文档编号：REALTIMEEDGEUG
第 2.0 版，2021 年 7 月 29 日

实时边缘软件用户指南

目录

第 1 章
简介

1.1 实时边缘软件

实时边缘软件是开放工业 Linux (OpenIL)的演进版本，适用于不同领域的实时和确定性系统。关键技术组件包括实时网络服务、实时系统和工业。

- 实时网络服务包括 TSN 技术、TSN 标准、管理、配置和应用程序。还支持网络服务和冗余功能。
- 实时系统包括 PREEMPT_RT Linux、Jailhouse 和基于 U-Boot 的裸机框架以及这些系统之间的不同组合。
- “协议”组件包括对 EtherCAT、CoE、OPC-UA 等行业标准协议的支持。

本文档介绍了恩智浦硬件平台上实时边缘软件的功能和实现。

1.2 实时边缘软件 Yocto Project

关于使用 Yocto 构建环境，请参见《实时边缘 Yocto Project 用户指南》。

《实时边缘 Yocto Project 用户指南》介绍了如何使用 Yocto Project 构建环境为 i.MX 和 QorIQ 板构建实时边缘镜像。

1.3 支持的恩智浦平台

下表列出了支持实时边缘软件的恩智浦硬件 SoC 和电路板。

表 1.支持的恩智浦平台

平台	架构	启动
i.MX 6ULL EVK	Arm v7	SD
i.MX 8M Mini EVK	Arm v8	SD
i.MX 8M Plus EVK	Arm v8	SD
LS1028ARDB	Arm v8	SD
LS1021AIOT	Arm v7	SD
LS1021ATSN	Arm v7	SD
LS1021ATWR	Arm v7	SD
LS1012ARDB	Arm v8	QSPI
LS1043ARDB	Arm v8	SD
LS1046ARDB	Arm v8	SD
LS1046AFRWY	Arm v8	SD
LX2160ARDB	Arm v8	SD
LX2160A Rev2	Arm v8	SD

1.4 相关文档

要使用 Yocto 构建环境，请参见 URL http://nww.preview.nxp.com/ design/software/development-software/real-time-edge-software:REALTIME-EDGE-SOFTWARE 上的《实时边缘 Yocto Project 用户指南》。

有关启动和设置相关板的详细说明，请参阅以下指南。

- i.MX 6ULL EVK 快速入门指南
- i.MX 8M Mini EVK 快速入门指南
- i.MX 8M Plus EVK 快速入门指南
- LS1028ARDB 快速入门指南
- LS1021AIoT 入门指南
- LS1021ATSN 入门指南
- LS1021ATWR 入门指南
- LS1012ARDB 入门指南
- LS1043ARDB 入门指南
- LS1046ARDB 入门指南
- LS1046AFRWY 入门指南
- LX2160A/LX2160A-Rev2 RDB 快速入门指南

1.5 缩略词与缩写

下表列出了本文档中使用的首字母缩略词。

表 2.缩略词与缩写

术语	说明
AVB	音视频桥接
BC	边界时钟
BLE	低功耗蓝牙
BMC	最佳 master 时钟
CA	客户端应用程序
CAN	控制器局域网
CBS	基于信用的整形器
DEI	丢弃资格指示
DP	显示端口
EtherCAT	用于控制自动化技术的以太网
ECU	电子控制单元
FDB	转发数据库
FQTSS	时间敏感型流的转发和排队增强
FMan	帧管理器
GPU	通用处理器单元
ICMP	互联网控制消息协议
IEEE	电气与电子工程师协会

表格接下页……

表 2.缩略词与缩写（续）

术语	说明
IETF	互联网工程任务组
IPC	处理器间通信
KM	密钥管理
LBT	延迟和带宽测试仪
MAC	介质访问控制
NFC	近距离通信
NCI	NFC 控制器接口
NMT	网络管理
OC	普通时钟
OpenIL	开放工业 Linux
OPC	开放平台通信
OP-TEE	开放的可移植可信执行环境
OS	操作系统
OTA	无线
OTPMK	一次性可编程主密钥
PCP	优先代码点
PDO	过程数据对象
PHC	PTP 硬件时钟
PIT	数据包到达间隔时间
PLC	可编程逻辑控制器
PTP	精确时间协议
QSPI	排队的串行外设接口
RCW	复位配置字
REE	富执行环境
RPC	远端程序调用
RTEdge	实时边缘
RTC	实时时钟
RTT	往返时间
SABRE	快速设计智能应用蓝图
SDO	服务数据对象
SPI	串行外设接口
SRP	流预留协议

表格接下页……

表 2.缩略词与缩写（续）

术语	说明
SRK	单根密钥
TA	受信任的应用程序
TAS	时间感知调度程序
TC	流量分类
TCP	传输控制协议
TEE	可信的执行环境
TFTP	简单文件传输协议
TSN	时间敏感网络服务
TZASC	可信区地址空间控制器
UDP	用户数据报协议
VLAN	虚拟局域网

第 2 章
发布说明

2.1 新功能

以下各节介绍了每个版本的新功能。

2.1.1 实时边缘软件 v2.0 的新功能

- **基于 Yocto project 3.2 (Gatesgarth)**
- **实时系统**
  - PREEMPT-RT Linux
  - 异构架构
    - 裸机：A 内核上的 PREEMPT-RT Linux + A 内核上的裸机架构
      - i.MX 8M Plus EVK、i.MX 8M Mini EVK、LS1028ARDB、LS1046ARDB、LS1043ARDB、LS1021A-IoT
    - Jailhouse：A 内核上的 PREEMPT-RT Linux + Jailhouse + A 内核上的 PREEMPT-RT Linux
      - i.MX 8M Plus EVK、LS1028ARDB、LS1046ARDB
- **实时网络服务**
  - TSN
    - TSN 标准
      - IEEE 802.1Qav
      - IEEE 802.1Qbv
      - IEEE 802.1Qbu
      - IEEE 802.1Qci
      - IEEE 802.1CB
      - IEEE 802.1AS-2020 (gPTP)
      - IEEE 802.1Qat-2010 (SRP)
    - TSN 配置
      - Linux tc 命令和 tsntool
      - NETCONF/YANG
      - 动态 TSN 配置——基于网络的 TSN 配置、动态拓扑发现
    - TSN 应用程序
      - 实时流量处理示例
  - 网络服务
    - 802.1 Q-in-Q
    - VCAP tc flower 链模式
      - 优先级设置、VLAN 标签推送/弹出/修改、监管器突发和速率配置、丢弃/捕捉/重定向
- **工业**
  - EtherCAT master

    - IGH EtherCAT master 协议栈
    - 原生支持 EtherCAT 的网络驱动模块(i.MX 8M Mini EVK)
  - FlexCAN
    - Linux 内核上的 SocketCAN
  - CANOpen
    - CANOpen master 和 slave 示例代码
  - CoE：CANOpen over EtherCAT
    - 基于 IGH CoE 接口的 CiA402(DS402)配置文件框架
    - EtherCAT CoE 6-8 轴控制(i.MX 8M Mini EVK)
  - OPC-UA/OPC-UA pub/sub
    - open62541
  - Modbus
    - Modbus master 和 slave
    - Modbus-RTU
    - Modbus-TCP
    - Modbus-ASCII
- **新增平台**
  - i.MX 6ULL EVK

2.1.2 OpenIL v1.11 中的新功能

新功能：

- **TSN**
  - 802.1AS-2020
    - 在 i.MX 8M Plus 和 LS1028A 上对多域的初始支持
- **硬件**
  - i.MX 8M Plus 芯片 A1
- **Linux 内核**
  - 适用于 i.MX 8 系列的 LTS 5.4.70
- **U-Boot**
  - 适用于 i.MX 8 系列的 v2020.04
- **裸机**
  - 适用于 Layerscape 和 i.MX 8 系列的 v2020.04
  - i.MX 8M Plus EVK

2.1.3 OpenIL v1.10 的新功能

新功能：

- **TSN**
  - VCAP 链模式
  - GenAVB/TSN 协议栈

- **实时**
  - i.MX 8M Mini 上的 PREEMPT-RT 5.4
    - 以太网
    - PCIe
    - GPIO
    - DSI
- **裸机**
  - i.MX 8M Mini EVK（A 内核至 A 内核）
    - ICC
    - 以太网
    - GPIO
- **OpenIL 框架**
  - 板
    - i.MX 8M Mini 平台
      - GPU：OpenGL ES
      - 显示：OpenGL ES、Weston、DSI-MIPI、CSI-MIPI

2.1.4 OpenIL v1.9 中的新功能

新功能：

- **TSN**
  - 对 Qbu 和 Qci 的 tc flower 支持
  - 802.1 QinQ
  - 多端口 TSN 交换机解决方案
  - i.MX 8M Plus——TSN
- **实时**
  - i.MX 8M Plus 上的 PREEMPT-RT 5.4
- **裸机**
  - LX2160ARDB rev2 支持和 ICC
- **OpenIL 框架**
  - linuxptp 升级到 3.0
  - 板
    - i.MX 8M Plus EVK
      - TSN：Qbv、Qbu、Qav
      - GPU：OpenGL ES、OpenCL
      - 显示：OpenGL ES、Weston
    - LS1028ARDB
      - 显示：OpenGL ES、Weston
      - GPU：OpenGL ES、OpenCL
    - LX2160ARDB rev2

2.1.5 OpenIL v1.8 中的新功能

新功能：

- TSN
  - 对 VLAN 重新标记的 tc VCAP 支持
  - 对监管的 tc VCAP 支持
  - 对 Qav 和 Qbv 的 tc 支持
  - SJA1105 DSA 支持和时钟同步
  - 用于网络配置（IP、MAC 和 VLAN）的 YANG 模块
- 实时
  - PREEMPT-RT 5.4
- 裸机
  - LX2160A rev1 ICC
- OpenIL 框架
  - buildroot 升级到 2020.02
  - 内核/U-Boot
    - Linux 升级到 LSDK20.04——Linux-5.4.3
    - U-Boot 升级到 LSDK20.04——U-Boot 2019.10
  - 板
    - i.MX 8M Mini
    - LS1028ATSN 板与 SJA1105

2.1.6 OpenIL v1.7 中的新功能

新功能：

- TSN
  - 基于 BC 的 802.1AS 桥接模式
  - 基于 sysrepo 的 Netopper2 支持。支持 Qbv、Qbu、Qci 配置
  - 基于 VLAN 的 tc flower 监管器
  - 基于网络的 TSN 配置工具——可用于 Qbv、Qbu 和 Qci 配置
- 实时
  - Xenomai
    - Xenomai I-pipe 升级到 4.19
  - 裸机
    - LS1028 上的 SAI 支持
    - i.MX6Q 裸机 ICC
- 工业协议
  - CANopen over EtherCAT
- OpenIL 框架

- 内核/U-Boot
  - Linux 升级到 LSDK1909 - 4.19
  - U-Boot 升级到 U-Boot-2019.04
- 电路板
  - LX2160ARDB SD 启动
  - LX2160ARDB XSPI 启动
  - LS1028ARDB XSPI 启动
  - LS1046ARDB eMMC 启动

2.1.7 OpenIL v1.6 中的新功能

新功能：

- TSN
  - 基于网络的 TSN 配置工具——可用于 Qbv 和 Qbu 配置
  - TSN 驱动程序增强
- 实时
  - 裸机
    - i.MX6Q-sabresd 裸机支持
- NETCONF/YANG
  - Qbu 和 Qci 协议的 NETCONF/YANG 模型
- 工业协议
  - LS1028A——BEE click 板

2.1.8 OpenIL v1.5 中的新功能

新功能：

- TSN
  - 基于网络的 TSN 配置工具——可用于 Qbv 和 Qbu 配置
  - LS1028A TSN 交换机的 802.1AS 端点模式
- 实时
  - Xenomai
    - LS1028 ENETC Xenomai RTNET 支持
  - 裸机
    - LS1028 裸机 ENETC 支持
- NETCONF/YANG
  - Qbv 协议的 NETCONF/YANG 模型
- 工业协议
  - LS1028A——BLE click 板

2.1.9 OpenIL v1.4 中的新功能

新功能：

- TSN
  - ENETC TSN 驱动程序：Qbv、Qbu、Qci、Qav
  - ENETC 1588 两步时间戳支持
  - 开关 TSN 驱动程序：Qbv、Qci、Qbu、Qav、802.1CB 支持
- 实时
  - Xenomai
    - LS1028ARDB
  - 裸机
    - LS1021AIoT、LS1043ARDB、LS1046ARDB
    - LS1028 裸机基本裸机支持
- 工业协议
- — LS1028A——NFC click 板
  - QT5.11
- OpenIL 框架
  - 板：LS1028ARDB

2.2 功能支持表

下表显示了此版本中支持的功能。

表 3.主要功能

功能			i.MX 6ULL 14x14 EVK	i.MX 8M Mini EVK	i.MX 8M Plus EVK	LS1028A RDB	LS1021A TSN	LS1021A IOT	LS1021A TWR	LS1012A RDB	LS1043A RDB	LS1046A RDB	LS1046A FRWY	LX2160A RDB
启动模式	SD		是	是	是	是	是	是	是		是	是	是	是
	QSPI									是				
实时系统	Preempt-RT Linux		是	是	是	是	是	是	是	是	是	是	是	是
	裸机	ICC		是	是	是		是			是	是		是
		PCIe				是		是			是	是		
		以太网		是	是	是					是	是		
		GPIO		是	是			是						
		IPI		是	是	是		是			是	是		是
		UART		是	是	是		是			是	是		是
		USB						是			是	是		
		SAI				是								
		CAN						是						
		I2C				是		是			是	是		

表格接下页……

表 3.主要功能（续）

功能				i.MX 6ULL 14x14 EVK	i.MX 8M Mini EVK	i.MX 8M Plus EVK	LS1028A RDB	LS1021A TSN	LS1021A IOT	LS1021A TWR	LS1012A RDB	LS1043A RDB	LS1046A RDB	LS1046A FRWY	LX2160A RDB
		QSPI											是		
		IFC											是		
		Linux（与裸机的通信）	ICC		是	是	是		是			是	是		是
			IPI		是	是	是		是			是	是		是
	Jailhouse					是	是					是	是		
实时网络	TSN 标准	Qbv				是	是								
		Qbu				是	是								
		QCI				不适用	是								
		Qav				是	是								
		802.1AS			是	是	是	是	是	是		是	是	是	是
		802.1CB				不适用	是								
		VCAP 链模式				不适用	是								
		802.1 Q-in-Q					是								
	TSN 配置	Linux tc 命令				是	是								
		TSN 工具				是	是								
		NETCONF/ YANG	Qbv			是	是								
			Qbu			是	是								
			QCI			不适用	是								
			IP			是	是								
			MAC			是	是								
			VLAN 配置			是	是								
		基于网络的配置	Qbv			是	是								
			Qbu			是	是								
			QCI			不适用	是								
		动态拓扑发现				是	是								
	IEEE 1588/802.1AS				是	是	是	是	是	是		是	是	是	是

表格接下页……

表 3.主要功能（续）

功能			i.MX 6ULL 14x14 EVK	i.MX 8M Mini EVK	i.MX 8M Plus EVK	LS1028A RDB	LS1021A TSN	LS1021A IOT	LS1021A TWR	LS1012A RDB	LS1043A RDB	LS1046A RDB	LS1046A FRWY	LX2160A RDB
工业协议	EtherCAT master	IGH EtherCAT master 协议栈	是	是	是	是	是	是	是	是	是	是	是	是
		原生支持 EtherCAT 的网络驱动程序模块		是										
	FlexCAN					是		是						
	CANopen							是						
	OPC-UA	open62541	是	是	是	是	是	是	是	是	是	是	是	是
	BEE（Mikro click 板）					是								
	BLE（Mikro click 板）					是								
	NFC（Mikro click 板）					是								
	Modbus	Modbus slave 和 master	是	是	是	是	是	是	是	是	是	是	是	是
		Modbus-RTU	是	是	是	是	是	是	是	是	是	是	是	是
		Modbus-TCP	是	是	是	是	是	是	是	是	是	是	是	是
		Modbus-ASCII	是	是	是	是	是	是	是	是	是	是	是	是

2.3 未解决、已修复和已关闭的问题

本节包含三个表：未解决、已修复和已关闭的问题。

- 未解决的问题目前没有解决方案。会尽可能提供解决方法建议。
- 已修复的问题已有集成到“*已修复*”版本中的软件解决方法。
- 已关闭的问题是根本原因和修复方法超出实时边缘软件范围的问题。处置是提供解释。

表 4.实时边缘软件 v2.0 中未解决的问题

ID	说明	未解决版本	解决方法
INDLINUX-2184	ICC：执行 icc 基准测试时，如果使用太多存储器，U-Boot 裸机应用程序可能会阻塞	实时边缘软件 v2.0	
INDLINUX-2207	PREEMPT_RT： 不支持 Futex 延迟基准测试	实时边缘软件 v2.0	

表格接下页……

表 4.实时边缘软件 v2.0 中未解决的问题（续）

ID	说明	未解决版本	解决方法
INDLINUX-2208	ptp4l：在 LS1028A 上 Qbv 门控关闭超过约 5s 后，tx 时间戳轮询超时	实时边缘软件 v2.0	使传输 1588 个数据包的队列保持打开
INDLINUX-2179	Qbv 由于 PHC 时间大调整而停止	实时边缘软件 v2.0	请在运行任何 TSN 协议之前进行时钟同步
INDLINUX-2209	ICC：i.MX 8M Plus 上的数据传输性能未优化	实时边缘软件 v2.0	

表 5.修复了实时边缘软件 v2.0 中的问题

ID	说明	未解决版本	已修复版本
此版本没有。			

表 6.实时边缘软件 v2.0 中已关闭的问题

ID	说明	未解决版本	处置
此版本没有。			

第 3 章
实时系统

实时边缘软件支持实时系统功能：Preempt-RT Linux、裸机和 Jailhouse。

3.1 Preempt-RT Linux

Preempt-RT Linux 选项将内核转换为实时内核。它通过用可抢占的优先级继承感知变体替换各种锁定原语（例如自旋锁、读写锁等）来实现这一点。Preempt-RT Linux 选项还强制执行中断线程，并引入了分解长的不可抢占部分的机制。这使得内核完全可抢占，并将大多数执行上下文置于调度程序控制之下。但是，非常低级和关键的代码路径（入口代码、调度程序、低级中断处理）仍然不可抢占。

3.1.1 系统实时延迟测试

实时 Linux 的基本测量工具是循环测试。

3.1.1.1 运行循环测试

循环测试提供有关系统延迟的统计信息。它准确且重复地测量线程的预期唤醒时间与实际唤醒的时间之间的差异。它可以测量由硬件、固件和操作系统引起的实时系统延迟。

最初的测试由 Thomas Gleixner(tglx)编写，但随后有几个人进行了修改。循环测试目前由 Clark Williams 和 John Kacur 维护，是测试套件 rt-tests 的一部分。

循环测试：

- 使用以下命令执行延迟测试：

```
$ cyclictest -p90 -h50 -D30m
```

> 注意
> 有关循环测试的详细参数，请参见循环测试网页。

3.1.2 实时应用程序开发

本节介绍开发实时应用程序的步骤。

实时应用程序：API、基本结构、背景：

- 基本的 Linux 应用程序规则相同；使用 POSIX API。
- 仍然有内核空间和用户空间的划分。
- Linux 应用程序在用户空间中运行。
- 更多详细信息，请参见：http://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO

实时应用程序：用户可以使用以下步骤构建实时应用程序：

- **使用交叉编译器示例：**

```
$ arm-linux-gnueabihf-gcc<filename>.c -o <filename>.out -lrt -Wall
```

- **在目标示例上使用本机编译器：**

```
$ gcc<filename>.c -o <filename>.out -lrt -Wall
```

调度策略有两类：

完全公平调度(CFS)

- SCHED_NORMAL
- SCHED_BATCH
- SCHED_IDLE

实时策略：

- SCHED_FIFO
- SCHED_RR
- SCHED_DEADLINE

3.2 裸机

3.2.1 概述

以下各节概述了实时边缘裸机框架，包括：

- 支持的功能
- 使用支持的平台开始使用裸机框架：
  - 恩智浦 Layerscape 平台
  - i.MX 8M 平台

它还介绍了如何在主机环境上运行示例裸机框架，以及基于裸机框架开发客户特定的应用程序。

3.2.1.1 裸机框架

裸机框架旨在支持需要低延迟、实时响应和高性能的方案。内核上没有运行操作系统，客户特定的应用程序直接在内核上运行。下图显示了裸机框架架构。

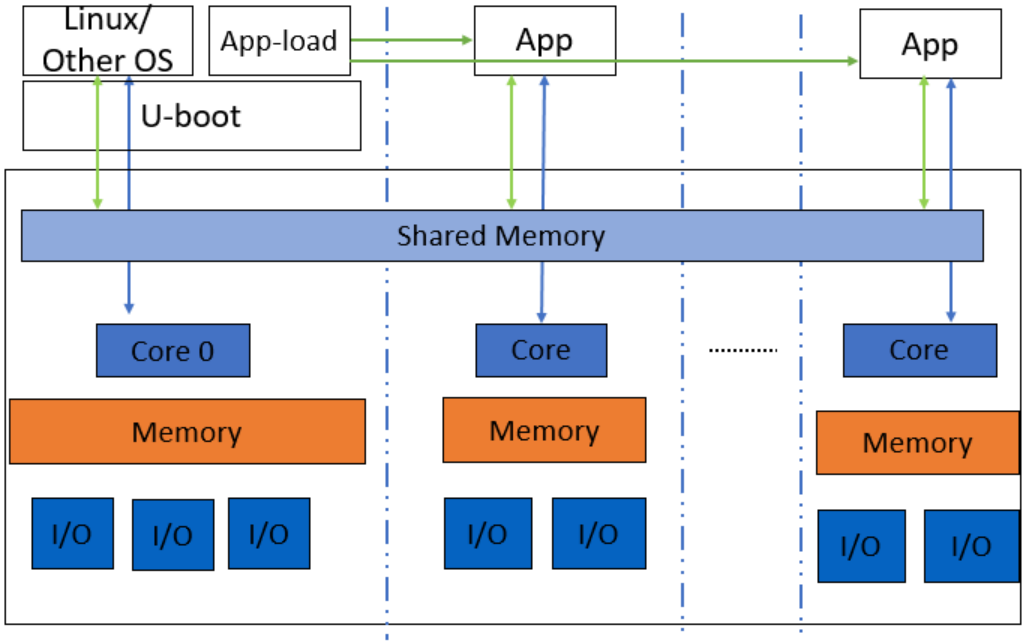


图 1.裸机框架架构

裸机框架的主要功能如下：

- 内核 0 作为 master 运行，它运行 Linux、Vxworks 等操作系统。
- slave 内核在裸机应用程序上运行。
- 将不同的 IP 块轻松分配给不同的内核。
- 不同内核之间中断和数据传输的高性能机制。
- 不同的 UART 用于内核 0 和 slave 内核，便于调试。
- 通过共享存储器进行通信。

master 内核 0 运行 U-Boot，然后将裸机应用程序加载到 slave 内核并启动裸机应用程序。下图显示了启动流程图：

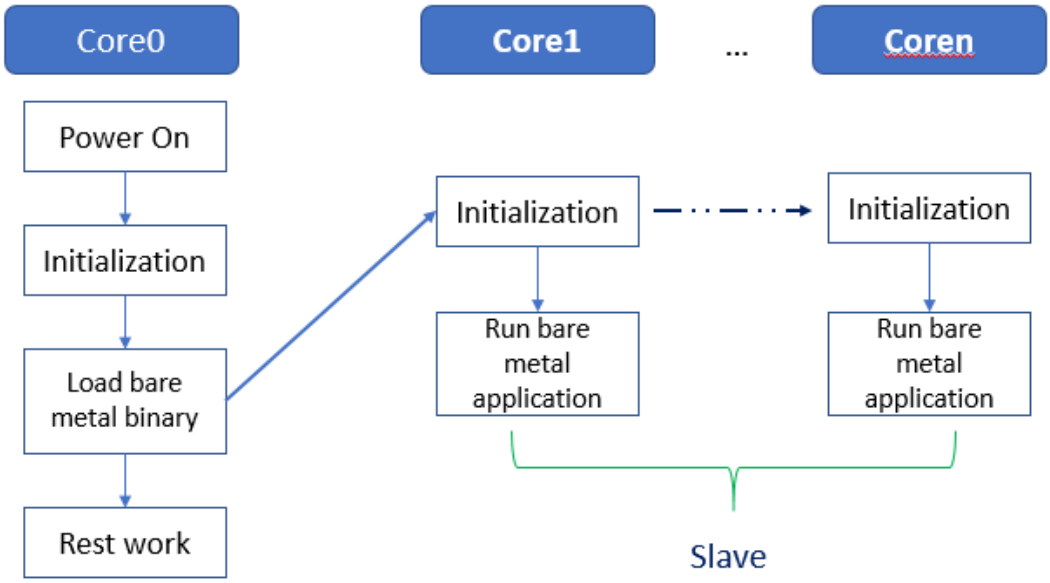


图 2.裸机框架启动流程图

3.2.1.2 支持的平台

下表列出了各种恩智浦处理器和板支持的工业物联网功能。

表 7.恩智浦处理器支持的工业物联网功能

处理器	板	支持的主要功能
i.MX 8M Mini	i.MX 8M Mini EVK	UART、IPI、数据传输、以太网、GPIO
i.MX 8M Plus	i.MX 8M Plus EVK	UART、IPI、数据传输、以太网、GPIO
LS1028A	LS1028ARDB	I2C、UART、ENETC、IPI、数据传输、SAI
LS1043A	LS1043ARDB	IRQ、IPI、数据传输、以太网、IFC、I2C、UART、FMan、USB、PCIe
LS1046A	LS1046ARDB	IRQ、IPI、数据传输、以太网、IFC、I2C、UART、FMan、QSPI、USB、PCIe
LS1021A	LS1021A-IoT	GPIO、IRQ、IPI、数据传输、IFC、I2C、UART、QSPI、USB、PCIe、FlexCAN
LX2160A/Rev2	LX2160ARDB	UART、IPI、数据传输

3.2.2 入门指南

本节介绍如何设置环境并在 slave 内核上运行裸机示例（假设内核 0 为 master 内核，其他核为 slave 内核）。

3.2.2.1 硬件和软件要求

运行裸机框架方案需要以下要素：

- 硬件：LS1021A-IoT、LS1043ARDB、LS1046ARDB、LS1028ARDB、i.MX 8M Mini EVK、i.MX 8M Plus EVK 或其他 Layerscape 板和串行线缆。
- 软件：OpenIL v1.4 或更高版本。

3.2.2.2 硬件设置

本节介绍恩智浦板运行裸机框架示例所需的硬件设置。

3.2.2.2.1 i.MX 8M Mini EVK 和 i.MX 8M Plus EVK 板

1.i.MX 8M Plus EVK

板上有一个 USB MicroB 调试端口。将 MicroB 线连接到 PC 时，可以找到四个 UART 端口。这些端口如下所示。

```
/dev/ttyUSB0
/dev/ttyUSB1
/dev/ttyUSB2
/dev/ttyUSB3
```

端口/dev/ttyUSB2 用于内核 0（master 内核），/dev/ttyUSB3 用于内核 1、内核 2 和内核 3（slave 内核）。

2.i.MX 8M Mini EVK

板上有 1 个 USB MicroB 调试端口，MicroB 线缆连接 PC 时可以找到 4 个 UART 端口。

```
/dev/ttyUSB0
/dev/ttyUSB1
```

/dev/ttyUSB1 用于内核 0（master 内核），/dev/ttyUSB0 用于内核 1、内核 2 和内核 3（slave 内核）。

3.GPIO 设置

对于 i.MX 8M Mini EVK 或 i.MX 8M Plus EVK gpio 测试，J1003 的引脚 7 和引脚 8 应直接连接。

3.2.2.2.2 LS1028ARDB、LX2160ARDB、LS1043ARDB 或 LS1046ARDB

如果 LS1028ARDB、LX2160ARDB、LS1043ARDB 或 LS1046ARDB 硬件板用于开发实时边缘裸机框架，则需要两根串行线缆。一个用于内核 0，连接到 UART1 端口，另一个用于 slave 内核，连接到 UART2 端口。

要在 LS1028ARDB 上支持 SAI 功能，请将开关 SW5_8 设置为“开启”。

3.2.2.2.3 LS1021A-IoT 板

需要两根串行线缆。一根用于内核 0，它连接到 UART0 的 USB0/K22 端口。另一根线缆用于 slave 内核，将 UART1 的 J8 和 J17 连接在一起。下表显示了 UART1 的引脚。

表 8.UART 引脚

引脚名称	功能
J8 引脚 7	接地
J17 引脚 1	Uart1_SIN
J17 引脚 2	Uart1_SOUT

下图显示了 UART1 硬件连接。

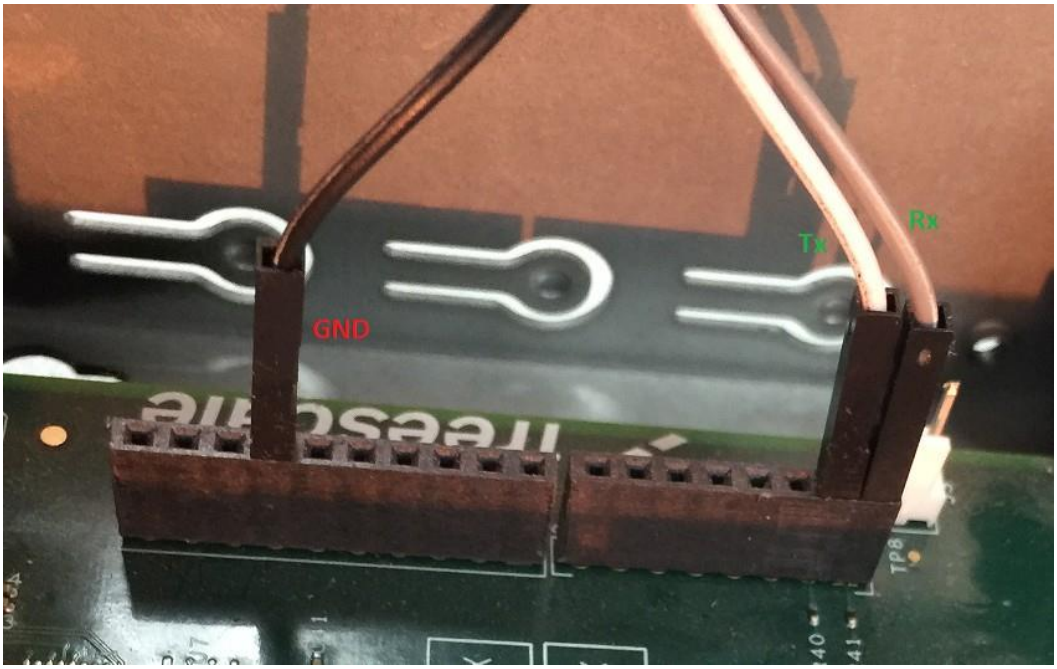


图 3.UART1 硬件连接

为了测试 GPIO，通过 J502.3 和 J502.5 将 GPIO24 和 GPIO25 两个引脚连接在一起。

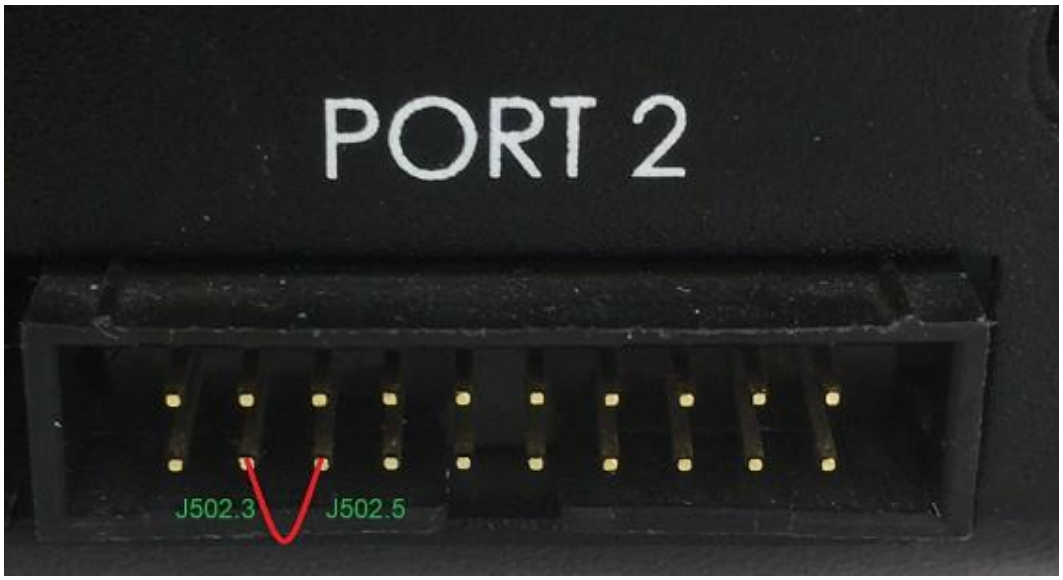


图 4.LS1021A-IoT 上的 GPIO24 和 GPIO25 连接

3.2.2.3 从 U-Boot 源代码构建裸机镜像

构建裸机镜像有两种方法。一种方法是独立编译镜像，本节将对此进行介绍。另一种方法是使用实时边缘框架构建裸机镜像，这在《实时边缘 Yocto Project 用户指南》的“通过 Yocto 构建镜像”一节中有介绍。

3.2.2.3.1 为 slave 内核构建裸机二进制文件

执行下述步骤：

1. 从以下路径下载项目源：

   https://github.com/real-time-edge-sw/real-time-edge-uboot.git

2. 签出到标签：

   - `Real-Time-Edge-v2.0-baremetal-202107`

3. 在您的主机环境中配置跨工具链。

4. 然后，运行以下命令：

```
/* build BareMetal image for LS1021A-IoT board */
 $make ls1021aiot_bm_defconfig
 $make
/* build BareMetal image for LS1028ARDB board */
 $make ls1028ardb_bm_defconfig
 $make
/* build BareMetal image with SAI for LS1028ARDB board*/
 $make ls1028ardb_bm_sai_defconfig
 $make
/* build BareMetal image for LS1043ARDB board */
 $make ls1043ardb_bm_defconfig
 $make
/* build BareMetal image for LS1046ARDB board * /
 $make ls1046ardb_bm_defconfig
 $make
/* build BareMetal image for LX2160ARDB board * /
 $make lx2160ardb_bm_defconfig
 $make
/* build BareMetal image for i.MX 8M Mini EVK board * /
 $make imx8mm_evk_bm_defconfig /* RevC */
 $make
/* build BareMetal image for i.MX 8M Plus EVK board * /
 $make imx8mp_evk_bm_defconfig
 $make
```

5. 最后，生成用于裸机的文件 `u-boot.bin`（或 `u-boot-dtb.bin`，仅适用于 lx2160ardb）。将其复制到 *tftp* 服务器目录。

3.2.2.4 通过 Yocto 构建镜像

构建裸机镜像有两种方法。一种方法是独立编译镜像，如从 U-Boot 源代码构建裸机镜像中所述。第二种方法是使用实时边缘软件框架构建裸机图像，本节将对此进行介绍。

实时边缘软件专为嵌入式工业用途而设计。它是面向工业的集成式 Linux 发行版。使用当前版本，可以方便地构建和实现裸机。

3.2.2.4.1 获取实时边缘软件

最新版本可从以下 URL 获取：

https://github.com/real-time-edge-sw/yocto-real-time-edge.git

按照《实时边缘 Yocto Project 用户指南》第三章获取代码并构建镜像。

3.2.2.4.2 构建裸机镜像

本节介绍为各种板（例如 LS1021A-IoT、LS1043ARDB、LS1046ARDB、LX2160ARDB、i.MX 8M Plus EVK 和 i.MX 8M Mini EVK）构建裸机镜像的步骤。

3.2.2.4.2.1 构建裸机镜像

运行以下命令为 Layerscape 和 i.IMX 平台构建最终的裸机镜像。

```
$ cd yocto-real-time-edge
```

对于 LS1021A-IOT 裸机镜像：

```
$ DISTRO=nxp-real-time-edge-baremetal MACHINE=ls1021aiot source real-time-edge-setup-env.sh -b build-ls1021aiot-bm
```

对于 LS1028ARDB 裸机镜像：

```
$ DISTRO=nxp-real-time-edge-baremetal MACHINE=ls1028ardb source real-time-edge-setup-env.sh -b build-ls1028ardb-bm
```

对于 LS1043ARDB 裸机镜像：

```
$ DISTRO=nxp-real-time-edge-baremetal MACHINE=ls1043ardb source real-time-edge-setup-env.sh -b build-ls1043ardb-bm
```

对于 LS1046ARDB 裸机镜像：

```
$ DISTRO=nxp-real-time-edge-baremetal MACHINE=ls1046ardb source real-time-edge-setup-env.sh -b build-ls1046ardb-bm
```

对于 LX2160ARDB 裸机镜像：

```
$ DISTRO=nxp-real-time-edge-baremetal MACHINE=lx2160ardb source real-time-edge-setup-env.sh -b build-lx2160ardb-bm
```

对于 i.MX 8M Plus EVK 裸机镜像：

```
$ DISTRO=nxp-real-time-edge-baremetal MACHINE=imx8mpevk source real-time-edge-setup-env.sh -b build-imx8mpevk-bm
```

对于 i.MX 8M Mini EVK 裸机镜像：

```
$ DISTRO=nxp-real-time-edge-baremetal MACHINE=imx8mmevk source real-time-edge-setup-env.sh -b build-ix8mmevk-bm
```

然后，使用：

```
$ bitbake nxp-image-real-time-edge
```

3.2.2.4.3 使用裸机启动 Linux

遵循以下步骤使用实时边缘软件构建的镜像启动 Linux + 裸机系统。

对于可以从 SD 卡启动的平台，只需两个步骤即可将镜像编程到 SD 卡中。

1. 将 SD 卡（至少 4 GB 大小）插入任何 Linux 主机。
2. 在构建目录中找到镜像文件（例如：ls1028ardb）：tmp/deploy/images/ls1028ardb/nxp-image-real-time- edge-ls1028ardb.wic.bz2

3. 然后，运行以下命令：

```
$ bzip2 -dnxp-image-real-time-edge-ls1028ardb.wic.bz2
$ sudo dd if=./nxp-image-real-time-edge-ls1028ardb.wicof=/dev/sdx
# or in some other host machine:
$ sudo dd if=./nxp-image-real-time-edge-ls1028ardb.wicof=/dev/mmcblkx
# find the right SD Card device name in your host machine and replace the "sdx" or "mmcblkx".
```

4. 然后，将 SD 卡插入目标板（例如 ls1028ardb）并上电。

上述步骤完成后，Linux 系统在 master 内核（内核 0）上启动，裸机系统在 slave 内核（内核 1）上自动启动。

3.2.3 运行示例

以下各节介绍了如何在 LS1021A-IoT 板的主机环境中运行裸机示例。对于 LS1028ARDB、LS1043ARDB、LS1046ARDB、i.MX 8M Mini EVK 和 i.MX 8M Plus EVK 板，可以遵循类似的步骤。

3.2.3.1 准备控制台

准备一根 USB 转 TTL 串行线，将引脚连接到 Arduino 引脚的 LPUART，以用作 UART1，请参见硬件设置。

要将 LPUART 更改为 UART 引脚，请将 RCW 的第 44 字节修改为 0x00。

在当前的裸机框架设计中，两个 UART 端口用作控制台。UART1 用于 master 内核，UART2 用于 slave 内核。

- 对于 LS1021A，UART2 与 LPUART 引脚复用，因此用户应更改 RCW 以使能 master 内核上的 UART2。此修改已在 LS1021A-IoT 板的代码中实现，因此用户无需修改此板的代码。
- 对于客户特定的板，将更改应用于 RCW 文件：

```
bit[354~356] = 000
bit[366~368] = 111
```

3.2.3.2 运行裸机二进制文件

如前所述，编译裸机框架有两种方法。一种是独立方法，另一种方法使用实时边缘软件。这些方法分别在从 U-Boot 源代码构建裸机镜像和通过 Yocto 构建镜像中介绍。如果您使用实时边缘软件编译裸机镜像，则裸机镜像包含在 *nxp-image-real-time-edge-xxxx.wic.bz2* 中，并且 master 内核自动在 slave 内核上启动裸机镜像。如果使用独立编译方法，您应当执行以下步骤，从 master 内核的 U-Boot 提示符运行裸机二进制文件。以 Layerscape 平台为例：

1. 在 master 上启动 U-Boot 后，使用以下命令在 0x84000000 上下载裸机镜像：`u-boot.bin`：

```
=>tftp 0x84000000 xxxx/u-boot.bin
```

其中

- `xxxx` 是您的 tftp 服务器目录。
- 0x84000000 是 Layerscape 平台裸机上 `CONFIG_SYS_TEXT_BASE` 的地址。

注：i.MX 8M Plus EVK 和 i.MX 8M Mini EVK 板的地址为 0x50200000。

2. 然后，使用以下命令启动裸机内核：

```
=> cpu start 0x84000000
```

3. 最后，UART1 端口显示日志，裸机应用程序在 slave 内核上成功运行。

下图显示了一个示例输出日志。

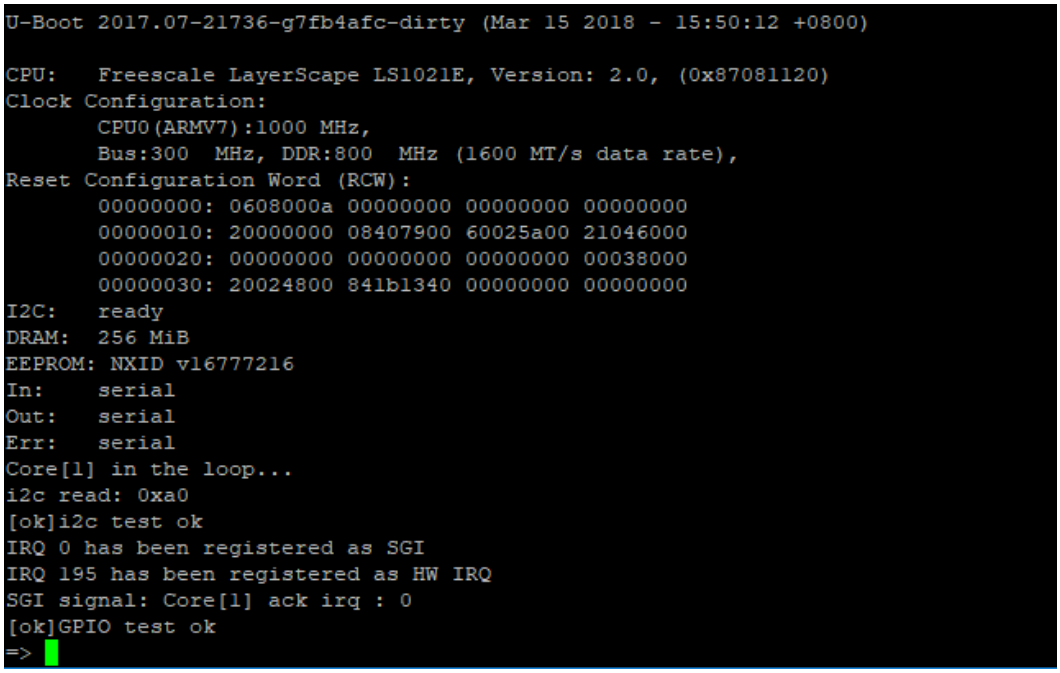


图 5.裸机输出日志

3.2.4 基于裸机框架开发

本章介绍如何基于裸机框架开发客户特定的应用程序。

3.2.4.1 开发裸机应用程序

U-boot 存储库中的目录“app”包括用于测试 I2C、GPIO 和 IRQ 初始化功能的测试案例。您可以编写实际的应用程序并将它们存储在此目录中。

3.2.4.2 示例软件

本节介绍如何分析 GPIO 示例代码并使用它来编写实际应用程序。

3.2.4.2.1 主文件 app.c

文件<industry-Uboot path>/app/app.c，是所有应用程序的主要入口。用户可以修改 app.c 文件来添加他们的应用程序。

- 如果按照从 U-Boot 源代码构建裸机镜像中所述使用独立方法构建裸机镜像，只需将目录更改为行业 Uboot 路径，以检查 app.c 文件。
- 如果使用实时边缘软件编译裸机二进制文件，您应该换到构建目录，以检查 app.c 文件。

以下是文件 app.c 的示例代码，展示了如何添加 I2C、IRQ 和 GPIO 的示例测试案例。

```
void core1_main(void)
{
    test_i2c();
    test_irq_init();
    test_gpio();
```

```
    return;
}
```

3.2.4.2.2 常用头文件

裸机提供了一些常见的 API。下表显示了包含 API 的头文件。

表 9.常用头文件说明

头文件	说明
asm/io.h	读/写 IO API。 例如，raw_readb、raw_writeb、out_be32 和 in_be32。
linux/string.h	用于操作字符串的 API。 例如，strlen、strcpy 和 strcmp。
linux/delay.h	用于延时的 API。 例如，udelay 和 mdelay.
linux/types.h	指定常用类型的 API。 例如，u32 和 u64。
common.h	通用 API。 例如，printf 和 puts。

3.2.4.2.3 GPIO 文件

文件 uboot/app/test_gpio.c 是测试 GPIO 功能的一个示例，它展示了如何编写 GPIO 应用程序。

这是 ls1021aiot 板的示例：

在 ls1021aiot 板上，首先需要 GPIO 头文件 asm-generic/gpio.h，其中包含 GPIO 的所有接口。然后，将 GPIO25 配置为输出方向，将 GPIO24 配置为输入方向。最后，通过将值 1 或 0 写入 GPIO25，可以从 GPIO24 接收相同的值。

下表显示了文件 test_gpio.c 示例中使用的 API。

表 10.GPIO API 及其说明

函数声明	说明
gpio_request (ngpio, label)	请求 GPIO。 • ngpio——用于 GPIO25 的 GPIO 编号，例如 25。 • label——GPIOI 请求的名称。 如果**正常**则返回 **0**，如果**错误**则返回**-1**。
gpio_direction_output (ngpio, value)	将 GPIO 的方向配置为输出并将值写入其中。 • ngpio——GPIO 编号，例如 25，即 GPIO25。 • value——写入此 GPIO 的值。 如果**低**则返回 **0**，如果**高**则返回 **1**，如果**错误**则返回**-1**。

表格接下页……

表 10.GPIO API 及其说明（续）

函数声明	说明
`gpio_direction_input (ngpio);`	将 GPIO 的方向配置为输入。 `ngpio`——GPIO 编号，例如 24，即 GPIO24； 如果**正常**则返回 **0**，如果**错误**则返回**-1**。
`gpio_get_value (ngpio)`	读取值。 • `ngpio`——GPIO 编号，例如 24，即 GPIO24； 如果**低**则返回 **0**，如果**高**则返回 **1**，如果**错误**则返回**-1**。
`gpio_free (ngpio)`	释放刚刚请求的 GPIO。 • `ngpio`——GPIO 编号，例如 24，即 GPIO24； 如果**正常**则返回 **0**，如果**错误**则返回**-1**。

3.2.4.2.4 I2C 文件

文件 `uboot/app/test_i2c.c` 可用作测试 I2C 功能的示例，并展示如何编写 I2C 应用程序。

在 ls1021aiot 板上，包含 I2C 头文件 `i2c.h`，其中包含 I2C 的所有接口。然后，从音频编解码器设备(0x2A)的偏移量 0 读取一个固定的数据。如果数据为 0xa0，则控制台上会显示消息`[ok]I2C test ok`。

在 ls1043ardb 板上，从 INA220 设备(0x40)的偏移量 0 读取固定的数据。如果数据是 0x39，控制台上会显示消息`[ok]I2C test ok`。

下表显示了示例文件 *test_i2c.c* 中使用的 API。

表 11.I2C API 及其说明

函数声明	说明
`int i2c_set_bus_num (unsigned int bus)`	设置 I2C 总线。 `bus`——总线指数，从零开始 如果**正常**则返回 **0**，如果错误则返回**-1**。
`int i2c_read (uint8_t chip, unsigned int addr, int alen, uint8_t *buffer, int len)`	从 I2C 器件芯片读取数据。 • `chip`——I2C 芯片地址，范围 0..127 • `addr`——芯片内的存储器（寄存器）地址 • `alen`——用于地址的字节数（通常为 1，较大的存储器为 2，只有一个寄存器的寄存器类型设备为 0） • `buffer`——读/写数据的位置 • `len`——读/写多少字节 如果**正常**则返回 **0**，如果错误则**不是 0**。
`int i2c_write (uint8_t chip, unsigned int addr, int alen, uint8_t *buffer, int len)`	将数据写入 I2C 器件芯片。 • `chip`——I2C 芯片地址，范围 0..127 • `addr`——芯片内的存储器（寄存器）地址

表格接下页……

表 11.I2C API 及其说明（续）

函数声明	说明
	• alen——用于地址的字节数（通常为 1，较大的存储器为 2，只有一个寄存器的寄存器类型设备为 0） • buffer——读/写数据的位置 • len——读/写多少字节 如果**正常**则返回 **0**，如果错误则**不是 0**。

3.2.4.2.5 IRQ 文件

文件 *uboot/app/test_irq_init.c* 是测试 IRQ 和 IPI（处理器间中断）功能的示例，并展示了如何编写 IRQ 应用程序。下面简要介绍该过程。

文件 asm/interrupt-gic.h 是 IRQ 的头文件，包括它的所有接口。然后，为 SGI 0 注册一个 IRQ 函数。设置 SGI 信号后，CPU 获取此 IRQ 并运行 IRQ 函数。然后，注册一个硬件中断函数，展示如何使用外部硬件中断。

SGI IRQ 用于处理器间中断，只能在裸机内核之间使用。如果您想在裸机内核和 Linux 内核之间进行通信，请参见 ICC 模块。SGI IRQ id 为 0-15。SGI IRQ id“8”为 ICC 保留。

注意

对于 i.MX 8M Mini EVK 和 i.MX 8M Plus EVK 板，SGI IRQ id 为 9。

下表显示了示例文件 *test_irq_init.c* 中使用的 API。

表 12.IRQ API 及其说明

返回类型 API 名称（参数列表）	说明
void gic_irq_register (int irq_num, void (*irq_handle)(int))	注册一个 IRQ 函数。 • irq_num——IRQ id，0-15 用于 SGI，16-31 用于 PPI，32-1019 用于 SPI • irq_handle——IRQ 函数
void gic_set_sgi (int core_mask, u32 hw_irq)	设置 SGI IRQ 信号。 • core_mask——目标内核掩码 • hw_irq——IRQ id
void gic_set_target (u32 core_mask, unsigned long hw_irq)	设置硬件 IRQ 的目标内核。 • core_mask——目标内核掩码 • hw_irq——IRQ id
void gic_set_type (unsigned long hw_irq)	设置硬件 IRQ 的类型，以识别相应的中断是边沿触发的还是电平敏感。 • hw_irq——IRQ id

3.2.4.2.6 QSPI 文件

文件 uboot/app/test_qspi.c 提供了一个可用于测试 QSPI 功能的示例。以下步骤显示了如何编写 QSPI 应用程序：

1. 首先，找到 QSPI 头文件 spi_flash.h 和 spi.h，其中包括 QSPI 的所有接口。
2. 然后，初始化 QSPI flash。随后，擦除对应的 flash 区域，确认擦除操作成功。
3. 现在，以 0x3f00000 的偏移量和 0x40000 的大小读取或写入 flash。

下表显示了文件 test_qsip.c 示例中使用的 API。

表 13.QSPI API

API 名称（类型）	说明
spi_find_bus_and_cs(bus,cs, &bus_dev, &new)	API 查找 SPI 器件是否已存在。 ● “bus”——总线指数，从零开始。 ● “cs”——片选模式的值。 ● “bus_dev”——如果找到总线。 ● “new”——如果找到设备。 如果正常则返回 0，如果错误则返回-ENODEV。
spi_flash_probe_bus_cs(bus, cs, speed, mode, &new)	初始化 SPI flash 设备。 ● “bus”——总线指数，从零开始。 ● “cs”——片选模式的值。 ● “speed”——SPI flash 速度，可以使用 0 或 CONFIG_SF_DEFAULT_SPEED。 ● “mode”——SPI flash 模式，可以使用 0 或 CONFIG_SF_DEFAULT_MODE。 ● “new”——如果设备已初始化。 如果正常则返回 0， 如果错误则返回-ENODEV。
dev_get_uclass_priv(new)	获取 SPI flash。 ● “new”——正在初始化的器件。 如果正常则返回 flash，如果错误则返回 NULL。
spi_flash_erase(flash, offset, size)	擦除指定位置和长度的 flash 内容，擦除所有内容。 ● “flash”——Flash 正在初始化。 ● “offset”——Flash 偏移地址。 ● “size”——擦除数据的长度。 如果正常则返回 0，如果错误则返回!0。
spi_flash_read(flash, offset, len, vbuf)	将 flash 数据读入存储器。 ● “flash”——正在初始化的 flash。 ● “offset”——Flash 偏移地址。 ● “len”——读取数据的长度。 ● “vbug”——存储数据读取的缓冲器 如果正常则返回 0，如果错误则返回!0。

表格接下页……

表 13.QSPI API（续）

API 名称（类型）	说明
`spi_flash_write(flash, offset, len, buf)`	将存储器数据写入 flash。 ● “`flash`”——正在初始化的 flash。 ● “`offset`”——Flash 偏移地址。 ● “`len`”——写入数据的长度。 ● “`buf`”——存储数据写入的缓冲器 如果正常则返回 `0`，如果错误则返回`!0`。

3.2.4.2.7 IFC

LS1043ARDB 和 LS1046ARDB 都有 IFC 控制器。但是，LS1043ARDB 支持 NOR flash 和 NAND flash，而 LS1046ARDB 仅支持 NAND flash。

启动裸机内核时会显示 NOR 和 NAND flash 消息，如下所示：

```
1:NAND: 512 MiB
1:Flash: 128 MiB
```

或(LS1046ARDB)

```
1:NAND: 512 MiB
```

没有示例代码来测试它，但我们可以使用一些命令来验证这些功能。

对于 LS1043ARDB NOR flash（映射存储器地址为 0x60000000），可以使用以下命令进行验证：

```
=> md 0x60000000
1:60000000: 55aa55aa 0001ee01 10001008 0000000a     .U.U............
1:60000010: 00000000 00000000 02005514 12400080     .........U....@.
1:60000020: 005002e0 002000c1 00000000 00000000     ..P.............
1:60000030: 00000000 00880300 00000000 01110000     ................
1:60000040: 96000000 01000000 78015709 10e00000     .........W.x....
1:60000050: 00001809 08000000 18045709 9e000000     .........W......
1:60000060: 1c045709 9e000000 20045709 9e000000     .W.......W.....
1:60000070: 00065709 00000000 04065709 00001060     .W.......W..`...
1:60000080: c000ee09 00440000 58015709 00220000     ......D..W.X..".
1:60000090: 40800089 01000000 40006108 f56b710a     ...@.....a.@.qk.
1:600000a0: ffffffff ffffffff ffffffff ffffffff     ................
```

对于 LS1043ARDB 和 LS1046ARDB 上的 NAND flash，可以使用“nand”命令对其进行验证（nand 擦除、nand 读取、nand 写入等）：

```
=> nand info

1:Device 0: nand0, sector size 128 KiB
1: Page size         2048 b
1: OOB size            64 b
1: Erase size      131072 b
1: subpagesize       2048 b
```

```
1: options              0x00004200
1: bbt options          0x00028000
```

```
=> nand
1:nand - NAND sub-system


1:Usage:
nand info - show available NAND devices
nand device [dev] - show or set current device
nand read - addr off|partition size
nand write - addr off|partition size
     read/write 'size' bytes starting at offset 'off'
     to/from memory address 'addr', skipping bad blocks.
nand read.raw - addr off|partition [count]
nand write.raw[.noverify] - addr off|partition [count]
     Use read.raw/write.raw to avoid ECC and access the flash as-is.
nand erase[.spread] [clean] off size - erase 'size' bytes from offset 'off'
     With '.spread', erase enough for given file size, otherwise,
     'size' includes skipped bad blocks.
nand erase.part [clean] partition - erase entire mtd partition'
nand erase.chip [clean] - erase entire chip'
nand bad - show bad blocks nand dump[.oob] off - dump page
nand scrub [-y] off size | scrub.part partition | scrub.chip
     really clean NAND erasing bad blocks (UNSAFE)
nand markbad off [...]- mark bad block(s) at offset (UNSAFE)
nand biterr off - make a bit error at offset
```

3.2.4.2.8 以太网

文件 uboot/app/test_net.c 提供了一个测试以太网功能的示例，并展示了如何编写网络应用程序来使用该功能。

这是 LS1043ARDB（或 LS1046ARDB）板的示例。

1. 使用以太网线缆将 LS1043ARDB 板的一个以太网端口连接到一台主机。
   - 对于 LS1046ARDB，默认 ethact 为 FM1@DTSEC5。网线应连接到 SGMII1 端口。
   - 对于 LS1043ARDB，默认 ethact 为 FM1@DTSEC3。网线应连接到 RGMII1 端口。
2. 将主机 IP 地址配置为 192.168.1.2。
3. 为 LS1043ARDB 板上电。如果网络已连接，控制台上会显示消息 host 192.168.1.2 is alive。
4. 板和主机的 IP 地址在文件 test_net.c 中定义。在此文件中，使用变量 ipaddr 修改 LS1043ARDB 板的 IP 地址，并使用变量 ping_ip 更改主机的 IP 地址。

下表列出了 Net API 及其说明。

表 14.网络 API 及其说明

API 名称（类型）	说明
void net_init (void)	初始化网络
int net_loop (enum proto_t protocol)	主网络处理循环。 ● 枚举 proto_t 协议——协议类型

表格接下页……

表 14.网络 API 及其说明（续）

API 名称（类型）	说明
`int eth_receive (void *packet, int length)`	从 NIC 器件芯片读取数据。 ● 无效*数据包 ● 长度——网络数据包长度 返回长度
`int eth_send (void *packet, int length)`	将数据写入 NIC 器件芯片。 ● 数据包——发送数据包的指针 ● 长度——网络数据包长度 返回长度。

3.2.4.2.9 USB 文件

文件 `uboot/app/test_usb.c` 提供了一个可用于测试 USB 功能的示例。以下步骤显示了如何编写 USB 应用程序：

1. 将 USB 磁盘连接到 USB 端口。
2. 包括头文件 `usb.h`，其中包含 USB 的所有 API。
3. 使用 `usb_init` API 初始化 USB 设备。
4. 使用 `usb_stor_scan` API 扫描 USB 总线上的 USB 存储设备。
5. 使用 `blk_get_devnum_by_type` API 获取设备编号。
6. 使用 `blk_dread` API 从 USB 磁盘读取数据。
7. 使用 `blk_dwrite` API 将数据写入 USB 磁盘。

下表显示了文件 `test_usb.c` 示例中使用的 API：

表 15.USB API 及其说明

API 名称（类型）	说明
`int usb_init(void)`	初始化 USB 控制器。
`int usb_stop(void)`	停止 USB 控制器。
`int usb_stor_scan(int mode)`	如果模式 = 1，则扫描 USB 并向用户报告设备信息 ● 模式——如果模式 = 1，则将信息返回给用户。 返回 ● 当前设备，或 ● -1（如果未找到设备）。
`struct blk_desc *blk_get_devnum_by_type(enum if_type if_type, int devnum)`	按类型和编号获取块设备。 ● `if_type`——块设备类型 ● `devnum`——设备编号 返回

表格接下页……

表 15.USB API 及其说明（续）

	• 指向块设备描述符，或 • NULL（如果未找到）。
`unsigned long blk_dread(struct blk_desc *block_dev, lbaint_t start, lbaint_t blkcnt, void *buffer);`	从 USB 设备读取数据。 • `block_dev`——块设备描述符 • `start`——开始块 • `blkcnt`——块号 • `buffer`——存储数据的缓冲器 返回从中读取数据的块号。
`unsigned long blk_dwrite(struct blk_desc *block_dev, lbaint_t start, lbaint_t blkcnt, const void *buffer);`	将数据写入 USB 设备。 • `block_dev`——块设备描述符 • `start`——开始块 • `blkcnt`——块号 • `buffer`——存储数据的缓冲器 返回写入数据的块号。

3.2.4.2.10 PCIe 文件

文件 `app/test_pcie.c` 提供了测试 PCIe 和网卡（如 e1000）功能的示例代码。以下步骤显示了如何编写 PCIe 和网络应用程序：

1. 将 PCIe 网卡（如 e1000）插入 PCIe2 或 PCIe3 插槽（如果存在）。
2. 将主机的 IP 地址配置为 192.168.1.2。
3. 包括文件 `include/pci.h` 和 `include/netdev.h`。
4. 使用 `pci_init` API 初始化 PCIe 控制器。
5. 使用 `uclass_get_device_by_seq` API 按名称获取 `uclass` 设备。
6. 使用 `pci_eth_init` API 初始化 PCIe 网络设备。
7. 开始使用 `net_loop` API ping 主机。

下表显示了文件 `test_pcie.c` 示例中使用的 API。

表 16.PCIe API 及其说明

API 名称（类型）	说明
`void pci_init(void)`	初始化 PCIe 控制器。不返回值。
`int uclass_get_device_by_seq(enum uclass_id id, int seq, struct udevice **devp)`	根据 ID 和序列获取 uclass 设备： • `id`——uclass ID • `seq`——序列 • `devp`——指向设备的指针 返回：

表格接下页……

表 16.PCIe API 及其说明（续）

	● 如果正常则为 0。 ● 如果错误则为负值。
`static inline int pci_eth_init(bd_t *bis)`	初始化 PCIe 总线上的网卡。 ● `Bis`——包含由共享代码访问的变量的结构 返回网卡的数量。
`int net_loop (enum proto_t protocol)`	主网络处理循环。 ● `enum proto_t protocol`——协议类型 返回： ● 如果正常则为 0。 ● 如果错误则为负值。

3.2.4.2.11 CAN 文件

文件 `app/test_flexcan.c` 提供了一个示例测试案例来测试 flexCAN 和 CANopen 功能。以下步骤显示了设计过程：

1. 为 flexCAN 注册接收中断函数。
2. 为 flextimer 注册溢出中断函数。
3. 初始化回调函数列表。
4. 设置该节点的节点 ID。

下表显示了文件 `test_flexcan.c` 示例中使用的 API。

表 17.CAN API 及其说明

API 名称（类型）	说明
`void test_flexcan(void)`	它是 flexCAN 的测试代码入口。
`void flexcan_rx_irq(struct can_module *canx)`	FlexCAN 接收中断函数。 ● `canx`——flexCAN 接口。
`void flexcan_receive(struct can_module *canx, struct can_frame *cf)`	FlexCAN 接收 CAN 数据帧。 ● `canx`——FlexCAN 接口。 ● `cf`——CAN 消息。
`UNS8 setState(CO_Data* d, e_nodeState newState)`	设置节点状态 ● `d`——对象字典 ● `newState`——需要设置的状态。 返回： ● 如果正常则为 0， ● 如果错误则> 0。
`void canDispatch(CO_Data* d, Message *m)`	CANopen 处理 CAN 接收的数据帧：

表格接下页……

表 17.CAN API 及其说明（续）

	• d——对象字典。 • m——接收到的 CAN 消息。
int flexcan_send(struct can_module *canx, struct can_frame *cf)	FlexCAN 接口发送 CAN 消息： • canx——FlexCAN 接口。 • cf——CAN 消息。
void flextimer_overflow_irq(void)	Flextimer 溢出中断处理程序函数。
void timerForCan(void)	CANopen 虚拟时钟。每 100 μs 调用此函数。

- 以下日志显示 CANopen slave 节点状态：

```
=>flexcan error: 0x42242!
Note: slave node entry into the stop mode!
Note: slave node initialization is complete!
Note: slave node entry into the preOperation mode!
Note: slave node entry into the operation mode!
Note: slave node initialization is complete!
Note: slave node entry into the preOperation mode!
Note: slave node entry into the operation mode!
```

3.2.4.2.12 ENETC 文件

文件 app/test_net.c 提供了一个测试 ENETC 以太网功能的示例，并展示了如何编写网络应用程序来使用该功能。这是使用网络 API 的一种特殊情况。

ENETC 的文件 test_net 只是 LS1028ARDB 板的一个示例（CONFIG_ENETC_COREID_SET 已使能）。

1. 使用以太网线缆将 LS1028ARDB 板的 ENETC 端口连接到一台主机。
2. 将主机 IP 地址配置为 192.168.1.2。
3. 为 LS1028ARDB 板上电。如果网络已连接，控制台上会显示消息 host 192.168.1.2 is alive。
4. 板和主机的 IP 地址在文件 test_net.c 中定义。在此文件中，使用变量 ipaddr 修改 LS1028ARDB 板的 IP 地址，并使用变量 ping_ip 更改主机的 IP 地址。

下表列出了 ENETC 的网络 API 及其说明，其他网络 API 请参见第 4.2.7 节。

表 18.ENETC API 及其说明

API 名称（类型）	说明
void pci_init(void)	初始化 PCIe 控制器。不返回值。
void eth_initialize(void)	初始化以太网。

3.2.4.2.13 SAI 文件

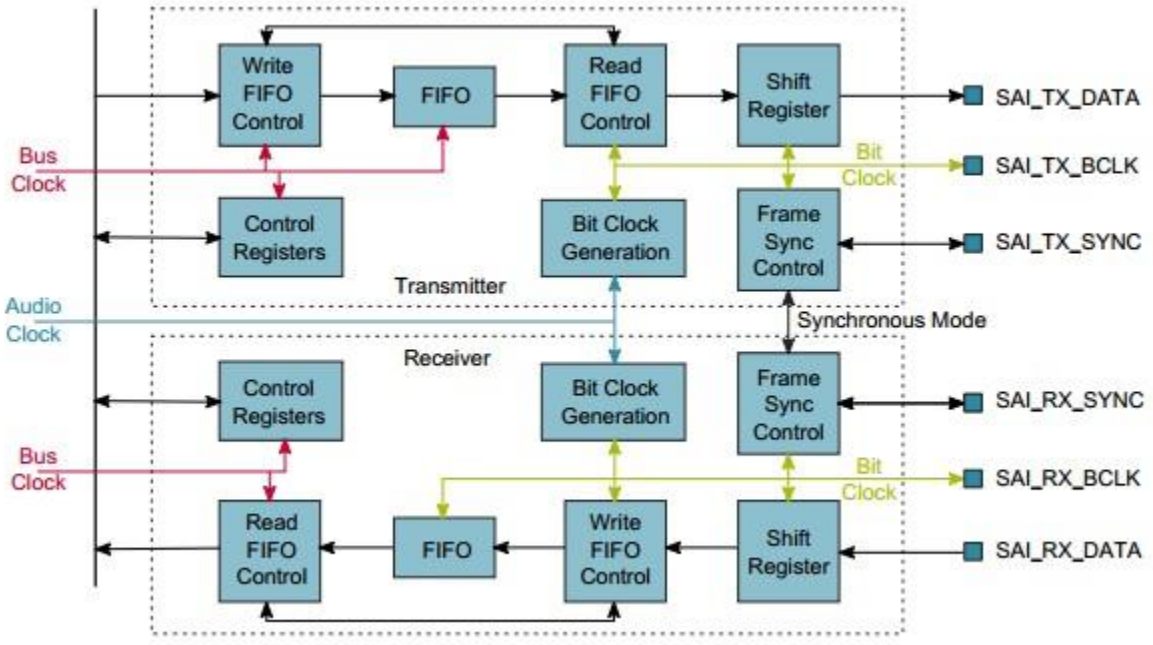
音频功能需要 SAI 模块和编解码器驱动程序。以下各节介绍了 SAI 模块、音频编解码器(SGTL5000)，并详细介绍了如何将音频与裸机集成以及在裸机上运行音频应用程序。

3.2.4.2.13.1 同步音频接口(SAI)

LS1028A 集成了 6 个 SAI 模块，但 LS1028ARDB 板只使用了 SAI4。同步音频接口(SAI)支持具有帧同步的全双工串行接口。SAI 的位时钟和帧同步都在外部产生(SGTL5000)。

- 具有独立位时钟和帧同步的发射器，支持 1 条数据线
- 具有独立位时钟和帧同步的接收器，支持 1 条数据线
- 最大帧大小为 32 个字
- 字大小在 8 位和 32 位之间
- 为帧中的第一个字和其余字单独配置字大小
- 每个发送和接收通道的异步 32 × 32 位 FIFO
- 支持 FIFO 错误后的正常重启

图 SAI 框图



3.2.4.2.13.2 音频编解码器(SGTL5000)

SGTL5000 是一款带有恩智浦耳机放大器的低功耗立体声编解码器。它旨在为需要 LINEIN、MIC_IN、LINEOUT、耳机输出和数字 I/O 的产品提供完整的音频解决方案。它允许 8.0 MHz 至 27 MHz 系统时钟作为输入。编解码器支持 8.0 kHz、11.025 kHz、12 kHz、16 kHz、22.05 kHz、24 kHz、32 kHz、44.1 kHz、48 kHz 和 96 kHz 采样频率。LS1028ARDB 板为 SGTL5000 提供 25 MHz 晶体振荡器。

SGTL5000 提供两个接口（I2C 和 SPI）来设置寄存器。LS1028ARDB 板使用 I2C 接口。

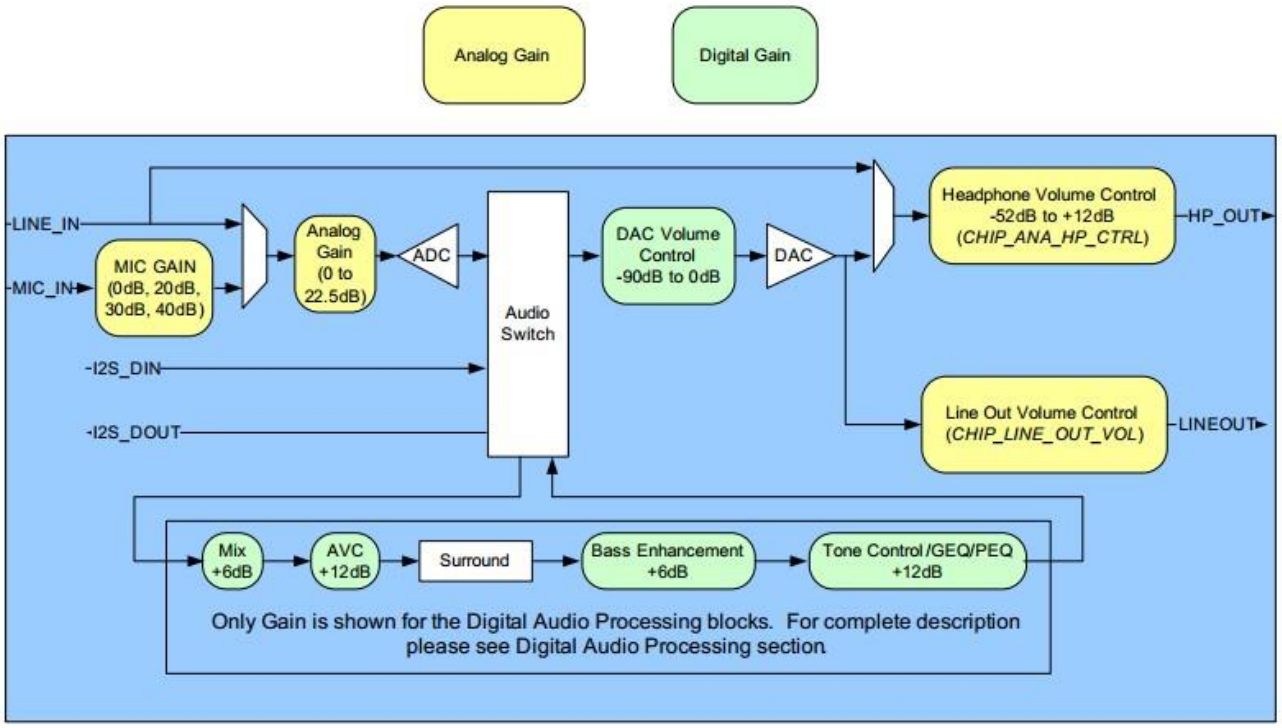
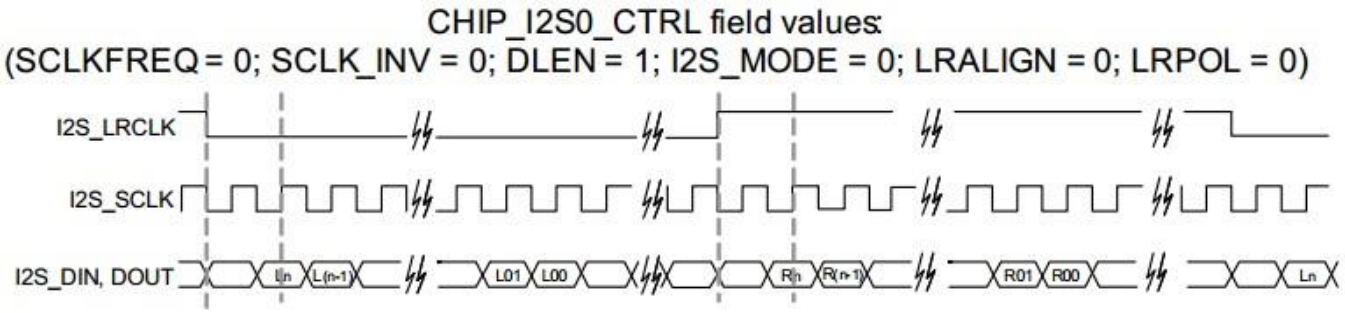


图 6.系统框图、信号流和增益

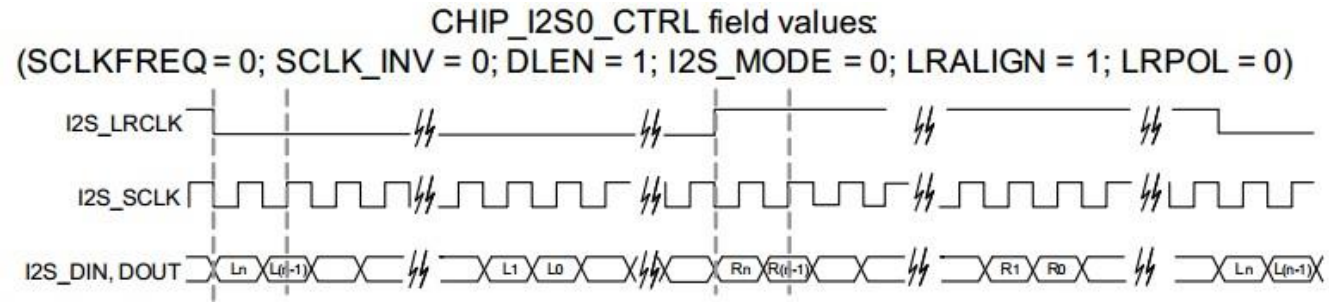
3.2.4.2.13.3 数字接口格式

SGTL5000 提供五种常见的数字接口格式。SAI 和 SGTL5000 数字接口格式必须相同。

- **I2S 格式（n = 位长度）**

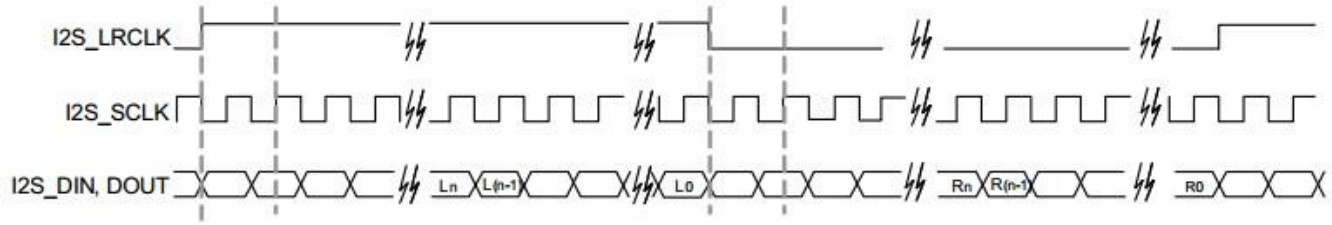


- **左对齐格式（n = 位长度）**



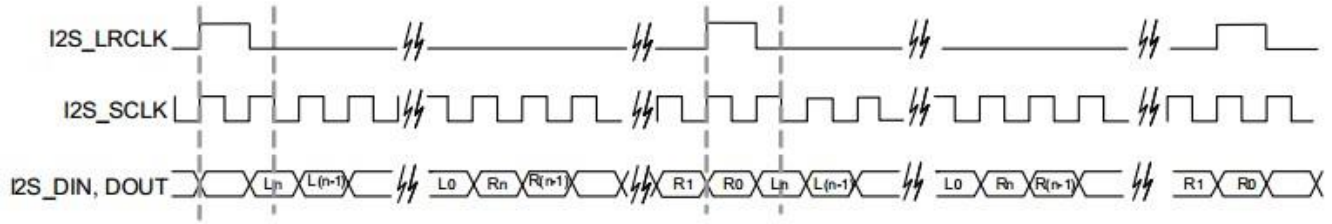
- **右对齐格式（n = 位长度）**

CHIP_I2S0_CTRL field values:
SCLKFREQ = 0; SCLK_INV = 0; DLEN = 1; I2S_MODE = 1; LRALIGN = 1; LRPOL = 0)



