイルの雛形作成

では、実習1の課題に入ります。実習1はサンプルソースにある関数 func1()を単体テストするためのテストデータを作成して、C0 カバレッジを 100%にすることを目標にします。
以下は、サンプルソース(main.c)に記載されている関数 func1()です。

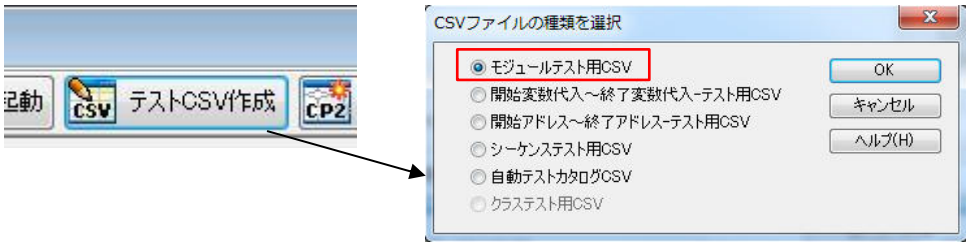
```
// グローバル構造体変数
struct ST_PARAM
{
     int data;
     int ret_code;
} gb_result;

void func1( int enable, int mode, int input )
{
     if( enable )
    {
          switch( mode )
          {
          case 0:
               gb_result.data = input ;
               break;
          case 1:
               gb_result.data = input * 10;
               break;
          case 2:
               gb_result.data = input * 100;
               break;
          case 3:
               if( input >100 )
                    gb_result.data = 10000;
               else
                    gb_result.data = input *100;
               break;
          default:
               gb_result.data = -1;
          }
          // return code
          gb_result.ret_code = TRUE;
     }
     else
     {
          gb_result.data = 0;
          gb_result.ret_code = FALSE;
     }
}
```

まず、この関数の入出力条件を確認します。赤色で書かれた部分が入力となる変数の記述です。このソースでは3つの引数がそれにあたります。この関数には戻り値はありません。実行結果は、関数のすぐ前に宣言されているグローバルの構造体、「gb_result」に格納される仕様になっています。

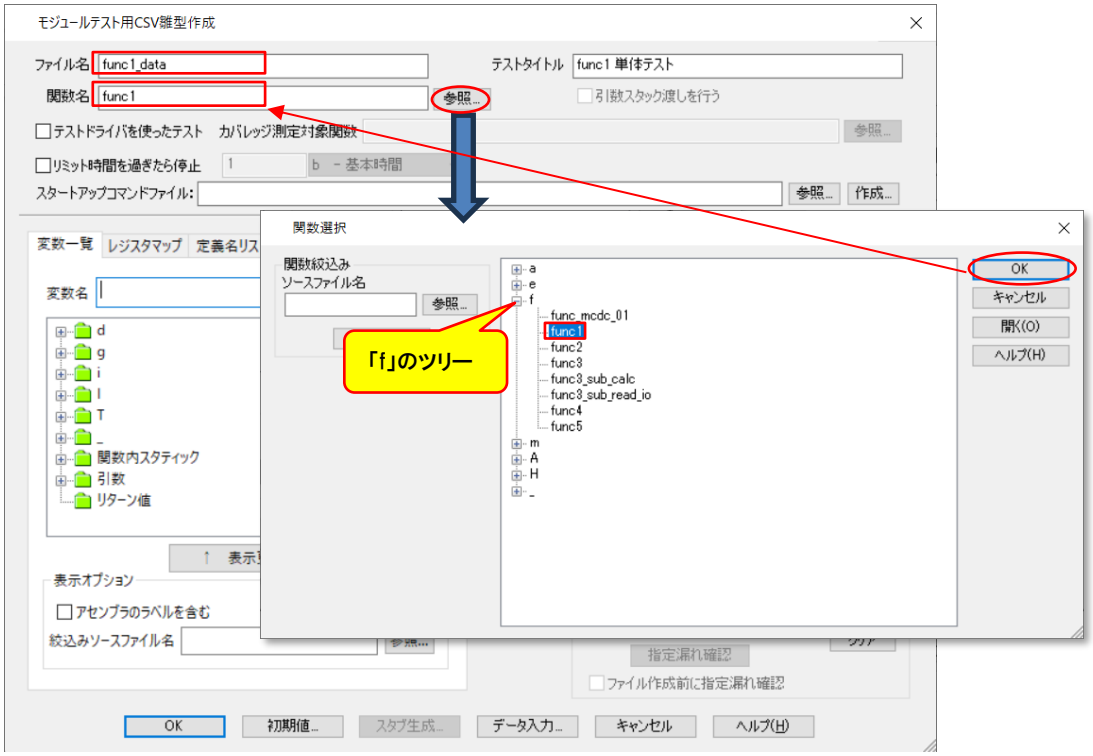
そこで、実習1では、テスト要件として、3つの引数「enable, mode, input」を入力変数に指定してテストデータを入力し、関数実行後にグローバル構造体の2つのメンバー「gb_result.data, gb_result.ret_code」を評価することにします。

では、テストのための CSV ファイルを作成します。

1. 上部の「テスト CSV 作成」ボタンを押します。
2. 「モジュールテスト用 CSV」を選択して「OK」を押します。



次の「モジュールテスト用 CSV 雛形作成」ダイアログが表示されます。



この「モジュールテスト用 CSV 雛形作成」ダイアログでは、テスト条件となる変数を選択します。まず、基本的な設定を行います。

3. ファイル名に「func1_data」を設定します。

これは、作成される CSV ファイルのファイル名の設定です。拡張子「.csv」は省略することができます。ファイル名は任意ですが、カバレッジマスターwinAMS では、1つの関数に対して1つの CSV ファイルを作成することが基本であるため、このファイル名には関数名の一部を使用すれば、管理が容易になります。
次に、テスト対象の関数を選択します。

4. 関数名の「参照…」ボタンを押して、「f」のツリーから、「func1」を選択し、「OK」を押します

参照ボタンで表示される関数名リストは、指定した評価対象のオブジェクトファイル「SAMP1.out.xlo」のデバッグ情報から取得されたものです。関数名ボックスに、直接「func1」とキーボード入力しても構いません。

5. テストタイトルに、「func1 単体テスト」と入力します

この項目の設定は任意です。テストタイトル欄は、後でテスト内容を分かり易くするためのメモ書きです。ここで設定したタイトルが、CSV ファイルリストや結果レポートに反映されます。必要に応じて使用します。

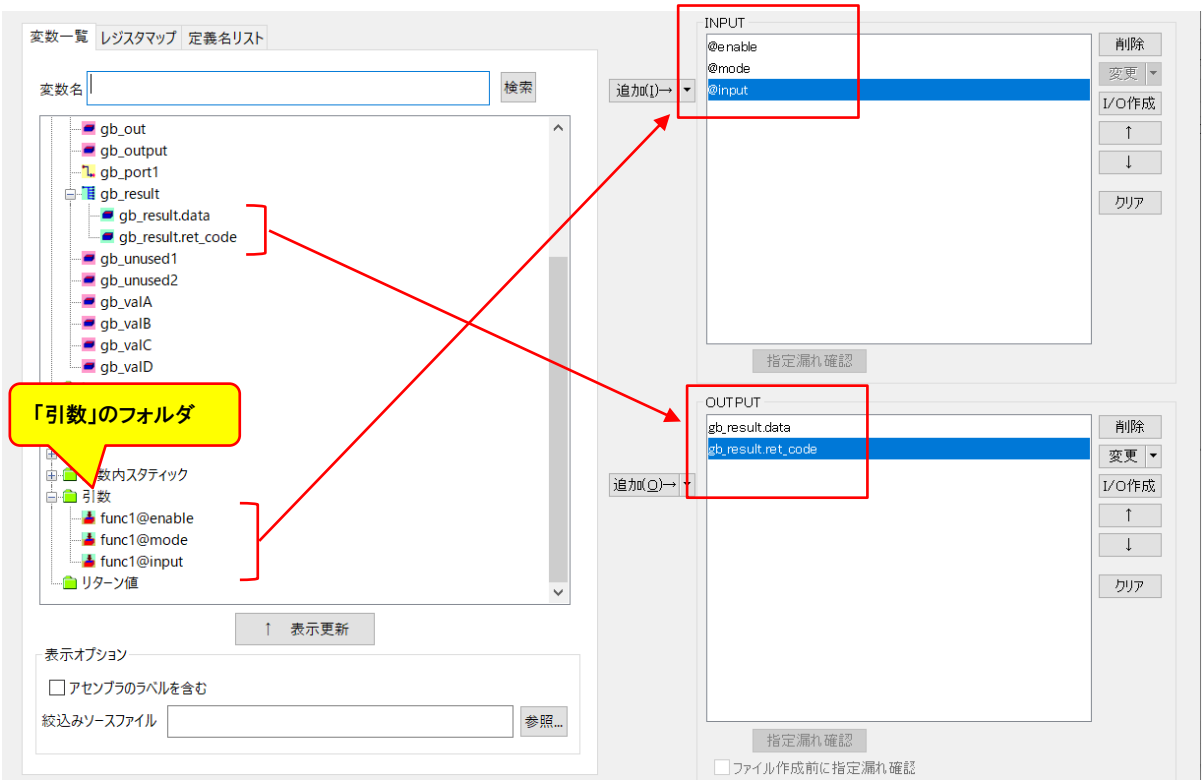
次に、func1()をテストするための入出力変数を選択します。まず、入力条件は3つの引数ですので、これらを「INPUT」のボックスに登録します。(下記の画面キャプチャを参照してください)

6. 「変数一覧」のレビューで、「引数」のフォルダを開けます (3つの引数が表示されます)
7. func1@enable、func1@mode、func1@input を順に選択して、INPUT に「追加」ボタンで登録します
※INPUT 画面では、「@引数名」の形式で表示(関数名は省略)されます

最後に、関数実行後に評価する変数を指定します。func1()では、グローバルの構造体のメンバー「gb_result.data、gb_result.ret_code」が評価対象です。

8. 「変数一覧」のレビューで、「g」のフォルダを開けます。“g”で始まるグローバル変数が表示されます

9. gb_result のツリーを開けて、gb_result.data、gb_result.ret_code を OUTPUT に追加します。
10. 「OK」を押します。



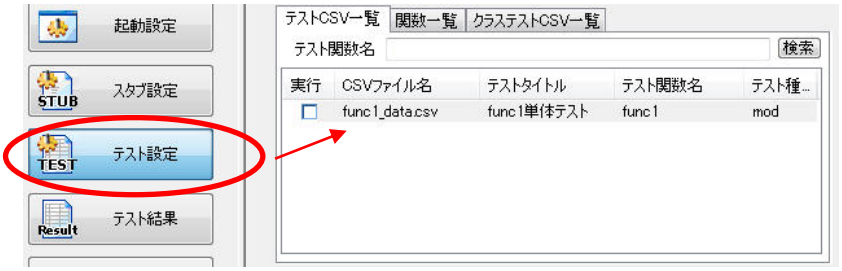
（参考）変数一覧から変数を登録する際に、変数一覧上で、[SHIFT]キーまたは[Ctrl]キーを併用すると、複数の変数を同時選択、同時登録できます。

これで、モジュールテスト用 CSV ファイル（func1_data.csv）の雛形が作成されました。

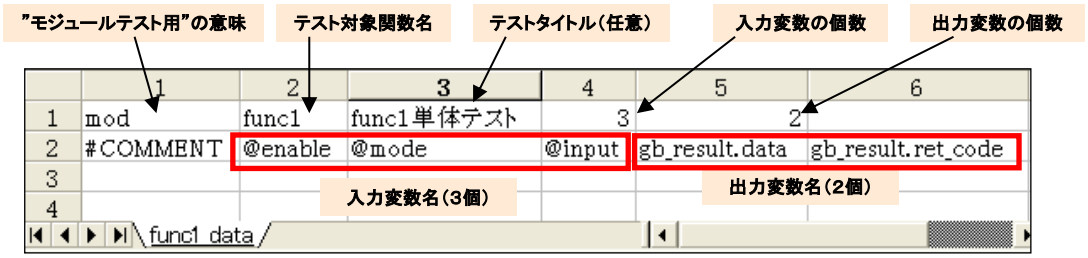
モジュールテスト用 CSV ファイルを確認

生成されたモジュールテスト用 CSV ファイル(雛形)は「テスト設定」の「テスト CSV 一覧」に表示されます。確認してみましょう。

1. ウインドウ左の「テスト設定」ボタンを押して画面を切り替えます。
2. 「テスト CSV 一覧」の「func1_data.csv」をダブルクリックして Excel を立ち上げます。

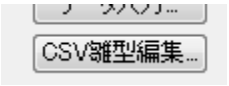


生成された CSV ファイルを EXCEL で確認してみます。作成されるのは、選択した関数名や入出力変数名などの、テスト条件の部分です。



このファイルは、カンマ区切りテキストフォーマットの CSV ファイルですので、EXCEL や一般のテキストエディタで編集することが可能ですが、一旦 CSV ファイルを生成した後で、変数の設定を追加、削除したい場合は、直接 CSV ファイルを編集せずに、もう一度「モジュールテスト用 CSV 雛形作成」ダイアログに戻って、変数の選択をやり直して下さい。既に、テストデータを入力した後でも、データを失うことなく、変数の追加を行うことができます。

再度、「モジュールテスト用 CSV 雛形作成」ダイアログに戻るには、ファイル一覧の右にある「CSV 雛形編集」ボタンを押します。



今回の実習では使用しませんが、CSV ファイルの「#COMMENT」欄について説明します。この列のセルは、入力したテストデータ行を無効にする(テストデータから除外する)ために使用します。このセルに「;」(半角セミコロン)を入力すると、その行のデータは使用されなくなります。一時的にデータを除外するために利用できます。

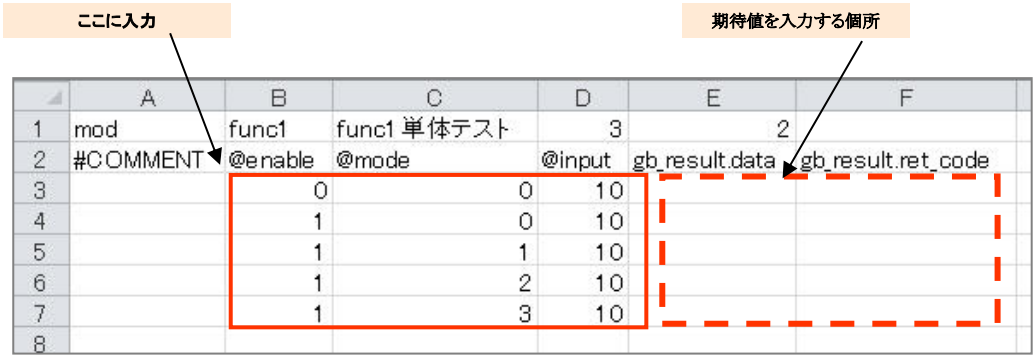
また、実際の運用においては、テストデータの意味をコメント記述するために使用できます。#COMMENT の欄にセミコロンを入れれば、その行は何を記述しても構いません。もちろん2バイト文字や日本語も問題ありません。

	A	B	C	D	E
2	#COMMENT	@enable	@mode	@input	
3	;	(ここは、自由な記述に使用できます)			
4					

モジュールテスト用 CSV ファイルにテストデータを追加

では、このモジュールテスト用 CSV ファイル(雛形)に、func1()をテストするためのテストデータを追加しましょう。まず最初は、カバレッジマスターwinAMS を動作させて、どのようなテスト結果が出力されるのかを体験してみます。EXCEL を使って、テストケースを追加します。 CSV の1行が関数1回のテスト実行に相当しますので、func1()を5回実行するためのテストデータを作成します。

1. 下図の 3～7 行目の INPUT 変数にテストデータを加えます。(5 回分のテストケース)
2. gb_result 構造体のメンバーには「期待値」が入力可能ですが、今回は空欄にしておきます。
   ※期待値の入力については、後述「func1()の再テストを行いカバレッジを 100%にする」で触れます。
3. この CSV ファイル「func1_data」を、EXCEL のファイルメニューから、上書き保存します。
4. EXCEL を閉じます。
   ※EXCEL は必ず閉じて下さい。EXCEL で CSV ファイルを開いたままの状態にしていると、カバレッジマスターwinAMS などの他アプリケーションから開こうとしてもファイルがオープンされたままになっているため、この CSV ファイルを開くことができません。

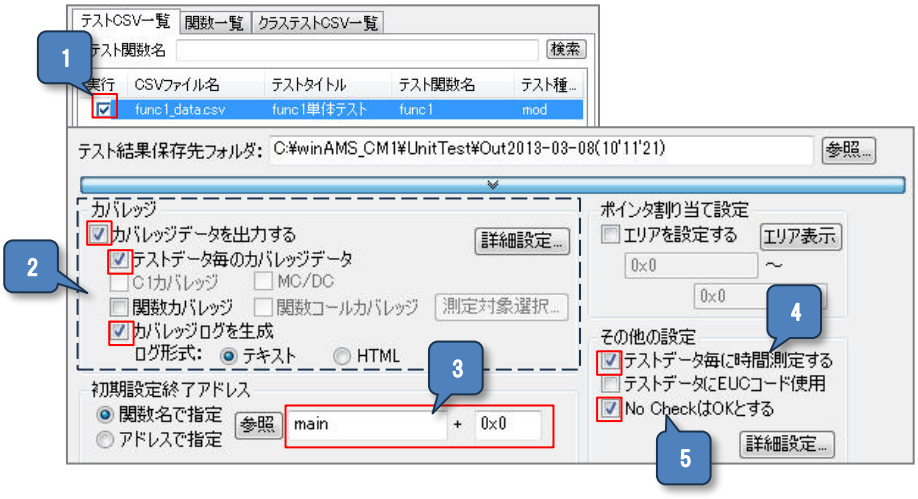


	A	B	C	D	E	F
1	mod	func1	func1 単体テスト	3	2	
2	#COMMENT	@enable	@mode	@input	gb_result.data	gb_result.ret_code
3		0	0	10		
4		1	0	10		
5		1	1	10		
6		1	2	10		
7		1	3	10		
8						

func1()のテスト実行を行う

作成した CSV ファイルで、func1()の単体テスト実行を行いましょう。その前に、下記(1～5)の設定項目を確認します。設定されていない項目は設定してください。

1. 「テスト CSV 一覧」で「func1_data.csv」の左のチェックボックスを ON にします。
2. 下図のように、「カバレッジ」欄のチェックボックス(3箇所)を ON にします。
3. 「初期設定終了アドレス」を「main」＋「0x0」にします。
4. 「その他の設定」欄の「テストデータ毎に時間計測する」を ON にします。
5. 「その他の設定」欄の「No Check は OK とする」を ON にします。



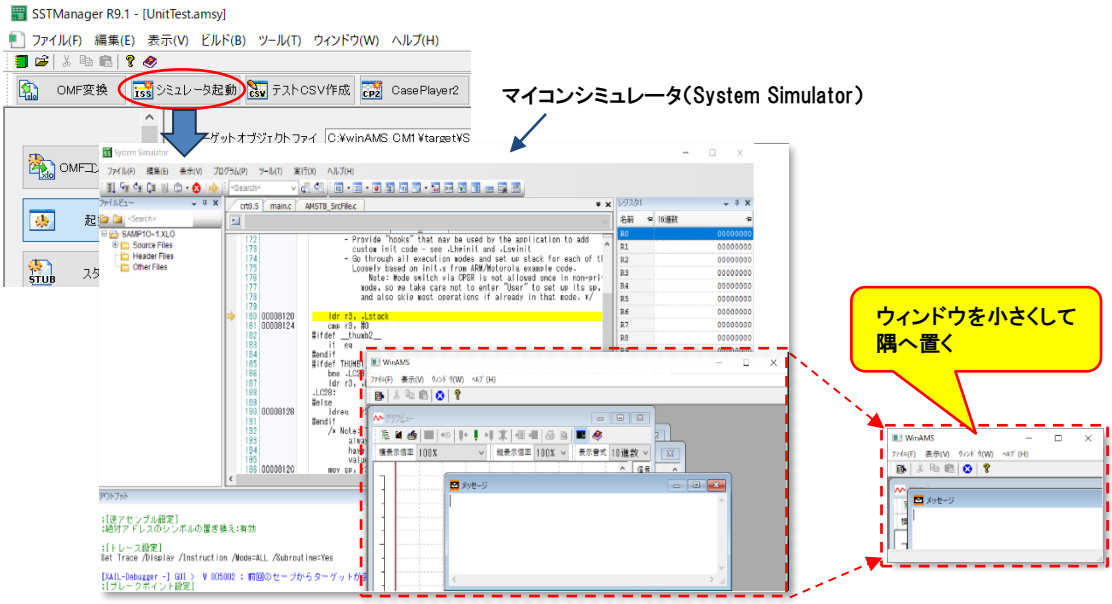
「その他の設定」欄の「テストデータ毎に時間計測する」の機能は必須設定ではありません。この機能は、テストデータをマイコンで処理した際に掛かる時間を、テストデータ毎に出力するものです。計測された時間は、出力 CSV ファイルに書き込まれます。

ただし、この計測時間は、実行した各命令当たりの実行サイクル数(クロック数)の累積を基に換算されます。実際のマイコンハードウエアで発生するメモリへの Read/Write 時間、パイプライン処理、キャッシュ実行などの動作時間は考慮されません。そのため、ここで計測される時間は、実際のマイコンチップでの実行時間とは厳密には一致しません。

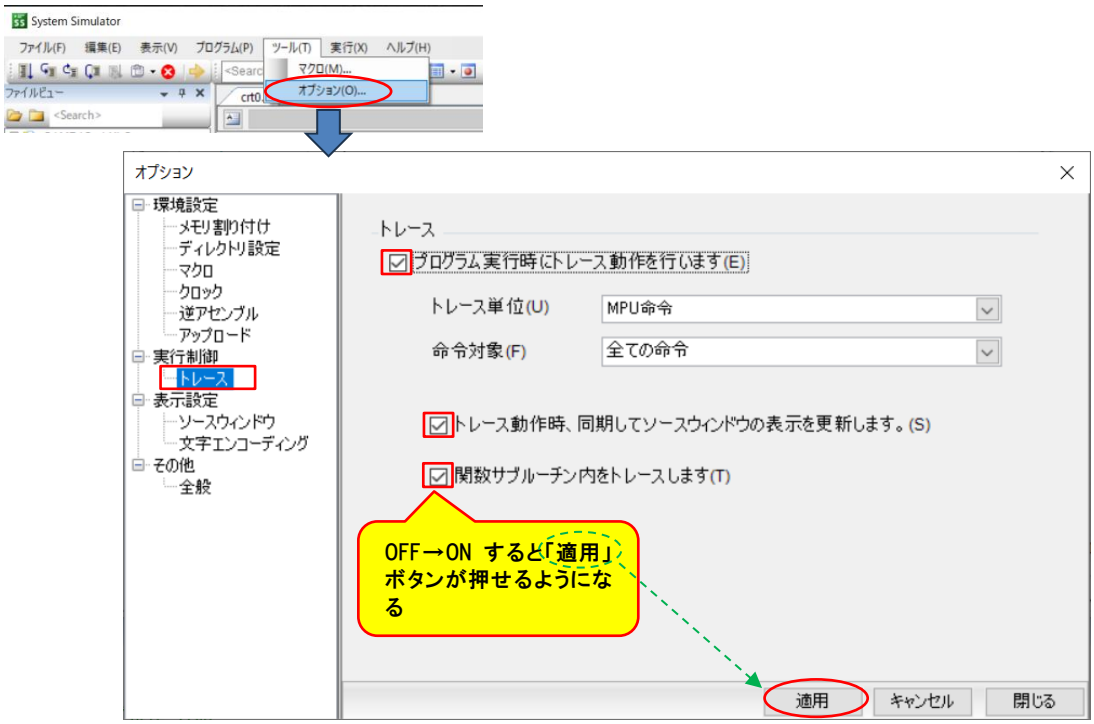
時間換算のためのマイコンのクロック数の指定は、「起動設定」ビューの、「MPU クロック数を設定する」で変更できます。本実習では、使用するマイコンが「100MHz」であると仮定して、「100」に設定して下さい。

では、マイコンシミュレータ(System Simulator)と単体テストシミュレータ(WinAMS)を起動して、テストを実行してみます。

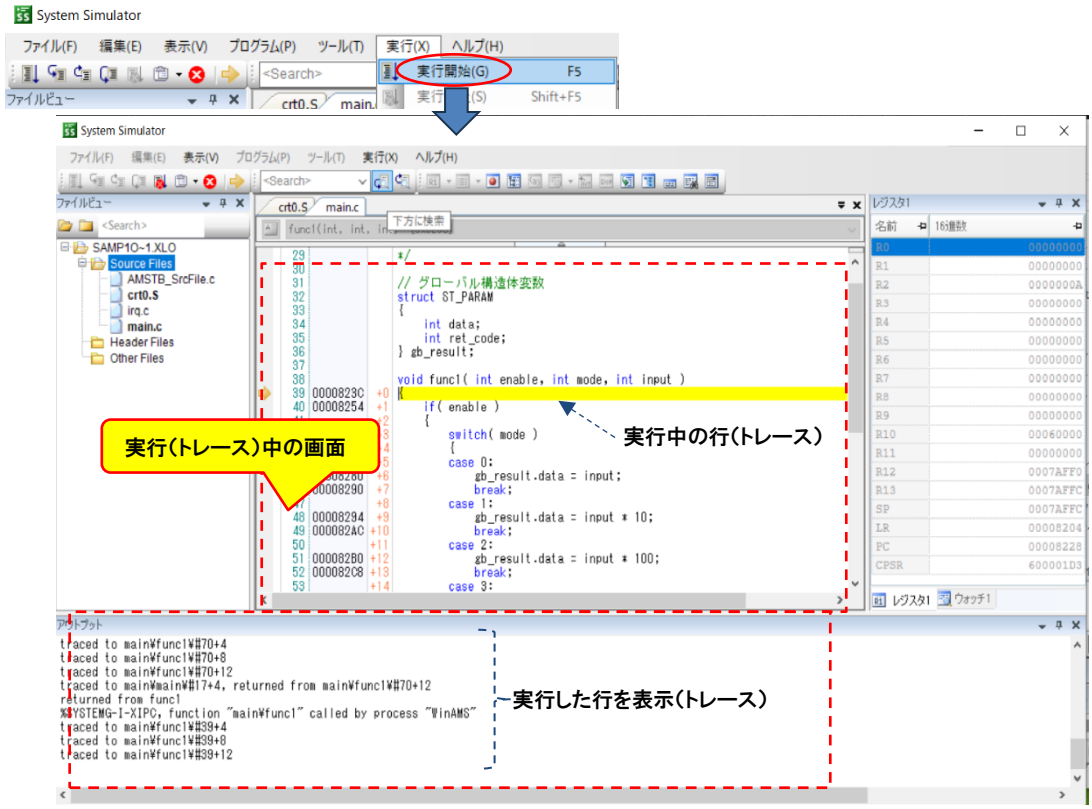
6. ウィンドウ上部の「シミュレータ起動」ボタンを押します。
7. マイコンシミュレータ(System Simulator)と単体テストシミュレータ(WinAMS)が起動します。
8. 「WinAMS」ウインドウを小さくして デスクトップの隅に置きます(このウインドウは使用しません)。



9. マイコンシミュレータ画面の「ツール」メニューから、「オプション」を選択します。
10. オプション画面(下図)のトレース項目のチェックボックス(3箇所)を ON にします。OFF→ON 操作することで「適用」ボタンを押すことがきるようになります。 ※本設定により、ソースコードの実行個所(ソースコードウィンドウの黄色で表示される行)がトレースされます。
11. 「適用」ボタンを押して、オプション画面を閉じます。

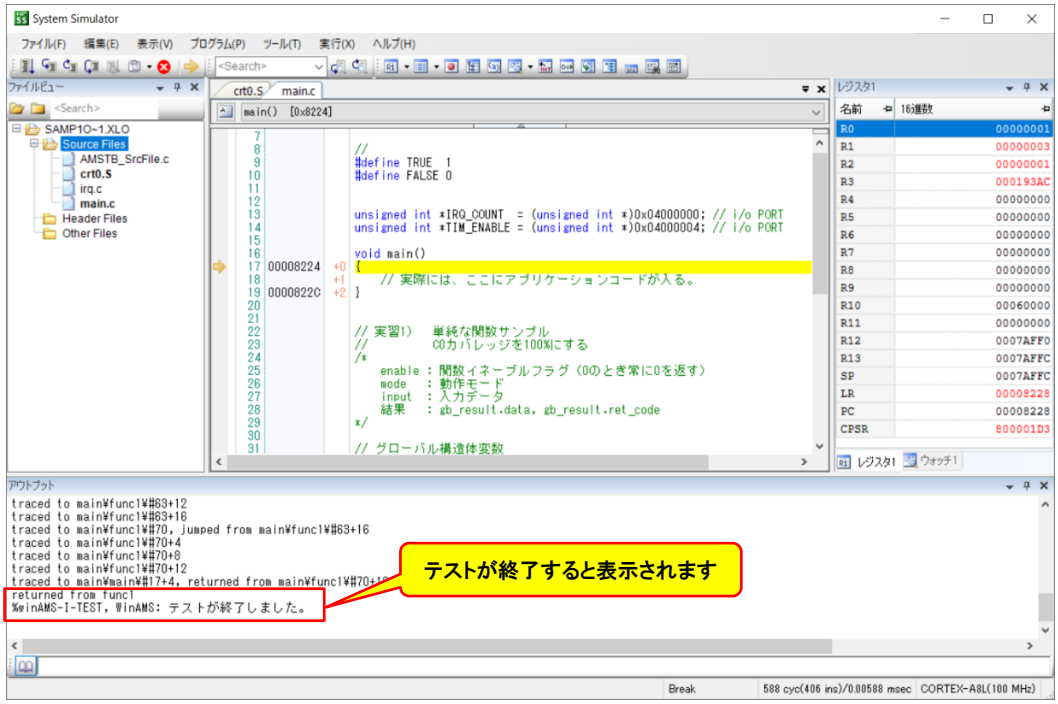


12. マイコンシミュレータ(System Simulator)の「実行」メニューから、「実行開始」を選択します。※マイコンシミュレータ(System Simulator)のソースコードのトレースが始まります。トレース位置が現在テスト実行している箇所を示しています。



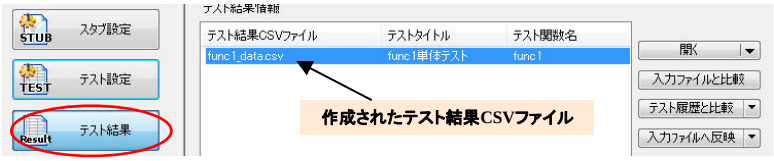
ここで利用した「トレースモード」は、実行行に合わせてソースコード画面を更新するため、実行速度が遅くなります。通常のテストでは、このトレースモードは使用しません。これ以降は、このトレースモードを OFF にして利用します。(→「func1()再テストを行いカバレッジを 100%にする」を参照)

「テストが終了しました。」が表示されたらテスト終了です。



13. マイコンシミュレータ(System Simulator)の「ファイル」メニューから、「終了」を選択します。

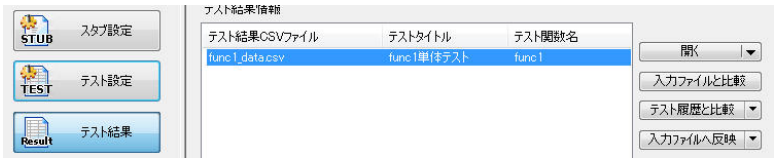
マイコンシミュレータ(System Simulator)が終了すると、SSTManagerの「テスト結果」ウィンドウに、テスト結果CSVファイルが作成されます。※入力用 CSV とファイル名は同一ですが、入力用 CSV とは別のフォルダに生成されています。



func1()入出力テスト結果を確認

まず、入出力テスト結果を確認します。

1. テスト結果「func1_data.csv」をダブルクリックします。EXCEL が起動しテスト結果が表示されます。

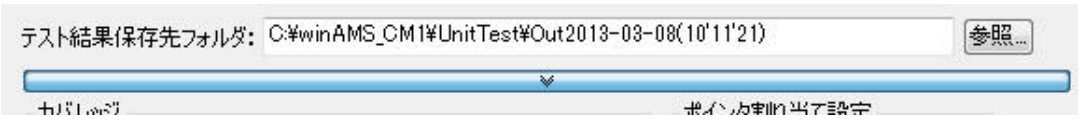


テスト結果(出力ファイル)は、モジュールテスト用 CSV ファイル(入力ファイル)と同じフォーマットになっています。3 つの引数に与えた入力データは、そのままコピーされて出力されます。構造体のメンバー gb_result.data、gb_result.ret_code に出力されたデータが実行結果です。入力データには、期待値を設定していないので、7 列目(G 列)の判定欄は、「NO Check」と表示されます。(期待値を設定すれば、その合否に合わせて、OK/NG が出力されます。)

入力データがそのままコピーされます　実行結果　期待値は設定していないので「チェックしていない」の意味　実行予測時間(目安)設定クロック周波数で換算

	A	B	C	D	E	F	G	H
1	mod	func1	func1 単体テスト	3	2			
2	#COMMENT	@enable	@mode	@input	gb_result.data	gb_result.ret_code		
3		0	0	10	0	0	NO Check	0.0003ms
4		1	0	10	10	1	NO Check	0.00039ms
5		1	1	10	100	1	NO Check	0.00044ms
6		1	2	10	1000	1	NO Check	0.00046ms
7		1	3	10	1000	1	NO Check	0.00052ms
8								

このテスト結果 CSV ファイルの保存先は、「テスト設定」にある「テスト結果保存先フォルダ」で指定することができます。初期状態では、プロジェクト作成時のタイムスタンプが付いたフォルダ名が適用されます。



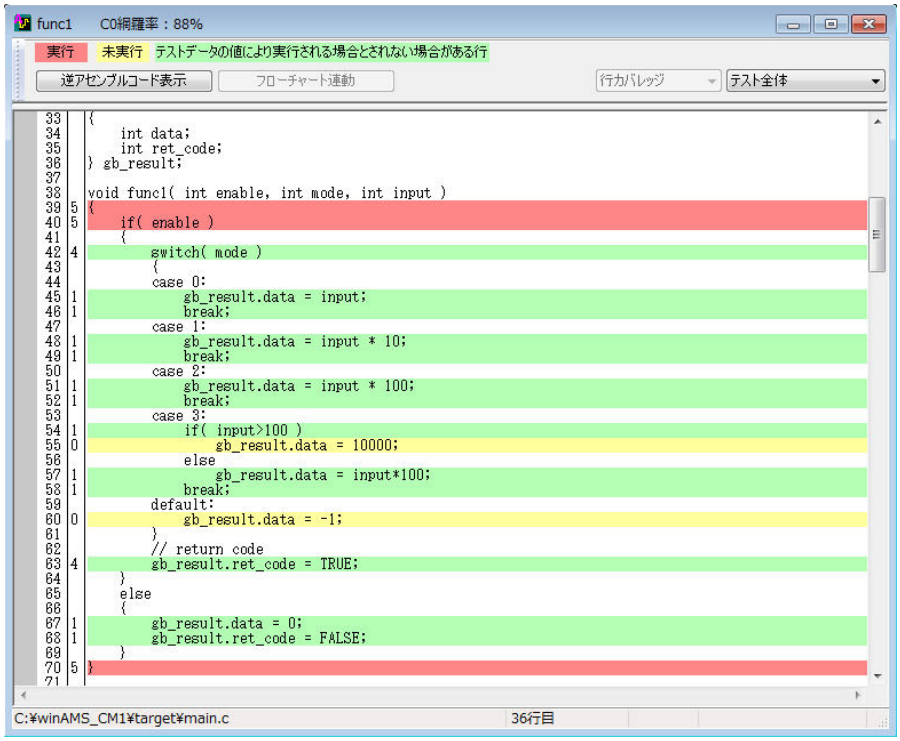
func1()のカバレッジ結果を確認

次に、C0 カバレッジの結果を確認します。テスト結果 CSV ファイルで確認した出力データもカバレッジ結果で参照しますので、EXCEL で開いた CSV ファイルを閉じる必要があります。

1. EXCEL で開いている出力 CSV ファイル(func1_data.csv)を閉じます。
2. 左側の「カバレッジ」ボタンを押します。
3. C0 網羅率 「88%」が確認できます。

テスト関数名	C0
func1	88%

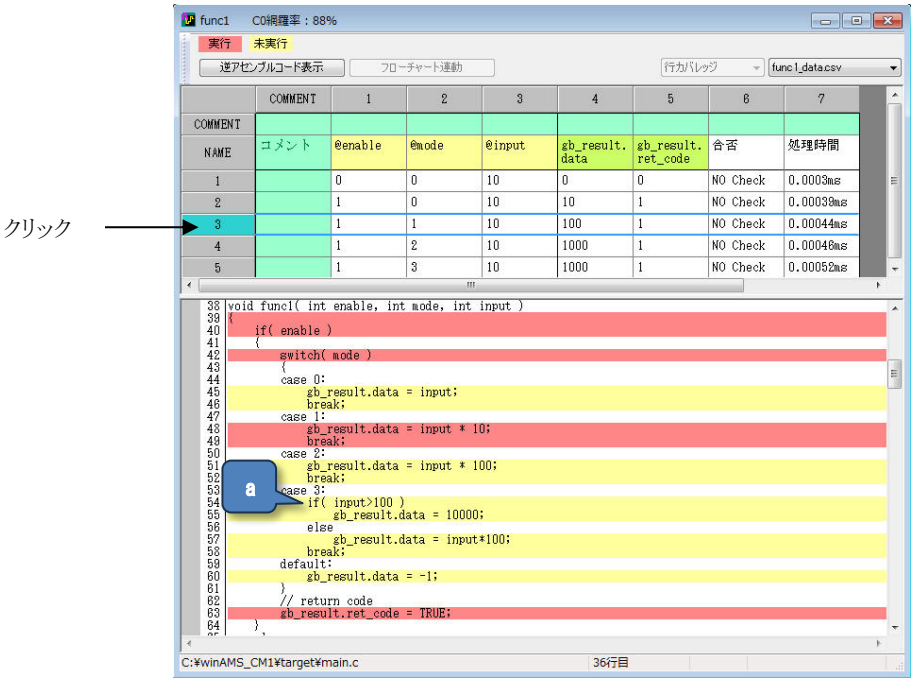
4. リストの「func1」をダブルクリックして カバレッジビューを表示します。



これが func1()の C0 カバレッジ結果です。ウインドウ上部にある凡例の通り、黄色の行が未実行(与えたテストデータでは、テストされなかった)であることを示しています。また、ソース行のすぐ左の数字は、各ソース行の実行回数です。

次に、テストデータを表示して、カバレッジ結果との対応を確認してみます。

5. カバレッジビューの右上にある「テスト全体」から、「func1_data.csv」を選択します。
6. カバレッジビューの上半分に、テスト出力データ(CSV)が表示されます。
   (表示されない場合は、上下のウインドウの境界線を操作して、ウインドウを表示して下さい。)

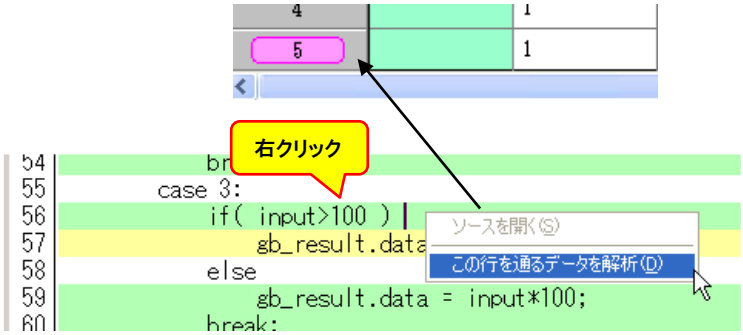


テストデータの番号をクリックしてみてください。クリックしたデータで実行されるソースコード上の実行パスが、赤く塗られて表示されます。

番号以外の箇所（COMMENT や NAME）をクリックすると、テストデータ全体によるカバレッジ結果表示に戻ります。

さらに、このカバレッジビューは、指定したソース行を実行するデータを見つけることもできます。

7. 番号以外の箇所（NAME など）をクリックして、全体のカバレッジ表示に戻します。
8. 上図「a」の if(input > 100)の行を「右クリック」します。
9. 「この行を通るデータを解析」を選択します。
10. 5 番目のデータに赤いマークが表示されます。



これは、この if 文を通過しているテストベクターが、5 番目のデータだけであることを示しています。

この直ぐ下の行は黄色で示されているとおり、テスト未実行の行であり、C0 カバレッジを満たすためには、この黄色の行を実行するテストデータを加える必要があります。この場合であれば、5 番目のデータを基にして、変数 input の値だけを 100 より大きい値にしたデータを追加することで、この黄色を実行することができます。

ここで紹介したソースコードからデータを解析する機能は、条件分岐が複雑にネストしている場合などに、末端の条件分岐にある実行文をテストするためのデータ作成に役立ちます。

カバレッジ結果と実行コードの対応を確認

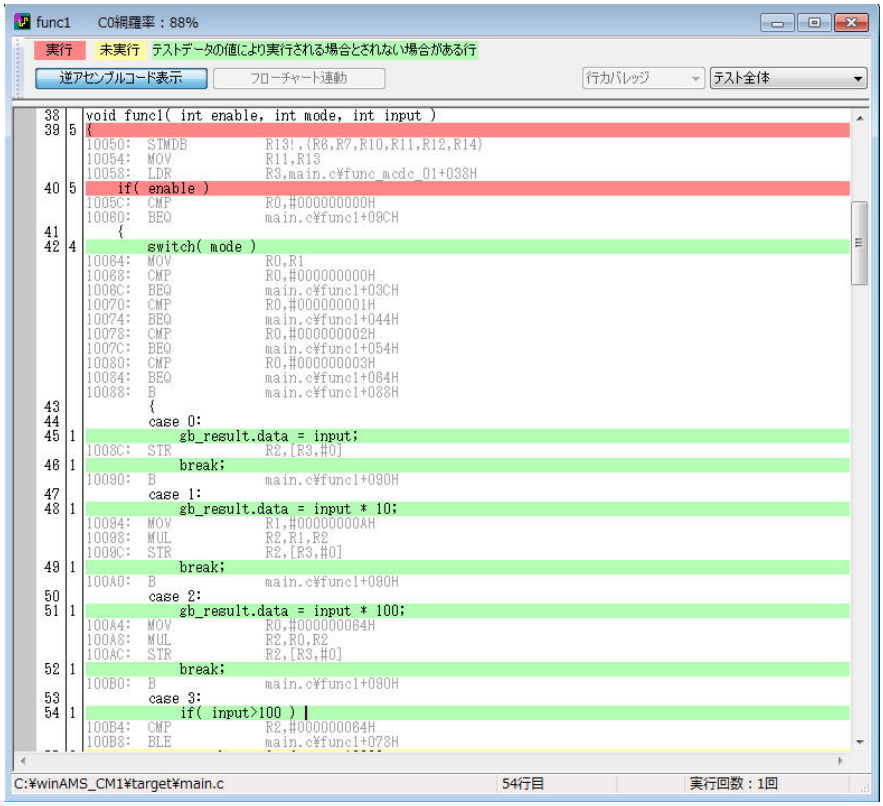
カバレッジマスターwinAMS は、実際のマイコンの命令コードを実行して、単体テスト、カバレッジテストを行うツールです。ここでは、実際の実行コードとカバレッジの対応について説明します。

1. カバレッジビューの上部にある「逆アセンブルコード表示」のボタンを押します。
2. カバレッジビューにマイコン実行コードが表示されます。

この情報は、クロスコンパイラが生成したデバッグ情報を元に表示されます。一般的な ICE デバッガやマイコンシミュレータのソースウインドウに表示される、Cソースとアセンブルソースの「混在表示」と同じものです。

カバレッジマスターwinAMS は、実行コードが実行されると、それに対応する C 言語ソース行をマーキングする仕組みで、カバレッジ記録を行っています。

クロスコンパイラの最適化機能を使用した場合、実行速度を上げるために、クロスコンパイラがマイコンでの実行に有利な様に、命令コードや処理の順番を変更（最適化）してしまうことがあります。このため、ソース行と実行コードの不整合が起こることがあり、カバレッジ結果にも影響を及ぼします。



実際の運用において、カバレッジ結果確認時に、ソース行の色分けが行われず、カバレッジ記録がないソース行に関しては、この「逆アセンブルコード表示」機能を使用して、動作状況を確認するとよいでしょう。

