指示

1.熟练使用掌握 Jira 缺陷管理工具的使用，对 BUG 进行跟踪与管理
3.熟练使用 CAN 工具进行抓包，分析日志，回放日志，模拟仿真等

6.了解高通音频架构、

7.了解QNX系统开发、

8.了解CAN和FDbus总线。

9.熟悉车载报警系统avas开发，

10.DSP

11.BSP

12.MPU

STM32移植MPU6050DMP库1*哔哩哔哩*bilibili

13.AVAS

14.ADAS

ADAS（Advanced Driving Assistance System，高级驾驶辅助系统)

了解ARM架构、了解高通音频架构、了解QNX系统开发、了解CAN和FDbus总线。熟悉车载报警系统AVAS开发，熟悉基于CAN对ECU节点的调试。

基于C51蓝牙循迹小车

Ø **项目描述：**接入HC-08蓝牙模块、红外传感模块和电机的89C51MCU。能够实现对路线循迹，与手机蓝牙连接，接受手机命令控制转向。

Ø **项目收获：**proteus仿真、线路搭建、开发板烧写、PWM控速、外设控制、安卓蓝牙开发等相关知识。

基于STM32水质检测终端

Ø **项目描述：**以STM32为核心，外接液晶显示模块、PH 传感器、浑浊度传感器、WIFI模块实现检测水质，并用手机APP获取到具体数据并显示出来。

Ø **项目收获：**STM32开发、WIFI透传、Usart、I2C、SPI、TCP/IP通信、串口调试等相关知识。

基于STM32智慧农业终端

Ø **项目描述：**传感器模块负责采集农业环境信息；STM32微控制器负责数据处理；无线通信模块采用NBIoT通信技术，将采集到的数据上传至云端。手机App通过http与云端相连，获取信息或下发指令。

Ø **项目收获：**STM32开发、NBIoT长距离通信、USB接口log抓取，GPS、HTTP、MQTT协议等相关知识。

基于STM32和ZigBee智能家居

Ø **项目描述：**用两个 ZigBee模块作为ZigBee网关和终端，终端可实现温湿度采集和电机驱动灯的开关。接入MCU的网关负责接收和下发数据，MCU外接WiFi模块与手机APP相连，实现应用层传输。

用两个连接STM32的 ZigBee模块作为ZigBee协调器和终端。可实现终端温湿度、光度、烟雾采集和远程驱动灯、空调的开关。协调器负责接收和下发数据，STM32外接无线通信模块，采用NBIoT通信技术，将采集到的数据上传至云端。手机App通过http与云端相连，获取信息或下发指令。

Zigbee模块（CC2530）详解-CSDN博客

30元让你实现远程遥控家里的空调 | 无损改装 | 自制红外遥控器*哔哩哔哩*bilibili

Ø **项目收获：**ZigBee组网、短距离通信、网络编程等相关知识。

熟练使用C语言，熟悉C++、Python，
能够阅读并编写基本的汇编语言
熟练使用 Keil、IAR、Proteus、Vscode 以及 Cubemx 等进行开发，熟练使用HAL库、标准库以
及寄存器进行STM32、GD32等MCU的开发
熟练使用 Ubuntu，熟悉ARM 裸机开发、Linux 系统编程、Linux 系统移植以及 Linux 系统驱动
熟悉IMX6ULL，PX30以及树莓派的使用，可以熟练进行 Linux 环境搭建以及硬件驱动测试
掌握MODBUS、YMODEM 等通讯协议，IIC、SPI、UART 等总线，熟悉OS 实时操作系统
熟练各种硬件焊接，熟悉万用表、示波器的使用，熟练通过信号波形看协议并调试软硬件
熟练串口屏设计开发、熟悉emWin，具有良好的英文文档阅读能力，有独立解决问题的能力

QNX

QNX 是一个实时操作系统。它提供用户可控制的、优先级驱动的、急者优先抢占的调度方式。QNX 内核自身开销小、上下文切换快，在同样的硬件条件下给实时应用留下更大的余地，因而它在实时控制、通信、多媒体信息处理等对时间敏感的应用领域大有用武之

