



TH244A001_UserGuide

TH244A001_UserGuide

Version: 0.40

TH244A001 UserGuide

1. 修订记录	7
2. 开发板概述	8
3. 开发板规格	10
3.1 供电框图.....	18
3.2 电源切换逻辑.....	19
3.3 外部电源 DC-JACK (Location DC1) :	19
3.4 外部电源 VIN(Location CN1):	20
3.5 USB1/USB2 供电(Location USB1, USB2).....	20
3.6 5V/3.3 输出(Location CN1 CN5):	21
3.7 J1 开发板工作电压选择	22
3.8 J2 ADC 外部参考电压切换	22
3.9 J3 虚拟串口使能.....	23
4. Digital I/O	24
4.1 基本介绍.....	24
4.2 驱动负载能力参考.....	26
4.3 Digital I/O list	27
关于 Digital I/O 特别说明	29
5. 时间函数 TIMER	30
6. 特殊功能复用.....	32

7.	PWM	33
8.	DAC.....	35
9.	ADC.....	41
9.1	ADC 基本说明	41
9.2	ANALOG IN list.....	43
10.	串行 I/O	46
10.1	UART 端口说明	47
10.2	IIC	50
10.3	SPI	53
11.	USB 界面.....	56
11.1	USB1.....	56
11.2	USB2.....	57
12.	RTC.....	60
13.	SLEEP/STOP Mode.....	61
14.	IWDT	66
14.1	独立看门狗定时器.....	66
14.2	IWDT 专用中断函数	67
14.3	IWDT 库主要用法.....	67
15.	定时器中断 TimerOne	69

TH244A001 UserGuide

15.1 TimerOne 介绍	69
15.2 TimerOne 库控制输出 PWM	70
15.3 TimerOne 库定时中断	70
16. 定时器中断 Mstimer2	71
17. EEPROM	73
18. 多段数码管库显示	77
19. 其他重要说明	78
19.1 AREF (CN3)	78
19.2 RESET PIN(CN1-RESET)	78
19.3 RESET KEY: SW2	79
19.4 USER KEY SW3	79
19.5 L13: LED1	80
19.6 TX LED/RX LED	80
19.7 主控器电源开关 SW1 / Power ON LED2	81
19.8 LED6/7/8 表示 MLink 端 USB1 与 PC 连接状态	81
19.9 IOREF	82
19.10 开发包路径管理	83
19.11 虚拟串口设置和程序下载调试	84
20. 开发板重启	86

21. 常用函数以及基本用法	87
21.1 GPIO 读写控制.....	87
21.2 延时函数	87
21.3 Timer 的运用	89
21.4 PWM	89
21.5 DAC	92
21.6 ADC 函数.....	93
21.7 UART	96
21.8 IIC	100
21.9 SPI 主模式.....	107
21.10 USB	109
21.11 USB-mouse.....	109
21.12 USB-KeyBoard.....	115
21.13 RTC.....	121
21.14 SLEEP/STOP mode	124
21.15 IWDT	126
21.16 TimerOne.....	131
21.17 MsTimer2.....	136
21.18 EEPROM.....	139

1. 修订记录

1. 20230712 初版发布 V0.20
2. 20231217 开发包 2.0.0 发布, 修改支持主频 36MHz, SPI 时钟 18MHz 等描述
3. 20240229 修改部分表达, 增加更多图片资料
4. 20240514 更新为 V0.30

由于开发包 2.2.0 发布, UserGuide 新增 SLEEP/IWDT 的运用, 详见

SLEEP/STOP Mode 和 IWDT
5. 20240618 发布开发包 2.3.0 版本。说明书更新为 V0.40, 修改若干文字表述和图片;

增加支持定时器中断 TimerOne 和 Mstimer2 的库文件;

增加支持 EEPROM 的库文件(内置 FLASH 模拟最大 8K EEPROM 操作);

增加支持数码管的库文件 (TM1637Display/ShiftRegister74HC595);

增加 PWM 可设置频率范围从 300Hz~5000Hz, 增加支持到低频率 1Hz;

增加串口禁用功能;

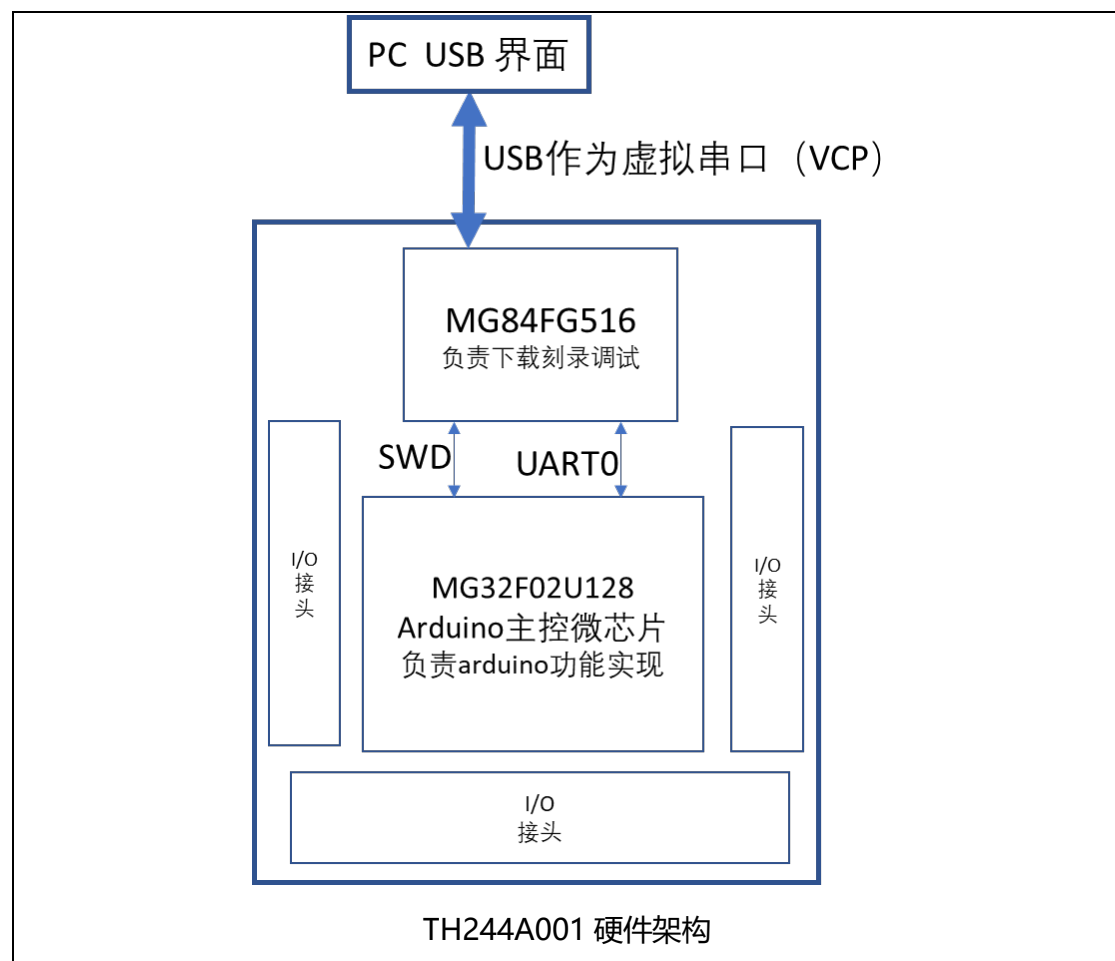
修改 SLEEP/IWDT 部分说明文字;
6. 20240830 更新 TH244A 为 TH244A001

2. 开发板概述

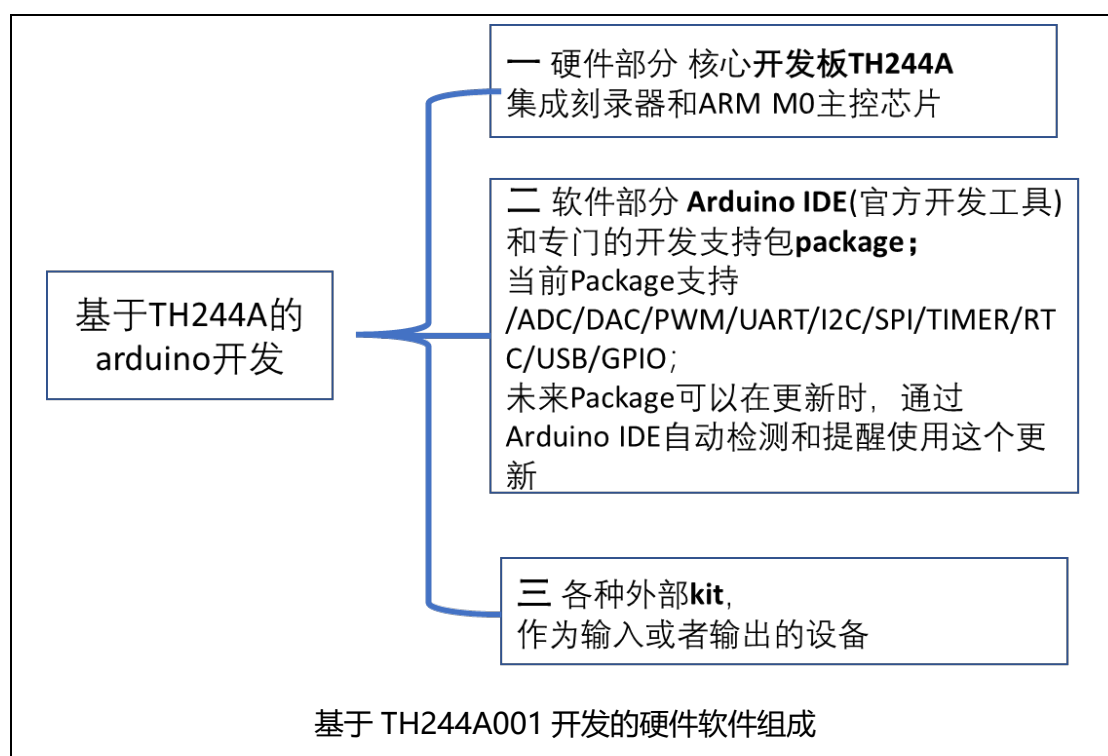
开发板 TH244A001，兼容 UNO R3 接口，并提供更多的可用 GPIO 和通信接口。结合了 Arduino UNO R3 和 Arduino MEGA2560 的硬件接口。主控芯片从 UNO/2560 的 8 位 MCU，升级到 32 位 ARM Cortex-M0，且主控芯片带有 USB1.1 接口,可以仿真为键盘或者鼠标使用。

开发板 TH244A001 是基于 Megawin M0 芯片 MG32F02U128AD64 设计。

TH244A001 自带下载 MLink，无需要购买其他下载器。



除了支持通用的单片机开发环境，比如 Keil，为了支持开源的 Arduino IDE 开发环境，专门设计了配套的程序开发包。只有安装了 TH244A001 对应的支持包 package，才能够在 Arduino IDE 中运用 Arduino 的常用方法来开发程序以及实现编译和下载程序。



开发包下载已经更新方式请参考 **TH244A001_UserManual-第三方开发板的开发包安装和更新**。

3. 开发板规格

板名称	工作电压	时钟速度	数字 IO Digital IO	模拟输入 ADC	模拟输出 DAC
Megawin TH244A001	5V /3.3V 可切换	时钟源为内嵌 IHRCO 12MHz; MCU 主频 36MHz	47	16 8bit 10bit 12bit (默认)	1 12bit

PWM	UART	SPI	IIC	USB DEVICE	程序设计界面
7 8bit 默认 1KHz, 1Hz~5KHz 可调	7 路 收发缓存 64 bytes 默认 9600 bps 可通过宏定义关闭指 定端口节约 SRAM	1 路(Master) 支持最高时钟 18MHz	2 路(Master/Slave) 支援设置时钟为 100KHz;400KHz;1MHz	1 USB2 作为 USB-HID 设 备: Mouse, Keyboard	USB1 通过 MLink MG84FG516 上传 程序

- MG32F02U128AD64 (ARM 32 位 Cortex™-M0 CPU);
- 时钟源为内嵌 IHRCO (内置高频 RC 振荡器) 12MHz;
- IWDI 时钟源:内嵌 ILRCO (内置高频 RC 振荡器) 32KHz;
- MCU 最高主频 36MHz;
- 开发板引出数字 I/O 引脚 47 个 (0~46) ;
 - 其 46 经过上拉电阻连接开发板上的 USER KEY, 方便按键方面的设计测试之用;
 - 7 路可设置 PWM 输出;

- ◆ 占空比 duty 为 8bit 分辨率。可设置范围 0~255 对应 0~100%;
- ◆ 预设频率均为 1000Hz;
- ◆ 提供可快速修改 PWM 频率的函数:

`analogWriteFrequency(PWM_FRQ_XX, Fre),`

频率 Fre 设置范围 1Hz~5KHz

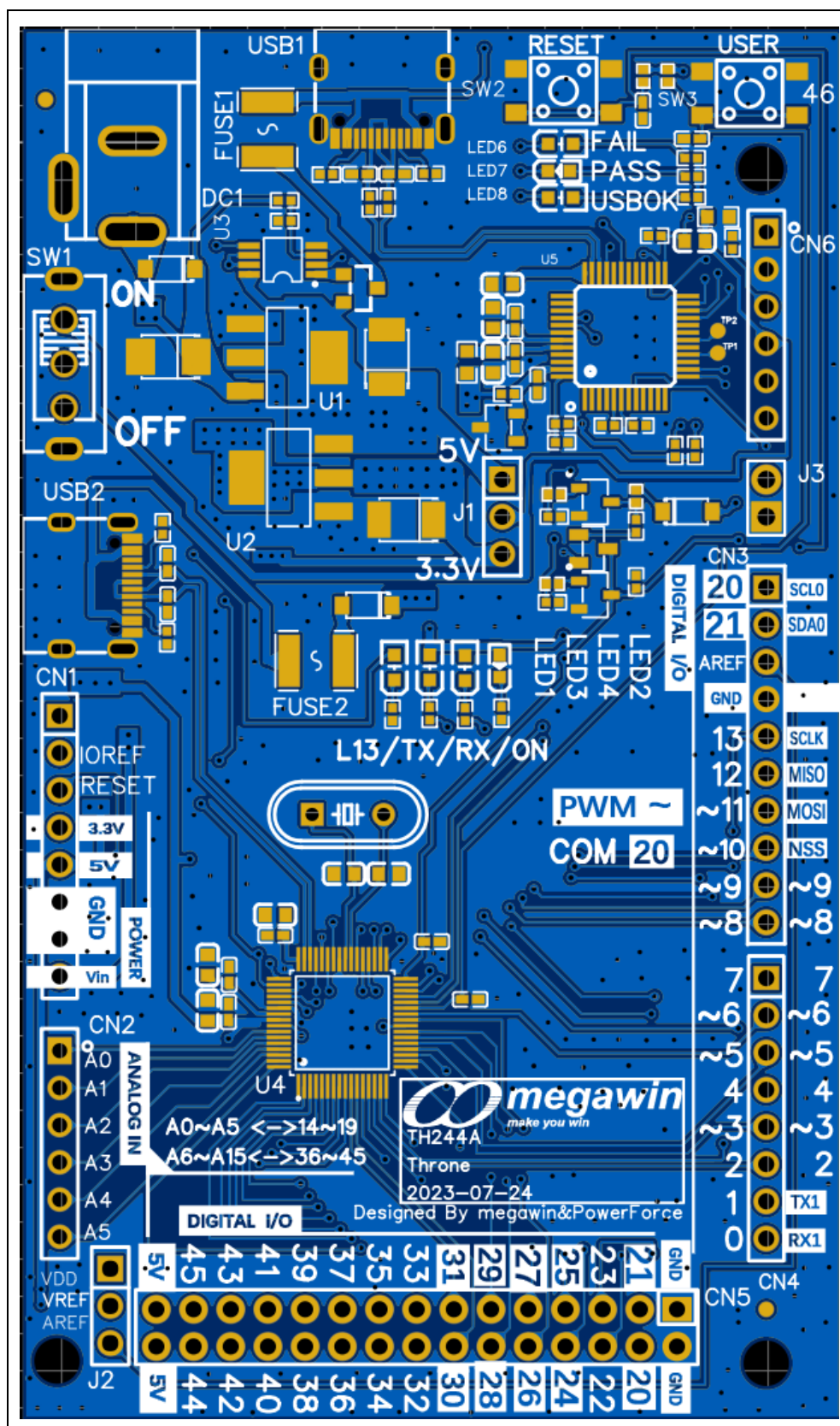
- ◆ 7 个 PWM 输出可以独立为 3 个组分别来设置 (详见 [PWM](#) 部分) ;
 - ◆ 7 个 PWM 输出占空比 duty 可以独立设定;
 - 2 路可以通过使用定时器库 [TimerOne](#) 设置 PWM 输出, 支持 42/43 脚位输出 PWM, 详见 [TimerOne](#);
 - 16 路模拟输入通道 ADC。使用 `analogRead()`快速获取模拟电压的 ADC 数值;
 - ADC 位数默认为 12bit, ADC 转换的数值范围 0~4095;
 - ADC 位数可以通过 `analogReadResolution(ref)`设定 `ref=8` (0-255) ,10 (0-1023) ,12 (0-4095) ;
- 注意: 这里获取的是 ADC 数值, 实际物理量, 需要公式转化, 重新标定.
- 1 路 12bit DAC 通道, 从开发板 21 输出, 通过 `analogWrite(21, Value)`输出范围 0~VDD, 呈线性; Value 范围 0~4095;
 - 7 路 UART 速度最大 115200bps
 - UART0 专门设计为下载程序或者串口监视器使用;

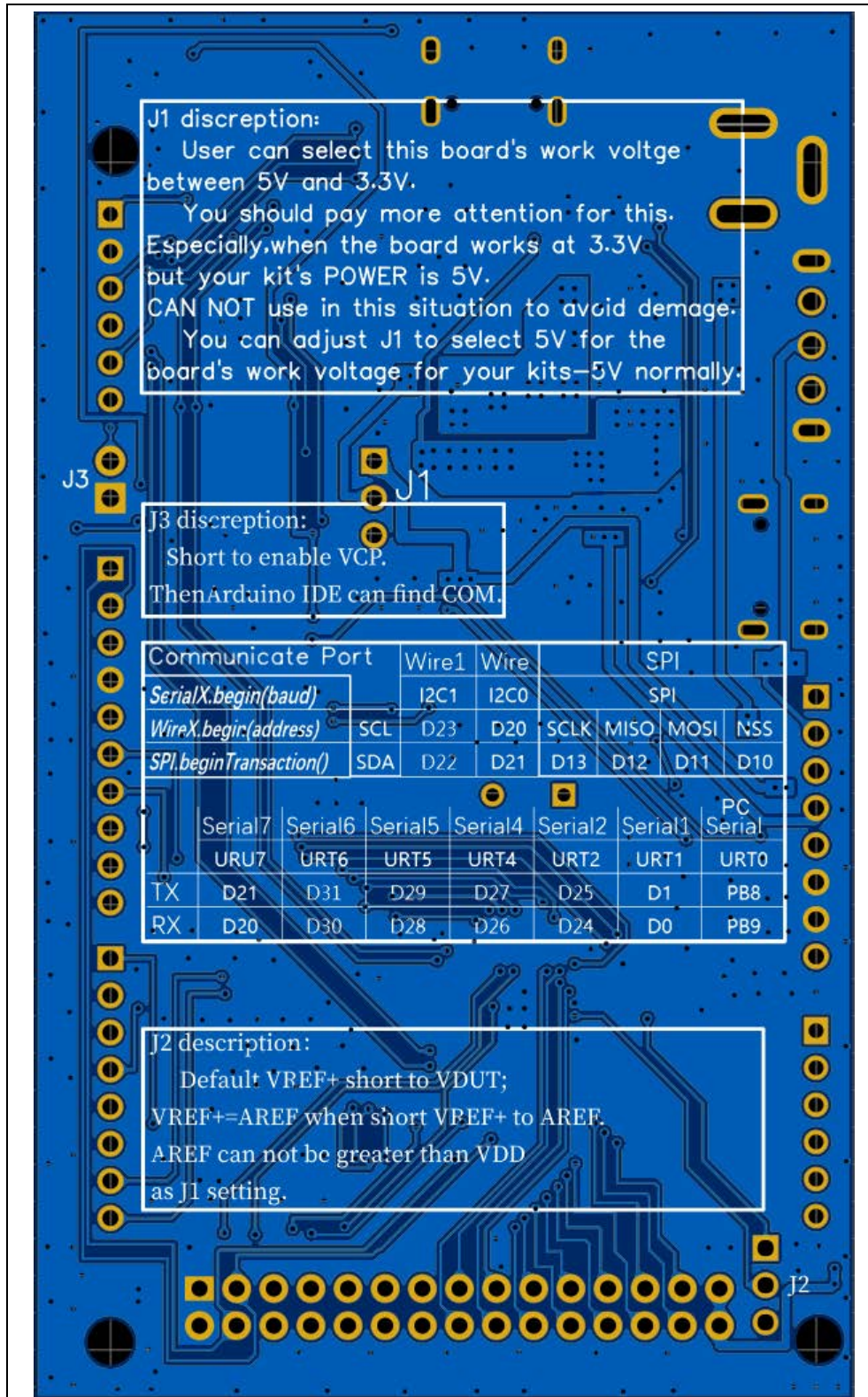
- UART1/2/4/5/6/7 调试或者和其他设备通信使用（UART7 与 IIC0 复用）；
- 开发包默认初始化所有串口 Serial/Serial1/2/4/5/6/7，默认速度为 9600bps
- 为了节约初始化串口占用 SRAM，开发包支持通过宏定义禁用指定串口，详见 [UART](#)
- 2 路 IIC（IIC0 与 UART7 复用）；SCL 速度默认 100KHz，可设置为 400KHz,1MHz
- 1 路 SPI，速度支持 18Mbps
- 任何 I/O 端口引脚或 RST 对地电压 -0.5V~VDD+0.5V；
- 128K 字节内置 Flash 用于存储代码和资料；其中最大 8K bytes 可以通过 EEPROM 库的使用，模拟为 EEPROM。为了节约 SRAM，开发包默认初始化 512 bytes EEPROM 容量，可以通过宏定义修改 EEPROM 大小，详见 [EEPROM](#)；
- 内建 16K 字节 SRAM
- 具有 USB2.0 全速，支持 HID,可以作为 USB-Mouse 或者 USB-Keybord 使用，可使用 Joystick 作为 USB-Mouse 功能；
- 开发板工作电压可通过 2.54mm 2pin 跳帽（Jumper）J1 选择为 5V 或者 3.3V；
- 供电方式可以通过 DC JACK，VIN 或两个 USB1/2 供电；
 - DC JACK 供电时，Vin 接口可以作为输出来使用；
- 3.3V 输出电流限制

- DC JACK/VIN 输入时, 50mA MAX;
- USB 输入时, 300mA MAX;
- 5V 输出电流限制
 - DC JACK/VIN 输入时, 50mA MAX;
 - USB 输入时, 300mA MAX;

TH244A001 UserGuide

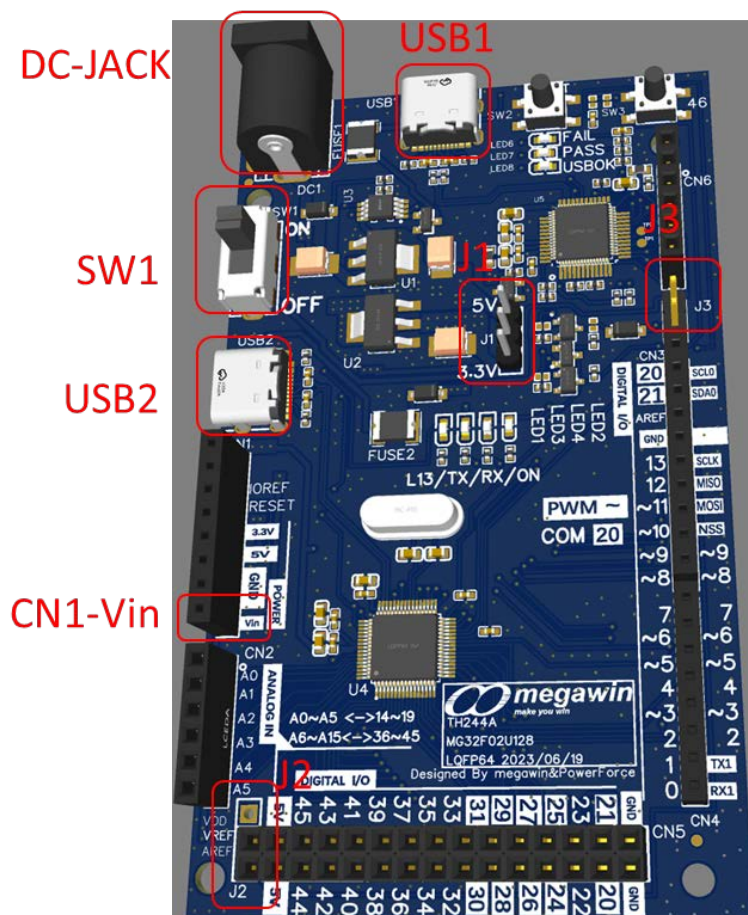
开发板示意图如下：



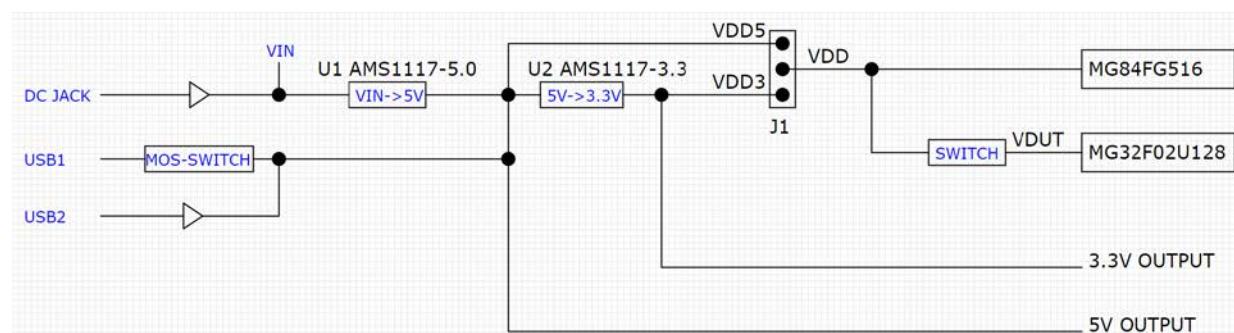


主控芯片 MG32F02	VDD 最大电流 200mA
主控芯片 MG32F02 I/O	I/O 的电流限制为 38.5mA(5V), 13mA(3.3V)
5V/3.3V 输出	限制为 50mA; 当需要更大电流时，需给外部模块单独供电，模块和开发板的地要可靠连接；
USB1/USB2	500mA

