A Software Guide to

Universal Debug Engine

Debugging, Trace and Test for Embedded Systems

Integrated Development Environment
for 64-, 32-, 16-bit Microcontrollers and Embedded Processors

AURIX, TriCore, Arm Cortex-M/R/A, Arm7/9/11, S32G/S/V,
Stellar G/P/E, RH850, R-Car, RISC-V, ARC, Power Architecture

universal debug engine ®

pls Development Tools

This manual contains 189 pages.

Contact us at:

E-Mail: support@pls-mc.com
info@pls-mc.com
WWW: https://www.pls-mc.com

PLS Programmierbare Logik & Systeme GmbH
Technologiepark Lauta
DE-02991 Lauta
Germany
Phone: + 49 35722 384 - 0

PLS Development Tools
10080 N. Wolfe Rd., Suite SW3-200
Cupertino, CA 95014
USA
Phone: +1-949-863-0327
Toll Free: +1-877-77-DEBUG

PLS reserves the right to make technical changes to the equipment or changes to this document without any prior notice.

Contents

Introduction

Overview

Thank you for choosing **Universal Debug Engine (UDE) 2023**, one of the most powerful development workbenches available for the 64-bit architectures S32V234, the 32-bit architectures AURIX™, TriCore™, S32G, S32S, Power Architecture™, Cortex™-M/R/A, ARM™-7/9/11, RH850, R-Car, SuperH™ SH-2A, RISC-V, ARC and for the 16-bit architectures C166*, ST10*, XC166, XC2000, XE166, C166CBC, C166S V2 derivatives.

The software which you are about to install is the UDE Standard License software. Included with the full licensed version comes a high-speed communication hardware which speeds up downloading your application into the target system. It offers a flexible way of communication via various communication channels to the supported microcontroller.

Special versions of UDE like the **MemTool Flash/OTP Memory Programming Tool** are available on your request.

This **Software UDE manual** describes the **Universal Debug Engine** based on the selective Evaluation Boards. However, the UDE is also working with other AURIX, TriCore, Power Architecture, Cortex-M/R/A, ARM-7/9/11, RH850, R-Car, SH-2A, C166, ST10, XC166, XC2000, XE166, and XScale based hardware and simulators. Please see the compatibility list in appendix of this manual or the up-to-date list on our Web site for supported MCUs.

The Hardware Appendix UDE manual, an appendix of this manual, supplements this manual. Please see this manual for detailed description of the hardware interfaces.

You are invited to browse to our Web site at **https://www.pls-mc.com** to get the newest information or to download the latest version of **Universal Debug Engine® (UDE)**. Please check your **registration for the PLS Newsletter**, which informs you about latest UDE news and updated software version of the UDE. The PLS Newsletter is provided via e-mail and you can register it via your profile at https://www.pls-mc.com/accounts/profile/. In addition, the latest update information is available via the **About Box** of UDE.

* C16x/ST10-related products and services are provided only to existing customers with existing projects. C16x/ST10 support is not available for new projects.

Feedback

PLS welcomes feedback on our products and documentations. If you have any comments, suggestions or improvements about the products you are using, please use the Feedback Form from our Web Site **https://www.pls-mc.com**, send an e-mail to **support@pls-mc.com** or call our Support Line.

Safety Instructions for Products and Equipment

Warning! It is critical that you read and follow this safety advice, the product description including technical data and the associated technical documentation. Do not use the product if you cannot read and/or understand the Information for safe operation. If you do have questions for safe operation, please contact the PLS support at support@pls-mc.com.

This PLS product enables users to control systems which accomplish safety functions (e.g., in electronic control systems), to change safety relevant data, or to allocate those for further processing. Hence, the application of this product can be hazardous. Improper use and unskilled application without adequate instruction and experience in handling of such products may cause threats to life and physical conditions as well as damages to property.

Our products have been developed and released exclusively for use in applications defined in the product description.

Fitness and suitability of the products for any intended use beyond the utilization for which the products have been released (e. g. other stresses/strains or technical conditions) need to be verified by the user on his own authority by taking appropriate actions and measures (e. g. by means of tests).

- PLS products made available as **beta versions** of firmware, hardware and software are to be used exclusively in testing and evaluation. These products may have not sufficient technical documentation and may not fulfill all requirements for quality and accuracy for market released series products. Therefore, product performance may differ from the product description and your expectations. The product should be used only in controlled test environments. Do not use data and results from **beta versions** without prior and separate verification and validation and do not pass them to third parties without prior examination.
- Do not use this product if you do not have proper experience and training in using the product.
- Data of any kind, which have been identified or collected by using PLS products, have to be verified with respect to reliability, quality and suitability prior to any use or dissemination.
- When using this product with systems which accomplish safety functions (e.g., in electronic control systems) that influence system behavior and can affect the safe operation of the system, you must ensure that the system can be transitioned to a safe condition (e.g. emergency shutdown or emergency operation mode) if a malfunction or hazardous incident should occur.
- All applicable regulations and statutes regarding operation must be strictly followed when using this product
- It is recommended to use the products only in closed and designated test environment.

Warning! If you fail to follow this safety advice, there might be a risk of death, serious injury or property damage. PLS and their representatives shall not be liable for any damage or injury caused by improper use of the product. PLS provides trainings regarding the proper and intended use of this product.

Regulatory Compliance and Compliance Statements

The UADx hardware is in conformity with the protection requirements of the EU Council Directive EMC 89/336/EWG, EMC 2004/108/EC, EMC 2014/30/EC. The UADx hardware has been tested and found to comply with the limits for Class B Information Technology Equipment according to the European Standard EN 55022, EN 55024.

The UADx hardware complies with the relevant provisions of the RoHS Directive for the European Union.

Software

- Install the software only on systems which fulfill the minimum requirements both in hard- and software.
- For installation of the software administrator rights are required to copy files in directories which are protected by the Windows OS, to install device drivers and modify the registry.
- The software enables the in-depth control of embedded systems. It should only be operated by persons who have the necessary expertise in the systems.
- Incorrect usage of the software can lead to irreparable destruction of components in the connected systems. This concerns in particular components whose integrated permanent memory (e.g. FLASH, PCM) is protected by special mechanisms.
- There is a particular danger if mechanical devices such as motors or actuators are controlled by the embedded systems. In this case, all necessary precautions must be taken to avoid accidents, e.g., emergency shutdown.
- There is also a particular danger if the embedded systems switch voltages that exceed the permissible contact voltages. In this case, all precautions must be taken to avoid accidents, e.g. insulation.

Electrical Safety Instructions

The Universal Debug Engine (UDE) shall only be used according to the installing instructions of the **UDE Manual** and **UDE Manual Appendix**. Any external power supply used with the Universal Access Device (UAD2^{pro}, UAD2^{+}, UAD2^{next}, UAD3^{+} ...) and its components shall comply with the relevant regulations and standards applicable in the country of intended use.

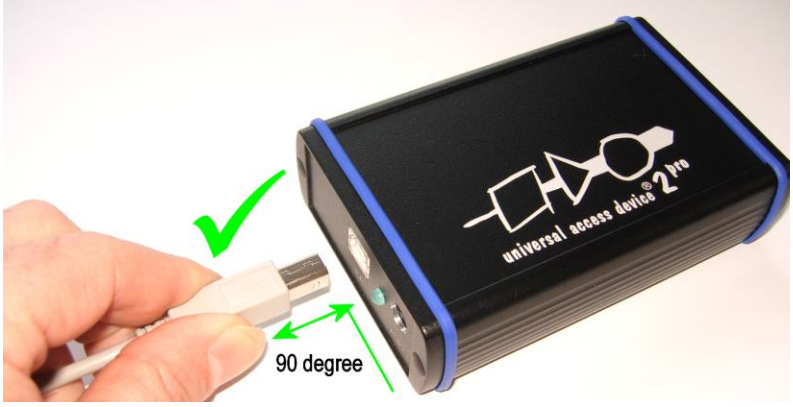
Please observe the following safety instructions when using the power supply:

- Always use the supplied power adapter, and connect it to an AC outlet of the rated voltage and frequency. If an AC adapter other than those specified by PLS is used, it may result in damage to the UADx and its accessories or AC adapter, fire or electric shock.
- Do not insert or disconnect the AC plug with wet hands. Doing so may cause electric shock.
- Insert the power plug fully and securely. Incomplete insertion may cause fire or electric shock.
- The power supply unit should be connected to an easily accessible socket outlet in the immediate vicinity of the unit.
- Always disconnect the power cord by holding the power plug. Pulling the power cord itself may damage it and cause fire or electric shock.
- Ensure that the device connections do not come into contact with liquids and do not touch them with wet or greasy hands or metal objects. If liquid gets into the device, stop using the device immediately and contact support@pls-mc.com.
- Do not store the devices in environments with high humidity or where the temperature may change suddenly. If condensation has formed, switch the devices off immediately and wait until all water drops have evaporated.
- Do not pour liquid substances over the UADx and its accessories or drop other objects on it, this could cause serious damage to the UADx and its components. If this should happen please stop all work with the UADx and its accessories immediately and contact support@pls-mc.com.
- Do not disassemble or attempt to repair the equipment. If a device is damaged, stop using the device immediately and contact support@pls-mc.com. Do not touch damaged areas. Avoid contact with eventually spilled liquids.

- If the UADx and its accessories is visibly damaged or its functionality is limited, it must not be used without prior instruction from support staff (support@pls-mc.com). Especially if components are damaged where voltage is flowing through them. These must be replaced by the manufacturer in order to avoid hazards.
- Unplug the power cord from the wall outlet during a thunderstorm or prolonged absence! Otherwise, damage to the unit could be caused by overvoltage.

Mechanical Safety Instructions

- Hold the head of the USB cable with your index finger and thumb on both sides and **insert** the cable straight into the USB port as shown in the illustration below. Make sure that you insert it straight and not at an angle.
- Hold both sides of the USB cable with your index finger and thumb at the point where it is connected to the computer and carefully **pull it out** horizontally to remove the cable from the USB port.



- Do not insert or remove a USB plug with excessive force.

- Do not plug in or pull out the USB plug upwards, downwards, left, right or forwards.
- Do not pull or tug on the USB cable when plugged into the port.

Safety Instructions

- Do not use the Universal Access Device (UAD2^{pro}, UAD2^{+}, UAD2^{next}, UAD3^{+} ...) and its accessories in places where flammable or combustible gases (gasoline etc.) are present. Doing so may cause a fire.
- The UADx and its components should be operated in a well-ventilated environment and should not be covered. The UADx and its accessories are only intended for use inside buildings.
- The UADx and its components should be placed on a stable, flat surface in use.
- Do not use excessive force when using the equipment. Do not pull on cables or bend them too much.
- Do not expose the devices to fire, microwaves or high temperatures.
- The UADx and its accessories must not be operated if it is damaged, or if smoke or odd smells occur. Doing so may result in a fire. In such situations, disconnect the power adapter from the AC outlet, and contact support@pls-mc.com.
- Make sure that the UADx and its accessories is stored at ground level and in a position that does not endanger persons and surrounding equipment.
- Do not place the UADx and its accessories on an unstable or sloping surface. Doing so may result in its dropping or overturning, causing injury. Be careful not to drop the UADx and its accessories when carrying it.
- Before cleaning, remove all connected cables to avoid the risk of electric shock. Clean the outside of the devices only, using a soft, damp cloth. Do not use chemicals or abrasives. Avoid under all circumstances the penetration of moisture into the device.
- The use of spare parts, accessories and special equipment which have not been tested and approved by PLS can have a negative influence on the function and properties of the UADx and its components. Therefore, PLS is not liable for any resulting damage.
- Improper operation of the UADx and its accessories may cause damage to the devices or other property. It may therefore only be used in technically perfect condition and for its intended purpose in accordance with the operating instructions given in the manual.
- Safe use of UADx and its accessories is only possible if the user manual is read completely and the instructions are followed completely. Non-observance of the instructions can lead to considerable damage or accidents.
- Anyone using UADx and its accessories must have access to the user manual. The user manual can be found here: in the delivery content of the UDE as printed manual, UDE Software installation as PDF.
- **Keep these operating instructions in a safe place for later use.**
- The product may only be used by persons instructed in the safe use of the product and understand the resulting dangers.
- Children should be supervised to ensure that they do not play with the UADx and its components.
- Keep the devices, all accessories and packaging materials out of reach of children and pets. Small objects such as the packaging materials could be accidentally swallowed. Cables could be tied around the neck.

Versions of UDE

UDE, the **Universal Debug Engine** for AURIX, TriCore, Power Architecture, Cortex, ARM7, ARM9, ARM11, C166*, ST10*, XC166, XC2000, XE166, XScale, RH850, R-Car and SuperH SH-2A derivatives is available in several high-speed full-featured versions with extra communication hardware. Versions with simulator support are available too.

The following table describes the differences between these versions of UDE.

The **Standard versions** are equipped with specialized communication hardware and allow high transmission rates. Additional various communication channels are featured.

1. The **Standard version UAD2^{pro}** allows a flexible way of communication and is suitable for desktop and notebook users in the same way. The UAD2^{pro} communicates with the host PC via the USB2.0 bus. Target communication is supported via ASC, SSC, CAN, JTAG, cJTAG, DAP, SWD.
2. The **Standard version UAD2^{+}** is the **all-in-one solution** for UDE. It communicates with the host PC via the USB2.0, IEEE1394 or Ethernet bus. Target communication is supported via ASC, SSC, 3Pin, CAN, JTAG, DAP, SWD. Additionally, the UAD2^{+} supports the program instruction trace with the Trace Board add-in feature.

   Please note the UAD2^{next} supersedes the UAD2^{+}. For new projects, the UAD2^{+} is no longer available. Of course, all existing UDE/UAD2^{+} licenses will be maintained continuously for the next years without limitations.

3. The **Standard version UAD2^{next}** is the next generation of the **all-in-one solution** for UDE. It communicates with the host PC via the USB3.0 or Gigabit Ethernet bus. Target communication is supported via ASC, CAN, JTAG, DAP, SWD. Additionally, the UAD2^{next} supports parallel and serial trace with the Trace Board add-in feature.
4. The **Standard version UAD3^{+}** is the **high-end-solution** for UDE. It communicates with the host PC via the USB2.0, IEEE1394b or Gigabit Ethernet bus. Target communication is supported via JTAG, cJTAG, DAP, SWD. Additionally, the UAD2^{+} supports the high-speed parallel and serial trace with the Trace Board add-in feature.

The **Simulator version** features a debugging environment via various simulators. The simulator can be used effectively in the early stages of software development, reducing the length of time spent later on system integration. The UDE debugger uses the simulator interface like a real hardware platform. All advantages of HLL-Debugging are offered with the combination of UDE and the simulator environment.

* C16x/ST10-related products and services are provided only to existing customers with existing projects. C16x/ST10 support is not available for new projects.

Demo versions of UDE are also available. These versions have restrictions in visualization functions at runtime, PCP assembler support, MCDS and trace capabilities (if available), and Script support. All other functions are available for testing. Such Demo versions are not described in this manual!
Please see the separate manual for more information or contact the PLS Support Team.

Delivery Contents

Depending on your UDE version, please check the delivery contents from the following table and make sure that the package contains all of the required parts.

	Standard version **UAD2^{pro}**	Standard version **UAD2+** *	Standard version **UAD2^{next}**	Standard version **UAD2^{next}** with **Trace Option**	Standard version **UAD3+**	Standard version **UAD3+** with **Trace Board**	**Simulator** version
Software and Manuals							
UDE Quick Reference Guide	✓	✓	✓	✓	✓	✓	✓
CD-ROM	✓	✓	✓	✓	✓	✓	✓
Access Devices							
UAD2^{pro}	✓						
UAD2+		✓					
UAD2+ Debug Extender with 10-wire (DAP), or 14-wire (JTAG/cJTAG), or 16-wire (JTAG), 20-wire (JTAG)		✓[3]					
UAD2^{next}			✓	✓			
UAD2^{next} Trace Module				✓			
UAD3^{+}					✓	✓	
UAD3^{+} Debug Pod					✓	✓	
UAD3^{+} Trace Board 2 (built-in)						✓[4]	
UAD3^{+} Parallel Trace Pod						✓[3]	
UAD3^{+} Aurora Trace Pod						✓[3]	
USB-Key							✓
Interface Devices							
UAD2 Debug Adapter (ARM, DAP, SWD, OnCE, COP, ..)		✓[3]					
UAD2^{pro}/UAD2^{next}/UAD3^{+} Debug Adapter (ARM, DAP, SWD, OnCE, COP, ..)	✓[3]		✓[3]	✓[3]	✓[3]	✓[3]	
UAD2^{next} /UAD3^{+} Trace Adapter (ETM, NEXUS, Aurora, ..)				✓[4]		✓[4]	

* Please note, the UAD2^{next} supersedes the UAD2^{+}. For new projects, the UAD2^{+} is no longer available. Of course, all existing UDE/UAD2^{+} licenses will be maintained continuously for the next years without limitations.

	Standard version **UAD2^{pro}**	Standard version **UAD2+**	Standard version **UAD2^{next}**	Standard version **UAD2^{next}** with **Trace Option**	Standard version **UAD3+**	Standard version **UAD3+** with **Trace Board**	**Simulator** version
Pod Interface Cables							
UAD2^{+} Debug Extender Cable (40-wire HD Flat Ribbon Cable)		✓[3]					
UAD3^{+} Debug Pod Cable (26-wire HD Cable)					✓	✓	
UAD3^{+} Trace Pod Cable (68-wire HD Cable)						✓	
Target Interface Cables							
UAD2 10" (25 cm) Flat Ribbon Cable, 10-wire, 3Pin, ASC, SSC		✓					
UAD2^{pro}/UAD2^{next}/UAD3+ Debug Adapter Cable (40-wire HD Flat Ribbon Cable)	✓		✓	✓	✓	✓	
UAD2^{next}/UAD3+ Parallel Trace Adapter Cable (38-wire HD Flat Ribbon Cable)				✓[3]		✓[3]	
UAD3^{+} HSS22 MCDS Trace Adapter + Cable (24cm flex cable)				✓[3]		✓[3]	
UAD3^{+} HSS34 NEXUS Trace Adapter + Cable (24cm flex cable)				✓[3]		✓[3]	
Host PC Adapters and Cables							
Host IEEE1394 Cable		✓[3]					
Host IEEE1394b Cable		✓[3]			✓[3]	✓[3]	
Host USB Cable	✓	✓	✓	✓	✓	✓	
Power Supply							
Wall Transformer 220 VAC 50~ / 12V DC (6W)	✓[5]						
Wall Transformer 110-220VAC 50-60~ / 12V DC or 18V DC (24W)	✓[5]	✓	✓	✓			
Wall Transformer 110-220VAC 50-60~ / 12V DC or 18V DC (100W)					✓	✓	

✓[3] optional shipped, corresponding your order ✓[4] with Trace Board option only ✓[5] depending on your country location

System Requirements

To run **Universal Debug Engine (UDE)** at least the following minimum system configuration is required:

	Minimum	Recommended
CPU	Intel or AMD x86_64 (64-bit) processor	Intel Core i7™ or AMD R7 processor
RAM	4 GByte	8 GByte
Free disk space	2 GByte HDD	8 GByte SSD
Display	SXGA	WUXGA
Operating System	Windows®10 64-bit or Windows®11 64-bit	Windows®10 64-bit or Windows®11 64-bit

Dependencies

- Microsoft Visual C++ 2015-2022 Redistributable (x64)

Note: Installations packages of these components are include with and installed by UDE setup package. However to avoid side effects on other applications these components are not uninstalled when UDE is uninstalled.

Additional requirements

- Optional: CD-, DVD- or BD-drive for installation from CD-ROM
- Microsoft .NET™ Framework 4
- Administrator permissions for the current login during installation
- Up to 200 GByte disk space (SSD recommended) may be required by features based on Aurora Trace.

Depending on the type of target access, you will additionally need one of the following interface ports:

- an USB port interface for the Standard version with UAD2^{pro} or UAD2^{+} or UAD2^{next} or UAD3^{+} or for the Demo version for Easy Kits XC166, XC2000, XE166 or for the Simulator version with USB-Key or the Standard version with USB-JTAG-Port
- or an Ethernet interface for the Standard version UAD2^{+} or UAD2^{next} or UAD3^{+}
- or an IEEE1394-OHCI interface for the Standard version with UAD2^{+} or UAD3^{+} via IEEE1394 or an IEEE1394b-OHCI interface for the Standard version with UAD3^{+} via IEEE1394b.

The UDE Eclipse Integration Package **Eclipse 4.x UDEEclipse4Integration** requires following further environment:

- 64-bit version of Java JRE 8, JRE 11 or higher
- 64-bit version of Eclipse 4.8 - 4.27 or newer and the appropriate CDT package.

Installing of Universal Debug Engine

Before you start the installation process, please ensure which version of **Universal Debug Engine (UDE)** you intend to use. Check that your package contains the required hardware parts and install the hardware and software according to the following description.

Depending on your UDE version, please follow the corresponding instructions.

⚠ Please note that you must have administrator rights for successful execution of the installation process.

Installation Notes - Before you install UDE

Proper functioning of the Universal Debug Engine and its hardware devices is only guaranteed for working with the original components tested and delivered by PLS. All delivered components have been verified based on the recommendations and standards of the chip manufacturers.

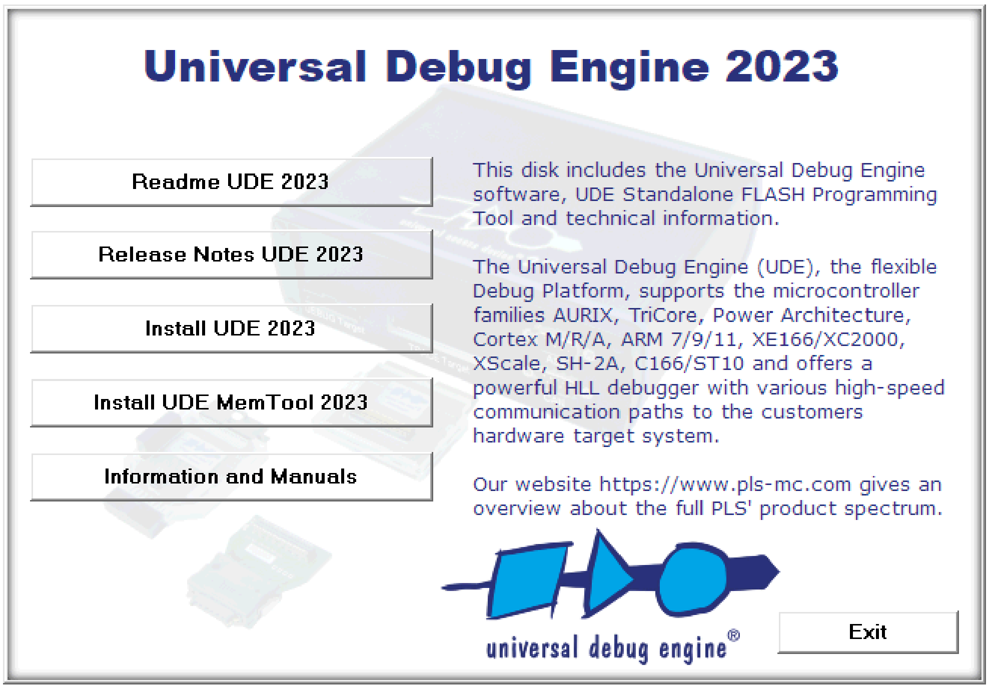
⚠ Please note the hints of **Safety Precautions** in the **UDE Manual Hardware Appendix** first.

⚠ When starting a newly installed version of UDE for the first time, a **firmware update** may be executed for the access device (UAD2$^{\text{pro}}$, UAD2^{+}, UAD2$^{\text{next}}$, and UAD3^{+}). This may take some more time than usual for the “target connect” operation. Please **DO NOT** power OFF or unplug the access device during this period!

Installing UDE Software

Working with the CD browser

The CD browser from the UDE CD-ROM helps you to install the UDE software.



Universal Debug Engine 2023

This disk includes the Universal Debug Engine software, UDE Standalone FLASH Programming Tool and technical information.

The Universal Debug Engine (UDE), the flexible Debug Platform, supports the microcontroller families AURIX, TriCore, Power Architecture, Cortex M/R/A, ARM 7/9/11, XE166/XC2000, XScale, SH-2A, C166/ST10 and offers a powerful HLL debugger with various high-speed communication paths to the customers hardware target system.

Our website https://www.pls-mc.com gives an overview about the full PLS' product spectrum.

Please insert the Universal Debug Engine CD-ROM. Please start the `Setup.exe` from the CD-ROM root directory manually.

The UDE CD-ROM contains following chapters:

- **Readme UDE 2023** – Contains important information about the current UDE installation and requirements. Please read it first.
- **Install UDE 2023** – Start the installation process of the full Universal Debug Engine version. Valid license keys are required.
- **Install UDE MemTool 2023** – Start the installation process of the full UDE MemTool version. Valid license keys are required.
- **Install UDE Samples 2023** – Start the installation process of the UDE's Samples. It contains code samples for supported compilers and target evaluation boards.
- **Information and Manuals** – Find out more about all of the UDE products.

Download the latest UDE Release from Website

Alternatively check the PLS' website for the latest UDE version. If you are here for the first time, create a new registration by opening the link **https://www.pls-mc.com/accounts/signup/** before and filling out the form. Do not forget to enter your **company e-mail address** and the **serial number** of your UAD device!

Full access to the UDE and UDE MemTool download area is only granted if a serial number exists. Otherwise access is limited to general information.

⚠ Your registration request will be processed directly by our **Support Team** and you **will be informed by e-mail about the progress of the registration process**. In the meantime, please **log out until** you receive the **welcome e-mail notification**. If you recently confirmed your registration via the e-mail link and have not yet received the Welcome to our Community e-mail, please wait for this information. Your login will not work until then.

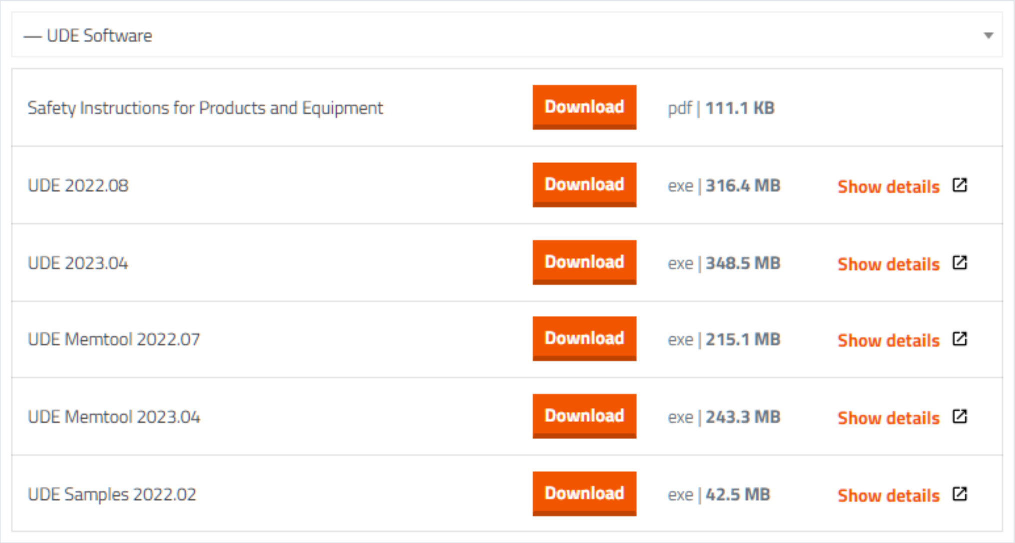
Login at **https://www.pls-mc.com/accounts/login/** and open

- **Download Latest UDE version** at **https://www.pls-mc.com/download.htm** (single file download) or
- **Service – Downloads and Updates** at **https://www.pls-mc.com/service/downloads/** (download all).

As a customer with a valid maintenance contract, you have access to

UDE Software

- Safety Instructions for Products and Equipment
- Current **UDE** version (and Legacy UDE versions)
- Current **UDE MemTool** version (and Legacy UDE MemTool versions)
- UDE Samples



— UDE Software

Safety Instructions for Products and Equipment	Download	pdf | 111.1 KB	
UDE 2022.08	Download	exe | 316.4 MB	Show details
UDE 2023.04	Download	exe | 348.5 MB	Show details
UDE Memtool 2022.07	Download	exe | 215.1 MB	Show details
UDE Memtool 2023.04	Download	exe | 243.3 MB	Show details
UDE Samples 2022.02	Download	exe | 42.5 MB	Show details

UDE Device Driver Software

- Drivers for legacy UDE versions, **not required for current UDE versions!**

Product Information and Manuals

- Product Information
- UDE Manual, UDE Manual Appendix, UDE MemTool Manual
- Application Notes.

Please note that only either UDE **or** UDE MemTool of a major version can be installed and used at the same time. UDE already contains the functionality of UDE MemTool.

In the case of questions contact the PLS support at support@pls-mc.com.

Start the UDE Installation

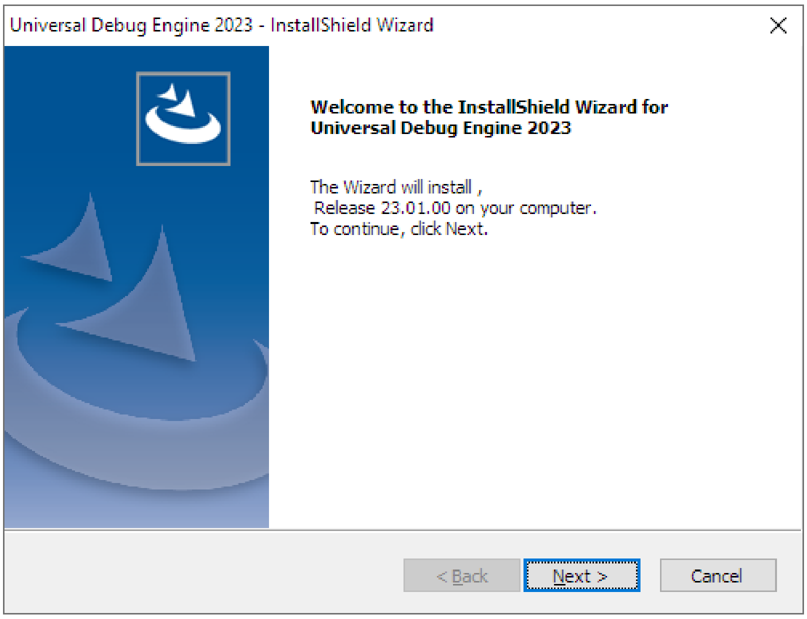
When a newly installed version of UDE is started for the first time, a **firmware update** may be executed for the access device (UAD2^{pro}, UAD2^{+}, UAD2^{next}, and UAD3^{+}). This may take cause the “target connect” operation to take longer than usual. Please **DO NOT** power off or unplug the access device during this time!

Please note that you must have **administrator rights** in order to successfully complete the installation process.

The UAD driver software is also installed as part of the installation process. The drivers are signed by PLS, and it is necessary to trust this signature. During the driver installation, the message box **“Windows Security: Would you like to install this device software? Name: pls Development Tools. Publisher: pls Programmierbare Logik & Systeme GmbH”** will appear. To trust and install the PLS driver software, click the Install button.

Before you can use the Universal Debug Engine, you must install the software components.

- Using the **UDE CD-ROM** start `Setup.exe` from the UDE CD-ROM
- Using the **UDE Download** start the downloaded file, e.g. `ude-2023-04.exe`.



Universal Debug Engine 2023 - InstallShield Wizard

Welcome to the InstallShield Wizard for Universal Debug Engine 2023

The Wizard will install ,
Release 23.01.00 on your computer.
To continue, click Next.

3. Click **Next** to continue the installing process or click **Cancel** for aborting.
4. Check the license terms and your customer information and click **Next** to continue.
5. Optionally, use the **Browse** button to select a different installation location. Please specify an empty or new directory for the UDE software. Click **Next**.
6. Select the Program Folder and click **Next** to continue.
7. Click **Next** to continue the installation process.

Installing Hardware

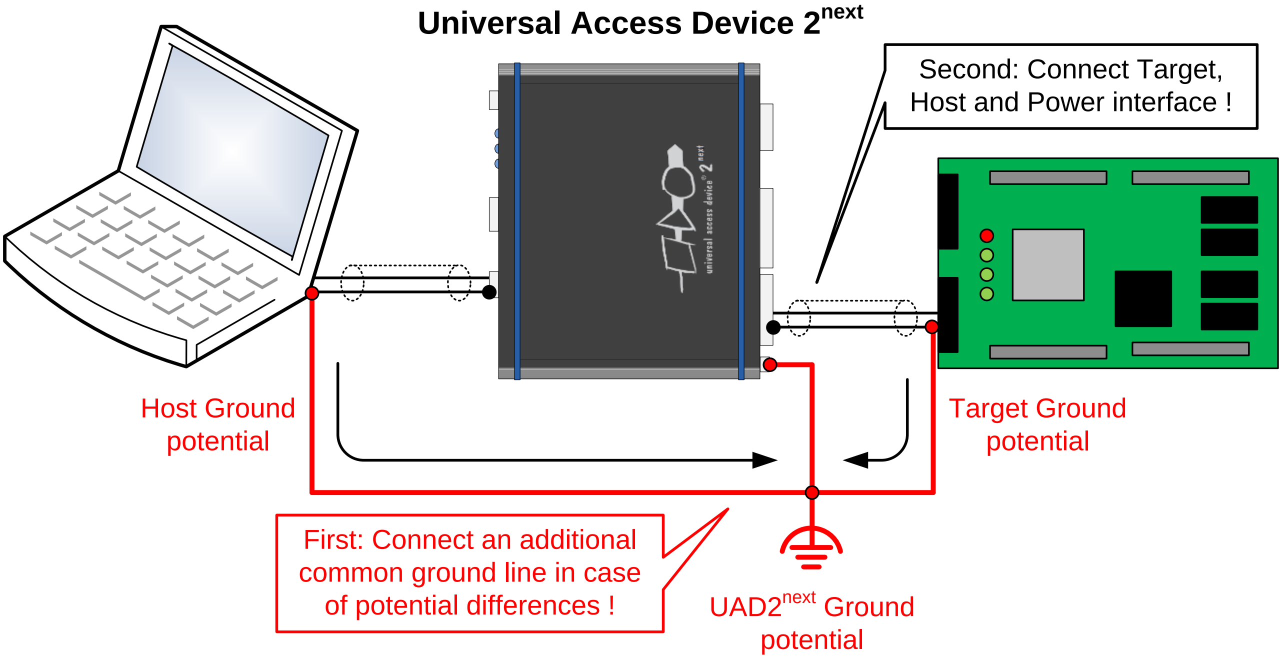
Static Electricity Precautions

Electrostatic Discharge (ESD) can damage a sensitive electronic component! Under several conditions static electricity and ground potential differences between the Access Device and the user's target hardware can build up high voltages - over 10000 Volts (10 kV) in some cases. The electrostatic discharge of this build-up voltage results in fast high current waveforms and fast magnetic (H-field) or electrostatic (E-field) disturbances. The discharge into the electronic components and circuitry can damage or destroy hardware components, resulting in failures and reduced reliability.

Because of the **non-hot-pluggable** 1.65 Volts / 5.0 Volts properties of the **JTAG/DAP/SWD** connectors, these ports are endangered especially by electrostatic discharging. The maximum voltage on these pins must not exceeded 5.5 Volts against the UAD's ground, especially in the case that the ground planes are not connected first.

To protect your hardware against damage from static electricity and ground potential discharge, you have to follow some basic precautions:

1. Before you change any cable connections from the Access Device, please **remove the power** from the Access Device and your target system.
2. Please ensure that the **static electricity** and **ground potentials** between the Access Device, the host PC and the target hardware are **balanced**. If there is a danger of high potential differences, you must connect the Access Device, the host PC and the target hardware to the same ground potential by **a low resistance connection**.
3. Establish the target connection and **power ON** the systems.


Universal Access Device 2^{next}
universal access device® 2 next
Second: Connect Target, Host and Power interface !
Host Ground potential
Target Ground potential
First: Connect an additional common ground line in case of potential differences !
UAD2^{next} Ground potential

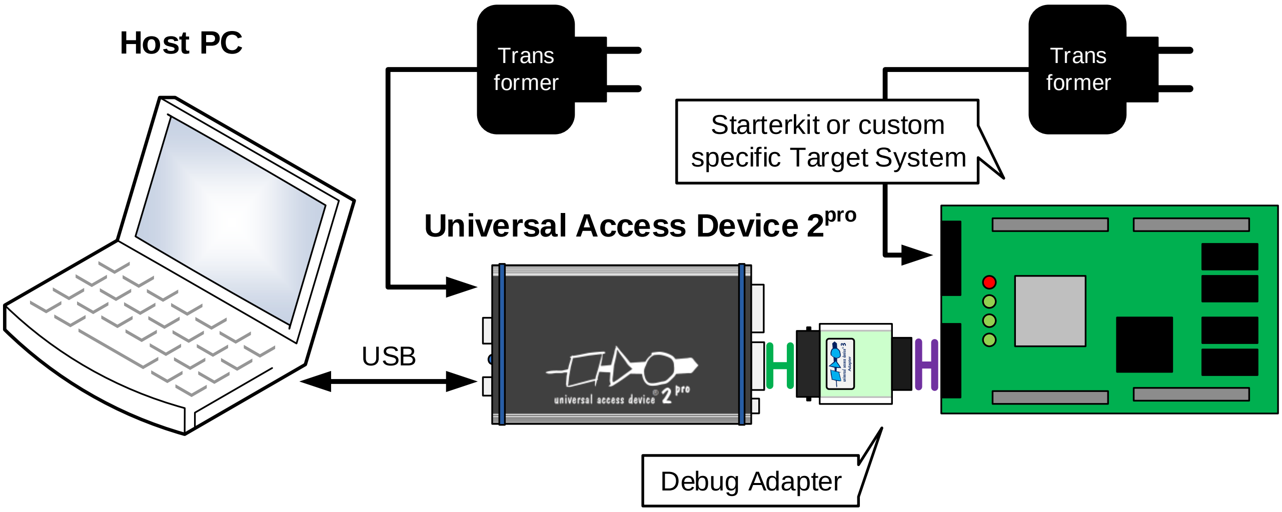
Hint: All Universal Access Devices are equipped with a ground socket on the front side. Please use this ground socket for discharging the static electricity and balancing ground potentials between the Universal Access Device, the host PC and the target hardware **BEFORE** you connect the target hardware to the Access Device.

Note: The next step depends on your selection of the target access and UDE version respectively. The following step describes what you have to do for each target communication channel.

Standard Version UAD2pro (via JTAG/DAP/SWD)

Install the hardware of UAD2pro as follows:

1. Connect the UAD2pro (connector 'USB') with an USB connector of the installed USB host adapter in your PC (USB 2.0 is recommended).
2. Connect the ground socket with the ground potential of your target hardware for discharging the static electricity and balancing ground potentials.
3. Connect the UAD2pro with the JTAG connector by the 40-pin cable and delivered adapter. Refer to the appendix **Hardware Description** of the user's manual for more information about the correct pin connections.


Host PC
Trans former
Trans former
Starterkit or custom specific Target System
Universal Access Device 2pro
USB
universal access device 2 pro
Debug Adapter

4. Connect the wall plug transformer's cable with the 'Power' jack to the UAD2pro and power on the system.
5. The driver software was installed during the previous software installation. After initialization, the UAD2pro is connected to the host system and ready. A detailed description of the driver installation can be found in chapter **Driver Installation for Universal Access Device**.

The UDE workbench is now installed and ready to use.

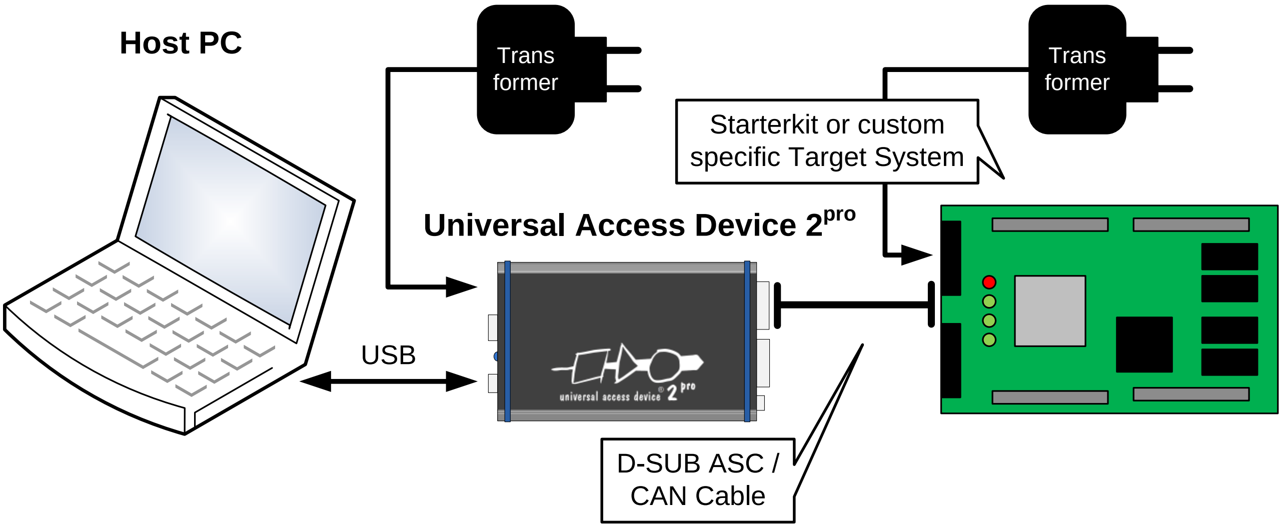
If you encounter difficulties installing the product, please contact the PLS Support Team at **support@pls-mc.com**.

Please note the important information about required ESD protection of the access devices in chapter **Static Electricity Precautions**!

Standard Version UAD2^{pro} (via ASC, CAN)

Install the hardware of UAD2^{pro} as follows:

1. Connect the UAD2^{pro} (connector 'USB') with an USB connector of the installed USB host adapter in your PC (USB 2.0 is recommended, USB 1.1 is possible).
2. Connect the ground socket with the ground potential of your target hardware for discharging the static electricity and balancing ground potentials.
3. Connect the 'ASC/CAN Target' connector of the UAD2^{pro} with the ASC (RS232) or CAN connector of the microcontroller board by a 9-pin 1:1 SUB-D9 (M) to SUB-D9 (F) extension cable. Refer to the appendix **Hardware Description** of the user's manual for more information about the correct pin connections.


Host PC
Trans former
Trans former
Starterkit or custom specific Target System
Universal Access Device 2pro
USB
universal access device 2 pro
D-SUB ASC / CAN Cable

4. Connect the wall plug transformer's cable with the 'Power' jack to the UAD2^{pro} and power on the system.
5. The driver software was installed during the previous software installation. After initialization, the UAD2^{pro} is connected to the host system and ready. A detailed description of the driver installation can be found in chapter **Driver Installation for Universal Access Device**.

The UDE workbench is now installed and ready to use.

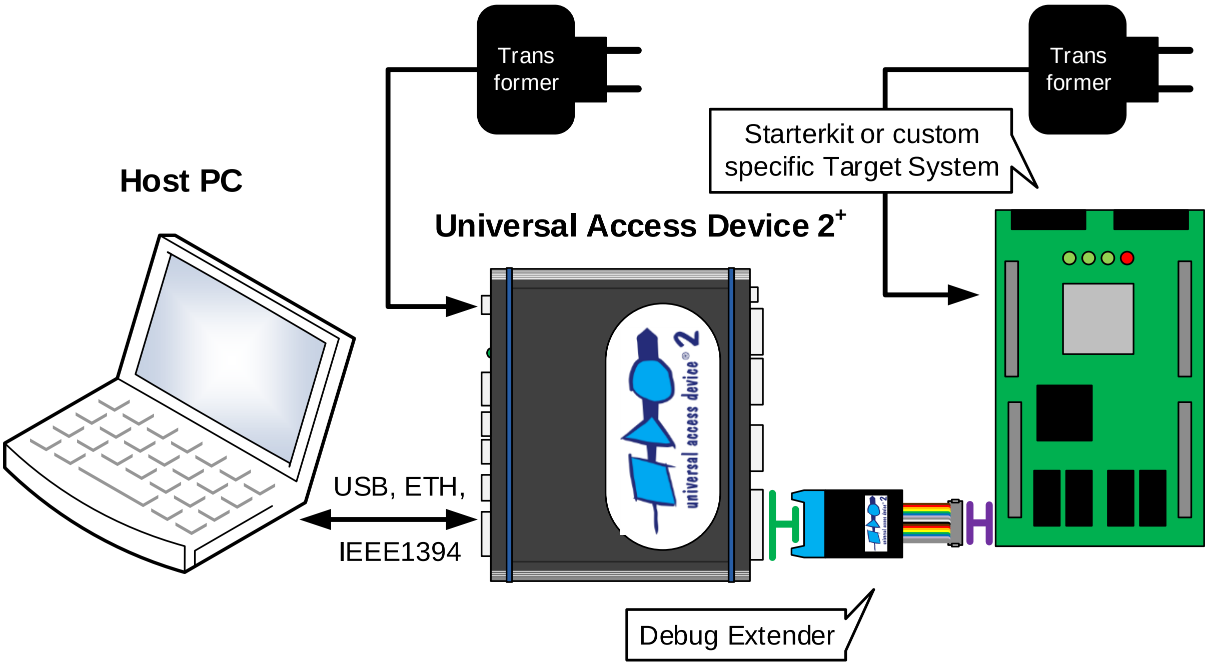
If you encounter difficulties installing the product, please contact the PLS Support Team at **support@pls-mc.com**.

Please note the important information about required ESD protection of the access devices in chapter **Static Electricity Precautions**!

Standard Version UAD2+ (via JTAG/DAP/SWD)

Install the hardware of UAD2+ as follows:

1. Connect the UAD2+ (connector 'IEEE1394') with an IEEE1394 connector of the installed IEEE1394 host adapter in your PC **or** connect the connector 'USB 2.0' with an USB connector of the installed USB host adapter in your PC (USB 2.0 is recommended) **or** the connector ETH with an Ethernet network.
2. Connect the ground socket with the ground potential of your target hardware for discharging the static electricity and balancing ground potentials.
3. Connect the 'JTAG/OCDS Target' connector of the UAD2+ with the Debug Extender via the 40-wire HD flat ribbon cable. Connect the Debug Extender with your target by the 14-wire, 16-wire or 20-wire JTAG cable. Refer to the appendix **Hardware Description** of the user's manual for more information about the correct pin connections.


Trans former
Trans former
Starterkit or custom specific Target System
Host PC
Universal Access Device 2+
universal access device 2
USB, ETH,
IEEE1394
Debug Extender

4. Connect the wall plug transformer's cable with the 'Power' jack to the UAD2+ and power on the system.
5. The driver software was installed during the previous software installation. After initialization, the UAD2+ is connected to the host system and ready. A detailed description of the driver installation can be found in chapter **Driver Installation for Universal Access Device**.

The UDE workbench is now installed and ready to use.

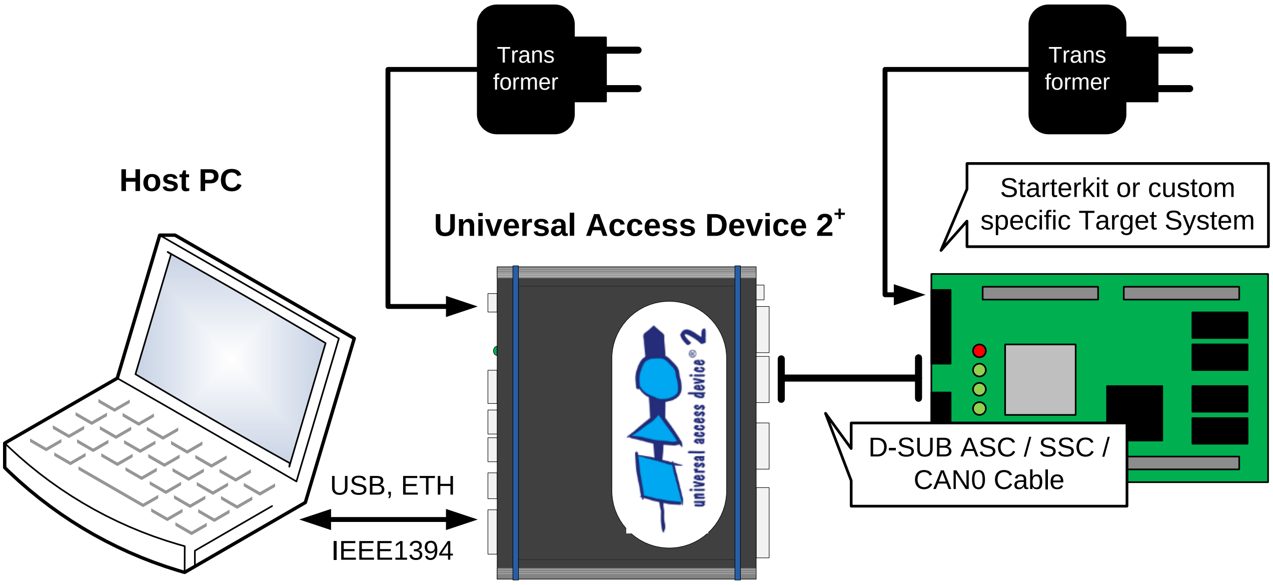
If you encounter difficulties installing the product, please contact the PLS Support Team at **support@pls-mc.com**.

Please note the important information about required ESD protection of the access devices in chapter **Static Electricity Precautions**!

Standard Version UAD2$^+$ (via ASC, SSC, CAN)

Install the hardware of UAD2$^+$ as follows:

1. Connect the UAD2$^+$ (connector 'IEEE1394') with an IEEE1394 connector of the installed IEEE1394 host adapter in your PC **or** connector 'USB 2.0' with an USB connector **or** the connector ETH with an Ethernet network.
2. Connect the ground socket with the ground potential of your target hardware for discharging the static electricity and balancing ground potentials.
3. For Bootstrap loading via ASC (RS232), connect the 'ASC/SSC/CAN0 Target' connector with the target connector via a 9-pin D-SUB extension cable. In case of ASC (TTL) or SSC (TTL), connect the 'ASC/SSC/3PIN Target' connector with the ASC or SSC hardware pins of the microcontroller board by the 10-wire flat ribbon extension cable. If CAN is used, additionally connect the 'CAN1 Target' connector with the target via a 9-pin D-SUB extension cable. Refer to the appendix **Hardware Description** of the user manual for more information about the correct pin connections of the ASC, SSC and CAN interfaces.


Trans former
Trans former
Host PC
Universal Access Device 2+
Starterkit or custom specific Target System
universal access device 2
D-SUB ASC / SSC / CAN0 Cable
USB, ETH
IEEE1394

4. Connect the wall plug transformer's cable with the 'Power' jack to the UAD2$^+$ and power on the system.
5. The driver software was installed during the previous software installation. After initialization, the UAD2$^+$ is connected to the host system and ready. A detailed description of the driver installation can be found in chapter **Driver Installation for Universal Access Device**.

The UDE workbench is now installed and ready to use.

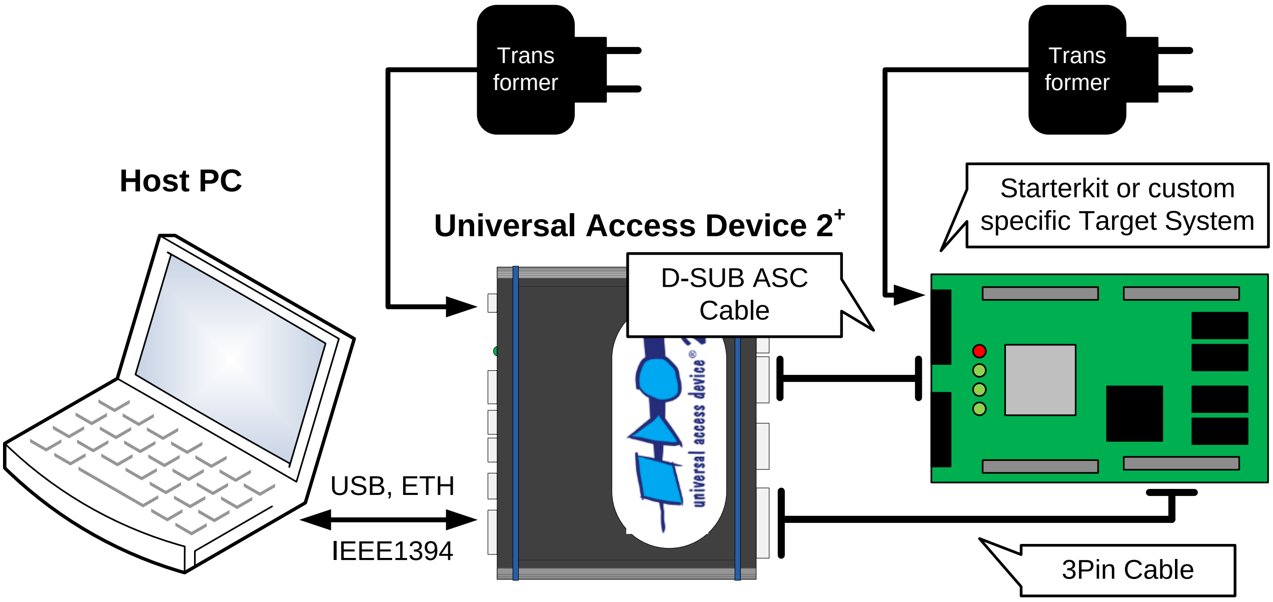
If you encounter difficulties installing the product, please contact the PLS Support Team at **support@pls-mc.com**.

Please note the important information about required ESD protection of the access devices in chapter **Static Electricity Precautions**!

Standard Version UAD2+ (via 3Pin)

Install the hardware of Universal Access Device2+ as follows:

1. Connect the UAD2+ (connector 'IEEE1394') with an IEEE1394 connector of the installed IEEE1394 host adapter in your PC **or** connector 'USB 2.0' with an USB connector **or** the connector ETH with an Ethernet network.
2. Connect the ground socket with the ground potential of your target hardware for discharging the static electricity and balancing ground potentials.
3. Connect the 'ASC/SSC/3Pin Target' connector of the UAD2+ with the 3Pin connector of the microcontroller board by the 10-wire 3Pin cable. Because the 3Pin connector is not standardized, it must be installed on your target hardware subsequently. For Bootstrap loading ASC (RS232)/3Pin solution, you must connect the ‘ASC Target’ connector with the ASC0 target connector. Refer to the appendix **Hardware Description** of the user’s manual for more information about the correct pin connections of the 3Pin interface.



4. Connect the wall plug transformer's cable with the 'Power' jack to the UAD2+ and power on the system.
5. The driver software was installed during the previous software installation. After initialization, the UAD2+ is connected to the host system and ready. A detailed description of the driver installation can be found in chapter **Driver Installation for Universal Access Device**.

The UDE workbench is now installed and ready to use.

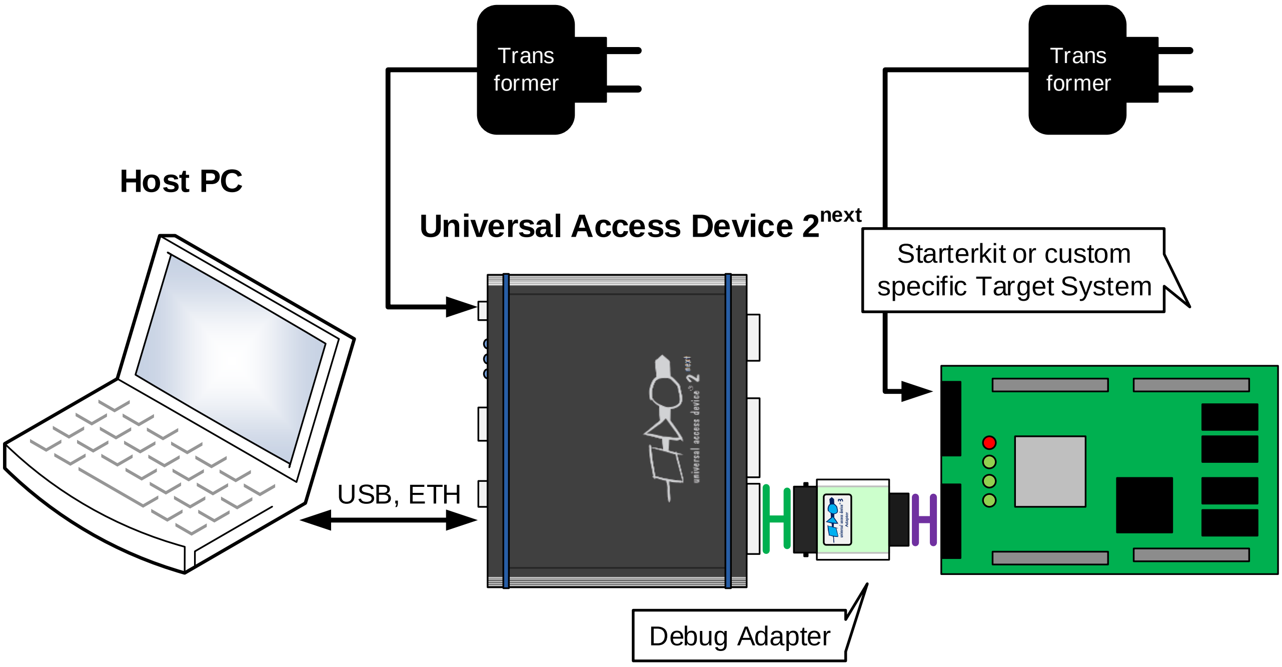
If you encounter difficulties installing the product, please contact the PLS Support Team at support@pls-mc.com.

Please note the important information about required ESD protection of the access devices in chapter **Static Electricity Precautions**!

Standard Version UAD2^{next} (via JTAG/DAP/SWD)

Install the hardware of UAD2^{next} as follows:

1. Connect the UAD2^{next} (connector 'USB 3.0') with an USB connector of the installed USB host adapter in your PC (USB 3.0 is recommended) **or** the connector ‘Ethernet’ with an Ethernet network.
2. Connect the ground socket with the ground potential of your target hardware for discharging the static electricity and balancing ground potentials.
3. Connect the UAD2^{next} with the JTAG connector by the 40-pin cable and delivered Debug Adapter. Refer to the appendix **Hardware Description** of the user’s manual for more information about the correct pin connections.


Trans former
Trans former
Host PC
Universal Access Device 2next
Starterkit or custom specific Target System
USB, ETH
Debug Adapter

4. Connect the wall plug transformer's cable with the 'Power' jack to the UAD2^{next} and power on the system.
5. The driver software was installed during the previous software installation. After initialization, the UAD2^{next} is connected to the host system and ready. A detailed description of the driver installation can be found in chapter **Driver Installation for Universal Access Device**.

The UDE workbench is now installed and ready to use.

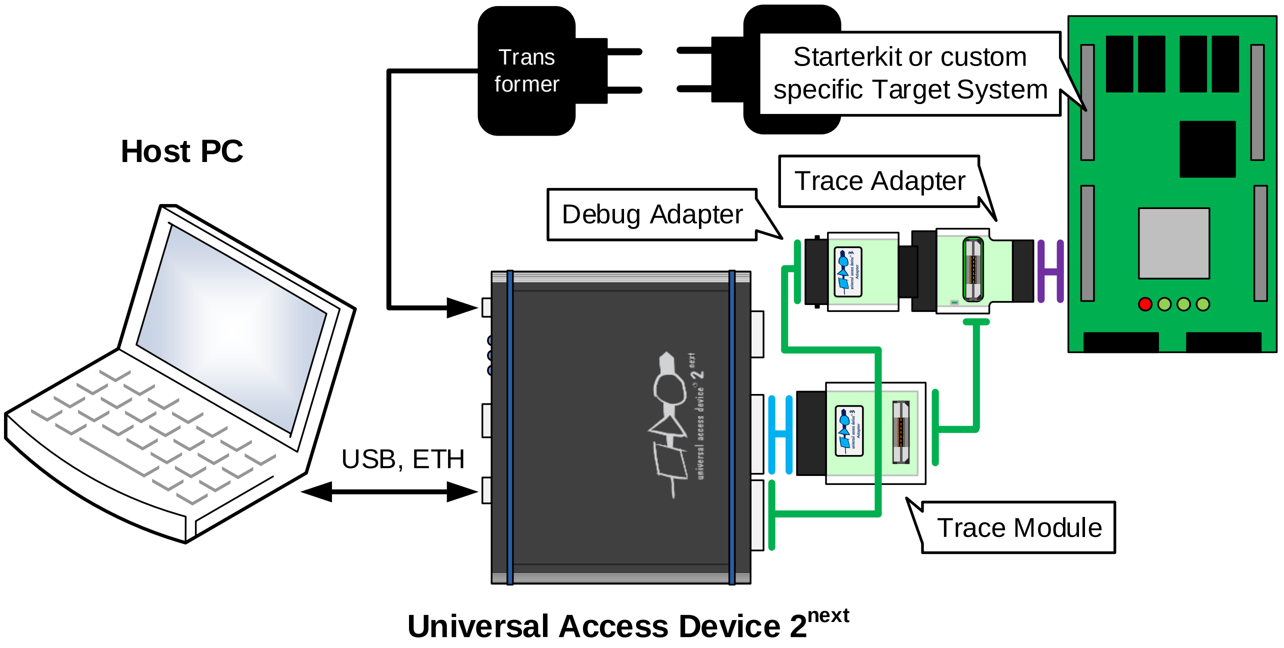
If you encounter difficulties installing the product, please contact the PLS Support Team at **support@pls-mc.com**.

Please note the important information about required ESD protection of the access devices in chapter **Static Electricity Precautions**!

Standard Version UAD2^{next} (with Trace support)

Install the hardware of UAD2^{next} as follows:

1. Connect the UAD2^{next} (connector 'USB 3.0') with an USB connector of the installed USB host adapter in your PC (USB 3.0 is recommended) **or** the connector ‘Ethernet’ with an Ethernet network.
2. Connect the ground socket with the ground potential of your target hardware for discharging the static electricity and balancing ground potentials.
3. Connect the Trace Module with the Trace Interface of UAD2^{next} and the Trace Adapter. Connect the Debug Adapter with the Trace Adapter. Connect the Trace Adapter with the Trace connector of the target board. Refer to the appendix **Hardware Description** of the user’s manual for more information about the correct pin connections.


Trans former
Starterkit or custom specific Target System
Host PC
Trace Adapter
Debug Adapter
USB, ETH
Trace Module
Universal Access Device 2next

4. Connect the wall plug transformer's cable with the 'Power' jack to the UAD2^{next} and power on the system.
5. The driver software was installed during the previous software installation. After initialization, the UAD2^{next} is connected to the host system and ready. A detailed description of the driver installation can be found in chapter **Driver Installation for Universal Access Device**.

The UDE workbench is now installed and ready to use.

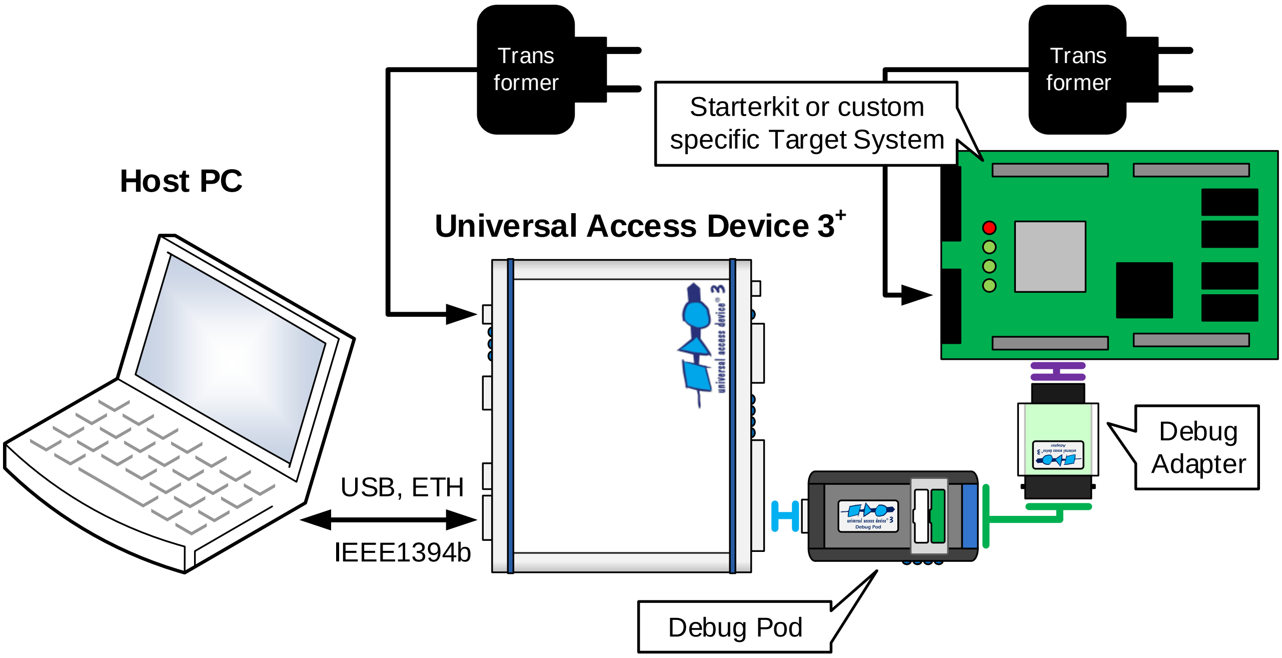
If you encounter difficulties installing the product, please contact the PLS Support Team at **support@pls-mc.com**. For Trace installation, several **Application Notes** can be provided on request.

Please note the important information about required ESD protection of the access devices in chapter **Static Electricity Precautions**!

Standard Version UAD3$^+$ (via JTAG/DAP/SWD)

Install the hardware of UAD3$^+$ as follows:

1. Connect the UAD3$^+$ (connector 'IEEE1394b') with an IEEE1394b connector of the installed IEEE1394b host adapter in your PC **or** connect the connector 'USB 2.0' with an USB connector of the installed USB host adapter in your PC (USB 2.0 is recommend) **or** the connector Ethernet with an Ethernet network.
2. Connect the ground socket with the ground potential of your target hardware for discharging the static electricity and balancing ground potentials.
3. Connect one of the 'Pod 1-4' connectors with the Debug Pod via the Debug Pod Cable. Connect the Debug Pod with the Debug Adapter via the Debug Adapter Cable. Connect the Debug Adapter with the Debug connector of the target board. Refer to the appendix **Hardware Description** of the user's manual for more information about the correct pin connections.



4. Connect the wall plug transformer's cable with the 'Power' jack to the UAD3$^+$ and power on the system.
5. The driver software was installed during the previous software installation. After initialization, the UAD3$^+$ is connected to the host system and ready. A detailed description of the driver installation can be found in chapter **Driver Installation for Universal Access Device**.

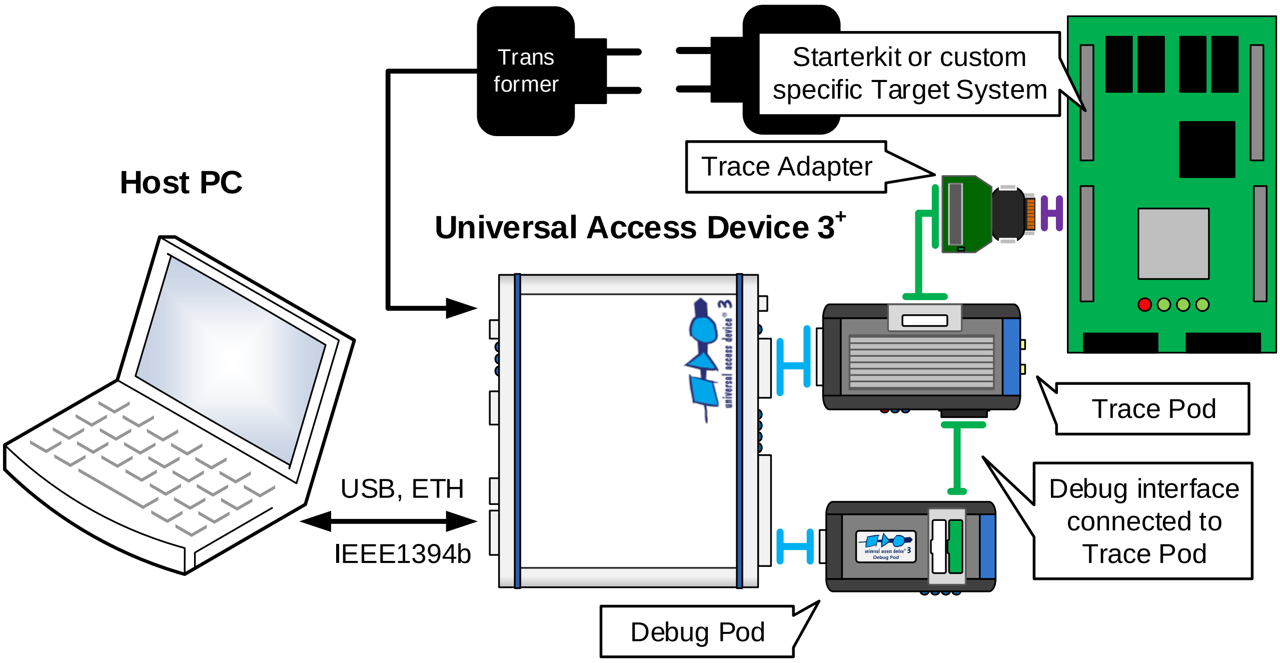
The UDE workbench is now installed and ready to use.

If you encounter difficulties installing the product, please contact the PLS Support Team at **support@pls-mc.com**.
Please see chapter **Debug/Trace Pod Configuration** in the UDE Manual Appendix for information about the firmware configuration of **UAD3$^+$ Debug/Trace Pods**.
Please note the important information about required ESD protection of the access devices in chapter **Static Electricity Precautions**!

Standard Version UAD3+ (with Aurora Trace support)

Install the hardware of UAD3+ as follows:

1. Connect the UAD3+ (connector 'IEEE1394b') with an IEEE1394b connector of the installed IEEE1394b host adapter in your PC **or** connect the connector 'USB 2.0' with an USB connector of the installed USB host adapter in your PC (USB 2.0 is recommend) **or** the connector ‘Ethernet’ with an Ethernet network.
2. Connect the ground socket with the ground potential of your target hardware for discharging the static electricity and balancing ground potentials.
3. Connect one of the ‘Pod 1-4’ connectors with the Debug Pod via the Debug Pod Cable. Connect the Debug Pod with the Trace Adapter via the Debug Adapter Cable. Connect the Trace Adapter with the Trace connector of the target board. Do the same with the Trace Pod interfaces. Refer to the appendix **Hardware Description** of the user’s manual for more information about the correct pin connections.


Trans former
Starterkit or custom specific Target System
Trace Adapter
Host PC
Universal Access Device 3+
Trace Pod
Debug interface connected to Trace Pod
USB, ETH
IEEE1394b
Debug Pod

4. Connect the wall plug transformer's cable with the 'Power' jack to the UAD3+ and power on the system.
5. The driver software was installed during the previous software installation. After initialization, the UAD3+ is connected to the host system and ready. A detailed description of the driver installation can be found in chapter **Driver Installation for Universal Access Device**.

The UDE workbench is now installed and ready to use.

If you encounter difficulties installing the product, please contact the PLS Support Team at support@pls-mc.com. For Trace installation, several **Application Notes** can be provided on request.
Please see chapter **Debug/Trace Pod Configuration** in the UDE Manual Appendix for information about the firmware configuration of **UAD3+ Debug/Trace Pods**.
Please note the important information about required ESD protection of the access devices in chapter **Static Electricity Precautions**!

Simulator Version (TSim)

Install the hardware of Universal Debug Engine as follows:

1. Connect the USB-Key with an USB connector of your PC.
2. Follow the driver installation when the 'New hardware found' dialog appears.
3. The driver software was installed during the software installation. After initialization, the USB-Key is connected to the host system and ready. A detailed description of the driver installation can be found in chapter **Driver Installation for Universal Access Device**.

The UDE workbench is now installed and ready to use.

If you encounter difficulties installing the product, please contact the PLS Support Team at **support@pls-mc.com**.

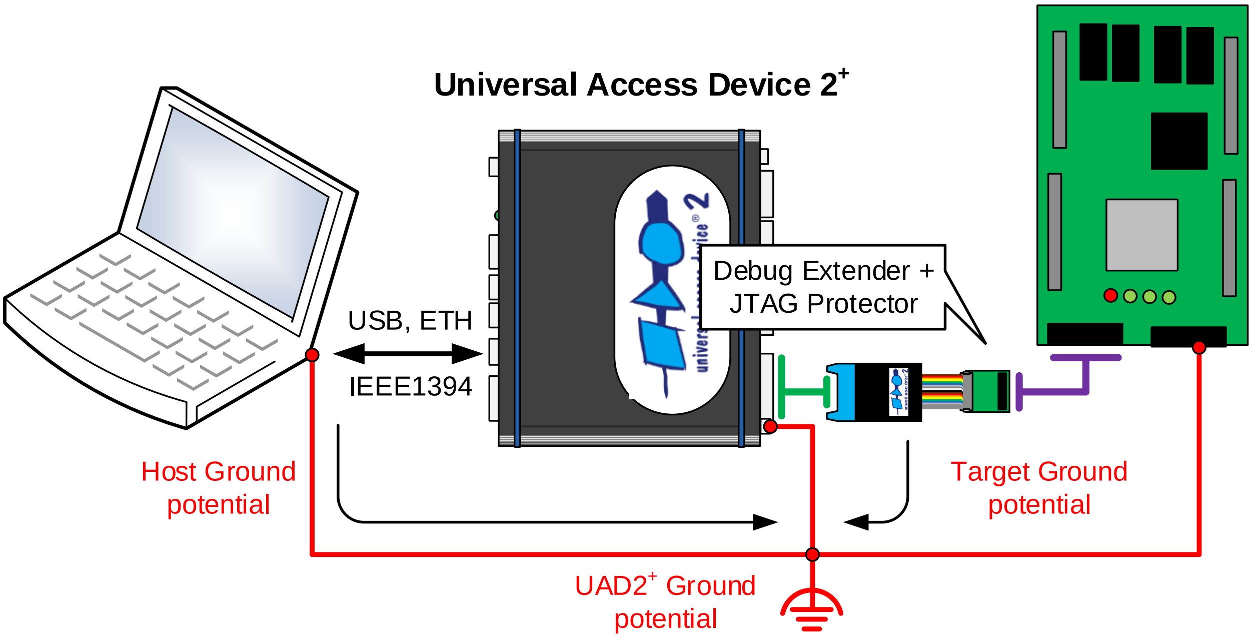
UAD-JTAG Protector 2 for UAD2⁺

In hard process environments it is strongly recommended to use the **JTAG-Protector** described below. This adapter allows an extended protection of the Universal Access Devices and other JTAG based devices from the danger of over-voltage and ESD.

Install the hardware of Universal Debug Engine **JTAG-Protector** as follows:

1. Connect the UAD-JTAG Protector 2 connector 'UAD' (female connector) with the JTAG connector of Debug Extender. If the Debug Extender is not equipped with a male shrouded header, please contact the PLS Support.
2. Connect the UAD-JTAG Protector 2 connector 'Target' (male connector) with the JTAG ribbon cable to your target.

Please note, that the function of the JTAG protections can be ensured only, when the ground and target power connections (MCU I/O voltage) are established.



If you encounter difficulties installing the product, please contact the PLS Support Team at **support@pls-mc.com**.

Please note the important information about required ESD protection of the access devices in chapter **Static Electricity Precautions**! Please note, that the JTAG Protector **DOES NOT** suspend the precautions described in this chapter.

Driver Installation for Universal Access Device

If the previous steps succeeded, the software is installed with all components.

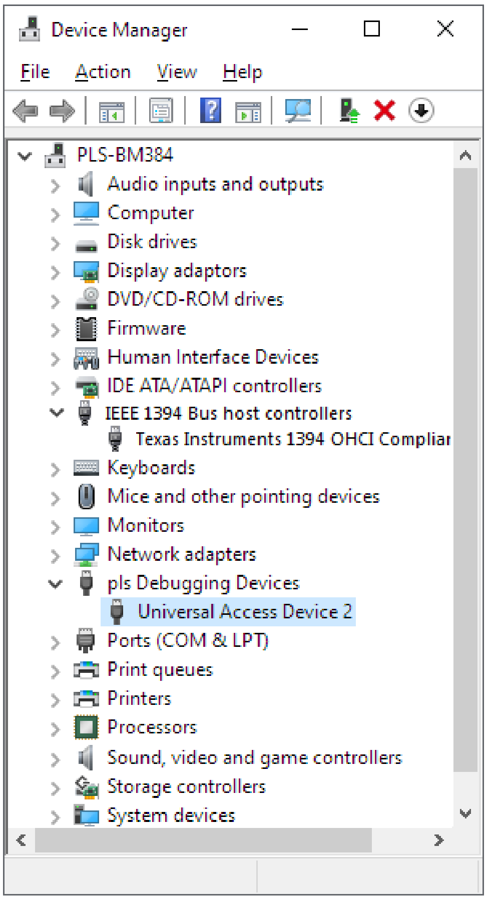
The installation process requires additional trusting and installing of the PLS signed drivers. A message box **“Windows Security: Would you like to install this device software? Name: pls Development Tools. Publisher: pls Programmierbare Logik & Systeme GmbH”** occurred during the installation and must be trusted during installation of the PLS device drivers.

When starting a newly installed version of UDE for the first time, a **firmware update** may be executed for the access device (UAD2^{pro}, UAD2^{+}, UAD2^{next}, and UAD3^{+}). This may take some more time than usual for the “target connect” operation. Please **DO NOT** power OFF or unplug the access device during this period!

UAD2^{+} via IEEE1394

Because of the Plug 'n Play-Capabilities of UAD and UAD2^{+} the IEEE1394 driver instantiation is started automatically, when the Universal Access Device is connected to the host PC the first time.

1. Power ON the Universal Access Device.
2. Connect the UAD or UAD2^{+} with the host adapter of your PC using the 6-wire cable. The Windows system will find a new hardware device on your system called **Universal Access Device 2** in the **pls Debugging Devices** group and will instantiate a new device driver.



In the case of problems, please verify the correct installation. Open the Device Manager, the following entries are shown:

- **IEEE 1394 Bus host controllers**
  - Your host bus controller
- **pls Debugging Devices**
  - Universal Access Device 2

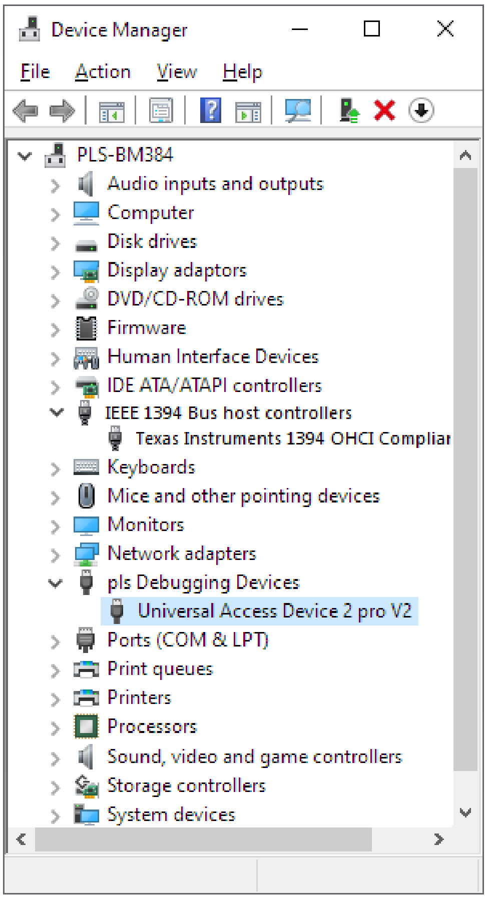
If one of the entries is marked with a question mark, please unplug UAD2 und repair the UDE Driver installation by running the program `<UDE_DIRECTORY>\Driver\ude-drivers.exe`.

Please see the chapter **Hardware Description** of the user’s manual for additional information.

UAD3$^+$ via IEEE1394b

Because of the Plug 'n Play-Capabilities of UAD3$^+$ the IEEE1394b driver instantiation is started automatically, when the Universal Access Device is connected to the host PC the first time.

1. Power ON the UAD3$^+$.
2. Connect the UAD3$^+$ with the host adapter of your PC using the 9-wire IEEE1394b cable or a 6-9-wire IEEE1394 bilingual cable. The Windows system will find a new hardware device on your system called **Universal Access Device 3** in the **pls Debugging Devices** group and will instantiate a new device driver.



- PLS-BM384
  - Audio inputs and outputs
  - Computer
  - Disk drives
  - Display adaptors
  - DVD/CD-ROM drives
  - Firmware
  - Human Interface Devices
  - IDE ATA/ATAPI controllers
  - IEEE 1394 Bus host controllers
    - Texas Instruments 1394 OHCI Compliar
  - Keyboards
  - Mice and other pointing devices
  - Monitors
  - Network adapters
  - pls Debugging Devices
    - Universal Access Device 2 pro V2
  - Ports (COM & LPT)
  - Print queues
  - Printers
  - Processors
  - Sound, video and game controllers
  - Storage controllers
  - System devices

In the case of problems, please verify the correct installation. Open the Device Manager, the following entries are shown:

- **IEEE 1394b Bus host controllers**
  - Your host bus controller
- **pls Debugging Devices**
  - Universal Access Device 3

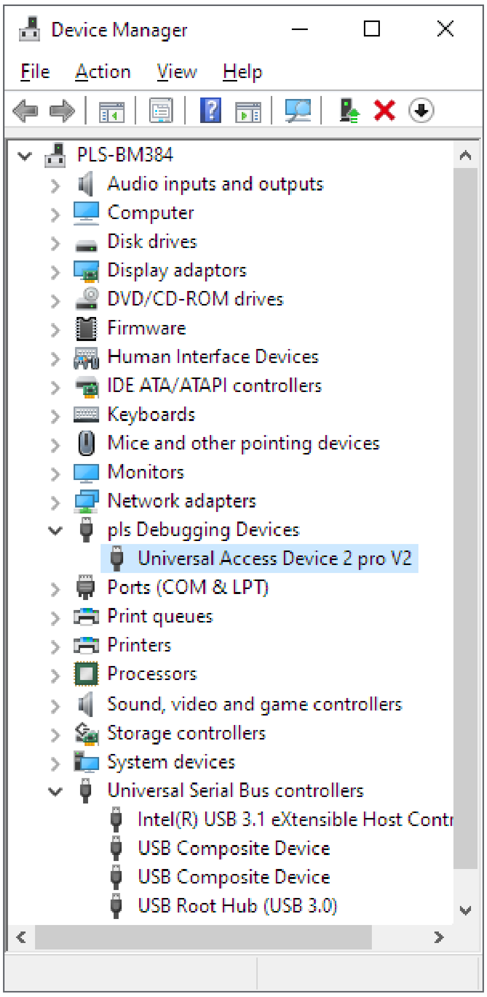
If one of the entries is marked with a question mark, please unplug UAD2 and repair the UDE Driver installation by running the program `<UDE_DIRECTORY>\Driver\ude-drivers.exe`.

Please see the chapter **Hardware Description** of the user’s manual for additional information.

UAD2$^{\text{pro}}$, UAD2^{+}, UAD2$^{\text{next}}$, UAD3^{+} via USB port

Because of the Plug 'n Play-Capabilities of UAD2, UAD2^{+}, UAD2$^{\text{next}}$ and UAD3^{+} the USB driver instantiation is starting automatically, when the UAD2 or UAD3 is connecting to the host PC the first time.

1. Connect the UAD2$^{\text{pro}}$, UAD2^{+}, UAD2$^{\text{next}}$ or UAD3^{+} to the PC host system using the USB cable. The Windows system will find a new hardware device on your system called **Universal Access Device 2/3** in the **pls Debugging Devices** group and will instantiate a new device driver.



In the case of problems, please verify the correct installation. Open the Device Manager, the following entries are shown:

- **Universal Serial Bus controllers**
  - Your host bus controller
- **pls Debugging Devices**
  - Universal Access Device 2/3

If one of the entries is marked with a question mark, please unplug UAD2 and repair the UDE Driver installation by running the program `<UDE_DIRECTORY>\Driver\ude-drivers.exe`.

Please see the chapter **Hardware Description** of the user’s manual for additional information.

UAD2$^+$, UAD2^{next}, UAD3$^+$ via Ethernet TCP/IP

The UAD2$^+$ and the UAD3$^+$ are equipped with a 100 Mbit/s fast Ethernet interface. The UAD2^{next} are equipped with a 1000 Mbit/s (1 Gbit/s) Ethernet interface. It can be connected to a local PC or to a local Network (LAN) via Hubs or Switches. The UAD2$^+$, UAD2^{next} and UAD3$^+$ support DHCP and static IP addressing.

Connection methods

Before using UAD2$^+$, UAD2^{next} or UAD3$^+$ for debugging purposes, the devices needed to be configured properly. The UAD2/3 can communicate to UDE via the TCP/IP protocol, if a valid IP (Internet Protocol) address is configured by:

1. Using DHCP, this requires a DHCP server on your network, or

2. Using a static IP address, this requires knowledge about the network structure, e.g. knowledge of free IP addresses so that there is no IP used twice in the network.

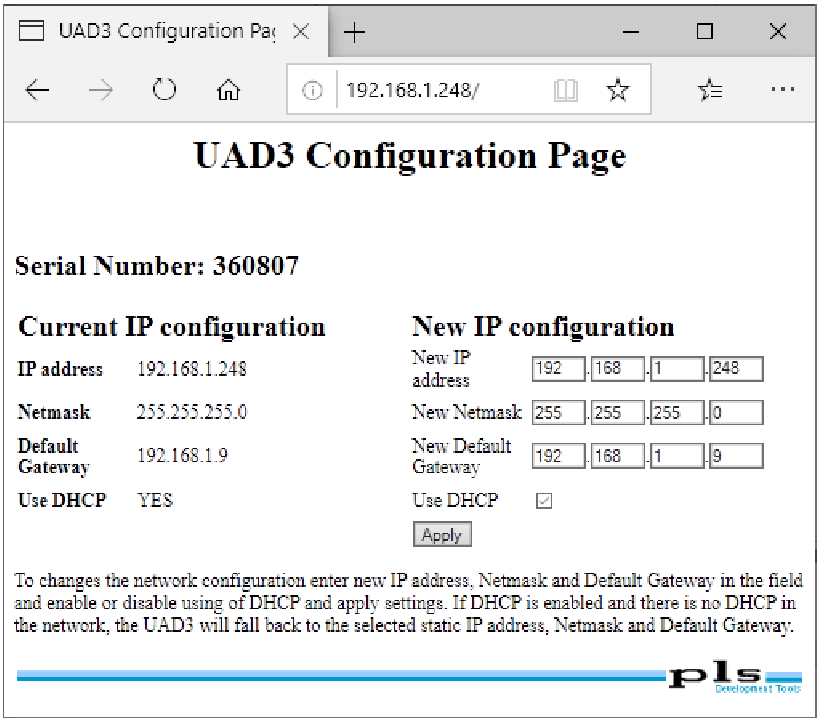
At factory settings, the UAD2$^+$, UAD2^{next} or UAD3$^+$ are configured with DHCP enabled. After power ON the UAD tries to receive an IP address from a DHCP server. If it receives no answer from a DHCP server, the UAD2$^+$, UAD2^{next} or UAD3$^+$ will fall back to a static IP address after 60 seconds.
The static fall back IP address is `192.168.1.100`. The UAD2$^+$, UAD2^{next}, UAD3$^+$ use the following TCP ports for communication: `43690 (0xAAAA)` and `43691 (0xAAAB)`.

Configuration of the IP address via Ethernet

The configuration of the UAD2$^+$, UAD2^{next} or UAD3$^+$ can be changed, using a web browser. After entering the current IP address, e.g.

```
http://192.168.1.248
```

the **UAD2 Configuration Page** or **UAD3 Configuration Page** appears as startup page. The configuration page contains the serial number of the UAD2$^+$, UAD2^{next} or UAD3$^+$ and the current configuration at the left side of then page.



UAD3 Configuration Page

Serial Number: 360807

Current IP configuration

IP address	192.168.1.248
Netmask	255.255.255.0
Default Gateway	192.168.1.9
Use DHCP	YES

New IP configuration

New IP address	192	168	1	248
New Netmask	255	255	255	0
New Default Gateway	192	168	1	9
Use DHCP	☑			

Apply

To changes the network configuration enter new IP address, Netmask and Default Gateway in the field and enable or disable using of DHCP and apply settings. If DHCP is enabled and there is no DHCP in the network, the UAD3 will fall back to the selected static IP address, Netmask and Default Gateway.

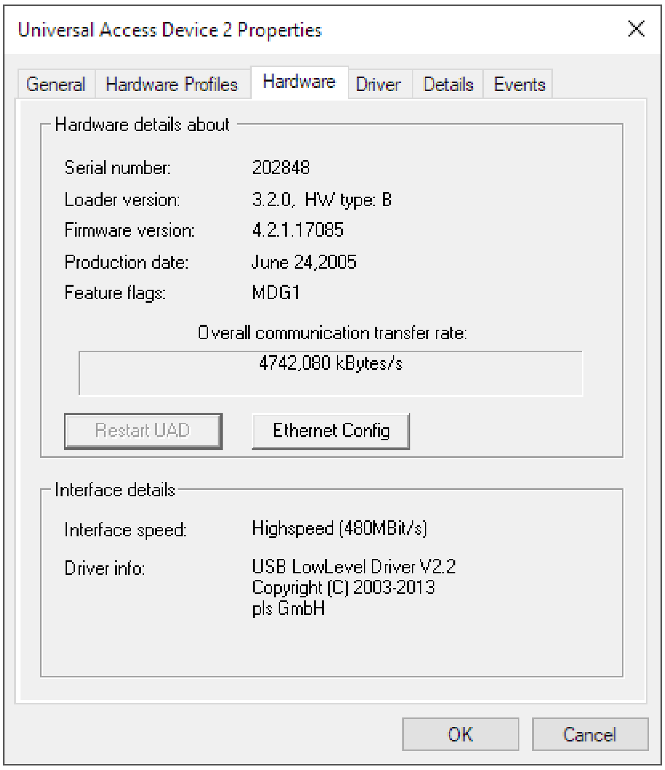
The example shows, that DHCP is enabled and the current IP address is `192.168.1.248`.

