嵌入式系统实验报告

**李在弦 2015080121**

[**15510380063@163.com**](mailto:15510380063@163.com)

**张天玮 2015013243**

[**thss15\_zhangtw@163.com**](mailto:thss15_zhangtw@163.com)

## 实验环境

Ubuntu 16.04版本操作系统 ，TinyOS 2.1.2

## 实验任务

（1）编程控制 Telosb 节点驱动 STM32 主控小车电机实现小车运动，包括前进、后退、左转、右转、停止。

（2）编程控制 Telosb 节点驱动 STM32 主控小车机械臂操作，包括机械臂上升、下降、左转、右转、归位。

（3）实现小车实时显示当前运动指令,通过 LED 灯不同点亮组合来表示运动指令。

（4）实现控制手柄控制小车完成运动，用摇杆控制小车前进、后退、左转、右转、停止，通过按钮控制机械臂上升、下降、左转、右转、归位。

## 设计思路

**手柄**

手柄部分各个模块的使用和连接方式与助教给出的大致相同：



手柄部分主模块BlinkToRadio使用一个每100ms 发射一次的时钟来周期性地采集每个按键和摇杆两个轴的度数，并根据不同的度数和按键时候按下来给小车发一个uint16\_t类型的指令。

**在JoyStickC.nc文件中定义JoyStick的通用类。当计时器触发事件的时候，在BlinkToRadioC.nc 通过接口Read<uint16\_t>中的read命令来读取摇杆输入。在JoyStickC.nc中定义两个组件AdcReadClientC作为X，Y轴ADC的输入。在JoyStickP.nc中定义模块JoyStickP，该模块提供两个接口AdcConfigure<const msp430adc12\_channel\_config\_t\*>，作为X,Y轴的配置，将AdcReadClientC的对应接口AdcConfigure连接到JoyStickP的对应接口上。BlinkToRadio通过接口Read<uint16\_t>来连接JoyStickC中的AdcReadClientC，AdcClientC的AdcConfigure接口连接到JoyStickP模块的AdcConfigure，在JoyStickP中定义了命令getConfiguration配置对应adc。在JoyStickC.nc文件中定义JoyStick的通用类。当计时器触发事件的时候，在BlinkToRadioC.nc 通过接口Read<uint16\_t>中的read命令来读取摇杆输入。在JoyStickC.nc中定义两个组件AdcReadClientC作为X，Y轴ADC的输入。在JoyStickP.nc中定义模块JoyStickP，该模块提供两个接口AdcConfigure<const msp430adc12\_channel\_config\_t\*>，作为X,Y轴的配置，将AdcReadClientC的对应接口AdcConfigure连接到JoyStickP的对应接口上。BlinkToRadio通过接口Read<uint16\_t>来连接JoyStickC中的AdcReadClientC，AdcClientC的AdcConfigure接口连接到JoyStickP模块的AdcConfigure，在JoyStickP中定义了命令getConfiguration配置对应adc。在JoyStickC.nc文件中定义JoyStick的通用类。当计时器触发事件的时候，在BlinkToRadioC.nc 通过接口Read<uint16\_t>中的read命令来读取摇杆输入。在JoyStickC.nc中定义两个组件AdcReadClientC作为X，Y轴ADC的输入。在JoyStickP.nc中定义模块JoyStickP，该模块提供两个接口AdcConfigure<const msp430adc12\_channel\_config\_t\*>，作为X,Y轴的配置，将AdcReadClientC的对应接口AdcConfigure连接到JoyStickP的对应接口上。BlinkToRadio通过接口Read<uint16\_t>来连接JoyStickC中的AdcReadClientC，AdcClientC的AdcConfigure接口连接到JoyStickP模块的AdcConfigure，在JoyStickP中定义了命令getConfiguration配置对应adc。在JoyStickC.nc文件中定义JoyStick的通用类。当计时器触发事件的时候，在BlinkToRadioC.nc 通过接口Read<uint16\_t>中的read命令来读取摇杆输入。在JoyStickC.nc中定义两个组件AdcReadClientC作为X，Y轴ADC的输入。在JoyStickP.nc中定义模块JoyStickP，该模块提供两个接口AdcConfigure<const msp430adc12\_channel\_config\_t\*>，作为X,Y轴的配置，将AdcReadClientC的对应接口AdcConfigure连接到JoyStickP的对应接口上。BlinkToRadio通过接口Read<uint16\_t>来连接JoyStickC中的AdcReadClientC，AdcClientC的AdcConfigure接口连接到JoyStickP模块的AdcConfigure，在JoyStickP中定义了命令getConfiguration配置对应adc。在JoyStickC.nc文件中定义JoyStick的通用类。