主要接口分布图

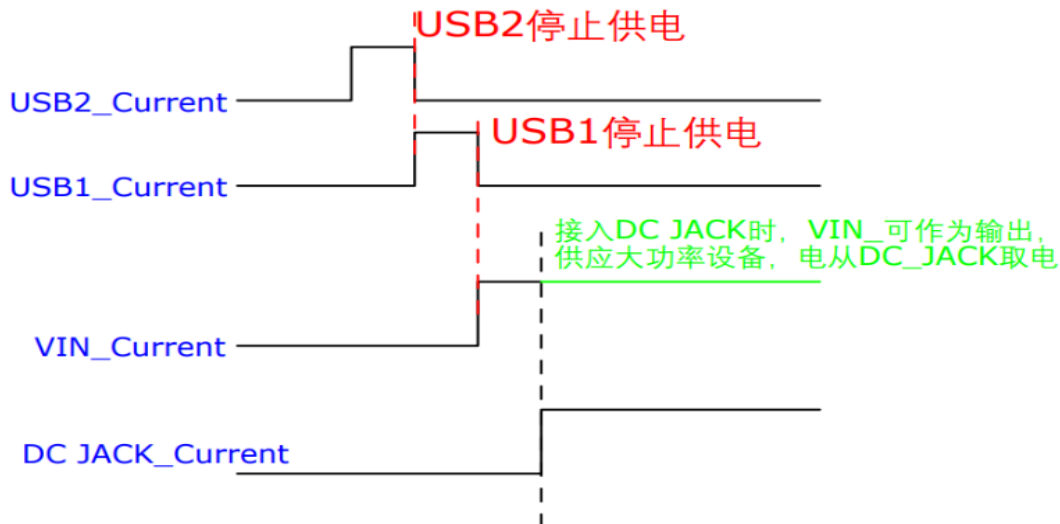


3.1 供电框图



3.2 电源切换逻辑

DC JACK/VIN > USB1 (Debug port) > USB2(Native port)



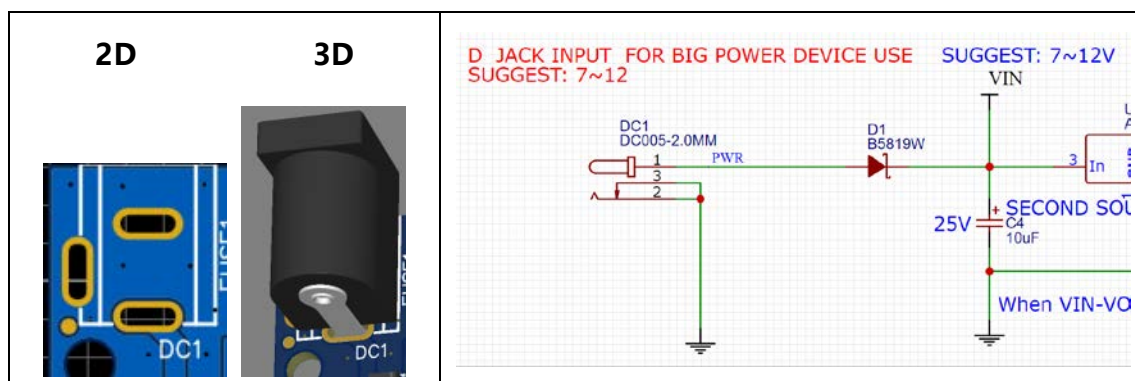
从上到下，接入不同电源,电路供电选择

3.3 外部电源 DC-JACK (Location DC1) :

可以用 AC-DC 适配器 (建议 DC 输出 7V~9V) 。

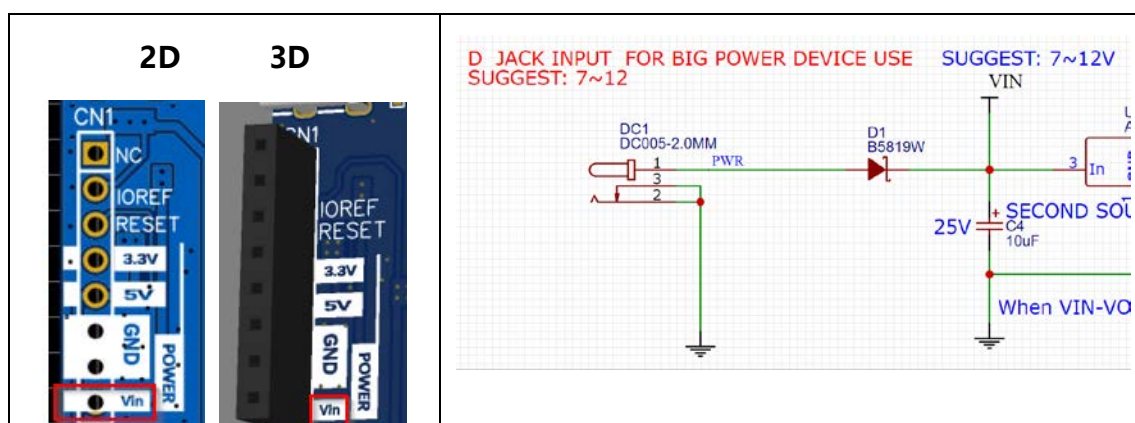
适配器可以通过 2.0mm 的插头连接到电源插座上 (DC-Jack 上) , 以此连接到电压转换芯片。电压转换芯片能支持 7 到 12V 电压输入。如果输入电压低于 7V, 5V 引脚可能得到的电压会低于 5V, 电压转换芯片运行可能会不稳定。如果使用输入电压超过 12V, 会造成电压转换芯片过热, 甚至过热损坏。推荐输入电压范围为 7V~9V。

当使用外部电源 DC-JACK 供电时, 此时 VIN 可以作为输出来使用, 可以从 VIN 连接到外部负载设备 (前提要明确外部设备工作电压是在 DC JACK 输出范围内) , 比如通过 VIN 给电机供电, 为电机可以提供足够的功率。



3.4 外部电源 VIN(Location CN1):

- Arduino 可使用 VIN 输入外部电源;
- 当通过 DC JACK 座供电时，还可通过 VIN 引脚给外部设备供电，特别注意电压是否符合模块需求



3.5 USB1/USB2 供电(Location USB1, USB2)

USB1, USB2 均为 Type C 形式接口，可以提供 5V/500mA 电源给开发板和配件板使用。USB1 是主要的开发阶段的供电方式。

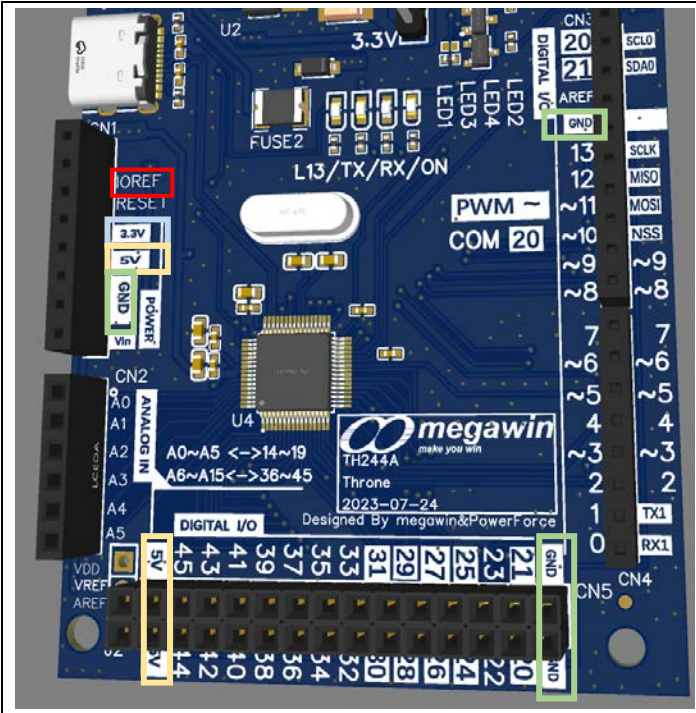
3.6 5V/3.3 输出(Location CN1 CN5):

5V: 通过板载稳压芯片输出或者 bypass USB1 或者 USB2 提供的 5V 的电压引脚。TH244A001 提供 3 个脚提供 5V 输出，分别位于 CN1 和 CN5。可以从 DC 电源口、USB1、USB2、VIN 三处给开发板供电。建议只把 5V 和 3.3V 作为输出，不允许输入。当 DC JACK 供电时，+5V 输出最大电流 50mA；当 USB 供电时，+5V 输出最大电流 300mA。

通过板载稳压芯片输出的 3.3V 的电压引脚电流限制，当 DC JACK 供电时，+3.3V 最大电流 50mA；当 USB 供电时，+3.3V 最大电流 300mA。该电压为 MG32F02U128 和 MG84FG516 的工作电压。详见**供电框图**。

5V 或者 3.3V 电压是 MG32F02U128 和 MG84FG516 的工作电压。

开发板工作电压由开关 J1 切换。根据需要，选择 5V 或者 3.3V 供电。

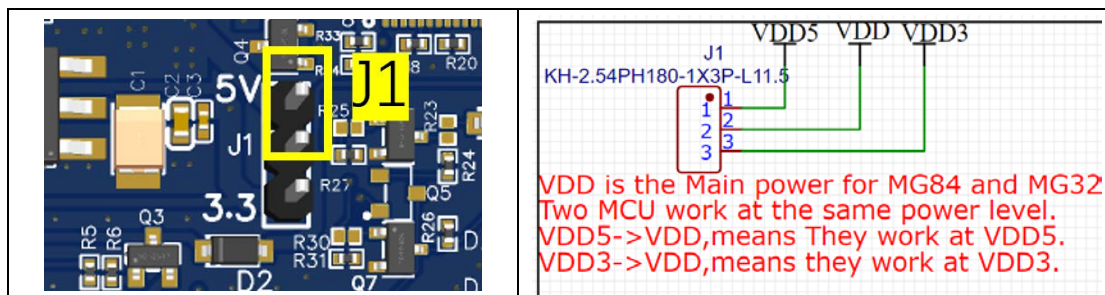


开发板提供:

- 1 个 3.3V 输出脚;
- 3 个 5V 输出脚;
- 5 个接地脚 (GND) ;
- 1 个 IOREF 电压同 MG32 工作电压 J1 设置电压;

便于使用者接入更多模块

3.7 J1 开发板工作电压选择

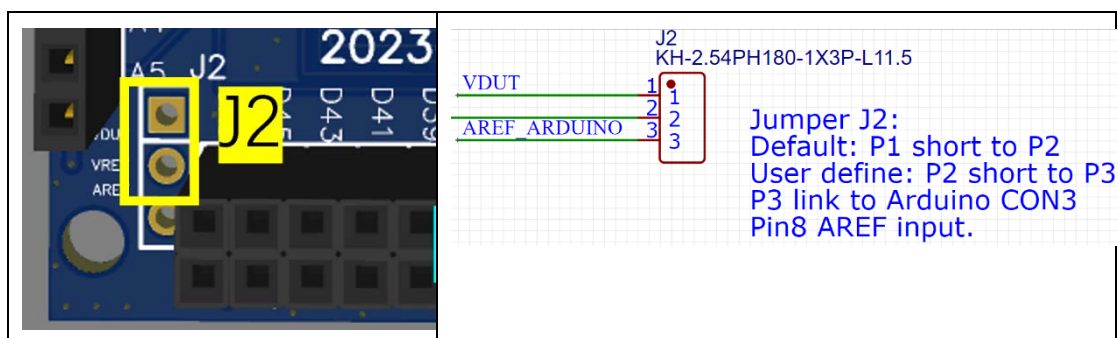


用户根据需要选择合适的工作电压。默认开发板为 5V 工作电压。

3.8 J2 ADC 外部参考电压切换

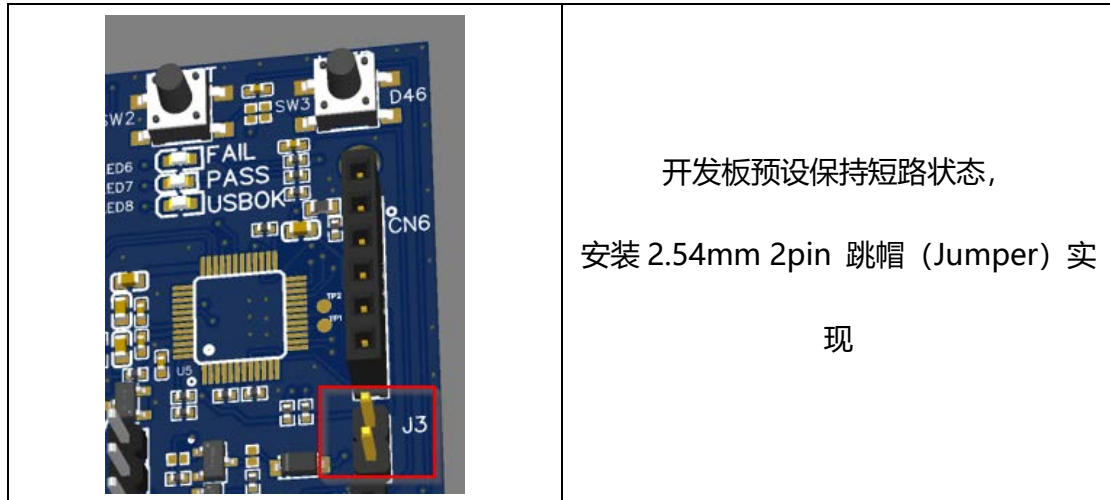
开发板 MG32F02U128 可设置 ADC 参考电压为外部电压 (VREF+) 或内部 IVR24。VREF+可以固定为 MCU 工作电源 VDUT(这是比较常用的方式), 也可以由外部设备提供参考电压 AREF。开发板使用 J2 切换两种连接, 默认为 VREF+ 直接连接 VDUT。但当外设的模拟信号较小电压, 比如 1.8V 或者 3.3V 或者其他范围, 使用外部电压作为参考, 这样可以提高采样精度。

ADC 参考电压默认设置为外部 VREF+作为参考电压, VREF+电压同 J1 的选择。



3.9 J3 虚拟串口使能

J3 选择短路，就是实现 USB1 虚拟串口(VCP)，用于下载程序和调试使用。

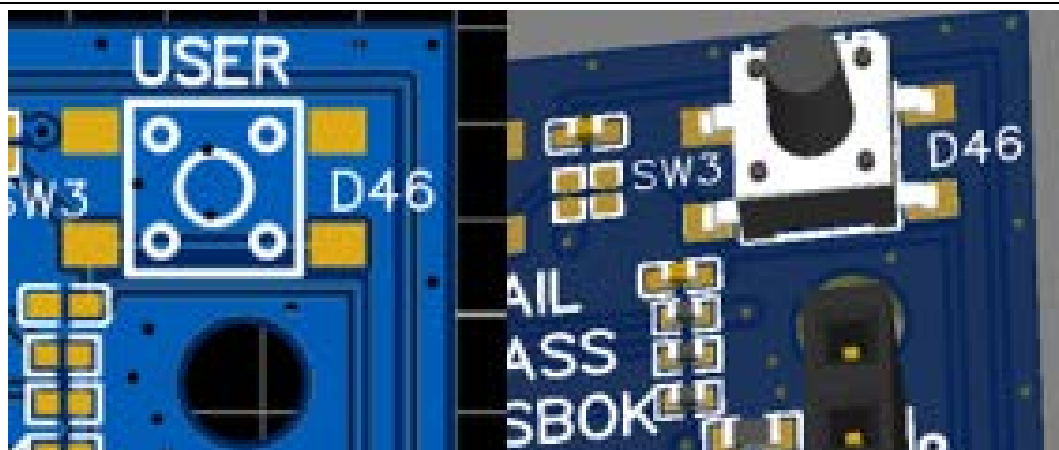


4. Digital I/O

4.1 基本介绍

Digital I/O: Arduino 编号 0~46, 总计 47 个

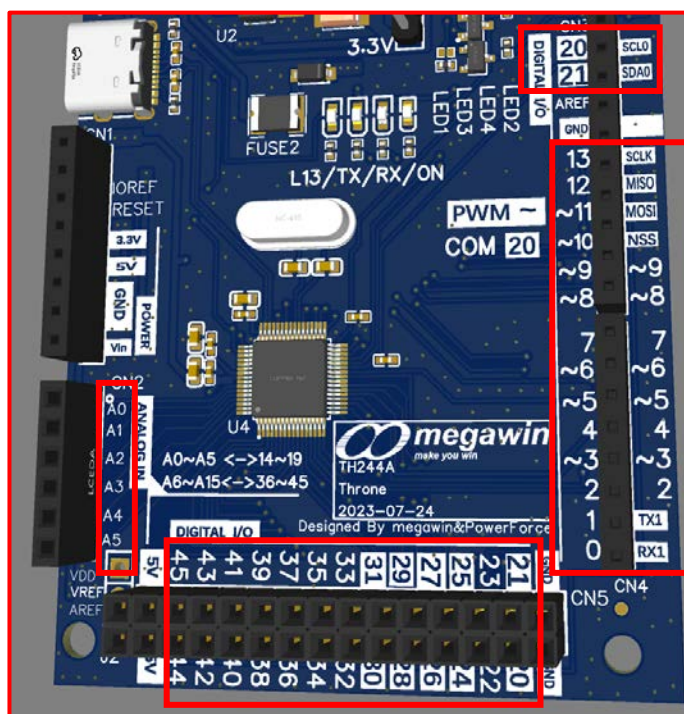
(其中 46 专门定义为 USER KEY 输入, 详见 **USER KEY SW3**)



46 默认按键 SW3 状态: 不按下为 HIGH, 按下为 LOW;

0~45 连接到接头，除 GPIO 功能外，复用 ADC/DAC/串行 I/O(UART/IIC/SPI)

等;



TH244A001 开发板按照 Arduino 的使用方式定义了 47 个可以直接程序设计使用的 GPIO。可直接使用 Arduino 官方的函数，比如 `pinMode()`，`digitalWrite()`，`digitalRead()`等函数。开发板工作在 3.3V 或者 5V（由开发板上的 J1 决定，默认是 5V）。工作电压是 5V 或者 3.3V，I/O 口的电流与输出功率的设定有关，具体可见下表。默认设定为是全功率输出，且都不启动上拉电阻。

当 GPIO 驱动负载时，需要考虑驱动电流不能超过极限，建议如下：

- 5.0V 工作电压时；以 38.5mA 为限；
- 3.3V 工作电压时，以 13mA 为限；

4.2 驱动负载能力参考

GPIO 默认都是全功率，低速模式，内部上拉电阻不使能的情况。在需要低功率或者高速模式时，需要专门设定。比如 IO 配置为通信用的 SWD/IIC/UART/SPI/USB 等模式，都默认配置为高速模式。

VDD=5V 状况，

I/O 全功率输出 HIGH 时，最大电流(推-拉 为负载提供电流)为 38.5mA

I/O 全功率输出 LOW 时，最大电流(挽-灌 从负载抽取电流)为 30.4mA

内部复位引脚上拉电阻为 250Kohm。其他 I/O 都有 13.3Kohm 内部上拉电阻；

IOH1	输出高电流（推挽输出 & 全功率级别）	VDD=5.0V, V _{PIN} = 2.4V		38.5		mA
IOH2	输出高电流(推挽输出 & 1/2 功率级别)	VDD=5.0V, V _{PIN} = 2.4V		19.8		mA
IOH3	输出高电流(推挽输出 & 1/4 级别)	VDD=5.0V, V _{PIN} = 2.4V		10.1		mA
IOH4	输出高电流(推挽输出 & 1/8 功率级别)	VDD=5.0V, V _{PIN} = 2.4V		5.2		mA
IOL1	输出低电流(全功率级别)	VDD=5.0V, V _{PIN} = 0.4V		30.4		mA
IOL2	输出低电流(1/2 功率级别)	VDD=5.0V, V _{PIN} = 0.4V		15.7		mA
IOL3	输出低电流(1/4 功率级别)	VDD=5.0V, V _{PIN} = 0.4V		8.0		mA
IOL4	输出低电流(1/8 功率级别)	VDD=5.0V, V _{PIN} = 0.4V		4.0		mA
RPU	IO 引脚上拉电阻	除 RSTN 外		13.3		Kohm
R _{RST}	内部复位引脚上拉电阻	RSTN pin		250		Kohm

VDD=3.3V 状况，

I/O 全功率输出 HIGH 时，最大电流(推-拉 为负载提供电流)为 13mA

I/O 全功率输出 LOW 时，最大电流(挽-灌 从负载抽取电流)为 11.3mA

I/O 复位引脚上拉电阻为 426Kohm。其他 I/O 都有 19.5Kohm 的内部上拉电阻。

IOH1	输出高电流（推挽输出 & 全功率级别）	VDD=3.3V, V _{PIN} = 2.4V		13		mA
IOH2	输出高电流(推挽输出 & 1/2 功率级别)	VDD=3.3V, V _{PIN} = 2.4V		6.5		mA
IOH3	输出高电流(推挽输出 & 1/4 级别)	VDD=3.3V, V _{PIN} = 2.4V		3.5		mA
IOH4	输出高电流(推挽输出 & 1/8 功率级别)	VDD=3.3V, V _{PIN} = 2.4V		1.7		mA
IOL1	输出低电流(全功率级别)	VDD=3.3V, V _{PIN} = 0.4V		22		mA
IOL2	输出低电流(1/2 功率级别)	VDD=3.3V, V _{PIN} = 0.4V		11.3		mA
IOL3	输出低电流(1/4 功率级别)	VDD=3.3V, V _{PIN} = 0.4V		5.6		mA
IOL4	输出低电流(1/8 功率级别)	VDD=3.3V, V _{PIN} = 0.4V		2.8		mA
RPU	IO 引脚上拉电阻	除 RSTN 外		19.5		Kohm
R _{RST}	内部复位引脚上拉电阻	RSTN pin		426		Kohm

4.3 Digital I/O list

No.	MG32F02U128AD6 4 LQFP64 pin No.	MG32F02U128AD64 Pin name	Arduino Pin name*
1	35	PC9/RXD1	0
2	34	PC8/TXD1	1
3	14	SD3/PB4	2
4	20	PB10	3
5	33	PC7	4
6	21	PB11	5
7	26	ICKO/PC0	6
8	15	SD2/PB5	7
9	38	PC12	8
10	42	PD0	9
11	51	NSS/PD9	10
12	44	MOSI/PD2	11
13	49	MISO/PD7	12
14	50	SPI0_CLK/PD8	13
15	58	PA0/ADC0	14/A0**
16	59	PA1/ADC1	15/A1**
17	60	PA2/ADC2	16/A2**

TH244A001 UserGuide

18	61	PA3/ADC3	17/A3**
19	62	PA4/ADC4	18/A4**
20	63	PA5/ADC5	19/A5**
21	13	MOSI/PB3	20
22	12	SCK/PB2	21 & DAC***
23	37	PC11/SDA1	22
24	36	PC10/SCL1	23
25	17	PB7	24
26	16	PB6	25
27	24	PB14	27
28	23	PB13	26
29	25	PB15	28
30	22	PB12	29
31	10	NSS/PB0	31
32	11	MISO/PB1	30
33	43	PD1	32
34	45	PD3	33
35	52	PD10	34
36	53	PD11	35
37	64	PA6/ADC6	36/A6**
38	1	ADC7/PA7	37/A7**

39	2	ADC8/PA8	38/A8**
40	3	ADC9/PA9	39/A9**
41	4	ADC10/PA10	40/A10**
42	5	ADC11/PA11	41/A11**
43	6	ADC12/PA12	42/A12**
44	7	ADC13/PA13	43/A13**
45	8	ADC14/PA14	44/A14**
46	9	ADC15/PA15	45/A15**
47	27	PC1	46 for USER KEY****

