Application Note

Jacinto 7 SoC 上的 UART 日志调试系统

Kangjia Dong and Kevin Peng

摘要

TI 全新的汽车处理器 Jacinto 7 系列适用于 ADAS 和网关不同场景下的汽车应用。它包含 TDA4X 和 DRA82X 系列，分别主要用于 ADAS 和网关。这些处理器基于多核异构架构，其中包含 Cortex®-R5、Cortex-A72、Cortex-M3/M4、DSP 和一些通用外设。这些处理器具有很高的可重用性。通常，该系列处理器需要在每个内核上运行一个操作系统和相关的应用程序线程，涉及内部数据传输和外设调用。因此，Jacinto™ 7 系列处理器的应用具有相当高的复杂性。有时，当出现问题时，调试并不容易。

本应用手册演示了 TI 提供的参考设计中通用异步接收器/发送器 (UART) 记录系统硬件和软件级别的基本信息。其中包括基于默认 SDK 和参考设计自定义客户自有日志输出串行端口的方法，以及通过打印 UART 日志来解决问题的多核异构片上系统 (SoC) 调试方法。

内容

1 UART 简介.....2
1.1 Jacinto 7 UART 概述.....2
1.2 Jacinto 7 UART 特性.....2
1.3 Jacinto 7 UART 功能简介.....3
2 UART 使用概述.....4
2.1 WKUP_UART0 用法.....4
2.2 MCU_UART0 用法.....6
2.3 MAIN_UARTx 用法.....6
3 软件模块上的日志级设计.....9
4 更改 UART 实例.....11
4.1 更改 MAIN 域的 MAIN_UARTx.....11
4.2 为 DSP/MCU 设置独立 UART 端口.....15
5 总结.....18
6 参考资料.....18

商标

1 UART 简介

UART 是处理器中的一种常见外设，通常用于系统日志信息输出、低成本的人机交互、器件之间的通信等。在车辆通信中尤为常见的是，Lin 总线通常使用 UART 作为低成本的串行通信协议。UART 不需要时钟同步和主/从设置，只需要配置开始位和停止位，因为它是异步通信，可以随时发送和接收数据。当不需要流控制和电平转换时，TX 和 RX 对于外部硬件连接来说是足够的。但在常见的调试过程中，日志信息通常需要输出到计算机中。因此，在硬件设计中，需要通过 USB 串行端口芯片将 TTL 电平转换为 USB 串行端口协议，并输出到计算机端口。PC 上的软件工具需要正确设置应用于处理器驱动程序的串行端口参数，包括波特率、开始位、数据位、奇偶校验位、停止位等，然后才能接收日志信息进行调试。

1.1 Jacinto 7 UART 概述

Jacinto 7 系列处理器均具有相同的 UART IP，因此该系列不同处理器上 UART 的功能和使用方法基本相同。表 1-1 显示每个处理器总共有 11 个 UART 接口，其中一个在 WKUP 域中，一个在 MCU 域中，其余九个在 MAIN 域中。在所有域正常上电后，每个内核都可以通过软件访问其中的任何 UART。但在系统软件架构中，多个内核不应同时访问 UART。这可能会导致某些系统冲突，从而导致某个内核挂起。

表 1-1. 跨器件域的 UART 分配

实例	域		
	WKUP	MCU	MAIN
WKUP_UART0	√	-	-
MCU_UART0	-	√	-
UART0	-	-	√
UART1	-	-	√
UART2	-	-	√
UART3	-	-	√
UART4	-	-	√
UART5	-	-	√
UART6	-	-	√
UART7	-	-	√
UART8	-	-	√
UART9	-	-	√

1.2 Jacinto 7 UART 特性

Jacinto 7 UART 包含以下特性：

- 16C750 兼容
- RS-485 外部收发器自动流量控制支持
- 用于接收器的 64 字节 FIFO 缓冲器和用于发送器的 64 字节 FIFO 缓冲器
- FIFO 的可编程中断触发级别
- 可编程睡眠模式
- 默认选择为 48MHz 功能时钟，波特率最高可达 3.6Mbps
- 在 1200 位/秒和 115.2kbit/s 之间自动选择波特率（仅当使用 48MHz 功能时钟时）
- 可选多点传输
- 可配置时间保护功能
- 可配置的数据格式：
  - 数据位：5、6、7、8 或 9 位
  - 奇偶校验位：偶数、奇数、无
  - 停止位：1、1.5、2 位
- 流量控制：硬件 (RTS/CTS) 或软件 (XON/XOFF)
- 检测错误的起始位
- 换行符生成和检测功能
- 完全优先化的中断系统控制

- 内部测试和环回功能
- 调制解调器控制功能（CTS、RTS）
- 模块实例具有扩展调制解调器控制信号（DCD、RI、DTR、DSR）

1.3 Jacinto 7 UART 功能简介

图 1-1 显示了 Jacinto 7 UART 功能方框图。当处理器需要发送数据时，只需要通过 CPU/DMA 将输出数据写入 FIFO。然后，数据通过 UART_THR 寄存器自动传输到 TX 引脚，并转换为 TTL 电平。如果 UART_THR 寄存器为空，则数据传输完成。在数据接收过程中，TTL 电平首先通过引脚转换为位数据，并由 UART_RHR 写入 FIFO。当 FIFO 达到阈值（最大 64 字节）时，将触发 CPU/DMA 中断，以将数据写入存储器。当读取足够的数据并且 FIFO 中的数据低于阈值时，中断条件将消失，数据接收完成。

