EMBEDDED OPEN SOURCE SUMMIT

Overview of Logging

- Aastha Grover, Intel Corporation

intel

Zephyr® Project Developer Summit 2023

Agenda

- Overview
- Log API's
- Architecture of Logging Subsystem
- Log Message and Memory
- Log Filtering
- Modes of Logging
- Multi Domain Logging
- Types of Logging formats: pros, cons
- Switch Formats at Runtime

Overview

- Logging outputs Human readable strings.
  - printf like results + more.
  - String formatting – cbprintf package
  - Supports all format specifiers.
- Identify source of Logging
  - Source module, severity level, time of generation (Timestamp)
- Timestamping
- Can log from any context – Interrupt context
- Functionality to dump data
  - LOG_HEXDUMP
- Log API for every Severity levels.
  - Information, Debugging, Warning & Error
- Supports Multiple output channels. Example:
  - UART
  - Filesystem,
  - Remote (Bluetooth, Net)
  - Multiple backends

How to Log using Zephyr?

Source Code:

```
#include <zephyr/logging/log.h>

LOG_MODULE_REGISTER(MODULE,Level);

LOG_INF(”String");
```

Kconfig:

```
CONFIG_LOG=y
```

Example: Hello World Application
zephyr/samples/hello_world

```
#include <zephyr/kernel.h>
#include <zephyr/logging/log.h>

LOG_MODULE_REGISTER(hello_world, LOG_LEVEL_INF);

int main(void)
{
        printk("Hello World! %s\n", CONFIG_BOARD);
        LOG_INF("Hello please");
        return 0;
}
```

Output ->

```
Booting from ROM..
*** Booting Zephyr OS build zephyr-v3.3.0-3615-gb21496f9d0fa ***
Hello World! qemu_x86
[00:00:00.010,000] <inf> hello_world: Hello please
```

Timestamp <level> <module name> <Message>

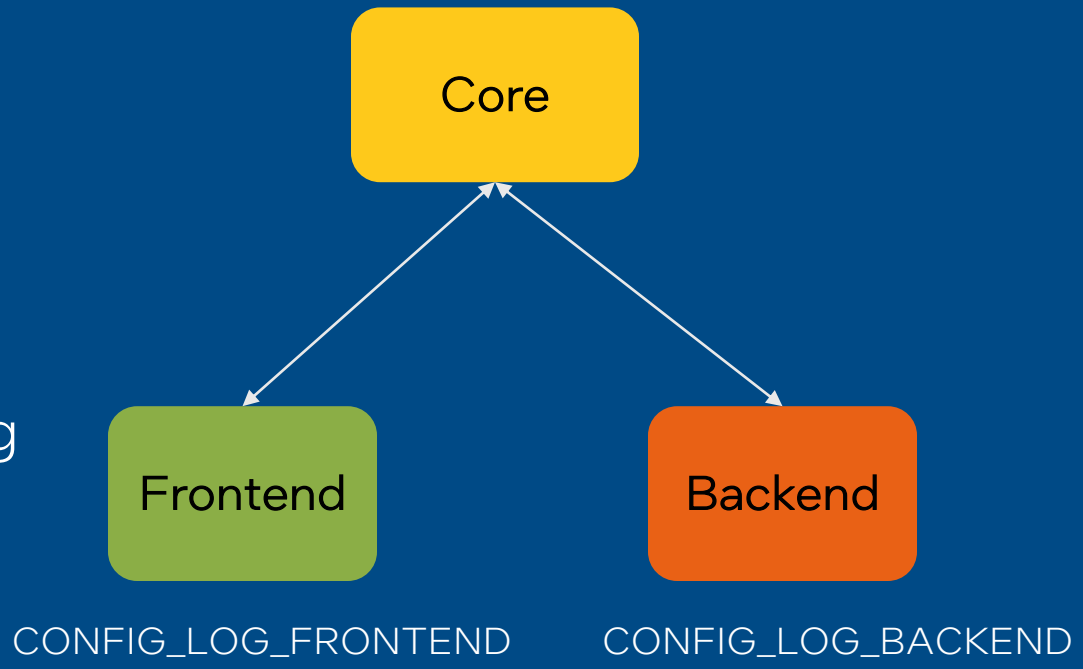
Logger API's

X = ERR, WRN, DBG, INFO

- LOG_X : Standard printf-like messages. Eg: LOG_ERR
- LOG_INST_X : Associated with the particular instance. Eg: LOG_INST_INFO
- LOG_HEXDUMP_X : Dump hex data.
- LOG_INST_HEXDUMP_X: Dump hex data associated with a particular instance.
- LOG_PRINTK /LOG_RAW: Unconditionally print raw log message.
- LOG_MODULE_REGISTER: Creates module-specific state and register the module with Logger.
- LOG_MODULE_DECLARE: Declares a log module (not register it).