ファイルに出力される報告書を確認する

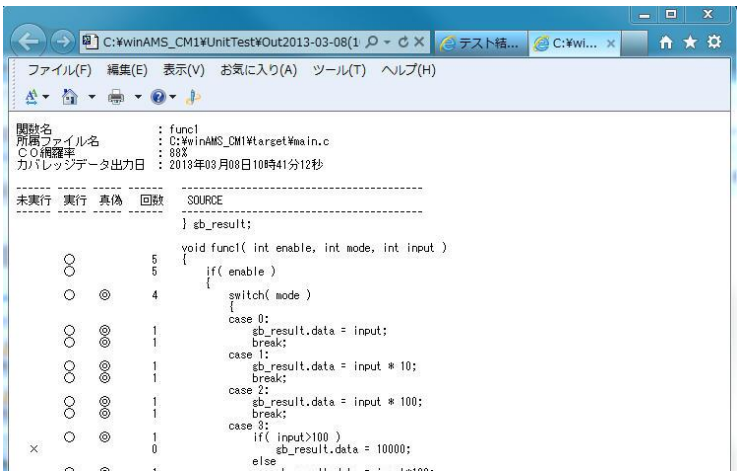
カバレッジ結果は、テキストファイルで出力されています。これを確認してみましょう。

1. 「テスト結果」のボタンで、ビューを切り替えます。
2. 右上にある「報告書を開く」のボタンを押します。
3. WEB ブラウザが開き、HTML のテスト結果報告書が表示されます。

この HTML には、直前に行ったテストの結果が纏められています。この HTML は、保存用のファイルとして使用すると言うよりも、直前のテストの結果を閲覧するための、一時的なファイルとして使用します。
表示内容は、以下の通りです。

テスト全体の情報 → トップの CSV ファイル： テストに使用した入力 CSV へのリンク
テスト結果情報 → CSV ファイル： テスト結果の出力 CSV ファイルへのリンク
カバレッジ情報 ： 対象の関数名とカバレッジ結果のテキストファイルへのリンク

カバレッジ情報のテキストファイルは、以下のようになっています。リンクをクリックして確認して下さい。WEB ブラウザが開きますが、ファイルはプレーンなテキストファイルです。



func1()の再テストを行いカバレッジを 100%にする

実習1の課題は、C0 カバレッジを 100%にすることですので、足りないデータを追加して、再テストを行います。また、期待値を設定して、期待値判定の結果も確認します。

1. 「テスト設定」のボタンで画面を切り替えて、入力 CSV ファイル「func1_data.csv」を開きます。
2. 以下の図に示す 2 回分のテストデータ(赤い実線)を追加します。
3. 期待値(赤の破線)を追加します。(いくつかのデータを 故意に異なった値に設定しておきます。)

下図の赤の破線で囲まれた期待値には、実行結果と異なる値(期待値=99 のセル、2 カ所)が入力してあります。期待値と異なる結果がどの様にレポートされるかを、実行後に確認したいと思います。

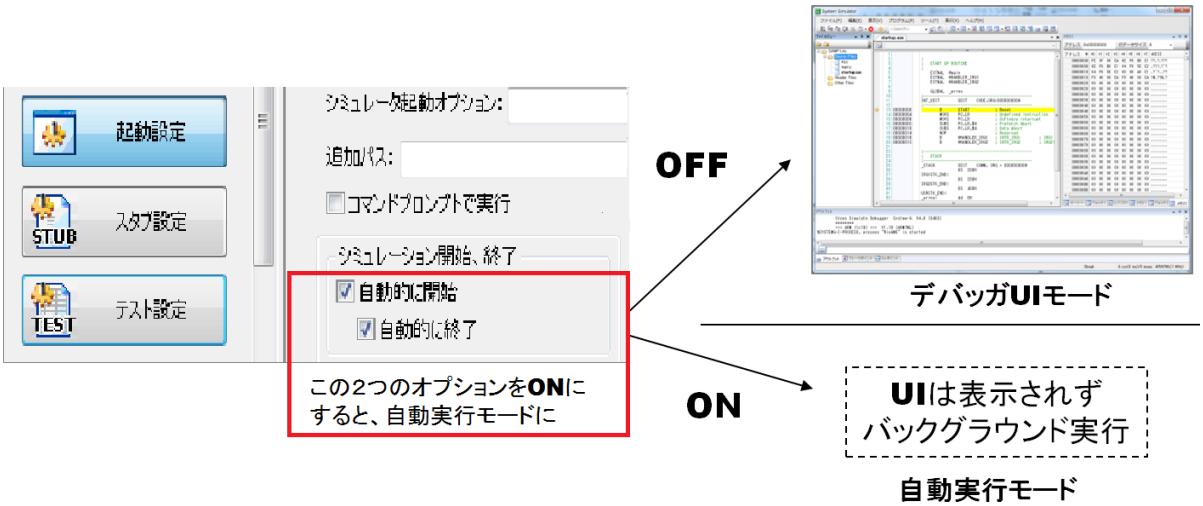
	A	B	C	D	E	F
1	mod	func1	func1 単体テスト	3	2	
2	#COMMENT	@enable	@mode	@input	gb_result.data	gb_result.ret_code
3		0	0	10	0	0
4		1	0	10	10	1
5		1	1	10	99	1
6		1	2	10	1000	1
7		1	3	10	1000	99
8		1	3	200		
9		1	4	10		
10						

[99]のセルは、実行結果と
異なる期待値が入れてあります

4. EXCEL の「ファイル」メニューから「上書き保存」で、CSV ファイルを保存します。
5. EXCEL を閉じます。

では、再テストを行います。前回のテスト実行では、「シミュレータ起動」のボタンを押した後、起動したマイコンシミュレータを手動操作してテスト実行を行いましたが、この部分を自動化します。(自動実行モード)

6. 左のボタンから「起動設定」を押して、画面を切り替えます。
7. 「シミュレーション開始、終了」の項目の「自動的に開始」「自動的に終了」のチェックを2つとも有効にします。



このオプションで、自動実行とデバッガ UI を使用したモードとの切換が行われます。このオプションをオンにすると、カバレッジマスターwinAMS から「シミュレータ起動」のボタンを押すだけで、マイコンシミュレータを操作することなく、テスト実行が可能になります。

テストの再実行を行います。

8. SSTManager の「シミュレータ起動」のボタンを押します。
9. テストが自動実行され、「テスト結果」ビューに切り替わります。

ソースコードトレースを行わないため、非常に高速にテスト実行が終了します。テストが終了したら、入出力結果とカバレッジを参照して下さい。

10. 「テスト結果」ビューで、「func1_data.csv」をダブルクリックして開きます、

下図のように、期待値に異なる値を設定した箇所には、「実行結果？（期待値）」の書式で出力されます。また、期待値判定のセルには、「OK/NG」が表示されます。

	A	B	C	D	E	F	G	H	I
1	mod	func1	func1 単体テスト	3	2				CPP
2	#COMMENT	@enable	@mode	@input	gb_result.data	gb_result.ret_code			
3		0	0	10	0	0	OK	0.0003ms	
4		1	0	10	10	1	OK	0.00039ms	
5		1	1	10	100?(99)	1	NG	0.00044ms	
6		1	2	10	1000	1	OK	0.00046ms	
7		1	3	10	1000	1?(99)	NG	0.00052ms	
8		1	3	200	10000	1	NO Check	0.00053ms	
9		1	4	10	-1	1	NO Check	0.00044ms	

カバレッジが 100%になっていることも確認して下さい。

（参考）自動実行モード使用中にシミュレータウインドウを表示する方法

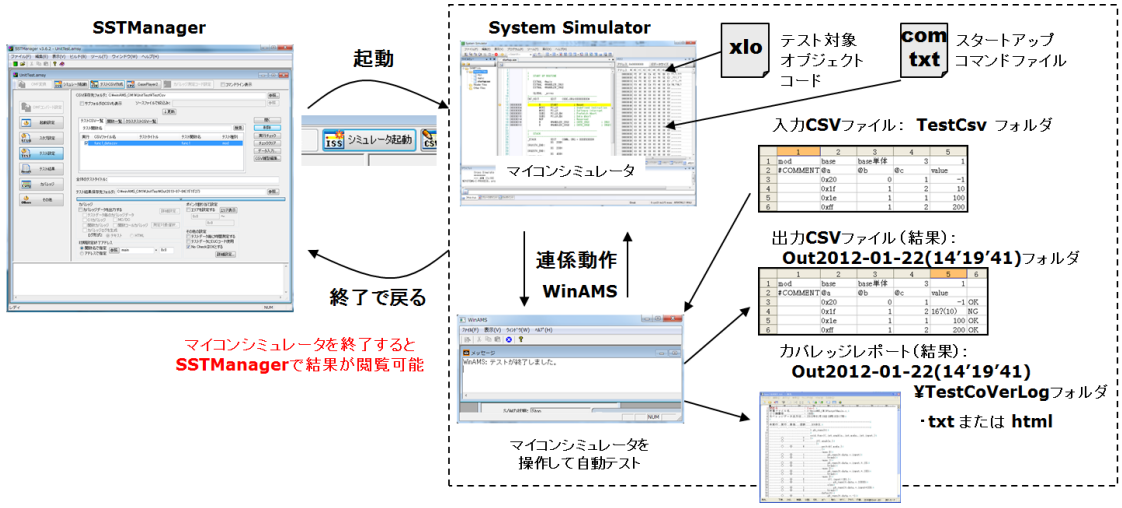
前頁で説明した、「自動テストを行う」の項目の「シミュレーション開始、終了」のチェックを有効にして、テスト実行を自動化した場合、マイコンシミュレータの UI は表示されません。データの設定ミスや、テスト対象関数そのものの不具合で、テスト実行中にシミュレータが止まってしまった場合は、バックグラウンドで動作しているマイコンシミュレータのウインドウ（ウインドウ名：Lix）を以下の方法で表示し、シミュレータを終了してください。

1. Windows タスクバーにある「L」のアイコンを右クリックする
2. ポップアップするメニューから「表示」を選択する
3. Lix の「ファイル」メニューから「終了」を選択する

（参考）単体テスト時のツール起動構成と結果出力ファイル

実習1で行った単体テスト実行時に、PC で起動するウインドウの構成をまとめておきます。
単体テストを管理するツール「SSTManager」から「シミュレータ起動」ボタンで単体テストを起動した際には、以下の様な順序でツールが動作します。

1. 「SSTManager」から「シミュレータ起動」ボタンで単体テストを起動する
2. マイコンシミュレータ（ウインドウ名：SystemSimulator）が起動する
3. 単体テストツール（ウインドウ名：WinAMS）が起動する
4. マイコンシミュレータが実行を開始する
   デバッガ UI モード時：「実行」メニューから「実行開始」を選択
   自動実行モード時：操作不要（自動的に開始）
5. 入力 CSV ファイルからテストケースをロードし、対象関数を実行
6. 実行後に出力 CSV ファイルへ結果を書き出し
7. カバレッジレポートファイル（txt/html）を作成
8. マイコンシミュレータを終了する
   デバッガ UI モード時：「ファイル」メニューから「終了」を選択
   自動実行モード時：操作不要（自動的に終了）
9. 「SSTManager」でテスト結果が閲覧可能になる



テストに使用したファイルは、下記のフォルダに保存されます。

- 入力 CSV ファイル： ［テストプロジェクトフォルダ］¥TestCSV
- 出力 CSV ファイル： ［テストプロジェクトフォルダ］¥Out[プロジェクト作成日付][プロジェクト作成時間]
- カバレッジレポート： ［テストプロジェクトフォルダ］¥Out[プロジェクト作成日付][プロジェクト作成時間]¥TestCoverLog

実習1のまとめ

以上が、カバレッジマスターwinAMS の一連のテストフローです。テストデータを設計して、テスト入力 CSV ファイルを作成すれば、単体テスト作業やカバレッジ記録は自動化できることが体験頂けたと思います。

実習2： ポインタ変数を持つ関数の単体テスト

次の実習では、ポインタ変数をもつ関数のテスト方法について体験します。

例えば、引数に構造体のポインタが渡される関数をテストする場合、関数テスト時には、その構造体の実体にデータを格納して、そのアドレスを関数に渡さなければなりません。
ポインタはアドレスであり、その実体は対象関数の管理外に置かれることがほとんどです。ポインタにアプリケーション全体で共用するグローバルのデータテーブル（配列）のアドレスが指定される場合もありますが、単体テストで関数の機能をテストする場合には、データテーブルを直接用いず、関数にデータを直接入力する方が、テスト作業の効率が高いこともあります。

この実習では、カバレッジマスターwinAMS の「実体自動割付機能」を使用して、関数で使用されるポインタの実体を、単体テストのために、アプリケーションのデータとは別に割付を行い、ここにCSVファイルからデータを入出力してテストを行う方法について学習します。

サンプルソース func2()を確認

実習にはサンプルソース main.c の func2()を使用します。この関数にはポインタが多用されています。まず、入出力変数を確認します。

```
// 実習2） ポインタ変数引数の関数サンプル
//          C0カバレッジを100%にする

// global valiables
char *gb_port1 ;  // i/oポート入力
int  *gb_data_out; // 結果

void func2( int mode, int *data_in )
{
    if( *gb_port1 & 0x00000001 ) // 最下位ビットが1のとき
    {
        switch( mode)
        {
            case 0:
                *gb_data_out= *data_in ;
                break;
            case 1:
                *gb_data_out= *data_in * 10;
                break;
            case 2:
                *gb_data_out= *data_in * 100;
                break;
            case 3:
                if( *data_in > 100 )
                    *gb_data_out= 10000;
                else
                    *gb_data_out= *data_in *100;
                    break;
                default:
                    *gb_data_out= -1;
            }
        }
        else
        {
            *gb_data_out= 0;
        }
}
```

関数の中では以下の 3 つの変数が参照されています。

char *gb_port1（グローバル　ポインタ変数），
int mode（引数），　int *data_in（引数）

このうち、2 つはポインタ変数ですが、その実体は配列ではなく、1 つの変数のポインタです。
また、関数内では、以下の変数が変更され、関数の処理結果になっています。

int　*gb_data_out;（グローバル　ポインタ変数）

では、この入出力変数をテスト要件として、単体テスト CSV ファイルを作成します。

ポインタ変数にはアドレス設定とテストケース設定が必要

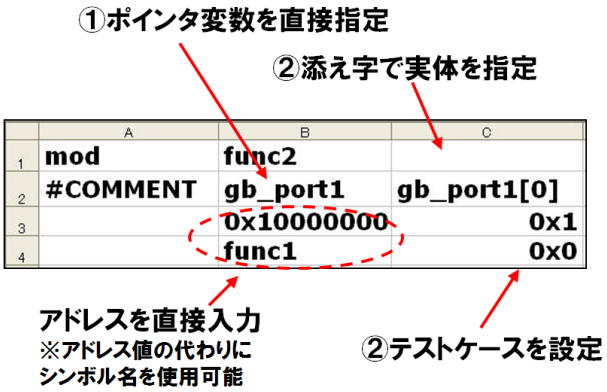
例として、グローバル変数「char *gb_port1」について説明します。
ポインタ変数を含む関数をテストするためには、ポインタ変数に対して、次の 2 つを設定する必要があります。

① ポインタ変数 gb_port1 のアドレスを設定する
② ポインタ変数の指す実体にテストケースを入れる

func2()がアプリケーションの中に組み込まれて使用される際は、ポインタの値(アドレス)は前もって設定されてからfunc2()がコールされますが、単体テストでは func2()を単独で動作させる必要があるため、ユーザー自身がこれらのポインタの値を、関数実行前に設定しておく必要があります。
カバレッジマスターwinAMS では、CSV ファイルの書式によって、ポインタにアドレスを与える方法が2つあります。

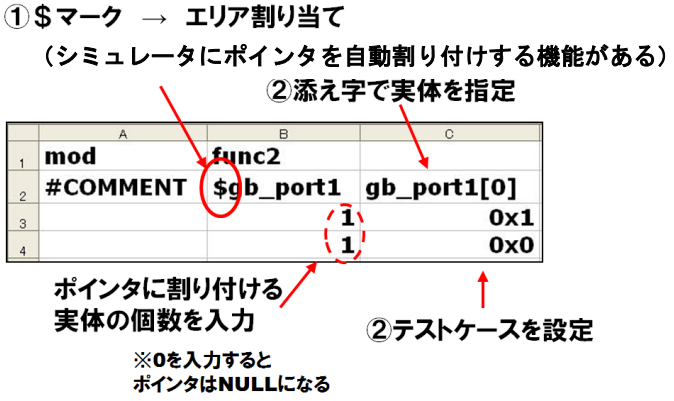
■ ポインタにアドレスを直接与える方法

ポインタにアドレスを直接与えるには、CSV ファイルの入力変数に、ポインタ変数名をそのまま指定します。例えば、グローバル変数「char *gb_port1」の場合であれば、「gb_port1」を指定します。ポインタ変数の中身はアドレス値ですので、テストデータにはアドレス値(0x で始まる 16 進数)を指定します。
ただし、この方法は、テスト設計時に、ポインタに割り付けるアドレスを、使用するマイコンのメモリマップ上で適切な領域に、ユーザー自身が管理し決定しなければなりません。マイコンの型番変更などの際には、割付可能なアドレスに再調整する必要があります。
ポインタ値を決定したら、gb_port1 の実体を示す「gb_port1[0]」を CSV ファイルの入力に追加します。ここには、ポインタが指すメモリに入力するテストデータを設定します。



■ ポインタのアドレスを自動割り付け機能で決定する方法

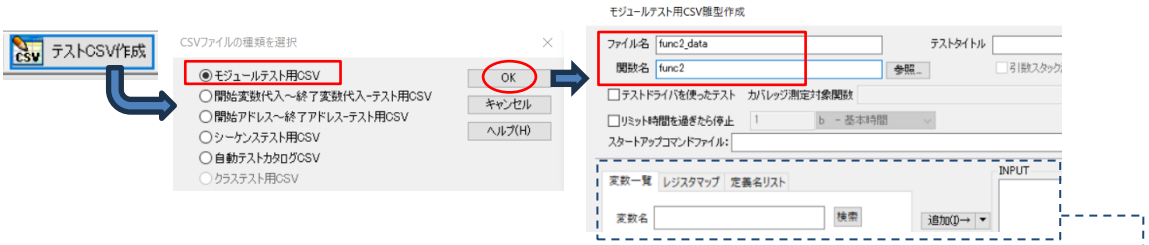
カバレッジマスターwinAMS には、ポインタに設定するアドレスを自動的に割り付ける機能があります。この方法を使用する場合には、ユーザー自身がアドレス値を管理する必要はありません。CSV ファイルには具体的なアドレスが記載されないため、マイコンの型番変更などの際に、割付可能なアドレスに再調整する必要がなくなります。
自動割付を行うためには、CSV ファイルの入力項目に、ポインタ名の頭に「$」を付けて追加します。例えば、グローバル変数「char *gb_port1」の場合であれば、「$gb_port1」を指定します。この「$」の記号により、シミュレータはポインタに対してアドレスを自動割り付けします。テストデータには、このポインタに割り付けたい実体の個数を記載します。変数の型のサイズに合わせて、設定した個数分のメモリエリアが割り付けられます。ポインタが配列の先頭アドレスの場合には、配列に必要な要素の個数を設定します。個数に「0」を指定すると、そのポインタは NULL になります。
ポインタの自動割付を行ったら、gb_port1 の実体を示す「gb_port1[0]」を CSV ファイルの入力に追加します。ここには、ポインタが指すメモリに入力するテストデータを設定します。



ポインタ自動割り付け機能を利用した CSV ファイルを作成する

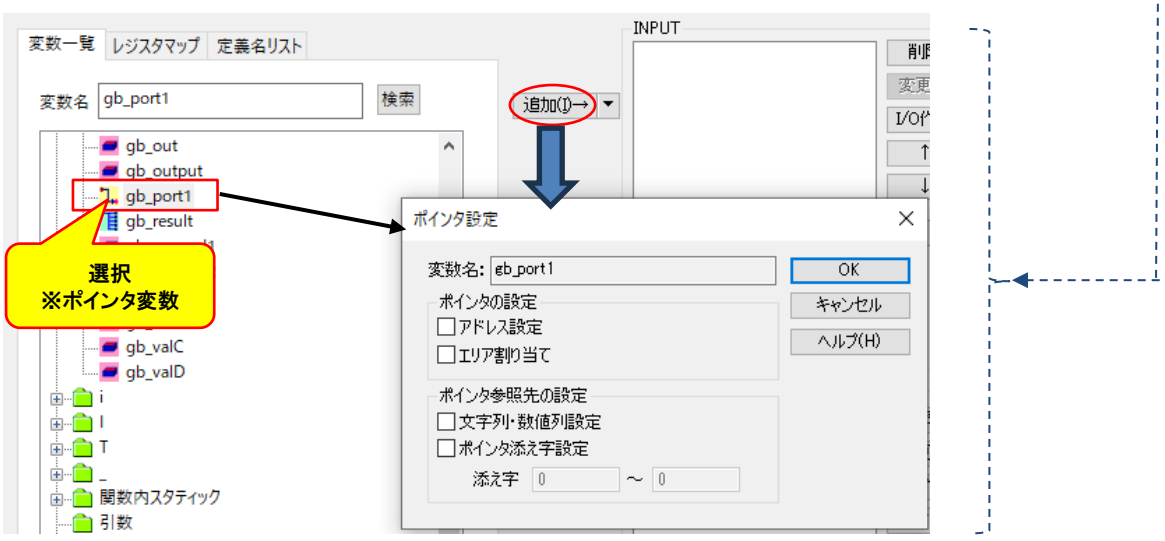
では、この指定方法で、CSV ファイルを実際に作って見ましょう。実習1と同様に、func2()テスト用の CSV ファイルを新規に作成します。

1. SSTManager 上部の「テスト CSV 作成」ボタンを押します。
2. 「モジュールテスト用 CSV」を選択して「OK」を押します。
3. 「モジュールテスト用 CSV 雛形作成」ウインドウで、ファイル名に「func2_data」指定します。
4. 同様に、関数名に「func2」を指定します。



最初に、「char *gb_port1」をテスト条件に設定します。この変数はポインタであり、テストのために実体を割り付ける必要があります。まず、前述の「$」付きの入力項目を作成します。

5. 変数一覧で「g」のフォルダを開き、gb_port1 を選択します。
6. INPUT 側の「追加」ボタンを押します。
   ※「変数一覧」でポインタ変数を選択した場合は、下の「ポインタ設定」ダイアログが開きます。



「ポインタ設定」ダイアログで作成される内容は、以下の通りです。

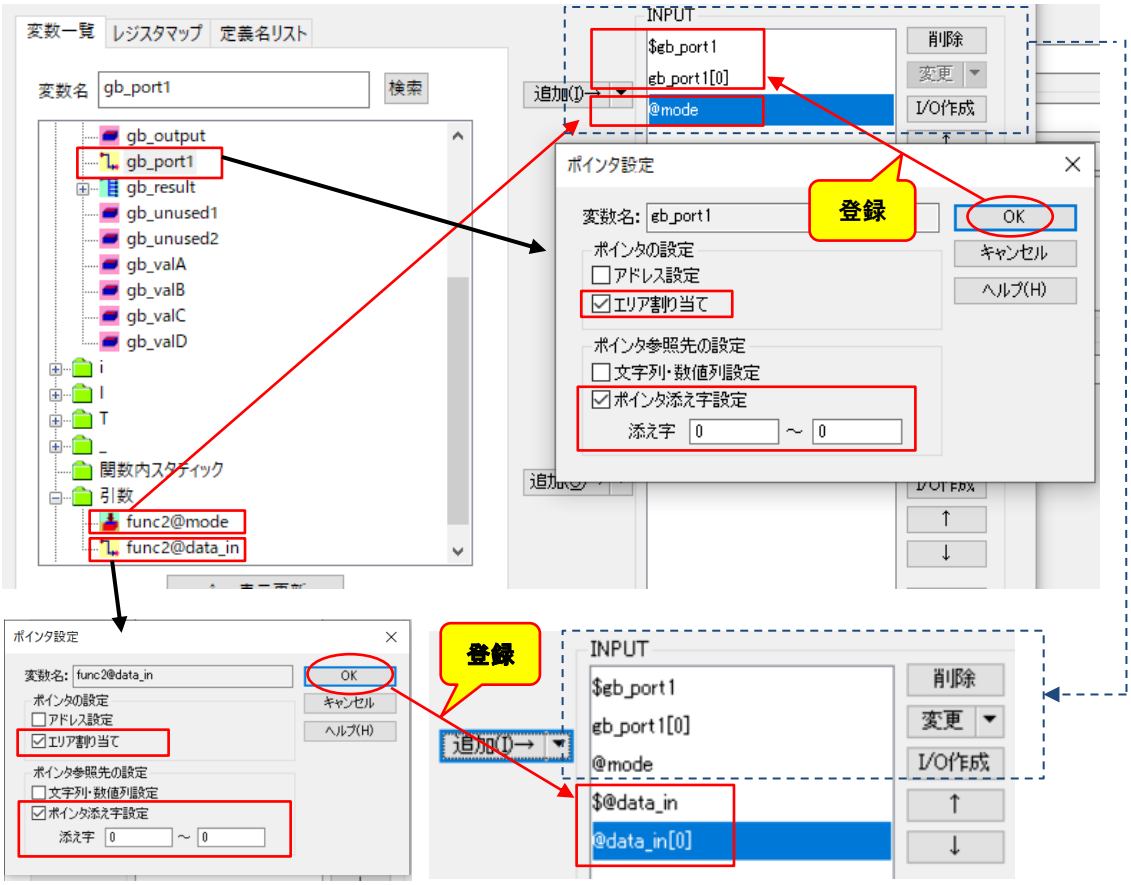
●アドレス設定： ポインタ変数にアドレスを指定する場合に使用します。「gb_port1」が作成されます。
●エリア割り当て： ポインタの実体をテスト用に自動割付する場合に使用します。「$gb_port1」が作成されます。
●文字列・数値列設定： ポインタ変数が配列の場合に、1つのセルに配列データをまとめて指定する場合に使用します。「*gb_port1」が作成されます。この課題では違いますが、もしも gb_port1 が文字列配列である場合には、このオプションを使用することで、入力データに「'abcdefg'」の形式で文字列を指定することができます。
●ポインタ添え字設定： ポインタの実体を指定する場合に使用します。「gb_port1[0]」が作成されます。

では、gb_port1 の実体の自動割り付けを行うための入力項目と、gb_port1 の実体にデータを入力するための入力項目を作成します。

7. 「エリア割り当て」を選択します。（$gb_port1 を追加）
8. 「ポインタ添え字設定」を選択します。
9. 添え字は、「0～0」のままにします。（gb_port1[0]を追加）
10. 「OK」を押します。

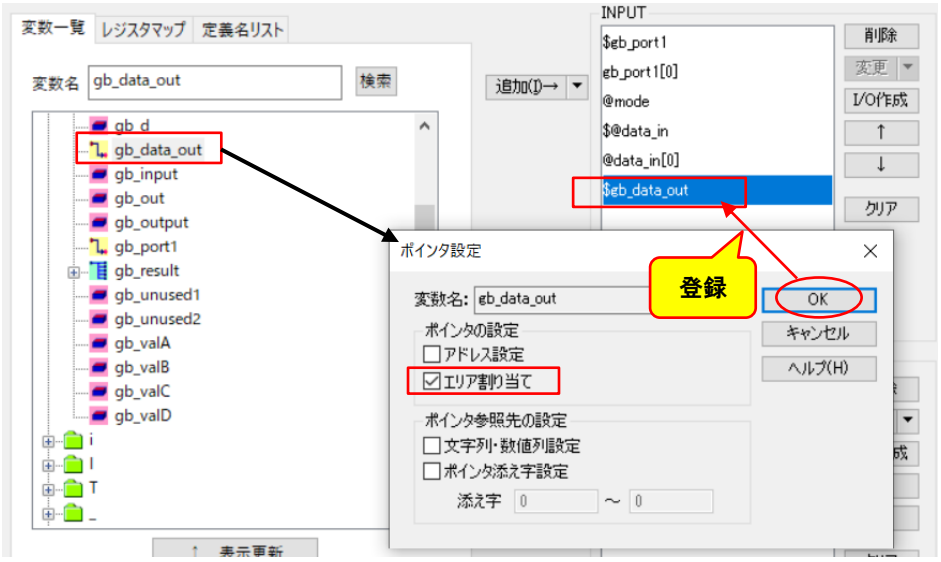
これにより、INPUT の項目に、「$gb_port1」と「gb_port1[0]」が登録されます。同様に、他の入力条件を INPUT に追加します。

@mode（引数）： int 型変数であるため、追加ボタンで INPUT に追加します。
@data_in（引数）： ポインタ変数であるため、「エリア割り当て」オプションで「$@data_in」を、また「ポインタ添え字設定（0～0）」で「@data_in[0]」を追加します



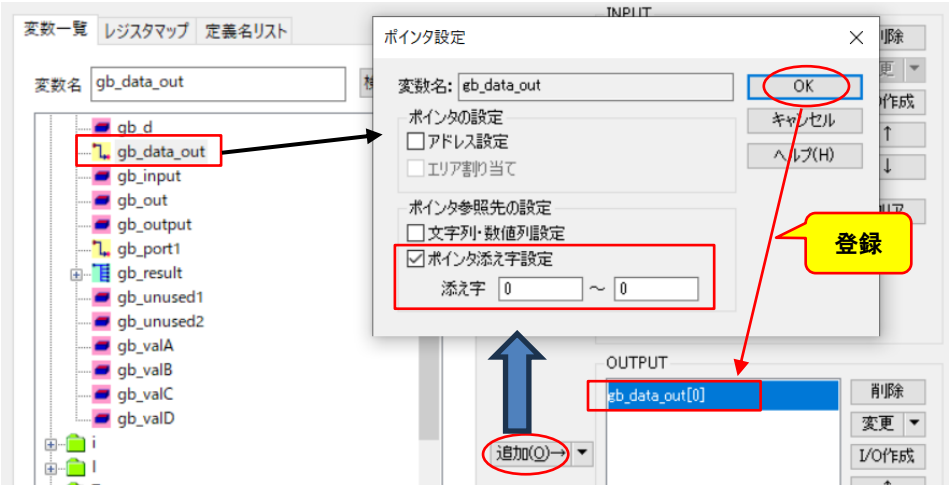
次に、評価対象の変数「int *gb_data_out」を指定します。この変数もポインタ変数であるため、エリア設定で実体の割り付けを行いますが、割り付け項目は実体の個数を指定する「入力条件」であるため、INPUT 側にエリア設定の項目を追加します。

11. 変数一覧の「g」のフォルダから、gb_data_out を選択します。
12. INPUT 側の「追加」ボタンを押します。
13. 「エリア割り当て」を選択して、「OK」を押します。

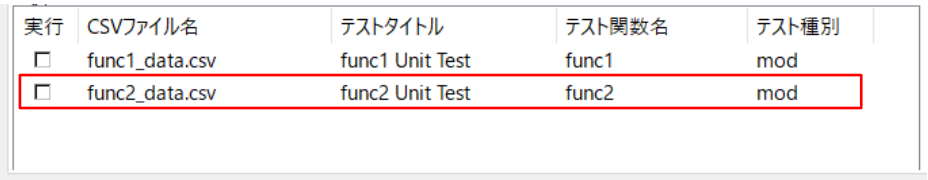


さらに、gb_data_out の実体を評価対象とするための指定を行います。これは、期待値を設定する評価変数の指定であるため、OUTPUT 側に加えます。

14. 変数一覧の「g」のフォルダから、もう一度 gb_data_out を選択します。
    ※この際、変数選択ボックスの上部の「カレント変数名」が「gb_data_out」であることを確認します。
15. OUTPUT 側の「追加」ボタンを押します。
16. 「ポインタ参照先の設定」の選択肢から、「ポインタ添え字設定」を選択します。
17. 添え字は、「0」～「0」のままにします。
18. 「OK」を押します。



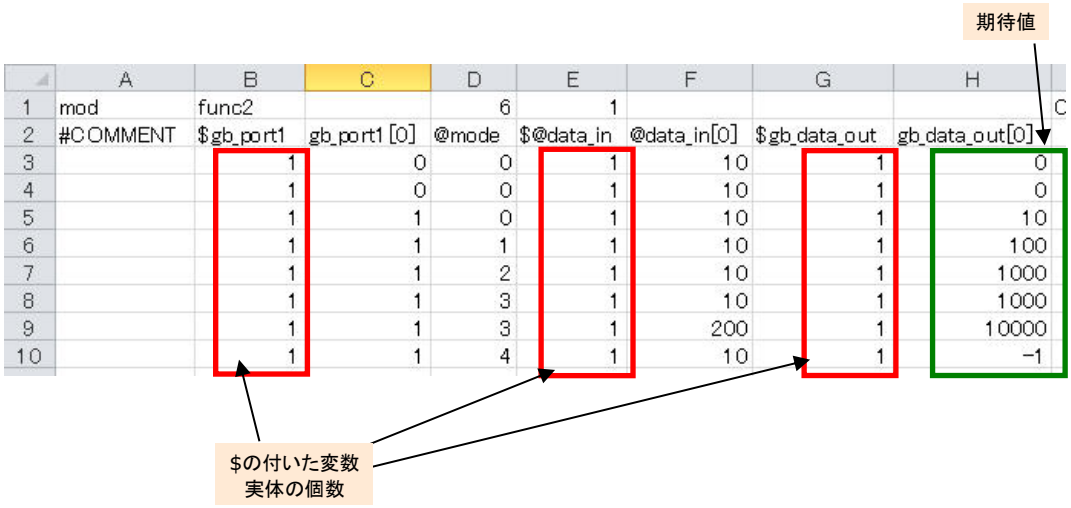
これで、入出力条件の指定が出来ました。「モジュールテスト用 CSV 雛形作成」ウインドウの「OK」ボタンを押すと、CSV ファイルが作成されます。



実行	CSVファイル名	テストタイトル	テスト関数名	テスト種別
☐	func1_data.csv	func1 Unit Test	func1	mod
☐	func2_data.csv	func2 Unit Test	func2	mod

19. SSTManager の「テスト設定」ボタンでビューを切り替えます。
20. 「func2_data.csv」の行をダブルクリックして EXCEL を開きます。

以下のようにテストデータを設定します。func2 のポインタ変数の実体は、全て配列ではなく 1 個の変数であるため、$で始まるエリア割り当て設定のデータ欄には、「1」が入ることに注意して下さい。



	A	B	C	D	E	F	G	H
1	mod	func2		6	1			
2	#COMMENT	$gb_port1	gb_port1 [0]	@mode	$@data_in	@data_in[0]	$gb_data_out	gb_data_out[0]
3		1	0	0	1	10	1	0
4		1	0	0	1	10	1	0
5		1	1	0	1	10	1	10
6		1	1	1	1	10	1	100
7		1	1	2	1	10	1	1000
8		1	1	3	1	10	1	1000
9		1	1	3	1	200	1	10000
10		1	1	4	1	10	1	-1

緑色の枠で囲ったデータは、評価対象 gb_data_out[0]の期待値です。全て正しい期待値になっています。

ポインタ自動割付機能を利用して 単体テスト実行を行う

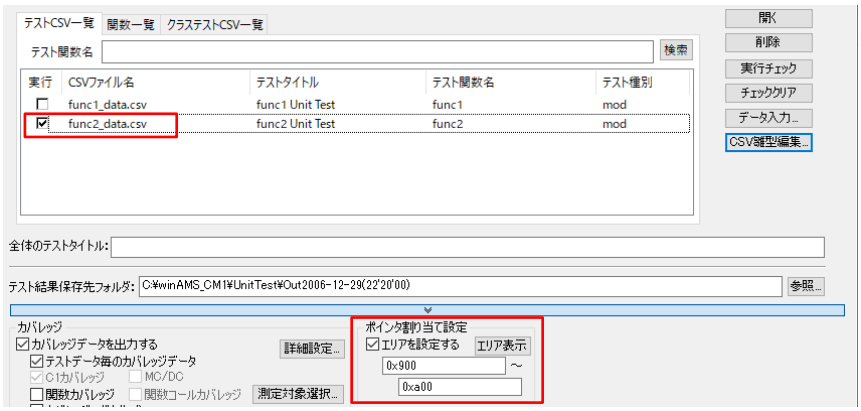
ポインタの実体自動割付機能を使用して、func2()の単体テストを実行します。まず、実行のための設定を追加します。

1. 「テスト CSV 一覧」で「func2_data.csv」の左のチェックボックスを ON にします。
   ※「func1_data.csv」のチェックボックスは OFF にしてください。
2. 「ポインタ割り当て設定」欄の「エリアを設定する」を ON にします。

自動割り付けを行うアドレスを指定します。カバレッジマスターwinAMS は、マイコンシミュレータを使用して実行を行うため、割り付けを行うメモリエリアも実際のマイコンのアドレスエリアを使います。自動割り付けに使用するエリアは、使用するマイコンの未使用領域(コード／変数／スタックやペリフェラル等がアサインされていない領域)を指定してください。

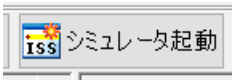
3. 「エリアを設定する」に、未使用領域のアドレス「先頭アドレス(0x…)～終了アドレス(0x…)」を指定します。

これで、指定したエリアに、「$」で割付を行うポインタ変数の実体が作られます。ただし、このエリア内のどの番地に割り付けられるかは、テスト実行時にツールが自動的に決定します。

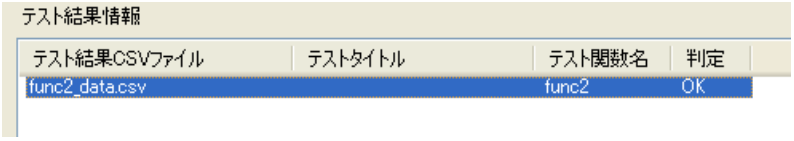


では、テストを実行して下さい。手順は、実習1と同様です。

4. SSTManager の上部の「シミュレータ起動」ボタンを押します。



テスト実行が行われ、終了後に「テスト結果」のビューに切り替わります。入出力テスト結果に「OK」が表示されます。



5. 「func2_data.csv」をダブルクリックして、ファイル出力内容を確認します。

	A	B	C	D	E	F	G	H	I	J
1	mod	func2		6	1				CPP	
2	#COMMENT	$gb_port1	gb_port1[0]	@mode	$@data_in	@data_in[0]	$gb_data_out	gb_data_out[0]		
3		1	0	0	1	10	1	0	OK	0.0004ms
4		1	0	0	1	10	1	0	OK	0.0004ms
5		1	1	0	1	10	1	10	OK	0.00051ms
6		1	1	1	1	10	1	100	OK	0.00056ms
7		1	1	2	1	10	1	1000	OK	0.00058ms
8		1	1	3	1	10	1	1000	OK	0.00064ms
9		1	1	3	1	200	1	10000	OK	0.00065ms
10		1	1	4	1	10	1	-1	OK	0.00053ms

今回は、正しい期待値を入力しましたので、9 列目（I 列）の判定が、全て「OK」になっていることが確認できます。

以上が、ポインタ変数を含む関数に対する単体テスト方法の実習でした。

参考：構造体ポインタのメンバー変数指定方法

構造体がポインタの場合に、その構造体のメンバーを入出力変数に指定するためには、まず、構造体の実体を作成する必要があります。構造体ポインタは、そのままではポインタ変数として変数選択のツリーに表示されていますが、添え字[0]を指定して実体を作成すると、その下にメンバーが表示されます。

作成された構造体（添え字[0]）のメンバー変数は、通常の変数と同様に、CSV ファイルの入力変数、出力変数に登録できます。

参考：文字列・数値列データの CSV ファイル指定について

ポインタや配列の変数に対して、文字列・数値列のデータを1つのセルに一括指定する方法について紹介します。文字列のデータは、変数がポインタ宣言されている場合は「*」、配列で宣言されている場合は「&」を頭に付けた指定で、入力項目を作ります。実習2で学習した「ポインタ設定」の選択で、「文字列・数値列指定」を選択することで作成できます。

文字列は以下のように、’abcdefg’の様にシングルクォーテーションで囲んで指定します。

```
/*
        文字列データ
*/
char outStr[64];

void StringTest( int limit )
{
        outStr[limit] = '¥0';
}
```

	A	B	C	D	E
1	mod	StringTest		2	1
2	#COMMENT	&outStr	@limit	&outStr	
3		'GAIO TECHNOLOGY'	4	'GAIO'	NO Check
4		'GAIO TECHNOLOGY'	6	'GAIO T'	NO Check
5		'GAIO TECHNOLOGY'	8	'GAIO TEC'	NO Check
6		'GAIO TECHNOLOGY'	10	'GAIO TECHN'	NO Check

StingTest

数値列の場合は、数値の区切り目を「|」で示して、以下のように指定します。評価対象の欄には、出力したい数値列の個数分だけ（実際の個数－1）「|」を入れた欄を作ることで、出力数値列の個数を指定します。

```
int ArrayTable[10];

void ArrayTest( int index )
{
        int i;
        if( index>10 ) index=10;
        for( i=0; i<index; i++ )
        {
                ArrayTable[i]
= 0;
        }
```

	A	B	C	D	E
1	mod	ArrayTest		2	1
2	#COMMENT	&ArrayTable	@index	&ArrayTable	
3		1|2|3|4|5|6|7|8|9	2	|||||||||	
4		1|2|3|4|5|6|7|8|10	4	|||||||||	
5		1|2|3|4|5|6|7|8|11	6	|||||||||	
6		1|2|3|4|5|6|7|8|12	8	|||||||||	

ArrayTest

D
2
&ArrayTable
0|0|3|4|5|6|7|8|9
0|0|0|0|5|6|7|8|10
0|0|0|0|0|0|7|8|11
0|0|0|0|0|0|0|0|12

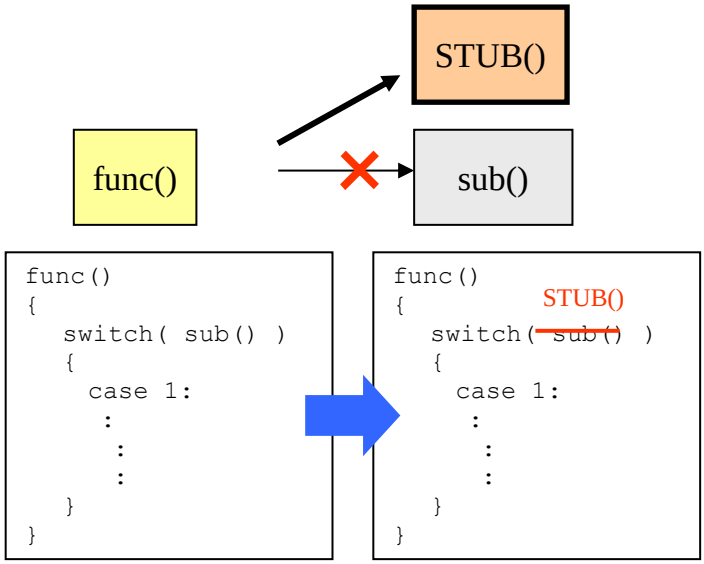
実習3： スタブ機能を使用して 呼び出し関数を置換

次は、関数コールを含む関数の単体テスト方法について実習します。

スタブ関数とは

通常、関数には他の関数を呼び出す処理が含まれています。例えば、関数 func()からサブルーチン sub()が呼ばれる場合を例にとります。

単体テストとは、関数毎の入出力テストにより、関数の網羅テストを行うものです。func()から sub()が呼ばれている場合であっても、その呼び出し関係には依存せず、func()、sub()各々をそれ単体で評価するのが一般的な単体テストの考え方です。



上の例のように、呼び出された sub()の戻り値で分岐が行われている場合には、func()の網羅テストを行うために、sub()が適切な戻り値を返す必要があり、これを行おうとすると sub()に与える条件が複雑になります。また、呼び出しをそのまま使ったテストを行っていると、この 2 つは常に依存しており、sub()が改変されたら必ず func()も再テストする必要が出てきます。

このため、通常は、func()のテスト時には sub()を使用せず、sub()の代用関数となるスタブ関数（STUB()）を作成して使用します。この場合は、戻り値を自由に返すことが出来ることを要件にして STUB()を設計すれば、func()のテストが容易に行えます。

カバレッジマスターwinAMS のスタブ機能

カバレッジマスターwinAMS には、このスタブ関数を作成、管理、実行する機能が搭載されています。

・スタブ関数を作成し、元の関数との対応を管理する機能
・テスト時に、スタブ関数に切り替え（置換）する機能

2 つめの置換機能は、カバレッジマスターwinAMS 特徴的な機能です。関数呼出し部分のソース書き換えを行わずに、スタブ関数に実行を切り替えることができます。上の例であれば、sub()をコールしたままのソースであっても、カバレッジマスターwinAMS の置換機能を使うと、テスト時に STUB()に切り替えて実行することができます。

カバレッジマスターwinAMS で作成したスタブ関数は、指定した1つのソースファイルにまとめられます。スタブ関数もマイコンシミュレータで実行をおこないますので、このスタブ関数は、クロスコンパイラでコンパイル・リンクを行い、実行可能なコードにする必要があります。