On the right of the form, new settings can be entered. The configured IP address will also be used as fallback, when DHCP is enabled but no DHCP answer is received. After clicking **Apply,** the new settings are stored. To apply the new settings immediately, power the UAD2^{+}, UAD2^{next} or UAD3^{+} OFF and ON again. Otherwise, they are applied after the next power ON event.

Configuration of the IP address via USB/IEEE1394

If the IP address of the UAD2^{+}, UAD2^{next} or UAD3^{+} is unknown, it can be configured using the USB or FireWire connection:

Connect the UAD2^{+}, UAD2^{next} or UAD3^{+} via USB or Firewire (when available) to a PC. Open the device manager's property page of the UAD2^{+}, UAD2^{next} or UAD3^{+} and select **Ethernet Config**.



Universal Access Device 2 Properties

Hardware details about

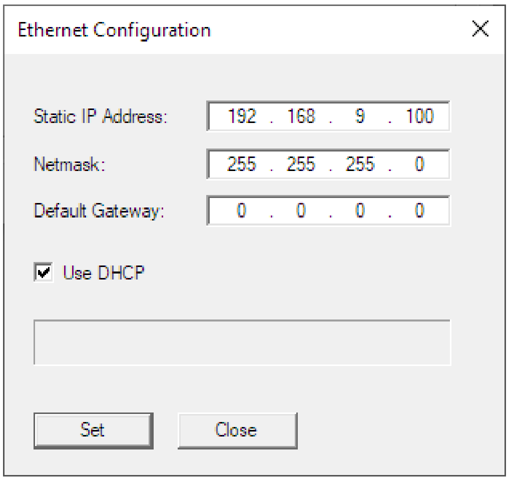
Serial number: 202848
Loader version: 3.2.0, HW type: B
Firmware version: 4.2.1.17085
Production date: June 24,2005
Feature flags: MDG1

Overall communication transfer rate:
4742,080 kBytes/s

Interface details

Interface speed: Highspeed (480MBit/s)
Driver info: USB LowLevel Driver V2.2
Copyright (C) 2003-2013
pls GmbH

The **Ethernet Configuration** dialog appears where the same settings can be made.



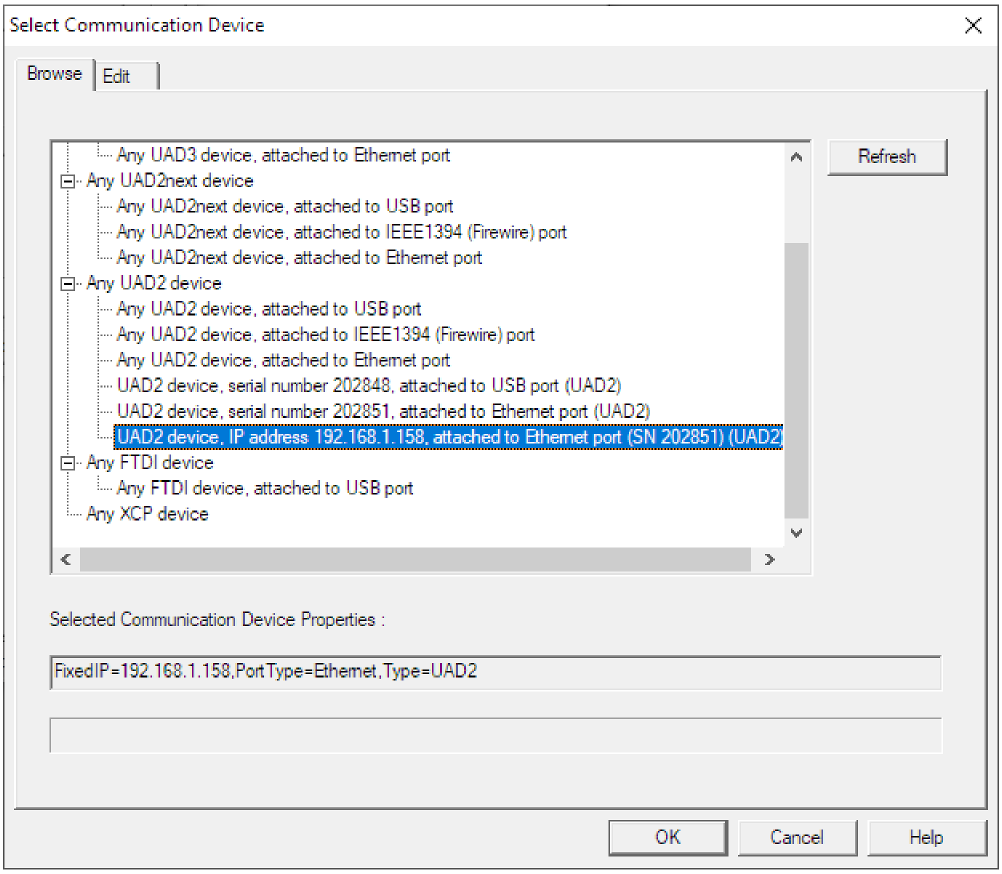
Ethernet Configuration

Static IP Address: 192 . 168 . 9 . 100
Netmask: 255 . 255 . 255 . 0
Default Gateway: 0 . 0 . 0 . 0

☑ Use DHCP

Once the UAD2$^+$, UAD2^{next} or UAD3$^+$ was configured, a connection via UAD2$^+$, UAD2^{next} or UAD3$^+$ can be established: Create a new workspace and select your target configuration. If **default** is set as communication device and there is no other UAD2$^+$ or UAD3$^+$ connected, the Ethernet device is found automatically.

If no UAD was found, open the menu entry **Config – Target interface...** in UDE or menu entry **Target – Setup** in MemTool. In the **Target Interface Setup,** dialog click on the **Setup** button.


Select Communication Device
Browse
Edit
Refresh
Selected Communication Device Properties :
OK
Cancel
Help

- Any UAD3 device, attached to Ethernet port
- Any UAD2next device
  - Any UAD2next device, attached to USB port
  - Any UAD2next device, attached to IEEE1394 (Firewire) port
  - Any UAD2next device, attached to Ethernet port
- Any UAD2 device
  - Any UAD2 device, attached to USB port
  - Any UAD2 device, attached to IEEE1394 (Firewire) port
  - Any UAD2 device, attached to Ethernet port
  - UAD2 device, serial number 202848, attached to USB port (UAD2)
  - UAD2 device, serial number 202851, attached to Ethernet port (UAD2)
  - UAD2 device, IP address 192.168.1.158, attached to Ethernet port (SN 202851) (UAD2)
- Any FTDI device
  - Any FTDI device, attached to USB port
- Any XCP device

FixedIP=192.168.1.158,PortType=Ethernet,Type=UAD2

For using the TCP/IP communication, the **Select Communication Device** dialog is opened. You can select the specific access device that you want to use. These settings are stored in the target configuration `*.cfg` file format.

For Ethernet connections select **UAD2 device, attached to Ethernet port**. A specific IP address to connect can be entered or an UAD2$^+$, UAD2^{next} or UAD3$^+$ can be selected from the list after retrieving available devices. Pressing **OK** applies the settings. A connection is established now.

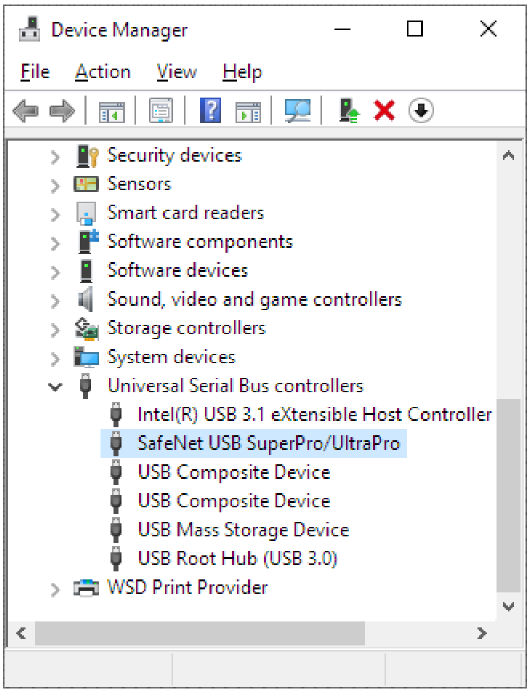
If multiple UAD2$^+$, UAD2^{next} or UAD3$^+$ are used at the same time (e.g. for automated FLASH programming), then every UAD2$^+$, UAD2^{next} or UAD3$^+$ have its own target configuration with either unique IP or unique serial number.

Driver Installation for USB-Key (Sentinel USB SuperPro)

Because of the Plug 'n Play-Capabilities of the USB-Key the USB driver installation is started automatically, when the USB-Key is connected to the host PC the first time.

USB-Key via USB port

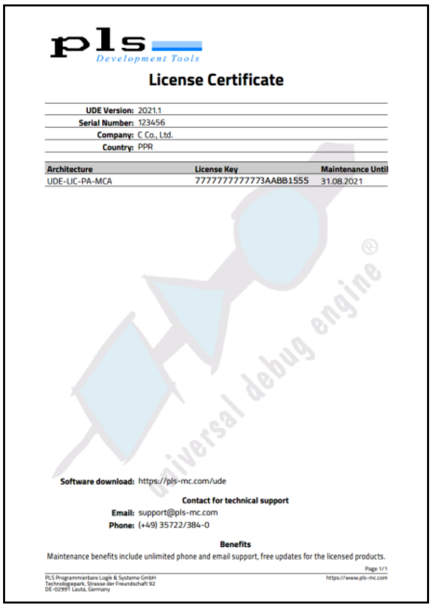
1. Connect the USB-Key to the PC host system using the USB port. The Windows system will find a new hardware device on your system called **USB SuperProNet** or **USB SuperPro** or **USB UltraPro** or **SafeNet USB SuperPro/UltraPro** in the **Universal Serial Bus controller** or **Other devices**. By default, the driver is installed during the UDE installation process.
2. Click **Next** and **Finish**.



License Manager

During the installation process, you will be asked for a license key.

1. You have got your personal license key as **License Certificate** within the consignment of Universal Debug Engine.



pls Development Tools

License Certificate

UDE Version:	2021.1
Serial Number:	123456
Company:	C Co., Ltd.
Country:	PPR

Architecture	License Key	Maintenance Until
UDE-LIC-PA-MCA	7777777777773AABB1555	31.08.2021

universal debug engine®

Software download: https://pls-mc.com/ude

Contact for technical support

Email: support@pls-mc.com

Phone: (+49) 35722/384-0

Benefits

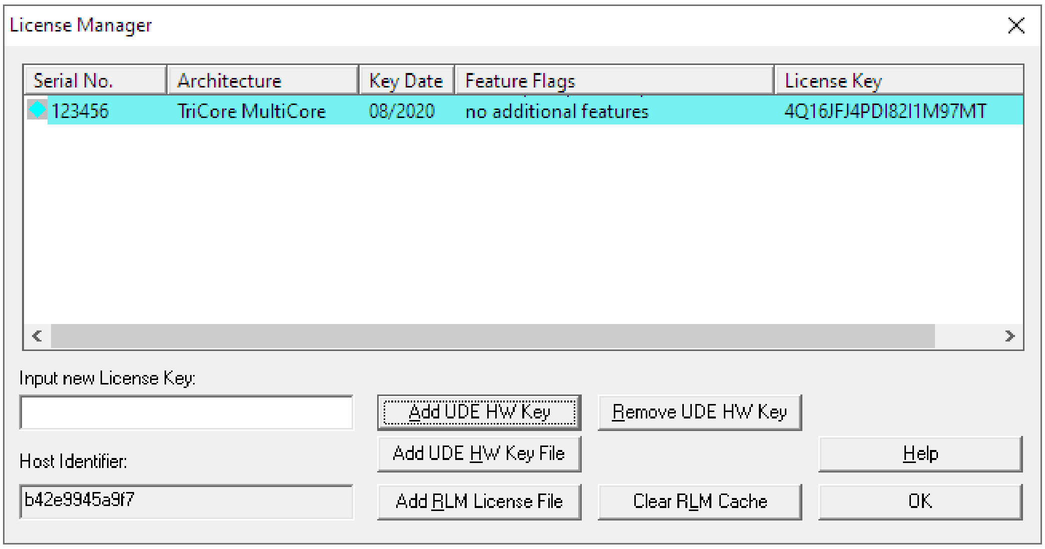
Maintenance benefits include unlimited phone and email support, free updates for the licensed products.

Page 1/1

PLS Programmierbare Logik & Systeme GmbH
Technologiepark, Strasse der Freundschaft 92
DE-02991 Lauta, Germany

https://www.pls-mc.com

Please enter the key into the **Input new License Key** edit field and click **Add UDE HW Key**. A new key entry will be added, if the license key is valid. Pushing the Button **OK** applies the changes to the system.



Serial No.	Architecture	Key Date	Feature Flags	License Key
123456	TriCore MultiCore	08/2020	no additional features	4Q16JFJ4PDI82I1M97MT

Input new License Key:

Host Identifier: b42e9945a9f7

2. Licenses from PLS are also distributed as key file with the extension `*.ukf`. Ask support@pls-mc.com for the license key file with the serial number of the affected UAD. Use the button **Add UDE HW Key File** to browse and select a new key file.

If you expand your feature list of UDE, you will get further license keys. These keys will either replace older keys or complement your key list. Please use the **Help – License Manager** from the system menu of UDE to open the License Manager.

If you do not have a valid license key for a used feature, a message box **License check failed** will appear as soon as you request this feature. Please contact **support@pls-mc.com** In this case.

Please note, that every license key applies to one Universal Access Device and one supported architecture only!

Node-locked licensing

Universal Debug Engine products can be licensed by node-locked licensing, if the product package does not contain any additional access device, which will be used normally for licensing of UDE. For this license type, the customer receives an activation key with his license documents. This activation key is the basic reference identifier assigned to his UDE license.

The activation process for a node-locked licensing based UDE product will cover the following steps:

1. The customer receives the **activation key** with his license documents and installs UDE on his Windows PC.
2. The customer determines the **host identifier** (host ID) of his PC. How to obtain a host ID will be described in the following chapter.
3. The host ID must be transmitted to PLS' support (e-mail to **support@pls-mc.com**) with the corresponding activation key from the UDE license documents.
4. The customer will receive the license file from PLS and install the license file using UDE license manager dialog.

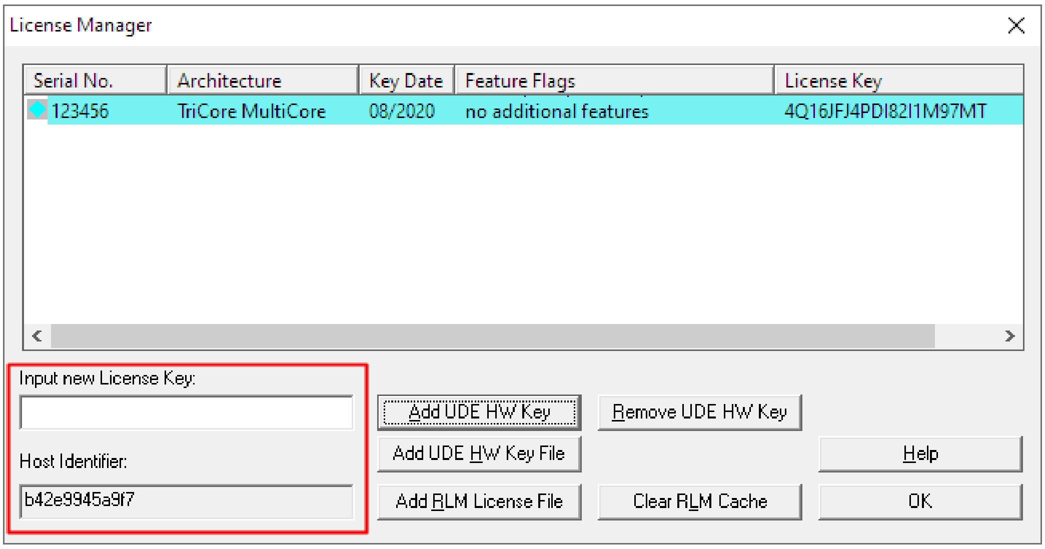
How to get the Host ID of UDE Installation

This chapter describes three methods to obtain the host ID, which are required for generating a node-locked license file.

Obtain Host Identifier via UDE License Manager

The following steps must be executed to obtain the host identifier by UDE license manager:

1. Start `UDE.exe`
2. Open UDE main menu **Help – License Manager** to open UDE license manager dialog



License Manager

Serial No.	Architecture	Key Date	Feature Flags	License Key
123456	TriCore MultiCore	08/2020	no additional features	4Q16JFJ4PDI82I1M97MT

Input new License Key:

Host Identifier: b42e9945a9f7

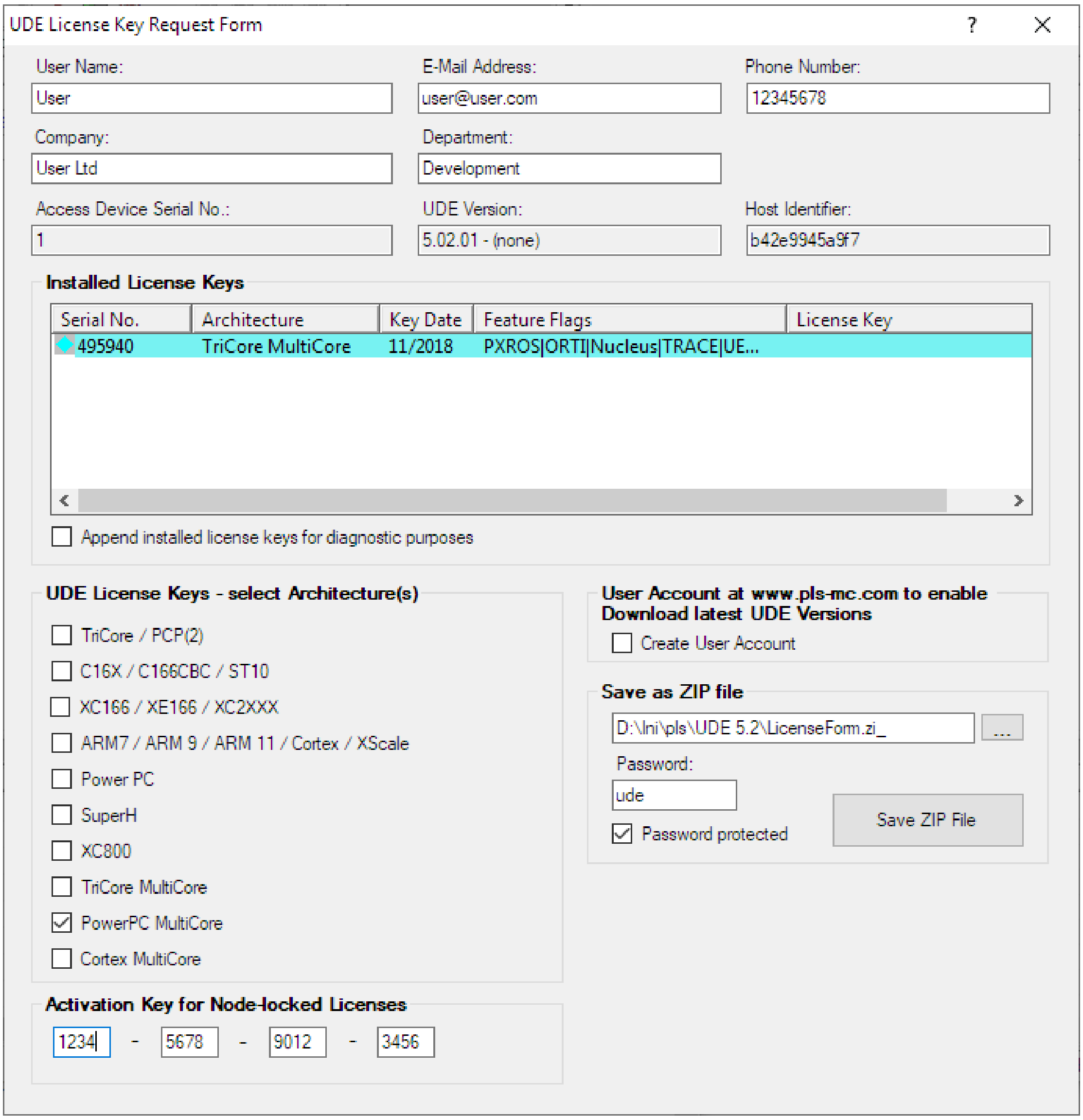
The content of the dialog field **Host Identifier** can now be copied to Windows clipboard and needs to be send to PLS' Support Team **support@pls-mc.com** with the activation key from license documents.

Obtain Host Identifier via UDE License Key Request Form

All information required for license activation can also be provided by **UDE License Request Form**. The usage of this service requires an active UDE configuration and is typically used for later requests for updated license keys.

The following step must be executed to create the appropriate information:

1. Start `UDE.exe` and connect to a target system.
2. Open UDE main menu **Help – UDE License Key Request Form** to open the request form.



UDE License Key Request Form

User Name: User	E-Mail Address: user@user.com	Phone Number: 12345678
Company: User Ltd	Department: Development	
Access Device Serial No.: 1	UDE Version: 5.02.01 - (none)	Host Identifier: b42e9945a9f7

Installed License Keys

Serial No.	Architecture	Key Date	Feature Flags	License Key
495940	TriCore MultiCore	11/2018	PXROS|ORTI|Nucleus|TRACE|UE...	

□ Append installed license keys for diagnostic purposes

UDE License Keys - select Architecture(s)

□ TriCore / PCP(2)
□ C16X / C166CBC / ST10
□ XC166 / XE166 / XC2XXX
□ ARM7 / ARM 9 / ARM 11 / Cortex / XScale
□ Power PC
□ SuperH
□ XC800
□ TriCore MultiCore
☑ PowerPC MultiCore
□ Cortex MultiCore

Activation Key for Node-locked Licenses

1234 - 5678 - 9012 - 3456

User Account at www.pls-mc.com to enable Download latest UDE Versions

□ Create User Account

Save as ZIP file

D:\Ini\pls\UDE 5.2\LicenseForm.zi_

Password: ude

☑ Password protected

3. Fill in all necessary fields of the **License Key Request Form**, such as User Name, Company, Department, E-Mail Address and **Activation Key for Node-locked Licenses**.
4. Fill the **Save as ZIP file** field by a local hard disk path and press the **Save ZIP File** button.
5. Close this dialog
6. The created ZIP file contains all necessary information. Send it directly to the PLS Support Team **support@pls-mc.com** without any further information.

Obtain Host Identifier via Script

The following three lines of script code must be saved to local disk as Visual Basic Script file `GetHostID.vbs`:

```
    set LicenseServer = CreateObject ("UDE.LicenseServer.2023")
    RequString = LicenseServer.RLMHostID
    WScript.Echo "PC Host ID:" & vbcrlf & RequString
```

This script must be executed on the Windows command line as follows:

```
    CScript GetHostID.vbs  > HostID.txt
```

The content of the file HostID.txt has to be send to PLS Support Team **support@pls-mc.com** with the activation key from the license documents.

The script COM-object identifier of the UDE license server is version depended. The COM-object id listed in the script code above is the COM-object id of UDE license server of UDE version 2023. If a different UDE version is used, the first line of code must be changed.

Using UDE version 2023 this script code line must be changed to:

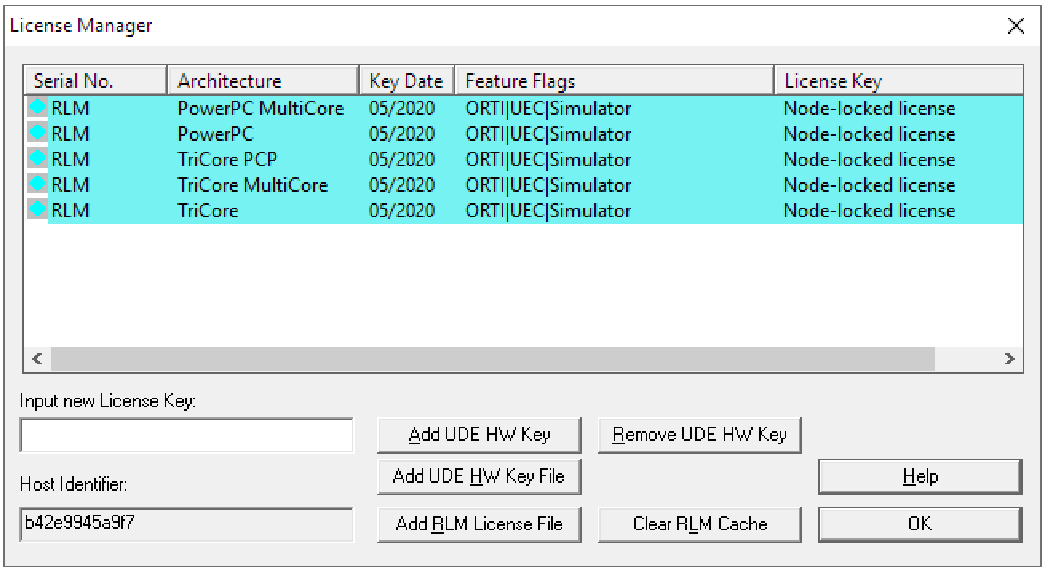
```
 set LicenseServer = CreateObject ("UDE.LicenseServer.2023")
```

Setup of Node-Locked License File

After sending the host identifier of your Windows PC to PLS, you will get a license file, which contains all licenses for all architectures, which are part of this license. The license file has the extension `*.lic` and the license file contains several lines:

Usually the license is sent as attachment of an e-mail from PLS Support Team. Save this attachment to a local hard disk path. Than execute the following steps to activate the node-locked license by UDE license manager dialog:

1. Start `UDE.exe`
2. Open UDE main menu **Help – License Manager** to open UDE license manager dialog
3. Press **Add NL License File** and browse with the opened file input explorer to your local copy of the license file.
4. After successful import, the license manger dialog will be updated and displays the additional node-locked licenses.



Serial No.	Architecture	Key Date	Feature Flags	License Key
RLM	PowerPC MultiCore	05/2020	ORTI|UEC|Simulator	Node-locked license
RLM	PowerPC	05/2020	ORTI|UEC|Simulator	Node-locked license
RLM	TriCore PCP	05/2020	ORTI|UEC|Simulator	Node-locked license
RLM	TriCore MultiCore	05/2020	ORTI|UEC|Simulator	Node-locked license
RLM	TriCore	05/2020	ORTI|UEC|Simulator	Node-locked license

Input new License Key:

Host Identifier: b42e9945a9f7

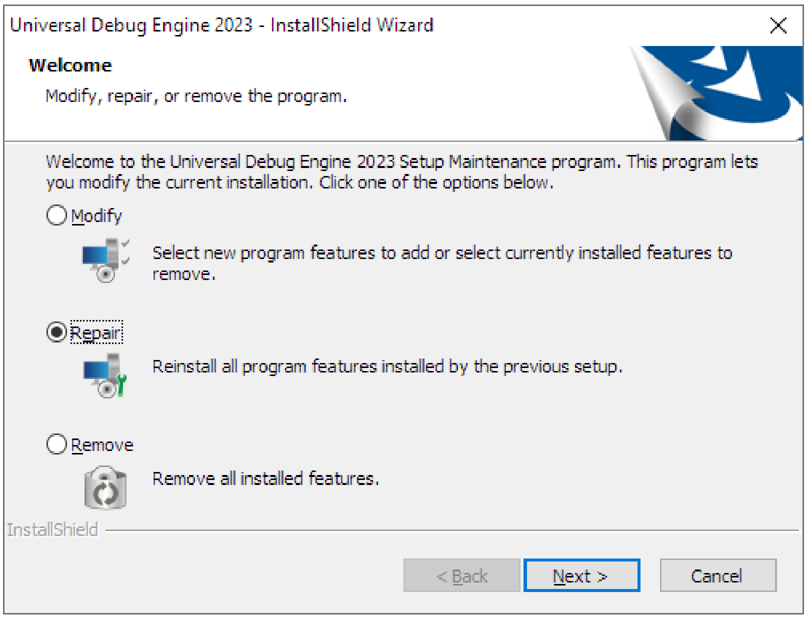
Uninstalling or Reinstalling UDE

For uninstalling the Universal Debug Engine (UDE) workbench, you may use the **Add/Remove Programs** feature from the Control Panel or the Remove function of the Setup program. The uninstall function will remove all UDE components, except user changed files, like workspace or configuration files.

Note: If you made any modifications of the UDE Examples delivered with UDE setup and you want to keep these changes, copy or move the example source files to another location before uninstalling or reinstalling of UDE. However, the uninstall procedure will not delete other projects and files.

For uninstalling of UDE, execute the following three steps:

1. Click **Start** from system menu, choose **Settings** and open the **Control Panel**.
2. Choose the **Add/Remove Programs** Icon and open it.
3. Select 'Universal Debug Engine' and click **Change/Remove ...** Button.



Universal Debug Engine 2023 - InstallShield Wizard

Welcome

Modify, repair, or remove the program.

Welcome to the Universal Debug Engine 2023 Setup Maintenance program. This program lets you modify the current installation. Click one of the options below.

○ Modify

Select new program features to add or select currently installed features to remove.

● Repair

Reinstall all program features installed by the previous setup.

○ Remove

Remove all installed features.

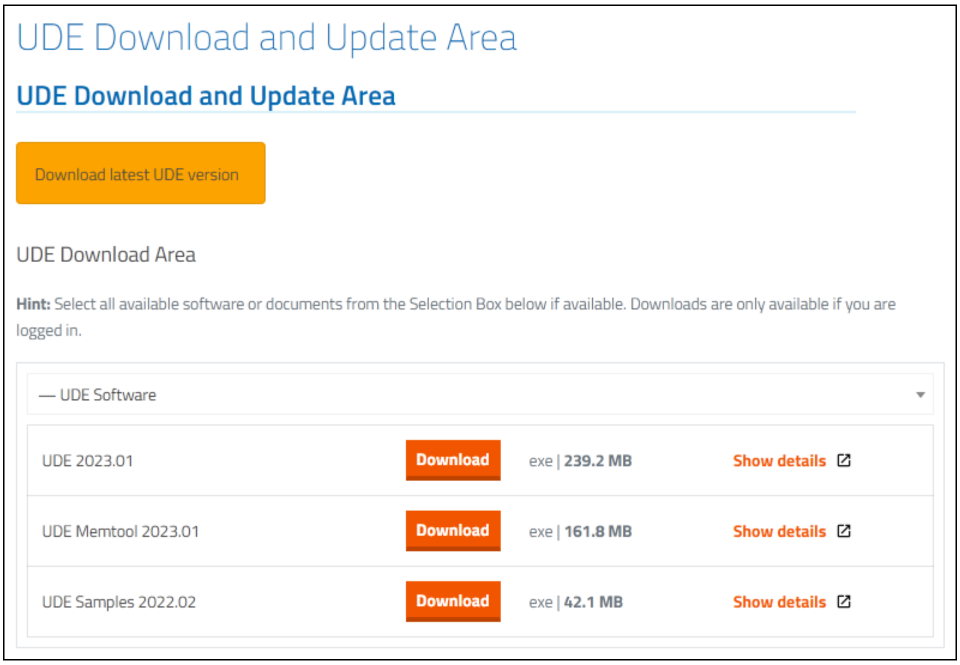
For reinstalling the **Universal Debug Engine** start the Setup program `Setup.exe` from the UDE CD-ROM or from the file you downloaded from **https://www.pls-mc.com/download.htm**. In the first step, click Repair and follow the instructions on the screen.

Find out about the latest UDE version

UDE is constantly being developed and improved. Therefore, new UDE major and minor versions are released regularly. The information and executable files of the latest UDE version are provided in the following ways:

1. On the website at https://www.pls-mc.com/downloads.html the button **Download latest UDE version** is linked with the latest UDE version for download (see the picture below). The download of the latest major UDE version will start immediately.
   If you're not logged in, you will be asked to log in. Please register your account once before. If the download is not available, please contact our **Support Team** at support@pls-mc.com.

2. Log on the website at https://www.pls-mc.com/accounts/login/ and browse to menu Service – Downloads and Updates. The dropdown box of the **UDE Download Area** offers all available UDE Version corresponding your maintenance contract.



UDE Download and Update Area

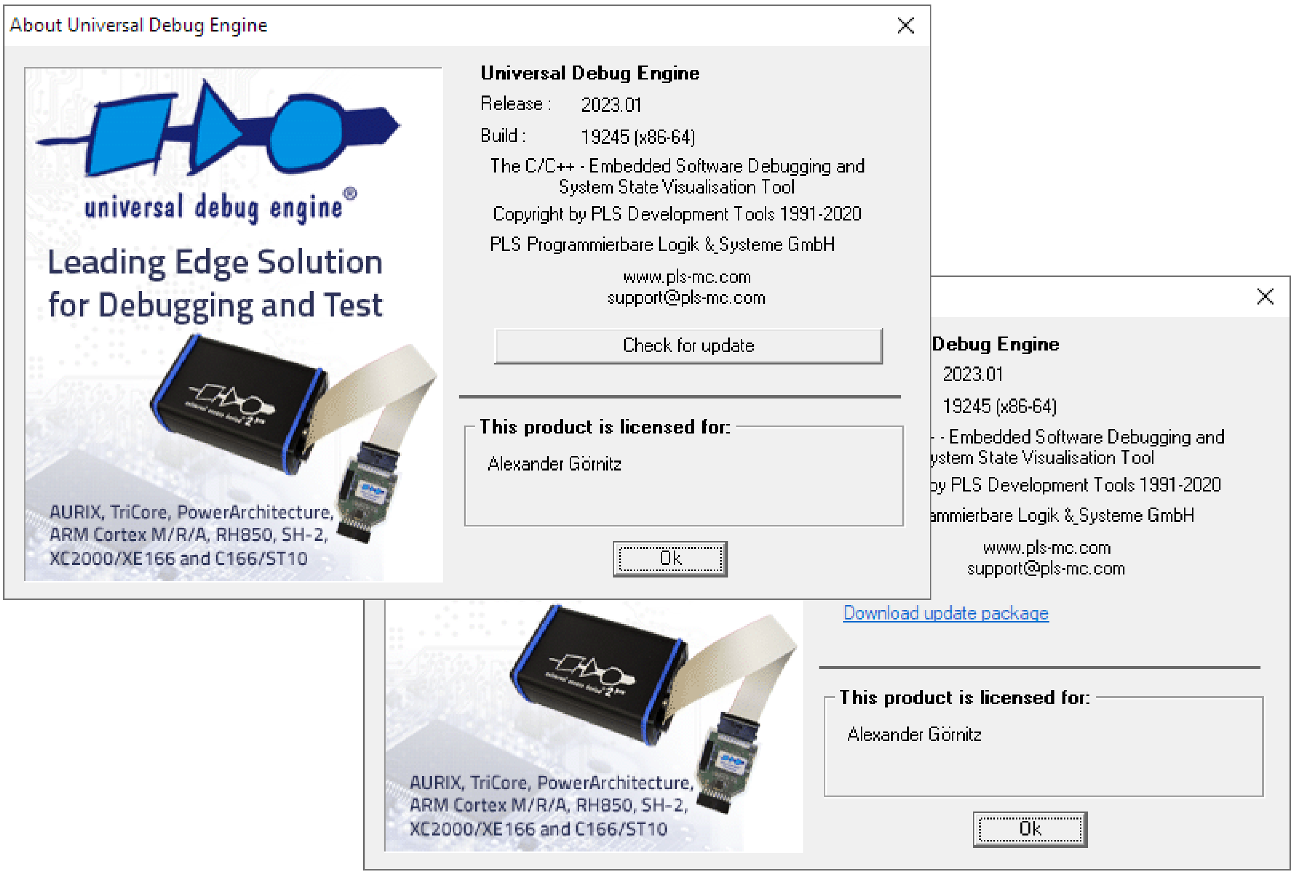
UDE Download and Update Area

UDE Download Area

Hint: Select all available software or documents from the Selection Box below if available. Downloads are only available if you are logged in.

UDE 2023.01	Download	exe | 239.2 MB	Show details
UDE Memtool 2023.01	Download	exe | 161.8 MB	Show details
UDE Samples 2022.02	Download	exe | 42.1 MB	Show details

The UDE about box can check, if a new UDE version is available. Open the **About Box** via UDE menu, the UDE MemTool menu or the UDEAdmin menu **Help – About UDE…** and push the **Check for update** button. If an UDE update is available, a download link will be provided as described in the first way.



About Universal Debug Engine

Universal Debug Engine

Release : 2023.01

Build : 19245 (x86-64)

The C/C++ - Embedded Software Debugging and System State Visualisation Tool

Copyright by PLS Development Tools 1991-2020

PLS Programmierbare Logik & Systeme GmbH

www.pls-mc.com
support@pls-mc.com

This product is licensed for:

Alexander Görnitz

universal debug engine®
Leading Edge Solution for Debugging and Test
AURIX, TriCore, PowerArchitecture, ARM Cortex M/R/A, RH850, SH-2, XC2000/XE166 and C166/ST10

Debug Engine
2023.01
19245 (x86-64)
- Embedded Software Debugging and
ystem State Visualisation Tool
by PLS Development Tools 1991-2020
ammierbare Logik & Systeme GmbH
www.pls-mc.com
support@pls-mc.com

Download update package

This product is licensed for:
Alexander Görnitz

Open the link and log in if required. Afterwards the download of the latest UDE version will start immediately.

Trouble Shooting

In case of trouble with UDE please read the UDE Manual and the UDE Manual Appendix very conscientiously. Make sure that you have connected all cables properly and selected the correct target device.

Frequently Asked Questions (FAQ's)

For technical questions, please consult the UDE FAQ list on our website **https://www.pls-mc.com/faqs.html** first.

Precautions when installing a new UDE version

⚠ When starting a newly installed version of UDE for the first time, a **firmware update** may be executed for the access device (UAD2^{pro}, UAD2^{+}, UAD2^{next}, and UAD3^{+}). This may take some more time than usual for the "target connect" operation. Please **DO NOT** power OFF or unplug the access device during this period!

Downloading the latest UDE Version

Our **PLS Newsletter** keeps you informed about latest news and software versions of the UDE. You can subscribe to the PLS Newsletter via your profile at **https://www.pls-mc.com/accounts/profile/**.

You may find the latest version of Universal Debug Engine and other components on our website **https://www.pls-mc.com/download.htm** for downloading. It is required that you are registered and logged in before.

Reporting a Problem in UDE

If a problem remains, after reading the hints of the UDE manual of the latest UDE version and the FAQs, please use an **UDE Support Request Form to request further help**.

Run UDE, open the affected workspace, set the Message View Log level to Maximum and reproduce the problem. Open menu **Help – UDE Support Request Form**, fill out the necessary fields and save it as a ZIP file. Now please send the ZIP file as attachment including the used password to the PLS Support Line at **support@pls-mc.com**.

Our Support team will contact you as soon as possible.

Known Issues

Installation fails

Please make sure that you have full rights (administrator rights) for the installation process of UDE.

UAD2/ UAD2^{+} does not enumerate at USB-Bus

This happens, if an UAD2, UAD2^{+} is connected to a target without powering UAD2, UAD2^{+}. If the target system gets power before the UAD2, UAD2^{+}, it is possible that it does not connect correctly to USB. In this case, power OFF target, power OFF UAD2, UAD2^{+}, disconnect USB and disconnect the target from UAD2. Then power ON UAD2, UAD2^{+}, connect to USB, power ON target and connect both together.

If the problem persists, please do contact the PLS Support Team at **support@pls-mc.com**.

Getting Started

Examples delivered with UDE

The Universal Debug Engine comes with a set of example programs that demonstrates the features of UDE. These examples are compiled with the available compilers for known Starterkit and Evaluation boards and are installed in <PROGRAMDATA>, which is located usually:

C:\Users\<USERNAME>\Documents\pls\UDE 2023\Samples

Automation	**Example programs for UDE automation (Phyton, Perl, C#, C++)**
C16x	**Example programs for C167, ST10F167, ST10276 and other derivatives**
Cortex	**Example programs for Cortex-M4 S32V234 derivative**
Cortexv8	**Example programs for Cortex-A53 S32V234 derivative**
PPC	**Example programs for PowerPC derivatives**
TriCore	**Example programs for TriCore derivatives**
TriCore2	**Example programs for AURIX TC2xx derivatives**
TriBoard_TC23x	**Example programs for AURIX TC23x**
...	...
TriBoard_TC29x	**Example programs for AURIX TC29x**
MultiCoreDemo	Example demonstration of multi-core features
SimioDemo	Example demonstration of the Simulated I/O capabilities, source code directory
TimeDemo	Example demonstration of the variables auto refresh mode and using interrupts
IntRam	Example located for using in internal RAM, contains binary and mapping file
IntRom	Example located for using in internal ROM, contains binary and mapping file
Src	Example source directory HighTec GNU compiler
TriCore3	**Example programs for AURIX TC3xx derivatives**
XC16x	**Example programs for XC161, XC167 and other derivatives**
XC2000	**Example programs for XC2287, XC2267 and other derivatives**

An Example with AURIX TC3xx

We assume that you now have successfully installed the **Universal Debug Engine (UDE)** version. In this section, you will learn about how to ...

- start UDE,
- use the windows in UDE and
- load, start and debug existing applications.

We recommend you to go through this tutorial step-by-step. This example is using the **AURIX TC399 TriBoard Starterkit** offered by Infineon Technologies with the communication hardware add-on UAD2^{pro}.

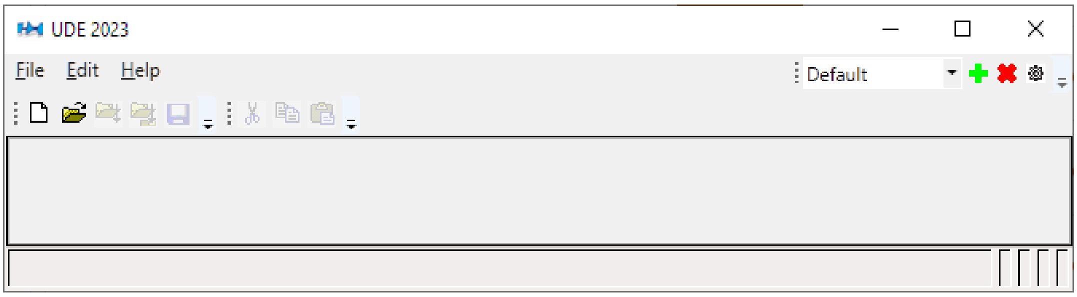
For further supported controllers, please have a look at the compatibility list below. For the correct setup of the board look up the chapter **Hardware Description Starterkit boards** in the Hardware Appendix of the UDE manual.

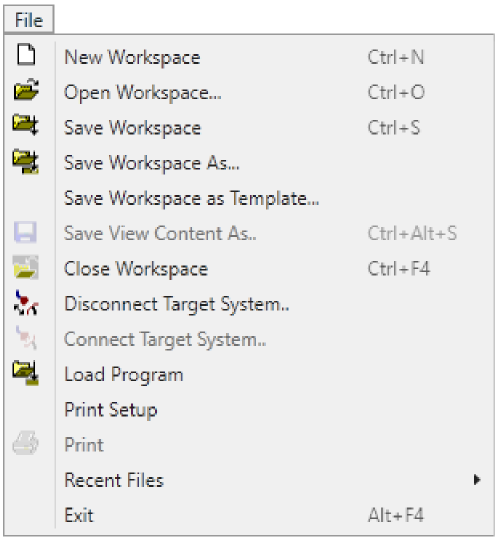
Precautions

When starting a newly installed version of UDE for the first time, a **firmware update** may be executed for the access device (UAD2^{pro}, UAD2^{+}, UAD2^{next}, and UAD3^{+}). This may take some more time than usual for the “target connect” operation. Please **DO NOT** power OFF or unplug the access device during this period! For UAD3^{+} specifics, please see the chapter **Debug/Trace Pod Configuration** from the UDE Manual Appendix.

Starting Universal Debug Engine

Once the operating system is up and running double-clicks on the icon **UDE** on the desktop. Alternatively, UDE may be launched also via **Start – Programs – Universal Debug Engine 2023 – UDE 2023**. This will start the desktop of UDE development system.





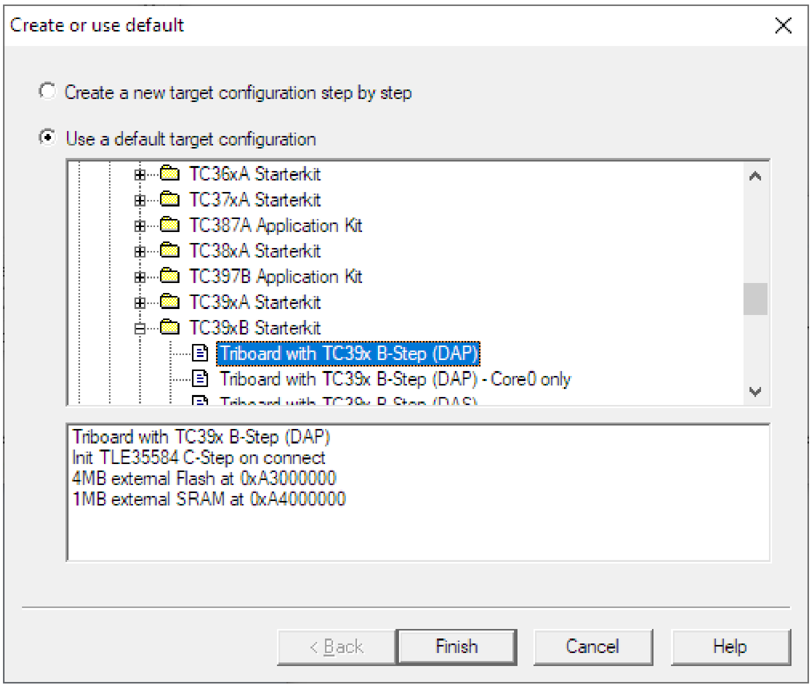
The next step is to create a new workspace. An UDE workspace saves all configurations and settings of UDE, windows and their content, path and name of loaded files. The file extension is `*.wsx`.

Click **New Workspace** at the **File** menu and create a new file, e.g. `TC399_TriBoard.wsx`.

After creating the new workspace, you will be asked to select a target hardware configuration. UDE provides some default target configurations of Starterkits. Click the **Default** button and enable **Use a default target configuration** to select a predefined configuration.

In a later chapter, the creation of a new target configuration will be described as well. In this example, a communication hardware add-on, like the UAD2^{pro}, is used.

The TriBoard with TC399XE-BA configuration is available under the entry **TriCore AURIX TC3xx** – **Infineon** – **TC39xB Starterkit** – **TriBoard with TC39x B-Step (DAP)**. The content of the parentheses describes the target communication channel.



Create or use default

○ Create a new target configuration step by step

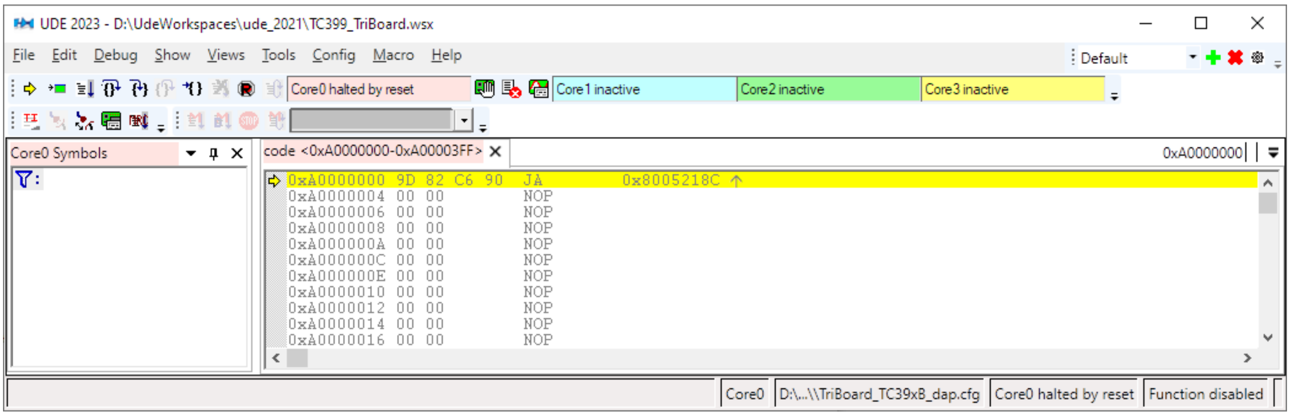
● Use a default target configuration

- TC36xA Starterkit
- TC37xA Starterkit
- TC387A Application Kit
- TC38xA Starterkit
- TC397B Application Kit
- TC39xA Starterkit
- TC39xB Starterkit
  - Triboard with TC39x B-Step (DAP)
  - Triboard with TC39x B-Step (DAP) - Core0 only
  - Triboard with TC39x B-Step (DAS)

Triboard with TC39x B-Step (DAP)
Init TLE35584 C-Step on connect
4MB external Flash at 0xA3000000
1MB external SRAM at 0xA4000000

Select this entry and click **Finish.** Set the name for the target configuration, e.g. `TriBoard_TC39xB_dap.cfg`.In the **Select target Configuration** dialog, click **OK** and select the cores for debugging.

If hardware initialization process is executed successfully, the UDE connects to the target hardware. For a first view the symbols and the target manager can be added via menu **Views – Symbols** and **Views – Target Manager**.



UDE 2023 - D:\UdeWorkspaces\ude_2021\TC399_TriBoard.wsx

```
0xA0000000  9D  82  C6  90   JA          0x8005218C ↑
0xA0000004  00  00           NOP
0xA0000006  00  00           NOP
0xA0000008  00  00           NOP
0xA000000A  00  00           NOP
0xA000000C  00  00           NOP
0xA000000E  00  00           NOP
0xA0000010  00  00           NOP
0xA0000012  00  00           NOP
0xA0000014  00  00           NOP
0xA0000016  00  00           NOP
```

Core0 | D:\...\TriBoard_TC39xB_dap.cfg | Core0 halted by reset | Function disabled

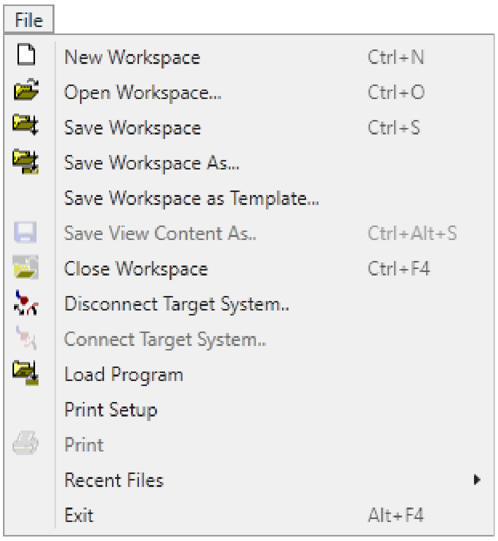
Furthermore, UDE generates a lot of hints and messages during its usage. It is helpful to check this. For that, enable the message view via menu **Views – Other windows – Messages** additionally.

If a valid license could not be found, a dialog **License Check Summary** appears. Please check if the license key, you have got in the delivered UDE package, was inserted in the License Manager. Use the menu **Help – License manager** for verification and inserting the key string. Alternatively, the hardware add-on is not connected or powered on. Please use the Device Manager from the Control panel for validation.

For loading an existing workspace file, use either **File – Open Workspace** or **File – Recent Workspaces** to select the workspace and start a new session with settings from the saved workspace.

If you got an error message, return to the section “Installing UDE Software” and make sure that all settings are correct. Please refer to section **Trouble Shooting** for further information. If the problem persists, contact the PLS Support Team at **support@pls-mc.com** for qualified help.

Loading a AURIX Executable



After having UDE for TriCore started, we want to load a program that can be executed on the TriCore Starterkit board.

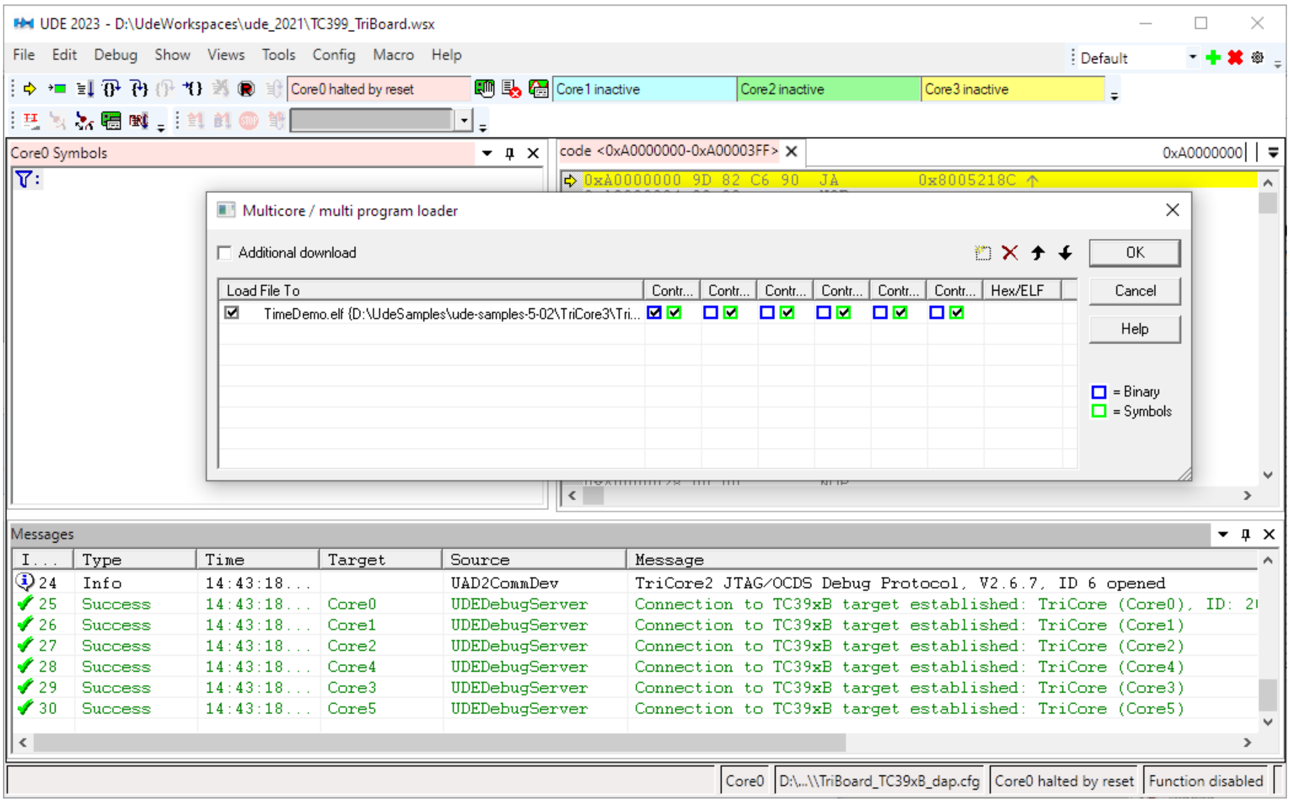
We will use the HighTec GNU compiler variant of the `TimeDemo`. This example will toggle the LED on the TriBoard only, for a basic demonstration of the debugger capabilities.

Select the menu entry **Load Program** in the **File** menu, browse to folder `<UDE_SAMPLE_DIRECTORY>\TriCore3\TriBoard_TC39x\TimeDemo\HighTec_IntRam\`

and load the file `TimeDemo.elf`.

Binary and Symbols

Because of the multi-core nature of the AURIX architecture, the dialog offers the capability to select binary and symbol information loading for each core separately.



☐ Additional download

Load File To	Contr...	Contr...	Contr...	Contr...	Contr...	Contr...	Hex/ELF
☑ TimeDemo.elf {D:\UdeSamples\ude-samples-5-02\TriCore3\Tri...	☑ ☑	☐ ☑	☐ ☑	☐ ☑	☐ ☑	☐ ☑	

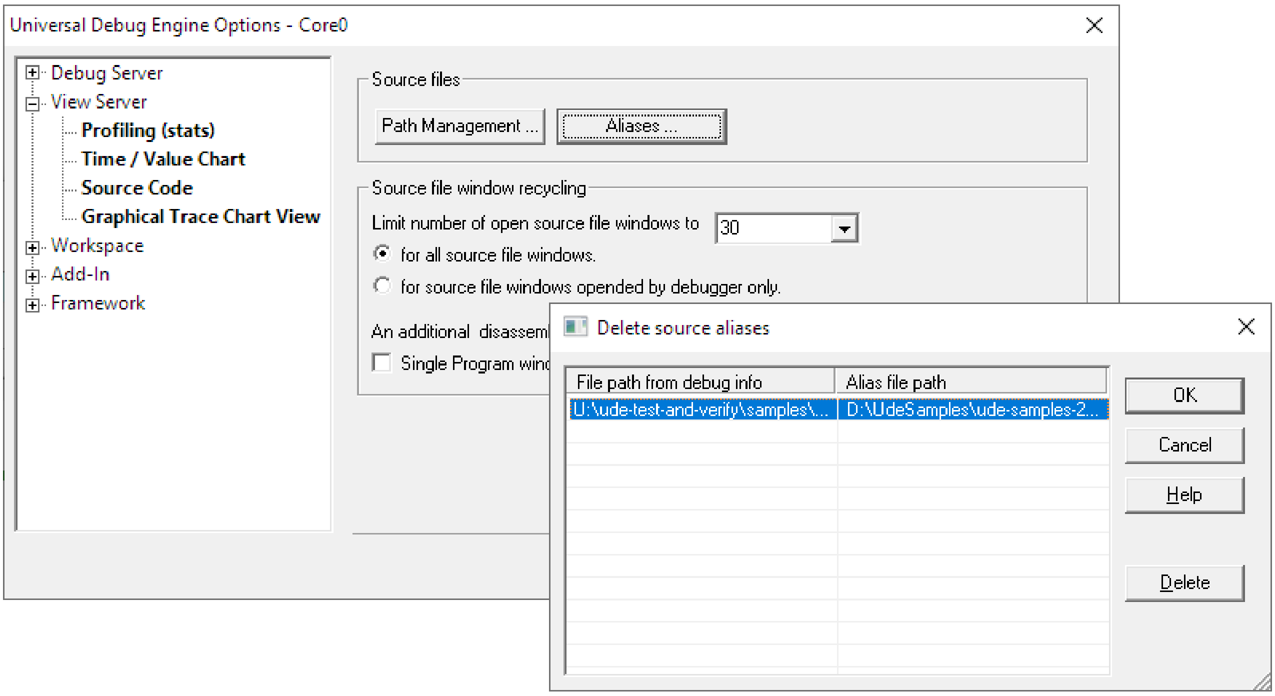
```
I...  Type     Time        Target  Source          Message
24    Info     14:43:18...         UAD2CommDev     TriCore2 JTAG/OCDS Debug Protocol, V2.6.7, ID 6 opened
25    Success  14:43:18...  Core0  UDEDebugServer  Connection to TC39xB target established: TriCore (Core0), ID: 2
26    Success  14:43:18...  Core1  UDEDebugServer  Connection to TC39xB target established: TriCore (Core1)
27    Success  14:43:18...  Core2  UDEDebugServer  Connection to TC39xB target established: TriCore (Core2)
28    Success  14:43:18...  Core4  UDEDebugServer  Connection to TC39xB target established: TriCore (Core4)
29    Success  14:43:18...  Core3  UDEDebugServer  Connection to TC39xB target established: TriCore (Core3)
30    Success  14:43:18...  Core5  UDEDebugServer  Connection to TC39xB target established: TriCore (Core5)
```

Source File Management

If the debugger cannot find the source files specified in the symbol containing files, it can request the location of the correct source path. This can happen, if the compiler’s-built environment differs from the environment used by the debugging process.

The examples source files are located in the `\src` and `\board` subfolders. You need to provide the location on request. UDE will remember your decision and will not ask again for this source path.

Every entry of this relocated source is called **Alias**. The saved aliases of a workspace are accessible via the menu **Config – Debug Server Configuration – View Server – Source Code – Aliases …**



Limit number of open source file windows to 30

● for all source file windows.

○ for source file windows opended by debugger only.

An additional disassem

□ Single Program winc

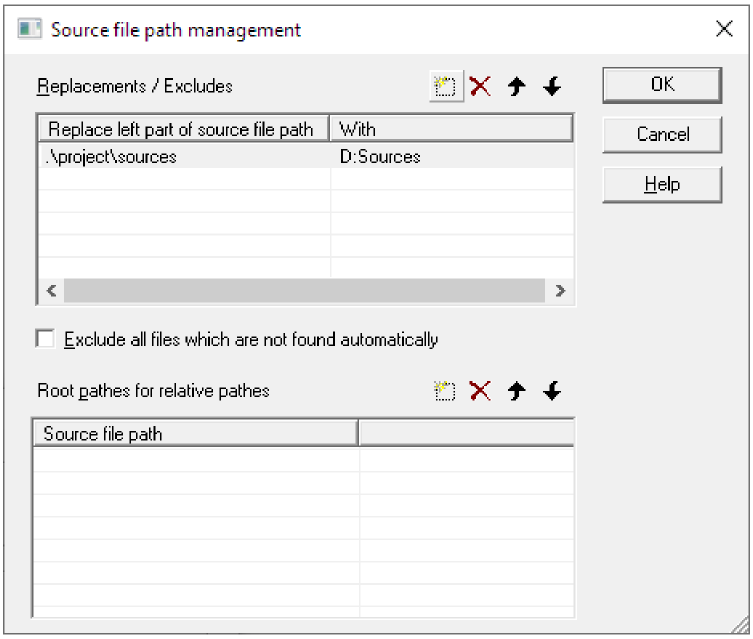
File path from debug info	Alias file path
U:\ude-test-and-verify\samples\...	D:\UdeSamples\ude-samples-2...

In some cases, the forced source alias may have changed. Use the dialog to delete the affected entry. The next time the source file is used, UDE will ask for the location again.

Source Path Replacement

The behavior described above is useful, if only few source file locations have changed. If a complete source file tree is affected, you can give UDE a replacement path, which will be applied to all source file paths.

Use the menu **Config – Debug Server Configuration – View Server – Source Code** – **Path Management …**



Replace left part of source file path	With
.\project\sources	D:Sources

□ Exclude all files which are not found automatically

Source file path	

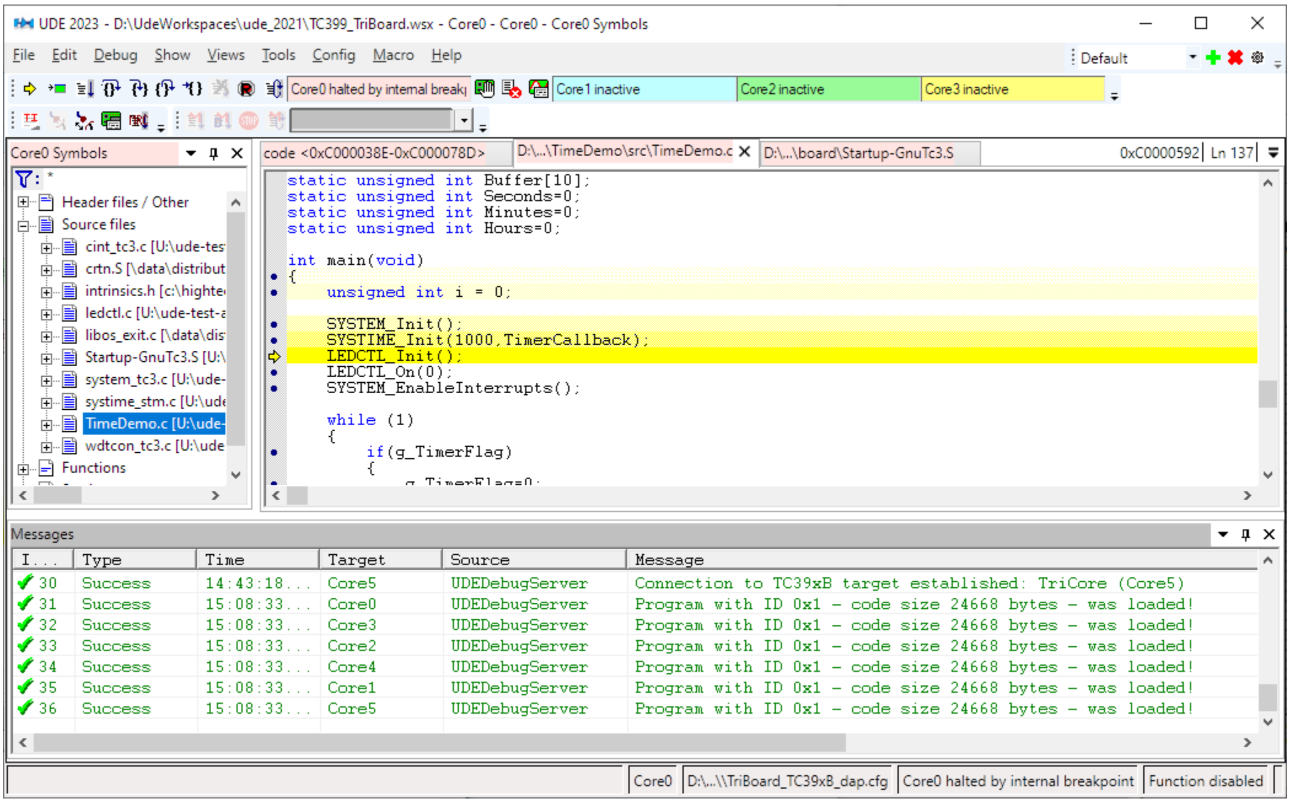
The dialog defines a part for replacing and a part for including the new source location. Via double-click you can change the item.

You will see now the source code of the main function of the sample application. After clicking with the right-hand mouse button into the Program window, a context menu appears to switch the display between Source code only and Mixed Source/Assembly code via the **Mixed Mode** entry.

Project management

A docked window at the left-hand side of UDE houses the Core Symbols tab where the application's source files and their inside procedures are shown after unfolding the markers. If no symbols window is shown, you can make it visible via the menu **Views – Symbols**. The UDE project contains source files, C/C++ functions, address sections and user-defined breakpoints.

By double-clicking on one of the source files in the Symbols window, the selected file will be opened in the Program window; by double-clicking on one of the procedures, the selected procedure is displayed. In the Program window, a yellow arrow indicates the current IP position.

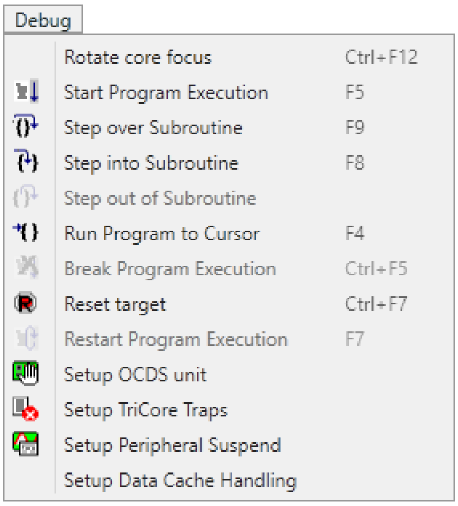


```
static unsigned int Buffer[10];
static unsigned int Seconds=0;
static unsigned int Minutes=0;
static unsigned int Hours=0;

int main(void)
{
    unsigned int i = 0;

    SYSTEM_Init();
    SYSTIME_Init(1000,TimerCallback);
    LEDCTL_Init();
    LEDCTL_On(0);
    SYSTEM_EnableInterrupts();

    while (1)
    {
        if(g_TimerFlag)
        {
            g_TimerFlag=0;
```

I...	Type	Time	Target	Source	Message
✓ 30	Success	14:43:18...	Core5	UDEDebugServer	Connection to TC39xB target established: TriCore (Core5)
✓ 31	Success	15:08:33...	Core0	UDEDebugServer	Program with ID 0x1 - code size 24668 bytes - was loaded!
✓ 32	Success	15:08:33...	Core3	UDEDebugServer	Program with ID 0x1 - code size 24668 bytes - was loaded!
✓ 33	Success	15:08:33...	Core2	UDEDebugServer	Program with ID 0x1 - code size 24668 bytes - was loaded!
✓ 34	Success	15:08:33...	Core4	UDEDebugServer	Program with ID 0x1 - code size 24668 bytes - was loaded!
✓ 35	Success	15:08:33...	Core1	UDEDebugServer	Program with ID 0x1 - code size 24668 bytes - was loaded!
✓ 36	Success	15:08:33...	Core5	UDEDebugServer	Program with ID 0x1 - code size 24668 bytes - was loaded!

Please note: After downloading a program executable, the IP (Instruction Pointer) is set to the entry point of the program. Usually the entry point is located at the start-up code. To force the view of the current IP, use the context menu **Show Next Statement** or the main menu **Show – Show IP**.

Running and Stepping through the Application



After the application has been loaded, you may open now the menu **Debug** to run or step through the example procedures.

Click now on the **Start Program Execution** entry or button and watch the LED on the TriCore Starterkit board flashing. When clicking on **Break Program Execution**, the application is halted and the current IP position (code line) is displayed. You may also step through the application by using **Step over Subroutine** or **Step into Subroutine** with following function calls and executing subroutines instruction by instruction.

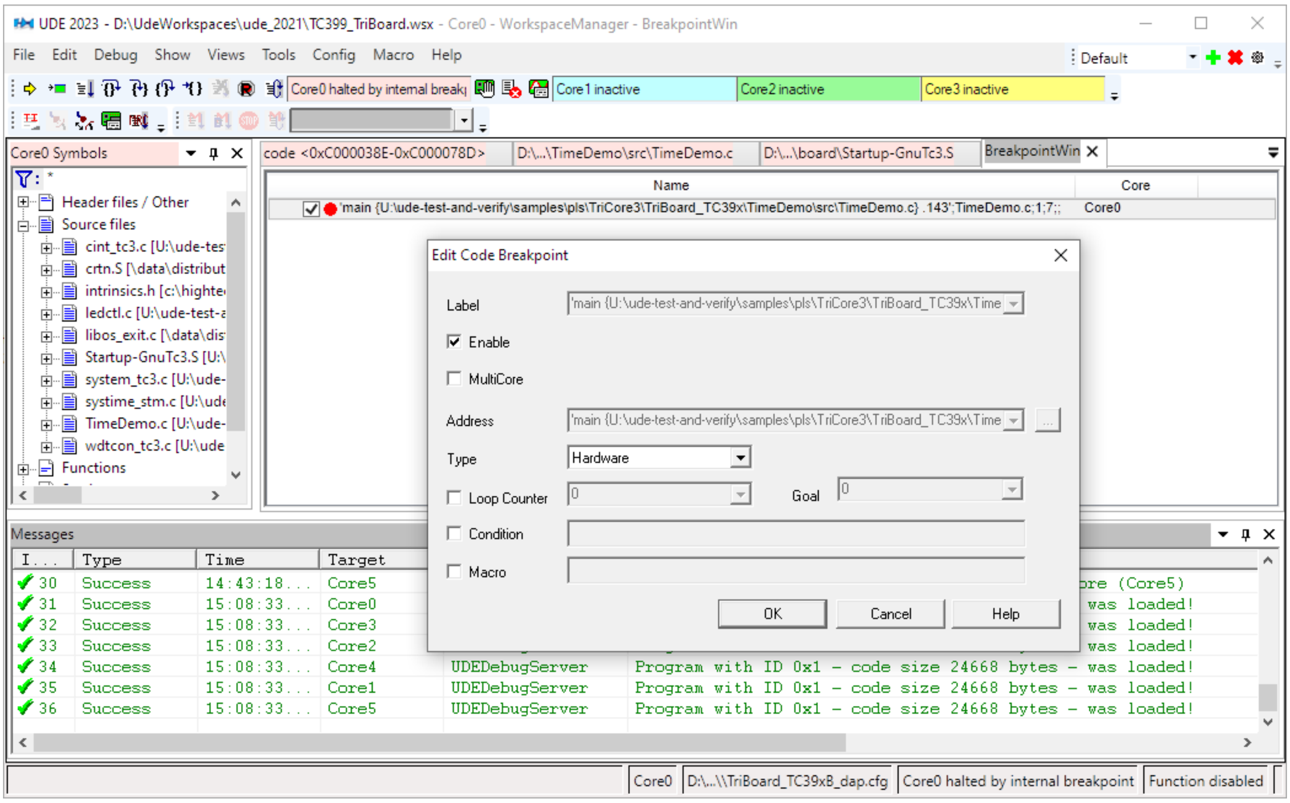
Note: For debugging the C/C++ parts of the example program only, the start-up code must be executed first. For this, make a **Step over Subroutine** from the **Debug** menu of UDE. After that, the IP will be shown at the main function; the start-up code has been executed.

The application can be reloaded with **Restart Program Execution**. If the program is running already, it will be started immediately after the reload terminates.

Setting Breakpoints

Now, we assumed that a loaded application is error-free and ready for running. However, for debugging purposes single step executions and breakpoints have to be performed to watch program behavior and processor status.

To set a breakpoint in the `TimeDemo` program, either left-click with the mouse on the grey column on the left of the Program window or activate a line in a procedure and then click on the **Hand** symbol in the tool bar. A red-filled dot appears in the line indicating that the breakpoint has been successfully set. Alternatively, you may also select the menu **Views – Breakpoints...**



UDE 2023 - D:\UdeWorkspaces\ude_2021\TC399_TriBoard.wsx - Core0 - WorkspaceManager - BreakpointWin

Name	Core
☑ ● 'main {U:\ude-test-and-verify\samples\pls\TriCore3\TriBoard_TC39x\TimeDemo\src\TimeDemo.c} .143';TimeDemo.c;1;7;;	Core0

Edit Code Breakpoint

- Label: 'main {U:\ude-test-and-verify\samples\pls\TriCore3\TriBoard_TC39x\Time
- ☑ Enable
- □ MultiCore
- Address: 'main {U:\ude-test-and-verify\samples\pls\TriCore3\TriBoard_TC39x\Time
- Type: Hardware
- □ Loop Counter: 0 Goal: 0
- □ Condition
- □ Macro

```
I...  Type     Time         Target
30    Success  14:43:18...  Core5                                     ore (Core5)
31    Success  15:08:33...  Core0                                     was loaded!
32    Success  15:08:33...  Core3                                     was loaded!
33    Success  15:08:33...  Core2                                     was loaded!
34    Success  15:08:33...  Core4  UDEDebugServer  Program with ID 0x1 - code size 24668 bytes - was loaded!
35    Success  15:08:33...  Core1  UDEDebugServer  Program with ID 0x1 - code size 24668 bytes - was loaded!
36    Success  15:08:33...  Core5  UDEDebugServer  Program with ID 0x1 - code size 24668 bytes - was loaded!
```

Core0 | D:\...\\TriBoard_TC39xB_dap.cfg | Core0 halted by internal breakpoint | Function disabled

In the Breakpoint dialog, breakpoints can be added, enabled, deleted and type changed, using the corresponding buttons. By clicking on the E/D (Enable/Disable), checkbox toggles the breakpoint between active and suspended. Disabled breakpoints are indicated by a red-shaped circle. Now start the application again. The application will be executed until the first breakpoint in the execution path is reached. The application will be stopped then immediately.

Note: The OCDS unit of the AURIX microcontroller supports an unlimited count of software breakpoints, but only four hardware breakpoints. Software breakpoints are realized by software code patching, which is only supported in writeable RAM areas. Hardware breakpoints are supported by the OCDS unit directly and can be used in read-only ROM areas only. Patching ROM areas (e.g. FLASH) is possible in principle, but not supported by UDE because of the vendor-defined limited FLASH programming cycles.

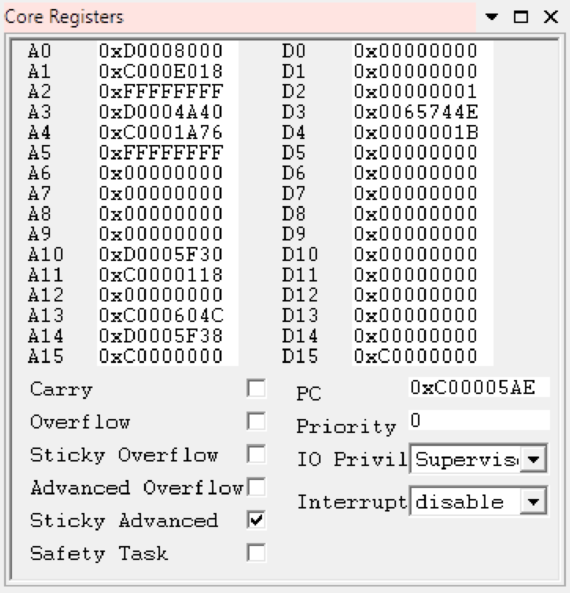
Another possibility to execute certain portions of code without setting a breakpoint explicitly is by placing the cursor into the line where the application is required to halt and then select **Debug – Run Program to Cursor** from the menu or **Run to Cursor (F4-Button)** from the context menu.

Core Registers

The Core Registers window is opened by the menu **Views – Core Registers** or the corresponding tool bar button.

Perform a few single steps to see the Core Registers values changing according to the executed instructions. If register value has been changed compared to the previous state are highlighted in red color.

If the program is stopped (e.g. between single steps) you may alter the content of registers. Click on the register's value in the Core Registers window and enter the new value.

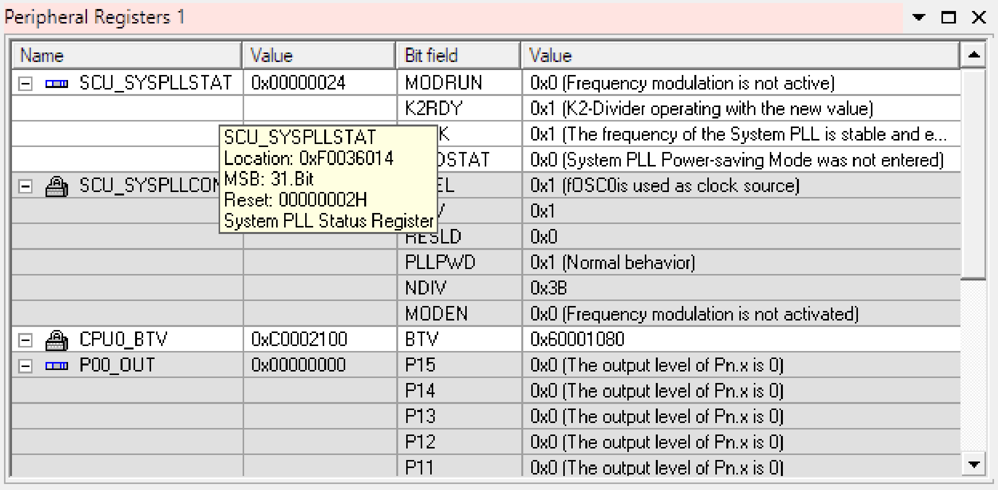


Core Registers

```
A0    0xD0008000    D0    0x00000000
A1    0xC000E018    D1    0x00000000
A2    0xFFFFFFFF    D2    0x00000001
A3    0xD0004A40    D3    0x0065744E
A4    0xC0001A76    D4    0x0000001B
A5    0xFFFFFFFF    D5    0x00000000
A6    0x00000000    D6    0x00000000
A7    0x00000000    D7    0x00000000
A8    0x00000000    D8    0x00000000
A9    0x00000000    D9    0x00000000
A10   0xD0005F30    D10   0x00000000
A11   0xC0000118    D11   0x00000000
A12   0x00000000    D12   0x00000000
A13   0xC000604C    D13   0x00000000
A14   0xD0005F38    D14   0x00000000
A15   0xC0000000    D15   0xC0000000

Carry               □    PC          0xC00005AE
Overflow            □    Priority    0
Sticky Overflow     □    IO Privil   Supervis▾
Advanced Overflow   □
Sticky Advanced     ☑    Interrupt   disable ▾
Safety Task         □
```

Peripheral Registers

Peripheral register values are changed in the same way as in the Core Registers window (open this window by selecting the menu **Views – Peripheral Registers**). To add a new register entry, select **Browse Ins** from the context menu of the Peripheral Registers window and double-click on the Peripheral register from the list to insert it into the Peripheral Registers window.



Peripheral Registers 1

Name	Value	Bit field	Value
SCU_SYSPLLSTAT	0x00000024	MODRUN	0x0 (Frequency modulation is not active)
		K2RDY	0x1 (K2-Divider operating with the new value)
		K	0x1 (The frequency of the System PLL is stable and e...
		DSTAT	0x0 (System PLL Power-saving Mode was not entered)
SCU_SYSPLLCON		EL	0x1 (fOSC0is used as clock source)
		/	0x1
		RESLD	0x0
		PLLPWD	0x1 (Normal behavior)
		NDIV	0x3B
		MODEN	0x0 (Frequency modulation is not activated)
CPU0_BTV	0xC0002100	BTV	0x60001080
P00_OUT	0x00000000	P15	0x0 (The output level of Pn.x is 0)
		P14	0x0 (The output level of Pn.x is 0)
		P13	0x0 (The output level of Pn.x is 0)
		P12	0x0 (The output level of Pn.x is 0)
		P11	0x0 (The output level of Pn.x is 0)

SCU_SYSPLLSTAT
Location: 0xF0036014
MSB: 31.Bit
Reset: 00000002H
System PLL Status Register

Tooltips show the address, the reset value of the current Peripheral Registers and further information about the focused register. The expanded Peripheral Registers view contains all available fields this register is composed off. The value can be changed by right-click on the values and entering the new value.

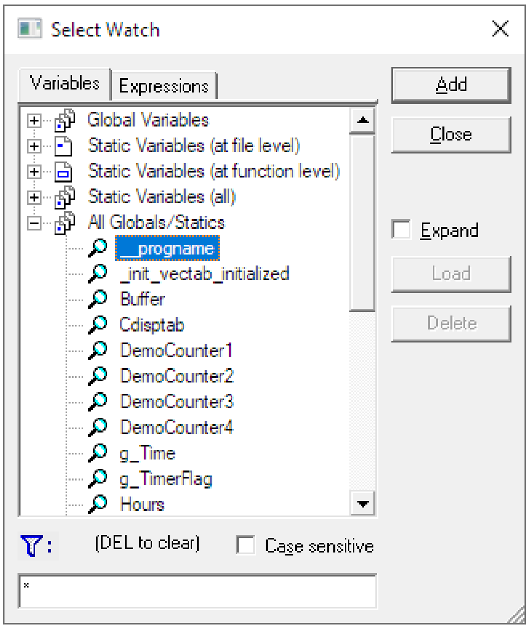
Various registers are protected, which means that a special unlock sequence is required to change the register value. UDE can unlock these registers. Use the context menu of the register name and disable the entry Write protect.

Viewing Variables

Viewing and changing global/static variables

All global and static variables from the C/C++ source code may be observed directly using the Watches window. Open the Watches Window by selecting the menu **Views – Watches** window or the corresponding tool bar button.

The variables can be added by double-clicking on <new variable> or using the context menu of the <new variable> entry via **Browse Ins**. The browser dialog shows you all available global and static variables. Click **Add** for adding a new variable to the Watches window.

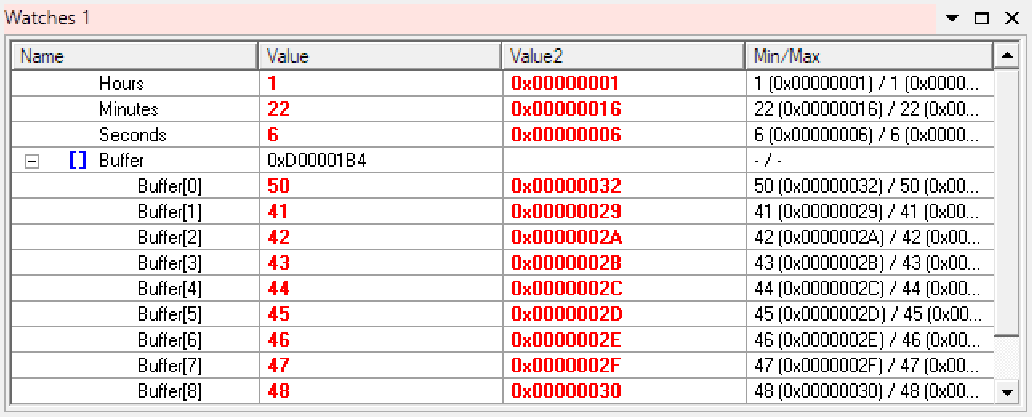


Global Variables
Static Variables (at file level)
Static Variables (at function level)
Static Variables (all)
All Globals/Statics
_progname
_init_vectab_initialized
Buffer
Cdisptab
DemoCounter1
DemoCounter2
DemoCounter3
DemoCounter4
g_Time
g_TimerFlag
Hours

The variables are sorted in groups as follows:

- Global Variables – shows all global variables with global scope.
- Static Variables (at file level) – shows only variables visible at a specific file.
- Static Variables (at function level) – shows only static variables visible at a specific function.
- Static Variables (all) – shows all static variables, which are not global variables.
- All Global/Static Variables – shows all global and static variables.
- If the variable is expandable, i.e. it is a pointer or an array, clicking on the '+' sign in front of the variable's name will expand it. This means, that the content of the target location of the pointer or the content of the elements of the array will be displayed. Variables can be expanded for more than one level.

Variable values can be changed easily by double-clicking in the value area or pressing **F2** and typing in the new value.



Name	Value	Value2	Min/Max
Hours	1	0x00000001	1 (0x00000001) / 1 (0x0000...
Minutes	22	0x00000016	22 (0x00000016) / 22 (0x00...
Seconds	6	0x00000006	6 (0x00000006) / 6 (0x0000...
⊟ [] Buffer	0xD00001B4		- / -
Buffer[0]	50	0x00000032	50 (0x00000032) / 50 (0x00...
Buffer[1]	41	0x00000029	41 (0x00000029) / 41 (0x00...
Buffer[2]	42	0x0000002A	42 (0x0000002A) / 42 (0x00...
Buffer[3]	43	0x0000002B	43 (0x0000002B) / 43 (0x00...
Buffer[4]	44	0x0000002C	44 (0x0000002C) / 44 (0x00...
Buffer[5]	45	0x0000002D	45 (0x0000002D) / 45 (0x00...
Buffer[6]	46	0x0000002E	46 (0x0000002E) / 46 (0x00...
Buffer[7]	47	0x0000002F	47 (0x0000002F) / 47 (0x00...
Buffer[8]	48	0x00000030	48 (0x00000030) / 48 (0x00...

Watch tips

Furthermore, UDE offers so-called watch tips, which show you the content of simple variables in the Program Window. Highlight i.e. the `Seconds` variable from the `main()` function by a double-click, move the mouse pointer over and the content will be displayed in a watch tip after a short waiting time.

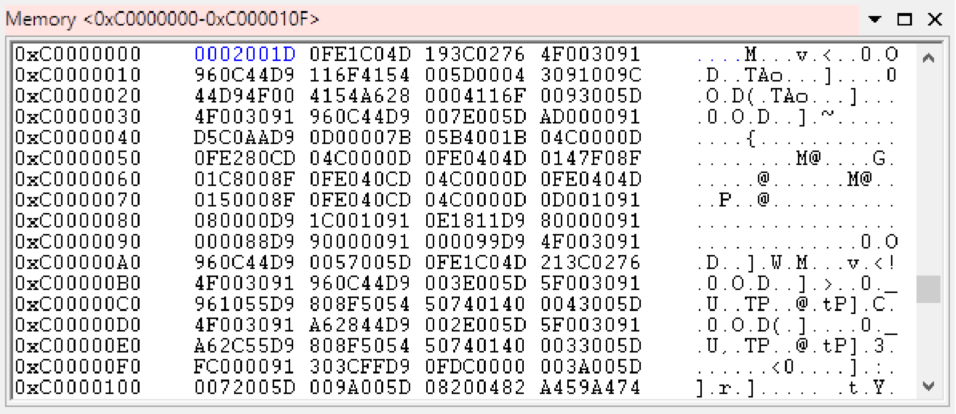
Viewing and changing local variables

Viewing local variables is provided by the Locals window that can be reached via the menu **Views – Locals**. In this window, all currently valid local variables are displayed. The value of the local variable can be changed in the same way as in the Watches window.

Viewing Memory Locations

The Memory window, which is available via **Views – Memory**, displays memory areas in the AURIX's memory space and provides means to change. Select the displayed memory area by setting the cursor in the address column of the Memory window and type in the new address. Memory content can be changed by clicking on a memory location value and typing in the new value. Changed values are marked blue before they are written back to the memory.

The Memory window supports the viewing modes as Byte, Word, DWord, Double, Fract and the ASCII representation of the memory area.



Memory <0xC0000000-0xC000010F>

```
0xC0000000     0002001D 0FE1C04D 193C0276 4F003091     ....M...v.<..0.O
0xC0000010     960C44D9 116F4154 005D0004 3091009C     .D..TAo...]....0
0xC0000020     44D94F00 4154A628 0004116F 0093005D     .O.D(.TAo...]...
0xC0000030     4F003091 960C44D9 007E005D AD000091     .0.O.D..].~.....
0xC0000040     D5C0AAD9 0D00007B 05B4001B 04C0000D     ....{...........
0xC0000050     0FE280CD 04C0000D 0FE0404D 0147F08F     ........M@....G.
0xC0000060     01C8008F 0FE040CD 04C0000D 0FE0404D     .....@......M@..
0xC0000070     0150008F 0FE040CD 04C0000D 0D001091     ..P..@..........
0xC0000080     080000D9 1C001091 0E1811D9 80000091     ................
0xC0000090     000088D9 90000091 000099D9 4F003091     ..............0.O
0xC00000A0     960C44D9 0057005D 0FE1C04D 213C0276     .D..].W.M...v.<!
0xC00000B0     4F003091 960C44D9 003E005D 5F003091     .0.O.D..].>..0._
0xC00000C0     961055D9 808F5054 50740140 0043005D     .U..TP..@.tP].C.
0xC00000D0     4F003091 A62844D9 002E005D 5F003091     .0.O.D(.]....0._
0xC00000E0     A62C55D9 808F5054 50740140 0033005D     .U,.TP..@.tP].3.
0xC00000F0     FC000091 303CFFD9 0FDC0000 003A005D     ......<0....].:.
0xC0000100     0072005D 009A005D 08200482 A459A474     ].r.]......t.Y.
```

Leaving the Project

To leave the current Project, select **File – Close Workspace** from the UDE menu. The workspace with all settings will be saved automatically. If you want to save the current project under a different project name, select **Save Workspace As ...** from **File** menu. In the file selection box, the new workspace name must be selected and confirmed.

