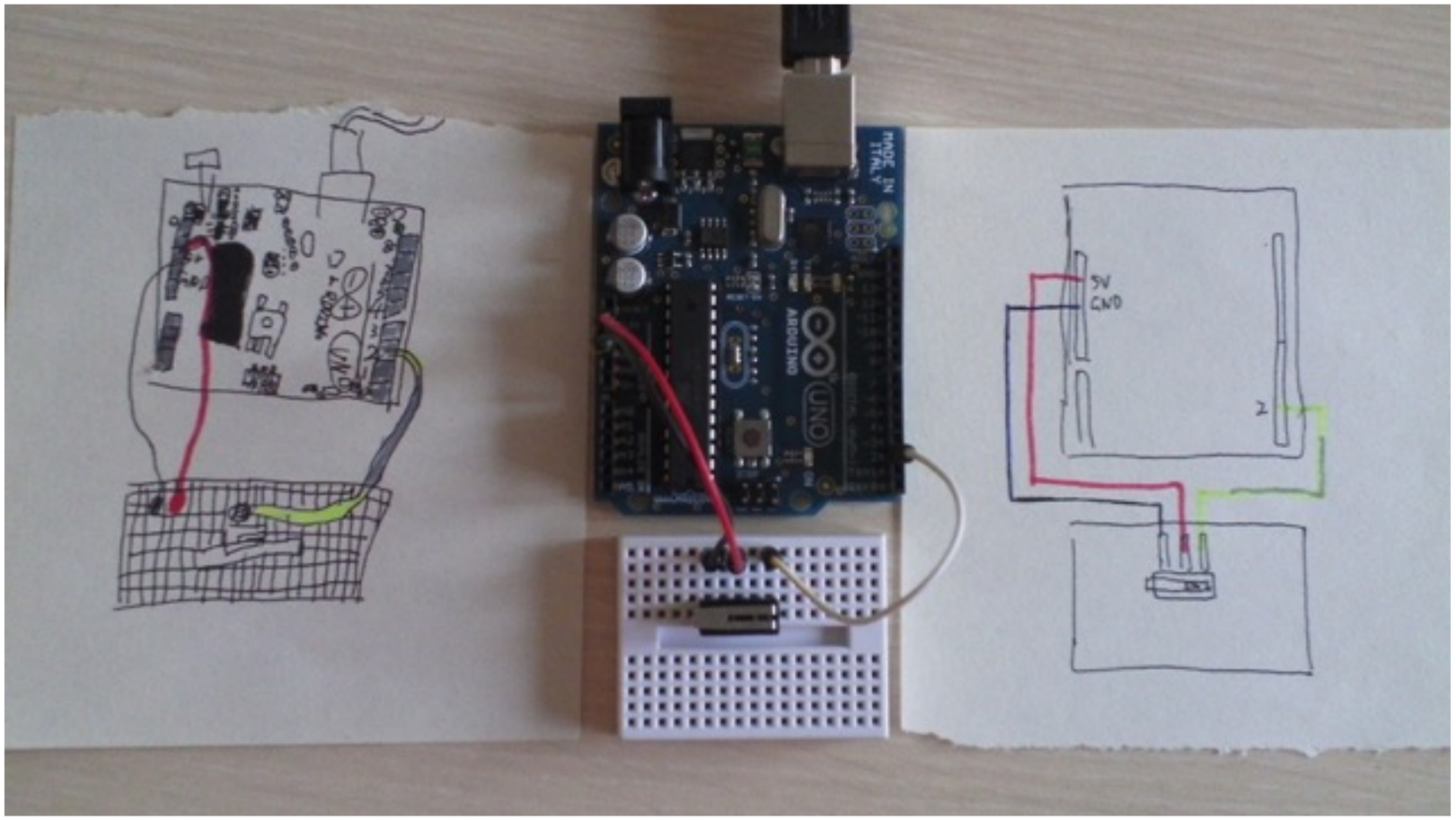
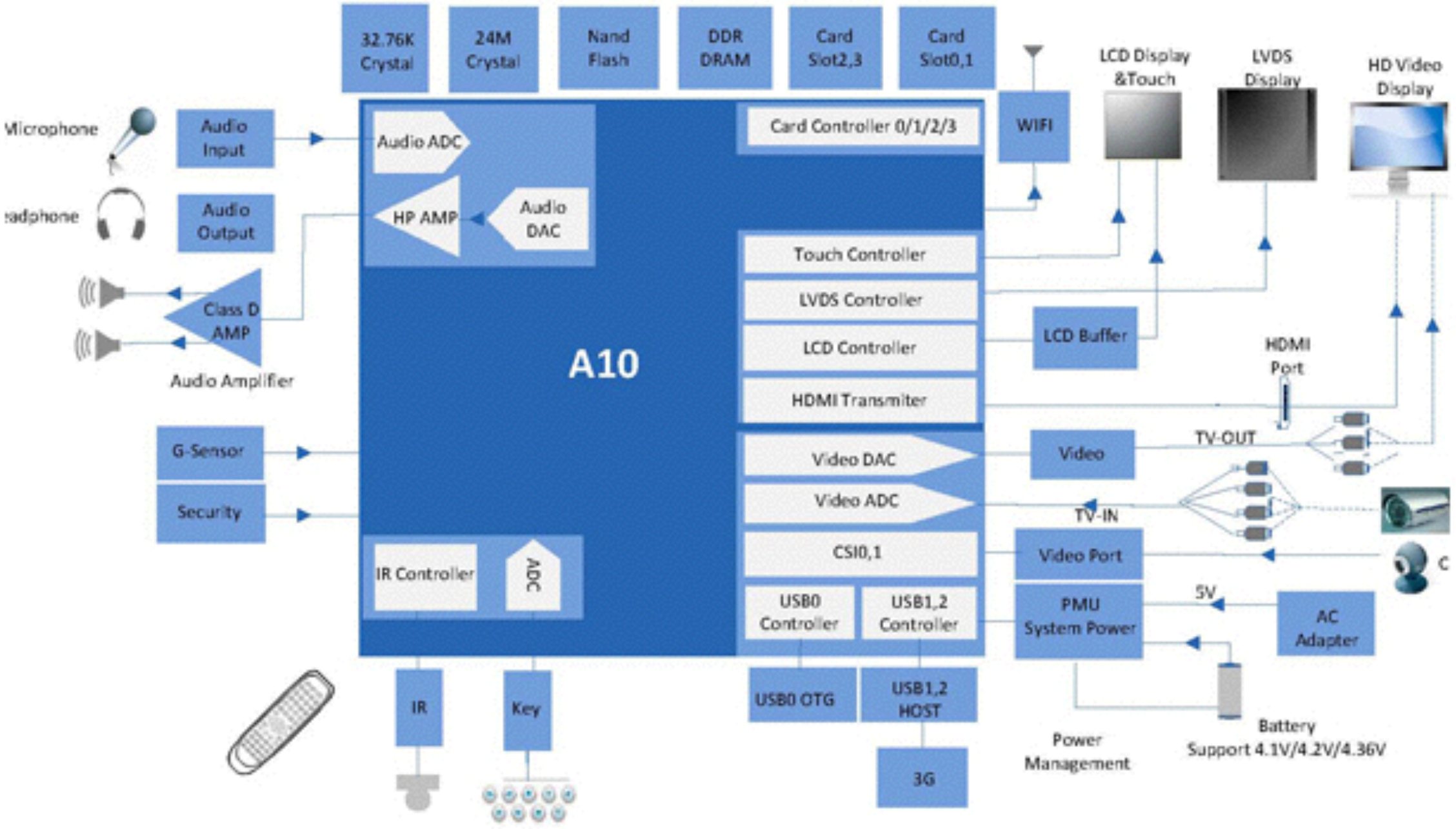