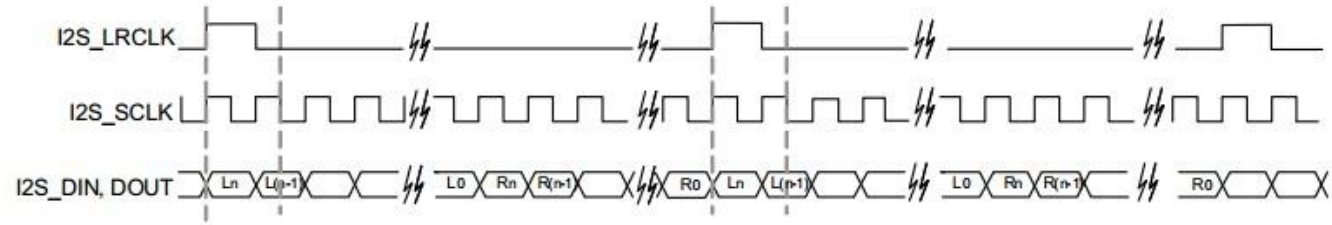
- **PCM 格式 A**

CHIP_I2S0_CTRL = 0x01F4
(SCLKFREQ = 1; MS = 1; SCLK_INV = 1; DLEN = 3; I2S_MODE = 2; LRALIGN = 0)



- **PCM 格式 B**

CHIP_I2S0_CTRL = 0x01F6
(SCLKFREQ = 1; MS = 1; SCLK_INV = 1; DLEN = 3; I2S_MODE = 2; LRALIGN = 1)



3.2.4.2.13.4 运行 SAI 应用程序

为了运行 SAI 应用程序，裸机镜像应该在 SAI 支持下重新构建。

1.在实时边缘软件中使能 SAI 支持

```
$ cd yocto-real-time-edge/sources/meta-real-time-edge
# Open file "conf/distro/include/real-time-edge-base.inc", add "sai" to
"DISTRO_FEATURES_append_ls1028ardb" like this:
DISTRO_FEATURES_append_ls1028ardb = " jailhouse real-time-edge-libbee real-time-edge-libblep libnfc-
nci \
    wayland-protocols weston imx-gpu-viv libdrm kmscube \
    real-time-edge-sysrepo tsn-scripts wayland sai"
```

2. 构建镜像

```
$ cd yocto-real-time-edge $ DISTRO=nxp-real-time-edge-baremetal MACHINE=ls1028ardb source real-time-edge-setup-env.sh -b build-ls1028ardb-bm $ bitbake nxp-image-real-time-edge
```

3. 启动板后在 slave 内核心中播放演示音频文件：

```
=>wavplayer ********************************************************** audioformat: PCM nchannels: 1
samplerate: 16000 bitrate: 256000 blockalign: 2 bps: 16 datasize: 67968 datastart: 44
********************************************************** sgtl5000 revision 0x11
fsl_sai_ofdata_to_platdata Probed sound 'sound' with codec 'codec@a' and i2s 'sai@f130000'
i2s_transfer_tx_data The music waits for the end! The music is finished!
**********************************************************
```

3.2.4.3 ICC 模块

内核间通信(ICC)模块适用于 Linux 内核（master）和裸机内核（slave）。它通过 SGI 内核间中断和共享存储器块提供内核间的数据传输。它可以支持多核芯片平台，同时高效地传输数据。

ICC 模块结构基于两个基础：

- SGI：Arm GIC 中的软件生成中断，用于生成内核间中断。ICC 模块对所有 Linux 和裸机内核使用 8 号 SGI 中断。
- 共享的存储器：所有平台内核共享的存储器空间。共享存储器的基地址和大小应在编译前在头文件中进行定义。

ICC 模块可以同时工作，在多核平台之间无锁，支持带有缓冲器描述符环机制的广播案例。

下图显示了从内核 0 到内核 1 的数据传输的基本工作原理。在数据写入和头点移动到下一个位置之后，内核 0 触发一个 SGI(8)到内核 1。此步骤之后，内核 1 获取 BD 环更新状态并读取新数据，然后将尾点移动到下一个位置。

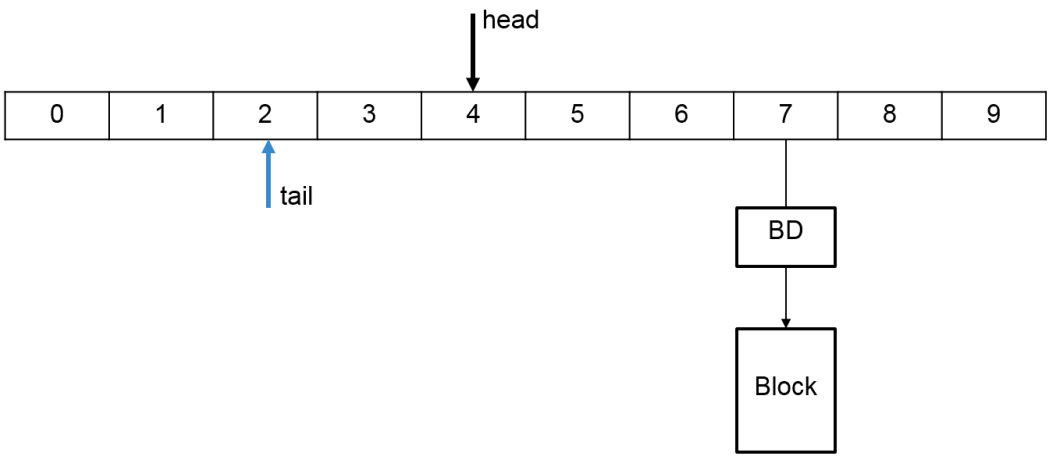


图 7.用于内核间通信的 BD 环

对于多核平台（即四核），总 BD 环排列如下图所示。（参见内核 0 和内核 1 上的 BD 环。）

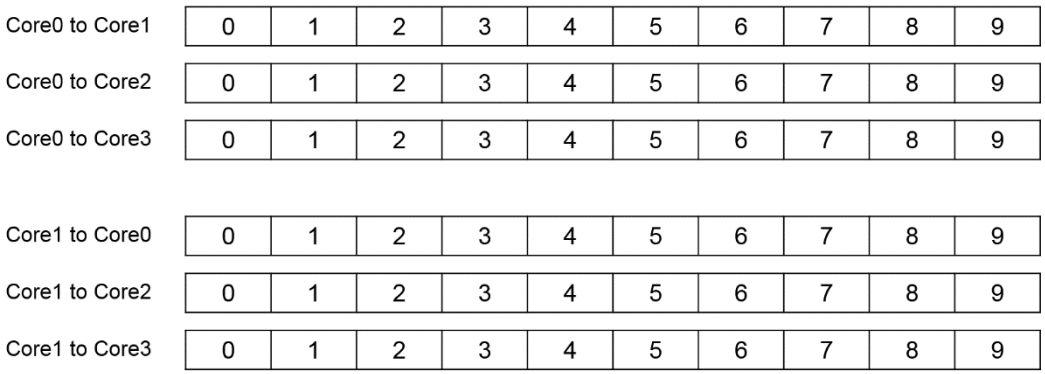


图 8.用于多核平台的 BD 环

所有的 ICC 环结构、BD 结构和数据块都位于共享存储器中。四核平台 ICC 模块将映射共享存储器，如下图所示。

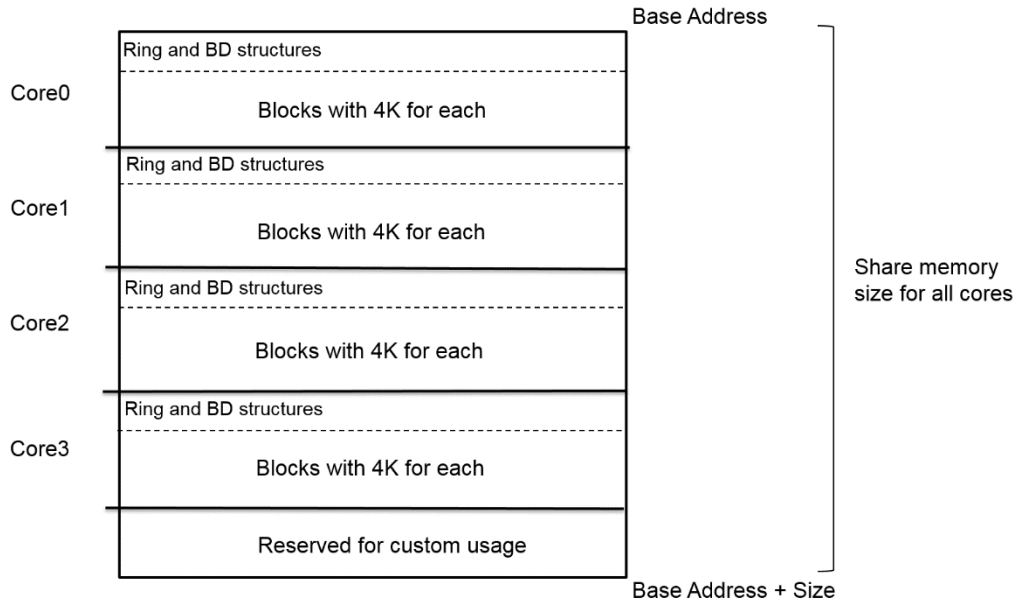


图 9.四核平台的 ICC 共享存储器映射

通常，内核 0 作为 master 内核运行 Linux，其他内核作为 slave 运行裸机。它们获得相同大小的共享存储器来构造环和 BD，并为每个块以 4k 为单位分割块空间。共享存储器顶部的保留空间超出 ICC 模块，用于自定义使用。

对于具有两个内核的 LS1021A 平台，共享存储器映射定义为：

- 共享存储器总大小为 256 MB。
- 为自定义使用保留的空间是共享存储器空间顶部的 16 MB。
- 内核 0 作为 master 内核运行 Linux，ICC 的共享存储器大小为 120 MB，其中环和 BD 结构空间为 2 M，数据的块空间为 118 MB，每个块 4K。
- 内核 1 作为 slave 内核运行裸机，ICC 的共享存储器大小为 120 MB，其中环和 BD 结构空间为 2M，数据的块空间为 118 MB，每个块 4K。

ICC 模块包括两部分代码：

- Linux 用户空间的 ICC 代码，用于 master 内核和 slave 内核之间的数据传输。该代码集成到实时边缘软件中并命名为 `real-time-edge-icc`。编译完成后，将 icc 二进制文件放入 Linux 文件系统。
- 裸机的 ICC 代码在每个 slave 内核上运行，用于裸机内核和 master 内核之间的数据传输。

存储库中 Linux 用户空间的 ICC 代码：https://github.com/real-time-edge-sw/real-time-edge-icc.git

```
├──── icc-main.c——示例案例命令
├──── inter-core-comm.c
├──── inter-core-comm.h——包含使用 ICC 模块的头文件
└──── Makefile
```

裸机目录下裸机的 ICC 代码：

```
baremetal/
├──── arch/arm/lib/inter-core-comm.c
├──── arch/arm/include/asm/inter-core-comm.h——包含使用 ICC 模块的头文件
└──── cmd/icc.c——示例案例命令
```

API ICC 被导出用于 Linux 用户空间和裸机代码中。

表 19.ICC API

API	说明
`unsigned long icc_ring_state(int coreid)`	检查环和块状态。 返回： • `0`——如果为空。 • `!0`——当前的工作块地址。
`Unsigned long icc_block_request(void)`	请求一个块，大小为 ICC_BLOCK_UNIT_SIZE。 返回： • `0`——失败。 • `!0`——可以使用块地址。
`void icc_block_free(unsigned long block)`	释放请求的块。 如果目标内核正在使用此块，请谨慎。
`int icc_irq_register(int src_coreid, void (*irq_handle)(int, unsigned long, unsigned int))`	为接收到的数据注册 ICC 回调处理程序。 返回： • `0`——如果成功 • `-1`——如果失败。
`int icc_set_block(int core_mask, unsigned int byte_count, unsigned long block)`	将块中的数据发送到内核或多核。 这会触发 SGI 中断。 返回：

表格接下页……

表 19.ICC API（续）

API	说明
	• 0——如果成功 • -1——如果失败。
void icc_show(void)	显示 ICC 基本信息。
int icc_init(void)	初始化 ICC 模块。

3.2.4.3.1 ICC 示例

本节提供 Linux 用户空间和裸机代码中用例的示例命令。它们可用于方便地检查和验证 ICC 模块。

1. 在 Linux 用户空间，使用命令 icc 显示支持的案例。

```
[root@LS1046ARDB ~] # icc
icc show - Shows all icc rings status at this core
icc perf <core_mask><counts> - ICC performance to cores <core_mask> with <counts> bytes
icc send <core_mask><data><counts> - Sends <counts><data> to cores <core_mask>
icc irq <core_mask><irq> - Sends SGI <irq> ID[0 - 15] to<core_mask>
icc read <addr><counts> - Reads <counts> 32bit register from <addr>
icc write <addr><data> - Writes <data> to a register <addr>
```

同样，在裸机系统中，使用命令 icc 查看支持的案例。

```
=>icc
1:icc - Inter-core communication via SGI interrupt

1:Usage:
icc show                              - Show all icc rings status at thiscore
icc perf <core_mask><counts>          - ICC performance to cores <core_mask>with
<counts> bytes
icc send <core_mask><data><counts>    - Send <counts><data> to cores<core_mask>
icc irq <core_mask><irq>              - Send SGI <irq>ID[0 - 15] to<core_mask>
```

2. 采用 Linux（内核 0） + 裸机(内核 1、2、3)系统的 LS1046ARDB 上的 ICC 模块命令示例：

运行 icc send 0x2 0x55 128 将 128 字节数据 0x55 发送到内核 1。

```
[root@LS1046ARDB ~] # icc send 0x2 0x55 128
gic_base: 0xffffa033f000, share_base: 0xffff9133f000, share_phy:0xd0000000,
block_phy: 0xd0200000

ICC send testing ...
Target cores: 0x2, bytes: 128
ICC send: 128 bytes to 0x2 cores success

all cores: reserved_share_memory_base: 0xdf000000; size:16777216

mycoreid: 0; ICC_SGI: 8; share_memory_size: 62914560
block_unit_size: 4096; block number: 14848; block_idx: 0

#ring 0 base: 0xffff9133f000; dest_core: 0; SGI: 8
desc_num: 128; desc_base: 0xd00000c0; head: 0; tail: 0
busy_counts: 0; interrupt_counts: 0
```

```
#ring 1 base: 0xffff9133f030; dest_core: 1; SGI: 8
desc_num: 128; desc_base: 0xd00008c0; head: 1; tail: 1
busy_counts: 0; interrupt_counts: 1

#ring 2 base: 0xffff9133f060; dest_core: 2; SGI: 8
desc_num: 128; desc_base: 0xd00010c0; head: 0; tail: 0
busy_counts: 0; interrupt_counts: 0

#ring 3 base: 0xffff9133f090; dest_core: 3; SGI: 8
desc_num: 128; desc_base: 0xd00018c0; head: 0; tail: 0
busy_counts: 0; interrupt_counts: 0
```

同时，内核 1 打印接收信息。

```
=> 1:Get the ICC from core 0; block: 0xd0200000, bytes: 128, value:0x55
```

3. ICC 命令在裸机端运行

```
=>icc send 0x1 0xaa 128
1:ICC send testing ...
1:Target cores: 0x1, bytes: 128
1:ICC send: 128 bytes to 0x1 cores success

1:all cores: reserved_share_memory_base: 0xdf000000; size:16777216

1:mycoreid: 1; ICC_SGI: 8; share_memory_size: 62914560
1:block_unit_size: 4096; block number: 14848; block_idx:0

1:#ring 0 base: 00000000d3c00000; dest_core: 0; SGI: 8
1:desc_num: 128; desc_base: 00000000d3c000c0; head: 1; tail:1
1:busy_counts: 0; interrupt_counts: 1

1:#ring 1 base: 00000000d3c00030; dest_core: 1; SGI: 8
1:desc_num: 128; desc_base: 00000000d3c008c0; head: 0; tail:0
1:busy_counts: 0; interrupt_counts: 0

1:#ring 2 base: 00000000d3c00060; dest_core: 2; SGI: 8
1:desc_num: 128; desc_base: 00000000d3c010c0; head: 0; tail:0
1:busy_counts: 0; interrupt_counts: 0

1:#ring 3 base: 00000000d3c00090; dest_core: 3; SGI: 8
1:desc_num: 128; desc_base: 00000000d3c018c0; head: 0; tail:0
1:busy_counts: 0; interrupt_counts: 0
```

然后，内核 0 端(Linux)会接收这些数据：

```
[root@LS1046ARDB ~] # [ 4247.733753] 000: Get the ICC from core 1; block: 0xd3e00000, bytes:
128, value: 0xaa
```

3.2.4.4 硬件资源分配

本节介绍如何根据应用和使用的参考设计板修改硬件资源分配。

3.2.4.4.1 LS1021A-IoT 板

3.2.4.4.1.1 Linux DTS

移除 DTS 上的 cpu1 节点，并移除裸机使用过的所有设备。

3.2.4.4.1.2 内存配置

LS1021A-IoT 板具有 1 GB 大小的 DDR。DDR 存储器可配置为三个分区：内核 0(Linux)为 512M，内核 1（裸机）为 256M，共享存储器为 256M。

配置位于路径：`include/configs/ls1021aiot_config.h`。

```
#define CONFIG_SYS_DDR_SDRAM_SLAVE_SIZE (256 * 1024 * 1024)
#define CONFIG_SYS_DDR_SDRAM_MASTER_SIZE (512 * 1024 * 1024)
```

注意

内存配置必须与内核 0 的 U-Boot 配置一致。

修改 *include/configs/ls1021aiot_config.h 文件*中的“`CONFIG_SYS_MALLOC_LEN`”，以更改 malloc 的最大大小。

您可以使用 `malloc.h` 中包含的函数在程序中分配或释放存储器。下表列出了这些函数。

表 20.存储器 API 的说明

API 名称（类型）	说明
void_t* malloc (size_t n)	分配存储器 • `n`——分配块的长度 • 返回指向新分配块的指针
void free (void *ptr)	释放 ptr 指向的存储器块（其中 `ptr` 是指向存储器块的指针）

裸机的内存配置如下图所示。

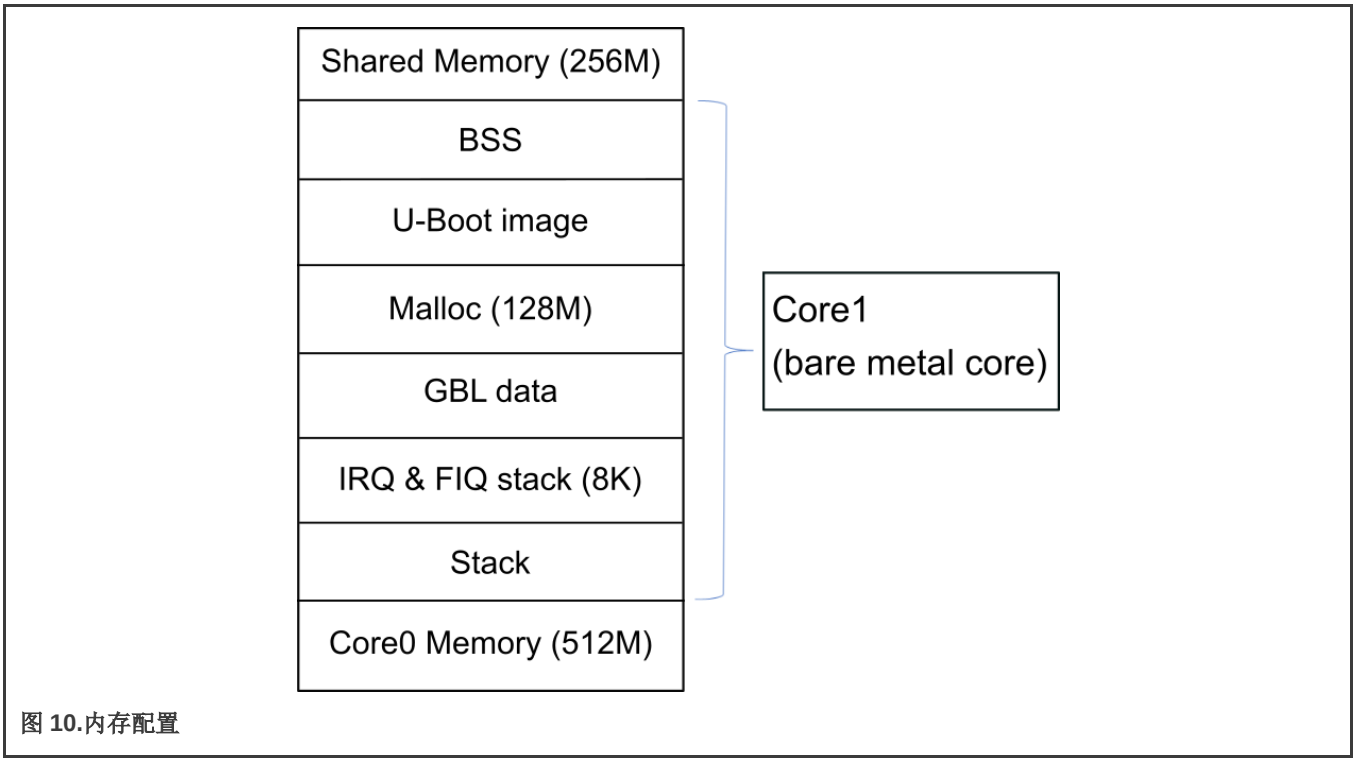


图 10.内存配置

3.2.4.4.1.3 GPIO

LS1021A 有四个 GPIO 控制器。配置在文件：*arch/arm/dts/ls1021a-iot.dtsi* 中定义。您可以在文件 ls1021a-iot.dtsi 中添加 GPIO 节点，以将 GPIO 资源分配给不同的内核。以下是添加 GPIO 节点的示例代码。

```
&gpio2
{
 status = "okay";
};
```

3.2.4.4.1.4 I2C

LS1021A 具有三个 I2C 控制器。使用以下命令在 `ls1021aiot_config.h` 上配置 I2C 总线：

```
// include/configs/ls1021aiot_config.h:
#define CONFIG_SYS_I2C_MXC_I2C1 /* enable I2C bus 1 */
#define CONFIG_SYS_I2C_MXC_I2C2 /* enable I2C bus 2 */
#define CONFIG_SYS_I2C_MXC_I2C3 /* enable I2C bus 3 */
```

3.2.4.4.1.5 硬件中断

LS1021A 有 6 个 IRQ 作为外部 IO 信号连接到中断控制器。我们可以在裸机内核上使用这 6 个 IRQ。这些信号 IRQ0-IRQ5 的 ID 为：195、196、197、199、200 和 201。

GIC 中断 API 在文件 `asm/interrupt-gic.h` 中定义。以下示例显示了如何注册硬件中断：

```
//register HW interrupt
void gic_irq_register(int irq_num, void (*irq_handle)(int));
void gic_set_target(u32 core_mask, unsigned long hw_irq);
void gic_set_type(unsigned long hw_irq);
```

3.2.4.4.1.6 IFC

LS1021A-IoT 板没有 IFC 设备，但 LS1021A SoC 有一个 IFC 接口。由于 IFC 与 QSPI 复用，您应当修改 RCW 以使用 IFC 接口，并添加以下提供的示例代码中所示的配置。用户可以根据实际情况修改代码以支持 IFC。

```
#define CONFIG_FSL_IFC
#define CONFIG_SYS_CPLD_BASE 0x7fb00000
#define CPLD_BASE_PHYS CONFIG_SYS_CPLD_BASE
#define CONFIG_SYS_FPGA_CSPR_EXT (0x0)

#define CONFIG_SYS_FPGA_CSPR (CSPR_PHYS_ADDR(CPLD_BASE_PHYS) | \
CSPR_PORT_SIZE_8 | \
CSPR_MSEL_GPCM | \
CSPR_V)

#define CONFIG_SYS_FPGA_AMASK IFC_AMASK(64 * 1024)
#define CONFIG_SYS_FPGA_CSOR (CSOR_NOR_ADM_SHIFT(4) | \
CSOR_NOR_NOR_MODE_AVD_NOR | \
CSOR_NOR_TRHZ_80)

/* CPLD Timing parameters for IFC GPCM */
#define CONFIG_SYS_FPGA_FTIM0 (FTIM0_GPCM_TACSE(0xf) | \
FTIM0_GPCM_TEADC(0xf) | \
FTIM0_GPCM_TEAHC(0xf))
```

```
#define CONFIG_SYS_FPGA_FTIM1 (FTIM1_GPCM_TACO(0xff) | \
FTIM1_GPCM_TRAD(0x3f))

#define CONFIG_SYS_FPGA_FTIM2 (FTIM2_GPCM_TCS(0xf) | \
FTIM2_GPCM_TCH(0xf) | \
FTIM2_GPCM_TWP(0xff))

#define CONFIG_SYS_FPGA_FTIM3 0x0
#define CONFIG_SYS_CSPR1_EXT CONFIG_SYS_FPGA_CSPR_EXT
#define CONFIG_SYS_CSPR1 CONFIG_SYS_FPGA_CSPR
#define CONFIG_SYS_AMASK1 CONFIG_SYS_FPGA_AMASK
#define CONFIG_SYS_CSOR1 CONFIG_SYS_FPGA_CSOR
#define CONFIG_SYS_CS1_FTIM0 CONFIG_SYS_FPGA_FTIM0
#define CONFIG_SYS_CS1_FTIM1 CONFIG_SYS_FPGA_FTIM1
#define CONFIG_SYS_CS1_FTIM2 CONFIG_SYS_FPGA_FTIM2
#define CONFIG_SYS_CS1_FTIM3 CONFIG_SYS_FPGA_FTIM3
```

3.2.4.4.1.7 USB

LS1021AIOT 有一个 DW3 USB 控制器，默认分配给第二个内核。使用命令 `make menuconfig` 重新配置 U-Boot 以将其分配给其他内核。

```
ARM architecture --->
[*] Enable baremetal
[*] Enable USB for baremetal
(1) USB0 is assigned to core1
(1) USB Controller numbers
```

3.2.4.4.1.8 PCIe

LS1021AIOT 有两个 PCIe 控制器。默认情况下，一个分配给内核 0，另一个分配给内核 1。使用 `make menuconfig` 命令重新配置 U-Boot，以便重新分配内核。

```
ARM architecture --->
[*] Enable baremetal
[*] Enable PCIE for baremetal
(0)     PCIe1 is assigned to core0
(1)     PCIe2 is assigned to core1
(2)     PCIe Controller numbers
```

3.2.4.4.1.9 FlexCAN

将 CAN3 分配给裸机

在裸机中，端口通过 `flexcan.c` 文件进行分配。`Flexcan.c` 路径是 `industry-uboot/drivers/flexcan/ flexcan.c`。在此文件中，您应当定义以下变量：

```
struct can_bittiming_t flexcan3_bittiming = CAN_BITTIM_INIT(CAN_500K);
```

注：您还应当设置 CAN 端口的位时序和波特率(500K)。

```
struct can_ctrlmode_t flexcan3_ctrlmode = {
.loopmode = 0, /* Indicates whether the loop mode is enabled */
```

```
.listenonly = 0, /* Indicates whether the only-listen mode is enabled */
.samples = 0,
.err_report = 1,
};
struct can_init_t flexcan3 = {
.canx = CAN3,  /* Specify CAN port */
.bt = &flexcan3_bittiming,
.ctrlmode = &flexcan3_ctrlmode,
.reg_ctrl_default = 0,
.reg_esr = 0
};
```

可选参数

- **CAN 端口**

```
#define CAN3   (struct can_module *)CAN3_BASE)
#define CAN4   (struct can_module *)CAN4_BASE)
```

- **波特率**

```
#define CAN_1000K 10
#define CAN_500K 20
#define CAN_250K 40
#define CAN_200K 50
#define CAN_125K 80
#define CAN_100K 100
#define CAN_50K 200
#define CAN_20K 500
#define CAN_10K 1000
#define CAN_5K 2000
```

使用命令 `make menuconfig` 为 LS1021A 参考设计板配置 FlexCAN 设置，如下所示：.

```
Device Drivers --->
CAN support    --->
[*] Support for Freescale FLEXCAN based chips
[*]     Support for canfestival
```

3.2.4.4.2 LS1028ARDB 板

本节介绍 LS1028A 参考设计板的 ENETC 配置设置。

3.2.4.4.2.1 ENETC

LS1028ARDB 只有一个 ENETC 控制器在使用，作为默认设置分配给内核 1。可以使用命令 `make menuconfig` 重新配置控制器。

请参见以下内容：

```
ARM architecture --->
[*] Enable baremetal
[*] Enable ENETC for baremetal
     (1)  Enetc1 is assigned to core1
     (1)  ENETC Controller numbers
```

3.2.4.4.2.2 I2C

本节介绍如何在 LS1028A 参考设计板上配置 I2C 总线。

LS1028ARDB 有 8 个 I2C 控制器，但只有控制器 0 用于 I2C 设备（例如 RTC、温度检测器），Linux（内核 0）将使用此控制器来实现某些功能（例如 RTC）。因此，以下代码仅用于演示如何在裸机端使能 I2C。

注：小心操作裸机端的 I2C 器件。

```
#define CONFIG_SYS_I2C_MXC_I2C1 /* enable I2C bus 0 */
#define CONFIG_SYS_I2C_MXC_I2C2 /* enable I2C bus 1 */
#define CONFIG_SYS_I2C_MXC_I2C3 /* enable I2C bus 2 */
#define CONFIG_SYS_I2C_MXC_I2C4 /* enable I2C bus 3 */

#define CONFIG_I2C_BUS_CORE_ID_SET
#define CONFIG_SYS_I2C_MXC_I2C0_COREID 1
```

`CONFIG_SYS_I2C_MXC_I2C0_COREID` 定义了运行 I2C 总线的 slave 内核。

由于 I2C 在裸机端使能了 DM 模式，因此没有自动代码对其进行测试。按照以下步骤在裸机端读取 RTC（0x51 地址，在总线 2 上）：

```
=> i2c bus
Bus 0:  i2c@2000000  (active 0)
   77: i2c-mux@77, offset len 1, flags 0
   57: generic_57, offset len 1, flags 0
Bus 1: i2c@2000000->i2c-mux@77->i2c@1
Bus 2: i2c@2000000->i2c-mux@77->i2c@3
   51: rtc@51, offset len 1, flags 0
Bus 3: i2c@2010000
Bus 4: i2c@2020000
Bus 5: i2c@2030000
Bus 6: i2c@2040000
Bus 7: i2c@2050000
Bus 8: i2c@2060000
Bus 9: i2c@2070000
=> i2c md 0x51 0
Error reading the chip: -121
=> i2c dev 2 Setting bus to 2
=> i2c md 0x51 0
0000: 04 00 36 03 12 15 02 12 20 80 80 80 80 80 00 c2       ..6.............
```

3.2.4.4.2.3 SAI

LS1028ARDB 仅有一个 SAI 模块在使用，作为默认设置分配给内核 1。可以使用命令 `make menuconfig` 重新配置 SAI 模块。

请参见以下内容：

```
Command line interface --->
  Misc commands --->
    [*] wavplayer

Device Drivers --->
  Sound support --->
    [*] Enable sound support
    [*]    Enable I2S support
    [*] Freescale sound
```

```
[*] Freescale sgtl5000 audio codec
[*] Freescale SAI module
```

3.2.4.4.2.3.1 裸机中的音频集成

对于音频功能，我们应该在裸机中添加 SAI 和 SGTL5000 驱动程序。

- 将 SAI 驱动源码添加到 *drivers/sound* 目录
- 将 SGTL5000 驱动源码添加到 *drivers/sound* 目录
- 将音频设备源代码添加到 *drivers/sound* 目录
- 将可以播放 wav 文件的命令添加到 *cmd* 目录
- 在 LS1028ARDB dts 文件中添加了对 SAI 和 sgtl5000 的支持

  在 fsl-ls1028a.dtsi 文件中：

```
sai4: sai@f130000 {
    #sound-dai-cells = <0>;
    compatible = "fsl,ls1028a-sai";
    reg = <0x0 0xf130000 0x0 0x10000>;
    status = "disabled";
};
```

  在 fsl-ls1028a-rdb.dts 文件中：

```
sound {
  compatible = "fsl,audio-sgtl5000";
  model = "ls1028a-sgtl5000";
  audio-cpu = <&sai4>;
  audio-codec = <&sgtl5000>;
  audio-routing =
    "LINE_IN", "Line In Jack",
    "MIC_IN", "Mic Jack",
    "Mic Jack", "Mic Bias",
    "Headphone Jack", "HP_OUT";
};


i2c@1 {
  #address-cells = <1>;
  #size-cells = <0>;
  reg = <0x1>;
  sgtl5000: codec@a {
      #sound-dai-cells = <0>;
      compatible = "fsl,sgtl5000";
      reg = <0xa>;
      VDDA-supply = <1800>;
      VDDIO-supply = <1800>;
      sys_mclk = <25000000>;
      sclk-strength = <3>;
  };
};
&sai4 {
  status = "okay";
};
```

- 将所有源代码添加到相应的 makefile 文件中。
- 向 ls1028ardb_sdcard_baremetal_defconfig 文件添加新的默认配置

3.2.4.4.3 LS1043ARDB 或 LS1046ARDB 板

以下各节介绍了 LS1043ARDB 或 LS1046ARDB 板用于实现支持的功能的硬件资源分配。

3.2.4.4.3.1 Linux DTS

移除 DTS 上的 cpu1、cpu2、cpu3 节点，并移除裸机使用过的所有设备。

3.2.4.4.3.2 内存配置

本节介绍 LS1043ARDB 或 LS1046ARDB 板的内存配置。

LS1043ARDB 或 LS1046ARDB 板具有 2GB 大小的 DDR。要使用裸机框架，请将 DDR 配置为三个分区：

- 内核 0 (Linux)为 512M
- 内核 1（裸机）为 256M
- 内核 2（裸机）为 256M
- 内核 3（裸机）为 256M，共享存储器为 256M。

可以在文件 `include/configs/ls1043ardb.h` 中定义配置。

```
#define CONFIG_SYS_DDR_SDRAM_SLAVE_SIZE (256 * 1024 * 1024)
#define CONFIG_SYS_DDR_SDRAM_MASTER_SIZE (512 * 1024 * 1024)
#define CONFIG_SYS_DDR_SDRAM_SHARE_RESERVE_SIZE (16 * 1024 * 1024)
#define CONFIG_SYS_DDR_SDRAM_SHARE_SIZE \ ((256 * 1024 * 1024)
- CONFIG_SYS_DDR_SDRAM_SHARE_RESERVE_SIZE)
```

注意

内存配置必须与内核 0 的 U-Boot 配置一致。

裸机的内存配置如下图所示。

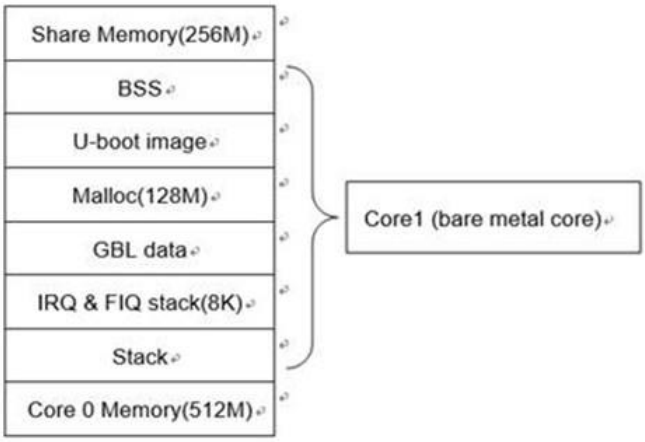


图 11.LS1043ARDB 或 LS1046ARDB 的内存配置

下表中 `malloc.h` 中包含的函数可用于在程序中分配或释放存储器。修改 `include/configs/ls1043a_common.h` 中的 `CONFIG_SYS_MALLOC_LEN`，以更改 malloc 的最大大小。

表 21.存储器 API 说明

API 名称（类型）	说明
void_t* malloc (size_t n)	分配存储器 ● “n”——分配块的长度 ● 返回指向新分配块的指针
void free (void *ptr)	释放 ptr 指向的存储器块（其中“ptr”是指向存储器块的指针）

LS1043ARDB（或 LS1046ARDB）的 GPIO 有四个 GPIO 控制器。您需要在文件 `ls1043/6a- rdb.dts` 中添加一个 GPIO 节点，以将 GPIO 资源分配给不同的内核。可以在文件 `arch/arm/dts/fsl- ls1043/6a-rdb.dts` 中进行配置。

3.2.4.4.3.3 GPIO

LS1043/6A 有四个 GPIO 控制器。您可以在文件 `ls1043/6a-rdb.dts` 中添加一个 GPIO 节点，以将 GPIO 资源分配给不同的内核。配置位于 `arch/arm/dts/fsl-ls1043/6a-rdb.dts` 中。使用以下命令添加 GPIO 节点：

```
&gpio2 {
     status = "okay";
};
```

3.2.4.4.3.4 I2C

本节介绍如何在 LS1028A、LS1043A 或 LS1046A 参考设计板上配置 I2C 总线。

LS1043ARDB（或 LS1028ARDB / LS1046ARDB）有四个 I2C 控制器。您可以使用以下命令通过 `ls1043ardb_config.h`（或 `ls1043ardb_config.h`）文件配置 I2C 总线：

```
// include/configs/ls1043ardb_config.h:
#define CONFIG_SYS_I2C_MXC_I2C1       /* enable I2C bus 0 */
#define CONFIG_SYS_I2C_MXC_I2C2       /* enable I2C bus 1 */
#define CONFIG_SYS_I2C_MXC_I2C3       /* enable I2C bus 2 */
#define CONFIG_SYS_I2C_MXC_I2C4       /* enable I2C bus 3 */
#define CONFIG_SYS_I2C_MXC_I2C0_COREID 1
#define CONFIG_SYS_I2C_MXC_I2C1_COREID 2
#define CONFIG_SYS_I2C_MXC_I2C2_COREID 3
#define CONFIG_SYS_I2C_MXC_I2C3_COREID 1
```

`CONFIG_SYS_I2C_MXC_I2C0_COREID` 定义了运行 I2C 总线的 slave 内核。

3.2.4.4.3.5 硬件中断

LS1043A 有 12 个 IRQ 作为外部 IO 信号连接到中断控制器。这 12 个 IRQ 可用于裸机内核。这些信号 IRQ0-IRQ11 的 ID 为：163、164、165、167、168、169、177、178、179、181、182 和 183。GIC 中断 API 在 `asm/interrupt-gic.h` 中定义。以下示例显示了如何注册硬件中断：

```
//register HW interrupt
void gic_irq_register(int irq_num, void (*irq_handle)(int));
void gic_set_target(u32 core_mask, unsigned long hw_irq);
void gic_set_type(unsigned long hw_irq);
```

3.2.4.4.3.6 QSPI

LS1046ARDB 有一个 QSPI flash 设备。要在 ls1046ardb_config.h 上配置 QSPI，请使用以下命令：

```
#define CONFIG_FSL_QSPI_COREID 1
```

在此处，`CONFIG_FSL_QSPI_COREID` 定义了运行该 QSPI 的 slave 内核。

3.2.4.4.3.7 IFC

LS1043A 和 LS1046A 具有 IFC 控制器。ls1043ardb 支持 Nor flash 和 NAND flash，ls1046ardb 只支持 NAND flash。

1. 通过禁用“ifc”节点在 Linux 内核中禁用 IFC：

```
&ifc {
     status = "disabled";
};
```

2. 输入 `BareMetal-Framework` 目录路径，然后执行以下命令：（默认情况下使能 IFC）

```
make menuconfig ARM architecture ---> [*] Enable baremetal [*] Enable IFC for baremetal (1) IFC
is assigned to that core
```

3.2.4.4.3.8 以太网

本节介绍 LS1043A 或 LS1046A 参考设计板的以太网配置设置。

LS1043A 或 LS1046A 只有一个 FMan，因此您应当在 Linux 中移除 DPAA 驱动程序。

1. 在 Linux 内核中禁用 DPAA 驱动程序：

```
   Device Drivers --->
      Staging drivers--->
        <> Freescale Datapath Queue and Buffer management
```

2. 进入 `BareMetal-Framework` 目录，然后执行以下命令：

```
make menuconfig ARM architecture ---> [*] Enable baremetal [*] Enable fman for baremetal (1)
FMAN1 is assigned to that core
```

通过将 `FMan1 is assigned to that core` 值（为默认配置）更改为 `core1`，将 FMan 配置给指定的内核。

3.2.4.4.3.9 USB

本节介绍 LS1043A 和 LS1046A 参考设计板的 USB 配置设置。

LS1043A 和 LS1046A 都有三个 DW3 USB 控制器，我们将它们作为默认设置分配给内核 1、内核 2 和内核 3。使用命令“make menuconfig”重新配置控制器。

```
ARM architecture --->
[*] Enable baremetal
[*]     Enable USB for baremetal
(1)     USB0 is assigned to core1
(2)     USB1 is assigned to core2
(3)     USB2 is assigned to core3
(3)     USB Controller numbers
```

3.2.4.4.3.10 PCI Express(PCIe)

本节介绍 LS1043A 和 LS1046A 参考设计板的 PCIe 配置设置。

LS1043A 和 LS1046A 都有三个 PCIe 控制器，我们将它们作为默认设置分配给内核 0、内核 1 和内核 2。使用命令“make menuconfig”重新配置控制器。

```
ARM architecture --->
[*] Enable baremetal
(0)     PCIe1 is assigned to core0
(1)     PCIe2 is assigned to core1
(2)     PCIe3 is assigned to core2
(3)     PCIe Controller numbers
```

3.2.4.4.4 LX2160ARDB 板

以下各节介绍了 LX2160ARDB 板用于实现支持的功能的硬件资源分配。

3.2.4.4.4.1 内存配置

本节介绍 LX2160ARDB 板的内存配置。

LX2160ARDB 板具有 16GB 大小的 DDR。要使用裸机框架，请将 DDR 配置为三个分区：

- 内核 0 (Linux)为 15G
- 内核 1 到内核 15（裸机）每个内核为 64M，共享存储器为 64M。

可以在文件 `include/configs/lx2160ardb_config.h` 中定义配置。

```
#define CONFIG_SYS_DDR_SDRAM_SLAVE_SIZE (64 * 1024 * 1024)
#define CONFIG_SYS_DDR_SDRAM_MASTER_SIZE (512 * 1024 * 1024)
#define CONFIG_SYS_DDR_SDRAM_SHARE_RESERVE_SIZE (16 * 1024 * 1024)
#define CONFIG_SYS_DDR_SDRAM_SHARE_SIZE \ ((64 * 1024 * 1024)
- CONFIG_SYS_DDR_SDRAM_SHARE_RESERVE_SIZE)
```

图 12.LX2160ARDB 的内存配置

下表中 `malloc.h` 中包含的函数可用于在程序中分配或释放存储器。修改 `include/configs/lx2160ardb_config.h` 中的 `CONFIG_SYS_MALLOC_LEN`，以更改 malloc 的最大大小。

表 22.存储器 API 说明

API 名称（类型）	说明
void_t* malloc (size_t n)	分配存储器 ● “n”——分配块的长度 ● 返回指向新分配块的指针
void free (void *ptr)	释放 ptr 指向的存储器块（其中“ptr”是指向存储器块的指针）

3.2.4.4.5 i.MX 8M Mini EVK 和 i.MX 8M Plus EVK 板

3.2.4.4.5.1 Linux DTS

在使用裸机时，用户应当从内核中移除裸机使用过的所有设备，例如：

```
&fec1 {
status = "disabled";
     };
&gpio5
       {
status = "disabled";
       };
&uart3 {
status = "disabled";
       };
```

3.2.4.4.5.2 内存配置

本节介绍 i.MX 8M Mini EVK 或 i.MX 8M Plus EVK 板的内存配置。

1. 该板具有 6 GB DDR 存储器。要使用裸机框架，请将 DDR 配置为五个分区：
   - 内核 0 (Linux)为 6016M
   - 内核 1（裸机）为 32M
   - 内核 2（裸机）为 32M
   - 内核 3（裸机）为 32M
   - 共享存储器为 32M。

可以在文件 `include/configs/imx8mm_evk.h.`或 `include/configs/imx8mp_evk.h.`中定义配置

```
#define CONFIG_SYS_DDR_SDRAM_SLAVE_RESERVE_SIZE (SZ_32M)
#define CONFIG_SYS_DDR_SDRAM_SHARE_RESERVE_SIZE (SZ_4M)
#define CONFIG_SYS_DDR_SDRAM_SLAVE_SIZE (SZ_32M)
```

2. 存储器保留

对于 IPI 数据传输，裸机需要在 master 内核和 slave 内核之间共享存储器，所以用户应当从 linux 内核中保留一些存储器，如以下 dts 文件所示：

```
reserved-memory { #address-cells = <2>; #size-cells = <2>; ranges; bm_reserved: baremetal@0x60000000
{ no-map; reg = <0 0x60000000 0 0x10000000>; }; };
```

3.2.4.4.5.3 GPIO

1. 连接 J1003 的引脚 7 和引脚 8。裸机中的 test_gpio 案例使用 J1003 的引脚 7 和引脚 8，因此要连接这两个引脚。
2. 在 slave 内核上启动裸机。如果 GPIO 工作正常，则会显示以下消息：

```
[ok]GPIO test ok
```

3. 从内核禁用设备。

对于 test_gpio 情况，使用 GPIO5_7（J1003 的引脚 8）和 GPIO5_8（J1003 的引脚 7）。这两个引脚复用为 UART3_TXD 和 UART3_CTS，因此应从内核禁用 GPIO5 和 UART3。

```
&gpio5 { status = "disabled"; }; &uart3 { status = "disabled"; };
```

3.2.4.4.5.4 以太网

本节介绍 i.MX 8M Mini EVK 或 i.MX 8M Plus EVK 板的以太网配置设置。

1. 从 dts 文件中禁用以太网卡：

```
&fec1 {
status = "disabled";
};
```

注意

1. i.MX 8M Mini EVK 只有一个 NIC，eth0(fec1)默认状态为禁用。如果用户在裸机中不使用 eth0，可以在内核 dts 文件中使能 fec1。
2. i.MX 8M Plus EVK 有两个 NIC，裸机的默认设置是 eth0，Linux 的默认设置是 eth1。

2. 使用以下命令确认裸机配置：

```
make menuconfig ARM architecture ---> [*] Enable baremetal [*] Enable NIC for baremetal (1) which core
that NIC is assigned to
```

通过修改 NIC 将该内核值（是默认配置）分配给内核 1，从而将 NIC 配置给指定的内核。

3.3 Jailhouse

3.3.1 概述

Jailhouse 是一个基于 Linux 的分区管理程序。它能够运行除 Linux 之外的裸机应用程序或（适应的）操作系统。为此，它配置硬件平台的 CPU 和设备虚拟化功能，使这些域（此处称为“单元”）不会以不可接受的方式相互干扰。

Jailhouse 针对简易性进行了优化，而非针对功能丰富程度。与 KVM 或 Xen 等基于 Linux 的全功能管理程序不同，Jailhouse 不支持 CPU、RAM 或设备等资源的过度使用。它不执行调度，仅在软件中虚拟化那些对平台至关重要且不能在硬件中分区的资源。

激活 Jailhouse 后，它就会运行裸机。这意味着它可以完全控制硬件并且不需要外部支持。但是，与其他裸机管理程序不同，它是由普通的 Linux 系统加载和配置。它的管理界面基于 Linux 基础架构。因此，您要首先启动 Linux，然后使能 Jailhouse，最后分离出部分系统资源并将它们分配给其他单元。

3.3.2 在 Inmate 中运行 PREEMPT_RT Linux

3.3.2.1 i.MX 8M Plus EVK

执行以下步骤：

1. 在 U-Boot 阶段执行 `run jh_mmcboot`。
2. 等待 Linux 操作系统启动并登录。
3. 使用以下命令复制内核镜像：

```
# cp /run/media/mmcblk1p1/Image/usr/share/jailhouse/inmates/kernel/
```

4. 执行非根 Linux 演示：

```
# cd /usr/share/jailhouse/scripts
# ./linux-demo-imx8mp.sh
```

5. 检查串行端口上结果：

```
[    0.000000] Booting Linux on physical CPU 0x0000000002 [0x410fd034]
[    0.000000] Linux version 5.10.9-rt24+g1d502fe65979 (oe-user@oe-host) (aarch64-fsl-linux-gcc (GCC) 10.2.0
[    0.000000] Machine model: LS1043A RDB Board
[    0.000000] earlycon: uart8250 at MMIO 0x00000000021c0600 (options '')
[    0.000000] printk: bootconsole [uart8250] enabled
[    0.000000] efi: UEFI not found.
[    0.000000] cma: Reserved 32 MiB at 0x00000000f2000000
[    0.000000] NUMA: No NUMA configuration found
[    0.000000] NUMA: Faking a node at [mem 0x00000000c0900000-0x00000000f3ffffff]
[    0.000000] NUMA: NODE_DATA [mem 0xf1e62a80-0xf1e64fff]
[    0.000000] Zone ranges:
[    0.000000]   DMA      [mem 0x00000000c0900000-0x00000000f3ffffff]
[    0.000000]   DMA32    empty
[    0.000000]   Normal   empty
[    0.000000] Movable zone start for each node
[    0.000000] Early memory node ranges
[    0.000000]   node   0: [mem 0x00000000c0900000-0x00000000f3ffffff]
[    0.000000] Initmem setup node 0 [mem 0x00000000c0900000-0x00000000f3ffffff]
[    0.000000] On node 0 totalpages: 210688
[    0.000000]   DMA zone: 3292 pages used for memmap
[    0.000000]   DMA zone: 0 pages reserved

...

[root@rt-edge ~] # uname -a
Linux rt-edge 5.10.9-rt24+g1d502fe65979 #1 SMP PREEMPT_RT Tue Jun 15 03:13:38 UTC 2021 aarch64 GNU/Linux
[root@rt-edge ~] #
```

注：如果程序因为 rootfs 错误而失败，我们可以使用以下命令更新 rootfs：

```
# rm -fr /run/media/mmcblk2p2/*
# cp -frd /usr /bin /etc /home /fat /lib /linuxrc /lost+found/ /media/ /mnt /opt /root / sbin
/run/media/mmcblk2p2/
```

6. 退出 Jailhouse。

3.3.2.2 LS1028ARDB

执行以下步骤，以在 LS1028ARDB 上的 Inmate 中运行 PREEMPT_RT Linux：

1. 在 U-Boot 阶段执行 `run jh_mmcboot`。
2. 等待 Linux 操作系统启动并登录。
3. 复制内核镜像：

```
# cp /boot/Image /usr/share/jailhouse/inmates/kernel/
```

4. 执行非根 Linux 演示：

```
# cd /usr/share/jailhouse/scripts
# ./linux-demo-ls1028ardb.sh
```

5. 退出 Jailhouse。

```
# ../tools/jailhouse disable
```

3.3.2.3 LS1046ARDB

执行以下步骤：

1. 在 U-Boot 阶段执行 `run jh_mmcboot`。
2. 等待 Linux 操作系统启动并登录。
3. 复制内核镜像：

```
# cp /boot/Image /usr/share/jailhouse/inmates/kernel/
```

4. 执行非根 Linux 演示：

```
# cd /usr/share/jailhouse/scripts
# ./linux-demo-ls1046ardb.sh
```

5. 退出 Jailhouse：

```
# ../tools/jailhouse disable
```

3.3.3 在 Inmate 中运行 Jailhouse 示例

3.3.3.1 i.MX 8M Plus EVK

1. 在 U-Boot 阶段执行 `run jh_mmcboot`
2. 等待 Linux 操作系统启动并登录。
3. 执行 GIC 演示。

```
# cd /usr/share/jailhouse/scripts
# ./gic-demo-imx8mp.sh
```

4. 检查串行端口上结果：

```
Initializing the GIC...
Initializing the timer...
Timer fired, jitter:   2039 ns, min:   2039 ns, max:   2039 ns
Timer fired, jitter:   1039 ns, min:   1039 ns, max:   2039 ns
Timer fired, jitter:    879 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    959 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:   1039 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    919 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    919 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    919 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:   1079 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    919 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    919 ns, min:    879 ns, max:   2039 ns
Timer fired, jitter:    959 ns, min:    879 ns, max:   2039 ns
```

5. 执行 UART 演示：

```
# ./uart-demo-imx8mp.sh
```

6. 检查串行端口上结果：

```
Hello 1 from cell!
Hello 2 from cell!
Hello 3 from cell!
Hello 4 from cell!
Hello 5 from cell!
Hello 6 from cell!
Hello 7 from cell!
Hello 8 from cell!
Hello 9 from cell!
Hello 10 from cell!
Hello 11 from cell!
Hello 12 from cell!
Hello 13 from cell!
Hello 14 from cell!
Hello 15 from cell!
Hello 16 from cell!
Hello 17 from cell!
Hello 18 from cell!
Hello 19 from cell!
Hello 20 from cell!
```

7. 退出 Jailhouse。

```
# ../tools/jailhouse disable
```

3.3.3.2 Inmate 中的 LS1028ARDB Jailhouse 示例

执行以下步骤，以在 Inmate 中运行 LS1028ARDB Jailhouse 示例：

1. 在 U-Boot 阶段执行 `run jh_mmcboot`。
2. 等待 Linux 操作系统启动并登录。
3. 执行 GIC 演示。

```
# cd /usr/share/jailhouse/scripts
# ./gic-demo-ls1028ardb.sh
```

4. 执行 UART 演示。

```
# ./uart-demo-ls1028ardb.sh
```

5. 执行 ivshmem 演示。

```
# ./ivshmem-demo-ls1028ardb.sh
```

> 注意
> 如果 ivshmem 案例失败，则重启板并再次测试案例。

检查第二个串行端口上的结果：

```
IVSHMEM: Found device at 00:00.0
IVSHMEM: bar0 is at 0x00000000ff000000
IVSHMEM: bar1 is at 0x00000000ff001000
IVSHMEM: ID is 1
IVSHMEM: max. peers is 1
IVSHMEM: state table is at 0x00000000c0500000
IVSHMEM: R/W section is at 0x00000000c0501000
IVSHMEM: input sections start at 0x00000000c050a000
IVSHMEM: output section is at 0x00000000c050c000
IVSHMEM: initialized device
state[0] = 1
state[1] = 2
state[2] = 0
rw[0] = 1
rw[1] = 0
rw[2] = -1001599800
in@0x0000 = 10
in@0x2000 = 0
in@0x4000 = 1758252876

IVSHMEM: got interrupt 0 (#1)
state[0] = 1
state[1] = 2
state[2] = 0
rw[0] = 1
rw[1] = 1
rw[2] = -1001599800
in@0x0000 = 10
in@0x2000 = 10
in@0x4000 = 1758252876
```

6. 退出 Jailhouse。

3.3.3.3 LS1046ARDB Jailhouse 示例

执行以下步骤，以在 LS1046ARDB 上的 Inmate 中运行 Jailhouse 示例：

1. 在 U-Boot 阶段执行 `run jh_mmcboot`。
2. 等待 Linux 操作系统启动并登录。
3. 执行 GIC 演示：

```
# cd /usr/share/jailhouse/scripts
# ./gic-demo-ls1046ardb.sh
```

4. 执行 UART 演示：

```
# ./uart-demo-ls1046ardb.sh
```

5. 执行 ivshmem 演示：

```
# ./ivshmem-demo-ls1046ardb.sh
```

6. 退出 Jailhouse。

```
# ../tools/jailhouse disable
```

第 4 章
实时网络服务

4.1 恩智浦平台上的时间敏感网络服务(TSN)

时间敏感网络服务(TSN)是对传统以太网网络的扩展，提供了一组与 IEEE 802.1 和 802.3 兼容的标准。这些扩展旨在解决标准以太网在一些领域的限制，包括从工业和汽车应用到直播音频和视频系统等。在传统以太网上运行的应用程序必须设计得非常稳定，才能承受诸如丢包、延迟甚至重新排序等极端情况。TSN 旨在为拥塞情况下的确定性延迟和丢包提供保证。因此，它允许关键和非关键流量聚合在同一网络中。

本章介绍在 i.MX 8M Plus、LS1028ARDB 和 LS1021A-TSN 板上实现 TSN 功能的过程和用例。

4.1.1 TSN 硬件能力

表 23.不同平台上的 TSN 硬件能力

平台	802.1Qbv（计划流量增强）	802.1Qbu 和 802.3br（帧抢占）	802.1Qav（基于信用的整形器）	802.1AS（精确时间协议）	802.1CB（提升可靠性的帧复制和消除）	802.1Qci（逐流滤波和监管）
ENETC (LS1028a)	是	是	是	是	否	是
Felix 交换机 (LS1028a)	是	是	是	是	是	是
SJA1105 (LS1021a-TSN)	是	否	是	是	否	预标准
Stmac (i.MX 8M Plus)	是	是	是	是	否	否

4.1.2 TSN 配置

下表显示了不同平台上的 TSN 配置工具支持

表 24.不同平台上的 TSN 配置工具支持

平台	802.1Qbv（计划流量增强）	802.1Qbu 和 802.3br（帧抢占）	802.1Qav（基于信用的整形器）	802.1AS（精确时间协议）	802.1CB（提升可靠性的帧复制和消除）	802.1Qci（逐流滤波和监管）
ENETC (LS1028A)	tc-taprio tsntool	ethtool tsntool	tc-cbs tsntool	ptp4l	不适用	tc-flower tsntool
Felix 交换机 (LS1028A)	tc-taprio tsntool	ethtool tsntool	tc-cbs tsntool	ptp4l，GenAVB/TSN 协议栈	tsntool	tc-flower tsntool

表格接下页……

表 24.不同平台上的 TSN 配置工具支持（续）

平台	802.1Qbv（计划流量增强）	802.1Qbu 和 802.3br（帧抢占）	802.1Qav（基于信用的整形器）	802.1AS（精确时间协议）	802.1CB（提升可靠性的帧复制和消除）	802.1Qci（逐流滤波和监管）
SJA1105 (LS1021A-TSN)	tc-taprio	不适用	tc-cbs	ptp4l	不适用	tc-flower
Stmac (i.MX 8M Plus)	tc-taprio	ethtool	tc-cbs	ptp4l，GenAVB/TSN 协议栈	不适用	不适用

4.1.2.1 使用 Linux 流量控制(tc)

使用 Linux 流量控制(tc)时在内核中使能以下配置：

```
Symbol: NET_SCH_MQPRIO [=y] && NET_SCH_CBS [=y] && NET_SCH_TAPRIO [=y]
  [*] Networking support --->
   Networking options --->
     [*] QoS and/or fair queueing --->
         <*>    Credit Based Shaper (CBS)
         <*>    Time Aware Priority (taprio) Scheduler
         <*>    Multi-queue priority scheduler (MQPRIO)
     [*]   Actions --->
         <*>    Traffic Policing
         <*>    Generic actions
         <*>    Redirecting and Mirroring
         <*>    SKB Editing
         <*>    Vlan manipulation
         <*>    Frame gate entry list control tc action
```

在 IS1028A 平台上，需要设置 ENETC QoS 驱动程序以支持 tc 配置。

```
Symbol: FSL_ENETC_QOS [=y]
  Device Drivers--->
    [*]   Network device support --->
          [*]   Ethernet driver support --->
               [*]    Freescale devices
               [*]       ENETC hardware Time-sensitive Network support
```

1. 以下链接提供了使用 tc-taprio 设置 Qbv 的详细信息：

https://man7.org/linux/man-pages/man8/tc-taprio.8.html

2. 以下链接提供了使用 tc-cbs 设置 Qav 的详细信息：

https://man7.org/linux/man-pages/man8/tc-cbs.8.html

3. 以下链接提供了使用 tc-flower 设置 Qci 和 ACL 的详细信息：

https://man7.org/linux/man-pages/man8/tc-flower.8.html

4.1.2.2 Tsntool

Tsntool 是设置 TSN 端点和 TSN 交换机以太网端口的 TSN 能力的工具。该工具用于 LS1028a 平台，所以在 LS1028a 上使能 TSN、ENETC_TSN 和 MSCC_FELIX_SWITCH_TSN 来支持 tsntool 配置。

```
Symbol: TSN [=y]
  [*] Networking support --->
    Networking options --->
      [*] 802.1 Time-Sensitive Networking support

Symbol: ENETC_TSN [=y] && FSL_ENETC_PTP_CLOCK [=y] && FSL_ENETC_HW_TIMESTAMPING [=y]
  Device Drivers --->
    [*] Network device support --->
      [*] Ethernet driver support --->
        [*] Freescale devices
        <*>   ENETC PF driver
        <*>   ENETC VF driver
        -*-   ENETC MDIO driver
        <*>   ENETC PTP clock driver
        [*]   ENETC hardware timestamping support
        [*]   TSN Support for NXP ENETC driver

Symbol: MSCC_FELIX_SWITCH_TSN [=y]
  Device Drivers --->
    [*] Network device support --->
      Distributed Switch Architecture drivers --->
        <*> Ocelot / Felix Ethernet switch support --->
        <*>   TSN on FELIX switch driver
```

在内核中使能 PKTGEN 以使用 pktgen 进行测试，

```
Symbol: NET_PKTGEN [=y]
  [*] Networking support --->
    Networking options --->
      Network testing --->
        <*> Packet Generator (USE WITH CAUTION)
```

有关详细信息，请参见《Tsntool 用户手册》。

4.1.2.2.1 Tsntool 用户手册

Tsntool 是设置 TSN 端点和 TSN 交换机以太网端口的 TSN 能力的工具。本文档介绍了如何将 tsntool 用于恩智浦的 LS1028ARDB 硬件平台。

注意

Tsntool 仅支持 LS1028ARDB 平台。

4.1.2.2.1.1 获取源代码

tsntool 代码的 Github 如下所示。

https://source.codeaurora.org/external/qoriq/qoriq-components/tsntool/

4.1.2.2.1.2 tsn 工具命令

下表列出了 TSN 工具命令及其说明。

表 25.TSN 工具命令及其说明

命令	说明
help	列出命令支持
version	显示软件版本
verbose	调试 tsntool 的开/关
quit	退出提示模式
qbvset	设置<ifname>的时间门控调度配置
qbvget	获取<ifname>的时间调度条目
cbstreamidset	设置流标识表
cbstreamidget	获取流标识表和计数器
qcisfiset	设置流滤波器实例
qcisfiget	获取流滤波器实例
qcisgiset	设置流门控实例
qcisgiget	获取流门控实例
qcisficounterget	获取流滤波器计数器
qcifmiset	设置流量计量实例
qcifmiget	获取流量计量实例
cbsset	设置 TC 基于信用的整形器配置
cbsget	获取 TC 基于信用的整形器状态
qbuset	设置一个 8 位向量，显示可抢占的流量类别
qbugetstatus	不支持
tsdset	不支持
tsdget	不支持
ctset	设置直通队列状态（特定于 ls1028 交换机）
cbgen	设置序列生成配置（特定于 ls1028 交换机）
cbrec	设置序列恢复配置（特定于 ls1028 交换机）
dscpset	将队列映射设置为 Qos 标记的 DSCP（特定于 ls1028 交换机）
sendpkt	不支持

表格接下页……

表 25.TSN 工具命令及其说明（续）

命令	说明
regtool	注册 PF 的 bar0 的读/写（特定于 ls1028 enetc）

ptptool	ptptool 获取/设置 ptp 时间戳。有用的命令：
	`#get ptp0 clock time`
	`ptptool -g`
	`#get ptp1 clock time`
	`ptptool -g -d /dev/ptp1`

dscpset	将队列映射设置为 QoS 标记的 DSCP（特定于 ls1028 交换机）
qcicapget	获取 qci 实例的最大能力
tsncapget	获取设备的 tsn 能力

4.1.2.2.1.3 Tsntool 命令和参数

本节列出了 tsntool 命令以及其可以使用的参数。

表 26. qbvset

参数<参数>	说明
`--device <ifname>`	诸如 `eno0/swp0` 的接口。

`--entryfile <filename>`	用于输入门控列表格式的文件脚本。它有以下参数：
	`#'NUMBER' 'GATE_VALUE' 'TIME_LONG'`
	• `NUMBER`：#“t”或“T”头。加上条目号。重复的条目号将导致错误。
	• `GATE_VALUE`：# 格式：xxxxxxxxb。# MSB 对应流量类别 7。LSB 对应于流量类别 0。# 位值 `0` 表示关闭，位值 1 表示打开。
	• `TIME_LONG`：# 纳秒。不要输入 0 时长。
	`t0 11101111b 10000 t1 11011111b 10000`
	注意
	必须设置 `Entryfile` 参数。如果不设置，会出现 vi 文本编辑器提示，“`require to input the gate list`”。

`--basetime <value>`	管理员基准时间 到目前为止，64 位十六进制值表示纳秒。 或值输入格式为：`Seconds.decimalSecond` 示例：`115.000125` 表示 115 秒和 125 µs。
`--cycletime <value>`	管理员周期时间

表格接下页……

表 26. qbvset（续）

参数<参数>	说明
--cycleextend <value>	管理员周期时间延长
--enable | --disable	• enable：使能此端口的 qbv。 • disable：禁用此端口的 qbv。 默认情况下，如果用户不提供任何输入，则该值设置为 enable。
--maxsdu <value>	queueMaxSDU
--initgate <value>	管理员门状态
--configchange	配置更改。默认设置为 1。
-- configchangetime <value>	配置更改时间

表 27. qbvget

参数<参数>	说明
--device <ifname>	诸如 eno0/swp0 的接口

表 28. cbstreamidset

参数<参数>	说明
--enable | --disable	• 使能：使能此索引的条目。 • 禁用：禁用此索引的条目。 默认情况下，如果没有使能或禁用输入，则此字段设置为 enable。
--index <value>	此控制器中的索引条目号。强制参数。 此值对应于交换机配置上的 tsnStreamIdHandle。
--device <string>	诸如 eno0/swp0 的接口
--streamhandle <value>	tsnStreamIdHandle
--infacoutport <value>	tsnStreamIdInFacOutputPortList
--outfacoutport <value>	tsnStreamIdOutFacOutputPortList
--infacinport <value>	tsnStreamIdInFacInputPortList
--outfacinport <value>	tsnStreamIdOutFacInputPortList
--nullstreamid | --sourcemacvid | --destmacvid | --ipstreamid	tsnStreamIdIdentificationType:

表格接下页……

表 28. cbstreamidset（续）

参数<参数>	说明
	• -nullstreamid：空流识别 • -sourcemacvid：源 MAC 和 VLAN 流识别 • -destmacvid：不支持 • -ipstreamid：不支持
--nulldmac <value>	tsnCpeNullDownDestMac
--nulltagged <value>	tsnCpeNullDownTagged
--nullvid <value>	tsnCpeNullDownVlan
--sourcemac <value>	tsnCpeSmacVlanDownSrcMac
--sourcetagged <value>	tsnCpeSmacVlanDownTagged
--sourcevid <value>	tsnCpeSmacVlanDownVlan

表 29. cbstreamidget

参数<参数>	说明
--device <ifname>	诸如 eno0/swp0 的接口
--index <value>	此控制器中的索引条目号。必须要有。

表 30. qcisfiset

参数<参数>	说明
--device <ifname>	诸如 eno0/swp0 的接口
--enable | --disable	• enable：使能该索引的条目 • disable：禁用该索引的条目 默认情况下，如果没有使能或禁用输入，则此字段设置为 enable。
--maxsdu <value>	最大 SDU 大小。
--flowmeterid <value>	流量计实例标识符索引号。
--index <value>	StreamFilterInstance。此控制器中的索引条目号。 此值对应于交换机配置上的 cbstreamidset 命令的 tsnStreamIdHandle。
--streamhandle <value>	StreamHandleSpec 该值对应于 cbstreamidset 命令的 tsnStreamIdHandle。

表格接下页……

表 30. qcisfiset（续）

参数<参数>	说明
--priority <value>	PrioritySpec
--gateid <value>	StreamGateInstanceID
--oversizeenable	StreamBlockedDueToOversizeFrameEnable
--oversize	StreamBlockedDueToOversizeFrame

表 31. qcisfiget

参数<参数>	说明
--device <ifname>	诸如 eno0/swp0 的接口
--index <value>	此控制器中的索引条目号。必须要有。

表 32. qcisgiset

参数<参数>	说明
--device <ifname>	诸如 eno0/swp0 的接口
--index <value>	此控制器中的索引条目号。必须要有。
--enable | --disable	• enable：使能该索引的条目。PSFP 门已启用 • disable：禁用此索引的条目。 默认情况下，如果没有使能或禁用输入，则此字段设置为 enable。
--configchange	configchange
--enblkinvrx	PSFPGateClosedDueToInvalidRxEnable
--blkinvrx	PSFPGateClosedDueToInvalidRx
--initgate	PSFPAdminGateStates
--initipv	AdminIPV
--cycletime	默认未设置。通过 gatelistfile 获取。
--cycletimeext	PSFPAdminCycleTimeExtension
--basetime	PSFPAdminBaseTime 到目前为止，64 位十六进制值表示纳秒。 或值输入格式为：Seconds.decimalSecond 示例：115.000125 表示 115 秒和 125 μs。

表格接下页……

表 32. qcisgiset（续）

参数<参数>	说明
--gatelistfile	PSFPAdminControlList。文件输入门列表：'NUMBER' 'GATE_VALUE' 'IPV' 'TIME_LONG' 'OCTET_MAX' • NUMBER：#“t”或“T”头。加上条目号。重复的条目号将导致错误。 • GATE_VALUE：格式：xb：MSB 对应于流量类别 7。LSB 对应于流量类别 0。位值 0 表示关闭，位值 1 表示打开。 • IPV：# 0~7 • TIME_LONG：以纳秒为单位。不要输入 0 时长。 • OCTET_MAX：允许通过门的最大八位字节数。如果为零，则没有最大值。 示例： `t0 1b -1 50000 10`

表 33. qcisgiget

参数<参数>	说明
--device <ifname>	诸如 eno0/swp0 的接口
--index <value>	此控制器中的索引条目号。必须要有。

表 34. qcifmiset

参数<参数>	说明
--device <ifname>	诸如 eno0/swp0 的接口
--index <value>	此控制器中的索引条目号。必须要有。
--disable	如果没有设置禁用，则要设置使能。
--cir <value>	cir. kbit/s。
--cbs <value>	cbs.八位字节。
--eir <value>	eir.kbit/s。
--ebs <value>	ebs.八位字节。
--cf	cf.双标志。
--cm	cm.颜色模式。
--dropyellow	丢弃黄色。
--markred_enable	标记红色使能。

表格接下页……

表 34. qcifmiset（续）

参数<参数>	说明
--markred	标记红色。

表 35. qcifmiget 参数

参数<参数>	说明
--device <ifname>	诸如 eno0/swp0 的接口
--index <value>	此控制器中的索引条目号。必须要有。

表 36. qbuset 参数

参数<参数>	说明
--device <ifname>	诸如 eno0/swp0 的接口
--preemptable <value>	8 位十六进制值。示例：0xfe MS 位对应于流量类别 7。 流量类别 0 的 LS 位。位值 0 表示快速。位值为 1 表示可抢占。

表 37. cbsset 命令

参数<参数>	说明
--device <ifname>	诸如 eno0/swp0 的接口
--tc <value>	流量类别编号。
--percentage <value>	设置 tc 限制的百分比。
--all <tc-percent:tc-percent...>	不支持。

表 38. cbsget

参数<参数>	说明
--device <ifname>	诸如 eno0/swp0 的接口
--tc <value>	流量类别编号。

表 39. regtool

参数<参数>	说明
Usage: regtool { pf number } { offset } [data]	pf number：要操作的 pci 资源的 pf 编号
	offset：要操作的 pci 存储器区域的偏移量
	data：要写入的数据

表 40. ctset

参数<参数>	说明
--device <ifname>	诸如 swp0 的接口
--queue_stat <value>	指定必须在直通工作模式下处理哪些优先级队列。位 0 对应优先级 0，位 1 对应优先级 1，依此类推。

表 41. cbgen

参数<参数>	说明
--device <ifname>	诸如 swp0 的接口
--index <value>	此控制器中的索引条目号。必须要有。 该值对应于 cbstreamidset 命令的 tsnStreamIdHandle。
--iport_mask <value>	INPUT_PORT_MASK：如果数据包来自属于此端口掩码的输入端口，则它是一个已知流，并且可以应用序列生成参数
--split_mask <value>	SPLIT_MASK：用于添加冗余路径（或端口）的端口掩码。如果为流使能了拆分(STREAM_SPLIT)。这与转发引擎确定的最终端口掩码进行“或”运算。
--seq_len <value>	SEQ_SPACE_LOG2：最小值为 1，最大值为 28。 tsnSeqGenSpace = 2**SEQ_SPACE_LOG2 例如，如果该值为 12，则有效序列号为 0x0 到 0xFFF。
--seq_num <value>	GEN_REC_SEQ_NUM：用于传递给 SEQ_GEN 函数的传出数据包的序列号。 注：在 RED_TAG 中仅发送低 16 位。

表 42. cbrec

参数<参数>	说明
--device <ifname>	诸如 swp0 的接口
--index <value>	此控制器中的索引条目号。必须要有。

表格接下页……

表 42. cbrec（续）

参数<参数>	说明
	该值对应于 cbstreamidset 命令的 tsnStreamIdHandle。
--seq_len <value>	SEQ_SPACE_LOG2：最小值为 1，最大值为 28。 tsnSeqRecSeqSpace = 2**SEQ_REC_SPACE_LOG2 例如，如果该值为 12，则有效序列号为 0x0 到 0xFFF。
--his_len <value>	SEQ_HISTORY_LEN：请参见 SEQ_HISTORY，最小值 1 和最大值 32。
--rtag_pop_en	REDTAG_POP：如果为 True，则重写器会弹出冗余标记。

表 43. dscpset

参数<参数>	说明
--device <ifname>	诸如 swp0 的接口
--disable	对帧流量类禁用 DSCP。
--index	DSCP 值
--cos	映射到的队列的优先级编号
--dpl	映射到的丢弃级别

