COMPREHENSIVE REAL-TIME ANALYSIS ON INFINEON AURIX™

Application Note No: 50-0005-00
Revision 1.1
September 28th, 2020

COPYRIGHT NOTICE

NOTICE

The information contained in this application is subject to change without notice. Product specifications cited are those in effect at time of publication. INCHRON shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

INCHRON expressly disclaims all responsibility and liability for the installation, use, performance, maintenance and support of third-party products. Customers are advised to make their own independent evaluation of such products.

DISCLAIMER OF WARRANTIES AND LIMITATION OF LIABILITY

There are no understandings, agreements, representations, or warranties either expressed or implied, including warranties of merchantability or fitness for a particular purpose, other than those specifically set out by any existing contract between the parties. Any such contract states the entire obligation of the seller. The contents of this application note shall not become part of or modify any prior or existing agreement, commitment, or relationship.

The information, recommendations, descriptions, and safety notices in this application note are based on INCHRON experience and judgment with respect to operation and maintenance of the described product. This information should not be considered as all–inclusive or covering all contingencies. If further information is required, INCHRON AG, should be consulted.

No warranties, either expressed or implied, including warranties of fitness for a particular purpose or merchantability, or warranties arising from the course of dealing or usage of trade, are made regarding the information, recommendations, descriptions, warnings, and cautions contained herein.

In no event will INCHRON be responsible to the user in contract, in tort (including negligence), strict liability or otherwise for any special, indirect, incidental, or consequential damage or loss whatsoever, including but not limited to: damage or loss of use of equipment, cost of capital, loss of profits or revenues, or claims against the user by its customers from the use of the information, recommendations, descriptions, and safety notices contained herein.

CONTACT

INCHRON AG

Phone +49 331 / 27978920

Neumühle 24-26
D-91056 Erlangen
GERMANY

info@inchron.com
www.inchron.com

1 EXECUTIVE SUMMARY

Infineon's Multi-Core Debug Solution (MCDS) provides sophisticated on-chip debugging means on the Infineon AURIX™ platform. The MCDS can be accessed and configured via a PC that runs the Device Access Server (DAS) software.

Among other features, this solution supports tracing of various on-chip signals, with no need for external tracing hardware. For software development on AURIX™, this is a huge benefit in terms of cost-effectiveness and usability. By default, Infineon's MCDS Trace Viewer (MTV) is the tool of choice to analyze MCDS trace logs.

When it comes to comprehensive end-to-end analysis of real-time systems, however, tools developed and optimized for this particular purpose are required on top. The INCHRON Tool-Suite provides a complete set of seamlessly integrated tools, supporting real-time analysis methods at the state of the art. Well-known players in the automotive, avionics, and communications industries are relying on the INCHRON Tool-Suite as their timing tool of choice for professional real-time systems development.

The purpose of this application note is to enable developers to benefit from the combined advantages of Infineon's Multi-Core Debug Solution and the unique real-time performance analysis capabilities of the INCHRON Tool-Suite. This combination provides fully automated real-time analysis of MCDS trace logs (pre-recorded or live-recorded), as well as computer-aided visual inspection and analysis.

2 INTRODUCTION

This application note describes the usage of the MTV tool, the INCHRON MCDS Trace Importer which is required to import MCDS trace logs into the INCHRON Tool-Suite, and the visualization as well as the automated real-time analysis of the imported data.

First, the required trace information will be identified. The configuration and tracing via INCHRON MCDS Trace Importer or MTV will be explained.

Also, an example for a MCDSJSON file for trace import configuration is given, and the import is described step-by-step. For a comprehensive list of all supported MCDSJSON file features see the MCDS Trace Importer manual.
As last point, some examples for data visualization and analysis are described.

For this application note, a task system (Erika OS) with some application function calls and data is being used as an example for trace visualization. For all examples, an AURIX™ TC397 microcontroller was used.

3 TOOLS USED

INCHRON Tool-Suite 2.9.2
INCHRON MCDS Trace Importer V1.0
Infineon DAS with MCDS Trace Viewer (MTV) V7.1

4 IDENTIFICATION OF TRACE INFORMATION

As mentioned before, this application note describes the tracing of a simple single core task system. Infineon provides for all AURIX™ TC3xx example software frameworks (BaseFrameworks_A2G) for this task system. It is available for download from Infineon's 'myInfineon' portal.
The example comes with the following tasks:

- Task1ms
- Task5ms
- Task10ms
- Task20ms
- Task50ms
- Task100ms
- BackgroundTask
- InitTask

For demonstrating the function tracing, the following functions are used:

- void TestApp_10ms_WorkerA_C0(void)
- void TestApp_10ms_WorkerA_Sub_C0(void)
- void TestApp_100ms_WorkerB_C0(void)

The data value tracing is shown for the variable:

- ExampleCounterC0

4.1 TRACE POINTS FOR TASKS

The information of task states can be stored in different ways inside the task system.
In our example, the information about the task state is stored in a variable for each task. The corresponding array is called ‘EE_th_status’.

