# How to debug SW based on C166-OSE-CMN PST

# How to debug SW based on C166-OSE-CMN PST

- Introduction
- Debugging with trace tools (Mobile Analyser)
- Real time debugging
  - Types of exceptions and related breakpoints
  - Logging of the exceptions
  - Non-treatable crashes
- Tips on the usage of a RT debugger

# Introduction

- To debug embedded systems comprising CMN Protocol Stack and equipped with OSE operating systems on C166-based targets, several approaches can be followed.
- The first method is the inspection of standard and/or customized trace logs
- Alternatively, real time debugging shall be persued. This is useful and adviceble for the following types of bugs:
  - Crash and silent reset (their occurrence can be found in the trace logs)
  - Power off (silent connection loss)

The present document aims to suggest procedures and technique to debug the above-mentioned bugs. All other types of problems (e.g. analysis of function calls, system profiling, etc.) are out of the scope of the present document.

# Debugging with trace tools

- This approach helps particularly in pointing out logical faults and programming error that do not lead to a system exit.
- The MS's logs can be obtained by means of the Mobile Analyser trace tool. The tool collects (in .trx format, which can be converted to .txt) the following types of information:
  - SDL signals, showing the signals and the decoded parameters, the destination tasks and their current state;
  - Low Level Traces (LLT), representing the current values of L1 internal data structures (e.g. commands delivered to the DSP, data read out of the Shared Memory, the driver's events and processing, etc.). They are heavy to handle and shall be activated when necessary by selecting only the LLT groups and classes of interest  with the following command

```
at+xl1set="L<#group> <#class1> <#class2>...L"
```

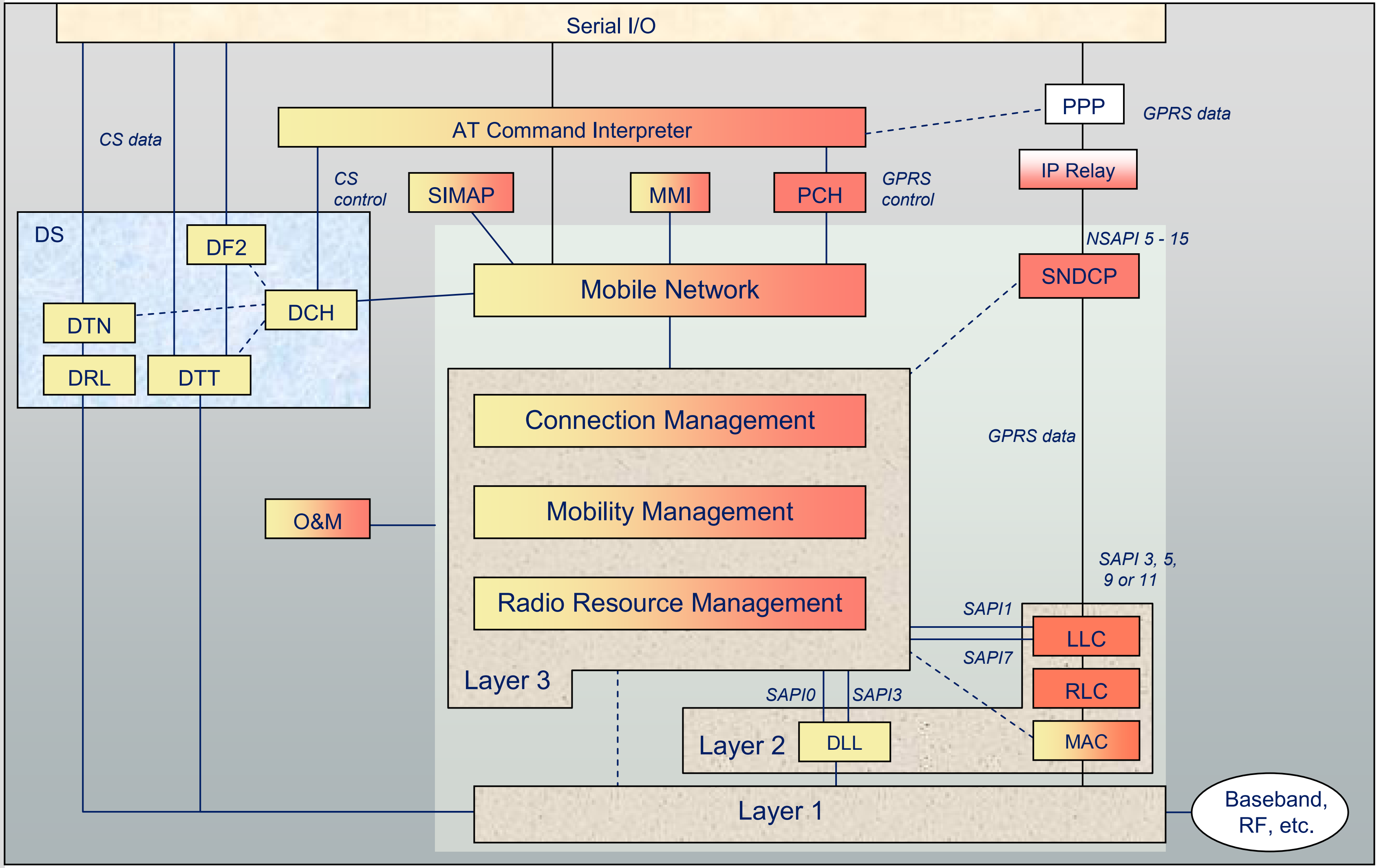
# Debugging with trace tools (2)

- ASCII strings, which are the result of the several “printf” instructions disseminated in the source code that can be added if required;
- Specific tasks’ debug signals: debug signals can be built by concatenating a pseudo task Id and an event number, thus creating a structure similar to SDL signals; by sending it via the following function, a header with additional trace information (i.e. fn) is appended:

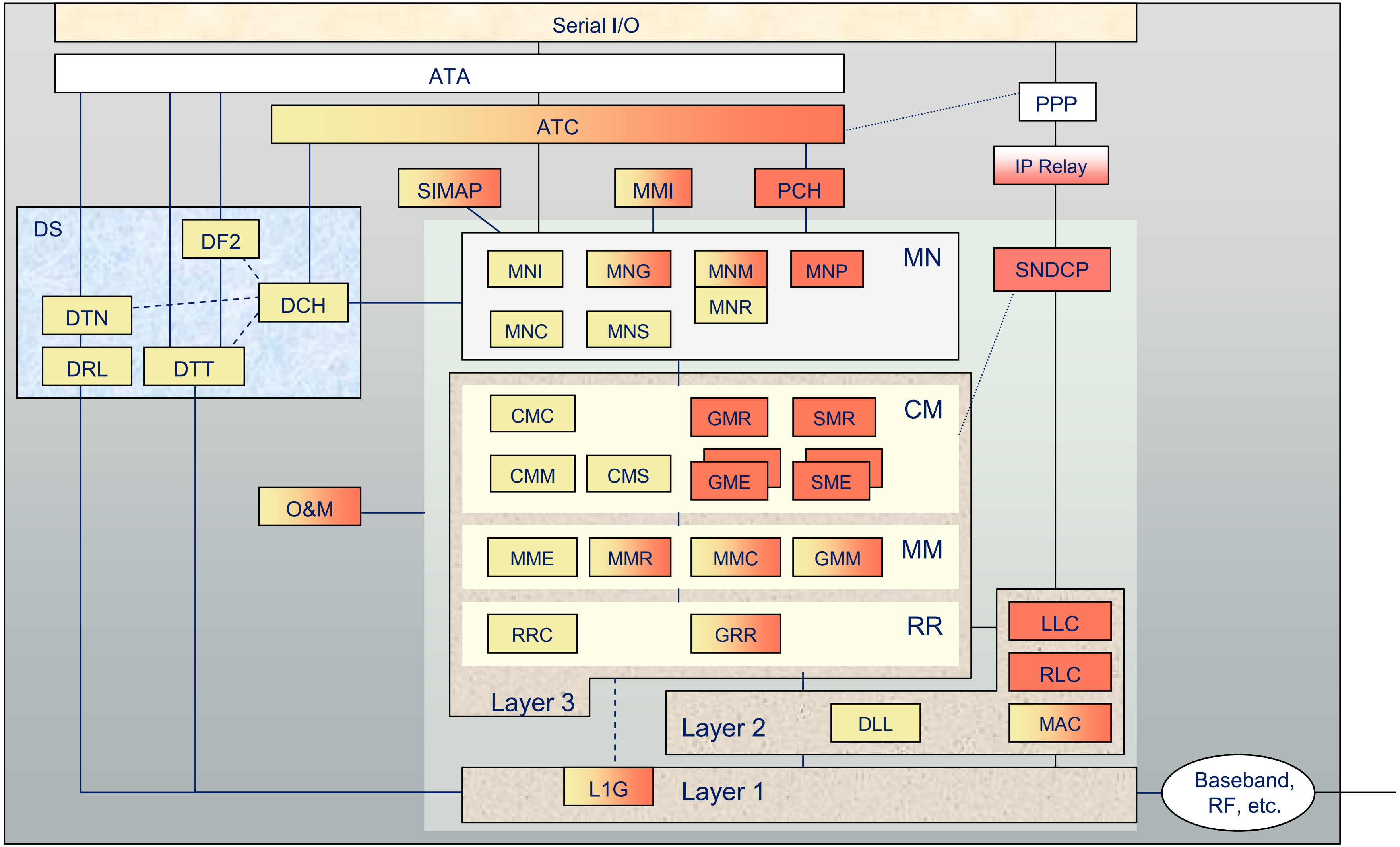
  ```
  hwtrc_task_info(1, P_mac_db, ((P_mac_db <<
  OS_SDL_SIGNAL_SHIFT)| debug_cause), NULL, 0);
  ```

  In this case no parameters is used (msg=NULL). If it is required to explore a data struct, a global variable with the appropriate data type shall be included in the project and a new Message Library shall be generated.

# System overview at module level



# System overview at process level



# List of processes

| | |
|---|---|
| ATA | ATC Adapter |
| ATC | AT Command Interpreter |
| DCH | Data Services Connection Handler |
| DTT | Data Services Terminal Adapter Transparent |
| DTN | Data Services Terminal Adapter Non-transparent |
| DRL | Data Services Radio Link Protocol |
| DF2 | Data Services FAX Adapter Class 2 |
| PCH | PDP Context Handler |
| PPP | Point-to-Point Protocol |
| IPR IP | Relay Function |
| SNDCP | Subnetwork Dependent Convergence Protocol |
| MMI | Man Machine Interface |
| SIMAP | SIM Application |
| O&M | Operation & Maintenance |
| MNI | Mobile Network Input Manager for CC, FDN, Emerg. Call |
| MNG | Mobile Network Registration Manager |
| MNM | Mobile Network SMS Manager |
| MNR | Mobile Network SMS Relay Function |
| MNP | Mobile Network PCH Server |
| MNC | Mobile Network Call Control |
| MNS | Mobile Network Supplementary Services |

# List of processes (2)

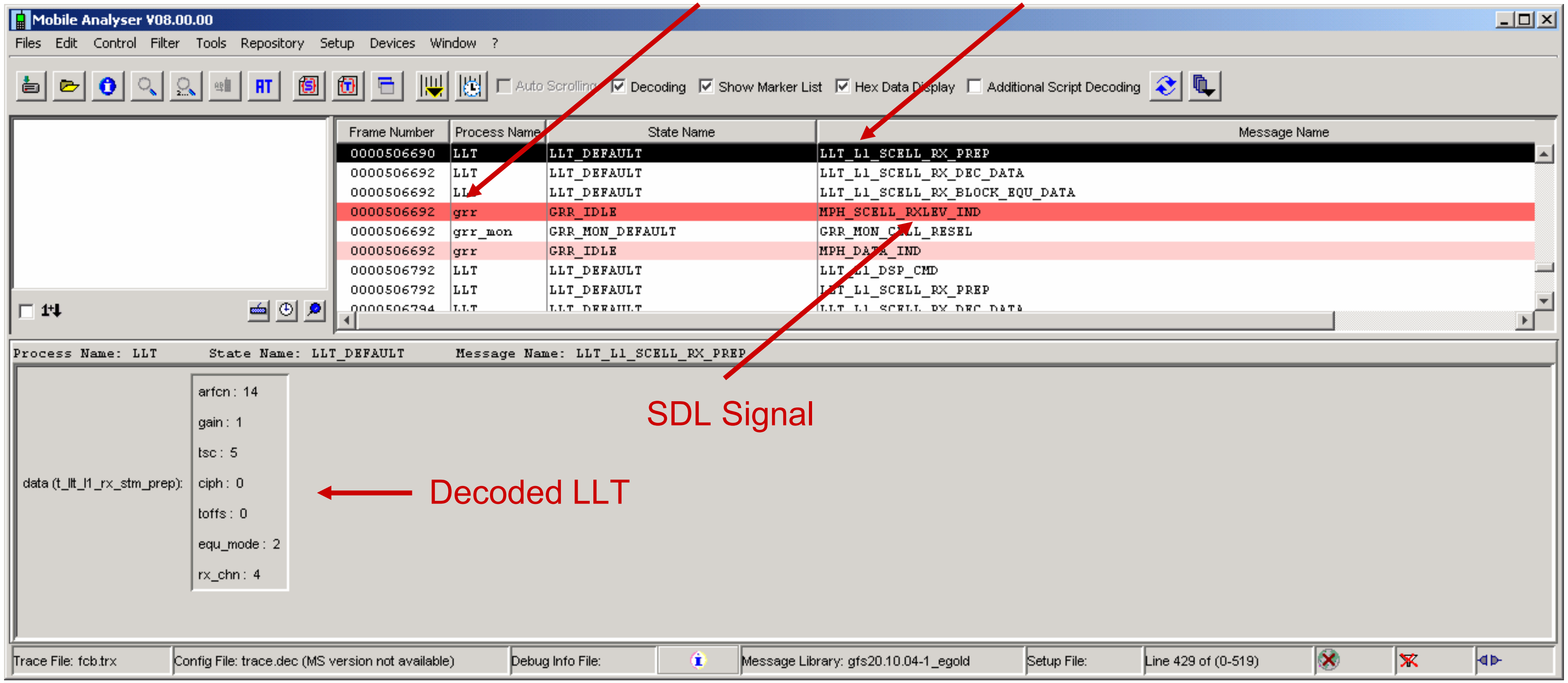
| | |
|---|---|
| CMC | Connection Management Call Control |
| CMM | Connection Management, Short Message Manager |
| CMS | Connection Management Call independent Suppl. Services |
| SMR | Connection Management, Session Manager Router |
| SME | Connection Management, Session Manager Entity |
| GMR | Connection Management GPRS SMS Router |
| GME | Connection Management GPRS SMS Entity |
| MME | Mobility Management: Location Registration (CS) and Idle Mode Handling (CS) |
| MMR | Mobility Management: Management of MM connection |
| MMC | Mobility Management: Coordination between MME and GMM, Common Functions |
| GMM | Mobility Management for GPRS |
| RRC | Radio Resource Management for Circuit Switched Mode |
| GRR | Radio Resource Management for IDLE Mode and GPRS |
| DLL | Data Link Layer |
| LLC | Logical Link Control |
| RLC | Radio Link Control |
| MAC | Medium Access Control |
| L1G | Layer 1 SDL Access level |

# Examples of trace logs

- SDL signals and LLT

Destination SDL Task

LLT info



Mobile Analyser V08.00.00
Files Edit Control Filter Tools Repository Setup Devices Window ?
□ Auto Scrolling
☑ Decoding
☑ Show Marker List
☑ Hex Data Display
□ Additional Script Decoding
□ 1↑↓

| Frame Number | Process Name | State Name | Message Name |
|---|---|---|---|
| 0000506690 | LLT | LLT_DEFAULT | LLT_L1_SCELL_RX_PREP |
| 0000506692 | LLT | LLT_DEFAULT | LLT_L1_SCELL_RX_DEC_DATA |
| 0000506692 | LLT | LLT_DEFAULT | LLT_L1_SCELL_RX_BLOCK_EQU_DATA |
| 0000506692 | grr | GRR_IDLE | MPH_SCELL_RXLEV_IND |
| 0000506692 | grr_mon | GRR_MON_DEFAULT | GRR_MON_CELL_RESEL |
| 0000506692 | grr | GRR_IDLE | MPH_DATA_IND |
| 0000506792 | LLT | LLT_DEFAULT | LLT_L1_DSP_CMD |
| 0000506792 | LLT | LLT_DEFAULT | LLT_L1_SCELL_RX_PREP |
| 0000506794 | LLT | LLT_DEFAULT | LLT_L1_SCELL_RX_DEC_DATA |

```
Process Name: LLT       State Name: LLT_DEFAULT       Message Name: LLT_L1_SCELL_RX_PREP
```

data (t_llt_l1_rx_stm_prep):
- arfcn : 14
- gain : 1
- tsc : 5
- ciph : 0
- toffs : 0
- equ_mode : 2
- rx_chn : 4

