

## Application Note

## 在 AM64x 和 AM243x 上启用五个以太网端口



Ashwani Goel, Teja Rowthu

Sitara MPU

## 摘要

本文档为德州仪器 (TI) 基于 Arm® 的片上系统 (SoC) AM64x 和 AM243x 器件上 (支持使用 R5F 和 A53 内核) 五个以太网®端口的设计、开发、实施和性能指标提供了指导。目前, 由于通用平台以太网交换机 (CPSW3G) 和工业通信子系统 (PRU-ICSSG) 以太网端口同时工作, 因此该实施仅适用于基于 RTOS 的多核方案。

目前, 公开发布的 [MCU+ SDK](#) 不支持该实施。请关注[常见问题解答](#), 了解更多更新的详细信息。

## 内容

1 引言.....	3
1.1 AM64x 和 AM243x EVM.....	3
1.2 SoC 架构.....	3
1.3 外设.....	5
1.4 以太网软件架构.....	6
1.5 先决条件.....	7
2 多核 5 以太网端口实现.....	11
3 PRU-ICSSG 上支持的配置.....	12
4 实施.....	12
4.1 系统示例.....	13
5 调试步骤.....	15
6 参考日志.....	19
7 ICSSG0 和 ICSSG1 功能测试.....	22
8 ICSSG 和 CPSW.....	22
9 总结.....	23
10 参考资料.....	23

## 插图清单

图 1-1. AM64x.....	4
图 1-2. AM243x.....	5
图 1-3. 以太网网络层.....	6
图 1-4. EVM 方框图.....	7
图 1-5. HSE 连接器: ICSSG0 引脚排列.....	8
图 1-6. SysConfig: 数据包 DMA.....	9
图 1-7. SysConfig: 数据包 DMA 环形加速器.....	10
图 4-1. SysConfig: IPC 设置.....	12
图 4-2. ICSSG 和 CPSW 分配.....	13
图 4-3. 示例的简化流程图.....	14
图 5-1. SysConfig: ICSSG0 和 ICSSG1: MII 模式: 对 R5FSS0-0 的分配.....	15
图 5-2. SysConfig: INTC 配置: PRU 引脚 41.....	16
图 5-3. SysConfig: INTC 配置: PRU 引脚 53.....	16
图 5-4. SysConfig: GPIO 引脚 31: DP83826e PHY 复位引脚.....	17
图 5-5. SysConfig: GPIO 引脚 32: DP83826e PHY 复位引脚.....	18

## 商标

Sitara™ is a trademark of Texas Instruments.

Yocto Project™ is a trademark of The Linux Foundation.

Arm® and Cortex® are registered trademarks of Arm Limited.

以太网® is a registered trademark of Xerox Corporation.

EtherCAT® is a registered trademark of Beckhoff Automation GmbH.

PROFINET® is a registered trademark of PROFIBUS Nutzerorganisation e.V. (PNO).

所有商标均为其各自所有者的财产。

## 1 引言

### 1.1 AM64x 和 AM243x EVM

AM64x 和 AM243x 是 Sitara™ MCU 系列中的工业级器件。

AM64x 器件专为需要结合实时通信和处理的工业应用（例如电机驱动和远程 I/O 模块）而构建。

AM64x 系列通过多达两个支持 TSN 技术的千兆位 PRU\_ICSSG 实例，在 Sitara 系列中提供可扩展的性能。有关更多信息，请参阅 [AM64x 和 AM243x 技术参考手册](#)。

### 1.2 SoC 架构

本节介绍了 SITARA MPU 器件 AM64x 和 AM243x 的 SOC 级描述。

#### 1.2.1 AM64x

AM64x 器件是 Sitara MCU 工业级异构 Arm 处理器产品系列的新增产品，专为要求独特结合实时处理和通信与应用处理的电机驱动器和可编程逻辑控制器 (PLC) 等工业应用而构建。AM64x 组合了两个支持 TSN 技术的 Sitara 器件千兆位 PRU-ICSSG 实例、多达两个 Arm® Cortex®-A53 内核、多达四个 Cortex-R5F MCU 和一个 Cortex-M4F MCU。AM64x 旨在通过高性能 R5F 内核、紧密耦合的存储器组、可配置的 SRAM 分区和进出外设的专用低延迟路径提供出色的实时性能，从而实现数据快速进出 SoC。这种确定性架构允许 AM64x 处理伺服驱动器中的严格控制环路，同时快速串行接口 (FSI)、通用存储器控制器 (GPMC)、脉宽调制 (PWM)、 $\Delta$ - $\Sigma$  抽取滤波器和绝对编码器接口等外设可帮助启用这些系统中的多种不同架构。

Cortex-A53 提供了 Linux 应用所必需的强大计算元件。Linux 和实时 (RT) Linux 则通过 TI 的 Processor SDK Linux 提供，后者会每年更新为最新的长期支持 (LTS) Linux 内核、引导加载程序和 Yocto Project™ 文件系统。AM64x 通过可配置的存储器分区实现 Linux 应用与实时数据流之间的隔离。可以指定 Cortex-A53 严格使用 DDR 运行 Linux，而内部 SRAM 可以拆分成不同大小，供 Cortex-R5F 一起使用或单独使用。AM64x 提供灵活的工业通信能力，包括用于 EtherCAT® 子器件、PROFINET® 器件、以太网、图像提示 (IP) 适配器和 IO-Link 控制器的全协议栈。PRU-ICSSG 进一步提供了千兆位和基于 TSN 技术的协议所需的能力。此外，PRU-ICSSG 还支持 SoC 中的其他接口，包括  $\Sigma$ - $\Delta$  抽取滤波器和绝对编码器接口。可通过集成的 Cortex-M4F 及其专用外设启用功能安全特性，这些外设可与 SoC 的其余部分隔离。AM64x 还支持安全启动。

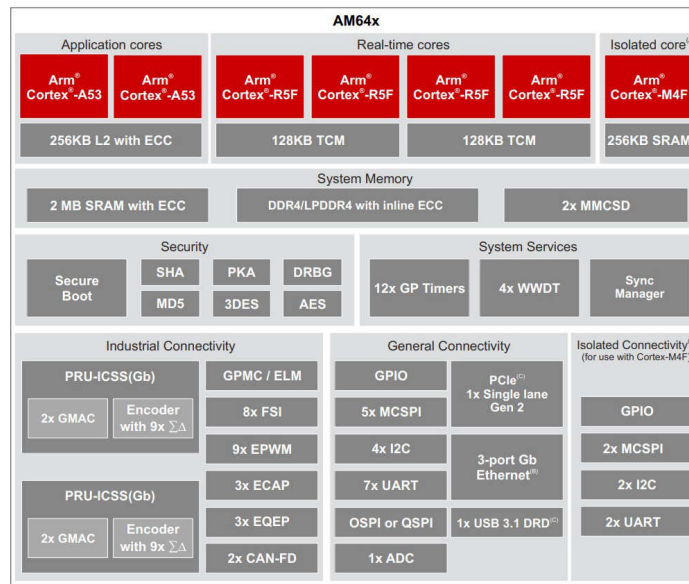


图 1-1. AM64x

### 1.2.2 AM243x

AM243x 器件是采用 Sitara 技术的高性能微控制器新增的工业级产品系列，专为需要结合实时通信和处理的工业应用（例如电机驱动和远程 I/O 模块）而构建。AM243x 系列通过多达四个 Cortex-R5F MCU、一个 Cortex-M4F 和两个支持 TSN 的千兆位 PRU\_ICSSG 实例，在 Sitara 微控制器中提供了可扩展的性能。AM243x SoC 架构旨在通过高性能 Arm Cortex-R5F 内核、紧密耦合的存储器组、可配置的 SRAM 分区和与外设之间的专用低延迟路径提供出色的实时性能，从而实现数据快速进出 SoC。

这种确定性架构允许 AM243x 处理伺服驱动器中的严格控制环路，同时 FSI、GPMC、增强型捕获模块 (eCAP)、PWM 和编码器接口等外设可帮助启用这些系统中的多种不同架构。该 SoC 提供灵活的工业通信能力，包括用于 EtherCAT 目标、PROFINET 器件、以太网或 IP 适配器和 IO-Link 控制器的完整协议栈。PRU\_ICSSG 进一步提供了千兆位和基于 TSN 技术的协议所需的能力。PRU\_ICSSG 还支持其他接口，包括 UART 接口、 $\Delta$ - $\Sigma$  抽取滤波器和绝对编码器接口。可通过集成的 Cortex-M4F 及其专用外设启用功能安全特性，这些外设可与 SoC 的其余部分隔离。AM243x 还支持安全启动。

