

[TRACE32 Online Help](#)

[TRACE32 Directory](#)

[TRACE32 Index](#)

[TRACE32 Training](#) 

[Training ICE Emulator](#) 

[Training ICE Analyzer](#) **1**

[Introduction](#) **4**

Functional Units of the Analyzer 4

Probe Connectors 5

 HA120 5

 HAC 8

 ECC8 8

[The Trace Buffer](#) **9**

States of the Trace 10

Sampling Modes of the Trace 11

Trace Contents 13

 Record Number 13

 Run Conditions 14

 CPU Signals 15

 Time Information 17

 Set the Global Zero Point 17

 Set the Reference Point 17

 Information about the Trigger Level 19

 Marker 20

 External Trigger Input 21

Trace Configuration Window 22

 Trace Configuration Commands 23

 Size of the Trace Buffer 24

 Trace Modes 25

 Prestore Mode 25

 PrePost Mode 26

Format the Trace Buffer 27

Correlate Trace List to Source 29

Browse through the Trace Buffer 30

Find a Specific Record 31

 Example 1: Find a specific Address 31

 Example 2: Find a Variable 32

Example 3: Find the CPU Cycle where a specific Value is written to a specific Variable	32
Additional Commands to Display the Trace Buffer	33
Trace.Timing	33
Correlate Trace Timing Window and Trace List Window	34
Trigger Programming	35
The Trigger Programming Window	35
First Examples for a Trigger Program	37
Example: Sample all write accesses to the address flags	38
Trigger Conditions	38
Example: Sample all cycles where a 0 is written to the array flags	
Combined Output Commands	39
Structure of a Trigger Program	40
Declarations	40
Global Instructions	40
Local Instructions	41
Programming	42
Selective Tracing	42
Example 1: No sample command in the trigger program	42
Example 2: Only Sample.Enable commands, no Sample.ON/OFF in the trigger program	42
Example 3: Only Sample.ON/OFF instructions, no Sample.Enable in the trigger program	43
Example 4: Sample.ON/OFF instructions and Sample.Enable in the trigger program	43
Address Selectors	44
Data Selector	46
Example	47
Stop the Analyzer Recording	48
Example: Stop the trace when 0 is written to the array flags	48
Stop the Program Execution	49
Example: Stop the program when 0 is written to the array flags	50
Event Counter	51
Example 1	51
Example 2	52
Example 3	53
Time Counter	54
Example	55
Trigger Levels	56
Example	57
Markers	58
Example 1	58
Example 2	59
Trigger Inputs	60
Trigger Outputs	61
Example	61
Conflicting Instructions	62

The Analyzer Programming Dialog Window	63
How to Start	63
First Example	64
Sampling of Memory/Variable Accesses	71
Sample any Access	71
Sample all Write Accesses	72
Sample Write Accesses with a Specific Value	73
Sample a Function	74
Stop the Program Execution	76
After a Wrong Data Value is Written	76
After n Accesses to a Memory Location/Variable	78
After n Function Calls	81
After Sampling n Cycles	83
Statistic Functions	84
Function Runtime Analysis	85
Numeric Analysis	85
Linkage Analysis	92
Tree Analysis	92
Statistic Distribution of a Single Event	93
Statistic Distribution between 2 Events	94
The Performance Analyzer	96
Function Performance Analysis	96
Performance Scanning	99
Performance Analyzer Programs	101
Context Tracking System (CTS)	102
Trace Based Debugging	103
HLL Analysis of the Trace Contents	106
Background	109

Introduction

Functional Units of the Analyzer

The analyzer consists of the following parts:

- Trace buffer
- Programmable trigger unit
- Performance analyzer unit

Available as:

- HA120: High Speed State Analyzer
Analyzer is a separate module of TRACE32-ICE
- HAC:32-bit compact Analyzer
Analyzer is part of the ECC32 module
- ECC: 8-bit compact Analyzer
Analyzer is part of the ECC8 module

	HA120	HAC	ECC8
Trace buffer size	32 K * 120 bit	32 K * 96 bit	32 K *88 bit
Smallest cycle time	50 ns	50 ns	150 ns
Resolution of the time stamp unit	25 ns 48 bit	100 ns 32 bit	100 ns 40 bit
Trigger Inputs	2 channels / 2 * 8 bit 1 coax input A	1 channels / 1 * 8 bit	1 channel / 1*8 bit 1 coax input A
Trigger Outputs	1 coax output A 2 outputs		1 coax output A 2 outputs
Performance Analyzer	64 areas	-	32 areas



For the main differences for the programmable trigger unit, refer to the section *Trigger Programming*.

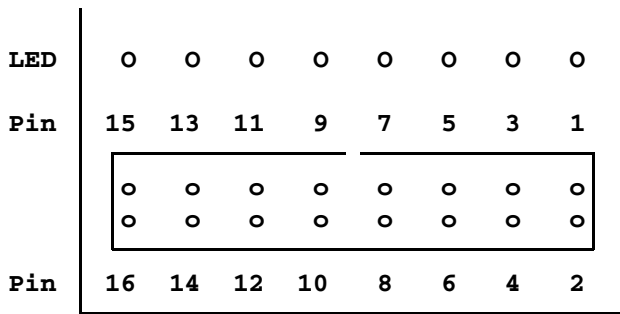
Probe Connectors

HA120



Probe Connectors		
TRIGGER A/TRIGGER B	Input	External trigger inputs
TRACEBUS		Not used
OUT	Output	Trigger outputs C and D

TRIGGER A/B



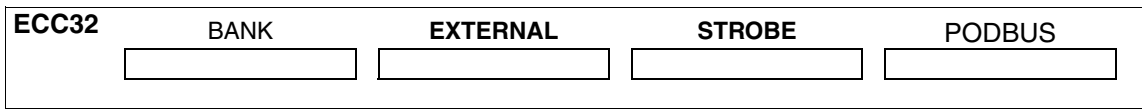
Pin 1	Line 0	Data 0
Pin 3	Line 1	Data 1
Pin 5	Line 2	Data 2
Pin 7	Line 3	Data 3
Pin 9	Line 4	Data 4
Pin 11	Line 5	Data 5
Pin 13	Line 6	Data 6
Pin 15	Line 7	Data 7
Pin 2, 4, 6, 8, 10, 12, 14, 16	Ground	

LED	0	0	0	0	0	0	0	0
Pin	15	13	11	9	7	5	3	1
	o	o	o	o	o	o	o	o
	o	o	o	o	o	o	o	o
Pin	16	14	12	10	8	6	4	2

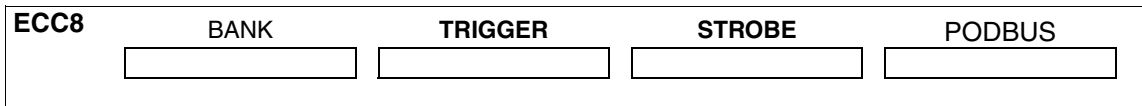
HA120:

Pin 1 Trigger output C high-active
 Pin 3 Trigger output D high-active
 Pin 5 Not assigned
 Pin 7 Not assigned
 Pin 9 Not assigned
 Pin 11 Not assigned
 Pin 13 Cycle signal (synchr. signal to CPU)
 Pin 15 Not assigned

Pin 2,4,6,8,10,12,14,16 GND



Probe Connectors		
EXTERNAL	Input	External trigger inputs
STROBE	Output	Not used

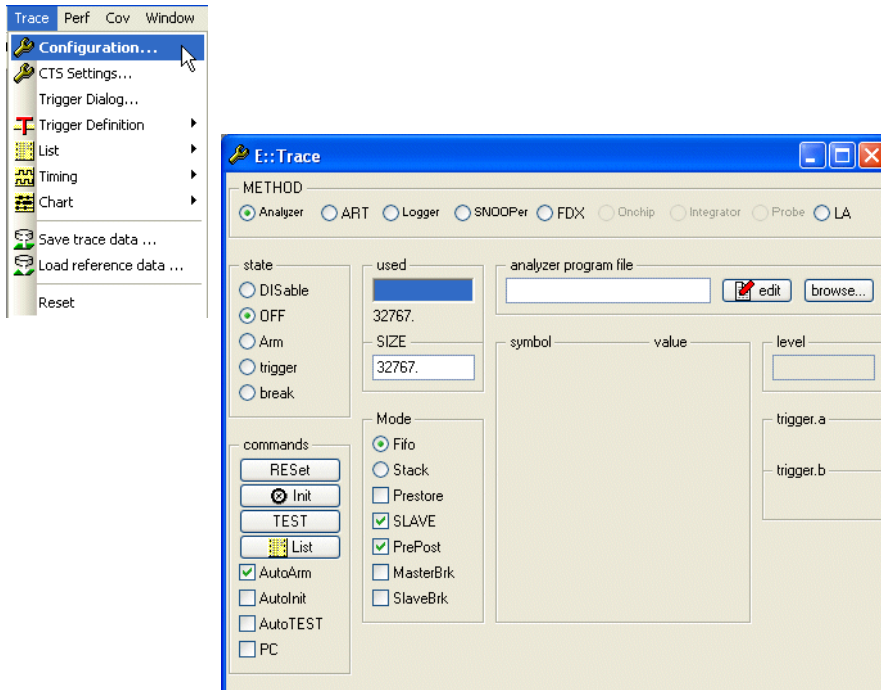


Probe Connectors		
TRIGGER	Input	External trigger inputs
STROBE	Output	Trigger outputs C and D

The Trace Buffer

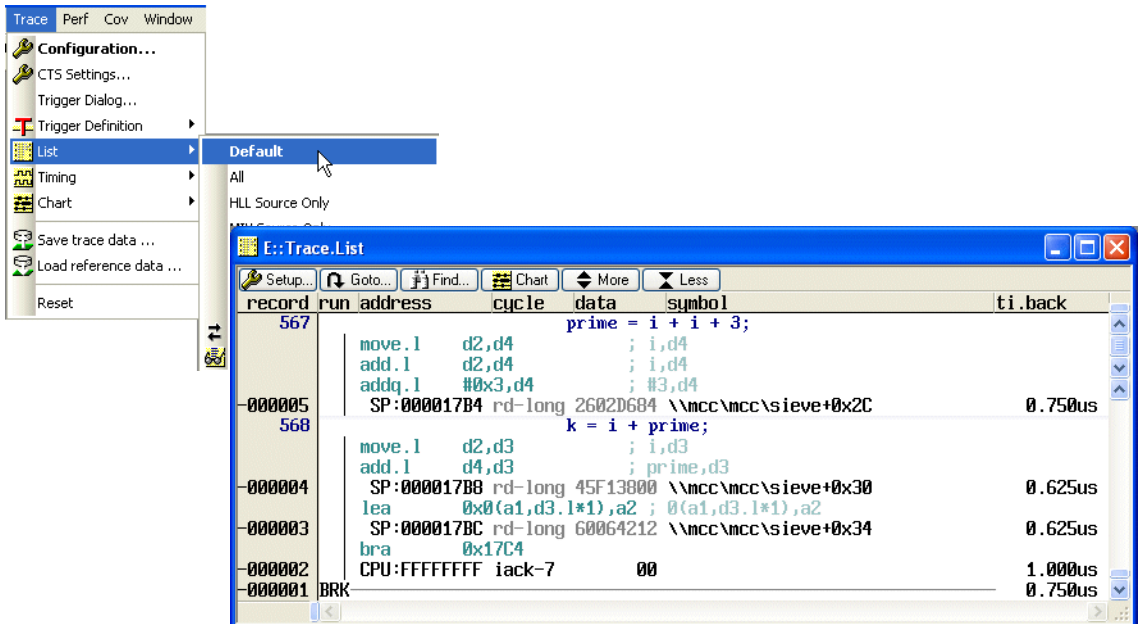
The recording to the trace buffer is controlled

- By the settings of the TTrace Configuration Window



- By the Analyzer Trigger Unit (Trigger/Filter)

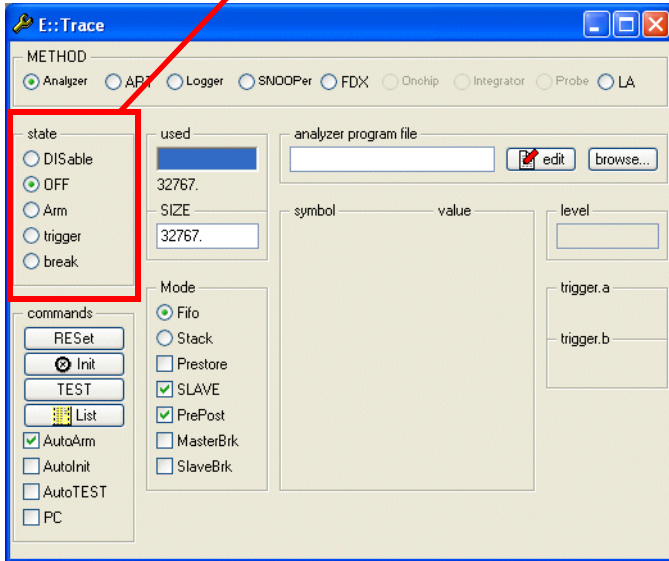
If no trigger program is active, the trace will record all CPU cycles.



States of the Trace

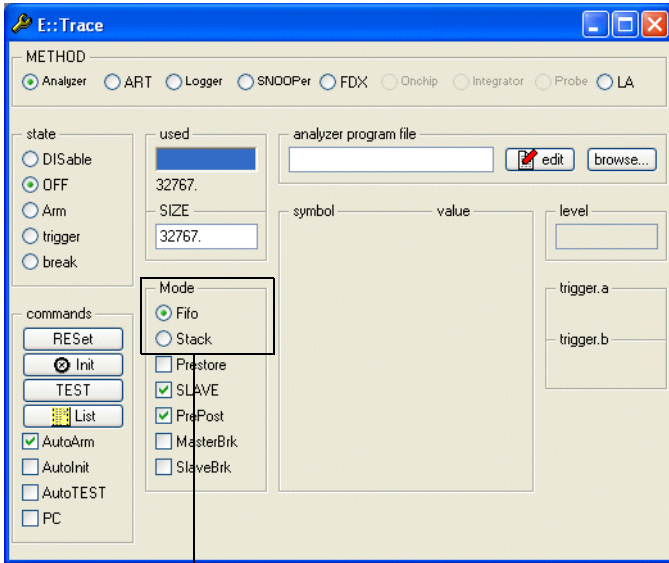
The trace buffer can either sample or display the information.

State of the Trace



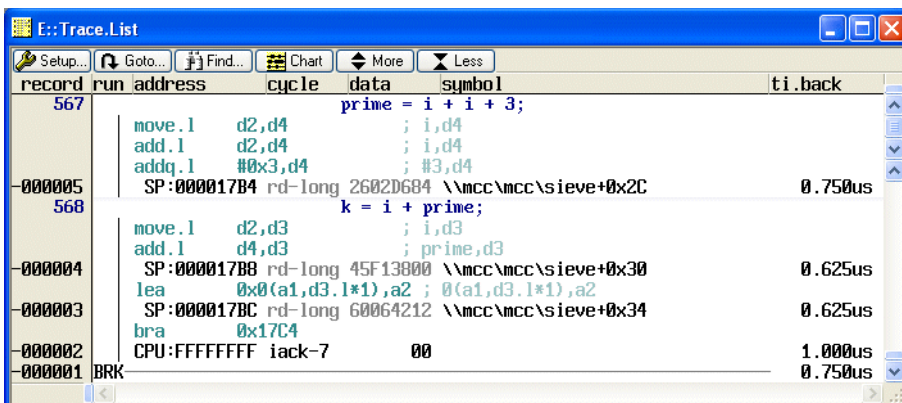
State of the Trace	
DISable	The Trace is not working
OFF	The Trace is not active. Trace buffer can be displayed.
Arm	The Trace is active (information is sampled into the trace buffer). Trace buffer can not be displayed.
trigger	Not used for TRACE32-ICE
break	Trace is in break state (BREAK command from a trigger program), trace buffer can be displayed.

Sampling Modes of the Trace



Sampling Modes

Sampling Modes	
Fifo	The trace is working in FIFO mode. When the trace buffer is full, the new records will overwrite the older ones. The trace records always the last cycle before the program execution is stopped.
Stack	The trace is working in STACK mode. If the trace buffer is full, the trace will stop sampling. The trace records always the first cycle after the program start.



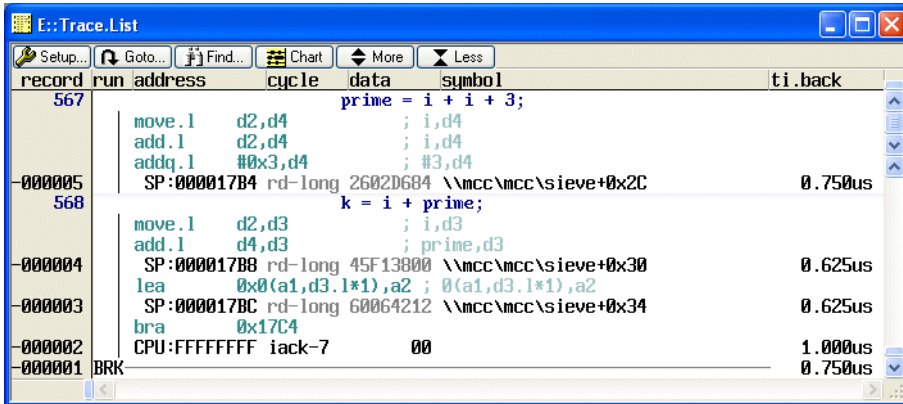
The trace is working in FIFO mode, last record sampled before the stop is record (-1)

Trace is working in STACK mode, the first cycle has record number +1

record	run	address	cycle	data	symbol	ti.back

+000001	G0	SP:00001788	rd-long	45F13800	\\mcc\mcc\sieve+0x30	
		move.w	d0,d4		; d0,prime	
+000002		SP:000017BC	rd-long	60064212	\\mcc\mcc\sieve+0x34	0.500
		bra	0x17C4			
+000003		SP:000017C0	rd-long	D5C4D684	\\mcc\mcc\sieve+0x38	0.750
+000004		SP:000017C4	rd-long	7012B083	\\mcc\mcc\sieve+0x3C	0.500
569					while (k <= SIZE)	
		moveq	#0x12,d0		; #18,d0	

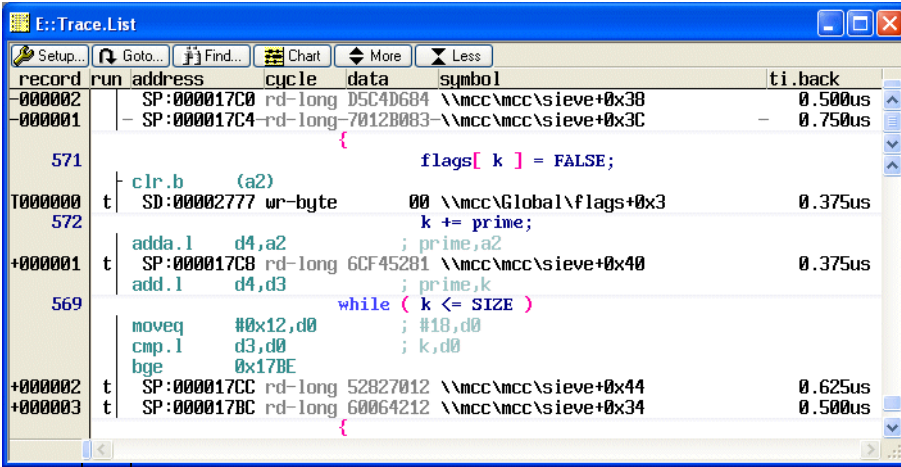
Record Number



record	run	address	cycle	data	symbol	ti.back
567				prime = i + i + 3;		
		move.l	d2,d4		; i,d4	
		add.l	d2,d4		; i,d4	
		addq.l	#0x3,d4		#3,d4	
-000005		SP:000017B4	rd-long	2602D684	\\mcc\mcc\sieve+0x2C	0.750us
568				k = i + prime;		
		move.l	d2,d3		; i,d3	
		add.l	d4,d3		; prime,d3	
-000004		SP:000017B8	rd-long	45F13800	\\mcc\mcc\sieve+0x30	0.625us
		lea	0x0(a1,d3.l*1),a2		0(a1,d3.l*1),a2	
-000003		SP:000017BC	rd-long	60064212	\\mcc\mcc\sieve+0x34	0.625us
		bra	0x17C4			
-000002		CPU:FFFFFFFF	iack-7	00		1.000us
-000001		BRK				0.750us

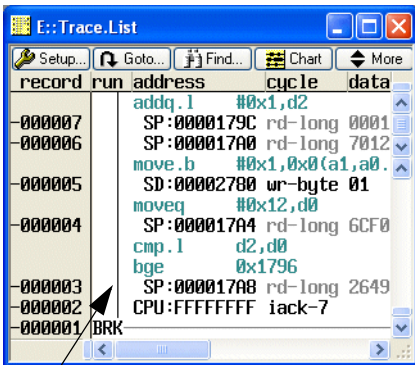
Record Number

Run Conditions

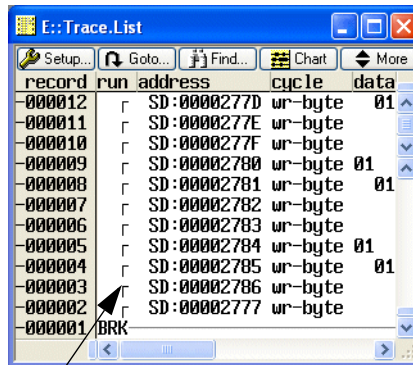


Run condition

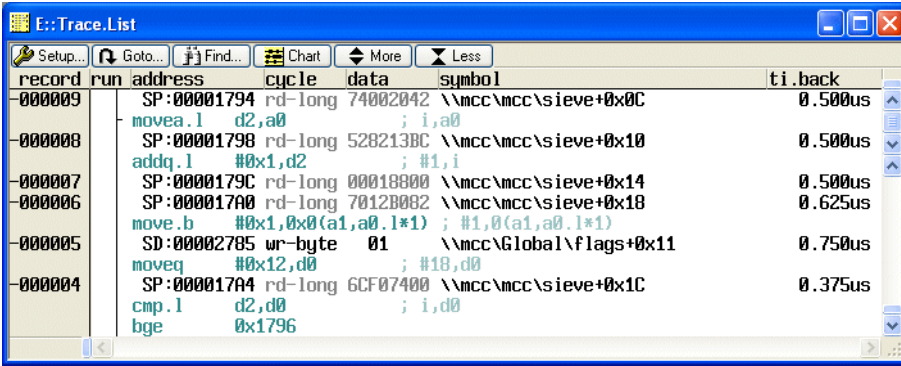
T	Record where an Emulator Trigger Event happened (Trigger Point). This record becomes record 0.
t	Record was sampled after an Emulator Trigger Event happened



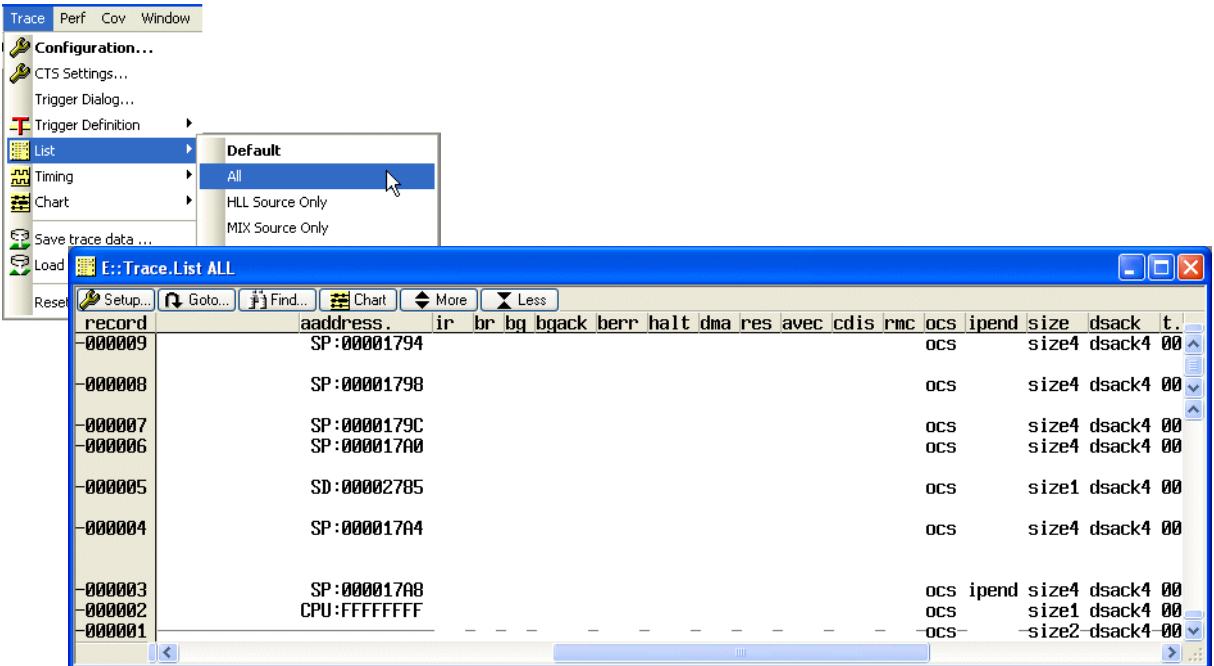
A straight line indicates that all CPU cycles were sampled continuously.



A broken line indicates that between two sampled records there were not sampled CPU cycles.



Address	CPU address bus
Cycle	CPU bus cycle type
Data.B/Data.W/Data.L	CPU data bus
sYmbol	Symbolic address with path and offset



AAddress	Absolute (physical) address
-----------------	-----------------------------

For a list and a description of the CPU specific signals that are sampled in the trace buffer refer to **ICE Target Guide**, section **State Analyzer** chapter **Keyword for Display** (Menu: **Help**→**ICE Target Guide**).

Additional CPU lines can be sampled using the Port Analyzer (64 lines) or the external trigger inputs (16 lines).

Time Information

record	symbol	ti.back	ti.fore	ti.zero
-000010	4A1B \mcc\mcc\sieve+0x20	0.625us	0.500us	-1.500us
-000009	2042 \mcc\mcc\sieve+0x0C	0.500us	0.500us	-1.000us
-000008	13BC \mcc\mcc\sieve+0x10	0.500us	0.500us	-0.500us
-000007	8800 \mcc\mcc\sieve+0x14	0.500us	0.625us	0.000
-000006	B082 \mcc\mcc\sieve+0x18	0.625us	0.750us	0.625us
-000005	1*1) ; #1,0(a1,a0.l*1) \mcc\Global\flags+0x11	0.750us	0.375us	1.375us
-000004	7400 \mcc\mcc\sieve+0x1C	0.375us	0.625us	1.750us

Time.Back	Time relative to the previous record.
Time.Fore	Time relative to the next record.
Time.Zero	Time relative to the global zero point.