SDL Signal

Decoded LLT

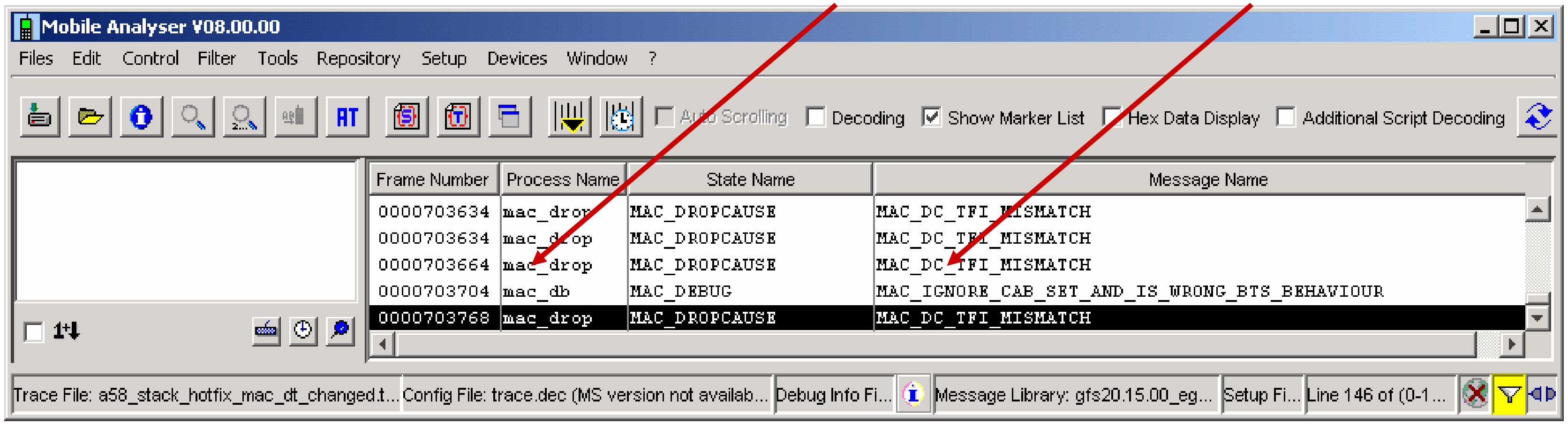
Trace File: fcb.trx | Config File: trace.dec (MS version not available) | Debug Info File: | Message Library: gfs20.10.04-1_egold | Setup File: | Line 429 of (0-519)

# Examples of trace logs (2)

- Printf and task debug signals

Task debug sub_Pid

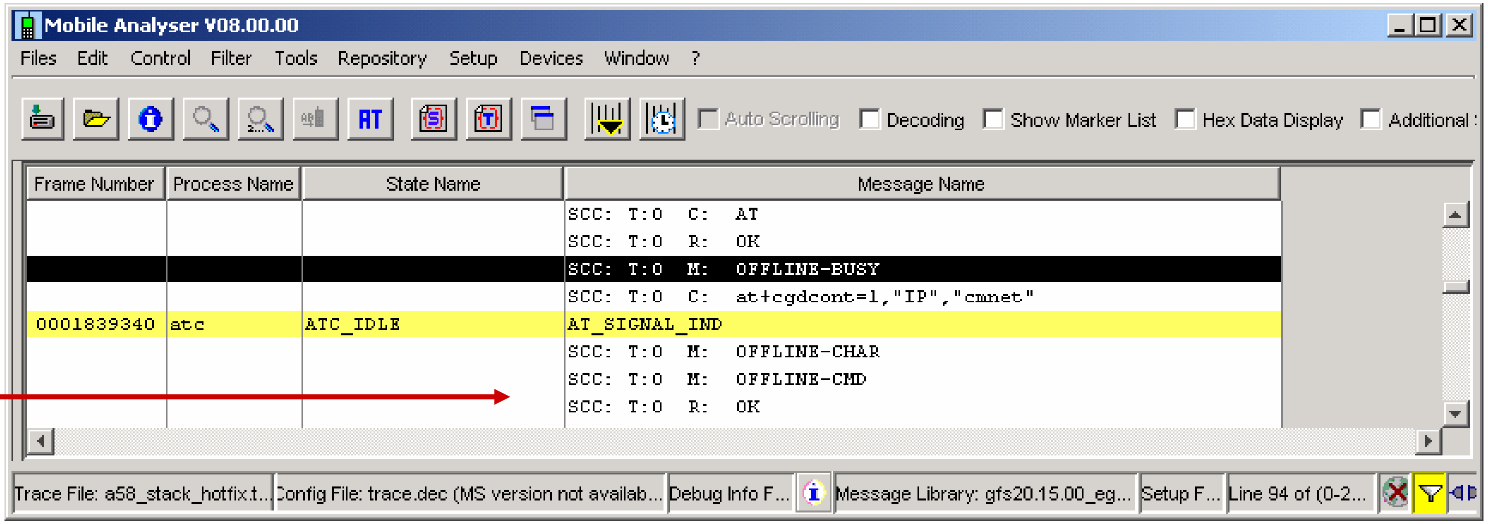
Task debug signal


Mobile Analyser V08.00.00
Files Edit Control Filter Tools Repository Setup Devices Window ?
AT

☐ Auto Scrolling ☐ Decoding ☑ Show Marker List ☐ Hex Data Display ☐ Additional Script Decoding

| Frame Number | Process Name | State Name | Message Name |
|---|---|---|---|
| 0000703634 | mac_drop | MAC_DROPCAUSE | MAC_DC_TFI_MISMATCH |
| 0000703634 | mac_drop | MAC_DROPCAUSE | MAC_DC_TFI_MISMATCH |
| 0000703664 | mac_drop | MAC_DROPCAUSE | MAC_DC_TFI_MISMATCH |
| 0000703704 | mac_db | MAC_DEBUG | MAC_IGNORE_CAB_SET_AND_IS_WRONG_BTS_BEHAVIOUR |
| 0000703768 | mac_drop | MAC_DROPCAUSE | MAC_DC_TFI_MISMATCH |

Trace File: a58_stack_hotfix_mac_dt_changed.t... Config File: trace.dec (MS version not availab... Debug Info Fi... Message Library: gfs20.15.00_eg... Setup Fi... Line 146 of (0-1...


Mobile Analyser V08.00.00
Files Edit Control Filter Tools Repository Setup Devices Window ?
AT

☐ Auto Scrolling ☐ Decoding ☐ Show Marker List ☐ Hex Data Display ☐ Additional

| Frame Number | Process Name | State Name | Message Name |
|---|---|---|---|
| | | | SCC: T:0 C: AT |
| | | | SCC: T:0 R: OK |
| | | | SCC: T:0 M: OFFLINE-BUSY |
| | | | SCC: T:0 C: at+cgdcont=1,"IP","cmnet" |
| 0001839340 | atc | ATC_IDLE | AT_SIGNAL_IND |
| | | | SCC: T:0 M: OFFLINE-CHAR |
| | | | SCC: T:0 M: OFFLINE-CMD |
| | | | SCC: T:0 R: OK |

Trace File: a58_stack_hotfix.t... Config File: trace.dec (MS version not availab... Debug Info F... Message Library: gfs20.15.00_eg... Setup F... Line 94 of (0-2...

Printf called by

Serial Controller

(AT commands)

# Real time debugging

- System crashes and system exits with power off are the most serious and tricky type of errors. There are two kinds of errors: treatable exceptions and silent crashes.

1. “Treatable” exceptions are errors recognized and intercepted by the system, e.g. HW and SW traps due to wrong MCU processing, SW traps raised by the operating system, violations of asserts, explicit “exit” instructions called by the SW when a logical or procedural error condition is met.
2. “Silent” crashes are all system exits that cannot be numbered among the previous ones. Corruptions of the stack pointer can lead to such errors.

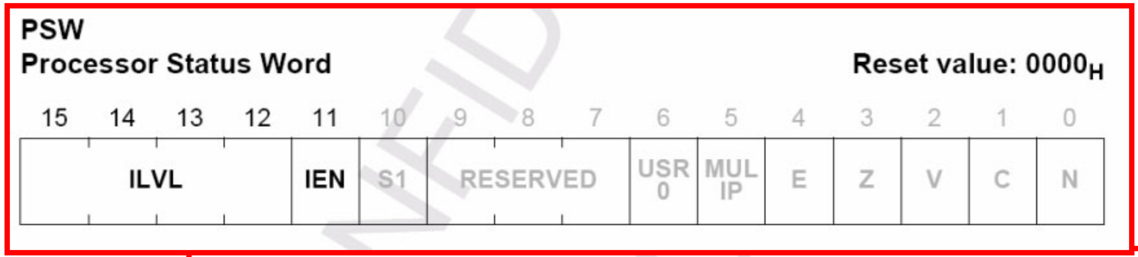
# “Treatable” exceptions

- To investigate the occurrence of a crash, an appropriate set of breakpoints shall be selected and used with the real-time debugger tool (Lauterbach).
- Once the system has halted, the tool will make some useful information available to the developer, i.e:
  - After an exception, **PSW**, **CSP** (in segmentation mode) and **IP** have been pushed into the system stack. PSW, Stack Pointer and Registers are accessible via CPU→CPU Registers
  - All chipset registers, available via menu/view/peripherals once the file <chipset>.per is made visible.
  - In case the .cmm file does not provide it by default, the system stack can be inspected by entering the following command:

```
d.v %SYMBOL.LONG register(sp) /TRACK
```

- Let’s discuss the main exceptions with more detail.

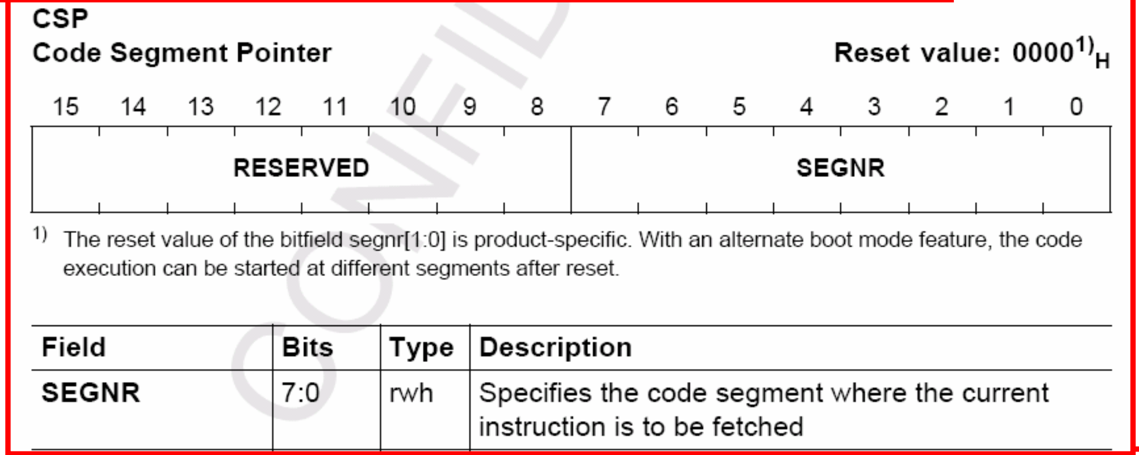
# PSW, CPS, IP



**PSW**

**Processor Status Word** — **Reset value: $0000_H$**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ILVL | | | | IEN | S1 | RESERVED | | | USR 0 | MUL IP | E | Z | V | C | N |



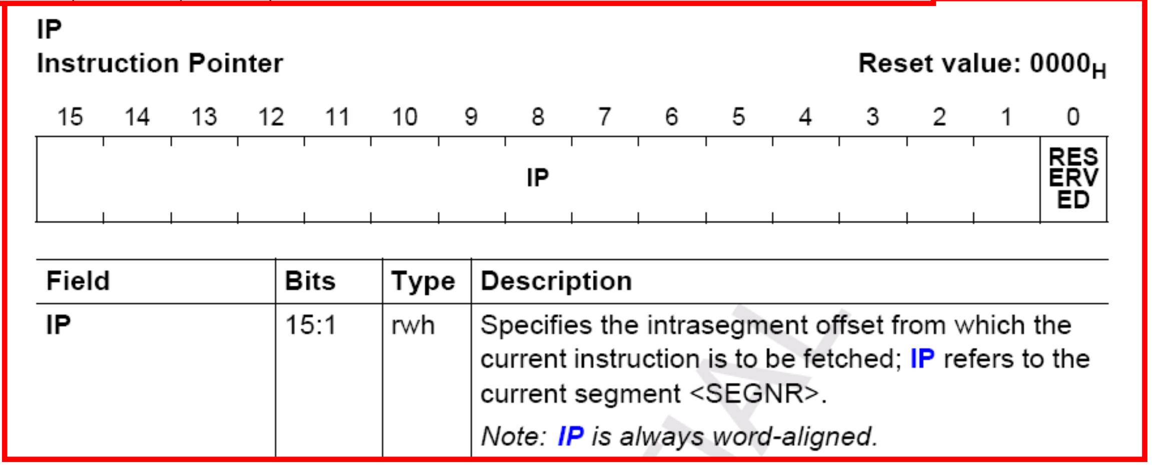
**CSP**

**Code Segment Pointer** — **Reset value: $0000^{1)}_H$**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RESERVED | | | | | | | | SEGNR | | | | | | | |

[1] The reset value of the bitfield segnr[1:0] is product-specific. With an alternate boot mode feature, the code execution can be started at different segments after reset.

| Field | Bits | Type | Description |
|---|---|---|---|
| **SEGNR** | 7:0 | rwh | Specifies the code segment where the current instruction is to be fetched |



**IP**

**Instruction Pointer** — **Reset value: $0000_H$**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IP | | | | | | | | | | | | | | | RES ERV ED |

| Field | Bits | Type | Description |
|---|---|---|---|
| **IP** | 15:1 | rwh | Specifies the intrasegment offset from which the current instruction is to be fetched; IP refers to the current segment <SEGNR>.<br>*Note: IP is always word-aligned.* |

# Software traps

- The TRAP instruction is used to cause a software call to an ISR. The trap number that is specified in the operand field of the trap instruction determines which vector location of the vector table will be used.
- The TRAP instruction’s effect is similar to that of an interrupt request that uses the same vector. **PSW**, **CSP** and **IP** are pushed into the system stack and then a jump is taken to the specified vector location.
- SW traps and invoked by setting the NMI flag:

```
void TRAP_envoke_sw_trap( unsigned int id_number)
{  exception_id_number = id_number;
  TFR |= TRAP_NMI; /* Envoke the Non Maskable Interrupt Trap flag */
}
```

- This function is called by ms_exit and ose_exception_handler.