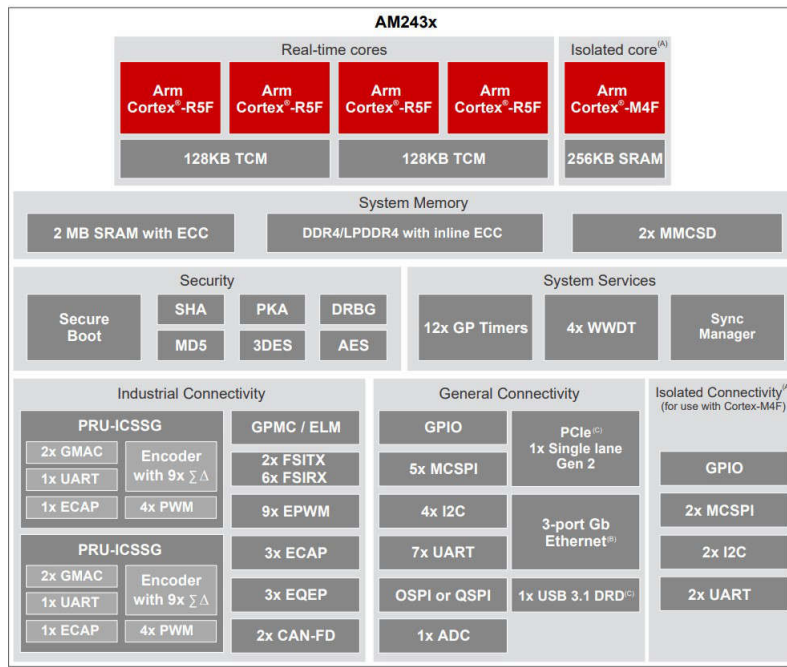


图 1-2. AM243x

### 1.3 外设

本节简要介绍了 AM64x/AM243x 上可用的 PRU-ICSSG 和 CPSW3G 外设。

#### 1.3.1 CPSW3G

**CPSW (通用平台交换机)** : CPSW 子系统为器件提供 IEEE 802.3 标准以太网千兆位速度数据包通信，也可配置为以太网交换机。

**小心**  
CPSW 在图 1-1 和图 1-2 中显示为 3 端口千兆位以太网。

**小心**  
AM64x 和 AM243x 上的 CPSW3G 支持 RGMII 和 RMII 接口。

#### 1.3.2 PRU-ICSSG

**可编程实时单元和工业通信子系统千兆位 (PRU-ICSSG)** : PRU-ICSSG 可通过固件进行编程，并可实现多种功能，如工业通信协议交换机 (适用于 EtherCAT、Profinet、以太网/IP 等协议)、以太网交换机、以太网 MAC、工业驱动器等。