当计时器触发事件的时候，在BlinkToRadioC.nc 通过接口Read<uint16\_t>中的read命令来读取摇杆输入。在JoyStickC.nc中定义两个组件AdcReadClientC作为X，Y轴ADC的输入。在JoyStickP.nc中定义模块JoyStickP，该模块提供两个接口AdcConfigure<const msp430adc12\_channel\_config\_t\*>，作为X,Y轴的配置，将AdcReadClientC的对应接口AdcConfigure连接到JoyStickP的对应接口上。BlinkToRadio通过接口Read<uint16\_t>来连接JoyStickC中的AdcReadClientC，AdcClientC的AdcConfigure接口连接到JoyStickP模块的AdcConfigure，在JoyStickP中定义了命令getConfiguration配置对应adc。在JoyStickC.nc文件中定义JoyStick的通用类。当计时器触发事件的时候，在BlinkToRadioC.nc 通过接口Read<uint16\_t>中的read命令来读取摇杆输入。在JoyStickC.nc中定义两个组件AdcReadClientC作为X，Y轴ADC的输入。在JoyStickP.nc中定义模块JoyStickP，该模块提供两个接口AdcConfigure<const msp430adc12\_channel\_config\_t\*>，作为X,Y轴的配置，将AdcReadClientC的对应接口AdcConfigure连接到JoyStickP的对应接口上。BlinkToRadio通过接口Read<uint16\_t>来连接JoyStickC中的AdcReadClientC，AdcClientC的AdcConfigure接口连接到JoyStickP模块的AdcConfigure，在JoyStickP中定义了命令getConfiguration配置对应adc。在JoyStickC.nc文件中定义JoyStick的通用类。当计时器触发事件的时候，在BlinkToRadioC.nc 通过接口Read<uint16\_t>中的read命令来读取摇杆输入。在JoyStickC.nc中定义两个组件AdcReadClientC作为X，Y轴ADC的输入。在JoyStickP.nc中定义模块JoyStickP，该模块提供两个接口AdcConfigure<const msp430adc12\_channel\_config\_t\*>，作为X,Y轴的配置，将AdcReadClientC的对应接口AdcConfigure连接到JoyStickP的对应接口上。BlinkToRadio通过接口Read<uint16\_t>来连接JoyStickC中的AdcReadClientC，AdcClientC的AdcConfigure接口连接到JoyStickP模块的AdcConfigure，在JoyStickP中定义了命令getConfiguration配置对应adc。在JoyStickC.nc文件中定义JoyStick的通用类。当计时器触发事件的时候，在BlinkToRadioC.nc 通过接口Read<uint16\_t>中的read命令来读取摇杆输入。在JoyStickC.nc中定义两个组件AdcReadClientC作为X，Y轴ADC的输入。在JoyStickP.nc中定义模块JoyStickP，该模块提供两个接口AdcConfigure<const msp430adc12\_channel\_config\_t\*>，作为X,Y轴的配置，将AdcReadClientC的对应接口AdcConfigure连接到JoyStickP的对应接口上。BlinkToRadio通过接口Read<uint16\_t>来连接JoyStickC中的AdcReadClientC，AdcClientC的AdcConfigure接口连接到JoyStickP模块的AdcConfigure，在JoyStickP中定义了命令getConfiguration配置对应adc。在JoyStickC.nc文件中定义JoyStick的通用类。当计时器触发事件的时候，在BlinkToRadioC.nc 通过接口Read<uint16\_t>中的read命令来读取摇杆输入。在JoyStickC.nc中定义两个组件AdcReadClientC作为X，Y轴ADC的输入。在JoyStickP.nc中定义模块JoyStickP，该模块提供两个接口AdcConfigure<const msp430adc12\_channel\_config\_t\*>，作为X,Y轴的配置，将AdcReadClientC的对应接口AdcConfigure连接到JoyStickP的对应接口上。BlinkToRadio通过接口Read<uint16\_t>来连接JoyStickC中的AdcReadClientC，AdcClientC的AdcConfigure接口连接到JoyStickP模块的AdcConfigure，在JoyStickP中定义了命令getConfiguration配置对应adc。在JoyStickC.nc文件中定义JoyStick的通用类。当计时器触发事件的时候，在BlinkToRadioC.nc 通过接口Read<uint16\_t>中的read命令来读取摇杆输入。在JoyStickC.nc中定义两个组件AdcReadClientC作为X，Y轴ADC的输入。在JoyStickP.nc中定义模块JoyStickP，该模块提供两个接口AdcConfigure<const msp430adc12\_channel\_config\_t\*>，作为X,Y轴的配置，将AdcReadClientC的对应接口AdcConfigure连接到JoyStickP的对应接口上。