「カバレッジマスターwinAMS と Sanitizer の連携」チュートリアル

目次

1. はじめに 2
1.1 本チュートリアルの対象ユーザー 2
2. Sanitizer の概要 3
2.1 Undefined Behavior Sanitizer(UBSan)で検出可能な不具合 3
2.1.1 UBSan で検出する不具合の例[配列の範囲外アクセス] 3
3. Clang Sanitizer について 5
3.1 Clang Sanitizer の仕組み 5
4. winAMS と Clang-UBSan 連携の構成 6
4.1 連携の構成について 6
4.2 提供する連携環境について 7
5. 実習環境の準備 8
5.1 ① 各種ファイルのインストール 8
5.1.1 マイコンシミュレータのインストール 9
5.1.2 シミュレーションカーネル （XAIL）のインストール 9
5.1.3 SSTManager（winAMS 本体）のインストール 10
5.2 ② ARM 向け Clang コンパイラのダウンロードと展開 10
5.3 ③ チュートリアル環境サンプルコードのダウンロードと展開 11
5.4 ④ 実習環境パスの設定 12
5.5 ⑤ 実習環境のフォルダ構成について 12
5.6 ⑥ マイコンシミュレータ バージョン確認について 13
6. チュートリアル 14
6.1 GUI 版単独 winAMS プロジェクトの実行手順 14
6.1.1 プログラムのビルド実行手順 14
6.1.2 winAMS シミュレーションの実行手順 16
6.1.3 Sanitizer 警告の確認手順 22
6.2 CLI を利用した複数 winAMS プロジェクトの連続実行手順 24
6.2.1 レポート機能：winAMS と Sanitizer 連携のユーティリティー 24
6.2.2 フォルダ構成 28
6.2.3 スクリプト動作の概要 29
6.2.4 スクリプトの実行手順 31
6.2.5 レポート内容 32
6.3 レポート内容の留意点 36

1. はじめに

車載システムを代表とするミッションクリティカルなソフトウェア開発で、GAIO の単体テストツール「カバレッジマスターwinAMS(以下、winAMS)」は、多くのお客様にご利用を頂いておりますが、昨今、従来以上にソフトウェア開発規模が拡大しており、更なる効果的な運用や、活用領域の拡大が求められています。

一方で、OSS(Open Source Software)の発展が著しく、開発やテストに有効なツールの改良やリリースが日々、行われております。

このような背景から、winAMS と OSS ツールである 「Clang Undefined Behavior Sanitizer(以下、Clang-UBSan)」の連携運用について、ご案内をさせて頂きます。

注意事項：
Clang- UBSan の品質保証を、GAIO は行っておりません。

1.1 本チュートリアルの対象ユーザー

- C プログラムのコーディングが可能な方
- winAMS の利用経験を持つ方

2. Sanitizer の概要

Sanitizer は、ソフトウェア実行時にメモリ使用やデーターアクセスなどの潜在的な問題を動的テストで検出するツールです。主にコンパイラの機能として提供されており、Sanitizer 機能が実装されている代表的なコンパイラとして、GCC、Clang、Microsoft コンパイラ(Microsoft C/C++統合開発環境)などがあります。

主に 4 種類の Sanitizer が存在します。

- AddressSanitizer(ASsan)：メモリの不正アクセスを検出
- UndefinedBehaviorSanitizer(UBSan)：未定義動作等の不具合を検出
- ThreadSanitizer(TSan)：データ競合を検出
- LeadSanitizer(LSan)：メモリリークを検出

2.1 Undefined Behavior Sanitizer(UBSan)で検出可能な不具合

UBSan は、以下のような未定義動作の不具合を検出します。

- 配列の範囲外アクセス
- データ型の範囲外のビットシフト
- 不整合ポインタまたは NULL ポインタの参照
- 符号付整数オーバーフロー
- 浮動小数点型への変換不整合

2.1.1 UBSan で検出する不具合の例[配列の範囲外アクセス]

具体的に「配列の範囲外アクセス」を例に説明します。
以下のソフトウェアには、「配列の範囲外アクセス」が含まれています。

```
1  #include <stdio.h>
2
3  int main(void){
4    int     buf[10];
5    int     i = 0;
6
7    while( i < 10 ) {
8      i++;          //i インクリメント
9      buf[i] = i;   //配列bufにiを代入処理
10   }
11
12   return 0;
13 }
14
```

図 1　配列の範囲外アクセスのソフトウェア

9 行目が「配列の範囲外アクセス」となりますが、目視で確実に検出することは、困難です。

2.1.1.1 処理の流れと解説

1. 配列 buf 添え字 10 を宣言
2. i を 0 で宣言
3. While 文で i が 10 未満の場合、
4. i をインクリメント
5. 配列 buf に i を代入処理

上記の i=9 の時、i をインクリメント(4 の処理)により i=10 となり、buf[10]にアクセス(5 の処理)した際に、「配列の範囲外へのアクセス」となってしまいます。

2.1.1.2 配列の範囲外アクセスの Clang-UBSan 実行ログと解説

実際に Clang-UBSan を使用し「配列の範囲外アクセス」を検出している実行ログが下左図です。

実行ログには UBSan 警告が表示され「配列の範囲外アクセス」を検出しています(下左図①青枠)。

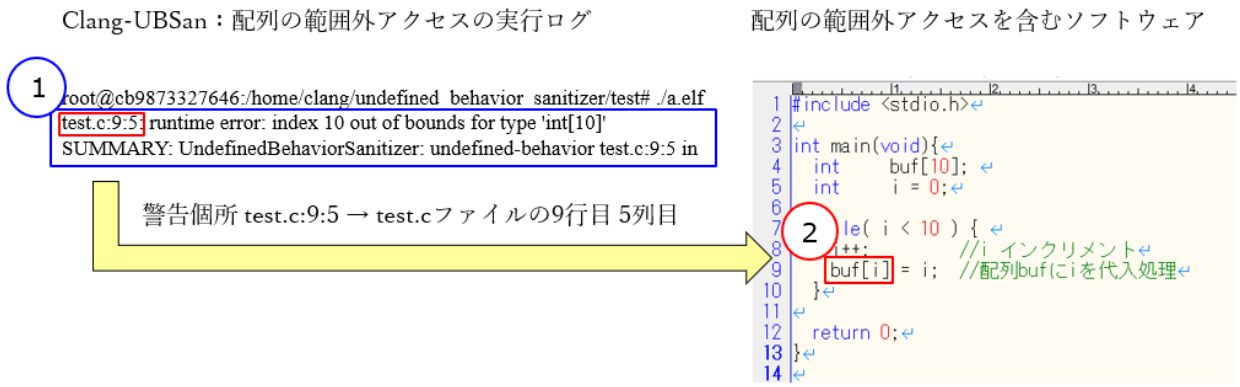


図 2 Clang-UBSan を使用した「配列の範囲外アクセス」の実行ログとコード

上図の①個所では、警告個所と警告内容が表示されます。警告個所「test.c:9:5」(上左図①赤枠)では、test.c ファイルの 9 行目 5 列目の buf[i](上右図②赤枠)を示しています。次に、警告内容「index 10 out of bounds for type 'int[10]'」は、「index 10 は int[10]の範囲外」と「buf[]配列の範囲外アクセス」を警告している事が分かります。

3. Clang Sanitizer について

Clang Sanitizer とは、Clang が提供する Sanitizer です。Clang(クラン)とは C/C++/Objective-C 言語向けのオープンソースコンパイラで、LLVM(※)と呼ばれるコンパイラバックエンドと連携し、様々なプラットフォーム向けに最適化されたバイナリ生成機能を提供しています。

(※)The LLVM Compiler Infrastructure Project

3.1 Clang Sanitizer の仕組み

次に Clang Sanitizer の不具合検出の仕組みを説明します。

Clang Sanitizer は Instrumentation 機能を用いて、不具合を検出します。Instrumentation とは、ソフトウェアやバイナリファイルに対して、パフォーマンス計測やエラー診断、トレースなどのためにコードを埋め込む技術のことを指します。

本技術を使用し、Clang コンパイラはコンパイルオプションで Sanitizer 機能を有効にすることで、不具合検出コード（Tag コード）を埋め込んだバイナリを生成します。埋め込まれた Tag コードが実行されると、ハンドラ関数がキャッチし、実行結果（不具合指摘）を出力する仕組みとなっています。

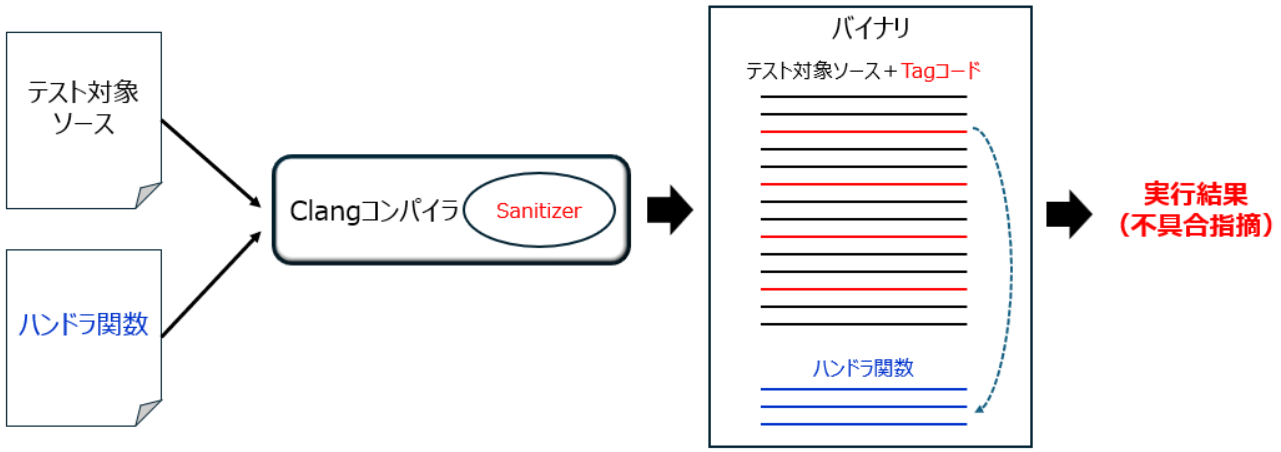


図 3 Clang Sanitizer の仕組み

4. winAMS と Clang-UBSan 連携の構成

4.1 連携の構成について

winAMS と Clang-UBSan 連携の構成について説明します。

最初に、テスト対象ソースとハンドラ関数ソースを Clang コンパイラでコンパイル（UBSan オプション付き）し、テスト実行バイナリファイルを生成します。次に、生成したテスト実行バイナリファイルを winAMS のマイコンシミュレータで実行します。実行後、UBSan の結果をシミュレーションログファイルへ出力します。

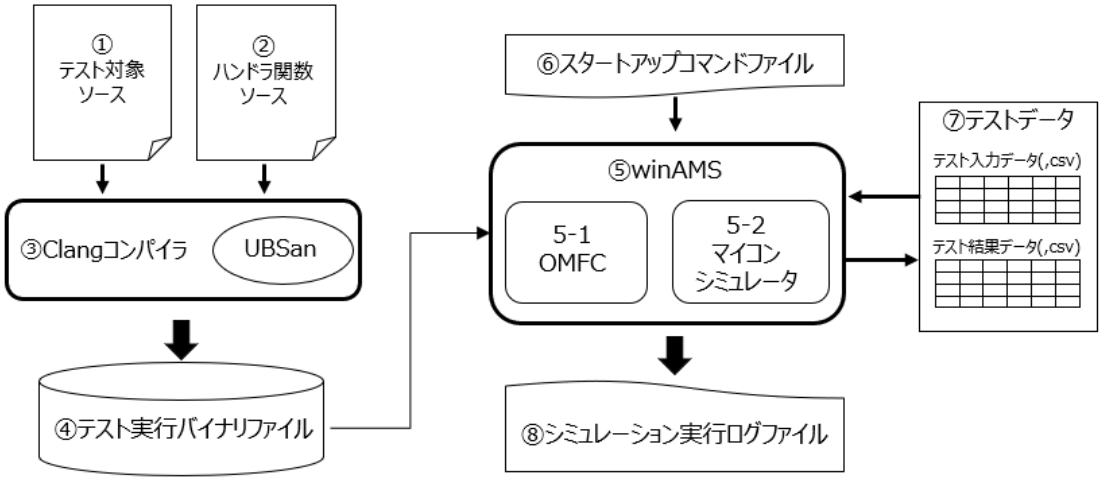


図 4 winAMS と Clang-UBSan 連携のシステム構成

① **テスト対象ソース：**

UBSan を用いて、検証するテスト対象ソース

② **ハンドラ関数ソース：**

UBSan で埋め込まれた Tag コードが実行された際に呼び出される関数（ハンドラ関数）を記載したソース。ハンドラ関数は、UBSan 実行結果（不具合指摘）を特定の変数に格納し、アウトプットするための実装が行われています。GAIO が作成した UBSan 用ハンドル関数をサンプル提供します。