There are other possible solutions, like a variable with a current task ID and an additional variable with the state of the scheduled task. This can also be traced, but it is outside of the scope of this application note.

```
/* thread status */
volatile    EE_TYPESTATUS EE_th_status[EE_MAX_TASK+1] = {
```

From the header file you get the information about the different values for the task/process state:

```
/* Constant of data type TaskStateType for task state running. */
#define RUNNING   ((EE_TYPESTATUS)0U)

/* Constant of data type TaskStateType for task state waiting. */
#define WAITING   ((EE_TYPESTATUS)1U)

/* Constant of data type TaskStateType for task state ready. */
#define READY     ((EE_TYPESTATUS)2U)

/* Constant of data type TaskStateType for task state suspended. */
#define SUSPENDED ((EE_TYPESTATUS)3U)
```

The task definition is also given by a header file:

```
/* TASK definition */
#define EE_MAX_TASK 11
#define IFX_OSTASK_EVENT1 0
#define IFX_OSTASK_EVENT2 1
#define IFX_OSTASK_EVENT3 2
#define IFX_OSTASK_1MS 3
#define IFX_OSTASK_5MS 4
#define IFX_OSTASK_10MS 5
#define IFX_OSTASK_20MS 6
#define IFX_OSTASK_50MS 7
#define IFX_OSTASK_100MS 8
#define IFX_OSTASK_BACKGROUND 9
#define IFX_OSTASK_INIT 10
#define COUNTER_OFFSET 11
```

The MCDS Trace Importer uses the ELF file to resolve the variable addresses.

For the MTV tool you must manually configure the addresses of the variables from the map file information:

```
| EE_th_rnact_max                    | 0x80003e0c |              |
| EE_th_status                       | 0x70000230 |              |
| EE_th_terminate_nextask            | 0x70000260 |              |
```

4.2 TRACE POINTS FOR FUNCTIONS

The MCDS Trace Importer uses the ELF file to resolve the function addresses. Therefore, manual configuration is not needed. The importer will show all functions that lie within the configured address range as well as calls from and returns to this address range.

For the MTV configuration you need the function addresses. After a fresh build, these might change and have to be determined again by consulting the MAP file. The trace file will cover a short duration only, if you trace all functions, because of the trace buffer limit.

From the map file:

```
| mpe:pfls0  |                 | .text.TestApp.TestApp_100ms_C0 (53)
| 0x0000001a | 0x80001bb0 | 0x00001bb0 | 0x00000002 |
| mpe:pfls0  |                 | .text.TestApp.TestApp_100ms_C1 (58)
| 0x00000040 | 0x80001bca | 0x00001bca | 0x00000002 |
| mpe:pfls0  |                 | .text.TestApp.TestApp_100ms_WorkerB_C0 (52)
| 0x0000001a | 0x80001c0a | 0x00001c0a | 0x00000002 |
| mpe:pfls0  |                 | .text.TestApp.TestApp_100ms_WorkerB_C1 (57)
| 0x00000018 | 0x80001c24 | 0x00001c24 | 0x00000002 |
| mpe:pfls0  |                 | .text.TestApp.TestApp_10ms_C0 (56)
| 0x0000001a | 0x80001c3c | 0x00001c3c | 0x00000002 |
| mpe:pfls0  |                 | .text.TestApp.TestApp_10ms_Cx (61)
| 0x0000001a | 0x80001c56 | 0x00001c56 | 0x00000002 |
| mpe:pfls0  |                 | .text.TestApp.TestApp_10ms_WorkerA_C0 (55)
| 0x0000002c | 0x80001c70 | 0x00001c70 | 0x00000002 |
| mpe:pfls0  |                 | .text.TestApp.TestApp_10ms_WorkerA_Cx (60)
| 0x0000002e | 0x80001c9c | 0x00001c9c | 0x00000002 |
| mpe:pfls0  |                 | .text.TestApp.TestApp_10ms_WorkerA_Sub_C0 (54)
| 0x00000018 | 0x80001cca | 0x00001cca | 0x00000002 |
| mpe:pfls0  |                 | .text.TestApp.TestApp_10ms_WorkerA_Sub_Cx (59)
| 0x0000001a | 0x80001ce2 | 0x00001ce2 | 0x00000002 |
```

4.3 TRACE POINTS FOR DATA VARIABLES

For this example, the array for the task information was extended by one entry to store a data variable (EE_th_status[11]).

```
/* thread status */
volatile    EE_TYPESTATUS EE_th_status[EE_MAX_TASK+1] = {
```

The definition of the offset was also extended inside the header file:

```
/* TASK definition */
#define EE_MAX_TASK 11
#define IFX_OSTASK_EVENT1 0
#define IFX_OSTASK_EVENT2 1
#define IFX_OSTASK_EVENT3 2
#define IFX_OSTASK_1MS 3
#define IFX_OSTASK_5MS 4
#define IFX_OSTASK_10MS 5
#define IFX_OSTASK_20MS 6
#define IFX_OSTASK_50MS 7
#define IFX_OSTASK_100MS 8
#define IFX_OSTASK_BACKGROUND 9
#define IFX_OSTASK_INIT 10
#define COUNTER_OFFSET 11
```

5 TRACING

The MCDS tracing can be configured and started either from within the INCHRON Tool-Suite or from within the Infineon MTV tool.

The advantage of using the INCHRON Tool-Suite is, that you will get the trace and the visualization in one step. Moreover, all your configuration is stored inside one file (MCDSJSON) and there is no need to adapt your configuration after using a new version of your application with new addresses for the data variables or functions.
Controlling the trace generation via the INCHRON Tool-Suite also provides another important benefit: Due to improved trace buffer handling, tracing will be continuous without any limitation by the trace buffer sizes on the AURIX MCDS.

You can also use MCDS files from the MTV that offer the possibility to create dedicated MCDS config files (MCDSC). These files can be exported from inside the MTV. However, inside the MTV you must again adapt all addresses manually if you have a fresh build with changed addresses. The configuration then must be exported yet again.

5.1 INCHRON TOOL-SUITE

For more information regarding the import see the INCHRON MCDS Trace Importer User Manual.

The configuration for the import is defined inside a MCDSJSON file. This file contains all information that is not derived automatically from the ELF file.

5.1.1 ADAPT MCDSJSON FILE

The file ‘MANIFEST_Suite.mcdsjson’ already contains all parts, which are listed inside this chapter.

