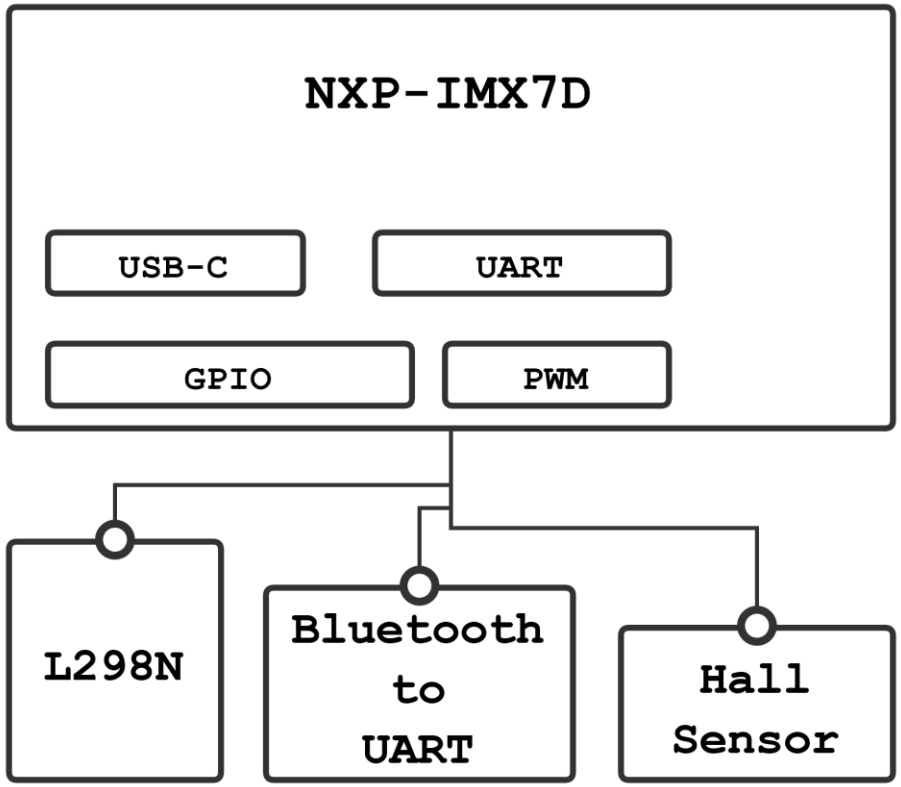
Android things 项目介绍

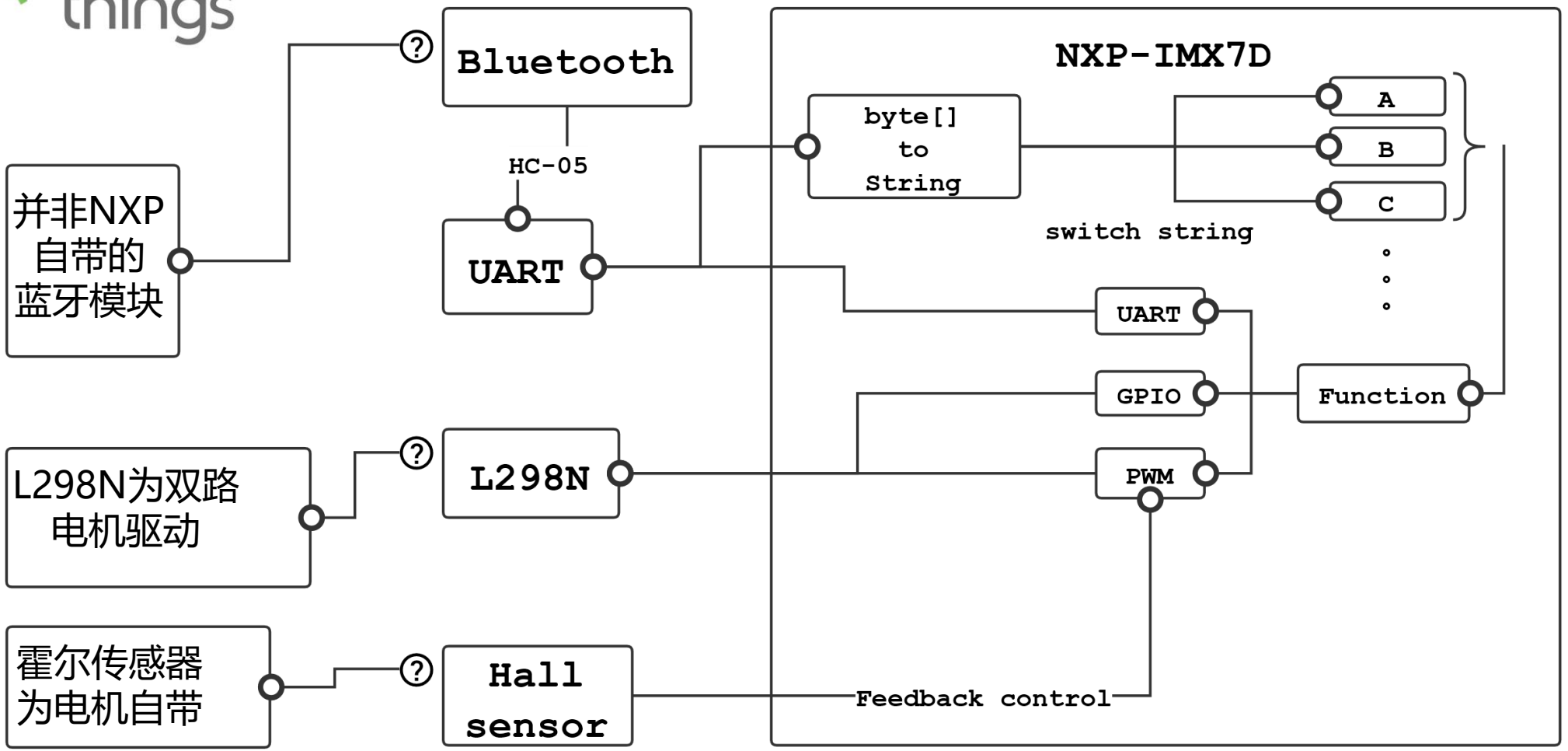
——蓝牙遥控小车

android things

系统结构



原理和过程控制


Bluetooth
HC-05
UART
并非NXP
自带的
蓝牙模块
L298N
L298N为双路
电机驱动
Hall
sensor
霍尔传感器
为电机自带
NXP-IMX7D
byte[]
to
String
A
B
C
switch string
UART
GPIO
PWM
Function
Feedback control

代码分析

- 串口通信
- GPIO控制
- PWM反馈

串口通信代码分析

- 串口参数设置
- 串口开关
- 串口收发操作

```
// 串口参数设置:波特率,数据位,截止位
private static final int BAUD_RATE = 9600;//HC-05的波特率为9600
private static final int DATA_BITS = 8;//数据位一般为八
private static final int STOP_BITS = 1;//截止位持续一位的时间

private static final int CHUNK_SIZE = 1;
```

串口通信代码分析

- 串口参数设置
- 串口开关
- 串口收发操作

```
private void openUart(String name, int baudRate) throws
IOException {
mLoopbackDevice = mService.openUartDevice(name);
mLoopbackDevice.setBaudrate(baudRate);
mLoopbackDevice.setDataSize(DATA_BITS);
mLoopbackDevice.setParity(UartDevice.PARITY_NONE);
mLoopbackDevice.setStopBits(STOP_BITS);
mLoopbackDevice.registerUartDeviceCallback(mCallback,
mInputHandler);
}//打开串口
private void closeUart() throws IOException {
if (mLoopbackDevice != null) {
mLoopbackDevice.unregisterUartDeviceCallback(mCallback);
try {
mLoopbackDevice.close();
} finally {
mLoopbackDevice = null;
}
}
}//关闭串口
```

串口通信代码分析

- 串口参数设置
- 串口开关
- 串口收发操作

```
private void transferUartData() {
if (mLoopbackDevice != null) {
// 串口数据需要时刻占满数据位，否则有几率出现漏检
try {
byte[] input = new byte[CHUNK_SIZE];
int read;
while ((read = mLoopbackDevice.read(input, input.length)) > 0)
{
。
。
。
mLoopbackDevice.write(output,read);
}
} catch (IOException e) {
Log.w(TAG, "Unable to transfer data over UART", e);
}
}
}
```