关于 Digital I/O 特别说明

*** Arduino Pin name** 此标记的数字可以在 Arduino 编程中直接引用。整数

还可以方便使用循环，如下：

- digitalWrite(40, HIGH), 写 40 引脚输出高；
- digitalRead(15) 读取 15 引脚状态；
- analogWrite(3, 200)对引脚 3 输出 duty 为 $200/255=78.4\%$ 的方波；默认频率为 1000Hz；
- analogRead(A0)读取 AO 端口的模拟数据值
- ```
for(int i = 0; i<20;i++){

 pinMode(i, OUTPUT);

} //使用循环，设定 0~19 这 20 个脚位为输出
```

**\*\*表示数字端口**，并可复用为模拟输入口；A0~A15 表示模拟输入通道

**\*\*\* 21** 为 DAC 输出口，analogWrite(21, value) value: 0~4095，不带负载时  
对应输出电压 0~VDD；

**\*\*\*\* USER KEY**，用于单独按键设计需求，硬件已经设计了上拉电阻，只能作为  
INPUT 使用；

常用函数

pinMode()

digitalRead()

digitalWrite()

详见常用函数以及基本用法之 [GPIO 读写控制](#)

## 5. 时间函数 TIMER

时间函数，在时序控制中非常重要。对于 Arduino 来说有专门的处理与时间相关的函数。Arduino 官方提供了如下四个主要的时间函数。

- 阻塞函数：delay(ms)和 delayMicroseconds(us)
  - delay(ms): 让设备暂停执行毫秒(ms)
  - delayMicroseconds(us): 让设备暂停执行微秒(us)

在 delay 时间段内，MCU 只能做计时等待，直到 delay 时间完成。如果需要延时等待阶段，还需要随时响应其他的请求，就不能够使用 delay 这类阻塞函数。一般程序设计谨慎使用。

### 例如使用 `delay()` 延时 1s:

`delay(1000);` //延时 1s, MCU 在这个阶段不能做其他任何事情;

- 非阻塞形式: 函数获取 `systick`。当使用 `millis` 或者 `micros()`, MCU 在计时时, 还可以处理另外的事情

- `millis()` 返回自 Arduino 板启动后, 至今的毫秒数 ms
- `micros()` 返回自 Arduino 板启动后, 至今的微秒数 us

### 例如利用 `millis()` 实现间隔 1s, 系统计时阶段可以执行其他任务:

`unsigned long previousMillis = 0;` // 定义存储当前系统运行时间 `systick` 的变量;

`const long interval = 1000;` // 需要等待的时间间隔;

`unsigned long currentMillis = millis();` //返回系统运行的 ms 数, 重上电或者 RESET,

这个时间均会从 0 开始;

`if (currentMillis - previousMillis >= interval)`

`{`

`previousMillis = currentMillis;`

`// 放置满足时间间隔后需要执行的任务;`

`}`

常用函数:

`delay(ms)`

`delayMicroseconds(us)`

`millis()`

`micros()`

详见常用函数以及基本用法之[延时函数](#)，[Timer 的运用](#)

# 6. 特殊功能复用

很多 Digital I/O 引脚可以被配置为特殊功能来使用，这部分在芯片手册 AFS 部分有介绍。

特定功能这里主要指 PWM、模拟输出（DAC）、模拟输入（ADC）、UART、IIC、SPI 等。

一般使用者应该在 setup()中完成这些复用功能的初始化设定。一个管脚在初始化了某个功能后，就不能在定义使用另外的功能，以免冲突。比如 21 脚设定为模拟电压输出 DAC，那么 IIC0 (20->SCL,21->SDA) 就不能被配置。比如初始化了 IIC0 (20->SCL,21->SDA)，就不能使用 UART7(21->TX,20->RX)。



## 7. PWM

开发板提供 7 个 PWM 输出，分别在脚位 3, 6, 5, 8, 9, 10, 11。默认频率均为 1000Hz。7 路 PWM 可以独立控制 duty，范围 0~100%。

有别于其他官方开发板，本开发板的开发包支持设定频率。用户可设定频率范围为 1Hz~5000Hz。频率的设定按下图分为三个组别独立设定：3/6 (TM20)、5/8/9(TM36)和 10/11(TM26)。

| 分组      | MG32F02U128AD64<br>LQFP64 pin No. | MG32F02U128AD64<br>Pin name | Arduino<br>Pin name | TMxx      | 频率      |
|---------|-----------------------------------|-----------------------------|---------------------|-----------|---------|
| 组别 TM20 | 20                                | PB10                        | 3                   | TM20_OC11 | 默认 1KHz |
|         | 26                                | PC0                         | 6                   | TM20_OC00 | 默认 1KHz |
| 组别 TM36 | 21                                | PB11                        | 5                   | TM36_OC12 | 默认 1KHz |
|         | 38                                | PC12                        | 8                   | TM36_OC3  | 默认 1KHz |
|         | 42                                | PD0                         | 9                   | TM36_OC2  | 默认 1KHz |
| 组别 TM26 | 44                                | PD2                         | 11                  | TM26_OC00 | 默认 1KHz |
|         | 51                                | PD9                         | 10                  | TM26_OC11 | 默认 1KHz |

参照 Arduino 官方板的 PWM 固定频率：

| BOARD                     | PWM PINS                       | PWM FREQUENCY                              |
|---------------------------|--------------------------------|--------------------------------------------|
| Uno, Nano, Mini           | 3, 5, 6, 9, 10, 11             | 490 Hz (pins 5 and 6: 980 Hz)              |
| Mega                      | 2 - 13, 44 - 46                | 490 Hz (pins 4 and 13: 980 Hz)             |
| Leonardo, Micro, Yún      | 3, 5, 6, 9, 10, 11, 13         | 490 Hz (pins 3 and 11: 980 Hz)             |
| Uno WiFi Rev2, Nano Every | 3, 5, 6, 9, 10                 | 976 Hz                                     |
| MKR boards *              | 0 - 8, 10, A3, A4              | 732 Hz                                     |
| MKR1000 WiFi *            | 0 - 8, 10, 11, A3, A4          | 732 Hz                                     |
| Zero *                    | 3 - 13, A0, A1                 | 732 Hz                                     |
| Nano 33 IoT *             | 2, 3, 5, 6, 9 - 12, A2, A3, A5 | 732 Hz                                     |
| Nano 33 BLE/BLE Sense     | 1 - 13, A0 - A7                | 500 Hz                                     |
| Due **                    | 2-13                           | 1000 Hz                                    |
| 101                       | 3, 5, 6, 9                     | pins 3 and 9: 490 Hz, pins 5 and 6: 980 Hz |

## TH244A001 UserGuide

---

本开发板只有 3/6、5/8/9、10/11 支持 PWM 功能的脚位可使用函数设定频率。

**特别使用说明 1:** 3/6 (TM20) 、5/8/9(TM36)和 10/11(TM26)三个可以独立设置。

**默认频率均为 1000Hz;**

- `analogWriteFrequency( PWM_FRQ_D3_D6, Frq )`      设置 3/6 输出频率
- `analogWriteFrequency(PWM_FRQ_D5_D8_D9, Frq )`    设置 5/8/9 输出频率
- `analogWriteFrequency( PWM_FRQ_D10_D11, Frq )`    设置 10/11 输出频率
- `analogWriteFrequency( PWM_FRQ_ALL, Frq )`      设置 3/6, 5/8/9, 10/11 输出频率
- Frq 为设定频率, 可设定范围为 1Hz~5000Hz

**特别使用说明 2:**

**当使用 TimerOne 库功能时, PWM 10/11 的功能就不能使用。**

**当使用 MsTimer2 库功能时, PWM D3/D6 的功能就不能使用。**

**当使用 `tone()`函数时, 所有的 PWM 都不能使用。**

常用函数

**函数:** `analogWrite( pin, ulValue )`      // ulValue=0~255

**函数:** `analogWriteFrequency(PWM_FRQ_XX, Fre);`设定特定脚位输出 PWM 的频率

详见常用函数以及基本用法之 **PWM**

## 8. DAC

开发板提供一个 12bit(0~4095)的 DAC 输出通道 **DAC\_P0**。满量程输出模拟电压范围为 0V~VDD，并且输出呈线性关系。

开发板 21 为专门的 DAC 输出信道物理输出接口。可以将 21 连接到外部的模块，硬件注意事项如下图，在考虑电压稳定性上的改良对策。

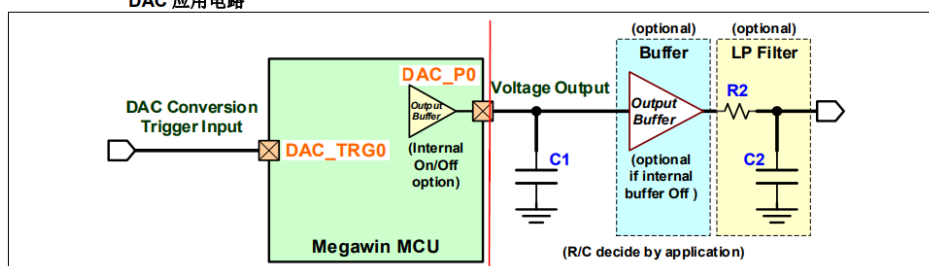
### DAC\_P0 为 MG32 专门提供的 12bit 仿真输出通道

#### DAC 应用电路

DAC 内置输出电阻负载为 7.5K 欧姆的输出缓冲，建议增加 1 个电容（**C1**）以保证更稳定的输出电压。当内部输出缓冲使能，DAC 输出电压范围为 0.2V ~ VDD-0.2V。

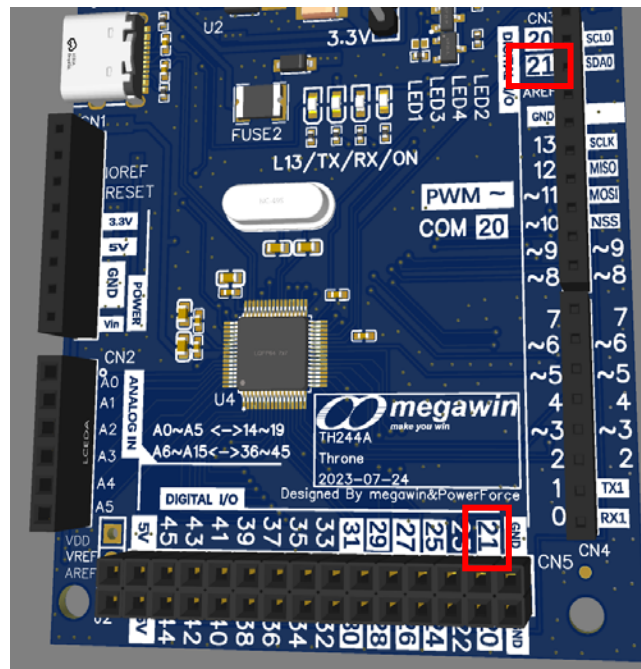
若输出电阻负载超过 7.5K 欧姆，建议增加外部输出缓冲。为了更好地输出效果，建议增加低通滤波电路（**R2/C2**）。1 个可选的 **DAC\_TRG0** 引脚能够输入 DAC 输出转换的触发信号。

DAC 应用电路



## TH244A001 UserGuide

任意一个 21 脚位都可以使用



**函数:** analogWrite(21, value)

此时 value 范围是 0~4095。DAC 采用和 PWM 控制相同的函数，对应实际输出仿真电压为 0~VDD;

量测测试程序:

```
void setup() {

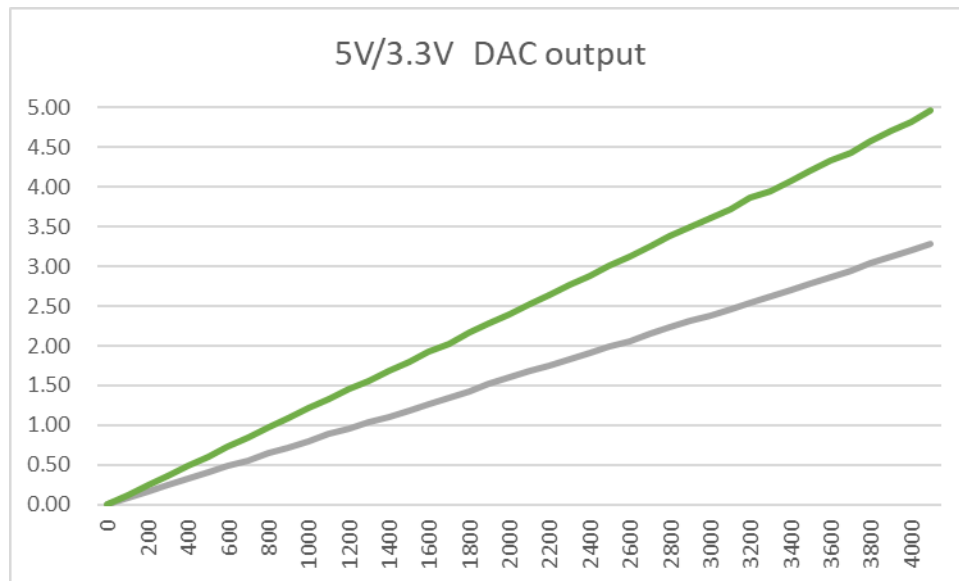
 analogWrite(21,500); //修改不同数值,仿真输出电压为 500/4095*VDD

}

void loop() {

}
```

量测曲线参考（不连接负载情况下）



## TH244A001 UserGuide

### 参考数据

| analogWrite(21, Value) | VDD3.3V |       | VDD5V |       |
|------------------------|---------|-------|-------|-------|
| Value                  | mV      | V     | mV    | V     |
| 0                      | 0       | 0     | 0     | 0     |
| 100                    | 80      | 0.08  | 122   | 0.122 |
| 200                    | 160     | 0.16  | 243   | 0.243 |
| 300                    | 240     | 0.24  | 363   | 0.363 |
| 400                    | 320     | 0.32  | 484   | 0.484 |
| 500                    | 400     | 0.4   | 606   | 0.606 |
| 600                    | 481     | 0.481 | 727   | 0.727 |
| 700                    | 560     | 0.56  | 840   | 0.84  |
| 800                    | 643     | 0.643 | 968   | 0.968 |
| 900                    | 718     | 0.718 | 1080  | 1.08  |
| 1000                   | 800     | 0.8   | 1210  | 1.21  |
| 1100                   | 884     | 0.884 | 1330  | 1.33  |
| 1200                   | 960     | 0.96  | 1450  | 1.45  |
| 1300                   | 1030    | 1.03  | 1560  | 1.56  |
| 1400                   | 1110    | 1.11  | 1680  | 1.68  |
| 1500                   | 1190    | 1.19  | 1800  | 1.8   |
| 1600                   | 1270    | 1.27  | 1920  | 1.92  |
| 1700                   | 1340    | 1.34  | 2030  | 2.03  |

|      |      |      |      |      |
|------|------|------|------|------|
| 1800 | 1430 | 1.43 | 2170 | 2.17 |
| 1900 | 1520 | 1.52 | 2280 | 2.28 |
| 2000 | 1600 | 1.6  | 2400 | 2.4  |
| 2100 | 1680 | 1.68 | 2520 | 2.52 |
| 2200 | 1750 | 1.75 | 2640 | 2.64 |
| 2300 | 1830 | 1.83 | 2760 | 2.76 |
| 2400 | 1910 | 1.91 | 2880 | 2.88 |
| 2500 | 1990 | 1.99 | 3010 | 3.01 |
| 2600 | 2060 | 2.06 | 3120 | 3.12 |
| 2700 | 2150 | 2.15 | 3260 | 3.26 |
| 2800 | 2240 | 2.24 | 3380 | 3.38 |
| 2900 | 2320 | 2.32 | 3490 | 3.49 |
| 3000 | 2380 | 2.38 | 3600 | 3.6  |
| 3100 | 2460 | 2.46 | 3720 | 3.72 |
| 3200 | 2540 | 2.54 | 3860 | 3.86 |
| 3300 | 2620 | 2.62 | 3940 | 3.94 |
| 3400 | 2700 | 2.7  | 4080 | 4.08 |
| 3500 | 2780 | 2.78 | 4200 | 4.2  |
| 3600 | 2860 | 2.86 | 4340 | 4.34 |
| 3700 | 2940 | 2.94 | 4430 | 4.43 |
| 3800 | 3040 | 3.04 | 4580 | 4.58 |

## TH244A001 UserGuide

|      |      |      |      |      |
|------|------|------|------|------|
| 3900 | 3120 | 3.12 | 4700 | 4.7  |
| 4000 | 3210 | 3.21 | 4820 | 4.82 |
| 4050 | 3240 | 3.24 | 4860 | 4.86 |
| 4095 | 3280 | 3.28 | 4960 | 4.96 |

常用函数

**函数:** analogWrite(21, Value)

详见常用函数以及基本用法之 **DAC**

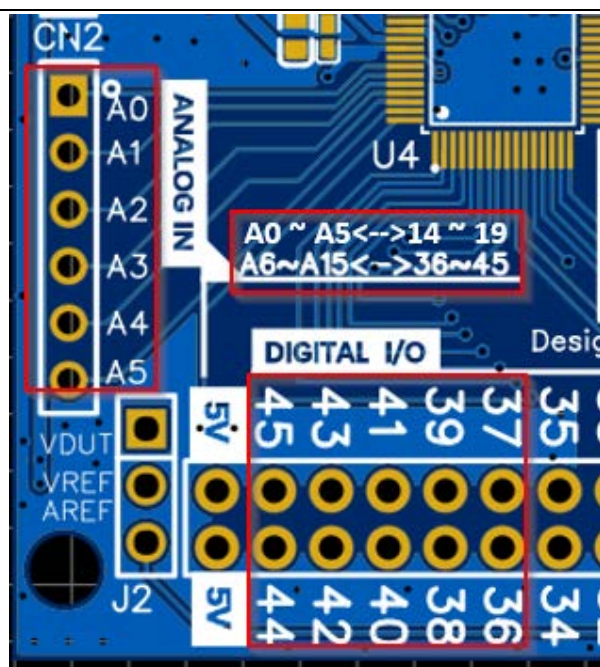
特别说明, 当 21 设置为仿真电压输出模式时, I2C0 (SCL0->20;SDA0->21) 不可用;



## 9. ADC

### 9.1 ADC 基本说明

模拟输入埠总计 16 个，其中 A0-A5 对应 14-19; A6-A15 对应 36-45



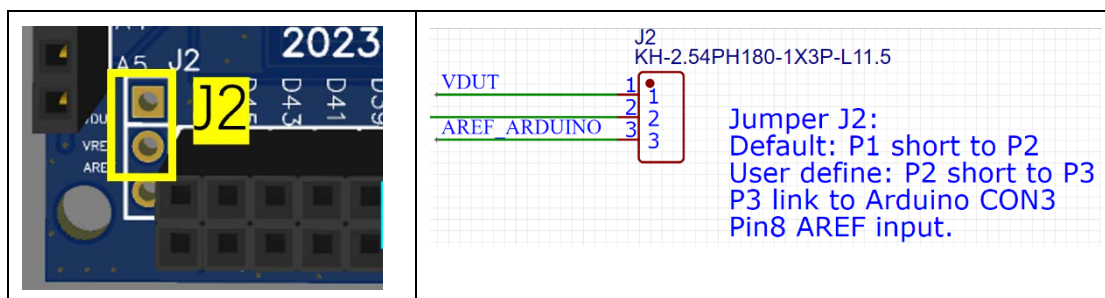
12 位 1.5Msps 的 SAR ADC

- 可设置分辨率: 12/10/8 位。开发板默认模拟输入精度为 12 位。
- 开发板提供 16 条外部输入通道
- 可设置 ADC 最高电压参考值为外部参考电压 VREF+ 或内部 IVR24
- 通过 `analogRead()` 可以读取直接读取 A0~A15，模拟量对应的映射数值，默认为 12bit，对于返回数值为 0~4095。 `analogRead(A0)`，与 `analogRead(14)` 一样。一般习惯使用 `analogRead(Ax)`，这样符合通用写法，此处 x:0~15;
- 通过 `analogReadResolution(ref)` 可以改变 ADC 的采样精度 8，10，12。默认为 12bit;

## TH244A001 UserGuide

- 通过 `analogReference(type)` 设置 ADC 模拟最高参考电压源。可以为外部 VREF+ 或者内部 IVR24; 推荐使用默认外部电压源 VREF+ 为参考电压;
- 参数 `type` 为: `AR_INTERNAL`, `AR_EXTERNAL`, `AR_DEFAULT`  
(`AR_EXTERNAL`)

开发板 MG32F02U128 可设置 ADC 最高电压参考值为 VREF+ 或内部 IVR24. VREF+ 固定为 MCU 工作电源 VDD(这是比较常用的方式), 也可以由外部设备提供参考电压 AREF。开发板使用 J2 切换两种连接, 默认为 VREF+ 直接连接 VDD。但当外设模块的模拟信号较小电压, 比如 1.8V 或者 3.3V 或者其他范围, 使用外部电压作为参考, 这样可以提高采样精度。



关于 ADC 的采样精度如下介绍:

假设仿真输入设定为 10 位分辨率, 相当于在 VDD=5V 时,

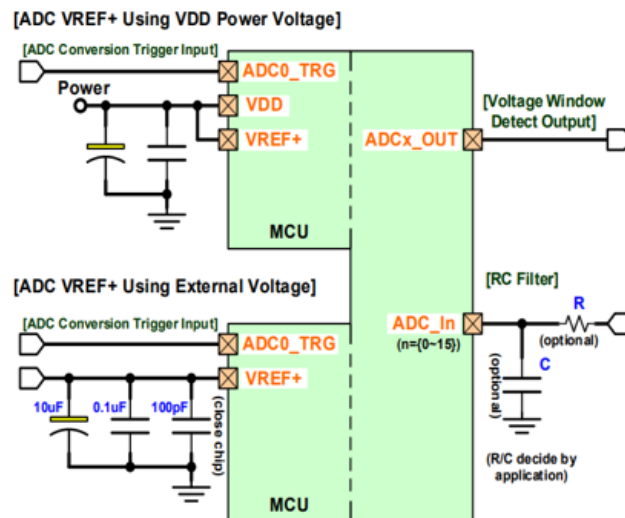
### 1. VREF+ 固定为 VDD 5V

- 当模拟输入量输入 0~5V 会被映射到 0~1023 精度是  $5/1024 = 4.882\text{mV}$  就是  
0~4.882mV 之间的电压值映像到数字 0,  
4.883mV~9.764mV 之间的电压值映像到数字 1.....  
9.765mV~14.646mV 之间的电压值映像到数字 2.....