③ **Clang コンパイラ：**

UBSan 機能を搭載したコンパイラ。コンパイラオプションを設定することで、Tag コード付きテスト実行バイナリファイルを生成します。

④ **テスト実行バイナリファイル：**

UBSan により、不具合検出 Tag コードが埋め込まれたテスト実行バイナリファイル。

⑤ **winAMS　5-1：OMFC**：

デバッグ情報フォーマット変換コンバータ。winAMS がコンパイラ対応する際に必要となるモジュール。コンパイラが出力するデバッグ情報をカバレッジマスター向けに変換します。

winAMS　5-2：マイコンシミュレータ：

テスト実行バイナリファイルを実行するための命令セットシミュレータ。

⑥ **スタートアップコマンドファイル：**

マイコンシミュレータのデバッグ機能や動作をスクリプトで設定する設定ファイル。ハンドラ関数に実装した UBSan 実行結果（不具合指摘）アウトプットをシミュレーション実行ログファイルに出力するためのスクリプトが定義されています。

⑦ **テストデータ：**

winAMS 用のテストデータ。UBSan を利用するための追加設定は不要です。

※過去作成したテストデータの再利用が可能

⑧ **シミュレーション実行ログファイル：**

マイコンシミュレータの実行ログを出力するファイル。スタートアップコマンドファイルで定義したスクリプトにより、UBSan 実行結果（不具合指摘）がシミュレーション実行ログファイルに出力されています。

4.2　提供する連携環境について

winAMS は Clang-UBSan と連携するために、以下の環境を提供しています。

- OMFC：ARM 用 OSS 版 Clang　V17.0.1　対応
- マイコンシミュレータ：ARM Cortex-R4/R4F/R5/R5F(Sanitizer Limited Edition)

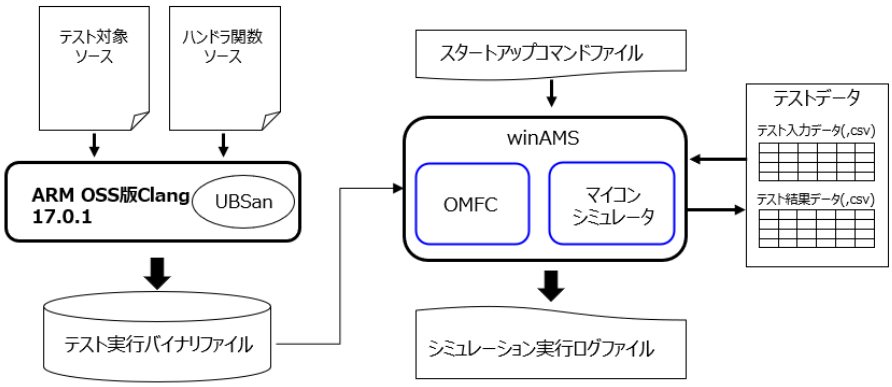


図 5 winAMS と Clang-UBSan 連携のシステム構成（詳細）

ARM 用 OSS 版 Clang でコンパイル可能なソースコードであれば、実際のターゲットコンパイラやターゲットマイコンとは異なる環境のソースコードでも、winAMS と連携して、Clang-UBSan をご利用頂けます。

※　マイコン非依存版カバレッジマスターGeneral と同一の位置付けで、ソースコードレベル（ソース論理レベル）のテストや不具合検出が可能です。

※　「ARM Cortex-R4/R4F/R5/R5F」をすでにインストールされた PC で「ARM Cortex-R4/R4F/R5/R5F(Sanitizer Limited Edition)」をインストールすると、マイコンシミュレータが上書きされますが、そのままご利用頂いて、問題ありません。

5. 実習環境の準備

まず、最初に環境構築には 4 つの①～⑥の手順が必要です。

①各種ファイルのダウンロードとインストール

1. マイコンシミュレータ
2. シミュレーションカーネル
3. SSTManager（winAMS 本体）

②ARM 向け Clang コンパイラのインストール

③チュートリアル環境サンプルコードのダウンロード

④実習環境パスの設定

⑤実習環境のフォルダ構成について

⑥マイコンシミュレータ、OMFC のバージョン確認について

各種ファイルのダウンロードとインストール時のライセンスについて

既に winAMS のライセンスをお持ちの方は、マイコンシミュレータ Cortex R4/R5 を利用するために、ライセンスを変更ください。(マイコン変更サービスをご活用ください)

5.1 ① 各種ファイルのインストール

以下のリンクより、各種インストールを行います。

▶カバレッジマスターを使ってみよう（gaio.co.jp）

<インストールファイル>

1. マイコンシミュレータ
2. シミュレーションカーネル
3. SSTManager（winAMS 本体）

既に winAMS をご利用されている場合は、「2.シミュレーションカーネル」、「3. SSTManager（winAMS 本体)」のインストールは不要になります。

5.1.1 マイコンシミュレータのインストール

以下のリンクよりマイコンシミュレータをダウンロードとインストールをお願いします。

▶GAIO カバレッジマスターwinAMS/ゼネラル　ダウンロード

マイコンシミュレータ、OMFC のバージョン確認は別途「5.6 ⑥ マイコンシミュレータ バージョン確認について 5.6 」で行います。順に手順を進めてください。

【 シミュレータエンジン： SX 】Cortex-R4/R4F/R5/R5F(Sanitizer Limited Edition)

※現在は Cortex R4/R5 の対応

【 シミュレータエンジン： SX 】

ARM	Cortex-M7	Cortex-A53/A57	Cortex-R7	Cortex-R52
	Cortex-A55/A75/A76	Cortex-R4/R4F/R5/R5F	Cortex-M33	Cortex-M23
	Cortex-M55	Cortex-A65/A78/A78AE	Cortex-A35/A72	Cortex-R4/R4F/R5/R5F(Sanitizer Limited Edition)
Infineon	TriCore			
Renesas	RXv2	RXv3	RH850(G4)	RH850
RISC-V	RISC-V(RV32)	RISC-V(RV64)		
Xilinx	MicroBlaze			

図 6 マイコンシミュレータのダウンロードページ

5.1.2 シミュレーションカーネル （XAIL）のインストール

以下のリンクよりダウンロードとインストールをお願いします。既に winAMS ご利用の場合は、インストールは不要になります。

▶GAIO カバレッジマスターwinAMS/ゼネラル　ダウンロード

2．シミュレーションカーネル （XAIL） インストール

次に、シミュレーションカーネル（XAIL）をインストールします。下のリンクから直接インストーラを実行して下さい。
途中で、マイクロソフト社のモジュール「.NET Framework」、「Visual C++ 再頒布可能パッケージ」のインストーラーが起動する場合があります。
これらは、XAILの実行に必要なモジュールです。インストーラのメッセージに従って、インストールを行って下さい。
【参考】 シミュレーションカーネル（XAIL）インストール手順

【重要】「InstallShieldウィザードを完了しました」のダイアログが表示されるまで、次のインストールには進まないで下さい。

- シミュレーションカーネルXAIL最新版アップデートモジュール

図 7　シミュレーションカーネルのダウンロードページ

5.1.3 SSTManager（winAMS 本体）のインストール

以下のリンクよりダウンロードとインストールをお願いします。バージョンは R9.1 以降に対応しています。バージョン R9.1 未満をご利用の場合は、最新化をお願いします。

▶GAIO カバレッジマスターwinAMS/ゼネラル　ダウンロード

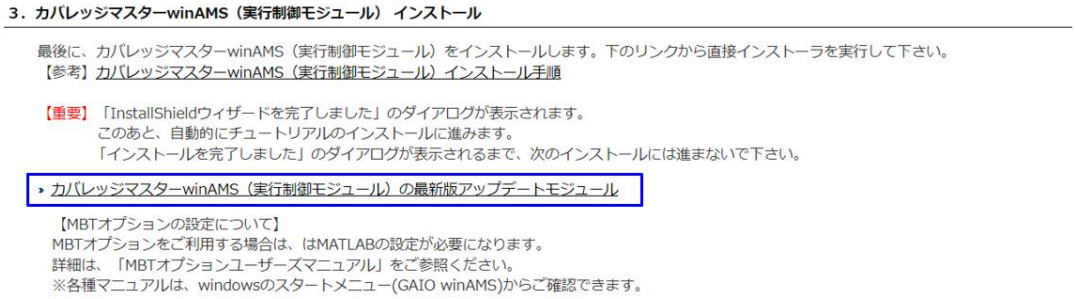


図 8　SSTManager ダウンロードページ

5.2　② ARM 向け Clang コンパイラのダウンロードと展開

ARM Software から ARM 向け Clang コンパイラがリリースされています。前節の「4.2 提供する連携環境について」にあるように OMFC は、LLVM-embedded-toolchain-for-Arm V17.0.1 のみ対応しています。

こちらよりダウンロードをお願いします。▶ LLVM-embedded-toolchain-for-Arm release-17.0.1

- LLVMEmbeddedToolchainForArm-17.0.1-Windows-x86_64.zip

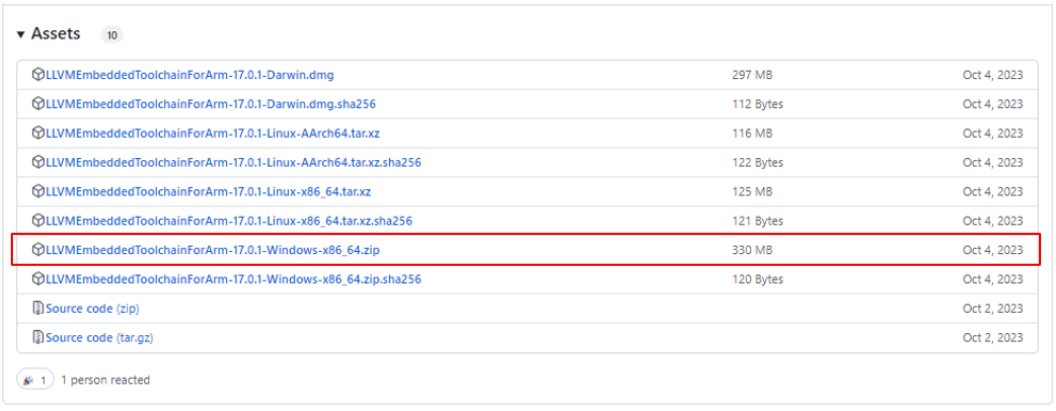


図 9 ARM 向け Clang コンパイラのダウンロードページ

C ドライブ直下に「work」フォルダを作成頂き、「work」フォルダ直下へ展開ください。展開後のフォルダ容量は約 1G Byte になりますので、展開前に C ドライブ空き容量を確保頂き、展開をお願いします。

5.3 ③ チュートリアル環境サンプルコードのダウンロードと展開

続いて、チュートリアル環境サンプルコードになります。チュートリアル環境サンプルコードには、チュートリアルで使用するテスト対象ソース、ハンドラ関数ソース、サンプル winAMS プロジェクトとレポート生成を行うスクリプトを同封しています。

以下の順に実施をお願いします。

1. 本書をダウンロード頂いた「カバレッチマスターwinAMS と Sanitizer の連携」チュートリアル内の「チュートリアル環境サンプルコード」をダウンロードしてください。

2. ダウンロード頂いた「チュートリアル環境サンプルコード.zip」をテスクトップ等の任意のフォルダへ展開してください。展開フォルダの中に「src」フォルダがあります。

