Application Note

基于 TI 可编程实时单元(PRU)的高速(12Mb)串口通信

李彪, 唐超伦 *China Central FAE*

摘要

串行通信接口（UART），以其结构简单、抗干扰能力强以及成本低廉，至今仍广泛应用于各种工业，医疗应用。而且有些应用场景要求支持高达 12Mbps, 甚至 16Mbps，基于通用处理器（MPU）配合硬件 UART 在高波特率串口通信中，由于操作系统（如 Linux）的线程调度延迟、中断响应延时不确定性以及多任务并发，容易导致数据缓冲区溢出，引发高丢包率，难以满足高可靠性要求的工业应用场景。

针对上述瓶颈，本文提出并实现了一种基于 TI Sitara 处理器可编程实时单元（PRU）的高速串口通信方案。最高可实现 12Mbps 的高速串口通信，该方案利用 PRU 完全独立且具备确定性执行周期的单周期指令集硬件架构，直接控制 PRU 的 IO 模拟串口通信流程。摆脱对通用处理器中断机制的依赖，在 PRU 与 ARM 核之间构建共享内存数据交互通道，消除了操作系统带来的实时性抖动的影响，实现 12Mbps 高波特率下串口数据的高效、无丢包的传输，并且显著降低了 ARM 主核的 CPU 负载。该设计使用 TI 处理器内部的可 PRU 核，无需增加额外的成本，为客户的实际工业应用与高速串口的通信需求提供了一种低成本、高可靠性的解决方案。

内容

1 引言 2
1.1 TI PRU 介绍 2
1.2 高速串口应用背景与工业需求 3
1.3 12Mbps 高速串口技术挑战 3
2 基于 PRU 的高速串口通信系统设计与实现 5
2.1 串口底层物理层设计 5
2.2 PRU 软件利用 PEFIF 接口实现多路高速串口 6
2.3 串口 ARM Linux 侧应用层驱动设计 7
3 高速串口实验验证 8
3.1 高速 PRU 串口裸机实验 8
3.2 基于 Linux 应用高速 PRU 串口 8
4 总结 10
5 参考文献 10

插图清单

图 1-1. AM62x PRU-ICSS 功能框图 2
图 2-1. PRU UART 数据链路层硬件实现设计 6
图 3-1. 基于 AM62 SK EVM 硬件环境搭建 8
图 3-2. 基于 AM62 SK EVM 软件环境搭建 8
图 3-3. Linux 随机串口数据波形 9

表格清单

表 1-1. Sitara AM6x 系列支持串口数量 3

商标

所有商标均为其各自所有者的财产。

1 引言

1.1 TI PRU 介绍

TI SOC 上的 PRU(Programmable Real-Time Unit) 的框图如 图 1-1 所示，PRU 是一个专门为了硬实时、低延迟控制而设计的 RISC 内核。它通过无流水线单周期执行机制以及引脚直接映射架构，在物理边界上保证了时序的确定性。