表 44. qcicapget

参数<参数>	说明
--device <ifname>	诸如 swp0 的接口

表 45. tsncapget

参数<参数>	说明
--device <ifname>	诸如 swp0 的接口

4.1.2.2.1.4 输入提示

在提供命令输入的同时，用户可以使用以下快捷键来加快输入：

- 当用户输入命令时，使用 **TAB** 键帮助列出相关命令。

  例如：

  ```
  tsntool> qbv
  ```

  然后按 **TAB** 键，获取所有相关的 qbv*启动命令。

  如果只有一个选项，它会自动填充为整个命令。
- 当用户输入参数时，如果用户不记得参数名称。用户只需输入“--”，然后按 **TAB** 键即可。它显示所有参数。

如果用户输入了一半参数名称，按 **TAB** 键会列出所有相关名称。

- 历史记录：按向上箭头“↑”。用户将获得命令历史记录并可以重复使用该命令。

4.1.2.2.1.5 非交互模式

Tsntool 也支持非交互模式。

例如：

在交互模式下：

```
tsntool> qbuset --device eno0 --preemptable 0xfe
```

在非交互模式下：

```
tsntool qbuset --device eno0 --preemptable 0xfe
```

4.1.2.3 使用 NETCONF/YANG 进行远程配置

1.概述

NETCONF 协议定义了一种用于设备管理和配置检索和修改的机制。它使客户端能够通过使用远程过程调用(RPC)范式和公开设备（服务器）功能的系统来适应任何网络设备的特定功能。

YANG 是一种基于标准的可扩展分层数据建模语言。YANG 用于对 NETCONF 操作、RPC 和服务器事件通知使用的配置和状态数据进行建模。

2.支持实时边缘中的不同平台

TSN 分流	实时边缘			
	LS1028		SJA1105	i.MX 8M Plus
	libtsn	tc	tc	tc
802.1Qbv （时间感知整形器）	是	是	是	是
802.1Qbu/802.3br （帧抢占）	是	是	不适用	是
802.1Qav （基于信用的整形器）	-	-	-	-
802.1CB （提升可靠性的帧复制和消除）	-	-	不适用	不适用
802.1Qci （逐流过滤和监管）	是	是	是	不适用
IP 配置	是	是	是	是
MAC 配置	是	是	是	是
VLAN 配置	是	是	是	是

3.安装和配置

Netopeer 是一套基于 `libnetconf` 库构建的 NETCONF 工具。`sysrepo-tsn` (https://github.com/real-time-edge-sw/real-time-edge-sysrepo.git)帮助配置 TSN 功能，包括 `Qbv`、 `Qbu`、`Qci`，并通过网络进行流识别，无需登录设备。通过 Netopeer 配置 TSN 功能的详细信息，请参见 NETCONF/YANG。`tsn` 的一些应用场景，请参见应用场景。

4.1.2.4 基于网络的配置

4.1.2.4.1 设置网络服务器

网络 UI 允许远程控制 YANG 模型并通过网络套接字获取设备信息。此演示已添加到文件夹 tsntool/demos/cnc/中的 tsntool。

在 PC 端启动网络服务器有两种方法：docker 或者逐步安装。

在 PC 端启动网络服务器的简单方法是只需在 Ubuntu PC 上运行命令（安装 docker 支持）：

```
docker run --net=host -t -i liupoer/cncdemo:v2.0 /bin/bash /etc/rc.d/rc.local
```

然后直接跳到第 8 步，直接在板上启动代理。

如果用户想逐步设置网络服务器，只需按照以下各个步骤：

1. 安装相关库：假设用户在 Centos PC 或 Ubuntu PC 上安装演示作为网络服务器。CNC 演示需要 python3 和相关库：pyang、libnetconf 和 libssh。

   对于 Ubuntu：

   ```
   $ sudo apt install -y libtool python-argparse libtool-bin python-sphinx libffi-dev
   $ sudo apt install -y libxslt1-dev libcurl4-openssl-dev xsltproc python-setuptools
   $ sudo apt install -y zlib1g-dev libssl-dev python-libxml2libaugeas-dev
   $ sudo apt install -y libreadline-dev python-dev pkg-configlibxml2-dev
   $ sudo apt install -y cmake openssh-server
   $ sudo apt install -y python3-sphinx python3-setuptoolspython3-libxml2
   $ sudo apt install -y python3-pip python3-dev python3-flaskpython3-pexpect
   $ sudo apt install -y libnss-mdns avahi-utils
   $ pip3 install flask-restful
   $ pip3 install websockets
   ```

   对于 Centos 7.2：

   ```
   $ sudo yum install libxml2-devel libxslt-devel openssl-devel libgcrypt dbus-devel
   $ sudo yum install doxygen libevent readline.x86_64ncurses-libs.x86_64
   $ sudo yum install ncurses-devel.x86_64 libssh.x86_64libssh2-devel.x86_64
   $ sudo yum install libssh2.x86_64 libssh2-devel.x86_64
   $ sudo yum install nss-mdns avahi avahi-tools
   ```

2. 安装 pyang

   ```
   $ git clone https://github.com/mbj4668/pyang.git
   $ cd pyang
   $ git checkout b92b17718de53758c4c8a05b6818ea66fc0cd4d8 -bfornetconf1
   $ sudo python setup.py install
   ```

3. .安装 libssh：

   ```
   $ git clonehttps://git.libssh.org/projects/libssh.git
   $ cd libssh
   $ git checkout fe18ef279881b65434e3e44fc4743e4b1c7cb891 -bfornetconf1
   $ mkdir build; cd build/
   $ cmake ..
   ```

```
$ make
$ sudo make install
```

注意

在低于 16.04 版本的 Ubuntu 上安装 libssh 存在版本问题。apt-get 安装 libssh 可获得 0.6.4 版本。但 libnetconf 需要 0.7.3 或更高版本。删除默认版本，下载源代码并手动安装来重新安装。

4. 安装 libnetconf：

```
$ git clone https://github.com/CESNET/libnetconf.git
$ cd libnetconf
$ git checkout 8e934324e4b1e0ba6077b537e55636e1d7c85aed -bfornetconf1
$ autoreconf --force --install
$ ./configure
$ make
$ sudo make install
```

5. 在网络服务器 PC 上获取 tsntool 源代码

```
git clonehttps://source.codeaurora.org/external/qoriq/qoriq-components/tsntool cd
tsntool/demos/cnc/
```

6. 安装 python 库：

   在以下命令段中，

   - PATH-to-libnetconf 是 libnetconf 源代码的路径。
   - PATH-to-tsntool 是 tsntool 源代码的路径。

```
$ cd PATH-to-libnetconf/
```

   libnetconf 需要基于以下提交点添加两个补丁来修复演示 python 支持。

   确保提交 ID 为 313fdadd15427f7287801b92fe81ff84c08dd970。

```
$ git checkout 313fdadd15427f7287801b92fe81ff84c08dd970 -bcnc-server
$ cp PATH-to-tsntool/demos/cnc/*patch .
$ git am0001-lnctool-to-make-install-transapi-yang-model-proper.patch
$ git am0002-automatic-python3-authorizing-with-root-password-non.patch
$ cd PATH-to-libnetconf/python
$ python3 setup.py build; sudo python3 setup.py install
```

注意

如果重建 python 库，用户需要在重建之前通过命令 rm build -rf 删除构建文件夹。在实时边缘支持的板上，需要 avahi-daemon 和 netopeer 服务器。请记住还要添加在板上运行的 netopeer2-server。

7. 要在网络服务器 PC 上启动网络服务器，请在 shell 中将以下命令输入到文件夹：`PATH-to-tsntool/demos/cnc/`中：

```
sudo python3 cnc.py
```

8. 在支持的板上启动拓扑代理服务器
   - 确保 netopeer2-server 在板上运行（拓扑发现不需要）。
   - 确保 lldpd 守护进程正在板上运行。
   - 确保 avahi 守护进程正在板上运行。

在板上启动拓扑服务器：

```
#If the hostname is not real-time-edge, change toreal-time-edge
avahi-set-host-name real-time-edge
cd /home/root/samples/cncdemo/
python3 topoagent.py
```

9. 使用网页浏览器跟踪设备拓扑和配置。在浏览器中输入网络服务器的 IP，端口为 8180。例如：

```
http://10.193.20.147:8180
```

注意

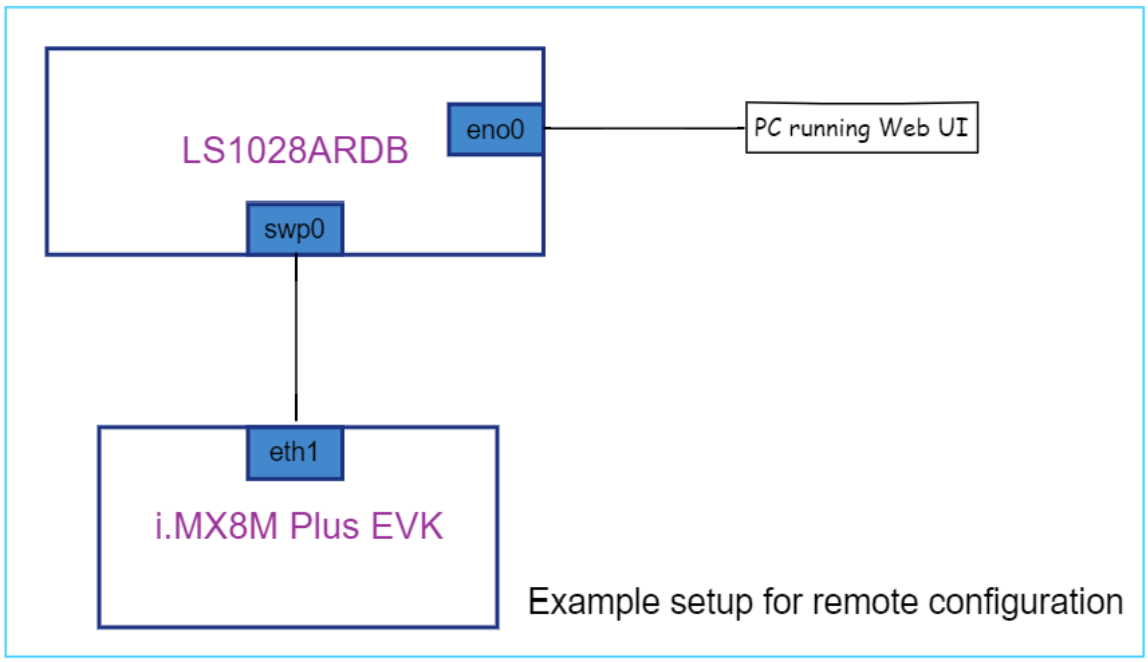
TSN 配置调试：

- 建议使用 tsntool 来跟踪板，检查真实的 tsn 配置，以进行比较。
- 另外，还建议跟踪 netopeer2-server 是否正在板上运行，以进行 tsn 配置。

网络 UI 的限制是：

- Centos PC 或 Ubuntu PC 上的设置服务器可能更具兼容性。
- 当前版本支持 Qbv、Qbu 和 Qci。
- 对于 Qci 设置，应在将流过滤设置为所需的 sysrepo 之前设置流门条目。否则，用户在没有流门 ID 链接的情况下设置流过滤会失败。
- 由于网桥可能会阻断探测帧，因此板和网络服务器 PC 必须位于同一 IP 域中。

4.1.2.4.2 远程配置



1.概述

网络 UI 允许远程控制 YANG 模型。用户可以连接 http 服务器，在网络 UI 上输入 TSN 参数，单击“是，确认”按钮将其发送到板。

2.用户界面

2.1 Qbv 配置

2.2 Qbu 配置

2.3 Qci 配置

在该界面中，用户可以选择“流识别”、“流过滤”、“流门”和“流量计量”的配置。

4.1.3 i.MX 8M Plus 上的 TSN

4.1.3.1 测试环境

在 i.MX 8M Pluse 平台上，只有 eth1 有 TSN 功能，将 eth1 连接到测试中心来测试 TSN 功能。

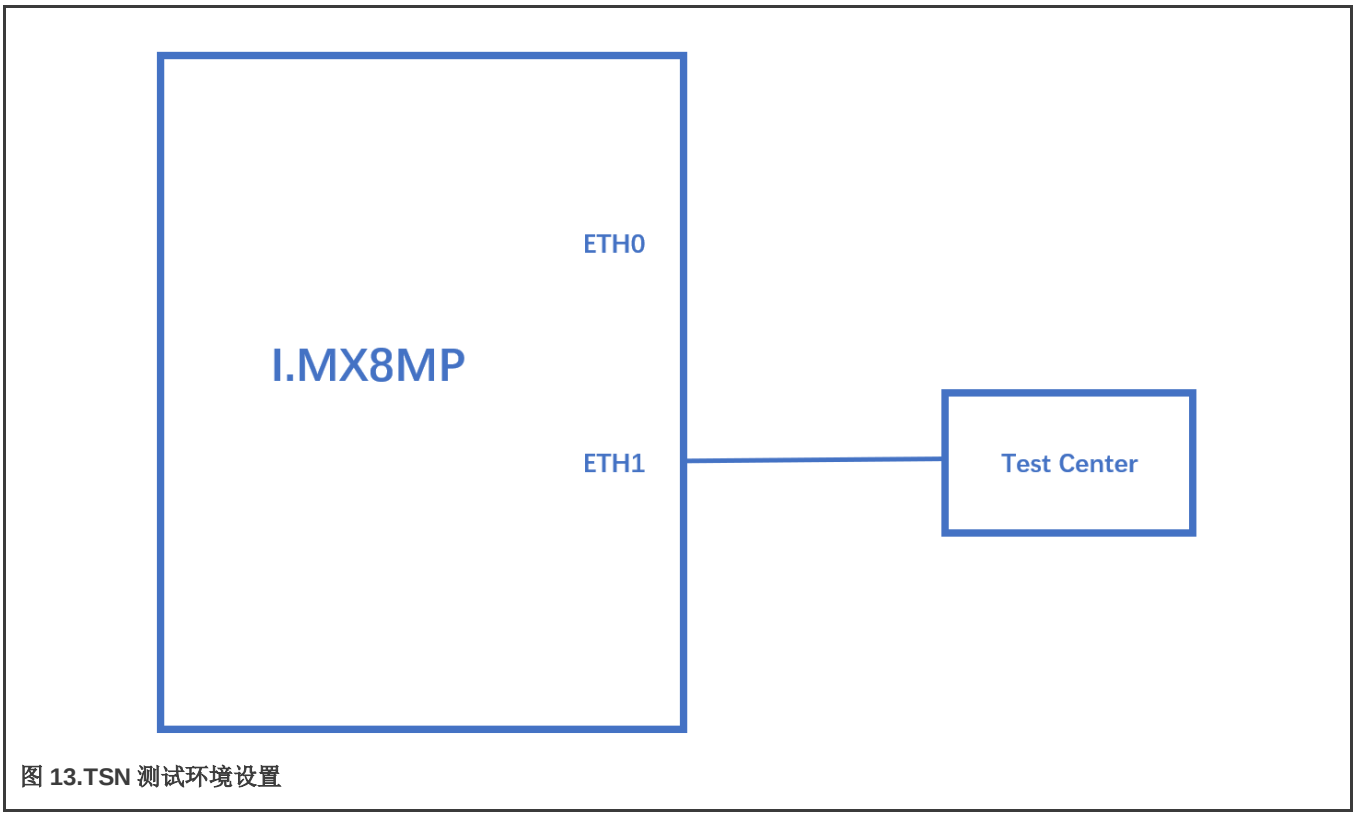


图 13.TSN 测试环境设置

注：测试中心是一个从 i.mx8mp 板的 eth1 捕获流的设备。我们使用 Spirent，它可以在 Qbu 测试用例中捕获可抢占的帧。

4.1.3.2 时钟同步

要在 dwcmac 接口上测试 1588 同步，请使用以下步骤：

1.将两块板上的 eth1 接口背靠背连接。

linux 启动日志如下：

```
...
pps pps0: new PPS source ptp0
...
```

2.配置 IP 地址

```
ifconfig eth1 192.168.3.1
```

3.检查 PTP 时钟和时间戳功能：

```
# ethtool -T eth1
Time stamping parameters for eth1:
Capabilities:
       hardware-transmit      (SOF_TIMESTAMPING_TX_HARDWARE)
       software-transmit      (SOF_TIMESTAMPING_TX_SOFTWARE)
       hardware-receive       (SOF_TIMESTAMPING_RX_HARDWARE)
       software-receive       (SOF_TIMESTAMPING_RX_SOFTWARE)
       software-system-clock  (SOF_TIMESTAMPING_SOFTWARE)
       hardware-raw-clock     (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 1
Hardware Transmit Timestamp Modes:
       off                    (HWTSTAMP_TX_OFF)
       on                     (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
       none                    (HWTSTAMP_FILTER_NONE)
       all                     (HWTSTAMP_FILTER_ALL)
       ptpv1-l4-event          (HWTSTAMP_FILTER_PTP_V1_L4_EVENT)
       ptpv1-l4-sync           (HWTSTAMP_FILTER_PTP_V1_L4_SYNC)
       ptpv1-l4-delay-req      (HWTSTAMP_FILTER_PTP_V1_L4_DELAY_REQ)
       ptpv2-l4-event          (HWTSTAMP_FILTER_PTP_V2_L4_EVENT)
       ptpv2-l4-sync           (HWTSTAMP_FILTER_PTP_V2_L4_SYNC)
       ptpv2-l4-delay-req      (HWTSTAMP_FILTER_PTP_V2_L4_DELAY_REQ)
       ptpv2-event             (HWTSTAMP_FILTER_PTP_V2_EVENT)
       ptpv2-sync              (HWTSTAMP_FILTER_PTP_V2_SYNC)
       ptpv2-delay-req         (HWTSTAMP_FILTER_PTP_V2_DELAY_REQ)
```

4.在两块板上运行 ptp4l：

```
ptp4l -i eth1 -p /dev/ptp1 -m -2
```

5.运行后，自动选择一块板作为 master，slave 板将打印同步消息。

6.对于 802.1AS 测试，只需使用 linuxptp 源中的配置文件 `gPTP.cfg`。改为在板上运行以下命令：

```
ptp4l -i eth1 -p /dev/ptp1 -f /etc/ptp4l_cfg/gPTP.cfg -m
```

或者通过以下命令使用 GenAVB/TSN 协议栈：`'avb.sh start'`。请注意，会自动使用配置文件`/etc/genavb/fgptp.cfg`。

注意

i.MX 8M Plus 当前的 dwmac 驱动程序(eth1)在打开网络设备时会初始化一些硬件功能，包括 PTP 初始化。在此之前，ethtool 查询、PTP 操作等操作可能无法正常工作。所以，解决方法是在“ifconfig eth1 up”之后才对 dwmac 的 eth1 和 PTP 进行操作。

4.1.3.3 Qbv

1.使能 `ptp device`，并获取当前 `ptp time`。

```
ptp4l -i eth1 -p /dev/ptp1 -m

#Get current time(seconds)
devmem2 0x30bf0b08
0x5E01F9B2
```

2.获取 2 分钟后的基准时间。

```
#Basetime = (currentime + 120) * 1000000000 = 1577187882000000000
```

3.设置时间表，在 100 µs 内打开队列 1，在 100 µs 内打开队列 2。

```
tc qdisc replace dev eth1 parent root handle 100 taprio \
     num_tc 5 map 0 1 2 3 4 queues 1@0 1@1 1@2 1@3 1@4  base-time 1577187882000000000 \
          sched-entry S  1 100000 \
          sched-entry S  2 100000 \
          sched-entry S  4 100000 flags 2
```

4.将两个流发送到队列 1 和队列 2。

```
/home/root/samples/pktgen/pktgen_twoqueue.sh -i eth1 -q 1 -s 1000 -n 0 -m 90:e2:ba:ff:ff:ff
```

5.在测试中心捕获流，会得到 100 µs 队列 1 帧（长度=1004）和 100 µs 队列 2 帧（长度=1504）。或者，如果以太网端口连接到另一块板上，则可以使用 Linux tcpdump 命令在该板上捕获帧，如下所示：

```
tcpdump -i eth0 -e -n -t -xx -c 10000 -w tsn.pcap
```

然后可以使用 Wireshark 分析主机 PC 上的 `pcap` 文件。

注意

- 每个 `tc taprio` 命令需要设置多个条目。
- 使用“`devmem2 0x30bf0c58`”获取 `Qbv status`，并检查 `Qbv status` 是否处于活动状态。请参见 `MTL_EST_Status` 寄存器。

4.1.3.4 Qbu

1.使用 ethtool 在 eth1 上使能 Qbu，将队列 2 设置为可抢占。

```
ethtool --set-frame-preemption eth1 preemptible-queues-mask 0x04 min-frag-size 60
```

注意

使能 Qbu 后，队列 0 始终是可抢占队列。

2.将两个流发送到队列 1 和队列 2。

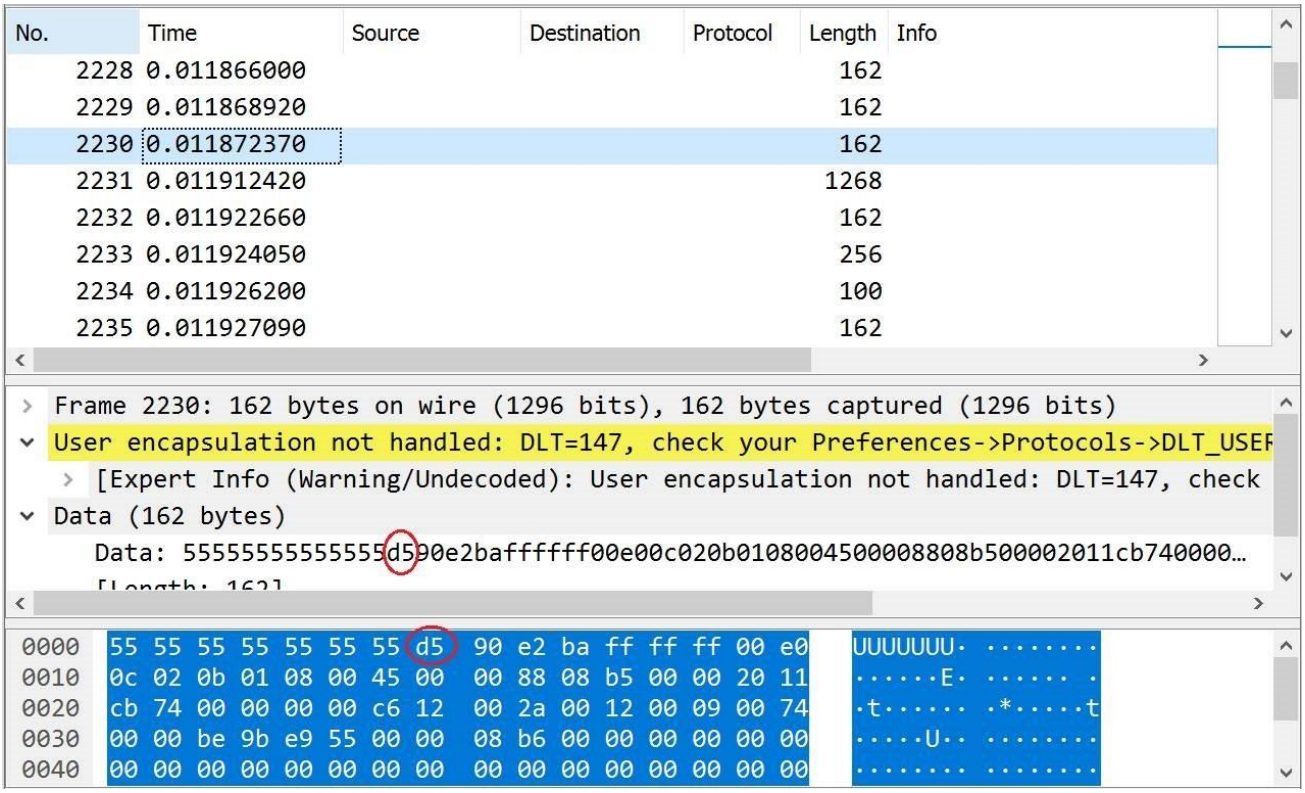
```
/home/root/samples/pktgen/pktgen_twoqueue.sh -i eth1 -q 1 -s 150 -n 0 -m 90:e2:ba:ff:ff:ff
```

3.在 Spirent 测试中心捕获 mPacket。用户可以观察到 Q2 帧被抢占为分片。

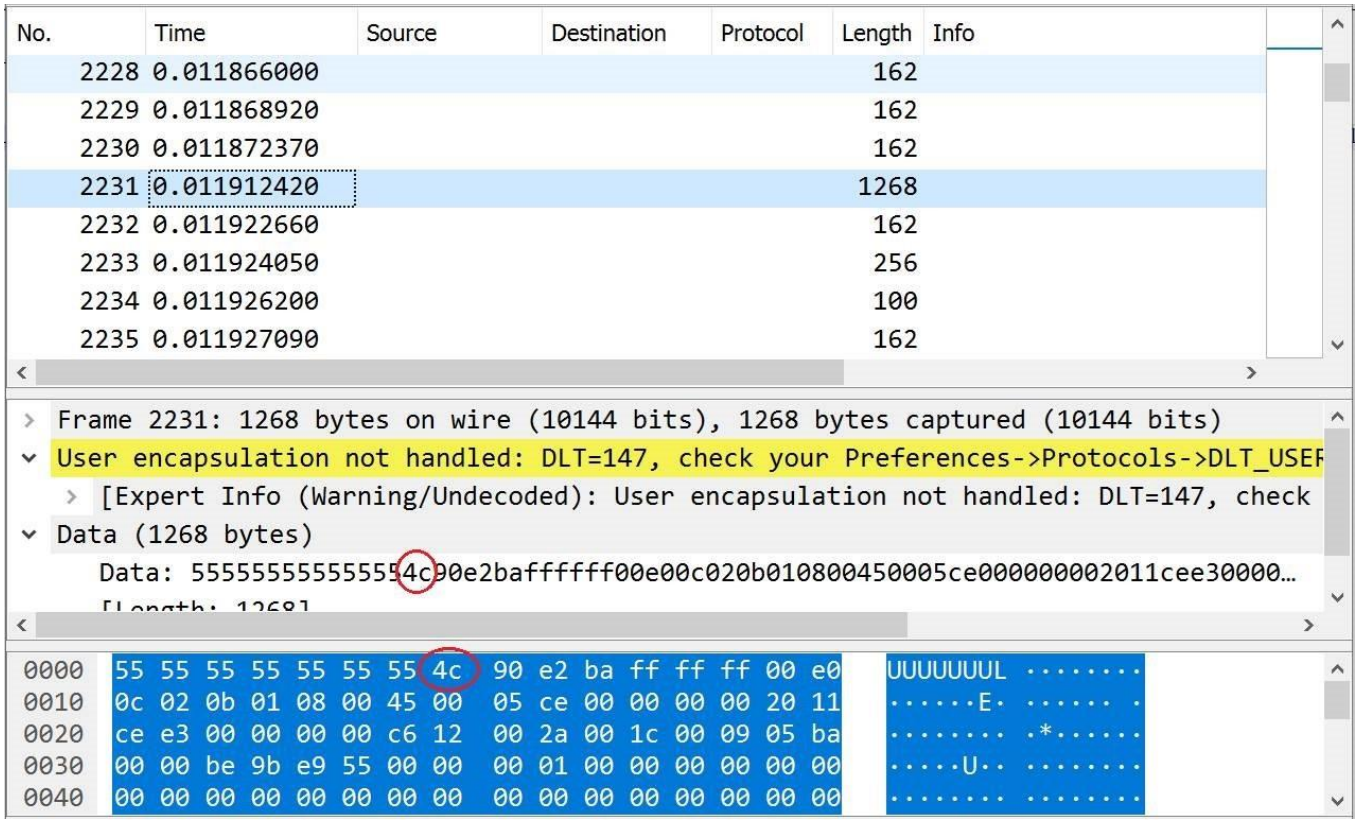
注意

Spirent 测试中心可以捕获 mPacket 的前导码。有关 mPacket 格式，请参见以太网 802.3-2018 的 IEEE 标准的 *第 99.3 节 “MAC 合并数据包(mPacket)”*。

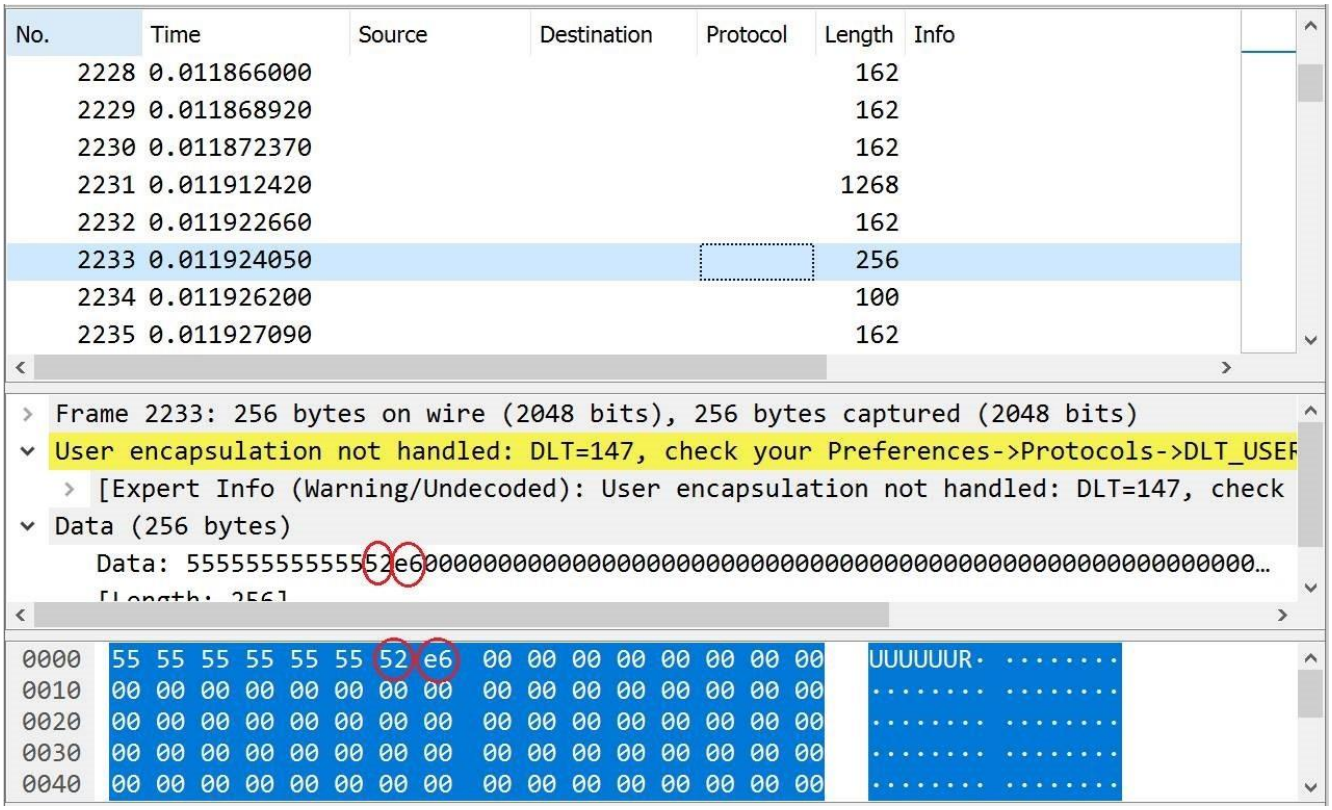
- 下面是一个示例 mPacket，其中包含一个快速数据包，其 SMD 值为 0xD5。



- 下面是一个示例 mPacket，其中包含可抢占数据包的初始分片，其 SMD-S1 值为 0x4C。



- 下面是一个示例 mPacket，其中包含可抢占数据包的连续分片，其 `SMD-C1` 值为 0x52，`frag_count` 值为 0xE6。



4.用户还可以检查以下计数器，以了解传输的分片数。

```
ethtool -S eth1 | grep "mmc_tx_fpe_fragment_cntr"
```

5.Qbu 与 Qbv 测试结合

一旦队列设置为可抢占队列，并且 Qbv 门控列表中的门打开/关闭无效，则该队列被视为始终“打开”。使用**保持/释放**控制所有可抢占队列。当 GCL 条目从**保持**设置为**释放**时，可抢占队列开始传输。当 GCL 条目从“释放”设置为“保持”时，可抢占队列被保持。

```
tc qdisc replace dev eth1 parent root handle 100 taprio \
  num_tc 5 map 0 1 2 3 4 queues 1@0 1@1 1@2 1@3 1@4  base-time 1577187882000000000 \
    sched-entry H  2 100000 \
    sched-entry R  4 100000 flags 2
```

4.1.3.5 Qav

1.设置队列映射句柄。

```
tc qdisc add dev eth1 root handle 1: mqprio num_tc 5 map 0 1 2 3 4
```

2.将队列 3 的带宽设置为 20Mbps

```
tc qdisc replace dev eth1 parent 1:4 cbs locredit -1470 hicredit 30 sendslope -980000 idleslope 20000
offload 1
```

3.将流发送到队列 3

```
/home/root/samples/pktgen/pktgen_sample01_simple.sh -i eth1 -q 3 -s 500 -n 3000
```

4.获取结果，带宽是 19Mbps。

```
WARN : Missing destination MAC address
WARN : Missing destination IP address
Running... ctrl^C to stop
Done
Result device: eth1
Params: count 3000  min_pkt_size: 500  max_pkt_size: 500
       frags: 0  delay: 0  clone_skb: 0  ifname: eth1
       flows: 0 flowlen: 0
       queue_map_min: 3  queue_map_max: 3
       dst_min: 198.18.0.42 dst_max:
       src_min:          src_max:
       src_mac: a6:85:82:fc:89:bf dst_mac: 02:5d:ae:ba:e0:00
       udp_src_min: 9  udp_src_max: 109  udp_dst_min: 9  udp_dst_max: 9
       src_mac_count: 0  dst_mac_count: 0
       Flags: UDPSRC_RND  NO_TIMESTAMP QUEUE_MAP_RND
Current:
       pkts-sofar: 3000  errors: 0
       started: 5631940023us stopped: 5632560030us idle: 79984us
       seq_num: 3001  cur_dst_mac_offset: 0  cur_src_mac_offset: 0
       cur_saddr: 0.0.0.0  cur_daddr: 198.18.0.42
       cur_udp_dst: 9  cur_udp_src: 41
       cur_queue_map: 3
       flows: 0
Result: OK: 620007(c540023+d79984) usec, 3000 (500byte,0frags)
  4838pps 19Mb/sec (19352000bps) errors: 0
```

5.将队列 4 的带宽设置为 40Mbps

```
tc qdisc replace dev eth1 parent 1:5 cbs locredit -1440 hicredit 60 sendslope -960000 idleslope 40000
offload 1
```

6.将流发送到队列 4 并获取结果。

```
/home/root/samples/pktgen/pktgen_sample01_simple.sh -i eth1 -q 4 -s 500 -n 3000
WARN : Missing destination MAC address
WARN : Missing destination IP address
Running... ctrl^C to stop
Done
Result device: eth1
Params: count 3000 min_pkt_size: 500 max_pkt_size: 500
     frags: 0 delay: 0 clone_skb: 0 ifname: eth1
     flows: 0 flowlen: 0
     queue_map_min: 4 queue_map_max: 4
     dst_min: 198.18.0.42 dst_max:
     src_min:    src_max:
     src_mac: a6:85:82:fc:89:bf dst_mac: 02:5d:ae:ba:e0:00
     udp_src_min: 9 udp_src_max: 109 udp_dst_min: 9 udp_dst_max: 9
     src_mac_count: 0 dst_mac_count: 0
     Flags: UDPSRC_RND NO_TIMESTAMP QUEUE_MAP_RND
Current:
     pkts-sofar: 3000 errors: 0
     started: 6113136017us stopped: 6113443758us idle: 38457us
     seq_num: 3001 cur_dst_mac_offset: 0 cur_src_mac_offset: 0
     cur_saddr: 0.0.0.0 cur_daddr: 198.18.0.42
     cur_udp_dst: 9 cur_udp_src: 17
     cur_queue_map: 4
     flows: 0
Result: OK: 307741(c269283+d38457) usec, 3000 (500byte,0frags)
     9748pps 38Mb/sec (38992000bps) errors: 0
```

7.将两个流发送到队列 3 和队列 4

```
/home/root/samples/pktgen/pktgen_twoqueue.sh -i eth1 -q 3 -s 1500 -n 0
```

8.在测试中心捕获流，帧按 1 个 Q3 帧和 2 个 Q4 帧排序

4.1.4 LS1028A 上的 TSN

tsntool 是在 LS1028ARDB 上配置 TSN 功能的应用配置工具。文件**/usr/bin/tsntool** 和**/usr/lib/libtsn.so** 位于 rootfs 中。运行 **tsntool** 以启动设置 shell。

4.1.4.1 ENETC 上的 TSN 配置

tsntool 是配置 TSN 功能的应用配置工具。用户可以在 rootfs 中找到文件`/usr/bin/tsntool` 和`/usr/lib/libtsn.so`。运行 `tsntool` 以启动设置 shell。以下部分描述了 ENETC 以太网驱动程序接口上的 TSN 配置示例。

在测试 ENETC TSN 测试案例之前，您必须使用以下命令使能 `mqprio`：

```
tc qdisc add dev eno0 root handle 1: mqprio num_tc 8 map 0 1 2 3 4 5 6 7 hw 1
```

4.1.4.1.1 时钟同步

要在 ENETC 接口上测试 1588 同步，请使用以下步骤：

1. 将两块板上的 ENETC 接口背靠背连接。（例如，eno0 到 eno0。）

linux 启动日志如下：

```
...
pps pps0: new PPS source ptp0
...
```

2. 检查 PTP 时钟和时间戳功能：

```
 # ethtool -T eno0
 Time stamping parameters for
 eno0: Capabilities:
     hardware-transmit           (SOF_TIMESTAMPING_TX_HARDWAR
     E) hardware-receive         (SOF_TIMESTAMPING_RX_HARDWAR
     E) hardware-raw-clock       (SOF_TIMESTAMPING_RAW_HARDWARE)
 PTP Hardware Clock: 0
 Hardware Transmit Timestamp Modes:
      off          (HWTSTAMP_TX_OFF)
      on           (HWTSTAMP_TX_ON)Hardware ReceiveFilterModes:
      none         (HWTSTAMP_FILTER_NONE)
      all          (HWTSTAMP_FILTER_ALL)
```

3. 配置 IP 地址，并在两块板上运行 `ptp4l`：

```
# ifconfig eno0 <ip_addr>
# ptp4l -i eno0 -p /dev/ptp0 -m
```

4. 运行后，自动选择一块板作为 master，slave 板将打印同步消息。
5. 对于 802.1AS 测试，只需使用 linuxptp 源中的配置文件 `gPTP.cfg`。改为在板上运行以下命令：

```
# ptp4l -i eno0 -p /dev/ptp0 -f /etc/ptp4l_cfg/gPTP.cfg -m
```

4.1.4.1.2 Qbv

此测试包括基本门关闭测试、基准时间测试和 `Qbv` 性能测试。这些将在后续几节中进行介绍。

4.1.4.1.2.1 基本门关闭

以下命令描述了关闭基本门的步骤：

```
cat > qbv0.txt << EOF
t0      00000000b      20000
EOF
```

```
#Explanation:
# 'NUMBER'     :       t0
# 'GATE_VALUE' :       00000000b
# 'TIME_LONG'  :       20000 ns
```

```
tsntool
tsntool> verbose
tsntool> qbvset --device eno0 --entryfile ./qbv0.txt
```

```
ethtool -S eno0
ping 192.168.0.2 -c 1  #Should not pass any frame since gates are all off.
```

4.1.4.1.2.2 基准时间测试

基于案例 1 `qbv1.txt` 门列表。

```
#create 1s gate
cat > qbv1.txt << EOF
t0 11111111b   10000
t1 00000000b   99990000
EOF

#ENETC Qbv basetime can be set any past time or future time.
#For the past time, hardware calculate by:
#       effective-base-time = base-time + N x cycle-time
#where N is the smallest integer number of cycles such that effective-base-time >= now.
#If you want a future time, you can get current time by:

tsntool> ptptool -g

#Below example shows basetime start at 260.666 s (start of 1 January 1970):

tsntool> qbvset --device eno0 --entryfile qbv1.txt --basetime 260.666
tsntool> qbvget --device eno0 #User can check configchange time
tsntool> regtool 0 0x11a10 #Check pending status, 0x1 means time gate is working

#Waiting to change state, ping remote computer
ping 192.168.0.2 -A -s 1000

#The reply time will be about 100 ms
```

由于 10000 ns 是封装尺寸 1250 B 的最大限制。

```
ping 192.168.0.2 -c 1 -s 1300 #frame should not pass
```

4.1.4.1.2.3 Qbv 性能测试

使用下图中所示的设置来测试 ENETC 端口 0 (MAC0)。

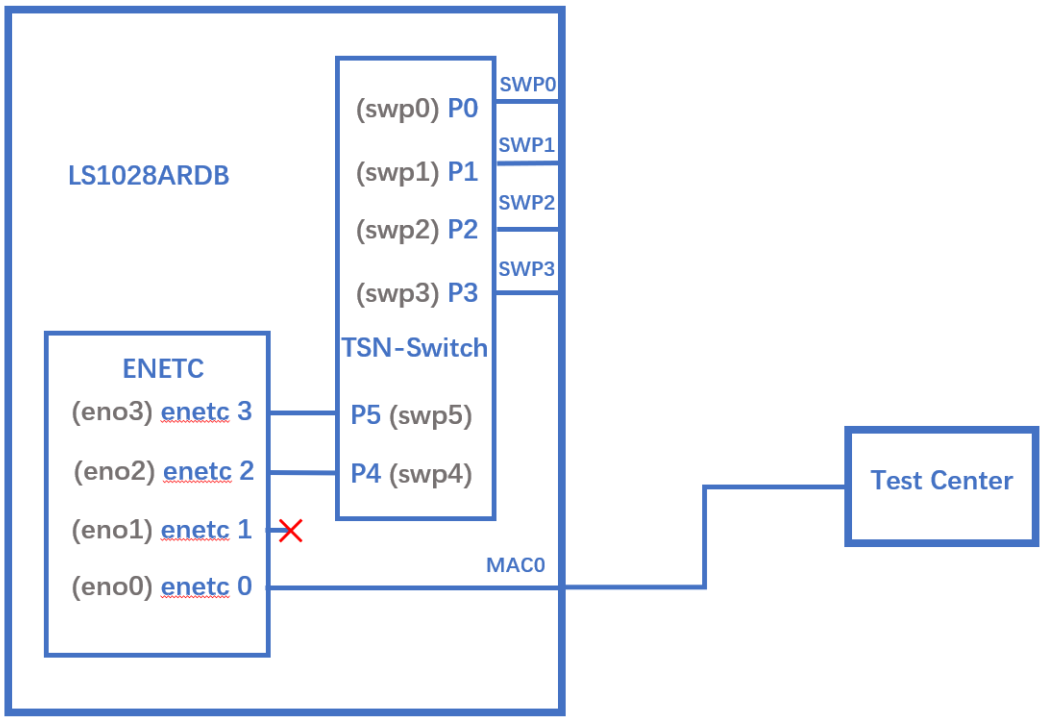


图 14.测试 ENETC 端口 0 的设置

注意

测试中心是一个从 LS1028ARDB 板的 `enetc0` 捕获流的设备。用户可以使用另一块板通过 `tcpdump` 命令捕获流，然后使用 Wireshark 对其进行分析。

```
cat > qbv5.txt << EOF
t0 11111111b 1000000 t1 00000000b 1000000
EOF
qbvset --device eno0 --entryfile qbv5.txt
/home/root/samples/pktgen/pktgen_twoqueue.sh -i eno0 -q 3 -n 0
#The stream would get about half line rate
```

4.1.4.1.2.4 使用 taprio Qdisc 设置 Qbv

LS1028ardb 也支持 taprio qdisc 设置 Qbv。下面是一个示例设置。

```
#Qbv test do not require the mqprio setting.
# If mqprio is enabled, try to disable it by below command:
tc qdisc del dev eno0 root handle 1: mqprio

# Enable the Qbv for ENETC eno0 port
# Below command set eno0 with gate 0x01, means queue 0 open, the other queues gate close.
tc qdisc replace dev eno0 parent root handle 100 taprio num_tc 8 map 0 1 2 3 4 5 6 7 queues 1@0 1@1 1@2
1@3 1@4 1@5 1@6 1@7 base-time 0 sched-entry S 01 300000 flags 0x2
# Ping through eno0 port should be ok

# Then close the gate queue 0.Open gate queue 1.The other queues gate close.
tc qdisc replace dev eno0 parent root handle 100 taprio num_tc 8 map 0 1 2 3 4 5 6 7 queues 1@0 1@1 1@2
1@3 1@4 1@5 1@6 1@7 base-time 0 sched-entry S 02 300000 flags 0x2
# Ping through eno0 port should be dropped
```

```
#Disable the Qbv for ENETC eno0 port as below
tc qdisc del dev eno0 parent root handle 100 taprio
```

4.1.4.1.3 Qbu

- 如果用户有两个 LS1028ARDB 板，则将两个 eno0 端口背靠背连接。在这种情况下，测试不需要设置交换机。用户可以省略步骤 2、3 和 4，只执行步骤 1、5 和 6。
- 如果用户只有一个板，用户可以通过连接 enetc 端口 MAC0——SWP0 来设置从 eno0 到交换机的帧路径。该设置使能交换机 SWP0 端口合并功能。然后 enetc eno0 可以显示抢占功能。使用下图所示的设置进行 Qbu 测试。

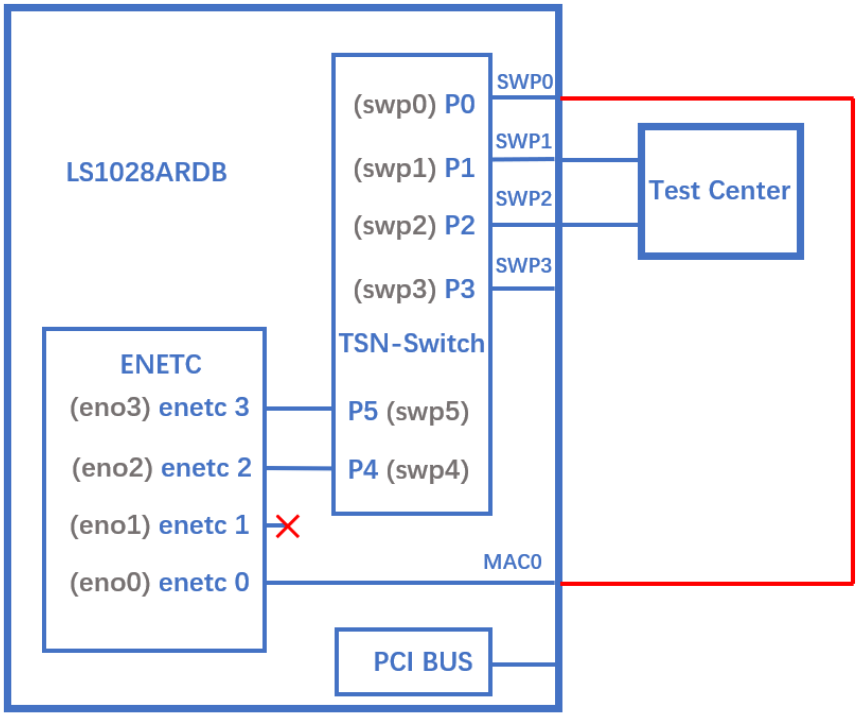


图 15.Qbu 测试

在连接 ENETC 端口 0 与 SWP0 之间的电缆之前，请设置交换机（参见交换机配置）并为 ENETC 端口 0 设置 IP。要确保 ENETC 端口 0 连接到 SWP0，请使用以下步骤：

1. 不要忘记为每个流量类别使能优先级：

```
tc qdisc add dev eno0 root handle 1: mqprio num_tc 8 map 0 1 2 3 4 5 6 7 hw1
```

2. 使用以下命令确保连接速度为 1 Gbps：

```
ethtool eno0
```

3. 如果不是 1 Gbps，请使用以下命令将其设置为 1 Gbps：

```
ethtool -s swp0 speed 1000 duplex full autoneg on
```

4. 将交换机设置为使能合并（或者用户可以连接到另一个板上的另一个合并功能端口）：

```
devmem2 0x1fc100048 w 0x111#DEV_GMII:MM_CONFIG:ENABLE_CONFIG
```

5. ENETC 端口设置集和帧抢占测试

```
ip link set eno0 address 90:e2:ba:ff:ff:ff
tsntool qbuset --device eno0 --preemptable 0xfe
/home/root/samples/pktgen/pktgen_twoqueue.sh -i eno0 -q 0 -s 100 -n 20000 -m 90:e2:ba:ff:ff:ff
```

pktgen 会在 TC0 和 TC1 上清除帧。

6. 检查 TX 合并计数器，如果它的值为非零值，则表明 Qbu 正在工作。

```
tsntool regtool 0 0x11f18
```

注意

0x11f18 计数合并帧数：

```
0x11f18 Port MAC Merge Fragment Count TX Register(MAC_MERGE_MMFCTXR)
```

LS1028ARDB 还支持 ethtool 设置抢占，例如：

```
ethtool --set-frame-preemption eno0 preemptible-queues-mask 0xfe
```

这意味着我们可以通过使用 TC0 传递快速 MAC，使用 TC1~TC7 传递可抢占 MAC，获得相同的结果。

4.1.4.1.4 QCI

使用以下作为背景设置：

- 设置 eno0 MAC 地址

```
ip link set eno0 address 10:00:80:00:00:00
```

例如，对面端口 MAC 地址 **99:aa:bb:cc:dd:ee** 作为帧提供者。

- 使用下图作为硬件设置。

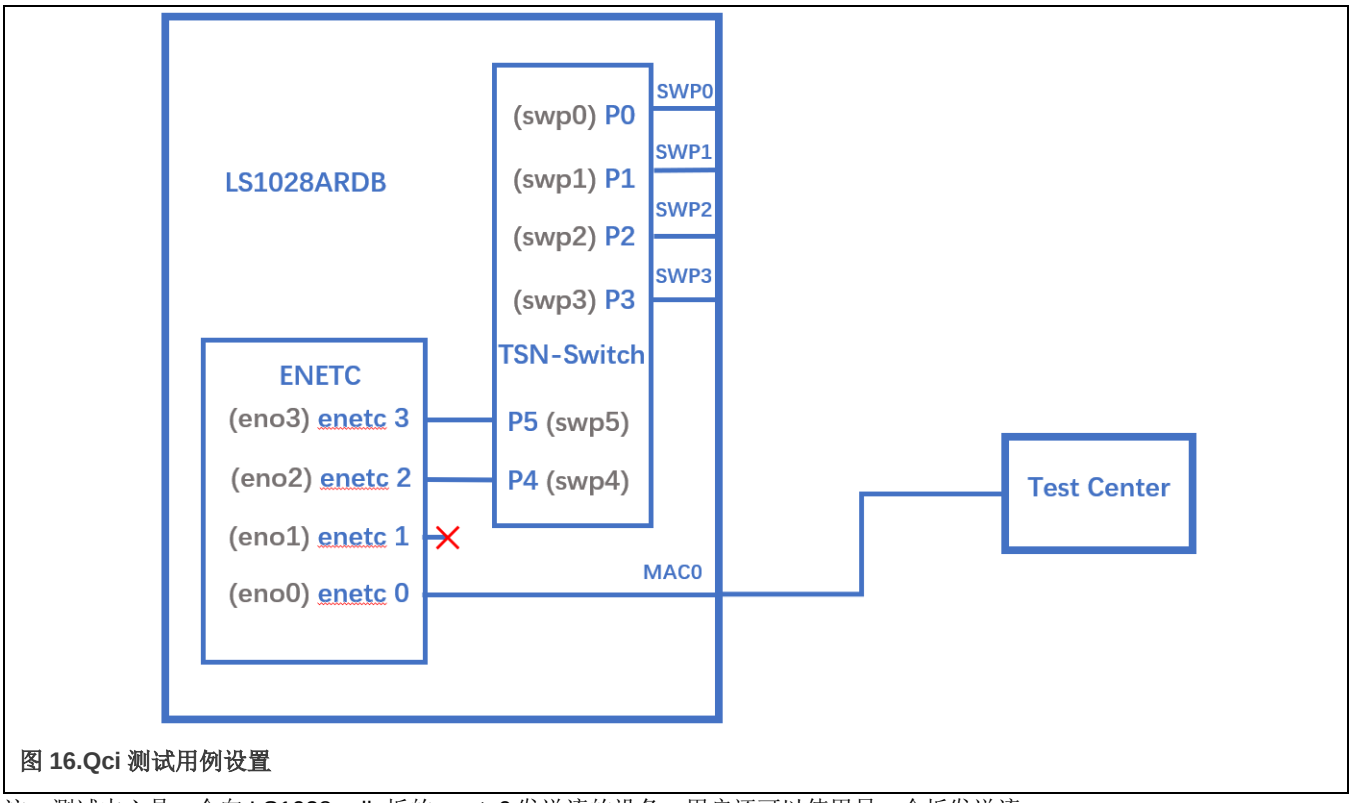


图 16.Qci 测试用例设置

注：测试中心是一个向 LS1028ardb 板的 enetc0 发送流的设备。用户还可以使用另一个板发送流。

4.1.4.1.4.1 测试 SFI 没有流句柄

Qci PSFP 可以用于没有流识别模块的流，即没有 MAC 地址和 vid 过滤的流。这种过滤设置总是设置更大的索引号流过滤条目。这些帧不会经过过滤，然后流入此流过滤条目。

下面的示例测试流过滤中没有流句柄，设置在流过滤条目索引 2 上，门流条目 ID 为 2。然后没有流识别帧将流入流过滤条目索引 2，然后通过门条目索引 2，如以下示例所示：

```
tsntool> qcisfiset --device eno0 --index 2 --gateid 2
```

- 流没有流句柄应当通过此过滤。

```
tsntool> qcisfiget --device eno0 --index 2
```

- 从对面设备端口发送帧（例如 ping）。

```
tsntool> qcisfiget --device eno0 --index 2
```

- 设置流门条目 2

```
tsntool> qcisgiset --device eno0 --index 2 --initgate 1
```

- 从对面设备端口发送帧。

```
tsntool> qcisfiget --device eno0 --index 2
```

- 设置流门条目 2，门永久关闭。

```
tsntool> qcisgiset --device eno0 --index 2 --initgate 0
```

- 从对面设备端口发送帧。

```
tsntool> qcisfiget --device eno0 --index 2

#The result should look like below:
 match pass gate_drop sdu_pass sdu_drop red
   1   0   1     1    0    0
```

4.1.4.1.4.2 测试空流识别条目

流识别模块中的空流识别意味着尝试作为目标 mac 地址和 vlan ID 过滤。

以下步骤显示了流识别条目索引 1 设置，过滤目标 MAC 地址为 10:00:80:00:00:00，vlan ID 被忽略（有或没有 vland ID）。然后在条目索引 1 上设置流过滤，流门索引条目 ID 为 1。

1. 通过关闭门设置主流。
2. 设置流识别空流识别条目 1。

```
tsntool> cbstreamidset --device eno0 --index 1 --nullstreamid--nulldmac
0x000000800010 --nulltagged 3 --nullvid 10 --streamhandle100
```

3. 获取流识别条目索引 1。

```
tsntool> cbstreamidget --device eno0 --index 1
```

4. 使用流门条目 ID 1 设置流过滤条目 1。

```
tsntool> qcisfiset --device eno0 --streamhandle 100 --index 1 --gateid1
```

5. 设置流门条目 1，保持门状态关闭（所有帧已丢弃。如果要求用户编辑门列表，则直接返回）。

```
tsntool> qcisgiset --device eno0 --index 1 --initgate 0
```

6. 从对面设备端口发送一帧应传递到关闭门条目 ID 1。

```
tsntool> qcisfiget --device eno0 --index 1
```

7. 结果应类似于以下输出：

```
match pass gate_drop sdu_pass sdu_drop red
1 0 1 1 0 0
```

4.1.4.1.4.3 测试源流识别入口

源流识别是指流通过源 mac 地址和 vlan ID 来识别帧。

使用以下步骤进行此测试：

1. 保留流过滤条目 1 和流门条目 1。
2. 在对面设备端口中添加流 2：`SMAC is 66:55:44:33:22:11 DMAC:20:00:80:00:00:00`（没有目标 MAC 地址 10:00:80:00:00:00，流识别条目索引 1 正在过滤该 dmac 地址）

3. 设置流识别源流识别条目 3

```
tsntool> cbstreamidset --device eno0 --index 3 --sourcemacvid --sourcemac 0x112233445566 -
sourcetagged 3 --sourcevid 20 --streamhandle 100
```

4. 从对面设备端口发送帧。帧传递到流过滤索引 1。

```
tsntool> qcisfiget --device eno0 --index 1
```

4.1.4.1.4.4 SGI 流门列表

使用以下命令进行此测试：

```
cat > sgi1.txt << EOF
t0 0b -1 100000000 0
t1 1b -1 100000000 0
EOF
tsntool> qcisfiset --device eno0 --index 2 --gateid 2
tsntool> qcisgiset --device eno0 --index 2 --initgate 1 --gatelistfile sgi1.txt

#flooding frame size 64bytes from opposite device port.(iperf or netperf as example)
tsntool> qcisfiget --device eno0 --index 2
```

检查丢弃和传递的帧，它们应当相同，因为流门列表设置定期 100ms 打开和 100ms 关闭。

4.1.4.1.4.5 FMI 测试

只发送绿色帧（通常是 802.1Q 标记中的 TCI 位值）。针对速度达 10000kbsp/s 的 eno0 端口泛洪流：

```
tsntool> qcisfiset --device eno0 --index 2 --gateid 2 --flowmeterid 2
tsntool> qcifmiset --device eno0 --index 2 --cm --cf --cbs 1500 --cir 5000 --ebs 1500 --eir 5000
```

“cm”参数设置颜色模式使能表示帧根据帧中 TCI 位的判断分开绿色帧和黄色帧。否则，任何帧都是绿色帧。

“cf”参数设置耦合标志使能。当 CF 设置为 0 时，判定为黄色帧受 EIR 制约。当 CF 设置为 1 时，判定为黄色帧受 CIR + EIR 制约，具体取决于提供的判定为绿色帧的数量。

上层命令设置后，由于绿色帧不大于 EIR + CIR 10Mbit/s。所以绿色帧不会被丢弃。

以下设置显示了丢弃的帧：

```
tsntool> qcifmiset --device eno0 --index 2 --cm --cf --cbs 1500 --cir 5000 --ebs 1500 --eir 2000
```

这种情况下，绿色帧在 CIR 中以 5Mbit/s 传递，然后传递到 EIR 空间，但 EIR 为 2Mbit/s，因此总 EIR + CIR 7Mbit/s 仍然不符合 10Mbit/s 总带宽。所以绿色帧会被丢弃。

要获取在应用层显示的颜色帧计数器的信息，请使用以下示例中的代码：

```
tsntool> qcifmiget --device eno0 --index 2
=========================================================================
bytecount drop dr0_green dr1_green dr2_yellow remark_yellow dr3_red remark_red
1c89 0 4c 0 0 0 0 0
=========================================================================
index = 2
cir = c34c
```

```
cbs = 5dc
eir = 4c4b3c
ebs = 5dc couple flag
color mode
```

4.1.4.1.5 Qav

4.1.4.1.5.1 使用 tsntool

下图显示了 Qav 测试的硬件设置图。

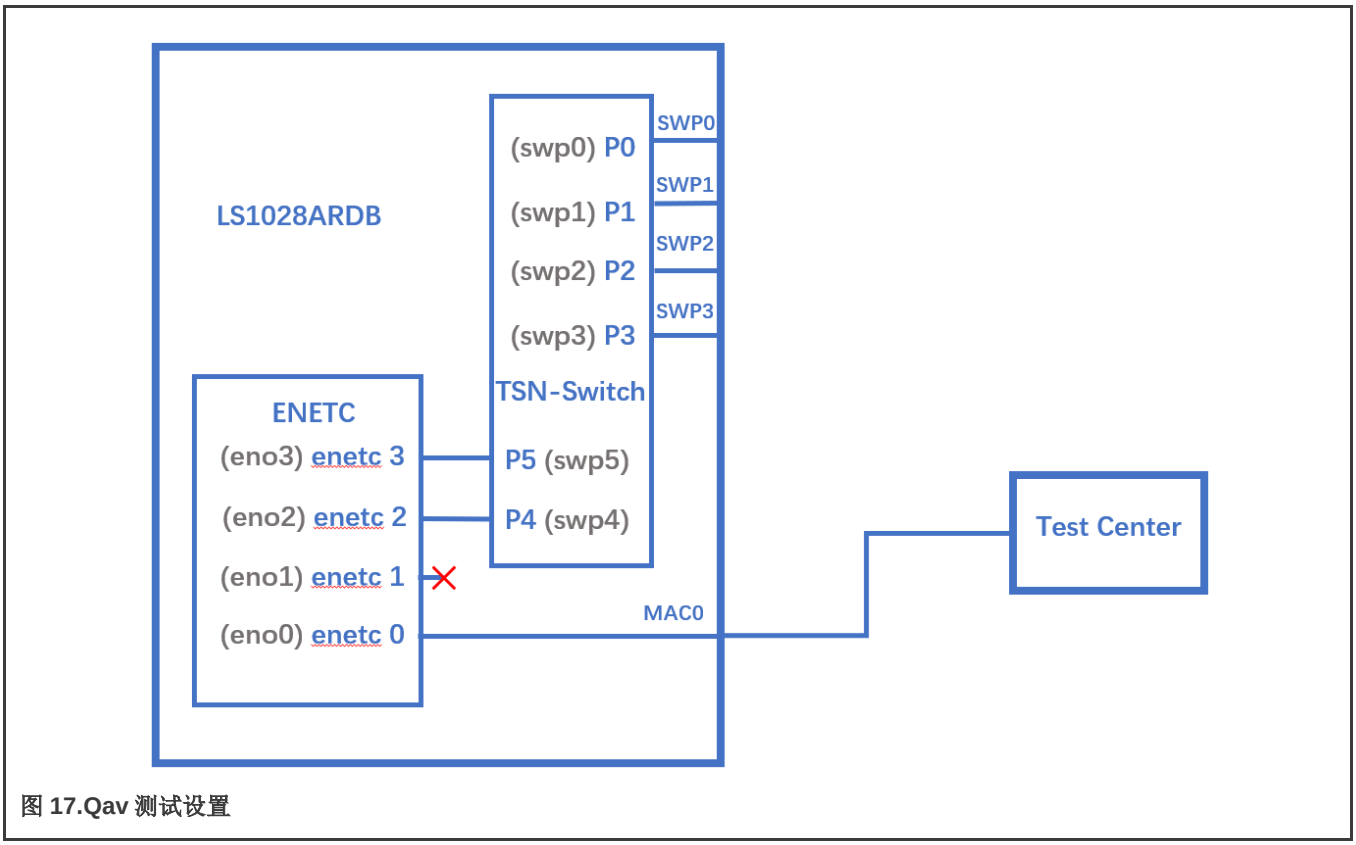


图 17.Qav 测试设置

注：测试中心是一个从 LS1028ardb 板的 enetc0 捕获流的设备。用户还可以使用另一块板通过“tcpdump”捕获流，并使用 Wireshark 对其进行分析。

0.不要忘记为每个流量类别使能优先级：

```
tc qdisc add dev eno0 root handle 1: mqprio num_tc 8 map 0 1 2 3 4 5 6 7 hw 1
```

1. 运行以下命令：

```
tsntool cbsset --device eno0 --tc 7 --percentage 60
tsntool cbsset --device eno0 --tc 6 --percentage 20
```

2. 检查每个队列带宽（pktgen 需要在内核中使能 NET_PKTGEN）

```
/home/root/samples/pktgen/pktgen_sample01_simple.sh -i eno0 -q 7 -s 500 -n 30000
```

等待几秒钟后检查结果。它应该获得大约 60%的线路速率。

```
/home/root/samples/pktgen/pktgen_sample01_simple.sh -i eno0 -q 6 -s 500 -n 30000
```

等待几秒钟后检查结果。它应该获得大约 20%百分比的线路速率。

4.1.4.1.5.2 使用 CBS Qdisc 设置 Qav

LS1028a 支持 CBS qdisc 设置基于信用的整形器。下面的命令设置 CBS，队列 7 为 100 Mbit/s 和队列 6 为 300 Mbit/s。

```
tc qdisc add dev eno0 root handle 1: mqprio num_tc 8 map 0 1 2 3 4 5 6 7 hw 1
tc qdisc replace dev eno0 parent 1:8 cbs locredit -1350 hicredit 150 sendslope -900000 idleslope 100000
offload 1
tc qdisc replace dev eno0 parent 1:7 cbs locredit -1050 hicredit 950 sendslope -700000 idleslope 300000
offload 1
```

```
# Try to flood stream here (require kernel enable NET_PKTGEN)
/home/root/samples/pktgen/pktgen_sample01_simple.sh -i eno0 -q 7 -s 500 -n 20000
/home/root/samples/pktgen/pktgen_sample01_simple.sh -i eno0 -q 6 -s 500 -n 20000
tc qdisc del dev eno0 parent 1:7 cbs
tc qdisc del dev eno0 parent 1:8 cbs
```

4.1.4.2 Felix 交换机上的 TSN 配置

以下几节介绍 TSN 交换机的基本配置示例。

4.1.4.2.1 Linux 交换机配置

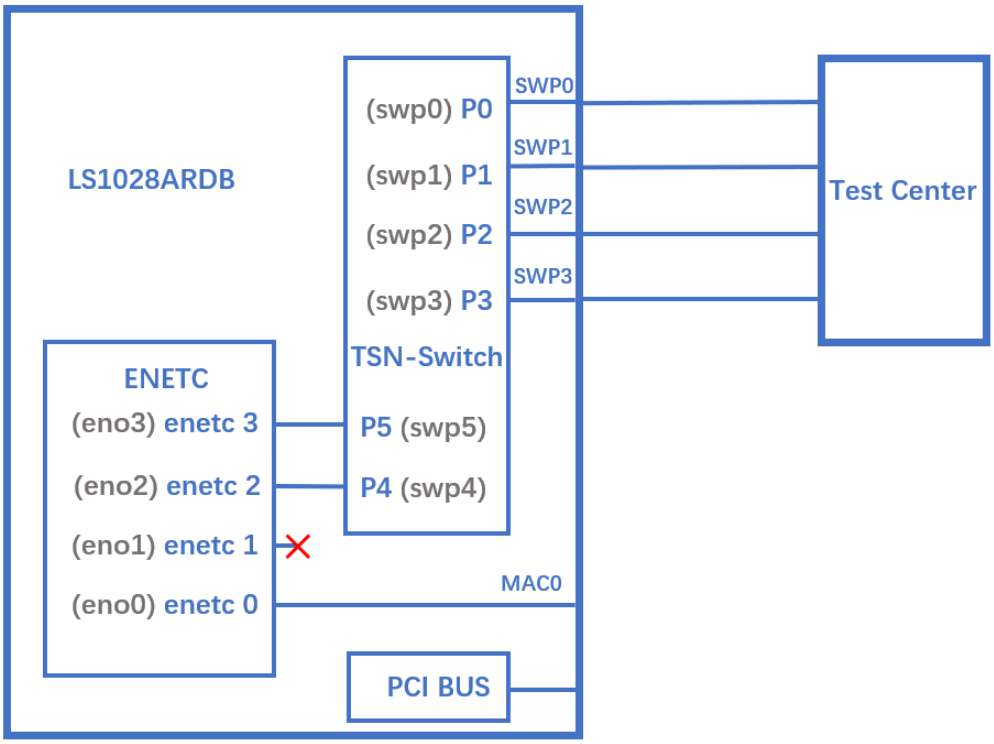


图 18.TSN 交换机配置

注：测试中心是一个创建（需要 802.1Q Vlan 标记）发送到 LS1028a 交换机的流和捕获 LS1028a 交换机转发的流的设备。

使用以下命令在 LS1028ARDB 上配置交换机：

```
ls /sys/bus/pci/devices/0000:00:00.5/net/
```

获取交换机设备接口：swp0 swp1 swp2 swp3>

```
ifconfig eno2 up
ip link add name switch type bridge
ip link set switch up
ip link set swp0 master switch && ip link set swp0 up
ip link set swp1 master switch && ip link set swp1 up
ip link set swp2 master switch && ip link set swp2 up
ip link set swp3 master switch && ip link set swp3 up
```

4.1.4.2.2 时钟同步

要在 felix 交换机接口上测试 1588 同步，请使用以下步骤：

1. 使用交换机接口将两块板背靠背连接。例如，swp0 到 swp0。

   Linux 启动日志如下所示：

   ```
   ...
   pps pps0: new PPS source ptp1
   ...
   ```

2. 使用以下命令检查 PTP 时钟和时间戳功能：

   ```
   $ ethtool -T swp0
   Time stamping parameters for swp0:
   Capabilities:
          hardware-transmit          SOF_TIMESTAMPING_TX_HARDWARE)
          hardware-receive           (SOF_TIMESTAMPING_RX_HARDWARE)
          hardware-raw-clock         (SOF_TIMESTAMPING_RAW_HARDWARE)
   PTP Hardware Clock: 1
   Hardware Transmit Timestamp Modes:
          off       (HWTSTAMP_TX_OFF)
          on        (HWTSTAMP_TX_ON)
   Hardware Receive Filter Modes:
          none      (HWTSTAMP_FILTER_NONE)
          all       (HWTSTAMP_FILTER_ALL)
   ```

3. 在两块板上设置交换机 IP，并互相 ping 通。

   ```
   $ ifconfig switch 192.168.1.2 /* On board A */
   $ ifconfig switch 192.168.1.3 /* On board B */
   $ ping 192.168.1.3 /* On board A */
   ```

4. 对于 802.1AS 测试，请使用 linuxptp 源中的配置文件 `gPTP.cfg`。改为在两个板上运行以下命令。

   ```
   $ ptp4l -i swp0 -p /dev/ptp1 -f /etc/ptp4l_cfg/gPTP.cfg -2-m
   ```

   或者通过以下命令使用 GenAVB/TSN 协议栈：`'avb.sh start'`。请注意，会自动使用配置文件`/etc/genavb/fgptp-br.cfg`。

注：必须在设置 Qbv 之前运行时钟同步。如果 Qbv 正在运行，用户应在时钟同步后禁用并重新配置 Qbv。

4.1.4.2.3 LS1028ARDB 的 Qbv 测试设置

下图显示了在 LS1028ARDB 上进行 `Qbv test` 的设置。

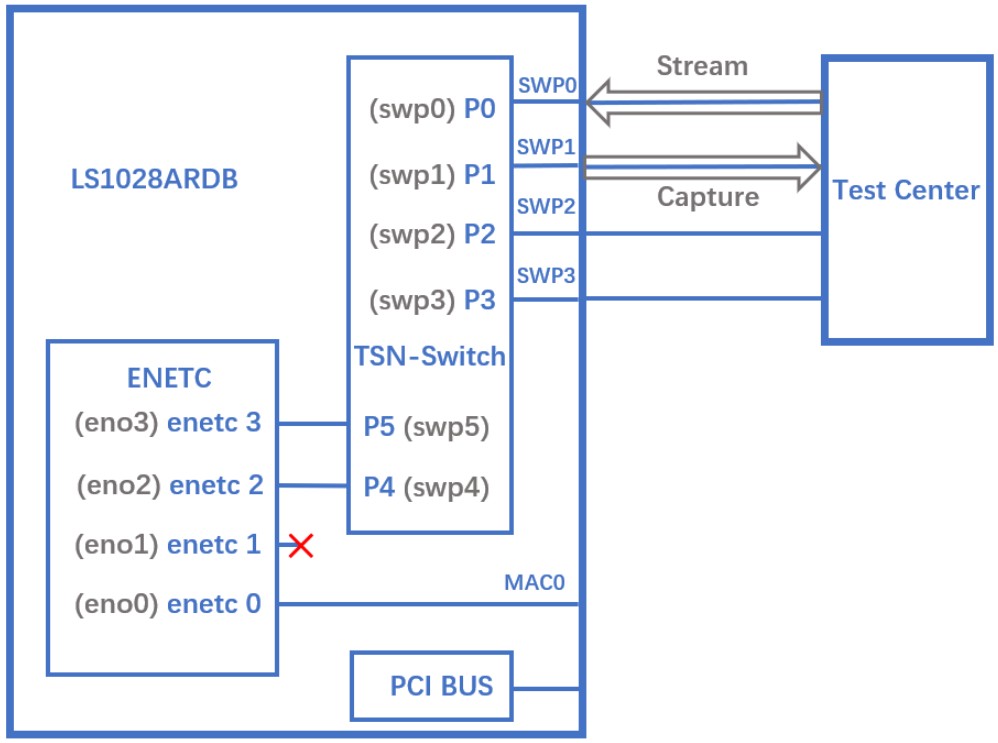


图 19.Qbv 测试

4.1.4.2.3.1 Tsntool 用法

4.1.4.2.3.1.1 关闭基本门

使用以下命令集进行基本门关闭。

```
echo “t0 00000000b 20000” > qbv0.txt
#Explaination:
# 'NUMBER'      : t0
# 'GATE_VALUE'  : 00000000b
# 'TIME_LONG'   : 20000 ns

./tsntool
tsntool> verbose
tsntool> qbvset --device swp1 --entryfile ./qbv0.txt

#Send one broadcast frame to swp0 from TestCenter.
ethtool -S swp1
#Should not get any frame from swp1 on TestCenter.

echo “t0 11111111b 20000” > qbv0.txt
tsntool> qbvset --device swp1 --entryfile ./qbv0.txt
```

```
#Send one broadcast frame to swp0 on TestCenter.
ethtool -S swp1
#Should get one frame from swp1 on TestCenter.
```

4.1.4.2.3.1.2 基准时间测试

对于基准时间测试，首先获取当前的第二个时间：

```
#Get current time:
tsntool> ptptool -g -d /dev/ptp1

#add some seconds, for example user gets 200.666 time clock, then set 260.666 as result
tsntool> qbvset --device swp1 --entryfile ./qbv0.txt --basetime 260.666

#Send one broadcast frame to swp0 on the Test Center.
#Frame could not pass swp1 until time offset.
```

4.1.4.2.3.1.3 Qbv 性能测试

使用以下命令进行 QBv 性能测试：

```
cat > qbv5.txt << EOF
t0 11111111b 1000000
t1 00000000b 1000000
EOF
qbvset --device swp1 --entryfile qbv5.txt
```

#发送 1G 速率流到测试中心上的 swp0。

#流将从 swp1 获取大约一半的线路速度。

4.1.4.2.3.2 tc-taprio 用法

LS1028ARDB 也支持 tarprio qdisc 设置 Qbv。下面是一个示例设置。

1.为 swp1 端口使能 Qbv，设置队列 1 门打开，设置循环时间为 300 µs。

```
tc qdisc replace dev swp1 parent root handle 100 taprio num_tc 8 map 0 1 2 3 4 5 6 7 \
  queues 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 base-time 0 sched-entry S 02 300000 flags 0x2
```

注意

由于硬件只能使用 PCP、DSCP 或其他方法对 Qos 进行分类，因此无法将 Qos 映射到不同的硬件队列。mqprio 没有内置在 felix 驱动程序，所以 tc-taprio 命令中的“映射 0 1 2 3 4 5 6 7”无效。

2.从测试中心向 swp0 发送 vlan tag 中 PCP=1 的一帧，我们将从 swp1 捕获该帧。

3.从测试中心向 swp0 发送 vlan 标记中 PCP=2 的一帧，门关闭，我们无法从 swp1 捕获该帧。

4.按如下所示为 swp1 端口禁用 Qbv

```
tc qdisc del dev swp1 parent root handle 100 taprio
```