3. 前章の ARM 向け Clang コンパイラ「LLVMEmbeddedToolchainForArm-17.0.1」内の「src」へコピーしていきます。「C¥work¥LLVMEmbeddedToolchainForArm-17.0.1-Windows-x86_64¥samples」直下の「src」フォルダへチュートリアル環境サンプルコード内の「src」を上書きコピーしてください。

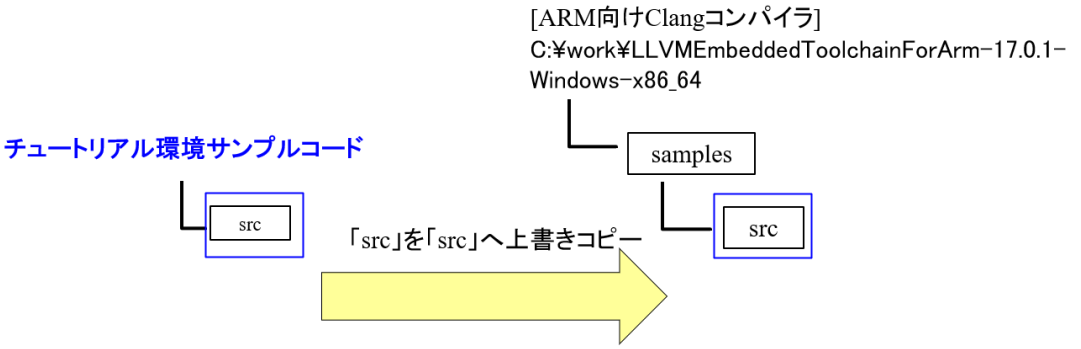


図 10 チュートリアル環境サンプルコードと ARM 向け Clang コンパイラのフォルダ上書きコピー位置の関係

上書きコピーすると、以下のフォルダ構成となっている事を確認してください。

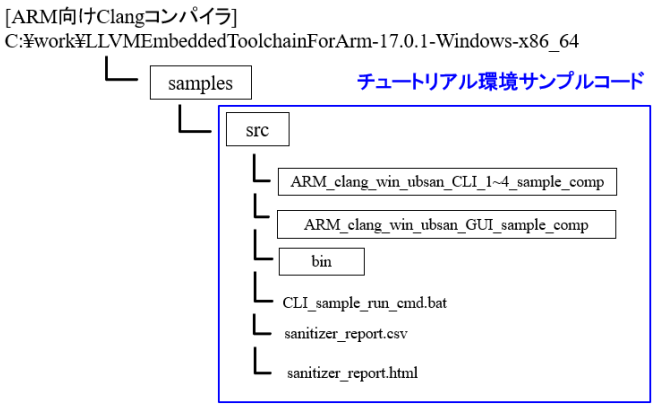


図 11 チュートリアル環境サンプルコード内のフォルダ構成

5.4 ④ 実習環境パスの設定

チュートリアル環境サンプルコード内の Make file に Clang コンパイラの格納場所が必要なため、Windows の「ユーザー環境変数」設定から新規に変数の追加設定を行います。

<追加設定>

- 変数：BIN_PATH
- 値：C:¥work¥LLVMEmbeddedToolchainForArm-17.0.1-Windows-x86_64¥bin

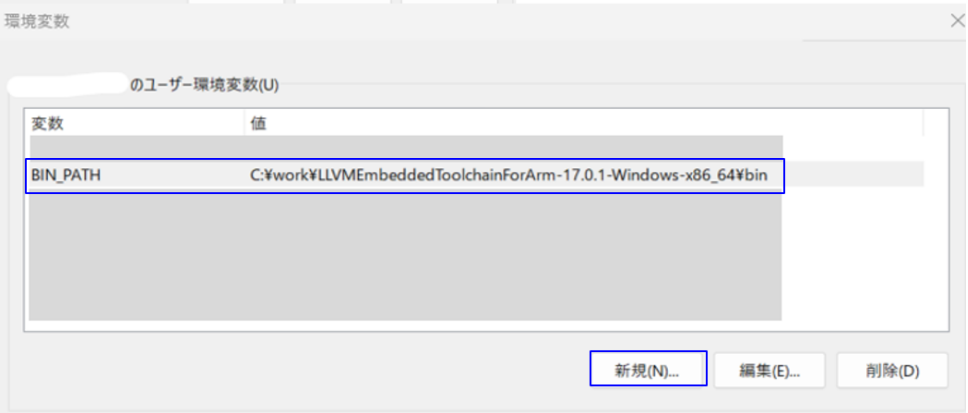


図 12 ユーザー環境変数の Windows 設定画面

5.5 ⑤ 実習環境のフォルダ構成について

本実習環境のフォルダ構成は以下の通りです。ARM_clang_win_ubsan_CLI1～4_sample_comp は CIL を使い複数 winAMS プロジェクトを連続実行させる際に使用します。GUI 版単独 winAMS プロジェクトは単独で実行させる際に使用します。レポート生成スクリプトは、複数 winAMS プロジェクトを実行した際に、複数の Sanitizer 警告文を集約しレポートする際に使用します。

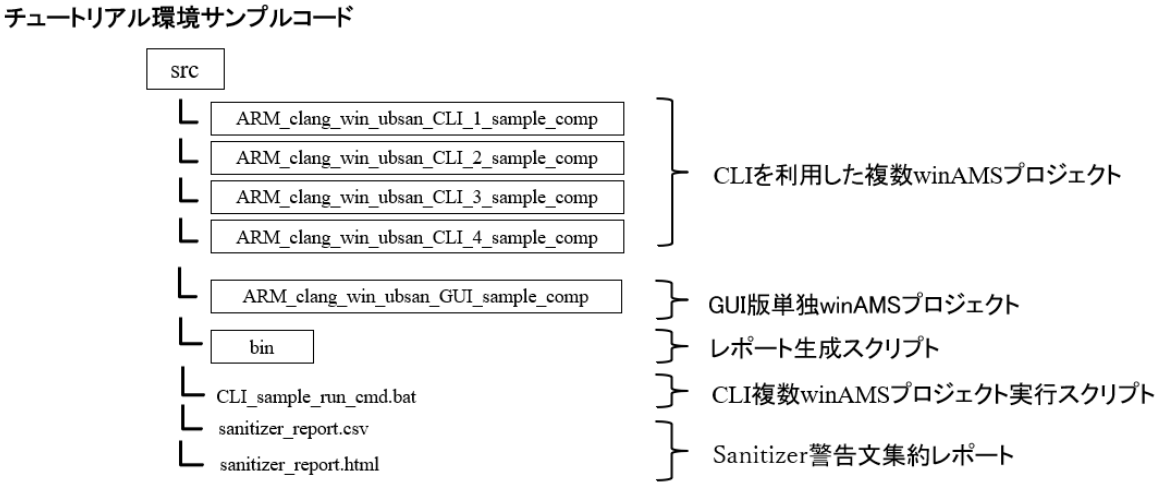


図 13 実習環境のファイル構成

5.6 ⑥ マイコンシミュレータ バージョン確認について

マイコンシミュレータのインストール後のバージョン確認を行います。
以下の winAMS プロジェクトファイルをダブルクリックし、SSTManager を開きます。

- C:¥work¥LLVMEmbeddedToolchainForArm-17.0.1-Windows-x86_64¥samples¥src¥ARM_clang_win_ubsan_GUI_sample_comp¥winAMS_env_comp¥winAMS_env_comp.amsy

メニュー内の「ヘルプ(H)」をクリックし、「ツールバージョン一括表示(T)」を押下します。

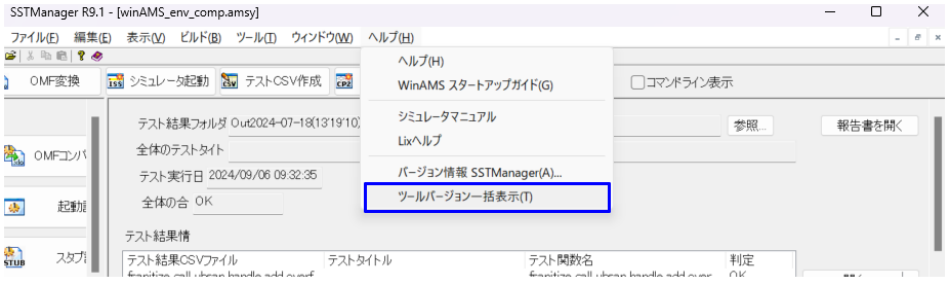


図 14 「ツールバージョン一括表示」ボタン位置

メッセージウィンドウにツールバージョンが表示されます。

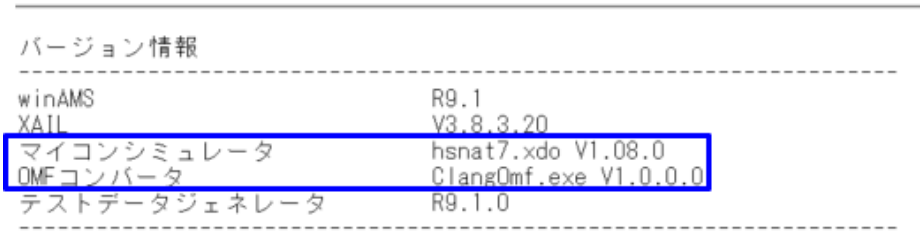


図 15 ツールバージョン一括表示の結果

上記の通り、「マイコンシミュレータ　hsnat7.xdo V1.08.0」、「OMF コンバータ ClangOmf.exe V1.0.0.0」と表示されている事を確認してください。

6. チュートリアル

チュートリアルでは、winAMS と Clang-UBSan を連携し実行する例とし、以下の 2 種類をご紹介します。

1. GUI 版単独 winAMS プロジェクトの実行
2. CLI を利用した複数 winAMS プロジェクトの連続実行

本チュートリアルでは、以下の通り、連携における必要な設定や内容を理解・習得いただきます。

- UBSan 機能有効とした Clang Sanitizer を使ったビルド方法
- winAMS 設定とシミュレーション実行方法
- Sanitizer 警告検出結果[UBSan 実行結果(不具合指摘)]の確認方法
- レポート機能
- スクリプト動作の概要と手順

6.1 GUI 版単独 winAMS プロジェクトの実行手順

まずは、GUI 版単独 winAMS プロジェクトの実行手順から説明します。

6.1.1 プログラムのビルド実行手順

コマンドプロンプトを起動し、ローカル環境へ展開頂いたチュートリアル環境サンプルコード内の「ARM_clang_win_ubsan_GUI_sample_comp」へ移動します。

```
>cd  C:¥work¥LLVMEmbeddedToolchainForArm-17.0.1-Windows-x86_64¥samples¥src¥ARM_clang_win_ubsan_GUI_sample_comp
```

続いて、プログラムをビルドします。

```
>make.bat build
```

サンプル環境では既にビルドされたファイルを同封していますが、同フォルダのバイナリファイル(a.elf)の日時が最新化されれば、ビルドは成功です。

6.1.1.1 ビルドオプション

make.bat では UBSan を有効に設定し(太文字部分)ビルドを実施しており変更する必要はありません。

ビルドプションについては、以下の通りです。

表 6-1 ビルドオプション一覧

オプション項目	内容
--target=arm-none-eabi	ターゲットアーキテクチャの指定
-mcpu=cortex-R4	ターゲットプロセッサの指定
-mfloat-abi=soft	浮動小数点演算の ABI 指定(soft/hard)
-O0	最適化レベル：無効
-lsemihost	セミホスト機能を提供するライブラリをリンク
-gdwarf-4	デバッグ情報のフォーマット
-fsanitize=undefined,unsigned-integer-overflow	**Sanitizer UBSan 機能：有効**
-g	デバッグ情報を生成機能：有効
-T	リンカスクリプトファイルの指定

6.1.2 winAMS シミュレーションの実行手順

次に、winAMS シミュレーションの実行手順を順に説明します。

6.1.2.1 プロジェクトファイルを開く

以下の winAMS プロジェクトファイルを開くと、SSTManager の画面が起動します。

- C:¥work¥LLVMEmbeddedToolchainForArm-17.0.1-Windows-x86_64¥samples¥src¥ARM_clang_win_ubsan_GUI_sample_comp¥winAMS_env_comp¥winAMS_env_comp.amsy