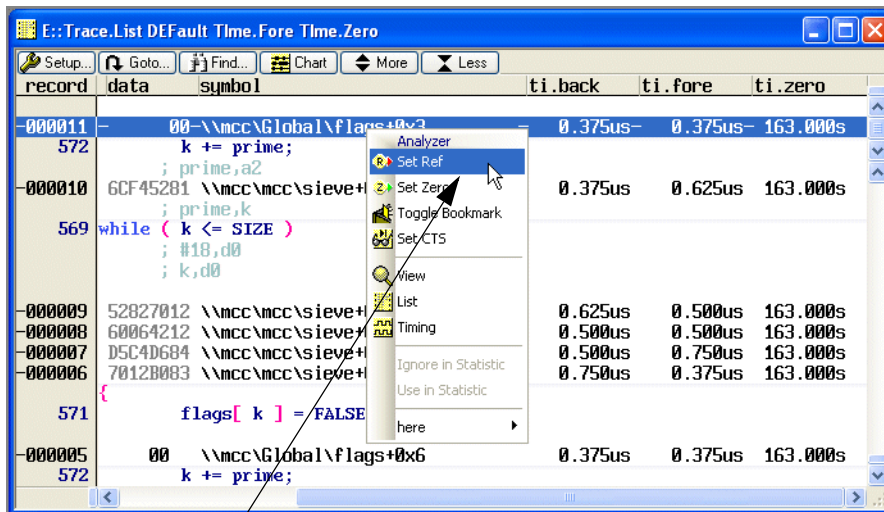
Set the Global Zero Point

record	data	symbol	ti.back	ti.fore	ti.zero
-000014	528213BC	\mcc\mcc\sieve+0x10	0.500us	0.500us	0.000
-000013	00018800	\mcc\mcc\sieve+0x14	0.625us	0.500us	0.000
-000012	7012B082	\mcc\mcc\sieve+0x18	0.750us	0.750us	1.125us
-000011	01	\mcc\Global\flags+0x0D	0.375us	0.375us	1.875us
-000010	6CF07400	\mcc\mcc\sieve+0x1C	0.625us	0.625us	2.250us
-000009	26494A1B	\mcc\mcc\sieve+0x20	0.500us	0.500us	2.875us
-000008	74002042	\mcc\mcc\sieve+0x0C	0.500us	0.500us	3.375us
-000007	528213BC	\mcc\mcc\sieve+0x10	0.500us	0.500us	3.875us
-000006	00018800	\mcc\mcc\sieve+0x14	0.625us	0.625us	4.375us
-000005	7012B082	\mcc\mcc\sieve+0x18	0.750us	0.750us	5.000us

Establish the selected record as global zero point

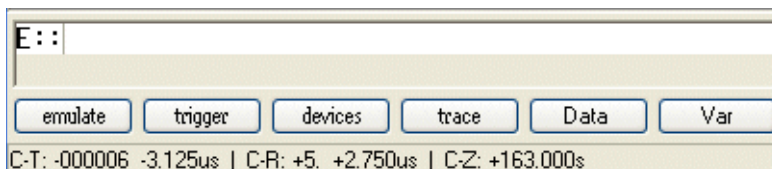
Set the Reference Point

By default, the reference point is always the last record in the trace buffer.



Establish the selected record as reference point

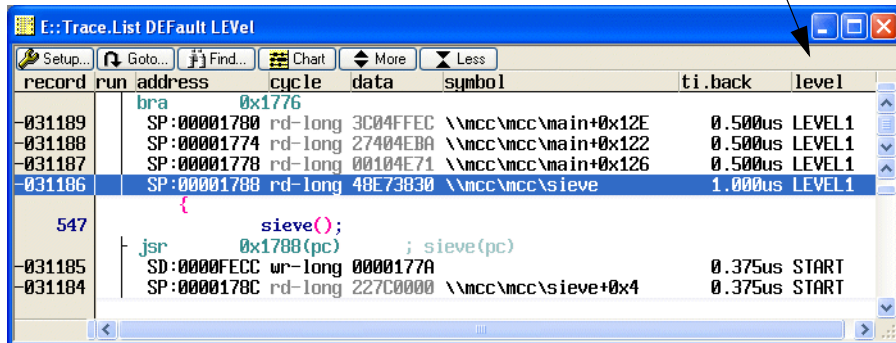
If you select a record in the trace buffer, you get the following time information in the state line:



C-T Cursor-Trigger	Number of records that were sample from the record at the cursor position to the trigger point.	Time distance from the record at the cursor position to the trigger point.
C-R Cursor-Reference	Number of records that were sample from the record at the cursor position to the reference point.	Time distance from the record at the cursor position to the reference point.
C-Z Cursor-Zero		Time distance from the record at the cursor position to the global zero point.

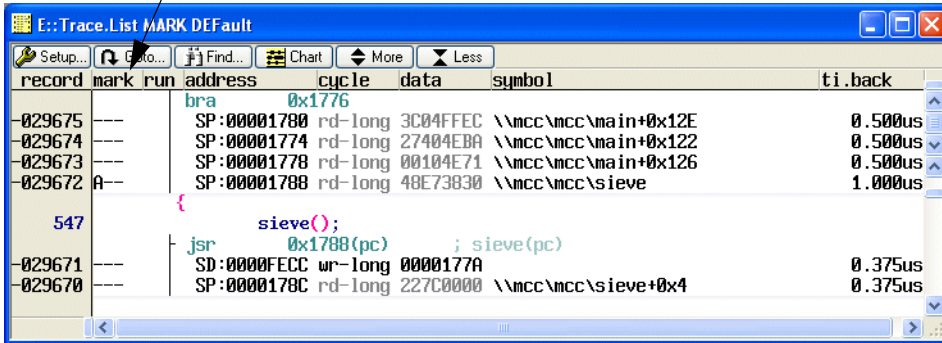
Information about the Trigger Level

If the recording to the trace buffer is controlled by a trigger program with several trigger levels, the trigger level, that was active, when the record was sampled is displayed here.



record	run	address	cycle	data	symbol	ti.back	level
		bra	0x1776				
-031189		SP:00001780	rd-long	3C04FFEC	\\mcc\mcc\main+0x12E	0.500us	LEVEL1
-031188		SP:00001774	rd-long	27404EBA	\\mcc\mcc\main+0x122	0.500us	LEVEL1
-031187		SP:00001778	rd-long	00104E71	\\mcc\mcc\main+0x126	0.500us	LEVEL1
-031186		SP:00001788	rd-long	48E73830	\\mcc\mcc\sieve	1.000us	LEVEL1
	547	{					
		sieve();					
		jsr	0x1788(pc)		; sieve(pc)		
-031185		SD:0000FECC	wr-long	0000177A		0.375us	START
-031184		SP:0000178C	rd-long	227C0000	\\mcc\mcc\sieve+0x4	0.375us	START

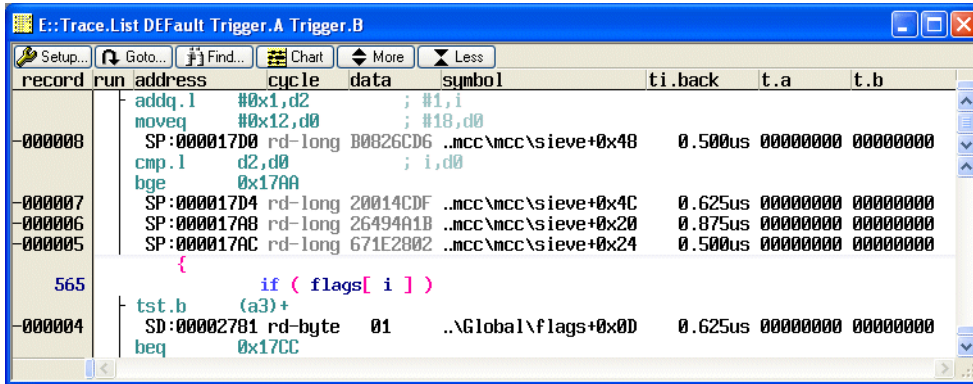
Specific record in the trace buffer can be marked, to make it easier to find and display specific events



Available markers:

HA120	A, B, C
ECC8	A, B
HAC	A, B, C

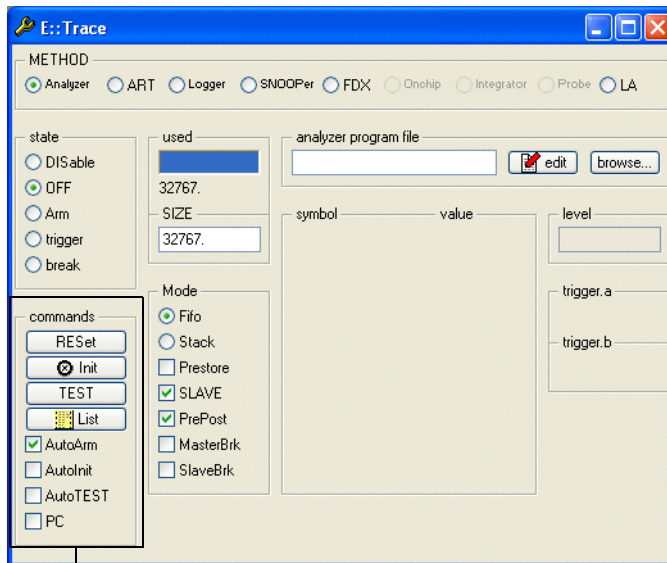
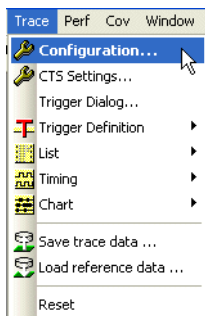
External Trigger Input



Available External Trigger Inputs:

HA120	TRIGGER A, TRIGGER B	16 trigger lines
ECC8	TRIGGER	8 trigger lines
HAC	EXTERNAL	8 trigger lines

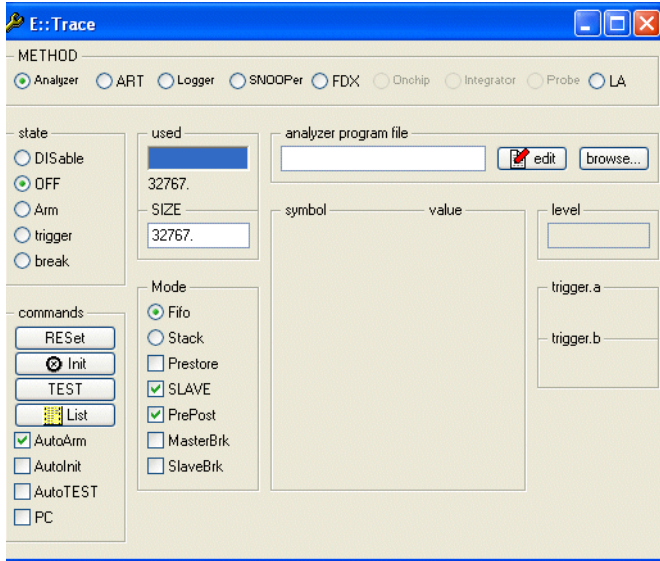
Trace Configuration Window

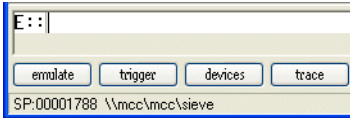


Trace Configuration Commands

Trace.state

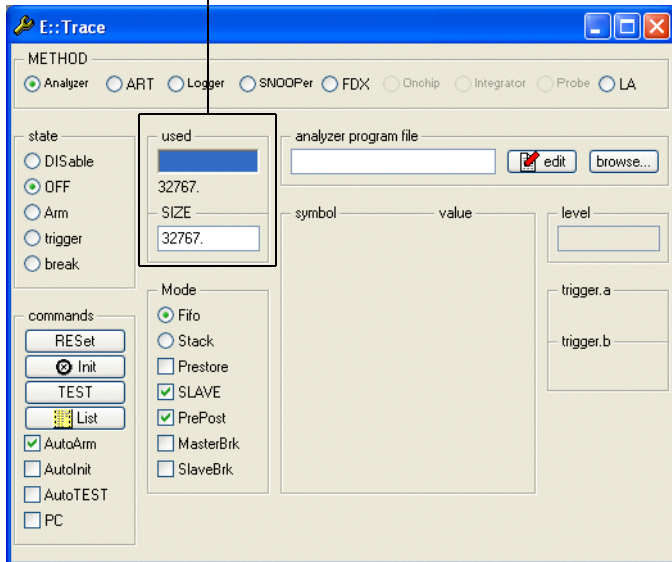
Display Trace configuration window



RESet	Reset the Trace Configuration Window to its default settings.
Init	Clears the trace buffer and initializes the programmable trigger unit.
TEST	Rarely used.
AutoArm	Trace is automatically armed, when user program is started. Trace is automatically switched off, when user program stops.
AutoTEST	Rarely used.
AutoInit	By default, the trace buffer is not cleared when the user program is restarted by Go or Step. Autoinit automatically clears the trace buffer and initializes the programmable trigger unit at every program start.
PC	ON: Displays the PC in the state line, while the CPU is running in real time. 

Size of the Trace Buffer

Size field



used	Number of sampled records in the trace buffer If the number of used records is gray, this indicates a trace buffer overflow.
SIZE	Size can be decreased for faster search or save operations.

Prestore Mode

record	run	address	cycle	data	symbol	ti.back
-000011		SD:00002780	wr-byte	00	\\mcc\Global\flags+0x0C	3.125us
-000010		SD:00002783	wr-byte	00	\\mcc\Global\flags+0x0F	3.125us
-000009		SD:00002786	wr-byte	00	\\mcc\Global\flags+0x12	3.125us
-000008		SD:0000277A	wr-byte	00	\\mcc\Global\flags+0x6	11.250us
-000007		SD:0000277F	wr-byte	00	\\mcc\Global\flags+0x0B	3.125us
-000006		SD:00002784	wr-byte	00	\\mcc\Global\flags+0x10	3.125us
-000005		SD:0000277D	wr-byte	00	\\mcc\Global\flags+0x9	11.250us
-000004		SD:00002784	wr-byte	00	\\mcc\Global\flags+0x10	3.125us
-000003		SD:00002783	wr-byte	00	\\mcc\Global\flags+0x0F	15.250us

At a selective trace on data accesses it might be interesting to see, which function made the data access.

E::Trace

METHOD: Analyzer ART Logger SNOOPer FDX Onchip Integrator Probe LA

state: Disable OFF Arm trigger break

commands: RESet, Init, TEST, List, AutoArm, AutoInit, AutoTEST, PC

used: 0, SIZE: 16384

analyzer program file: ([]) ts

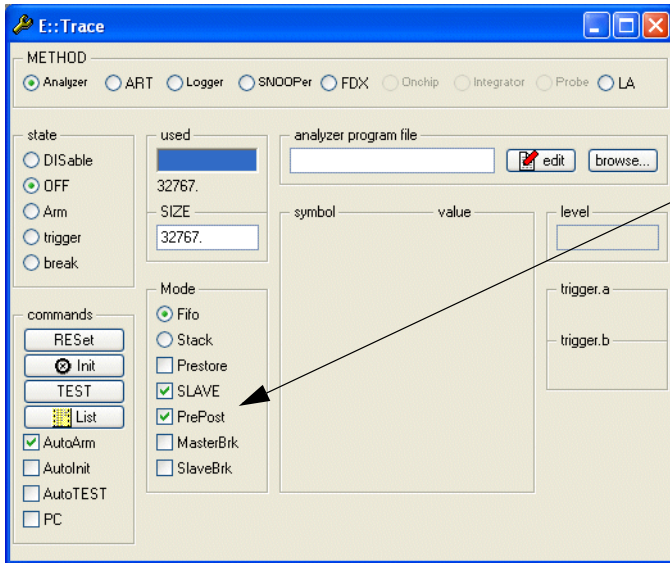
Mode: Fifo Stack Prestore SLAVE PrePost MasterBrk SlaveBrk

Switch the Prestore mode ON, to sample also the last opfetch before the selective data access.

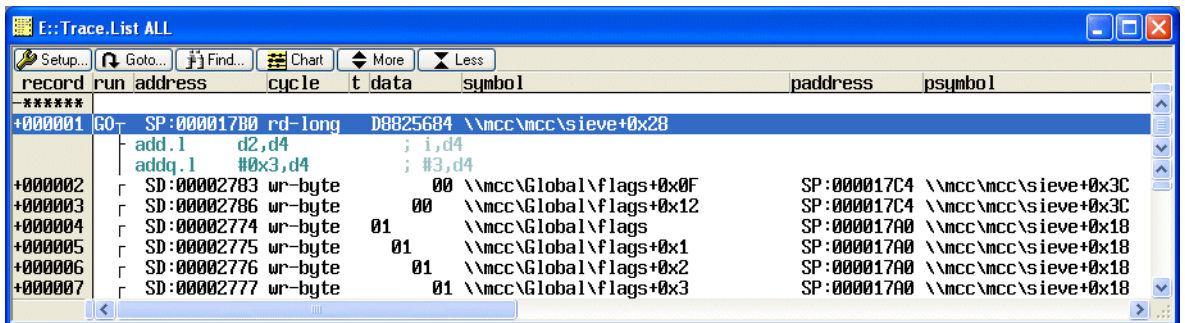
The Prestore mode halves the size of the trace buffer.

record	run	address	cycle	t data	symbol	paddress	psymbol
-000011		SD:0000277A	wr-byte	00	\\mcc\Global\flags+0x6	SP:000017C4	\\mcc\mcc\sieve+0x3C
-000010		SD:0000277D	wr-byte	00	\\mcc\Global\flags+0x9	SP:000017C4	\\mcc\mcc\sieve+0x3C
-000009		SD:00002780	wr-byte	00	\\mcc\Global\flags+0x0C	SP:000017C4	\\mcc\mcc\sieve+0x3C
-000008		SD:00002783	wr-byte	00	\\mcc\Global\flags+0x0F	SP:000017C4	\\mcc\mcc\sieve+0x3C
-000007		SD:00002786	wr-byte	00	\\mcc\Global\flags+0x12	SP:000017C4	\\mcc\mcc\sieve+0x3C
-000006		SD:0000277A	wr-byte	00	\\mcc\Global\flags+0x6	SP:000017C4	\\mcc\mcc\sieve+0x3C
-000005		SD:0000277F	wr-byte	00	\\mcc\Global\flags+0x0B	SP:000017C4	\\mcc\mcc\sieve+0x3C
-000004		SD:00002784	wr-byte	00	\\mcc\Global\flags+0x10	SP:000017C4	\\mcc\mcc\sieve+0x3C
-000003		SD:0000277D	wr-byte	00	\\mcc\Global\flags+0x9	SP:000017C4	\\mcc\mcc\sieve+0x3C
-000002		SD:00002784	wr-byte	00	\\mcc\Global\flags+0x10	SP:000017C4	\\mcc\mcc\sieve+0x3C

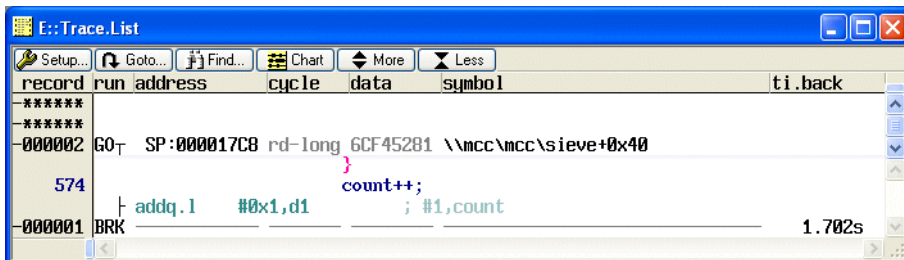
Opfetch before the data access



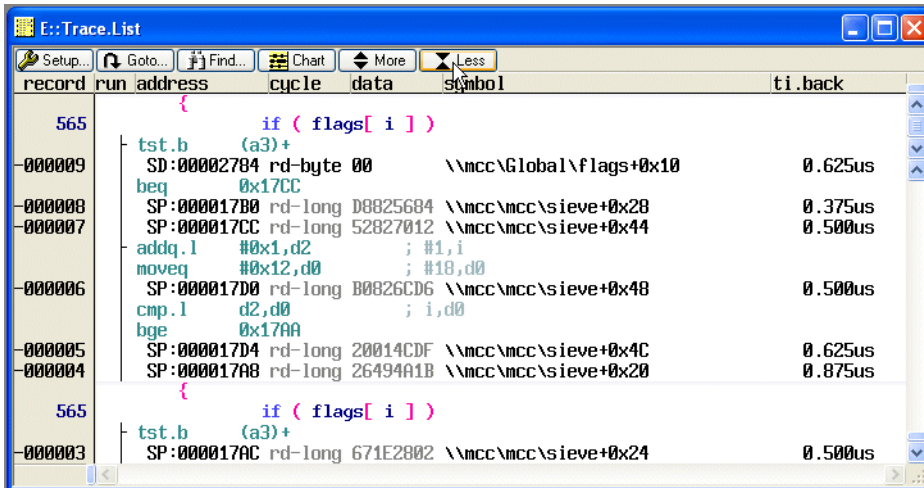
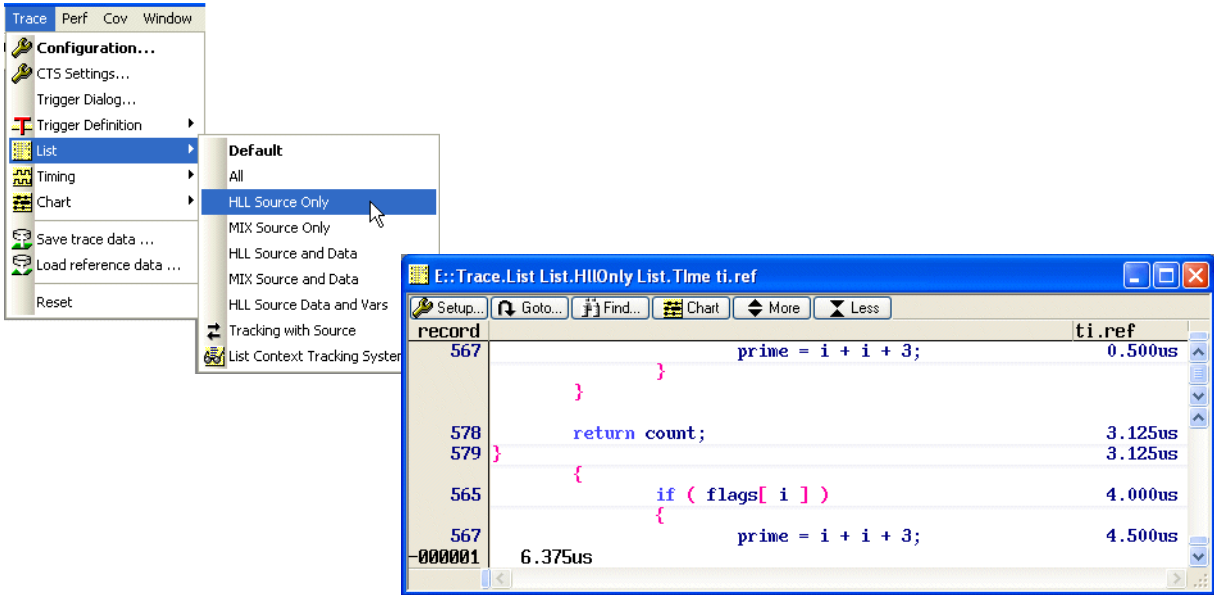
The PrePost Mode forces a recording of the opfetch cycle at the beginning of the program execution (**GO**) and at the end of the program execution (**BRK**).



The PrePost option is useful, if you write a trigger program for a selective trace, but nothing is sampled in the trace buffer. The GO and BRK record in the trace buffer show you at least, the user program was executed from GO to BRK and how much time was spent.



Format the Trace Buffer



1. time Less	Suppress the display of dummy cycles
2. time Less	Suppress the display of opfetch cycles
3. time Less	Display only HLL sources and data fetches
4. time Less	Display only HLL sources

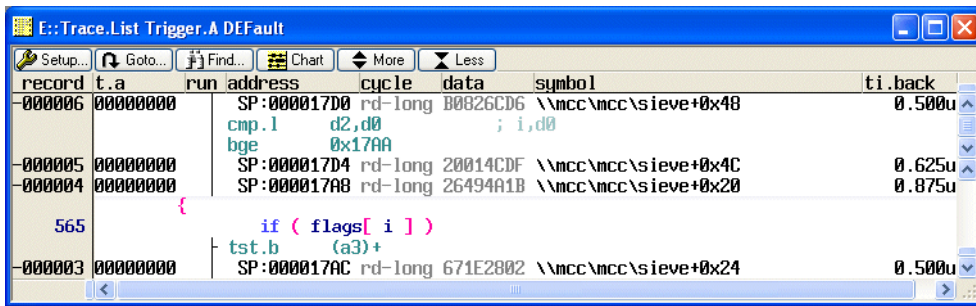
The **More** button works viserversa.

Trace.List

Display Trace contents

Example 1: External trigger input TRIGGER A plus default items.

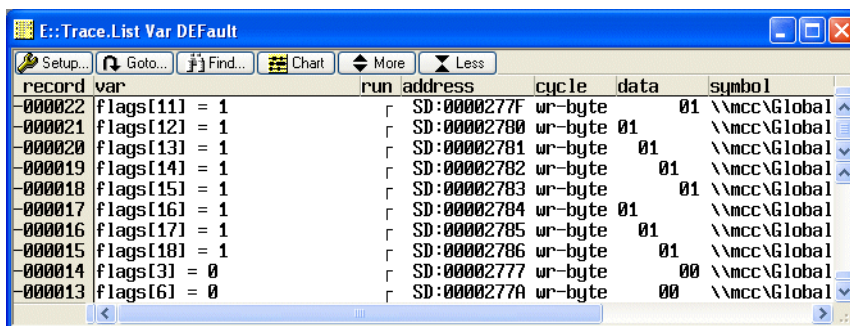
Trace.List Trigger.A Default



record	t.a	run	address	cycle	data	symbol	ti.back
-000006	00000000		SP:000017D0	rd-long	B0826CD6	\\mcc\mcc\sieve+0x48	0.500u
			cmp.l		d2,d0	; i,d0	
			bge		0x17AA		
-000005	00000000		SP:000017D4	rd-long	20014CDF	\\mcc\mcc\sieve+0x4C	0.625u
-000004	00000000		SP:000017A8	rd-long	26494A1B	\\mcc\mcc\sieve+0x20	0.875u
	565		{				
			if (flags[i])				
			tst.b		(a3)+		
-000003	00000000		SP:000017AC	rd-long	671E2802	\\mcc\mcc\sieve+0x24	0.500u

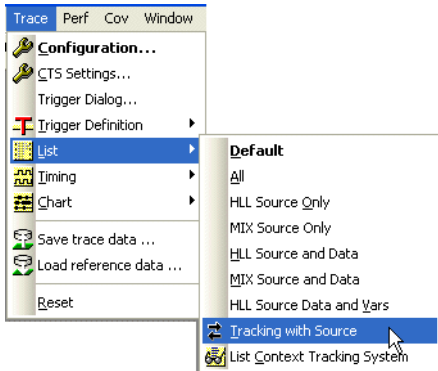
Example 2: Symbolic display of data accesses as HLL variables plus default items.