A Multi-Core Debugging Example with TriCore/PCP

The HelloPCP program provides an example of how to use the PCP (Peripheral Communication Processor) of the TriCore chip. It demonstrates the multi-processor support of UDE. When running, a flashing LED on the TriBoard is displayed while the flashing pattern itself is defined by a string in the source code. The string is translated into Morse code, which provides the flash pattern of the LED.

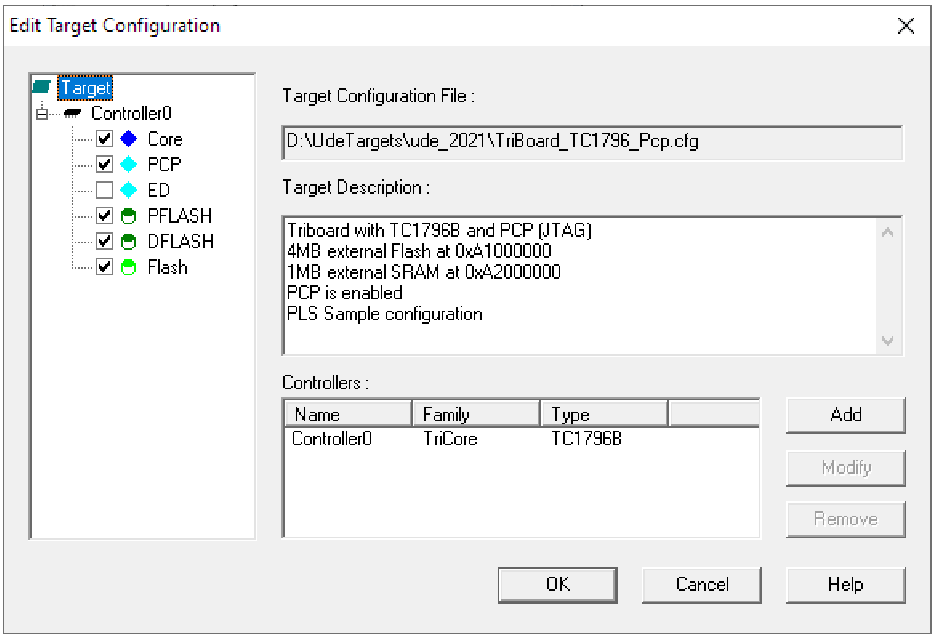
Please note in this example the PCP is the working horse serving the interrupts of the GPTU. The PCP code is contained in the source file morse.pcp. The TriCore itself is used only to initialize the PCP.

Creating a New Workspace with changed configuration

At first, a new workspace has to be created. Click **New Workspace** from the **File** menu and choose a new file name in the opening file selection box. The next step extends the target configuration with the PCP controller.

UDE provides a number of prepared target configurations. For this example, the TriBoard TC1796 configuration has to be copied and changed. Select the entry **TriBoard with TC1796 (JTAG)** and click **Copy**. In the configuration box, choose a new file, e.g. TriBoard_TC1796_PCP.cfg.

The Target Configuration Dialog will be opened. This dialog gives the user the possibility to change the target hardware descriptions and settings.



- Target
  - Controller0
    - ☑ Core
    - ☑ PCP
    - ☐ ED
    - ☑ PFLASH
    - ☑ DFLASH
    - ☑ Flash

Target Configuration File : D:\UdeTargets\ude_2021\TriBoard_TC1796_Pcp.cfg

Target Description :

Triboard with TC1796B and PCP (JTAG)
4MB external Flash at 0xA1000000
1MB external SRAM at 0xA2000000
PCP is enabled
PLS Sample configuration

Controllers :

Name	Family	Type
Controller0	TriCore	TC1796B

Select the PCP entry as shown above, it enables the PCP debugging. Change the Target Description to **.. TC1796B and PCP (JTAG)**. Click **OK**.

Your workspace will be reconfigured and the workspace view shows two core debugger entries.

Every window is connected to one of the cores only, except the command window. To distinct between the cores, each window displays the core name as part of the descriptor in the title bar, i.e. Controller0.Core and Controller0.Pcp.

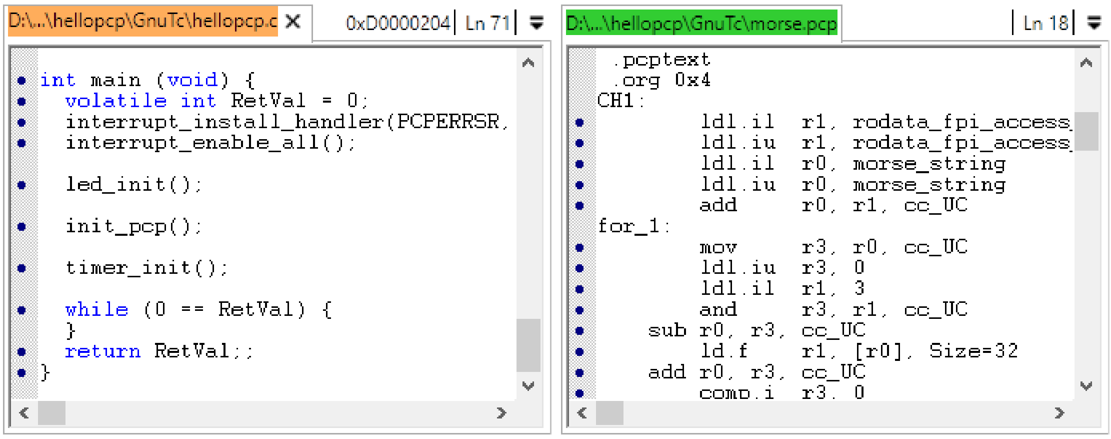
If the connection is successfully established, the following messages will appear in the Messages window:

```
Connection to TC1796 target monitor established: TriCore(Core),
       ID: 200B8083h
Connection to TC1796 target monitor established: TriCore(PCP),
       ID: 200B8083h
```

When both debuggers are launched, UDE is ready for loading the application file `HelloPCP.elf`.

Running the Program

For loading `HelloPCP.elf` enable only the Binary and the Symbols for `Controller0.Core` in the Multicore / multi program loader dialog. After loading `HelloPCP.elf` into the TriCore target system, two program windows are displayed. One of them is the PCP program window, while the other is the TriCore program window. Toolbar Buttons and menu commands always relate to the active debugger window.


D:\...\hellopcp\GnuTc\hellopcp.c
0xD0000204
Ln 71
D:\...\hellopcp\GnuTc\morse.pcp
Ln 18

```
int main (void) {
  volatile int RetVal = 0;
  interrupt_install_handler(PCPERRSR,
  interrupt_enable_all();

  led_init();

  init_pcp();

  timer_init();

  while (0 == RetVal) {
  }
  return RetVal;;
}
```

```
 .pcptext
 .org 0x4
CH1:
        ldl.il  r1, rodata_fpi_access_
        ldl.iu  r1, rodata_fpi_access_
        ldl.il  r0, morse_string
        ldl.iu  r0, morse_string
        add     r0, r1, cc_UC
for_1:
        mov     r3, r0, cc_UC
        ldl.iu  r3, 0
        ldl.il  r1, 3
        and     r3, r1, cc_UC
    sub r0, r3, cc_UC
        ld.f    r1, [r0], Size=32
    add r0, r3, cc_UC
        comp.i  r3, 0
```

To debug the PCP program, set a breakpoint in the PCP code, switch to the Core Window and execute a **Step over Subroutine**. This will initialize the PCP. Then click on **Start Program Execution** of the TriCore code.

Please note, that the sequence described above must be executed exactly in this order. The background is that the start-up code transfers the PCP code to the PCP memory and overwrites any software breakpoints. To give the UDE the possibility to set a breakpoint to the PCP, UDE must have access to the PCP after the initialization. That is why the first step must be a **Step over Subroutine**!

The PCP program will stop at the breakpoint. Now you may for example step through the code, continue running the PCP program or watch the PCP registers.

HelloPCP Internals

The TriCore program initializes the GPTU0, the PCP and the port P0. At the end of the program, the GPTU0 timer is started to provide periodic interrupts to execute the PCP routines.

The PCP program may be interpreted as state machine. Each interrupt causes the PCP to start in the following state (EXIT instruction with restart at next address). Each state ends with setting GPTU0 registers to proper values. When the timer of GPTU0 overflows, channel 1 of the PCP is restarted. This way, the PCP program will run infinitely and continuously send the string defined by

```
const char morse_string[] ...
```

A Multi-Core Debugging Example with AURIX TC2xx

Infineon’s 32-bit multi-core architecture AURIX (AUtomotive Realtime Integrated neXt generation architecture) based MCU, part number TC277D, incorporates three TriCore processor cores (version 1.6). The following chapter describes the debugging options for three cores of the example architecture AURIX. Multi-core debugging for other architectures is supported in a similar way.

Please run **UDE** from the Window’s start menu.

Understanding a Multi-Core Configuration

UDE supports several scenarios for Multi-Core debugging:

1. Several cores on one Chip using the same JTAG-Chain (AURIX)
2. Several cores on different Chips using the same JTAG-Chain
3. Several cores on different Chips using different Debug-Channels

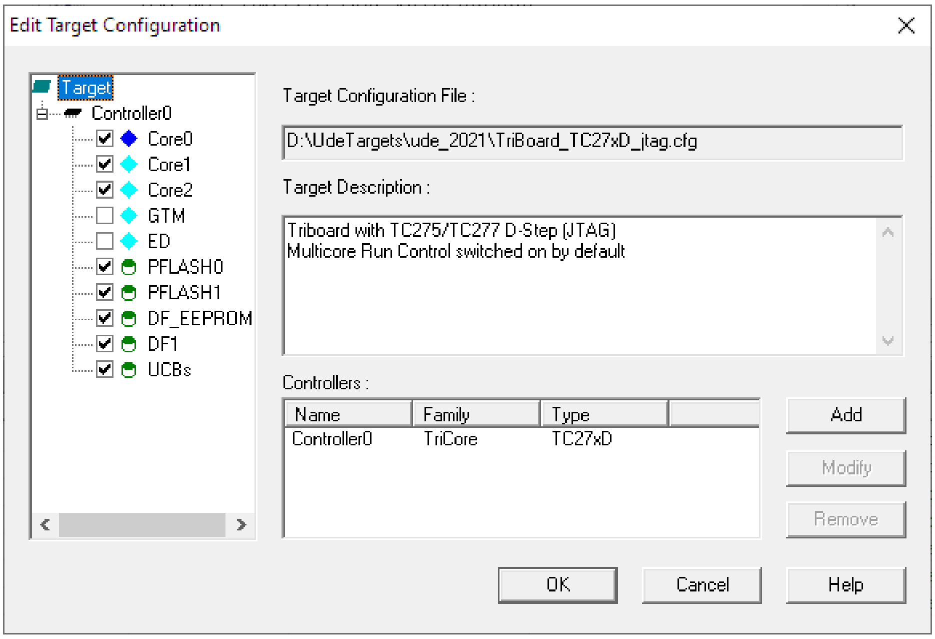
The configuration is unchanged during a debug session and is stored in the `*.cfg` -File. Configuration settings are managed via the **Target Configuration Wizard**.

Creating a New Workspace

At first, a new workspace has to be created. Click **New Workspace** from the **File** menu and choose a new file name in the opening file selection box.

UDE provides a number of prepared target configurations (**Default** button). This example uses the TriBoard TC277D configuration. Select the entry **TriCore AURIX** – **Infineon** – **TC27xD Starterkit** – **TriBoard with TC275/TC277 D-Step (JTAG)** and click on **Finish**. In the configuration box, choose a file name, e.g. `TriBoard_TC27xD_jtag.cfg`.

Select the created configuration and click on **Edit** to modify the target configuration.



Edit Target Configuration

- Target
  - Controller0
    - ☑ Core0
    - ☑ Core1
    - ☑ Core2
    - ☐ GTM
    - ☐ ED
    - ☑ PFLASH0
    - ☑ PFLASH1
    - ☑ DF_EEPROM
    - ☑ DF1
    - ☑ UCBs

Target Configuration File :

D:\UdeTargets\ude_2021\TriBoard_TC27xD_jtag.cfg

Target Description :

Triboard with TC275/TC277 D-Step (JTAG)
Multicore Run Control switched on by default

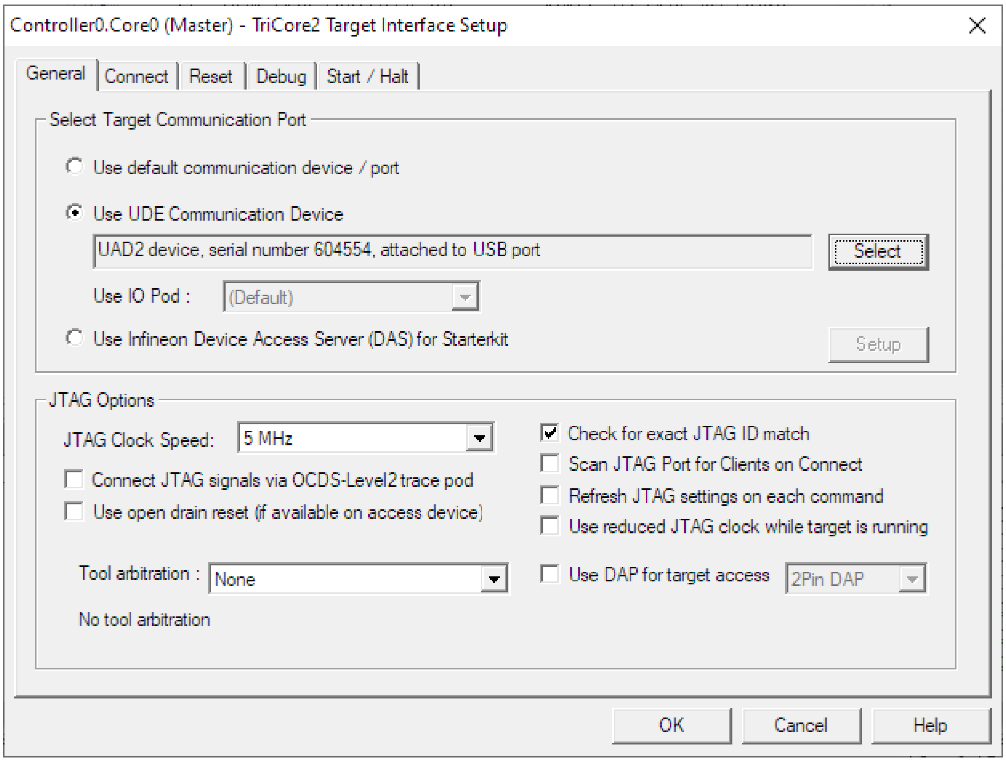
Controllers :

Name	Family	Type
Controller0	TriCore	TC27xD

The default target configuration contains all information about the Core0, Core1, Core2, GTM, ED and FLASH devices. Core0 is the master core (TriCore1.6P), Core1 and Core2 are slave cores (TriCore1.6E) of Core0. GTM support is disabled in this example.

To change a target or a core-specific property, select one of the Controller0 items and click the **Setup** button.

The **General** page of Core0 selects the communication device and contains the JTAG access options. If more than one communication device is connected to the PC, it can be specified via **Use UDE Communication Device** which one should be used. Otherwise, the option **Use default device / port** is suitable.



Controller0.Core0 (Master) - TriCore2 Target Interface Setup

General | Connect | Reset | Debug | Start / Halt

Select Target Communication Port

○ Use default communication device / port

● Use UDE Communication Device

UAD2 device, serial number 604554, attached to USB port [Select]

Use IO Pod : (Default)

○ Use Infineon Device Access Server (DAS) for Starterkit [Setup]

JTAG Options

JTAG Clock Speed: 5 MHz

□ Connect JTAG signals via OCDS-Level2 trace pod

□ Use open drain reset (if available on access device)

Tool arbitration : None

No tool arbitration

☑ Check for exact JTAG ID match

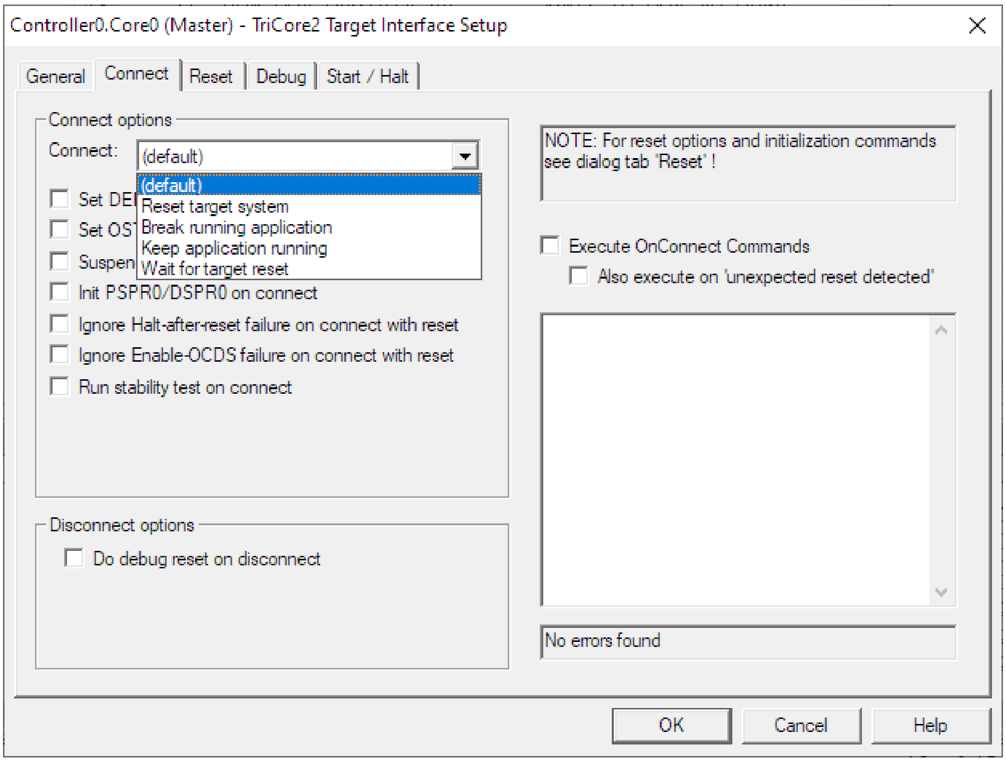
□ Scan JTAG Port for Clients on Connect

□ Refresh JTAG settings on each command

□ Use reduced JTAG clock while target is running

□ Use DAP for target access 2Pin DAP

The page **Connect** contains the connect options and the initialization commands on connect.



Controller0.Core0 (Master) - TriCore2 Target Interface Setup

General | Connect | Reset | Debug | Start / Halt

Connect options

Connect: (default)

(default)
Reset target system
Break running application
Keep application running
Wait for target reset

□ Set DE
□ Set OS
□ Suspen
□ Init PSPR0/DSPR0 on connect
□ Ignore Halt-after-reset failure on connect with reset
□ Ignore Enable-OCDS failure on connect with reset
□ Run stability test on connect

Disconnect options

□ Do debug reset on disconnect

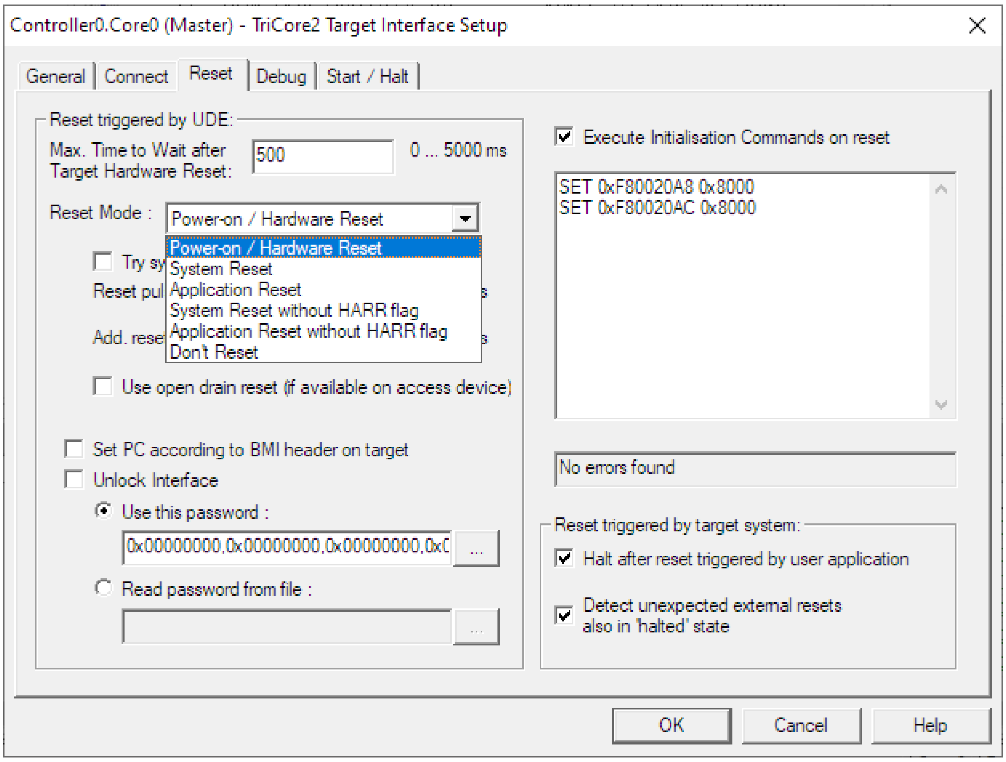
NOTE: For reset options and initialization commands see dialog tab 'Reset' !

□ Execute OnConnect Commands

□ Also execute on 'unexpected reset detected'

No errors found

The page **Reset** contains the reset mode and the initialization commands on reset.



Controller0.Core0 (Master) - TriCore2 Target Interface Setup

Reset triggered by UDE:
Max. Time to Wait after Target Hardware Reset: 500 0 ... 5000 ms
Reset Mode : Power-on / Hardware Reset
- Power-on / Hardware Reset
- System Reset
- Application Reset
- System Reset without HARR flag
- Application Reset without HARR flag
- Don't Reset

□ Try sy
Reset pul
Add. rese
□ Use open drain reset (if available on access device)
□ Set PC according to BMI header on target
□ Unlock Interface
● Use this password : 0x00000000,0x00000000,0x00000000,0x0
○ Read password from file :

☑ Execute Initialisation Commands on reset

```
SET 0xF80020A8 0x8000
SET 0xF80020AC 0x8000
```

No errors found

Reset triggered by target system:
☑ Halt after reset triggered by user application
☑ Detect unexpected external resets also in 'halted' state

The pages **Debug** and **Start / Halt** contain more debug options and the start/halt commands. Please see the UDE help description for more information about the meaning of each setting.

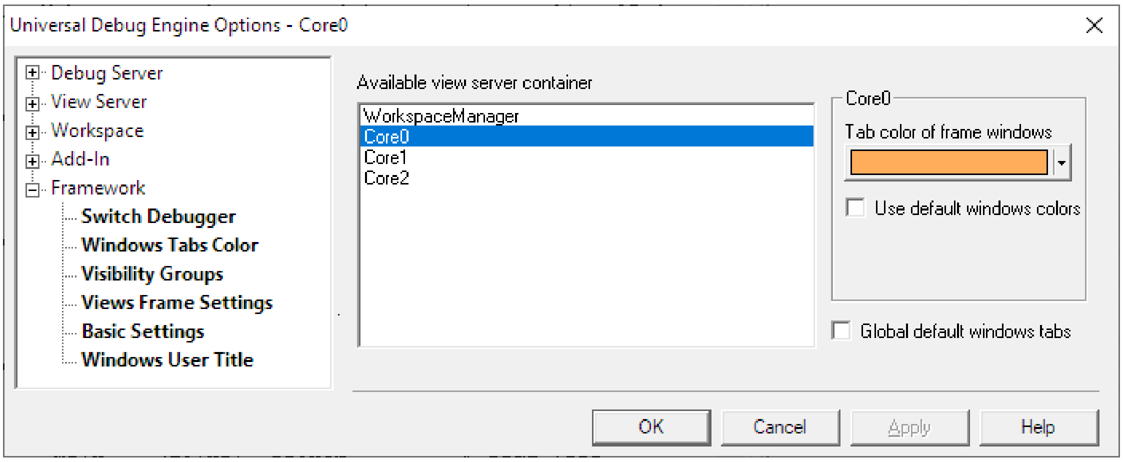
Leave the **Target Interface Setup** and click **OK**. If the connection is established successfully, the following messages will appear in the Messages window:

```
    Connection to TC27xD target established: TriCore(Core0),
           ID: 501DA083h
    Connection to TC27xD target established: TriCore(Core1)
    Connection to TC27xD target established: TriCore(Core2)
```

If all debuggers are launched, UDE is ready for loading the multi-core application `Timedemo.elf`.

Preparing the Debugger

Windows, which are associated to a single core, use a default coloring of the title bar and/or tabs in tabbed window containers. This default coloring can be customized to the user's needs. Open the menu **Config – Debug Server Configuration – Framework – Windows Tabs Color**. Select one of the cores and change the assigned color. The selected color is now used for each window associated with the selected core.



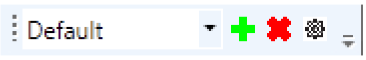
Universal Debug Engine Options - Core0

Available view server container
- WorkspaceManager
- Core0
- Core1
- Core2

Core0
Tab color of frame windows: 🟧
□ Use default windows colors
□ Global default windows tabs

Show, Hide and Group Windows-Related Perspectives

Debugging a multi-core system with many open debugger windows may lead into a very confusing debug session. Perspectives are helping to keep the debug session clearly. A perspective is a container for windows, which is visible at a certain point of time.

Perspectives can be activated using the toolbar menu. 

The '+' adds a new perspective, the 'X' removes it, the 'gear wheel' configures the visibility of the toolbars, the dropdown menu selects a specific perspective.

A new perspective is cloned from the current active perspective, but can be adapted to the requirements.

Loading a Multi-Core Executable

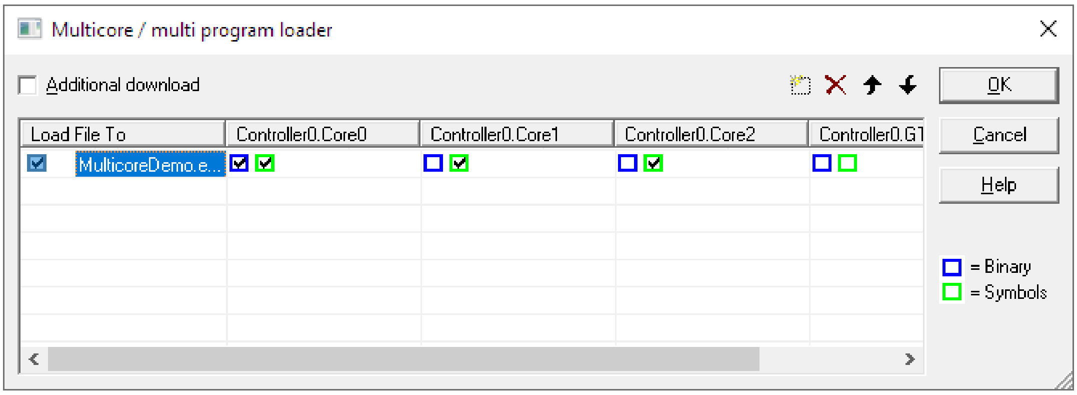
UDE supports several options to handle executables for multi-core architectures:

1. One executable for all cores, combined in a single binary file.
2. Separate executables for each core, combined in a single binary file.
3. Separate binary files for each core.

UDE handles these requirements in a multi-core / multi-program loader box, which provides a switch matrix to assign executables to cores. Additionally, to selecting binary files for cores, the dialog also provides switches to assign symbol information containing files.

Select from the UDE menu **File – Load Program** and load the file `<UDE_SAMPLE_DIRECTORY>\TriCore2\TriBoard_TC27x\MulticoreDemo\HighTec_IntRam \TimeDemo.elf`.

Because this is an example of a single executable, which is executed by each core, there is only one entry for `MulticoreDemo.elf`. Select the binary check marks only for **Core 0** (master core) in the **Controller0.Core0** column, to avoid loading the binary code twice. Select the symbol check marks for each core to assign symbolic information to each core.



Multicore / multi program loader

□ Additional download

Load File To	Controller0.Core0	Controller0.Core1	Controller0.Core2	Controller0.G1
☑ MulticoreDemo.e...	☑ ☑	□ ☑	□ ☑	□ □

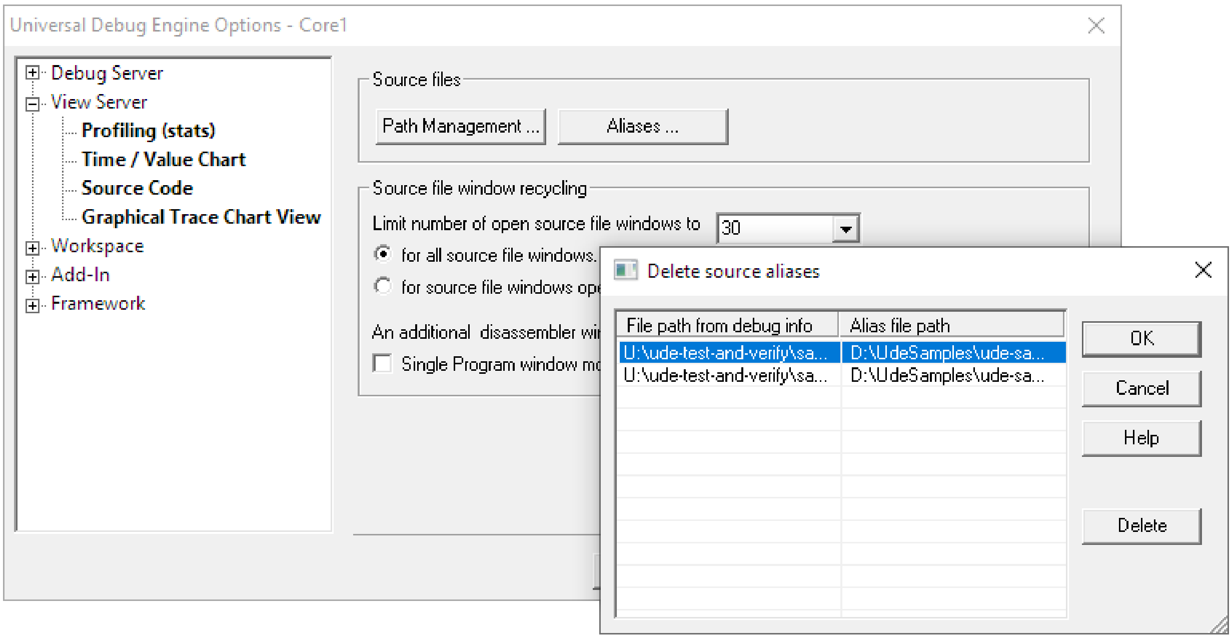
UDE will load and transfer all binaries into the corresponding core and the symbols to the debugger symbol server.

Source File Management

If the debugger cannot find the source files specified in the symbol containing files, it can request the location of the correct source path. This can happen, if the compiler's-built environment differs from the environment used by the debugging process.

The examples source files are located in the `\src` and `\board` subfolders. You need to provide the location on request. UDE will remember your decision and will not ask again for this source path.

Every entry of this relocated source is called **Alias**. The saved aliases of a workspace are accessible via the menu **Config – Debug Server Configuration – View Server – Source Code – Aliases …**


Universal Debug Engine Options - Core1

- Debug Server
- View Server
  - Profiling (stats)
  - Time / Value Chart
  - Source Code
  - Graphical Trace Chart View
- Workspace
- Add-In
- Framework

Source files

Path Management ... | Aliases ...

Source file window recycling

Limit number of open source file windows to: 30

● for all source file windows.

○ for source file windows ope

An additional disassembler wi

□ Single Program window mo

Delete source aliases

File path from debug info	Alias file path
U:\ude-test-and-verify\sa...	D:\UdeSamples\ude-sa...
U:\ude-test-and-verify\sa...	D:\UdeSamples\ude-sa...

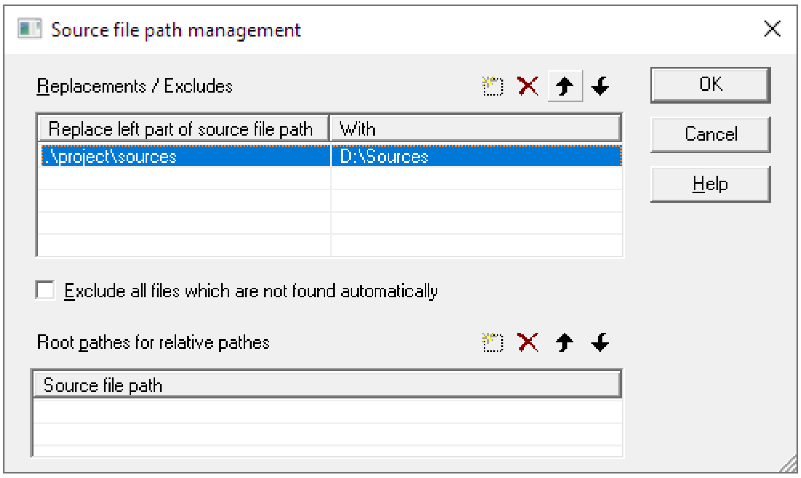
OK | Cancel | Help | Delete

In some cases, the forced source alias may have changed. Use the dialog to delete the affected entry. The next time the source file is used, UDE will ask for the location again.

Source Path Replacement

The behavior described above is useful, if only few source file locations have changed. If a complete source file tree is affected, you can give UDE a replacement path, which will be applied to all source file paths.

Use the menu **Config – Debug Server Configuration – View Server – Source Code** – **Path Management …**


Source file path management

Replacements / Excludes

Replace left part of source file path	With
.\project\sources	D:\Sources

□ Exclude all files which are not found automatically

Root pathes for relative pathes

Source file path

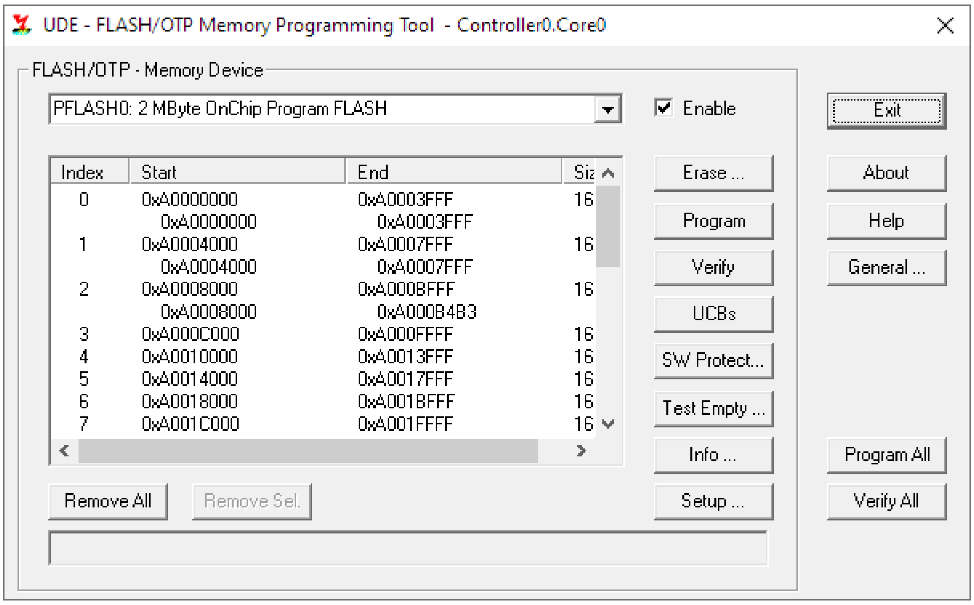
OK | Cancel | Help

The dialog defines a part for replacing and a part for including the new source location.

You will now see the source code of the main function from the sample application. When clicking with the right-hand mouse button into the Program window, a context button appears to switch between Source code only and Source/Assembly code display via the **Mixed Mode** entry.

FLASH programming

In AURIX TC2xx architecture, only core0 is used to program FLASH memory. The **FLASH/OTP Programming Tool** will be opened automatically, if code has to be downloaded to FLASH memory. It can be opened manually, via menu **Tools** – **FLASH Programming**.



UDE - FLASH/OTP Memory Programming Tool - Controller0.Core0

FLASH/OTP - Memory Device

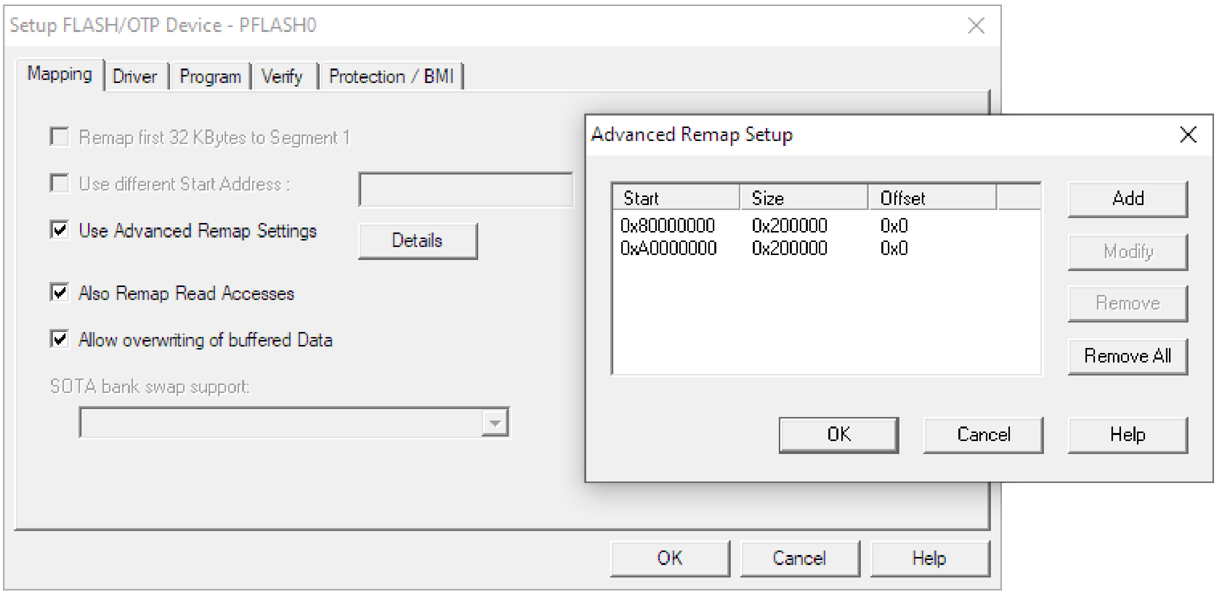
PFLASH0: 2 MByte OnChip Program FLASH

☑ Enable

Index	Start	End	Siz
0	0xA0000000 0xA0000000	0xA0003FFF 0xA0003FFF	16
1	0xA0004000 0xA0004000	0xA0007FFF 0xA0007FFF	16
2	0xA0008000 0xA0008000	0xA000BFFF 0xA000B4B3	16
3	0xA000C000	0xA000FFFF	16
4	0xA0010000	0xA0013FFF	16
5	0xA0014000	0xA0017FFF	16
6	0xA0018000	0xA001BFFF	16
7	0xA001C000	0xA001FFFF	16

Use the buttons **Program All** and **Verify All** to program and verify all FLASH memory at once.

The **FLASH/OTP Programming Tool** supports **Remap settings** for download into cached regions. Click on **Setup ...** button and select the page **Mapping**.



Setup FLASH/OTP Device - PFLASH0

☐ Remap first 32 KBytes to Segment 1

☐ Use different Start Address :

☑ Use Advanced Remap Settings

☑ Also Remap Read Accesses

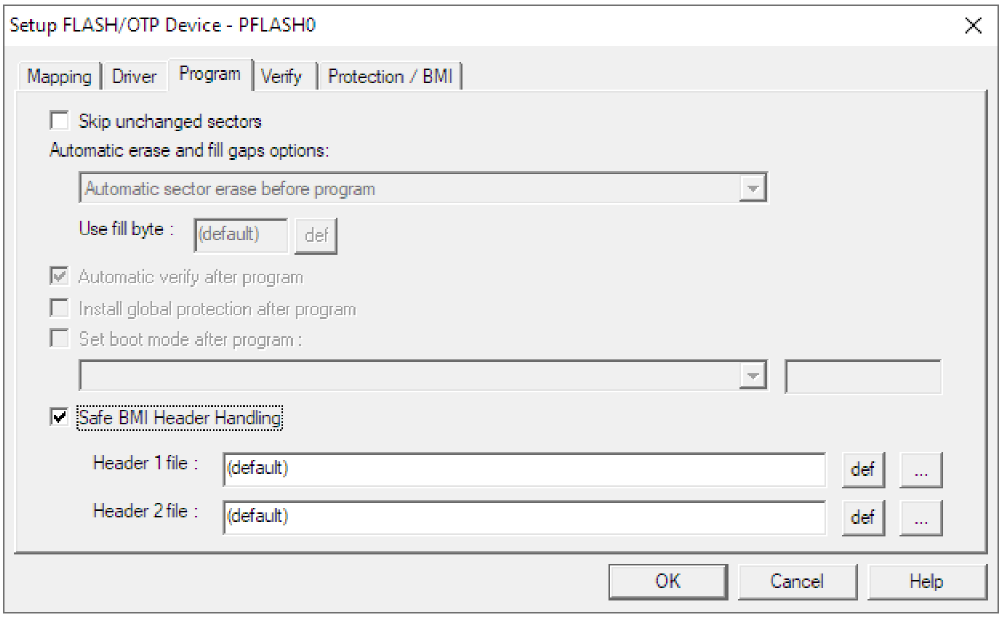
☑ Allow overwriting of buffered Data

SOTA bank swap support:

Advanced Remap Setup

Start	Size	Offset
0x80000000	0x200000	0x0
0xA0000000	0x200000	0x0

The **FLASH/OTP Programming Tool** supports the **Safe BMI header** handling, too.



Setup FLASH/OTP Device - PFLASH0

☐ Skip unchanged sectors

Automatic erase and fill gaps options: Automatic sector erase before program

Use fill byte : (default)

☑ Automatic verify after program

☐ Install global protection after program

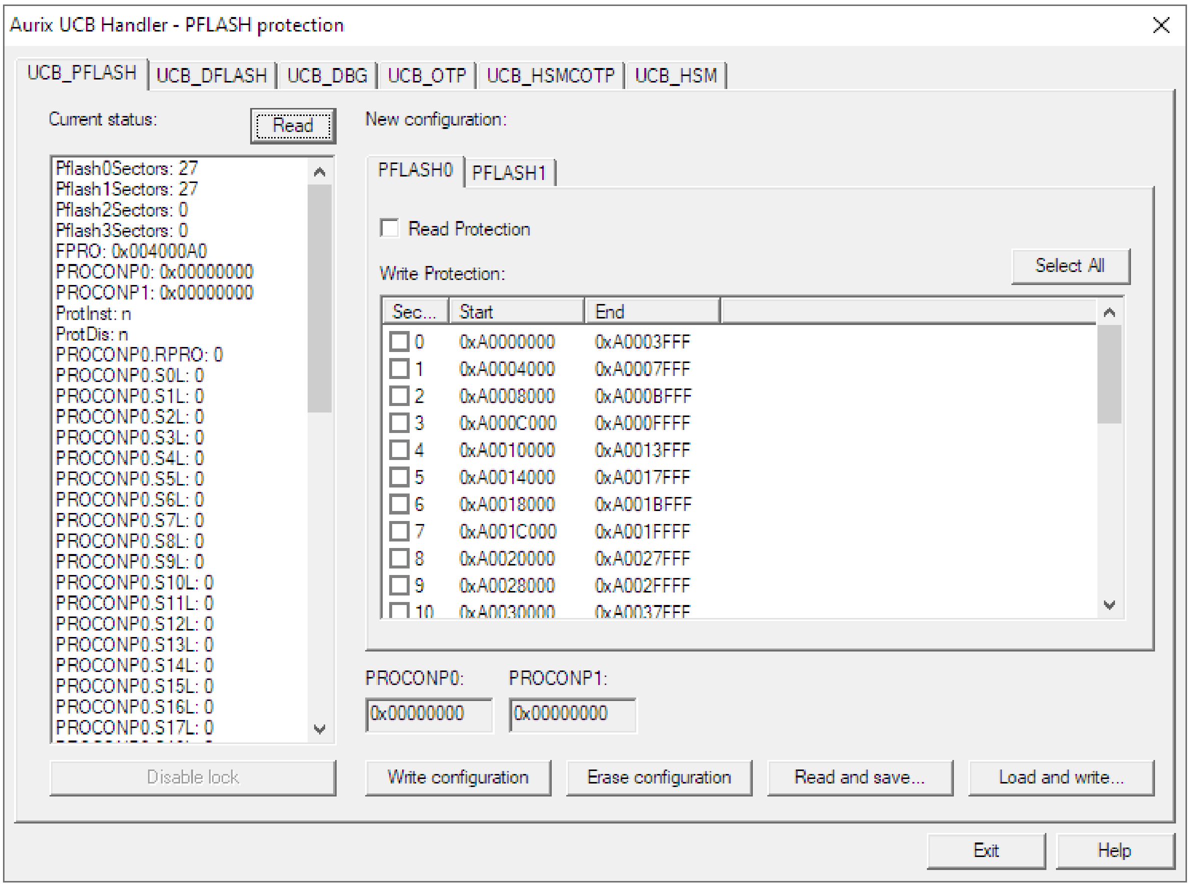
☐ Set boot mode after program :

☑ Safe BMI Header Handling

Header 1 file : (default)

Header 2 file : (default)

The **AURIX UCB** handling is also supported via the **UCB** button of the **FLASH/OTP Programming Tool**.



Aurix UCB Handler - PFLASH protection

Current status:

```
Pflash0Sectors: 27
Pflash1Sectors: 27
Pflash2Sectors: 0
Pflash3Sectors: 0
FPRO: 0x004000A0
PROCONP0: 0x00000000
PROCONP1: 0x00000000
ProtInst: n
ProtDis: n
PROCONP0.RPRO: 0
PROCONP0.S0L: 0
PROCONP0.S1L: 0
PROCONP0.S2L: 0
PROCONP0.S3L: 0
PROCONP0.S4L: 0
PROCONP0.S5L: 0
PROCONP0.S6L: 0
PROCONP0.S7L: 0
PROCONP0.S8L: 0
PROCONP0.S9L: 0
PROCONP0.S10L: 0
PROCONP0.S11L: 0
PROCONP0.S12L: 0
PROCONP0.S13L: 0
PROCONP0.S14L: 0
PROCONP0.S15L: 0
PROCONP0.S16L: 0
PROCONP0.S17L: 0
```

New configuration:

☐ Read Protection

Write Protection:

Sec...	Start	End
☐ 0	0xA0000000	0xA0003FFF
☐ 1	0xA0004000	0xA0007FFF
☐ 2	0xA0008000	0xA000BFFF
☐ 3	0xA000C000	0xA000FFFF
☐ 4	0xA0010000	0xA0013FFF
☐ 5	0xA0014000	0xA0017FFF
☐ 6	0xA0018000	0xA001BFFF
☐ 7	0xA001C000	0xA001FFFF
☐ 8	0xA0020000	0xA0027FFF
☐ 9	0xA0028000	0xA002FFFF
☐ 10	0xA0030000	0xA0037FFF

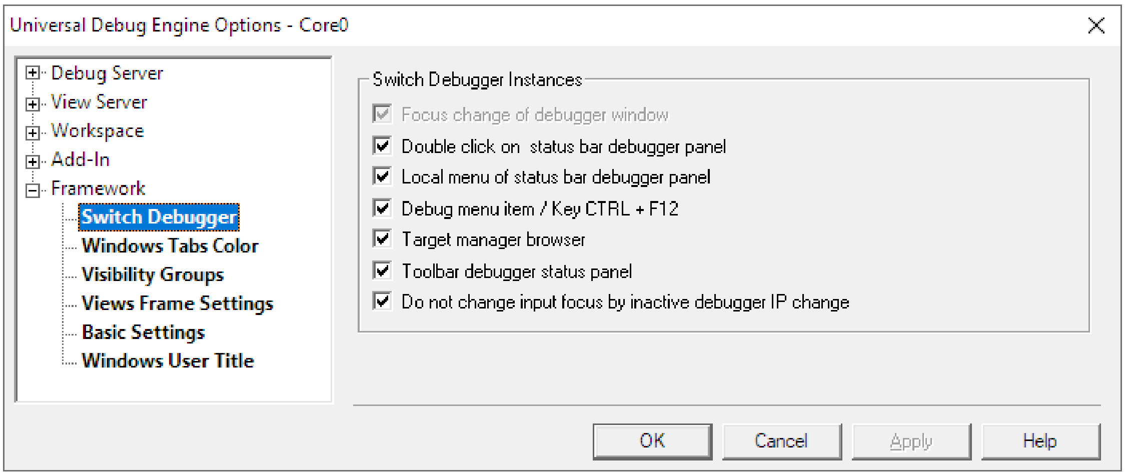
PROCONP0: 0x00000000

PROCONP1: 0x00000000

Core Selection

To select a specific core debugger, click into a core-specific window (with a colored frame and/or tab) and the focus will be on this core debugger. The **status bar** at the bottom of the UDE workbench shows the currently selected core debugger (marked red in the screenhot below). The state bar field also provides a context menu (right-click) for selection of a core debugger. At last, the context menu of a specific core in the **Target Manager** windows contains an entry to activate and focus this core.

This behavior of UDE is highly configurable. Use the menu **Config** – **Debug Server Configuration** – **Framework** – **Switch Debugger** to explore all options.

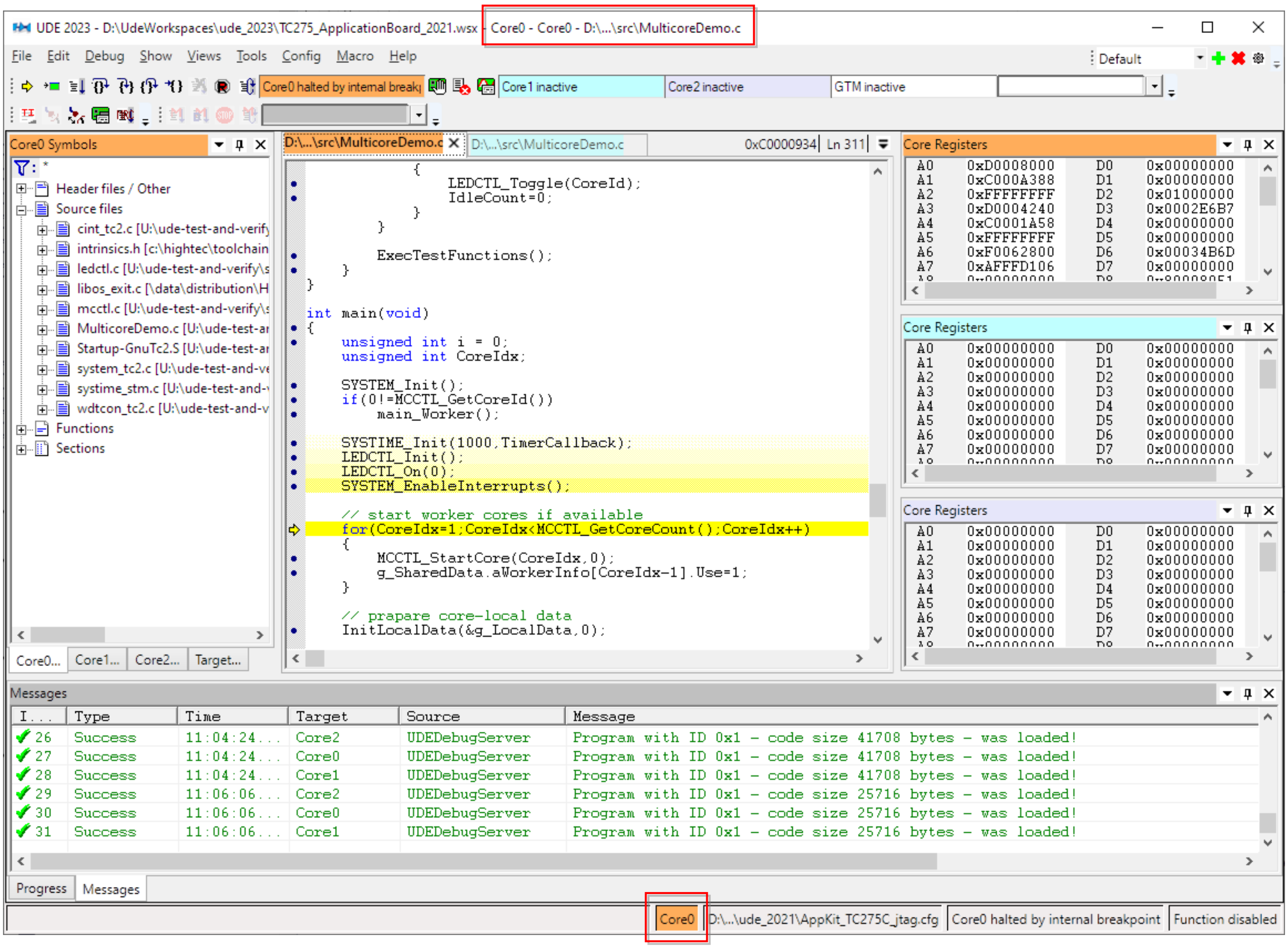


Universal Debug Engine Options - Core0

Switch Debugger Instances

- ☑ Focus change of debugger window
- ☑ Double click on status bar debugger panel
- ☑ Local menu of status bar debugger panel
- ☑ Debug menu item / Key CTRL + F12
- ☑ Target manager browser
- ☑ Toolbar debugger status panel
- ☑ Do not change input focus by inactive debugger IP change

This screenshot shows an example configuration for a multicore workbench:



UDE 2023 - D:\UdeWorkspaces\ude_2023\TC275_ApplicationBoard_2021.wsx - Core0 - Core0 - D:\...\src\MulticoreDemo.c

File Edit Debug Show Views Tools Config Macro Help
Default
Core0 halted by internal breakp
Core1 inactive
Core2 inactive
GTM inactive
Core0 Symbols
Header files / Other
Source files
cint_tc2.c [U:\ude-test-and-verify
intrinsics.h [c:\hightec\toolchain
ledctl.c [U:\ude-test-and-verify\s
libos_exit.c [\data\distribution\H
mcctl.c [U:\ude-test-and-verify\s
MulticoreDemo.c [U:\ude-test-ar
Startup-GnuTc2.S [U:\ude-test-ar
system_tc2.c [U:\ude-test-and-ve
systime_stm.c [U:\ude-test-and-
wdtcon_tc2.c [U:\ude-test-and-v
Functions
Sections
Core0... Core1... Core2... Target...
D:\...\src\MulticoreDemo.c
D:\...\src\MulticoreDemo.c
0xC0000934 Ln 311
Core Registers
Core Registers
Core Registers
Messages
Progress Messages
Core0
D:\...\ude_2021\AppKit_TC275C_jtag.cfg
Core0 halted by internal breakpoint
Function disabled

```
            {
                LEDCTL_Toggle(CoreId);
                IdleCount=0;
            }
        }

        ExecTestFunctions();
    }
}

int main(void)
{
    unsigned int i = 0;
    unsigned int CoreIdx;

    SYSTEM_Init();
    if(0!=MCCTL_GetCoreId())
        main_Worker();

    SYSTIME_Init(1000,TimerCallback);
    LEDCTL_Init();
    LEDCTL_On(0);
    SYSTEM_EnableInterrupts();

    // start worker cores if available
    for(CoreIdx=1;CoreIdx<MCCTL_GetCoreCount();CoreIdx++)
    {
        MCCTL_StartCore(CoreIdx,0);
        g_SharedData.aWorkerInfo[CoreIdx-1].Use=1;
    }

    // prapare core-local data
    InitLocalData(&g_LocalData,0);
```

```
A0   0xD0008000   D0   0x00000000
A1   0xC000A388   D1   0x00000000
A2   0xFFFFFFFF   D2   0x01000000
A3   0xD0004240   D3   0x0002E6B7
A4   0xC0001A58   D4   0x00000000
A5   0xFFFFFFFF   D5   0x00000000
A6   0xF0062800   D6   0x00034B6D
A7   0xAFFFD106   D7   0x00000000
```

```
A0   0x00000000   D0   0x00000000
A1   0x00000000   D1   0x00000000
A2   0x00000000   D2   0x00000000
A3   0x00000000   D3   0x00000000
A4   0x00000000   D4   0x00000000
A5   0x00000000   D5   0x00000000
A6   0x00000000   D6   0x00000000
A7   0x00000000   D7   0x00000000
```

```
A0   0x00000000   D0   0x00000000
A1   0x00000000   D1   0x00000000
A2   0x00000000   D2   0x00000000
A3   0x00000000   D3   0x00000000
A4   0x00000000   D4   0x00000000
A5   0x00000000   D5   0x00000000
A6   0x00000000   D6   0x00000000
A7   0x00000000   D7   0x00000000
```

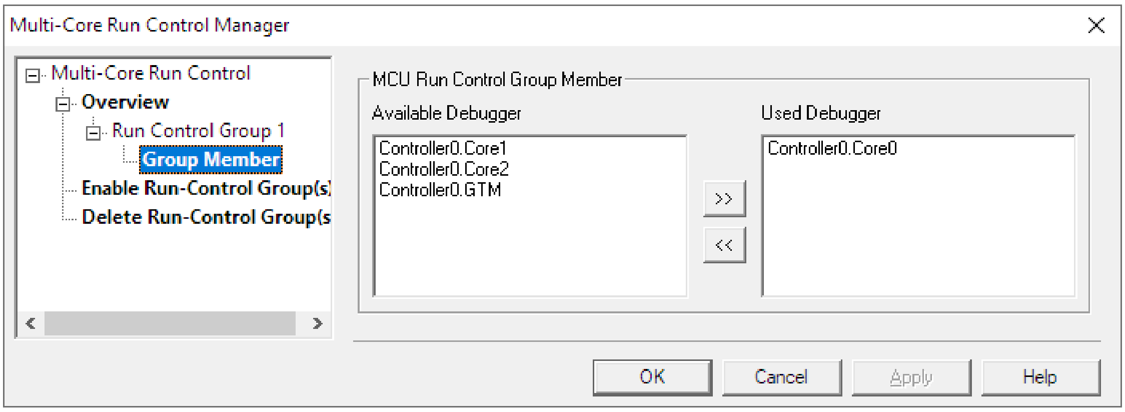
I...	Type	Time	Target	Source	Message
✓ 26	Success	11:04:24...	Core2	UDEDebugServer	Program with ID 0x1 - code size 41708 bytes - was loaded!
✓ 27	Success	11:04:24...	Core0	UDEDebugServer	Program with ID 0x1 - code size 41708 bytes - was loaded!
✓ 28	Success	11:04:24...	Core1	UDEDebugServer	Program with ID 0x1 - code size 41708 bytes - was loaded!
✓ 29	Success	11:06:06...	Core2	UDEDebugServer	Program with ID 0x1 - code size 25716 bytes - was loaded!
✓ 30	Success	11:06:06...	Core0	UDEDebugServer	Program with ID 0x1 - code size 25716 bytes - was loaded!
✓ 31	Success	11:06:06...	Core1	UDEDebugServer	Program with ID 0x1 - code size 25716 bytes - was loaded!

Core0 is halted and focused, Core1 and Core2 (slave cores) are inactive and not focused (see the red marking). A Core Registers window is opened for each core.

Single-Core Breakpoints in a multi-core environment

This single-core breakpoint example will set a breakpoint for the specific Core1, where the Core1 will only halt if that core runs over the **single-core breakpoint** code. If the other cores run over the same breakpoint address, nothing will happen.

The behavior of the other cores in the case of a stopped Core1 depends from the settings of the multi-core run control group. All cores of the same group as Core1 are stopped simultaneously. Use menu **Config – MultiCore Run Control Configuration** and set, that the **Run Control Group 1** has only member Controller0.Core0, the master core. So, there is Controller0.Core1 in no other group contained and Controller0.Core1 will stopped as single core.


Multi-Core Run Control Manager

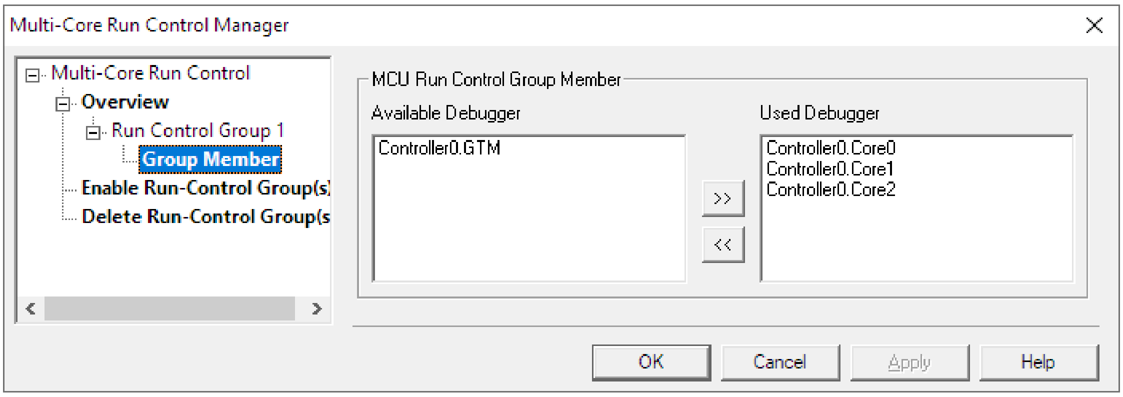
Available Debugger	Used Debugger
Controller0.Core1 Controller0.Core2 Controller0.GTM	Controller0.Core0

To set the core-specific single-core breakpoint, open the source of the specific core. If the program window for Core1 is not open already, use the **Core1 Symbols** window and click on the source file name MulticoreDemo.cpp to open the corresponding **Program window**. Right-click on line 319

```
SYSTEM_Init();
```

in the program window of MulticoreDemo.cpp and use the context menu **Insert a Single Breakpoint** to set a single-core breakpoint. Select Core0 (see section **Core Selection** above) again, because only this core can be used for initialization in the AURIX TC2xx architecture and start the program via **Start Program Execution**. This will run the MulticoreDemo.elf on Core0. Core0 will start the slave cores Core1 und Core2 automatically. After a while, Core1 stops at the breakpoint (halted by user breakpoint), while Core0 (running) and Core2 (running) are still running. After inspection of Core1, it can be started again via **Start Program Execution**.

If it is wanted that all cores stop when Core1 stops, add Controller0.Core1 and Controller0.Core2 into the run control group of Controller0.Core0.


Multi-Core Run Control Manager

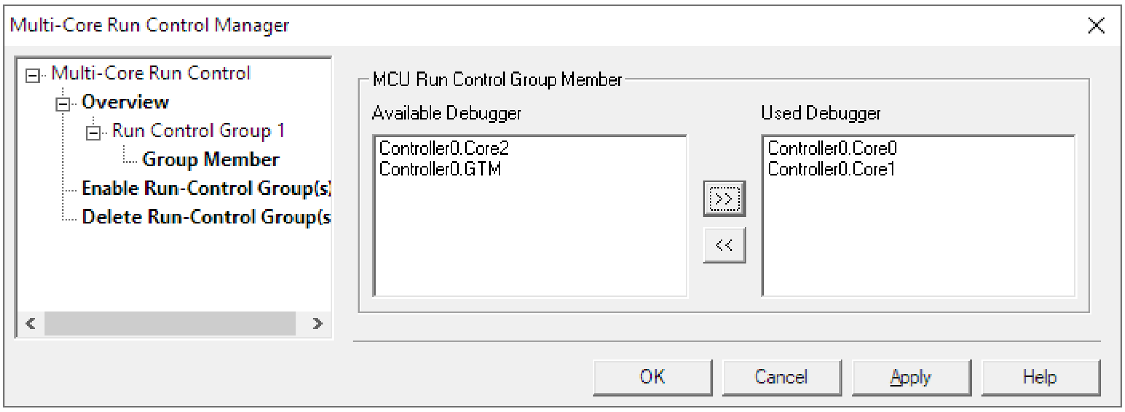
Available Debugger	Used Debugger
Controller0.GTM	Controller0.Core0 Controller0.Core1 Controller0.Core2

Without defined multi-core run control groups, cores are running independently. **Start Program Execution** / **Break** / **Step-Into** / **Step-Over** / **Step-Out** commands will regard the active core / debugger only, while Reset will still regard all cores.

Multi-Core Breakpoints

Multi-core breakpoints are useful in shared code. These breakpoints are associated to all cores of a corresponding run control group.

Open the manager via **Config – Multi-core Run Control Configuration**. Select **Overview - Run Control Group 1 – Group Member** and select `Controller0.Core0` and `Controller0.Core1` as **Used Debugger** and `Controller0.Core2` as **Available Debugger**. `Core0` and `Core1` build a run group and are treated simultaneously in debugging, while `Core2` acts as separate core.


Multi-Core Run Control Manager
Multi-Core Run Control
Overview
Run Control Group 1
Group Member
Enable Run-Control Group(s)
Delete Run-Control Group(s
MCU Run Control Group Member
>>
<<
OK
Cancel
Apply
Help

Available Debugger

```
Controller0.Core2
Controller0.GTM
```

Used Debugger

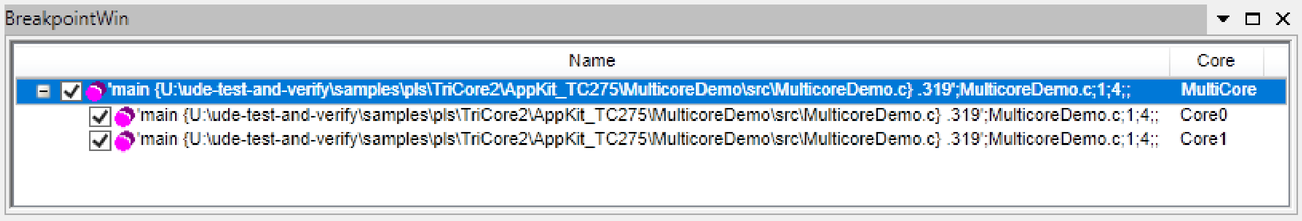
```
Controller0.Core0
Controller0.Core1
```

Reload, run and break the program. `Core0` (halted by user break) and `Core1` (halted) are stopped simultaneously; `Core2` (running) is still running (because it is not part of a run control group). Select `Core2` and stop it manually. Remove all breakpoints.

Reload Program via **F7**. Set now a new multi-core breakpoint via context menu **Insert Multicore Breakpoint** in `Core1` in line 319

```
  SYSTEM_Init();
```

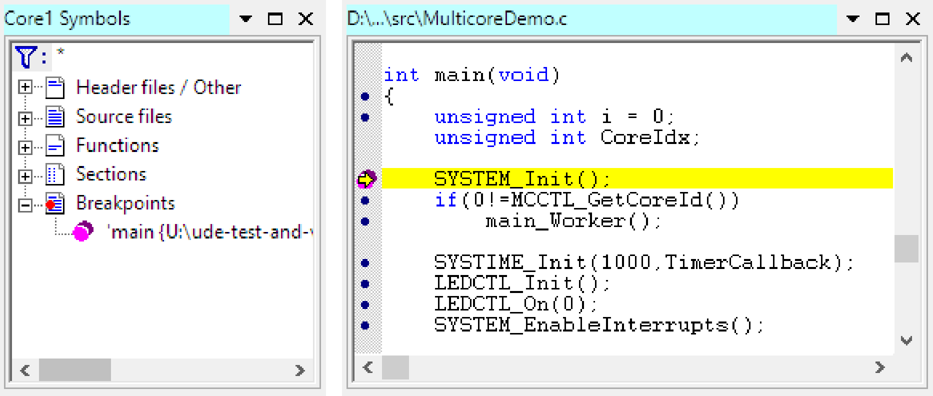
Open the Breakpoint Window via menu **Views – Breakpoints**. The defined multi-core breakpoint is assigned to `Core0` and `Core1` and is shown as follow:



Name	Core
☑ 'main {U:\ude-test-and-verify\samples\pls\TriCore2\AppKit_TC275\MulticoreDemo\src\MulticoreDemo.c} .319';MulticoreDemo.c;1;4;;	MultiCore
☑ 'main {U:\ude-test-and-verify\samples\pls\TriCore2\AppKit_TC275\MulticoreDemo\src\MulticoreDemo.c} .319';MulticoreDemo.c;1;4;;	Core0
☑ 'main {U:\ude-test-and-verify\samples\pls\TriCore2\AppKit_TC275\MulticoreDemo\src\MulticoreDemo.c} .319';MulticoreDemo.c;1;4;;	Core1

Start the program execution. Now `Core0` (halted by user breakpoint) is stopped by the breakpoint condition, `Core1` (inactive) was not yet started by `Core0` and is inactive.

Continue the program execution by **Start Program Execution**. Now the `Core1` (halted by user breakpoint) is stopped by the breakpoint condition at the same code line. `Core0` (halted) is stopped by the run-control group. `Core2` is still running.



```
int main(void)
{
    unsigned int i = 0;
    unsigned int CoreIdx;

    SYSTEM_Init();
    if(0!=MCCTL_GetCoreId())
        main_Worker();

    SYSTIME_Init(1000,TimerCallback);
    LEDCTL_Init();
    LEDCTL_On(0);
    SYSTEM_EnableInterrupts();
```

A Multi-Core GTM Debugging Example with Power Architecture SPC58NG

32-bit multi-core architectures like TriCore, AURIX or PowerPC SPC58NG contain a multipurpose Timer module called Generic Timer Module (GTM).

Because of the complexity of the GTM, this example will only introduce the debug capabilities. At the beginning, an UDE workspace will be created. Then, common debugging operations, like the start-, stop-control and the multi-core behavior, are shown. At last, it will dive into more debugging details of the GTM by debugging the Multi-Core-Sequencer (MCS).

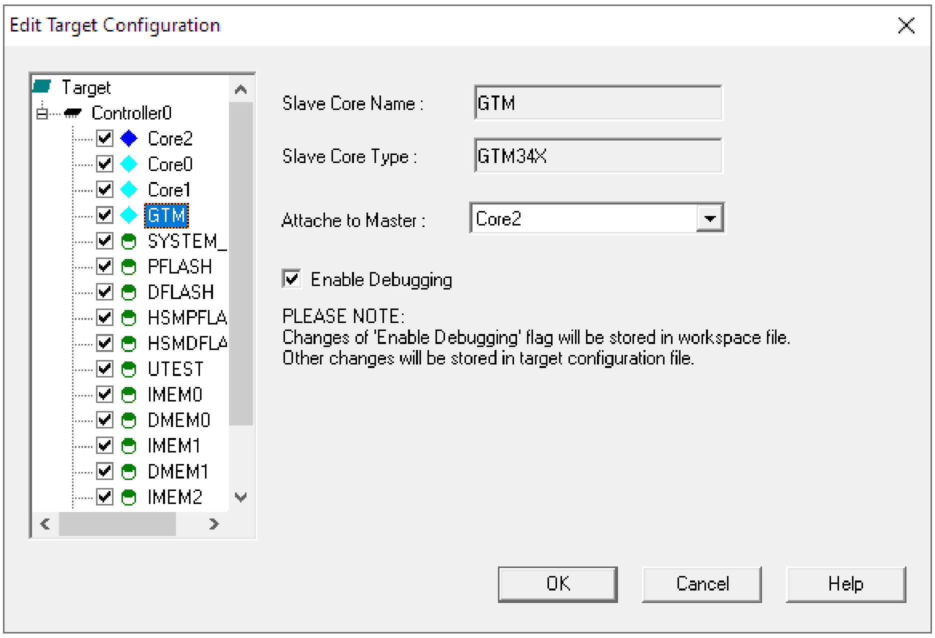
Please run **UDE** from the Window's start menu.

Creating a New Workspace

At first, a new workspace has to be created. Click **New Workspace** from the **File** menu and choose a new file name in the opening file selection box.

UDE provides a number of prepared target configurations (**Default** button). This example uses the PowerPC SPC58NG configuration. Select the entry **PowerPC – STM – SPC58NG Evaluation Board – STM Chorus 6M SPC58NG-DISP Discovery Starter Kit with SPC58NG84 (Jtag/Core2/Core0/Core1)** and click on **Finish**. In the configuration box, choose a file name, e.g. `stm_spc58ng84_cut2_chorus6m_discovery_starterkit_core2_core0_core1_debug_jtag.cfg`.

Select the created configuration and click on **Edit** to modify the target configuration.



Edit Target Configuration

Slave Core Name : GTM

Slave Core Type : GTM34X

Attache to Master : Core2

☑ Enable Debugging

PLEASE NOTE:
Changes of 'Enable Debugging' flag will be stored in workspace file.
Other changes will be stored in target configuration file.

The default target configuration contains all information about the `Core2`, `Core0`, `Core1`, GTM and FLASH devices. `Core2` is the master I/O core (e200z425), `Core0` and `Core1` are slave cores (e200z420) of `Core2`.

Please check that the GTM core is also enabled.

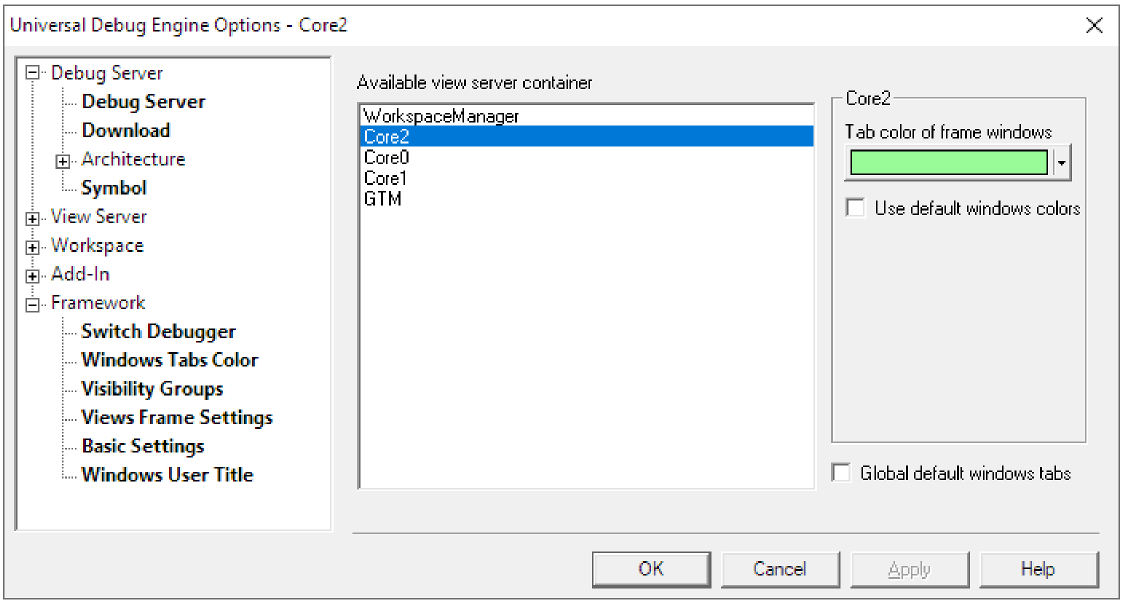
Click **OK**. If the connection is successfully established, the following messages will appear in the Messages window:

```
Core2, UDEDebugServer, Connection to SPC58NG84_CUT2 target established:
PowerPC Target, JTAG-ID: 0x11110041
Core0, UDEDebugServer, Connection to SPC58NG84_CUT2 target established:
PowerPC Target, JTAG-ID: 0x11110041
Core1, UDEDebugServer, Connection to SPC58NG84_CUT2 target established:
PowerPC Target, JTAG-ID: 0x11110041
GTM, UDEDebugServer, Connection to SPC58NG84_CUT2 target established:
GTM, ID: 00000000h
```

Please note that the GTM ID is always `0h` after connect, because it is not enabled yet! If all debuggers are launched, UDE is ready for loading the multi-core GTM application `Timedemo_GTM.elf`.

Preparing the debugger

Windows, which are associated to a single core, use a default coloring of the title bar and/or tabs in tabbed window containers. This default coloring can be customized to the user's needs. Open the menu **Config – Debug Server Configuration – Framework – Windows Tabs Color**. Select one of the cores and change the assigned color. The selected color is now associated with the selected core.



Universal Debug Engine Options - Core2

Available view server container: WorkspaceManager, Core2 (selected), Core0, Core1, GTM

Core2

Tab color of frame windows: 🟩

☐ Use default windows colors

☐ Global default windows tabs

Loading a Multi-Core Executable

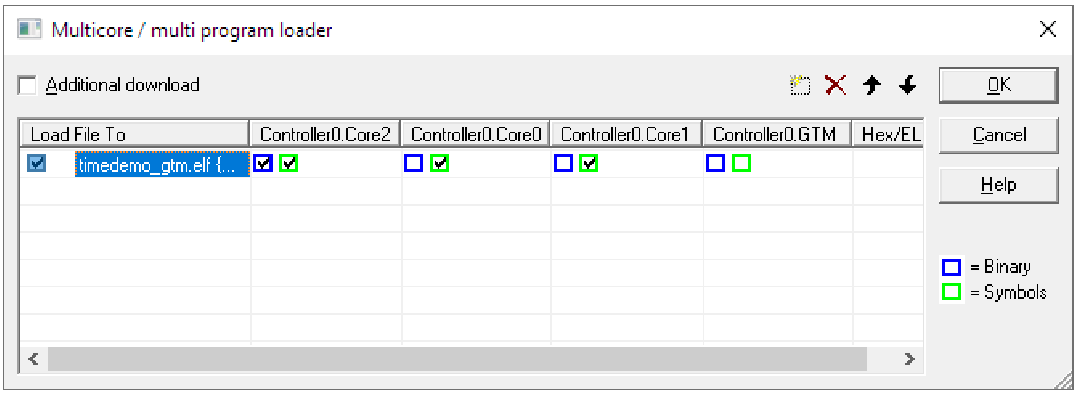
UDE supports several options to handle executables for multi-core architectures:

1. One executable for all cores, combined in a single binary file.
2. Separate executables for each core, combined in a single binary file.
3. Separate binary files for each core.

UDE handles these requirements in a multi-core / multi-program loader box, which provides a switch matrix to assign executables to cores. Additionally, to selecting binary files for cores, the dialog also provides switches to assign symbol information containing files.

Select from the UDE menu **File – Load Program** and load the file `<UDE_SAMPLE_DIRECTORY>\SAMPLES\PPC\SPC84EG84_C2_VLE\Timedemo_GTM\obj-iRam-MC\Timedemo_GTM.elf`.

Because this is an example of a single executable, which is executed by each core, there is only one entry `Timedemo_GTM.elf`. Select the binary check marks only for `Core2` (master core), to avoid loading the binary code twice. Select the symbol check marks for each core, except GTM, to assign symbolic information to each core.



Multicore / multi program loader

□ Additional download

Load File To	Controller0.Core2	Controller0.Core0	Controller0.Core1	Controller0.GTM	Hex/EL
☑ timedemo_gtm.elf (...	☑ ☑	□ ☑	□ ☑	□ □	

□ = Binary
□ = Symbols

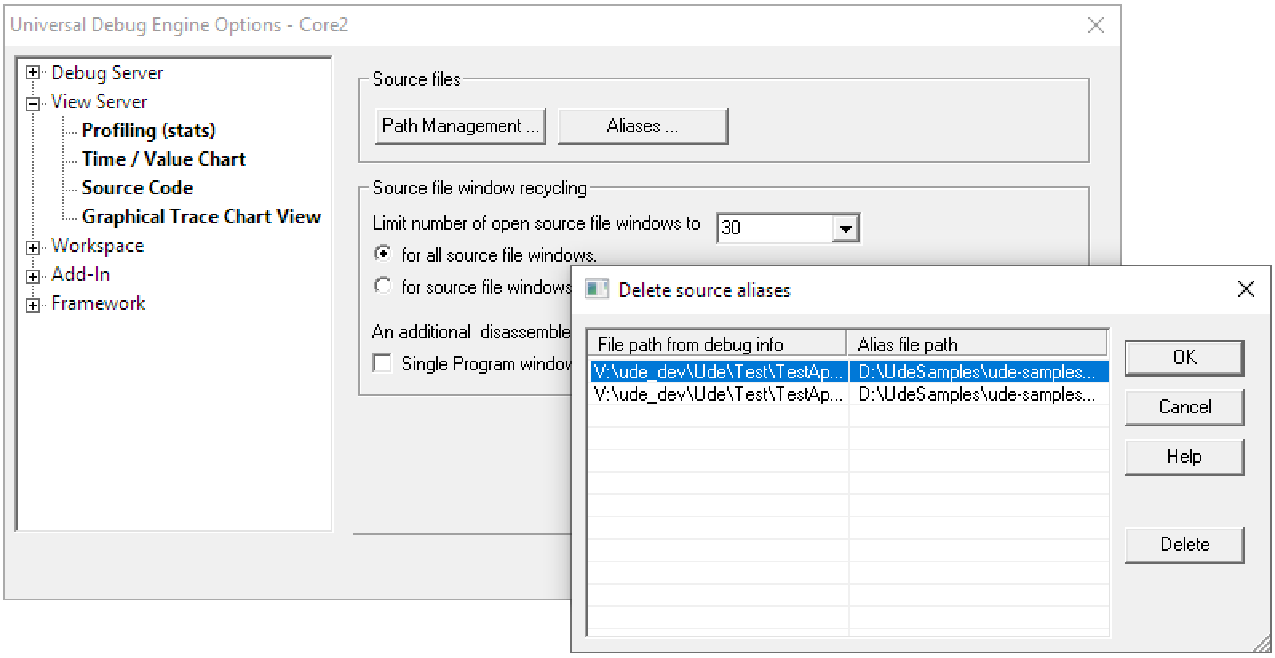
UDE will load and transfer all binaries into the corresponding core and the core symbols to the debugger symbol server.

Source File Management

If the debugger cannot find the source files specified in the symbol containing files, it can request the location of the correct source path. This can happen, if the compiler’s-built environment differs from the environment used by the debugging process.

The examples source files are located in the `\src` and `\board` subfolders. You need to provide the location on request. UDE will remember your decision and will not ask again for this source path.

Every entry of this relocated source is called **Alias**. The saved aliases of a workspace are accessible via the menu **Config – Debug Server Configuration – View Server – Source Code – Aliases …**



Universal Debug Engine Options - Core2

Limit number of open source file windows to 30

● for all source file windows.

○ for source file windows

An additional disassemble

□ Single Program window

Delete source aliases

File path from debug info	Alias file path
V:\ude_dev\Ude\Test\TestAp...	D:\UdeSamples\ude-samples...
V:\ude_dev\Ude\Test\TestAp...	D:\UdeSamples\ude-samples...

In some cases, the forced source alias may have changed. Use the dialog to delete the affected entry. The next time the source file is used, UDE will ask for the location again.

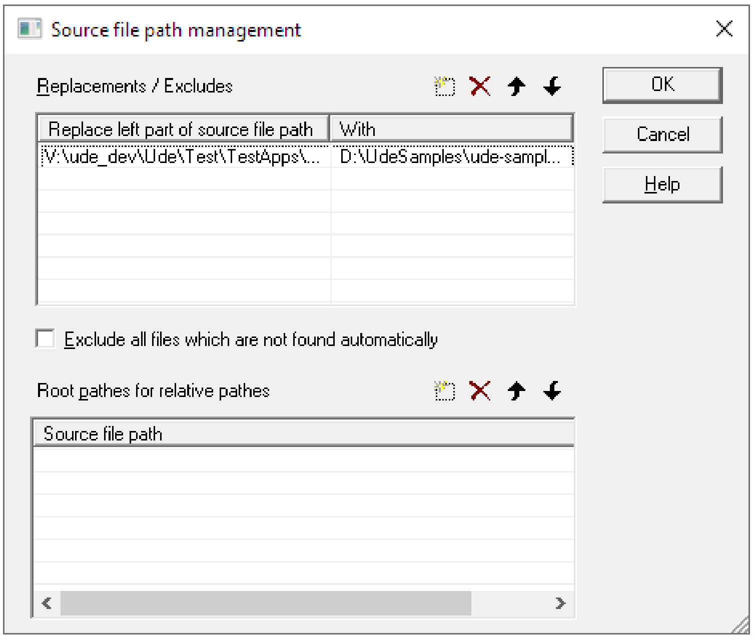
Source Path Replacement

The behavior described above is useful, if only few source file locations have changed. If a complete source file tree is affected, you can give UDE a replacement path, which will be applied to all source file paths.

Use the menu **Config – Debug Server Configuration – View Server – Source Code – Path Management …**

The dialog defines a part for replacing and a part for including the new source location.

You will now see the source code of the main function from the sample application. When clicking with the right-hand mouse button into the Program window, a context button appears to switch between Source code only and Source/Assembly code display via the **Mixed Mode** entry.



Source file path management

Replacements / Excludes

Replace left part of source file path	With
V:\ude_dev\Ude\Test\TestApps\...	D:\UdeSamples\ude-sampl...

☐ Exclude all files which are not found automatically

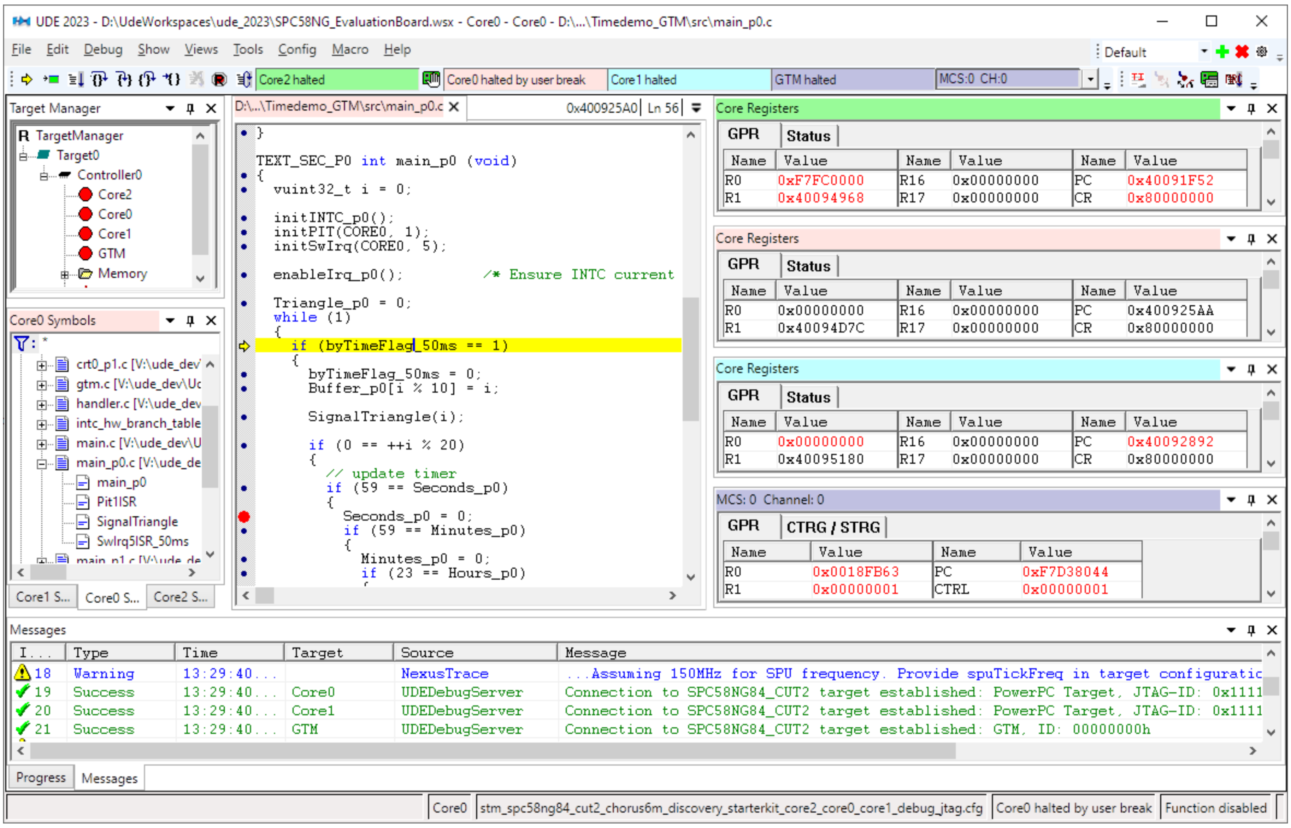
Root pathes for relative pathes

Source file path

Core selection

To select a specific core debugger, click into a core-specific window (with a colored frame and/or tab) and the focus will be on this core debugger. The state bar at the bottom of the UDE workbench shows the currently selected core debugger. The state bar field also provides a context menu (right-click) for selection of a core debugger. At least, the context menu of a specific core in the **Target Manager** windows contains an entry to activate and focus this core.