では、実習で、スタブ関数を作成・設定する方法について学習します。

評価対象のスタブ対象を確認

実習3では、func3()を評価対象にします。入出力条件は、

入力変数: enable、mode (共に引数)
評価変数: gb_result.data、 gb_result.ret_code (共に外部変数構造体のメンバ)

です。

この関数の途中には、関数呼出し(func3_sub_read_io()) があります。呼び出し関数の戻り値は、一旦ローカル変数 retval に入りますが、これを使って、すぐ下の switch 文で分岐が行われています。func3()を網羅してカバレッジを100%にするためには、この戻り値を switch 文の分岐に合わせて変える必要があります。

今回は、この呼び出し関数 func3_sub_read_io()のスタブ関数を作成して、テスト時に戻り値を自由に変化させるために、入力 CSV ファイルに戻り値が設定できるようにスタブ関数の設計をします。

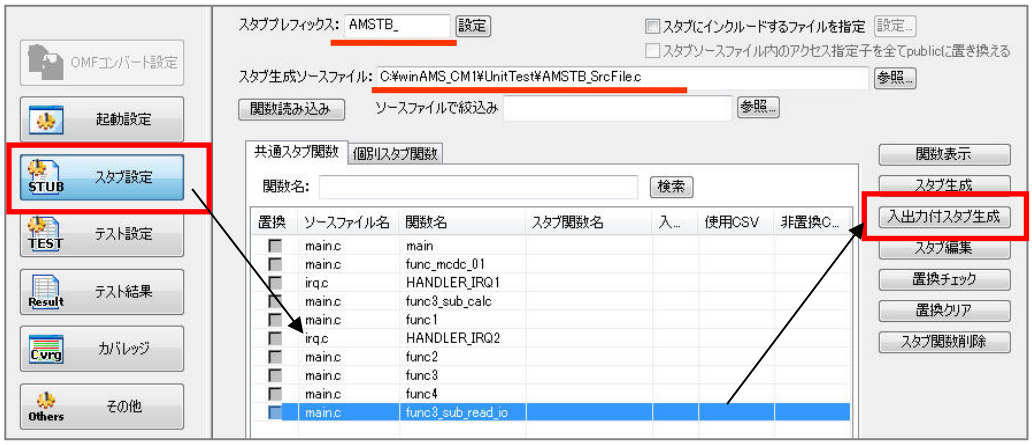
```
int func3_sub_read_io( int index )
{
    // 戻り値がdata_tableに依存して複雑
    if( data_table[index]>0x7f )
    {
        return data_table[index];
    }
    else
    {
        return -data_table[index];
    }
}
```

```
void func3( int enable, int mode )
{
    int retval;

    if( enable )
    {
        // スタブを作成して 戻り値を自由に設定する
        retval = func3_sub_read_io( mode );

        // 戻り値を使って分岐
        switch( retval )
        {
            case 0:
                gb_result.data = 0;
                break;
            case 1:
                gb_result.data = 50;
                break;
            case 2:
                gb_result.data = 100;
                break;
            default:
                gb_result.data = -1;
        }
        gb_result.ret_code = TRUE;
    }
    else
    {
        gb_result.data = 0;
        gb_result.ret_code = FALSE;
    }
}
```

スタブ関数の作成と指定

カバレッジマスターwinAMS のスタブ設定機能を使用して、func3_sub_read_io()のスタブ関数を作成します。

1. 「スタブ設定」のボタンを押します。



まず、設定の確認を行います。一番上にある項目「スタブプレフィックス:」とは、作成するスタブ関数の前に付ける文字列の指定です。この場合は、func3_sub_read_io()の前に AMSTB_を付けた、AMSTB_ func3_sub_read_io()の関数名のスタブ関数が作成されます。

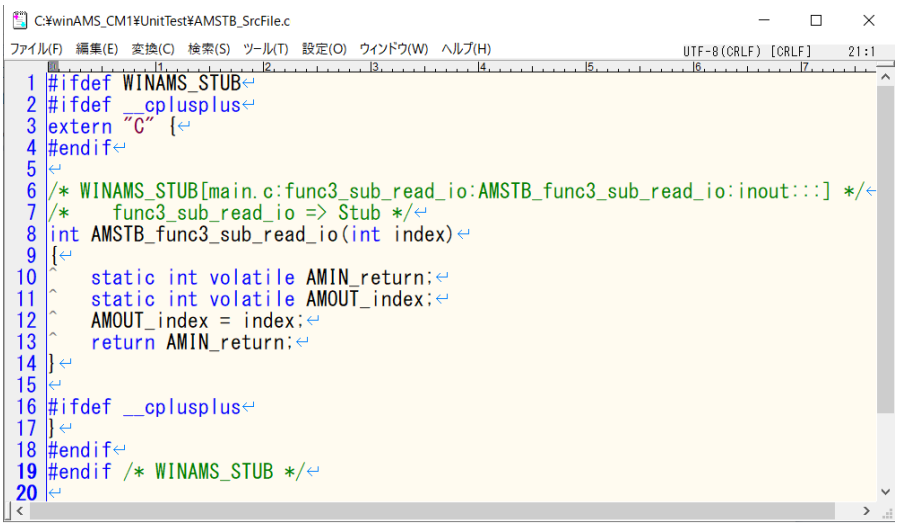
すぐ下の「スタブ生成ソースファイル:」には、作成するスタブ関数を保存するソースファイルの指定を行います。上図のようにファイル名が設定されていない場合は、「参照」ボタンを押して、ファイルパスを決定して下さい。C:¥winAMS_CM1¥UnitTest を指定して、この中にスタブ関数のソースファイル「AMSTB_SrcFile.c」が出来るようにします。

では、スタブの作成を行います。

2. func3_sub_read_io を選択して「入出力付スタブ生成」を押します。

これにより、入出力付きのスタブ関数「AMSTB_func3_sub_read_io」が別ファイル(AMSTB_SrcFile.c)に作成され、カバレッジマスターwinAMS のソースエディタが開きます。

このスタブ関数は、CSV ファイルの INPUT/OUTPUT で使用可能な変数を宣言したスタブとして自動生成されます。



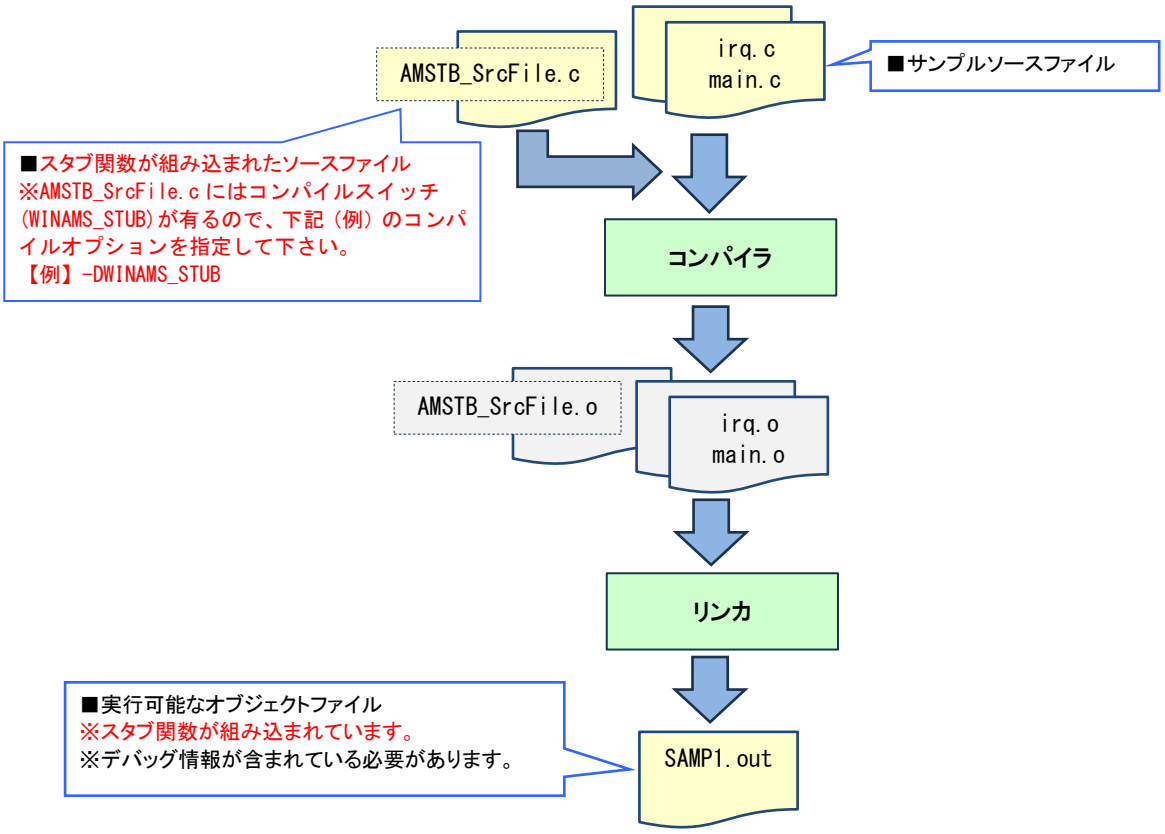
自動生成されたスタブ関数内のコードは、

① 戻り値をテストデータとして CSV ファイルから入力するための関数内 static 変数(AMIN_return)
② 引数を CSV ファイルの OUTPUT で確認するための関数内 static 変数(AMOUT_index)
③ ②で宣言した変数へ引数を代入する処理
④ ①で宣言した変数をリターンする処理

対象となる呼び出し関数の引数の数に応じて、②、③の生成数は異なります。

スタブ関数を実行可能なコードにする

では、このスタブ関数を組み込んで、ご利用のコンパイラ(開発環境)でビルドを行い、実行可能なオブジェクトファイルを作成します。



以上で、スタブ関数が追加されたオブジェクトファイル「SAMP1.out」が作成されました。

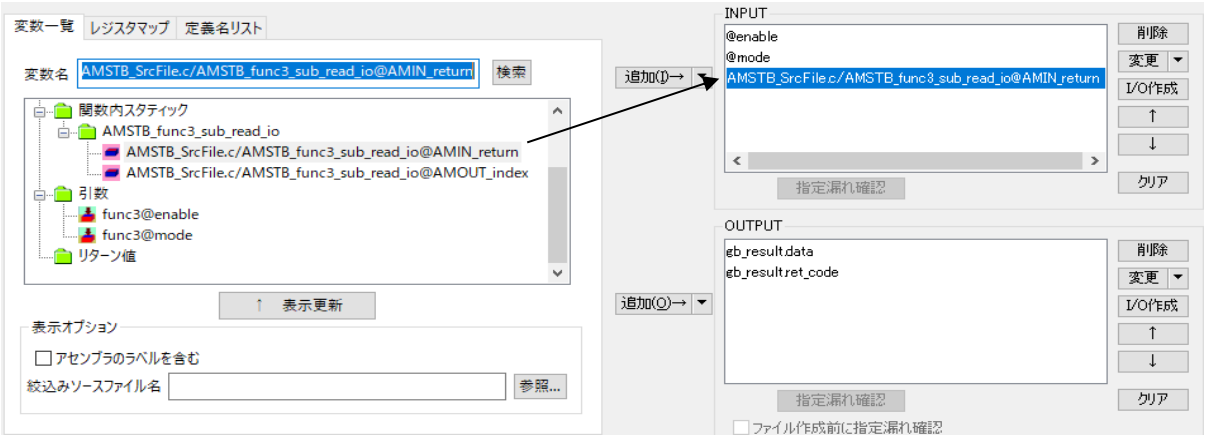
func3()のテスト入力データを作成する

では、前の実習と同様に、CSV ファイルに func3()のテスト入力データを作成します。func3()の入出力テスト条件を確認します。func3()の入出力条件は、以下の通りです。

入力変数: 引数 int enable, int mode
出力(評価)変数: グローバル変数 gb_result.data, gb_result.ret_code

これに、今回はスタブ関数の戻り値、ret を入力変数として追加します。CSV ファイル作成の手順は、実習1，2と同様です。

1. 「テスト CSV 作成」のボタンを押します。
2. 「モジュールテスト用 CSV」を選択します。
3. ファイル名に「func3_data」、関数名に「func3」を指定します。
4. 下図の様に、上の入出力条件を INPUT、OUTPUT に指定します。



5. 「OK」を押して、CSV ファイルを作成します。
6. 「テスト設定」のビューに切り替えて、「func3_data.csv」を EXCEL で開きます。

C0 カバレッジを 100%にするためのテストデータを作成します。下のようにデータを入力して下さい。
3つめの変数「AMSTB_SrcFile.c/AMSTB_func3_sub_read_io@AMIN_return」は、スタブ関数に作成した戻り値を指定するための変数です。ここでは、switch 文の分岐条件に合わせて、「0, 1, 2, default 値」をデータとして設定して、テスト毎の戻り値を変化させています。

また、引数の「@mode」は、実際の関数においては、この値が呼び出し関数の引数に渡されて使用されますが、今回はこの呼び出し関数のスタブを作成しているため、@mode は使用されません。今回のテストデータには、@mode 欄がありますが、データ値は任意で良いことになります。ただし、mode はサブ関数に渡される引数であるため、テスト対象の func3()の出力要因に当たります。そのため、サブ関数に正しい引数を渡しているかを確認することを要求される場合も有ります。

下記のデータでは、テストに使用されないデータはすべて、「111」を設定しています。このデータは、テストで使われることは無いため、実際はどんな値であっても構いませんが、テストに関係しないデータであることを明示して、特に設計したテストデータのレビュー時に、テストデータの可読性を高める工夫をしています。
実際の運用でのテスト設計においては、この様に、未使用のデータ値を予め決めておくようなテスト指針を設定しておくことが、単体テストを効率的にこなすための、1つのノウハウとなります。

	A	B	C	D	E	F
1	mod	func3		3	2	
2	#COMME	@enable	@mode	AMSTB_SrcFile.c/AMSTB_func3_sub_read_io@AMIN_return	gb_result.data	gb_result.ret_code
3		0	111	111		
4		1	111	0		
5		1	111	1		
6		1	111	2		
7		1	111	3		

呼び出し関数の置換とテスト実行

では、作成したスタブ関数とテストデータを使って、func3()の単体テストを実行します。まず、スタブ関数の置換設定を行います。

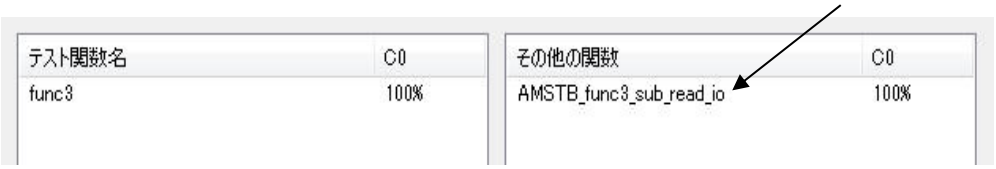
1. 「スタブ設定」のビューに切り替えます。
2. func3_sub_read_io の行の置換ボックスをチェックします。
3. 「テスト設定」のビューに切り替えます。
4. テスト CSV ファイル「func3_data.csv」を選択します。
   ※「func2_data.csv」の選択は外して(OFF にする)ください。

テスト実行します。手順は、実習1、2と同様です。

5. 「シミュレータ起動」のボタンを押します。

結果が出力されます。まず、カバレッジを確認して下さい。

「カバレッジ」ボタンを押して、ビューを切り替えます
「テスト関数名」→「その他の関数」の欄に、スタブ関数(AMSTB_func3_sub_read_io)が表示されていることを確認します。この欄に表示される関数は、func3 をテストした際に呼び出された他の関数です。確かに、スタブ関数に置換されて実行されていることが分かります。



また、func3 のカバレッジは 100%となっており、正しく switch 文を分岐して網羅実行されたことが分かります。出力結果の CSV ファイルは、以下のようになります。

	A	B	C	D	E	F	G
1	mod	func3		3	2		
2	#COMME	@enable	@mode	AMSTB_SrcFile.c/AMSTB_func3_sub_read_io@AMIN_return	gb_result.data	gb_result.ret_code	
3		0	111	111	0	0	NO Check
4		1	111	0	0	1	NO Check
5		1	111	1	50	1	NO Check
6		1	111	2	100	1	NO Check
7		1	111	3	-1	1	NO Check

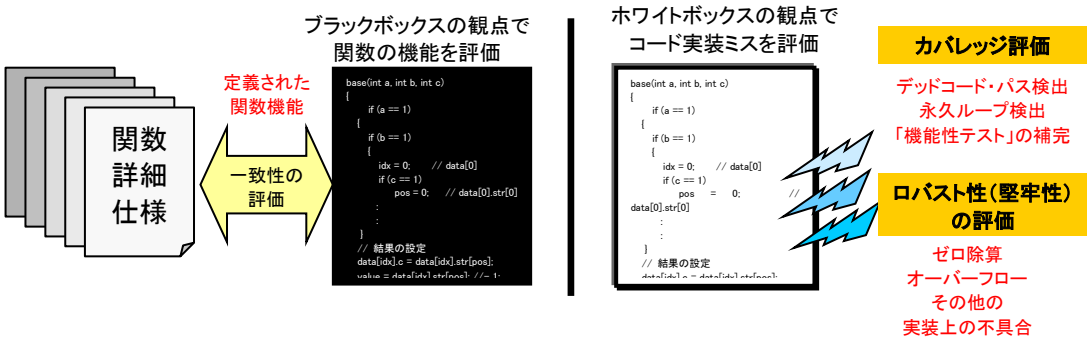
以上が、呼び出し関数のスタブ作成と、置換機能についての実習でした。元の評価対象ソースを書き換えることなく、関数の呼び替え(置換)が行えることが、ご理解頂けたと思います。

C1、MC/DC カバレッジテスト入力データ作成支援機能

実習4の内容と目的

前回の実習3までは、関数仕様書やテスト項目書などの設計情報から作成した CSV テストデータを作成したと想定して、それを使用したテスト作業の方法を学習しました。

開発したソースコードがその関数の仕様と一致しているかを確認する観点では、テストデータは関数設計仕様から作成されるべきであり、これをテストすることで開発したソースコードが設計仕様を満たしているかを確認することが可能です。これは、関数をブラックボックスとした、機能性の評価に当たります。
しかしながら、単体テストにおいては、仕様に規定された以外の例外要素により、関数が異常状態を起こさないかどうかの「ロバスト性(堅牢性)」確認や、開発したソースコードに、あってはならない分岐やコード(デッドコード)が存在しないかを確認するための「カバレッジ」テストが必要になります。この確認は、関数の潜在バグを減らし関数の品質を高める上で重要なテストの観点となります。



カバレッジマスターwinAMS には、ガイオの他の解析ツール「CasePlayer2」と連携して、テストデータの効率化を図るための以下の機能が搭載されています。

■テスト入出力変数の自動検索機能
評価対象関数を CasePlayer2 で解析することで、その関数のテスト条件となる入出力変数を自動検索します。
実際の製品開発プロジェクトでは、グローバル変数の個数が膨大であるため、関数のテスト条件となる変数の選択が容易ではありません。
この機能を利用すると、関数が読み書きしている外部変数のみをまとめて表示するため、関数の入出力条件の選択が容易となり、設定ミスも起こりにくくなります。

■C1 カバレッジを満たすテストデータ設計支援機能
CasePlayer2 の解析により、対象関数の C1 カバレッジを満たす最少数のテストデータを生成することが可能です。カバレッジマスターwinAMS では、CasePlayer2 の解析データを基に、関数内に存在する if や switch などの条件文を自動検索して表示し、各々の条件文の分岐を実行するための入力条件を整理して設計するための機能が搭載されています。また、各分岐のネスト構造も CasePlayer2 により解析され、全てのネストを実行するための条件の組み合わせを自動生成します。

ただし、本機能で作成できるテストデータは、ソースコードの解析により作成されます。ソースコードの構造を満たすカバレッジテスト(構造カバレッジ計測)は可能ですが、関数仕様に基づいたテストとは異なります。

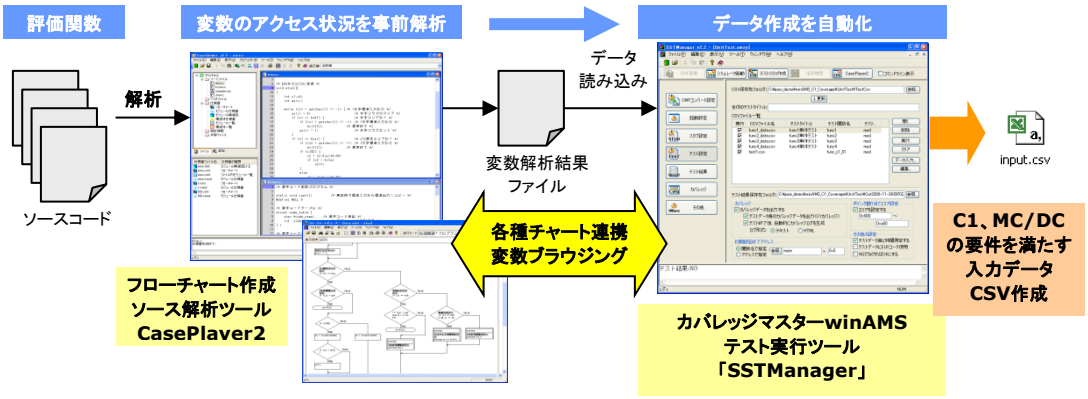
■MC/DC カバレッジを満たすテストデータ設計支援機能
上記の C1 カバレッジの場合と同様に、MC/DC の要件を満たすテストデータの組み合わせを自動生成します。

この実習では、関数の仕様確認のテストではなく、C 言語でコーディングしたソースコードの構造を満たすテストデータを自動生成する事で、デッドコードや、不要な、あるいは間違った分岐などの実装ミスが無いことを確認するための構造カバレッジテストを効率的に行う方法を学習します。

CasePlayer2 との連携

CasePlayer2 は、ソースコードからフローチャートやモジュール構造図などの仕様書を作る、リバース CASE ツールです。このツールには、仕様書を作るために、ソースコードを解析する機能が含まれています。関数に対する解析には、その関数がアクセスする外部変数の検索とリスト化、変数の参照/代入箇所の検索とリスト化などの内容が含まれます。

カバレッジマスターwinAMS は、CasePlayer2 で予めソースコードを解析した結果を取り入れると、前述の機能が使用出来るように、ツール連携を行っています。



では、これから、上記の機能を使用して、対象関数の C1 カバレッジテストのためのテストデータ作成を行う実習に進みます。

実習4：CasePlayer2 と連携したカバレッジテストデータ作成機能を使う

この実習では、CasePlayer2 との連携で、C0、C1 カバレッジテストを効率的に行う方法について学習します。課題としては、以下の関数 func4()を使用して、この func4()の C1 カバレッジを 100%にするデータを作成することを目標とします。

func4()のソースコードは以下の通りです。関数の上には、多数のグローバル変数が宣言されていますが、この変数の中から、func4()が使用している変数のみを自動検索するところから始めます。

```
int gb_input;
int gb_output;
int gb_valA;
int gb_valB;
int gb_valC;
int gb_valD;
int  gb_a, gb_b, gb_c, gb_d, gb_out;
char gb_unused1, gb_unused2;

int func4( int code )
{
    int return_value=FALSE;
    int i;
    if( gb_a > 10 )
    {
        if( gb_b > 20 && gb_c > 30 )
        {
            gb_out = 0;
        }
        else
        {
            gb_out = -1;
        }
        return_value = FALSE;
    }
    else
    {
        switch( code )
        {
            case 1:
                gb_out = 1;
                break;
            case 2:
                gb_out = 2;
                break;
            case 3:
                gb_out = 3;
                break;
            default:
                gb_out = -1;
                break;
        }
        return_value = FALSE;
    }

// 下の条件式は、上の演算結果(gb_out)を使って比較している
    // 動的に決まる境界値は静的解析できない
    gb_a = gb_out;
    if( gb_d == gb_a )
    {
        gb_out = 4;
    }
    else
    {
        gb_out = 5;
    }

    return return_value;
}
```

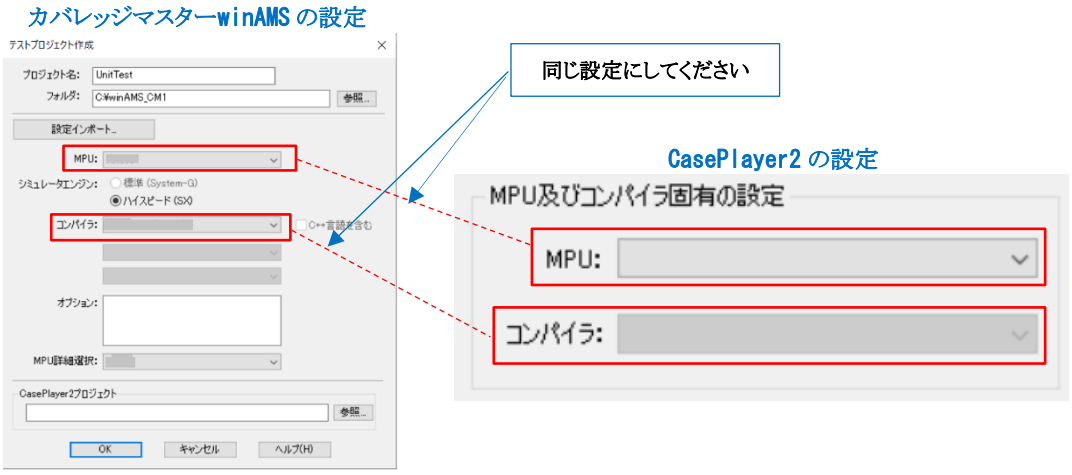
CasePlayer2 の静的解析プロジェクトの作成

では、まず CasePlayer2 を起動して、ソースコードを解析するためのプロジェクトの作成を行います。

1. Windows「スタート」→「GAIO CasePlayer2」 →「CaseViewer.exe」 を起動します。
2. 「ファイル」メニュー →「新規作成」→「プロジェクト」 を選択します。
3. 「プロジェクト名」、「場所」を設定します。
   ー プロジェクト名： CP2Project　　場所： C:¥winAMS_CM1
4. 言語設定が「ANSI-C」であることを確認します。

加えて、使用するマイコン名を選択（CasePlayer2 でソースコードを解析する際に必要）します。

5. 「MPU 及びコンパイラ固有の設定」で「MPU」と「コンパイラ」を選択します。
   ※カバレッジマスターwinAMS の「テストプロジェクト作成」ダイアログで設定した「MPU」及び「コンパイラ」と同じ設定にしてください。



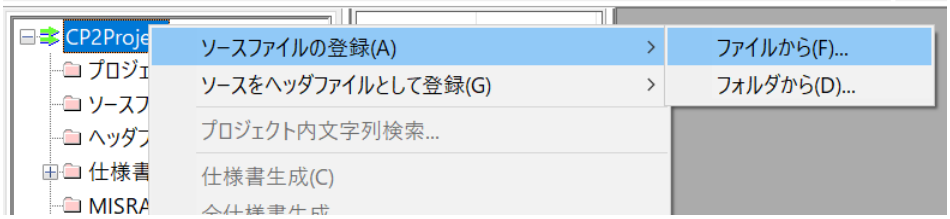
この「MPU 及びコンパイラ固有の設定」は、ソースコードにあるクロスコンパイラ特有の記述（方言）に対応するためのものです。クロスコンパイラの方言は、そのままでは CasePlayer2 で解析エラーとなってしまうため、CasePlayer2 の機能「C オプションパラメータ」を使用して、方言の解釈方法を指定して対応します。ここで該当するマイコンとコンパイラを選択しておくことで、「C オプションパラメータ」が自動設定されます。

本実習環境では、サンプルコードに方言は含んでいないため、「C オプションパラメータ」の機能は使用されませんが、設定例として上記の設定を行います。

※C オプションパラメータについては、この章末の「【参考】C オプションパラメータについて」をご参照ください。

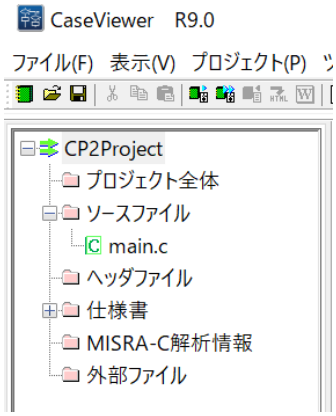
次に、解析対象のソースコード(main.c)をプロジェクトに登録します。今回の実習では、main.c の中に全てのテスト対象関数が記述されているため、このソースファイルだけを登録します。

6. CaseViewer のツリーのプロジェクト名「CP2Project」を右クリックします
7. 「ソースファイルの登録」→「ファイルから」を選択します



8. 「main.c」を選択します
   C:¥winAMS_CM1¥target¥main.c
9. 「開く」を押して、main.c を登録します
   CasePlayer2 プロジェクトへソースが登録されます

CasePlayer2 のプロジェクトが作成されました。左の CP2Project のツリーを開いてみて下さい。ソースファイルに main.c が登録されているのが確認できます。



CasePlayer2 に必要な設定ー1： 仕様書生成

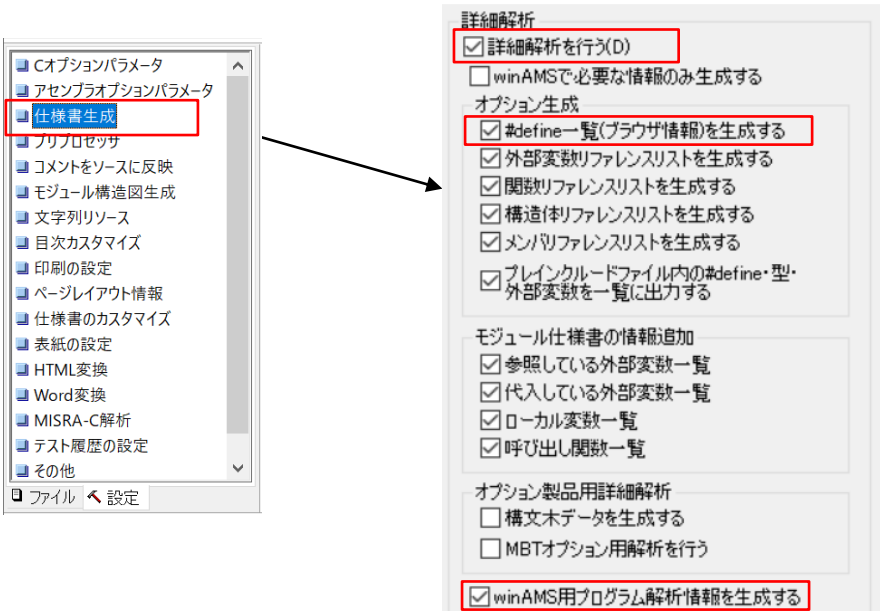
この main.c を対象にして、この中に含まれるテスト対象関数 func4()を事前解析します。その前に、CasePlayer2 の解析設定を確認します。
1つめは、CasePlayer2 の基本設定です。

1. 「設定」タブの「仕様書生成」をダブルクリックして設定画面を表示します。
2. 「詳細解析」のオプションを下図のように設定します。

■詳細解析を行う
■#define 一覧(ブラウザ情報)を生成する
■winAMS 用プログラム解析情報を生成する

上記の3つのオプションは、カバレッジマスターwinAMS と連携して使用するために必ず ON にする必要があります。他のオプションは生成する仕様書(帳票の種類)の選択です。必要に応じて使用します。

3. 「OK」で閉じます。



CasePlayer2 に必ず必要な設定

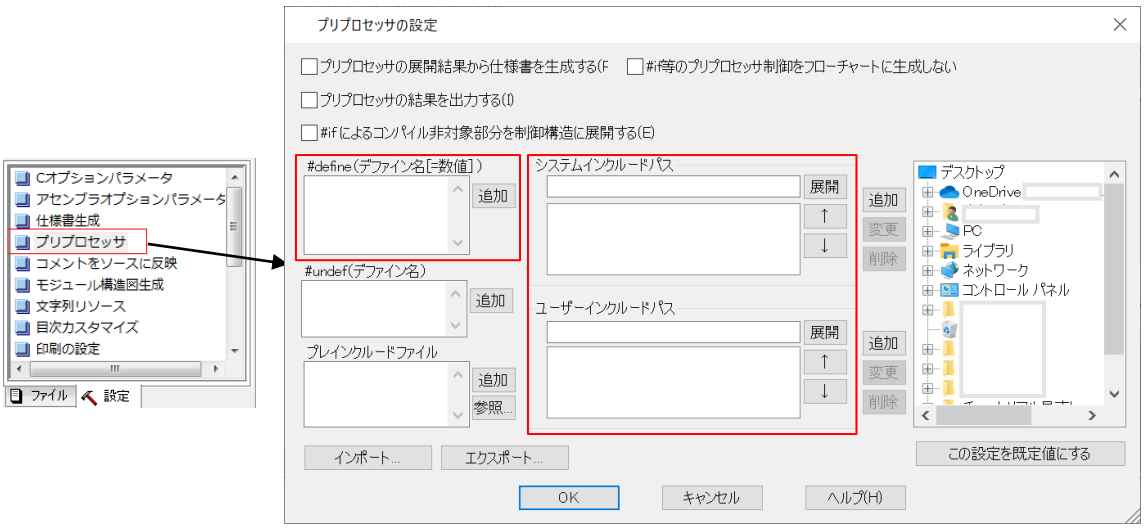
CasePlayer2 に必要な設定－2： プリプロセッサ

2 つめは、解析時に必要なヘッダファイルや#define 設定の扱いに関するプリプロセッサの設定です。

1. 「設定」タブの「プリプロセッサ」をダブルクリックして設定画面を表示します。
2. そのまま「OK」で閉じます

このプリプロセッサでは、クロスコンパイラでソースコードをコンパイルする場合と同様に、ソースにインクルードされたヘッダファイルの検索パスと、#ifdef などの切換に使用している#define 値の設定を与えます。基本的には、クロスコンパイラのプリプロセッサと同じ内容を設定する必要があります。

本チュートリアルにはヘッダファイルや#define による切換はないため、設定を行う必要はありませんが、実際の運用においては、下図の「システムインクルードパス」(<>で囲まれた#include 記述)と「ユーザーインクルードパス」("" で囲まれた#include 記述)、#defineを設定する必要があります。これを設定しない場合は、解析時にヘッダファイルが読み込めず、FILE I/O エラーが発生します。



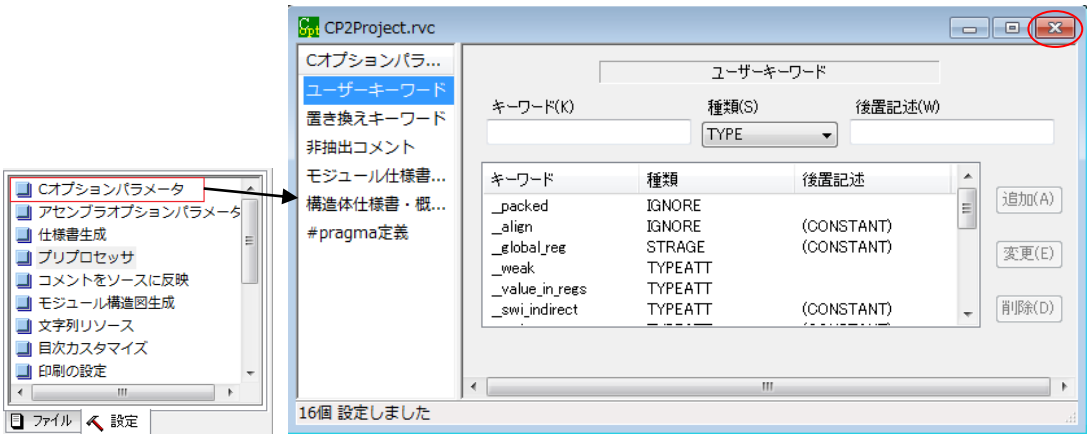
CasePlayer2 に必要な設定－3： C オプションパラメータ

3 つめの設定は、「C オプションパラメータ」です。これは、開発に使用しているクロスコンパイラ独自の方言に対応するためのものです。CasePlayer2 の解析エンジンは、指定した言語仕様（ANSI-C、C99、GNU-C）に基づいてソースの解析を行いますが、クロスコンパイラ独自の言語拡張（方言）には対応できす、そのままでは解析エラーとなります。そこで、「C オプションパラメータ」に方言のキーワードとその解釈方法を設定することで、ソースコードを書き換えることなく、正しく解析できる様になります。

CasePlayer2 のプロジェクト新規作成時に、「ターゲット CPU 及びコンパイラ固有の設定」で使用するマイコンとコンパイラを選択しておけば、「C オプションパラメータ」は自動設定されます。各コンパイラの標準的な記述であれば、自動設定されるパラメータで、方言を含むソースは正しく解析できます。ただし、コンパイラのバージョンアップや仕様拡張などで、新たな記述やキーワードが追加された場合は、ユーザーが「C オプションパラメータ」に設定を加えることで、対応が可能です。

1. 「設定」タブの「C オプションパラメータ」をダブルクリックして設定画面を表示します。
2. キャプションバー右側の「×」ボタンを押して設定画面を閉じます

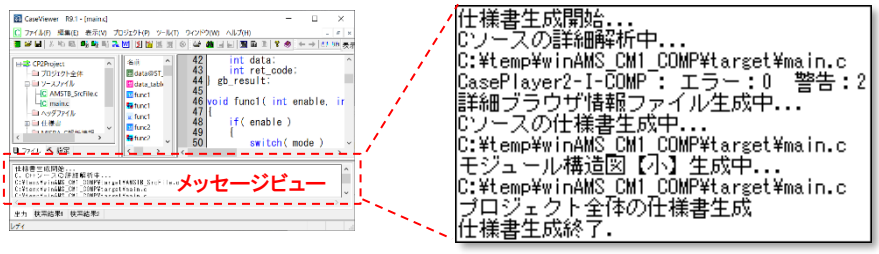
本チュートリアルのサンプルには、CasePlayer2 が解析できない方言は含まれていないため、設定の追加は必要有りません。設定例については、章末の「（参考）C オプションパラメータについて」に解説されています。



ソース解析と仕様書生成の実行

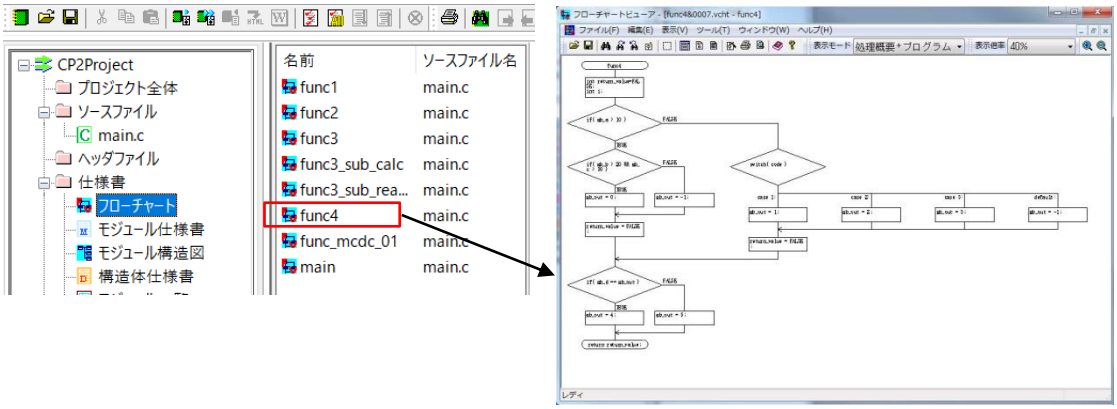
では、解析と仕様書の生成を実行します。

1. 「プロジェクト」メニューの「仕様書を生成」を選択します。
   ー メッセージビューに解析状況（下記）が表示されます。



これで、解析と仕様書生成が終了しました。この CasePlayer2 の基本的な機能を少し試してみましょう。例えば、簡単な例として、評価対象の関数 func4()のフローチャートを表示してみます。

2. 「仕様書」フォルダの「フローチャート」を選択します。
3. 「名前」のリストから func4 をダブルクリックします。

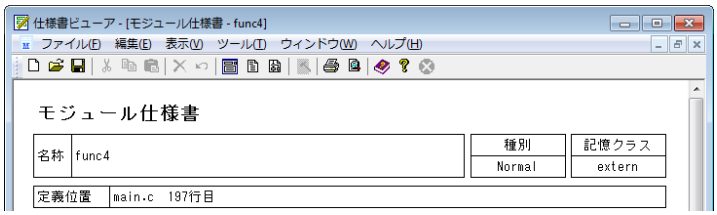


表示されるチャートが、これから単体テストの C1 カバレッジデータを作成する関数 func4()のフローチャートです。条件文のネスト構造などが、分かり易く表示されています。

CasePlayer2 のツリーにある「仕様書」には、このツールが作成する仕様書の種類が示されています。では、評価対象関数 func4()の解析結果を参照してみましょう。

4. 「仕様書」フォルダの「モジュール仕様書」を選択します。
5. 「名前」のリストから func4 をダブルクリックします。

表示される「モジュール仕様書」は、CasePlayer2 がソースコードを解析して自動作成した func4()の関数仕様書です。



この中には、func4()が参照、代入している外部変数のリストが含まれています。これらの情報を、カバレッジマスターwinAMS に取り込むことで、func4()のテスト条件設定が容易に行えるようになります。

参照している外部変数一覧		
変数名	定義ファイル名	行番号
gb_b	main.c	192
gb_c	main.c	192
gb_d	main.c	192
gb_out	main.c	192
gb_a	main.c	192

代入している外部変数一覧		
変数名	定義ファイル名	行番号
gb_out	main.c	192

CasePlayer2 が解析した func4()の変数参照情報例

上記の外部変数一覧を表示するためには、CasePlayer2 の仕様書生成の設定のオプションを追加で ON にする必要があります。

「設定」タブ→「仕様書生成」→「モジュール仕様書の情報追加」で、下記のオプションを ON にしてください。

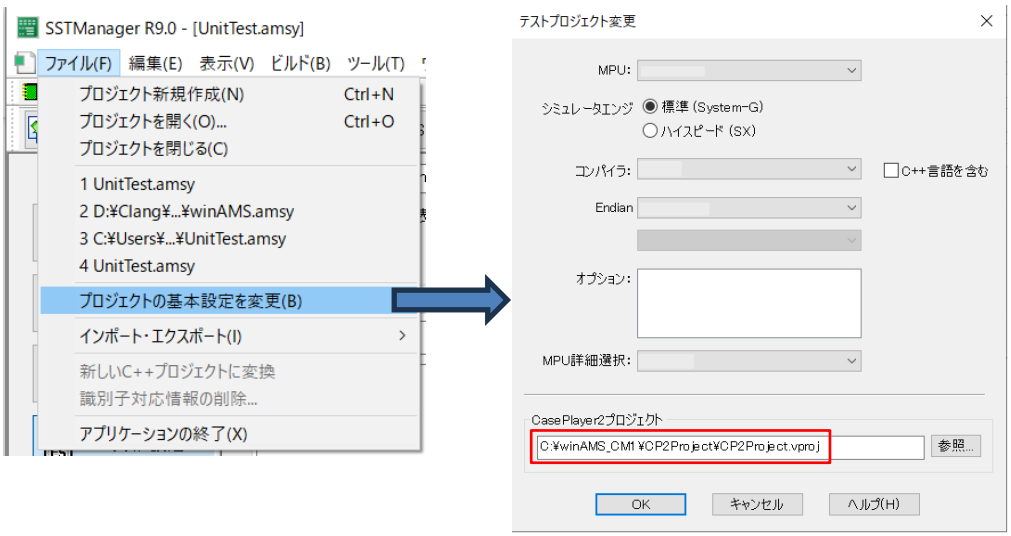
・参照している外部変数一覧
・代入している外部変数一覧

カバレッジマスターwinAMS に CasePlayer2 を連携

では、カバレッジマスターwinAMS の SSTManager に戻って、func4()の C1 カバレッジテストデータを作成します。解析結果は既に取り込まれています。最初に、カバレッジマスターwinAMS に CasePlayer2 を連携するための設定をします。

1. SSTManager の「ファイル」メニューの、「プロジェクトの基本設定を変更」を選択します。
2. 表示されるダイアログで、CasePlayer2 のプロジェクトファイルを設定します。
   - C:¥winAMS_CM1¥CP2Project¥CP2Project.vproj
3. 「OK」を押して閉じます。

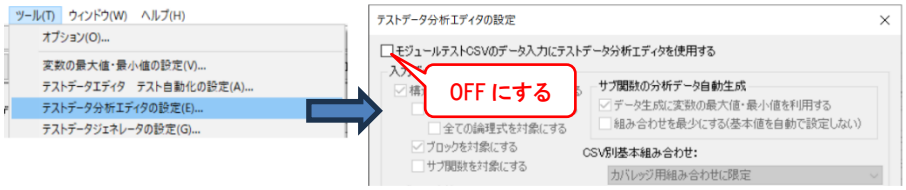
これで、CasePlayer2 での静的解析の結果が、カバレッジマスターwinAMS で利用できる状態になります。



