# 1 ダミー関数

winAMS で検証を行うアプリケーション（C プログラム）は、通常デバッグ用にプログラムを変更する必要はありませんが、一部のマイコン開発環境では、オブジェクトファイルに含まれるデバッグ情報が System-G で取り扱う情報と合わないため、winAMS プロセスから、検証対象とする関数の実行制御（関数 CALL や、関数の終了判断）が不可能な場合や、関数の引き数にテストデータを設定出来ないことがあります。このようなケースでも検証を実現するために、ダミー関数（デバッグ用）による検証手法を提供しています。ダミー関数では while 文等を使い、検証対象とする関数を呼び出す永久ループを作成します。関数 CALL の前には「開始変数」への代入を行います。関数 CALL の後ろには「終了変数」への代入を行います。winAMS は、この「開始変数」「終了変数」への代入を監視して検証対象となる関数の開始と終了を判断します。

## ダミー関数の作成方法

ダミー関数はプログラムソースに直接記述しても構いませんが、実ソースファイルに手を入れることは、品質を保持する上であまり好ましいことではありません。デバッグ用のソースファイル（例: Debug.c）をプログラムソースファイルとは別に作成して、その中にダミー関数を記述することをお勧めします。 下記は、検証対象となる関数 sample()を、ダミー関数を使用して検証する場合の例です。

プログラムソースファイル

```
int x;
char ary[10];
struct {
    int   x;
    int   y;
} table;

int sample (int arg1, int arg2)   /* 検証対象となる関数 */
{
    |
    return 1;
}
```

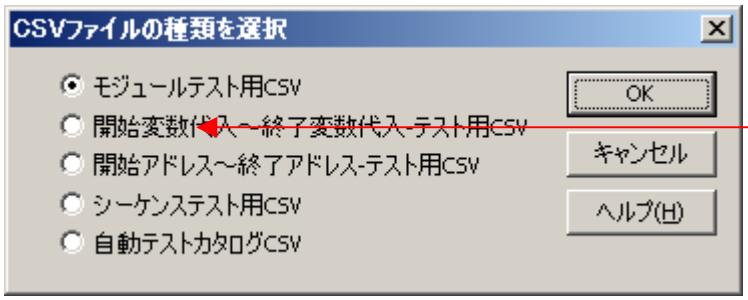
デバッグ用ソースファイル(Debug.c)

```
int   out;                      /* デバッグ用 関数戻り値（代行） */
int   a, b;
int   DBG_IN;                   /* 開始変数 */
int   DBG_OUT;                  /* 終了変数 */
extern   int   sample (int);    /* 検証対象関数のプロトタイプ宣言 */

void dummy ()   /* ダミー関数 */
{
    while (1) {        /* 検証関数 CALL を永久ループ */
        DBG_IN = 1;                 /* 開始変数への代入 */
        out = sample (a, b);        /* 検証する関数を CALL */
        DBG_OUT = 1;                /* 終了変数への代入 */
    }
}
```