**小心**  
AM64x/AM243x 上的 PRU-ICSSG 仅支持 RGMII 和 MII 模式。

**小心**  
PRU-ICSSG 在图 1-1 和图 1-2 中显示为 PRU-ICSSG。

### 1.4 以太网软件架构

图 1-3 中展示了软件组件概览，重点介绍了网络软件开发过程中使用的组件。

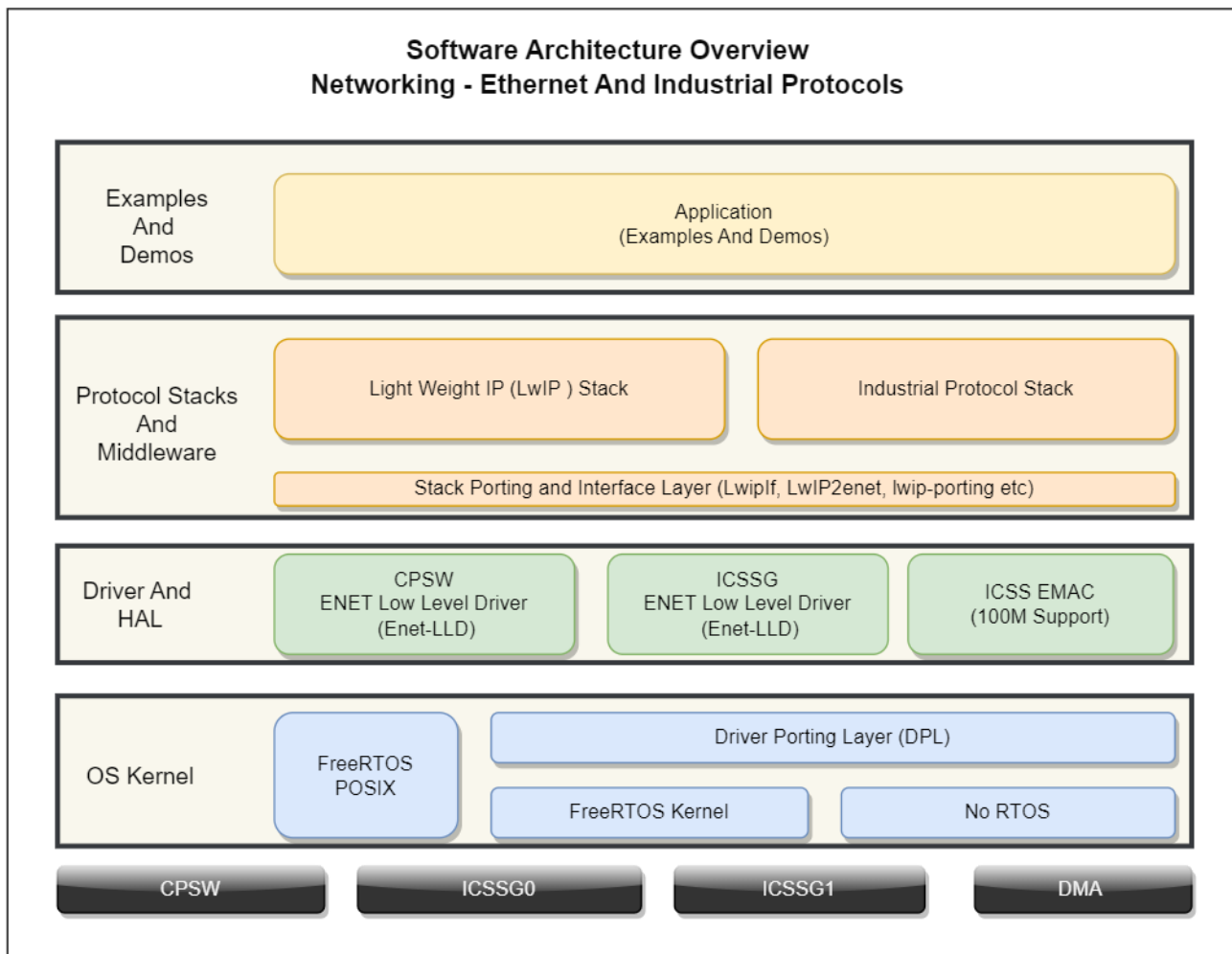


图 1-3. 以太网网络层

## 1.5 先决条件

这些是使用 AM64x-EVM 和 SEM 板启用五个以太网端口的先决条件。

### 1.5.1 硬件要求

- AM64x/AM243x-EVM

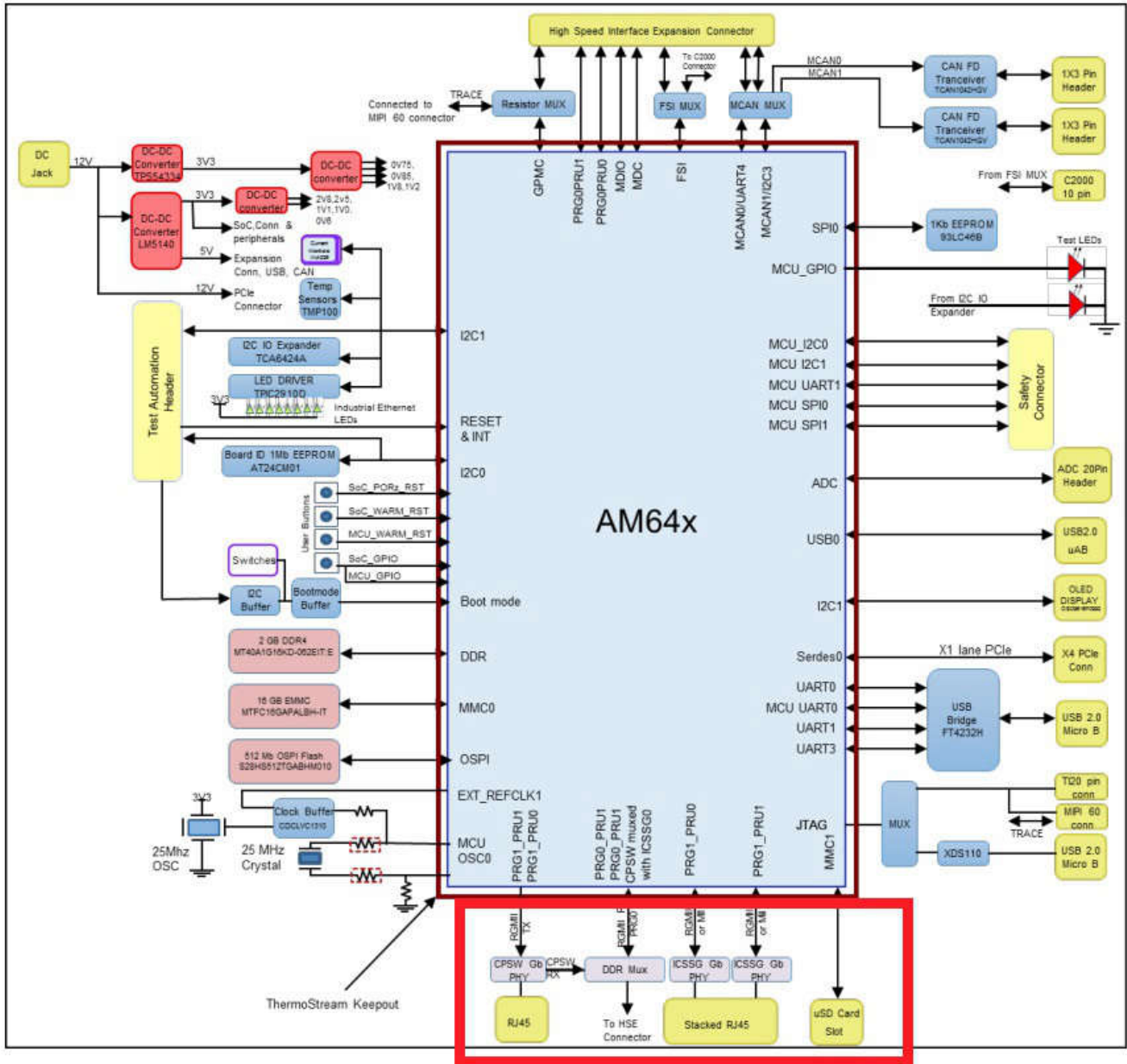


图 1-4. EVM 方框图

- SEM 板通过 HSE 连接器进行连接
  - DP83826e PHY
  - 测试时仅支持 100M。

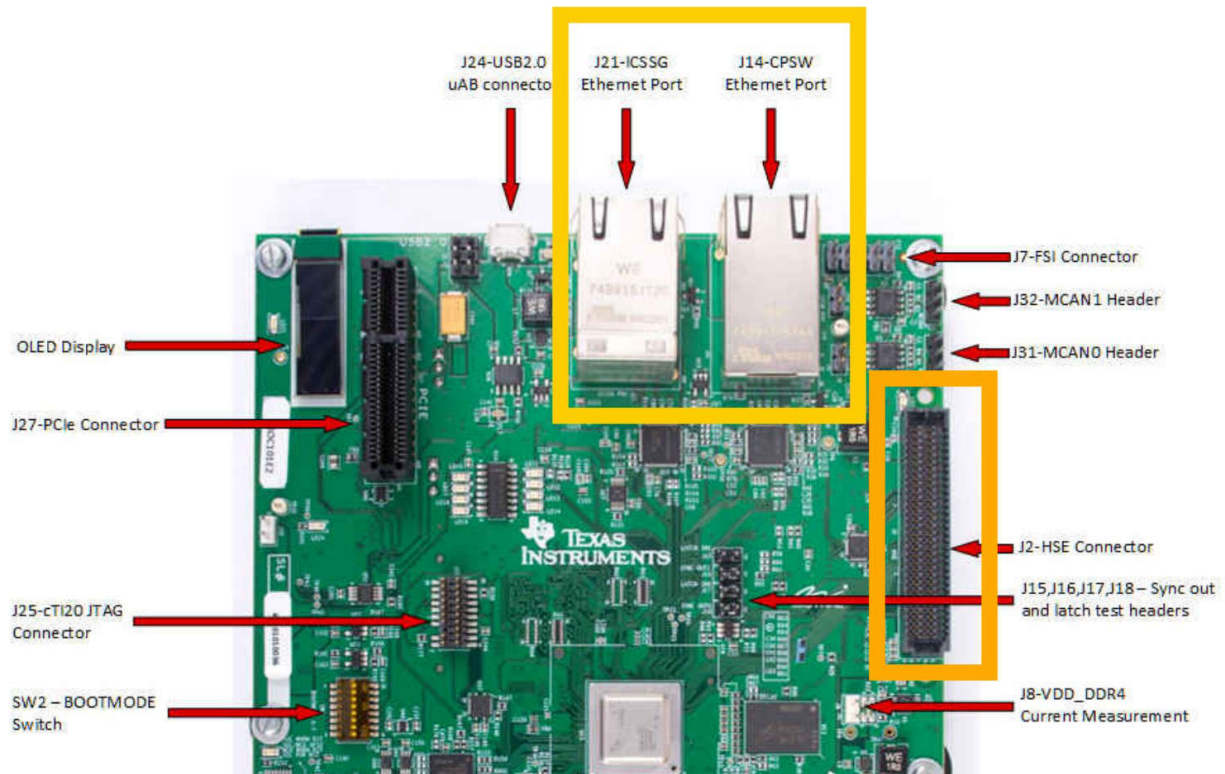


图 1-5. HSE 连接器：ICSSG0 引脚排列

### 1.5.2 软件要求

- 安装用于 [AM64x](#) 或 [AM243x](#) 的 MCU-PLUS-SDK 及相关工具。如需更多信息，请参阅[入门](#)。
- [SYSCONFIG](#)

#### 小心

确保先更新 SBL，然后再在电路板上运行该示例。默认 SBL 不允许 CPSW 在 R5FSS\_0-1 上运行。



### 1.5.2.1 资源分配 - AM64x

首先必须使用资源管理器来将任何外设分配给任何内核。用户可以使用 [sysconfig 工具](#) 以不同模式 ( MII、RMII、RGMII ) 配置 ICSSG0、ICSSG1 和 CPSW 以太网端口，并根据需要连接到内核。

#### 小心

以下示例适用于具有 A53 内核的 AM64x：

AM243x 没有 A53 内核。

根据用例进行资源分配。

使用以下步骤启用该特性：

1. 导航至 MCU-PLUS-SDK 安装目录 ( 在本文档中，这称为 MCU\_SDK\_HOME )。  
例如：`C:\ti\mcu_plus_sdk_am64x_09_01_00_41`
2. 打开命令提示符并运行以下命令：  
`gmake -s -C tools/sysfw/boardcfg configure SOC=am64x`
3. 在 A53 内核中将 CPSW、ICSSG0 和 ICSSG1 的 Tx 和 Rx 通道数更改为 0。

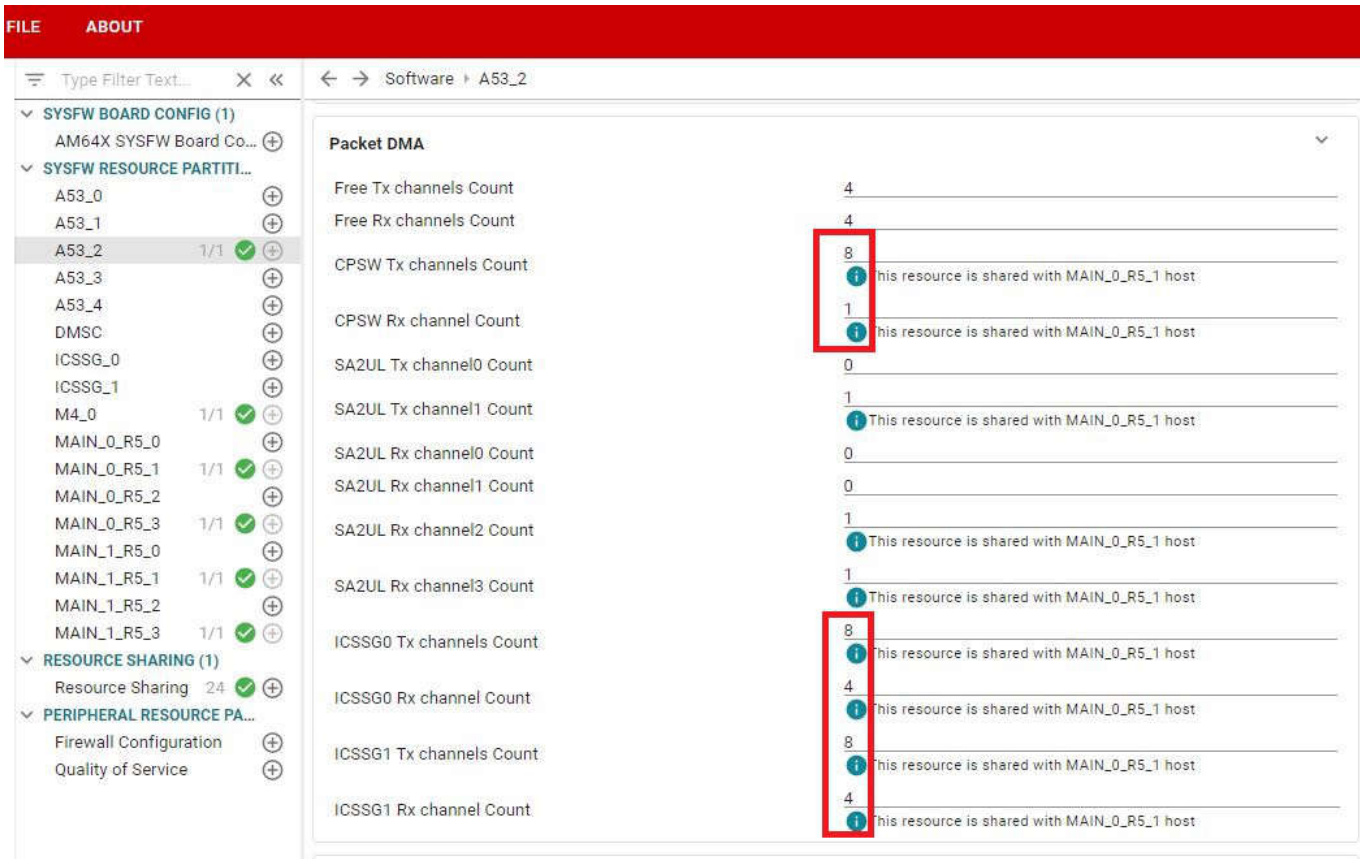


图 1-6. SysConfig : 数据包 DMA

4. 更新通道数后，值如下所示：
- CPSW Tx 通道数 = 0
  - CPSW Rx 通道数 = 0
  - ICSSG0 Tx 通道数 = 0
  - ICSSG0 Rx 通道数 = 0
  - ICSSG1 Tx 通道数 = 0
  - ICSSG1 Rx 通道数 = 0