This screenshot shows an example configuration for a multicore workbench:



UDE 2023 - D:\UdeWorkspaces\ude_2023\SPC58NG_EvaluationBoard.wsx - Core0 - Core0 - D:\...\Timedemo_GTM\src\main_p0.c

```
}

TEXT_SEC_P0 int main_p0 (void)
{
  vuint32_t i = 0;

  initINTC_p0();
  initPIT(CORE0, 1);
  initSwIrq(CORE0, 5);

  enableIrq_p0();           /* Ensure INTC current

  Triangle_p0 = 0;
  while (1)
  {
    if (byTimeFlag_50ms == 1)
    {
      byTimeFlag_50ms = 0;
      Buffer_p0[i % 10] = i;

      SignalTriangle(i);

      if (0 == ++i % 20)
      {
        // update timer
        if (59 == Seconds_p0)
        {
          Seconds_p0 = 0;
          if (59 == Minutes_p0)
          {
            Minutes_p0 = 0;
            if (23 == Hours_p0)
```

Core Registers

GPR | Status

Name	Value	Name	Value	Name	Value
R0	0xF7FC0000	R16	0x00000000	PC	0x40091F52
R1	0x40094968	R17	0x00000000	CR	0x80000000

Core Registers

GPR | Status

Name	Value	Name	Value	Name	Value
R0	0x00000000	R16	0x00000000	PC	0x400925AA
R1	0x40094D7C	R17	0x00000000	CR	0x80000000

Core Registers

GPR | Status

Name	Value	Name	Value	Name	Value
R0	0x00000000	R16	0x00000000	PC	0x40092892
R1	0x40095180	R17	0x00000000	CR	0x80000000

MCS: 0 Channel: 0

GPR | CTRG / STRG

Name	Value	Name	Value
R0	0x0018FB63	PC	0xF7D38044
R1	0x00000001	CTRL	0x00000001

Messages

I...	Type	Time	Target	Source	Message
18	Warning	13:29:40...		NexusTrace	...Assuming 150MHz for SPU frequency. Provide spuTickFreq in target configuratic
19	Success	13:29:40...	Core0	UDEDebugServer	Connection to SPC58NG84_CUT2 target established: PowerPC Target, JTAG-ID: 0x1111
20	Success	13:29:40...	Core1	UDEDebugServer	Connection to SPC58NG84_CUT2 target established: PowerPC Target, JTAG-ID: 0x1111
21	Success	13:29:40...	GTM	UDEDebugServer	Connection to SPC58NG84_CUT2 target established: GTM, ID: 00000000h

Core0 | stm_spc58ng84_cut2_chorus6m_discovery_starterkit_core2_core0_core1_debug_jtag.cfg | Core0 halted by user break | Function disabled

`Core0` is halted by user break and is focused, a breakpoint is set, but not reached. `Core1`, `Core2` and `GTM` are halted and not focused.

Single-Core Breakpoints

Our first example will set a breakpoint for a single core at a single program location. We will set a breakpoint for Core 0, leaving every other core running. If the program window for Core 0 is not open already, use the **Core0 Symbols** window and click on the source file name to open the **Program window**.

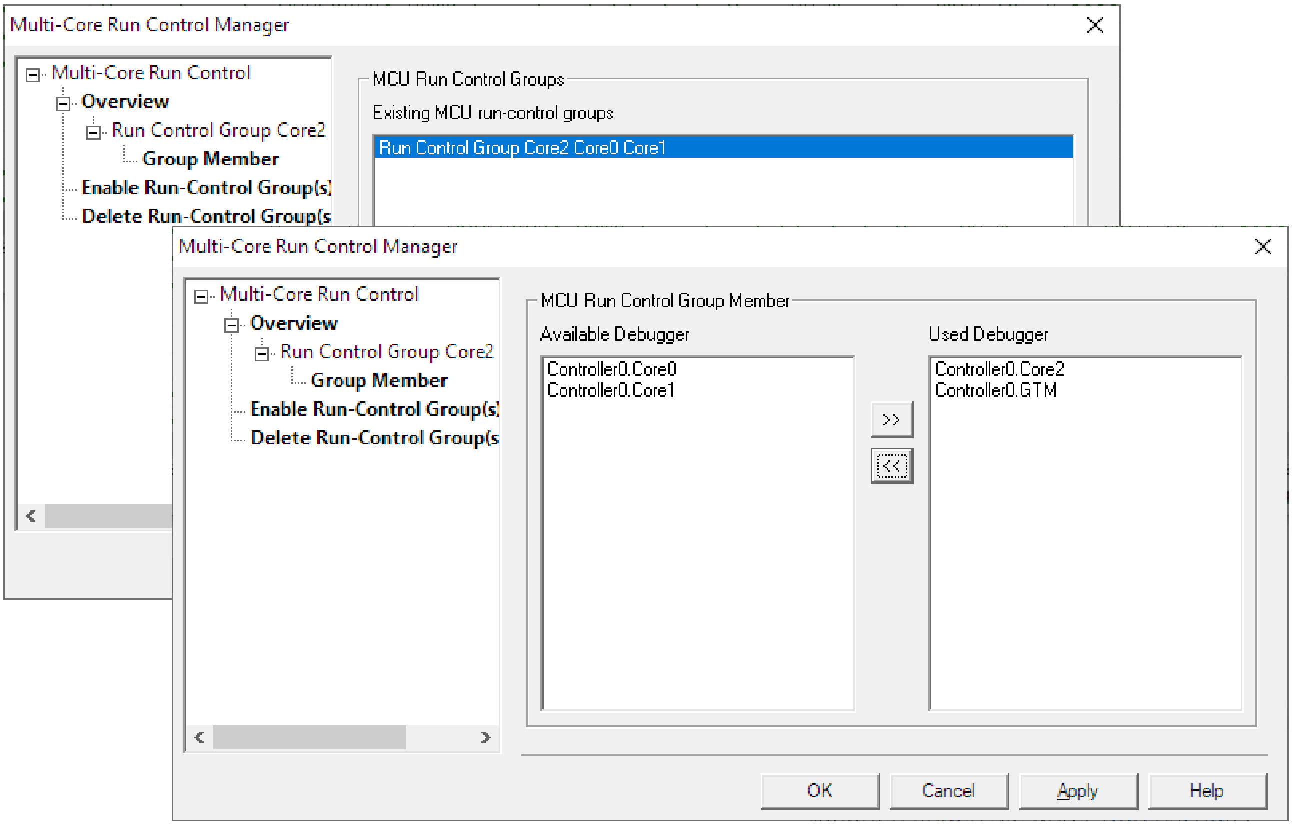
Right-click on line 71 in the Program window and use the context menu via **Insert/Remove Single Breakpoint** to set a breakpoint. Now select `Core2` (see section **Core Selection**), because only this core can be used for initialization in the PowerPC architecture. Now start the program via **Debug – Start Program Execution**. This will run the `Timedemo_GTM.elf` on `Core2`. In the Timedemo sample, `Core2` will start the slave cores `Core1` und `Core0` automatically. After a while, `Core0` stops at the breakpoint, while `Core1` and `Core2` are still running. After inspection of `Core0`, it can be started again via **Debug – Start Program Execution**.

To orchestrate the behavior of cores, one can use run control groups. Without run control groups, cores are running independently. **Start Program Execution** / **Break** / **Step-Into** / **Step-Over** / **Step-Out** commands will regard the active core / debugger only, while Reset will still regard all cores.

Multi-Core Breakpoints and Stepping

A multi-core application may require that all or some cores be halted too, if one core reaches a breakpoint, and/or that all halted cores must be started simultaneously. The **Multi-core run group manager** helps you to set up the combination of halting and starting of core groups. Open the manager via **Config – Multi-core Run Control Configuration**.

Change the existing group **Run Control Group 1** and add `Controller0.Core2` and `Controller0.GTM` to this group.


Multi-Core Run Control Manager
Multi-Core Run Control
Overview
Run Control Group Core2
Group Member
Enable Run-Control Group(s)
Delete Run-Control Group(s
MCU Run Control Groups
Existing MCU run-control groups
Multi-Core Run Control Manager
Multi-Core Run Control
Overview
Run Control Group Core2
Group Member
Enable Run-Control Group(s)
Delete Run-Control Group(s
MCU Run Control Group Member
Available Debugger
Used Debugger
>>
<<
OK
Cancel
Apply
Help

Run Control Group Core2 Core0 Core1

Available Debugger: Controller0.Core0, Controller0.Core1

Used Debugger: Controller0.Core2, Controller0.GTM

Select the desired MCU group members by moving the entry to the appropriate column.

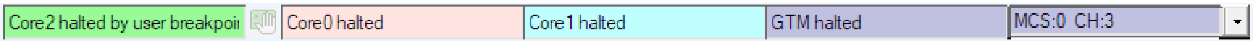
Break the program via **Ctrl+F5**. `Core2` and `GTM` are stopped simultaneously; `Core0` and `Core1` are still running (because these are not part of a run control group). Remove all breakpoints. Restart Program via **F7**. Stepping through the program code will operate the same way.

Inspecting Multi-Channel-Sequencer (MCS) Channels

The Generic Timer Module (GTM) contains several modules, which communicate over an Advanced Routing Unit (ARU). The ARU connects all GTM modules, including the MCS, to each other. Depending on the GTM version, a different number of MCS cores are available, which can be used to modify signals that are routed to them. One MCS core contains thereby its own RAM that is divided up to 8 channel programs. Each channel run its own program, based on special sequencer instructions, which modifies the signals routed to it. This section shows how different **MCS cores** and channels can be inspected.

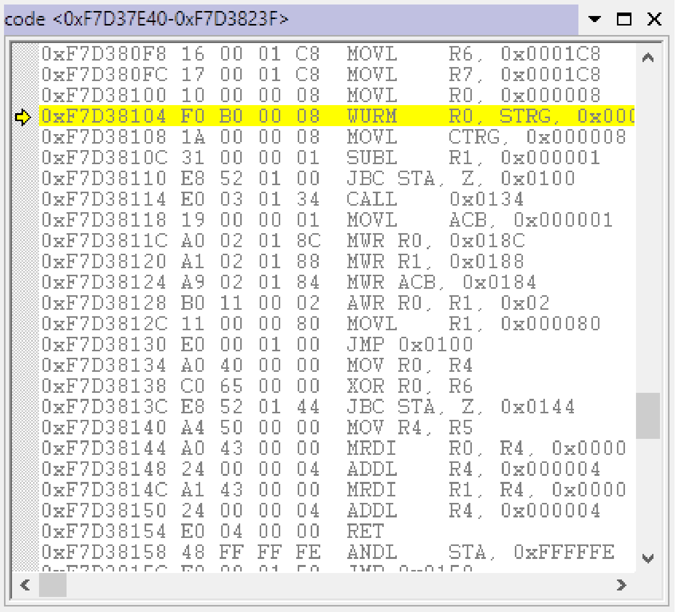
At first, restart the Program via F7. If not already done, create a new **Run-Control-Group** as mentioned above. Now select `Core2`and set a breakpoint to **line 584** (behind the `initGTM()` -function) in `main.c`. Select `Core2` and run the program via **Start Program Execution**. After a short time, the `Core2` and `GTM` core stop.

Click on the GTM list box to select the desired MCS channel, here right in the image:



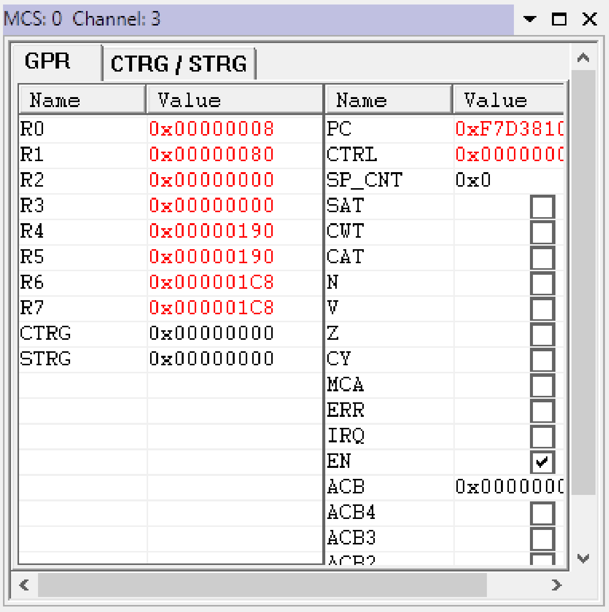
Core2 halted by user breakpoi	Core0 halted	Core1 halted	GTM halted	MCS:0 CH:3

Using the right selector, the MCS and its channel can be selected. Open the **Code Window** by selecting the colored tab corresponding to the GTM core to look for the channel instructions as shown below:



code <0xF7D37E40-0xF7D3823F>

```
0xF7D380F8 16 00 01 C8  MOVL    R6, 0x0001C8
0xF7D380FC 17 00 01 C8  MOVL    R7, 0x0001C8
0xF7D38100 10 00 00 08  MOVL    R0, 0x000008
0xF7D38104 F0 B0 00 08  WURM    R0, STRG, 0x000
0xF7D38108 1A 00 00 08  MOVL    CTRG, 0x000008
0xF7D3810C 31 00 00 01  SUBL    R1, 0x000001
0xF7D38110 E8 52 01 00  JBC STA, Z, 0x0100
0xF7D38114 E0 03 01 34  CALL    0x0134
0xF7D38118 19 00 00 01  MOVL    ACB, 0x000001
0xF7D3811C A0 02 01 8C  MWR R0, 0x018C
0xF7D38120 A1 02 01 88  MWR R1, 0x0188
0xF7D38124 A9 02 01 84  MWR ACB, 0x0184
0xF7D38128 B0 11 00 02  AWR R0, R1, 0x02
0xF7D3812C 11 00 00 80  MOVL    R1, 0x000080
0xF7D38130 E0 00 01 00  JMP 0x0100
0xF7D38134 A0 40 00 00  MOV R0, R4
0xF7D38138 C0 65 00 00  XOR R0, R6
0xF7D3813C E8 52 01 44  JBC STA, Z, 0x0144
0xF7D38140 A4 50 00 00  MOV R4, R5
0xF7D38144 A0 43 00 00  MRDI    R0, R4, 0x0000
0xF7D38148 24 00 00 04  ADDL    R4, 0x000004
0xF7D3814C A1 43 00 00  MRDI    R1, R4, 0x0000
0xF7D38150 24 00 00 04  ADDL    R4, 0x000004
0xF7D38154 E0 04 00 00  RET
0xF7D38158 48 FF FF FE  ANDL    STA, 0xFFFFFE
```



MCS: 0 Channel: 3

Name	Value	Name	Value
R0	0x00000008	PC	0xF7D3810
R1	0x00000080	CTRL	0x0000000
R2	0x00000000	SP_CNT	0x0
R3	0x00000000	SAT	□
R4	0x00000190	CWT	□
R5	0x00000190	CAT	□
R6	0x000001C8	N	□
R7	0x000001C8	V	□
CTRG	0x00000000	Z	□
STRG	0x00000000	CY	□
		MCA	□
		ERR	□
		IRQ	□
		EN	☑
		ACB	0x0000000
		ACB4	□
		ACB3	□
		ACB2	□

The **MCS Core Registers Window** shows the current context register values.

After each MCS channel select, the IP inside the **Code Window** and the **MCS Core Registers Window** contents are updated properly.

Using the MCDS On-Chip Trace with AURIX TC2xx

Preparations

The further steps require AURIX target hardware containing a TC27xED emulation device. Please check, that the board is equipped with an emulation device chip.

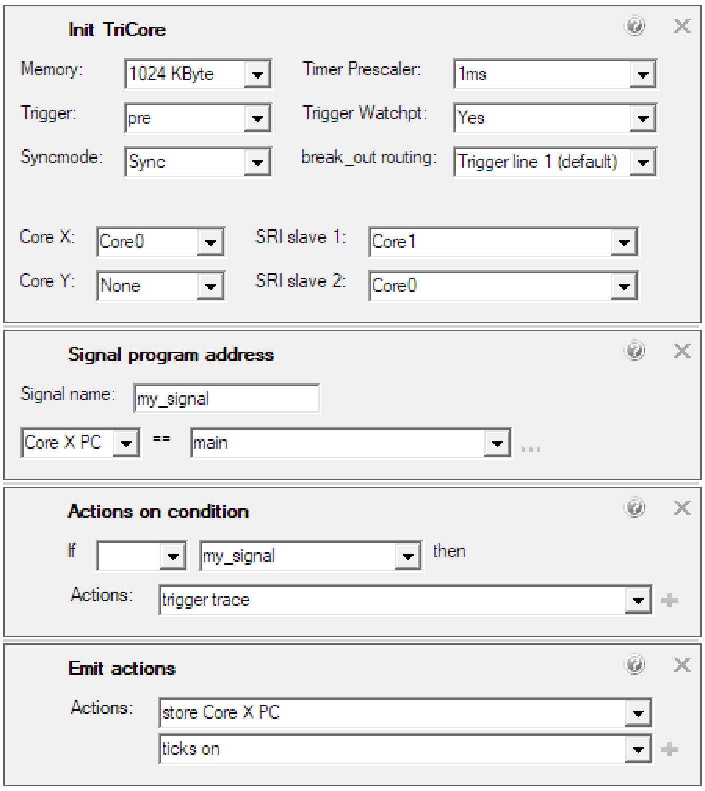
The following preparations are required:

- that the connection to the AURIX target is established, and
- that the samples application `TimeDemo.elf` for TC27xED is loaded.

Recording the first Samples

Open a new Trace window via menu **Views** – **Trace**. Open the UEC trace configuration window via menu **Tools** – **Configure Trace** to create a trace configuration for recording the program trace of **Core0** of TC27xED. Execute the following steps to create the required configuration via menu **Tools – Configure trace …** :

- Switch from **Compact** to **Advanced Configuration** library
- Drag the **Init TriCore** library element (1.0) to configuration area
- Change **Memory Size** to maximum value of **1024 kByte**, change **Syncmode** to mode **Sync** to set the best tick resolution between traced code samples
- Drag the **Signal program address** library element (2.1) to configuration area, set **Signal name** to e.g. `my_signal`, select **main** function start address as comparison address for **Core X PC**
- Drag the **Actions on condition** library element (5.1) to configuration area, browse `my_signal` as signal name for **If** condition, select as action **Trigger trace**
- Drag the **Emit actions** library element (5.3) to configuration area, add emit action **store Core X PC** and add **ticks on**



Init TriCore

Memory: 1024 KByte

Timer Prescaler: 1ms

Trigger: pre

Trigger Watchpt: Yes

Syncmode: Sync

break_out routing: Trigger line 1 (default)

Core X: Core0

SRI slave 1: Core1

Core Y: None

SRI slave 2: Core0

Signal program address

Signal name: my_signal

Core X PC == main

Actions on condition

If my_signal then

Actions: trigger trace

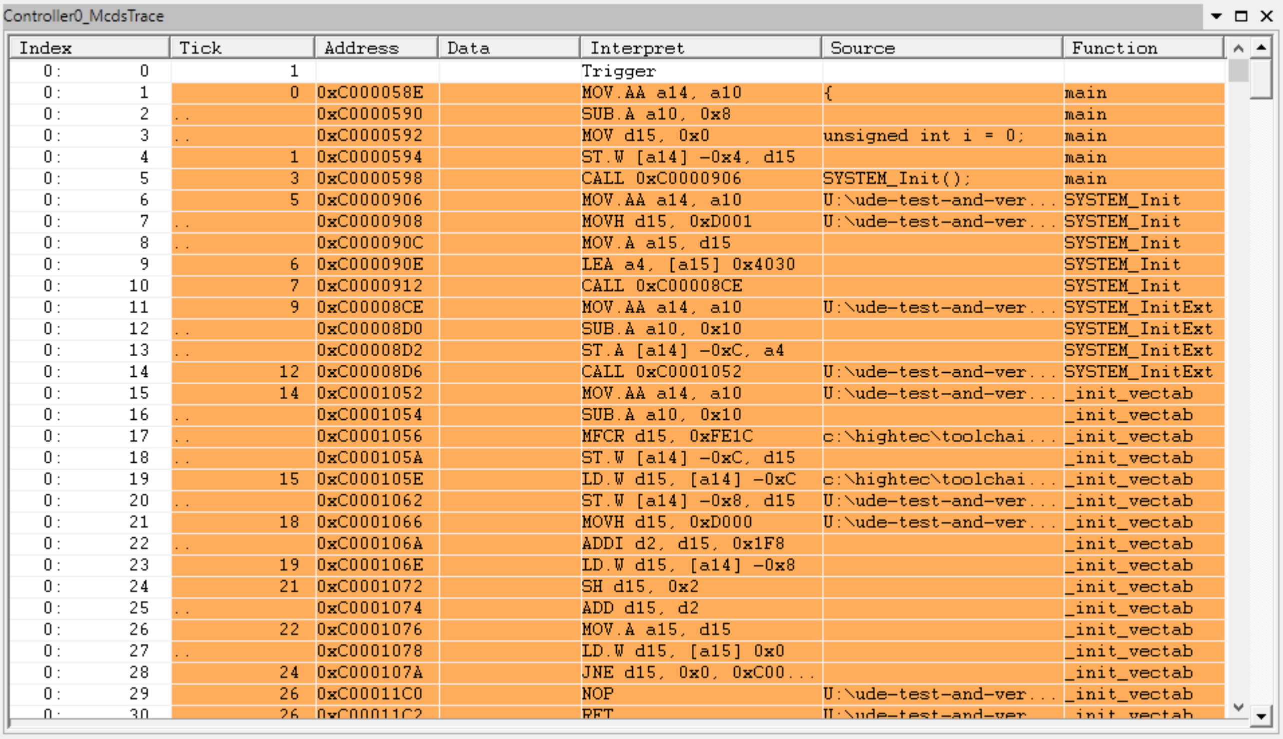
Emit actions

Actions: store Core X PC

ticks on

The MCDS trace configuration for this special task is now completed.

Now the can be started the trace recording via toolbar icon or menu **Tools – Start trace...** . The program execution can be started using the menu **Debug** – **Start Program Execution**. As soon as MCDS trace memory is full, the Trace window will be filled.



Controller0_McdsTrace

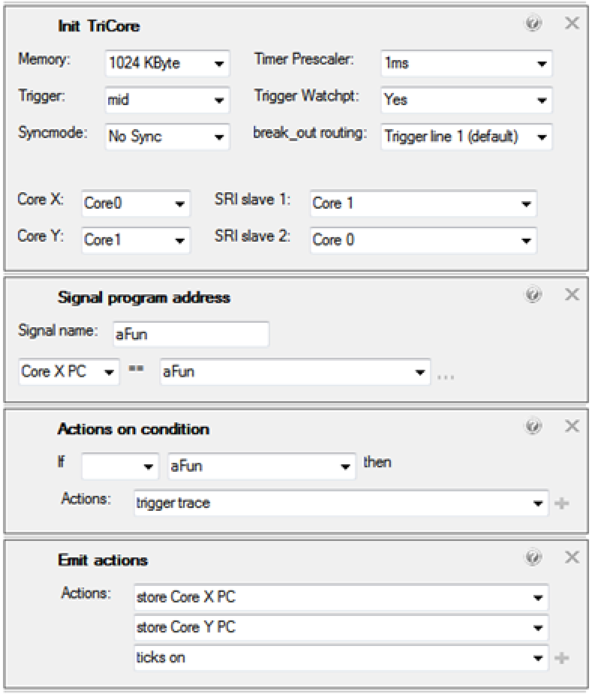
Index	Tick	Address	Data	Interpret	Source	Function
0: 0	1			Trigger		
0: 1	0	0xC000058E		MOV.AA a14, a10	{	main
0: 2	..	0xC0000590		SUB.A a10, 0x8		main
0: 3	..	0xC0000592		MOV d15, 0x0	unsigned int i = 0;	main
0: 4	1	0xC0000594		ST.W [a14] -0x4, d15		main
0: 5	3	0xC0000598		CALL 0xC0000906	SYSTEM_Init();	main
0: 6	5	0xC0000906		MOV.AA a14, a10	U:\ude-test-and-ver...	SYSTEM_Init
0: 7	..	0xC0000908		MOVH d15, 0xD001	U:\ude-test-and-ver...	SYSTEM_Init
0: 8	..	0xC000090C		MOV.A a15, d15		SYSTEM_Init
0: 9	6	0xC000090E		LEA a4, [a15] 0x4030		SYSTEM_Init
0: 10	7	0xC0000912		CALL 0xC00008CE		SYSTEM_Init
0: 11	9	0xC00008CE		MOV.AA a14, a10	U:\ude-test-and-ver...	SYSTEM_InitExt
0: 12	..	0xC00008D0		SUB.A a10, 0x10		SYSTEM_InitExt
0: 13	..	0xC00008D2		ST.A [a14] -0xC, a4		SYSTEM_InitExt
0: 14	12	0xC00008D6		CALL 0xC0001052	U:\ude-test-and-ver...	SYSTEM_InitExt
0: 15	14	0xC0001052		MOV.AA a14, a10	U:\ude-test-and-ver...	_init_vectab
0: 16	..	0xC0001054		SUB.A a10, 0x10		_init_vectab
0: 17	..	0xC0001056		MFCR d15, 0xFE1C	c:\hightec\toolchai...	_init_vectab
0: 18	..	0xC000105A		ST.W [a14] -0xC, d15		_init_vectab
0: 19	15	0xC000105E		LD.W d15, [a14] -0xC	c:\hightec\toolchai...	_init_vectab
0: 20	..	0xC0001062		ST.W [a14] -0x8, d15	U:\ude-test-and-ver...	_init_vectab
0: 21	18	0xC0001066		MOVH d15, 0xD000	U:\ude-test-and-ver...	_init_vectab
0: 22	..	0xC000106A		ADDI d2, d15, 0x1F8		_init_vectab
0: 23	19	0xC000106E		LD.W d15, [a14] -0x8		_init_vectab
0: 24	21	0xC0001072		SH d15, 0x2		_init_vectab
0: 25	..	0xC0001074		ADD d15, d2		_init_vectab
0: 26	22	0xC0001076		MOV.A a15, d15		_init_vectab
0: 27	..	0xC0001078		LD.W d15, [a15] 0x0		_init_vectab
0: 28	24	0xC000107A		JNE d15, 0x0, 0xC00...		_init_vectab
0: 29	26	0xC00011C0		NOP	U:\ude-test-and-ver...	_init_vectab
0: 30	26	0xC00011C2		RET	U:\ude-test-and-ver	_init_vectab

The recorded samples are the instructions of program execution of core 0 of TC27xED after call of **main** function.

Hints for Multi-Core Trace

The emulation devices (TC27xED) of the AURIX uController support parallel trace of two cores. The Graphical trace configuration can be done by Universal Emulation Configurator (UEC) using the **Advanced** library.

Here is an example for a suitable trace configuration:



Init TriCore

Memory: 1024 KByte
Timer Prescaler: 1ms
Trigger: mid
Trigger Watchpt: Yes
Syncmode: No Sync
break_out routing: Trigger line 1 (default)
Core X: Core0
SRI slave 1: Core 1
Core Y: Core1
SRI slave 2: Core 0

Signal program address

Signal name: aFun
Core X PC == aFun

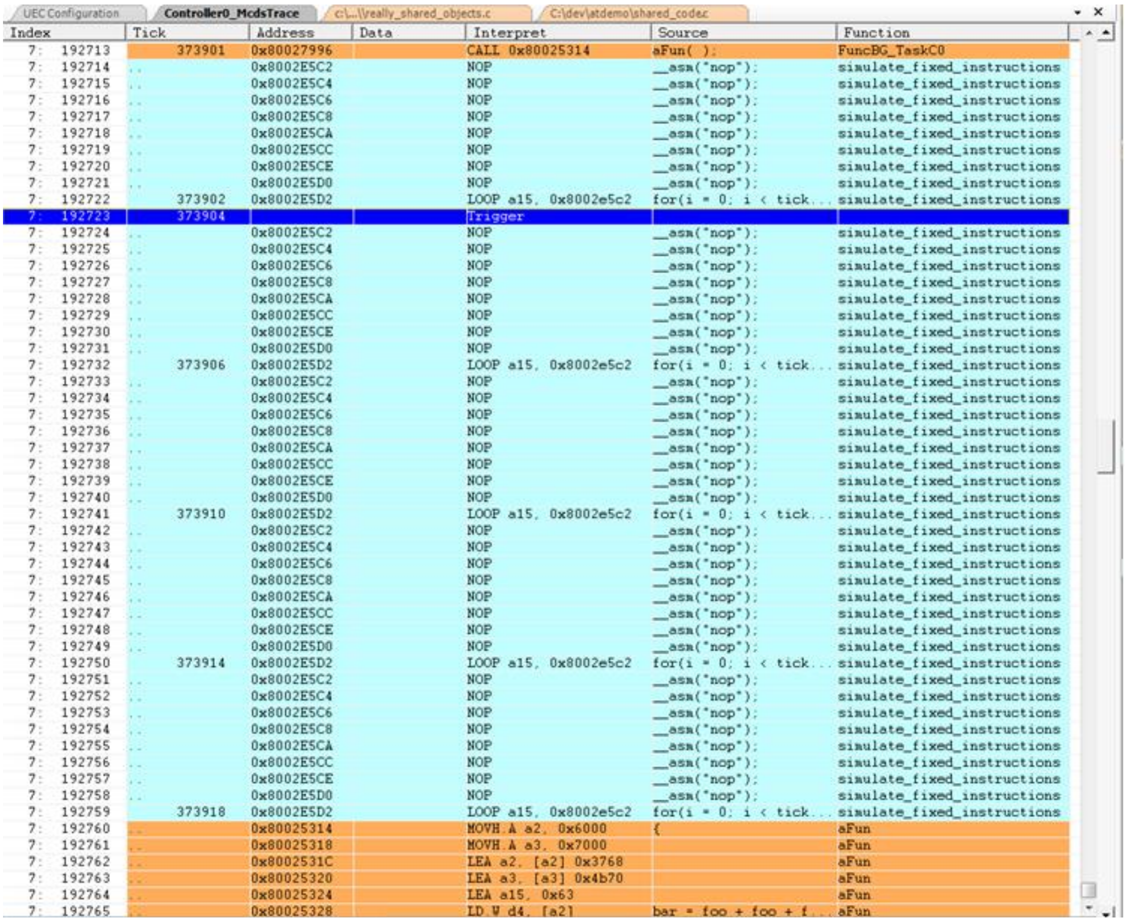
Actions on condition

If aFun then
Actions: trigger trace

Emit actions

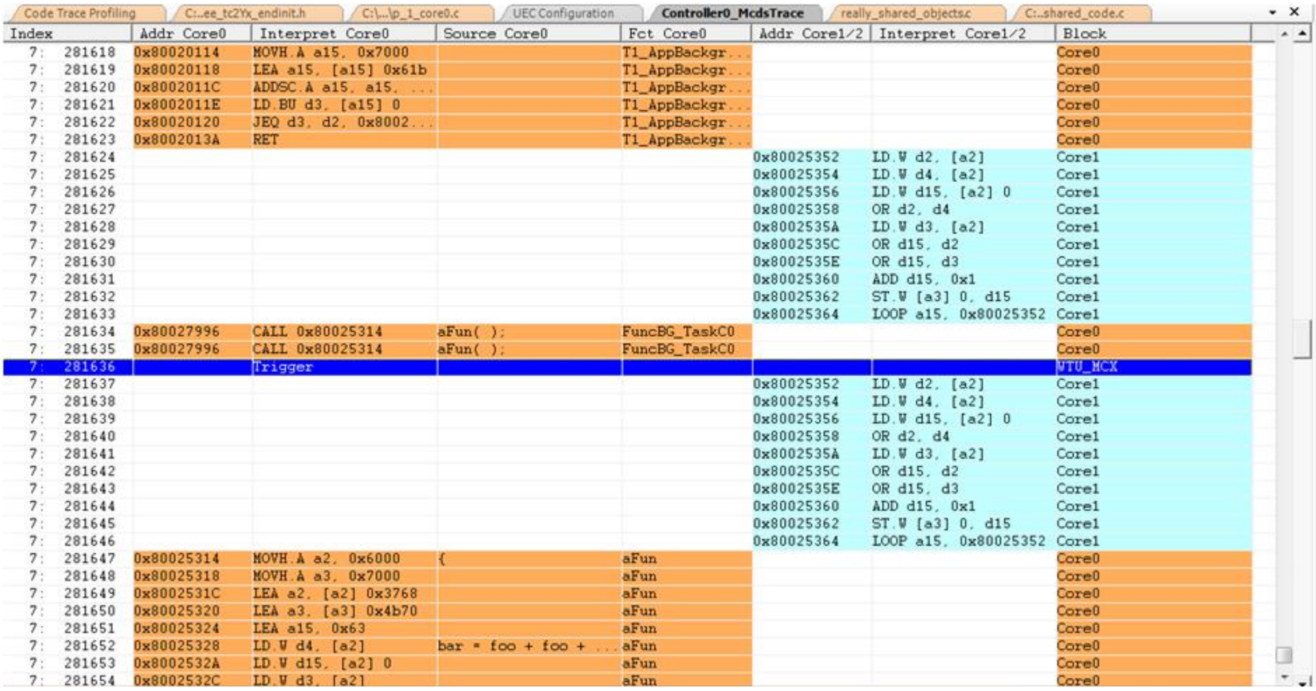
Actions: store Core X PC
store Core Y PC
ticks on

The screenshots shows an example of the resulting multi-core trace in a single Trace window:



Index	Tick	Address	Data	Interpret	Source	Function
7: 192713	373901	0x80027996		CALL 0x80025314	aFun( );	FuncBG_TaskC0
7: 192714	..	0x8002E5C2		NOP	__asm("nop");	simulate_fixed_instructions
7: 192715	..	0x8002E5C4		NOP	__asm("nop");	simulate_fixed_instructions
7: 192716	..	0x8002E5C6		NOP	__asm("nop");	simulate_fixed_instructions
7: 192717	..	0x8002E5C8		NOP	__asm("nop");	simulate_fixed_instructions
7: 192718	..	0x8002E5CA		NOP	__asm("nop");	simulate_fixed_instructions
7: 192719	..	0x8002E5CC		NOP	__asm("nop");	simulate_fixed_instructions
7: 192720	..	0x8002E5CE		NOP	__asm("nop");	simulate_fixed_instructions
7: 192721	..	0x8002E5D0		NOP	__asm("nop");	simulate_fixed_instructions
7: 192722	373902	0x8002E5D2		LOOP a15, 0x8002e5c2	for(i = 0; i < tick...	simulate_fixed_instructions
7: 192723	373904			Trigger		
7: 192724	..	0x8002E5C2		NOP	__asm("nop");	simulate_fixed_instructions
7: 192725	..	0x8002E5C4		NOP	__asm("nop");	simulate_fixed_instructions
7: 192726	..	0x8002E5C6		NOP	__asm("nop");	simulate_fixed_instructions
7: 192727	..	0x8002E5C8		NOP	__asm("nop");	simulate_fixed_instructions
7: 192728	..	0x8002E5CA		NOP	__asm("nop");	simulate_fixed_instructions
7: 192729	..	0x8002E5CC		NOP	__asm("nop");	simulate_fixed_instructions
7: 192730	..	0x8002E5CE		NOP	__asm("nop");	simulate_fixed_instructions
7: 192731	..	0x8002E5D0		NOP	__asm("nop");	simulate_fixed_instructions
7: 192732	373906	0x8002E5D2		LOOP a15, 0x8002e5c2	for(i = 0; i < tick...	simulate_fixed_instructions
7: 192733	..	0x8002E5C2		NOP	__asm("nop");	simulate_fixed_instructions
7: 192734	..	0x8002E5C4		NOP	__asm("nop");	simulate_fixed_instructions
7: 192735	..	0x8002E5C6		NOP	__asm("nop");	simulate_fixed_instructions
7: 192736	..	0x8002E5C8		NOP	__asm("nop");	simulate_fixed_instructions
7: 192737	..	0x8002E5CA		NOP	__asm("nop");	simulate_fixed_instructions
7: 192738	..	0x8002E5CC		NOP	__asm("nop");	simulate_fixed_instructions
7: 192739	..	0x8002E5CE		NOP	__asm("nop");	simulate_fixed_instructions
7: 192740	..	0x8002E5D0		NOP	__asm("nop");	simulate_fixed_instructions
7: 192741	373910	0x8002E5D2		LOOP a15, 0x8002e5c2	for(i = 0; i < tick...	simulate_fixed_instructions
7: 192742	..	0x8002E5C2		NOP	__asm("nop");	simulate_fixed_instructions
7: 192743	..	0x8002E5C4		NOP	__asm("nop");	simulate_fixed_instructions
7: 192744	..	0x8002E5C6		NOP	__asm("nop");	simulate_fixed_instructions
7: 192745	..	0x8002E5C8		NOP	__asm("nop");	simulate_fixed_instructions
7: 192746	..	0x8002E5CA		NOP	__asm("nop");	simulate_fixed_instructions
7: 192747	..	0x8002E5CC		NOP	__asm("nop");	simulate_fixed_instructions
7: 192748	..	0x8002E5CE		NOP	__asm("nop");	simulate_fixed_instructions
7: 192749	..	0x8002E5D0		NOP	__asm("nop");	simulate_fixed_instructions
7: 192750	373914	0x8002E5D2		LOOP a15, 0x8002e5c2	for(i = 0; i < tick...	simulate_fixed_instructions
7: 192751	..	0x8002E5C2		NOP	__asm("nop");	simulate_fixed_instructions
7: 192752	..	0x8002E5C4		NOP	__asm("nop");	simulate_fixed_instructions
7: 192753	..	0x8002E5C6		NOP	__asm("nop");	simulate_fixed_instructions
7: 192754	..	0x8002E5C8		NOP	__asm("nop");	simulate_fixed_instructions
7: 192755	..	0x8002E5CA		NOP	__asm("nop");	simulate_fixed_instructions
7: 192756	..	0x8002E5CC		NOP	__asm("nop");	simulate_fixed_instructions
7: 192757	..	0x8002E5CE		NOP	__asm("nop");	simulate_fixed_instructions
7: 192758	..	0x8002E5D0		NOP	__asm("nop");	simulate_fixed_instructions
7: 192759	373918	0x8002E5D2		LOOP a15, 0x8002e5c2	for(i = 0; i < tick...	simulate_fixed_instructions
7: 192760	..	0x80025314		MOVH.A a2, 0x6000	{	aFun
7: 192761	..	0x80025318		MOVH.A a3, 0x7000		aFun
7: 192762	..	0x8002531C		LEA a2, [a2] 0x3768		aFun
7: 192763	..	0x80025320		LEA a3, [a3] 0x4b70		aFun
7: 192764	..	0x80025324		LEA a15, 0x63		aFun
7: 192765	..	0x80025328		LD.W d4, [a2]	bar = foo + foo + f...	aFun

The multi-core trace can also be grouped in the Trace window, using core specific filter:



Index	Addr Core0	Interpret Core0	Source Core0	Fct Core0	Addr Core1/2	Interpret Core1/2	Block
7: 281618	0x80020114	MOVH.A a15, 0x7000		T1_AppBackgr...			Core0
7: 281619	0x80020118	LEA a15, [a15] 0x61b		T1_AppBackgr...			Core0
7: 281620	0x8002011C	ADDSC.A a15, a15, ...		T1_AppBackgr...			Core0
7: 281621	0x8002011E	LD.BU d3, [a15] 0		T1_AppBackgr...			Core0
7: 281622	0x80020120	JEQ d3, d2, 0x8002...		T1_AppBackgr...			Core0
7: 281623	0x8002013A	RET		T1_AppBackgr...			Core0
7: 281624					0x80025352	LD.W d2, [a2]	Core1
7: 281625					0x80025354	LD.W d4, [a2]	Core1
7: 281626					0x80025356	LD.W d15, [a2] 0	Core1
7: 281627					0x80025358	OR d2, d4	Core1
7: 281628					0x8002535A	LD.W d3, [a2]	Core1
7: 281629					0x8002535C	OR d15, d2	Core1
7: 281630					0x8002535E	OR d15, d3	Core1
7: 281631					0x80025360	ADD d15, 0x1	Core1
7: 281632					0x80025362	ST.W [a3] 0, d15	Core1
7: 281633					0x80025364	LOOP a15, 0x80025352	Core1
7: 281634	0x80027996	CALL 0x80025314	aFun( );	FuncBG_TaskC0			Core0
7: 281635	0x80027996	CALL 0x80025314	aFun( );	FuncBG_TaskC0			Core0
7: 281636		Trigger					WTU_MCX
7: 281637					0x80025352	LD.W d2, [a2]	Core1
7: 281638					0x80025354	LD.W d4, [a2]	Core1
7: 281639					0x80025356	LD.W d15, [a2] 0	Core1
7: 281640					0x80025358	OR d2, d4	Core1
7: 281641					0x8002535A	LD.W d3, [a2]	Core1
7: 281642					0x8002535C	OR d15, d2	Core1
7: 281643					0x8002535E	OR d15, d3	Core1
7: 281644					0x80025360	ADD d15, 0x1	Core1
7: 281645					0x80025362	ST.W [a3] 0, d15	Core1
7: 281646					0x80025364	LOOP a15, 0x80025352	Core1
7: 281647	0x80025314	MOVH.A a2, 0x6000	{	aFun			Core0
7: 281648	0x80025318	MOVH.A a3, 0x7000		aFun			Core0
7: 281649	0x8002531C	LEA a2, [a2] 0x3768		aFun			Core0
7: 281650	0x80025320	LEA a3, [a3] 0x4b70		aFun			Core0
7: 281651	0x80025324	LEA a15, 0x63		aFun			Core0
7: 281652	0x80025328	LD.W d4, [a2]	bar = foo + foo + ...	aFun			Core0
7: 281653	0x8002532A	LD.W d15, [a2] 0		aFun			Core0
7: 281654	0x8002532C	LD.W d3, [a2]		aFun			Core0

Using miniMCDS Trace for AURIX TC2xx/TC3xx

The miniMCDS is a subset of the regular MCDS for the observation of just one trace source and available on selected devices of the AURIX TC2xx and TC3xx family. miniMCDS allows trace based debugging even on production devices. The available trace memory is limited to 8 kByte.

A limited feature set of miniMCDS is supported by UDE as part of the standard debugger licence (UDE-TC-MCA). The extended feature set requires an additional trace license (UDE-TC UEC).

Preparations

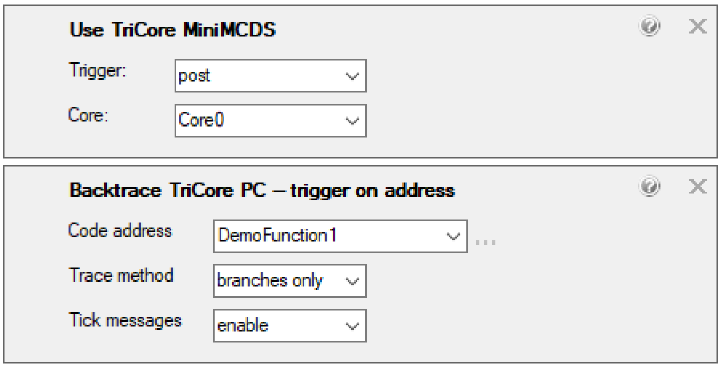
The further steps require AURIX target hardware containing a TC38x. The following preparations are required:

- that the connection to the AURIX target is established, and
- that the samples application `TimeDemo.elf` for TC38x is loaded.

Recording the first Samples

Open a new Trace window via menu **Views** – **Trace**. Open the UEC trace configuration window via menu **Tools** – **Configure Trace** to create a trace configuration for recording the program trace of `Core0` of TC38. Execute the following steps to create the required configuration via menu **Tools – Configure trace …** :

- Drag the **Use TriCore MiniMCDS** configuration block to the configuration area.
- Set **Trigger** to **pre** and select **Core0** from the **Core** dropdown box.
- Drag the **Backtrace TriCore PC – trigger on address** configuration block to the configuration area. This block enables the code trace.
- Set **Code address** to `DemoFunction1` (use **…** to browse for a code label). The trace will be started at this address if **Trigger** is set to **pre** or stops the trace if **Trigger** is set to **post** (see above).
- Enable **Tick messages** in order to get timing information for the recorded instructions.



Use TriCore MiniMCDS

Trigger: post

Core: Core0

Backtrace TriCore PC – trigger on address

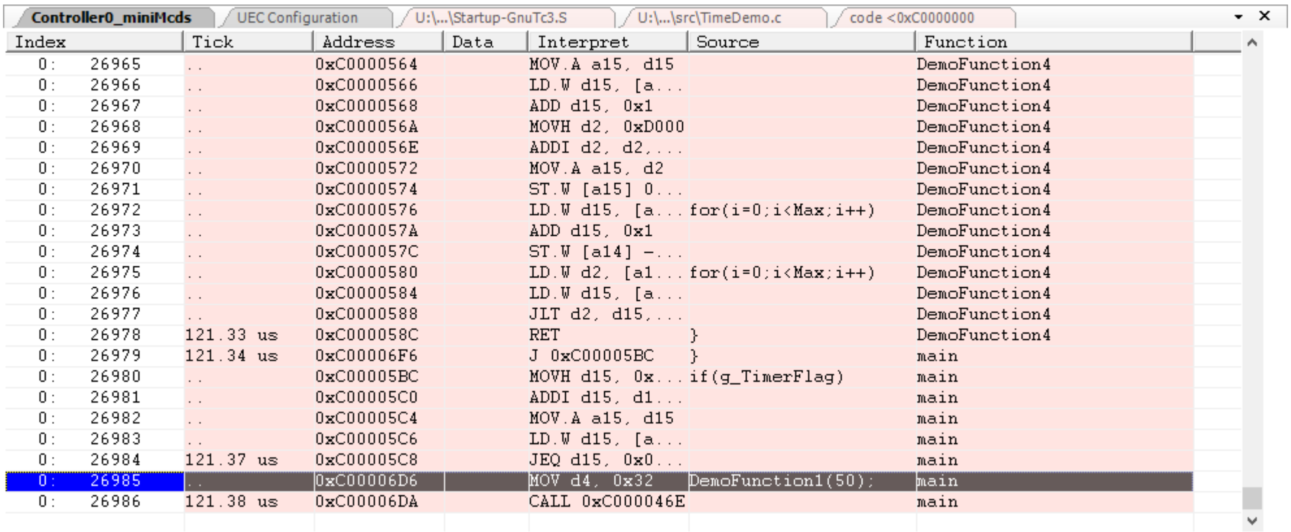
Code address: DemoFunction1

Trace method: branches only

Tick messages: enable

The miniMCDS trace configuration for this special task is now complete.

The trace recording can be started now via toolbar icon or menu **Tools – Start trace...**. The program execution can be started using the menu **Debug** – **Start Program Execution**. The trace recording is automatically stopped when the function `DemoFunction1` is executed. The trace is decoded instantly. Finally, the trace window shows the code that was executed right before the `DemoFunction1` was entered.



Index	Tick	Address	Data	Interpret	Source	Function
0: 26965	..	0xC0000564		MOV.A a15, d15		DemoFunction4
0: 26966	..	0xC0000566		LD.W d15, [a...		DemoFunction4
0: 26967	..	0xC0000568		ADD d15, 0x1		DemoFunction4
0: 26968	..	0xC000056A		MOVH d2, 0xD000		DemoFunction4
0: 26969	..	0xC000056E		ADDI d2, d2,...		DemoFunction4
0: 26970	..	0xC0000572		MOV.A a15, d2		DemoFunction4
0: 26971	..	0xC0000574		ST.W [a15] 0...		DemoFunction4
0: 26972	..	0xC0000576		LD.W d15, [a...	for(i=0;i<Max;i++)	DemoFunction4
0: 26973	..	0xC000057A		ADD d15, 0x1		DemoFunction4
0: 26974	..	0xC000057C		ST.W [a14] -...		DemoFunction4
0: 26975	..	0xC0000580		LD.W d2, [a1...	for(i=0;i<Max;i++)	DemoFunction4
0: 26976	..	0xC0000584		LD.W d15, [a...		DemoFunction4
0: 26977	..	0xC0000588		JLT d2, d15,...		DemoFunction4
0: 26978	121.33 us	0xC000058C		RET	}	DemoFunction4
0: 26979	121.34 us	0xC00006F6		J 0xC00005BC	}	main
0: 26980	..	0xC00005BC		MOVH d15, 0x...	if(g_TimerFlag)	main
0: 26981	..	0xC00005C0		ADDI d15, d1...		main
0: 26982	..	0xC00005C4		MOV.A a15, d15		main
0: 26983	..	0xC00005C6		LD.W d15, [a...		main
0: 26984	121.37 us	0xC00005C8		JEQ d15, 0x0...		main
0: 26985	..	0xC00006D6		MOV d4, 0x32	DemoFunction1(50);	main
0: 26986	121.38 us	0xC00006DA		CALL 0xC000046E		main

An Example with C166S V2 / XC2000 via JTAG/OCDS L1

Starting with Universal Debug Engine

This chapter describes how to connect to Infineon's XC2000 EasyKit. Create a new target workspace and select the default target configuration XC2000 – Infineon – XC2000ED EasyKit – EasyKit with XC2000-ED (JTAG-Debugging). The Universal Debug Engine will now try to connect to the target system. If the connection is successfully established, the following message will appear in the message window via **Views – Other windows – Messages**:

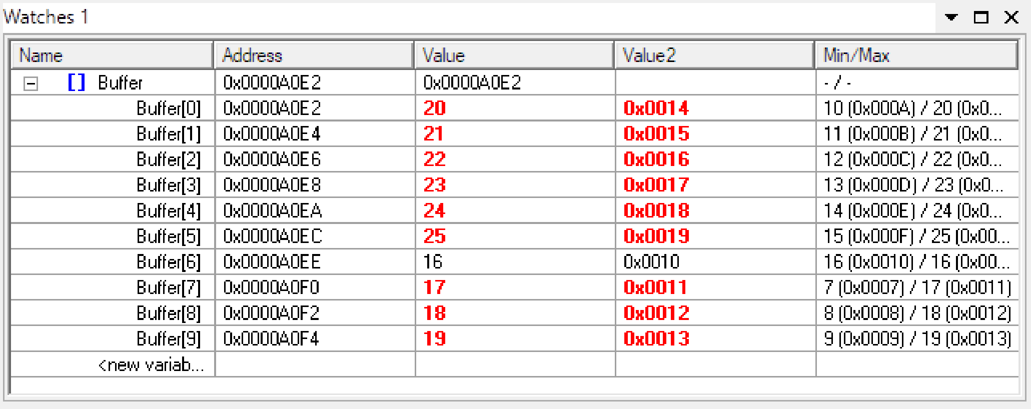
```
UDEDebugServer: Connection to XC2000-ED target established: C166S-V2
Target, JTAG-ID: 0x0018A083
```

Automatic Variables Refresh

For observing variables while the target is running, UDE supports automatic variable refresh. Load the example `<UDE_SAMPLES>\XC2000\EasyKit_XC2000M\TimeDemo\Tasking_IntRam\TimeDemo.out` and enable the Binary and Symbols loading in the **Multicore / multi program loader**.

Open a **Watches** window, insert e.g. the array variable `Buffer`, and unfold it. Open the context menu of the variable and select **Refresh Period**. The dialog **Refresh Period for Watches Entry will appear and** allows configuration of the refresh periods for the selected variable for a running and/or halted target. Enable the refresh for running target and set the period time to 25ms. Click on OK button to close the dialog box and apply the changed settings.

As the `Buffer` variable will be refreshed automatically now, you can watch how the circular buffer is filled. Changed values are highlighted in red.



Watches 1

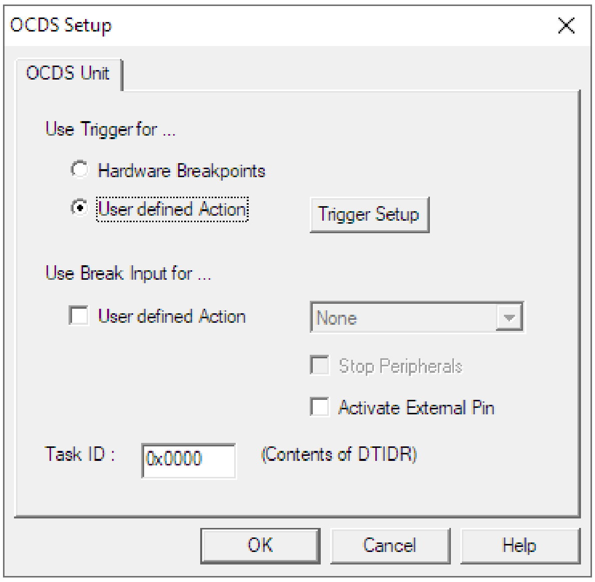
Name	Address	Value	Value2	Min/Max
⊟ [] Buffer	0x0000A0E2	0x0000A0E2		- / -
Buffer[0]	0x0000A0E2	20	0x0014	10 (0x000A) / 20 (0x0...
Buffer[1]	0x0000A0E4	21	0x0015	11 (0x000B) / 21 (0x0...
Buffer[2]	0x0000A0E6	22	0x0016	12 (0x000C) / 22 (0x0...
Buffer[3]	0x0000A0E8	23	0x0017	13 (0x000D) / 23 (0x0...
Buffer[4]	0x0000A0EA	24	0x0018	14 (0x000E) / 24 (0x0...
Buffer[5]	0x0000A0EC	25	0x0019	15 (0x000F) / 25 (0x00...
Buffer[6]	0x0000A0EE	16	0x0010	16 (0x0010) / 16 (0x00...
Buffer[7]	0x0000A0F0	17	0x0011	7 (0x0007) / 17 (0x0011)
Buffer[8]	0x0000A0F2	18	0x0012	8 (0x0008) / 18 (0x0012)
Buffer[9]	0x0000A0F4	19	0x0013	9 (0x0009) / 19 (0x0013)
<new variab...				

Trigger Functions

This chapter demonstrates how to use the OCDS unit of the XC16x derivative to implement trigger functionality. Load the application `timedemo.out` like described before. We want to create a trigger configuration that stops program execution, if a write access to the variable `Buffer[0]` occurs.

In most cases, one can use the name of a variable or its memory location likewise in UDE. If you want to know the memory location, the **Watches** window can assist you. As shown in the screenshot, the **Address** column shows the address of the `Buffer[0]` array element. Additionally, the tooltip of the variable or its components displays the address too, in this case `0xD0002838`.

Open the 'OCDS L1 Setup' dialog using menu entry **Debug – Setup OCDS unit**. Set the check mark 'User Defined Action' in section **Use Trigger for ....** Then click on the button **Trigger Setup**.



Use Trigger for ...

○ Hardware Breakpoints

● User defined Action

Use Break Input for ...

□ User defined Action   None

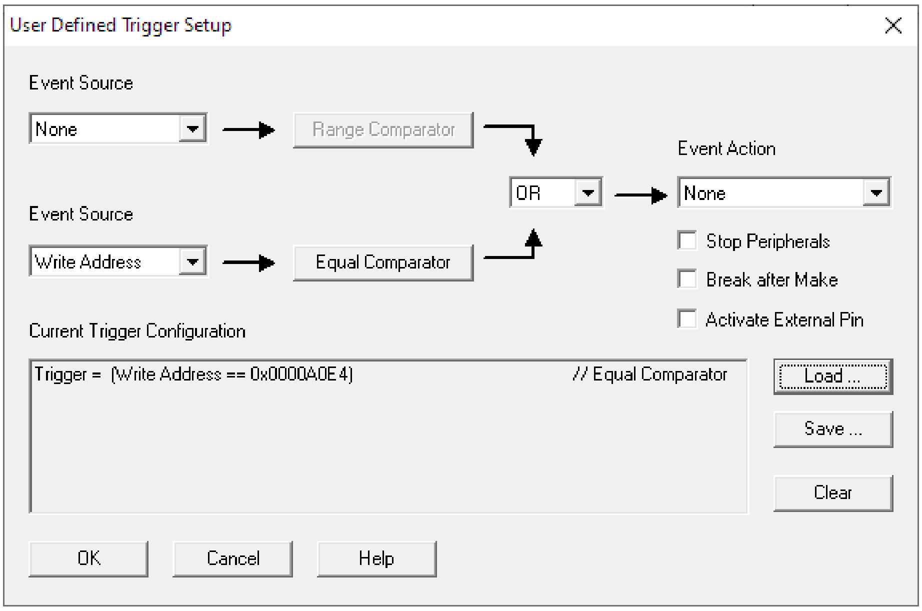
□ Stop Peripherals

□ Activate External Pin

Task ID : 0x0000 (Contents of DTIDR)

This will open the **Trigger Setup** dialog. To detect write access to the variable `Buffer[1]` (a single location at `0x0A0E4`), we will use the Equal Comparator. Therefore, please select the option **Write Address** as **Event Source** for the **Equal Comparator**. This condition will cause a trigger event as soon as the write access occurs. To stop the application in the event of the trigger, select **Break User Application** in the **Event Action** field.

To select the address to be monitored, click on the button **Equal Comparator**. In this example, we want to set the address of the variable to be displayed directly. Select Mode I and enter the address of the variable `0x0000A0E4` into the field Value.



Event Source: None

Event Source: Write Address

OR

Event Action: None

□ Stop Peripherals

□ Break after Make

□ Activate External Pin

Current Trigger Configuration

```
Trigger =  (Write Address == 0x0000A0E4)                // Equal Comparator
```

After closing the dialog, the target is halted automatically. In the command window, the reason is written:

```
UDEDebugServer: Target execution - halted by trigger event - program
location 0x0A0E4
```

Switch the program window to mixed mode view, the IP shows `0x0A0E4`.

An Example with MPC5567 via JTAG

Starting with Universal Debug Engine

When starting a newly installed version of UDE for the first time, a **firmware update** may be executed for the access device (UAD2^{pro}, UAD2^{+}, UAD2^{next}, and UAD3^{+}). This may take some more time than usual for the “target connect” operation. Please **DO NOT** power OFF or unplug the access device during this period!

In this chapter is shown, how to connect to NXP's MPC5567 board using a high-speed connection via an UAD. First launch UDE, create a new workspace and select the default target configuration for PowerPC/MPC5567 **Freescale MPC5567EVB Evalboard with PC5567**.

Loading and Starting of an Executable

Select the menu entry **Load Program** in the **File** menu, browse to folder

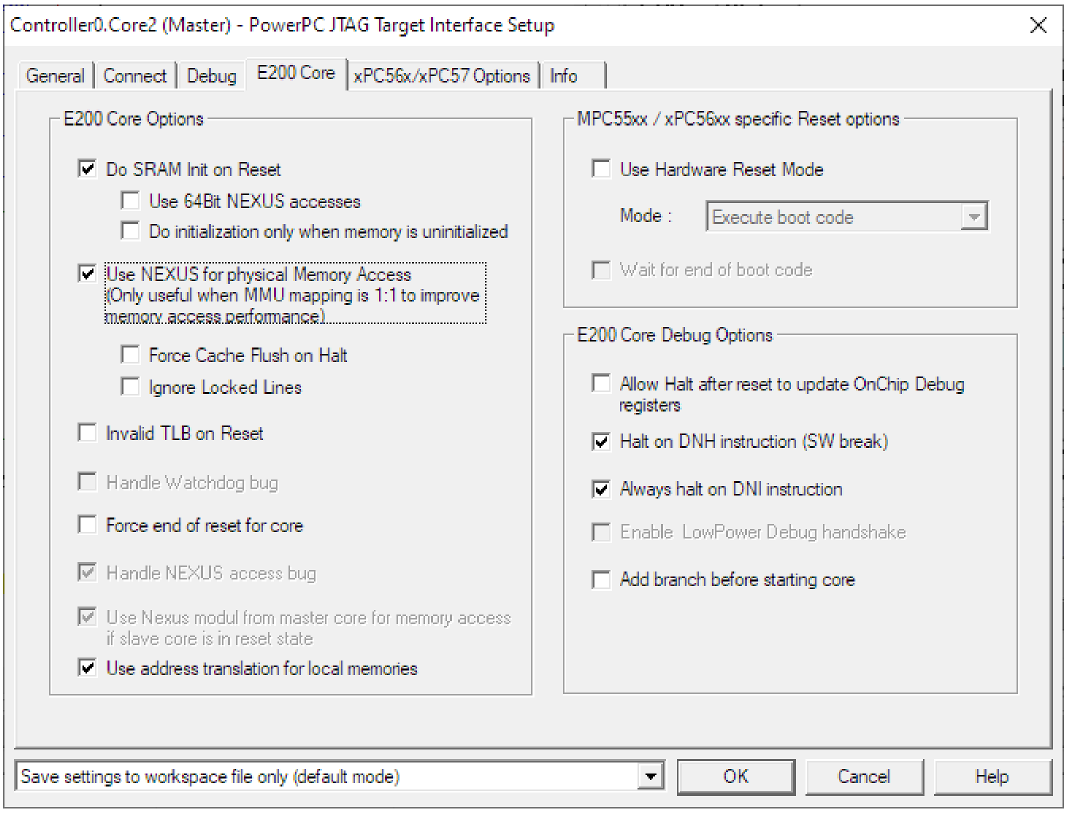
`<UDE_SAMPLE_DIRECTORY>\PPC\MPC5567\timedemo\obj-iRam`

and load the file `Timedemo.elf`

Please note, that the Instruction Pointer (IP) is placed at the beginning of the start-up code of the loaded application. Now start the program via menu entry **Debug - Start Program Execution F5**.

Automatic Variables Refresh

For observing variables while the target is running, UDE supports automatic variable refresh. For using this feature with MPC55xx, the NEXUS access must be enabled. To enable the NEXUS access, open the **Target Interface Setup** dialog page **E200 Core** via menu **Config – Target interface ... – E200 Code** and set the check mark at option **Use NEXUS ..** to enable the access.



Controller0.Core2 (Master) - PowerPC JTAG Target Interface Setup
General
Connect
Debug
E200 Core
xPC56x/xPC57 Options
Info
OK
Cancel
Help

E200 Core Options

☑ Do SRAM Init on Reset

☐ Use 64Bit NEXUS accesses

☐ Do initialization only when memory is uninitialized

☑ Use NEXUS for physical Memory Access (Only useful when MMU mapping is 1:1 to improve memory access performance)

☐ Force Cache Flush on Halt

☐ Ignore Locked Lines

☐ Invalid TLB on Reset

☐ Handle Watchdog bug

☐ Force end of reset for core

☑ Handle NEXUS access bug

☑ Use Nexus modul from master core for memory access if slave core is in reset state

☑ Use address translation for local memories

MPC55xx / xPC56xx specific Reset options

☐ Use Hardware Reset Mode

Mode : Execute boot code

☐ Wait for end of boot code

E200 Core Debug Options

☐ Allow Halt after reset to update OnChip Debug registers

☑ Halt on DNH instruction (SW break)

☑ Always halt on DNI instruction

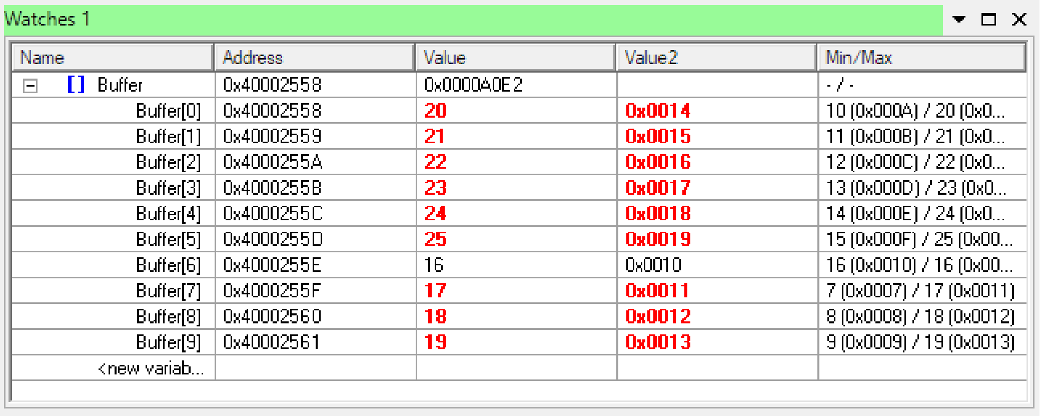
☐ Enable LowPower Debug handshake

☐ Add branch before starting core

Save settings to workspace file only (default mode)

Open a **Watches** window, insert e.g. the array variable `Buffer`, and unfold it. Open the context menu of the variable and select **Refresh Period**. The dialog **Refresh Period for Watches Entry will appear and** allows configuration of the refresh periods for the selected variable for a running and/or halted target. Enable the refresh for running target and set the period time to 25ms. Click on OK button to close the dialog box and apply the changed settings.

As the Buffer variable will be refreshed automatically now, you can watch how the circular buffer is filled. Changed values are highlighted in red.



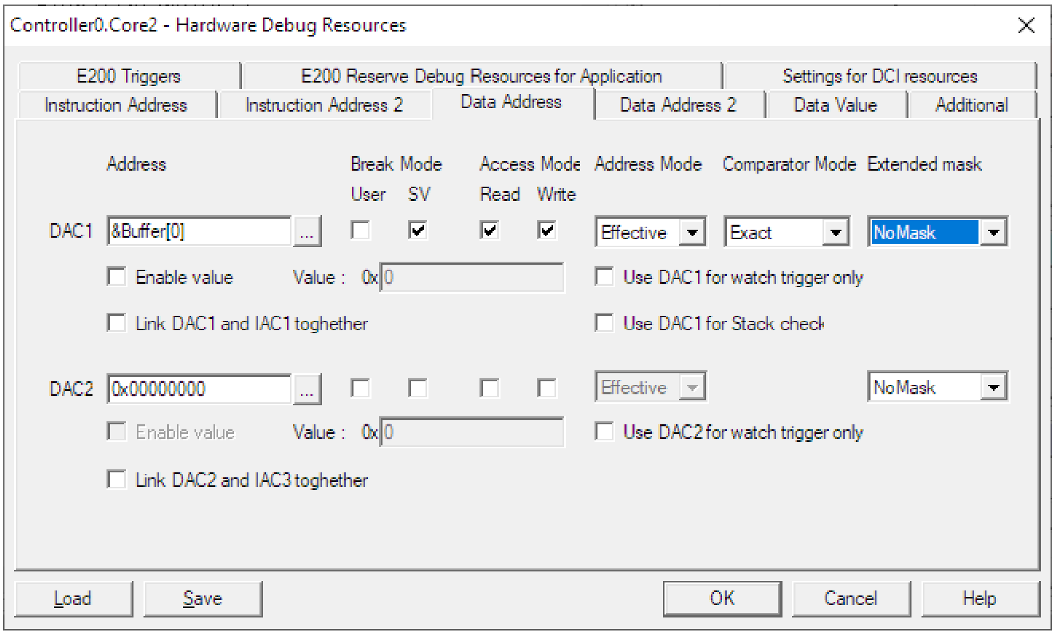
Name	Address	Value	Value2	Min/Max
⊟ [] Buffer	0x40002558	0x0000A0E2		- / -
Buffer[0]	0x40002558	20	0x0014	10 (0x000A) / 20 (0x0...
Buffer[1]	0x40002559	21	0x0015	11 (0x000B) / 21 (0x0...
Buffer[2]	0x4000255A	22	0x0016	12 (0x000C) / 22 (0x0...
Buffer[3]	0x4000255B	23	0x0017	13 (0x000D) / 23 (0x0...
Buffer[4]	0x4000255C	24	0x0018	14 (0x000E) / 24 (0x0...
Buffer[5]	0x4000255D	25	0x0019	15 (0x000F) / 25 (0x00...
Buffer[6]	0x4000255E	16	0x0010	16 (0x0010) / 16 (0x00...
Buffer[7]	0x4000255F	17	0x0011	7 (0x0007) / 17 (0x0011)
Buffer[8]	0x40002560	18	0x0012	8 (0x0008) / 18 (0x0012)
Buffer[9]	0x40002561	19	0x0013	9 (0x0009) / 19 (0x0013)
<new variab...				

Please note, if the integrated caching mechanism of the PowerPC MCU is enabled, the view will only show the memory content. Modified cached values are not visible until a cache write back is executed. A cache write back is executed, for example, if the program execution will be interrupted by manual user break or breakpoint events.

Trigger Functions

This chapter demonstrates how the use the Power Architecture Book E defined triggers of the MPC55xx derivatives for debugging purposes. Create a workspace and load the application `TimeDemo.elf` as described before. We want to create a trigger configuration that stops program execution, if a write access to the variable `Buffer[0]` occurs.

Open the **Hardware Debug Resources** dialog by menu **Debug – Setup Trigger unit**. Select the tab **Data Address**. We want to use DAC1 as trigger comparator. Enter the address of `Buffer[0]` into the address box. For simplification C-style expressions can be used, so simple enter `&Buffer[0]`.



DAC1 &Buffer[0] ... User □ SV ☑ Read ☑ Write ☑ Effective Exact NoMask

□ Enable value Value : 0x 0 □ Use DAC1 for watch trigger only

□ Link DAC1 and IAC1 toghether □ Use DAC1 for Stack check

DAC2 0x00000000 ... User □ SV □ Read □ Write □ Effective NoMask

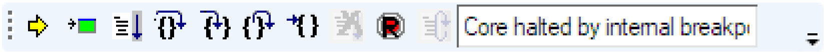
□ Enable value Value : 0x 0 □ Use DAC2 for watch trigger only

□ Link DAC2 and IAC3 toghether

Enable **User** and **SV** (Supervisor) as Break Mode, select **Write** Access mode, **Effective** address and **Exact** comparator.

Start the application.

The application stops, when the first read/write to `Buffer[0]` occurs. The controller state shows the new state: **Core halted by internal breakpoint**.


Core halted by internal breakp

Hints for using the MPC55xx via JTAG

For accessing memory on a running target, the NEXUS functionality is required. The NEXUS module on the MPC55xx allows access to the physical system memory during runtime. Therefore, a 1:1 mapping of the MMU is also required, since the MMU cannot be disabled for the core. If a 1:1 mapping is not ensured, the NEXUS feature is not useable!

For accessing physical memory resources (e.g. Special function registers), the MMU is analyzed by the debugger to find out the correct effective addresses. If the requested physical address is not mapped, the debugger is able to set up the TLB entries inside the MMU, when the special option **Allow MMU setup for physical access** inside of the Target Interface's Setup is enabled.

Creating hardware-specific Target Configurations

The chapters above used Starterkits with predefined target configurations. In this chapter, you will learn how you can build a target configuration with your custom specific target hardware yourself. This is realized, using the **Target Configuration Wizard**.

First, you have to define the target hardware that may have multiple different microcontrollers and memories.

In this example, a fictitious target equipped with one C167CR and two 29F010 connected to /CS0 of the controller is used. Furthermore, the target has one external 16-bit sRAM connected to /CS1. The communication channel is the ASC0 with RS232 driver. The target controller is started via bootstrap loading.

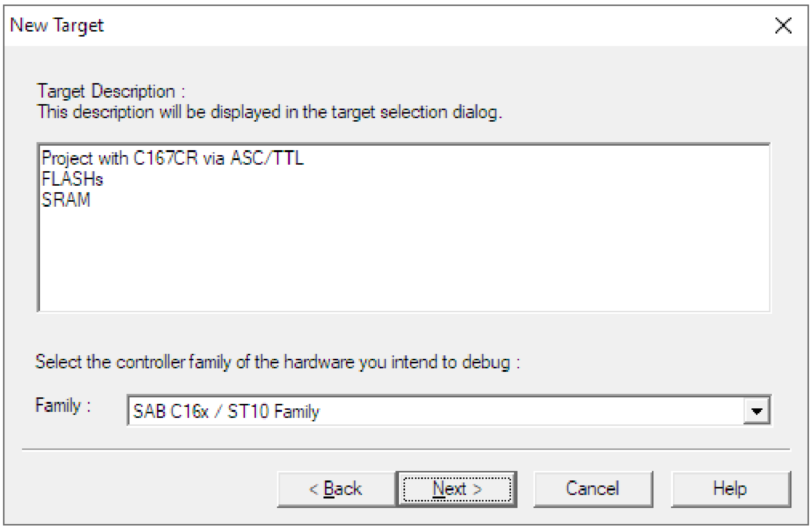
- Controller: C167CR
- FLASH: 2x 8-bit FLASH NX29F010 at /CS0
- sRAM: 1x 16-bit RAM at /CS1
- Communication Channel: ASC0 via RS232
- Monitor: Bootstrap Loader Monitor

Creating a new workspace

A workspace always refers to a single target system. Choose **File – New Workspace** from the menu and enter a new name for the workspace in the appearing dialog box. Press the **OK** button to apply the entered name.

Invoking the Wizard

The next dialog is the Target Selection Dialog. Press the button **New** and click **Next** to create a new target configuration. The next dialog allows both describing the new target and choosing the controller family. If you do not enter a description, the file name will be displayed in this dialog only.


New Target

Target Description :
This description will be displayed in the target selection dialog.

Project with C167CR via ASC/TTL
FLASHs
SRAM

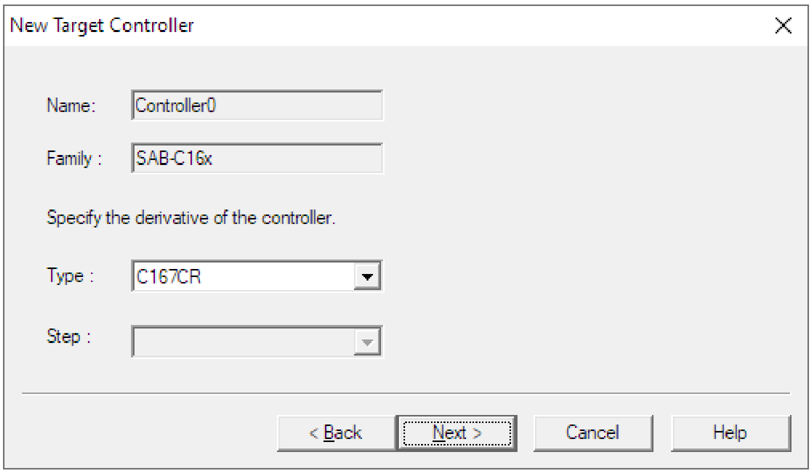
Select the controller family of the hardware you intend to debug :

Family : SAB C16x / ST10 Family

Press the **Next** button, when you are ready.

Selecting the Controller Derivative

The next step is the selection of the used derivative with the correct databases of the used derivative. It is important to select the correct corresponding type of your derivative. Select the **C167CR** derivative for this example and click **Next**.


New Target Controller

Name: Controller0

Family : SAB-C16x

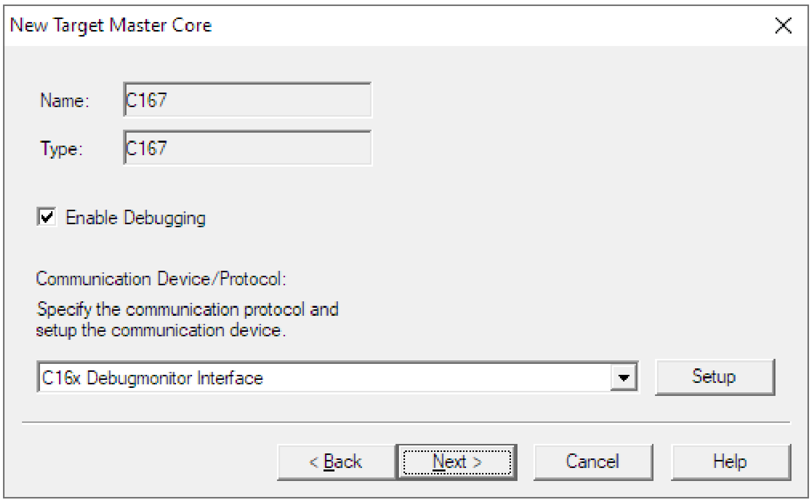
Specify the derivative of the controller.

Type : C167CR

Step :

Selecting the Target Interface

In this step, the target communication interface has to be selected to ASC0 based monitor via bootstrap loading. The **C16x Debugmonitor Interface** must be selected.


New Target Master Core

Name: C167

Type: C167

☑ Enable Debugging

Communication Device/Protocol:

Specify the communication protocol and setup the communication device.

C16x Debugmonitor Interface

To specify the interface settings e.g. the baud rate and the communication channel properties to the target click **Setup**.

Setting up the Target Interface

The dialog is divided in a number of pages for setting up the Target Interface. Please check the content of each page for correct content.

Booting the controller by using the bootstrap loading mechanism of the C167CR will be done within three steps.

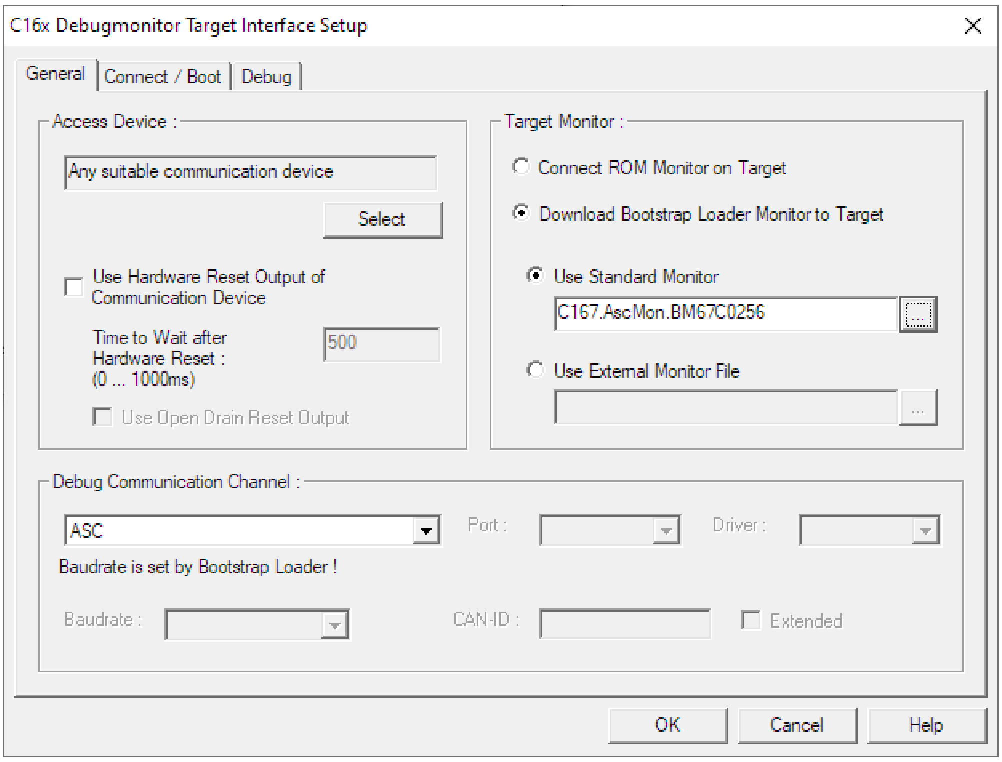
1. Via the internal bootstrap loading the Second Level Loader is transferred to the internal RAM and will be started.

2. This Second Level Loader loads the primary Boot Code.

3. The Boot Code initializes the controller system and loads the application or monitor code.

General

Select and configure the used communication hardware.



Access Device :

Any suitable communication device

□ Use Hardware Reset Output of Communication Device

Time to Wait after Hardware Reset : (0 ... 1000ms) 500

□ Use Open Drain Reset Output

Target Monitor :

○ Connect ROM Monitor on Target

● Download Bootstrap Loader Monitor to Target

● Use Standard Monitor

C167.AscMon.BM67C0256

○ Use External Monitor File

Debug Communication Channel :

ASC

Port :

Driver :

Baudrate is set by Bootstrap Loader !

Baudrate :

CAN-ID :

□ Extended

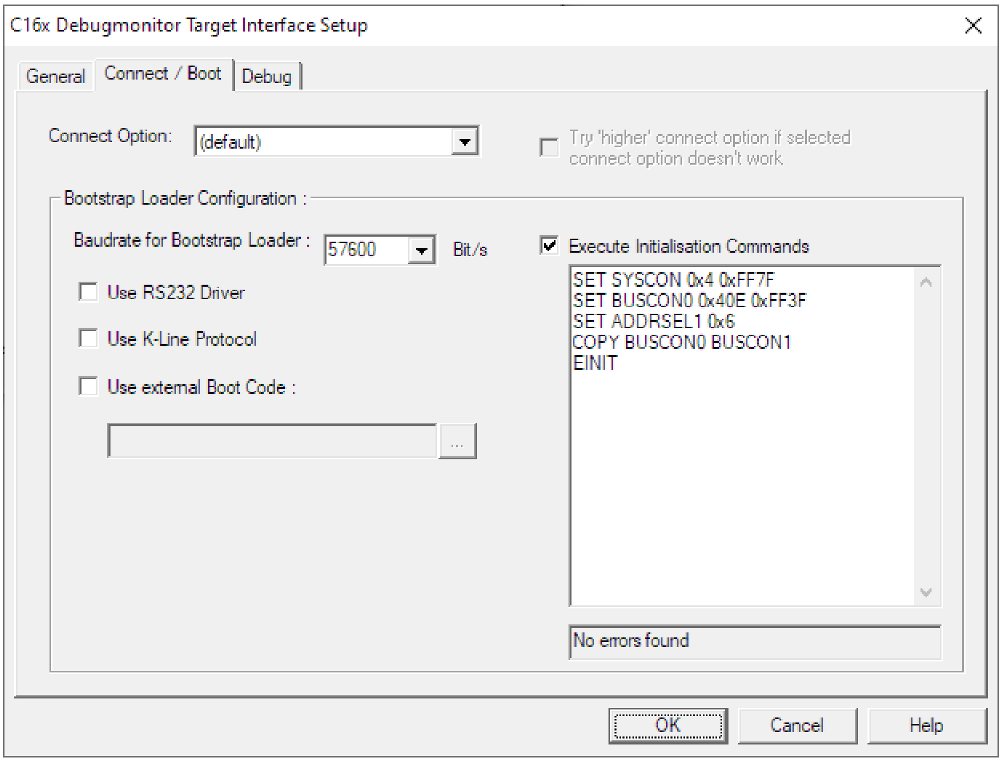
In this example, a special **Target Monitor** using the bootstrap loading mechanism has to be used. The various offered standard ASC monitors differs only in the location where the monitor is located. For example, `C167.ASCMon.BM67C0256` means a C167 ASC0 monitor is located at `3F290h – 3FFFFh`, below the 256 kByte RAM memory boundary. Click the **...** button to select a different standard monitor.

Select ASC as **Debug Communication Channel**.

Connect/Boot

In this example, the settings of **Connect Option** are not relevant, because the UAD cannot cause a reset on your target via an ASC0/TTL line. Leave the **Connect Option** on default.

In this example, the standard Boot Code is used. The **Use external Boot Code** setting is required in rare cases only, for the system loading process.



Connect Option: (default)

□ Try 'higher' connect option if selected connect option doesn't work

Bootstrap Loader Configuration :

Baudrate for Bootstrap Loader : 57600 Bit/s

□ Use RS232 Driver

□ Use K-Line Protocol

□ Use external Boot Code :

☑ Execute Initialisation Commands

```
SET SYSCON 0x4 0xFF7F
SET BUSCON0 0x40E 0xFF3F
SET ADDRSEL1 0x6
COPY BUSCON0 BUSCON1
EINIT
```

No errors found

The **Initialization Commands** are executed by the Boot Code, during the system configuration process. The commands initialize the memories, which are connected on /CS0 and /CS1 via a demultiplexed 16-bit bus. The address range of the /CS1 memory is mapped `0x00000` to `0xFFFFF`. Finally, the EINIT instruction will be executed.

Refer to the UDE manual for a complete set of initialization commands and to the C167 architecture manual for a description of external bus unit of the C167 microcontroller architecture.

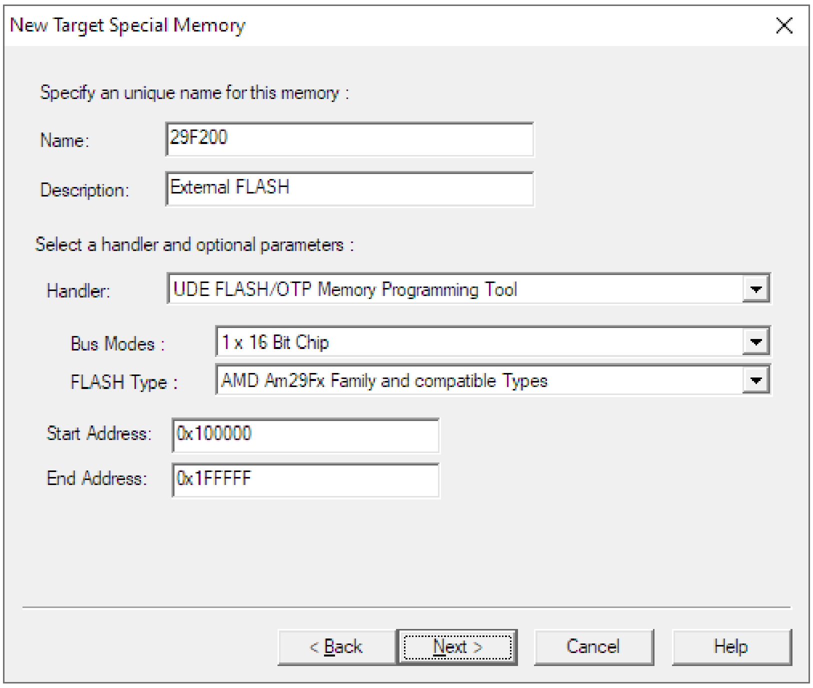
Close the Target Interface Setup via the **OK** button and click **Next**. Later modifications can be made within UDE.

Configuring the FLASH memory

In the dialog **Target Specify Memories**, you must set the number of memories at least to **1**, if you want to use FLASH programming support. Click **Next**.

Enter in the dialog **Special Memories** a unique name for the flash name and the description. The description will be displayed when you select a memory device in the FLASH/OTP Memory Programming Tool.

For each specially treated memory device, a special Handler is required. Select for FLASH support the **UDE FLASH/OTP Memory Programming Tool**. Setup both the bus mode organization of the chips and an address range, where the memory is visible. These settings must correspond with the configuration of the external bus controller done by the initialization command.



New Target Special Memory

Specify an unique name for this memory :

Name: 29F200

Description: External FLASH

Select a handler and optional parameters :

Handler: UDE FLASH/OTP Memory Programming Tool

Bus Modes : 1 x 16 Bit Chip

FLASH Type : AMD Am29Fx Family and compatible Types

Start Address: 0x100000

End Address: 0x1FFFFF

Click **Next**.

With these steps, the FLASH/OTP Memory Programming is prepared. Further steps are required, if you are using the Programming tool. Please refer to the user's manual chapter **FLASH Programming** for further information.

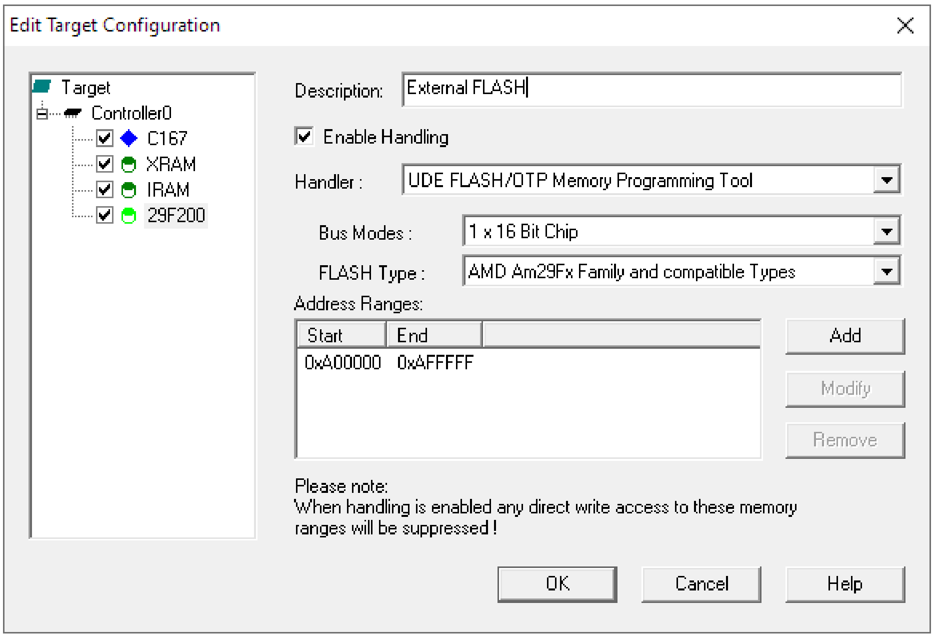
Finish the Wizard

Enter the name of the file where you intend to save the configuration of target. Usually the Target Configuration files are stored in `<UDE_DIRECTORY>/TARGETS` folder.

By pressing the **Finish** button, you apply the settings to the target configuration. The debugger loads all necessary components and connects to the target. If the following message is displayed in the Command window, you were successfully.

```
UDEDebugServer: Connection to C167CR target monitor established:
bootmon167 V3.55 11/10
```

To edit this target configuration, you can select the menu **Config – Target Configuration**. In the appearing dialog, you can modify the target configuration.



Description: External FLASH

☑ Enable Handling

Handler : UDE FLASH/OTP Memory Programming Tool

Bus Modes : 1 x 16 Bit Chip

FLASH Type : AMD Am29Fx Family and compatible Types

Address Ranges:

Start	End
0xA00000	0xAFFFFF

Please note:
When handling is enabled any direct write access to these memory ranges will be suppressed !

Conclusion

Congratulations! You have learned about the basic features of UDE. This will enable you to load and debug an application using UDE with a Starterkit board as the target system.

You may now create your own applications using the GNU, the Tasking, the Keil or the Greenhills C-Compiler and debug them with UDE. As UDE is constantly improved, please check out our web site at **https://www.pls-mc.com** for the latest version of UDE. Additionally, if you have any questions or if you need any help regarding your development tools we would like to encourage you to contact the PLS Support Team via e-mail at **support@pls-mc.com** or via phone at **+49 35722 384 0**.

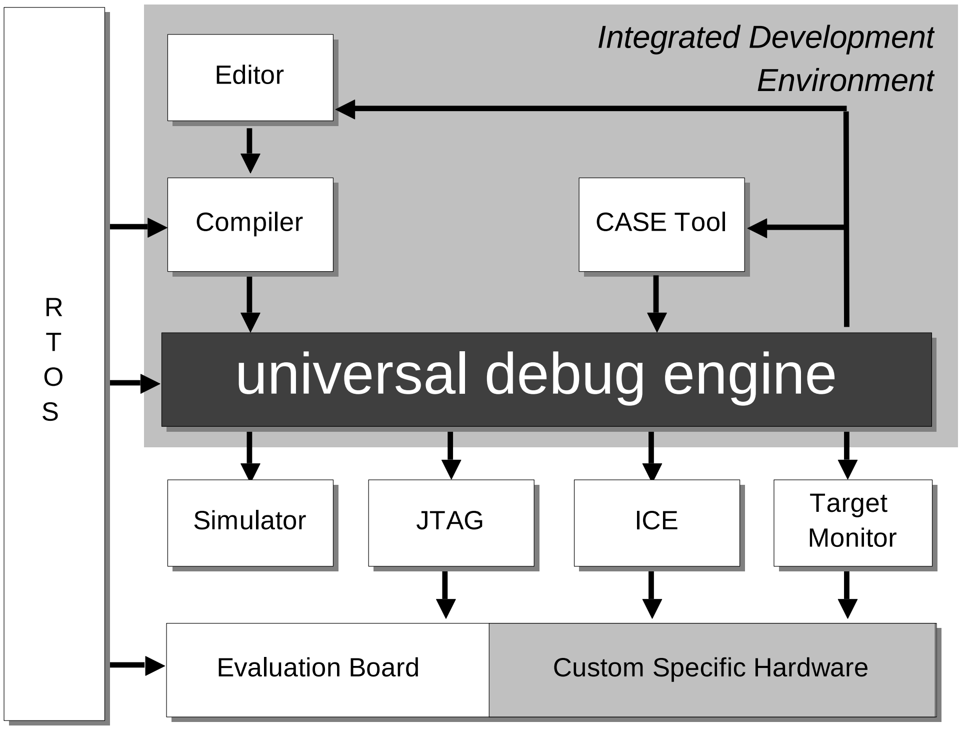
Thank you for using Universal Debug Engine.