## 2 テストデータの作成

ダミー関数を使用するときは「開始変数代入～終了変数代入-テスト用 CSV」を使用します。



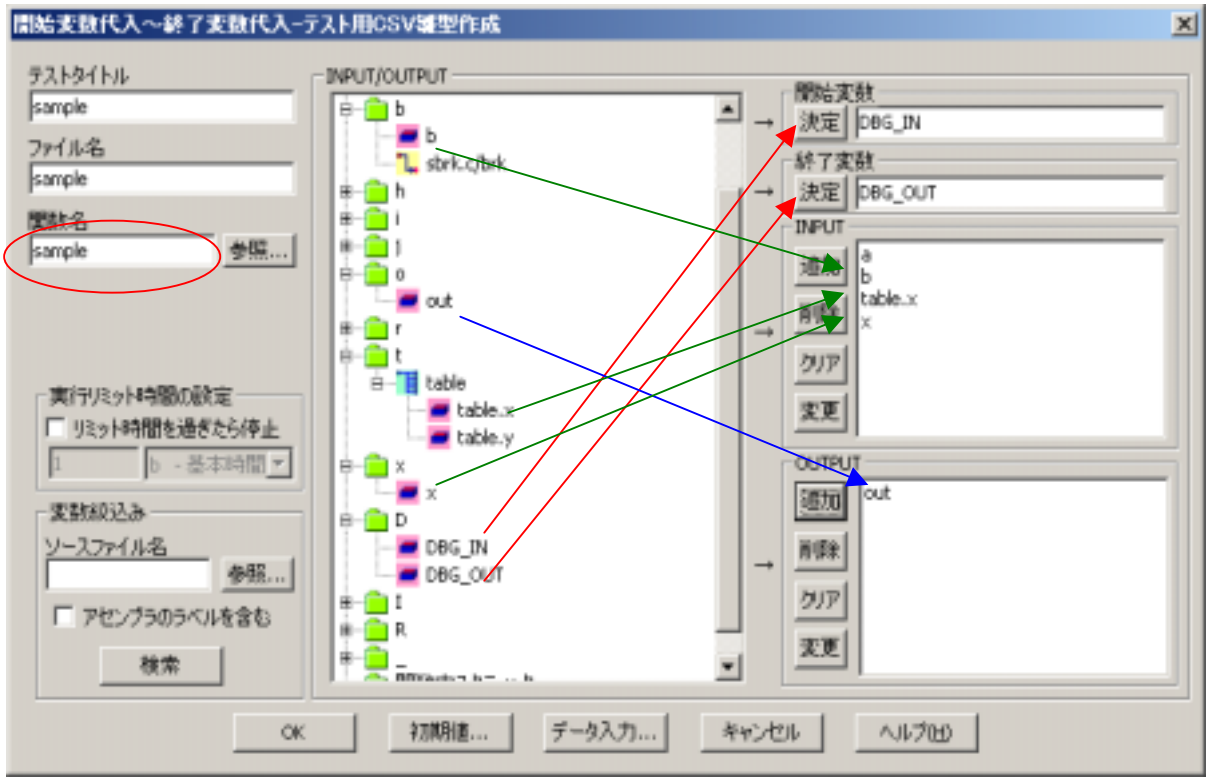

C 言語関数（制限付き）テスト用これを使用します。

### 開始変数代入～終了変数代入-テスト用 csv の作成

「開始変数代入～終了変数代入-テスト用 CSV 雛型作成」ダイアログを使用して、検証する関数名の選択や入出力変数を選択します。

### 操作手順

開始変数として“DBG_IN”を、終了変数として“DBG_OUT”を登録します。これから検証する関数 "sample"を「関数名」ボックスに設定します。”Input/Output”には、アプリケーションで定義している変数シンボルの一覧が表示されていますので、ここから"sample" 関数の入力となる変数シンボルを選択して”Input”へ、出力となる変数シンボルを選択して”Output”へ、それぞれ登録します。 “開始変数代入～終了変数代入-テスト用 csv”の場合は、"sample" 関数の引き数 “argument”を登録できませんので、ダミー関数で定義している代行グローバル変数 “x” を登録します。また、sample 関数の戻り値も登録することができませんので、代行グローバル変数 “out”を登録します。



開始変数代入～終了変数代入-テスト用 csv

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | api | DBG_IN | DBG_OUT | sample2 | 5 | 1 | sample |
| 2 | a | b | ary[1] | table.x | x | out | |
| 3 | 1 | 3 | 10 | 120 | 1 | 1 | |
| 4 | 1 | 4 | 20 | 121 | 1 | 1 | |
| 5 | 2 | 4 | 30 | 122 | 1 | 153 | |
| 6 | 3 | 5 | 40 | 123 | 1 | 164 | |
| 7 | 4 | 6 | 50 | 124 | 1 | 175 | |
| 8 | 5 | 7 | 60 | 125 | 1 | 186 | |
| 9 | 6 | 8 | 70 | 126 | 1 | 198 | |
| 10 | 7 | 9 | 80 | 127 | 1 | 209 | |
| 11 | | | 入力データ | | | 出力データ期待値 | |

１行目:

"api"は、「開始変数代入～終了変数代入-テスト用 CSV」であることを示す識別子です。

"DBG_IN"は、sample 関数の開始確認、デバッグ用変数です。

"DBG_OUT"は、sample 関数の終了確認、デバッグ用変数です。

"sample"は、検証対象となる関数名です。

“sample2”は、テスト名です。

"5"は、入力変数シンボルの数を表しています。

"2"は、出力変数シンボルの数を表しています。

２行目:

"a","b"は、関数の引き数名"arg1","arg2"をダミー関数で代行した変数です。

“ary[1]”,"table.y","x"は、入力変数シンボル名です。

“out”は、関数の戻り値をダミー関数で代行した変数です。

入力データ(３～１０行目):

“a” ～ "x"の列は、関数の入力変数に対する入力データです。

ここには必要とされるテストデータの組み合わせを定義します。

入力データが不要なとき、記述を省略できます。

期待値(３～１０行目):

"out"の列は、出力変数シンボル(関数の戻り値)に対しての検証結果の期待値を定義します。

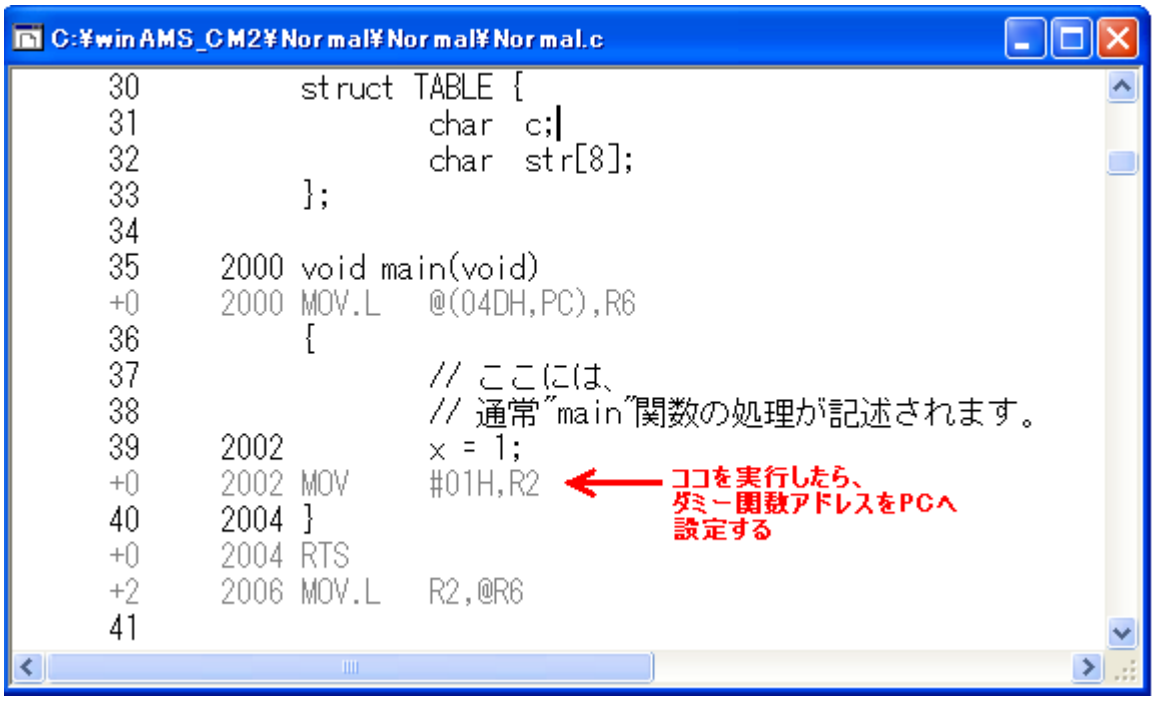
期待値を設定しておくと、シミュレーション結果が期待値通りとなったか？自動判定されます。

