Inside Log

Walter Zhou

walterzh@marvell.com

2015-03-31

Warning

Don't use **dbg_printf()** directly unless you know what you are doing.

Because user almost could not disable the log from dbg_printf().

How to disable debug output in DEBUG version

==> dbg enable 0

```
static void _add_char(char c, void* unused)
{
  if (_printf_enable) dbg_outbuf_add_char(c);
}
```

if _printf_enable is 0, the log data will not be added to output buffer.

Debug log output

Log output requester and log output thread are running different thread, so the log output is asynchronous essentially.

But programmer could make synchronous partly by log command.

Log output asynchronously or synchronous

==> dbg blocking 0 / 1

if dbg blocking 0, the log output and log requester are asynchronous; otherwise, they are synchronous.

In theory, **isr** could output log by invoking dbg_printf --- no matter what, dbg blocking is 0 or 1, the output log in **isr** is always asynchronous, so it is safe to invoke dbg_printf in isr.

But...

Log output from UART is slow

Please don't forget, output log by UART is very very slow. Usually, it take nearly 90 microseconds to output one character.

in 9600 baud rate, it take nearly 1000 microseconds (1 millisecond) to output one character.

The UART in our ASIC is work at 115200 baud rate (9600 * 12)

The isr needs response quickly as soon as possible. I prefer output log by **logStringImpl()** API

void logStringImpl(const char *pFormat, ...);

You could get the logs by slog command.

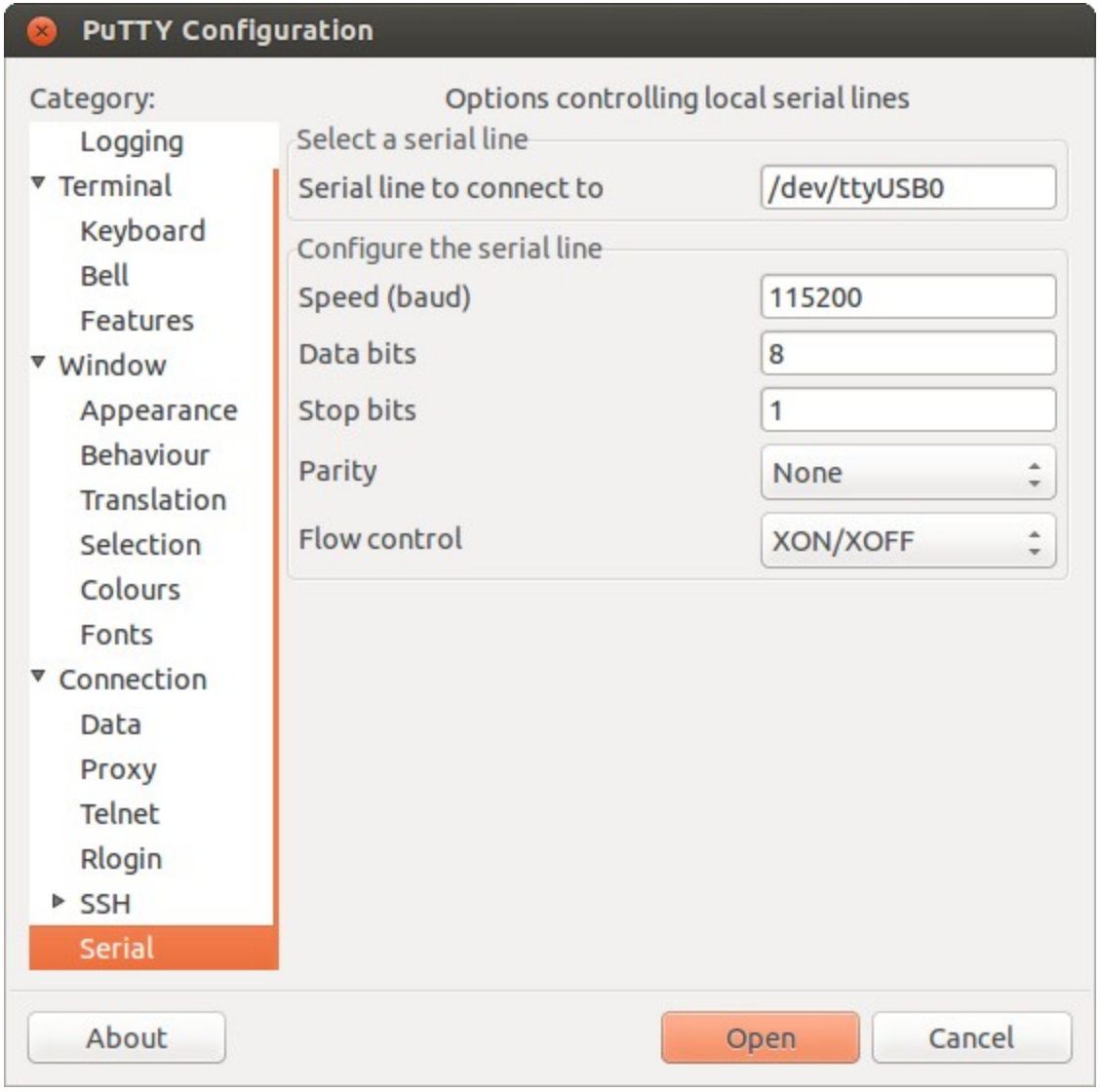
UART config

in dbg.c

```
static void* _uart;
static UART_CONFIG _config = { 115200,
                               UART_DATA_BITS_8,
                               UART_STOP_BITS_1,
                               UART_PARITY_NONE };

uartOpen(hwGetDebugUARTNumber(), &_uart);
uartSetConfig(_uart, &_config);
```

The configuration for debug uart is hardcode.


