**SimpleLink SDK调试打印**

一般来说printf是您学习调试打印的第一种方法。在这里我们将介绍我们将探索以下在SimpleLink™SDK中也可用的API以及为什么printf不是在嵌入式设备上调试的最佳API的原因。在SimpleLink™SDK中也可用的API有以下几种：

* Display\_printf
* System\_printf
* UART\_PRINT
* UARTprintf

在这里将重点介绍通过UART和IDE的控制台console输出调试信息。

**硬件**

* 任何受支持的SimpleLink LaunchPad™开发套件(实验使用MSP432P401R LAUNCHPAD )

**软件**

* CCS（在SimpleLink SDK的发行说明中指定的版本，实验使用CCS9.2）
* 任何SimpleLink SDK（实验中使用的是simplelink\_msp432p4\_sdk\_3\_40\_00\_05）
* 具有图1设置的任何终端程序（如果使用Windows，请使用设备管理器确定端口）



图1 UART设置

**API说明**

将概述了每个API以及相关差别。

* printf

printf从一开始就是“ C”语言的一部分。这是一个众所周知的API，大多数工程师都会选择使用它。printf 适用于TI-RTOS，FreeRTOS和no-RTOS应用程序。但是printf对于嵌入式开发而言并不理想。让我们看看printf工作原理以及相关原因。

* IDE Console Output

默认情况下，在CCS中（或IAR），printf输出将进入CIO（“ C” Input/Output）缓冲区。当CIO缓冲区已满或写入EOL字符（即'\ n'）时，将命中一个断点，CCS会读取该缓冲区的内容。然后CCS会resume target。由此看来，此操作可能会对实时性能产生非常严重的影响。

* Non IDE Console Output

printf输出可以重定向。对于下面的对比中，我们就通过add\_device，freopen和setvbuf重定向到UART 。

* 相关限制

printf当输出到IDE控制台时，可以从任何位置（例如main，ISR，task等）调用它，但有一个例外。默认情况下，在TI-RTOS将在TI Compiler中使用锁定机制locking mechanism（GateMutex）以确保线程安全。TI Compiler中printf会调用此锁定机制。因此在默认情况下，在使用TI-RTOS和TI编译器的应用程序中，不能在Hwi或Swi中调用printf。如果您在TI-RTOS内核中启用了assert，则会得到assert失败。如果未启用assert，则可能会得到不确定的结果。

可以通过将BIOS.rtsGateType 内核配置文件（即.cfg文件）中的变量 设置为来禁用锁定BIOS.NoLocking。

* 更多信息可以参考Tips for Using Printf

（<http://software-dl.ti.com/ccs/esd/documents/sdto_cgt_tips_for_using_printf.html>）

* Display\_printf

Display\_printf允许用户将输出发送到不同位置：LCD，IDE控制台，或UART。我们将专注于IDE控制台和UART。 Display\_printf的格式选项与printf相似。Display\_printf 在基于RTOS和no-RTOS的应用程序中都可用。许多SDK示例都使用Display来通过UART输出数据，以实现与OS和IDE的隔离。Display\_open和Display\_config 变量中指定的参数指示使用哪种机制。

* IDE 控制台 (Host)

Display\_printf也可以使用DisplayHost定向到IDE的控制台。Display\_printf使用HOST时，将忽略行和列中的参数，如图2

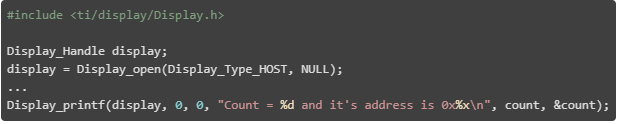


图2 IDE Console

* UART

UART接口有两种类型：

DisplayUartAnsi：功能丰富的模块，但会占用空间。

DisplayUartMin：简单的UART管理，功能较少，但体积较小。

大多数示例的默认值是DisplayUartMin。Display\_printf使用UART时，将忽略其中的行和列参数，如图3。

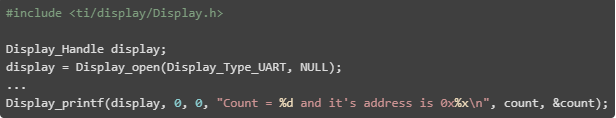


图3 UART

* 相关限制
* IDE控制台

DisplayHost没有调用限制（只要通过写入CIO缓冲区引起的潜在实时影响是可以接受的）。

* UART

如果您使用的是RTOS，则只能从任务（或pthread）中调用DisplayUart。如果您不使用RTOS，则可以从main（或通过在main上下文中调用的任何函数）调用DisplayUart 。