チュートリアル環境サンプルコード内の winAMS プロジェクトは、既に各種設定済みのファイルになります。設定済みプロジェクトファイルを元に説明します。

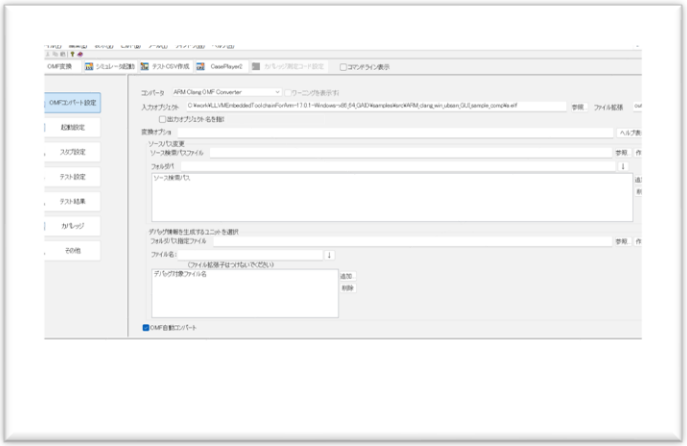


図 16 SSTManager 起動画面

6.1.2.2 「OMF 変換」

入力オブジェクトには「プログラムのビルド実行手順」でビルドしたバイナリファイル「a.elf」が設定されています(赤枠参照)。次に、OMF 変換ボタン(青枠参照)を押下します。

図 17 winAMS プロジェクトウィンドウ内の「OMF 変換」

画面下側のログエリアに「変換が完了しました」と表示され、「a.elf」と同フォルダの「a.elf.xlo」が更新されれば、OMF 変換は完了です。

図 18 winAMS メッセージウィンドウ

6.1.2.3 「起動設定」

続いて、左メニューの「起動設定」です。

6.1.2.3.1 シミュレーション開始、終了

本サンプルプロジェクト環境では、シミュレーション開始、終了を自動的に開始、終了に設定しています。手動でシミュレーション開始と終了を試行されたい場合は、以下の設定のチェックを外してシミュレーション実行してください。

図 19 「起動設定」内の「シミュレーション開始、終了部分」

6.1.2.3.2 スタートアップコマンド内の Sanitizer 警告出力マクロについて

Sanitizer 警告出力マクロを使用する事で、既存のテスト資産を有効活用することができます。既存のテスト資産を使用する際は本マクロの追加が必要になりますので、マクロの動作概要を詳しく説明します。

スタートアップコマンドファイルを開き確認していきます。以下の通り「起動設定」からスタートアップコマンドファイルの編集ボタンを押下すると、スタートアップコマンドファイル(SS__STARTUP.txt)が開きます。

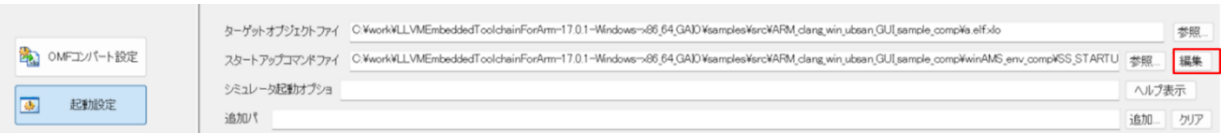


図 20 winAMS プロジェクトウィンドウ内のスタートアップコマンドファイル編集ボタン

スタートアップコマンドファイル(SS__STARTUP.txt)を開くと、Sanitizer 警告出力マクロが記載されています。

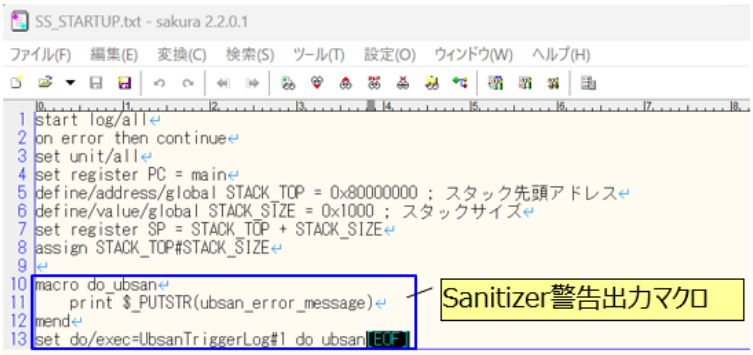


図 21 スタートアップコマンドファイルの内容

Sanitizer 警告出力マクロ内容の詳細を説明します。

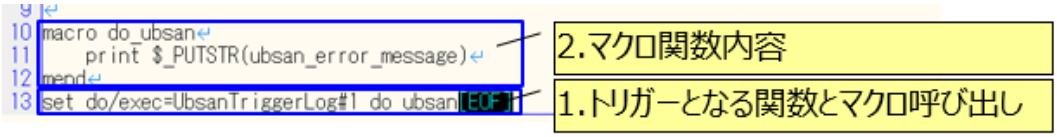


図 22 Sanitizer 警告マクロ詳細

1. **トリガーとなる関数とマクロ呼び出し**

青枠内の「UbsanTriggerLog#」がトリガーとなる関数です。トリガーとなる関数が実行されると、do_ubsan マクロ関数を実行します。

2. **マクロ関数内容**

do_ubsan マクロ関数では、変数(ubsan_error_message)をシミュレーションログ(sx.log)へ出力します。

Sanitizer 警告出力マクロの動作概要

動作概要を説明します。

下図は、左にハンドラ関数、右にスタートアップコマンドファイルを示しています。

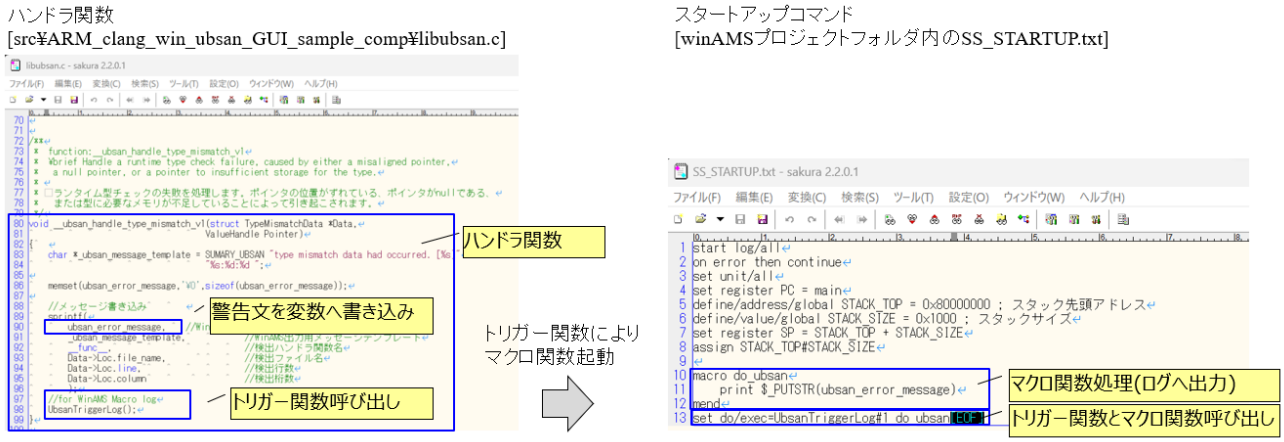


図 23 Sanitizer 警告出力マクロとスタートアップコマンドの関係

上左図のハンドラ関数が記載されています。ハンドラ関数では発生した状態を変数(ubsan_error_message)へ書き込み、トリガー関数を呼び出し終了します。上右図のスタートアップコマンドではトリガー関数が呼ばれると、マクロ関数が呼ばれ変数(ubsan_error_message)をシミュレーションログ(sx.log)へ出力します。

6.1.2.4 「テスト設定」

左メニューの「テスト設定」を開き、テストする関数を確認します。

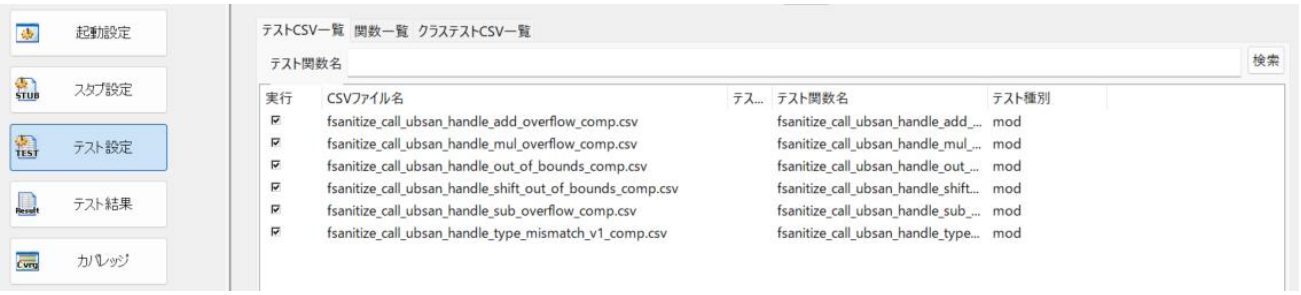


図 24 winAMS プロジェクトウィンドウ内のテスト設定

実行するテスト CSV とプログラム概要は以下の通りです。

※テスト関数のファイルパス：ARM_clang_win_ubsan_GUI_sample_comp¥testcode.c

表 6-2 各テスト CSV とプログラム概要の一覧

CSV 名	プログラム概要[関数名]
fsanitize_call_ubsan_handle_add_overflow_comp.csv	整数加算オーバーフロー [fsanitize_call_ubsan_handle_add_overflow]
fsanitize_call_ubsan_handle_mul_overflow_comp.csv	整数乗算オーバーフロー [fsanitize_call_ubsan_handle_sub_overflow]
fsanitize_call_ubsan_handle_out_of_bounds_comp.csv	配列の範囲外アクセス [fsanitize_call_ubsan_handle_out_of_bounds] ※「配列の範囲外アクセスの例」章の内容
fsanitize_call_ubsan_handle_shift_out_of_bounds_comp.csv	データ型の範囲外のビットシフト [fsanitize_call_ubsan_handle_shift_out_of_bounds]
fsanitize_call_ubsan_handle_sub_overflow_comp.csv	整数減算オーバーフロー [fsanitize_call_ubsan_handle_mul_overflow]
fsanitize_call_ubsan_handle_type_mismatch_v1_comp.csv	NULL ポインタの参照 [fsanitize_call_ubsan_handle_type_mismatch_v1]

チュートリアル環境サンプルコードの winAMS プロジェクトには、以下の「モジュールテスト用 CSV 雛型作成」画面の通り、テスト実行するための入出力変数及びテストデータは設定済みですが、既存テスト CSV ファイルを使用する場合も、同様に「テスト設定」で Sanitaizer 警告出力の為の追加、修正する必要はなく、既存テスト資産をそのまま流用する事が可能です。※下図赤枠は関数呼び出しの為の INPUT です。

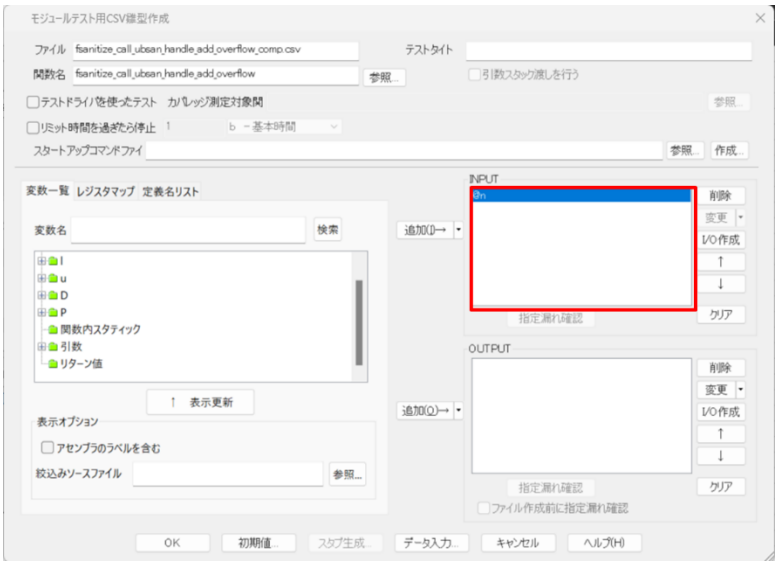


図 25 モジュールテスト用 CSV 雛型作成

6.1.2.4.1 テスト結果の保存先

テスト結果の保存先は、以下赤枠部分のパスに保存されます。

図 26 winAMS プロジェクトウィンドウ内のテスト結果保存先

6.1.2.5 「シミュレーション実行」

メニュー内の「シミュレータ起動」ボタンを押下します。(赤枠参照)