GPIO控制代码分析

- GPIO引脚指定
- GPIO初始状态设定
- GPIO状态修改

```
private Gpio MotoA;
private Gpio MotoB;
private Gpio MotoC;
private Gpio MotoD;
String pinNameA = "GPIO2_IO07";
String pinNameB = "GPIO6_IO15";
String pinNameC = "GPIO2_IO00";
String pinNameD = "GPIO2_IO05";
```

GPIO控制代码分析

- GPIO引脚指定
- GPIO初始状态设定
- GPIO状态修改

```
protected void onCreate(Bundle savedInstanceState) {
.
.
.
try {
MotoA = service.openGpio(pinNameA);
MotoA.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
MotoB = service.openGpio(pinNameB);
MotoB.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
MotoC = service.openGpio(pinNameC);
MotoC.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
MotoD = service.openGpio(pinNameD);
MotoD.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
Log.i(TAG, "Start robbery");
// Post a Runnable that continuously switch the state of the
GPIO, blinking the
// corresponding LED
} catch (IOException e) {
Log.e(TAG, "Error on PeripheralIO API", e);
}
}
```

GPIO控制代码分析

- GPIO引脚指定
- GPIO初始状态设定
- GPIO状态修改

```
protected void onCreate(Bundle savedInstanceState) {
.
.
.
try {
MotoA = service.openGpio(pinNameA);
MotoA.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
MotoB = service.openGpio(pinNameB);
MotoB.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
MotoC = service.openGpio(pinNameC);
MotoC.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
MotoD = service.openGpio(pinNameD);
MotoD.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
Log.i(TAG, "Start robbery");
// Post a Runnable that continuously switch the state of the
GPIO, blinking the
// corresponding LED
} catch (IOException e) {
Log.e(TAG, "Error on PeripheralIO API", e);
}
}
```

PWM输出方式

- PWM参数设置
- PWM工作方式
- 注意事项

```
private static final double PULSE_PERIOD_MS = 20;
// 此时的频率为50HZ (1000/20)
/*一般的开发板中都有固定的PWM引脚
这是因为PWM需要更多的系统资源以减少时滞
但是应对一般情况完全可以利用普通的数字引脚达到近似的效果
*/
private static final double PULSE_CHANGE_PER_STEP_MS = 0.5;
//这里的阶跃时间用来调速。
private static final int INTERVAL_BETWEEN_STEPS_MS = 1000;
```

PWM输出方式

- PWM参数设置
- PWM工作方式
- 注意事项

```
String pinName = BoardDefaults.getPWMPort();
//获取自带的PWM引脚编号
mActivePulseDuration = INTIME_ACTIVE_PULSE_DURATION_MS;
//载入当前设置的占空比时间系数
mPwm = service.openPwm(pinName);
//打开PWM功能（内部开启中断）
mPwm.setPwmFrequencyHz(1000 / PULSE_PERIOD_MS);
mPwm.setPwmDutyCycle(mActivePulseDuration);
mPwm.setEnabled(true);
```