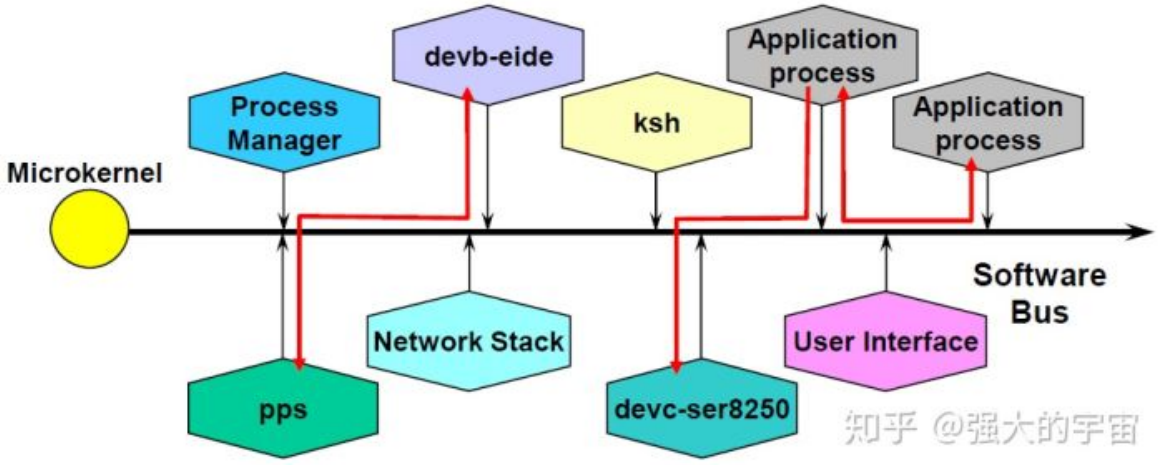
地。

QNX 同时也是一个可嵌入的操作系统。它由微内核和一组共操作的进程构成，具有高度可伸缩性，可灵活的剪裁。最小配置只占用几十 KB 内存。因此，它可以广泛地嵌入到智能机器、智能仪器仪表、通信设备等应用中去。

QNX采用微内核结构，也就是说，内核非常非常非常小。这样一方面启动速度非常快，另一方面安全性稳定性大大提高。

QNX构架是有一个微型内核，然后又包含许多相关进程。这样的好处是，即使有一个进程出错，也不会影响内核。

各个服务进程以及应用进程之间通过**内部进程通信IPC**的方式进行沟通，如下图:



漫谈QNX（架构/进程，线程，同步，进程间通信IPC）_linux_aFakeProgramer-华为云开发者联盟 (csdn.net)

QNX的微内核结构

内核独立自处于一个被保护的地址空间；驱动程序、网络协议和应用程序处于程序空间中。

> 微内核结构的优点：
> ①驱动程序、网络协议、文件系统等操作系统模块和内核相互独立，任何模块的故障都不会导致内核的崩溃；
> ②驱动程序、网络协议、文件系统和应用程序都处于程序空间，

> 都调用相同的内核API，开发与调试和应用程序没有区别；
> ③操作系统功能模块可以根据需要动态地加载或卸载，不需要编译内核。

同时，使用QNX等具有功能安全的操作系统也是目前业界所广泛认同的一个策略。

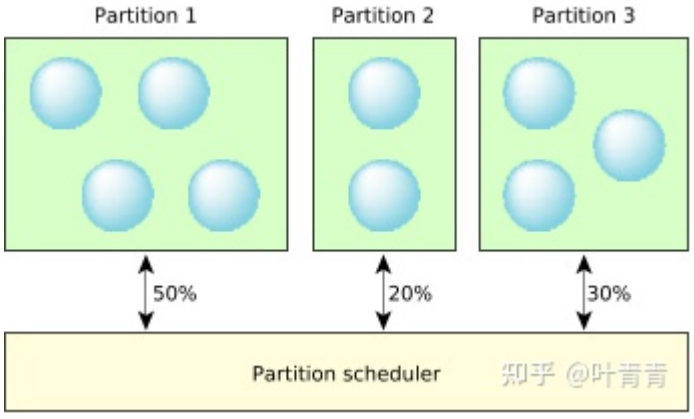
那么为什么说QNX系统更安全呢？QNX的开发者们如何通过系统组件来进行软件设计/开发呢？下面是一些QNX特性的例子：

AP(Adaptive Partitioning)

有时候们会遇到这样一种问题，某进程里有个bug，调试的时候发现该进程一起来整个一个核的CPU都满了，我们猜测系统内可能是出现了死循环。

自适应分区是QNX的重要特性之一，也是一个在开发调试阶段很好用的一个工具。

在软件集成在QNX系统之后，开始优化整个系统之前，为了保护不同的应用群组/应用，独立运行而不被其他应用破坏或干扰，操作系统采用“虚拟墙（virtual walls）”将系统的共享资源（CPU执行时间/内存/存储空间等）以一定的比例划分，以确保每个分区都有一组经过工程设计的资源，每个分区内可以运行一个或多个线程。



ap分区概念

分区能够提供：

- 内存保护：提供内存保护，即每个分区是离散的，由内存管理单元控制（MMU）；
- 过载保护：提供过载保护，即根据系统设计人员的指定，每个分区都有一段执行时间。

自适应的含义是：在运行时可以改变配置。例如，空闲时间被重新分配给其他调度程序分区，系统会使用一种机制，使得CPU可以在一个时间分区之间临时移动线程。

在QNX的系统定义中，包含着以下两个概念：

微内核（Micro Kernel）:是提供操作系统核心功能的内核精简版本。

`实时操作系统（RTOS）`：是指当外界事件和数据产生时，能够接受并以足够快的速度予以处理，其处理的结果又能在规定的时间之内，来控制生成过程或对处理系统做出快速响应，调度一切可以利用的资源完成实时任务，并控制所有实时任务协调一致运行的操作系统。

