Framework for Worst Case Performance Test for MCAL Drivers

Thesis submitted in partial fulfilment of the requirements for the award of degree of

Master of Engineering

In

Computer Science and Engineering

Submitted By

Vinny Ohri

(801632056)

Under the supervision of:
Dr. Rinkle Rani
Associate Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
PATIALA – 147004

JUNE 2018

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **Framework for Worst Case Performance Test for MCAL Drivers**,in partial fulfillment of the requirements for the award of degree of Master of Engineering in Computer Science and Engineering submitted in Computer Science and Engineering Department of Thapar Institute of Engineering and Technology, Patiala, is an authentic record of my own work carried out under the supervision of ***Dr. Rinkle Rani*** and refers other researchers work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Vinny

(Vinny Ohri)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Rinkle 27/06/18

(Dr. Rinkle Rani)

Associate Professor, CSED

Acknowledgement

I would like to thank all those people whose support and cooperation has been invaluable asset during the course of this project. First and foremost, I would like to thank **Dr. Maninder Singh**, Head of Department,Computer Science and Engineering Department, Thapar Institute of Engineering and Technology for his moral support towards completing my project work. I also extend my cordial thanks to **Infineon Technologies India Pvt. Ltd., Bangalore** for providing me an opportunity to carry out this project.

I extend my most profound gratitude, immense pleasure and heartfelt thanks to my esteemed guide **Dr. Rinkle Rani** for her positive attitude, excellent guidance, constant encouragement, invaluable co-operation and generous attitude.I convey my gratitude to **Dr. Ashutosh Mishra**,PG Coordinator for the motivation and inspiration for the completion of this thesis.

Last but not the least I would like to express my heartfelt thanks to my parents and my friends who with their thought provoking views, veracity and whole hearted cooperation helped in doing this thesis.

Vinny Ohri

(801632056)

Abstract

For real-time systems, correctness of a program not only depends on the produced computational results, but also on its ability to deliver on time to satisfy specified time constraints. The obvious example is safety-critical hard real time systems, such as automobiles, avionic applications, for which failure to meet a specified deadline not only renders the computations useless, but also can have catastrophic consequences.

The speed and density of multi-core SoCs have outgrown traditional debugging methodologies and makes computing the Worst Case Execution Time even more difficult in its target environment.

We have addressed the problem using Infineon developed technology known as the Multi-Core Debug Solution (MCDS). Using advanced on-chip trace techniques that include on-chip trigger generation, trace data compression and trace storage. MCDS provides only the relevant trace data to the debug tool.Without doing Software Instrumentation or adding pins to the chip, MCDS enables real-time, in-system debug and performance optimization non-intrusively.

Worst-case execution time analysis is the fundamental of real-time system design and is therefore,an area which has been subjected to great scientific interest for a long time.

Contents

List of Figures **vi**

1 Introduction **1**

1.1 Background 1

1.2 Motivation 2

1.3 Problem Statement 2

1.4 Objectives 2

1.5 Research Challenges 3

1.6 Scope 3

2 Related Work **4**

2.1 Current Methodology: Rapitime 6

2.2 Chipcoach 8

3 Literature Survey **10**

3.1 Tricore 10

3.2 Infineon Aurix 10

3.3 MCAL Drivers 12

3.4 Infineon's Multicore Debug Solution 13

3.5 Automotive Software Engineering Environment 14

3.6 Robot Framework 16

3.6.1 High-level Architecture 16

3.6.2 Example Robot File 17

3.6.3 Exmaple Test Data File 18

3.6.4 Log File 19

4 Multicore Debug Solution **21**

4.1 MCDS Architecture . . . 21

4.2 MCDS Features . . . 22

4.3 Trace Memory . . . 23

4.3.1 Continuous Tracing with High Speed Tool Interface . . . 23

4.4 MCDS Clock . . . 24

4.5 Functional Description . . . 25

4.5.1 Trigger logic . . . 25

4.5.2 Event logic . . . 25

4.5.3 Action logic . . . 26

4.6 MCDS Tracing . . . 27

4.6.1 Instruction tracing . . . 27

4.6.2 Flow tracing . . . 27

4.6.3 Function tracing . . . 28

4.6.4 Data tracing . . . 28

4.6.5 Bus tracing . . . 28

4.6.6 OLDA trace . . . 29

4.7 Aurix Tools . . . 29

4.7.1 DAS . . . 30

4.7.2 MCDS TraceViewer (MTV) . . . 31

5 Implementation and Experimental Evaluation **41**

5.1 Execution Flow . . . 41

5.2 Methodology . . . 42

5.3 Design of the Library (ExecutionTraceLib) . . . 43

5.4 Execution of the Library . . . 45

5.5 Configuring Hardware Directly . . . 53

5.6 Automation with bat files . . . 54

6 Conclusion and Future Work **56**

6.1 Conclusion . . . 56

6.2 Future Work . . . 57

References **58**

List of Figures

1.1 Worst Case Execution Time 2

2.1 WCET Analysis 4
2.2 Different paths of Execution Flow 5
2.3 Architecture of Rapitime 6
2.4 Software Instrumentation done by Rapitime. 7
2.5 WCET Results from Rapita Tool. 7
2.6 Architecture of Chipcoach 8
2.7 Screenshot of Chipcoach 9

3.1 Aurix 2G Architecture 11
3.2 AUTOSAR MCAL Software Module Architecture 13
3.3 Architecture of Aurix TC39x Micro controller 13
3.4 ASEE Architecture 14
3.5 Robot Framework Architecture 16
3.6 Example of Robot File 18
3.7 Example of xml Test Data File 19
3.8 Example of Generated Log File 20

4.1 Architecture of Multicore Debug Solution 22
4.2 MCDS Clock 24
4.3 Trigger Logic 25
4.4 Trigger Logic 26
4.5 Trigger Logic 26
4.6 Instruction Tracing 27
4.7 Function tracing 28

4.8 Bus tracing 29
4.9 Block Diagram of DAS 30
4.10 .mcds File Format 32
4.11 Message Format 36
4.12 File Option 36
4.13 Device Option 37
4.14 MCDS Option 37
4.15 MTV Default Setting 38
4.16 Trigger Trace on IP address 38
4.17 Program Flow and Data Trace 39
4.18 Qualified Flow and Data Trace 40
4.19 Recording 40

5.1 Execution Flow 41
5.2 Data Trace for Global registers 42
5.3 Performance Test for MCAL Drivers 42
5.4 Integration of ExecutionTraceLib with ASEE Framework 43
5.5 Architecture of ExecutionTraceLib 44
5.6 Block Diagram of Execution Trace Library 44
5.7 Execution Setup 45
5.8 Robot File snapshot 45
5.9 Ase_env Configuration File snapshot 46
5.10 ExecutionTrace Lib snapshot 46
5.11 DAS Lib snapshot 47
5.12 DutPy.py snapshot 48
5.13 IFxMcds.py snapshot 49
5.14 MCDS.py snapshot I) 49
5.15 MCDS.py snapshot II) 50
5.16 Compiler.bat file snapshot 50
5.17 Building Modules snapshot 51
5.18 Flashing the .hex files to controller snapshot 51
5.19 Universal Validation Platform (Executing Test cases Framework) . . 51
5.20 Traces collection 52

5.21 Log File 52
5.22 Trace Results 53
5.23 Register Configuration 53
5.24 Traces 54
5.25 mcds.bat file 54
5.26 Trace Data 54

6.1 Comparison of Timings by RapiTime and MCDS 56

Chapter 1

Introduction

1.1 Background

The utilization of embedded programmable units in regular day to day existence is continually expanding.A conspicuous case for this is current automobiles.Wiring outfits are supplanted by bus frameworks, switches by savy switches and motor controllers by ground-breaking CPUs.

This takes into consideration of extra sensors and a more powerful and effective fault analysis and detection in case of any malfunction. Since the units in an automobile need to work under outrageous conditions (as respects vibration and temperature) with basically no preventive maintenance, the requirements on dependability are raised impressively.

This remains constant at the hardware level, as well as for the expanding software part in such frameworks. As these frameworks are likewise time bound frameworks, the correctness of programming in such frameworks depends on functional accuracy as well as on the on-time delivery of the figured outcomes.Most pessimistic scenario execution time (Worst Case) investigation processes upper limits on the execution time of assignments in a framework [3].

As the general issue of deciding the WCET of a subjective bit of code is undecidable (it is fundamentally the same as the stopping issue) one can't hope to acquire the correct esteem, rather, just an upper bound can be given.

Nonetheless, it is attractive for the WCET to be as tight as could reasonably be expected.

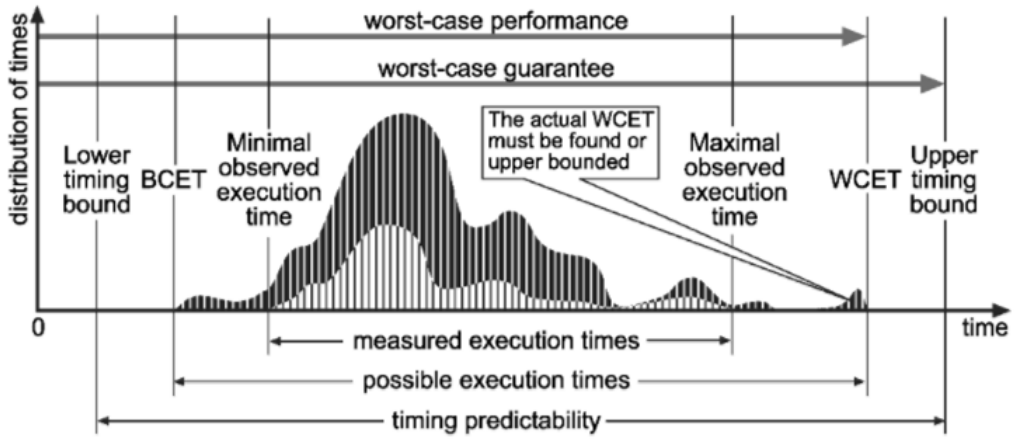


Figure 1.1: Worst Case Execution Time

1.2 Motivation

For real time frameworks, correctness of a program relies upon the final computational outcomes as well as on its capacity to convey results on time to fulfill specified time limitations.Along with these lines, for a real time application, predictability concerning time is of farthest significance.

The conspicuous case is security basic hard real time frameworks,for example-automobiles, aeronautical applications for which inability to meet a predetermined due date renders the calculations pointless as well as leads to cataclysmic outcomes.

1.3 Problem Statement

The project encapsulates Feasibility on the Performance test for Worst Case Execution Test for Microcontroller Abstraction Layer (MCAL) Drivers, Proposal on the methods using the Infineon Hardware Capability, Design and Implementation of the Worst Case Execution framework.

1.4 Objectives

- Calculate the most correct Execution time of MCAL Driver APIs with no overhead

- To locate the most proficient strategy to Data Trace for Global registers
- Design and Implementation of the Worst Case Execution Test framework
- To reduce the Interrupt Locking time for MCAL Drivers

1.5 Research Challenges

To provide guaranteed WCET with minimal pessimism for MCAL drivers running on Infineon TriCore family. Regardless of research going ahead finished the most recent thirty years, there is no real way to locate the correct on-chip most pessimistic scenario execution time for any sensibly extensive bit of code running on current processors.
Rather, the point of WCET examination when connected with regards to designing ventures must be to locate a helpful,most appropriate and legitimate estimate of the genuine WCET.

1.6 Scope

The scope of this project Framework for Worst Case Performance Test for Microcontroller Abstraction Layer (MCAL) Drivers is only to Infineon's Tricore family microcontrollers.

Chapter 2

Related Work

WCET investigation is normally happened at two levels.The low-level, which is done at object code, considering the effects of gear level features like pipelining and cache. On the other side, high level examination is performed at the source code which basically concentrating on describing conceivable paths of execution.

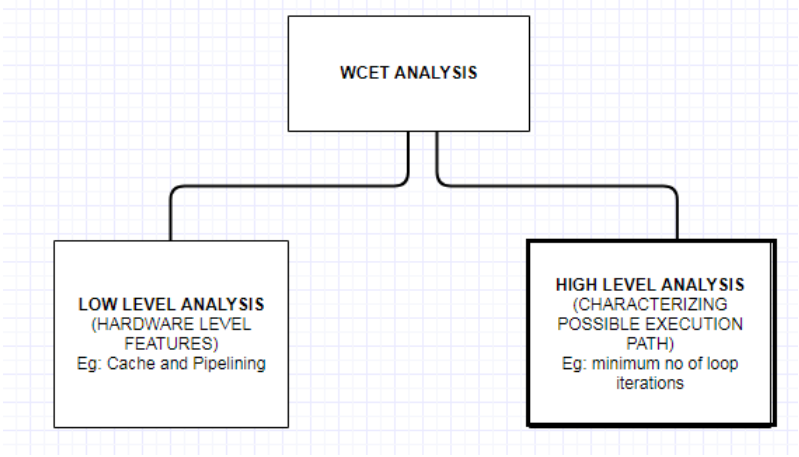


Figure 2.1: WCET Analysis

As a rule, it depends on explanations given by the users to depict code execution frequencies like maximum number of loop iterations.The problem of getting high level analysis of the WCET by characterizing the context for the executed code. With a single technique in view of symbolic articulations of the WCET, there is a need to address two sources of over estimation of high level analysis:
subprogram calls and number of loop iterations.

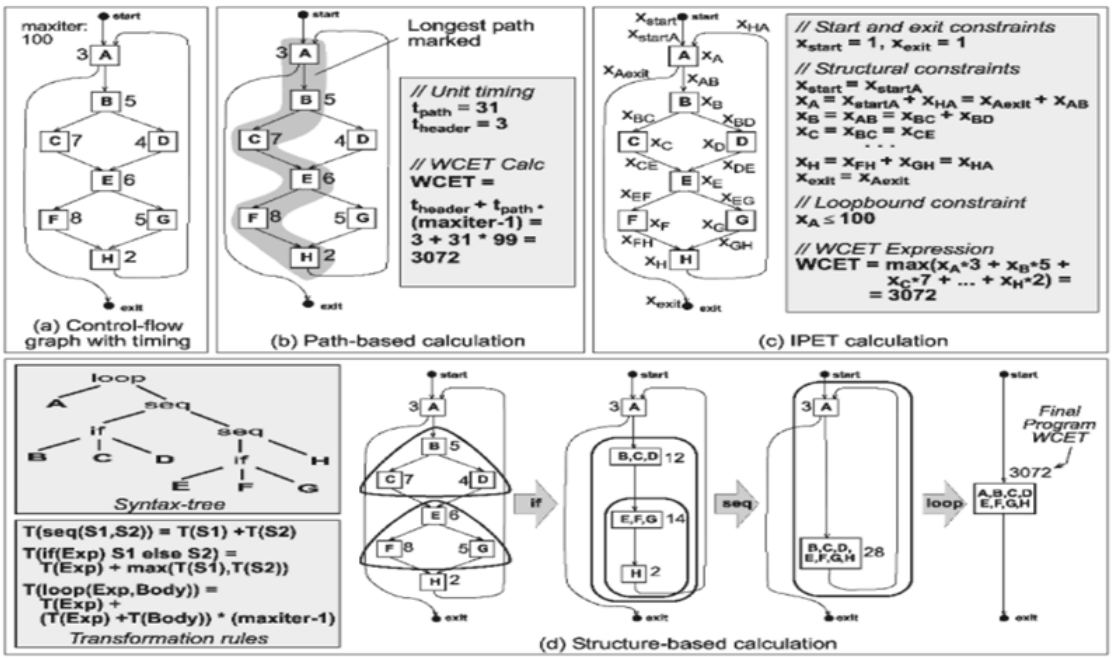


Figure 2.2: Different paths of Execution Flow

Customary systems register the WCET of a subprogram first and then, utilize this value in all the calls to it, thus not exploiting the fact that a particular call may have substantially more tightly WCET than others [4]. Similarly, a maximum loop bound is utilized to acquire the most pessimistic scenario execution time of a loop, overlooking the way that this loop might be executed by not as much as this greatest number in various contexts.

This issue can be addressed by representing the WCET of a bit of code as an algebraic articulation rather than a numerical number.

We depend on the intensity of computational polynomial math frameworks like Mathematica or Maple to control, disentangle and assess these articulations.Note that customary WCET is centered around nearby investigation and scarcely considers the context in which a bit of code is executed. Though the new strategy is all encompassing as in,it particularly addresses all conditions among different code sections.

2.1 Current Methodology: Rapitime

Rapitime is a 3rd party software by Rapita Systems, which creates programming tools for on-target confirmation, improvement and code coverage of basic real time implanted aviation and automotive softwares [3]. It is a robotized execution estimation on-target timing examination apparatus. For real-time, embedded applications, RapiTime gathers execution traces to furnish you with execution time estimation insights, helps assurance of most worst scenario execution time and aides your advancement endeavors.

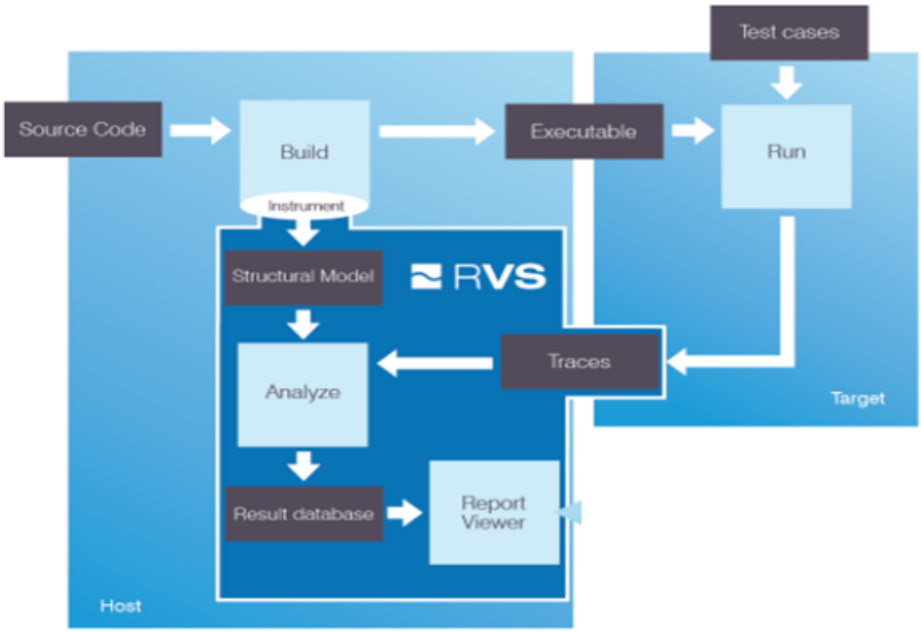


Figure 2.3: Architecture of Rapitime

To calculate WCET, RapiTime collect the Execution traces, by putting some form of Instrumentation in the start and end of the unit [11].
Instrumentation is a mechanism that is added to the code running on the target that measures the time spent executing region of code.
The Software Instrumentation (additional code) might record the start and the stop time of specific code units. This could be recorded in memory and retrieved subsequently to derive high water mark. Frequently, a safety margin is added to the value.

```
Dio_LevelType Dio_ReadChannel(Dio_ChannelType ChannelId)
{RVS_I(10);{
  uint32          PinPosition;
  Dio_PortType    PortNumber;
  uint8           PinNumber;
  Dio_LevelType   RetVal;
  const Ifx_P *GetPortAddressPtr;
  RetVal = (Dio_LevelType)0x00u;

  PortNumber = Dio_lGetPortNumber(ChannelId);
  PinNumber = Dio_lGetPinNumber(ChannelId);
  {
    GetPortAddressPtr = Dio_lGetPortAdr(PortNumber);


    PinPosition = ((uint32)(0x1U) << PinNumber);


    if ((PinPosition & (uint32)(GetPortAddressPtr->IN.U)) != (uint32)0x00u)
    {
      RetVal = (Dio_LevelType)0x01u;
    }
  }
  {RVS_I(11);return RetVal;}
}}
```

Figure 2.4: Software Instrumentation done by Rapitime.