Architecture

- Frontend
  - Engaged by default when LOG API is called.
  - Responsible for log filtering (Compile & Run-Time), buffer allocation, creating & committing the message.
  - Optimized to log messages faster – interrupt context.
- Core
  - Macros and functions needed for creating logging messages
  - Enables/disables logging formats.
- Backends
  - Outputs the log messages
  - Cannot be selected by user.
  - Messages are passed through a frontend and are then processed by active backends.



Frontend: Logging Process

- Filtering a message
  - Compile Time Filtering
  - Run Time Filtering
- Allocating buffer for the message
- Message Creation
- Commit the Message

Log Message

- Self-contained, continuous block of memory
- Log message contents:
  - Message Descriptor (Source, Domain & Level)
  - Timestamp
  - Formatted String details
  - Optional data

Log Message Format

- Message Header
  - MPSC packet buffer header: 2 bits
  - Trace/Log message flag : 1 bit
  - Domain ID: 3 bits
  - Level: 3 bits
  - Cbprintf Package Length: 10 bits
  - Data length: 12 bits
  - Reserved: 1 bit
  - Pointer: Pointer to the source descriptor
  - Timestamp: 32 or 64 bits
  - Optional padding
- Cbprintf package (optional) – Header, Arguments & Appended strings
- Hexdump data (optional)
- Alignment padding(optional)

Log Message Memory

- Messages stored in Multi Producer Single Consumer Packet Buffer (MPSC_PBUF) – circular buffer as continuous block of memory
- Messages must be sequentially freed – suited for copying the messages for offline processing.
- Backend processing is synchronous. Backend can make a copy for deferred processing.

Multi Producer Single Consumer Packet Buffer¶

- Circular Buffer - First-in First Out Order, Continous block of memory.
- Allocation Policy, when requested space cannot be allocated:
  - Overwrite - CONFIG_LOG_MODE_OVERFLOW (Degrades Performance)
  - No-overwrite
- Producing the packet – 2 steps
  - Requested amount of data allocated, producer fills the data
  - Commits it
- Consuming the packet - 2 steps
  - Consumer claims the packet (gets pointer to it and length)
  - Packet is freed (Reduces memory copying)

Multi Producer Single Consumer Packet Buffer(Contd..)

- Each packet contains MPSC_PBUF specific header.
- Header – 2 bits
  - Valid
  - Busy (Packet being consumed)

Valid	Busy	Description
0	0	Space is Free
1	0	Valid Packet
1	1	Claimed valid Packet
0	1	Internal Skip Packet

Log Filtering

- Need?
  - To reduce Image size, to not overload the system
- Compile Time Filtering
  - Module
  - Severity Levels
- RunTime Filtering
  - Source : Module or specific instance of module
  - Independent for each backend

Run-Time Filtering

- Filter structure (Ten 3bit slots in RAM) for each source of logging:

Slot 0	INF
Slot 1	ERR
Slot 2	INF
Slot 3	OFF
Slot 4	…
Slot 5	…
Slot 6	…
Slot 7	…
Slot 8	…
Slot 9	OFF