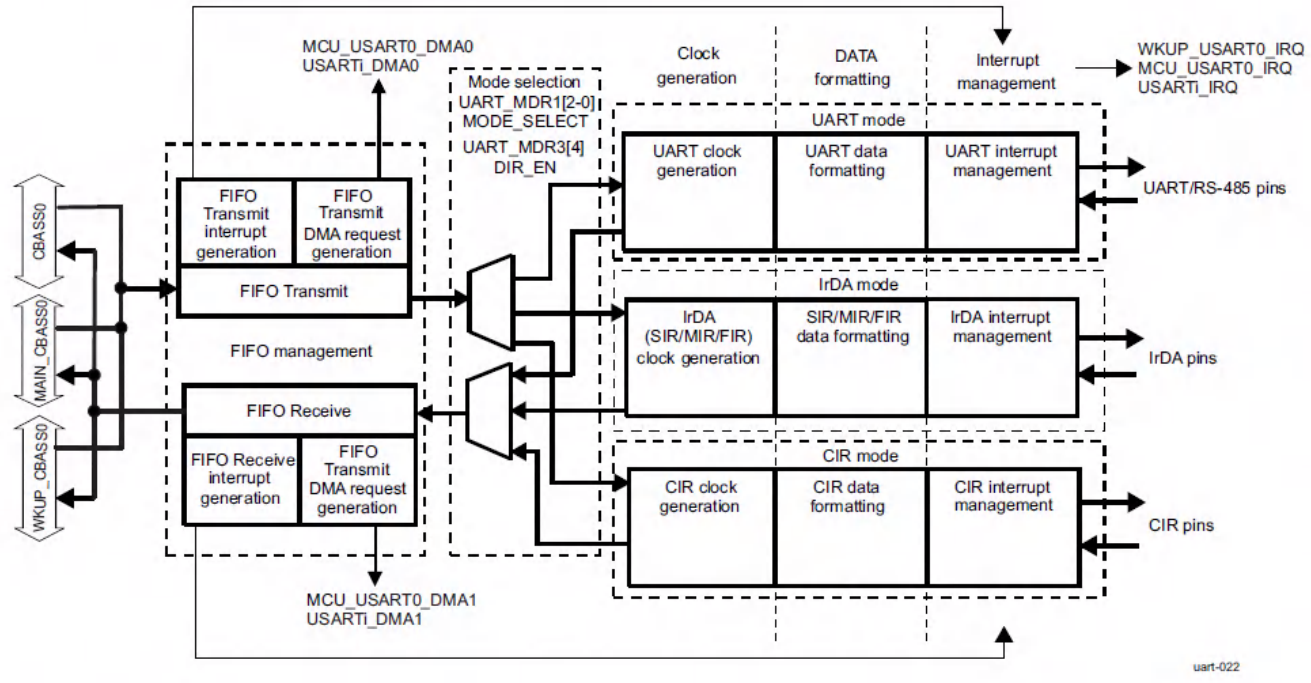


图 1-1. UART 功能方框图

2 UART 使用概述

Jacinto 7 系列处理器的 EVM 板均具有多个串行端口，如表 2-1 所示。通常情况下，在硬件设计期间，至少会保留三个串行端口用于日志信息调试，其余的串行端口可用于与外部器件进行通信。默认情况下，所有串行端口的软件参数都相同，波特率为 115200 位/秒，开始位为 0，有 8 个数据位，奇偶校验位为无，停止位为 1。

WKUP_UART0 保留用于 DSMC 调试，这在系统意外触发防火墙时很常见。对于 SBL 引导，MCU_UART0 用于 MCU1_0 的串行端口输出。对于 UART 引导，MCU_UART0 用于打印“C”字符来判断处理器是否正常工作以及调试 HS 密钥烧录是否成功。DRA821 中的 A72 内核日志打印到 MAIN_UARTx。TDA4X 中所有内核的日志都会打印到 MAIN_UARTx 中，但 MCU1_0 日志会在 SBL 引导时打印到 MCU_UART0 中。

因此，在设计电路板时，至少要预留 WKUP_UART0、MCU_UART0 和 1 个 MAIN_UARTx。此外，建议它们在 TI 参考设计中采用相同的引脚配置。尤其是对于 MCU_UART0，如果更改了引脚，则无法在工程开发的早期阶段使用“C”字符打印进行调试，因为此功能是通过默认引脚设置在 ROM 代码中实现的。

连接外部器件时，需要连接流量控制。始终建议为 UART 通信连接硬件流控制线。另外，软件应该明确启用硬件流控制。否则，会发生数据丢失和数据损坏。

> **小心**
> *TI 提供的参考设计使用 USB 转 UART 来实现串行打印。在 Windows 上连接时，需要安装附加的 驱动程序。*

表 2-1. 参考设计中的 UART 引脚分配

实例	器件			
	DRA821	TDA4VM	TDA4VL/Eco/AL	TDA4VH
WKUP_UART0	√	√	√	√
MCU_UART0	√	√	√	√
UART0	√	√	-	-
UART1	√	√	-	-
UART2	-	√	√	√
UART3	√	-	-	√
UART4	-	√	-	-
UART5	-	-	√	√
UART6	-	-	-	-
UART7	-	-	-	-
UART8	-	-	√	√
UART9	-	-	-	-

2.1 WKUP_UART0 用法

Jacinto 7 系列处理器使用 WKUP_UART0 打印 DMSC（设备管理和安全控制）日志，该日志可用于检查防火墙或 SYSFW（系统固件）是否有错误。默认情况下，WKUP_UART0 日志输出是不够的，需要执行一些额外的步骤来获取完整的日志。下面详细介绍了这些步骤。

1. 对于 SPL 引导：
   a. 在 ti-processor-sdk-linux-xxxx-evm-0x_0x_00_xx/board-support/k3-image-gen-xxxxxxxx/soc/j7xxxx/evm/board-cfg.c 中启用 ENABLE_TRACE 宏
   b. 在 Linux SDK 主目录下重新编译板配置，$make sysfw-image
   c. 重新编译 tiboot3.bin，$make u-boot
   d. 将 tiboot3.bin 和 sysfw.itb 复制到 SD 引导分区
   e. 启动板，将 WUKUP_UART0 日志（在屏幕上）复制到 input_log.txt 文件
   f. RTOS SDK 中有一个脚本解析器 sysfw_trace_parser.py
   g. ./sysfw_trace_parser.py -l input_log.txt -o output_log.txt

```
#ifdef ENABLE_TRACE
        .trace_dst_enables = BOARDCFG_TRACE_DST_UART0 |
                             BOARDCFG_TRACE_DST_ITM |
                             BOARDCFG_TRACE_DST_MEM,
        .trace_src_enables = BOARDCFG_TRACE_SRC_PM |
                             BOARDCFG_TRACE_SRC_RM |
                             BOARDCFG_TRACE_SRC_SEC |
                             BOARDCFG_TRACE_SRC_BASE |
                             BOARDCFG_TRACE_SRC_USER |
                             BOARDCFG_TRACE_SRC_SUPR,
#endif
    },
};
```