# ms_exit

- This function is called by all error conditions raised by the SW. In particular, it is called by ms_severe_exception (all tasks of the stack) and by ms_error (L1 specific) only when:

```
if( severity <= ms_error_severity )
  ms_severe_exception( line_number, filename, error_code );
```

- There are 3 levels of the **severity** parameters of ms_error:

```
#define MS_ERROR_SEVERE 0
#define MS_ERROR_LOCAL 1
#define MS_ERROR_WARN 2
```

- Debug versions are released with **ms_error_severity** set to 3.
- In case TRAP_HANDLING is not defined, expicit calls to ms_exit by the source code are substituted by occurrences of exit(0), which later calls ms_exit as well.

```
#ifdef TRAP_HANDLING
  ms_exit(i2_filename,i1_line_number,i3_error_code);
#else
  exit (0);
```

# ms_exit (2)

- When you debug MMI or high level task in a step by step fashion or using SW breakpoints that introduces delays in the system executions, you affect the timings of the Layer 1. This often leads to system exits due to timings violations, e.g. the so called frame overruns.
- To prevent calls to ms_exit when such warnings are produced, the global variable **ms_error_severity** shall be set to 0.
- This can be achieved:
  - by modifying its value in file system-build\make\makeoptions.mk

    ```
    SYSTEM_DEFS += MS_ERROR_SEVERITY=0
    ```
  - runtime, by changing it with the real-time debugger once the system has halted.
  - runtime, by entering the following command

    ```
    at+xl1set="sev0"
    ```

# Ose_exception_handler

- This is the error function called by OSE operating system when an unrecoverable error occurs.
- In order to distinguish the kind of error for debugging purposes, the following variable (symbol) can be tracked with the debugger: **ERR_MSG**.
- In particolar (see OSE_manuals\Ec166KrnRM3_0.pdf)

```
    switch(ERR_MSG[0]) {
case 0x02: //The memory pool was empty when the designated process
    tried to allocate memory .
case 0x03: //The designated process called FREE providing a NIL-pointer
case 0x1E: //Internal stack overflow. The internal stack for designated
    process is too small. Increase the internal stack size in os166.con
case 0x3C: //Interrupt occurred from a source that has no process
    associated with it. To get the trap number subtract the value pointed to by
    (stackpointer+4) by 4 and then divide it with 4.
```

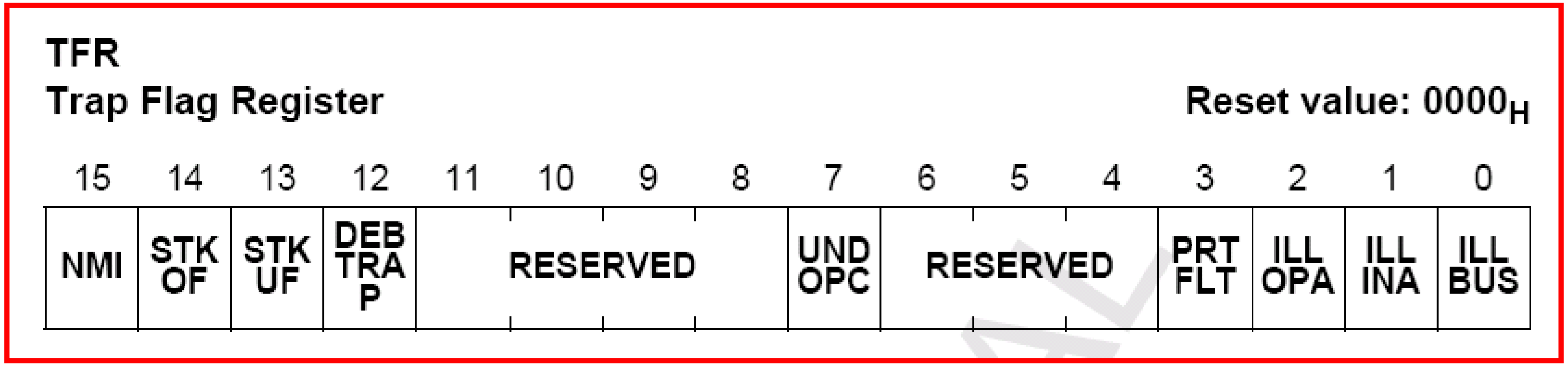
# Hardware traps

- To summarize, we have 3 exception categories:
  - Software raised exceptions and software generated traps (already discussed),
  - Hardware (EGL/EGR) detected abnormalities (HW traps).
- Hardware traps are issued by faults or specific system states that occur during runtime.
- When a hardware trap condition has been detected, the MCU branches to the related trap vector location and a TRAP instruction is injected into the pipeline:
  - Push **PSW**, **CSP** and **IP** onto the system stack
  - Set the **PSW** to the highest priority level, which disables all interrupts
  - Branch to the trap vector location specified by the trap number
- Hardware traps are not-maskable and always have a higher priority than any other MCU task.

# Hardware traps (2)

- The C166S distinguishes eight different hardware trap functions, collected in 2 classes.
- Class A traps (same trap priority, individual vector address):
  - External NMIs
  - Stack overflow
  - Stack underflow
  - Software Break.
- Class B traps (same interrupt vector, trap identified by **TFR**):
  - Undefined opcode
  - Protection fault
  - Illegal word operand access
  - Illegal instruction access
  - Illegal external bus access.

# Hardware traps (3)



**TFR**

**Trap Flag Register** **Reset value: $0000_H$**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NMI | STK<br>OF | STK<br>UF | DEB<br>TRA<br>P | RESERVED | | | | UND<br>OPC | RESERVED | | | PRT<br>FLT | ILL<br>OPA | ILL<br>INA | ILL<br>BUS |

# Hardware traps (4)

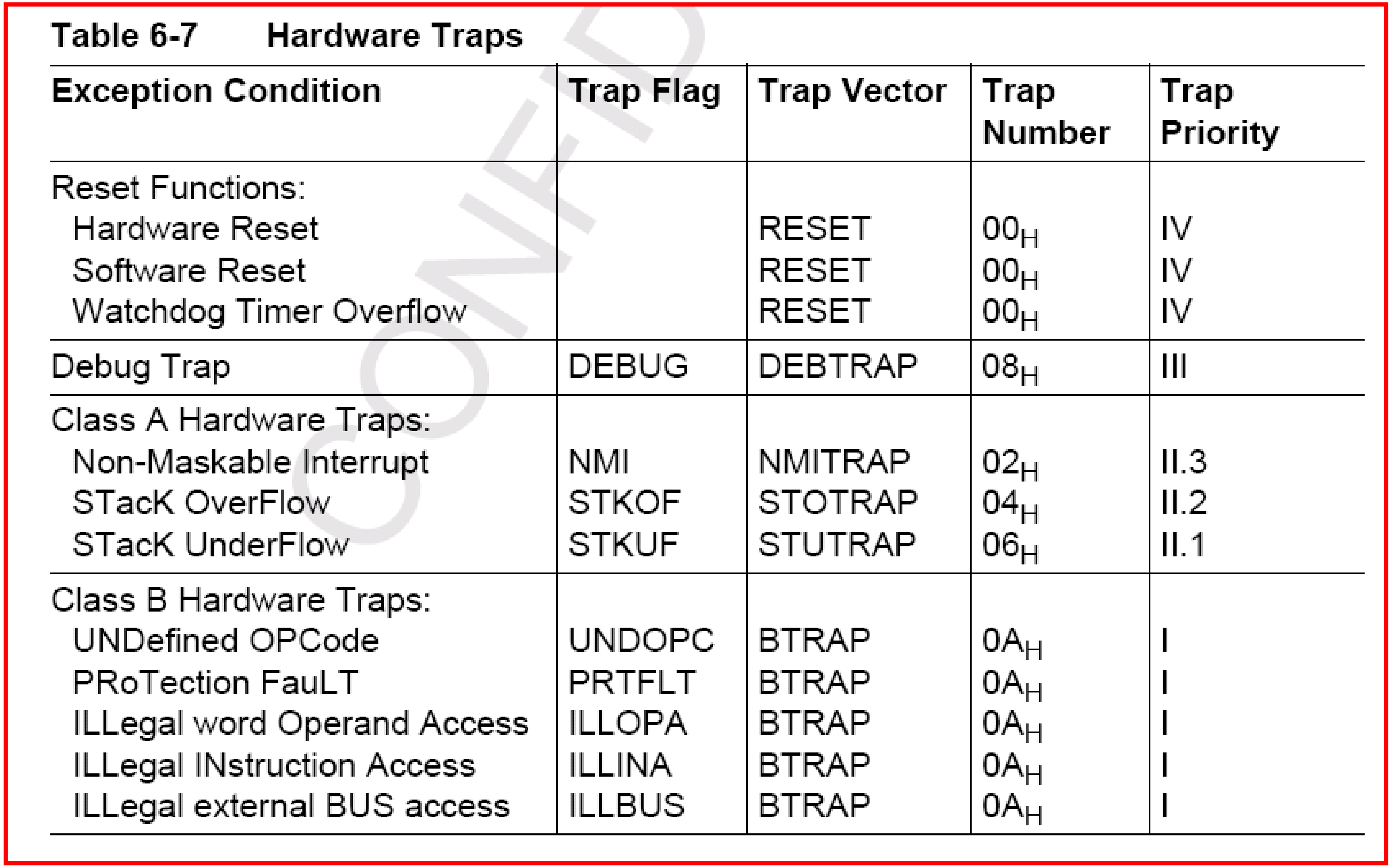


**Table 6-7 Hardware Traps**

| Exception Condition | Trap Flag | Trap Vector | Trap Number | Trap Priority |
|---|---|---|---|---|
| Reset Functions: | | | | |
| Hardware Reset | | RESET | $00_H$ | IV |
| Software Reset | | RESET | $00_H$ | IV |
| Watchdog Timer Overflow | | RESET | $00_H$ | IV |
| Debug Trap | DEBUG | DEBTRAP | $08_H$ | III |
| Class A Hardware Traps: | | | | |
| Non-Maskable Interrupt | NMI | NMITRAP | $02_H$ | II.3 |
| STacK OverFlow | STKOF | STOTRAP | $04_H$ | II.2 |
| STacK UnderFlow | STKUF | STUTRAP | $06_H$ | II.1 |
| Class B Hardware Traps: | | | | |
| UNDefined OPCode | UNDOPC | BTRAP | $0A_H$ | I |
| PRoTection FauLT | PRTFLT | BTRAP | $0A_H$ | I |
| ILLegal word Operand Access | ILLOPA | BTRAP | $0A_H$ | I |
| ILLegal INstruction Access | ILLINA | BTRAP | $0A_H$ | I |
| ILLegal external BUS access | ILLBUS | BTRAP | $0A_H$ | I |

# TRAP_class_a_handling

- **External NMIs**: any transition of the NMI flag bit leads to the invocation of the trap function TRAP_class_a_handling
- **Stack overflow/Stack underflow**: whenever the stack pointer (**SP**) is de/incremented to a value less/more than the value in the stack overflow/underflow registers **STKOV/STKUN**; in CSTART:

```
MOV STKOV, #?SYSSTACK_BOTTOM + 6*2 ;Set stack underflow pointer
MOV STKUN, #?SYSSTACK_TOP           ;Set stack overflow pointer
```

- **Software Break**: related to JTAG debugging features
- Class A traps cannot interrupt an atomic/extend sequence.

# TRAP_class_b_handling

- **UNDefined OPCode Trap (UNDOPC):** the current instruction decoded by the MCU does not contain a valid C166S opcode.
- **PRoTection FauLT Trap (PRTFLT):** whenever one protected instruction (e.g. EINIT - end of initialization, IDLE - power down CPU, SRST – SW reset) is executed, the **TFR.PRTFLT** flag is set and the MCU enters the protection fault trap routine.
- **ILLegal word OPerand Access Trap (ILLOPA):** a word operand read or write access is attempted to an odd byte address.
- **ILLegal INstruction Access Trap (ILLINA):** a branch is made to an odd byte address.
- **ILLegal external BUS access Trap (ILLBUS):** the MCU requests an external instruction fetch or a data read or write and no external bus configuration has been specified.

- Class B traps can interrupt an atomic/extend sequence.

# TRAP handling

- The trap interrupt vectors (4 bytes) are located in file prolog.scf.in:

```
CLASSES('TRAP_CLASS_A_NMI'   (0000008h TO 000000Bh UNIQUE))
CLASSES('TRAP_CLASS_A_STO'   (0000010h TO 0000013h UNIQUE))
CLASSES('TRAP_CLASS_A_STU'   (0000018h TO 000001Bh UNIQUE))
CLASSES('T32_TRAP_CLASS'     (0000020h TO 0000023h UNIQUE))
CLASSES('TRAP_CLASS_B'       (0000028h TO 000002Bh UNIQUE))
```

- and are described in prolog_debug.asm:

```
sNMIJMPSINT SECTION CODE WORD 'TRAP_CLASS_A_NMI'
pNMIJMPSINT PROC NEAR
    JMPS    _TRAP_class_a_handling
    RETV
pNMIJMPSINT ENDP
sNMIJMPSINT ENDS
sSTOJMPSINT SECTION CODE WORD 'TRAP_CLASS_A_STO'
pSTOJMPSINT PROC NEAR
    JMPS    _TRAP_class_a_handling
    RETV
pSTOJMPSINT ENDP
sSTOJMPSINT ENDS
```

# TRAP handling (2)

- All other interrupt vectors are described in os166.src → see next slide
- Both TRAP_class_a_handling and TRAP_class_b_handling end with the SRST instruction if SILENT_RESET is defined.
- For TRAP handling you can also refer to document by DWD How2Debug\Trap_Exception_presentation.pdf

# Interrupt handling (os166.src)

```
OS166_V_43(50) SECTION CODE AT 010Ch+00h (@200)
       DB     0FAh
       DB     SEG ZZ_I_43
       DW     SOF ZZ_I_43
OS166_V_43     ENDS
       PUBLIC l1x_lisr1_actions_
l1x_lisr1_actions_    EQU    DATA8 14
OS166_VC_43    SECTION CODE WORD PUBLIC 'OSE_C'
OS166_VCR_43   PROC NEAR
ZZ_I_43:
       ATOMIC #3
       ...
       SCXT   DPP2,#PAG DPP2_LARGE
       SCXT   DPP0,#PAG BASE_DPP0
       SCXT   M_DC,#010h
       PUSH   MDH
       PUSH   MDL
       ...
       CALLS  SEG _l1x_lisr1_actions,_l1x_lisr1_actions
```

# epAllOff

- Sometime the system cannot boot and switches off.
- In order to verify whether the power off is abnormal, you can set a BP at **epAllOff** function, which may be called by OMS in case of exceptions in the power-on sequence, for example if not all drivers have correclty indicated their initialization to MMI.

# Trap::Assertion

- APOXI and MMI are disseminated with ASSERT(cond) instruction, which checks that the condition <cond> holds.
- If the check fails, depending on the define APOXI_ENABLE_ TRAP_INFO_SCREEN, the system can either display the filename and line where the error occurred or call **ms_exit** (eventually a silent reset can be trigger via SRST instruction).

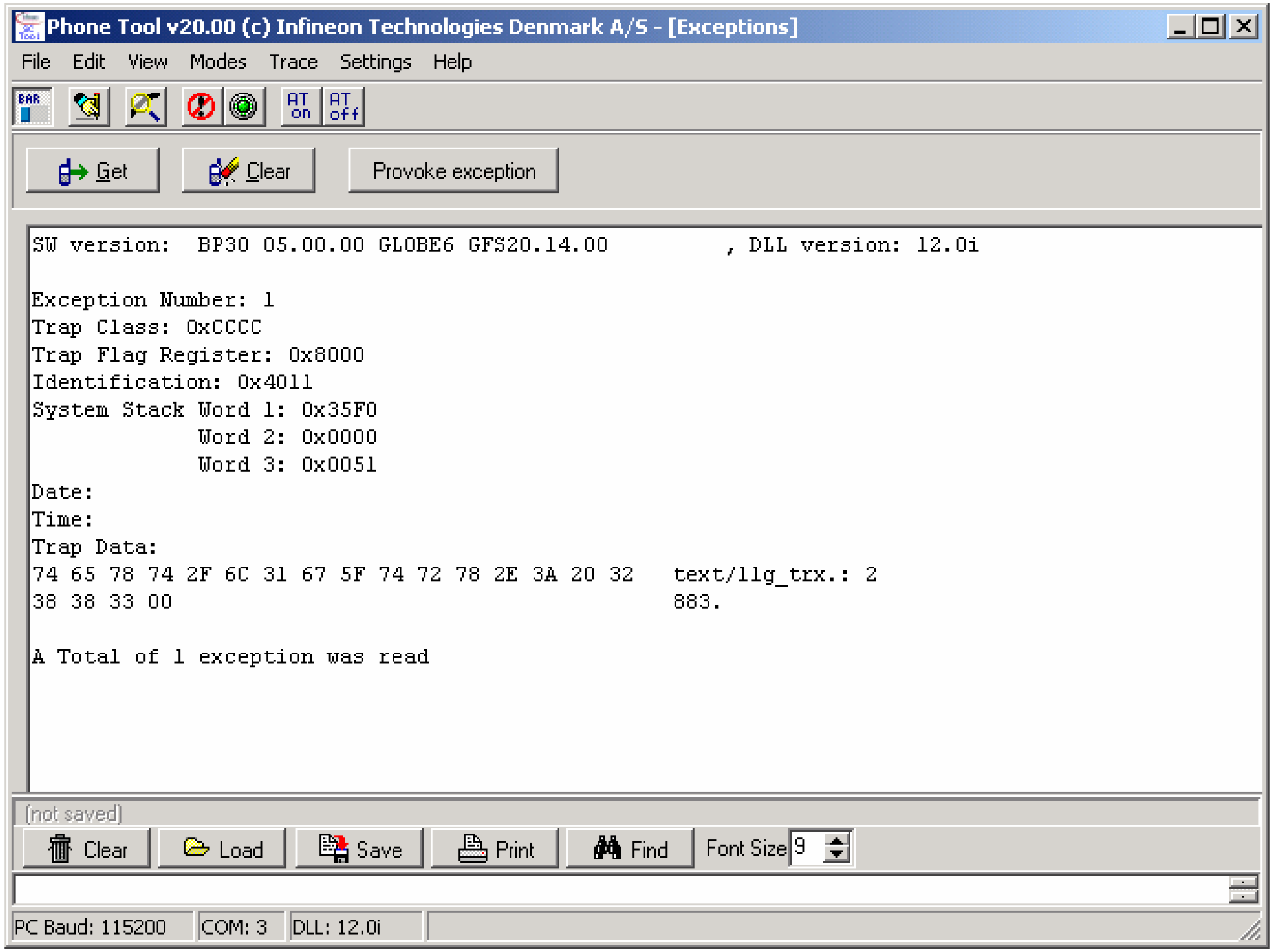
# Logging of exceptions

- All “treatable” or intercepted exceptions are (should be) stored to a predefined area in the EEProm and can be read/cleared using Phonetool.
- If TRAP_HANDLING is defined, the same memory area can host also non-unrecoverable error conditions (e.g. warnings and errors whose severity is lower than the mimimum tolerable one), which are stored by means of the function:

  void **TRAP_store_exception**(unsigned int id_number, unsigned char log_data_size, void *log_data);
- This function can be used anytime an exception has to be stored in the *exception store* structure. This store will be transferred to NVRAM at traps or at drivers deactivation during power down.
- Please note that not all traps stored to EEP correspond to system crashes.