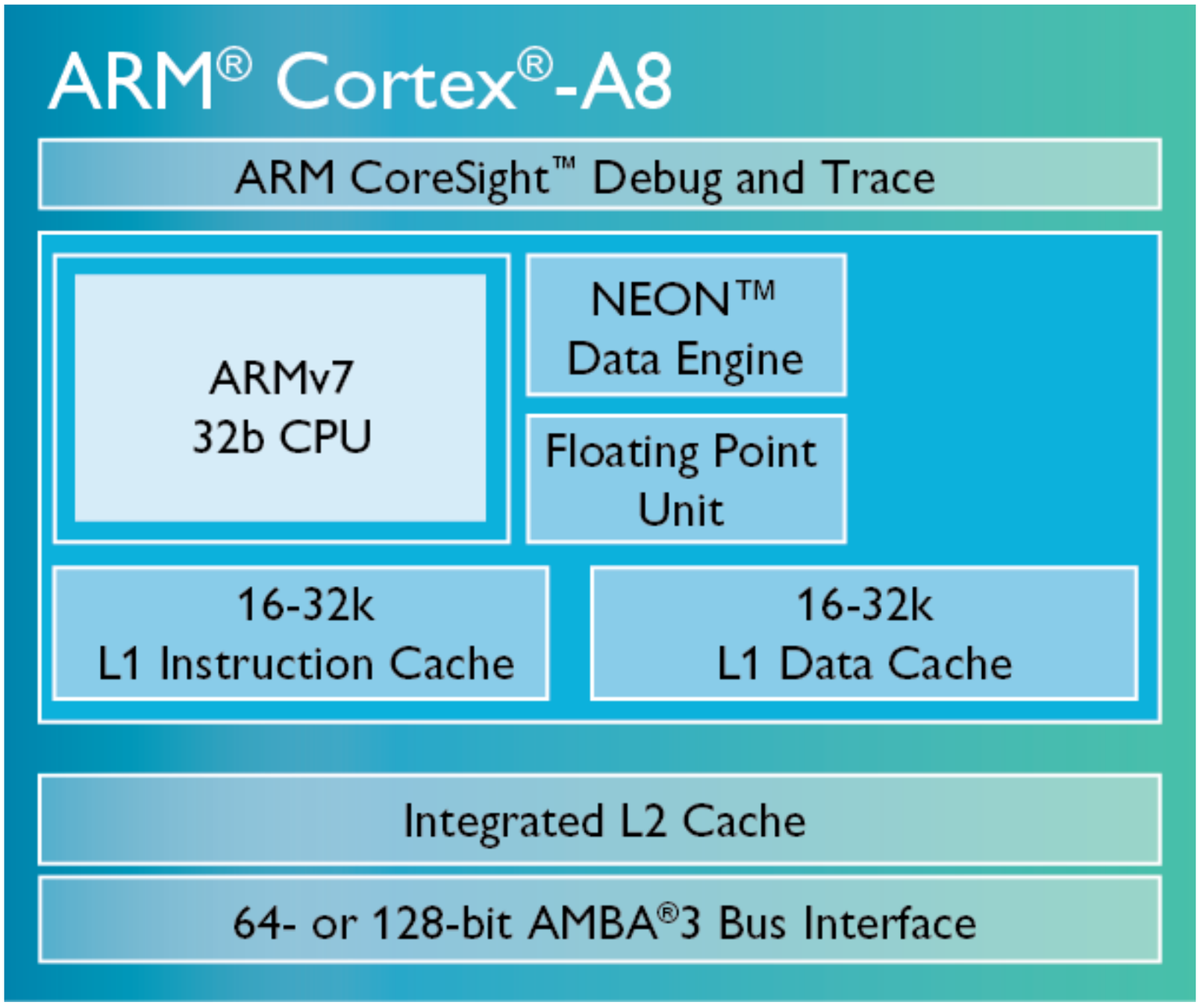
嵌入式系统的接口与外设

快速入门的物理计算

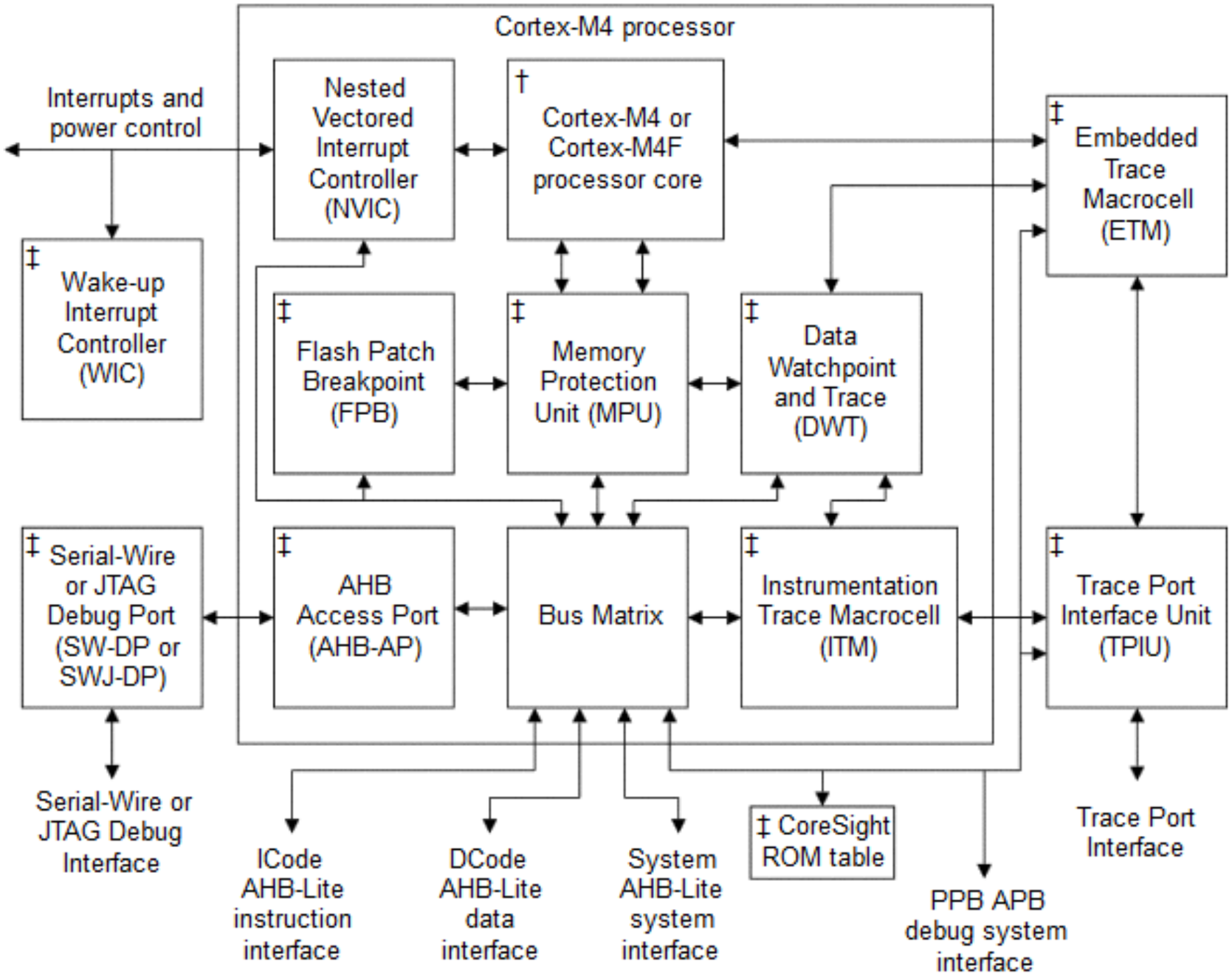
全志A10


32.76K Crystal
24M Crystal
Nand Flash
DDR DRAM
Card Slot2,3
Card Slot0,1
Microphone
Audio Input
Audio ADC
Card Controller 0/1/2/3
WIFI
LCD Display &Touch
LVDS Display
HD Video Display
Audio Output
HP AMP
Audio DAC
Touch Controller
LVDS Controller
Class D AMP
LCD Controller
LCD Buffer
HDMI Port
Audio Amplifier