User's Guide

Introduction

The **Universal Debug Engine (UDE)** is one of the most powerful development workbenches available for the AURIX, TriCore, Power Architecture, Cortex, ARM7, ARM9, ARM11, C166, XC166, XC2000, XE166, XScale, RH850, R-Car, SuperH SH-2A and further microcontroller families.

The UDE workbench lets you edit and organize your projects, supports you while building the applications and lets you run and test your software on the supported microcontrollers-based customer specific hardware or different evaluation boards in a very convenient and cost-efficient way. The vast capabilities of the UDE High-End Debugger enable you to develop fast and reliable software as well as to get short turn-around times for your microcontroller projects.



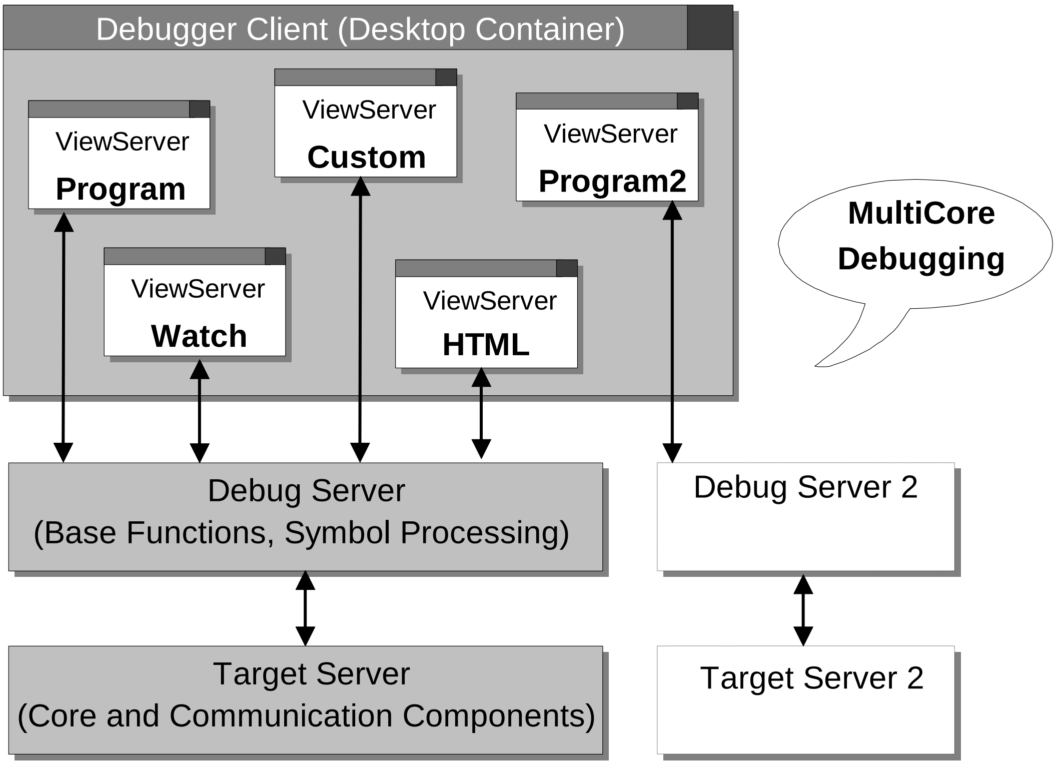
Hand-in-hand with our target access hardware the Universal Debug Engine offers a flexible debug platform. It allows...

- High-speed and flexible access to target systems
- Full-featured on-chip debug modules, OCDS, DAP, EmbeddedICE, CoreSight, SWD support

- Full code and data trace via MCDS, ETM, ETB, NEXUS, CoreSight and triggered Transfer support
- Multi-Core Debugging
- Various Operation system support (RTOS)
- Support of further Third-Party Tools.

Architecture of Universal Debug Engine

The **Universal Debug Engine (UDE)** is a new concept of our cross debugger based on a set of standard components and core specific components. UDE supports different cores and multi core debugging. The system is built based on components to ensure extension with additional cores and development tools. This way, the user is able to extend the debug system due to custom requirements.



The UDE concept splits the basic debugger tasks into three independent program parts:

- The user interface (debugger client) works as a client for both servers, generates requests for target data access and symbol processing and displays the results of the processed requests after receiving the ready signal.
- The symbol engine (debug server) works as an independent server that processes requests to resolve symbolic relations of the program code. If the request is processed, the requesting client receives a message that the resolution is available.
- The target communication server (target server) serves all requests for downloading program code into the target and accessing target data to display target system states (program variables etc.). The target core specific debugger engines implement the basic debugger functions of a specific core (e.g. debugger engine for TriCore, PCP, and C166 ...).

The partitioning of debugger tasks into client/server architecture significantly enhances the performance of complex debugger tasks. To add debug support for additional cores is very simple due to the client/server architecture. One target communication server and usually one symbol server must be included to add debug support for a particular core. If several cores use the program code of the same application, no additional symbol server is necessary for the core. The multiple independent engines enhance utilization of the host CPU as well as the turn-around cycle of the debug process decreases.

Moreover, the architecture of an UDE Debugger contains following parts:

- The desktop frame works as a container for several components. The container is based on framework to instantiate windows of the basic debugger engines and export interfaces to extend the user interface dynamically (e.g. add menu entries, tool bars).
- The windows implement standard debugger functions (Program window, Watches window, Memory windows, Register windows etc.), programmable windows based on page description languages (HTML) and application specific windows (special windows based on e.g. COM components).

Using On-line Help

The **Universal Debug Engine** provides an integrated, context sensitive help. To use this help simply press the **F1** key. You may also select **Help Viewer Window** from menu **Help** for viewing the UDE Manual. This manual introduces the architecture of UDE and describes the UDE main features by examples.

Furthermore, please visit our Web site on the Internet at **https://www.pls-mc.com**. There you can obtain the newest information and download the latest version of UDE.

Project Management

Working with Projects

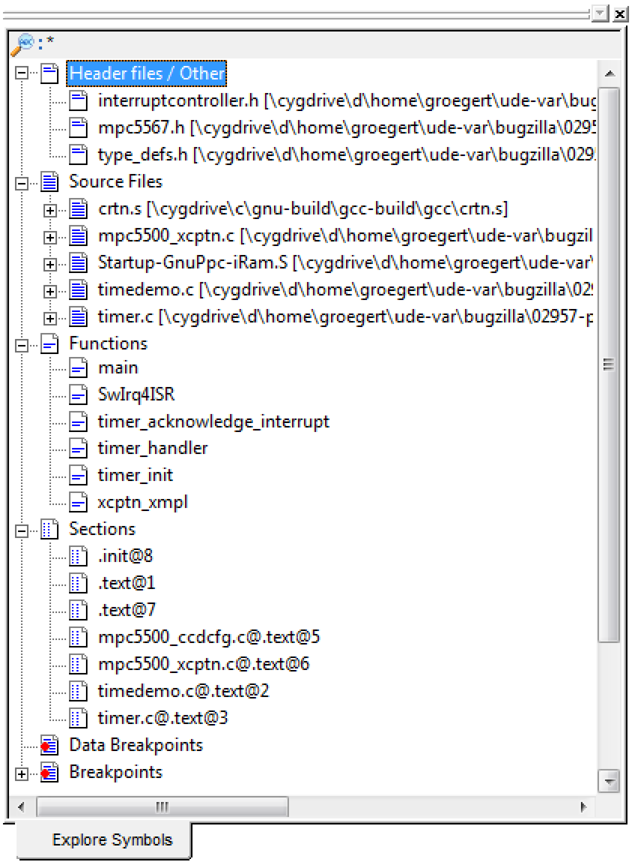
Projects let you organize your work more efficient by combining all target specific and desktop settings in one workspace. This workspace stores all project relevant files and settings of UDE.

A main role of the project management plays the source files and their associated binaries. These files are displayed in the workspace window. The view is subdivided into file names and functions.

If the workspace window is invisible, select menu entry **Views – Target Manager** to open the window.

In the **Core Symbols** window of your current workspace, the source files are grouped under the folder Source files. If the folder content is invisible, click on the [+] item or double-click on the 'Source files' folder to expand it. To collapse a folder in the workspace window, click on [-].

An easy way to open a new source code window is by double-clicking on the corresponding source file entry. In a similar way, it is possible to view a function of the source code by clicking on the function name displayed in the folder Functions.



```
:*
Header files / Other
    interruptcontroller.h [\cygdrive\d\home\groegert\ude-var\bug
    mpc5567.h [\cygdrive\d\home\groegert\ude-var\bugzilla\0295
    type_defs.h [\cygdrive\d\home\groegert\ude-var\bugzilla\029
Source Files
    crtn.s [\cygdrive\c\gnu-build\gcc-build\gcc\crtn.s]
    mpc5500_xcptn.c [\cygdrive\d\home\groegert\ude-var\bugzil
    Startup-GnuPpc-iRam.S [\cygdrive\d\home\groegert\ude-var
    timedemo.c [\cygdrive\d\home\groegert\ude-var\bugzilla\02
    timer.c [\cygdrive\d\home\groegert\ude-var\bugzilla\02957-p
Functions
    main
    SwIrq4ISR
    timer_acknowledge_interrupt
    timer_handler
    timer_init
    xcptn_xmpl
Sections
    .init@8
    .text@1
    .text@7
    mpc5500_ccdcfg.c@.text@5
    mpc5500_xcptn.c@.text@6
    timedemo.c@.text@2
    timer.c@.text@3
Data Breakpoints
Breakpoints
```

Creating a New Project

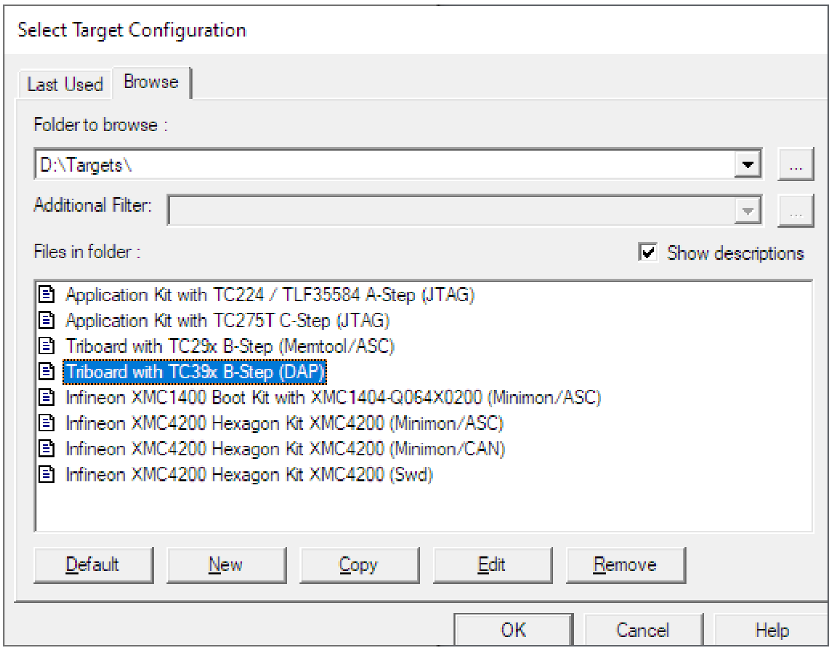
If you want to build a new project, create a new workspace that contains all settings and files of the new project. Select **New Workspace** from the **File** menu of UDE and choose a new file from the file selection box. Usually, the workspace file is located in the project folder.

Note the possibility to create new folders in the file selection box: Right-click into an empty area of the file selection field for the context menu and select **New – Folders**. This way you can create new folders without the need of an additional Explorer window.

When starting a newly installed version of UDE for the first time, a **firmware update** may be executed for the access device (UAD2^{pro}, UAD2^{+}, UAD2^{next}, and UAD3^{+}). This may take some more time than usual for the “target connect” operation. Please **DO NOT** power OFF or unplug the access device during this period!

Select Target Configuration

After choosing the workspace file, you may select a predefined target configuration, create a new or derive an adapted configuration.



Select Target Configuration

Folder to browse : D:\Targets\

Additional Filter:

Files in folder : ☑ Show descriptions

- Application Kit with TC224 / TLF35584 A-Step (JTAG)
- Application Kit with TC275T C-Step (JTAG)
- Triboard with TC29x B-Step (Memtool/ASC)
- Triboard with TC39x B-Step (DAP)
- Infineon XMC1400 Boot Kit with XMC1404-Q064X0200 (Minimon/ASC)
- Infineon XMC4200 Hexagon Kit XMC4200 (Minimon/ASC)
- Infineon XMC4200 Hexagon Kit XMC4200 (Minimon/CAN)
- Infineon XMC4200 Hexagon Kit XMC4200 (Swd)

Press the **Default** button to select a predefined configuration delivered with UDE.

Press the **New** button to create a new configuration and follow the Configuration wizard.

Select a predefined target configuration and press the **Copy** button to derive a new configuration from the selected configuration. This is recommending for adaptation of a similar configuration.

Select a predefined target configuration and press **Edit** to change an existing configuration. Please note that the changes are saved permanently.

Loading a Project

Open the **File** menu and select **Open Workspace**. Select a valid workspace file in the opened file selection box. Usually, workspace files are named with the file extension `*.wsx`.

To load the application binary file, use the **Load Program** from **File** menu and select the binary file.

Saving Project Settings

Usually the project settings are stored automatically if the workspace or the UDE will be closed. However, if you want to create a new project that is derived from a current project you may store the current project under a different path or project name. To do so, choose menu **File – Save Workspace as ...**, select a new project path, and project name.

Closing a Project

To close the current workspace, select menu **File – Close Workspace**. This will save all settings of UDE, view windows and their contents, paths and names of loaded files and will close your current project. The project and its workspace are also saved when UDE is closed.

Command line options of UDE

UDE can be parameterized via command line interface to automatically open or create a workspace and execute a start-up script.

Synopsis:

```
    UDE[.exe] [ -p<wsxfile>] [ -s<scriptfile>]
```

Options:

```
      -p<wsxfile>       ... open or create a workspace
      -s<scriptfile>    ... run a startup script
      -d<logfile>       ... write diagnostic output into file
```

Preparing a binary File

The Universal Debug Engine supports the compiler tool chain of various compiler manufacturers. Please see the compatibility list for a selection of suitable compilers and the compiler manual for correct usage of the compiler.

The following output formats of binary and debug symbol data are supported by UDE:

Output format	Expected content
*.elf	ELF/DWARF binary object file with debug information
*.out	Binary objects file with debug information
*.axf	ELF/DWARF binary objects file with debug information
*.abs	Binary objects file
*.hex *.h66 *.h86	Intel HEX file, ASCII text
*.bin	Intel binary objects file
*.sre	Motorola S records file, ASCII text
*.s19	Motorola S records file, ASCII text

Compiler Support

The recommended file extension for the output file are `*.out` (C166 derivatives) and `*.elf` (all supported derivatives). If you want use the HLL source code-debugging feature in the debugger, the corresponding debug information can be linked to the executable file or can be loaded as separate file. See the compiler manual for the correct options.

Compiler Support for C16x, XC166, XC2000, XE166

The Universal Debug Engine supports the Tasking OUT format, the GNU OUT format and the Keil OMF166 format for binaries.

To build such a file applies the following compiler switches:

Compiler	**Command line options for output**
Tasking C/C++ Compiler	Compiler: "`-g`"
Keil C Compiler	Compiler: "`DEBUG`"
GNU C/C++ Compiler (HighTec)	Compiler: "`-g`"

The recommended file extension for the output file is `*.OUT`.

Compiler Support for AURIX, TriCore

The Universal Debug Engine supports the ELF/DWARF 1.1, ELF/DWARF 2 format containing both the binary target pattern and the debug information.

To build such a file applies the following compiler switches:

Compiler	**Command line options for ELF/DWARF output**
Tasking C/C++ Compiler for AURIX, TriCore	Compiler: "`-g `" PCP Assembler: “`-gal`” Locator: "`-f4`"
GNU C/C++ Compiler for AURIX, TriCore (HighTec)	Compiler (gcc): "`-g -O0`" PCP Assembler: “`-Wa, --gdwarf2`”
Wind River C/C++ Compiler for AURIX, TriCore	Compiler: "`-g -O0`"
Greenhills C/C++ Compiler for AURIX, TriCore	Compiler (cctri): "`-G -dwarf `"

Compiler Support for PowerArchitecture

The Universal Debug Engine supports the ELF/DWARF 1.1, ELF/DWARF 2 format containing both the binary target pattern and the debug information.

To build such a file applies the following compiler switches:

Compiler	**Command line options for ELF/DWARF output**
GNU C/C++ Compiler for Power Architecture	Compiler : "`-g -O0`"
Wind River C/C++ Compiler for Power Architecture	Compiler: "`-g -O0`"
NXP C/C++ Compiler for Power Architecture	Compiler: "`-g `"
Byte Craft eTPU Compiler	Compiler: "`-g `"

Compiler Support for Cortex, ARM7, ARM9, ARM11, XScale

The Universal Debug Engine supports the ELF/DWARF 1.1, ELF/DWARF 2 format containing both the binary target pattern and the debug information.

To build such a file applies the following compiler switches:

Compiler	Command line options for ELF/DWARF output
RealView MDK-ARM and RealView Development Suite for Cortex, ARM7, ARM9, ARM11	Compiler: "`DEBUG`" Assembler: “`--debug --dwarf2`”
GNU C/C++ Compiler for ARM7, ARM9, ARM11 (HighTec)	Compiler: "`-g -O0`"
Wind River C/C++ Compiler for Cortex, ARM7, ARM9, ARM11	Compiler: "`-g -O0`"
C Compiler for ARM7, ARM9, ARM11 (ImageCraft)	Compiler: "`-g` "

The recommended file extension for the output file is `*.elf`.

HexFile Support

In addition, the simple Intel hex file and the Motorola S record format are supported.

Connecting the target system

Before debugging an application, a debug communication channel to the target system must be established. The following chapter gives you an overview about the offered solutions of the Universal Debug Engine and its add-ons.

Overview about Debug Communication Channels

Communication Channel via JTAG, DAP, SWD, OnCE, COP interfaces

JTAG, DAP, SWD, OnCE, COP are examples of debug interfaces, which provides debug communication channels to AURIX, TriCore, Power Architecture, Cortex, ARM, SuperH, XE166, XC2000 microcontrollers.

Communication Channel via monitor-based ASC, SSC, CAN, 3Pin interfaces

UDE supports asynchronous and synchronous communication to the target system, too. The monitors can be available as RAM monitor for bootstrap loading or as ROM monitor. ROM monitor solutions are available whithin the Monitor Development Tool offered by PLS.

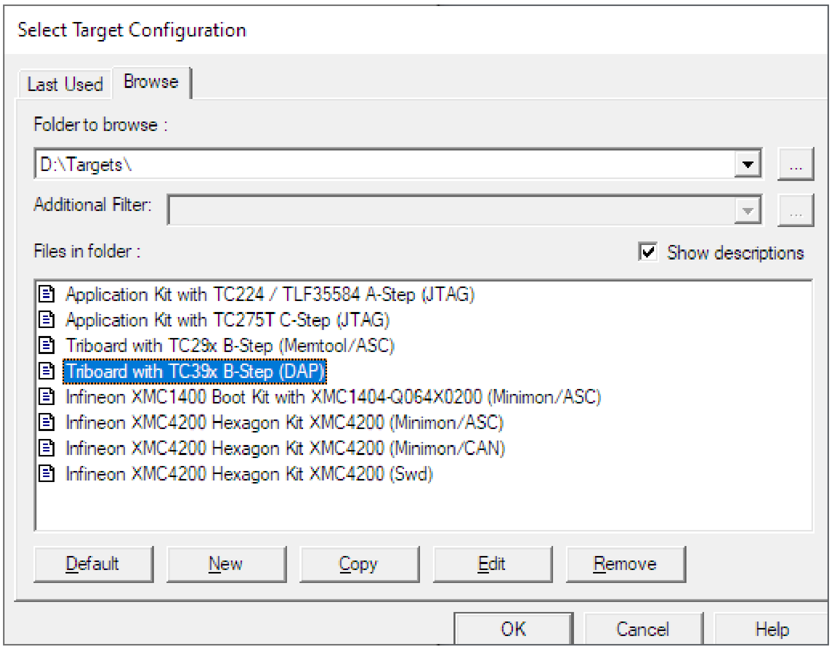
Preparing the Communication

To open a communication channel, some preparations are required.

The primary selection of the communication channel is done by choosing a suitable configuration, saved as `*.cfg`. You can select a new target configuration file while creating a new workspace or you can modify an existing configuration via the **Setup Target Config** button in the ‘connection failed dialog’ or the menu entry **Config – Target Configuration**.

Create a new configuration

While creating a new workspace you can choose a new configuration for your target hardware. The **Select Target Configuration** dialog will be shown.



Folder to browse : D:\Targets\

Additional Filter:

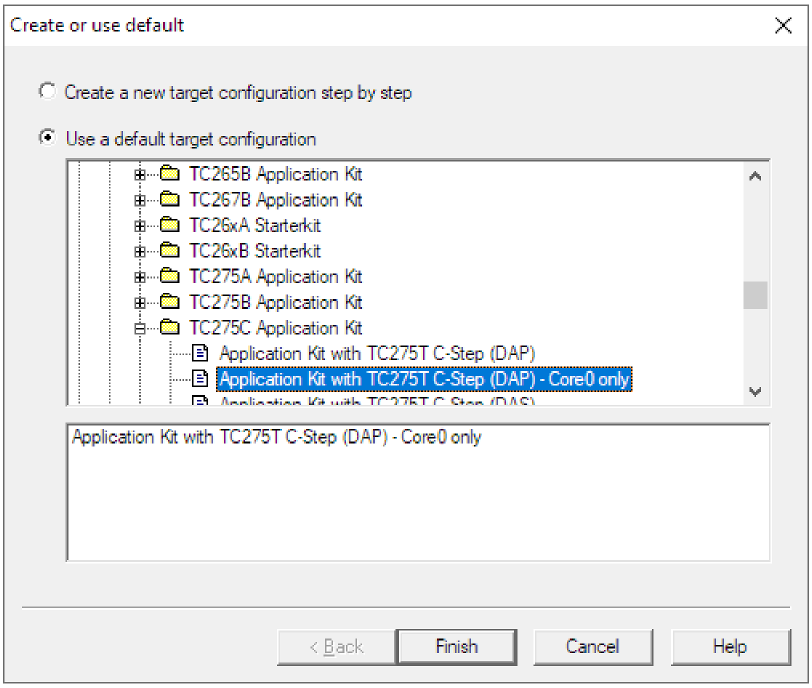
Files in folder :

☑ Show descriptions

- Application Kit with TC224 / TLF35584 A-Step (JTAG)
- Application Kit with TC275T C-Step (JTAG)
- Triboard with TC29x B-Step (Memtool/ASC)
- Triboard with TC39x B-Step (DAP)
- Infineon XMC1400 Boot Kit with XMC1404-Q064X0200 (Minimon/ASC)
- Infineon XMC4200 Hexagon Kit XMC4200 (Minimon/ASC)
- Infineon XMC4200 Hexagon Kit XMC4200 (Minimon/CAN)
- Infineon XMC4200 Hexagon Kit XMC4200 (Swd)

The Universal Debug Engine is equipped with some predefined configurations already. With these configurations, you have a short way to the first example, so you can derive a user-defined configuration from an existing Evaluation Board configuration file.

Push the button **Default** and a selection of predefined target configurations will be offered.



Create or use default

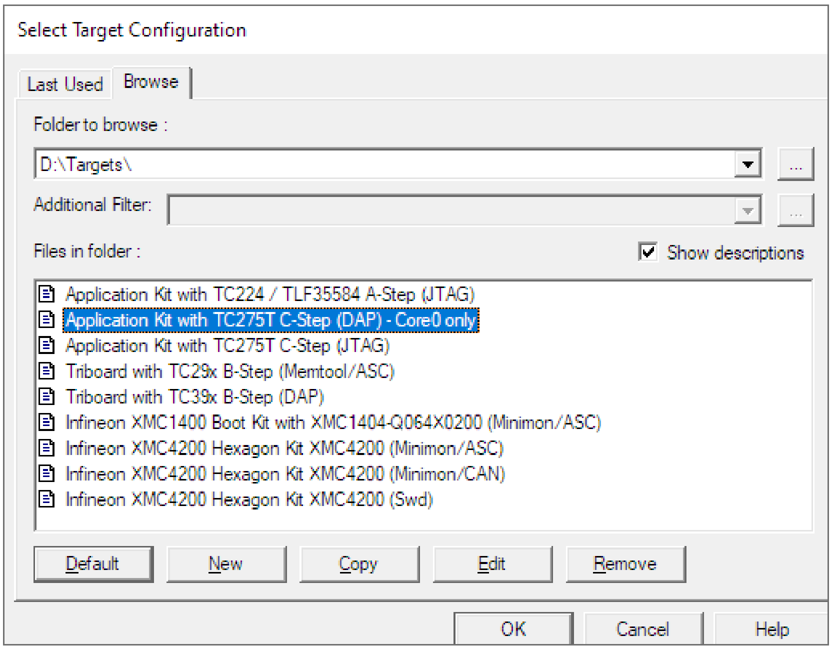
○ Create a new target configuration step by step

● Use a default target configuration

- TC265B Application Kit
- TC267B Application Kit
- TC26xA Starterkit
- TC26xB Starterkit
- TC275A Application Kit
- TC275B Application Kit
- TC275C Application Kit
  - Application Kit with TC275T C-Step (DAP)
  - Application Kit with TC275T C-Step (DAP) - Core0 only
  - Application Kit with TC275T C-Step (DAS)

Application Kit with TC275T C-Step (DAP) - Core0 only

Browse the best suitable configuration file, finish the dialog, save the configuration file.



Select Target Configuration

Last Used | Browse

Folder to browse : D:\Targets\

Additional Filter:

Files in folder :

☑ Show descriptions

- Application Kit with TC224 / TLF35584 A-Step (JTAG)
- Application Kit with TC275T C-Step (DAP) - Core0 only
- Application Kit with TC275T C-Step (JTAG)
- Triboard with TC29x B-Step (Memtool/ASC)
- Triboard with TC39x B-Step (DAP)
- Infineon XMC1400 Boot Kit with XMC1404-Q064X0200 (Minimon/ASC)
- Infineon XMC4200 Hexagon Kit XMC4200 (Minimon/ASC)
- Infineon XMC4200 Hexagon Kit XMC4200 (Minimon/CAN)
- Infineon XMC4200 Hexagon Kit XMC4200 (Swd)

Now it is possible to connect to the target system via button **OK** directly or to edit the target configuration via button **Edit** before. After that, connect to the target system.

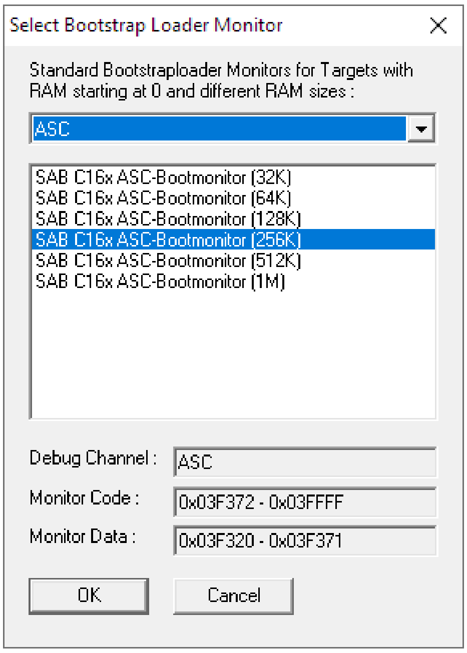
Later you can edit the target configuration via menu entry **Config – Target Configuration**.

Select a RAM based monitor program

The Universal Debug Engine provides a set of target monitors as binary images. The user may adapt the monitors to the demands of the target system.

The monitor will be downloaded via the on-chip bootstrap loading or a specialized on-chip debug module to the target system RAM before every debug session.

Select an ASC monitor program



Select Bootstrap Loader Monitor

Standard Bootstraploader Monitors for Targets with RAM starting at 0 and different RAM sizes :

ASC

SAB C16x ASC-Bootmonitor (32K)
SAB C16x ASC-Bootmonitor (64K)
SAB C16x ASC-Bootmonitor (128K)
SAB C16x ASC-Bootmonitor (256K)
SAB C16x ASC-Bootmonitor (512K)
SAB C16x ASC-Bootmonitor (1M)

Debug Channel : ASC
Monitor Code : 0x03F372 - 0x03FFFF
Monitor Data : 0x03F320 - 0x03F371

Several predefined ASC monitors are available as standard monitor in the internal monitor database. The monitors differ regarding the location of the monitor code and data area. To select the monitor program, use the Target Interface Setup dialog, page **Monitor** and check **Download Bootstrap Loader to Target** and **Use Standard Monitor**. With the **...** button, you can browse existing monitor configurations from the database.

The monitor code will be downloaded via bootstrap loading sequence. The properties, like baud rate and TTL-RS232 driver usage, will be taken from the bootstrap loading settings. To change this properties, use the page **Boot** and **Init**.

Custom defined monitors are possible with the help of the Monitor Development Tool. Use such monitors as external monitor in the page **Monitor**. Note, that the correct type of monitor must be selected as **Debug Communication Channel**.

Select a CAN monitor program

Select the CAN monitor as **Debug Communication Channel**. Furthermore, you have to configure the correct baud rate and CAN-ID. Additionally, make sure, that the bootstrap loading and initialization settings are correct in the page **Boot** and **Init**. Additional, ensure the bootstrap loading and initialization settings are correct in the page **Boot** and **Init**.

Select a ROM based monitor

ROM monitors are located in the target's non-volatile memory (FLASH or EPROM) and will be started by power ON of the target board.

The build of a ROM monitor is supported by the Monitor Development Toolkit, offered by the PLS. This toolkit provides the source code of our monitors prepared for the supported compilers. On this way, the user can build its "own" monitor program.

Connect the Target system

After the preparations of the communication channel, you can connect the debugger with your target hardware. Make sure, that the hardware is reset and correctly initialized. Push the button **Single retry** and the debugger will connect the target via the new communication channel.

After the successful connection, the following message is shown:

```
Success Core0::UDEDebugServer, Connection to TC22x target established:
TriCore (Core0), ID: 10201083h
```

Multi-Target Debugging

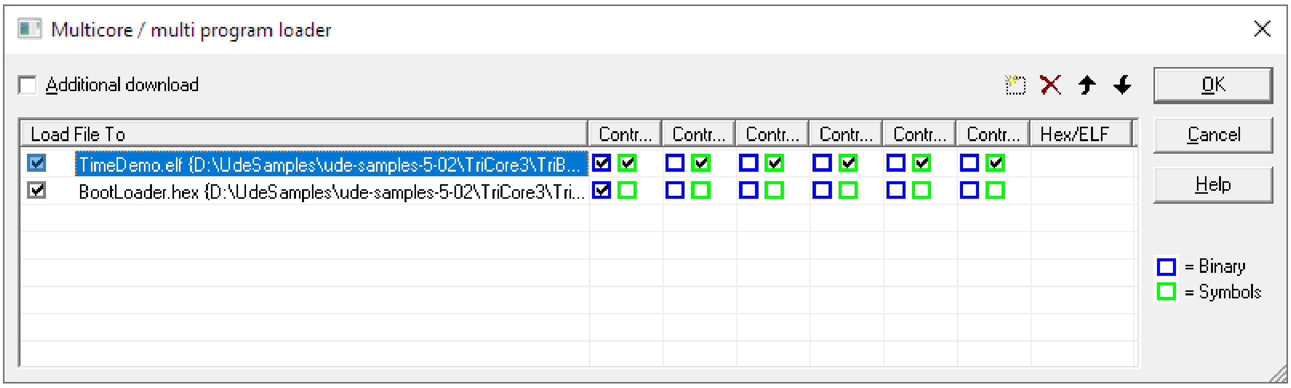
UDE supports multiple targets/controllers. It is possible to debug the controllers independently in one debugger instance. With the **UAD3+ Multi AURIX Debug Adapter** the controllers can be synchronized. It is then possible to use the multicore run control and break, single step or restart both controllers at almost the same time. This functionality is achieved by using additional trigger pins from AURIX TC3xx controllers, which are connected to the Multi AURIX Debug Adapter.

Please ask the PLS Support Team at support@pls-mc.com for detailed information and further hints about Multi-Target Debugging.

Downloading a binary File

Download a multi-core and multi-program Application

To download an application select **Load Program** from menu **File**. In the following multi-core / multi-program loader selection box, you can browse an elf/hex file and open it.



☐ Additional download

Load File To	Contr...	Contr...	Contr...	Contr...	Contr...	Contr...	Hex/ELF
☑ TimeDemo.elf {D:\UdeSamples\ude-samples-5-02\TriCore3\TriB...	☑ ☑	☐ ☑	☐ ☑	☐ ☑	☐ ☑	☐ ☑	
☑ BootLoader.hex {D:\UdeSamples\ude-samples-5-02\TriCore3\Tri...	☑ ☐	☐ ☐	☐ ☐	☐ ☐	☐ ☐	☐ ☐	

UDE supports the loading of multiple binaries/symbols into one or more cores of the target microcontroller. Via ⬚ and ✕ button further elf/hex files can be added or removed. For every loaded elf/hex file, it is possible to select the destination of binaries and symbols.

Binary

The **Binary** is used primary for FLASH programming. The microcontroller architecture decides, whether and which binary has to transfer in which core. Select the right assignment. It is usually, that the binary has to load to one core only, because the selected core can program the binaries into the FLASH memories of all cores.

Symbols

The **Symbols** are defined inside of the elf file are required to solved references in code by the debugger. That is why the symbols of an elf file must be assigned to the core, which contains these references.

Hex/ELF

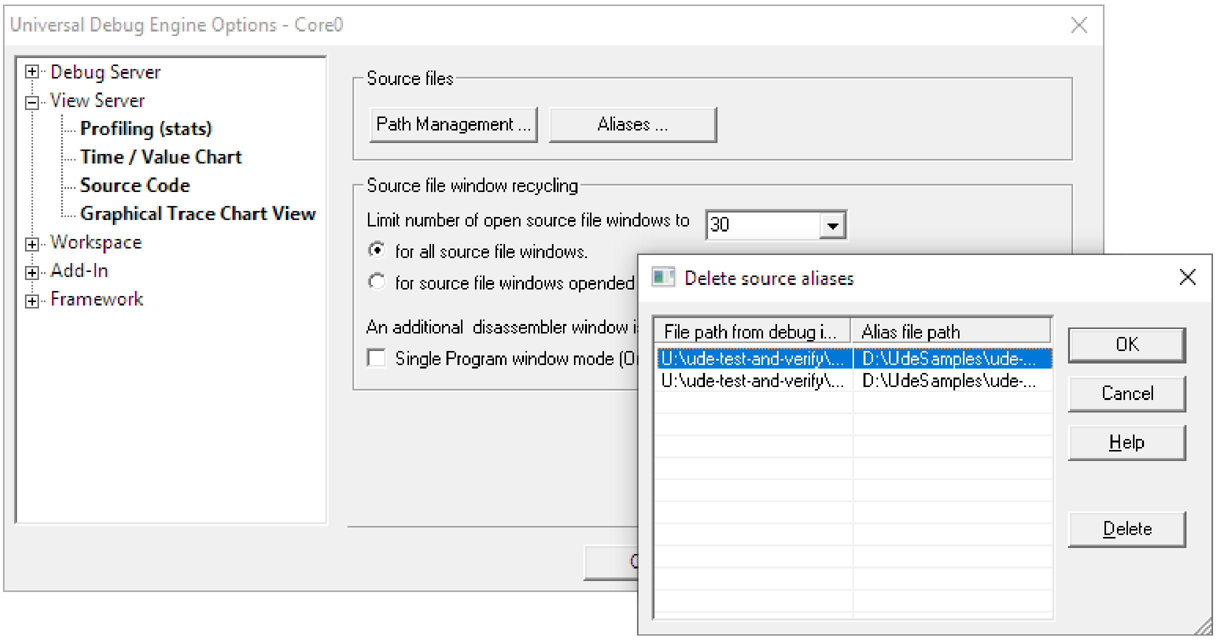
The **Hex/ELF** option is used when a hex file is selected for download, which does not contain any symbol information. In this case, a corresponding elf file with the symbol information is searched and its symbols are included automatically. If the elf file is not found, the user will be asking for it.

After that, the program can be downloaded into the target and the debugger is ready. The IP is set to the default entry point of your program, usually the start-up code. The start address also depends on the used target architecture. The debugger observes the time stamp of the application file(s). If a binary target file is changed, e.g. by rebuilding after it was loaded the debugger offers the possibility to reload the file.

Source File Management

If the debugger cannot find the source files specified in the symbol containing files, it can request the location of the correct source path. This can happen, if the compilers-built environment differs from the environment used by the debugging process. You need to provide the location on request. UDE will remember your decision and will not ask again for this source path.

Every entry of this relocated source is called **Alias**. The saved aliases of a workspace are accessible via the menu **Config – Debug Server Configuration – View Server – Source Code – Aliases …**



Limit number of open source file windows to 30

● for all source file windows.

○ for source file windows opended

An additional disassembler window i

□ Single Program window mode (O

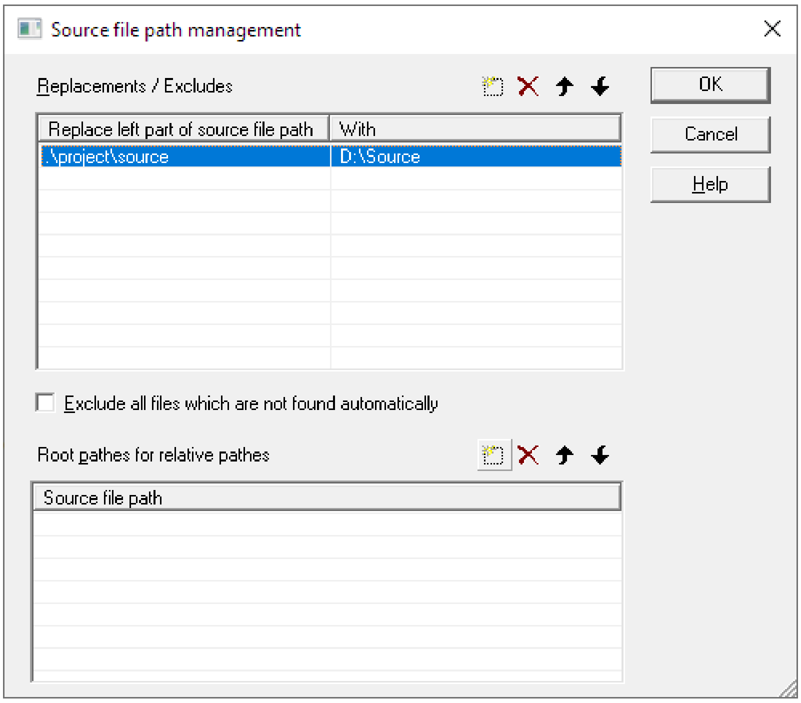
File path from debug i...	Alias file path
U:\ude-test-and-verify\...	D:\UdeSamples\ude-...
U:\ude-test-and-verify\...	D:\UdeSamples\ude-...

In some cases, the forced source alias may have changed. Use the dialog to delete the affected entry. The next time the source file is used, UDE will ask for the location again.

Source Path Replacement

The behavior described above is useful, if only few source file locations have changed. If a complete source file tree is affected, you can give UDE a replacement path, which will be applied to all source file paths.

Use the menu **Config – Debug Server Configuration – View Server – Source Code** – **Path Management …**



Replace left part of source file path	With
.\project\source	D:\Source

□ Exclude all files which are not found automatically

Source file path

The dialog defines a part for replacing and a part for including the new source location.

You will now see the source code of the main function from the sample application. When clicking with the right-hand mouse button into the Program window, a context button appears to switch between Source code only and Source/Assembly code display via the **Mixed Mode** entry.

Viewing Program Code

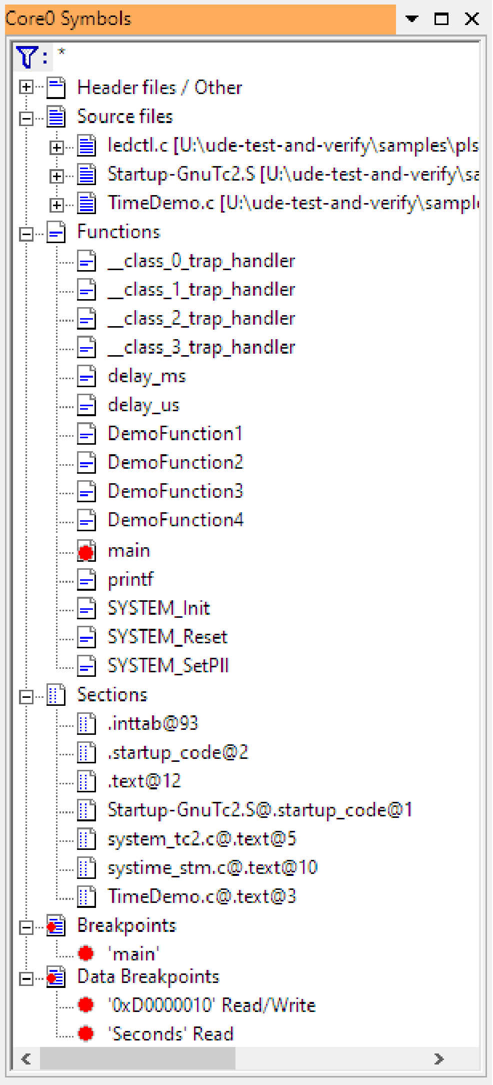
The Universal Debug Engine features the viewing of program code in C/C++ and Assembler level and provides the view of the program from three different perspectives. It shows the source files, the functions and the sections of the loaded program. The workspace window helps you to navigate into the parts of the program.

Note: UDE provides the view of the source code parts located in the program sections by default. For viewing of code outside of the program location, you may use the single program window view.

Workspace

The workspace window shows all program specific information and simplifies the navigation through the program. The project is displayed as tree control with source files, functions and sections. You can expand and collapse the folders by left clicking on the [+] and [-] Symbols.

If no workspace window is visible, open it via menu **Views – Target Manager**.



Header files / Other Source files

The folder **Header files / Other / Source files** lists all sources of the current project. With double-click you can open the source or bring the source window to the top.

Functions

In the folder **Functions** all C/C++ functions of the loaded program are displayed. By double-clicking on an entry in the workspace window, the cursor is set to the corresponding line in the source code windows.

It is possible to set single and multicore breakpoints at the function's entry point via context menu.

Sections

Sections are consecutive memory areas of code or data. In the folder **Sections**, all allocated memory areas are listed, also the non-C/C++ parts of the program as the startup code, vector table and so on.

Breakpoints

Breakpoints are listed as reference to a source location. The context menu allows the editing, disabling and removing of each breakpoint as well of all breakpoints.

Data Breakpoints

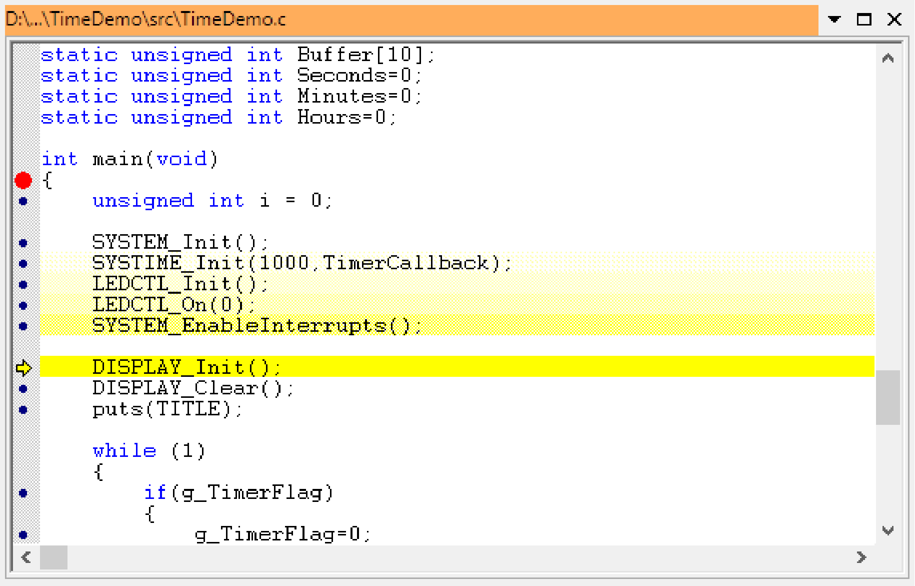
Data breakpoints are listed as reference to a data location. The context menu allows the editing, disabling and removing of each breakpoint as well of all breakpoints.

Source Code Window

In the **Source Code** window, the source code is displayed. The Universal Debug Engine supports a C/C++/Assembler source-oriented view mode and a disassembly view mode. Both display modes can be merged as **Mixed Mode**. If no C/C++/Assembler code is available, UDE shows disassembled instructions per default.

To switch between the view modes C/C++ and C/C++/Assembler, use the context menu entry **Mixed Mode**.

C/C++ oriented view mode



D:\...\TimeDemo\src\TimeDemo.c

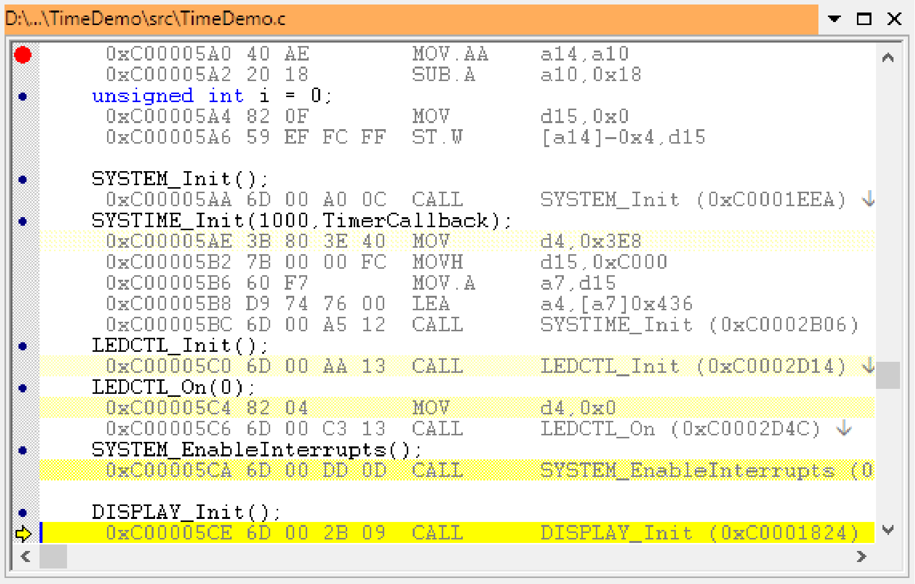
```
static unsigned int Buffer[10];
static unsigned int Seconds=0;
static unsigned int Minutes=0;
static unsigned int Hours=0;

int main(void)
{
    unsigned int i = 0;

    SYSTEM_Init();
    SYSTIME_Init(1000,TimerCallback);
    LEDCTL_Init();
    LEDCTL_On(0);
    SYSTEM_EnableInterrupts();

    DISPLAY_Init();
    DISPLAY_Clear();
    puts(TITLE);

    while (1)
    {
        if(g_TimerFlag)
        {
            g_TimerFlag=0;
```

The yellow pointer indicates the current location of the IP. After each step, the position is updated. The solid red circle shows that a breakpoint is located at this position. You can only set breakpoints to C/C++ source line indicated with a small blue dot or to assembler lines. The small blue dot indicates that the compiler has generated machine code for this C/C++ source line.

The Source Code window supports an IP history, and displays the last five IP entries as highlighted lines in customizable color shades. Use the Refresh/Properties entry from the context menu, to customize this and other colors.

C/C++/Assembler mixed mode

The picture below shows the example from above in **Mixed Mode**. Each C/C++ source line is shown with the corresponding machine code lines. The breakpoint indication is set to the real address location in the machine code.

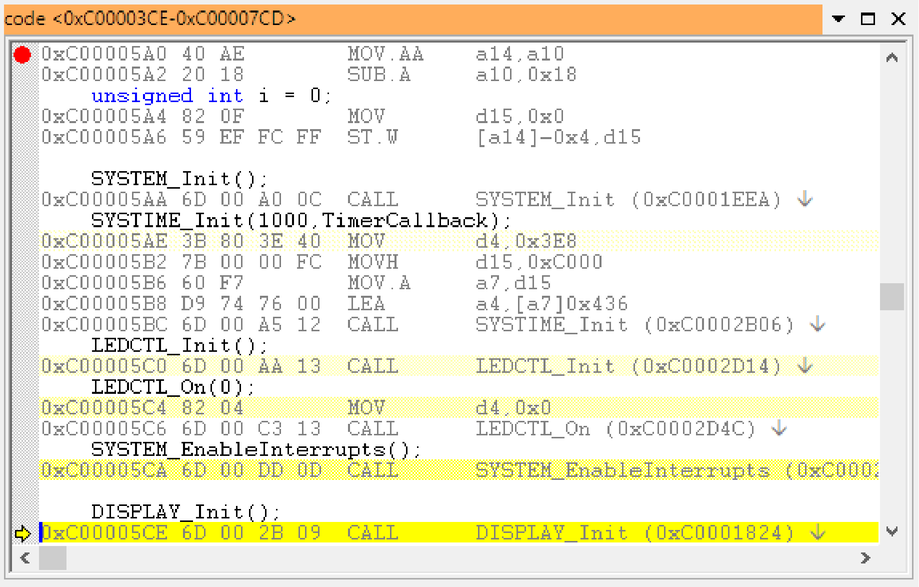


D:\...\TimeDemo\src\TimeDemo.c

```
     0xC00005A0 40 AE        MOV.AA    a14,a10
     0xC00005A2 20 18        SUB.A     a10,0x18
    unsigned int i = 0;
     0xC00005A4 82 0F        MOV       d15,0x0
     0xC00005A6 59 EF FC FF  ST.W      [a14]-0x4,d15

    SYSTEM_Init();
     0xC00005AA 6D 00 A0 0C  CALL      SYSTEM_Init (0xC0001EEA) ↓
    SYSTIME_Init(1000,TimerCallback);
     0xC00005AE 3B 80 3E 40  MOV       d4,0x3E8
     0xC00005B2 7B 00 00 FC  MOVH      d15,0xC000
     0xC00005B6 60 F7        MOV.A     a7,d15
     0xC00005B8 D9 74 76 00  LEA       a4,[a7]0x436
     0xC00005BC 6D 00 A5 12  CALL      SYSTIME_Init (0xC0002B06)
    LEDCTL_Init();
     0xC00005C0 6D 00 AA 13  CALL      LEDCTL_Init (0xC0002D14) ↓
    LEDCTL_On(0);
     0xC00005C4 82 04        MOV       d4,0x0
     0xC00005C6 6D 00 C3 13  CALL      LEDCTL_On (0xC0002D4C) ↓
    SYSTEM_EnableInterrupts();
     0xC00005CA 6D 00 DD 0D  CALL      SYSTEM_EnableInterrupts (0

    DISPLAY_Init();
     0xC00005CE 6D 00 2B 09  CALL      DISPLAY_Init (0xC0001824)
```

Disassembly (complete range) window mode

In **Disassembly (complete range) window** mode, a single **Program** window displays the target memory content as machine code with symbolical information (code labels, C/C++ source lines, etc.), if available. If a source file was not found by the debugger automatically, the file name and source line number will be displayed. In this case, a mouse double click on the source line opens a file dialog to browse in the file.

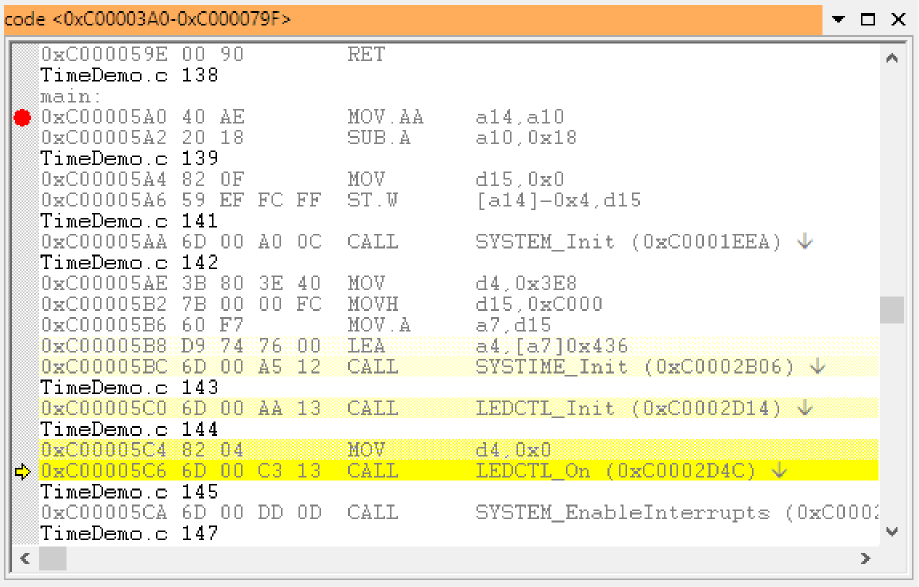


```
code <0xC00003CE-0xC00007CD>
0xC00005A0 40 AE       MOV.AA    a14,a10
0xC00005A2 20 18       SUB.A     a10,0x18
    unsigned int i = 0;
0xC00005A4 82 0F       MOV       d15,0x0
0xC00005A6 59 EF FC FF ST.W      [a14]-0x4,d15

    SYSTEM_Init();
0xC00005AA 6D 00 A0 0C CALL      SYSTEM_Init (0xC0001EEA)
    SYSTIME_Init(1000,TimerCallback);
0xC00005AE 3B 80 3E 40 MOV       d4,0x3E8
0xC00005B2 7B 00 00 FC MOVH      d15,0xC000
0xC00005B6 60 F7       MOV.A     a7,d15
0xC00005B8 D9 74 76 00 LEA       a4,[a7]0x436
0xC00005BC 6D 00 A5 12 CALL      SYSTIME_Init (0xC0002B06)
    LEDCTL_Init();
0xC00005C0 6D 00 AA 13 CALL      LEDCTL_Init (0xC0002D14)
    LEDCTL_On(0);
0xC00005C4 82 04       MOV       d4,0x0
0xC00005C6 6D 00 C3 13 CALL      LEDCTL_On (0xC0002D4C)
    SYSTEM_EnableInterrupts();
0xC00005CA 6D 00 DD 0D CALL      SYSTEM_EnableInterrupts (0xC000

    DISPLAY_Init();
0xC00005CE 6D 00 2B 09 CALL      DISPLAY_Init (0xC0001824)
```

The **Single Program Window** mode can be switched on/off in menu **Views – Disassembly (complete range)** and/or with the corresponding tool bar button.

Disassembly mode

The **Disassembly** view mode is used only, if no source is available.



```
code <0xC00003A0-0xC000079F>
0xC000059E 00 90       RET
TimeDemo.c 138
main:
0xC00005A0 40 AE       MOV.AA    a14,a10
0xC00005A2 20 18       SUB.A     a10,0x18
TimeDemo.c 139
0xC00005A4 82 0F       MOV       d15,0x0
0xC00005A6 59 EF FC FF ST.W      [a14]-0x4,d15
TimeDemo.c 141
0xC00005AA 6D 00 A0 0C CALL      SYSTEM_Init (0xC0001EEA)
TimeDemo.c 142
0xC00005AE 3B 80 3E 40 MOV       d4,0x3E8
0xC00005B2 7B 00 00 FC MOVH      d15,0xC000
0xC00005B6 60 F7       MOV.A     a7,d15
0xC00005B8 D9 74 76 00 LEA       a4,[a7]0x436
0xC00005BC 6D 00 A5 12 CALL      SYSTIME_Init (0xC0002B06)
TimeDemo.c 143
0xC00005C0 6D 00 AA 13 CALL      LEDCTL_Init (0xC0002D14)
TimeDemo.c 144
0xC00005C4 82 04       MOV       d4,0x0
0xC00005C6 6D 00 C3 13 CALL      LEDCTL_On (0xC0002D4C)
TimeDemo.c 145
0xC00005CA 6D 00 DD 0D CALL      SYSTEM_EnableInterrupts (0xC000
TimeDemo.c 147
```

Printing of program code

Via menu **File – Print** or the shortcut **Ctrl+P** the content of the **Program** window can be printed via the system printers. It is possible to print all pages, a set of pages or selections. All view modes are supported.

Running a program

After successfully downloading the program to the target, the program can be started. The IP is set automatically to the first instruction of the code. For example, in the C166 architecture, the address 0x0000 is used. If available, the programs entry point can also be read from ELF file or target register and/or memory locations.

To start your program, use the menu **Debug – Start Program Execution**, or the **F5** key.

Your program will stop under following conditions:

1. A breakpoint is reached.

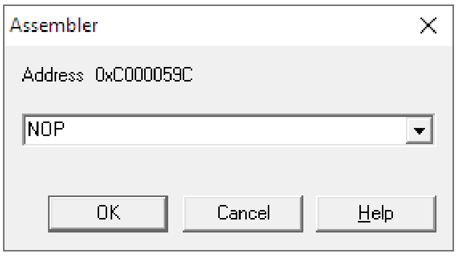
2. A manual user break is made via menu **Debug – Break Program Execution**.

Note: Usually the compiler implements code for the case that the program returns from the main function. Often it is implemented as an infinite loop, so that the program will continue running after the main's return. A solution for that problem is using a breakpoint at the last return instruction.

Inline Assembler

The integrated inline assembler allows the change of several machine instructions. It is available in program windows in mixed mode or disassembly mode only.

To input a new machine instruction, set the mouse cursor to the assembler line and use the context menu. The entry **Integrated Assembler** allows to start the Inline Assembler. In the following dialog box, you may input the new machine instruction. Push the button **OK** and the new instruction will be written into the target's memory.


Assembler

Address 0xC000059C

NOP

Attention: If the new instruction does not fit in the alignment and length of the original instruction, a further instruction may be inserted with non-valid code. Always verify the inserted instructions carefully!

Viewing and Modifying of Core Registers

The Universal Debug Engine enables you to display and modify the complete register set of the current microcontroller derivative in different ways. In addition to the predefined Core Registers window and to the configurable Peripheral Registers window UDE offers the feature of a user-definable HTML based extension window.

Kinds of Core Register windows

Description of SFR, CSFR and GPR Registers

Core Special Function Registers (**CSFR**) control the operations of the microcontroller core and provide status information about core operation. The General-Purpose Registers (**GPR**) complete the CSFRs with a set of multifunction registers. All other peripheral registers of the used derivatives (except the CSFRs and GPRs as stated) are denoted as Special Function Registers (**SFR**).

Core Registers window

The **Core Registers** window displays a selection of CSFR and GPR registers of the target controller. It allows a fast access to the PSW, IP, Address and Data Registers. It is not possible to configure the view of the register set for the Core Registers window.

Peripheral Registers window

The **Peripheral Registers** window allows a more flexible view to the registers. You can select any free collection of CSFR, SFR and GPR registers to be displayed in the Peripheral Registers window. This way, a simple custom specific view is possible.

HTML window

Maximum customization is offered by the **HTML Document** window. It allows creating customer-specific HTML documents, embedding UDE ActiveX-controls, user components and scripts with access to the UDE Object Model. This allows you to visualize and control aspects of your target system in a more understandable and attractive way.

Core Registers window

To open the Core Registers window, use the menu **Views – Core Registers**. A predefined selection of CSFR and GPR registers of the target controller will be shown.

Changing the Core Register content

To modify the value of a register in hexadecimal mode move the mouse cursor over the target register content and double-click. An edit box with the content will open and allow you to modify the register value. To enter the value, push the **<RETURN>** key or click with the mouse outside of the register box.

Color coding

The register values appear in three colors indicating the current state of the value.

Register Value Color	State of the Register Value
Red	Value has been modified by the program during the last step
Blue	Value was changed by the user but not yet written into the target
Black	Value was not changed

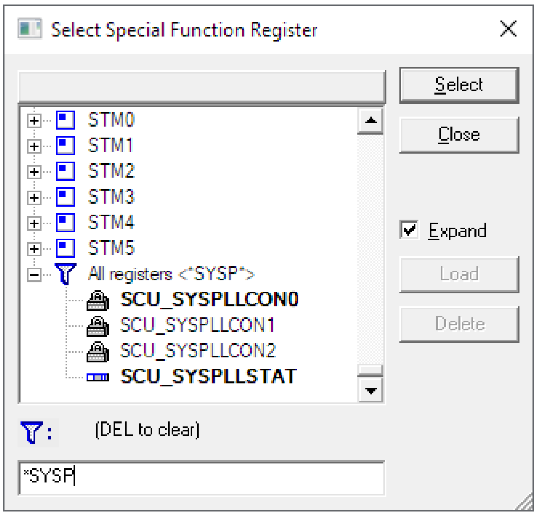
Note: For writing the user-changed value into the target, click outside the edit box, but inside the Core Registers window or push the **<RETURN>** key. Otherwise, the value will not be written into the target.

Peripheral Registers View

Creating or changing a Peripheral Registers window

In the **Peripheral Registers window** all registers of the CSFR-, GPR- and SFR-sets of the current microcontroller can be displayed and modified. To open the window, select **Views – Peripheral Registers** from the menu.

For selecting or deselecting of registers, simply use the context menu via a right-click on the Peripheral Registers window. Now click on the **Browse** menu item to open the Register Selection dialog window. All available registers of the current microcontroller will be displayed in that window. They are assorted according to the peripherals they belong.



To select a register, expand the list and scroll to the special entry for that register. Click on **Select** or simply double-click the entry to insert it into the Peripheral Registers window. A ToolTip will give you a short description of every register in the list.

If the **Expand** checkbox is activated, the new register is automatically expanded in the Peripheral Registers window. If a register is already in the Peripheral Registers window, it is marked bold.

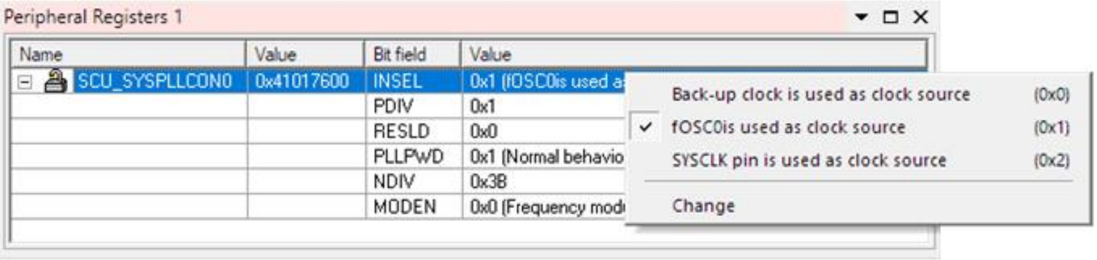
To remove a register from the Peripheral Registers window, move the mouse over the register and open the context menu via right click. Then select **Delete** to remove the entries in the Peripheral Registers window.

Changing the layout

The Peripheral Registers window displays the registers and the values in a table format. Two columns on the left, display the name and the current value of the complete register. If you move the mouse over the name, a short tool tip will show you the address, the reset value and a short description of the register.

If you expand a register, the fields and bits, it is composed of, will be shown in columns right of the name and current value of the register. The names of the fields correspond to the names in the controller’s manual. The current value of each field is shown in a hexadecimal value and an interpretation of the setting.

The following figure displays a Peripheral Register window with an example register added to the window. The `SCU_SYSPLLCON0` register is expanded and you can see all the bit fields and the interpretation of the current settings.



Name	Value	Bit field	Value
SCU_SYSPLLCON0	0x41017600	INSEL	0x1 (fOSC0is used a
		PDIV	0x1
		RESLD	0x0
		PLLPWD	0x1 (Normal behavio
		NDIV	0x3B
		MODEN	0x0 (Frequency mod

Back-up clock is used as clock source (0x0)
✓ fOSC0is used as clock source (0x1)
SYSCLK pin is used as clock source (0x2)
Change

It is possible to display more than one-bit field per row. This is useful if the register has a great number of fields (e.g. port registers). You can select 1, 2, 3 or 4 fields per row. To change the layout, you can use the context menu and go to **Layout.** In the submenu, select one of the alternatives. You can also change the order inside the Peripheral Registers window. Move the register per Drag ’n Drop from one location to that place that you want.

The layout of the Peripheral Registers window content can be exported and imported via **Content** from the context menu.

Changing the register content

To change the register content, you have two options. The first one is to change the complete value for a register. To do this, select the register (complete row will be marked blue) and move the mouse cursor over the register value (second column). After that open the context menu via right click. Select **Change** will prompt a cursor inside the value. Now you can modify the data. Press return to accept the new data and to write it into the target. As a shortcut, press **F2** to enter the change mode.

The second option is to change only a single component of the register. Expand the register to display all fields. Then select the field that you want to change, move the mouse cursor over the value and open the context menu. Now you can select one possible setting for this special field from the menu or select **Change** to enter a hexadecimal value manually. To enter the change mode, you can use the shortcut **F2**

Note: Some registers have a lock symbol left to their name (e.g. the BIV register in the figure above). These registers are EndInit protected. To change their values, you have to unlock them first with the entry **Write protect** from the context menu. If this option is disabled, the lock symbol is in an unlocked state, you can change the data and the Debugger performs a special access cycle to write the value to the target.

Saving and Restoring

If you want to save your current window configuration for later use or for other projects, you can use the **Content** option from the context menu. Select **Save** from the submenu opens a file dialog where you specify the filename. To load the Peripheral Registers window content again, select **Load** from the context submenu.

Color coding

The register values appear in two colors indicating the current state of the register value.

Register Value Color	State of the Register Value
Red	Value has been modified by the program during the last step
Black	Value was not changed

HTML View based on the UDE Object Model

The HTML view is suitable for visualizing and controlling of contexts of custom specific target hardware.

Note: All HTML windows are interactive only, if they are running in the UDE environment. That is why the HTML window of UDE must be used.

The online help system contains a detailed description of the UDE object model illustrated with many examples. The documentation can be found at

`<UDE_DIRECTORY>\Help\UdeObjectModel.chm`

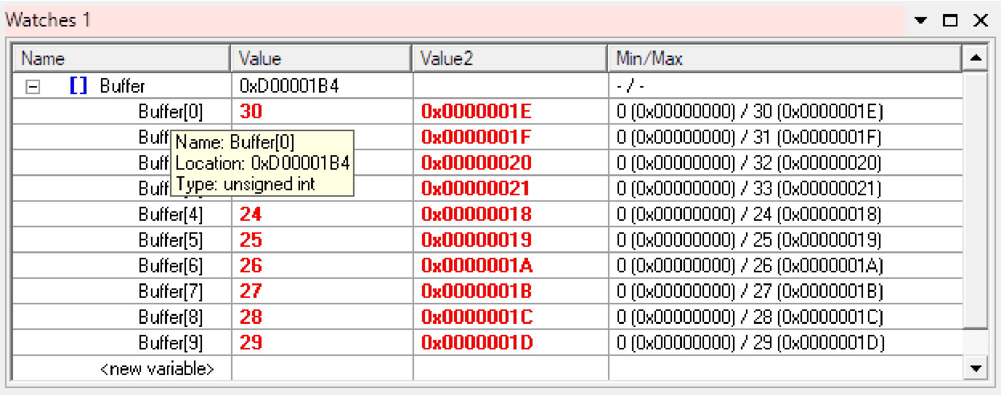
Watching Variables

The Universal Debug Engine features two kinds of windows for viewing the contents of C/C++ variables.

1. The content of the **Watches** window can be defined by the user. All types of variables (automatic, static and global) are possible.

2. The **Locals** window shows all-automatic and function/block local static variables that are valid in the current scope.

Watches



Name	Value	Value2	Min/Max
[] Buffer	0xD00001B4		- / -
Buffer[0]	30	0x0000001E	0 (0x00000000) / 30 (0x0000001E)
Buff		0x0000001F	0 (0x00000000) / 31 (0x0000001F)
Buff		0x00000020	0 (0x00000000) / 32 (0x00000020)
Buff		0x00000021	0 (0x00000000) / 33 (0x00000021)
Buffer[4]	24	0x00000018	0 (0x00000000) / 24 (0x00000018)
Buffer[5]	25	0x00000019	0 (0x00000000) / 25 (0x00000019)
Buffer[6]	26	0x0000001A	0 (0x00000000) / 26 (0x0000001A)
Buffer[7]	27	0x0000001B	0 (0x00000000) / 27 (0x0000001B)
Buffer[8]	28	0x0000001C	0 (0x00000000) / 28 (0x0000001C)
Buffer[9]	29	0x0000001D	0 (0x00000000) / 29 (0x0000001D)
<new variable>			

Name: Buffer[0]
Location: 0xD00001B4
Type: unsigned int

You may open the Watches window by clicking menu **Views – Watches**.

The Watches window content can be defined by the user. Adding of variables may be done via right clicking in the Name column and using the context menu entries **Browse** or **Expression**. Alternatively, push the **Ins** button for selecting a variable from the variable list or the **F2** button for editing the name of the variable.

Expandable variables, i.e. pointer and array, are shown with a [+] / [-] flag in front of the variable. Use this flag to expand/collapse the variable's elements. The view format of the variables value can be selected from decimal, hexadecimal and ASCII. Click on the Value column and select the format from the context menu. The window has optionally selectable columns for displaying the variable's physical address, a second display of the value (in a different format) and the minimum and maximum of the value seen.

Variable values can be easily changed by clicking in the value area and typing in the new value. Additionally, the context menu entry **Change** and the button **F2** are usable.

Note: The address in the first line of an expandable variable is the base address of the variable. In the example above, the `Buffer` variable is an array. The first line value entry `0xD00001B4` points to the base address of `Buffer` in the target's memory.

Watch Expressions

In the previous section, the variables have been selected by name from a list. This section describes how to define watch expressions, to manually select variables and/or parts of it. The expressions shown below can be entered in the Watches window into a line with `<new variable>` in it.

Global Variable description

Syntax

```
VariableName (global)
VariableName#
```

The second form is for compatibility reasons to fast-view66 only.

Example

```
g_nCount (global)
```

Module static variable description

Syntax

```
VariableName [source_file]
```

Example

```
nLastTime [time.c]
```

Function static variable description

Syntax

```
VariableName [source_file]:{function}
```

Element	Description	Example

`source_file`	Name of source file with extension and without path (A relative or absolute path from the command line of compiler is also possible but has to match exactly)	`..\time.c`
		`v:\time.c`

`function`	Function name	

Example

```
nLocalCount [time.c] : {timer_overflow}
```

Global, static variables and parts of it

Global and static program variables are variables with a fixed address, which means not stored in registers or on the stack. Such variables and parts of them can be described in common C-Syntax, similar to the expression in a C program.

Syntax

```
?(<C variable or part of it>)
```

Examples

```
?(abyBuffer[5])
?(timeStruct)
?(timeStruct.hour)
?(arrayOfTimeStructsInstance[4].hour)
```

Real expressions

Real expressions do not have a fixed address. These expressions consist of two or more global or static program variables and may contain C-operators or constants. They can be described in common C-Syntax, similar to expressions in a C program.

Syntax

```
?(<C expression>)
```

For compatibility reasons to former products, following inline function in expressions is supported

```
fconvert("<double_scale_factor>[,<double_offset>]",<C expression>)

<double_scale_factor> and <double-offset> are in %G syntax.
<double_offset> default value is 0.
<C expression> is integer expression.
```

This inline function will be converted after input to following expression

```
(<C expression>)*<double_scale_factor>-<double_offset>
```

Examples

```
?(timeStruct.hour*60)
?(timeStruct.hour*60+timeStruct.minutes)
?(arrayInstance[i])
?(arrayOfTimeStructsInstance[i+1].hour)
?(i+3)
```

Please note, that ' `i` ' can be a variable of global or static scope.

Real expressions with alias name

Real expressions with alias name refer to real expressions, as described before, with an additional alias string for displaying the value assigned.

Syntax

```
    ?("<printf format string one format specifier>", <C expression>)
```

Input Examples	Output Examples
`?("%d minutes left", timeStruct.hour*60+timeStruct.minutes)`	`73 minutes left`
`?("Temperature: %G °C",((a+b)*2)/30)`	`Temperature: 47.21 °C`

Description of watches content in a file

It is possible to describe the watches window content in a text file. The file can be loaded and saved via Load/Save in the context menu in the Watches window.

Syntax description

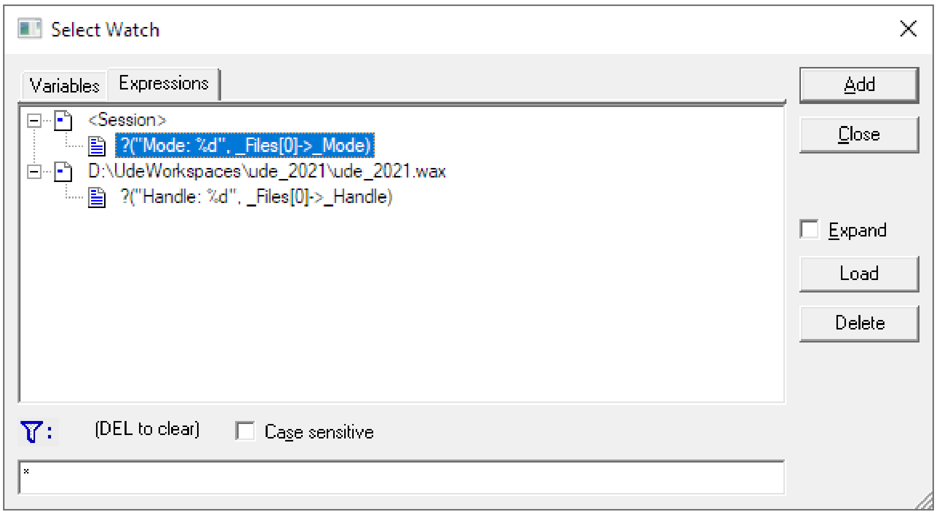
Every line contains one watch variable description or watch expression as described in the previous sections. Blank lines and comment lines are allowed. Comment lines start with a semicolon character (';') as the first non-whitespace-character of the line.

Adding Variables and Expressions using Select Watches Dialog

This sizeable dialog lets the user select static and global variables (all variables with fixed address) and complex expressions using these variables of loaded programs to the **Watches** window.

The dialog contains two tabs:

- **Variables Tab -** This dialog page displays global and static variables, which can be added to the current **Watches** window. Variables are sorted by scope: Global variables, Module static variables, Function static variables, All static variables and a list of all global and static variables. Within the scope, the entries are sorted in alphabetical order.
- **Expressions Tab -** The expression tab is the viewer for the global debugger **Expression Clipboard**. It shows user defined expressions from Watches and Graph window and expressions loaded via **Load** button from `*.wat` or `*.wax` files. An expression can added to the Watch or Graph window from here.



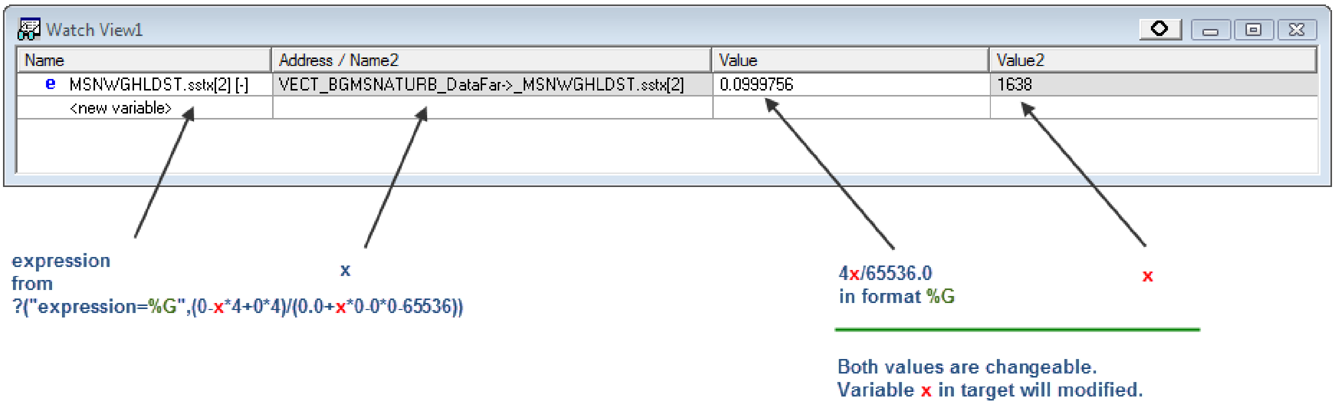
```
<Session>
    ?("Mode: %d", _Files[0]->_Mode)
D:\UdeWorkspaces\ude_2021\ude_2021.wax
    ?("Handle: %d", _Files[0]->_Handle)
```

If expressions contain only one integer variable, this variable is extracted automatically and can be selected separately.

This expression itself enables an additional display feature, which is called ***Advanced Expression Resolution***:

This means the parallel display of multiple values for the expression, which contains only one integer variable. This allows displaying of the calculated value of the expression and of the corresponding integer variable. To use the **Advanced Expression Resolution** enable it in the **Properties – View Servers – Watch&Locals Windows – Configuration** of the Watch window. Following additional implicit functions are available for such expression in the **Watches** window:

- Formula simplification
- Recognition of such formula type
- Automatic extraction of integer variable
- Modification of expression and variable
- Recalculation and changing of integer variable in target, if expression value is changed



Name	Address / Name2	Value	Value2
e MSNWGHLDST.sstx[2] [-]	VECT_BGMSNATURB_DataFar->_MSNWGHLDST.sstx[2]	0.0999756	1638
<new variable>			

expression
from
?("expression=%G",(0-x*4+0*4)/(0.0+x*0-0*0-65536))

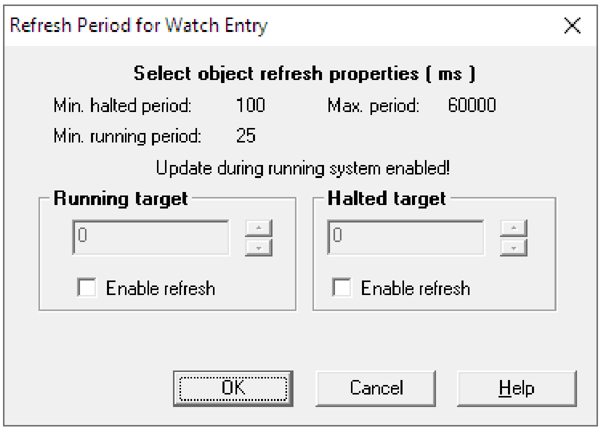
Both values are changeable.
Variable x in target will modified.

Locals

Open the **Locals** window by clicking **Views – Locals.** The **Locals** window displays all automatic and function/block local static variables defined by C/C++ scope rules. This view is not configurable. The variable values will be refreshed automatically after a debugging step. The display of variables is comparable to the Watches windows and modifying the values, will operate in the same way.

Automatic variable content refresh

UDE can refresh the variable content during all states of the target, assuming the target is in a connected state. Via the context menu, the automatic refresh period can be configured. A manual refresh can be issued with the **F11** key or the corresponding context menu entry.



Select object refresh properties (ms)

Min. halted period: 100 Max. period: 60000

Min. running period: 25

Update during running system enabled!

Running target

0

☐ Enable refresh

Halted target

0

☐ Enable refresh

Stepping and Breakpoints

Overview

In general, breakpoints are used to stop a running target system at a user defined program state. Breakpoints help you to follow the program flow and allow you to see the current program and processor status. In short, they are helpful to observe the behavior of the program.

With the support of the OCDS modules of C166CBC, AURIX, TriCore, UDE offers some new features of debugging. The OCDS L1 module can be programmed as a user defined trigger, supports hardware breakpoints, stops the microcontroller when a target read/write access occurs and stops in dependency of the ALU result.

Following the program flow

The Universal Debug Engine supports the stepping through the program in a very comfortable way.

Many features are available for controlling the program flow. You may use the menu **Debug** or the debug tool bar for selecting the matching feature.

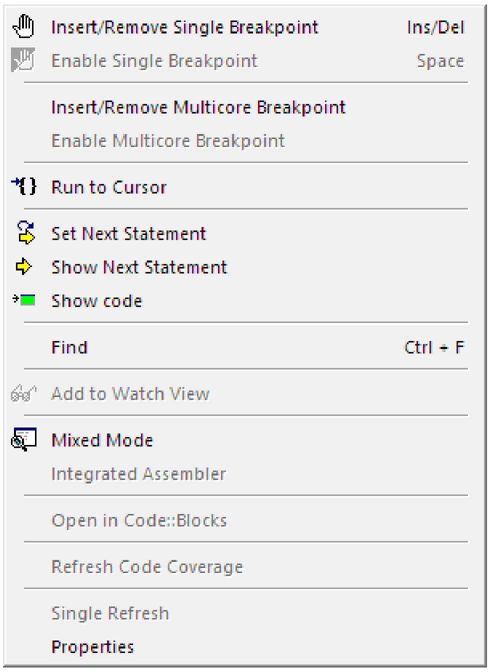
Step into Subroutine F8
Step into the Subroutine means that all code lines are executed step by step. The debugger will follow the control flow by stepping into subroutine calls.

Step over Subroutine F9
Step over the Subroutine means that all code lines are executed step by step, but the debugger will not be step into subroutine calls, but to the line after the subroutine call. This means, the called subroutines are executed, until they return to the currently executed subroutine

Step out Subroutine
If the IP is located within a subroutine, the Step out command jumps to the calling procedure of the subroutine.

Run Program to Cursor F4
The program is started and runs until the IP reaches the location defined by the current cursor location.

Please bear in mind, if the control flow encounters a breakpoint during a run to cursor, step over or step out, the target will still be stopped.



Stop the program at a specified location

The Universal Debug Engine allows setting breakpoints in the program manually. These breakpoints stop the execution of the program, if the microcontroller reaches this location. Breakpoints are either absolute or conditional. Additionally, data breakpoints are supported by some targets.

The breakpoint dialog is reachable via menu **Views – Breakpoints. It** gives you an overview about all breakpoints and allows modification of locations and break conditions.

Absolute Breakpoints

An absolute breakpoint stops the program at a specified target location without any conditions. Breakpoints can be set via the context menu of the program window. Set the cursor to the program line and open the context menu by a right click. The entry **Insert/Remove Breakpoint** sets or clears a breakpoint at the specified program line. A set, enabled breakpoint is displayed as solid red dot in the symbol column left to the program line. The entries **Enable Breakpoint/Disable Breakpoint** enable or disable the specified Breakpoint. Disabled breakpoints are shown as red circles.

Conditional Breakpoints

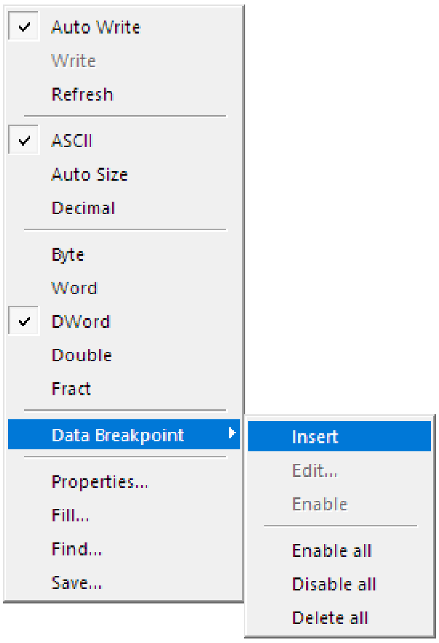
Conditional breakpoints are used to halt a program based on a predefined condition. To configure breakpoint conditions, use the breakpoint dialog, reachable via menu **Views – Breakpoints ...**

Data Breakpoints

UDE supports the usage of data breakpoints, if on-chip debug support on target MCU is available (e.g. OCDS). This means that the program will stop, if a read or write access to a memory location was executed. Data breakpoints can be set in **Watches** and in **Memory** window.

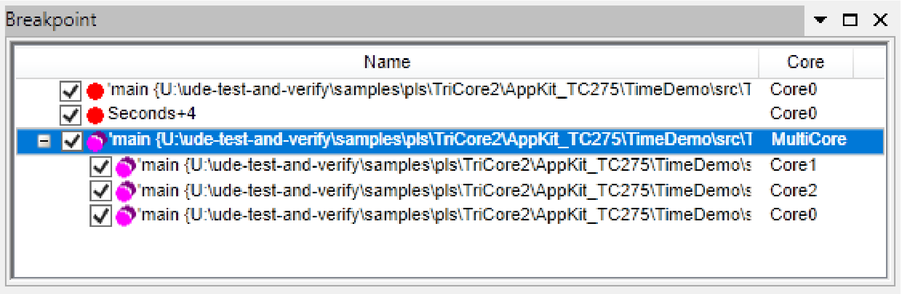
The context menu of a **Watches** and **Memory** window contains an entry for setting data breakpoints. This option is available only, if the connected target supports data breakpoints. It can be used at one or more selected main and child entries in all columns except for the `<new variable entry>`. The memory range for the data breakpoint consists of the start address and the length of the selected entry. If the base entry of a complex data type (e.g. structure or array) is selected, the data breakpoint will have the range of the complete entry. The breakpoint can be set on Write, Read or Read/Write accesses. All data breakpoints can be enabled, disabled or cleared together.

Data breakpoints can be used in the Memory window in the same way, the same context menu entries are available. These breakpoints are shown with red marks, analog to code breakpoints.



Breakpoints window

To modify code and data breakpoints open the **Breakpoints** window via **Views – Breakpoints**.



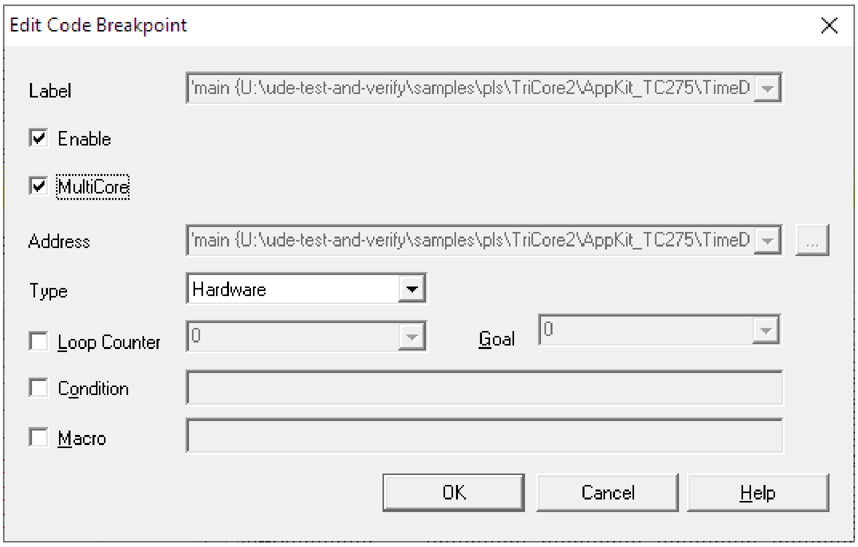
Name	Core
☑ ● 'main {U:\ude-test-and-verify\samples\pls\TriCore2\AppKit_TC275\TimeDemo\src\T	Core0
☑ ● Seconds+4	Core0
⊟ ☑ 'main {U:\ude-test-and-verify\samples\pls\TriCore2\AppKit_TC275\TimeDemo\src\T	MultiCore
☑ 'main {U:\ude-test-and-verify\samples\pls\TriCore2\AppKit_TC275\TimeDemo\s	Core1
☑ 'main {U:\ude-test-and-verify\samples\pls\TriCore2\AppKit_TC275\TimeDemo\s	Core2
☑ 'main {U:\ude-test-and-verify\samples\pls\TriCore2\AppKit_TC275\TimeDemo\s	Core0

Software breakpoints are microcontroller instructions that will cause an (debug) interrupt or trap. To set a software breakpoint, the program code has to be modified. From this follows, software breakpoints can only be used in parts of the program, which are located in a RAM memory area.

Hardware breakpoints are only available by the support of specialized hardware architecture. Hardware breakpoints can be used in all kinds of memory systems.

Note: The **hardware breakpoints** are a limited resource. That is why not all breakpoint constellations are solvable with hardware breakpoints, i.e. breakpoints in complex case expressions. The debugger will use hardware breakpoints automatically, if debugging programs, located in FLASH memory. If too many hardware breakpoints are used by user specified breakpoints, the debugger cannot perform debug steps.

To modify the **code** and **data breakpoints** use the context menu of the specified breakpoint.



