Jetson Nano B01

▼ 1 Jetson Nano 资料

1. Jetson Nano主板资料： https://www.yahboom.com/study/jetson-nano 密码： 72q3
2. 安装顺序：先装网卡，后装其他外壳。
   WiFi无线网卡安装：
   https://www.yahboom.com/study_module/jn-wireless_card
   英伟达NX&Nano铝合金机箱外壳安装视频：
   https://www.yahboom.com/study_module/Jetson-Metal
   摄像头支架安装：
   https://www.yahboom.com/study_module/jn-camera
   7寸屏支架安装视频：
   https://www.yahboom.com/study_module/LCD-7
3. 接上电源没有启动，无指示灯灯亮解决方案：需要拔掉J48的跳线帽重新查一下（出厂时是错位插着的，所以需要重插），可以访问以下链接查看跳线帽图示
   https://www.yahboom.com/build.html?id=2495&cid=301

▼ 2 GPIO

可以使用Python或者C++来控制Jetson Nano的40针引脚。Jetson Nano B01的引脚的工作电平是 3.3V，所以使用的时候尽量不要接 5V 电平。

- C++：https://github.com/pjueon/JetsonGPIO

  https://github.com/pjueon/JetsonGPIO/blob/master/docs/how_to_link_to_your_project.md
- Python：https://www.yahboom.com/build.html?id=6193&cid=586

▼ 2.1 引脚定义图

BCM编码	功能名	物理引脚		功能名	BCM编码
	3V3	1	2	5V	
2	SDA	3	4	5V	
3	SCL	5	6	GND	
4	D4	7	8	D14(TXD)	14
	GND	9	10	D15(RXD)	15
17	D17	11	12	D18	18
27	D27	13	14	GND	
22	D22	15	16	D23	23
	3V3	17	18	D24	24
10	D10	19	20	GND	
9	D9	21	22	D25	25
11	D11	23	24	D8	8
	GND	25	26	D7	7
0	D0(ID_SD)	27	28	D1(ID_SC)	1
5	D5	29	30	GND	
6	D6	31	32	D12	12
13	D13	33	34	GND	
19	D19	35	36	D16	16
26	D26	37	38	D20	20
	GND	39	40	D21	21

▼ 2.2 C++库安装、使用教程

▼ 2.2.1 Installation

https://github.com/pjueon/JetsonGPIO/blob/master/docs/installation_guide.md

▼ 2.2.2 Apply with cmake

Add this to your CMakeLists.txt

```
find_package(JetsonGPIO)
```

assuming you added a target called `mytarget`, then you can link it with:

```
target_link_libraries(mytarget JetsonGPIO::JetsonGPIO)
```

▼ 2.2.3 API

https://github.com/pjueon/JetsonGPIO/blob/master/docs/library_api.md

▼ 2.3 Python库安装、使用教程

- installation

  https://github.com/NVIDIA/jetson-gpio

- apply

  https://www.yahboom.com/build.html?id=6193&cid=586

▼ 2.4 GPIO的使用

首先应该明确每个引脚在库中的编码和功能，方便我们在程序中精确操作到指定的引脚上，以下是一些基本操作流程：

▼ step1：程序初始化（initialization）

- 导入库

  C++中头文件：

  ```
  #include<JetsonGPIO.h>
  ```

  CMakeLists.txt文件中应该加入：

  ```
  find_package(JetsonGPIO)

  ...

  target_link_libraries(mytarget JetsonGPIO::JetsonGPIO)
  ```

- 设置编码模式：

  通常我们使用BCM编码，这种编码在树莓派中同样适用。Jetson Nano B01官方板上的编号不遵循BCM编码，显示的是引脚的物理位置，但是诸如5V和GND的引脚标注都是一致的。BCM模式下，每个可操作的引脚的编号是int类型，与RPi.GPIO不同的编码模式则是标准库定义的string类型。

  ```
  using namespace GPIO
  GPIO::setmode(GPIO::BCM)
  ```

