

# UDE® Trace Support

## UDE® Trace Data Visualization and Analyzing

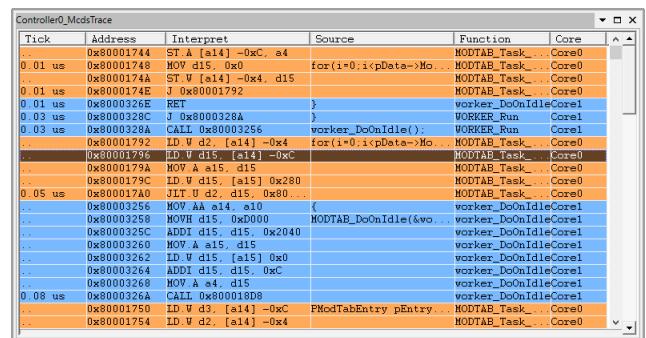
Hard real-time and multicore applications with parallel code execution inherently do not allow you to influence the runtime behavior during debugging. Trace is an appropriate and powerful method for debugging and analyzing these types of applications. It allows non-intrusive observation – without affecting the runtime behavior – and also records the exact timing of program execution or other parameters. Among other things, tracing enables the investigation of timing problems or misbehavior caused by parallel execution.

Based on the recorded trace information, the UDE® Universal Debug Engine provides a variety of analysis functions and visualizations to help developers perform debugging tasks, run-time analysis, and system-level analysis.

### Trace Window

The Trace Window shows the captured trace data in tabular form and provides an precise reproduction of the program flow.

- Support for different trace sources depending on the device-specific trace system (multiple cores, busses, peripherals, etc.)
- Core-specific coloring and filtering of trace information
- Easy navigation to source code
- Highly configurable
- Fast search function for searching the entire trace recording



Tick	Address	Interpret	Source	Function	Core
0.01 us	0x80001744	ST A [a14] -0xC, a4	for(i=0; i<pData->Mo...	MODTAB_Task...	Core0
0.01 us	0x80001748	MOV d15, 0x0		MODTAB_Task...	Core0
0.01 us	0x8000174A	ST V [a14] -0x4, d15		MODTAB_Task...	Core0
0.01 us	0x8000174E	J 0x80001792		MODTAB_Task...	Core0
0.01 us	0x8000326E	RET		worker_DoOnIdleCore...	Core1
0.03 us	0x8000328C	J 0x8000328A		WORKER_Run	Core1
0.03 us	0x8000328A	CALL 0x80003255	worker_DoOnIdle()	WORKER_Run	Core1
0.03 us	0x80001792	LD V d2, [a14] -0x4	for(i=0; i<pData->Mo...	MODTAB_Task...	Core0
0.03 us	0x80001796	LD V d15, [a14] -0xC		MODTAB_Task...	Core0
0.03 us	0x8000179A	MOV A #15, d15		MODTAB_Task...	Core0
0.03 us	0x8000179C	LD V d15, [a15] 0x280		MODTAB_Task...	Core0
0.05 us	0x800017A0	JLT V d2, d15, 0x80...		MODTAB_Task...	Core0
0.05 us	0x80003255	MOV A #14, #10		worker_DoOnIdleCore...	Core1
0.05 us	0x80003258	MOVH d15, 0xD000	MODTAB_DoOnIdle(&wo...	worker_DoOnIdleCore...	Core1
0.05 us	0x8000325C	ADDI d15, d15, 0x2040		worker_DoOnIdleCore...	Core1
0.05 us	0x80003260	MOV A #15, d15		worker_DoOnIdleCore...	Core1
0.05 us	0x80003262	LD V d15, [a15] 0x0		worker_DoOnIdleCore...	Core1
0.05 us	0x80003264	ADDI d15, d15, 0xC		worker_DoOnIdleCore...	Core1
0.05 us	0x80003268	MOV A #4, d15		worker_DoOnIdleCore...	Core1
0.08 us	0x8000326A	CALL 0x800018D8		worker_DoOnIdleCore...	Core1
0.08 us	0x80001750	LD V d3, [a14] -0xC	PMoTabEntry pEntry...	MODTAB_Task...	Core0
0.08 us	0x80001754	LD V d2, [a14] -0x4		MODTAB_Task...	Core0

### Execution Sequence Chart

The Execution Sequence Chart visualizes the program flow and its timing and is the perfect tool to find bottlenecks or synchronization problems in software that is executed in parallel on a multicore system.