* 更多信息可以参考"SimpleLink MCU SDK User's Guide" and "Drivers Runtime APIs (doxygen)"
* System\_printf

System\_printf仅在应用程序使用TI-RTOS时可用。System模块允许应用程序插入其他系统“Proxy”。Proxy负责基础实现。主要的支持代理为SysMin以及SysCallback。您可以在内核配置文件（例如.cfg文件）中分配Proxy。下面是将SysMin分配为支持Proxy的代码示例，如图4。



图4 SysMin分配为支持Proxy

System\_printf无论底层Proxy是什么，顶层都是相同的。以下是API的示例，如图5。



图5 API示例

* SysMin（ROV / IDE控制台输出）

SysMin模块将ASCII数据存储到内部缓冲区中。内部缓冲区的大小是可配置的。除非有指示，否则数据**不会**刷新到CIO缓冲区（例如，调用System\_flush或配置为在应用程序上刷新exit）。因此SysMin不会影响实时性能。您可以通过查看ROV→SysMin→OutputBuffer来查看该缓冲区。

* SysCallback（UART输出）

SysCallback模块在格式完成后允许用户插入类似putc的函数（putchFxn）。默认的putchFxn函数不执行任何操作。常见的用例是插入一个putchFxn函数，该函数将字符写入UART。

* 相关限制
  + - IDE控制台

使用SysMin时，没有限制。

* + - UART

使用SysCallback和UART时，调用限制取决于基础代码。System\_printf可以从基于TI-RTOS的应用程序中的任务（或pthread）中调用。

* 更多详细信息可以在下图中的路径内找到： simplelink\_msp432p4\_sdk\_3\_40\_00\_05\docs\tirtos\sysbios\docs\cdoc\xdc\runtime
* UARTprintf

UARTprintf仅适用于SimpleLink MSP432E4 SDK中的no-RTOS示例。出于遗留原因将其包括在内。此API仅支持通过UART输出。 该代码在 某些SimpleLink MSP432E4 SDK示例所包含的uartstdio.c和uartstdio.h文件中进行维护，如图6所示。



图6 UARTprintf

* 相关限制
  + - IDE控制台

不适用

* + - UART

可以从任何上下文调用此API，但仅在MSP432E4设备的非RTOS示例中进行了测试。

* 更多信息请参考SDK示例中的uartstdio.c和uartstdio.h文件的源代码。

**相关实验**

下面的实验是在simplelink\_msp432p4\_sdk\_3\_40\_00\_05\examples\rtos\MSP\_EXP432P401R\drivers\gpiointerrupt的例程基础上进行修改的。printf 从控制台输出，请按照下面图7的高亮提示进行代码修改