→ Aggregate maximal filter for given source

Each slot stores current filter for one backend in the system

Modes of Logging

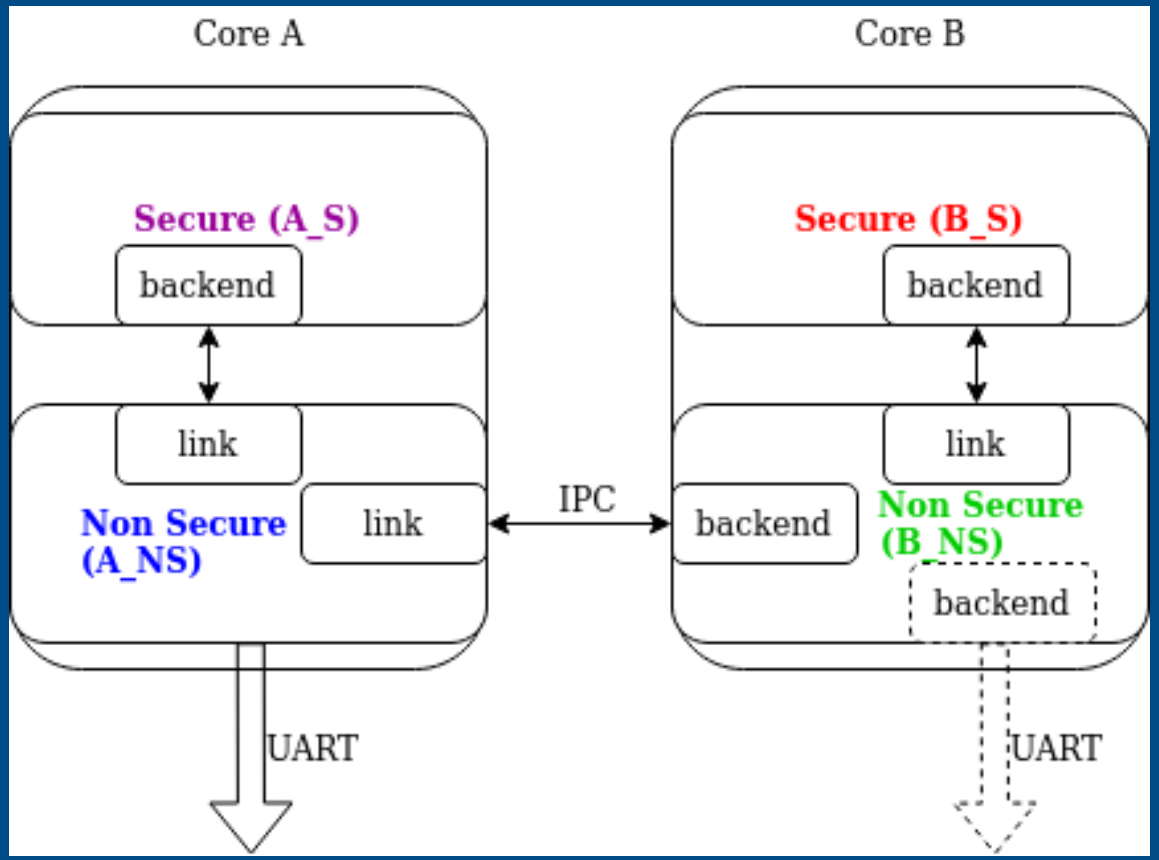
Deferred	Immediate	Minimal
• Log messages are buffered and processed later.	• Log is processed in the context of the call. Immediate output.	• Redirected to printk
• Time consuming processing is deferred to the known context. Least impact on the application.	• Impacts performance, time consuming operations performed in context of the log entry (e.g. high priority interrupt)	• No run time filtering, formatting, timestamping
• Build-time, run-time filtering	• Filtering and formatting	• Low Footprint
• Rich formatting, timestamping.	• Intrusive	
• Close to non-intrusive	• Limited backends supported	
• Any backend		

Multi-Domain Logging

- Domain : Independent Binary Build
  - Restricted Memory access – cross domains
- Approaches:
  1. Inside each domain independently.
     - Not scalable
  2. Use a multi-domain logging system.
     - Log messages inside each domain goes to one root domain.
     - Passed using link between domains. One backend -> Other backend.
- Log Link Interface:
  - Receives log messages from another domain.
  - Creates copy & put local log message copy into message queue.
  - Matches complementary backend implementation.

Multi-Domain Logging (Contd..)

- Types of Domains:
  - End Domain:
    - Logging core implementation & a cross-domain backend.
  - Relay Domain
    - Links to other domains but does not have backends.
    - Has a cross-domain backend either to another relay or to the root domain.
  - Root Domain
    - One or multiple links and a backend that outputs logs to the user.
- Runtime - filtering works identically in both multi-domain and single-domain scenarios.