# Logging of exceptions (2)

- Exception list obtained by Phonetool



```
SW version:  BP30 05.00.00 GLOBE6 GFS20.14.00         , DLL version: 12.0i

Exception Number: 1
Trap Class: 0xCCCC
Trap Flag Register: 0x8000
Identification: 0x4011
System Stack Word 1: 0x35F0
             Word 2: 0x0000
             Word 3: 0x0051
Date:
Time:
Trap Data:
74 65 78 74 2F 6C 31 67 5F 74 72 78 2E 3A 20 32   text/llg_trx.: 2
38 38 33 00                                        883.

A Total of 1 exception was read
```

# Logging of exceptions (3)

- Examples of warning and errors in the source code of L1

Warnings:

```
if (pdch_rx_tx.deact_flg)
{
  /* re-activation not allowed before deactivation is confirmed */
  MS_ERROR( MS_L1_STM_PREPARE_FAIL, MS_ERROR_WARN, FALSE );
  return;
}
```

Local error:

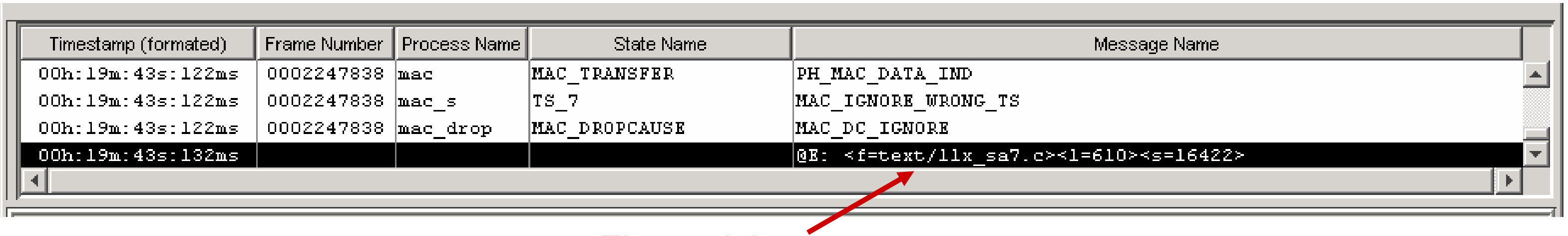
```
else
 MS_ERROR( MS_L1_STM_PREPARE_FAIL, MS_ERROR_LOCAL, FALSE );
```

Severe error:

```
 if (l1d_fcb_stm_in_use() ||  l1d_sb_stm_in_use() )
  MS_ERROR( MS_L1_STM_FAULT, MS_ERROR_SEVERE, FALSE );
```

# Logging of exceptions (4)

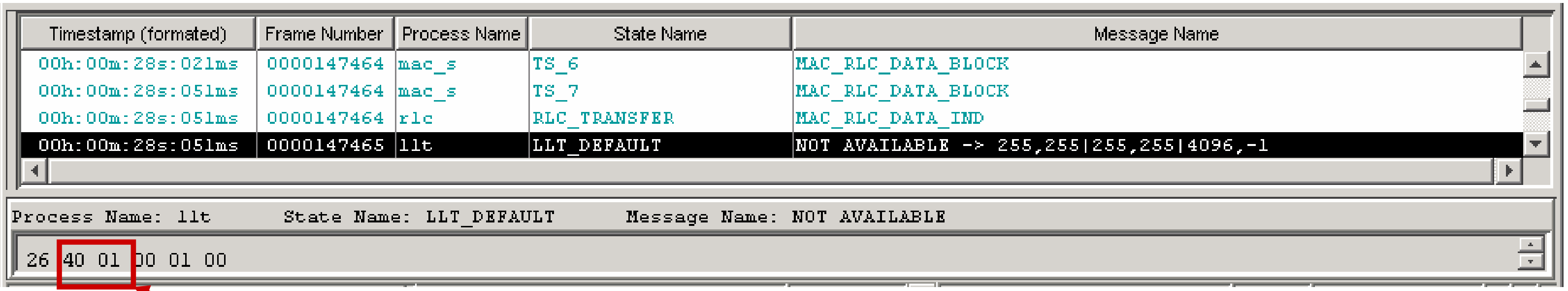
- The trace logs indicate that an exception occurred by printing some information at the end of the trace.



| Timestamp (formated) | Frame Number | Process Name | State Name | Message Name |
|---|---|---|---|---|
| 00h:19m:43s:122ms | 0002247838 | mac | MAC_TRANSFER | PH_MAC_DATA_IND |
| 00h:19m:43s:122ms | 0002247838 | mac_s | TS_7 | MAC_IGNORE_WRONG_TS |
| 00h:19m:43s:122ms | 0002247838 | mac_drop | MAC_DROPCAUSE | MAC_DC_IGNORE |
| 00h:19m:43s:132ms | | | | @E: <f=text/llx_sa7.c><l=610><s=16422> |

File and line

- If the error severity is set to 0 (i.e. only severe exceptions lead to ms_exit), L1 warnings are traced with a LLT message:



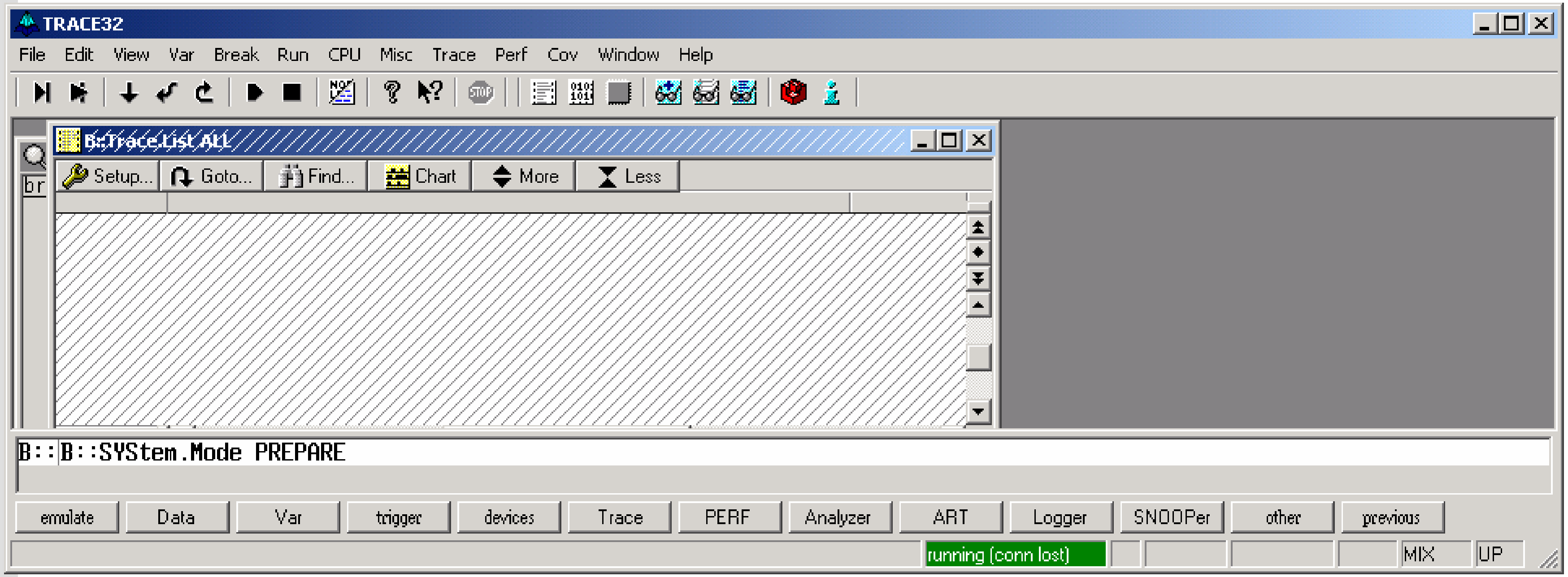
| Timestamp (formated) | Frame Number | Process Name | State Name | Message Name |
|---|---|---|---|---|
| 00h:00m:28s:021ms | 0000147464 | mac_s | TS_6 | MAC_RLC_DATA_BLOCK |
| 00h:00m:28s:051ms | 0000147464 | mac_s | TS_7 | MAC_RLC_DATA_BLOCK |
| 00h:00m:28s:051ms | 0000147464 | rlc | RLC_TRANSFER | MAC_RLC_DATA_IND |
| 00h:00m:28s:051ms | 0000147465 | llt | LLT_DEFAULT | NOT AVAILABLE -> 255,255\|255,255\|4096,-1 |

```
Process Name: llt      State Name: LLT_DEFAULT      Message Name: NOT AVAILABLE
26 40 01 00 01 00
```

Error type

# Non-treatable exceptions: Connection lost

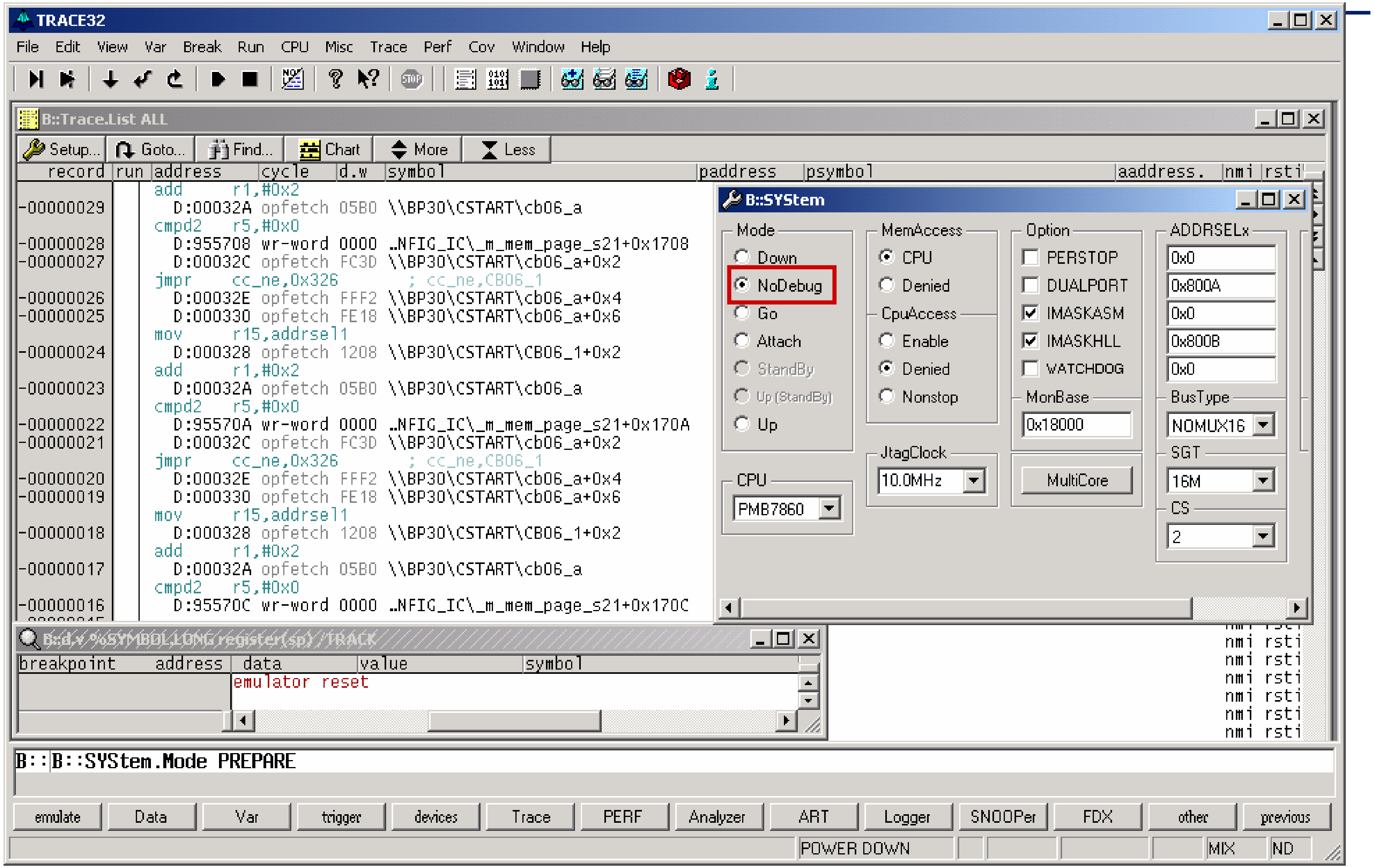
- When no BP is available for debugging, the DUT goes to “connection loss” state. In this case you have two options:


TRACE32
File Edit View Var Break Run CPU Misc Trace Perf Cov Window Help
B::Trace.List ALL
Setup... Goto... Find... Chart More Less
emulate Data Var trigger devices Trace PERF Analyzer ART Logger SNOOPer other previous
running (conn lost)
MIX UP

```
B::B::SYStem.Mode PREPARE
```

1. Pre-requisite: have the Power Trace connected
2. Select “CPU→System Settings→No debug” and then “Trace→List→All” : the backtrace will be populated!