Trace.List Var Default

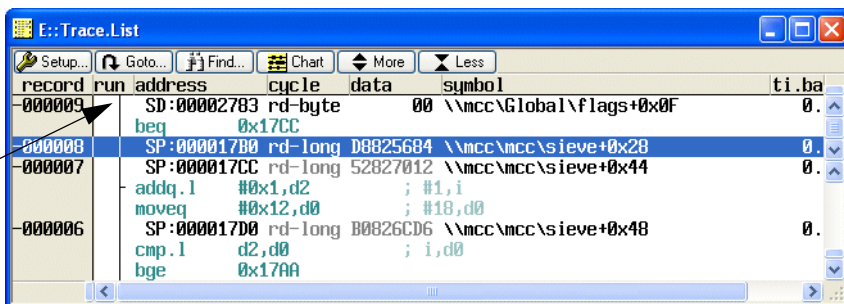


record	var	run	address	cycle	data	symbol
-000022	flags[11] = 1	┌	SD:0000277F	wr-byte	01	\\mcc\Global
-000021	flags[12] = 1	┌	SD:00002780	wr-byte	01	\\mcc\Global
-000020	flags[13] = 1	┌	SD:00002781	wr-byte	01	\\mcc\Global
-000019	flags[14] = 1	┌	SD:00002782	wr-byte	01	\\mcc\Global
-000018	flags[15] = 1	┌	SD:00002783	wr-byte	01	\\mcc\Global
-000017	flags[16] = 1	┌	SD:00002784	wr-byte	01	\\mcc\Global
-000016	flags[17] = 1	┌	SD:00002785	wr-byte	01	\\mcc\Global
-000015	flags[18] = 1	┌	SD:00002786	wr-byte	01	\\mcc\Global
-000014	flags[13] = 0	┌	SD:00002777	wr-byte	00	\\mcc\Global
-000013	flags[16] = 0	┌	SD:0000277A	wr-byte	00	\\mcc\Global

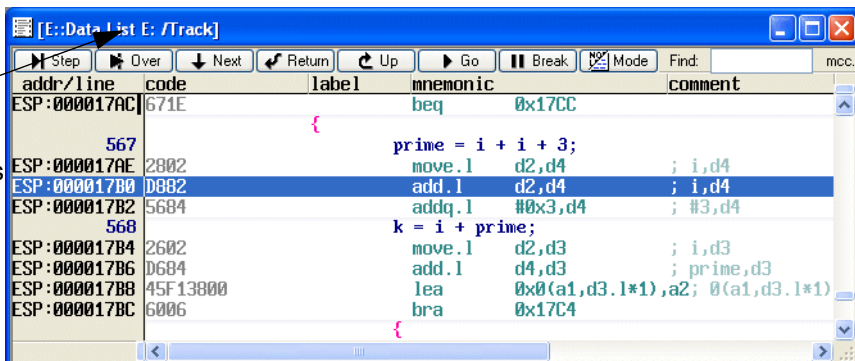
Correlate Trace List to Source



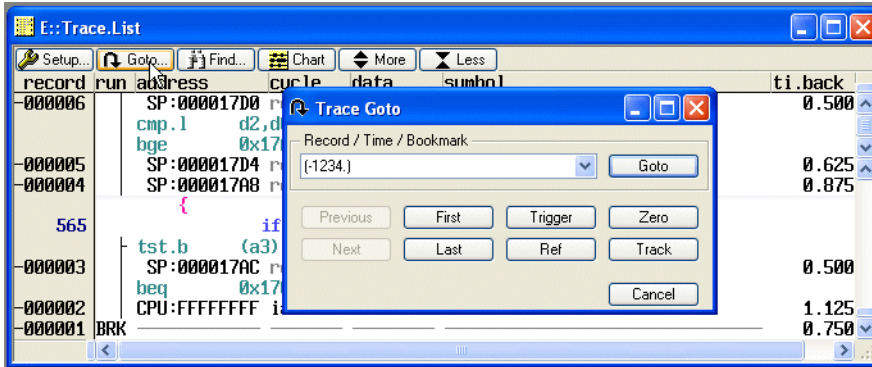
Active Window



All windows opened with the **Track Option** follow the cursor movements in the active window



Browse through the Trace Buffer



Pg ↑	Scroll page up
Pg ↓	Scroll page down
Ctrl - Pg ↑	Go to the first record
Ctrl - Pg ↓	Go to the last record

Find a Specific Record

The screenshot shows the 'E::Trace.List' window with a list of records. A 'Trace Find' dialog is open in the foreground. The dialog has several radio buttons: 'Expert' (unselected), 'Cycle' (selected), 'Group' (unselected), 'Changes' (unselected), 'Up' (unselected), 'Signal' (unselected), and 'Down' (selected). The 'address / expression' field is empty. Below it are fields for 'Cycle' and 'Data', both with dropdown menus. At the bottom are buttons for 'Find Next', 'Find First', 'Find Here', 'Find All', 'Clear', and 'Cancel'.

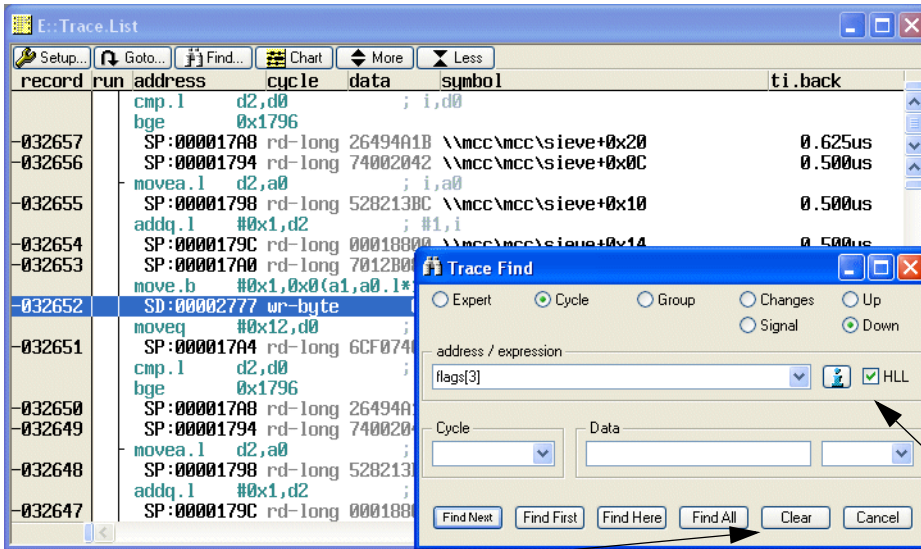
record	run	address	cycle	data	symbol	ti.back
-000011		SP:000017A8	rd-long	26494A1B	\\mcc\mcc\sieve+0x20	0.875us
-000010		SP:000017AC	rd-long	671E2802	\\mcc\mcc\sieve+0x24	0.500us
565					if (flags[i])	
-000009		tst.b (a3)+				
		SD:00002783	rd-byte	00	\\mcc\Global\flags+0x0F	0.625us
		beg		0x17CC		
-000008		SP:000017B0	rd-long	D8825604	\\mcc\mcc\sieve+0x28	0.275us
-000007		SP:000017CC	rd-long	52827000	\\mcc\mcc\sieve+0x2C	0.275us
		addq.l #0x1,d2				
		moveq #0x12,d0				
-000006		SP:000017D0	rd-long	B0826C00	\\mcc\mcc\sieve+0x30	0.275us
		cmp.l d2,d0				
		bge		0x17AA		
-000005		SP:000017D4	rd-long	20014C00	\\mcc\mcc\sieve+0x34	0.275us
-000004		SP:000017A8	rd-long	26494A1B	\\mcc\mcc\sieve+0x20	0.875us
565					if (flags[i])	
-000003		tst.b (a3)+				
		SD:00002781	rd-byte	01	\\mcc\Global\flags+0x0F	0.625us
		beg		0x17CC		
		SP:000017B0	rd-long	D8825604	\\mcc\mcc\sieve+0x28	0.275us
					prime =	
		move.l d2,d4				
		add.l d2,d4				
		addq.l #0x3,d4				
-032755		SP:000017B4	rd-long	2602D600	\\mcc\mcc\sieve+0x38	0.275us
568					k = i +	
		move.l d2,d3				
		add.l d4,d3				
-032754		SP:000017B8	rd-long	45F13800	\\mcc\mcc\sieve+0x3C	0.275us

Example 1: Find a specific Address

The screenshot shows the 'E::Trace.List' window with a list of records. A 'Trace Find' dialog is open in the foreground. The 'Cycle' radio button is selected. The 'address / expression' field contains '17a8'. The 'Signal' radio button is selected. The 'Find All' button is highlighted.

record	run	address	cycle	data	symbol	ti.back
		cmp.l d2,d0			; i,d0	
		bge		0x17AA		
-032760		SP:000017D4	rd-long	20014C00	\\mcc\mcc\sieve+0x40	0.625us
-032759		SP:000017A8	rd-long	26494A1B	\\mcc\mcc\sieve+0x20	0.875us
-032758		SP:000017AC	rd-long	671E2802	\\mcc\mcc\sieve+0x24	0.500us
565					if (flags[i])	
-032757		tst.b (a3)+				
		SD:00002781	rd-byte	01	\\mcc\Global\flags+0x0F	0.625us
		beg		0x17CC		
-032756		SP:000017B0	rd-long	D8825604	\\mcc\mcc\sieve+0x28	0.275us
567					prime =	
		move.l d2,d4				
		add.l d2,d4				
		addq.l #0x3,d4				
-032755		SP:000017B4	rd-long	2602D600	\\mcc\mcc\sieve+0x38	0.275us
568					k = i +	
		move.l d2,d3				
		add.l d4,d3				
-032754		SP:000017B8	rd-long	45F13800	\\mcc\mcc\sieve+0x3C	0.275us

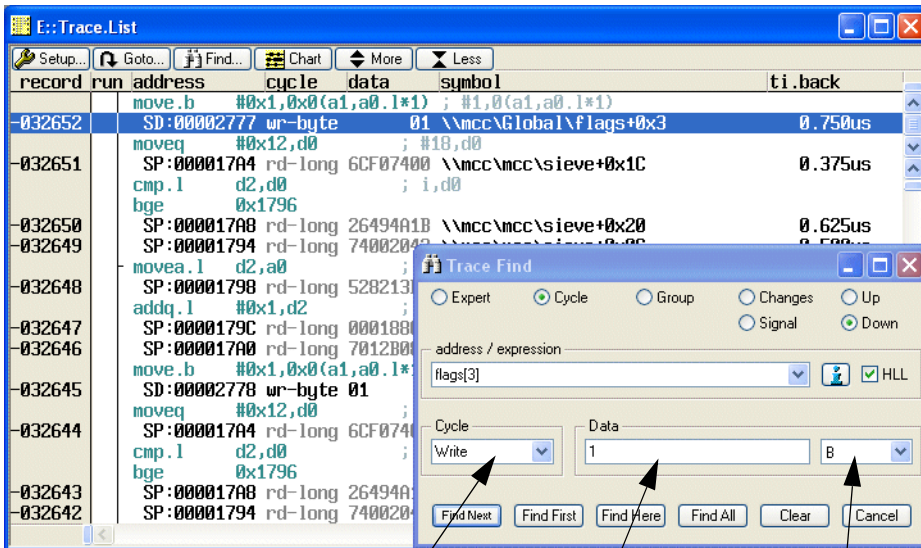
Example 2: Find a Variable



Push **Clear** to reset the settings in the Trace Find window

If the check box **HLL** is ON, you can look for any hll symbol

Example 3: Find the CPU Cycle where a specific Value is written to a specific Variable



Define the CPU cycle

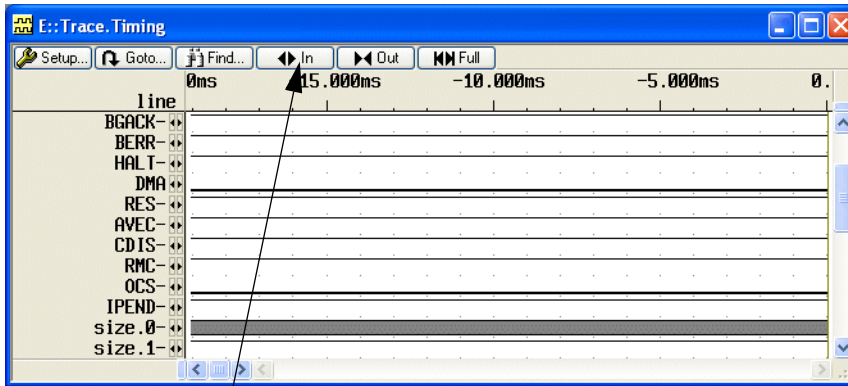
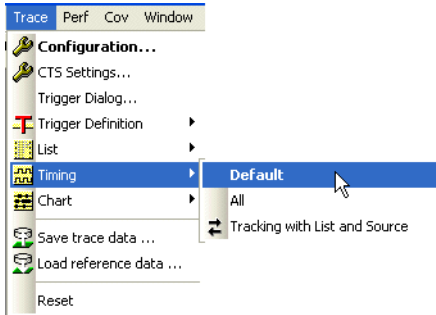
Define the data

Define the data width (position on the data bus)

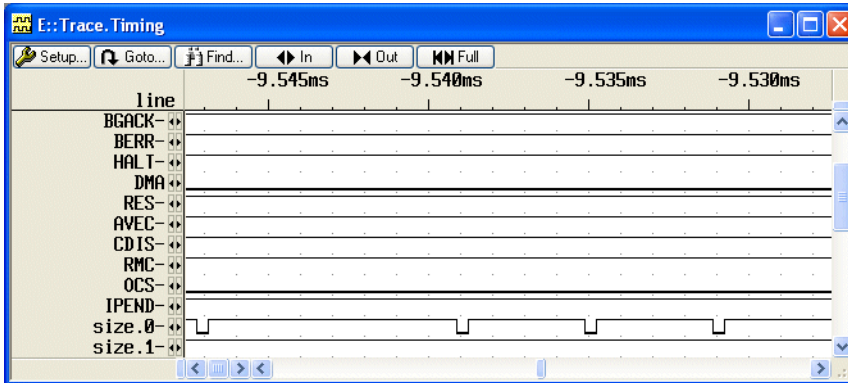
Trace.Find

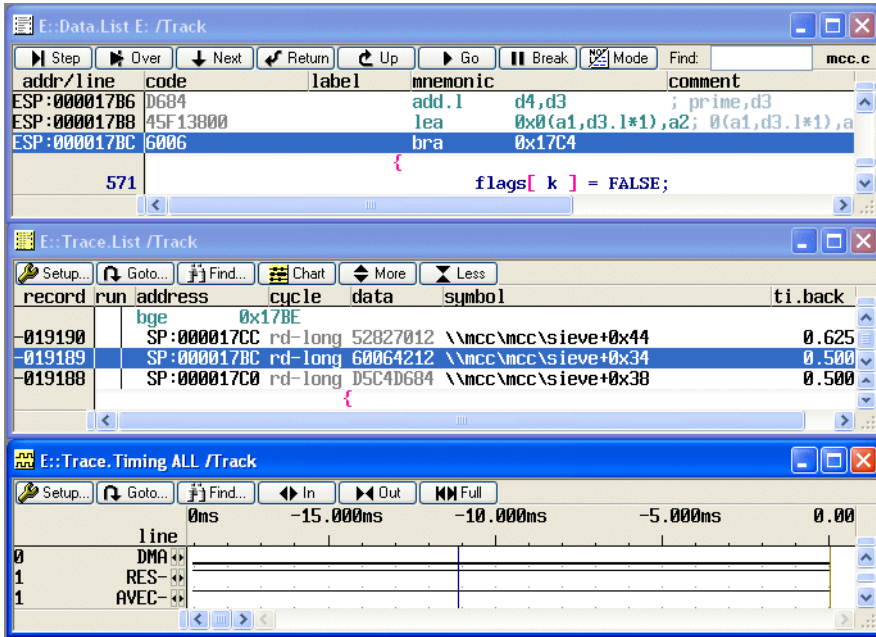
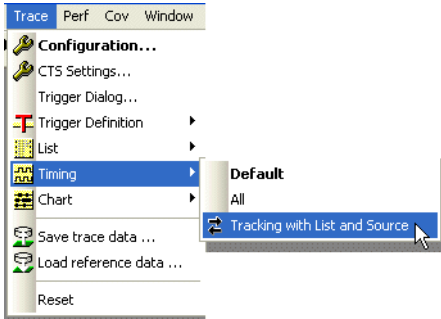
Find a specific entry in the Trace

Trace.Timing



Use *In* to modify the horizontal scaling factor





All windows are opened with the */Track* option.

The cursor in the inactive window will always follow the cursor in the active window.

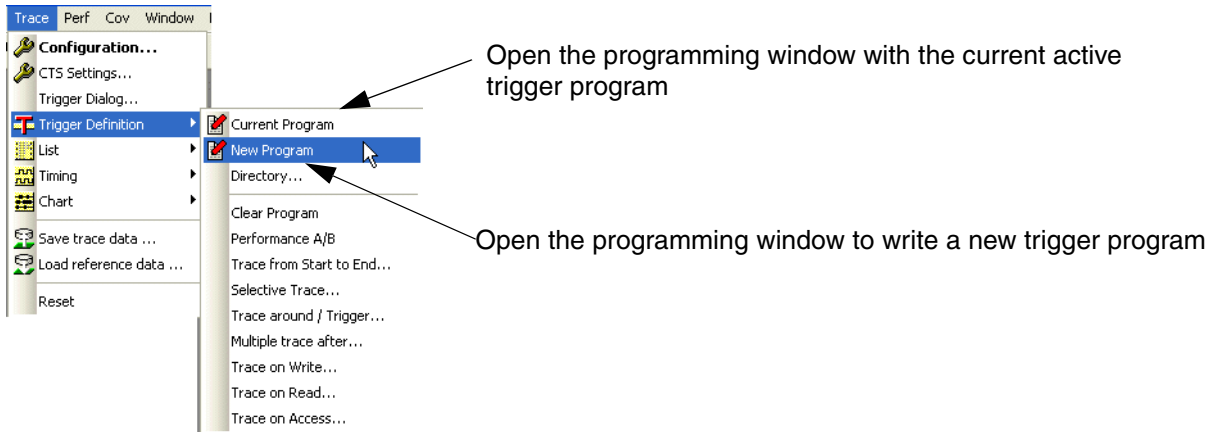
Trace.Timing

Display trace contents as timing diagram

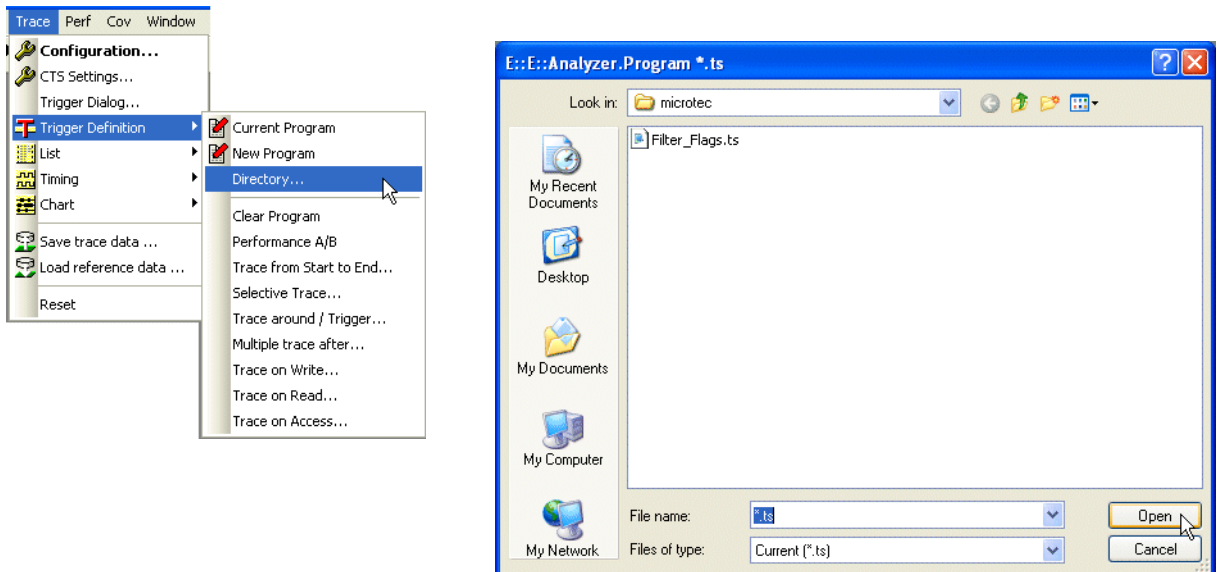
Trigger Programming

Standard extension for trigger programs is **.TS**.

The Trigger Programming Window

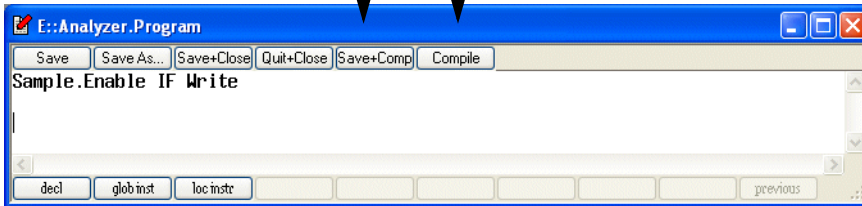


Open the programming window for an already existing trigger program.



Compile the trigger program and load it into the analyzer trigger unit and save it

Compile the trigger program and load it into the analyzer trigger unit



Softkey support for the trigger programming

Analyzer.Program

Open an Analyzer Program window to program the Analyzer Trigger Unit.

Analyzer.ReProgram

Program a ready-to-run trigger program into the Analyzer Trigger Unit.

First Examples for a Trigger Program

Example:

Sample.Enable IF Write
└──────────┘ └──┘
Output command/ Condition
Action

Structure of instructions for the analyzer trigger unit:

<output_command> if <condition>

A condition is a combination of **input events**.

If no condition is defined, the output command is always executed.

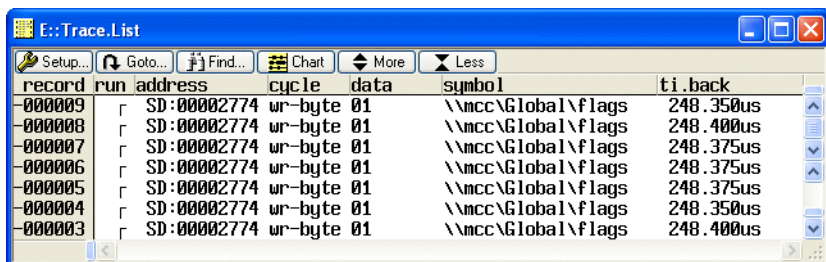
Sample.Enable
└──────────┘
Output command

If a condition is defined, the output command is only executed when the condition is true.

Example: Sample all write accesses to the address flags

```
ADDRESS AlphaBreak flags
```

```
Sample.Enable IF AlphaBreak&&Write
```



record	run	address	cycle	data	symbol	ti.back
-000009	┌	SD:00002774	wr-byte	01	\\mcc\Global\flags	248.350us
-000008	┌	SD:00002774	wr-byte	01	\\mcc\Global\flags	248.400us
-000007	┌	SD:00002774	wr-byte	01	\\mcc\Global\flags	248.375us
-000006	┌	SD:00002774	wr-byte	01	\\mcc\Global\flags	248.375us
-000005	┌	SD:00002774	wr-byte	01	\\mcc\Global\flags	248.375us
-000004	┌	SD:00002774	wr-byte	01	\\mcc\Global\flags	248.350us
-000003	┌	SD:00002774	wr-byte	01	\\mcc\Global\flags	248.400us

Trigger Conditions

The following logical operators are available to combine input events to a trigger condition:

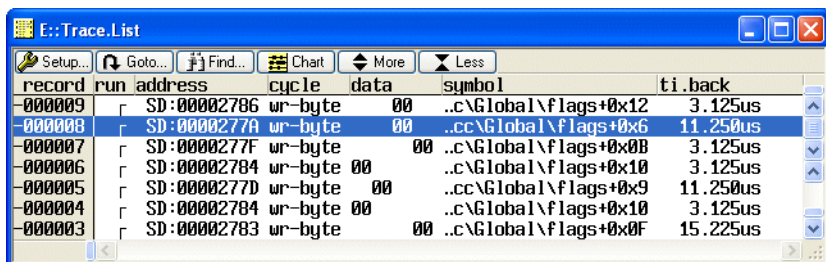
AND	&&
OR	
XOR	^^
NOT	!

Example: Sample all cycles where a 0 is written to the array flags

```
ADDRESS AlphaBreak V.RANGE(flags)
```

```
DATA.B Zero 0
```

```
Sample.Enable IF AlphaBreak&&Write&&Zero
```



record	run	address	cycle	data	symbol	ti.back
-000009	┌	SD:00002786	wr-byte	00	..c\Global\flags+0x12	3.125us
-000008	┌	SD:0000277A	wr-byte	00	..c\Global\flags+0x6	11.250us
-000007	┌	SD:0000277F	wr-byte	00	..c\Global\flags+0x0B	3.125us
-000006	┌	SD:00002784	wr-byte	00	..c\Global\flags+0x10	3.125us
-000005	┌	SD:0000277D	wr-byte	00	..c\Global\flags+0x9	11.250us
-000004	┌	SD:00002784	wr-byte	00	..c\Global\flags+0x10	3.125us
-000003	┌	SD:00002783	wr-byte	00	..c\Global\flags+0x0F	15.225us

Combined Output Commands

```
ADDRESS AlphaBreak flags  
EVENtCouNTER ev1 5000.
```

```
Count.Enable ev1, Sample.Enable IF AlphaBreak  
Trigger.A IF ev1&&AlphaBreak
```

Several output commands can be activated if a condition becomes true. They are separated by comma.

Structure of a Trigger Program

A trigger program has the following structure:

- Declarations
- Global Instructions
- Local Instructions

Declarations

Declarations define an input event which need to be declared (assign name and value to an input event).

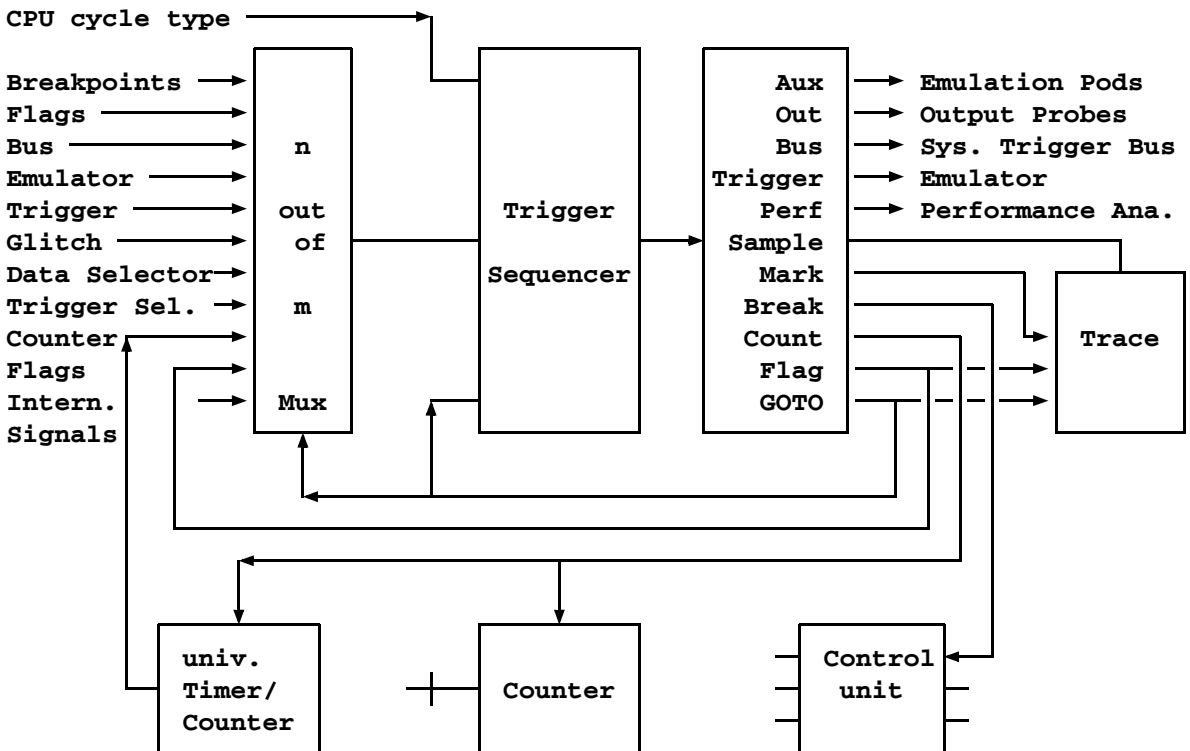
ADDRESS AlphaBreak flags

DATA.B zero 0

Global Instructions

Global Instructions are valid in all levels of a trigger program.

All global instructions are executed at the same time.