PuTTY Configuration
Category:
Logging
Terminal
Keyboard
Bell
Features
Window
Appearance
Behaviour
Translation
Selection
Colours
Fonts
Connection
Data
Proxy
Telnet
Rlogin
SSH
Serial
Options controlling local serial lines
Select a serial line
Serial line to connect to
/dev/ttyUSB0
Configure the serial line
Speed (baud)
115200
Data bits
8
Stop bits
1
Parity
None
Flow control
XON/XOFF
About
Open
Cancel

Log into memory

We could add log into ring-buffer (memory)

```
#include "logger.h"

void logStringImpl(char *pFormat, ...)

for example:

void job_attr_register(map_handle_t job_attr_map)
{
    uint32_t i;

    ASSERT(job_attr_map != MAP_INVALID_HANDLE);

    for (i = 0; i < num_job_attrs; i++)
    {
        ASSERT(map_lookup(job_attr_map, job_attrs[i].name, strlen(job_attrs[i].name)) ==
      NULL);
        REL_ASSERT(map_insert(job_attr_map, job_attrs[i].name, strlen(job_attrs[i].name),
      job_attrs + i) == SYS_OK);
        logStringImpl("%s-%s\n", __func__, job_attrs[i].name); <=== dump log to ring-buffer
    }
}
```

How we could get the log in ring buffer ?

method 1:
CMD==> slob dump

Dumping 0x1402768 to 0x1402d1c

```
    0     336963 URF: Init
    1    3088229 ENG: engine mech entry
    2     452908 ENG: ==> engine_stop_high_voltage
    3         56 ENG: <== engine_stop_high_voltage
    4     397855 job_attr_register-confirmation-sheet-print
    5         94 job_attr_register-destination-uris
    6         67 job_attr_register-number-of-retries
    7         58 job_attr_register-retry-interval
    8         56 job_attr_register-retry-time-out
    9         59 job_attr_register-date-time-at-completed
   10         59 job_attr_register-date-time-at-creation
   11         61 job_attr_register-date-time-at-processing
   12         60 job_attr_register-destination-statuses
   13         52 job_attr_register-job-id
   14         51 job_attr_register-job-impressions
   15        127 job_attr_register-job-impressions-completed
   16         67 job_attr_register-job-name
   17         60 job_attr_register-job-originating-user-name
   18         60 job_attr_register-job-printer-up-time
   19         57 job_attr_register-job-printer-uri
   20         53 job_attr_register-job-state
   21         54 job_attr_register-job-state-reasons
   22         53 job_attr_register-job-uri
   23         48 job_attr_register-job-uuid
   24         55 job_attr_register-time-at-completed
   25         55 job_attr_register-time-at-creation
   26         56 job_attr_register-time-at-processing
   27         59 job_attr_register-input-attributes-actual
```

How we could get the log in ring buffer ?

run the following command on your PC

$ nc -l 50310

create a server that listens to 50310 port by netcat

50310 port is hardcoded in source.