5.1.2 SET COMMON INFORMATION

First, set the project name and common CPU information.

```
"project": {
    "name": "DAS_TC39B_Importer"
    "cpu": [{
        "name": "TC397",
        "num_cores": 1,
        "frequency": "150Mhz"
    }]
},
```

5.1.2.1 CONFIGURE OBSERVATION POINTS

Inside the opoints part of the file, set status trace to true.

```
"opoints": [{
    "status_trace": true,
```

By setting the program trace address range, we limit which functions are traced (see also Sec. 4.2).

```
    "program_trace": {
        "address": [
        { "from": "TestApp_100ms_C0",
          "to": "TestApp_10ms_WorkerA_Sub+0x13"}
        ]
    },
```

The program trigger defines a point in the trace where a specific action happens. In the case of this example, the trace recording finished some time after the trigger hits.

```
    "program_trigger": {
        "address": { "from": "TestApp_100ms_C0" },
        "action": "trace_rec"
    },
```

The data trace is the most important field for this example. All state changes for tasks are recorded as data values inside ee_th_status. The data trace field enables the observation of write accesses of these addresses.

```
    "data_trace": {
        "mode": "ad_w",
        "range": "in",
        "address": [
        { "from": "EE_th_status+0xC",
          "to": "EE_th_status+0x2F" }
        ]
    }
}]
```

5.1.2.2 CONFIGURE TASK

Tasks are translated in the MCDS Importer using a pattern matching methodology. That is, a pattern refers to a received message and a record specifies an INCHRON ISF trace entry.

For each task (1ms, 5ms, …) and each value (start, preempted, terminate) of the task variables provide one event entry.

For example, the terminate entry for the 1ms task is:

```
"events": [
    {
        "pattern": {
            "type": "data",
            "address": "EE_th_status+0xC",
            "data": "0x03",
            "readnotwrite": "false"
        },
        "records": [
            {
                "process": "Task1msC0",
                "type": "process",
                "event": "terminate"
            }
        ]
    }
]
```

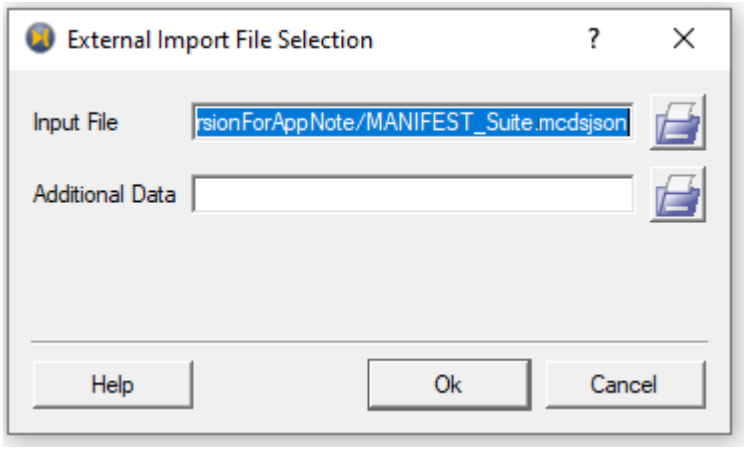
5.1.2.3 CONFIGURE DATA VARIABLES

To be able to trace plain data values, a counter event is used. Thus, the following entry for the data variable (EE_th_status[11]) needed:

```
"events": [
    {
        "pattern": {
            "type": "data",
            "address": "EE_th_status+0x2C",
            "data": "0",
            "datamask": "0",
            "readnotwrite": "false"},
        "name": "Ctr001",
        "records": [
            {
                "type": "event",
                "event": "ExampleCounterC0",
                "value" : "datalowword"
            }
        ],
        "description": "Counter on CPU0"
    }
]
```

5.1.3 TRACING

Open the INCHRON Tool-Suite and click on '*File* → *External Trace Importer…*'.
The dialog below appears:



Select your MCDSJSON file 'MANIFEST_Suite.mcdsjson'.

After click on 'Ok' the tracing process starts. Once the import is completed, the INCHRON ISF trace file created will be opened automatically in chronVIEW.

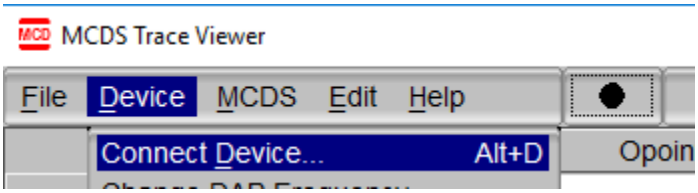
5.2 MTV TOOL

The MCDS Trace Viewer (MTV) is part of the Infineon DAS tools.
The installation and setup of the DAS server and tools are not part of this application note.
The DAS tools are provided by Infineon and can be found here
https://www.infineon.com/cms/en/product/promopages/das/.

A prerequisite for the next steps is the successful installation of the DAS tools and a connected Infineon Aurix device.

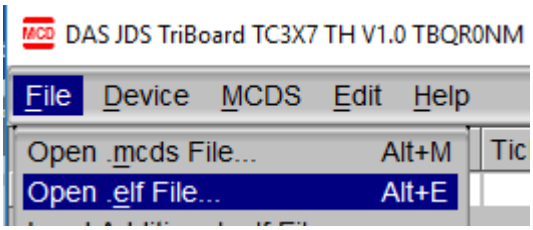
5.2.1 CONFIGURATION

1. Set the used device and connect:



Select the device, in this example ‘TriCore-Family’.