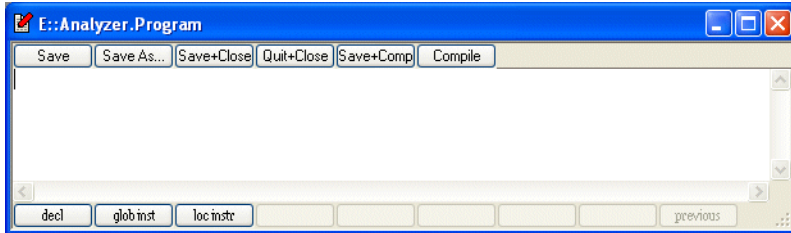
Hardware structure of the trigger unit

Local Instructions

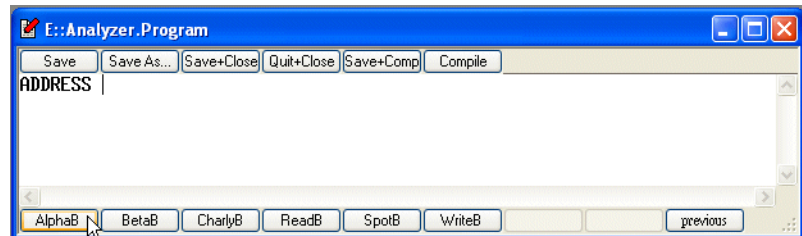
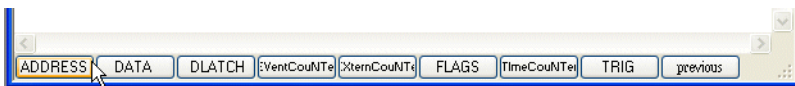
Local Instructions are assigned to a trigger level. They are only activated, when their trigger level is active.

HA120	8
HAC	2 (3)
ECC8	4

The softkey support for the trigger programming reflects this structure:



Declarations:



Selective Tracing

Output commands to control selective tracing:

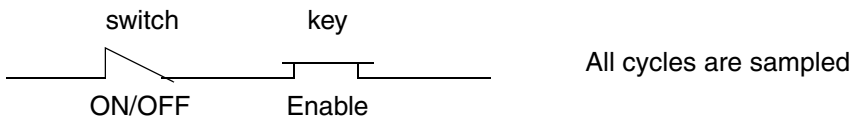
Sample.Enable [if <condition>]

Sample.ON [if <condition>]

Sample.OFF [if <condition>]

The instructions **Sample.ON**, **Sample.OFF** and **Sample.Enable** work as a controlled switch and a key in series.

Example 1: No sample command in the trigger program



Example 2: Only **Sample.Enable** commands, no **Sample.ON/OFF** in the trigger program

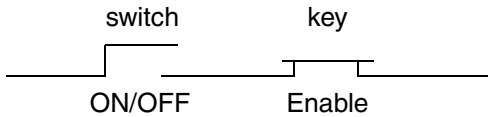
	We recommend to use Sample.enable whenever possible.
--	---



Sample only write accesses to the variable flags

```
ADDRESS AlphaBreak V.RANGE(flags)
Sample.Enable IF AlphaBreak&&Write
```

Example 3: Only Sample.ON/OFF instructions, no Sample.Enable in the trigger program



Sampling is controlled via the **Sample.ON/OFF** command



The initial state after RESet or Init is Sample.ON

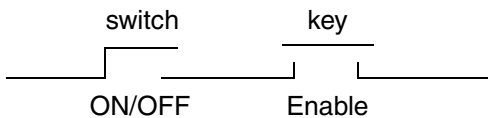
Example: Sample only the function sieve

```
ADDRESS AlphaBreak P:0x1776          ; Call of sieve
ADDRESS BetaBreak P:0x17DA          ; Return from sieve

Level1:
  Sample.OFF                          ; Switch of sampling
  CONTinue

Level2:
  Sample.ON IF AlphaBreak             ; Start sampling (next cycle)
  Sample.OFF IF BetaBreak            ; Stop sampling
```

Example 4: Sample.ON/OFF instructions and Sample.Enable in the trigger program



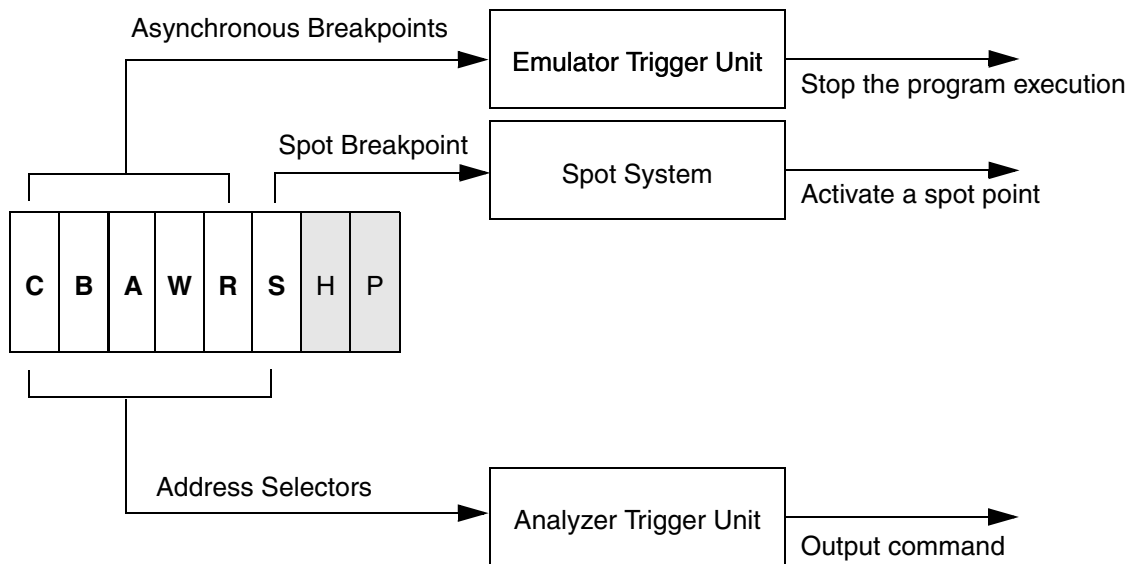
Sampling is controlled via the Sample.ON/OFF and Sample.Enable command

Address Selectors

Address selectors are used to mark addresses that should be used e.g. for selective tracing.

The following address selectors are available:

AlphaBreak	HA120, ECC8
BetaBreak	HA120, ECC8
CharlyBreak	HA120, ECC8
SpotBreak	HA120
ReadBreak	HA120
WriteBreak	HA120



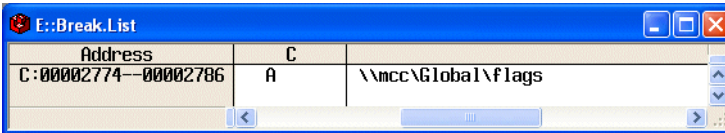
If a breakpoint is used as an address selector, it has to be disabled as a breakpoint.

Declaration: ADDRESS <breakpoint> <address>

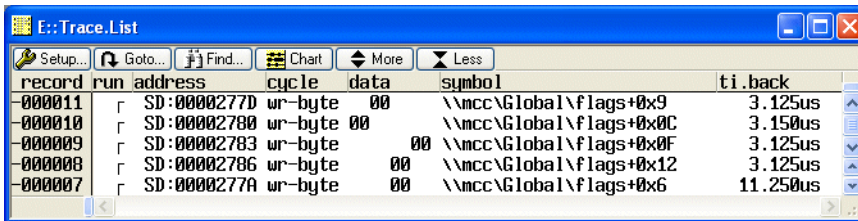
It is not necessary to declare the address selectors, if the corresponding breakpoint is set outside of the trigger program.

```
ADDRESS AlphaBreak V.RANGE(flags)
```

```
Sample.Enable IF AlphaBreak&&Write
```



Address	C	
C:00002774--00002786	A	\\mcc\Global\flags



record	run	address	cycle	data	symbol	ti.back
-000011		SD:0000277D	wr-byte	00	\\mcc\Global\flags+0x9	3.125us
-000010		SD:00002780	wr-byte	00	\\mcc\Global\flags+0x0C	3.150us
-000009		SD:00002783	wr-byte	00	\\mcc\Global\flags+0x0F	3.125us
-000008		SD:00002786	wr-byte	00	\\mcc\Global\flags+0x12	3.125us
-000007		SD:0000277A	wr-byte	00	\\mcc\Global\flags+0x6	11.250us



If a breakpoint type is used as an address selector, all set breakpoints of this type are deleted, when the trigger program is compiled.

More examples:

```
ADDRESS AlphaBreak 1000
```

```
ADDRESS AlphaBreak V.RANGE(flags)
```

```
ADDRESS AlphaBreak V.RANGE(flags[4])
```

```
ADDRESS AlphaBreak sieve func1 func2
```

```
ADDRESS AlphaBreak sieve||func1||func2 ;|identical to previous
```

Declaration: Data.<width> <name> <data>

Examples for available data selectors:

- B0** Byte transfer, data positioned at D0--7
- B1** Byte transfer, data positioned at D8--15
- B** Byte (dynamic data selector), data positioned anywhere
- W0** Word transfer, data positioned at D0--15
- W1** Word transfer, data positioned at D15--D31
- W** Word transfer (dynamic data selector), any alignment, any bus size
- L** Long transfer

Available data selectors:

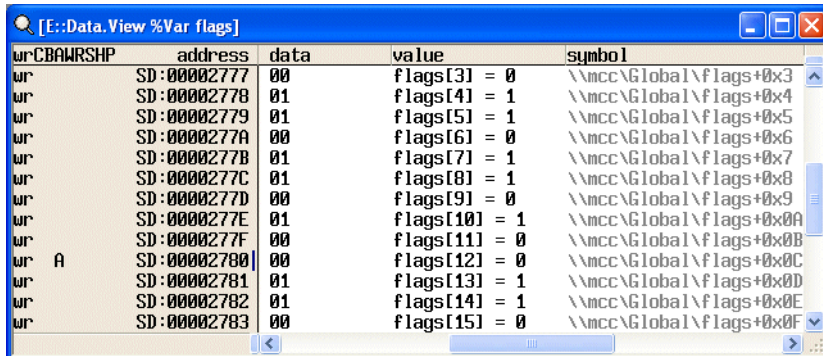
2 * 16 bit 1 * 32 bit on each trigger level	HA120
2 * 8 bit 1 * 16 bit on each trigger level	ECC8

Example

Sample all cycles where a 01 is written to flags[12].

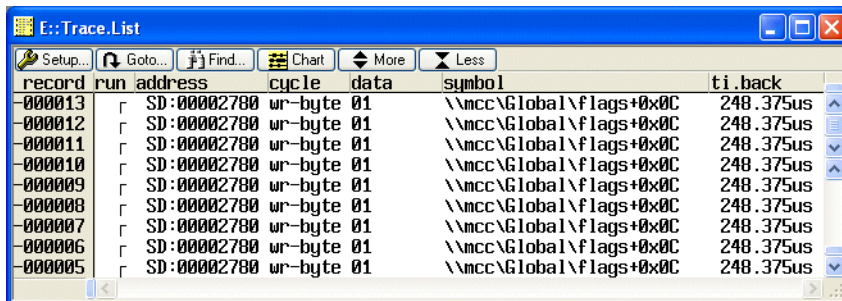
```
ADDRESS AlphaBreak V.RANGE(flags[12])
DATA.B one 01
```

```
Sample.Enable IF AlphaBreak&&one&&Write
```



[E::Data.View %Var flags]

wrCBAWRSH	address	data	value	symbol
wr	SD:00002777	00	flags[3] = 0	\\mcc\Global\flags+0x3
wr	SD:00002778	01	flags[4] = 1	\\mcc\Global\flags+0x4
wr	SD:00002779	01	flags[5] = 1	\\mcc\Global\flags+0x5
wr	SD:0000277A	00	flags[6] = 0	\\mcc\Global\flags+0x6
wr	SD:0000277B	01	flags[7] = 1	\\mcc\Global\flags+0x7
wr	SD:0000277C	01	flags[8] = 1	\\mcc\Global\flags+0x8
wr	SD:0000277D	00	flags[9] = 0	\\mcc\Global\flags+0x9
wr	SD:0000277E	01	flags[10] = 1	\\mcc\Global\flags+0x0A
wr	SD:0000277F	00	flags[11] = 0	\\mcc\Global\flags+0x0B
wr A	SD:00002780	00	flags[12] = 0	\\mcc\Global\flags+0x0C
wr	SD:00002781	01	flags[13] = 1	\\mcc\Global\flags+0x0D
wr	SD:00002782	01	flags[14] = 1	\\mcc\Global\flags+0x0E
wr	SD:00002783	00	flags[15] = 0	\\mcc\Global\flags+0x0F



E::Trace.List

record	run	address	cycle	data	symbol	ti.back
-000013	r	SD:00002780	wr-byte 01	01	\\mcc\Global\flags+0x0C	248.375us
-000012	r	SD:00002780	wr-byte 01	01	\\mcc\Global\flags+0x0C	248.375us
-000011	r	SD:00002780	wr-byte 01	01	\\mcc\Global\flags+0x0C	248.375us
-000010	r	SD:00002780	wr-byte 01	01	\\mcc\Global\flags+0x0C	248.375us
-000009	r	SD:00002780	wr-byte 01	01	\\mcc\Global\flags+0x0C	248.375us
-000008	r	SD:00002780	wr-byte 01	01	\\mcc\Global\flags+0x0C	248.375us
-000007	r	SD:00002780	wr-byte 01	01	\\mcc\Global\flags+0x0C	248.375us
-000006	r	SD:00002780	wr-byte 01	01	\\mcc\Global\flags+0x0C	248.375us
-000005	r	SD:00002780	wr-byte 01	01	\\mcc\Global\flags+0x0C	248.375us

More examples:

Data.B Error_Val 0x0ff; hex. value

Data.B Error_Val 0x0xf; hex mask

Data.B Error_Val 0y0xxxx1xx0; binary mask

Data.B Error_Val 11 22 33 ; one of the 3 values

Stop the Analyzer Recording

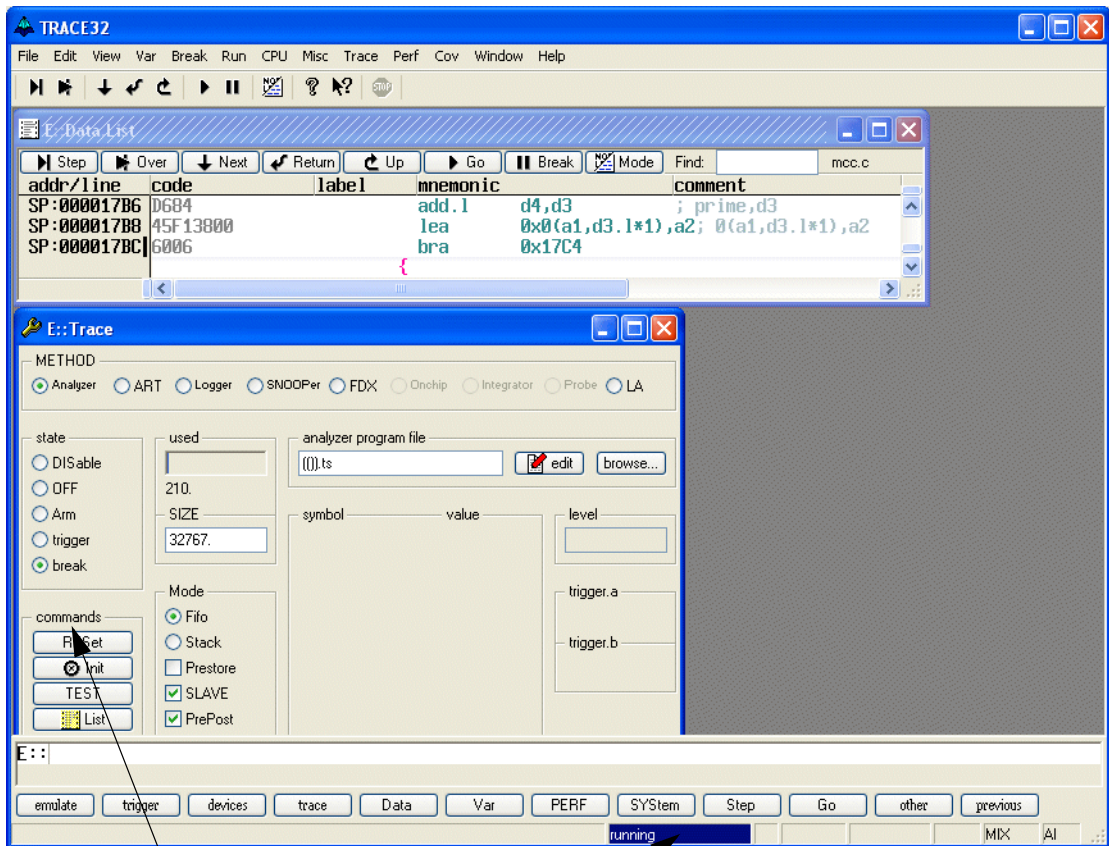
Break IF <condition>

Stops the recording in the trace buffer and deactivates the analyzer trigger unit at the specified condition.

Example: Stop the trace when 0 is written to the array flags

```
ADDRESS AlphaB v.range(flags)
DATA.B Zero 0

BREAK IF AlphaB&&Write&&Zero
```

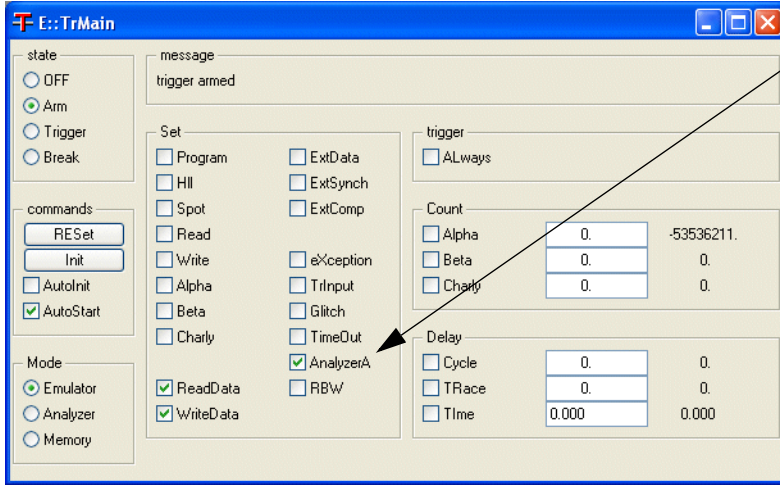


Trace is in the break state

running in blue indicates that the program execution is still running, but the trace is stopped

Trigger.A IF <condition>

Stop the program at the specified condition

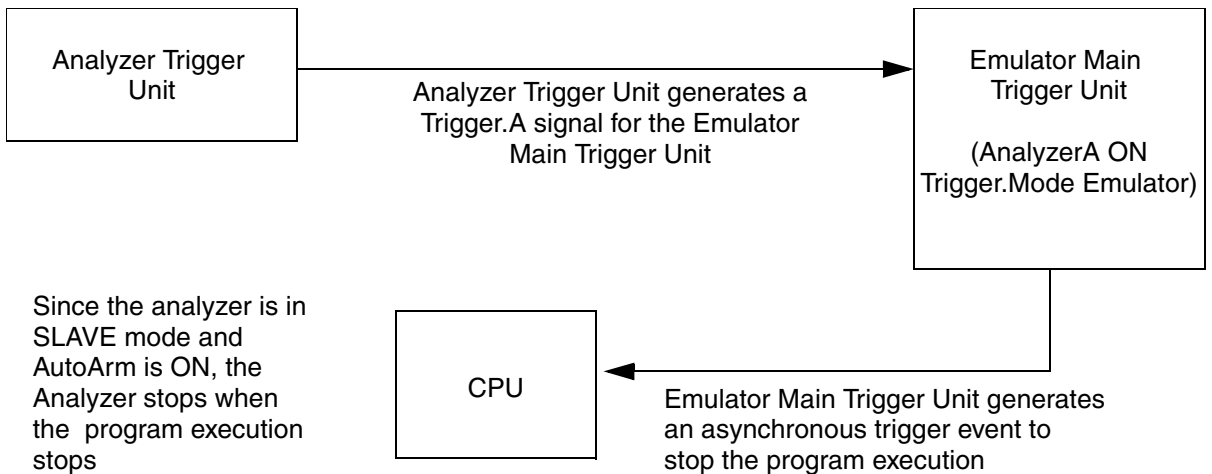
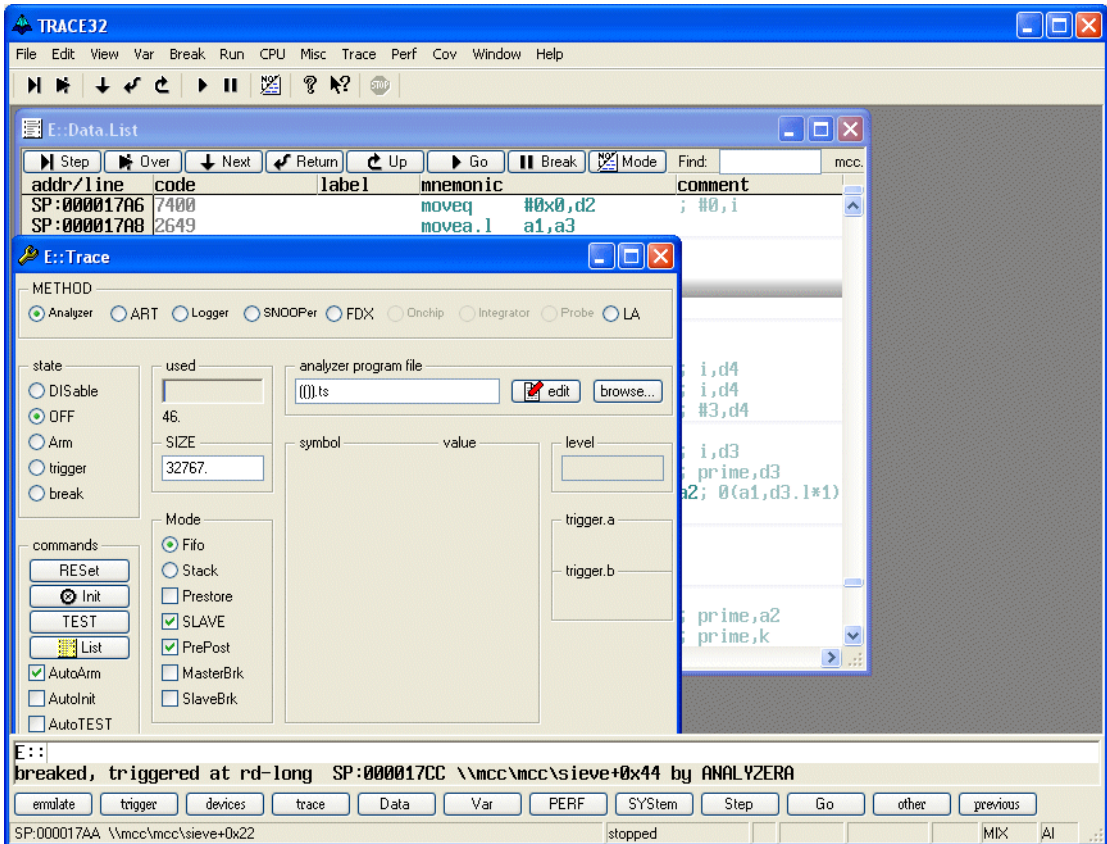


The program execution can only be stopped by an Trigger.A command if AnalyzerA is set in the Emulator Main Trigger Unit

Example: Stop the program when 0 is written to the array flags

```
ADDRESS AlphaBreak V.RANGE(flags)
DATA.B Zero 0

Trigger.A IF AlphaBreak&&Write&&Zero
```



Available event counters:

HA120	3 * 48 bit restart cap.	Max. Count 5.5e11
ECC8	2 * 16 bit or + 1 * 32 bit restart cap.	Max. Count 65535 or 4.2e9
HAC	2+1	65535

Declaration: EVentCouNter <name> [<event>]

Instructions:

Counter.Increment <name> [if <condition>];Counter is incremented whenever the condition is true

Counter.ON <name> [if <condition>];Counter is incremented each CPU cycle until Counter.OFF

Counter.OFF <name> [if <condition>]

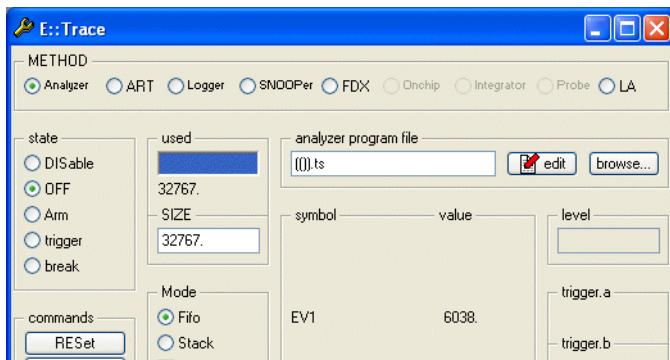
Counter.Restart <name> [if <condition>];Counter is reset to 0

Example 1

Counter ev1 counts how often something is written to flags[4].

```
ADDRESS AlphaBreak V.RANGE(flags[4])
EventCouNter ev1

Counter.Increment ev1 if AlphaBreak&&Write
```



Example 2

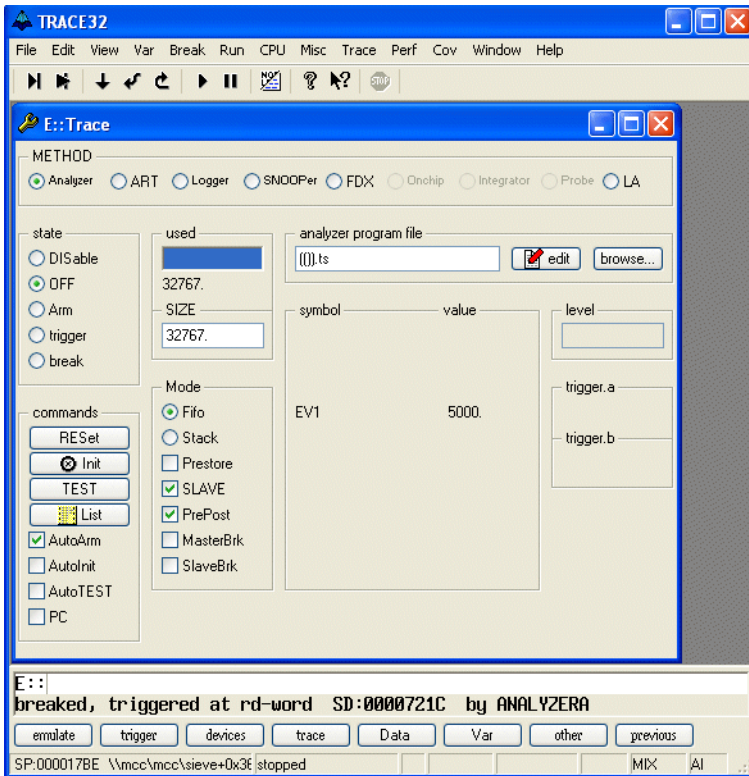
The program execution is stopped, after 5000. write accesses to the variable flags.

```
ADDRESS AlphaBreak V.RANGE(flags)
EventCouNTER ev1 5000.
```

```
Counter.Increment ev1 IF AlphaBreak&&Write
Trigger.A IF ev1
```



The input event <counter name> is **TRUE** when the counter has reached its final value.