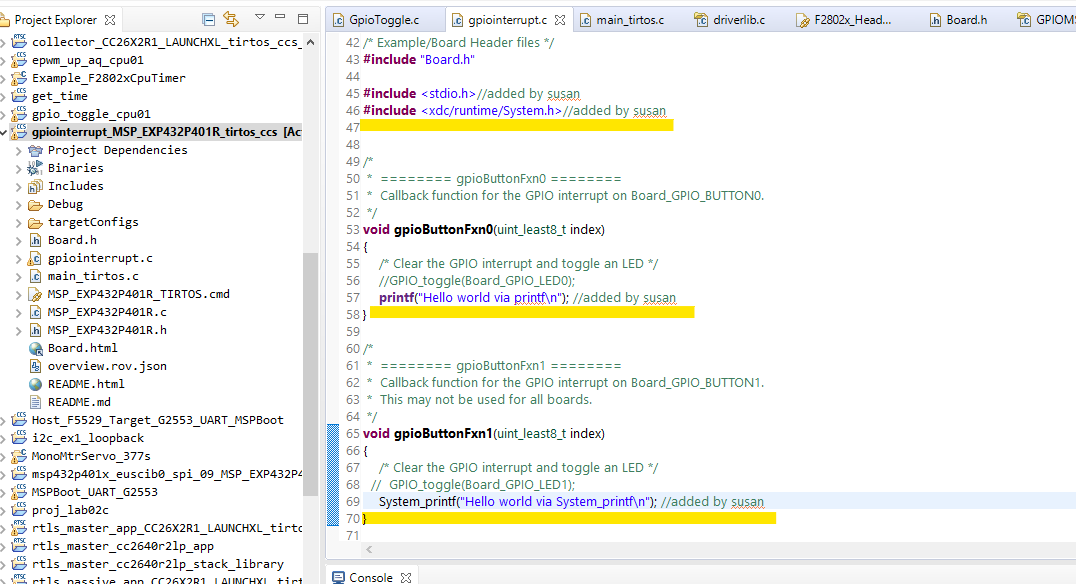


图7 代码修改

在mainThread 的return之前添加下面的代码，如图8

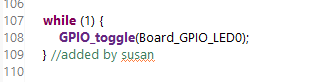


图8 mainThread代码修改

运行结果如下图9，控制台可以输出Hello world via printf，但是不能输出Hello world via System\_printf。

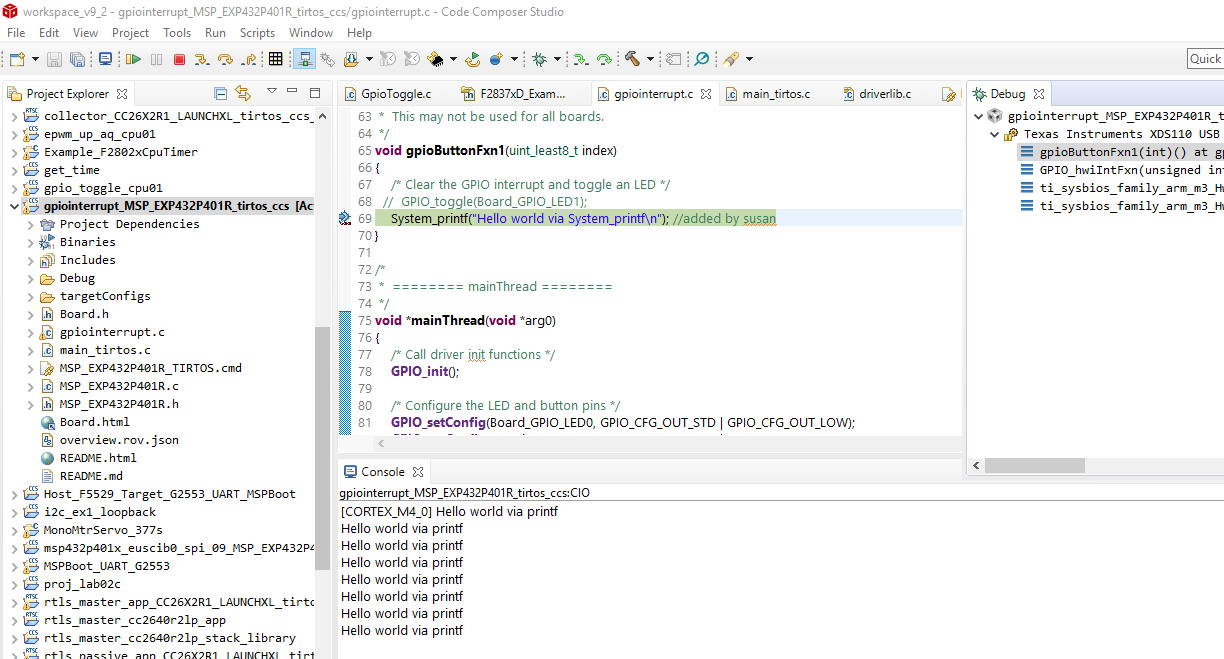


图9 运行结果

下面就看一下System\_printf的使用

* System\_printf 从控制台输出

TI大多数示例都将SysCallback作为默认的系统proxy ，而默认的功能则什么也不做。

接下来的实验中，我们将使用SysMin System proxy。可以通过以下三种方式之一来完成，以允许将输出字符串放置在可通过ROV查看的内部缓冲区中。

* 更改release.cfg文件以使用SysMin：最简单的方法，但是您应该查看这是否会对使用此内核项目的其他项目产生影响。
* 创建并使用新的内核项目：您可以复制/粘贴发行版内核项目，将其重命名，使用SysMin并将GPIO Interrupt指向新的内核项目。这将最小化对其他项目的影响。
* 使用debug项目：由于调试内核项目使用SysMin以及许多其他调试功能，因此这也将起作用。

我们将使用最后一种方法。在此导入simplelink\_msp432p4\_sdk\_3\_10\_00\_08\kernel\tirtos\builds\MSP\_EXP432P401R\debug\ccs工程，而后进行编译。之后进行gpiointerrupt工程的修改，如图10