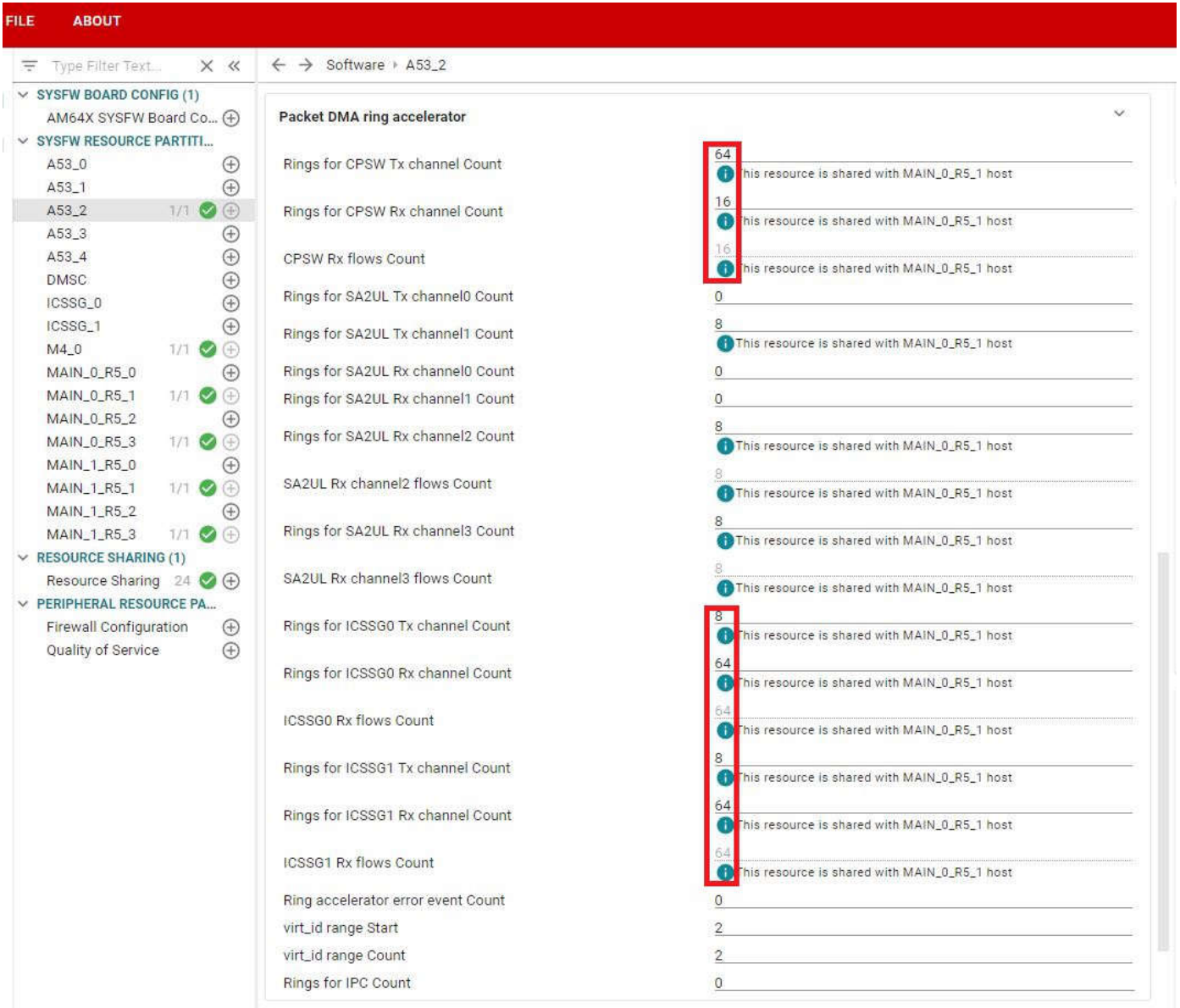


图 1-7. SysConfig : 数据包 DMA 环形加速器

- CPSW Tx 通道的环数 = 0
  - CPSW Rx 通道的环数 = 0
  - CPSW Rx 流的环数 = 0
  - ICSSG0 Tx 通道的环数 = 0
  - ICSSG0 Rx 通道的环数 = 0
  - ICSSG0 Rx 流的环数 = 0
  - ICSSG1 Tx 通道的环数 = 0
  - ICSSG1 Rx 通道的环数 = 0
  - ICSSG1 Rx 流的环数 = 0
5. R5 非安全 0-1 内核 (main-0-r5-0-1) 中的 (0,0) 更改为 (8,1)。
  6. 将资源共享从 A52 -> r5\_ss\_0-1 更改为 r5\_ss\_0-1 -> r5\_ss\_0-3。
  7. `gmake -s -C tools/sysfw/boardcfg configure-gen SOC=am64x`
  8. 导航至以下文件。  
`source/drivers/sciclient/sciclient_default_boardcfg/am64x/  
sciclient_defaultBoardcfg_rm.c`
  9. 对于 CPSW 相关更改，将内核分配从 A53 -> r5\_ss\_0-1 更改为 r5\_ss\_0-1 -> r5\_ss\_0-3。有三处 A53 更改和  
三处 r5-0-1 更改。

### 1.5.2.2 SBL 更新

#### 小心

此部分仅适用于使用 SBL 引导的情况。

请按照以下步骤操作：

- 对于编译 SBL：

```
gmake -s -C tools/sysfw/boardcfg sciclient_boardcfg SOC=am64x  
gmake -j -s -f makefile.am64x sb1-clean  
gmake -j -s -f makefile.am64x sb1
```

- 对于刷写 SBL：

```
cd tools/boot/  
python3 uart_uniflash.py -p /dev/ttyUSB0 --cfg=sb1_prebuilt/am64x-evm/default_sb1_null.cfg
```

## 2 多核 5 以太网端口实现

多核 5 以太网端口可通过以下两种方法之一实现：

- 2 个以太网端口来自 ICSSG0，2 个以太网端口来自 ICSSG1，1 个以太网端口来自 CPSW。

或

- 2 个以太网端口来自 ICSSG0，1 个以太网端口来自 ICSSG1，2 个以太网端口来自 CPSW。

以上是可通过使用具有 HSE 端口的 SEM 扩展卡（为 ICSSG0 添加两个额外的以太网端口）实现的方法的详细信息。

### 3 PRU-ICSSG 上支持的配置

以下链接显示了 ICSSG0/1 上支持的以太网配置：[ICSSG0/1 上可能有哪些以太网配置？](#)。

### 4 实施

HSE 端口上的 SEM 扩展卡用于为 ICSSG0 添加两个额外的以太网端口，为现有 ICSSG1 以太网端口添加两个以太网端口。这四个 ICSSG 以太网端口由 R5FSS\_0-0 控制。这通过实施 ICSSG0 和 ICSSG1 来实现。有关更多信息，请参阅[以太网 PRU\\_ICSSG 实例 0 \(PRU\\_ICSSG\) 使用指南](#)。

用户可以使用 ICSSG0 和 ICSSG1 释放多达四个以太网端口。这可以是：

- 2 对 (2 端口以太网交换机模式)，
- 或 4 个以太网端口 (MAC 模式)，
- 或任何其他组合。

该特定实施使用 ICSSG 的全部四个以太网端口。用户可从 ICSSG0 配置两个以太网端口，向 ICSSG1 配置一个以太网端口。

控制 CPSW 的驱动器应在 R5FSS\_0-1 CPU 内核上运行，以 MAC 模式控制单个以太网端口。该实施可根据要求进行更改，并可运行两个板载 RGMII 以太网端口。

两个内核均通过 IPC 进行通信。

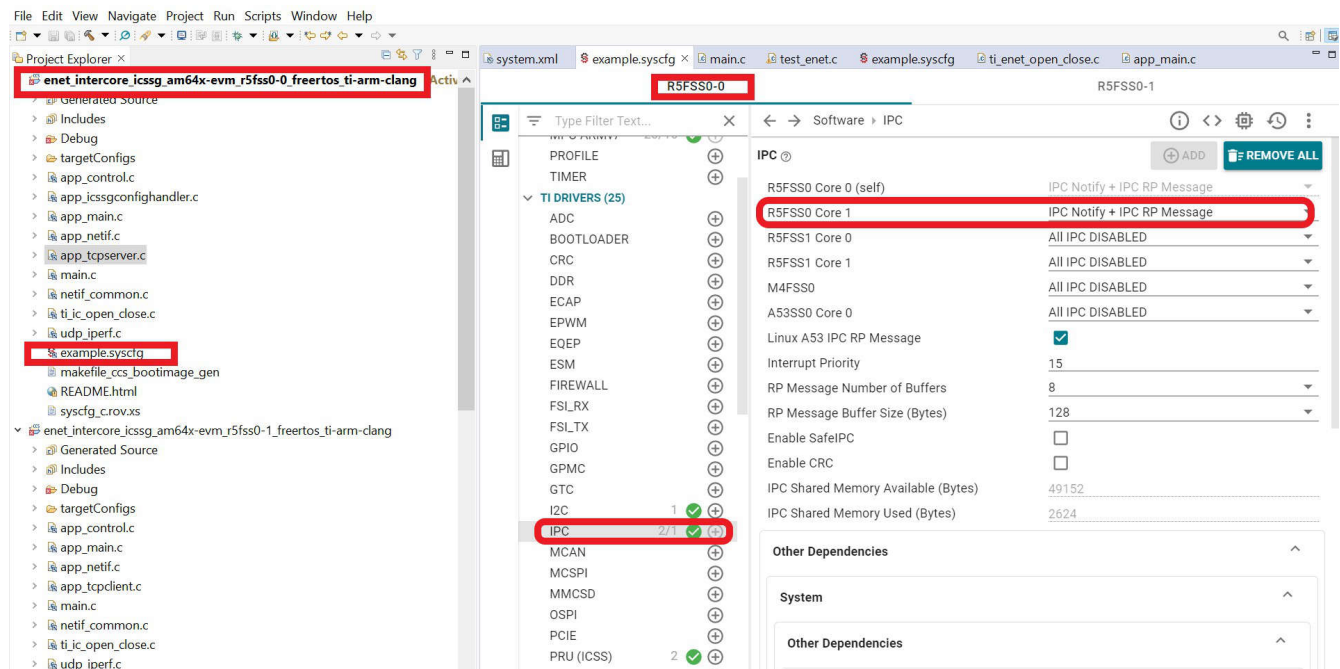


图 4-1. SysConfig : IPC 设置

#### 小心

请注意，需要使用系统工程来运行和触发包含的 IPC 相关代码。

IPC 示例必须编译为系统工程。单核编译失败，因为 IPC 代码生成取决于应用程序中的所有内核上下文。

## 4.1 系统示例

以下链接提供了系统示例：[TI 测试](#)。该示例适用于 AM243x 和 AM64x 器件。在 AM64x 系列上测试了这一过程。AM64x 配置为 4× ( MAC 模式 ) + 1× ( MAC 模式 ) 以太网端口。