Example 3

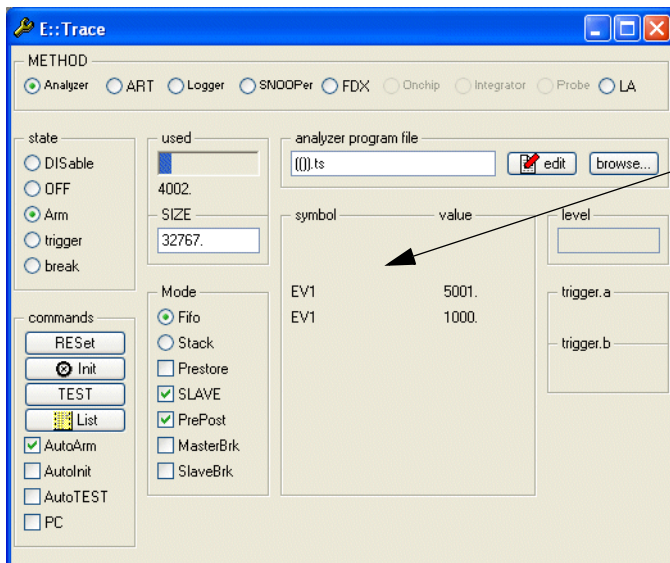
The trace buffer samples the 1000. up to the 5000. write access to flags[4].

```
ADDRESS AlphaBreak V.RANGE(flags[4])
EventCounter ev1 1000.--5000.
```

```
Counter.Increment ev1 IF AlphaBreak&&Write
Sample.Enable IF ev1&&AlphaBreak&&Write
```



The input event <counter name> is **TRUE** when the counter value is within the specified counter range.



If the counter value is a range, the analyzer trigger unit needs 2 counters

Available time counters:

Event counter and time counter share the same resources.

HA120	3 * 48 bit Restart cap.	Max. time 3.8 hours	Resolution 100 ns
ECC8	2 * 16 bit or 1 * 32 bit restart cap.	Max. time 6.5 ms or 7.1 min	Resolution 100 ns
HAC	1	65 ms	Resolution 100 ns

Time values can be entered in the following units:

Nanoseconds (ns)

Microseconds (μ s)

Milliseconds (ms)

Seconds (s)

Kiloseconds (ks)

Declaration: TimeCouNter <name> [<time>]

Commands:

Counter.Increment <name> [if <condition>];

Counter.ON <name> [if <condition>]

Counter.OFF <name> [if <condition>]

Counter.Restart <name> [if <condition>]

Example

Stop the program execution, if the function sieve need longer, then 250.us.

```
ADDRESS AlphaBreak sieve
ADDRESS BetaBreak P:0x17DA ; end of sieve
TImeCouNTER sievec 250.us

start:
  Counter.OFF sievec
  CONTinue

  level1:

  Counter.ON sievec IF AlphaBreak
  Counter.Restart sievec IF BetaBreak
  Trigger.A IF sievec&&!BetaBreak
```

Trigger Levels

Available trigger levels:

HA120	8
ECC8	4
HAC	

- A trigger level start with a label
- A trigger level ends at the next label or with the end of the program
- Only one trigger level can be active
- The first level, that is active, when a trigger program is started is the level named "start". If no level "start" is used in the trigger program, the first level is the active level.

Commands to change the trigger levels:

CONTInue - sequential level switch, continue with the next trigger level

GOTO <label> - free level change

Example

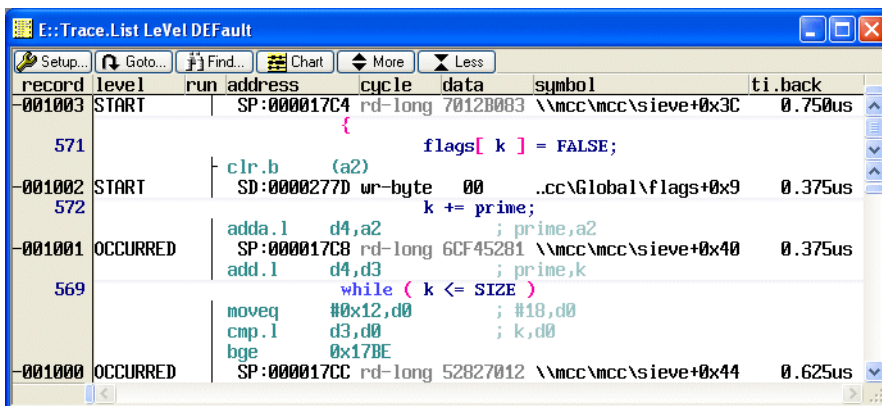
Sample another 1000. records to the trace buffer, after the first write access to the variable flags.

```
ADDRESS AlphaBreak V.RANGE(flags)
EventCounTter delay 1000.
```

```
start:
    CONTINUE IF AlphaBreak&&Write
```

```
occured:
    Counter.Increment delay
    BREAK IF delay
```

Trace.List LeVel DEFault



record	level	run address	cycle	data	symbol	ti.back
-001003	START	SP:000017C4	rd-long	7012B003	\\mcc\mcc\sieve+0x3C	0.750us
571				{	flags[k] = FALSE;	
-001002	START	SD:0000277D	wr-byte	00	..cc\Global\flags+0x9	0.375us
572					k += prime;	
-001001	OCCURRED	adda.l d4,a2			; prime,a2	
		SP:000017C8	rd-long	6CF45281	\\mcc\mcc\sieve+0x40	0.375us
		add.l d4,d3			; prime,k	
569					while (k <= SIZE)	
		moveq #0x12,d0			; #10,d0	
		cmp.l d3,d0			; k,d0	
		bge 0x17BE				
-001000	OCCURRED	SP:000017CC	rd-long	52827012	\\mcc\mcc\sieve+0x44	0.625us

Available markers:

HA120	A, B, C
ECC8	A, B
HAC	

Commands for markers:

Mark.<name> [if <condition>]

Example 1

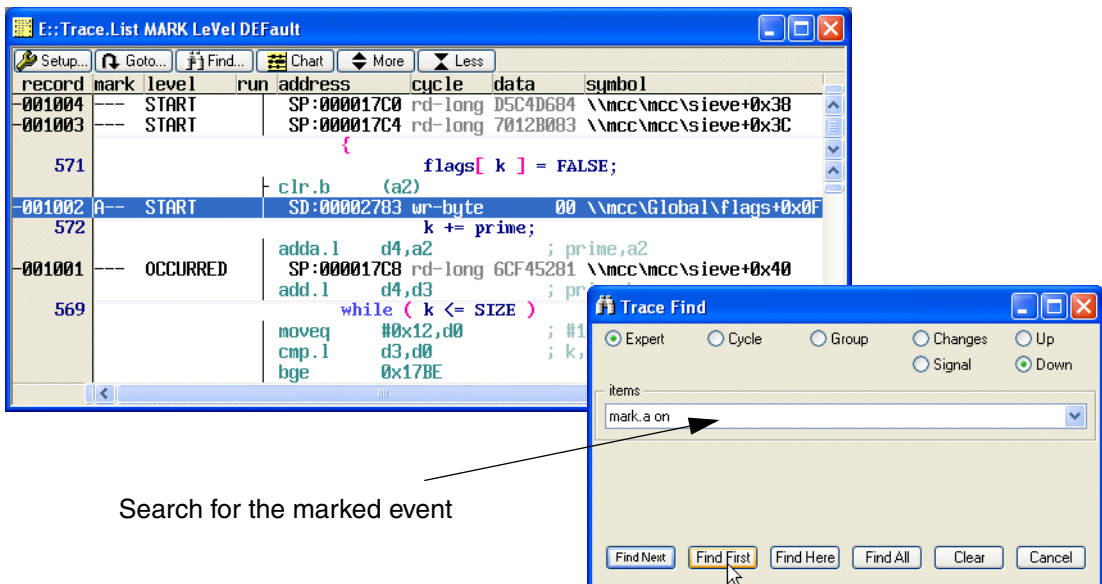
Sample another 1000. records to the trace buffer, after the first write access to the variable flags and mark this write access in the trace buffer.

```
ADDRESS AlphaBreak V.RANGE(flags)
EventCounter delay 1000.

start:
    Mark.A IF AlphaBreak&&Write
    Continue IF AlphaBreak&&Write

occured:
    Counter.Increment delay
    BREAK IF delay
```

Trace.List MARK LeVel DEFault



Search for the marked event

Example 2

Sample only the function sieve.

```
ADDRESS AlphaBreak sieve
ADDRESS BetaBreak P:0x17DA

start:
    Mark.A IF AlphaBreak
    Sample.Enable IF AlphaBreak
    CONTinue IF AlphaBreak

infunc:
    Sample.Enable
    Mark.B IF BetaBreak
    GOTO start IF BetaBreak
```

	HA120	HAC	ECC8
Trigger Inputs	2 channels / 2 * 8 bit 1 coax input A	1 channels / 12 * 8 bit	1 channel / 1*8 bit 1 coax input A

Declaration: **TRIG.A** <name> <data>;Input TRIGGER A

TRIG.B <name> <data>;Input TRIGGER B

INOUTA;Coax input A

```
TRIG.A   T_SEL1  0y0xxxxx11xx  
BREAK   IF      T_SEL1
```

```
BREAK   IF      INOUTA
```

More examples:

TRIG.A Error_Val 0x0ff; hex. value

TRIG.A Error_Val 0x0xf; hex mask

TRIG.A Error_Val 0y0xxxx1xx0; binary mask

TRIG.A Error_Val 11 22 33 ; one of the 3 values

Trigger Outputs

	HA120	HAC	ECC8
Trigger Outputs	1 coax output A 2 outputs (C, D)		1 coax output A 2 outputs (C, D)

Commands:

OUT.<pin> [if <condition>]

Example

Stimulate coax output A if function sieve is entered.

```
ADDRESS AlphaBreak sieve
OUT.A          IF          AlphaBreak
```

Stimulate coax output A for 10.ms every time when the function sieve is entered.

```
ADDRESS AlphaBreak sieve
TImeCouNTER C_Delay 10.ms

start:
    Counter.Restart C_Delay
    CONTinue IF AlphaBreak
level1:
    OUT.A IF !C_Delay
    Counter.Increment C_Delay
    GOTO start IF C_Delay
```



Rule 1:
Instructions are executed from left to right.

```
Counter.ON count1, Counter.OFF count1 IF AlphaBreak
```



Rule 2:
Instructions are executed top down.

```
GOTO Count_Level  
GOTO Error_Level IF INOUTA
```

```
Count_Level:  
.  
Error_Level:  
.  
.
```



Rule 3:
Global statements have lower priority than local statements. (see Rule 2)

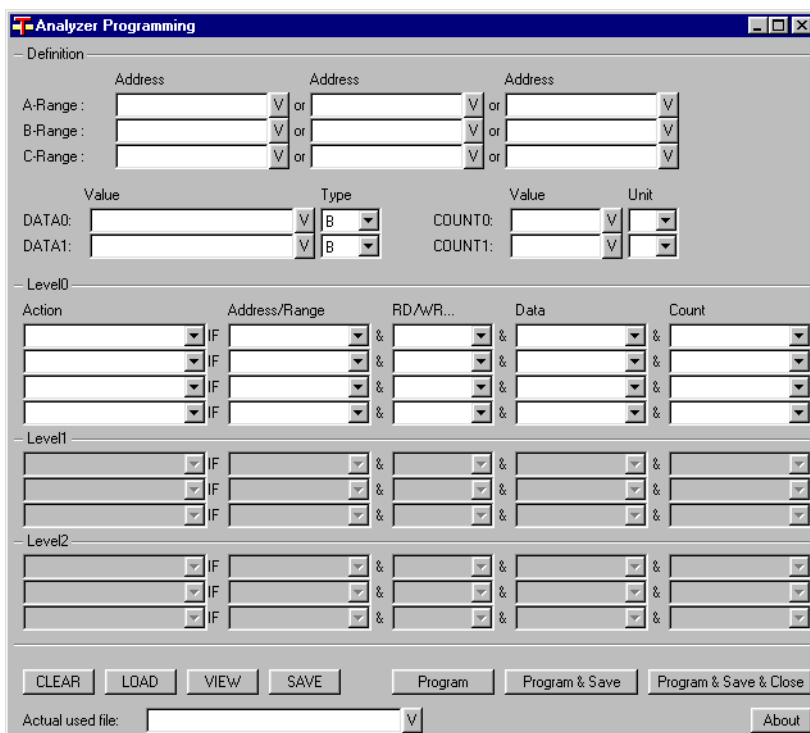
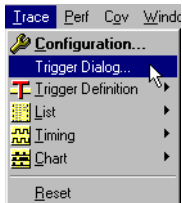


Rule 4:
Commands that change the trigger level (CONTInue, GOTO) can not be combined with other output commands. This rule applies also to the BREAK command.

The Analyzer Programming Dialog Window

- The intension of the Analyzer Programming Dialog Window is to provide an intuitive and easy to use interface to program the Analyzer Trigger Unit.
- The Analyzer Programming Dialog Window does not represent the full functionality of the Analyzer Trigger Unit.

How to Start



The 'Analyzer Programming' dialog window is shown. It contains the following sections:

- Definition:** Fields for A-Range, B-Range, and C-Range, each with an 'Address' input and a 'V' button. Below these are fields for DATA0, DATA1, COUNT0, and COUNT1, each with a 'Value' input, a 'Type' dropdown (set to 'B'), and a 'Unit' dropdown.
- Level0:** A table with columns: Action, Address/Range, RD/WR..., Data, and Count. It contains three rows of configuration options.
- Level1:** A table with the same columns as Level0, containing three rows.
- Level2:** A table with the same columns as Level0, containing three rows.
- Buttons:** CLEAR, LOAD, VIEW, SAVE, Program, Program & Save, Program & Save & Close.
- Actual used file:** A text input field with an 'About' button.

First Example

Example:

Sample all write accesses to the variable flags.

1. Define the address range (here the address range of the variable flags).
2. Select the action (here selective tracing) and define the condition on which the action should be executed (here write accesses to the address/range defined in step 1).

Define the Address Range

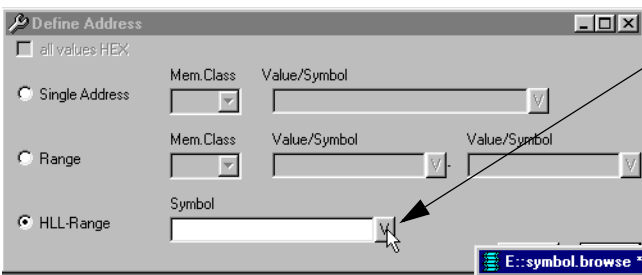
The Analyzer Trigger Dialog Box allows the definition of 3 different address ranges, that can be used by a Trigger Program:

A-Range, B-Range, C-Range

The image shows the 'Analyzer Programming' dialog box. The 'Definition' section has three rows for 'A-Range', 'B-Range', and 'C-Range', each with an 'Address' field and a 'V' button. The 'Action' section has three levels (Level0, Level1, Level2) with 'IF' dropdowns. The 'Define Address' sub-dialog box is open, showing options for 'all values HEX', 'Single Address', 'Range', and 'HLL-Range'. The 'HLL-Range' option is selected. The 'Define Address' dialog has fields for 'Mem. Class', 'Value/Symbol', and 'Symbol'. Arrows point from text annotations to the 'V' buttons in the 'A-Range' field and the 'HLL-Range' radio button.

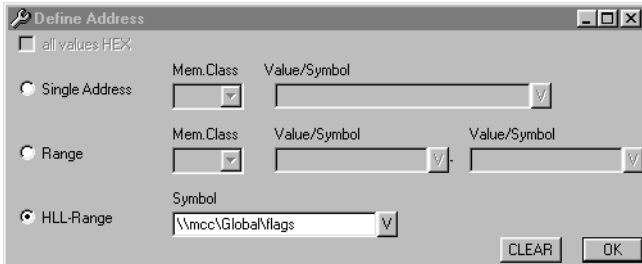
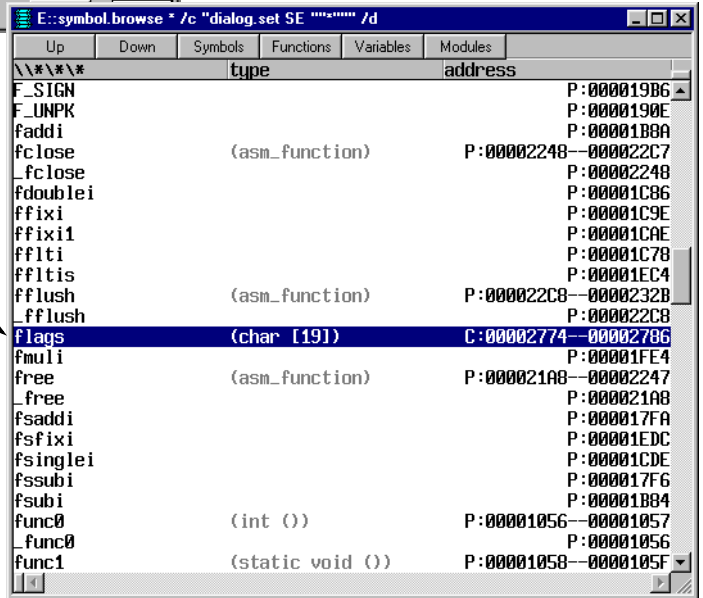
Click here to open the "Define Address" dialog box to define the address/range that should be used as A-range

Select **HLL-Range** to define the address on HLL level



Click here to open the browser window for all HLL symbols

Select the variable flags by a double click (fast browsing through first letters possible)



Select the Action

An Analyzer Trigger Program can execute **Actions** depending on the current state of the user program and the target hardware.

Definition

A-Range: or or

B-Range: or or

C-Range: or or

DATA0: Type: COUNT0: Unit:

DATA1: Type: COUNT1: Unit:

Level0

Action	Address/Range	RD/WR...	Data	Count
<input type="text" value="IF"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Sample	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Stop Analyzer	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Stop CPU&Analyzer	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Goto Level1	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Goto Level2	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Inc COUNT0	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Inc COUNT1	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Restart COUNT0	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Restart COUNT1	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Mark A	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Mark B	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Mark C	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text" value="IF"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

CLEAR LOAD VIEW Program Analyzer Program Analyzer & Close

Actual used file: About

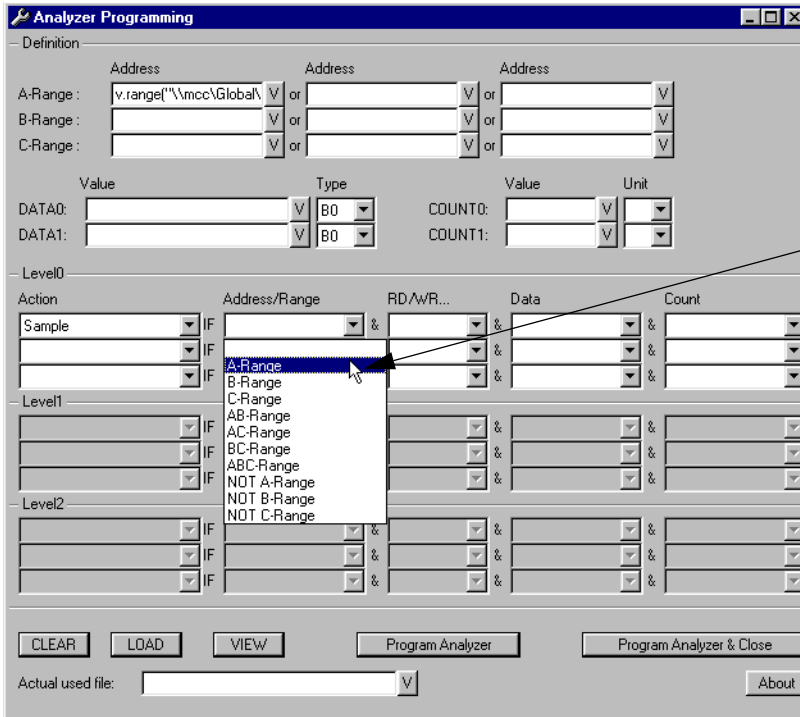
Select the action **Sample** to do a selective trace

Define the Condition

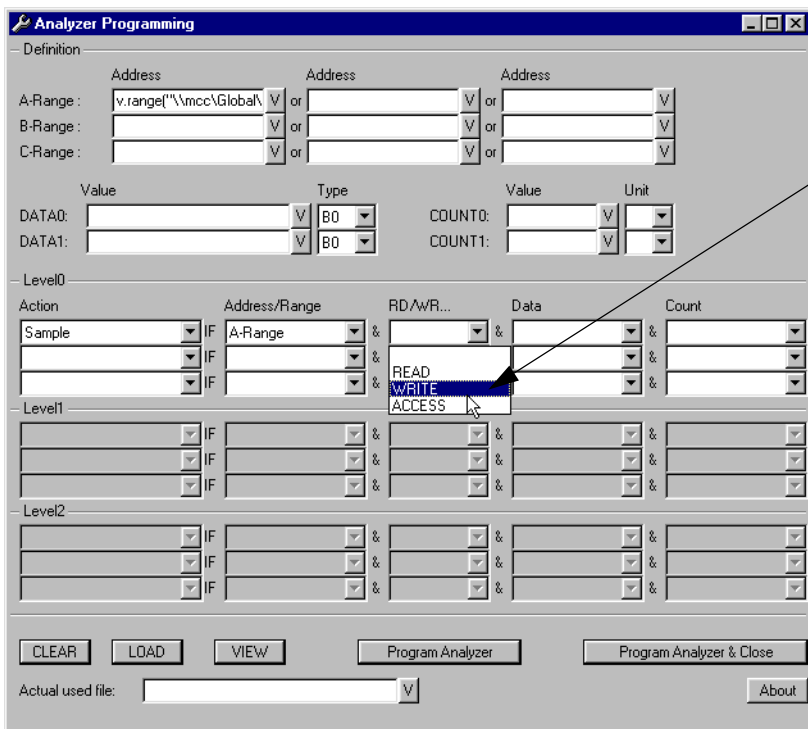
The current state of the user program/target hardware can be described by different characteristics. The different characteristics can be combined by a LOGICAL AND to form the condition on which the action should be executed.

The following characteristics can be combined to form a condition:

Address/Range	Address on the address bus
RD/WR	Cycles type Read access, write access, any access ...
Data	Data on the data bus
Count	State of an event or time counter



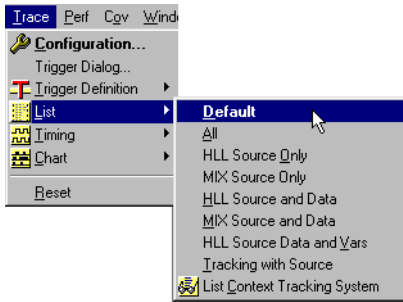
The address/range defined by A-Range is the characteristic for address bus



Write access is the characteristic for the cycles type

Display the Result

Start and stop the user program.



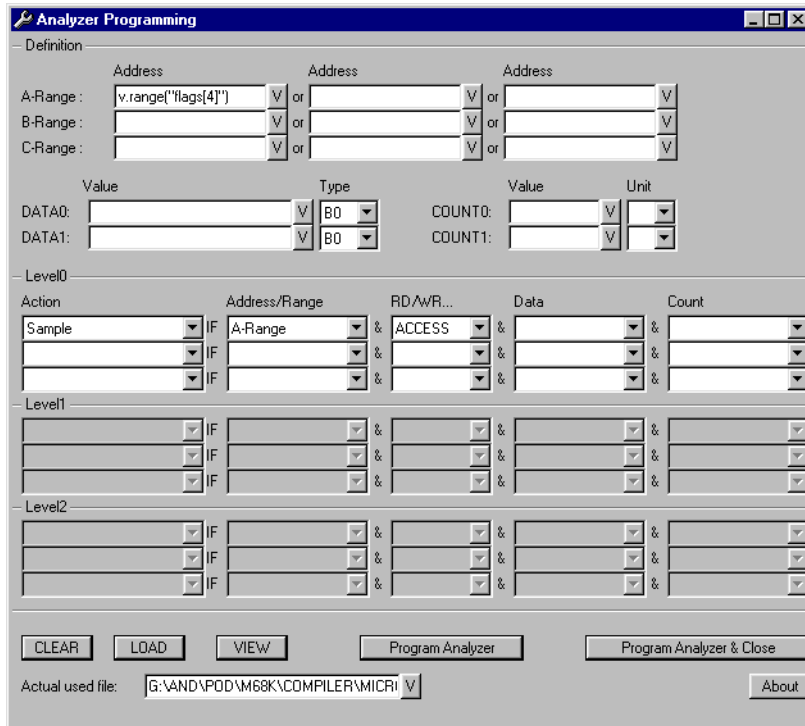
The screenshot shows a window titled 'F::TRace.List' with a table of trace records. The table has columns: record, run, address, cycle, d.l, symbol, and ti.back. The records are as follows:

record	run	address	cycle	d.l	symbol	ti.back
-00000008	f r	D:0000678A	wr-byte	01	\\thumble\Global\flags+0x0E	4.500us
-00000007	f r	D:0000678B	wr-byte	01	\\thumble\Global\flags+0x0F	4.500us
-00000006	f r	D:0000678C	wr-byte	01	\\thumble\Global\flags+0x10	4.500us
-00000005	f r	D:0000678D	wr-byte	01	\\thumble\Global\flags+0x11	4.500us
-00000004	f r	D:0000678E	wr-byte	01	\\thumble\Global\flags+0x12	4.500us
-00000003	f r	D:0000677F	wr-byte	00	\\thumble\Global\flags+0x3	8.000us
-00000002	f r	D:00006782	wr-byte	00	\\thumble\Global\flags+0x6	3.000us
-00000001	BRK					9.760us

Sample any Access

1. Use A-Range to define the memory location/variable.
2. Select the action Sample and specify the condition by using the characteristic Address/Range and RD/WR...

Example: Sample all accesses to flags[4].



Sample all read and write cycles to the Address/Range defined by A-Range

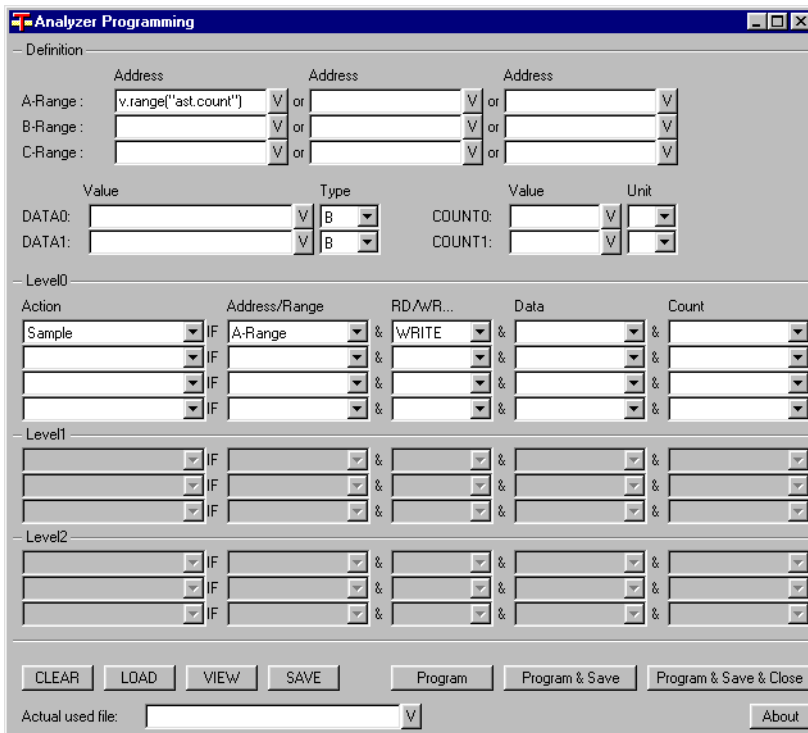
F:Trace.List

record	run	address	cycle	d.l	symbol	ti.back
00000009	f r	D:00006780	wr-byte	01	\\thumble\Global\flags+0x4	123.000us
00000008	f r	D:00006780	rd-byte	01	\\thumble\Global\flags+0x4	126.000us
00000007	f r	D:00006780	wr-byte	01	\\thumble\Global\flags+0x4	123.000us
00000006	f r	D:00006780	rd-byte	01	\\thumble\Global\flags+0x4	126.000us
00000005	f r	D:00006780	wr-byte	01	\\thumble\Global\flags+0x4	123.000us
00000004	f r	D:00006780	rd-byte	01	\\thumble\Global\flags+0x4	126.000us
00000003	f r	D:00006780	wr-byte	01	\\thumble\Global\flags+0x4	123.000us
00000002	f r	D:00006780	rd-byte	01	\\thumble\Global\flags+0x4	126.000us
00000001	BRK					25.240us

Sample all Write Accesses

1. Use A-Range to define the memory location/variable.
2. Select the action Sample and specify the condition by using the characteristic Address/Range and RD/WR...