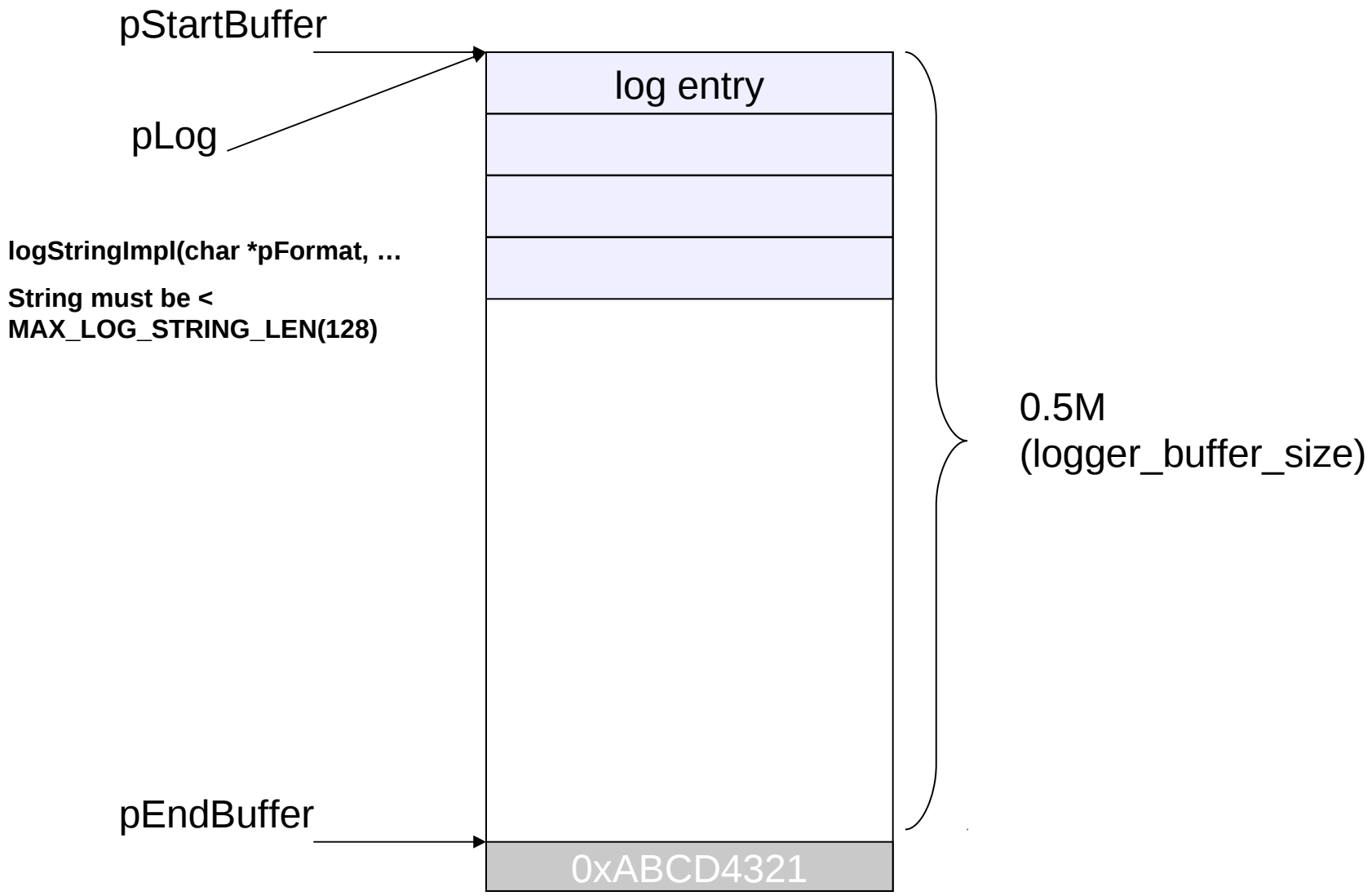
How we could get the log in ring buffer ?

CMD==> slog ip 10.38.52.107
Dumping 0x1402768 to 0x1402d1c
ipdump socket opened...sending...done

You will get the log from ring buffer on you PC

```
walterzh@walterzh-Precision-T1650:~/work/FaxOut_for_IPP_over_USB/src/marvell_6110_sdk_0113_0209$ nc -l 50310
    0     336963 URF: Init
    1    3088229 ENG: engine mech entry
    2     452908 ENG: ==> engine_stop_high_voltage
    3         56 ENG: <== engine_stop_high_voltage
    4     397855 job_attr_register-confirmation-sheet-print
    5         94 job_attr_register-destination-uris
    6         67 job_attr_register-number-of-retries
    7         58 job_attr_register-retry-interval
    8         56 job_attr_register-retry-time-out
    9         59 job_attr_register-date-time-at-completed
   10         59 job_attr_register-date-time-at-creation
   11         61 job_attr_register-date-time-at-processing
   12         60 job_attr_register-destination-statuses
   13         52 job_attr_register-job-id
   14         51 job_attr_register-job-impressions
   15        127 job_attr_register-job-impressions-completed
   16         67 job_attr_register-job-name
   17         60 job_attr_register-job-originating-user-name
   18         60 job_attr_register-job-printer-up-time
   19         57 job_attr_register-job-printer-uri
   20         53 job_attr_register-job-state
   21         54 job_attr_register-job-state-reasons
   22         53 job_attr_register-job-uri
```

Log memory internal



How to output log in your program (The recommended approach)

for example:

define DBG_PRFX and LOGGER_MODULE_MASK macros in your module

in ipp_request.c

```
#define DBG_PRFX "IPP: "
#define LOGGER_MODULE_MASK DEBUG_LOGGER_MODULE_NETWORK
    | NET_LOGGER_SUBMOD_IPP
```

Define output log macros

```
#define DBG_ERR(...) DBG_PRINTF(LOG_ERR, DBG_PRFX __VA_ARGS__)
#define DBG_MSG(...) DBG_PRINTF(LOG_NOTICE, DBG_PRFX __VA_ARGS__)
#define DBG_VERBOSE(...) DBG_PRINTF(LOG_DEBUG, DBG_PRFX __VA_ARGS__)
```