PWM输出方式

- PWM参数设置
- PWM工作方式
- 注意事项

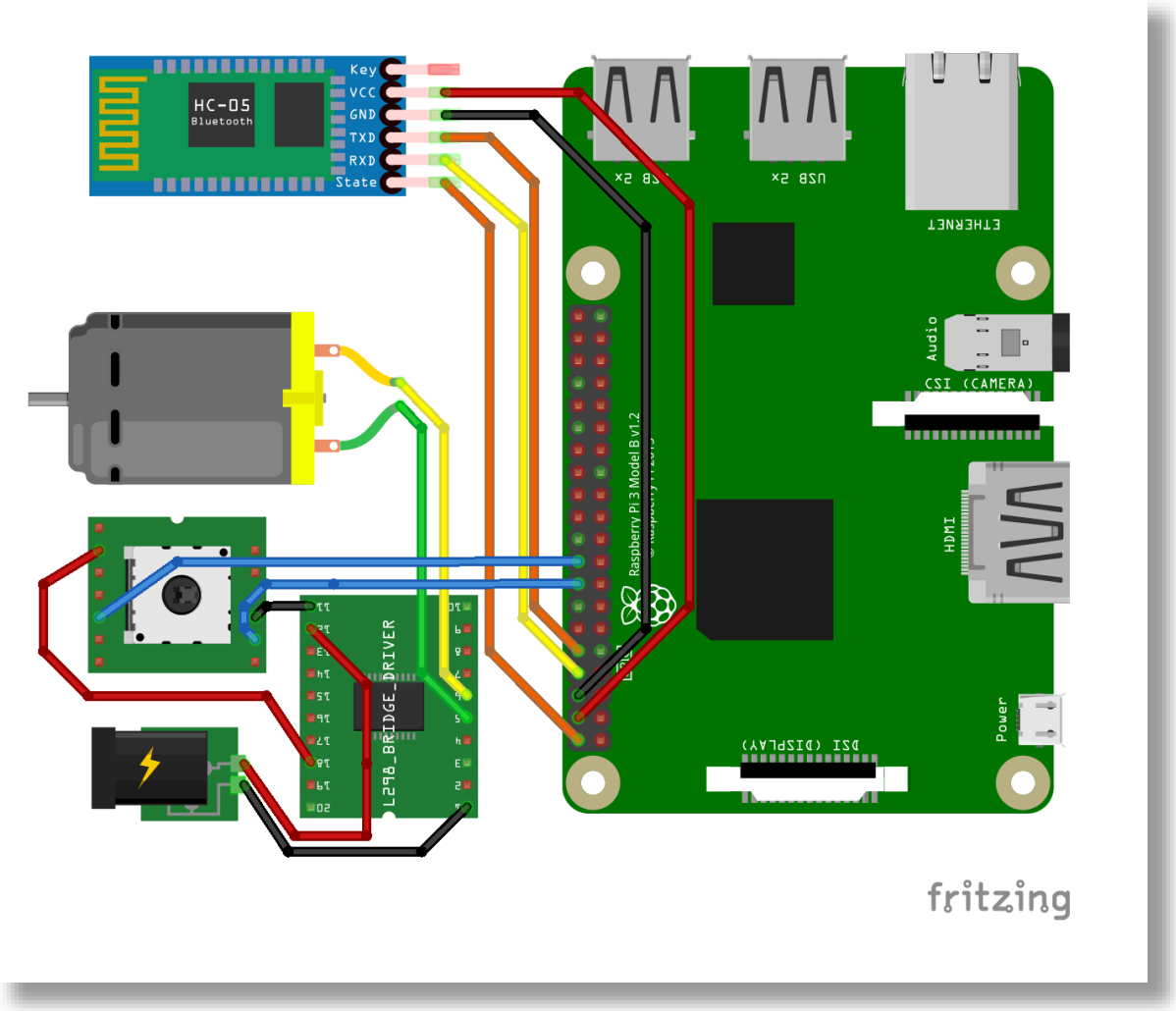
- PWM的一般工作场景为特定角度的舵机驱动或者全向舵机的转速控制。
- 一般直流电机的调速也可以采用简单的模拟方式，但是NXP和Android things 不提供模拟接口。
- 开发过程中如果需要开启PWM，切记及时回收引脚并且避免多路PWM调制（例如生成SBUS信号），这样容易造成时滞和误差。
- 总而言之，应该尽可能降低Android things和硬件底层之间的耦合性。

体会和心得

- 硬件性能评价
- Android things软件特性
- 后期工作和展望

硬件开发体验

- Android things对硬件支持良好，基本无时滞，校验周期短，体验统一。
- 驱动外设尽量采用外接供电方式，RPi3B和NXP的开发板没有逆向保护，请务必做好强弱电隔离。
- 引脚需要及时回收，这一点对很多硬件开发者而言比较陌生，虽然后果并不十分严重，但是还是建议养成习惯。
- 小规模的原型使用面包板杜邦线即可，如果需要产品化则需要在PCB模拟设计中注意保持时钟的统一，当然基于Core board的开发不需要考虑这一点。



软件开发特性

Android things

- 不需要专注优化问题，系统已经足够健壮。
- 适用于物联网应用场景，配合云服务，可以获得足够强大的性能。
- 开发学习成本极低，和传统android开发一脉相承。

Linux Embedded

- 需要定制和优化系统才能发挥足够的性能。
- 适用场景广，变化和版本多。
- 开发学习成本较高，上手时间较长。

后期工作和展望

- 希望能够进一步挖掘Android things的硬件特性，将其应用在无人机方面。
- 探索Android things另一天然的优势，基于Firebase的云服务和AI相关功能。
- 设计一些简单的拓展板并且开源硬件设计，为Android things社区贡献力量。