ATOM_LINKER 技术手册

2021/4/20 V1.1

更新记录：2021/4/18 V1.0 系统状态的定义

2021/4/20 V1.1 数据包的结构阐述

ATOM_LINKER 一种适用于 IIC，UART，SPI 通讯方式的小端数据包结构，可扩展性强，部署快。目前为 v1.0 版本,参考工业传感器数据设计框架，今后目标为一种可变结构的数据包快速部署框架。目前公布状态：未公开公布。预计在完成目标申请软著后公布。

目录

1. 导言

1.1 目标

西北工业大学足球机器人基地的核心成果为 HAWKING 系列足球全向机器人，该机器人目前虽已完成初步的体系搭建但仍有许多不成熟之处。其中一点便是其上下位机通讯的不可靠，不可控与不可拓展性。为了解决这个问题，本项目推出 ATOM_LINKER 协议为上下位机与单片机传感器间的可靠通讯做好准备。

本手册专为 ATOM_LINKER 的数据包结构与 API 协议提供说明指导。手册描述了使用 Atom 通信协议与 单片机与上位机进行连接的方法和细节，并提供了示例代码。

1.2 目标人群

本手册目标人群为进行机器人与嵌入式开发的设计人员。软件工作人员。本手册也可用于指导学习调试更改 ATOM_LINKER 协议的工程师。

1.3 手册范围

从工程师角度本文可以分为以下几个部分。

1. 介绍一种通过底层协议进行通信的方法
2. 解释上下位机通讯代码中的所有 API
3. 扩展空间与改动建议

本手册总结了有关数据包的所有技术设计，旨在使读者能快速结合代码部署 ATOM_LINKER 协议。如果对细节具有疑问或者改进意见，欢迎联系：1361505905@qq.com

1.4 缩写词，专业术语与缩略词表

术语	描述
Hz	Hertz
DRDY	Data Ready
AHRS	Attitude&Heading Referance System

1.5 文档格式

正文格式： 我是正文

引用文字： “我是引用”

状态/标志位： 滚起来写代码

命令： *倒立*

例程： **#define tty 有钱**

2. 示例代码包说明

2.1 示例代码包结构

提供的示例代码为能够在不同平台中编译的项目文件，包含 Measure_Example 文件夹：

- Linux_Demo：Linux 平台 Demo，由 ubuntu 18.04 系统编译和运行。包括：

  MeasureDataDemo 文件夹中为 Linux 环境下获取下位机数据，建立通讯的例程。

- Windows_Demo：Windows 平台 Demo，由 VS2019 环境编译运行，包括 MeasureDataDemo 文件夹中为 windows 下获取下位机数据，建立通讯的例程。

- STM32_Demo：STM32 平台 Demo，由 IAR 8 在 Windows10 系统上编译并在 STM32F411 环境下运行，包括：

MeasureDataDemo 文件夹，分为下面几个工程：

STM32 Uart 版，由 IAR 在 Windows10 系统上编译并在 STM32F411 环境下运行，演示了如何利用 Uart 获取下位机数据，建立通讯。

STM32 IIC 版，由 IAR 在 Windows10 系统上编译并在 STM32F411 环境下运行，此版本分为两个子版本，使用 DRDY 和不使用 DRDY 版本，演示了如何利用 IIC 去使用 ATOM_LINKER 的底层协议与设备通信。

STM32 SPI 版，由 IAR 在 Windows10 系统上编译并在 STM32F411 环境下运行，此版本为使用 DRDY 的版本（SPI 只能使用 DRDY 进行通信），演示了如何利用 SPI 去使用 ATOM_LINKER 的底层协议与设备通信。

SDK 文件夹：

ROS 文件夹：

- catkin_ws 文件夹是 ROS 和模组通信的例程。ROS 版本为 Kinetic Kame，环境为 Ubuntu 18.04，64 位操作系统

ROS 例程使用方法： 1. 打开新终端进入 catkin_ws；创建工作空间，将 imu_publish 中的源码放入对应工作空间中；详细步骤参考 ReadMe.txt 运行命令 catkin_make 编译产生可执行文件“Atom”；编译结束后运行 roscore。

2. 打开新终端进入 catkin_ws；执行命令 source devel/setup.bash 给串口添加权限 sudo chmod 666 /dev/ttyUSB0 (代码写的是 ttyUSB0,如果需要更改，可在代码中更改)；执行命令 rosrun Atom_publish Atom(Atom_publish 是 package 的名字,Atom 是可执行文件名)。

3. 打开新终端进入 catkin_ws；执行命令 source devel/setup.bash；执行命令 rostopic echo /Atom_data 可以看见发布在 topic 的 Atom_data。

3. 系统结构

3.1 系统概述

为了能使上位机轻松快捷的与下位机通讯，在示例文件夹中提供了 IIC，UART 与 SPI 三种硬件接口供选择。UART 接口，SPI 和 IIC 接口可以通过 API 进行配置。

以下是应用层（ATOM 攻设备）与 ATOM 从属设备连接架构图。

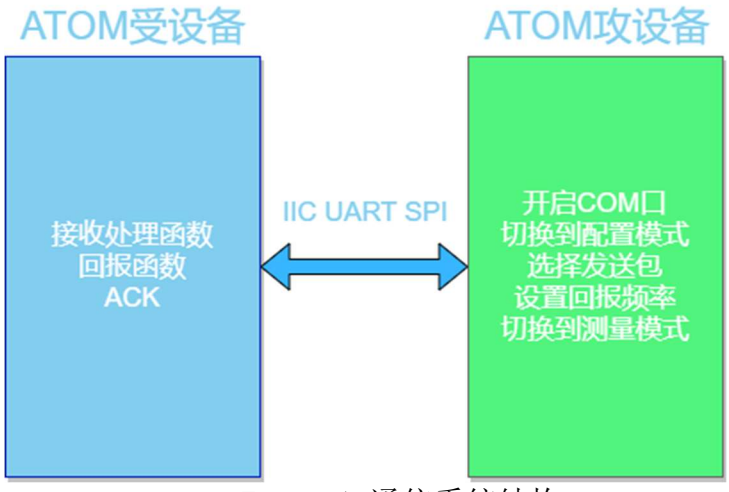

Figure1-通信系统结构

3.2 设置设备系统状态

ATOM_LINKER 定义了几种系统状态以方便用户从一种状态模式切换到另一种状态模式。总共定义了 5 种系统状态：WAKEUP、CONFIG、MEASURE、SELFTEST、SENSOR_CAL（传感器校准）。“Figure 2”是 ATOM_LINKER 协议机系统状态转换流程图，它显示了如何通过响应不同的命令从一个状态切换到另一个状态。

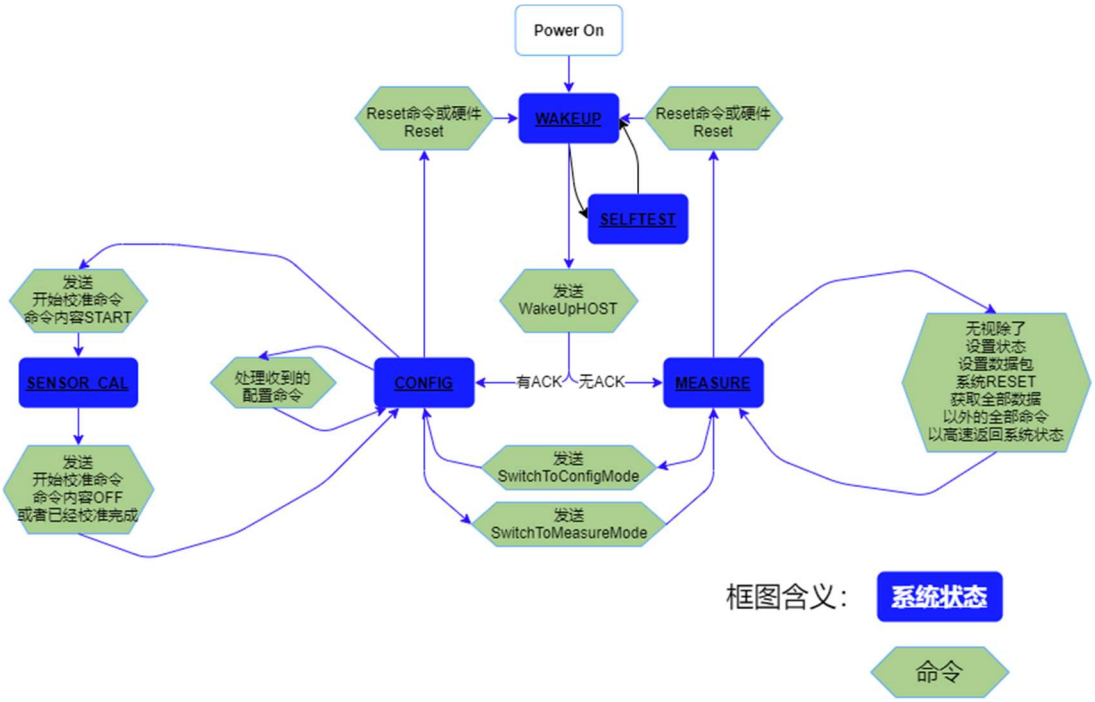

Figure 2 系统状态框图

1.2.1 WAKEUP 状态

当下位机启动时，它将运行在 WAKEUP 状态。以下程序将在 WAKEUP 状态下执行:

- 硬件参数初始化。
- 变量参数初始化。
- 算法参数初始化。
- 操作系统初始化。
- 转入 SELFTEST 状态进行自检测。
- 参数初始化。
- 发送 *WakeUpHost* 命令。

1.2.2 CONFIG 状态

当下位机从主机接收到 *SwitchToConfigMode* 命令时，它将切换到 CONFIG 状态，在该状态下，下位机将处理由主机发送的命令，并向主机发送确认响应。

1.2.3 MEASURE 状态

当下位机从主机收到 *SwitchToMeasureMode* 命令时，它将执行以下操作：

- 从传感器获取原始数据。

- 根据算法处理原始数据(如校准/卡尔曼滤波器设置)。
- 准备数据并通过接口(UART，SPI 或 IIC)将数据发送

1.2.4 SELFTEST 状态

下位机在 WAKEUP 状态中完成参数初始化后，如果配置选择进行自检将会进入 SELFTEST 状态，进行电机状态，传感器状态的自检测。自检成功完成后，回到 WAKEUP 状态。

1.2.5 SENSOR_CAL 状态

SetSensorCalibrate 命令控制下位机进入或退出 SENSOR_CAL 状态，进行电机&传感器的校准。在校准完成后自动退出该状态。（或者发送 OFF 命令取消该状态）。

4. 协议描述

4.1 消息通用结构

该协议是用于上位机与目标下位机之间的一组消息，虽然上位机的 USB 接口通常统一通用，但是下位机由于资源限制有着 UART，IIC，SPI 等多种通信接口，为此理解好该协议包的底层结构是非常重要的。

该通信协议的数据包传输统一采用小端格式（Little-Endian）传输。

以下为该通信协议的结构，此结构定义了主机与下位机通信数据包每个字段的详细含义，数据包结构如图。（Figure-3）

通用消息结构

第0字节	第1字节	第2字节	第3字节	第4字节	第5字节	第6字节	第7字节	第8-n字节	第n+1字节	第n+2字节
前导码		目标地址	操作类型	信息类型	数据长度	超长数据长度		数据包内容	异或校验	包尾
'A'	'X'	0xFF或地址	0xXX	0xXX	0~0xFE或0xFF	低八位	高八位	阿巴阿巴阿巴阿巴阿巴阿巴	啊？巴！	'm'

Figure-3 通用消息结构

4.2 消息头字段

头字段与包头（前导码）含义不同，在 ATOM_LINKER 下，每个头字段占八个字节，包括：1.前导码 2.目标地址 3.操作类型 4.信息类型 5.数据长度 6.超长数据长度，**Figure-3 中头字段均标红**。

头字段的每个子节段的详细说明见下。

4.2.1 前导码（Preamble）

前导码起到包头的作用，共由两个字节组成，第一字节为’A’（0x41），第二字节为’x’（0x78），A 与 x 在 16 进制上有着较大区别，基本不可能作为数据包内容出现且方便记忆，故使用 A x 两字符作为数据包开始的象征。

4.2.2 目标地址（MADDR）

ATOM_LINKER 设备地址标识符，若为 0xFF 代表该条消息为广播形式。下位机（ATOM 受设备）响应上位机命令时目标地址位为自己的地址。

综上所述，所有 ATOM 受设备的地址应该设定为 0x00 到 0xFE 之间。同时，上位机可通过 *SetModuleAddress* 命令配置下位机的地址，原则上，ATOM 设备的地址应该储存于 Flash 中方便调用更改，降低耦合。

4.2.3 操作类型（CID）

CID 占一个字节，用于区分不同的消息组，用于从机或主机发送时描述正确的类别 ID。详细组成见（Figure-4）

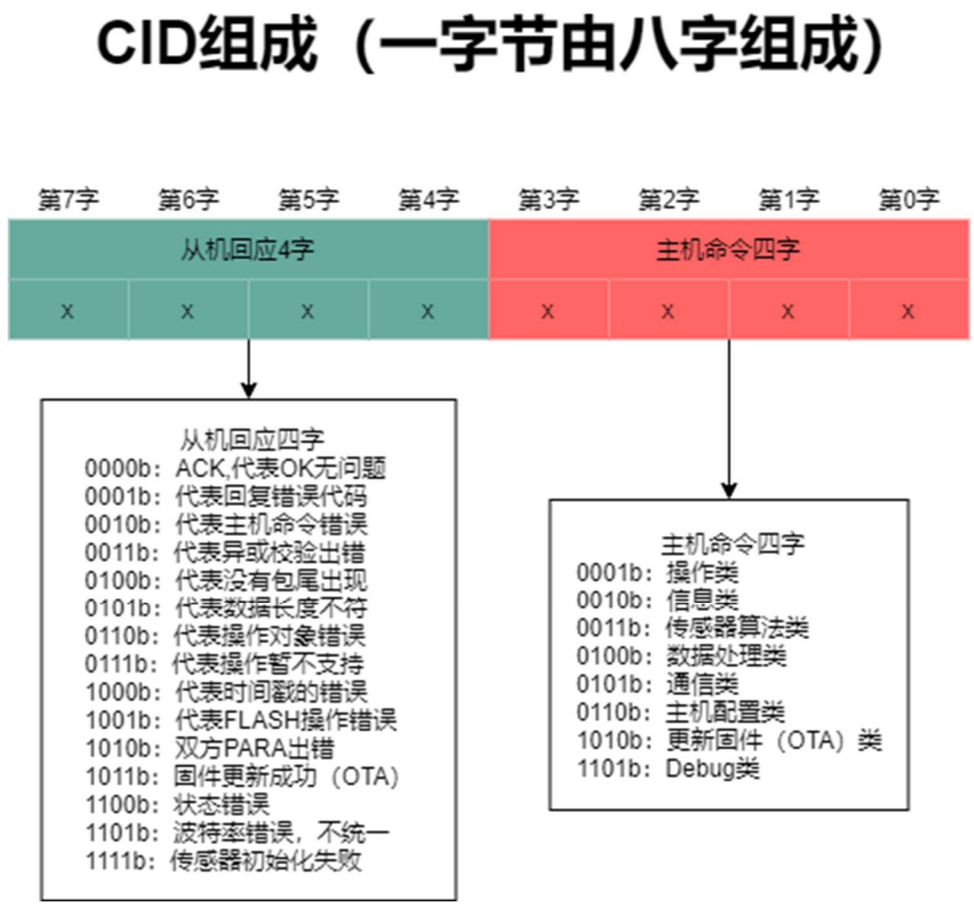

Figure-4 CID 字节组成

在响应消息时用最高 4 位来标识 ERROR 状态，如果最高 4 位为 0，则表

示在相应操作中没有发现错误，其他情况下，则说明有错误发生。

4.2.4 信息类型（MID）

信息类型用于区分特定 CID 下的不同命令

4.2.5 数据长度（PL）

数据长度占一个字节，0xFF（255）代表有效荷载过长，需要使用额外数据长度表示，若数据包长度小于 255，则应为 0-0xFE 之间的一个数，特别的，当 PL 为 0 时，表示数据长度为空。

4.2.6 额外数据长度（SLP）

该字段占用两个字节，表示数据包的总长度，其值应在 255-1024 之间。SLP 是可选字段，只有在有效载荷长度等于 0xFF 时才有效（其余时刻解包时应直接跳过该字段），这意味着有效载荷字节超过了正常的数据包长度 0-254，那么它需要超长数据包在 255-1024 发送有效载荷。

需要注意的是，额外数据长度由高八位与第八位组成，第一字节代表低八位（DATAL），第二字节代表高八位（DATAH）。

4.3 异或校验

本数据包结构使用异或校验来产生一位的 BCC 数据。校验字节产生过程见下。

```
u8   BCC_Correct(u8 *addr, u16 len)
{
 unsigned char *DataPoint;
 DataPoint = addr;
 unsigned char XorData = 0;
 unsigned short DataLength = len;

 while (DataLength--)
 {
  XorData = XorData ^ *DataPoint;
  DataPoint++;
 }

 return XorData;
}
```

4.4 包尾

与前导码对应，包尾为一固定字节’m’。