Types of Logging Formats

- MIPI-SyS-T: `CONFIG_LOG_MIPI_SYST_ENABLE`
  - Universal data format for sharing software debug and trace information between test systems and devices such as systems-on-chip (SoC) or platforms.
  - Simplify Integration of embedded software and debug hardware
  - Developing test products
- Dictionary: `LOG_DICTIONARY_SUPPORT`
  - Outputs in binary format.
  - Encodes arguments to formatted strings in native storage formats - compact
  - long double - Not supported.
- Text: Default
  - Requires less processing than dict.
- Custom Format: `CONFIG_LOG_CUSTOM_FORMAT_SUPPORT`

Dynamically Switch Logging formats

- User can pass the output format needed and use API's like
  - `log_format_set(const struct log_backend *backend, uint32_t log_type)`
  - `log_format_set_all_active_backends(size_t log_type)`
- Limitations – Deferred mode.
  - Results in dropped messages – large number of messages to be printed.
  - Won't be able to see complete results.
- Recommendations:
  - Use Immediate Mode

SYS-T RAW DATA: 020A000B42000200000000000000002A2A2A20426F6F74696E67205A6570687972204F53206275696C64207A65706879722D76332E332E302D333631352D6762323134393666396430666120 2A2A2A0A00
SYS-T RAW DATA: 220A000B17000200000000000000004572726F72206D657373616765206578616D706C652E00
SYS-T RAW DATA: 320A000B19000200000000000000005761726E696E67206D657373616765206578616D706C652E00
SYS-T RAW DATA: 420A000B16000200000000000000496E666F206D657373616765206578616D706C652E00
SYS-T RAW DATA: 720A000B17000200000000000000446562756720 6D657373616765206578616D706C652E00
SYS-T RAW DATA: 720A000B1E000200000000000000446562756720 6D657373616765206578616D706C652C202640001000000
SYS-T RAW DATA: 720A000B26000200000000000000446562756720 6D657373616765206578616D706C652C2025642C202564000100000002000000
SYS-T RAW DATA: 720A000B2E000200000000000000446562756720 6D657373616765206578616D706C652C2025642C2025642C202564000100000002000000030000000
SYS-T RAW DATA: 720A000B38000200000000000000446562756720 6D657373616765206578616D706C652C2025642C2025642C2025642C203078257800010000000200000003000000040000000
SYS-T RAW DATA: 720A000B0C000200000000000006368617220256300210000000
SYS-T RAW DATA: 720A000B1D000200000000000007320737472202573202573007374617469632073747200632073747200
SYS-T RAW DATA: 720A000B15000300000000000006420737472202573006479 6E616D69632073747200
SYS-T RAW DATA: 720A000B6B000300000000000006D6978656420737472202573202573202573202573202573202573202573006479 6E616D69632073747200 2D2D2D0064796E616D69632073747200 2D2D2D00616E6F746865722064796E616D69632073747200
SYS-T RAW DATA: 720A000B43000300000000000006D6978656420632F732025632025732025732025732025630021000000737461746963207374720064796E616D69632073747200737461746963207374720021000000
SYS-T RAW DATA: 720A000B22000300000000000000446562756720 6D657373616765206578616D706C652C20256600EA2E4454FB210940
SYS-T RAW DATA: 220A000B09000300000000000002573006672616D6500
SYS-T RAW DATA: 220A000B47000300000000000002573003033206432203034203030203030203038203031203032202030332030342030352030362030372030382020202020207C2E2E2E2E2E2E2E2E202E2E2E2E2E2E2E202000
SYS-T RAW DATA: 320A000B09000300000000000002573006672616D6500
SYS-T RAW DATA: 320A000B47000300000000000002573003033206432203034203030203030203038203031203032202030332030342030352030362030372030382020202020207C2E2E2E2E2E2E2E2E202E2E2E2E2E2E2E202000
SYS-T RAW DATA: 420A000B09000300000000000002573006672616D6500
SYS-T RAW DATA: 420A000B47000300000000000002573003033206432203034203030203030203038203031203032202030332030342030352030362030372030382020202020207C2E2E2E2E2E2E2E2E202E2E2E2E2E2E2E202000
SYS-T RAW DATA: 720A000B09000300000000000002573006672616D6500