BlinkToRadio通过接口Read<uint16\_t>来连接JoyStickC中的AdcReadClientC，AdcClientC的AdcConfigure接口连接到JoyStickP模块的AdcConfigure，在JoyStickP中定义了命令getConfiguration配置对应adc。在JoyStickC.nc文件中定义JoyStick的通用类。当计时器触发事件的时候，在BlinkToRadioC.nc 通过接口Read<uint16\_t>中的read命令来读取摇杆输入。在JoyStickC.nc中定义两个组件AdcReadClientC作为X，Y轴ADC的输入。在JoyStickP.nc中定义模块JoyStickP，该模块提供两个接口AdcConfigure<const msp430adc12\_channel\_config\_t\*>，作为X,Y轴的配置，将AdcReadClientC的对应接口AdcConfigure连接到JoyStickP的对应接口上。BlinkToRadio通过接口Read<uint16\_t>来连接JoyStickC中的AdcReadClientC，AdcClientC的AdcConfigure接口连接到JoyStickP模块的AdcConfigure，在JoyStickP中定义了命令getConfiguration配置对应adc。在JoyStickC.nc文件中定义JoyStick的通用类。当计时器触发事件的时候，在BlinkToRadioC.nc 通过接口Read<uint16\_t>中的read命令来读取摇杆输入。在JoyStickC.nc中定义两个组件AdcReadClientC作为X，Y轴ADC的输入。在JoyStickP.nc中定义模块JoyStickP，该模块提供两个接口AdcConfigure<const msp430adc12\_channel\_config\_t\*>，作为X,Y轴的配置，将AdcReadClientC的对应接口AdcConfigure连接到JoyStickP的对应接口上。BlinkToRadio通过接口Read<uint16\_t>来连接JoyStickC中的AdcReadClientC，AdcClientC的AdcConfigure接口连接到JoyStickP模块的AdcConfigure，在JoyStickP中定义了命令getConfiguration配置对应adc。在JoyStickC.nc文件中定义JoyStick的通用类。当计时器触发事件的时候，在BlinkToRadioC.nc 通过接口Read<uint16\_t>中的read命令来读取摇杆输入。在JoyStickC.nc中定义两个组件AdcReadClientC作为X，Y轴ADC的输入。在JoyStickP.nc中定义模块JoyStickP，该模块提供两个接口AdcConfigure<const msp430adc12\_channel\_config\_t\*>，作为X,Y轴的配置，将AdcReadClientC的对应接口AdcConfigure连接到JoyStickP的对应接口上。BlinkToRadio通过接口Read<uint16\_t>来连接JoyStickC中的AdcReadClientC，AdcClientC的AdcConfigure接口连接到JoyStickP模块的AdcConfigure，在JoyStickP中定义了命令getConfiguration配置对应adc。在JoyStickC.nc文件中定义JoyStick的通用类。当计时器触发事件的时候，在BlinkToRadioC.nc 通过接口Read<uint16\_t>中的read命令来读取摇杆输入。在JoyStickC.nc中定义两个组件AdcReadClientC作为X，Y轴ADC的输入。在JoyStickP.nc中定义模块JoyStickP，该模块提供两个接口AdcConfigure<const msp430adc12\_channel\_config\_t\*>，作为X,Y轴的配置，将AdcReadClientC的对应接口AdcConfigure连接到JoyStickP的对应接口上。BlinkToRadio通过接口Read<uint16\_t>来连接JoyStickC中的AdcReadClientC，AdcClientC的AdcConfigure接口连接到JoyStickP模块的AdcConfigure，在JoyStickP中定义了命令getConfiguration配置对应adc。小车 在JoyStickC.nc文件中定义JoyStick的通用类。当计时器触发事件的时候，在BlinkToRadioC.nc 通过接口Read<uint16\_t>中的read命令来读取摇杆输入。在JoyStickC.nc中定义两个组件AdcReadClientC作为X，Y轴ADC的输入。在JoyStickP.nc中定义模块JoyStickP，该模块提供两个接口AdcConfigure<const msp430adc12\_channel\_config\_t\*>，作为X,Y轴的配置，将AdcReadClientC的对应接口AdcConfigure连接到JoyStickP的对应接口上。BlinkToRadio通过接口Read<uint16\_t>来连接JoyStickC中的AdcReadClientC，AdcClientC的AdcConfigure接口连接到JoyStickP模块的AdcConfigure，在JoyStickP中定义了命令getConfiguration配置对应adc。**