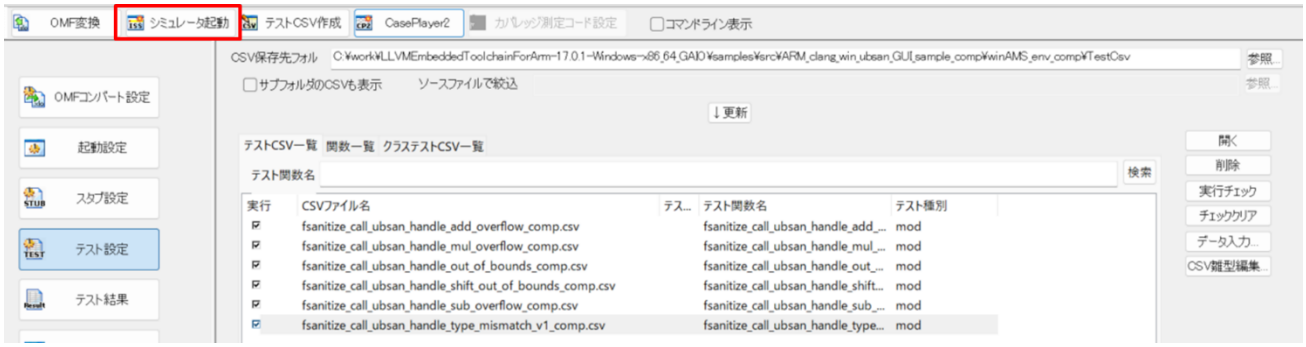


図 27 winAMS プロジェクトウィンドウ内のシミュレーション起動ボタン

シミュレーションが終了すると、winAMS メッセージウィンドウに以下のログが表示されます。

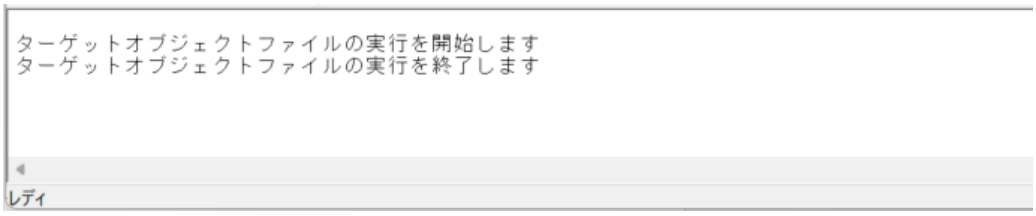


図 28 winAMS メッセージウィンドウ

6.1.3 Sanitizer 警告の確認手順

次に Sanitizer 警告の確認手順を説明します。

6.1.3.1 Sanitizer 警告の確認シミュレーションログの確認方法

「起動設定」章の「スタートアップコマンド内の Sanitizer 警告出力マクロ」により、Sanitizer 警告を検出した場合、シミュレーションログ(sx.log)へ出力されます。

以下のファイルパスに存在するシミュレーションログ(sx.log)を確認します。

- C:¥work¥LLVMEmbeddedToolchainForArm-17.0.1-Windows-x86_64¥samples¥src¥ARM_clang_win_ubsan_GUI_sample_comp¥winAMS_env_comp¥sx.log

シミュレーションログの内容を説明していきます。

シミュレーションログには、「%SX-I-XIPC～returned from」ブロックが 6 つ並んでいます。これは、6 つのテスト CSV(「テスト設定」参照)に基づいて、シミュレーション実行されたログです。(青枠参照)。

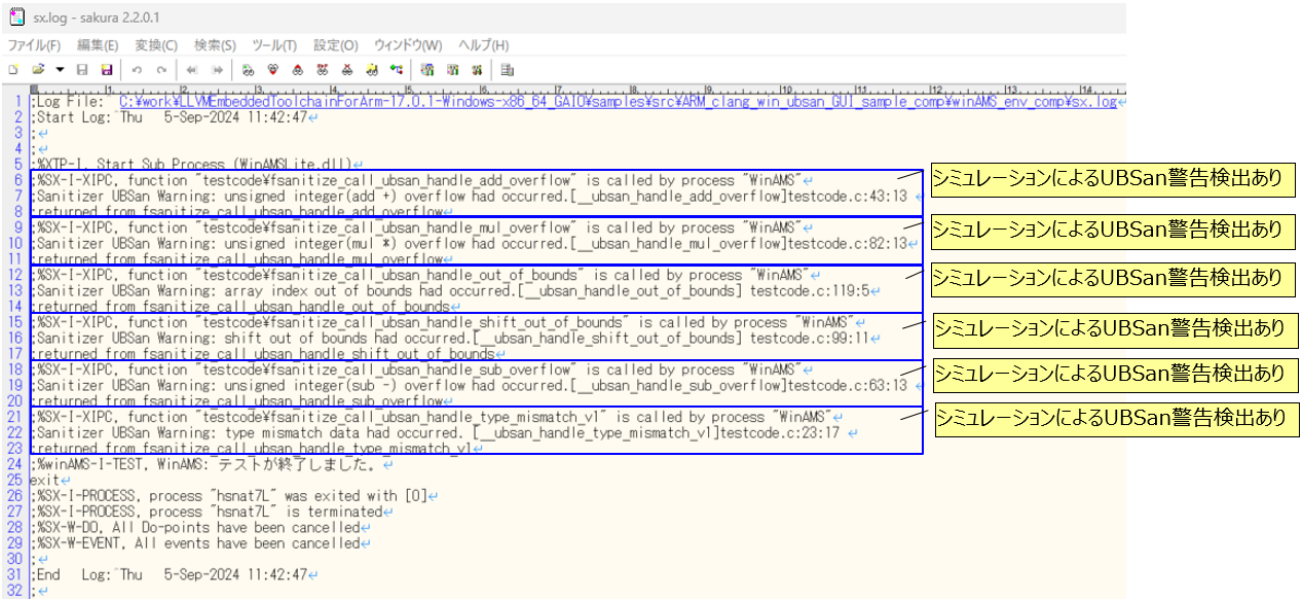


図 29 シミュレーションログ全体内容

ブロック内の詳細を説明します。

テスト関数は「function "testcode¥fsanitize_call_ubsan_handle_add_overflow"」(青枠参照)になります。テスト関数をシミュレーション実行した際に、未定義動作を検出した場合、「Sanitizer UBSan Warning:～」と出力されます(赤枠部分)。

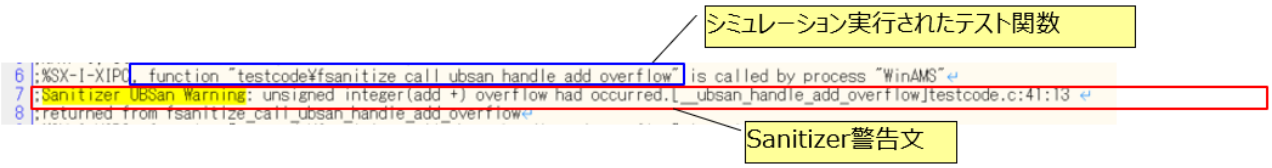


図 30 シミュレーションログ内のテスト関数と Sanitizer 警告文

Sanitizer 警告は、警告キーワード、警告内容、ハンドル関数名、未定義検出の該当箇所で構成されています。

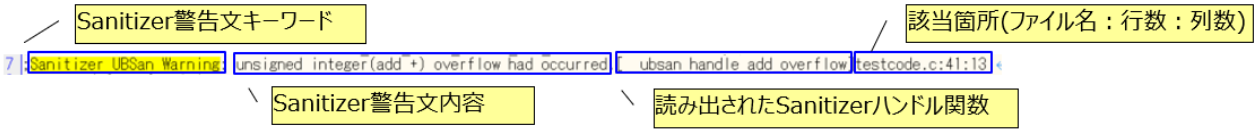


図 31 シミュレーションログ内の Sanitizer 警告文の構成

以上で GUI 版チュートリアルは終了になります。

6.2 CLI を利用した複数 winAMS プロジェクトの連続実行手順

続いて、CLI 版でのチュートリアルです。チュートリアル環境サンプルコード内の CLI 版は 4 つの winAMS プロジェクトを使用します。それぞれの winAMS プロジェクトは以下のテスト CSV を設定しています。テスト CSV の詳細は GUI 版の「テスト設定」章の「表 2 各テスト CSV とプログラム概要の一覧」をご参照ください。また、本 CLI コマンドでは追加の「オプションライセンス」は不要です。

表 6-3 各フォルダ名とテスト CSV 一覧

フォルダ名	テスト CSV
ARM_clang_win_ubsan_CLI_1_sample_comp	fsanitize_call_ubsan_handle_add_overflow_comp.csv
ARM_clang_win_ubsan_CLI_2_sample_comp	fsanitize_call_ubsan_handle_mul_overflow_comp.csv
ARM_clang_win_ubsan_CLI_3_sample_comp	fsanitize_call_ubsan_handle_sub_overflow_comp.csv
ARM_clang_win_ubsan_CLI_4_sample_comp	fsanitize_call_ubsan_handle_type_mismatch_v1_comp.csv

6.2.1 レポート機能：winAMS と Sanitizer 連携のユーティリティー

本章の「複数 winAMS プロジェクトの連続実行」のチュートリアルの前に説明が必要な事項として、Sanitizer 警告のレポート機能について説明します。

前章の「GUI 版単独 winAMS プロジェクトの実行手順」では Sanitizer 警告を確認する際に、winAMS プロジェクト内のシミュレーションログを確認する方法をご紹介しました。しかし、シミュレーションログはシミュレーション実行毎に上書きする仕様となっています。今回のような「複数の winAMS プロジェクトの連続実行」の場合、シミュレーションログが上書きされてしまうため、Sanitizer 警告を確認する事ができません。

そこで、レポート機能ではシミュレーション実行毎にシミュレーションログを退避させ、退避させた複数のシミュレーションログから Sanitizer 警告をまとめて確認できるようレポート出力した機能になります。

6.2.1.1 レポート機能システム構成

レポート機能は、シミュレーションログを退避する機能(ログ退避実行ファイル)と、退避したシミュレーションログ元に Sanitizer 警告を集約し HTML 版、CSV 版のレポート生成する 2 つの機能(レポート生成スクリプト)で構成されています。