入出力変数自動検索機能を使って 変数を選択する

※(準備)**********
テスト CSV ファイル作成の前に下記設定を行って下さい。

[設定項目]本チュートリアルにおける「ATDEditor」を使用するための設定になります。
SSTManager のメニュー [ツール]→[テストデータ分析エディタの設定]→[モジュールテスト CSV のデータ入力にテストデータ分析エディタを使用する]のチェックを OFF にして下さい。

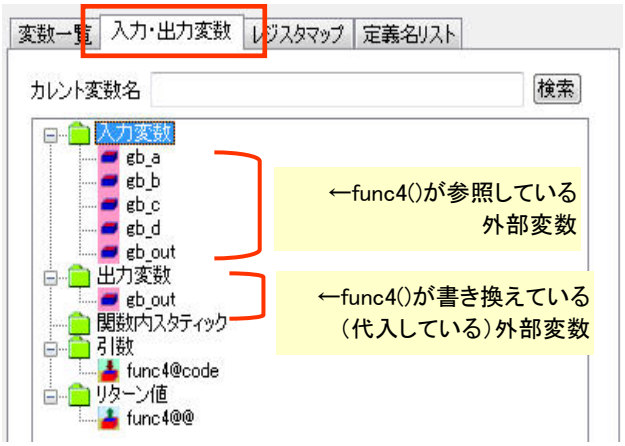


****

では、func4()のテスト CSV ファイル作成に進みます。CSV ファイルの作成手順は、実習1～3と同じです。

1. SSTManager の「テスト CSV 作成」ボタン押して、「モジュールテスト用 CSV」を選択します。
2. テストタイトル、ファイル名(func4_data)、関数名(func4)を設定します。
3. 変数リスト欄に、「入力・出力変数」のタブがあることを確認します。

この「入力・出力変数」のタブは、CasePlayer2 の解析結果を使用している時のみ利用可能なタブです。ここに、CasePlayer2 の「モジュール仕様書」に解析結果として表示されていた内容が取り込まれます。「入力変数」は func4()内で参照している外部変数、「出力変数」は func4()内で書き換えて(代入して)いる外部変数です。

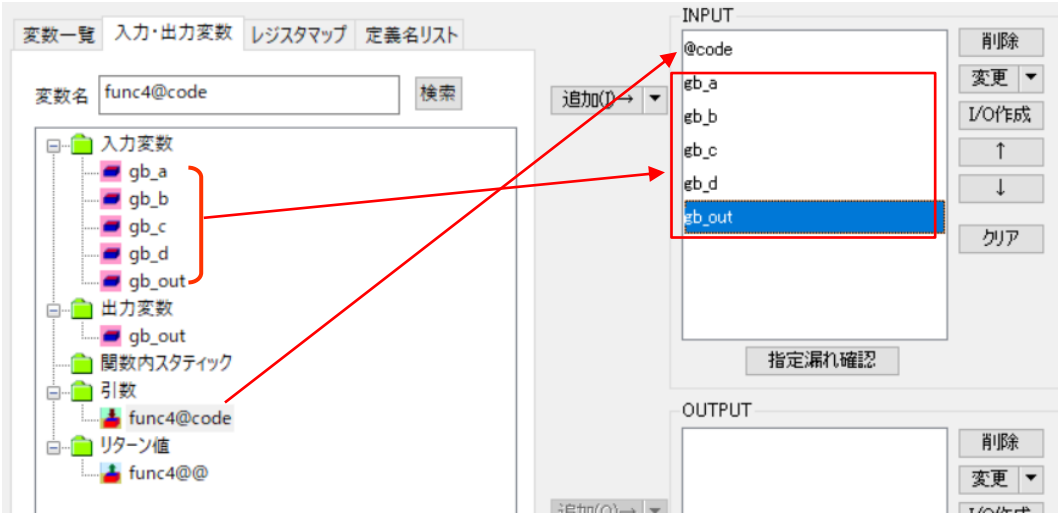


この関数 func4()を実行するためには、参照されている外部変数には値を設定する必要があります。そのため、「入力変数」に表示される参照外部変数の全てと引数をテスト入力条件とすれば、単体テストを実施するための必要十分条件となります。
また、テスト結果として評価する対象の変数は、この関数がアクセスして値を変化させる変数の中から選択されるはずです。そのため、「出力変数」に表示される変数やリターン値(関数の戻り値)の中から、この候補を選択すれば良いことになります。

このプロジェクトには実際には多数の変数があります。直ぐ左の「変数一覧」タブで外部変数の全てを確認することができます。(実習1～3では、この「変数一覧」から変数を選択しました。)テストに関係する変数は、関数がアクセスしている変数だけであるため、この「入力・出力変数」に表示されない変数は、この関数のテストには無関係であり、テスト条件に加わることはない事になります。

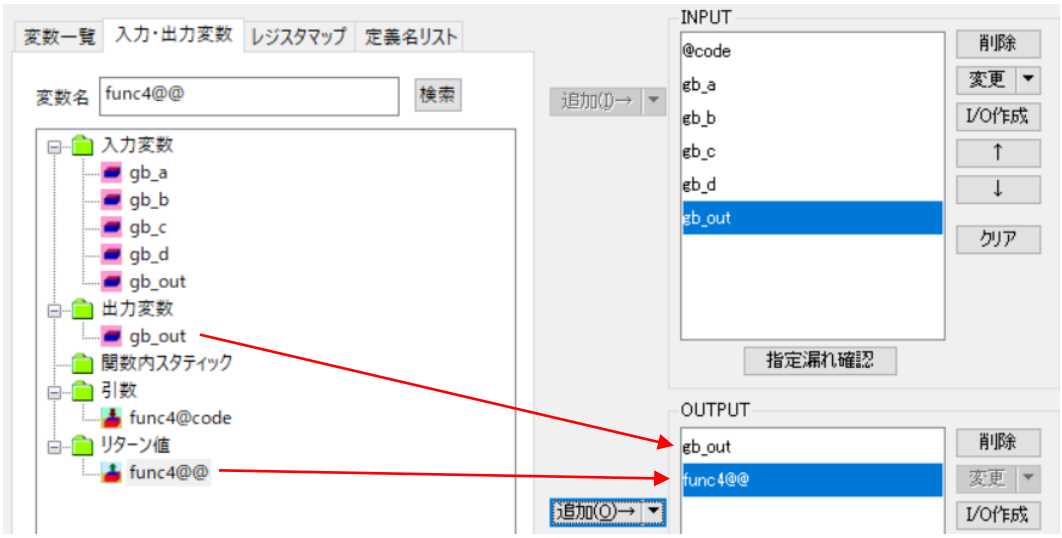
では、この要件を実際に設定してみましょう。ますは、入力条件から設定します。

4. 引数のフォルダから func4@code を INPUT に追加します。
5. 入力変数のフォルダから gb_a、gb_b、gb_c、gb_d、gb_out の 5 つを INPUT に加えます。



次に評価対象(出力)の変数を設定します。今回は、変化する変数をすべて評価対象としてみます。

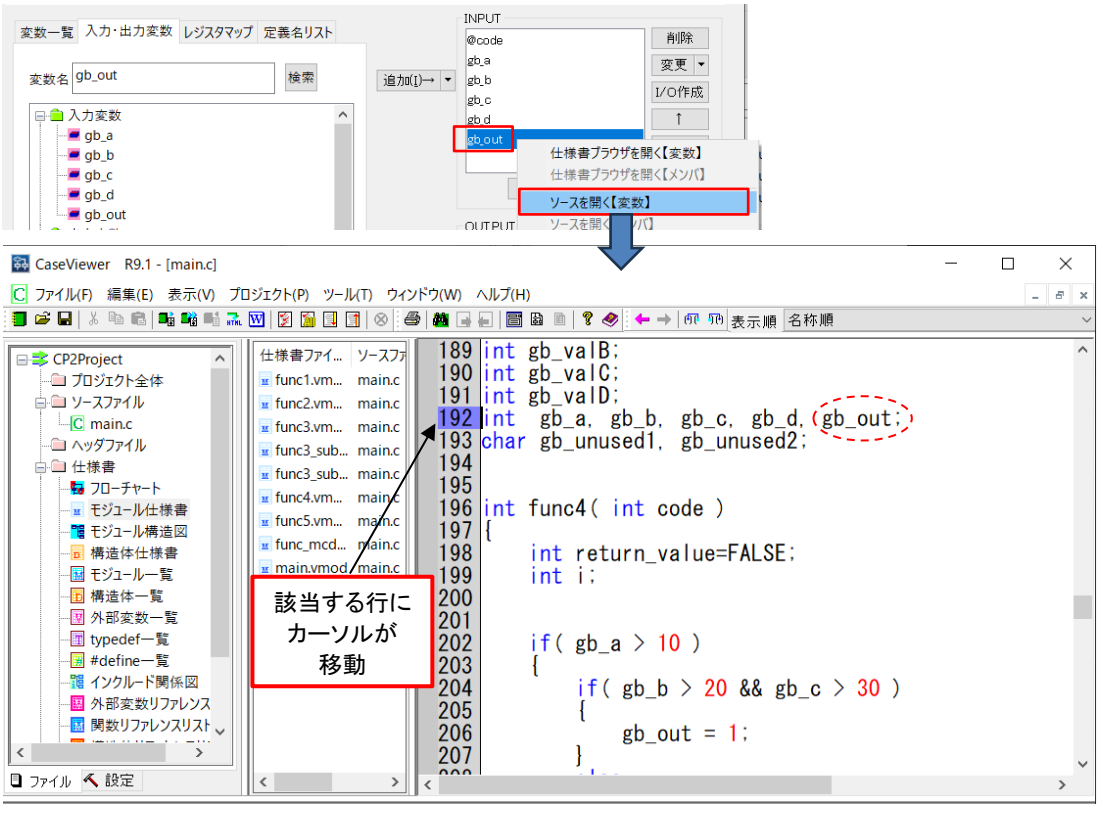
6. 出力変数のフォルダから gb_out を OUTPUT に追加します。
7. リターン値のフォルダから func4@@ を OUTPUT に追加します。



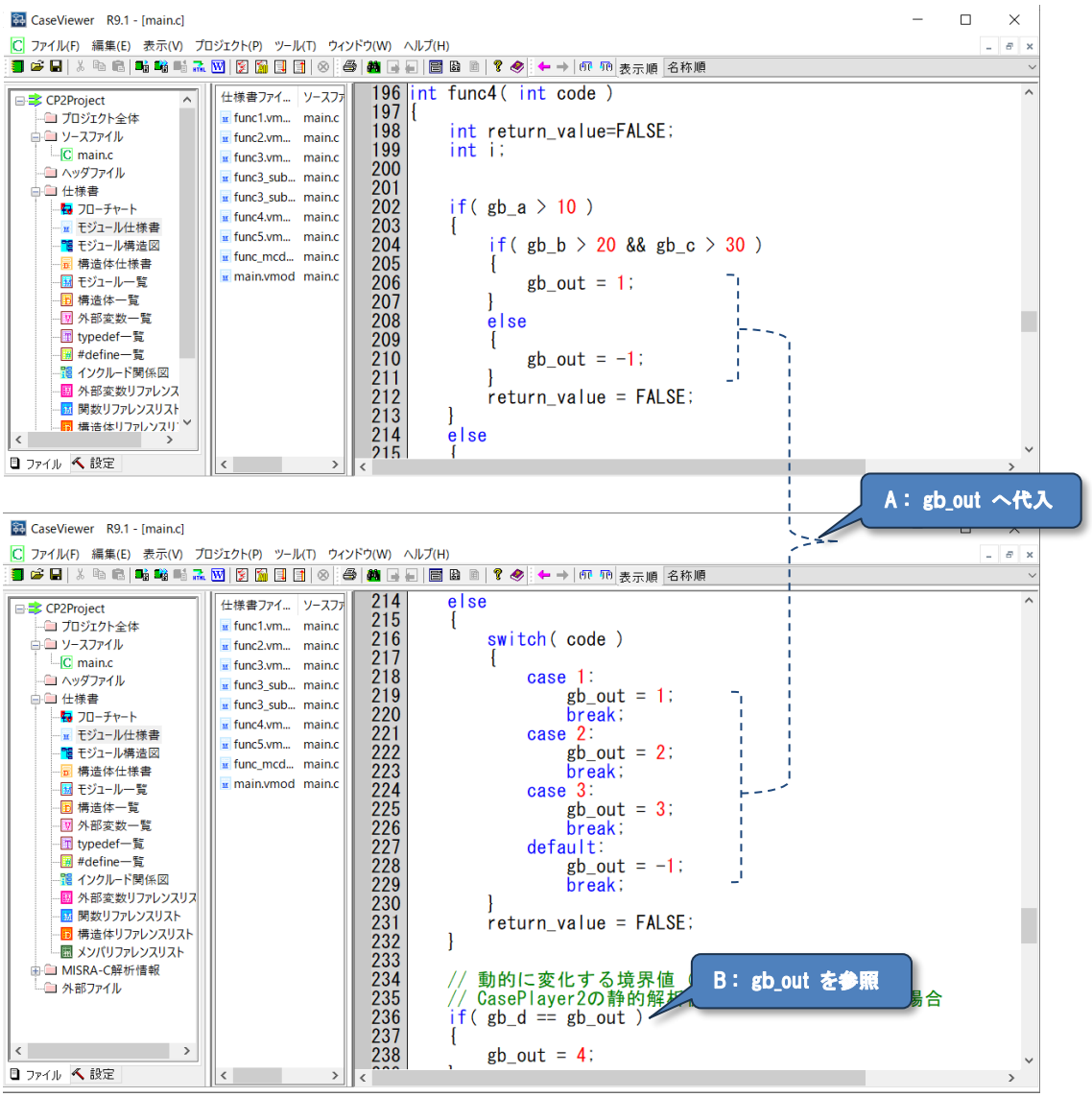
これで、func4()の単体テストの入出力条件を指定することができました。ただ、変数 gb_out は入力変数と出力変数の両方に指定しています。ソースコードでこの変数の使われ方を確認してみましょう。ここで、カバレッジマスターwinAMS と CasePlayer2 との連携機能を使用します。

8. INPUT欄の gb_out の上で右クリックし、「ソースを開く【変数】」を選択します。

CasePlayer2 のエディタで、gb_out が使用されている最初の行にジャンプします。この機能は、変数名を右クリックだけで該当するソースコードを参照できるため、特に多量のソースファイルを扱うプロジェクトの場合では、非常に便利な機能です。



この位置から下へ順に gb_out の変数の使用状況を確認すると、if 文と switch 文の内部で定数値を代入(下図:A)する処理があります。その後の if 文「if(gb_d == gb_out)」(下図:B)で、他の外部変数 gb_d との比較のために参照されています。CasePlayer2 の解析結果で、入力変数と出力変数の両方に gb_out が出現しているのは、このためです。



変数 gb_out は外部変数ですが、func4()内の処理で値が決定するため、テスト入力条件に加える必要はありません。そこで、今回は gb_out を入力条件対象から外します。(※関数内部で決定される変数であっても、テスト指針によっては、値が設定されない実行パスが存在しないことを確認するために、必ず初期値を与えることを要件とする場合もあります。)

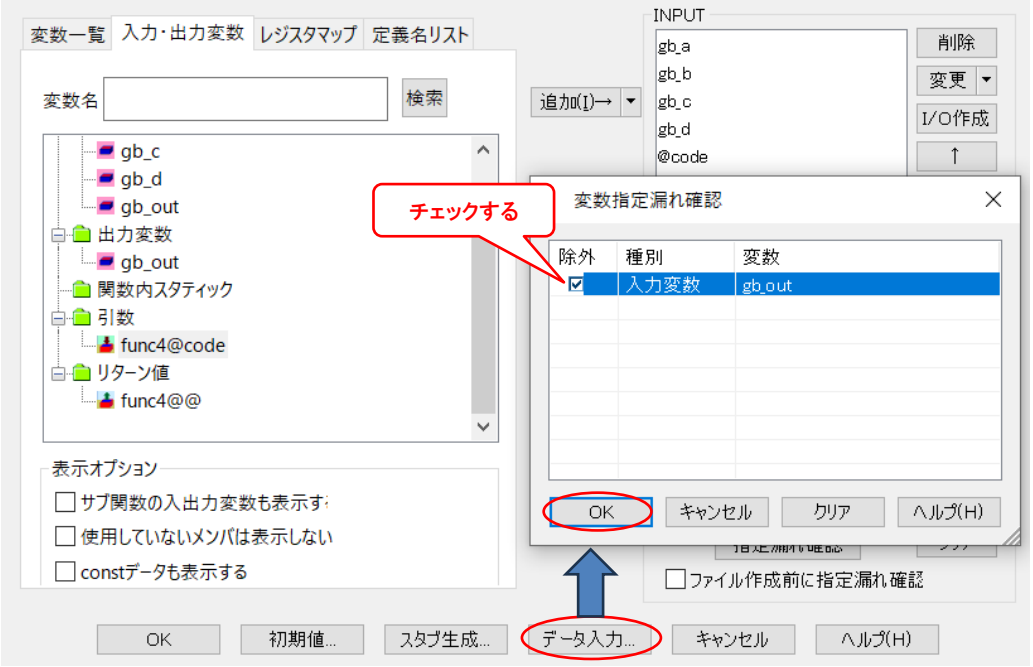
9. INPUT の欄で gb_out を選択し、削除ボタンを押します。
   －gb_out が入力条件の対象から外れます。

これで、テスト条件となる変数の選択が終わりました。では、C1 カバレッジテストデータ作成に進みます。

10. 下に並んだボタンから、「データ入力」を押します。

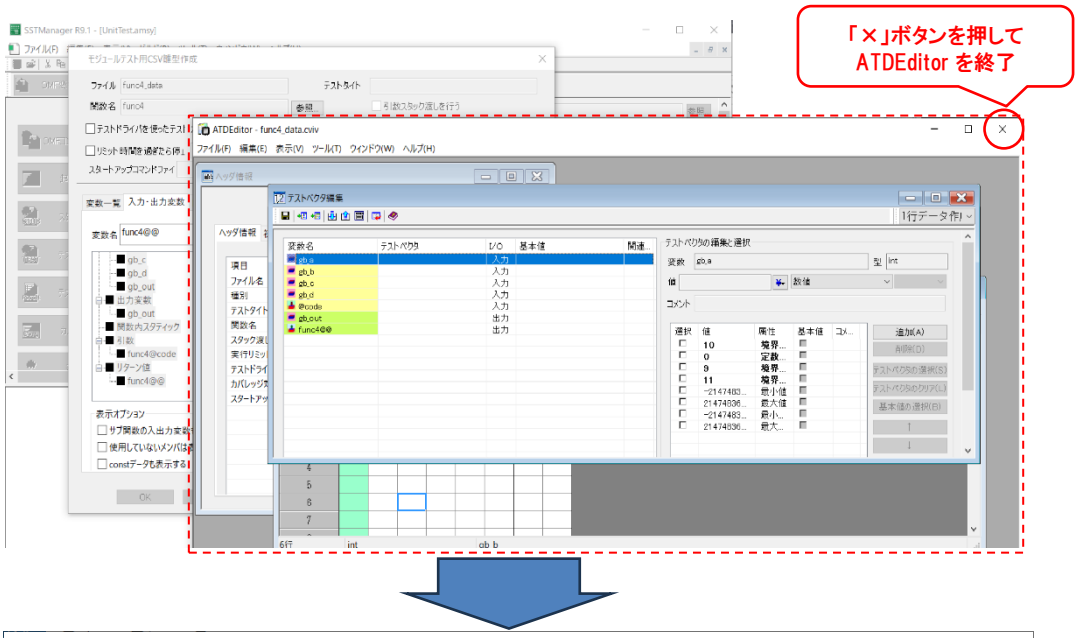
「変数指定漏れ確認」ダイアログが表示されます。これは、入力対象に指定可能な変数が、INPUT 欄に指定されていない場合に、次のデータ設計に進む前に確認を求めるものです。今回は、gb_out を入力対象から外したため、この確認が表示されています。「除外」をチェックしておけば、次にこのウインドウに戻ったときに、確認を求められなくなります。

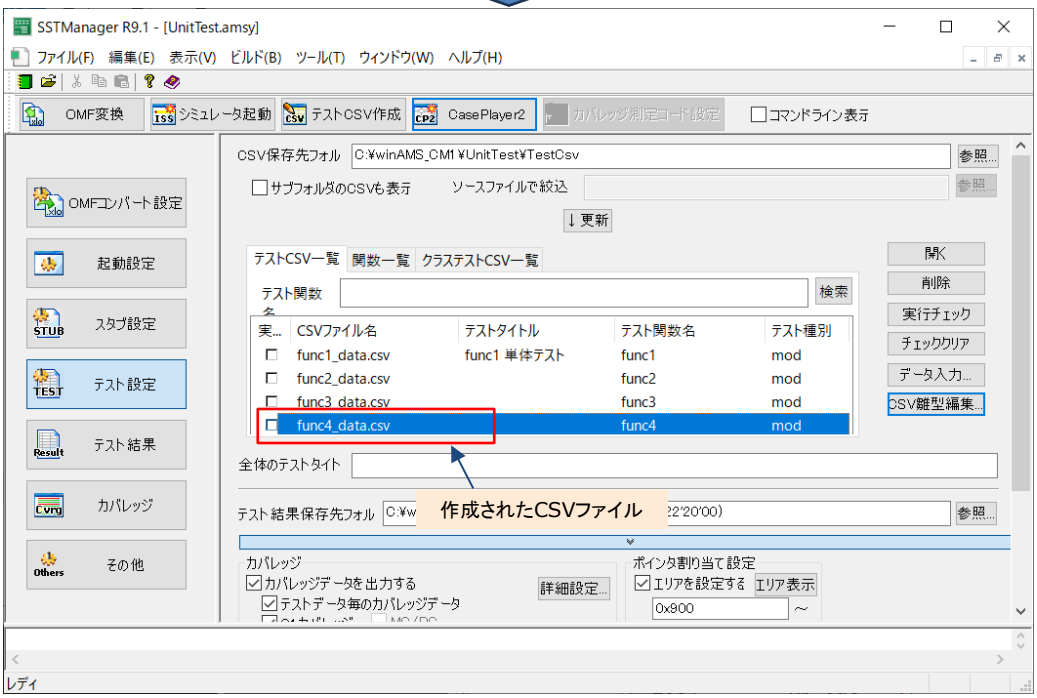
11. 「変数指定漏れ確認」ダイアログで gb_out の「除外」をチェックします。
12. 「OK」ボタンを押して CSV ファイルを作成します。



「ATDEditor」画面が表示されますので、キャプションバーの「×」ボタンを押して一旦 ATDEditor を閉じ、CSV ファイルが作成されていることを確認します。

次頁では、再度 ATDEditor を開き、C1 カバレッジを満たすためのテストデータを設計します。





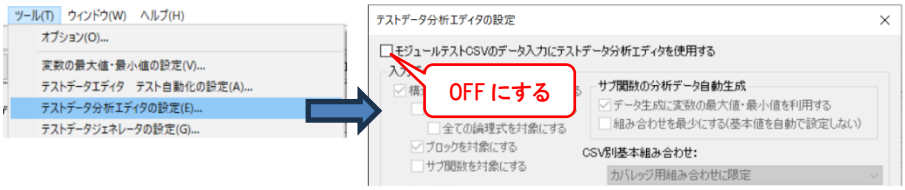
C1 カバレッジのためのデータを設計する

では、作成した CSV ファイルに、C1 カバレッジを満たすテストケースを、テストデータ生成機能を使用して作成します。

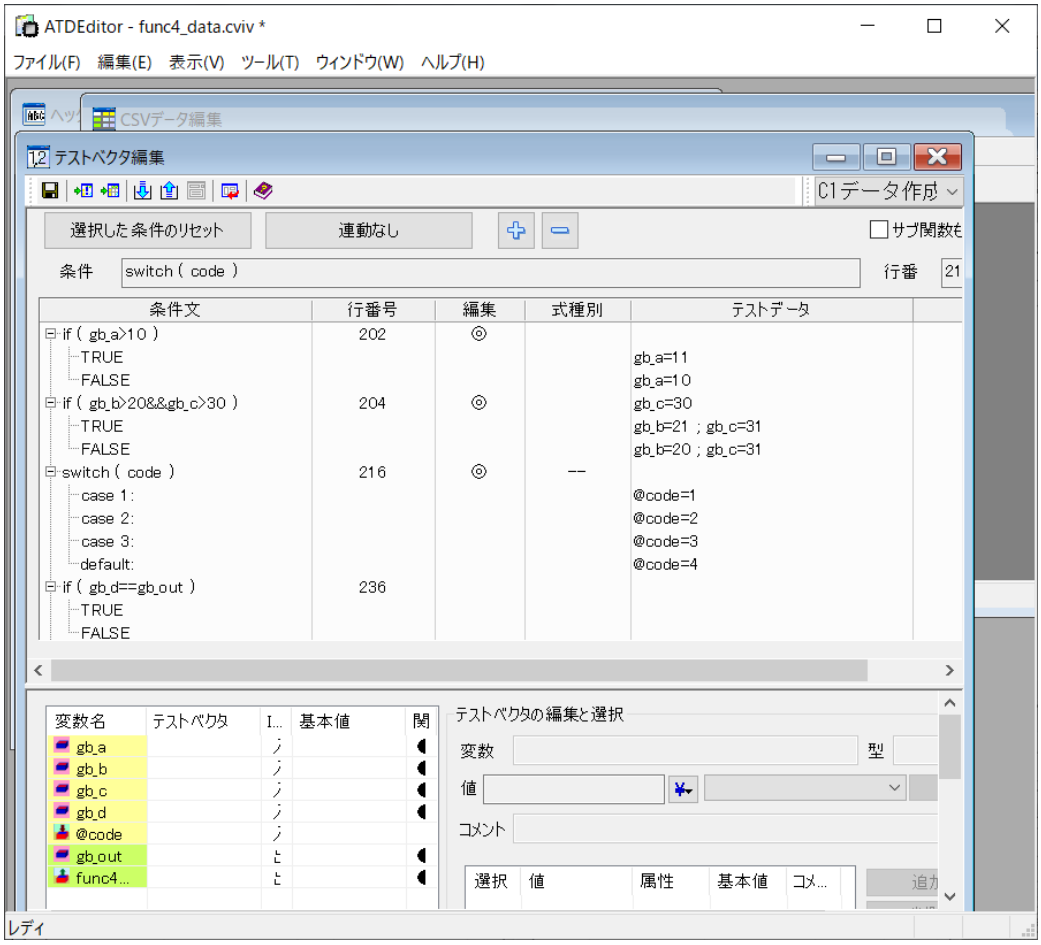
1. 「テスト設定」ビューの「func4_data.csv」リストを選択した状態で、右側の「データ入力…」ボタンを押します。

ATDEditor が開きます。これは、各変数へのテストデータ設計を行った後、各変数のデータの組み合わせを生成して、最終的な CSV テスト入力ファイルを生成するエディタです。この中には、いくつかのウインドウがありますが、このなかから、「テストベクタ編集」ウインドウを使用します。

※ATDEditor が開かず、テストデータ分析エディタが開く場合の対応
メニューの「ツール」から、「テストデータ分析エディタの設定」を選択し、ポップアップウィンドウの最上部にある「モジュールテスト CSV のデータ入力にテストデータ分析エディタを使用する」のチェックボックスをオフにして下さい。テストデータ分析エディタの詳細については、「【応用編】テストデータ分析によるテストケース作成」を参照してください。



2. ATDEditor の「表示」メニューの「テストベクタ編集ウインドウ」を選択します。
   ― 「テストベクタ編集ウインドウ」にチェックが付いていることを確認します。



ATDEditor の「テストベクタ編集」ウィンドウには、4つの編集モードがあります。このモードは、右上のプルダウンボックスで切り替えることができます。

これらの機能の概要は以下の通りです。

■「C1 データ作成」モード： C1 カバレッジ入力データ作成のためのモード
・CasePlayer2 解析情報により、C1 を満たす最小限のデータを自動生成する
・各条件式ごとに、関連する変数の真偽を満たすデータを自動生成
・最小限のデータで、C1 カバレッジを満たすことができる

■「MC/DC データ作成」モード： MC/DC カバレッジ入力データ作成のためのモード
・上と同様に、MC/DC に準拠したデータを生成

■「マニュアル作成」モード： データカバレッジのためのデータを生成するモード
・CasePlayer2 解析情報を使用しない
・各変数に与えるデータは手動設定
・データの組み合わせは「デシジョンテーブル」の手法により設定した「基本値」により、組み合わせが作成

される （「ディシジョンテーブル」については、後述します。）

■「1行データ作成」モード： CSV の1行文のデータを作成するモード
・各変数には1つの値を設定可能
・組み合わせを考えず、単に1つのテストデータを作る機能

これらの機能は、切り替えて使用することが出来ます。例えば、マニュアルモードで CSV に出力した関数仕様確認のデータに、C1 カバレッジモードで作成したカバレッジテストのためのデータを追加する様な使い方です。
本実習では、関数 func4()の C1 カバレッジを満たすデータを生成することにフォーカスします。

3. 右上のプルダウンボックスから、「C1 データ作成」を選択します。

条件文	行番号	編集	式種別	テストデータ
⊟ if (gb_a>10)	202	◎		
TRUE				gb_a=11
FALSE				gb_a=10
⊟ if (gb_b>20&&gb_c>30)	204	◎		gb_c=30
TRUE				gb_b=21 ; gb_c=31
FALSE				gb_b=20 ; gb_c=31
⊟ switch (code)	216	◎	--	
case 1:				@code=1
case 2:				@code=2
case 3:				@code=3
default:				@code=4
⊟ if (gb_d==gb_out)	236			
TRUE				
FALSE				
その他	---			

このビューには、テスト対象の func4()に含まれる条件分岐がリストアップされます。C1 カバレッジは、「全ての条件分岐を少なくとも1度は実行してテストすること」が要件であるため、このビューでは、各 if 文の TRUE/FALSE ケース、switch 文の全 case の条件を満たすための、テストデータを設計します。

このビューでは、互いの条件文のネスト状態（階層状態）には触れず、各々独立した条件文として捉え、各条件文だけに着目して、その条件を満たすための変数条件を設計します。例えば2番目の「if(gb_b>20 && gb_c>30)」の条件文は、実際は上の if(gb_a >10)の条件文の TRUE ケースにネストしていますが、ここでは gb_a の条件には触れず、単に if(gb_b>20 && gb_c>30)の中の条件のみに着目して設定を行います。

ネスト状態については、CasePlayer2 が別に解析しており、ここで各条件文に設定されたテストデータを使用して、ネスト状態の解析に基づいた変数データの組み合わせを自動生成する仕組みになっています。（この内容については後述します。）

自動生成されたテストデータの内容を確認

まず、このビューで最初に確認することは、CasePlayer2 の解析結果です。これは「編集」コラムに記号で示されます。「◎」が付いている条件文については、その条件文の境界値条件を CasePlayer2 が解析し、テストデータを自動設定したことを示しています。上から3つの条件文は全て CasePlayer2 が条件を自動設定しています。

しかしながら、最後の条件文「if(gb_d == gb_out)」については、自動設定が行われていません。これは、条件式の比較対象が動的に変化する2つの変数であり、この条件式の境界条件を一意的に決定できないためです。境界条件が自動設定できない場合には、「編集」欄と「テストデータ」欄には何も出力されません。この部分の条件は、ユーザー自身が決定する必要があります。

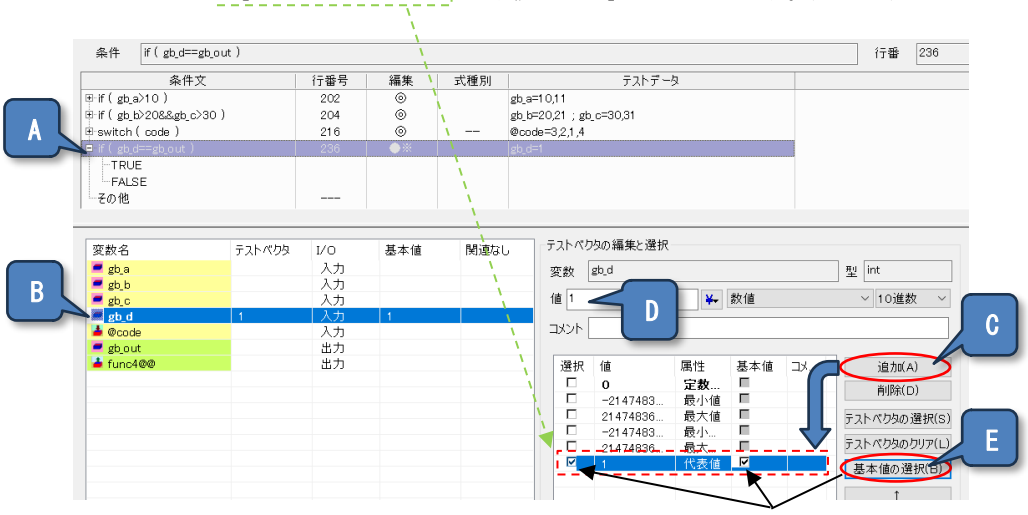
解析範囲外の条件を手動設定

では、最後の条件文「if(gb_d == gb_out)」に対して、テストデータを自分で設定します。この条件文は、外部変数 gb_d と gb_out を比較していますが、gb_out は関数内で定数値が代入されています。C1 の要件を満たすためには、変数 gb_d に値を設定して、関数内で代入された gb_out の値と比較して、TRUE と FALSE の両方を実行できるようにしなければなりません。

gb_out の値は、例えば switch 文の case 1: を通過した場合は gb_out=1 となり、case 2: を通過した場合は gb_out=2 になります。ならば、gb_d が常に 1 になるようにしておけば、switch 文の case 1: を通過した場合は条件文が TRUE となり、case 2: を通過した場合は FALSE となるため、gb_d=1 の1つのデータ設定だけで C1 カバレッジを満たすことができます。

では、gb_d が常に 1 になるようにデータを設定します。

1. if(gb_d == gb_out) の条件式を選択します。(下図:A)
2. 変数リストから gb_d を選択します。(下図:B)
3. 「テストベクタの編集と選択」の欄で、追加ボタンを押します。(下図:C)
   ※「値」と「コメント」欄に入力ができるようになり、リストに1行追加されます。
4. 値のボックスに「1」を入れます。(下図:D)
5. リストに追加された「1」のデータを選択して「基本値の選択」ボタンを押します。(下図:E)



これで、全ての条件文の分岐条件が設定できました。

(参考)「ディシジョンテーブル」、「基本値」について

前図の操作 5.でチェックした「基本値」とは、データの組み合わせを作る手法である「デシジョンテーブル」の考え方に基づいています。ディシジョンテーブルとは、複数の要素の組み合わせを作る際に、全数組み合わせでなく、必要な要素のみの組み合わせを作るための組み合わせ定義手法です。

例えば、変数 a、b、c が関数に含まれており、この各々の要素のテストのために、a = 9,10,11、b = 19,20,21、c=30,31 のデータが与えられるとします。この際、これらの組み合わせを生成して、単体テストデータとして CSV に記述することになりますが、全てを組み合わせてしまうと、3 x 3 x 2 = 18 パターンのテストデータが考えられます。

もしも、これらの変数a,b,cがお互いに関連のない(依存しない)変数であれば、全ての組み合わせを作成する必要はありません。変数aに対しては、9,10,11の3つが要件となるため、このデータ値は必ず含める必要があります。変数a,b,c がお互いに関連しないのであれば、変数 a に着目したテストの際には、他の変数 b、c の値は何であってもよいことになります。

そこで、この様な組み合わせの限定を、「基本値」と言う属性で定義します。各変数のデータには、必ず1つ以上の「基本値」を設定します。例えば、以下のように基本値([ ]付きの太字)を決定したとします。

```
a:  9  [10]  11      b: 19  [20]  21      c: [30]  31
```

ディシジョンテーブルの手法は、以下のようにして組み合わせを生成します。

・a のデータ 9,10,11 に対して、他の変数 b,c は、「基本値」の属性を付けたデータとのみ組み合わせる
・b のデータ 19,20,21 に対して、他の変数 a,c は、「基本値」の属性を付けたデータとのみ組み合わせる
・c のデータ 30,31 に対して、他の変数 a,b は、「基本値」の属性を付けたデータとのみ組み合わせる

これにより、以下のテストデータの組み合わせが生成されます。

```
a:  9  10  11   10  10  10   10  10
b: 20  20  20   19  20  21   20  20
c: 30  30  30   30  30  30   30  31

赤は各変数のテストデータ  黒は基本値
```

もしも、全てのテストデータを基本値にしてしまえば、全数組み合わせが作られることになります。

このように、各変数へのデータと、基本値の属性設定により、組み合わせを限定する方法が、ディシジョンテーブル法です。

今回の実習では、gb_d のデータに「1」を設定しましたが、データは1つしかないため、このデータが「基本値」になります。

組み合わせを生成してテストデータにする

では、上で設定した条件を基にして、テストデータ(CSV ファイル)を作成します。上で行ったテストデータ設計では、互いの条件文のネスト状態(階層状態)には触れず、各々独立した条件文として捉え、各条件文だけに着目して、その条件を満たすための変数条件を設計しました。

関数の条件文を網羅実行するためには、例えば、2番目の「if(gb_b>20 && gb_c>30)」の条件文をテストするためには、上の if(gb_a >10)の条件文が TRUE ケースである必要があるため、gb_a が TRUE ケースであるデータと組み合わせを生成する必要があります。

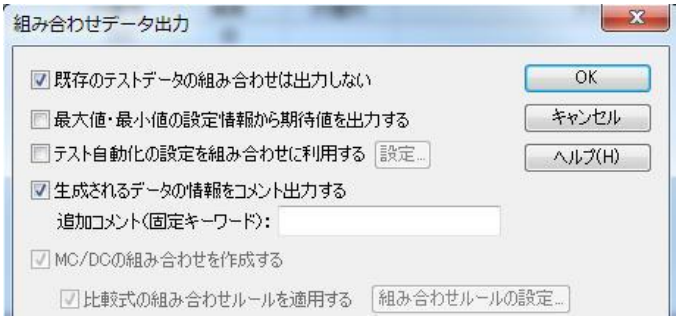
カバレッジマスターwinAMS は、CasePlayer2 で解析したネスト構造の情報を使用して、この組み合わせを自動生成する仕組みとなっています。また、全ての組み合わせを生成すると、上の 4 つの条件文の場合、

2(通り) x 2(通り) x 4(通り) x 1(通り)

の組み合わせがあり、16 通りのテストベクターになります。C1 を満たすだけであれば、同じ条件式の論理を重複して実行するテストベクターがあるため、カバレッジマスターwinAMS は、各条件文を最少回数実行する、C1 を満たすための最少数の組み合わせを生成します。

では、生成してみましょう。

1. 「編集」メニュー →「**設定して**組み合わせ出力」を選択します。
2. 「既存のテストデータの組み合わせは出力しない」をチェックします。
3. 「生成されるデータの情報をコメントを出力する」をチェックします。
4. 「OK」を押します。



組み合わせテストデータが出力され、「CSV データ編集」ウインドウが開きます。このウインドウは、データ入力、編集の機能を持った CSV エディタです。

	COMMENT	1	2	3	4	5	6	7
COMMENT								
NAME	コメント	@code	gb_a	gb_b	gb_c	gb_d	gb_out	func4@@
1		if (gb_a>10)						
2		TRUE						
3		1	11	21	31	1		
4		FALSE						
5		1	10	21	31	1		
6		if (gb_b>20&&gb_c>30)						
7		TRUE						
8		1	11	21	31	1		
9		FALSE						
10		1	11	20	31	1		
11		その他の値						
12		1	11	21	30	1		
13		switch (code)						
14		case 1:						
15		1	10	21	31	1		
16		case 2:						
17		2	10	21	31	1		
18		case 3:						
19		3	10	21	31	1		
20		default:						
21		4	10	21	31	1		
22		if (gb_d==gb_out)						
23		その他の値						
24		4	10	21	31	1		
25		その他の組み合わせ						

生成されたテストデータの組み合わせは7パターンであることが分かります。また、組み合わせ生成のダイアログで指定した「コメントを出力する」のオプションにより、生成された各データが、どの条件式のどの論理をテストするデー

タであるかの情報が併せて出力されます。

グリーンで塗られたエリアは、データとして扱われないコメントエリアを示していますが、8行目のデータは、このコメントエリアが指定されています。これは、3行目の最初のデータと同じ、重複した組み合わせのデータであるため、組み合わせ出力時に指定した「既存のテストデータの組み合わせは出力しない」のオプションにより、テストデータから外されたためです。

生成されたテストデータでカバレッジテストを実行

では、この CSV ファイルを保存して、単体テスト実行を行い、カバレッジを確認します。

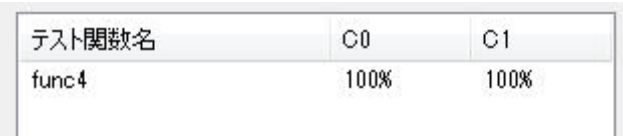
1. 「CSV データ編集」ウインドウを選択して、「ファイル」メニューから「上書き保存」を選択します。
2. 「ファイル」メニューから「アプリケーション終了」を選択します。
3. 保存を促すダイアログで、「OK」を押します （Func4_data.csv が保存される）。

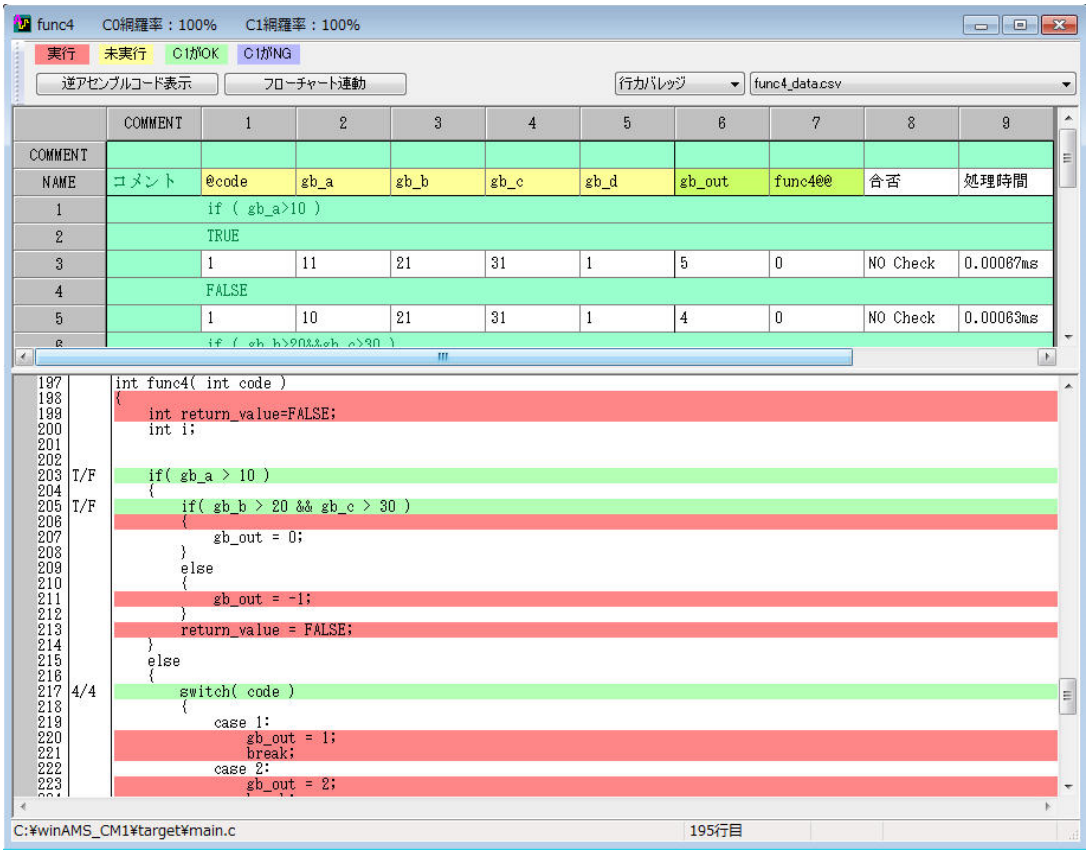
では、SSTManager のテスト設定ビューに戻って、テスト実行します。操作内容は実習1～3と同じですが、今回は CasePlayer2 の解析情報を取り込んでいるため、C1 カバレッジが出力可能になっています。

4. SSTManager の「テスト設定」で、「func4_data.csv」のチェックボックスを ON にします。
5. 「カバレッジ」設定の、「C1 カバレッジ」を ON にします。
6. 「シミュレータ起動」ボタンを押して、テスト実行します。