# Non-treatable exceptions: Connection lost (2)


TRACE32
File Edit View Var Break Run CPU Misc Trace Perf Cov Window Help
B::Trace.List ALL
Setup... Goto... Find... Chart More Less
B::SYStem
MultiCore
B::d.v %SYMBOL.LONG register(sp) /TRACK
emulate Data Var trigger devices Trace PERF Analyzer ART Logger SNOOPer FDX other previous
POWER DOWN
MIX
ND

```
    record run address    cycle   d.w  symbol                          paddress  psymbol   aaddress. nmi rsti
               add       r1,#0x2
-00000029        D:00032A opfetch 05B0 \\BP30\CSTART\cb06_a
               cmpd2     r5,#0x0
-00000028        D:955708 wr-word 0000 ..NFIG_IC\_m_mem_page_s21+0x1708
-00000027        D:00032C opfetch FC3D \\BP30\CSTART\cb06_a+0x2
               jmpr      cc_ne,0x326         ; cc_ne,CB06_1
-00000026        D:00032E opfetch FFF2 \\BP30\CSTART\cb06_a+0x4
-00000025        D:000330 opfetch FE18 \\BP30\CSTART\cb06_a+0x6
               mov       r15,addrsel1
-00000024        D:000328 opfetch 1208 \\BP30\CSTART\CB06_1+0x2
               add       r1,#0x2
-00000023        D:00032A opfetch 05B0 \\BP30\CSTART\cb06_a
               cmpd2     r5,#0x0
-00000022        D:95570A wr-word 0000 ..NFIG_IC\_m_mem_page_s21+0x170A
-00000021        D:00032C opfetch FC3D \\BP30\CSTART\cb06_a+0x2
               jmpr      cc_ne,0x326         ; cc_ne,CB06_1
-00000020        D:00032E opfetch FFF2 \\BP30\CSTART\cb06_a+0x4
-00000019        D:000330 opfetch FE18 \\BP30\CSTART\cb06_a+0x6
               mov       r15,addrsel1
-00000018        D:000328 opfetch 1208 \\BP30\CSTART\CB06_1+0x2
               add       r1,#0x2
-00000017        D:00032A opfetch 05B0 \\BP30\CSTART\cb06_a
               cmpd2     r5,#0x0
-00000016        D:95570C wr-word 0000 ..NFIG_IC\_m_mem_page_s21+0x170C
```

B::SYStem

Mode: ○ Down ● NoDebug ○ Go ○ Attach ○ StandBy ○ Up (StandBy) ○ Up

CPU: PMB7860

MemAccess: ● CPU ○ Denied

CpuAccess: ○ Enable ● Denied ○ Nonstop

JtagClock: 10.0MHz

Option: □ PERSTOP □ DUALPORT ☑ IMASKASM ☑ IMASKHLL □ WATCHDOG

MonBase: 0x18000

ADDRSELx: 0x0 / 0x800A / 0x0 / 0x800B / 0x0

BusType: NOMUX16

SGT: 16M

CS: 2

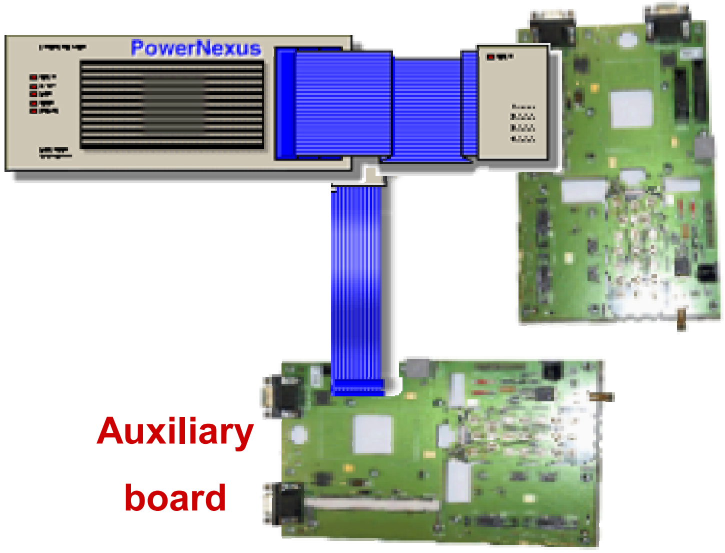
```
breakpoint     address   data             value         symbol
                         emulator reset
```

```
nmi rsti
nmi rsti
nmi rsti
nmi rsti
nmi rsti
nmi rsti
```

```
B::B::SYStem.Mode PREPARE
```

# Non-treatable exceptions: Connection lost (3)

2. Connect the Power Trace to the DUT and the JTAG to another board.
   - Make sure you are using the .abs file of the version loaded on the DUT
   - When the DUT crashes, stop the auxiliary board and look to the backtrace.

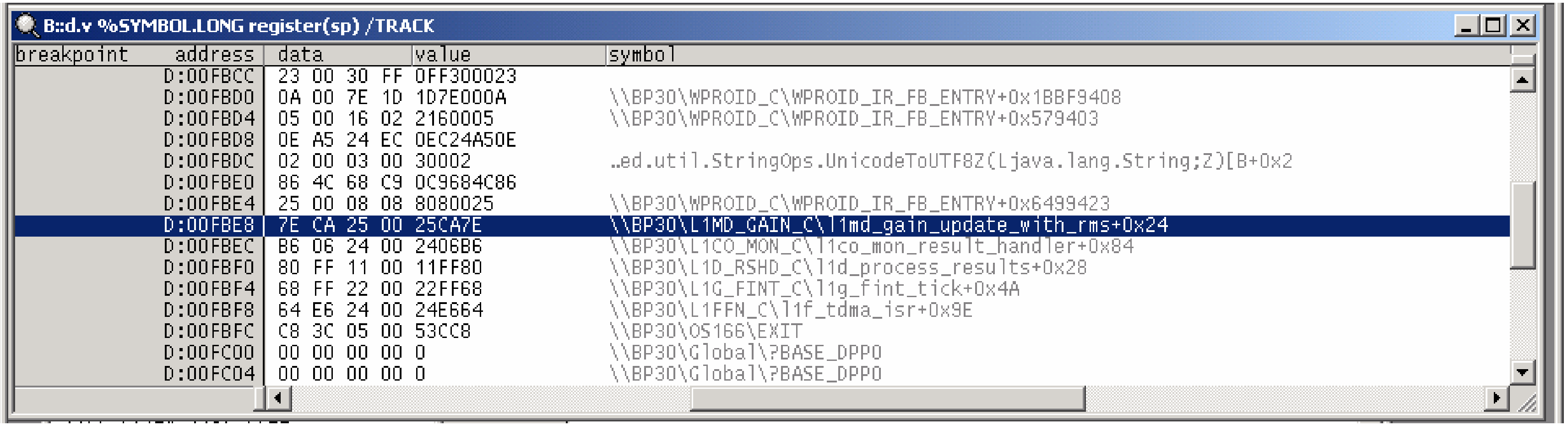


- This approach is useful to trace DUT in power saving mode, because it prevents the JTAG connector from keeping the MS awaken and lengthens the temporal duration of the backtrace (which is otherwise ca 2 seconds).

# Lauterbach: how to use the function call stack

- To show it, the .cmm file shall include at the end:

```
d.v %SYMBOL.LONG register(sp)  /TRACK
```



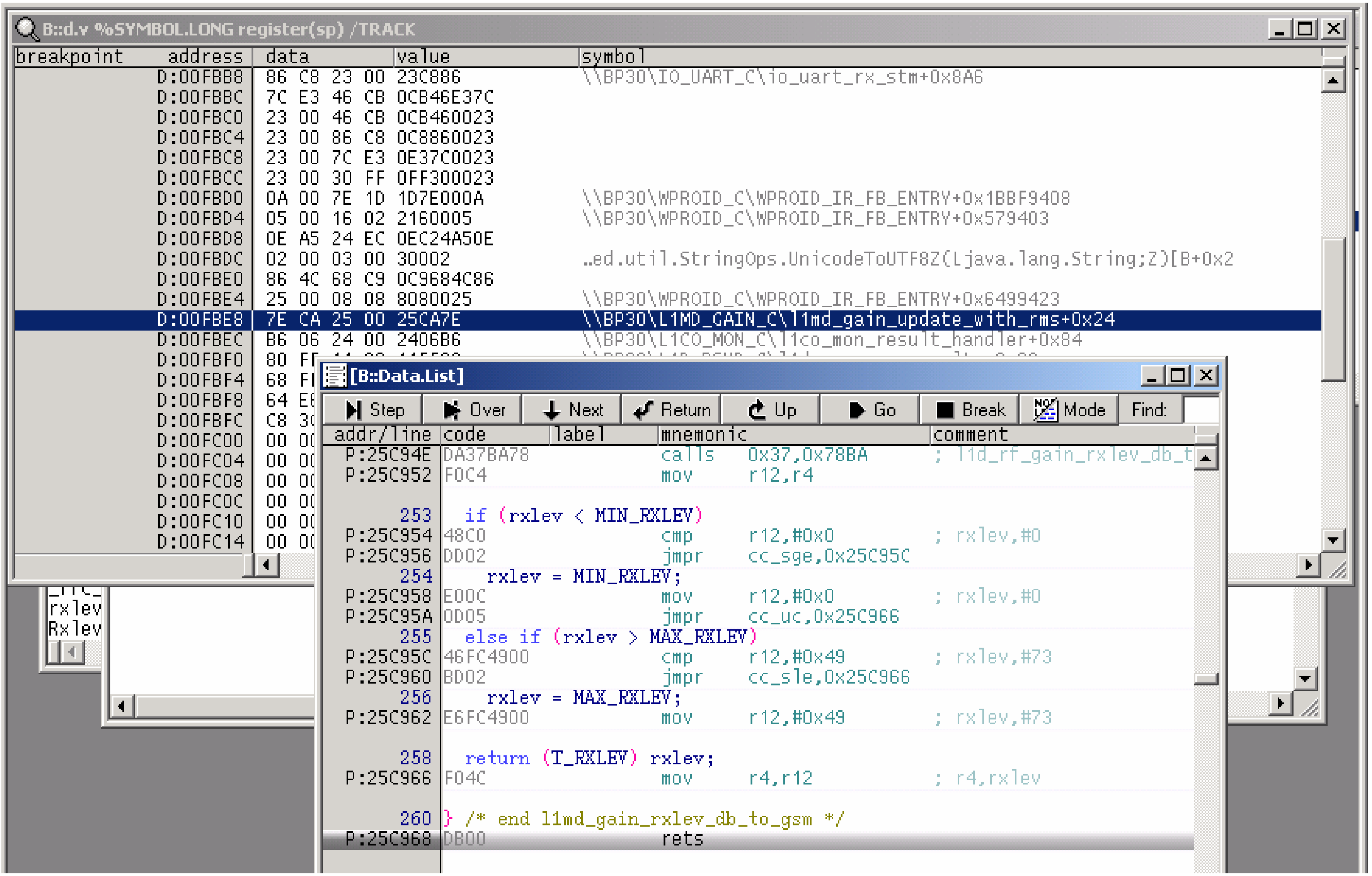
```
breakpoint      address | data        | value         symbol
              D:00FBCC  | 23 00 30 FF 0FF300023
              D:00FBD0  | 0A 00 7E 1D 1D7E000A           \\BP30\WPROID_C\WPROID_IR_FB_ENTRY+0x1BBF9408
              D:00FBD4  | 05 00 16 02 2160005            \\BP30\WPROID_C\WPROID_IR_FB_ENTRY+0x579403
              D:00FBD8  | 0E A5 24 EC 0EC24A50E
              D:00FBDC  | 02 00 03 00 30002              ..ed.util.StringOps.UnicodeToUTF8Z(Ljava.lang.String;Z)[B+0x2
              D:00FBE0  | 86 4C 68 C9 0C9684C86
              D:00FBE4  | 25 00 08 08 8080025            \\BP30\WPROID_C\WPROID_IR_FB_ENTRY+0x6499423
              D:00FBE8  | 7E CA 25 00 25CA7E             \\BP30\L1MD_GAIN_C\l1md_gain_update_with_rms+0x24
              D:00FBEC  | B6 06 24 00 2406B6             \\BP30\L1CO_MON_C\l1co_mon_result_handler+0x84
              D:00FBF0  | 80 FF 11 00 11FF80             \\BP30\L1D_RSHD_C\l1d_process_results+0x28
              D:00FBF4  | 68 FF 22 00 22FF68             \\BP30\L1G_FINT_C\l1g_fint_tick+0x4A
              D:00FBF8  | 64 E6 24 00 24E664             \\BP30\L1FFN_C\l1f_tdma_isr+0x9E
              D:00FBFC  | C8 3C 05 00 53CC8              \\BP30\OS166\EXIT
              D:00FC00  | 00 00 00 00 0                  \\BP30\Global\?BASE_DPP0
              D:00FC04  | 00 00 00 00 0                  \\BP30\Global\?BASE_DPP0
```

- Sometimes, expecially when traps occur or when the system stops in the Apoxi area, a 2 byte offset has to be applied to the SP in order to see the correct stack:

```
d.v %SYMBOL.LONG register(sp)+2 /TRACK
```

# Lauterbach: how to use the function call stack  (2)

- The current instruction can be displayed with View/List Source



B::d.v %SYMBOL.LONG register(sp) /TRACK

```
breakpoint      address  data         value          symbol
               D:00FBB8  86 C8 23 00  23C886         \\BP30\IO_UART_C\io_uart_rx_stm+0x8A6
               D:00FBBC  7C E3 46 CB  0CB46E37C
               D:00FBC0  23 00 46 CB  0CB460023
               D:00FBC4  23 00 86 C8  0C8860023
               D:00FBC8  23 00 7C E3  0E37C0023
               D:00FBCC  23 00 30 FF  0FF300023
               D:00FBD0  0A 00 7E 1D  1D7E000A       \\BP30\WPROID_C\WPROID_IR_FB_ENTRY+0x1BBF9408
               D:00FBD4  05 00 16 02  2160005        \\BP30\WPROID_C\WPROID_IR_FB_ENTRY+0x579403
               D:00FBD8  0E A5 24 EC  0EC24A50E
               D:00FBDC  02 00 03 00  30002          ..ed.util.StringOps.UnicodeToUTF8Z(Ljava.lang.String;Z)[B+0x2
               D:00FBE0  86 4C 68 C9  0C9684C86
               D:00FBE4  25 00 08 08  8080025        \\BP30\WPROID_C\WPROID_IR_FB_ENTRY+0x6499423
               D:00FBE8  7E CA 25 00  25CA7E         \\BP30\L1MD_GAIN_C\l1md_gain_update_with_rms+0x24
               D:00FBEC  B6 06 24 00  2406B6         \\BP30\L1CO_MON_C\l1co_mon_result_handler+0x84
               D:00FBF0  80 FF
               D:00FBF4  68 FI
               D:00FBF8  64 E6
               D:00FBFC  C8 30
               D:00FC00  00 00
               D:00FC04  00 00
               D:00FC08  00 00
               D:00FC0C  00 00
               D:00FC10  00 00
               D:00FC14  00 00
```

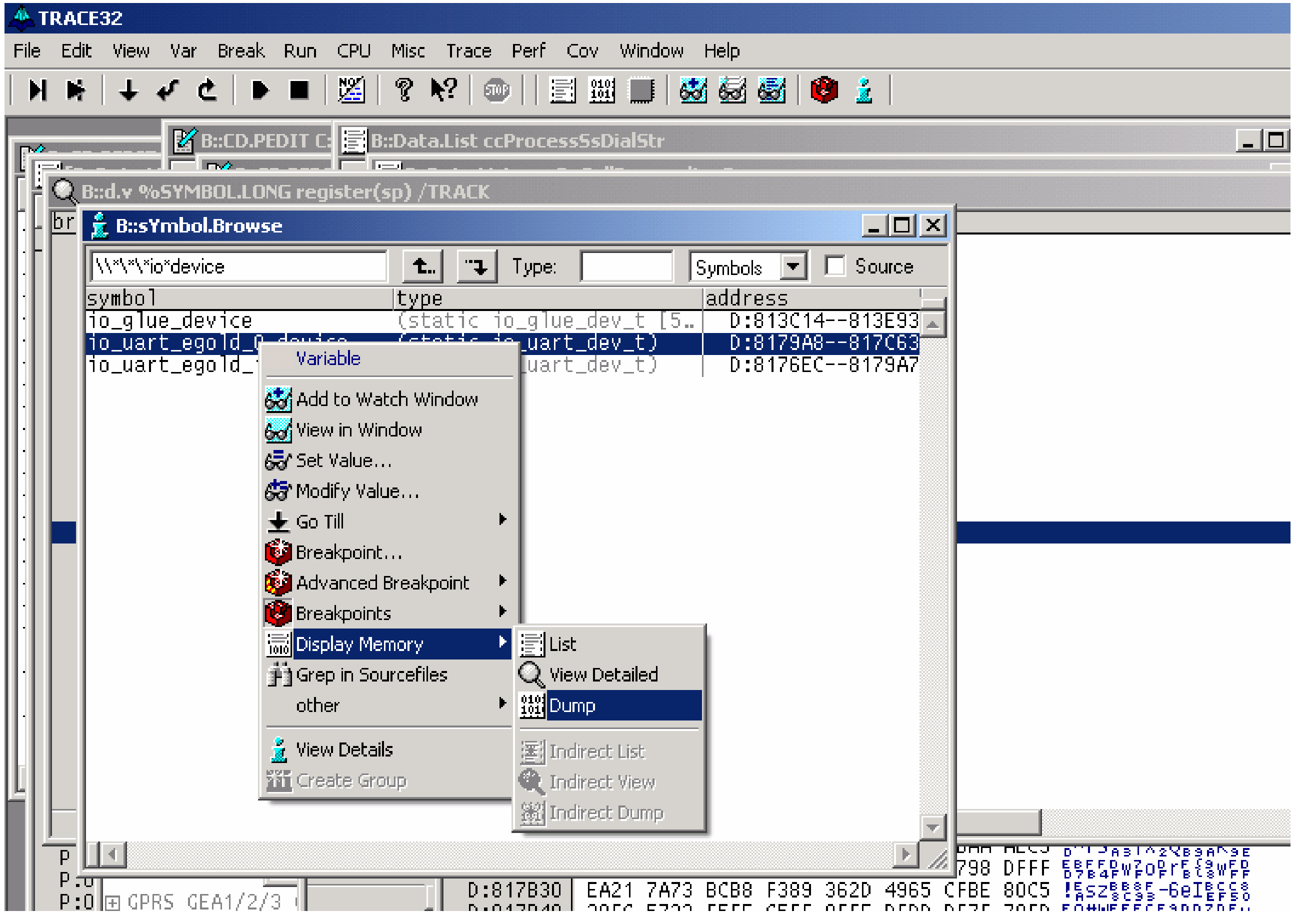
```
rxlev
Rxlev
```

[B::Data.List]

```
addr/line  code      label   mnemonic                        comment
P:25C94E   DA37BA78          calls   0x37,0x78BA             ; l1d_rf_gain_rxlev_db_t
P:25C952   F0C4              mov     r12,r4

      253    if (rxlev < MIN_RXLEV)
P:25C954   48C0              cmp     r12,#0x0                ; rxlev,#0
P:25C956   DD02              jmpr    cc_sge,0x25C95C
      254      rxlev = MIN_RXLEV;
P:25C958   E00C              mov     r12,#0x0                ; rxlev,#0
P:25C95A   0D05              jmpr    cc_uc,0x25C966
      255    else if (rxlev > MAX_RXLEV)
P:25C95C   46FC4900          cmp     r12,#0x49               ; rxlev,#73
P:25C960   BD02              jmpr    cc_sle,0x25C966
      256      rxlev = MAX_RXLEV;
P:25C962   E6FC4900          mov     r12,#0x49               ; rxlev,#73

      258    return (T_RXLEV) rxlev;
P:25C966   F04C              mov     r4,r12                  ; r4,rxlev

      260  } /* end l1md_gain_rxlev_db_to_gsm */
P:25C968   DB00              rets
```