- 检查模式

```
GPIO::NumberingModes mode = GPIO::getmode();
```

这个函数返回枚举类的实例。模式必须是`GPIO::BOARD`、`GPIO::BCM`、`GPIO::CVM`或`GPIO::TEGRA_SOC`之一`GPIO::NumberingModes::None`。

- 去掉报警

如果使用输入模式以外的引脚模式，Jetson Nano可能会向程序报警，因为有可能这些引脚正在被程序外的进程所使用，一般来说可以不必理会。

```
GPIO::setwarnings(false);
```

- 通常在step3处，回到这个程序块编写回调函数等程序。

▼ step2：功能初始化（setup）

第二步就需要设置对应的引脚，开启所需的功能（串口、定时、中断等），并且完成对应部分的初始化。

▼ 1 引脚模式初始化

- 设置引脚模式
  - 输入输出：

```
// (where channel is based on the pin numbering mode (
// input
GPIO::setup(channel, GPIO::IN); // channel must be in

// output
GPIO::setup(channel, GPIO::OUT);

// initialize at the same time
GPIO::setup(channel, GPIO::OUT, GPIO::HIGH); // or GP

// setup multiple channels as input
GPIO::setup({chan1, chan2}, GPIO::IN);

// setup multiple output channels. The initial value(
std::vector<int> channels = {chan3, chan4, chan5};  /
GPIO::setup(channels, GPIO::OUT, GPIO::HIGH);
```

```
// setup multiple output channels with multiple initi
GPIO::setup({chan6, chan7}, GPIO::OUT, {GPIO::HIGH, GI
```

- 读取输入

```
int value = GPIO::input(channel);
```

- 设置输出

```
GPIO::output(channel, state);
```

其中状态可以是`GPIO::LOW`(== 0) 或`GPIO::HIGH`(== 1)。

```
std::vector<int> channels = { 18, 12, 13 };   //
or std::vector<std::string>
GPIO::output(channels, GPIO::HIGH); // or GPIO::
LOW
// set the first channel to LOW and rest to HIGH
GPIO::output(channels, {GPIO::LOW, GPIO::HIGH, G
PIO::HIGH});
```

▼ 2 中断或定时初始化及其回调函数

首先，需要一个口作为输入口，设置完模式其实就已经完成它的初始化了。之后可以选择然后确认它的中断触发方式，选择之后就以及开始再此等待中断了。触发方式有三种：上升沿、下降沿以及两种皆可：`GPIO::RISING`、`GPIO::FALLING`或`GPIO::BOTH`

最后可以添加一个回调函数（此功能可用于运行回调函数的第二个线程。因此，回调函数可以与主程序并发运行以响应边缘）：

```
// 消除抖动
GPIO::add_event_detect(channel, GPIO::RISING, callback_fn, 
```

▼ 3 PWM脉宽调制

None temporarily for C++.

▼ step3：功能完备化（loop）

▼ 中断或定时

- step1：开启中断并等待

```
GPIO::wait_for_edge(channel, GPIO::RISING);
```

第二个参数指定要检测的边缘，可以是`GPIO::RISING`、`GPIO::FALLING`或`GPIO::BOTH`。还可以控制等待时间的长度来消除按钮抖动或者：

```
// 以毫秒为单位
// 第三个参数是用于消除抖动的时间，设置为10ms
// 等待0.5秒
GPIO::WaitResult result = GPIO::wait_for_edge(channel, G
```

该函数返回一个`GPIO::WaitResult`对象，其中包含检测到边缘的通道名称。

- step2：检查事件发生或超时

  要检查是否检测到事件或发生超时，可以使用`.is_event_detected()`返回对象的方法，只是将其强制转换为`bool`类型。

  返回的对象可以隐式转换为`bool`，并且其值等于result的返回值`.is_event_detected()`：

```
// returns the channel name for which the edge was detec
// ("None" if a timeout occurred)
std::string eventDetectedChannel = result.channel();

if(result.is_event_detected()){ /*...*/ }
// or
if(result){ /*...*/ } // is equal to if(result.is_event_
```