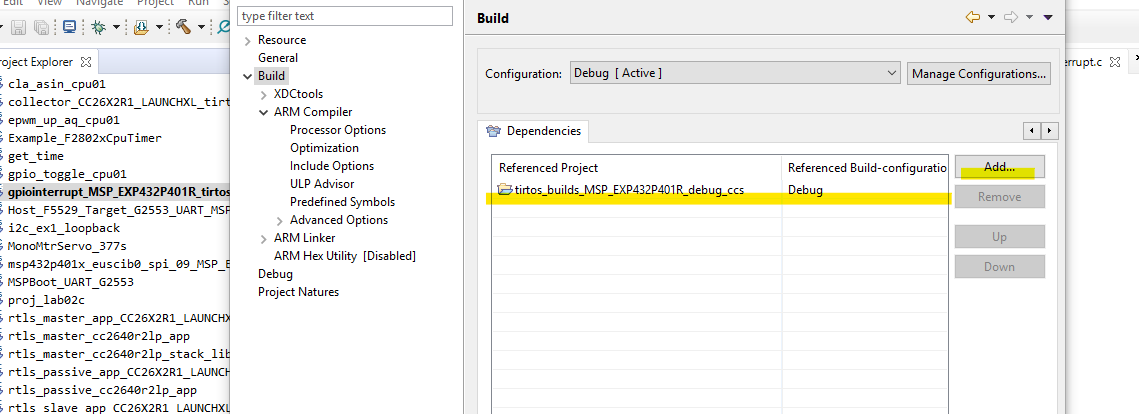


图10 工程设置修改

您需要在之前的代码中加上System\_flush();，

**while** (1) {

GPIO\_toggle(Board\_GPIO\_LED0);

System\_flush();

} //added by susan

Build, load并run之后可以在控制台看到输出，如图11所示。

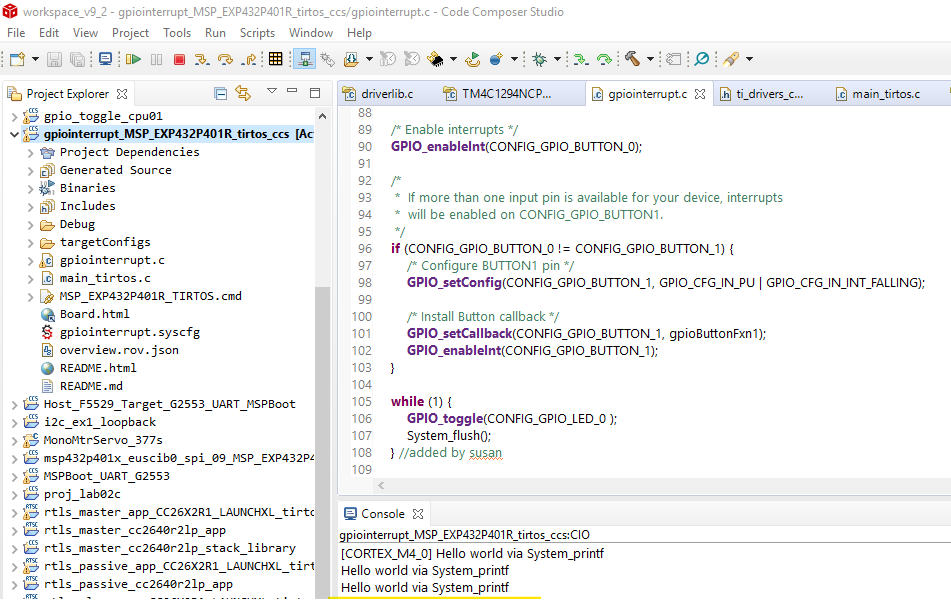


图11 控制台输出

* Display\_printf  从UART输出

在此我们使用的是empty的例程，默认路径如下

simplelink\_msp432p4\_sdk\_3\_40\_00\_05\examples\rtos\MSP\_EXP432P401R\drivers\empty