SYS-T RAW DATA: 720A000B47000300000000000002573003033206432203034203030203030203038203031203032202030332030342030352030362030372030382020202020207C2E2E2E2E2E2E2E2E202E2E2E2E2E2E2E202000
SYS-T RAW DATA: 020A000B22000400000000000000068656C6C6F207379732D74206F6E20626F6172642025730A0071656D755F78383600
[00:00:00.040,000] <err> syst: Error message example.
[00:00:00.040,000] <wrn> syst: Warning message example.
[00:00:00.040,000] <inf> syst: Info message example.
[00:00:00.040,000] <dbg> syst: Debug message example.
[00:00:00.040,000] <dbg> syst: Debug message example, 1
[00:00:00.040,000] <dbg> syst: Debug message example, 1, 2
[00:00:00.040,000] <dbg> syst: Debug message example, 1, 2, 3
[00:00:00.040,000] <dbg> syst: Debug message example, 1, 2, 3, 0x4
[00:00:00.040,000] <dbg> syst: char !
[00:00:00.040,000] <dbg> syst: s str static str c str
[00:00:00.040,000] <dbg> syst: d str dynamic str
[00:00:00.040,000] <dbg> syst: mixed str dynamic str --- dynamic str --- another dynamic str --- another dynamic str
[00:00:00.040,000] <dbg> syst: mixed c/s ! static str dynamic str static str !
[00:00:00.040,000] <dbg> syst: Debug message example, %f
[00:00:00.040,000] <err> syst: frame
                           03 d2 04 00 00 08 01 02 03 04 05 06 07 08 |........ ......
[00:00:00.040,000] <wrn> syst: frame
                           03 d2 04 00 00 08 01 02 03 04 05 06 07 08 |........ ......
[00:00:00.050,000] <inf> syst: frame
                           03 d2 04 00 00 08 01 02 03 04 05 06 07 08 |........ ......
[00:00:00.050,000] <dbg> syst: frame
                           03 d2 04 00 00 08 01 02 03 04 05 06 07 08 |........ ......
hello sys-t on board qemu_x86
SYS-T RAW DATA: 220A000B17000500000000000000004572726F72206D657373616765206578616D706C652E00
SYS-T RAW DATA: 320A000B19000500000000000000005761726E696E67206D657373616765206578616D706C652E00
SYS-T RAW DATA: 420A000B16000500000000000000496E666F206D657373616765206578616D706C652E00
SYS-T RAW DATA: 720A000B17000500000000000000446562756720 6D657373616765206578616D706C652E00
SYS-T RAW DATA: 720A000B1E000500000000000000446562756720 6D657373616765206578616D706C652C202640001000000
SYS-T RAW DATA: 720A000B26000500000000000000446562756720 6D657373616765206578616D706C652C2025642C202564000100000002000000
SYS-T RAW DATA: 720A000B2E000500000000000000446562756720 6D657373616765206578616D706C652C2025642C2025642C202564000100000002000000030000000
SYS-T RAW DATA: 720A000B38000500000000000000446562756720 6D657373616765206578616D706C652C2025642C2025642C2025642C203078257800010000000200000003000000040000000
SYS-T RAW DATA: 720A000B0C000500000000000006368617220256300210000000
SYS-T RAW DATA: 720A000B1D000500000000000007320737472202573202573007374617469632073747200632073747200
SYS-T RAW DATA: 720A000B15000500000000000006420737472202573006479 6E616D69632073747200
SYS-T RAW DATA: 720A000B6B000500000000000006D6978656420737472202573202573202573202573202573202573202573006479 6E616D69632073747200 2D2D2D0064796E616D69632073747200 2D2D2D00616E6F746865722064796E616D69632073747200
SYS-T RAW DATA: 720A000B43000600000000000006D6978656420632F732025632025732025732025732025630021000000737461746963207374720064796E616D69632073747200737461746963207374720021000000
SYS-T RAW DATA: 720A000B22000600000000000000446562756720 6D657373616765206578616D706C652C20256600EA2E4454FB210940

MIPI SyS-T

Text

MIPI SyS-T

References

- Logging Documentation
- Mailing Lists:
  - User List: users@lists.zephyrproject.org
  - Developer List: devel@lists.zephyrproject.org
- Discord Channel:
  - #logging
  - #tracing
- Contributing to Zephyr: See the Contribution Guide

Questions?
Comments?

Thank You
- Aastha Grover


intel