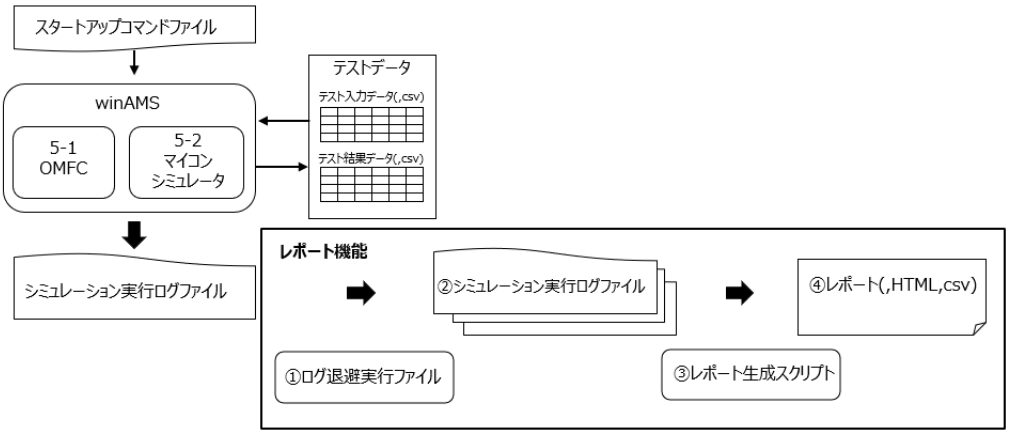


図 32 レポート機能システム構成

順に説明します。

① 「ログ退避実行ファイル」は、シミュレーションログを退避する実行ファイルです。

② 「シミュレーション実行ログファイル」は、①によって退避された複数のシミュレーションログファイルです。

③ 「レポート生成スクリプト」は、退避された複数のシミュレーションログから Sanitizer 警告文を集約しレポートするスクリプト群です。

④ 「レポート(HTML,csv)」は、HTML 形式、CSV 形式の集約結果レポートファイルです。

「ログ退避実行ファイル」、「レポート生成スクリプト」について、詳細に説明していきます。

6.2.1.2 ログ退避実行ファイル

ログ退避実行ファイルは、winAMS プロジェクトフォルダ内のシミュレーションログログ(sx.log)をテスト結果保存フォルダへリネームし退避します。

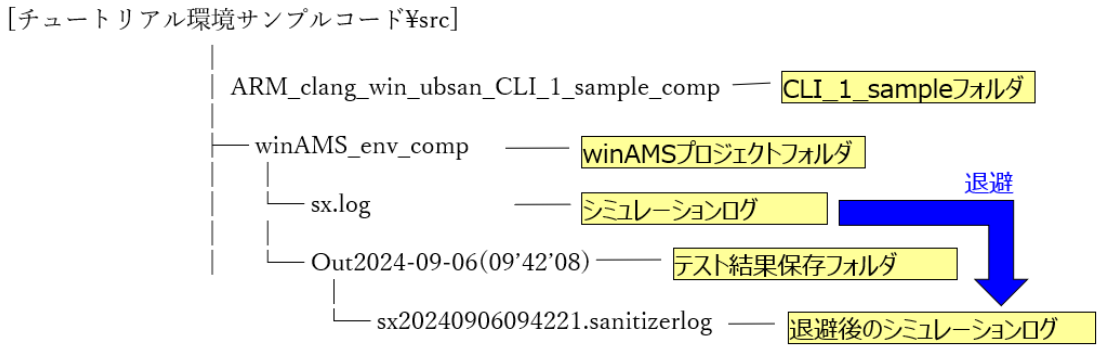


図 33 ログ退避実行ファイル動作概要

■ コマンド書式：

SimulaterlogSanitizerCheck.exe “winAMS プロジェクトファイルパス”

■ 使用例：

```
>bin¥SimulaterlogSanitizerCheck.exe "C:¥work¥LLVMEmbeddedToolchainForArm-17.0.1-Windows-x86_64¥samples¥src¥ ARM_clang_win_ubsan_CLI_1_sample_comp¥winAMS_env_comp¥winAMS_env_comp.amsy"
```

■ アウトプット：

winAMS プロジェクトで設定されているテスト結果の保存先であるフォルダ配下へシミュレーションログファイルをリネームし、拡張子.sanitizerlog ファイルを出力します。

ARM_clang_win_ubsan_CLI_1_sample_comp ¥winAMS_env_comp¥Out2024-09-06(09'42'08) sx20240906094221.sanitizerlog ※ファイ名は、実行日時となります。

6.2.1.3 レポート生成スクリプト

レポート生成スクリプトは、指定されたフォルダパス配下にある winAMS プロジェクトを検索し、退避されたシミュレーションログ(*.sanitizerlog)を検索します。退避されたシミュレーションログが存在した場合は、ログ解析・Sanitizer 警告を集約し、レポート出力するスクリプトです。

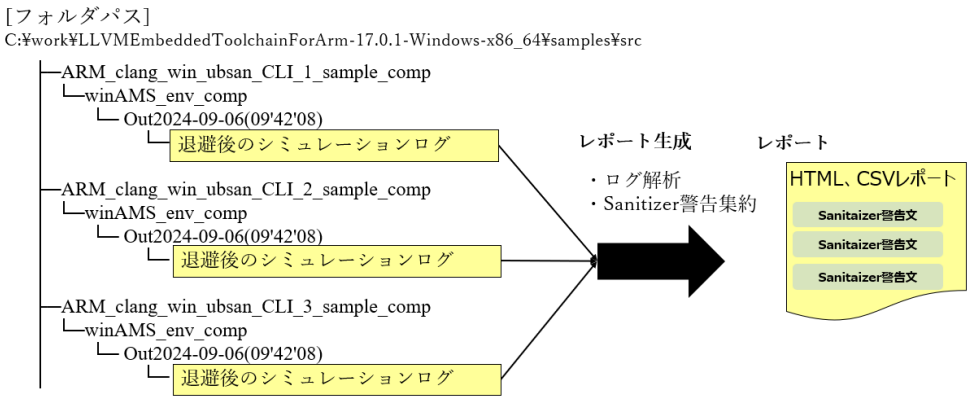


図 34 レポート生成スクリプト動作概要

■ コマンド書式：

main__sanitizer_summary.bat ［フォルダパス］

フォルダパス配下に winAMS プロジェクトフォルダが含まれているフォルダパスを指定してください。

■ 使用例：

```
>bin¥main__sanitizer_summary.bat "C:¥work¥LLVMEmbeddedToolchainForArm-17.0.1-Windows-x86_64¥samples¥src
```

■ アウトプット：

カレントディレクトリに HTML 形式と CSV 形式でレポートファイル(sanitizer_report)が生成されます。

以上が、レポート機能の説明になります。

6.2.2 フォルダ構成

複数 winAMS プロジェクトの連続実行で使用するフォルダ構成について「ARM_clang_win_ubsan_CLI_1_sample_comp」を例に説明します。その CLI_2~4_sample_comp の winAMS プロジェクトのフォルダ構成は同様です。

フォルダ全体の構成は下図の通りとなっています。また、下図のシミュレーション実行するためのスクリプトは太文字で示しており、次章で説明していきます。

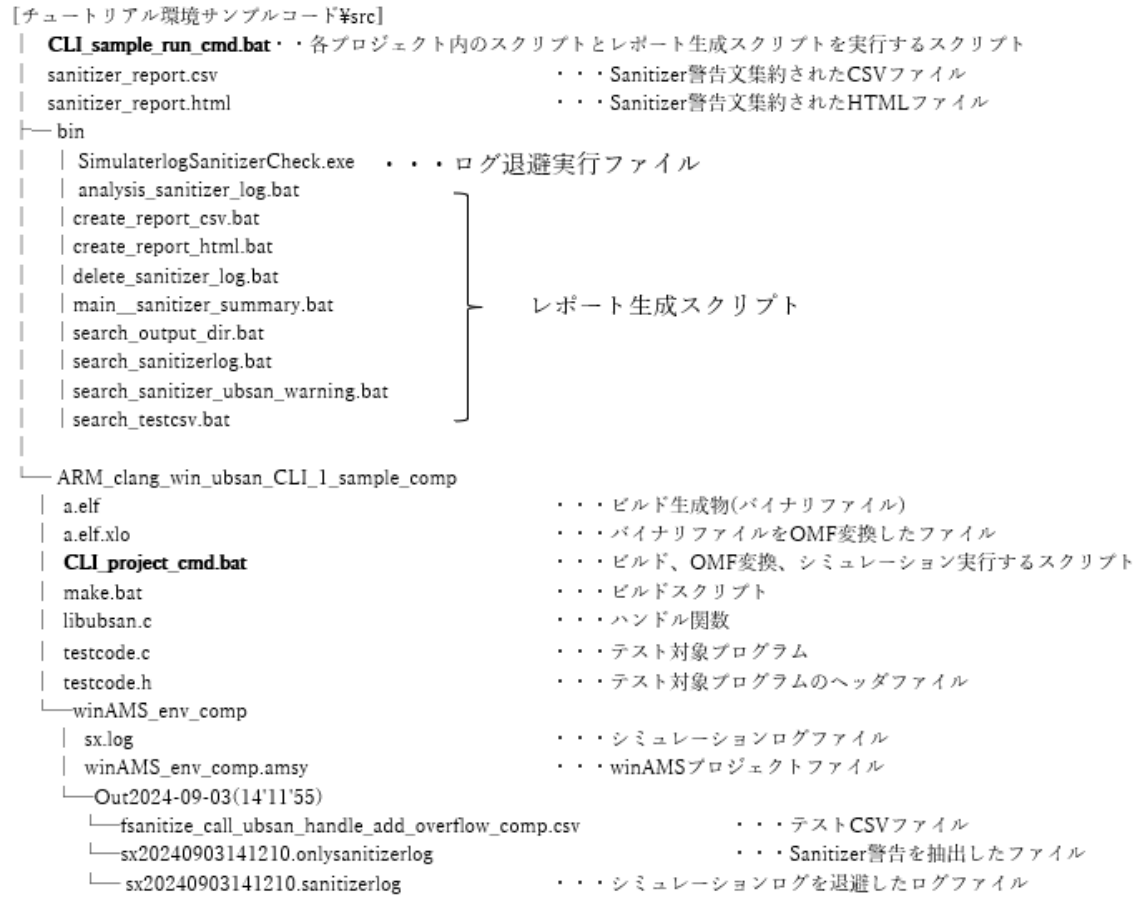


図 35 フォルダ構成

※winAMS と UBSan 連携に関連しないファイルの説明は省略しています。

6.2.3 スクリプト動作の概要

次に、シミュレーション実行に関するスクリプト内容について説明します。

6.2.3.1 CLI_sample_run_cmd.bat

本スクリプトは、各 winAMS プロジェクトスクリプト(次章 CLI_project_cmd.bat)を実行し、最後にレポート機能を実行します。

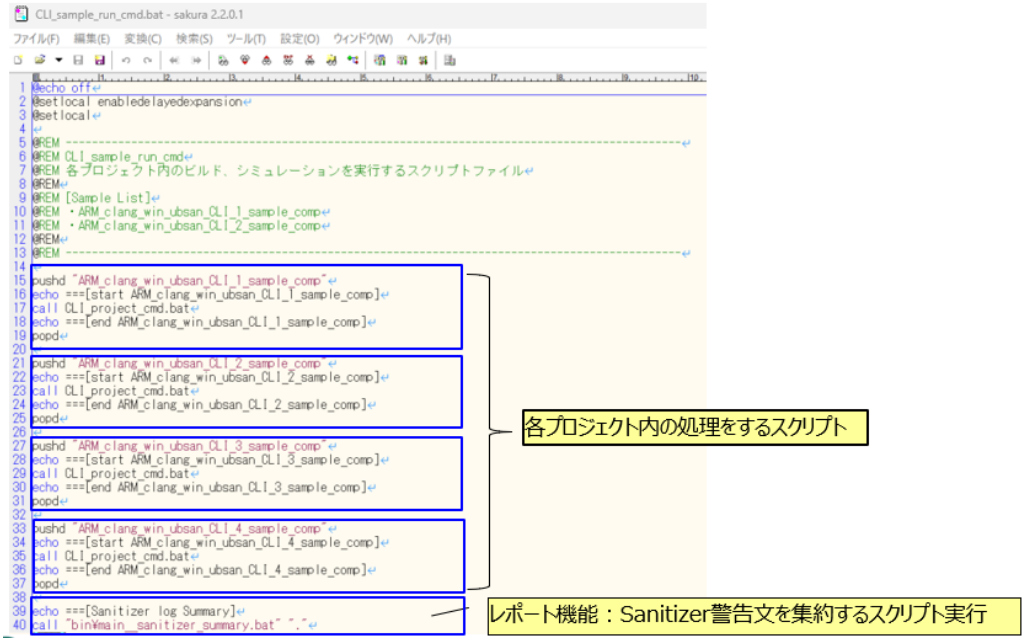


図 36 CLI_sample_run_cmd.bat 内容

6.2.3.2 CLI_project_cmd.bat

本スクリプトは、前節の CLI_sample_run_cmd.bat から呼び出されるスクリプトです。

スクリプト内容は、シミュレーション結果保存先の設定、winAMS プロジェクト内のプログラムのビルド、バイナリファイルのパス検索、OMF 変換、シミュレーション開始、シミュレーションログの退避を順に実行します。