Preferable Macros

```
#define DBG_PRINTF_EMERG(...) DBG_PRINTF(LOG_EMERG, DBG_PRFX __VA_ARGS__)

#define DBG_PRINTF_ALERT(...) DBG_PRINTF(LOG_ALERT, DBG_PRFX __VA_ARGS__)

#define DBG_PRINTF_CRIT(...) DBG_PRINTF(LOG_CRIT, DBG_PRFX __VA_ARGS__)

#define DBG_PRINTF_ERR(...) DBG_PRINTF(LOG_ERR, DBG_PRFX __VA_ARGS__)

#define DBG_PRINTF_WARNING(...) DBG_PRINTF(LOG_WARNING, DBG_PRFX __VA_ARGS__)

#define DBG_PRINTF_NOTICE(...) DBG_PRINTF(LOG_NOTICE, DBG_PRFX __VA_ARGS__)

#define DBG_PRINTF_INFO(...) DBG_PRINTF(LOG_INFO, DBG_PRFX __VA_ARGS__)

#define DBG_PRINTF_DEBUG(...) DBG_PRINTF(LOG_DEBUG, DBG_PRFX __VA_ARGS__)

#define DBG_PRINTF_DEBUG_M(...) DBG_PRINTF(LOG_DEBUG_M, DBG_PRFX __VA_ARGS__)

#define DBG_PRINTF_DEBUG_H(...) DBG_PRINTF(LOG_DEBUG_H, DBG_PRFX __VA_ARGS__)
```

Sample

in ipp_request.c

```
#define DBG_PRFX "IPP: "
#define LOGGER_MODULE_MASK DEBUG_LOGGER_MODULE_NETWORK | NET_LOGGER_SUBMOD_IPP

#define DBG_ERR(...) DBG_PRINTF(LOG_ERR, DBG_PRFX __VA_ARGS__)
#define DBG_MSG(...) DBG_PRINTF(LOG_NOTICE, DBG_PRFX __VA_ARGS__)
#define DBG_VERBOSE(...) DBG_PRINTF(LOG_DEBUG, DBG_PRFX __VA_ARGS__)

DBG_MSG("req %d: created job %d\n", ipp_req->http_hndl, job_id);
DBG_VERBOSE("enter %s\n", __func__);

smjob_rcode = smjob_get_status(job_id, &job);
if (smjob_rcode != SMJOB_OK)
{
  DBG_ERR( "_faxout_job_attributes - ERROR smjob_get_status call failed!\n");
  return NULL;
}
```

Sample

for example

```
DBG_MSG("req %d: created job %d\n", ipp_req->http_hndl, job_id);

DBG_VERBOSE("enter %s\n", __func__);

smjob_rcode = smjob_get_status(job_id, &job);
if (smjob_rcode != SMJOB_OK)
{
  DBG_ERR( "_faxout_job_attributes - ERROR smjob_get_status call
     failed!\n");
  return NULL;
}
```

Log Level

In logger.h

```
/// LOG_LEVEL 0-7
/// log type globally log anything less than a set threshold
/// if enabled_modules_flags > this statements LOG_XXX value it will print.
#define LOG_EMERG       0  ///< aliased to LOG_CRIT
#define LOG_ALERT       0  ///< aliased to LOG_CRIT
#define LOG_CRIT        0  ///< LOG_CRIT the most severe error
#define LOG_ERR         1  ///< second most severe error level
#define LOG_WARNING     2  ///< Recoverable.
#define LOG_NOTICE      3  ///< notice me I'm important but its all good.
#define LOG_INFO        4  ///< nothing is wrong I'm just logging.
#define LOG_DEBUG       5  ///< debug only lots of printfs
#define LOG_DEBUG_M     6  ///< medium debug info
#define LOG_DEBUG_H     7  ///< log every thing level
```

Logger commands

user could enable / disable debug output by logger command
Usage: logger set [<module>] [<submodule>] [<level>]

for example:

CMD==> **logger set NETWORK ipp LOG_ERR**

enable logging in ipp submodule in NETWORK module at LOG_ERR level

CMD==> **logger set NETWORK LOG_ERR**

enable logging in all submodules in NETWORK at LOG_ERR level

CMD==> **logger set LOG_ERR**

enable logging in all submodules in all modules at LOG_ERR level

Under the hood show for logging framework

in release version

```
static __inline__ int DBG_PRINTF(int flags, const char* fmt, ...){return 0;}
```

in debug version

```
#define DBG_PRINTF(level, fmt,...) do {   \
    if (  DBG_WOULD_PRINTF(level) )        \
    {  my_dbg_printf(fmt,## __VA_ARGS__); } } while (0)

# define my_dbg_printf dbg_printf

#define DBG_WOULD_PRINTF(level) \
    ( modules_dbg_flags[((LOGGER_MODULE_MASK) & 0x0000001f)][((level)
     & 0x00000007)] & ((LOGGER_MODULE_MASK) | (level)) )


/* \brief global variable used to implement DBG_PRINTF macros */
uint32_t modules_dbg_flags[32][8] = { {0} };
```