Worst Case Execution Timing Results

SlNum	API Name	Min Time(us)	Max Time(us)	WCET(us)
1	Dio_FlipChannel	0.5800000000us	0.6600000000us	0.6600000000us
2	Dio_ReadChannel	0.5300000000us	0.5300000000us	0.5300000000us
3	Dio_ReadChannelGroup	0.3000000000us	0.3500000000us	0.3500000000us
4	Dio_ReadPort	0.3000000000us	0.3000000000us	0.3000000000us
5	Dio_WriteChannel	0.5200000000us	0.5700000000us	0.5700000000us
6	Dio_WriteChannelGroup	0.2900000000us	0.3300000000us	0.3300000000us
7	Dio_WritePort	0.3000000000us	0.3000000000us	0.3000000000us

Figure 2.5: WCET Results from Rapita Tool.

2.2 Chipcoach

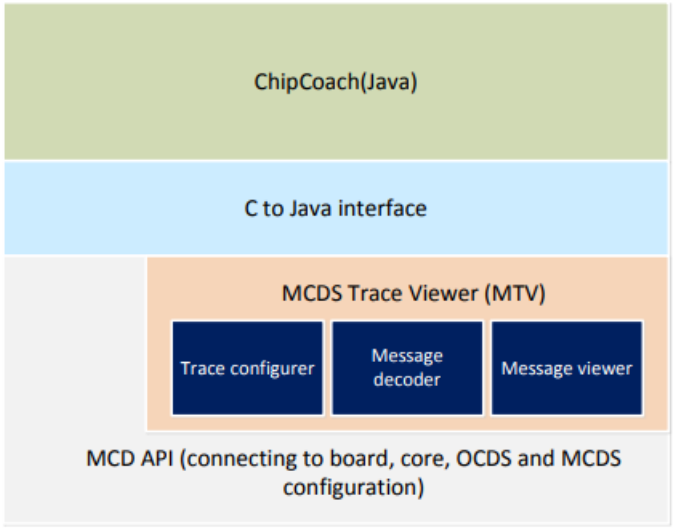


Figure 2.6: Architecture of Chipcoach

The implemented software tool named ChipCoach is a post-processing tool for embedded systems, which makes use of Infineons MCDS to gather trace data and then analyze issues based on the trace data [2]. It exploits the tracing and debugging functionalities of the MCDS infrastructure integrated in the AURIX ED. The main purpose of using ChipCoach include:

- Automatic hardware configuration check: ChipCoach is designed to monitor the configuration process of hardware modules and report the detected violations of rules, which are predefined by the SoC designer and stored in ChipCoach.
- Performance profiling and bottleneck analysis: ChipCoach are targeted at the automatic detection of several types of performance issues including shared resource contention, data locality issues, lock contention, blocked interrupts etc. These issues are usually hard-to-detect and even invisible for software developers.
- Specific module support: ChipCoach facilitates the usage of several hard-to-use or important hardware modules for instance, memory protection unit, DMA, clock control unit etc.

- System exploration: ChipCoach helps users to understand and explore their systems from different perspectives.

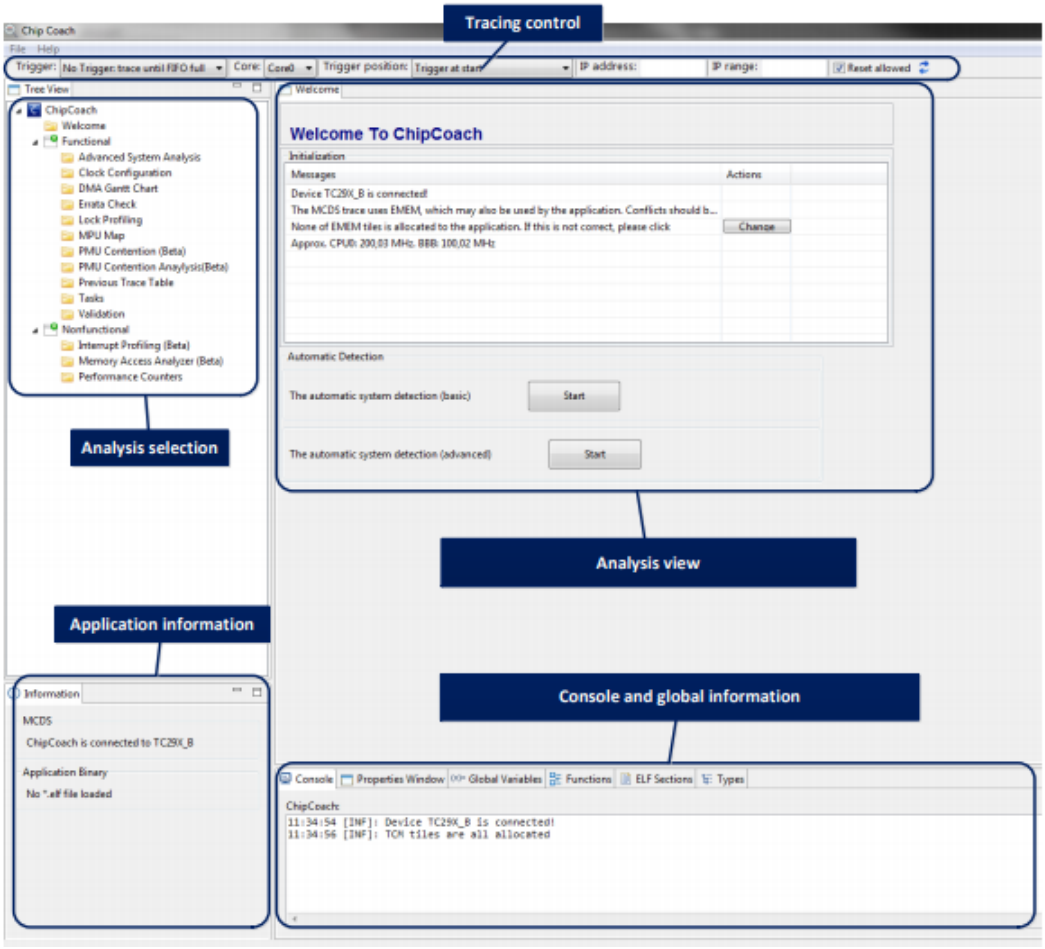


Figure 2.7: Screenshot of Chipcoach

Chapter 3

Literature Survey

3.1 Tricore

TriCore is a 32-bit microcontroller architecture from Infineon, optimized for real-time embedded systems.It unites the elements of a Reduced Instruction Set Computer (RISC) processor core, a microcontroller and a DSP in one chip package [4]. The Instruction Set Architecture (ISA) supports both 16-bit and 32-bit instruction formats. TriCore avoids long muli-cycle instructions and provides hardware-supported interrupts to reduce the interrupt latency and realtime responsiveness.

3.2 Infineon Aurix

The AURIX family is especially designed for automotive applications, e.g. Advanced Driver Assistance Systems (ADAS), powertrains and chassis control units for braking, steering and suspension [8]. There are many device classes namely AURIX TC38x, TC39x. The Infineon AURIX TC39x device is a widely-used automotive microcontroller as shown in Figure 3.1.

It joins three ground-breaking advances inside one silicon die including RISC processor design (Tricore), DSP and on-chip memory/peripherals [6]. The TC39x device comprises of six Tricore processors having frequency upto 300 MHz.

A Tricore processor has L1 data cache and program cache implemented. Both program cache and data cache are two-way set associative and least recently used (LRU) based.

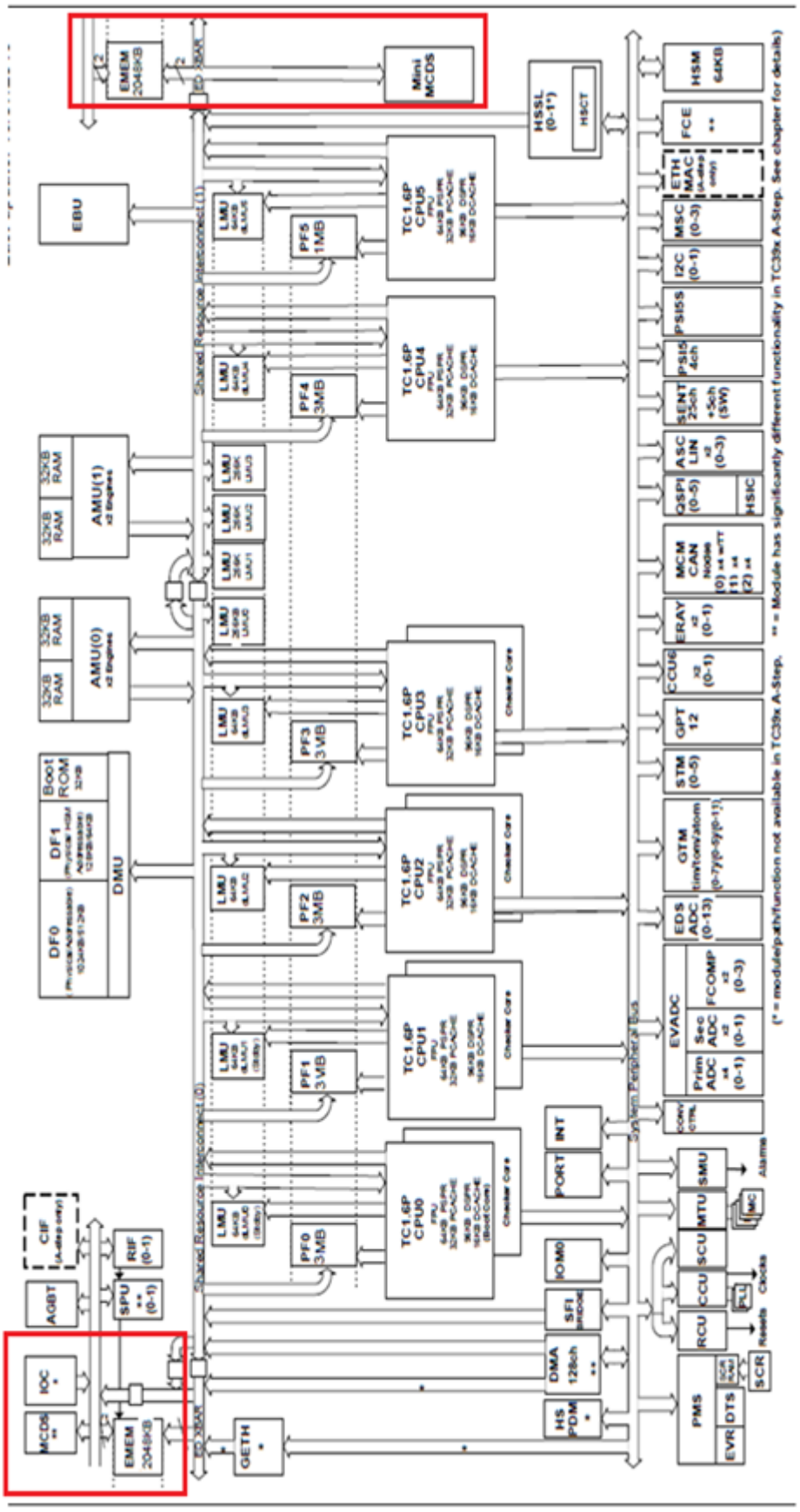


Figure 3.1: Aurix 2G Architecture

Local data memory in the core actually consists of several memory modules namely Data Scratch Pad RAM (DSPR), Data Cache (DCACHE) and cache tag memory. Similar to local data memory, local program memory has several blocks including Program Scratch Pad RAM (PSPR), PSPR and cache tag memory. Cache tag memory meant to be used as general memory blocks but for testing purpose, which is not covered in this research. All these memory blocks are Error Correcting Code (ECC) protected [7]. Global memories such as Local Memory Unit (LMU) and Program Memory Unit (PMU) are used to store data and program. Providing local memory to universally useful use and gives access to isolate blocks of Emulation and Debug Memory (EMEM) is the main task performed by LMU. Data stored in LMU is also protected by ECC. A special feature named Online Data Acquisition (OLDA) is also supported by LMU. The OLDA is a range of memory which can be written without causing errors but no memory is addressed. The PMU controls the flash memory and the boot ROM.

3.3 MCAL Drivers

MCAL remains for Microcontroller Abstraction Layer. With regards to embedded programming improvement, the MCAL can be characterized as follows:
MCAL is a product module that directly accesses to all the external devices and on-chip MCU fringe modules,which are mapped to memory.
Moreover,it makes the upper software layers (Basic programming layer, or BSW, Application Layer) free of the MCU.MCAL empowers an extremely noteworthy favorable position of the layered engineering of the AUTOSAR consistent design which makes both the application and the middleware (Basic Software layer) autonomous of the hidden equipment platform.This renders tremendous advantage to the product advancement cost and time, as there is a move from coding to configuration in the ECU configuration approach.

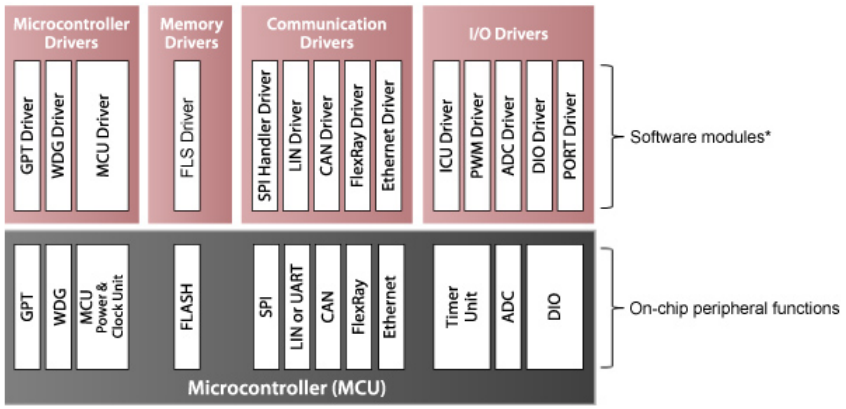


Figure 3.2: AUTOSAR MCAL Software Module Architecture

3.4 Infineon's Multicore Debug Solution

Infineon MCDS IP is designed for debugging and tracing the CPU execution, bus, or any other IP functionality during run time without halting or stopping the target execution [1] .The module is a part of Emulation Extension Chip of AURIX/AURIX-plus family (Emulation Device). The detailed information about the IP is discussed in the next Chapter.

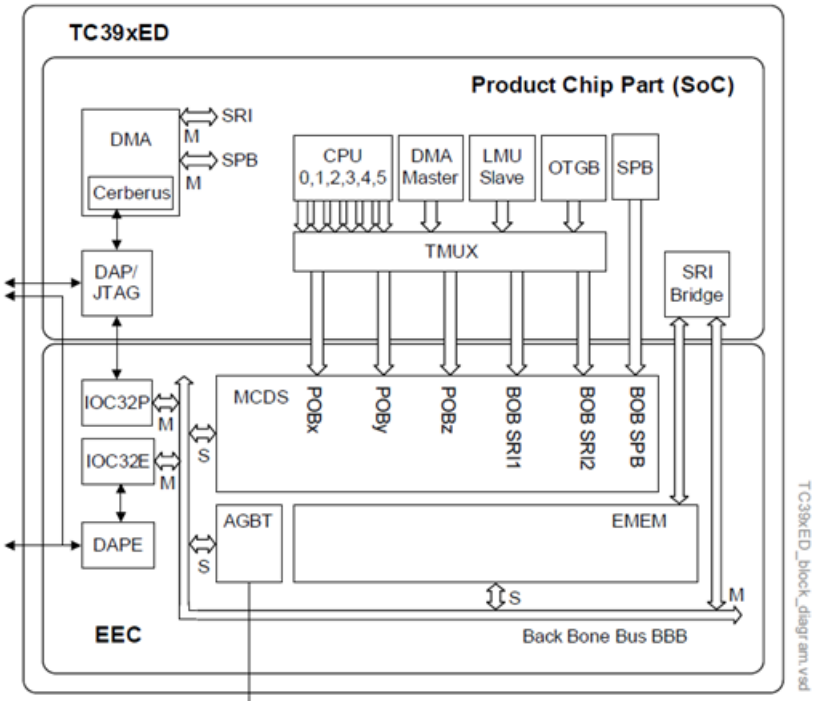


Figure 3.3: Architecture of Aurix TC39x Micro controller

3.5 Automotive Software Engineering Environment

Automotive Software Engineering Environment (ASEE) is conceptualized to provide a unified software development,testing and debug environment across all stakeholders (Development, Software Validation, System Testing, Test Automation, Continuous Integration etc.) for the Infineon AURIXTM embedded software platforms [6]. It is an umbrella of state-of-the-art tools and frameworks with high level of automation.