A10
HDMI Transmiter
G-Sensor
Video DAC
Video
TV-OUT
Security
Video ADC
TV-IN
CSI0,1
Video Port
IR Controller
ADC
USB0 Controller
USB1,2 Controller
PMU System Power
5V
AC Adapter
IR
Key
USB0 OTG
USB1,2 HOST
3G
Power Management
Battery
Support 4.1V/4.2V/4.36V

ARM Cortex-A8



Cortex-M4 Core



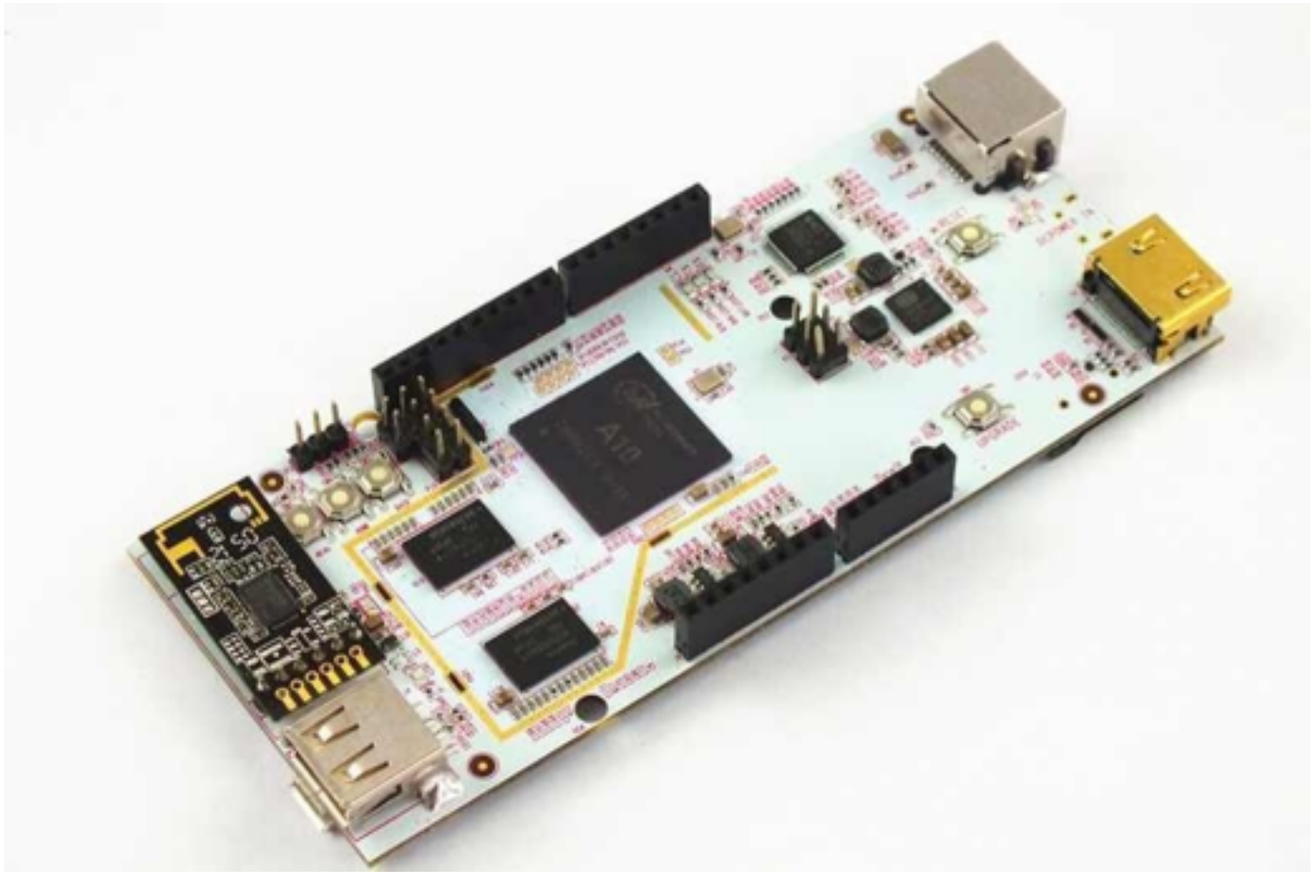
† For the Cortex-M4F processor, the core includes a Floating Point Unit (FPU)