Under the hood show for logging framework

All module names in current SDK

```
/// LOG_MODULE 0-31
/// must be a sequential with unused at the end.
#define DEBUG_LOGGER_MODULE_PRINT          0 ///< PRINT MODULE has submodules.
#define DEBUG_LOGGER_MODULE_CNTRL_PANEL    1 ///< control panel
#define DEBUG_LOGGER_MODULE_USB_DEVICE     2
#define DEBUG_LOGGER_MODULE_NVRAM          3 ///< non-volatile spi flash etc.
#define DEBUG_LOGGER_MODULE_SYSTEM         4 ///< SYSTEM is a big bag.
#define DEBUG_LOGGER_MODULE_NETWORK        5 ///< Lots of submodules in NETWORK
#define DEBUG_LOGGER_MODULE_GPIO           6 ///< DEVICE with submodule might be better idea.
#define DEBUG_LOGGER_MODULE_SCAN           7
#define DEBUG_LOGGER_MODULE_ENGINE         8 ///< is this print engine
#define DEBUG_LOGGER_MODULE_DPRINTF        9 ///< DPRINTF's without LOGGER_MODULE_MASK defined go
      here.
#define DEBUG_LOGGER_MODULE_DEVICES        10 ///< DEVICE with submodule
#define DEBUG_LOGGER_MODULE_VIDEO          11 ///< VIDEO
#define DEBUG_LOGGER_MODULE_JBIG           12 ///< JBIG
#define DEBUG_LOGGER_MODULE_HTTP           13 ///< HTTP
#define DEBUG_LOGGER_MODULE_FILESYSTEM     14 ///< Filesystem
#define DEBUG_LOGGER_MODULE_GENERATORS     15 ///< Generators
#define DEBUG_LOGGER_MODULE_CONSUMABLES    16 ///< Consumables
#define DEBUG_LOGGER_MODULE_VPI            17 ///< Virtual Printer
#define DEBUG_LOGGER_MODULE_FAX            18 ///< Fax
#define DEBUG_LOGGER_MODULE_JPEG           19 ///< JPEG codec
#define DEBUG_LOGGER_MODULE_DEC_ENGINE     20 ///< DEC engine control
#define DEBUG_LOGGER_LAST_MODULE           20 ///< keep this updated to the last in use module number,
                                              ///< keep logger.c::debug_logger_idx_names[] updated.
```

Under the hood show for logging framework

If you are the owner of some modules, you need define submodule.

for example:

```
#define NET_LOGGER_SUBMOD_IPP          LOGGER_SUBMODULE_BIT(7)

#define LOGGER_SUBMODULE_BIT(submodule_id)  ( 1 << (submodule_id + 5 ))
```

2 ^ 5 = 32 (up to 32 modules)

32 - 5 = 27 submodules (up to 27 submodules in one module)

NET_LOGGER_SUBMOD_IPP = (1 << (7 + 5))

Under the hood show for logging framework

for example:

```
#define LOGGER_MODULE_MASK
    DEBUG_LOGGER_MODULE_NETWORK |
    NET_LOGGER_SUBMOD_IPP
```

==> LOGGER_MODULE_MASK include 2 parts, the lowest 5 bits (bit 0 to bit 4) denote module and others denote submodule.

```
LOGGER_MODULE_MASK =
    0000,0000,0000,0000,0000,0000,0000,0101 |
    0000,0000,0000,0000,0001,0000,0000,0000
                                =
    0000,0000,0000,0000,0001,0000,0000,0101
```

Under the hood show for logging framework

```
#define DBG_WOULD_PRINTF(level) \

    ( modules_dbg_flags[((LOGGER_MODULE_MA
    SK) & 0x0000001f)][((level) & 0x00000007)] &
    ((LOGGER_MODULE_MASK) | (level)) )
```

(LOGGER_MODULE_MASK) & 0x0000001f ==> get which module

(level) & 0x00000007 ==> which level

	CRIT	ERR	WARNING	NOTICE	INFO	DEBUG	DEBUG_M	DEBUG_H
PRINT	0	0	0	0	0	0	0	0
CP	0	0	0	0	0	0	0	0
USB	0	0	0	0	0	0	0	0
NVRAM	0	0	0	0	0	0	0	0
SYSTEM	0	0	0	0	0	0	0	0
NETWORK	0	0	0	0	0	0	0	0
GPIO	0	0	0	0	0	0	0	0
SCAN	0	0	0	0	0	0	0	0
ENGINE	0	0	0	0	0	0	0	0
DPRINTF	0	0	0	0	0	0	0	0
DEVICES	0	0	0	0	0	0	0	0
VIDEO	0	0	0	0	0	0	0	0
JBIG	0	0	0	0	0	0	0	0
HTTP	0	0	0	0	0	0	0	0
FILESYSTEM	0	0	0	0	0	0	0	0
GENERATORS	0	0	0	0	0	0	0	0
CONSUMABLES	0	0	0	0	0	0	0	0
VPI	0	0	0	0	0	0	0	0
FAX	0	0	0	0	0	0	0	0
JPEG	0	0	0	0	0	0	0	0
DEC	0	0	0	0	0	0	0	0