图 2-1. SPL 引导板配置

2. 对于 SBL 引导：
   a. 在 ti-processor-sdk-rtos-j7xxxx-evm-xx_xx/pdk_xxxx/packages/ti/drv/sciclient/soc/Vx/sciclient_defaultBoardcfg.c 中启用代码注释
   b. 在 pdk_xxxx/packages/ti/build, $make sciclient_boardcfg 下重新编译板配置
   c. 更新 pdk_xxxx/packages/ti/build 下的 PDK 库，$make pdk_libs_allcores BOARD=j7xxx_evm SOC=j7xxx
   d. 重新编译 pdk_xxxx/packages/ti/build $make -j BOARD=j7xxx_evm CORE=mcu1_0 BUILD_PROFILE=release sbl_mmcsd_img 下的 sbl_mmcsd_img_mcu1_0_release.tiimage
   e. 将 sbl_mmcsd_img_mcu1_0_release.tiimage 作为 tiboot3.bin $cp pdk_xxxx/packages/ti/boot/sbl/binary/j7xxx_evm/mmcsd/bin/sbl_mmcsd_img_mcu1_0_release.tiimage /media/BOOT/tiboot3.bin 复制到 SD 引导分区
   f. 启动板，将 WUKUP_UART0 日志（在屏幕上）复制到 input_log.txt 文件
   g. RTOS SDK 中有一个脚本解析器 sysfw_trace_parser.py
   h. ./sysfw_trace_parser.py -l input_log.txt -o output_log.txt

```
.debug_cfg = {
    .subhdr = {
        .magic = TISCI_BOARDCFG_DBG_CFG_MAGIC_NUM,
        .size = (uint16_t) sizeof(struct tisci_boardcfg_dbg_cfg),
    },
    .trace_dst_enables = 0,
    .trace_src_enables = 0,
    /* This enables the trace for DMSC logging. Should be used only for
     * debug. Comment below if sysfw trace is not needed */
    /* .trace_dst_enables = (TISCI_BOARDCFG_TRACE_DST_UART0 | */
    /*                       TISCI_BOARDCFG_TRACE_DST_ITM | */
    /*                       TISCI_BOARDCFG_TRACE_DST_MEM), */
    /* .trace_src_enables = (TISCI_BOARDCFG_TRACE_SRC_PM | */
    /*                       TISCI_BOARDCFG_TRACE_SRC_RM | */
    /*                       TISCI_BOARDCFG_TRACE_SRC_SEC | */
    /*                       TISCI_BOARDCFG_TRACE_SRC_BASE | */
    /*                       TISCI_BOARDCFG_TRACE_SRC_USER | */
    /*                       TISCI_BOARDCFG_TRACE_SRC_SUPR), */
},
```

图 2-2. SBL 引导板配置

小心

不同的 SOC 和 SDK 版本具有不同的代码路径，因此这里使用 xxxxx 来表示。BOARD 的值包括 j7200_evm/j721e_evm/j721s2_evm/j784s4_evm，SOC 的值包括 j7200/j721e/j784s4/j721s2

2.2 MCU_UART0 用法

1. 保留 MCU_UART0 后，可使用它来检查处理器是否正常工作。方法是首先将电路板设置为 UART 引导模式，然后 MCU_UART0 串行端口将打印出“CCCCC”字符串，从而检测电路板的电源是否正常。
2. MCU_UART0 也可用于根据此链接烧录 OSPI 闪存。
3. 对于 HS 处理器，MCU_UART0 也可用于检查电子保险丝是否成功烧录密钥，有关详细步骤，请参阅本文档的 *TDA4 HS Prime* 一章。
4. 对于 SBL 引导，MCU_UART0 将用作 SBL 和 MCU1_0 固件用于打印日志的调试串行端口。sbl_main.c 中的 main 函数条目对该串行端口进行配置和初始化。

2.3 MAIN_UARTx 用法

1. 默认 SDK 将使用 MAIN_UARTx 来打印 HLOS 的引导日志。
2. 对于 TDA4X 系列处理器，MAIN_UARTx 用于打印 APP 日志以及 A72、R5F 和 DSP 内核的引导日志。

将多核日志打印到串行端口的软件级设计是一个复杂的过程。了解此多核日志输出系统有助于自定义自有系统输出的设计。默认情况下，U-BOOT 和内核会初始化一个串行端口用于日志输出。有关驱动程序和其他相关信息的详情，请参阅此 U-BOOT / 内核 文档。该串行端口始终在 A72 侧进行控制。A72 内核的日志将直接通过此串行端口而打印。但是，其他内核的日志首先被放入共享存储器中，然后由 A72 应用程序读取。具体过程如下：

1. 除 A72 内核之外，每个内核的 OS 引导都将调用 appInit 函数来初始化图 2-3 中所示的 256KB 共享存储器，用于存储日志。

```
app_log_shared_mem_t g_app_log_shared_mem
__attribute__ ((section(".bss:app_log_mem")))
__attribute__ ((aligned(4096)))
        ;
```

图 2-3. 代码上的日志共享存储器定义

2. 此共享存储器将分为 16 个部分。每个部分总共有 16KB，并且每个部分的前 32 个字节用来存储一个显示了以下代码块的结构。此结构分别用于显示指针在读取和写入中的位置。此共享存储器的其余部分用于日志输出。

```
typedef struct {

 /**< Init by reader to 0 */

    uint32_t log_rd_idx;/**< Init by writer to 0 */
    uint32_t log_wr_idx;

/**<  Init by writer to APP_LOG_AREA_VALID_FLAG.
 reader will ignore this CPU shared mem log
 until the writer sets this
 to APP_LOG_AREA_VALID_FLAG */

    uint32_t log_area_is_valid;

 /**< CPU sync state */
    uint32_t log_cpu_sync_state;

/**< Init by writer to CPU name, used by reader to add a prefix when writing to console device
*/
    uint8_t  log_cpu_name[APP_LOG_MAX_CPU_NAME];

    /**< memory into which logs are written by this CPU */

    uint8_t  log_mem[APP_LOG_PER_CPU_MEM_SIZE];
} app_log_cpu_shared_mem_t;
```

3. 除了 A72 内核之外，无论是使用 printf 还是 UART_print，其他内核的日志都将通过 appLogPrintf 写入此共享存储器。同时，此输出日志的时间戳也将写入此共享存储器，如图 2-4 所示。

```
void appLogPrintf(const char *format,
                  ...)
{
    va_list va_args_ptr;
    uint32_t cookie;
    uint32_t str_len = 0;
    uint64_t cur_time;
    app_log_wr_obj_t *obj = &g_app_log_wr_obj;

    cookie = appLogWrLock(obj);

    cur_time = appLogGetTimeInUsec();
    str_len = (uint32_t)snprintf(obj->buf, APP_LOG_BUF_MAX,
                                 "%6d.%06u s: ",
                                 (uint32_t)(cur_time / 1000000U),
                                 (uint32_t)(cur_time % 1000000U));

    /* if str_len is equal to APP_LOG_BUF_MAX, i.e string overflows buffer,
     * then don't write string. */
    if (str_len < APP_LOG_BUF_MAX)
    {
        va_start(va_args_ptr, format);

        /* MISRA.PTR.ARITH
         * MISRAC_2004_Rule_17.1 and MISRAC_2004_Rule_17.4
         * Pointer is used in arithmatic or array index expression
         * KW State: Ignore -> Waiver -> Case by case
         * MISRAC_WAIVER: buf is pointing to printBuf array of size
         * REMOTE_LOG_SERVER_PRINT_BUF_LEN and it is passed to vsnprint api,
         * which makes sure that the buf is never accessed beyond
         * its REMOTE_LOG_SERVER_PRINT_BUF_LEN size
         */
        vsnprintf((char *)(obj->buf + str_len),
                  APP_LOG_BUF_MAX - str_len,
                  format, va_args_ptr);
        va_end(va_args_ptr);

        appLogWrPutString(obj);

        appLogWrUnLock(obj, cookie);
    }
}
```