- （extra）：定期检查事件是否发生过了，然后执行

  此函数可用于定期检查自上次调用以来是否发生了事件。该函数可以按如下方式设置和调用：

```
// set rising edge detection on the channel
GPIO::add_event_detect(channel, GPIO::RISING);
run_other_code();
if(GPIO::event_detected(channel))
    do_something();
```

和以前一样，您可以检测`GPIO::RISING`、`GPIO::FALLING`或 的事件`GPIO::BOTH`。

- 删掉边缘检测

```
GPIO::remove_event_detect(channel);
```

▼ **（extra）：回调函数编写**

回调函数的类型可以如下：

- 可使用`const std::string&`类型参数（对于通道名称）**或**不带任何参数进行调用。返回类型必须是`void`
- 可复制构造
- 相等 - 与相同类型可比较 (ex> `func0 == func1`)

```
// define callback function
void callback_fn(const std::string& channel)
{
    std::cout << "Callback called from channel " << cl
}

// add rising edge detection
GPIO::add_event_detect(channel, GPIO::RISING, callbac
// 防止多个事件折叠成单个时间多次调用回调函数，增加去抖
// bouncetime set in milliseconds
GPIO::add_event_detect(channel, GPIO::RISING, callbac
```

- 设置多个回调函数

```
// you can also use callbacks witout any argument
void callback_one()
{
    std::cout << "First Callback" << std::endl;
}

void callback_two()
{
    std::cout << "Second Callback" << std::endl;
}

GPIO::add_event_detect(channel, GPIO::RISING);
```

```
GPIO::add_event_callback(channel, callback_one);
GPIO::add_event_callback(channel, callback_two);
```

- 去掉、删除回调函数

```
// 去掉回调函数
GPIO::remove_event_callback(channel, callback_two);
// 删除边缘检测
GPIO::remove_event_detect(channel);
```

▼ PWM脉宽调制

None temporarily for C++.

▼ step4：程序退出（exit）

- 去掉引脚

```
GPIO::cleanup();

GPIO::cleanup(chan1); // cleanup only chan1
GPIO::cleanup({chan1, chan2}); // cleanup only chan1 and cha
```

- 去掉中断

```
// 去掉回调函数
GPIO::remove_event_callback(channel, callback_two);
// 去掉事件检测
GPIO::remove_event_detect(channel);
```

▼ （extra）：额外信息调试

要获取有关 Jetson 模块的信息，请使用/阅读：

```
std::string info = GPIO::JETSON_INFO;
// or
std::string info = GPIO::JETSON_INFO();
```

要获取 Jetson 设备的型号名称，请使用/阅读：

```
std::string model = GPIO::model;
// or
```

```
std::string model = GPIO::model();
```

要获取有关库版本的信息，请使用/阅读：

```
std::string version = GPIO::VERSION;
```

这提供了具有 XYZ 版本格式的字符串。

回调函数案例：

这是一个用户定义类型回调示例：

```
// define callback object
class MyCallback
{
public:
    MyCallback(const std::string& name) : name(name) {}
    MyCallback(const MyCallback&) = default; // Copy-constru
ctible

    void operator()(const std::string& channel) // Callable
with one string type argument
    {
        std::cout << "A callback named " << name;
        std::cout << " called from channel " << channel << s
td::endl;
    }

    bool operator==(const MyCallback& other) const // Equali
ty-comparable
    {
        return name == other.name;
    }

    bool operator!=(const MyCallback& other) const
    {
        return !(*this == other);
    }

private:
```

```
    std::string name;
};

// create callback object
MyCallback my_callback("foo");
// add rising edge detection
GPIO::add_event_detect(channel, GPIO::RISING, my_callback);
```

