第 5 章　原型系统设计与实现

在本章中，我们对原型系统的设计与实现进行了详述。主要内容包括协议栈描述，系统结构，本文第 3、4章中设计的协议与算法（SATMAC、MBR 以及 DTN 层，路由算法的实现）的实现，以及原型系统基本性能的评测。由于实现的复杂程度，本章的主要篇幅集中于 MAC 下层的实现。原型系统主要由授时型 GPS、FPGA+ARM 异构芯片以及射频模块组成。GPS 模块提供高精度的时间同步来满足时隙接入算法的需求；为了保证时隙接入的精度和高优先级数据包的实时性，我们在 FPGA 中实现了 MAC 下层，并在 ARM 上运行的 Linux 操作系统中基于已有的网络协议栈实现了 MAC 上层、网络层以及 DTN 层的跨层协作。为了在真实场景中更方便地控制多节点的轨迹，我们使用多辆基于 GPS 轨迹的无人小车搭载原型系统进行轨迹控制。在本章接下来的内容中我们将根据协议栈由下至上的顺序描述原型系统的实现，并在最后对原型系统的基本性能进行了评测。

5.1　原型系统整体介绍

原型系统的设计与实现分为四个部分：MAC 下层（Sub-1GHz 信道接口和 SAT-MAC 信道接口）；MAC 上层（即 MBR 中继算法）；DTN 层、网络层路由的实现；基于无人小车的测试床。

原型系统基于 Xilinx 公司的 Zynq-7015 SoC [94]（整合了一个主频 667MHz 的双核 ARMv7 CPU 和一个 FPGA）和 u-blox 公司的 Neo-M8T 授时型 GPS 模块 [8] 进行构建。硬件的连接关系如图 5.1所示。CPU 与 FPGA 通过 AXI 总线（Advanced eXtensible Interface，AXI）接口连接；网卡（5.9GHz）使用的是 Atheros 公司的 AR9382，并接入 PCIe 总线，通过 AXI-PCIe 转换核接入 AXI 总线；Sub-1GHz（433MHz）射频模块使用的 Silicon 公司的 Si4463 模块，通过 SPI 接口与 FPGA 子系统进行连接；GPS 模块则通过 GPIO 和串口（UART）与 FPGA 进行连接。图 5.2展示的是开发板的正反面，可以看到其由两个开发板连接而成，分别为核心板和 IO 扩展板。核心板为 AVNET 公司

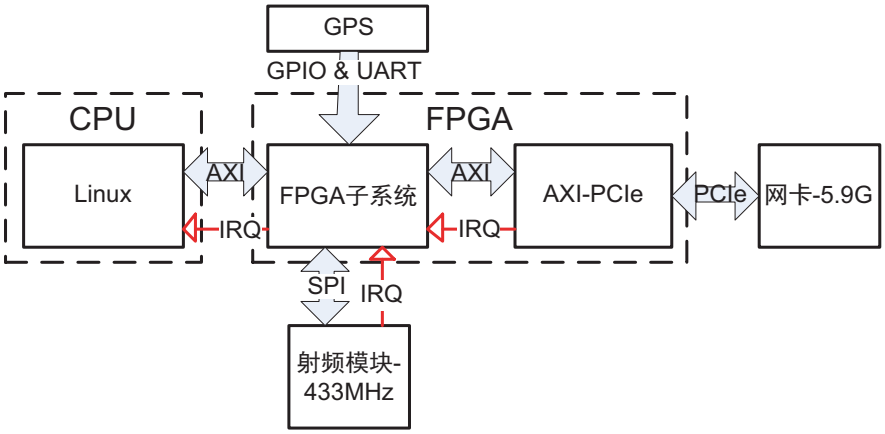
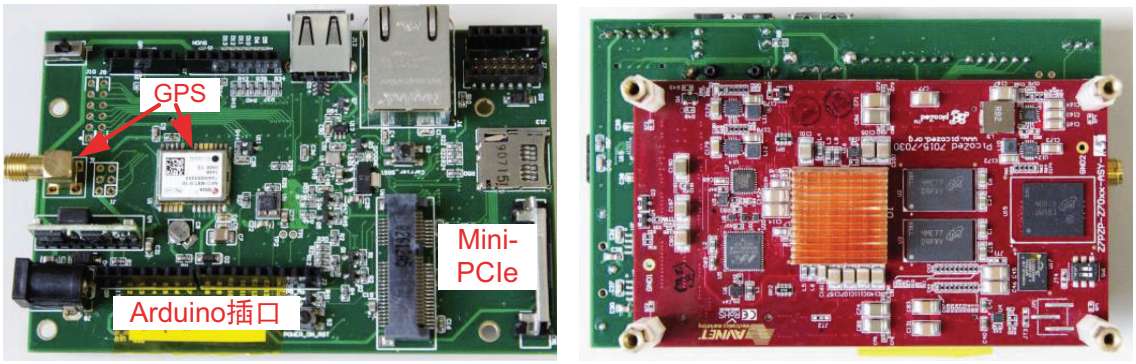


