K210 入门基础

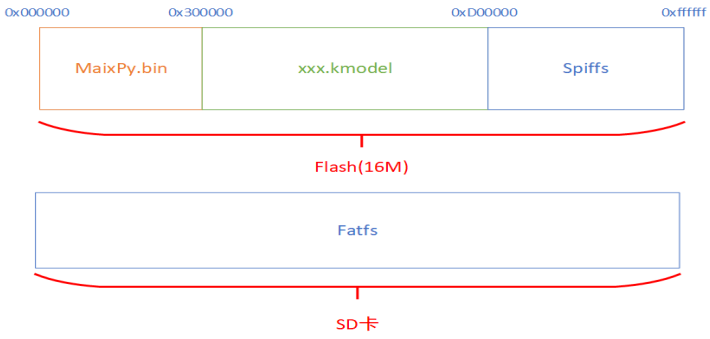
Liam

2023 年 7 月 22 日

摘要

本文将介绍 K210 的基础部分。

1 存储系统



固件区

用来存储 MaixPy.py 固件，从 0x000000 开始

1.1 模型区

但模型区不存放模型

1.2 文件系统区

该区域交给文件系统管理，文件系统格式为 *spiffs*. 不同的文件系统，统一由虚拟文件系统进行管理，提供统一的接口。3MB 的挂载在 *SPIFF*,SD 卡挂载到*/sd* 目录。使用示例：

```
        import uos
        print("files:", uos.listdir("/flash"))
        with open("/flash/test.txt", "w") as f:
        f.write("hello text")
        print("files:", uos.listdir("/flash"))
        with open("/flash/test.txt", "r") as f:
        content = f.read()
        print("read:", content)
```

SD 卡应该满足：

1. 支持 SPI 模式
2. 分区为 MBR （msdos）
3. 格式化为 FAT32

1.3 开机自启动脚本

系统会在 /flash 或者 /sd(优先) 目录创建 boot.py 文件和 main.py，开机会自动先执行 boot.py，然后执行 main.py（如果检测到 SD 卡则执行 SD 卡里的），编辑这两个脚本的内容即可实现开机自启，如果在 boot.py 里面写死循环（While True）程序，将会导致 main.py 不能运行（先调用 boot.py 后调用 main.py），重新发送不带死循环的 boot.py 即可解决。

- boot.py 主要用于配置硬件，只配置一次即可。
- main.py 可以用于主要的运行的程序。

2 功能

2.1 CPU/RAM

- 复位

```
        import machine
        machine.reset()
```

- 主频

```
        from Maix import freq
        freq.set(cpu = 400, kpu = 400)
```

2.2 内存管理

- GC 垃圾回收
- 系统堆内存

GC 内存的总大小是可以设置的，所以，根据具体的使用情况可以适当修改 GC 内存大小，比如：

- 为了加载更大的模型，可以把 GC 内存设置小一点
- 如果分配新的变量提示内存不足，可以适当将 GC 内存设置大一点即可
- 如果都不够了，就要考虑缩减固件大小，或者优化代码了

```
        from Maix import utils
        import machine

        print(utils.gc_heap_size())

        utils.gc_heap_size(1024*1024) # 1MiB
        machine.reset()
        # reboot after run it
```

3 片上外设

3.1 GPIO

K210 上有高速 GPIO 和通用 GPIO, 每个 IO 可以分配到 FPIOA 上 48 个管脚之一.

- GPIOHS 共 32 个

  可配置输入输出信号

  每个 IO 具有独立中断源

  中断支持**边沿触发电平触发**

  可配置上下拉和 **高阻态**

- GPIO 8 个

  都使用同一个中断源

  中断支持**边沿触发电平触发**

3.1.1 构造函数

```
class GPIO(ID,MODE,PULL,VALUE)
```

3.1.2 方法

参数为空则返回当前状态

```
# value
GPIO.value([value])
```

GPIO 中断模式:

1. GPIO.IRQ_RISING
2. GPIO.TRQ_FALLING
3. GPIO.TRQ_BOTH

fd

```
# irq
GPIO.irq(CALLBACK_FUNC,TRIGGER_CONDITION,
GPIO.WAKEUP_NOT_SUPPORT,PRORITY)
```

关闭中断

```
GPIO.disirq()
```

```
        GPIO.mode(MODE)
        # GPIO.IN
        # GPIO.PULL_UP
        # GPIO.PULL_DOWN
        # GPIO.OUT
```

3.2 I2C

隶属构造函数

```
class machine.I2C(id,mode=I2C.MODE_MASTER, scl = None,
 sda = None, freq=400000, timeout=1000, addr=0,
 addr_size=7,on_recieve=None, on_transmit=None,
 on_event=None)
```

- id: I2C ID, [0 2] (I2C.I2C0 I2C.I2C2) [3 5] (I2C.I2C3 I2C.I2C5, I2C_SOFT) 是软模拟 I2C 的编号。

3.2.1 方法

- `I2C.init(...)` 初始化参数
- `i2c.scan()` 扫描从机
- `i2c.readfrom(addr, len, stop=True)` 从总线读取数据
- `i2c.readfrom_into(addr, buf, stop=True)` 读取数据放在变量中
- `i2c.writeto(addr, buf, stop=True)` 发送数据到从机
- `i2c.readfrom_mem(addr, memaddr, nbytes, mem_size=8)` 读取从机寄存器
- `i2c.readfrom_mem_into(addr, memaddr, buf, mem_size=8)` 读取从机寄存器值到指定变量中
- `i2c.writeto_mem(addr, memaddr, buf, mem_size=8)` 写数据到从机寄存器