カバレッジ結果を確認する

カバレッジビューで、C0、C1 カバレッジ結果を確認してみましょう。両方とも、100%になり、生成された7つのテストデーターで、カバレッジが満たされていることがわかります。

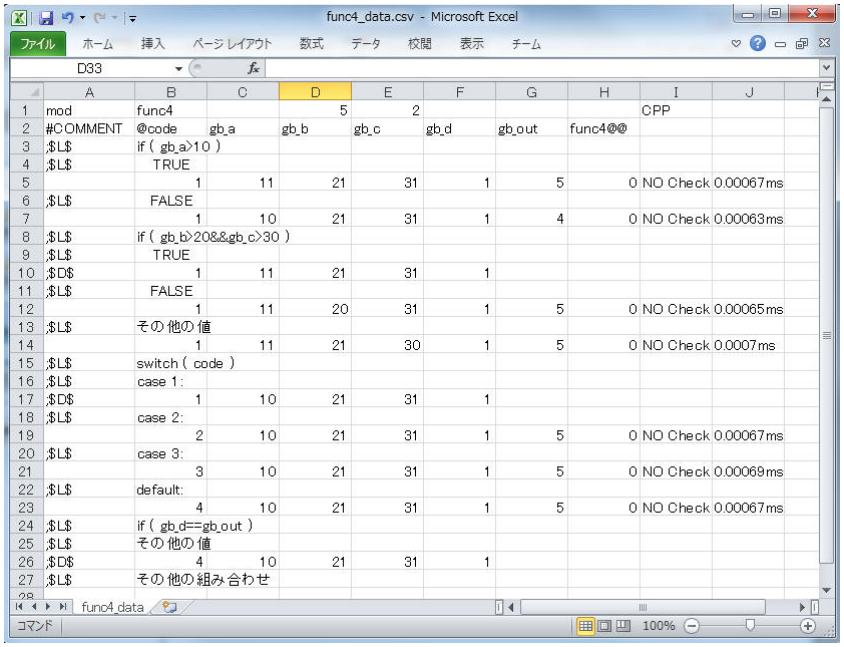




カバレッジビューでは、上のように、各条件文の C1 実行結果が表示されます。「T/F」は、if 文の TRUE と FALSE のケースが両方実行されたことを示しています。もしも、TRUE ケースが未実行であれば、「 /F」の表示になります。
switch 文については、「4/4」の様に、[実行ケース数]/[全ケース数]で示されます。

入出力結果の CSV ファイルも併せて確認してみます。

1. 「テスト結果」ビューで、「func4_data.csv」をダブルクリックして Excel で開きます。



内蔵の CSV データ編集ウインドウでは、コメント欄などが色付けされ、分かり易く表示されていましたが、実際の CSV ファイルは上のようになっています。検証箇所を示すソースコードの抜粋や、重複データの除外などは、「#COMMENT」のコラムの機能を使用して出力されていることが分かります。

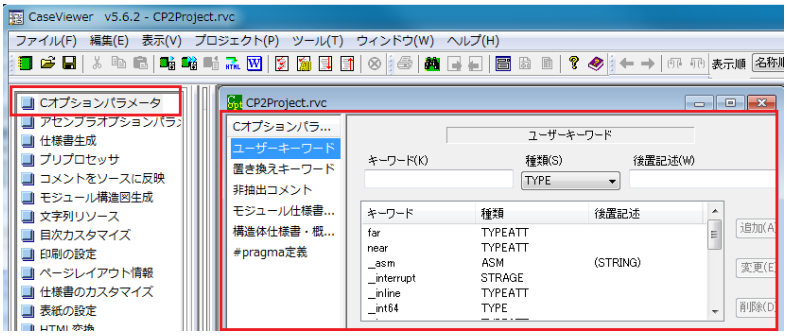
（参考） CSV ファイルにコメントを記載する場合、先頭のセルに「;(セミコロン)」を入れることで、その行をコメント行とすることができます。カバレッジマスターwinAMS の CSV 編集ウインドウで CSV ファイルを作成した場合は、コメントの属性によって L の様な記号が追加され、区別できるようになっています。フォーマットは以下の通りです。

コメントの種類	フォーマット
種類が不明のコメント行	;, あるいは、ヘッダ2の先頭が＃COMMENT ではない
コメント行	;L,
実行しないデータ行	;D,
データのコメント行	;C,

（参考）C オプションパラメータの設定について

「C オプションパラメータ」は、開発に使用しているクロスコンパイラ独自の方言に対応するためのものです。CasePlayer2 の解析エンジンは、指定した言語仕様（ANSI-C、C99、GNU-C）に基づいてソースの解析を行いますが、クロスコンパイラ独自の言語拡張（方言）には対応できす、そのままでは解析エラーとなります。そこで、「C オプションパラメータ」に方言のキーワードとその解釈方法を設定することで、ソースコードを書き換えることなく、正しく解析できる様になります。

CasePlayer2 のプロジェクト新規作成時に、「ターゲット CPU 及びコンパイラ固有の設定」で使用するマイコンとコンパイラを選択しておけば、「C オプションパラメータ」は自動設定されます。各コンパイラの標準的な記述であれば、自動設定されるパラメータで、方言を含むソースは正しく解析できます。ただし、コンパイラのバージョンアップや仕様拡張などで、新たな記述やキーワードが追加された場合は、ユーザーが「Cオプションパラメータ」に設定を加えることで、対応が可能です。



「C オプションパラメータ」で主に使用する機能は、「ユーザーキーワード」と「置き換えキーワード」です。代表的な設定方法には、以下のようなものがあります。

■ユーザーキーワードの設定例

記述例1： int near p1; ←near がエラーとなる場合

near は「型修飾子」です。これを型修飾子として認識させるために、near のキーワードに「TYPEATT」指定します。

キーワード:near、 種類:TYPEATT、 後置記述:[空白]

記述例2： direct int array[100]; ←direct がエラーとなる場合
direct は「記憶クラス」です。これを記憶クラスとして認識させるために、direct のキーワードに「STRAGE」指定します。

キーワード:direct、 種類:STRAGE、 後置記述:[空白]

記述例3： _asm (“ ..... “) ←_asm がエラーとなる場合
_asm は「インラインアセンブラ記述」です。これをインラインアセンブラ記述として認識させるために、_asm のキーワードに「ASM」指定します。また、このキーワードに続く記述が_asm の内容の記述であることを示すために、後置記述に「(EXPRESSION)」を指定します。

キーワード:_asm、 種類:ASM、 後置記述:(EXPRESSION)

記述例4： _except ←_except がエラーとなる場合
_except は例外処理を示すコンパイラ浦独自の記述文法です。この記述全体を無視させるために、_except のキーワードに「IGNORE」指定します。また、このキーワードに続く記述が_except の内容の記述であり、同時に無視させるために、後置記述に「(EXPRESSION)」を指定します。

キーワード:_except、 種類:IGNORE、 後置記述:(EXPRESSION)

■置き換えキーワードの設定例

記述例1： typedef _WCHAR_T_TYPE_ _Wchart; ←_WCHAR_T_TYPE_がエラーとなる
このエラーを回避するために、_WCHAR_T_TYPE_を int 型に置換して解析させます。このために、置き換えキーワードの機能を使います。

新キーワード:int，既にあるキーワード _WCHAR_T_TYPE_

記述例2： typedef _SIZE_T_TYPE_ _Sizet; ←_SIZE_T_TYPE_がエラーとなる
このエラーを回避するために、_SIZE_T_TYPE_を int 型に置換して解析させます。このために、置き換えキーワードの機能を使います。

新キーワード:int，既にあるキーワード _SIZE_T_TYPE_

まとめ

以上で、カバレッジマスターwinAMS の基本操作に関する実習は終了です。

本チュートリアルでは、

- テストデータを CSV ファイルで用意さえすれば、テスト作業自体は自動化されること
- ポインタ変数を含む関数のテストにおいて、ポインタの実体を自動割り付けする機能が利用できること
- スタブ関数の作成機能がサポートされており、スタブ関数の入れ替え実行が、ソースの書き換え無しで行えること
- C1カバレッジを満たすテストデータが、ソースコードから容易に生成できること

を体験しました。

基礎的な使用方法については、ご理解頂けたと思います。今回体験した内容を基本に、皆様の実開発業務にお役立て下さい。

【応用編】テストデータ分析によるテストケース作成

はじめに

カバレッジマスターwinAMS には、実習4で使用したテストデータ自動生成機能に加えて、関数の機能を確認するために、関数設計仕様を基にしたテストケースを作成する際の支援機能がサポートされています。

この応用編では、「テストデータ分析エディタ」を使用して、効率よく仕様ベースのテストケースを作成し、レビューを行う機能について学びます。

ユニットテストデータ分析とは

単体テスト受託サービスの手法、ノウハウをツールに実装

ガイオは長年にわたり、単体テストの受託作業サービスを行っています。このサービスでは、お客様から提示された関数設計仕様書やソースコードから、ブラックボックス、ホワイトボックスの観点で、テスト設計を行い、テスト実行から得られる結果を評価、納品する業務を行っています。

この際に、関数の入出力に対して、入出力変数とそれに与えるテストデータ値を記載した「入出力分析表」と呼ぶ一覧表を作成し、テスト項目の漏れや付与値の妥当性を確認した上で、テストケースを作成しています。

通常、単体テストのテストケース設計においては、予め決められた設計手順書に従って、単体テストに必要なテスト要素、「境界値」、「最大/最小値」、「同値クラス毎のテスト値」などを仕様情報から抽出し、これらを漏れなくテストできる様に組合せを行いテストケースを生成します。

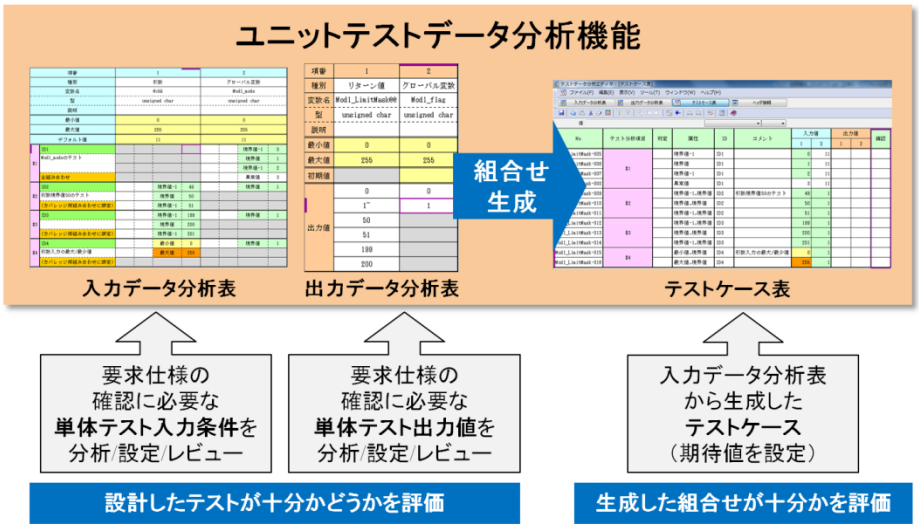
「ユニットテストデータ分析」とは、ガイオの単体テスト受託サービスで培ったノウハウを基に、単体テストの作業フローを手法化したものです。カバレッジマスターwinAMS には、この手法を使って効率的にテストケースを設計し、関数仕様との照らし合わせを行いながら設計したテストデータの妥当性を確認するための支援機能、「テストデータ分析エディタ」が搭載されています。

テスト設計のレビュー、クロスチェックを容易にするテストデータ分析表

カバレッジマスターwinAMS のユニットテストデータ分析手法による単体テスト設計では、関数に与えるテストケースを作成する前に、「入力データ分析表」により、テスト入力データの妥当性を確認する方法を採っています。要求仕様に基づいたテスト項目毎に各入力変数に与えるテストデータを一覧表にまとめ、入力データが十分であるかの確認、評価を行った後で、このデータを漏れなく組み合わせて、関数に与えるテストケースを生成します。

さらに、出力値に対するテスト設計を「出力データ分析表」により行います。関数を実行した時に得られなければならない出力値（期待値）を確認し、この表にまとめます。設計した入力テストケースの毎の期待値が、この出力データ分析表に抽出した出力値を網羅しているかを評価し、テストの十分さを確認します。

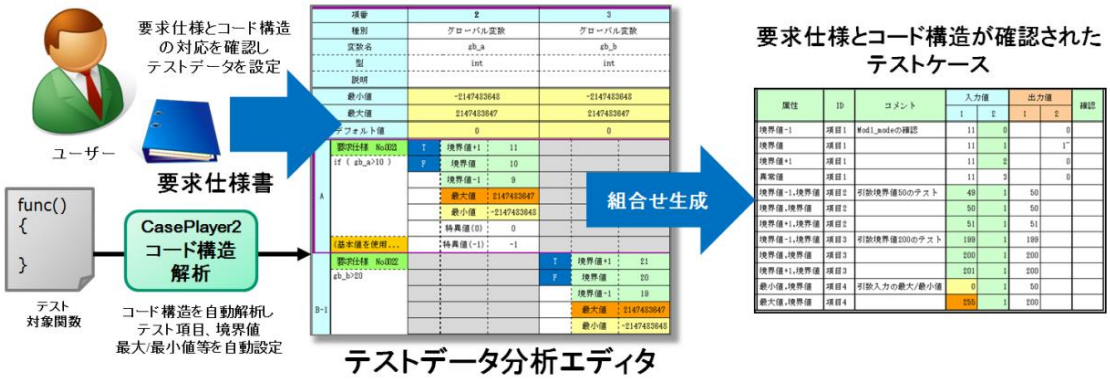
この手法により、要求仕様の確認に必要な単体テストのテスト項目、入力/出力条件を明確化し、テスト設計のレビュー、妥当性の確認を容易にします。



コード構造と要求仕様を照らし合わせながら効率的にテストケースを設計、評価

本来、仕様ベースのテストとは、定義された関数仕様通りに関数が動作するかを確認するテストであるため、そのテスト設計においては、仕様書を読み解きながら、関数の動作確認に必要なテスト項目を仕様情報から抽出して、テストケースにすることが必要です。ただし、この作業は、設計のレビューまでを含めると、非常に大きな工数を必要とします。

この作業を効率化するために、本章で扱うテストデータ分析エディタには、CasePlayer2 で解析したコード構造情報を基に「テスト分析項目」に分割し、効率的にテスト設計に利用する機能がサポートされています。

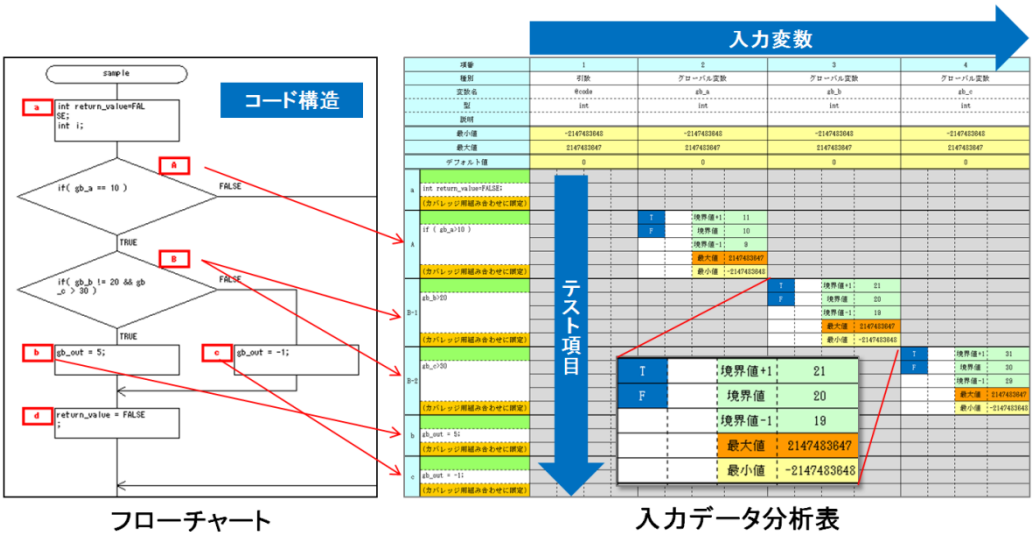


テスト分析項目とテストデータの自動抽出により テストデータ設計を効率化

ユニットテストデータ分析機能は、最初に CasePlayer2 のコード解析機能によりコード構造を分解し、「入力データ分析表」にコード構造に対応した「テスト分析項目」欄を自動作成します。ユーザーは、生成された「テスト分析項目」欄と、要求仕様項目との対応を確認することで、ソースコード構造の妥当性を評価することができます。この際に、「テスト分析項目」からソースコードを確認したり、表と連動する CasePlayer2 のフローチャートで構造の確認を行ったりすることができます。

仕様項目とコード構造の対応が確認できたら、次に「テスト分析項目」毎に詳細な要求仕様を確認するためのテストデータを設定します。分岐構造部分については、CasePlayer2 のコード解析機能により、分岐条件に係わる変数や境界条件を解析し、単体テストの要因となる境界値、最大最小値、特異値などの値を自動抽出します。ユーザーは、自動抽出されたテストデータで、「テスト分析項目」の確認が可能かどうかを評価し、過不足や誤りがあれば、これを修正することで、テスト入力データを効率的に設計することが出来ます。

これらにより、要求仕様の確認である「ブラックボックステスト」と、コード構造の確認である「ホワイトボックステスト」の両面を効率的に行う事が出来ます。



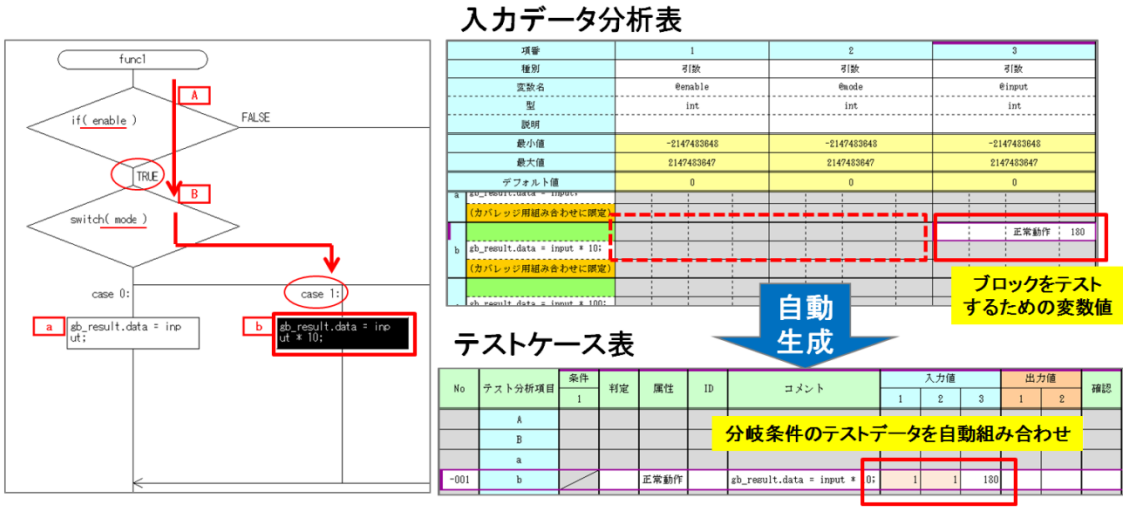
入力データ分析表からテストケースを自動生成

「入力データ分析表」で入力テスト条件の妥当性、網羅性を確認した後で、設定したテストデータを組み合わせて関数に与えるテストケースを生成します。

組み合わせを作成する上でテスト設計者の頭を悩ませるポイントの1つは、分岐のネスト構造の把握と、そのための条件の組み合わせです。例えば、下図のフローチャートの構造の関数の場合で、赤の処理ブロック「b」の動作をテストする場合、この処理が行われる条件は、分岐「A」の「変数 enable = TRUE」、かつ分岐「B」の「変数 mode = 1」が成り立つことが必要です。このテスト設計を手作業で行う場合、分岐条件とブロックをテストするための他の変数値を組み合わせてテストケースにする必要があります。

「テストデータ分析エディタ」は、ソースコードのネスト構造に基づく条件と、「テスト分析項目」に入力されたテストデータを管理し、自動的にテストデータの組み合わせを生成する機能を持っています。下の例の場合、ブロック「b」をテストするための入力条件「input = 100」を入力するだけで、このブロックに分岐するための条件（enable = TRUE、mode = 1）を自動的に組み合わせてテストケースに生成します。

これにより、ユーザーは、煩わしい条件の組み合わせ設計から解放され、意図する条件下のテストケースを短時間で設計することが出来ます。

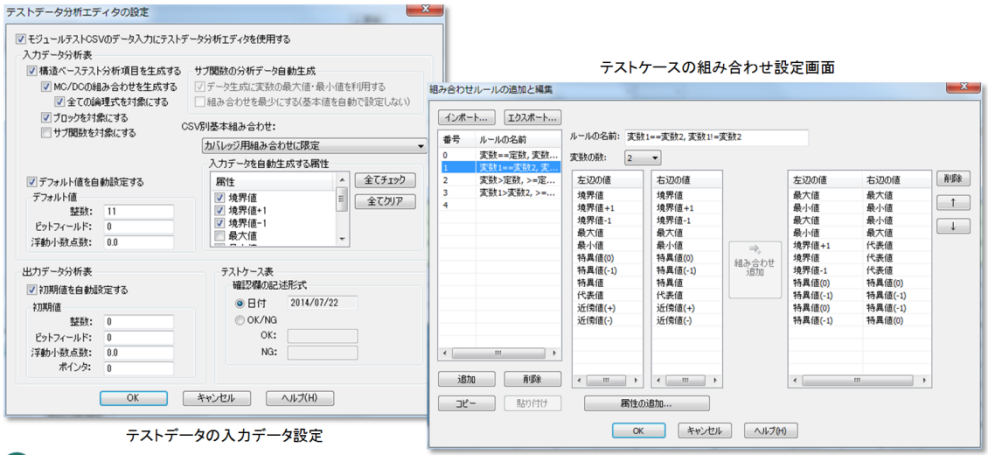


テスト設計の粒度を標準化するための 設計ルールを指定

要求仕様から単体テスト設計を行う場合、設計の方法は自由度が高いため、その粒度は担当者によって異なってしまいます。開発プロジェクトの単体テストを複数の担当者が行う場合は特に、テストデータ設計の方法を手順書のようなマニュアルにまとめ、担当者のスキルに依存しない標準的な設計を可能にすることが重要です。

テストデータ分析エディタには、関数に与えるテストデータの設計ルール設定する機能があります。単体テストの一般的なテストケース導出手法である、境界値、最大/最小値、同値分割による代表値、0、-1 等の特異点等のデータに対し、どのデータをデフォルトで与えるかをルールとして決めて置くことで、テスト担当者に依存したデータ抽出の粒度のばらつきやテスト項目の抜け漏れの発生を防ぎ、担当者間で標準化されたテストデータ設計を可能にします。

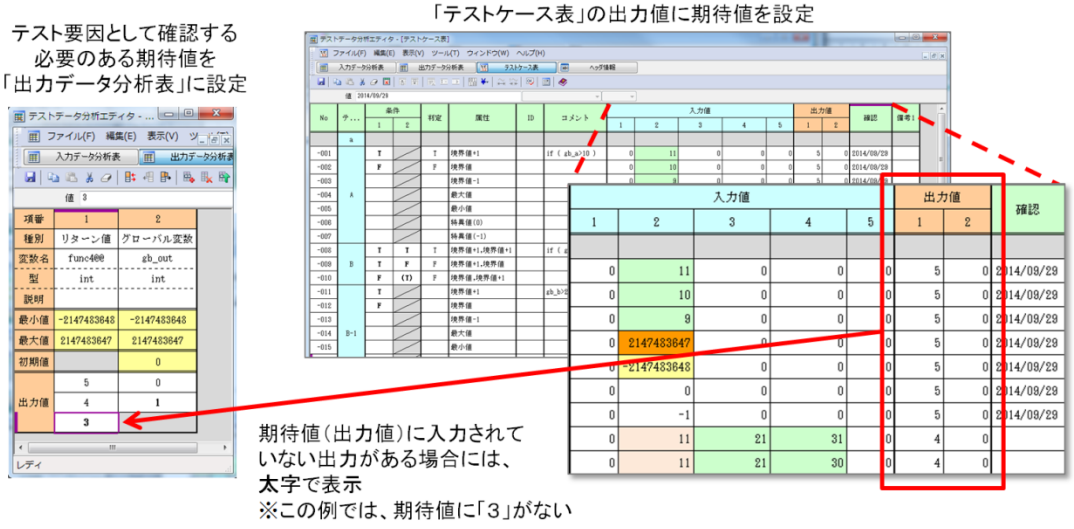
また、入力データ分析表に設定したテストデータからテストケースの組み合わせを生成する際には、どの属性のデータを組み合わせるかのルールをカスタマイズして設定する機能もサポートされています。



出力データ分析表による期待値設計確認

さらに、テスト設計の精度を高めたい場合には、「出力データ分析表」を使用して、期待値に対する設計確認を行う事ができます。この出力データ分析表には、要求仕様を確認する際に、入力条件を設定した結果、出力として確認する必要があるデータを予め抽出します。

例えば、5、4、3の3通りの出力を持つ仕様の関数がある場合、この仕様を網羅するためには、これらの3つの値それぞれが出力される入力条件が設定されているかを確認する必要があります。この場合は、「出力データ分析表」に、5、4、3の出力を設定しておき、組み合わせ生成したテストケースの期待値（出力値）で、この3つの値が網羅されるかを確認します。「出力データ分析表」の中に、テストケースの期待値に設定されていないデータがある場合、これを検出し太字にハイライトして表示することができます。これにより、期待値に基づいたテスト条件の漏れを検出できます。



実習5：テストデータ分析エディタで func5()のテスト設計を行う

では、テストデータ分析エディタを使用して、1つの関数のテスト設計を行って見ましょう。この実習では、次の内容を関数 func5()を使用して実施します。※この実習を行うためには、前章の実習 4 が終了していることが必要です。

- ソース構造に基づいた入力データ分析表の設定と確認
- 特定条件のテスト分析項目の設定と追加
- テストケースの組み合わせ生成
- 出力データ分析表の設定と期待値の設定

テスト対象関数の要求仕様とソースコードを確認

最初に、実習で使用する関数の仕様とコードを確認します。まず、関数 func5()の仕様条件は以下の様に定義されているとします。

```
--------------------------------------------------------------------------
< 関数 func5 設計仕様 >

2つの入力 input1、input2 に対し、モード切替値(mode)によって値を決定し、
gb_result.data に出力する。なお、機能全体の ON/OFF は入力フラグ enable で決定する。

入力:(全て引数)
    enable (フラグ): 機能の ON/OFF OFF の場合  出力 gb_result→(0, FALSE)
    mode (変数)      : モード切替 0: 出力 gb_result→(input1, TRUE)
                                  1: 出力 gb_result→(input2, TRUE)
                                  2: 出力 gb_result→(input1+input2, TRUE)
                                  default: 出力 gb_result→(255, TRUE)
    input1(変数)     : 入力1 レンジ(0～150)
    input2(変数)     : 入力2 レンジ(0～150)

出力:(グローバル構造体)
    gb_result.data : 出力データ
    gb_result.ret_code : エラーコード(機能 ON の時 TRUE、それ以外 FALSE)
--------------------------------------------------------------------------
```

この仕様に基づき、以下のコードが作成されたとします。

```
--------------------------------------------------------------------------
void func5( int enable, int mode, unsigned char input1, unsigned char input2 )
{
    if( enable )
    {
            switch( mode )
            {
            case 0:
                    gb_result.data = input1;
                    break;
            case 1:
                    gb_result.data = input2;
                    break;
            case 2:
                    gb_result.data = input1 + input2;
                    break;
            default:
                    gb_result.data = 255;
                    break;
            }
            gb_result.ret_code = TRUE;
    }
    else
    {
            gb_result.data = 0;
            gb_result.ret_code = FALSE;
    }
}
--------------------------------------------------------------------------
```

上記の func5()は、前章までの実習サンプルで使用した main.c に含まれており、実習1でコンパイルしたオブジェクトコードに含まれています。また実習 4 で CasePlayer2 による解析も行われていますので、これをそのまま使用します。

要求仕様に基づいたテスト分析項目の確認

本実習では、関数 func5 の要求仕様を確認するために、以下の5つの要求仕様項目を設定し、項番を付けたとします。これから、この要求仕様をテストするためのテストデータ設計を、テストデータ分析エディタを使用して行います。

＜要求仕様サンプル＞

要求仕様 001: enable フラグで機能全体が切り替わること
要求仕様 002: enable が OFF(FALSE)の場合、出力 gb_result→(0, FALSE)であること
要求仕様 003: enable が ON(TRUE)の場合、出力 gb_resul.ret_code→TRUE であること
要求仕様 004: mode でモード切替が行われること
要求仕様 005: mode が 0,1,2 以外の値で出力 gb_result.data→255 に固定されること
要求仕様 006: モード 0 で出力 gb_result.data に input1 が選択されること
要求仕様 007: モード 1 で出力 gb_result.data に input2 が選択されること
要求仕様 008: モード 2 で出力 gb_result.data に input1、input2 の加算値が出力されること

テスト指針(設計ルール)の確認

本実習では、以下の様なテスト指針(設計ルール)を設定します。実際の運用においては、より詳細かつ具体的なテスト指針が設定されますが、今回はサンプルとして、簡単なルールのみを扱います。

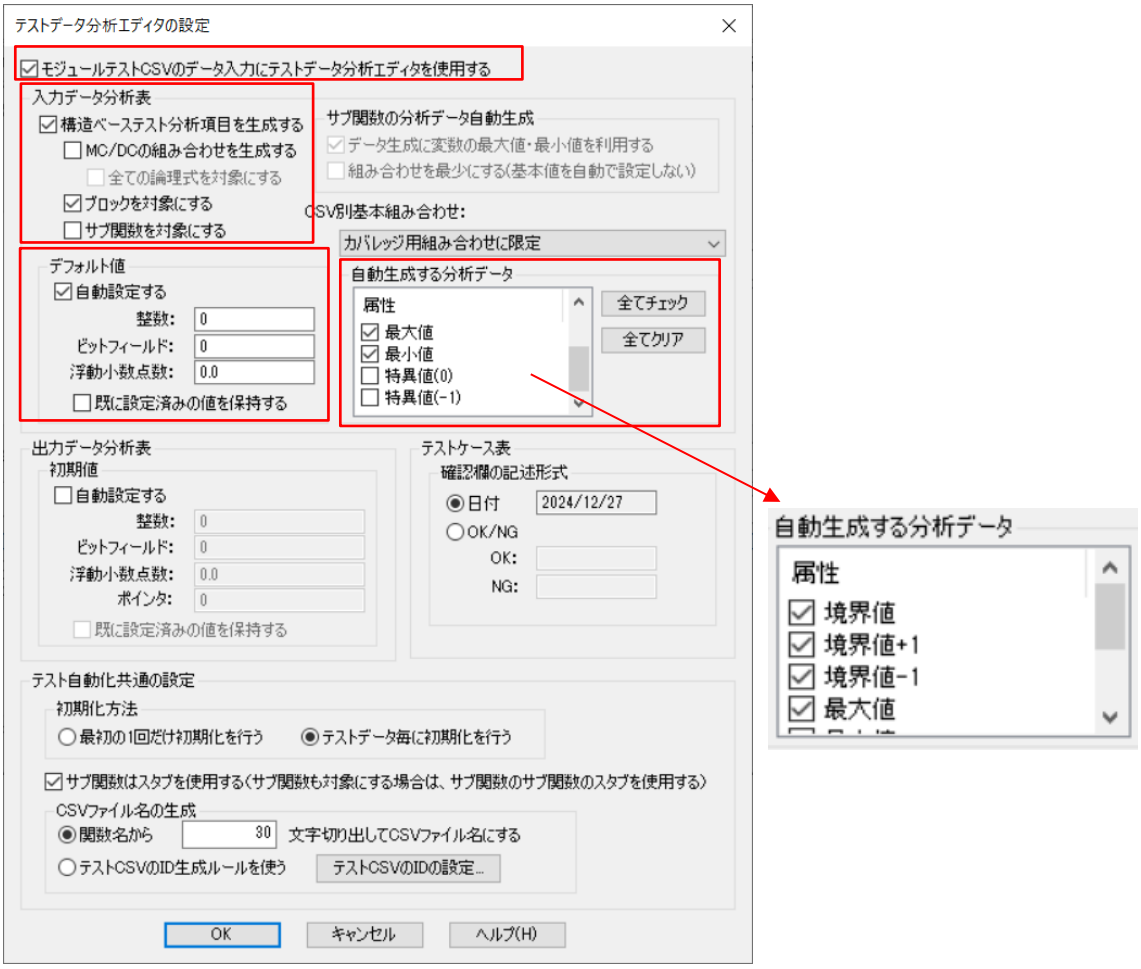
＜テスト指針サンプル＞

指針 1: フラグには TRUE/FALSE のみを付与
指針 2: 通常変数には 必要なデータと最大値/最小値を付加
指針 3: レンジ指定のある変数にはレンジの最大値/最小値/中間値を付与
指針 4: レンジ指定のない変数には型の最大値/最小値を付与
指針 5: 演算部には、最大値/最小値によるオーバーフローを確認

テストデータ分析エディタの設定

テストデータ分析エディタを使用するための設定を行います。

1. SSTManager の「ツール」メニュー →「テストデータ分析エディタの設定...」を選択します。
2. 下図の様にチェックボックスを設定します。
3. 「CSV 別基本組み合わせ」→「カバレッジ用組み合わせに限定」を選択します。
4. 「入力データを自動生成する属性」で、下図のオプションにチェックします。
5. 「OK」で設定を閉じます。



［オプションについて］

□モジュールテスト CSV のデータ入力にテストデータ分析エディタを使用する

これは、「テストデータ分析エディタ」を使用するためのメインオプションです。このチェックボックスが ON の状態で、「テスト設定」ビューや「モジュールテスト用 CSV 雛形作成」画面で「データ入力」ボタンを押した際に、「テストデータ分析エディタ」が起動します。このチェックボックスが OFF の場合は、実習 4 で使用した「ATDEditor」が起動します。

＜注意点＞

また、「テストデータ分析エディタ」と「ATDEditor」の選択は、CSV ファイルにフラグとして記録されています。作成済みの CSV ファイルを選択して「データ入力」のボタンを押した際には、先頭行の M 列(13 列目)に「1」が記述されている場合には「テストデータ分析エディタ」、無記入あるいは「0」が記述されている場合には、「ATDEditor」が起動します。

□構造ベーステスト分析表項目を生成する

これは、コード構造を分解し、「テスト分析項目」欄を自動生成するオプションです。これが OFF の場合は、テスト分析項目は生成されず、全てマニュアルで分析項目設定を行う必要があります。コード構造を参照せず、全て仕様情報のみからマニュアルでテスト分析項目を設定する場合に OFF にします。「テストデータ分析エディタ」を使用する際の推奨設定は、ON です。

□MC/DC の組み合わせを生成する

このオプションは、複合条件式を持つ分岐に対して、MC/DC を満たすテストケースの組み合わせを生成するためのスイッチです。MC/DC 計測を適用している場合に使用します。本実習では、OFF で使用します。

□ブロックを対象にする

これは、コード構造を分解してテスト分析項目欄を自動生成する際に、分岐以外の処理ブロック(フローチャートの長方形)をテスト分析項目として抽出する場合に ON にします。このオプションで生成したテスト分析項目(処理ブロック)に分岐するための条件の組み合わせは、自動解析されます。「テストデータ分析エディタ」を使用する際の推奨設定は、ON です。

ただし、実習 4 で行った様な、分岐のみに着目をして、全てのパスを実行するテストに特化したい場合には、OFF を適用します。この場合は、分岐部分のみがテスト分析項目に抽出されます。

□サブ関数を対象にする

これは、実際のサブ関数を含めた結合状態で単体テストを行う場合に、サブ関数の入出力条件を含めてテストデータを作成したい場合に使用します。ただし、このサブ関数の適用範囲は、テスト対象の関数を基点にした1階層下のサブ関数までです。通常の単体テストではサブ関数はスタブ化されるため、実際のサブ関数はテストに使用されません。そのため通常は OFF で使用します。

□自動設定する

これは、テストデータ分析表を作成後にデータの組み合わせを行いテストケース表を生成した際に、テストデータ分析表でデータを設定しなかった変数に対して、組み合わせに使用するテストデータのデフォルト値を設定するオプションです。

□自動生成する分析データ

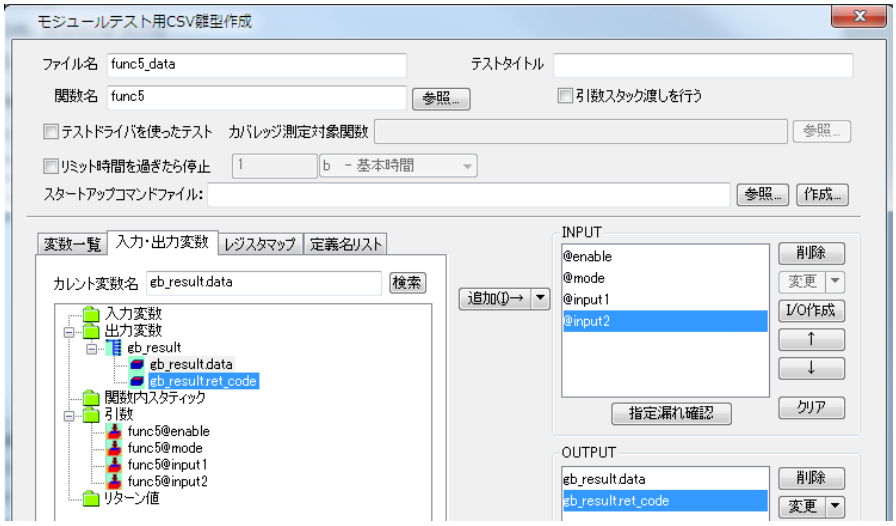
これは、条件分岐に関連する変数として抽出された変数に対して、デフォルトで作成するデータの種類を指定するオプションです。テスト指針(設計ルール)に従って選択します。本実習では、特異値以外の全ての属性のデータをデフォルトにします。

その他のオプションは、本実習では使用しません。詳細はヘルプマニュアルを参照してください。

func5()のテスト CSV を作成

では、func5()のテスト CSV ファイル作成に進みます。CSV ファイルの作成手順は、実習4と同じです。

1. SSTManager の「テスト CSV 作成」ボタン押して、「モジュールテスト用 CSV」を選択します。
2. テストタイトル、ファイル名(func5_data)、関数名(func5)を設定します。
3. 変数リスト欄の「入力・出力変数」タブから、func5()の入出力条件を設定します。
   INPUT: @enable, @mode, @input1, @input2
   OUTPUT: gb_result.data, gb_result.ret_code
4. 「OK」ボタンを押して、一旦 CSV ファイルを保存します。



作成された CSV ファイルの雛形(フォーマット)を確認します。

5. 「テスト設定」ビューで、テスト CSV 一覧から func5_data.csv をダブルクリックします。
6. Excel で開いた CSV ファイルを確認します。

入出力変数名が入力されたテスト CSV ファイルが作成されています。先頭行の M 列(13 列名)に記述されている「1」が、「テストデータ分析エディタ」を使用するモードであることを示しています。無記入の場合や「0」が記述されている場合は、「テストデータ分析エディタ」は使用されず、実習 4 で使用した「ATDEditor」が使用されます。

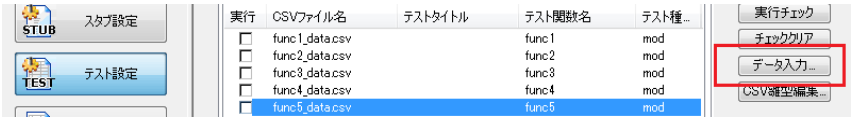
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	mod	func5		4	2				CPP				1
2	#COMMENT	@enable	@mode	@input1	@input2	gb_result.data	gb_result.ret_code						
3													
4													
5													
6													

7. 確認後 CSV ファイルを閉じます。

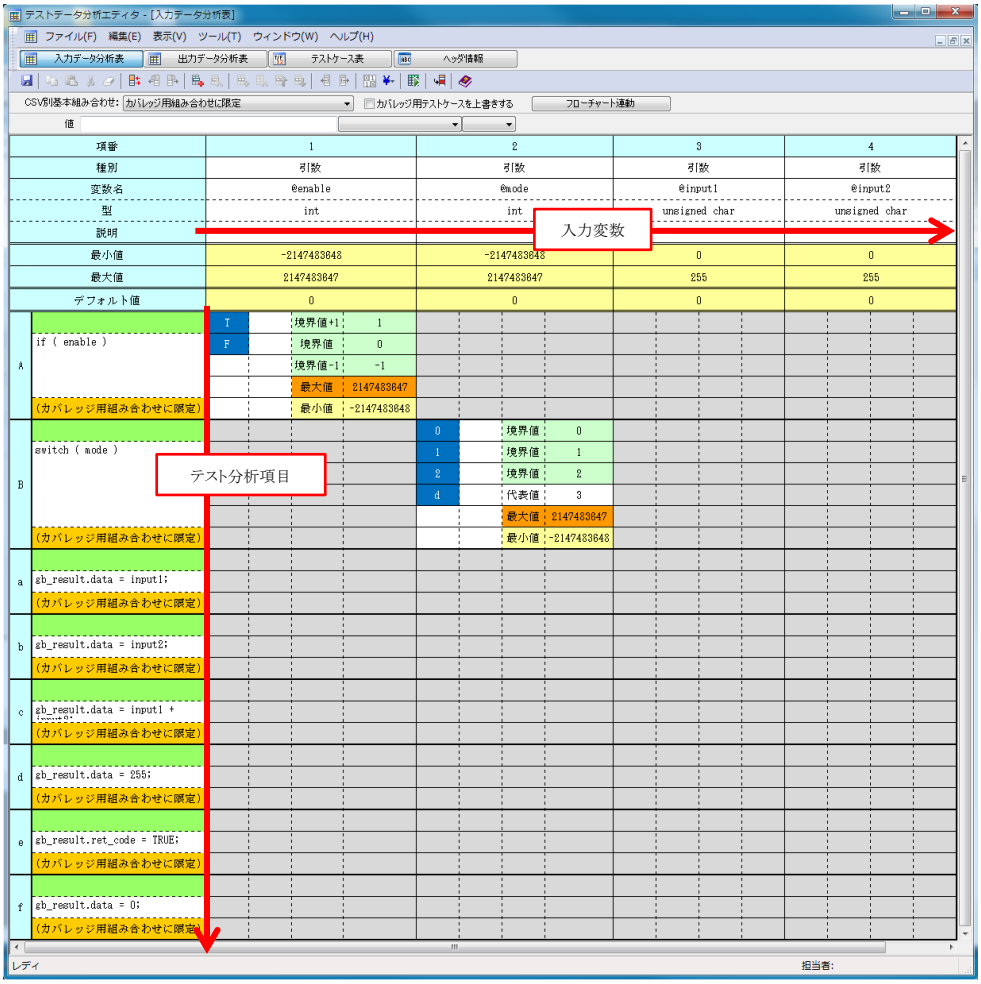
テストデータ分析エディタを起動

では、テストデータ分析エディタを起動します。

1. 「テスト設定」ビューで、テスト CSV 一覧から func5_data.csv を選択し、「データ入力」ボタンを押します。



この際に、「テストデータ分析エディタの設定」で行った設定が反映され、「入力データ分析表」が自動作成されます。「入力データ分析表」は、横方向が「入力変数」、縦方向がコードを分解した「テスト分析項目」欄のマトリクスになっています。



まず、横方向の入力変数について説明します。これらの変数欄には CSV 雛形作成で選択した変数が自動入力されます。また、引数、グローバル変数などの種別、型、最大値/最小値、「テストデータ分析エディタの設定」で設定したデフォルト値などの情報が自動入力されます。最大値/最小値には、変数の仕様上の最大値/最小値を設定しておくことも出来ます。

先頭号の「項番」は、この後に使用する「テストケース表」等の変数 ID として使用されます。(テストケース表などでは、変数名の代わりに、この項番が表示されます。)