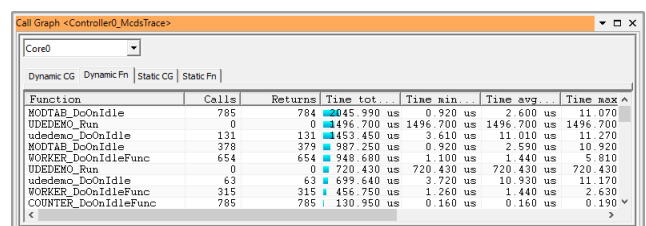
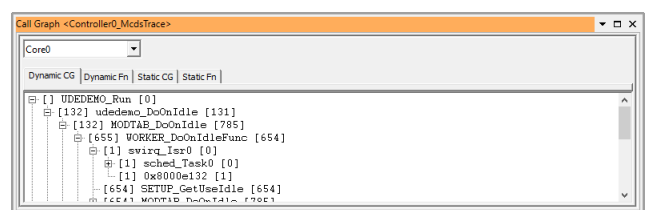
- Exact temporal display of the executed tasks and functions
- Call-depth for every point in time
- Visualization of execution sequence for multicore applications
- Easy and fast navigation and zoom
- Time measurement between user-definable markers



### Call Graph Analysis and Profiling

The Call Graph Analysis creates a representation of the control flow at software level. It represents the calling relationships between functions and sub-functions.

- Dynamic call graph. Each subroutine call is treated as a unique event unless the same subroutine is called again from the same call stack parent
- Static call graph. All calls to a subroutine are accumulated regardless of caller relationship
- Profiling information and statistics about runtime behavior
- Total, minimum, maximum and average execution time
- Number of calls and returns



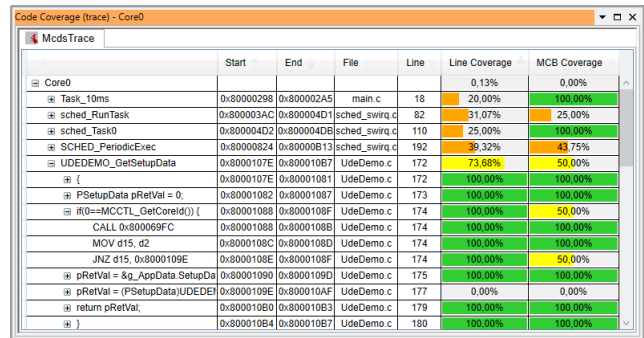
Function	Calls	Returns	Time tot...	Time min...	Time avg...	Time max...
MODTAB_DoOnIdle	785	784	14045.990 us	0.920 us	2.600 us	11.070
UDEDEMO_Run	0	0	1496.700 us	1496.700 us	1496.700 us	1496.700
UDEDEMO_DoOnIdle	131	131	4453.450 us	3.610 us	11.010 us	11.270
MODTAB_DoOnIdleFunc	378	378	987.250 us	0.920 us	2.590 us	10.920
WORKER_DoOnIdleFunc	654	654	948.680 us	1.100 us	1.440 us	5.810
UDEDEMO_Run	0	0	720.430 us	720.430 us	720.430 us	720.430
UDEDEMO_DoOnIdle	63	63	699.640 us	3.720 us	10.930 us	11.170
WORKER_DoOnIdleFunc	315	315	456.750 us	1.260 us	1.440 us	2.630
COUNTER_DoOnIdleFunc	785	785	130.950 us	0.160 us	0.160 us	0.190

# UDE® Trace Support

## Code Coverage Support

The trace-based Code Coverage Support in UDE® is a non-intrusive method that allows to determine the statement coverage (CO coverage) and branch coverage (C1 coverage) even with optimized code. No code instrumentation is required.