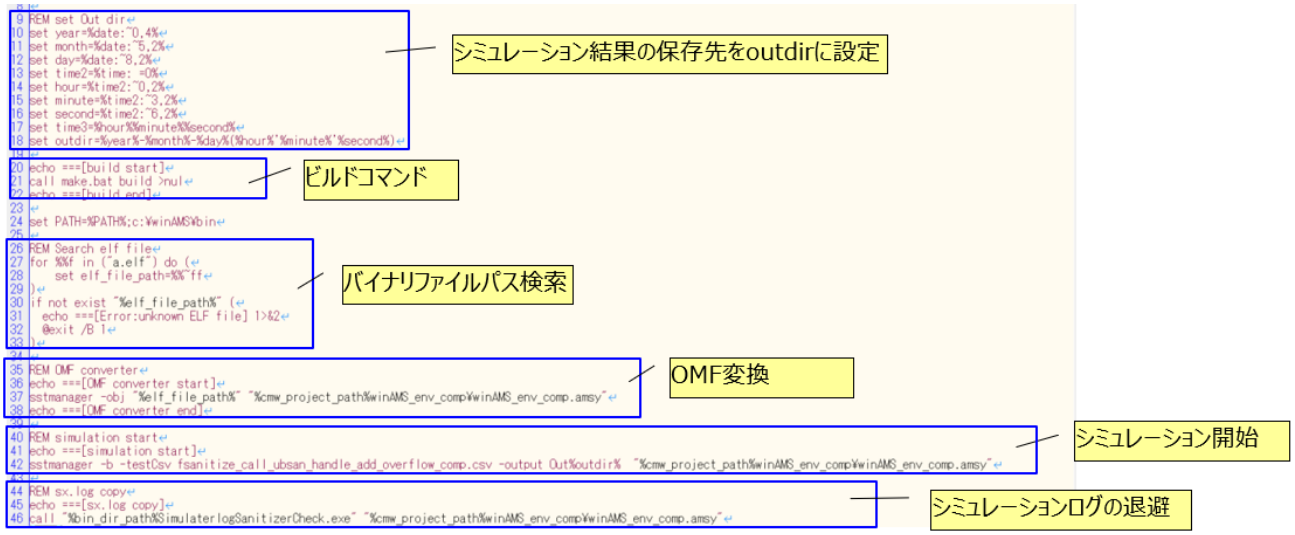


図 37 CLI_project_cmd.bat 内容

6.2.3.2.1 補足：シミュレーション開始コマンドについて

CLI_project_cmd.bat 内の「シミュレーション開始」コマンドについて補足します。今回のシミュレーション開始コマンドでは、1 つのテスト CSV(青字部分)を指定し、実行しています。

```
sstmanager -b -testCsv fsanitize_call_ubsan_handle_add_overflow_comp.csv -output Out%outdir%
"%cmw_project_path%winAMS_env_comp¥winAMS_env_comp.amsy"
```

別のテスト CSV を指定する場合は、上記の青字部分を変更します。その他のオプション、コマンドの詳細は、SSTManager のメニューから「ヘルプ」を選択し、「バッチ実行」で検索頂き、ご確認ください。

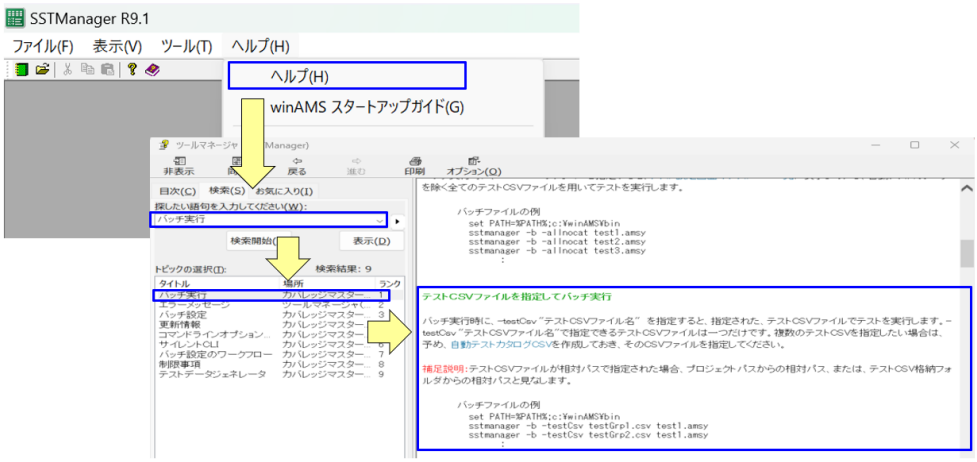


図 38 シミュレーション開始コマンドのヘルプ画面

6.2.4 スクリプトの実行手順

> SSTManager を開いている場合、SSTManager を終了させ進めてください。
> SSTManager は多重起動ができないため、スクリプト実行がエラーになります。

コマンドプロンプトを起動し、フォルダ階層を以下の「src」フォルダへ移動します。

```
>cd  C:¥work¥LLVMEmbeddedToolchainForArm-17.0.1-Windows-x86_64¥samples¥src
```

続いて、「CLI_sample_run_cmd.bat」を実行します。

```
>CLI_sample_run_cmd.bat
```

コマンドプロンプトには以下の実行ログが表示されます。CLI_1_sample_comp、CLI_2_sample_comp と順番に実行され、最後にレポート機能が実行します。

図 39 スクリプト実行ログ

処理が完了すると、以下のメッセージ(青文字部分)が出力され、レポート機能によりカレントディレクトリへ sanitizer_report.html、sanitizer_report.csv が生成されます。

```
===[complete create sanitizer summary csv file ] Export C:¥work¥LLVMEmbeddedToolchainForArm-17.0.1-Windows-x86_64¥samples¥src¥sanitizer_report.csv
===[create sanitizer data html file]
===[complete create sanitizer summary html file ] Export C:¥work¥LLVMEmbeddedToolchainForArm-17.0.1-Windows-x86_64¥samples¥src¥sanitizer_report.html
===[delete "C:¥work¥LLVMEmbeddedToolchainForArm-17.0.1-Windows-x86_64¥samples¥src¥bin¥analysis_report.temp"]
```

6.2.5 レポート内容

レポート機能により生成されるレポート内容について説明します。生成されるレポートは、HTML 形式と CSV 形式の 2 種類です。

- sanitizer_report.html
- sanitizer_report.csv

6.2.5.1 HTML 形式レポート(sanitizer_report.html)

HTML 形式レポートは「src」直下の sanitizer_report.html ファイルです。ファイルをダブルクリックしブラウザで内容を確認していきます。

レポート内容は、「ヘッダー部」と「警告の概要部」に別れます。警告概要部には各 winAMS プロジェクトの Sanitizer 警告の結果が表示されます。

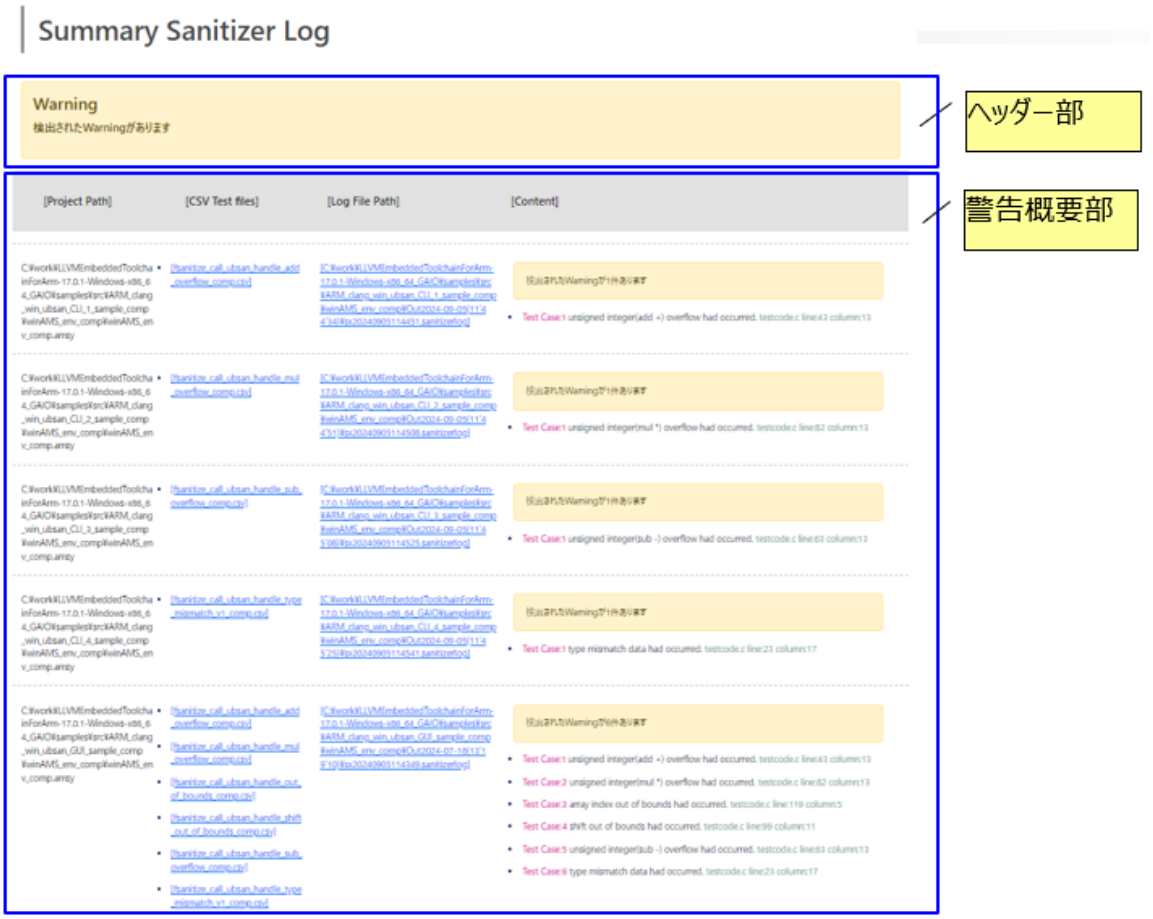
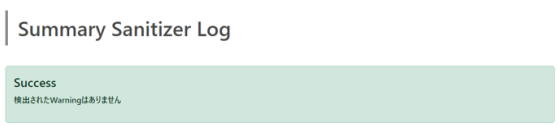
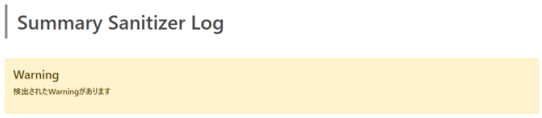
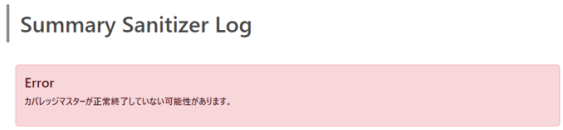


図 40 HTML 形式レポート

6.2.5.2 [ヘッダー部]

ヘッダー部は、以下の表の通り警告概要部として 3 種類存在します。

表 6-4 ヘッダー部 警告種別

ヘッダー部分	説明
 図 41 ヘッダー部 警告なし画面	警告概要部に警告がない場合、「Success 検出された Warning はありません」と表示します。
 図 42 ヘッダー部警告あり画面	警告概要部に警告内容が 1 件以上ある場合、「Warning された Warning があります」と表示します。
 図 43 ヘッダー部エラー画面	警告概要部にエラーが 1 件以上ある場合、「Error ガバレッチマスターが正常終了していない可能性があります」と表示します。その他の表示より優先され表示します。

6.2.5.2.1 [警告概要部]

警告概要部には、「C:¥work¥LLVMEmbeddedToolchainForArm-17.0.1-Windows-x86_64¥samples¥src」配下に存在する winAMS プロジェクトすべてを対象にログ収集した結果が表示されます。

以下の通り、CLI_1～CLI_4_sample の Sanitizer 警告、前章で実施した「ARM_clang_win_ubsan_GUI_sample_comp」の Sanitizer 警告文が表示されています。