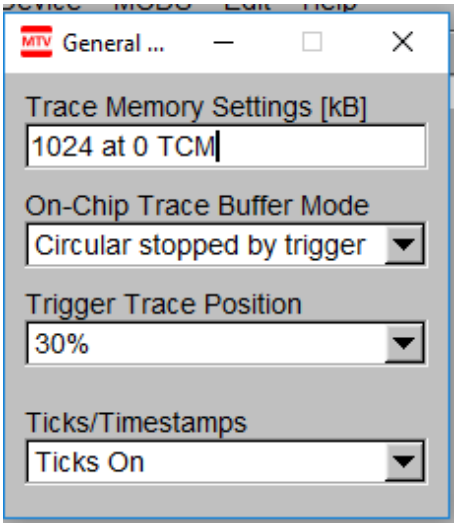
2. Load the Elf-File:



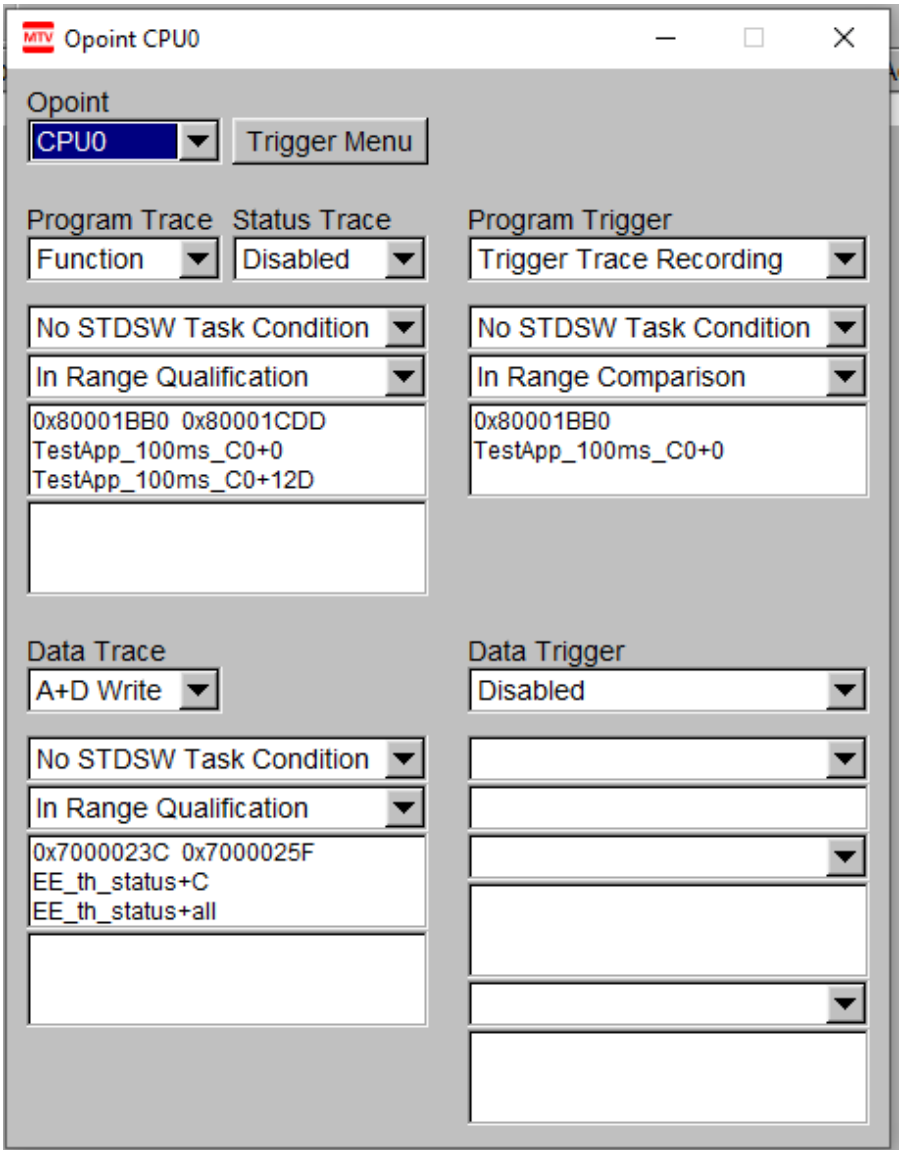
Use the same ELF file as you have used for flashing the device.

3. Create configuration:

Set ‘reset device first’ option by selecting menu item ‘*Device → Reset Device First’*.
Set up general settings by selecting menu item ‘*MCDS → General’* as shown below:

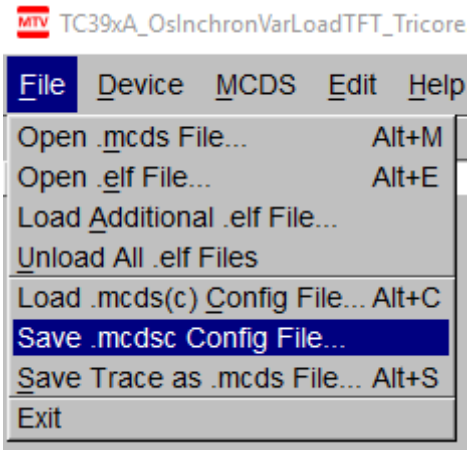


Now add the trigger and trace ranges by selecting menu item '*MCDS → Opoint CPU0'* as shown below:



- For the function trace add the start address (0x80001BB0) of the first function (TestApp_100ms_C0) and the end address (0x80001CDD). The symbolic names are completed automatically.
- For the data trace add the start address (0x7000023C) of the first relevant task (IFX_OSTASK_1MS) and the end address (0x7000025F). The symbolic names are again completed automatically.
- The function 'TestApp_100ms_C0' is used as a trigger. The address 0x80001BB0 must be inserted in the 'Program Trigger' part.