4.1.4.2.4 Qbu

下图显示了使用 TSN 交换机执行 Qbu 测试的设置。

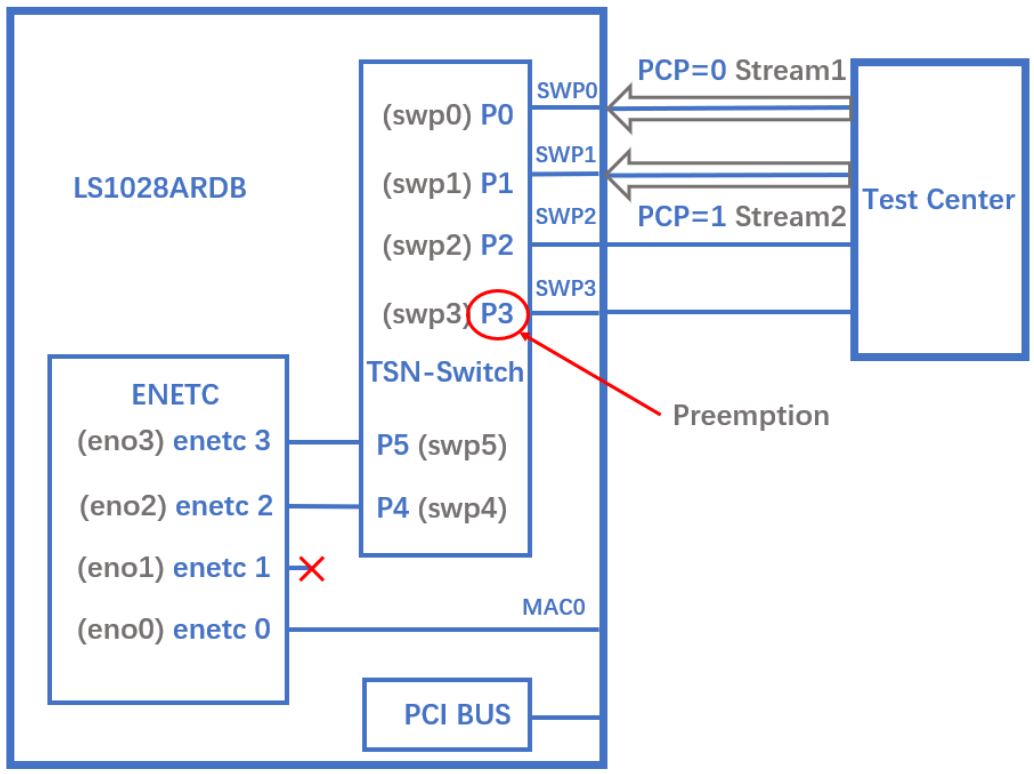


图 20.交换机上的 Qbu 测试

4.1.4.2.4.1 Tsntool 用法

1. 将队列 1 设置为可抢占。设置可抢占队列有两种方式，用户可以选择 tsntool 或 ethtool 来设置。

```
#tsntool command to set preemptable queues:
tsntool> qbuset --device swp3 --preemptable 0x02
```

2. 从测试中心发送两个流，设置数据包大小为 1500 字节，带宽为 1G，然后查看 PMAC 发送的额外 mPackets 数量：

```
ethtool -S swp3 | grep tx_merge_fragments
```

3. Qbu 与 Qbv 测试结合。

   设置队列 0 门打开 20us，队列 1 门打开 20us。

```
cat> qbv0.txt << EOF
t0 00000001b 200000
t1 00000010b 200000
EOF
qbvset --device swp3 --entryfile qbv0.txt
```

   从测试中心发送两个流，当门 1 关闭时，队列 1 中的数据包将被抢占

4.1.4.2.4.2 Ethtool 用法

1. 将队列 1 设置为可抢占。设置可抢占队列有两种方式，用户可以选择 tsntool 或 ethtool 来设置。

```
#ethtool command to set preemptable queues:
ethtool --set-frame-preemption swp3 preemptible-queues-mask 0x02 min-frag-size 124
```

解释：

- `preemptible-queues-mask`: 一个 8 位向量，指定 8 个优先级中的可抢占队列（位 0 表示优先级 0，位 7 表示优先级 7）。
- `min-frag-size`：分片中至少传输了帧字节，最小非最终分片大小为 64、128、192 或 256 个八位字节（包括 4 字节分片标头）。

2. 从测试中心发送两个流，设置数据包大小为 1500 字节，带宽为 1G，然后查看 PMAC 发送的额外 mPackets 数量：

```
ethtool -S swp3 | grep tx_merge_fragments
```

3. Qbu 与 Qbv 测试结合。

   设置队列 0 门打开 20us，队列 1 门打开 20us。

```
tc qdisc replace dev swp3 parent root handle 100 taprio num_tc 8 map 0 1 2 3 4 5 6 7\
    queues 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 base-time 0 \
        sched-entry S 01 200000 \
        sched-entry S 02 200000 flags 0x2
```

   从测试中心发送两个流，当门 1 关闭时，队列 1 中的数据包将被抢占

4.1.4.2.5 QCI

下图显示了 Qci 测试用例设置。

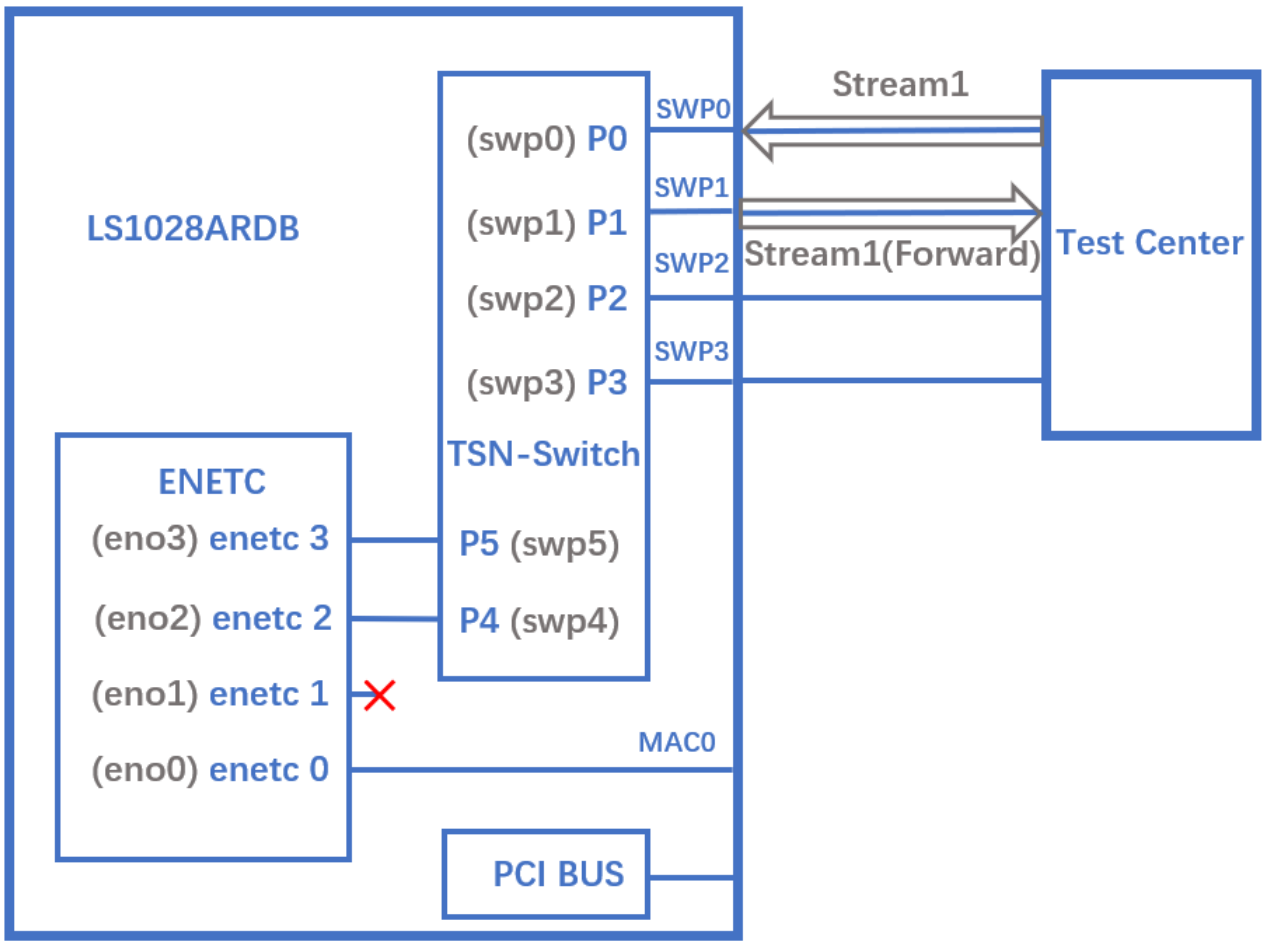


图 21.Qci 测试用例

4.1.4.2.5.1 Tsntool 用法

4.1.4.2.5.1.1 流识别

使用以下命令进行流识别：

1. 在测试中心将流设置为 `swp0`。编辑流，将目标 MAC 设置为：`00:01:83:fe:12:01, Vlan ID : 1`
2. 将 MAC 添加到 LS1028a 上的 MAC 表。（如果已经在端口上了解 mac，则不需要此步骤）

```
bridge fdb add 00:01:83:fe:12:01 dev swp1 vlan 1
```

3. 将目标 MAC 用作：`00:01:83:fe:12:01, Vlan ID : 1` 在 LS1028a 上设置流识别。

```
tsntool> cbstreamidset --device swp1 --nullstreamid --index 1 --nulldmac 0x000183fe1201 --
nullvid 1 --streamhandle 1
```

解释：

- `device`：设置流转发到的设备端口。如果交换机已经了解{destmac, VID}，交换机将不关注设备端口。
- `nulltagged`：交换机只支持 nulltagged=1 模式，所以不需要设置。

- `nullvid`: 使用“网桥 vlan 显示”查看交换机端口的入口 VID。

```
tsntool> qcisfiset --device swp0 --index 1 --streamhandle 1 --gateid 1 --priority 0 --
flowmeterid 68
```

解释：

- `device`：可以是任意一个交换机端口。
- `index`：值与 cbstreamidset 的流句柄相同。
- `streamhandle`：值与 cbstreamidset 的流句柄相同。
- `flowmeterid`：PSFP 监管其 ID，范围从 63 到 383。

4. 发送一帧，然后检查帧。

```
ethtool -S swp1
ethtool -S swp2
```

只有 `swp1` 可以获取该帧。

5. 使用以下命令检查和调试流识别状态。

```
qcisfiget --device swp0 --index 1
```

注意

参数 `streamhandle` 与流过滤设置中的 `index` 相同，我们使用 `streamhandle` 作为 SFID 来识别流，使用 `index` 来设置流过滤表条目。

4.1.4.2.5.1.2 流门控

1. 使用以下命令进行流门控：

```
echo "t0 1b 3 50000 200" > sgi.txt
tsntool> qcisgiset --device swp0 --enable --index 1 --initgate 1 --initipv 0 --gatelistfile
sgi.txt --basetime 0x0
```

解释：

- “device”：可以是任意一个交换机端口。
- “index”：门 ID
- “basetime”：与 Qbv 集相同。

2. 在测试中心发送一帧。

```
ethtool -S swp1
```

请注意，帧可以传递，并且 green_prio_3 已提高。

3. 现在运行以下命令：

```
echo "t0 0b 3 50000 200" > sgi.txtx
tsntool> qcisgiset --device swp0 --enable --index 1 --initgate 1 --initipv 0 --gatelistfile
sgi.txt --basetime 0x0
```

4. 接下来，在测试中心发送一帧。

```
ethtool -S swp1
```

请注意，该帧无法传递。

4.1.4.2.5.1.3 SFI maxSDU 测试

使用以下命令运行此测试：

```
tsntool> qcisfiset --device swp0 --index 1 --gateid 1 --priority 0 --flowmeterid 68 --maxsdu 200
```

现在，在测试中心发送一帧（帧大小 > 200）。

```
ethtool -S swp1
```

用户可以观察到该帧无法传递。

4.1.4.2.5.1.4 FMI 测试

使用以下命令集进行 FMI 测试。

1. 运行命令：

   ```
   tsntool> qcifmiset --device swp0 --index 68 --cir 100000 --cbs 4000 --ebs 4000 --eir 100000
   ```

   注意

   - 上述命令中的“device”可以是任意一个交换机端口。
   - `qcifmiset` 的索引必须与 `qcisfiset` 的流量计 ID 相同。

2. 现在，在测试中心发送一个流（速率 = 100M）。

   ```
   ethtool -S swp0
   ```

   请注意，所有帧都传递并获取所有绿色帧。

3. 现在，在测试中心发送一个流（速率 = 200M）。

   ```
   ethtool -S swp0
   ```

   观察所有帧是否都传递并获取绿色和黄色帧。

4. 在测试中心发送一个流（速率 = 300M）。

   ```
   ethtool -S swp0
   ```

   请注意，并非所有帧都可以传递并获得绿色、黄色和红色帧。

5. 在测试中心发送一个黄色的流（速率 = 100M）。

   ```
   ethtool -S swp0
   ```

   所有帧都传递并获取所有黄色帧。

6. 在测试中心发送一个黄色的流（速率 = 200M）。

   ```
   ethtool -S swp0
   ```

   请注意，并非所有帧都可以传递并获取黄色和红色帧。

7. 测试 cf 模式。

   ```
   tsntool> qcifmiset --device swp0 --index 68 --cir 100000 --cbs 4000 --ebs 4000 --eir 100000 --cf
   ```

8. 在测试中心发送一个黄色的流（速率 = 200M）。

```
ethtool -S swp0
```

所有帧都传递并获取所有黄色帧（使用 CIR 和 EIR）。

9. 在测试中心发送一个黄色的流（速率 = 300M）。

```
ethtool -S swp0
```

请注意，并非所有帧都可以传递并获取黄色和红色帧。

4.1.4.2.5.1.5 基于端口的 SFI 集

LS1028A 交换机可以在基于端口的 PSFP 集上工作。这表示当在入口端口上接收到空识别的流时，交换机将使用该端口，默认 SFI。

下面的示例测试 qcisfiset 中没有流句柄来设置端口，默认 SFI。

1.使用 SFID 2 将 swp0 端口设置为默认 SFI。

```
tsntool> qcisfiset --device swp0 --index 2 --gateid 1 --flowmeterid 68
```

设置端口默认 SFI 后，从 swp0 端口发送的任何流都将执行门 1 和流量计 68 策略。

2.设置流门控。

```
echo "t0 1b 4 50000 200" > sgi.txt
tsntool> qcisgiset --device swp0 --enable --index 1 --initgate 1 --initipv 0 --gatelistfile sgi.txt
```

3.将任何流发送到 swp0。

```
ethtool -S swp1
```

请注意，帧可以传递，并且 green_prio_4 已提高。

4.1.4.2.5.2 Tc-flower 用法

下图显示了基于 tc-flower 的 Qci 测试案例设置。

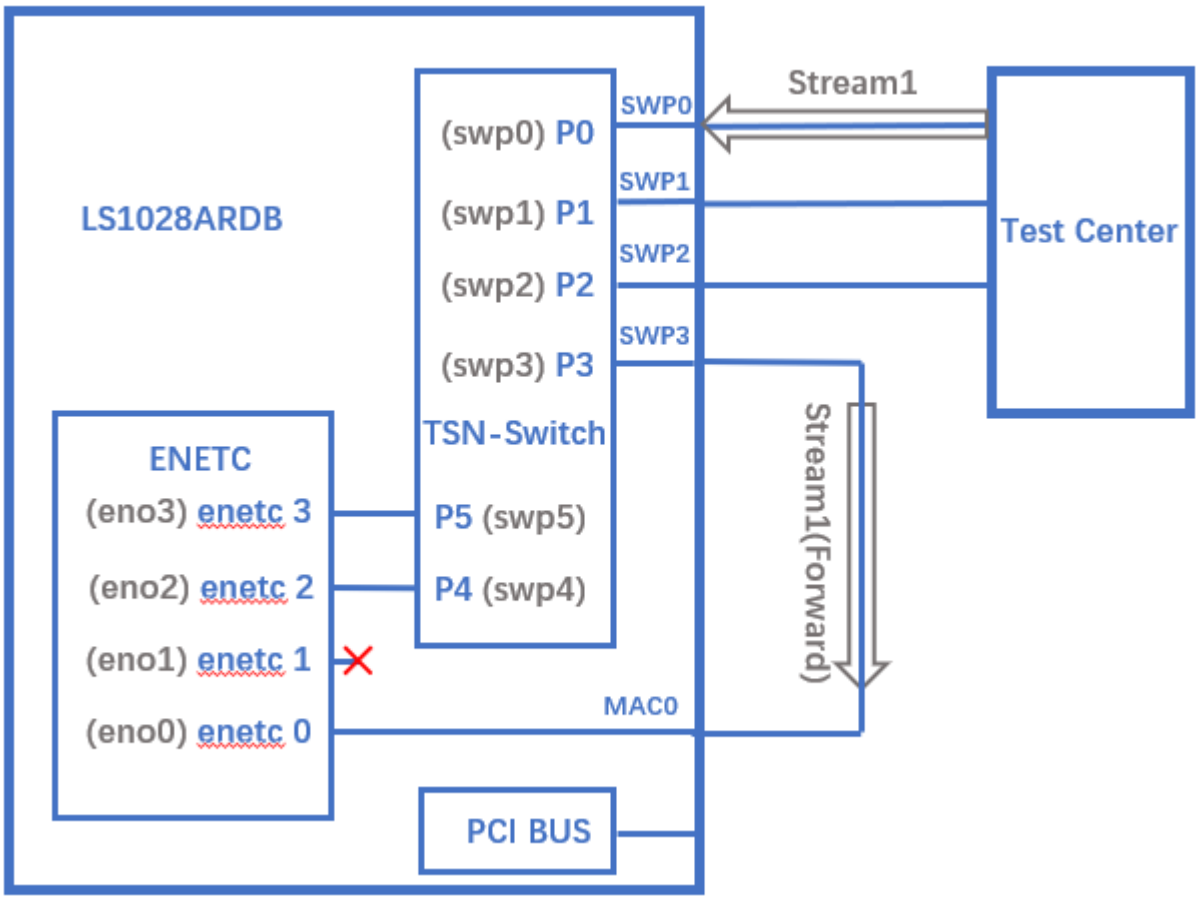


图 22.基于 TC-flower 的 Qci 测试案例

1.在 LS1028ARDB 板上获取目标 MAC，然后使用以下命令中所示的“dst_mac CA:9C:00:BC:6D:68”。

```
ifconfig eno0
Link encap: Ethernet HWaddr CA:9C:00:BC:6D:68
           inet addr:169.254.88.50  Bcast:169.254.255.255 Mask:255.255.0.0
           inet6 addr: fe80::ed36:c4ce:bb04:863d/64 Scope:Link
           UP BROADCAST RUNNING MULTICAST  MTU:1500 Metric:1
           RX packets:2 errors:0 dropped:0 overruns:0 frame:0
           TX packets:1529 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:1000
           RX bytes:152 (152.0 B)  TX bytes:118456 (115.6 KiB)
```

2.在入口端口 swp0 上设置 Qci。

a)使用以下命令设置 Qci 门。

```
tc qdisc add dev swp0 ingress
tc filter add dev swp0 chain 30000 protocol 802.1Q parent ffff: flower skip_sw dst_mac
CA:9C:00:BC:6D:68 vlan_id 1 action gate index 1 base-time 0 sched-entry CLOSE 6000 -1 -1
```

b).使用以下命令设置 Qci 流量计。

```
tc qdisc add dev swp0 ingress
tc filter add dev swp0 chain 30000 protocol 802.1Q parent ffff: flower skip_sw dst_mac
CA:9C:00:BC:6D:68 vlan_id 1 action police index 1 rate 10Mbit burst 10000
```

c).使用以下命令设置 Qci SFI 优先级。

```
tc qdisc add dev swp0 ingress
tc filter add dev swp0 chain 30000 protocol 802.1Q parent ffff: flower skip_sw dst_mac
CA:9C:00:BC:6D:68 vlan_id 1 vlan_prio 1 action gate index 1 base-time 0 sched-entry CLOSE 6000 -1
-1
```

d).使用以下命令设置门和流量计。

```
tc qdisc add dev swp0 ingress
tc filter add dev swp0 chain 30000 protocol 802.1Q parent ffff: flower skip_sw dst_mac
CA:9C:00:BC:6D:68 vlan_id 1 action gate index 1 base-time 0 sched-entry OPEN 6000 2 -1 action police
index 1 rate 10Mbit burst 10000
```

3.从测试中心发送流，将流目标 mac 设置为 `CA:9C:00:BC:6D:68`，在 vlan 标记中设置 `vid=1` 和 `vlan_prio=1`。

4.使用“`tcpdump -i eno0 -w eno0.pcap`”接收 eno0 上的流，检查是否接收到数据包。

5.使用以下命令删除流规则。

```
tc -s filter show dev swp0 ingress chain 30000
tc filter del dev swp0 ingress  chain 30000 pref 49152
```

注意

- 每个流只能添加一次。如果用户想要更新规则，请删除该规则并添加一个新规则。
- 如果需要添加流，则必须在交换机 MAC 表中了解流的 MAC 和 VID。
- Qci 门周期时间预计将超过 5 µs。
- Qci 流量计现在只能设置 `cir` 和 `cbs`，监管器与 ACL VCAP 共享。

4.1.4.2.6 Qav

下图显示了 Qav 测试案例设置。

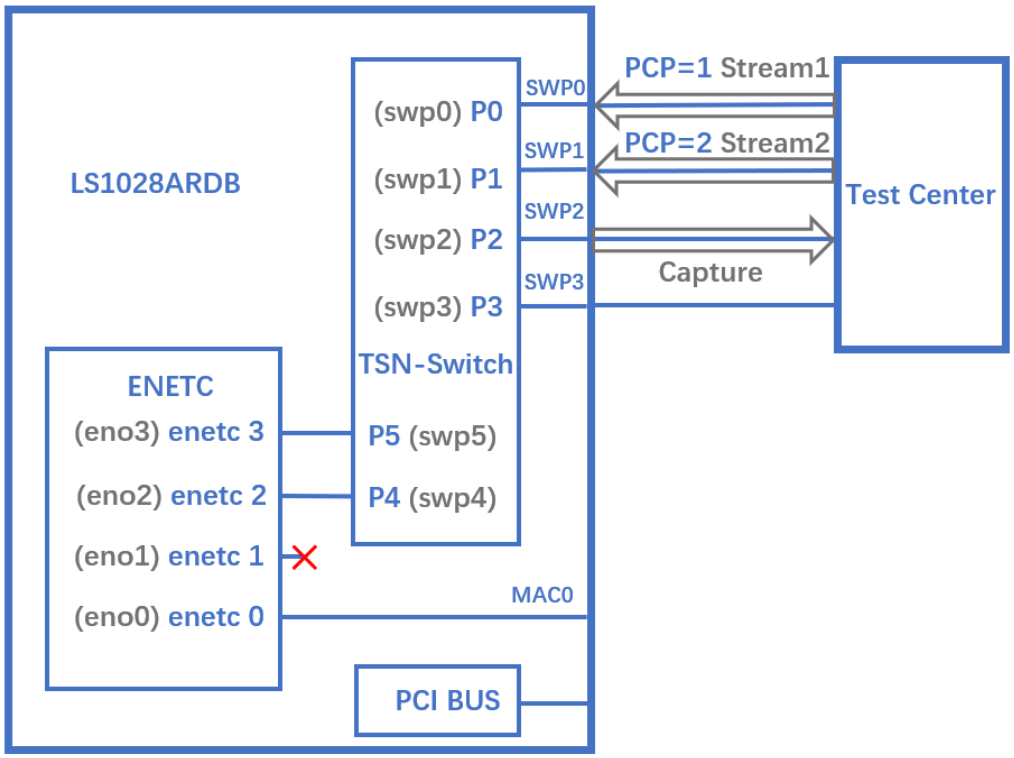


图 23.Qav 测试案例

4.1.4.2.6.1 Tsntool 用法

1. 设置两个流量类别的百分比：

```
tsntool> cbsset --device swp2 --tc 1 --percentage 20
tsntool> cbsset --device swp2 --tc 2 --percentage 40
```

2. 从测试中心发送两个流，然后检查帧数。

```
ethtool -S swp2
```

注意队列 1 的帧数是队列 2 的一半。

> 注意
> 流率必须大于队列限制的带宽。

3. 在测试中心的 swp2 上捕获帧。

```
# The Get Frame sequence is: (PCP=1), (PCP=2), (PCP=2), (PCP=1), (PCP=2), (PCP=2),…
```

4.1.4.2.6.2 Tc-cbs 用法

LS1028A 支持 CBS qdisc 设置基于信用的整形器。以下命令将队列 1 的 CBS 设置为 20 Mbit/s，队列 2 设置为 40 Mbit/s。

1.设置两个流量类别的 cbs：

```
tc qdisc add dev swp2 root handle 1: mqprio num_tc 8 map 0 1 2 3 4 5 6 7 \
  queues 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 hw 0
tc qdisc replace dev swp2 parent 1:2 cbs locredit -1470 hicredit 30 \
  sendslope -980000 idleslope 20000 offload 1
tc qdisc replace dev swp2 parent 1:3 cbs locredit -1440 hicredit 60 \
  sendslope -960000 idleslope 40000 offload 1
```

2.从测试中心发送一个 PCP=1 的流，我们可以从 swp2 获取带宽为 20 Mbit/s 的流。

3.从测试中心发送两个流，然后检查帧数。

```
ethtool -S swp2
```

注意

队列 1 的帧数是队列 2 的一半。

4.删除 cbs 规则。

```
tc qdisc del dev swp2 parent 1:2 cbs
tc qdisc del dev swp2 parent 1:3 cbs
```

4.1.4.2.7 802.1CB

下图显示了无缝冗余测试案例的测试设置。

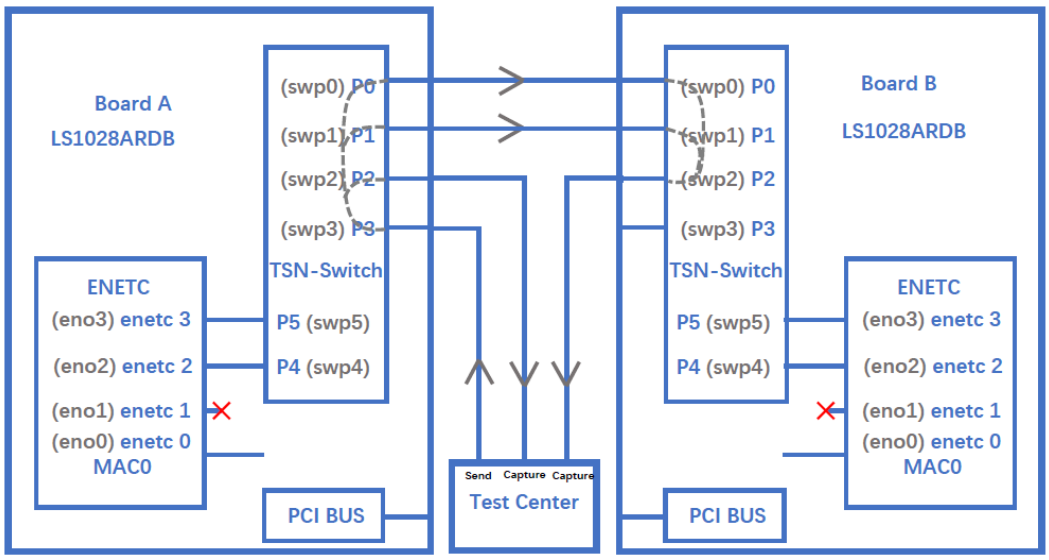


图 24.无缝冗余测试

4.1.4.2.7.1 序列生成器测试

使用以下命令集进行“序列生成器”测试。

1. 将交换机端口配置为转发模式。

   A 板上：

   ```
   ifconfig eno2 up
   ip link add name switch type bridge vlan_filtering 1
   ip link set switch up
   ```

```
ip link set swp0 master switch && ip link set swp0 up
ip link set swp1 up
ip link set swp2 master switch && ip link set swp2 up
ip link set swp3 master switch && ip link set swp3 up
bridge vlan add dev swp0 vid 1 pvid
bridge vlan add dev swp2 vid 1 pvid
bridge vlan add dev swp3 vid 1 pvid
```

B 板上

```
ifconfig eno2 up
ip link add name switch type bridge vlan_filtering 1
ip link set switch up
ip link set swp0 master switch && ip link set swp0 up
ip link set swp1 master switch && ip link set swp1 up
ip link set swp2 master switch && ip link set swp2 up
ip link set swp3 master switch && ip link set swp3 up
bridge vlan add dev swp0 vid 1 pvid
bridge vlan add dev swp1 vid 1 pvid
bridge vlan add dev swp2 vid 1 pvid
bridge vlan add dev swp3 vid 1 pvid
```

2. 在 A 板上，运行命令：

```
bridge fdb add 7E:A8:8C:9B:41:DD dev swp0 vlan 1
tsntool> cbstreamidset --device swp0 --index 1 --nullstreamid --nulldmac 0x7EA88C9B41DD --
nullvid 1 --streamhandle 1
tsntool> cbgen --device swp3 --index 1 --iport_mask 0x08 --split_mask 0x07 --seq_len 16 --
seq_num 2048
```

在上面的命令中，

- `device`：可以是任意一个交换机端口。
- `index`：值与 cbstreamidset 的流句柄相同。

3. 从测试中心向 A 板的 swp3 发送流，将目标 mac 设置为 7E:A8:8C:9B:41:DD。
4. 在测试中心的 swp2 上捕获帧。

   我们可以从测试中心的 swp2 获取帧，每帧都添加序列号：23450801、23450802、23450803……
5. 从测试中心 B 板的 swp2 获取帧，我们可以获取相同的帧。

4.1.4.2.7.2 序列恢复测试

对**序列恢复**测试使用以下步骤：

1. 在 B 板上，运行以下命令：

```
bridge fdb add 7E:A8:8C:9B:41:DD dev swp2 vlan 1
tsntool> cbstreamidset --device swp2 --index 1 --nullstreamid --nulldmac 0x7EA88C9B41DD -
- nullvid 1  --streamhandle 1
tsntool> cbrec --device swp0 --index 1 --seq_len 16 --his_len 31--rtag_pop_en
```

在上述 cbrec 命令中：

- `device`：可以是任意一个交换机端口。
- `index`：值与 `cbstreamidset` 的 `streamhandle` 相同。

2. 从测试中心向 A 板的 swp3 发送一帧，将 dest mac 设置为 7E:A8:8C:9B:41:DD。
3. 从测试中心 B 板的 swp2 获取帧，我们只能获取没有序列标记的一帧。

4.1.4.2.8 TSN 流识别

TSN 模块使用 QoS 类别来识别和控制流。有三种方法可以将流识别为不同的 QoS 类别。这些将在以下几节中进行讲解。

4.1.4.2.8.1 基于 Vlan 标记 PCP 值的流识别

默认的 QoS 类别基于帧的 Vlan 标记的 PCP。如果帧没有 Vlan 标记，则默认 QoS 类别为 0。

在测试中心设置 PCP 值。

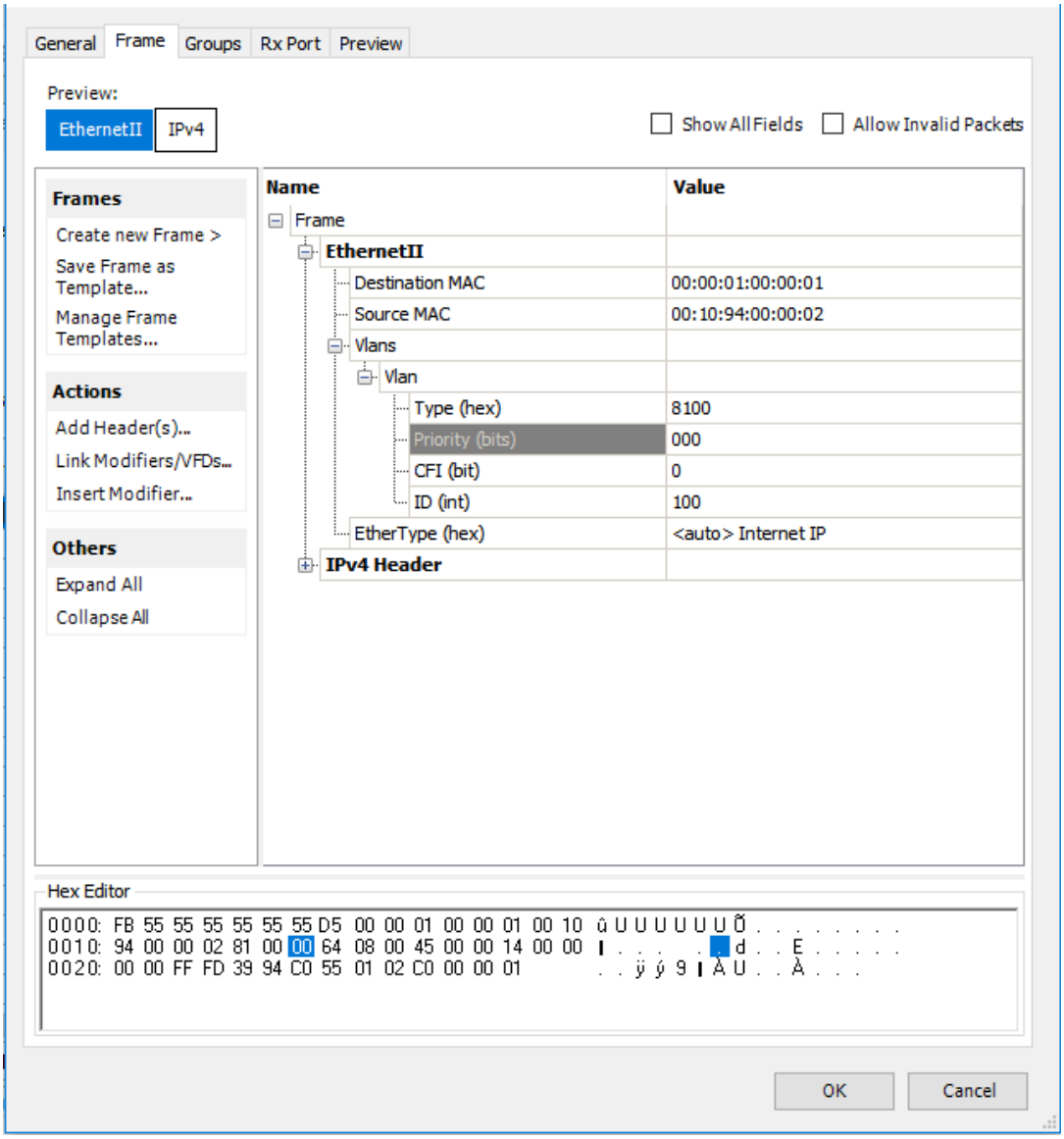


图 25.使用 Vlan 标记的 PCP 值

4.1.4.2.8.2 基于 ToS 标记的 DSCP

使用以下步骤基于 ToS 标记的 DSCP 值识别流。

1. 使用以下命令将 DSCP 值映射到特定的 QoS 类别：

```
tsntool> dscpset --device swp0 --index 1 --cos 1 --dpl 0
```

解释：

- `index`：流的 DSCP 值，0-63。
- `cos`：映射到的 QoS 类别。
- `dpl`：映射到的丢弃级别。

2. 在测试中心设置 DSCP 值。DSCP 值是 IP 标头中 ToS 的高六位，在测试中心设置 DSCP 值，如下图所示。

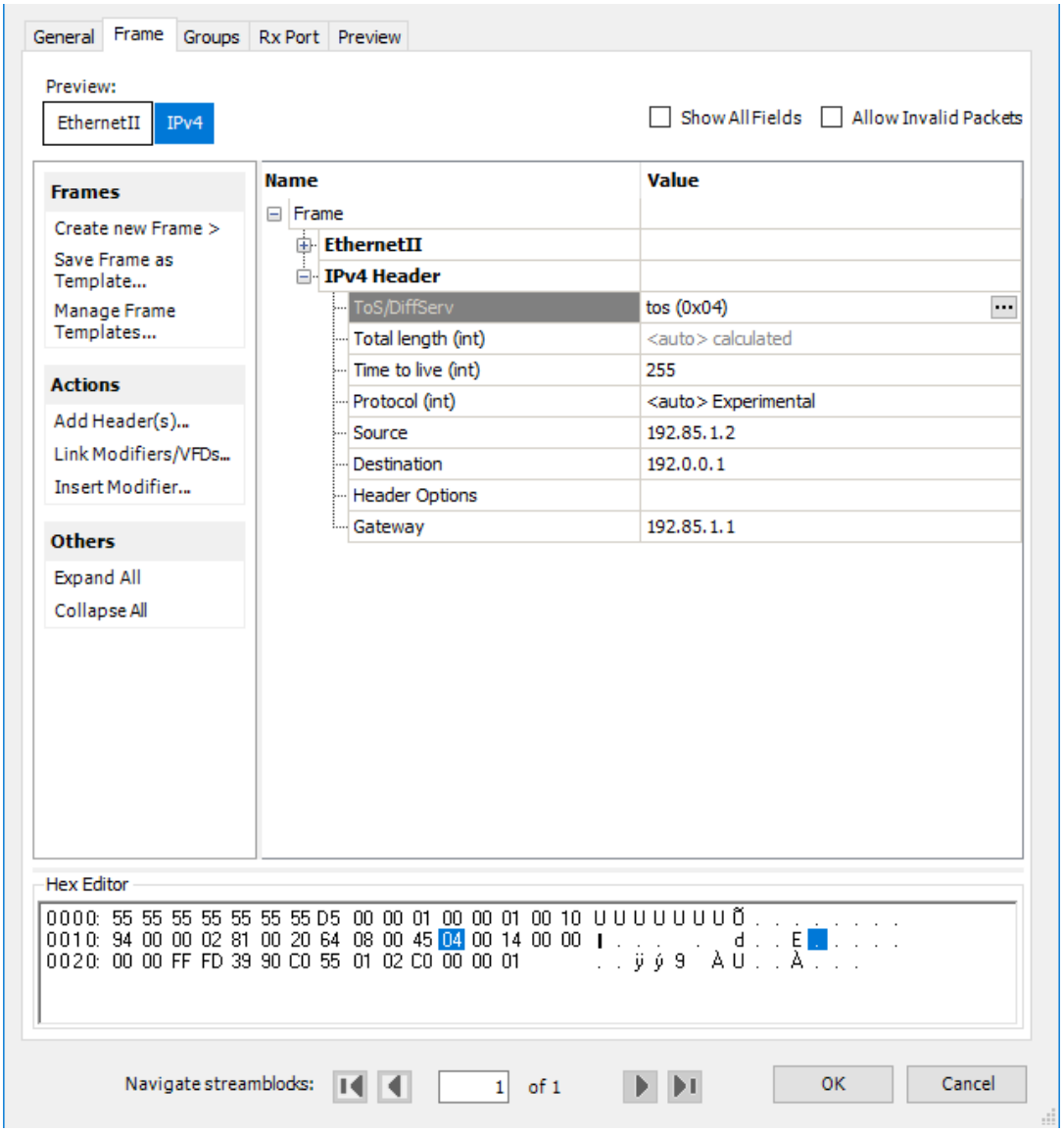


图 26.在测试中心设置 DSCP 值

4.1.4.2.8.3 基于 qci 流识别

以下步骤说明了如何使用 qci 来识别流并将其设置为 QoS 类别。

1. 识别流。

```
tsntool> cbstreamidset --device swp1 --nullstreamid --nulldmac 0x000183fe1201 --nullvid 1 --
streamhandle 1
tsntool> qcisfiset --device swp0 --index 1 --gateid 1 --flowmeterid68
```

2. 使用流门控设置为 Qos 类别 3。

```
echo "t0 1b 3 50000 200" > sgi.txt
tsntool> qcisgiset --device swp0 --enable --index 1 --initgate 1 --initipv 0—
gatelistfile sgi.txt
```

注意

基于 Qci 的识别流只能用于均为网桥端口的入口和出口。无法为 Qci 配置通过 CPU 端口注入或提取流。

4.1.5 LS1021A-TSN 上的 TSN

在 LS1021A-TSN 平台上，TSN 功能由 SJA1105TEL 汽车以太网交换机提供。这些硬件功能符合以下 IEEE 规范的预标准（草案）版本：

- 802.1Qbv——时间感知整形
- 802.1Qci——逐流过滤和监管
- 1588v2——精确时间协议

以下演示介绍下列 SJA1105 硬件功能：

- 通过 L2（最佳）监管器限制入口速率
- 时间感知整形
- 802.1AS gPTP 同步

4.1.5.1 拓扑结构

为了演示 SJA1105 TSN 功能，需要以下拓扑：

- 1 个 LS1021A-TSN 板，作为 TSN 交换机
- 1 个具有 PTP 硬件时间戳功能的通用主机（可以是 PC 或其他板），充当延迟敏感流量的发送器
- 1 个通用主机（可以是 PC 或其他板），充当高带宽流量的发送器
- 1 个具有 PTP 硬件时间戳功能的通用主机（可以是 PC 或其他板），充当延迟敏感和高带宽流量的接收器

通用主机所需的软件包是：

- 来自 linuxptp 包的 ptp4l、phc2sys 和 phc_ctl：https://sourceforge.net/projects/linuxptp/files/v3.1/linuxptp-3.1.tgz
- iperf3
- 来自 tsn-scripts 包的等时线 https://github.com/vladimiroltean/tsn-scripts/tree/isochron

假设通用主机通过名为 eth0 的接口连接到 LS1021A-TSN 板。

此拓扑如下图所示。

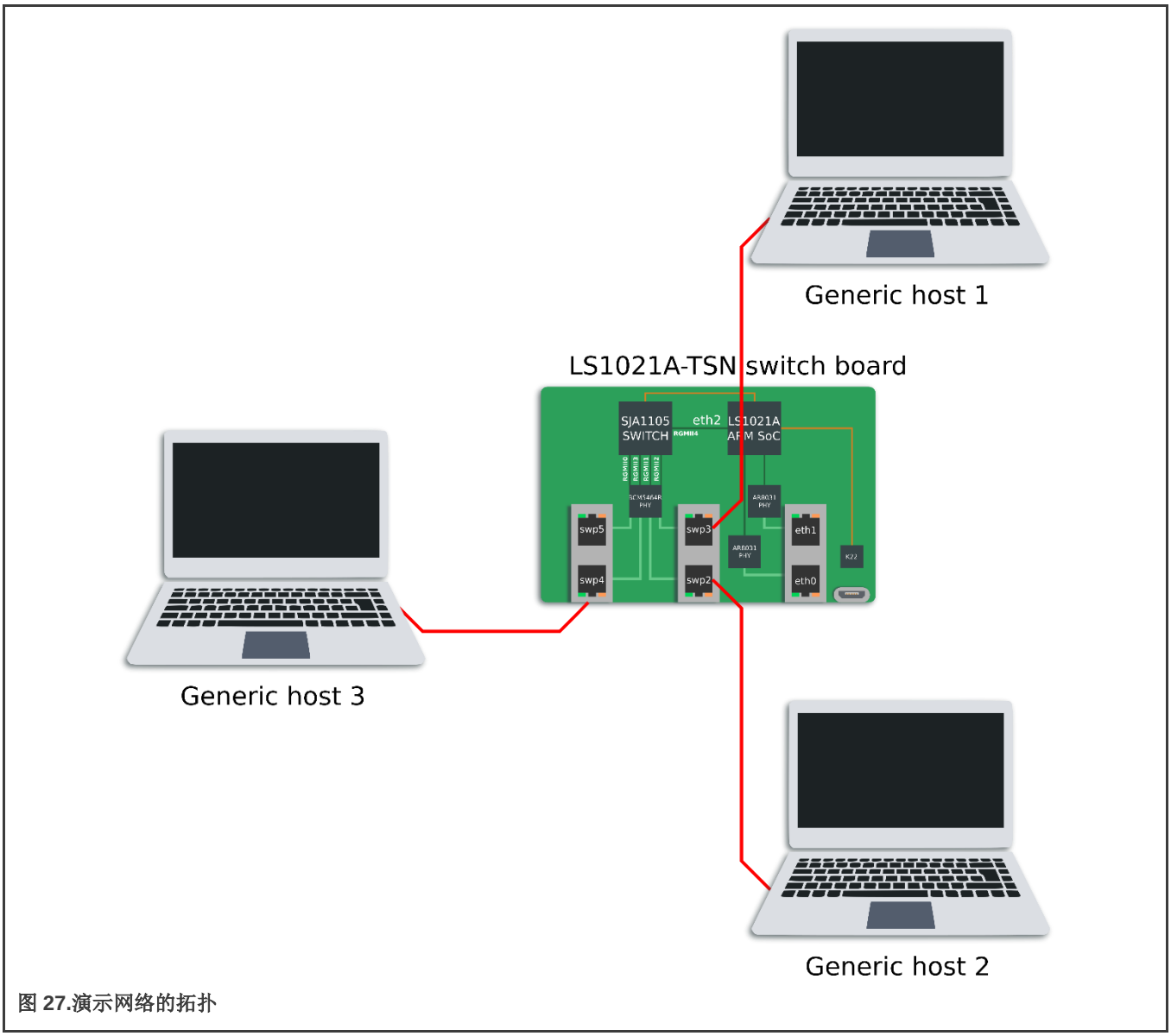


图 27.演示网络的拓扑

4.1.5.2 SJA1105 Linux 支持

使用分布式交换机架构(DSA)框架的实时边缘 Linux 内核支持 SJA1105 交换芯片（其概述可参见 https://netdevconf.info/2.1/papers/distributed-switch-architecture.pdf）。

以下内核配置选项可用于控制其功能：

- CONFIG_NET_DSA_SJA1105：使能基本支持，将 SJA1105 端口作为 4 个能够发送和接收流量的独立网络设备进行探测
- CONFIG_NET_DSA_SJA1105_PTP：使能对 PTP 硬件时钟(PHC)的额外支持，在 LS1021A-TSN 板上的/dev/ptp1 中可见，以及用于 SJA1105 端口上的 PTP 时间戳
- CONFIG_NET_DSA_SJA1105_TAS：使能对时间感知调度程序(TAS)的额外支持，该调度程序通过 tc-taprio qdisc 分流进行配置

此内核驱动程序的文档位于 https://www.kernel.org/doc/html/latest/networking/dsa/sja1105.html。下面列出了几个驱动程序功能。

LS1021A-TSN 设备树(arch/arm/boot/dts/ls1021a-tsn.dts)将 sja1105 端口名称定义为 swp2、swp3、swp4 和 swp5。这些数字与机箱标签 ETH2、ETH3、ETH4 和 ETH5 直接对应。ETH2 机箱标签（在 Linux 中由 swp2 网络设备表示）不应与 eth2 网络设备混淆，后者表示此交换机的 LS1021A 主机端口（称为 DSA master）。

在 LS1021A-TSN 板上，网络管理由 systemd-networkd 守护进程完成，其配置文件位于/etc/systemd/network/。在此板上，默认情况下存在以下 systemd-networkd 配置文件：

- br0.netdev：创建一个禁用 VLAN 过滤、禁用 STP 和禁用 MVRP 的网桥网络设备
- br0.network：将从属于 br0 网络的设备配置为通过 DHCP 请求 IPv4 地址
- eth0.network、eth1.network、swp.network：将 LS1021A-TSN 板的所有 6 个端口配置为同一个 br0 网桥的一部分（4 个端口在硬件中桥接，2 个端口在软件中桥接）
- eth2.network：将 DSA master 端口配置为自动启动，并为其分配一个虚拟链路本地 IP 地址。启动 DSA master 接口是使用交换网络设备的要求。

尽管默认情况下所有端口都配置为 L2 转发（因此该板的唯一 IP 地址应分配给 br0），这可以通过从/etc/systemd/network/中的文件中删除“Bridge=br0”行来更改，然后运行“systemctl restart systemd-networkd”。

在独立模式下，每个 SJA1105 端口都能够获取 IP 地址，并将通用数据包传输到内核或从内核传输。通过将 VLAN 标记功能重新用于交换机端口分离和识别，内核驱动程序在内部支持该操作。因此，只有在用户不通过网桥 vlan_filtering 选项请求 VLAN 标记时，才能支持通用流量 I/O。发生这种情况时，交换机驱动程序进入缩减功能模式，其中 swpN 网络设备不能再向内核/从内核发送和接收通用数据包。这是一个硬件限制，可以通过使能 best_effort_vlan_filtering devlink 参数（按照内核文档中的步骤）得到一定程度的缓解。

实际上还有第二种帧标记机制，它适用于 STP 和 PTP 流量，不依赖于 VLAN 标记。因此，STP 和 PTP 协议在 sja1105 驱动程序上仍可运行，即使端口从属于 vlan_filtering=1 的网桥。

当禁用 VLAN 感知时，sja1105 端口不会检查 VLAN 端口成员资格或 PCP，也不会更改 VLAN 标记。对于这些操作，需要以下命令：

```
ip link set dev br0 type bridge vlan_filtering 1
```

使能 VLAN 过滤后，可以使用 iproute2 包中的“bridge vlan”命令检查和修改每个交换机端口的 VLAN 表。

可以使用以下命令在网桥上启动 STP 状态机：

```
ip link set dev br0 type bridge stp_state 1
ip link set dev br0 down
ip link set dev br0 up
```

可以使用“bridge fdb”命令集检查和修改交换机 L2 地址转发数据库(FDB)。

可以使用 ethtool -S swpN 命令检查端口统计计数器。

可以使用以下命令将 sja1105 端口 MTU 配置为最大 2021：

```
ip link set dev swp2 mtu 2000
```

可以通过以下命令集配置 sja1105 端口上的端口镜像（入口和/或出口数据包的镜像）：

```
tc qdisc add dev swp2 clsact
tc filter add dev swp2 ingress matchall skip_sw \
    action mirred egress mirror dev swp3
```

```
tc filter show dev swp2 ingress
tc filter del dev swp2 ingress pref 49152
```

sja1105 驱动程序当前支持 3 种类型的监管器：

- 端口监管器：这些会影响端口上传入的所有流量，但符合更具体规则的流量除外（见下文）。这些配置如下：

```
tc qdisc add dev swp2 clsact
tc filter add dev swp2 ingress matchall skip_sw \
  action police rate 10mbit burst 64k
```

- 流量类别监管器：这些仅影响具有特定 VLAN PCP 的流量。仅在端口 swp2 上将 VLAN PCP 0 的流量（也包括未标记的流量）限制为 100 Mbit/s：

```
tc qdisc add dev swp2 clsact
tc filter add dev swp2 ingress protocol 802.1Q flower skip_sw \
  vlan_prio 0 action police rate 100mbit burst 64k
```

- 广播监管器：这些仅影响在入口端口上接收的广播流量（目标 MAC ff:ff:ff:ff:ff:ff）。

```
tc qdisc add dev swp2 clsact
tc filter add dev swp2 ingress flower skip_sw dst_mac ff:ff:ff:ff:ff:ff\
  action police rate 10mbit burst 64k
```

在没有分配给流量类别或广播流量的特定监管器的情况下，这些数据包将消耗端口监管器的带宽预算。

还可以组合流量类别的带宽分配，或多个端口上的广播流量，并将它们分配给单个监管器。此功能称为“共享过滤块”，可按如下方式配置（以下示例将来自所有交换机端口的广播流量限制为总计 10 Mbit/s）：

```
tc qdisc add dev swp2 ingress_block 1 clsact
tc qdisc add dev swp3 ingress_block 1 clsact
tc qdisc add dev swp4 ingress_block 1 clsact
tc qdisc add dev swp5 ingress_block 1 clsact
tc filter add block 1 flower skip_sw dst_mac ff:ff:ff:ff:ff:ff \
    action police rate 10mbit burst 64k
```

对于 PTP，sja1105 驱动程序实现了与 linuxptp 和其他开源应用程序协议栈互操作所需的内核原语。LS1021A-TSN 上的实时边缘配置为在端口 swp2、swp3、swp4 和 swp5 上默认以 802.1AS 网桥模式启动 linuxptp。涉及以下系统组件：

- ptp4l：实施 IEEE 1588/802.1AS 状态机的守护进程。通过/etc/linuxptp.cfg 文件配置，并通过 linuxptp.service systemctl 服务进行控制。
- phc2sys：将系统时间(CLOCK_REALTIME)同步到活动的 PHC(/dev/ptp1)或反之亦然的守护程序，具体取决于网络中的板角色（PTP master 或 slave）。通过/etc/linuxptp-system-clock.cfg 文件配置，并通过 phc2sys.service systemctl 服务进行控制。

要查看单板的 PTP 同步状态，可以使用以下命令：

```
systemctl start --now ptp4l
systemctl start --now phc2sys
journalctl -b -u ptp4l -f
journalctl -b -u phc2sys -f
```

在稳定状态下，交换机端口预计与 PTP master 保持同步偏移量 +/- 100 ns。

正常运行时，sja1105 的静态配置需要由驱动程序来更改。进而，这需要复位交换机，这会暂时中断以太网流量和 PTP 同步。交换机复位后，PTP 同步偏移可能会跳到+/- 2 500 000 ns 的更高瞬时范围。sja1105 内核驱动中的复位原因列表为：

- 通过“ip link”命令使能或禁用 VLAN 过滤。
- 使能或禁用 PTP 时间戳。
- 配置老化时间（当 STP 处于活动状态时，由内核 STP 状态机自动完成）。
- 通过 tc-taprio 命令配置时间感知调度程序。
- 配置 L2 监管器（用于 MTU 或监管）。

4.1.5.3 同步 802.1Qbv 演示

本演示的目标如下：

- 使用 IEEE 802.1AS 同步 SJA1105 PTP 时钟。
- 基于 PTP 时钟运行 SJA1105 时间感知调度程序（802.1Qbv 引擎）。
- 对具有需要确定性延迟的流创建小型交换 TSN 网络。证明延迟不受干扰流量的影响。

在本章前面介绍的拓扑中，需要通过 PTP 同步的板是主机 1、2 和 LS1021A-TSN 板。主机 3 只产生 iperf 流量，对时间不敏感。

使用主机 1 和 2 上的 802.1AS 配置文件启动 PTP 同步需要以下命令：

```
ptp4l -i eth0 -f /etc/linuxptp/gPTP.cfg -m
phc2sys -a -rr --transportSpecific 0x1 --step_threshold 0.0002 --first_step_threshold 0.0002
```

预计两台主机上的输出不同。一块板将成为 PTP grandmaster 并显示以下日志：

- ptp4l：

```
ptp4l[13.067]: port 1: link up
ptp4l[13.104]: port 1: FAULTY to LISTENING onINIT_COMPLETE
ptp4l[16.113]: port 1: LISTENING to MASTER on ANNOUNCE_RECEIPT_TIMEOUT_EXPIRES
ptp4l[16.113]: selected local clock 00049f.fffe.05de06 as bestmaster
ptp4l[16.113]: port 1: assuming the grand master role
ptp4l[16.692]: port 1: new foreign master 001f7b.fffe.630248-1
ptp4l[16.692]: selected best master clock 00049f.fffe.05f627
ptp4l[16.692]: port 1: assuming the grand master role
```

- phc2sys：

```
phc2sys[73.382]: eno0 sys offset        12 s2 freq  +2009 delay   1560
phc2sys[74.382]: eno0 sys offset         2 s2 freq  +2003 delay   1560
phc2sys[75.382]: eno0 sys offset       -18 s2 freq  +1983 delay   1600
phc2sys[76.383]: eno0 sys offset        27 s2 freq  +2023 delay   1600
phc2sys[77.383]: eno0 sys offset         7 s2 freq  +2011 delay   1600
phc2sys[78.383]: eno0 sys offset       -18 s2 freq  +1988 delay   1560
phc2sys[79.383]: eno0 sys offset        -8 s2 freq  +1993 delay   1560
```

而另一块板将成为 PTP slave，如以下日志所示：

- ptp4l：

```
ptp4l[68484.668]: rms    17 max      36 freq  +1613 +/-  15 delay       737 +/-
0
ptp4l[68485.668]: rms    8 max       15 freq  +1622 +/-  11 delay       737 +/-
0
```

```
ptp4l[68486.669]: rms     14 max   28 freq     +1643 +/-    13  delay  737 +/-   0
ptp4l[68487.670]: rms     11 max   17 freq     +1650 +/-    10  delay  738 +/-   0
ptp4l[68488.671]: rms     11 max   20 freq     +1633 +/-    15  delay  738 +/-   0
ptp4l[68489.672]: rms     8 max    16 freq     +1640 +/-    11  delay  737 +/-   0
ptp4l[68490.673]: rms     16 max   32 freq     +1640 +/-    23  delay  737 +/-   0
ptp4l[68491.674]: rms     12 max   21 freq     +1622 +/-    13  delay  737 +/-   0
ptp4l[68492.675]: rms     13 max   19 freq     +1648 +/-    13  delay  738 +/-   0
ptp4l[68493.676]: rms     18 max   34 freq     +1668 +/-    15  delay  737 +/-   0
```

- phc2sys：

```
phc2sys[68508.790]:  CLOCK_REALTIME  phc  offset       10  s2  freq   -342  delay  1600
phc2sys[68509.791]:  CLOCK_REALTIME  phc  offset        2  s2  freq   -347  delay  1560
phc2sys[68510.791]:  CLOCK_REALTIME  phc  offset        9  s2  freq   -339  delay  1600
phc2sys[68511.791]:  CLOCK_REALTIME  phc  offset      -22  s2  freq   -368  delay  1560
phc2sys[68512.791]:  CLOCK_REALTIME  phc  offset      -19  s2  freq   -371  delay  1560
phc2sys[68513.791]:  CLOCK_REALTIME  phc  offset      -13  s2  freq   -371  delay  1560
phc2sys[68514.791]:  CLOCK_REALTIME  phc  offset       48  s2  freq   -314  delay  1560
phc2sys[68515.792]:  CLOCK_REALTIME  phc  offset       22  s2  freq   -325  delay  1560
phc2sys[68516.792]:  CLOCK_REALTIME  phc  offset       17  s2  freq   -324  delay  1560
phc2sys[68517.792]:  CLOCK_REALTIME  phc  offset      -29  s2  freq   -365  delay  1560
```

LS1021A-TSN 板的作用是将 PTP 时间从 802.1AS grandmaster 中继到 slave。它在连接到 GM 的端口上充当 slave，在连接到其他主机的端口上充当 master。

```
[root] # journalctl -b -u ptp4l -f
-- Logs begin at Tue 2020-04-07 14:02:11 UTC.--
ptp4l[86640.528]: rms  10 max  23 freq -19731 +/- 11 delay   737 +/- 0
ptp4l[86641.528]: rms  9 max   15 freq -19740 +/- 13 delay   736 +/- 0
ptp4l[86642.529]: rms  12 max  19 freq -19757 +/- 10 delay   737 +/- 0
ptp4l[86643.530]: rms  9 max   14 freq -19747 +/- 13 delay   737 +/- 0
ptp4l[86644.530]: rms  13 max  22 freq -19733 +/- 15 delay   736 +/- 0
ptp4l[86645.531]: rms  7 max   14 freq -19735 +/-  9 delay   737 +/- 0
ptp4l[86646.532]: rms  7 max   13 freq -19735 +/-  9 delay   737 +/- 0
ptp4l[86647.532]: rms  11 max  19 freq -19750 +/- 12 delay   737 +/- 0
ptp4l[86648.533]: rms  6 max   14 freq -19745 +/-  8 delay   737 +/- 0
ptp4l[86649.534]: rms  9 max   15 freq -19750 +/- 12 delay   736 +/- 0
```

以上信息可以解释如下（这里只解释最后一行）：

- 因为/etc/linuxptp.cfg 中的默认（隐式）summary_interval 为 0（每秒打印一次统计信息），而 802.1AS 所需的 logSyncInterval 为-3（同步消息以 1/8 秒的间隔发送——125 ms），这意味着同步统计信息不能完整打印（对于每个数据包），而是以缩写形式打印（日志中没有“偏移量”）。
- 到 master 的偏移量的均方根值为 9 ms，过去 1 秒内的最大值为 15 ns。
- 与 GM 同步所需的频率校正平均为 -19750 十亿分率(ppb)。如果频率调整超过某个健全阈值（取决于内核驱动程序），ptp4l 可能会打印“时钟检查”警告并停止同步。有时可以通过运行以下命令将 PTP 时钟频率调整重置为零，手动解决此问题：

```
phc_ctl /dev/ptp0 freq 0
```

- 其设备与其链路伙伴之间的测量路径延迟（MAC 到 MAC 传播延迟，1Gbps 时约 70 字节帧）恰好为 736 ns。

此网络中的时钟分布树如下：PTP GM（例如主机 1）的系统时钟使用 phc2sys 约束其 PTP 硬件时钟(/dev/ptp0)。通过以太网，PTP GM 约束 SJA1105 PHC，后者约束 PTP slave（例如主机 2）。在 slave 主机上，phc2sys 进程反向运行，将系统时钟(CLOCK_REALTIME)约束到 PTP 硬件时钟(/dev/ptp0)。

关于在这种情况下使用 LS1021A-TSN 板作为 gPTP GM 的说明（代替主机 1）。在这块板上没有电池支持的 RTC，因此板上没有持久的时间源。必须依靠 NTP 服务(ntpd.service)来提供时间，否则 1970 年的时间将被中继到 PTP 网络中。

在 slave 主机上使用 phc2sys 的说明。由于 phc2sys 尝试约束 CLOCK_REALTIME，因此必须手动确保系统中的其他守护进程不会尝试执行相同的操作，例如 ntpd。否则 phc2sys 和其他守护进程之间将存在访问冲突，并且 phc2sys 会继续打印时钟检查警告消息。

将以下时间表安装到向主机 2 出口的 sja1105 端口中：

```
tc qdisc add dev swp2 parent root taprio \
      num_tc 8 \
      map 0 1 2 3 4 5 6 7 \
      queues 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 \
      base-time 0 \
      sched-entry S 80  50000 \
      sched-entry S 40 50000 \
      sched-entry S 3f 300000 \
      flags 2
```

基准时间为 0 表示网络时间表的相位偏移。该时间对应于 1970 年 1 月 1 日，但它会自动提前到与即将到来的 PTP 未来等效的时间（提前整数个周期时间纳秒）。

此示例中的周期时间未明确提供，但它通过计算所有门事件持续时间的总和得出：400 微秒(us)。

swp2 出口的时间表划分如下：

- PTP 流量(S 80)为 50 us。链路本地管理流量（STP、PTP 等）的流量类别分配 7 在驱动程序级别固定为 7，此时用户无法配置。
- 流量类别 6(S 40)为 50 us。延迟敏感型流量生成器将注入此窗口。
- 所有其他流量界别 0-5(S 3f)为 300 us。

通过运行以下命令，在基于 sja1105 交换芯片的 VLAN PCP 上使能 QoS 分类：

```
ip link set dev br0 type bridge vlan_filtering 1
```

首先需要在主机 2 上启动延迟敏感型流量的接收器。该进程等待来自发送器的连接，然后将其统计信息传输给它。

```
ip addr add 192.168.1.2/24 dev eth0
isochron rcv --interface eth0 --quiet
```

在主机 1 上启动发送器，如下所示：

```
ip addr add 192.168.1.1/24 dev eth0
isochron send --interface eth0 --dmac 00:04:9f:05:de:06 --priority 6 --vid 0 \
         --base-time 0 --cycle-time 400000 --shift-time 50000 --advance-time 90000 \
         --num-frames 10000 --frame-size 64 --client 192.168.1.2 --quiet
```

日志应如下所示：

```
Base time 0.000040000 is in the past, winding it into the future
        Now: 1586282691.751150218
 Base time: 1586282691.751160000
Cycle time: 0.000400000
Collecting receiver stats
Summary:
Path delay: min 4329 max 4444 mean 4387.987 stddev 24.508
HW TX deadline delta: min -65238 max -18938 mean -59707.395 stddev 1371.995
SW TX deadline delta: min -33528 max 25058 mean -28221.001 stddev 1844.235
HW RX deadline delta: min -60874 max -14529 mean -55319.408 stddev 1372.222
SW RX deadline delta: min -43398 max 130659 mean -38212.966 stddev 2514.592
HW TX deadline misses: 0 (0.000%)
SW TX deadline misses: 1 (0.010%)
```

有必要进行以下澄清：

- 目标 MAC 是主机 2 的接口 eth0 的 MAC
- 发送的数据包具有带 VID 0 和 PCP 6 的 VLAN 标记。因为它们带有优先级标记(802.1p)，所以 sja1105 交换芯片端口将接受这些数据包，而无需任何“bridge vlan add vid 0 dev swp3”命令。
- 等时线程序以 400 us 的间隔发送 10000 帧。基准时间与 sja1105 出口端口 swp2 上的相同，但它向右移动了 50 us，以便与流量类别 6 窗口的开始（这是时间表中的第二个时隙）对齐。因此，数据包传输的最后期限为（基准时间 + 移位时间 + N * 循环时间）。
- 实际上，数据包必须在 TX 截止时间之前传输，以补偿 Linux 内核中的调度延迟和数据包的实际传播延迟。所以等时线程序会一直睡眠到下一个截止时间前的 90 us。
- 通过“将基准时间绕接到未来”，可以理解原始基准时间(0)增加最小周期数 N 的过程，这使其大于当前 PTP 时间(1586282691.751150218)。在这种情况下，新的基准时间是 1586282691.751160000。
- 对于每个数据包，发送器收集 2 个 TX 时间戳：一个硬件和一个软件。接收器还收集两个时间戳。这些时间戳不会打印到控制台，因为指定了--quiet 选项。
- 发送器和接收器的时间戳之间的关联是通过辅助套接字来完成。接收器在 TCP 端口 5000 上等待连接，并将其日志传输给发送器，发送器通过使用由{sequence number, scheduled TX time(deadline)}形成的密钥与自己的日志相关联。这两个值都嵌入到数据包有效负载中。如果省略--client 选项，则不执行统计关联。此 TCP 套接字是此网络中需要进行 IP 通信的唯一原因。
- 路径延迟通过接收器的 RX 硬件时间戳和发送器的 TX 硬件时间戳之间的增量计算得出。
- 每个“截止时间增量”都均通过时间戳和该数据包的计划发送时间之间的差值计算得出。硬件 TX 截止时间增量应始终为负数，因为这表明数据包是在计划的 TX 时间到期之前发送的。在这种情况下，软件 TX 时间戳在硬件 TX 时间戳之后获取，因此它们的含义与此驱动程序不太相关。802.1Qbv 时间表安装在 sja1105 交换芯片端口上后，RX 截止时间增量就会变得相关。

上述日志是在 sja1105 端口上没有活动的 802.1Qbv 时间表且没有后台流量的情况下获取的。启动后台流量后：

```
# Host 2
iperf3 -s > /dev/null &
sysctl -w kernel.sched_rt_runtime_us=-1
chrt --fifo 90 isochron rcv -i eth0 --quiet
# Host 3
ip addr add 192.168.1.3/24 dev eth0
```

```
iperf3 -c 192.168.1.2 -t 48600
Connecting to host 10.0.0.112, port 5201
[ 5] local 10.0.0.113 port 60360 connected to 10.0.0.112 port 5201
[ ID] Interval               Transfer     Bitrate         Retr    Cwnd
[ 5]    0.00-1.00      sec     105 MBytes     878 Mbits/sec  0       489 KBytes
[ 5]    1.00-2.00      sec     102 MBytes     859 Mbits/sec  0       513 KBytes
[ 5]    2.00-3.00      sec     102 MBytes     858 Mbits/sec  0       513 KBytes
[ 5]    3.00-4.00      sec     101 MBytes     851 Mbits/sec  0       513 KBytes
[ 5]    4.00-5.00      sec     102 MBytes     860 Mbits/sec  0       539 KBytes
```

主机 1 生成的等时线流量的重新运行如下所示：

```
chrt --fifo 90 isochron send -i eno0 -d 00:04:9f:05:de:06 -p 6 -v 0 -b 0 -S 50000 -c 400000 -a 90000
-n 10000 -s 64 -C 10.0.0.112 -q
Base time 0.000040000 is in the past, winding it into the future
        Now: 1586286409.635121693
 Base time: 1586286409.635160000
Cycle time: 0.000400000
Collecting receiver stats
Summary:
Path delay: min 4314 max 16774 mean 9725.688 stddev 3919.150
HW TX deadline delta: min -64273 max -8538 mean -59894.931 stddev 1467.284
SW TX deadline delta: min -33286 max 37575 mean -28498.114 stddev 2006.546
HW RX deadline delta: min -58924 max -904 mean -50169.243 stddev 4183.042
SW RX deadline delta: min -52757 max 1109472 mean -29436.032 stddev 23537.847
HW TX deadline misses: 0 (0.000%)
SW TX deadline misses: 4 (0.040%)
```

可以看出，由于数据包在 iperf3 生成的 MTU 大小的数据包完成传输之前等待的时间延长，路径延迟方差增加。

最后，在交换机上安装 802.1Qbv 时间表会对等时线计算的所有统计数据产生影响：

```
chrt --fifo 90 isochron send -i eno0 -d 00:04:9f:05:de:06 -p 6 -v 0 -b 0 -S 50000 -c 400000 -a 90000
-n 10000 -s 64 -C 10.0.0.112 -q
Base time 0.000040000 is in the past, winding it into the future
        Now: 1586286689.223100936
 Base time: 1586286689.223160000
Cycle time: 0.000400000
Collecting receiver stats
Summary:
Path delay: min 14199 max 65684 mean 61357.368 stddev 1494.831
HW TX deadline delta: min -64128 max -12643 mean -59822.445 stddev 1494.557
SW TX deadline delta: min -33616 max 25621 mean -28448.709 stddev 1974.185
HW RX deadline delta: min 1476 max 2041 mean 1534.924 stddev 24.822
SW RX deadline delta: min 5243 max 1122800 mean 21040.814 stddev 16752.155
HW TX deadline misses: 0 (0.000%)
SW TX deadline misses: 5 (0.050%)
```

路径延迟增加了，但这是因为现在它包含了交换机上阻塞的数据包等待门 6 打开所花费的时间。

硬件 RX 截止时间增量现在有了新的含义，因为在最后一个示例中（在交换机上使能了 802.1Qbv），门充当屏障并消除了硬件 TX 时间戳中的抖动，这是由发送器操作系统中的调度延迟引起。一般来说，发送器的抖动会在数据包进入 TSN 网络时由第一台交换机消除。效果是接收器会看到低抖动的数据包流。

可以通过减少提前时间来减少路径延迟。它配置为数据包在门打开之前到达交换机，这取决于发送器的抖动。最大程度减少 TX 抖动不在本演示的范围。

4.2 GenAVB/TSN 协议栈

4.2.1 简介

GenAVB/TSN 协议栈为恩智浦 SoC 和硬件平台上的音视频桥接(AVB)和时间敏感网络(TSN)功能提供高级实施。这些功能需要在 LS1028A 和 i.MX 8M Plus SoC 中提供的 TSN 硬件支持。

本节提供有关如何设置和评估 GenAVB/TSN 协议栈的信息。在这种情况下，它提供了有关支持的 SoC 和电路板、编译时软件包配置和运行时配置设置的信息。

4.2.1.1 gPTP 协议栈

gPTP 协议栈实施 IEEE 802.1AS-2020 标准，并支持时间感知端点和网桥系统。该协议栈完全在用户空间中运行，使用 Linux 套接字 API 进行数据包传输、接收和加盖时间戳。Linux 时钟 API 用于时钟调整。配置文件用于在初始化时配置协议栈，并在运行时提供大量日志记录。

4.2.1.2 SRP 协议栈

SRP 协议栈实施了 IEEE 802.1Q-2018 第 10、11 和 35 节中定义的 MRP、MVRP 和 MSRP。该协议栈完全在用户空间中运行，使用 Linux 套接字 API 进行数据包发送和接收。Linux tc 和网桥网络链路 API 用于更新多播 FDB 条目和 FQTSS 基于信用的整形器(CBS)配置。配置文件用于在初始化时配置协议栈，并在运行时提供大量日志记录。

4.2.1.3 TSN 端点示例应用

TSN 示例应用提供了执行 GenAVB/TSN API 的示例代码和可重用中间件。它用于练习和验证本地系统的实时行为以及端点之间网络的 TSN 属性。

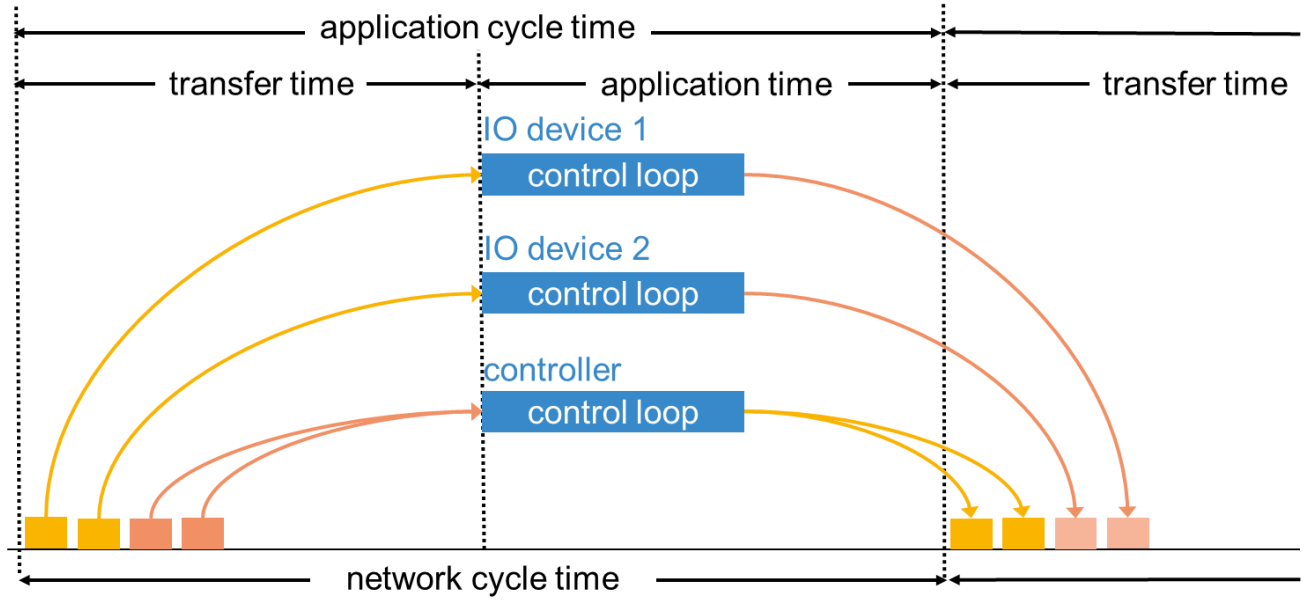


图 28.TSN 应用周期

TSN 示例应用实施了一个控制回路，类似于需要通过网络进行循环同步交换的工业用例。

TSN 端点在同一 gPTP 功能域中将其应用同步到一个公共时间网格，以便它们可以用循环同步模式发送和接收网络流量（应用循环时间与网络循环时间相等并同步，如上图所示）。目前该周期配置为 2ms 的周期。当调度应用时，来自其他端点的帧准备好被读取，并且在应用时间帧结束时发送到其他端点。