图 2-4. MCU/DSP 内核写入日志至共享存储器

4. A72 应用程序 vx_app_arm_remote_log.out 将从 Linux 映射此共享存储器。它每秒读取存储器以提取除 A72 内核之外的每个内核的日志，并添加相应内核的名称并将其输出到图 2-5 中所示的串行端口。

```
void* appLogRdRun(app_log_rd_obj_t *obj)
{
    uint32_t done = 0, cpu_id;
    uint32_t num_bytes, str_len;

    #if defined(FREERTOS) || defined(SYSBIOS) || defined(SAFERTOS)
    appUtilsTaskInit();
    #endif

    while(!done)
    {
        appLogWaitMsecs(obj->log_rd_poll_interval_in_msecs);

        for(cpu_id=0; cpu_id<obj->log_rd_max_cpus; cpu_id++)
        {
            app_log_cpu_shared_mem_t *cpu_shared_mem;

            cpu_shared_mem = &obj->shared_mem->cpu_shared_mem[cpu_id];

            if(cpu_shared_mem->log_area_is_valid == APP_LOG_AREA_VALID_FLAG
                && obj->log_rd_cpu_enable[cpu_id] == 1
             )
            {
                do
                {
                    str_len = 0;
                    num_bytes = appLogRdGetString(cpu_shared_mem,
                                    obj->buf,
                                    APP_LOG_BUF_MAX,
                                    &str_len );
                    if(str_len > 0)
                    {
                        if(obj->device_write)
                        {
                            snprintf(obj->print_buf, APP_LOG_PRINT_BUF_MAX, "[%-6s] %s\r\n",
                                cpu_shared_mem->log_cpu_name,
                                obj->buf);

                            obj->device_write(obj->print_buf, APP_LOG_PRINT_BUF_MAX);
                        }
                    }
                } while(num_bytes);
            }
        }
    }
    return NULL;
}
```

图 2-5. A72 内核从共享存储器中读取日志

3 软件模块上的日志级设计

在大型软件工程中，通常有一个用于调试的日志级别，TI 的 SDK 也有一些用于控制和打印更有用信息以协助调试的常用日志级别。有关更多详细信息，请参阅以下步骤。

1. Linux 内核日志级别：

   在引导阶段，Linux 内核可以在器件树（*如果使用 TDA4VM，则为 k3-j721e-common-proc-board.dts*）中的引导参数上配置不同的日志级别。通过在 Linux 内核配置中添加 loglevel=8 参数，其值范围从 0（最不详细）到 8（最详细）。图 3-1 显示的默认 loglevel 为 7。

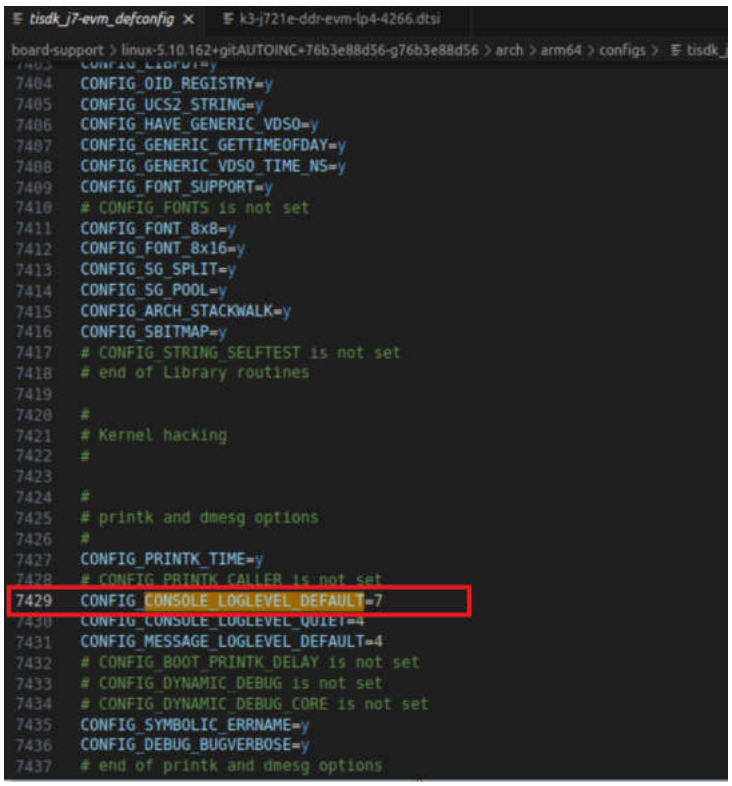


图 3-1. Linux 默认内核日志级别

2. SBL 引导日志级别：

   SBL 日志级别范围从 0（最不详细）到 3（最详细）。下面介绍了将日志级别设置为 3 并消除日志输出限制所需的步骤。

   更改 mcusw/mcuss_demos/boot_app_mcu_rtos/makefile 中的 DSBL_LOG_LEVEL=3

   更改 pdk_xxxx /packages/ti/boot/sbl/sbl_component.mk 中的 DSBL_LOG_LEVEL=3

   pdk_xxxx/packages/ti/boot/sbl/soc/k3/sbl_log.h 中的更改 如下所示：*#define SBL_log(dbg_level, ...) if (1) { UART_printf(__VA_ARGS__); }*

   更改 pdk_xxxx/packages/ti/build/makerules/build_config.mk 中的 DSBL_LOG_LEVEL=3

3. Openvx 日志级别：

作为 TDA4X 应用的重要组成部分，Openvx 还提供用于调试的日志级别。在图 3-2 中，g_debug_zonemask（默认值 0）用于计算当前日志级别。为便于打印 Openvx 日志级别，只需注释掉以下 if 条件。

```
void tivx_print(vx_enum zone, const char *format, ...)
{
    if ((g_debug_zonemask & ZONE_BIT((vx_uint32)zone)) != 0U)
    {
        uint32_t size;
        char string[1024];
        va_list ap;

        va_start(ap, format);

        snprintf(string, sizeof(string), " %s:", find_zone_name(zone));
        size = (uint32_t)strlen(string);
        vsnprintf(&string[size], sizeof(string)-size, format, ap);
        ownPlatformPrintf(string);
        va_end(ap);
    }
}
```

图 3-2. Openvx 低级打印 API

vx_debug.c | main.c .../example_tidl_tb | vx_tidl_target.c | vision_apps_tools_path.mak

tiovx > source > framework > vx_debug.c

```
1   /*
2    * Copyright (c) 2012-2016 The Khronos Group Inc.
3    *
4    * Licensed under the Apache License, Version 2.0 (the "License");
5    * you may not use this file except in compliance with the License.
6    * You may obtain a copy of the License at
7    *
8    *    http://www.apache.org/licenses/LICENSE-2.0
9    *
10   * Unless required by applicable law or agreed to in writing, software
11   * distributed under the License is distributed on an "AS IS" BASIS,
12   * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13   * See the License for the specific language governing permissions and
14   * limitations under the License.
15   */
16
17
18  #include <vx_internal.h>
19
20  static vx_char *find_zone_name(vx_enum zone);
21
22  static vx_uint32 g_debug_zonemask = 0;
23
24  #ifdef ZONE_BIT
25  #undef  ZONE_BIT
26  #endif
27
28  #define ZONE_BIT(zone)  ((vx_uint32)1U << (zone))
29
30  #define _STR2(x) {#x, (vx_enum)x}
```