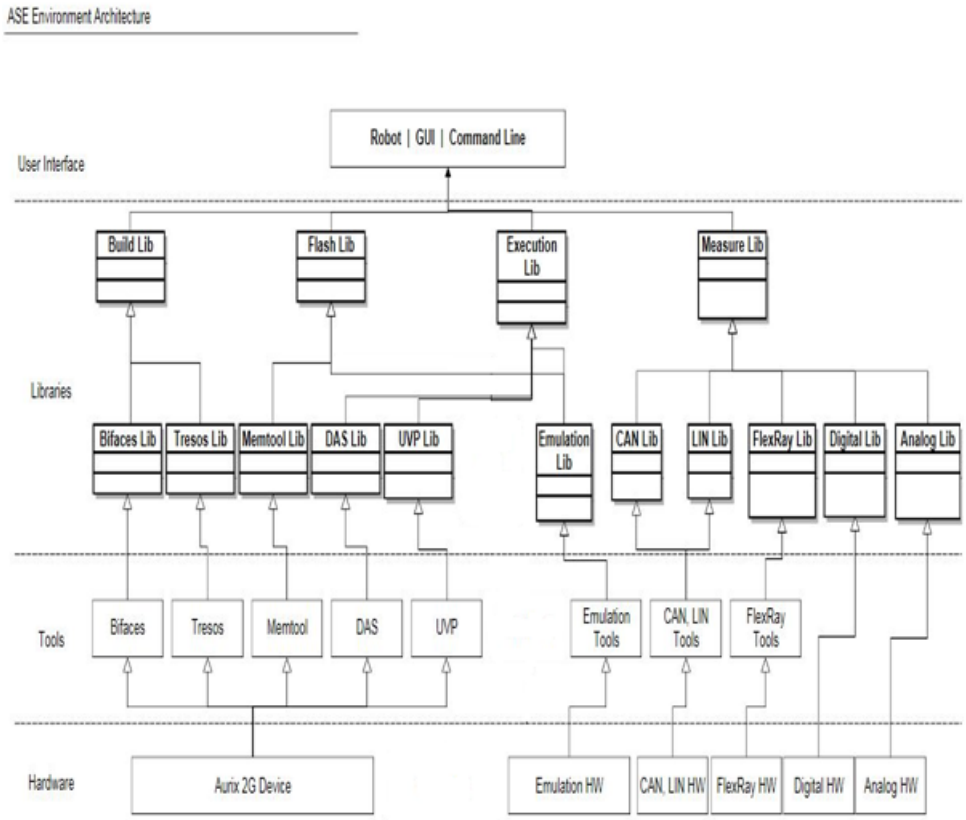


Figure 3.4: ASEE Architecture

The ASEE uses robot framework as a skeleton and extends the already available features by writing its own (python) libraries which are developed following the generic approach of robot framework. These libraries aim to encapsulate the entire complexity by providing simple keywords (functions) to interact with the device in order to build, flash, execute and/or communicate with the device [5]. From the software engineer point of view this looks like just calling simple words: "Start

Simulation", "Start Flash" and Run Test Case 2" from an Microsoft Excel, html, tsv, plain text, or robot file. The Figure 3.4 shows the architecture of Automotive Software Engineering Environment. The ASEE framework is divided into four segments User Interface, Libraries, Tools and Hardware.

User Interface: As seen from the ASEE architecture the user will not directly interact with the hardware directly. The user will interact with the hardware using keywords (API) which are provided by python libraries through Robot Framework. These python libraries are written in such a way that they can configure the hardware by using appropriate tools by taking input from configuration files. This will eliminate the difficulty that the user will face in configuring different the tools to communicate with the device and save lot of time.

Tools: Tools helps to configure the hardware based on the requirements. These tools may be developed in house or taken from the vendors. For example, MTV TraceViewer, which helps to configure, trace and decode the traces generated by MCDS.EB tresos Studio permits ECU designers and incorporates to check the consistency of arrangements and to create code for essential programming modules for an AUTOSAR standard core.

Libraries: Robot Framework is a generic automation framework which provides complex functions in a simple way using simple words, enabling the user to focus on their main topics. This is possible using Robot Framework Libraries. These libraries aim to encapsulate the entire complexity by providing simple keywords (functions) to interact with the device. These libraries aim to encapsulate the entire complexity from the user and abstracting hardware components (LIN, Flex ray, CAN) necessary to communicate between PC and Device under test. ASEE makes use of the standard libraries which are provided already along with Robot Framework to handle generic operations like String manipulations, XML parsing, operating system and so on.

Hardware: Hardware layer consists of all the hardware that is being used in the particular test environment. For example the controller used is AURIX Tricore . PCAN device is used to generate CAN traffic in some CAN test setup.

3.6 Robot Framework

Robot Framework is a python-based, extensible catchphrase driven test automation system for end-to-end acknowledgment testing and acknowledgment test-driven advancement (ATDD)[8]. It can be utilized for testing disseminated, heterogeneous applications where check requires contacting a few advances and interfaces.

The system has a rich system around it comprising of different non-exclusive test libraries and instruments that are created as independent tasks. Robot Framework is open source programming released under the Apache License 2.0.

3.6.1 High-level Architecture

Robot Framework is a non specific, application and technology autonomous system. It has an exceptionally secluded engineering as outlined in the Figure 3.5.

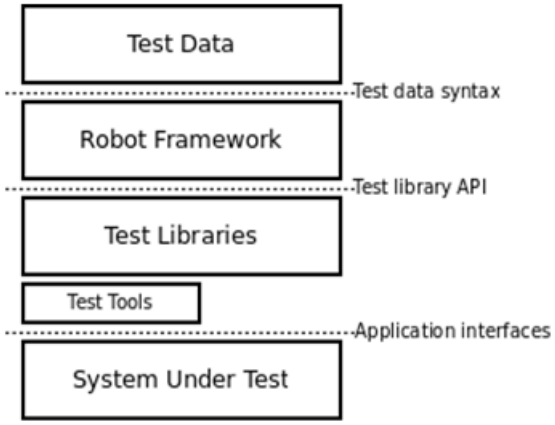


Figure 3.5: Robot Framework Architecture

The test data is in straightforward, simple to- alter tabular format. At the point when Robot Framework is begun, it forms the test data, executes test cases and creates logs and reports. The center system does not know anything about the objective under test and the connection with it is taken care of by test libraries. Libraries can either utilize application interfaces specifically or utilize bring down level test devices as drivers.

- **Test Data**

Robot Framework test data is characterized in table form, utilizing either hypertext markup language(HTML), tab-separated values (TSV), plain content, or restructuredText (reST) formats. Which format to utilize relies upon the unique condition, however the plain content format is prescribed if there are no extraordinary needs.

Robot Framework chooses a parser for the test information in light of the document augmentation. The expansion is not case-sensitive and the perceived extensions are .html, .htm and .xhtml for HTML, .tsv for TSV, .txt and spacial.robot for plain text, and .rst and .rest for reStructuredText.

- **Test Libraries**

  Robot Framework's actual testing abilities are given by test libraries. There are numerous current libraries, some of which are even packaged with the core framework, however there is still a need to make new ones.

  Python classes resides inside a module.If the name of a class implementing a library is exactly same as the name of the module, Robot Framework permits dropping the class name when bringing in the library. For instance, class MyLib in MyLib.py record can be utilized as a library with simply name MyLib.

  This additionally works with sub modules so that if, for instance, parent.MyLib module has class MyLib, bringing in it utilizing just parent.MyLib works.

3.6.2 Example Robot File

The users can interact with the ASEE through Robot file. The user can control what has to be done using Robot file. User will use the APIs (keywords) provided by different ASEE python libraries to define what has to be done in what order for a particular test condition. The Figure 3.6 show how a simple Robot file looks like.

```
*** Settings ***
Documentation          Keyword-driven example test cases.
...
# This way it is possible to include keywords from other files
Library                BuildLib
Library                UVPLib          ${COM_PORT}
Library                FlashLib

Suite Setup            Suite Init
Suite Teardown

*** Variables ***

*** Keywords ***
Suite Init
    [Documentation]    Suite init sequence
    Setup Suite
    Init Build Lib
    ${bin_file} =   Start Build
    Init Flash Lib
    Start Flash    ${bin_file}
    Reset Device
    Connect To Dut

Suite Delete
    [Documentation]    Suite delete sequence

*** Test Cases ***

Test Case Demo
    Log To Console  Hello ASE_ENV 2.0!

Run_Dio_Demo
    Run Demoapp 0334x
Run_Dma_Demo
    Run Demoapp 04
```

Figure 3.6: Example of Robot File

3.6.3 Exmaple Test Data File

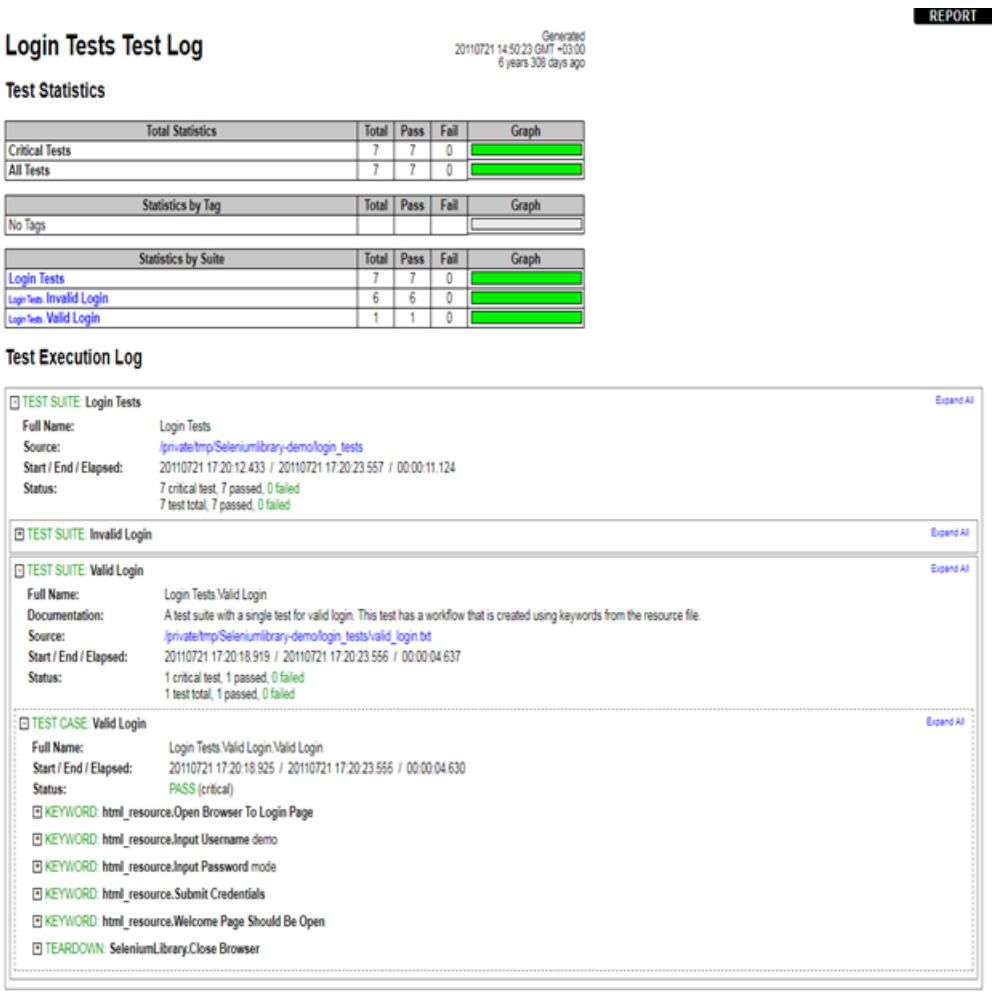
Figure 3.7 demonstrates an illustration Test Data record which is .xml format. The user can send the data required through this file to the test libraries. As soon as the Robot Framework is started,processing of the test data gives the required data.

```
<root>
  <AseMain>
    <ASE_VERSION>2.0</ASE_VERSION>
    <PYTHON_PATH sys_path="yes">tools/python</PYTHON_PATH>
    <RESULT_PATH>../demo/results</RESULT_PATH>
    <ASE_LIB>tools/ase_lib</ASE_LIB>
    <DOC_PATH>doc</DOC_PATH>
    <LOG_LEVEL>DEBUG</LOG_LEVEL>
    <ECLIPSE_PATH>c:/sofit/aurix2g_sw_mcal/eclipse/eclipse.exe</ECLIPSE_PATH>
    <TOOLS_BIN sys_path="yes">tools/bin</TOOLS_BIN>
  </AseMain>
  <RobotLib>
    <ROBOT_OPTIONS/>
    <REBOT_OPTIONS/>
    <ROBOT_TAGS/>
  </RobotLib>
  <TresosLib>
    <TRESOS_ROOT>C:/sofit/aurix2g_sw_mcal/tresos/tresos</TRESOS_ROOT>
    <TRESOS_CFG_MODE>XDM</TRESOS_CFG_MODE>
    <TRESOS_WORKSPACE>1_ToolEnv/2_Tresos</TRESOS_WORKSPACE>
    <TRESOS_PROJECT_NAME>DemoApp</TRESOS_PROJECT_NAME>
    <TRESOS_OUT_DIR>0_Src/AppSw/CfgMcal</TRESOS_OUT_DIR>
  </TresosLib>
  <BuildLib>
    <BUILD_TOOL>BifacesLib</BUILD_TOOL>
    <COMPILER_PATH>
      <Gnuc_PATH>c:\sofit\aurix2g_sw_mcal\hightec\bin</Gnuc_PATH>
      <Tasking_PATH>c:\sofit\aurix2g_sw_mcal\tasking\ctc\bin</Tasking_PATH>
    </COMPILER_PATH>
    <SRC_FOLDERS>../demo/extern/McIsar, ..\demo\DemoWorkspace\McalDemo\0_Src</SRC_FOLDERS>
    <INCREMENTAL>YES</INCREMENTAL>
    <DEVICE>AURIX2G</DEVICE>
    <COMPILER>Gnuc</COMPILER>
    <DERIVATE>TC399</DERIVATE>
    <AUTOSAR>422</AUTOSAR>
    <BUILD_DEPENDENCY/>
    <BUILD_DISCARD/>
    <RLM_LICENSE>1748@ulicserv1.muc.infineon.com</RLM_LICENSE>
  </BuildLib>
```

Figure 3.7: Example of xml Test Data File

3.6.4 Log File

Robot framework creates reports and log documents after the test execution, Figure 3.8 demonstrates an example Log File.Log files contain insights about the executed experiments in HTML format. They have a hierarchical structure demonstrating test suite, test cases and catchphrase points of interest. Log records are required each and every time when test outcomes about are to be investigated in detail. Despite the fact that log records likewise have insights, reports are better to get a more detailed and high level overview.



REPORT

Login Tests Test Log

Generated
20110721 14:50:23 GMT +03:00
6 years 308 days ago

Test Statistics

Total Statistics	Total	Pass	Fail	Graph
Critical Tests	7	7	0	
All Tests	7	7	0	

Statistics by Tag	Total	Pass	Fail	Graph
No Tags				

Statistics by Suite	Total	Pass	Fail	Graph
Login Tests	7	7	0	
Login Tests Invalid Login	6	6	0	
Login Tests Valid Login	1	1	0	

Test Execution Log

TEST SUITE: **Login Tests** Expand All

Full Name: Login Tests
Source: /private/tmp/Seleniumlibrary-demo/login_tests
Start / End / Elapsed: 20110721 17:20:12.433 / 20110721 17:20:23.557 / 00:00:11.124
Status: 7 critical test, 7 passed, 0 failed
7 test total, 7 passed, 0 failed

TEST SUITE: **Invalid Login** Expand All

TEST SUITE: **Valid Login** Expand All

Full Name: Login Tests.Valid Login
Documentation: A test suite with a single test for valid login. This test has a workflow that is created using keywords from the resource file.
Source: /private/tmp/Seleniumlibrary-demo/login_tests/valid_login.txt
Start / End / Elapsed: 20110721 17:20:18.919 / 20110721 17:20:23.556 / 00:00:04.637
Status: 1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed

TEST CASE: **Valid Login** Expand All

Full Name: Login Tests.Valid Login.Valid Login
Start / End / Elapsed: 20110721 17:20:18.925 / 20110721 17:20:23.555 / 00:00:04.630
Status: PASS (critical)

KEYWORD: **html_resource.Open Browser To Login Page**
KEYWORD: **html_resource.Input Username** demo
KEYWORD: **html_resource.Input Password** mode
KEYWORD: **html_resource.Submit Credentials**
KEYWORD: **html_resource.Welcome Page Should Be Open**
TEARDOWN: **SeleniumLibrary.Close Browser**

Figure 3.8: Example of Generated Log File

Chapter 4

Multicore Debug Solution

4.1 MCDS Architecture

The hardware-level operations are gathered by MCDS via observation blocks. There are two types of observation blocks: Processor Observation Block (POB) for core tracing and Bus Observation Block (BOB) for bus tracing.
The Shared Resource Interconnect (SRI) allows parallel transaction among different hardware modules.
The System Peripheral Bus (SPB) mainly connects CPU's, peripherals. In order to trace both bus transfers, two BOBs for both SRI and SPB are embedded in MCDS [9].
It creates time stamps on the buffered trace messages. On the other hand, it also handles the trace qualification across core boundaries. For example if a core executes a specific routine, then bus tracing is enabled . Debug Memory Controller (DMC) is an equipment module that is in charge of bundling tracing messages into a packed structure and send from various sources into the tracing buffer(EMEM) which keeps the temporal order of messages in binary format.