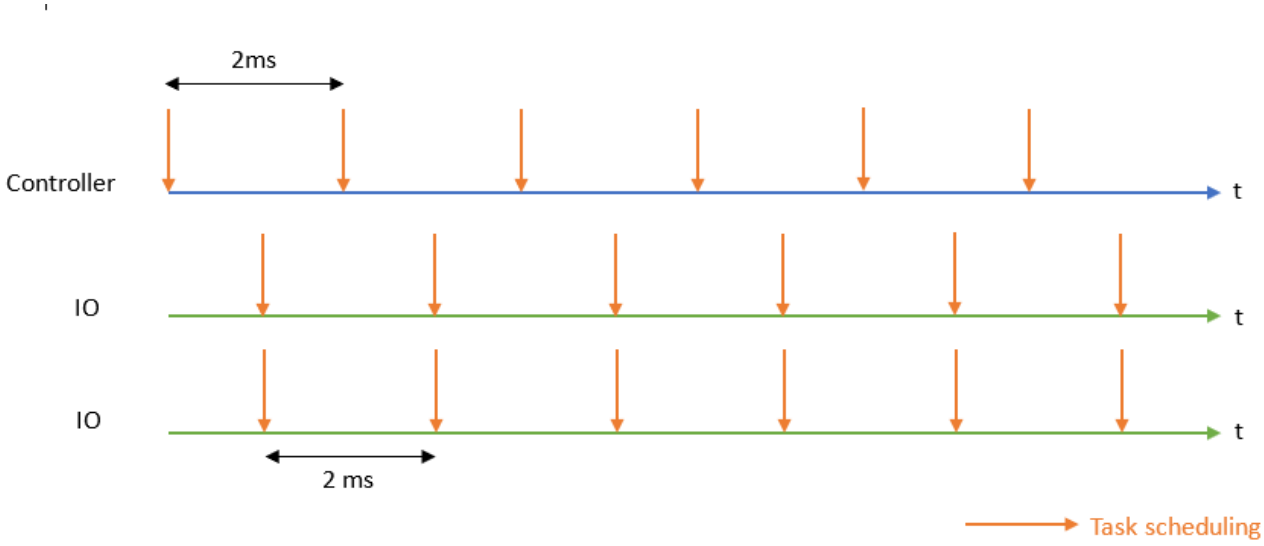


图 29.TSN 应用调度

如上图所示，控制器和 IO 设备以半周期偏移进行调度，以减少处理延迟。

时间敏感型流量是带 VLAN 标头和专有 Ether 类型的第 2 层多播。其优先级使用 VLAN 标头的 PCP 字段定义。

此外，TSN 应用提供详细的日志和时间敏感型流量时序统计信息（基于数据包的硬件时间戳），从而可以表征整个实时分布式系统。

最后，实现了一个 OPCUA 服务器，并可以浏览和检索作为 OPCUA 对象公开的 TSN 应用统计数据。OPCUA 服务器通过 TCP 运行并允许访问任何 OPCUA 客户端。

4.2.1.4 支持的配置

GenAVB/TSN 协议栈目前支持以下板和相关角色：

- LS1028ARDB：gPTP 时间感知网桥和 SRP 网桥
- i.MX 8M Plus EVK：gPTP 时间感知端点站和 TSN 端点示例应用

该协议栈在以下 Yocto 实时边缘机器中使能并对其提供支持：

- imx8mpevk
- ls1028ardb

4.2.2 通过 Yocto 构建镜像

在为以下机器进行构建时，GenAVB/TSN 包 `genavb-tsn` 默认包含在 Yocto 实时边缘镜像 `nxp-real-time-edge` 中：

- imx8mpevk
- ls1028ardb

按照实施边缘软件 Yocto Project 的说明，获取代码并设置构建环境。

运行以下命令为这些机器构建镜像：

```
$ cd yocto-real-time-edge
```

对于 LS1028ARDB 镜像：

```
$ DISTRO=nxp-real-time-edge MACHINE=ls1028ardb source real-time-edge-setup-env.sh -b build-ls1028ardb
```

对于 i.MX 8M Plus EVK 镜像：

```
$ DISTRO=nxp-real-time-edge MACHINE=imx8mpevk source real-time-edge-setup-env.sh -b build-imx8mpevk
```

然后，使用：

```
$ bitbake nxp-image-real-time-edge
```

4.2.3 GenAVB/TSN 协议栈启动/停止

可以使用以下命令在运行时手动启动/停止 GenAVB/TSN 协议栈：

```
# avb.sh <start|stop>
```

这将启动所有协议栈组件和示例应用。通过在文件`/etc/genavb/config` 中将变量 CFG_AUTO_START 设置为 1，可以在系统启动期间自动启动协议栈。

要启动/停止 gPTP 协议栈，请使用：

```
# fgptp.sh <start|stop>
```

4.2.4 用例说明

4.2.4.1 gPTP 网桥

LS1028ARDB 可用作通用时间感知网桥，连接到其他时间感知终端节点或网桥。

默认情况下，如果 Linux 下没有配置网桥接口，LS1028ARDB 不会转发数据包。是否使能网桥接口取决于所使用的板。例如，如何在 LS1028ARDB 上配置网桥接口，请参见 Linux 交换机配置一节。

启动 gPTP 协议栈后，可以使用以下命令显示日志：

```
# tail -f /var/log/fgptp-br
```

在此日志文件中，可以观察到哪些端口已连接，哪些端口当前正在传送同步时间，以及端口在时间感知系统中的作用。

如果网桥的一个端口连接到另一个能够传送同步时间的端口，则每个使能的 gPTP 功能域都应显示以下日志：

```
gptp_stats_dump : Port(1) domain(0,0): Role: Master Link: Up asCapable: Yes neighborGptpCapable: Yes
...
gptp_stats_dump : Port(1) domain(1,20): Role: Master Link: Up asCapable: Yes neighborGptpCapable: Yes
```

Role 状态也可以根据 Grandmaster 参数一节中所述的*参数*取值 *Slave*。

如果端口未连接，则 *Link* 状态取值 *Down*。

如果端口无法传送同步时间，则 *AS_Capable* 状态将采用值 *No*。

有关 gPTP 日志的更多详细信息，请参见 gPTP 网桥一节。

4.2.4.2 gPTP 端点

启动 gPTP 协议栈后，可以使用以下命令显示日志：

```
# tail -f /var/log/fgptp
```

在此日志文件中，可以观察到端口在时间感知系统中的作用。

如果端点的端口连接到另一个能够传送同步时间的端口，则每个 gPTP 功能域都应显示以下日志：

```
gptp_stats_dump : Port(0) domain(0,0): Role: Slave Link: Up AS_Capable: Yes neighborGptpCapable: Yes
...
gptp_stats_dump : Port(0) domain(1,20): Role: Slave Link: Up AS_Capable: Yes neighborGptpCapable: Yes
```

Role 状态也可以根据 Grandmaster 参数一节中所述的 *Grandmaster 参数*取值 *Master*。

如果端口未连接，则 *Link 状态*取值 *Down*。

如果端口无法传送同步时间，则 *AS_Capable* 状态将采用值 *No*。

有关 gPTP 日志的更多详细信息，请参见 gPTP 端点一节。

4.2.4.3 gPTP 多功能域

此用例说明了在 TSN 网络上独立共存的两个 gPTP 功能域，位于不同的 802.1AS-2020 时间感知系统上。

第一个功能域使用 PTP 时间范围，而第二个功能域使用 ARB（任意）时间范围。

4.2.4.3.1 要求

gPTP 多功能域的参考设置包括：

- 两个 gPTP 端点(i.MX 8M Plus EVK)：EP1-DUT 和 EP2-DUT
- 一个 gPTP 网桥(LS1028ARDB)：BR-DUT

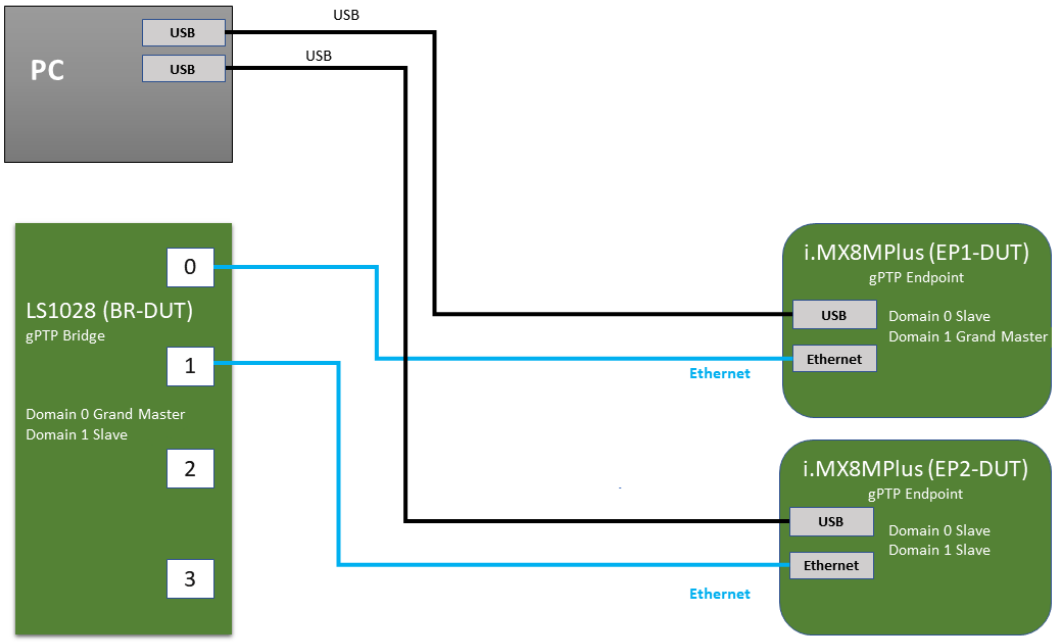


图 30. gPTP 多功能域设置

4.2.4.3.2 gPTP 协议栈配置

gPTP 协议栈可以通过配置文件独立使能或禁用每个功能域。

默认配置文件（例如：`/etc/genavb/fgptp.cfg`）用于通用 gPTP 参数以及功能域 0 参数。要使能其他功能域，必须创建新文件，并在文件名后附加“-N”（例如：`/etc/genavb/fgptp.cfg-1` 用于功能域 1）。

对于 gPTP 多功能域，应更改所有设备配置以支持两个功能域。第一个功能域（功能域 0）必须分配功能域号 0。第二个功能域（功能域 1）分配功能域号 20。

BR-DUT 定义为第一个功能域（功能域 0）的 GrandMaster。EP1-DUT 定义为第二个功能域（功能域 1）的 GrandMaster。

在 EP1-DUT 上，编辑文件`/etc/genavb/fgptp.cfg-1` 并按如下所示更改 domain_number 和 priority1 参数：

```
domain_number = 20
priority1 = 245
```

在 EP2-DUT 上，编辑文件`/etc/genavb/fgptp.cfg-1` 并按如下所示更改 domain_number 参数：

```
domain_number = 20
```

在 BR-DUT 上，编辑文件`/etc/genavb/fgptp-br.cfg-1` 并按如下所示更改 domain_number 参数：

```
domain_number = 20
```

注意

在功能域 0 上，BR-DUT 是功能域内所有设备中优先级最高（最低值）的 GrandMaster（BR-DUT priority1=246、EP1-DUT 和 EP2-DUT priority1=248）

在功能域 1 上，EP1-DUT 是功能域内所有设备中优先级最高（最低值）的 GrandMaster（BR-DUT priority1=246、EP1-DUT priority1=245 和 EP2-DUT priority1=248）

4.2.4.3.3 评估说明

测试步骤

1. 通过发出以下命令在所有 DUT 上手动启动 gPTP 协议栈

```
# fgptp.sh start
```

2. 等待 30 秒
3. 检查 BR-DUT(`/var/log/fgptp-br`)、EP1-DUT 和 EP2-DUT(`/var/log/fgptp`)上的 gPTP 协议栈日志

验证

在第 3 步后，EP1-DUT 上的日志报告端口 0 仅在功能域 0 上已同步：

```
Port(0) domain(0, 0) SYNCHRONIZED - synchronization time (ms):250
```

在第 3 步后，EP2-DUT 上的日志报告端口 0 在所有功能域上已同步：

```
Port(0) domain(0, 0) SYNCHRONIZED - synchronization time (ms):250
```

```
Port(0) domain(1, 20) SYNCHRONIZED - synchronization time (ms):250
```

在第 3 步之后，BR-DUT 上的日志报告端口 0 仅在功能域 1 上已同步：

```
Port(0) domain(1, 20) SYNCHRONIZED - synchronization time (ms): 250
```

每个同步功能域（功能域 0 用于 EP1-DUT 和 EP2-DUT，功能域 1 用于 EP2-DUT 和 BR-DUT）只应报告一次“*初始调整*”消息：

```
domain(0,0) Initial Adjustment, offset: 125486471315484 ns, freq_adj: 32764
```

```
domain(1,20) Initial Adjustment, offset: 125455671332661 ns, freq_adj: 16384
```

实现同步后，所有报告的时钟偏移平均值应稳定在-50 到+50 ns 范围内（功能域 0 用于 EP1-DUT 和 EP2-DUT，功能域 1 用于 EP2-DUT 和 BR-DUT）：

```
domain(0,0) Offset between GM and local clock (ns): min -45 avg 0 max 35
```

```
domain(1,20) Offset between GM and local clock (ns): min -66 avg 0 max 15
```

4.2.4.4 AVB 桥

此用例说明了 AVB 桥（混合 gPTP 和 SRP 协议栈）与其他 AVB 端点

4.2.4.4.1 要求

- 两个 AVB 端点（不是此版本的一部分）
- 1 个 AVB 桥(LS1028ARDB)

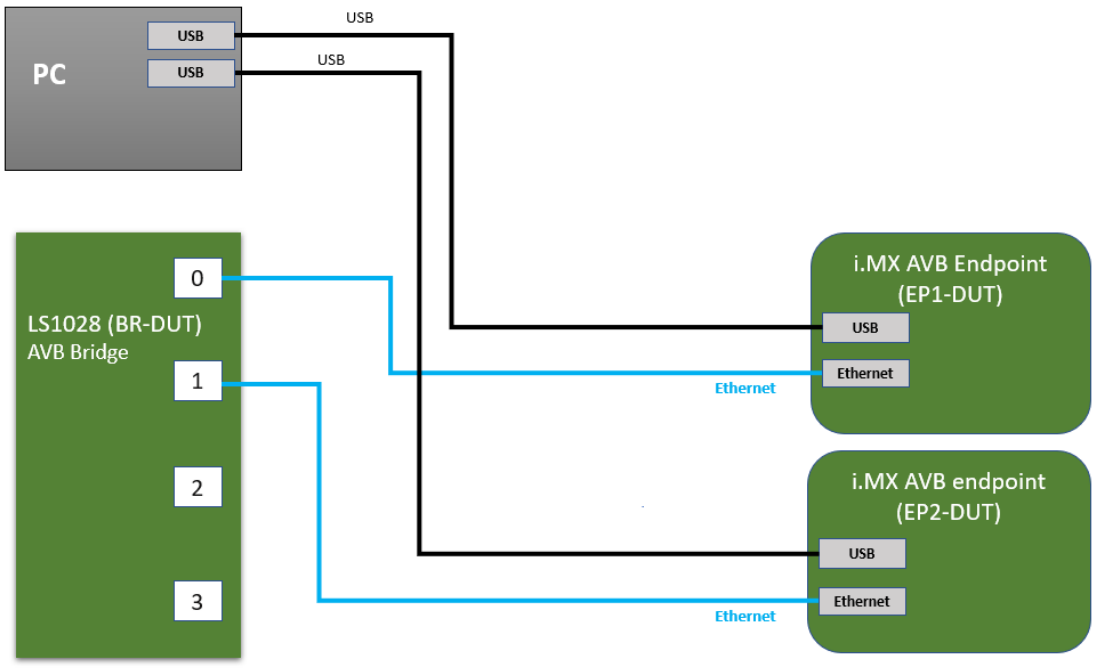


图 31.AVB 桥设置

4.2.4.4.2 AVB 网络配置

本主题介绍 AVB 配置。

4.2.4.4.2.1 流量类别映射的优先级

用于网桥的流量类别被映射的优先级直接来自 IEEE Std 802.1Q-2018 表 34-1 中两个 SR 类别的推荐映射：

表 47.流量类别映射的优先级

优先级	0	1	2	3	4	5	6	7
流量类别	1	0	6	7	2	3	4	5

网桥应根据此映射基于其 PCP 值配置为转发 VLAN 标记的数据包，并应为 SR 类别 A（优先级 3）和 SR 类别 B（优先级 2）流量在两个最高流量类别（流量类别 6 和流量类别 7）上配置基于信用的整形器。

有关网桥 PCP 映射配置，请参见设置准备。

4.2.4.4.2.2 基于 FQTSS 信用的整形器配置

SRP 桥协议栈依靠具有特定句柄的预配置 qdisc，在具有两个最高流量类别的两个硬件队列上为每个端口配置硬件的基于信用的整形器。因此，具有 8 个流量类别的 mqprio qdisc 应配置上述流量类别映射优先级和具有以下句柄的基于信用的整形器 qdisc：0x9006 用于流量类别 6 上的 CBS，0x9007 用于流量类别 7 上的 CBS。

有关网桥 qdisc 配置，请参见设置准备。

4.2.4.4.2.3 Linux“非时间关键型”流量分类

Linux 根据 skb 优先级对出口数据包进行分类，以分配给流量类别。为避免将出口“非时间关键型”流量分配给配置了基于信用的整形器的流量类别，应重写 skb 优先级，以便出口上不存在具有 skb 优先级 2 和 3 的数据包。此外，网桥代码使用 skb 优先级作为从 CPU 端口注入的数据包的流量类别，使得具有 skb 优先级 6 和 7 的数据包最终在外部端口上进入硬件的流量类别 6 和 7，这进而又损害了流量整形。同样，强制重新映射这些 skb 优先级可以避免这种情况。

以下命令是在 `swp0` 端口上重新映射 skb 优先级的示例：

```
# tc qdisc add dev swp0 clsact
# tc filter add dev swp0 egress basic match 'meta(priority eq 2)' or 'meta(priority eq 4)' action skbedit
priority 0
# tc filter add dev swp0 egress basic match 'meta(priority eq 6)' action skbedit priority 4
# tc filter add dev swp0 egress basic match 'meta(priority eq 7)' action skbedit priority 5
```

注意

使用流量控制基本过滤器需要使能 linux 内核配置 `CONFIG_NET_EMATCH_META`

4.2.4.4.2.4 网桥 VLAN 感知

正确的 AVB 桥功能要求交换机仅将 AVB 流（具有多播目标 MAC 地址和特定 VLAN ID）转发到转发数据库(FDB)中配置的端口。为此，我们应该在网桥级别使能 VLAN 过滤，将所需的 VLAN ID 添加到所有端口，并在所有外部端口上禁用默认的多播泛洪配置（至少对于两个最高优先级队列）。

有关网桥 vlan 配置，请参见设置准备。

4.2.4.4.3 设置准备

在每次启动时完成：

1. 设置网桥转发：

```
# ip link set dev eno2 up
# ip link add name br0 type bridge
# ip link set br0 up
# ip link set master br0 swp0 up
# ip link set master br0 swp1 up
# ip link set master br0 swp2 up
# ip link set master br0 swp3 up
```

2. 通过设置 `ES_ANA_PORT_QOS_PCP_DEI_MAP_CFG` 寄存器，为网桥上的每个端口建立 PCP 到 QoS 的映射：

```
# pcp_to_qos_map=([0]="1" [1]="0" [2]="6" [3]="7" [4]="2" [5]="3" [6]="4" [7]="5"); \
for (( i=0; i < 6; ++i )); do \
    for (( pcp=0; pcp < 8; ++pcp )); do \
        devmem2 $((0x1fc287800 + $i * 0x100 + 0x20 + 0x4 * ($pcp + 8 * 0))) w$(($
{pcp_to_qos_map[$pcp]} + 8 * 0)) ;\
        devmem2 $((0x1fc287800 + $i * 0x100 + 0x20 + 0x4 * ($pcp + 8 * 1))) w$(($
{pcp_to_qos_map[$pcp]} + 8 *
    1)) ;\ done ;\
done
```

3. 使用正确的句柄为每个外部端口配置 qdiscs 和整形器：

```
# pcp_to_qos_map=([0]="1" [1]="0" [2]="6" [3]="7" [4]="2" [5]="3" [6]="4" [7]="5"); \
avb_ports="swp0 swp1 swp2 swp3";
\ for port in $avb_ports; do \
    tc qdisc add dev $port root handle 100: mqprio num_tc 8 map ${pcp_to_qos_map[@]} queues
1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 hw 0 ; \
    tc qdisc replace dev $port handle 0x9007 parent 100:8 \
                          cbs locredit -2147483646 hicredit 2147483647 sendslope-1000000
idleslope 0 offload 0 ; \
    tc qdisc replace dev $port handle 0x9006 parent 100:7 \
                          cbs locredit -2147483646 hicredit 2147483647 sendslope-1000000
idleslope 0 offload
0 ; \ done
```

注意

每个端口设备最重要的 CBS 参数是**父级**，它应当与流量类别 6 和 7 匹配，以及**句柄**，它应当是 0x9006 和 0x9007。其他参数只是初始化值，将在运行时流配置时被协议栈覆盖：offload 将设置为 1，以将操作分流到硬件，idleslope 和 sendslope 将根据流和端口比特率设置，信用值将保持在它们的最小值和最大值，因为它们不会直接影响硬件整形操作）

4. 对于每个外部端口，在两个最高队列优先级（通过将 `ES_ANA_ANA_FLOODING` 寄存器设置为 `PGID_CPU 58`）上使能 Vlan 过滤，设置正确的 Vlan ID 并禁用多播泛洪：

```
# ip link set br0 type bridge vlan_filtering 1
# bridge vlan add dev swp0 vid 2 master
# bridge vlan add dev swp1 vid 2 master
# bridge vlan add dev swp2 vid 2 master
# bridge vlan add dev swp3 vid 2 master
# devmem2 0x1fc288a00 w 0x0003CF7A
# devmem2 0x1fc288a04 w 0x0003CF7A
```

5. 启动 AVB 和 gPTP 协议栈：

```
# avb.sh start
```

4.2.4.4.4 评估说明

1. 复位所有端点和网桥。
2. 使用上述步骤，配置网桥并在所有连接的设备（网桥和端点）上启动协议栈
3. 几秒钟后，AVB 端点应通过 gPTP 同步
4. 将 SR 类别 A（或 SR 类别 B）流从 `EP-DUT2`（作为对讲器）连接到 `EP-DUT1`（作为侦听器）：流应正确转发到侦听器端点

4.2.4.4.4.1 gPTP 操作

如果 gPTP 协议在所有设备上都正确运行，则以下行应出现在网桥 gptp 日志文件中，用于连接到支持 gPTP 的设备的每个端口：

```
gptp_stats_dump: Port(0) domain(0, 0): Role: Master  Link: Up    asCapable: Yes neighborGptpCapable: Yes
...
```

```
gptp_stats_dump: Port(1) domain(0, 0): Role: Master  Link: Up       asCapable: Yes neighborGptpCapable:
Yes
```

有关 gPTP 网桥操作的更多详细信息，请参见。

注意

AVB 端点可以配置为以 100 Mbits/s 的固定链路速度运行，并导致 pDelay > 800 ns，使端口不是 asCapable。在这种情况下，将链路速度强制为 1Gbits/s 将使 pDelay 低于阈值并具有适当的 gPTP 操作

4.2.4.4.4.2 SRP 操作

详细查看 SRP 协议通信（功能域声明、SRP 端口边界、对讲器/侦听器声明和注册……）之后，可以从 AVB 桥协议栈日志文件 `/var/log/avb-br` 打印 SRP 特定日志：

```
# tail -f /var/log/avb-br | grep srp
```

在流连接上，FQTSS 和 FDB 操作应打印在 AVB 桥协议栈日志文件中：

- 协议栈日志显示面向 AVB 侦听器的端口的 FQTSS 配置：

```
fqtss_set_oper_idle_slope : logical_port(2) port (swp0, ifindex 5)tc(7)
cbs_qdisc_handle(9007:0): set idle_slope 7872000
```

- 协议栈日志显示面向 AVB 侦听器的端口的 FDB 配置：

```
bridge_rtnetlink : add MDB: bridge (br0, ifindex 9) logical_port(2) port (swp0, ifindex 5)
mac_addr(91:e0:f0:00:fe:11) vlan_id(2)
```

此外，可以使用 Linux 标准工具（`tc` 和 `bridge`）检查相同的配置

- TC 工具显示面向 AVB 侦听器的端口的 FQTSS 配置：

```
# tc qdisc show dev swp0
qdisc mqprio 100: root  tc 8 map 1 0 6 7 2 3 4 5 0 0 0 0 0 0 0 0
            queues:(0:0) (1:1) (2:2) (3:3) (4:4) (5:5) (6:6) (7:7)
qdisc pfifo 0: parent 9006: limit 1000p
qdisc pfifo 0: parent 9007: limit 1000p
qdisc pfifo_fast 0: parent 100:6 bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 11
qdisc pfifo_fast 0: parent 100:5 bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 11
qdisc pfifo_fast 0: parent 100:4 bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 11
qdisc pfifo_fast 0: parent 100:3 bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 11
qdisc pfifo_fast 0: parent 100:2 bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 11
qdisc pfifo_fast 0: parent 100:1 bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 11
qdisc cbs 9006: parent 100:7 hicredit 2147483647 locredit -2147483646 sendslope -1000000
idleslope 0 offload 0
qdisc cbs 9007: parent 100:8 hicredit 2147483647 locredit -2147483648 sendslope -992128
idleslope
7872 offload 1
```

- 网桥工具显示面向 AVB 侦听器的端口的 FDB 配置：

```
# bridge mdb show
dev br0 port swp0 grp 91e0:f000:fe11:: permanent offload vid2
```

4.2.4.5 TSN 端点示例应用程序

4.2.4.5.1 要求

- 两个 TSN 端点(i.MX 8M Plus EVK)
- 一个 TSN 桥(LS1028ARDB)

注意

第二个 IO 设备为可选。

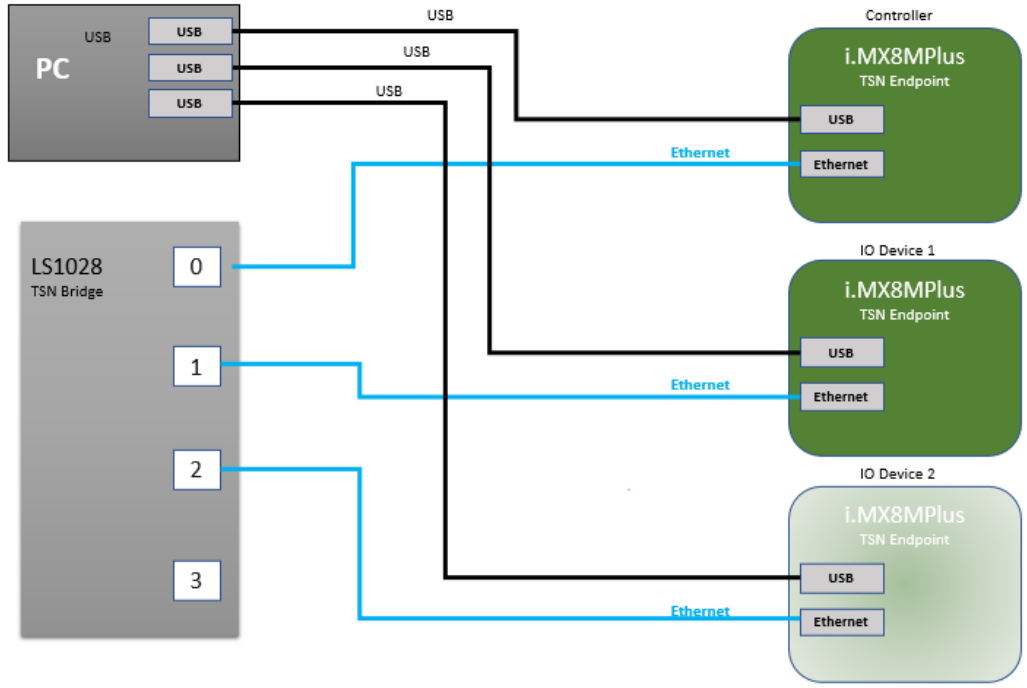


图 32.TSN 端点示例应用程序设置

4.2.4.5.2 配置 GenAVB/TSN 协议栈和示例应用程序

通过修改`/etc/genavb/config`文件，从一种配置更改为另一种配置。该文件指定了一对配置文件：

1. APPS_CFG_FILE (apps-*.cfg)指向一个包含演示配置（要使用的应用程序、选项……）的文件。它由启动脚本 `avb.sh` 解析
2. GENAVB_CFG_FILE (genavb-*.cfg)指向一个包含 GenAVB/TSN 协议栈配置的文件，并由协议栈解析。

配置文件由一对 cfg 文件组成。文件`/etc/genavb/config` 已经将 cfg 文件成对分组。设置 `PROFILE` 变量，以选择所需的配置文件。

4.2.4.5.3 TSN 网络配置

本主题介绍 TSN 配置。

4.2.4.5.3.1 流

流详细信息可用于分析，也可用于计算计划流量时序。

表 48.TSN 流定义

流编号	源	目标	单播/多播	目标 MAC 地址	VLAN ID	Vlan PCP	帧长[1]（字节）
流 1	控制器	IO 设备	多播	91:e0:f0:00:fe:70	2	5	84
流 2	IO 设备 1	控制器	多播	91:e0:f0:00:fe:71	2	5	84
流 3	IO 设备 2	控制器	多播	91:e0:f0:00:fe:80	2	5	84

1.帧长度包括帧间间隙、前导码、帧起始和 CRC（可按原样用于时序计算）

4.2.4.5.3.2 计划流量

对于确定性数据包传输，端点和网桥都需要使用计划流量。

如图 29 所示，TSN 端点示例应用的默认调度配置导致以下流量时间表。

4.2.4.5.3.2.1 端点

端点正在运行一个有 2000us 周期的时间表。时间表的基本偏移量与 gPTP 时间模数 1 秒保持一致。

控制器发送门（用于流 1）在 500us 偏移量时打开（相对于周期开始）。

IO 设备发送门（用于流 2/3）在 1000us + 500us 偏移量时打开（相对于周期开始）。

门打开间隔约为 4us（足以容纳流帧长度加上一些裕量）。

500us 偏移量与将其帧发送到其对等设备的最坏情况应用程序延迟有关。该值为 Linux PREEMPT-RT 系统提供了良好的裕量，但在经过微调的系统上可以降低。

4.2.4.5.3.2.2 网桥

发送上述其中一个流的所有网桥和所有网桥端口的时间表必须具有 2000 µs 的周期，以及与 gPTP 时间模数 1 秒保持一致的基本偏移量。

一种可能的时间表是在偏移量 500 µs 时打开发送门（用于发送流 1 的端口和队列），并使用可适应最差传播延迟的门打开间隔。

也可以使用固定的门打开间隔，但增加沿流路径的每一跳的发送时间偏移量。

对于发送流 2 和 3 的端口和队列，在偏移量 1000 + 500 µs 时打开发送门。

4.2.4.5.4 设置准备

需要将其中一个 TSN 端点配置为“控制器”，另一个配置为“IO 设备”。两个端点都连接到 TSN 桥。

4.2.4.5.4.1 准备控制器

只需完成一次：

1. 在 Linux 提示符下使用以下命令编辑 GenAVB 配置文件：

```
# vi /etc/genavb/config
```

2. 将配置文件设置为 PROFILE 1：

```
PROFILE=1
```

3. 退出并保存。

在每次启动时完成：

1. 在 VLAN 2 上使能数据包接收（在最近的内核上默认使能 VLAN 硬件过滤）：

```
# ip link add link eth1 type vlan id 2
```

2. 使能线程化 NAPI 并微调 NAPI kthread 的 CPU 内核关联：

```
# echo 1 > /sys/class/net/eth1/threaded
# taskset -p 4 `pgrep irq/61-eth1`
# chrt -pf 66 `pgrep irq/61-eth1`
# taskset -p 4 `pgrep napi/eth1-rx-0`
# chrt -pf 65 `pgrep napi/eth1-rx-0`
# taskset -p 4 `pgrep napi/eth1-tx-1`
# chrt -pf 1 `pgrep napi/eth1-tx-1`
# taskset -p 2 `pgrep napi/eth1-tx-2`
# chrt -pf 1 `pgrep napi/eth1-tx-2`
# taskset -p 8 `pgrep napi/eth1-rx-1`
# taskset -p 8 `pgrep napi/eth1-rx-2`
# taskset -p 8 `pgrep napi/eth1-rx-3`
# taskset -p 8 `pgrep napi/eth1-rx-4`
# taskset -p 8 `pgrep napi/eth1-tx-0`
```

3. 启动 tsn-app 应用程序：

```
# avb.sh start
```

4. 使用 tc 设置计划流量（可以在启动 tsn-app 之前或之后完成）：

```
# tc qdisc add dev eth1 parent root handle 100 taprio \
num_tc 3 \
map 0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 \
queues 1@0 1@1 1@2 \
base-time 000500000 \
sched-entry S 0x2 4000 \
sched-entry S 0x5 1996000 \
flags 0x2
```

4.2.4.5.4.2 准备 IO 设备

只需完成一次：

1. 在 Linux 提示符下使用以下命令编辑 GenAVB 配置文件：

```
# vi /etc/genavb/config
```

2. 将配置文件设置为 PROFILE 2：

```
PROFILE=2
```

3. 退出并保存。

在每次启动时完成：

1. 在 VLAN 2 上使能数据包接收（在最近的内核上默认使能 VLAN 硬件过滤）：

```
# ip link add link eth1 type vlan id 2
```

2. 使能线程化 NAPI 并微调 NAPI kthread 的 CPU 内核关联：

```
# echo 1 > /sys/class/net/eth1/threaded
# taskset -p 4 `pgrep irq/61-eth1`
# chrt -pf 66 `pgrep irq/61-eth1`
# taskset -p 4 `pgrep napi/eth1-rx-0`
# chrt -pf 65 `pgrep napi/eth1-rx-0`
# taskset -p 4 `pgrep napi/eth1-tx-1`
# chrt -pf 1 `pgrep napi/eth1-tx-1`
# taskset -p 2 `pgrep napi/eth1-tx-2`
# chrt -pf 1 `pgrep napi/eth1-tx-2`
# taskset -p 8 `pgrep napi/eth1-rx-1`
# taskset -p 8 `pgrep napi/eth1-rx-2`
# taskset -p 8 `pgrep napi/eth1-rx-3`
# taskset -p 8 `pgrep napi/eth1-rx-4`
# taskset -p 8 `pgrep napi/eth1-tx-0`
```

3. 启动 tsn-app 应用程序：

```
# avb.sh start
```

4. 使用 tc 设置计划流量（可以在启动 tsn-app 之前或之后完成）：

```
# tc qdisc add dev eth1 parent root handle 100 taprio \
num_tc 3 \
map 0 0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 \
queues 1@0 1@1 1@2 \
base-time 001500000 \
sched-entry S 0x2 4000 \
sched-entry S 0x5 1996000 \
flags 0x2
```

4.2.4.5.4.3 准备网桥

请参阅 TSN 配置和 Tc-taprio 用法一节，配置 LS1028ARDB 板上计划流量。

应遵循网桥一节中所述的时间表。

每次启动时应执行以下步骤：

1. 设置网桥转发：

```
# ip link set dev eno2 up
# ip link add name br0 type bridge
# ip link set br0 up
# ip link set master br0 swp0 up
# ip link set master br0 swp1 up
# ip link set master br0 swp2 up
# ip link set master br0 swp3 up
```

2. 启动 gPTP 协议栈：

```
# fgptp.sh start
```

3. 设置计划流量（见上文）

```
# tc qdisc del dev swp0 root
# tc qdisc del dev swp1 root
# tc qdisc del dev swp2 root

# tc qdisc replace dev swp0 root taprio \
        num_tc 8 \
        map 0 1 2 3 4 5 6 7 \
        queues 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 \
        base-time 1500000 \
        sched-entry S 0x20 20000 \
        sched-entry S 0xdf 1980000 \
        flags 0x2

# tc qdisc replace dev swp1 root taprio \
        num_tc 8 \
        map 0 1 2 3 4 5 6 7 \
        queues 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 \
        base-time 500000 \
        sched-entry S 0x20 20000 \
        sched-entry S 0xdf 1980000 \
        flags 0x2

# tc qdisc replace dev swp2 root taprio \
        num_tc 8 \
        map 0 1 2 3 4 5 6 7 \
        queues 1@0 1@1 1@2 1@3 1@4 1@5 1@6 1@7 \
        base-time 500000 \
        sched-entry S 0x20 20000 \
        sched-entry S 0xdf 1980000 \
        flags 0x2
```

4.2.4.5.4.4 准备 OPC UA 客户端

为了可视化 TSN 端点应用程序 OPC UA 服务器公开的数据，需要在连接到网桥的 PC 上使用 OPC UA 客户端。

1. 在 PC 上安装 OPC UA 客户端：
   a. FreeOpcUa：带有 Qt GUI 界面的客户端。

      可在此处找到：http://freeopcua.github.io/

   b. opcua-commander：基于 CLI 替代的 nodejs node-opcua 协议栈。可在此处找到：https://github.com/node-opcua/opcua-commander
2. 将 PC 连接到网桥。如果尚未完成，请在运行 TSN 示例应用程序的端点以及 PC 上设置 IP 地址。然后，确保您可以使用 PC 成功 ping 端点。

4.2.4.5.5 评估说明

1. 复位所有端点。
2. 使用上述步骤，在网桥上启动 gPTP 协议栈，并在端点上启动 tsn-app 应用程序。

3. 使用上述配置在端点（和网桥）上使能计划流量。
4. 几秒后，TSN 端点应通过 gPTP 同步并以每秒 500 个数据包(pps)的速率交换数据包。为了观察这种行为，应检查日志。

4.2.4.5.5.1 gPTP 操作

如果 gPTP 协议在端点或网桥上正确运行，则 gptp 日志文件中应出现以下行（有关更多详细信息，请参见 gPTP 端点）：

```
gptp_stats_dump: Port(0) domain(0,0) : Role: Slave Link : Up AS_Capable: Yes
```

如果设备是 grand master，则角色字段应为“Master”，否则应为“Slave”。线路会定期出现，但角色不应随时间而改变，除非发生重大事件（例如线缆断开）。

4.2.4.5.5.2 基线 tsn-app 操作

如果 TSN 端点示例应用程序正确运行并接收到有效数据包，则可以在 tsn_app 日志文件中验证以下几点（有关更多详细信息，请参见 TSN 端点示例应用程序）。

以下行应定期出现：

```
socket_stats_print : link up
```

“有效帧”计数器应在以下日志的两次出现之间以 2500（5 秒 500 pps）递增：

```
socket_stats_print : valid frames : XXXXX
```

各种错误计数器不应递增（具有非零值很正常，因为在启动期 gPTP 和/或远程 tsn-app 端点可能无法运行且稳定）：

- “sched early”、“sched late”、“sched missed”、“sched timeout”、“sched discont”、“clock err”
- “err id”、“err ts”、“err underflow”
- “frames err”（对于 RX 和 TX 方向）

[其他]

上述检查适用于所有 tsn-app 端点，无论它们是控制器还是其中一个 IO 设备。

4.2.4.5.5.3 无并发流量的计划流量评估

下面的观察假设一个原本空闲的系统仅通过 tsn-app 应用程序接收和发送流量，所有设备（tsn-app 端点、网桥）上都有 802.1Qbv 时间表。

调度错误统计数据（“sched err”）应遵循以下值：

- 最小值约 8µs
- 平均值约 11µs
- 最大值约 25µs

```
stats(0xaaab06ed74b0) sched err min 8817 mean 11120 max 22077 rms^2 125202075 stddev^2 1544829 absmin
7417 absmax 1882057
```

处理时间统计数据（“processing time”）应遵循以下值：

- 最小值约 23µs
- 平均值约 29µs

- 最大值约 70µs

```
stats(0xaaab06ed7910) processing time min 23400 mean 29185 max 59100 rms^2 857707540 stddev^2 5943315
absmin 19560 absmax 4143240
```

流量延迟统计数据应遵循以下值：

- 最小值约 503µs
- 平均值约 503µs
- 最大值约 503µs
- stddev^2 小于 3000

```
stats(0x419a28) traffic latency min 503417 mean 503503 max 503637 rms^2 253515981945 stddev^2 2004
absmin 503397 absmax 504337
```

4.2.4.5.5.4 有 TX“非时间关键型”流量的计划流量评估

1. 将 PC 连接到 LS1028ARDB 开关(swp3)的第 4 个端口。
2. 在 PC 上以服务器模式运行 iperf3（用连接 LS1028 的 PC 接口代替 ethX）：

```
# ifconfig ethX 192.168.1.10 up
# iperf3 -s &
# iperf3 -s -p 5202 &
# iperf3 -s -p 5203 &
# iperf3 -s -p 5204 &
```

3. 在控制器上以客户端模式运行 iperf3：

```
# ifconfig eth1 192.168.1.80
# taskset 1 iperf3 -c 192.168.1.10 -u -b 900m -i 2 -t 100 &
# taskset 2 iperf3 -p 5202 -c 192.168.1.10 -u -b 0 -i 2 -t 100 &
# taskset 4 iperf3 -p 5203 -c 192.168.1.10 -u -b 0 -i 2 -t 100 &
# taskset 8 iperf3 -p 5204 -c 192.168.1.10 -u -b 0 -i 2 -t 100 &
```

4. 观察 tsn-app 日志文件中的统计数据（可能必须通过 SSH 打开第二个终端）。这些值应与下表匹配（以 µs 为单位）：

	最小值	平均值	最大值	stddev^2
调度错误（控制器）	21	29	41	
处理时间（控制器）	47	80	260	
流量延迟（控制器和 IO 设备）	503	503	503	<3000

4.2.4.5.5.5 有 RX“非时间关键型”流量的计划流量评估

[其他]

默认情况下，tsn-app 流量与未标记“非时间关键型”的流量在同一队列中处理。为了更轻松地使用“非时间关键型”流量验证 tsn-app，我们应当将 PCP=0 的 VLAN 标记添加到“非时间关键型”数据包，以便在接收时将它们分派到不同的队列中。

1. 将 PC 连接到 LS1028ARDB 开关(swp3)的第 4 个端口。
2. 在控制器上以服务器模式运行 iperf3：

```
# ip link add link eth1 name eth1.5 type vlan id 5
# ifconfig eth1.5 192.168.5.80 up
# iperf3 -s
```

3. 在 PC 上以客户端模式运行 iperf3（用连接 LS1028 的 PC 接口代替 ethX）：

```
# ip link add link ethX name ethX.5 type vlan id 5
# ifconfig ethX.5 192.168.5.10 up
# iperf3 -c 192.168.5.80 -u -b 0 -l 64 -i2 -t 100
```

4. 观察 tsn-app 日志文件中的统计数据（可能必须通过 SSH 打开第二个终端）。这些值应与下表匹配（以 µs 为单位）：

	最小值	平均值	最大值	stddev^2
调度错误（控制器）	9	13	26	
处理时间（控制器）	25	33	70	
流量延迟（控制器和 IO 设备）	503	503	503	<130000

4.2.4.5.5.6 OPC UA 服务器评估

OPC UA 服务器地址采用以下格式：opc.tcp://<endpoint IP address>:4840/

连接后，即可浏览和访问服务器对象。TSN 示例应用程序日志中所述的相同统计数据可用作 OPC UA 对象。OPC UA 服务器流量归类为“非时间关键型”，不会影响时间敏感型流量。

使用 FreeOPCUA GUI 客户端查看下面的截图：

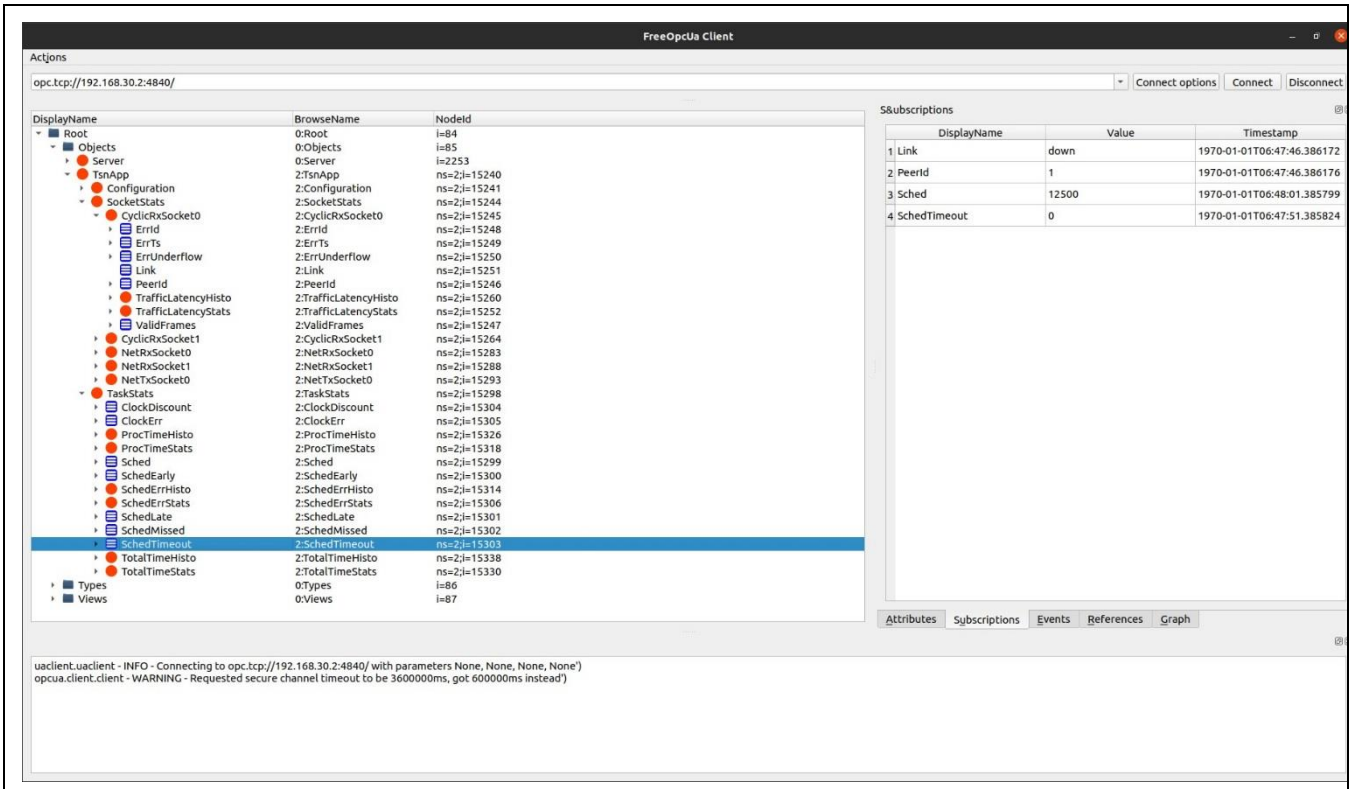


图 33.FreeOPCUA GUI 客户端

4.2.5 配置文件

4.2.5.1 系统

位于`/etc/genavb/system.cfg` 中的系统配置文件列出了系统网络接口名称和 PTP 硬件时钟设备名称。如果缺少配置文件或选项键，则使用默认值。安装文件中的值也可能需要更新以匹配系统配置。

本节列出了网络接口名称。

当前端点包支持单个端点和网桥包支持单个网桥（最多 5 个端口）。

表 49.逻辑端口

名称	键	默认值	说明
端点接口	`endpoint`	eth0	端点网络接口名称。仅对端点包有效，否则应设置为“关闭”
网桥 0 接口	`bridge_0`	SJA1105P_p0、SJA1105P_p1、SJA1105P_p2、SJA1105P_p3、SJA1105P_p4*	网桥 0 网络接口名称（顿号分隔）。仅对网桥包有效，否则应设置为“关闭”

本节列出了时钟设备名称。

时钟名称是 PHC 设备名称或通用软件时钟(`sw_clock`)。本地时钟指向 PHC 设备，目标时钟指向以下两者之一：

- 与本地时钟相同的 PHC 设备（gPTP 时间反映在本地时钟中）
- 通用软件时钟（在这种情况下，gPTP 时间不会反映在本地时钟中）。

表 50.时钟

名称	键	默认值	说明
端点 gPTP 功能域 0 目标时钟	`endpoint_gptp_0`	`/dev/ptp0`	gPTP 功能域 0 目标时钟的端点时钟。仅对端点包有效。
端点 gPTP 功能域 1 目标时钟	`endpoint_gptp_1`	`sw_clock`	gPTP 功能域 1 目标时钟的端点时钟。仅对端点包有效。
端点本地时钟	`endpoint_local`	`/dev/ptp0`	本地时钟的端点时钟。仅对端点包有效。
网桥 gPTP 功能域 0 目标时钟	`bridge_gptp_0`	`sw_clock`	gPTP 功能域 0 目标时钟的网桥时钟。仅对网桥包有效。
网桥 gPTP 功能域 1 目标时钟	`bridge_gptp_1`	`sw_clock`	gPTP 功能域 1 目标时钟的网桥时钟。仅对网桥包有效。
网桥本地时钟	`bridge_local`	`/dev/ptp1`	本地时钟的网桥时钟。仅对网桥包有效。

4.2.5.2 gPTP

gPTP 通用参数以及默认功能域（功能域 0）参数在以下配置文件中定义，具体取决于使用的包：

- 端点包：`/etc/genavb/fgptp.cfg`
- 网桥包：`/etc/genavb/fgptp-br.cfg`

要使能其他功能域，必须创建新的配置文件，并将关联的功能域实例附加到配置文件名，例如：

- 端点包，功能域 1：`/etc/genavb/fgptp.cfg-1`
- 网桥包，功能域 1：`/etc/genavb/fgptp-br.cfg-1`

注意

默认情况下，GenAVB/TSN gPTP 协议栈附有通用参数配置文件（`fgptp.cfg` 或 `fgptp-br.cfg`）和功能域 1 的参考配置（`fgptp.cfg-1` 或 `fgptp-br.cfg-1`）

4.2.5.2.1 综述

配置文件

gPTP 协议栈可以在名为“`standard`”或“`automotive`”配置文件的两种不同模式下运行。

选择“standard”配置文件时，gPTP 协议栈按照 IEEE 802.1AS 中所述的规范运行。选择“automotive”配置文件时，gPTP 协议栈按照 AVnuAutoCDS 功能规范_1.4 中所述的规范运行，该规范是针对汽车应用优化的 IEEE 802.1AS 标准的一个子集。IEEE 802.1AS-2020 功能在“automotive”配置文件中不可用（例如多功能域）。

汽车环境的独特之处在于它是一个封闭系统。每个网络设备在启动之前都是已知的，并且设备不会进入或离开网络，除非出现故障。由于汽车网络的封闭性，可以简化和提高 gPTP 的启动性能。具体来说，像选择 grand master 和计算线路延迟等功能均是可以针对封闭系统进行优化的任务。

反向同步功能控制

反向同步功能（Avnu 规范）应仅用于测试/评估目的。通常，为了测量时钟同步的准确性，传统的方法是使用每秒 1 个脉冲(1PPS)的物理输出。虽然这是一种好方法，但在某些情况下使用 1PPS 输出不可行。反向同步功能更灵活且完全依赖于软件实施，可使用标准 gPTP 同步/跟随消息将时序信息从 Slave 传回 GM，从而达到相同的目标。

邻近传播延迟阈值

参数 neighborPropDelayThresh 定义了传播时间阈值，超过该阈值的端口被认为不能参与 IEEE 802.1AS 协议（参见 IEEE 802.1AS-2020——11.2.2 确定 asCapable 和 asCapableAcrossDomains）。如果计算出的 neighborPropDelay 超过了 neighborPropDelayThresh，则端口的 asCapable 设置为 FALSE。此设置不适用于始终认为链路具有能力或运行 IEEE 802.1AS 的汽车配置文件。

IEEE 802.1AS-2011 兼容性

参数 force_2011 定义 gPTP 协议栈是否按照 IEEE 802.1AS-2011 标准运行，即禁用 IEEE 802.1AS-2020 特定功能，例如多功能域支持。在某些情况下，使用此选项可以提高与不支持 IEEE 802.1AS-2020 标准的 gPTP 设备的兼容性。

通用配置参数 [1]

表 51.通用参数

名称	键	默认值	范围	说明
配置文件	配置文件	“标准”	“标准”或“汽车”	设置 fgptp 主配置文件。“标准”——IEEE 802.1AS 规范，“汽车”——AVnu 汽车配置文件
Grandmaster ID	gm_id	“0x0001f2fffe0025fe”	64 位 EUI 格式	按主机顺序设置静态 grandmaster ID（汽车配置文件使用，标准配置文件情况下忽略）
功能域	domain_number	0：为默认功能域 -1：为 0 以外的功能域	-1 至 127	禁用(-1)或将 gPTP 功能域号分配给功能域实例。
802.1AS-2011 模式	force_2011	否	“否”或“是”	设置为“是”则强制执行 802.1AS-2011 标准。“否”则提供 802.1AS-2020 全面支持。

表格接下页……

表 51.通用参数（续）

名称	键	默认值	范围	说明
日志输出级别	log_level	info	crit、err、init、info 或 dbg	将此配置设置为 dbg，以使能调试模式
反向同步功能控制	reverse_sync	0	0 或 1	设置为 1，以在 slave 端使能反向同步发送
反向同步功能间隔	reverse_sync_interval	112	32 至 10000	以毫秒为单位的反向同步发送间隔
邻近传播延迟阈值	neighborPropDelayThresh	800	32 至 10000000	以 ns 表示的邻近传播延迟阈值
统计输出间隔	statsInterval	10	0 至 255	以秒表示的统计数据输出间隔。使用 0 禁用统计数据

1.对于 0 以外的功能域实例，本节中仅 domain_number 可配置。

4.2.5.2.2 Grandmaster 参数

本节定义时间感知系统的原生 Grand Master 能力（请参见 IEEE 802.1AS-2020——8.6.2 PTP 实例属性）。Grand Master 能力参数在 gPTP 功能域 0 的主配置文件（例如 fgptp.cfg）和其他功能域的附加每个功能域配置文件（例如 fgptp.cfg-1）中定义。

gmCapable 定义时间感知系统是否能够成为 grandmaster。默认情况下，gmCapable 设置为 1，因为在标准配置文件操作中，Grand Master 由 BMCA 动态选择。在汽车配置文件的情况下，必须在每个 AED 节点上设置 gmCapable，以匹配所需的网络拓扑（即在给定的 gPTP 功能域内，只有一个节点必须将其 gmCapable 属性设置为 1）。

priority1、priority2、clockClass、clockAccuracy 和 offsetScaledLogVariance 是最佳 Master 时钟算法使用的参数，用于确定 gPTP 功能域中哪个 Grand Master 功能节点具有最高优先级/质量。请注意，这些参数的最低值与最高优先级/质量相匹配。

Grandmaster 能力参数 [1]

表 52.Grandmaster 参数

名称	键	默认值	范围	说明
GrandMaster 功能设置	gmCapable	1	0 或 1	如果设备具有 grandmaster 能力，则设置为 1。如果端口是 SLAVE，则在汽车配置文件中忽略。
Grandmaster priority1 值	priority1	AED-E 为 248，AED-B 为 246	0 至 255	设置此时钟的 priority1 值
Grandmaster priority2 值	priority2	248	0 至 255	设置此时钟的 priority2 值

表格接下页……

表 52.Grandmaster 参数（续）

名称	键	默认值	范围	说明
Grandmaster 时钟类别值	clockClass	248	0 至 255	设置此时钟的类别值
Grandmaster 时钟精度值	clockAccuracy	0xfe	0x0 至 0xff	设置此时钟的精度值
Grandmaster 方差值	offsetScaledLogVariance	17258	0x0 至 0xffff	设置此时钟的偏移量缩放对数方差值

1.本节中的参数可针对所有受支持的功能域进行配置。

4.2.5.2.3 汽车参数

仅当 gPTP 协议栈在汽车配置文件配置中运行时，才使用静态 pdelay 功能。

在初始化时，gPTP 协议栈的配置文件被解析，并且基于 neighborPropDelay_mode，指定的 initial_neighborPropDelay 应用于所有端口并用于同步，直到收到来自对等设备的 pdelay 响应。仅当 nvram_file 指定的 nvram 数据库中没有先前存储的 pdelay 时才执行此操作。一旦收到来自对等设备的 pdelay 响应，就会计算“真实”pdelay 值，并将其用于当前同步。然后可以通过回调将指令发送到依赖于操作系统的层。根据新指令，主机可能会更新其 nvram 数据库，并且存储的值将在下次重启时用于相应端口，而不是 initial_neighborPropDelay。pdelay 更改指令发送到主机情况下的粒度由 neighborPropDelay_sensitivity 参数定义。

在 gPTP 配置文件中，neighborPropDelay_mode 参数默认设置为“static”，这意味着如上所述使用预定义的传播延迟，同时仍将 pdelay 请求发送到网络。

“静默”模式的行为方式与“静态”模式相同，只是 pdelay 请求根本不会发送到网络。

可选择将 neighborPropDelay_mode 参数设置为“standard”，以强制协议栈按照 802.1AS 规范中的规定进行传播延迟测量，即使选择了汽车配置文件也是如此。

（参见 AutoCDS 功能规范-1_4——6.2.2 持久 gPTP 值）

表 53.汽车参数

名称	键	默认值	值和范围	说明
Pdelay 模式	neighborPropDelay_mode	静态	“静态”、“静默”或“标准”	定义使用的 pdelay 机制
静态 pdelay 值	initial_neighborPropDelay	250	0 至 10000	预定义的 pdelay 值应用于所有端口。以 ns 表示。
静态 pdelay 灵敏度	neighborPropDelay_sensitivity	10	0 至 1000	触发变化指示所需的两次 pdelay 测量之间的纳秒量。以 ns 表示。
Nvram 文件名	nvram_file	/etc/genavb/fgptp.nvram		路径和 nvram 文件名。

4.2.5.2.4 时序

Pdelay 请求和 Sync 消息发送间隔对系统同步性能有直接影响。为了在优化整体系统负载的同时减少同步时间，定义了两个级别的间隔。第一个级别称为“初始”，定义了在 pdelay 值稳定并实现同步之前使用的消息间隔。第二个级别称为“操作”，定义了系统同步后使用的消息间隔。

initialLogPdelayReqInterval 和 operLogPdelayReqInterval 定义了发送连续 Pdelay_Req 消息之间的间隔。initialLogSyncInterval 和 operLogSyncInterval 定义发送连续同步消息之间的间隔。initialLogAnnounceInterval 定义发送连续公告消息之间的间隔

（参见 AutoCDS 功能规范-1_4——6.2.1 静态 gPTP 值、IEC-60802 第 5 节、802.1AS-2020 第 10.7 和 11.5 节）

表 54.时序参数

名称	键	默认值	值和范围	说明
初始 pdelay 请求间隔值	initialLogPdelayReqInterval	0	0 至 3	设置发送连续 Pdelay_Req 消息之间的 pdelay 请求初始间隔。以 log2 单位表示（默认 0 -> 1s）。
初始同步间隔值	initialLogSyncInterval	-3	-5 至 0	设置发送连续同步消息之间的同步发送初始间隔。以 log2 单位表示（默认-3-> 125ms）。
初始公告间隔值	initialLogAnnounceInterval	0	0 至 3	设置发送连续公告消息之间的初始公告发送间隔。以 log2 单位表示（默认 0 -> 1s）。
操作 pdelay 请求间隔值	operLogPdelayReqInterval	0	0 至 3	设置在正常操作状态下使用的 pdelay 请求发送间隔。以 log2 单位表示（默认 0 -> 1s）。
操作同步间隔值	operLogSyncInterval	-3	-5 至 0	设置在正常操作状态下使用的同步发送间隔。以 log2 单位表示（默认-3 -> 125ms）。

4.2.5.2.5 端口 n

本节介绍每个端口的设置，其中 n 表示从 n=1 开始的端口索引。

表 55.端口相关参数

名称	键	默认值	值和范围	说明
端口角色	portRole	禁用	“slave”、“master”、“禁用”	静态端口角色（参见 802.1AS-2011，第 14.6.3 节，表 10-1），仅适用于“汽车”配置文件。
Ptp 端口已使能	ptpPortEnabled	1	0 或 1	如果应当使用端口的时间同步和最佳 master 选择功能，则设置为 1（参见 802.1AS-2011，第 14.6.4 和 10.2.4.12 节）。

4.2.5.3 SRP

SRP 参数在以下配置文件中定义，具体取决于使用的包：

- 端点：`/etc/genavb/srp.cfg`
- 网桥：`/etc/genavb/srp-br.cfg`

如果缺少配置文件或选项键，则使用默认值。安装文件中的值也可能需要更新以匹配系统配置。

本节列出了通用 SRP 协议栈组件参数。

表 56.通用 SRP

名称	键	默认值	范围	说明
日志输出级别	log_level	info	crit、err、init、info 或 dbg	SRP 协议栈组件的日志级别。

本节列出了 MSRP 参数。

表 57.MSRP

名称	键	默认值	范围	说明
使能	使能	1	0-已禁用，1-已使能	在运行时使能/禁用 MSRP。

4.2.6 日志文件

在运行时可以使用多个日志文件来监测不同的协议栈组件。

4.2.6.1 gPTP 端点

日志存储在`/var/log/fgptp` 中。

- Linux 命令：

```
# tail -f /var/log/fgptp
```

- 如果协议栈配置为汽车模式，则日志包含：

```
Running fgptp in automotive profile on interface eth0
```

- 每次链路状态发生变化（链路是否支持 802.1AS）或 Grand Master (GM)更改时，都会报告端口角色、端口 AS 能力和链路状态。此信息还会与每个使能的 gPTP 功能域的当前同步和 pdelay 统计数据一起定期显示：

```
Port(0) domain(0,0): role changed from DISABLED to SLAVE
…
Port(0) domain(0,0): Slave - Link: Up - AS_Capable: Yes
```

- 所选 Grand Master (GM)能力会在新的 GM 选择时报告。*Root 身份*表示当前所选 GM 的时钟 ID。*Priority1*、*Priority2*、*类别*和*精度*描述了所选 GM 的时钟质量。最后，对等 master 端口的*源端口身份*（例如，本地 slave 端口连接到的网桥端口）。为每个使能的 gPTP 功能域显示此信息：

```
domain(0,0) Grand master: root identity 00049ffffe039e35
domain(0,0) Grand master: priority1 245 priority2
domain(0,0) Grand master: class 248 accuracy 248
domain(0,0) Grand master: variance 17258
domain(0,0) Grand master: source port identity 0001f2fffe0025fe, port number2
```

- 在 GM 选择（已同步）或没有检测到 GM 时（未同步）报告*同步状态*。以 ms 表示的*同步时间*表示本地时钟从接收到的第一个 SYNC 消息开始到达到同步阈值所用的时间。为每个使能的功能域显示此信息。

```
Port(0) domain(0) SYNCHRONIZED - synchronization time (ms):250
```

- *Pdelay*（传播延迟）和本地时钟调整每 5 秒打印一次。*PDelay* 以 ns 为单位表示，表示来自端点及其对等 master 的单向延迟。*校正*以十亿分率表示，表示对本地时钟执行的频率调整。*偏移量*以 ns 表示，表示本地调整的时钟与参考 gPTP GrandMaster 时钟之间的差值。（针对校正和偏移量统计数据计算最小值/最大值/平均值和方差）。*PDelay* 仅针对功能域 0 显示。为每个使能的功能域显示*校正*和*偏移量*。

```
Port 0 domain(0,0): Propagation delay (ns): 37.60  min 34 avg 36 max 45 variance17
Port 0 domain(0,0): Correction applied to local clock (ppb): min -5603 avg 5572 max 5538
variance 148
Port 0 domain(0,0): Offset between GM and local clock (ns) min -12 avg 4 max 22 variance 111
...
Port 0 domain(1,20): Correction applied to local clock (ppb): min 32074 avg 32314 max 32574
variance 17695
Port 0 domain(1,20): Offset between GM and local clock (ns) min -61 avg 3 max 70 variance 1149
```

- 以下每个端口每个功能域的统计数据（32 位计数器）每 15 秒在 slave 和 master 实体上打印一次：

表 58.slave 和 master 实体上显示的端口统计数据

接收计数器	
PortStatRxPkts	接收到的 gPTP 数据包数（以太类型 0x88F7）
PortStatRxSyncCount	接收到的同步数据包数
PortStatRxSyncReceiptTimeouts	同步数据包接收超时数
PortStatRxFollowUpCount	收到的跟随数据包数

表格接下页……

表 58.slave 和 master 实体上显示的端口统计数据（续）

PortStatRxAnnounce	收到的公告数据包数
PortStatAnnounceReceiptTimeouts	公告数据包超时数
PortStatAnnounceReceiptDropped	实体丢弃的公告数据包数
PortStatRxSignaling	收到的信号传递数据包数
PortStatRxPdelayRequest	收到的 PDELAY 请求数据包数
PortStatRxPdelayResponse	收到的 PDELAY 响应数据包数
PortStatPdelayAllowedLostResponsesExceeded	允许丢失对 PDELAY 请求响应的额外数
PortStatRxPdelayResponseFollowUp	收到的 PDELAY 跟随数据包数
PortStatRxErrEtype	以太类型错误数（不是 0x88F7）
PortStatRxErrPortId	编号或端口 ID 错误
发送计数器	
PortStatTxPkts	发送的 gPTP 数据包数
PortStatTxSyncCount	发送的 SYNC 数据包数
PortStatTxFollowUpCount	发送的跟随数据包数
PortStatTxAnnounce	发送的公告数据包数
PortStatTxSignaling	发送的信号传递数据包数
PortStatTxPdelayReques	发送的 PDELAY 请求数据包数
PortStatTxPdelayResponse	发送的 PDELAY 响应数据包数
PortStatTxPdelayResponseFollowUp	发送的 PDELAY 跟随数据包数
PortStatTxErr	发送错误数
PortStatTxErrAlloc	发送数据包分配错误数
其他计数器	
PortStatAdjustOnSync	收到同步时执行的调整次数
PortStatMdPdelayReqSmReset	PDELAY 请求状态机的复位次数
PortStatMdSyncRcvSmReset	同步接收状态机的复位次数
PortStatHwTsRequest	出口时间戳请求数

表格接下页……

表 58.slave 和 master 实体上显示的端口统计数据（续）

PortStatHwTsHandler	出口时间戳通知数
PortStatNumSynchronizationLoss	slave 端点上的数量或同步丢失（例如 GM 更改、GM 参考时钟不连续……）
PortStatNumNotAsCapable	从 AS_Capable=TRUE 到 AS_Capable=FALSE 的转换次数

4.2.6.2 gPTP 网桥

日志存储在/var/log/fgptp-br 中。

- Linux 命令：

```
# tail -f /var/log/fgptp-br
```

- 网桥协议栈统计数据类似于端点协议栈统计数据，不同之处在于前者针对交换机的每个外部端口（端口 0 到 3）以及在混合设置情况下连接到端点协议栈的内部端口（端口 4）进行报告。
- *Pdelay*（传播延迟）仅针对功能域 0 打印。打印出每个端口和每个 gPTP 功能域的*链接状态*、*AS 能力*和*端口角色*。

```
Port 0 domain(0,0): Role: Disabled Link: Up AS_Capable: No
Port 1 domain(0,0): Role: Disabled Link: Up AS_Capable: No
Port 2 domain(0,0): Role: Disabled Link: Up AS_Capable:Yes
Port 2 domain(0,0): Propagation delay (ns): 433.98 min 425 avg 438 max 457 variance 87
Port 3 domain(0,0): Role: Disabled Link: Up AS_Capable: No
Port 4 domain(0,0): Role Master Link: Up AS_Capable: Yes
Port 4 domain(0,0): Propagation delay (ns): 433.98 min 425 avg 438 max 457 variance 87
...
Port 0 domain(1,20): Role: Disabled Link: Up AS_Capable: No
Port 1 domain(1,20): Role: Disabled Link: Up AS_Capable: No
Port 2 domain(1,20): Role: Disabled Link: Up AS_Capable: Yes
Port 3 domain(1,20): Role: Disabled Link: Up AS_Capable: No
Port 4 domain(1,20): Role Master Link: Up AS_Capable: Yes
```

4.2.6.3 SRP 桥

日志存储在/var/log/avb-br 中。

- Linux 命令：

```
# tail -f /var/log/avb-br | grep srp
```

- 每个端口报告 SRP 协议信息

```
INFOsrp      msrp_vector_add_event           : port(0) domain(5, 2, 2) MSRP_ATTR_TYPE_DOMAIN
MRP_ATTR_EVT_JOINMT
INFOsrp      msrp_vector_add_event            : port(0) domain(6, 3,2)MSRP_ATTR_TYPE_DOMAIN
MRP_ATTR_EVT_JOINMT
INFOsrp      msrp_vector_add_event            : port(1) domain(5, 2,2)MSRP_ATTR_TYPE_DOMAIN
MRP_ATTR_EVT_JOINMT
INFOsrp      msrp_vector_add_event            : port(1) domain(6, 3,2)MSRP_ATTR_TYPE_DOMAIN
MRP_ATTR_EVT_JOINMT
INFOsrp      msrp_vector_add_event            : port(2) domain(5, 2, 2)MSRP_ATTR_TYPE_DOMAIN
```

```
MRP_ATTR_EVT_JOINMT
INFO srp       msrp_vector_add_event  : port(2) domain(6, 3, 2) MSRP_ATTR_TYPE_DOMAIN
MRP_ATTR_EVT_JOINMT
INFO srp       msrp_vector_add_event  : port(4) domain(5, 2, 2) MSRP_ATTR_TYPE_DOMAIN
MRP_ATTR_EVT_JOINMT
INFO srp       msrp_vector_add_event  : port(4) domain(6, 3, 2) MSRP_ATTR_TYPE_DOMAIN
MRP_ATTR_EVT_JOINMT
INFO srp       msrp_vector_handler    : port(3) domain(5, 2, 2) MRP_ATTR_EVT_MT
INFO srp       msrp_vector_handler    : port(3) domain(6, 3, 2) MRP_ATTR_EVT_MT
INFO srp       msrp_vector_handler    : port(3) domain(5, 2, 2) MRP_ATTR_EVT_JOINMT
INFO srp       msrp_vector_handler    : port(3) domain(6, 3, 2) MRP_ATTR_EVT_JOINMT
INFO srp       msrp_vector_add_event  : port(3) domain(5, 2, 2) MSRP_ATTR_TYPE_DOMAIN
MRP_ATTR_EVT_JOININ
INFO srp       msrp_vector_add_event  : port(3) domain(6, 3, 2) MSRP_ATTR_TYPE_DOMAIN
MRP_ATTR_EVT_JOININ
INFO srp       msrp_vector_add_event  : port(3) domain(5, 2, 2) MSRP_ATTR_TYPE_DOMAIN
MRP_ATTR_EVT_JOININ
INFO srp       msrp_vector_add_event  : port(3) domain(6, 3, 2) MSRP_ATTR_TYPE_DOMAIN
MRP_ATTR_EVT_JOININ
INFO srp       msrp_vector_add_event  : port(0) domain(5, 2, 2) MSRP_ATTR_TYPE_DOMAIN
MRP_ATTR_EVT_MT
INFO srp       msrp_vector_add_event  : port(0) domain(6, 3, 2) MSRP_ATTR_TYPE_DOMAIN
MRP_ATTR_EVT_MT
INFO srp       msrp_vector_add_event  : port(1) domain(5, 2, 2) MSRP_ATTR_TYPE_DOMAIN
MRP_ATTR_EVT_MT
INFO srp       msrp_vector_add_event  : port(1) domain(6, 3, 2) MSRP_ATTR_TYPE_DOMAIN
MRP_ATTR_EVT_MT
INFO srp       msrp_vector_add_event  : port(2) domain(5, 2, 2) MSRP_ATTR_TYPE_DOMAIN
MRP_ATTR_EVT_MT
INFO srp       msrp_vector_add_event  : port(2) domain(6, 3, 2) MSRP_ATTR_TYPE_DOMAIN
MRP_ATTR_EVT_MT
INFO srp       msrp_vector_add_event  : port(4) domain(5, 2, 2) MSRP_ATTR_TYPE_DOMAIN
MRP_ATTR_EVT_MT
INFO srp       msrp_vector_add_event  : port(4) domain(6, 3, 2) MSRP_ATTR_TYPE_DOMAIN
MRP_ATTR_EVT_MT
```

4.2.6.4 TSN 端点示例应用程序

日志存储在`/var/log/tsn_app` 中。

TSN 应用程序具有各种有助于验证的计数器和统计数据：

- 应用程序调度和处理时序统计数据
- 网络流量正确性和延迟统计数据

日志中的大部分信息是：

- 计数器：单个整数值计数特定事件（接收、发送、错误等）
- 统计数据：一系列测量的复合数据：最小值（上一周期的最小值）、平均值（上一周期的测量平均值）、最大值（上一周期的最大值）、rms^~（上一周期的测量均方根）上一期）、stddev^2（上一周期的标准差的平方）、absmin（自应用程序启动以来的绝对最小值）、absmax（自应用程序启动以来的绝对最大值）
- 直方图：第 1 行直方图插槽的数量和大小，第 2 行每个插槽的计数器数组。

4.2.6.4.1 主要 TSN 任务

主要 TSN 任务日志如下所述：

- 调度计数器（“sched”应该每秒递增 500）：

```
 INFO 1604531064     tsn_task_stats_print      tsn task(0x37d8d630)
 INFO 1604531064     tsn_task_stats_print      sched               : 1700000
 INFO 1604531064     tsn_task_stats_print      sched early         : 0
 INFO 1604531064     tsn_task_stats_print      sched late          : 0
 INFO 1604531064     tsn_task_stats_print      sched missed        : 0
 INFO 1604531064     tsn_task_stats_print      sched timeout       : 0
 INFO 1604531064     tsn_task_stats_print      clock discont       : 0
 INFO 1604531064     tsn_task_stats_print      clock err           : 0
```

- 调度错误统计（实际任务调度时间与编程时间之差，以 ns 为单位）：

```
stats(0x2ddd4560) sched err min 8062 mean 10662 max 18862 rms^2 116252492 stddev^2 2558383
absmin
2842 absmax 35082
```

- 调度错误直方图（XXX ns 漏桶）

```
n_slot 101 slot_size 10000
18081119 21722197 1676 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 00
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
```

- 处理时间统计数据（主要任务持续时间，以 ns 为单位）

```
stats(0x2ddd49c0) processing time min 21600 mean 27564 max 47460 rms^2 768483628 stddev^2
8664988
absmin 12540 absmax 152100
```

- 处理时间直方图（XXX ns 漏桶）

```
n_slot 101 slot_size 1000
0 0 0 0 0 0 0 0 0 0 0 0 9 13 7 259 302 187 14273 64041 91896 427011 1473790 21548894239252
4681777 5682883 5516429 4915621 3789351 2688466 1709802 1036649 587716 327653 170711 89620 47637
25422 15211 9573 6908 4852 3694 3101 2910 3028 3036 3331 3091 2842 2375 1748 1251 895 585 342
208 100 63 64 26 21 13 9 14 6 7 4 4 6 3 4 1 0 2 2 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 01
0
```

- 总时间统计数据（调度错误+处理时间，以 ns 为单位）

```
stats(0x2ddd4e20) total time min 30082 mean 38227 max 55862 rms^2 1473506486 stddev^2 12160755
absmin 18962 absmax 170082
```

- 总时间直方图（XXX ns 漏桶）

```
n_slot 101 slot_size 1000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 2 15 16 57 273 1674 7795 30165 80952 240707651779
1497160 2290907 3606869 4077757 4761502 4647779 4239893 3631222 2831749 2209868 1631293 1173544
819740 536234 338535 201816 115114 65529 36480 21413 12892 8249 5613 4178 3310 3034 2890 2921
2615 2461 2132 1790 1427 1082 844 585 353 266 170 98 73 48 20 25 6 14 14 1 6 2 3 8 2 2 1 3 2 11
2 1 2 0 1 1 0 0 0 10 0
```

4.2.6.4.2 网络套接字

以下是网络套接字日志的示例：

- 低级别网络套接字。此处仅计数与网络套接字（第 2 层）相关的帧：

```
INFO 1604531059 net_socket_stats_print net rx socket(0x37d8d660)0
INFO 1604531059 net_socket_stats_print frames : 1697802
```

```
INFO 1604531059 net_socket_stats_print frames err : 0
INFO 1604531059 net_socket_stats_print net tx socket(0x37d8d6e0)0
INFO 1604531059 net_socket_stats_print frames : 1697500
INFO 1604531059 net_socket_stats_print frames err : 0
```

4.2.6.4.3 应用程序套接字

- 在此级别检查应用程序标头。此外，还验证了来自远程对等设备的时间戳，这保证了只处理预期的和按顺序的数据。

```
INFO 1604531069 socket_stats_print cyclic rx socket(0x419560) net_sock(0x37d8d660) peer id: 1
INFO 1604531069 socket_stats_print valid frames :1702497
INFO 1604531069 socket_stats_print err id :
0 INFO 1604531069 socket_stats_print err ts : 305
INFO 1604531069 socket_stats_print err underflow : 2
INFO 1604531069 socket_stats_print link up
```