**小心**  
在测试期间，AM243x 配置为 2× ( 2 端口以太网交换机模式 ) + 1× ( MAC 模式 )。

使用系统 make 命令构建示例。将 .out 文件加载到相应的内核，或以 UART 或 OSPI 模式刷写组合的应用程序映像。

```
Example:
make -s -C test/networking/lwip/enet_icssg_tcpserver/am64x-evm/system_cpsw_icssg clean
make -s -C test/networking/lwip/enet_icssg_tcpserver/am64x-evm/system_cpsw_icssg
make -s -C test/networking/lwip/enet_icssg_tcpserver/am64x-evm/system_cpsw_icssg all
```

### 4.1.1 软件架构

图 4-2 展示了 5 以太网端口分配的总体软件架构。

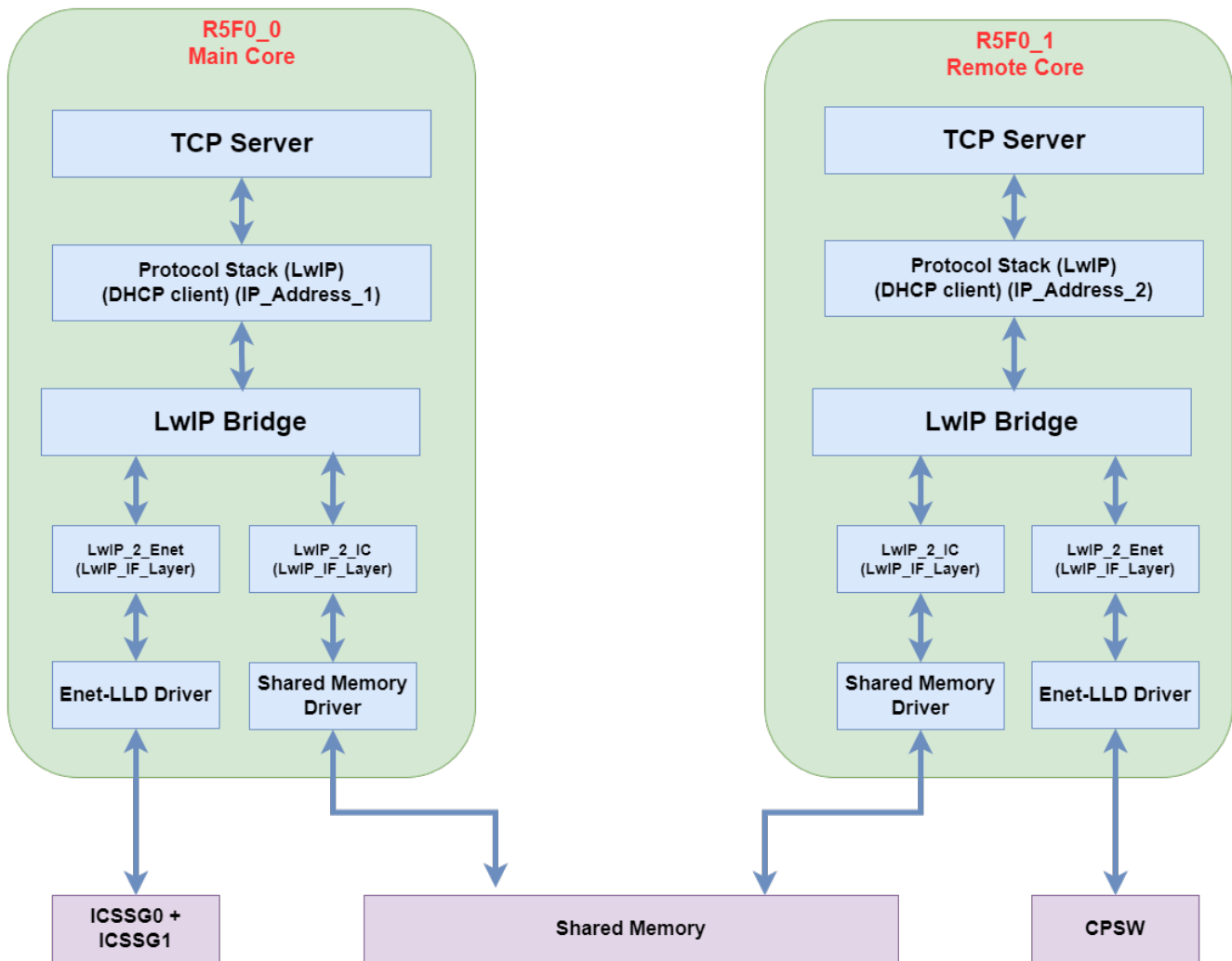


图 4-2. ICSSG 和 CPSW 分配

### 4.1.2.5 以太网端口示例

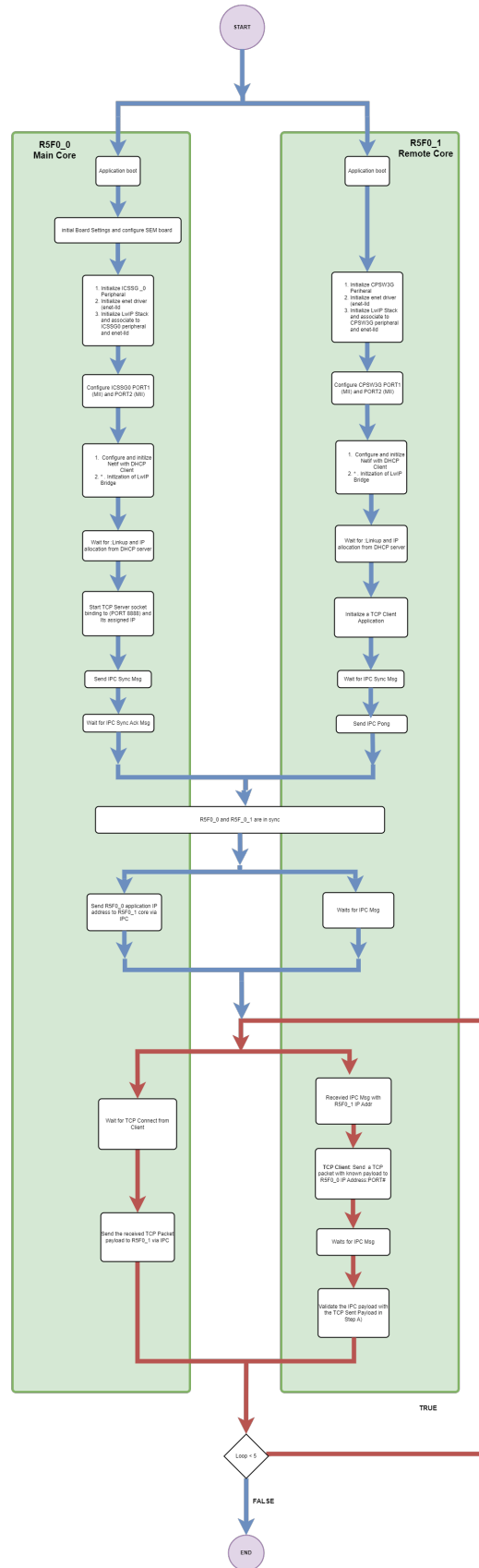


图 4-3. 示例的简化流程图

## 5 调试步骤

1. linker.cmd 文件必须包含未缓存的共享存储器区域所需的内容。
  - a. 必须在 syscfg 中指定该区域。
2. 首先在 syscfg-GUI 中打开 ICSSG1 模块。
  - a. 接下来，打开 ICSSG0 模块。
3. ICSSG0 MII 引脚配置和 CPSW RGMII1 引脚配置共用四个引脚。
  - a. 这些引脚用于冲突检测，CRS 用于两个 MII 以太网端口。
  - b. 禁用 ICSSG0 MII 引脚配置上的两个信号。