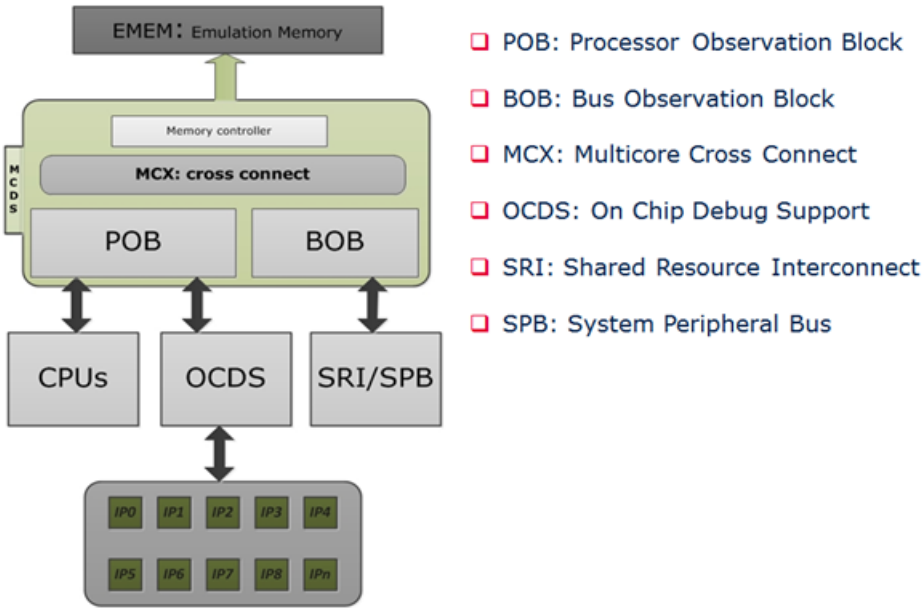


Figure 4.1: Architecture of Multicore Debug Solution

4.2 MCDS Features

- Infineon has successfully developed and deployed a technology known as the Multi-Core Debug Solution (MCDS) to address that problem.
- Using advanced on-chip trace techniques that include on-chip trigger generation, trace data compression, and trace storage, MCDS provides only the relevant trace data to the debug tool.
- Without adding pins to the chip, MCDS enables real-time, in-system debug and performance analysis practical by tracing the relevant information at the relevant time.
- Non-intrusive debugging.
- Compression of trace messages to save memory.
- Can be placed outside of production die.
- Independent of physical interfaces between chip and debug host.
- Core runs at full speed

- Access to internal buses
- No changes to the application code are required.
- The target does not need to be halted to gather data.
- The real-time performance of your application is not affected by data collection

4.3 Trace Memory

The term trace memory is found in many different places when tracing is in progress or has been done.

- Small, local (primary) FIFOs inside the Trace Units. Each observation block contains several trace units, which in turn may all generate a different message at the same time. Therefore, small local FIFOs are required inside the trace units.
- Width and depth depend on kind and size of messages characteristic for the containing unit.
- Local (secondary) FIFO's in the Observation Blocks.
- The building block is called MSU (message sequencer unit) and takes care that the primary FIFOs are emptied strictly sorted by time tag.

The buffer memory, which is available to store the trace data is 1MB in TC39x and 8KB in TC38x. This imposes a difficulty to gathertrace data for all test cases being executed in one shot.

4.3.1 Continuous Tracing with High Speed Tool Interface

The use case described can be considered as single shot measurement. The recorded time(trace depth) is limited by the amount of on-chip memory. For some applications like performance measurement the trace depth can be extended off-chip. Four preconditions must be met: At least two memory tiles can be

used for tracing. The data transfer channel to the tool has enough bandwidth. The average trace bandwidth is less than the tool bandwidth. The peak trace bandwidth does not cause overruns of the on-chip buffer memory.

Debugging an Infineon TriCore device requires a Lauterbach Debug Cable together with a Lauterbach Debug Module.The Debug Cable comes with a license for debugging.Lauterbach offers an off-chip trace solution for AURIX devices equipped with a serial off-chip trace port (Aurora Giga-Bit Trace, AGBT). For devices of the AUDO-NG family, a parallel off-chip trace is available. Lauterbach offers multicore debugging and tracing solutions (particularly for AURIX devices), which can be done in two different setups: Symmetric Multiprocessing (SMP) and Asymmetric Multiprocessing (AMP).AURIX devices can be traced with an On-chip Trace or a Serial Off-chip Trace (AGBT) if a connector is available on the target.

4.4 MCDS Clock

MCDS runs on BBB clock, which is read from CCU registers. We can change the clock of the IP(doesnt make any sense to it). If the target application runs on different frequency then u have to confirm the clock of BBB to make your calculation accurate.

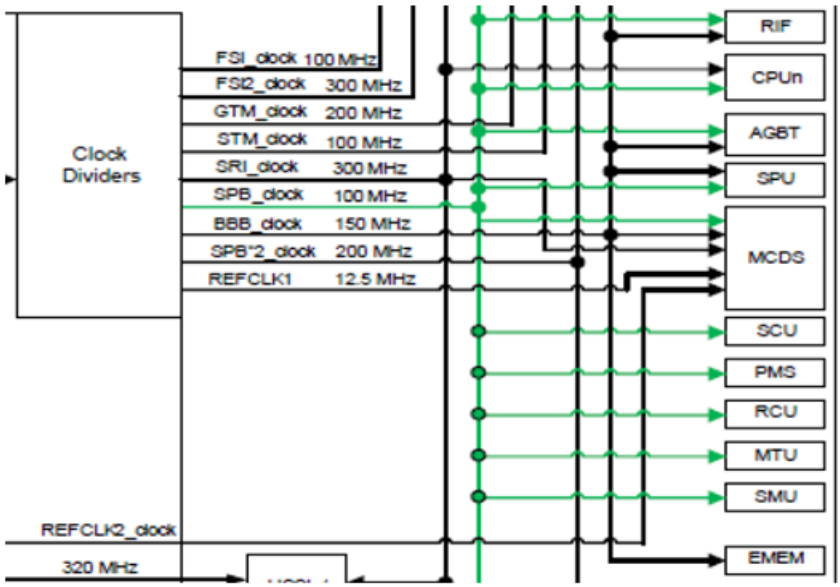


Figure 4.2: MCDS Clock

MTV sets the clock automatically, when the device is connected (or reset), according to core 0. Please note that in MTV between reset and clock system configuration the frequency can be different. To avoid that the clock systemis not yet configured, this automatic measurement will only bedone while CPU0 is running.MTV gives us the timings in form of ticks. Suppose the BBB clock is set to 150MHz. Then the execution time of 1 tick is 6.6ns.

4.5 Functional Description

4.5.1 Trigger logic

Trigger logic is depending on the wide range of comparators present inside MCDS.Each observation blocks has these different comparators for the triggering Logic.

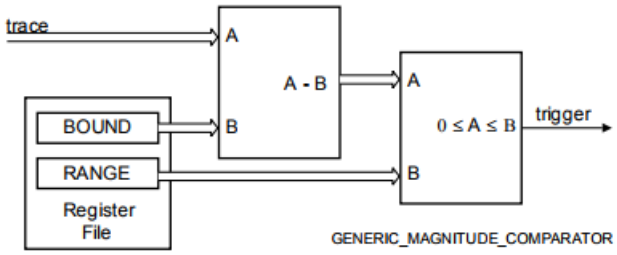


Figure 4.3: Trigger Logic

4.5.2 Event logic

Event logic will qualify the trigger pool. When consolidating trigger outcomes, two primary cases are conceivable:Either the event is given by various triggers which need to coordinate simultaneously (e.g. address in range AND data equivalent to numerical value) or something needs to happen if no less than one from an arrangement of triggers (e.g.address lower than base OR higher than top of stack) matches.

As the second case regularly requires ANDing triggers to figure out the components of the OR set, it was chosen to consign the OR function to the event's customers, specifically the Action Definitions.The number and sort of triggers associated with every event rely upon the area of its usage.

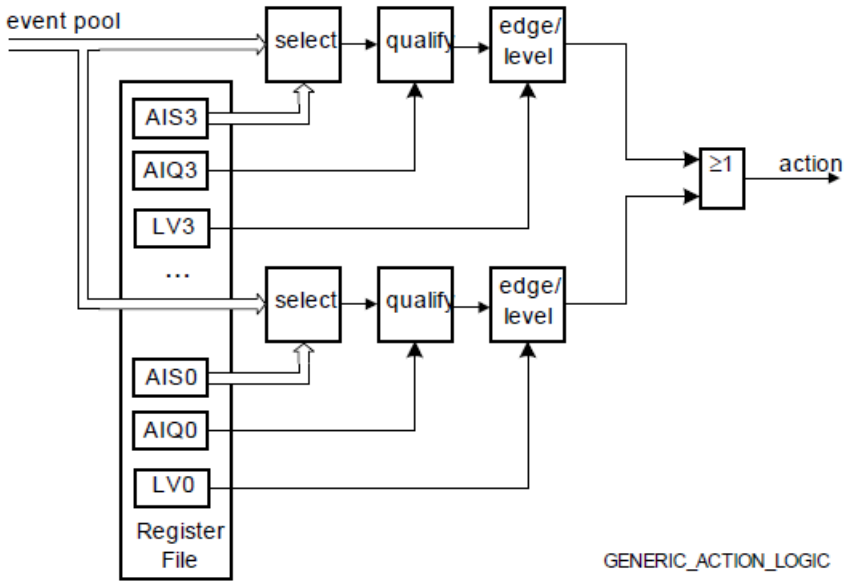


Figure 4.4: Trigger Logic

4.5.3 Action logic

Action logic activates the required trace capture.All the activities inside MCDS is regulated by standardized registers called activity definitions. Frequently, something needs to happen if no less than one from an arrangement of events (e.g. address lower than base OR higher than best of stack) happens.While a simple boolean OR seems to fit the bill, the distinction between edge and level requires a little more effort.

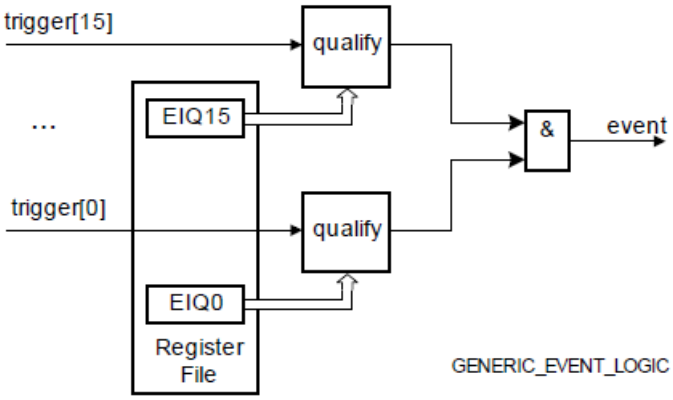


Figure 4.5: Trigger Logic

4.6 MCDS Tracing

Various kind of information can be collected by the MCDS. POB is able to trace core related information including program flow, data accesses and core states. For program flow tracing, three alternatives with different granularity are provided, which are instruction tracing, flow tracing and compact function tracing. The alternative is selected as a compromise of tracing duration and granularity.

4.6.1 Instruction tracing

The Instruction tracing records the ending time of each instruction execution. It provides very detailed information of the program execution but consumes the tracing buffer fast.

Index	TimeR	Opoint	Origin	Operation	Address	Symbol
1	1	CPU0	CPU0	IP	A001C2CC	core0_main
2	1	CPU0	CPU0	IP	A001C2D0	core0_main
3	3	CPU0	CPU0	IP	A001C2D4	core0_main
4	4	CPU0	CPU0	IP	A001C2D8	core0_main
5	6	CPU0	CPU0	IP	A001C2DC	core0_main
6	8	CPU0	CPU0	IP	A001C2DE	core0_main
7	9	CPU0	CPU0	IP	A001C2E2	core0_main
8	11	CPU0	CPU0	IP	A001C2E6	core0_main
9	12	CPU0	CPU0	IP	A001C2EA	core0_main
10	14	CPU0	CPU0	IP	A001C2EE	core0_main
11	16	CPU0	CPU0	IP CALL	A001C2F2	core0_main
12	25	CPU0	CPU0	IP	A00022C4	Ifx_OSTask_ApplicationInit
13	26	CPU0	CPU0	IP	A00022C6	Ifx_OSTask_ApplicationInit
14	40	CPU0	CPU0	IP	A001F2E0	EE_oo_StartOS
15	41	CPU0	CPU0	IP	A001F2E4	EE_oo_StartOS
16	41	CPU0	CPU0	IP	A001F2E8	EE_oo_StartOS
17	41	CPU0	CPU0	IP	A001F2EC	EE_oo_StartOS

Figure 4.6: Instruction Tracing

4.6.2 Flow tracing

Flow tracing offers a balance between the tracing duration and the tracing granularity. It does not trace detailed instruction execution but the discontinuity of the program flow, which are caused by function calls, branches and interrupts. The continuous part is by default considered as that the PC is continuously incremented. In this way, the ending time of each instruction is not available but it saves the tracing bandwidth.

4.6.3 Function tracing

The least bandwidth-requiring alternative is compact Function tracing, which only generates messages when function calls and function returns are observed.
This alternative consumes little bandwidth but it also omits detailed execution information. For instance, the detailed branch execution and even leaf functions are not contained in compact function tracing.

Index	TimeR	Opoint	Origin	Operation	Address	Symbol
1	17	CPU0	CPU0	IP CALL	A001C2F2	core0_main
2	17	CPU0	CPU0	IP CALL	A00022C4	Ifx_OSTask_Application
3	77	CPU0	CPU0	IP CALL	A001F36E	EE_oo_StartOS
4	77	CPU0	CPU0	IP CALL	A0001C50	StartupHook

Figure 4.7: Function tracing

The examples above are decoded by TraceViewer,which is an MCDS configuration and trace decoding tool. The .elf file is also given to provide the detailed binary and symbolic information. Timer shows the time stamp information. The time stamp is added after a message is generated. For instruction tracing, a message is generated when the instruction has been executed. While, for flow tracing, the generation of a message is triggered when a discontinuity happens. Accordingly, a function call or return creates a compact function tracing message.

4.6.4 Data tracing

The Data Trace gives us all the read/write operations being done. It provides the value been registered at specific address. We can even configure the IP in a range qualification to get the read/write operation traces.

4.6.5 Bus tracing

The bus trace gives all the operations being performed by the SPB/SRI.

Index	TimeR	Opoint	Origin	Data	Operation	Address	Symbol
1	19889	SPB	CPU0.DMI	FFFC000E	R32 SV	F0036100	.SCU
2	19911	SPB	CPU0.DMI	FFFC000E	R32 SV	F0036100	.SCU
3	19916	SPB	CPU0.DMI	FFFC000E	R32 SV	F0036100	.SCU
4	19934	SPB	CPU0.DMI	FFFC00F1	W32 SV	F0036100	.SCU
5	19938	SPB	CPU0.DMI	FFFC00F2	W32 SV	F0036100	.SCU

Figure 4.8: Bus tracing

4.6.6 OLDA trace

The Online Data Acquisition (OLDA) is an address space where writes can complete without error but no memory is addressed. This allows production code to be writtern which writes data to memory, where the memory is only present in Emulation Device. When running on an Emulation Device, the user can map EMEM to the OLDA region address space using the memory overlay feature, and the write data is then stored in the EMEM. When running in the Production Device, a default slave(when OLDA is enabled) terminates writes to the OLDA address space without error, even though no memory exists at the target address, and the write data is discarded.

In the event that OLDA support is empowered in a default slave, direct write accesses to (without redirection) the OLDA run are not so much execute and they do not create a bus error trap.

Redirection of accesses to real memory is accomplished internally to the processor generating the access. Redirected accesses will therefore not have an address lying within the OLDA address range when they are placed on the SRI.

4.7 Aurix Tools

The implementation of the proposed methodology is developed on the basis of several existing tools. DAS and MTV are the most important ones. DAS deals with the connection to devices while MTV is focused on MCDS tracing and decoding.

4.7.1 DAS

The DAS architecture was designed for multi-device multi-core systems with very demanding emulation requirements. It ensures a single interface to different equipments including system and software debugging, silicon debugging, silicon validation and tool chain debugging. The same tooling interface supports various device representations from ESL model to end product to reduce cost and risk.

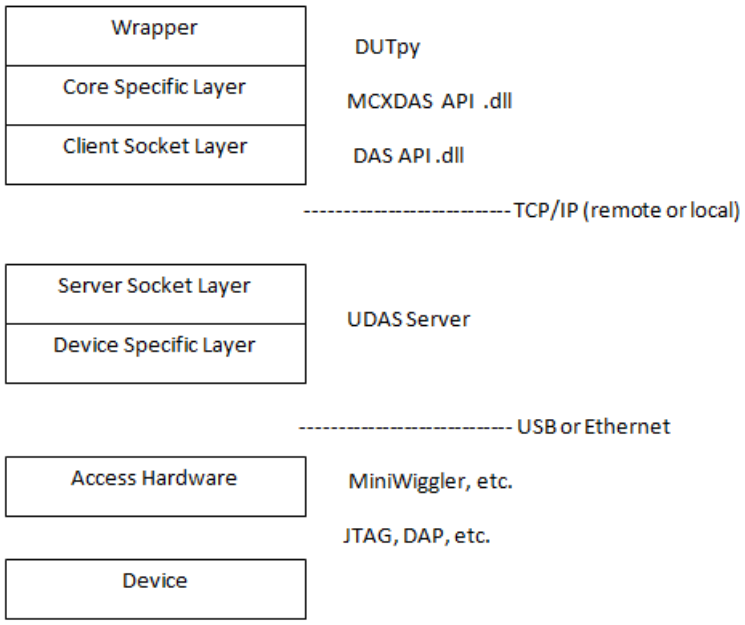


Figure 4.9: Block Diagram of DAS

The detailed structure of DAS is shown in the block diagram Figure 4.7. The tool interface is on software level (DAS Application Programming Interface (API)), which is implemented in a .dll file. This layer provides the abstraction of the device connection. In this way, the connection will be transparent for the tool. On top of this layer, MCD API is created to better fulfill the requirement mentioned in [5]. The connection between computer and access hardware is via either USB or Ethernet. The access hardware here means miniWiggler, which is a converter from the USB to the debugging interface for example DAP/SPD/SWD/JTAG.
For development boards provided by Infineon, an onboard miniWiggler is already integrated on the PCB so that the board can be connected directly from the computer via USB.

4.7.2 MCDS TraceViewer (MTV)

Behind the graphical user interface of MTV there are two main building blocks:

- The Trace Decoder
- The MCDS Configuration Control

The trace decoder is very generic and decodes any trace generated with any MCDS configuration. The MTV user controlled MCDS configuration is restricted to a quite static MCDS resource usage. This allows extracting the user level representation of the MCDS configuration just from the MCDS register settings. There is no restriction for MCDS configurations provided with .mcdsc files, but please note that arbitrary configuration changes with the MTV user interface are note supported. We can say, as if it provides 10% of configurations options as that of MCDS IP. MTV is a part of the DAS installation

MTV file formats

MTV supports the following file formats:

- .mcds for trace data and configuration exchange
- .mcdsc for configuration only exchange
- Standard ELF files for code, data and symbols

.mcds data file format

The .mcds file contains all the information needed to decode a trace, in a very compact format.

Features

- Compact
- Contains all the information which is needed for a complete trace decoding
- Traced data
- MCDS version and configuration
- Binary code snippets, needed for program flow decoding (optional)

- Data and Code symbols (optional)
- Supports a comparison with a text diff tool
- Can be used as a configuration file to produce traces.