- 当模拟输入量输入 0~3.3v, 是映射为 0~676 ( $676 \times 4.882\text{mV} = 3.3\text{v}$ )
2. 如果 VREF+ 设置为外部模块提供参考电压, 假设为 3.3V , 那么就可以实现 0~3.3v 映射到 0~1023 精度  $3.3/1024 = 3.223\text{mV}$  这样的调整, 实际精度就从 4.882mV 提升到 3.223mV
  - 3 从 10 位升到 12 位, 相同条件下, 分辨率上升明显  
 比如 VDD= 5V, VREF+ 连接 VDD 的情况,  
 10bit 时, 分辨率  $5\text{V}/1024 = 4.882\text{mV}$ , TH244A001 开发板默认为 10bit;  
 12bit 时, 分辨率  $5\text{V}/4096 = 1.221\text{mV}$ ,

## ADC 参考电压源说明:

ADC 参考电压源可来自 (1) VDD 功率通过直接连接 +VREF 引脚到 VDD 引脚 (2) 外部静压参考电压源。  
 当使用 VDD 电源作为 ADC 的参考电压时, 它必须将 +VREF 引脚跟踪连接到电源电容器后面的电流点。当使用外部参考电压源作为 ADC 参考电压时, 它必须添加一些去耦和旁路电容器, 如下图所示。  
 1 个可选的 ADCx\_TRG 引脚能够输入用于 ADC 输入转换的触发信号, 并有 1 个可选的 ADCx\_OUT 引脚用于输出内部 ADC 窗口检测状态。



## 9.2 ANALOG IN list

| No. | MG32F02U128AD64<br>LQFP64 pin No. | MG32F02U128AD64<br>Pin name | Arduino Pin name | Arduino name2 |
|-----|-----------------------------------|-----------------------------|------------------|---------------|
| 1   | 58                                | PA0/ADC0                    | 14               | A0            |

## TH244A001 UserGuide

|    |    |            |    |     |
|----|----|------------|----|-----|
| 2  | 59 | PA1/ADC1   | 15 | A1  |
| 3  | 60 | PA2/ADC2   | 16 | A2  |
| 4  | 61 | PA3/ADC3   | 17 | A3  |
| 5  | 62 | PA4/ADC4   | 18 | A4  |
| 6  | 63 | PA5/ADC5   | 19 | A5  |
| 7  | 64 | PA6/ADC6   | 36 | A6  |
| 8  | 1  | ADC7/PA7   | 37 | A7  |
| 9  | 2  | ADC8/PA8   | 38 | A8  |
| 10 | 3  | ADC9/PA9   | 39 | A9  |
| 11 | 4  | ADC10/PA10 | 40 | A10 |
| 12 | 5  | ADC11/PA11 | 41 | A11 |
| 13 | 6  | ADC12/PA12 | 42 | A12 |
| 14 | 7  | ADC13/PA13 | 43 | A13 |
| 15 | 8  | ADC14/PA14 | 44 | A14 |
| 16 | 9  | ADC15/PA15 | 45 | A15 |

常用函数

函数： analogRead()

函数： analogReadResolution(ref)

函数： analogReference(type)

函数: map ()

详见常用函数以及基本用法之 ADC

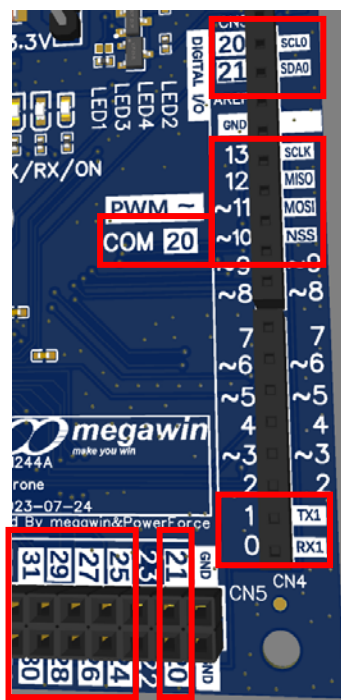
## 10. 串行 I/O

开发板可以通过多种方式与计算机、Arduino 开发板或者其他模块通信。开发板 TH244A001 提供 7 个 UART 模块、2 路 IIC 接口、1 路 SPI 接口。可以使用这些接口和计算机、外部设备或者模块通信。

Arduino 生态下的学习和开发，常常会在库文件的基础上进行。常用的三种通信方式 UART，IIC 和 SPI 均是通过对应的 Arduino 库来建立通信和命令、数据等传输。使用者需要学习和掌握相关库的运用方法。当有需要编写属于自己的库时，就需要根据通信协议，写设备间通信的专用库。

UART -> Serial.h ; IIC-> Wire.h ; SPI->SPI.h

TH244A001 正面丝印有专门白色方块标记通信用的 UART/IIC/SPI 脚位分布



TH244A001 背面丝印有专门列出通信可用的 UART/IIC/SPI 脚位分布

| Communicate Port              |         | Wire1   | Wire    | SPI     |         |         |           |
|-------------------------------|---------|---------|---------|---------|---------|---------|-----------|
| <i>SerialX.begin(baud)</i>    |         | I2C1    | I2C0    | SPI     |         |         |           |
| <i>WireX.begin(address)</i>   | SCL     | D23     | D20     | SCLK    | MISO    | MOSI    | NSS       |
| <i>SPI.beginTransaction()</i> | SDA     | D22     | D21     | D13     | D12     | D11     | D10       |
|                               |         |         |         |         |         |         |           |
|                               | Serial7 | Serial6 | Serial5 | Serial4 | Serial2 | Serial1 | PC Serial |
|                               | URT7    | URT6    | URT5    | URT4    | URT2    | URT1    | URT0      |
| TX                            | D21     | D31     | D29     | D27     | D25     | D1      | PB8       |
| RX                            | D20     | D30     | D28     | D26     | D24     | D0      | PB9       |

## 10.1 UART 端口说明

开发板提供 7 路 UART：UART0(下载程序专用)/UART1/UART2/UART4/

UART5/ UART6/ UART7。Arduino 专用库 Serial.h。UART 收发缓存为 64-bytes。

支持设置波特率为：

1200bps,2400bps,4800bps,9600bps,19200bps,38400bps,57600bps,115200bps

默认 9600bps

| MG32F02U128AD64<br>LQFP64 pin No. | MG32F02U128AD64<br>Pin name | Arduino<br>Pin name | 初始化串口函数             | UART No. | remark          |
|-----------------------------------|-----------------------------|---------------------|---------------------|----------|-----------------|
| 18                                | PB8                         | NA                  | Serial.begin(baud)  | UART0    | UART0_TX 下载程序专用 |
| 19                                | PB9                         | NA                  |                     |          | UART0_RX 下载程序专用 |
| 35                                | PC9/RXD1                    | 0                   | Serial1.begin(baud) | UART1    | UART1_RX        |
| 34                                | PC8/TXD1                    | 1                   |                     |          | UART1_TX        |
| 17                                | PB7                         | 24                  | Serial2.begin(baud) | UART2    | UART2_RX        |
| 16                                | PB6                         | 25                  |                     |          | UART2_TX        |
| 23                                | PB13                        | 26                  | Serial4.begin(baud) | UART4    | UART4_RX        |
| 24                                | PB14                        | 27                  |                     |          | UART4_TX        |
| 25                                | PB15                        | 28                  | Serial5.begin(baud) | UART5    | UART5_RX        |

## TH244A001 UserGuide

|    |          |    |                     |       |                           |
|----|----------|----|---------------------|-------|---------------------------|
| 22 | PB12     | 29 |                     |       | UART5_TX                  |
| 11 | PB1      | 30 | Serial6.begin(baud) | UART6 | UART6_RX                  |
| 10 | PB0      | 31 |                     |       | UART6_TX                  |
| 13 | MOSI/PB3 | 20 | Serial7.begin(baud) | UART7 | UART7_RX 与 IIC0_SCL<br>复用 |
| 12 | SCK/PB2  | 21 |                     |       | UART7_TX 与 IIC0_SDA<br>复用 |

### 启用和禁用串口

开发板支持 7 路串口，默认都是使能的，速度默认是 9600bps。

为了节约初始化串口收发 buffer 占用 SRAM，在不需要串口时，可以通过修改 pins\_arduino.h 中宏定义来禁用部分串口。禁用的方法如下：

在 Arduino 的路径下（根据本地电脑情况决定，x.x.x 代表实际使用的版本号）

.....\Arduino15\packages\megawin\hardware\MG32x02z\x.x.x\variants\TH244A00

1 中，打开 pins\_arduino.h 文件，如下部分：

```
//The serial port is enabled by default.
//Disable this macro could diable serial port.
#define HWSERIAL0
#define HWSERIAL1
#define HWSERIAL2
#if defined(URT3)
#define HWSERIAL3
#endif
#define HWSERIAL4
#define HWSERIAL5
#define HWSERIAL6
#define HWSERIAL7
```

默认 UART0, UART1/2/4/5/6/7 均是使能的。

如果需要禁用某个串口，就将此行注释掉即可。比如只保留 UART0, UART1，其他串口

在开发中不需要，可以如下设置，禁用除 UART0, UART1 之外的所有串口，保存文件。此后



使用 Arduino IDE 编译项目时，就不再初始化被禁用的串口。

```
//The serial port is enabled by default.
//Disable this macro could diable serial port.
#define HWSERIAL0
#define HWSERIAL1
//#define HWSERIAL2
//#if defined(URT3)
//#define HWSERIAL3
//#endif
//#define HWSERIAL4
//#define HWSERIAL5
//#define HWSERIAL6
//#define HWSERIAL7
```

### 常用函数

**函数：** Serial.begin(baud)      启动初始化串口 0

Serial**x**.begin(baud)   x:1,2,4,5,6,7

波特率 baud 是指每秒可以传输的二进制位 bit。比如波特率 9600 是指，每秒传输 9600 个二进制位 bit，换算字节就是  $9600/8=1200$  个字节 Byte。

**函数：** Serial.println() ； Serial.print()      按字符串形式打印信息，用于调试使用；

**函数：** Serial.available()   获取串行端口读取的字节数。特指到达并存储在串行接收缓冲区中的数据。一般结合 Serial.read()使用；

**函数：** Serial.read()      读取串行接收缓冲区中的数据，每次返回一个字节。需要结合 Serial.available()才能把接收缓存的所有数据读取出来；

**函数：** Serial.Write() 往串口写数据

**说明：** 重点区分 Serial.print()和 Serial.write()的使用差异

Serial.print() 发送的是字符；Serial.println()多一个回车，发送的同样是字符；

Serial.write() 发送的字节；

Serial.println(int 97)输出就是 97； 而 Serial.Write(int 97)输出时 ASCII 97 对应的字符，即字母 a

详见常用函数以及基本用法之 [UART](#)

## 10.2 IIC

开发板提供 2 路 IIC。开发板支持 IIC 作为主机或者从机使用。Arduino 专用库 Wire.h。

开发板作为主机形式和外部设备通信。使用 Wire 库来进行 IIC 通信。开发板默认 IIC 通信 SCL 速度是 100KHz，可设置为 400KHz 或者 1MHz。。

因为开发板没有在 SDA 和 SCL 设计上拉电阻（因为要考虑管脚作为 GPIO 的基本功能），所以要求用户在使用 IIC 通信时，检查所接之模块或者主机有上拉电阻，才能保证稳定可靠的通信。

具备 IIC 接口的模块，一般都是有兼容 Arduino 环境的库。设备的读写操作，都已经被库定义好。使用者可以直接使用 Wire 库里面的方法进行操作。使用者也可以根据特定需要自行设计库档，来实现更加灵活的通信需求。

| MG32F02U128AD64<br>LQFP64 pin No. | MG32F02U128AD64<br>Pin name | Arduino<br>Pin name | 初始化函数 | IIC 编号 | IIC No.                |
|-----------------------------------|-----------------------------|---------------------|-------|--------|------------------------|
| 13                                | MOSI/PB3                    | 20                  |       | IIC0   | IIC0_SCL 与 UART7_RX 复用 |

|    |           |    |               |      |                        |
|----|-----------|----|---------------|------|------------------------|
| 12 | SCK/PB2   | 21 | Wire.begin()  |      | IIC0_SDA 与 UART7_TX 复用 |
| 36 | PC10/SCL1 | 23 | Wire1.begin() | IIC1 | IIC1_SCL               |
| 37 | PC11/SDA1 | 22 |               |      | IIC1_SDA               |

| Wire 库常用函数                            | 基础说明                                                                                                                                                                                                    |
|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Wire.begin() ;<br>Wire.begin(address) | 初始化总线                                                                                                                                                                                                   |
| Wire.onReceive(receiveEvent)          | 注册 IIC 事件。从机响应主机发送数据请求<br>主机写和发送，从机接收和读                                                                                                                                                                 |
| Wire.onRequest(requestEvent)          | 注册 IIC 事件。主机请求从机发送数据<br>主机接收和读，从机写和发送                                                                                                                                                                   |
| Wire.requestFrom(address,quantity)    | 配合主机 Wire.onRequest(requestEvent)数据请求使用。表示主机向特定地址从机请求特定长度字节的数据。                                                                                                                                         |
| Wire.available()                      | 主机通过 Wire.requestFrom()请求从机数据后, 主机可以使用 Wire.available () 接受当前接收缓存中存在的数据的字节数; 使用 Wire.read () 逐个字节读取后, Wire.available () 返回值递减;<br><br>从机通过 Wire.onReceive(receiveEvent)响应主机发送数据,从机可以使用 Wire.available() |

|                                                                                                                                                                  |                                                                                                             |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
|                                                                                                                                                                  | 接受当前接收缓存中存在的数据的字节数；使用 <code>Wire.read ( )</code> 逐个字节读取后， <code>Wire.available ( )</code> 返回值递减；            |
| <code>Wire.read()</code>                                                                                                                                         | 主机或者从机接收到数据后，可以使用 <code>Wire.read()</code> 按字节读取缓存的数据；一般结合 <code>Wire.available ( )</code> 使用；              |
| <code>Wire.onRequest(requestEvent),</code><br><br><code>Wire.requestFrom(address,quantity) , Wire.available(), Wire.read()</code><br><br>一般主机请求从机数据，都会结合这几个函数使用； |                                                                                                             |
| <code>Wire.beginTransmission(address)</code>                                                                                                                     | 往发送缓存传输一个开始字符，并指定接受数据的从机地址。一般结合 <code>Wire.write()</code> 使用。                                               |
| <code>Wire.write(value)</code><br><br><code>Wire.write(string)</code><br><br><code>Wire.write(data, length)</code>                                               | 向缓存发送数据，可以是字符串，数字，或者指定数据指定长度部分。                                                                             |
| <code>Wire.endTransmission()</code><br><br><code>Wire.endTransmission(stop)</code>                                                                               | 完成主机到从机的传输过程。<br><br>结束由 <code>Wire.beginTransmissio()</code> 开始，<br>由 <code>Wire.write()</code> 排列的从机传输过程。 |
| 主机发送一般按照如下几个函数一次进行：<br><br><code>Wire.beginTransmission(address), Wire.write(), Wire.endTransmission()</code>                                                    |                                                                                                             |

可参考 Arduino 官方更多关于 Wire 的说明：

## Wire - Arduino Reference

<https://www.arduino.cc/reference/en/language/functions/communication/wire/>

详见常用函数以及基本用法之 **IIC**

## 10.3 SPI

开发板 1 路 SPI。开发板支持作为主机形式使用。Arduino 专用库 SPI.h。

SPI 接口常见是 SPI Flash 和 SPI LCD 显示屏。可以直接使用这些外设官方开发库来实现控制功能。SPI 最大速度为 18Mbps。。

| MG32F02U128AD64<br>LQFP64 pin No. | MG32F02U128AD64<br>Pin name | Arduino Pin name | 初始化函数       |
|-----------------------------------|-----------------------------|------------------|-------------|
| 51                                | NSS/PD9                     | 10               | SPI.begin() |
| 44                                | MOSI/PD2                    | 11               |             |
| 49                                | MISO/PD7                    | 12               |             |
| 50                                | SPI0_CLK/PD8                | 13               |             |

SPI 传输数据时有两种方式：MSB first 和 LSB first:

- LSBFIRST: least significant bit first 表示二进制数据的最低位先传输或者接收
- MSBFIRST : most significant bit first 表示二进制数据的最高位先传输或者接收

要求发送的方式和接收的方式必须一致。比如主机按照 MSBFIRST 传送数据，从机就要按照 MSBFIRST 接收数据，反之亦然。

SPI 传输模式支持四种 MODE0/1/2/3，区别在于 SCLK 空闲状态和采样时机差异。

SCLK 的空闲状态由 CPOL(时钟极性)决定:

- 当 CPOL 为 0 时，时钟空闲时候的电平是低电平；
- 当 CPOL 为 1 时，时钟空闲时候的电平是高电平；

采样时机由 CPHA（时钟相位）决定：

- 当 CPHA 为 0 时，时钟周期的第一个边缘开始采集数据；
- 当 CPHA 为 1 时，时钟周期的第二个边缘开始采集数据；

| CPOL | CPHA | Mode  |
|------|------|-------|
| 0    | 0    | MODE0 |
| 0    | 1    | MODE1 |
| 1    | 0    | MODE2 |
| 1    | 1    | MODE3 |

采用何种方式通信，需要查询从机的规格，约定主机和从机都支持的模式进行即可。

主要常用函数，

|                        |                   |
|------------------------|-------------------|
| SPI.begin()            | 开关总线函数            |
| SPI.end()              |                   |
| SPI.beginTransaction() | 带 SPI 端口配置的总线开关函数 |
| SPI.endTransaction ()  |                   |

|                                                                                                                                                                                                                                                                                |                                                             |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------|
| <p>SPISettings</p> <p>作为 SPI.beginTransaction()的参数使用</p>                                                                                                                                                                                                                       | <p>配置 SPI 端口的对象，用于设置速度，传输数据的顺序和传输模式选择</p>                   |
| <p>SPI.transfer()</p>                                                                                                                                                                                                                                                          | <p>数据传输函数</p>                                               |
| <p>SPI.setClockDivider(SPI_CLOCK_DIVx)</p> <ul style="list-style-type: none"> <li>● SPI_CLOCK_DIV2</li> <li>● SPI_CLOCK_DIV4</li> <li>● SPI_CLOCK_DIV8</li> <li>● SPI_CLOCK_DIV16</li> <li>● SPI_CLOCK_DIV32</li> <li>● SPI_CLOCK_DIV64</li> <li>● SPI_CLOCK_DIV128</li> </ul> | <p>设置 SPI 时钟，通过 CPU 时钟（36MHz）分配得到。一般不设置，保持预设最高时钟 18MHz。</p> |

详见常用函数以及基本用法之 **SPI**

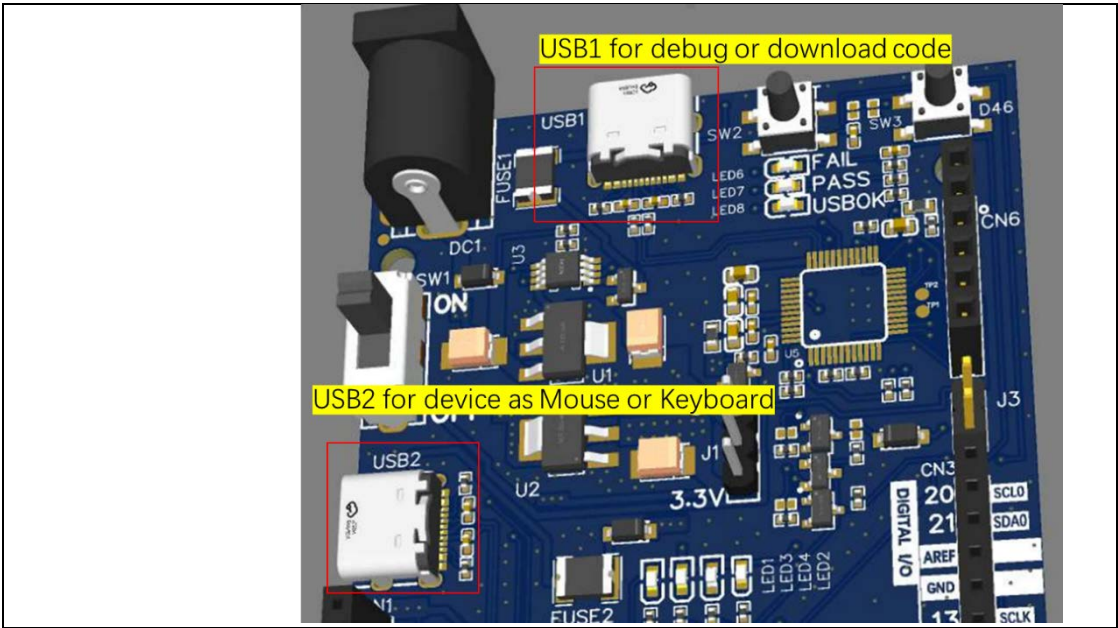
可参考 Arduino 官方更多关于 SPI 的说明：

SPI - Arduino Reference

<https://www.arduino.cc/reference/en/language/functions/communication/spi/>

11. USB 界面

开发板提供两个 USB 接口 (Type C 形式)。其中 USB1 专门用于调试和下载程序使用；  
USB2 可用于 device，连接到计算机端，作为 Mouse 或者 Keyboard。

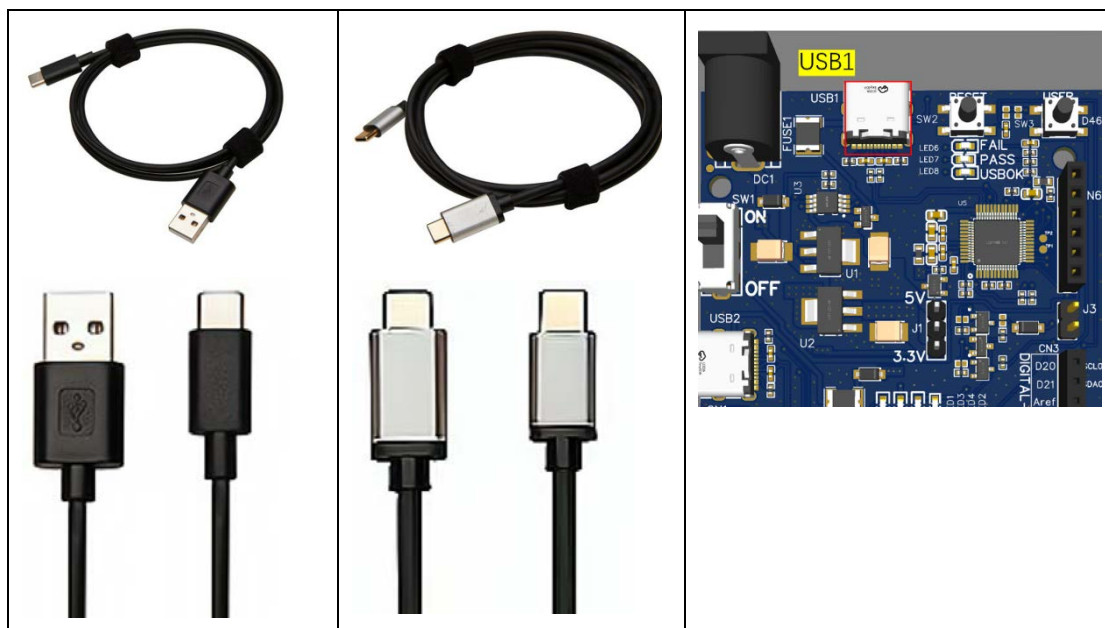


11.1 USB1

USB1 作为调试和下载端口。使用 USB A TO C 数据线，或者 USB 3.1 C TO C 数据线  
线材连接到 PC 端。USB1 主要用来下载程序使用。PC 会识别到虚拟串口，Arduino IDE 通  
过串口实现程序下载，或者数据的传输。

|                                                 |                                                                        |                   |
|-------------------------------------------------|------------------------------------------------------------------------|-------------------|
| <b>USB A TO C 数据线；</b><br>必须含 USB2.0 信号线(DP/DM) | <b>USB 3.1 C TO C 数据线；</b><br>实际不会用 USB3.1 信号，只会用<br>USB2.0 信号线(DP/DM) | 用于下载调试的专用 USB1 界面 |
|-------------------------------------------------|------------------------------------------------------------------------|-------------------|





## 11.2 USB2

开发板 USB2 作为 device 端口，可以通过 Keyboard.begin()或者 Mouse.begin()初始化为一个 native USB HID Keyboard 或者 native USB HID Mouse,可以作为键盘或者鼠标。

通过 Arduino 的 Keyboard.h 库实现键盘字符输入。

通过 Arduino 的 Mouse.h 库，可以轻松实现鼠标功能（比如定义五个按键分别代表前后左右移动和点击，就可以模拟鼠标移动和点击长按等运用，主要运用 digitalRead()获取按键状态；可以结合游戏杆来实现鼠标移动和点击功能（主要使用 ADC 来检测游戏杆调整时出现的电压数值来判定相对位置）。

USB2 作为 device 端口，可以作为鼠标键盘等运用



# TH244A001 UserGuide

## MG32F02U128    USB 作为 device

- USB1.1 的低速与全速
  - 支持 USB 规格为 v1.1
- 支持 USB 挂起/恢复和远程唤醒
- 支持 8 个带输入和输出的端点
  - 每个断点都灵活支持输入、输出和同时输入输出
  - 除端点-0 外 7 个端点可设置复位位地址值
  - 每个端点支持设置不同的起始地址进行独立收发
  - 每个端点独立可设置双倍缓冲模式
- 支持端点 0 控制发送
- 除端点-0 外所有端点支持中断、批量和同步传输
- 支持设置 USB SRAM 空间为 512 字节用于共享所有端点
- 支持 USB 2.0 链路电源管理
- 端点-3、4 支持使用 DMA 收发数据

### 开发板 USB2 作为键盘或者鼠标相关常用函数

|                                                                                       |                                       |        |
|---------------------------------------------------------------------------------------|---------------------------------------|--------|
| Native USB HID Mouse<br><br>button 通常有三个按键<br><br>●    MOUSE_LEFT (default)<br><br>左键 | Mouse.begin()                         | 开启鼠标   |
|                                                                                       | Mouse.end()                           | 关闭鼠标   |
|                                                                                       | Mouse.click();Mouse.click(button)     | 鼠标点击事件 |
|                                                                                       | Mouse.press();Mouse.press(button)     | 长按鼠标   |
|                                                                                       | Mouse.release(),Mouse.release(button) | 鼠标按键释放 |

|                                                                                               |                                           |          |
|-----------------------------------------------------------------------------------------------|-------------------------------------------|----------|
| <ul style="list-style-type: none"> <li>● MOUSE_RIGHT 右键</li> <li>● MOUSE_MIDDLE 中键</li> </ul> | Mouse.isPressed(),Mouse.isPressed(button) | 鼠标状态检测   |
|                                                                                               | Mouse.move(x, y, wheel)                   | 鼠标移动事件   |
| Native USB HID Keyboard                                                                       | Keyboard.begin()                          | 开启键盘     |
|                                                                                               | Keyboard.end()                            | 关闭键盘     |
|                                                                                               | Keyboard.press()                          | 长摁键盘     |
|                                                                                               | Keyboard.release()                        | 释放按键     |
|                                                                                               | Keyboard.releaseAll()                     | 释放全部按键   |
|                                                                                               | Keyboard.print()                          | 敲击事件     |
|                                                                                               | Keyboard.println()                        | 带回车的敲击事件 |
|                                                                                               | Keyboard.write()                          | 发送单个敲击指令 |

详见常用函数以及基本用法之 USB

### 12. RTC

Arduino 没有提供官方的标准函数。开发包设计了一个类 **class MG32x02z\_RTC**，使用者可以直接使用如下方式来使用开发板的 RTC 功能。因为 RTC 是依赖开发板运行，当断电后，RTC 就会停止。重新上电后，需要重新初始化 RTC。

针对开发板专门定义类 `class MG32x02z_RTC`，来实现 RTC 相关功能。RTC 使用一般先启动 RTC，再设置时间和日期。在后续实际需要的时候获取时间信息。

在使用开发包的 RTC 功能时，分两部分进行：初始化 RTC 和获取时间信息。

常见函数：

初始化 RTC:

```
/*这部分初始化一般放在 setup 中*/

Rtc.begin(); //启动 RTC

Rtc.setDate(day, month, year); //设置日期

Rtc.setTime(hour, minute, sec); //设置时间
```