小车部分各个模块的使用和连接方式也与助教给出的大致相同：



控制信息处理的流程如下：

* BlinkToRadioC模块通过无线电接收到来自手柄的信息后，首先通过接口让CarC调用HplMsp430Usart接口的request函数获取资源。
* 成功后会启动granted事件，在该事件中对进行串口信息写入的初始化，并将成功的信号发送到BlinkToRadioC模块。
* 接着BlinkToRadioC模块会根据信息指令数据域的不同数值，调用Car接口的不同函数来控制CarC向串口写入不同命令。
* 接到不同指令的CarC通过调用接口HplMsp430Usart中的tx函数来发送操控小车的信息，一次发送一个byte数据，每次发完需要通过函数isTxEmpty来检查是否空闲，再去发送下一个byte。

这样就可以实现对小车的控制

## 遇到的问题和解决方案

* 问题一：在代码逻辑正常的情况下，为什么小车无法收到手柄的控制信息？

解决方案：测试中发现某个信号一直被往小车发送，干扰我们通过手柄操作发送信号的 -> 发现某个按钮是坏了的（按钮D）,所以屏蔽按钮D之后就可以正常地发送信号，而且小车也正常地接收信号。

测试中发现某个信号一直被往小车发送，干扰我们通过手柄操作发送信号的 -> 发现某个按钮是坏了的（按钮D）,所以屏蔽按钮D之后就可以正常地发送信号，而且小车也正常地接收信号。测试中发现某个信号一直被往小车发送，干扰我们通过手柄操作发送信号的 -> 发现某个按钮是坏了的（按钮D）,所以屏蔽按钮D之后就可以正常地发送信号，而且小车也正常地接收信号。测试中发现某个信号一直被往小车发送，干扰我们通过手柄操作发送信号的 -> 发现某个按钮是坏了的（按钮D）,所以屏蔽按钮D之后就可以正常地发送信号，而且小车也正常地接收信号。测试中发现某个信号一直被往小车发送，干扰我们通过手柄操作发送信号的 -> 发现某个按钮是坏了的（按钮D）,所以屏蔽按钮D之后就可以正常地发送信号，而且小车也正常地接收信号。测试中发现某个信号一直被往小车发送，干扰我们通过手柄操作发送信号的 -> 发现某个按钮是坏了的（按钮D）,所以屏蔽按钮D之后就可以正常地发送信号，而且小车也正常地接收信号。测试中发现某个信号一直被往小车发送，干扰我们通过手柄操作发送信号的 -> 发现某个按钮是坏了的（按钮D）,所以屏蔽按钮D之后就可以正常地发送信号，而且小车也正常地接收信号。测试中发现某个信号一直被往小车发送，干扰我们通过手柄操作发送信号的 -> 发现某个按钮是坏了的（按钮D）,所以屏蔽按钮D之后就可以正常地发送信号，而且小车也正常地接收信号。测试中发现某个信号一直被往小车发送，干扰我们通过手柄操作发送信号的 -> 发现某个按钮是坏了的（按钮D）,所以屏蔽按钮D之后就可以正常地发送信号，而且小车也正常地接收信号。测试中发现某个信号一直被往小车发送，干扰我们通过手柄操作发送信号的 -> 发现某个按钮是坏了的（按钮D）,所以屏蔽按钮D之后就可以正常地发送信号，而且小车也正常地接收信号。测试中发现某个信号一直被往小车发送，干扰我们通过手柄操作发送信号的 -> 发现某个按钮是坏了的（按钮D）,所以屏蔽按钮D之后就可以正常地发送信号，而且小车也正常地接收信号。测试中发现某个信号一直被往小车发送，干扰我们通过手柄操作发送信号的 -> 发现某个按钮是坏了的（按钮D）,所以屏蔽按钮D之后就可以正常地发送信号，而且小车也正常地接收信号。测试中发现某个信号一直被往小车发送，干扰我们通过手柄操作发送信号的 -> 发现某个按钮是坏了的（按钮D）,所以屏蔽按钮D之后就可以正常地发送信号，而且小车也正常地接收信号。测试中发现某个信号一直被往小车发送，干扰我们通过手柄操作发送信号的 -> 发现某个按钮是坏了的（按钮D）,所以屏蔽按钮D之后就可以正常地发送信号，而且小车也正常地接收信号。测试中发现某个信号一直被往小车发送，干扰我们通过手柄操作发送信号的 -> 发现某个按钮是坏了的（按钮D）,所以屏蔽按钮D之后就可以正常地发送信号，而且小车也正常地接收信号。