Edit Code Breakpoint

Label: 'main {U:\ude-test-and-verify\samples\pls\TriCore2\AppKit_TC275\TimeD

☑ Enable

☑ MultiCore

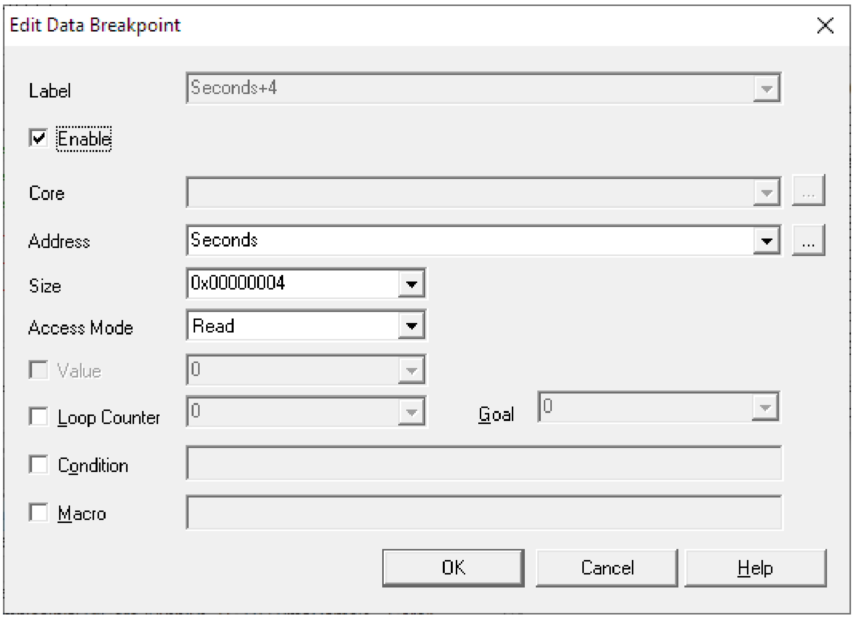
Address: 'main {U:\ude-test-and-verify\samples\pls\TriCore2\AppKit_TC275\TimeD

Type: Hardware

□ Loop Counter: 0 Goal: 0

□ Condition

□ Macro



Edit Data Breakpoint

Label: Seconds+4

☑ Enable

Core:

Address: Seconds

Size: 0x00000004

Access Mode: Read

□ Value: 0

□ Loop Counter: 0 Goal: 0

□ Condition

□ Macro

Loop qualified breakpoints – Loop qualified breakpoints will cause a microcontroller stop after a predefined count of loops over the breakpoint location. The checkbox **Loop Counter** enables the condition for the selected breakpoint. The value shows the current loop count and the **Goal** box defines the endpoint. Both values are editable. Counting the loops is done at the host side, thus the user application is stopped and eventually restarted if the program counter passes the given location.

Condition – Enter a C-Expression here. The expression is evaluated each time the program counter passes the location of the breakpoint. If the result of the expression is not equal to zero, the user application is halted.

Macro – If the user application is halted due to the breakpoint, the macro with the specified name will be executed. Please refer to macros and UDE Object Model how to write and load macros.

Breakpoint identifier

Assembler Breakpoints are defined at their physical address location at assembler level. However, High-Level-Language Breakpoints have a reference to the source code, specified via module, function, source code line and so on. That is why the breakpoint description of an HLL-Breakpoint is more complex.

Syntax description

```
'function {source_file}.line[+]';module[,module2...];line_range;
line_offset;[address_offset];

'label [+ address_offset]';module;

address
```

Element	Description	Example
function	Function name	quick_sort
source_file	Source file name	sort.cpp
line	Line number - A plus sign (optional) means that the breakpoint is near the corresponding address only and it is not a high-level language breakpoint (Assembler breakpoint). Assembler breakpoints have one address only.	50
module	Module name - Module names separated by commas.	SORT

Element	Description	Example
line_range	High-level language breakpoints at source lines can cover different code ranges. This is the position number of the corresponding code range. The number starts with one and increases with growing addresses.	
line_offset	This is the line number offset from start of function to the breakpoint line. This saves the breakpoint description against source file changes around the function.	
address_offset	For Assembler breakpoints, this parameter describes the address offset from the corresponding line or label address to the breakpoint address. Can be null (0x0) to describe an Assembler breakpoint at a line address.	
label	Label name.	quick_sort
address	Pure hexadecimal address with 0x prefix.	0x200

Viewing Memory Locations

For viewing and modifying of the target's memory locations, the **Memory** window is usable. It is available via menu **Views – Memory** or the corresponding tool bar button.

The leftmost column displays the address of the data, and the columns beside the address column show the data in the selected format. You may specify the format using the context menu as Byte for 8-bit data, Word for 16-bit data and DWord for 32-bit data.

Independent of the data width, an entry named Decimal in the context menu allows toggling between hexadecimal and decimal display.

To change values, simply overwrite the selected location with the desired value. The Memory window ignores the input if a key is pressed, which is out of range for the current format (for example, if typing in the letter X in any but ASCII format range).

The size for editing addresses is limited to eight chars. Addresses in integer-based modes must be entered in hexadecimal form without any extensions like `0x` or `h`. This differs from entering addresses in floating point modes.

Please note that you can also type in symbolic names of programs object into the address field.

Writing data to target

There are two modes to write data to the target. The first one **Auto Write** writes the user-modified data as soon as the user moves the cursor to a different location (address) while in the second mode the user is responsible to write the data to the target using the context menu entry **Write** explicitly. Data modified by the user, but not written into the target system, is displayed in blue color.

Note that data marked in blue color might differ from the data in the target.

Toggling between these modes is done by the context menu entry **Auto Write**. After writing the data to the target, the value will be verified. If the value read back from target has the expected value, the color is changed from blue to black. If the writing failed, the color is changed to red.

Updating data from target

The value displayed will be updated, if the user application switches from running to stop and if the **Memory** window is scrolled. The memory takes a new snapshot of the target memory and compares it to the old one. Modified data will be displayed in red color.

Data coloring for different states of the entries:

Register Value Color	State of the Register Value
Red on White	Value has been modified by the program during the last step
Blue on White	Value was changed by the user, but not yet written into the target
Black on White	Value was not changed
White on Red	Data Breakpoint enabled
Black on-pink	Data Breakpoint disabled
White on Blue	Data Range selected

Printing of memory locations

To print the content of the **Memory** window via the system printers, use menu **File – Print** or the shortcut **Ctrl+P**. It is possible to print all pages, a set of pages or a selection. All view modes are supported.

Using the Simulated I/O channel

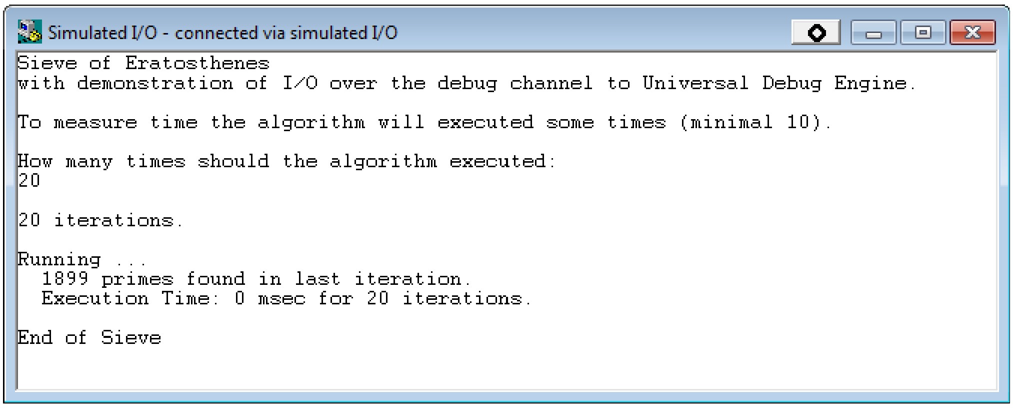
The Universal Debug Engine features the I/O communication of the target application via the debug channel. This means, that the application can use `scanf(), printf()` and equivalent functions to input and output of text.

Requirement is the linking of the `SIMIO.C` software with your application software. `SIMIO.C` is available for the TriCore, AURIX, PowerArchitecture, RISC-V, XC2000, C166 architecture and supports the compilers from Tasking, Keil and HighTec (GNU). The Sieve example from the examples folder demonstrates the usage of the simulated I/O.

If your hardware preparations were successful and your target is connected, load the program `\SAMPLES\C16X\SIEVE\TASKING\SIEVE.OUT` to your target. Open the Simulated I/O Window from the menu **Views – Other Windows – Simulated I/O** or use the corresponding button from the tool bar.

Start the program via menu **Debug – Start Program Execution**. The program output will be redirected to the Simulated I/O window. In the last line, you are asked for the input of the count of executions. The program is waiting in the `scanf()` function until the user press **RETURN**.

Activate the Simulated I/O and type in i.e. `20` and press **RETURN**. The program will continue and calculate the primes. In addition, an execution time measurement is done.



```
Sieve of Eratosthenes
with demonstration of I/O over the debug channel to Universal Debug Engine.

To measure time the algorithm will executed some times (minimal 10).

How many times should the algorithm executed:
20

20 iterations.

Running ...
  1899 primes found in last iteration.
  Execution Time: 0 msec for 20 iterations.

End of Sieve
```

Viewing Data as Scientific Charts

The UDE graphical window displays target program data as series of a scientific chart. It is a powerful visualization tool, which helps to accelerate the evaluation of complex target program data from process environment and the verification of complex software algorithms. It can be used with all microcontroller families supported by UDE.

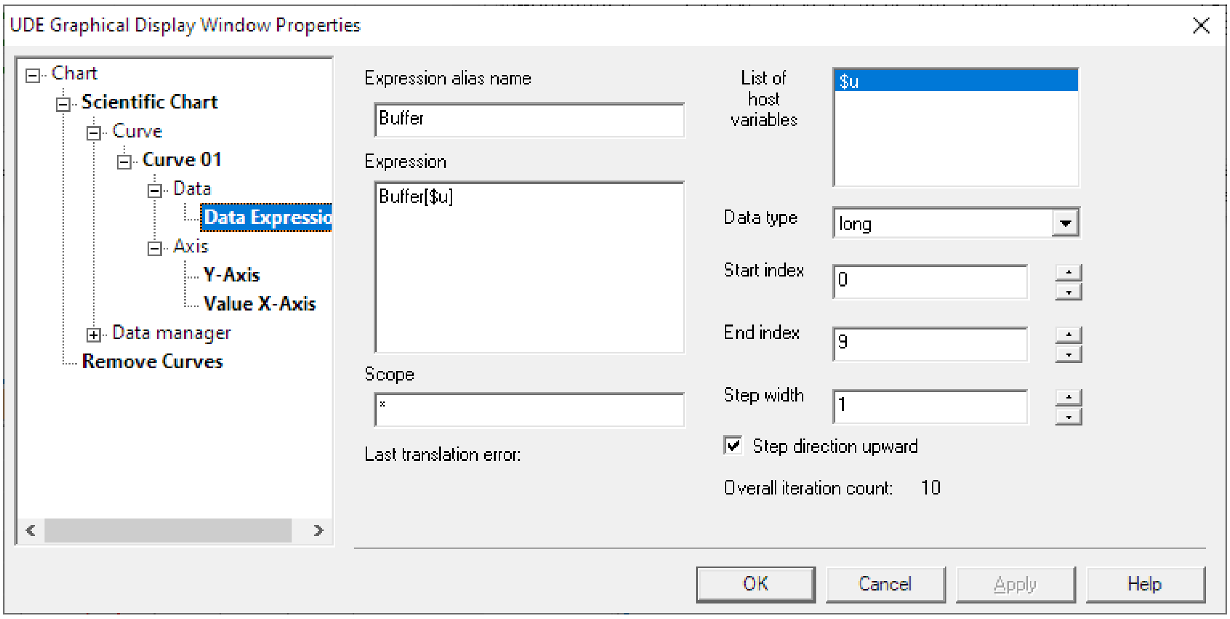
Array Chart

The UDE window displays pre-processed target system data as data series of a 2-dimensional scientific diagram. This feature makes it easier to visualize and evaluate target data to accelerate the verification of complex software algorithms and input from process environment.

To use the graphical window, the array chart is available from the menu **Views – Trace & Analyze Windows – Array Chart** or the corresponding tool bar button.

Using Expressions

The window uses flexible expression specifications, to describe, which parts from target array data are displayed. Using the `timedemo` example, as array expression `Buffer[$u]` can be used.



Expression alias name: Buffer

Expression: Buffer[$u]

Scope: *

Last translation error:

List of host variables: $u

Data type: long

Start index: 0

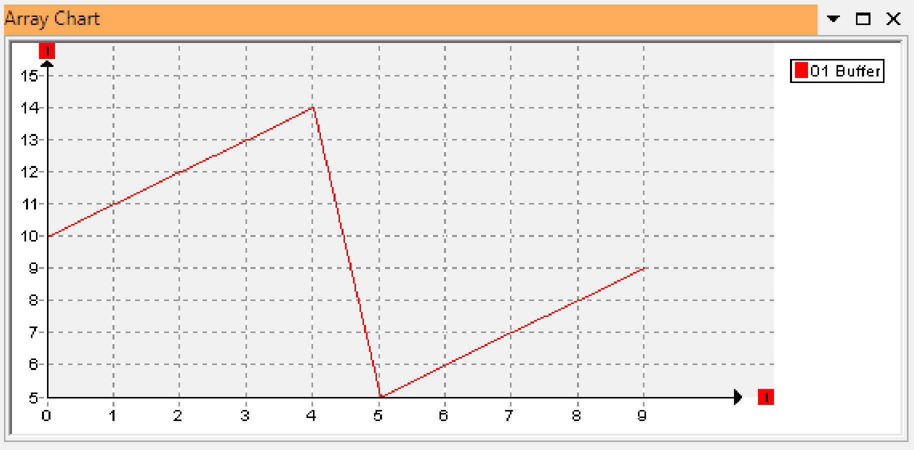
End index: 9

Step width: 1

☑ Step direction upward

Overall iteration count: 10

`$u` is a temporarily host variable defined in minimum `0` and maximum `9`. These limits can be taken from the source code or from the watch expressions.



Array Chart Properties

The context menu allows following modifications:

Zoom changes the cursor to cross cursor to select the Zoom range. Push the left mouse button down at the diagram position, where the zoom rectangle has to start, hold the button down and pull the zoom rectangle to the opposite endpoint. If the button is released, the zoom operation occurs. The nested zoom state is saved and can be recovered with Zoom out function or via the Reset function to return to un-zoomed original state immediately. To release the zoom operation mode, re-open context menu and activate the **Select** function. If a nested zoom-out operation is pending, the context menu contains an additional Zoom out entry.

Zoom out restore last previously stored zoom state or the original window state, if all nested zoom states are restored.

Enter the **Pan** mode and change the cursor into a typical move cursor. If the left mouse button is held down, the whole diagram can be dragged from current view position into another view position. The axis coordinates are changed according to the new position. The Reset function restores the original window state. To release the pan operation mode, re-open context menu and activate the **Select** function.

The **Cursor** mode opens a line cross over the whole diagram area and an additional window, that displays the x- and y- values of all curves at the current line cross center. To release the cursor operation mode, re-open context menu and activate the **Select** function.

Time / Value Chart

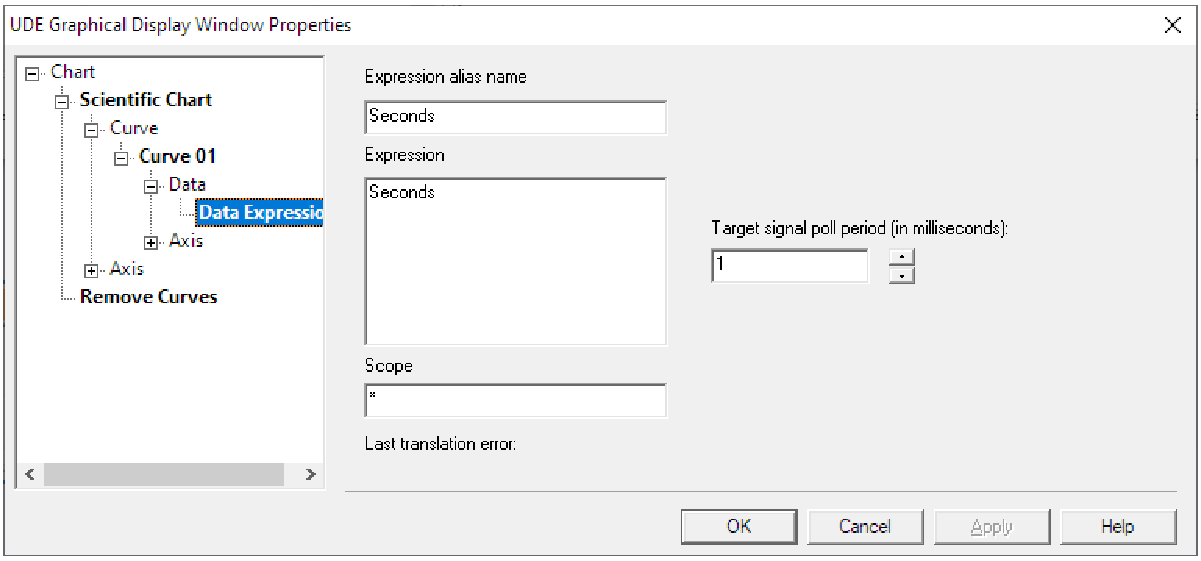
UDE supports real-time graphical-monitoring of target program variables to monitor and analyze the values of variables and complex expressions while the application is running. It can be used with all microcontroller families, which support real-time memory access (currently available for AURIX, TriCore, Power Architecture, which support the IEEE-ISTO NEXUS 5001 compliance classification classes 3/4, ARM Cortex, XC166, XC2000).

Open the window via menu **Views – Trace & Analyze Windows – Time / Value Chart** or the corresponding tool bar button.

The variable browse dialog of **New Signal Wizard** or **Expression Property Page** allows to select expression from new **Expression Clipboard** (see **Adding Variables and Expressions using Select Watches Dialog**) as curve signals. New expressions created by **New Signal Wizard** or **Expression Property Page** are always saved to the UDE global expression clipboard, if scientific time traced signal chart mode is active.

Using Expressions

In this example the `Seconds` variable from the timedemo code is observed.



UDE Graphical Display Window Properties

Expression alias name: Seconds

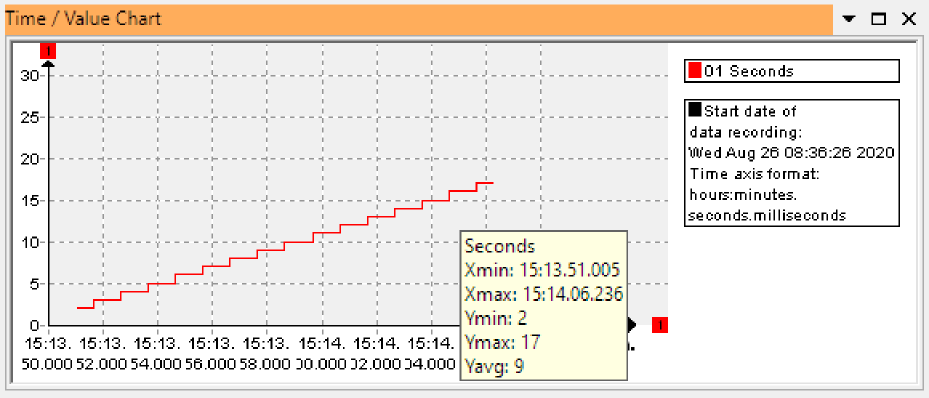
Expression: Seconds

Scope: *

Last translation error:

Target signal poll period (in milliseconds): 1

A **Tooltip** popup window displays all summary information about the curve data or axis data range of the appropriate cursor position.



Start date of data recording: Wed Aug 26 08:36:26 2020
Time axis format: hours:minutes. seconds.milliseconds

Seconds
Xmin: 15:13.51.005
Xmax: 15:14.06.236
Ymin: 2
Ymax: 17
Yavg: 9

Using Modes

The Time / Value Chart supports two display modes, which can be chosen over the context menu **Change Mode of graphical view**.

Target variable time sample mode:

- Evaluation of expressions at regular time intervals. Display as line chart with x-Axis for sample time and y-Axis for evaluated value.

Target variable time sample mode with variable time base:

- X-Y-Plot of two expressions at regular time intervals.

Typical real-time monitoring expressions can consist of:

- Simple basic program variables of integral types.
- Expressions, which describes member of complex variables of integral types.
- Expressions, which calculate results of integral type, based on any expression consisting of multiple basic program variables or basic member of complex types.

Please note, the term “integral types” refers to `signed` / `unsigned character`, `signed` / `unsigned short`, `signed` / `unsigned long` of single or double precision floating point types.

Using complex expressions enables e.g. the collection, calculation and display of real physical values from multiples program variables in real-time. This includes the dynamical access to basic members of complex types, array members and special function register.

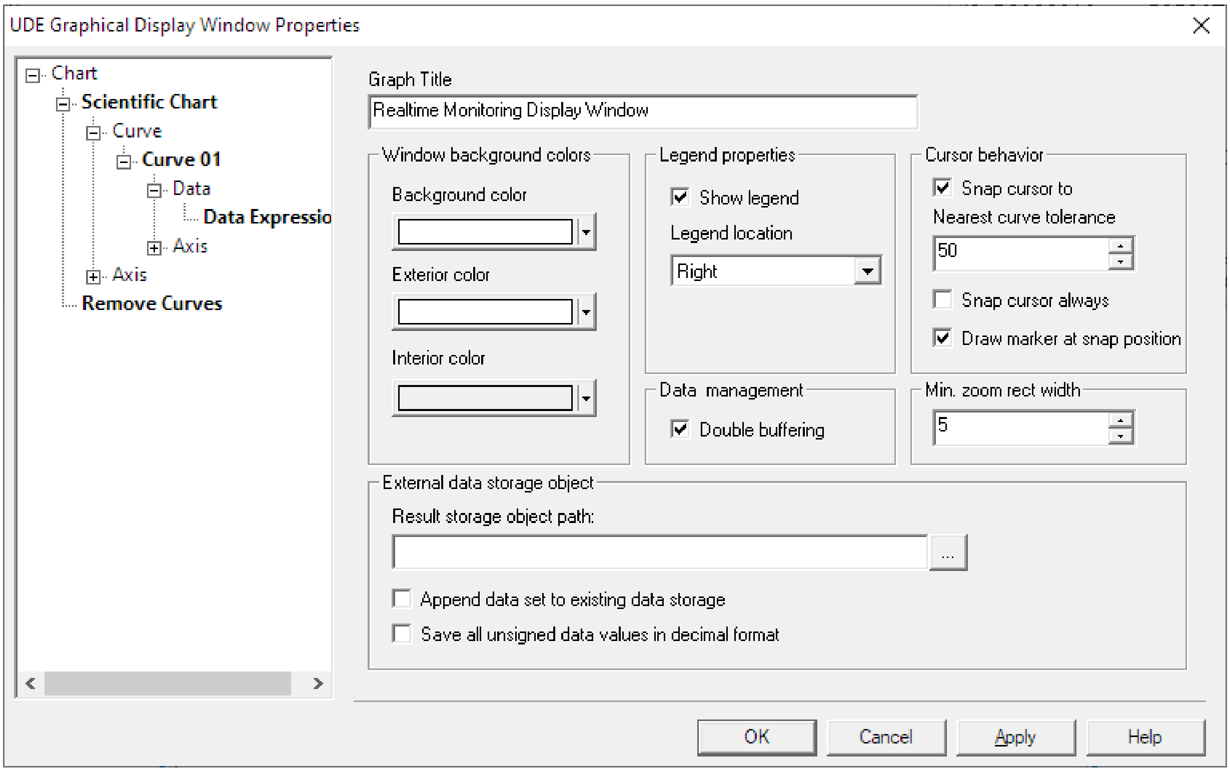
Setting up Real-time Monitoring Display Mode

The real-time monitoring mode has two preconditions:

- The target debug interface must be able to access the target memory locations during running target, with a minimum of real-time violation by this target access.
- Due to the minimum sample period of 1 millisecond, the data collection and pre-processing (expression calculation) must be executed by the firmware of the access-device.

Enable the real-time monitoring display mode by using the context menu entry **Change Mode of graphical view ... Target variable time sample mode**. The wizard dialog for creating a new signal can be opened via context menu entry **New Signal**.

To modify properties of the charts e.g. color, line style etc., use the context menu entry **Properties**. The property page **Scientific Chart** contains all settings covering global user interface settings (colors), the settings for the displayed axis and the settings for each configured signal. The expressions for the signals can be changed over the properties dialog too.



UDE Graphical Display Window Properties

Graph Title: Realtime Monitoring Display Window

Window background colors

Background color

Exterior color

Interior color

Legend properties

☑ Show legend

Legend location: Right

Cursor behavior

☑ Snap cursor to

Nearest curve tolerance: 50

□ Snap cursor always

☑ Draw marker at snap position

Data management

☑ Double buffering

Min. zoom rect width: 5

External data storage object

Result storage object path:

□ Append data set to existing data storage

□ Save all unsigned data values in decimal format

If **Cursor Mode** is active, the **Snap to cursor to nearest curve** flag enables to snap current cursor location to the nearest valid curve coordinates. If **Double buffering** is on, the curve data are stored double buffered. Double buffering avoids flicker effects during re-paint operations.

The trace results of the Data Sequences can be stored in an XML based data set collection. A new data set will be generated after each measurement period (depends from selected display mode). The **Result storage object path** of this XML file can be selected. The measurement period starts with the first program code run after program download. It will be closed by termination of debug session or program reload or new program download.

A new data set, which was generated after each measurement period, can be added to an existing data set collection or it can discard earlier generated data set via **Append data set to existing data storage**.

Setting up Memory Locations Display Mode

Enable the memory locations display mode by using the context menu entry **Change Mode of graphical view ... Target variable time sample mode with variable time base**. The expressions are specified via context menu: First select **New X-axis translation Signal** and then **New Signal** to open a wizard dialog for creating the expression. If multiple signals are specified, they will use the same X-axis.

To modify properties of the charts e.g. color, line style etc., use the context menu entry **Properties**. The property page **Scientific Chart** contains all settings covering global user interface settings (colors), the settings for the displayed axis and the settings for each configured signal. The expressions for the x-axis and signals can be changed over the properties dialog too.

Viewing Call Stack

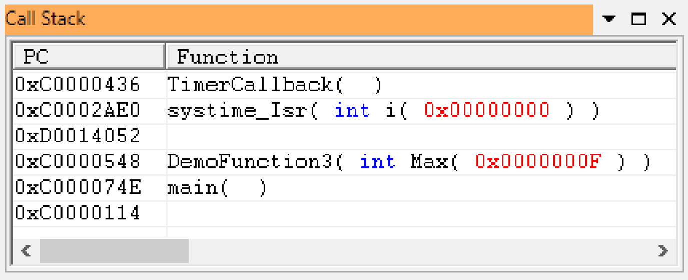
The Call Stack window displays the calling hierarchy of the program. The top entry shows the actually IP location and the name of the currently executed function, if available. The entries below list the return addresses in the sequence of their occurrence.

The stack window supports an easy navigation through the current program hierarchy: double-click on a stack entry, and the corresponding location is shown in the program window.

Once a line is selected, the default context in UDE is changed to this location and all information in watches, locals and in the Core Registers window will be updated to the selected context.

The example at the right shows the current IP location is `0xC0000436` in the `TimerCallback()` function. This function was called by an interrupt occurring while executing the `DemoFunction3()` function. The interrupt handler will return to the address `0xC0000548` in the `DemoFunction3()` function. The entry between the `systime_Isr()` and `DemoFunction3()` is the interrupt vector code.

The entry at the bottom shows the return point in the start-up code. Because the interrupt vector and the start-up code are not part of the high-level-language of the program, no function names are displayed.



PC	Function
0xC0000436	TimerCallback( )
0xC0002AE0	systime_Isr(int i(0x00000000))
0xD0014052	
0xC0000548	DemoFunction3(int Max(0x0000000F))
0xC000074E	main( )
0xC0000114	

Call Graph Analysis

The Call Graph Analysis (CGA) is a trace analysis to create a representation of the control flow on software level from the control flow content in trace messages and accumulates method call count and runtime information. The call graph represents the Caller - Callee relation of subroutines and functions (later generally referred to as subroutines) of a software module.

The analysis uses symbolic information provided to UDE by the loaded ELF and the instruction addresses contained in the trace messages to reconstruct subroutine enter and leave events. These events are used to reconstruct a call graph and collecting the number of calls of a subroutine. If timing information are available (that is the tick value of the trace messages), the difference between method enter and leave is calculated and used to provide statistic information, like minimum, maximum and average call time inclusive and exclusive subroutine calls.

Enabling the Call Graph Analysis

The UDE Call Graph Analysis is an Add-In of UDE and must be activated before it can be used. To activate it, open the Add-In Manager, menu **Config – Add-in Components**. Enable the entry **Call Graph Analysis** and open menu **Views – Trace & Analyze Windows – Call Graph**.

Configuring of Trace Configuration

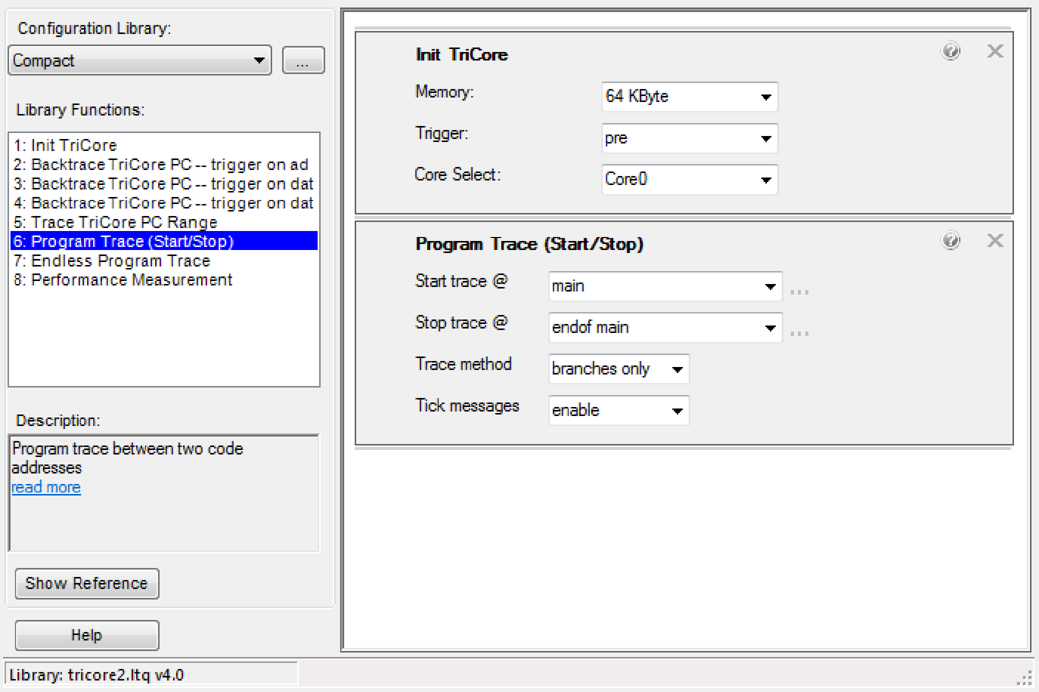
This example uses a TC1766ED with MCDS support and as program the `Timedemo.elf` from the example directory.

Open the UEC trace configuration window via menu **Tools** – **Configure Trace…** to create a suitable trace configuration to record program control flow for Core 0 of the TC1766ED.

Execute the following steps to create the required configuration:

- Switch to UEC configuration window.
- Leave the **Compact** library.
- Drag the **Init TriCore** library element to configuration area.
- Drag the **Program Trace (Start/Stop)** library element to configuration area.
- Set **Start trace** @ to e.g. `main`
- Set **Stop trace** @ to e.g. `endof main`

The optimized MCDS trace configuration for this use-case is now completed.



Configuration Library: Compact

Library Functions:

1: Init TriCore
2: Backtrace TriCore PC -- trigger on ad
3: Backtrace TriCore PC -- trigger on dat
4: Backtrace TriCore PC -- trigger on dat
5: Trace TriCore PC Range
6: Program Trace (Start/Stop)
7: Endless Program Trace
8: Performance Measurement

Description:

Program trace between two code addresses
read more

Init TriCore

Memory: 64 KByte
Trigger: pre
Core Select: Core0

Program Trace (Start/Stop)

Start trace @ main
Stop trace @ endof main
Trace method branches only
Tick messages enable

Library: tricore2.ltq v4.0

Using the Call Graph Analysis

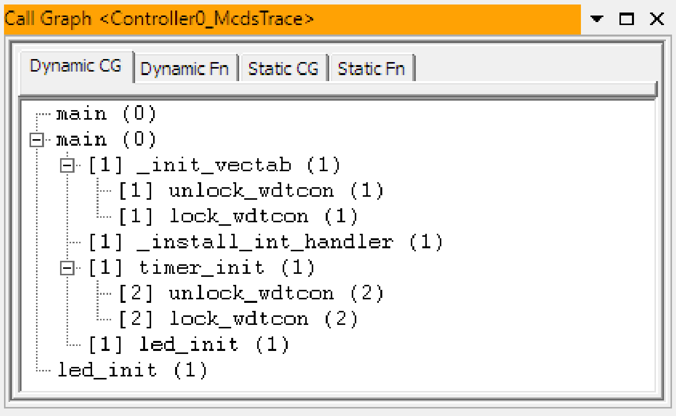
Now start the trace recording via menu **Tools** – **Start trace …** and start the program via menu **Debug** – **Start Program execution**. When the trace recording has stopped (because of a manual stop or because the trace memory is full), you can start the **Call Graph Analysis** via context menu **Start**.

The CGA creates per default 2 kind of graphs: The static call graph and the dynamic call graph:

- The dynamic call graph describes the Caller – Callee relationship with respect only to the immediate parent of a node (local). This means, if the same subroutine is called from two different parents or at different levels of the caller hierarchy, the subroutine will be treated as separate instances. Statistic value will be accumulated for each instance separately.
- The static call graph describes the Caller – Callee relationship with respect to all parents of a node (global). This means each subroutine will be treated as unique instance in the complete graph. Statistic value will be accumulated from all calls to the subroutine, regardless of the immediate caller or call level.

This difference results in two sets of graph representation and accumulated statistic value.

Dynamic Call Graph

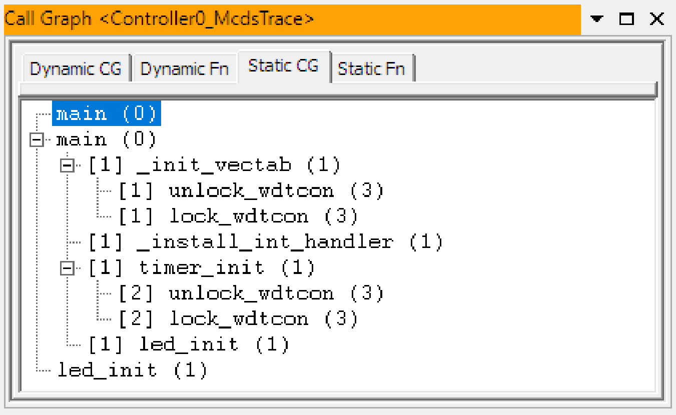


```
main (0)
main (0)
  [1] _init_vectab (1)
    [1] unlock_wdtcon (1)
    [1] lock_wdtcon (1)
  [1] _install_int_handler (1)
  [1] timer_init (1)
    [2] unlock_wdtcon (2)
    [2] lock_wdtcon (2)
  [1] led_init (1)
led_init (1)
```

The label of the call graph nodes includes additional information for the number of times a subroutine was called by the immediate parent node (the value in squared braces, the **Edge Call** value) and how often the node was called totally (the value in round braces, the **Node Call** value).

In a dynamic call graph, both values are always the same, since each subroutine call from a different parent is treated as different instance, for which calls are accumulated independently.

Static Call Graph

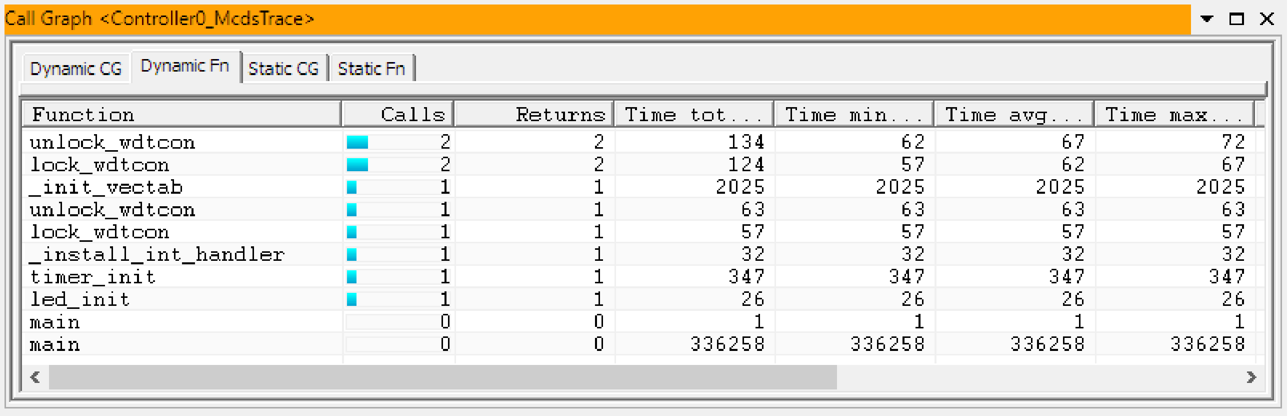


```
main (0)
main (0)
  [1] _init_vectab (1)
    [1] unlock_wdtcon (3)
    [1] lock_wdtcon (3)
  [1] _install_int_handler (1)
  [1] timer_init (1)
    [2] unlock_wdtcon (3)
    [2] lock_wdtcon (3)
  [1] led_init (1)
led_init (1)
```

The label of the call graph nodes includes additional information about the number of times a subroutine was called by the immediate parent node (the value in squared braces, the **Edge Call** value) and how often the node was called totally (the value in round braces, the **Node Call** value).

In a static call graph, both values can differ, if the same subroutine was called from different parents or at different call levels.

Dynamic Node Call and Timing Value



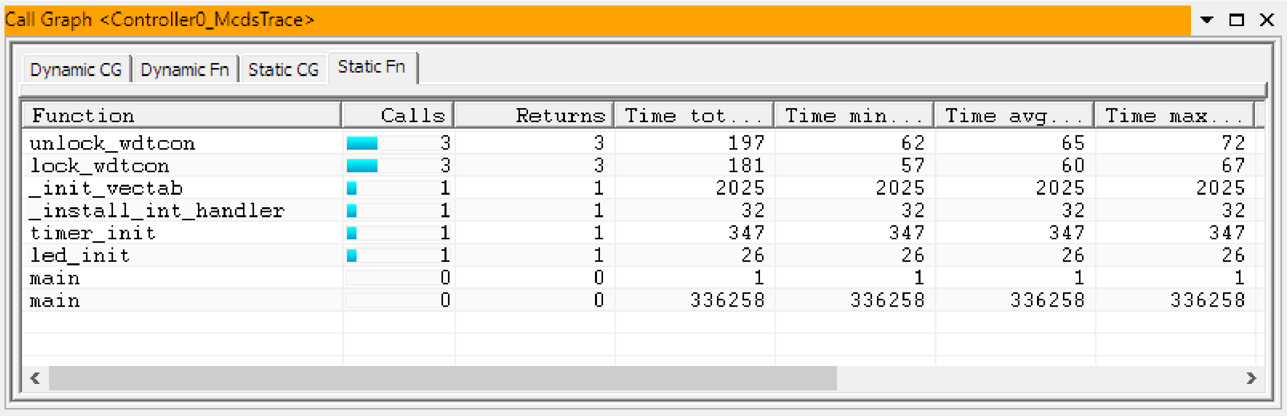
Function	Calls	Returns	Time tot...	Time min...	Time avg...	Time max...
unlock_wdtcon	2	2	134	62	67	72
lock_wdtcon	2	2	124	57	62	67
_init_vectab	1	1	2025	2025	2025	2025
unlock_wdtcon	1	1	63	63	63	63
lock_wdtcon	1	1	57	57	57	57
_install_int_handler	1	1	32	32	32	32
timer_init	1	1	347	347	347	347
led_init	1	1	26	26	26	26
main	0	0	1	1	1	1
main	0	0	336258	336258	336258	336258

The table in Dynamic Node Calls and Timing describes the statistical node call and timing value related to Dynamic Call Graph example. It is a listing of all nodes of the dynamic call graph. Since each call from a different subroutine is treated as different instance of the subroutine, the function name of subroutines called from different parents appear multiple times. This is also true for recursive calls, since the parent subroutine in the upper call graph is treated as separate instance.

The **Calls** column shows the accumulated number of calls of the subroutine from a parent subroutine. The **Return** column shows, how many times the subroutine returned to a parent. Please note, that only calls and returns recorded in trace are counted in these columns. If the trace starts in a subroutine, the start of trace is **not** recorded as call, since there is no explicit call from a parent subroutine recorded in trace. Likewise, for the return from trace. In the example above, the trace started and ended in subroutine CGARecursive. Since no call to CGARecursive and no return from CGARecursive to a parent subroutine were recorded, the Calls and Returns columns are zero, even though trace **inside** the subroutine was recorded.

Other columns are displaying the statistical timing value for the inclusive or exclusive subroutine calls: total time (Time tot incl/excl), minimal execution time of a single call (Time min incl/excl), the average execution time of a single call (Time avg incl/excl) and the maximal execution time of a single call (Time max incl/excl).

Static Node Call and Timing Value



Function	Calls	Returns	Time tot...	Time min...	Time avg...	Time max...
unlock_wdtcon	3	3	197	62	65	72
lock_wdtcon	3	3	181	57	60	67
_init_vectab	1	1	2025	2025	2025	2025
_install_int_handler	1	1	32	32	32	32
timer_init	1	1	347	347	347	347
led_init	1	1	26	26	26	26
main	0	0	1	1	1	1
main	0	0	336258	336258	336258	336258

The table in Static Node Calls and Timing Example is the description of the node call and timing value, and related to Static Call Graph Example. It is a listing of all nodes of the static call graph. Since the recursive call to subroutine Level is recorded for a unique instance, only one node is displayed here, which accumulates the calls, returns and timing value. The columns of the table are the same as in the dynamic call graph table, but the value will differ because of their different accumulation method.

Program Execution Time Measuring

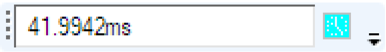
Program execution time measuring is supported by UDE via

- Trace unit on target. Please refer to the chapter about **Trace, Visualization and Analyzing**.
- Timer unit on target.

To use time measuring via target timers the feature has to be enabled in the target interface when using some target architectures. Connect to the target and open the menu **Config – Target Interface…** setup. Open the page **Monitor** respectively the page **Debug** and enable the Program execution time measuring. Select the used timers for **Time Measurement** if available.

Open the tool bar **Tools**, double-click to the field labeled with **Function disabled** per default. A property dialog will be opened. Enable the time measuring in this dialog.

Select the adequate working mode:



41.9942ms

- Continuous mode - the timer accumulates the lapsed time
- Single step mode - the timer shows the lapsed time of the last single step of UDE.

Trace, Visualization and Analyzing

Hard real-time debugging requires close interaction with the processor. Tracing shall provide a chronological picture of a system's inner workings up to, starting from or near an event, mainly to guide a human in understanding a faulty program.

System Level Debugging

	Profiling	Code Coverage	Trace Timeline	Variable access
Measurement	Amount of execution time for each function	Execution of code	Call hierarchy and execution time sequence	Memory changes
Requirements	Code trace with tick information (Subroutine only possible)	Code trace without tick information	Code trace with tick information	Data trace with tick information
Results	Graphical chart and reports	Graphical chart and reports	Graphical chart	Graphical chart

Different semiconductor vendors have defined trace interfaces like **MCDS, NEXUS, CoreSight, ETM** and **ETB** for this purpose. These trace ports are available on the AURIX, TriCore ED, Power Architecture, XC2000ED and ARM derivatives. The UAD2^{next} with Trace add-on and UAD3^{+} with parallel Trace Pod supports **NEXUS** and **ETM** parallel trace.

MCDS and ETB are embedded trace ports including the trace buffer on-chip, which reduces the requirements of an external trace analyzer to a minimum. AURIX ED processors support MCDS trace, several Automotive Power Architecture processors support NEXUS trace and several ARM processors support ARM HSSTP over high-speed serial Aurora trace interface. UAD3^{+} supports these high-speed serial trace interfaces by special Trace Pods.

Triggered Transfer is a feature of AURIX, TriCore and XC2000/XE166 microcontrollers for high performance polling of target memory. Memory Transfer means the transfer of data via debug channel.

Cortex can provide trace event messages via the Serial Wire Output SWO or Instrumentation Trace Messages ITM.

MCDS support for TriCore ED, AURIX ED microcontrollers requires an additional UEC license (UDE-TC UEC).
The AURIX TC29, TC37, TC38 microcontrollers are equipped with **miniMCDS**, which offers limited functionality for trace-based debugging even for production devices. **miniMCDS** can be used for simple trace-based debugging with the standard version of UDE. An additional trace license (UDE-TC UEC) is only required for using advanced features of miniMCDS together with the graphical trace configurator UEC.

Trace Sources

	MCDS	**MCDS Aurora Trace**	**NEXUS**	**NEXUS Aurora Trace**
Processor support	EmulationDevice TriCore ED, XC2000 ED, AURIX ED	EmulationDevice AURIX ED,	Automotive Power Architecture	Automotive Power Architecture
Output format	Event Messages	Event Messages	Event Messages	Event Messages
Time accuracy[1]	Message Tick	Message Tick	Message Tick	Message Tick
Trace memory	On-Chip Memory: up to 2048 kByte	UAD2^{next} - 512 MByte, UAD3^{+} - 4 GByte	UAD2^{next} - 512 MByte, UAD3^{+} - 4 GByte	UAD2^{next} - 512 MByte, UAD3^{+} - 4 GByte
Trace covering	Instruction Trace Data Trace	Instruction Trace Data Trace	Instruction Trace Data Trace	Instruction Trace Data Trace
Advantages	Continuous trace, complex trace filter	Continuous trace, complex trace filter	Continuous trace	Continuous trace
Disadvantages	Limited by size of Trace Board buffer	Limited by size of Trace Board buffer	Limited by size of Trace Board buffer	Limited by size of Trace Board buffer

Trace Sources (cont'd)

	CoreSight ETM	**Triggered Transfer**	**Memory Transfer**	**ETB**
Processor support	ARM, Cortex	TriCore, XC166, XC2000, XE166	All except ARM	ARM, Cortex
Output format	Event Messages	Data Snapshots	Signal and IP Snapshots	Pipeline States
Time accuracy[1]	Message Tick	Endless Trace ≥ 100ms	Endless Trace ≥ 1ms	Processor Tick
Trace memory	UAD2^{next} - 512 MByte, UAD3^{+} - 4 GByte	Buffered in UAD2, UAD3	Buffered in UAD2, UAD3	On-Chip Memory: up to 6 kByte
Trace covering	Instruction Trace Data Trace	Data Trace	IP and Data Trace	Instruction Trace Data Trace
Advantages	Continuous trace	Endless trace possible	Endless trace possible	Continuous trace
Disadvantages	Limited by size of Trace Board buffer	Non-continuous trace	Non-continuous trace	Limited by size of Trace Board buffer

Trace Sources (cont'd)

	ITM / SWO	ARM HSSTP Aurora Trace
Processor support	Cortex	Automotive ARM/Cortex
Output format	Event Messages	Event Messages
Time accuracy[1]	Endless Trace	Message Tick
Trace memory	Buffered in UAD2, UAD3	UAD2^{next} - 512 MByte, UAD3^{+} - 4 GByte
Trace covering	Data Trace	Instruction Trace Data Trace
Advantages	Endless trace possible	Continuous trace
Disadvantages	Non-continuous trace	Limited by size of Trace Board buffer

[1] Depends on target system

Trace Analyzing Features and Windows

Analyzing of traced data is one of the features supported by UDE. The analyzing functions support different use-cases. Different types of windows support the visualization of the results of these use-cases. The table below shows an overview of the supported trace visualization and analyzing use-cases.

Trace Visualization and Analyzing

	Profiling (stats)	Profiling (trace)	Code Coverage (trace)	Program Flow
View format	Bar chart in **Profiling Window**	Bar chart in **Profiling Window**	Tree chart in **Code Coverage Window**	Gantt chart in **Execution Sequences**
Trace source	Memory Transfer Periodical debug read by UAD2/3	MCDS NEXUS, ETM, ETB, CoreSight	MCDS NEXUS, ETM, ETB, CoreSight	MCDS NEXUS, ETM, ETB, CoreSight
Method of Visualization	Displays hits of sampled IP addresses assigned to function ranges	Displays recorded IP addresses hits assigned to function ranges	Displays code coverage results of all executed functions in multiple levels	Displays traced code addresses hits assigned to function ranges or hexadecimal addresses over time axis
Advantages	Endless trace, good approximation about most runtime-consuming program parts	Exact profiling results over the recorded time range of IP addresses in trace buffer	Flow of recorded IP addresses, detection of incomplete execution of code	Graphical display of program trace records to get a better global overview about program flow

Trace Visualization and Analyzing (cont’d)

	Data Trace	**Trace View**	**Signal Trace**	**Signal Log**
View format	Line chart in **Data Trace Chart**	Message Log in **Trace window**	Line Chart in **Time / Value Chart**	Message Log in **Data Sequences**
Trace source	MCDS NEXUS, ETM, ETB, CoreSight	MCDS NEXUS, ETM, ETB, CoreSight, ITM	Memory Transfer Periodical debug read by UAD FW	ITM, Trigger Transfer
Method of Visualization	Displays recorded read- and/or write access of variables over time axis	Displays all recorded information contained in raw trace data	Displays values of a variable or expression over time axis	Displays all recorded information contained in raw trace data
Advantages	Graphical display of variable trace records to get a better global overview about time of variable access	Raw trace, flow of recorded IP data related to appropriate source code	Endless trace, full scaling of time and value interval	Raw trace, flow of recorded data

Trace Views

MCDS, miniMCDS trace, NEXUS trace (some types) and ETB trace can be used without any additional hardware add-on.

Dedicated AURIX, Power Architecture and ARM/Cortex emulation devices support high-speed serial trace output via Aurora protocol. Therefore, a UAD2^{next} or UAD3^{+} with Aurora Serial Trace Pod is required.

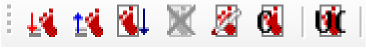
Using the NEXUS parallel trace, CoreSight, ETM trace features with UDE is only possible with UAD2^{next} or UAD3^{+} with Parallel Trace Pod.

Usage of external trace requires an additional target specific interface cable. The complete overview over all target specific interface solutions is available within the actual PLS’ product information or on our website **https://www.pls-mc.com** or ask PLS support (**support@pls-mc.com**) for further details.

Configuring the Trace Window

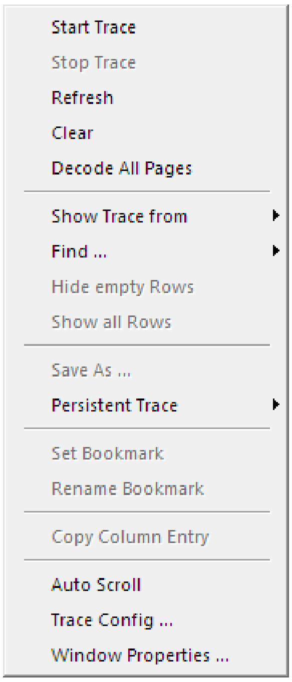
The actual UDE configuration must be configured to support one of the hardware specific UDE trace channels. If this feature is configured, a tool bar and a menu item to open the Trace window are available. Then, open the Trace window by menu **Views – Trace & Analyze Windows – Trace**. Following user interface elements are additional available, if trace feature is available:

- Trace functions in **Tools** tool bar

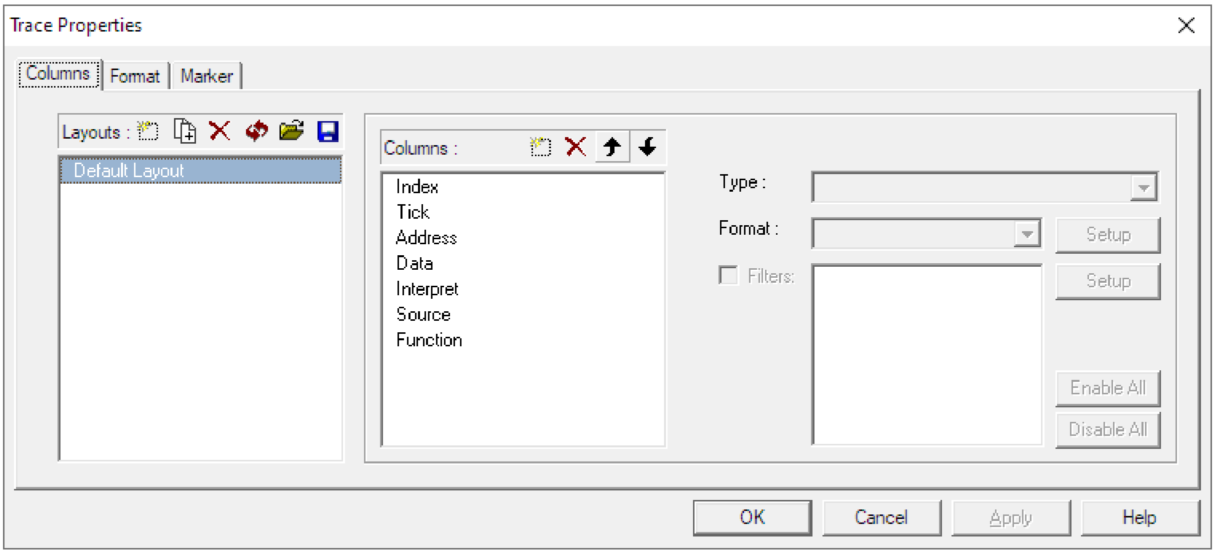


  - Save trace steam
  - Load trace stream
  - Start trace
  - Stop trace
  - Clear trace data
  - Configure trace
  - Trace use case
- Same functions are available within **Tools** menu

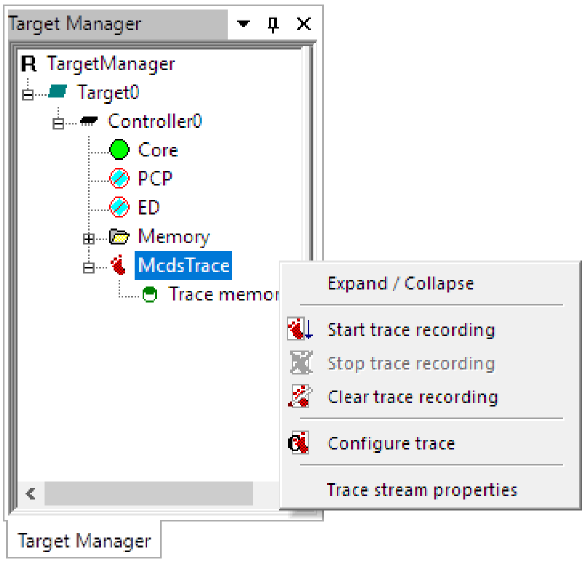
- The local menu of the Trace window provides also the most functions of these bars.
  - **Start / Stop Trace** activate/deactivate the trace manually. Before the trace can be started, the trace must be configured correctly.
  - **Clear** deletes trace data so the window is completely empty after execution of this command.
  - **Decode All Pages**
  - **Show Trace from** helps you to navigate to the top, the bottom and the trigger position. For a finer resolution, use the Find options.
  - With **Find ...** you can search stepwise for a term in the specified column of trace buffer or all occurrences in the trace stream at once.
  - The **Hide / Show Empty Rows** menu-point toggles the display mode of the window. The trace buffer contains often many empty samples (Pipeline holes). For the choice of a compact view of the program trace, hide the empty rows. For an isochronous view, choose the empty rows for showing.



  - Use the **Save As ...** to save the displayed samples to a format file. Multiple formats are possible; most commonly used are the columns tabulator separated (`*.tab`) or space characters separated (`*.txt`) format.
  - **Copy Column Entry**
  - **Auto Scroll**
  - **Trace Config …** opens the trace channel specific trace configuration window or dialog.
  - Configure the Trace window properties via the **Window Properties ...** menu entry:



- The local menu of the appropriate trace stream node in the **Target Manager** window also provides a local menu with all global control functions of the trace stream.



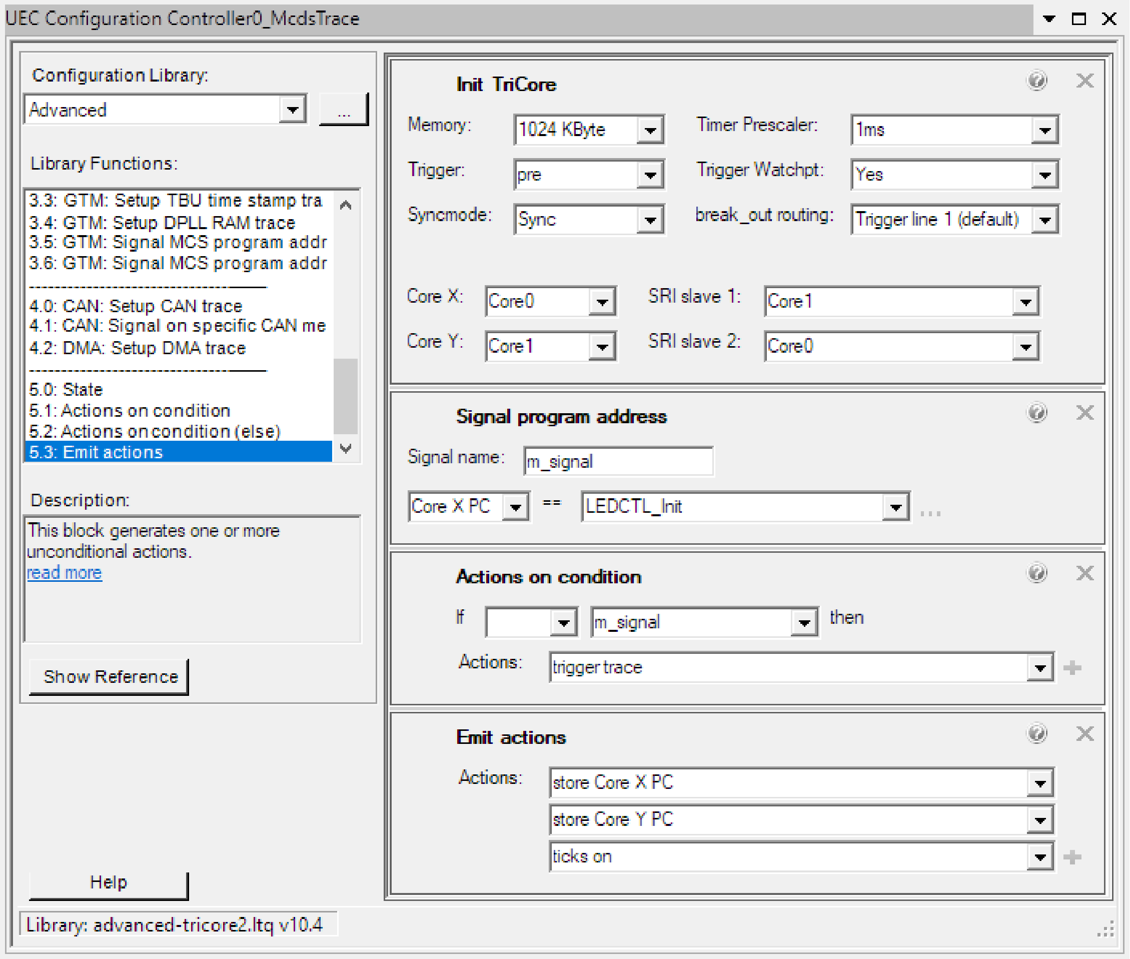
Configuring of Trace Configuration

The Trace window can display different kinds of trace messages and is an ideal tool to identify complex problems, e.g. to find multi-core software run-conditions. This example uses a TC27xxED with MCDS support and the program `TimeDemo.elf` from the example directory.

Open the UEC trace configuration window via menu **Tools** – **Configure Trace** to create a trace configuration to record the multi-core start sequence between core 0 and core 1 of TC27xx. Execute the following steps to create the required configuration:

- Switch to UEC configuration window.
- Switch from **Compact** to **Advanced Configuration** library.
- Drag the **Init TriCore** library element (1.0) to configuration area.
- Change **Memory Size** to maximum value of 1024 kByte
- Change **Syncmode** to mode Sync to set the best tick resolution between traced code samples
- Change **Core Y** to core 1.
- Drag the **Signal program address** library element (2.1) to configuration area.
- Set **Signal name** to e.g. `my_signal`
- Select `LEDCTL_Init` function start address as comparison address for Core X PC
- Drag the **Actions on condition** library element (5.1) to configuration area.
- Browse `my_signal` as signal name for If condition.
- Select as action **Trigger trace**
- Drag the **Emit actions** library element (5.3) to configuration area.
- Add emit action **store Core X PC**
- Add emit action **store Core Y PC**
- Add emit action **ticks on**

The MCDS trace configuration for this special task is now completed.



UEC Configuration Controller0_McdsTrace

Configuration Library: Advanced

Library Functions:

3.3: GTM: Setup TBU time stamp tra
3.4: GTM: Setup DPLL RAM trace
3.5: GTM: Signal MCS program addr
3.6: GTM: Signal MCS program addr

4.0: CAN: Setup CAN trace
4.1: CAN: Signal on specific CAN me
4.2: DMA: Setup DMA trace

5.0: State
5.1: Actions on condition
5.2: Actions on condition (else)
5.3: Emit actions

Description:

This block generates one or more unconditional actions.
read more

Library: advanced-tricore2.ltq v10.4

Init TriCore

Memory: 1024 KByte
Timer Prescaler: 1ms
Trigger: pre
Trigger Watchpt: Yes
Syncmode: Sync
break_out routing: Trigger line 1 (default)
Core X: Core0
SRI slave 1: Core1
Core Y: Core1
SRI slave 2: Core0

Signal program address

Signal name: m_signal
Core X PC == LEDCTL_Init ...

Actions on condition

If m_signal then
Actions: trigger trace

Emit actions

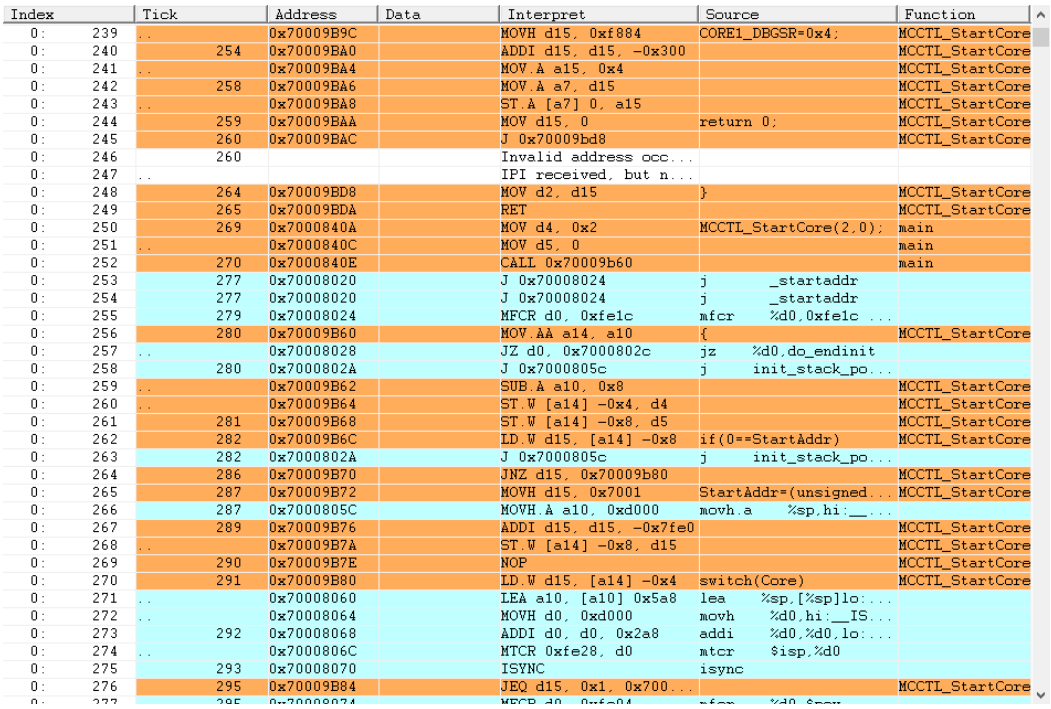
Actions: store Core X PC
store Core Y PC
ticks on

Now you can start the trace recording (tool bar icon or menu **Tools – Start trace...** or Trace window local menu start trace function). After start of program execution, the **Progress** window completes the task:

- ➢ Record trace stream to memory

Analyzing Results

After completion of this task, the Trace window displays the first page of the decoded trace stream.

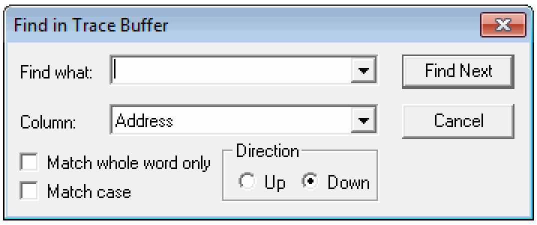


Index	Tick	Address	Data	Interpret	Source	Function
0: 239	..	0x70009B9C		MOVH d15, 0xf884	CORE1_DBGSR=0x4;	MCCTL_StartCore
0: 240	254	0x70009BA0		ADDI d15, d15, -0x300		MCCTL_StartCore
0: 241	..	0x70009BA4		MOV.A a15, 0x4		MCCTL_StartCore
0: 242	258	0x70009BA6		MOV.A a7, d15		MCCTL_StartCore
0: 243	..	0x70009BA8		ST.A [a7] 0, a15		MCCTL_StartCore
0: 244	259	0x70009BAA		MOV d15, 0	return 0;	MCCTL_StartCore
0: 245	260	0x70009BAC		J 0x70009bd8		MCCTL_StartCore
0: 246	260			Invalid address occ...		
0: 247	..			IPI received, but n...		
0: 248	264	0x70009BD8		MOV d2, d15	}	MCCTL_StartCore
0: 249	265	0x70009BDA		RET		MCCTL_StartCore
0: 250	269	0x7000840A		MOV d4, 0x2	MCCTL_StartCore(2,0);	main
0: 251	..	0x7000840C		MOV d5, 0		main
0: 252	270	0x7000840E		CALL 0x70009b60		main
0: 253	277	0x70008020		J 0x70008024	j _startaddr	
0: 254	277	0x70008020		J 0x70008024	j _startaddr	
0: 255	279	0x70008024		MFCR d0, 0xfe1c	mfcr %d0,0xfe1c ...	
0: 256	280	0x70009B60		MOV.AA a14, a10	{	MCCTL_StartCore
0: 257	..	0x70008028		JZ d0, 0x7000802c	jz %d0,do_endinit	
0: 258	280	0x7000802A		J 0x7000805c	j init_stack_po...	
0: 259	..	0x70009B62		SUB.A a10, 0x8		MCCTL_StartCore
0: 260	..	0x70009B64		ST.W [a14] -0x4, d4		MCCTL_StartCore
0: 261	281	0x70009B68		ST.W [a14] -0x8, d5		MCCTL_StartCore
0: 262	282	0x70009B6C		LD.W d15, [a14] -0x8	if(0==StartAddr)	MCCTL_StartCore
0: 263	282	0x7000802A		J 0x7000805c	j init_stack_po...	
0: 264	286	0x70009B70		JNZ d15, 0x70009b80		MCCTL_StartCore
0: 265	287	0x70009B72		MOVH d15, 0x7001	StartAddr=(unsigned...	MCCTL_StartCore
0: 266	287	0x7000805C		MOVH.A a10, 0xd000	movh.a %sp,hi:__...	
0: 267	289	0x70009B76		ADDI d15, d15, -0x7fe0		MCCTL_StartCore
0: 268	..	0x70009B7A		ST.W [a14] -0x8, d15		MCCTL_StartCore
0: 269	290	0x70009B7E		NOP		MCCTL_StartCore
0: 270	291	0x70009B80		LD.W d15, [a14] -0x4	switch(Core)	MCCTL_StartCore
0: 271	..	0x70008060		LEA a10, [a10] 0x5a8	lea %sp,[%sp]lo:...	
0: 272	..	0x70008064		MOVH d0, 0xd000	movh %d0,hi:__IS...	
0: 273	292	0x70008068		ADDI d0, d0, 0x2a8	addi %d0,%d0,lo:...	
0: 274	..	0x7000806C		MTCR 0xfe28, d0	mtcr $isp,%d0	
0: 275	293	0x70008070		ISYNC	isync	
0: 276	295	0x70009B84		JEQ d15, 0x1, 0x700...		MCCTL_StartCore
0: 277	295	0x70008074		MFCR d0, 0xfe04	mfcr %d0,$psw	

Identify Core related Messages

The core related trace messages are colored in the core specific framework windows tabs colors. These colors can be assigned by UDE configuration dialog (menu **Config – Debug Server Configuration – Windows Tabs Color**).

Trace Buffer Find Dialog



Find what defines the string to find in a column. The search string can be made case sensitive by setting the check mark. The find operation returns any row in the selected column that contains a substring, (or the whole word, if checked) which matches the input search pattern. The dialog saves user input values in a history for later re-use for additional searches. The history is accessible with the drop-down box.

Select the **Column** from the drop-down box, which has to contain the search string. This value is preselected corresponding to the right-click position automatically.

The **Up** and **Down** specifies the direction to search for. The search starts at the selected row. **Find Next** starts the search. The dialog remains open without successful search operation. Otherwise, the dialog is automatically closed and the window scrolls to the found row in the view area and highlights it.

Switch to Source Code

After selection of a row, which contains code trace messages, a double click on this row forces the program window to open the source code of the selected code address and shows the related code position in the current mode of this program window (assembly or high-level language source code).

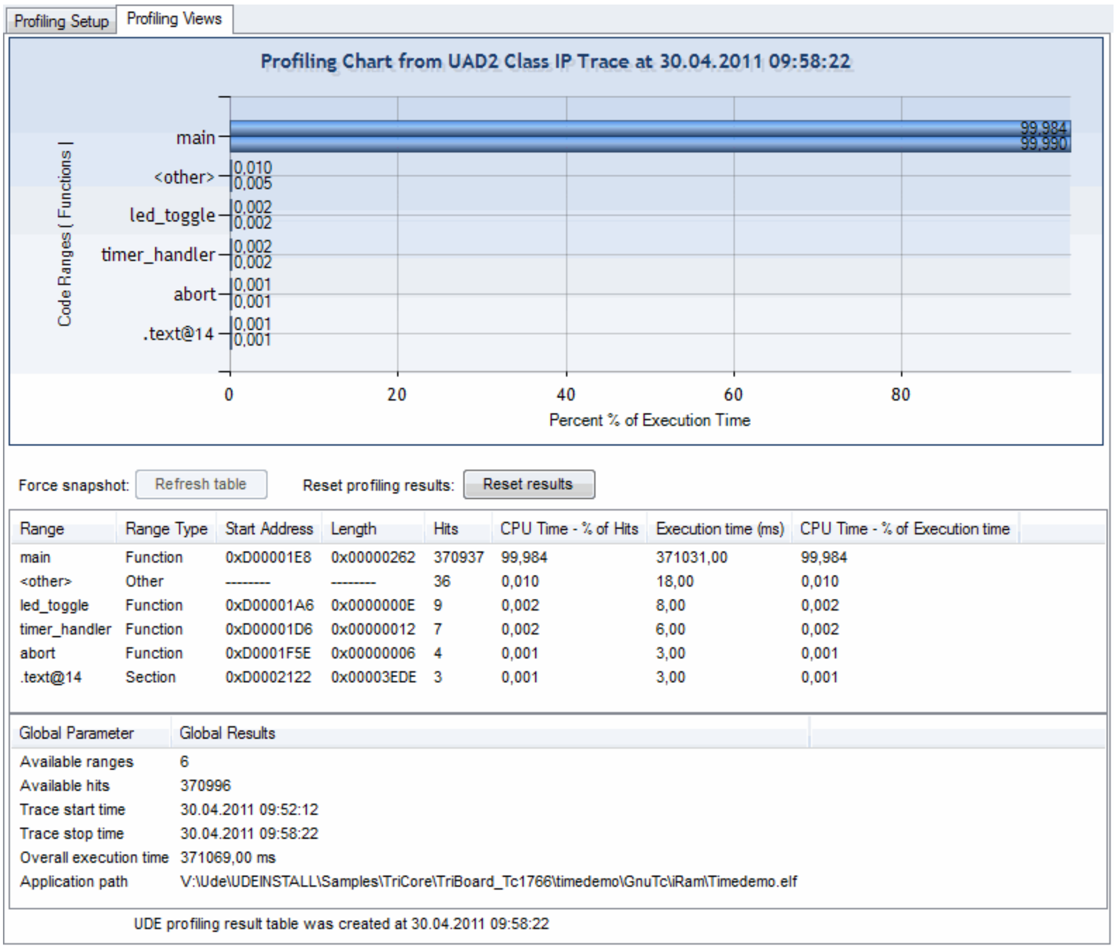
Profiling (stats)

The IP Profiling window displays profiling results of IP samples inside of functions of a running program. The input data will be recorded by periodical reads of the IP over debug channels by the UAD firmware.

Configuring the Profiling (stats) Window

A distribution statistic over polling results will be displayed as bar chart over all functions or sections (if no symbolic information about functions is found for the appropriate address), which are hit by periodical polling of the value of the IP. This method requires an architecture, which provides the capability to read the IP value periodically during program execution (e.g. Infineon AURIX, TriCore and XC2000/XE166 microcontrollers). Using the UAD2, a sampling period in the range of 1 millisecond (1 ms) is assured for snapshots of the actual content of the IP in a selectable period during running target program.

Open the Trace window by menu **Views – Profiling (stats)**.



Range	Range Type	Start Address	Length	Hits	CPU Time - % of Hits	Execution time (ms)	CPU Time - % of Execution time
main	Function	0xD00001E8	0x00000262	370937	99,984	371031,00	99,984
<other>	Other	--------	--------	36	0,010	18,00	0,010
led_toggle	Function	0xD00001A6	0x0000000E	9	0,002	8,00	0,002
timer_handler	Function	0xD00001D6	0x00000012	7	0,002	6,00	0,002
abort	Function	0xD0001F5E	0x00000006	4	0,001	3,00	0,001
.text@14	Section	0xD0002122	0x00003EDE	3	0,001	3,00	0,001

Global Parameter	Global Results
Available ranges	6
Available hits	370996
Trace start time	30.04.2011 09:52:12
Trace stop time	30.04.2011 09:58:22
Overall execution time	371069,00 ms
Application path	V:\Ude\UDEINSTALL\Samples\TriCore\TriBoard_Tc1766\timedemo\GnuTc\iRam\Timedemo.elf

UDE profiling result table was created at 30.04.2011 09:58:22

The chart displays the actual statistics of IP sampling results. The profiling method of periodical polling of the IP generates two kinds of results for each function or section range:

- Number of hits per function or section range and
- the approximated execution time of the function.

The execution time will be calculated from hits per function multiplied with the period between two samples. The **Profiling Setup** page enables setup of different parameters like the refresh period.

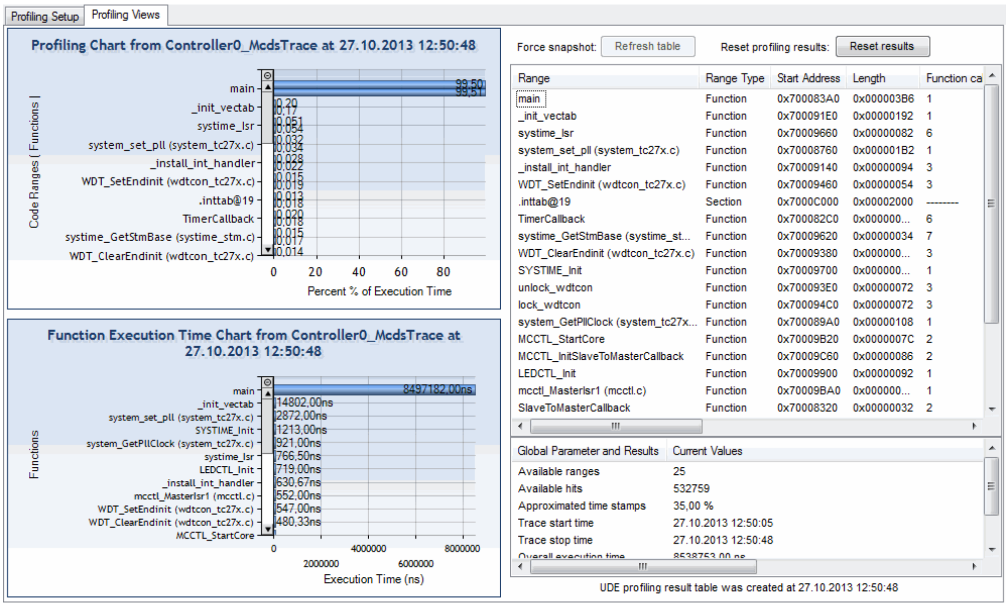
The profiling result table lists all details of IP profiling data set.

Profiling (trace)

The **Trace Profiling** window displays profiling results of IP samples inside of functions of a running program, which are recorded by code trace from **MCDS, NEXUS, CoreSight** or **ETM** trace channel. The full program trace enables measurement results of function execution time, which are much more exact than results from IP polling.

Use of Trace Profiling Window

The results of trace profiling will be displayed by the trace profiling window, using two bar charts (one for execution time of a function from overall execution time and one for the absolute execution time of a function) and a table, which provides all details of the profiling calculation. Open the trace profiling window by menu **Views – Trace & Analyze Windows – Profiling (trace)**. The window content will be updated, if the corresponding hardware trace stream was recorded and has decoded new trace data.



Range	Range Type	Start Address	Length	Function ca
main	Function	0x700083A0	0x000003B6	1
_init_vectab	Function	0x700091E0	0x00000192	1
systime_Isr	Function	0x70009660	0x00000082	6
system_set_pll (system_tc27x.c)	Function	0x70008760	0x000001B2	1
_install_int_handler	Function	0x70009140	0x00000094	3
WDT_SetEndinit (wdtcon_tc27x.c)	Function	0x70009460	0x00000054	3
.inttab@19	Section	0x7000C000	0x00002000	--------
TimerCallback	Function	0x700082C0	0x000000...	6
systime_GetStmBase (systime_st...	Function	0x70009620	0x00000034	7
WDT_ClearEndinit (wdtcon_tc27x.c)	Function	0x70009380	0x000000...	3
SYSTIME_Init	Function	0x70009700	0x000000...	1
unlock_wdtcon	Function	0x700093E0	0x00000072	3
lock_wdtcon	Function	0x700094C0	0x00000072	3
system_GetPllClock (system_tc27x...	Function	0x700089A0	0x00000108	1
MCCTL_StartCore	Function	0x70009B20	0x0000007C	2
MCCTL_InitSlaveToMasterCallback	Function	0x70009C60	0x00000086	2
LEDCTL_Init	Function	0x70009900	0x00000092	1
mcctl_MasterIsr1 (mcctl.c)	Function	0x70009BA0	0x000000...	1
SlaveToMasterCallback	Function	0x70008320	0x00000032	2

Global Parameter and Results	Current Values
Available ranges	25
Available hits	532759
Approximated time stamps	35,00 %
Trace start time	27.10.2013 12:50:05
Trace stop time	27.10.2013 12:50:48
Overall execution time	8538753,00 ns

UDE profiling result table was created at 27.10.2013 12:50:48

Configuring of full Program Trace

The measurement requires a suitable trace stream configuration, which depends on the kind of the trace channel. The trace configuration must ensure that the recorded trace contains program flow trace of the core, which is assigned to the **Trace Profiling** window.

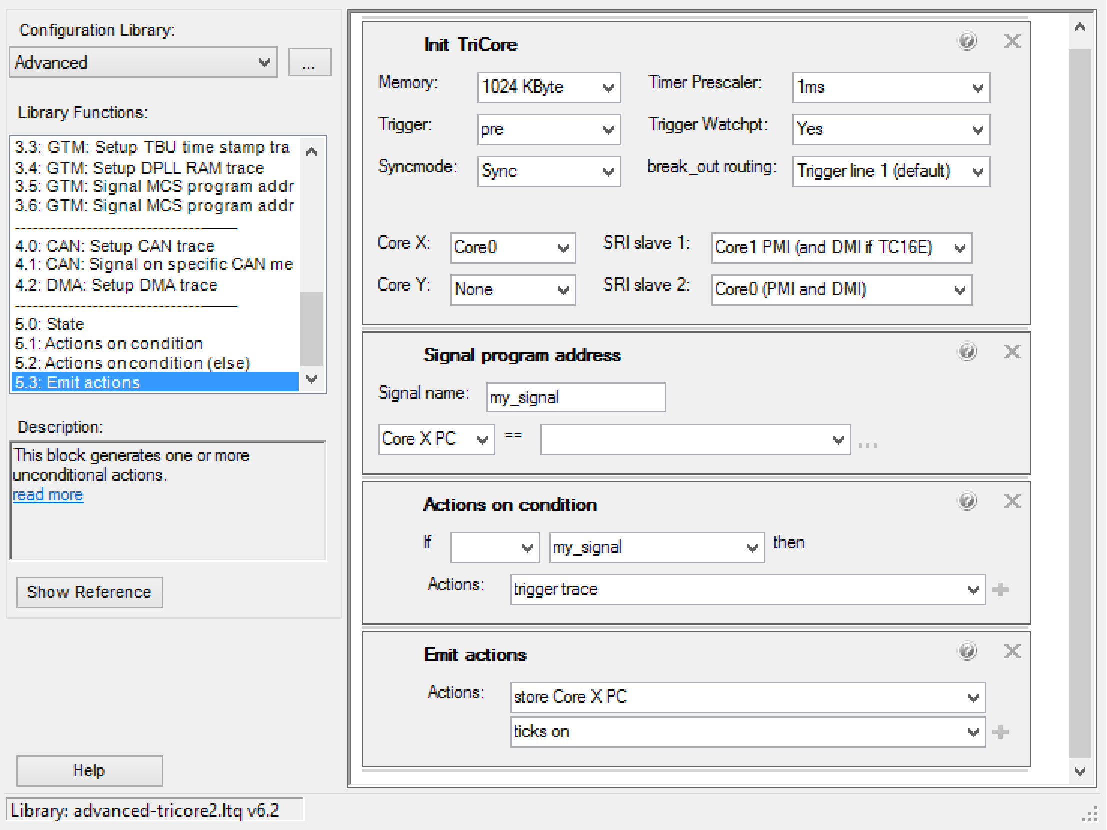
This example uses a TC27xxED with MCDS support and the TimeDemo.elf program from the example directory.

Open the UEC trace configuration window via menu **Tools** – **Configure Trace** to create a suitable trace configuration to record a program trace data with ticks of core 0 of TC27xx. Execute the following steps to create the required configuration:

- Switch to UEC configuration window.
- Switch from **Compact** to **Advanced Configuration** library.
- Drag the **Init TriCore** library element (1.0) to configuration area.
- Change **Memory Size** to maximum value of 1024 kByte.
- Change **Syncmode** to mode Sync to set the best tick resolution between traced code samples.
- Drag the **Signal program address** library element (2.1) to configuration area.
- Set **Signal name** to e.g. `my_signal`
- Select **main** function start address as comparison address for Core X PC
- Drag the **Actions on condition** library element (5.1) to configuration area.

- ➢ Browse `my_signal` as signal name for If condition.
- ➢ Select as action **Trigger trace**
- ➢ Drag the **Emit actions** library element (5.3) to configuration area.
- ➢ Add emit action **store Core X PC**
- ➢ Add emit action **ticks on**

The optimized MCDS trace configuration for this use case is now completed.



Configuration Library: Advanced

Library Functions:

3.3: GTM: Setup TBU time stamp tra
3.4: GTM: Setup DPLL RAM trace
3.5: GTM: Signal MCS program addr
3.6: GTM: Signal MCS program addr

4.0: CAN: Setup CAN trace
4.1: CAN: Signal on specific CAN me
4.2: DMA: Setup DMA trace

5.0: State
5.1: Actions on condition
5.2: Actions on condition (else)
5.3: Emit actions

Description:

This block generates one or more unconditional actions.
read more

Show Reference

Help

Library: advanced-tricore2.ltq v6.2

Init TriCore

Memory: 1024 KByte
Timer Prescaler: 1ms
Trigger: pre
Trigger Watchpt: Yes
Syncmode: Sync
break_out routing: Trigger line 1 (default)
Core X: Core0
SRI slave 1: Core1 PMI (and DMI if TC16E)
Core Y: None
SRI slave 2: Core0 (PMI and DMI)

Signal program address

Signal name: my_signal
Core X PC ==

Actions on condition

If my_signal then
Actions: trigger trace

Emit actions

Actions: store Core X PC
ticks on

It is recommended to enable automatic refresh of profiling data after end of trace (if 1 MByte on-chip trace memory is filled). Use the **Profiling Setup** page, group **Basic Profiling Settings**, and enable the check box **Enable automatic refresh of trace data**.

After configuring the trace, start trace recording (tool bar icon or menu **Tools – Start trace..** tool bar). After start of program execution, the **Progress** window displays two subsequent tasks:

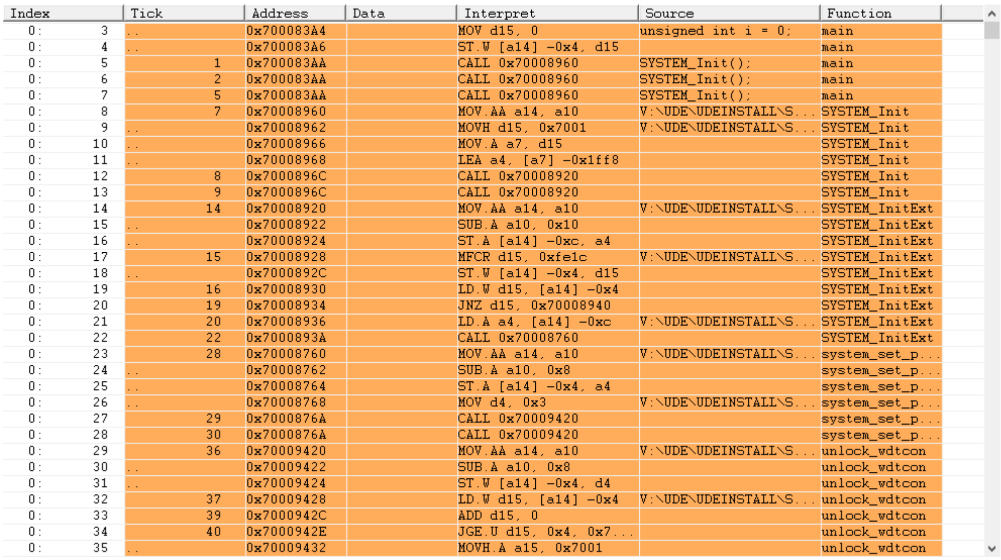
- ➢ Record trace stream to memory
- ➢ Decode trace data

After completion of the last task, the page **Profiling Views** of the **Code Trace Profiling** window will be updated with the results from the last trace recording. This method generates more detailed results than the measurement by IP polling. The results contain additional information:

- ➢ Number of recorded functions calls.
- ➢ The average function execution time.

The quality of measurement depends on the amount of recorded program addresses without tick information and therefore on the type of trace channel. The result table contains a summary (percent value) of **Approximate time stamps**, which allows an evaluation of the results precision. For all recorded trace samples without tick information, the appropriate ticks must be approximated based on the last and next known tick values.

To illustrate this, the following figure shows a Trace window with recorded program trace, which contains some code address samples without tick information.