获取时间信息：

```
/*这部分可用作实际调用需求*/

Rtc.getHour(); //获取当前的小时

Rtc.getMinute(); //获取当前的分钟

Rtc.getSecond(); //获取当前的秒

Rtc.getDay(); //获取当前的日期

Rtc.getMonth(); //获取当前的月份
```

```
Rtc.getYear();
```

```
//获取当前的年份
```

详见常用函数以及基本用法之 [RTC](#)

## 13. SLEEP/STOP Mode

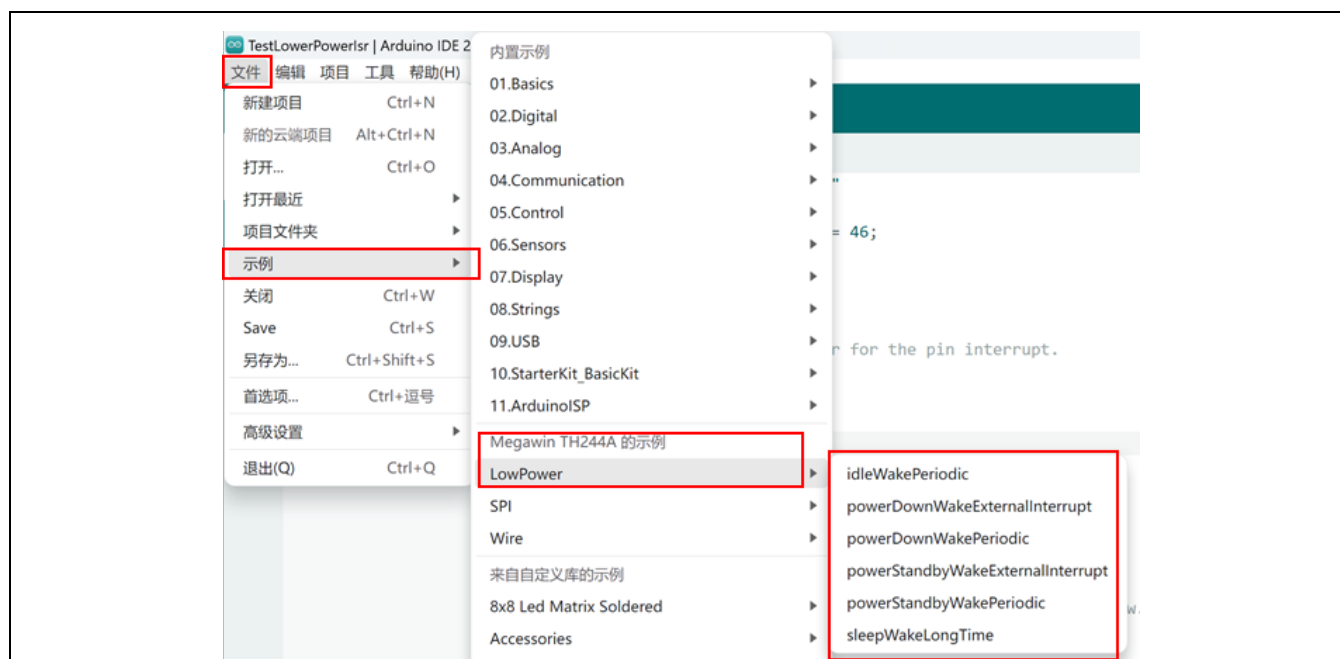
Megawin M0 MCU MG32F02U128 的电源控制器一共支持 ON, SLEEP, STOP 三种电源运行模式。其中两种掉电模式: SLEEP 模式和 STOP 模式。掉电模式可以降低芯片功耗和提供多种不同的针对芯片应用程序的节电方案。arduino 开发包从 2.2.0 版本开始支持掉电模式在 Arduino IDE 中使用。

- ON 模式: ON 模式下, CPU 能够以全速运行, 所有的外设均可以满功率正常进行工作。同时, 这些模块也可以为了降低功耗而独立的进行启用和禁用;
- SLEEP 模式: SLEEP 模式下, 只有 CPU 会被冻结, 所有的外设可以自行设置继续工作或者休眠。在该模式下, 芯片可被关联的中断或者事件发生唤醒; □
- STOP 模式: STOP 模式可提供最低的功耗, 与 SLEEP 模式不同的地方是 CPU 进入深度睡眠模式, 除部分特殊模块或设备外, 所有外设均被禁用。这部分特殊模块或设备可被设置在 STOP 模式中是否继续工作, 这些特殊模块分别为、IWDT、RTC、CMP、LVR、BOD0、BOD1、BOD2。内部的电压调节器也同样会在低电量模式运行。在该模式下, 芯片可被一些外部输入线路 (GPIO) 和一些事件检测唤醒。

这里主要支持外部引脚中断 `attachInterrupt(pin,ISR,MODE)`和看门狗 IWDT 唤醒。

**备注：**开发包示例程序针对 STOP 模式和 SLEEP 模式的唤醒，当前设计了外部引脚中断唤醒和 IWDG（看门狗唤醒，使用 ILRCO 32KHz 作为时钟源）示例。

## 示例程序说明



| 示例程序                                  | 模式    | 基本说明                      |
|---------------------------------------|-------|---------------------------|
| idleWakePeriodic.ino                  | STOP  | 进入 STOP 模式持续一段时间，自动唤醒     |
| powerDownWakeExternalInterrupt.ino    | STOP  | 进入 STOP 模式，通过外部中断唤醒       |
| powerDownWakePeriodic.ino             | STOP  | 进入 STOP 模式，通过 IWDG 看门狗唤醒  |
| sleepWakeLongTime.ino                 | STOP  | 进入 STOP 模式，通过 IWDG 看门狗唤醒  |
| powerStandbyWakeExternalInterrupt.ino | SLEEP | 进入 SLEEP 模式，通过外部中断唤醒      |
| powerStandbyWakePeriodic.ino          | SLEEP | 进入 SLEEP 模式，通过 IWDG 看门狗唤醒 |

## TH244A001 UserGuide

| 示例程序                                  | 模式    | 默认开关                           | 可选开关                          | 说明                                                  |
|---------------------------------------|-------|--------------------------------|-------------------------------|-----------------------------------------------------|
| idleWakePeriodic.ino                  | STOP  | ADC_OFF<br>BOD_OFF<br>USB_OFF  |                               | 实际运行只有<br>IWDT,RTC、外加<br>IO 中断以便唤醒                  |
| powerDownWakeExternalInterrupt.ino    | STOP  | ADC_OFF<br>BOD_OFF<br>USB_OFF  | ADC_OFF<br>BOD_OFF<br>USB_OFF | 除可选模块外,<br>实际运行只有<br>IWDT,RTC、外加<br>IO 中断以便唤醒       |
| powerDownWakePeriodic.ino             | STOP  | ADC_OFF<br>BOD_OFF<br>USB_OFF  | ADC_OFF<br>BOD_OFF<br>USB_OFF | 除可选模块外,<br>实际运行只有<br>IWDT,RTC、外加<br>IO 中断以便唤醒       |
| sleepWakeLongTime.ino                 | STOP  | ADC_OFF<br>BOD_OFF<br>USB_OFF  | ADC_OFF<br>BOD_OFF<br>USB_OFF | 除可选模块外,<br>实际运行只有<br>IWDT,RTC、外加<br>IO 中断以便唤醒       |
| powerStandbyWakeExternalInterrupt.ino | SLEEP | ADC_OFF<br>USB_OFF<br>SYS_TICK | ADC_OFF<br>USB_OFF            | SLEEP 模式下,<br>实际所有模块处<br>于工作状态, 理<br>论上所有模块都<br>可唤醒 |



|                              |       |                                |                    |                                                                    |
|------------------------------|-------|--------------------------------|--------------------|--------------------------------------------------------------------|
|                              |       |                                |                    | SLEEP 模式下，<br><br>实际所有模块处<br><br>于工作状态，理<br><br>论上所有模块都<br><br>可唤醒 |
| powerStandbyWakePeriodic.ino | SLEEP | ADC_OFF<br>USB_OFF<br>SYS_TICK | ADC_OFF<br>USB_OFF |                                                                    |

使用这些功能，必须加载 LowPower.h

```
#include "LowPower.h"
```

详见常用函数以及基本用法之 **SLEEP/STOP mode**

### 14. IWDT

#### 14.1 独立看门狗定时器

当看门狗定时器被使能时，软件需要总是在定时器超时之前复位定时器，当看门狗定时器被复位，定时器将会将重装载时间值并重新开始计时。若芯片由于受到干扰失控时，固件有可能会因为不能复位定时器而导致定时器超时的到来，它会让 IWDT 产生复位事件，并发送到复位源控制器（RST）并作为热复位或冷复位来进行复位。也可以运用 `wdt_disableRST()` 设置看门狗中断时不进行系统复位，而是执行专用中断函数 `ISR(WDT_vect){}`。

开发包支持在 SLEEP 或者 STOP 节能模式下，通过 IWDT 进行唤醒。

**IWDT 计时是基于 ILRCO 32KHz。支持如下时间常数。**

|                          |                            |                                |
|--------------------------|----------------------------|--------------------------------|
| <code>SLEEP_15MS</code>  | <code>= WDTO_15MS,</code>  | <code>//看门狗定时器 15ms 超时</code>  |
| <code>SLEEP_30MS</code>  | <code>= WDTO_30MS,</code>  | <code>//看门狗定时器 30ms 超时</code>  |
| <code>SLEEP_60MS</code>  | <code>= WDTO_60MS,</code>  | <code>//看门狗定时器 60ms 超时</code>  |
| <code>SLEEP_120MS</code> | <code>= WDTO_120MS,</code> | <code>//看门狗定时器 120ms 超时</code> |
| <code>SLEEP_250MS</code> | <code>= WDTO_250MS,</code> | <code>//看门狗定时器 250ms 超时</code> |
| <code>SLEEP_500MS</code> | <code>= WDTO_500MS,</code> | <code>//看门狗定时器 500ms 超时</code> |
| <code>SLEEP_1S</code>    | <code>= WDTO_1S,</code>    | <code>//看门狗定时器 1s 超时</code>    |
| <code>SLEEP_2S</code>    | <code>= WDTO_2S,</code>    | <code>//看门狗定时器 2s 超时</code>    |
| <code>SLEEP_4S</code>    | <code>= WDTO_4S,</code>    | <code>//看门狗定时器 4s 超时</code>    |
| <code>SLEEP_8S</code>    | <code>= WDTO_8S,</code>    | <code>//看门狗定时器 8s 超时</code>    |

## 14.2 IWDT 专用中断函数

```
ISR(WDT_vect)

{

 // do anything

 wdt_reset(); // 喂狗，重置定时器；

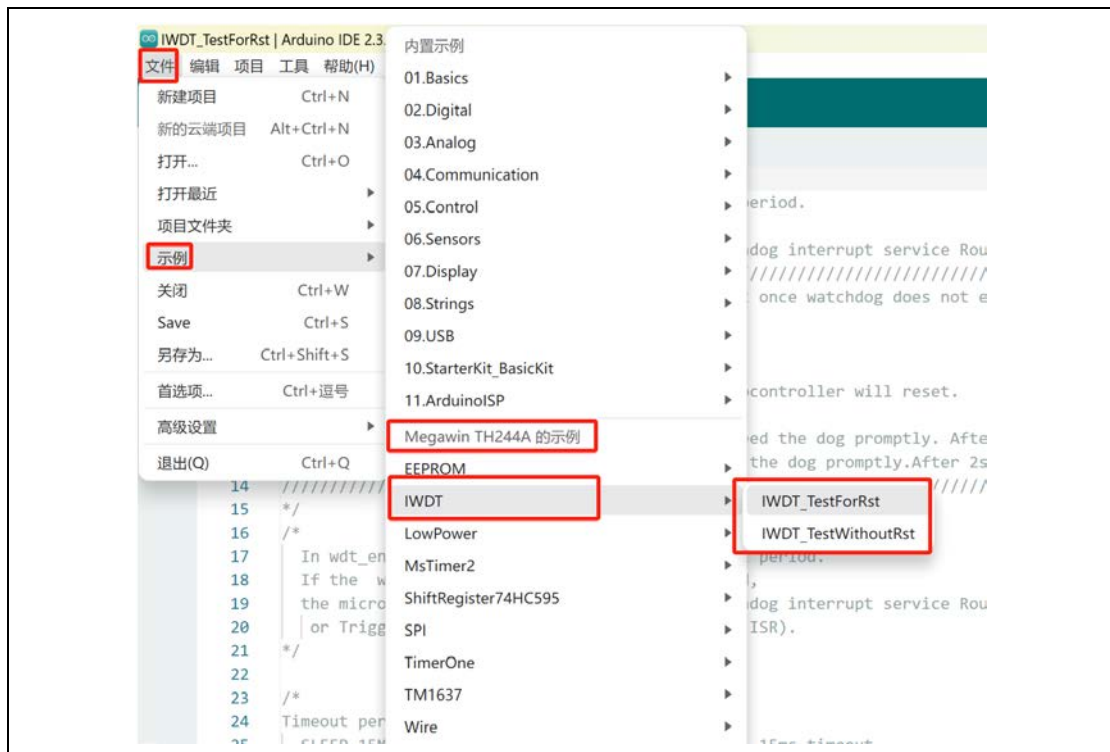
}
```

## 14.3 IWDT 库主要用法

基于 megawin MG32F02U128 设计的 IWDT 库的主要用法：

- `wdt_disableRST();` //禁用 IWDT 中断复位系统功能，一般结合 `ISR(WDT_vect){}` 运用.
- `wdt_enableRST();` //使能 IWDT 中断复位系统功能，按时喂狗，当 MCU 系统程序跑飞或者死机时，无法喂狗，达到复位系统的运用
- `wdt_enable(WDTO_4S);` //启动看门狗功能，定时器 timeout 为 `WDTO_4S`，即 4s
- `wdt_disable()` //关闭看门狗功能
- `wdt_reset();` // 喂狗，超出 timeout 时间，没有喂狗，就会进入 IWDT 专属中断 `ISR(WDT_vect){}` 或者 RESET MCU

# TH244A001 UserGuide



使用这些功能，必须加载 IWDt.h

```
#include "IWDt.h"
```

详见常用函数以及基本用法之 [IWDt](#)

## 15. 定时器中断 TimerOne

### 15.1 TimerOne 介绍

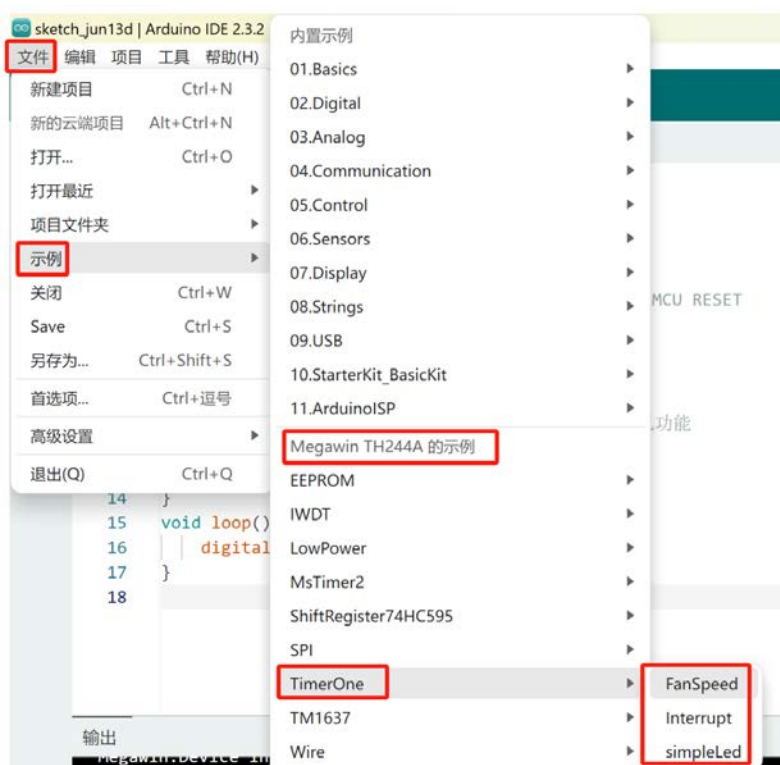
TimerOne 库是使用 MCU 内部定时器, 实现灵活的时间间隔控制。从开发包 2.3.0 开始支持定时器中断 TimerOne 库。基本函数和使用方法和 Arduino 当前平台相同的方式。

库文件只适用于 MG32F02U128 平台-TH244A001。

TimerOne 库支持设置从 D42/D43 输出 PWM 信号。当使用 TimerOne 时, D10/D11 的 PWM 功能就不能使用。

TimerOne 库支持中断设置中断函数。

示例程序:



### 15.2 TimerOne 库控制输出 PWM

`const int pin_PWM = 43; //设置定义需要设置 PWM 的输出脚位, 只有 42,`

`43 可用; 不需要输出 PWM, 可不做设定`

`Timer1.initialize(timeout); //单位 us, timeout=40 us ,就表示 PWM 频率为 25`

`KHz;`

`Timer1.pwm(pin_PWM, duty); //此时 duty 为 0~1023。实际对应正占空比`

`0~100%;`

### 15.3 TimerOne 库定时中断

`Timer1.initialize(timeout); // 单位 us, timeout=1000000us , 初始化定时器`

`为 1s; 这里时间可以支持最大 10s; 一般建议作为中断使用时, 这里时间最大设置`

`1s, 以保证时间准确性, 需要更大的时间可以使用循环计数进行;`

`Timer1.attachInterrupt(timer1_ISR); //设置中断回调函数`

`void timer1_ISR(){}//定时器中断处理函数`

使用这些功能, 必须加载 **TimerOne.h** 库

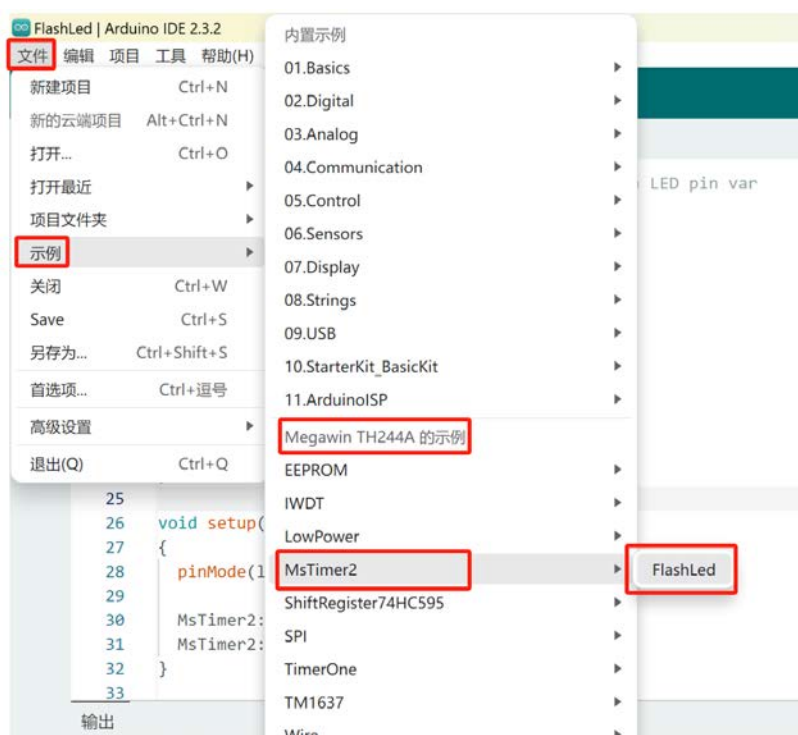
`#include <TimerOne.h>`

详见常用函数以及基本用法之 [TimerOne](#)

## 16. 定时器中断 Mstimer2

MsTimer2 库是使用 MCU 内部定时器, 实现灵活的时间间隔控制。从开发包 2.3.0 开始支持定时器中断 Mstimer2 库。基本函数和使用方法和 Arduino 当前平台相同的方式。库文件只适用于 MG32F02U128 平台-TH244A001。

当使用 MsTimer2 库定时中断, D3/D6 的 PWM 功能就不能使用。



### 库函数

```
MsTimer2::set(unsigned long ms, void (*f)());
```

```
//设置定时中断的时间间隔和调用的中断服务程序。
```

```
//ms 表示的是定时时间的间隔长度，单位是 ms;
```

```
//void(*f)()表示被调用中断服务程序函数名
```

```
MsTimer2::start();//启动定时器中断
```

```
MsTimer2::stop(); //停止定时中断,随时可以用 start 方法重新启动中断
```

```
void(*f)(){} //定时器中断处理函数
```

使用这些功能，必须加载 **MsTimer2.h** 库

```
#include <MsTimer2.h>
```

详见常用函数以及基本用法之 **MsTimer2**



## 17. EEPROM

Megawin-MG32x02z-TH244A001-2.3.0 的 EEPROM，可以掉电不丢失存储的数据。TH244A001 开发包有设计划分闪存区(Flash memory) 来模拟 Arduino 的 EEPROM。因为模拟的 EEPROM 会在初始化后，在 SRAM 中开辟相同大小的空间。为了高效运用好有限的内存 SRAM，所以 TH244A001 采用默认 512 字节的 falsh 模拟作为 EEPROM。开发者如果需要更大的空间，可以在开发包的安装路径下手动修改 EEPROM.h 文件来实现。开发包中的 EEPROM.h 只适用于 megawin MG32F02U128 TH244A001 平台。

特别说明的是闪存和 EEPROM 支持对其数据的无限制读取,但限制了数据的写入次数。EEPROM 对写入操作的数量有限制，因此要注意不要将写入操作放在循环或其他循环执行函数中。



## TH244A001 UserGuide

---

EEPROM.h 库中默认已经定义了一个名为 EEPROM 的 EEPROMClass 对象。基本的使用方式如下：

- 加载#include <EEPROM.h>来使用 EEPROM;

EEPROM 库中默认已经定义了一个名为 EEPROM 的对象，默认大小为 512bytes。

可以直接使用该对象即可。如果你需要自己定义其他名字的对象，可以使用 EEPROMClass 去定义。

默认定义好的 EEPROM 对象大小为 512 字节，用户可操作地址为 0~511。

在默认情况下，使用 EEPROM 时：

- ◆ 首先加载 EEPROM.begin(size)，size 为需要读写的数据字节最大地址+1，取值 1~512；
- ◆ 使用 EEPROM.write(addr, data)来写数据，参数分别为地址&单字节数据；
- ◆ 所有写数据后，需要通过 EEPROM.commit()提交，使数据从暂存区保存到 flash 中，实现掉电保护。
- ◆ 使用 EEPROM.read(addr)来读指定地址的单字节数据；
- 修改 EEPROM 容量

如果有需要更大的容量，用户可以通过修改库文件 EEPROM.h 中的宏来实现。注意，不能再 Arduino IDE ino 项目文件中定义这个宏。

在 Arduino 的路径下（根据本地电脑情况决定，x.x.x 代表实际使用的版本号）  
.....\Arduino15\packages\megawin\hardware\MG32x02z\x.x.x\libraries\EEPROM\src  
中修改 EEPROM.h 中的 EEPROM\_SIZE 宏参数。

支持从 512 bytes 到 8K bytes 可选择，默认设置为 512

```
//=====Support EEPROM_SIZE list begin=====

#define EEPROM_SIZE 512 //default set EEPROM 0.5K 512 bytes

//#define EEPROM_SIZE 1024 // set EEPROM 1K 1024 bytes

//#define EEPROM_SIZE 1536 // set EEPROM 1.5K 1536 bytes

//#define EEPROM_SIZE 2048 // set EEPROM 2K 2048 bytes

//#define EEPROM_SIZE 2560 // set EEPROM 2.5K 2560 bytes

//#define EEPROM_SIZE 3072 // set EEPROM 3K 3072 bytes

//#define EEPROM_SIZE 3584 // set EEPROM 3.5K 3584 bytes

//#define EEPROM_SIZE 4096 // set EEPROM 4K 4096 bytes

//#define EEPROM_SIZE 4608 // set EEPROM 4.5K 4608 bytes

//#define EEPROM_SIZE 5120 // set EEPROM 5K 5120 bytes

//#define EEPROM_SIZE 5632 // set EEPROM 5.5K 5632 bytes

//#define EEPROM_SIZE 6144 // set EEPROM 6K 6144 bytes

//#define EEPROM_SIZE 6656 // set EEPROM 6.5K 6656 bytes

//#define EEPROM_SIZE 7168 // set EEPROM 7K 7168 bytes

//#define EEPROM_SIZE 7680 // set EEPROM 7.5K 7680 bytes

//#define EEPROM_SIZE 8192 // set EEPROM 8K 8192 bytes

//=====Support EEPROM_SIZE list end=====
```

常用函数:

EEPROM.begin(Size);           //申请读写操作的大小,Size 是必须小于 EEPROM\_SIZE

定

## TH244A001 UserGuide

---

义的大小的。比如默认 EEPROM\_SIZE=512, 那么此处 Size 就为 1~512;

当需要写入或者读取单字节时:

EEPROM.write(address, data) //默认 EEPROM\_SIZE=512 时, 可用地址为 0~511.

write 方法是以字节为存储单位, data 只能为 0x00~0xFF,用于存储读取到的数据;

EEPROM.commit (); //使数据从暂存区 SRAM 保存到 flash 中, 实现掉电保护

EEPROM.read (address) //读取指定地址的数据,read()方法是以字节为单位读取

当需要写入或者读取多个字节时, 比如浮点数据或者整形数据或者其他数据结构 (需要多字节存储数据), 可以用如下方法来读写:

EEPROM.put(address, data) //向指定地址写数据, data 为待存储的数据

EEPROM.get(address, data) //读取指定地址的数据, data 为存储读取到的数据

使用这些功能, 必须加载 **EEPROM.h** 库

**#include <EEPROM.h>**

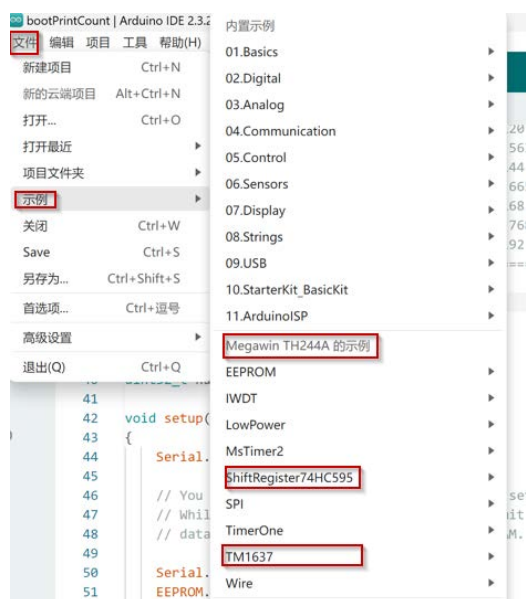
详见常用函数以及基本用法之 [EEPROM](#)

## 18. 多段数码管库显示

必须加载 TM1637Display.h 或者 ShiftRegister74HC595.h, 可实现数码管的显示控制。

#include <TM1637Display.h>或者#include <ShiftRegister74HC595.h>

基本操作可参考示例。

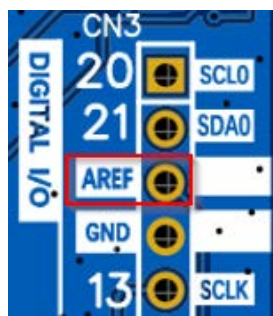


## 19. 其他重要说明

### 19.1 AREF (CN3)

模拟输入参考电压。通过 J2 来切换 VDD 或者 AREF 作为 VREF+ 的输入。可以参考 ADC 部分说明。analogReference() 用来选择是外部 VREF+ 作为参考输入，或者 IVR24 作为参考电压。详见 **ADC** 部分。

AREF 是作为 VREF+ 的另外一种电压输入，当需要使用 AREF 这个电压作为 VREF+ 参考电压时，需要设置 2.54mm 2pin 跳帽 (Jumper) J2 到对应位置。



### 19.2 RESET PIN(CN1-RESET)

连接扩展板上的复位信号来源，可以实现通过扩展板或者其他模块产生的特定低信号(可以是中断，可以是 GPIO 等)来复位开发板主控制器 MG32F02U128。作用和开发板上按下 RESET KEY 一样。

RESET 位于 CN1，是用于连接外部信号来复位主控芯片 MG32F02U128。

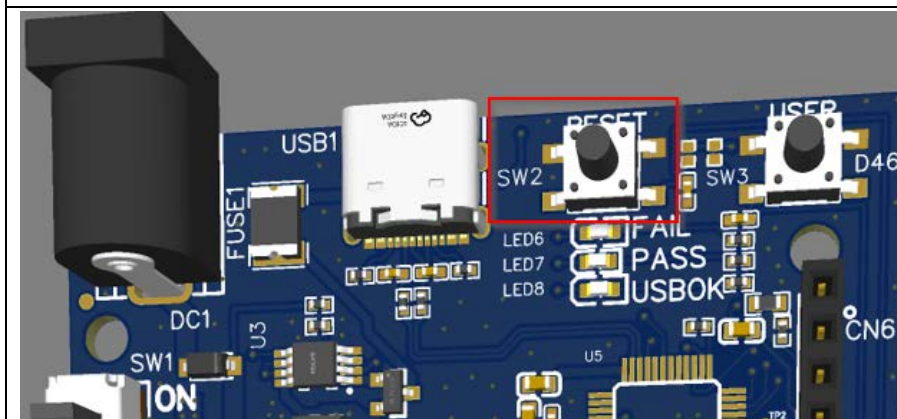


## 19.3 RESET KEY: SW2

按下 RESET KEY SW2，复位开发板主控制器 MG32F02XXX。SW2 产生的复位信号，就是连接带 PC6 RSTN 引脚，当按下时，就会复位芯片。

按键 SW2，按下，RESET 信号被拉低到地，从而产生复位信号。

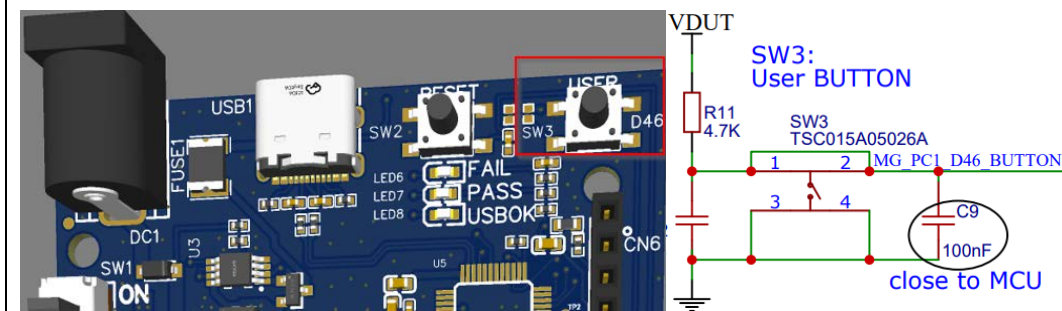
当按下时，状态为 LOW；当没有按下，状态为 HIGH。



## 19.4 USER KEY SW3

连接到 46，使用者可以运用这个作为单独按键使用，硬件上已经设计了上拉电阻。可以使用 46 来访问这个开关状态。

用户可以程序设计使用的按键 SW3，可以用 Arduino 函数设置为数字输入状态，并读取高低状态，pinMode(46, INPUT)，digitalRead(46)获取该脚状态。当按下时，状态为 LOW；当没有按下，状态为 HIGH；

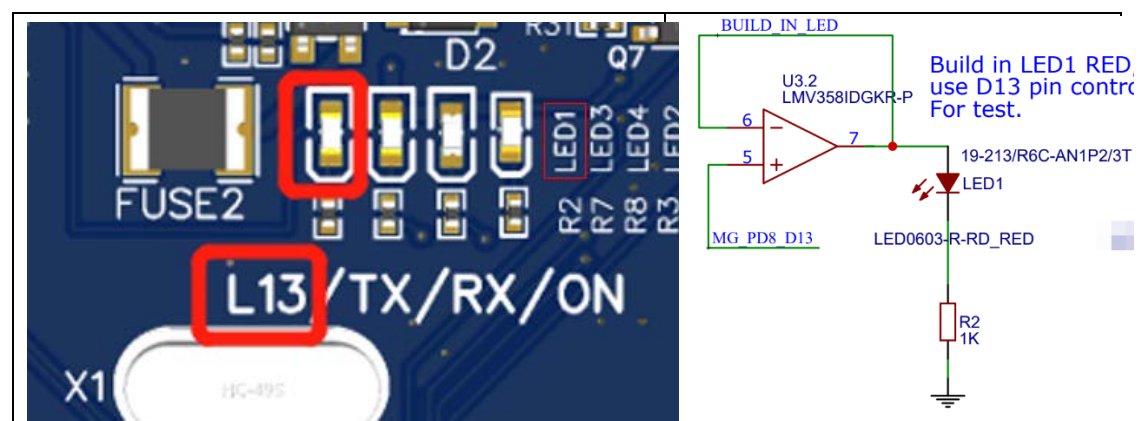


### 19.5 L13: LED1

数位脚 13，宏定义为 LED\_BUILTIN。程序设计中可以使用 13 或者 LED\_BUILTIN；其通过比较器连接到红光 LED，硬件位号 LED1。13 脚还可以配置为 SPI 的 SCLK。线路中运用运放是为了最大程度减少 LED1 这个负载对 13 脚 SPI 功能的影响。

当引脚 13 是高电平，LED1 亮；引脚为低电平时，LED1 灭。

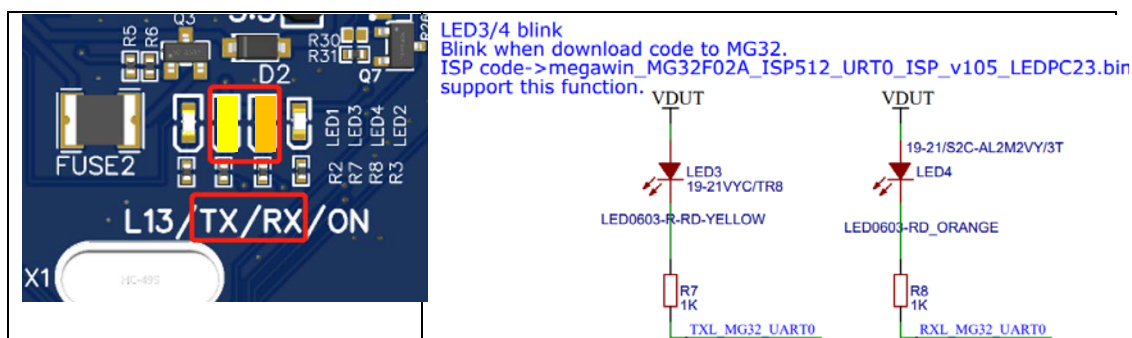
出厂默认程序：红色 LED1 以 ON 1s，OFF 1s 闪烁



### 19.6 TX LED/RX LED

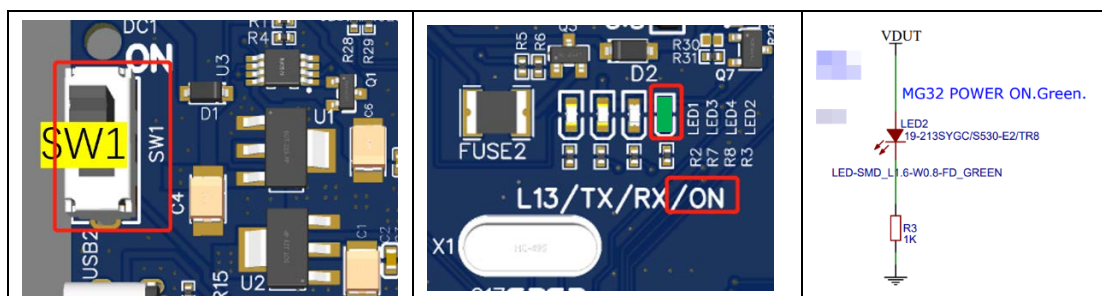
当通过 PC 升级 MG32F02U128 AP code 过程中，YELLOW LED3/ORANGE LED4 不断闪烁，升级完成，就会一直处于不亮的状态。





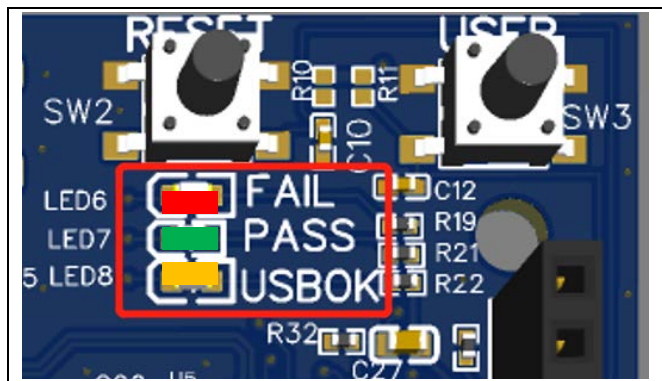
## 19.7 主控器电源开关 SW1 / Power ON LED2

用户有需要断电或者 repower 主控芯片时，通过切换 SW1 来实现。Green LED2 可以表示 MG32F02XXX 是否通电。



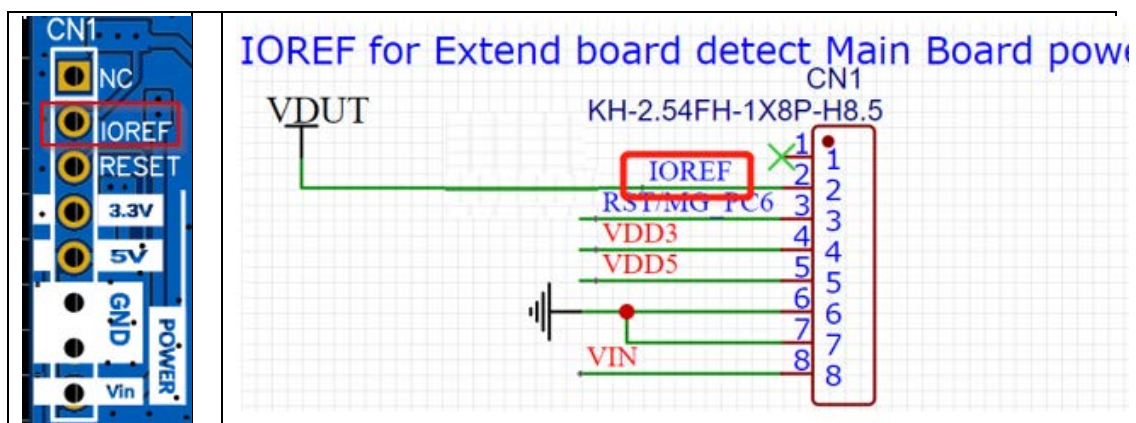
## 19.8 LED6/7/8 表示 MLink 端 USB1 与 PC 连接状态

USB1 正常连接状态 GREEN LED7/ORANGE LED8 均为亮，如果通信不良 RED LED6 会亮



### 19.9 IOREF

该引脚连接 Arduino 主控制器 MG32F02XXX 的工作参考电压。主要作用是 Arduino 扩展板读取 IOREF 引脚电压，从而选择合适的工作电压（这是扩展板本身的功能），或者提供 3.3V 或 5V 的电平转换。IOREF 电压同 J1 选择的电压 5V 或者 3.3V。



## **19.10 开发包路径管理**

第三方的开发板对应的开发包，需要放置在指定的路径，以便于 Arduino IDE 启动后能够正常的识别和加载。

常用的 Arduino AVR 开发板，例如 UNO、MEGA、NANO 这类官方 AVR 开发板，在安装 Arduino IDE2.x.x 时，不会自动把官方的开发包下载，需要手动下载安装(Arduino IDE1.8.x 的版本会默认下载安装)。详细参考 **TH244A001\_UserManual**-官方 AVR 开发包安装。

第三方开发的开发板，例如 STM32 Nucleo、Nu\_edu\_Arduino、Megawin TH244A001 等第三方开发板，不能被 IDE 直接识别，需要安装专门的开发包。非官方的开发板，都称为第三方贡献的开发板，需要手动设置与安装开发包。详细参考 **TH244A001\_UserManual**-第三方开发板的开发包安装和更新。

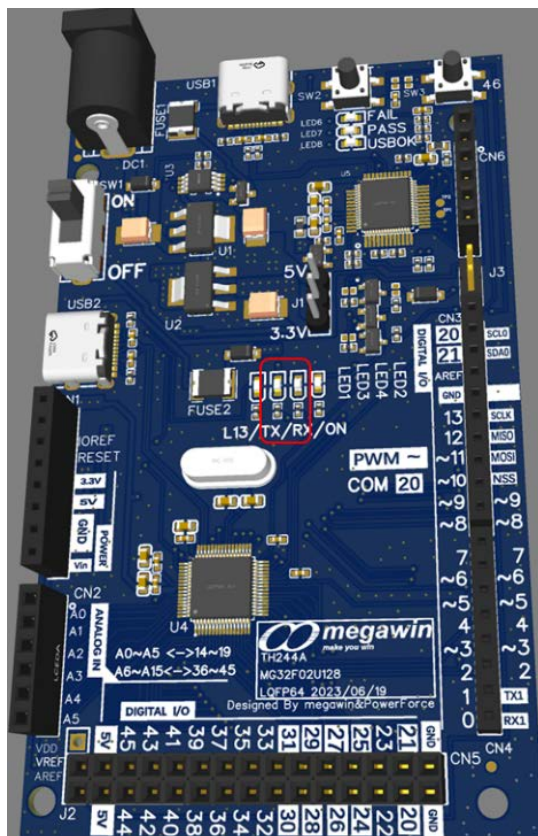
### 19.11 虚拟串口设置和程序下载调试

程序下载是通过 USB1 接口连接着 MG84FG516。设置 MG84FG516 VCP 模式启动虚拟串口(megawin 提供 SHA-1 and SHA-2 驱动)。J3 选择短路，就是使能虚拟串口 VCP。

MG32F02U128 的 UART0 连接着 MG84FG516，用于下载程序，以及与 Arduino IDE 自带的串口调试器通信。Arduino IDE 包含了一个串口监视器，可以通过串口发送或接收指令或者数据，这也是调试的主要手段。当通过 ARDUINO IDE 下载程序时，板子上的 RX 和 TX 两个 LED 会闪烁。

串口监视器的主要功能是使用 Serial.println () 等打印函数，通过 UART0 在 PC 端打印特别定义的信息来实现调试和观察程序运行结果。





烧录过程 TX LED/RX LED 会闪烁

MG84FG516 MCU 透过 SWD 与 MG32F02U128 通信，可以实现实时的调试（Arduino IDE2.X 版本当前仅仅支持官方的部分开发板的实时调试）。TH244A001 作为第三方贡献的开发板，无法直接使用 Arduino IDE2.X 来实现 SWD 的调试，但可以用其他通用开发工具来实现 SWD 调试。

### 20. 开发板重启

1. 透过 Arduino IDE 上传程序时，开发板会重启；
2. 按下按钮 SW2 (RESET) ，开发板会重启，参考 RESET
3. 开发板重新上电，开发板会重启；
4. 通过将 CN1 RESET 脚送低电位，开发板会重启，原理同 2；

## 21. 常用函数以及基本用法

### 21.1 GPIO 读写控制

通用数字端口的基本读写操作

主要常用函数，可参考 Arduino 官方说明

**函数：** `pinMode(pin, MODE)`

**功能：** 设定 pin 的 MODE 为 OUTPUT（输出模式），INPUT（输入模式），  
INPUT\_PULLUP（使能内部上拉电阻的输入模式）三种模式

**参考：** <https://www.arduino.cc/reference/en/language/functions/digital-io/pinmode/>

**函数：** `digitalWrite(pin, value)`

**功能：** 可对引脚 pin 写入电势位,; value 为 HIGH（电压为 VDD）或 LOW(电压为 0)

**参考：** <https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/>

**函数：** `digitalRead(pin)`

**功能：** 可从引脚 pin 中获取电势位；所得数据为 HIGH 或 LOW

**参考：** <https://www.arduino.cc/reference/en/language/functions/digital-io/digitalread/>

### 21.2 延时函数

**函数：** `delay(ms)`

**功能：** 特定延时，参数为毫秒数值

delay(ms)有一个微秒版本称为，

**函数：** delayMicroseconds(us)

**功能：** 特定延时，参数为微秒数值

delay(), delayMicroseconds(us)的缺点是，会占据 MCU 资源，阻塞其他语句执行。

### 举例 1：通过延时，实现 LED 闪烁控制

//通过延时，实现 LED 闪烁控制

```
void setup() {

 //初始化 13 脚为输出 ， 13 宏定义为 LED_BUILTIN

 pinMode(LED_BUILTIN, OUTPUT);

}

void loop() {

 digitalWrite(LED_BUILTIN, HIGH); //输出高，点亮 LED

 delay(1000); //输出高持续 1s

 digitalWrite(LED_BUILTIN, LOW); //输出低，LED 灭

 delay(1000); //输出低持续 1s

}
```



## 21.3 Timer 的运用

millis()或者 micros()数据都很大, 使用时, 定义变量类型为 unsigned long。一般情况 ms 级别的运用就可以满足需求。

**函数:** millis()

**功能:** 返回自 Arduino 板开始运行当前程序以来经过的毫秒数。非阻塞编程必须。这是替代 delay()的最好方式。

**函数:** micros()

**功能:** 返回自 Arduino 板开始运行当前程序以来经过的微妙数。如果您需要更好的分辨率, micros()则可以采用。

## 21.4 PWM

**函数:** analogWrite( pin, Value )

**功能:** 设置指定的管 PWM 的占空比。(本开发板 3、5、6、8、9、10、11 支持 PWM 功能), 默认为 8bit, 即 Value 为 0~255, 对应 duty 为 0%~100%;

**举例:** analogWrite(3,25)    3 号脚位输出占空比约  $25/255=10\%$

analogWrite(10,230)    10 号脚位输出占空比约  $230/255=90\%$

**参考:**

<https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/>

## TH244A001 UserGuide

---

**函数：** analogWriteFrequency(PWM\_FRQ\_XX, Fre)

**功能：** 修改指定端口的 PWM 频率。Fre 可设置范围为 300Hz~5000Hz，默认为 1KHz。

超出这个范围的数值，都会约束为 300Hz（小于 300Hz）或者 5000Hz(高于 5000Hz)。

### 使用说明：

本开发板 3/6、5/8/9、10/11 这些支持 PWM 功能的脚位才能使用此函数设定频率。

3/6（TM20）、5/8/9(TM36)和 10/11(TM26)三个可以独立设置频率。默认频率均为 1000Hz;

举例：

### 方式一

- analogWriteFrequency( PWM\_FRQ\_D3\_D6, Frq )      设置 3/6 输出频率
- analogWriteFrequency(PWM\_FRQ\_D5\_D8\_D9, Frq )    设置 5/8/9 输出频率
- analogWriteFrequency( PWM\_FRQ\_D10\_D11, Frq )    设置 10/11 输出频率
- analogWriteFrequency( PWM\_FRQ\_ALL, Frq )        设置 3/6, 5/8/9, 10/11 输出频率

**举例：** 设置 3 脚输出 PWM 为 3KHz,duty 90%;6 脚输出 PWM 为 3KHz,duty 10%

```
void setup() {
```

```
 analogWriteFrequency(PWM_FRQ_D3_D6, 3000); //设置 PWM_FRQ_D3_D6 输出
 频率 3000Hz
}

void loop() {

 analogWrite(3, 230); //设置 3 输出 duty 为 230/255=90%

 analogWrite(6, 25); //设置 3 输出 duty 为 25/255=10%

}
```

### 方式二

- `analogWriteFrequency(3, Fre )`      设置 3 输出频率,同时 6 也被设置为此频率
- `analogWriteFrequency(5, Frq )`      设置 5 输出频率, 同时 8, 9 也被设置为此频率
- `analogWriteFrequency(10, Frq )`      设置 10 输出频率, 同时 11 也被设置为此频率
- `analogWriteFrequency( PWM_FRQ_ALL, Frq )`      设置 3/6, 5/8/9, 10/11 输出频率

**举例：设置 3 脚输出 PWM 为 3KHz,duty 90%;6 脚输出 PWM 为 3KHz,duty 10%**

```
void setup() {

 analogWriteFrequency(3, 3000); //设置 3 输出频率 3000Hz,同时设置 6 的输出频
 率;

}

void loop() {

 analogWrite(3, 230); //设置 3 输出 duty 为 230/255=90%

 analogWrite(6, 25); //设置 3 输出 duty 为 25/255=10%
```

```
}
```

### 21.5 DAC

**函数：** analogWrite(21, Value )

**功能：** 将模拟值写入 21。当使用此函数时，21 脚位被设置为仿真电压输出模式。(本开

发板仅 21 支持 DAC 输出功能)。因为 DAC 是 12bit，所以 Value 有效范围为

0~4095，DAC 实际输出模拟值为 0~VDD

**举例 1：**

```
void setup() {
```

```
}
```

```
void loop() {
```

```
 analogWrite(21,3000); //修改不同数值,模拟输出电压为 3000/4095*VDD
```

```
 //VDD=5V 时 ， 输出电压 3.6V； VDD=3.3V 时，输出电压 2.38V
```

```
}
```

## 21.6 ADC 函数

**函数：** `analogRead( pin )`

**功能：** 返回从指定的模拟引脚读取值。pin 为指定的模拟输入管脚。

`analogRead(A0)` 。开发板 TH244A001 模拟口为 A0~A15

A0~A5 对应 14~19

A6~A15 对应 36~45

**参考：**

<https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/>

**举例 1： 默认分辨率为 12bit ； VDD=5V**

```
void setup() {

 Serial.begin(9600); //初始化串口

}

void loop() {

 int sensorValue = analogRead(A0); //读取 A0 模拟口，返回数值为 ADC
 数值,范围是 0~4095 (12bit 分辨率下)

 float voltage = sensorValue * (5.0 / 4096.0); //VDD=5V， 默认 12bit 分辨率时，计
 算实际电压数值

 Serial.println(voltage); //打印数据，便于 debug

}
```

**函数：** `analogReadResolution(res);`

## TH244A001 UserGuide

---

**功能：**设置 ADC 的采样分辨率；

可选参数为：8，10，12。默认值为 12 位。主要是和 AVR 开发板兼容，用户可以根据自己需要设定其他分辨率，达到更高精度的控制要求。在切换分辨率后，计算实际物理量或者使用 map() 时特别注意映像范围。

8bit 0~255    255 对应 ADC 参考电压，默认为外部 VDD=5v

10bit 0~1023    1023 对应 ADC 参考电压，默认为外部 VDD=5v

12bit 0~4095    4095 对应 ADC 参考电压，默认为外部 VDD=5v

**参考：**

<https://www.arduino.cc/reference/en/language/functions/zero-due-mkr-family/analogreadresolution/>