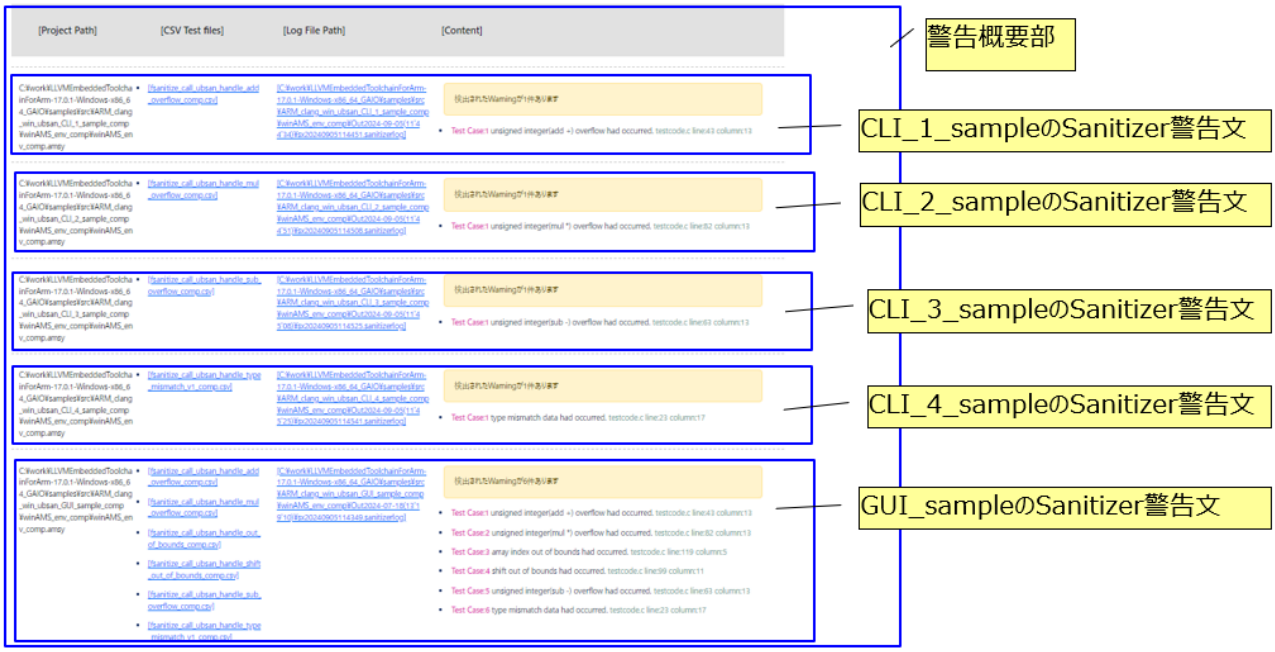


図 44 警告概要部

警告概要の詳細を説明します。大きく 4 つ情報を表示しています。

図 45 警告概要の詳細

① [Project Path]：対象フォルダ配下に存在する winAMS プロジェクトファイル

② [CSV Test files]：winAMS プロジェクトのテスト結果保存フォルダ内のテスト CSV ファイル

③ [Log File Path]：Sanitizer 警告集約を行った退避シミュレーションログファイル(拡張子 sanitizerlog)

④ [Content]：Sanitizer 警告を表示

6.2.5.2.2 [Content]の詳細について

[Content]は 3 つの部分で構成されます。

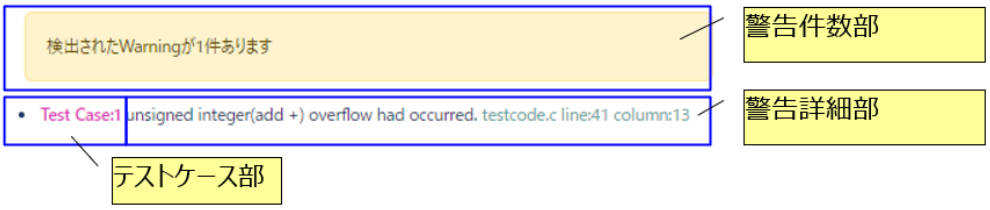


図 46 [Content]内容

- 警告件数部：警告の数を表示
- 警告詳細部：Sanitizer 警告文、対象ファイル、行番号、列番号を表示
- テストケース部：Test Case [数字] シミュレーションログのテスト関数の順番を表示

テストケース部の詳細について

CLI では 1 シミュレーションで 1 つのテスト CSV をシミュレーションしているため、「Test Case:1」と表示されます。1 シミュレーションで複数のテスト CSV を実行している GUI 版のシミュレーションログを例に説明します。

以下の図のように、%SX-I-XIPC 毎にテスト CSV によるシミュレーションが実行され、Sanitizer 警告文が出力されています。テストケース部はシミュレーションログを上から順番に Test Case1、Test Case2・・・Test Case6 と表示しています。

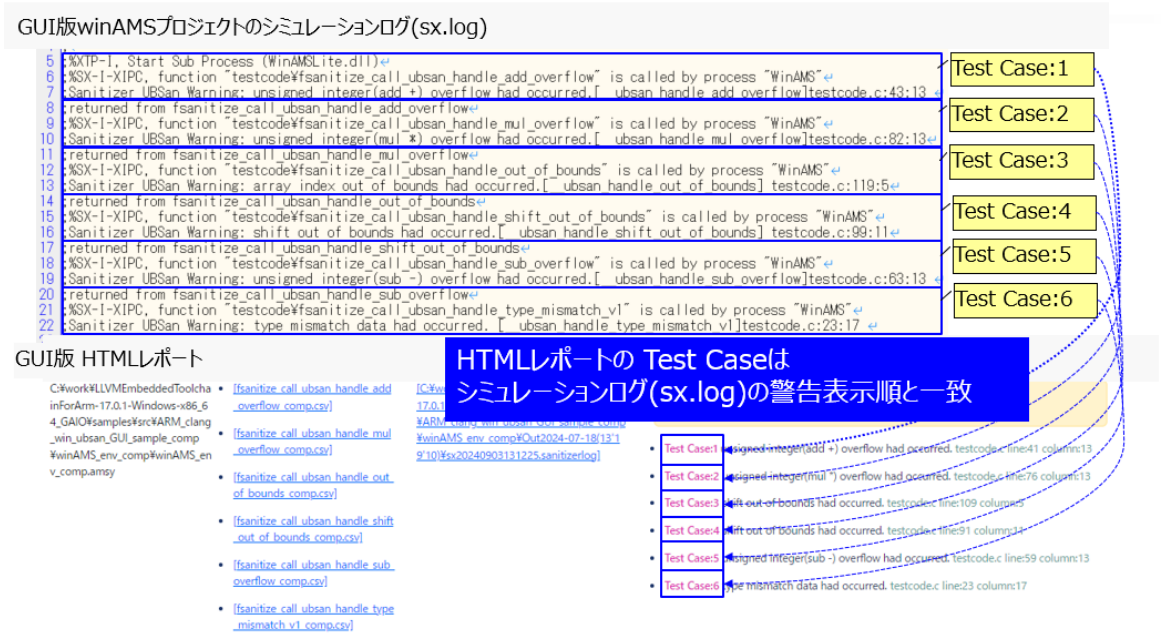


図 47 シミュレーションログ内の Sanitizer 警告順番と HTML レポート内のテストケース部の関係

6.2.5.3　　CSV 形式レポート(sanitizer_report.csv)

CSV 形式レポートは HTML 形式同様に「src」直下の sanitizer_report.csv ファイルです。CSV 形式レポート内容は、HTML 形式と同様になります。データ整理や集計時にご活用ください。

	A	B	C	D	E	F	G	H	I	J
1	[Project Path]	[CSV Test File]	[Log File Path]	[Content]	[Test Case]	[Warning Message]	[File Name]	[Line No]	[Column No]	
2	ARM_clang_win_ubsan_CLI_1_sample_comp¥winAMS_env_comp¥winAMS_env_comp.amsy	・fsanitize_call_ubsan_handle_add_overflow_comp.csv	C:¥work¥LLVMEmbeddedToolcl	検出されたWarningが1件あります	Test Case:1	unsigned integer(add +) overflow had occurred.	testcode.c	43	13	
3	ARM_clang_win_ubsan_CLI_2_sample_comp¥winAMS_env_comp¥winAMS_env_comp.amsy	・fsanitize_call_ubsan_handle_mul_overflow_comp.csv	C:¥work¥LLVMEmbeddedToolcl	検出されたWarningが1件あります	Test Case:1	unsigned integer(mul *) overflow had occurred.	testcode.c	82	13	
4	ARM_clang_win_ubsan_CLI_3_sample_comp¥winAMS_env_comp¥winAMS_env_comp.amsy	・fsanitize_call_ubsan_handle_sub_overflow_comp.csv	C:¥work¥LLVMEmbeddedToolcl	検出されたWarningが1件あります	Test Case:1	unsigned integer(sub -) overflow had occurred.	testcode.c	63	13	
5	ARM_clang_win_ubsan_CLI_4_sample_comp¥winAMS_env_comp¥winAMS_env_comp.amsy	・fsanitize_call_ubsan_handle_type_mismatch_v1_comp.c	C:¥work¥LLVMEmbeddedToolcl	検出されたWarningが1件あります	Test Case:1	type mismatch data had occurred.	testcode.c	23	17	
6	ARM_clang_win_ubsan_GUI_sample_comp¥winAMS_env_comp¥winAMS_env_comp.amsy	・fsanitize_call_ubsan_handle_add_overflow_comp.csv - f	C:¥work¥LLVMEmbeddedToolcl	検出されたWarningが6件あります	Test Case:1	unsigned integer(add +) overflow had occurred.	testcode.c	43	13	
7					Test Case:2	unsigned integer(mul *) overflow had occurred.	testcode.c	82	13	
8					Test Case:3	array index out of bounds had occurred.	testcode.c	119	5	
9					Test Case:4	shift out of bounds had occurred.	testcode.c	99	11	
10					Test Case:5	unsigned integer(sub -) overflow had occurred.	testcode.c	63	13	
11					Test Case:6	type mismatch data had occurred.	testcode.c	23	17	
12										

図 48 CSV 形式レポート

6.3　レポート内容の留意点

今回のチュートリアルでは、「GUI 版単独 winAMS プロジェクトの実行手順」と「CLI を利用した複数 winAMS プロジェクトの連続実行手順」を行いました。

「GUI 版単独 winAMS プロジェクトの実行手順」では、複数のテスト CSV をシミュレーション実行した際に複数の Sanitizer 警告が検出されましたが、Sanitizer 警告のテストケース部から該当するテスト CSV は特定する事ができません。

例えば、以下の[CST Test files]に記載の①CSV Test ファイルの順番と[Content]内の②Sanitizer 警告前のテストケース部の順番には関連がありません。

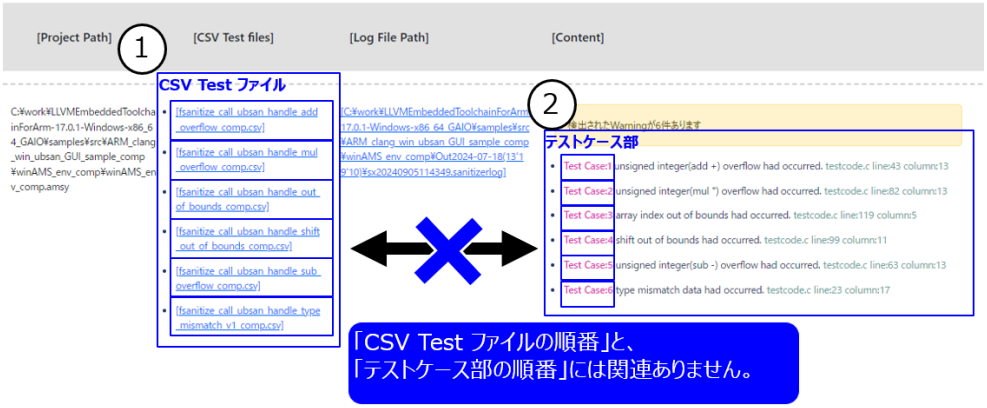


図 49　CSV テストファイル順番とテストケース部

その為、本件を回避するためには、「テスト設定」にて作成するテスト CSV 1 つとし、1 回のシミュレーションを行い運用いただくようお願いします。

以上になります。

「カバレッジマスターwinAMS と Sanitizer の連携」チュートリアルをご利用いただき、ありがとうございました。

「カバレッジマスターwinAMS と Sanitizer の連携」チュートリアル

※会社名・商品名は各社の商標または登録商標です。
※本資料の無断転載、複写は禁止しております。

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

■使用方法に関する問い合わせ方法
お問い合わせは、ユーザーサポートお問い合わせフォームへご連絡下さい。
ユーザーサポート窓口への質問には、ユーザーID が必要です。
お問い合わせの際に、ユーザーID をお知らせください。
※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