图 3-3. Openvx 日志级别设置

4. TIDL 日志级别：

   在 TIDL 的导入和推理阶段，debugTraceLevel=3 可用于在 TIDL 推理配置文件中配置从 0（最不详细）到 3（最详细）的不同日志级别。
5. 存储器分配日志级别：

   此目录 ti-processor-sdk-rtos-j7xxxx-evm-xx_xx_xx_xx/app_utils/utils/mem/src/ 下的 C 文件，当定义了 **APP_MEM_DEBUG**（默认值未定义）宏时，将输出存储器分配日志。

4 更改 UART 实例

通常情况下，不建议在客户的硬件设计阶段修改 MCU_UART0 和 WKUP_UART0 串行端口的默认引脚配置。原因是在进行 OSPI 烧录时，ROM 代码使用 MCU_UART0 的默认引脚设置，而 SYSFW 使用 WUKP_UART0 中的默认引脚设置。但是，由于引脚冲突和其他原因，MAIN_UARTX 经常被更改为新的串行端口。此外，在多核调试中，通常需要为 DSP/MAIN_R5F 内核配置单独的 MAIN_UARTX 串行端口进行调试。

4.1 更改 MAIN 域的 MAIN_UARTx

不同 Jacinto 7 系列处理器上的默认端子输出是不同的。例如，UART0 用于 J721E 和 J7200 EVM 上的终端输出，但 UART8 是 J721S2 和 J784S4 EVM 的主要 UART 接口。由于 UART 端口的引脚也可以被其他应用使用，因此通常需要根据客户的用例定制 UART 端口。以下是基于 SDK 8.6 将 TDA4VH 主域上的默认 UART8 更改为 UART2 的示例，这也可以作为更改到其他 Jacinto 7 处理器上其他端口的参考。

1. 添加 UART2 的时钟设置。

   389 是 J784S4 系列的器件 ID，可从此链接找到。

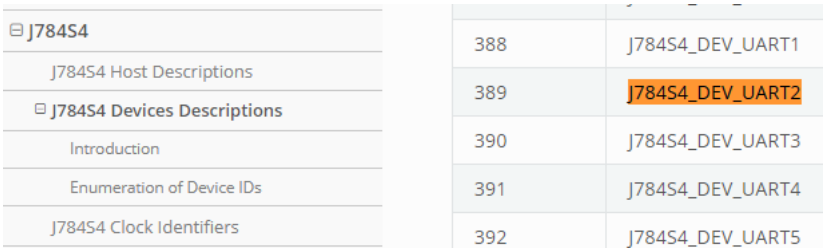
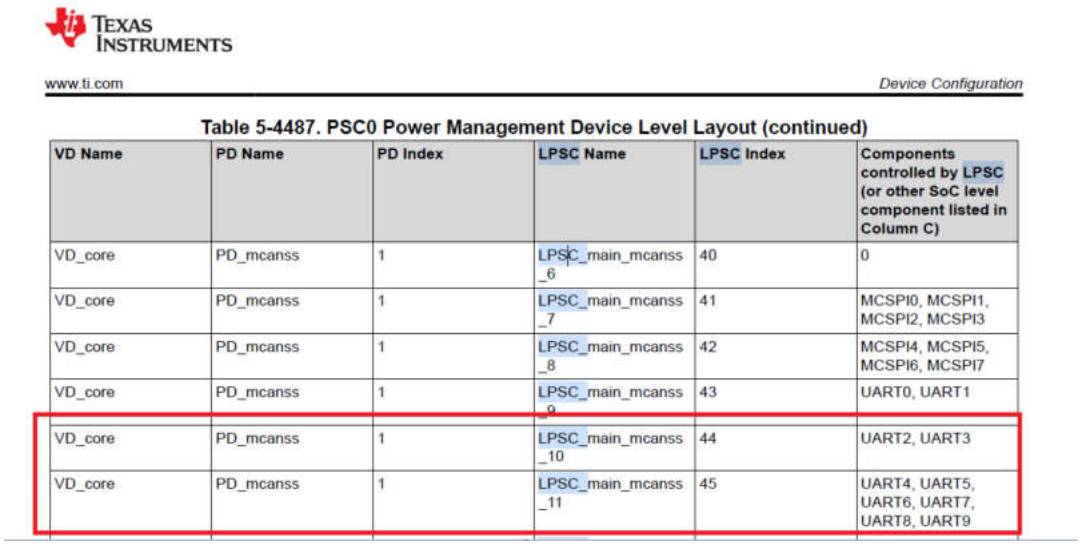


图 4-1. TDA4VH UART 时钟 ID

```
diff --git a/arch/arm/mach-k3/j784s4/clk-data.c b/arch/arm/mach-k3/j784s4/clk-data.c
index a266735cc0..081c6d8970 100644
--- a/arch/arm/mach-k3/j784s4/clk-data.c
+++ b/arch/arm/mach-k3/j784s4/clk-data.c
@@ -283,7 +283,7 @@ static const struct clk_data clk_list[] = {
    CLK_MUX("emmcsd_refclk_sel_out1", emmcsd_refclk_sel_out1_parents, 4, 0x1080b4, 0, 2, 0),
    CLK_MUX("gtc_clk_mux_out0", gtc_clk_mux_out0_parents, 16, 0x108030, 0, 4, 0),
    CLK_DIV_DEFFREQ("usart_programmable_clock_divider_out0",
"hsdiv4_16fft_main_1_hsdivout0_clk", 0x1081c0, 0, 2, 0, 0, 48000000),
-   CLK_DIV("usart_programmable_clock_divider_out5", "hsdiv4_16fft_main_1_hsdivout0_clk",
0x1081d4, 0, 2, 0, 0),
+   CLK_DIV("usart_programmable_clock_divider_out2", "hsdiv4_16fft_main_1_hsdivout0_clk",
0x1081c8, 0, 2, 0, 0),
    CLK_DIV("usart_programmable_clock_divider_out8", "hsdiv4_16fft_main_1_hsdivout0_clk",
0x1081e0, 0, 2, 0, 0),
    CLK_DIV("k3_pll_ctrl_wrap_main_0_chip_div24_clk_clk",
"k3_pll_ctrl_wrap_main_0_sysclkout_clk", 0x41011c, 0, 5, 0, 0),
    CLK_DIV("k3_pll_ctrl_wrap_wkup_0_chip_div24_clk_clk",
"k3_pll_ctrl_wrap_wkup_0_sysclkout_clk", 0x4201011c, 0, 5, 0, 0),
@@ -405,8 +405,8 @@ static const struct dev_clk soc_dev_clk_data[] = {
    DEV_CLK(279, 2, "wkup_i2c_mcupll_bypass_out0"),
    DEV_CLK(279, 3, "hsdiv4_16fft_mcu_1_hsdivout3_clk"),
    DEV_CLK(279, 4, "gluelogic_hfosc0_clkout"),
-   DEV_CLK(392, 0, "usart_programmable_clock_divider_out5"),
-   DEV_CLK(392, 3, "k3_pll_ctrl_wrap_main_0_chip_div1_clk_clk"),
+   DEV_CLK(389, 0, "usart_programmable_clock_divider_out2"),
+   DEV_CLK(389, 3, "k3_pll_ctrl_wrap_main_0_chip_div1_clk_clk"),
    DEV_CLK(395, 0, "usart_programmable_clock_divider_out8"),
    DEV_CLK(395, 3, "k3_pll_ctrl_wrap_main_0_chip_div1_clk_clk"),
    DEV_CLK(398, 0, "k3_pll_ctrl_wrap_main_0_chip_div1_clk_clk"),
```