Under the hood show for logging framework

logging framework modify the **modules_dbg_flags[ ][ ]** by the following APIs

int logger_set_1(const char * level)

int logger_set_2(const char * module, const char * level)

int logger_set_3(const char * module, const char * submodule, const char * level)

Under the hood show for logging framework

module name

```
static const char* debug_logger_module_idx_names[32] =
{
    "PRINT",
    "CNTRL_PANEL",
    "USB_DEVICE",
    "NVRAM",
    "SYSTEM",
    "NETWORK",
    "GPIO",
    "SCAN",
    "ENGINE",
    "DPRINTF",
    "DEVICES",
    "VIDEO",
    "JBIG",
    "HTTP",
    "FILESYSTEM",
    "GENERATORS",
    "CONSUMABLES",
    "FAX",
    "VPI",
    "JPEG",
    "DEC_ENGINE",
    0
};
```

Under the hood show for logging framework

submodule name is registered dynamically

please search
int logger_submodule_register(int module_index, int submodule_id, const char * module_name);

for example:

```
void net_logger_init(void)
{
    logger_submodule_register( DEBUG_LOGGER_MODULE_NETWORK, 0,  "iface" );
    logger_submodule_register( DEBUG_LOGGER_MODULE_NETWORK, 1,  "link" );
    logger_submodule_register( DEBUG_LOGGER_MODULE_NETWORK, 2,  "net_io" );
    logger_submodule_register( DEBUG_LOGGER_MODULE_NETWORK, 3,  "raw_io" );
    logger_submodule_register( DEBUG_LOGGER_MODULE_NETWORK, 4,  "print" );
    logger_submodule_register( DEBUG_LOGGER_MODULE_NETWORK, 5,  "scan" );
//    logger_submodule_register( DEBUG_LOGGER_MODULE_NETWORK, 6,  "sm_job" );
//    logger_submodule_register( DEBUG_LOGGER_MODULE_NETWORK, 7,  "ipp" );
    logger_submodule_register( DEBUG_LOGGER_MODULE_NETWORK, 8,  "gcpp" );
    logger_submodule_register( DEBUG_LOGGER_MODULE_NETWORK, 9,  "wsd" );
    logger_submodule_register( DEBUG_LOGGER_MODULE_NETWORK, 10, "telnet" );
    logger_submodule_register( DEBUG_LOGGER_MODULE_NETWORK, 11, "mdns" );
    logger_submodule_register( DEBUG_LOGGER_MODULE_NETWORK, 12, "httpc" );
}
```

Under the hood show for logging framework

for example:

==> logger set NETWORK ipp LOG_ERR

	CRIT	ERR	WARNING	NOTICE	INFO	DEBUG	DEBUG_M	DEBUG_H
NETWORK	XX1XX	XX1XX	XX0XX	XX0XX	XX0XX	XX0XX	XX0XX	XX0XX

1 --- ipp submodule flag

modules_dbg_flags[NETWORK][ERR] denotes all submodule flags in NETWORK module and ERR level

Under the hood show for logging framework

==> logger set NETWORK ipp off

disable all level debug output in ipp submodule[NETWORK module]

	CRIT	ERR	WARNING	NOTICE	INFO	DEBUG	DEBUG_M	DEBUG_H
NETWORK	XX0XX	XX0XX	XX0XX	XX0XX	XX0XX	XX0XX	XX0XX	XX0XX

0 --- ipp submodule flag

clear ipp submodule flag in all level

Get Log by telnet

If we could not access uart port, we could get log from telnet (if network is OK)

And, output log by network is quicker than outputing by UART.

If enable telnet log, you will be no longer to interact with UART terminal.

Get Log by telnet

1. USBCmd.linux "net info" to get the ip address of the printer

$ USBCmd.linux "net info"

Registered Links:

ethernet: 46:46:33:46:41:42
WLAN : 00:50:43:01:29:e2
uap : 00:50:43:01:29:e2

Registered Interfaces:

(1) LOOPBACK:
IP Address : 127.0.0.1
Subnet Mask: 255.255.255.255

(2) NetDrvr:
MAC Address: 46:46:33:46:41:42
IP Address : 10.38.52.204
Subnet Mask: 255.255.254.0
Gateway : 10.38.52.1
DNS Srvr(s): 10.38.116.23, 10.38.116.24

(3) uAP_drvr:
This interface is not open/up

Get Log by telnet

2. telnet to the print

walterzh@walterzh-Precision-T1650:~/work/FaxOut_for_IPP_over_USB/src/marvell-sdk$ telnet 10.38.52.204

Trying 10.38.52.204...

Connected to 10.38.52.204.

Escape character is '^]'.