- 流量延迟统计数据（发送帧的对等设备的理论调度时间与帧接收时间（由 MAC 测量）之间的差值，以 ns 为单位）

```
stats(0x419a28) traffic latency min 503417 mean 503503 max 503637 rms^2 253515981945 stddev^2
2004 absmin 503397 absmax 504337
```

- 流量延迟直方图（XXX ns 漏桶）

```
n_slot 101 slot_size 1000
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 299998 0
```

4.3 IEEE 1588/802.1AS

4.3.1 简介

恩智浦 i.MX 和 Layerscape 平台通过 1588 定时器模块为符合 1588 的时间戳提供硬件支持。下面列出了利用硬件功能运行 IEEE 1588/802.1AS 协议所需的软件组件：

1.Linux PTP 硬件时钟(PHC)驱动程序

2.具有硬件时间戳支持的 Linux 以太网控制器驱动程序

3.IEEE 1588/802.1AS 的软件协议栈应用程序

> **注意**
>
> 在本文档中，提到的 IEEE 1588 是 IEEE 1588-2008，提到的 IEEE 802.1AS 是 IEEE 802.1AS-2011。

4.3.2 IEEE 1588 设备类型

IEEE 1588 中有五种基本 PTP 设备类型。

- **普通时钟**

在功能域中具有单个精确时间协议(PTP)端口并维护功能域中使用的时间范围的时钟。它可以用作时间源（如果用作 master 时钟），也可以与另一个时钟同步（如果用作 slave 时钟）。

- **边界时钟**

在功能域中具有多个精确时间协议(PTP)端口并维护功能域中使用的时间范围的时钟。它可以用作时间源（作为 master 时钟）或可以与另一个时钟同步（作为 slave 时钟）。

- **端到端透明时钟**

支持在 slave 时钟和 master 时钟之间使用端到端延迟测量机制的透明时钟。

- **点对点透明时钟**

提供精确时间协议(PTP)事件传输时间信息的透明时钟。它还提供了对连接到接收 PTP 事件消息的端口的链路传播延迟的校正。在点对点透明时钟存在的情况下，使用点对点延迟测量机制进行 slave 时钟和 master 时钟之间的延迟测量。

- 管理节点

配置和监测时钟的设备。

注意

透明时钟是一种测量 PTP 事件消息发送到设备所用时间的设备。它将此信息提供给接收 PTP 事件消息的时钟。

4.3.3 IEEE 802.1AS 时间感知系统

在 gPTP 中，只有两种时间感知系统：终端节点和网桥，而 IEEE 1588 有普通时钟、边界时钟、端到端透明时钟和 P2P 透明时钟。时间感知终端节点对应于 IEEE 1588 普通时钟，时间感知网桥是一种 IEEE 1588 边界时钟，其操作定义非常严格，因此，就如何执行同步而言，具有以太网端口的时间感知网桥可以在数学上等同于 P2P 透明时钟。

1.时间感知终端节点

能够充当网络上的同步时间源，或充当使用 IEEE 802.1AS 协议同步时间的目标，或两者皆可的终端节点。

2.时间感知网桥

能够使用 IEEE 802.1AS 协议将在一个端口上接收到的同步时间传送到其他端口的网桥。

4.3.4 软件协议栈

4.3.4.1 linuxptp 协议栈

开源 linuxptp 的功能

- 通过 Linux SO_TIMESTAMPING 套接字选项支持硬件和软件时间戳。
- 通过使用 clock_gettime 系列调用（包括 clock_adjtimex 系统调用）来支持 Linux PTP 硬件时钟(PHC)子系统。
- 实施边界时钟(BC)、普通时钟(OC)和透明时钟(TC)。
- 通过 UDP/IPv4、UDP/IPv6 和原始以太网（第 2 层）传输。
- 支持 IEEE 802.1AS-2011 作为终端节点。
- 模块化设计允许轻松添加新的传输和时钟伺服系统。
- 实施单播操作。
- 支持多种配置文件，包括：
  - 汽车配置文件。
  - 默认的 1588 配置文件。

- 企业配置文件。
- 电信配置文件 G.8265.1、G.8275.1 和 G.8275.2。
- 支持 NetSync 监测协议。

- 一步实施点对点。
- 支持结合接口、IPoIB 和 vlan 接口。

注意：列出的功能来自 linuxptp 网站。这并不意味着所有这些功能都会在发布板上有效。需要考虑硬件 1588 能力、驱动程序支持和 ptp4l 版本。有关已验证的内容，请参见本章的以下用户手册。

实时边缘添加的功能

- 支持 IEEE 802.1AS-2011 作为时间感知网桥。
- 支持 ts2phc 中的动态方向，以与 ptp4l 配合。

4.3.4.2 恩智浦 GenAVB/TSN gPTP 协议栈

以下是恩智浦 GenAVB/TSN gPTP 协议栈的功能：

- 为时间感知端点和网桥系统实施 gPTP IEEE 802.1AS-2020
- 实施 gPTP BMCA
- 支持 GrandMaster、Master 和 Slave 能力
- 支持多个 gPTP 功能域
- 支持 Avnu 联盟汽车配置文件
- 支持协议栈的配置文件
- 通过 Linux `SO_TIMESTAMPING` 套接字选项支持硬件时间戳
- 通过使用 `clock_gettime` 系列调用（包括 `clock_adjtimex` 系统调用）来支持 Linux PTP 硬件时钟(PHC)子系统。

4.3.5 IEEE 1588 快速入门

4.3.5.1 普通时钟验证

将两块板的两个网络接口背靠背连接。确保板上没有 MAC 地址冲突，IP 地址设置正确并 ping 测试网络。在每个板上运行 `linuxptp`。例如，在每个板上使用 `eth0`。

```
$ ptp4l -i eth0 -m
```

运行上面的命令时会开始时间同步，自动选择的 slave linuxptp 会同步到 master，并显示同步信息，如时间偏移量、路径延迟等。例如，

```
ptp4l[878.504]: master offset     -10 s2 freq -2508 path delay       1826
ptp4l[878.629]: master offset      -5 s2 freq -2502 path delay       1826
ptp4l[878.754]: master offset       0 s2 freq -2495 path delay       1826
ptp4l[878.879]: master offset       9 s2 freq -2482 path delay       1826
ptp4l[879.004]: master offset      -9 s2 freq -2507 path delay       1826
ptp4l[879.129]: master offset     -24 s2 freq -2530 path delay       1826
ptp4l[879.255]: master offset      -7 s2 freq -2508 path delay       1826
ptp4l[879.380]: master offset      -2 s2 freq -2502 path delay       1826
ptp4l[879.505]: master offset     -17 s2 freq -2524 path delay       1827
ptp4l[879.630]: master offset       6 s2 freq -2493 path delay       1827
```

```
ptp4l[879.755]: master offset         6 s2 freq -2492 path delay      1827
ptp4l[879.880]: master offset         0 s2 freq -2500 path delay     1827
```

ptp4l 的一些其他选项

```
Delay Mechanism
-E      E2E, delay request-response (default)
-P      P2P, peer delay mechanism

Network Transport
-2      IEEE 802.3
-4      UDP IPV4 (default)
-6      UDP IPV6
```

注意：必须保持在两个板上使用相同的延迟机制和网络传输协议。

配置 master 模式

默认情况下，master 时钟由 BMC（最佳 Master 时钟）算法选择。要将特定时钟指定为 master，可以设置比其他时钟低的“priority1”属性值。较低的值优先。例如，在当前案例中，使用以下选项指定一个时钟作为 master。（另一个时钟使用默认的 priority1 值 128。）

```
--priority1=127
```

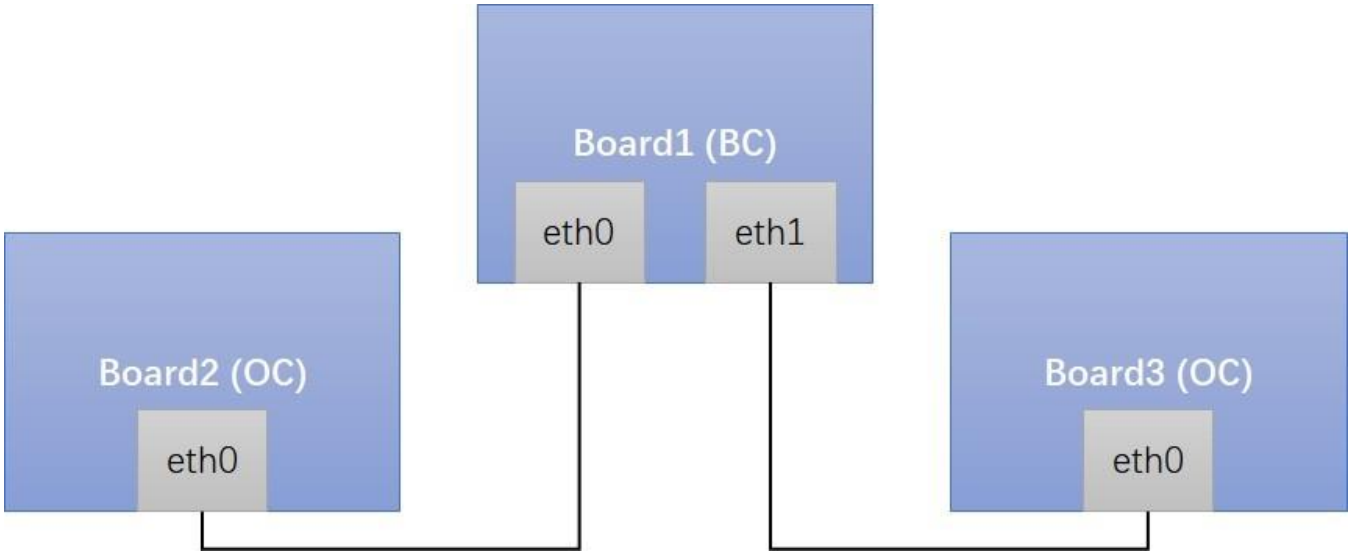
单步时间戳

目前仅 DPAA2 支持单步时间戳。要使用单步时间戳，请添加以下选项以运行 ptp4l。

```
--twoStepFlag=0
```

4.3.5.2 边界时钟验证

至少需要三个板。下面是一个三板网络连接的例子。确保板上没有 MAC 地址冲突，IP 地址设置正确并 ping 测试网络。



在板 1（边界时钟）上运行 `linuxptp`。

```
$ ptp4l -i eth0 -i eth1 -m
```

在板 2/板 3（普通时钟）上运行 linuxptp。

```
$ ptp4l -i eth0 -m
```

运行上面的命令时，会开始时间同步，自动选择的 slave linuxptp 会同步到特有的 master，并显示同步信息，如时间偏移量、路径延迟等。例如，

```
ptp4l[878.504]: master offset         -10 s2 freq      -2508 path delay   1826
ptp4l[878.629]: master offset          -5 s2 freq      -2502 path delay   1826
ptp4l[878.754]: master offset           0 s2 freq      -2495 path delay   1826
ptp4l[878.879]: master offset           9 s2 freq      -2482 path delay   1826
ptp4l[879.004]: master offset          -9 s2 freq      -2507 path delay   1826
ptp4l[879.129]: master offset         -24 s2 freq      -2530 path delay   1826
ptp4l[879.255]: master offset          -7 s2 freq      -2508 path delay   1826
ptp4l[879.380]: master offset          -2 s2 freq      -2502 path delay   1826
ptp4l[879.505]: master offset         -17 s2 freq      -2524 path delay   1827
ptp4l[879.630]: master offset           6 s2 freq      -2493 path delay   1827
ptp4l[879.755]: master offset           6 s2 freq      -2492 path delay   1827
ptp4l[879.880]: master offset           0 s2 freq      -2500 path delay   1827
```

ptp4l 的一些其他选项

```
Delay Mechanism
-E      E2E, delay request-response (default)
-P      P2P, peer delay mechanism

Network Transport
-2      IEEE 802.3
-4      UDP IPV4 (default)
-6      UDP IPV6
```

注意：必须保持在这些板上使用相同的延迟机制和网络传输协议。

配置 master 模式

默认情况下，master 时钟由 BMC（最佳 Master 时钟）算法选择。要将特定时钟指定为 master，可以设置比其他时钟低的“priority1”属性值。较低的值优先。例如，在当前案例中，使用以下选项指定一个时钟作为 master。（其他时钟使用默认的 priority1 值 128。）

```
--priority1=127
```

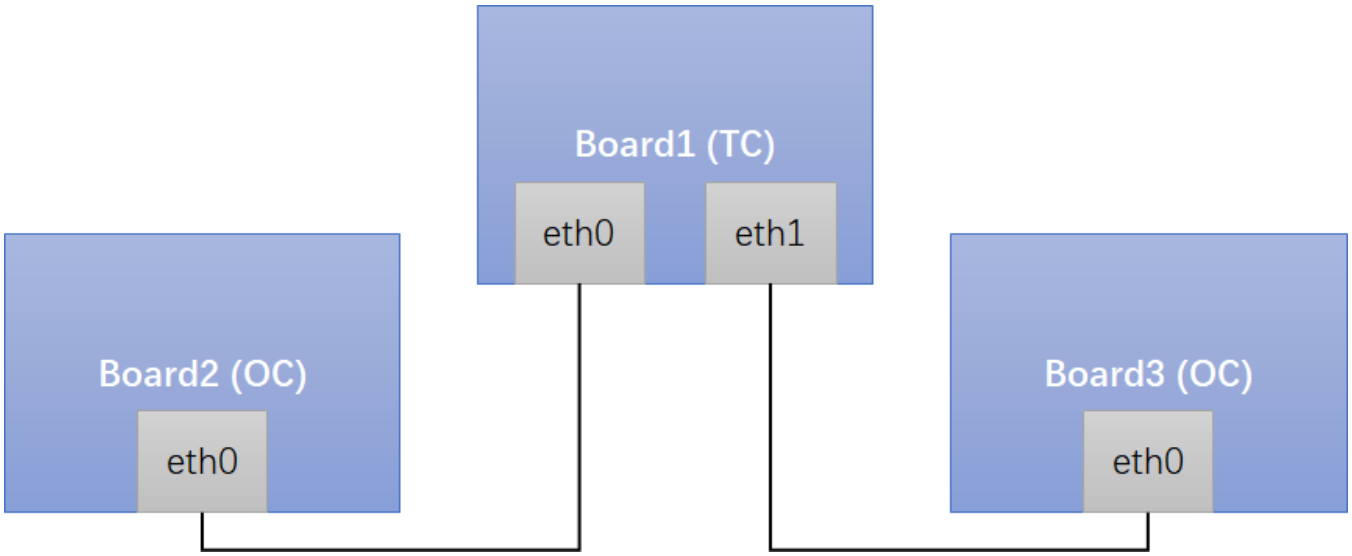
单步时间戳

目前仅 DPAA2 支持单步时间戳。要使用单步时间戳，请添加以下选项以运行 ptp4l。

```
--twoStepFlag=0
```

4.3.5.3 透明时钟验证

至少需要三个板。下面是一个三板网络连接的例子。确保板上没有 MAC 地址冲突，IP 地址设置正确，并 ping 测试网络。



在板 1（透明时钟）上运行 `linuxptp`。如果希望板 1 作为 E2E TC，请使用 E2E-TC.cfg。如果希望板 1 作为 P2P TC，请使用 P2P-TC.cfg。

```
$ ptp4l -i eth0 -i eth1 -f /etc/linuxptp/E2E-TC.cfg -m
```

在板 2/板 3（普通时钟）上运行 `linuxptp`。

```
$ ptp4l -i eth0 -m
```

运行上面的命令时，普通时钟之间会开始时间同步，自动选择的 slave linuxptp 会同步到 master，并显示同步信息，如时间偏移量、路径延迟等。

4.3.6 IEEE 802.1AS 快速入门

以下几节介绍了在恩智浦板上实施 IEEE 802.1AS 的步骤。以下步骤使用 linuxptp 协议栈，但可以使用恩智浦 GenAVB/TSN gPTP 协议栈在支持的板上执行类似的命令，如此处所述。

4.3.6.1 时间感知终端节点验证

将两块板的两个网络接口背靠背连接。确保板上没有 MAC 地址冲突，IP 地址设置正确，并 ping 测试正常。

删除/etc/linuxptp/gPTP.cfg 中的以下选项以使用默认较大的值，因为估计的路径延迟（包括 PHY 延迟）可能超过 800ns，因为硬件使用 MAC 时间戳。

```
neighborPropDelayThresh 800
```

在每个板上运行 linuxptp。例如，在每个板上使用 eth0。

```
$ ptp4l -i eth0 -f /etc/linuxptp/gPTP.cfg -m
```

时间同步将开始，自动选择的 slave linuxptp 会同步到 master，并打印同步消息，如时间偏移量、路径延迟等。

4.3.6.2 时间感知网桥验证

时间感知网桥验证至少需要三块板。下面是三块板之间的网络连接示例。确保板上没有 MAC 地址冲突。

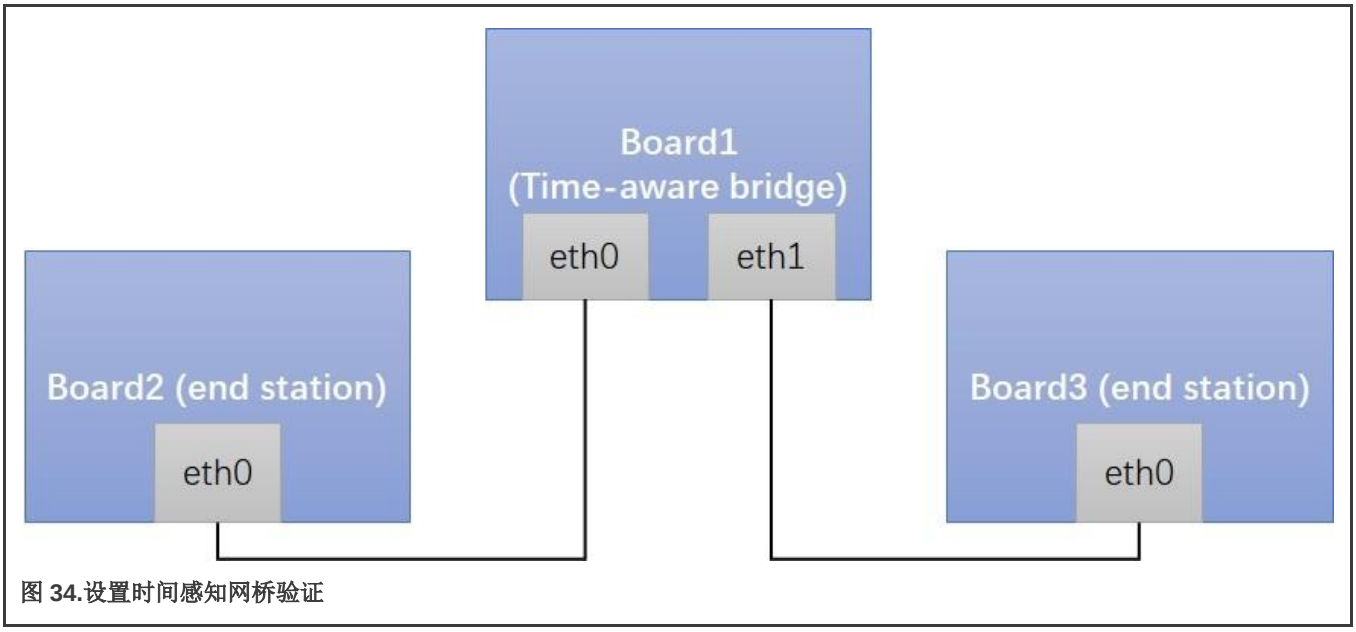


图 34.设置时间感知网桥验证

删除`/etc/linuxptp/gPTP.cfg`文件中的以下选项，以使用默认的较大值，因为估计的路径延迟（包括 PHY 延迟）可能超过 800 ns，因为硬件正在使用 MAC 时间戳。

```
neighborPropDelayThresh 800
```

使用以下命令在板 1（时间感知网桥）上运行 linuxptp：

```
$ ptp4l -i eth0 -i eth1 -f /etc/linuxptp/gPTP.cfg -m
```

使用以下命令在板 2/板 3（时间感知终端节点）上运行 linuxptp：

```
$ ptp4l -i eth0 -f /etc/linuxptp/gPTP.cfg -m
```

三块板之间会开始时间同步，选择的 linuxptp slave 会自动同步到特有的 master，并显示同步消息（如时间偏移量、路径延迟等）。

4.3.7 长期测试

本节介绍 Linux PTP 协议栈实施的长期测试结果。

4.3.7.1 linuxptp 基本同步

Linux PTP

连接：背靠背 master 至 slave

配置：同步内部为 -3

测试板：两块 LS1021A-TSN 板，一个作为 master，一个作为 slave。

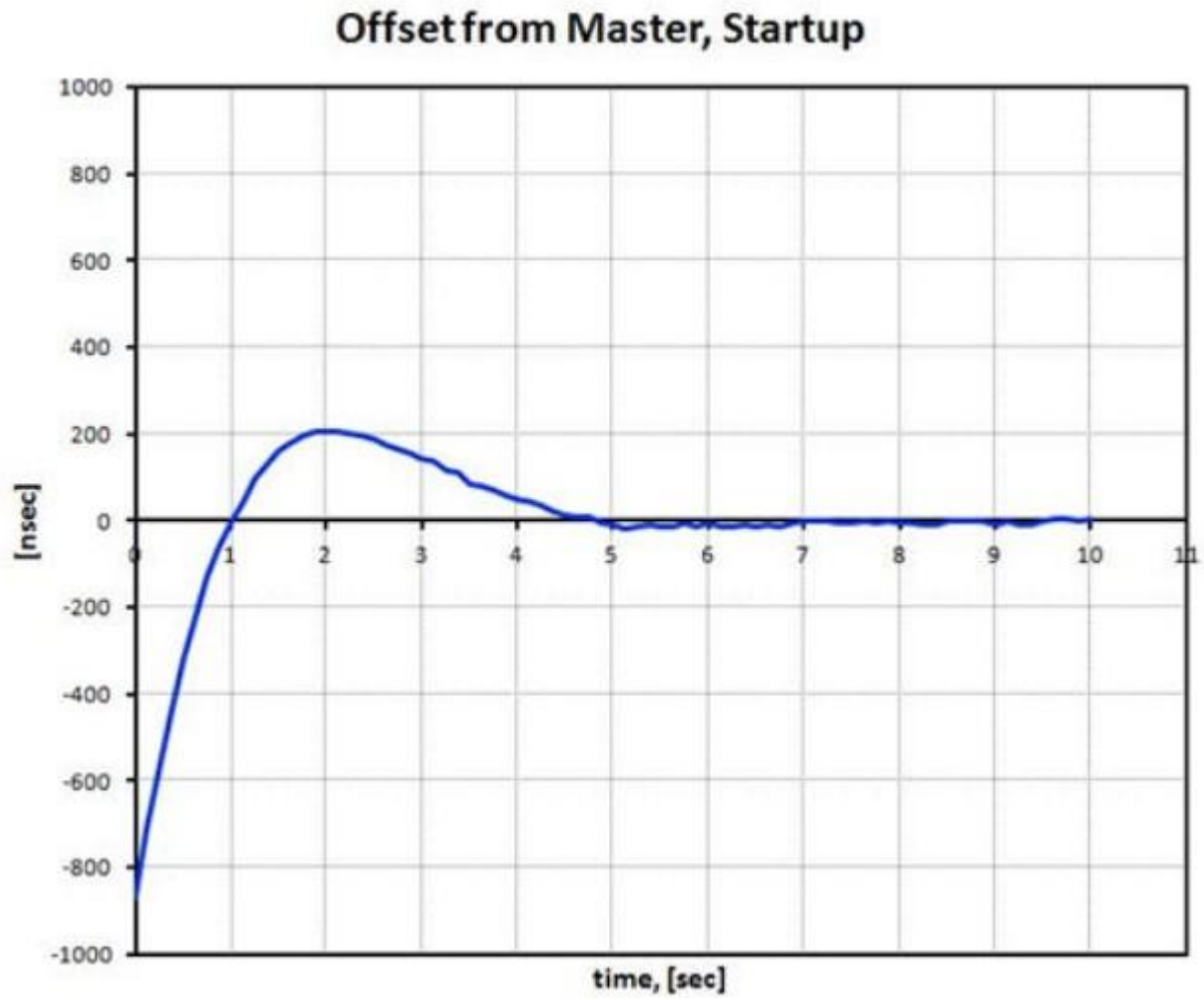


图 35.在启动状态下距 master 的偏移量

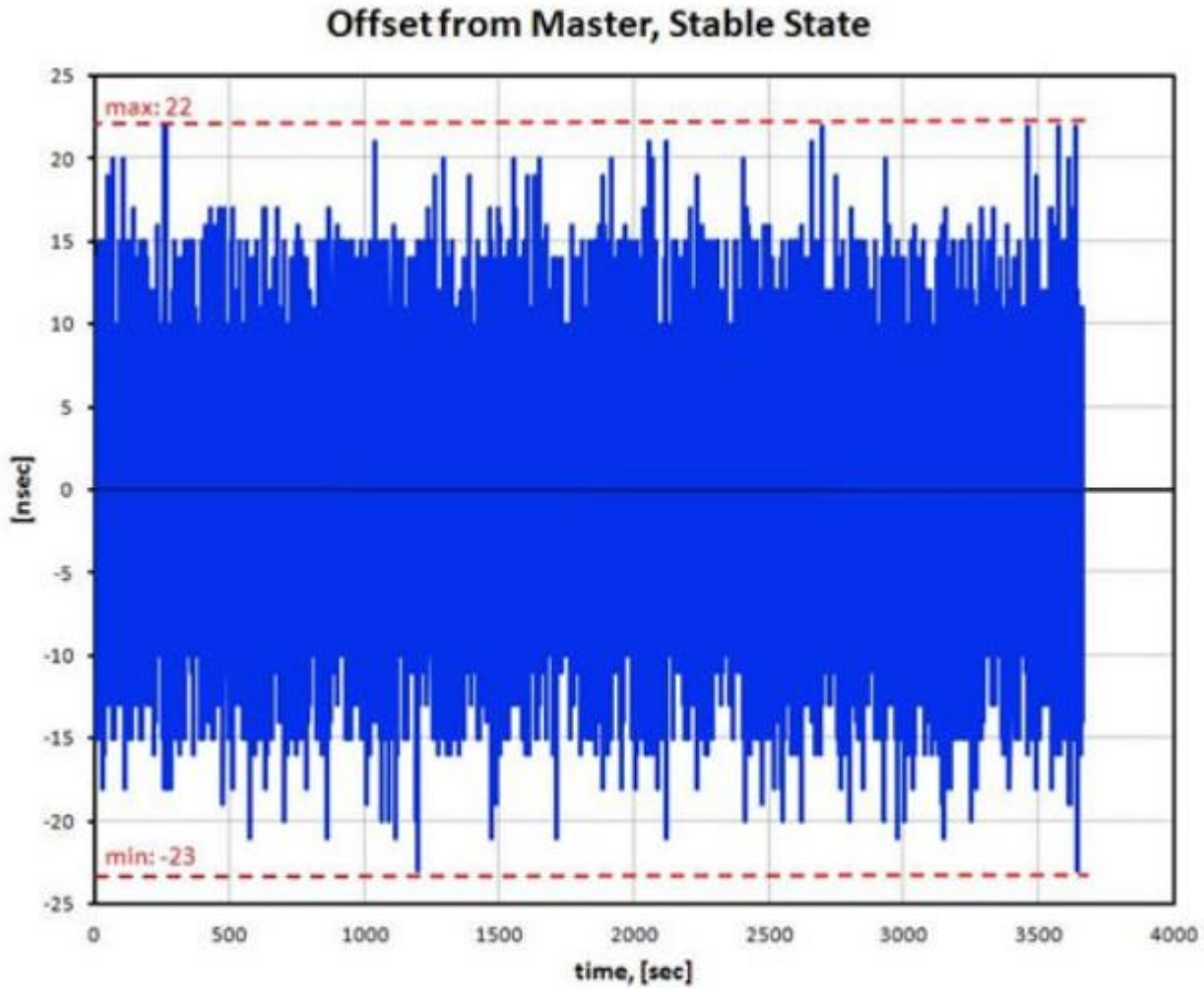


图 36.在稳定状态下距 master 的偏移量

4.3.8 已知问题和限制

1.Linux 中的 LS1028A TSN 交换机配置为 L2 交换机时，接口不应配置 IP 地址。在这些接口上运行 linuxptp 必须使用以太网协议而不是 UDP/IP。方法是添加一个选项“-2”执行 ptp4l 命令。例如，

```
$ ptp4l -i eth0 -2 -m
```

2.i.MX 8M Plus 当前的 dwmac 驱动程序(eth1)在打开网络设备时会初始化一些硬件功能，包括 PTP 初始化。在此之前，对它的操作可能无效，例如 ethtool 查询和 PTP 操作。所以，解决方法是，只在“`ifconfig eth1 up`”之后对 dwmac 的 eth1 和 PTP 进行操作。

3.如果在 ptp4l 运行过程中报告以下错误，请尝试增加 `tx_timestamp_timeout`。用户空间可能需要等待更长时间来获取 TX 时间戳。

例如，在运行 ptp4l 时使用选项`--tx_timestamp_timeout=20`，如下所示：

```
ptp4l[1560.726]: timed out while polling for tx timestamp
ptp4l[1560.726]: increasing tx_timestamp_timeout may correct this issue, but it is likely caused by a
driver bug
```

4.4 网络服务

4.4.1 LS1028A Felix 交换机上的 Q-in-Q

1. **Q-in-Q 功能**

   Q-in-Q 功能允许服务提供商在两个用户站点之间创建第 2 层以太网连接。提供商可以使用不同的用户 VLAN 在链路或捆绑包上隔离不同用户的 VLAN 流量。使用 Q-in-Q 时，用户的 802.1Q VLAN 标记 `(C-TAG:0x8100)` 前为服务 VLAN 标记 `(S-TAG: 0x88A8)`。

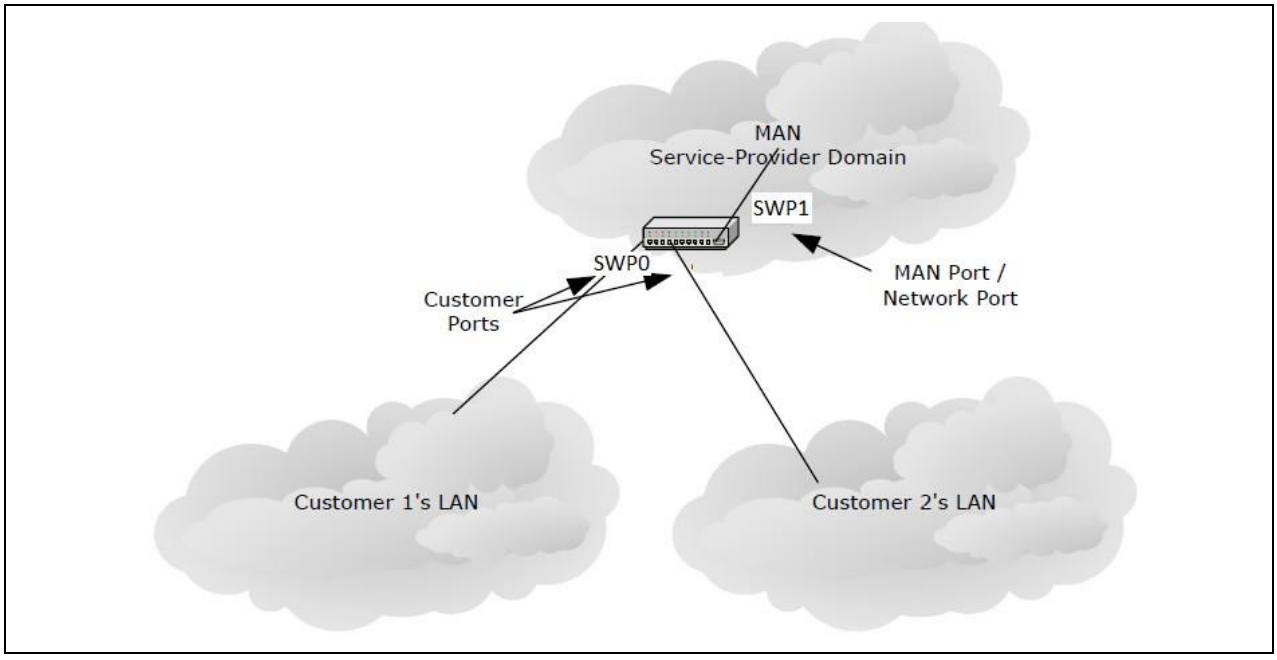
2. **Q-in-Q 应用场景**

   在以下场景中，交换机的端口 swp0 连接客户 1 的 LAN，swp1 连接 ISP 的 MAN。

   带有 VLAN 标记的流量如下所示：

   上行链路：客户 LAN `(only C-TAG) -> swp0 -> swp1 (add S-TAG) -> ISP MAN (S-TAG + C-TAG)`

   下行链路：ISP MAN `(S-TAG + C-TAG) -> swp1 (pop S-TAG)-> swp0 (only C-TAG) -> Customer LAN`



3. Q-in-Q 配置示例

   a. 使能 swp1 Q-in-Q 模式

   ```
   devlink dev param set pci/0000:00:00.5 name qinq_port_bitmap value 2 cmode runtime
   ```

   注意

   - 0000:00:00.5是ocelot交换机的PCIe总线和设备编号。
   - 值2是端口1的位图。如果端口n链接到ISP MAN，则相关位n应设置为1。

   b. 创建网桥并添加端口：

   ```
   ip link add dev br0 type bridge vlan_protocol 802.1ad
   ip link set dev swp0 master br0
   ip link set dev swp1 master br0
   ip link set dev br0 type bridge vlan_filtering 1
   ```

c. 设置 swp0 pvid 并为出口流量设置“非标记”：

```
bridge vlan del dev swp0 vid 1 pvid
bridge vlan add dev swp0 vid 100 pvid untagged
bridge vlan add dev swp1 vid 100
```

d. 结果

```
Customer(tpid:8100 vid:111) -> swp0 -> swp1 -> ISP(STAG tpid:88A8 vid:100,CTAG
tpid:8100 vid:111)
ISP(tpid:88A8 vid:100 tpid:8100 vid:222) -> swp1 -> swp0 -> Customer(tpid:8100 vid:222)
```

4.4.2 LS1028A Felix 交换机上的 VCAP

VCAP 是用于线速数据包检查的内容感知数据包处理器。它使用“tc flower”命令来设置过滤器和操作。LS1028A 支持以下键和操作：

键：

vlan_id

vlan_prio

dst_mac/src_mac for non IP frames

dst_ip/src_ip

dst_port/src_port

操作：

trap

drop

police

vlan modify

vlan push(Egress)

使用以下命令设置、获取和删除 VCAP 规则：

```
tc qdisc add dev swp0 clsact
tc filter add dev swp0 ingress chain [chain-id] protocol [ip/802.1Q] flower skip_sw [keys]
action [actions]
tc -s filter show dev swp0 ingress chain [chain-id]
tc filter del dev swp0 ingress chain [chain-id] pref [pref_id]

tc qdisc add dev swp1 clsact
tc filter add dev swp1 egress protocol 802.1Q flower skip_sw [keys] action vlan push id [value]
priority [value]
tc filter show dev swp1 egress
tc filter del dev swp1 egress pref [pref_id]
```

有两个入口 VCAP 和一个出口 VCAP。tc-flower 链用于 LS1028A 入口端口。每个操作都有一个固定的链。以下是链分配：

表 59.链分配

链 ID	操作	硬件模块	键
10000	skbedit 优先级	IS1 查找 0	源 MAC 地址、 源 IP 地址（32 位）、 外层 VLAN、IP 协议、

表格接下页……

表 59.链分配（续）

链 ID	操作	硬件模块	键
			源 TCP/UDP 端口。
11000	vlan 弹出窗口；vlan 修改	IS1 查找 1	内层和外层 VLAN、 源和目标 IP 地址（32 位）、 IP 协议、 源和目标 TCP/UDP 端口。
12000	转到链[PAG]	IS1 查找 2	源 MAC 地址、 源 IP 地址（32 位）、 外层 VLAN、IP 协议、 源 TCP/UDP 端口。
20000-20255	监管	IS2 查找 0	源和目标 MAC 地址、 源和目标 IP 地址（32 位）、 IP 协议、 源和目标 TCP/UDP 端口。
21000-21255	丢弃；捕捉；重定向	IS2 查找 1	源和目标 MAC 地址、 源和目标 IP 地址（32 位）、 IP 协议、 源和目标 TCP/UDP 端口。
30000	门；监管	PSFP	目标 MAC 地址和 VLAN ID

在使用链之前，用户应该注册每个链并为数据包设置链流水线顺序。硬件入口顺序为：**IS1->IS2->PSFP。**

```
tc qdisc add dev swp0 clsact
tc filter add dev swp0 ingress chain 0 pref 49152 flower skip_sw action goto chain 10000
tc filter add dev swp0 ingress chain 10000 pref 49152 flower skip_sw action goto chain 11000
tc filter add dev swp0 ingress chain 11000 pref 49152 flower skip_sw action goto chain 12000
tc filter add dev swp0 ingress chain 12000 pref 49152 flower skip_sw action goto chain 20000
tc filter add dev swp0 ingress chain 20000 pref 49152 flower skip_sw action goto chain 21000
tc filter add dev swp0 ingress chain 21000 pref 49152 flower skip_sw action goto chain 30000
```

注册链后，将规则添加到对应的链中。以下是测试用例：

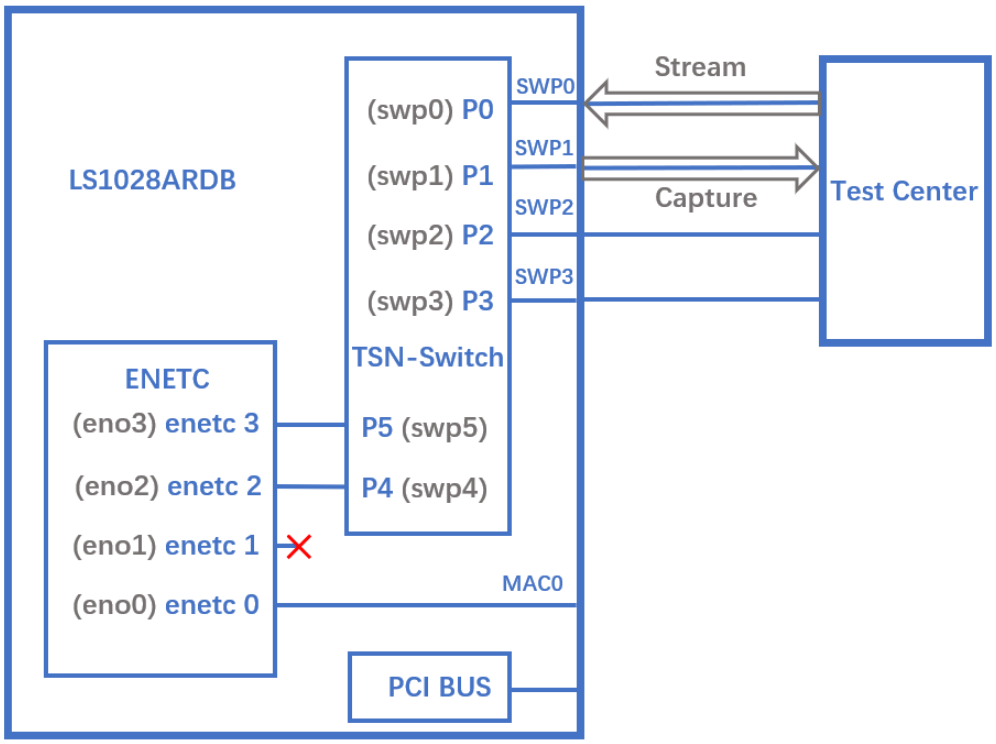


图 37.VCAP 测试

1. 丢弃来自源 IP 192.168.2.1 的所有帧。

```
tc filter add dev swp0 ingress chain 21000 protocol ip flower skip_sw src_ip 192.168.2.1
action drop
```

将源 IP 设置为 192.168.2.1 并从测试中心发送 IP 包，包将被丢弃在 `swp0` 上。

2. 将 HTTP 流的带宽限制为 10 Mbps。

```
tc filter add dev swp0 ingress chain 20000 protocol ip flower skip_sw ip_proto tcp dst_port 80
action police rate 10Mbit burst 10000 action goto chain21000
```

发送 TCP 包，在测试中心上设置目标端口为 80，将流带宽设置为 1Gbit/s，可以得到 10Mbits/s 的流速率。

3. 过滤具有特定 vlan 标记（VID=1 和 PCP=1）的帧。然后，修改 vlan 标记（VID=2，PCP=2），分类为 QoS 流量类别 2。

```
ip link set switch type bridge vlan_filtering 1
tc filter add dev swp0 ingress chain 11000 protocol 802.1Q flower skip_sw vlan_id 1 vlan_prio 1
action vlan modify id 2 priority 2 action goto chain 12000
bridge vlan add dev swp0 vid 2
bridge vlan add dev swp1 vid 2
```

在 vlan 标记中设置 `vid=1` 和 `pcp=1`。然后，从测试中心发送 IP 包。因此，您可以从测试中心上的 `swp1` 获得 vid=2、pcp=2 的包。

4. 将特定的 vlan 标记（vid=3、pcp=3）推送到来自 swp1 的帧（分类 vid=2，pcp=2 在交换机）出口。

```
tc qdisc add dev swp1 clsact
tc filter add dev swp1 egress protocol 802.1Q flower skip_sw vlan_id 2 vlan_prio 2 action vlan
push id 3 priority 3
```

在 vlan 标记中设置 vid=1 和 pcp=1，然后从测试中心发送 IP 包，该帧将在用例 3 中符合规则并重新标记 vlan(vid=2、pcp=2)。因此，用户可以从测试中心的 swp1 中获取 vid=3、pcp=3 的帧。

5. 将双 vlan 标记(Q-in-Q)推入接至 swp1 的帧出口。

```
ip link add dev br0 type bridge
ip link set dev swp0 master br0
ip link set dev swp1 master br0
ip link set br0 type bridge vlan_filtering 1
bridge vlan add dev swp0 vid 222
bridge vlan add dev swp1 vid 222
tc qdisc add dev swp1 clsact
tc filter add dev swp1 egress protocol 802.1Q flower skip_sw \
     vlan_id 222 vlan_prio 2 \
     action vlan push id 200 priority 1 protocol 802.1AD \
     action vlan push id 300 priority 3
```

结果： TX(tpid:8100 vid:222 pri:2) -> swp0 -> swp1 -> RX(S-TAG tpid:88A8 vid:200 pri:1, C-TAG tpid:8100 vid:300 pri:3)

6. 从来自 swp0 的帧入口中弹出单或双 vlan 标记(Q-in-Q)。

```
ip link add dev br0 type bridge
ip link set dev swp0 master br0
ip link set dev swp1 master br0
tc filter add dev swp0 ingress chain 11000 \
     protocol 802.1ad flower \
     vlan_id 111 vlan_prio 1 vlan_ethtype 802.1q \
     cvlan_id 222 cvlan_prio 2 cvlan_ethtype ipv4 \
     action vlan pop action goto chain 12000
```

结果： TX(S-TAG tpid:88A8 vid:111 pri:1, C-TAG tpid:8100 vid:222 pri:2) -> swp0 -> swp1 -> RX(TAG tpid:8100 vid:222 pri:2)

```
tc filter add dev swp0 ingress chain 11000 \
     protocol 802.1ad flower \
     vlan_id 111 vlan_prio 1 vlan_ethtype 802.1q \
     cvlan_id 223 cvlan_prio 2 cvlan_ethtype ipv4 \
     action vlan pop \
     action vlan pop action goto chain 12000
```

结果： TX(S-TAG tpid:88A8 vid:111 pri:1, C-TAG tpid:8100 vid:223 pri:2) -> swp0 -> swp1 -> RX(received packets without VLAN tag)

第 5 章
协议

5.1 EtherCAT

实时边缘支持使用 EtherCAT（用于控制自动化技术的以太网）并集成了 IGH EtherCAT master 协议栈。EtherCAT 在恩智浦的平台上得到验证。

5.1.1 简介

EtherCAT 是一种基于以太网的现场总线系统，由 BECKHOFF Automation 发明。该协议在 IEC 61158 中进行了标准化，适用于自动化技术中的软硬实时计算要求。EtherCAT 开发期间的目标是将以太网应用于需要较短数据更新时间（也称为循环时间；≤ 100 µs）的自动化应用，通信抖动低（用于精确同步；≤ 1 µs）并降低硬件成本。

- EtherCAT 速度快：1000 个数字。I/O: 30 µs，100 个 slave：100 µs。
- EtherCAT 是以太网：I/O 级别的标准以太网。
- EtherCAT 非常灵活：星型、线型、丢弃，带或不带交换机。
- EtherCAT 价格低廉：以太网是主流技术，因此价格低廉。
- EtherCAT 很简单：每个人都知道以太网，它使用起来很简单。

目前 EtherCAT master 支持 RT-LAB 开发的 SOEM（简单开源 EtherCAT Master）和 EtherLab、IGH EtherCAT master 的通用开源代码。使用 SOEM 比使用 IGH EtherCAT Master 更简单，但用于实现 EtherCAT 的 IGH 更完整。例如，IGH 支持更多的 NIC。有关更多信息，请参见 https://rt-labs.com/ethercat/ 和 http://www.etherlab.org。实时边缘中的集成是 IGH EtherCAT master。

5.1.2 IGH EtherCAT 架构

Master 环境的组件如下所述：

- **Master 模块：**这是包含一个或多个 EtherCAT master 实例、“设备接口”和“应用程序接口”的内核模块。
- **设备模块：**这些是支持 EtherCAT 的以太网设备驱动程序模块，可通过设备接口将其设备提供给 EtherCAT master。这些修改后的网络驱动程序可以并行处理用于 EtherCAT 操作的网络设备和“普通”以太网设备。Master 可以接受某个设备，然后能够发送和接收 EtherCAT 帧。像往常一样，被 master 模块拒绝的以太网设备连接到内核的网络协议栈。
- **应用程序：**使用 EtherCAT master 的程序（通常用于与 EtherCAT slave 循环交换过程数据）。这些程序不是 EtherCAT master 代码的一部分，但需要由用户生成或编写。应用程序可以通过应用程序接口请求 master。如果成功，它就可以控制 master：它可以提供总线配置和交换过程数据。应用程序可以是直接使用内核应用程序接口的内核模块。它们还包括用户空间程序，这些程序通过 EtherCAT 库或 RTDM 库使用应用程序接口。

  下图显示了 IGH EtherCAT master 架构。

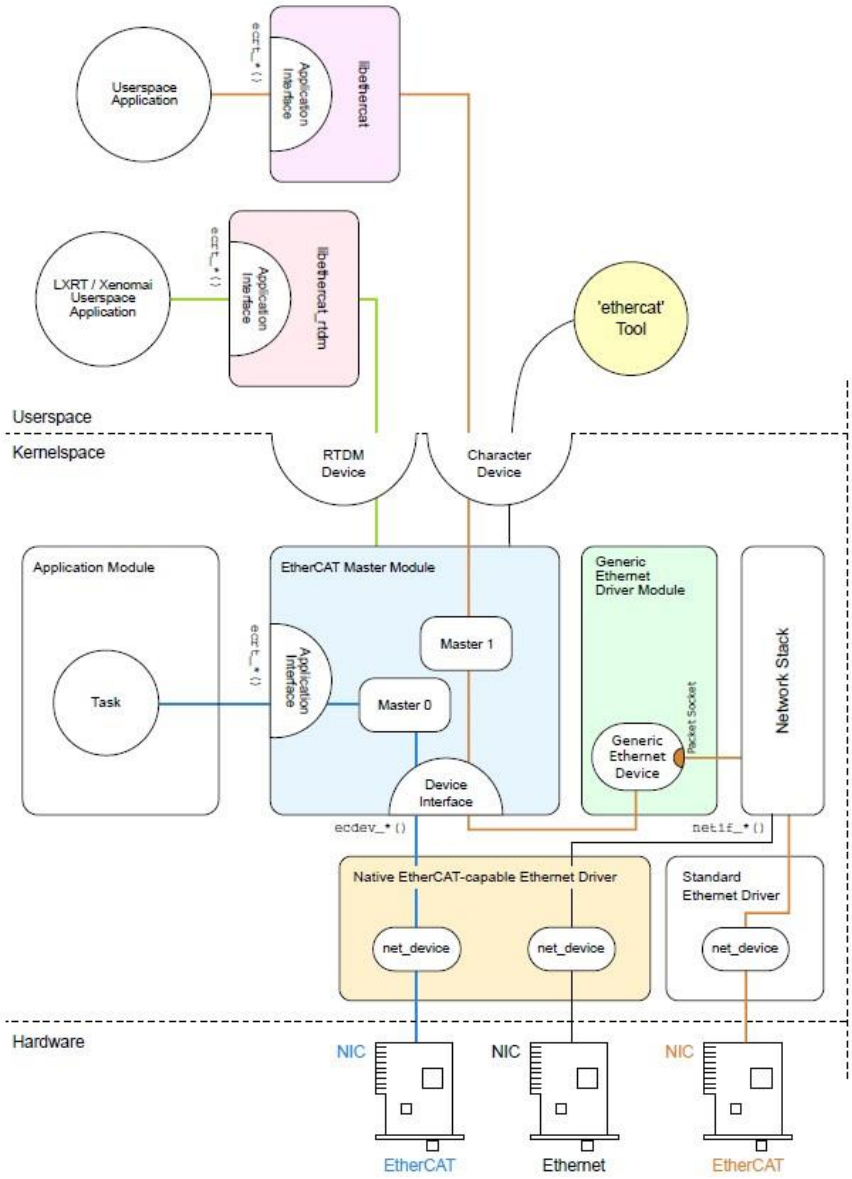


图 38.IGH EtherCAT master 架构

5.1.3 EtherCAT 协议

以下是 EtherCAT 协议的特点：

- EtherCAT 协议针对过程数据进行了优化，并使用 Ethertype 0x88a4 在标准 IEEE 802.3 以太网帧内直接传输。
- 数据顺序与网络中节点的物理顺序无关；寻址可以是任何顺序。
- slave 之间可以进行广播、多播和通信，但必须由 master 设备发起。
- 如果需要 IP 路由，可以将 EtherCAT 协议插入 UDP/IP 数据报。这也使任何带有以太网协议栈的控制都能够处理 EtherCAT 系统。
- 它不支持缩短的帧。

下图显示了 EtherCAT 帧结构。

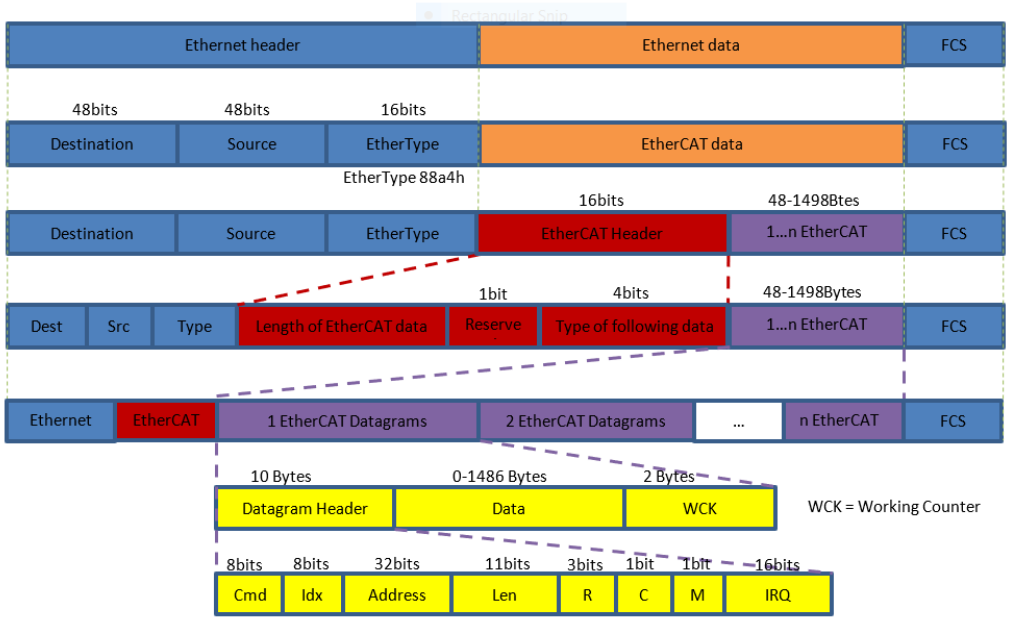


图 39.EtherCAT 帧结构

5.1.4 IGH EtherCAT 设备模块

有两种设备驱动程序模块：

1.原生以太网设备驱动程序

在实时边缘软件中，仅提供了一个原生驱动程序“ec_fec”，可用于 i.MX 8M Mini EVK 上的 FEC MAC。此驱动程序未在此版本的其他 i.MX 平台上得到验证。

注意：使用 ec_fec 原生驱动程序时，必须通过使用命令“make menuconfig”重新配置 Linux，将原始以太网 fec 驱动程序重新编译为模块。

2.通用以太网设备驱动程序

通用驱动程序使用 Linux 网络协议栈的较低层连接到硬件，独立于实际的硬件驱动程序。因此它可以用于实时边缘支持的所有平台。但缺点是性能比原生驱动程序略差，因为以太网帧数据要遍历 Linux 网络协议栈。

5.1.5 IGH EtherCAT 集成和设置

本节介绍如何构建 IGH 协议栈以及如何设置 EtherCAT 服务。

5.1.5.1 构建 IGH EtherCAT

IGH EtherCAT 包默认在实时边缘镜像上使能。i.MX 8M Mini EVK 默认使能原生驱动程序 ec_fec 模块。对于其他平台，仅支持通用驱动程序。

5.1.5.2 IGH EtherCAT 设置

1.配置

第一次需要配置/etc/ethercat.conf 配置文件，如下所示：

- 将“MASTER0_DEVICE”设置为以太网卡的 MAC 地址，该地址将用作 EtherCAT 网络端口。例如：

```
MASTER0_DEVICE="00:04:9f:07:11:a6"
```

- 通过 MAC 地址为上面选择的 EtherCAT 设备指定设备驱动程序模块。

```
DEVICE_MODULES="generic"
```

对于 i.MX 8M Mini EVK，“fec”选项可用。但请注意，如果选择“fec”，则必须通过重新配置 Linux menuconfig 将原始以太网 fec 驱动程序编译为模块。在实时边缘上，fec 以太网驱动程序已默认配置为 i.MX 8M Mini EVK 的模块。

2.启动守护进程

使用以下命令启动 IGH-EtherCAT 守护进程：

```
$ ethercatctl start
```

此外，以下命令用于停止或重启守护进程。

```
$ ethercatctl stop
$ ethercatctl restart
```

注：如果使用通用驱动程序，请确保使用“ifconfig<ethX> up”命令使能网卡。

IGH 提供了一个强大的辅助命令行工具，名为“ethercat”。可用于查询 master 和所有 slave 的信息和状态。用法如下：

```
用途：ethercat <COMMAND> [OPTIONS] [ARGUMENTS]

     命令（可以缩写）：
          alias            写入别名地址。
          config           显示 slave 配置。
          crc              CRC 错误寄存器诊断。
          cstruct          用 C 语言生成 slave PDO 信息。
          data             输出二进制功能域过程数据。
          debug            设置 master 的调试级别。
          domains          显示配置的功能域。
          download         将 SDO 条目写入 slave。
          eoe              通过 EtherCAT 统计数据显示以太网。
          foe_read         通过 FoE 从 slave 读取文件。
          foe_write        通过 FoE 将文件存储在 slave 上。
          graph            将总线拓扑输出为图形。
          master           显示 master 和以太网设备信息。
          pdos             列出同步管理器、PDO 分配和映射。
          reg_read         输出 slave 的寄存器内容。
          reg_write        将数据写入 slave 的寄存器。
          rescan           重新扫描总线。
          sdos             列出 SDO 字典。
          sii_read         输出 slave 的 SII 内容。
          sii_write        将 SII 内容写入 slave。
          slaves           显示总线上的 slave。
          soe_read         从 slave 读取 SoE IDN。
          soe_write        将 SoE IDN 写入 slave。
          states           请求应用层状态。
          upload           从 slave 读取 SDO 条目。
          version          显示版本信息。
          xml              生成 slave 信息 XML。
```

3.使能 systemd IGH-EtherCAT 服务

实时边缘提供 IGH-EtherCAT systemd 服务，以将 IGH 守护进程作为系统服务运行。

```
$ systemctl enable ethercat
$ systemctl start ethercat
```

以下命令也用于停止或禁用此服务：

```
$ systemctl stop ethercat
$ systemctl disable ethercat
```

5.1.6 real-time-edge-servo 协议栈

real-time-edge-servo 是基于 Igh CoE 接口的 CiA402（也称为 DS402）配置文件框架（EtherCAT Master 协议栈，详见 EtherCAT 一节）。它抽象了 CiA 402 配置文件，并为应用程序开发人员提供了一个易于使用的 API。

real-time-edge-servo 项目包含基本库 *libnservo* 和几个辅助工具。

使用 *libnservo* 开发的应用程序非常灵活，可以通过修改应用程序启动时加载的 *xml* 配置文件来适应 CoE 网络的变化。*xml* 配置文件说明了必要的信息，包括 EtherCAT 网络拓扑、slave 配置、master 配置和所有轴定义。

该协议栈已在以下 CoE 伺服系统生产中进行了测试：DELTA ASDA-B3、HCFA SV-X6EB 和 SV-X3EB、仅运动控制 2HSS458-EC。

5.1.6.1 CoE 网络

典型的 CoE 网络如下图所示：

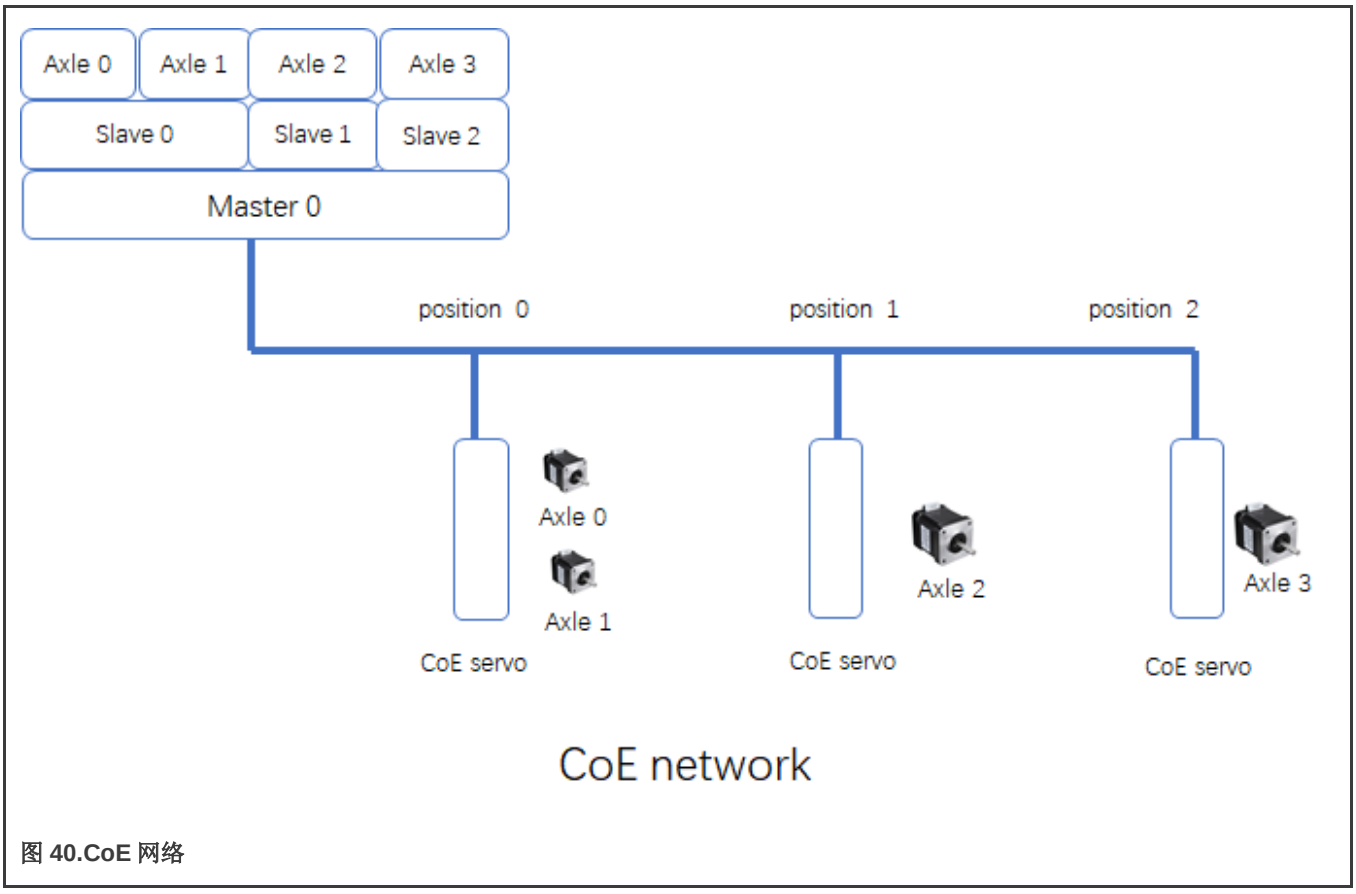


图 40.CoE 网络

该网络上有三个 CoE 伺服系统，我们按照它们所在的位置将它们命名为 slave *x*。每个 CoE 伺服系统可以有多个轴。然后 libnservo 启动 CoE 网络并将网络拓扑的详细信息封装到轴节点中。因此，开发人员可以专注于每个轴操作，而无需考虑网络拓扑。

5.1.6.2 Libnservo 架构

real-time-edge-servo 在 *Igh* EtherCAT 协议栈顶部运行。并且 *Igh* 协议栈提供 CoE 通信机制——邮箱和过程数据。使用这些机制，实时边缘伺服系统可以访问位于 CoE 伺服系统上的 CiA 对象字典。

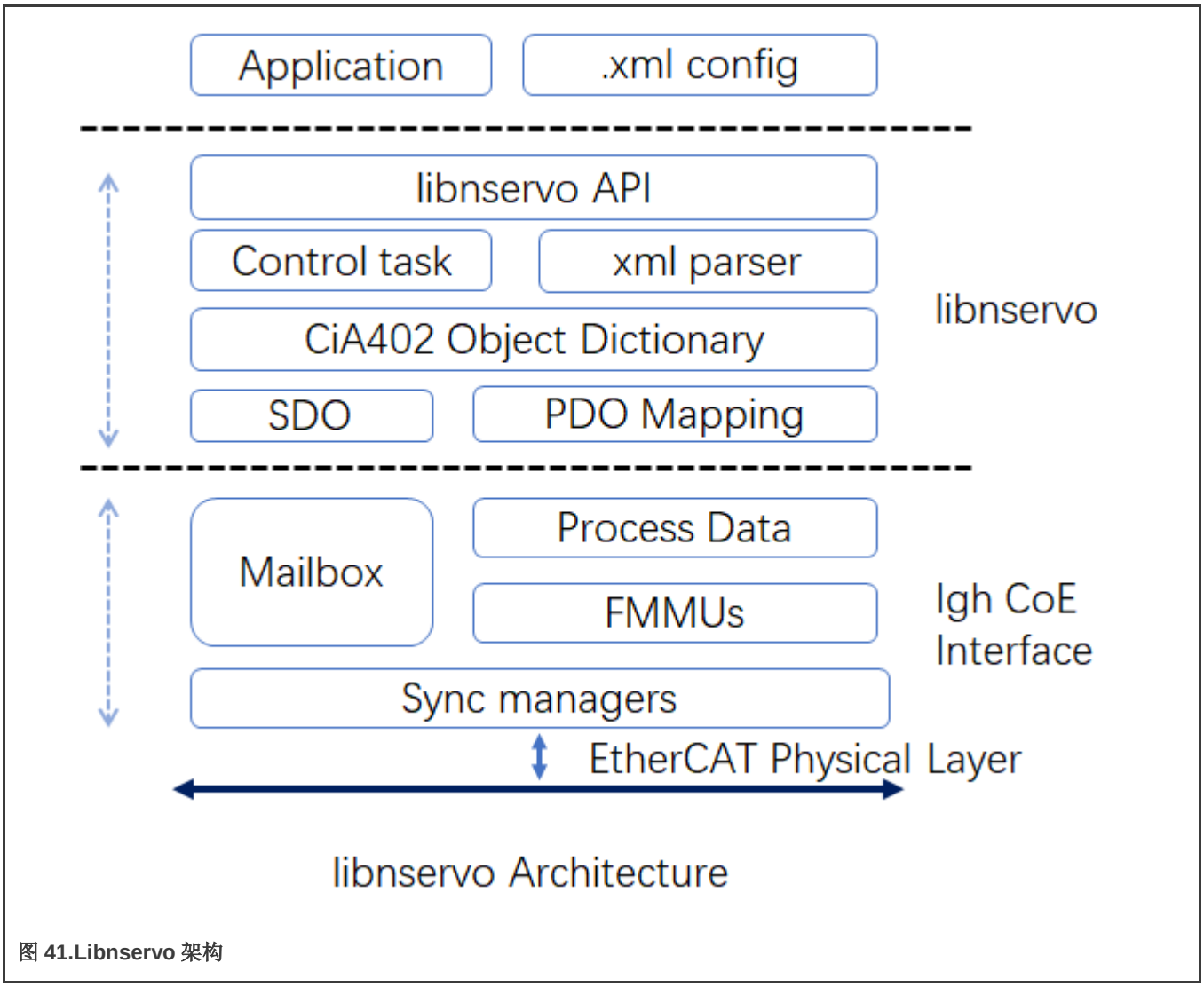


图 41.Libnservo 架构

控制任务启动 CoE 网络上的 master 和所有 slave，并将所有 PDO 注册到 Igh 协议栈，然后构造一个数据结构来描述每个轴。最后，控制任务创建一个任务来定期运行用户任务。

5.1.6.3 real-time-edge-servo Xml 配置

本节重点介绍 xml 配置文件如何描述 CoE 网络。

XML 配置的骨架如下图所示：

```
<?xml version="1.0" encoding="utf-8"?>
<Config Version="1.2">
    <PeriodTime>#10000000</PeriodTime>
```

```
    <MaxSafeStack>#8192</MaxSafeStack>
    <master_status_update_freq>#1</master_status_update_freq>
    <slave_status_update_freq>#1</slave_status_update_freq>
    <axle_status_update_freq>#1</axle_status_update_freq>
    <sync_ref_update_freq>#2</sync_ref_update_freq>
    <sched_priority>#90</sched_priority>
        <sched_policy>#SCHED_FIFO</sched_policy>
    <Masters>
        <Master>
          ...
        <\Master>
        <Master>
          ...
        <\Master>
    <\Master>

    <Axles>
        <Axle>
          ...
        <\Axle>
        <Axle>
          ...
        <\Axle>
    <\Axles>
</Config>
```

- 所有配置元素必须在<Config>元素内。
- 上面显示的所有配置元素都是强制性的。
- 以#开头的数值表示它是十进制值。
- 以#x 开头的数值表示它是一个十六进制值。
- <PeriodTime>元素表示控制任务的周期为 10ms。
- <MaxSafeStack>表示协议栈大小，它是一个估计值。8K 足以满足大多数应用。
- <master_status_update_freq>元素表示 master 状态更新的频率。值#x 表示每个任务周期更新 master 状态。
- <slave_status_update_freq>元素表示 slave 状态更新的频率。值#1 表示每个任务周期更新 slave 状态。
- <axle_status_update_freq>元素表示轴状态更新的频率。值#1 表示每个任务周期更新轴状态。
- <sync_ref_update_freq>元素表示参考时钟更新的频率。值#2 表示每两个任务周期更新轴状态。
- <sched_policy>元素指定用户任务的调度策略。
- <sched_priority>元素表示用户任务的优先级。
- <Masters>元素可以包含多个 Master 元素。大多数情况下，主机上只有一个 master。
- <Axles>元素可以包含多个轴元素，这是开发人员真正关心的内容。

5.1.6.3.1 Master 元素

如 *CoE 网络*一节所示，Master 可以有许多 slave，因此 Master 元素将由一些 *Slave* 元素组成。

```
    <Master>
        <Master_index>#0</Master_index>
        <Reference_clock>#0</Reference_clock>
```

```
<Slave  alias="#0" slave_position="#0">
          ....
     </Slave>
     <Slave  alias="#1" slave_position="#1">
          ....
     </Slave>
</Master>
```

- <Master_index>元素表示 master 的索引。如上所述，很多情况下只有一个 master，所以这个元素的值总是#0。
- <Reference_clock>元素用于指示将使用哪个 slave 作为参考时钟。
- <Slave>元素表示此 master 上有一个 slave。

5.1.6.3.1.1 Slave 元素

```
<Slave  alias="#0" slave_position="#0">
<VendorId>#x66668888</VendorId>
<ProductCode>#x20181302</ProductCode>
     <Name>2HSS458-EC</Name>
     <Emerg_size>#x08</Emerg_size>
<WatchDog>
     <Divider>#x0</Divider>
     <Intervals>#4000</Intervals>
</WatchDog>
<DC>
     <SYNC SubIndex='#0'>
          <Shift>#0</Shift>
     </SYNC>
</DC>
<SyncManagers force_pdo_assign="#1">
     <SyncManager SubIndex="#0">
               ...
          </SyncManager>
     <SyncManager SubIndex="#1">
          ...
     </SyncManager>
</SyncManagers>
<Sdos>
     <Sdo>
          ...
     </Sdo>
     <Sdo>
     ...
     </Sdo>
</Sdos>
</Slave>
```

- *alias* 属性表示此 slave 的别名。
- *slave_position* 属性表示 slave 在该网络上的哪个位置。
- <Name>元素是 slave 的名称。
- <Emerg_size>对于所有 CoE 设备，该元素始终为 8。
- <WatchDog>元素用于设置此 slave 的看门狗。
- <DC>元素用于设置同步信息。
- <SyncManagers>元素应包含所有 syncManager 通道。

- <Sdos>元素包含我们希望通过 SDO 通道启动的默认值。

5.1.6.3.1.1.1 SyncManagers 元素

对于一个 CoE 设备，一般有四个 syncManager 通道。

- SM0：邮箱输出
- SM1：邮箱输入
- SM2：过程数据输出
- SM3：过程数据输入

```
<SyncManager SubIndex="#2">
     <Index>#x1c12</Index>
     <Name>Sync Manager 2</Name>
     <Dir>OUTPUT</Dir>
     <Watchdog>ENABLE</Watchdog>
     <PdoNum>#1</PdoNum>
     <Pdo SubIndex="#1">
          <Index>#x1600</Index>
          <Name>RxPdo 1</Name>
          <Entry SubIndex="#1">
               ...
          </Entry>
          <Entry SubIndex="#2">
               ...
          </Entry>
     </Pdo>
</SyncManager>
```

- <Index>元素是对象地址。
- <Name>是此 syncmanager 通道的名称。
- <Dir>元素是此 syncmanager 通道的方向。
- <Watchdog>用于设置此 syncmanager 通道的看门狗。
- <PdoNum>元素表示我们要设置多少个 PDO。
- <Pdo SubIndex="#1>元素包含我们要映射的对象字典条目。
  - <Index>PDO 地址。
  - <Name>PDO 名称
  - <Entry>我们要映射的对象字典。

Entry 元素用于描述我们要映射的对象字典。

```
<Entry SubIndex="#1">
     <Index>#x6041</Index>
     <SubIndex>#x0</SubIndex>
     <DataType>UINT</DataType>
     <BitLen>#16</BitLen>
     <Name>statusword</Name>
</Entry>
```

5.1.6.3.1.1.2 Sdo 元素

Sdo 元素用于设置对象字典的默认值。

```
<Sdo>
     <Index>#x6085</Index>
     <Subindex>#x0</Subindex>
     <value>#x1000</value>
     <BitLen>#32</BitLen>
     <DataType>DINT</DataType>
     <Name>Quick_stop_deceleration</Name>
</Sdo>
```

上图中的元素表示将对象字典“6085”设置为 0x1000。

5.1.6.3.2 轴元素

```
<Axle master_index='#0' slave_position="#0" AxleIndex="#0" AxleOffset="#0">
     <Mode>pp</Mode>
     <Name>x-axle</Name>
     <reg_pdo>
     ...
     </reg_pdo>
     <reg_pdo>
     ...
     </reg_pdo>
</Axle>
```

- *master_index* 属性表示该*轴*属于哪个 *master*。
- *slave_position* 属性表示该*轴*属于哪个 *slave*。
- *AxleOffset* 属性表示该*轴*是 slave 上的哪个*轴*。如上所述，CoE slave 可能有多个*轴*。如果该轴是 slave 上的第二个轴，则设置 *AxleOffset="#1"*。
- <Mode>表示该轴将在哪种模式下工作。
- <Name>是该轴的名称。
- <reg_pdo>是我们要注册的 PDO 条目。

reg_pdo 元素

```
<reg_pdo>
     <Index>#x606c</Index>
     <Subindex>#x0</Subindex>
     <Name></Name>
</reg_pdo>
```

5.1.6.4 测试

5.1.6.4.1 硬件准备

- CoE 伺服系统

  CoE 伺服系统包括 CoE 伺服和电机。在本次测试中，将使用下图所示的“2HSS458-EC”伺服系统。

- 实时边缘支持的板

  在本测试中，将使用 LS1046ARDB。

2HSS458-EC Servo System

5.1.6.4.2 软件准备

确保在配置实时边缘时选择了以下配置选项。

- igh-ethercat
- libxml2
- real-time-edge-servo

5.1.6.4.3 CoE 网络检测

- Igh 配置
  - 配置*/etc/ethercat.conf* 的 MASTER0_DEVICE 字段

    将 MASTER0_DEVICE 设置为 MAC 地址，以指示 Igh 使用哪个端口。
  - 配置*/etc/ethercat.conf* 的 DEVICE_MODULES="generic"
- 使用命令

  ```
  [root]# ethercatctl start
  ```

  启动 Igh 服务。
- 使用以下命令检查 CoE 伺服器。

  ```
  [root]# ethercat slaves
  0  0:0  PREOP  +2HSS458-EC
  ```

5.1.6.4.4 开始测试

注：“2HSS458-EC”伺服系统的*位置编码器分辨率*和*速度编码器分辨率*均为 4000。表示电机每转一圈编码器增量的比率。

- *配置文件位置*模式测试
- 按如下所示启动测试服务。

```
[root]# nservo_run -f /root/nservo_example/hss248_ec_config_pp.xml&
```

- 检查 slave 的状态是否已从“PREOP”转移到“OP”。

```
[root]# ethercat slaves
0  0:0  OP  +2HSS458-EC
```

- 检查 master 的阶段是否已经从“空闲”转移到“运行”。

```
[root]# ethercat master | grep Phase
   Phase: Operation
```

- 运行以下命令测试电机是否工作。
  - 获取轴 0 的当前模式。

    ```
    [root]# nservo_client -a 0 -c get_mode
    get_mode of the axle 0 : Profile PositionMode
    ```

  - 获取轴 0 的当前位置。

    ```
    [root]# nservo_client -a 0 -c get_position
    get_current_position of the axle 0 : 0
    ```

  - 获取轴 0 的配置文件速度。

    ```
    [root# nservo_client -a 0 -c get_profile_speed
    get_profile_speed of the axle 0 : 800000
    ```

    值 800000 表示每秒 200 转。

  - 设置轴 0 的配置文件速度。

    ```
    [root]# nservo_client -a 0 -c set_profile_speed:20000
    set_profile_speed of the axle 0 : 20000
    ```

    将配置文件速度设置为每秒 5 转。

  - 设置轴 0 的目标位置

    ```
    [root]# nservo_client -c set_position:400000
    set_position of the axle 0 : 400000
    ```