Index	Tick	Address	Data	Interpret	Source	Function
0: 3	..	0x700083A4		MOV d15, 0	unsigned int i = 0;	main
0: 4	..	0x700083A6		ST.W [a14] -0x4, d15		main
0: 5	1	0x700083AA		CALL 0x70008960	SYSTEM_Init();	main
0: 6	2	0x700083AA		CALL 0x70008960	SYSTEM_Init();	main
0: 7	5	0x700083AA		CALL 0x70008960	SYSTEM_Init();	main
0: 8	7	0x70008960		MOV.AA a14, a10	V:\UDE\UDEINSTALL\S...	SYSTEM_Init
0: 9	..	0x70008962		MOVH d15, 0x7001	V:\UDE\UDEINSTALL\S...	SYSTEM_Init
0: 10	..	0x70008966		MOV.A a7, d15		SYSTEM_Init
0: 11	..	0x70008968		LEA a4, [a7] -0x1ff8		SYSTEM_Init
0: 12	8	0x7000896C		CALL 0x70008920		SYSTEM_Init
0: 13	9	0x7000896C		CALL 0x70008920		SYSTEM_Init
0: 14	14	0x70008920		MOV.AA a14, a10	V:\UDE\UDEINSTALL\S...	SYSTEM_InitExt
0: 15	..	0x70008922		SUB.A a10, 0x10		SYSTEM_InitExt
0: 16	..	0x70008924		ST.A [a14] -0xc, a4		SYSTEM_InitExt
0: 17	15	0x70008928		MFCR d15, 0xfe1c	V:\UDE\UDEINSTALL\S...	SYSTEM_InitExt
0: 18	..	0x7000892C		ST.W [a14] -0x4, d15		SYSTEM_InitExt
0: 19	16	0x70008930		LD.W d15, [a14] -0x4		SYSTEM_InitExt
0: 20	19	0x70008934		JNZ d15, 0x70008940		SYSTEM_InitExt
0: 21	20	0x70008936		LD.A a4, [a14] -0xc	V:\UDE\UDEINSTALL\S...	SYSTEM_InitExt
0: 22	22	0x7000893A		CALL 0x70008760		SYSTEM_InitExt
0: 23	28	0x70008760		MOV.AA a14, a10	V:\UDE\UDEINSTALL\S...	system_set_p...
0: 24	..	0x70008762		SUB.A a10, 0x8		system_set_p...
0: 25	..	0x70008764		ST.A [a14] -0x4, a4		system_set_p...
0: 26	..	0x70008768		MOV d4, 0x3	V:\UDE\UDEINSTALL\S...	system_set_p...
0: 27	29	0x7000876A		CALL 0x70009420		system_set_p...
0: 28	30	0x7000876A		CALL 0x70009420		system_set_p...
0: 29	36	0x70009420		MOV.AA a14, a10	V:\UDE\UDEINSTALL\S...	unlock_wdtcon
0: 30	..	0x70009422		SUB.A a10, 0x8		unlock_wdtcon
0: 31	..	0x70009424		ST.W [a14] -0x4, d4		unlock_wdtcon
0: 32	37	0x70009428		LD.W d15, [a14] -0x4	V:\UDE\UDEINSTALL\S...	unlock_wdtcon
0: 33	39	0x7000942C		ADD d15, 0		unlock_wdtcon
0: 34	40	0x7000942E		JGE.U d15, 0x4, 0x7...		unlock_wdtcon
0: 35	..	0x70009432		MOVH.A a15, 0x7001		unlock_wdtcon

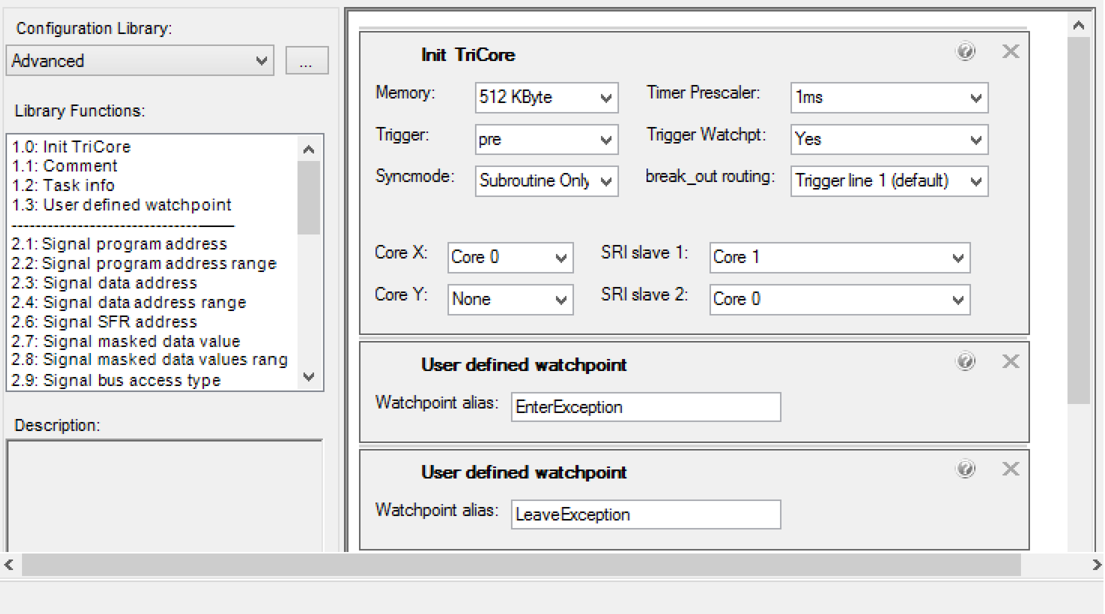
Configuring of MCDS Compact Function Trace

The MCDS trace of Infineon TriCore, AURIX supports an additional code trace mode, which is named **Compact Function Trace**. This mode records only the call and return instructions. Therefore, this mode provides important advantages over normal program full trace:

- ➢ It reduces the consumption of trace memory in comparison to full program trace, therefore the available memory can store trace data for a longer period (typically ten time longer and more).
- ➢ Each recorded function call and return instruction has an assigned own tick value, therefore the precision of the measured function execution times will be increased.

Various compilers perform a sibling/tail call optimization that prevents the Compact Function Trace from recognizing call and return instructions. The Compact Function Trace does not work under these conditions.

The following two figures show the complete UEC configuration for a profiling task using compact function trace.


Configuration Library:
...
Library Functions:
1.0: Init TriCore
1.1: Comment
1.2: Task info
1.3: User defined watchpoint
2.1: Signal program address
2.2: Signal program address range
2.3: Signal data address
2.4: Signal data address range
2.6: Signal SFR address
2.7: Signal masked data value
2.8: Signal masked data values rang
2.9: Signal bus access type
Description:

Configuration Library: Advanced

Init TriCore

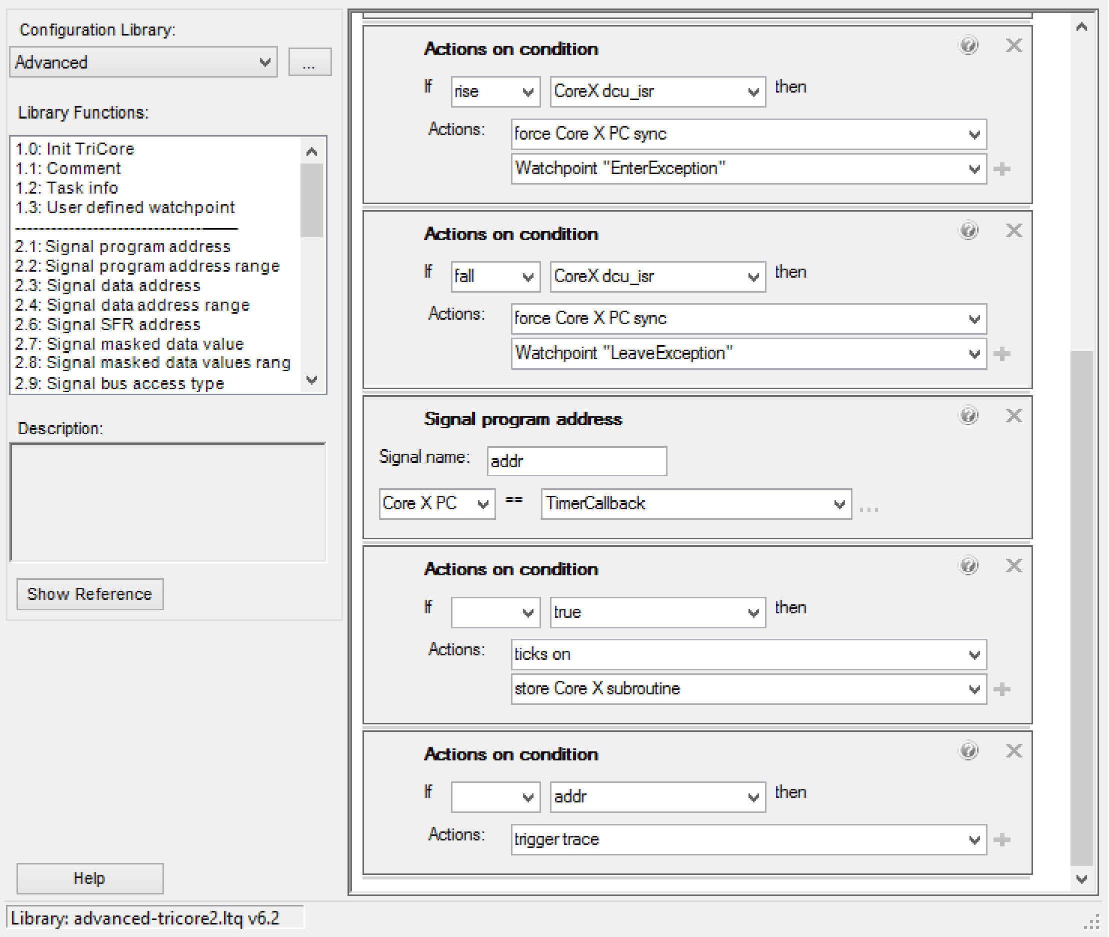
Memory: 512 KByte
Timer Prescaler: 1ms
Trigger: pre
Trigger Watchpt: Yes
Syncmode: Subroutine Only
break_out routing: Trigger line 1 (default)
Core X: Core 0
SRI slave 1: Core 1
Core Y: None
SRI slave 2: Core 0

User defined watchpoint

Watchpoint alias: EnterException

User defined watchpoint

Watchpoint alias: LeaveException



Configuration Library: Advanced

Library Functions:

1.0: Init TriCore
1.1: Comment
1.2: Task info
1.3: User defined watchpoint

2.1: Signal program address
2.2: Signal program address range
2.3: Signal data address
2.4: Signal data address range
2.6: Signal SFR address
2.7: Signal masked data value
2.8: Signal masked data values rang
2.9: Signal bus access type

Description:

Library: advanced-tricore2.ltq v6.2

Actions on condition
If rise CoreX dcu_isr then
Actions: force Core X PC sync
Watchpoint "EnterException"

Actions on condition
If fall CoreX dcu_isr then
Actions: force Core X PC sync
Watchpoint "LeaveException"

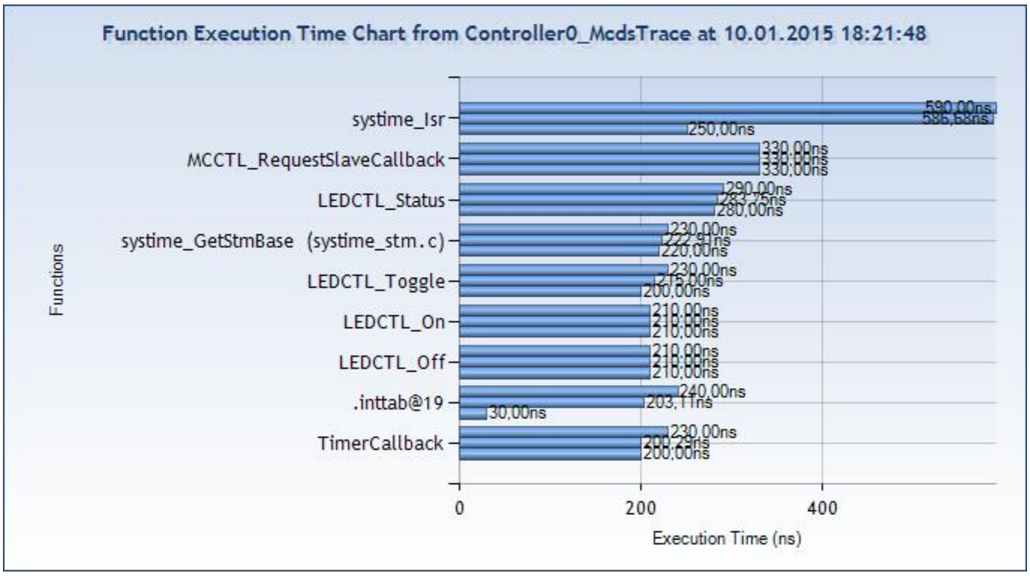
Signal program address
Signal name: addr
Core X PC == TimerCallback

Actions on condition
If true then
Actions: ticks on
store Core X subroutine

Actions on condition
If addr then
Actions: trigger trace

The profiling data analysis detects the compact function trace mode automatically from trace configuration. Due to the enhanced accuracy and the longer time of trace recording, the result of the compact function trace profiling analysis provides two additional important results:

- Minimum function execution time
- Maximum function execution time



Code Coverage

UDE supports the calculation of code coverage by analyzing recorded program trace data of **MCDS, NEXUS, CoreSight and ETM** trace channels:

- **Statement Coverage of Machine Code –** This method calculates code coverage data for statement coverage based on information of executed machine instructions. The results of statement coverage are available in the **Program** window too.
- **Branch Coverage of Machine Code –** This method calculates code coverage data for branch coverage based on information of executed machine branch instructions.
- **Branch Coverage of Control Flow –** This method additionally uses program flow information from compiler symbol information to determine the successors of indirect branches, which are not available for trace-only based branch coverage. This mode requires the availability of the additional compiler option to generate the debug information about the program control flow. This mode generates coverage results, which are comparable to the coverage results generated by common known GNU compiler `gcov-option`.

The **Code Coverage** window displays all results of statement and branch coverage. The calculation requires additional information about the program flow from compiler provided symbol information.

Preparing the Trace Configuration

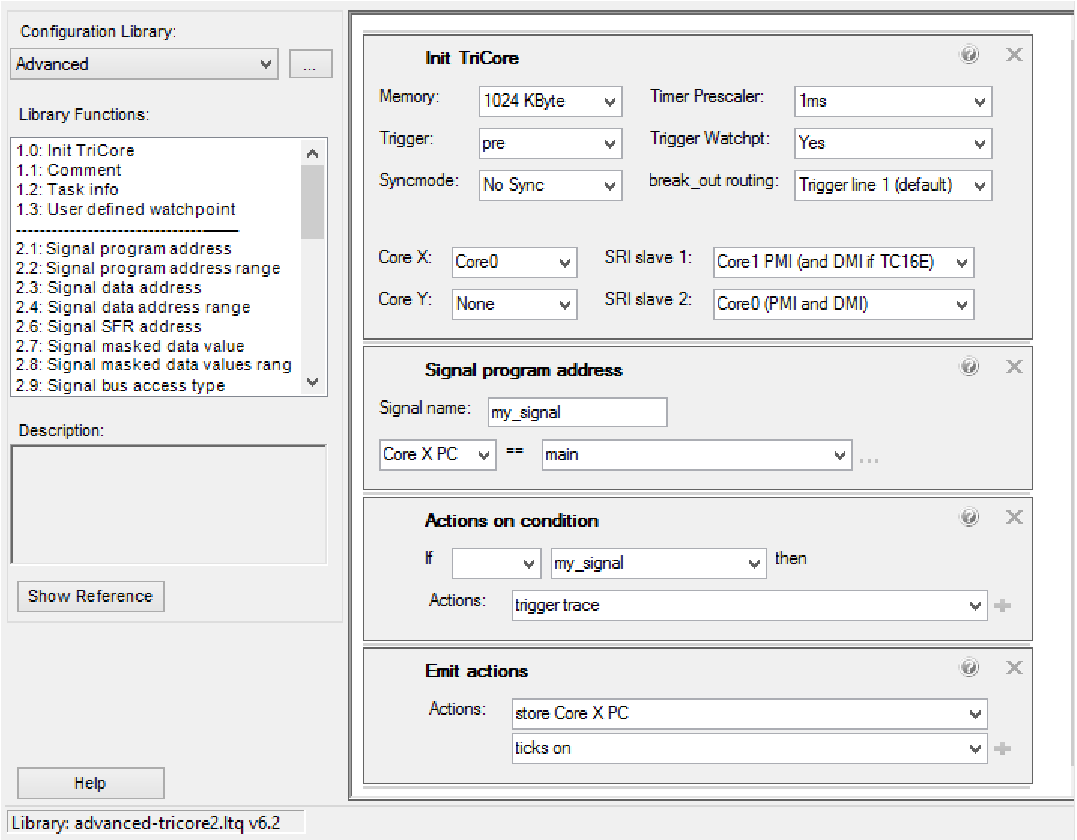
The code coverage measurement requires a suitable trace stream configuration, which depends on the kind of the trace channel. The trace configuration must ensure that the recorded trace contains program flow trace of the core, which is assigned to the **Code Coverage** window.

This example uses a TC27xxED with MCDS support and the `TimeDemo.elf` program from the example directory.

Open the UEC trace configuration window via menu **Tools** – **Configure Trace** to create a suitable trace configuration to record program trace data of core 0 of TC27xx. The code coverage algorithm uses these trace data to calculate code coverage results. The optimized code coverage configuration need not contain additional time information (compared with profiling configuration), which saves trace memory space. Execute the following steps to create the required configuration:

- Switch **to UEC configuration window**.
- Switch from **Compact** to **Advanced Configuration** library.
- Drag the **Init TriCore** library element (1.0) to configuration area.
- Change **Memory Size** to maximum value of 1024 kByte
- Drag the **Signal program address** library element (2.1) to configuration area.
- Set **Signal name** to e.g. `my_signal`
- Select `main` function start address as comparison address for Core X PC
- Drag the **Actions on condition** library element (5.1) to configuration area.
- Browse `my_signal` as signal name for If condition.
- Select as action **Trigger trace**
- Drag the **Emit actions** library element (5.3) to configuration area.
- Add emit action **store Core X PC**

The optimized MCDS trace configuration for this use-case is now completed.



Configuration Library: Advanced

Library Functions:

- 1.0: Init TriCore
- 1.1: Comment
- 1.2: Task info
- 1.3: User defined watchpoint
- ---------------------------------------
- 2.1: Signal program address
- 2.2: Signal program address range
- 2.3: Signal data address
- 2.4: Signal data address range
- 2.6: Signal SFR address
- 2.7: Signal masked data value
- 2.8: Signal masked data values rang
- 2.9: Signal bus access type

Description:

Library: advanced-tricore2.ltq v6.2

Init TriCore

Memory:	1024 KByte	Timer Prescaler:	1ms
Trigger:	pre	Trigger Watchpt:	Yes
Syncmode:	No Sync	break_out routing:	Trigger line 1 (default)
Core X:	Core0	SRI slave 1:	Core1 PMI (and DMI if TC16E)
Core Y:	None	SRI slave 2:	Core0 (PMI and DMI)

Signal program address

Signal name: my_signal

Core X PC == main

Actions on condition

If my_signal then

Actions: trigger trace

Emit actions

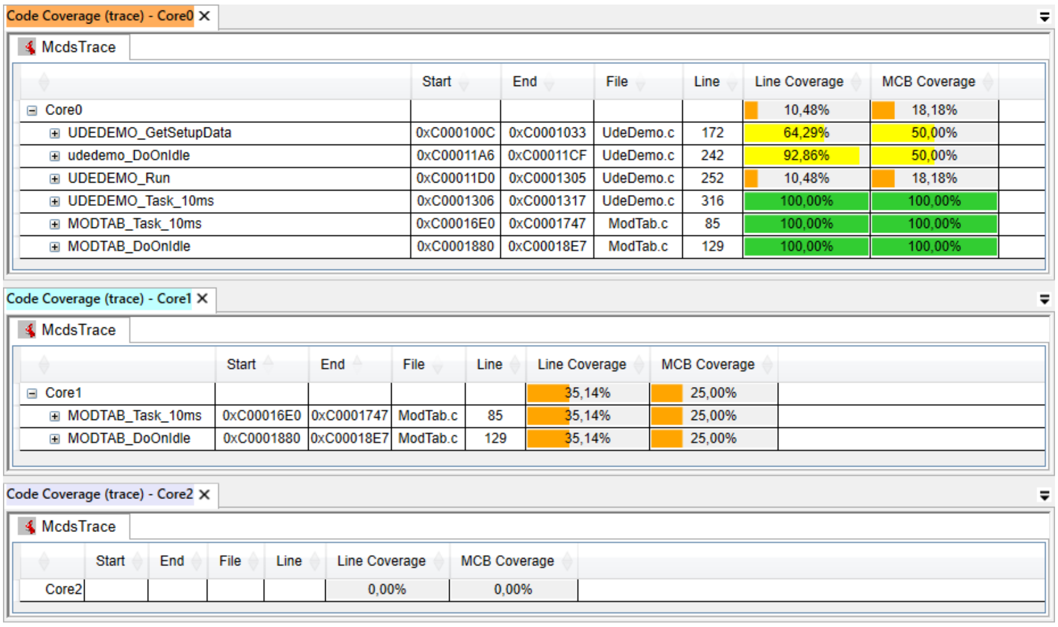
Actions: store Core X PC

ticks on

Use of Code Coverage Window to evaluate the Coverage Results

The trace coverage window displays all results of trace coverage in detail. Open the trace coverage window by menu **Views – Trace & Analyze Windows – Code Coverage (trace)**.

To update the window content after recording the trace data, use the context menu entry “**Start**” for running the analysis. If the entry is grayed out, the analysis will start immediately, after the trace stream stopped recording. The window displays the accumulated coverage data from all functions covered by the trace data.



Code Coverage (trace) - Core0

McdsTrace

	Start	End	File	Line	Line Coverage	MCB Coverage
⊟ Core0					10,48%	18,18%
⊞ UDEDEMO_GetSetupData	0xC000100C	0xC0001033	UdeDemo.c	172	64,29%	50,00%
⊞ udedemo_DoOnIdle	0xC00011A6	0xC00011CF	UdeDemo.c	242	92,86%	50,00%
⊞ UDEDEMO_Run	0xC00011D0	0xC0001305	UdeDemo.c	252	10,48%	18,18%
⊞ UDEDEMO_Task_10ms	0xC0001306	0xC0001317	UdeDemo.c	316	100,00%	100,00%
⊞ MODTAB_Task_10ms	0xC00016E0	0xC0001747	ModTab.c	85	100,00%	100,00%
⊞ MODTAB_DoOnIdle	0xC0001880	0xC00018E7	ModTab.c	129	100,00%	100,00%

Code Coverage (trace) - Core1

McdsTrace

	Start	End	File	Line	Line Coverage	MCB Coverage
⊟ Core1					35,14%	25,00%
⊞ MODTAB_Task_10ms	0xC00016E0	0xC0001747	ModTab.c	85	35,14%	25,00%
⊞ MODTAB_DoOnIdle	0xC0001880	0xC00018E7	ModTab.c	129	35,14%	25,00%

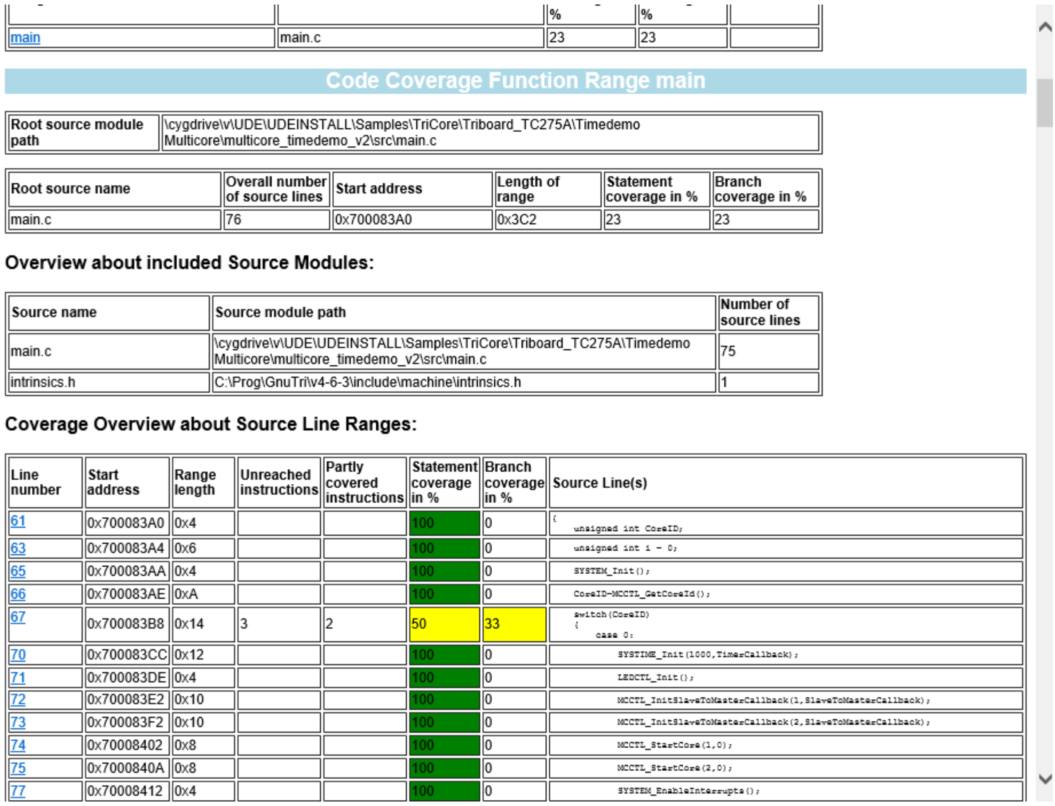
Code Coverage (trace) - Core2

McdsTrace

	Start	End	File	Line	Line Coverage	MCB Coverage
Core2					0,00%	0,00%

The window can be opened individually for each core and, by disabling the context menu entry “**Show all Cores**”, provide coverage data for only that core.

The **Code Coverage window** supports the creation of HTML reports, which contain all details of code coverage measurement of one or more functions. These reports are intended for documentation of the complete process of software qualification. These reports can be generated manually via context menu entry “Create Report” or automatically after completion of coverage analysis. The latter requires configuration of the output name and path for the report files in the code coverage setup page. See the online help of the code coverage windows for details.



		%	%	
main	main.c	23	23	

Code Coverage Function Range main

Root source module path	\cygdrive\v\UDE\UDEINSTALL\Samples\TriCore\Triboard_TC275A\Timedemo Multicore\multicore_timedemo_v2\src\main.c

Root source name	Overall number of source lines	Start address	Length of range	Statement coverage in %	Branch coverage in %
main.c	76	0x700083A0	0x3C2	23	23

Overview about included Source Modules:

Source name	Source module path	Number of source lines
main.c	\cygdrive\v\UDE\UDEINSTALL\Samples\TriCore\Triboard_TC275A\Timedemo Multicore\multicore_timedemo_v2\src\main.c	75
intrinsics.h	C:\Prog\GnuTri\v4-6-3\include\machine\intrinsics.h	1

Coverage Overview about Source Line Ranges:

Line number	Start address	Range length	Unreached instructions	Partly covered instructions	Statement coverage in %	Branch coverage in %	Source Line(s)

61	0x700083A0	0x4			100	0	{
							unsigned int CoreID;

63	0x700083A4	0x6			100	0	unsigned int i - 0;
65	0x700083AA	0x4			100	0	SYSTEM_Init();
66	0x700083AE	0xA			100	0	CoreID-MCCTL_GetCoreId();

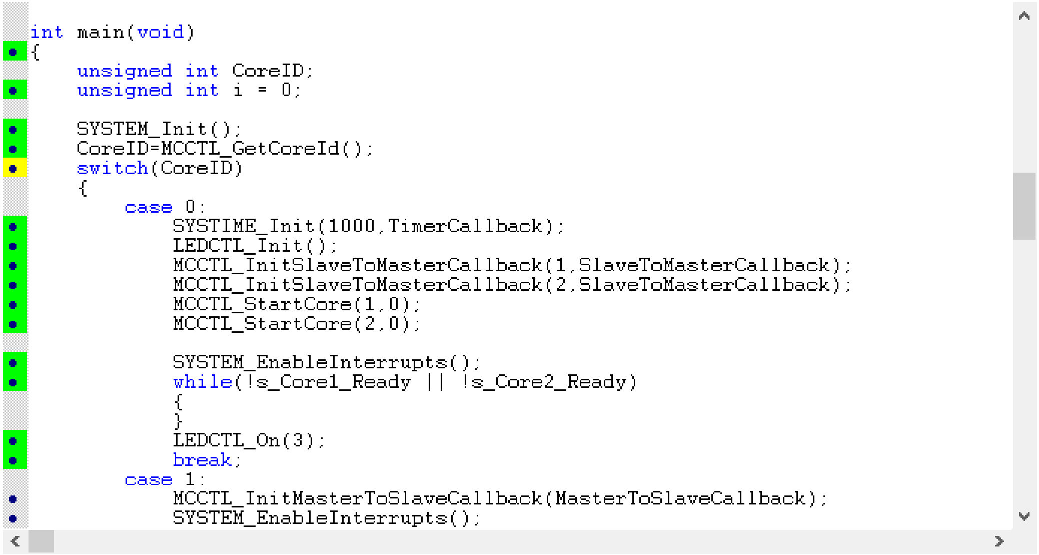
67	0x700083B8	0x14	3	2	50	33	switch(CoreID)
							{
							case 0:

70	0x700083CC	0x12			100	0	SYSTIME_Init(1000,TimerCallback);
71	0x700083DE	0x4			100	0	LEDCTL_Init();
72	0x700083E2	0x10			100	0	MCCTL_InitSlaveToMasterCallback(1,SlaveToMasterCallback);
73	0x700083F2	0x10			100	0	MCCTL_InitSlaveToMasterCallback(2,SlaveToMasterCallback);
74	0x70008402	0x8			100	0	MCCTL_StartCore(1,0);
75	0x7000840A	0x8			100	0	MCCTL_StartCore(2,0);
77	0x70008412	0x4			100	0	SYSTEM_EnableInterrupts();

Use of Program Windows to evaluate the Statement Coverage Results

The **Program** window provides basic information about statement coverage:

➢



```
int main(void)
{
    unsigned int CoreID;
    unsigned int i = 0;

    SYSTEM_Init();
    CoreID=MCCTL_GetCoreId();
    switch(CoreID)
    {
        case 0:
            SYSTIME_Init(1000,TimerCallback);
            LEDCTL_Init();
            MCCTL_InitSlaveToMasterCallback(1,SlaveToMasterCallback);
            MCCTL_InitSlaveToMasterCallback(2,SlaveToMasterCallback);
            MCCTL_StartCore(1,0);
            MCCTL_StartCore(2,0);

            SYSTEM_EnableInterrupts();
            while(!s_Core1_Ready || !s_Core2_Ready)
            {
            }
            LEDCTL_On(3);
            break;
        case 1:
            MCCTL_InitMasterToSlaveCallback(MasterToSlaveCallback);
            SYSTEM_EnableInterrupts();
```

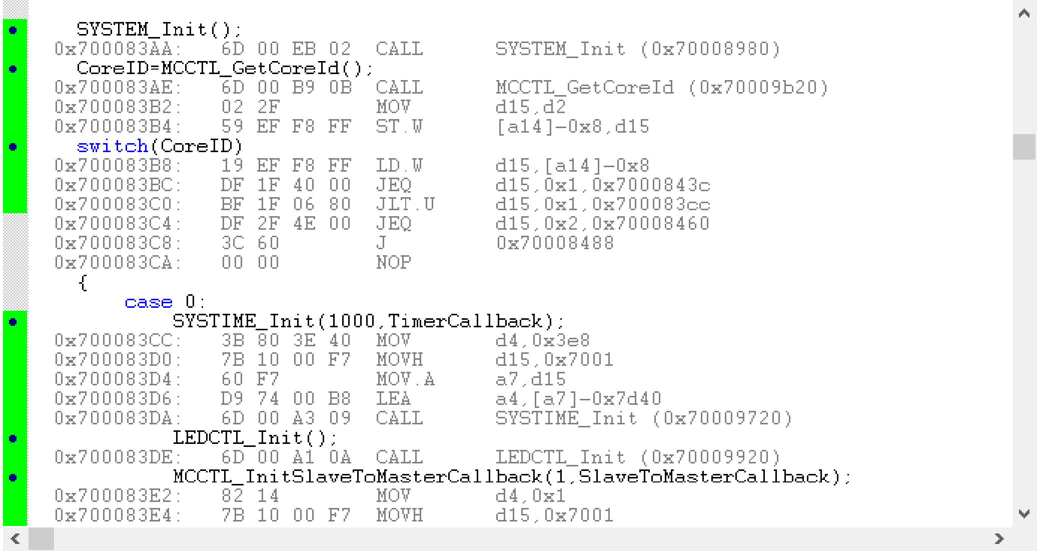
The background color of the left column of the **Program** window is used to visualize, whether the appropriate source line or instruction was covered by the latest code coverage measurement.

If source mode is active,

- the green background color indicates, that all machine instructions of the source line have been covered,
- the yellow background color indicates, that only a subset of the machine instructions of the source line have been covered,
- the default background color indicates that none of the machine instructions of the source line have been covered.

If mixed- or disassembly mode is active,

- the green background color indicates, that the machine instruction was executed,
- the default background color indicates that the machine instruction was not executed.



```
    SYSTEM_Init();
  0x700083AA:   6D 00 EB 02  CALL      SYSTEM_Init (0x70008980)
    CoreID=MCCTL_GetCoreId();
  0x700083AE:   6D 00 B9 0B  CALL      MCCTL_GetCoreId (0x70009b20)
  0x700083B2:   02 2F        MOV       d15,d2
  0x700083B4:   59 EF F8 FF  ST.W      [a14]-0x8,d15
    switch(CoreID)
  0x700083B8:   19 EF F8 FF  LD.W      d15,[a14]-0x8
  0x700083BC:   DF 1F 40 00  JEQ       d15,0x1,0x7000843c
  0x700083C0:   BF 1F 06 80  JLT.U     d15,0x1,0x700083cc
  0x700083C4:   DF 2F 4E 00  JEQ       d15,0x2,0x70008460
  0x700083C8:   3C 60        J         0x70008488
  0x700083CA:   00 00        NOP
    {
        case 0:
            SYSTIME_Init(1000,TimerCallback);
  0x700083CC:   3B 80 3E 40  MOV       d4,0x3e8
  0x700083D0:   7B 10 00 F7  MOVH      d15,0x7001
  0x700083D4:   60 F7        MOV.A     a7,d15
  0x700083D6:   D9 74 00 B8  LEA       a4,[a7]-0x7d40
  0x700083DA:   6D 00 A3 09  CALL      SYSTIME_Init (0x70009720)
            LEDCTL_Init();
  0x700083DE:   6D 00 A1 0A  CALL      LEDCTL_Init (0x70009920)
            MCCTL_InitSlaveToMasterCallback(1,SlaveToMasterCallback);
  0x700083E2:   82 14        MOV       d4,0x1
  0x700083E4:   7B 10 00 F7  MOVH      d15,0x7001
```

Execution Sequence Chart

The Execution Sequence Analysis analyses the program-flow on the task- and function-level based on trace data. The results are visualized, using the Execution Sequence Chart. This feature makes it easier to visualize and evaluate target data to accelerate the verification of complex software algorithms and input from process environment. The Execution Sequence Analysis view can be used with all targets, which provide a hardware trace channel for **MCDS, NEXUS, CoreSight, ETM** or **ETB** trace.

Features of the Execution Sequence Chart

The Execution Sequence Chart (formerly Function Sequence Chart) displays the execution of the program on the task- and function-level based on trace data. The execution sequence is displayed separately for each core, but using a common timeline to observe the parallel execution.

Note, that task information is only available if a separately available UDE add-in component provides it. The add-in components “µC/OS-II Awareness”, “OSEK Awareness”, “PXROS-HR Awareness” and “SAFERTOS/FreeRTOS Awareness” provide these Information. The “OSEK Awareness" add-in uses information from an ORTI (OSEK Real-Time Interface) file to identify the running task. See the manual for this add-in for further information.

The execution sequence is displayed as a Gantt chart. The tree view on the left side can be used to show or hide the execution on specific cores and tasks. The bars on the right side show which function is executed at which time. During the execution of sub-functions, the bar of the parent function has a gap. Vertical dashed grey lines show events within the trace stream. The tool-tip (mouse over) of such a line will provide a short description. See section Event Marker for a more detailed explanation.

Configuring of Trace Configuration

To use the Execution Sequence Chart, a trace configuration with the following properties is required:

- Program Trace for (at least) each executed call and return instruction and
- Ticks enabled, to calculate the relative time for the execution.

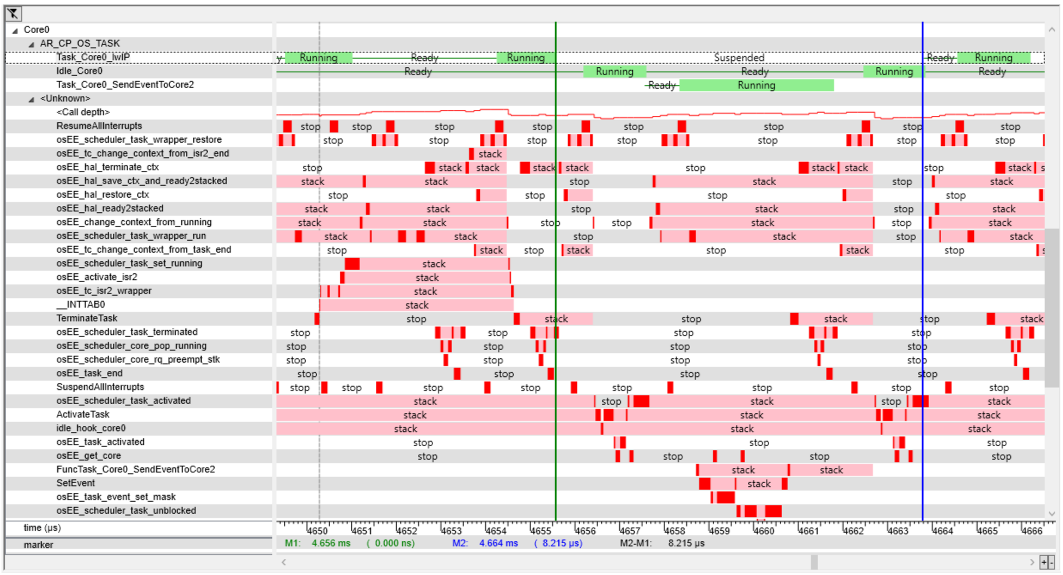
Therefore, the same trace configurations, which were described in chapter **Trace Profiling, Configuring of full Program Trace** and chapter **Trace Profiling, Configuring of MCDS Compact Function Trace,** can be used.

Use of Execution Sequence Chart

Open the chart by menu **Views – Execution Sequence Chart**. Before any data can be displayed, the Execution Sequence Analysis must be started. Depending on the size and complexity of the trace stream, the analysis may take a lot of time. Thus, the analysis must be started manually, even if a new trace stream has been captured. To start the analysis, record a trace first or load a persistent trace stream. Then right-click within the empty area of the chart, to open the context menu, and choose **Start analysis**.

The analysis is started as a background task (see **Progress Window**). After it is completed, the chart is updated automatically.

The tree can be filtered on a per-item basis. First, select the item that (direct) sub-items you want to filter. Then input the filter string into the text-box on the upper-left corner.



Event Marker

The following events may be displayed within the Gantt chart using vertical dashed grey lines.

- **Trace Broken** - The trace could not be fully decoded due to special circumstances. For example, if the on-chip trace hardware of the target has omitted some required information due to its limitations (buffer overflows, etc.). See messages in Trace Window at respective time and earlier.

The blue and green marker can be set similar to bookmarks and allow time measurement in between.

Data Trace Chart

This UDE window displays all data-access trace records of a core as data series of a 2-dimensional scientific diagram. This feature makes it easier to detect time-related variable access problems and to accelerate the verification of complex software algorithms and input from process environment. The Scientific Chart view can be used with all targets, which provide a hardware trace channel for **MCDS, NEXUS, CoreSight, ETM** or **ETB** trace and data-access trace.

Features of the Data Trace Chart

A **Find** function is available as enhancement of the search- and zoom-function to find specific trace samples and zoom into specific sample ranges. For that, the chart has to switch into **Zoom** mode via context menu.

The pre-processing algorithm, which analyses the data-access trace records of the last recorded stream, automatically detects the high-level variables, which are modified or read by the core. Therefore, the trace has only to be configured for filtering of the relevant data accesses.

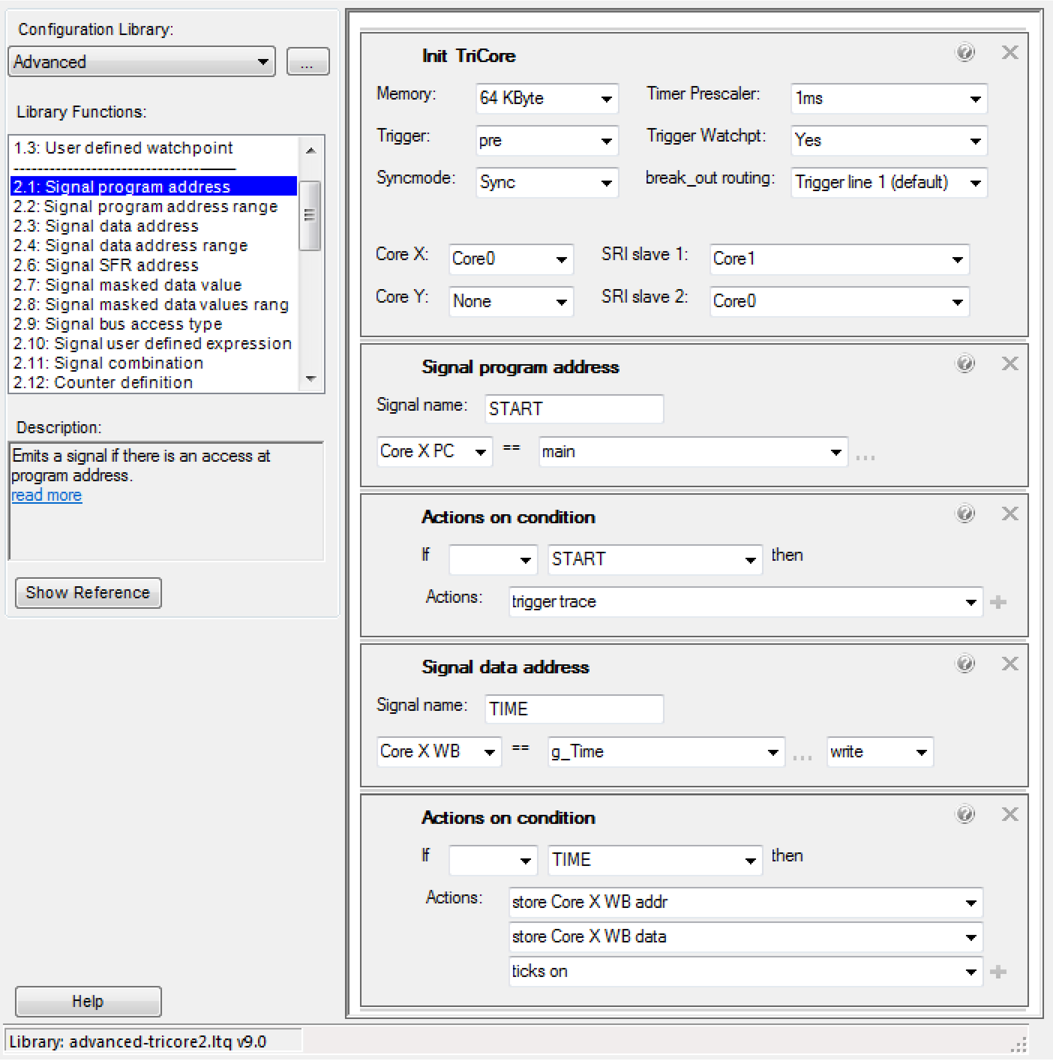
Configuring of Trace Configuration

This example uses a TC27xxED with MCDS support and the `TimeDemo.elf` program from the example directory.

Open the UEC trace configuration window via menu **Tools** – **Configure Trace** to create a suitable trace configuration to record write-access trace from **Core0** to the global variable `g_Time` with timestamps. The data trace chart pre-processing algorithm assigns the address-based data-write trace message to the original variable names automatically. Execute the following steps to create the required configuration:

1. Switch to UEC configuration window, switch from **Compact** to **Advanced Configuration** library.
2. Drag the **Init TriCore** library element (1.0) to configuration area.
3. Change **Syncmode** to mode Sync to set the best tick resolution between traced data samples
4. Drag the **Signal program address** library element (2.2) to configuration area.
5. Set **Signal name** to e.g. `START`
6. Select **main** function as start address for Core X PC
7. Drag the **Actions on condition** library element (5.1) to configuration area.
8. Browse `START` as signal name for If condition.
9. Select as action **Trigger trace**.
10. Drag the **Signal data address** library element (2.3) to configuration area
11. Assign `TIME` as signal name of this signal.
12. Select the variable `g_time` from watch browse dialog as signal address.
13. Drag the **Actions on condition** library element (5.1) to configuration area.
14. Browse `TIME` as signal name for If condition.
15. Select as action **store Core X WB addr**, add further action **store Core X WB data**.
16. Add further action **ticks on**.

The optimized MCDS trace configuration for this use-case is now completed. It is recommended to enable the automatic refresh of trace data after the end of data trace (if 1Mbyte on-chip trace memory is filled) by **Code Trace Use – Cases Property Page** of UDE configuration dialog (menu **Config – Debug Server Configuration**”), setting the check mark in **Enable automatic refresh of trace data**.



Configuration Library: Advanced

Library Functions:

- 1.3: User defined watchpoint
- 2.1: Signal program address
- 2.2: Signal program address range
- 2.3: Signal data address
- 2.4: Signal data address range
- 2.6: Signal SFR address
- 2.7: Signal masked data value
- 2.8: Signal masked data values rang
- 2.9: Signal bus access type
- 2.10: Signal user defined expression
- 2.11: Signal combination
- 2.12: Counter definition

Description:

Emits a signal if there is an access at program address.
read more

Show Reference

Help

Library: advanced-tricore2.ltq v9.0

Init TriCore

Memory: 64 KByte
Timer Prescaler: 1ms
Trigger: pre
Trigger Watchpt: Yes
Syncmode: Sync
break_out routing: Trigger line 1 (default)
Core X: Core0
SRI slave 1: Core1
Core Y: None
SRI slave 2: Core0

Signal program address

Signal name: START
Core X PC == main

Actions on condition

If START then
Actions: trigger trace

Signal data address

Signal name: TIME
Core X WB == g_Time write

Actions on condition

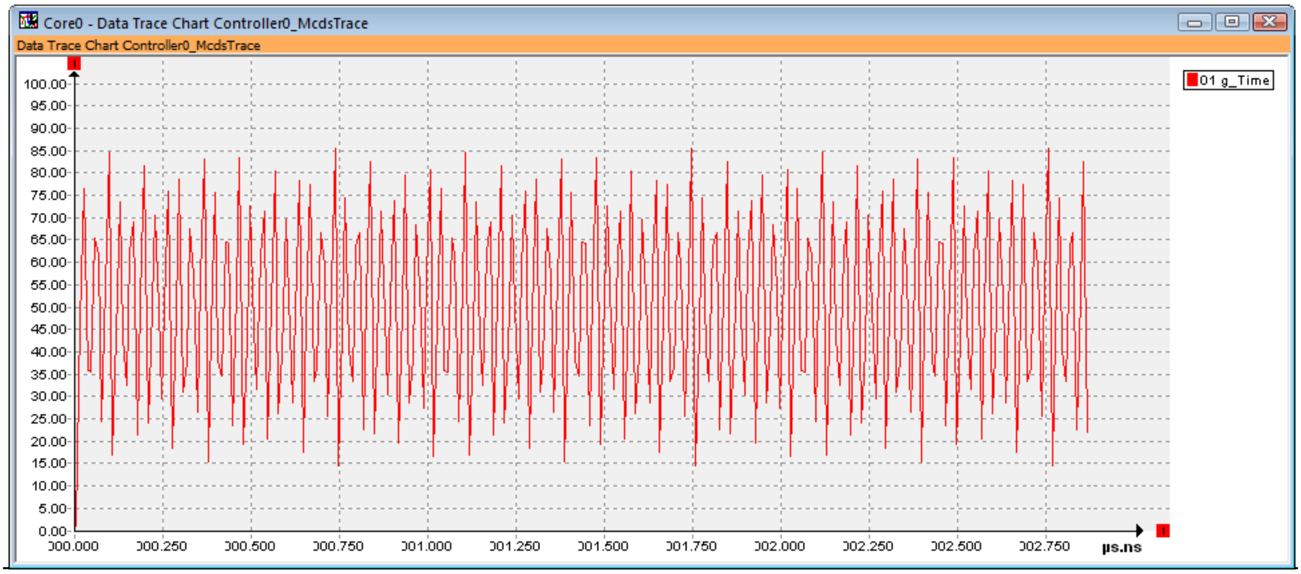
If TIME then
Actions: store Core X WB addr
store Core X WB data
ticks on

Usage of Data Trace Charts

Open the Data Trace Chart by menu **Views – Trace & Analyze Windows – Data Trace Chart**.

Now you can start trace (tool bar icon or menu **Tools – Start trace..**). After start of program execution, the **Progress** window displays two subsequent tasks:

- Record trace stream to memory
- Decode trace data



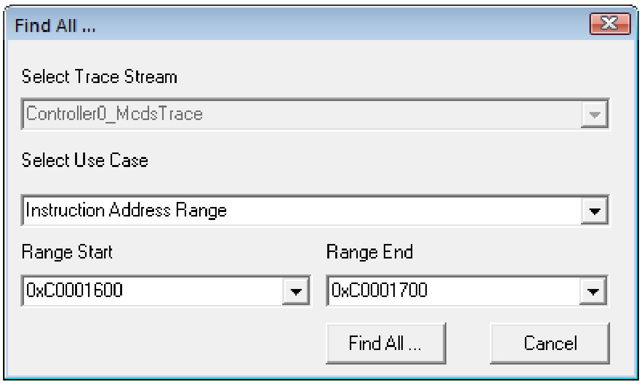
After completion of the last task, the **Data Trace Chart** window will be updated with the results from last run of variable access trace.

Find All in Trace

The **Find All** feature is used to search a trace stream for trace messages with a specified property or combination of properties. The result is displayed in the **Trace** Window as marker at the left side of the window and in a separate **Marker** window. The **Find All** dialog can be opened from the **Trace** windows context menu, by selecting the **Find All ...** menu entry. The Marker window is opened via menu **Views – Trace & Analyze Windows – Markers**.

The **Find All** dialog allows selection of several predefined configurations and parameters used for the search:

- ➢ Single Instruction execution at specified addresses.
- ➢ Data accesses (read/write) at specified addresses and/or with given values.
- ➢ Code execution in specified functions.
- ➢ Code sequences between a given start and end address.



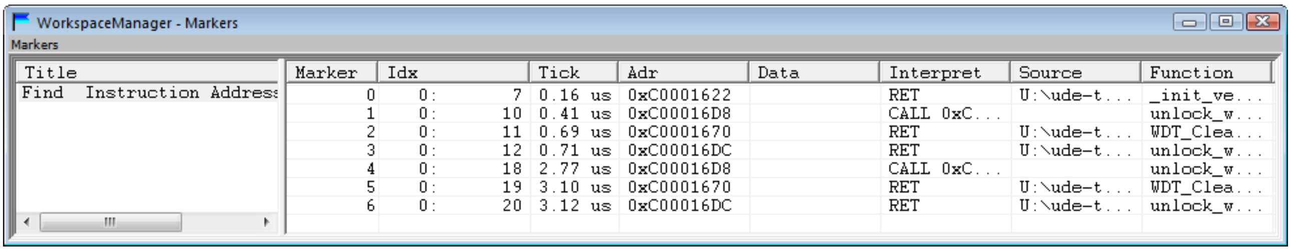
Select Trace Stream: Controller0_McdsTrace

Select Use Case: Instruction Address Range

Range Start: 0xC0001600

Range End: 0xC0001700

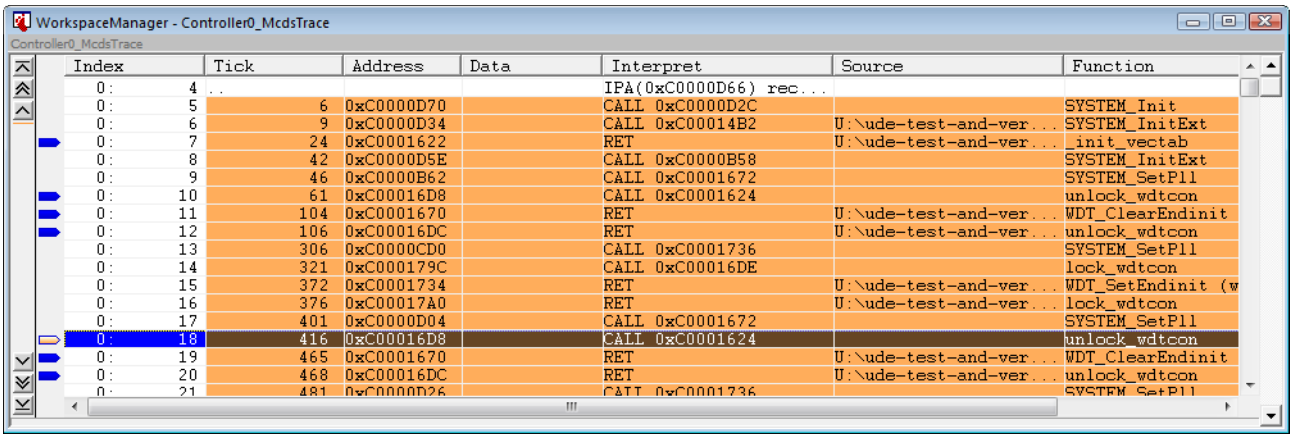
In the example above, an address range in code memory is selected. The **Markers** window shows the search results of all recorded occurrences of instructions in the address range.



Title
Find Instruction Address

Marker	Idx		Tick	Adr	Data	Interpret	Source	Function
0	0:	7	0.16 us	0xC0001622		RET	U:\ude-t...	_init_ve...
1	0:	10	0.41 us	0xC00016D8		CALL 0xC...		unlock_w...
2	0:	11	0.69 us	0xC0001670		RET	U:\ude-t...	WDT_Clea...
3	0:	12	0.71 us	0xC00016DC		RET	U:\ude-t...	unlock_w...
4	0:	18	2.77 us	0xC00016D8		CALL 0xC...		unlock_w...
5	0:	19	3.10 us	0xC0001670		RET	U:\ude-t...	WDT_Clea...
6	0:	20	3.12 us	0xC00016DC		RET	U:\ude-t...	unlock_w...

A click on a marker entry navigates the trace window to the corresponding position.

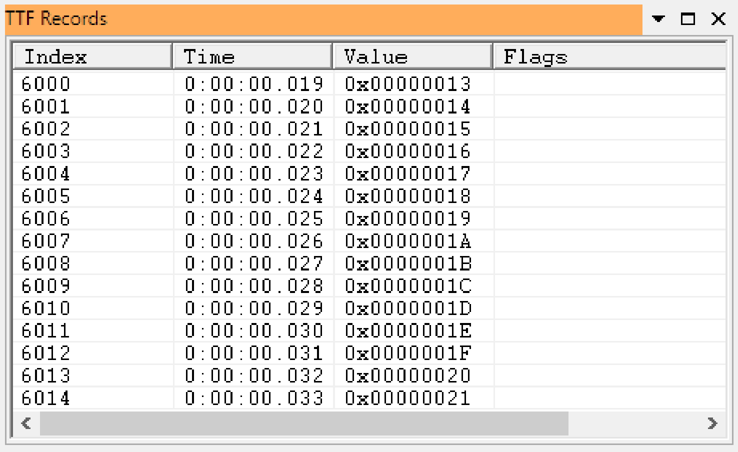


	Index		Tick	Address	Data	Interpret	Source	Function
	0:	4	..			IPA(0xC0000D66) rec...		
	0:	5	6	0xC0000D70		CALL 0xC0000D2C		SYSTEM_Init
	0:	6	9	0xC0000D34		CALL 0xC00014B2	U:\ude-test-and-ver...	SYSTEM_InitExt
➡	0:	7	24	0xC0001622		RET	U:\ude-test-and-ver...	_init_vectab
	0:	8	42	0xC0000D5E		CALL 0xC0000B58		SYSTEM_InitExt
	0:	9	46	0xC0000B62		CALL 0xC0001672		SYSTEM_SetPll
➡	0:	10	61	0xC00016D8		CALL 0xC0001624		unlock_wdtcon
➡	0:	11	104	0xC0001670		RET	U:\ude-test-and-ver...	WDT_ClearEndinit
➡	0:	12	106	0xC00016DC		RET	U:\ude-test-and-ver...	unlock_wdtcon
	0:	13	306	0xC0000CD0		CALL 0xC0001736		SYSTEM_SetPll
	0:	14	321	0xC000179C		CALL 0xC00016DE		lock_wdtcon
	0:	15	372	0xC0001734		RET	U:\ude-test-and-ver...	WDT_SetEndinit (w
	0:	16	376	0xC00017A0		RET	U:\ude-test-and-ver...	lock_wdtcon
	0:	17	401	0xC0000D04		CALL 0xC0001672		SYSTEM_SetPll
⇨	0:	18	416	0xC00016D8		CALL 0xC0001624		unlock_wdtcon
➡	0:	19	465	0xC0001670		RET	U:\ude-test-and-ver...	WDT_ClearEndinit
➡	0:	20	468	0xC00016DC		RET	U:\ude-test-and-ver...	unlock_wdtcon
	0:	21	481	0xC0000D26		CALL 0xC0001736		SYSTEM_SetPll

Triggered Transfer Recorder

UDE TTF Recorder uses the Triggered Transfer Feature of Infineon microcontrollers. Triggered Transfer is part of the on-chip debug support implemented on these controllers. It allows the transfer of a value of a single memory location via the JTAG debug interface. The transfer is triggered by a debug event of the on-chip debug support (OCDS) unit. There are several types of debug events, which can trigger the transfer, depending on the actual type of controller. A typical use case, provided by all supported controller types, is to trigger on write accesses to a single variable and to transfer the new value of the variable.

The recording is done while the target system is in running state. Activate the Add-In **TTF Recorder** and open menu **Views – Add-In Windows – TTF Records**.

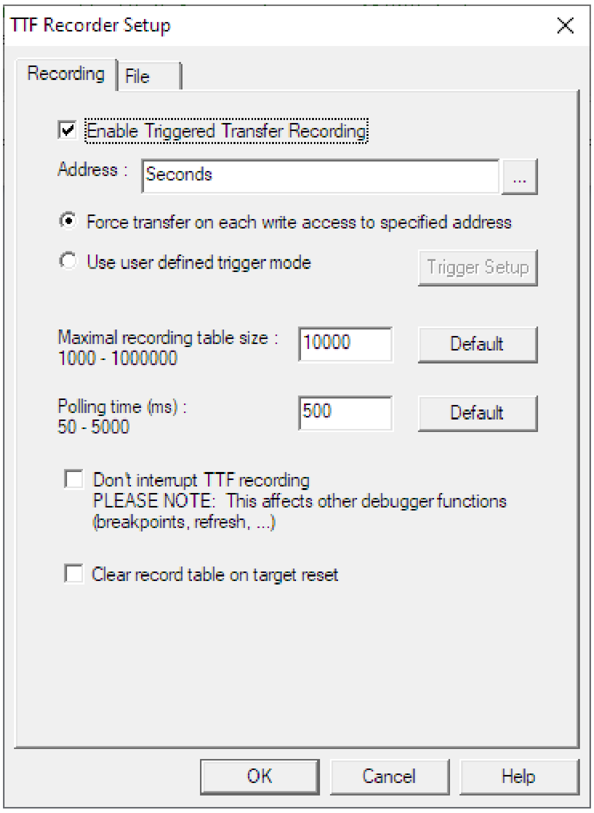


Index	Time	Value	Flags
6000	0:00:00.019	0x00000013	
6001	0:00:00.020	0x00000014	
6002	0:00:00.021	0x00000015	
6003	0:00:00.022	0x00000016	
6004	0:00:00.023	0x00000017	
6005	0:00:00.024	0x00000018	
6006	0:00:00.025	0x00000019	
6007	0:00:00.026	0x0000001A	
6008	0:00:00.027	0x0000001B	
6009	0:00:00.028	0x0000001C	
6010	0:00:00.029	0x0000001D	
6011	0:00:00.030	0x0000001E	
6012	0:00:00.031	0x0000001F	
6013	0:00:00.032	0x00000020	
6014	0:00:00.033	0x00000021	

Setup

For configuration of the TTF recorder, use the context menu **Setup Recording**. Enable the recording of transferred values and set the **Address** of the memory location or the symbolic variable to be transferred when the trigger event has occurred. Depending on the data wide of the transferred value, this has to be a valid 16-bit or 32-bit address.

For special purposes the trigger can be setup on hardware level as **User defined trigger mode**. It manages the possibilities of the debug hardware and special knowledge is required.



☑ Enable Triggered Transfer Recording

Address : Seconds

● Force transfer on each write access to specified address

○ Use user defined trigger mode

Maximal recording table size : 1000 - 1000000: 10000

Polling time (ms) : 50 - 5000: 500

□ Don't interrupt TTF recording
PLEASE NOTE: This affects other debugger functions (breakpoints, refresh, ...)

□ Clear record table on target reset

FLASH / OTP Programming

The UDE Memory Programming Tool supports handling of on-chip FLASH, OTP and EEPROM memory on all supported microcontrollers and external FLASH memory devices. According to the capabilities of the respective programmable memory device this tool allows to erase, program, verify and protect the module. The Intel-Hex and Motorola-SRE format are supported.

There are two versions of UDE MemTool:

1. an integrated UDE Debugger Add-In used within an UDE Workspace,
   - build-in component of the UDE architecture
   - can use all communication channels supported by the UDE Debugger
2. a stand-alone application used outside of the UDE Debugger
   - contains a separate front-end with direct access to all programming functions
   - offers additional batch- and remote functions
   - allows target communication via ASC bootstrap loading and Host PC serial port additionally

Supported Functions

Depending on the capabilities of the memory device (and the driver used to handle the device), MemTool provides functions to handle the device:

- a **chip erase** function, that deletes the contents of the whole memory device
- a **sector erase** function, that deletes the contents of one or more sectors of the device
- a **program** function to write data into the memory device. This function also takes care about certain data alignment requirements of the device
- a **verify** function that is implemented using the read functions of the target interface
- a **query state** Function that can read the state registers of the memory device
- a **protect** function that can be used to install, enable and disable read and write protection mechanisms

Basic Concept

UDE MemTool is intended to handle on-chip and external memory devices that do not permit direct and random write accesses, e.g. as a RAM device does. Typically, on-chip FLASH/OTP memory devices and external FLASH are of this type.

A target may contain several on-chip and external memory devices that can all be handled by UDE MemTool. At a given time, only one device is activated. For each memory device, a Memory Device Handler inside UDE MemTool handles all accesses to the corresponding device. These Memory Device Handlers may be activated and deactivated individually.

Programming of the memory device is done by the Memory Device Driver which is a small application executed by the target MCU. MemTool uses functions, provided by the UDE Target Interface, to load and run this driver application.

Target Communication

In the UDE MemTool

- JTAG, DAP, SWD, OnCE, COP or
- ASC, CAN mini-monitors

are used for communication and FLASH programming. Target communication via JTAG, DAP, SWD, OnCE, COP needs no external RAM and supports **single-chip systems**.

Mini-monitors uses on-chip resources, so that the FLASH programming works without external RAM. **Single-chip systems** are supported.

The integrated UDE Debugger Add-In uses the selected debug communication channel for communication, debugging and FLASH programming. These debug-monitors require external RAM per default.

The following table gives an overview about all available communication channels supported by the integrated and the stand-alone UDE MemTool:

Target µController and communication interface	Host Serial RS232	Universal Access Device[2]
AURIX, TriCore, Power Architecture, Cortex, ARM, RH850, SH-2A, XC200, XC166, XScale via **JTAG, DAP, SWD, OnCE, COP**		✓
AURIX, TriCore, C166, ST10, XC166, XC2000, MPC55xx via **ASC-BSL**	✓[1]	✓
ST10, XC166, XC2000, AURIX, TriCore via **CAN-BSL**		✓
C166, ST10, XC166, XC2000, AURIX, TriCore via **K-Line**		✓

[1] only available with the stand-alone UDE MemTool

[2] all variants.

Supported FLASH/OTP Memory Devices

On-chip FLASH/OTP memory devices on microcontroller supported by UDE can be programmed with UDE MemTool.

External FLASH-EPROM's are also programmable …

- AMD Am29Fxxx, M29Wxxx family and 100% compatible types
- Atmel AT29Cxxx, AT49Cxxx family
- SST39VFxxx, SST39LFxxx family
- M58BWxxx family, ST58BWxxx family
- Intel i28Fxxx family
- I2C 24LCxx family
- all JEDEC Standard Command Set compatible.

For the list of supported external memory devices, see the file `EXTFLASH.DAT` in the UDE installation folder. If newer FLASH devices are available, which are not supported by UDE MemTool, please contact the PLS Support Team at **support@pls-mc.com** to receive a newer version of the FLASH database.

Please note: Versions of UDE MemTool shipped with UDE Starterkits are limited to memory devices provided by the respective Starterkit hardware.

Memory mapping variants

To understand how MemTool programs a user application into a programmable memory device, it is necessary to consider the two memory mapping types used by MemTool. It defines the memory mapping variants **Program Time Mapping** and **Run Time Mapping**.

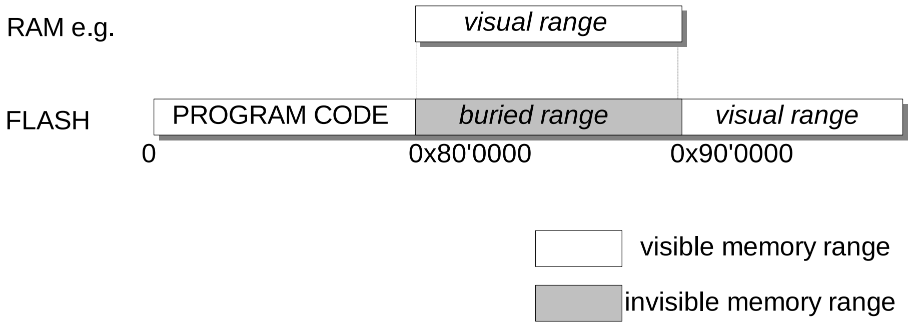
To understand the difference between these two mapping types, consider the following scenario: Imagine a target system with the C16x, 1 MByte external RAM and 256 kByte external FLASH.

After a target reset, the program code is fetched from the FLASH memory at segment `0` (physical address `0x00'0000`). That is why the RAM is located by the initialization code at segment `8` (physical address `0x80'0000`). This constellation is called **Run Time Mapping**.

The program, which is in development state, is located at segment `0` (physical address `0x00'0000`). For debugging purposes, the on-board RAM is mapped to segment `0`, so that breakpoints can be handled, for instance. After the development stage finished, the program is programmed into the FLASH at segment `0`, so that the power-on execution can be done from the FLASH program content. However, at this moment the FLASH is not visible at segment `0`, because the RAM overlaps the FLASH range. That's why the UDE MemTool remaps all program code sections to another visible range of the FLASH for FLASH programming, in this case at segment 8 (physical address `0x80'0000`). This constellation is called **Program Time Mapping**.

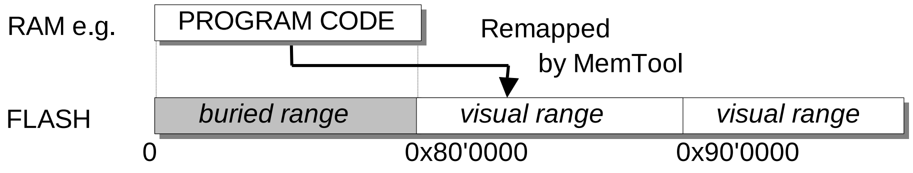
Run Time Mapping

Mapping type is used when the user application is executed.



Program Time Mapping

Mapping type used when the user application is about to be programmed into the Memory Device,



To configure this example in UDE following settings must be done: The FLASH memory must be defined in the target configuration as memory device with a visible **address range**: `0x80'0000-0x8F'FFFF`. To allow the remapping of the program code sections the MemTool must be setup with a **different start address**: 0x00'0000 (via **Setup of FLASH/OTP Device** dialog).

Please see the next chapter for more detailed information about the configuring of FLASH memory devices

Definition of external FLASH Memories

Before FLASH programming is possible, the external FLASH devices must be defined in target configuration of UDE. This can be done during creating of a new Target Configuration, but by editing an existing configuration too.

Create a new Target Configuration with FLASH support

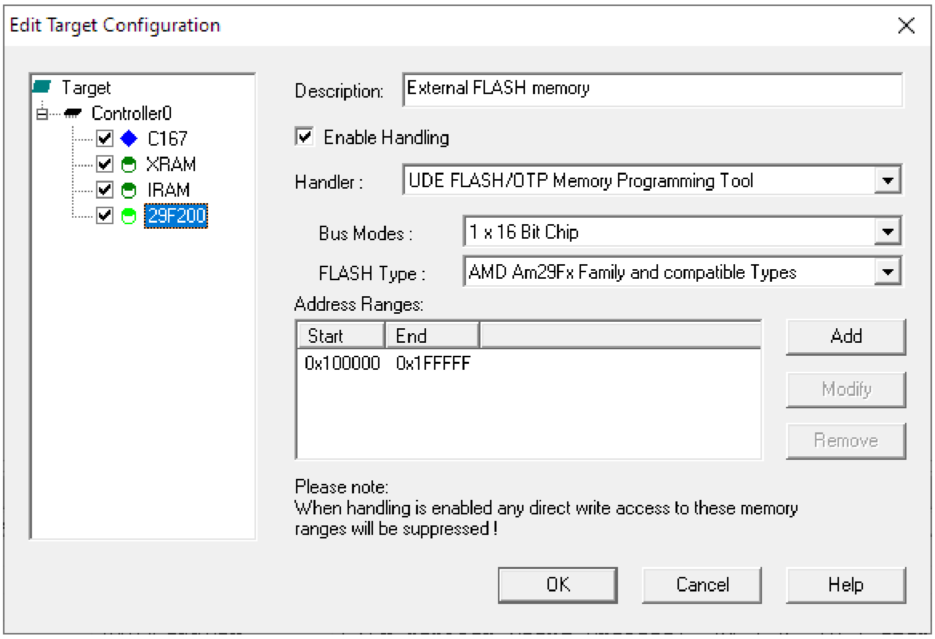
Create a new target configuration and specify the count of FLASH memory devices in page **New Target Specify Memory**. Define the device with name and description in page **New Target Special Memory** and select as **Handler UDE FLASH/OTP Memory Programming Tool**. Setup the used **Bus Mode** of the FLASH device, the **FLASH Type**, which means the family of FLASH devices and the **Start** and **End Address**, where the FLASH device is visible completely.

Edit an existing Target Configuration with FLASH support

When the target is connected, open the **Edit Target Configuration** dialog via menu **Config** – **Target Configuration** and select the controller item in the target tree. Via the **Add** and **Modify** button, you can change the list of registered memory devices. Use **Remove** for deleting of entries.

Push **Add** to setup a new memory device. Enter a name and a description of the FLASH memory and enable the handling by UDE FLASH/OTP Memory Programming Tool. Select the used **Bus Mode** of the FLASH device and the **FLASH Type**, which means the family of FLASH devices.

Add a new address range, where the FLASH device is visible completely.



Description: External FLASH memory

☑ Enable Handling

Handler : UDE FLASH/OTP Memory Programming Tool

Bus Modes : 1 x 16 Bit Chip

FLASH Type : AMD Am29Fx Family and compatible Types

Address Ranges:

Start	End
0x100000	0x1FFFFF

Please note:
When handling is enabled any direct write access to these memory ranges will be suppressed !

Definition of on-chip FLASH Memories

The on-chip FLASH devices are registered automatically. The requirement is that the correct controller derivative is selected in the target configuration. In this case, an additional memory device labeled with **PFLASH** or **DFLASH** is included. Please check, that the on-chip FLASH is enabled on the property sheet of the device.

Please note: For programming on-chip FLASH modules of C16x and ST10Fx derivatives, the on-chip FLASH module must be activated (`SYSCON.ROMEN=1`) and mapped to starting address `0x10000` (`SYSCON.ROMS1=1`).
To run the FLASH driver software, XRAM must be enabled (`SYSCON.XPEN=1`).

Definition of Memory Access Filters

A memory access filter is used like a FLASH Memory Device with the difference, that no FLASH programming functionality is provided, but various memory access filter functions. Memory access filters are useful, if devices, visible in the address range of the microcontroller, must not be read and/or written by UDE. A memory access filter reserves an address range and can be specified as

- Block any access
- Block any write access
- Notify user on write access

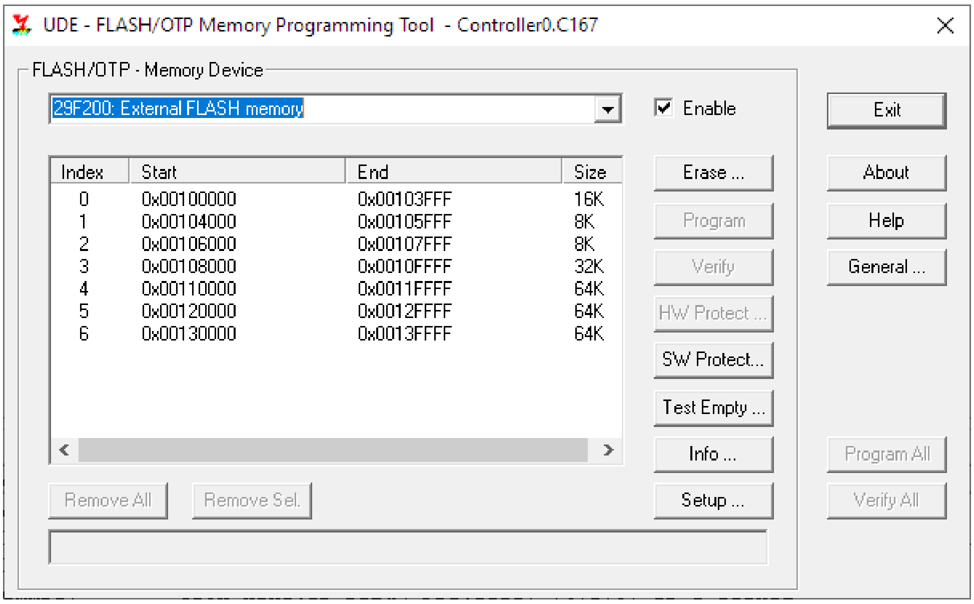
Define the memory access filter as new memory device with name and description in page **New Target Special Memory** of the target configuration and select as **Handler Memory access filter**.

Enabling the FLASH Programming

The UDE Memory Programming Tool is an Add-In of UDE and must be activated. This is done via the Add-In Manager, menu **Config** – **Add-in Components**. Enable the entry **UDE FLASH/OTP Memory Programming Tool**.

The detailed description about Add-Ins is contained in the chapter **Activating and Using Add-Ins**.

If the UDE FLASH/OTP Memory Programming Tool is enabled, a new menu entry is created in UDE menu **Tools** – **FLASH Programming …**. Open this dialog and the main front-end of MemTool will be opened. Choose the FLASH device and try to enable it. If all settings were correct, a list of FLASH sectors will be displayed as shown below.



29F200: External FLASH memory

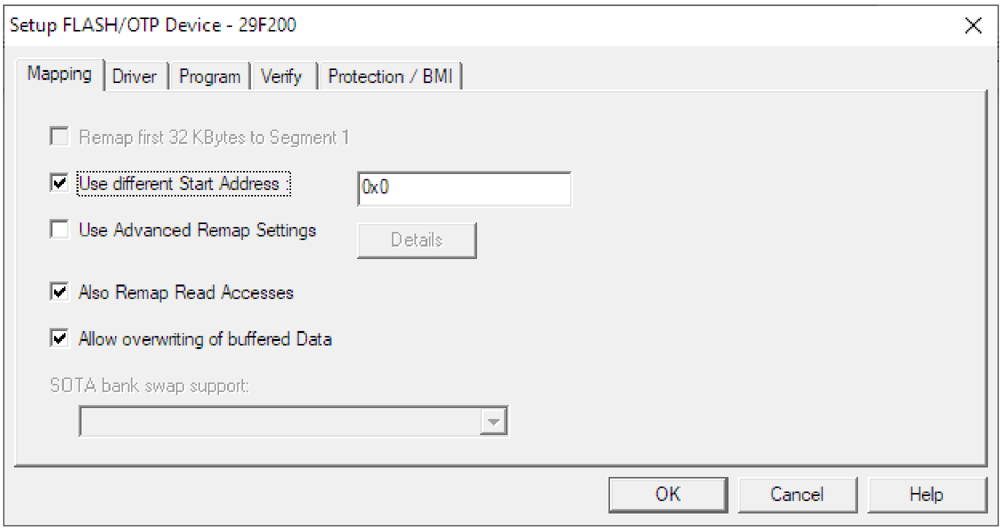
☑ Enable

Index	Start	End	Size
0	0x00100000	0x00103FFF	16K
1	0x00104000	0x00105FFF	8K
2	0x00106000	0x00107FFF	8K
3	0x00108000	0x0010FFFF	32K
4	0x00110000	0x0011FFFF	64K
5	0x00120000	0x0012FFFF	64K
6	0x00130000	0x0013FFFF	64K

If not, please check carefully all FLASH device settings, the target hardware and the content of the SYSTEM and BUSCONFIG registers again. The FLASH device must be visible in full size. Use the Memory window to verify your assumption about the hardware in terms of the address ranges of RAM and FLASH.

As shown above, a list of FLASH sectors is displayed. These sectors are located on the FLASH addresses adjusted to the corresponding physical address ranges defined in the target configuration. Because the program code is mostly located at `0x0`, the 'Program Time Mapping' must be activated.

Push the **Setup…** button to set the **Use different Start Address** to `0x0`. Now the 'Program Time Mapping' is selected within UDE and the code range filter will be activated from physical address `0x0` with the size of the FLASH device.



□ Remap first 32 KBytes to Segment 1

☑ Use different Start Address 0x0

□ Use Advanced Remap Settings

☑ Also Remap Read Accesses

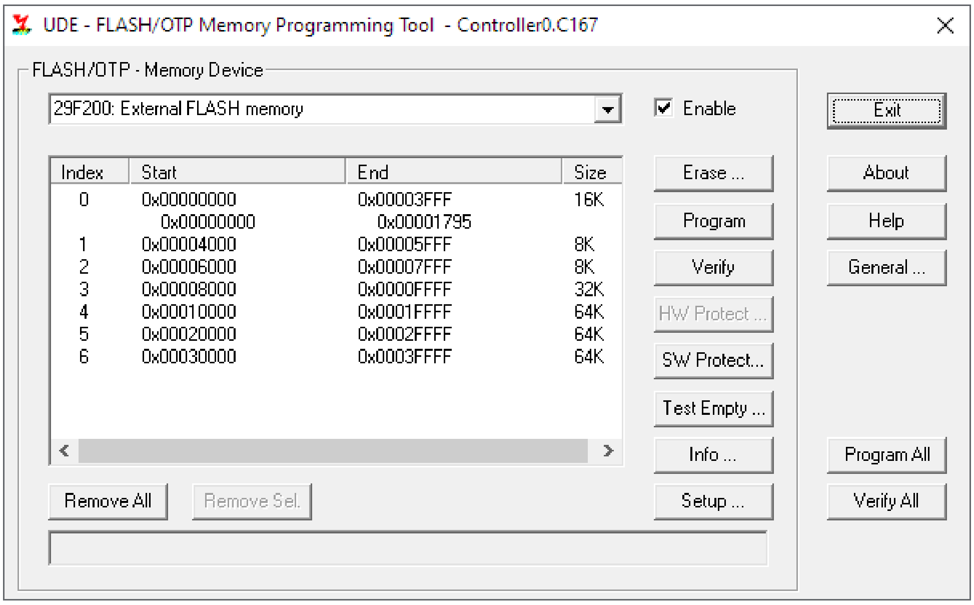
☑ Allow overwriting of buffered Data

SOTA bank swap support:

FLASH Programming

If the UDE FLASH/OTP Memory Programming Tool is enabled, all registered FLASH devices are installed with special filters. These filters watch the download stream for address ranges, matching a registered FLASH device. If the filters detect, that a code section is loaded, which is destined for the FLASH device, the code section will be marked for FLASH programming.

After loading the program code sections, the FLASH Programming Tool will open the main dialog and will offer the erasing, programming and verifying of code sections.



29F200: External FLASH memory

☑ Enable

Index	Start	End	Size
0	0x00000000	0x00003FFF	16K
	0x00000000	0x00001795	
1	0x00004000	0x00005FFF	8K
2	0x00006000	0x00007FFF	8K
3	0x00008000	0x0000FFFF	32K
4	0x00010000	0x0001FFFF	64K
5	0x00020000	0x0002FFFF	64K
6	0x00030000	0x0003FFFF	64K

The **Erase ...** button allows the erasing of single FLASH sectors or complete FLASH devices. Because of the FLASH architecture, a FLASH sector has to be erased before new FLASH programming. Before starting the Erase Function, the **Select FLASH Sectors for Erase** dialog is shown. One or more sectors to be erased can be selected here. If the current Memory Device supports a Chip Erase Function, you may choose **Erase whole FLASH Module** to execute this function. Otherwise, Sector Erase Function is used.

Program immediately starts to write all sections loaded into the current Memory Device Handler. It presents the Execute Command Box showing the function progress. Canceling of the operation is also provided.

The **Verify** function immediately starts to compare all sections loaded into the current Memory Device Handler against the contents of the Memory Device on target. It presents the **Execute Command** Box showing the function progress. Canceling of the operation is also provided. As the result, after the function has finished, the number of different bytes is shown.

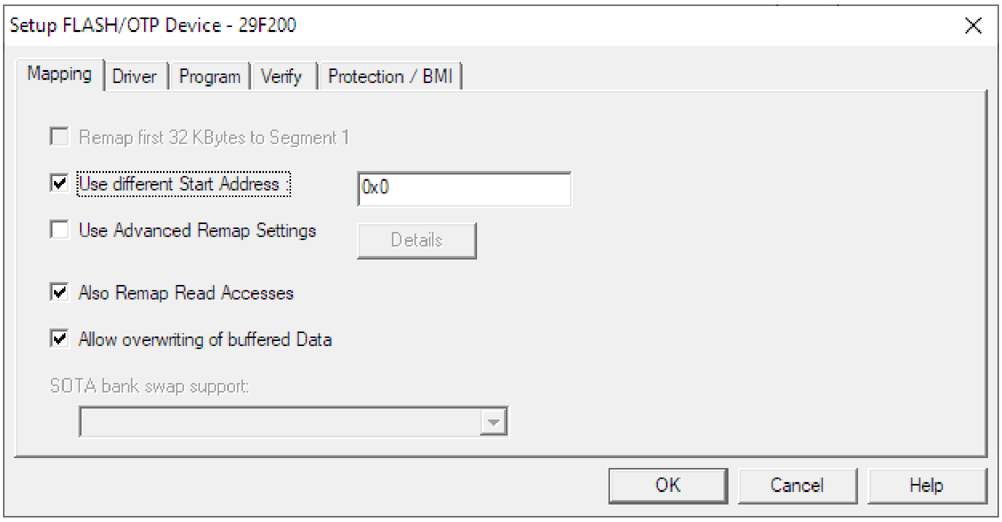
Setup ... provides setup for FLASH programming options.

The **Info ...** window shows information about the current Memory Device and the Memory Device Driver used to handle the device.

Setup FLASH Programming options

Mapping – If Run Time Memory Mapping and Program Time Memory Mapping are not equal, check the **Use different Start Address** checkbox and enter a valid Run Time Mapping start address for the Memory Device.

Driver – The Driver page allows the replacement of FLASH drivers and is destined for special purposes. Changes have to be only done in accordance with the PLS Support Team.



□ Remap first 32 KBytes to Segment 1

☑ Use different Start Address 0x0

□ Use Advanced Remap Settings

☑ Also Remap Read Accesses

☑ Allow overwriting of buffered Data

SOTA bank swap support:

Program – The options **Automatic Chip Erase Before Program** and **Automatic Sector Erase Before Program** allows the automatic erasing of sectors or devices before programming with new data.

With **Simulate Random Access Mode,** all sectors to be programmed with new data are completely read out into buffers. Then, these sectors are erased. Now the buffered data is merged with the new data to be programmed and the result is written back to the memory device.

The verify function is started after programming has finished when **Automatic Verify after Program** is enabled.

UDE CAN Recorder

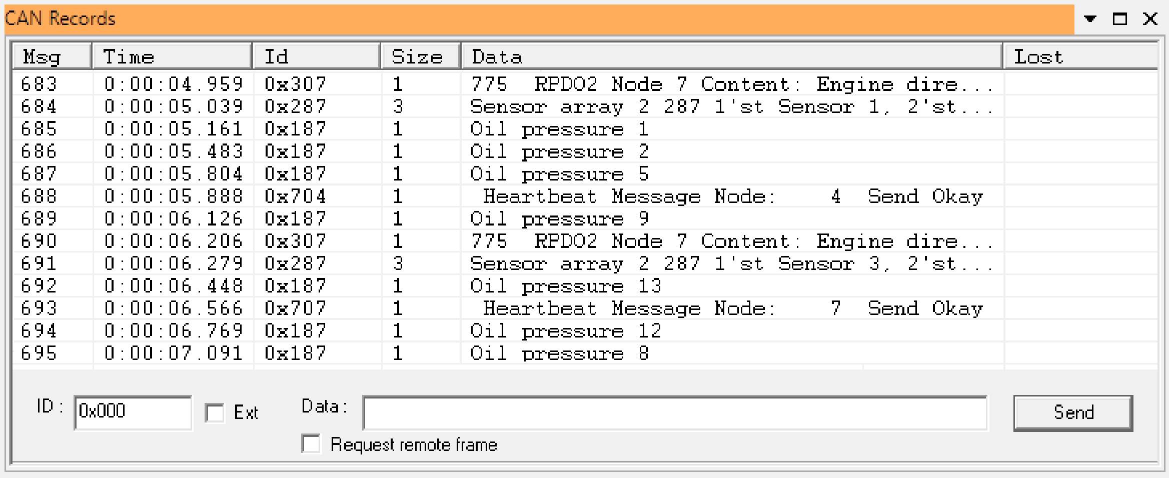
The UDE CAN Recorder is an Add-In of UDE and can send and record CAN messages from a CAN network. Following features are supported:

- Configuring the CAN controller
- Edit and Send CAN messages (CAN bus stimulation)
- Receive CAN messages (CAN bus observation) with filter mechanism
- Symbolic representation of CAN messages

Enabling the CAN Recorder

The UDE CAN Recorder is an Add-In of UDE and must be activated. This is done via the Add-In Manager, menu **Config** – **Add-in Components**. Enable the entry **UDE CAN Recorder**.

If the UDE CAN Recorder is enabled, a new menu entry is created in UDE menu **Views – Add-In Windows – CAN Records**. Start the CAN Recorder via this menu entry.



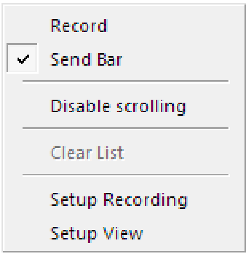
CAN Records

Msg	Time	Id	Size	Data	Lost
683	0:00:04.959	0x307	1	775 RPDO2 Node 7 Content: Engine dire...	
684	0:00:05.039	0x287	3	Sensor array 2 287 1'st Sensor 1, 2'st...	
685	0:00:05.161	0x187	1	Oil pressure 1	
686	0:00:05.483	0x187	1	Oil pressure 2	
687	0:00:05.804	0x187	1	Oil pressure 5	
688	0:00:05.888	0x704	1	Heartbeat Message Node: 4 Send Okay	
689	0:00:06.126	0x187	1	Oil pressure 9	
690	0:00:06.206	0x307	1	775 RPDO2 Node 7 Content: Engine dire...	
691	0:00:06.279	0x287	3	Sensor array 2 287 1'st Sensor 3, 2'st...	
692	0:00:06.448	0x187	1	Oil pressure 13	
693	0:00:06.566	0x707	1	Heartbeat Message Node: 7 Send Okay	
694	0:00:06.769	0x187	1	Oil pressure 12	
695	0:00:07.091	0x187	1	Oil pressure 8	

ID : 0x000 □ Ext Data : Send

□ Request remote frame

Send and Record CAN Messages



Before using the CAN Recorder, it has to be configured. Use the context menu and click on **Setup Recording**. Specify the baud rate of the CAN bus in the tab **Timing**. Setup the CAN message format and mask in the tab **Recording**.

To record messages from the CAN network, use the context menu and enable the Record entry. All messages are recorded and displayed, which correspond to the defined filter mask.

To send a message, type in the **ID** and **Data** form and push the **Send** button.

UDE RTX Awareness

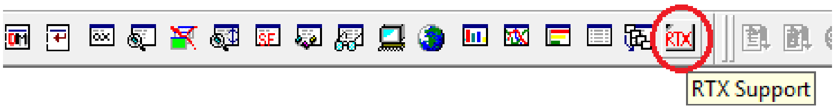
The UDE RTX Awareness is an Add-In of UDE that give a detailed overview about the RTX RTOS internals. These internals can be:

- Tasks overview including all states
- Queue overview (per Message/Semaphore/Mutex Control Block information)
- Timers overview (name, timer count, unique callback information)
- Signals overview (waiting/active events mask)

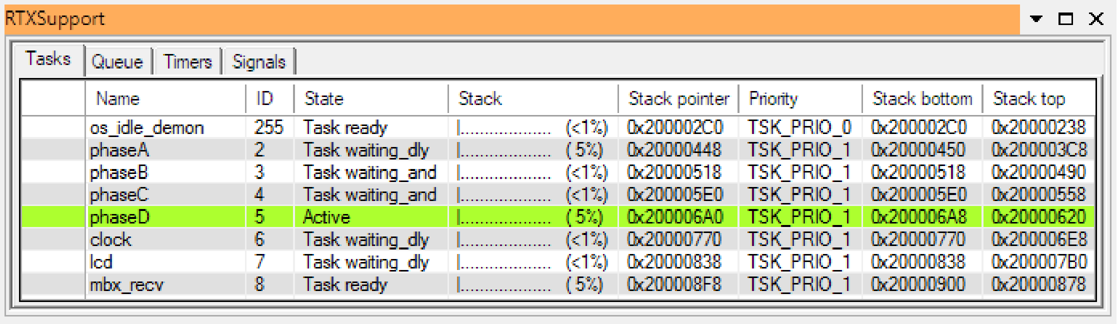
Enabling the RTX Awareness

The UDE RTX Awareness is an Add-In of UDE and must be activated before it can be used. To activate it, open the Add-In Manager, menu **Config** – **Add-in Components**. Enable the entry **RTX Awareness**.

After the UDE RTX Awareness was enabled, a new menu entry is created in UDE menu **Views – Add-In Windows – RTX Awareness**. Open the RTX Awareness by clicking on this menu entry.



Depending on the functions enabled in the RTOS configuration file the RTX Awareness window immediately shows a snapshot of the current system state:



	Name	ID	State	Stack	Stack pointer	Priority	Stack bottom	Stack top
	os_idle_demon	255	Task ready	|.................. (<1%)	0x200002C0	TSK_PRIO_0	0x200002C0	0x20000238
	phaseA	2	Task waiting_dly	|.................. (5%)	0x20000448	TSK_PRIO_1	0x20000450	0x200003C8
	phaseB	3	Task waiting_and	|.................. (<1%)	0x20000518	TSK_PRIO_1	0x20000518	0x20000490
	phaseC	4	Task waiting_and	|.................. (<1%)	0x200005E0	TSK_PRIO_1	0x200005E0	0x20000558
	phaseD	5	Active	|.................. (5%)	0x200006A0	TSK_PRIO_1	0x200006A8	0x20000620
	clock	6	Task waiting_dly	|.................. (<1%)	0x20000770	TSK_PRIO_1	0x20000770	0x200006E8
	lcd	7	Task waiting_dly	|.................. (<1%)	0x20000838	TSK_PRIO_1	0x20000838	0x200007B0
	mbx_recv	8	Task ready	|.................. (5%)	0x200008F8	TSK_PRIO_1	0x20000900	0x20000878

The green line inside the RTX Awareness marks the task that was active/running before the system was halted by the debugger. Please note, because the RTOS creates the tasks at runtime, maybe not all tasks are displayed if the system was broken immediately after first go/step.