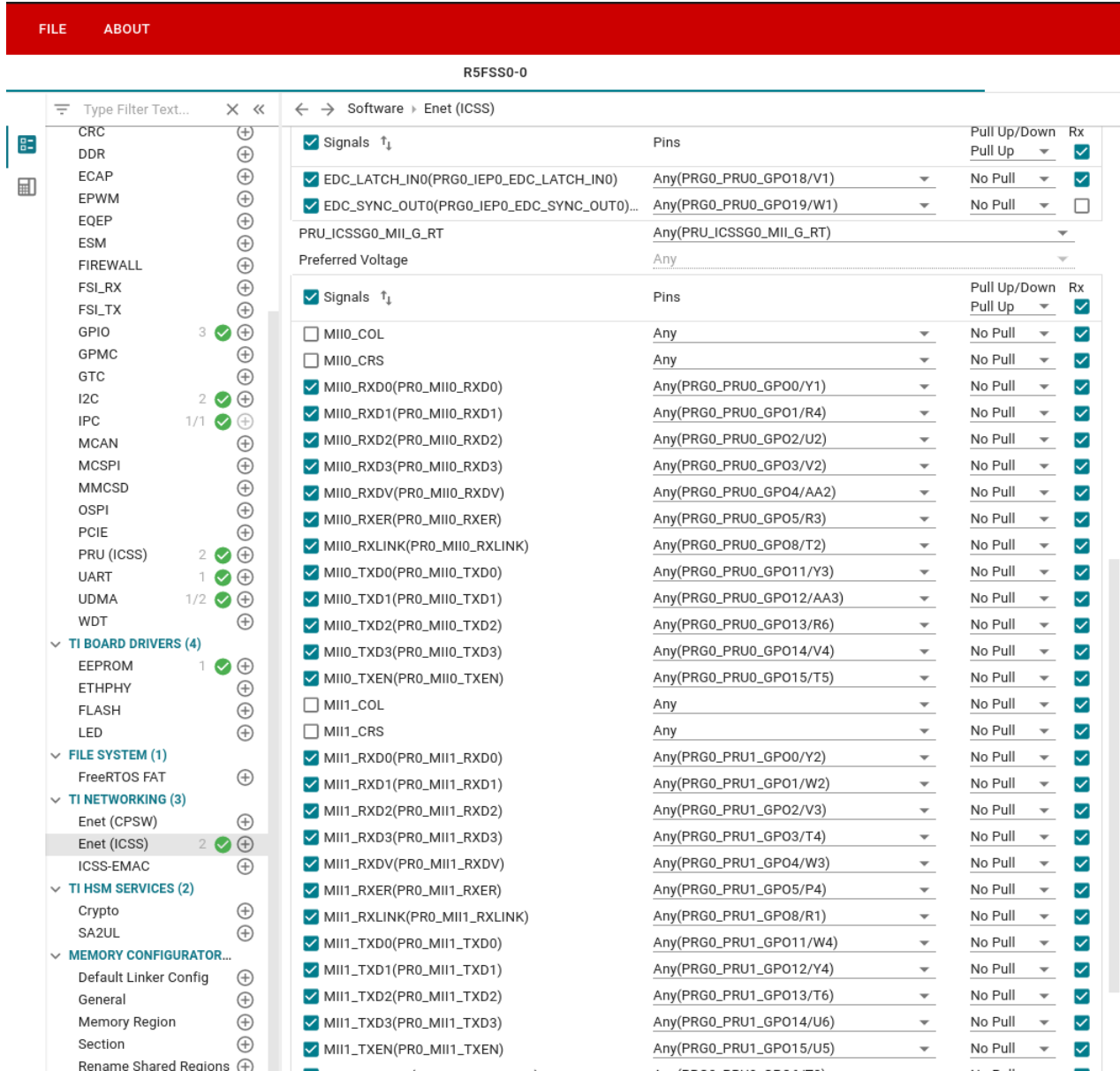


图 5-1. SysConfig : ICSSG0 和 ICSSG1 : MII 模式 : 对 R5FSS0-0 的分配

4. 在 syscfg-GUI 中声明 PRUSS 模块的 INTC 配置，如下所示：
  - a. → 引脚 41 : MII[0] → 通道 7

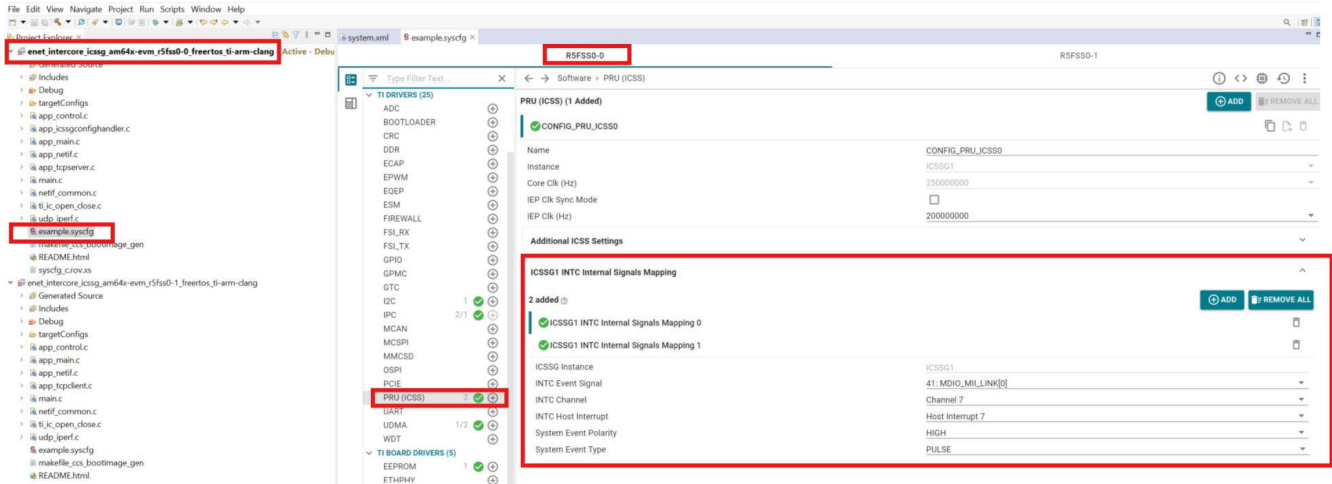


图 5-2. SysConfig : INTC 配置 : PRU 引脚 41

- b. → 引脚 53 : MII[1] → 通道 7

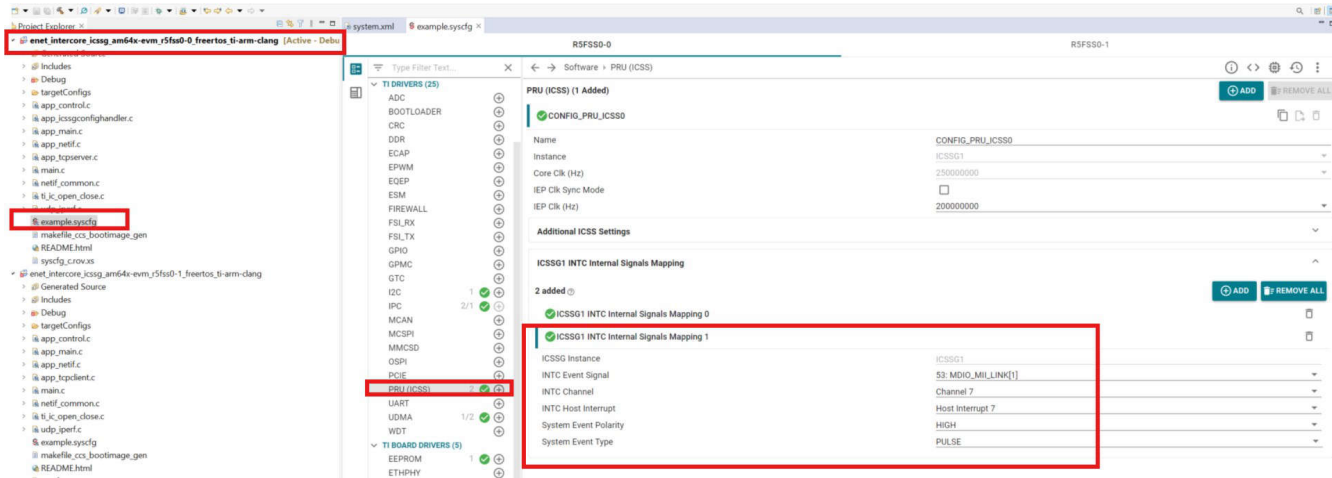


图 5-3. SysConfig : INTC 配置 : PRU 引脚 53

5. 确保在所有模块中打开 LwIP。
6. SEM 板仅支持 MII PHY (DP83826e)。
  - a. 因此，仅支持使用 100M 进行测试。



7. DP83826e PHY 需要在 syscfg-GUI 中声明复位引脚，配置如下：
- a. → AM64x : CONFIG\_GPIO\_31 → 输出 → GPIO0 → 引脚 R17 → 上拉

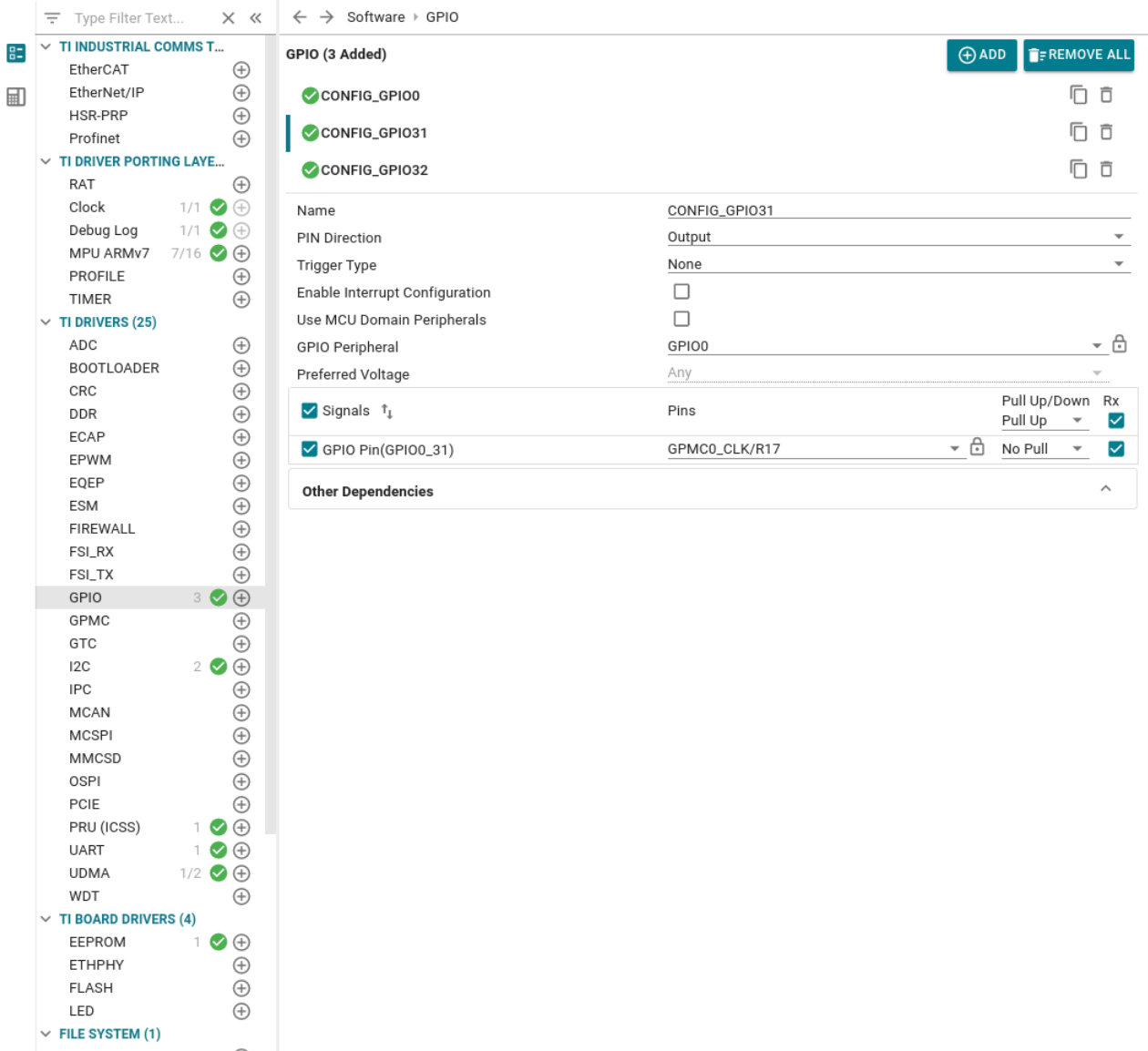
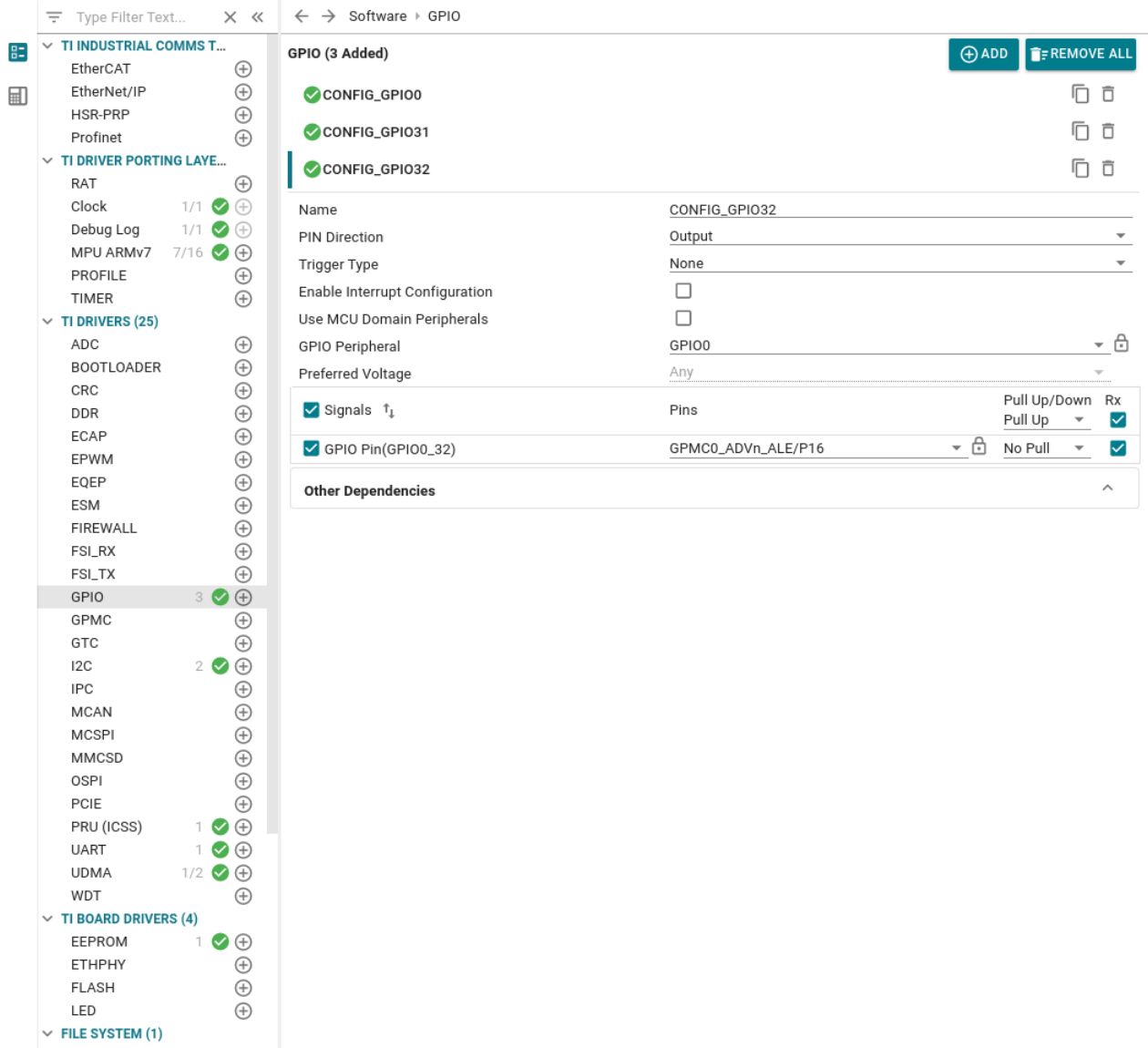


图 5-4. SysConfig : GPIO 引脚 31 : DP83826e PHY 复位引脚

b. CONFIG\_GPIO\_32 → 输出 → GPIO0 → 引脚 P16 → 上拉



TI INDUSTRIAL COMMS T...  
EtherCAT (+)  
EtherNet/IP (+)  
HSR-PRP (+)  
Profinet (+)

TI DRIVER PORTING LAYE...  
RAT (+)  
Clock 1/1 (+)  