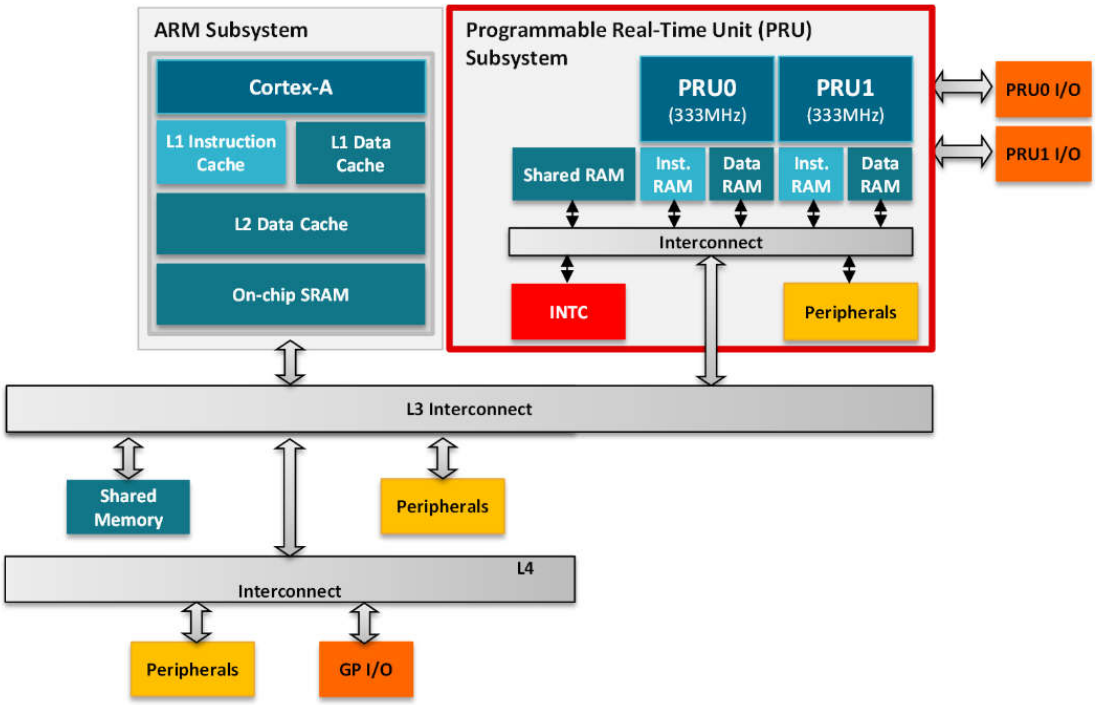

图 1-1. AM62x PRU-ICSS 功能框图

在实际的工业应用中，系统经常需要与不使用标准协议的外部设备进行交互。对于特殊时序接口，通过 SOC 可能不具备直接连接的方式，传统的做法是使用 FPGA 来实现 SOC 和特殊时序接口外设的高带宽通信，Sitara 处理器内部包含可编程实时单元 (PRU) 内核，在这种情况下无需在设计中添加 FPGA、ASIC、CPLD 等器件。另一种方式则是使用微控制器 (MCU) 的 GPIO 进行模拟，然而，使用 MCU 内核来操作 GPIO 引脚存在一些根本性的缺陷，与 FPGA 相比，标准 MCU 在与 GPIO 交互时将具有更高的延迟和更低的确定性。使用软件来控制 GPIO 的翻转，由于内部总线的延时，可以支持的频率和精度将极大受限，无法满足占空比控制和时序控制的要求。

PRU 系统包含 32 个 32 位寄存器，其中两个寄存器专用于 GPIO 功能，一个寄存器直接连接到通用输出 (GPO) 信号，通过写入该寄存器来切换 GPO 信号，另一个寄存器直接连接到通用输入 (GPI) 信号，通过读取该寄存器来读取 GPI 信号。通过这种设计，从 PRU 到实际的 GPIO 直接连接，输入输出信号的性能也都得到了提升。PRU 具有暂停程序运行直至 GPI 事件发生的指令，这使得在检测到输入触发信号后，能够以最佳延迟产生输出响应，PRU 能够实现单个输入和输出之间，以及多个输入信号或多个输出信号之间完全确定的时序关系。高确定性的物理层通信需求与上层复杂的非实时操作系统解耦，在 333MHz 主频下，将单周期 GPIO 的执行精度控制在了 3ns，中断响应周期可以控制在 6ns。并且 PRU 核心集成了创新的宽边接口（Broadside Interface），该接口打破了传统通用 CPU 逐字（Word-by-Word）读取的总线瓶颈，允许 PRU 在单个时钟周期内完成与本地硬件加速器之间高达 124 字节的大容量数据并行吞吐。此外，每个 PRU 核心均配有 2kB 的专用宽边专用内存（Broadside RAM）。该内存挂载于相同的宽边专用总线上，不仅具备与共享内存同等的访问速度，更通过物理隔离保障了多核并发时的零总线冲突。

以上介绍适用于所有带有 PRU 子系统的 Sitara 处理器，包括 AM243x、AM263x、AM335x、AM437x、AM57x、AM62x、AM64x 和 AM65x。TI 处理器中包含多种 PRU 子系统如 PRU-ICSS，PRU-ICSSG，PRUSS 等，本文主要针对 AM625 系列 PRUSS 进行研究和讨论，每个 PRU 子系统具体的差异可以参见以下文章: PRU-

ICSS Feature Comparison (Rev. H) 。本文基于 PRUSS 实现高速串口通讯功能，理论上可以直接迁移到其他 PRU 子系统。

1.2 高速串口应用背景与工业需求

随着现代工业对生产精度和监控频率要求的提升，串口通信正向着高频、大吞吐量、兆比特（Mbps）级的方向演进。这个趋势往往需要以数兆比特的速率不间断地向主控单元推送数据。这种高速串口通信对底层控制系统的实时响应能力和吞吐性能提出了更高的要求。在现有的工业嵌入式设计中，主流方案通常采用通用处理器（MPU，如 ARM Cortex-A 系列）挂载 Linux 等通用操作系统，并利用芯片内置的硬件 UART 控制器来处理串口数据。这种"ARM + Linux + 硬件 UART" 的传统架构在低速通信时表现良好，但在 Mbps 级的高速场景下，其任务调度器、虚拟内存管理和复杂的网络协议栈会引入不可预测的延时抖动（Jitter）。当高速串口产生高频中断时，Linux 操作系统的线程调度延迟往往在数十微秒到数百微秒不等。在 Mbps 级的波特率下，传输一个字节的时间仅为几微秒或几百纳秒。尽管硬件 UART 芯片带有 16 字节或 64 字节的 FIFO（先进先出缓存），但在高速通信的情况下，FIFO 极易在几微秒内被填满。如果无法及时处理，极容易导致溢出错误和丢包，为了防止丢包，系统不得不耗费大量算力进行串口数据的处理。

在 TI 的处理器上均可以支持多个串口通信，表 1-1 列举了 TI 处理器分别能支持的串口数量和最高速度，以 AM62x 为例，芯片原生可以支持高达 9 个串口，传统 Linux 加上串口硬件的限制，TI Sitara 系列处理器软件上可以支持的串口速度的上限在 3Mbps。即使 TI 处理器的串口可以支持 DMA 模式等优化的措施，可以减少串口的接收和发送时 CPU 占用，但使用传统方案实际测试仍然无法在不影响系统其他任务（大幅增加 CPU 负载）的情况下满足超过 3Mbps 的高吞吐需求。因此本文提出使用 PRU 来模拟串口，以实现更高速（12Mbps）的串口，并且不大幅增加 CPU 的负载。

表 1-1. Sitara AM6x 系列支持串口数量

Domain	AM62x	AM62L	AM62P	AM64x	AM62A	Max Speed	Uart IP
WKUP	1	1	1	0	1	3Mbps	16C750
MCU	1	0	1	2	1	3Mbps	16C750
Main	7	7	7	7	7	3Mbps	16C750
Total	9	8	9	9	9	3Mbps	16C750

1.3 12Mbps 高速串口技术挑战

使用 ARM 芯片实现 12Mbps 超高波特率串口通信时，传统方案的 CPU 的处理能力面临极大的挑战。在 12 Mbps 的波特率下，每个数据位的传输时间约为 83.3 纳秒。由于标准串口（UART）通信一个字节通常包含 1 个起始位、8 个数据位和 1 个停止位（共 10 个位），因此每隔大约 0.83 微秒（即 833 纳秒）就会传输完一个完整的字节。这意味着，如果是用传统的接收中断机制，CPU 每秒需要响应高达 120 万次中断（1.2 MHz 的中断频率），这在嵌入式 CPU 的应用中是完全无法接受和实现的。每次中断触发时，CPU 都需要执行保存现场、跳转到中断服务程序、读取寄存器、恢复现场等一系列操作。

为了让 CPU 在如此高的数据吞吐量下存活并正常工作，设计者必须彻底剥离 CPU 对底层数据搬运的干预，转而依赖 DMA（直接内存访问）控制器。在 DMA 模式下，串口外设在接收或发送数据时，直接通过 DMA 总线将数据搬运到内存中，整个过程无需 CPU 介入。但并且如何设计 DMA 的长度也是一个问题，设置的过大，会导致串口数据严重滞后，但过小将无法实质性减少中断数量，目前 TI SDK 中串口的 DMA 模式为了保证数据的实时性，设置的单字节搬运，因此目前产生中断的数量过多的问题没有得到实质性的解决。

硬件上，在 12Mbps 下，每个数据位的时间仅约 83 纳秒，标准串口所用的 TTL 或 CMOS 电平引脚会受到寄生电容和电感带来的严重影响，导致信号上升沿和下降沿变缓、波形失真。因此 12Mbps 的串口仅适合 PCB 内，较短走线的通信，跨板通信需要考虑信号完整性带来影响。

针对以上通用处理器在高速实时通信中的先天不足，无法支持高速的串口通信的瓶颈，Sitara 系列处理器中的可编程实时单元（PRU）可以作为一个可行方案。PRU 通过无流水线单周期执行机制以及引脚直接映射架构，在物理边界上可以保证了时序的绝对确定性。这种高确定性的物理层通信需求与上层复杂的非实时操作系统可以进行解耦， PRU 可以通过编写确定性的汇编或 C 语言固件，通过对时钟周期计数来精确模拟 UART 的时序逻辑。

发送（TX）实现：当检测到发送队列有数据时，固件拉低 TX 引脚（起始位），随后根据设定的波特率计算出每位所需的时钟周期数。固件通过循环或递减计数器精确延时，依次将数据位（Bit 0~7）、校验位、停止位写入 R30。

接收（RX）实现：固件通过轮询 R31 或触发引脚下沿中断来捕获起始位。一旦捕获，PRU 会延迟 1.5 个位周期（跳过起始位并定位到第一个数据位的正中间），以最稳定的采样点读取数据，后续每隔 1 个位周期采样一次，从而确保高吞吐量下的极低误码率。

2 基于 PRU 的高速串口通信系统设计与实现

2.1 串口底层物理层设计

PRU UART 外设是基于工业标准的 TL16C550 异步通信协议，而该协议本身是 TL16C450 的功能升级版本。可以支持 FIFO，并且可以支持 DMA 模式。接收和发送 FIFO 均可存储高达 16 字节的数据，PRU UART 的主要任务是对从外设接收的数据进行串行至并行（串并）转换，并对从 CPU 接收的数据进行并行至串行（并串）转换。对于 PRU 子系统来说，最大 16bytes 的 FIFO，由于 PRU 的时序确定性的特性，因此对 PRU 串口来说完全足够，并且可以根据实际的应用场景进行灵活调整，因此这里选择 16C550 作为参考来实现 PRU 串口，这样可以尽可能的减小资源的消耗。PRU 高速串口是专为工业实时通信设计，为了满足以串口为基础的工业实时通信如 Modbus，Profibus 等协议，如 Profibus-DP 属于高速总线（速率可达 12Mbps），它采用主从轮询和令牌机制。由于线路采用半双工（RS-485），从站收到主站命令后，必须在极短且固定的时间（Tsl，槽时间）内，将物理方向从"接收"切换为"发送"并回传数据，此时如果 FIFO 里残留数据，驱动很难判断最后一个比特何时真正离开芯片发送器。小 FIFO 设计可以让 PRU 能在最后一个字节发送完毕的瞬间，微秒级关闭 RS-485 的 DE（发送使能）并打开 RE（接收使能）。如果切换速度稍慢了，就会错过主站的下一个总线高电平，导致通信失败。

串口通信的最高速率主要受信号质量和软件架构的限制，采用 PRU 可以打破这个限制，实现 12Mbps 的高速传输。在发送协议部分，设计包含一个发送保持寄存器（THR）和一个发送移位寄存器（TSR）。其中，THR 映射在内存中，具体对应为 UART_RBR_TBR[17-8] 寄存器的 TBR_DATA 位域，而 TSR 则未映射到内存空间。当 PRU 串口处于 FIFO 模式时，THR 作为一个 16 字节的 FIFO 缓存器工作。发送器部分的控制功能由 PRU 串口的线路控制寄存器 UART_LCTR 实现。实现的 PRU 串口可以根据该寄存器中所选择的设置，PRU 串口发送器向接收端设备发送以下内容，发送数据格式：

1 START bit + data bits (5, 6, 7, 8) + 1 PARITY bit (optional) + STOP bit (1, 1.5, 2)

THR 从内部数据总线接收数据，当 TSR（发送移位寄存器）就绪时，PRU 串口将数据从 THR 移入 TSR。随后，PRU 串口对 TSR 中的数据进行串行化处理，并通过 UART*_TXD 引脚将数据发射出去。

在非 FIFO 模式下，如果 THR 为空，且中断使能寄存器（UART_INT_EN[1] ETBEI）中的"发送保持寄存器空（THRE）"中断已使能，系统将产生一个中断。当有字符加载到 THR 中，或者读取中断识别寄存器的 UART_INT_FIFO 位域 [3-1] IIR_INTID 时，该中断将被清除。在 FIFO 模式下，当发送 FIFO 为空时产生中断；当至少有 1 个字节加载到 FIFO 中，或者读取 UART_INT_FIFO[3-1] IIR_INTID 位域时，该中断将被清除。

在接收协议部分，设计包含一个接收器移位寄存器（RSR）和一个接收器缓冲寄存器（RBR）。其中，RSR 未映射到内存空间；而 RBR 映射到了内存，具体对应为 UART_RBR_TBR[7-0] 寄存器的 RBR_DATA 位域。当 PRU 串口处于 FIFO 模式时，RBR 作为一个 16 字节的 FIFO 缓存器工作。接收器部分的控制功能由 PRU 串口的线路控制寄存器 UART_LCTR 实现。根据该寄存器中所选择的设置，PRU 串口接收器接收来自发送端设备的以下内容，接收数据格式：

1 START bit + data bits (5, 6, 7, 8) + 1 PARITY bit (optional) + 1 STOP bit（STOP bit，发送端随上述数据一起传输的其他任何多余停止位都将无法被检测到）。

RSR 负责从 UART*_RXD 引脚接收数据位。随后，RSR 将这些数据位进行拼接（串并转换），并将组合后的数值移入 RBR（或接收 FIFO）中。该数值可通过访问 RBR_TBR[7-0] 寄存器的 RBR_DATA 位域来读取。此外，PRU 串口还会向每个接收到的字符旁存入 3 位的错误状态信息，用以记录是否存在奇偶校验错误（parity error）、帧错误（framing error）或中止信号（break）。

在非 FIFO 模式下，当一个字符被放入 RBR，且中断使能寄存器 UART_INT_EN[0] ERBI 中的"接收数据可用"中断已使能时，系统将产生一个中断。当从 RBR 中读取该字符时，该中断将被清除。在 FIFO 模式下，当 FIFO 内部被填充至 UART_INT_FIFO 寄存器的 FIFO 控制最高有效位（MSB）部分所选择的触发阈值（trigger level）时，系统将产生中断；当 FIFO 里的内容物（字节数）降至该触发阈值以下时，该中断将被清除。

PRU 串口设计需要支持更改波特率和与 ARM 应用层进行交互，该系统可根据实际的用例进行定制，以最大程度地减少对通信链路的软件管理。需要包含一个可编程波特率发生器，该发生器能够将 PRU UART 的输入时钟乘以 1 至 65535 的除数（分频因子），从而为内部发送和接收逻辑生成 16× 或 13× 的参考时钟（过采样时钟）。

2.2 PRU 软件利用 PEFIF 接口实现多路高速串口

由于 PRU 内部仅集成一个硬件的高速 PRU 串口，当需要多路高速串口时，我们可以利用 PRU IO 的 PEFIF（Peripheral I/F 外设接口）模式对串口数据进行硬件的接收和发送。每个 PRU 上存在的三通道外设接口，一般是用来支持 EnDat 2.2 和 BiSS 等串口为基础的电机编码器协议，每路发送 PERIF 通道都存在 32bits 的 FIFO 用于直接存储发送的数据，每路发送 PERIF 通道都存在 4bits 的 FIFO 用于接收数据，我们可以使用该接口进行串口协议的接收和发送。

基于 PRU 实现的低延迟、多通道的高速 UART 传输系统架构即数据链路层的硬件实现设计如 图 3-1 所示，通过将传统的低速/高延迟任务硬件化，并利用 PRU 实时内核进行数据调度，实现了极高的吞吐率和微秒级的超低确定性延迟。

整个架构在物理和逻辑上明确划分为两个核心时延域，以隔离 ARM 侧由于系统调度带来的不确定性，高延迟域，主要包含 ARM 核心（负责高级应用层逻辑）、DMASS (DMA 系统)、通用 Timer、标准 UARTs（带 64 字节 TX-FIFO）、MSRAM 和 DDR 内存。以上组件的处理由于经过多层总线仲裁和通用操作系统调度，整体访问延迟在 250 纳秒左右，并且根据负载变化在不断变化，抖动可能达数微秒。在低延迟域中，主要由 ICSS 内部互联总线（32-bit, 250 MHz）、PRU 0 / PRU 1 双内核、专用 DMA（XFR2VBUS RD DMA）、SPAD 共享内存以及外设 perif TX/RX 硬件组成，基于 250 MHz 的 ICSS 总线，可以实现了 9 ns 读延迟和 6 ns 写延迟。并且系统通过专门定制的通道架构，为了实现数据从高延迟存储区到实时发送端的平滑过渡与高效搬运，设计了缓冲通道设计 (Ping-Pong Buffer Channel)，在低延迟域内部署了 4 个独立的逻辑通道（channel 1 - 4），每个通道均配备了双缓冲结构（如 Channel 1 包含 Tx1a 和 Tx1b 进行 Ping-Pong），每个单缓冲区大小为 128 字节。这种设计允许 DMA 在搬运当前块数据的同时，PRU 能够无冲突地读取并发送另一块数据。ARM 端通过 INTC（中断控制器）来与 PRU 核心进行交互。ARM 通过 Main CBASS 搬运并刷新这 4 个通道数据，每次刷新 88 个字节数据预计时间大约需要 70 µs，按照 12Mbps 速率进行发送完成 128Bytes（1280bits）所需时间大致为 106.6 µs，因此 70 µs 的时间是能够满足串口发送和接收的需求的，串口数据的的核心计算与控制由两个运行在 333 MHz 的 PRU 内核协作完成，PRU 0 专门负责 4 个通道的 UART 数据发送。发送数据的获取是通过专用的 XFR2VBUS RD DMA 直接从 PRU-ICSS 互联总线上的通道缓冲区拉取数据，无需经过 ARM 核心干预。PRU 1 专门负责 4 个通道的 UART 数据接收工作，通过底部的 XFR2VBUS RD DMA 将接收到的数据反馈回总线。

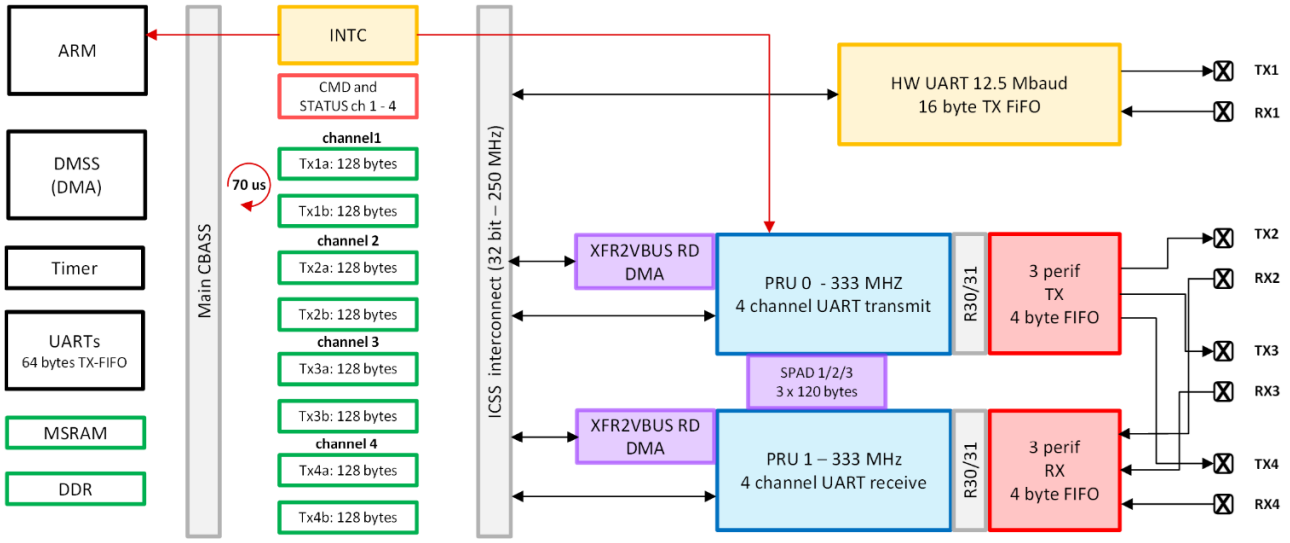

图 2-1. PRU UART 数据链路层硬件实现设计

发送数据最终通过硬件外设和引脚进行串行输出，其表现出极高的确定性：高速硬件 UART 直接集成在低延迟域，支持高达 12Mbps 的速率，内置 16 字节 TX FIFO。其余三路利用 PRU 专用外设发送和接收接口（3 perif TX/RX），PRU 核心通过专用的 R30/R31 寄存器接口，以极低的抖动直接驱动外设接口 TX 和 RX，输出至物理引脚，在高吞吐率下，串行帧之间也不会出现长延时空闲。能够实现全速的 12Mbps 的数据发送，通过 SPAD（Scratchpad 共享内存）PRU0 与 PRU 1 直接可以进行高速、零延迟的数据交换，可直接用于 PRU 的核间同步。

2.3 串口 ARM Linux 侧应用层驱动设计

为降低终端用户的集成门槛，驱动层需要采用 Linux 字符设备或 TTY 核心框架（tty_driver）进行封装，向用户空间抽象出标准的 POSIX 串口 API。最终目标是对终端客户来说，PRU 串口和普通串口在使用时没有区别。

Linux 内核通过标准 remoteproc (远程处理器控制) 框架对 PRU 的启动和固件进行管理，在固件加载阶段，Linux 启动时，Linux 端 remoteproc 驱动在 /lib/firmware/ 路径下加载对应的 PRU 二进制文件，将其解析并加载到 PRU 专有的指令内存（IRAM）中。PRU 固件中内嵌了一个资源表，Linux 读取该表，可以得知 PRU 申请了哪些系统资源（如虚拟中断号、RPMsg 通道、预留内存段）。IPC 核间通信机制使用 RPMsg 与硬件 Mailbox 机制来实现修改波特率、配置校验位、启动/停止流控等低频、非实时的控制指令。系统直接利用 AM62 内部紧邻 PRU 内核的 PRU 共享内存，PRU 和 ARM 均有权限进行访问。使用虚拟地址映射（ioremap）：在 Linux 内核驱动中，通过 ioremap() 函数将共享物理内存映射到 Linux 内核的虚拟地址空间。

PRU 固件只需要处理串口波形的生成和内存数据的解析，ARM 负责将需要发送的数据提供给 PRU，减少中断数量和 Linux 的调度开销，变成了每隔约 70 μs 进行一次的周期性数据操作。上层应用可以通过 mmap() 甚至可以直接将用户空间地址映射到 PRU 共享内存，数据从物理引脚到用户态程序，不需要进行内存拷贝。ARM 与 PRU 固件之间的接口是一个简单的、带有命令和状态寄存器的乒乓缓冲区。目前设计使用 PRU 的 Data RAM 0，在 ARM 端（A53 核心）映射的全局物理地址为 0x30040000。其中，数据缓冲区位于偏移量 0 处；命令和状态寄存器位于偏移量 0x400（即十进制 1024 字节）处。

从驱动程序侧来看，具体的编程/执行序列如下：

- 打开并初始化驱动：（此步骤执行下载并启动 PRU 固件，引脚复用 PIN MUX 和时钟设置目前已在 PRU 内部完成配置）。
- 设置波特率：（此步骤无 PRU 固件参与。直接由 ARM 端进行编程。PRU 在启动期间会默认设置为 12 Mbps baud rate。ARM 可以在加载第一帧数据之前修改波特率，并通过命令寄存器触发发送）。
- 检查发送（TX）通道缓冲区 A 的状态：需满足 TXA_CH0_STATUS != STATUS_ACTIVE（即确认当前缓冲区 A 处于非激活/空闲状态）。
- 将数据帧加载到发送帧缓冲区 A 中（即物理地址对应的 TX_CH0_A）。
- 设置命令寄存器 TXA_CH0_CMD = CMD_FULL：（此时 PRU 固件会轮询探测到 CMD_FULL 命令，并开始以设定的波特率发送该帧数据）。
- 与此同时，ARM 端可以同步处理发送缓冲区 B（即 TX_CH0_B）。
- 当 PRU 完成该帧数据的发送后，它会将状态寄存器修改为 TXA_CH0_STATUS = STATUS_DONE。
- ARM 端检测到该状态后，可以再次对发送缓冲区 A（TX_CH0_A）进行读写操作。
- 对所有通道重复上述步骤。

驱动程序将在 Linux 用户空间创建/dev/ttyPRU0/1/2/3 节点，用户空间可以通过这 4 个节点使用对应的串口通道进行串口数据的发送和接收。

3 高速串口实验验证

在完成以上软件理论设计以后，使用 TI EVM 可以对实现的 RPU 高速串口功能进行测试。

3.1 高速 PRU 串口裸机实验

在 SK-AM62B-K1 Board 上可以对应找到 PRU IO 的对应引脚进行测试，具体的硬件引脚分布如下：

3 channel RX: J10.14 - PRU1_GPI9 – N23 ；J10.15 - PRU1_GPI10 – N24；J10.16 - PRU1_GPI11 – N25

3 channel TX: AA3, SOC_MMC0_DAT2 rework via R376. PRU0_GPO1 – AA3；J10.17 – PRU0_GPO4 – P24；J10.20 – PRU0_GPO7 – R23

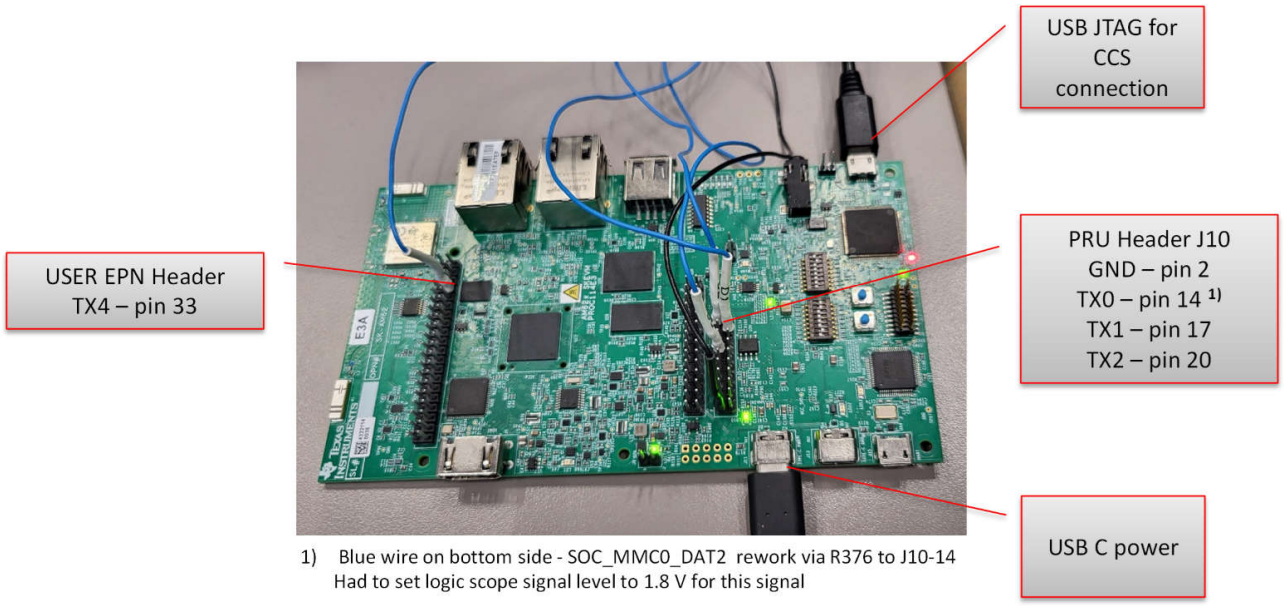

图 3-1. 基于 AM62 SK EVM 硬件环境搭建

软件环境上主要使用 CCS 通过 JTAG 来进行调试， 测试使用的 CCS 版本为 12.8.1，在连接到 AM62 EVM 后，需要按照该链接 描述进行初始化。之后连接到 PRU 核，将 PRU_UART_12Mbps.out 进行加载并运行，后续实际项目应用可以通过 Linux remoteproc 进行加载和运行。在 CCS 中可以方便的对内存进行操作，以观察对应引脚输出的串口波形。

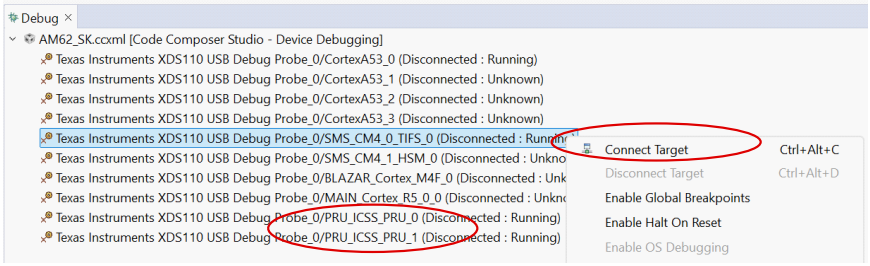

图 3-2. 基于 AM62 SK EVM 软件环境搭建

3.2 基于 Linux 应用高速 PRU 串口

基于 SDK11.1 对 PRU 的高速串口进行应用和实验验证，首先需要在原生的 kernel 中添加 PRU 串口的驱动，使用以下命令和对应补丁进行应用。

1. 将内核补丁应用到 SDK11.1 内核，重新编译内核，并将新的内核映像（Image）、模块以及 k3-am625-sk.dtb 安装到 SD 卡的根文件系统（rootfs）中。

```
host# patch -p1 < pru-tty-driver-v6.12-20251204.diff
host# make linux
host# sudo DESTDIR=/media/tony/root make linux_install
```

2. 将 PRU 固件复制到根文件系统的 /lib/firmware/ 目录中，然后为其创建一个名为 am62x-pru0-fw 的软链接。

```
# ln -sf PRU_UART_12Mb.out.v1.1 am62x-pru0-fw
```

3. 启动 am62x sk evm 开发板。新的 pru-tty 驱动已内置在内核映像中，它会创建 4 个 UART 节点：

```
# ls /dev/ttyPRU*
  /dev/ttyPRU0  /dev/ttyPRU1  /dev/ttyPRU2  /dev/ttyPRU3
```

4. 运行以下命令来启动 PRU0.

```
root@am62xx-evm:~# echo start > /sys/class/remoteproc/remoteproc2/state
[  703.729726] remoteproc remoteproc2: powering up 30074000.pru
[  703.737406] remoteproc remoteproc2: Booting fw image am62x-pru0-fw, size 25852
[  703.744705] remoteproc remoteproc2: remote processor 30074000.pru is now up
```

现在可以运行串口应用程序向这些 UART 节点写入数据了，目前只需使用 open() 和 write() 函数就可以实现使用 PRU 的高速串口发送数据。以下是使用 Open 和 Write 进行发送数据的参考代码。

```
write(fp0, data, data_len);
fp0 = open("/dev/ttyPRU0",  O_RDWR | O_NONBLOCK);
```

图 3-3 是通过 Linux 侧随机生成数据发送来验证整体从应用到底层数据通路，并使用逻辑分析仪进行测量得到的波形。

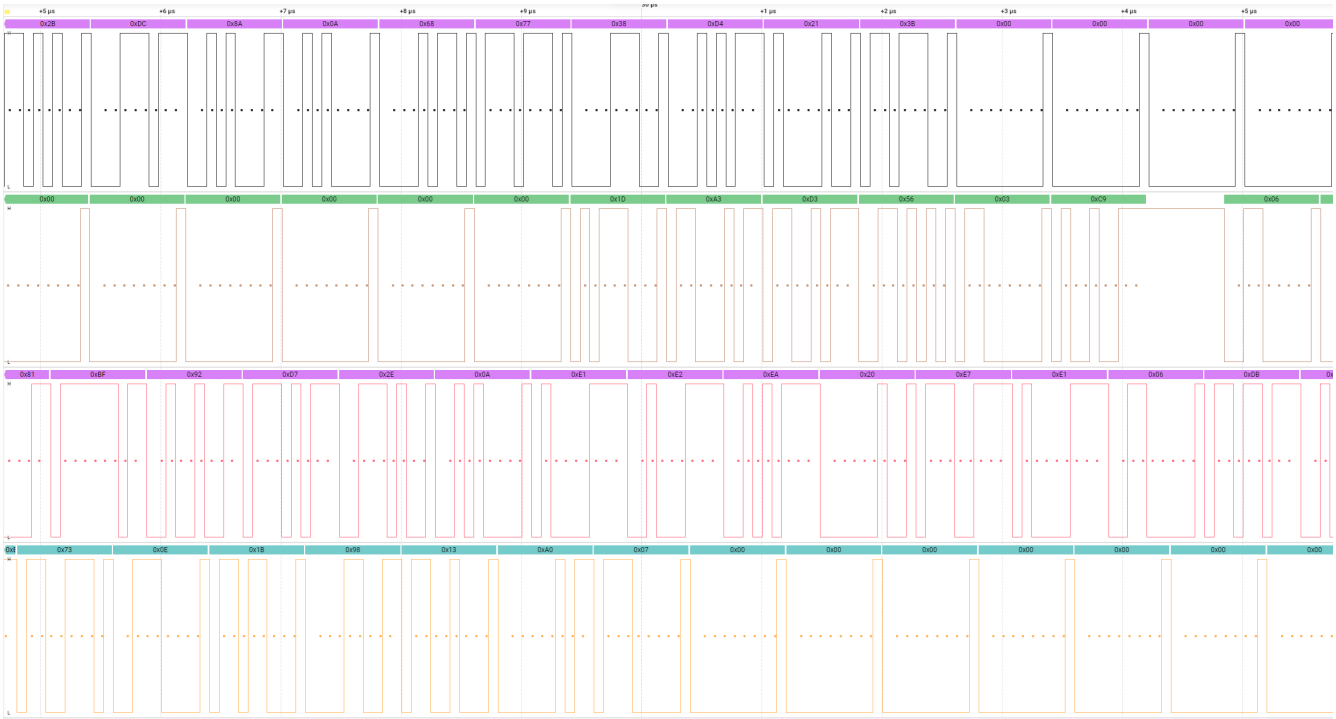

图 3-3. Linux 随机串口数据波形

4 总结

本文基于 TI Sitara 处理器开发了基于可编程实时单元（PRU）的高速（12Mb）串口通信方案，利用 PRU 的确定性架构模拟串口通信，在无需额外硬件成本的前提下实现了多路最高 12Mbps 的高速、零丢包串口通信。该设计通过 PRU 与 ARM 核心的共享内存机制构建高效数据通道，显著降低了 CPU 负载，为工业自动化场景提供了一种低成本、高可靠性的高速串口通信替代方案。

5 参考文献

1. PRU-ICSS Feature Comparison (Rev. H)
2. PRU Getting Started Labs
3. AM62x Sitara™ Processors datasheet (Rev. C)
4. open-pru/examples/uart_12mb at eb29734cbc0913e28ff71c6e046d9b10a1c35e5d · TexasInstruments/open-pru · GitHub
5. PRU-ICSS Getting Starting Guide on Linux (Rev. A)
6. Programmable real-time unit and industrial communications subsystem | TI.com
7. [FAQ] What is a PRU core? Why are PRU GPIO signals different from regular GPIOs? - Processors forum - Processors - TI - E2E support forums
8. AM62x MCU+ SDK: CCS Launch, Load and Run
9. AM62x Sitara Processors Technical Reference Manual (Rev. C)

重要通知和免责声明

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

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

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月