* 问题二：怎样用非抢占的方式处理isTxDone返回FALSE的情况？

解决方案：将写指令的部分设置成一个任务，通过一个外部变量byte来控制从指令的第几位开始写起。如果想要写当前位的时候tx函数没有执行完，便会post写的任务一段时间后再写；如果成功写入就将外部变量byte++，并继续试图写下一个byte的数据。当8个byte都写完之后，将变量byte重新置为1并释放串口的资源。

task void writeOrder(){

switch(byte){

case 1:

if(!(call HplMsp430Usart.isTxEmpty())){

post writeOrder();

break;

}

data\_end = data % 0x100;

call HplMsp430Usart.tx(0x01);

byte++;

case 2:

…

case 3:

…

case 4:

…

case 5:

…

case 6:

…

case 7:

…

case 8:

…

call Resource.release();

busy\_write = FALSE;

byte = 1;

}

}

* 问题二：用上述方法处理后怎样保证一个指令的8个byte都写完之后再写入下一个指令？

解决方案：设置一个busy\_write变量对post命令做出限制。只有当busy\_write为FALSE的时候才可一个post一个写一条指令的命令，而一个指令开始写之后便会将busy\_write设为TRUE，直到这条指令被完全写完再将busy\_write置为FALSE。这样一来可以避免一条指令在写入的时候另一条指令post写指令的任务的情况。因为这种情况发生的时候如果前面的指令没有写完，想要post的时候会因为队列中已经有一个相同任务而失败。

* 问题四：怎样实现一键复位？

一开始我们试图设计机制使得控制舵机1的指令写完后立刻开始写控制舵机2的指令，可是都不太成功。后来我们转变思路，设置一个bool变量来选择发出复位命令的时候复位的命令，TRUE时发送复位舵机1的命令，FALSE时发送复位舵机2的命令。因为手柄模块扫描的频率是10HZ，所以一般按一下会发出多条交替的“复位舵机1”、“复位舵机2”的命令，这样从外部看起来效果与一键复位相同。

## 实验收获

这次大作业实现的功能虽然简单，但是让我们对嵌入式系统的开发有了初步的了解。在编写代码的过程中，我们发现nesC的事件机制和 QT很相像，split phase的设计思想、task execution的模型又涉及到很多操作系统调度的思想。感觉随着学习的深入和实践经验的增加，我们可以触类旁通，更快地上手一个新的工具，并且加深对原有知识的认识。因为我之前只是听课，没有做过关于嵌入式系统的实验或者作业的，所以我刚开始不太了解实际上怎么实现嵌入式系统。通过这次大作业知道了实际实现嵌入式系统的方法，获得了关于sensing和广播的知识。