图 5.1　硬件连接示意图



(a) 正面　　(b) 反面

图 5.2　开发板正反面

生产的 Picozed 开发板，包括 Zynq-7015 核心，1 GB 内存和 4 GB 闪存。基于 Picozed 的 IO 扩展要求，我们自行设计并生产了 IO 扩展板，主要包括 GPS 模块，Mini-PCIe 插槽（连接 5.9GHz 网卡），Arduino 扩展 IO（连接 Sub-1GHz 射频模块），Micro-SD 卡插槽和 RJ-45、USB-2.0、串口、JTAG 等接口。表 5.1列出了测试床的构成模块和参数，其中 Zigbee 和蓝牙为管理接口，用于接收用户发送的指令来做到同时开始实验（同步启动）。5.9GHz 网卡运行在 178 号信道（5.885 GHz 到 5.895 GHz 频段）的 OCB 模式中（Outside the Context of a BSS，OCB），其数据发送速率固定为 12 Mbps（12 Mbps 是 IEEE 802.11p 协议指定支持的数据发送速率）；Si4463 模块运行在 433 MHz 频道上，数据速率设为 100 Kbps。

第1章已经提到过，由于安全应用的 QoS 需求，车载自组网中的时隙接入原型系统的设计要考虑的关键问题是高精度的分布式时钟同步以及高优先级数据包的硬实时收发。首先，系统利用 GPS 提供的时钟脉冲信号结合 FPGA 的高精度处理做到了净

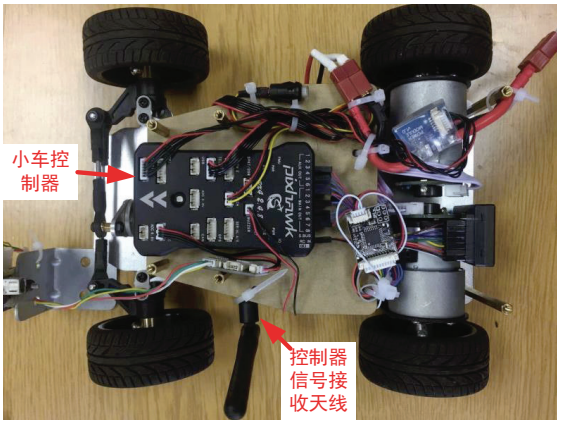
空地带纳秒级的分布式时钟同步精度；其次，系统需要具备实时处理网络设备中断以及数据包收发的能力。在保证了硬实时这个需求后，系统要提供全栈网络协议，且保证各种应用程序以及其他网络协议实现的兼容性。为了满足上述需求，本工作的设计思路是软硬件结合，利用 Linux 提供完整的网络协议栈以及应用兼容性；利用 FPGA 处理高优先级的网卡中断并处理高优先级报文收发。

表 5.1　原型系统基本参数

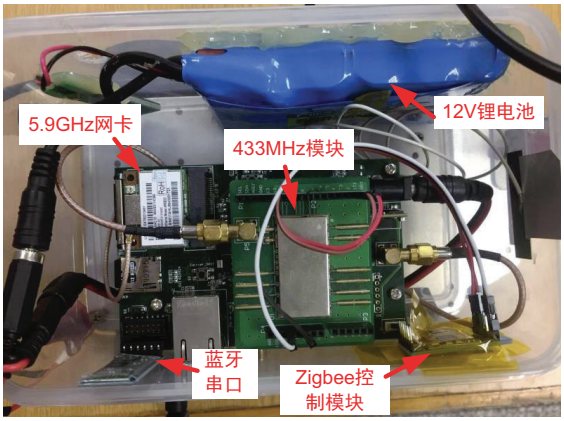
CPU+FPGA	Xilinx Zynq-7015 SoC Dual-core CPU @ 667 MHz
内存	1 GB
射频	Atheros AR9462 @ 5.9GHz（CH178），10 MHz 带宽 Si4463 @ 433 MHz
数据速率	5.9 GHz：12 Mbps 433 MHz：100Kbps
接入方式	5.9 GHz：TDMA（SATMAC） 433 MHz：CSMA
GPS 模块	u-blox NEO-M8T
操作系统	Linux（内核版本 4.4）
时隙长度	1 ms
小车速度	0.8 m/s
电池	12 V，5600 mAh
其他通讯模块	Zigbee 和蓝牙串口

为了评估本文提出的算法以及协议在实际环境下的性能，一个关键问题是怎样控制多个节点的移动性。为了解决这个问题，我们将原型系统搭载在无人小车上（如图 5.3(a)所示），利用多辆小车组成动态场景。每台小车由电机和舵机组成，并由一个名为 Ardupilot 的开源飞控软件 [109] 单独控制。通过利用 GPS 和电子罗盘，小车能够以固定的行驶速率（比如 1 m/s）根据用户定义的轨迹进行运动；用户轨迹的定义如图 5.4所示，在控制端配置好轨迹后，再通过无线传输方式将轨迹上传至对应的小车。如图 5.3(c)所示，在小车上我们使用了双 GPS 天线，并使用了差分 GPS 定位系统（Differential GPS，DGPS），以此达到了足够高的定位精度（实测定位误差小于 0.5 米）。基于此，我们可以轻松地控制所有节点的移动性。

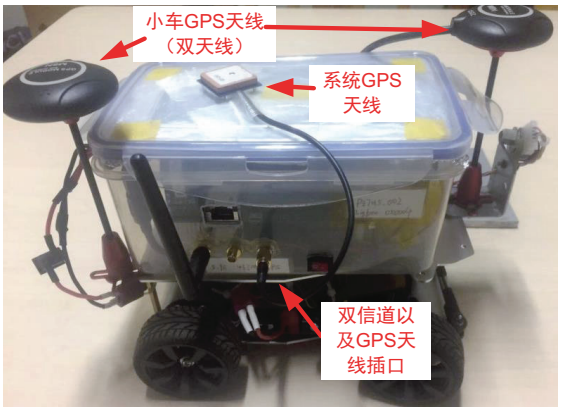
图 5.3(b)展示了小车上搭载的原型系统内部模块，除了双信道以及 GPS 模块（在 433 MHz 模块之下）之外，我们使用了一个由 6 根 18650 锂电池组成的 12 V，5600 mAh 锂电池组对电子系统以及小车进行供电（充满电可以支持小车、系统不间断运行 2 到 3 小时）；使用了一个 zigbee 模块进行实验的控制，使用了一个蓝牙串口模块作为 Linux 终端（console）。另一方面，为了控制实验环境，我们需要在一个面积较小的区域内进行实验，这要求对节点的通讯距离进行限制。我们通过控制网卡的发送功率来实现



(a) 无人小车俯视图



(b) 原型系统内部俯视图



(c) 小车 + 原型系统整体侧面图

图 5.3 (a) 无人小车俯视图，(b) 原型系统内部俯视图，(c) 小车 + 原型系统整体侧面图

这一目标，在我们的原型系统中，节点的通讯范围可以根据需求控制在 5 m 到 100 m 的区间内。

协议栈整体结构如图 5.5所示，简单来说，DTN 模块在应用层中基于 DTN2 [110] 进行实现，在网络层中我们分别实现了 AODV+ 与 GeoSVR+（包含了与 DTN 层的交互模块），同时只能开启一个路由协议。我们根据4.2节中描述的层次接口以及 DTN 链路的自适应调整算法的设计对 DTN2 功能和 AODV 以及 GeoSVR 实现代码进行了对应的修改。在 MAC 上层中我们实现了 MBR 模块，它的实现严格按照4.1节中的协议设计进行，包括层次间接口、电子地图瓦片缓存、道路匹配算法与中继区域的表示和计算。在链路层下层包括了 Sub-1GHz 信道接口和 SATMAC 信道接口，Sub-1GHz 模块使用的是 CSMA/CA 的信道接入模式，模块结构相对简单；SATMAC 模块严格按照第 3章中的设计进行实现，包括了时隙的申请与冲突解决、时隙状态的维护、不安全占用时隙的调整以及基于二进制与时隙调整的动态帧长。在接下来的章节中我们将详

图 5.4　Ardupilot 用户定义轨迹示意图

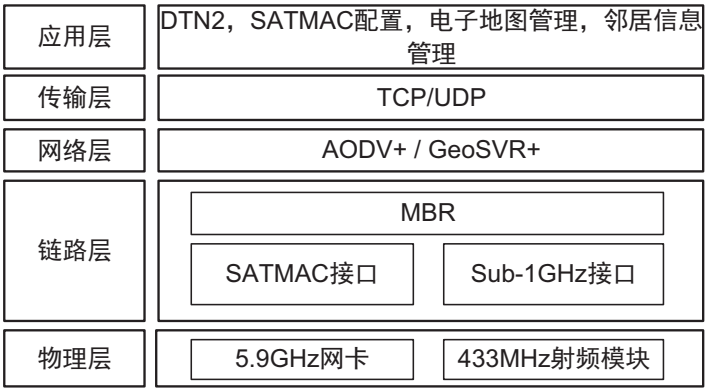


图 5.5　原型系统协议栈示意图

细描述各个模块的设计与实现。

5.2　MAC 下层的设计与实现

在一般的基于 Linux 的网络节点中，软件网卡驱动直接控制系统与网卡之间的交互，其中的处理延迟是无法保证上界的 [32]。为了在系统中对高优先级报文提供硬实时的数据传输，我们在软件数据链路层的最底端，即网卡驱动和网卡硬件之间加入了一个 FPGA 子系统来处理与网卡相关的关键任务，包括高优先级中断的处理与响应以及高优先级数据报文的发送与接收。

图 5.6展示了原型系统中 MAC 下层的设计框架，其中分为两个信道接口的设计。Sub-1GHz 信道接口包括软件接口驱动和位于 FPGA 的状态机控制模块，前者负责软

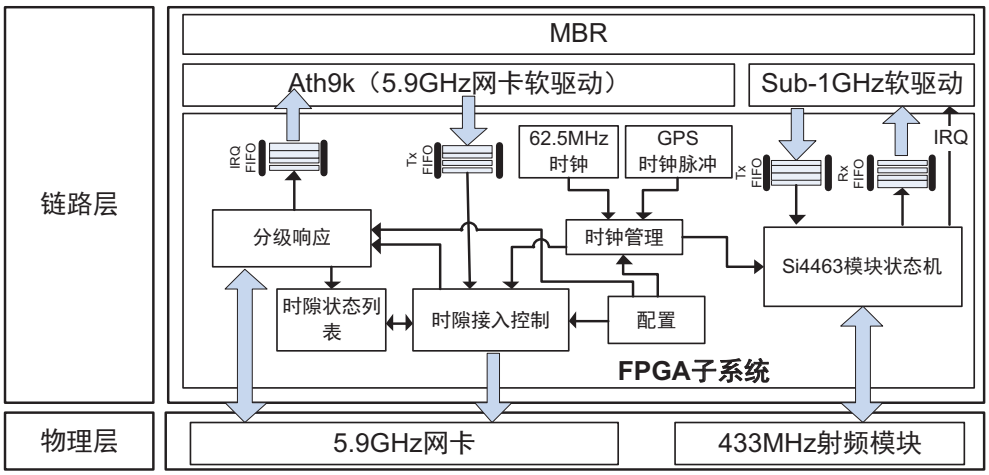


图 5.6　原型系统实现架构

件和 FPGA 状态机之间的数据包交互；后者负责控制 Si4463 模块的配置、中断处理以及数据包收发。Si4463 模块通过 SPI 接口与 FPGA 子系统连接，并通过该接口收发数据包。Sub-1GHz 模块的逻辑相对简单，本节余下的内容将集中描述 SATMAC 模块的设计与实现。与 Sub-1GHz 信道接口的设计类似，SATMAC 信道接口也由软件部分和 FPGA 部分构成，软件控制的是网卡的初始化，配置以及软硬件之间数据包的接口；FPGA 部分控制高精度时间、时隙状态维护与时隙接入。从总体上来看，FPGA 子系统中的 SATMAC 信道接口部分包括一个时间管理模块，一个分级响应模块，一个时隙接入控制模块和一个配置模块。其中分级响应模块负责处理中断以及数据包的收发：对于中断的处理，该模块捕获来自网卡的中断请求，并处理其中的高优先级中断请求（例如高优先级数据报文的接收中断），另外的普通中断请求将通过中断队列传递到 Linux 中进行处理；对于数据包的发送，上层待发的数据包将在发送队列中进行缓存并等待由时隙接入控制模块传递的发送使能信号；而当从网卡收到高优先级数据报文的接收中断时，模块会解析该数据包并进行对应的处理。例如，当节点收到一个 FI 报文，分级响应模块将首先解析该 FI 报文，并相应地更新时隙状态列表。时隙接入控制模块控制时隙的接入和 FI 报文的传输，其根据时隙状态列表在每一帧构建一次 FI 报文，并且在节点进入其基础时隙时发送该 FI 报文。之后，发送使能信号将在基础时隙余下的时间中被激活。

在系统中，如同 3.1节中所描述的那样，同步后的时间被划分为帧，每帧由可变数量的固定长度时隙组成。图 5.7展示了一个基础时隙中的数据包发送情况：FI 报文，基础安全报文（BSM），以及上层的数据报文。保护时间设为 100 μs。SIFS 为最短帧间隔

（Short Inter-Frame Space），DIFS 为分布协调功能帧间间隔（DCF Inter-frame Space），其中 SIFS 为 32 μs，DIFS 为 58 μs。

5.2.1 时钟管理

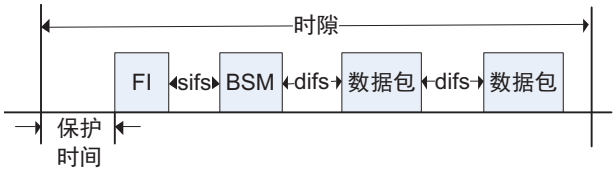


图 5.7 一个时隙之内传输的数据包示意图

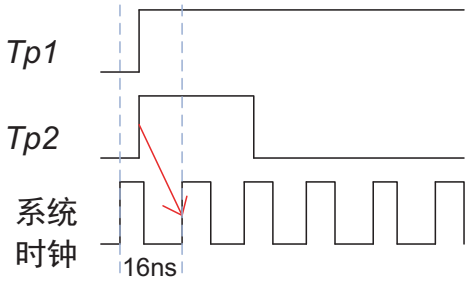


图 5.8 将 GPS 时钟脉冲对齐到系统时钟上

为了实现硬实时的高优先级数据包的处理以及高带宽利用率（较小的时隙保护时间），在 FPGA 子系统的时钟模块中执行了基于 GPS 的高精度分布式时间同步。系统中使用了 u-blox 公司的 NEO-M8T GPS 模块，其可以提供两个同步时钟脉冲，分别用 *Tp1* 和 *Tp2* 表示（图 5.8）。*Tp1* 的频率为 1 Hz 且与 GPS 秒钟进行同步：每个信号上升沿表示一个 GPS 秒的开始。此外，一个 *Tp1* 的上升沿同样代表着一帧的开始，换句话说，一帧不能横跨 *Tp1* 的上升沿。*Tp2* 与时隙一一对应，即一个 *Tp2* 的上升沿表示一个时隙的开始。基于 *Tp1* 与 *Tp2*，系统将时间划分为帧。

FPGA 子系统使用了一个频率为 62.5 MHz 的晶振作为系统时钟源，其中一个时钟周期为 16 ns。如图 5.8所示，系统会将 GPS 提供的时钟脉冲对齐到系统时钟上，因此时钟同步的误差为一个系统时钟周期（16 ns）加上 GPS 的时钟同步精度（净空下小于 20 ns [8]）。这个时钟同步的精度足以满足微秒级别时隙长度的时间要求。需要注明的是，当 GPS 信号丢失时，例如，当节点进入隧道时，本地时钟晶振可以在短时间内保持足够的精度（小于 500 ns [8]）以进行时间同步。如果 GPS 信号长时间丢失，则需要另一个替代的时间同步方案，直到 GPS 信号恢复。本工作不考虑这个替代方案的设计，它可以作为未来工作进行开展。

在原型系统中，*Tp2* 的频率是可以进行调整的，因此时隙的长度可以按需进行设定。同样，帧的长度也可以进行动态配置，并且支持 $\{4, 8, 16, 32, ...\}$。我们将 *Tp2* 的

频率定为 1024 Hz，即每秒有 1024 个时隙，此时每个时隙长约为 976.5 微秒。

5.2.2 中断的分级响应机制

时隙的精度和信道利用率由数据包处理时延和时钟同步精度这两个关键决定。在上一节中我们已经对时钟同步精度进行了分析，原型系统在 GPS 和 FPGA 的协作下可以达到纳秒级别的时钟同步精度，足以满足时隙的需求。在本节中，我们利用3.5.2节中设计的中断分级响应机制保证数据包的处理时延。

在原型系统中，数据包在系统和网卡之间的传输通过 EDMA 模块（Enhanced Direct Memory Access，EDMA）完成，接收和待发送的数据包都会被缓存在特定内存块中，并通过 “描述符” 这一数据结构进行系统和网卡之间的交互。在系统需要发送数据包时，Linux 将构造固定长度的发送（Tx）描述符，其中包括数据包负载的内存地址以及发送该数据包所指定的控制字段（例如重传的次数，传输速率等等指标），最后将发送描述符传递至网卡的发送队列中。在 Linux 中创建的发送描述符将缓冲在 FPGA 子系统的发送队列中，并等待时隙接入控制模块给出的发送使能信号，并在信号有效时将发送描述符发送到网卡。当网卡收到一个数据包时（有可能是正常接收或者接收错误），网卡将从接收（Rx）描述符队列中获取一个空描述符，并将刚收到的数据包的相关信息记录到该接收描述符中；最后生成一个中断请求（IRQ）并发送至系统。当系统收到一个中断请求时，它将向网卡查询中断原因（中断码）并处理对应的任务。

图 5.9展示了中断分级响应模块的结构。模块负责处理网卡和 Linux 之间的交互，包括中断的响应和处理以及上层数据包的发送。模块拦截并处理网卡发起的关键中断请求，并将非关键中断请求传递给 Linux 进行处理。关键中断请求包括系统故障事件以及 FI 和高优先级报文的发送与接收事件；非关键中断请求包括上层应用数据包的收发事件和其他的网卡控制事件，例如初始化配置以及接收错误等。模块将不同的中断请求分为三类：TxOK，RxOK 和网卡控制相关。TxOK 代表一个数据包已经被网卡成功发送至信道中，网卡将在数据包的发送流程完成的时刻给出 TxOK 中断请求。如果在数据包发送描述符已经传递给网卡后的一定时间内（比如 1 ms）没有收到网卡给出的 TxOK 中断请求，则看门狗模块将重置整个系统。

模块利用网卡内拥有的不同的队列将数据包区分为高优先级报文和低优先级报

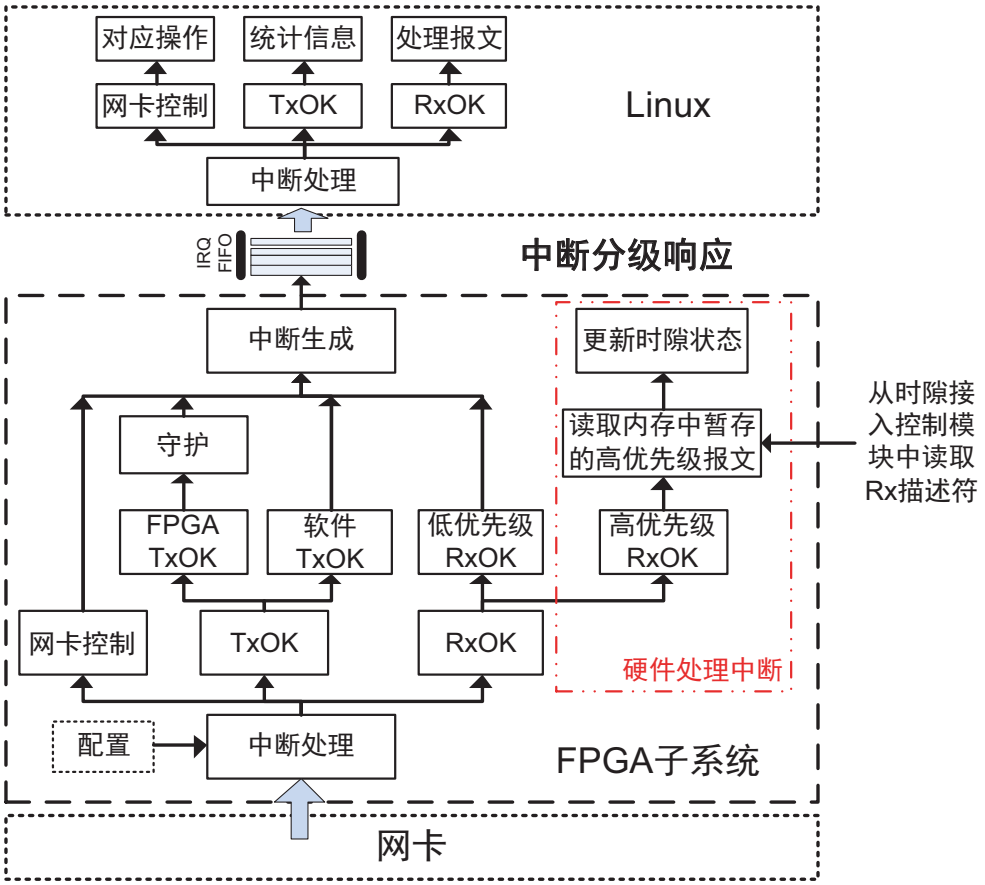


图 5.9　中断分级响应模块

文。网卡拥有十条不同优先级的发送队列以及两条接收队列：低优先级报文接收队列，标记为 *LP*；以及高优先级报文接收队列，标记为 *HP*。当网卡接收到一个数据包时，它将根据报文头部中的对应字段将该数据包放入 *LP* 或 *HP*，然后发起一个 RxOK 中断请求表示该报文接收事件。当模块收到一个中断并且判断为高优先级数据包的接收中断时，模块将直接对该数据包进行处理。例如当模块收到一个 FI 报文时，它首先解析该报文，并基于 SATMAC 协议的处理规则（第 3章）根据报文中携带的时隙状态更新本地维护的时隙状态列表。

除了报文的接收处理，中断分级响应模块还负责处理上层发送的数据包。这些数据包的发送描述符将首先在 FPGA 子系统的发送队列中进行缓存，并等待发送使能信号。当发送使能信号被激活时，模块将计算发送队列中第一个数据包的预计发送时间，如果发送时间小于时隙的剩余时间，则将数据包（逐个）发送至网卡。计算公式如下：

$$T_{Remain} = T_{Slot} - T_{Guard} - T_{FI} - T_{BSM} - T_{Used} \tag{5.1}$$

其中 T_{Slot} 为时隙的长度，T_{Guard} 为时隙保护时间，T_{FI} 和 T_{BSM} 分别为传输 FI 以及 BSM 的发送时间（包括了 SIFS），T_{Used} 为在本时隙中用于传输上层数据包所用的时间（包括了 DIFS）。

中断分级响应模块的系统延迟包括两个部分：处理延迟以及内存的读写延迟。内存读写延迟包括两个部分：第一部分是通过 PCIe 总线访问网卡寄存器所需要的读写延迟。由于网卡与 FPGA/CPU 之间的数据交换只占用 PCIe 总线的一小部分带宽，这部分延迟在系统中是固定的，约 100 个时钟周期（1.6 μs）；第二部分是通过 AXI 总线访问内存（DRAM）的读写延迟，需要约 50 个时钟周期（0.8 μs）。此外，读写延迟有可能被当前内存上的读写量所影响。根据在本章之后的 5.5节中的实验结果，CPU 和 DRAM 上的重负载不会影响原型系统中 FPGA 子系统的性能。对于每一个中断，模块需要通过读取网卡中断寄存器得到中断码，需要约 100 个时钟周期；处理完中断后需要清空网卡以及 AXI-PCIe 软核中的中断寄存器，需要约 200 个时钟周期；则响应一个中断最少耗时约 300 个时钟周期，即 $300 * 16ns = 4800ns = 4.8\mu s$。处理延迟为模块中每一个处理流程所需要的时间。其中最耗时的流程为 FI 的处理，且处理所需的时间与当前的帧长相关。例如，当帧长为 128 时，模块需要大约 400 个时钟周期来遍历 FI 报文中所有的时隙状态并将其更新到本地维护的时隙状态列表中，加上中断的读取和清除时延，这个过程共需要约 $700 * 16ns = 11200ns = 11.2\mu s$。由于 FI 只在每个时隙的开始位置被接收到且 11.2 μs 的延迟远小于 SIFS，因此在模块处理完 FI 报文之前不会再接收到新的数据包导致排队时延的出现。除此之外，模块中其他的流程所需要的时间都是纳秒级别的，只需要考虑中断码的处理时延，即 $4.8\mu s$。因此，本系统可以达到微秒级别的高优先级数据包处理时延，这足以满足时隙接入系统的需求。

5.2.3 时隙接入控制模块

如图 5.10所示，时隙接入控制模块包括六个部分：协议状态机、候选时隙搜索模块、发送/接收描述符管理模块、FI 报文构造模块、数据发送模块以及测试/BSM 模块。

总的来说，我们在协议状态机模块中实现了 SATMAC 协议的协议状态机，它负责控制 FI 报文的传输，基础时隙的申请以及数据包传输的冲突检测与解决（融合冲突与接入冲突 [73]）。FI 报文构造模块控制 FI 报文的构建（图 3.2展示了 FI 的报文格式）。

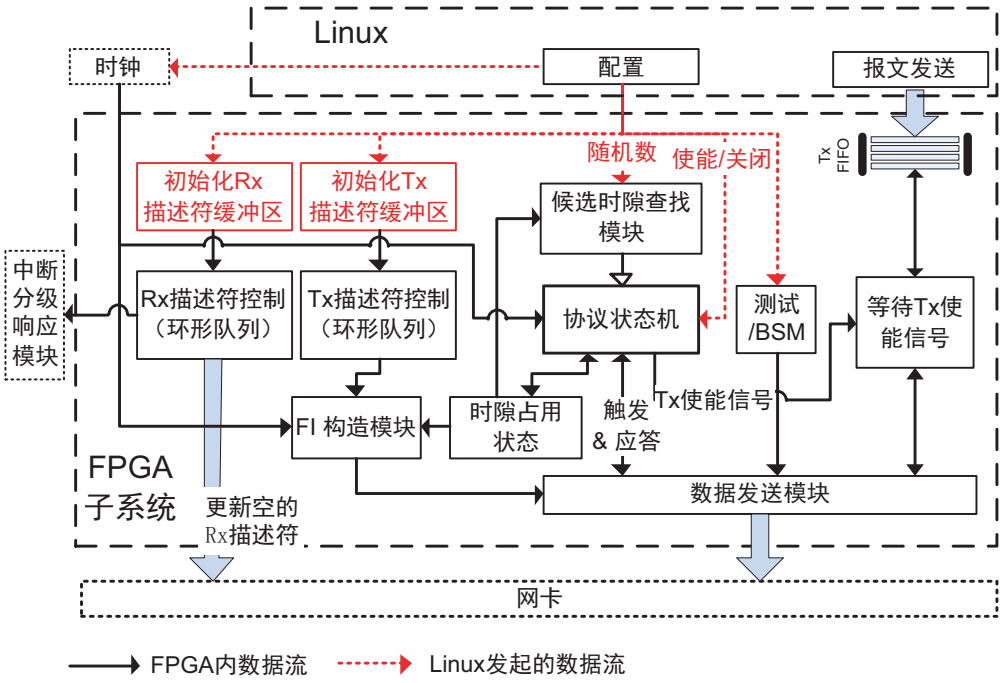


图 5.10　时隙接入控制模块

当一个节点需要申请基础时隙时（节点刚刚启动，或者由于冲突导致协议放弃已经申请到的基础时隙），协议状态机模块将占用候选时隙搜索模块选出的候选时隙，并更新时隙状态列表。候选时隙的搜索包括遍历本地维护的时隙状态列表，以及按照规则选出一个空闲的时隙。由于遍历状态列表是一个很耗时的过程，因此我们将候选时隙搜索模块设计为独立于协议状态机模块运行，并且连续地在每一帧开始后进行一次候选时隙的选择。此外，候选时隙搜索模块用到的随机数由 Linux 通过内存映射接口提供。

在基础时隙开始后，协议状态机模块将生成一个信号触发 FI 报文的传输。该报文由 FI 报文构造模块在基础时隙开始前基于时隙状态列表进行构建。与候选时隙搜索模块类似，FI 报文构造模块独立于协议状态机模块运行，因此当协议状态机模块生成的触发信号到达时，数据发送模块可以立即将已经构建好的 FI 报文发送到网卡。在发送完 FI 报文后，数据发送模块接着将 BSM 报文发送到网卡，然后向协议状态机模块发出 ACK 信号。在接收到 ACK 信号后，协议状态机模块将在基础时隙剩余时间内激活发送使能信号。

正如在之前描述的，网卡和系统之间的数据包交互接口基于 EDMA 和发送/接收描述符。在原型系统中，Linux 端的数据包描述符在 ath9k 驱动中进行初始化和维护；

FPGA 子系统中的描述符由软件进行初始化，并在发送/接收描述符管理模块中进行维护，系统初始化后，管理模块将循环使用这些描述符直到系统重启。网卡会把新接收到的数据包内存地址写入可用的接收描述符内，该描述符由接收描述符管理模块负责传递与更新，并且将传递给网卡使用的描述符同时交给中断分级响应模块，以便它可以在高优先级接收队列的 RxOK 中断请求到达后提取接收到的数据包。

通过内存映射接口，上层应用可以按如下方式配置时隙接入控制模块：首先，时隙接入功能可以由上层决定是否开启。功能关闭时，协议状态机模块不会尝试申请基础时隙并始终激活发送使能信号。其次，为了测试或调试等目的，上层可以通过给 FPGA 子系统提供特定的随机数来控制协议状态机模块中的基础时隙申请。例如，上层可以通过固定地提供零作为随机数来让协议状态机模块获取帧中的第一个可用时隙。第三，上层可以操纵测试/BSM 模块来控制 BSM 中的内容或发送测试报文以用于测试或调试等目的。在当前的原型系统设计和实现中，测试/BSM 模块仅生成和发送数据包，在系统中没有包含安全相关的信息收集与处理，这一部分内容可以作为未来的工作进行开展。此外，上层应用可以通过配置接口监视并收集协议状态，例如数据包收发计数，数据包传输冲突计数（融合冲突与接入冲突）等等。

5.2.4 协议状态机的设计

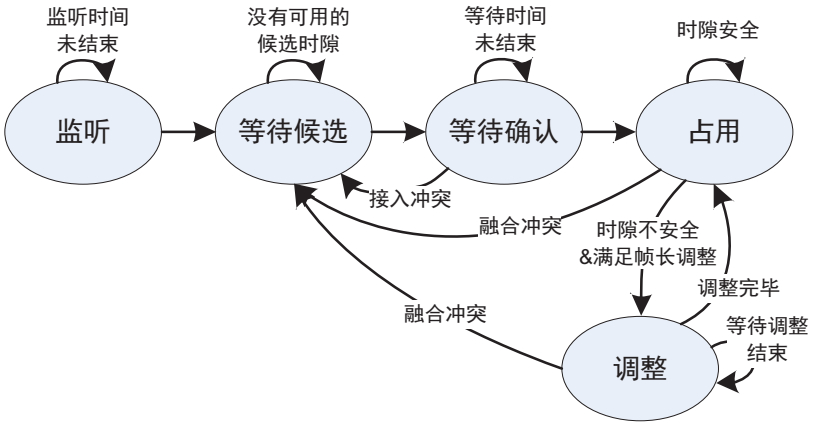


图 5.11　协议状态机的状态转换图

图 5.11展示了 SATMAC 协议的状态转换图。当节点刚刚开启（或重置）且节点需要申请基础时隙时，节点将侦听一帧的时间以收集 2HS 中的时隙占用情况 (“监听”状态)。然后，它将选择一个候选时隙并将该时隙标记为本节点占用，并在选中时隙

到达时发送 FI 报文。之后，节点将等待一帧的时间并检查时隙是否申请成功（没有接入冲突）。如果申请成功，则节点将进入“占用”状态；如果申请失败则协议判断出现接入冲突，节点进入“等待候选”状态重新开始申请时隙。节点处于“占用”状态时，如果占用时隙变为占用不安全或者满足了帧长调整的准备条件，则节点进入“调整”状态尝试调整基础时隙占用；当调整完毕后节点退回“占用”状态，如果产生了融合冲突则退回“等待候选”状态重新申请时隙。

5.3 MAC 上层（MBR）的实现

MBR 的实现的关键问题有两方面：电子地图数据（OSM）的导入以及对 Linux 网络层到链路层数据包处理逻辑的修改。

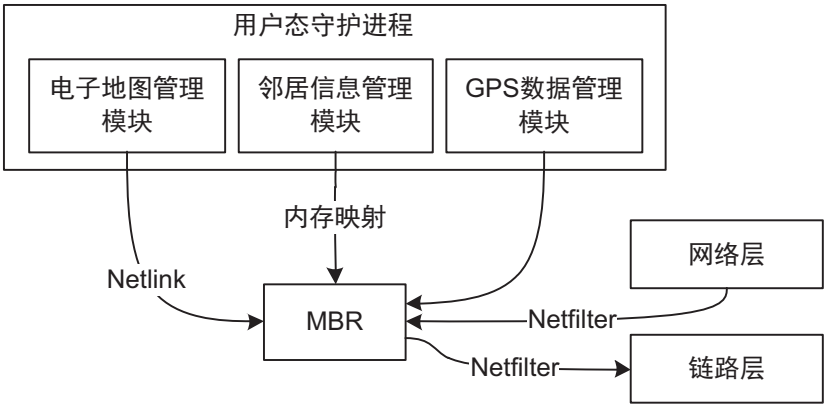


图 5.12　实现结构示意图

整体的实现结构如图 5.12所示。MBR 模块位于网络层与数据链路层之间，运行在 Linux 内核态，通过 Netfilter [105] 进行连接。电子地图的管理、邻居信息的管理以及 GPS 数据的管理都在用户态守护进程中进行，并通过 Netlink，内存映射两种方式与内核态的 MBR 进行数据交互。

图 5.13描述了原型系统中 MBR 的数据流。数据包被分为了上层应用数据包和邻居管理模块控制的广播信标。应用数据包又被进一步拆分为单播报文和广播报文。在发送方，当一个单播报文要被转发到下一跳节点时，MBR 将判断是否对该数据包进行中继。特别的，算法首先检查当前节点和下一跳节点是否在同一条路上，如果不是，则算法会在邻居列表中寻找合适的中继节点；成功找到中继节点后则在数据包的 MAC 头部标记“MBR”标志，并在头部中填充本节点、中继节点和下一跳节点的 MAC 地