Debug Log 1/1 (+)  
MPU ARMv7 7/16 (+)  
PROFILE (+)  
TIMER (+)

TI DRIVERS (25)  
ADC (+)  
BOOTLOADER (+)  
CRC (+)  
DDR (+)  
ECAP (+)  
EPWM (+)  
EQEP (+)  
ESM (+)  
FIREWALL (+)  
FSL\_RX (+)  
FSL\_TX (+)  
GPIO 3 (+)  
GPMC (+)  
GTC (+)  
I2C 2 (+)  
IPC (+)  
MCAN (+)  
MCSPi (+)  
MMCSd (+)  
OSPI (+)  
PCIE (+)  
PRU (ICSS) 1 (+)  
UART 1 (+)  
UDMA 1/2 (+)  
WDT (+)

TI BOARD DRIVERS (4)  
EEPROM 1 (+)  
ETHPHY (+)  
FLASH (+)  
LED (+)

FILE SYSTEM (1)  
----- (+)

GPIO (3 Added) [ADD] [REMOVE ALL]

- CONFIG\_GPIO00
- CONFIG\_GPIO31
- CONFIG\_GPIO32

Name: CONFIG\_GPIO32

PIN Direction: Output

Trigger Type: None

Enable Interrupt Configuration:

Use MCU Domain Peripherals:

GPIO Peripheral: GPIO0

Preferred Voltage: Any

Signals	Pins	Pull Up/Down	Rx
<input checked="" type="checkbox"/> GPIO Pin(GPIO0_32)	GPMC0_ADVn_ALE/P16	Pull Up	<input checked="" type="checkbox"/>