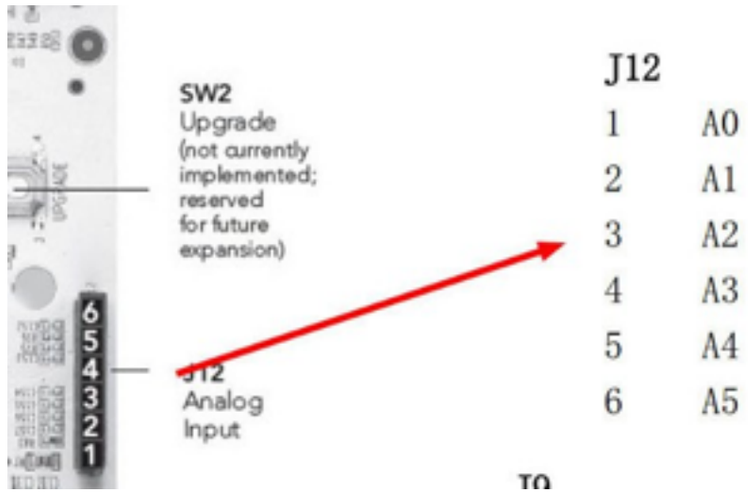
‡ Optional component

处理器vs微控制器

- 处理器的用途是手持、桌面或服务器，IO是为人机交互服务的
- 嵌入式微控制器的用途是控制其他设备，IO是为控制（输入/输出）服务的

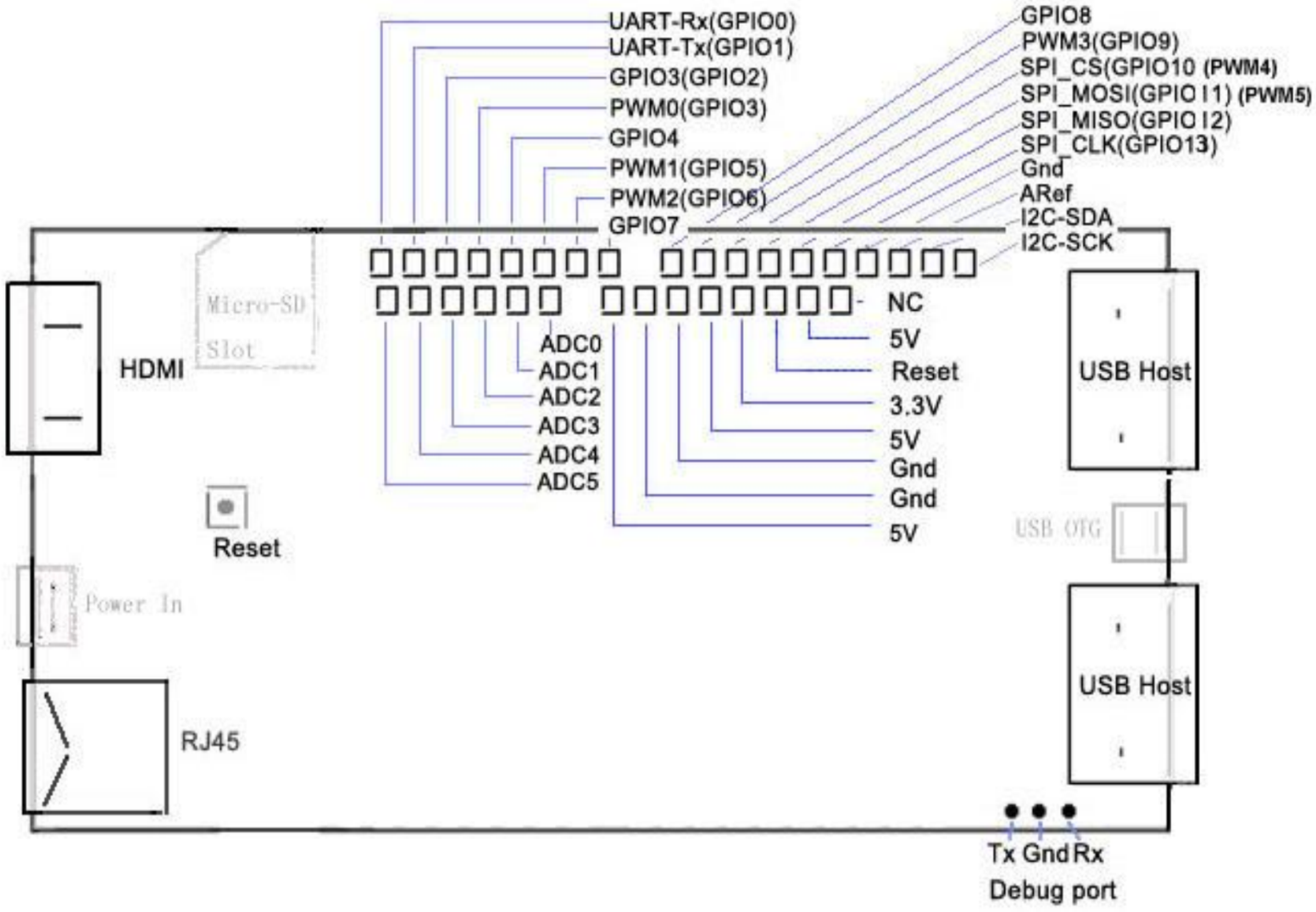
嵌入式系统的GPIO





- GPIO就是可以由CPU直接操纵的引脚，可以写，可以读，可以接通信单元

pcd的端口



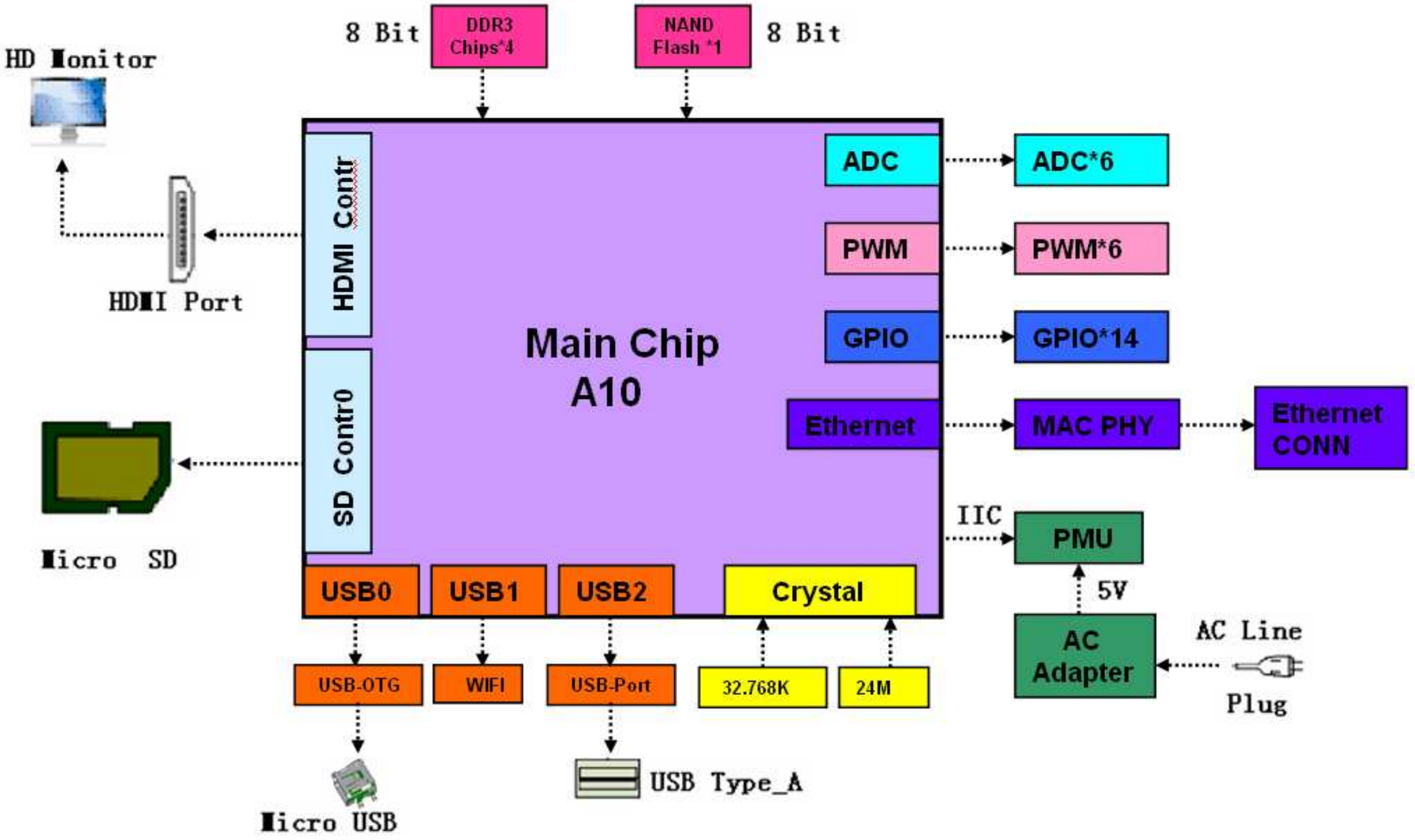
pcDuino

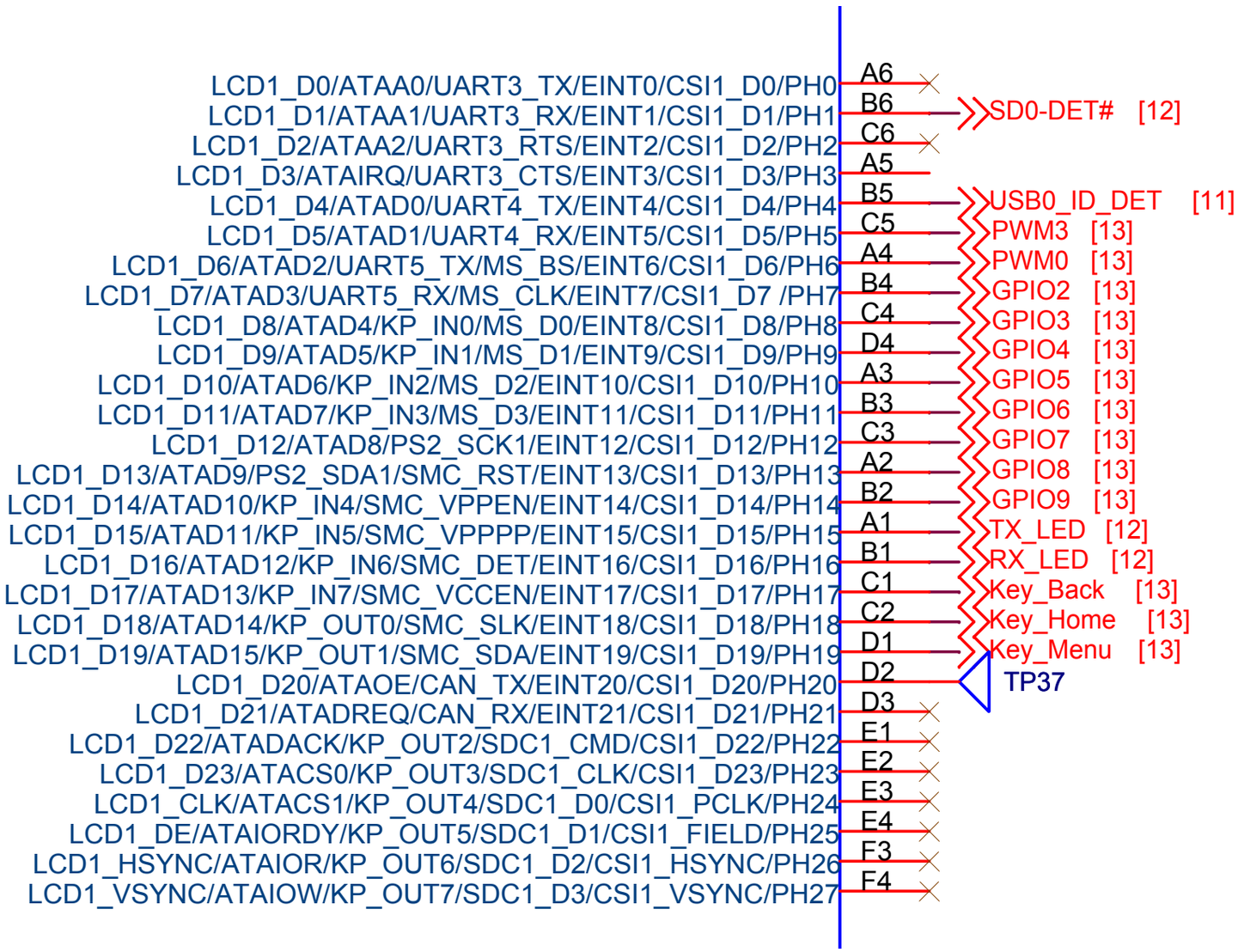
LinkSprite

具体的数据?

- http://www.pcduino.com/images-for-pcduino-v2/
- https://s3.amazonaws.com/pcduino/Hardware/v2/pcDuino_v2_sch.pdf

框图


8 Bit
DDR3 Chips*4
NAND Flash *1
8 Bit
HD Monitor
HDMI Port
HDMI Contr
Main Chip
A10
SD Contr0
Micro SD
ADC
ADC*6
PWM
PWM*6
GPIO
GPIO*14
Ethernet
MAC PHY
Ethernet CONN
IIC
PMU
5V
AC Adapter
AC Line
Plug
USB0
USB1
USB2
Crystal
USB-OTG
WIFI
USB-Port
32.768K
24M
Micro USB
USB Type_A