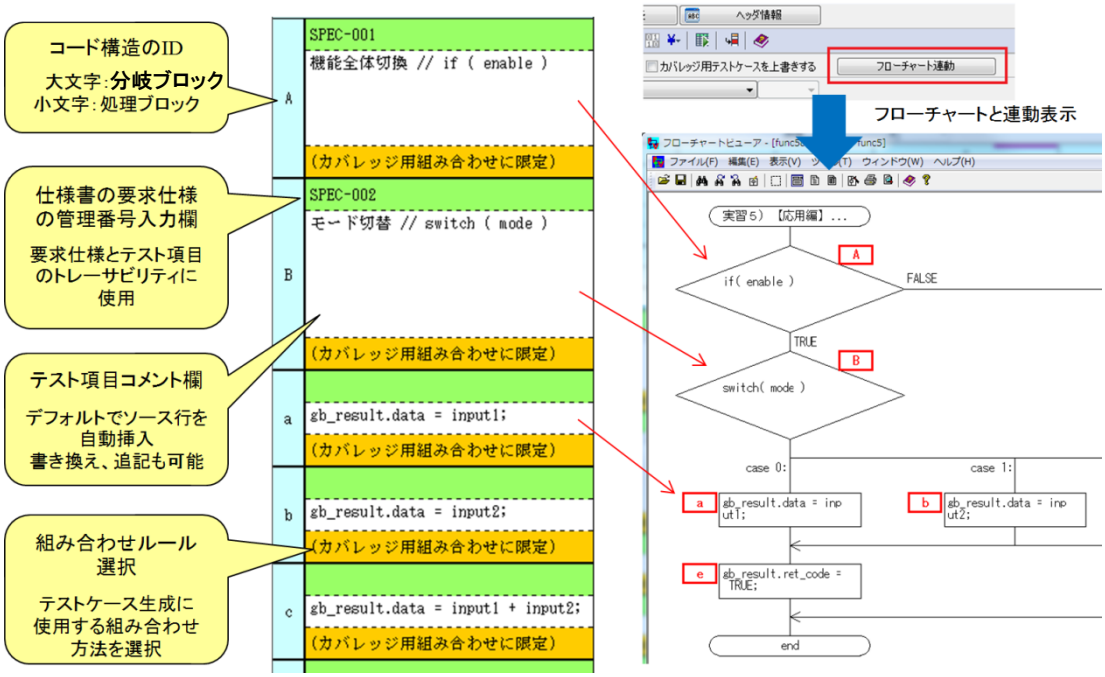
項番	1
種別	引数
変数名	@enable
型	int
説明	
最小値	-2147483648
最大値	2147483647
デフォルト値	0

次に、縦方向の「テスト分析項目」欄について説明します。テスト分析項目欄は、ソース構造が分解され、これに基づいて自動作成されます。左端の記号は分解されたテスト分析項目の ID です。大文字が「分岐ブロック」を示しており、小文字は「処理ブロック」を示します。エディタ上部の「フローチャート連動」ボタンを ON にすると、CasePlayer2 のフローチャートに ID が表示され、その対応を連動表示します。

緑色の行は、仕様書に付けた要求仕様の管理番号との対応に使用します。機能安全認証などで要求される、要求仕様とテスト分析項目の対応(トレーサビリティ)の管理に使用できます。

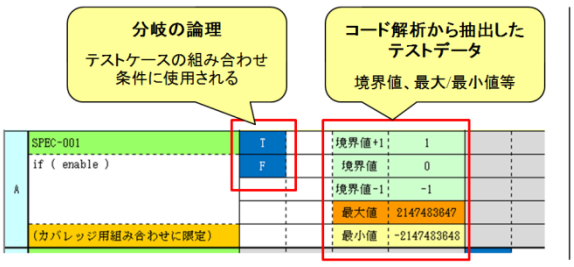
白の欄はテスト分析項目のコメント欄です。デフォルトではソースコードの抜粋が表示されていますが、書き換え、追記などの編集を行う事ができます。

オレンジ色の欄は、入力データ分析表からテストケースを自動生成する際に適用する、「組み合わせルール」の選択です。デフォルトでは、全ての分岐を実行する最小限の組み合わせを生成するルール「カバレッジ用組み合わせに限定」が適用されていますが、特定のルールで組み合わせを生成したい場合に、変更して使用します。



さらに、自動設定されるテストデータについて説明します。実習4で使用した ATDEditor の場合と同様に、CasePlayer2 の静的解析により、分岐条件に関連する変数であること、またその境界値が抽出できた変数に対して、入力データ分析表にテストデータが自動設定されます。

また、条件分岐を実行する際の論理が、青のボックスに自動設定されます。分岐のネスト構造による分岐条件の組み合わせが必要な際には、ここに設定された論理に従って組み合わせが作成されますが、例えば、上記の「B」の分岐以下のテストを行う際には、その上位の「A」の分岐はTRUEである必要があるため、「B」の分岐以下のテストケースの組み合わせ生成時には、変数 enable の TRUE のデータ、すなわち「1(境界値+1)」が組み合わせに用いられます。



右クリックでデータの挿入/削除が可能

これらのテストデータは、該当箇所を右クリックして表示されるメニューで、追加、削除などの編集が可能です。デフォルトで設定されたデータが、予め決めたテスト指針(ルール)で必要ない場合に削除(右クリック「分析データ削除」)を行う事ができます。

もしも、条件分岐の実行文(if 文、switch 文等)が静的解析できなかった場合には、分岐を示す大文字 ID のテスト分析項目に対して、テストデータは設定されません。この場合は、ユーザー自身がデータを新規作成(右クリック「分析データの挿入」)し、その条件分岐の論理を決定する必要があります。

要求仕様との対応を確認し入力データ分析表に追記する

最初に、「入力データ分析表」に自動抽出されたコード構造を基に、要求仕様との対応を確認します。ここで、要求仕様として上がった項目に対応するコード構造が入力データ分析表に無い場合には、その要求仕様が実装されていない可能性を検証しなければなりません。また、要求仕様にないコードが実装されている場合には、不要コードが残っている可能性を検証する必要があります。

このステップは、要求仕様とコード構造を照らし合わせて確認する事で、要求仕様が漏れていないこと、無駄なコード構造が実装されていないことの両面、すなわちブラックボックステストとホワイトボックステストの両面のテストを効率的に行う重要なポイントです。

まず、要求仕様として設定した項番 001～006 が、入力データ分析表に抽出されたテスト分析項目のどれに対応するかを確認して、項番とコメントを入力データ分析表に追記します。

まず、

要求仕様 001 「enable フラグで機能全体が切り替わること(gb_result.ret_code=FALSE)」

は、テスト分析項目「A」に対応していることが確認出来ます。そこで、これを入力データ分析表に追記します。

1. テスト分析項目「A」の欄の緑のボックスをクリックして、「要求仕様 001」と入力します。
2. 中央の「if(enable)」のコメントを、テスト内容「enable フラグで機能全体が切り替わること」に書き換え

A	要求仕様 001	T		境界値+1	1
	enableフラグで機能全体が切り替わること	F		境界値	0
				境界値-1	-1
				最大値	2147483647
	(カバレッジ用組み合わせに限定)			最小値	-2147483648

さらに、

要求仕様 002: enable が OFF(FALSE)の場合、出力 gb_result→(0, FALSE)であること

は、「f」の処理ブロックに実装されています。これを入力データ分析表に追記します。

f	要求仕様002			
	enable が OFF (FALSE) の場合、出力gb_result→(0, FALSE)であること			
	(カバレッジ用組み合わせに限定)			

同様にして、残りの要求仕様の項番と、入力データ分析表との対応を確認し、追記します。

A	要求仕様 001 enableフラグで機能全体が切り替わることを確認 (カバレッジ用組み合わせに限定)
B	要求仕様 004 modeでモード切替が行われること (カバレッジ用組み合わせに限定)
a	要求仕様 006 モード0で出力gb_result.dataにinput1が選択されること (カバレッジ用組み合わせに限定)
b	要求仕様 007 モード1で出力gb_result.dataにinput2が選択されること (カバレッジ用組み合わせに限定)
c	要求仕様 008 モード2で出力gb_result.dataにinput1、input2の加算値が出力されること (カバレッジ用組み合わせに限定)
d	要求仕様 005 modeが0,1,2以外の値で出力gb_result.data→255に固定されること (カバレッジ用組み合わせに限定)
e	要求仕様 003 enable が ON (TRUE) の場合、出力gb_resul.ret_code→TRUE であること (カバレッジ用組み合わせに限定)
f	要求仕様002 enable が OFF (FALSE) の場合、出力gb_result→(0, FALSE)であること (カバレッジ用組み合わせに限定)

これで、要求仕様に上げられた全ての項目にコード構造が対応していることが確認出来ました。これにより、不要なコード構造が無いことも確認出来たことになります。

テスト指針に基づいてテストデータを編集する

次のステップでは、要求仕様の確認に必要なテストデータを、予め定義したテスト指針(設計ルール)に従って設計します。

最初に、自動抽出されたテストデータの編集を行います。まず、最初の条件分岐の if 文のテストに使用される入力変数 enable は、仕様書には「フラグ」に分類されているため、前に確認したのテスト指針

「指針 1. フラグには TRUE/FALSE のみを付与」

に従い、TRUE/FALSE のデータのみを設定します。このため、自動抽出された境界値－1、最大値、最小値を表から削除します。

1. 変数 enable の境界値－1(緑色)の上で右クリックし、「分析データの削除」を選択します。
2. 変数 enable の最大値(オレンジ色)の上で右クリックし、「分析データの削除」を選択します。
3. 変数 enable の最小値(黄色)の上で右クリックし、「分析データの削除」を選択します。

A	要求仕様 001	T		境界値+1	1
	enableフラグで機能全体が切り替わることを確認	F		境界値	0
	(カバレッジ用組み合わせに限定)				

2 つめの条件分岐の switch 文に使用される変数 mode は、仕様書には「通常変数」に分類されているため、前に確認したのテスト指針、

「指針 2. 通常変数には 必要なデータと最大値/最小値を付加」
「指針 4. レンジ指定のない変数には型の最大値/最小値を付与」

に従い、抽出されたデータをそのまま使用します。

B	要求仕様 004			0		境界値	0
	modeでモード切替が行われること			1		境界値	1
				2		境界値	2
				d		代表値	3
						最大値	2147483647
	(カバレッジ用組み合わせに限定)					最小値	-2147483648

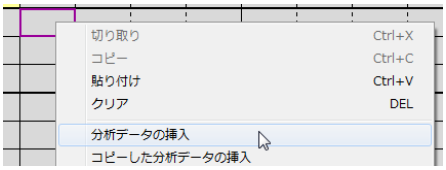
これにより、変数 mode による全てのモード切替(状態遷移)が行われるテストデータが設定できました。

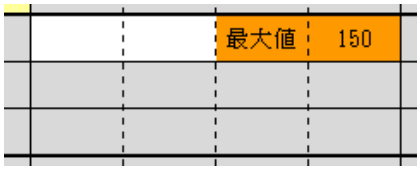
次に、各モードによる動作を確認するテストケースを設定します。まず、要求仕様 006 のテストでは、input1 に設定した値が出力 gb_result.data に正しく出力されることを確認する必要があります。そこで、テスト指針、

「指針 3. レンジ指定のある変数にはレンジの最大値/最小値/中間値を付与」

に従い、最大値→150、最小値→0、中間値(代表値)→75 を設定します。

4. テスト分析項目「a」の欄の「input1」のエリアで右クリックし、「分析データの挿入」を選択します。
5. 右から 2 番目のセルを「最大値」に設定し、右端のセルに「150」を入力します。



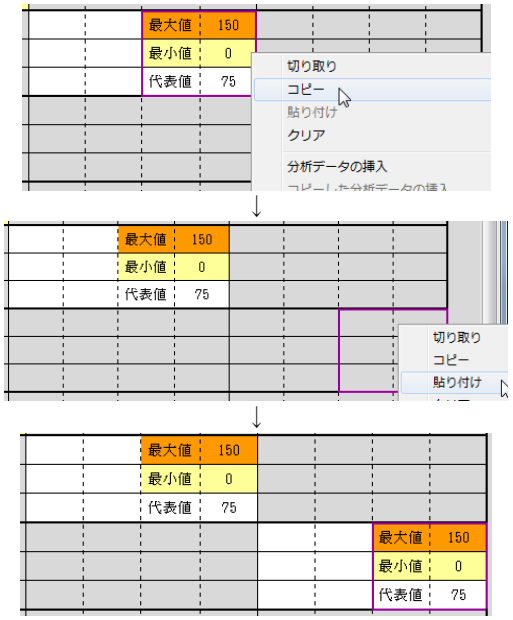


同様にして、最小値→0、中間値(代表値)→75 を設定します。(補足:「代表値」は、テストケースとして与える一般的なデータを指します。)

要求仕様 007のテストも、要求仕様 006 の設定内容と同じです。このため、要求仕様 007 の欄の「input2」に対して、上記と同様の設定を行います。このとき、エディタのコピー&ペースト機能を使用すると、簡単に同じ設定を行う事が

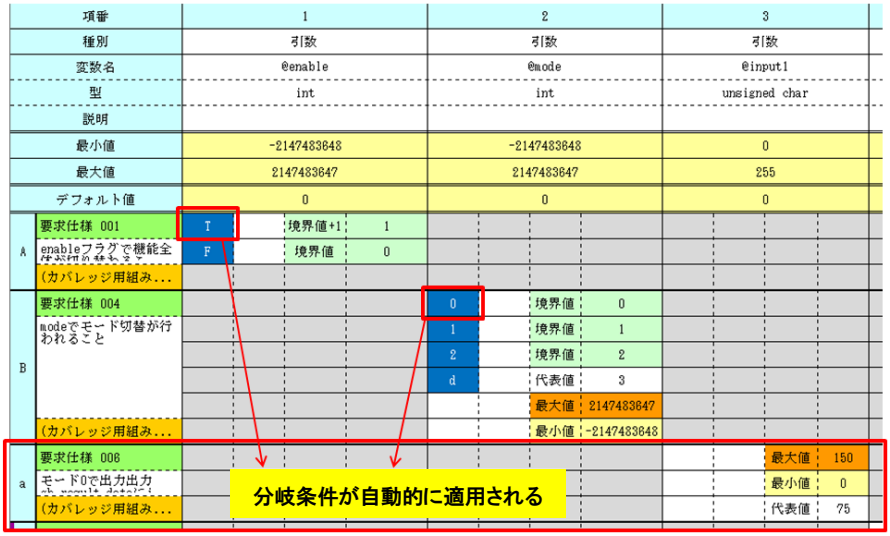
できます。

6. 下図の様に、要求仕様 006 に設定したセルの範囲をマウスで選択します。
7. 右クリックで「コピー」を選択します。(キーボードの Ctrl+C も使用できます。)
8. ペースト先のセル(要求仕様 007 の欄の「input2」)を選択します。
9. 右クリックで「貼り付け」を選択します。(キーボードの Ctrl+V も使用できます。)



ここまでで、要求仕様 006(テスト分析項目「a」)と要求仕様 007(テスト分析項目「b」)のテストデータを設定しました。ただし、これらの処理ブロックに分岐させるためには、分岐条件を考慮する必要があります。例えば、要求仕様 006(テスト分析項目「a」)を実行するためには、enable=TRUE、かつ mode=0 の条件を与えることが必要です。

この設計や作業を効率化するために、テストデータ分析エディタの機能は、この分岐条件の管理と組み合わせをユーザーに代わって行い、自動化する仕組みを持っています。要求仕様 006(テスト分析項目「a」)は、CasePlayer 2により、enable=TRUE、かつ mode=0 の条件のネストに入っていることが解析されています。このため、後でテストケースの組み合わせを生成する際には、変数 enable の「T」の論理に指定されたデータと、変数 mode の「0」の論理に指定されたデータが、自動的に適用されます。そのため、入力テストデータ分析においては、条件分岐をユーザーが考慮する必要はありません。



テスト指針に基づいて組み合わせを指定する

次に、テスト分析項目「c」(要求仕様 008)にある、「モード 2 で出力 gb_result.data に input1、input2 の加算値が出力されること」をテストするためのデータを追加します。ここでは、演算に対するテスト指針、

指針 3: レンジ指定のある変数にはレンジの最大値/最小値/中間値を付与
指針 5: 演算部には、最大値/最小値によるオーバーフローを確認

を適用します。このテスト分析項目「c」には、加算を含む演算があるため、指針 5 に従って、最大値、最小値によるオーバーフローのテストが必要です。ここでは、input1、input2 の各々にレンジの最大値、最小値を設定し、お互いの全ての組み合わせを作成し、テストケースに出力できる様に設計します。

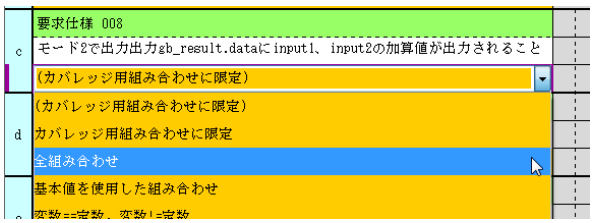
まず、テスト分析項目「c」(要求仕様 008)のテスト分析項目欄の input1、input2 に、最大値、最小値を設定します。

1. テスト分析項目「a」に入力した最大値、最小値のセルを選択します。
   右クリックで「コピー」を選択します。(キーボードの Ctrl+C も使用できます。)
2. テスト分析項目「c」(要求仕様 008)の欄の「input1」のセルを選択します。
   右クリックで「貼り付け」を選択します。(キーボードの Ctrl+V も使用できます。)
3. テスト分析項目「c」(要求仕様 008)の欄の「input2」のセルを選択します。
   右クリックで「貼り付け」を選択します。(キーボードの Ctrl+V も使用できます。)

	項番		3		4	
	種別		引数		引数	
	変数名		@input1		@input2	
	型		unsigned char		unsigned char	
	説明					
	最小値	648	0		0	
	最大値	647	255		255	
	デフォルト値		0		0	
	(カバレッジ用組み合わせに限定)	値 -2147483648				
a	要求仕様 006					
	モード0で出力出力gb_result.dataにinput1が選択されること			最大値 150		
	(カバレッジ用組み合わせに限定)			最小値 0		
				代表値 75		
b	要求仕様 007					最大値 150
	モード1で出力出力gb_result.dataにinput2が選択されること					最小値 0
	(カバレッジ用組み合わせに限定)					代表値 75
c	要求仕様 008			最大値 150		最大値 150
	モード2で出力出力gb_result.dataにinput1、input2の加算値が出力されること			最小値 0		最小値 0
	(カバレッジ用組み合わせに限定)					

テスト分析項目「c」(要求仕様 008)のテスト分析項目について、テストケース生成時に、最大値と最小値のお互いの全ての組み合わせが生成されるように、組み合わせルールを設定します。

4. テスト分析項目「c」(要求仕様 008)の組み合わせルール(オレンジ色のプルダウン)から、「全組み合わせ」を選択します。



さらに、

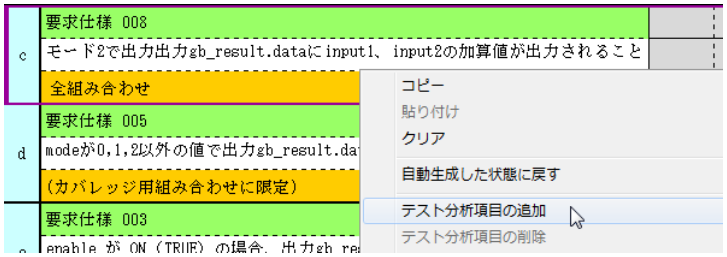
指針 3: レンジ指定のある変数にはレンジの最大値/最小値/中間値を付与

の指針に従ったテストデータを追加します。最大値、最小値については既に全数組み合わせを作成しているため、こ

こでは中間値のテストデータのみを追加し、最大値、最小値とは別の組み合わせとしてテストケースを設計します。

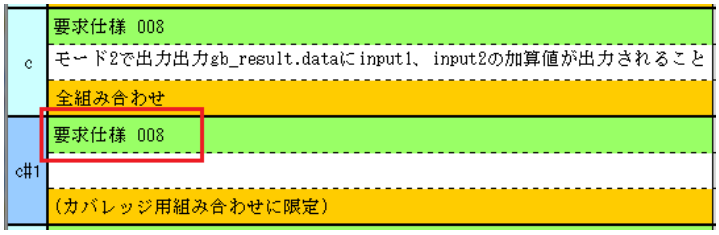
ここで、テスト分析項目「c」のサブテスト分析項目を作成します。テスト分析項目に属するサブテスト分析項目欄を追加し、テストデータを設定した場合は、その処理ブロックへ分岐するための他の変数の条件は、親のテスト分析項目と同様に自動的に適用されます。

5. テスト分析項目「c」(要求仕様 008)を選択します。
6. 右クリックで、「テスト分析項目の追加」を選択します。
7. テスト分析項目「c」のサブブロック、「c#1」が追加されます。



要求仕様の項番を記載しておきます。

8. テスト分析項目「c#1」の欄の緑のボックスをクリックして、「要求仕様 008」と入力します。



9. テスト分析項目「a」に入力した代表値のセルを選択します。
   右クリックで「コピー」を選択します。(キーボードの Ctrl+C も使用できます。)
10. テスト分析項目「c#1」(要求仕様 008)の欄の「input1」のセルを選択します。
    右クリックで「貼り付け」を選択します。(キーボードの Ctrl+V も使用できます。)
11. テスト分析項目「c#1」(要求仕様 008)の欄の「input2」のセルを選択します。
    右クリックで「貼り付け」を選択します。(キーボードの Ctrl+V も使用できます。)

項番				3			4			
種別				引数			引数			
変数名				@input1			@input2			
型				unsigned char			unsigned char			
説明										
最小値				0			0			
最大値				255			255			
デフォルト値				0			0			
a	要求仕様 006				最大値	150				
	モード0で出力出力gb_result.dataにinput1が選択されること				最小値	0				
	(カバレッジ用組み合わせに限定)				代表値	75				
b	要求仕様 007							最大値	150	
	モード1で出力出力gb_result.dataにinput2が選択されること							最小値	0	
	(カバレッジ用組み合わせに限定)							代表値	75	
c	要求仕様 008				最大値	150		最大値	150	
	モード2で出力出力gb_result.dataにinput1、input2の加算値が出力されること				最小値	0		最小値	0	
	全組み合わせ									
c#1	要求仕様 008				代表値	75		代表値	75	
	(カバレッジ用組み合わせに限定)									

テスト分析項目「c#1」(要求仕様 008)のテスト分析項目について、テストケース生成時に、代表値の組み合わせが生成されるように、組み合わせルールを設定します。

12. テスト分析項目「c#1」(要求仕様 008)の組み合わせルール(オレンジ色のプルダウン)から、「全組み合わせ」を選択します。

c	要求仕様 008
	モード2で出力出力gb_result.dataにinput1、input2の加算値が出力されること
	全組み合わせ
c#1	要求仕様 008
	全組み合わせ

これで、演算部のテストケースとして、input1 と input2 の最大値、最小値の全組み合わせ、代表値同士の組み合わせがテストケースに生成されるように設定されました。

残りの動作仕様を確認するためのテストデータ追加する

入力データ分析表には、テスト分析項目「d」、「e」、「f」が残っています。テスト分析項目「d」は、モードが 0、1、2 以外の時の動作をテストする目的で作成し、要求仕様の項番を「005」としましたが、このテストデータは、テスト分析項目「B」(要求仕様 004)に、分岐条件を指定するデータとして既に設定されています。そのため、テスト分析項目「d」にテストデータを設定しなくても、このモードのテストを実行するテストケースはテスト分析項目「B」に生成されるのですが、ここでは、要求仕様との対応を明確にするために、テスト分析項目「B」とは別に、テスト分析項目「d」としてテストケースが生成される様に設定します。

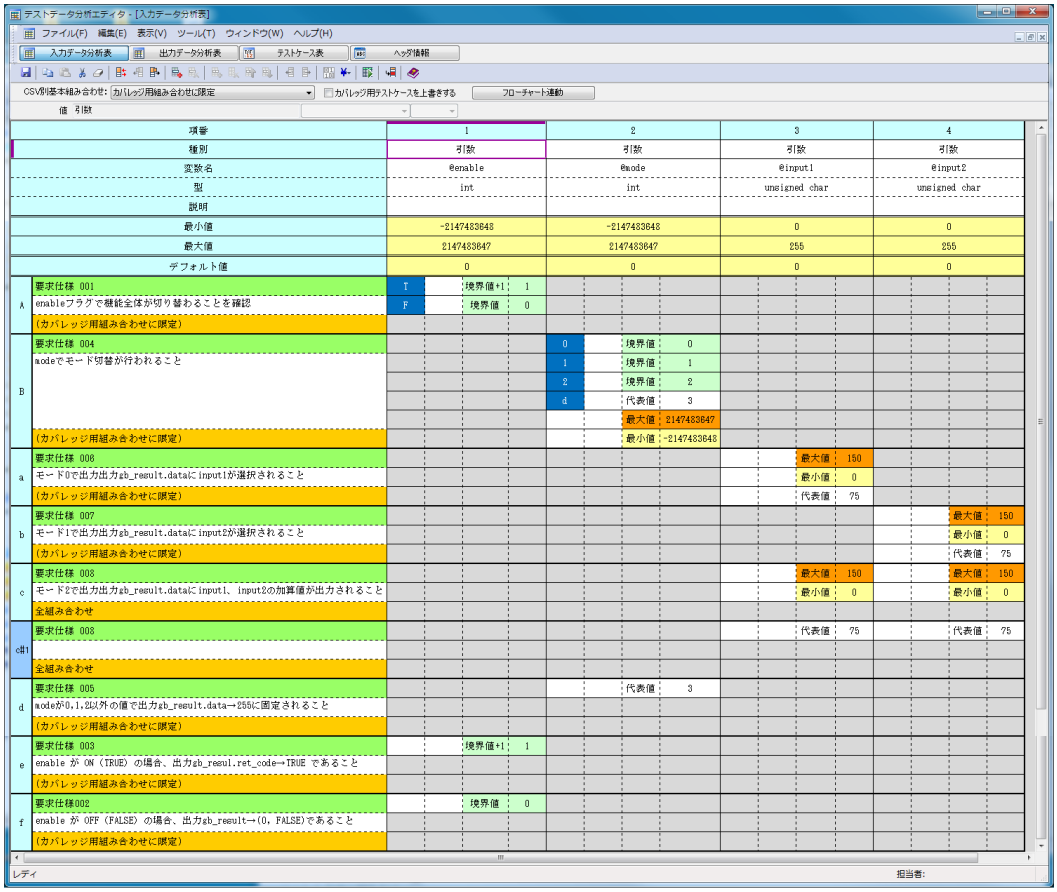
1. テスト分析項目「d」(要求仕様 005)の欄で、変数「@mode」で右クリックし、「分析データの挿入」を選択します。
2. 「代表値 3」を設定します。

項番			2			
種別			引数			
変数名			@mode			
型			int			
説明						
最小値			-2147483648			
最大値			2147483647			
デフォルト値			0			
c#1	要求仕様 008					
	全組み合わせ					
d	要求仕様 005				代表値	3
	modeが0,1,2以外の値で出力gb_result.data→255に固定されること					
	(カバレッジ用組み合わせに限定)					

これで、テスト分析項目「d」としてテストケースが1つ生成されます。同様に、テスト分析項目「e」、「f」についても、既にテスト分析項目「A」でテストデータは設定されていますが、要求仕様との対応を明確にするために、変数@enable にデータを設定しておきます。下図の様に設定します。

項番		1				2		
種別		引数				引数		
変数名		@enable				@mode		
型		int				int		
説明								
最小値		-2147483648				-2147483648		
最大値		2147483647				2147483647		
デフォルト値		0				0		
	全組み合わせ							
d	要求仕様 005						代表値	3
	modeが0,1,2以外の値で出力gb_result.data→255に固定されること							
	(カバレッジ用組み合わせに限定)							
e	要求仕様 003		境界値+1	1				
	enable が ON (TRUE) の場合、出力gb_resul.ret_code→TRUE であること							
	(カバレッジ用組み合わせに限定)							
f	要求仕様002		境界値	0				
	enable が OFF (FALSE) の場合、出力gb_result→(0, FALSE)であること							
	(カバレッジ用組み合わせに限定)							

以上で、各要求仕項目とテスト分析項目との対応確認と、テストデータの設定ができました。入力データ分析表の全体は、以下の様になっています。



テストケースを自動生成する

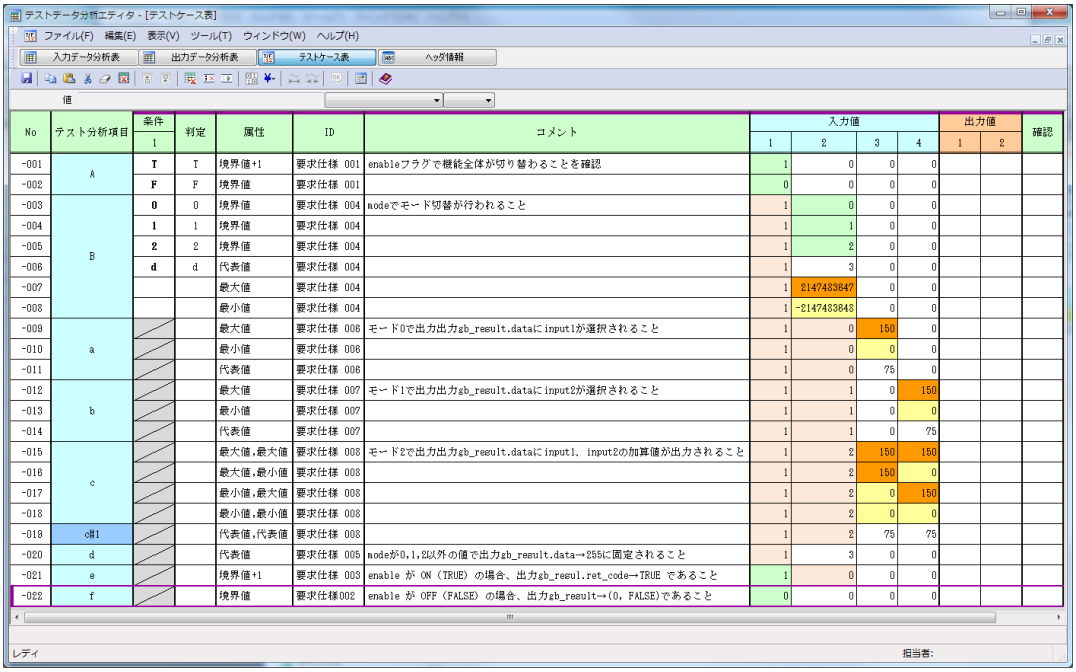
では、完成した入力データ分析表からテストケースを生成します。

1. 入力データ分析表の任意の1つのセル(下図の例は「引数」)を選択します。(全てのテスト分析項目のテストケースを生成するために、特定のテスト分析項目が選択されていない状態にします。)

項番	1	
種別	引数	
変数名	@enable	
型	int	
説明		

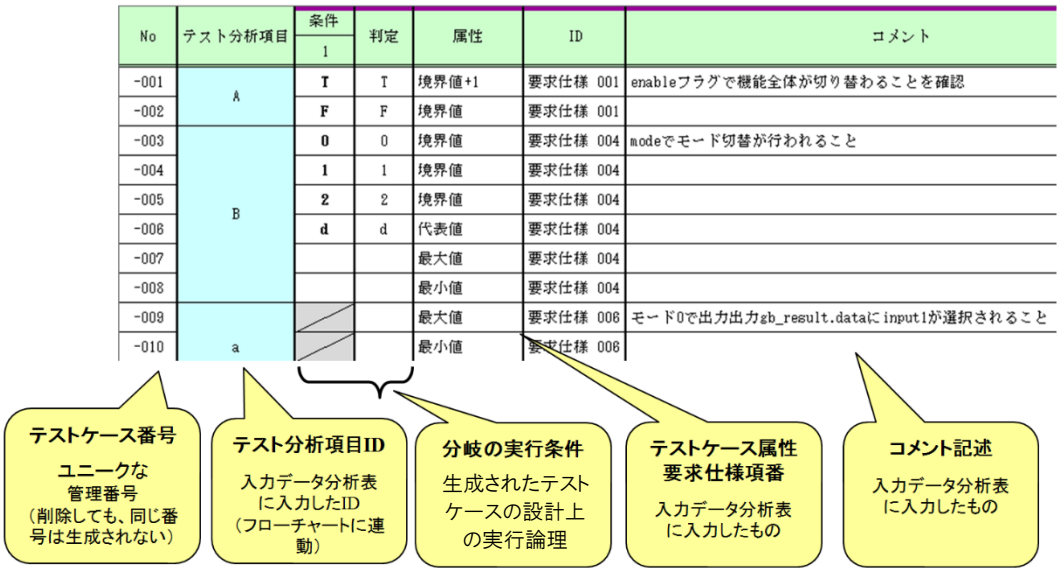
2. 「編集」メニューから「テスト分析項目の組み合わせ生成」を選択します。

テストケース表が生成されます。



No	テスト分析項目	条件	判定	属性	ID	コメント	入力値				出力値		確認
		1					1	2	3	4	1	2	
-001	A	T	T	境界値+1	要求仕様 001	enableフラグで機能全体が切り替わることを確認	1	0	0	0			
-002		F	F	境界値	要求仕様 001		0	0	0	0			
-003	B	0	0	境界値	要求仕様 004	modeでモード切替が行われること	1	0	0	0			
-004		1	1	境界値	要求仕様 004		1	1	0	0			
-005		2	2	境界値	要求仕様 004		1	2	0	0			
-006		d	d	代表値	要求仕様 004		1	3	0	0			
-007				最大値	要求仕様 004		1	2147483647	0	0			
-008				最小値	要求仕様 004		1	-2147483648	0	0			
-009	a			最大値	要求仕様 006	モード0で出力出力gb_result.dataにinput1が選択されること	1	0	150	0			
-010				最小値	要求仕様 006		1	0	0	0			
-011				代表値	要求仕様 006		1	0	75	0			
-012	b			最大値	要求仕様 007	モード1で出力出力gb_result.dataにinput2が選択されること	1	1	0	150			
-013				最小値	要求仕様 007		1	1	0	0			
-014				代表値	要求仕様 007		1	1	0	75			
-015	c			最大値,最大値	要求仕様 008	モード2で出力出力gb_result.dataにinput1、input2の加算値が出力されること	1	2	150	150			
-016				最大値,最小値	要求仕様 008		1	2	150	0			
-017				最小値,最大値	要求仕様 008		1	2	0	150			
-018				最小値,最小値	要求仕様 008		1	2	0	0			
-019	c補1			代表値,代表値	要求仕様 008		1	2	75	75			
-020	d			代表値	要求仕様 005	modeが0,1,2以外の値で出力gb_result.data→255に固定されること	1	3	0	0			
-021	e			境界値+1	要求仕様 003	enable が ON (TRUE) の場合、出力gb_resul.ret_code→TRUE であること	1	0	0	0			
-022	f			境界値	要求仕様002	enable が OFF (FALSE) の場合、出力gb_result→(0, FALSE)であること	0	0	0	0			

生成されるテストケース表は、最終的にカバレッジマスターwinAMS に入力する CSV ファイルの入出データ、期待値の他に、データの属性や、分岐箇所の実行論理、要求仕様への参照情報を含んでいます。



No	テスト分析項目	条件	判定	属性	ID	コメント
		1				
-001	A	T	T	境界値+1	要求仕様 001	enableフラグで機能全体が切り替わることを確認
-002		F	F	境界値	要求仕様 001	
-003	B	0	0	境界値	要求仕様 004	modeでモード切替が行われること
-004		1	1	境界値	要求仕様 004	
-005		2	2	境界値	要求仕様 004	
-006		d	d	代表値	要求仕様 004	
-007				最大値	要求仕様 004	
-008				最小値	要求仕様 004	
-009	a			最大値	要求仕様 006	モード0で出力出力gb_result.dataにinput1が選択されること
-010				最小値	要求仕様 006	

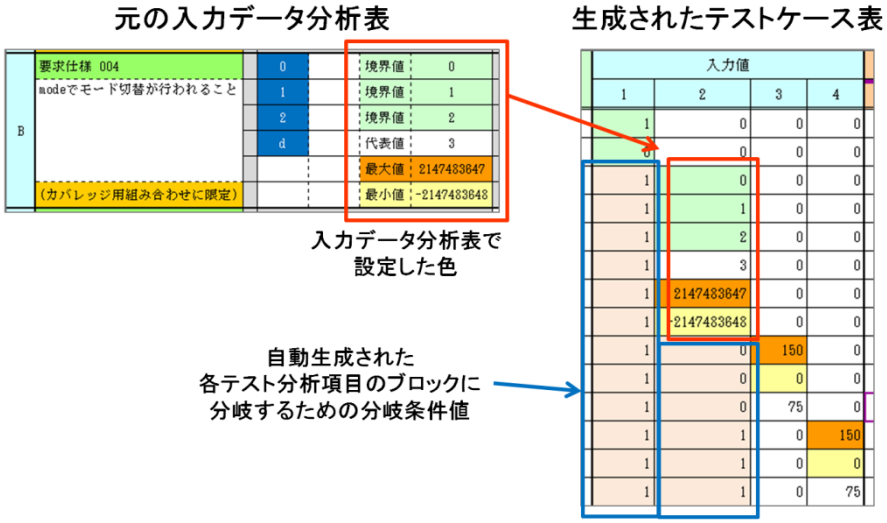
上図のテストケース表で、特に、「条件」「判定」の項目は、テスト分析項目に条件分岐を含む場合に、設計されたテストケースの実行論理が記載されており、テストケースをレビューする際に有効な情報です。

［参考］

条件 ： 条件式の実行論理(複合条件がある場合には番号が追加され、個別の論理が示される)
判定 ： 条件式全体の論理(複合条件式の場合には、その全体の論理が示される)

また、テストケース表の幅を小さくするために、入力変数、出力変数の名称は表示しておらず、すべてIDで表示されます。変数名は、テストケースを選択した際に、表の下部のバーに表示されます。

テストデータのセルの色は、入力データ分析表に設定した境界値、最小値、最大値、代表値などの色がそのまま表示されます。また、薄いピンク色のセルは、自身のテスト分析項目以外の値を使用していることを示します。ここでは、各分析項目のブロックに分岐するための条件値(A の@enable の T の分析データ)が自動生成されていることを示しています。



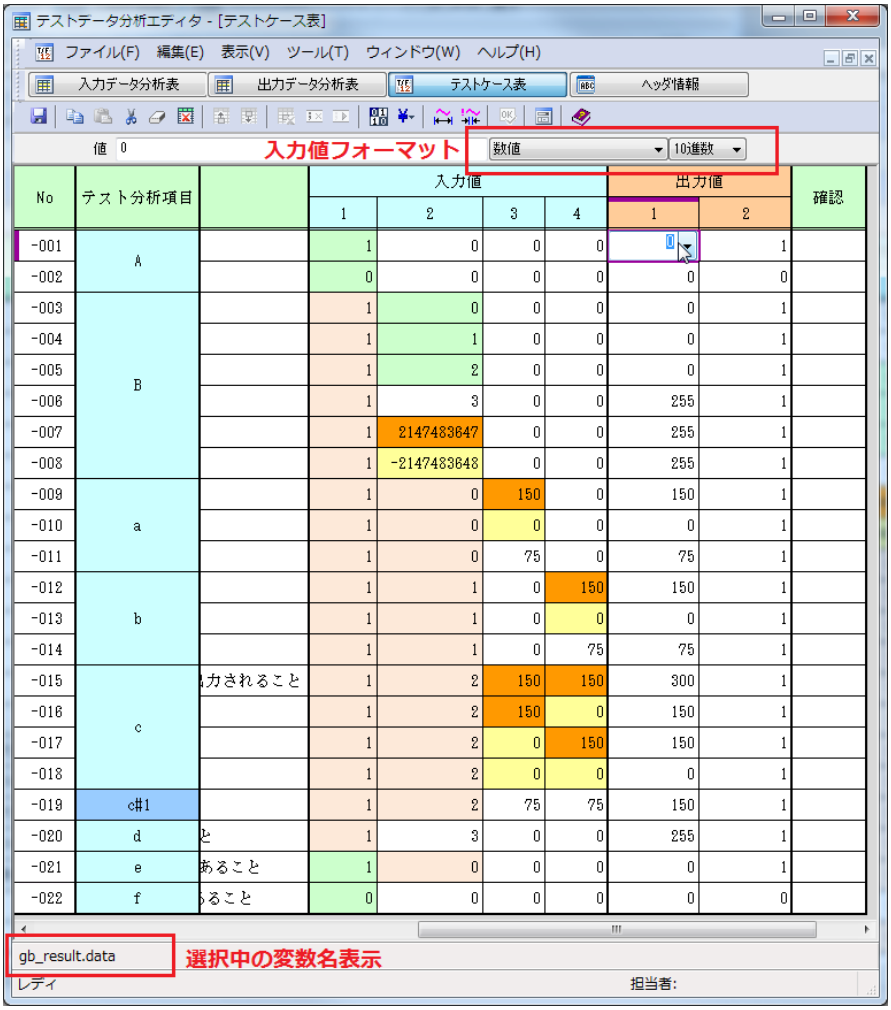
テストケース表をレビューして期待値を入力する

では、前述の関数設計仕様に基づいて、期待値を設定します。参照のため、下に関数仕様を示します。この作業は、コードからではなく、関数仕様を基に設定しなければなりません。func5 設計仕様を参照して期待値を算出し、テストケース表の「出力値」欄を埋めてください。

〈 関数 func5 設計仕様 〉
2つの入力 input1、input2 に対し、モード切替値(mode)によって値を決定し、
gb_result.data に出力する。なお、機能全体の ON/OFF は入力フラグ enable で決定する。

入力:(全て引数)
enable (フラグ): 機能の ON/OFF OFF の場合 出力 gb_result→(0, FALSE)
mode (変数) : モード切替 0: 出力 gb_result→(input1, TRUE)
1: 出力 gb_result→(input2, TRUE)
2: 出力 gb_result→(input1+input2, TRUE)
default: 出力 gb_result→(255, TRUE)
input1(変数) : 入力1 レンジ(0～150)
input2(変数) : 入力2 レンジ(0～150)

出力:(グローバル構造体)
gb_result.data : 出力データ
gb_result.ret_code : エラーコード(機能 ON の時 TRUE、それ以外 FALSE)



No	テスト分析項目		入力値 1	入力値 2	入力値 3	入力値 4	出力値 1	出力値 2	確認
-001	A		1	0	0	0		1	
-002			0	0	0	0	0	0	
-003	B		1	0	0	0	0	1	
-004			1	1	0	0	0	1	
-005			1	2	0	0	0	1	
-006			1	3	0	0	255	1	
-007			1	2147483647	0	0	255	1	
-008			1	-2147483648	0	0	255	1	
-009	a		1	0	150	0	150	1	
-010			1	0	0	0	0	1	
-011			1	0	75	0	75	1	
-012	b		1	1	0	150	150	1	
-013			1	1	0	0	0	1	
-014			1	1	0	75	75	1	
-015	c	力されること	1	2	150	150	300	1	
-016			1	2	150	0	150	1	
-017			1	2	0	150	150	1	
-018			1	2	0	0	0	1	
-019	c#1		1	2	75	75	150	1	
-020	d	と	1	3	0	0	255	1	
-021	e	あること	1	0	0	0	0	1	
-022	f	ること	0	0	0	0	0	0	

また、テスト分析項目「c」については、このブロックに演算（加算）の処理があるため、テストケースの組み合わせ時に、最大値、最小値の全数組み合わせを生成するように指定を行いました。この組み合わせが生成されていることを確認してください。

元の入力データ分析表

	項番		3	4
	種別		引数	引数
	変数名		@input1	@input2
	型		unsigned char	unsigned char
	説明			
	最小値		0	0
	最大値		255	255
	デフォルト値		0	0
c	要求仕様 008		最大値 150	最大値 150
c	モード2で出力出力gb_result.dataにinput1、input2の加算値が出力されること		最小値 0	最小値 0
c	全組み合わせ			
c#1	要求仕様 008		代表値 75	代表値 75
c#1				
c#1	全組み合わせ			

全数組み合わせが生成されていることを確認

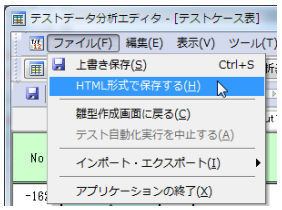
生成されたテストケース表

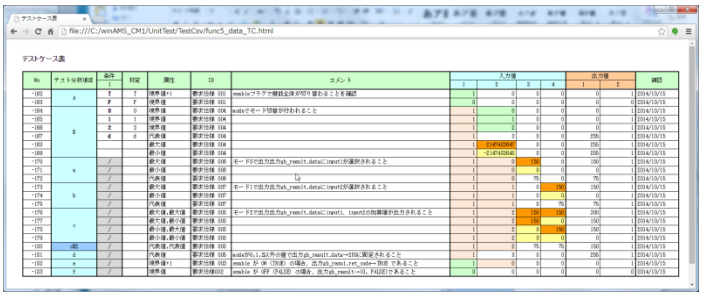
c	モード2で出力出力gb_result.dataにinput1、input2の加算値が出力されること	1	2	150	150
c		1	2	150	0
c		1	2	0	150
c		1	2	0	0
c#1		1	2	75	75

テストケースを確認したことを記録するために、テストケース表の「確認」欄が使用できます。今日の日付を自動入力できますが、任意の文字を入力する事も可能です。

[参考]各分析表を HTML に出力する

入力データ分析表、出力データ分析表、テストケース表の各表は、エディタのイメージのまま HTML ファイルに保存可能です。これにより、カバレッジマスターwinAMS を起動しないオフラインでも、生成したテストケース表などの確認が可能です。