# Lauterbach: how to save a data dump

- To save any information on a text file, you can dump the symbol:

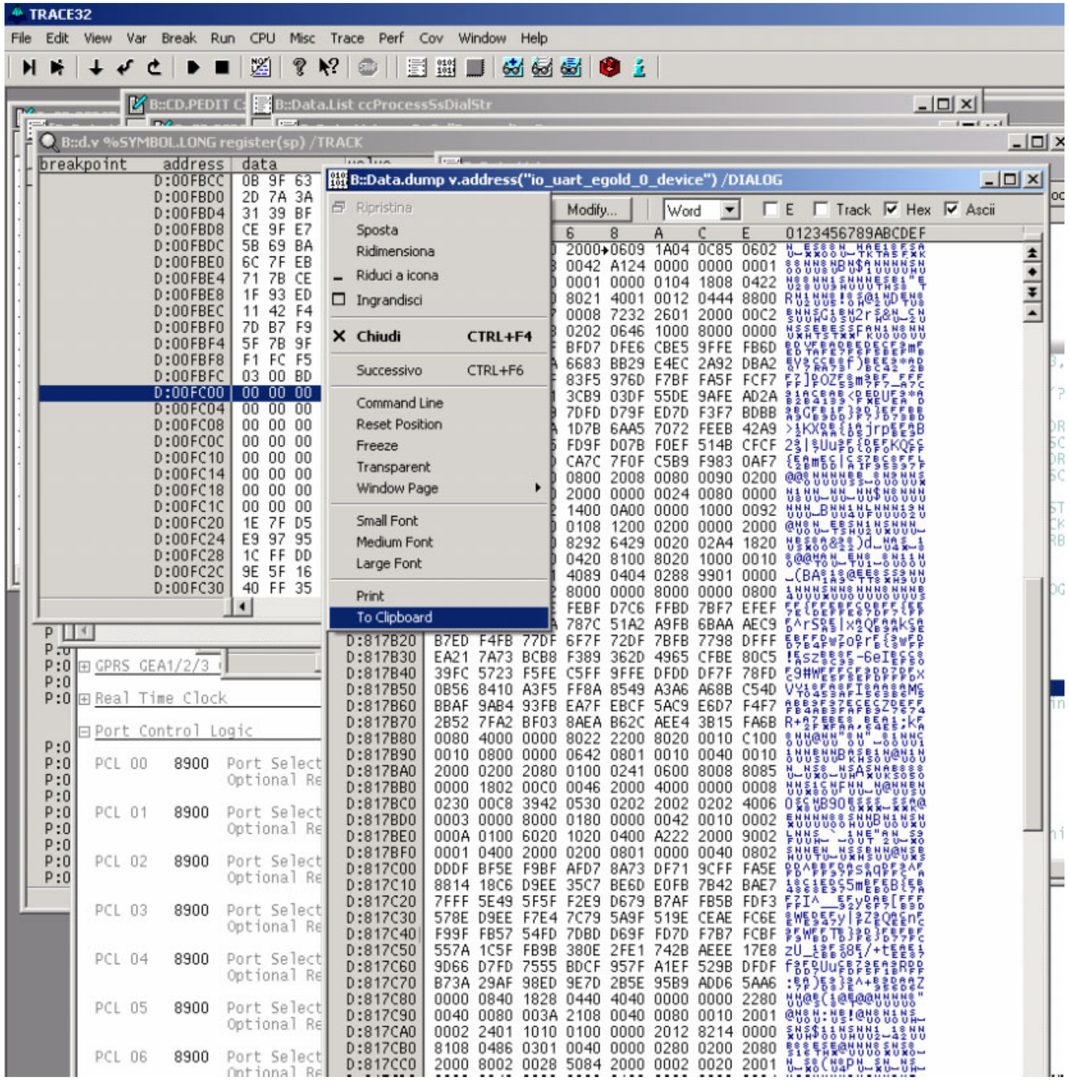


| symbol | type | address |
|---|---|---|
| io_glue_device | (static io_glue_dev_t [5.. | D:813C14--813E93 |
| io_uart_egold_0_device | (static io_uart_dev_t) | D:8179A8--817C63 |
| io_uart_egold_ | uart_dev_t) | D:8176EC--8179A7 |

```
D:817B30  EA21 7A73 BCB8 F389 362D 4965 CFBE 80C5
```

# Lauterbach: how to save a data dump (2)

- You can save it to clipboard by clicking on the upper left corner of the dump window with the left button of the mouse.
- Then you can paste it to a file
- If you have compiled with DEBUG option, you can copy the view/watch window where the variable is displayed by using the same procedure.



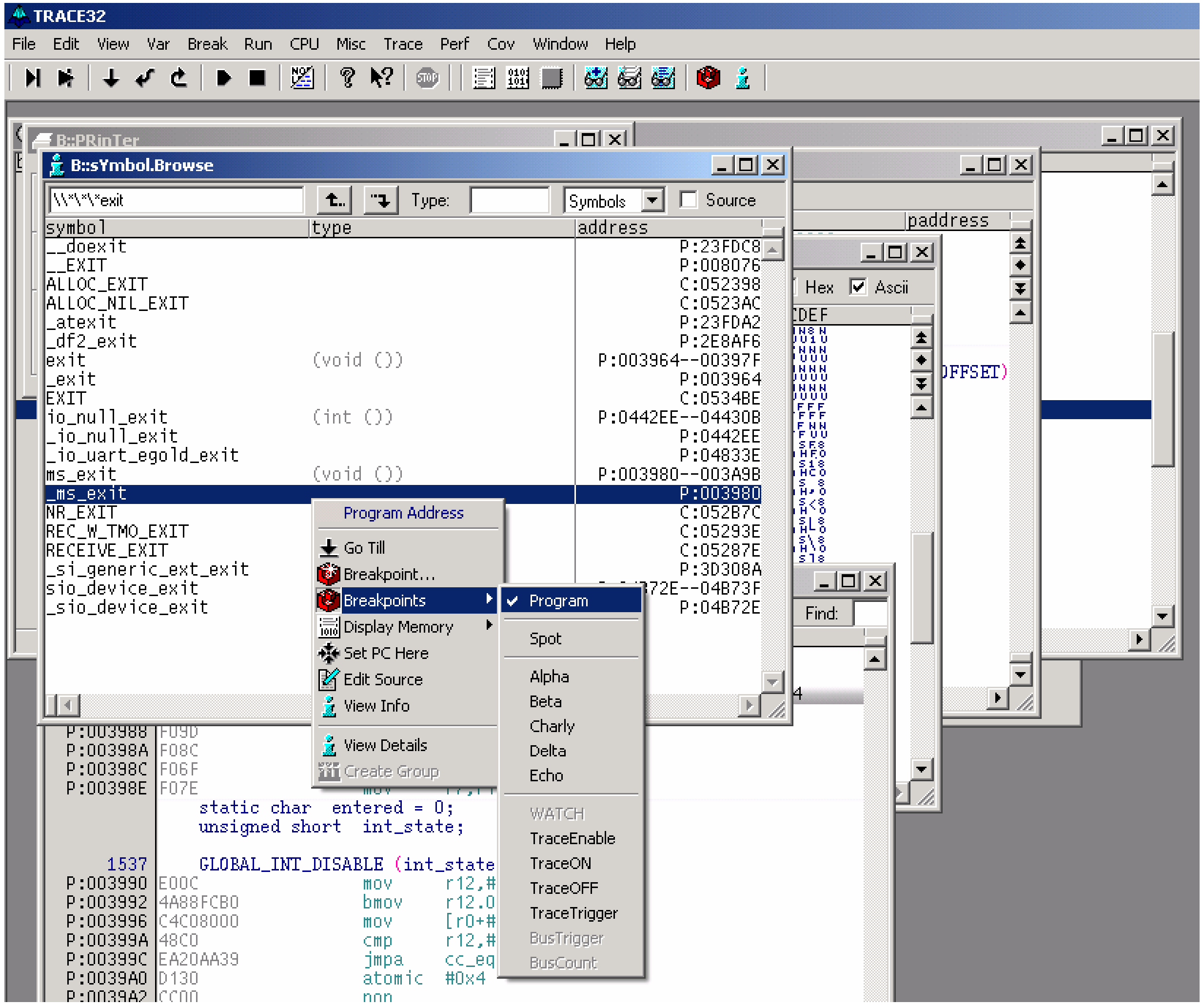
```
breakpoint    address     data
              D:00FBCC    0B 9F 63
              D:00FBD0    2D 7A 3A
              D:00FBD4    31 39 BF
              D:00FBD8    CE 9F E7
              D:00FBDC    5B 69 BA
              D:00FBE0    6C 7F EB
              D:00FBE4    71 7B CE
              D:00FBE8    1F 93 ED
              D:00FBEC    11 42 F4
              D:00FBF0    7D B7 F9
              D:00FBF4    5F 7B 9F
              D:00FBF8    F1 FC F5
              D:00FBFC    03 00 BD
              D:00FC00    00 00 00
              D:00FC04    00 00 00
              D:00FC08    00 00 00
              D:00FC0C    00 00 00
              D:00FC10    00 00 00
              D:00FC14    00 00 00
              D:00FC18    00 00 00
              D:00FC1C    00 00 00
              D:00FC20    1E 7F D5
              D:00FC24    E9 97 95
              D:00FC28    1C FF DD
              D:00FC2C    9E 5F 16
              D:00FC30    40 FF 35
```

```
PCL 00    8900    Port Select
                  Optional Re
PCL 01    8900    Port Select
                  Optional Re
PCL 02    8900    Port Select
                  Optional Re
PCL 03    8900    Port Select
                  Optional Re
PCL 04    8900    Port Select
                  Optional Re
PCL 05    8900    Port Select
                  Optional Re
PCL 06    8900    Port Select
                  Optional Re
```

```
6     8     A     C     E     0123456789ABCDEF
2000→0609  1A04  0C85  0602
0042  A124  0000  0000  0001
0001  0000  0104  1808  0422
8021  4001  0012  0444  8800
0008  7232  2601  2000  00C2
0202  0646  1000  8000  0000
BFD7  DFE6  CBE5  9FFE  FB6D
6683  BB29  E4EC  2A92  DBA2
83F5  976D  F7BF  FA5F  FCF7
3CB9  03DF  55DE  9AFE  AD2A
7DFD  D79F  ED7D  F3F7  BDBB
1D7B  6AA5  7072  FEEB  42A9
FD9F  D07B  F0EF  514B  CFCF
CA7C  7F0F  C5B9  F983  0AF7
0800  2008  0080  0090  0200
2000  0000  0024  0080  0000
1400  0A00  0000  1000  0092
0108  1200  0200  0000  2000
8292  6429  0020  02A4  1820
0420  8100  8020  1000  0010
4089  0404  0288  9901  0000
8000  0000  8000  0000  0800
FEBF  D7C6  FFBD  7BF7  EFEF
787C  51A2  A9FB  6BAA  AEC9
D:817B20  B7ED F4FB 77DF 6F7F 72DF 7BFB 7798 DFFF
D:817B30  EA21 7A73 BCB8 F389 362D 4965 CFBE 80C5
D:817B40  39FC 5723 F5FE C5FF 9FFE DFDD DF7F 78FD
D:817B50  0B56 8410 A3F5 FF8A 8549 A3A6 A68B C54D
D:817B60  BBAF 9AB4 93FB EA7F EBCF 5AC9 E6D7 F4F7
D:817B70  2B52 7FA2 BF03 8AEA B62C AEE4 3B15 FA6B
D:817B80  0080 4000 0000 8022 2200 8020 0010 C100
D:817B90  0010 0800 0000 0642 0801 0010 0040 0010
D:817BA0  2000 0200 2080 0100 0241 0600 8008 8085
D:817BB0  0000 1802 00C0 0046 2000 4000 0000 0008
D:817BC0  0230 00C8 3942 0530 0202 2002 0202 4006
D:817BD0  0003 0000 8000 0180 0000 0042 0010 0002
D:817BE0  000A 0100 6020 1020 0400 A222 2000 9002
D:817BF0  0001 0400 2000 0200 0801 0000 0040 0802
D:817C00  DDDF BF5E F9BF AFD7 8A73 DF71 9CFF FA5E
D:817C10  8814 18C6 D9EE 35C7 BE6D E0FB 7B42 BAE7
D:817C20  7FFF 5E49 5F5F F2E9 D679 B7AF FB5B FDF3
D:817C30  578E D9EE F7E4 7C79 5A9F 519E CEAE FC6E
D:817C40  F99F FB57 54FD 7DBD D69F FD7D F7B7 FCBF
D:817C50  557A 1C5F FB9B 380E 2FE1 742B AEEE 17E8
D:817C60  9D66 D7FD 7555 BDCF 957F A1EF 529B DFDF
D:817C70  B73A 29AF 98ED 9E7D 2B5E 95B9 ADD6 5AA6
D:817C80  0000 0840 1828 0440 4040 0000 0000 2280
D:817C90  0040 0080 003A 2108 0040 0080 0010 2001
D:817CA0  0002 2401 1010 0100 0000 2012 8214 0000
D:817CB0  8108 0486 0301 0040 0000 0280 0200 2080
D:817CC0  2000 8002 0028 5084 2000 0002 0020 2001
```

# Power Trace: how to save a backtrace

- Set a program BP e.g. at symbol _ms_exit

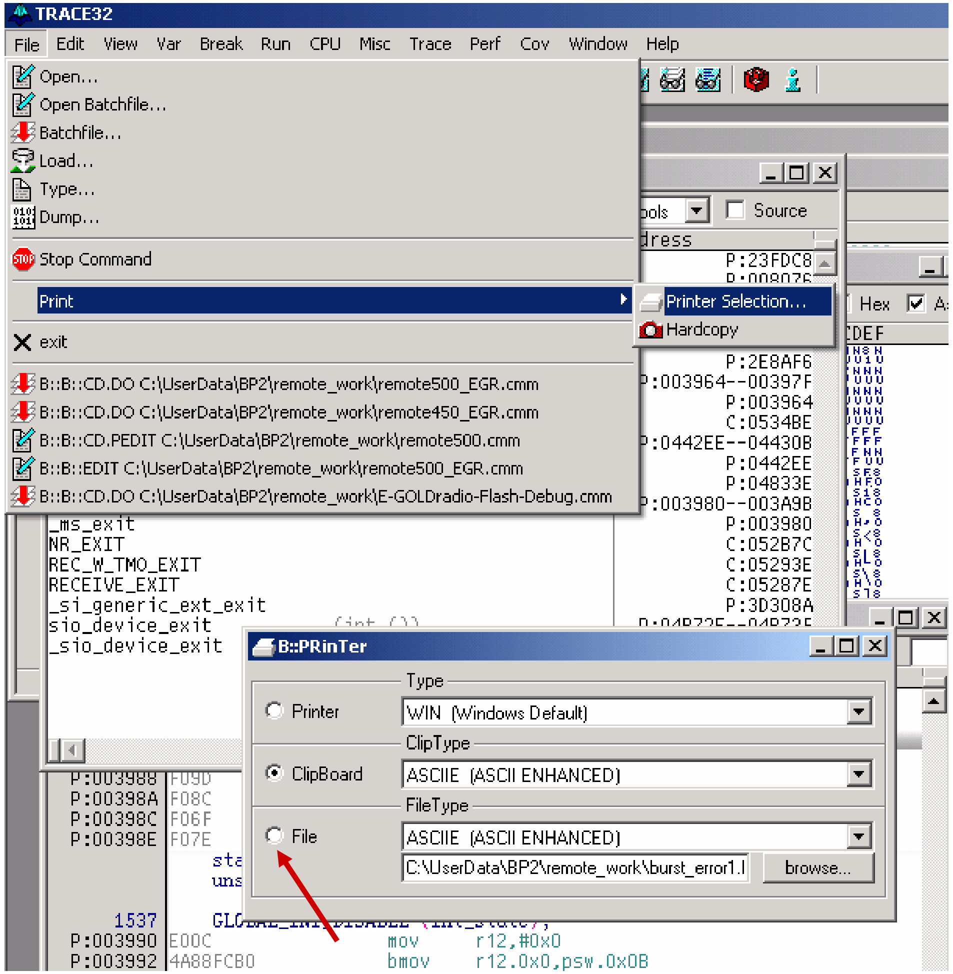


| symbol | type | address |
|---|---|---|
| __doexit | | P:23FDC8 |
| __EXIT | | P:008076 |
| ALLOC_EXIT | | C:052398 |
| ALLOC_NIL_EXIT | | C:0523AC |
| _atexit | | P:23FDA2 |
| _df2_exit | | P:2E8AF6 |
| exit | (void ()) | P:003964--00397F |
| _exit | | P:003964 |
| EXIT | | C:0534BE |
| io_null_exit | (int ()) | P:0442EE--04430B |
| _io_null_exit | | P:0442EE |
| _io_uart_egold_exit | | P:04833E |
| ms_exit | (void ()) | P:003980--003A9B |
| _ms_exit | | P:003980 |
| NR_EXIT | | C:052B7C |
| REC_W_TMO_EXIT | | C:05293E |
| RECEIVE_EXIT | | C:05287E |
| _si_generic_ext_exit | | P:3D308A |
| sio_device_exit | | 72E--04B73F |
| _sio_device_exit | | P:04B72E |

```
P:003988 F09D
P:00398A F08C
P:00398C F06F
P:00398E F07E
                 static char  entered = 0;
                 unsigned short  int_state;

            1537 GLOBAL_INT_DISABLE (int_state
P:003990 E00C                     mov     r12,#
P:003992 4A88FCB0                 bmov    r12.0
P:003996 C4C08000                 mov     [r0+#
P:00399A 48C0                     cmp     r12,#
P:00399C EA20AA39                 jmpa    cc_eq
P:0039A0 D130                     atomic  #0x4
P:0039A2 CC00                     nop
```

# Power Trace: how to save a backtrace (2)

- When the system stops, select the item “File→Print→Printer selection”.
- Select a name with a numeric suffix, so that its number is automatically updated every time you print.


TRACE32
File
Edit
View
Var
Break
Run
CPU
Misc
Trace
Perf
Cov
Window
Help
Open...
Open Batchfile...
Batchfile...
Load...
Type...
Dump...
Stop Command
Print
Printer Selection...
Hardcopy
exit
B::B::CD.DO C:\UserData\BP2\remote_work\remote500_EGR.cmm
B::B::CD.DO C:\UserData\BP2\remote_work\remote450_EGR.cmm
B::B::CD.PEDIT C:\UserData\BP2\remote_work\remote500.cmm
B::B::EDIT C:\UserData\BP2\remote_work\remote500_EGR.cmm
B::B::CD.DO C:\UserData\BP2\remote_work\E-GOLDradio-Flash-Debug.cmm
Source
Hex
B::PRinTer
browse...

B::PRinTer

Type
○ Printer WIN (Windows Default)

ClipType
● ClipBoard ASCIIE (ASCII ENHANCED)

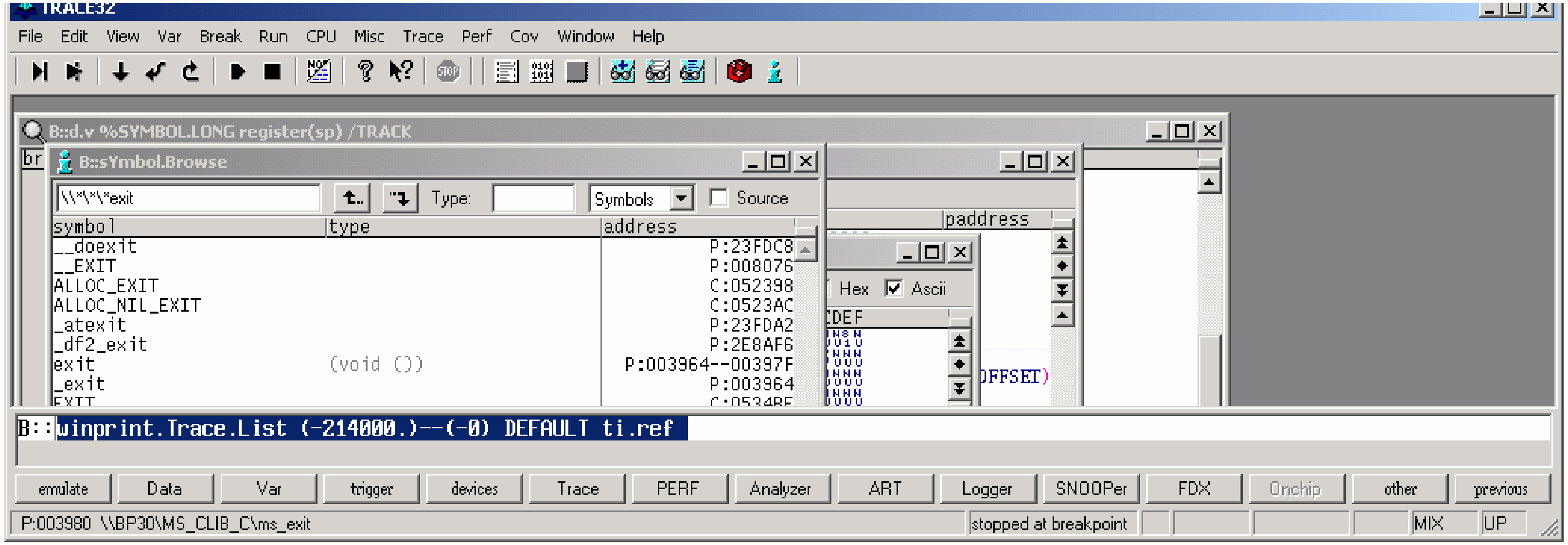
FileType
○ File ASCIIE (ASCII ENHANCED)
C:\UserData\BP2\remote_work\burst_error1.l

# Power Trace: how to save a backtrace (3)

- Then type the following command

  winprint.Trace.List (-<index>.)--(-0) DEFAULT ti.ref

- Pay attention to the “minus” (the index is negative) and to the “dot” at the end if you use the decimal format.



TRACE32
File Edit View Var Break Run CPU Misc Trace Perf Cov Window Help
B::d.v %SYMBOL.LONG register(sp) /TRACK
B::sYmbol.Browse
Type:
Symbols
□ Source
paddress
Hex ☑ Ascii
emulate Data Var trigger devices Trace PERF Analyzer ART Logger SNOOPer FDX Onchip other previous
stopped at breakpoint
MIX
UP

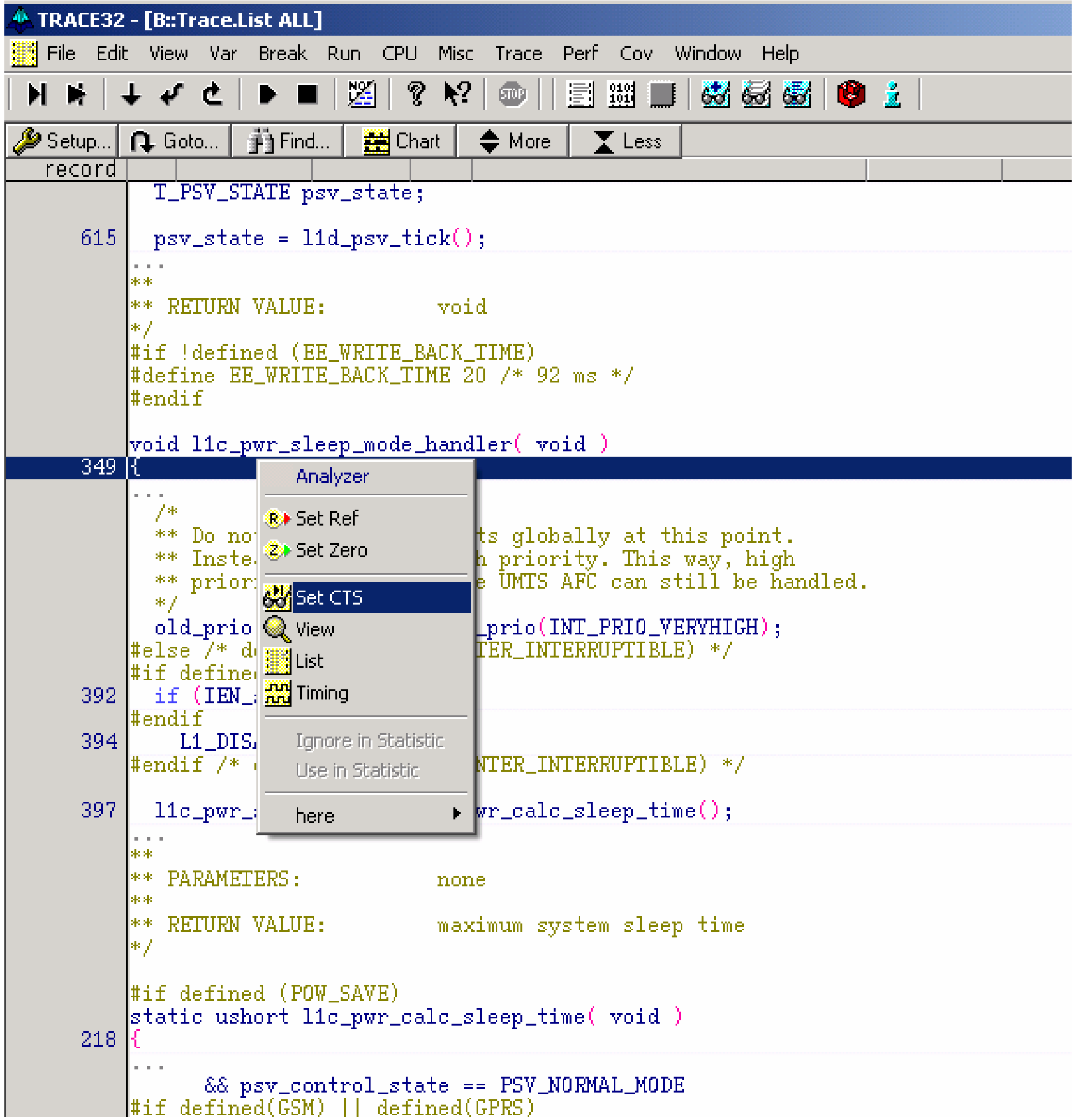
```
\\*\*\*exit
symbol          type          address
__doexit                      P:23FDC8
__EXIT                        P:008076
ALLOC_EXIT                    C:052398
ALLOC_NIL_EXIT                C:0523AC
_atexit                       P:23FDA2
_df2_exit                     P:2E8AF6
exit            (void ())     P:003964--00397F
_exit                         P:003964
EXIT                          C:0534BF
```

```
B::winprint.Trace.List (-214000.)--(-0) DEFAULT ti.ref
```

```
P:003980 \\BP30\MS_CLIB_C\ms_exit
```

# Power Trace: how to use a backtrace

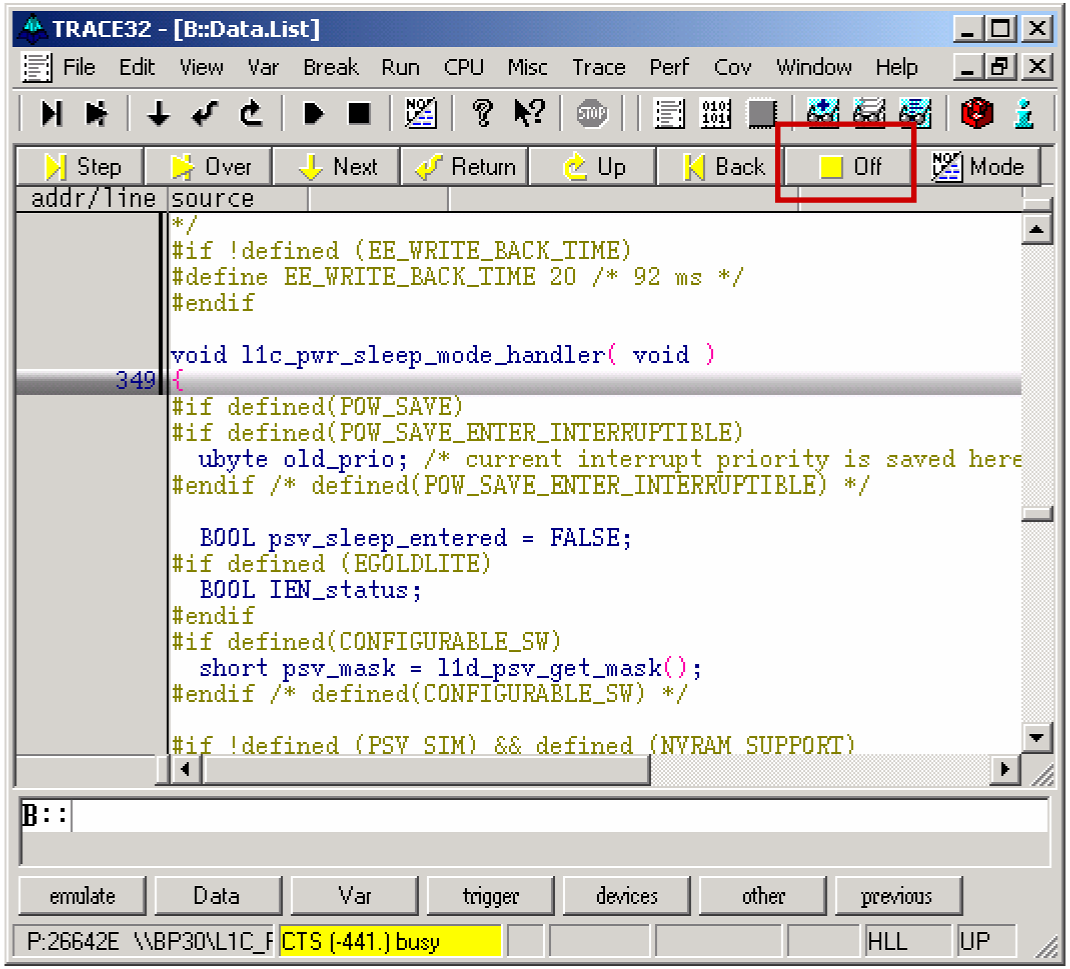
- You can simulate the recorded execution in a step-by-step fashion. The contents of the variables and the registers are updated but not fully reliable.
- Simply select the point where you would like to start, press right-click and SET CTS



```
  T_PSV_STATE psv_state;

  psv_state = l1d_psv_tick();
...
**
** RETURN VALUE:        void
*/
#if !defined (EE_WRITE_BACK_TIME)
#define EE_WRITE_BACK_TIME 20 /* 92 ms */
#endif

void l1c_pwr_sleep_mode_handler( void )
{
...
  /*
  ** Do no                   ts globally at this point.
  ** Inste                   h priority. This way, high
  ** prior                   e UMTS AFC can still be handled.
  */
  old_prio d                 _prio(INT_PRIO_VERYHIGH);
#else /* d                   TER_INTERRUPTIBLE) */
#if define
  if (IEN_
#endif
    L1_DIS
#endif /*                    NTER_INTERRUPTIBLE) */

  l1c_pwr_                   wr_calc_sleep_time();
...
**
** PARAMETERS:          none
**
** RETURN VALUE:        maximum system sleep time
*/

#if defined (POW_SAVE)
static ushort l1c_pwr_calc_sleep_time( void )
{
...
      && psv_control_state == PSV_NORMAL_MODE
#if defined(GSM) || defined(GPRS)
```

# Power Trace: how to use a backtrace (2)

- To start the step-by-step execution, use View/List Source.
- Pay attention that, if you select the HLL source only mode, you can miss passages to non-symbolic functions.  Besides, the interruptions by LISRs are not handled.
- Press Off to exit CTS mode and possibly change the starting point.



```
*/
#if !defined (EE_WRITE_BACK_TIME)
#define EE_WRITE_BACK_TIME 20 /* 92 ms */
#endif

void l1c_pwr_sleep_mode_handler( void )
349 {
#if defined(POW_SAVE)
#if defined(POW_SAVE_ENTER_INTERRUPTIBLE)
  ubyte old_prio; /* current interrupt priority is saved here
#endif /* defined(POW_SAVE_ENTER_INTERRUPTIBLE) */

  BOOL psv_sleep_entered = FALSE;
#if defined (EGOLDLITE)
  BOOL IEN_status;
#endif
#if defined(CONFIGURABLE_SW)
  short psv_mask = l1d_psv_get_mask();
#endif /* defined(CONFIGURABLE_SW) */

#if !defined (PSV_SIM) && defined (NVRAM_SUPPORT)
```

# Tips on the Power Trace: effects of the page mode

- Unfortunately the trace.list output of the Power Trace is fully correct only **when the page mode is disabled**.
- If the page mode is active, as it happens in our released SW, occasionally (at most 3 out of 4) consecutive *opcode fetch* instructions can be lost.
- Therefore, to get a complete and reliable backtrace enter (via Mobile Analyser of AT command window) the command: **AT+XL1SET="PAGEMODE_OFF"**
- To be able to explore rapidly and effectively the backtrace to look for unexpected behaviors, you can open it with Source Insight and look for the references of the string “calls”.
- This holds of course if the page mode is disabled. Otherwise a few call instructions could be missing.
  - Pay attention in general that ASM instructions can be only fetched due to proximity and not actually executed.