Using the RTX Awareness

To work with the RTX Awareness, first an RTX RTOS application AXF file must be loaded. The RTX RTOS application itself can be built using the Keil compiler. Please also note that the RTX RTOS version must be greater than or equal to v4.x.

After the RTX application was loaded, the RTX must be initialized and all needed tasks must be created. This is normally done by an `init()` function. The init function itself is therefore called by `os_sys_init` inside the main-function.
If the initialization was performed successfully, all created tasks are listed in the RTX Awareness as shown above. By setting breakpoints inside the application, the behavior of each task can now be monitored by using the RTX Awareness.

UDE rcX Awareness

The UDE rcX Awareness is an Add-In of UDE that gives a detailed overview about the rcX RTOS internals. These internals can be:

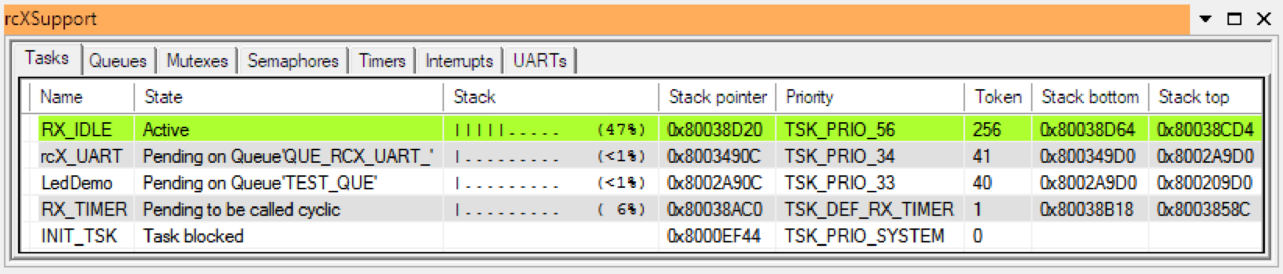
- Tasks overview including all states
- Queues overview (name, fill level, memory usage)
- Mutex overview (name, owner task)
- Semaphores overview (name, modes, state, callback)
- Interrupts overview (name, type, priority, ISR)
- UARTs overview (name, state, speed, callbacks)

The functions are mostly similar to the RTX Awareness above.

Enabling the rcX Awareness

The UDE rcX Awareness is an Add-In of UDE and must be activated before it can be used. To activate it, open the Add-In Manager, menu **Config** – **Add-in Components**. Enable the entry **rcX Awareness**.

After the UDE rcX Awareness was enabled, a new menu entry is created in UDE menu **Views – Add-In Windows – rcX Awareness**. Open the rcX Awareness by clicking on this menu entry. Depending on the functions enabled in the RTOS configuration file the rcX Awareness window immediately shows a snapshot of the current system state:



Name	State	Stack	Stack pointer	Priority	Token	Stack bottom	Stack top
RX_IDLE	Active	IIIII----- (47%)	0x80038D20	TSK_PRIO_56	256	0x80038D64	0x80038CD4
rcX_UART	Pending on Queue'QUE_RCX_UART_'	I---------- (<1%)	0x8003490C	TSK_PRIO_34	41	0x800349D0	0x8002A9D0
LedDemo	Pending on Queue'TEST_QUE'	I---------- (<1%)	0x8002A90C	TSK_PRIO_33	40	0x8002A9D0	0x800209D0
RX_TIMER	Pending to be called cyclic	I---------- (6%)	0x80038AC0	TSK_DEF_RX_TIMER	1	0x80038B18	0x8003858C
INIT_TSK	Task blocked		0x8000EF44	TSK_PRIO_SYSTEM	0		

The green line inside the rcX Awareness marks the task, which was active/running before the system was halted by the debugger. Please note, because the RTOS creates the tasks at runtime, not all tasks might be displayed, if the system was halted immediately after the first go/step.

Using the rcX Awareness

To use the rcX Awareness, load an rcX RTOS application ELF file. The rcX RTOS application itself can be built using the GNU ARM compiler. Please also note that the rcX RTOS version must be greater than or equal to v2.x.
After the rcX application was loaded, the rcX RTOS must be initialized and all needed tasks must be created. This is normally done by an `init()` function. The init function itself is therefore called by `rX_SysEnterKernelExt` inside the main-function.
If the initialization was performed successfully, all created tasks are listed in the rcX Awareness as shown above. By setting breakpoints inside the application, the behavior of each task can now be monitored, using the rcX Awareness.

UDE FreeRTOS Awareness

The UDE FreeRTOS Awareness is an Add-In of UDE that gives a detailed overview about the FreeRTOS internals. These internals can be:

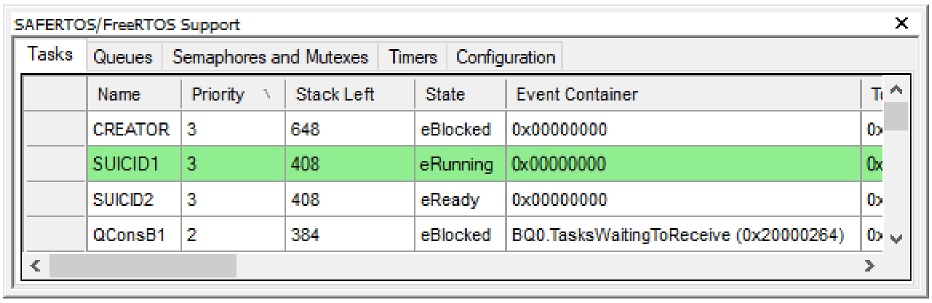
- Tasks overview (name, state, priority, stack …)
- Queues overview (name, size, items, blocked tasks, …)
- Semaphore and mutexes overview (name, count, blocked tasks, …)
- Timers overview (name, ticks, callback, …)
- FreeRTOS Configuration

Additionally, the Add-In provides an automated configuration for task trace and analyses configuration for the Execution Sequence Chart.

Enabling the FreeRTOS Awareness.

The UDE SAFERTOS/FreeRTOS Awareness is an Add-In of UDE and must be activated before it can be used. To activate it, open the Add-In Manager, menu **Config – Add-in Components**. Enable the entry **SAFERTOS/FreeRTOS Awareness**.

After the UDE SAFERTOS/FreeRTOS Awareness was enabled, a new menu entry is created in UDE menu **Views – Add-In Windows – SAFERTOS/FreeRTOS Awareness**. Open the SAFERTOS/FreeRTOS Awareness by clicking on this menu entry. Depending on the functions enabled in the FreeRTOS configuration header file the FreeRTOS Awareness window immediately shows a snapshot of the current system state:



Name	Priority	Stack Left	State	Event Container	T
CREATOR	3	648	eBlocked	0x00000000	0x
SUICID1	3	408	eRunning	0x00000000	0x
SUICID2	3	408	eReady	0x00000000	0x
QConsB1	2	384	eBlocked	BQ0.TasksWaitingToReceive (0x20000264)	0x

The green line marks the task, which was active/running before the system was halted by the debugger. Please note, because the RTOS creates tasks at runtime, not all tasks might be displayed, if the system was halted immediately after the first go/step.

FreeRTOS Task Trace

The task trace can be configured right clicking on any table and selecting **Configure Trace – Task Trace**. A Dialog shows all selectable cores. Select **Code Trace** and **Running Task Trace** of the core, FreeRTOS is running on. Start the Trace using the menu Entry **Tools – Start Trace…** and analyze it using the Execution Sequence Chart. Please note, that only Tasks, available during configuration, can be displayed in the Execution Sequence Chart.

Using the FreeRTOS Awareness

To use the FreeRTOS Awareness, load an FreeRTOS application ELF file. Please also note that the FreeRTOS Awareness was developed for FreeRTOS version greater than or equal to 10.2.x.

If an FreeRTOS application was loaded and the initialization was performed successfully, all created tasks are listed in the FreeRTOS Awareness as shown above. By setting breakpoints inside the application, the behavior of each task can now be monitored, using the FreeRTOS Awareness.

Refer to the online help (**F1** or **Shift+F1**) for a more detailed description.

UDE SAFERTOS Awareness

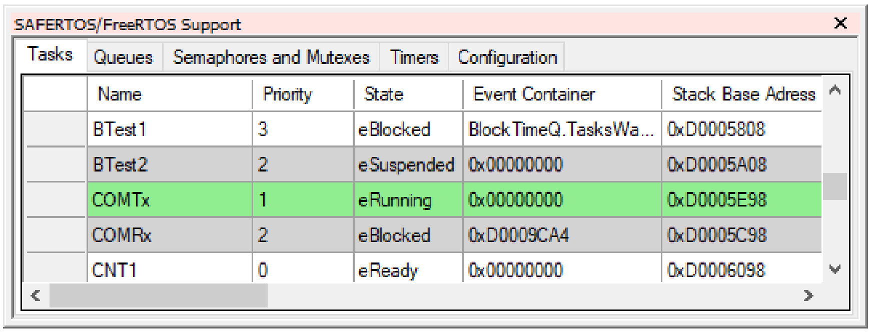
The UDE SAFERTOS Awareness is an Add-In of UDE that gives a detailed overview about the SAFERTOS internals. These internals can be:

- Tasks overview (name, state, priority, stack …)
- Queues overview (name, size, items, blocked tasks, …)
- Semaphore and mutexes overview (name, count, blocked tasks, …)
- Timers overview (name, ticks, callback, …)

Enabling the SAFERTOS Awareness

The UDE SAFERTOS/FreeRTOS Awareness is an Add-In of UDE and must be activated before it can be used. To activate it, open the Add-In Manager, menu **Config – Add-in Components.** Enable the entry **SAFERTOS/FreeRTOS Awareness**.

After the UDE SAFERTOS/FreeRTOS Awareness was enabled, a new menu entry is created in UDE menu **Views – Add-In Windows – SAFERTOS/FreeRTOS Awareness**. Open the SAFERTOS/FreeRTOS Awareness by clicking on this menu entry. Depending on the functions enabled in the SAFERTOS configuration header file the SAFERTOS Awareness window immediately shows a snapshot of the current system state:

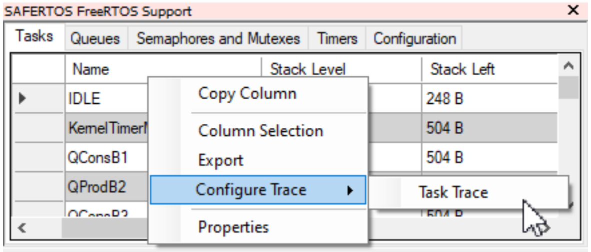


Name	Priority	State	Event Container	Stack Base Adress
BTest1	3	eBlocked	BlockTimeQ.TasksWa...	0xD0005808
BTest2	2	eSuspended	0x00000000	0xD0005A08
COMTx	1	eRunning	0x00000000	0xD0005E98
COMRx	2	eBlocked	0xD0009CA4	0xD0005C98
CNT1	0	eReady	0x00000000	0xD0006098

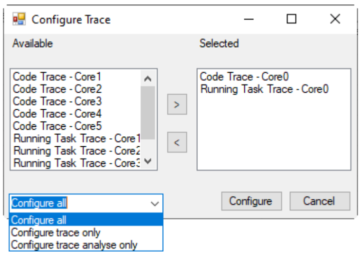
The green line marks the task, which was active/running before the system was halted by the debugger. Please note, because the RTOS creates the tasks at runtime, not all tasks might be displayed, if the system was halted immediately after the first go/step.

SAFERTOS Task Trace

The task trace can be configured right clicking on any table and selecting **Configure Trace – Task Trace**.



Name	Stack Level	Stack Left
IDLE		248 B
KernelTimerI		504 B
QConsB1		504 B
QProdB2		
QConsB3		504 B



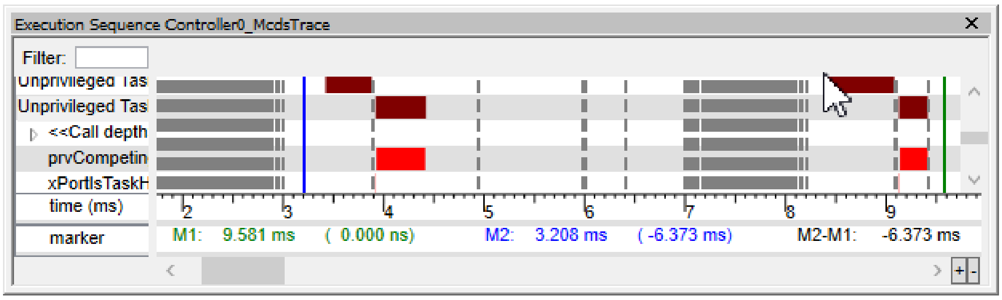
Available:
Code Trace - Core1
Code Trace - Core2
Code Trace - Core3
Code Trace - Core4
Code Trace - Core5
Running Task Trace - Core1
Running Task Trace - Core2
Running Task Trace - Core3

Selected:
Code Trace - Core0
Running Task Trace - Core0

Configure all
Configure all
Configure trace only
Configure trace analyse only

A Dialog shows all selectable cores. Select **Code Trace** and **Running Task Trace** of the core, SAFERTOS is running on.

Start the Trace using the menu Entry **Tools – Start Trace…** and analyze it using the Execution Sequence Chart. Please note, that only Tasks, available during configuration, can be displayed in the Execution Sequence Chart.



If tasks are available after the trace is run, you can repeat configuring the trace. If you do, select the option **Configure trace analyze only**.

Using the SAFERTOS Awareness

To use the SAFERTOS Awareness, load an SAFERTOS application ELF file. Please also note that the SAFERTOS Awareness was developed for SAFERTOS version greater than or equal to 6.2.

If an SAFERTOS application was loaded and the initialization was performed successfully, all created tasks are listed in the SAFERTOS Awareness as shown above. By setting breakpoints inside the application, the behavior of each task can now be monitored, using the SAFERTOS Awareness.

Refer to the online help (**F1** or **Shift+F1**) for a more detailed description.

UDE PXROS-HR Awareness

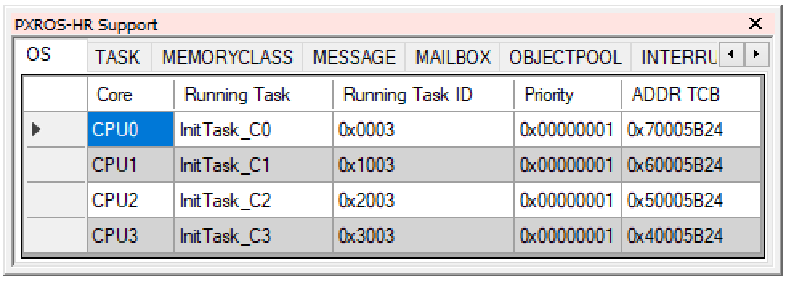
The UDE PXROS-HR Awareness is an Add-In of UDE that gives a detailed overview about the PXROS-HR internals. These internals can be:

- OS overview (running task information)
- Tasks overview (name, id, priority, stack, default objects, …)
- Memory classes overview (id, type, left memory)
- Messages overview (id, sender, recipient, data)
- Mailboxes overview (id, waiting tasks, messages, …)
- Object pools overview (id, type, objects left)
- Interrupts overview (id, interrupt, handle information)
- Timers overview (id, type, ticks, callback/events, …)
- Inter core communication

Enabling the PXROS-HR Awareness

The UDE PXROS-HR Awareness is an Add-In of UDE and must be activated before it can be used. To activate it, open the Add-In Manager, menu **Config – Add-in Components**. Enable the entry **PXROS-HR Awareness**.

After the UDE PXROS-HR Awareness was enabled, a new menu entry is created in UDE menu **Views – Add-In Windows – PXROS-HR Awareness**. Open the PXROS-HR Awareness by clicking on this menu entry. Depending on the functions enabled in the SAFERTOS configuration header file the PXROS-HR Awareness window immediately shows a snapshot of the current system state:



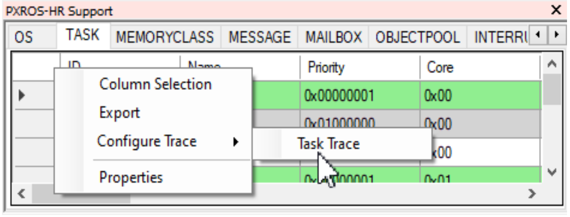
PXROS-HR Support

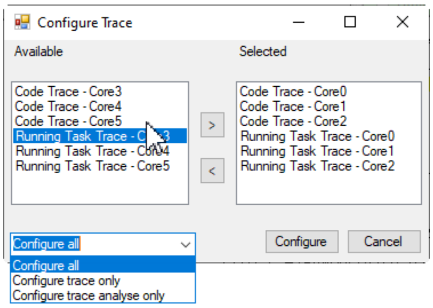
Core	Running Task	Running Task ID	Priority	ADDR TCB
CPU0	InitTask_C0	0x0003	0x00000001	0x70005B24
CPU1	InitTask_C1	0x1003	0x00000001	0x60005B24
CPU2	InitTask_C2	0x2003	0x00000001	0x50005B24
CPU3	InitTask_C3	0x3003	0x00000001	0x40005B24

Please note, because PXROS-HR creates most objects at runtime, not all of them might be displayed, if the system was halted immediately after the first go/step.

PXROS-HR Task Trace

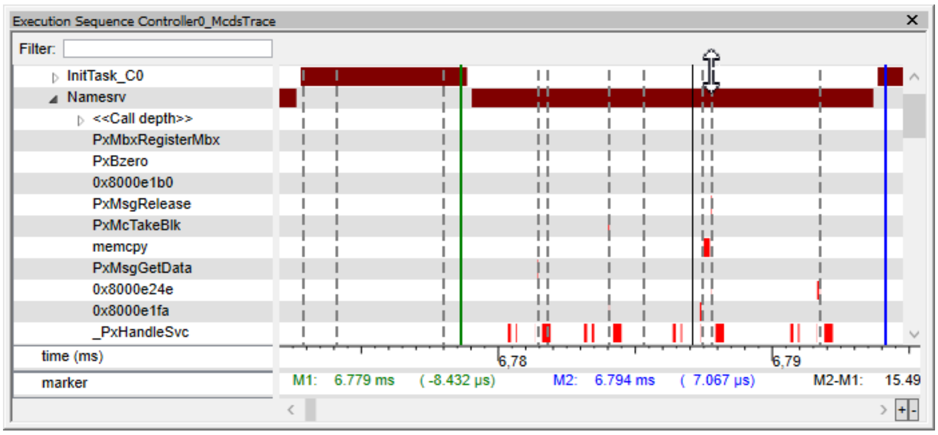
The task trace can be configured right clicking on any table and selecting **Configure Trace – Task Trace**.





A Dialog shows all selectable cores. Select **Code Trace** and **Running Task Trace** of the cores, you want to see. The number of selectable cores depends on the hardware you are using.

Start the Trace using the menu Entry **Tools – Start Trace…** and analyze it using the Execution Sequence Chart. Please note, that only tasks, available during configuration, can be named in the Execution Sequence Chart. Other tasks are named after their object id.



If tasks are available after the trace is run, repeat configuring the task trace. If you do so, use the option **Configure trace analyze only**.

Using the PXROS-HR Awareness

To use the PXROS-HR Awareness, load an PXROS-HR application ELF file. Please also note that the PXROS-HR Awareness was developed for PXROS-HR version greater than or equal to 7.3. After the PXROS-HR application was loaded, PXROS-HR must be initialized. This is normally done by the `PxInit()` function.

If the initialization was performed successfully, all system objects are listed in the PXROS-HR Awareness. The OS tab refers to the running tasks as shown above. By setting breakpoints inside the application, the behavior of each task can now be monitored by using the PXROS-HR Awareness.

UDE µC/OS-II Awareness

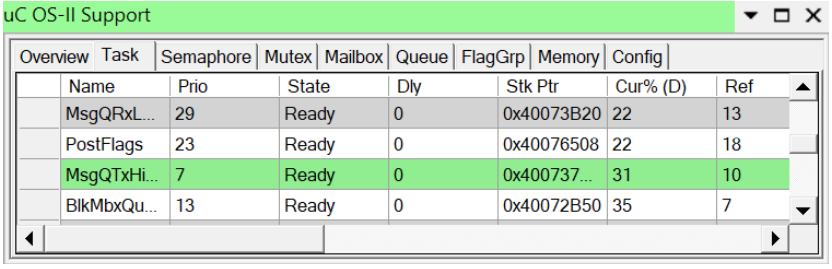
The UDE µC/OS-II Awareness is an Add-In of UDE that gives a detailed overview about the µC/OS-II internals. These internals can be:

- Overview (running task, statistics, …)
- Tasks overview (name, state, priority, stack, reference, …)
- Semaphore overview (name, reference, count)
- Mutex (name, reference, priority information)
- Mailboxes overview (name, reference, msg)
- Queue (name, reference, fill level, …)
- Flag Group (name, reference, flags, wait type)
- Memory (name, reference, memory partition and usage)
- µC/OS-II Configuration

Enabling the µC/OS-II Awareness

The UDE µC/OS-II Awareness is an Add-In of UDE and must be activated before it can be used. To activate it, open the Add-In Manager, menu **Config – Add-in Components**. Enable the entry **µC/OS-II Awareness**.

After the UDE PXROS-HR Awareness was enabled, a new menu entry is created in UDE menu **Views – Add-In Windows – µC/OS-II Awareness**. Open the PXROS-HR Awareness by clicking on this menu entry. Depending on the functions enabled in the SAFERTOS configuration header file the µC/OS-II Awareness window immediately shows a snapshot of the current system state:

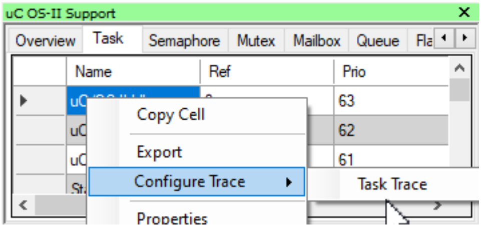


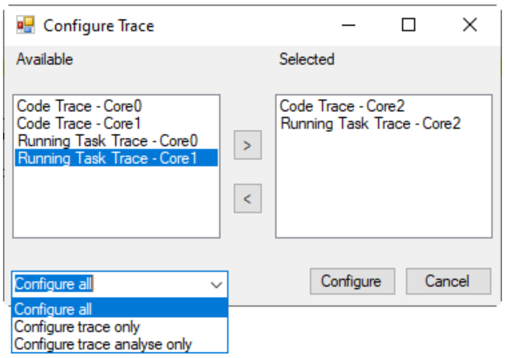
Name	Prio	State	Dly	Stk Ptr	Cur% (D)	Ref
MsgQRxL...	29	Ready	0	0x40073B20	22	13
PostFlags	23	Ready	0	0x40076508	22	18
MsgQTxHi...	7	Ready	0	0x400737...	31	10
BlkMbxQu...	13	Ready	0	0x40072B50	35	7

Please note, because PXROS-HR creates most objects at runtime, not all of them might be displayed, if the system was halted immediately after the first go/step.

µC/OS-II Trace

The task trace can be configured right clicking on any table and selecting **Configure Trace – Task Trace**.





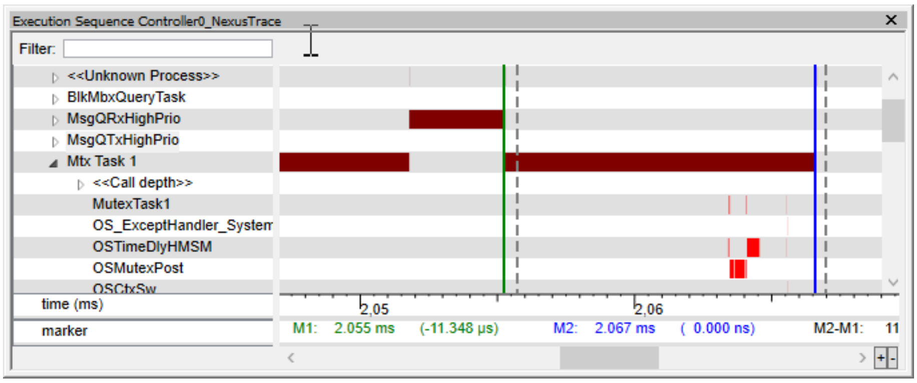
Available:
Code Trace - Core0
Code Trace - Core1
Running Task Trace - Core0
Running Task Trace - Core1

Selected:
Code Trace - Core2
Running Task Trace - Core2

Configure all
Configure all
Configure trace only
Configure trace analyse only

A Dialog shows all selectable cores. Select **Code Trace** and **Running Task Trace** of the core, µC/OS-II is running on.

Start the Trace using the menu Entry **Tools – Start Trace…** and analyze it using the Execution Sequence Chart. Please note, that only tasks, available during configuration, can be named in the Execution Sequence Chart. Other tasks are named after their reference.



Using the µC/OS-II Awareness

To use the µC/OS-II Awareness, load an µC/OS-II application ELF file. Please also note that the µC/OS-II Awareness was developed for µC/OS-II only. µC/OS and µC/OS-III may not be displayed correctly. After the µC/OS-II application was loaded, µC/OS-II must be initialized. This is normally done by the `OSStart()` function.

If the initialization was performed successfully, all system objects are listed in the µC/OS-II Awareness. The Task tab refers to the created tasks as shown above. By setting breakpoints inside the application, the behavior of each task can now be monitored by using the µC/OS-II Awareness.

Refer to the online help (**F1** or **Shift+F1**) for a more detailed description.

A2L File Support

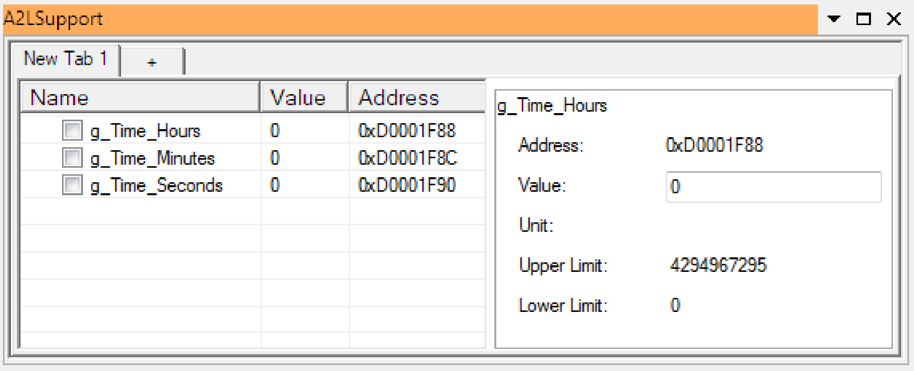
A2L is the file format of the ASAP2 standard, which implements ASAM MCD-2 MC for internal ECU variables, used for measurement and calibration. To activate UDE A2L File Add-In, open the Add-In Manager, menu **Config** – **Add-in Components**. Enable the entry **A2L File Support**.

Using the A2L File Support

After the UDE A2L File Support was enabled, a new menu entry is created in UDE menu **Views – Add-In Windows – A2L File Support**. Open the A2L File Support by clicking on this menu entry.

The context menu of the A2Lsupport window allows loading and unloading an A2L file. After loading the file, select variables from the different A2L sections of the file. By activating a line in the variable list, the variables properties are displayed at the right side of the window. Change the value by editing the text or selecting a different drop-down box item. Invalid (numerical) entries will ignored, if they are of a different data type as the variable, or cut to the upper and lower limit range of the variable. A modified value is written to memory immediately.

If the value of a variable changed otherwise, either manually editing in the the memory window or by the program, the corresponding variable name will be highlighted in red color.



Name	Value	Address
□ g_Time_Hours	0	0xD0001F88
□ g_Time_Minutes	0	0xD0001F8C
□ g_Time_Seconds	0	0xD0001F90

g_Time_Hours

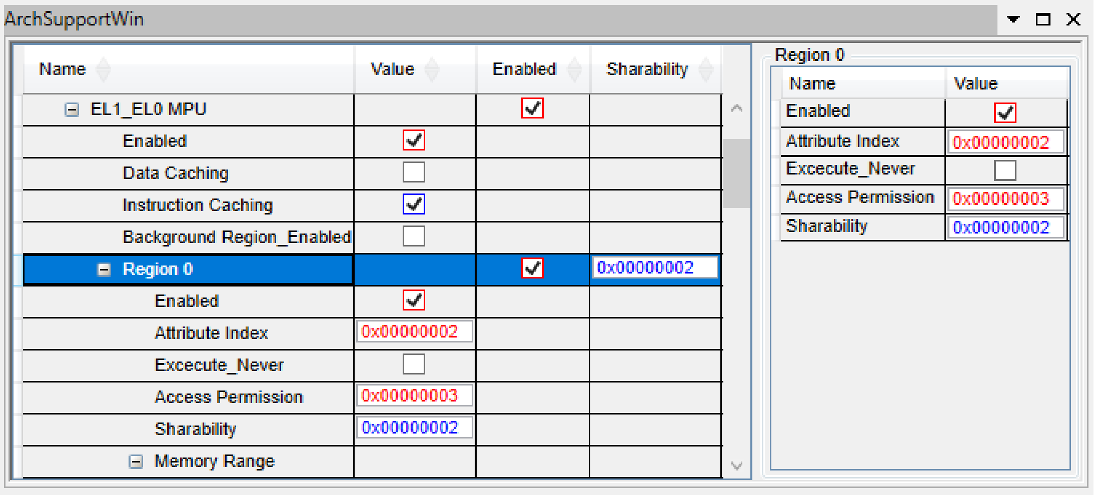
Address:	0xD0001F88
Value:	0
Unit:	
Upper Limit:	4294967295
Lower Limit:	0

Protection Settings

Protection Settings provides the possibility to observe and edit the configuration of all supported Protection Modules (e.g. MPUs).

Using Protection Settings

Use menu **View – Architecture Support** to open the Protection Settings window. The Table shows all available Protection Modules and Sub-Modules as an expandable Tree.



Name	Value	Enabled	Sharability
⊟ EL1_EL0 MPU		☑	
Enabled	☑		
Data Caching	□		
Instruction Caching	☑		
Background Region_Enabled	□		
⊟ Region 0		☑	0x00000002
Enabled	☑		
Attribute Index	0x00000002		
Excecute_Never	□		
Access Permission	0x00000003		
Sharability	0x00000002		
⊟ Memory Range			

Region 0

Name	Value
Enabled	☑
Attribute Index	0x00000002
Excecute_Never	□
Access Permission	0x00000003
Sharability	0x00000002

To display a more detailed view of a Module/Submodule select it in the table, the Detailed View is on the left-hand side. In each of the two views (Table/Detail) values that are modified by the User are displayed blue and those which are changed in the memory since last refresh are displayed red.

Via the context-menu it is possible to manually refresh or apply the settings. In the properties dialog the values that are displayed as columns of the table can be chosen to ensure comparability. It is possible to store and load configurations through the Context Menu.

Activating and Using Add-Ins

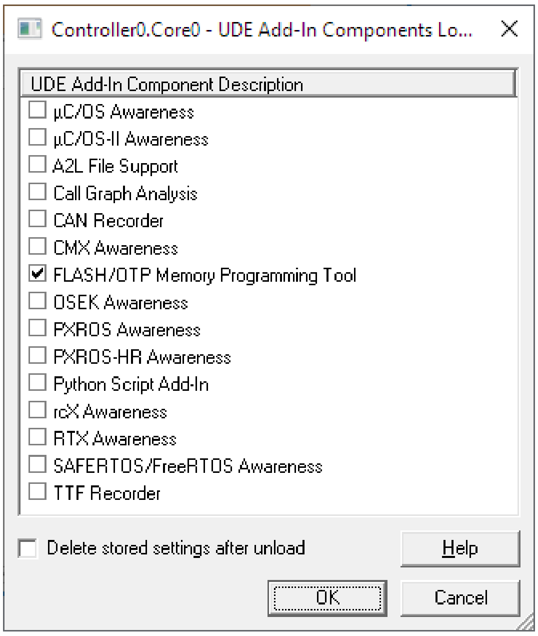
UDE is a component-oriented development workbench. UDE as the working environments of UDE can be extended with Add-Ins to expand the functionality of the UDE development workbench.

Each Add-In can be activated individually for each project workspace. If a new project workspace is created, add-ins are deactivated by default.

Activating an Add-In

To activate an Add-In within your project workspace, follow these steps:

1. Run UDE and load your project workspace.
2. Use menu **Config – Add-In Components** to open the UDE Add-In Manager. The manager dialog shows a list of all UDE Add-Ins that are available within your UDE installation. A checkbox shows whether an Add-In Component is already loaded into the current workspace.
3. Mark the checkbox left of the Add-In to activate.
4. Use the **OK** button to close the Add-In manager dialog.
5. The add-in inserts new menu entries. Please see the reference of the add-in for more information using the add-in.



UDE Add-In Component Description

- ☐ µC/OS Awareness
- ☐ µC/OS-II Awareness
- ☐ A2L File Support
- ☐ Call Graph Analysis
- ☐ CAN Recorder
- ☐ CMX Awareness
- ☑ FLASH/OTP Memory Programming Tool
- ☐ OSEK Awareness
- ☐ PXROS Awareness
- ☐ PXROS-HR Awareness
- ☐ Python Script Add-In
- ☐ rcX Awareness
- ☐ RTX Awareness
- ☐ SAFERTOS/FreeRTOS Awareness
- ☐ TTF Recorder

☐ Delete stored settings after unload

Removing an Add-In

To remove an Add-In from your project workspace, use menu **Config – Add-In Components** to open the UDE Add-In Manager. Now uncheck the entry of the used Add-In.

Eclipse IDE for UDE

Supported Eclipse IDE Versions

UDE provides following Eclipse Integration package:

- UDEEclipse4Integration.zip - supports the Eclipse 4.x based platform.

The selection of the appropriate installation package depends from basic Eclipse version.

Please note, that the Windows ***64-bit Installation Package*** of the appropriate **Eclipse** version must be used for UDE Eclipse Integration plug-in.

Eclipse Version	Required C/C++-Development Plug-In Version	Start Bitmap	Supported by UDE Eclipse Integration Package
Eclipse 4.8 (Photon), Java 8	CDT 9.5		UDEEclipse4Integration.zip
Eclipse 4.27 (2023-03), Java 11	CDT 11.1.0		UDEEclipse4Integration.zip
ST SPC5 Studio Based on Eclipse (Juno)	CDT 8.1 with additional ST CDT extension plug-ins		UDEEclipse4Integration.zip

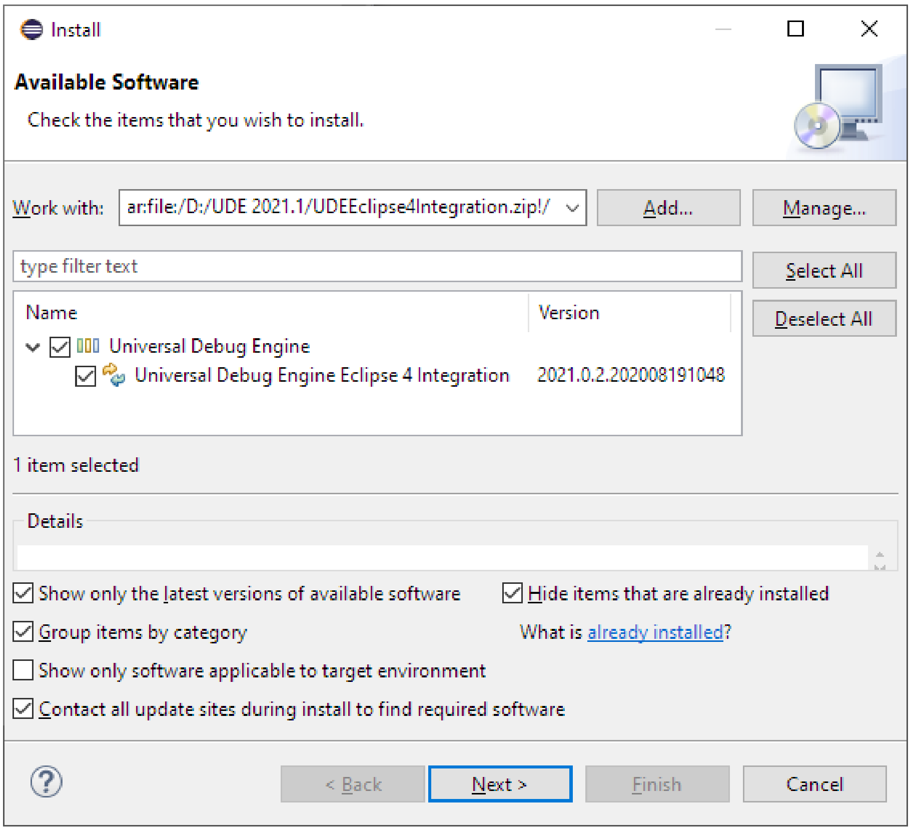
Prepare Eclipse IDE for UDE Integration Package

The UDE Eclipse Integration Package **Eclipse 4.x UDEEclipse4Integration** requires following further environment:

- 64-bit version of Eclipse package including an appropriate CDT package
- 64-bit version of Java JRE 8 (Eclipse 4.8-4.13), JRE 11 (Eclipse 4.13-4.27) or higher.

Install UDE Eclipse Integration Package

1. Start appropriate Eclipse IDE.
2. Open the Software Update dialog of the Eclipse Help menu **Install New Software...**
3. Select button **Add...**
4. Select button **Archive...** and browse for
`<UDE_DIRECTORY>\UDEEclipse4Integration.zip`
for **Eclipse 4.x** IDE from UDE root installation directory.
5. Check **Universal Debug Engine Eclipse 4 Integration** package in the list control of available software packages.



Install

Available Software

Check the items that you wish to install.

Work with: ar:file:/D:/UDE 2021.1/UDEEclipse4Integration.zip!/

type filter text

Name	Version
☑ Universal Debug Engine	
☑ Universal Debug Engine Eclipse 4 Integration	2021.0.2.202008191048

1 item selected

Details

☑ Show only the latest versions of available software

☑ Hide items that are already installed

☑ Group items by category

What is already installed?

☐ Show only software applicable to target environment

☑ Contact all update sites during install to find required software

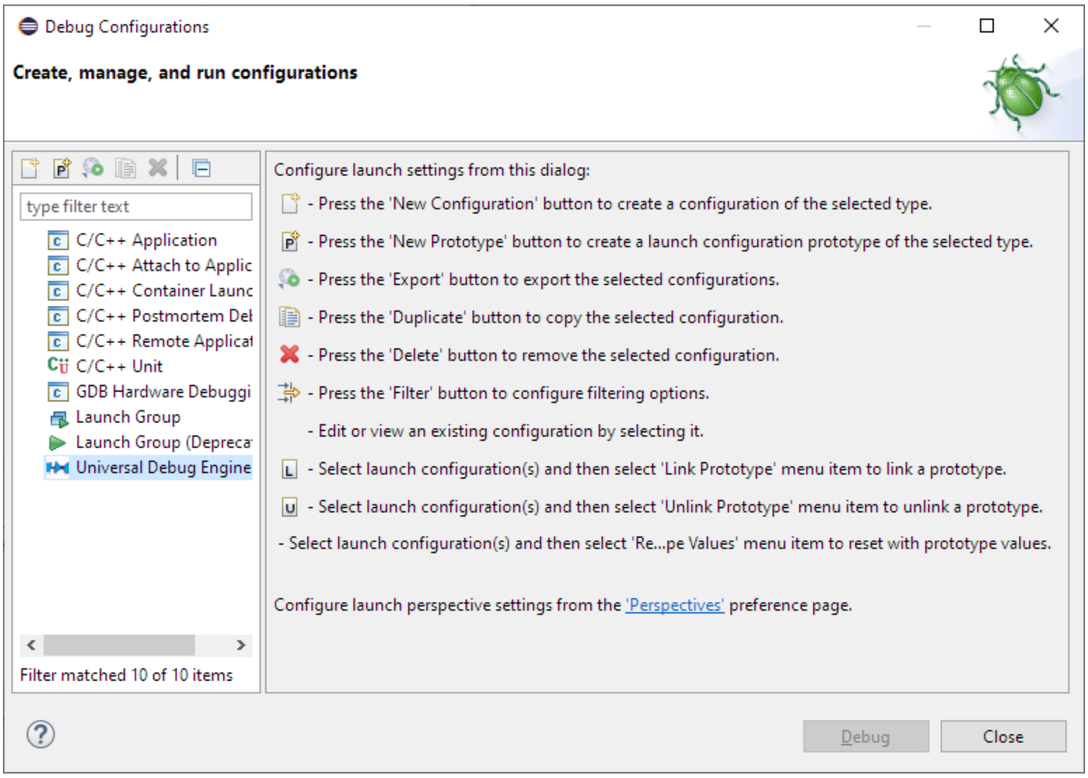
6. Select **Next** button for installation.
7. Press **Install...** button.
8. Finish installation of **Universal Debug Engine Eclipse 4 Integration** feature.

Launching UDE Debug Session in Eclipse IDE

Creating UDE Eclipse Platform Launch Configuration

The integration of Universal Debug Engine into Eclipse IDE for C/C++ developers enables to launch an UDE debug session by a specific debug configuration of a C/C++ project. This launch configuration can be created from local menu of the appropriate C/C++ project:

- Change to C/C++ perspective
- Click with right mouse button into the project tree root of the **Project Explorer** view to open local menu
- Select **Debug As** menu, submenu **Debug Configurations...** to open **Debug Configurations** dialog



Debug Configurations

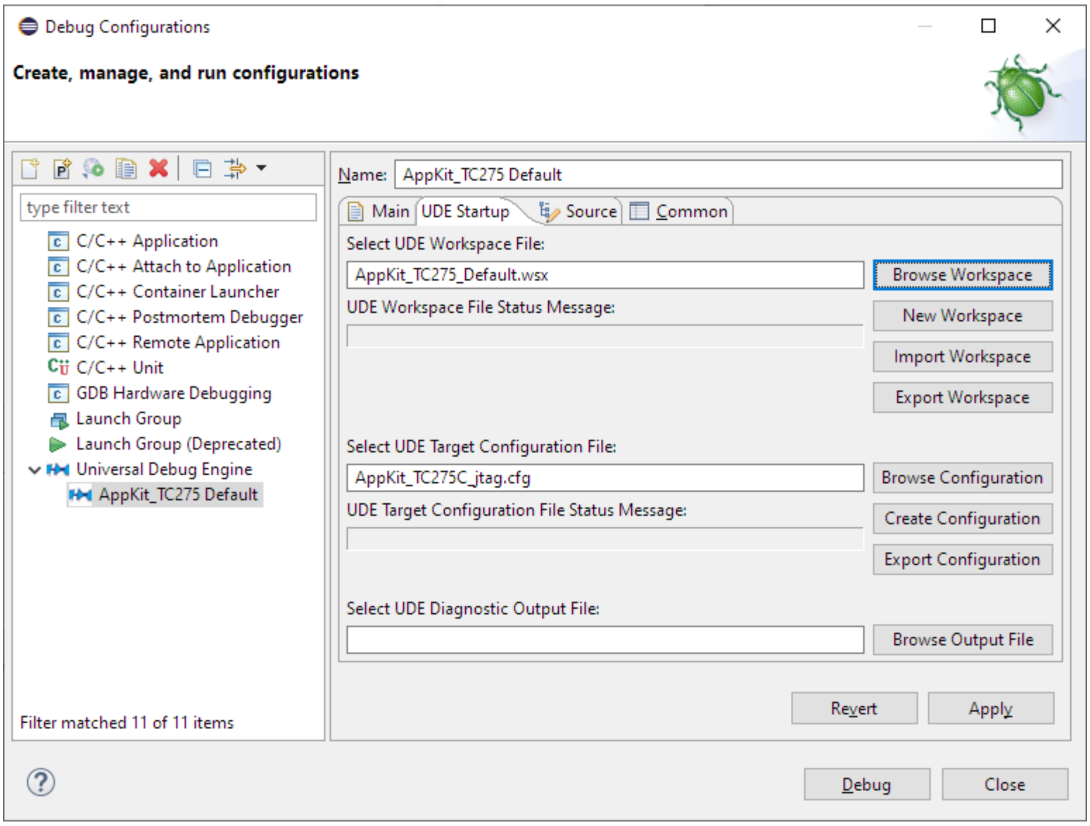
Create, manage, and run configurations

Configure launch settings from this dialog:

- Press the 'New Configuration' button to create a configuration of the selected type.
- Press the 'New Prototype' button to create a launch configuration prototype of the selected type.
- Press the 'Export' button to export the selected configurations.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.
- Select launch configuration(s) and then select 'Link Prototype' menu item to link a prototype.
- Select launch configuration(s) and then select 'Unlink Prototype' menu item to unlink a prototype.
- Select launch configuration(s) and then select 'Re...pe Values' menu item to reset with prototype values.

Configure launch perspective settings from the 'Perspectives' preference page.

- ➢ Select the **Universal Debug Engine** launch configuration type
- ➢ Press the **New Launch Configuration** button to create a new launch configuration for the selected project.



Debug Configurations

Create, manage, and run configurations

Name: AppKit_TC275 Default

Select UDE Workspace File: AppKit_TC275_Default.wsx

UDE Workspace File Status Message:

Select UDE Target Configuration File: AppKit_TC275C_jtag.cfg

UDE Target Configuration File Status Message:

Select UDE Diagnostic Output File:

- ➢ The **Main**, **Source** and **Common** tabs are common elements of each C/C++ launch configuration dialog.
- ➢ The **UDE Startup** tab contains all settings, which are necessary for a specific UDE launch configuration.

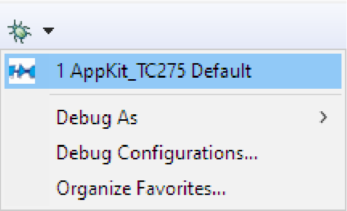
- The two settings for **UDE Workspace File** and **UDE Target Configuration File** are required for a valid UDE launch configuration.
- The selection of an **UDE Diagnostic output file** is optional.

The file system directory of an UDE Workspace File for an UDE Eclipse launch configuration is the C/C++ project directory subdirectory `../.UDE`. The default location of the UDE Target Configuration File is the C/C++ project file system location subdirectory `../.UDE/.TARGET`. The additional functions allow to import or export existing UDE workspaces and UDE target configurations to or from default or common location of Universal Debug Engine from or to the relative paths of the Eclipse C/C++ project locations. A detailed description of all buttons is contained in the UDE helps system.

Steps to start Debug Session with UDE Eclipse Launch Configuration

The newly created UDE launch configuration allows launching a debug session using UDE via **Debug** button of Eclipse IDE tool bar.

Use the **Debug** button of the tool bar of the **Run** menu Debug item (shortcut **F11**) to start the created launch configuration, which opens the UDE perspective, loads the selected workspace file and uses the selected target configuration to connect to the target debug system.

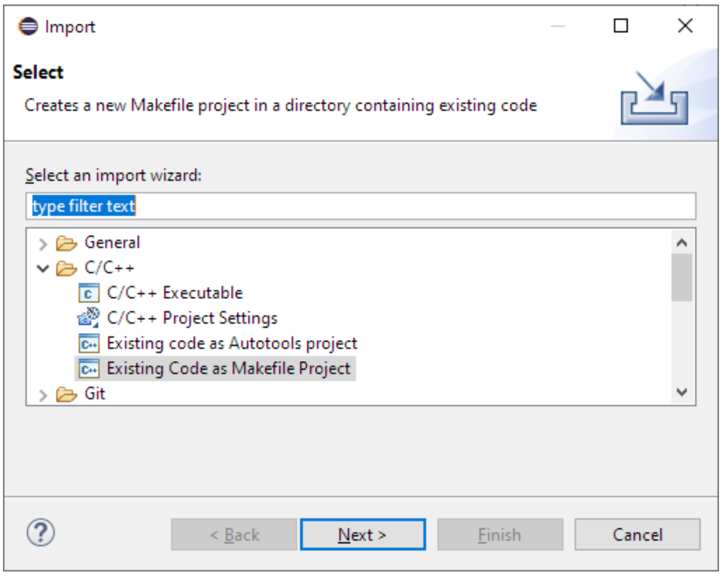


Add UDE Sample Project to Eclipse C/C++ IDE

Creating Eclipse Makefile Project from UDE TimeDemo Sample

UDE sample makefile projects can be easily imported as Eclipse C/C++ projects using the Eclipse import wizard:

- Change to C/C++ perspective
- Open the **Import...** wizard of Eclipse File menu.
- Select C/C++ -> Existing Code as Makefile Project


Import

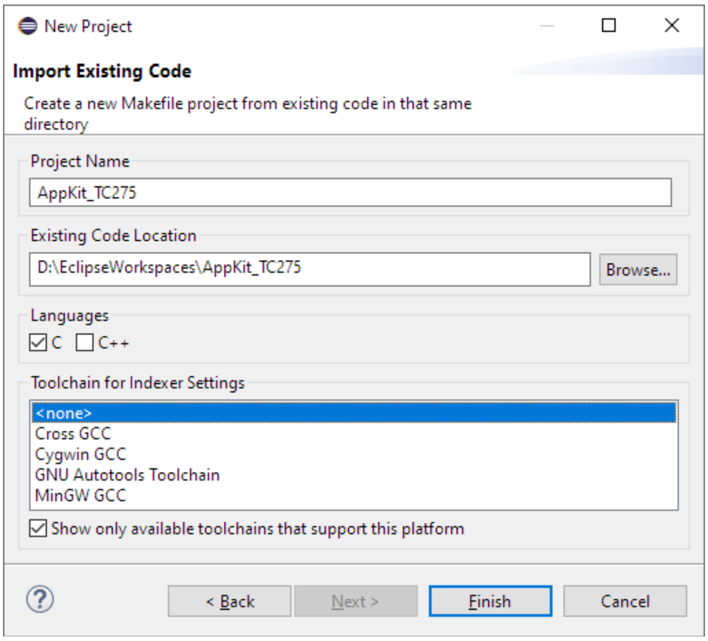
Select

Creates a new Makefile project in a directory containing existing code

Select an import wizard:

type filter text

- On the next wizard page use browse button to select UDE sample directory
- Add an appropriate project name
- Check the Languages checkbox C only
- Select `<none>` as tool chain



New Project

Import Existing Code

Create a new Makefile project from existing code in that same directory

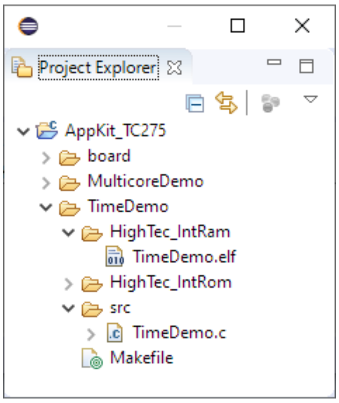
Project Name: AppKit_TC275

Existing Code Location: D:\EclipseWorkspaces\AppKit_TC275

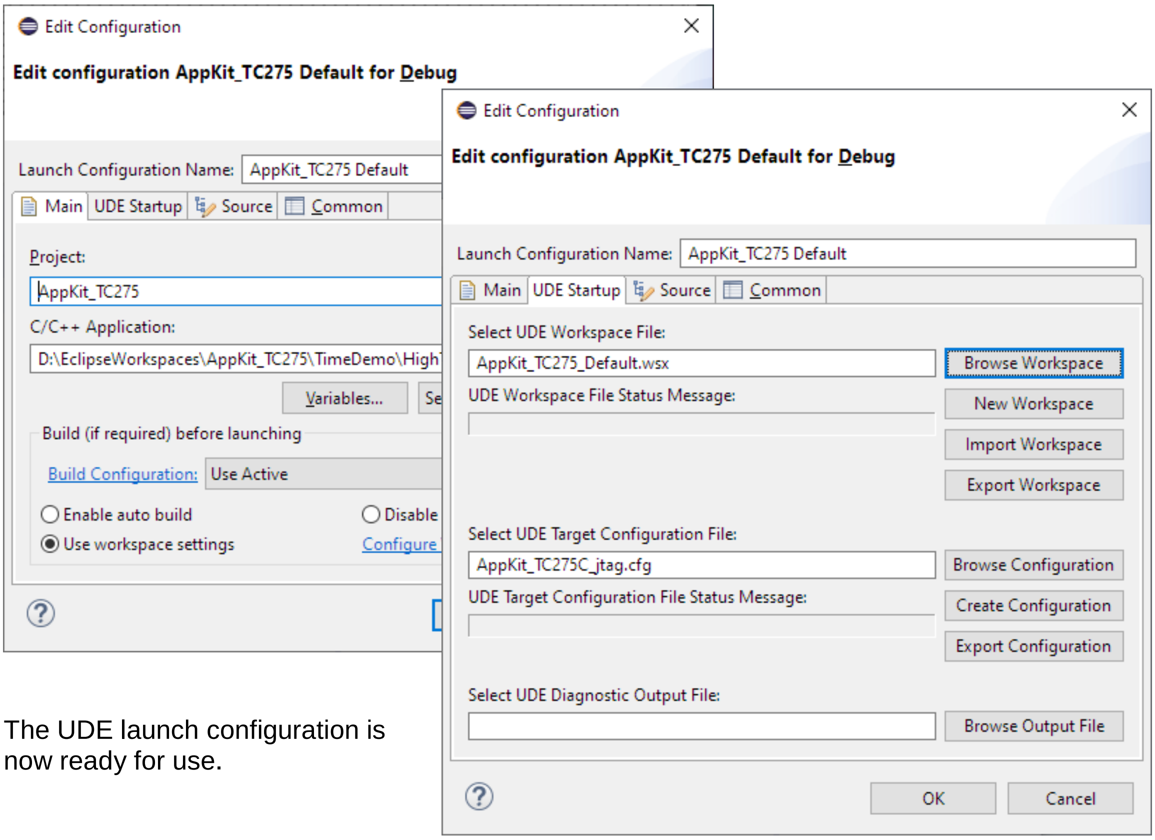
Languages: ☑ C ☐ C++

Toolchain for Indexer Settings: <none>
Cross GCC
Cygwin GCC
GNU Autotools Toolchain
MinGW GCC

☑ Show only available toolchains that support this platform



- Select the context menu of `TimeDemo – HighTec_IntRam – TimeDemo.elf` in the project explorer and **Debug As – Debug Configurations…** to prepare start of debug session
- The **Debug Configurations** dialog **Main** page will be opened to select the type of launch configuration to start debug session
- Select the context menu of **Universal Debug Engine** and **New Configuration**, a new `AppKit_TC275 Default` configuration is created
- **Browse…** the **C/C++ Local Application** to select the binary file (`*.elf/*.out`), which will be loaded by the launch configuration into target system. Select the appropriate binary file of current build configuration.
- Change to the UDE Startup page and check the correct **Select UDE Target Configuration File**. See the chapter above.



Edit Configuration

Edit configuration AppKit_TC275 Default for Debug

Launch Configuration Name: AppKit_TC275 Default

Main | UDE Startup | Source | Common

Project: AppKit_TC275

C/C++ Application: D:\EclipseWorkspaces\AppKit_TC275\TimeDemo\High

Build (if required) before launching

Build Configuration: Use Active

○ Enable auto build ○ Disable
● Use workspace settings Configure

Edit Configuration

Edit configuration AppKit_TC275 Default for Debug

Launch Configuration Name: AppKit_TC275 Default

Main | UDE Startup | Source | Common

Select UDE Workspace File: AppKit_TC275_Default.wsx

UDE Workspace File Status Message:

Select UDE Target Configuration File: AppKit_TC275C_jtag.cfg

UDE Target Configuration File Status Message:

Select UDE Diagnostic Output File:

The UDE launch configuration is now ready for use.

UDE Object Model

Overview

With the UDE object model built on the Microsoft COM technology you can access the target, control the **Universal Debug Engine** and **UDE MemTool**, start and stop programs running on the target, set breakpoints, flashing code and much more. The UDE object model is completely scriptable and supported by a wide range of script languages. The UDE objects are acccessable from external scripts as well from internal scripts in UDE.

A detailed documentation and examples for using the UDE object model can be found at

`<UDE_DIRECTORY>\Help\UDEObjectModel.chm`

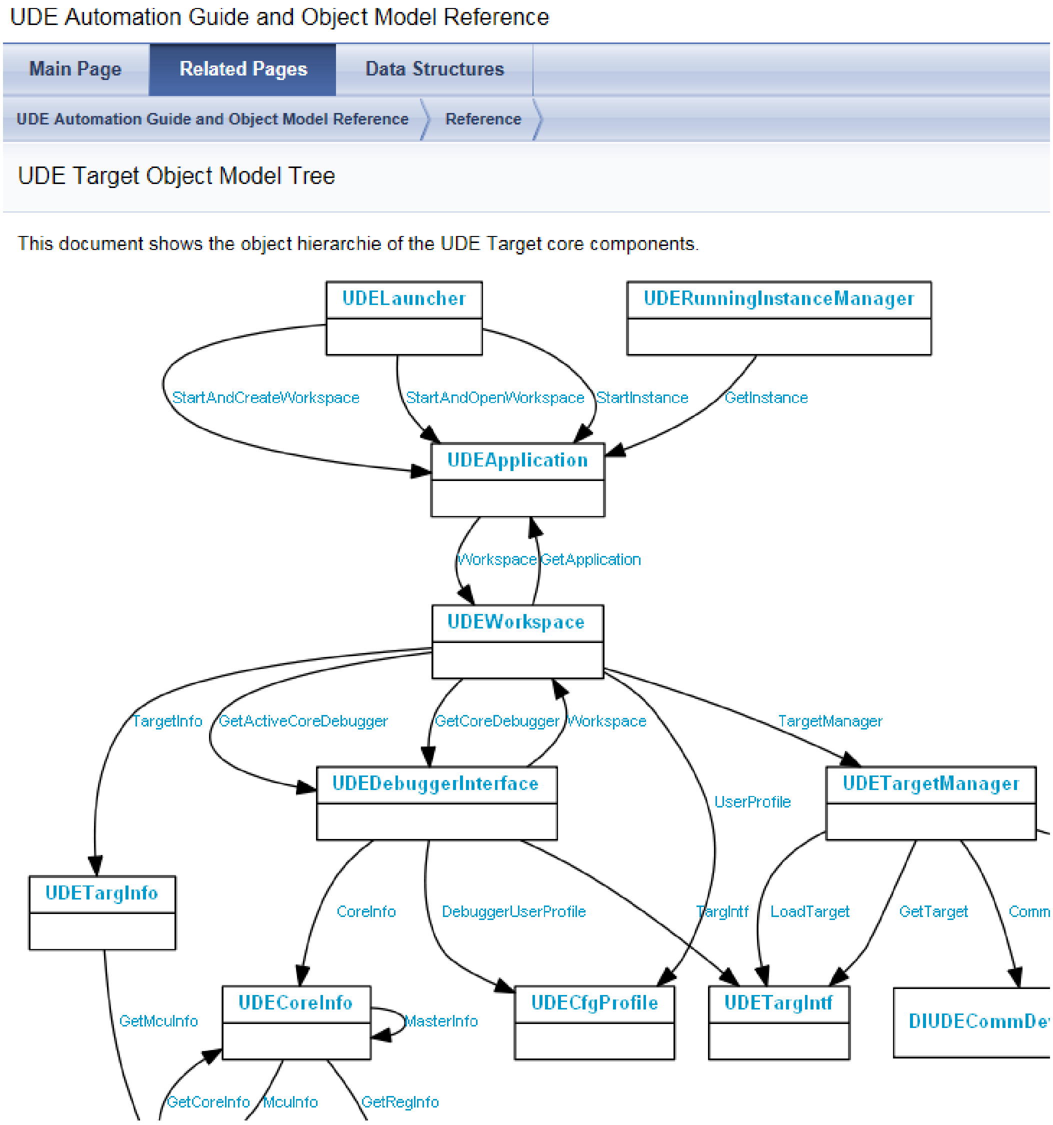
`<UDE_DIRECTORY>\Help\UDEAutomation.chm`

`<UDE_DIRECTORY>\Help\PythonAddOn.chm`

and via the UDE Help system (press **F1**).

Automation Guide and Object Model Reference

The full descriptions and relationships of the UDE objects are explained in the UDE Object Model Help. Open it via menu **Help – Help Index** and open the content **Universal Debug Engine (UDE) – UDE Common Components – UDE Automation Guide and Object Model Reference**.



UDE Automation Guide and Object Model Reference

UDE Target Object Model Tree

This document shows the object hierarchie of the UDE Target core components.

Please contact support@pls-mc.com for more samples of using the UDE object model.

An external Script-Example for TriCore in Python

The following automation example shows how to start, debug and close UDE from an external Python script interpreter.

The external script creates a new workspace (1), connect to the target (2), loads the target application (3), starts the application (4), read a variable value from the target (5), disconnects the target and closes UDE (6).

```python
# UDE automation basics - Python demo script
#   On command line run: "python UdeAutomationDemo.py"

import os
import win32com.client

print("Create new UDELauncher object")
ProgId = "UDELauncher"
UDELauncher = win32com.client.Dispatch(ProgId)
print("  Launcher: " + UDELauncher.Version)

WspFile = os.path.abspath(".") + "\\TC275_ApplicationBoard.wsx"
CfgFile = os.path.abspath(".") + "\\AppKit_TC275C_jtag.cfg"
ElfFile = os.path.abspath(".") + "\\MulticoreDemo.elf"

print("Start new UDE instance")                                     # (1)
UDEApplication = UDELauncher.StartAndCreateWorkspace("UDE.exe", WspFile, \
                                                              CfgFile, 1)
print("  UDE version: " + UDEApplication.VersionInfo)

print("Access workspace")
UDEWorkspace = UDEApplication.Workspace
print("  Workspace: " + UDEWorkspace.ProjectTitle)

print("Wait for target connected")                                  # (2)
if not UDEWorkspace.WaitForTargetConnected(10000): # 10s timeout
    print("ERROR: target not connected !")
UDEDebugger = UDEWorkspace.CoreDebugger(0)

print("Load target application")                                    # (3)
if not UDEDebugger.LoadAndFlash(ElfFile,"VerifyOnly"):
    if not UDEDebugger.LoadAndFlash(ElfFile):
        print("ERROR: failed to load elf file !")

print("Start target application")                                   # (4)

UDEDebugger.Go()
if not UDEWorkspace.WaitForAllCoresRunning(1000, False): # 1s timeout
    print("ERROR: target not running !")

print("Wait a second")
UDEWorkspace.Sleep(1000)

print("Read some variables")                                        # (5)
Seconds = UDEDebugger.ReadVariable("g_SharedData.Seconds")
print("  Time: %02d" % Seconds)

print("Set a breakpoint and wait for halt")
UDEDebugger.Breakpoints.Add("MulticoreDemo.c 352")
UDEDebugger.WriteVariable("g_SharedData.Seconds",58) # trigger halt breakpoint
if not UDEDebugger.WaitForHalt(3000): # 3s timeout
    print("ERROR: not halted at breakpoint !")
print("Release objects and Close UDE instance")                  # (6)
UDEDebugger = None
UDEWorkspace = None
UDELauncher.StopInstance(UDEApplication)

print("Finished")
```

Python Script Console

The Python Script Add-In implements an interactive command line interface to an embedded Python 3 interpreter provided in a UDE Python Script console.

Supported Functions

Input can be entered directly into the console or loaded from external sources, the output is displayed in the UDE Python Script console view. Following functions are supported

- Embedded Python V3.7.0a0 (UDE 2022) / V3.10.4+ (UDE 2023) interpreter
- Inline execution of simple Python statements
- Execution of Python scripts
- Auto-completion of Python keywords and UDE class methods and properties
- Variable inspections
- Callback functions for UDE events
- External dialog interface
- Support of an external Python interpreter.

Enabling the Python Script Console

Open the Python Script Console via UDE menu **Views - Other Windows - Python Script Console**.

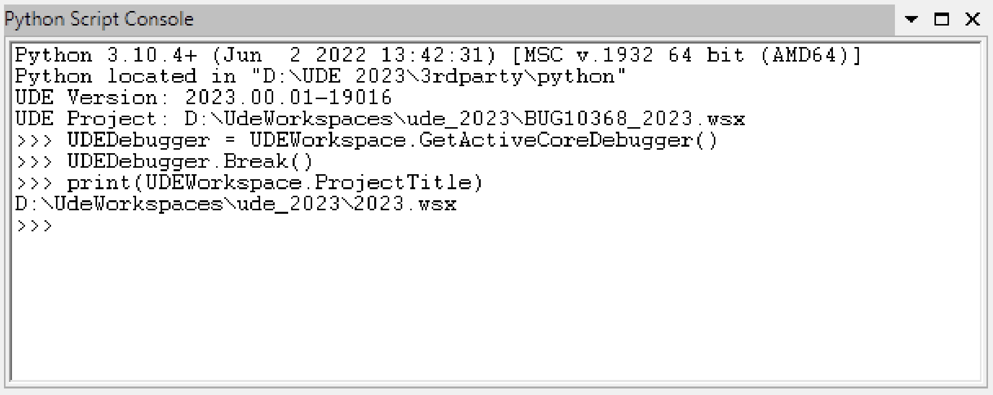
Accessing the UDE object model

The Python script console provides some predefined UDE COM objects. Currently these objects are available:

```
UDEUtilities # .. DIUDEUtilities
UDEApplication # .. IUDEApplication
UDEWorkspace # .. IUDEWorkspace
```

An example of accessing the UDE debugger and break the current program execution:

```
UDEDebugger = UDEWorkspace.GetActiveDebugger()
UDEDebugger.Break()
```



```
Python 3.10.4+ (Jun  2 2022 13:42:31) [MSC v.1932 64 bit (AMD64)]
Python located in "D:\UDE 2023\3rdparty\python"
UDE Version: 2023.00.01-19016
UDE Project: D:\UdeWorkspaces\ude_2023\BUG10368_2023.wsx
>>> UDEDebugger = UDEWorkspace.GetActiveCoreDebugger()
>>> UDEDebugger.Break()
>>> print(UDEWorkspace.ProjectTitle)
D:\UdeWorkspaces\ude_2023\2023.wsx
>>>
```

Refer to the online help (**F1**) for a more detailed description.

User Definable Enhancements

The UDE concept allows extending the debugger functions by additional components in the following levels:

- HTML scripts based on UDE ActiveX™ controls, customer specific controls and access via the UDE object model to display consumer-defined windows with UDE HTML windows server. For a description of these features, see the chapter about Viewing and Modifying Registers and the UDE object model.
- Programming a custom view server can solve complex problems and can provide detailed and optimized target related windows, e.g. for integrating specific RTOS, CAN windows.
- The UDE client may be substituted by a customer GUI for production and field maintenance purposes, e.g. for Service Tools, Matlab or CASE tool integration. The functionality of the core debugger can be extended or even reduced to enable essential windows and features only. Please see for further information the chapter about the UDE object model.

Please contact **support@pls-mc.com** for more information and examples about the UDE extensions.

Reference

The Reference of the Universal Debug Engine (UDE) is available via the Window's Help system of UDE. To use the help, please open UDE and press **F1** or browse to the UDE installation menu and select in UDE menu **Help**.

Copyrights

List of Open Source Software Components

This chapter contains a list of open source software (OSS) components used within the product under the terms of the respective licenses. The source code corresponding to the open source components is also provided along with the product wherever mandated by the respective OSS license.

MCD Software License: *ARM Ltd, Infineon Technologies, NXP, Lauterbach, STMicroelectronics, TIMA Laboratory*

Copyright (c) 2008, ARM Ltd., Infineon Technologies, NXP Semiconductors, Lauterbach, STMicroelectronics and TIMA Laboratory. All rights reserved. PREAMBLE The MCD API (Multi-Core Debug) has been designed as an interface between software development tools and simulated or real systems with multi-core SoCs. The target is to allow consistent software tooling throughout the whole SoC development flow. The MCD API (the "SOFTWARE") has been developed jointly by ARM Ltd., Infineon Technologies, NXP Semiconductors, Lauterbach, STMicroelectronics and TIMA Laboratory as part of the SPRINT project (www.sprint-project.net). The SPRINT project has been funded by the European Commission. LICENSE Any redistribution and use of the SOFTWARE in source and binary forms, with or without modification constitutes the full acceptance of the following disclaimer as well as of the license herein and is permitted provided that the following conditions are met: - Redistributions of source code must retain the above copyright notice, this list of conditions and the disclaimer detailed below. - Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the disclaimer detailed below in the documentation and/or other materials provided with the distribution. - Neither the name of its copyright holders nor the names of its contributors may be used to endorse or promote products derived from the Software without specific prior written permission. - Modification of any or all of the source code, documentation and other materials provided under this license are subject to acknowledgement of the modification(s) by including a prominent notice on the modification(s) stating the change(s) to the file(s), identifying the date of such change and stating the name of the publisher of any such modification(s). DISCLAIMER OF WARRANTY AND LIABILITY THE SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE, MISREPRESENTATION OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Demangle Software License: *Free Software Foundation*

Copyright 1992, 1993, 1994, 1995, 1996, 1997, 1998, 2000, 2001, 2002, 2003, 2004, 2005, 2007 Free Software Foundation, Inc. This program is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version. In addition to the permissions in the GNU Library General Public License, the Free Software Foundation gives you unlimited permission to link the compiled version of this file into combinations with other programs, and to distribute those combinations without any restriction coming from the use of this file. (The Library Public License restrictions do apply in other respects; for example, they cover modification of the file, and distribution when not linked into a combined executable.) This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details. You should have received a copy of the GNU Library General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street - Fifth Floor, Boston, MA 02110-1301, USA

In addition to the permissions in the GNU General Public License, the Free Software Foundation gives you unlimited permission to link the compiled version of this file into combinations with other programs, and to distribute those combinations without any restriction coming from the use of this file. (The General Public License restrictions do apply in other respects; for example, they cover modification of the file, and distribution when not linked into a combined executable.) **The files 'cp-demangle.cpp', 'cp-demangle.h' and 'demangle.h', which are published using this license, are used in UDE program only unmodified and the compiled versions of these files are linked into combinations with other program parts.**

On request we deliver the source code of these files. GNU GENERAL PUBLIC LICENSE Version 2, June 1991 Copyright (C) 1989, 1991 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. Preamble The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too. When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things. To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it. For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights. We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software. Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations. Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all. The precise terms and conditions for copying, distribution and modification follow. TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION 0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you". Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does. 1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program. You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee. 2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions: a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change. b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License. c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.) These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it. Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program. In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License. 3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following: a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or, b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or, c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.) The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable. If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code. 4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance. 5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it. 6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License. 7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program. If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances. It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice. This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License. 8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License. 9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation. 10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally. NO WARRANTY 11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. 12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. END OF TERMS AND CONDITIONS

JPEG Software License: Thomas G. Lane - JPEG Group

Copyright (C) 1991-1995, Thomas G. Lane. (https://jpegclub.org/reference/libjpeg-license/) This file is part of the Independent JPEG Group's software. For conditions of distribution and use, see the accompanying README file. This file defines the application interface for the JPEG library. Most applications using the library need only include this file, and perhaps jerror.h if they want to know the exact error codes. LICENSE TERMS (ships as a part of the libjpeg package in the README file) ============ 1. We don't promise that this software works. (But if you find any bugs, please let us know!) 2. You can use this software for whatever you want. You don't have to pay us. 3. You may not pretend that you wrote this software. If you use it in a program, you must acknowledge somewhere in your documentation that you've used the IJG code. In legalese: The authors make NO WARRANTY or representation, either express or implied, with respect to this software, its quality, accuracy, merchantability, or fitness for a particular purpose. This software is provided "AS IS", and you, its user, assume the entire risk as to its quality and accuracy. This software is copyright (C) 1991-2016, Thomas G. Lane, Guido Vollbeding. All Rights Reserved except as specified below. Permission is hereby granted to use, copy, modify, and distribute this software (or portions thereof) for any purpose, without fee, subject to these conditions: (1) If any part of the source code for this software is distributed, then this README file must be included, with this copyright and no-warranty notice unaltered; and any additions, deletions, or changes to the original files must be clearly indicated in accompanying documentation. (2) If only executable code is distributed, then the accompanying documentation must state that "this software is based in part on the work of the Independent JPEG Group". (3) Permission for use of this software is granted only if the user accepts full responsibility for any undesirable consequences; the authors accept NO LIABILITY for damages of any kind. These conditions apply to any software derived from or based on the IJG code, not just to the unmodified library.

If you use our work, you ought to acknowledge us. Permission is NOT granted for the use of any IJG author's name or company name in advertising or publicity relating to this software or products derived from it. This software may be referred to only as "the Independent JPEG Group's software". We specifically permit and encourage the use of this software as the basis of commercial products, provided that all warranty or liability claims are assumed by the product vendor.

Resizable Elements Software License: *Paolo Messina*

Copyright (C) 2000-2002 by Paolo Messina (http://www.geocities.com/ppescher - ppescher@yahoo.com) The contents of this file are subject to the Artistic License (the "License"). You may not use this file except in compliance with the License. You may obtain a copy of the License at: https://opensource.org/license/artistic-license-2-0-php/ If you find this code useful, credits would be nice! SPDX short identifier: Artistic-2.0 Copyright (c) 2000-2006, The Perl Foundation. Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. Preamble This license establishes the terms under which a given free software Package may be copied, modified, distributed, and/or redistributed. The intent is that the Copyright Holder maintains some artistic control over the development of that Package while still keeping the Package available as open source and free software. You are always permitted to make arrangements wholly outside of this license directly with the Copyright Holder of a given Package. If the terms of this license do not permit the full use that you propose to make of the Package, you should contact the Copyright Holder and seek a different licensing arrangement. Definitions “Copyright Holder” means the individual(s) or organization(s) named in the copyright notice for the entire Package. “Contributor” means any party that has contributed code or other material to the Package, in accordance with the Copyright Holder’s procedures. “You” and “your” means any person who would like to copy, distribute, or modify the Package. “Package” means the collection of files distributed by the Copyright Holder, and derivatives of that collection and/or of those files. A given Package may consist of either the Standard Version, or a Modified Version. “Distribute” means providing a copy of the Package or making it accessible to anyone else, or in the case of a company or organization, to others outside of your company or organization. “Distributor Fee” means any fee that you charge for Distributing this Package or providing support for this Package to another party. It does not mean licensing fees. “Standard Version” refers to the Package if it has not been modified, or has been modified only in ways explicitly requested by the Copyright Holder. “Modified Version” means the Package, if it has been changed, and such changes were not explicitly requested by the Copyright Holder. “Original License” means this Artistic License as Distributed with the Standard Version of the Package, in its current version or as it may be modified by The Perl Foundation in the future. “Source” form means the source code, documentation source, and configuration files for the Package. Compiled” form means the compiled bytecode, object code, binary, or any other form resulting from mechanical transformation or translation of the Source form. Permission for Use and Modification Without Distribution (1) You are permitted to use the Standard Version and create and use Modified Versions for any purpose without restriction, provided that you do not Distribute the Modified Version. Permissions for Redistribution of the Standard Version (2) You may Distribute verbatim copies of the Source form of the Standard Version of this Package in any medium without restriction, either gratis or for a Distributor Fee, provided that you duplicate all of the original copyright notices and associated disclaimers. At your discretion, such verbatim copies may or may not include a Compiled form of the Package. (3) You may apply any bug fixes, portability changes, and other modifications made available from the Copyright Holder. The resulting Package will still be considered the Standard Version, and as such will be subject to the Original License. Distribution of Modified Versions of the Package as Source (4) You may Distribute your Modified Version as Source (either gratis or for a Distributor Fee, and with or without a Compiled form of the Modified Version) provided that you clearly document how it differs from the Standard Version, including, but not limited to, documenting any non-standard features, executables, or modules, and provided that you do at least ONE of the following: (a) make the Modified Version available to the Copyright Holder of the Standard Version, under the Original License, so that the Copyright Holder may include your modifications in the Standard Version. (b) ensure that installation of your Modified Version does not prevent the user installing or running the Standard Version. In addition, the Modified Version must bear a name that is different from the name of the Standard Version. (c) allow anyone who receives a copy of the Modified Version to make the Source form of the Modified Version available to others under (i) the Original License or (ii) a license that permits the licensee to freely copy, modify and redistribute the Modified Version using the same licensing terms that apply to the copy that the licensee received, and requires that the Source form of the Modified Version, and of any works derived from it, be made freely available in that license fees are prohibited but Distributor Fees are allowed. Distribution of Compiled Forms of the Standard Version or Modified Versions without the Source (5) You may Distribute Compiled forms of the Standard Version without the Source, provided that you include complete instructions on how to get the Source of the Standard Version. Such instructions must be valid at the time of your distribution. If these instructions, at any time while you are carrying out such distribution, become invalid, you must provide new instructions on demand or cease further distribution. If you provide valid instructions or cease distribution within thirty days after you become aware that the instructions are invalid, then you do not forfeit any of your rights under this license. (6) You may Distribute a Modified Version in Compiled form without the Source, provided that you comply with Section 4 with respect to the Source of the Modified Version. Aggregating or Linking the Package (7) You may aggregate the Package (either the Standard Version or Modified Version) with other packages and Distribute the resulting aggregation provided that you do not charge a licensing fee for the Package. Distributor Fees are permitted, and licensing fees for other components in the aggregation are permitted. The terms of this license apply to the use and Distribution of the Standard or Modified Versions as included in the aggregation. (8) You are permitted to link Modified and Standard Versions with other works, to embed the Package in a larger work of your own, or to build stand-alone binary or bytecode versions of applications that include the Package, and Distribute the result without restriction, provided the result does not expose a direct interface to the Package. Items That are Not Considered Part of a Modified Version (9) Works (including, but not limited to, modules and scripts) that merely extend or make use of the Package, do not, by themselves, cause the Package to be a Modified Version. In addition, such works are not considered parts of the Package itself, and are not subject to the terms of this license. General Provisions (10) Any use, modification, and distribution of the Standard or Modified Versions is governed by this Artistic License. By using, modifying or distributing the Package, you accept this license. Do not use, modify, or distribute the Package, if you do not accept this license. (11) If your Modified Version has been derived from a Modified Version made by someone other than you, you are nevertheless required to ensure that your Modified Version complies with the requirements of this license. (12) This license does not grant you the right to use any trademark, service mark, tradename, or logo of the Copyright Holder. (13) This license includes the non-exclusive, worldwide, free-of-charge patent license to make, have made, use, offer to sell, sell, import and otherwise transfer the Package with respect to any patent claims licensable by the Copyright Holder that are necessarily infringed by the Package. If you institute patent litigation (including a cross-claim or counterclaim) against any party alleging that the Package constitutes direct or contributory patent infringement, then this Artistic License to you shall terminate on the date that such litigation is filed. (14) Disclaimer of Warranty: THE PACKAGE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS “AS IS’ AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES. THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT ARE DISCLAIMED TO THE EXTENT PERMITTED BY YOUR LOCAL LAW. UNLESS REQUIRED BY LAW, NO COPYRIGHT HOLDER OR CONTRIBUTOR WILL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING IN ANY WAY OUT OF THE USE OF THE PACKAGE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

VB2PY Software License: *Paul Paterson*

Copyright (c) 2003, Paul Paterson All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of the vb2Py Project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

CSPGen Software License: *Sun Microsystems, Inc.*

Copyright (c) 2006, Sun Microsystems, Inc. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of the Sun Microsystems, Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF

LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

TreePropSheet Software License: *Yves Tkaczyk*

Copyright (C) 2004 by Yves Tkaczyk (http://www.tkaczyk.net - yves@tkaczyk.net) The contents of this file are subject to the Artistic License (the "License"). You may not use this file except in compliance with the License. You may obtain a copy of the License at: https://opensource.org/license/artistic-license-2-0-php/ SPDX short identifier: Artistic-2.0 Copyright (c) 2000-2006, The Perl Foundation. Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. Preamble This license establishes the terms under which a given free software Package may be copied, modified, distributed, and/or redistributed. The intent is that the Copyright Holder maintains some artistic control over the development of that Package while still keeping the Package available as open source and free software. You are always permitted to make arrangements wholly outside of this license directly with the Copyright Holder of a given Package. If the terms of this license do not permit the full use that you propose to make of the Package, you should contact the Copyright Holder and seek a different licensing arrangement. Definitions “Copyright Holder” means the individual(s) or organization(s) named in the copyright notice for the entire Package. “Contributor” means any party that has contributed code or other material to the Package, in accordance with the Copyright Holder’s procedures. “You” and “your” means any person who would like to copy, distribute, or modify the Package. “Package” means the collection of files distributed by the Copyright Holder, and derivatives of that collection and/or of those files. A given Package may consist of either the Standard Version, or a Modified Version. “Distribute” means providing a copy of the Package or making it accessible to anyone else, or in the case of a company or organization, to others outside of your company or organization. “Distributor Fee” means any fee that you charge for Distributing this Package or providing support for this Package to another party. It does not mean licensing fees. “Standard Version” refers to the Package if it has not been modified, or has been modified only in ways explicitly requested by the Copyright Holder. “Modified Version” means the Package, if it has been changed, and such changes were not explicitly requested by the Copyright Holder. “Original License” means this Artistic License as Distributed with the Standard Version of the Package, in its current version or as it may be modified by The Perl Foundation in the future. “Source” form means the source code, documentation source, and configuration files for the Package. “Compiled” form means the compiled bytecode, object code, binary, or any other form resulting from mechanical transformation or translation of the Source form. Permission for Use and Modification Without Distribution (1) You are permitted to use the Standard Version and create and use Modified Versions for any purpose without restriction, provided that you do not Distribute the Modified Version. Permissions for Redistribution of the Standard Version (2) You may Distribute verbatim copies of the Source form of the Standard Version of this Package in any medium without restriction, either gratis or for a Distributor Fee, provided that you duplicate all of the original copyright notices and associated disclaimers. At your discretion, such verbatim copies may or may not include a Compiled form of the Package. (3) You may apply any bug fixes, portability changes, and other modifications made available from the Copyright Holder. The resulting Package will still be considered the Standard Version, and as such will be subject to the Original License. Distribution of Modified Versions of the Package as Source (4) You may Distribute your Modified Version as Source (either gratis or for a Distributor Fee, and with or without a Compiled form of the Modified Version) provided that you clearly document how it differs from the Standard Version, including, but not limited to, documenting any non-standard features, executables, or modules, and provided that you do at least ONE of the following: (a) make the Modified Version available to the Copyright Holder of the Standard Version, under the Original License, so that the Copyright Holder may include your modifications in the Standard Version. (b) ensure that installation of your Modified Version does not prevent the user installing or running the Standard Version. In addition, the Modified Version must bear a name that is different from the name of the Standard Version. (c) allow anyone who receives a copy of the Modified Version to make the Source form of the Modified Version available to others under (i) the Original License or (ii) a license that permits the licensee to freely copy, modify and redistribute the Modified Version using the same licensing terms that apply to the copy that the licensee received, and requires that the Source form of the Modified Version, and of any works derived from it, be made freely available in that license fees are prohibited but Distributor Fees are allowed. Distribution of Compiled Forms of the Standard Version or Modified Versions without the Source (5) You may Distribute Compiled forms of the Standard Version without the Source, provided that you include complete instructions on how to get the Source of the Standard Version. Such instructions must be valid at the time of your distribution. If these instructions, at any time while you are carrying out such distribution, become invalid, you must provide new instructions on demand or cease further distribution. If you provide valid instructions or cease distribution within thirty days after you become aware that the instructions are invalid, then you do not forfeit any of your rights under this license. (6) You may Distribute a Modified Version in Compiled form without the Source, provided that you comply with Section 4 with respect to the Source of the Modified Version. Aggregating or Linking the Package (7) You may aggregate the Package (either the Standard Version or Modified Version) with other packages and Distribute the resulting aggregation provided that you do not charge a licensing fee for the Package. Distributor Fees are permitted, and licensing fees for other components in the aggregation are permitted. The terms of this license apply to the use and Distribution of the Standard or Modified Versions as included in the aggregation. (8) You are permitted to link Modified and Standard Versions with other works, to embed the Package in a larger work of your own, or to build stand-alone binary or bytecode versions of applications that include the Package, and Distribute the result without restriction, provided the result does not expose a direct interface to the Package. Items That are Not Considered Part of a Modified Version (9) Works (including, but not limited to, modules and scripts) that merely extend or make use of the Package, do not, by themselves, cause the Package to be a Modified Version. In addition, such works are not considered parts of the Package itself, and are not subject to the terms of this license. General Provisions (10) Any use, modification, and distribution of the Standard or Modified Versions is governed by this Artistic License. By using, modifying or distributing the Package, you accept this license. Do not use, modify, or distribute the Package, if you do not accept this license. (11) If your Modified Version has been derived from a Modified Version made by someone other than you, you are nevertheless required to ensure that your Modified Version complies with the requirements of this license. (12) This license does not grant you the right to use any trademark, service mark, tradename, or logo of the Copyright Holder. (13) This license includes the non-exclusive, worldwide, free-of-charge patent license to make, have made, use, offer to sell, sell, import and otherwise transfer the Package with respect to any patent claims licensable by the Copyright Holder that are necessarily infringed by the Package. If you institute patent litigation (including a cross-claim or counterclaim) against any party alleging that the Package constitutes direct or contributory patent infringement, then this Artistic License to you shall terminate on the date that such litigation is filed. (14) Disclaimer of Warranty: THE PACKAGE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS “AS IS’ AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES. THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT ARE DISCLAIMED TO THE EXTENT PERMITTED BY YOUR LOCAL LAW. UNLESS REQUIRED BY LAW, NO COPYRIGHT HOLDER OR CONTRIBUTOR WILL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING IN ANY WAY OUT OF THE USE OF THE PACKAGE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Avalon Dock Software License: *Dirkster99/AvalonDock*

Ms-PL Microsoft Public License (Ms-PL) This license governs use of the accompanying software. If you use the software, you accept this license. If you do not accept the license, do not use the software. 1. Definitions The terms "reproduce," "reproduction," "derivative works," and "distribution" have the same meaning here as under U.S. copyright law. A "contribution" is the original software, or any additions or changes to the software. A "contributor" is any person that distributes its contribution under this license. "Licensed patents" are a contributor's patent claims that read directly on its contribution. 2. Grant of Rights (A) Copyright Grant- Subject to the terms of this license, including the license conditions and limitations in section 3, each contributor grants you a non-exclusive, worldwide, royalty-free copyright license to reproduce its contribution, prepare derivative works of its contribution, and distribute its contribution or any derivative works that you create. (B) Patent Grant- Subject to the terms of this license, including the license conditions and limitations in section 3, each contributor grants you a non-exclusive, worldwide, royalty-free license under its licensed patents to make, have made, use, sell, offer for sale, import, and/or otherwise dispose of its contribution in the software or derivative works of the contribution in the software. 3. Conditions and Limitations (A) No Trademark License- This license does not grant you rights to use any contributors' name, logo, or trademarks. (B) If you bring a patent claim against any contributor over patents that you claim are infringed by the software, your patent license from such contributor to the software ends automatically. (C) If you distribute any portion of the software, you must retain all copyright, patent, trademark, and attribution notices that are present in the software. (D) If you distribute any portion of the software in source code form, you may do so only under this license by including a complete copy of this license with your distribution. If you distribute any portion of the software in compiled or object code form, you may only do so under a license that complies with this license. (E) The software is licensed "as-is." You bear the risk of using it. The contributors give no express warranties, guarantees or conditions. You may have additional consumer rights under your local laws which this license cannot change. To the extent permitted under your local laws, the contributors exclude the implied warranties of merchantability, fitness for a particular purpose and non-infringement.

Python Software License: *Python Software Foundation*

PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2 1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using this software ("Python") in source or binary form and its associated documentation. 2. Subject to the terms and conditions of this License Agreement, PSF hereby grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use Python alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright (c) 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022 Python Software Foundation; All

Rights Reserved" are retained in Python alone or in any derivative version prepared by Licensee. 3. In the event Licensee prepares a derivative work that is based on or incorporates Python or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to Python. 4. PSF is making Python available to Licensee on an "AS IS" basis. PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF PYTHON WILL NOT INFRINGE ANY THIRD PARTY RIGHTS. 5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. 6. This License Agreement will automatically terminate upon a material breach of its terms and conditions. 7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between PSF and Licensee. This License Agreement does not grant permission to use PSF trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party. 8. By copying, installing or otherwise using Python, Licensee agrees to be bound by the terms and conditions of this License Agreement.

SQLite Consortium: *SQLite Is Public Domain*

All of the code and documentation in SQLite has been dedicated to the public domain by the authors. All code authors, and representatives of the companies they work for, have signed affidavits dedicating their contributions to the public domain and originals of those signed affidavits are stored in a firesafe at the main offices of Hwaci. All contibutors are citizens of countries that allow creative works to be dedicated into the public domain. Anyone is free to copy, modify, publish, use, compile, sell, or distribute the original SQLite code, either in source code form or as a compiled binary, for any purpose, commercial or non-commercial, and by any means. The previous paragraph applies to the deliverable code and documentation in SQLite - those parts of the SQLite library that you actually bundle and ship with a larger application. Some scripts used as part of the build process (for example the "configure" scripts generated by autoconf) might fall under other open-source licenses. Nothing from these build scripts ever reaches the final deliverable SQLite library, however, and so the licenses associated with those scripts should not be a factor in assessing your rights to copy and use the SQLite library. All of the deliverable code in SQLite has been written from scratch. No code has been taken from other projects or from the open internet. Every line of code can be traced back to its original author, and all of those authors have public domain dedications on file. So the SQLite code base is clean and is uncontaminated with licensed code from other projects. The author of the 'Public Domain' - Software 'SQLite' is: Hipp, Wyrick & Company, Inc. 6200 Maple Cove Lane, Charlotte, NC 28269 USA +1.704.948.4565 www.hwaci.com

MDFLib Software License: *Michael Bührer & Bernd Sparrer*

Copyright 2011 Michael Bührer & Bernd Sparrer. All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. THIS SOFTWARE IS PROVIDED BY Michael Bührer & Bernd Sparrer ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL Michael Bührer OR Bernd Sparrer OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. The views and conclusions contained in the software and documentation are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of Michael Bührer & Bernd Sparrer.

GreenWaves GAP8 IoT Register Definition License: *GreenWaves Technologies SAS*

Copyright (c) 2017 GreenWaves Technologies SAS All rights reserved. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. Neither the name of GreenWaves Technologies SAS nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Kinetis FLASH Driver License: *Freescale Semiconductor, Inc.*

(c) Copyright 2010-2014 Freescale Semiconductor, Inc. | Freescale Semiconductor License ALL RIGHTS RESERVED. Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. * Neither the name of the <organization> nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL <COPYRIGHT HOLDER> BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Index