首先需要向工程内添加Display driver。双击empty.syscfg 点击ADD以添加Display 模块，如下图12进行设置

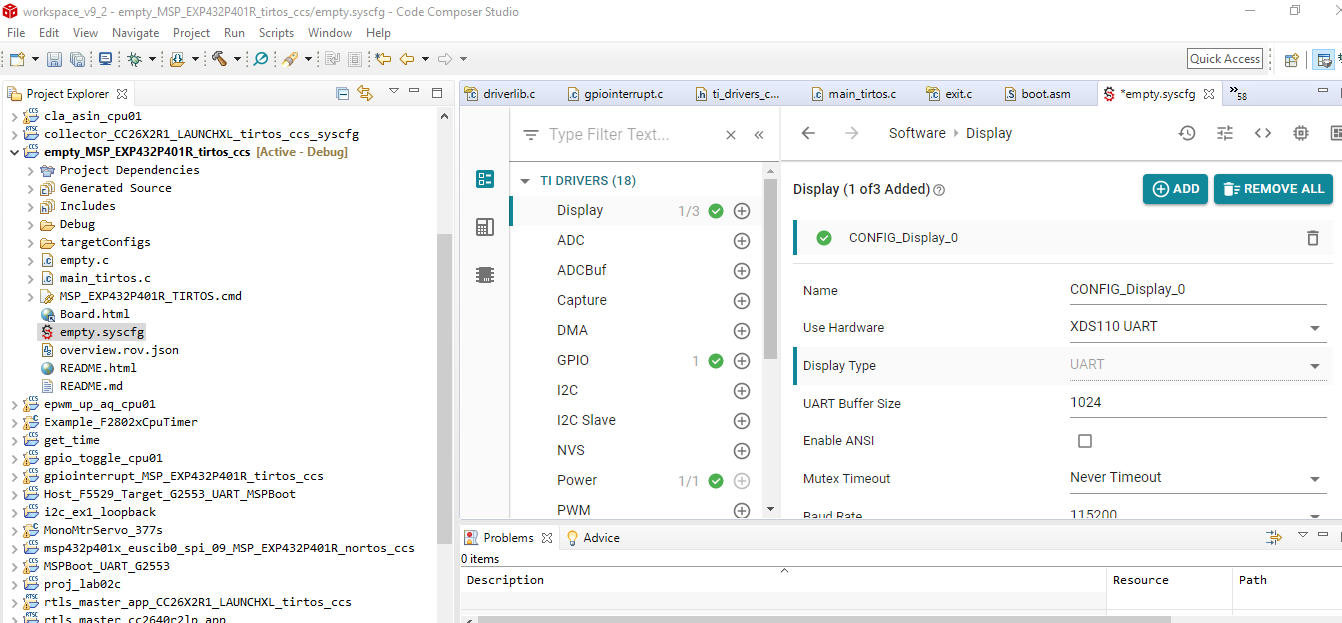


图12 empty.syscfg修改

然后在 empty.c中mainThread的开始添加下面的代码：

Display\_Handle display;

display = Display\_open(Display\_Type\_UART, NULL);//added by susan

在mainThread末尾的while（1）中添加

Display\_printf(display, 0, 0, "Hello world via Display\_printf");//add by susan

在empty.c中添加头文件

#include <ti/display/Display.h>//added by susan

Build, load and run 工程，可以通过串口调试工具得到输出结果，如下图13所示

（在此使用的是CCS内置的串口工具）

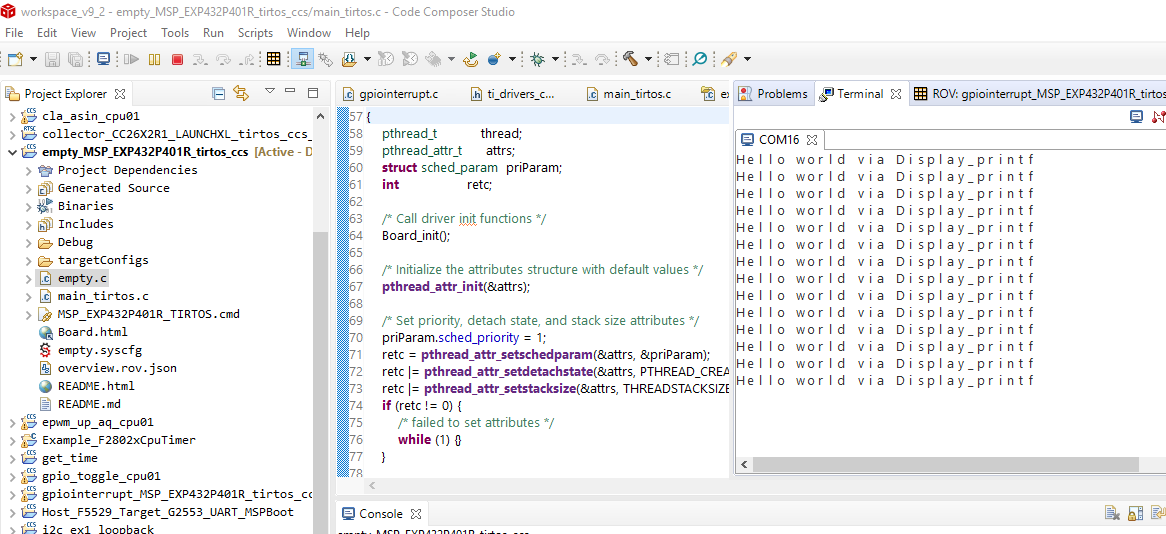


图13 CCS内置的串口工具输出

**总结**

以上介绍了几种在SimpleLink™SDK中也可用的API以及实际的使用方法，希望对大家有所帮助。欢迎大家讨论或指正相关的问题，谢谢！