!"'#

CMD==> help

help

You should request help for one of the commands:

adc asicpwr base64 cat ccons cdma
cm cmd copy cp cpipe dbg
dcmotor dectop df dhcpd dns echo
edger eng error eth ethdrv ethphy
file_to_pipe flowtext fuser gloss_test gpio help
hips http https httpscan hwconfig i2c

……

Log out buffer internal

DEBUG_OUTBUF_SIZE ?= 8192

MACROS += DEBUG_OUTBUF_SIZE=$(DEBUG_OUTBUF_SIZE)

Log out buffer is a ring buffer. (If the speed of output log is too fast, the log will be overrided.)

Debug UART and Telnet server is only the “watcher” of debug log.

File Log API

```
void rlf_reset(void);

int rlf_printf( const char *fmt, ... );
int rlf_write_buf( const void *buf, unsigned int length );
int rlf_write( const char *s );

void rlf_flush(void);
```

dump log to file

```
#include “file_logger_api.h”

while(1)
{
   bytes_rx = callback(buffer, FILE_BUF_SIZE, &timeout, ipp_req);
   if(bytes_rx > 0)
   {
      file_size += bytes_rx;

      rlf_write_buf(buffer, bytes_rx);     <=== dump ipp data to rlf log file

      if(bytes_rx < FILE_BUF_SIZE)
      {
         rlf_flush();                      <=== flush rlf log file

         break;
      }
   }
   else
   {
      dbg_printf("fif_loop: read error\n");
      break;
   }
}
```

get the log file from the printer

CMD==> ls /Region1 1/rlf

Directory contents: /Region1/rlf

drwxrwxrwx 0 .

drwxrwxrwx 0 ..

-rwxrwxrwx 4 rlf.cfg

-rwxrwxrwx 223232 **rlf-1.log**

$ getfile_from_device.linux -vid 200b -pid 0200
/Region1/rlf/rlf-1.log rlf-1.log

When the system crash and the stack has been destroyed completely, how could you know the call down chain ?

What need you do if you want to understand the unfamiliar module in run-time ? Maybe it is a good start to know the whole call down chain in the module. But how to get the function trace ?

-finstrument-functions option

The great gift from gcc --- -finstrument-functions option

Generate instrumentation calls for entry and exit to functions. Just after function entry and just before function exit, the following profiling functions will be called with the address of the current function and its call site. (On some platforms, "__builtin_return_address" does not work beyond the current function, so the call site information may not be available to the profiling functions otherwise.)

```
void __cyg_profile_func_enter (void *this_fn,
                               void *call_site);
void __cyg_profile_func_exit  (void *this_fn,
                               void *call_site);
```

Sample

$ cat hello.c

```
#include <stdio.h>

int main()
{
	printf("Hello world\n");
	return 0;
}

#define DUMP(func, call) printf("%s: func = %p, called by = %p\n", __func__, this_fn, call_site);

void __attribute__ ((__no_instrument_function__))
__cyg_profile_func_enter (void *this_fn, void *call_site)
{
	DUMP(this_fn, call_site);
}

void __attribute__ ((__no_instrument_function__))
__cyg_profile_func_exit  (void *this_fn, void *call_site)
{
	DUMP(this_fn, call_site);
}
```

==> gcc -finstrument-functions hello.c -o hello

Sample

```
$ objdump -d hello

0000000000400616 <main>:
  400616: 55                    push   %rbp
  400617: 48 89 e5              mov    %rsp,%rbp
  40061a: 53                    push   %rbx
  40061b: 48 83 ec 08           sub    $0x8,%rsp
  40061f:  48 8b 45 08          mov    0x8(%rbp),%rax
  400623: 48 89 c6              mov    %rax,%rsi
  400626: bf 16 06 40 00        mov    $0x400616,%edi
  40062b: e8 84 ff ff ff        callq  4005b4 <__cyg_profile_func_enter>
  400630: bf 7f 07 40 00        mov    $0x40077f,%edi
  400635: e8 66 fe ff ff        callq  4004a0 <puts@plt>
  40063a: bb 00 00 00 00        mov    $0x0,%ebx
  40063f:  48 8b 45 08          mov    0x8(%rbp),%rax
  400643: 48 89 c6              mov    %rax,%rsi
  400646: bf 16 06 40 00        mov    $0x400616,%edi
  40064b: e8 95 ff ff ff        callq  4005e5 <__cyg_profile_func_exit>
  400650: 89 d8                 mov    %ebx,%eax
  400652: 48 83 c4 08           add    $0x8,%rsp
  400656: 5b                    pop    %rbx
  400657: 5d                    pop    %rbp
  400658: c3                    retq
```

Sample

-finstrument-functions feature application

If I want to get the all call-down-chain in sm_job module

common/network/apps/sm_job/makefile

```
SOURCE = sm_job_mgr.c
SOURCE += sm_job_pif.c
SOURCE += sm_job_support.c
SOURCE += sm_job_core.c
SOURCE += sm_job_cmd.c

ifdef HAVE_FAXOUT
SOURCE += sm_job_fif.c
SOURCE += ftrace.c
endif

ifdef HAVE_SCAN_SUPPORT
SOURCE += sm_job_sif.c
endif

# weak linkage config
SOURCE += sm_job_dflt_config.c

include $(BUILD_ROOT)/stdtargets.mk
CCPARAM += -Wall -Werror -finstrument-functions
```