The .mcds file format consists of several sections. The sequence of sections has to be as listed as shown in figure.

Section	Marker	Note
Header	None	Version and device information
MCDS Config	START_OF_MCDS_CONFIG	MCDS configuration
Symbols	START_OF_SYMBOLS_IN_TRACE	Symbols in trace, imported from ELF file (optional)
Code	START_OF_CODE_IN_TRACE	Code in trace, read from device (optional)
Trace Data	START_OF_TDR	Trace Data Raw (TDR)

Figure 4.10: .mcds File Format

File header

The file header contains version and device information:

McdsFileFormatVersion=1.0

McdsDevice=AURIX_ED

McdsDeviceUsed=0x201E9083

McdsToolUsed="MTV V1.1"

McdsFrequencyMHz=150.0

RecordingDate=2016-06-02

RecordingTime=14:19:59

McrcmVersion=1.1.0

McdsDevice := AURIX_ED — A2G_ED — AURIX_MINIMCDS — A2G_MINIMCDS

McrcmVersion reflects the version of the mapping of user features to MCDS register settings.

Section MCDS config

This section lists all MCDS registers which do not have their reset value.

- The addresses are from the CPU point of view.

- The redundant name of the register is for information only and can be omitted.

- The redundant name of the MCDS registers is for information only and can be omitted. A tool reading a .mcds file will usually just ignore the names and not check the consistency of register addresses and names.

START_OF_MCDS_CONFIG
F90E8014 00018E84 MCDS_MUX
F90E8204 0000E000 MCDS_FIFOBOT
F90E820C 00011FFF MCDS_FIFOTOP ...

Usually only the MCDS_MUX register value is required for full trace decoding. Exceptions are Function (CFT) traces which need to be indicated to the decoder by MCDS register settings.
The full register information can be used to reproduce the same trace situation with the device.

Section symbols

This section lists code and data symbols which were used in the trace and found in the loaded ELF files. Note: Code Symbol format is CS@¡address¿¡size¿¡name¿ Data Symbols start with DS@.
START_OF_SYMBOLS_IN_TRACE
CS@800009DA 5C FuncIFX_OSTASK_EVENT1
CS@80000AEE 12 FuncIFX_OSTASK_BACKGROUND
CS@8000196C 26 IfxPort_setPinMode
CS@80001BFE B4 IfxScuCcu_getPllFrequency
...
DS@600007D4 4 EE_IRQ_nesting_level
DS@600007D8 4 EE_tc_active_utid
DS@70019600 500 __USTACK0
DS@800004E0 2C EE_th_dispatch_prio

Section code

This section contains all code which is needed for the trace decoding. A text format was chosen to enable a file comparison with a text diff tool.

- Each line represents 32 address aligned bytes.
- The format is CS@¡address¿ followed by the data starting with byte 31 and ending with byte 0, structured in 4 byte words.

START_OF_CODE_IN_TRACE
31 28 27 24 23 20 19 16 15 12 11 8 ... 0
C@A0000860 F054FF59 1FC2F054 FF19F600 0091AE40 80000240
000D18ED ... C@A0000880 2041F24B FCCCDF1B F3D4D07B 00342319
0000F2C5 122A006D ... C@A0000940 0000FFC5 2020AE40 80000240
000D1BD9 006D0004 F25FF280 ... ...

Section trace data

This section contains the trace data. This can be linear trace of any size or the image of a circular buffer.

START_OF_TDR
<binary data>

The decoding starts with the first byte until no trace data is available or an MCDS <endoftrace>message is found. In case of an <endoftrace>message it is automatically checked, whether this is the image of a circular buffer. The precondition is that the size of the trace data is a multiple of an MCDS paragraph. If the <endoftrace>message is not found in the highest paragraph, the decoder starts a second circular pass from the beginning of the following paragraph. If this circular pass ends at the same <endoftrace>message, it is considered as the image of a circular buffer and the second pass is displayed, otherwise the first pass.

.mcdsc configuration file format

The .mcdsc configuration file contains the complete sequence of register accesses required for setting up and starting the trace recording, including OCDS enabling

and EMEM configuration. The addresses are from CPU point of view. Note: The redundant name of the MCDS registers is for information only and can be omitted. A tool reading a .mcdsc file will usually just ignore the names and not check the consistency of register addresses and names.

McdsConfigFileFormatVersion=1.0

McdsDevice=AURIX_ED

McdsDeviceUsed=0x201E9083

McdsToolUsed="MTV V1.1"

RecordingDate=2016-06-02

RecordingTime=14:19:59

McrcmVersion=1.1.0

F0000478 000000A1 CBS_OEC

F0000478 0000005E CBS_OEC

F0000478 000000A1 CBS_OEC

F0000478 0000005E CBS_OEC

F000047C 00000300 CBS_ONTRL

F90E6000 00000100 EMEM_CLC

F90E6034 00000002 EMEM_SBRCTR

...

F90E8000 00000000 MCDS_CLC

F90E8018 AAAAAAAA MCDS_SESSIDL

...

Message Format

traced data: This is the information itself. Examples are the current value of the instruction pointer or the number of a watchpoint. trace type:It is obvious that the traced binary data is impossible to interpret unless the type of information is given. core ID:As soon as the message is taken from the primary FIFO, the source has to be attached to the message to facilitate reconstruction. An alternative would be to personalize the trace type, but that would spoil the reuse concept. time tag: At some time the messages are stored in FIFOs. To maintain the chronological order

when they are taken from different FIFOs, a tag is attached. Same tag means same instant of creation. time stamp: When sorting messages into the trace buffer, care is taken to put time stamp messages first. All subsequent messages are then not older than the most recent time stamp.

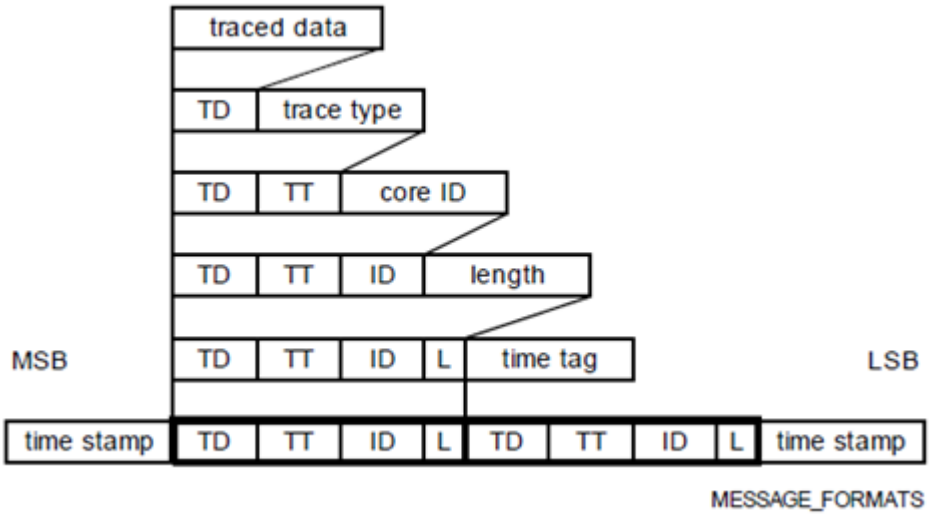


Figure 4.11: Message Format

Snapshot of MTV TraceViewer

File menu:

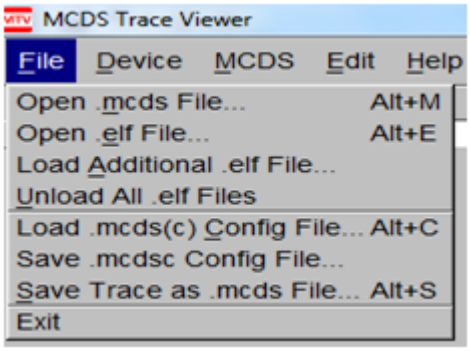


Figure 4.12: File Option

Device menu:

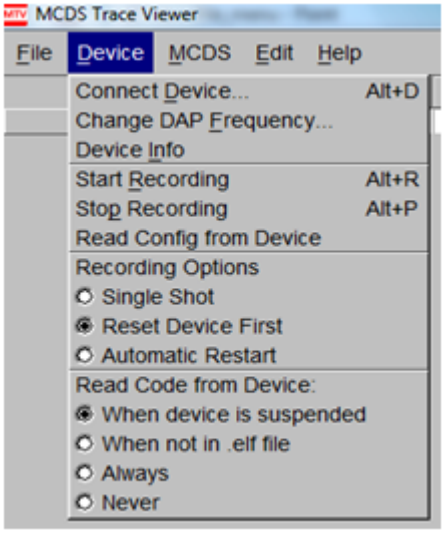


Figure 4.13: Device Option

MCDS menu:

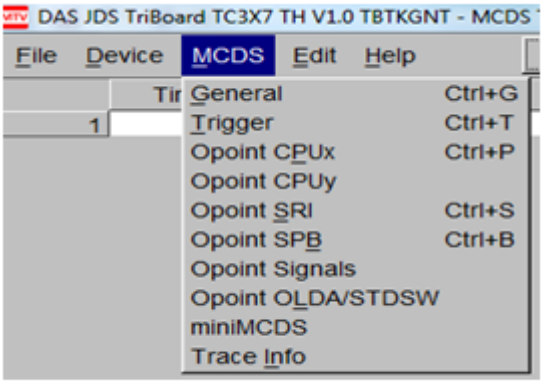


Figure 4.14: MCDS Option

MCDS can trace 2 CPUs at a time, Therefore, we can configure two CPUs differently in form of OpointCPUx and OpointCPUy.

MTV Default Settings:

Opoint CPU0

- Status Trace
- Function Trace
- OLDA Trace

Opoint SPB

- SPB Error Trigger

Opoint SRI

- SRI Error Trigger

Trigger

- Application Reset Trigger
- OCDS Suspend Trigger

General

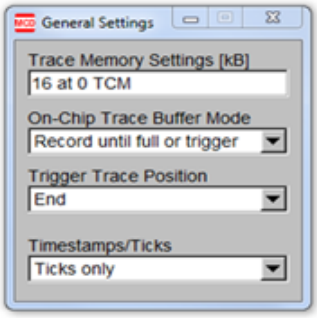


Figure 4.15: MTV Default Setting

Trigger Trace on IP address:

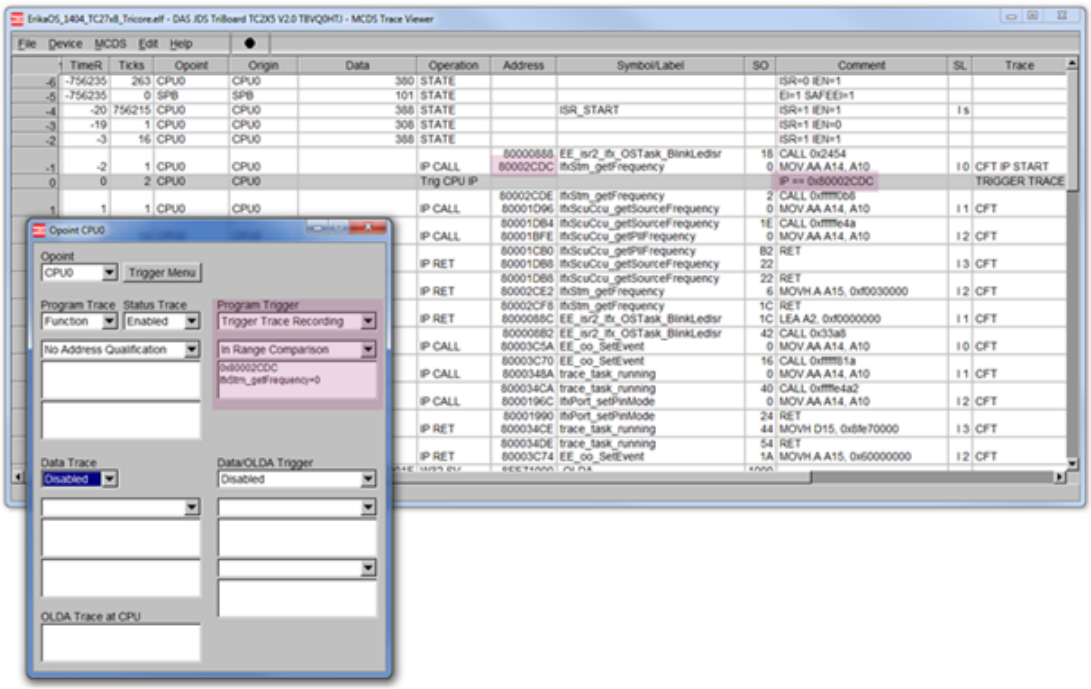


Figure 4.16: Trigger Trace on IP address

Program Flow and Data Trace:

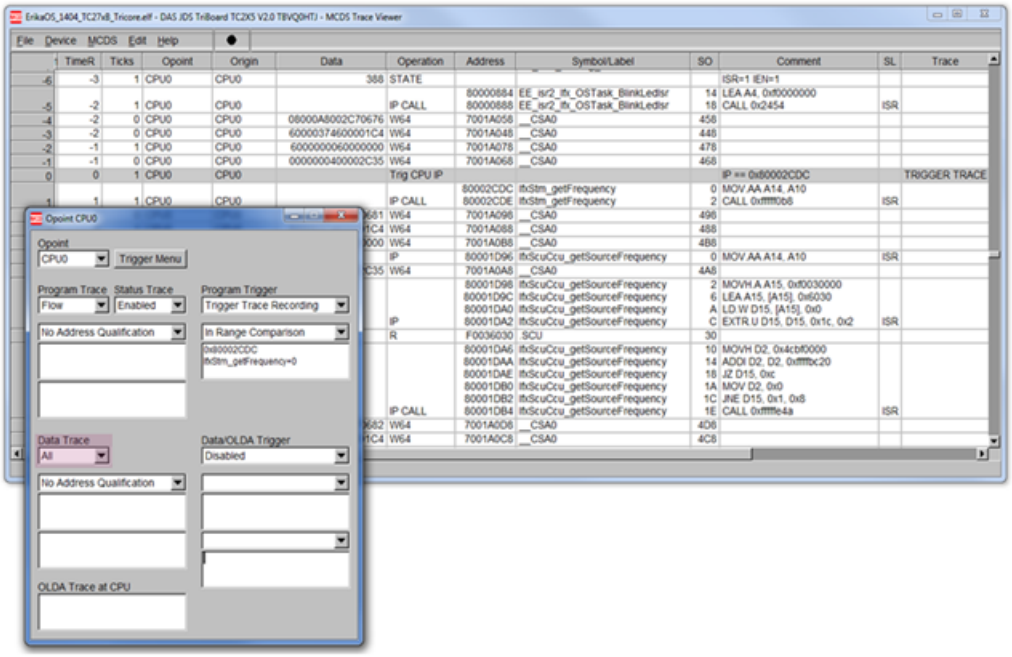


Figure 4.17: Program Flow and Data Trace

Qualified Flow and Data Trace:

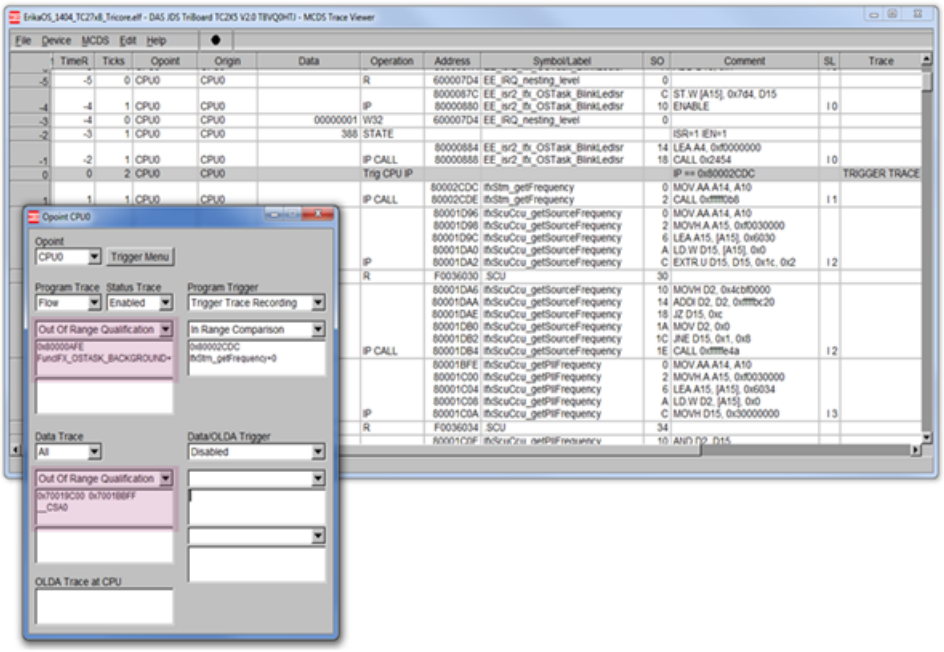


Figure 4.18: Qualified Flow and Data Trace

Recording:

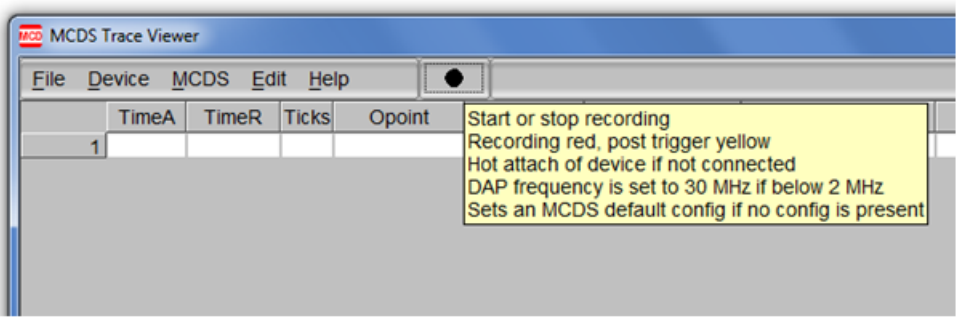


Figure 4.19: Recording

Chapter 5

Implementation and Experimental Evaluation

5.1 Execution Flow

Figure 5.1 demonstrates the Flow of Execution of the MCAL Driver module. Code flashing is finished by Infineon Memtool. Regression Execution is done by Universal Validation Platform (UVP).