QNX的核心提供4种服务：

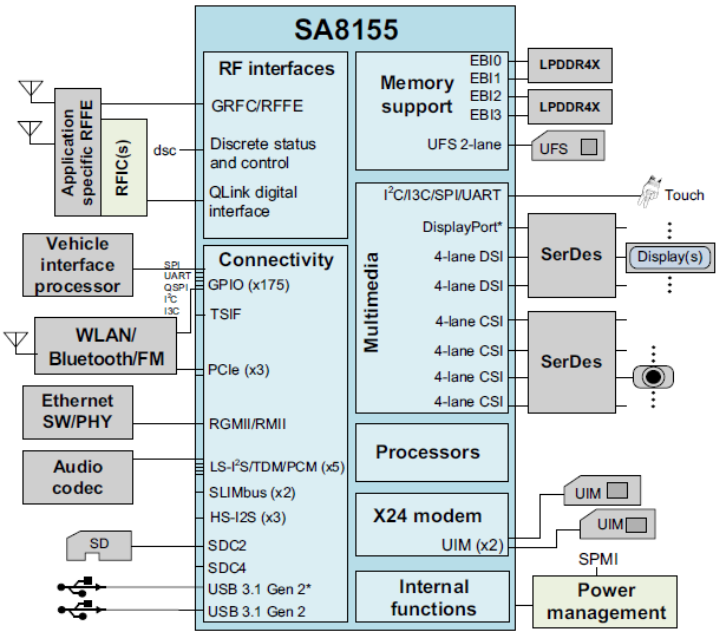
1. 进程调度
2. 进程间通信
3. 底层网络通信
4. 中断处理

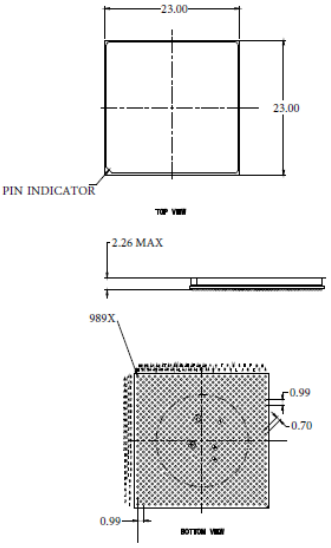
因此QNX内核非常的精致小巧，比传统的宏内核（Linux）系统可靠性更高。

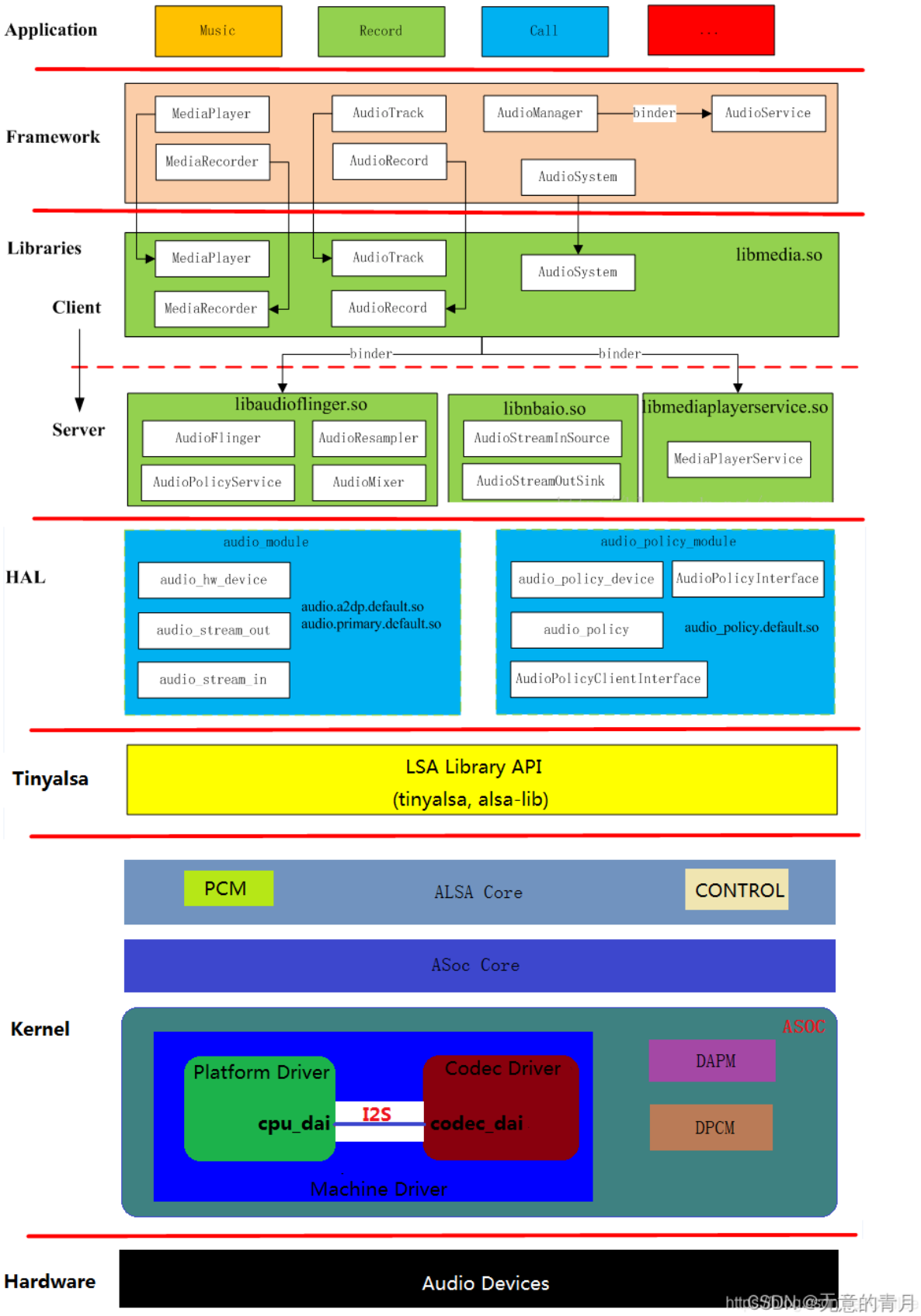
QNX调度策略

> QNX 提供POSIX.1b标准进程调度：
>
> 1. 255个进程优先级
> 2. 抢占式的、基于优先级的正文切换
> 3. 可选调度策略：FIFO、轮转策略、适应性策略

高通音频架构






Application
Music
Record
Call
...
Framework
MediaPlayer
AudioTrack
AudioManager
binder
AudioService
MediaRecorder
AudioRecord
AudioSystem
Libraries
Client
MediaPlayer
AudioTrack
AudioSystem
libmedia.so
MediaRecorder
AudioRecord
binder
binder
Server
libaudioflinger.so
AudioFlinger
AudioResampler
AudioPolicyService
AudioMixer
libnbaio.so
AudioStreamInSource
AudioStreamOutSink
libmediaplayerservice.so
MediaPlayerService
HAL
audio_module
audio_hw_device
audio_stream_out
audio_stream_in
audio.a2dp.default.so
audio.primary.default.so
audio_policy_module
audio_policy_device
AudioPolicyInterface
audio_policy
audio_policy.default.so
AudioPolicyClientInterface
Tinyalsa
LSA Library API
(tinyalsa, alsa-lib)
PCM
ALSA Core
CONTROL
ASoc Core
Kernel
ASOC
Platform Driver
cpu_dai
I2S
Codec Driver
codec_dai
Machine Driver
DAPM
DPCM
Hardware
Audio Devices
CSDN@无意的青月

一、概述

音频是几乎是任何一个机器都是必备的一项功能，从早起的单纯发声的录音机，到后来的MP3，以及到现在的手机，它一直陪伴在我们的生活中，功能不变，形式却一直在变，包括它的架构也在变化。从早期的OSS到现在的ALSA，这个介绍在上篇文档是有介绍的，这里我们就着重说一下ALSA。首先高通的音频结构分为以下几个部分：

> `应用层`，主要使用音频的用户主体
>
> `架构层(framework)`，这一层主要是为应用层提供了相关处理接口，并且链接了HAL层
>
> `硬件抽象(HAL)层`，在音频开发中可能大部分主要逻辑都是放在这个层次来处理，链接了 framework层和kernel层，
>
> 这里面还包含了ALSA库用于链接ALSA驱动
>
> `内核(kernel)层`，链接硬件的驱动程序
>
> `硬件`，包含MODEM，CODEC，ADSP

Android 音频(Audio)架构_android audio-CSDN博客

高通音频架构（一）-CSDN博客

高通音频架构（二）_高通音频架构(二)-CSDN博客

高通音频架构（三）_adsp-CSDN博客

音频处理——详解PCM数据格式_pcm格式-CSDN博客

AVAS

[让安全听得见 | APM32F103RCT7电动汽车AVAS应用方案_apm32f103vct7-CSDN博客]

DSP

[dsp音频处理器，ADSP_adsp和dsp区别-CSDN博客]

车上的DSP的意思是数字音频处理器，DSP是英文DigitalSignalProcessing的缩写，翻译成中文是数字信号处理，DSP的作用就是控制频响，可以达到对音频信号进行一定的处理、修饰作用。例如听歌时，有一些低音、流行、摇滚、布鲁斯等不同选项，选了之后歌的调和声音就会不一样。数字信号处理是利用数字处理，通过计算机或更专业的数字信号处理器，来执行各种各样的信号处理操作，以这种方式处理的信号是表示时域、空域或频域中连续变量样本的一系列数字。

https://www.pcauto.com.cn/jxwd/3508/35089051.html

DSP是数字音频处理器，可以对音频信号做一定修改。音频信号来自音源，车载主机、CD机、导航等。把这些设备的音频信号提取出来，经过DSP处理器，加以修饰处理，处理后的信号送入功放机放大，功放机把音频信号放大最后推动喇叭发声。

(633条消息) 高通音频架构（三）*adsp*无意的青月的博客-CSDN博客

一、Kernel层

音频由于其特殊的工作，使得它的结构特别的复杂，而且在自己的结构基础上还引入了ALSA架构，不过在android系统上所引入的并非完整的ALSA架构而是精简版的tinyalsa，但是就算精简版也是内容相当丰厚。除此，音频还拥有自己的单独的处理器ADSP以及独立的电源管理系统DAPM(便携式动态音频电源管理），使得音频在任

何时候都是以最低功耗运行，降低了便携设备的功耗。在某些播放场景甚至不需要CPU的介入，比如接打电话的通过音频，如果手机处于休眠可以不需要唤醒CPU直接传递语音数据。要想知道整个过程中音频数据的流转需要一步步去了解，音频架构中所涉及到的各个部分，缺一环则不可，先看看ALSA架构。

BSP

BSP_bsp测试-CSDN博客

*BSP*即*Board Support Package*，板级支持包。它来源于嵌入式操作系统与硬件无关的设计思想，操作系统被设计为运行在虚拟的硬件平台上。对于具体的硬件平台，与硬件相关的代码都被封装在*BSP*中，由*BSP*向上提供虚拟的硬件平台，*BSP*与操作系统通过定义好的接口进行交互。****BSP****是所有与硬件相关的代码体的集合****。