此功能允许您检查所提供的 GPIO 通道的功能：

```
GPIO::Directions direction = GPIO::gpio_function(channel);
```

该函数返回`GPIO::IN`、`GPIO::OUT`，都是枚举类型`GPIO::Directions`的实例。

▼ 3 串口

开启串口权限，注意这个权限关机后就也被关闭，下次需要重新开启

```
sudo chmod 777 /dev/ttyTHS1
```

使用linux的串口助手测试

运行以下命令

```
sudo apt install cutecomsudo cutecom
```

就可以看见cutecom打开了。一般不需要设置，直接点击**open**就能使用，然后通过InPUT输入文本，按回车键就能发送内容了。

▼ 4 Example：Joystick with Maxon Motor

使用Joystick需要Jerson提供5个接口：Vcc，GND以及三个输入口，那么程序应该如下书写：

```
// initialize
#include <JetsonGPIO.h>
#include "Definitions.h"
#include <iostream>
#include <vector>
```

```
using namespace GPIO;
using namespace std;

int key = 1;
DWORD errorCode = 1;
BOOL status = 0;
long limitation = 800000;
vector<long> targetPosition = {0, 0};

void delay(int sec);
bool detectIfExceedLimitation(int value, int limitation, long targe
void KeyInterrupt();

void delay(int sec)
{
    time_t start_time, cur_time;// 声明变量
    time(&start_time);

    do {
        time(&cur_time);
    } while ((cur_time - start_time) < sec);
}

bool detectIfExceedLimitation(int value, int limitation, long Targe
    if (abs(TargetPosition) > limitation || (value>=480 && value <=
        cout << "Surpass the maximum position of the freedom" << en
        return false;}
    if (value > 520) targetPosition[i]+=30000;
    else targetPosition[i]-=30000;
    return true;
}

void KeyInterrupt() {
    cout << "================================" << endl;
    cout << "Exiting The Control Mode..." << endl;
    key = 0;
}
```

```
int main(){
    setmode(BCM)
    setwarnings(false);
    NumberingModes mode = getmode();
    cout << "Currently Coding mode is " << mode << endl; // should

    // setup
    vector<bool> allowMotorToMove = {0, 0};
    vector<HANDLE> keyHandle = {0, 0};
    vector<int> motorChannel = {8, 7};
  vector<WORD> nodeId = {1, 0};
    setup({25, 8, 7}, GPIO::IN); // channel must be int or std::str
    add_event_detect(25, GPIO::RISING, callback_fn, 200);


    // loop
    // 获取模拟引脚输入值并映射到一个范围上
    int value;
    while (1) {
        for (int i = 0 ; i<=1 ; i++) {
            value = GPIO::input(motorChannel[i]);
            allowMotorToMove[i] = detectIfExceedLimitation(int valu
        }

        while (allowMotorToMove[0]) {
            status = VCS_SetEnableState(keyHandle[0], nodeId[0],&er
            status = VCS_MoveToPosition(keyHandle[0], nodeId[0], ta
            delay(1);
            status = VCS_SetDisableState(keyHandle[0], nodeId[0],&e
        }
        while (allowMotorToMove[1]) {
            status = VCS_SetEnableState(keyHandle[0], nodeId[0],&er
            status = VCS_MoveToPosition(keyHandle[0], nodeId[0], ta
            delay(1);
            status = VCS_SetDisableState(keyHandle[0], nodeId[0],&e
        }

        // exit
        if (key) {
```

```
            cout << "Reset all the motors..........." << endl;
            /* reinitialize the motor state */
            cout << "Remove all the event detected..." << endl;
            // 去掉事件检测
            remove_event_detect(25);
            cout << "Clear up the functions of pins..." << endl;
            cleanup();
            cout << "================================" << endl;
            break;
        }
    }
}
```