# Parsing a backtrace

```
 |   + calls   0x9,0xC1D4         ; _mac_handle_rlc_data_block
 |   | calls   0x0,0xA842         ; _PS_cccl_0362_04
 |   | calls   0x15,0x2F7C        ; retrieve_timer
|   + calls   0x5,0x242A         ; _wait_sem
 |   | calls   0x5,0x3274         ; WAIT_SEM
...
 |   + calls   0x9,0xC170         ; _mac_handle_persistence_level_change
 |   | calls   0x0,0x87A4         ; _PS_cccl_0065_03
 |   + calls   0x0B,0xF850        ; _mac_tm_get_tbf_mode
 |   | calls   0x0,0xC3D2         ; ms_exit
```

```
 switch( mac_tm_get_tbf_mode() )
 {
 case AT_EGPRS:...
     break;
 case AT_GPRS:
     retrieval_basket.mSlotCl = data_base.ms_multislot_class_gprs;
     break;
 default: ...
}
 if( retrieval_basket.mSlotCl == NO_MULTI_SLOT_CAPABILITY) /*zero*/
  ms_except(MS_UNKNOWN_ERROR);
```

# Parsing a backtrace (2)

- Since the element `data_base.ms_multislot_class_gprs` has a fixed value for the whole system life, an unwanted memory overwriting should have occurred (casually to 0).
- We look for operations on the element, whose offset is 0x20:

```
D:837AAA opfetch   00 \\BP30\MAC_IM_C\_data_base+0x20
```

- If we are lucky and the overwriting happened within the sampled interval of ca 2 seconds, we will find (“may” if we have not disabled the page mode) for instance:

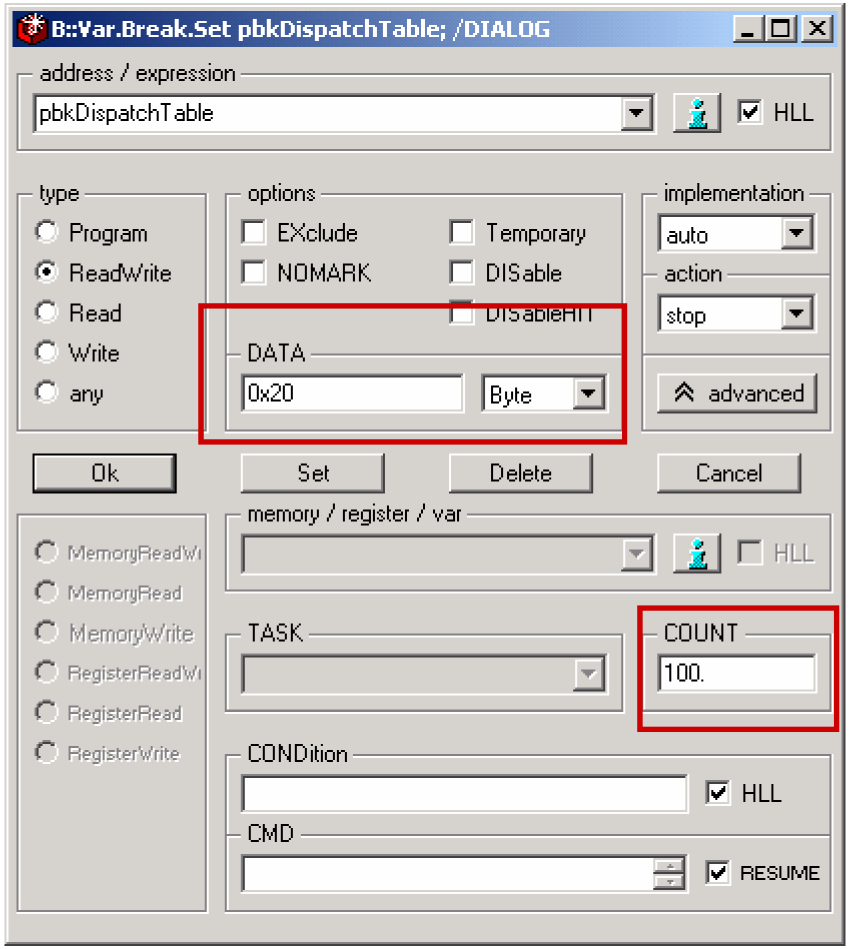
```
D:837AAA wr-word 0000 \\BP30\MAC_IM_C\_data_base+0x20
```

- In this case the cause was an overrun of the user stack of the l1f_tdma_isr process which interrupted the MAC processing;
- The fix was to increase its user stack in file os166.con to 1500:

```
%PRI_PROC l1f_tdma_isr,                     C,   1500, 128, 1
```

# Note on Write BP

- Write BP can be useful to debug the above-mentioned type of errors, although the overwriting can affect different memory areas.
- If you want to break at writing of a specific value, you can rely on conditioned BP, but this implies <u>high delays</u> for the Lautebach to verify the condition:
  - You have to work with sev=0, otherwise the system crashes
  - Same problem with counters
  - Shortcut: set a BP specifying the data → no delay



B::Var.Break.Set pbkDispatchTable; /DIALOG

address / expression: pbkDispatchTable ☑ HLL

type: ○ Program ● ReadWrite ○ Read ○ Write ○ any

options: □ EXclude □ Temporary □ NOMARK □ DISable □ DISableHIT

implementation: auto

action: stop

DATA: 0x20 Byte

memory / register / var: ○ MemoryReadWrite ○ MemoryRead ○ MemoryWrite ○ RegisterReadWrite ○ RegisterRead ○ RegisterWrite □ HLL

TASK:

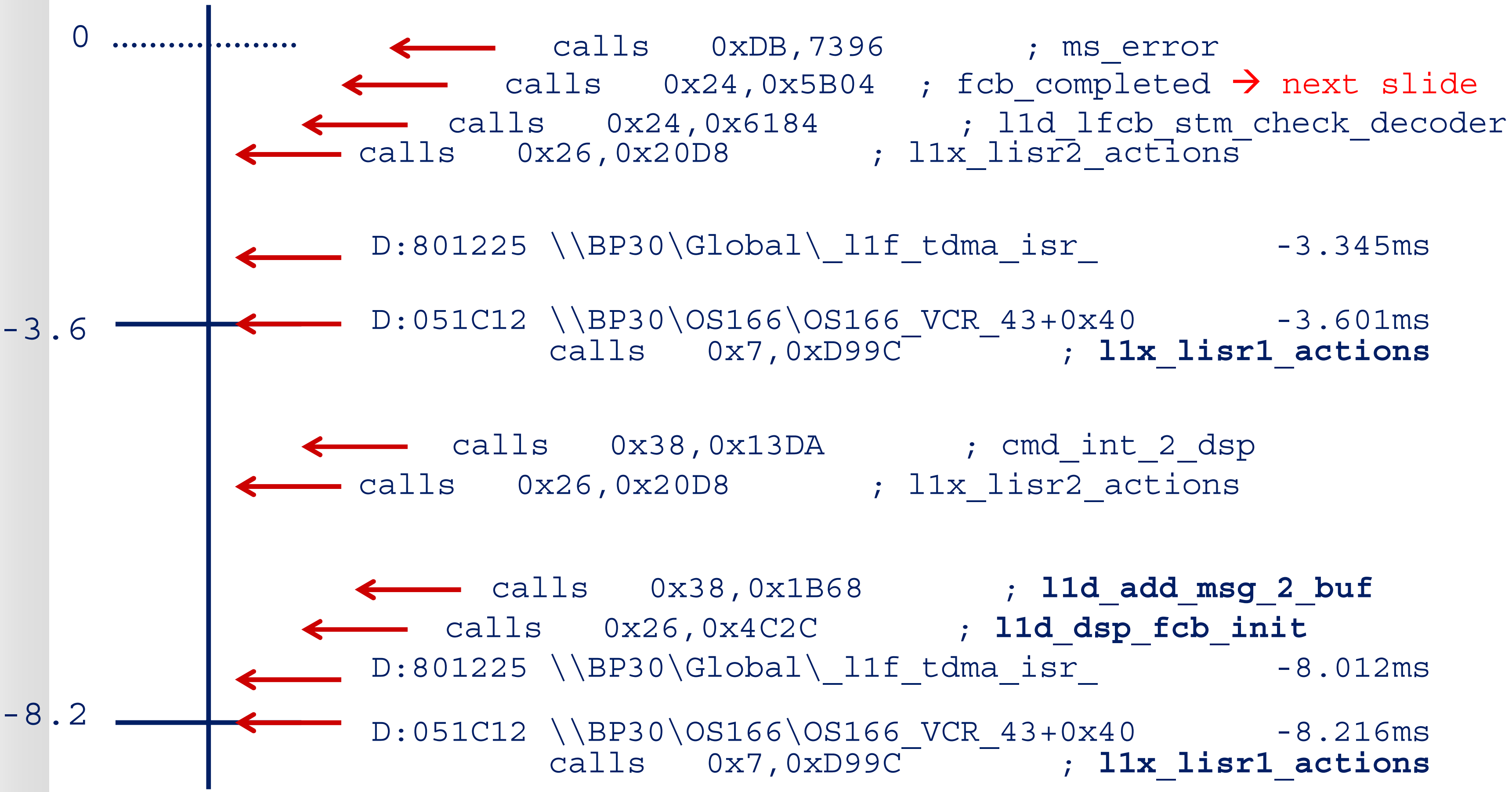
COUNT: 100.

CONDition: ☑ HLL

CMD: ☑ RESUME

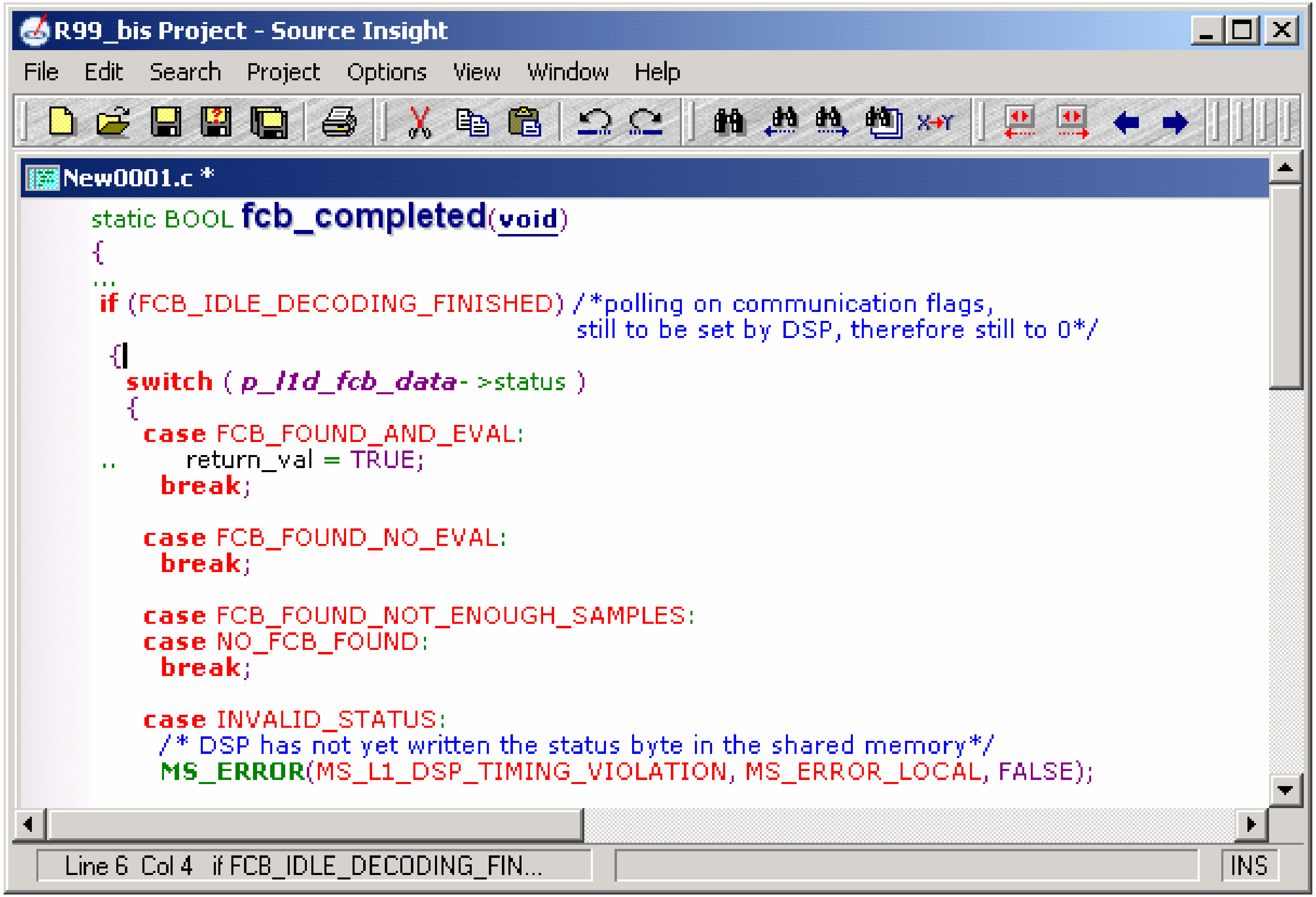
# Parsing a backtrace: timings of function calls

- You can collect information on the timings of the functions.



```
                calls    0xDB,7396         ; ms_error
            calls    0x24,0x5B04   ; fcb_completed → next slide
         calls    0x24,0x6184         ; l1d_lfcb_stm_check_decoder
     calls    0x26,0x20D8         ; l1x_lisr2_actions

     D:801225 \\BP30\Global\_l1f_tdma_isr_          -3.345ms

     D:051C12 \\BP30\OS166\OS166_VCR_43+0x40        -3.601ms
              calls    0x7,0xD99C          ; l1x_lisr1_actions

         calls    0x38,0x13DA         ; cmd_int_2_dsp
     calls    0x26,0x20D8         ; l1x_lisr2_actions

            calls    0x38,0x1B68         ; l1d_add_msg_2_buf
         calls    0x26,0x4C2C         ; l1d_dsp_fcb_init
     D:801225 \\BP30\Global\_l1f_tdma_isr_          -8.012ms

     D:051C12 \\BP30\OS166\OS166_VCR_43+0x40        -8.216ms
              calls    0x7,0xD99C          ; l1x_lisr1_actions
```

# Parsing a backtrace: timings of function calls (2)


R99_bis Project - Source Insight
File
Edit
Search
Project
Options
View
Window
Help
New0001.c *
Line 6 Col 4 if FCB_IDLE_DECODING_FIN...
INS

```
static BOOL fcb_completed(void)
{
...
 if (FCB_IDLE_DECODING_FINISHED) /*polling on communication flags,
                                   still to be set by DSP, therefore still to 0*/
  {
   switch ( p_l1d_fcb_data- >status )
   {
     case FCB_FOUND_AND_EVAL:
..        return_val = TRUE;
       break;

     case FCB_FOUND_NO_EVAL:
       break;

     case FCB_FOUND_NOT_ENOUGH_SAMPLES:
     case NO_FCB_FOUND:
       break;

     case INVALID_STATUS:
      /* DSP has not yet written the status byte in the shared memory*/
      MS_ERROR(MS_L1_DSP_TIMING_VIOLATION, MS_ERROR_LOCAL, FALSE);
```