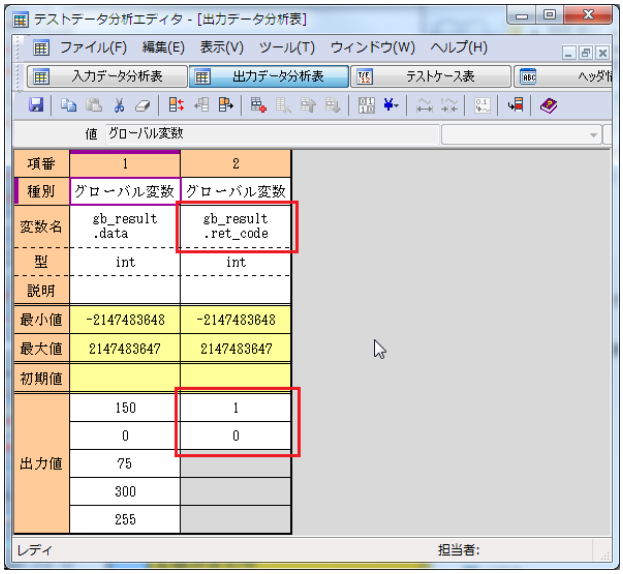




出力データ分析表を確認する

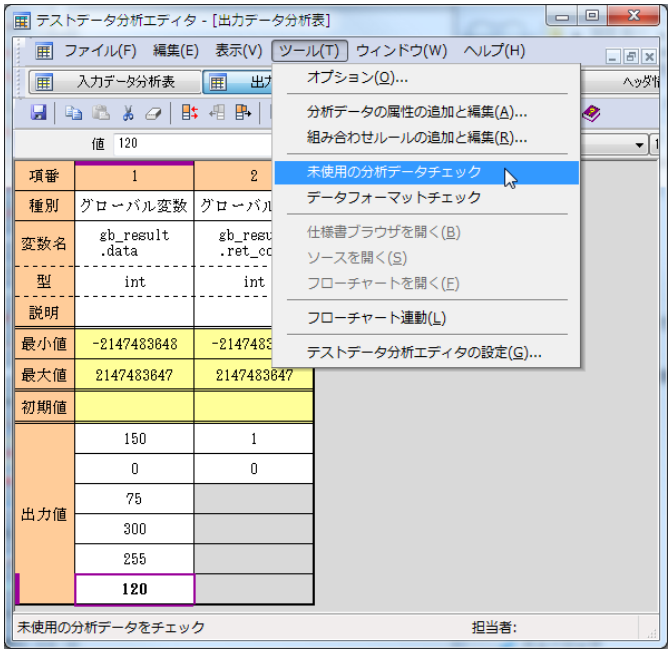
生成されたテストケース表に出力値(期待値)を設定すると、設定した値の一覧が「出力データ分析表」に表示されます。テストケース表の出力値に入力された値から重複するデータを1つにまとめて一覧にしたものです。この表は、今回の単体テスト設計で、出力値としてテストされるデータ値を示しています。

今回のケースでは、グローバル変数 gb_result.ret_code は、出力として TRUE/FALSE(1/0)を返す変数ですが、この仕様を確認するためには、出力値が1と0になるテストケースが作られている必要があります。出力データ分析表の gb_result.ret_code には1と0が表示されていますので、確かにこの条件をテストするテストケースが少なくとも1つは含まれており、テストケース漏れが発生していないことが確認出来ます。



[参考]予め出力データ分析表を設定し仕様確認に応用する

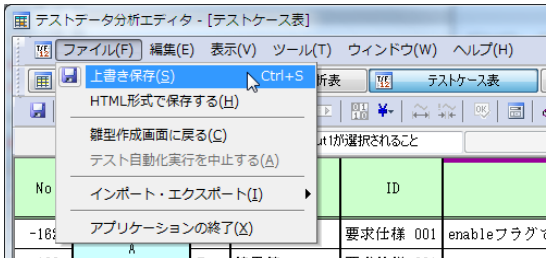
出力データ分析表は、テストケース表に出力値(期待値)を入力する前に、想定される出力値(要求仕様に基づいて出力値としてテストする必要がある値)を予め入力することもできます。要求仕様に基づき出力データ分析表を設定した後で、テストケース表に出力値を設定した場合、テストケース表の出力値が出力データ分析表の値を網羅しているかを確認出来ます。例えば、下図のように、「120」が出力値としてテストされなければならないとして出力データ分析表に入力されていた場合、その値がテストケース表の出力値に含まれていない場合、「ツール」メニュー→「未使用分析データチェック」を選択することで、該当する値をハイライト(太字表示)することができます。

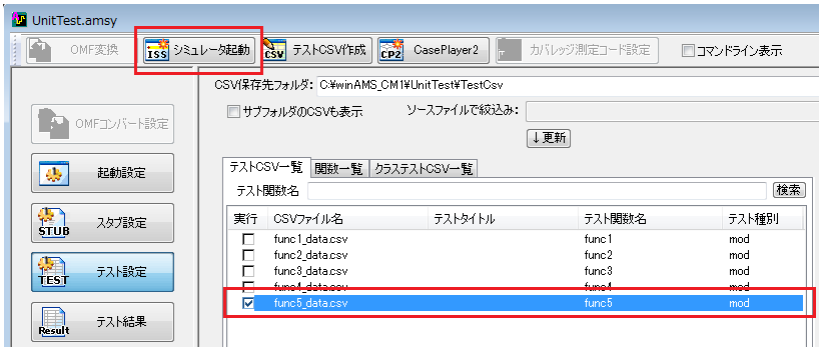


CSV ファイルを生成して単体テストを実行する

最後に、テストケース表から CSV ファイルを生成し、テスト実行後、結果を確認してみます。

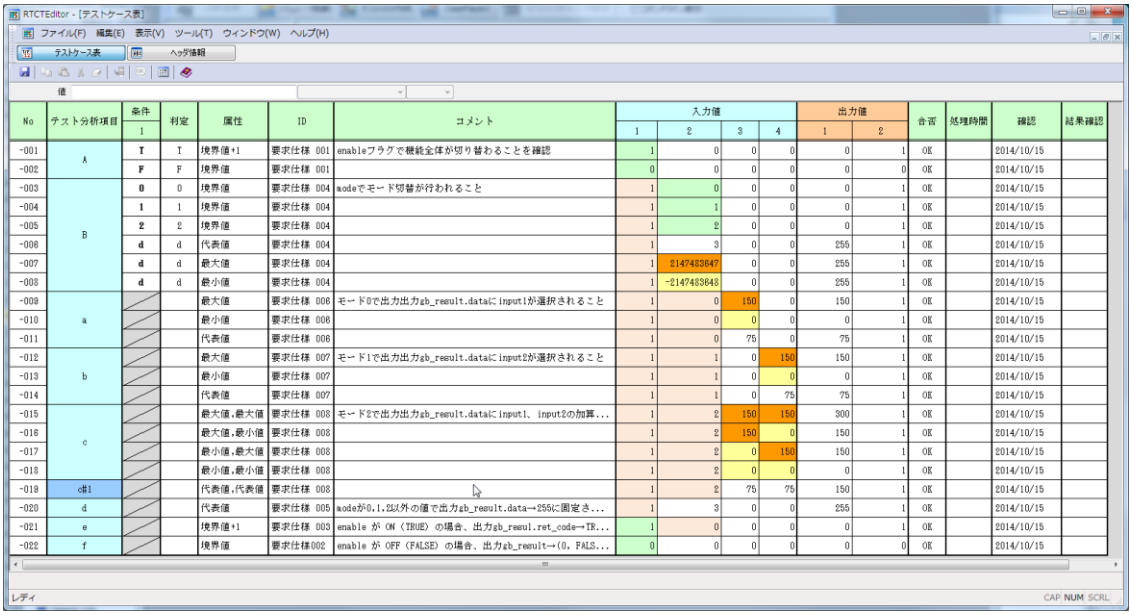
1. テストケース表を表示し、「ファイル」メニュー→「上書き保存」を選択します。(この時に、CSV ファイルにデータが書き込まれます。)
2. テストデータ分析エディタを終了します。
3. SSTManager の「テスト設定」ビューで「func5_data.csv」を選択します。
4. 「シミュレータ起動」ボタンを押して、テストを実行します。





実行が終了したら、結果を確認します。結果は Excel で CSV ファイルを直接見るのではなく、カバレッジマスター winAMS の内部ビューアを使用して、他の出力情報も合わせて表示してみます。

5. SSTManager の「その他」ビュー→「テスト結果 CSV ファイルを外部エディタで開く」のオプションを OFF にします。
6. 「テスト結果」ビュー→「func5_data.csv」をダブルクリックします。
7. テストケース表(出力結果)が内部エディタで表示されます。



このテストケース表には、出力値と期待値の判定結果に加え、分岐を含むテスト分析項目については、実際の分岐事項条件が出力されます。この論理は、実際にシミュレータにより計測した実測結果を元に表示されています。（入力データ設計時に使用したテストケース表にも条件の表示がありますが、これは、CasePlayer2 の解析を基にした設計時の論理です。）

入力値				出力値		合否	処理
1	2	3	4	1	2		
1	0	0	0	0	1	OK	
0	0	0	0	0	0	OK	
1	0	0	0	0	1	OK	
1	1	0	0	0	1	OK	
1	2	0	0	0	1	OK	
1	3	0	0	255	1	OK	
1	2147483647	0	0	255	1	OK	
1	-2147483648	0	0	255	1	OK	
1	0	150	0	150	1	OK	

期待値判定

シミュレータにより計測した実測結果を基にした実行論理

以上で、テストケース分析エディタを使用したテスト設計の学習（実習5）は終了です。

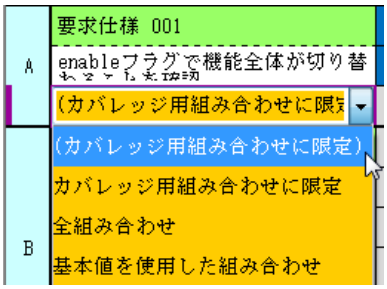
[応用]テスト分析項目の組み合わせルール

ここでは、入力データ分析表からテストケース表を作成する際の、テストデータの組み合わせについて解説します。実習5では、「全組み合わせ」を一部に適用し、他の詳細には触れませんでしたが、意図したテストを実施するためのテストケース生成には、この組み合わせの機能、仕組みを理解しておく必要があります。

デフォルトの組み合わせルール

基本ルールとして、入力データ分析表に入力されたテストデータは、組み合わせを作成する際に必ず一度はテストケース表に出力されます。入力データ分析表に入力されているにもかかわらず、テストケース表に出力されないデータはありません。ただし、この基本ルールのみでは、そのデータがテストケース表に出力される事は保証されていますが、他の変数に与えたデータと、どのように組み合わせが作成されるかについては決定されていません。そこで、ここで解説する組み合わせのルールを適用することで、他の変数に与えたデータとの組み合わせを考慮したテストケースを生成できるようになります。

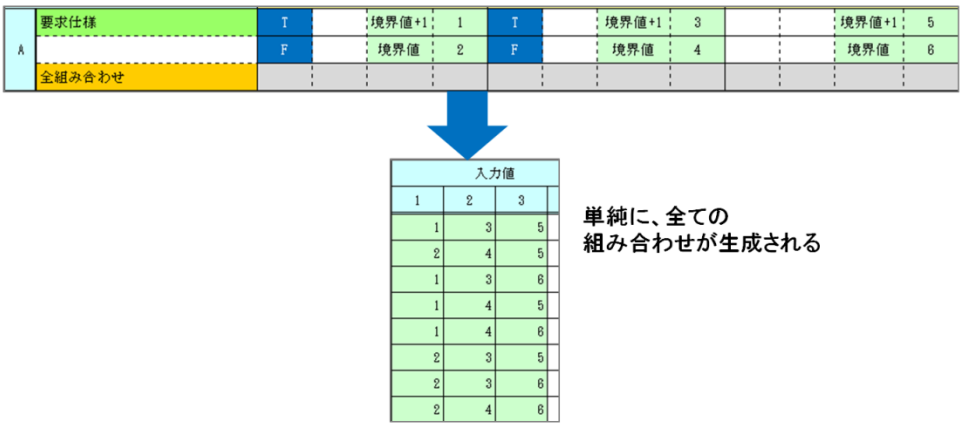
最初に、入力データ分析表にデフォルトで設定されている組み合わせルールについて説明します。予めツールに用意されている組み合わせルールの選択肢は以下の3つです。



■全組み合わせ

この組み合わせルールは、最も単純な組み合わせルールです。各変数に付けたフラグや属性にかかわらず、全てのデータの組み合わせを生成します。例えば3つの変数に各々2つずつのデータが与えられている場合にこのルールを適用して組み合わせを作成すると、これらの全てを組み合わせた、2×2×2＝8個のテストケースが生成されます。

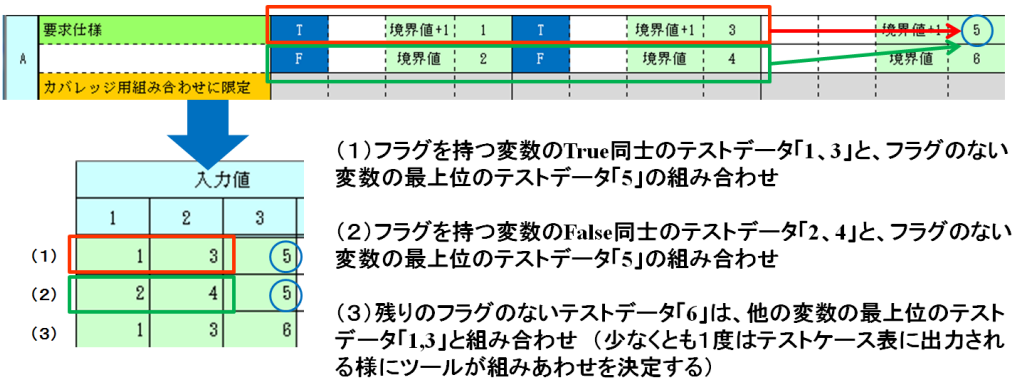
このルールは各変数同士の関連性が高く、例えば複数の変数の演算部分や複雑な複合条件による分岐などで、各データの組み合わせに依存する不具合が発生する可能性が高いテスト分析項目に適用します。全組み合わせのルールは、生成されるテストケースの個数は最も多くなりますが、網羅性は 100%となり、組み合わせが不足することでテスト漏れを発生してしまうリスクを最小限にできます。



■カバレッジ用組み合わせに限定

この組み合わせルールは、分岐条件に付けた「TRUE フラグ」、「FALSE フラグ」、「case ラベルフラグ」によって組み合わせを制御します。分岐を含むテスト分析項目には、その分岐の論理に対応するテストケースにこれらのフラグを設定しますが、このルールで組み合わせを作成した場合には、フラグの付いた変数に対しては、同じフラグのテストデータ同士の組み合わせを1つずつ作成します。フラグの付いていない変数のデータについては、組み合わせはツールに任され、他の変数に設定されたデータのうち、最上位の1つのデータとのみ、組み合わせが生成されます。

この組み合わせルールは、コードを網羅する(カバレッジを満たす)条件の組み合わせを確実に作成し、生成されるテストケースの個数を最小限に留めたい場合に適用します。



(1)フラグを持つ変数のTrue同士のテストデータ「1、3」と、フラグのない変数の最上位のテストデータ「5」の組み合わせ

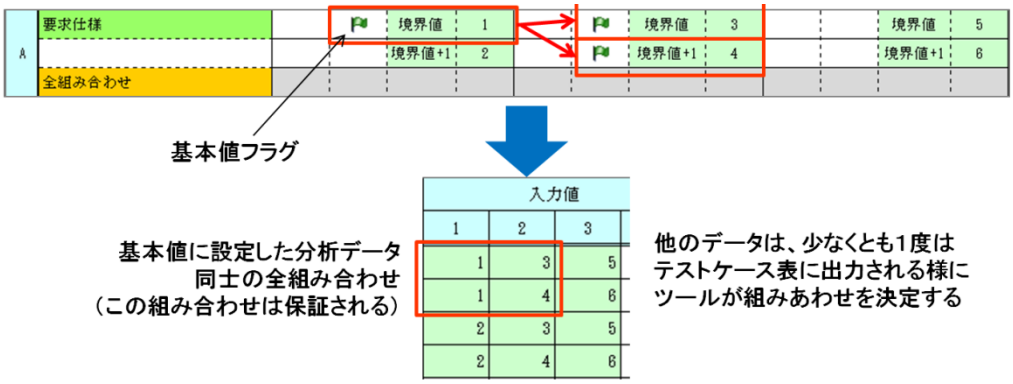
(2)フラグを持つ変数のFalse同士のテストデータ「2、4」と、フラグのない変数の最上位のテストデータ「5」の組み合わせ

(3)残りのフラグのないテストデータ「6」は、他の変数の最上位のテストデータ「1,3」と組み合わせ (少なくとも1度はテストケース表に出力される様にツールが組みあわせを決定する)

■基本値を使用した組み合わせ

この組み合わせルールは、入力データ分析表に設定した各テストデータの「基本値フラグ」によって組み合わせを制御します。1つのテスト分析項目の中に設定した変数のうち、基本値に指定したデータのみに全組み合わせを適用します。

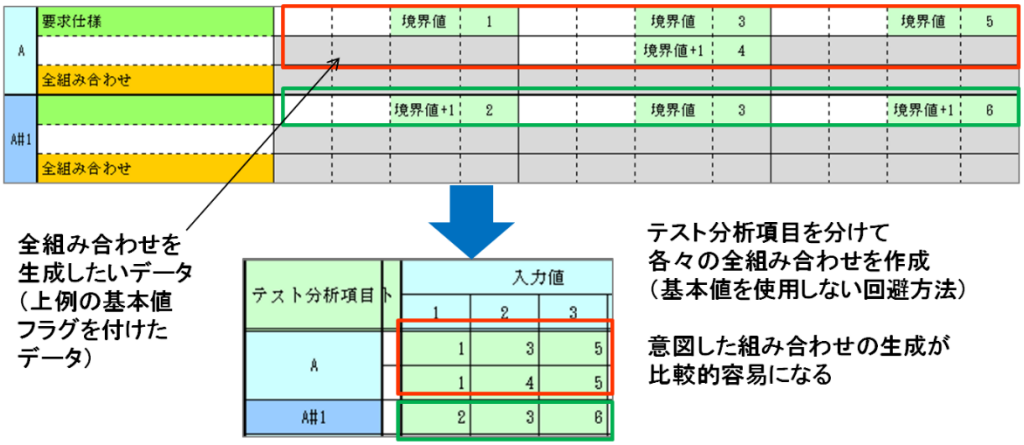
具体的には、基本値に指定したデータの全組み合わせを生成し、他の基本値に設定していない分析データに対しては、少なくとも1回はデータが出力される様にテストケースを生成します。この際のデータ組み合わせはツールに任されます。



基本値に設定した分析データ同士の全組み合わせ(この組み合わせは保証される)

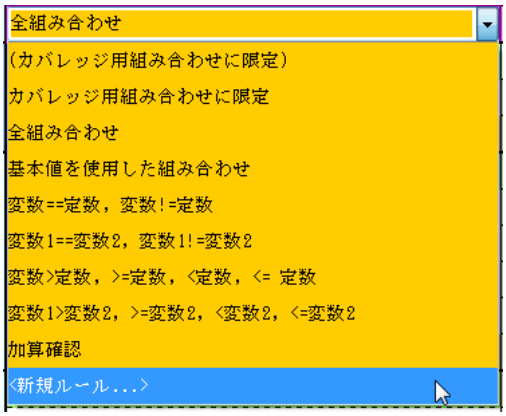
他のデータは、少なくとも1度はテストケース表に出力される様にツールが組みあわせを決定する

この基本値を使用する方法は、1つのテスト分析項目の中の部分的なデータに関してのみ、テストケースの組み合わせの網羅性を保証したい場合に使用できます。ただし、テストデータ数が多い場合には、基本値の設定に工数がかかることや、基本値を設定しないデータの組み合わせがツールに任されるため、最終的に出力されたテストケースの網羅性の確認が困難になることもあります。この様な場合には、テスト分析項目を複数に分け、各々のテスト分析項目の中で全組み合わせを適用する方が、テストケースの網羅性の確認が容易になります。

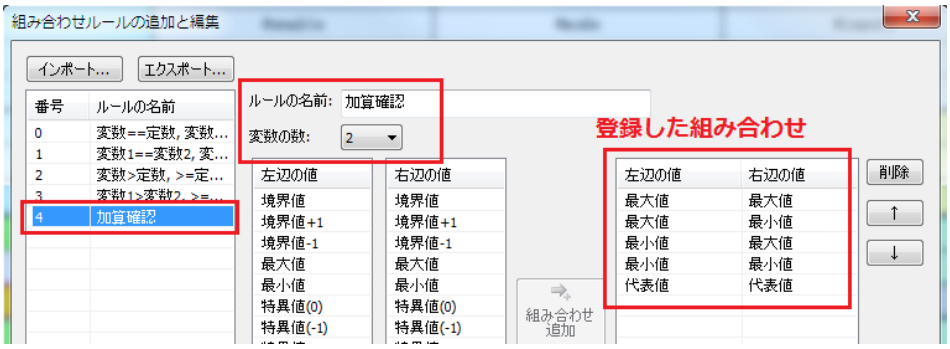


新規組み合わせルールの作成

入力データ分析表にデフォルトで設定されている3つの組み合わせルールの他に、ユーザー自身がデータに設定した属性に基づいて、組み合わせのルールを作成することができます。



組み合わせのルールの追加と編集は、上図の様に組み合わせ選択のプルダウンから「＜新規ルール…＞」を選択するか、「ツール」メニューから「組み合わせルールの追加と編集」を選択して行います。



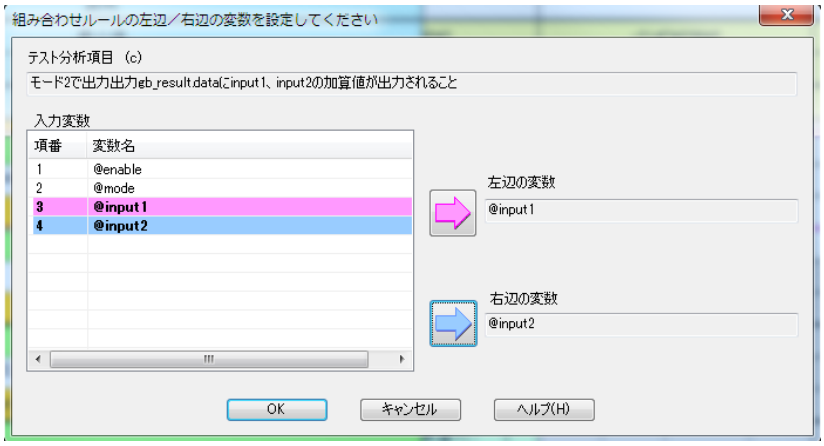
上図の例では、「加算確認」の名称を付けたルールを作成しています。2つの変数に与えたデータの組み合わせを作成する際に、「最大値」と「最小値」の属性の付いたデータの全組み合わせと、代表値同士の組み合わせが作成される様に設定されています。

この様に、入力データ分析表に設定したデータの属性を使用して、全組み合わせを作成するデータを指定する機能です。この組み合わせに指定しない属性のデータについては、少なくとも 1 度はテストケースに出力される様に組み合わせが作成されます。

例えば、実習5の演算確認部分に、作成した新規ルールを適用して組み合わせを作って見ます。

項番			3			4		
種別			引数			引数		
変数名			@input1			@input2		
型			unsigned char			unsigned char		
説明								
最小値			0			0		
最大値			255			255		
デフォルト値			0			0		
	要求仕様 008			最大値	150		最大値	150
c	モード2で出力出力gb_result.dataにinput1、input2の加算値が出力されること			最小値	0		最小値	0
	加算確認			代表値	75		代表値	75

作成したルール「加算確認」を適用した場合、まず、ルールに設定した「左辺の値」と「右辺の値」にどの変数を使用するかを指定する画面が表示されます。



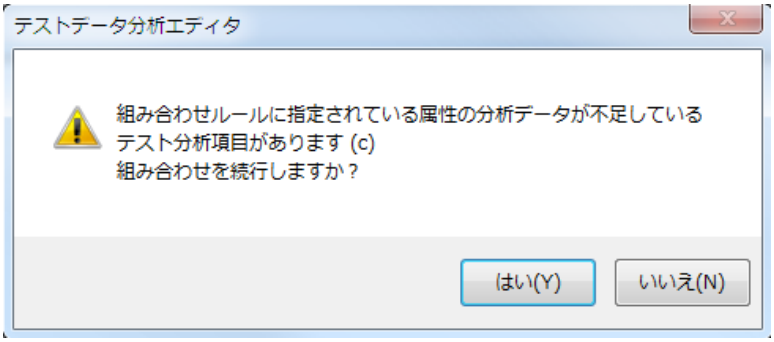
各々、input1、input2 を指定することで、これらの組み合わせが生成されます。

テスト分析項目	条件	判定	属性	ID	コ...	入力値			
	1					1	2	3	4
c			最大値,最大値	要求仕様 008	モ...	0	2	150	150
			最大値,最小値	要求仕様 008		0	2	150	0
			最小値,最大値	要求仕様 008		0	2	0	150
			最小値,最小値	要求仕様 008		0	2	0	0
			代表値,代表値	要求仕様 008		0	2	75	75

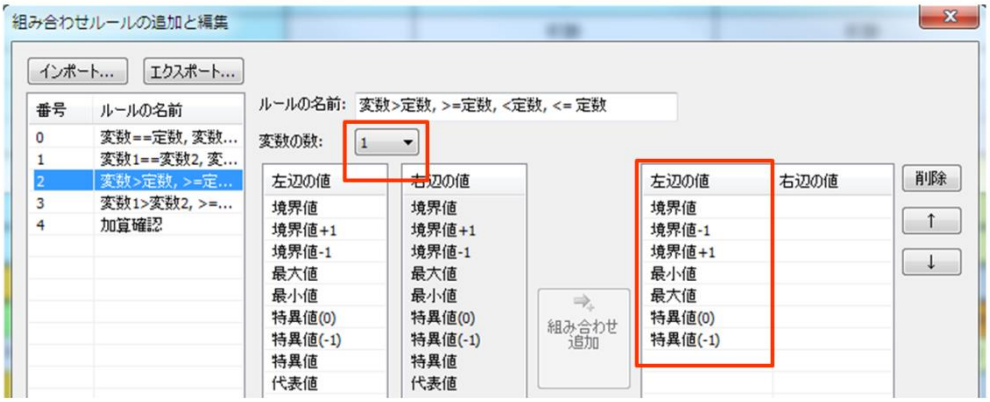
また、作成したルールの適用時には、組み合わせに指定した属性のデータが、入力データ分析表に設定されているかが確認されます。例えば、代表値のデータを入力しない状態で、上記の新規ルールを適用すると、組み合わせに必要なデータが不足していることを警告するダイアログが表示されます。

		最大値	150			最大値	150
		最小値	0			最小値	0
						代表値	75

代表値のデータの入力がない



組み合わせルールの追加と編集画面では、「変数の数」を「1」に指定することも出来ます。変数の指定は 1 つであるため、組み合わせは行われず、単に入力したデータがそのまま出力されますが、組み合わせに指定した属性のデータが入力データ分析表に無い場合には、上記の様な警告が表示されます。この様に、「変数の数」を「1」にした場合には、データの不足をチェックする目的で使用します。



まとめ

本チュートリアルでは、要求仕様とテスト項目の対応付けを中心に、代表的な使用方法を紹介しました。今回体験頂いた内容をベースにして、工数削減やテスト設計品質の向上など、現状抱えている課題を解決できるかどうかを検討頂きたいと思います。

また、既に単体テストのワークフローを構築している場合には、テストケース分析エディタによる効果を活した設計を実現するには、既存のワークフローをどのように変更すべきかについても検討の必要があります。

このテストケース分析エディタは、他の単体テストツールにはない、要求仕様に基づくテスト設計支援機能です。開発する組込みソフトウエアの品質を確認する標準ツールとして、ご活用をお願い致します。

【応用編】埋め込みコードによるカバレッジ計測

はじめに

カバレッジマスターwinAMS は製品に実装するコードと同じ「実コード」を使用して C0、C1 カバレッジを計測できることは、実習 1 から4までで学習しました。しかしながら、実際に使用するクロスコンパイラの最適化の影響により、C ソースコードの論理構造と、生成されるアセンブラコードの構造が一致せず、正確に C0、C1 カバレッジが計測できないケースがあります。また、分岐の網羅だけでなく、全ての複合条件式を網羅して実行が行えたかを評価する MC/DC 計測については、オリジナルのソースコードをコンパイルしたオブジェクトコードからは、原理的に計測が行えません。

そこで、カバレッジマスターwinAMS には、クロスコンパイラの最適化の影響を回避して C0、C1 カバレッジを計測するため、また、MC/DC カバレッジに必要な複合条件式の網羅を評価するために、「埋め込みコード」によるカバレッジ測定機能がサポートされています。

この応用編では、埋め込みコードによるカバレッジ計測の原理、環境作成、使用方法について学習します。本章は、前ページまでの実習を終了し、基本的な使用方法について学習済みの方を対象としています。

※MC/DC 計測機能を使用するには、「MC/DC オプション」のライセンスが必要です。埋め込みコードを使用した C0、C1 計測は、標準でサポートされています。

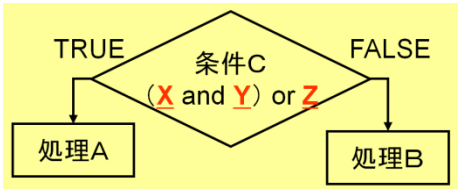
MC/DC とは？（予備知識）

カバレッジマスターwinAMS の埋め込みコードによるカバレッジ計測機能を使用するにあたり、予備知識として MC/DC の概要について解説します。C0、C1 計測が対象の場合も、参考としてご一読ください。

複合条件を網羅するコンディションカバレッジ

一般にブランチカバレッジ（C1）よりも網羅性の高いカバレッジ指標として、コンディションカバレッジ（C2）があります。ブランチカバレッジは、単に分岐の TRUE/FALSE が少なくとも 1 回実行されたかを評価するものですが、条件式が複合条件の場合に、この中の各条件式の論理の組合せについては網羅されていません。これでは網羅度が十分でないと判断される場合に、各条件式の論理のすべての組合せをテストするコンディションカバレッジが用いられます。

例えば、X、Y、Z の条件式をもつ複合条件が分岐にある場合、各条件式の論理の組合せは8通りとなり、コンディションカバレッジでは、このすべてのケースをテストすることが求められます。



条件式 X	F	F	F	F	T	T	T	T
条件式 Y	F	F	T	T	F	F	T	T
条件式 Z	F	T	F	T	F	T	F	T

コンディションカバレッジに必要な条件式の論理の組合せ

MC/DC テストケースの決定方法

しかしながら、コンディションカバレッジ（C2）で各条件式の論理のすべての組合せを考える場合、この組合せには無効なテストケースが発生します。例えば、上記の複合条件式の例では、（X and Y） の論理が TRUE になった場合、Z の論理にかかわらず、その時点で （X and Y） or Z 全体の論理は TRUE に決定してしまうため、上記論理表の右端2つの論理の組合せ（グリーンで囲んだ論理）を両方テストすることには意味はなく、どちらか1つで良いことになり

ます。
そこで、コンディションカバレッジ(C2)を含む意味のない論理の組合せを使用せず、各条件式1つ1つに着目して、その論理の変化による効果を確認する手法として考えられたものが、MC/DC(Modified Condition/Decision Coverage)です。
MC/DC のテストケースは以下の様にして導出します。

> 複合条件式の中から1つの条件式に着目し、この条件式の論理のみを変化させた時に、複合条件式全体の論理が変化する論理の組合せを2つ取り出し、着目した条件式のテストケースとする

例えば、条件式 X に着目した場合、Y,Z は変化させず、X のみを TRUE/FALSE に変化させた場合に、複合条件式全体の論理が変化する論理の組合せを条件式 X のテストケースとして決定します。
この条件を満たす論理の組合せは、例えば以下が考えられます。

(X, Y, Z) → (FALSE, TRUE, FALSE) と (TRUE, TRUE, FALSE)

続けて、Y、Z の論理についても同様に論理の組合せを導出します。最終的には、(X and Y) or Z を MC/DC 評価するために必要なテストケースは、下の論理表の中で、グリーンで塗られた4つのテストケースに決定されます。グリーン以外のテストケースは、MC/DC 評価においては、テストする必要のない論理の組合せです。

条件式 X	F	F	F	F	T	T	T	T
条件式 Y	F	F	T	T	F	F	T	T
条件式 Z	F	T	F	T	F	T	F	T
全体論理	F	T	F	T	F	T	T	T

2つのデータ組み合わせで、
各条件式を評価→ Z X Y

埋め込みコードによるカバレッジ計測の仕組み

埋め込みコードを使用した MC/DC 計測

MC/DC は、前述のようにテストケースを決定して評価しますが、これを行う場合、あるテストケースを与えた場合に、条件文の中に含まれる複合条件式の各条件式の論理が、TRUE/FALSE のどちらで実行されたかを計測する必要があります。
例えば以下の様な条件式を関数が含んでいる場合、この複合条件式を実行した際に、

```
if( ( x>10 && y>20 ) || z>30 )
```

x>10、y>20、z>30 の各条件式の論理が、TRUE/FALSE のどちらで実行されたかを計測する必要があります。
しかしながら、上記の複合条件式をコンパイルしてオブジェクトコードにした場合、このままの状態では、各条件式の論理がどちらで実行されているのかを検出することは原理的にできません。各条件式の実行論理を知るためには、x>10、y>20、z>30 の各条件式を個別に動作させて、それらの実行状態を検出する必要があります。
カバレッジマスターwinAMS の MC/DC 計測は、上記の各条件式の実行論理を検出するために、各条件式の論理を別の関数(以下、フック関数)へ引数で送り、この関数に送られた引数の値をカバレッジマスターwinAMS で判断する方法を採っています。具体的には、上記の複合条件式を下の様なコードに変更して実行します。

```
if(Hook(Hook(x>10) && Hook(y>20) ) || Hook(z>30) )
```

Hook()関数の中では、引数の値をカバレッジマスターwinAMS に送信し、そのときの条件式の論理をカバレッジマ

スターwinAMS が判断できるようにしています。MC/DC 計測の際には、この様なフック関数をテスト対象のコードに組み込むことが必要です。フック関数のソースコードは、カバレッジマスターwinAMS が自動生成します。

```
int Hook( int condition )
{
    [condition  の値をカバレッジマスターwinAMS に送信する仕組み];
    return condition; // 引数に送られた論理結果をそのまま返す
}
```

カバレッジマスターwinAMS では、元の評価対象の関数に、カバレッジ計測のためのコードを追加したコードのことを、「埋め込みコード」と呼んでいます。この埋め込みコードは、CasePlayer2 によるソースソースコード解析の機能により、自動生成されます。

埋め込みコードを使用した C0、C1 カバレッジ計測

C0、C1 カバレッジについても、上記と同様なフック関数をコードに挿入した埋め込みコードを使用することで、最適化による影響を受けることなく、C0、C1 カバレッジの計測が可能となります。まず、C1 カバレッジの計測例について説明します。

例えば、switch 文において、以下の様なコードがあるとします。

```
switch (mode){
    case 0:
        gb_out = 10;    // 代入処理
        break;
    case 1:
        ct+;
        break
    case 2:
        gb_out = 10;    //代入処理
        break;
    default:
        break;
}
```

このコードをコンパイルする際には、クロスコンパイラは、case 0 と case 2 の処理が同じであるため、この2つ処理を1つにまとめてコードを作成し、コードサイズを小さくする事があります。C 言語のイメージでは、以下の様なコード構造で、アセンブラコードが作成されます。

```
switch (mode){
    case 0:
    case 2:
        gb_out = 10;   //代入処理が1つにまとめられる
        break;
    case 1:
        ct+;
        break
    default:
        break;
}
```

この場合、case 0 と case 2 どちらの条件も同じコードが対応しているため、コンパイルされたオブジェクトコードを実行してカバレッジ計測を行うカバレッジマスターwinAMS は、原理的に2つの分岐を区別して検出することができず、正確なカバレッジ計測が行えない結果となります。

そこで、どの行が実行されたかを正確に計測するために、各分岐位置に足跡を残すフック関数（FootPrint 関数）を

挿入したコードを生成します。挿入後のイメージは以下の通りです。

```
switch (mode){
    case 0:
        FootPrint(1); gb_out = 10;    // 代入処理
        break;
    case 1:
        FootPrint(2); ct+;
        break
    case 2:
        FootPrint(3); gb_out = 10;    //代入処理
        break;
    default:
        FootPrint(4); break;
}
```

このコードの実行時に、フック関数 FootPrint () に送られる引数（分岐番号）を検出することで、どの分岐が実行されたかが正確に計測できる仕組みです。フック関数を挿入したソースコードは、カバレッジマスターwinAMS が自動生成します。

上記の例は C1 カバレッジ計測の場合ですが、分岐だけで無く全てのソース行に足跡を残すフック関数を挿入することで、最適化によるコードの欠落などの影響を受けることなく C0 カバレッジを計測する機能もサポートされています。

ターゲットコードに忠実なテストの品質を維持する仕組み

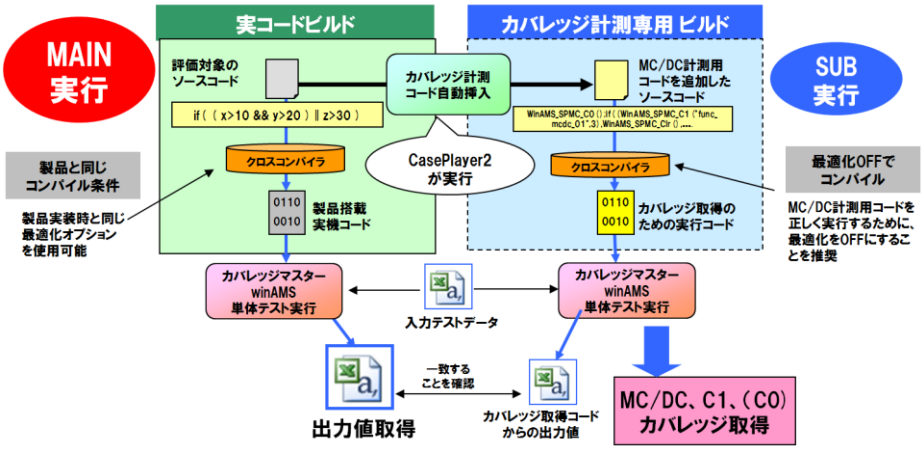
埋め込みコードによるカバレッジ計測を行うためには、前述の様にカバレッジ計測のためのフック関数を組み込んだ「埋め込みコード」を使用する必要があります。ただし、このコードには本来のテスト対象関数にフック関数が追加されてしまっているため、製品に組み込む実際のターゲットコードとは異なってしまいます。これでは、カバレッジマスターwinAMS のアドバンテージである、ターゲットコードに忠実なテストを行うための「実コードを使用した単体テスト」が行えなくなってしまいます。

そこで、可能な限りターゲットコードに忠実なテストの品質を維持するために、カバレッジマスターwinAMS の埋め込みコードを使用したカバレッジ計測においては、テスト対象のソースに手を加えない「実コード」と、カバレッジ計測のための「埋め込みコード」の両方を同時実行して、両者を比較しながら、信頼性の高いテスト結果を抽出する仕組みを持っています。

実コードと埋め込みコードの両方を同時実行

以下の図は、カバレッジマスターwinAMS の埋め込みコードによるカバレッジ計測の仕組みを示しています。左側の「実コードビルド」は、製品開発のビルド環境そのものです。これに加えて、埋め込みコードを適用した「カバレッジ計測専用ビルド」を別途作成します。実コードのビルド環境をフォルダごと複製し、複製したビルド環境にカバレッジ計測のためのフック関数を埋め込みます。埋め込みコードは、CasePlayer2 がソースコードの解析に基づいて、自動生成します。

埋め込みコードも実コードと同様に、同じクロスコンパイラでコンパイルを行い、実行可能なオブジェクトコードにします。



関数の出力値は「実コード」から、カバレッジ結果のみ「埋め込みコード」から取得

カバレッジマスターwinAMS で単体テストを実行する際には、「実コード」と「埋め込みコード」の両方にテストケースを与えて実行します。各関数の出力値は、テストの忠実度、信頼性が最も高い「実コード」から取得し、埋め込みコードからしか取得できないカバレッジ結果のみを「埋め込みコード」から取得します。この両者を総合して、全体のテスト結果としています。

「実コード」の実行は、カバレッジマスターwinAMS のマイコンシミュレータ機能により、命令レベルで実行され、実機と等価な実行結果を取得できます。「埋め込みコード」は、同じクロスコンパイラでコンパイルされ、同じマイコンシミュレータで実行されますが、カバレッジの実行結果は、埋め込まれたコードの論理動作(C ソースレベルの動作)により情報を取得します。「実コード」には製品搭載のコードをビルドする際と同じコンパイル条件(最適化など)を適用すべきですが、それに対して、「埋め込みコード」には、カバレッジ計測の仕組みを正しく動作させるために、最適化のオプションは OFF にしてビルドする事を推奨しています。

テストに埋め込みコードの影響がないことを確認する機能

カバレッジ計測のために追加したフックコードが、関数本来の機能に影響を与えていないことを確認するために、出力条件に設定された変数の出力値が、「実コード」と「埋め込みコード」で一致していることを確認する機能を持っています。

全てのテストケースにおいて、出力値が一致していることをもって、埋め込まれたコードが関数の分岐や演算などの関数本来の機能に影響を与えておらず、計測結果の信頼性が維持されていることを確認できます。

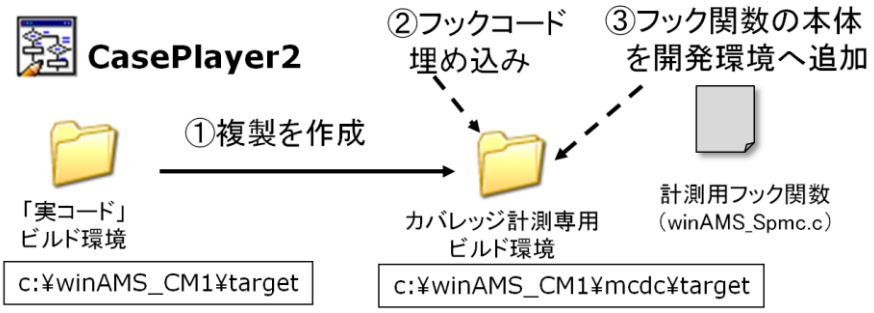
埋め込みコードによるカバレッジ計測環境の構築

ここからは、実習4で作成したテスト環境を元にして、埋め込みコードによるカバレッジ計測環境を追加構築する方法について解説します。

カバレッジ計測専用ビルド作成の流れ

埋め込みコードによるカバレッジ計測のために、実習で作成した「実コードビルド環境」を複製し、「カバレッジ計測専用ビルド」を作成します。以下の様な手順で行います。

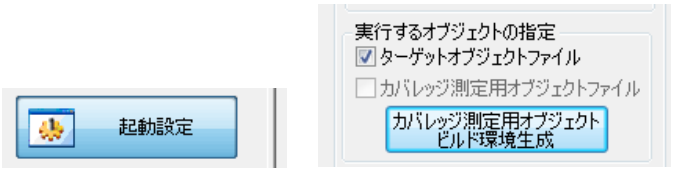
1. CasePlayer2 の機能を使用して、「実コード」開発環境をフォルダごと複製する。
2. 複製された開発環境のソースコードに、CasePlayer2 がフックコードを埋め込む。
3. CasePlayer2 により生成されるフック関数の本体を含むソースファイルを開発環境に加える。
4. 複製した環境でビルドし、実行可能なカバレッジ計測用オブジェクトコードを作成する。



「実コード」開発環境をフォルダごと複製

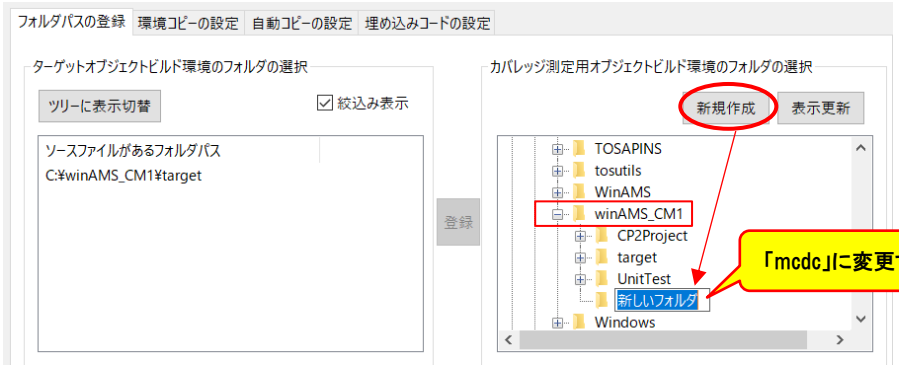
カバレッジ計測専用のビルド環境を、実コードのビルド環境をフォルダごと複製して作成します。複製には、CasePlayer2 の「カバレッジ測定用オブジェクトビルド環境生成」の機能を使用します。