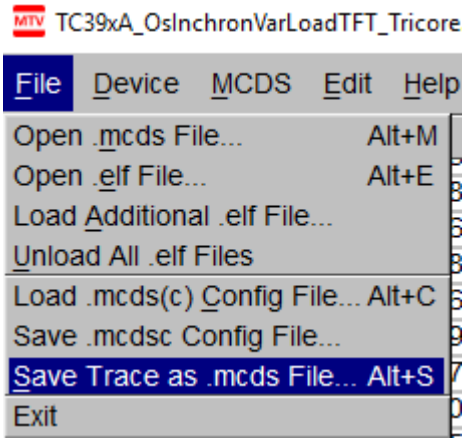
4. For later use the configurations can be saved into a MCDSC File.



You can later load this file by selecting the menu item '*File → Load .mcds(c) Config File…*'.

5.2.2 TRACING

Now you can start the trace by click on the button with the black dot.
After this, the dot color changes to red. If the trigger has occurred, the color changed to yellow. When the color changes back to black, the tracing is finished. Now you can see the trace data and save it to a MCDS file:



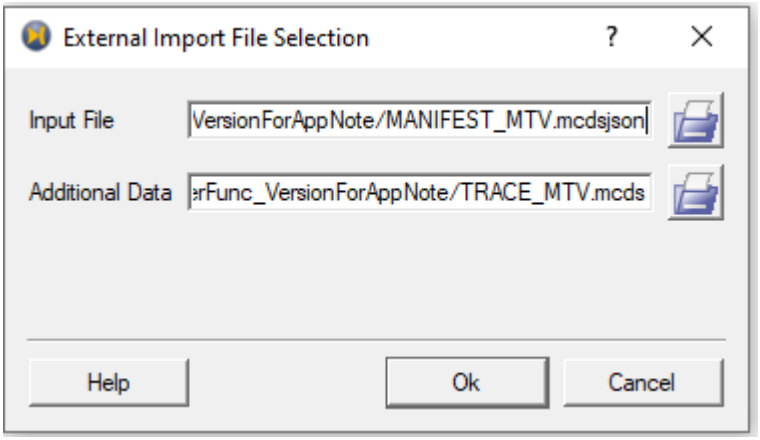
5.2.3 IMPORT USING THE INCHRON TOOL-SUITE

5.2.3.1 MCDSJSON FILE

The MCDSJSON file is the same as for the INCHRON Tool-Suite tracing, except that the 'MCDS' section is not required. If there is a MCDS section inside the MCDSJSON file, this part will be ignored.

5.2.3.2 IMPORT

Open the INCHRON Tool-Suite and click on '*File* → *External Trace Importer…*'.
The dialog below appears:



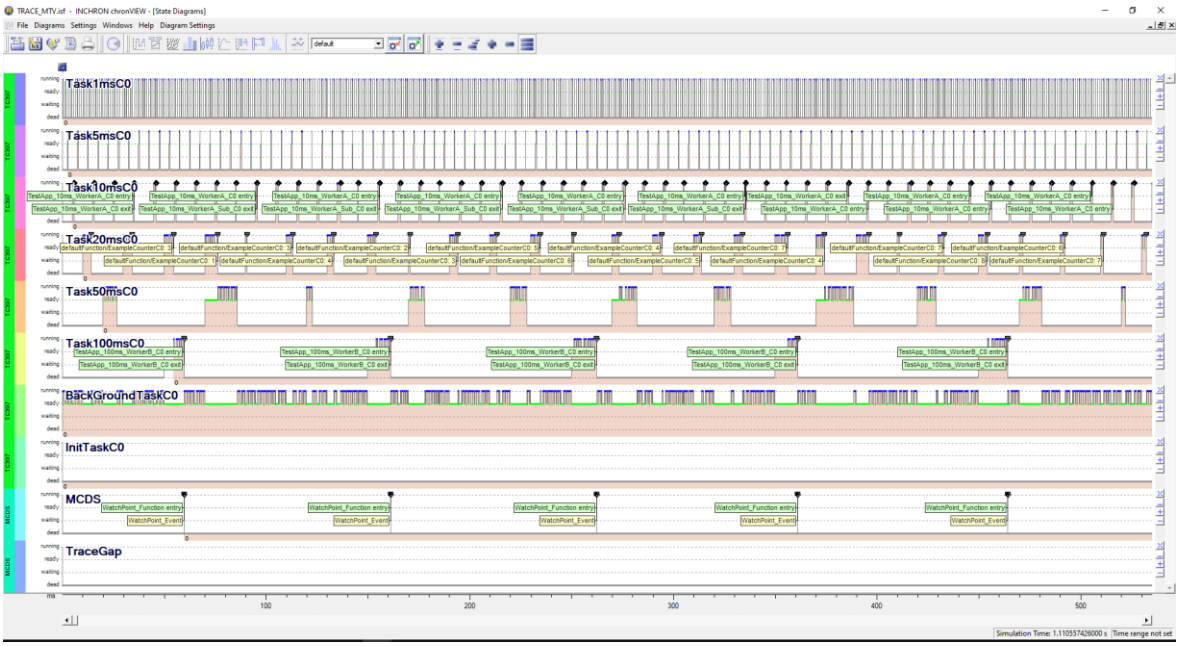
Select your MCDSJSON file 'MANIFEST_MTV.mcdsjson' as 'Input File' and your trace files 'TRACE_MTV.mcds' as 'Additional Data' file.

After click on 'Ok' the import process starts. When the import has ended, the created 'MANIFEST_MTV-0.isf' trace file will be opened automatically in chronVIEW.