LCD1_D0/ATAA0/UART3_TX/EINT0/CSI1_D0/PH0
LCD1_D1/ATAA1/UART3_RX/EINT1/CSI1_D1/PH1
LCD1_D2/ATAA2/UART3_RTS/EINT2/CSI1_D2/PH2
LCD1_D3/ATAIRQ/UART3_CTS/EINT3/CSI1_D3/PH3
LCD1_D4/ATAD0/UART4_TX/EINT4/CSI1_D4/PH4
LCD1_D5/ATAD1/UART4_RX/EINT5/CSI1_D5/PH5
LCD1_D6/ATAD2/UART5_TX/MS_BS/EINT6/CSI1_D6/PH6
LCD1_D7/ATAD3/UART5_RX/MS_CLK/EINT7/CSI1_D7 /PH7
LCD1_D8/ATAD4/KP_IN0/MS_D0/EINT8/CSI1_D8/PH8
LCD1_D9/ATAD5/KP_IN1/MS_D1/EINT9/CSI1_D9/PH9
LCD1_D10/ATAD6/KP_IN2/MS_D2/EINT10/CSI1_D10/PH10
LCD1_D11/ATAD7/KP_IN3/MS_D3/EINT11/CSI1_D11/PH11
LCD1_D12/ATAD8/PS2_SCK1/EINT12/CSI1_D12/PH12
LCD1_D13/ATAD9/PS2_SDA1/SMC_RST/EINT13/CSI1_D13/PH13
LCD1_D14/ATAD10/KP_IN4/SMC_VPPEN/EINT14/CSI1_D14/PH14
LCD1_D15/ATAD11/KP_IN5/SMC_VPPPP/EINT15/CSI1_D15/PH15
LCD1_D16/ATAD12/KP_IN6/SMC_DET/EINT16/CSI1_D16/PH16
LCD1_D17/ATAD13/KP_IN7/SMC_VCCEN/EINT17/CSI1_D17/PH17
LCD1_D18/ATAD14/KP_OUT0/SMC_SLK/EINT18/CSI1_D18/PH18
LCD1_D19/ATAD15/KP_OUT1/SMC_SDA/EINT19/CSI1_D19/PH19
LCD1_D20/ATAOE/CAN_TX/EINT20/CSI1_D20/PH20
LCD1_D21/ATADREQ/CAN_RX/EINT21/CSI1_D21/PH21
LCD1_D22/ATADACK/KP_OUT2/SDC1_CMD/CSI1_D22/PH22
LCD1_D23/ATACS0/KP_OUT3/SDC1_CLK/CSI1_D23/PH23
LCD1_CLK/ATACS1/KP_OUT4/SDC1_D0/CSI1_PCLK/PH24
LCD1_DE/ATAIORDY/KP_OUT5/SDC1_D1/CSI1_FIELD/PH25
LCD1_HSYNC/ATAIOR/KP_OUT6/SDC1_D2/CSI1_HSYNC/PH26
LCD1_VSYNC/ATAIOW/KP_OUT7/SDC1_D3/CSI1_VSYNC/PH27
A6
B6
C6
A5
B5
C5
A4
B4
C4
D4
A3
B3
C3
A2
B2
A1
B1
C1
C2
D1
D2
D3
E1
E2
E3
E4
F3
F4
SD0-DET# [12]
USB0_ID_DET [11]
PWM3 [13]
PWM0 [13]
GPIO2 [13]
GPIO3 [13]
GPIO4 [13]
GPIO5 [13]
GPIO6 [13]
GPIO7 [13]
GPIO8 [13]
GPIO9 [13]
TX_LED [12]
RX_LED [12]
Key_Back [13]
Key_Home [13]
Key_Menu [13]
TP37

概要

- **如何让一个程序根据开关的动作来点亮一个LED?**

- **GPIO**
  - 基本概念
  - 端口电路
  - 控制寄存器
  - 在C语言中访问硬件寄存器
  - 时钟和复用

- **电路接口**
  - 输入
  - 输出

- **其他端口配置**

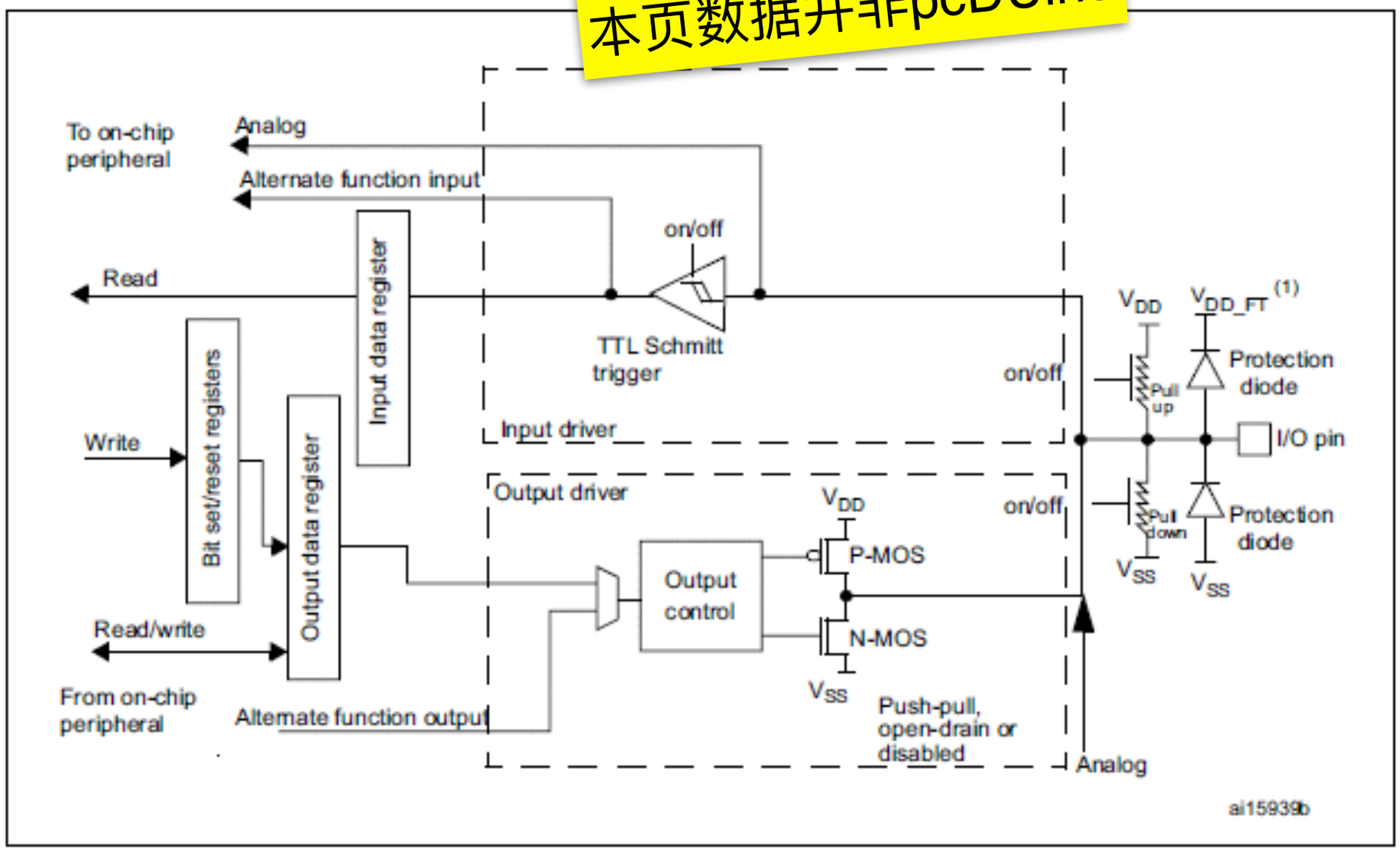
基本概念

- **GPIO = 通用输入输出（数字）**
  - 输入：程序可以判断输入信号是1还是0
  - 输出：程序可以设置输出1或0
- **可以以此来接外部器件**
  - 输入：开关/按钮
  - 输出：LED、扬声器

单片机中的GPIO端口位电路

- **配置**
  - 方向
  - 复用
  - 模式
  - 速度
- **数据**
  - 输出（不同的访问途径）
  - 输入
  - 模拟
- **锁定**