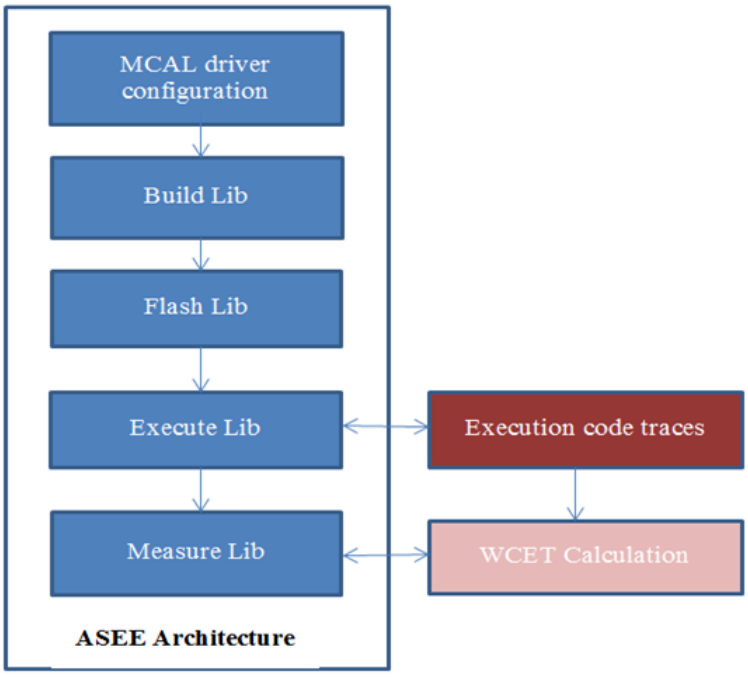


Figure 5.1: Execution Flow

5.2 Methodology

- Data Trace for Global registers (SFR)

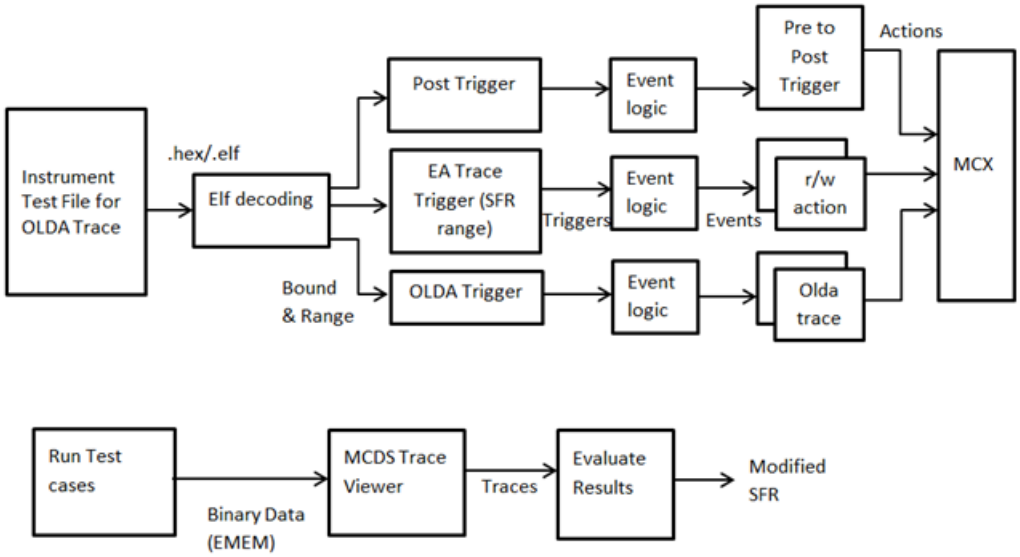


Figure 5.2: Data Trace for Global registers

- Performance Test: Maximum execution time of MCAL Driver APIs.

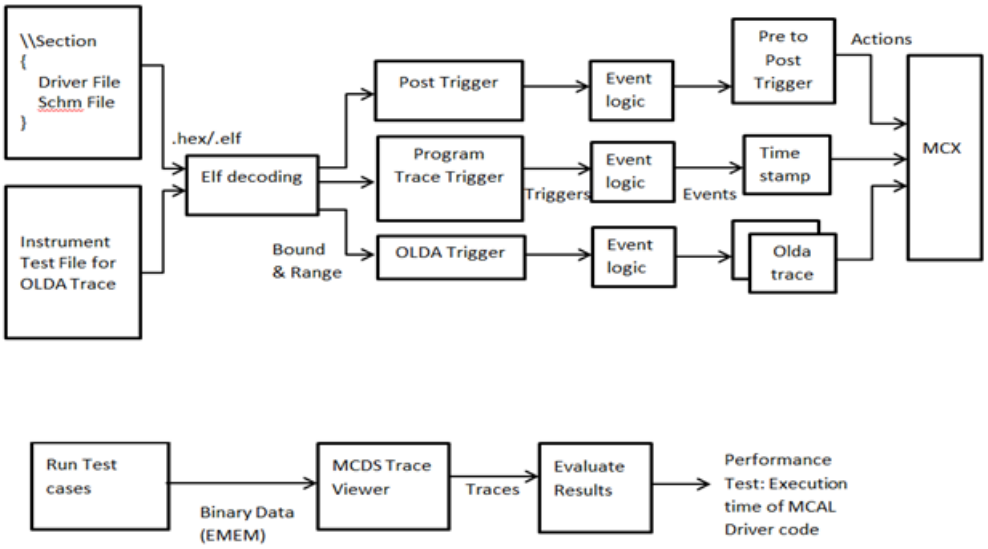


Figure 5.3: Performance Test for MCAL Drivers

5.3 Design of the Library (ExecutionTraceLib)

As the MTV GUI provides limited configuration options of MCDS, we perform various triggers and qualifications, configurations directly on MCDS registers.

- With the help of DUTpy Library (built on DAS), we can configure the MCDS registers using advanced on-chip trace techniques that include on-chip trigger generation mapped with different types of Traces like Instruction, Function, Flow, Data.
- The Binary trace generated in Buffer Memory is decoded to get to get the Execution Test Traces.
- And thereafter this decoded trace information can be mapped with symbol table and parsed to get the min/max time of an Driver API.

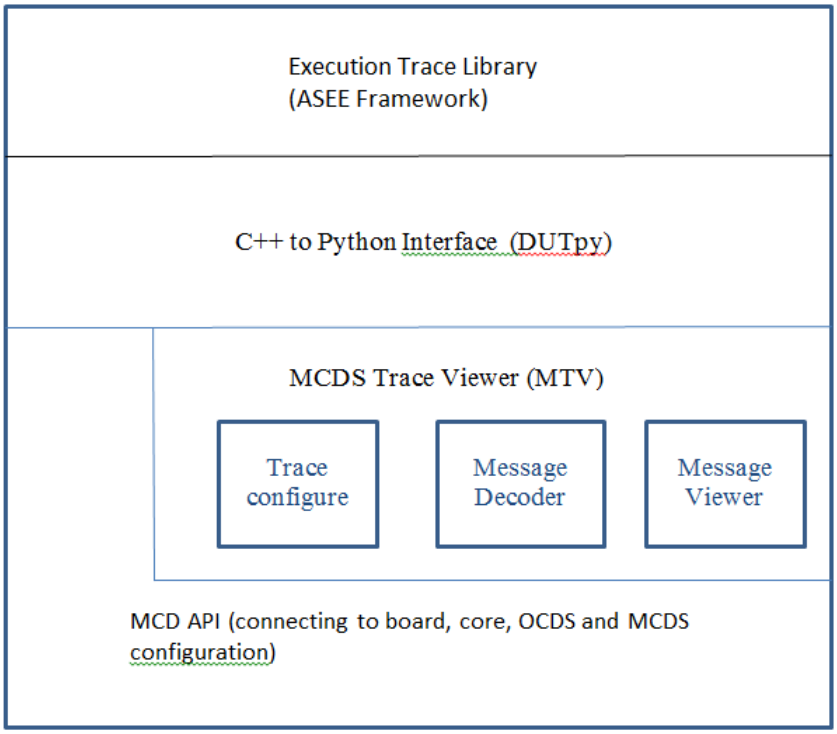


Figure 5.4: Integration of ExecutionTraceLib with ASEE Framework

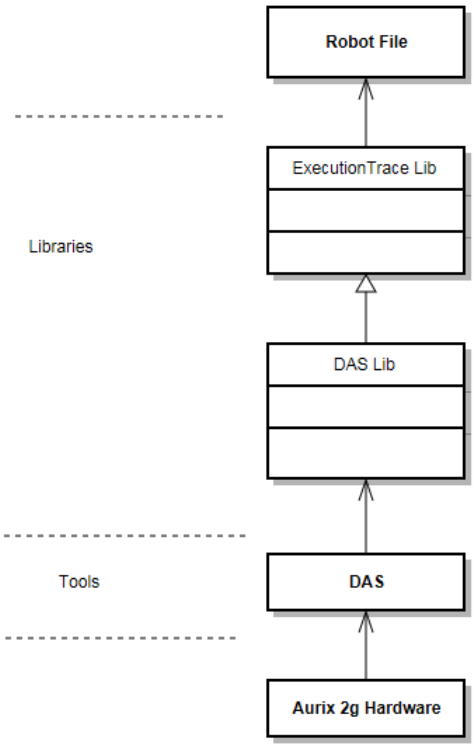


Figure 5.5: Architecture of ExecutionTraceLib

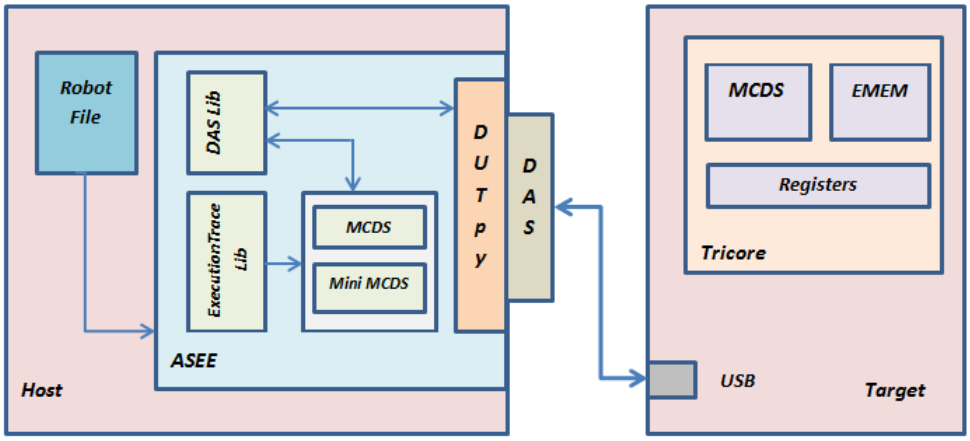


Figure 5.6: Block Diagram of Execution Trace Library

5.4 Execution of the Library

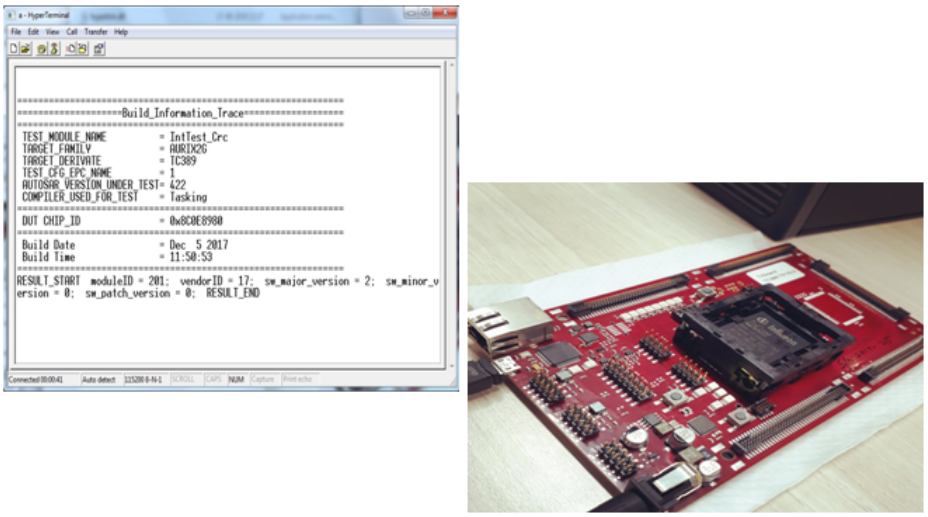


Figure 5.7: Execution Setup

```
DUTpy.py | IfxMcds_reg.py | __init__.py | ExecutionTraceLib.py | ase_env_cfg.xml | IntTest_Fls_17_Dmu (2).robot

 5  Library                 BuildLib
 6  Library                 UVPLib
 7  Library                 FlashLib
 8  Library                 ParamGenLib
 9  Library                 ExecutionTraceLib
10  Library                 ExecutionLib
11  #Library                 SaleaeLibOpt
12  Library                 DasLib
13
14  Test Setup              Suite Init
15  Test Teardown           Suite Delete
16
17  *** Variables ***
18
19  *** Keywords ***
20  Suite Init
21      [Documentation]     Suite init sequence
22      Setup Suite
23      Init Build Lib
24      Init Flash Lib
25      UVPLib.Init Execution Lib
26      Init Param Gen Lib
27      Init Execution Trace Lib
28
29  Suite Delete
30      [Documentation]     Suite delete sequence
31      Log to Console      TearDown
32
33  *** Test Cases ***
34  TestCase001
35      [Documentation]     These are UVP Testcases
36      [Tags]              Regression
37      Init Build Lib
38      ${bin_file} =   Start Build
39      Init Flash Lib
40      Program Flash     ${bin_file}
41      Init Param Gen Lib
42      Read STS
43      #Gen UVP
44      #Start Function Trace
45      ParamGenLib.Start UVP
46      #Stop Trace
47      Log to Console      UVP
48
49  TestCase002
50      [Documentation]     These are UVP Testcases
```

Figure 5.8: Robot File snapshot

Figure 5.9: Ase_env Configuration File snapshot

```
class ExecutionTraceLib(object):
    """
    ExecutionTraceLib help us to get the Execution Test Traces. It is a
    method of tracing the CPUs execution, Bus, or any IP's functionality
    during runtime without halting or stopping/ disturbing the target
    execution.

    It provides a way to interact with the target in order to get the minimum
    and maximum time of the Driver API being executed. This information is
    obtained by recording the execution traces and thereafter calculating the
    min/max timings required by the Driver API.In order to use the same
    from ASEE,
    a ``ExecutionTraceLib``element should be available in ASEE configuration.\n

    ExecutionTraceLib will use ``TRACE_TOOL`` to perform all trace operations.
    Currently, ExecutionTraceLib provides support of Mcds and Minimcds as
    Trace tool.\n

    ASEE provides a way to redefine ExecutionTraceLib configuration at Suite
    level. At the time of init, ExecutionTraceLib object will be iniatialised
    with the latest configuration.

    Mcds:
    - In Eumlation Devices (ED) MCDS tool is available

    Minimcds:
    - In ProductionDevices (PD) MiniMCDS tool is available.

    Mandatory items in ExecutionTraceLib configuration:
    | =Element=              | =Description=                          |
    | ``TRACE_TOOL``         | Trace tool to be used                  |
    | ``START_TRACE_FUNCTION``| Start Function to start the Trace     |

    This Library will work with DUTpy 64 bit.
    """
    ROBOT_LIBRARY_SCOPE = 'TEST SUITE'
    _executionTracelib_cfg = {}
    elf_file = None

    def __init__(self):
        """ Constructor. """
        logging.info("ExecutionTraceLib: Starting...")
```

Figure 5.10: ExecutionTrace Lib snapshot

```
"""This module provides interface to DUTpy"""

__author__      = "ASE^2 Core Developers"
__copyright__   = "Copyright (C) Infineon Technologies (2017)"
__version__     = "0.1.0"
__all__         = ['DasLib']


import logging
from das import DUTpy


class DasLib(object):
    """Interface to use 32bit DUTpy available only with 32bit ASEE"""

    def __init__(self):
        """DasLib"""
        logging.info("DasLib: Starting DasLib from ASEE")

    def get_keyword_names(self):
        """Get all keywords from DasLib"""
        attr_ls = dir(DUTpy)
        fn_ls = []
        for attr in attr_ls:
            if attr.startswith("_"):
                continue
            if callable(getattr(DUTpy, attr)):
                fn_ls.append(attr)
        return fn_ls

    def run_keyword(self, name, *args):
        """Run Keyword from DasLib"""
        _fn = None
        if hasattr(DUTpy, name):
            _fn = getattr(DUTpy, name)
            args = (eval(a) for a in args)
            #print("Evaluated args are:", args)
            return _fn(*args)
```

Figure 5.11: DAS Lib snapshot

```
from sys import version_info as _swig_python_version_info
if _swig_python_version_info >= (2, 7, 0):
    def swig_import_helper():
        import importlib
        pkg = __name__.rpartition('.')[0]
        mname = '.'.join((pkg, '_DUTpy')).lstrip('.')
        try:
            return importlib.import_module(mname)
        except ImportError:
            return importlib.import_module('_DUTpy')
    _DUTpy = swig_import_helper()
    del swig_import_helper
elif _swig_python_version_info >= (2, 6, 0):
    def swig_import_helper():
        from os.path import dirname
        import imp
        fp = None
        try:
            fp, pathname, description = imp.find_module('_DUTpy', [dirname(__file__)])
        except ImportError:
            import _DUTpy
            return _DUTpy
        try:
            _mod = imp.load_module('_DUTpy', fp, pathname, description)
        finally:
            if fp is not None:
                fp.close()
        return _mod
    _DUTpy = swig_import_helper()
    del swig_import_helper
else:
    import _DUTpy
del _swig_python_version_info

try:
    _swig_property = property
except NameError:
    pass  # Python < 2.2 doesn't have 'property'.

try:
    import builtins as __builtin__
except ImportError:
    import __builtin__
```

Figure 5.12: DutPy.py snapshot

```
2584  # \brief 8C1C, Comparator Sign Register ${x} #/
2585  MCDS_TCZIDSGN1 =0xFA018C1C
2586
2587  # \brief 9000, Comparator Bound Register ${x} #/
2588  MCDS_TCZIPBND0 =0xFA019000
2589
2590  # \brief 9004, Comparator Range Register ${x} #/
2591  MCDS_TCZIPRNG0 =0xFA019004
2592
2593  # \brief 9010, Comparator Bound Register ${x} #/
2594  MCDS_TCZIPBND1 =0xFA019010
2595
2596  # \brief 9014, Comparator Range Register ${x} #/
2597  MCDS_TCZIPRNG1 =0xFA019014
2598
2599  # \brief 9020, Comparator Bound Register ${x} #/
2600  MCDS_TCZIPBND2 =0xFA019020
2601
2602  # \brief 9024, Comparator Range Register ${x} #/
2603  MCDS_TCZIPRNG2 =0xFA019024
2604
2605  # \brief 9030, Comparator Bound Register ${x} #/
2606  MCDS_TCZIPBND3 =0xFA019030
2607
2608  # \brief 9034, Comparator Range Register ${x} #/
2609  MCDS_TCZIPRNG3 =0xFA019034
2610
2611  # \brief 9040, Comparator Bound Register ${x} #/
2612  MCDS_TCZIPBND4 =0xFA019040
2613
2614  # \brief 9044, Comparator Range Register ${x} #/
2615  MCDS_TCZIPRNG4 =0xFA019044
2616
2617  # \brief 9050, Comparator Bound Register ${x} #/
2618  MCDS_TCZIPBND5 =0xFA019050
2619
2620  # \brief 9054, Comparator Range Register ${x} #/
2621  MCDS_TCZIPRNG5 =0xFA019054
```

Figure 5.13: IFxMcds.py snapshot

```
class Mcds(object):
    """ Execution Traces for MCDS tool."""

    mcds_cfg = {}
    ROBOT_LIBRARY_SCOPE = 'TEST SUITE'

    def __init__(self, cfg_dict=None):
        if cfg_dict:
            self.mcds_cfg.update(cfg_dict)
        self.DasLib_obj = DasLib()
        self.result_path = self.mcds_cfg['RESULT_PATH']
        self.trace_function = self.mcds_cfg['START_TRACE_FUNCTION']

    def trace(func):
        """ Connect the Board through DAS, Configures Mcds and
        Starts Trace according to the action. """
        def start_trace(self,elf_file=None):
            logging.debug(elf_file)
            symbol_table_file_path = self.decode_elf(elf_file)
            post_trigger_start_address = self.find_symbol_address(symbol_table_file_path,
                                                                  self.trace_function)
            self.trace_section_start_address = self.find_symbol_address(symbol_table_file_path,
                                                                        '__START_TRACE_CODE__')
            self.trace_section_end_address = self.find_symbol_address(symbol_table_file_path,
                                                                      '__END_TRACE_CODE__')

            self.DasLib_obj.run_keyword("servers")
            self.DasLib_obj.run_keyword("connect", "0", "0")
            self.DasLib_obj.run_keyword("rsthlt")
            self.DasLib_obj.run_keyword("ocdsinit")
            self.DasLib_obj.run_keyword("mcdsinit")
            self.DasLib_obj.run_keyword("ememinit")

            self.DasLib_obj.run_keyword("configTraceBuffer", "0x0",
                                        "0x001FFFFF", "0x10000")
            self.DasLib_obj.run_keyword("write32", str(IfxMcds_reg.MCDS_MUX),
                                        "0x81")
            self.DasLib_obj.run_keyword("write32", str(IfxMcds_reg.MCDS_TCXIPBND0),
                                        str(post_trigger_start_address))  # Post-trigger
            self.DasLib_obj.run_keyword("write32", str(IfxMcds_reg.MCDS_TCXIPRNG0),
                                        "0x1")
            self.DasLib_obj.run_keyword("write32", str(IfxMcds_reg.MCDS_TCXEVT0),
                                        "0xFFFFEFF")
```

Figure 5.14: MCDS.py snapshot I)

```
207     def stop_trace(self, suite_name):
208         """Stops the Tracing and open the Execution Traces in MTV tool. """
209         self.DasLib_obj.run_keyword("trace_stop")
210         self.DasLib_obj.run_keyword("save_trace", "0xb9000000", "0xb91FFFFF")
211         self.DasLib_obj.run_keyword("disconnect")
212         logging.info('Trace Stopped.')
213 
214         mcds_file_name = suite_name + "_" + \
215             "_" + time.strftime("%Y%m%d-%H%M%S") + ".mcds"
216 
217         os.rename('test.mcds', mcds_file_name)
218         move(mcds_file_name, self.result_path)
219         result_file = self.result_path + "\\" + mcds_file_name
220         logging.info("Trace File Path: " + result_file)
221 
222         mtv_tool_path = self.mcds_cfg['TOOLS_BIN']['#text'] + \
223             "\\mcds_trace_viewer.exe"
224         cmd = [mtv_tool_path, result_file]
225         ret = subprocess.check_output(cmd, timeout=300, stderr=subprocess.PIPE)
226 
227     def decode_elf(self, elf_file=None):
228         """ Get the start address and size of the API from elf file. """
229         symbol_table_file_path = self.result_path + '\\symbols.txt'
230         symbol_table_file = open(symbol_table_file_path, 'w+')
231         subprocess.Popen("nm -S " + elf_file, stdout=symbol_table_file)
232         symbol_table_file.close()
233         time.sleep(2)
234         return symbol_table_file_path
235 
236     def find_symbol_address(self, symbol_table_file_name, symbol):
237         """ Find address of given symbol from given symbol table file. """
238         symbol_table_file = open(symbol_table_file_name, 'r+')
239         symbol_address = None
240         for line in symbol_table_file:
241             if symbol in line:
242                 line = line.split(" ")
243                 symbol_address = int(line[0], 16)
244 
245         symbol_table_file.close()
246         return symbol_address
```

Figure 5.15: MCDS.py snapshot II)

```
1  @ECHO ON
2  
3  set PYTHON_PATH=%~dp0\tools\python
4  set PATH=%PYTHON_PATH%;%PATH%
5  set script_path=%~dp0\Tools\ase_lib
6  
7  set PYTHONPATH=C:\git-repos\aurix2g_val_framework\ase_env\Tools\ase_lib
8  
9  python -c "from ExecutionTraceLib import ExecutionTraceLib; Executiontrace1=ExecutionTraceLib(); Executiontrace1.init_execution_trace_lib();
   Executiontrace1.stop_trace(); Executiontrace1.start_execution_trace()"
10 REM call C:/git-repos/aurix2g_val_framework/ase_env/tools/issue_reset/IssueReset.exe LocalHost UDAS 0
11 call set >> abc.txt
12 
```

Figure 5.16: Compiler.bat file snapshot

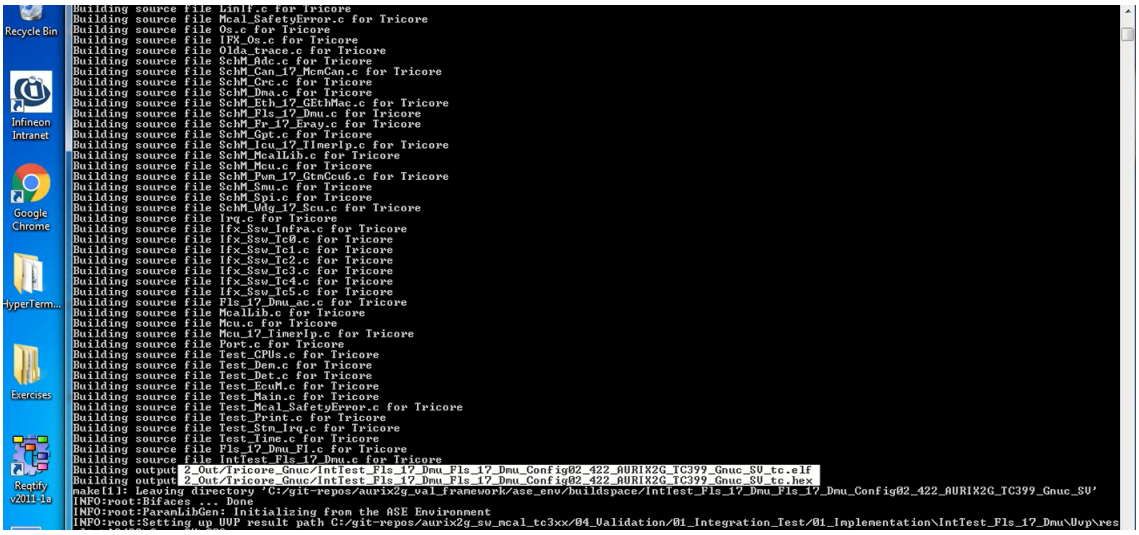


Figure 5.17: Building Modules snapshot

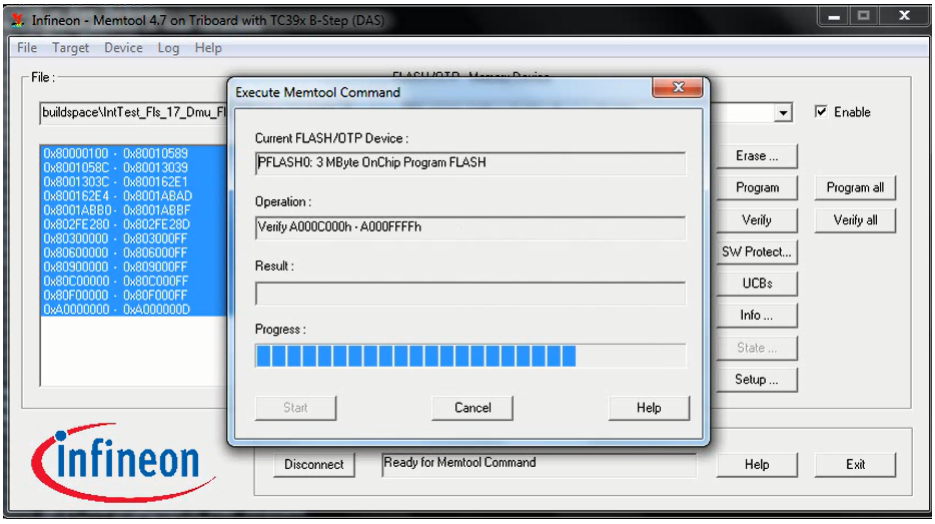


Figure 5.18: Flashing the .hex files to controller snapshot

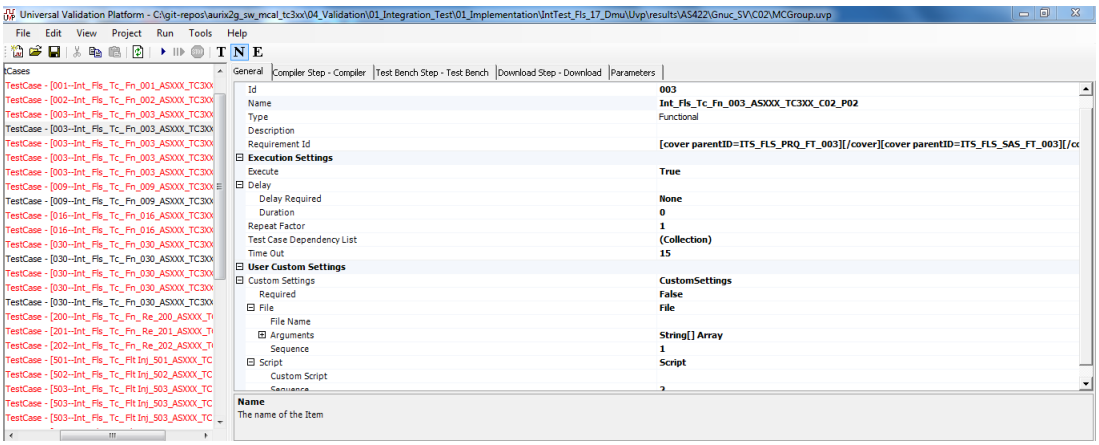


Figure 5.19: Universal Validation Platform (Executing Test cases Framework)

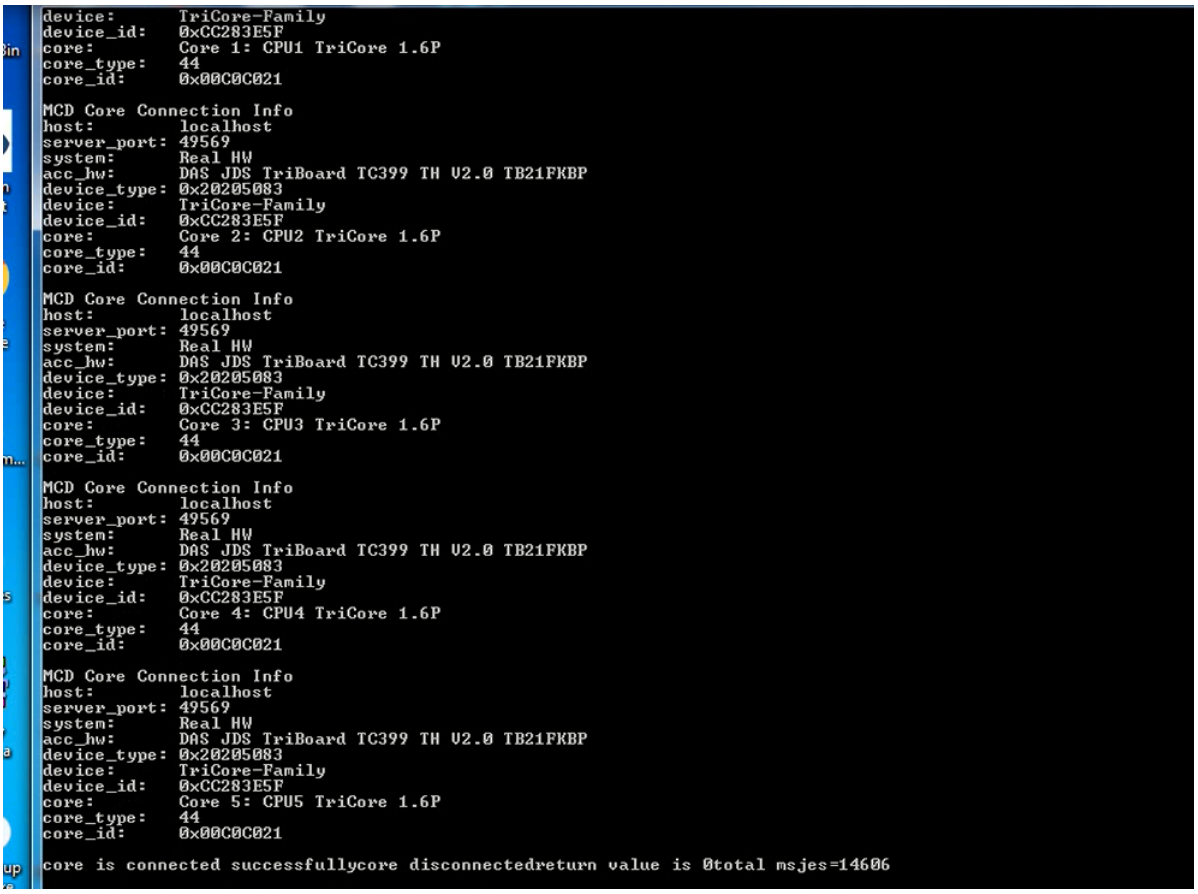


Figure 5.20: Traces collection

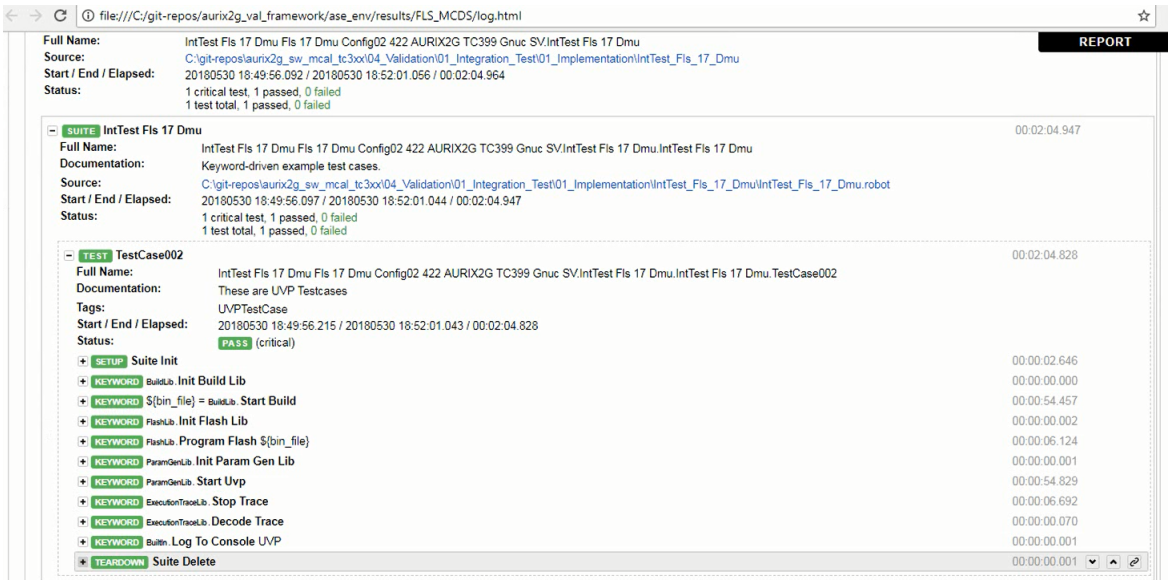
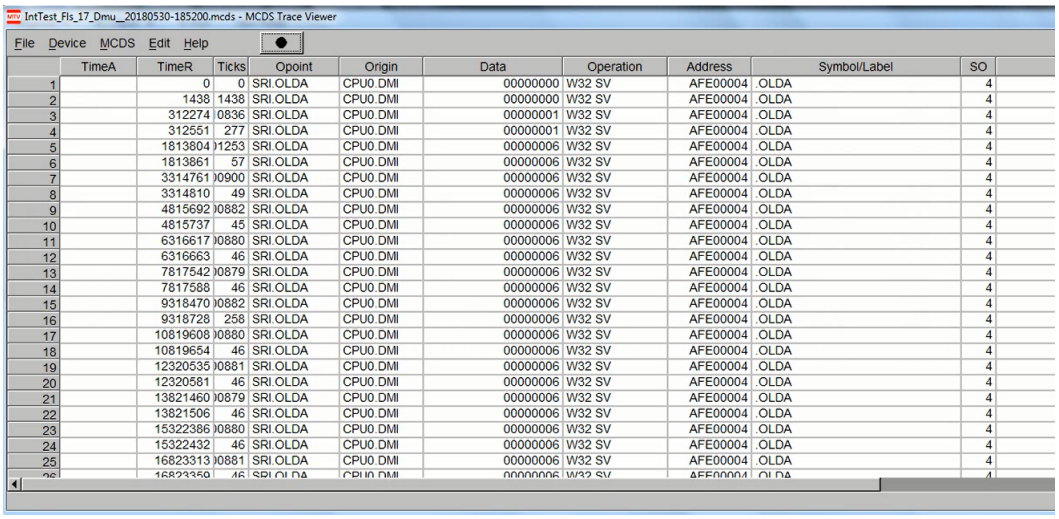