6 VISUALIZATION AND ANALYSIS

6.1 BASE CONFIGURATION CHRONVIEW

After the import, you will see a window like the one below:

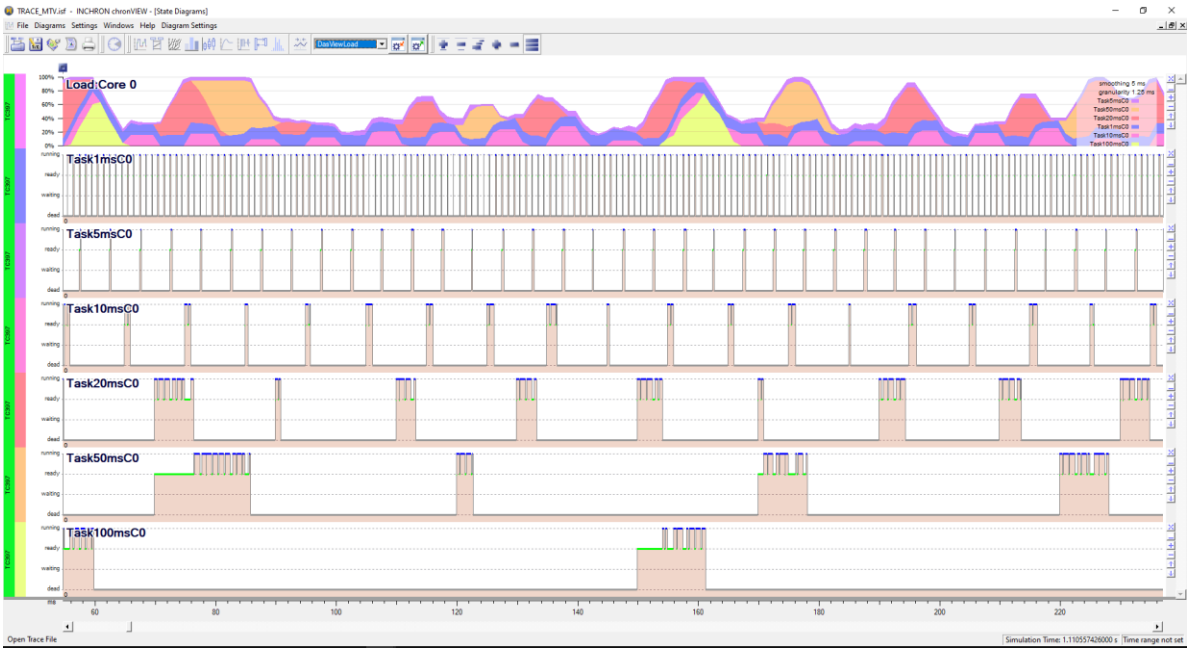


6.1.1 VISUALIZATION OF TASKS

To obtain more information, the diagram is configurable as follows:

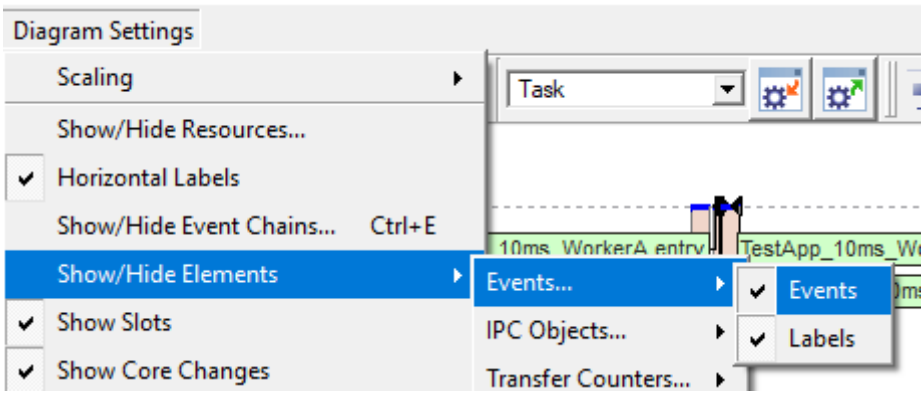
- Select menu item '*Diagram Setting* → *Show/Hide Resources…*', show the 'CPU0 load' and hide the 'MCDS' process, the 'TracGap' process and the 'InitTask' process.
- Right click on BackgroundTask and select 'Idle Task' to prevent it from being shown in the load diagram.
- Save this view profile for the next run.

Now you will get an overview over your cyclic tasks like this:

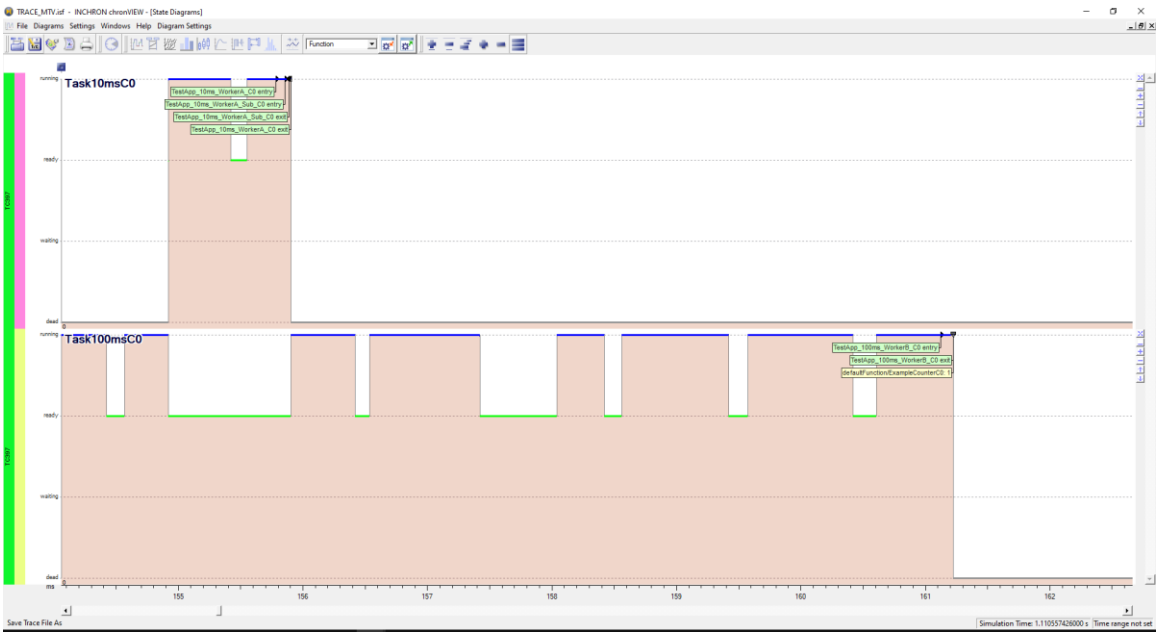


6.1.2 VISUALIZATION OF FUNCTIONS

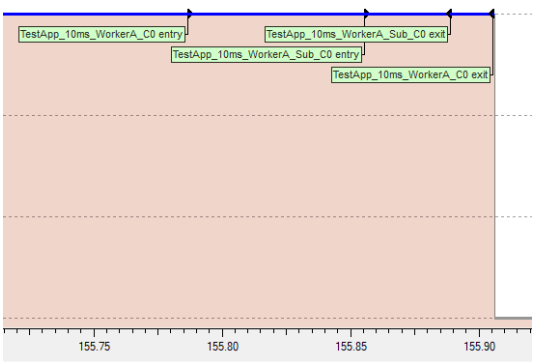
To get a detailed view of function calls and returns, click the menu item '*Diagram Settings → Show/Hide Resources…*' again and hide all tasks except from the 10ms and 100ms task.
Disable the events (menu ''*Diagram Setting → Show/Hide Elements → Events… → Events'*).



Now zoom in and you will get a detailed view on the observed functions:

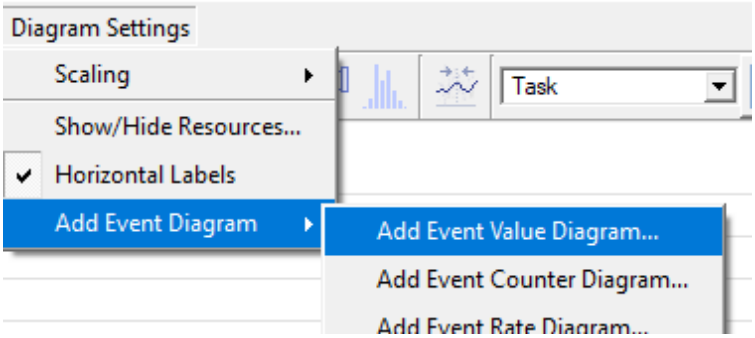


Zoom in further:



6.1.3 VISUALIZATION OF DATA VARIABLES

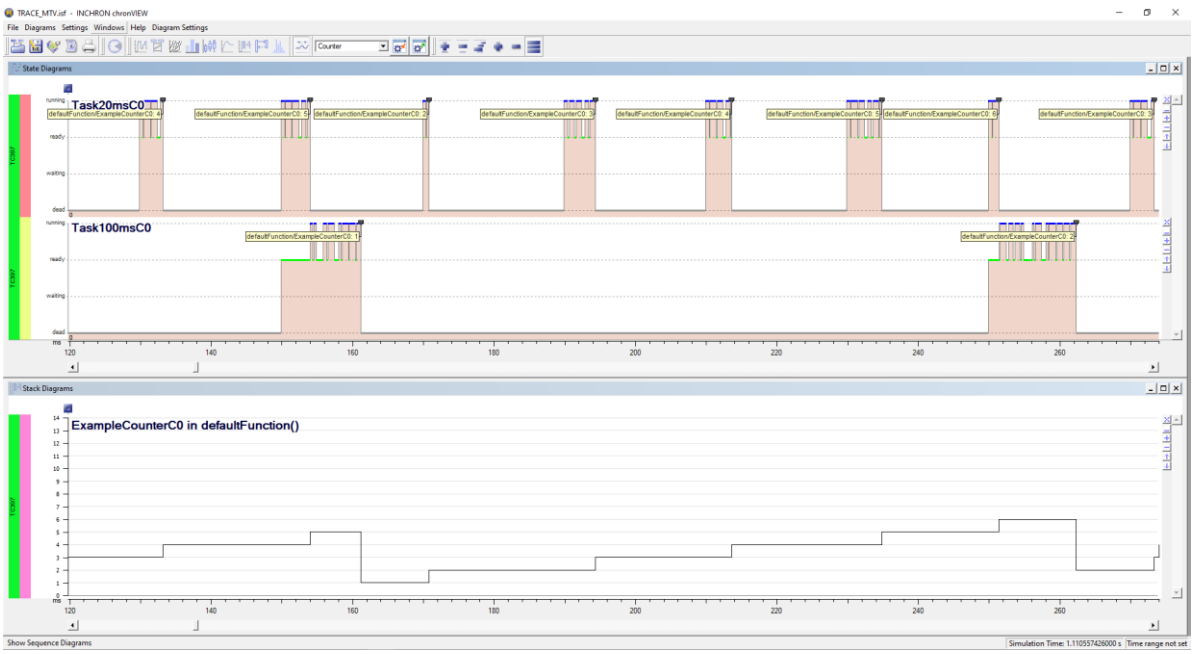
Open a Stack diagram (menu '*Diagrams* → *Show Stack Diagram*') and add an event value diagram:



Here select the 'ExampleCounterC0'. This name comes from the MCDSJSON file event configuration.
For better visibility perform these steps:

- Select menu '*Diagram Setting* → *Show/Hide Resources…*' and hide all other processes.
- Menu *'Window* → *Tile Windows Horizontally'*
- Select the 'State Diagram Window' and use the menu '*Diagram Setting* → *Show/Hide Resources…*' to hide all processes except the 20ms and 100ms task.
- Switch on the events '*Diagram Setting* → *Show/Hide Elements* → *Events…* → *Events'*
- Switch off the function events '*Diagram Setting* → *Show/Hide Elements* → *Functions Events…* → *Functions Events'*
- Synchronize the diagrams via '*Diagrams* → *Sync Diagrams*'

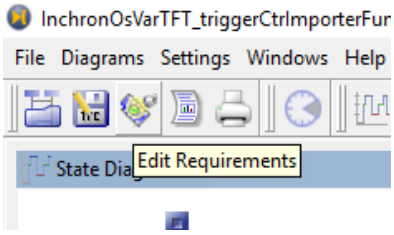
Now we have a good overview of the data values und the processes that change the data value:



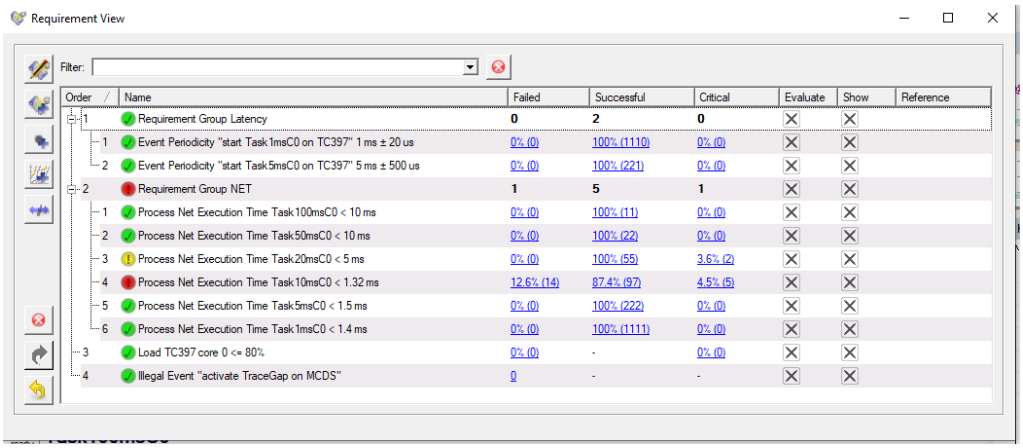
6.2 REQUIREMENTS AND SEQUENCE OF EVENTS

Now you can also define requirements and a chain of events inside this INCHRON.isf trace file:

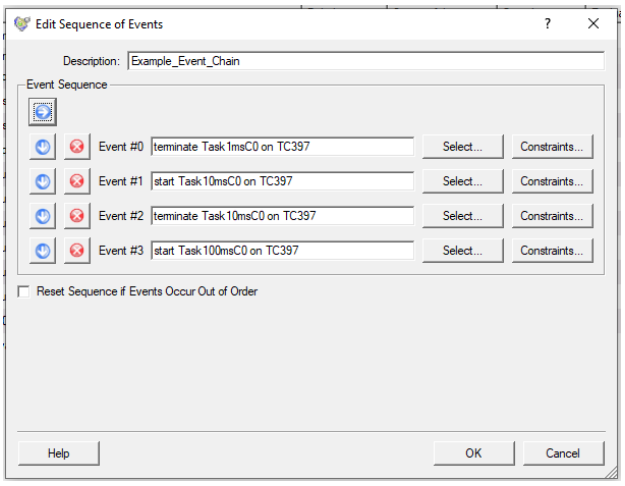
1. Open requirements view:



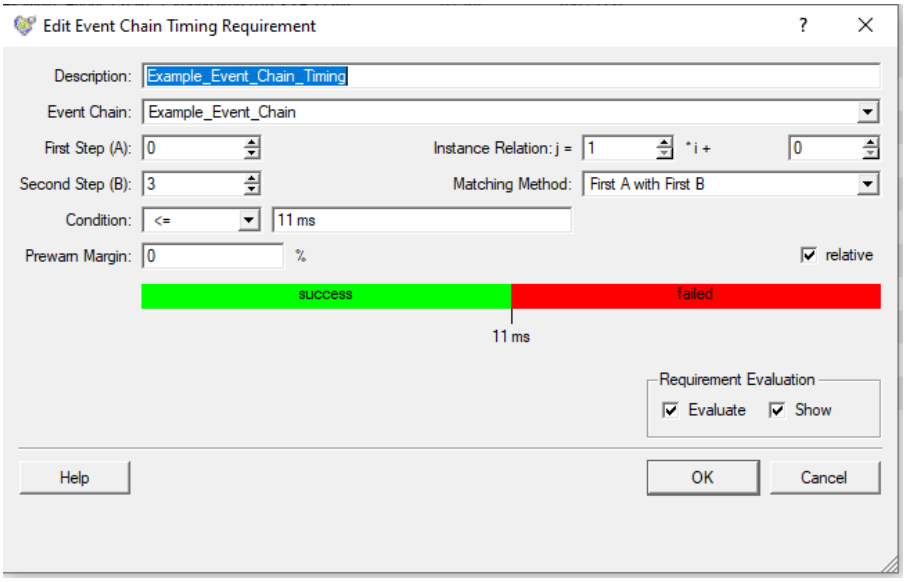
2. Add some requirements to the tasks:



3. Add a Sequence of Events:



4. Add a requirement to this sequence:



5. After some configuration of the view profile:
   - click the menu item '*Diagram Setting* → *Show/Hide Resources…*' and show all Tasks from 1ms to 100ms task
   - zoom the time axis until you see one complete event chain

Now you will have this view: