SCM1612
Wi-Fi 6 and BLE 5 Low-Power SoC

Debug Guide

Revision 0.1
Date 2024-4-17

Contact Information
Senscomm Semiconductor (www.senscomm.com)
Room 303, International Building, West 2 Suzhou Avenue,
SIP, Suzhou, China
For sales or technical support, please send email to
info@senscomm.com

Disclaimer and Notice

This document is provided on an “as-is” basis only. Senscomm reserves the right to make corrections, improvements and other changes to it or any specification contained herein without further notice.
All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.
All third party’s information in this document is provided as is with NO warranties to its authenticity and accuracy.
All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners and are hereby acknowledged.

Version History

Version	Date	Description
0.1	2024-4-17	Initial draft

Table of Contents

1 Overview

This guide provides essential debugging techniques for common issues encountered during application development. It serves both as an instructional tool and a quick reference for troubleshooting.

1.1 Coverage

This document focuses on addressing specific types of issues:

- Core exception
- Assertion failure
- Stack overrun
- Memory leak

Additional problems such as deadlocks may arise; however, the tactics and tools discussed herein are applicable to these scenarios as well. For in-depth debugging, a JTAG debugger like AICE-MICRO in conjunction with OpenOCD is recommended. Detailed setup instructions are available in the SCM1612_SDK_Getting_Started_Guide.

1.2 Logging Mechanisms

The SCM1612 SDK provides two primary methods for logging via the UART console:

Function	Header file	Description
int printf (const char *format, ...)	include/stdio.h	- Directly sends a byte string to the UART port. - Not suitable for ISR context as it relies on OS-dependent wait functions. - Introduces significant runtime overhead. - Excluded from core dumps (See section 1.3).
Int printk (const char *format, ...)	Include/hal/console.h	- Outputs byte strings to a configurable internal console buffer.

		- Usable in both thread and ISR contexts. - The `dmesg` CLI command displays the entire console buffer; subsequent calls clear the buffer. - `dmesg start` enables real-time logging with minimal performance impact. - Included in core dumps (Refer to section 1.3).

For effective troubleshooting, utilize printf or printk to isolate issues quickly, minimizing time and effort in subsequent steps.

1.3 Utilizing the 'ps' Command

The `ps` command provides insights into the system's active tasks, helping identify potential issues related to task management and resource utilization.

Command Output:

```
$ ps
 PID   PR  STWM  S  %CPU+     TIME+   TASK
   1    3   480  X    2.3    0:00:01 init                          (0x21bda0-0x21cd90, 0x21c85c)
   2    0   966  R   78.7    0:00:45 idle                          (0x20fc04-0x210c00, 0x210b1c)
   3    5   199  B   18.9    0:00:10 ksofttimerd                   (0x20f780-0x20fb70, 0x20fa9c)
   4    3   683  B    0.0    0:00:00 knetd                         (0x21d0b0-0x21dca0, 0x21db5c)
   5    3   571  B    0.0    0:00:00 scm2020-wlan fast taskq       (0x21f710-0x220100, 0x21fffc)
   7    3   303  B    0.0    0:00:00 rt_msg                       (0x221600-0x221bf0, 0x221acc)
   6    3   315  B    0.0    0:00:00 knet80211d/wlan0             (0x220d30-0x221320, 0x22121c)
   8    7    48  B    0.0    0:00:00 ll                            (0x2222e0-0x222550, 0x22244c)
$
```

The command `ps` shows information as described in the below table.

Column	Meaning	Description
PID	Task identifier	Assigned by the OS.
PR	Task priority	Refer to SCM1612_CMSIS-FreeRTOS_API_Guide.
STWM	Stack watermark	Indicating the minimum free stack space observed.
S	Task state	'X': Running 'R': Ready

		‘B’: Blocked ‘S’: Suspended ‘D’: Deleted ‘I’ : Invalid
%CPU+	Relative CPU utilization	CPU utilization in percentage (Relative)
TIME+	Absolute CPU time consumed by the task.	CPU utilization in time (Absolute)
TASK	Task name	Given at creation
Stack	(Start-End, sp)	See below.

This information provides a snapshot of the current system state, helping to diagnose issues effectively.

The three hexadecimal numbers in parentheses represent the start address, end address, and last recorded stack pointer of the task's stack, respectively.

For instance, the 'init' task has its stack located from 0x21bda0 to 0x21cd90, with the last recorded stack pointer at 0x21c85c.

The 'STWM' (Stack Watermark) column reflects the smallest amount of free space (in units of uint32) that was available on the task's stack during its most resource-intensive operation. Lower values in this column suggest a higher risk of a stack overrun.

The term 'last recorded stack pointer' refers to the stack pointer value saved by the OS—FreeRTOS in this case—during context switches, such as between tasks or from a task to an interrupt. The 'ps' command displays this value, providing insights into the stack usage patterns of each task.

Although the 'last recorded stack pointer' does not provide a real-time stack pointer value, especially for a task currently running, it is instrumental in debugging. Analyzing the memory content of the last known state of the stack can reveal crucial information about the task's status before a switch or error occurred.

To further understand the stack's state at the time of the last context switch for the 'init' task, we can examine the memory as follows:

```
$ hexdump 0x21c85c 64
0x0021c850: .... ....  .... ....  .... ....  4ac7 1000    ............J
0x0021c860: 4000 0000  0000 0000  a5a5 a5a5  0ec8 1000    @
0x0021c870: a500 0000  a0bd 2100  90cd 2100  f02c 2200          !   !  ,"
```

```
0x0021c880: 2800 0000  0000 0000  708e 0980  0800 0000   (      p
0x0021c890: d000 0000  0000 0000  082e 2200  .... ....           ." ....
$
```

Note that the hexdump command will display the memory content as bytes, maintaining the byte order. In contrast, the read command presents the data in word units, each consisting of 32 bits. This format is particularly useful for understanding the structure of data as processed by the system.

Example output from the read command is as follows:

```
$ read 0x21c85c 16
[0x21c85c] 0x0010c74a
[0x21c860] 0x00000040
[0x21c864] 0x00000000
[0x21c868] 0xa5a5a5a5
[0x21c86c] 0x0010c80e
[0x21c870] 0x000000a5
[0x21c874] 0x0021bda0
[0x21c878] 0x0021cd90
[0x21c87c] 0x00222cf0
[0x21c880] 0x00000028
[0x21c884] 0x00000000
[0x21c888] 0x80098e70
[0x21c88c] 0x00000008
[0x21c890] 0x000000d0
[0x21c894] 0x00000000
[0x21c898] 0x00222e08
$
```

This output provides a clear view of how data is structured in the memory at specific addresses, which is critical for debugging and tracing issues in the system.

Understanding how to deduce a call flow from a stack dump involves analyzing such memory content to track back through the function calls that led to the current state, a process that will be elaborated on later in this document. This analysis helps identify the sequence of events or errors leading up to an exception or crash, thereby aiding in pinpointing the exact source of a problem.

1.4 Core dump

A core dump is generated automatically when a trap exception or an assertion failure occurs. It provides a detailed snapshot of the memory and processor state at the time of the exception, which is crucial for debugging. The core dump is output directly to the UART console, allowing developers to analyze the conditions that led to the failure.

Example of a Core Dump:

```
---------------------------------------------------------------------------------------------------------
WISE 2018.02+ riscv32-elf-gcc.gnu (2022-02-07_nds32le-elf-mculib-v5-86807094a2f) 10.3.0
[000361.560214] BOARD: SCM2010 QFN40 EVB V1.0
[000361.560368] VFS: filesystem devfs mounted onto /dev
[000361.562230] PINCTRL: pin controller scm2010,pinctrl registered
[000361.562440] GPIO: scm2010,pinctrl registered as /dev/gpio
[000361.562728] atcwdt: @0xf1300000, clk=32768
[000361.562936] WDT: atcwdt registered as /dev/watchdog
[000361.563279] PINCTRL: atcspi200-xip/cs request pin 11
[000361.563471] PINCTRL: atcspi200-xip/clk request pin 12
[000361.563636] PINCTRL: atcspi200-xip/mosi request pin 13
[000361.563808] PINCTRL: atcspi200-xip/miso request pin 14
[000361.563993] PINCTRL: atcspi200-xip/hold request pin 9
[000361.564173] PINCTRL: atcspi200-xip/wp request pin 10
[000361.564667] EFUSE: efuse-scm2010 registered as /dev/efuse
[000361.564799] trng version : 5e5e0010
[000361.564962] TRNG: trng registered as /dev/trng
[000361.565084] pke version : 0x5e5e0010
[000361.607174] Use fixed MAC address: 64.f9.47.f0.01.20
[000361.608169] 261 usec elapsed in downloading MAC FW.
[000362.902695] Wlan PM ctx init done
[000362.904031] UART: atcuart.0 registered as /dev/ttyS0
[000362.904346] CONSOLE: add /dev/ttyS0
[000362.904664] UART: atcuart.1 registered as /dev/ttyS1
[000362.904788] SOC: SCM2010
[000362.905306] Use fixed BLE public address: 01.02.03.04.05.06
[000362.907563] ble phy init 35
[000362.907711] PM feature : 0x1d
[000362.908487] PM LS : 6500+1120=7620
[000362.908655] PM SL : 6500+7100=13600
[000362.908817] PM DS : 11500+12800=24300
[000362.908990] PM HB : 1655000+60000=1715000
[000362.909248] PINCTRL: atcuart.0/txd request pin 22
[000362.909412] PINCTRL: atcuart.0/rxd request pin 21
[000377.371510] Unhandled Trap : Ilegal instruction (mcause = 0x2), mepc = 0xc0000000
[000377.379216] EPC:c0000000
[000377.381940] MSTATUS:00001880
[000377.385019] MXSTATUS:00000080
[000377.388188] MTVAL:00003003
[000377.391092] A0:c0000000 A1:00000000 A2:00000010 A3:00000001 A4:00000002 A5:ffffffff A6:00000008
A7:00210f37
[000377.401165] T0:8008e6f8 T1:001096f0 T2:00000000 T3:00210f2c T4:000001b0 T5:8008e71c T6:00000008
[000377.410197] S0:00000002 S1:0020d488 S2:ffffffff S3:00000001 S4:8010a220 S5:a5a5a5a5 S6:a5a5a5a5
S7:a5a5a5a5
[000377.420373] S8:a5a5a5a5 S9:a5a5a5a5 S10:a5a5a5a5 S11:a5a5a5a5
[000377.426377] FP:00000002 RA:8008e6fe
[000377.429989] sp:    0021cb6c
[000377.432977] IRQ stack:
[000377.435443]   base: 0021b590
[000377.438433]   size: 00000800
[000377.441429] ERROR: Stack pointer is not within the interrupt stack
[000377.447766] 0021b580: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.456531] 0021b5a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.465294] 0021b5c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.474056] 0021b5e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.482819] 0021b600: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.491582] 0021b620: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.500345] 0021b640: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.509108] 0021b660: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.517871] 0021b680: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.526634] 0021b6a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.535397] 0021b6c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.544160] 0021b6e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.552923] 0021b700: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.561686] 0021b720: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.570448] 0021b740: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.579211] 0021b760: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.587974] 0021b780: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.596737] 0021b7a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.605500] 0021b7c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.614263] 0021b7e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

```
[000377.623026] 0021b800: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.631789] 0021b820: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.640551] 0021b840: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.649314] 0021b860: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.658077] 0021b880: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.666840] 0021b8a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.675603] 0021b8c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.684366] 0021b8e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.693129] 0021b900: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.701892] 0021b920: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.710655] 0021b940: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.719417] 0021b960: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.728180] 0021b980: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.736943] 0021b9a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.745706] 0021b9c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.754469] 0021b9e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.763231] 0021ba00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.771994] 0021ba20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.780757] 0021ba40: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.789520] 0021ba60: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.798283] 0021ba80: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.807046] 0021baa0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.815809] 0021bac0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.824572] 0021bae0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.833334] 0021bb00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.842097] 0021bb20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.850860] 0021bb40: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.859623] 0021bb60: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.868386] 0021bb80: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.877149] 0021bba0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.885911] 0021bbc0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.894674] 0021bbe0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.903437] 0021bc00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.912200] 0021bc20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.920963] 0021bc40: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.929726] 0021bc60: 00000000 00000000 00000000 00000000 0020c310 00203bf4 0020c9b8 80100c3a
[000377.938604] 0021bc80: 0020c310 00203bf4 0020c9b8 00000000 a5a5a5a5 a5a5a5a5 a5a5a5a5 0021bd2c
[000377.947582] 0021bca0: 00000005 00000000 00000100 8010b5ec 0010d29c 0020f780 0020fb80 8008e8e8
[000377.956508] 0021bcc0: a5a5a5a5 8000bad4 0020fb80 00000084 a5a5a5a5 8000bad4 00000001 ffffffff
[000377.965486] 0021bce0: 80080020 0020d5f8 0020d5f8 8008eaa6 a5a5a5a5 0020fb80 0020f780 00000380
[000377.974475] 0021bd00: 80080020 0020d5f8 0020d5f8 0021cdc8 00000000 ffffffff 00222c20 0010a9c6
[000377.983442] 0021bd20: 80122218 00000000 e0871aa7 00215798 00000004 0020dea8 00223500 80100878
[000377.992394] 0021bd40: 000048e2 00203bf4 0010d8d4 00206ef4 00000009 00000000 0010b70e 0000001c
[000378.001283] 0021bd60: 167be6a8 00000000 002228e0 000002a0 00003d2a 00000009 00206ef4 0010d8d4
[000378.010177] 0021bd80: 00203bf4 000048e2 00000000 00000414 00000000 80001010 00000000 00000000
[000378.019024] sp:    0021cbfc
[000378.022018] User stack:
[000378.024568]  base: 0021bda0
[000378.027562]  size: 00000ff0
[000378.030556] 0021cbe0: 002065d4 c0000000 00000002 ffffffff 0020d488 00000002 8008e6fe c0000000
[000378.039515] 0021cc00: 00000080 00003003 00206658 8008e6fe 8008e6f8 001096f0 00000000 00000002
[000378.048420] 0021cc20: 0020d488 c0000000 00000000 00000010 00000001 00000002 ffffffff 00000008
[000378.057283] 0021cc40: 00210f37 ffffffff 00000001 8010a220 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5
[000378.066287] 0021cc60: a5a5a5a5 a5a5a5a5 a5a5a5a5 00210f2c 000001b0 8008e71c 00000008 00001880
[000378.075244] 0021cc80: a5a5a5a5 a5a5a5a5 8010a220 c0000000 ffffffff 0020d488 00000002 8008e742
[000378.084249] 0021cca0: ffffffff 0020d488 00000002 800981cc e2a0c53d 00210d70 0020d3f8 00210f37
[000378.093222] 0021ccc0: 00210f20 00210f2d 00000000 00000000 00000000 00000000 00000000 00000000
[000378.102032] 0021cce0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000378.110795] 0021cd00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000378.119558] 0021cd20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000378.128321] 0021cd40: a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 80113eac 00210f20 800985b2
[000378.137367] 0021cd60: a5a5a5a5 0020d5fc 00202c04 800985c6 a5a5a5a5 0020d5fc 00202c04 800ff356
[000378.146382] 0021cd80: a5a5a5a5 a5a5a5a5 a5a5a5a5 00000000 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5
[000378.159763]  PID   PR  STWM  S   TASK
[000378.167018]    1    3   531  X   init                          (0x21bda0-0x21cd90, 0x21c6f8)
[000378.175996]    2    0   958  R   idle                          (0x20fc04-0x210c00, 0x210b1c)
[000378.184966]    4    3   683  B   knetd                         (0x21d0b0-0x21dca0, 0x21db5c)
[000378.193936]    3    5   199  B   ksofttimerd                    (0x20f780-0x20fb70, 0x20fa9c)
[000378.202906]    8    7    48  B   ll                            (0x2223f0-0x222660, 0x22255c)
[000378.211871]    5    3   571  B   scm2020-wlan fast taskq          (0x21f710-0x220100, 0x21fffc)
[000378.220841]    7    3   303  B   rt_msg                        (0x221600-0x221bf0, 0x221acc)
[000378.229811]    6    3   315  B   knet80211d/wlan0               (0x220d30-0x221320, 0x22121c)
```

A core dump is composed of several sections, each providing critical information about the system's state at the time of the failure. Below is an outline of the typical components of a core dump, presented in the order they typically appear:

1.4.1 Console buffer content

At the beginning of a core dump, the console buffer content is flushed and displayed. This content provides a log of system activities leading up to the core dump, captured through the `printk` function if utilized as suggested for tracing call flows. This aids in the postmortem debugging process by offering insights into the events and system states prior to the failure.

Example Console Buffer Output:

```
WISE 2018.02+ riscv32-elf-gcc.gnu (2022-02-07_nds32le-elf-mculib-v5-86807094a2f) 10.3.0
[000361.560214] BOARD: SCM2010 QFN40 EVB V1.0
[000361.560368] VFS: filesystem devfs mounted onto /dev
[000361.562230] PINCTRL: pin controller scm2010,pinctrl registered
[000361.562440] GPIO: scm2010,pinctrl registered as /dev/gpio
[000361.562728] atcwdt: @0xf1300000, clk=32768
[000361.562936] WDT: atcwdt registered as /dev/watchdog
[000361.563279] PINCTRL: atcspi200-xip/cs request pin 11
[000361.563471] PINCTRL: atcspi200-xip/clk request pin 12
[000361.563636] PINCTRL: atcspi200-xip/mosi request pin 13
[000361.563808] PINCTRL: atcspi200-xip/miso request pin 14
[000361.563993] PINCTRL: atcspi200-xip/hold request pin 9
[000361.564173] PINCTRL: atcspi200-xip/wp request pin 10
[000361.564667] EFUSE: efuse-scm2010 registered as /dev/efuse
[000361.564799] trng version : 5e5e0010
[000361.564962] TRNG: trng registered as /dev/trng
[000361.565084] pke version : 0x5e5e0010
[000361.607174] Use fixed MAC address: 64.f9.47.f0.01.20
[000361.608169] 261 usec elapsed in downloading MAC FW.
[000362.902695] Wlan PM ctx init done
[000362.904031] UART: atcuart.0 registered as /dev/ttyS0
[000362.904346] CONSOLE: add /dev/ttyS0
[000362.904664] UART: atcuart.1 registered as /dev/ttyS1
[000362.904788] SOC: SCM2010
[000362.905306] Use fixed BLE public address: 01.02.03.04.05.06
[000362.907563] ble phy init 35
[000362.907711] PM feature : 0x1d
[000362.908487] PM LS : 6500+1120=7620
[000362.908655] PM SL : 6500+7100=13600
[000362.908817] PM DS : 11500+12800=24300
[000362.908990] PM HB : 1655000+60000=1715000
[000362.909248] PINCTRL: atcuart.0/txd request pin 22
[000362.909412] PINCTRL: atcuart.0/rxd request pin 21
```

1.4.2 Reason of core dump

The reason for the core dump is explicitly detailed in the dump, indicating whether it resulted from a trap exception or an assertion failure. Specific error

descriptions, along with relevant codes from registers like `mcause` and `mepc`, are displayed to clarify the cause of the dump.

Example Reason Output:

```
-------------------------------------------------------------------------------------------------------------
[000377.371510] Unhandled Trap : Ilegal instruction (mcause = 0x2), mepc = 0xc0000000
-------------------------------------------------------------------------------------------------------------
```

1.4.3 Dump of core registers

This section provides a comprehensive list of the General Purpose Registers (GPRs) and Control Status Registers (CSRs) at the time of the crash. This data is crucial for debugging as it helps recreate the state of the application at the point of failure.

Example Reason Output:

```
-------------------------------------------------------------------------------------------------------------
[000377.379216] EPC:c0000000
[000377.381940] MSTATUS:00001880
[000377.385019] MXSTATUS:00000080
[000377.388188] MTVAL:00003003
[000377.391092] A0:c0000000 A1:00000000 A2:00000010 A3:00000001 A4:00000002 A5:ffffffff A6:00000008
A7:00210f37
[000377.401165] T0:8008e6f8 T1:001096f0 T2:00000000 T3:00210f2c T4:000001b0 T5:8008e71c T6:00000008
[000377.410197] S0:00000002 S1:0020d488 S2:ffffffff S3:00000001 S4:8010a220 S5:a5a5a5a5 S6:a5a5a5a5
S7:a5a5a5a5
[000377.420373] S8:a5a5a5a5 S9:a5a5a5a5 S10:a5a5a5a5 S11:a5a5a5a5
[000377.426377] FP:00000002 RA:8008e6fe
-------------------------------------------------------------------------------------------------------------
```

1.4.4 Dump of Interrupt Stack

This segment contains the contents of the interrupt stack. Even if the stack pointer (sp) is not within the interrupt stack, its complete contents are dumped for thorough analysis.

Example Reason Output:

```
-------------------------------------------------------------------------------------------------------------
[000377.429989] sp:    0021cb6c
[000377.432977] IRQ stack:
[000377.435443]   base: 0021b590
[000377.438433]   size: 00000800
[000377.441429] ERROR: Stack pointer is not within the interrupt stack
[000377.447766] 0021b580: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.456531] 0021b5a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.465294] 0021b5c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.474056] 0021b5e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.482819] 0021b600: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.491582] 0021b620: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.500345] 0021b640: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.509108] 0021b660: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.517871] 0021b680: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

```
[000377.526634] 0021b6a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.535397] 0021b6c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.544160] 0021b6e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.552923] 0021b700: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.561686] 0021b720: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.570448] 0021b740: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.579211] 0021b760: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.587974] 0021b780: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.596737] 0021b7a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.605500] 0021b7c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.614263] 0021b7e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.623026] 0021b800: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.631789] 0021b820: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.640551] 0021b840: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.649314] 0021b860: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.658077] 0021b880: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.666840] 0021b8a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.675603] 0021b8c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.684366] 0021b8e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.693129] 0021b900: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.701892] 0021b920: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.710655] 0021b940: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.719417] 0021b960: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.728180] 0021b980: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.736943] 0021b9a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.745706] 0021b9c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.754469] 0021b9e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.763231] 0021ba00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.771994] 0021ba20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.780757] 0021ba40: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.789520] 0021ba60: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.798283] 0021ba80: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.807046] 0021baa0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.815809] 0021bac0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.824572] 0021bae0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.833334] 0021bb00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.842097] 0021bb20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.850860] 0021bb40: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.859623] 0021bb60: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.868386] 0021bb80: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.877149] 0021bba0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.885911] 0021bbc0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.894674] 0021bbe0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.903437] 0021bc00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.912200] 0021bc20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.920963] 0021bc40: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000377.929726] 0021bc60: 00000000 00000000 00000000 00000000 0020c310 00203bf4 0020c9b8 80100c3a
[000377.938604] 0021bc80: 0020c310 00203bf4 0020c9b8 00000000 a5a5a5a5 a5a5a5a5 a5a5a5a5 0021bd2c
[000377.947582] 0021bca0: 00000005 00000000 00000100 8010b5ec 0010d29c 0020f780 0020fb80 8008e8e8
[000377.956508] 0021bcc0: a5a5a5a5 8000bad4 0020fb80 00000084 a5a5a5a5 8000bad4 00000001 ffffffff
[000377.965486] 0021bce0: 80080020 0020d5f8 0020d5f8 8008eaa6 a5a5a5a5 0020fb80 0020f780 00000380
[000377.974475] 0021bd00: 80080020 0020d5f8 0020d5f8 0021cdc8 00000000 ffffffff 00222c20 0010a9c6
[000377.983442] 0021bd20: 80122218 00000000 e0871aa7 00215798 00000004 0020dea8 00223500 80100878
[000377.992394] 0021bd40: 000048e2 00203bf4 0010d8d4 00206ef4 00000009 00000000 0010b70e 0000001c
[000378.001283] 0021bd60: 167be6a8 00000000 002228e0 000002a0 00003d2a 00000009 00206ef4 0010d8d4
[000378.010177] 0021bd80: 00203bf4 000048e2 00000000 00000414 00000000 80001010 00000000 00000000
```

1.4.5 Dump of User Stack

The user stack section includes detailed memory contents from the bottom of the stack up to the current stack pointer (sp). This information is particularly valuable for debugging as it likely contains the call flow and other critical data relevant at the time of the core dump.

Example Reason Output:

```
[000378.019024] sp:    0021cbfc
[000378.022018] User stack:
[000378.024568]   base: 0021bda0
[000378.027562]   size: 00000ff0
[000378.030556] 0021cbe0: 002065d4 c0000000 00000002 ffffffff 0020d488 00000002 8008e6fe c0000000
[000378.039515] 0021cc00: 00000080 00003003 00206658 8008e6fe 8008e6f8 001096f0 00000000 00000002
[000378.048420] 0021cc20: 0020d488 c0000000 00000000 00000010 00000001 00000002 ffffffff 00000008
[000378.057283] 0021cc40: 00210f37 ffffffff 00000001 8010a220 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5
[000378.066287] 0021cc60: a5a5a5a5 a5a5a5a5 a5a5a5a5 00210f2c 000001b0 8008e71c 00000008 00001880
[000378.075244] 0021cc80: a5a5a5a5 a5a5a5a5 8010a220 c0000000 ffffffff 0020d488 00000002 8008e742
[000378.084249] 0021cca0: ffffffff 0020d488 00000002 800981cc e2a0c53d 00210d70 0020d3f8 00210f37
[000378.093222] 0021ccc0: 00210f20 00210f2d 00000000 00000000 00000000 00000000 00000000 00000000
[000378.102032] 0021cce0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000378.110795] 0021cd00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000378.119558] 0021cd20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000378.128321] 0021cd40: a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 80113eac 00210f20 800985b2
[000378.137367] 0021cd60: a5a5a5a5 0020d5fc 00202c04 800985c6 a5a5a5a5 0020d5fc 00202c04 800ff356
[000378.146382] 0021cd80: a5a5a5a5 a5a5a5a5 a5a5a5a5 00000000 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5
```

1.4.6 List of Active Tasks

Finally, the list of active tasks provides a snapshot of all running and blocked tasks in the system, similar to the output of the ps command. This helps identify which tasks were active and how their stacks were being utilized at the time of the crash.

Example Task List:

```
[000378.159763]  PID   PR  STWM  S   TASK
[000378.167018]    1    3   531  X   init                        (0x21bda0-0x21cd90, 0x21c6f8)
[000378.175996]    2    0   958  R   idle                        (0x20fc04-0x210c00, 0x210b1c)
[000378.184966]    4    3   683  B   knetd                       (0x21d0b0-0x21dca0, 0x21db5c)
[000378.193936]    3    5   199  B   ksofttimerd                 (0x20f780-0x20fb70, 0x20fa9c)
[000378.202906]    8    7    48  B   ll                          (0x2223f0-0x222660, 0x22255c)
[000378.211871]    5    3   571  B   scm2020-wlan fast taskq     (0x21f710-0x220100, 0x21fffc)
[000378.220841]    7    3   303  B   rt_msg                      (0x221600-0x221bf0, 0x221acc)
[000378.229811]    6    3   315  B   knet80211d/wlan0            (0x220d30-0x221320, 0x22121c)
```

In the event of a core dump when interactive CLI is unavailable, tools like a JTAG debugger can be used to further investigate the last stack contents of the active threads, offering deeper insights into the circumstances leading to the system failure.

1.5 Use of Disassembler

Analyzing stack content from a core dump or from the output of the `hexdump` command necessitates disassembling the executable, referred to here as "wise", to understand the program's instruction flow.

There are two primary disassembled outputs relevant for debugging:

	Content	Note
wise.dis	This is the disassembled output of the "wise" executable.	**Generation**: Can be created using the command make wise.dis. **Content**: Includes only the executable code, excluding any instructions located in the ROM library.
wise_rom.dis	This contains the disassembled output of the ROM library.	**Location**: Available in /hal/soc/scm2010. **Content**: Includes only the instructions that are part of the ROM (Read-Only Memory).

(*) The ROM library is a software library embedded in SCM1612's internal ROM, and it is crucial for operations that are frequently accessed or critical for performance.

Example of wise.dis

The beginning of `wise.dis` shows the assembly instructions for initializing the system and setting up the environment, crucial for the boot process:

```
---------------------------------------------------------------------------------------------------------
wise:     file format elf32-littleriscv


Disassembly of section .text.boot:

80080020 <_start>:

_start:
        /* Initialize global pointer */
        .option push
        .option norelax
        la gp, __global_pointer$
80080020:         8018b197          auipc    gp,0x8018b
80080024:         3e018193          addi     gp,gp,992 # 20b400 <__global_pointer$>
        .option pop

        /* Initialize stack pointer */
        la t0, _stack
80080028:         8019c297          auipc    t0,0x8019c
8008002c:         d6828293          addi     t0,t0,-664 # 21bd90 <__stack_end>
        mv sp, t0
80080030:         8116                       c.mv     sp,t0
```

```
        /* Initialize FCSR */
        fscsr zero
#endif

        /* Disable all interrupts (i.e. timer, external) in mie */
        csrw mie, zero
80080032:         30401073          csrw     mie,zero

        /* Initial machine trap-vector Base. Use FreeRTOS trap function. */
        la t0, freertos_risc_v_trap_handler
80080036:         7ff80297          auipc    t0,0x7ff80
8008003a:         34a28293          addi     t0,t0,842 # 380 <__heapext2_end+0x5fdd0380>
        csrw mtvec, t0
8008003e:         30529073          csrw     mtvec,t0
```

Example of wise_rom.dis

`wise_rom.dis` provides disassembled content of the ROM section, showing how built-in library functions are implemented:

```
wise_rom:     file format elf32-littleriscv


Disassembly of section .rom.text:

00106000 <rom_version>:
  106000:01 00 00 00 01 00 00 00                          ........

00106008 <abort>:
  106008:00004317          auipc    t1,0x4
  10600c:e2a302e7          jalr     t0,-470(t1) # 109e32 <__riscv_save_0>
  106010:4505                       c.li     a0,1
  106012:00004097          auipc    ra,0x4
  106016:04a080e7          jalr     74(ra) # 10a05c <_exit>

0010601a <abs>:
  10601a:41f55793          srai     a5,a0,0x1f
  10601e:8d3d                       c.xor    a0,a5
  106020:8d1d                       c.sub    a0,a5
  106022:8082                       c.jr     ra

00106024 <atoi>:
  106024:4581                       c.li     a1,0
  106026:4629                       c.li     a2,10
  106028:00002317          auipc    t1,0x2
  10602c:b8430067          jr       -1148(t1) # 107bac <strtol>
  106030:0001                       c.nop
        ...

00106034 <bcopy>:
  106034:86aa                       c.mv     a3,a0
  106036:852e                       c.mv     a0,a1
  106038:85b6                       c.mv     a1,a3
  10603a:00000317          auipc    t1,0x0
  10603e:35630067          jr       854(t1) # 106390 <memmove>
        ...

00106044 <bzero>:
  106044:862e                       c.mv     a2,a1
  106046:4581                       c.li     a1,0
  106048:00000317          auipc    t1,0x0
  10604c:42430067          jr       1060(t1) # 10646c <memset>
  106050:0001                       c.nop
```

Any typical call flow will include both functions being built within wise-sdk and

functions, i.e., instructions, that have already been put into ROM inside SCM1612, and they will be glued by jump tables, which is why it is difficult to provide a consolidated, easy-to-use debugging environment and/or tools to encompass loaded and pre-installed instructions and data all at once. To make things complicated even further, those jump tables are to be updated at run-time, e.g., to patch ROM library functions.

It is worth noting that an executable built as a standalone mode, in other words using scm1612s_defconfig or derived one, will read instructions from three different locations in target memory space.

Memroy Location	Address Range	Note
ILM (Instruction Local Memory)	0x00000000 – 0x00007fff	- 32KB
XIP flash (*)	0x80080000 – 0x8013ffff or 0x80080000 – 0x801fffff	- 768KB or - 1536KB
ROM	0x106000 – 0x153fff	- 312KB - Built-in in scm1612 - Can be patched

(*) Default location, two options for the size

Referring this address ranges will expedite analyzing stack content to find out the call flow because it becomes possible to focus only on addresses which are eligible for instructions.

1.6 Demo: How-to-Debug with SCM1612 SDK

1.6.1 Overview

This section provides a step-by-step guide on how to utilize the "how-to-debug" demo application included in the SCM1612 SDK. This demo is designed to facilitate the understanding of various debugging techniques corresponding to common issues such as core exceptions, assertion failures, stack overruns, and memory leaks.

1.6.2 Preparing the Demo Environment

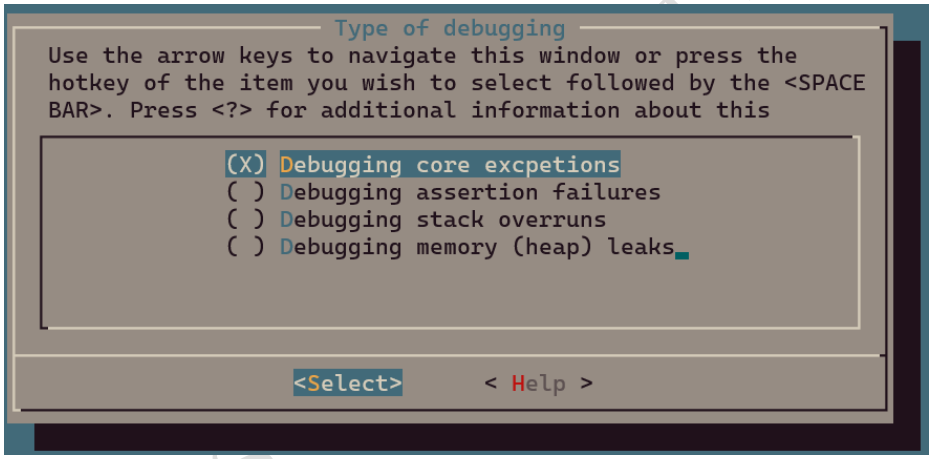
1. Set Up Build Configuration
   Start by configuring the build environment to use the "how-to-debug" demo as the main application:

   ```
   $ make scm1612s_defconfig
   $ make menuconfig
   ```

2. Select the Demo Application
   - Navigate to `Applications -> How-to-debug demo` in the menu configuration.
   - Confirm the selection, exit the configuration menu, and save the changes.

3. Select the Debugging Scenario

Depending on the specific issue you want to demonstrate, select the appropriate debugging type under `Applications -> How-to-debug demo configuration -> Type of debugging`:



Selecting different options will be required per the following sections.

Option	Corresponding section
Debugging core exceptions	2 Core exception
Debugging assertion failures	3 Assertion failure
Debugging stack overruns	4 Stack overrun
Debugging memory (heap) leaks	5 Memory leak

Exit the menu and save the configuration.

1.6.3 Building the Demo

- Build the executable binary:
  $ make

- Note: Depending on the type of problem selected, you may encounter build errors. If this occurs, review the error message provided and adjust the configuration by enabling additional features as suggested.

1.6.4 Running the Demo

- Download and Run the Application:
  - Follow the instructions in the `SDK_Getting_Started_Guide` to download and flash the `wise.mcuboot.bin` onto the scm1612 Evaluation Kit (EVK).
  - Observe the demo's behavior, which will showcase how the selected debugging issue manifests and can be diagnosed using the techniques described in this guide.

2 Core Exception

2.1 Overview

Core exceptions in the SCM1612, which uses a RISC-V core, can arise from various issues identifiable through the `mcause` register. This register helps in diagnosing the exact cause of exceptions by providing specific exception codes.

Exception Codes and Descriptions
Here is a summary of the common exception codes found in the mcause register, along with their meanings:

mcause[11:0]	Description
0	Instruction address misaligned
1	Instruction access fault
2	Illegal instruction
3	Breakpoint
4	Load address misaligned
5	Load access fault
6	Store/AMO address misaligned
7	Store/AMO access fault
8	Environment call from U-mode

The SCM1612 SDK includes a demo application specifically designed to demonstrate the handling of `Illegal instruction`, `Instruction access fault`, and `Breakpoint` exceptions. This is achieved through a CLI command called `jump_to_func`, which simulates jumping to specified instruction addresses that are crafted to trigger these exceptions.

```
WISE 2018.02+ (Apr 18 2024 - 10:40:27 -0700)
Exception demo.
$ help
dhcps               - Configure, start and stop DHCP server
dmesg               - display kernel messages
heap                - kernel heap status
help                - print command description and usage
hexdump             - hexdump address size
history             - show/get history
ifconfig            - configure network interfaces
iperf3              - A TCP, UDP, and SCTP network bandwidth measurement tool
irq                 - display irq information
jump_to_func        - Jump to a function address
mcuboot_agent       - MCUBoot update agent
mcuboot_confirm     - MCUBoot confirm
mcuboot_set_img     - MCUBoot set image
mcuboot_version     - MCUBoot version
memcmp              - compare memory
net                 - test routines for net (lwIP/net80211/driver)
ping                - send ICMP ECHO_REQUEST to network hosts
pm                  - CLI for PM API test
ps                  - report the current process snapshot
read                - read -(d|b|s|l) address length
reboot              - reboot <n>
rtc_cal             - set RTC calibration interval
top                 - display FreeRTOS tasks
version             - display wise, compiler and linker version
watcher             - CLI commands for WIFI PM
wifi                - CLI for wifi API test
write               - write -(b|s|l) address value
$
$ jump_to_func
Usage: jump_to_func <address of the function in hex.>
$
```

How jump_to_func Works

`jump_to_func` is a straightforward CLI command that facilitates the simulation of core exceptions by allowing the user to jump to arbitrary instruction addresses. This command is particularly useful for educational and debugging purposes, as it can directly induce error conditions.

Triggering Different Exceptions

Here's how you can use `jump_to_func` to trigger specific exceptions by passing different, invalid addresses:

2.2 Illegal Instruction Exception

Passing 0xc0000000 to jump_to_func command will lead to 'Illegal instruction' exception.

```
---------------------------------------------------------------------------------------------------------------
$ jump_to_func
Usage: jump_to_func <address of the function in hex.>
$
$ jump_to_func 0xc0000000
WISE 2018.02+ riscv32-elf-gcc.gnu (2022-02-07_nds32le-elf-mculib-v5-86807094a2f) 10.3.0
[003086.625667] BOARD: SCM2010 QFN40 EVB V1.0
[003086.625827] VFS: filesystem devfs mounted onto /dev
[003086.627695] PINCTRL: pin controller scm2010,pinctrl registered
[003086.627906] GPIO: scm2010,pinctrl registered as /dev/gpio
[003086.628180] atcwdt: @0xf1300000, clk=32768
[003086.628394] WDT: atcwdt registered as /dev/watchdog
[003086.628747] PINCTRL: atcspi200-xip/cs request pin 11
```

```
[003086.628947] PINCTRL: atcspi200-xip/clk request pin 12
[003086.629118] PINCTRL: atcspi200-xip/mosi request pin 13
[003086.629297] PINCTRL: atcspi200-xip/miso request pin 14
[003086.629488] PINCTRL: atcspi200-xip/hold request pin 9
[003086.629675] PINCTRL: atcspi200-xip/wp request pin 10
[003086.630177] EFUSE: efuse-scm2010 registered as /dev/efuse
[003086.630314] trng version : 5e5e0010
[003086.630484] TRNG: trng registered as /dev/trng
[003086.630612] pke version : 0x5e5e0010
[003086.672324] Use fixed MAC address: 64.f9.47.f0.01.20
[003086.673338] 264 usec elapsed in downloading MAC FW.
[003087.967932] Wlan PM ctx init done
[003087.969345] UART: atcuart.0 registered as /dev/ttyS0
[003087.969499] CONSOLE: add /dev/ttyS0
[003087.969953] UART: atcuart.1 registered as /dev/ttyS1
[003087.970085] SOC: SCM2010
[003087.970611] Use fixed BLE public address: 01.02.03.04.05.06
[003087.972878] ble phy init 35
[003087.973030] PM feature : 0x1d
[003087.973811] PM LS : 6500+1120=7620
[003087.973979] PM SL : 6500+7100=13600
[003087.974147] PM DS : 11500+12800=24300
[003087.974326] PM HB : 1655000+60000=1715000
[003087.974589] PINCTRL: atcuart.0/txd request pin 22
[003087.974759] PINCTRL: atcuart.0/rxd request pin 21
[002333.517925] Unhandled Trap : Ilegal instruction (mcause = 0x2), mepc = 0xc0000000
[002333.525639] EPC:c0000000
[002333.528363] MSTATUS:00001880
[002333.531442] MXSTATUS:00000080
[002333.534611] MTVAL:00003003
[002333.537515] A0:c0000000 A1:00000000 A2:00000010 A3:00000001 A4:00000002 A5:ffffffff A6:00000008
A7:00210f37
[002333.547596] T0:8008e6f8 T1:001096f0 T2:00000000 T3:00210f2c T4:000001b0 T5:8008e71c T6:00000008
[002333.556635] S0:00000002 S1:0020d488 S2:ffffffff S3:00000001 S4:8010a220 S5:a5a5a5a5 S6:a5a5a5a5
S7:a5a5a5a5
[002333.566817] S8:a5a5a5a5 S9:a5a5a5a5 S10:a5a5a5a5 S11:a5a5a5a5
[002333.572828] FP:00000002 RA:8008e6fe
[002333.576440] sp:    0021cb6c
[002333.579428] IRQ stack:
[002333.581894]   base: 0021b590
[002333.584884]   size: 00000800
[002333.587880] ERROR: Stack pointer is not within the interrupt stack
[002333.594217] 0021b580: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.602990] 0021b5a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.611760] 0021b5c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.620529] 0021b5e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.629299] 0021b600: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.638068] 0021b620: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.646838] 0021b640: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.655607] 0021b660: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.664376] 0021b680: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.673146] 0021b6a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.681915] 0021b6c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.690685] 0021b6e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.699454] 0021b700: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.708223] 0021b720: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.716993] 0021b740: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.725762] 0021b760: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.734531] 0021b780: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.743300] 0021b7a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.752070] 0021b7c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.760839] 0021b7e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.769608] 0021b800: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.778378] 0021b820: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.787147] 0021b840: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.795917] 0021b860: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.804686] 0021b880: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.813455] 0021b8a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.822225] 0021b8c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.830994] 0021b8e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.839763] 0021b900: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.848533] 0021b920: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.857302] 0021b940: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

```
[002333.866071] 0021b960: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.874841] 0021b980: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.883610] 0021b9a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.892379] 0021b9c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.901149] 0021b9e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.909918] 0021ba00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.918687] 0021ba20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.927457] 0021ba40: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.936226] 0021ba60: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.944996] 0021ba80: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.953765] 0021baa0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.962534] 0021bac0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.971304] 0021bae0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.980073] 0021bb00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.988842] 0021bb20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002333.997612] 0021bb40: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002334.006381] 0021bb60: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002334.015140] 0021bb80: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002334.023904] 0021bba0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002334.032668] 0021bbc0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002334.041432] 0021bbe0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002334.050196] 0021bc00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002334.058961] 0021bc20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002334.067725] 0021bc40: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002334.076489] 0021bc60: 00000000 00000000 00000000 00000000 0020c310 00203bf4 0020c9b8 80100c3a
[002334.085369] 0021bc80: 0020c310 00203bf4 0020c9b8 00000000 a5a5a5a5 a5a5a5a5 a5a5a5a5 0021bd2c
[002334.094348] 0021bca0: 00000005 00000000 00000100 8010b5ec 0010d29c 0020f780 0020fb80 8008e8e8
[002334.103275] 0021bcc0: a5a5a5a5 a5a5a5a5 0020fb80 00000084 a5a5a5a5 a5a5a5a5 8000b660 00202bc8
[002334.112286] 0021bce0: 80080020 0020d5f8 0020d5f8 8008eaa6 a5a5a5a5 0020fb80 0020f780 00000380
[002334.121282] 0021bd00: 80080020 0020d5f8 0020d5f8 0021cdc8 00000000 ffffffff 00222bb0 0010a9c6
[002334.130256] 0021bd20: 80122218 0000002d 6e9c9552 00215798 00000004 0020dea8 00223090 80100878
[002334.139219] 0021bd40: 00361682 00203bf4 0010d8d4 00206ef4 00000009 00000000 0010b70e 0000001c
[002334.148125] 0021bd60: 8b145395 00000004 002224e0 000002a0 00360aca 00000009 00206ef4 0010d8d4
[002334.157047] 0021bd80: 00203bf4 00361682 00000000 00000414 00000000 80001010 00000000 00000000
[002334.165916] sp:    0021cbfc
[002334.168909] User stack:
[002334.171460]  base: 0021bda0
[002334.174453]  size: 00000ff0
[002334.177447] 0021cbe0: 002065d4 c0000000 00000002 ffffffff 0020d488 00000002 8008e6fe c0000000
[002334.186418] 0021cc00: 00000080 00003003 00206658 8008e6fe 8008e6f8 001096f0 00000000 00000002
[002334.195335] 0021cc20: 0020d488 c0000000 00000000 00000010 00000001 00000002 ffffffff 00000008
[002334.204209] 0021cc40: 00210f37 ffffffff 00000001 8010a220 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5
[002334.213225] 0021cc60: a5a5a5a5 a5a5a5a5 a5a5a5a5 00210f2c 000001b0 8008e71c 00000008 00001880
[002334.222194] 0021cc80: a5a5a5a5 a5a5a5a5 8010a220 c0000000 ffffffff 0020d488 00000002 8008e742
[002334.231211] 0021cca0: ffffffff 0020d488 00000002 800981cc 7095067b 00210d70 0020d3f8 00210f37
[002334.240195] 0021ccc0: 00210f20 00210f2d 00000000 00000000 00000000 00000000 00000000 00000000
[002334.249017] 0021cce0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002334.257787] 0021cd00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002334.266556] 0021cd20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002334.275325] 0021cd40: a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 80113eac 00210f20 800985b2
[002334.284378] 0021cd60: a5a5a5a5 0020d5fc 00202c04 800985c6 a5a5a5a5 0020d5fc 00202c04 800ff356
[002334.293400] 0021cd80: a5a5a5a5 a5a5a5a5 a5a5a5a5 00000000 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5
[002334.306791]  PID   PR  STWM  S   TASK
[002334.314053]    1    3   532  X   init                        (0x21bda0-0x21cd90, 0x21c6f8)
[002334.323036]    2    0   958  R   idle                        (0x20fc04-0x210c00, 0x210b1c)
[002334.332012]    4    3   683  B   knetd                       (0x21d0b0-0x21dca0, 0x21db5c)
[002334.340989]    3    5   199  B   ksofttimerd                 (0x20f780-0x20fb70, 0x20fa9c)
[002334.349966]    6    3   315  B   knet80211d/wlan0            (0x220d30-0x221320, 0x22121c)
[002334.358942]    7    3   303  B   rt_msg                      (0x221600-0x221bf0, 0x221acc)
[002334.367915]    8    7    48  B   ll                          (0x2227b0-0x222a20, 0x22291c)
[002334.376883]    5    3   571  B   scm2020-wlan fast taskq      (0x21f710-0x220100, 0x21fffc)
```

From the core dump, we can understand:

- 'Illegal instruction exception' occurred due to an access to an invalid address of 0xc0000000, and
- 'init' task was running when this exception occurred.

Examining the dumped user stack would reveal further information regarding the specific call flow that caused this exception.
Note that a stack grows from bottom to top, i.e., from higher address to lower address, which means that instruction addresses found at bottom correspond to beginning of the call flow.

```
---------------------------------------------------------------------------------------------------------------
[002334.165916] sp:    0021cbfc
[002334.168909] User stack:
[002334.171460]   base: 0021bda0
[002334.174453]   size: 00000ff0
[002334.177447] 0021cbe0: 002065d4 c0000000 00000002 ffffffff 0020d488 00000002 8008e6fe c0000000
[002334.186418] 0021cc00: 00000080 00003003 00206658 8008e6fe 8008e6f8 001096f0 00000000 00000002
[002334.195335] 0021cc20: 0020d488 c0000000 00000000 00000010 00000001 00000002 ffffffff 00000008
[002334.204209] 0021cc40: 00210f37 ffffffff 00000001 8010a220 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5
[002334.213225] 0021cc60: a5a5a5a5 a5a5a5a5 a5a5a5a5 00210f2c 000001b0 8008e71c 00000008 00001880
[002334.222194] 0021cc80: a5a5a5a5 a5a5a5a5 8010a220 c0000000 ffffffff 0020d488 00000002 8008e742
[002334.231211] 0021cca0: ffffffff 0020d488 00000002 800981cc 7095067b 00210d70 0020d3f8 00210f37
[002334.240195] 0021ccc0: 00210f20 00210f2d 00000000 00000000 00000000 00000000 00000000 00000000
[002334.249017] 0021cce0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002334.257787] 0021cd00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002334.266556] 0021cd20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[002334.275325] 0021cd40: a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 80113eac 00210f20 800985b2
[002334.284378] 0021cd60: a5a5a5a5 0020d5fc 00202c04 800985c6 a5a5a5a5 0020d5fc 00202c04 800ff356
[002334.293400] 0021cd80: a5a5a5a5 a5a5a5a5 a5a5a5a5 00000000 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5
[002334.306791]  PID   PR  STWM  S   TASK
[002334.314053]    1    3   532  X   init                        (0x21bda0-0x21cd90, 0x21c6f8)
[002334.323036]    2    0   958  R   idle                        (0x20fc04-0x210c00, 0x210b1c)
[002334.332012]    4    3   683  B   knetd                       (0x21d0b0-0x21dca0, 0x21db5c)
[002334.340989]    3    5   199  B   ksofttimerd                 (0x20f780-0x20fb70, 0x20fa9c)
[002334.349966]    6    3   315  B   knet80211d/wlan0            (0x220d30-0x221320, 0x22121c)
[002334.358942]    7    3   303  B   rt_msg                      (0x221600-0x221bf0, 0x221acc)
[002334.367915]    8    7    48  B   ll                          (0x2227b0-0x222a20, 0x22291c)
[002334.376883]    5    3   571  B   scm2020-wlan fast taskq      (0x21f710-0x220100, 0x21fffc)
---------------------------------------------------------------------------------------------------------------
```

Candidates of instruction addresses are highlighted above in accordance with the table shown in 1.5. In this specific case, there will be no ILM or ROM addresses involved because the whole call flow happened within XIP flash region.

Looking up wise.dis for these addresses will reveal the call flow up to the point of exception as follows.

(800ff356)

```
800ff338:   ee058593            addi    a1,a1,-288 # 80109214 <version_string>
800ff33c:   00022517            auipc   a0,0x22
800ff340:   fd850513            addi    a0,a0,-40 # 80121314 <boot_img_magic+0x4ec>
800ff344:   9802                    c.jalr  a6

    /* Calls to main() if there is one, or weak main() declared above */
    main();
800ff346:   fff8f097            auipc   ra,0xfff8f
800ff34a:   36c080e7            jalr    876(ra) # 8008e6b2 <main>
     * coexist and switch between each other.
     */
    do {
        extern void cli_loop(void);

        cli_loop();
800ff34e:   fff99097            auipc   ra,0xfff99
800ff352:   266080e7            jalr    614(ra) # 800985b4 <cli_loop>
    } while (0);
#else
    while (1) osDelay(pdMS_TO_TICKS(100));
#endif
    assert(0); /* should not come here */
800ff356:   80105897            auipc   a7,0x80105
800ff35a:   89e8a883            lw  a7,-1890(a7) # 203bf4 <hal_assert_fail>
800ff35e:   00023697            auipc   a3,0x23
800ff362:   f8268693            addi    a3,a3,-126 # 801222e0 <__func__.0>
800ff366:   07b00613            li  a2,123
800ff36a:   4581                    c.li    a1,0
800ff36c:   00023517            auipc   a0,0x23
800ff370:   9ec50513            addi    a0,a0,-1556 # 80121d58 <__func__.3+0x174>
800ff374:   9882                    c.jalr  a7
```

(800985c6)

```
void cli_loop(void)
{
800985b4:   80072317            auipc   t1,0x80072
800985b8:   87e302e7            jalr    t0,-1922(t1) # 109e32 <__riscv_save_0>
    cli_task(NULL);
800985bc:   4501                    c.li    a0,0
800985be:   00000097            auipc   ra,0x0
800985c2:   fbe080e7            jalr    -66(ra) # 8009857c <cli_task>

800985c6 <memfile_write>:
    size_t bytes_written;
    size_t size;
};
```

(800985b2)

```
80098584:   80179417            auipc   s0,0x80179
80098588:   99c40413            addi    s0,s0,-1636 # 210f20 <input.0>
8009858c:   0007c497            auipc   s1,0x7c
80098590:   92048493            addi    s1,s1,-1760 # 80113eac <ptable+0xb8>
80098594:   10000613            li  a2,256
80098598:   85a2                    c.mv    a1,s0
8009859a:   8526                    c.mv    a0,s1
8009859c:   00000097            auipc   ra,0x0
800985a0:   d12080e7            jalr    -750(ra) # 800982ae <cli_readline>
        if (len <= 0)
800985a4:   fea058e3            blez    a0,80098594 <cli_task+0x18>
            continue;

        cli_run_command(input);
800985a8:   8522                    c.mv    a0,s0
800985aa:   00000097            auipc   ra,0x0
800985ae:   acc080e7            jalr    -1332(ra) # 80098076 <cli_run_command>
800985b2:   b7cd                    c.j 80098594 <cli_task+0x18>

800985b4 <cli_loop>:
#endif
}
```

(800981cc)

```
    cmd_history[hindex(cmd_hindex++)] = str;
800981a8:   c098                    c.sw    a4,0(s1)
    if (cmd_history[index]) {
800981aa:   d13d                    c.beqz  a0,80098110 <cli_run_command+0x9a>
        free(cmd_history[index]);
800981ac:   80080097            auipc   ra,0x80080
800981b0:   c14080e7            jalr    -1004(ra) # 117dc0 <os_free>
        cmd_history[index] = NULL;
800981b4:   00092023            sw  zero,0(s2)
800981b8:   bfa1                    c.j 80098110 <cli_run_command+0x9a>
    rc = cmd->handler(argc, argv);
800981ba:   00452f03            lw  t5,4(a0)
    optind = 0;
800981be:   80170697            auipc   a3,0x80170
800981c2:   4206ab23            sw  zero,1078(a3) # 2085f4 <optind>
    rc = cmd->handler(argc, argv);
800981c6:   8522                    c.mv    a0,s0
800981c8:   080c                    c.addi4spn  a1,sp,16
800981ca:   9f02                    c.jalr  t5
800981cc:   842a                    c.mv    s0,a0
    if (rc == CMD_RET_USAGE) {
800981ce:   03251663            bne a0,s2,800981fa <cli_run_command+0x184>
        if (cmd->usage) {
800981d2:   44cc                    c.lw    a1,12(s1)
800981d4:   c999                    c.beqz  a1,800981ea <cli_run_command+0x174>
```

(8008e742)

```
8008e71c <do_jump_to_func>:
{
    func_to_jump g_func_to_jump;

    argc--, argv++;

    if (argc == 1) {
8008e71c:   4709                    c.li    a4,2
8008e71e:   02e51763            bne a0,a4,8008e74c <do_jump_to_func+0x30>
{
8008e722:   8007b317            auipc   t1,0x8007b
8008e726:   710302e7            jalr    t0,1808(t1) # 109e32 <__riscv_save_0>
8008e72a:   87ae                    c.mv    a5,a1
        g_func_to_jump = (func_to_jump)strtoul(argv[0], NULL, 16);
8008e72c:   43c8                    c.lw    a0,4(a5)
8008e72e:   4641                    c.li    a2,16
8008e730:   4581                    c.li    a1,0
8008e732:   80079097            auipc   ra,0x80079
8008e736:   5c8080e7            jalr    1480(ra) # 107cfa <strtoul>
    } else {
        return CMD_RET_USAGE;
    }

    jump_to_func(g_func_to_jump);
8008e73a:   00000097            auipc   ra,0x0
8008e73e:   fb6080e7            jalr    -74(ra) # 8008e6f0 <jump_to_func>

    return CMD_RET_SUCCESS;
8008e742:   4501                    c.li    a0,0
}
8008e744:   8007b317            auipc   t1,0x8007b
8008e748:   71230067            jr  1810(t1) # 109e56 <__riscv_restore_0>
        return CMD_RET_USAGE;
8008e74c:   557d                    c.li    a0,-1
}
8008e74e:   8082                    c.jr    ra
```

As a result, the reconstructed call flow is:

cli_loop -> cli_task -> cli_run_command -> cmd->handler -> jump_to_func

As can be seen from this example, an instruction address that appears from the stack dump will always correspond to the one following an instruction ‘jalr’, i.e., jump-and-link, or similar.
This is because ‘jalr’ will store a return address, i.e., address of the next instruction, in ‘ra’ register and ‘ra’ register will be saved to the stack by a preface of a destination.

```
void jump_to_func(func_to_jump func)
{
8008e6f0:   8007b317            auipc   t1,0x8007b
8008e6f4:   742302e7            jalr    t0,1858(t1) # 109e32 <__riscv_save_0>
8008e6f8:   1141                    c.addi  sp,-16
    (*func)();
8008e6fa:   c62a                    c.swsp  a0,12(sp)
8008e6fc:   9502                    c.jalr  a0
    printf("Jump to %p was successful.\n", func);
8008e6fe:   45b2                    c.lwsp  a1,12(sp)
8008e700:   80174797            auipc   a5,0x80174
8008e704:   4d47a783            lw  a5,1236(a5) # 202bd4 <os_printf>
8008e708:   0007d517            auipc   a0,0x7d
8008e70c:   da850513            addi    a0,a0,-600 # 8010b4b0 <s_log_color+0xc4>
8008e710:   9782                    c.jalr  a5
}
8008e712:   0141                    c.addi  sp,16
8008e714:   8007b317            auipc   t1,0x8007b
8008e718:   74230067            jr  1858(t1) # 109e56 <__riscv_restore_0>
```

(wise.dis)

```
00109e32 <__riscv_save_0>:
  109e32:   1141                    c.addi  sp,-16
  109e34:   c04a                    c.swsp  s2,0(sp)
  109e36:   c226                    c.swsp  s1,4(sp)
  109e38:   c422                    c.swsp  s0,8(sp)
  109e3a:   c606                    c.swsp  ra,12(sp)
  109e3c:   8282                    c.jr    t0
```

(wise_rom.dis)

2.3 Instruction Access Fault

Passing 0x12345678 to jump_to_func command will lead to ‘Instruction access fault’ exception.

```
---------------------------------------------------------------------------------------------------------
WISE 2018.02+ (Apr 18 2024 - 10:40:27 -0700)
Exception demo.
$
$ jump_to_func
Usage: jump_to_func <address of the function in hex.>
$ jump_to_func 0x12345678
WISE 2018.02+ riscv32-elf-gcc.gnu (2022-02-07_nds32le-elf-mculib-v5-86807094a2f) 10.3.0
[000005.734989] BOARD: SCM2010 QFN40 EVB V1.0
[000005.735131] VFS: filesystem devfs mounted onto /dev
[000005.736982] PINCTRL: pin controller scm2010,pinctrl registered
[000005.737176] GPIO: scm2010,pinctrl registered as /dev/gpio
[000005.737429] atcwdt: @0xf1300000, clk=32768
[000005.737626] WDT: atcwdt registered as /dev/watchdog
[000005.737956] PINCTRL: atcspi200-xip/cs request pin 11
[000005.738139] PINCTRL: atcspi200-xip/clk request pin 12
[000005.738298] PINCTRL: atcspi200-xip/mosi request pin 13
[000005.738459] PINCTRL: atcspi200-xip/miso request pin 14
[000005.738634] PINCTRL: atcspi200-xip/hold request pin 9
[000005.738800] PINCTRL: atcspi200-xip/wp request pin 10
[000005.739284] EFUSE: efuse-scm2010 registered as /dev/efuse
[000005.739406] trng version : 5e5e0010
[000005.739558] TRNG: trng registered as /dev/trng
```

```
[000005.739665] pke version : 0x5e5e0010
[000005.781668] Use fixed MAC address: 64.f9.47.f0.01.20
[000005.782652] 260 usec elapsed in downloading MAC FW.
[000007.077060] Wlan PM ctx init done
[000007.078476] UART: atcuart.0 registered as /dev/ttyS0
[000007.078614] CONSOLE: add /dev/ttyS0
[000007.079001] UART: atcuart.1 registered as /dev/ttyS1
[000007.079112] SOC: SCM2010
[000007.079607] Use fixed BLE public address: 01.02.03.04.05.06
[000007.081854] ble phy init 35
[000007.081975] PM feature : 0x1d
[000007.082731] PM LS : 6500+1120=7620
[000007.082885] PM SL : 6500+7100=13600
[000007.083031] PM DS : 11500+12800=24300
[000007.083189] PM HB : 1655000+60000=1715000
[000007.083429] PINCTRL: atcuart.0/txd request pin 22
[000007.083576] PINCTRL: atcuart.0/rxd request pin 21
[000020.695688] Unhandled Trap : Instruction access fault (mcause = 0x1), mepc = 0x12345678
[000020.703912] EPC:12345678
[000020.706636] MSTATUS:00001880
[000020.709714] MXSTATUS:00000080
[000020.712884] MTVAL:12345678
[000020.715788] A0:12345678 A1:00000000 A2:00000010 A3:00000001 A4:00000002 A5:ffffffff A6:00000004
A7:00210f37
[000020.725855] T0:8008e6f8 T1:001096f0 T2:00000000 T3:00210f2c T4:000001b0 T5:8008e71c T6:00000008
[000020.734882] S0:00000002 S1:0020d488 S2:ffffffff S3:00000001 S4:8010a220 S5:a5a5a5a5 S6:a5a5a5a5
S7:a5a5a5a5
[000020.745053] S8:a5a5a5a5 S9:a5a5a5a5 S10:a5a5a5a5 S11:a5a5a5a5
[000020.751052] FP:00000002 RA:8008e6fe
[000020.754664] sp:    0021cb6c
[000020.757652] IRQ stack:
[000020.760118]   base: 0021b590
[000020.763108]   size: 00000800
[000020.766104] ERROR: Stack pointer is not within the interrupt stack
[000020.772441] 0021b580: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.781203] 0021b5a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.789961] 0021b5c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.798718] 0021b5e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.807476] 0021b600: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.816234] 0021b620: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.824991] 0021b640: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.833749] 0021b660: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.842507] 0021b680: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.851265] 0021b6a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.860022] 0021b6c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.868780] 0021b6e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.877538] 0021b700: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.886296] 0021b720: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.895053] 0021b740: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.903811] 0021b760: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.912569] 0021b780: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.921326] 0021b7a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.930084] 0021b7c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.938842] 0021b7e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.947599] 0021b800: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.956357] 0021b820: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.965115] 0021b840: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.973872] 0021b860: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.982630] 0021b880: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000020.991388] 0021b8a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.000145] 0021b8c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.008887] 0021b8e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.017635] 0021b900: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.026387] 0021b920: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.035140] 0021b940: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.043892] 0021b960: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.052644] 0021b980: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.061397] 0021b9a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.070149] 0021b9c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.078902] 0021b9e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.087654] 0021ba00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.096407] 0021ba20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.105159] 0021ba40: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

```
[000021.113917] 0021ba60: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.122675] 0021ba80: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.131432] 0021baa0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.140190] 0021bac0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.148948] 0021bae0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.157706] 0021bb00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.166463] 0021bb20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.175221] 0021bb40: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.183979] 0021bb60: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.192736] 0021bb80: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.201494] 0021bba0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.210252] 0021bbc0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.219009] 0021bbe0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.227767] 0021bc00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.236525] 0021bc20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.245283] 0021bc40: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.254040] 0021bc60: 00000000 00000000 00000000 00000000 0020c310 00203bf4 0020c9b8 80100c3a
[000021.262914] 0021bc80: 0020c310 00203bf4 0020c9b8 00000000 a5a5a5a5 a5a5a5a5 a5a5a5a5 0021bd2c
[000021.271887] 0021bca0: 00000005 00000000 00000100 8010b5ec 0010d29c 0020f780 0020fb80 8008e8e8
[000021.280807] 0021bcc0: a5a5a5a5 a5a5a5a5 0020fb80 00000084 a5a5a5a5 a5a5a5a5 8000b660 00202bc8
[000021.289807] 0021bce0: 80080020 0020d5f8 0020d5f8 8008eaa6 a5a5a5a5 0020fb80 0020f780 00000380
[000021.298791] 0021bd00: 80080020 0020d5f8 0020d5f8 0021cdc8 00000000 ffffffff 00222f60 0010a9c6
[000021.307753] 0021bd20: 80122218 00000000 0bf04f4c 00215798 00000004 0020dea8 002225d0 80100878
[000021.316694] 0021bd40: 00004590 00203bf4 0010d8d4 00206ef4 00000009 00000000 0010b70e 0000001c
[000021.325578] 0021bd60: 013975b9 00000000 00222490 000002a0 000039d8 00000009 00206ef4 0010d8d4
[000021.334472] 0021bd80: 00203bf4 00004590 00000000 00000414 00000000 80001010 00000000 00000000
[000021.343319] sp:    0021cbfc
[000021.346312] User stack:
[000021.348863]  base: 0021bda0
[000021.351856]  size: 00000ff0
[000021.354850] 0021cbe0: 002065d4 12345678 00000001 ffffffff 0020d488 00000002 8008e6fe 12345678
[000021.363809] 0021cc00: 00000080 12345678 00206658 8008e6fe 8008e6f8 001096f0 00000000 00000002
[000021.372735] 0021cc20: 0020d488 12345678 00000000 00000010 00000001 00000002 ffffffff 00000004
[000021.381597] 0021cc40: 00210f37 ffffffff 00000001 8010a220 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5
[000021.390601] 0021cc60: a5a5a5a5 a5a5a5a5 a5a5a5a5 00210f2c 000001b0 8008e71c 00000008 00001880
[000021.399559] 0021cc80: a5a5a5a5 a5a5a5a5 8010a220 12345678 ffffffff 0020d488 00000002 8008e742
[000021.408563] 0021cca0: ffffffff 0020d488 00000002 800981cc 0e085bef 00210d70 0020d3f8 00210f37
[000021.417530] 0021ccc0: 00210f20 00210f2d 00000000 00000000 00000000 00000000 00000000 00000000
[000021.426341] 0021cce0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.435099] 0021cd00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.443856] 0021cd20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000021.452614] 0021cd40: a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 80113eac 00210f20 800985b2
[000021.461655] 0021cd60: a5a5a5a5 0020d5fc 00202c04 800985c6 a5a5a5a5 0020d5fc 00202c04 800ff356
[000021.470665] 0021cd80: a5a5a5a5 a5a5a5a5 a5a5a5a5 00000000 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5
[000021.484043]  PID   PR  STWM  S   TASK
[000021.491295]    1    3   532  X   init                          (0x21bda0-0x21cd90, 0x21c6f8)
[000021.500266]    2    0   966  R   idle                          (0x20fc04-0x210c00, 0x210b1c)
[000021.509233]    4    3   683  B   knetd                         (0x21d0b0-0x21dca0, 0x21db5c)
[000021.518199]    3    5   199  B   ksofttimerd                    (0x20f780-0x20fb70, 0x20fa9c)
[000021.527165]    7    3   303  B   rt_msg                        (0x221600-0x221bf0, 0x221acc)
[000021.536132]    6    3   315  B   knet80211d/wlan0               (0x220d30-0x221320, 0x22121c)
[000021.545098]    8    7    48  B   ll                           (0x222b60-0x222dd0, 0x222ccc)
[000021.554060]    5    3   571  B   scm2020-wlan fast taskq          (0x21f710-0x220100, 0x21fffc)
```

2.4 Breakpoint Exception

Passing an address corresponding to ‘ebreak’ instruction, in this case 0x422, to jump_to_func command will lead to ‘Breakpoint’ exception.

```
WISE 2018.02+ (Apr 18 2024 - 10:40:27 -0700)
Exception demo.
$
$
$ jump_to_func
Usage: jump_to_func <address of the function in hex.>
$
```

```
$ jump_to_func 0x422
WISE 2018.02+ riscv32-elf-gcc.gnu (2022-02-07_nds32le-elf-mculib-v5-86807094a2f) 10.3.0
[000005.734989] BOARD: SCM2010 QFN40 EVB V1.0
[000005.735131] VFS: filesystem devfs mounted onto /dev
[000005.736982] PINCTRL: pin controller scm2010,pinctrl registered
[000005.737176] GPIO: scm2010,pinctrl registered as /dev/gpio
[000005.737444] atcwdt: @0xf1300000, clk=32768
[000005.737641] WDT: atcwdt registered as /dev/watchdog
[000005.737971] PINCTRL: atcspi200-xip/cs request pin 11
[000005.738154] PINCTRL: atcspi200-xip/clk request pin 12
[000005.738313] PINCTRL: atcspi200-xip/mosi request pin 13
[000005.738474] PINCTRL: atcspi200-xip/miso request pin 14
[000005.738649] PINCTRL: atcspi200-xip/hold request pin 9
[000005.738815] PINCTRL: atcspi200-xip/wp request pin 10
[000005.739299] EFUSE: efuse-scm2010 registered as /dev/efuse
[000005.739421] trng version : 5e5e0010
[000005.739573] TRNG: trng registered as /dev/trng
[000005.739680] pke version : 0x5e5e0010
[000005.781669] Use fixed MAC address: 64.f9.47.f0.01.20
[000005.782653] 260 usec elapsed in downloading MAC FW.
[000007.077076] Wlan PM ctx init done
[000007.078495] UART: atcuart.0 registered as /dev/ttyS0
[000007.078632] CONSOLE: add /dev/ttyS0
[000007.079019] UART: atcuart.1 registered as /dev/ttyS1
[000007.079130] SOC: SCM2010
[000007.079625] Use fixed BLE public address: 01.02.03.04.05.06
[000007.081873] ble phy init 35
[000007.081994] PM feature : 0x1d
[000007.082750] PM LS : 6500+1120=7620
[000007.082904] PM SL : 6500+7100=13600
[000007.083050] PM DS : 11500+12800=24300
[000007.083207] PM HB : 1655000+60000=1715000
[000007.083447] PINCTRL: atcuart.0/txd request pin 22
[000007.083594] PINCTRL: atcuart.0/rxd request pin 21
[000092.145991] Unhandled Trap : Breakpoint (ebreak) (mcause = 0x3), mepc = 0x422
[000092.153305] EPC:00000422
[000092.156030] MSTATUS:00001880
[000092.159108] MXSTATUS:00000080
[000092.162277] MTVAL:00000000
[000092.165182] A0:00000422 A1:00000000 A2:00000010 A3:00000001 A4:00000002 A5:ffffffff A6:00000004
A7:00210f32
[000092.175223] T0:8008e6f8 T1:001096f0 T2:00000000 T3:00210f2c T4:000001b0 T5:8008e71c T6:00000008
[000092.184250] S0:00000002 S1:0020d488 S2:ffffffff S3:00000001 S4:8010a220 S5:a5a5a5a5 S6:a5a5a5a5
S7:a5a5a5a5
[000092.194421] S8:a5a5a5a5 S9:a5a5a5a5 S10:a5a5a5a5 S11:a5a5a5a5
[000092.200420] FP:00000002 RA:8008e6fe
[000092.204032] sp:    0021cb6c
[000092.207020] IRQ stack:
[000092.209485]  base: 0021b590
[000092.212476]  size: 00000800
[000092.215472] ERROR: Stack pointer is not within the interrupt stack
[000092.221809] 0021b580: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.230571] 0021b5a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.239328] 0021b5c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.248086] 0021b5e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.256844] 0021b600: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.265601] 0021b620: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.274359] 0021b640: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.283117] 0021b660: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.291875] 0021b680: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.300632] 0021b6a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.309390] 0021b6c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.318148] 0021b6e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.326905] 0021b700: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.335663] 0021b720: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.344421] 0021b740: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.353179] 0021b760: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.361936] 0021b780: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.370694] 0021b7a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.379451] 0021b7c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.388209] 0021b7e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.396967] 0021b800: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.405724] 0021b820: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

```
[000092.414482] 0021b840: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.423240] 0021b860: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.431998] 0021b880: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.440755] 0021b8a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.449513] 0021b8c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.458271] 0021b8e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.467029] 0021b900: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.475786] 0021b920: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.484544] 0021b940: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.493302] 0021b960: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.502059] 0021b980: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.510817] 0021b9a0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.519575] 0021b9c0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.528332] 0021b9e0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.537090] 0021ba00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.545847] 0021ba20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.554605] 0021ba40: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.563363] 0021ba60: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.572121] 0021ba80: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.580878] 0021baa0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.589636] 0021bac0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.598394] 0021bae0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.607152] 0021bb00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.615909] 0021bb20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.624667] 0021bb40: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.633425] 0021bb60: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.642182] 0021bb80: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.650940] 0021bba0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.659698] 0021bbc0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.668455] 0021bbe0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.677213] 0021bc00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.685971] 0021bc20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.694728] 0021bc40: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.703486] 0021bc60: 00000000 00000000 00000000 00000000 0020c310 00203bf4 0020c9b8 80100c3a
[000092.712359] 0021bc80: 0020c310 00203bf4 0020c9b8 00000000 a5a5a5a5 a5a5a5a5 a5a5a5a5 0021bd2c
[000092.721332] 0021bca0: 00000005 00000000 00000100 8010b5ec 0010d29c 0020f780 0020fb80 8008e8e8
[000092.730253] 0021bcc0: a5a5a5a5 a5a5a5a5 0020fb80 00000084 a5a5a5a5 a5a5a5a5 8000b660 00202bc8
[000092.739252] 0021bce0: 80080020 0020d5f8 0020d5f8 8008eaa6 a5a5a5a5 0020fb80 0020f780 00000380
[000092.748236] 0021bd00: 80080020 0020d5f8 0020d5f8 0021cdc8 00000000 ffffffff 00222f60 0010a9c6
[000092.757198] 0021bd20: 80122218 00000000 36a53b6d 00215798 00000004 0020dea8 002225d0 80100878
[000092.766144] 0021bd40: 00015caa 00203bf4 0010d8d4 00206ef4 00000009 00000000 0010b70e 0000001c
[000092.775033] 0021bd60: 057bb479 00000000 00222490 000002a0 000150f2 00000009 00206ef4 0010d8d4
[000092.783932] 0021bd80: 00203bf4 00015caa 00000000 00000414 00000000 80001010 00000000 00000000
[000092.792785] sp:    0021cbfc
[000092.795778] User stack:
[000092.798328]   base: 0021bda0
[000092.801322]   size: 00000ff0
[000092.804316] 0021cbe0: 002065d4 00000422 00000003 ffffffff 0020d488 00000002 8008e6fe 00000422
[000092.813222] 0021cc00: 00000080 00000000 00206658 8008e6fe 8008e6f8 001096f0 00000000 00000002
[000092.822111] 0021cc20: 0020d488 00000422 00000000 00000010 00000001 00000002 ffffffff 00000004
[000092.830948] 0021cc40: 00210f32 ffffffff 00000001 8010a220 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5
[000092.839952] 0021cc60: a5a5a5a5 a5a5a5a5 a5a5a5a5 00210f2c 000001b0 8008e71c 00000008 00001880
[000092.848909] 0021cc80: a5a5a5a5 a5a5a5a5 a5a5a5a5 8010a220 00000422 ffffffff 0020d488 00000002 8008e742
[000092.857887] 0021cca0: ffffffff 0020d488 00000002 800981cc 389ecf69 00210d70 0020d3f8 00210f32
[000092.866860] 0021ccc0: 00210f20 00210f2d 00000000 00000000 00000000 00000000 00000000 00000000
[000092.875670] 0021cce0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.884428] 0021cd00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.893186] 0021cd20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000092.901944] 0021cd40: a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 80113eac 00210f20 800985b2
[000092.910985] 0021cd60: a5a5a5a5 0020d5fc 00202c04 800985c6 a5a5a5a5 0020d5fc 00202c04 800ff356
[000092.919994] 0021cd80: a5a5a5a5 a5a5a5a5 a5a5a5a5 00000000 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5
[000092.933368]  PID   PR  STWM  S   TASK
[000092.940620]    1    3   532  X   init                          (0x21bda0-0x21cd90, 0x21c6f8)
[000092.949592]    2    0   966  R   idle                          (0x20fc04-0x210c00, 0x210b1c)
[000092.958557]    4    3   683  B   knetd                         (0x21d0b0-0x21dca0, 0x21db5c)
[000092.967518]    3    5   199  B   ksofttimerd                    (0x20f780-0x20fb70, 0x20fa9c)
[000092.976479]    7    3   303  B   rt_msg                        (0x221600-0x221bf0, 0x221acc)
[000092.985440]    6    3   315  B   knet80211d/wlan0               (0x220d30-0x221320, 0x22121c)
[000092.994401]    8    7    48  B   ll                           (0x222b60-0x222dd0, 0x222ccc)
[000093.003357]    5    3   571  B   scm2020-wlan fast taskq          (0x21f710-0x220100, 0x21fffc)
```

3 Assertion Failure

3.1 Overview

Assertion failures are critical for identifying exact points of malfunction, which aids both in debugging and troubleshooting within operational environments.

Purpose of Assertions
Assertions provide a structured way to pinpoint and articulate specific failures. They serve as a direct indicator that a predefined condition has failed, facilitating early detection and remediation of potential bugs.

Implementing Assertions
To employ assertions:
1. Include the header <assert.h> in your code.
2. Specify the condition that you expect to be true.

Example usage of assertions is provided below. This technique ensures timely recognition of problems, helping to avoid more complex issues that may arise if initial errors are overlooked.

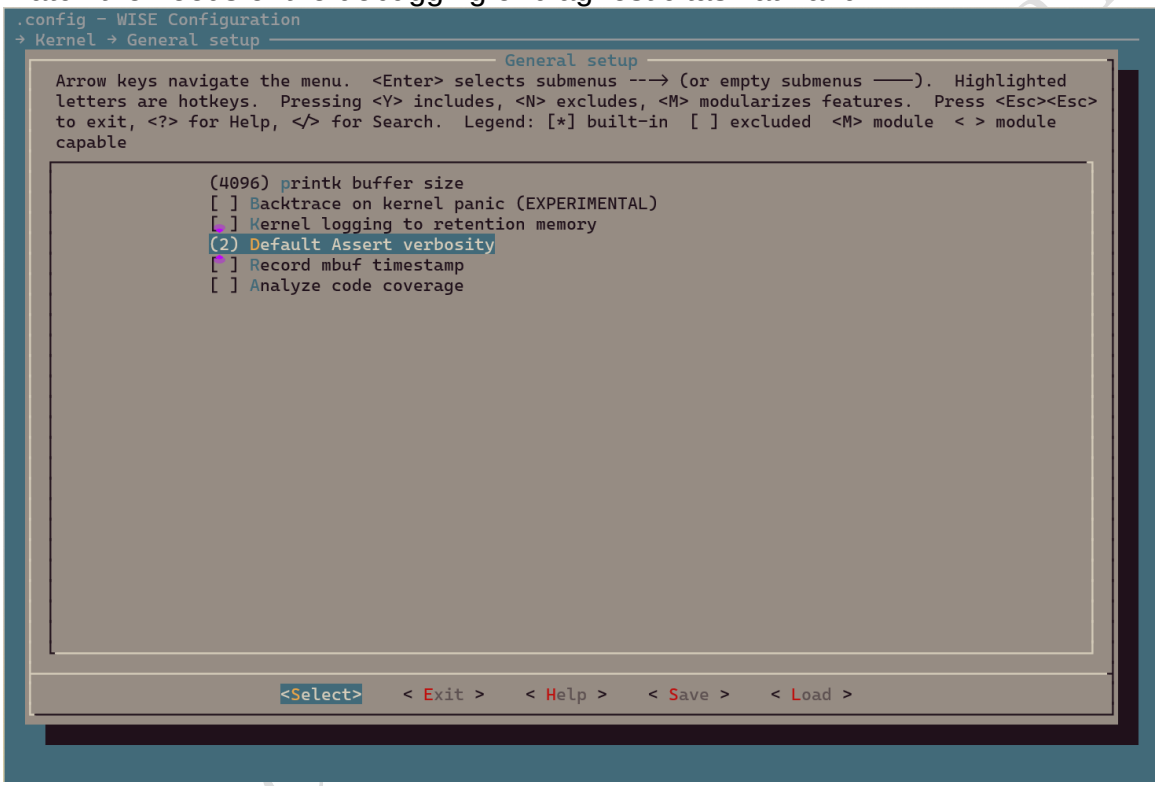
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <cli.h>
#include <hal/rom.h>

#if ASSERT_VERBOSITY == 0
#error "Need to set ASSERT_VERBOSITY as non-zero for this demo."
#endif

int debug_assertion(void)
{
    printf("Assertion demo.\n");

    return 0;
}

/* Simple assertion to check divide-by-zero.
 */

void check_divide_by_zero(int divisor)
{
    assert(divisor != 0);
}
```

Configuring Assertion Verbosity

The verbosity of assertion messages can be set to one of three levels, allowing developers to choose the amount of detail provided during failure handling. This setting is adjustable during the build configuration process as follows:

Navigate to: `Kernel -> General Setup -> Default`

By configuring the verbosity level, developers can tailor the assert output to match the needs of the debugging or diagnostic task at hand.



Assert Verbosity	Display condition of assertion	Display function name and line number	Impact on memory footprint
2 (Default)	Y	Y	High (**)
1	Y	N	Low
0 (*)	N	N	None

(*) Setting verbosity to 0 completely disables all assertions, which is not recommended in general.

(**) It is because potentially lengthy function names will be included in the executable binary.

3.2 Assertion failure from the demo

How to use assertion in an application will be demonstrated by running the demo application. Refer to 1.6 to build and run how-to-debug demo for this purpose.

There will be a CLI command named ‘divide’.

```
WISE 2018.02+ (Apr 19 2024 - 10:26:01 -0700)
Assertion demo.
$
$ help
dhcps               - Configure, start and stop DHCP server
divide              - Divide an integer
dmesg               - display kernel messages
heap                - kernel heap status
help                - print command description and usage
hexdump             - hexdump address size
history             - show/get history
ifconfig            - configure network interfaces
iperf3              - A TCP, UDP, and SCTP network bandwidth measurement tool
irq                 - display irq information
mcuboot_agent       - MCUBoot update agent
mcuboot_confirm     - MCUBoot confirm
mcuboot_set_img     - MCUBoot set image
mcuboot_version     - MCUBoot version
memcmp              - compare memory
net                 - test routines for net (lwIP/net80211/driver)
nullify_ni          - Nullify ni for demo
ping                - send ICMP ECHO_REQUEST to network hosts
pm                  - CLI for PM API test
ps                  - report the current process snapshot
read                - read -(d|b|s|l) address length
reboot              - reboot <n>
rtc_cal             - set RTC calibration interval
top                 - display FreeRTOS tasks
version             - display wise, compiler and linker version
watcher             - CLI commands for WIFI PM
wifi                - CLI for wifi API test
write               - write -(b|s|l) address value
$
$ help divide
Usage: divide <dividend> <divisor>
Divide an integer
$
$
```

It is a very simple CLI command to perform an integer division as follows.

```
WISE 2018.02+ (Apr 19 2024 - 10:26:01 -0700)
Assertion demo.
$
$ help
dhcps               - Configure, start and stop DHCP server
divide              - Divide an integer
dmesg               - display kernel messages
heap                - kernel heap status
help                - print command description and usage
hexdump             - hexdump address size
history             - show/get history
ifconfig            - configure network interfaces
iperf3              - A TCP, UDP, and SCTP network bandwidth measurement tool
irq                 - display irq information
mcuboot_agent       - MCUBoot update agent
mcuboot_confirm     - MCUBoot confirm
mcuboot_set_img     - MCUBoot set image
mcuboot_version     - MCUBoot version
memcmp              - compare memory
net                 - test routines for net (lwIP/net80211/driver)
nullify_ni          - Nullify ni for demo
ping                - send ICMP ECHO_REQUEST to network hosts
pm                  - CLI for PM API test
ps                  - report the current process snapshot
read                - read -(d|b|s|l) address length
reboot              - reboot <n>
rtc_cal             - set RTC calibration interval
top                 - display FreeRTOS tasks
version             - display wise, compiler and linker version
watcher             - CLI commands for WIFI PM
wifi                - CLI for wifi API test
write               - write -(b|s|l) address value
$
$ help divide
Usage: divide <dividend> <divisor>
Divide an integer
$
$
$ divide 100 10
Quotient is 10.
$
```

We can see it throw an assertion failure when we pass 0 as a divisor.

```
---------------------------------------------------------------------------------------------------------------
$ divide 100 0
WISE 2018.02+ riscv32-elf-gcc.gnu (2022-02-07_nds32le-elf-mculib-v5-86807094a2f) 10.3.0
[000005.735229] BOARD: SCM2010 QFN40 EVB V1.0
[000005.735371] VFS: filesystem devfs mounted onto /dev
[000005.737228] PINCTRL: pin controller scm2010,pinctrl registered
[000005.737428] GPIO: scm2010,pinctrl registered as /dev/gpio
[000005.737676] atcwdt: @0xf1300000, clk=32768
[000005.737873] WDT: atcwdt registered as /dev/watchdog
[000005.738206] PINCTRL: atcspi200-xip/cs request pin 11
[000005.738391] PINCTRL: atcspi200-xip/clk request pin 12
[000005.738543] PINCTRL: atcspi200-xip/mosi request pin 13
[000005.738705] PINCTRL: atcspi200-xip/miso request pin 14
[000005.738879] PINCTRL: atcspi200-xip/hold request pin 9
[000005.739045] PINCTRL: atcspi200-xip/wp request pin 10
[000005.739537] EFUSE: efuse-scm2010 registered as /dev/efuse
[000005.739664] trng version : 5e5e0010
[000005.739817] TRNG: trng registered as /dev/trng
[000005.739922] pke version : 0x5e5e0010
[000005.781866] Use fixed MAC address: 64.f9.47.f0.01.20
[000005.782856] 263 usec elapsed in downloading MAC FW.
[000007.077269] Wlan PM ctx init done
[000007.078721] UART: atcuart.0 registered as /dev/ttyS0
[000007.078854] CONSOLE: add /dev/ttyS0
[000007.079249] UART: atcuart.1 registered as /dev/ttyS1
[000007.079362] SOC: SCM2010
[000007.079852] Use fixed BLE public address: 01.02.03.04.05.06
```

```
[000007.082093] ble phy init 35
[000007.082216] PM feature : 0x1d
[000007.082975] PM LS : 6500+1120=7620
[000007.083125] PM SL : 6500+7100=13600
[000007.083273] PM DS : 11500+12800=24300
[000007.083429] PM HB : 1655000+60000=1715000
[000007.083677] PINCTRL: atcuart.0/txd request pin 22
[000007.083829] PINCTRL: atcuart.0/rxd request pin 21
[000336.345105] BUG: assertion(divisor != 0), check_divide_by_zero() at 36
[000336.351793] sp:    0021cc10
[000336.354780] IRQ stack:
[000336.357243]   base: 0021b5b0
[000336.360236]   size: 00000800
[000336.363230] User stack:
[000336.365779]   base: 0021bdc0
[000336.368772]   size: 00000ff0
[000336.371767] 0021cc00: 0021cc10 00203be8 00000800 8008f094 a5a5a5a5 a5a5a5a5 a5a5a5a5 0aa5a5a5
[000336.380772] 0021cc20: a5a5a5a5 a5a5a5a5 8010a2c8 00000001 0000004b 00001880 0021cdd0 0021ce04
[000336.389719] 0021cc40: 00000001 00000000 00000003 00000003 0014d210 0021bdc0 80100000 0021cdb0
[000336.398597] 0021cc60: a5a5a5a5 a5a5a5a5 8010a2c8 00000001 ffffffff 8008e7d8 8010b558 8008f0dc
[000336.407617] 0021cc80: a5a5a5a5 a5a5a5a5 00206f34 0020f1c0 ffffffff 8008e7d8 8010b558 80088df0
[000336.416653] 0021cca0: a5a5a5a5 a5a5a5a5 8010b57c 00000024 ffffffff 00000064 00000000 8008e7d8
[000336.425610] 0021ccc0: ffffffff 0020d408 00000003 80098278 ca2ca324 00210d88 0020d3f8 00210f44
[000336.434588] 0021cce0: 00210f38 00210f3f 00210f43 00000000 00000000 00000000 00000000 00000000
[000336.443430] 0021cd00: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000336.452193] 0021cd20: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000336.460956] 0021cd40: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
[000336.469719] 0021cd60: a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5 80113f84 00210f38 8009865e
[000336.478765] 0021cd80: a5a5a5a5 0020d60c 00202c04 80098672 a5a5a5a5 0020d60c 00202c04 800ff402
[000336.487780] 0021cda0: a5a5a5a5 a5a5a5a5 a5a5a5a5 00000000 a5a5a5a5 a5a5a5a5 a5a5a5a5 a5a5a5a5
[000336.498464]  PID   PR  STWM  S   TASK
[000336.505721]    1    3   532  X   init                        (0x21bdc0-0x21cdb0, 0x21c79c)
[000336.514696]    2    0   964  R   idle                        (0x20fc1c-0x210c10, 0x210b2c)
[000336.523662]    4    3   683  B   knetd                       (0x21d0d0-0x21dcc0, 0x21db7c)
[000336.532629]    3    5   193  B   ksofttimerd                 (0x20f798-0x20fb90, 0x20fabc)
[000336.541595]    7    3   303  B   rt_msg                      (0x221620-0x221c10, 0x221aec)
[000336.550561]    6    3   315  B   knet80211d/wlan0            (0x220d50-0x221340, 0x22123c)
[000336.559527]    8    7    48  B   ll                          (0x222ba0-0x222e10, 0x222d0c)
[000336.568489]    5    3   571  B   scm2020-wlan fast taskq     (0x21f730-0x220120, 0x22001c)
```

Last console buffer content, an user stack content and information of active tasks will also be shown. Refer to 1.4 for details.

Figuring out the call flow which led to this particular assertion can also be done by analyzing the dumped user stack content with wise.dis and wise_rom.dis, if applicable, as illustrated in 2.2.

4 Stack Overrun

4.1 Overview

Stack overrun involves the unintentional modification of stack memory which can occur due to various causes, such as improper pointer handling, task racing for the same data, incorrect data structure sizing, or erroneous memory deallocation. This issue may manifest in multiple symptoms, including system crashes, operational hangs, heap memory leaks, assertion failures, or performance degradation. Resolving such problems typically requires considerable time and a strategic approach using various debugging tools and techniques. This document focuses specifically on stack overrun as a primary example of stack corruption.

Stack overrun occurs when a task or interrupt service routine excessively utilizes its allocated stack memory, exceeding its boundaries. This can be equated to the oversight of enforcing stack size limits during operations.

Despite its potential severity, a stack overrun might not immediately result in a crash, hang, or assertion failure, as illustrated in the subsequent example.

4.2 Demonstration of Stack Overrun

Stack overrun in an application will be demonstrated by running the demo application. Refer to 1.6 to build and run how-to-debug demo for this purpose.

There will be a CLI command named ‘print_string’.

```
WISE 2018.02+ (Apr 19 2024 - 10:26:01 -0700)
Stack overflow demo.
$
$ help
dhcps                - Configure, start and stop DHCP server
dmesg                - display kernel messages
heap                 - kernel heap status
help                 - print command description and usage
hexdump              - hexdump address size
history              - show/get history
ifconfig             - configure network interfaces
iperf3               - A TCP, UDP, and SCTP network bandwidth measurement tool
irq                  - display irq information
mcuboot_agent        - MCUBoot update agent
mcuboot_confirm      - MCUBoot confirm
mcuboot_set_img      - MCUBoot set image
mcuboot_version      - MCUBoot version
memcmp               - compare memory
net                  - test routines for net (lwIP/net80211/driver)
ping                 - send ICMP ECHO_REQUEST to network hosts
pm                   - CLI for PM API test
print_string         - Print a string
ps                   - report the current process snapshot
read                 - read -(d|b|s|l) address length
reboot               - reboot <n>
rtc_cal              - set RTC calibration interval
top                  - display FreeRTOS tasks
version              - display wise, compiler and linker version
watcher              - CLI commands for WIFI PM
wifi                 - CLI for wifi API test
write                - write -(b|s|l) address value
$
$ help print_string
Usage: print_string <string you want to print>
Print a string
$
```

It is a very simple CLI command to print a given string to UART console as follows.

```
WISE 2018.02+ (Apr 19 2024 - 10:26:01 -0700)
Stack overflow demo.
$
$ help
dhcps               - Configure, start and stop DHCP server
dmesg               - display kernel messages
heap                - kernel heap status
help                - print command description and usage
hexdump             - hexdump address size
history             - show/get history
ifconfig            - configure network interfaces
iperf3              - A TCP, UDP, and SCTP network bandwidth measurement tool
irq                 - display irq information
mcuboot_agent       - MCUBoot update agent
mcuboot_confirm     - MCUBoot confirm
mcuboot_set_img     - MCUBoot set image
mcuboot_version     - MCUBoot version
memcmp              - compare memory
net                 - test routines for net (lwIP/net80211/driver)
ping                - send ICMP ECHO_REQUEST to network hosts
pm                  - CLI for PM API test
print_string        - Print a string
ps                  - report the current process snapshot
read                - read -(d|b|s|l) address length
reboot              - reboot <n>
rtc_cal             - set RTC calibration interval
top                 - display FreeRTOS tasks
version             - display wise, compiler and linker version
watcher             - CLI commands for WIFI PM
wifi                - CLI for wifi API test
write               - write -(b|s|l) address value
$
$ help print_string
Usage: print_string <string you want to print>
Print a string
$
$ print_string hello
What you want is " hello "
$
```

But this demo app is a contrived one which has a significant error in its essence by the design.

```c
/* Stack overflow happens in 'init' thread, a.k.a. CLI thread.
 */

static int print_string(const char *whatuwant)
{
    char str[4*2048]; /* Oops! */

    /* Clear the content.
     */
    memset(str, '0', sizeof(str)); /* Oops! */

    sprintf(str, "What you want is \" %s \"\n", whatuwant);

    printf(str);

    return 0;
}

int do_print_string(int argc, char *argv[])
{
    const char *str_to_print;

    argc--, argv++;

    if (argc == 1) {
        str_to_print = argv[0];
        print_string(str_to_print);
    } else {
        return CMD_RET_USAGE;
    }

    return CMD_RET_SUCCESS;
}

CMD(print_string, do_print_string,
    "Print a string",
    "print_string <string you want to print>"
);
```

As you can see above, the function, print_string, uses too much space from its caller's stack, in this case 'init' task's stack, for its use.

As a result, although it looks like the whole system keeps working fine without any issue, the stack of 'init' task has been overrun.

We could verify it by 2 different ways.

The first method is to run 'ps' command and check STWM values.

```
WISE 2018.02+ (Apr 19 2024 - 10:26:01 -0700)
Stack overflow demo.
$
$ help
dhcps                - Configure, start and stop DHCP server
dmesg                - display kernel messages
heap                 - kernel heap status
help                 - print command description and usage
hexdump              - hexdump address size
history              - show/get history
ifconfig             - configure network interfaces
iperf3               - A TCP, UDP, and SCTP network bandwidth measurement tool
irq                  - display irq information
mcuboot_agent        - MCUBoot update agent
mcuboot_confirm      - MCUBoot confirm
mcuboot_set_img      - MCUBoot set image
mcuboot_version      - MCUBoot version
memcmp               - compare memory
net                  - test routines for net (lwIP/net80211/driver)
ping                 - send ICMP ECHO_REQUEST to network hosts
pm                   - CLI for PM API test
print_string         - Print a string
ps                   - report the current process snapshot
read                 - read -(d|b|s|l) address length
reboot               - reboot <n>
rtc_cal              - set RTC calibration interval
top                  - display FreeRTOS tasks
version              - display wise, compiler and linker version
watcher              - CLI commands for WIFI PM
wifi                 - CLI for wifi API test
write                - write -(b|s|l) address value
$
$ help print_string
Usage: print_string <string you want to print>
Print a string
$
$ print_string hello
What you want is " hello "
$
$ ps
 PID   PR  STWM  S  %CPU+      TIME+   TASK
   1    3     0  X    0.1    0:00:01 init                          (0x21bda0-0x21cd90, 0x21c85c)
   2    0   966  R   78.2    0:13:09 idle                          (0x20fc04-0x210c00, 0x210b1c)
   3    5   199  B   21.6    0:03:38 ksofttimerd                   (0x20f780-0x20fb70, 0x20fa9c)
   4    3   683  B    0.0    0:00:00 knetd                         (0x21d0b0-0x21dca0, 0x21db5c)
   6    3   315  B    0.0    0:00:00 knet80211d/wlan0              (0x220d30-0x221320, 0x22121c)
   8    7    48  B    0.0    0:00:00 ll                            (0x222b60-0x222dd0, 0x222ccc)
   5    3   571  B    0.0    0:00:00 scm2020-wlan fast taskq       (0x21f710-0x220100, 0x21fffc)
   7    3   303  B    0.0    0:00:00 rt_msg                        (0x221600-0x221bf0, 0x221acc)
$
```

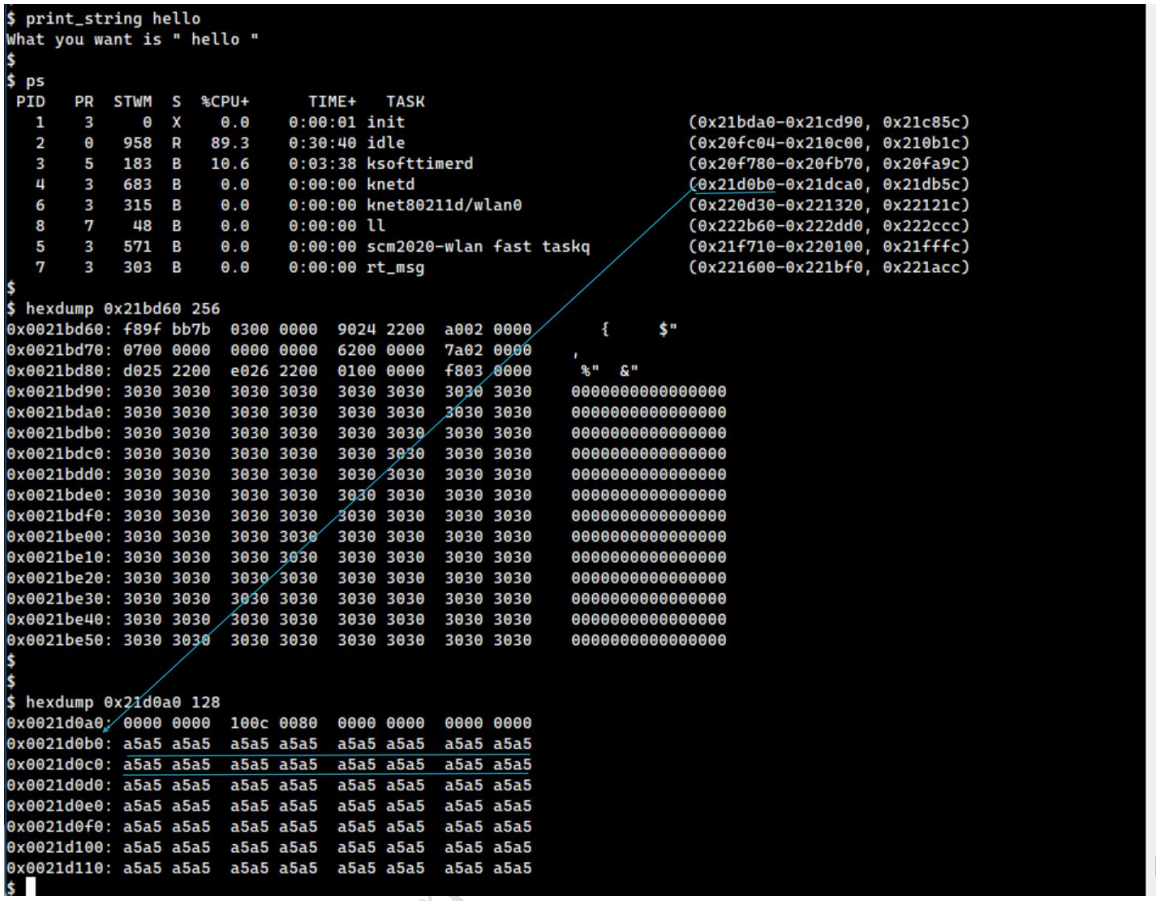
STWM, i.e., stack watermark, of the init task is shown now as 0, which means that at some point there became zero free space left in the stack.
Be aware that we can't rely on the last recorded stack pointer, in this case 0x21c85c, for the purpose of detecting a stack overrun. As you can see, it still legitimately resides inside the original stack space, i.e., between 0x21bda0 and 0x21cd90.
It is because the function's suffix will still return the space of the stack and perform pop operation.

Secondly, we could directly examine the memory region around the specific stack's top, in this case 0x21bda0.

```
$
$ print_string hello
What you want is " hello "
$
$ ps
 PID   PR  STWM  S  %CPU+      TIME+   TASK                                    (0x21bda0-0x21cd90, 0x21c85c)
   1    3     0  X    0.0    0:00:01 init                                     (0x20fc04-0x210c00, 0x210b1c)
   2    0   958  R   89.3    0:30:40 idle                                     (0x20f780-0x20fb70, 0x20fa9c)
   3    5   183  B   10.6    0:03:38 ksofttimerd                              (0x21d0b0-0x21dca0, 0x21db5c)
   4    3   683  B    0.0    0:00:00 knetd                                    (0x220d30-0x221320, 0x22121c)
   6    3   315  B    0.0    0:00:00 knet80211d/wlan0                         (0x222b60-0x222dd0, 0x222ccc)
   8    7    48  B    0.0    0:00:00 ll                                       (0x21f710-0x220100, 0x21fffc)
   5    3   571  B    0.0    0:00:00 scm2020-wlan fast taskq                  (0x221600-0x221bf0, 0x221acc)
   7    3   303  B    0.0    0:00:00 rt_msg
$
$ hexdump 0x21bd60 256
0x0021bd60: f89f bb7b  0300 0000  9024 2200  a002 0000       {     $"
0x0021bd70: 0700 0000  0000 0000  6200 0000  7a02 0000     ,
0x0021bd80: d025 2200  e026 2200  0100 0000  f803 0000     %"  &"
0x0021bd90: 3030 3030  3030 3030  3030 3030  3030 3030    0000000000000000
0x0021bda0: 3030 3030  3030 3030  3030 3030  3030 3030    0000000000000000
0x0021bdb0: 3030 3030  3030 3030  3030 3030  3030 3030    0000000000000000
0x0021bdc0: 3030 3030  3030 3030  3030 3030  3030 3030    0000000000000000
0x0021bdd0: 3030 3030  3030 3030  3030 3030  3030 3030    0000000000000000
0x0021bde0: 3030 3030  3030 3030  3030 3030  3030 3030    0000000000000000
0x0021bdf0: 3030 3030  3030 3030  3030 3030  3030 3030    0000000000000000
0x0021be00: 3030 3030  3030 3030  3030 3030  3030 3030    0000000000000000
0x0021be10: 3030 3030  3030 3030  3030 3030  3030 3030    0000000000000000
0x0021be20: 3030 3030  3030 3030  3030 3030  3030 3030    0000000000000000
0x0021be30: 3030 3030  3030 3030  3030 3030  3030 3030    0000000000000000
0x0021be40: 3030 3030  3030 3030  3030 3030  3030 3030    0000000000000000
0x0021be50: 3030 3030  3030 3030  3030 3030  3030 3030    0000000000000000
$
```

It can be proven that the stack has been overrun.

It is noteworthy that a stack memory will be filled with a specific pattern of bytes, ‘0xa5’, during creation of the corresponding task.
This fact can be illustrated by looking at the upper boundary of an intact stack.



Therefore, seeing any data different from 0xa5 at the upper boundary of a stack proves there has been overrun.

In this example, as mentioned earlier, the whole system goes on without any noticeable defect. It allows us to access CLI commands to investigate further. But if there were an immediate crash, hang-up, or assertion failure after which we can't access interactive CLI any longer, we would need to use alternative methods to move forward such as connecting JTAG debugger to dump the relevant stack memory.

5 Memory Leak

5.1 Overview

scm1612 SDK provides the following functions to allow user applications to allocate and free memory blocks from a dynamic memory pool, i.e., heap.

Function (*)	Header file	Description
void *malloc(size_t sz)	<stdlib.h>	- Allocate a memory block of sz bytes and return its starting address - If 0 is passed as a size, NULL will be returned.
void *zalloc(size_t sz)	<stdlib.h>	- Allocate a memory block of sz bytes and return its starting address - If 0 is passed as a size, NULL will be returned. - If successful, content of the newly allocated block will be cleared to 0 before being returned.
void *calloc(size_t n, size_t sz)	<stdlib.h>	- Allocate a memory block of n * sz bytes and return its starting address - If 0 is passed as a size, NULL will be returned.
void free(void *p)	<stdlib.h>	- Return a previously allocated memory block starting at p to the system heap. - If p is NULL, do nothing.

While a user application is supposed to be written with care not to fail to free any dynamic memory that has been used and is no longer needed, scm1612 SDK provides a CLI command, ‘heap’, to help ensure it.

CLI	Config	Purpose	Description
heap	- Command line interface -> Miscellaneous utilities - -> heap command - Enabled as default	- Understand overall usage of heap - Detect heap memory leakage over time	Shows: - **Current** usage of heap - **Maximum** usage of heap since boot - **Free**, i.e., available space in heap - **Total** size of heap space
heap check (*)	- Command line interface -> Miscellaneous utilities -> heap command - Kernel -> FreeRTOS kernel -> Memory -> Debug enabled malloc/free	- Detect any overrun of boundaries of dynamically allocated memory blocks	- Examines every allocated memory block in the system to see if there is overrun of boundaries. - Specifically checks a head canary pattern, 0xcafe1234, and a tail canary pattern, 0xdeaf5678, of each allocated block. - Displays details regarding the corrupted block, if the check fails.
heap list (**)	- Command line interface -> Miscellaneous utilities -> heap command - Kernel -> FreeRTOS kernel -> Memory ->	- List all memory blocks currently allocated from the heap - Identify blocks that are not recliamed,	Shows: - Index of the block - Name of the owner task - Name of the function that allocated the block, if available - Starting address of the block

	Debug enabled malloc/free - Kernel -> FreeRTOS kernel -> Memory -> Record memory allocated function	across specific operations and/or over time	- Size of the block

(*) ‘heap check’ command might be yet another way to detect stack overrun described in 4 for dynamically allocated stacks.
(**) Name of the function is available only when a function directly calls malloc or its variant. If a function calls kmalloc or its variant, ‘heap list’ will display ‘km-XXXXXXXX’ where 0xXXXXXXXX indicates an address of the next instruction, i.e., the return address, of the jump to kmalloc, by which the context of the caller can be deduced with look-up to wise.dis and/or wise_rom.dis.
If a memory block is allocated from a function bult into the internal ROM of scm1612, the corresponding entry will show ‘ro-XXXXXXXX’ where 0xXXXXXXXX indicates a return address of the specific malloc call.

An example of running these commands is shown below.

```
WISE 2018.02+ (Apr 21 2024 - 14:12:16 -0700)
Heap leak demo.
$
$
$ heap
Current:             31680 B
Maximum:             32272 B
Free:               270416 B
Total:              302096 B
$
$ heap check
Okay.
$
$
$ heap list
[   0] (none)                    ro-8008ef02     address   21de80 size  4096
[   1] (none)                    ro-8008ef10     address   21eec0 size   132
[   2] init                      _if_alloc       address   21ef80 size   428
[   3] init                      ro-0010a484     address   21f160 size   132
[   4] init                      ro-8008ef02     address   21f220 size  3072
[   5] init                      ro-8008ef10     address   21fe60 size   132
[   6] init                      km-0010f152     address   21ff20 size    48
[   7] init                      ro-00117e48     address   21ff90 size    10
[   8] init                      km-800906b6     address   21ffd0 size    24
[   9] init                      km-800906b6     address   220020 size    24
[  10] init                      km-0010f152     address   220070 size    48
[  11] init                      ro-00117e48     address   2200e0 size    14
[  12] init                      km-80093656     address   220130 size    40
[  13] init                      km-800891ec     address   220190 size   412
[  14] init                      spi_create_defa  address   220360 size    16
[  15] init                      km-0010f152     address   2203b0 size    48
```

```
[  16] init                    ro-00117e48     address   220420 size    11
[  17] init                    km-0010f152     address   220460 size    48
[  18] init                    ro-00117e48     address   2204d0 size    10
[  19] init                    km-800906b6     address   220510 size    24
[  20] init                    km-800891ec     address   220560 size  4696
[  21] init                    km-800891ec     address   2217f0 size   640
[  22] init                    km-0010e66c     address   221ab0 size   120
[  23] init                    ro-0010c052     address   221b60 size  2560
[  24] init                    ro-0010c060     address   2225a0 size   124
[  25] init                    _if_alloc       address   222650 size   428
[  26] init                    _if_alloc       address   222830 size  2336
[  27] init                    km-0010e66c     address   223190 size   120
[  28] init                    ro-0010c052     address   223240 size  1536
[  29] init                    ro-0010c060     address   223880 size   124
[  30] init                    ro-0011beaa     address   223930 size     8
[  31] init                    ro-0011beb8     address   223970 size     8
[  32] init                    ro-00117d74     address   2239b0 size   480
[  33] init                    ro-8008ef02     address   223bd0 size  1536
[  34] init                    ro-8008ef10     address   224210 size   132
[  35] init                    scm2020_init_pm  address   2242d0 size   320
[  36] init                    _if_alloc       address   224450 size   428
[  37] init                    _if_alloc       address   224630 size    16
[  38] init                    ro-00117d74     address   224680 size    12
[  39] init                    ifa_alloc       address   2246c0 size   132
[  40] init                    ro-00117d74     address   224780 size    12
[  41] init                    ifa_alloc       address   2247c0 size   132
[  42] init                    _if_alloc       address   2250c0 size   428
[  43] init                    _if_alloc       address   2252a0 size    16
[  44] init                    km-8010737a     address   224880 size   252
[  45] init                    km-0010f152     address   2249b0 size    48
[  46] init                    km-8010737a     address   224d20 size   252
[  47] init                    km-0010f152     address   224e50 size    48
[  48] init                    ro-00117e48     address   224ec0 size    11
[  49] init                    ro-00117e48     address   224a20 size    11
[  50] init                    ro-0010a484     address   224a60 size   212
[  51] init                    ro-0010a484     address   224b70 size   212
[  52] init                    ro-0010c052     address   2252f0 size   640
[  53] init                    ro-0010c060     address   224f00 size   124
[  54] ll                      km-800906b6     address   224c80 size    24
[  55] init                    km-800906b6     address   224fb0 size    24
[  56] init                    ro-0010a484     address   2255b0 size   340
[  57] init                    ro-0010a484     address   225740 size   340
[  58] init                    km-800906b6     address   225000 size    24
[  59] init                    ro-001179a0     address   225050 size    16
[  60] init                    ro-001179a0     address   2258d0 size    16
[  61] init                    ro-001179a0     address   225920 size    16
[  62] init                    ro-00117e48     address   225970 size     5
[  63] init                    ro-00117e48     address   2259b0 size    11
[  64] init                    ro-00117e48     address   2259f0 size    10
$
$
$
```

5.2 Memory leak from the demo

Memory leak in an application will be demonstrated by running the demo application. Refer to 1.6 to build and run how-to-debug demo for this purpose. Besides what has been described in 1.6, there are 2 more adjustments to be done to build the demo application for the purpose of this section.

- Select Kernel -> FreeRTOS kernel -> Memory option -> Debug enabled malloc/free.

- Select Kernel -> FreeRTOS kernel -> Memory option -> Record memory allocated function.
- (*) Unselect Command line interface -> Command history.

(*) Enabling command history will use additional heap memory, which will make this discussion obfuscated unncessarily.

There will be a group of CLI commands, ‘save_string’, ‘purge_string’, and ‘list_string’.

```
WISE 2018.02+ (Apr 21 2024 - 14:12:16 -0700)
Heap leak demo.
$
$ help
dhcps                - Configure, start and stop DHCP server
dmesg                - display kernel messages
heap                 - kernel heap status
help                 - print command description and usage
hexdump              - hexdump address size
history              - show/get history
ifconfig             - configure network interfaces
iperf3               - A TCP, UDP, and SCTP network bandwidth measurement tool
irq                  - display irq information
list_string          - List saved string(s)
mcuboot_agent        - MCUBoot update agent
mcuboot_confirm      - MCUBoot confirm
mcuboot_set_img      - MCUBoot set image
mcuboot_version      - MCUBoot version
memcmp               - compare memory
net                  - test routines for net (lwIP/net80211/driver)
ping                 - send ICMP ECHO_REQUEST to network hosts
pm                   - CLI for PM API test
ps                   - report the current process snapshot
purge_string         - Purge a string
read                 - read -(d|b|s|l) address length
reboot               - reboot <n>
rtc_cal              - set RTC calibration interval
save_string          - Save a string
top                  - display FreeRTOS tasks
version              - display wise, compiler and linker version
watcher              - CLI commands for WIFI PM
wifi                 - CLI for wifi API test
write                - write -(b|s|l) address value
$
$ help save_string
Usage: save_string <string you want to save>
Save a string
$
$ help purge_string
Usage: purge_string <index returned from save_string>
Purge a string
$
$ help list_string
Usage: list_string (<index>)
List saved string(s)
$
$
```

The commands are described in the below table, which is followed by an example of running them.

Command	Description	Notes
save_string	- Saves a string literal into an array.	- Up to 10 strings can be saved.

	- Returns an index of the array where the string has been stored.	- Any excessive string will be discarded.
purge_string	- Removes a string literal corresponding to the given index from the array and reclaim the associated memory.	
list_string	- List content of the string array.	

```
WISE 2018.02+ (Apr 22 2024 - 11:20:31 -0700)
Heap leak demo.
$
$ help
dhcps                - Configure, start and stop DHCP server
dmesg                - display kernel messages
heap                 - kernel heap status
help                 - print command description and usage
hexdump              - hexdump address size
history              - show/get history
ifconfig             - configure network interfaces
iperf3               - A TCP, UDP, and SCTP network bandwidth measurement tool
irq                  - display irq information
list_string          - List saved string(s)
mcuboot_agent        - MCUBoot update agent
mcuboot_confirm      - MCUBoot confirm
mcuboot_set_img      - MCUBoot set image
mcuboot_version      - MCUBoot version
memcmp               - compare memory
net                  - test routines for net (lwIP/net80211/driver)
ping                 - send ICMP ECHO_REQUEST to network hosts
pm                   - CLI for PM API test
pmp                  - CLI for PM debug
ps                   - report the current process snapshot
purge_string         - Purge a string
read                 - read -(d|b|s|l) address length
reboot               - reboot <n>
rtc_cal              - set RTC calibration interval
save_string          - Save a string
top                  - display FreeRTOS tasks
version              - display wise, compiler and linker version
watcher              - CLI commands for WIFI PM
wifi                 - CLI for wifi API test
write                - write -(b|s|l) address value
$
$ list_string
$
$ save_string hello
Saved hello to 0.
$ save_string hello1
Saved hello1 to 1.
$ save_string hello2
Saved hello2 to 2.
$
$ list_string
[00] hello
[01] hello1
[02] hello2
$
$ purge_string 0
$
$ list_string
[01] hello1
[02] hello2
$
```

The demo application has an intentionally planted bug, which will incur heap memory leak when we try to save more than the maximum number, ten in this case, of strings.

```
/* Demonstrate how to use the 'heap list' command.
 */

static char *saved_string[10] = {0,};

static int save_string(const char *whatusave)
{
    char *str;
    int i;

    str = zalloc(strlen(whatusave) + 1);
    strncpy(str, whatusave, strlen(whatusave));

    for (i = 0; i < 10; i++) {
        if (saved_string[i] == NULL) {
            saved_string[i] = str;
            break;
        }
    }

    if (i == 10) {
        /* Oops, we forgot to free str.
         */
        return -1;
    }

    return i;
}
```

How this error would be detected by using ‘heap’ CLI command will be illustrated.

Firstly, we can see there will be no leak from the heap unless we reach the maximum number of strings to save.

```
WISE 2018.02+ (Apr 22 2024 - 12:32:26 -0700)
Heap leak demo.
$
$ heap
Current:              31568 B
Maximum:              34640 B
Free:                270480 B
Total:               302048 B
$
$ save_string hello
Saved hello to 0.
$ save_string hello1
Saved hello1 to 1.
$ save_string hello2
Saved hello2 to 2.
$ save_string hello3
Saved hello3 to 3.
$
$ list_string
[00] hello
[01] hello1
[02] hello2
[03] hello3
$
$ heap
Current:              31824 B
Maximum:              34640 B
Free:                270224 B
Total:               302048 B
$
$ purge_string 0
$ purge_string 1
$ purge_string 2
$ purge_string 3
$
$ list_string
$
$ heap
Current:              31568 B
Maximum:              34640 B
Free:                270480 B
Total:               302048 B
$
$
```

But the above-mentioned error takes effect once we reach the maximum number of strings to save as below.

```
$ list_string
$
$ heap
Current:              31568 B
Maximum:              34640 B
Free:                270480 B
Total:               302048 B
$
$
$ save_string hello
Saved hello to 0.
$ save_string hello1
Saved hello1 to 1.
$ save_string hello2
Saved hello2 to 2.
$ save_string hello3
Saved hello3 to 3.
$ save_string hello4
Saved hello4 to 4.
$ save_string hello5
Saved hello5 to 5.
$ save_string hello6
```

```
Saved hello6 to 6.
$ save_string hello7
Saved hello7 to 7.
$ save_string hello8
Saved hello8 to 8.
$ save_string hello9
Saved hello9 to 9.
$ save_string hello10
Couldn't save hello10 because there is no empty slot.
$ save_string hello11
Couldn't save hello11 because there is no empty slot.
$ save_string hello12
Couldn't save hello12 because there is no empty slot.
$ save_string hello13
Couldn't save hello13 because there is no empty slot.
$
$ list_string
[00] hello
[01] hello1
[02] hello2
[03] hello3
[04] hello4
[05] hello5
[06] hello6
[07] hello7
[08] hello8
[09] hello9
$
$ heap
Current:             32464 B
Maximum:             34640 B
Free:               269584 B
Total:              302048 B
$
$ purge_string 0
$ purge_string 1
$ purge_string 2
$ purge_string 3
$ purge_string 4
$ purge_string 5
$ purge_string 6
$ purge_string 7
$ purge_string 8
$ purge_string 9
$
$ list_string
$
$ heap
Current:             31824 B
Maximum:             34640 B
Free:               270224 B
Total:              302048 B
$
$
```

Because all 10 stored strings were purged in the above example, the usage of the heap must be same. But above result shows otherwise because we failed to free 4 strings that could not be stored, again due to the bug we mentioned above.

Comparing the available space in heap can indicate that there is some memory leak as shown above. But it is not saying anything about any further details.

To list all the memory blocks outstanding in the system will help us understand what is going on. ‘heap list’ command comes into play for this reason.

```
$ heap list
[   0] (none)                    ro-8008f122     address   21deb0 size  4096
[   1] (none)                    ro-8008f130     address   21eef0 size   132
[   2] init                      _if_alloc       address   21efb0 size   428
[   3] init                      ro-0010a484     address   21f190 size   132
[   4] init                      ro-8008f122     address   21f250 size  3072
[   5] init                      ro-8008f130     address   21fe90 size   132
[   6] init                      km-0010f152     address   21ff50 size    48
[   7] init                      ro-00117e48     address   21ffc0 size    10
[   8] init                      km-800908d0     address   220000 size    24
[   9] init                      km-800908d0     address   220050 size    24
[  10] init                      km-0010f152     address   2200a0 size    48
[  11] init                      ro-00117e48     address   220110 size    14
[  12] init                      km-80093c10     address   220160 size    40
[  13] init                      km-80089214     address   2201c0 size   412
[  14] init                      spi_create_defa  address   220390 size    16
[  15] init                      km-0010f152     address   2203e0 size    48
[  16] init                      ro-00117e48     address   220450 size    11
[  17] init                      km-0010f152     address   220490 size    48
[  18] init                      ro-00117e48     address   220500 size    10
[  19] init                      km-800908d0     address   220540 size    24
[  20] init                      km-80089214     address   220590 size  4696
[  21] init                      km-80089214     address   221820 size   640
[  22] init                      km-0010e66c     address   221ae0 size   120
[  23] init                      ro-0010c052     address   221b90 size  2560
[  24] init                      ro-0010c060     address   2225d0 size   124
[  25] init                      _if_alloc       address   222680 size   428
[  26] init                      _if_alloc       address   222860 size  2336
[  27] init                      km-0010e66c     address   2231c0 size   120
[  28] init                      ro-0010c052     address   223270 size  1536
[  29] init                      ro-0010c060     address   2238b0 size   124
[  30] init                      ro-0011beaa     address   223960 size     8
[  31] init                      ro-0011beb8     address   2239a0 size     8
[  32] init                      ro-00117d74     address   2239e0 size   480
[  33] init                      ro-8008f122     address   223c00 size  1536
[  34] init                      ro-8008f130     address   224240 size   132
[  35] init                      scm2020_init_pm  address   224300 size   320
[  36] init                      _if_alloc       address   224480 size   428
[  37] init                      _if_alloc       address   224660 size    16
[  38] init                      ro-00117d74     address   2246b0 size    12
[  39] init                      ifa_alloc       address   2246f0 size   132
[  40] init                      km-801079f4     address   2247b0 size   252
[  41] init                      km-0010f152     address   2248e0 size    48
[  42] init                      _if_alloc       address   2250f0 size   428
[  43] init                      _if_alloc       address   2252d0 size    16
[  44] init                      ro-00117d74     address   225320 size    12
[  45] init                      ifa_alloc       address   225360 size   132
[  46] init                      ro-00117e48     address   224950 size    11
[  47] init                      ro-0010a484     address   224990 size   212
[  48] init                      km-801079f4     address   225d60 size   252
[  49] init                      km-0010f152     address   225e90 size    48
[  50] init                      ro-00117e48     address   225f00 size    11
[  51] init                      ro-0010a484     address   224aa0 size   212
[  52] init                      ro-0010c052     address   224bb0 size   640
[  53] init                      ro-0010c060     address   224e70 size   124
[  54] ll                        km-800908d0     address   224f20 size    24
[  55] init                      km-800908d0     address   224f70 size    24
[  56] init                      ro-0010a484     address   225420 size   340
[  57] init                      ro-0010a484     address   2255b0 size   340
[  58] init                      km-800908d0     address   224fc0 size    24
[  59] init                      ro-001179a0     address   225010 size    16
[  60] init                      ro-001179a0     address   225060 size    16
[  61] init                      ro-001179a0     address   225740 size    16
[  62] init                      save_string     address   225a10 size     8
[  63] init                      save_string     address   225a50 size     8
[  64] init                      save_string     address   225a90 size     8
[  65] init                      save_string     address   225ad0 size     8
$
```

4 memory blocks are found to be outstanding, i.e., not reclaimed, where they should have been so.

While this example may look too contrived, using ‘heap’ and ‘heap list’ commands will greatly help resolve much harder memory leak issues.