Example: Sample all write accesses to ast.count.



Sample all write cycles to the Address/Range defined by A-Range

F::TTrace.List VAR DEFault

record	var	run	address	cycle	d.l	symbol	ti.back

00000003	ast.count = 12345	f r	D:00005118	wr-long	00003039	\\thumble\Global\ast+0x4	
00000002	ast.count = 12346	f r	D:00005118	wr-long	0000303A	\\thumble\Global\ast+0x4	25.000us
00000001		BRK					3.647s

Sample Write Accesses with a Specific Value

1. Use A-Range to define the memory location/variable.
2. Use DATA0 to define the data value.

To define a data value it is necessary to know if the data value is transferred as a byte, a word or a long.

3. Select the action Sample and specify the condition by using the characteristic Address/Range and RD/WR... and DATA.

Example 1: Sample all cycles where a 0 is written to flags[12]

Analyzer Programming

Definition

A-Range: v.range('flags[12]') or
 B-Range: or
 C-Range: or

Value Type Value Unit
 DATA0: 0 B COUNT0:
 DATA1: B COUNT1:

Level0

Action	Address/Range	RD/WR...	Data	Count
Sample	IF A-Range	& WRITE	& DATA0	&
	IF	&	&	&
	IF	&	&	&
	IF	&	&	&

Level1

	IF	&	&	&
	IF	&	&	&
	IF	&	&	&

Level2

	IF	&	&	&
	IF	&	&	&
	IF	&	&	&

CLEAR LOAD VIEW SAVE Program Program & Save Program & Save & Close

Actual used file: About

record	var	run	address	cycle	d.l	symbol	ti.back
-00000010	flags[12] = 0	f r	D:00006788	wr-byte	00	\\thumble\Global\flags+0x0C	249.000us
-00000009	flags[12] = 0	f r	D:00006788	wr-byte	00	\\thumble\Global\flags+0x0C	249.000us
-00000008	flags[12] = 0	f r	D:00006788	wr-byte	00	\\thumble\Global\flags+0x0C	249.000us
-00000007	flags[12] = 0	f r	D:00006788	wr-byte	00	\\thumble\Global\flags+0x0C	249.000us
-00000006	flags[12] = 0	f r	D:00006788	wr-byte	00	\\thumble\Global\flags+0x0C	249.000us
-00000005	flags[12] = 0	f r	D:00006788	wr-byte	00	\\thumble\Global\flags+0x0C	249.000us
-00000004	flags[12] = 0	f r	D:00006788	wr-byte	00	\\thumble\Global\flags+0x0C	249.000us
-00000003	flags[12] = 0	f r	D:00006788	wr-byte	00	\\thumble\Global\flags+0x0C	249.000us
-00000002	flags[12] = 0	f r	D:00006788	wr-byte	00	\\thumble\Global\flags+0x0C	249.000us
-00000001		f r	D:00006788	wr-byte	00	\\thumble\Global\flags+0x0C	249.000us
			BRK				187.760us

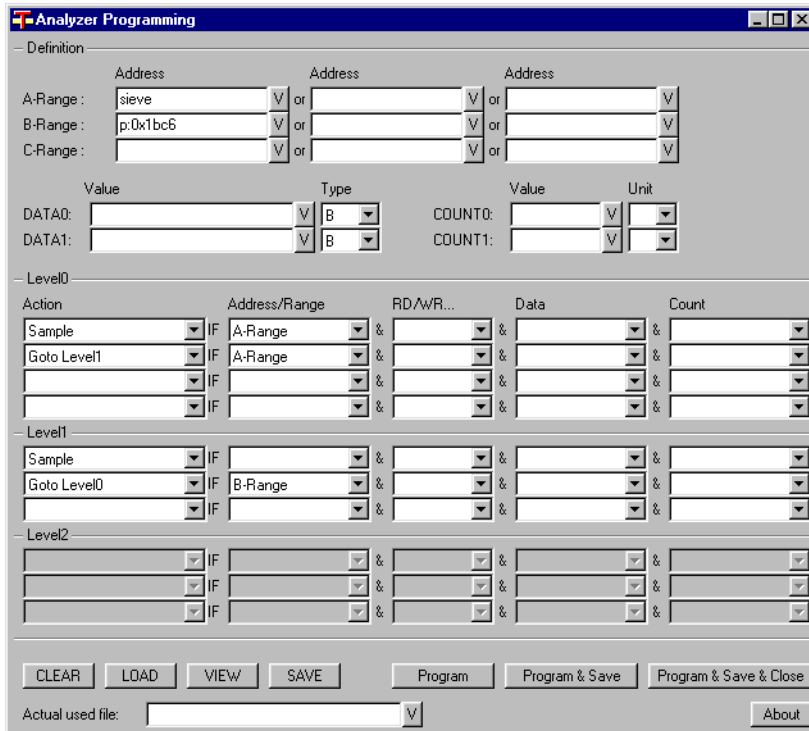
Sample a Function

1. Use A-Range to define the function entry.
2. Use B-Range to define the function exit.
3. Select a set of actions by using 2 trigger levels.

In the trigger level Level0 the analyzer waits for the function entry and changes to Level1 when the function is entered.

In the trigger level Level1 the analyzer samples all cycles and changes back to Level0 at the function exit.

Example 1: Sample only the function sieve.



Level0:

Sample only the function entry

Change to Level1 at the function entry

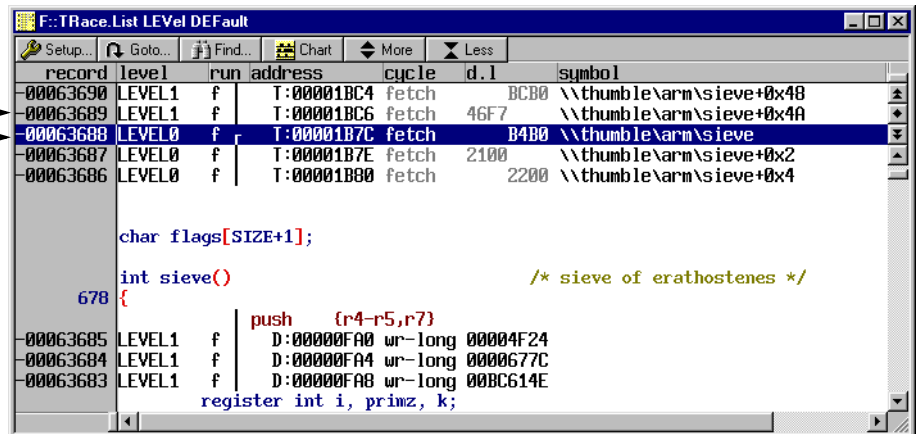
Level1:

Sample all cycles

Change to Level0 at the function exit

Function exit

Function entry



Example 2: Sample only the function sieve and mark the function entry with an A marker and the function exit with a B marker.

The Analyzer Programming dialog box is configured as follows:

- Definition:**
 - A-Range: sieve
 - B-Range: p:0x1bc6
- Level 0:**
 - Action: Sample (IF A-Range)
 - Action: Mark A (IF A-Range)
 - Action: Goto Level 1 (IF A-Range)
- Level 1:**
 - Action: Sample (IF B-Range)
 - Action: Mark B (IF B-Range)
 - Action: Goto Level 0 (IF B-Range)

Level0:

Sample only the function entry and mark it with an A marker

Change to Level1 at the function entry

Level1:

Sample all cycles

Change to Level0 at the function exit and mark the function exit by a B marker

The "Analyzer Find" dialog box allows to search for the markers

Function exit

Function entry

The Trace List window shows the following instructions:

record	mark	run	address	cycle	d.l	symbol	ti.back
00063694	---	f	T:00001BC4	fetch	BCB0	\\thumble\arm\sieve+0x48	0.260us
00063693	-B-	f	T:00001BC6	fetch	46F7	\\thumble\arm\sieve+0x4A	0.240us
00063692	A--	f	T:00001B7C	fetch	B4B0	\\thumble\arm\sieve	3.500us
00063691	---	f	T:00001B7E	fetch	2100	\\thumble\arm\sieve+0x2	0.260us
00063690	---	f	T:00001B80	fetch	2200	\\thumble\arm\sieve+0x4	0.240us

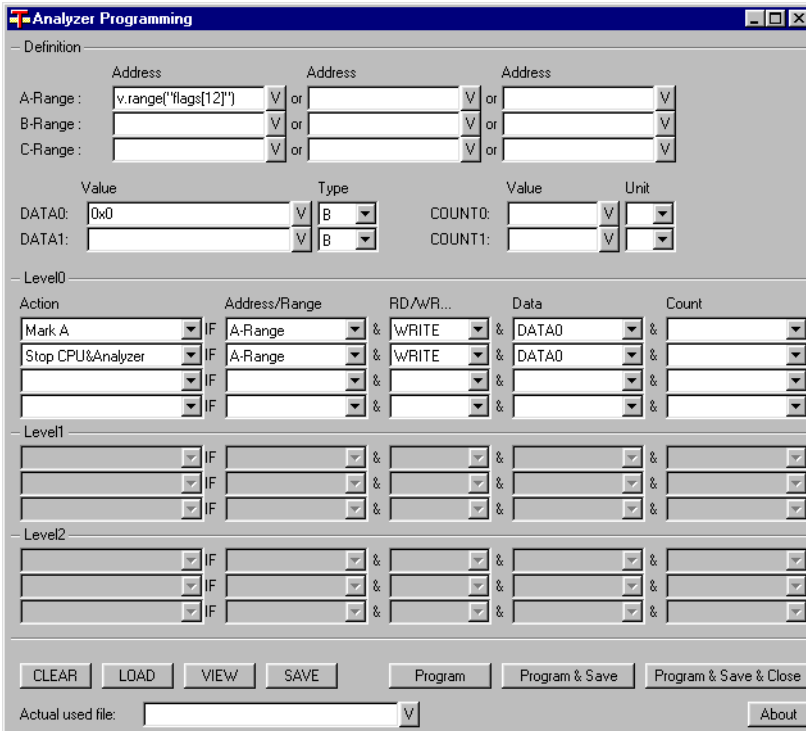
The Analyzer Find dialog box is configured with:

- Expert Find: mark.a on
- Address/Symbol: (empty)
- Data: (empty)
- Cycle: (empty)

After a Wrong Data Value is Written

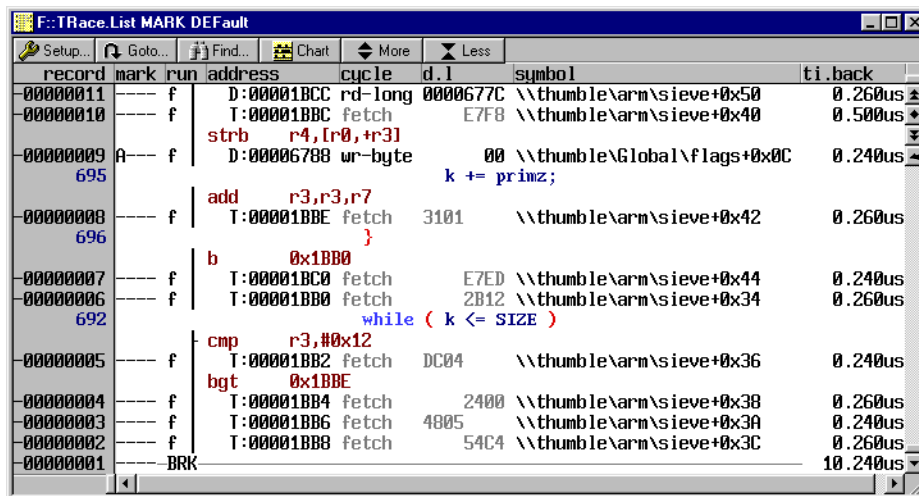
1. Use A-Range to define the memory location/variable.
2. Use DATA0 to define the data value.
To define a data value it is necessary to know if the data value is transferred as a byte, a word or a long.
3. Select the action Stop CPU&Analyzer and specify the condition by using the characteristic Address/Range and RD/WR... and DATA.

Example: Stop the program execution when 0 is written to flags[12]. Mark the record where the condition was true with an A marker.



Stop the program execution an the analyzer, when the data val defined by DATA0 is written to t address/range defined by A-Range.
Mark this cycle by an A marker

0 is written to flags[12] in the cycles marked by A. The CPU executes a few more cycles until the program execution stops



After n Accesses to a Memory Location/Variable

1. Use A-Range to define the memory location/variable.
2. Use COUNT0 to define a number counter, that counts the accesses to the memory location/variable.
3. Select a set of actions:

To increment the number counter COUNT0 on each access to memory location/variable defined by A-Range

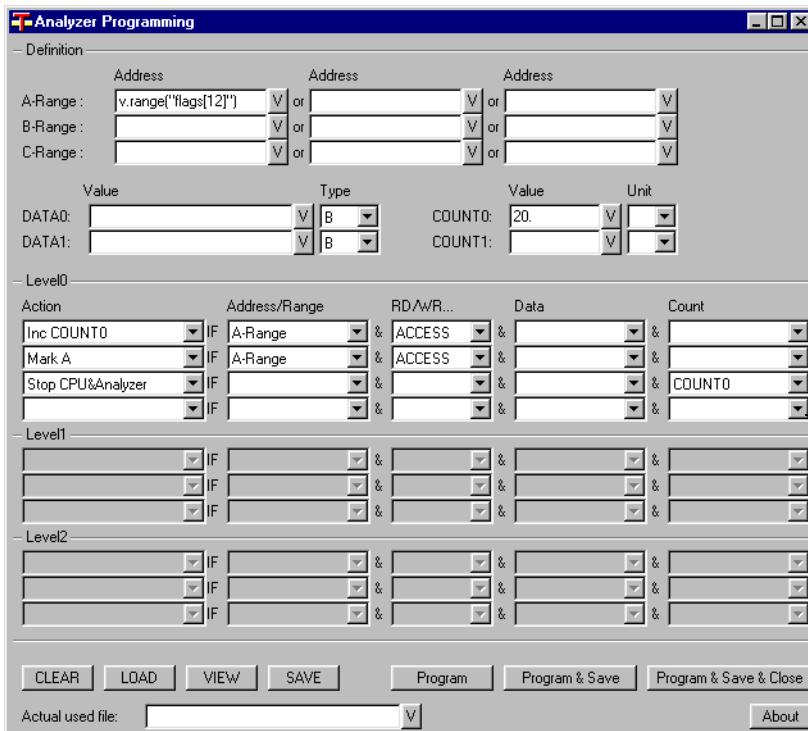
To stop the program execution when the number counter COUNT0 has reached its final value

Example: Stop the program execution after flags[12] was accessed 20 times. Mark each access by an A marker.

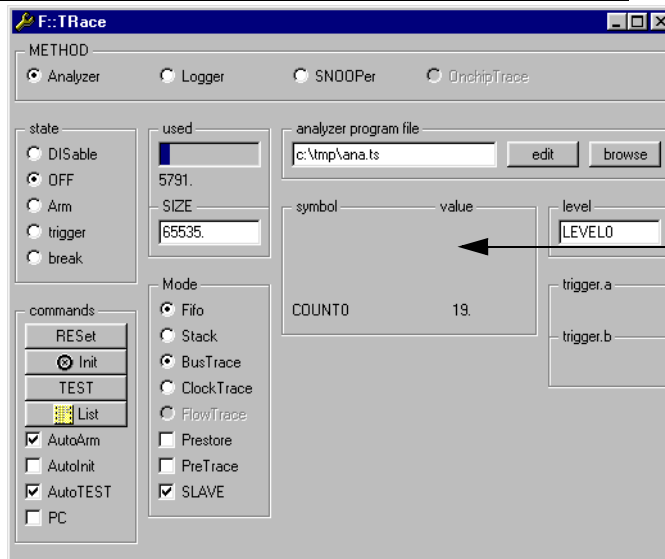
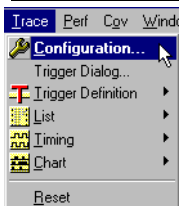
The screenshot displays the 'Analyzer Programming' window. In the 'Definition' section, the 'A-Range' is set to 'v.range("flags[12]"). Below this, the 'COUNT0' field is highlighted with an arrow and the text 'Click here to define a number counter'. The 'Level0' section contains a table for defining actions:

Action	Address/Range	RD/WR...	Data	Count
IF		&		
IF		&		
IF		&		
IF		&		

An inset window titled 'Define Counter' is open, showing the 'Number Counter' option selected. The 'Value' field is set to '20', with an arrow pointing to it and the text 'Define the value for the number counter'. The 'Unit' field is empty, and there are 'CLEAR' and 'OK' buttons at the bottom.



Increment the number counter COUNT0 on each access to the address/range defined by A-Range.
 Mark each access to the address/range marked by A-Range by an A marker.
 Stop the program execution when COUNT0 has reached its final value



The current state of the number counter can be watched in realtime in the Trace Configuration Window

F:\Trace-List MARK VAR Default

Setup... Goto... Find... Chart More Less

record	mark	var	run	address	cycle	d.l	symbol
-00000013	A	flags121 = 1	f	D:00006788	wr-byte	01	\\thumble\Global\Flags+0x0C
-00000012			f				eve+0x10
-00000011			f				eve+0x06
-00000010			f				eve+0x08
-00000009			f				eve+0x00
-00000008			f				eve+0x0C
-00000007			f				eve+0x18
-00000006			f				eve+0x10
-00000005			f				eve+0x1C
-00000004			f				eve+0x0C
-00000003			f				eve+0x0E
-00000002			f				eve+0x10

Analyzer Find

Expert Find

Mark A ON

Address/Symbol

HLL

Data

Cycle

Find Next Find First Find All Clear Cancel

F:\A.FINDALL Mark A ON

run	address	cycle	d.l	symbol	ti_back
-00005563	D:00006788	wr-byte	01	\\thumble\Global\Flags+0x0C	
-00005399	D:00006788	wr-byte	00	\\thumble\Global\Flags+0x0C	44.000us
-00005025	D:00006788	rd-byte	00	\\thumble\Global\Flags+0x0C	101.500us
-00004638	D:00006788	wr-byte	01	\\thumble\Global\Flags+0x0C	103.500us
-00004474	D:00006788	wr-byte	00	\\thumble\Global\Flags+0x0C	44.000us
-00004100	D:00006788	rd-byte	00	\\thumble\Global\Flags+0x0C	101.500us
-00003713	D:00006788	wr-byte	01	\\thumble\Global\Flags+0x0C	103.500us
-00003549	D:00006788	wr-byte	00	\\thumble\Global\Flags+0x0C	44.000us
-00003175	D:00006788	rd-byte	00	\\thumble\Global\Flags+0x0C	101.500us
-00002788	D:00006788	wr-byte	01	\\thumble\Global\Flags+0x0C	103.500us
-00002624	D:00006788	wr-byte	00	\\thumble\Global\Flags+0x0C	44.000us
-00002250	D:00006788	rd-byte	00	\\thumble\Global\Flags+0x0C	101.500us
-00001863	D:00006788	wr-byte	01	\\thumble\Global\Flags+0x0C	103.500us
-00001699	D:00006788	wr-byte	00	\\thumble\Global\Flags+0x0C	44.000us
-00001325	D:00006788	rd-byte	00	\\thumble\Global\Flags+0x0C	101.500us
-00000938	D:00006788	wr-byte	01	\\thumble\Global\Flags+0x0C	103.500us

After n Function Calls

1. Use A-Range to define the function entry.
2. Use COUNT0 to define a number counter, that counts each function entry.
3. Select a set of actions:

To increment the number counter COUNT0 on each function entry

To stop the program execution when the number counter COUNT0 has reached its final value

Example 1: Stop the program execution after the function sieve was entered the 15th time. Mark each function entry by an A marker.

The screenshot shows the 'Analyzer Programming' dialog box. In the 'Definition' section, the A-Range is set to 'sieve', B-Range and C-Range are empty. DATA0 and DATA1 are set to 'B'. COUNT0 is set to '15' and COUNT1 is empty. The 'Level0' section has three actions: 'Inc COUNT0' (IF A-Range, RD/WR, Data, Count), 'Mark A' (IF A-Range, RD/WR, Data, Count), and 'Stop CPU&Analyzer' (IF, RD/WR, Data, Count, COUNT0). The 'Level1' and 'Level2' sections are empty. Buttons at the bottom include CLEAR, LOAD, VIEW, SAVE, Program, Program & Save, Program & Save & Close, and About.

Increment the number counter COUNT0 on entry to the function defined by A-Range.

Mark each function entry by an A marker.

Stop the program execution when COUNT0 has reached its final value

F:\Trace.List MARK Default

Setup... Goto... Find... Chart More Less

record	mark	run	address	cycle	d.l	symbol	ti.back
00012040	---	f	T:00001B4B	fetch	2000	\\thumbnail\arm\main+0x150	0.240us
00012039	A---	f	T:00001B7C	fetch	B4B0	\\thumbnail\arm\sieve	0.260us
00012038	---	f	T:00001B7E	fetch	2100	\\thumbnail\arm\sieve+0x2	0.240us
00012037	---	f				sieve+0x4	0.260us

char fl: Mark.A ON Up Down

int sie: Address/Symbol HLL

678 {

00012036 --- f 0.240us

00012035 --- f 0.260us

00012034 --- f 0.240us

682

Analyzer Find

Find Next Find First Find All Clear Cancel

F::A.FINDALL, Mark.A ON

14	run	address	cycle	d.l	symbol	ti.back
00012039	f	T:00001B7C	fetch	B4B0	\\thumbnail\arm\sieve	
00011114	f	T:00001B7C	fetch	B4B0	\\thumbnail\arm\sieve	249.000us
00010189	f	T:00001B7C	fetch	B4B0	\\thumbnail\arm\sieve	249.000us
00009264	f	T:00001B7C	fetch	B4B0	\\thumbnail\arm\sieve	249.000us
00008339	f	T:00001B7C	fetch	B4B0	\\thumbnail\arm\sieve	249.000us
00007414	f	T:00001B7C	fetch	B4B0	\\thumbnail\arm\sieve	249.000us
00006489	f	T:00001B7C	fetch	B4B0	\\thumbnail\arm\sieve	249.000us
00005564	f	T:00001B7C	fetch	B4B0	\\thumbnail\arm\sieve	249.000us
00004639	f	T:00001B7C	fetch	B4B0	\\thumbnail\arm\sieve	249.000us
00003714	f	T:00001B7C	fetch	B4B0	\\thumbnail\arm\sieve	249.000us
00002789	f	T:00001B7C	fetch	B4B0	\\thumbnail\arm\sieve	249.000us
00001864	f	T:00001B7C	fetch	B4B0	\\thumbnail\arm\sieve	249.000us
00000939	f	T:00001B7C	fetch	B4B0	\\thumbnail\arm\sieve	249.000us
00000014	f	T:00001B7C	fetch	B4B0	\\thumbnail\arm\sieve	249.000us

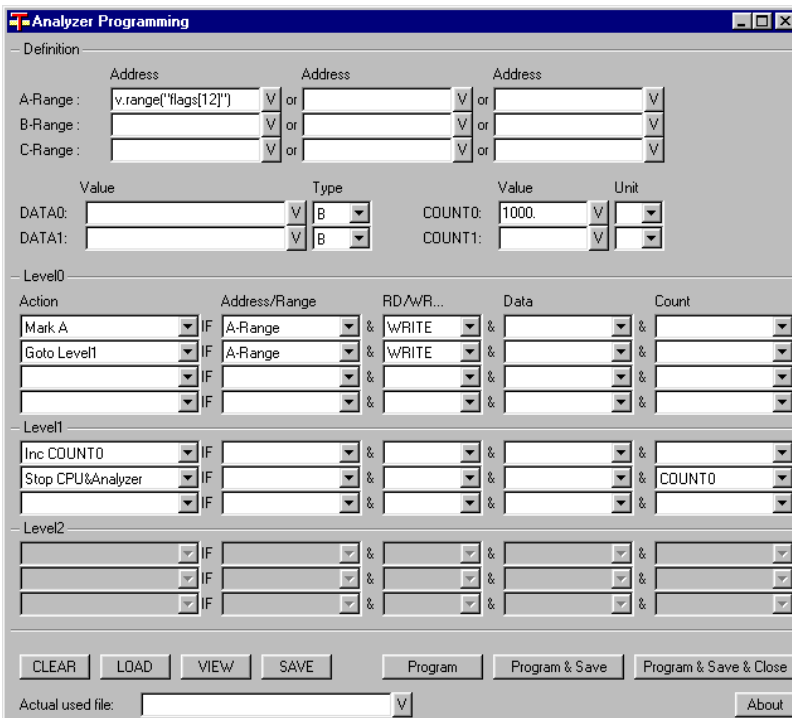
After Sampling n Cycles

1. Use A-Range to define the memory location/variable.
2. Use COUNT0 to define a number counter, that counts the number of records sampled to the trace buffer after the access to memory location/variable.
3. Select a set of actions by using 2 trigger levels.

In the trigger level Level0 the analyzer program waits for the access to memory location/variable and changes to Level1 after the access.

In the trigger level Level1 the analyzer program samples the defined number of CPU cycles and stops the program execution afterwards.

Example: Sample 1000. cycles after a write access to the variable flags[12].



Level 0:

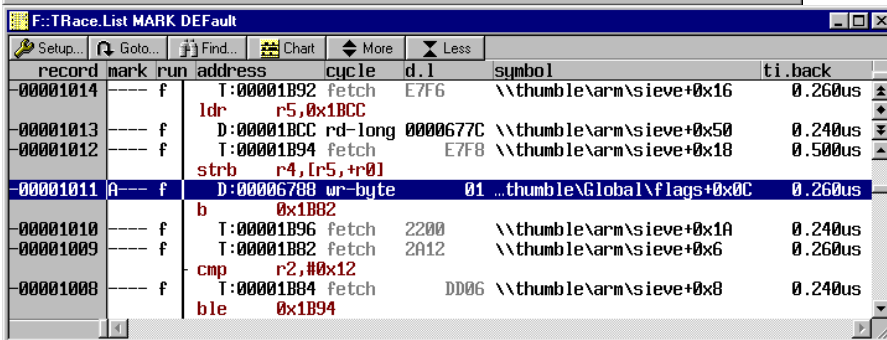
Mark the access to the memory location/variable with an A marker

Change to Level1 after the access to memory location/variable

Level 1:

Increment COUNT0 on each cycle

Stop the program execution, when COUNT0 has reached its final value



Sample another 1000 cycles after the write access to flags[12]

The statistic functions can be used to verify the runtime behavior of your embedded application.

The statistic functions are based on:

- Selective tracing
- Time stamp on each sampled record

The statistic functions have a high precision and provide detailed information, but since they are based on the records sampled in the trace buffer their observation time is limited.

Numeric Analysis

Perf Cgv Window Help

- Perf Configuration...
- Perf List
- Perf List Dynamic
- Perf Off

Function Runtime

- Prepare
- Distribution
 - Show Numerical
- Duration A to B
 - Show as Tree
- Distance trace records
 - Show Detailed Tree
- Reset
 - Show as Timing
 - Show Nesting

Start and stop the program

F::Data.List

Step Over Next Return Up Go Break Mode

```

addr/line source
675 for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ;
677 for ( i = 0 ; i <= SIZE ; i++ )
679 {
    if ( flags[ i ] )
    {

```

Perf Cgv Window Help

- Perf Configuration...
- Perf List
- Perf List Dynamic
- Perf Off

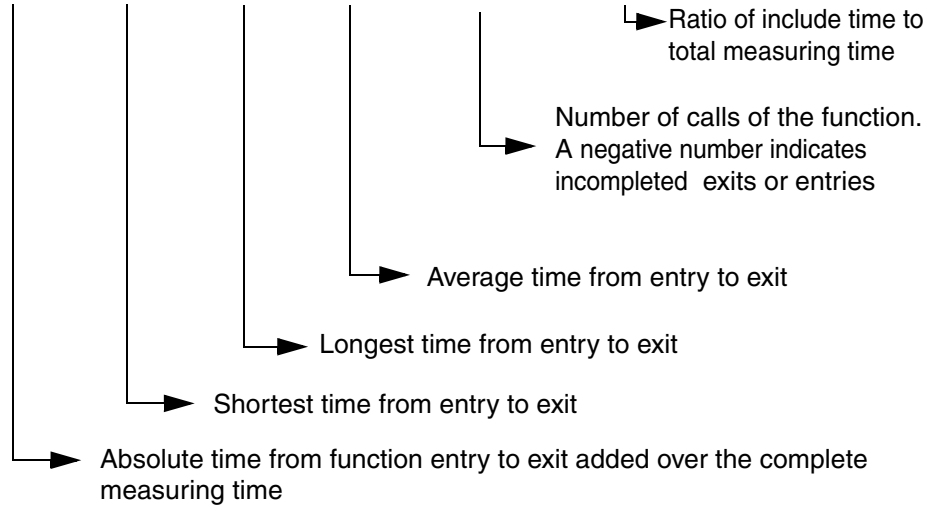
Function Runtime

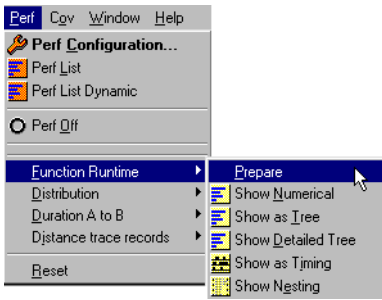
- Prepare
- Distribution
 - Show Numerical
- Duration A to B
 - Show as Tree
- Distance trace records
 - Show Detailed Tree
- Reset
 - Show as Timing
 - Show Nesting

Absolute measuring time

F::TRace.STATistic.FUNC

range	total time	min	max	avr	count	ratio 1%	2%	5%	10%
(root)	34.591ms	0.000	34.591ms	34.591ms	1. (-2)	0.000%			
rh8s\iarh8\main+0x2	34.591ms	0.000	34.591ms	34.591ms	1. (-1)	9.705%			
rh8s\iarh8\func2+0x2	76.600us	76.600us	76.600us	76.600us	1.	0.208%			
rh8s\iarh8\func1+0x2	10.500us	1.500us	1.500us	1.500us	7.	0.030%			
rh8s\iarh8\func2a+0x2	20.400us	20.400us	20.400us	20.400us	1.	0.058%			
rh8s\iarh8\func2b+0x2	141.100us	141.100us	141.100us	141.100us	1.	0.407%			
rh8s\iarh8\func2c+0x2	713.500us	713.500us	713.500us	713.500us	1.	2.062%			
rh8s\iarh8\func2d+0x2	48.400us	48.400us	48.400us	48.400us	1.	0.139%			
rh8s\iarh8\func4+0x2	36.600us	36.600us	36.600us	36.600us	1.	0.105%			





← What does *Function Runtime Prepare* ?

1. All function entries are marked with an Alpha Breakpoint, all function exits are marked with a Beta Breakpoint.

The breakpoints can be set automatically, if HLL-functions are loaded.

Break.SetFunc [<range>|<module>] [/<options>] Mark HLL functions

address	types	state	
C:010438	A	HARD	\\iarh8s\\iarh8\\func1+0x2
C:01043E	B	HARD	\\iarh8s\\iarh8\\func1\4
C:010442	A	HARD	\\iarh8s\\iarh8\\func2+0x2
C:0104B2	B	HARD	\\iarh8s\\iarh8\\func2\22
C:0104BA	A	HARD	\\iarh8s\\iarh8\\func2a+0x2
C:0104DE	B	HARD	\\iarh8s\\iarh8\\func2a\10
C:0104E2	A	HARD	\\iarh8s\\iarh8\\func2b+0x2
C:010520	B	HARD	\\iarh8s\\iarh8\\func2b\10
C:010528	A	HARD	\\iarh8s\\iarh8\\func2c+0x2
C:010586	B	HARD	\\iarh8s\\iarh8\\func2c\10

By default Alpha Breakpoints are set to *function entry + prefetch offset*, to avoid that accesses to function entries caused by prefetches are sampled.



2. A trigger program is loaded, that samples only the function entries and exits. The function entries are marked with marker A and the function exits are marked with marker B.

```
Sample.enable IF AlphaBreak||BetaBreak
```

```
Mark.A IF AlphaBreak
```

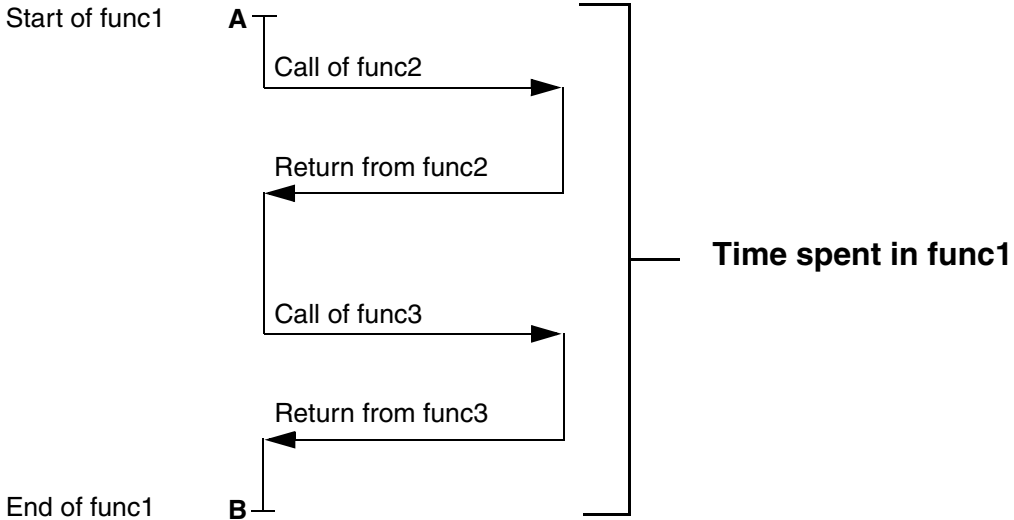
```
Mark.B IF BetaBreak
```

record	mark	run	address	cycle	d.w	symbol	ti.back
256			return a+b*c;				
			pop.w r0				
+00000064	A---	f r	P:010622	fetch	6DF0	\\iarh8s\\iarh8\\func6+0x2	16.400us
+00000065	-B--	f r	P:010666	fetch	0100	\\iarh8s\\iarh8\\func6+0x46	63.000us
+00000066	A---	f r	P:01066E	fetch	6DF0	\\iarh8s\\iarh8\\func7+0x2	17.200us
+00000067	-B--	f r	P:0106B2	fetch	0100	\\iarh8s\\iarh8\\func7+0x46	63.000us
280			return a*b;				
			undef 0x100				
+00000068	A---	f r	P:0106BA	fetch	0020	\\iarh8s\\iarh8\\func8+0x2	8.300us
+00000069	-B--	f r	P:01082A	fetch	5470	\\iarh8s\\iarh8\\func8+0x172	70.000us
+00000070	A---	f r	P:01082E	fetch	6DF1	\\iarh8s\\iarh8\\func9+0x2	6.700us
+00000071	A---	f r	P:010438	fetch	7915	\\iarh8s\\iarh8\\func1+0x2	16.700us
+00000072	-B--	f r	P:01043E	fetch	5470	\\iarh8s\\iarh8\\func1+0x8	1.500us
+00000073	A---	f r	P:010438	fetch	7915	\\iarh8s\\iarh8\\func1+0x2	6.800us
+00000074	-B--	f r	P:01043E	fetch	5470	\\iarh8s\\iarh8\\func1+0x8	1.500us
+00000075	A---	f r	P:010438	fetch	7915	\\iarh8s\\iarh8\\func1+0x2	7.000us
+00000076	-B--	f r	P:01043E	fetch	5470	\\iarh8s\\iarh8\\func1+0x8	1.500us

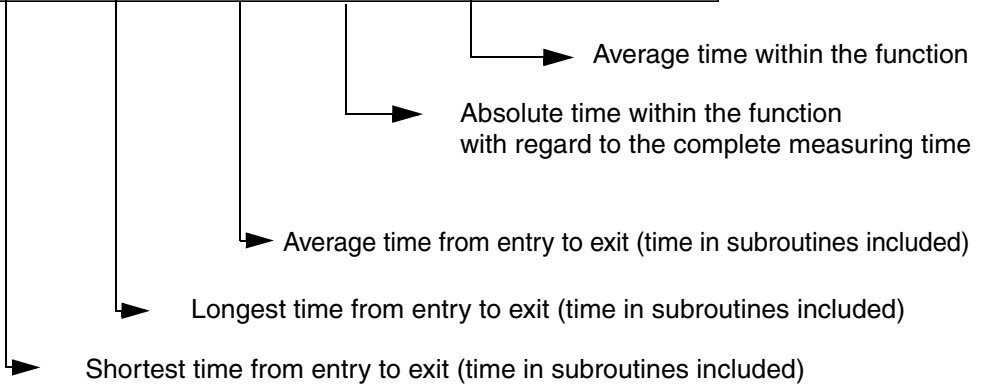


- Initialize the trace buffer before you start the measurement.

Subfunctions

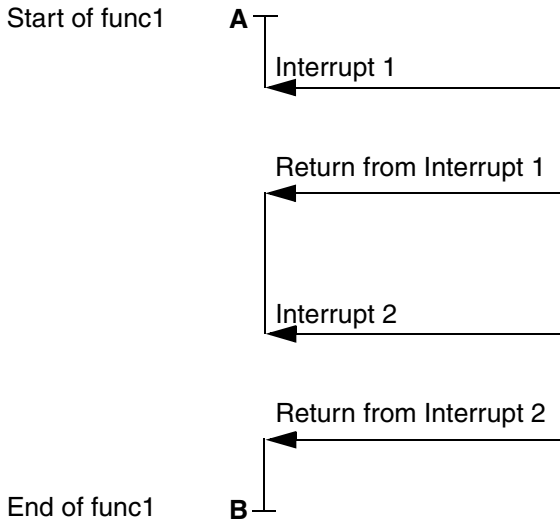


F:\TRace.STATistic.FUNC MIN MAX AVERAGE INTERNAL I AVERAGE COUNT							
total: 34.591ms							
range	min	max	avr	internal	iavr	count	
(root)	0.000	34.591ms	34.591ms	0.000	0.000	1. (-2)	
h8s\iarh8\main+0x2	0.000	34.591ms	34.591ms	3.357ms	3.357ms	1. (-1)	
h8s\iarh8\func2+0x2	76.600us	76.600us	76.600us	72.100us	72.100us	1.	
h8s\iarh8\func1+0x2	1.500us	1.500us	1.500us	10.500us	1.500us	7.	
h8s\iarh8\func2a+0x2	20.400us	20.400us	20.400us	20.400us	20.400us	1.	
h8s\iarh8\func2b+0x2	141.100us	141.100us	141.100us	141.100us	141.100us	1.	
h8s\iarh8\func2c+0x2	713.500us	713.500us	713.500us	713.500us	713.500us	1.	
h8s\iarh8\func2d+0x2	48.400us	48.400us	48.400us	48.400us	48.400us	1.	
h8s\iarh8\func4+0x2	36.600us	36.600us	36.600us	36.600us	36.600us	1.	
h8s\iarh8\func3+0x2	0.100us	0.100us	0.100us	0.100us	0.100us	1.	
h8s\iarh8\func5+0x2	7.100us	7.100us	7.100us	7.100us	7.100us	1.	



F:\Trace.STATistic.FUNC MIN MAX AVERAGE EXTERNAL EAVERAGE COUNT						
total: 34.591ms						
range	min	max	avr	external	eavr	count
(root)	0.000	34.591ms	34.591ms	34.591ms	34.591ms	1. (-2)
rh8s\iarh8\main+0x2	0.000	34.591ms	34.591ms	31.234ms	31.234ms	1. (-1)
h8s\iarh8\func2+0x2	76.600us	76.600us	76.600us	4.500us	4.500us	1.
h8s\iarh8\func1+0x2	1.500us	1.500us	1.500us	0.000	0.000	7.
8s\iarh8\func2a+0x2	20.400us	20.400us	20.400us	0.000	0.000	1.
8s\iarh8\func2b+0x2	141.100us	141.100us	141.100us	0.000	0.000	1.
8s\iarh8\func2c+0x2	713.500us	713.500us	713.500us	0.000	0.000	1.
8s\iarh8\func2d+0x2	48.400us	48.400us	48.400us	0.000	0.000	1.
h8s\iarh8\func4+0x2	36.600us	36.600us	36.600us	0.000	0.000	1.

Average time spent in subfuntions
 Absolute time spent in subfuntions with regard to the complete measuring time



The statistic functions exclude interrupts and double fetches caused by interrupts, if the interrupt routines are marked as follows:

- Interrupt entry is marked by Alpha Breakpoint and a Charly breakpoint, interrupt exit is marked by Beta Breakpoint.

Modified trigger program:

```
Sample.enable IF AlphaBreak|BetaBreak

Mark.A IF AlphaBreak
Mark.B IF BetaBreak
Mark.C IF CharlyBreak
```

-

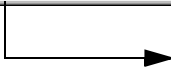
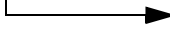
```
Break.SetFunc [<range>|<module>] /INTR Mark HLL functions
```

-

```
sYmbol.ForEach <cmd> [<name_pattern>] Symbol wildcard command
```

```
sYmbol.ForEach "Break.SetFunc * /INTR" INT*
```

F:\Trace.STATistic.FUNC MAXINTR DEFAULT						
total: 34.594ms intr: 10.500us						
range	maxintr	time	min	max	avr	count
(root)	10.500us	34.584ms	0.000	34.584ms	34.584ms	1.(-2)
rh8s\iarh8\main+0x2	10.500us	34.584ms	0.000	34.584ms	34.584ms	1.(-1)
h8s\iarh8\func2+0x2	4.500us	72.100us	72.100us	72.100us	72.100us	1.
h8s\iarh8\func1+0x2	0.000	10.500us	1.500us	1.500us	1.500us	7.
8s\iarh8\func2a+0x2	0.000	20.400us	20.400us	20.400us	20.400us	1.
8s\iarh8\func2b+0x2	0.000	141.100us	141.100us	141.100us	141.100us	1.
8s\iarh8\func2c+0x2	0.000	710.300us	710.300us	710.300us	710.300us	1.
8s\iarh8\func2d+0x2	0.000	48.400us	48.400us	48.400us	48.400us	1.
h8s\iarh8\func4+0x2	0.000	36.600us	36.600us	36.600us	36.600us	1.
h8s\iarh8\func3+0x2	0.000	0.100us	0.100us	0.100us	0.100us	1.
h8s\iarh8\func5+0x2	0.000	7.100us	7.100us	7.100us	7.100us	1.
h8s\iarh8\func6+0x2	0.000	63.000us	63.000us	63.000us	63.000us	1.
h8s\iarh8\func7+0x2	0.000	63.000us	63.000us	63.000us	63.000us	1.
h8s\iarh8\func8+0x2	0.000	70.000us	70.000us	70.000us	70.000us	1.
h8s\iarh8\func9+0x2	6.000us	43.300us	43.300us	43.300us	43.300us	1.
8s\iarh8\func10+0x2	0.000	105.800us	105.800us	105.800us	105.800us	1.
8s\iarh8\func11+0x2	0.000	1.900us	1.900us	1.900us	1.900us	1.

 Absolute time from function entry to exit added for the complete measuring time minus time spent in interrupts
 Maximum time, the function was interrupted.

`<trace>.STATistic.Func` [%<format>][<item> ...] [/<options>] Function runtime analysis

Statistic Distribution of a Single Event

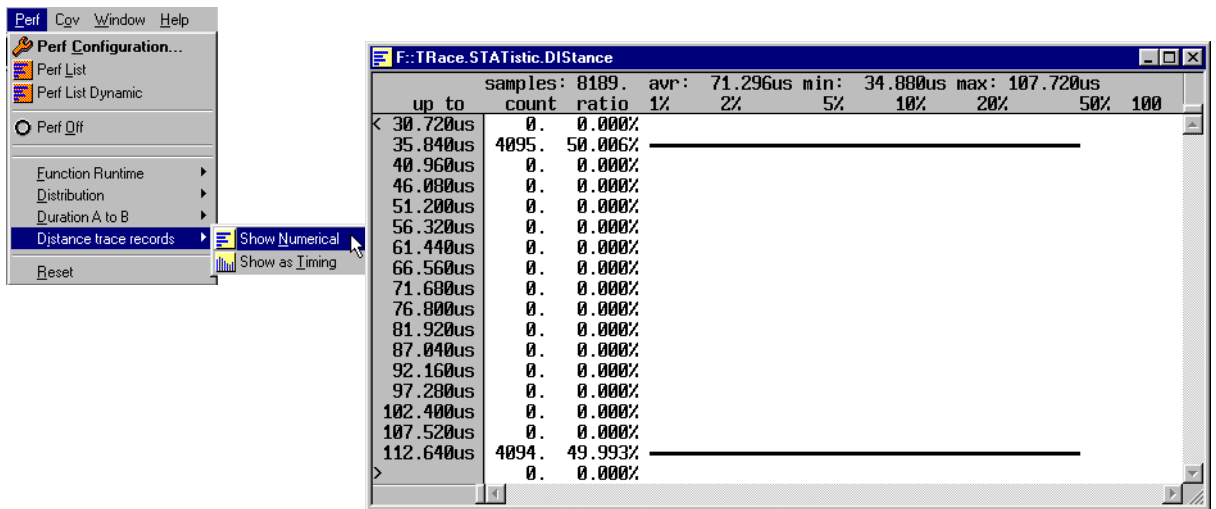
The statistic distribution of the time distance between the analyzer records is analyzed.

- Make a selective trace of a specific event
- Analyse the time distance

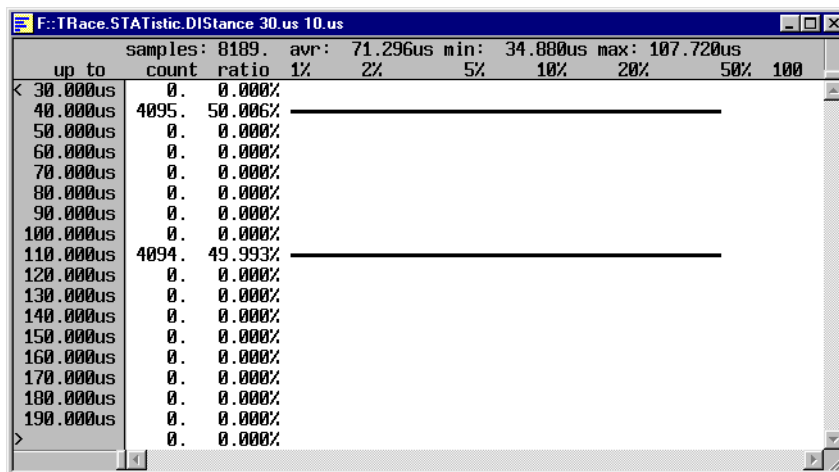
Example:

Analyse the statistic distribution of write accesses to flags[3].

```
ADDR AlphaBreak V.RANGE(flags[3])  
  
Sample.enable IF AlphaBreak&&Write
```



TRace.STATistic.DISTance 30.us 10.us



<trace>.STATistic.DISTance [**<timemin>**] [**<inc>**]

Time interval for a single event

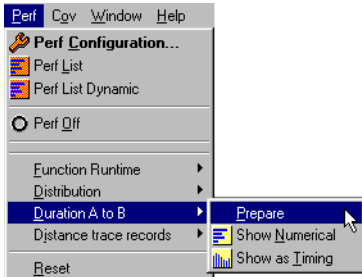
Statistic Distribution between 2 Events

The statistic distribution of the time distance between the record marked with A and the record marked with B is analyzed.

- Set an AlphaBreak to the first event
- Set a BetaBreak to the second event
- Prepare Analyzer Trigger Unit
- Analyse the time distance

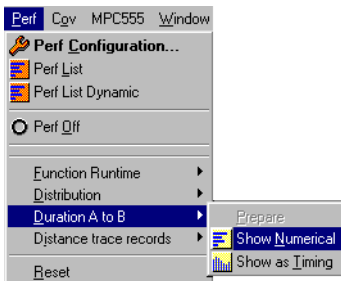
Example:

address	types	state	
C:010E4C	A	HARD	\\iarh8s\iarh8s\ieve
C:010EB2	B	HARD	\\iarh8s\iarh8s\ieve\24



Generates the following trigger program:

```
Sample.enable IF AlphaBreak||BetaBreak
Mark.A IF AlphaBreak
Mark.B IF BetaBreak
```



F::Trace.STATistic.DURation									
samples: 389. avr: 111.742us min: 111.740us max: 111.760us									
up to	count	ratio	1%	2%	5%	10%	20%	50%	100%
<111.740us	0.	0.000%							
111.743us	343.	88.174%							
111.745us	0.	0.000%							
111.748us	0.	0.000%							
111.750us	0.	0.000%							
111.753us	0.	0.000%							
111.755us	0.	0.000%							
111.758us	0.	0.000%							
111.760us	0.	0.000%							
111.763us	46.	11.825%							
111.765us	0.	0.000%							
111.768us	0.	0.000%							
111.770us	0.	0.000%							
111.773us	0.	0.000%							
111.775us	0.	0.000%							
111.778us	0.	0.000%							
111.780us	0.	0.000%							
>	0.	0.000%							

`<trace>.STATistic.DURation` [`<timemin>`] [`<inc>`]

Time between two events

For more complex events, write your own trigger program.

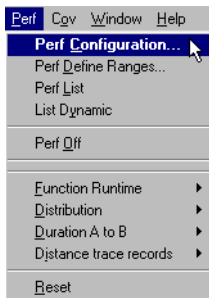
The Performance Analyzer

The performance analyzer displays the percentage of time spent in the different functions, modules etc.

Compared to the statistic function, the performance analyzer offers a long time measurement, but with less information.

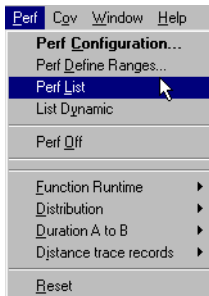
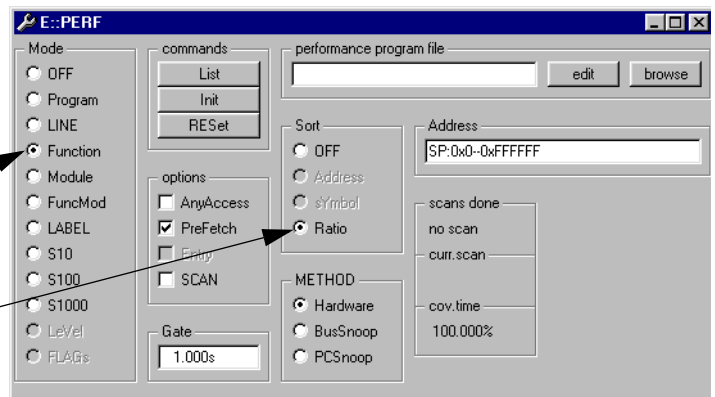
Function Performance Analysis

Example

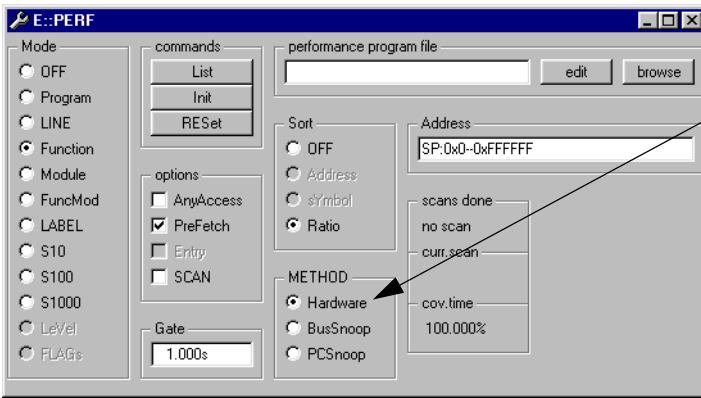


Measure the function performance

Display the result by ratio



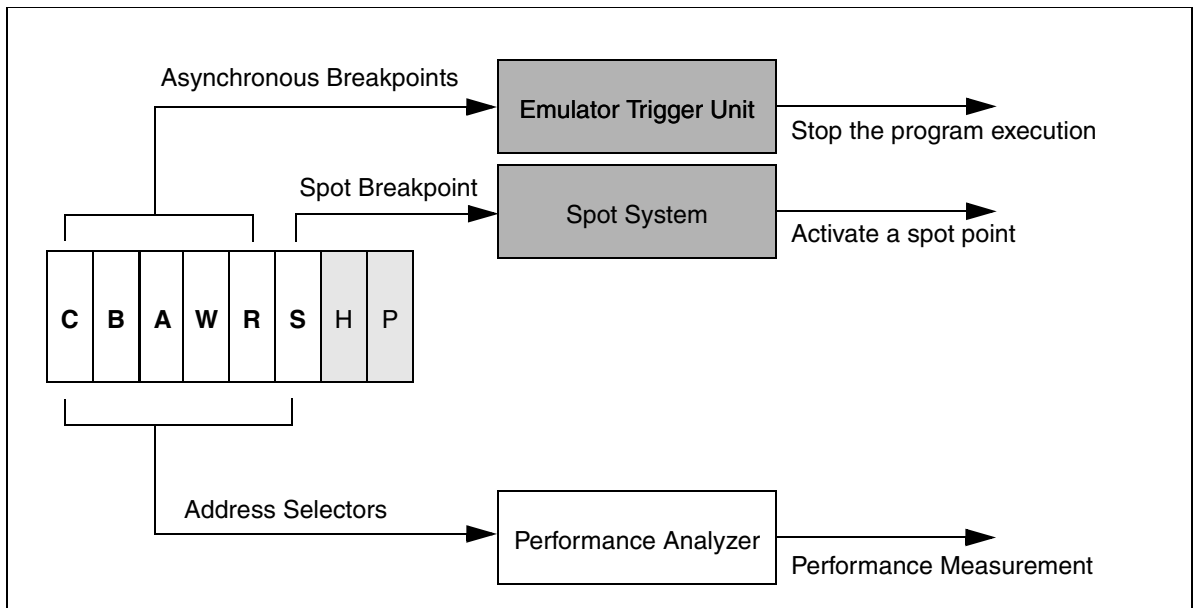
scan:	none	1	2	5	10	20	50	100
symbolName	ratio							
sieve	98.466%							
(other)	0.760%							
main	0.696%							
func10	0.025%							
func13a	0.012%							
func13	0.008%							
func8	0.008%							
func1g	0.006%							
func2	0.005%							
func4	0.003%							
func9	0.002%							
func7	0.002%							
func6	0.001%							
func11a	0.000%							
func12	0.000%							



Default measurement method for TRACE32-ICE is **Hardware**

If the measurement method is Hardware, the performance analyzer can be seen as an array of 64 counters. Only one counter is getting the clock to count, all other counters are stopped. The resolution of the counter is 1 μ s.

The counter selection is done by using the asynchronous breakpoints.



The asynchronous breakpoints are automatically disabled in the Emulator Main Trigger Unit and the Spot System is switched off as long as the Performance Analyzer is active.

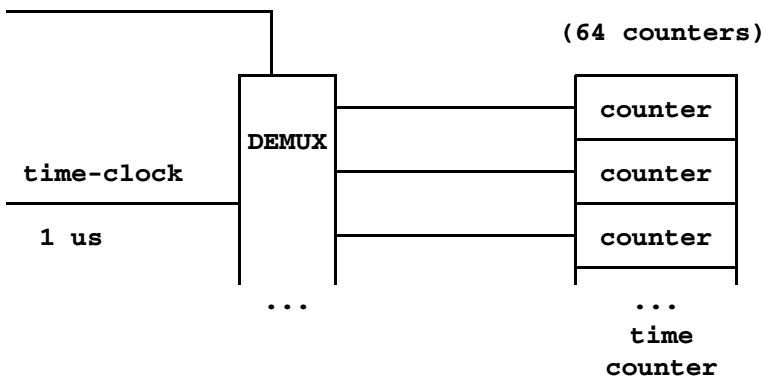
The breakpoints are automatically set to the functions, modules etc. if HLL information is loaded.

Address	C	
C:0000105A-0000105F	RS	\\mcc\mcc\func1+0x2
C:00001062-00001069	R	\\mcc\mcc\func1g+0x2
C:0000106C-000010CD	WR	\\mcc\mcc\func2+0x2
C:000010D0-000010D1	WRS	\\mcc\mcc\func3+0x2
C:000010D4-000010ED	W S	\\mcc\mcc\func4+0x2
C:000010F0-00001105	W	\\mcc\mcc\func5+0x2
C:00001108-0000114D	AW	\\mcc\mcc\func6+0x2
C:00001150-000011B1	AW S	\\mcc\mcc\func7+0x2
C:000011B4-0000132F	AWRS	\\mcc\mcc\func8+0x2
C:00001332-00001381	AWR	\\mcc\mcc\func9+0x2
C:00001384-00001517	A R	\\mcc\mcc\func10+0x2
C:0000151A-0000155D	A RS	\\mcc\mcc\func11+0x2
C:00001560-0000158F	A S	\\mcc\mcc\func11a+0x2
C:00001592-00001599	A	\\mcc\mcc\func12+0x2
C:0000159C-000015E3	BA	\\mcc\mcc\func13+0x2
C:000015E6-00001651	BA S	\\mcc\mcc\func13a+0x2
C:00001654-00001787	BA RS	\\mcc\mcc\main+0x2
C:0000178A-000017DB	BA R	\\mcc\mcc\sieve+0x2

With 6 address selectors up to 64 functions can be marked with different breakpoint combinations.

The functions are marked according the predefined prefetch offset (SETUP.PreFetch) if the PreFetch check box is switched on.

breakpoint combination



The measurement is made in intervals, after each interval the results in the display windows are refreshed. The default interval is 1 s.

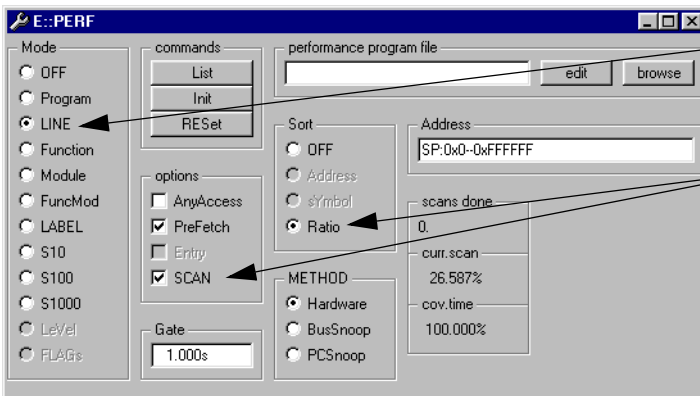
Performance Scanning

If the number of symbols exceed the number of counters, the performance analyzer can run in scan mode.

Performance Scanning:

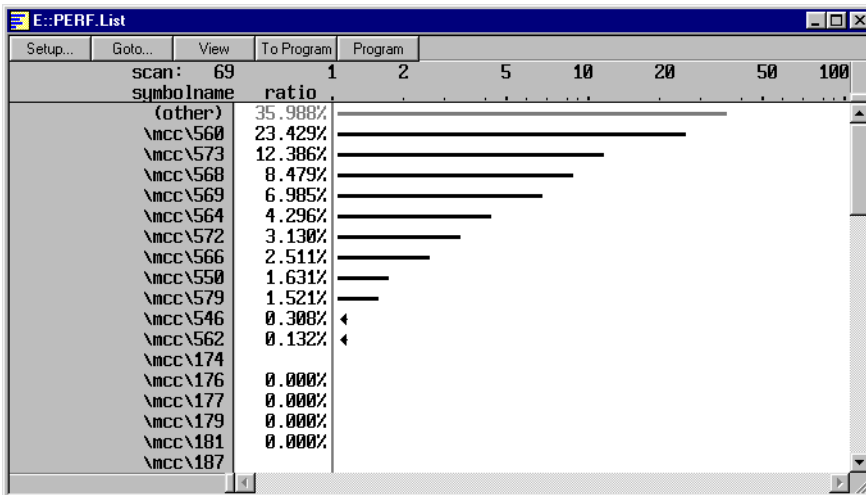
1. The first 64 areas are marked.
2. After each measurement interval the 32 less time consuming functions are thrown out of the measurement.
The next 32 areas are marked
3. The result of the performance scanning is a list of the 32 most time consuming areas.

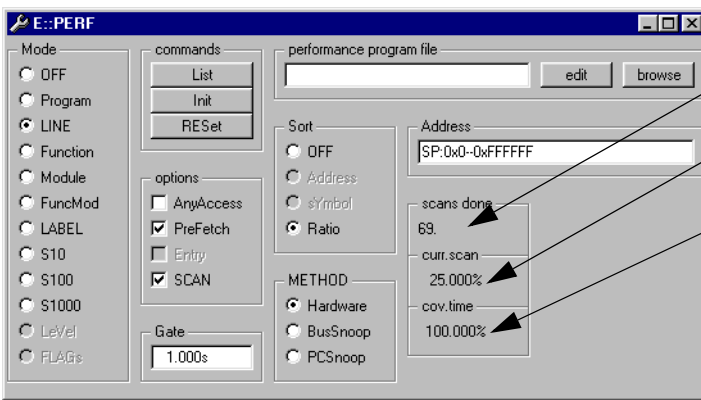
Example



Measure the hll lines

If SCAN and Ratio is on, the performance analyzer does a performance scanning





Number of complete scans

Percentage of the areas that were scanned in the current scan

Time covered by the currently measured 64 ranges

PERF.List ALL

symbolname	time	watchtime	ratio	dratio	address	break
(other)	97.333s	275.766s	35.295%	35.100%		
\mcc\560	64.692s	275.766s	23.459%	23.794%	SP:00001796--000017A5	C RS
\mcc\573	33.543s	275.766s	12.163%	12.154%	SP:000017CC--000017D3	C AWRS
\mcc\568	23.615s	275.766s	8.563%	8.362%	SP:000017B6--000017BD	C W S
\mcc\569	19.257s	275.766s	6.983%	6.942%	SP:000017C6--000017C9	C AW S
\mcc\564	11.846s	275.766s	4.295%	4.261%	SP:000017AC--000017AD	C WR
\mcc\572	8.642s	275.766s	3.133%	3.244%	SP:000017C2--000017C3	C AW
\mcc\566	6.926s	275.766s	2.511%	2.501%	SP:000017B0--000017B3	C WRS
\mcc\550	4.498s	275.766s	1.631%	1.644%	SP:0000178A--00001791	CB
\mcc\579	4.196s	275.766s	1.521%	1.527%	SP:000017D8--000017DB	C A R
\mcc\546	851.781ms	275.766s	0.308%	0.321%	SP:00001778--0000177B	CB RS
\mcc\562	365.549ms	275.766s	0.132%	0.143%	SP:000017A8--000017A9	C R
\mcc\491	0.000s	269.155s	0.000%	0.000%	SP:0000162E--00001631	A RS
\mcc\176	45.000us	275.766s	0.000%	0.000%	SP:00001096--000010AB	A S
\mcc\375		272.162s				

Ratio of time spent in this range in the last measurement interval (dynamic)

Ratio of time spent in this range

Time this range was watched

Time spent in the range



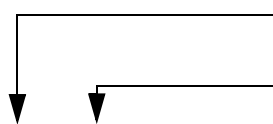
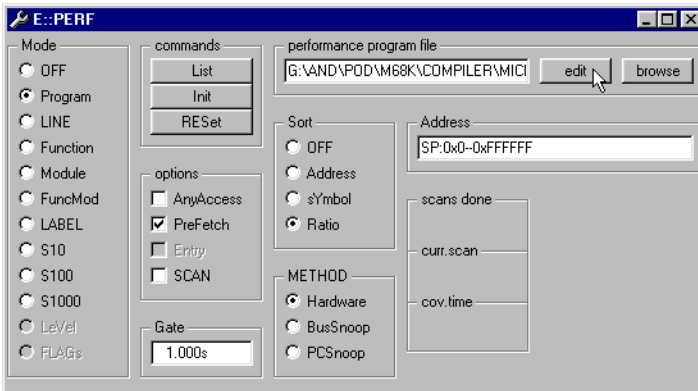
If a range is thrown out of the currently watched ranges, **not information about this range is stored.**

If the range reenters later the currently watched ranges, watchtime and time start at 0.

The ratio spent on this range depends on: time spent in the range and the time this range is watched. So if a very time consuming range is only watched for a short time, it can come up in the ratio list very quickly.

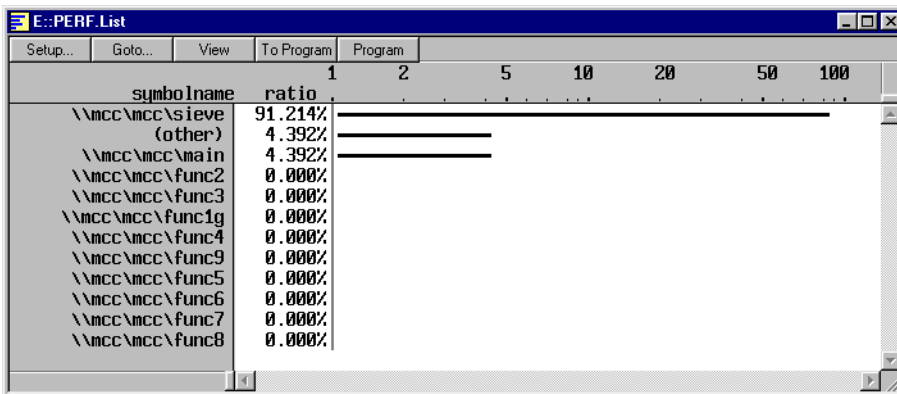
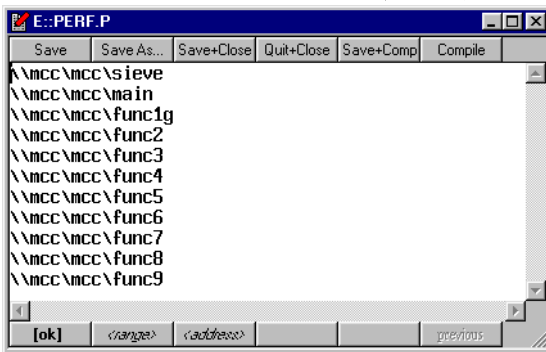
Performance Analyzer Programs

It is also possible to define specific address ranges for the performance analyzer by using the performance analyzer programming window.



Compile the performance analyzer program and load it into the performance analyzer and save

Compile the performance analyzer program and load it into the performance analyzer unit



Context Tracking System (CTS)

Context tracking is a technique that allows the context of the CPU/target to be reconstructed for a selected trace record based on the information sampled in the trace buffer. Context in this case means the contents of memory and registers.



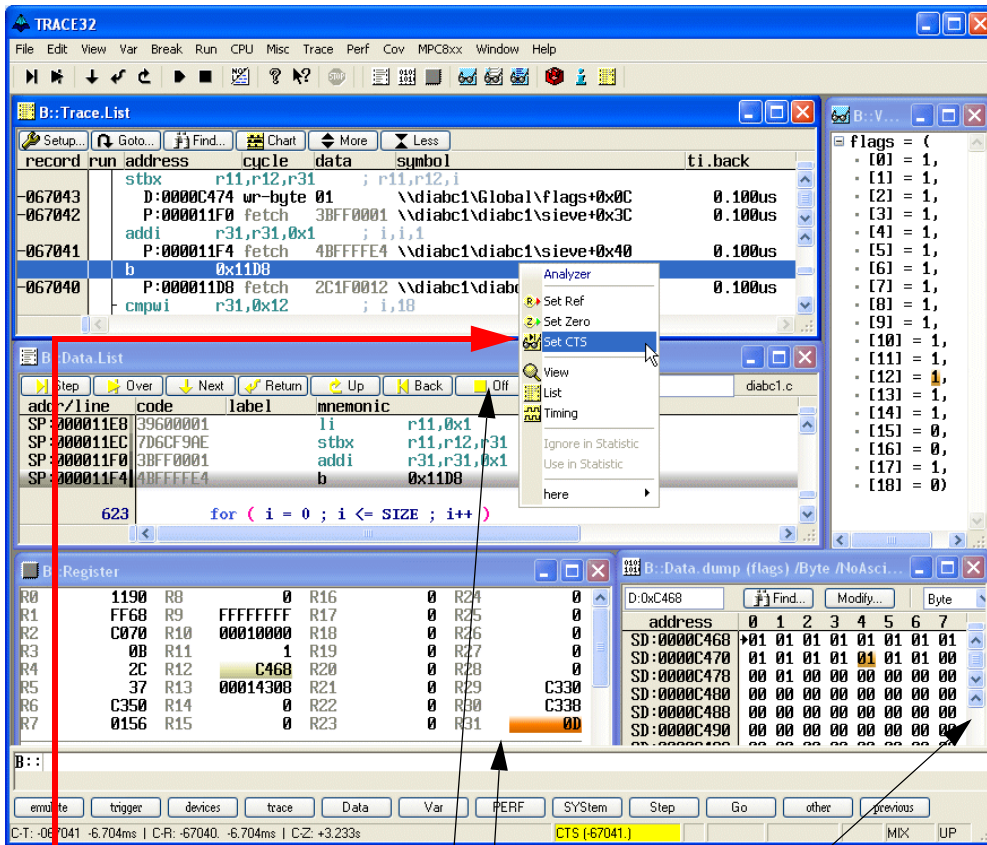
The basic features of the Context Tracking System as described in this section require that:

- full program and data information (all read accesses) is available
- all CPU cycles until the program execution stopped are sampled to the trace buffer

More features and settings for CTS are described with the **CTS** command group.

Trace Based Debugging

Trace based debugging allows to re-run the program and data flow sampled into the trace buffer on the TRACE32 screen. Trace based debugging is set-up as follows:

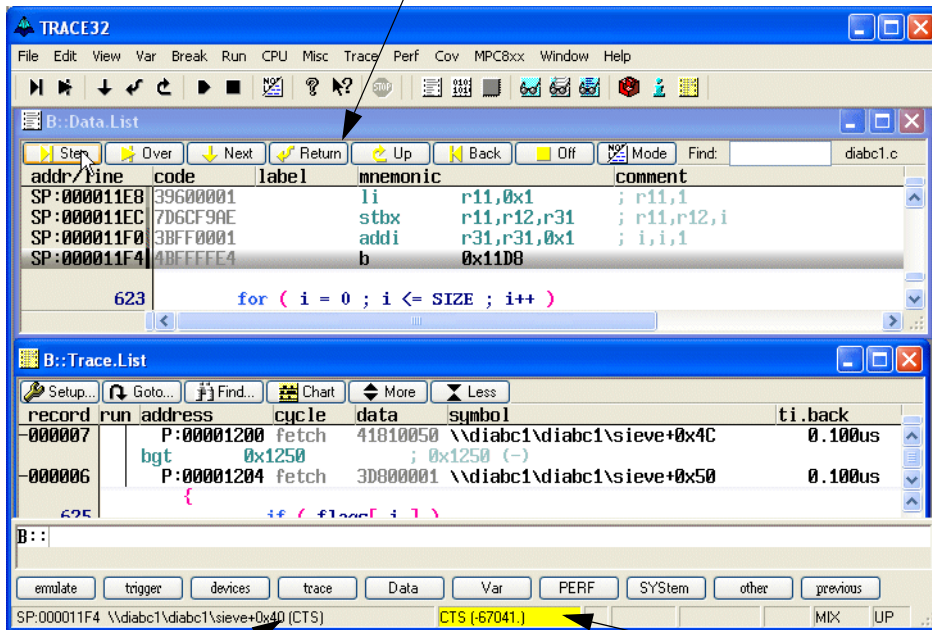


- 1.) Select the recording point for which the state of the CPU/target should be reconstructed.
- 2.) The PC will be set automatically to this recording point.
- 3.) During the trace-based debugging the changes of memories, variables and registers can be watched

With active CTS the windows in the TRACE32 screen don't show the current state of the CPU and the current contents of the memory and registers. The windows display the memory, registers etc. as they were at the moment, when the selected record was sampled.

To avoid irritations there are a number of indicators that show you that CTS is active.

Yellow buttons in the Data.List window indicate that CTS is active



A (CTS) after the logical and symbolical address in the state line shows you, that CTS is active

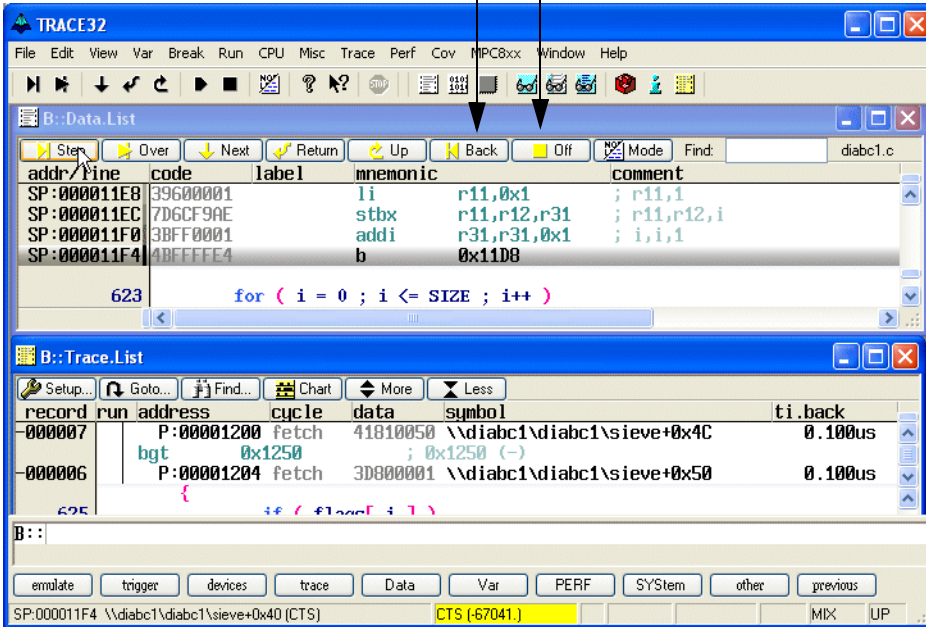
A yellow field in the state line shows you, that you see the memories and registers as they were when record number -xxxx. was sampled

All basic debugging commands can be executed: **Step**, **Step.Over**, **Go.Next**, **Go.Return**, **Step.Change**, **Var.Step.Change**, **Step.Till**, **Var.Step.Till**.

CTS also supports a **Step.Back** command.

With **OFF** CTS is switched off. The current state of the CPU and the current values of registers and variables are displayed on the TRACE32 screen.

Step.Back ————— **OFF** (Switch CTS OFF)



HLL Analysis of the Trace Contents

CTS provides also a number of features for hll trace display.

Analyse each HLL Step

The screenshot shows the CTS List window with a menu open over the 'List' option. The menu includes options like 'Default', 'All', 'HLL Source Only', 'MIX Source Only', 'HLL Source and Data', 'MIX Source and Data', 'HLL Source Data and Vars', 'Tracking with Source', and 'List Context: Tracking System'. Below the menu, the CTS List window displays a table of HLL steps with columns for record number, source code, and execution time. A second window, 'B::y.info i', shows register information for R31, including the address and value.

record	Source Code	Time
000053	anzahl = 6 flags = (1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0) i = 8 if (flags[i])	0.500us
625 000048	primz = 17 i = 8 primz = i + i + 3;	0.600us
627 000042	k = 24 i = 8 primz = 19 k = i + primz;	0.200us
628	k = 27	

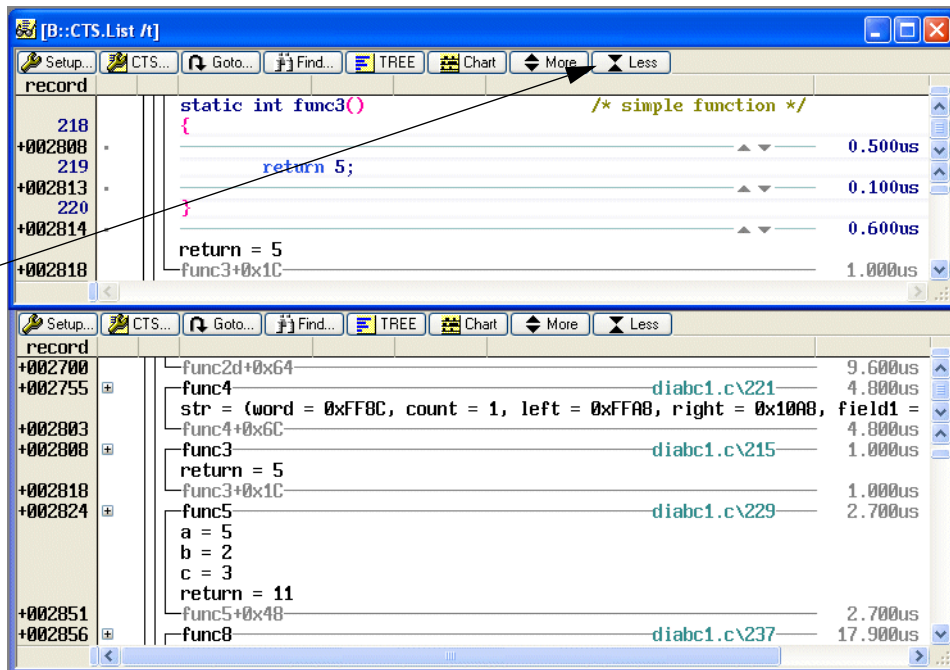
B::y.info i
\\diabc1\diabc1\sieve\i
R31 block-local register alive: P:0x11B8--0x1273
(register int) (signed 32 bits)

For each hll step the following information is displayed:

- The values of the local and global variables used in the hll step
- The result of the hll step
- The time needed for the hll step

CTS.List [<recordrange>] [<item> ...] [/option]

List pure hll trace.



Push the **Less** button to get a function nesting analysis

For each function you get the following information:

- Function parameters (and return value - not available on all CPUs)
- Time spent in the function

Tree Display

Click here to get a tree analysis of the function nesting

range	tree	time	min	max
(root)	(root)	13.107ms	0.000	13.107
\\diabc1\\diabc1\\main	└─ main	12.884ms	0.000	12.884
\\diabc1\\diabc1\\func2	└─ func2	16.300us	16.300us	16.300
\\diabc1\\diabc1\\func1	└─ func1	5.700us	1.900us	1.900
\\diabc1\\diabc1\\func2a	└─ func2a	9.100us	9.100us	9.100
\\diabc1\\diabc1\\func2b	└─ func2b	8.500us	8.500us	8.500
\\diabc1\\diabc1\\func2d	└─ func2d	9.600us	9.600us	9.600

Timing Display

Click here to get a graphical analysis of the function runtime

address	300.000us	310.000us	320.000us	330.000us
\\diabc1\\diabc1\\func2				
\\diabc1\\diabc1\\func1		█	█	
\\diabc1\\diabc1\\func2a				
\\diabc1\\diabc1\\func2b				
\\diabc1\\diabc1\\func2d				
\\diabc1\\diabc1\\func4				
\\diabc1\\diabc1\\func3				
\\diabc1\\diabc1\\func5				
\\diabc1\\diabc1\\func8	█			
\\diabc1\\diabc1\\func9		█		
\\diabc1\\diabc1\\func10			█	

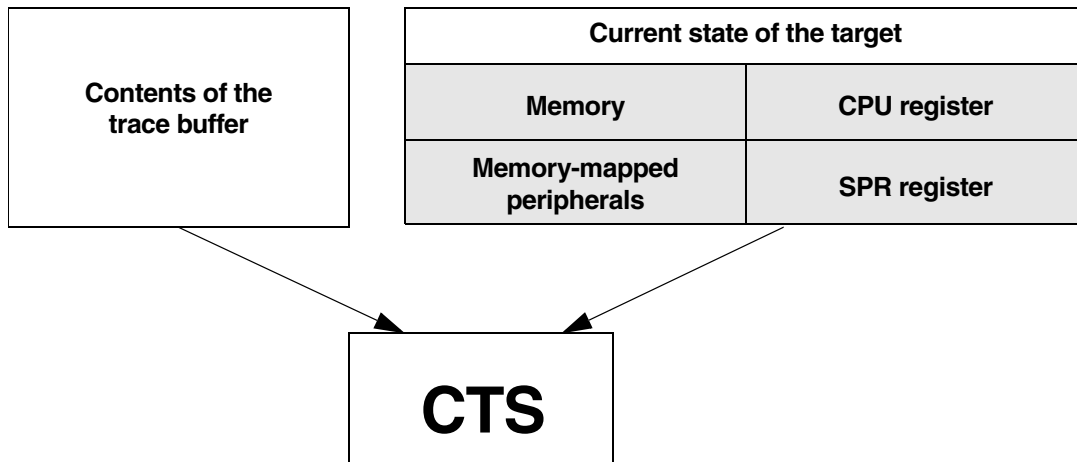
CTS.STATistic.TREE [%<format>][<item> ...] [</options>]

Display trace contents as call tree

CTS.Chart.sYmbol [<record_range>] [<scale>] [</option>]

Display trace contents as timing based on the symbol information

Background

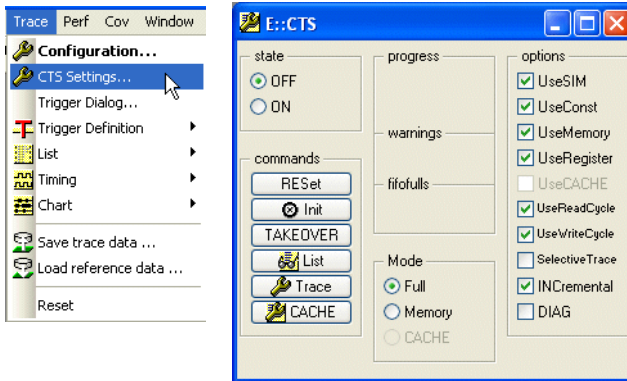


CTS read and evaluated the current state of the target.

1. CTS can only perform a correct context reconstruction, if solely the processor core, for which the data flow is sampled into the trace buffer, writes to memory. If there are memory addresses, e.g. dual-ported memories or peripherals, that are change otherwise, these addresses have to be excluded by the command **MAP.VOLATILE** from the CTS context reconstruction. These memory addresses are then displayed as unknown if CTS is used.
2. CTS performs memory reads while performing a context reconstruction. If read accesses to specific memory-mapped peripherals should be prevented, the addresses have to be excluded by the command **MAP.VOLATILE** from the CTS context reconstruction. These memory addresses are then displayed as unknown if CTS is used.
3. Under certain circumstances the reconstruction of the program flow can cause BUSERRORS on the target system. If this is the case, it is recommended to load the code to virtual memory.

4. CTS has to be re-configured if:
- the program execution is still running while CTS is used.
 - not all CPU cycles until the stop of the program execution are sampled to the trace.
 - the trace contents is reprocessed with a TRACE32 instruction set simulator.
 - only the program flow is sampled to the trace buffer.

In all these cases the current state of the target can not be used by CTS. For more information refer to the command **CTS.state**.



MAP.VOLATILE <range>	Exclude addresses from CTS
CTS.state	Reconfigure CTS.