- Line markers in the program window indicating fully covered, partially covered, and uncovered source lines and statements
- Code Coverage Window with detailed information and bar chart for CO and C1 coverage



	Start	End	File	Line	Line Coverage	MCB Coverage
Core0					0.13%	0.00%
Task_10ms	0x80000298	0x800002A5	main.c	18	20.00%	100.00%
sched_RunTask	0x800003AC	0x800004D1	sched_swirq.c	82	31.07%	25.00%
sched_Task0	0x800004D2	0x800004D6	sched_swirq.c	110	25.00%	100.00%
SCHED_PeriodicExec	0x80000824	0x80000813	sched_swirq.c	192	39.32%	43.75%
UDEDEMO_GetSetupData	0x8000107E	0x800010B7	UdeDemo.c	172	73.68%	50.00%
{	0x8000107E	0x80001081	UdeDemo.c	172	100.00%	100.00%
PSetupData pRetVal = 0;	0x80001082	0x80001087	UdeDemo.c	173	100.00%	100.00%
if(0==MCCTL_GetCoreId()){	0x80001088	0x8000108F	UdeDemo.c	174	100.00%	50.00%
CALL 0x800069FC	0x80001088	0x80001088	UdeDemo.c	174	100.00%	100.00%
MOV d15, d2	0x8000108C	0x8000108D	UdeDemo.c	174	100.00%	100.00%
JNZ d15, 0x8000109E	0x8000108E	0x8000108F	UdeDemo.c	174	100.00%	50.00%
pRetVal = &g_AppData.SetupDa	0x80001090	0x80001090	UdeDemo.c	175	100.00%	100.00%
pRetVal = (PSetupData)UDEDE	0x8000109E	0x800010AF	UdeDemo.c	177	0.00%	0.00%
return pRetVal;	0x800010B0	0x800010B3	UdeDemo.c	179	100.00%	100.00%
}	0x800010B4	0x800010B7	UdeDemo.c	180	100.00%	100.00%

## Trace Recording

UDE® provides powerful capabilities for recording trace data from a variety of trace sources, including on-chip trace memories and various external trace interfaces for single and multicore SoCs and microcontrollers.

### UAD2next – Combining debugging and trace

- Easy mounting plug-in modules for a wide range of trace interfaces
- AURORA serial high-speed trace with up to 1.25 Gbit/s
- Parallel trace with up to 12 bit @ 125 MHz DDR
- 512 Mbyte trace memory

### UAD3+ – High end trace pushing the limits

- Separate Trace Pod connected to UAD3+ by a gigabit serial multi-lane cable (length up to 5 meters)
- AURORA serial high-speed trace with up to 12.5 Gbit/s
- Parallel trace with up to 20 bit @ 500 MHz Up to 8 Gbyte trace memory



### Supported Trace Sources

**MCDS/miniMCDS** for  
Infineon AURIX / TriCore

IEEE-ISTO 5001 **Nexus** for  
NXP MPC5xxx, ST SPC5

Arm **CoreSight** (ETM, ETB,  
TMC, ITM, PTM, FTM) for  
Arm Cortex A/R/M based  
devices

ARC **SmaRT Trace**

**RH850** Trace

### Supported Trace I/F

Serial AURORA trace

Arm HSSTP

Nexus parallel trace

Arm parallel trace

Arm SWO

On-Chip trace buffers,  
trace data to be  
transferred by standard  
debug interface

**If you have any questions about our products, please feel free to contact us:**

PLS Programmierbare Logik & Systeme GmbH  
Technologiepark Lauta  
D-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

Your local partner:

[www.pls-mc.com](http://www.pls-mc.com)  
[info@pls-mc.com](mailto:info@pls-mc.com)