****BSP**在嵌入式系统中的角色，很相似于在**PC**系统中的**BIOS*****和驱动程序的***地位****。

****定义****

*BSP*就是为软件操作系统正常运行提供最基本、最原始的硬件操作的软件模块，它和操作系统息息相关，但又不属于操作系统的一部分。*BSP*可以分为三大部分：

1：系统上电时的硬件初始化。

2：为操作系统访问硬件驱动程序提供支持。

3：集成的硬件相关和硬件无关的操作系统所需的软件模块。

*Ø**BSP******的表现形式****

*BSP*主要以两种形式来表现：

1：源代码（*C*代码、汇编代码）、系统编译连接依靠文件。

2：二进制的目标代码和目标代码库。

> ****BSP****向上层提供的接口有****
>
> *l* 与操作系统内核的接口（如报告*DRAM*大小、修改中断屏蔽级别等）
>
> *l* 与操作系统的*I/O*系统的接口
>
> *l* 与应用程序的接口

Ø**BSP******的主要功能****

*BSP*的主要功能在于配置系统硬件使其工作于正常的状态，完成硬件与软件之间的数据交互，为*OS*及上层应用程序提供一个与硬件无关的软件平台。因此从***执行角度***来说，其可以分为两大部分：

1）目标板启动时的硬件初始化及多任务环境的初始化

2）目标板上控制各个硬件设备正常运行的设备驱动程序，由它来完成硬件与软件之间的信息交互

通常我们认为*BSP*是***为*****OS******服务****的，但实际上，*BSP*软件包中的部分程序对*OS*也并不是必须的，从这个角度，又可以将*BSP*划分为两部分：

1）最小系统*BSP*，即我们通常所称的*BSP*

2) 设备驱动程序

FDbus

FDBus：高速分布式总线以及中间件开发框架_fdbus 自动驾驶-CSDN博客

1 FDBus简介

FDBus 基于 Socket (TCP 和 Unix domain) 之上的IPC机制, 采用 Google protobuf 做序列化和反序列化。 FDBus还支持字符串形式的名字作为server地址。通过 name server 自动为 server 分配 Unix domain 地址和 TCP 端口号, 实现 client 和server 之间用服务名字寻址。

> 一句话描述：FDBus (Fast Distributed Bus) 是一种 IPC 机制, 用于进程间通信。

进程间通信（IPC）介绍_ipc协议-CSDN博客

特点：
分布式：基于TCP socket和Unix Domain socket（UDS），既可用于本地IPC，也支持网络主机之间的IPC；
跨平台：目前已在Windows，Linux和QNX上验证；
高性能：点对点直接通信，不通过中央Hub或Broker转发；
安全性：能够为server的方法调用也事件广播配置不同等级的访问权限，只有权限足够高的client才能特点方法和接收特定事件；
服务名解析：server地址用名字标识，通过name server注册服务和解析名字，从而server可以在全网络任意部署。

2 车载Android设备间通讯

我们知道车载Android设备不止一台，如果我们要在中控和仪表盘这两个Android系统设备之间要进行通信怎么实现呢？同学们思考一下

不同Android设备间的通信已经无法通过Binder、Broadcast、Service来完成，所以我们是否可以参考FDBus通过Socket来进行不同Android设备的通讯呢？下面我们来看看大致思路：

通过Socket+动态代理实现简单的跨设备通讯:

1.YFDBus库中分为IPCNameManager类：实现客户端Socket的管理，动态代理的相关的逻辑；NameServerManager类：服务端的Socket管理。
2.Server模块为服务端，依赖YFDBus库，初始化时服务注册到IPCNameManager中，提供自定义的服务。接收到客户端Socket发送过来的消息后，将该消息解析做出响应返回对应的数据给客户端。
3.Client模块为客户端，依赖YFDBus库，通过Socket实现对Server端的服务发现，确实是数据传输后使用动态代理解析来自Server的响应数据，得到的服务对象。然后通过该对象进行服务调用，确实就是Socket发送数据到Server做处理然后给出响应，来实现服务的发现和调用。

最简单的理解就是：

1.通过Socket客户端向服务端发送消息，实现服务注册、服务发现、服务调用。

2.通过动态代理的方式获取到服务端的类及方法在客户端进行调用。

CAN

ourdev_553420.pdf (amobbs.com)