**举例 2： 修改分辨率从 12bit 改为 10bit, VDD=5V**

```
void setup() {

 analogReadResolution (10); //分辨率从默认 12bit 修改为 10bit

 Serial.begin(9600); //初始化串口

}

void loop() {

 int sensorValue = analogRead(A0); //读取 A0 模拟口，返回数值为 ADC
 数值范围是 0~1023 (10bit 分辨率下)
```

```
float voltage = sensorValue * (5.0 /1024.0); //VDD=5V, 10bit 分辨率时, 计算实际电压数值
```

```
Serial.println(voltage); //打印数据, 便于 debug
}
```

**函数:** `analogReference( res );` 默认为外部参考 VREF+, 参数为 `AR_DEFAULT` (`AR_EXTERNAL`)

**功能:** 设置 ADC 的采样参考电压源;

可选参数为: `AR_INTERNAL`, `AR_EXTERNAL`, `AR_DEFAULT` (`AR_EXTERNAL`)

注意: 当使用 DAC 去驱动负载, 需要专门设计放大线路增强驱动能力;

**说明:** 推荐使用默认外部电压源 VREF+为参考电压, 不做修改

**函数:** `map(value, fromLow, fromHigh, toLow, toHigh)`

**功能:** `map()`常常结合 ADC 使用, 用于线性映射运算

**参考:**

<https://www.arduino.cc/reference/en/language/functions/math/map/>

**举例 3:** 使用 `map()` 将 0~5000 范围的数据 1000 按照线性关系转换到范围 0~255,

得到新数据 51

//将 0~5000 范围的数据 1000 按照线性关系转换到范围 0~255 的新数据 51

```
int fromLow = 0;
```

```
int fromHigh = 5000;
```

```
int toLow = 0;
```

```
int toHigh = 255;

int value = 1000;

void setup() {

 Serial.begin(9600); //初始化串口

}

void loop() {

 int x = map(value, fromLow, fromHigh, toLow, toHigh); //采用线性运算: x =
a/(fromHigh-fromLow)*(toHigh-fromLow)

 Serial.println(value); //打印原始范围 0~5000 的原始数据

 value = 1000;

 Serial.println(x); //打印新范围 0~255 的线性转换得到
的数据 x= 1000/(5000-0)*(255-0)=51

 delay(1000); //延时 1s, 便于使用者观察输出

}
```

## 21.7 UART

UART0(开发板下载程序专用)/UART1/UART2/UART4/UART5/UART6/UART7

串口通信主要使用 **Serial 库**，主要使用的函数如下：

**函数：** Serial.begin(baud)

**功能：** 初始化串口，设置串口传输波特率 9600,19200, 115200 等，默认 9600bps。



**说明：** 串口工具中要设置一致的波特率，才能够和 MCU 的串口正常通信；

### 举例 1：初始化串口

```
void setup() {

 Serial.begin(9600); //打开串口通讯，设置传输速率为 9600bps

}

void loop() {}
```

**函数：** Serial.println() 带回车功能； Serial.print() 不带回车功能

**功能：** 串口打印信息，用于调试使用输出特定信息，也可以打印用户设计的输出内容

**函数：** Serial.available()

**功能：** 返回串行缓存区接收到的数据的字节数

**函数：** Serial.read()

**功能：** 读取串口的输入，每一次只能读取一个字节。若没有输入则返回-1。一般结合

Serial.available()使用，用于循环读取缓存区字节，直到读取完毕。

### 举例 2：当串口接收到数据，就打印出来

//当串口接收到数据，就打印出来；

```
void setup() {

 Serial.begin(115200); //设置串口速度为 115200

}

void loop() {
```

```
while (Serial.available()) { // 当串口接收到信息后, Serial.available()返回 1,
 可用于条件判断

 char serialData = Serial.read(); // 将接收到的信息使用 read 读取 每次只能读一
 个字节

 Serial.println((char)serialData); // 然后通过串口监视器输出 read 函数读取的信息
}
}
```

### 举例 3：UART0 收到数据，发送给 UART1 打印；UART1 收到数据，发送给 UART0 打印

```
//使用开发板 TH244A001 UART0 和 UART1

//Serial.begin()初始化串口 0，连接到计算机端;

//Serial1.begin()初始化串口 1，通过串口工具连接到计算机，或者连到其他开发版的串口

//通过 Serial.available()判断，UART0 的接收缓存是否有资料;

//通过 Serial1.available()判断，UART1 的接收缓存是否有资料;

//当接收缓存有数据时，执行读取操作，并打印

void setup() {

 Serial.begin(9600); //打开串口通讯，设置传输速率为 9600bps

 Serial1.begin(9600); //打开串口 1 通讯，设置传输速率为 9600bps

}

void loop() {

 // 从串口 0 的数据缓存读取数据； 并按字节发送到串口 1 的数据缓存区;
```

```
if (Serial.available()) {

 int inByte = Serial.read();

 Serial1.println(inByte, DEC);

}

// 从串口 1 的数据缓存读取数据; 并按字节发送到串口 0 的数据缓存区

if (Serial1.available()) {

 int inByte = Serial1.read();

 Serial.println(inByte, DEC);

}

}
```

**函数：** Serial.write()

**功能：** 通过串口输出一个字节，实际传递 ASCII 码

### 举例 4：对比说明 Serial.println()和 Serial.write()差异

//对比说明 Serial.println()和 Serial.write()差异

```
int num = 65; //定义一个十进制的数字

void setup() {

 Serial.begin(115200);

 Serial.println(num); //打印字符串 65

 delay(500);

 Serial.write(num); //打印 ASCII 码为 65 的字符 A

}
```

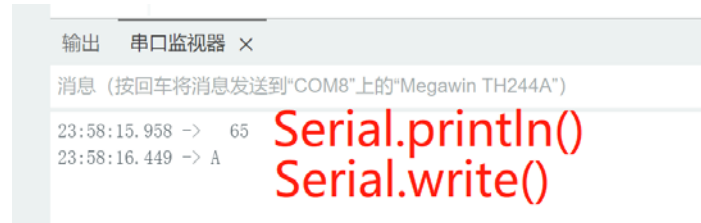
## TH244A001 UserGuide

```
Serial.println(); //输出回车符

}

void loop(){

}
```



说明：

- Serial 表示 UART0
- Serialx 表示 UARTx X : 1, 2, 4, 5, 6, 7

参考 Arduino 官方更多关于串口的说明：

[Serial - Arduino Reference](https://www.arduino.cc/reference/en/language/functions/communication/serial/)

<https://www.arduino.cc/reference/en/language/functions/communication/serial/>

## 21.8 IIC

一般开发板作为主机，模块作为从机使用。

当有其他开发板或者设备作为主机时，TH244A001 可以作为从机的形式加入到总线中。

Wire 表示 IIC0。 Wire1 表示 IIC1 ，默认 IIC clock 为 100KHz。

**函数：**Wire.begin() 和 Wire.begin(address)

**功能：**初始化 Wire 库 ，并且加入到 IIC 总线。Wire.begin() 初始化为主机；

Wire.begin(address) 初始化为从机，只能调用一次，一般在 setup()中完成初始化；

**参数：**

address : 7 位的器件地址（可选），如果为空，则以主机的形式加入到 IIC 总线；如果有设置地址，则是以从机的形式加入到 IIC 总线

**函数：** Wire.onReceive(receiveEvent) ， 此处 Wire 是从机

**功能：** 从机响应主机发来的数据。注册一个函数 receiveEvent，当从机接收到来自主机的传输时，该函数被调用。函数 receiveEvent () 应该使用一个 int 参数（存储从机读取到字节数），并且这个函数不返回任何内容

**函数：** Wire.onRequest(requestEvent) ， 此处 Wire 是从机

**功能：** 主机请求从机发送数据

当主机请求从机发送数据时，**从机**通过 onRequest 设置调用的函数。

注册一个函数 requestEvent，当从机接收到来自主机数据请求时，该函数被调用。函数 requestEvent ()

**函数：** Wire.requestFrom(address,quantity) 主机请求从机数据

Wire.requestFrom(address,quantity,stop) 主机请求从机数据，带停止位

**功能：** 主机向请求从机数据，数据可以被主机用 Wire.read()或 Wire.available()接受。

**参数：**

address : 7 位的器件地址

quantity : 请求数据的数量（字节数）

stop : 布尔型数据，1 则在请求结束后发送一个停止命令，并释放总线；0 则继续发送请求保持连接。

返回值：

函数返回从从机接受到的字节数目；

**函数：** Wire.beginTransmission(address)

**功能：** 主机往发送缓存传输一个开始字符，并指定接受数据的从机地址

**参数：**

address : 从机的 7 位地址

返回：无返回值

**函数：** Wire.endTransmission()和 Wire.endTransmission(stop)

**功能：** 结束由 Wire.beginTransmissio()开始,由 Wire.write()排列的从机传输过程。在 Arduino 中 Wire.endTransmission()可以接受到一个布尔形变量 stop。如果为 1 则 endTransmission () 发送一个停止信息；如果为 0 则发送开始信息

**返回**

- 0 成功
- 1 数据溢出
- 2 发送 address 时从机接受到 NACK
- 3 发送数据时接受到 NACK
- 4 其他错误

**函数：** Wire.write(value)

Wire.write(string)

Wire.write(data, length)

**功能：** 向从机发送数据，可以是字符串，数字，或者指定数据指定长度部分

**参数：**

value : 要发送的数值

string : 字符组的指针

data : 一个字节数组

length : 传输的数量

**函数：** Wire.available()

**功能：** 主机或者从机获取到资料后，可以使用 available 接收当前缓存字节数

**函数：** Wire.read()

**功能：** 主机或者从机获取到资料后，可以使用 Wire.read()接收，一般结合 Wire.available()

使用；

### 举例 1： 主机发送数据，从机接受数据并通过从机打印出来

//使用 TH244A001 测试 IIC0 IIC1 的通信

//TH244A001 IIC0 SCL->20;SDA ->21;IIC1 SCL ->23;SDA ->22

//将开发板 20 接 23； 21 接 22； SDA 上加一个 4.7K 奥姆的电阻上拉到 VDD(5V 或者 3.3V)

//设置 TH244A001 IIC0 为主机， IIC1 为从机； 主机向从机请求特定长度的数据；

```
#include <Wire.h>
```

```
void setup() {
```

```
 Wire.begin(); // 将 IIC0 初始化为主机
```

## TH244A001 UserGuide

---

Wire1.begin(100); // 将 IIC1 初始化为地址是 100 的从机； 实验地址必须是 7bit 地址即使 0~127 范围以内。

Wire1.onReceive(receiveEvent); // 注册接收事件，该函数 receiveEvent () 为处理的事宜，需要定义；

Serial.begin(9600); // start serial for output

}

byte x = 0; //范围 0~255

void loop() {

Wire.beginTransmission(100); // 准备传输数据到地址为 100 的从机；

Wire.write("x is "); // 发送 5 个字节 字母 x,空格, 字母 i,字母 s,空格; 字符型

Wire.write(x); // 发送一个字节,字符型

Wire.endTransmission(); // 执行传输，总计发送 6 个字节

x++;

delay(1000); //间隔 1s 发送一次数据，每次发送 6 个字节

}

//定义 receiveEvent()函数体，IIC1 接受到主机发送的信息后，打印出来。

void receiveEvent(int howMany) // int howmany 是必须定义的，此处 howmany 实

际是统计了实际收受的字节数

{

Serial.println("howMany");

Serial.println(howMany); //打印从机实际接受的字节数 此处为 6



```
while (1 < Wire1.available()) // Wire1.available()检索接收的字节数，每执行一次
Wire.read(),该字节数就减 1

//此处设置为 Wire1.available()>1 ,是因为要把最后一个字
节作为整数打印;

//否者字符型的数字，在通过 Serial.println()打印时，会被
识别为 ASCII 码对于的字母打印，而不是数字;

{

char c = Wire1.read(); // 读取接收数据的一个字节

Serial.print(c); // 打印接收的一个字节

}

int x = Wire1.read(); // 将接受的最后一个字节，转为整数型打印

Serial.println(x); // 打印整数

}
```

### 举例 2：主机向从机请求特定数据，主机接收数据，从机发送数据

```
//使用 TH244A001 测试 IIC0 IIC1 的通信

//TH244A001 IIC0 SCL->20;SDA ->21;IIC1 SCL ->23;SDA ->22

//将开发板 20 接 23; 21 接 22; SDA 上加一个 4.7K 奥姆的电阻上拉到 VDD(5V 或者
3.3V)

//设置 TH244A001 IIC0 为主机，IIC1 为从机；主机向从机请求特定长度的数据；

#include <Wire.h>

void setup() {
```

## TH244A001 UserGuide

---

```
Wire.begin(); // 初始化 IIC0 为主设备

Wire1.begin(2); // 初始化 IIC2 为从设备，地址 2

Wire1.onRequest(requestEvent); // 注册主设备请求从设备数据的事件

Serial.begin(9600); // 初始化串口，用于打印信息

}

void loop() {

 Wire.requestFrom(2, 8); // 结合 setup () 中主设备请求数据事件使用，向从设备 2

 //请求 8 个字节数据

 while (Wire.available()) //少于请求的数据量，会出问题。

 // Wire.available()返回接收缓存数据的字节数，并//随 Wire.read()读取递减，直到读取

 完成为 0。此处//用 Wire.available()作为条件判断是否读取完毕。

 {

 char c = Wire.read(); // 读取一个字节，并转为字符型

 Serial.print(c); // 打印读取的字符

 }

 Serial.println();

 delay(500);

}

//主设备请求数据事件的函数体。从机准备特定数量的数据，供主设备获取

void requestEvent() //

{
```

```
Wire1.write("hello,TH244A001 "); //从设备发送主设备请求的数据
}
```

## 21.9 SPI 主模式

开发板只支持 SPI 设置为主模式。常见从机是 SPI flash, SPI 接口的传感器或者 SPI 显示屏。

**函数:** SPI.begin()

**功能:** 开启并初始化 SPI 总线,。预设 SCK, MOSI 和 SS 为输出, SCK, MOSI 拉低; SS 拉高; 无返回数据;

**函数:** SPI.end()

**功能:** 停止使用 SPI 总线,但是初始化时预设的引脚状态会保留, 以便随时可以再次启动总线。此时 SPI 总线上所有设备都不可再使用总线。无返回数据;

**函数:** SPI.beginTransaction (SPISettings(6000000, MSBFIRST, SPI\_MODE0))

**功能:** 配置传输速度, 数据顺序和模式, 无返回数据;

定义一个 **SPISettings** 对象, SPISettings 对象是用于为 SPI 从机设备配置 SPI 端口的传输参数。单个 SPISettings 对象由 speedMaximum, dataOrder, dataMode 三个参数组成。

**举例:**

```
SPISettings SlaveA(6000000, MSBFIRST, SPI_MODE0) //从机 A 的传输设定;
```

```
SPI.beginTransaction (SlaveA) //将 SPI 配置为从机 A 要求的传输设定;
```

**函数:** SPI.endTransaction ()

**功能:** 停止使用 SPI 总线，特指停止某个从设备不再使用总线，一般禁用（就是把某个从设备的选择脚位 CS 拉高）某个从设备后使用该函数。此操作不影响其他从设备和主机建立通信。无返回数据；

**函数:** SPI.transfer (val) ; SPI.transfer (val16) ; SPI.transfer (buffer, size)

**功能:** 主机向从机发送数据，同时接受来自从机数据。该函数返回数据。

SPI 传输是同时发送和接收的：接收到的数据以 receivedVal（或 receivedVal16）形式返回。在缓冲区传输的情况下，接收到的数据就地存储在缓冲区。

**参数:**

val: 通过总线发送一个字节变量

val16:通过总线发送的两个字节变量

buffer: 要传输的数据数组

size:传输的数据长度

**函数:** SPI.setClockDivider(SPI\_CLOCK\_DIV8)

**功能:** 设置 SPI 总线时钟，无返回数据；SPI\_CLOCK\_DIV8 是指 SPI 时钟设置为 CPU 时钟 8 分频；一般采用默认时钟，不需要修改。

## 21.10 USB

开发板 USB2 作为 device 连接到计算机，可以作为键盘或者鼠标使用。

当程序设计和运用中启动 USB 模块为 Mouse 或者 Keyboard 时，Arduino 开发板将作为键盘或者鼠标输入链接到计算机。

## 21.11 USB-mouse

此处指开发板 USB2，可以初始化为鼠标使用。可以用键盘来模拟鼠标上下左右键，也可以用游戏杆来实现鼠标移动功能。

需要 `#include <Mouse.h>`

**函数：** `Mouse.begin()`

**功能：** 启动 USB 接口为鼠标

**函数：** `Mouse.end()`

**功能：** 关闭鼠标

**函数：** `Mouse.click();Mouse.click(button)`

**功能：** 鼠标点击事件且弹起,相当于瞬间的鼠标点击事件，比如平时使用鼠标时的左击选中，或者右击菜单。

**说明：** Button 有三种状态,分别代表鼠标左键，右键，中键：

- `Mouse.click(MOUSE_LEFT)` -(default) 点击左键
- `Mouse.click(MOUSE_RIGHT)` 点击右键
- `Mouse.click(MOUSE_MIDDLE)` 点击中键

## TH244A001 UserGuide

---

**函数：** Mouse.press(); Mouse.press(button)

**功能：** 长按鼠标按键（不弹起）。比如常用长按左键选择、拖拽的时候，通常配合 Mouse.move () 函数一起使用。需要释放的时候，使用 Mouse.release()

**说明：** Button 有三种状态,分别代表鼠标左键，右键，中键；

- Mouse.press(MOUSE\_LEFT) -default 按下左键
- Mouse.press(MOUSE\_RIGHT) 按下右键
- Mouse.press(MOUSE\_MIDDLE) 按下中键
- Mouse.press(MOUSE\_LEFT | MOUSE\_RIGHT) 同时按下左右键

**函数：** Mouse.release(),Mouse.release(button)

**功能：** 鼠标按键释放。如果鼠标有某个按键被长时间按下，使用这个函数可以释放该按键。

结合 Mouse.press()使用。

**说明：** Button 有三种状态,分别代表鼠标左键，右键，中键；

- MOUSE\_LEFT (default)
- MOUSE\_RIGHT
- MOUSE\_MIDDLE

**函数：** Mouse.isPressed(), Mouse.isPressed(button)

**功能：** 检测鼠标按键当前状态。如果某个鼠标按键被按下或者 Mouse.press () 发送了长按命令，则返回 true。

**说明：** Button 有三种状态,分别代表鼠标左键，右键，中键；

- MOUSE\_LEFT (default)
- MOUSE\_RIGHT
- MOUSE\_MIDDLE

**函数：**Mouse.move(x, y, wheel)

**功能：**把光标移动到指定的位置，这里是相对位置。屏幕上的运动总是相对于游目标当前位置。在使用 Mouse.move() 之前，必须先调用 Mouse.begin()，一般在 setup()中调用。

参数说明：

xVal: 沿 x 轴移动的量。允许的数据类型：signed char.

yVal: 沿 y 轴移动的量。允许的数据类型：signed char.

wheel: 移动滚轮的量。允许的数据类型：signed char.

因为三个参数类型均是 signed char，所以范围是 x (-127~127)，y (-127~127) 或者 wheel (-127~127)

**举例 1： 使用 Mouse.h 库实现游戏杆功能，带一个开关 (46)。通过游戏杆 X 轴和 Y 轴移动，来控制鼠标光标移动。并带一个按键功能（默认鼠标左键）。**

//使用官方的 文件->示例->USB->Mouse->JoystickMouseControl 修改

//游戏杆的 X/Y 轴坐标说句，采用 ADC 读取

//游戏杆带的一个按键仿真鼠标左键

//13 脚连接的 LED，用来表示鼠目标开关状态

//46 连接的 USER KEY 用来开关 USB 鼠标功能

//游戏杆的 X/Y 轴输出分别连接 A0,A1;游戏杆的击键连接到 0 脚;

#include "Mouse.h"

//设置游戏杆坐标轴， 开关，LED 的脚位

const int switchPin = 46; // USER KEY 用来开关鼠标功能 和官方差异

const int mouseButton = 0; // 游戏杆的击键连接到 0 脚，本程序，将按键设置为左键

-MOUSE\_LEFT 和官方差异

## TH244A001 UserGuide

---

```
const int xAxis = A0; // 游戏杆的 X 轴模拟输入

const int yAxis = A1; // 游戏杆的 Y 轴模拟输入

const int ledPin = 13; //鼠标控制提示 LED，本程序此 LED 为开发板上 13 脚对应的
 LED1,打开鼠标功能，此 LED 亮；关闭鼠标功能，此 LED 灭； 和官方差异

//定义读取游戏杆数据的变量

int range = 21; // X, Y 轴，总计移动的范围,这里将游戏杆 X 或者 Y 轴，总移
 动量定义为 21 级； 和官方差异

int responseDelay = 5; // 相应鼠标移动延迟时间，此数值可以调节敏感度；越小，
 移动越敏感；

int threshold = range / 7; // 静默门限 在正负 threshold 之间，不做任何反应，当成
 没有鼠标移动动作发生；

int center = range / 2; // 中间位置

bool mouselsActive = false; // 鼠标状态

int lastSwitchState = LOW; // 开关的默认状态

void setup() {

 pinMode(switchPin, INPUT); // 初始化开关引脚为输入

 pinMode(ledPin, OUTPUT); // 初始化鼠标状态 LED 控制引脚为输出

 pinMode(mouseButton, INPUT); //

 Mouse.begin(); //启动鼠标功能

}

void loop() {
```



```
//读取开关状态，如果有变化，且最新状态为 HIGH,表示开关有按下一次。状态变化为
HIGH->LOW->HIGH

int switchState = digitalRead(switchPin);

if (switchState != lastSwitchState) { //检测到开关状态变化

 if (switchState == HIGH) { //判断状态变化完成后，为 HIGH

 mouselsActive = !mouselsActive; //鼠标状态切换

 digitalWrite(ledPin, mouselsActive); //鼠标控制提示 LED 状态切换

 }

}

//保留开关最新状态，以便下一轮对比

lastSwitchState = switchState;

// 使用 ADC 读取游戏杆的 X, Y 轴数据

int xReading = readAxis(A0); //通过函数 readAxis () 处理 X 轴读取的数据；返回鼠
标 X 轴相对移动的位移量；

int yReading = readAxis(A1); //通过函数 readAxis () 处理 Y 轴读取的数据；返回鼠
标 Y 轴相对移动的位移量；

//如果鼠标状态设置为 HIGH，就启 USB 动鼠标移动操作

if (mouselsActive) {

 Mouse.move(xReading, yReading, 0); //0 表示 Wheel 滑轮没有变化 我们当前
游戏杆是没有配滑轮输入的

 //xReading 为正 左移；xReading 为负 右移；

 //yReading 为正 下移；yReading 为负 上移；
```

```
}

// 判断游戏杆按键是否按下（本程序设置为左键效果），如果没有按下左键，就按下左
键;

if (digitalRead(mouseButton) == HIGH) {

 if (!Mouse.isPressed(MOUSE_LEFT)) {

 Mouse.press(MOUSE_LEFT);

 }

}

// else 如果没有按下，就发送按键

else {

 // if the mouse is pressed, release it:

 if (Mouse.isPressed(MOUSE_LEFT)) {

 Mouse.release(MOUSE_LEFT);

 }

}

delay(responseDelay);

}

//定义 ADC 读取数据，并做转换 。计算 X 或者 Y 轴相对移动量;

int readAxis(int thisAxis) {

 int reading = analogRead(thisAxis); //通过 ADC,读取 X 或者 Y 轴数值 ,

 //使用 map()函数将 ADC 读取的数值线性转换到定义范围 从 0->1023 对应到

 0->range
```

```
reading = map(reading, 0, 1023, 0, range);

// if the output reading is outside from the rest position threshold, use it:

int distance = reading - center;

if (abs(distance) < threshold) {

 distance = 0;

}

// return the distance for this axis:

return distance;

}
```

## 21.12 USB-Keyboard

**函数：** Keyboard.begin()

**功能：** 开启键盘。这个函数启动一个虚拟的键盘，并连接到计算机上，如果需要终止这个 USB 连接，使用 Keyboard.end()

**函数：** Keyboard.end()

**功能：** 关闭键盘

**函数：** Keyboard.press()

**功能：** 仿真长按键盘，用于持续向计算机输入某个被长按的按键。释放某个按键，就使用 Keyboard.release() 或者 Keyboard.releaseAll()。比如长按向上键。

**函数：** Keyboard.release()

**功能：** 释放特定按键，和函数 Keyboard.press()对应，

**函数：** Keyboard.releaseAll()

**功能：**释放全部按键

**函数：**Keyboard.print()

**功能：**敲击键盘，模拟敲击事件，比如模拟文本输入，可以使用这个指令。

**函数：**Keyboard.println()

**功能：**和上面函数 Keyboard.print () 类似，带回车的敲击事件

**函数：**Keyboard.write()

**功能：**发送单个键盘敲击指令。和 Keyboard.print()、Keyboard.println()的作用相似，只是 Keyboard.write()只发送一个键位的敲击指令。Keyboard.println()、Keyboard.print()或者 Keyboard.write()指令，都只能发送 ASCII 可打印字符，如字符的 ABC，数字的 123，或者空格、回车等。