2. 在 UART2 列表中添加器件 ID 和 LPSC 设置。

   44 是 J784S4 系列 UART2 的 LPSC 索引，可从器件特定 TRM 中搜索。



www.ti.com Device Configuration

Table 5-4487. PSC0 Power Management Device Level Layout (continued)

VD Name	PD Name	PD Index	LPSC Name	LPSC Index	Components controlled by LPSC (or other SoC level component listed in Column C)
VD_core	PD_mcanss	1	LPSC_main_mcanss_6	40	0
VD_core	PD_mcanss	1	LPSC_main_mcanss_7	41	MCSPI0, MCSPI1, MCSPI2, MCSPI3
VD_core	PD_mcanss	1	LPSC_main_mcanss_8	42	MCSPI4, MCSPI5, MCSPI6, MCSPI7
VD_core	PD_mcanss	1	LPSC_main_mcanss_9	43	UART0, UART1
VD_core	PD_mcanss	1	LPSC_main_mcanss_10	44	UART2, UART3
VD_core	PD_mcanss	1	LPSC_main_mcanss_11	45	UART4, UART5, UART6, UART7, UART8, UART9

图 4-2. TDA4VH UART LPSC

```
diff --git a/arch/arm/mach-k3/j784s4/dev-data.c b/arch/arm/mach-k3/j784s4/dev-data.c
index e44afad3ec..b5ae132b4a 100644
--- a/arch/arm/mach-k3/j784s4/dev-data.c
+++ b/arch/arm/mach-k3/j784s4/dev-data.c
@@ -51,6 +51,7 @@ static struct ti_lpsc soc_lpsc_list[] = {
     [20] = PSC_LPSC(81, &soc_psc_list[2], &soc_pd_list[6], &soc_lpsc_list[18]),
     [21] = PSC_LPSC(120, &soc_psc_list[2], &soc_pd_list[7], &soc_lpsc_list[22]),
     [22] = PSC_LPSC(121, &soc_psc_list[2], &soc_pd_list[7], NULL),
+    [23] = PSC_LPSC(44, &soc_psc_list[2], &soc_pd_list[3], NULL),
 };
static struct ti_dev soc_dev_list[] = {
@@ -76,7 +77,7 @@ static struct ti_dev soc_dev_list[] = {
     PSC_DEV(141, &soc_lpsc_list[14]),
     PSC_DEV(140, &soc_lpsc_list[15]),
     PSC_DEV(146, &soc_lpsc_list[16]),
-    PSC_DEV(392, &soc_lpsc_list[17]),
+    PSC_DEV(389, &soc_lpsc_list[23]),
     PSC_DEV(395, &soc_lpsc_list[17]),
     PSC_DEV(198, &soc_lpsc_list[18]),
     PSC_DEV(202, &soc_lpsc_list[19]),
--
```

3. 设置串行端口和引脚，应同时为 UBOOT dts 和内核 dts 文件设置。

```
diff --git a/arch/arm/dts/k3-j784s4-evm-u-boot.dtsi b/arch/arm/dts/k3-j784s4-evm-u-boot.dtsi
index 9d6f7dbbd5..3846d90f9a 100644
--- a/arch/arm/dts/k3-j784s4-evm-u-boot.dtsi
+++ b/arch/arm/dts/k3-j784s4-evm-u-boot.dtsi
@@ -12,7 +12,7 @@
     aliases {
         serial0 = &wkup_uart0;
         serial1 = &mcu_uart0;
-        serial2 = &main_uart8;
+        serial2 = &main_uart2;
         i2c0 = &wkup_i2c0;
         i2c1 = &mcu_i2c0;
         i2c2 = &mcu_i2c1;
@@ -105,6 +105,10 @@
     u-boot,dm-spl;
 };

+&main_uart2_pins_default {
+    u-boot,dm-spl;
+};
+
```

```
 &main_mmc1_pins_default {
     u-boot,dm-spl;
 };
@@ -132,6 +136,10 @@
     u-boot,dm-spl;
 };

+&main_uart2 {
+    u-boot,dm-spl;
+};
+
 &mcu_uart0 {
     u-boot,dm-spl;
 };
diff --git a/arch/arm/dts/k3-j784s4-evm.dts b/arch/arm/dts/k3-j784s4-evm.dts
index 5e213b2c11..7f8f507318 100644
--- a/arch/arm/dts/k3-j784s4-evm.dts
+++ b/arch/arm/dts/k3-j784s4-evm.dts
@@ -21,7 +21,7 @@
         stdout-path = "serial2:115200n8";
     };
     aliases {
-        serial2 = &main_uart8;
+        serial2 = &main_uart2;
         mmc0 = &main_sdhci0;
         mmc1 = &main_sdhci1;
         can0 = &mcu_mcan0;
@@ -402,6 +402,15 @@
         >;
     };

+    main_uart2_pins_default: main-uart2-pins-default {
+        pinctrl-single,pins = <
+            J784S4_IOPAD(0x0c4, PIN_INPUT, 11) /* (AD36) CTSn */
+            J784S4_IOPAD(0x0c8, PIN_OUTPUT, 11) /* (AJ32) RTSn */
+            J784S4_IOPAD(0x0dc, PIN_OUTPUT, 11) /* (AM36) TXD */
+            J784S4_IOPAD(0x0d8, PIN_INPUT, 11) /* (AM35) RXD*/
+        >;
+    };
+
     main_i2c3_pins_default: main-i2c3-pins-default {
         pinctrl-single,pins = <
             J784S4_IOPAD(0x064, PIN_INPUT_PULLUP, 13) /* (AF38) MCAN0_TX.I2C3_SCL */
@@ -743,11 +752,13 @@
     status = "disabled";
 };
-&main_uart1 {
-    status = "disabled";
+&main_uart2 {
+    status = "okay";
+    pinctrl-names = "default";
+    pinctrl-0 = <&main_uart2_pins_default>;
 };

-&main_uart2 {
+&main_uart1 {
     status = "disabled";
 };

diff --git a/arch/arm/dts/k3-j784s4-r5-evm.dts b/arch/arm/dts/k3-j784s4-r5-evm.dts
index 4a697e2738..154a07c802 100644
--- a/arch/arm/dts/k3-j784s4-r5-evm.dts
+++ b/arch/arm/dts/k3-j784s4-r5-evm.dts
@@ -13,7 +13,7 @@
 / {
     chosen {
         firmware-loader = &fs_loader0;
-        stdout-path = &main_uart8;
+        stdout-path = &main_uart2;
         tick-timer = &timer1;
     };

@@ -151,6 +151,15 @@
         >;
     };

+    main_uart2_pins_default: main-uart2-pins-default {
+        pinctrl-single,pins = <
```

```
+            J784S4_IOPAD(0x0c4, PIN_INPUT, 11) /* (AD36) CTSn */
+            J784S4_IOPAD(0x0c8, PIN_OUTPUT, 11) /* (AJ32) RTSn */
+            J784S4_IOPAD(0x0dc, PIN_OUTPUT, 11) /* (AM36) TXD */
+            J784S4_IOPAD(0x0d8, PIN_INPUT, 11) /* (AM35) RXD*/
+        >;
+    };
+
      main_mmc1_pins_default: main-mmc1-pins-default {
         pinctrl-single,pins = <
            J784S4_IOPAD(0x104, PIN_INPUT, 0) /* (AB38) MMC1_CLK */
@@ -253,6 +262,12 @@
     pinctrl-0 = <&main_uart8_pins_default>;
 };

+&main_uart2 {
+    status = "okay";
+    pinctrl-names = "default";
+    pinctrl-0 = <&main_uart2_pins_default>;
+};
```