Sample

for example:

sm_job_fif.c

```
bool smjob_fif_check_fax_busy()
{
    uint32_t status;
    status = fax_get_ongoing_status();
    if(status == SESSION_LINE_BUSY)
        g_fax_busy = true;
    else
        g_fax_busy = false;

    return( g_fax_busy );
}
```

Sample

/opt/arm-marvell-eabi-4.2.0/bin/arm-marvell-eabi-objdump -d sm_job_fif.o

```
00000264 <smjob_fif_check_fax_busy>:
 264:   e1a0c00d        mov     ip, sp
 268:   e92dd830        push    {r4, r5, fp, ip, lr, pc}
 26c:   e24cb004        sub     fp, ip, #4      ; 0x4
 270:   e1a0500e        mov     r5, lr
 274:   e59f002c        ldr     r0, [pc, #44]   ; 2a8 <smjob_fif_check_fax_busy+0x44>
 278:   e1a01005        mov     r1, r5
 27c:   ebfffffe        bl      0 <__cyg_profile_func_enter>
 280:   e59f3024        ldr     r3, [pc, #36]   ; 2ac <smjob_fif_check_fax_busy+0x48>
 284:   e5d33000        ldrb    r3, [r3]
 288:   e1a04003        mov     r4, r3
 28c:   e59f0014        ldr     r0, [pc, #20]   ; 2a8 <smjob_fif_check_fax_busy+0x44>
 290:   e1a01005        mov     r1, r5
 294:   ebfffffe        bl      0 <__cyg_profile_func_exit>
 298:   e1a03004        mov     r3, r4
 29c:   e1a00003        mov     r0, r3
 2a0:   e89d6830        ldm     sp, {r4, r5, fp, sp, lr}
 2a4:   e12fff1e        bx      lr
```

get all function-trace in sm_job module

```
CMD==> slog dump
Dumping 0x12fd550 to 0x12fd936
    0    238154 URF: Init
    1    179519 enter 0027a8d4
    2        53 enter 00279ba0
    3      1901 leave 00279ba0
    4        44 enter 0028addc
    5       112 enter 0028b164
    6        39 leave 0028addc
    7        25 enter 0028e0a8
    8      1496 enter 0028ec54
    9       125 enter 0028f7c0
   10      5304 leave 0028e0a8
   11        39 enter 002865c0
   12        65 enter 002869f0
   13      4057 leave 002865c0
   14        42 enter 00286e9c
   15        25 enter 00289d44
   16        27 leave 00289d44
   17        24 enter 00289c44
   18        24 leave 00289c44
   19        23 enter 00289be8
   20        23 leave 00289be8
   21        73 enter 0028a738
   22        36 enter 0028daac
   23        28 leave 0028daac
   24        33 leave 00286e9c
   25        26 leave 0027a8d4
   26   1247033 ENG: engine mech entry
   27    452494 ENG: ==> engine_stop_high_voltage
   28        59 ENG: <== engine_stop_high_voltage
```

get all function-trace in sm_job module

$ python ftrace.py marvell-sdk/marvell_6110_mfp_sdk-debug.elf ftrace.log

enter smjob_cmd_init () in sm_job_cmd.c:1800
enter smjob_cmd_scan_test () in sm_job_cmd.c:1502
leave smjob_cmd_scan_test () in sm_job_cmd.c:1502
enter smjob_mgr_loop () in sm_job_mgr.c:1711
enter smjob_pif_init () in sm_job_pif.c:264
leave smjob_mgr_loop () in sm_job_mgr.c:1711
enter _get_job_state_reason_str () in sm_job_pif.c:1446
enter smjob_sif_cancel_scan () in sm_job_sif.c:485
enter sif_data_process_loop () in sm_job_sif.c:709
leave _get_job_state_reason_str () in sm_job_pif.c:1446
enter smjob_oem_get_output_tray_pagedelivery () in sm_job_dflt_config.c:2681
enter smjob_fif_get_cur_faxout_job_id () in sm_job_fif.c:154
leave smjob_oem_get_output_tray_pagedelivery () in sm_job_dflt_config.c:2681
enter fif_loop () in sm_job_fif.c:287
enter init_job_lists () in sm_job_mgr.c:1351
leave init_job_lists () in sm_job_mgr.c:1351
enter smjob_mgr_turn_on_printing () in sm_job_mgr.c:1321
leave smjob_mgr_turn_on_printing () in sm_job_mgr.c:1321
enter smjob_mgr_turn_off_printing () in sm_job_mgr.c:1314
leave smjob_mgr_turn_off_printing () in sm_job_mgr.c:1314
enter remove_job_from_list () in sm_job_mgr.c:1555
enter smjob_pif_get_comp_print_job () in sm_job_pif.c:1301
leave smjob_pif_get_comp_print_job () in sm_job_pif.c:1301
leave fif_loop () in sm_job_fif.c:287
leave smjob_cmd_init () in sm_job_cmd.c:1800