複製されるフォルダは、実コードのビルド環境のフォルダと同じフォルダ名（実習では、「target」フォルダ）になります。そのため、同じフォルダに複製を作成する事はできないため、「C:¥winAMS_CM1」の中に「mcdc」のフォルダを作成し、この中に複製します。（このフォルダ名は任意の名称を付けられます。）



1. 「起動設定」ビューの「カバレッジ測定用オブジェクトビルド環境生成」ボタンを押します。
2. 「カバレッジ測定用オブジェクトビルド環境フォルダの選択」ツリーで、「C:¥winAMS_CM1」を選択して、「新規作成」ボタンを押してフォルダを作成します。フォルダ名を「mcdc」に変更します。

「C:¥winAMS_CM1」フォルダ内に、「mcdc」フォルダが作成されました。この 1,2 の工程の代わりに、Windows エクスプローラ上でフォルダを作成しても構いません。



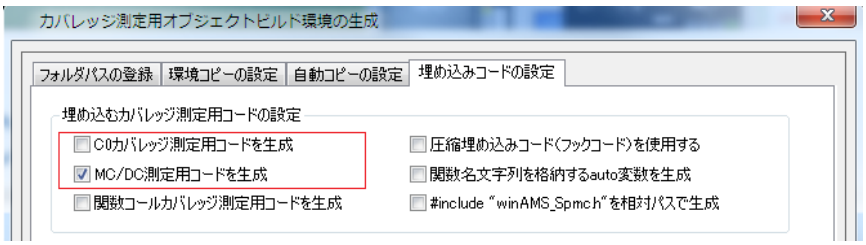
3. 作成された「mcdc」フォルダを選択します。
4. 左の「ソースファイルがあるフォルダパス」に表示されている「C:¥winAMS_CM1¥target」を選択します。
5. 中央の「登録」ボタンを押します。下のリストにパスが登録されます。

実コードビルド環境フォルダと、カバレッジ計測専用ビルド環境フォルダが CasePlayer2 に登録されました。

次に、埋め込みを行うカバレッジの種類を確認します。

6. カバレッジ測定用オブジェクトビルド環境の生成 ダイアログの「埋め込みコードの設定」タブを選択します。
7. MC/DC 計測を行う場合は、「MC/DC 測定用コードを生成」のオプションが ON になっていることを確認します。C1 カバレッジのみの場合は、このオプションは OFF にします。(C1 カバレッジの埋め込みコードは、このオプションの ON/OFF に係わらず、デフォルトで出力されます。)

「MC/DC 測定用コードを生成」のオプションを ON にした場合は、C1 カバレッジも「埋め込みコード」実行結果から取得されます。(MC/DC 測定時に、C1 カバレッジを「実コード」から取得することはできません。)
C0 カバレッジに関しては、「実コード」から、または「埋め込みコード」から取得する方法を選択できます。「C0 カバレッジ測定用コードを生成」のオプションを ON にすると、「埋め込みコード」から C0 カバレッジが取得されます。



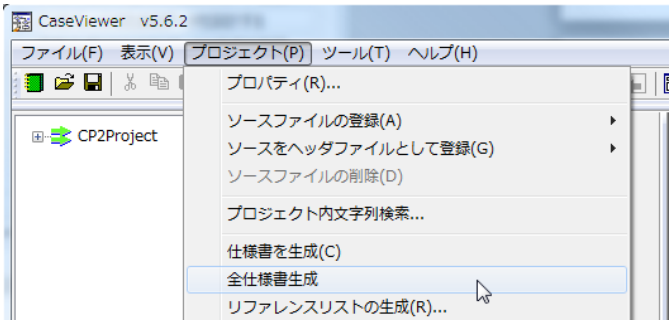
8. C0 カバレッジを埋め込みコードを使用して計測する場合は、「C0 カバレッジ測定用コードを生成」のオプションを ON にします。
9. カバレッジ測定用オブジェクトビルド環境の生成 ダイアログの下にある「環境コピー」ボタンを押します。mcdc フォルダの中に、target フォルダのコピーが作成されます。

フックコードを埋め込む

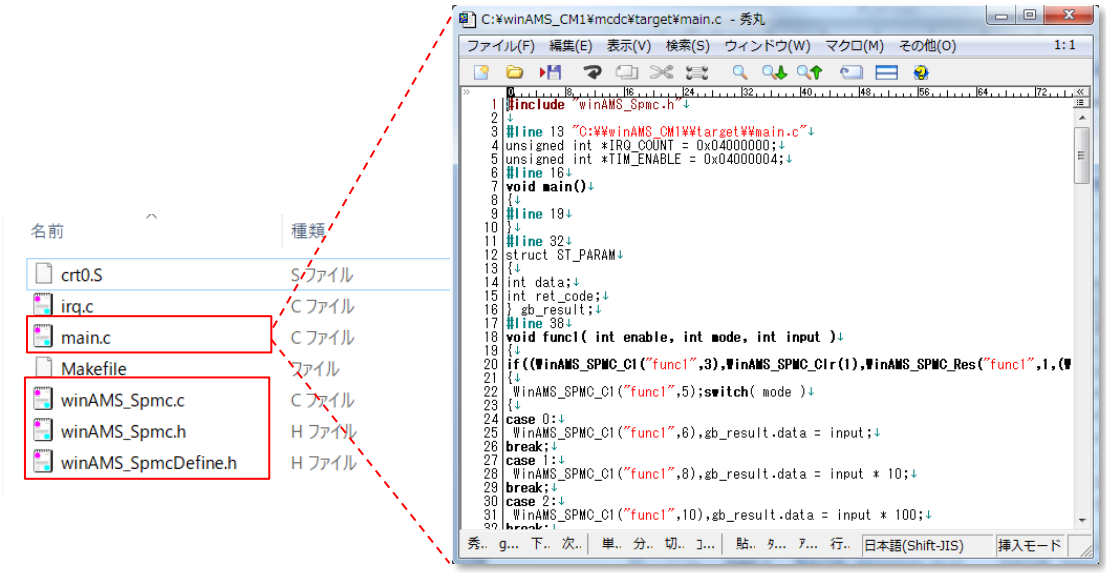
複製されたカバレッジ計測専用ビルド環境フォルダのソースコードに、カバレッジ計測用のフックコードを埋め込みます。この工程は、CasePlayer2 により行われます。

1. CasePlayer2 の「仕様書生成の設定」ダイアログ下部の「埋め込みコード生成」をチェックします。

2. 「OK」ボタンを押してダイアログを終了します。
3. 次に、CasePlayer2 の「プロジェクト」メニューから「全仕様書生成」を選択します。



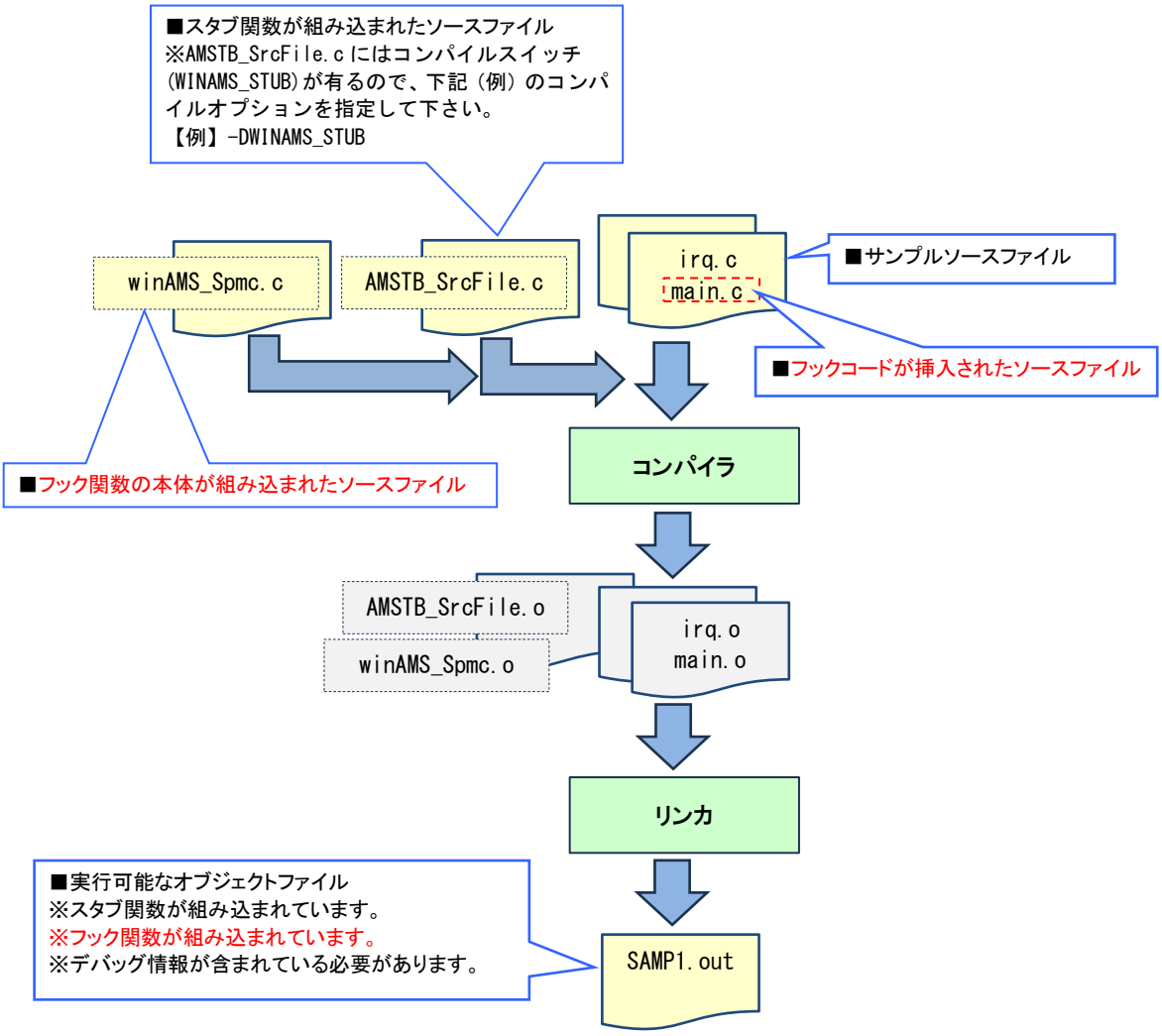
複製された環境のソースに、フックコードが書き込まれます。また、同時にフック関数の本体を含むソースファイル（winAMS_Spmc.c）が生成されます。



フック関数の本体を含むソースファイル（winAMS_Spmc.c）と フックコードが挿入された main.c

フック関数の本体を含むソースファイルをコンパイル

埋め込みコード生成時に、同時に作成されたフック関数の本体が組み込まれたソースファイル(winAMS_Spmc.c)を組み込んで、ご利用のコンパイラ(開発環境)でビルドを行い、実行可能なオブジェクトファイルを作成します。



以上の工程で、カバレッジ計測専用のビルド環境が構築できました。

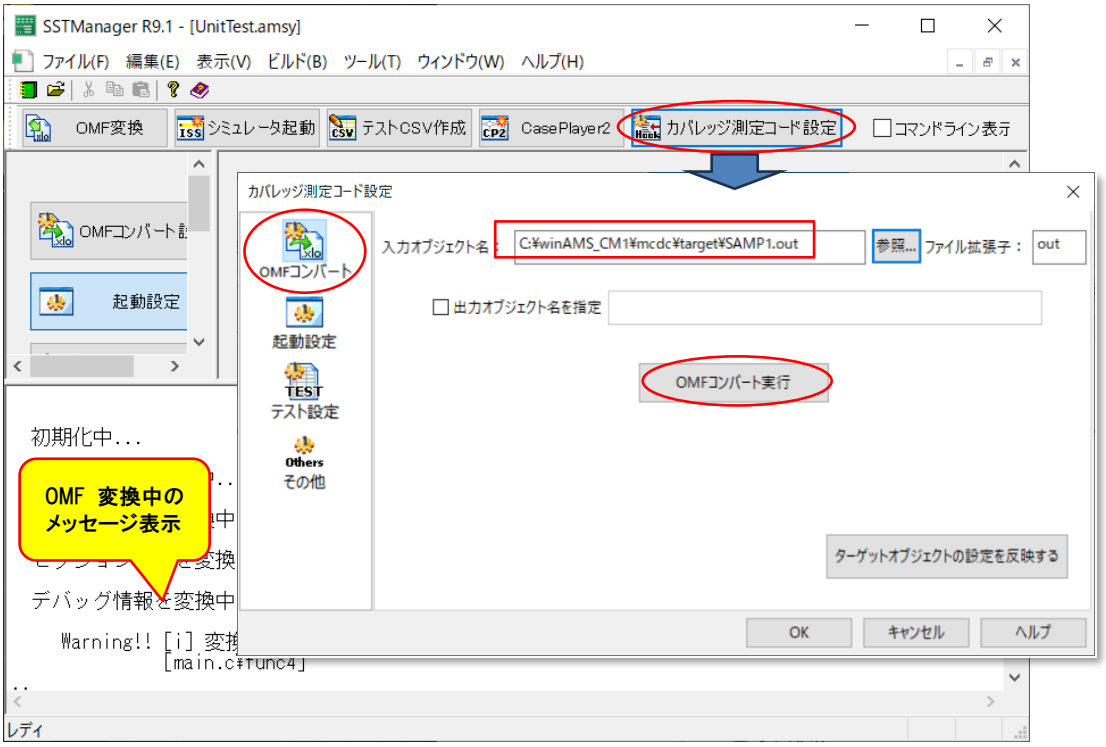
埋め込みコードによるカバレッジ計測を行う

では、カバレッジ計測専用ビルド環境でコンパイルしたカバレッジ計測用オブジェクトコードを使用して、MC/DC 計測、C0/C1 カバレッジ計測を行うための設定を SSTManager で行います。

カバレッジ計測用オブジェクトの OMF 変換と起動設定

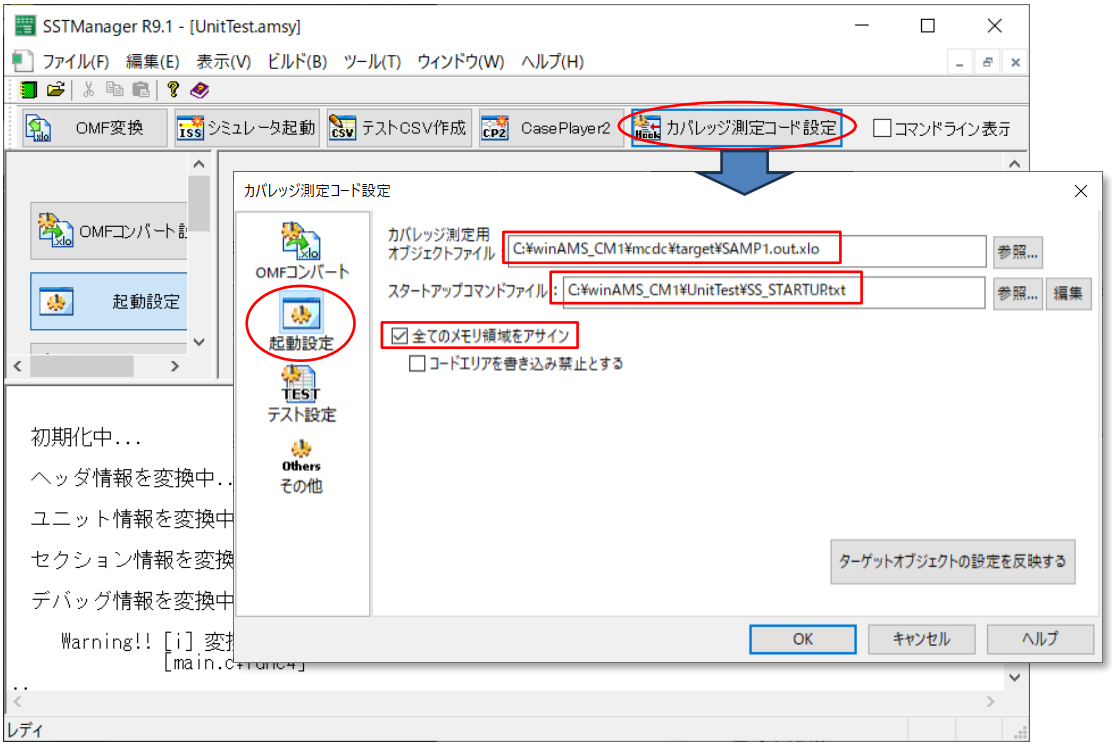
作成したカバレッジ計測用オブジェクトを「OMF 変換」します。

1. SSTManager の「カバレッジ測定コード設定」ボタンを押します。
2. 左側の「OMF コンバート」ボタンを押します。
3. 「入力オブジェクト名」の項目に、複製した埋め込みコードのフォルダに生成されたカバレッジ計測用オブジェクトコード「C:¥winAMS_CM1¥mcdc¥target¥SAMP1.out」を指定します。　※「実コード」環境のフォルダと間違えないようにしてください。
4. 「OMF コンバート実行」ボタンを押します。



以上の操作で、カバレッジ計測用オブジェクトを OMF 変換できました。

5. 次に、「起動設定」ボタンを押します。



「カバレッジ測定用オブジェクトファイル」の項目に、OMF 変換後のカバレッジ計測用オブジェクトファイル(C:¥winAMS_CM1¥mcdc¥target¥SAMP1.out.xlo)が登録されていることを確認してください。

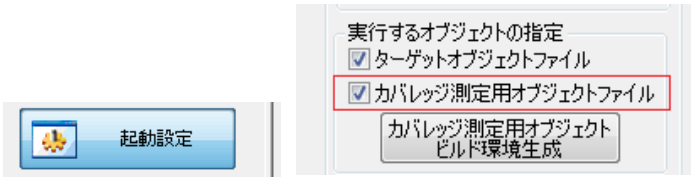
「スタートアップコマンドファイル」の項目に、「C:¥winAMS_CM1¥UnitTest¥SS_STARTUP.txt」が登録されていることを確認してください。
※スタートアップコマンドファイルは、「実コード」の実行と「埋め込みコード」の実行で、別のファイルを設定してもかまいません。

6. 「全てのメモリ領域をアサイン」をチェックします。
7. 「OK」ボタンを押して完了します。

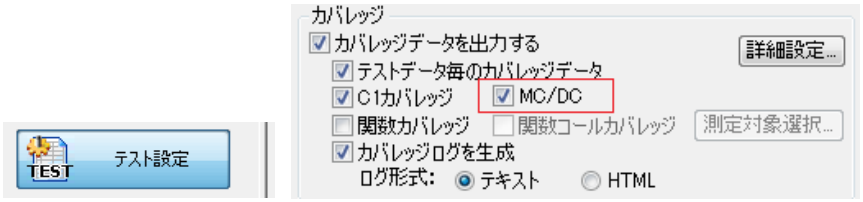
カバレッジ計測用オブジェクトの設定を追加する

SSTManager にカバレッジ計測用オブジェクトの設定を追加します。

1. 「起動設定」ビューの「カバレッジ測定用オブジェクトファイル」をチェックします、これにより、埋め込みコードがシミュレータで実行されます。



2. 「テスト設定」ビューの「MC/DC」をチェックします。これにより、MC/DC 計測結果が出力されます。C0/C1 カバレッジのみの場合は、OFF にします。

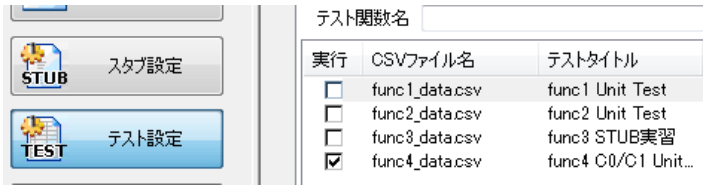


以上で、カバレッジ計測用オブジェクトによるカバレッジ測定に必要な設定は終了です。

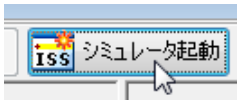
埋め込みコードによるカバレッジ計測を実行する

では、埋め込みコードによるカバレッジ計測を実行して結果を確認してみます。テストには、複合条件式を持つ func4()を選択します。

1. 「テスト設定」のボタンを押します。
2. テスト CSV 一覧のタブで、「func4_data.csv」の実行ボックスをチェックします。



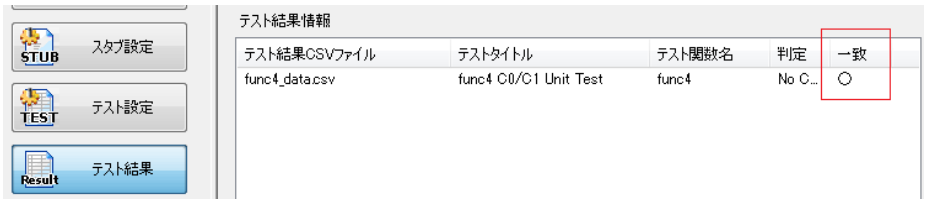
3. 「シミュレータ起動」ボタンを押します。



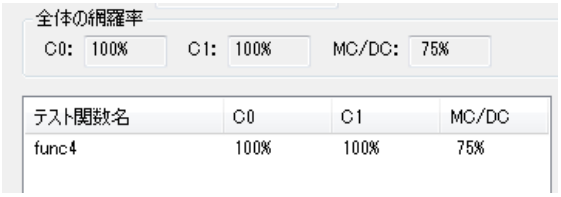
これにより、「実コード」実行、「カバレッジ計測用コード」実行の順で、シミュレータが 2 回自動起動し結果を出力します。

埋め込みコードによるカバレッジ計測結果を確認する

テスト結果ビューに、func4()のテスト結果が表示されます。右端に表示される「一致」の○印は、実行した全てのテストケースにおいて、「実コード」と「カバレッジ測定用コード」の出力変数の値が一致していることを示しています。これにより、埋め込みコードにより、関数本来の分岐などの動作に影響を与えていないことが確認できます。



カバレッジビューには、C0、C1、MC/DC のカバレッジ結果が表示されます。(MC/DC を計測していない場合は、表示されません。)このサンプルでは、C0、C1 カバレッジの網羅率は 100%ですが、MC/DC の網羅率のみが 75%で NG となっています。「func4」の行をダブルクリックすることで、カバレッジビューが表示されます。



MC/DC は、各条件式がすべて TRUE/FALSE で実行されているかどうかを評価します。このサンプルでは、204 行目の if 文全体の TRUE/FALSE 論理は実行され、各々の分岐は網羅しているため、C1 カバレッジは 100%となっていますが、複合条件式のうち「gb_c>30」については、FALSE 条件が実行されていないことが分かります。

MC/DC の網羅率は、条件式の総個数のうち、TRUE/FALSE の両方が実行された条件式が何個あるかで算出されます。

以上で、埋め込みコードによるカバレッジ測定に関する解説は終了です。

埋め込みコードによるカバレッジ計測適用時の作業フローについて

埋め込みコードによるカバレッジ計測環境が完成した後、テスト対象のソースコードが変更され後、カバレッジ計測を再実行するために必要な作業手順をまとめておきます。

テスト対象のソースコードが変更されたら

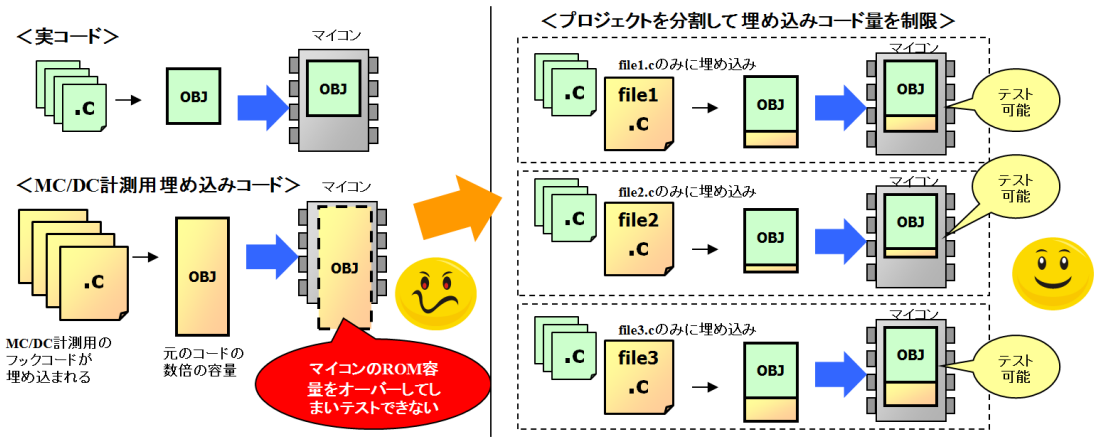
テスト対象のソースコードが変更された場合、それに伴い、「埋め込みコード」についても更新を行う必要があります。マイコンシミュレータで実行するために、オブジェクトコードの再ビルドも必要です。以下の様な手順です。

1. 「実コード」ビルド環境でオブジェクトをコンパイルする。(使用中の開発環境、IDE を操作)
2. CasePlayer2 の「全仕様書生成」を実行し、「埋め込みコード」を再生成する。
3. 生成した「埋め込みコード」を「カバレッジ計測専用ビルド環境」でコンパイルする。(使用中の開発環境、IDE を操作)
4. SSTManager の「シミュレータ起動」ボタンで、テストを再実行する

以上で、埋め込みコードによるカバレッジ計測に関する実習は終了です。

【参考】埋め込みコードによるコードサイズ増加

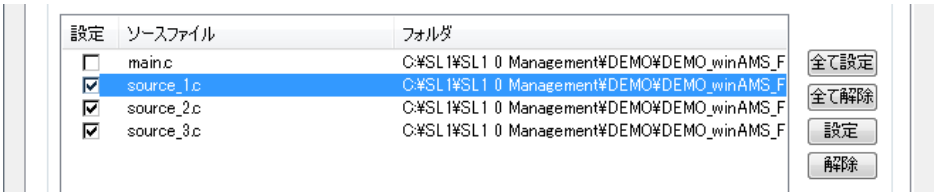
MC/DC 計測のために「フックコード」を埋め込むことで、コンパイル後のオブジェクトのサイズは増加します。全てのコードにフックコードを埋め込んでしまうと、数倍のオブジェクトサイズになる事があります。
この場合、使用している型番のマイコン ROM 容量をオーバーしてしまい、テストができなくなってしまいます。



これを回避するための方法は以下の通りです。

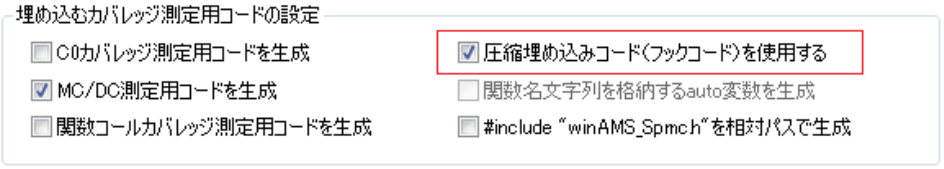
1) 埋め込みコードの適用範囲を制限する

埋め込みを行うコードは、ソースファイル単位に選択できます。テスト対象の関数が含まれるソースファイルのみに埋め込みを行う事で、オブジェクトコードサイズの増加を回避することができます。
CasePlayer2 → [プロジェクトメニュー] → [カバレッジ測定用オブジェクトビルド環境生成]にて、「埋め込みコードの設定」タブで、テスト対象の関数があるソースのみを選択してください。



2)圧縮埋め込みコードを使用する

また、V3.6 以降のバージョンには、埋め込みコードのサイズを小さくするオプション「圧縮埋め込みコード(フックコード)を使用する」が利用できます。マイコンやコンパイラにより縮小できるサイズは異なりますが、このオプションをONにすることで、埋め込みコードの冗長性を抑えて、生成オブジェクトコードを小さくします。



3)使用しているマイコンと同じシリーズの別型番で、ROM 容量の大きい型番でコンパイルする

同じシリーズのマイコンで、実際のマイコンよりもROM容量の大きなマイコンが有る場合であれば、埋め込みコードのコンパイル環境のみ、ROM 容量の大きな型番を指定して、埋め込みコードのオブジェクトを生成する回避方法があります。

埋め込みコードからの計測の仕組みは、マイコンのオブジェクト構造には依存せず、埋め込んだフック関数の機能により計測されます。このため、実際のマイコンと異なる型番で実行した場合でも、そのコードが実行可能であれば、MC/DC テスト結果には影響を与えません。

実コード(製品実装と同じ)のコンパイル環境はそのままにして、埋め込みコードのコンパイル環境のみに上記のROM 容量の大きい型番を選択してオブジェクトを生成してください。関数実行後の出力変数の値の評価は、埋め込みコード側ではなく、並行して動作する実コードの方から取得されますので、カバレッジ計測以外の単体テスト結果には影響はありません。

4)上記の回避方法を使用しても ROM 容量を超えてしまう場合は、オブジェクトを調整

1つのソースファイルを埋め込み対象にしただけでも、オブジェクトコードの ROM 容量をオーバーしてしまう場合は、テストに関係しないオブジェクトをリンクから外すなどの方法で、テスト実行に影響しない範囲でオブジェクトを小さくするしか、回避方法は有りません。

【応用編】関数カバレッジ、コールカバレッジ計測

はじめに

ここからは、カバレッジマスターwinAMS の関数カバレッジ、コールカバレッジ計測機能について解説しています。本章には実習はありません。

※関数カバレッジ、コールカバレッジ計測機能は、カバレッジマスターwinAMS V3.6 でサポートされた機能です。使用するためには、「関数/コールカバレッジ計測オプション」のライセンスが必要です。

カバレッジマスターwinAMS で行う結合テスト

カバレッジマスターwinAMS で行う C0、C1、MC/DC のカバレッジ計測は、「単体テスト」のフェーズで行われるテストに当たりますが、この章で扱う関数カバレッジ、コールカバレッジ計測は、「結合テスト」のフェーズで求められるテストです。

特に自動車機能安全規格 ISO26262 の「Part.6-10 Software integration and testing」では、結合レベルでの構造カバレッジ計測が求められており、この手法に「Function Coverage(1a)」、「Call Coverage(1b)」が規格化されています。

本機能は、ISO26262 に対応した関数カバレッジ、コールカバレッジ計測を効率的に行うためのものです。

単体テストと結合テストの違いを確認

本機能は「結合テスト」に当たりますが、カバレッジマスターwinAMS の基本的な操作方法やテスト CSV ファイルのフォーマットなどは単体テストと同じです。新たにカバレッジマスターwinAMS の操作方法を学習する必要はほとんどありません。

ただし、単体テストとの違いを正しく理解しておく必要があります。主な違いとしては、

- ●単体テストで作成したスタブ関数は使用せず、実際のサブ関数を結合する
- ●機能単位でテストを計画し、最上位の関数にテストケースを与えて駆動する
- ●実際のサブ関数の実行状態を「関数カバレッジ」「コールカバレッジ」の観点で計測する

が挙げられます。テストケースの設計については、

- ●サブ関数の実行を考慮した「機能コンポーネント」単位での結合テスト設計が必要

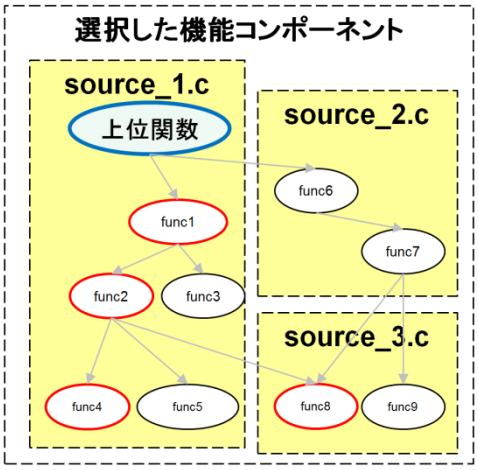
です。結合テストはスタブを使用せず、実際のサブ関数を結合するテストです。CSV ファイルのフォーマットや作成手順は単体テストで使用するものと同じですが、スタブを前提とした単体テストのテストデータをそのまま使用して、今回の結合テストを行う事はできません。

関数カバレッジとは

関数カバレッジとは、ある機能コンポーネントに含まれる全ての関数が、結合テストを通じて「少なくとも 1 回は実行されたか」を確認するものです。例えば、下図の様な機能コンポーネントがあり、上位関数の他にサブ関数（func1～9）がある場合、上位関数を駆動することで、サブ関数（func1～9）の全てが実行されたかを計測します。

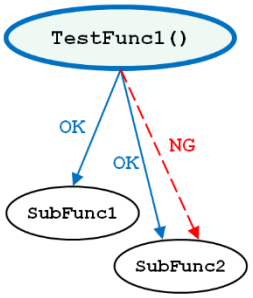
下図には、関数の結合状態を示す矢印がありますが、関数カバレッジはサブ関数が単に「実行されたか」を確認するだけであるため、どの関数からコールされたかは確認されません。例えば func1()の場合、もしも上位関数が対象の機能コンポーネント外の関数をコールし、コールされた外部の関数が func1()を呼んだ場合でも、func1()は実行されたと見なされます。

確かに、機能コンポーネントが含んでいるサブ関数が網羅されて呼ばれているかの指標にはなりますが、関数カバレッジだけでは、矢印が示す結合状態を確認することはできません。



コールカバレッジとは

そこで、矢印が示す結合状態を確認するために、関数間の呼び出しが行われたかを計測するものが、コールカバレッジに当たります。例えば、TestFunc1()から SubFunc1()と SubFunc2()が呼ばれている場合、この呼び出しが全て行われたかを計測します。下の例では、SubFunc2()は 2 カ所でコールされますが、これらは別々に計測します。




```
TestFunc1(){
    if( val == 0)
    {
        SubFunc1(); OK
    }
    else
    {
        SubFunc2(); OK
                :
                :
        if(ctl) SubFunc2(); NG
    }
}
```

コールカバレッジは関数単位に計測されます。TestFunc1()には 3 カ所のサブ関数呼び出しがありますが、この全てがコールされたかを計測します。

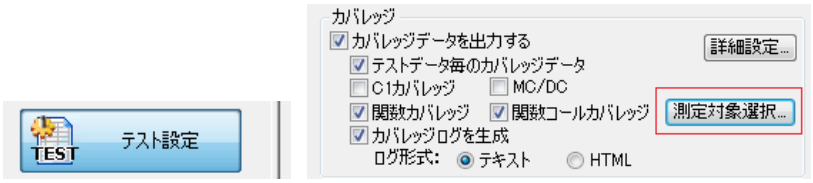
コールカバレッジは、特定の関数から特定のサブ関数が直接呼ばれていることを計測しているため、前述の関数カバレッジでは確認できなかった関数の結合状態を示す矢印を確認することができます。コールカバレッジが網羅されていることを確認すれば、機能コンポーネント内で想定される各関数が、実際に結合状態にあることが確認できることになります。

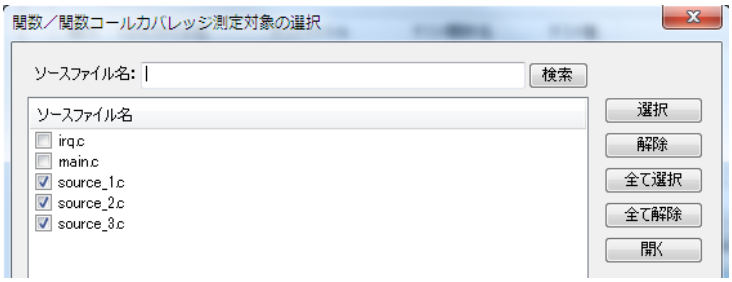
関数カバレッジ、コールカバレッジ計測に必要な設定

測定対象ソースファイルの選択

関数カバレッジ、コールカバレッジの測定対象は、ソースファイル単位で選択が可能です。関数個別の選択は行えません。選択の手順は以下の通りです。

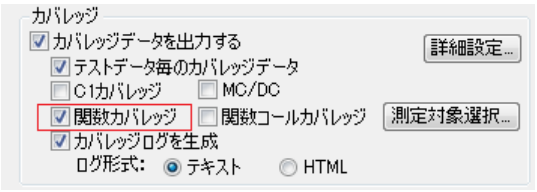
1. 「テスト設定」ビューのカバレッジの項目にある「測定対象選択…」ボタンを押します。
   (関数カバレッジ、コールカバレッジのどちらかのチェックボックスが ON である必要があります。)
2. 測定対象としたいソースファイルのチェックボックスを ON にします。





関数カバレッジ計測の設定

関数カバレッジは、「実コード」の実行から取得できます。必要な設定は、「テスト設定」ビューの「関数カバレッジ」オプションを ON にするだけです。(関数カバレッジのみであれば、「埋め込みコード」の実行は必要有りません。)



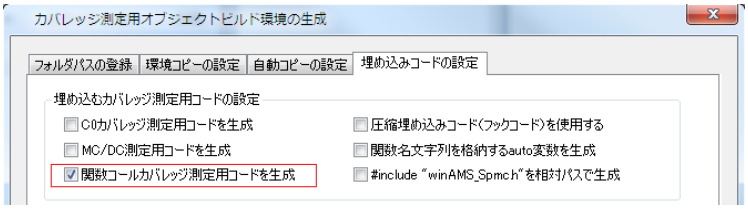
上記の設定後に、SSTManager の「シミュレータ起動」ボタンでテストを実行することで、関数カバレッジが計測できます。

コールカバレッジ計測の設定

コールカバレッジ計測は、「埋め込みコード」を適用する必要があります。MC/DC 計測と同じ方法で、カバレッジ計測専用ビルドを作成し、実コードと合わせて実行する必要があります。(「埋め込みコード」の適用方法については、前述の「MC/DC 計測環境の構築」の章を参照してください。)

MC/DC 計測環境が完成(「埋め込みコード」適用済み)している状態で、以下の手順で計測設定を行います。

1. CasePlayer2 の「プロジェクトメニュー」から、「カバレッジ測定用オブジェクトビルド環境生成」を選択します。
2. 「埋め込みコードの設定」タブで、「関数コールカバレッジ測定用コードを生成」を ON にします。



3. CasePlayer2 の「プロジェクト」メニューから「全仕様書生成」を実行し、「埋め込みコード」を再生成します。
4. 「カバレッジ測定専用ビルド」の環境で、「埋め込みコード」を再ビルドし、オブジェクトコードを更新します。
5. SSTManager の「テスト設定」ビューのカバレッジ項目にある「関数コールカバレッジ」のチェックボックスを ON にします。

上記の設定後に、SSTManager の「シミュレータ起動」ボタンでテストを実行することで、コールカバレッジが計測できます。

関数カバレッジ、コールカバレッジの計測結果

関数カバレッジ、コールカバレッジの計測結果は、HTML、CSV のフォーマットで一覧出力されます。

HTML ファイルでテスト結果を確認

関数カバレッジ、コールカバレッジの測定結果は、テスト実行直後に生成される HTML ファイル「テスト報告書」に出力されます。表示方法は以下の通りです。

1. SSTManager の「テスト結果」ビューで、右上の「報告書を開く」のボタンを押します。
2. 表示される「テスト結果報告書.htm」の下の方に、「関数／関数コールカバレッジ情報」が出力されます。

ファイル名	関数網羅率(ファイル単位)	関数コール網羅率(ファイル単位)	関数名	関数実行	関数コール網羅率(関数単位)	関数コール数	関数コール位置	関数コール実行
source_1.c	83%	75%	fc_cover_test	○	50%	2	23行目：sub_func1	○
							27行目：sub_func6	×
			sub_func1	○	100%	3	35行目：sub_func2	○
							36行目：sub_func3	○
							40行目：sub_func3	○
			sub_func2	○	66%	3	48行目：sub_func4	○
							52行目：sub_func5	×
							56行目：sub_func8	○
			sub_func3	○	N/A	0	-	-
			sub_func4	○	N/A	0	-	-
			sub_func5	×	N/A	0	-	-

■ファイル名： 測定対象として選択したソースファイル名
■関数網羅率(ファイル単位)： ソースファイルにある関数のうち、少なくとも一度実行された関数の割合
（「関数実行」列の○印の個数から計算、 上記では 5/6→83%）
■関数コール網羅率(ファイル単位)： ソースファイルにある関数が持つサブ関数コールの実行網羅率
（「関数コール実行」列の○印の個数から計算、 上記では 6/8→75%）
■関数名： 測定対象の関数名
■関数実行： 「関数名」の関数が少なくとも一度実行されていれば○
■関数コール網羅率(関数単位)： 「関数名」の関数が持つサブ関数コールの実行網羅率
■関数コール数： 「関数名」の関数が持つサブ関数コールの総数
■関数コール位置： 「関数名」の関数が持つサブ関数コールのソース行番号コール関数名
■関数コール実行： 「関数名」の関数が持つサブ関数コールが行われていれば○

この一覧表で、右端の「関数コール実行」の項目が全て○であれば、想定されるサブ関数コールが全て行われていることを示します。これにより、機能モジュールにある全ての関数が、想定した結合状態にあることが確認できます。

【注意】HTML ファイルで出力される「テスト報告書.htm」は一時ファイルです。直前のテスト結果が保存されていますが、テストを再実行すると、再び直前のテスト結果に書き換わります。HTML ファイルをテスト実行毎に別保存する機能はありません。必要な場合は、ユーザー自身で該当 HTML ファイルを待避して保存する必要があります。
HTML の「テスト報告書.htm」は、以下のフォルダに生成されています。

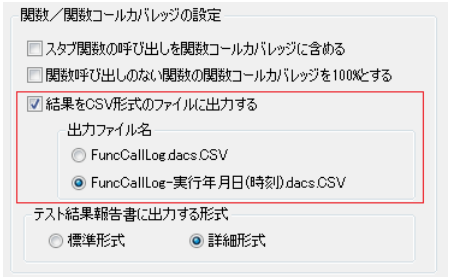
[テストプロジェクトフォルダ]¥Out2013-08-03(10'07'39)
※2013-08-03 はプロジェクト作成日付
※(10'07'39)はプロジェクト作成時間
※保存場所は、「テスト設定」ビューの「テスト結果保存先フォルダ」で変更可能

CSV ファイルでテスト結果を確認

上記の HTML ファイルに出力される「関数／関数コールカバレッジ情報」と同じ内容が、同時に CSV ファイルにも出力されます。
CSV ファイルは、HTML ファイルと同様に直前のテスト結果を保存する一時ファイルとするか、作成日時情報をファイル名に付加してテスト実行毎に別に保存するかを選択できます。

以下の手順で、CSV ファイルの保存方法を選択できます。

1. 「テスト設定」ビューのカバレッジの項目にある「詳細設定…」ボタンを押します。
2. 「カバレッジの詳細設定」ダイアログの「関数／関数コールカバレッジの設定」で、「出力ファイル名」を選択します。



CSV ファイルも前述の HTML ファイルと同じフォルダに保存されます。

	A	B	C	D	E	F	G	H	I
1	全体の関数／関数コールカバレッジ情報								
2	関数網羅率	83%							
3	関数コール網羅率	75%							
4									
5	関数／関数コールカバレッジ情報								
6	ファイル名	関数網羅率(ファイル単位)	関数コール網羅率(ファイル単位)	関数名	関数実行	関数コール網羅率(関数単位)	関数コール数	関数コール位置	関数コール実行
7	source_1.c	83%	75%	fc_cover_tes	○	50%	2	23行目：sub_func1	○
8								27行目：sub_func6	×
9				sub_func1	○	100%	3	35行目：sub_func2	○
10								36行目：sub_func3	○
11								40行目：sub_func3	○
12				sub_func2	○	66%	3	48行目：sub_func4	○
13								52行目：sub_func5	×
14								56行目：sub_func8	○
15				sub_func3	○	N/A	0	-	-
16				sub_func4	○	N/A	0	-	-
17				sub_func5	×	N/A	0	-	-
18									

以上で、関数/コールカバレッジ測定に関する説明は終了です。

カバレッジマスターwinAMS　チュートリアル

ガイオ・テクノロジー株式会社

■ユーザーサポートのご案内
http://www.gaio.co.jp/support/support_about.html

■使用方法に関するお問い合わせ方法
お問い合わせは、ユーザーサポート窓口をご利用ください。
http://www.gaio.co.jp/support/support_entry.html
ユーザーサポート窓口へのお問い合わせには、ユーザーIDが必要です。
※保守契約がない場合は、いかなるサポートも提供致しません。