本页数据并非pcDUino



控制寄存器

- 每个通用I/O端口具有
  - 四个32位配置寄存器 (
    - GPIOx_MODER (输入、输出、AF、模拟)
    - GPIOx_OTYPER (输出类型：推挽或开漏)
    - GPIOx_OSPEEDR(速度)
    - GPIOx_PUPDR(上拉/下拉)
  - 两个32位数据寄存器 (GPIOx_IDR 和GPIOx_ODR)
  - 一个32位置位/清除寄存器 (GPIOx_BSRR)
  - 一个32位锁定寄存器 (GPIOx_LCKR)
  - 两个32位可选功能选择寄存器 (GPIOx_AFRH 和GPIOx_AFRL)
- **每个端口一组控制寄存器（总共10个）**
- **控制寄存器中的每个位对应端口的一个位**
- **所有的寄存器都必须以32位的字来访问**

本页数据并非pcDUino

GPIO配置寄存器

- 每一位可以独立配置
- 启动的时候把每个端口位的方向都清除为0了
- 输出模式：推挽或开漏+上拉/下拉
- 从(GPIOx_ODR)输出数据寄存器输出数据，或使用外设（可选功能输出）
- 输入状态：悬空、上拉/下拉、模拟
- 输入数据到“输入数据寄存器” (GPIOx_IDR)或外设（可选功能输入）

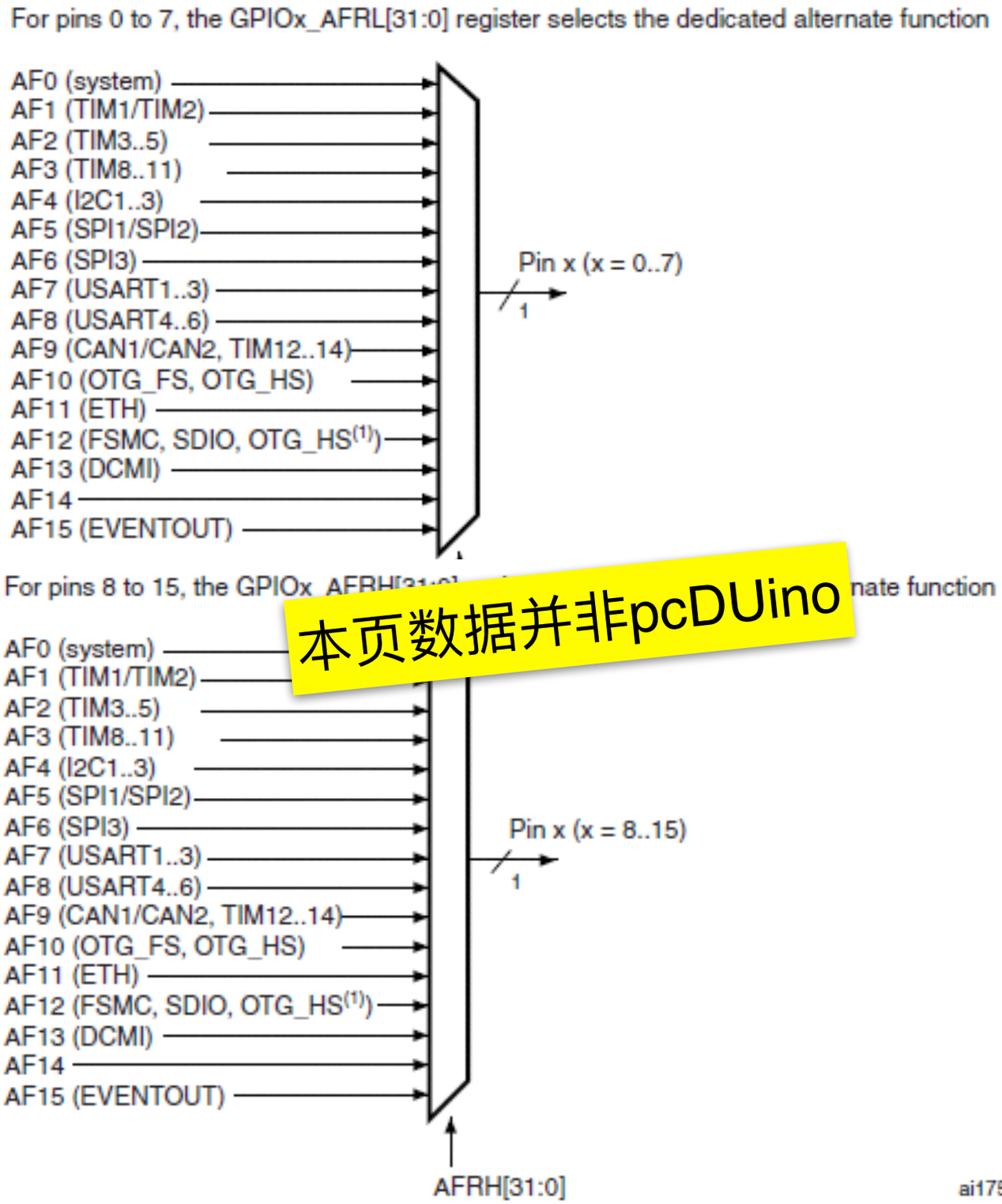
MODER(I) [1:0]	OTYPER(I)	OSPEEDR(I) [B:A]	PUPDR(I) [1:0]		I/O configuration	
01	0	SPEED [B:A]	0	0	GP output	PP
	0		0	1	GP output	PP + PU
	0		1	0	GP output	PP + PD
	0		1	1	Reserved	
	1				utput	OD
	1				utput	OD + PU
	1			0	GP output	OD + PD
	1		1	1	Reserved (GP output OD)	
10	0	SPEED [B:A]	0	0	AF	PP
	0		0	1	AF	PP + PU
	0		1	0	AF	PP + PD
	0		1	1	Reserved	
	1		0	0	AF	OD
	1		0	1	AF	OD + PU
	1		1	0	AF	OD + PD
	1		1	1	Reserved	
00	x	x x	0	0	Input	Floating
	x	x x	0	1	Input	PU
	x	x x	1	0	Input	PD
	x	x x	1	1	Reserved (input floating)	
11	x	x x	0	0	Input/output	Analog
	x	x x	0	1	Reserved	
	x	x x	1	0		
	x	x x	1	1		

1. GP = general-purpose, PP = push-pull, PU = pull-up, PD = pull-down, OD = open-drain, AF = alternate function.

本页数据并非pcDUino

可选功能选择寄存器

- **在AF模式，AFRL或AFRH需要配置过才能由特定的外设驱动引脚**
- **可以看作是复用器的选择信号**
- EVENTOUT不能被映射到以下 I/O 引脚： PC13、PC14、PC15、PH0、PH1 和PI8.