### 举例 1：启动键盘，打印数字和字符

```
//启动键盘，使用基本的 write(),print()和 println()

//write()只能打印一个字节，如果是数字，是对应 ASCII 对于字符

//print(),println()可以正常打印数字，字母，字符串等等

#include "Keyboard.h"

void setup() {

 Keyboard.begin(); //启动 USB 键盘

}

void loop() {

 Keyboard.write(97); //打印 ASCII 97 对应字符 a

 Keyboard.print("\n"); //回车换行

 Keyboard.write('a'); //打印字符 a
```

```
Keyboard.print('\n'); //回车换行

Keyboard.print(97); //打印 97

Keyboard.print('\n'); //回车换行

Keyboard.print('a'); //打印 字符 a

Keyboard.print('\n'); //回车换行

Keyboard.println(97); //打印 97

Keyboard.print('\n'); //回车换行

delay(5000); //延时 5s

}
```

## TH244A001 UserGuide

---

### 举例 2:

```
//检测 USER KEY 46 是否有按下，如果有按下，就通过 USB2 键盘功能一直打印

Hello,TH244A001!

//提示，如果计算机本身键盘 Caps 键设置为大写模式，Arudino 仿真的 USB 键盘输出的
大小写就会相反;

//提示提前将输入模式默认为英文模式

//比如 Caps 为小写模式本程序行印 Hello,TH244A001!

//比如 Caps 为大写模式本程序行印 hELLO,tH244A001!

#include <Keyboard.h>

int pinpin = 46;

void setup() {

 //因为 46 已经有硬件上拉电阻，所以初始化使用 INPUT，而不需要用 INPUT_PULLUP;

 pinMode(pinpin, INPUT); //如果连接没有外部电阻上拉的 GPIO，需要使用

 pinMode(pinpin, INPUT_PULLUP);

 Keyboard.begin();

}

void loop() {

 //if the button is pressed

 if (digitalRead(pinpin) == LOW) {

 //Send the message

 Keyboard.print("Hello,TH244A001!");

 }

}
```

```
}
```

**举例 3：通过检测 USER KEY 按下，启动键盘输出。输出内容为统计按键的次数；运用到**

**Keyboard.begin(); Keyboard.print(), Keyboard.print();**

//通过检测 USER KEY 按下，启动键盘输出按键的次数；

```
#include "Keyboard.h"
```

```
const int buttonPin = 46; // 按键连接引脚 TH244A001 开发板 自带的 46 对
应按键 USER KEY.
```

```
int previousButtonState = HIGH; // 按键状态 TH244A001 USER KEY 已经设计
上拉电阻，按下会从高变低
```

```
int counter = 0; // 按键计数器 用来记录我们按键的次数
```

```
void setup() {
```

```
 // 初始化按键引脚，如果没有上拉电阻，需要使用 INPUT_PULLUP
```

```
 pinMode(buttonPin, INPUT);
```

```
 // 初始化模拟键盘功能
```

```
 Keyboard.begin();
```

```
}
```

```
void loop() {
```

```
 // 读按键状态
```

```
 int buttonState = digitalRead(buttonPin);
```

```
 // 如果按键状态改变，且当前按键状态为高电平。完成按下，松开这个过程，46 脚位应
 该是 HIGH->LOW->HIGH 变化;
```

```
 if ((buttonState != previousButtonState) && (buttonState == HIGH)) {
```

```
// 按键计数器加 1

counter++;

// 模拟键盘输出信息：打印按键次数；

Keyboard.print("You pressed the button ");

Keyboard.print(counter);

Keyboard.println(" times.\n"); //输出完成后，回车换行

}

// 保存当前按键状态，用于下一次比较

previousButtonState = buttonState;

}
```



## 21.13 RTC

初始化 RTC:/\*这部分初始化一般放在 setup 中\*/

**函数:** Rtc.begin();

**功能:** 启动 RTC

**函数:** Rtc.setDate(day, month, year);

**功能:** 设置日期

**函数:** Rtc.setTime( hour, minute, sec);

**功能:** 设置时间 默认是 24 小时制

获取时间信息: /\*这部分可用作实际调用需求\*/

**函数:** Rtc.getHour();

**功能:** 获取当前的小时

**函数:** Rtc.getMinute();

**功能:** 获取当前的分钟

**函数:** Rtc.getSecond();

**功能:** 获取当前的秒

**函数:** Rtc.getDay();

**功能:** 获取当前的日期

**函数:** Rtc.getMonth();

**功能:** 获取当前的月份

**函数:** Rtc.getYear();

**功能:** 获取当前的年份

## TH244A001 UserGuide

---

### 举例 1：测试串口打印功能和 RTC 功能

```
// TH244A001 有专门定义一个 class MG32x02z_RTC 用于基础的 RTC 功能;

//本程序使用串口打印时间日期信息;

//测试串口打印功能和 RTC 功能;

#include <Keyboard.h>

void setup() {

 /*这部分初始化一般放在 setup 中*/

 Rtc.begin(); //启动 RTC

 Rtc.setDate(24, 6, 2023); //设置日期 2023-6-24

 Rtc.setTime(12, 48, 30); //设置时间 12: 48: 30

 Serial.begin(9600); //初始化串口, 用于显示时间

}

void loop() {

 //获取时间, 串口 UART0 打印, 增加一些排版, 用 " : " 分割;

 Serial.println("当前时间: ");

 Serial.print(Rtc.getHour());

 Serial.print(":");

 Serial.print(Rtc.getMinute());

 Serial.print(":");

 Serial.print(Rtc.getSecond());

 Serial.println();
```

```
//获取日期，串口 UART0 打印，增加一些排版，用 "-" 分割；

Serial.println("当前日期：");

Serial.print(Rtc.getMonth());

Serial.print("-");

Serial.print(Rtc.getDay());

Serial.print("-");

Serial.print(Rtc.getYear());

Serial.println();

delay(1000);

}
```

RTC 设置后输出结果如下



### 21.14 SLEEP/STOP mode

#### 时间常数

定义了 Arduino 里面通用的一些 LowPower 时间常数,可以在 SLEEP 和 IWDT 中使用。

用。IWDT 计时是基于 ILRCO 32KHz。

```
SLEEP_15MS = WDTO_15MS, //15ms 计时
SLEEP_30MS = WDTO_30MS, //30ms 计时
SLEEP_60MS = WDTO_60MS, //60ms 计时
SLEEP_120MS = WDTO_120MS, //120ms 计时
SLEEP_250MS = WDTO_250MS, //250ms 计时
SLEEP_500MS = WDTO_500MS, //500ms 计时
SLEEP_1S = WDTO_1S, //1s 计时
SLEEP_2S = WDTO_2S, //2s 计时
SLEEP_4S = WDTO_4S, //4s 计时
SLEEP_8S = WDTO_8S, //8s 计时
SLEEP_FOREVER = -1 //一直
```

#### LowPowerClass

定义了 LowPowerClass, 可以进行节能模式的设置。

```
class LowPowerClass
```

```
{
```

```
public:
```

```
void idle(period_t period=SLEEP_FOREVER);

void standby(period_t period=SLEEP_FOREVER);

void longPowerDown(uint32_t sleepTime);

void sleep(uint32_t sleepTime);

void powerDown(period_t period, adc_t adc=ADC_OFF, bod_t bod=BOD_OFF, usb_t
usb=USB_OFF);

void powerStandby(period_t period, adc_t adc=ADC_OFF, usb_t usb=USB_OFF);

};

extern LowPowerClass LowPower;
```

### LowPower 类和方法的基本运用:

```
LowPower.idle(SLEEP_FOREVER); //一直保持 STOP，直到有唤醒事件发生

LowPower.idle(SLEEP_500MS); //保持 STOP 500ms 就会唤醒，此处只支持使用时间常数;

LowPower.standby(SLEEP_FOREVER) //一直保持 SLEEP，直到有唤醒事件发生;

LowPower.standby(SLEEP_500MS); //保持 SLEEP 500ms 就唤醒,此处只支持使用时间常数;

LowPower. longPowerDown(uint32_t sleepTime) //设置 STOP 任意时间，单位 ms

LowPower. Sleep(uint32_t sleepTime) //设置 SLEEP 任意时间，单位 ms

LowPower. powerDown(period_t period,

 adc_t adc=ADC_OFF,

 bod_t bod=BOD_OFF,

 usb_t usb=USB_OFF);
```

//带选择关闭功能的 STOP 设置, period\_t period 为 STOP 状态时间,达到时间自动唤醒。此

处只支持使用时间常数。ADC\_OFF、BOD\_OFF、USB\_OFF 选择关闭项目

LowPower. powerStandby(period\_t period,

adc\_t adc=ADC\_OFF,

usb\_t usb=USB\_OFF);

//带选择关闭功能的 SLEEP 设置, period\_t period 为 SLEEP 状态时间,达到时间自动唤醒。此

处只支持使用时间常数。ADC\_OFF、USB\_OFF 选择关闭项目

## 21.15 IWDG

**IWDG 计时是基于 ILRCO 32KHz。**

SLEEP\_15MS = WDTO\_15MS, //看门狗定时器 15ms 超时

SLEEP\_30MS = WDTO\_30MS, //看门狗定时器 30ms 超时

SLEEP\_60MS = WDTO\_60MS, //看门狗定时器 60ms 超时

SLEEP\_120MS = WDTO\_120MS, //看门狗定时器 120ms 超时

SLEEP\_250MS = WDTO\_250MS, //看门狗定时器 250ms 超时

SLEEP\_500MS = WDTO\_500MS, //看门狗定时器 500ms 超时

SLEEP\_1S = WDTO\_1S, //看门狗定时器 1s 超时

SLEEP\_2S = WDTO\_2S, //看门狗定时器 2s 超时

SLEEP\_4S = WDTO\_4S, //看门狗定时器 4s 超时

SLEEP\_8S = WDTO\_8S, //看门狗定时器 8s 超时

### 结合中断的运用示例 1： 不复位 MCU 系统，而是执行中断服务程序

#### ISR(WDT\_vect)

/\*通过看门狗设置达到 4S，变化一次灯号的效果。因为进入中断后，计时器依旧在继续计时，所以不能在中断服务程序 ISR 中执行太多事情。一般都是执行一次标志位的设置或者状态的翻转,就要进行喂狗动作。

```
*/

#include "IWDT.h" //必须包含此文件

int LEDpin = 13;

volatile int state =HIGH;

ISR(WDT_vect)

{

 state = !state; //中断中执行的事情必须简单

 wdt_reset(); //喂狗

}

void setup() {

 pinMode(LEDpin, OUTPUT);

 wdt_disableRST(); //禁用开门狗复位 MCU 功能

 wdt_enable(WDTO_4S); //启动看门狗,超时设置为 4s

 wdt_reset(); //喂狗

}

void loop() {

 digitalWrite(LEDpin, state);
```

```
}
```

**结合复位的运用示例 2：** 开门狗 timeout 为 2s,因为程序没有设计喂狗，所以达到 2s，就会复位系统。程序每次从 Setup()开始执行，通过串口打印信息可知每 2s，发生一次 RESET。

```
#include "IWDWT.h" //必须包含此文件

void setup() {

 // put your setup code here, to run once:

 Serial.begin(9600);

 Serial.println("WDT timeout ,Reset MCU MG32F02U128.");

 wdt_enableRST(); //使能看门狗中断复位 MCU 功能

 wdt_enable(WDTO_2S); //启动看门狗，超时设置为 2s.

 wdt_reset(); //Watchdog timer reset (feeding dog)

}

void loop() {

 // put your main code here, to run repeatedly:

}
```



输出 串口监视器 ×

消息 (按回车将消息发送到“COM5”上的“Megawin TH244A”)

```
21:42:24.823 -> ? WDT timeout ,Reset MCU.
21:42:26.988 -> ? WDT timeout ,Reset MCU.
21:42:29.180 -> ? WDT timeout ,Reset MCU.
21:42:31.330 -> ? WDT timeout ,Reset MCU.
21:42:33.503 -> ? WDT timeout ,Reset MCU.
```

**结合复位的运用示例 3：** 开门狗 timeout 为 2s,因为程序设计了喂狗，不会发生复位系统。程序可以一致保持 0.5s 亮灭。

```
#include "IWDWT.h" //必须加载此文件
```

```
void setup() {
```

```
 // put your setup code here, to run once:
```

```
 pinMode(LED_BUILTIN, OUTPUT); //LED_BUILTIN 内置 LED, D13 脚位
```

```
 digitalWrite(LED_BUILTIN, LOW);
```

```
 wdt_enableRST(); //使能开门狗中断复位 MCU 功能
```

```
 wdt_enable(WDTO_2S); //启动开门狗,超时设置为 2s
```

```
 wdt_reset(); //喂狗 (看门狗定时器重置)
```

```
}
```

```
void loop() {
```

```
digitalWrite(LED_BUILTIN, HIGH);

delay(500);

digitalWrite(LED_BUILTIN, LOW);

delay(500);

digitalWrite(LED_BUILTIN, HIGH);

delay(500);

digitalWrite(LED_BUILTIN, LOW);

delay(500);

wdt_reset(); //闪灯时间 2s,在看门狗定时器 timeout 2s(实际时间长度略大于 2s)

之前喂狗（重置定时器），避免重启

// put your main code here, to run repeatedly:

}
```

## 21.16 TimerOne

主要运用输出 PWM 功能和中断函数功能。

示例 1：产生一个控制外部设备的 PWM 信号

//通过 TimerOne 产生一个控制外部设备的 PWM 信号；PWM 信号频率为 25KHz。PWM 的//占空比从 30% 按照 1%幅度逐步增到 100%；在返回到 30%重复增加过程。运行过程会把占//空比打印出来。

```
#include <TimerOne.h>
```

```
// This example creates a PWM signal with 25 kHz carrier.
```

```
//
```

```
const int fanPin = 43;
```

```
void setup(void)
```

```
{
```

```
 Timer1.initialize(40); // 40 us = 25 kHz
```

```
 Serial.begin(9600);
```

```
}
```

```
void loop(void)
```

```
{
```

```
 // slowly increase the PWM fan speed
```

```
 //
```

## TH244A001 UserGuide

```
for (float dutyCycle = 30.0; dutyCycle < 100.0; dutyCycle++) {

 Serial.print("PWM Fan, Duty Cycle = ");

 Serial.println(dutyCycle);

 Timer1.pwm(fanPin, (dutyCycle / 100) * 1023);

 Serial.println();

 delay(500);

}

}
```

输出 串口监视器 ×

消息 (按回车将消息发送到“COM22”上的“Megawin”)

```
16:29:05.007 -> PWM Fan, Duty Cycle = 92.00
16:29:05.507 -> PWM Fan, Duty Cycle = 93.00
16:29:06.085 -> PWM Fan, Duty Cycle = 94.00
16:29:06.570 -> PWM Fan, Duty Cycle = 95.00
16:29:07.111 -> PWM Fan, Duty Cycle = 96.00
16:29:07.643 -> PWM Fan, Duty Cycle = 97.00
16:29:08.158 -> PWM Fan, Duty Cycle = 98.00
16:29:08.710 -> PWM Fan, Duty Cycle = 99.00
16:29:09.263 -> PWM Fan, Duty Cycle = 30.00
16:29:09.755 -> PWM Fan, Duty Cycle = 31.00
16:29:10.278 -> PWM Fan, Duty Cycle = 32.00
16:29:10.826 -> PWM Fan, Duty Cycle = 33.00
16:29:11.351 -> PWM Fan, Duty Cycle = 34.00
16:29:11.894 -> PWM Fan, Duty Cycle = 35.00
16:29:12.411 -> PWM Fan, Duty Cycle = 36.00
16:29:12.948 -> PWM Fan, Duty Cycle = 37.00
```

示例二：运用 TimerOne 定时器中断

//运用 TimerOne 定时器中断，实现 0.5s 闪灯，并实时打印闪灯次数。

//程序也使用 volatile 修饰变量，实现中断函数和主程序之间的变量共享。并通过禁用中断

//和重启中断方式灵活控制中断。

```
#include <TimerOne.h>
```

//本例使用定时器中断使 LED 闪烁，还演示了中断函数和主程序之间如何使用 volatile 修饰共享变量

```
const int led = LED_BUILTIN; // the pin with a LED
```

```
void setup(void)
```

```
{
```

```
 pinMode(led, OUTPUT);
```

```
 Timer1.initialize(150000);
```

```
 //Timer1.setPeriod(150000);
```

```
 Timer1.attachInterrupt(blinkLED); // blinkLED to run every 0.15 seconds
```

```
 Serial.begin(9600);
```

```
}
```

```
// The interrupt will blink the LED, and keep
```

```
// track of how many times it has blinked.
```

```
int ledState = LOW;
```

```
volatile unsigned long blinkCount = 0; // use volatile for shared variables
```

```
void blinkLED(void)

{

 if (ledState == LOW) {

 ledState = HIGH;

 blinkCount = blinkCount + 1; // increase when LED turns on

 } else {

 ledState = LOW;

 }

 digitalWrite(led, ledState);

}

// The main program will print the blink count

// to the Arduino Serial Monitor

void loop(void)

{

 unsigned long blinkCopy; // holds a copy of the blinkCount

 // to read a variable which the interrupt code writes, we

 // must temporarily disable interrupts, to be sure it will

 // not change while we are reading. To minimize the time

 // with interrupts off, just quickly make a copy, and then

 // use the copy while allowing the interrupt to keep working.

 //保存 blinkCount 的副本
```

```
//要读取中断代码写入的变量，我们

//必须临时禁用中断，以确保它

//当我们读取时不会被改变。为了减少中断关闭时间，

//只需快速复制，然后在重新允许中断继续工作的同时使用副本。

noInterrupts(); //临时禁用中断

blinkCopy = blinkCount;

interrupts(); //启动中断

Serial.print("blinkCount = ");

Serial.println(blinkCopy);

delay(100);

}
```

### 21.17 MsTimer2

MsTimer2::set(unsigned long ms, void (\*f()));//设置定时中断的时间间隔和调

用的中断服务程序。

//ms 表示的是定时时间的间隔长度，单位是 ms;

//void(\*f())表示被调用中断服务程序函数名

MsTimer2::start();//启动定时器中断

MsTimer2::stop(); //停止定时中断,随时可以用 start 方法重新启动中断

void(\*f){} //定时器中断处理函数

示例：

// 闪烁 三次（亮 0.5s 灭 0.5s）,暂停 3s（暂停中断 3s）.重复。

```
#include <MsTimer2.h>
```

```
const int led_pin = LED_BUILTIN; // LED_BUILTIN = 13;
```

```
volatile int LED_times = 0;
```

```
//定义中断函数，记录闪灯次数，并在中断中打印信息
```

```
void flash() {
```

```
 LED_times++;
```

```
 Serial.println(LED_times); //打印中断触发的次数;
```

```
 static boolean output = HIGH;
```

```
 digitalWrite(led_pin, output);
```

```
 output = !output;
```

```
}
```



```
void setup() {

 pinMode(led_pin, OUTPUT);

 MsTimer2::set(500, flash); // 设置定时器周期 500ms 触发一次中断，中断函数为
flash

 MsTimer2::start(); //启动定时器
}

void loop() {

 if (LED_times == 6) // 中断触发 6 次，暂停定时器中断 3s，再启动定时器中断
 {

 MsTimer2::stop(); //暂停定时器

 delay(3000);

 LED_times = 0;

 MsTimer2::start(); //启动定时器
 }
}
```



## 21.18 EEPROM

EEPROM.begin(Size);                   //申请读写操作的大小,Size 是必须小于 EEPROM\_SIZE

定义的大小的。比如默认 EEPROM\_SIZE=512, 那么此处 Size 就为 1~512 任意数值;

### 单字节数据读写操作:

EEPROM.write(address,data)   //向指定地址写数据, write 方法是以字节为存储单位,

EEPROM.read(address)           //读取指定地址的数据,read()方法是以字节为单位读取

EEPROM.commit();               //使数据从暂存区 SRAM 保存到 flash 中, 实现掉电保护

### 多字节数据读写操作:

当需要写入或者读取多个字节时, 比如浮点数据或者整形数据或者其他数据结构 (需要多字节存储数据), 可以用如下方法来读写:

EEPROM.put(address, data)       //向指定地址写数据, data 为待存储的数据

EEPROM.get(address, data)       //读取指定地址的数据, data 为存储读取到的数

据

### Arduino ARM32 位开发板中常见数据的字节数参考:

| 数据类型        | 字节数 | 范围         | 备注                                               |
|-------------|-----|------------|--------------------------------------------------|
| void        |     |            | 只用作函数声明, 表示没有返回值                                 |
| boolean     | 1   | true,false |                                                  |
| bool        | 1   | true,false |                                                  |
| char        | 1   | -128~127   | Arduino 中的 char 是有符号的, 等价于 signed char。char 常被是用 |
| signed char | 1   | -128~127   |                                                  |

## TH244A001 UserGuide

|               |   |                                  |                                     |
|---------------|---|----------------------------------|-------------------------------------|
|               |   |                                  | 于储存 ASCII 字符。如果想存储数据，建议使用 byte 类型。  |
| unsigned char | 1 | 0~255                            | unsigned char , byte 和 uint8_t 类型等同 |
| byte          | 1 | 0~255                            |                                     |
| uint8_t       | 1 | 0~255                            |                                     |
| short         | 2 | -32768~32767                     |                                     |
| uint16_t      | 2 | 0~65535                          |                                     |
| int           | 4 | -2147483648~2147483647           |                                     |
| unsigned int  | 4 | 0 ~ 4,294,967,295                |                                     |
| word          | 4 | 0 ~ 4,294,967,295                |                                     |
| long          | 4 | -2147483648~2147483647           |                                     |
| unsigned long | 4 | 0 ~ 4,294,967,295                | unsigned long 等同于 uint32_t 类型       |
| uint32_t      | 4 | 0 ~ 4,294,967,295                |                                     |
| float         | 4 | -<br>3.4028235E+38~3.4028235E+38 | 只有 6~7 位小数精度                        |
| double        | 4 | -<br>3.4028235E+38~3.4028235E+38 | 只有 6~7 位小数精度                        |

示例程序：

存储 4 种不同类型的数据，数据占用的储存空间不一样，涉及单字节读写和多字节读写。

存储完后，在打印出来。

- char 数据 -----1 字节
- int 数据 -----4 字节
- float 型数据-----浮点型 4 字节
- 字符串（测试字符数组）-- 25 字节。一般都需要带一个结束标志字节 0。所以实际获取长度为 26.

/\*

Single byte read and write : EEPROM.read(address);        EEPROM.write(address, byte)

Multi bytes read and write :

EEPROM.get(address,data);    EEPROM.put(address,data)

EEPROM.commit(); Submit to save data from the temporary storage area to flash, achieving power-off protection

Here, store char data(1 byte)& int data (4 bytes)& float data(4 bytes) &

string(char array) (25 bytes) data to EEPROM

\*/

//Load Library Files

#include <EEPROM.h>

## TH244A001 UserGuide

---

```
//Define four different types of variables

char charVal = 'a'; // 1 byte of char data to be stored in
EEPROM

int intVar = 999999; // 4 bytes of int data to be stored in
EEPROM

float floatVar = 234.567; // 4 bytes of floating-point data to be
stored in EEPROM

char string[] = "Test EEPROM store string."; // 25 bytes of string to be stored in
EEPROM,include a stop byte 0.

//Define three addresses to store four variables

int charValAddr = 0;

int intVarAddr = 1;

int fVarAddr = 5;

int stringAddr = 9;

void setup() {

 EEPROM.begin(512);

 Serial.begin(9600);

 //sizeof() is used to calculate the type length of the sizeof operator, in bytes

 Serial.println(" ");
```

```
Serial.println("char data : " + String(sizeof(charVal)) + " byte."); //1 byte

Serial.println("int data : " + String(sizeof(intVar)) + " bytes."); //4 bytes

Serial.println("float data : " + String(sizeof(floatVar)) + " bytes."); //4ytes

Serial.println("string data : " + String(sizeof(string)) + " bytes."); //26 bytes of
string to be stored in EEPROM,include a stop byte 0.
```

```
EEPROM.write(charValAddr, charVal); // Store charVal in EEPROM address 0

delay(10);

EEPROM.put(intVarAddr, intVar); // Store intVar in EEPROM address 1

delay(10);

EEPROM.put(fVarAddr, floatVar); // Store floatVar in EEPROM address 5

delay(10);

EEPROM.put(stringAddr, string); // Store string in EEPROM address 9

delay(10);

EEPROM.commit(); //Submit to save data from the temporary storage area to
flash,

Serial.println("Finished writing data!");

Serial.println("Start reading float data:");

charVal = EEPROM.read(charValAddr);
```

## TH244A001 UserGuide

---

```
EEPROM.get(intVarAddr, intVar);

EEPROM.get(fVarAddr, floatVar);

EEPROM.get(stringAddr, string);

Serial.println("charVal is " + String(charVal));

Serial.println("intVar is " + String(intVar));

Serial.println("floatVar is " + String(floatVar, 3)); // three decimal places

Serial.println("string is " + String(string));

Serial.println("End read.");

for (int addr = 0; addr < 512; addr++) {

 int data = EEPROM.read(addr); //read one byte

 Serial.print(data);

 Serial.print(" ");

 delay(2);

 if ((addr + 1) % 256 == 0) //

 {

 Serial.println("");

 }

}

Serial.println("End read");

}
```



```
void loop() {

 // put your main code here, to run repeatedly:

}
```