- `i2c.deinit()` / `del i2c` 注销 I2C 硬件，释放占用的资源，关闭 I2C 时钟

3.3 machine.SPI

在 K210 上，SPI 有一下特征：

- 共有 4 个 SPI 设备，其中 SPI0 、SPI1、SPI3 只能工作在主机模式下，SPI2 只能工作在从机模式时下，在 MaixPy 上，SPI3 已经用来连接了 SPI Flash 作为保留硬件资源。
- 支持 1/2/4/8 线全双工模式，在 MaixPy 中，目前只支持标准（摩托罗拉）4 线全双工模式（即 SCK，MOSI，MISO，CS 四个引脚）
- 最高传输速率 45M：1/2 主频，约 200Mbps
- 支持 DMA
- 4 个可配置任意引脚的硬件片选

构造函数：

```
1         spi = machine.SPI(id, mode=SPI.MODE_MASTER, baudrate=500000,
2         polarity=0, phase=0, bits=8, firstbit=SPI.MSB, sck,
3         mosi, miso, cs0, cs1, cs2, cs3)
```

引脚可以使用 `fm` 统一管理引脚，从 `sck` 一下都可以不设置。

- `spi.init(id, mode=SPI.MODE_MASTER, baudrate=500000, polarity=0, phase=0, bits=`
- `spi.readinto(buf, write=0x00, cs=SPI.CS0)`
- `spi.write(buf, cs=SPI.CS0)`
- `spi.write(write_buf, read_buf, cs=SPI.CS0)`
- `spi.deinit()`

3.4 machine.PWM

隶属构造函数：`pwm = machine.PWM(tim, freq, duty, pin, enable=True)`，共有 3 个定时器，最大 12 路 `PWM`

`duty`：取值为 [0,100]

3.4.1 方法

- `pwm.init(tim, freq, duty, pin, enable=True)`
- `pwm.freq(freq)` 获取或者设置频率
- `pwm.duty(duty)`
- `pwm.enable() // pwm.disable()`
- `pwm.deinit()`

3.5 machine.Timer

构造函数：

```
tim = machine.Timer(id, channel, mode=Timer.MODE_ONE_SHOT, period=1000, unit=Ti
```

- `id`：`Timer.TIMER0~TIMER2`
- `channel`：`Timer.CHANNEL0~3`
- `div`：硬件定时器分频器，取值`[0,255]` 定时器时钟频率 $= \frac{clk\_timer}{2^{(div+1)}}$

3.5.1 方法

- `tim.init(id, channel, mode=Timer.MODE_ONE_SHOT, period=1000, unit=Timer.UNIT_M`
- `tim.callback(callback)`
- `tim.period(period)`
- `callback_arg` 获取设置的传给回调函数的参数。
- `tim.start()`
- `tim.start()`
- `tim.restart()`
- `tim.deinit()`

3.6 machine.UART

k210 一共有 3 个 uart，每个 uart 可以进行自由的引脚映射。
构造函数：`uart = machine.UART(uart,baudrate,bits,parity,stop,timeout, read_buf_le`

- `uart.init()`
- `uart.read()`
- `uart.readline()`
- `uart.write()`
- `uart.deinit()`

3.7 machine.WDT

MaixPy 的 WDT 看门狗模块，用于在应用程序崩溃且最终进入不可恢复状态时重启系统。一旦开始，当硬件运行期间没有定期进行喂狗（feed）就会在超时后自动复位。

```
from machine import WDT
wdt0 = WDT(id=1, timeout=4000, callback=on_wdt, context={})
```

3.7.1 方法

- `wdt0.feed()`
- `wdt0.stop()` 停止看门狗对象

4 Maix 库

4.1 FPIOA 现场可编程 IO 阵列,Field Programmable Input and Output Array

K210 支持每个外设随意映射到任意引脚，使用 FPIOA 功能来实现。
类初始：`FPIOA()`

4.1.1 方法

- `fpioa.help()`
- `fpioa.set_function(pin,func)`
- `get_Pin_num(func)`

5 helper

5.1 fpioa_manager

fpioa_manager：简称 fm，该模块用于注册芯片内部功能和引脚，帮助用户管理内部功能和引脚映射关系的功能模块. 实例：

```
from fpioa_manager import fm

fm.register(11, fm.fpioa.GPIO0, force=True)
fm.register(12, fm.fpioa.GPIOHS0, force=True)
fm.register(13, fm.fpioa.UART2_TX)
fm.register(14, fm.fpioa.UART2_RX)

# other code

fm.unregister(11)
fm.unregister(12)
fm.unregister(13)
fm.unregister(14)
```

如果设置 force=False ， 则会在 register 发现硬件功能已经被使用了，此时就会弹出异常.

有一些引脚已经被注册!!! 请查看https://wiki.sipeed.com/soft/maixpy/zh/api_reference/builtin_py/fm.html

- `get_pin_by_function(pin)`
- `get_gpio_used()`

get_gpio_used() 返回一个迭代器

5.2 Board

这是一个 MaixPy 板级配置模块，它可以在用户层统一 Python 代码，从而屏蔽许多硬件的引脚差异。

5.3 Micropython Editor

MaixPy 固件中集成了文件编辑器——pye, 用户可以直接通过串口终端修改板子里面的文件

```
from pye_mp import pye

pye("/sd/boot.py")
```

6 Media 资源

6.1 lcd

- `lcd.init(type=1, freq=15000000, color=lcd.BLACK, invert = 0, lcd_type = 0)`
- `lcd.deinit()`
- `lcd.width()`
- `lcd.height()`
- `lcd.type()`
- `lcd.freq(freq)`
- `lcd.get_backlight()`
- `lcd.display(image, roi=Auto, oft=(x, y))`
- `lcd.clear()`
- `lcd.rotation(dir)`
- `lcd.bgr_to_rgb(enable)`

6.2 sensor