    值 400000 表示电机将转动 100 圈。

    (target_position:400000 - current_position:0) / 4000 = 100

  - 获取轴 0 的当前速度

    ```
    [root]# nservo_client -a 0 -c get_speed
    get_speed of the axle 0 : 19999
    ```

- 获取轴 0 的目标位置

```
[root]# nservo_client -a 0 -c get_target_position
get_target_position of the axle 0 : 400000
```

– 退出

```
[root]# nservo_client -c exit
```

- *配置文件速度*模式测试
  – 按如下所示启动测试服务。

```
[root]# nservo_run -f /root/nservo_example/hss248_ec_config_pv.xml&
```

– 检查 slave 的状态是否已从“PREOP”转移到“OP”。

```
[root]#  ethercat  slaves
0  0:0  OP  +2HSS458-EC
```

– 检查 master 的阶段是否已经从“空闲”转移到“运行”。

```
[root]# ethercat master | grep Phase
  Phase: Operation
```

– 运行以下命令测试电机是否工作。

- 获取轴 0 的当前模式。

```
[root]# nservo_client -a 0 -c get_mode
get_mode of the axle 0 : Profile VelocityMode
```

- 设置轴 0 的目标速度。

```
[root]# nservo_client -a 0 -c set_speed:40000
set_speed of the axle 0 : 40000
```

值 40000 表示电机将以每秒 10 转的速度转动。

- 获取轴 0 的当前速度。

```
[root]# nservo_client -a 0 -c get_speed
get_speed of the axle 0 : 32000
```

- 获取轴 0 的目标速度。

```
[root]# nservo_client -a 0 -c get_target_speed
get_target_speed of the axle 0 : 40000
```

– 退出

```
[root]# nservo_client -c exit
```

5.2 FlexCAN 和 CANOpen

以下几节介绍 FlexCAN 标准、CAN 总线的详细信息、Canopen 通信系统、如何将 FlexCAN 与实时边缘集成的详细信息以及运行 FlexCAN 应用程序。

5.2.1 简介

LS1021A 和 LS1028A 板都有 FlexCAN 模块。FlexCAN 模块是根据 CAN 2.0 B 协议规范实施 CAN 协议的通信控制器。FlexCAN 模块中实施的主要子模块包括用于存储消息缓冲区的关联存储器、接收(RX)全局掩码寄存器、接收单个掩码寄存器、接收 FIFO 过滤器和接收 FIFO ID 过滤器。一般功能框图如下图所示。这些子模块的功能将在后续章节中介绍。

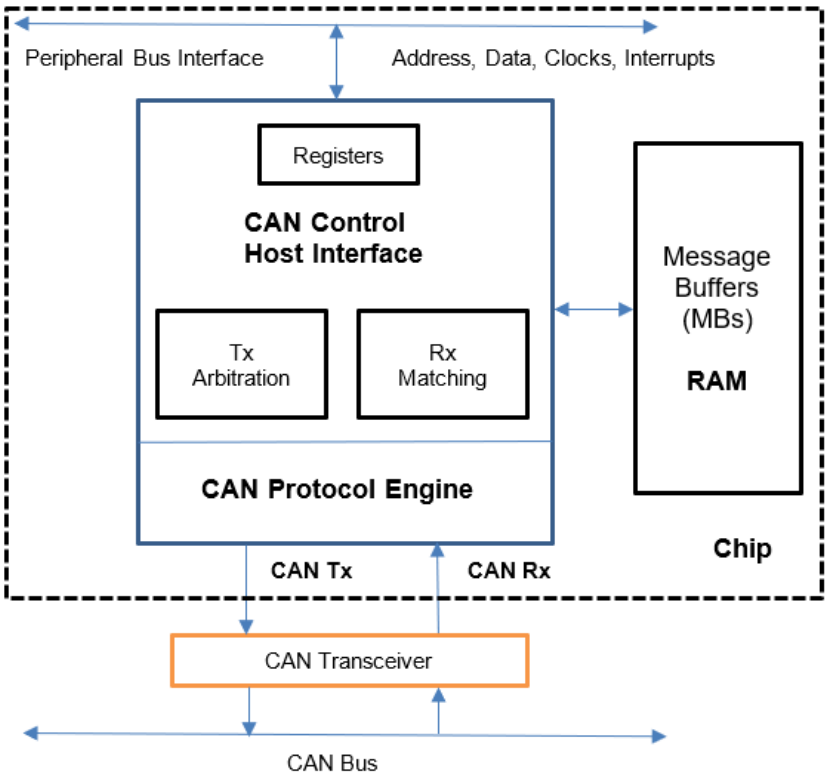


图 42.FlexCAN 功能框图

5.2.1.1 CAN 总线

CAN（控制器局域网）是一种串行总线系统。CAN 总线是一种可靠的汽车总线标准，旨在允许微控制器和设备在没有主机的应用中相互通信。博世发布了多个版本的 CAN 规范，最新的是 1991 年发布的 CAN 2.0。本规范有两部分；A 部分用于具有 11 位标识符的标准格式，B 部分用于具有 29 位标识符的扩展格式。使用 11 位标识符的 CAN 设备通常称为 CAN 2.0A，而使用 29 位标识符的 CAN 设备通常称为 CAN 2.0B。

CAN 是一种多 master 串行总线标准，用于连接电子控制单元[ECU]，也称为节点。CAN 网络上需要两个或更多节点进行通信。节点的复杂性可以从简单的 I/O 设备到具有 CAN 接口和复杂软件的嵌入式计算机。该节点也可以是网关，允许标准计算机通过 USB 或以太网端口与 CAN 网络上的设备进行通信。所有节点通过双线总线相互连接。电线是具有 120 Ω（标称）特性阻抗的双绞线。

高速 CAN 信令传递在发送显性(0)时将 CAN 高线驱动至 5 V，将 CAN 低线驱动至 0 V，而在发送隐性(1)时不驱动任一条线。显性差分电压为标称 2 V。端接电阻器被动地将两条线返回到 0 V 的标称差分电压。显性共模电压必须在 1.5 到 3.5 V 的共模电压范围内，隐性共模电压必须在+/-12 的共模电压范围内。

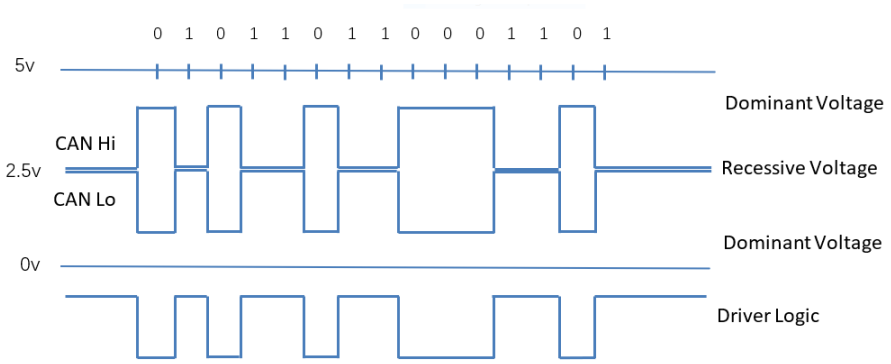


图 43.高速 CAN 信令

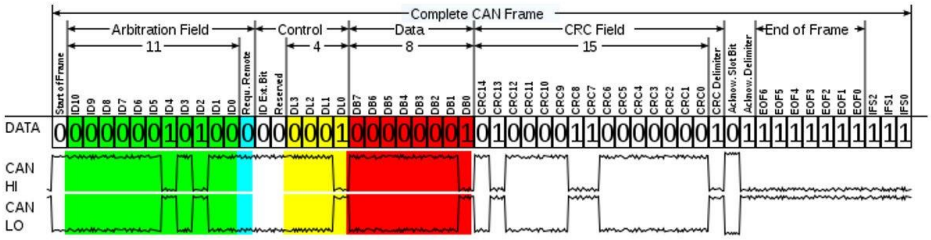


图 44.基本帧格式

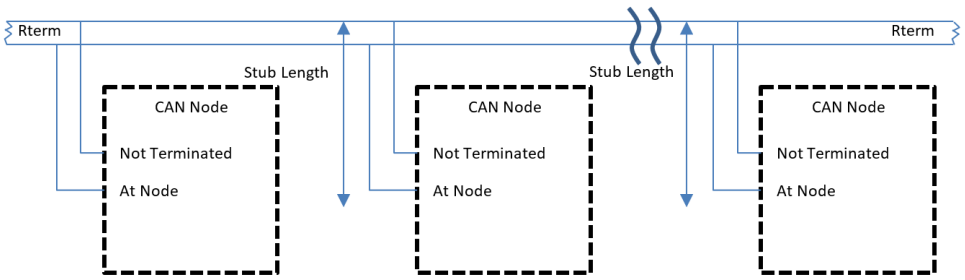


图 45.高速 CAN 网络

5.2.1.2 CANopen

CANopen 是一个基于 CAN 的通信系统。它包括更高层的协议和配置文件规范。CANopen 已开发为具有高度灵活配置能力的标准化嵌入式网络。如今，它用于各种应用领域，例如医疗设备、越野汽车、海事电子、铁路应用或楼宇自动化。

CANopen 提供了多个通信对象，使设备设计人员能够在设备中实施所需的网络特性。借助这些通信对象，设备设计人员可以提供能够传送过程数据、指示设备内部错误状态或影响以及控制网络行为的设备。由于 CANopen 定义了内部设备结构，系统设计人员确切地知道如何访问 CANopen 设备以及如何调整预期的设备行为。

- **CANopen 底层**

  CANopen 基于符合 ISO 11898-1 的数据链路层。CANopen 位时序在 CiA 301 中指定，允许将数据速率从 10 kbit/s 调整到 1000 kbit/s。尽管所有指定的 CAN-ID 寻址模式都基于 11 位 CAN-ID，CANopen 也支持 29 位 CAN-ID。但是，CANopen 并不排除其他物理层选项。

- **内部设备架构**

  CANopen 设备包含三个逻辑部分。CANopen 协议栈通过 CAN 网络处理通信。应用程序软件提供内部控制功能。CANopen 对象字典对接协议以及应用程序软件。它包含所有使用的数据类型的索引并存储所有通信和应用程序参数。CANopen 对象字典对于 CANopen 设备配置和诊断最为重要。

- **CANopen 协议**
  - SDO 协议
  - PDO 协议
  - NMT 协议
  - 特殊功能协议
  - 错误控制协议

下图显示了 CANopen 架构。

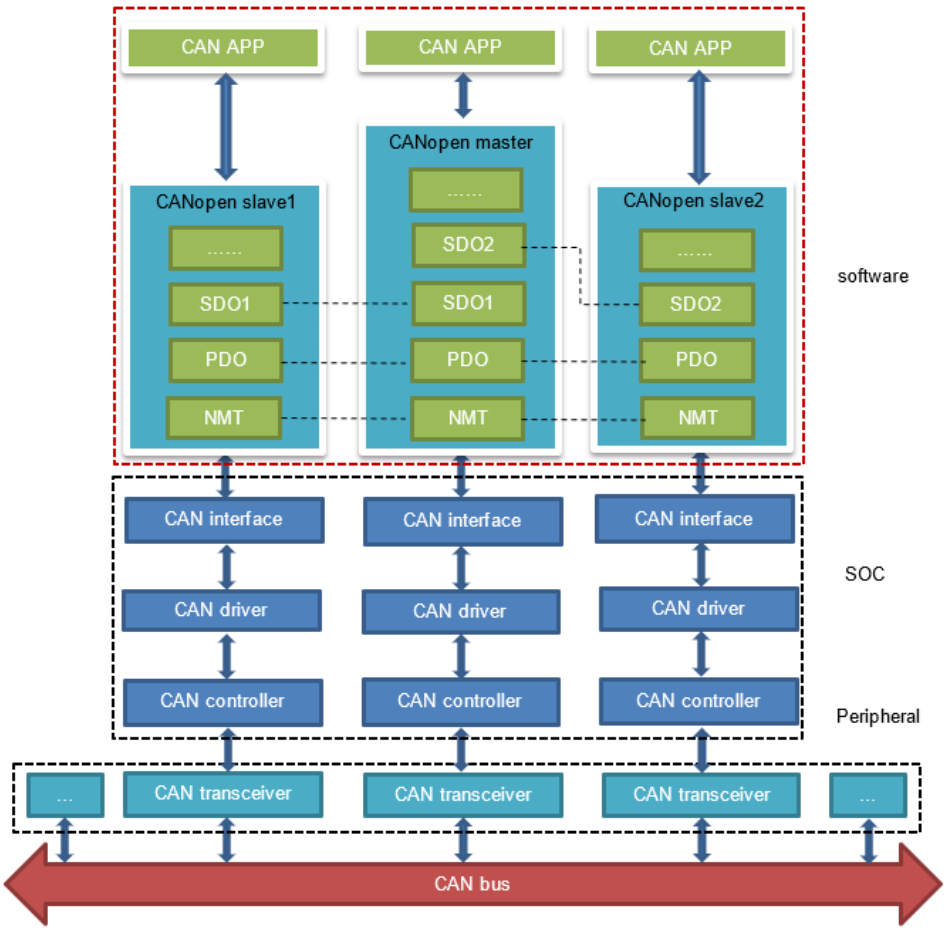


图 46.CANopen 架构

5.2.2 实时边缘中的 FlexCAN 集成

LS1021A 有四个 CAN 控制器。两个 CAN 控制器（CAN3 和 CAN4）用于相互通信。CAN4 分配给 core0，它运行 Linux 和 CANOpen 作为 master 节点，而 CAN3 分配给 core1，它运行裸机和 CANOpen 作为 slave 节点。LS1028A 有两个 CAN 控制器 CAN1 和 CAN2，它们都用于 LS1028ARDB 板。

5.2.2.1 LS1021AIOT CAN 资源分配

本节介绍将 CAN4 分配给 Linux 和将 CAN3 分配给裸机内核的步骤，以及如何进行更改或配置。这些示例假设 CAN1 和 CAN2 未使能，并且其他 IP 使用了 CAN1 和 CAN2 的引脚。

1. **将 CAN4 分配给 Linux**

   在 Linux 中，端口通过 DTS 文件进行分配。DTS 文件路径为 `industry-linux/arch/arm/boot/dts/ls1021a- iot.dts`。CAN 端口相关内容如下：

```
/* CAN3 port */
    &can2
        {
           status = " disabled ";
        };
/* CAN4 port */
    &can3
        {
            status = "okay";
        };
```

2. **将 CAN3 分配给裸机**

   在裸机中，端口通过 `flexcan.c` 文件进行分配。flexcan.c 路径是 `industry-uboot/drivers/flexcan/ flexcan.c`。在此文件中，用户应定义以下变量：

   a. `struct can_bittiming_t flexcan3_bittiming = CAN_BITTIM_INIT(CAN_500K);`

   注意
   设置 CAN 端口的位时序和波特率(500K)。

   b. struct can_ctrlmode_t flexcan3_ctrlmode

```
struct can_ctrlmode_t flexcan3_ctrlmode =
{
    .loopmode = 0, /* Indicates whether the loop mode is enabled*/
    .listenonly = 0, /* Indicates whether the only-listen mode is enabled*/
    .samples = 0,
    .err_report = 1,
};
```

   c. struct can_init_t flexcan3

```
struct can_init_t flexcan3 =
{
    .canx = CAN3, /* Specify CAN port */
    .bt = &flexcan3_bittiming,
    .ctrlmode = &flexcan3_ctrlmode,
    .reg_ctrl_default = 0,
    .reg_esr = 0
};
```

   d. 可选参数

- **CAN 端口**

```
#define CAN3 ((struct can_module *)CAN3_BASE)
#define CAN4 ((struct can_module *)CAN4_BASE)
```

- **波特率**

```
#define CAN_1000K 10
#define CAN_500K 20
#define CAN_250K 40
#define CAN_200K 50
#define CAN_125K 80
#define CAN_100K 100
#define CAN_50K 200
#define CAN_20K 500
#define CAN_10K 1000
#define CAN_5K 2000
```

5.2.2.2 CAN 示例代码的功能介绍

CAN 示例代码支持 CANopen 协议。主要实施三部分功能：网络管理功能（NMT 协议）、业务数据传输功能（SDO 协议）和流程数据传输功能（PDO 协议）。NMT 协议可以管理和监测 slave 节点，包括心跳消息。SDO 协议可以发送单个数据或块数据。PDO 协议可以发送需要实时的过程数据。

CAN 示例调用 CANopen 接口，如下表中所述：

表 60.CAN 网络 API 及其说明

API 名称（类型）	说明
UNS8 canReceive_driver (CAN_HANDLE fd0, Message * m)	SocketCAN 接收 CAN 消息 ● fd0——SocketCAN 句柄 ● m——接收缓冲区
UNS8 canSend_driver (CAN_HANDLE fd0, Message const * m)	SocketCAN 发送 CAN 消息 ● fd0——SocketCAN 句柄 ● m——要发送的 CAN 消息
void setNodeId(CO_Data* d, UNS8 nodeId)	设置此节点 ID 值。 ● d——对象字典 ● nodeId——ID 值（最多 127）
UNS8 setState(CO_Data* d, e_nodeState newState)	设置节点状态 ● d——对象字典 ● newState——需要设置的状态 正常返回 0，错误返回 > 0
void canDispatch(CO_Data* d, Message *m)	CANopen 处理 CAN 接收的数据帧。

表格接下页……

表 60.CAN 网络 API 及其说明（续）

API 名称（类型）	说明
	• d——对象字典 • m——接收到的 CAN 消息
void timerForCan(void)	CANopen 虚拟时钟计数器。
UNS8 sendPDOrequest (CO_Data * d, UNS16 RPDOIndex)	master 节点请求 slave 节点反馈指定的数据。 • d——对象字典 • RPDOIndex——指定数据的索引值
UNS8 readNetworkDictCallback (CO_Data* d, UNS8 nodeId, UNS16 index, UNS8 subIndex, UNS8 dataType, SDOCallback_t Callback, UNS8 useBlockMode)	master 节点从 slave 节点获取指定的数据。 • d——对象字典 • nodeId——slave 节点的 ID 值 • index——指定数据的索引值 • subIndex——指定数据的子索引值 • dataType——指定数据的数据类型 • Callback——回调函数 • useBlockMode——指定是否是块传输
UNS8 writeNetworkDictCallBack (CO_Data* d, UNS8 nodeId, UNS16 index, UNS8 subIndex, UNS32 count, UNS8 dataType, void *data, SDOCallback_t Callback, UNS8 useBlockMode)	master 节点将指定的数据设置给 slave 节点。 • d——对象字典 • nodeId——slave 节点的 ID 值 • index——指定数据的索引值 • subIndex——指定数据的子索引值 • count——指定数据的长度 • dataType——指定数据的数据类型 • Callback——回调函数 • useBlockMode——指定是否是块传输

5.2.3 运行 CAN 应用程序

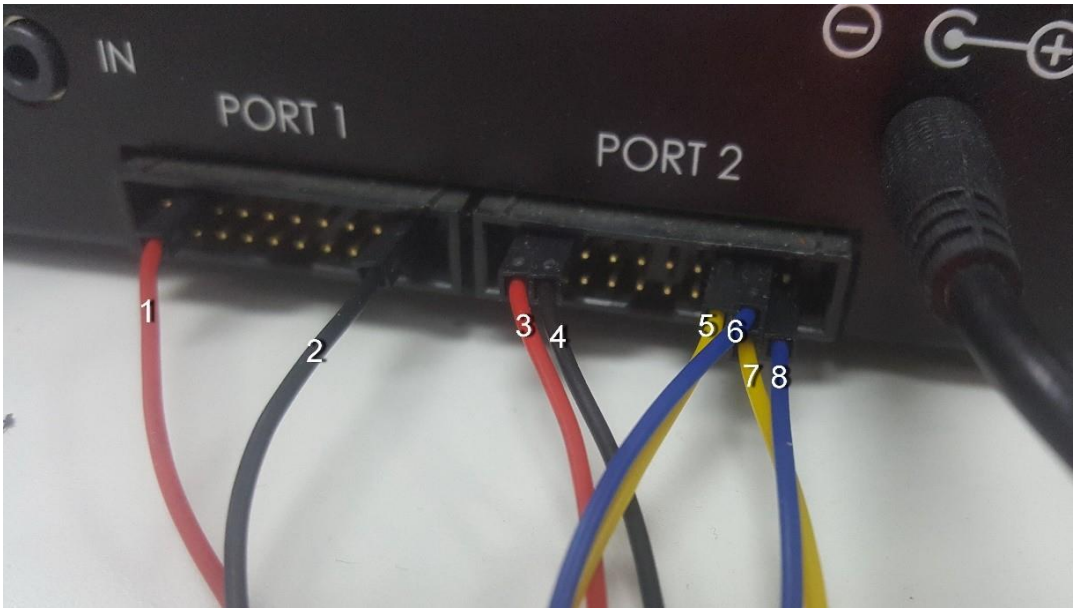
以下几节介绍运行 CAN 应用程序的硬件和软件准备步骤。对 LS1021A-IoT 和 LS1028ARDB 分别进行硬件准备说明，但为 master 节点编译 CANopen-app 二进制文件、运行 CANopen 应用程序和运行 SocketCAN 命令几节适用于 LS1021A-IoT 和 LS1028A 平台。

5.2.3.1 LS1021-IoT 的硬件准备

对于 LS1021-IoT，实施 FlexCAN 演示所需的硬件列表如下：

- LS1021A-IoT 板
- 两个 CAN 硬件接口（例如，用于 LS1021A-IoT 的 CAN3 和 CAN4）

- 两个 CAN 收发器（例如：TJA1050）



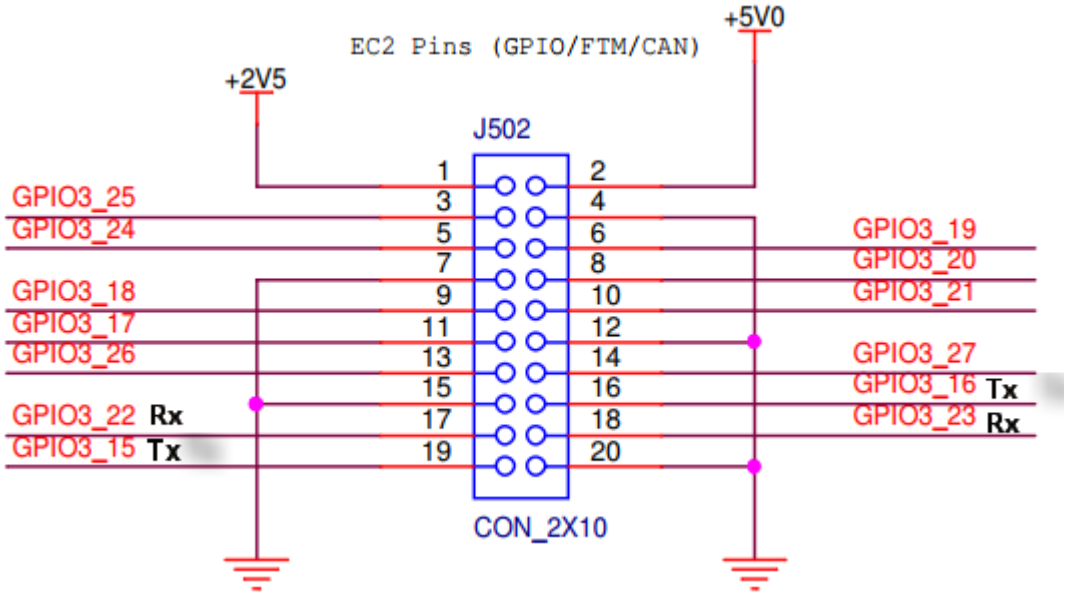


图 47.FlexCan 演示的硬件图

注意

— 线 1 和线 3 为 5.0 V。
— 线 2 和线 4 为 GND。
— 线 5 是 CAN3 TX。
— 线 6 是 CAN3 RX。
— 线 7 是 CAN4 RX。
— 线 8 是 CAN4 TX。

5.2.3.2 LS1028ARDB 的硬件准备

对于 LS1028ARDB，需要以下硬件：

- LS1028ARDB 板
- 两条连接 CAN1 和 CAN 的线缆。

硬件连接图如下图所示

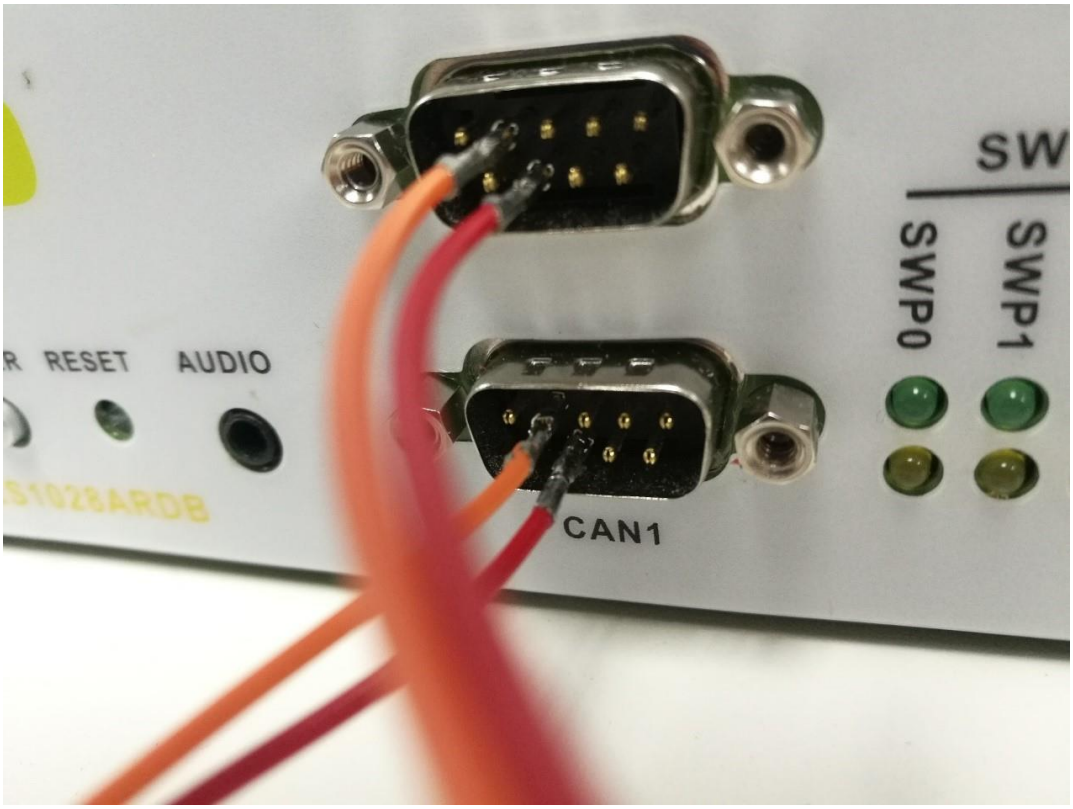


图 48.使用 LS1028ARDB 的 CAN 物理连接

5.2.3.3 为 master 节点编译 CANopen-app 二进制文件

本节介绍为 LS1021A 和 LS1028A 平台编译 master 节点的 CANopen-app 二进制文件的步骤。

CANopen 应用程序的名称是 **CANopen-app**。执行下列步骤，将 Canopen-app 作为 linux 命令编译到 `target/usr/bin` 目录。

1. 在用户主机环境中配置跨工具链。
2. 使用以下命令：

```
$ cd yocto-real-time-edge/meta-real-time-edge
# open file:./conf/distro/include/qoriq-baremetalenv.inc
# replace "ls1021aiot_baremetal_defconfig" with "ls1021aiot_baremetal_can_defconfig"
$ bitbake nxp-real-time-edge-baremetall
```

3. 生成的实时边缘镜像文件位于 `tmp/deploy/images/ls1021aiot/`目录中。
4. 下载镜像 `nxp-image-real-time-edge-ls1021aiot.wic.bz2` 并解压，然后将其烧写到 SD 卡：

在 U-Boot 模式下，首先运行 `tftp` 命令将 `nxp-image-real-time-edge-ls1021aiot.wic` 下载到缓冲区。然后，运行 `mmc` 命令将 `nxp-image-real-time-edge-ls1021aiot.wic` 下载到 SD 卡。

注意

- 仅当 `canfestival` 选项设置为 Y 时，才会显示以下选项。
- Linux 使用 SocketCAN 接口，因此 `driver` 选项选择套接字。
- 可以在 CANopen 的 `config.h` 文件中配置以下 `additional configure options`：

  参数说明：

  - --SDO_MAX_LENGTH_TRANSFER：设置 SDO 协议的缓冲区大小。
  - -- SDO_BLOCK_SIZE：设置 SDO 块传输协议可以发送的最大帧数。
  - --SDO_MAX_SIMULTANEOUS_TRANSFERS：设置 SDO 模块的数量。
- 如果 `install examples` 选项设置为 Y，则将二进制应用程序安装到文件系统。

5.2.3.4 运行 CANopen 应用程序

本节介绍运行 CANopen-app 应用程序的步骤。只有 LS1021A 平台支持此应用程序。

1. 首先，启动 LS1021A-IoT 板。
2. 等待裸机内核输出以下信息：

```
Note: the CANopen protocol starts to run!
=>
```

3. 然后，在 Linux 提示符下的任意目录中运行 `CANopen-app` 命令。执行此命令时，首先运行测试代码。
4. 测试代码完成后，用户可以执行所需的指令。命令 `CANopen-app` 执行过程步骤如下：
   a. 首先指示 CAN 接口是否已成功打开。所有命令都进行动态注册。然后，指示该命令是否已成功注册。
      - **命令注册日志**

```
Command Registration Log:
[root@]# CANopen-app
[    80.899975] IPv6: ADDRCONF(NETDEV_CHANGE): can0: linkbecomesready
Note: open the CAN interfacesuccessfully!
"can_quit" command: register OK!
"setState" command: register OK!
"showPdo" command: register OK!
"requestPdo" command: register OK!
"sdo" command: register OK!
"" command: register OK!
"test_startM" command: register OK!
"test_sdoSingle" command: register OK!
"test_sdoSingleW" command: register OK!
"test_sdoBlock" command: register OK!
"test_showPdoCyc" command: register OK!
"test showpdoreq" command: register OK!
"test_requestpdo" command: register OK!
```

   b. 共有九个测试代码，测试 1 到 9。测试代码详细信息显示在测试日志中。

- **测试代码日志**“---test---”表示测试代码开始。
- 首先，说明 SDO 和 PDO 协议的执行权。
- **测试 1 至 4** 是 SDO 协议测试代码。CANopen master 节点启动后，自动进入初始化和预工作模式。
- **测试 5** 是 master 节点进入工作模式并启动所有 slave 节点的测试代码。
- **测试 6 至 9** 是 PDO 协议测试代码。

```
Test Code Log:
------------------------ test ----------------------------
Note: Test code start execute...
      SDO protocol is valid in preoperation mode, but PDO protocol is invalid!
      SDO and PDO protocol are both valid in operation mode!
      Console is invalid when testing!
------------------------------------------------------------
Note: test1--Read slave node single data by SDO.
Note: master node initialization is complete!
Note: master node entry into the preOperation mode!
Note: Alarm timer is running!
Note: slave node "0x02" entry into "Initialisation" state!
------------------------------------------------------------
Note: test2--Write 0x2CD5 to slave node by SDO.
Note: Master write a data to 0x02 node successfully.
------------------------------------------------------------
Note: test3--Read slave node single data by SDO again.
Note: reveived data is 0x2CD5
------------------------------------------------------------
Note: test4--Read slave node block data by SDO.
---------------- text ------------------
Note: reveived string ==>
CANopen is a CAN-based communication system.
It comprises higher-layer protocols and profile specifications.
CANopen has been developed as a standardized embedded network with highly flexible
configuration capabilities.
It was designed originally for motion-oriented machine control systems, such as
handling systems.
Today it is used in various application fields, such as medical equipment, off-road
vehicles, maritime electronics, railway applications, or building automation.


----------------------------------------
------------------------------------------------------------
Note: test5--Master node entry operation mode, and start slave nodes!
Note: master node entry into the operation mode,and start all slave nodes!
------------------------------------------------------------
Note: test6--Master node show requested PDO data.Note: Rpdo4 data is "     "
------------------------------------------------------------
Note: test7--Master node request PDO data.
------------------------------------------------------------
Note: test8--Master node show requested PDO data.
Note: Rpdo4 data is "require"
Note: slave node "0x02" entry into "Operational" state!
------------------------------------------------------------
Note: test9--Master node show received cycle PDO data.
Note: Rpdo2 data is "  cycle"
------------------------------------------------------------
```

注意

测试 1 到 9 不是命令。

c. 执行测试代码后，它会自动打印命令列表。编号 00 至 06 为普通命令。执行这些不带参数的指令后，会显示指令用法。编号 08 至 14 为测试命令。除编号 10 之外的所有测试命令都没有参数。编号 10 的参数是一个 16 位整数。

- 现在用户可以执行命令列表中的任何命令。

命令列表

```
命令列表：
-------------------------------------------------------------------------
编号     |     命令          |       简介
-------------------------------------------------------------------------
00       | ctrl_quit        |  控制台线程退出!
-------------------------------------------------------------------------
01       | help             |  命令列表
-------------------------------------------------------------------------
02       | can_quit         |  退出 CANopen 线程
-------------------------------------------------------------------------
03       | setState         |  设置 CANopen 节点状态
-------------------------------------------------------------------------
04       | showPdo          |  显示 RPDO 的数据
-------------------------------------------------------------------------
05       | requestPdo       |  请求 RPDO 的数据
-------------------------------------------------------------------------
06       | sdo|  SDO 协议读取/写入一个条目
-------------------------------------------------------------------------
07       |                  |
-------------------------------------------------------------------------
08       | test_startM      |  测试 -- 启动 master
-------------------------------------------------------------------------
09       | test_sdoSingle   |  测试 -- 读取 slave 节点单个数据
-------------------------------------------------------------------------
10       | test_sdoSingleW  |  测试 -- 写入 slave 节点单个数据
-------------------------------------------------------------------------
11       | test_sdoBlock    |  测试 -- 读取 slave 节点块数据
-------------------------------------------------------------------------
12       | test_showPdoCyc  |  测试 -- 显示循环 PDO 数据
-------------------------------------------------------------------------
13       | test_showpdoreq  |  测试 -- 显示请求的 PDO 数据
-------------------------------------------------------------------------
14       | test_requestpdo  |  测试 -- 请求的 PDO 数据
-------------------------------------------------------------------------
注：用户可以通过控制台发送命令!
注：测试代码执行完成!
```

示例：以下示例显示了在不带任何参数运行 `sdo` 命令后的使用日志。

```
SDO Command:
sdo
usage: sdo -type index subindex nodeid data
       type = "r"(read), "w"(write),"b"(block)
       index = 0~0xFFFF,unsignedshort
       subindex = 0~0xFF,unsigned char
       nodeid = 1~127,unsigned char
       data = 0 ~ 0xFFFFFFFF
```

5.2.3.5 运行 SocketCAN 命令

本节介绍运行可在任一板（LS1021A-IoT 或 LS1028ARDB）上执行的 SocketCAN 命令的步骤。这些命令在 Linux 上执行。标准的 SocketCAN 命令如下：

1. 打开 can0 端口。

```
$ ip link set can0 up
```

2. 关闭 can0 端口。

```
$ ip link set can0 down
```

3. 将 can0 端口的波特率设置为 500K

```
$ ip link set can0 type can bitrate 500000
```

4. 将 can0 端口设置为环回模式。

```
$ ip link set can0 type can loopback on
```

5. 通过 can0 发送消息。002（十六进制）是节点 ID，此值必须是 3 个字符。2288DD（十六进制）是消息，最多可以取 8 个字节的值。

```
$ cansend can0 002#2288DD
```

6. 监测 can0 端口，等待接收数据。

```
$ candump can0
```

7. 请参见 can0 端口详细信息。

```
$ ip -details link show can0
```

注意

第三条和第四条命令在 can0 端口状态关闭时有效。

5.2.3.6 测试 CAN 总线

下面是在 LS1028ARDB 上测试 CAN 总线的示例代码。

```
[root]# ip link set can0 down
[root]# ip link set can1 down
[root]# ip link set can0 type can loopback off
[root]# ip link set can1 type can loopback off
[root]# ip link set can0 type can bitrate 500000
[root]# ip link set can1 type can bitrate 500000
[root]# ip link set can0 up
[root]# ip link set can1 up
[root]# candump can0 &
[root]# candump can1 &
[root]# cansend can0 001#224466
  can0 001     [3]  22 44 66
[root]# can1 001       [3]  22 44 66
[root]# cansend can1 001#224466
can0 001       [3]  22 44 66
can1 001       [3]  22 44 66
[root]# cansend can1 001#113355
```

```
  can0 001     [3]  11 33 55
  can1 001     [3]  11 33 55
[root]# cansend can0 000#224466
  can0 000     [3]  22 44 66
```

5.3 OPC UA

OPC（最初称为“过程控制的 OLE”，现称为“开放平台通信”）是多种规范的集合，其中最常见的是 OPC 数据访问(OPC DA)。

OPC 统一架构(OPC UA)由 OPC 基金会于 2010 年发布，作为 OPC 经典规范的向后不兼容标准，名称为 IEC 62541。

OPC UA 不再使用 OPC 经典规范的 COM/DCOM（微软专有技术）通信模型，转而使用基于 TCP/IP 的通信栈（异步请求/响应），分层如下：

- 原始连接
- 安全通道
- 会话

5.3.1 OPC 简介

OPC UA 定义：

- 通信传输协议（可以通过 HTTP、SOAP/XML 或直接通过 TCP 进行）。
- 在 OPC 服务器上运行的一组 37 个“服务”，客户端通过异步请求/响应 RPC 机制调用这些“服务”。
- 使用面向对象的概念和复杂关系创建数据信息模型的基础。

OPC 的主要目标是以最简单的方式从设备中提取数据。

*信息模型*为服务器提供了一种方法，不仅可以提供数据，而且可以通过简单明了且直观的方式提供数据。

注意

本文档中进一步提到“OPC”是指 OPC UA。本文档不讨论 OPC 经典规范。

以下是将支持 OPC 的设备嵌入到项目中的典型场景：

- 使用通用的 GUI 客户端（例如 Unified Automation 的 UaExpert 或本章介绍的 FreeOpcUa）手动调查（“浏览”）服务器的地址空间以查找用户需要的数据。
- 使用“引用”和“属性”，了解它的格式，以及转换数据可能需要的步骤。
- 让自定义 OPC 客户端（集成到应用程序中）直接订阅包含所需数据的节点数据更改。

在典型用例中：

- OPC 服务器在信息源附近运行（在工业环境中，这表示靠近物理过程——例如，在工厂车间的 PLC 上）。
- 客户端在运行时使用数据（例如，登录数据库，或将其输入另一个工业流程）。

支持 OPC 的应用程序可以由以下部分组成：工业设备可以运行 OPC 客户端，并将收集的数据馈送到另一个物理过程，同时还可以通过运行 OPC 服务器公开该过程。

5.3.2 节点模型

OPC 服务器中的数据以*节点*为结构。OPC 服务器向其客户端公开的所有节点的集合称为*地址空间*。一些节点具有预定义的含义，而其他节点具有特定于该特定 OPC 服务器的*信息模型*的含义。

每个节点都有以下***属性***：

- ***ID***（唯一）
- ***类别***（它是什么类型的节点）
- ***浏览名称***（机器使用的字符串）
- ***显示名称***（人类使用的字符串）

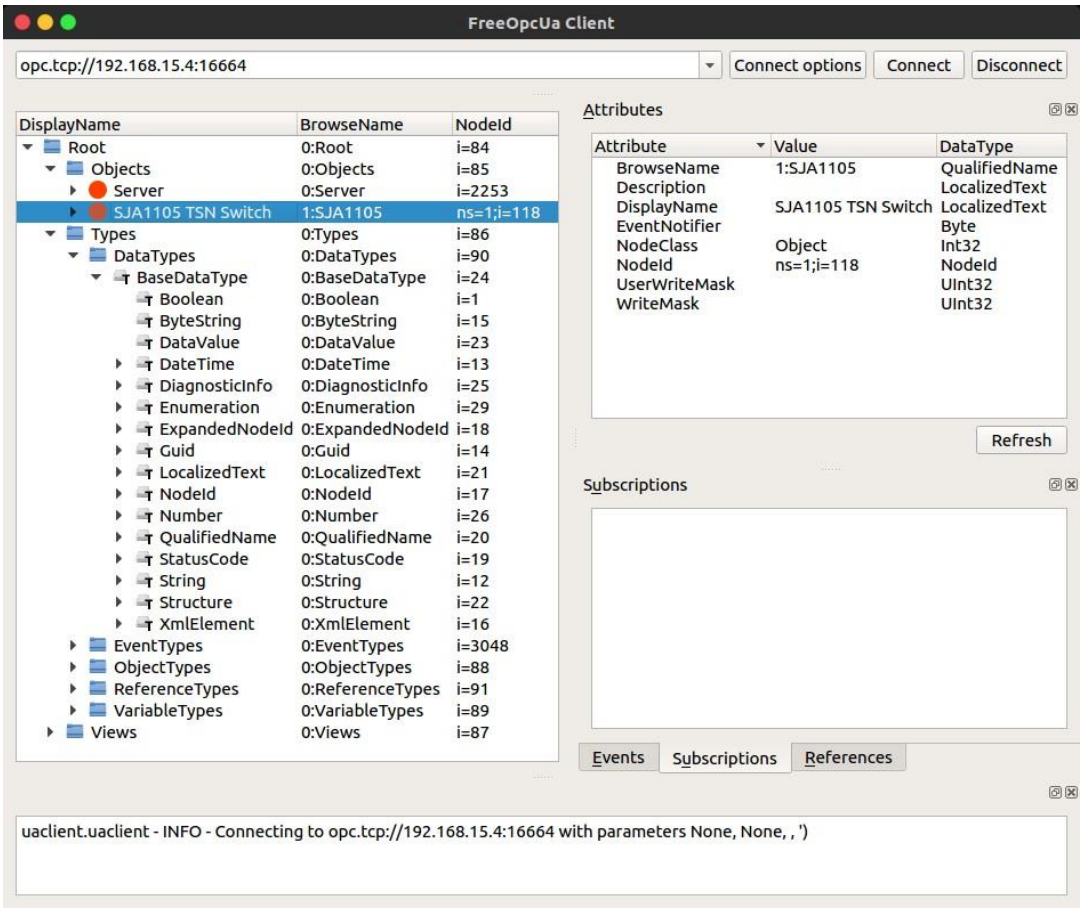


图 49.OPC UA 地址空间

图中左侧显示的是 OPC 服务器的*地址空间*（服务器向客户端提供的信息的集合），位于 `opc.tcp://192.168.15.4:16664`。

选择的是节点 ID 为 `ns=1;i=118`、浏览名称=`1:SJA1105` 和节点类别为 `Object` 的节点。

所选节点的完整路径为 `0:Root,0:Objects,1:SJA1105`。

5.3.3 节点命名空间

*命名空间*是分离服务器同一地址空间中存在的多个信息模型的方法。

- 没有 `ns=`前缀作为节点 ID 一部分的节点具有隐式 `ns=0`；前缀（是命名空间 `zero` 的一部分）。

- *命名空间 * 0* 中的节点具有由 OPC UA 标准预定义的节点 ID。例如，包含自描述信息（功能、诊断和供应商信息）的 `0:Server` 对象具有预定义的节点 ID `ns=0;i=2253;`。

不更改*命名空间 * 0* 中公开的任何节点公认是一种较好的做法。

5.3.4 节点类别

OPC 节点有一个继承模型，基于它们的 *节点类别*。

该标准定义了八个基本节点类别：

- 对象
- 变量
- 方法
- 视角
- 对象类型
- 变量类型
- 引用类型
- 数据类型

所有节点都具有相同的基本属性（继承自节点对象），以及取决于它们的 *节点类别*的附加属性。

5.3.5 节点图和引用

看起来节点只是以简单的父子关系进行分层链接。然而，实际上节点通过对其他节点的 *引用*，在一个复杂的有向图中进行链接。

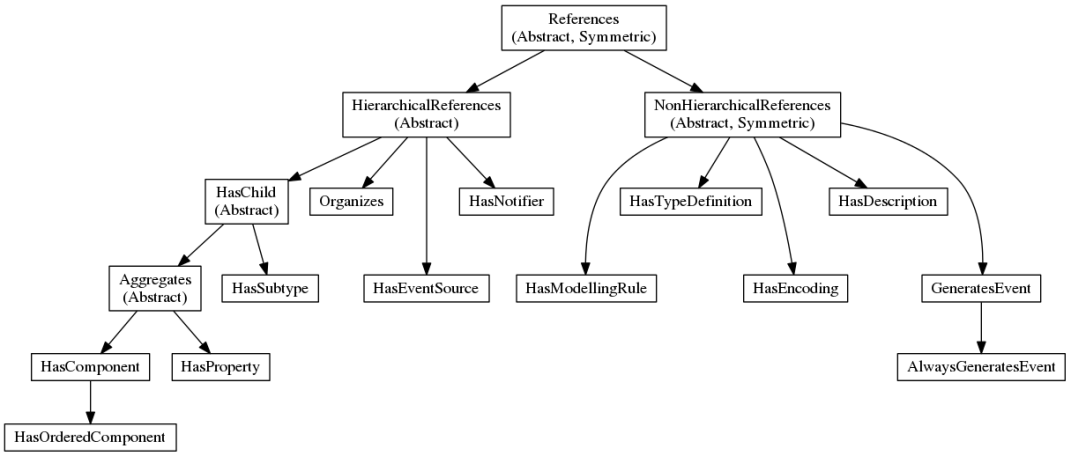


图 50. OPC UA 规范的第 3 部分中定义了标准引用类型的层次结构（图片取自 www.open62541.org）

在 OPC 中，甚至引用类型也是节点，因此是分层结构，如上图所示。

所有 OPC 引用类型的定义可在 `0:Root,0:Types,0:ReferenceTypes` 路径下找到。

可以通过创建自定义引用类型节点来丰富 OPC 引用的语义。

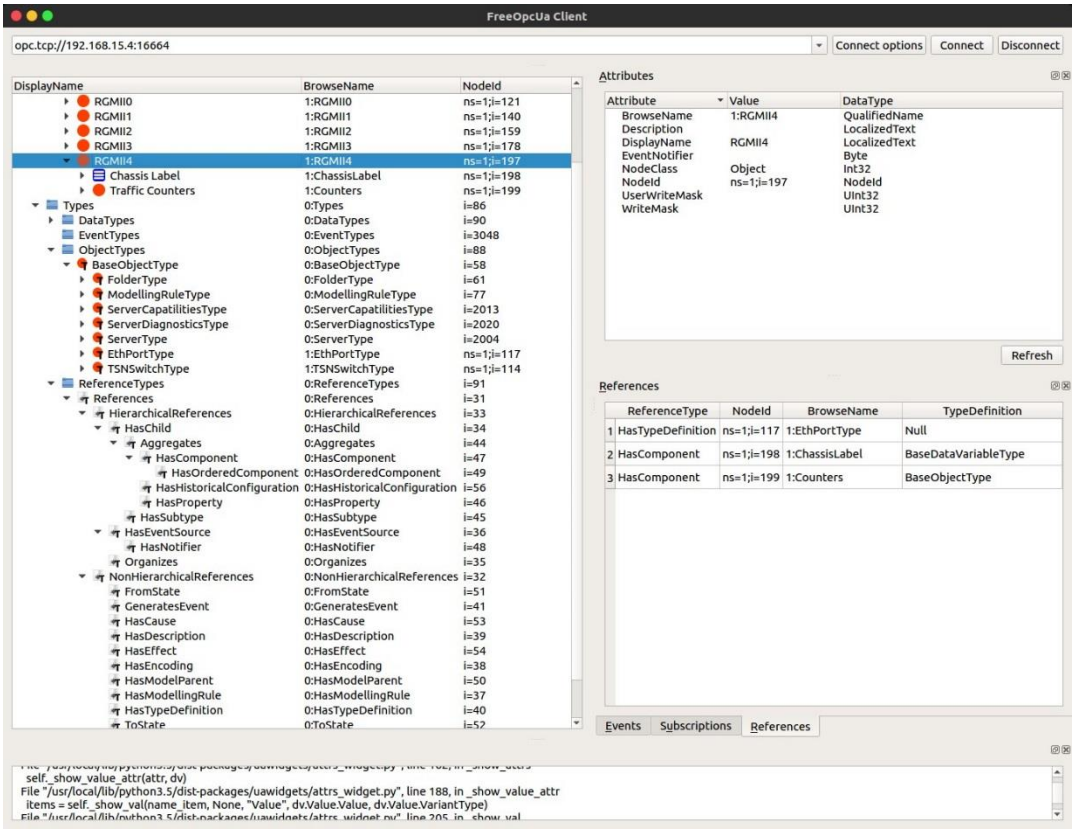


图 51.　　FreeOpcUa 客户端的“属性”和“引用”视图填充了 RGMII4 节点的详细信息

在地址空间中选择的是节点 `ns=1;i=197`。从概念上讲，这代表了 SJA1105 TSN 交换机的五个以太网端口之一。

它的节点类别是对象，但它有一个对节点 ID`ns=1;i=117` 的 `HasTypeDefinition` 类型的引用，即 `1:EthPortType`。因此，`1:RGMII4` 节点属于自定义对象类型 `EthPortType`。

5.3.6　Open62541

实时边缘集成了 Open62541 软件协议栈(https://open62541.org/)。这支持 OPC UA 应用程序的服务器端和客户端 API。此处仅显示 open62541 的服务器端功能。

Open62541 作为基于 C 的动态库(libopen62541.so)分发。服务在 pthread 上运行，应用程序代码在事件循环中运行。

在实时边缘文件./recipes-nxp/packagegroups/packagegroup-real-time-edge-industrial.bb"中使能 open62541：

```
libopen62541 \
```

为了安装 Open62541 示例应用程序，发行版配置中已包含文件“meta-real-time-edge/conf/distro/include/libopen62541.inc”。

目标镜像中包含以下 Open62541 示例应用程序：

- open62541_access_control_client
- open62541_access_control_server
- open62541_client
- open62541_client_async

- open62541_client_connect
- open62541_client_connectivitycheck_loop
- open62541_client_connect_loop
- open62541_client_subscription_loop
- open62541_custom_datatype_client
- open62541_custom_datatype_server
- open62541_server_ctt
- open62541_server_inheritance
- open62541_server_instantiation
- open62541_server_loglevel
- open62541_server_mainloop
- open62541_server_nodeset
- open62541_server_repeated_job
- open62541_tutorial_client_events
- open62541_tutorial_client_firststeps
- open62541_tutorial_datatypes
- open62541_tutorial_server_datasource
- open62541_tutorial_server_firststeps
- open62541_tutorial_server_method
- open62541_tutorial_server_monitoreditems
- open62541_tutorial_server_object
- open62541_tutorial_server_variable
- open62541_tutorial_server_variabletype

5.4 NETCONF/YANG

本章概述了 NETCONF 协议和 Yang（NETCONF 的数据建模语言）。它介绍了 NETCONF 的应用程序、安装和配置步骤、操作示例、Web UI 演示和故障排除方面。它还介绍了如何在此实时边缘软件中使能 NETCONF 功能。

5.4.1 概述

NETCONF 协议定义了一种用于设备管理和配置检索和修改的机制。它使用远程程序调用(RPC)范式和公开设备（服务器）功能的系统，使客户端能够适应任何网络设备的特定功能。NETCONF 进一步区分了状态数据（只读）和配置数据（可以修改）。任何 NETCONF 通信都发生在四个层上，如下表所示。XML 用作编码格式。

表 61.NETCONF 层

层	用途	示例
1	内容	配置数据、通知数据
2	运算	<edit-config>

表格接下页……

表 61.NETCONF 层（续）

层	用途	示例
3	消息	<rpc>, <rpc-reply>, <notification>
4	安全	传输 SSH、TLS

YANG 是一种基于标准的、可扩展的分层数据建模语言，用于对 NETCONF 操作、远程程序调用(RPC)和服务器事件通知使用的配置和状态数据进行建模。设备配置数据存储为 XML 文档格式。文档中的特定节点以及允许的值由模型定义，该模型通常为 YANG 格式，也可能使用基于 XML 的语法转换为 YIN 格式。IETF 直接创建了许多这样的模型，以进一步支持普通网络设备的 NETCONF 接口的标准化和统一。例如，IETF 系统模型(rfc7317)或 IETF 接口模型(rfc7223)定义的其网络接口配置中介绍了标准计算机的一般系统设置。但是，通常每个系统都有一些专属于自己的特定部分。在这种情况下，定义了一些机制来支持扩展，同时保持对标准化内核的支持。此外，由于整个机制是采用自由方式进行设计，因此配置不必严格关注网络。甚至可以对 NETCONF 定义的 RPC 之外的 RPC 进行表征，从而允许客户端向服务器请求显式操作。

YANG 模块通过其数据以及对该数据的分层组织和约束来定义数据模型。模块可以是完整、独立的实体，或者它可以引用其他模块和子模块中的定义，以及增加额外节点的其他数据模型。该模块规定了数据在 XML 中的表示方式。

YANG 模块不仅定义了数据的语法，还定义了数据的语义。它明确定义了数据之间的关系和数据约束。这使用户能够创建符合约束要求的语法正确的配置数据，并使用户能够在上传数据并将其提交到设备上之前针对模型验证数据。

有关 NETCONF 的信息，请参见 RFC 6241，NETCONF 配置协议。

有关 YANG 的信息，请参见 RFC 6020，YANG——网络配置协议(NETCONF)的数据建模语言和相关 RFC。

5.4.2 Netopeer2

5.4.2.1 概述

Netopeer2 是基于 NETCONF 协议实施网络配置工具的工具集。这是第二代工具集，最初作为 Netopeer 项目提供。它基于新一代的 NETCONF 和 YANG 库——**libyang** 和 **libnetconf2**。Netopeer2 server 使用 **sysrepo** 作为 NETCONF 数据存储实施。在实时边缘软件中，使用了 **v0.7-r2** 版本。它允许开发人员通过 NETCONF 控制其设备，并且操作员可以连接到其支持 NETCONF 的设备。

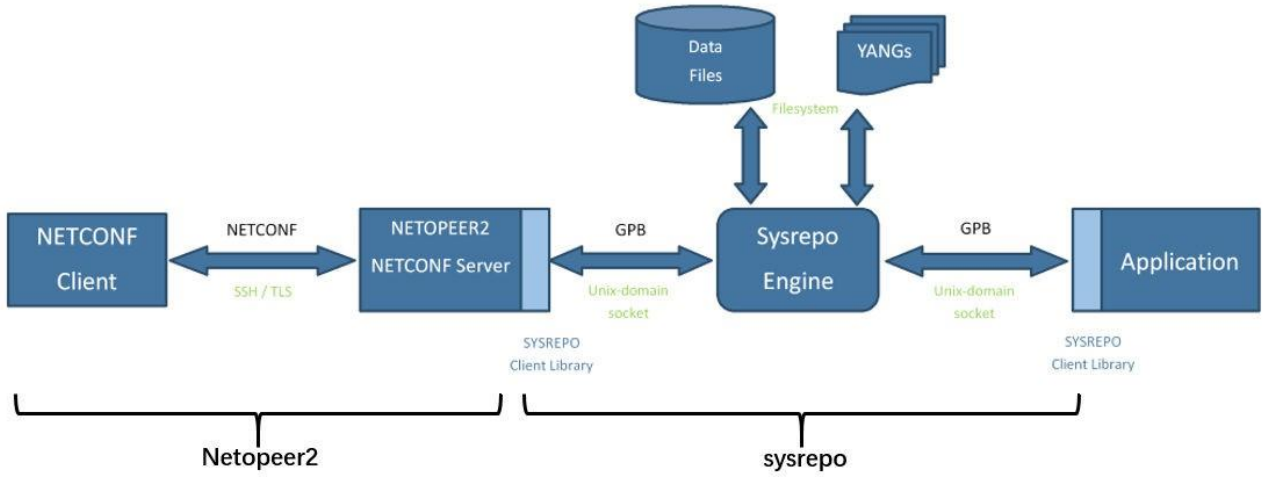


图 52.Netopeer 和 sysrepo 的高级架构

5.4.2.2 在 Ubuntu18.04 上安装 Netopeer2-cli

使用以下步骤在 Ubuntu18.04 操作系统上安装 Netopeer2-cli。

1. 安装以下软件包：

```
$ sudo apt install -y git cmake build-essential bison autoconf dh-autoreconf flex
$ sudo apt install -y libavl-dev libprotobuf-c-dev protobuf-c-compiler zlib1g-dev
$ sudo apt install -y libgcrypt20-dev libssh-dev libev-devlibpcre3-dev
```

2. 安装 libyang：

```
$ git clone https://github.com/CESNET/libyang.git
$ cd libyang;
$ git checkout v1.0-r4 -b v1.0-r4
$ mkdir build; cd build
$ cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr ..
$ make
$ sudo make install
```

3. 安装 sysrepo (v0.7.8)：

```
$ git clone https://github.com/sysrepo/sysrepo.git
$ cd sysrepo
$ git checkout v0.7.8 -b v0.7.8
$ mkdir build; cd build
$ cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX:PATH=/usr..
$ make
$ sudo make install
```

4. 安装 libnetconf2：

```
$ git clone https://github.com/CESNET/libnetconf2.git
$ cd libnetconf2
$ git checkout v0.12-r2 -b v0.12-r2
$ mkdir build; cd build
```

```
$ cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr ..
$ make
$ sudo make install
```

5. 安装 protobuf：

```
$ git clonehttps://github.com/protocolbuffers/protobuf.git
$ cd protobuf
$ git submodule update --init --recursive
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
$ sudo ldconfig # refresh shared library cache.
```

6. 安装 Netopeer2-cli(v0.7-r2)：

```
$ git clone https://github.com/CESNET/Netopeer2.git
$ cd Netopeer2
$ git checkout v0.7-r2 -b v0.7-r2
$ cd cli
$ cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr .
$ make
$ sudo make install
```

5.4.2.3 Sysrepo

Sysrepo 是用于 Unix/Linux 应用程序的基于 YANG 的配置和操作状态数据存储。

应用程序可以使用 sysrepo 来存储由提供的 YANG 模型建模的配置，而不是使用平面配置文件等。在实时边缘软件中，使用了 **v0.7.8** 版本。Sysrepo 将确保存储在数据存储中的数据的一致性，并强制执行 YANG 模型定义的数据约束。应用程序目前可以使用 sysrepo 客户端库的 C 语言 API 来访问数据存储中的配置，但对其他编程语言的支持也在计划实施（由于 sysrepo 使用 Google Protocol Buffers 作为数据存储和客户端库之间的接口，可以为任何能支持 GPB 的编程语言编写本地客户端库）。

有关 sysrepo 的信息，请参见：

http://www.sysrepo.org/static/doc/html/index.html

5.4.2.4 Netopeer2 server

Netopeer2 软件是一组实用程序和工具，用于支持主要应用程序 Netopeer2 server，它是 NETCONF 服务器的实施。它使用 libnetconf2 进行所有 NETCONF 通信。它符合相关的 RFCs2 并且仍然是上述库的一部分，支持强制 SSH 作为传输协议，也支持 TLS。一旦客户端使用这些传输协议中的任何一个成功连接并建立 NETCONF 会话，它就可以发送 NETCONF RPC，并且 Netopeer2 server 以正确的回复进行响应。

以下工具集是 Netopeer 服务器的一部分：

- Netopeer2-keystored 作为存储和处理密钥的工具。
- Netopeer2-server 作为集成 SSH/TLS 服务器的主要服务守护进程。

5.4.2.5 Netopeer2 客户端

Netopeer2-cli 是一个 CLI 接口，允许用户连接到使能 NETCONF 的设备并获取和操控其配置数据。

此应用程序是 Netopeer2 软件包的一部分，但单独编译和安装。它是一个带有命令行接口的 NETCONF 客户端，旨在主要用于 Netopeer2 server 测试，但允许使用成熟 NETCONF 客户端的所有标准甚至一些可选功能。

Netopeer2-cli 用作通用 NETCONF 客户端，提供简单的交互式命令行接口。它允许用户与网络上使能 NETCONF 的设备建立 NETCONF 会话，并获取和操控其配置数据。

5.4.2.6 应用实践中的工作流程

在实际应用中，我们使用 YANG 语言对设备进行建模，生成 YANG 模型。然后实例化模型以生成 XML 格式的配置文件。再通过 netopeer 使用此配置文件将设备配置为输入。

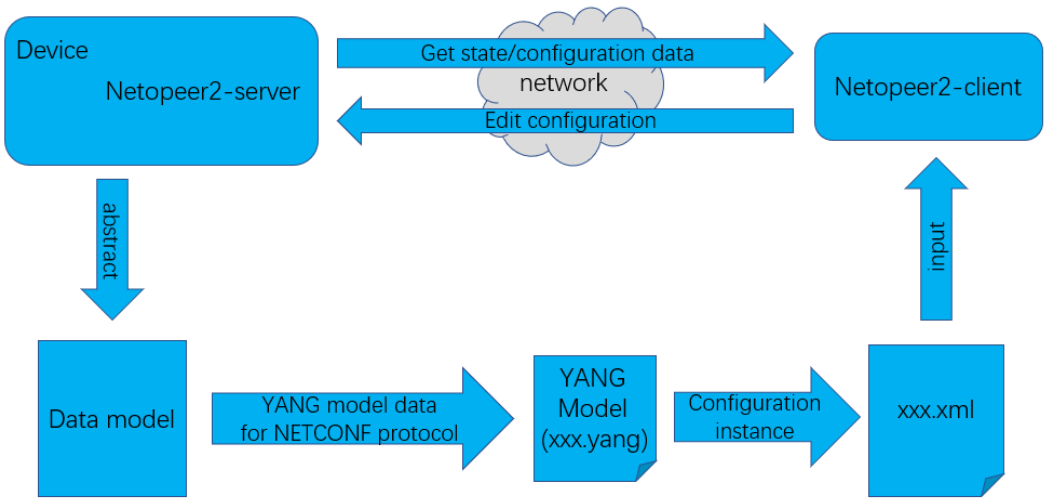


图 53.netopeer 的工作流程

5.4.3 配置

5.4.3.1 使能 NETCONF 功能

实时边缘软件默认使能此功能，使用以下命令构建镜像

```
DISTRO=nxp-real-time-edge MACHINE=ls1028ardb source real-time-edge-setup-env.sh -b build-ls1028ardb
```

或

```
DISTRO=nxp-real-time-edge MACHINE=ls1021atsn source real-time-edge-setup-env.sh -b build-ls1021atsn
```

实时边缘软件默认使能以下软件包：

```
netopeer2-keystored netopeer2-server real-time-edge-sysrepo
```

sysrepo-tsn 是基于 **sysrepo** 实施 tsn 配置的守护进程。已为 **LS1028ARDB**、**LS1021A-TSN** 和 **i.MX 8M Plus EVK** 使能该守护进程。

注意

- 对于 LS1028ARDB 板，支持 Qbv、Qbu、Qci、CB 中的流识别、IP、MAC 和 VLAN。
- 对于 LS1021ATSN 板，支持 Qbv、IP、MAC 和 VLAN。
- real-time-edge-sysrepo 仅在 **LS1028ARDB**、**LS1021ATSN** 和 i**MX 8M Plus EVK** 构建的环境中得到验证。

5.4.3.2 Netopeer2-server

netopeer2-server 是作为系统守护进程运行的 NETCONF 协议服务器。netopeer2-server 基于 sysrepo 和 libnetconf2 库。

- -U 在 unix 套接字上进行本地侦听
- -d 调试模式（不要设置守护进程并将详细消息打印到 stderr 而不是 syslog）
- -V：显示程序版本。
- -v 级详细输出级别（0：错误，1：错误和警告，2：错误、警告和详细消息）。

5.4.3.3 Netopeer2-cli

netopeer2-cli 是类似于 NETCONF 客户端的命令行接口。它用作通用 NETCONF 客户端，提供简单的交互式命令行接口。它允许用户与网络上使能 NETCONF 的设备建立 NETCONF 会话，并获取和操控其配置数据。netopeer2-cli 通过正向或反向(Call Home)连接方法一次仅限于单个 NETCONF 连接。

5.4.3.3.1 Netopeer2 CLI 命令

以下是 Netopeer2 CLI 命令：

1. **help**：显示命令列表。所有命令也接受`--help`选项，以显示有关命令的详细信息。
2. **connect**：连接到 NETCONF 服务器。

```
connect[--help] [--ssh] [--host <hostname>] [--port <num>] [--login <username>]
```

connect 命令具有以下参数：

- **--login** 用户名：指定在 NETCONF 服务器上登录的用户。如果未指定，则采用当前本地用户名。
- **--port** 号
  - NETCONF 服务器上要连接的端口。默认情况下，SSH 使用端口 830，TLS 传输使用端口 6513。
- **host**
  - 目标 NETCONF 服务器的主机名或 IP 地址。

3. **disconnect**：与 NETCONF 服务器断开。
4. **commit**
   - 执行 NETCONF `commit` 操作。有关详细信息，请参见 RFC 6241 第 8.3.4.1 节。
5. **copy-config**：执行 NETCONF `copy-config` 操作。有关详细信息，请参见 RFC 6241 第 7.3 节。

```
copy-config [--help] --target running|startup|candidate|url:<url> (--source running|startup|
candidate|url:<url> | --src-config[=<file>])
    [--defaultsreport-all|report-all-tagged|trim|explicit]
```

其中，参数如下：

- **--defaults** 模式：使用：具有指定检索模式的`-defaults` 功能。有关详细信息，请参见本手册的 RFC 6243 第 3 节或*默认值*一节。
- **--target** 数据存储：指定 `copy-config` 操作的目标数据存储。有关数据存储参数的说明，请参见 Netopeer2 CLI datastore。
- **--source** 数据存储：为 `copy-config` 操作指定源数据存储。有关数据存储参数的说明，请参见 Netopeer2 CLI datastore。

6. **delete-config** 执行 NETCONF`delete-config` 操作。有关详细信息，请参见 RFC 6241 规范的第 7.4 节。

```
delete-config [--help] --target startup|url:<url>
```

其中

- **target** 数据存储：为 `delete-config` 操作指定目标数据存储。

7. **edit-config**

执行 NETCONF`edit-config` 操作。有关详细信息，请参见 RFC 6241 第 7.2 节。

```
edit-config [--help] --target running|candidate (--config[=<file>] | --url <url>)
    [--defop merge|replace|none] [--test set|test-only|test-then-set] [--errorstop|
continue|rollback]
```

其中

- **`--defop`** 操作：

  指定应用配置数据的默认操作。

  - `merge`：合并相应级别的配置数据。默认情况下，该值为 `merge`。
  - `replace`：编辑配置数据完全替换目标数据存储中的配置。
  - `none`：目标数据存储不受编辑配置数据的影响，除非直到编辑配置数据包含请求不同操作的操作属性。有关详细信息，请参见 RFC 6241 的 EDIT-CONFIG 一节。

  注意

  要删除非强制项，应将 **nc:operation="delete"** 添加到要删除的项的开始标记的末尾。同时，命名空间 **xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"** 也应该添加到根节点的开始标记中。强制项不能单独删除。它们只能与其父节点一起删除。

- **--error** 操作

  设置对错误的反应。

  - `Stop`：在第一个错误时中止操作。这是默认值。
  - `Continue`：出错时继续处理配置数据。记录错误并返回否定响应。
  - `Rollback`：出错时停止操作处理，并将配置恢复到此操作开始时的完整状态。仅当服务器支持错误回滚功能时，此操作才可用（参见 RFC 6241 第 8.5 节）。

- **--test** 选项

  对修改后的配置数据进行验证。仅当服务器支持`:validate:1.1` 功能时，此选项才可用（参见 RFC 6241 第 8.6 节）。

  - `set`：不执行验证测试。

- `test-only`：不应用修改后的数据，仅执行验证测试。
- `test-then-set`：在尝试应用修改后的配置数据之前执行验证测试。`test-then-set` 是默认值。

- **--config** 文件
  - 指定包含编辑配置数据的文件的路径。文件的内容被放入 edit-config 操作的 `config` 元素中。因此，它不必是只有一个根元素的格式良好的 XML 文档。如果既没有指定--config 也没有指定--url，则提示用户手动编写编辑配置数据。例如，请参见 RFC 6241 的 EDIT-CONFIG 一节。
- **--url** URI
  - 指定包含要修改的配置数据层次结构的文件的远程位置，在 `urn:ietf:params:xml:ns:netconf:base:1.0` 命名空间中的 `config` 元素下以 XML 编码。请注意，这与文件参数不同，其中不需要 `config` 元素。
- **--target**
  - 要修改的目标数据存储。有关可能值的说明，请参见 Netopeer2 CLI 数据存储。请注意，无法修改 url 配置数据存储。

8. **get**：执行 NETCONF`get` 操作。从当前运行的数据存储接收状态和配置数据。有关详细信息，请参见 RFC 6241 规范的第 7.7 节。命令格式如下：

```
get[--help] [--filter-subtree[=<file>] | --filter-xpath <XPath>] [--defaults report-
all|report- all-tagged|trim|explicit] [--out <file>]
```

- **--defaults** 模式
  - 组合使用`-defaults` 功能和指定检索模式。有关更多详细信息，请参见 RFC 6241 规范的第 3 节或“默认值”一节。
- **--filter**[文件]
  - 指定请求是否包含子树过滤器（RFC 6241 第 6 节）。该选项能够接受包含过滤器规范的文件路径。如果未指定路径，则提示用户手动编写过滤器规范。

9. **get-config** 执行 NETCONF `get-config` 操作。仅从指定的 target_datastore 检索配置数据。有关详细信息，请参见 RFC 6241 第 7.1 节。

```
get-config [--help] --source running|startup|candidate [--filter-subtree[=<file>] | --
filter- xpath <XPath>]
    [--defaults report-all|report-all-tagged|trim|explicit] [--out<file>]
```

10. **--defaults** 模式
    - 使用：具有指定检索模式的`-defaults` 功能。有关更多详细信息，请参见本手册的 RFC 6243 第 3 节或“默认值”一节。
11. **--filter**[文件]
    - 指定请求是否包含子树过滤器（RFC 6241 第 6 节）。该选项能够接受包含过滤器规范的文件路径。如果未指定路径，则提示用户手动编写过滤器规范。
12. **--target**
    - 要检索的目标数据存储。有关可能值的说明，请参见 Netopeer2 CLI 数据存储。请注意，无法检索 url 配置数据存储。
13. **lock**

执行 NETCONF`lock` 操作，以锁定服务器的整个配置数据存储。有关详细信息，请参见 RFC 6241 第 7.5 节。

```
lock [--help] --target running|startup|candidate
```

其中

- **--target**：指定要锁定的目标数据存储。有关可能值的说明，请参见 Netopeer2 CLI 数据存储。请注意，无法锁定 url 配置数据存储。

14. **unlock**：执行 NETCONF`unlock` 操作，以释放先前通过 `lock` 操作获得的配置锁定。有关详细信息，请参见 RFC 6241 规范的第 7.6 节。

```
unlock [--help] --target running|startup|candidate
```

其中

- **--target**：指定要解锁的目标数据存储。有关可能值的说明，请参见 Netopeer2 CLI 数据存储。请注意，无法解锁 url 配置数据存储。

15. **verb**
    - 使能/禁用详细消息。
16. **quit**
    - 退出程序。

5.4.3.3.2 Netopeer2 CLI 数据存储

以下是 netopeer2 CLI 数据存储：

- **running**
  - 这是保存设备上当前活动的完整配置的基本 NETCONF 配置数据存储。此数据存储始终存在。
- **startup**
  - 保存设备启动时加载的配置的配置数据存储。此数据存储仅在实施`:startup` 功能的服务器上可用。
- **candidate**
  - 可以在不影响设备当前配置的情况下进行操控，并且可以提交到正在运行的配置数据存储的配置数据存储。此数据存储仅在实施`:candidate` 功能的服务器上可用。
- **url**:URI
  - 指位于 URI 的远程配置数据存储。URI 引用的文件包含要修改的配置数据层次结构，在 `urn:ietf:params:xml:ns:netconf:base:1.0` 命名空间中的 `config` 元素下以 XML 编码。此数据存储仅在实施`:url` 功能的服务器上可用。

5.4.3.4 Sysrepod

Sysrepo 守护进程提供系统上数据存储的功能（执行系统范围的 Sysrepo 引擎）。在正常情况下，它会在系统启动时自动启动。但是，自动启动不是由 `cmake install` 操作配置，用户应根据用户系统的指导进行手动配置。

用法：

```
sysrepod [-h] [-v] [-d] [-l <level>]
```

选项：

- -h 打印使用帮助。
- -v 打印版本。
- -d 调试模式——守护进程在前台运行，并将日志打印到 stderr 而不是 syslog。
- -l<level>设置日志记录的详细级别：
  - 0 = 关闭所有日志记录
  - 1 = 仅记录错误消息
  - 2 =（默认）记录错误和警告消息
  - 3 = 记录错误、警告和信息性消息
  - 4 = 记录所有内容，包括开发调试消息

5.4.3.5 Sysrepocfg

sysrepocfg 是用于编辑、导入和导出存储在 Sysrepo 数据存储中的配置的命令行工具。它允许用户在首选的文本编辑器中编辑指定模块的启动或运行配置。它还为所有订阅的应用程序透明地将执行的更改传播到数据存储中。此外，用户可以将当前配置导出到文件中或将其打印到标准输出中。同样，已经准备好的配置可以从文件中导入或从标准输入中读取。

在后台，**sysrepocfg** 使用 Sysrepo 客户端库进行任何数据操控，而不是直接访问配置数据文件。因此，它有效地继承了 Sysrepo 的所有主要功能，例如基于 YANG 的数据验证、完整的事务和并发支持。最重要的是，订阅的应用程序会收到有关使用 \fBsysrepocfg\fP 所做更改的通知，并且可以立即将新配置考虑在内。

5.4.3.6 Sysrepoctl

sysrepoctl 提供管理模块的功能。可以使用下面描述的选项和命令对其进行配置。

操作-操作

- **--help**：打印一般说明和命令列表。使用命令的`--help`参数显示特定命令的详细说明和参数列表。
- **--install**：将指定的模式安装到 sysrepo 中（必须指定--yang 或--yin）。
- **--uninstall**：从 sysrepo 卸载指定的模式（必须指定--module）。
- **--list**：列出安装在 sysrepo 中的 YANG 模块（请注意，已安装“一致性”意味着也已实施）。
- **--change**：更改 sysrepo 中的指定模块（必须指定--module）。
- **--feature-enable**：使能 sysrepo 中模块内的功能（功能名称是参数，必须指定--module）。
- **--feature-disable**：禁用 sysrepo 中模块内的功能（功能名称是参数，必须指定--module）。

其他-选项

- **--yang**：带有 YANG 格式模式的文件路径（--install 操作）。
- **--yin**：带有 YIN 格式模式的文件路径（--install 操作）。
- **--module**：要操作的模块的名称（--change、--feature-enable、--feature-disable 操作、--uninstall——几个模块可以用“,”分隔）。
- **--permissions**：chmod 格式的模块数据的访问权限（--install、--change 操作）。

示例

- 通过指定 YANG 文件、所有权和访问权限来安装新模块：

```
sysrepoctl --install --yang=/home/user/ietf-interfaces.yang --owner=admin:admin --permissions=644
```

- 更改现有 YANG 模块的所有权和权限：

```
sysrepoctl --change --module=ietf-interfaces --owner=admin:admin --permissions=644
```

- 使能 YANG 模块中的功能：

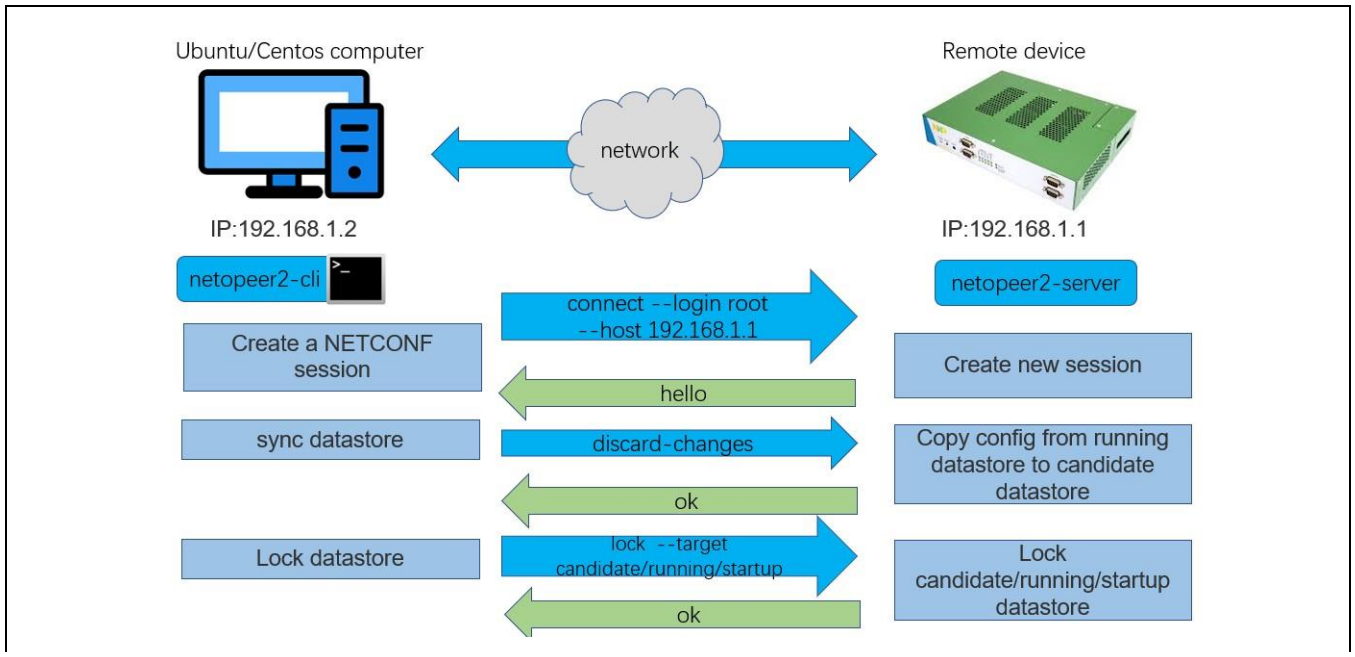
```
sysrepoctl --feature-enable=if-mib--module=ietf-interfaces
```

- 卸载 2 个模块，第二个没有修改：

```
sysrepoctl --uninstall --module=mod-a,mod-b--revision=2035-05-05
```

5.4.3.7 操作示例

下图描述了通过 netopeer2 配置设备的步骤：



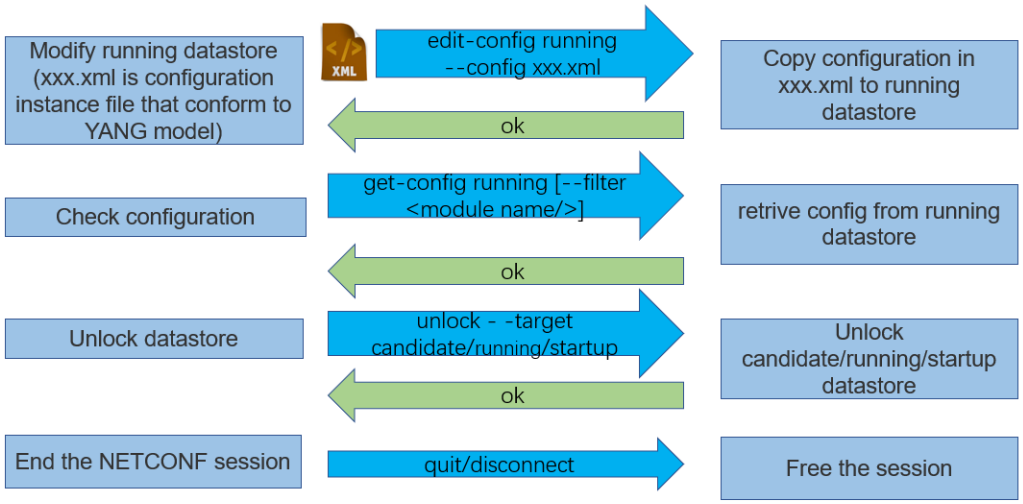


图 54.通过 netopeer2 配置设备的步骤

在 sysrepo-tsn 中，有一些实例文件用于在 LS1028ARDB 板上配置 TSN 功能：

- TSN 配置的实例文件

用户可以使用这些实例文件配置 LS1028ARDB 板的 TSN 功能。在开始之前，请确保 **sysrepod**、**sysrepo-plugind**、**sysrepo-tsn** 和 **netopeer2-server** 正在板上运行。使用以下步骤在 LS1028ARDB 板上配置 TSN 功能。

1. 在安装了 netopeer2-cli 的计算机上启动 netopeer2-cli：

```
$ netopeer2-cli
```

2. 连接到 LS1028ARDB 板上的 netopeer2-server（使用 LS1028ARDB 上的 IP，这里以 10.193.20.53 为例）：

```
> connect --login root --host 10.193.20.53
```

3. 获取服务器状态数据：

```
> get
```

4. 在正在运行的数据存储中获取配置数据：

```
>get-config --source running
```

5. 使用 qbv-eno0-enable.xml 配置 LS1028ARDB 的 QBV 功能

```
>edit-config --target running--config=qbv-eno0-enable.xml
```

6. 检查 QBV 的配置数据：

```
>get-config --source running --filter-xpath /ietf-interfaces:interfaces/interface[name='eno0']/
ieee802-dot1q-sched:gate-parameters
```

7. 将 **running** 数据存储中的配置数据复制到 **startup** 数据存储中：

```
>copy-config --source running --target startup
```

8. 断开与 netopeer2-server 的连接：

```
> disconnect
```

5.4.3.8 应用场景

注意

以下情况下的相关 xml 文件可从此链接获取：https://github.com/real-time-edge-sw/real-time-edge-sysrepo/blob/master/Instances.