Other Dependencies

图 5-5. SysConfig : GPIO 引脚 32 : DP83826e PHY 复位引脚

8. 仅为活动内核启用 IPC。
  - a. 切勿为不同的内核打开不匹配的 IPC 配置。
  - b. 确保所有内核的 IPC 配置方式相同。

## 6 参考日志

主内核：

```

=====
ICSSG LWIP TCP ECHO SERVER
=====
EXT PHY Reset
Enabling clocks!
Enabling clocks!
Enabling clocks!
Enabling clocks!

Init configs EnetType:1, InstId :2
-----
PHY 3 is alive

Init configs EnetType:1, InstId :3
-----
EnetPhy_bindDriver: PHY 3: OUI:080028 Model:0f Ver:01 <-> 'dp83869' : OK
PHY 3 is alive
PHY 15 is alive

Init configs EnetType:1, InstId :0
-----
EnetPhy_bindDriver: PHY 15: OUI:080028 Model:0f Ver:01 <-> 'dp83869' : OK
PHY 1 is alive
PHY 3 is alive

Init configs EnetType:1, InstId :1
-----
EnetPhy_bindDriver: PHY 3: OUI:080028 Model:11 Ver:01 <-> 'generic' : OK
PHY 1 is alive
PHY 3 is alive
Starting lwIP, local interface IP is dhcp-enabled
[LWIPIF_LWIP] NETIF INIT SUCCESS
Host MAC address-0 : 34:08:e1:80:a9:36
[LWIPIF_LWIP] Enet has been started successfully
[0]Enet IF UP Event. Local interface IP:0.0.0.0
[LWIPIF_LWIP] NETIF INIT SUCCESS
Host MAC address-1 : 70:ff:76:1e:9c:4e
[1]Enet IF UP Event. Local interface IP:0.0.0.0
[LWIPIF_LWIP] NETIF INIT SUCCESS
Host MAC address-2 : 70:ff:76:1e:9c:4f
[2]Enet IF UP Event. Local interface IP:0.0.0.0
[LWIPIF_LWIP] NETIF INIT SUCCESS
Host MAC address-3 : 70:ff:76:1e:9c:50
[3]Enet IF UP Event. Local interface IP:0.0.0.0
[0]waiting for network UP ...
EnetPhy_bindDriver: PHY 1: OUI:080028 Model:11 Ver:01 <-> 'generic' : OK
Icssg_handleLinkUp: icssg0-1: Port 1: Link up: 100-Mbps Full-Duplex
[2]Network Link UP Event
[1]waiting for network UP ...
Icssg_handleLinkUp: icssg1-1: Port 1: Link up: 1-Gbps Full-Duplex
[0]Network Link UP Event
[2]waiting for network UP ...
[3]waiting for network UP ...
[0]waiting for network UP ...
[1]waiting for network UP ...
[2]Enet IF UP Event. Local interface IP:10.24.68.135
[0]Enet IF UP Event. Local interface IP:10.24.68.74
[3]waiting for network UP ...
[1]waiting for network UP ...
[3]waiting for network UP ...
[1]waiting for network UP ...
[3]waiting for network UP ...
Network is UP ...
[IPC RPMSG ECHO] Main core start !!!
[IPC RPMSG ECHO] Message exchange started by main core !!!
[IPC RPMSG ECHO] All echoed messages received by main core from 1 remote cores !!!
[IPC RPMSG ECHO] Messages sent to each core = 100
[IPC RPMSG ECHO] Number of remote cores = 1
[IPC RPMSG ECHO] Total execution time = 3332 usecs
[IPC RPMSG ECHO] One way message latency = 16660 nsec
All tests have passed!!
[IPC txn]Remote Server interface IP:10.24.68.74
30.59s : CPU load = 8.08 %
  
```

```
35. 60s : CPU load = 8.41 %
40. 61s : CPU load = 8.30 %
accepted new connection 810C4280
[IPC_ECHO_BACK] Sent Packet number 0
[IPC_ECHO_BACK] Sent Packet number 0
[IPC_ECHO_BACK] Sent Packet number 0
45. 62s : CPU load = 8.54 %
[IPC_ECHO_BACK] Sent Packet number 0
[IPC_ECHO_BACK] Sent Packet number 0
accepted new connection 810C4280
[IPC_ECHO_BACK] Sent Packet number 0
[IPC_ECHO_BACK] Sent Packet number 0
[IPC_ECHO_BACK] Sent Packet number 0
[IPC_ECHO_BACK] Sent Packet number 0
[IPC_ECHO_BACK] Sent Packet number 0
50. 63s : CPU load = 8.54 %
55. 64s : CPU load = 8.40 %
60. 65s : CPU load = 8.40 %
65. 66s : CPU load = 8.31 %
70. 67s : CPU load = 8.39 %
```

远程内核：

```
Remote Core:
[IPC RMSG ECHO] Remote core start !!!
[IPC RMSG ECHO] Remote Core Sending sync messages to main core ... !!!
[IPC RMSG ECHO] Remote Core waiting for messages from main core ... !!!
Closing Remote Core!!
[IPC txn]Remote Server interface IP:10.24.68.74
=====
  CPSW LWIP TCP ECHO SERVER
=====
Enabling clocks!
EnetApputils_reduceCoreMacAllocation: Reduced Mac Address Allocation for CoreId:2 From 4 To 1
Mdio_open: MDIO Manual_Mode enabled

EnetPhy_bindDriver: PHY 0: OUI:080028 Model:23 Ver:01 <-> 'dp83867' : OK

PHY 0 is alive
Starting lwIP, local interface IP is dhcp-enabled
[LWIPIF_LWIP] NETIF INIT SUCCESS
Host MAC address-0 : 70:ff:76:1e:9c:51

[0]Enet IF UP Event. Local interface IP:0.0.0.0
[LWIPIF_LWIP] Enet has been started successfully
[0]waiting for network UP ...
[0]waiting for network UP ...
Cpsw_handleLinkUp: Port 1: Link up: 1-Gbps Full-Duplex

MAC Port 1: link up
[0]Network Link UP Event
[0]waiting for network UP ...
[0]waiting for network UP ...
[0]waiting for network UP ...
[0]waiting for network UP ...
[0]Enet IF UP Event. Local interface IP:10.24.69.120
Network is UP ...
  IP eneterd is: 10.24.68.74
<<<< ITERATION 1 >>>>
  Connecting to: :8888
  Connection with the server is established
  "Hello over TCP 1" was sent to the Server
[IPC_ECHO_BACK]"Hello over TCP 1" was received from the Server: Packet num 1
  Successfully received the packet 1
  "Hello over TCP 2" was sent to the Server
[IPC_ECHO_BACK]"Hello over TCP 2" was received from the Server: Packet num 2
  Successfully received the packet 2
  "Hello over TCP 3" was sent to the Server
[IPC_ECHO_BACK]"Hello over TCP 3" was received from the Server: Packet num 3
  Successfully received the packet 3
  "Hello over TCP 4" was sent to the Server
[IPC_ECHO_BACK]"Hello over TCP 4" was received from the Server: Packet num 4
  Successfully received the packet 4
  "Hello over TCP 5" was sent to the Server
[IPC_ECHO_BACK]"Hello over TCP 5" was received from the Server: Packet num 5
  Successfully received the packet 5
  Connection closed
<<<< ITERATION 2 >>>>
  Connecting to: :8888
  Connection with the server is established
  "Hello over TCP 1" was sent to the Server
[IPC_ECHO_BACK]"Hello over TCP 1" was received from the Server: Packet num 1
  Successfully received the packet 1
  "Hello over TCP 2" was sent to the Server
[IPC_ECHO_BACK]"Hello over TCP 2" was received from the Server: Packet num 2
  Successfully received the packet 2
  "Hello over TCP 3" was sent to the Server
[IPC_ECHO_BACK]"Hello over TCP 3" was received from the Server: Packet num 3
  Successfully received the packet 3
  "Hello over TCP 4" was sent to the Server
[IPC_ECHO_BACK]"Hello over TCP 4" was received from the Server: Packet num 4
  Successfully received the packet 4
  "Hello over TCP 5" was sent to the Server
[IPC_ECHO_BACK]"Hello over TCP 5" was received from the Server: Packet num 5
  Successfully received the packet 5
  Connection closed
```

## 7 ICSSG0 和 ICSSG1 功能测试

1. 使用 MDIO 扫描找到附加电路板的 PHY 地址，或检查 MDIO-PHY 接口中的相应寄存器。
2. 将 SysConfig-GUI 的“Board Config”部分中 ICSSG0 实例的 PHY 地址更改为特定电路板上的 PHY 地址位置。
3. 在测试文件夹中构建以下示例：  

```
make -s -C test/networking/lwip/enet_icssg_tcpserver/am64x-evm/  
r5fss0-0_freertos/ti-arm-clang clean all
```

## 8 ICSSG 和 CPSW

ICSSG 和 CPSW 可以在不同的内核上一起运行。当前示例使用内核 R5Fss\_0-0 上的 ICSSG 作为主内核，使用 CPSW 作为远程内核。该示例使用独立的 LwIP 协议栈，占用的内存比典型情况下更多。可将其改为 CPSW 运行第 2 层测试用例，ICSSG 作为远程控制器且只运行一个 LwIP 协议栈，ICSSG 和 CPSW 之间的接口层以 IPC 作为传输介质。

```
make -s -C test/networking/lwip/enet_icssg_tcpserver/am64x-evm/system_cpsw_icssg clean all
```

这会在相应的测试文件夹中生成两个 .out 文件。

1. 通过 CCS 加载该文件，先启动 ICSSG 内核。
2. 启动 CPSW 内核。
  - a. 这样便可看到 ICSSG 在 UART 控制台上的四个 IP，
  - b. 以及 CCS 控制台上的一个 IP。
3. 所有端口开箱即处于 MAC 模式。

该示例使用 CPSW 进行了测试 (CPSW 是在 R5Fss\_0-1 上独立运行的默认 LwIP 示例)，并使用 iPerf 软件进行了测试。现在，已将 CPSW TCP 服务器示例移植到测试文件夹中，并添加了 IPC 远程内核功能。这不能与默认 IPC 示例测试并行使用，因为它在 ti\_drivers\_config.c 文件中使用不同的 Vring 配置。确保为示例组合分配正确数量的内核以进行测试，并根据 RPmessage 数量分配正确数量的 gcount。

将其加载到 R5Fss\_0-1 内核并测试其功能。

## 9 总结

本文档详细介绍了如何使用 [MCU+SDK](#) 在 AM64x/AM243x 上为用户实现五个以太网端口。用户可以使用 AM64x/AM243x SoC 上提供的两个 ICSSG0 以太网端口、两个 ICSSG1 以太网端口和一个 CPSW3G 以太网端口。

## 10 参考资料

- 德州仪器 (TI), [以太网 PRU\\_ICSSG 实例 0 \(PRU\\_ICSSG0\) 使用指南](#), 网页。
- [AM243x-Academy](#)
- [AM64x-Academy](#)
- [MCU-PLUS-SDK-AM64x-Ethernet-Networking-Documentation](#)
- [MCU-PLUS-SDK-AM64x-Ethernet-Networking-Examples](#)
- [MCU-PLUS-SDK-AM243x-Ethernet-Networking-Documentation](#)
- [MCU-PLUS-SDK-AM243x-Ethernet-Networking-Examples](#)
- [MCU-PLUS-SDK-Git](#)

## 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  
Copyright © 2024，德州仪器 (TI) 公司