**Spezifikation zu der auf einer beliebigen Hardware lauffähigen BTE-Software**

1. Motivation

Das sogenannte Basic Test Environment (BTE) ist ein in Vector entwickeltes und eingesetztes Softwaretool, um auf einem PC hardwareunabhängige (AUTOSAR-) BSW-Komponeneten zu testen [1]. Eine gute Darstellung der obigen Funktionalitäten wird in Bild Figure 1 wiedergegeben.

Component Under Test

Framework:

Emulated ECU Environment

ECU/OS APIs

Interface Components (Stubs)

PC Environment

Log/Report

Testplan

Figure 1 Framework-Emulation of ECU environment on a PC. Entnommen aus [1].

Die in Figure 1 vorgestellte BTE-Version bietet verschiedene Testsfunktionalitäten, wie das Emulieren einer inneren ECU-Umgebung, die Ereignisprotokollierung und das Erstellen von entsprechenden Reports[1].

Gegenüger dieser Vorteile gibt es jedoch auch einen Nachteil, der sich als Einschränkung äußert. Und zwar lassen sich die jeweiligen Testreports lediglich auf einem PC erstellen, da die BTE betriebssystemabhängige Funktionen zur formatierten Ein- und Ausgaben von Strings verwendet.

Es wäre diesbezüglich auch vorteilhalft, dass die Komponententests, die die BTE anbietet, direkt auf der ECU lauffähig sind, um die Tests unter realen bzw. Echtzeitbedingungen durchzuführen.

1. Aufgabenstellung

Die Aufgabestellung basiert auf der in Kapitel 1 beschriebene Möglichkeit, das BTE-Tool so zu erweitern, dass dieses auch auf der Hardware lauffähig ist, wo gleichzeitig die CUT laufen soll.

1. Durchführung:
   1. Ansatz

Mit vorhandenen und zusätzlich eingebauten Präprozessor-Direktiven #define wird die Möglichkeit geboten, die Methodensaufrufe auszublenden, die zu viele Ressourcen brauchen oder auf der Zielhardware nicht kompilierbar sind. Letztere sind vor allem Methodensaufrufe wie: fopen, fclose, sprintf, usw.

Mit Hilfe der neuen Direktive #define USE\_PRINTF werden alle Methodensaufrufe ausgeblendet, die lediglich mit Zeichenketten umgehen. Dies ermöglicht somit solche Methoden nur dann einzusetzen, wenn die Hardware genug RAM zur Verfügung stellt.

Ein Ausblenden von Methodensaufrufe, die mit dem Erstellen und Bearbeiten von Dateiobjekten umgehen, ist durch das Umdefinieren der schon vorhandenen Direktive #define BTE\_ENABLE\_TESTREPORT möglich. Diese Methoden sind in der Regel aufgrund der limitierten Hardwareressourcen nicht einsetzbar und können beim Erstellen eines Reports auf dem betrachteten, eingebetteten System nicht verwendet werden.

Erstes Speicherlayout

Folgende Spezifikationen gelten für die Struktur der Liste, die über ein struct verwaltet wird:

typedef struct stBteEventLog

{

uint16 code;

uint32 data;

} tBteEventLog;

typedef struct stBteLogList

{

uint16 size;

tBteEventLog elem[kBteLogList\_size];

} tBteLogList;

Das Protokoll, mit dem die Daten auf der RAM gespeichert werden, sieht wie folgt aus:

|  |  |
| --- | --- |
| **Kennung** | **Beschreibung** |
| size | Größe der Liste |
| 0x66 | Kennung für einen neuen Testcase |
| Id | ID-Nummer eines neuen Testcases |
| (numParam<<8) | (pEvent->type) | (Anzahl der event-Parameter<<8)|(Event-Typ) |
| ((pEvent->time)<<16)|((pEvent->comp)<<8)|(pEvent->code) | (Zeitstempel<<16)|(Component-ID<<8)|(Code-ID) |
| 0x21 | Kennung für 1. Datenelement |
| data1 | 1. Datenelement |
| 0x22 | Kennung für 2. Datenelement |
| data2 | 1. Datenelement |
| 0x23 | Kennung für 2. Datenelement |
| data3 | 1. Datenelement |
| 0x24 | Data4 |
| data4 | 1. Datenelement |
| 0x25 | Data5 |
| data5 | 1. Datenelement |
| 0x66 | Kennung für einen neuen Testcase |
| … | … |

Aufgrund des 32-Bit großen Teilelements uint32 data der definierten Struktur tBteEventLog, wird vom Compiler einen 32-Bit großen Speicherplatz für jedes Element der Liste tBteEventLog elem[kBteLogList\_size] reserviert. Letzteres wird unabhängig von der Größe der restlichen Elemente der Liste, nämlich uint16 code, durchgeführt, obwohl diese Varable 16-Bit groß ist wie angegeben.

Dieser Nebeneffekt ist bezüglich des unnötig reservierten Speicherplatzes unerwünscht. Weitere Einstellungen könnte man beim Compiler durchführen, damit dieser Sachverhalt vermieden wird. Letzteres ist aber mit mehr Aufwand verwunden als notwendig.

Zweites Speicherlayout

Der oben genannte Nachteil lässt sich dadurch umgehen, dass die Ergebnisse vom Report in ein Byte-Array gespeichert werden:

static uint8 BteLogArray[kBteLogArray\_size]

Dadurch ist man nicht mehr von einer festen, vorgegebenen Speichergröße (32-Bit) der Datenelemente abhängig. Je nach Größe der betrachteten Variablen, die beim Report gespeichert werden sollen, geht man so vor, dass diese softwareseitig in 1-byte große Zahlen aufgesplittert und somit in die Einträge des Bytearrays gespeichert werden. Die Struktur, wie die Liste auf der RAM gespeichert wird, wird im Folgenden angegeben:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| FF | TC\_ID | TC\_ID | NumParam<<4|(pEvent->type) |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Hinweis: Die Speicherstruktur wird von links nach rechts gelesen

Ergebnis:

Inhalt dieses Berichtteils: Welche Files wurden erstellt und welche Einstellungen müssen gemacht werden.

Die Quelldatei BteLogList.c und das entsprechende Headerfile BteLogList.h wurden erstellt, um das Report lokal auf der RAM der Hardware zu speichern.

**Bedienungsanleitung und Einstellungen von meinem C-File Library:**

Neu erstellte C- bzw. H-Files:

Durch die Direktive #define kBteLogArray\_size ist es möglich, die Größe des Bytearrays anzugeben.

Ein Perl-File wurde erstellt, um die Transformation der Log-Daten in einen vtr-Report durchführen zu können.

**Bedienungsanleitung von meinem Perl Skript und user file:**

Perl-Script:

User File:

ToDo:

Man müsste untersuchen, wie der Umfang eines entsprechenden Tests sich auf die Größe des Bytearrays auswirkt. Wie kann man am sinnvollsten einen Zusammenhang dazwischen erkennen.

Reference Documents

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Source | Title | Version |
|  | vismas, vispkg | [Basic Test Environment](http://wiki.vi.vector.int/wiki-pes/BTE) | 1.00.00 |
|  | visra | [Data And State Consistency (PowerPoint)](https://vistrpscsvn1.vi.vector.int/svn/Presentations/SoftwareDevelopment/InternalTraining_DataAndStateConsistency/trunk/DataAndStateConsistency.ppt) | 1.00.00 |