4. 配置 UART2 的引导命令。

```
diff --git a/include/configs/j784s4_evm.h b/include/configs/j784s4_evm.h
index eb609100b0..942d6c3dbe 100644
--- a/include/configs/j784s4_evm.h
+++ b/include/configs/j784s4_evm.h
@@ -75,7 +75,7 @@
         "setenv fdtfile ${name_fdt}\0"                \
     "name_kern=Image\0"                               \
     "console=ttyS2,115200n8\0"                          \
-    "args_all=setenv optargs earlycon=ns16550a,mmio32,0x02880000 "    \
+    "args_all=setenv optargs earlycon=ns16550a,mmio32,0x02820000 "    \
         "${mtdparts}\0"                               \
     "run_kern=booti ${loadaddr} ${rd_spec} ${fdtaddr}\0"

--
```

5. 如果使用 OPTEE，请重建它。

   只需根据此链接将导出 CFG_CONSOLE_UART=0x8 中的 8 更改为 2 即可。

6. 更改 Arm® 信任固件。

```
diff --git a/plat/ti/k3/include/platform_def.h b/plat/ti/k3/include/platform_def.h
index 690c68e5c..db083ca2f 100644
--- a/plat/ti/k3/include/platform_def.h
+++ b/plat/ti/k3/include/platform_def.h
@@ -91,14 +91,14 @@
 /* Platform default console definitions */
 #ifndef K3_USART_BASE
-#define K3_USART_BASE 0x02800000
+#define K3_USART_BASE 0x02820000
 #endif
```

   修改 UART2 的上述 K3_USART_BASE 后，需要以下指令重新编译 bl31.bin

   a. *cd $SDK_PATH/board-support/trusted-firmware-a-2.8+gitAUTOINC+2fcd408bb3*
   b. *make CROSS_COMPILE=aarch64-none-linux-gnu- ARCH=aarch64 PLAT=k3 TARGET_BOARD=generic SPD=opted*
   c. *cp ./build/k3/generic/release/bl31.bin ../prebuilt-images/*

   然后，根据重建的 bl31.bin，需要如下指令将其制作为 tispl.bin 的一部分，并将 tispl.bin 和 u-boot.img 复制到 SD 卡的 BOOT 中。

   a. *cd ../../*
   b. *make u-boot-spl-jacinto*
   c. *cp board-support/u-boot_build/a72/tispl.bin board-support/u-boot_build/a72/u-boot.img /media/$USER/BOOT*

7. 清除 UBOOT Environment，使其恢复为默认值并保存更改。

图 4-3. 设置 UBOOT Environment

> **小心**
> *从 SDK9.0 开始，不再要求执行第 7 步。应用上述 7 个步骤即可将 TDA4VH 上的默认主 UART 端口从 8 更改为 2。*

4.2 为 DSP/MCU 设置独立 UART 端口

默认情况下，J7 SOC 的 MCU 和 DSP 内核不会为串行端口输出设置单独的 UART。对于 TDA4X，默认情况下，除 A72 内核之外的每个内核的日志都将写入一个共享存储器，然后 A72 应用程序会读取这些日志并将其打印到 MAIN_UARTX 串行端口。但是，为了方便调试，通常有必要同时打印多个内核的日志，或者在内核 A 不能正常工作时继续打印日志。在这种情况下，需要为 MCU/DSP 内核配置一个单独的串行端口。下面以 TDA4VM 为例为 C7 设置单独的串行端口。

1. 在 MAIN 函数中添加 UART 初始配置。

```
diff --git a/vision_apps/platform/j721e/rtos/c7x_1/main.c
b/vision_apps/platform/j721e/rtos/c7x_1/main.c
index 0dcfa4fd..e857838b 100755
--- a/vision_apps/platform/j721e/rtos/c7x_1/main.c
+++ b/vision_apps/platform/j721e/rtos/c7x_1/main.c
@@ -88,7 +88,8 @@
#include <ti/sysbios/family/c7x/Hwi.h>
#include <ti/sysbios/family/c7x/Mmu.h>
#endif
+#include <ti/drv/uart/UART.h>
+#include <ti/drv/uart/UART_stdio.h>
/* For J7ES/J721E/TDA4VM the upper 2GB DDR starts from 0x0008_8000_0000 */
/* This address is mapped to a virtual address of 0x0001_0000_0000 */
#define DDR_C7X_1_LOCAL_HEAP_VADDR (DDR_C7X_1_LOCAL_HEAP_ADDR)
@@ -96,18 +97,33 @@
+extern int uart_print_test(void);
+extern int uart_test(void);
@@ -181,9 +197,13 @@ int main(void)
{
TaskP_Params tskParams;
TaskP_Handle task;
OS_init();
+/* Set TDA4VM PINMUX UART2_RX(PIN Y1)&UART2_TX(PIN Y5)
+*  We can get the register address from the datasheet
+*/
+ *((int *)(0x00011C1DC))=0x50003;
+ *((int *)(0x00011C1E0))=0x10003;
+ uart_print_test();
appC7xClecInitDru();
```