期待値は省略できます。

## 3 スタートアップコマンドファイルの定義

デバッグ用に作成したダミー関数は、そのままでは呼び出されることはありません。ダミー関数の呼び出しは、「スタートアップコマンドファイル」の定義により、main 関数先頭から数ステップ実行したところで、強制的にダミー関数へ制御を移行することにより実現します。プログラムを修正することはありません。

main 関数 から +2 バイト目（39 行目）を実行したとき、
ダミー関数アドレスを PC へ設定する場合の例




スタートアップコマンドファイルの定義

```
start log/all
on error then continue
set unit/all
@reset
set mode source

; ダミー関数へ、
; 制御を移行するための定義
macro   ChangeDmy      ; ** “ChangeDmy”マクロの定義 **
  set reg pc = dummy                    ; ダミー関数アドレスを PC へ設定
mend
set do/exec=main+2¥#1 ChangeDmy         ; main 関数の 2002 番地を実行したら
                                        ; “ChangeDmy”マクロを呼び出す。
```

## 4　１つのダミー関数で、複数の関数を検証する方法

ダミー関数は、検証対象とする関数の実行制御（関数 CALL や、関数の終了判断）が不可能な場合や、関数の引き数にテストデータの設定が出来ないなどの制限が有るときの対応策（5.2 で説明）として使用します。 ここでは、複数の関数を続けて検証（１つの関数に１つの CSV を使用）するためのダミー関数の作成方法について説明します。

検証する関数が含まれるＣソース

```
int sample1 (int argument)   /* 検証対象となる関数１  */
{
    |
    return 1;
}

void sample2 (char a)        /* 検証対象となる関数２  */
{
}

int sample3 (int i, int j)   /* 検証対象となる関数３  */
{
}
```

検証対象としたい関数は sample1～sample3 とし、それぞれの関数は任意の引き数を入力し、戻り値を返します。

sample1～sample3 の関数を起動するダミー関数

```
int   no;                          /* sample1～3 現在検証関数 No   */
int   x;                           /* デバッグ用 関数引き数（代行）*/
int   y;                           /* デバッグ用 関数引き数 (代行)  */
int   out;                         /* デバッグ用 関数戻り値（代行）*/
int   DBG_IN;                      /* 開始変数   */
int   DBG_OUT;                     /* 終了変数   */
extern  int   sample (int);    /* 検証対象関数のプロトタイプ宣言 */

void dummy ()   /* ダミー関数 */
{
    while (1) {        /* 検証関数 CALL を永久ループ */
        DBG_IN = 1;          /* 開始変数への代入 */
        switch   (no)   {     /* sample1～3 を認識 */
            case 1:   out = sample1 (x);      /* 検証する関数を CALL */
                      break;
            case 2:   sample2 (x);            /* 検証する関数を CALL */
                      break;
            case 3:   out = sample3 (x, y);   /* 検証する関数を CALL */
                      break;
        }
        DBG_OUT = 1;      /* 終了変数への代入 */
    }
}
```

新たに、sample1～sample3 関数を識別するための変数 "no"を追加して、この変数に設定する値を sample1 は"1"、sample2 は"2"、sample3 は"3" とし、この"no"の値を見て、sample1～3 を呼び出すようにする。 sample1～3 毎に作成する「開始変数代入～終了変数代入-テスト用 CSV」には、入出力変数の他に関数を識別するための変数 "no"を入力変数として追加し、ここに sample1～3 を認識するための番号 "1"～"３"を設定する。

sample1 関数のとき

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | api | DBG_IN | DBG_OUT | test1 | 2 | 1 | sample1 |
| 2 | no | x | out | | | | |
| 3 | 1 | 1 | | | | | |
| 4 | | 2 | | | | | |
| 5 | | 3 | | | | | |

sample2 関数のとき

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | api | DBG_IN | DBG_OUT | test2 | 2 | 0 | sample2 |
| 2 | no | x | | | | | |
| 3 | 2 | 1 | | | | | |
| 4 | | 2 | | | | | |
| 5 | | 3 | | | | | |

sample3 関数のとき

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | api | DBG_IN | DBG_OUT | test3 | 3 | 1 | sample3 |
| 2 | no | x | y | out | | | |
| 3 | 3 | 1 | 3 | | | | |
| 4 | | 2 | 2 | | | | |
| 5 | | 3 | 1 | | | | |