Figure 5.21: Log File



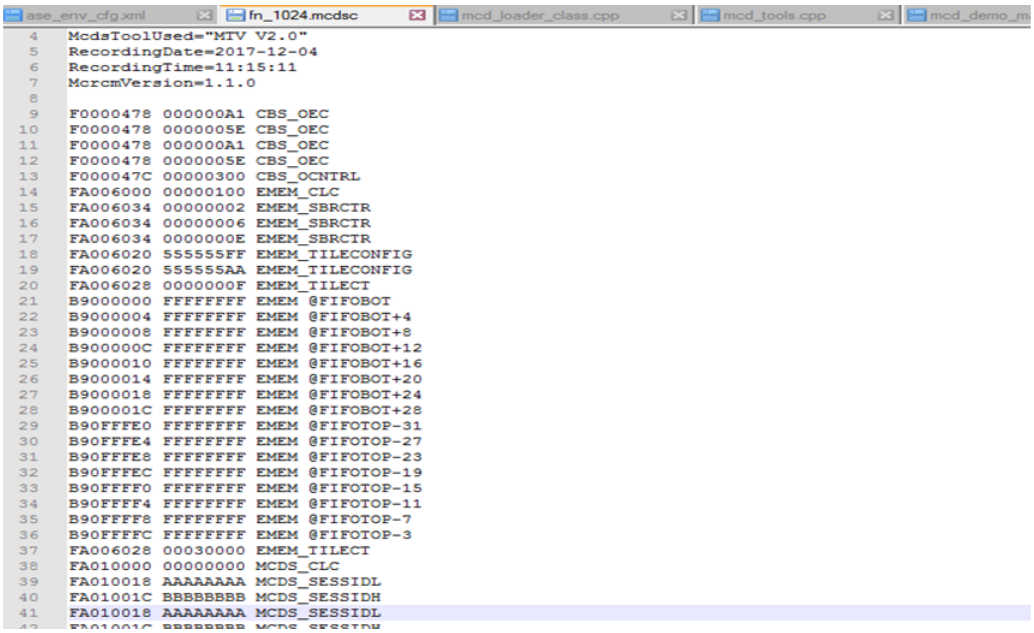
IntTest_Fls_17_Dmu__20180530-185200.mcds - MCDS Trace Viewer

File Device MCDS Edit Help

	TimeA	TimeR	Ticks	Opoint	Origin	Data	Operation	Address	Symbol/Label	SO
1		0	0	SRI.OLDA	CPU0.DMI	00000000	W32 SV	AFE00004	.OLDA	4
2		1438	1438	SRI.OLDA	CPU0.DMI	00000000	W32 SV	AFE00004	.OLDA	4
3		312274	0836	SRI.OLDA	CPU0.DMI	00000001	W32 SV	AFE00004	.OLDA	4
4		312551	277	SRI.OLDA	CPU0.DMI	00000001	W32 SV	AFE00004	.OLDA	4
5		1813804	1253	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
6		1813861	57	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
7		3314761	0900	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
8		3314810	49	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
9		4815692	0882	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
10		4815737	45	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
11		6316617	0880	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
12		6316663	46	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
13		7817542	0879	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
14		7817588	46	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
15		9318470	0882	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
16		9318728	258	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
17		10819608	0880	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
18		10819654	46	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
19		12320535	0881	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
20		12320581	46	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
21		13821460	0879	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
22		13821506	46	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
23		15322386	0880	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
24		15322432	46	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4
25		16823313	0881	SRI.OLDA	CPU0.DMI	00000006	W32 SV	AFE00004	.OLDA	4

Figure 5.22: Trace Results

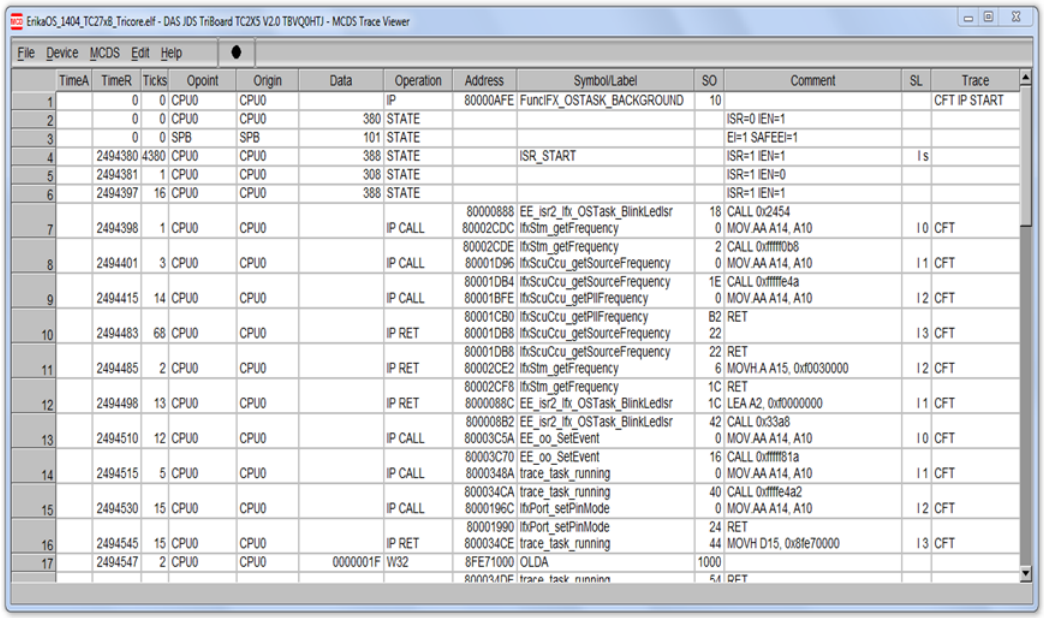
5.5 Configuring Hardware Directly



ase_env_cfg.xml | fn_1024.mcdsc | mcd_loader_class.cpp | mcd_tools.cpp | mcd_demo_m

```
4   McdsToolUsed="MTV V2.0"
5   RecordingDate=2017-12-04
6   RecordingTime=11:15:11
7   McrcmVersion=1.1.0
8
9   F0000478 000000A1 CBS_OEC
10  F0000478 0000005E CBS_OEC
11  F0000478 000000A1 CBS_OEC
12  F0000478 0000005E CBS_OEC
13  F000047C 00000300 CBS_OCNTRL
14  FA006000 00000100 EMEM_CLC
15  FA006034 00000002 EMEM_SBRCTR
16  FA006034 00000006 EMEM_SBRCTR
17  FA006034 0000000E EMEM_SBRCTR
18  FA006020 555555FF EMEM_TILECONFIG
19  FA006020 555555AA EMEM_TILECONFIG
20  FA006028 0000000F EMEM_TILECT
21  B9000000 FFFFFFFF EMEM @FIFOBOT
22  B9000004 FFFFFFFF EMEM @FIFOBOT+4
23  B9000008 FFFFFFFF EMEM @FIFOBOT+8
24  B900000C FFFFFFFF EMEM @FIFOBOT+12
25  B9000010 FFFFFFFF EMEM @FIFOBOT+16
26  B9000014 FFFFFFFF EMEM @FIFOBOT+20
27  B9000018 FFFFFFFF EMEM @FIFOBOT+24
28  B900001C FFFFFFFF EMEM @FIFOBOT+28
29  B90FFFE0 FFFFFFFF EMEM @FIFOTOP-31
30  B90FFFE4 FFFFFFFF EMEM @FIFOTOP-27
31  B90FFFE8 FFFFFFFF EMEM @FIFOTOP-23
32  B90FFFEC FFFFFFFF EMEM @FIFOTOP-19
33  B90FFFF0 FFFFFFFF EMEM @FIFOTOP-15
34  B90FFFF4 FFFFFFFF EMEM @FIFOTOP-11
35  B90FFFF8 FFFFFFFF EMEM @FIFOTOP-7
36  B90FFFFC FFFFFFFF EMEM @FIFOTOP-3
37  FA006028 00030000 EMEM_TILECT
38  FA010000 00000000 MCDS_CLC
39  FA010018 AAAAAAAA MCDS_SESSIDL
40  FA01001C BBBBBBBB MCDS_SESSIDH
41  FA010018 AAAAAAAA MCDS_SESSIDL
```

Figure 5.23: Register Configuration



	TimeA	TimeR	Ticks	Opoint	Origin	Data	Operation	Address	Symbol/Label	SO	Comment	SL	Trace
1		0	0	CPU0	CPU0		IP	80000AFE	FuncIFX_OSTASK_BACKGROUND	10			CFT IP START
2		0	0	CPU0	CPU0	380	STATE				ISR=0 IEN=1		
3		0	0	SPB	SPB	101	STATE				EI=1 SAFEEI=1		
4		2494380	4380	CPU0	CPU0	388	STATE		ISR_START		ISR=1 IEN=1	I s	
5		2494381	1	CPU0	CPU0	308	STATE				ISR=1 IEN=0		
6		2494397	16	CPU0	CPU0	388	STATE				ISR=1 IEN=1		
7		2494398	1	CPU0	CPU0		IP CALL	80000888 80002CDC	EE_isr2_Ifx_OSTask_BlinkLedIsr IfxStm_getFrequency	18 0	CALL 0x2454 MOV.AA A14, A10	I 0	CFT
8		2494401	3	CPU0	CPU0		IP CALL	80002CDE 80001D96	IfxStm_getFrequency IfxScuCcu_getSourceFrequency	2 0	CALL 0xfffff0b8 MOV.AA A14, A10	I 1	CFT
9		2494415	14	CPU0	CPU0		IP CALL	80001DB4 80001BFE	IfxScuCcu_getSourceFrequency IfxScuCcu_getPllFrequency	1E 0	CALL 0xfffffe4a MOV.AA A14, A10	I 2	CFT
10		2494483	68	CPU0	CPU0		IP RET	80001CB0 80001DB8	IfxScuCcu_getPllFrequency IfxScuCcu_getSourceFrequency	B2 22	RET	I 3	CFT
11		2494485	2	CPU0	CPU0		IP RET	80001DB8 80002CE2	IfxScuCcu_getSourceFrequency IfxStm_getFrequency	22 6	RET MOVH.A A15, 0xf0030000	I 2	CFT
12		2494498	13	CPU0	CPU0		IP RET	80002CF8 8000088C	IfxStm_getFrequency EE_isr2_Ifx_OSTask_BlinkLedIsr	1C 1C	RET LEA A2, 0xf0000000	I 1	CFT
13		2494510	12	CPU0	CPU0		IP CALL	800008B2 80003C5A	EE_isr2_Ifx_OSTask_BlinkLedIsr EE_oo_SetEvent	42 0	CALL 0x33a8 MOV.AA A14, A10	I 0	CFT
14		2494515	5	CPU0	CPU0		IP CALL	80003C70 8000348A	EE_oo_SetEvent trace_task_running	16 0	CALL 0xfffff81a MOV.AA A14, A10	I 1	CFT
15		2494530	15	CPU0	CPU0		IP CALL	800034CA 8000196C	trace_task_running IfxPort_setPinMode	40 0	CALL 0xfffffe4a2 MOV.AA A14, A10	I 2	CFT
16		2494545	15	CPU0	CPU0		IP RET	80001990 800034CE	IfxPort_setPinMode trace_task_running	24 44	RET MOVH D15, 0x8fe70000	I 3	CFT
17		2494547	2	CPU0	CPU0	0000001F	W32	8FE71000	OLDA	1000			

Figure 5.24: Traces

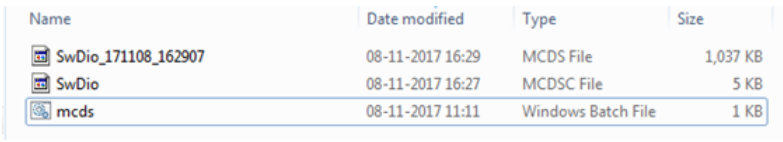
5.6 Automation with bat files

mcds.bat file

```
set MTV_EXE="mcds_trace_viewer.exe"
set MCDSC_FILE="SwDio.mcdsc"

%MTV_EXE% -batchMode %MCDSC_FILE% -resetDeviceFirst -
readCodeFromDevice -timeInName
```

Figure 5.25: mcds.bat file



Name	Date modified	Type	Size
SwDio_171108_162907	08-11-2017 16:29	MCDS File	1,037 KB
SwDio	08-11-2017 16:27	MCDSC File	5 KB
mcds	08-11-2017 11:11	Windows Batch File	1 KB

Figure 5.26: Trace Data

Cmd options

- Show Trace Data Raw information:

  mcds_trace_viewer.exe showTdrInfo

- Windows Explorer double click mode:

mcds_trace_viewer.exe .mcds file name

mcds_trace_viewer.exe .mcdsc file name

- Trace recording batch mode:

  mcds_trace_viewer.exe -batchMode [-resetDeviceFirst] [-readCodeFromDevice] [-timeInName] .mcdscfn

- File conversion batch mode:

  mcds_trace_viewer.exe -batchMode [-showTdrInfo] .mcds file name

Options:

-showTdrInf - Show Trace Data Raw (TDR) information as additional columns in trace table

-batchMode - Record or convert a file as a command line option

-resetDeviceFirst - Device is reset (PORST), MCDS is configured and then CPU0 is started

-readCodeFromDevice - After recording the code, which is used in the trace, is read from the device and included in the .mcds file

-timeInName - The name of the .mcds file will be .mcdsc filename_YYMMDD_HHMMSS.mcds

Chapter 6

Conclusion and Future Work

6.1 Conclusion

SlNum	API Name		Min Time(us)	Max Time(us)	WCET(us)
1	Dio_FlipChannel	RapiTime	0.5800000000us	0.6600000000us	0.6600000000us
		MCDS(us)	0.2574	0.3762	
2	Dio_ReadChannel	RapiTime	0.5300000000us	0.5300000000us	0.5300000000us
		MCDS	0.1518	0.2838	
3	Dio_ReadChannelGroup	RapiTime	0.3000000000us	0.3500000000us	0.3500000000us
		MCDS	0.1452	0.1782	
4	Dio_ReadPort	RapiTime	0.3000000000us	0.3000000000us	0.3000000000us
		MCDS	0.1452	0.198	
5	Dio_WriteChannel	RapiTime	0.5200000000us	0.5700000000us	0.5700000000us
		MCDS	0.1518	0.264	
6	Dio_WriteChannelGroup	RapiTime	0.2900000000us	0.3300000000us	0.3300000000us
		MCDS	0.1056	0.1188	

Figure 6.1: Comparison of Timings by RapiTime and MCDS

Above Figure shows the difference of timings calculated by previous Third Party software Rapitime and Non-Intrusive current proven methodology. From the Results above we can conclude that current methodology is better than previous as,

- It improves accuracy of results.
- Inclusion of less overhead time, as its not intrusive.
- Independent of Third party software, timings calculated by hardware itself.
- Integrated with the present Architecture

- Can work in the Regression Framework
- Cost Effective

6.2 Future Work

Currently, these timings are Execution Timings. There are many factors which needs to be considered and has a major impact on the WCET such as cache memory, interrupts, scheduling, bus availability etc. By considering all these factors, an enhancement to current methodology will be done.
This is implemented only for one device. The management and configuration framework for various devices with different version of MCDS, like MiniMCDS, MCDSLite has to be done.

References

[1] Colin, Antoine and Guillem Bernat."Scope-tree: A program representation for symbolic worst-case execution time analysis",*in Proceedings of 14th Euromicro Conference on Real-Time Systems*, pp. 50-59, 2002.

[2] Petru Eles, Zebo Peng and Alexandru Andrei,"Predictable Worst Case Execution Time Analysis for Multiprocessor Systems on Chip",*in Proceedings of Sixth IEEE International Symposium on Electronic Design and Application*, pp. 99-104, 2011.

[3] Philippa Ryan, Mikel, Perez, Jon and Mezzetti,"WCET analysis methods: Pitfalls and challenges on their trustworthiness",*in Proceedings of 10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pp. 1-10, 2015.

[4] Park, Minho and Brewerton,"Intelligent ECU End of Line Testing to Support ISO26262 Functional Safety Requirements",*SAE International Journal of Passenger Cars-Electronic and Electrical Systems*,vol. 6, pp. 162-168, 2013.

[5] Christian Zoubek, Peter Trommler,"Overview of worst case execution time analysis in single-and multicore environments",*in Proceedings of Conference on Architecture of Computing System*, pp. 1-5, 2017.

[6] Bernat, Guillem and Burns,"An approach to symbolic worst-case execution time analysis",*in Proceedings of International Federation of Automatic Control (IFAC)*, vol. 33, no. 7, pp. 43-48, 2000.

[7] Bernat, Guillem and Colin, Stefan M,"WCET Analysis of Probabilistic Hard Real-Time Systems", *Institute of Electrical and Electronics Engineers (IEEE)*, pp. 279-288, 2002.

[8] Yiqiang Ding and Wei Zhang, "Loop-Based Instruction Prefetching to Reduce the Worst-Case Execution Time" ,*IEEE TRANSACTIONS ON COMPUTERS*, vol. 59, pp. 855-864, 2010.

[9] Gil, George, Santinelli, Adriana and Cucu-Grosjean, "Open Challenges for Probabilistic Measurement-Based Worst-Case Execution Time",*IEEE Embedded Systems Letters*, vol. 9, pp. 69-72, 2017.

[10] Thomas and Saarinen, Sami, "Worst-case execution time analysis for digital signal processors",*in Proceedings of 10th European Conference on Signal Processing*, pp. 1-4, 2000.

[11] Ashraf Suyyagh, Zeljko Zilic, "Real-time benchmark set synthesis based on pWCET estimation and bounded hyper-periods",*in Proceedings of International Conference on Circuits System and Simulation*, pp. 129-133, 2017.

vinni

ORIGINALITY REPORT

9%	7%	4%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

MATCH ALL SOURCES (ONLY SELECTED SOURCE PRINTED)

4%

★ robotframework.googlecode.com

Internet Source

Exclude quotes	On	Exclude matches	< 8 words
Exclude bibliography	On		