2. 为 C7 创建 UART 实例。

   在路径 *vision_apps/basic_demos/* 下创建 c7_uart_print 临时文件夹，并创建 c7_uart_print.c 和 concerto .mk 文件作为初始配置 UART 库。

   以下内容针对 c7_uart_print.c。

```
#include <ti/drv/uart/UART.h>
#include <ti/drv/uart/UART_stdio.h>
#include <ti/board/src/j721e_evm/include/board_utils.h>
#include <ti/board/board.h>
#include <ti/board/src/j721e_evm/include/board_cfg.h>
int uart_test(void)
{
   UART_printf("\n==============================================\n");
   UART_printf("\n**********c7x uart printf*******************\n");
   UART_printf("*                  UART Test                  *\n");
   UART_printf("==============================================\n");
   return 0;
}

int uart_print_test(void)
{
    Board_initParams_t initParams;

    /* Verify the SoC UART0 */
    Board_getInitParams(&initParams);
    initParams.uartInst = 2;
    initParams.uartSocDomain = BOARD_SOC_DOMAIN_MAIN;
    Board_setInitParams(&initParams);
    Board_init(BOARD_INIT_UART_STDIO);

    uart_test();

return 0;
}
```

   以下内容针对 concerto.mk。

```
ifeq ($(TARGET_CPU),$(filter $(TARGET_CPU), x86_64 C71 C7120))

include $(PRELUDE)
TARGET      := c7_uart_print
TARGETTYPE  := library
CSOURCES    := $(call all-c-files)
CPPSOURCES  := $(call all-cpp-files)
CFLAGS+= -mv7100 --c11
ifeq ($(TARGET_CPU), x86_64)
IDIRS       += $(CGT7X_ROOT)/host_emulation/include/C7100
CFLAGS += --std=c++14 -D_HOST_EMULATION -pedantic -fPIC -w -c -g -o4
CFLAGS += -Wno-sign-compare
endif

include $(FINALE)

endif
```

3. 将输出日志从共享存储器更改为 UART FIFO。

```
diff --git a/vision_apps/utils/console_io/src/app_log_writer.c b/vision_apps/utils/
console_io/src/app_log_writer.c
index a02a785c..561d1434 100755
--- a/vision_apps/utils/console_io/src/app_log_writer.c
+++ b/vision_apps/utils/console_io/src/app_log_writer.c
@@ -220,6 +220,32 @@ int32_t  appLogWrPutString(app_log_wr_obj_t *obj)

     return status;
 }
+#if defined C71
+int32_t  c7x_appLogWrPutString(app_log_wr_obj_t *obj)
+{
+    int32_t status = 0;
+    volatile uint32_t copy_bytes,num_bytes;
+    volatile uint8_t *buf = (uint8_t*)obj->buf;
+
+
+    if (0 == status)
+    {
+        num_bytes = strlen((char*)buf);
+
+        if (num_bytes <= 0)
+        {
+            status = -1;
+        }
+    }
+
+    if (0 == status)
+    {
+        UART_puts(buf,num_bytes);
+    }
+
+    return status;
+}
+#endif

 void appLogPrintf(const char *format, ...)
 {
@@ -266,6 +292,8 @@ void appLogPrintf(const char *format, ...)
             printf(obj->buf);
             #endif
         }
+        #elif defined C71
+        c7x_appLogWrPutString(obj);
         #else
         appLogWrPutString(obj);
         #endif
```

4. 为编译二进制文件添加库路径。

```
diff --git a/vision_apps/platform/j721e/rtos/concerto_c7x_inc.mak b/vision_apps/platform/j721e/
rtos/concerto_c7x_inc.mak
index 4e3c5a29..4a9c94db 100755
--- a/vision_apps/platform/j721e/rtos/concerto_c7x_inc.mak
+++ b/vision_apps/platform/j721e/rtos/concerto_c7x_inc.mak
@@ -19,6 +19,10 @@ endif
 ifeq ($(RTOS),SAFERTOS)
        LDIRS += $(PDK_PATH)/packages/ti/osal/lib/safertos/$(SOC)/c7x/$(TARGET_BUILD)/
 endif
+
+LDIRS += $(PDK_PATH)/packages/ti/drv/uart/lib/$(SOC)/c7x/$(TARGET_BUILD)/
+LDIRS += $(PDK_PATH)/packages/ti/drv/i2c/lib/$(SOC)/c7x/$(TARGET_BUILD)/
+LDIRS += $(PDK_PATH)/packages/ti/board/lib/$(SOC)_evm/c7x/$(TARGET_BUILD)/
 LDIRS += $(PDK_PATH)/packages/ti/csl/lib/$(SOC)/c7x/$(TARGET_BUILD)/
 LDIRS += $(PDK_PATH)/packages/ti/drv/ipc/lib/$(SOC)/c7x_1/$(TARGET_BUILD)/
 LDIRS += $(PDK_PATH)/packages/ti/drv/udma/lib/$(SOC)/c7x_1/$(TARGET_BUILD)/
@@ -45,6 +49,7 @@ STATIC_LIBS += vx_app_ptk_demo_common
 STATIC_LIBS += vx_kernels_common
 STATIC_LIBS += vx_target_kernels_img_proc_c71
 STATIC_LIBS += vx_app_c7x_voxel2point
+STATIC_LIBS += c7_uart_print

 PTK_LIBS =
 PTK_LIBS += ptk_algos
@@ -76,6 +81,9 @@ ADDITIONAL_STATIC_LIBS += ipc.ae71
 ADDITIONAL_STATIC_LIBS += dmautils.ae71
```

```
 ADDITIONAL_STATIC_LIBS += sciclient.ae71
 ADDITIONAL_STATIC_LIBS += udma.ae71
+ADDITIONAL_STATIC_LIBS += ti.drv.uart.ae71
+ADDITIONAL_STATIC_LIBS += ti.board.ae71
+ADDITIONAL_STATIC_LIBS += ti.drv.i2c.ae71

 ifeq ($(RTOS),FREERTOS)
       ADDITIONAL_STATIC_LIBS += ti.kernel.freertos.ae71
```

上面介绍了 SDK 级别的所有更改。只需要执行下一步操作即可重新编译 C7 固件并将其刷写到 SD 卡或 EMMC 中。

5 总结

本应用手册旨在为您提供有关 UART 分析和常见 UART 应用的基本介绍。您可以参考该指南，根据硬件连接在电路板上设置您自己的软件设计。本文档还介绍了用户基于 UART 执行更好调试的方法。

6 参考资料

- 布线层
- 内核 UART 驱动程序
- PDK UART 用户界面
- 德州仪器 (TI)：*TDA4VM 技术参考手册*

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265