注意

xml 文件中的接口名称应与板上实际使用的接口名称匹配。

1. 前提条件：
   a. 在安装了 netopeer2-cli 的计算机上启动 netopeer2-cli：

```
$ netopeer2-cli
```

   b. 使用以下命令连接到 notopeer2-server：

```
> connect --login root --host 10.193.20.53
```

2. 配置 IP 地址：
   a. 编辑配置文件，更改以太网接口名称和 IP：

```
$ vim ietf-ip-cfg.xml
```

   b. 发送配置文件：

```
>edit-config --target running--config=ietf-ip-cfg.xml
```

3. 配置网桥的 MAC 地址：
   a. 创建一个名为 `br1` 的网桥：

```
$ ip link add name br1 type bridge
```

   b. 编辑配置文件，更改网桥名称和 MAC：

```
$ vim ietf-mac-cfg.xml
```

   c. 发送配置文件：

```
$ edit-config --target running--config=ietf-mac-cfg.xml
```

4. 为以太网接口添加 VLAN：
   a. 如果不存在，则创建名为“`br1`”的网桥：

```
$ ip link add name br1 type bridge
```

   b. 编辑配置文件以更改接口名称和 VLAN ID：

```
$ vim ietf-vlan-cfg.xml
```

c. 发送配置文件：

```
>edit-config --target running--config=ietf-vlan-cfg.xml
```

5. 通过 tc 配置 LS1028ARDB Qbv。

a. 编辑配置文件以更改接口名称和 VLAN ID：

```
$ vim qbv-swp0-enable.xml
```

b. 发送配置文件：

```
>edit-config --target running--config=qbv-swp0-enable.xml
```

c. 显示结果。

```
# tc qdisc show dev swp0
```

注意

如果使用 tc 或 ethtool 命令而不是 libtsn，请在 conf/ distro/include/real-time-edge-base.inc 中使能“ real-time-edge-sysrepo-tc”，如下所示：

```
REAL_TIME_EDGE_SYSREPO_ls1028ardb ="real-time-edge-sysrepo-tc"
```

否则，请禁用“real-time-edge-sysrepo-tc”：

```
REAL_TIME_EDGE_SYSREPO_ls1028ardb = ""
```

- 对于 LS1028ARDB 板，如果使能了 real-time-edge-sysrepo-tc，则应使用以下命令设置 swpx（swp0 swp1 或 swp2.....）端口的前提条件：

```
# tc qdisc add dev swpx ingress
# tc filter add dev swpx ingress chain 0 pref 49152 flower skip_sw action goto chain 10000
# tc filter add dev swpx ingress chain 10000 pref 49152 flower skip_sw # actiongoto chain 11000
# tc filter add dev swpx ingress chain 11000 pref 49152 flower skip_sw actiongoto chain 12000
# tc filter add dev swpx ingress chain 12000 pref 49152 flower skip_sw actiongoto chain 20000
# tc filter add dev swpx ingress chain 20000 pref 49152 flower skip_sw actiongoto chain 21000
# tc filter add dev swpx ingress chain 21000 pref 49152 flower skip_sw actiongoto chain 30000
```

6. 使用以下步骤通过 tc 配置 LS1028ARDB Qci。

a. 如果不存在，则创建一个名为“switch”的网桥：

```
# ip link add name switch type bridge
```

b. 编辑并发送配置文件：

```
edit-config --target running--config=switch-qci-fm-gate-enable.xml
```

c. 显示结果。

```
# tc filter show dev swp0 ingress
```

d. 禁用配置。

```
>edit-config --target running--config=switch-qci-fm-gate-disable.xml
```

注意

- 交换机应知晓实例文件中的目标地址。
- 用户应在 switch-qci-fm-gate-enable.xml 之后发送 switch-qci-fm-gate-disable.xml

7. 使用以下步骤通过 ethtool 配置 LS1028ARDB Qbu。

a. 编辑并发送配置文件：

```
>edit-config --target running --config=qbu-swp0.xml
```

b. 显示结果：

```
# ethtool --show-frame-preemption swp0
```

8. 通过 tc 配置 LS1028ARDB VLAN ID 和优先级过滤器：

a. 编辑配置文件，更改接口名称和操作规范：

```
$ vim ietf-br-vlan-cfg.xml
```

b. 发送配置文件：

```
>edit-config --target running--config=ietf-br-vlan-cfg.xml
```

9. 通过 tc 配置 SJA1105 Qbv

a. 编辑并发送配置文件

```
>edit-config --target running--config=qbv-swp5-tc.xml
```

b. 显示结果：

```
# tc qdisc show dev swp5
```

注意

由于 SJA1105 的硬件限制，如果您想要重新配置 `qbv`，则应当使用以下命令删除之前的 `qdisc`：

```
# tc qdisc del dev swp5 parent root handle 100
```

10. 通过 tc 配置 SJA1105 Qci 门

a. 如果不存在，则创建名为“`switch`”的网桥：

```
# ip link add name switch type bridge
```

b. 编辑并发送配置文件：

```
>edit-config --target running--config=switch-qci-gate-swp2-enable.xml
```

c. 显示结果：

```
# tc filter show dev swp2 ingress
```

注意

如果您想要测试 Qci 流量计，则必须发送 `switch-qci-fm-swp2-enable.xml`。

注意

用于调度入口门操作的引擎与用于 tc-taprio 分流的引擎相同。因此，不能同时触发两个门操作（tc-gate 或 tc-taprio 门）这一事实的相关限制（在相同的 200 ns 时隙内）仍然适用。

11. 通过 tc 配置 i.MX 8M Plus Qbv。
    a. 编辑并发送配置文件：

```
>edit-config --target running--config=qbv-eth1-enable.xml
```

    b. 使用以下命令显示结果：

```
# tc qdisc show dev eth1
```

12. 通过 ethtool 配置 i.MX 8M Plus Qbu。
    a. 编辑并发送配置文件：

```
>edit-config --target running --config=qbu-eth1.xml
```

    b. 使用以下命令显示结果：

```
# ethtool --show-frame-preemption eth1
```

5.4.4 故障排除

1. 客户端连接失败：

```
nc ERROR: Remote host key changed, the connection will beterminated!
nc ERROR: Checking the host key failed.
cmd_connect: Connecting to the 10.193.20.4:830 as user "root"failed.
```

修复：

原因是服务器上的 SSHD 密钥发生了变化。

- 首先，用户应当使用命令 `knownhosts` 获取主机列表。
- 然后，删除相关项目。例如 `knownhosts --del 19`。

2. 请求无法完成，因为相关数据模型内容不存在。

```
type:     application
tag:      data-missing
severity: error
path:     /ietf-interfaces:interfaces/interface[name='eno0']/ieee802-dot1q-sched:gate-
parameters/admin-gate-states
message: Request could not be completed because the relevant data model content does not exist.
```

修复：

原因是数据存储中不存在 xpath 中的配置数据。比如删除一个不存在的节点。

遇到此类错误时，用户可以通过 `get-config` 命令获取板中的配置数据，检查路径中节点的操作类型(`add/delete/modify`)是否合理。

5.5 LS1028A 上的图形

本章节适用于 LS1028A。对于 i.MX 8M Plus 和 8M Mini，请参见 i.MX 图形用户指南以验证功能。

5.5.1 GPU

GPU 包括一个 3D 图形内核和一个 2D 图形内核。

3D 图形内核功能如下：

- 支持每秒 1.66 亿个三角形
- 支持每秒 1 千兆像素的填充率
- 支持 16 GFLOP
- 支持 OpenGL ES 1.1、2.0、3.0、3.1
- 支持 OpenCL 1.2
- Vulkan

2D 图形内核功能有：

- 支持多源合成
- 支持单通滤波器
- 支持来自 3D 图形内核的平铺格式
- 支持来自 VPU 的平铺格式

第 1 步：软件设置和配置

为 i.MX 8M Plus、i.MX 8M Mini 和 LS1028A 编译镜像时默认使能 GPU。

第 2 步：硬件设置

- 对于 LS1028ARDB，用 DP 线缆连接监测器和 LS1028ARDB。
- 对于 i.MX 8M Mini EVK，将 MIPI-DSI 连接到 HDMI 模块，然后连接到监测器。
- 对于 i.MX 8M Plus EVK，使用 HDMI 线缆连接监测器和 i.MX 8M Plus EVK。

将 USB 鼠标插入键盘的 USB 端口。

第 3 步：运行 GPU 演示

OpenCL 演示（示例 A 和 B）仅适用于 LS1028ARDB 和 i.MX 8M Plus EVK。i.MX 8M Mini EVK 不支持此功能。

A. OpenCL 信息

```
root@LS1028ARDB:~# cd /usr/share/examples/viv_samples/cl11/UnitTest
root@LS1028ARDB-Ubuntu:/usr/share/examples/viv_samples/cl11/UnitTest# ./clinfo

>>>>>>>> ./clinfo Starting...

Available platforms: 1

Platform ID: 0

     CL_PLATFORM_NAME:       Vivante OpenCL Platform
     CL_PLATFORM_PROFILE:    FULL_PROFILE
     CL_PLATFORM_VERSION:    OpenCL 1.2 V6.4.0.p2.234062
     CL_PLATFORM_VENDOR:     Vivante Corporation
     CL_PLATFORM_EXTENSIONS: cl_khr_icd
```

```
      Available devices:      1

      Device ID:       0
      Device Ptr:      0xd04742f0
            CL_DEVICE_NAME: Vivante OpenCL Device GC7000UL.6202.0000
            CL_DEVICE_VENDOR: Vivante Corporation
            CL_DEVICE_TYPE:                                         GPU
            CL_DEVICE_OPENCL_C_VERSION:                             OpenCL C 1.2
            CL_DEVICE_VENDOR_ID:                                    0x00564956
            CL_DEVICE_PLATFORM:                                     0x9e272728
            CL_DEVICE_VERSION:                                      OpenCL 1.2
            CL_DEVICE_PROFILE:                                      FULL_PROFILE
            CL_DRIVER_VERSION:                                      OpenCL 1.2 V6.4.0.p2.234062
            CL_DEVICE_MAX_COMPUTE_UNITS:                            1
            CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS:                     3
                  CL_DEVICE_MAX_WORK_ITEM_SIZES[0]:                 512
                  CL_DEVICE_MAX_WORK_ITEM_SIZES[1]:                 512
                  CL_DEVICE_MAX_WORK_ITEM_SIZES[2]:                 512
                  CL_DEVICE_MAX_WORK_GROUP_SIZE:                    512
            CL_DEVICE_MAX_CLOCK_FREQUENCY:                          650 MHz
            CL_DEVICE_IMAGE_SUPPORT:                                Yes
                  CL_DEVICE_MAX_READ_IMAGE_ARGS:                    128
                  CL_DEVICE_MAX_WRITE_IMAGE_ARGS:                   8
                  CL_DEVICE_IMAGE2D_MAX_WIDTH:                      8192
                  CL_DEVICE_IMAGE2D_MAX_HEIGHT:                     8192
                  CL_DEVICE_IMAGE3D_MAX_WIDTH:                      8192
                  CL_DEVICE_IMAGE3D_MAX_HEIGHT:                     8192
                  CL_DEVICE_IMAGE3D_MAX_DEPTH:                      8192
                  CL_DEVICE_MAX_SAMPLERS:                           16
...
```

B. 基于 GPU 的傅里叶变换

```
root@LS1028ARDB-Ubuntu:~# cd /usr/share/examples/viv_samples/cl11/fft/
root@LS1028ARDB-Ubuntu:/usr/share/examples/viv_samples/cl11/fft# ./fft 16
Block size: 16
Print result: yes

Initializing device(s)...
Get the Device info and select Device...
# of Devices Available = 1
# of Compute Units = 1
# compute units = 1 Creating Command Queue...
log2(fft size) = log2(16)=4
Compiling radix-2 FFT Program for GPU...
creating radix-2 kernels...
Creating kernel fft_radix2 0 (p=1)...
Creating kernel fft_radix2 1 (p=2)...
Creating kernel fft_radix2 2 (p=4)...
Creating kernel fft_radix2 3 (p=8)...
Setting kernel args for kernel 0 (p=1)...
Setting kernel args for kernel 1 (p=2)...
Setting kernel args for kernel 2 (p=4)...
Setting kernel args for kernel 3 (p=8)...
running kernel 0 (p=1)...
running kernel 1 (p=2)...
running kernel 2 (p=4)...
running kernel 3 (p=8)...
Kernel execution time on GPU (kernel 0) :    0.000118 seconds
```

```
Kernel execution time on GPU (kernel 1) :    0.000122 seconds
Kernel execution time on GPU (kernel 2) :    0.000102 seconds
Kernel execution time on GPU (kernel 3) :    0.000076 seconds
Total Kernel execution time on GPU :  0.000418 seconds
Successful.
```

C. OpenGL ES 演示

kmscube 用于测试 OpenGL ES，它支持 HDMI 和 eDP 接口。

对于 eDP 接口，由于固件限制，不支持 4K 分辨率。

```
root@LS1028ARDB:~# kmscube
Using display 0x3107b6d0 with EGL version 1.5
===================================
EGL information:
   version: "1.5"
   vendor: "Vivante Corporation"
   client extensions: "EGL_EXT_client_extensions EGL_EXT_platform_base EGL_KHR_platform_wayland
EGL_EXT_platform_wayland EGL_KHR_platform_gbm"
   display extensions: "EGL_KHR_fence_sync EGL_KHR_reusable_sync EGL_KHR_wait_sync EGL_KHR_image
EGL_KHR_image_base EGL_KHR_image_pixmap EGL_KHR_gl_texture_2D_image EGL_KHR_gl_texture_cubemap_image
EGL_KHR_gl_renderbuffer_image EGL_EXT_image_dma_buf_import EGL_EXT_image_dma_buf_import_modifiers
EGL_KHR_lock_surface EGL_KHR_create_context EGL_KHR_no_config_context EGL_KHR_surfaceless_context
EGL_KHR_get_all_proc_addresses EGL_EXT_create_context_robustness EGL_EXT_protected_surface
EGL_EXT_protected_content EGL_EXT_buffer_age EGL_ANDROID_native_fence_sync EGL_WL_bind_wayland_display
EGL_WL_create_wayland_buffer_from_image EGL_KHR_partial_update EGL_EXT_swap_buffers_with_damage
EGL_KHR_swap_buffers_with_damage"
===================================
OpenGL ES 2.x information:
   version: "OpenGL ES 3.1 V6.4.0.p2.234062"
   shading language version: "OpenGL ES GLSL ES 3.10"
   vendor: "Vivante Corporation"
   renderer: "Vivante GC7000UL"
   extensions: "GL_OES_vertex_type_10_10_10_2 GL_OES_vertex_half_float GL_OES_element_index_uint
GL_OES_mapbuffer GL_OES_vertex_array_object GL_OES_compressed_ETC1_RGB8_texture
GL_OES_compressed_paletted_texture GL_OES_texture_npot GL_OES_rgb8_rgba8 GL_OES_depth_texture
GL_OES_depth_texture_cube_map GL_OES_depth24 GL_OES_depth32 GL_OES_packed_depth_stencil
GL_OES_fbo_render_mipmap GL_OES_get_program_binary GL_OES_fragment_precision_high
GL_OES_standard_derivatives GL_OES_EGL_image GL_OES_EGL_sync GL_OES_texture_stencil8
GL_OES_shader_image_atomic GL_OES_texture_storage_multisample_2d_array GL_OES_required_internalformat
GL_OES_surfaceless_context GL_OES_draw_buffers_indexed GL_OES_texture_border_clamp
GL_OES_texture_buffer GL_OES_texture_cube_map_array GL_OES_draw_elements_base_vertex
GL_OES_texture_half_float GL_OES_texture_float GL_KHR_blend_equation_advanced GL_KHR_debug
GL_KHR_robustness GL_KHR_robust_buffer_access_behavior GL_EXT_texture_type_2_10_10_10_REV
GL_EXT_texture_compression_dxt1 GL_EXT_texture_format_BGRA8888 GL_EXT_texture_compression_s3tc
GL_EXT_read_format_bgra GL_EXT_multi_draw_arrays GL_EXT_frag_depth GL_EXT_discard_framebuffer
GL_EXT_blend_minmax GL_EXT_multisampled_render_to_texture GL_EXT_color_buffer_half_float
GL_EXT_color_buffer_float GL_EXT_robustness GL_EXT_texture_sRGB_decode GL_EXT_draw_buffers_indexed
GL_EXT_texture_border_clamp GL_EXT_texture_buffer GL_EXT_texture_cube_map_array
GL_EXT_multi_draw_indirect GL_EXT_draw_elements_base_vertex GL_EXT_texture_rg
GL_EXT_protected_textures GL_EXT_sRGB GL_VIV_direct_texture "
===================================
Rendered 120 frames in 2.000008 sec (59.999758 fps)
Rendered 241 frames in 4.016689 sec (59.999663 fps)
Rendered 361 frames in 6.016730 sec (59.999368 fps)
```

下面是屏幕上的快照。

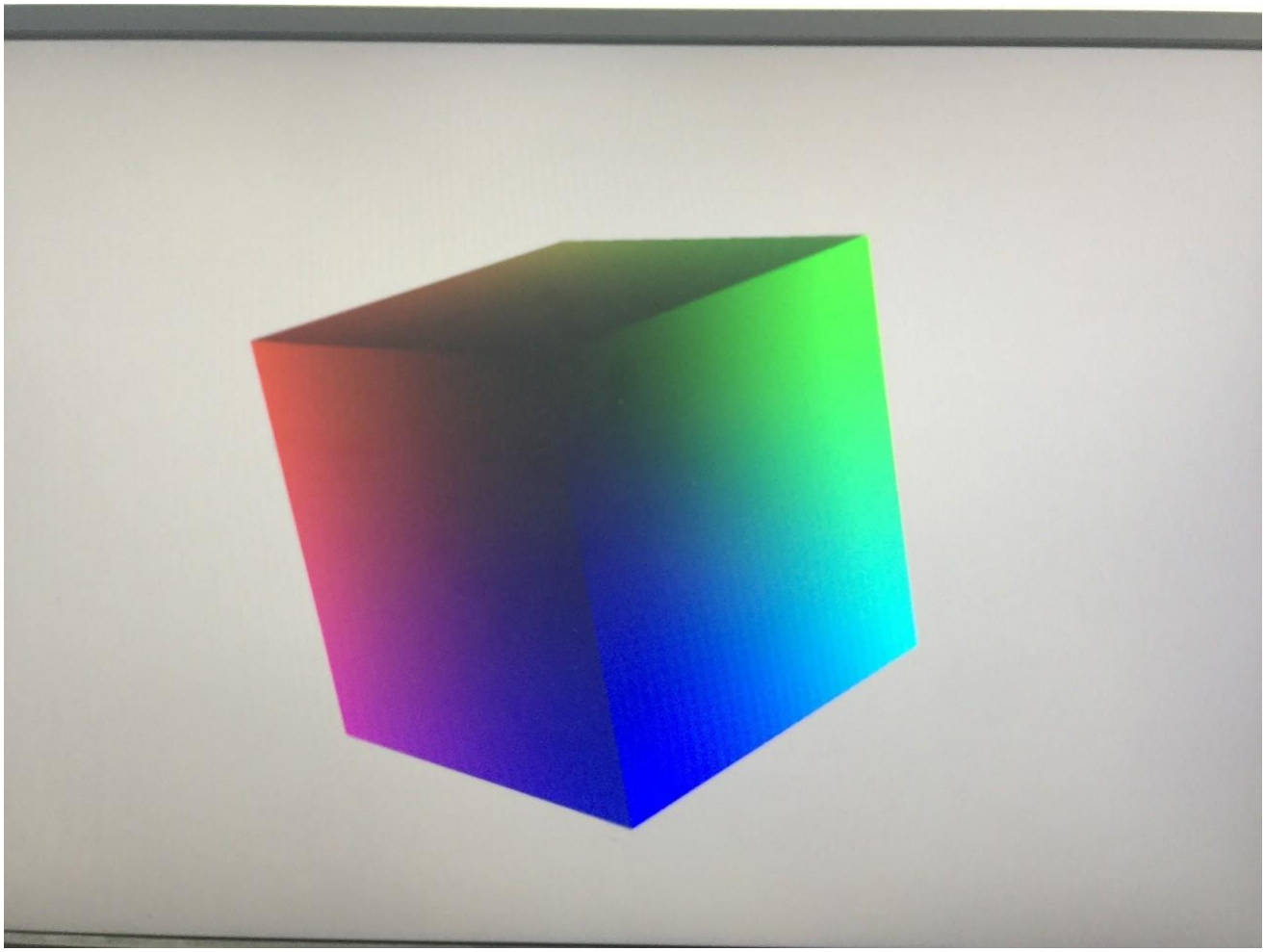


图 55.带有 kmscube 的 OpenGL ES 演示

5.5.2 Wayland 和 Weston

Weston 是 Wayland 的引用实施，本节介绍如何在恩智浦平台上使能 Weston。LS1028ARDB、iMX 8M Plus EVK 和 iMX 8M Mini EVK 平台支持 Weston。

1.软件设置和配置

在为 i.MX 8M Plus、i.MX 8M Mini 和 LS1028A 编译镜像时默认使能。

2.硬件设置

- 对于 LS1028ARDB，用 DP 线缆连接监测器和 LS1028ARDB。
- 对于 i.MX 8M Plus EVK，使用 HDMI 线缆连接监测器和 i.MX 8M Plus EVK。
- 对于 i.MX 8M Mini EVK，将 MIPI-DSI 连接到 HDMI 模块，然后连接到监测器。然后，将 USB 鼠标和键盘插入 USB 端口。

3.运行轻量级桌面

```
root@LS1028ARDB:~# mkdir -p /run/user/0/
root@LS1028ARDB:~# export XDG_RUNTIME_DIR="/run/user/0/"
root@LS1028ARDB:~# weston --tty=1
# With parameter "-i" or "--idle-time" to set the time to enter idle state, default is 300s.
# "0" means weston will not enter idle state.
```

```
root@LS1028ARDB:~# weston --tty=1 -i 0       # or
root@LS1028ARDB:~# weston --tty=1 --idle-time=0

Date: 2020-08-20 UTC
[14:38:00.002]  weston 8.0.0
                https://wayland.freedesktop.org
                Bug reports to: https://gitlab.freedesktop.org/wayland/weston/issues/
                Build: 8.0.0
[14:38:00.002]  Command line: weston --tty=1
[14:38:00.002]  OS: Linux, 5.4.3-rt1, #1 SMP PREEMPT_RT Tue Aug 18 14:49:14 CST 2020, aarch64
[14:38:00.002]  Starting with no config file.
[14:38:00.005]  Output repaint window is 16 ms maximum.
[14:38:00.007]  Loading module '/usr/lib/libweston-8/drm-backend.so'
[14:38:00.050]  initializing drm backend
[14:38:00.054]  using /dev/dri/card0 [
14:38:00.054]   DRM: supports universal planes
[14:38:00.054]  DRM: supports atomic modesetting
[14:38:00.054]  DRM: supports picture aspect ratio
[14:38:00.056]  Loading module '/usr/lib/libweston-8/gl-renderer.so'
[14:38:00.208]  EGL client extensions: EGL_EXT_client_extensions
                EGL_EXT_platform_base EGL_KHR_platform_wayland
                EGL_EXT_platform_wayland EGL_KHR_platform_gbm
[14:38:00.224]  EGL version: 1.5
[14:38:00.224]  EGL vendor: Vivante Corporation
[14:38:00.224]  EGL client APIs: OpenGL_ES OpenGL OpenVG
[14:38:00.224]  EGL extensions: EGL_KHR_fence_sync EGL_KHR_reusable_sync
                EGL_KHR_wait_sync EGL_KHR_image EGL_KHR_image_base
                EGL_KHR_image_pixmap EGL_KHR_gl_texture_2D_image
                EGL_KHR_gl_texture_cubemap_image EGL_KHR_gl_renderbuffer_image
                EGL_EXT_image_dma_buf_import
                EGL_EXT_image_dma_buf_import_modifiers EGL_KHR_lock_surface
                EGL_KHR_create_context EGL_KHR_no_config_context
                EGL_KHR_surfaceless_context EGL_KHR_get_all_proc_addresses
                EGL_EXT_create_context_robustness EGL_EXT_protected_surface
                EGL_EXT_protected_content EGL_EXT_buffer_age
                EGL_ANDROID_native_fence_sync EGL_WL_bind_wayland_display
                EGL_WL_create_wayland_buffer_from_image EGL_KHR_partial_update
                EGL_EXT_swap_buffers_with_damage
                EGL_KHR_swap_buffers_with_damage
[14:38:00.224]  EGL_KHR_surfaceless_context available
[14:38:00.310]  GL version: OpenGL ES 3.1 V6.4.0.p2.234062
[14:38:00.311]  GLSL version: OpenGL ES GLSL ES 3.10
[14:38:00.311]  GL vendor: Vivante Corporation
[14:38:00.311]  GL renderer: Vivante GC7000UL
[14:38:00.311]  GL extensions: GL_OES_vertex_type_10_10_10_2
                GL_OES_vertex_half_float GL_OES_element_index_uint
                GL_OES_mapbuffer GL_OES_vertex_array_object
                GL_OES_compressed_ETC1_RGB8_texture
                GL_OES_compressed_paletted_texture GL_OES_texture_npot
                GL_OES_rgb8_rgba8 GL_OES_depth_texture
                GL_OES_depth_texture_cube_map GL_OES_depth24 GL_OES_depth32
                GL_OES_packed_depth_stencil GL_OES_fbo_render_mipmap
                GL_OES_get_program_binary GL_OES_fragment_precision_high
                GL_OES_standard_derivatives GL_OES_EGL_image GL_OES_EGL_sync
                GL_OES_texture_stencil8 GL_OES_shader_image_atomic
                GL_OES_texture_storage_multisample_2d_array
                GL_OES_required_internalformat GL_OES_surfaceless_context
                GL_OES_draw_buffers_indexed GL_OES_texture_border_clamp
                GL_OES_texture_buffer GL_OES_texture_cube_map_array
                GL_OES_draw_elements_base_vertex GL_OES_texture_half_float
```

```
                GL_OES_texture_float GL_KHR_blend_equation_advanced
                GL_KHR_debug GL_KHR_robustness
                GL_KHR_robust_buffer_access_behavior
                GL_EXT_texture_type_2_10_10_10_REV
                GL_EXT_texture_compression_dxt1 GL_EXT_texture_format_BGRA8888
                GL_EXT_texture_compression_s3tc GL_EXT_read_format_bgra
                GL_EXT_multi_draw_arrays GL_EXT_frag_depth
                GL_EXT_discard_framebuffer GL_EXT_blend_minmax
                GL_EXT_multisampled_render_to_texture
                GL_EXT_color_buffer_half_float GL_EXT_color_buffer_float
                GL_EXT_robustness GL_EXT_texture_sRGB_decode
                GL_EXT_draw_buffers_indexed GL_EXT_texture_border_clamp
                GL_EXT_texture_buffer GL_EXT_texture_cube_map_array
                GL_EXT_multi_draw_indirect GL_EXT_draw_elements_base_vertex
                GL_EXT_texture_rg GL_EXT_protected_textures GL_EXT_sRGB
                GL_VIV_direct_texture
[14:38:00.311]  GL ES 2 renderer features:
                read-back format: BGRA
                wl_shm sub-image to texture: yes
                EGL Wayland extension: yes
[14:38:00.343] warning: no input devices on entering Weston.Possible causes:
           -no permissions to read /dev/input/event*
           -seats misconfigured (Weston backend option 'seat', udev device property ID_SEAT)
[14:38:00.343]  failed to create input devices
[14:38:00.349]  DRM: head 'DP-1' found, connector 56 is connected, EDID make 'DEL', model 'DELL
P2417H', serial 'C9G5D7561ECB'
[14:38:00.349]  Registered plugin API 'weston_drm_output_api_v1' of size 24
[14:38:00.357]  Chosen EGL config details: id: 41 rgba: 8 8 8 0 buf: 24 dep: 0 stcl: 0 int: 1-60 type:
win|pix|pbf|swap_preserved vis_id: XRGB8888 (0x34325258)
[14:38:00.357]  Output DP-1 (crtc 48) video modes:
                1920x1080@60.0, preferred, current, 148.5 MHz
                1600x900@60.0, 108.0 MHz
                1280x1024@75.0, 135.0 MHz
                1280x1024@60.0, 108.0 MHz
                1152x864@75.0, 108.0 MHz
                1024x768@75.0, 78.8 MHz
                1024x768@60.0, 65.0 MHz
                800x600@75.0, 49.5 MHz
                800x600@60.3, 40.0 MHz
                640x480@75.0, 31.5 MHz
                640x480@59.9, 25.2 MHz
                720x400@70.1, 28.3 MHz
[14:38:00.357]  Output 'DP-1' enabled with head(s) DP-1
[14:38:00.357]  Compositor capabilities:
                arbitrary surface rotation: yes
                screen capture uses y-flip: yes
                presentation clock: CLOCK_MONOTONIC, id 1
                presentation clock resolution: 0.000000001 s
[14:38:00.359]  Loading module '/usr/lib/weston/desktop-shell.so'
[14:38:00.367]  launching '/usr/libexec/weston-keyboard'
[14:38:00.373]  launching '/usr/libexec/weston-desktop-shell'
[14:39:23.341]  event0 - Logitech USB Optical Mouse: is tagged by udev as: Mouse
[14:39:23.341]  event0 - Logitech USB Optical Mouse: device is a pointer
[14:39:23.341]  libinput: configuring device "Logitech USB Optical Mouse".
[14:39:23.342]  associating input device event0 with output DP-1 (none by udev)
could not load cursor 'dnd-move'
could not load cursor 'dnd-copy'
could not load cursor 'dnd-none'
[14:39:33.459]  event0  - Logitech USB Optical Mouse: device removed
[14:39:51.794]  event0  - Dell Dell USB Keyboard: is tagged by udev as: Keyboard
```

```
[14:39:51.794] event0 - Dell Dell USB Keyboard: device is a keyboard
[14:39:51.859] libinput: configuring device "Dell Dell USB Keyboard".
[14:39:51.859] associating input device event0 with output DP-1 (none by udev)
[14:40:03.937] event0  - Dell Dell USB Keyboard: device removed
[14:40:11.758] event0 - Logitech USB Optical Mouse: is tagged by udev as: Mouse
[14:40:11.758] event0 - Logitech USB Optical Mouse: device is a pointer
[14:40:11.758] libinput: configuring device "Logitech USB Optical Mouse".
[14:40:11.758] associating input device event0 with output DP-1 (none by udev)
[14:40:19.403] event0  - Logitech USB Optical Mouse: device removed
[14:40:29.454] event0 - Dell Dell USB Keyboard: is tagged by udev as: Keyboard
[14:40:29.454] event0 - Dell Dell USB Keyboard: device is a keyboard
[14:40:29.454] libinput: configuring device "Dell Dell USB Keyboard".
[14:40:29.454] associating input device event0 with output DP-1 (none by udev)
[14:41:00.156] event0  - Dell Dell USB Keyboard: device removed

[14:42:29.287] event0 - Logitech USB Optical Mouse: is tagged by udev as: Mouse
[14:42:29.287] event0 - Logitech USB Optical Mouse: device is a pointer
[14:42:29.287] libinput: configuring device "Logitech USB Optical Mouse".
[14:42:29.287] associating input device event0 with output DP-1 (none by udev)
[14:42:35.418] event1 - Dell Dell USB Keyboard: is tagged by udev as: Keyboard
[14:42:35.419] event1 - Dell Dell USB Keyboard: device is a keyboard
[14:42:35.419] libinput: configuring device "Dell Dell USB Keyboard".
[14:42:35.419] associating input device event1 with output DP-1 (none by udev)
```

下面是快照。

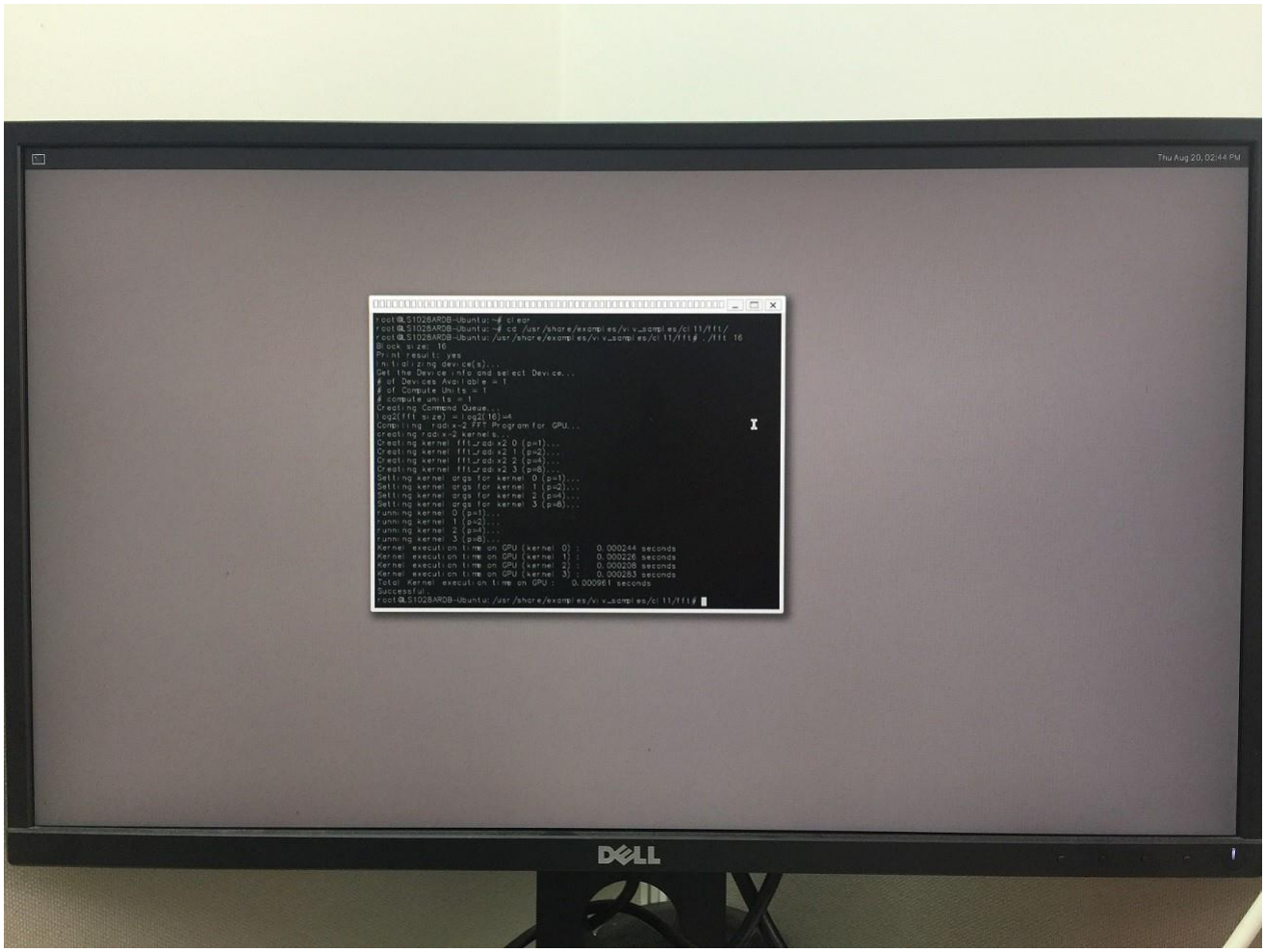


图 56.Weston 演示

5.5.3 CSI 摄像头

恩智浦提供具有 CSI MIPI 和 DSI MIPI 接口的 i.MX 8M Mini EVK 板。`gstreamer` 视频流从 CSI 摄像头捕获视频帧，并通过 DSI MIPI 接口将其显示到屏幕上。元素 `waylandsink` 基于 Wayland 库和 Weston 桌面。请参见 Wayland 和 Weston 以使能它们。用户应按照以下步骤在目标板上使能 `gstreamer`。

1. **软件设置和配置**

   默认使能 `Gstream`。

2. **硬件设置**

   - 对于 i.MX 8M Mini EVK 板，DSI MIPI 和 CSI MIPI 模块连接到板上。
   - MIPI-DSI 转 HDMI 接口：

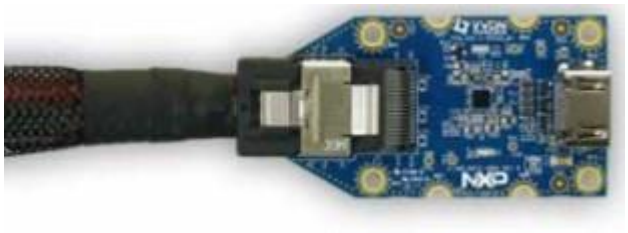


图 57.MIPI-DSI 转 HDMI 接口

3. MIPI-CSI 摄像头模块：下图为 MIPI-CSI 摄像头模块：

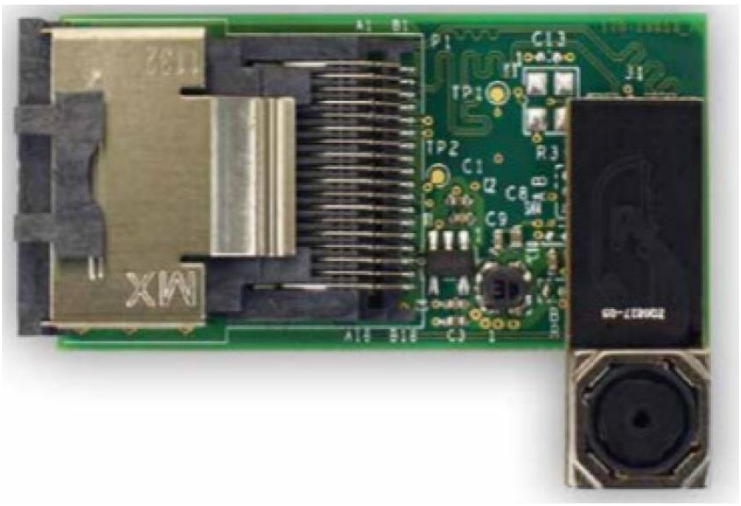


图 58.MIPI-CSI 摄像头模块

4. 为摄像头运行 `gstreamer`。

   进入 Linux 提示符后，运行 `gstreamer` 命令，步骤如下所示：

```
[root@imx8mmevk ~] # gst-launch-1.0 v4l2src device= /dev/video0 ! 'video/x-
raw,width=640,height=480,framerate=(fraction)30/1' ! videoconvert !fbdevsink
```

   下面是 `fbdevsink` 的快照：

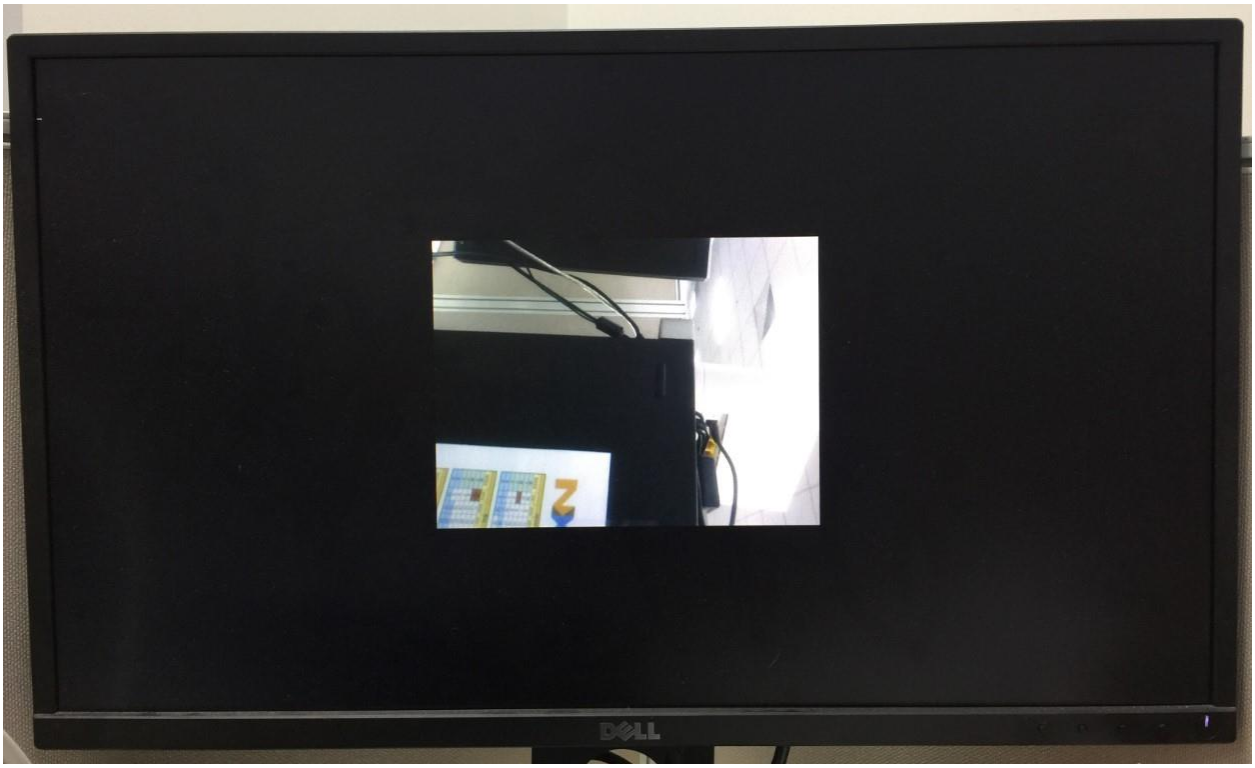


图 59.**fbdevsink**

5. 使用如下所示的命令为 `waylandsink` 运行 `wayland` 和 `weston`：

```
[root@imx8mmevk ~] # mkdir -p /run/user/0/
[root@imx8mmevk ~] # export XDG_RUNTIME_DIR="/run/user/0/"
[root@imx8mmevk ~] # weston --tty=1 &
[root@imx8mmevk ~] # gst-launch-1.0 v4l2src device= /dev/video0 ! 'video/x-
raw,width=640,height=480,framerate=(fraction)30/1' ! videoconvert !waylandsink
```

下面的快照显示了 `waylandsink` 的实施：

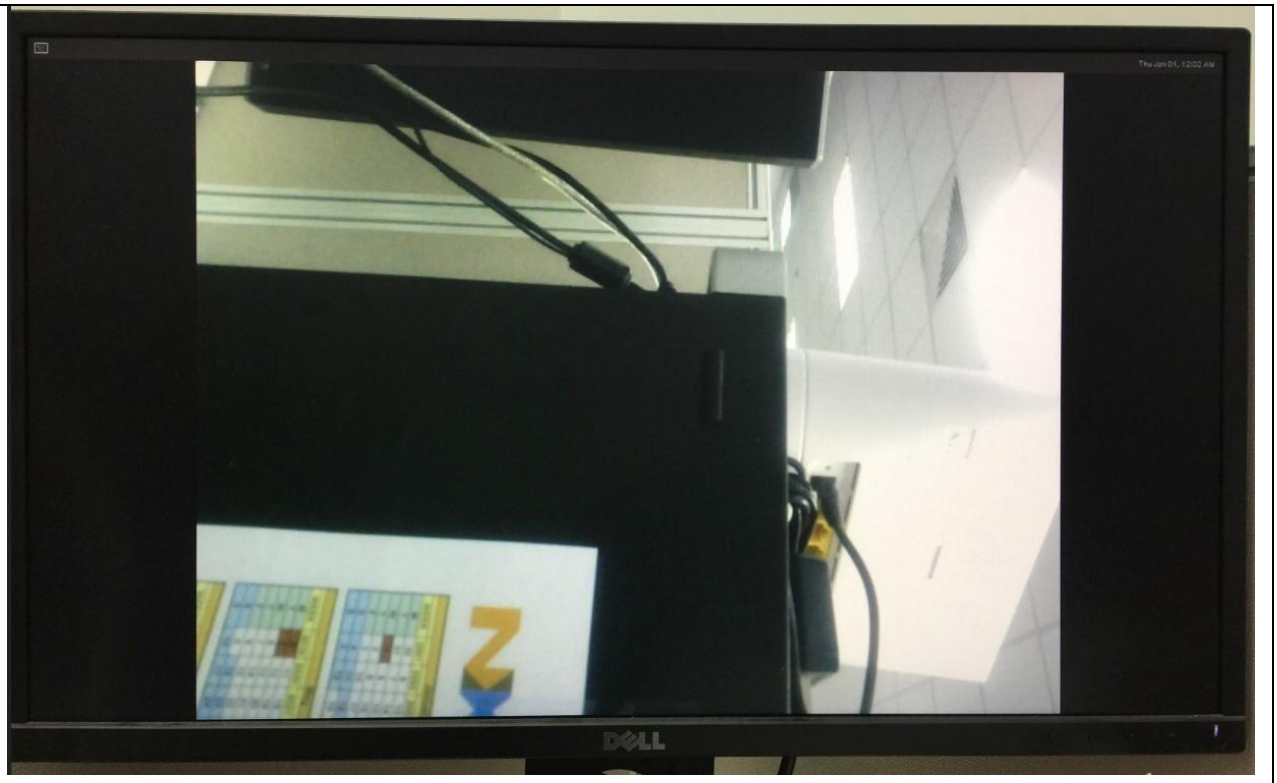


图 60.Waylandsink 实施

5.6 LS1028A 上的无线

5.6.1 NFC

NFC click 板是 mikroBUS™附加板，搭载恩智浦多功能近距离通信控制器——PN7120IC。NFC 设备用于免接触式支付系统、电子票务、智能卡。在零售和广告中，价格低廉的 NFC 标签可以嵌入到包装标签、传单或海报中。

该板完全符合 NFC Forum 规范。这意味着用户可以充分利用 NFC 的全部潜力及其下列三种不同的操作模式：

1. 卡模拟
2. 读/写
3. P2P

5.6.1.1 简介

恩智浦的 PN7120 IC 集成了 Arm ™ Cortex-M0 MCU，可以更轻松地集成到设计中，因为它需要的主机 MCU 资源较少。集成固件提供所有 NFC 协议，用于执行负责调制、数据处理和错误检测的免接触式通信。

该板通过符合 NCI（NFC 控制器接口）1.0 主机协议的 mikroBUS™ I2C 接口与目标板 MCU 通信。RST 和 INT 引脚提供额外功能。该板使用 3.3 V 电源。

5.6.1.2 PN7120 功能

PN7120 IC 内置新一代 RF 免接触式前端，根据 NFCIP-1 和 NFCIP-2、ISO/IEC14443、ISO/IEC 15693、ISO/IEC 18000-3、MIFARE 和 FeliCa 规范，支持各种传输模式。内置的 Arm Cortex-M0 微控制器内核搭载支持 NCI 1.0 主机通信的集成固件。

5.6.1.3 硬件准备

使用以下硬件项目进行 NFC click 板演示设置：

1. LS1028ARDB
2. NFC click 板
3. NFC 样品卡（标签）

注意

用户应当将 NFC click 板插入 LS1028ARDB mikroBUS1 插槽。

5.6.1.4 软件准备

为了支持 NFC click 板，请使用以下步骤：

1. 在实时边缘中，默认使能 libnfc-nci。
2. 在 Linux 内核配置中，确保使能以下选项：

```
[*] Networking support --->
    <M>   NFCsubsystemsupport     --->
        Near Field Communication (NFC) devices--->
            <M> NXP PN5XX based driver
```

注意

基于**恩智浦 PN5XX 的驱动程序**仅支持模块模式。

3. 使用 `make` 命令创建镜像。

5.6.1.5 测试 NFC click 板

使用以下步骤测试 NFC Click 板：

1. 安装 NFC 驱动程序模块

```
[root]# modprobe pn5xx_i2c.ko
```

2. 上述命令成功后，控制台中出现如下日志。在这种情况下可以忽略错误信息。

```
[root@OpenIL:~]# insmod /lib/modules/4.14.47-ipipe/kernel/[  195.547601] random: crng init
done
[  195.551016] random: 5 urandom warning(s) missed due to ratelimiting
[root@OpenIL:~]# insmod /lib/modules/4.14.47-ipipe/kernel/drivers/misc/nxp-pn5xx
/pn5xx_i2c.ko
[  777.503246] pn54x_dev_init
[  777.506048] pn54x_probe
[  777.508523] pn544 7-0028: FIRM GPIO <OPTIONAL> error getting from OF node
[  777.515344] pn544 7-0028: CLKREQ GPIO <OPTIONAL> error getting from OF node
[  777.522347] pn544 7-0028: 7-0028 supply nxp,pn54x-pvdd not found, using dummy regulator
[  777.530424] pn544 7-0028: 7-0028 supply nxp,pn54x-vbat not found, using dummy regulator
[  777.538490] pn544 7-0028: 7-0028 supply nxp,pn54x-pmuvcc not found, using dummy regulato
r
[  777.546723] pn544 7-0028: 7-0028 supply nxp,pn54x-sevdd not found, using dummy regulator
```

3. 运行 **nfcDemoApp** 应用程序：

```
[root]# nfcDemoApp poll
```

```
[root@OpenIL:~]# nfcDemoApp poll
##############################################################################
##                                    NFC demo                                ##
##############################################################################
##                                Poll mode activated                         [ 1251.20807
1] pn54x_dev_open : 10,55
        ##
####[ 1251.212807] pn54x_dev_ioctl, cmd=1074063617, arg=1
################[ 1251.219006] pn544_enable power on
##############################################################################
                              ... press enter to quit ...

[ 1251.431597] pn54x_dev_ioctl, cmd=1074063617, arg=0
[ 1251.436416] pn544_disable power off
[ 1251.647586] pn54x_dev_ioctl, cmd=1074063617, arg=1
[ 1251.652401] pn544_enable power on
NfcHcpX:8103
NfcHcpR:8180
NfcHcpX:810103020304
NfcHcpR:8180
NfcHcpX:81010143da67663bda6766
NfcHcpR:8180
NfcHcpX:810204
NfcHcpR:818000
Waiting for a Tag/Device...
```

4. 将 NFC 样品卡（标签）放在 NFC click 板的顶部：

```
Waiting for a Tag/Device...

        NFC Tag Found

        Type :          'Type A - Mifare Ul'
        NFCID1 :         '04 67 66 D2 9C 39 81 '
                Record Found :
                                NDEF Content Max size :      '868 bytes'
                                NDEF Actual Content size :      '29 bytes'
                                ReadOnly :                       'FALSE'
                Read NDEF URL Error

                29 bytes of NDEF data received :
                D1 01 19 55 01 6E 78 70 2E 63 6F 6D 2F 64 65 6D 6F 62 6F 61 72 64 2F 4F 4D
35 35 37 38

        NFC Tag Lost

Waiting for a Tag/Device...
```

显示以上信息表示读卡成功。

5.6.2 低功耗蓝牙

本章介绍了低功耗蓝牙 P click 板的功能以及如何在恩智浦 LS1028A 参考设计板(RDB)上使用它。

5.6.2.1 简介

低功耗蓝牙 P click 搭载 nRF8001 IC，允许用户将蓝牙 4.0 添加到用户的设备。click 通过 mikroBUS™ SPI（CS、SCK、MISO、MOSI）、RDY 和 ACT 线路与目标板 MCU 通信，并使用 3.3 V 电源运行。

低功耗蓝牙 P click 具有 PCB 走线天线，专为 2400 MHz 至 2483.5 MHz 频段而设计。在开放空间中，设备的最大覆盖范围可达 40 米。

5.6.2.2 低功耗蓝牙

LS1028ARDB 支持低功耗蓝牙 click 板，低功耗蓝牙 P click 带有 nRF8001 IC，允许用户将蓝牙 4.0 添加到设备中。

5.6.2.3 功能

以下是 BLE P click 板提供的功能：

- nRF8001 低功耗蓝牙 RF 收发器
  - 16 MHz 晶体振荡器
  - 超低峰值电流消耗 <14 mA
  - 面向连接的配置文件的低电流，通常为 2 µA
- PCB 走线天线（2400-2483.5 MHz，最长 40 米）
- BLE Android 应用
- 接口：SPI（CS、SCK、MISO、MOSI）、RDY 和 ACT 线路
- 3.3 V 电源

5.6.2.4 硬件准备

使用以下硬件项目进行 BLE P click 板演示设置：

1. LS1028ARDB
2. BLE P Click 板
3. Android 手机（可选）

   下图显示了演示所需的硬件设置：

图 61.BLE P click 板硬件设置

5.6.2.5 软件准备

使用以下步骤进行 BLE P click 板演示软件设置：

- 使用以下链接下载 JUMA UART（Android 应用）：https://apkpure.com/juma-uart/com.juma.UART
- 然后，运行以下步骤以支持 BLE P click 板：
  1. 在实时边缘中，默认使能 libblep。
  2. 在 Linux 内核配置中，确保使能以下选项：

```
Device Drivers --->
    SPI support --->
        <*>   Freescale DSPIcontroller
        <*>   User mode SPI device driversupport
```

3. 使用 `make` 命令创建镜像。

5.6.2.6 测试 BLE P click 板

使用以下步骤测试 BLE P click 板：

1. **运行 blep_demo 应用程序。**

   显示如下日志，表明 BLE P click 板已初始化。之后，用户可以使用手机或电脑的蓝牙设备扫描 BLE P click 板。使用的 BLE P click 板的名称是“**MikroE**”。

```
root@OpenIL-Ubuntu:~# blep_demo
spi mode: 0x0
bits per word: 8
max speed: 500000 Hz (500 KHz)
Please input a command!
 Event device started: Setup
Error:no
Start setup command
.............................
Setup complete
 Event device started: Standby
 Advertising started : Tap Connect on the nRF UART app
Error:no
Send broadcast command successfully
```

2. **连接日志**

   通过移动应用连接 BLE P click 板。成功连接后，将显示以下日志。此后，应用程序可以与 BLE P click 板通信。

```
Evt Connected
Evt Pipe Status
Evt link connection interval changed
ConnectionInterval:0x0006
SlaveLatency:0x0000
SupervisionTimeout:0x01F4
Evt Pipe Status
Evt link connection interval changed
ConnectionInterval:0x0027
SlaveLatency:0x0000
SupervisionTimeout:0x01F4
```

3. **断开日志**

   单击 Android 应用的“断开”按钮，断开与 BLE P click 板的连接。以下日志显示断开成功：

```
Evt Disconnected
Advertising started : Tap Connect on the nRF UART app Send broadcast command successfully
```

4. **命令行简介**

blep _demo 应用程序支持四个命令行：**devaddr**、**name=**、**version** 和 **echo**。

a. **devaddr**

该命令用于获取 BLE P click 板的 MAC 地址。用户可以随时运行此命令。

```
devaddr
Please input a command!
Device address:DC:E2:6C:17:07:45
```

b. **name=**

该命令用于在广播时设置 BLE P click 板的蓝牙名称。等号“=”后面不需要空格，“等号”后面的内容是设置的名称。最大长度为 16 个字符。

```
name=ble_demo
Name set. New name: ble_demo, 8
Please input a command!
Unknow event:0x00
Set local data successfully
```

c. **version**

该命令用于获取 BLE P click 板的版本。用户可以随时运行此命令。

```
version
Please input a command!
Unknow event:0x00
Device version
 Configuration ID:0x41
 ACI protocol version:2
 Current setup format:3
 Setup ID:0x00
 Configuration status:open(VM)
```

d. **echo**

该命令用于向 Android 应用发送字符串。该命令应在连接建立后执行。最大长度为 20 个字符。

以下日志显示在未建立连接时用户尝试发送字符串后显示的消息：

```
echo hi
Please input a command!
Unknow event:0x00
 ACI Evt Pipe Error: Pipe #9
 Pipe Error Code: 0x83
 Pipe Error Data: 0x00
 Please connect the device before sending data
```

当用户在建立连接后发送字符串时，将显示以下日志：

```
echo hello,world!
Please input a command!
Unknow event:0x00
The number of data command buffer is 1
```

5. 接收数据

   当 Android 应用发送字符串时：

```
DataReceivedEvent: hi.yugxdr
```

5.6.3 BEE

本章介绍 BEE Click 板的功能以及如何在 LS1028ARDB 上使用它。

5.6.3.1 BEE/ZigBEE

LS1028ARDB 支持 BEE click 板，可以实施微芯的 MRF24J40MA 2.4 GHz IEEE 802.15.4 无线电收发器模块。

5.6.3.2 简介

BEE Click 板采用微芯的 MRF24J40MA 2.4 GHz IEEE 802.15.4 无线电收发器模块。Click 设计为仅在 3.3 V 电源上运行。它通过 SPI 接口与目标控制器通信。

5.6.3.3 功能

BEE Click 板的功能如下：

- PCB 天线
- MRF24J40MA 模块
- 低电流消耗（TX 23 mA、RX 19 mA、睡眠 2 μA）
- ZigBee 协议栈
- MiWi™ 协议栈
- SPI 接口
- 3.3 V 电源

5.6.3.4 硬件准备

使用以下硬件项目进行 BEE Click 板演示设置：

- 两块 LS1028ARDB 板
- 两块 BEE Click 板

下图显示了 BEE Click 板的硬件设置。

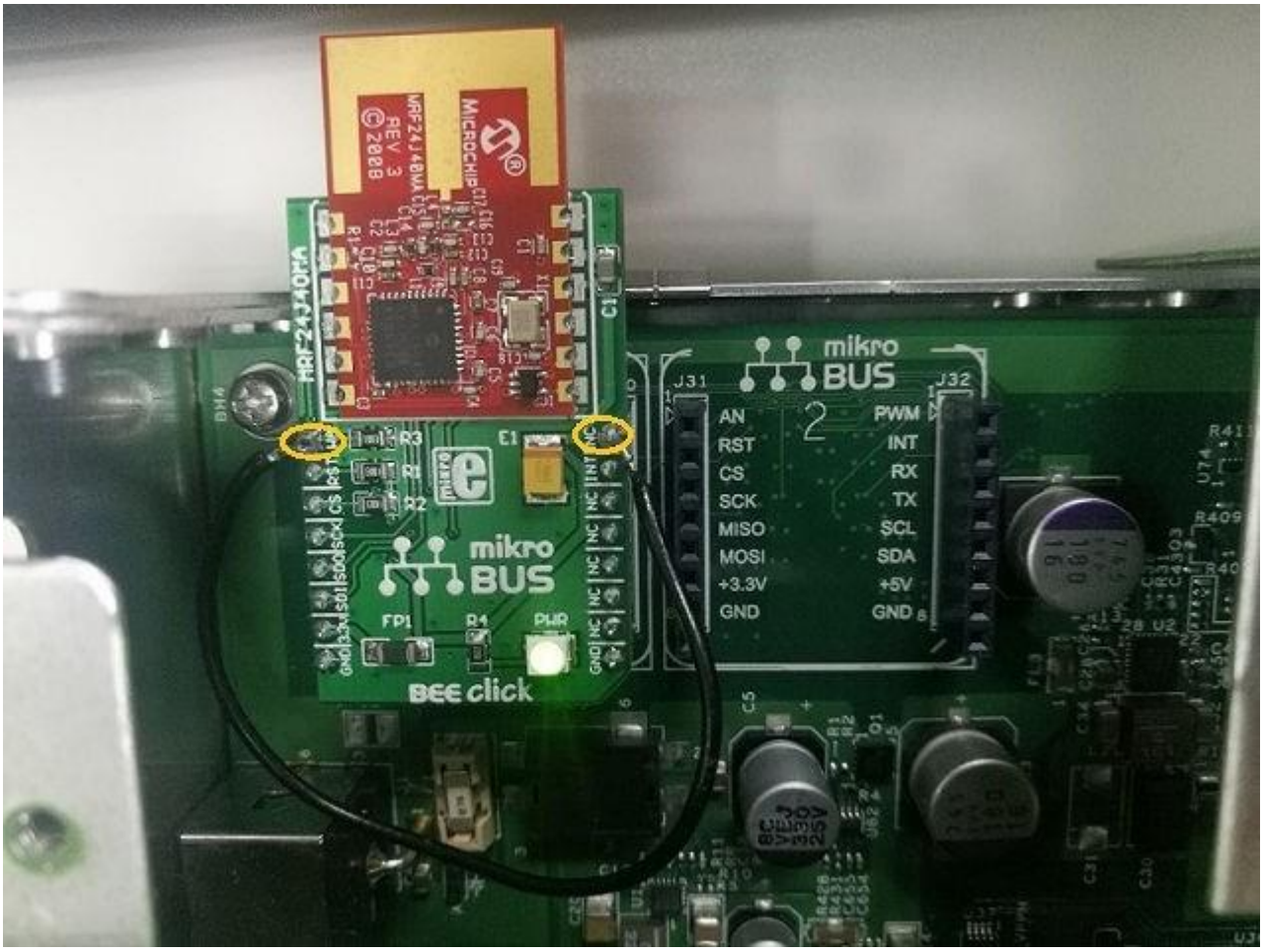


图 62.BEE Click 板硬件设置

注意

BEE Click 板的 WA 引脚与 NC 引脚相连。

5.6.3.5 软件准备

为了支持 BEE click 板，请使用以下步骤：

1. 在实时边缘中，默认使能 libbee。
2. 在 Linux 内核配置中，确保使能以下选项：

```
Device Drivers --->
  SPI support --->
    <*>   Freescale DSPIcontroller
    <*> User mode SPI device driver support
  -*- GPIO Support--->
    [*]   /sys/class/gpio/...(sysfsinterface)
          Memory mapped GPIO drivers--->
            [*] MPC512x/MPC8xxx/QorIQ GPIO support
```

3. 使用 `make` 命令创建镜像。

5.6.3.6 测试 BEE click 板

测试应用程序 bee_demo 是使用 BEE Click 板库创建而成。此应用程序可以在两块 BEE Click 板之间传输文件。

1. 用户应当在任何路径中创建文件。例如，./samples/test.txt。
2. 首先，通过运行以下命令启动服务器节点：

```
bee_demo -s -f=XXX
```

命令参数如下：

- **-s**：该设备节点充当服务器。
- **-f=XXX**：该参数仅在服务器节点上有效。XXX 是要传输的文件路径（相对或绝对）。

```
[root]# ls
samples
[root]# bee_demo -s -f=./samples/test.txt
spi mode: 0x0
bits per word: 8
max speed: 500000 Hz (500 KHz)
BEE  Click  BoardDemo.
This node is a server node.
Waiting for a client
Reading the content of the file
```

3. 通过运行命令 bee_demo -c 在另一个 LS1028ARDB 上启动客户端节点。在上述命令中，参数-c 表示该设备节点充当客户端。客户端节点收到文件后自动退出。接收到的文件保存在当前路径中。

```
[root]# ls
samples
[root]# bee_demo -c
                    spi mode: 0x0
                    bits per word: 8
                    max speed: 500000 Hz (500 KHz)
                    BEE  Click  Board Demo.
                    This node is a client node.
                    Starting to get a file
                    Send the SEQ_REQ command.
                    Send the SEQ_START command.
                    Send the SEQ_START command.
[root]# ls
samples test.txt
[root]#
```

4. 以下日志表明服务器节点已完成发送文件。

```
Send the SEQ_INFO command.
Start to send the file
It's completed to send a file.
```

第 6 章
修订记录

下表总结了本文档的修订。

表 62.文档修订记录

日期	文档版本	主题交叉引用	更改说明
2021 年 7 月 29 日	2.0	-	• 引入了“实时边缘处理软件”而不是“开放式工业 Linux”，具有实时功能支持。 • 重新排列文档结构以包括章节：实时系统、实时网络服务和协议。
		-	添加了对使用 Yocto Project 构建环境来构建实时边缘镜像的支持。*实时边缘 Yocto Project 用户指南*中提供了详细信息。
		新内容	添加了本节。
		-	文档中纳入了裸机框架。
2021 年 4 月 26 日	1.11	新内容	添加了介绍每个版本新功能的章节。
		-	更新了“获取 Open IL”一节。
		-	删除了整个文档中对 Edgescale、OP-TEE、OTA 的引用以及其他次要更新。
2020 年 12 月 22 日	1.10	GenAVB/TSN 协议栈	添加了本章及相关内容。
		-	添加了“摄像头”一节和相关详细信息。
		-	添加了 i.MX 8M Plus EVK 板详细信息的主机设置。
2020 年 9 月 15 日	1.9	IEEE 1588/802.1AS	添加了本节。
		GPU	添加了本章及相关说明。
		Wayland 和 Weston	添加了本章及相关说明。
		real-time-edge-servo 协议栈	使其成为“EtherCAT”一章的一部分。
2020 年 5 月 29 日	1.8	Preempt-RT Linux	在协议中添加了本节。
		-	更新了 LS1028ARDB 的“Linux 中的接口命名”一节。
		-	更新了“Open IL 的主机系统要求”一节。
		-	更新了“运行 Selinux 演示”一节。
2020 年 2 月 20 日	1.7.1	操作示例	更新了本节。
2020 年 1 月 17 日	1.7	real-time-edge-servo 协议栈	添加了本章（恩智浦伺服器）。
		IEEE 1588/802.1AS	添加了本章
		获取 Open IL	更新了本节。

表格接下页……

表 62.文档修订记录（续）

日期	文档版本	主题交叉引用	更改说明
		NETCONF/YANG	其他更新。
2019 年 8 月 31 日	1.6	LS1028A 上的 TSN	• 从 ENETC 上的 TSN 配置和 Felix 交换机上的 TSN 配置一节中删除了与 pcpmap 命令相关的信息。 • 对于少数 tsntool 命令，端口名称“eno/swp0”更改为“swp0”。 • 在流识别一节中添加了有关使用 `nulltagged` 和 `streamhandle` 参数的注释。 • 添加了 TSN 流识别一节。 • 其他少量更新。
		-	更新了表格“主机系统强制包”。添加了 `autogen autoconf libtool` 和 `pkg-config` 包。
		BEE	添加了本章。
		-	更新了 NETCONF/YANG。
		NETCONF/YANG	• 添加了使能 NETCONF 功能一节和其他更新。
2019 年 5 月 1 日	1.5	-	添加了介绍 LS1028ARDB 的 U-Boot 和 Linux 接口命名的章节。
		LS1028A 上的 TSN	更新了恩智浦平台上的时间敏感网络服务(TSN)一章中的此节。
		低功耗蓝牙	添加了本章。
		-	添加了“EdgeScale 客户端”一章。
		-	更新了“获取 Open IL”一节中的 OpenIL 版本和 Git 标记。
2019 年 2 月 1 日	1.4	支持的恩智浦平台	添加了对 LS1028ARDB（64 位和 Ubuntu）的支持。相应地更新了各节。
		-	更新了“获取 Open IL”一节中的 OpenIL 版本和 Git 标记。
		-	为 LS1028ARDB 支持添加了本节。
		恩智浦平台上的时间敏感网络服务(TSN)	重新组织了本章并为 LS1028A 上的 TSN 添加了单独的章节。
		NFC	添加了本章。
		FlexCAN 和 CANOpen	本章中的少量更新。还添加了 LS1028ARDB 的硬件准备和测试 CAN 总线章节。
		-	添加了 QT 一章。

表格接下页……

表 62.文档修订记录（续）

日期	文档版本	主题交叉引用	更改说明
2018 年 10 月 15 日	1.3.1	-	更新了“获取 Open IL”一节中的 OpenIL 版本和 Git 标记。
2018 年 8 月 31 日	1.3	EtherCAT	添加了本章。
		FlexCAN	添加了本章。
		i.MX6QSabreSD 支持。	在恩智浦 OpenIL 平台一章中添加了本节。更新了 i.MX6Q Sabre 支持的其他几节。
		获取 Open IL	更新了本节。
		Selinux 演示	添加了#unique_362 一节并更新了基本设置。其他几节的更新。
2018 年 5 月 31 日	1.2	-	更新了 RTnet 的“硬件要求”一节。
		-	更新了 RTnet 的“软件要求”一节。
2018 年 4 月 18 日	1.1.1	-	添加了“RTnet”一节。
		-	添加了 LS1043A 交换机设置的注释。
2018 年 3 月 30 日	1.1	-	本节添加了对工业物联网裸机框架的支持。
		-	添加了启动板之前要执行的步骤的注释。
		相关文档	添加了本节。
2017 年 12 月 22 日	1.0	OPC UA	添加了本章。
		恩智浦平台上的时间敏感网络服务(TSN)	“1-板 TSN 演示”和“3-板 TSN 演示”两章替换为“TSN 演示”一章。
		协议	• 更新了“工业功能”一节。 • -IEEE 1588 -'sja1105-ptp'支持被移除。
2017 年 8 月 25 日	0.3	-	设置 OpenIL 网站 http://www.openil.org/。
		-	OTA——添加了 Xenomai Cobalt 64 位和 SJA1105 支持。
		恩智浦平台上的时间敏感网络服务(TSN)	添加了 Qbv 支持。
		-	添加了对 LS1043 / LS1046 Ubuntu 用户区的 SELinux 支持。
		-	添加了对 LS1021ATSN 平台的 OP-TEE 支持。
		-	4G LTE 模块——添加了对 LS1043ARDB、LS1046ARDB 和 LS1012ARDB 的 64 位支持。
		实时网络服务	添加了对 64 位 LS1043ARDB 和 64 位 LS1046ARDB 的 Ubuntu 用户区支持。
2017 年 5 月 26 日	0.2	-	初版。

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Limited warranty and liability — Information in this document is provided solely to enable system and software implementers to use NXP products.There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages.“Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time.All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts.NXP does not convey any license under its patent rights nor the rights of others.NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice.This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities.Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products.Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications.NXP accepts no liability for any vulnerability.Customer should regularly check security updates from NXP and follow up appropriately.Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX,EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE, VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners.AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.The related technology may be protected by any or all of patents, copyrights, designs and trade secrets.All rights reserved.Oracle and Java are registered trademarks of Oracle and/or its affiliates.The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.M, M Mobileye and other Mobileye trademarks or logos appearing herein are trademarks of Mobileye Vision Technologies Ltd. in the United States, the EU and/or other jurisdictions.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 29-Jul-2021

Document identifier: REALTIMEEDGEUG

arm