本页数据并非pcDUino

代码风格和按位访问

- **直接用二进制和十六进制的常数容易犯错**
  - “要设置第13和19位，用0000 0000 0000 1000 0010 0000 0000 0000 或 0x00082000”

- **用位移位来产生常数值**

```
n = (1UL << 19) | (1UL << 13);
```

- **为所需的位定义名字**

```
#define POS_0 (13)
#define POS_1 (19)
n = (1UL << POS_0) | (1UL << POS_1);
```

- **建立宏MASK来做移位产生屏蔽字**

```
#define MASK(x) (1UL << (x))
n = MASK(POS_0) | MASK(POS_1);
```

使用MASK宏

- **用屏蔽字覆盖已有的值**

  n = MASK(foo);

- **置屏蔽字中为1的位为1，其他位不变**

  n |= MASK(foo);

- **计算屏蔽字的补码**

  ~MASK(foo);

- **清除屏蔽字中为0的位为0，其他位不变**

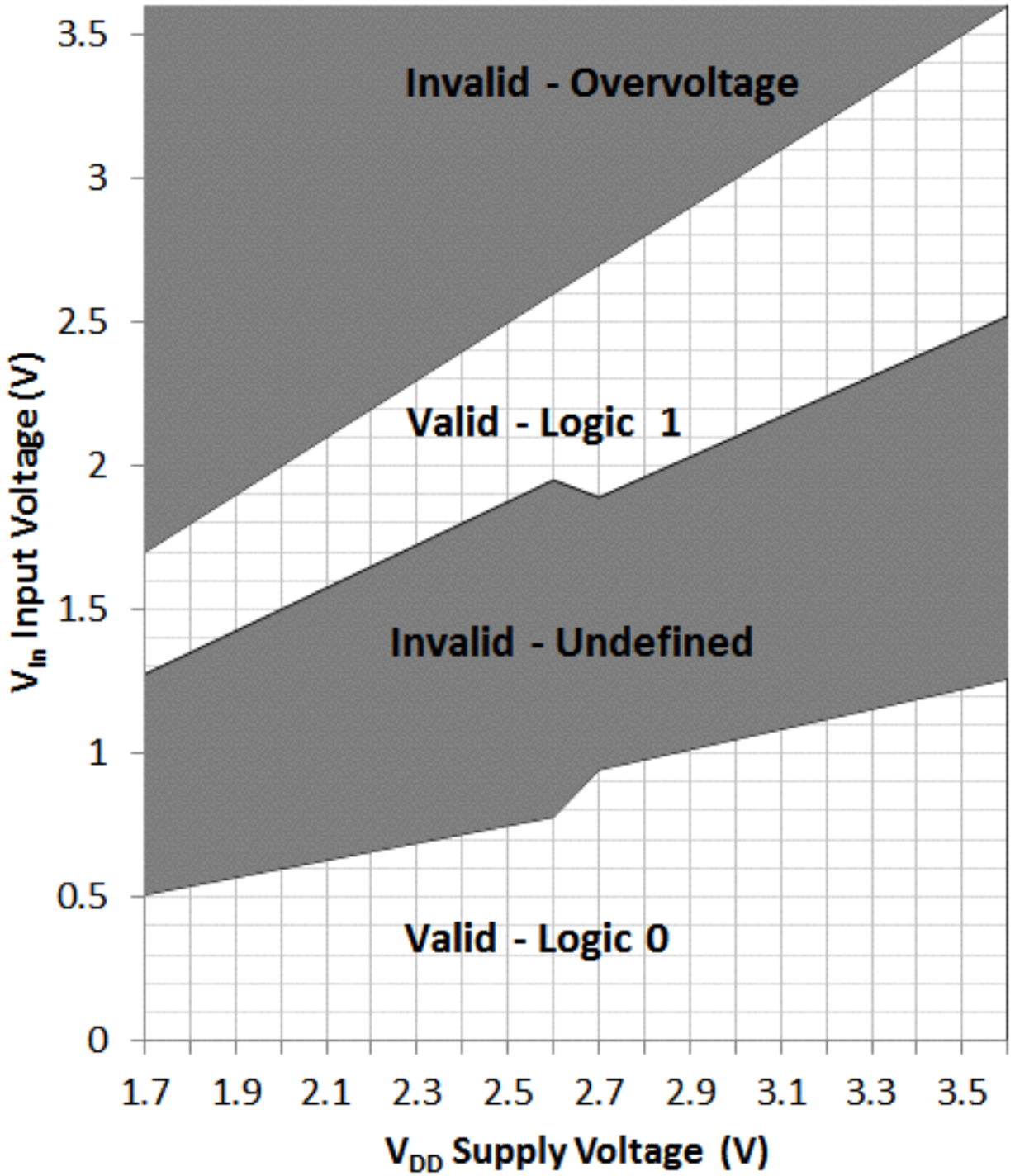
  n &= MASK(foo);

- **清除屏蔽字中为1的位为0，其他位不变**

  n &= ~MASK(foo);

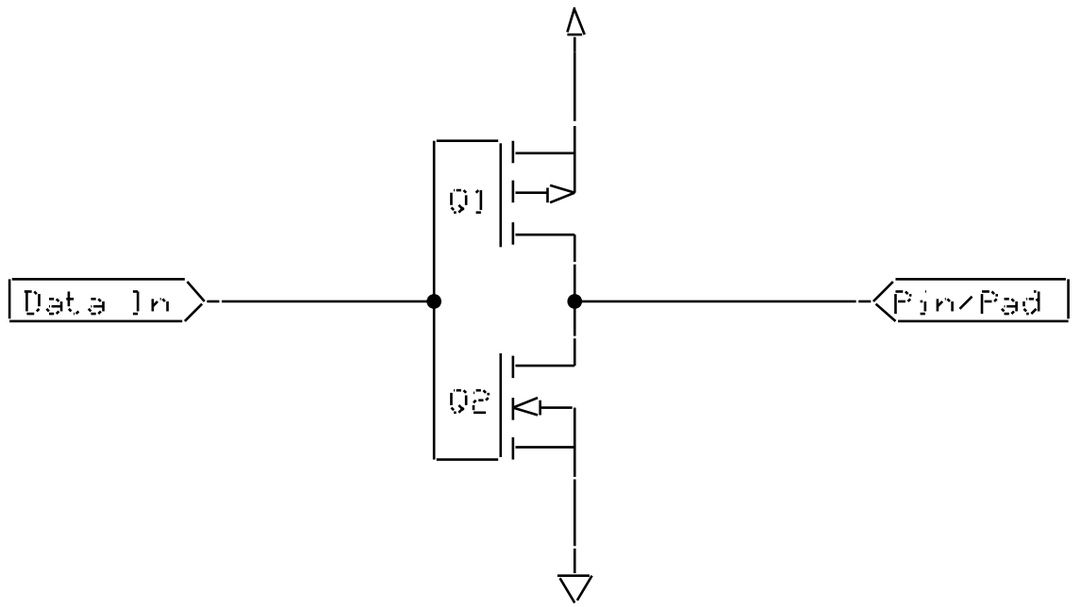
输入：什么是1？0呢?

- 输入信号的值是由电压决定的
- 输入阈值电压是由供电电压V_{DD}所决定的
- 超过V_{DD} 或 GND 可能会损坏芯片

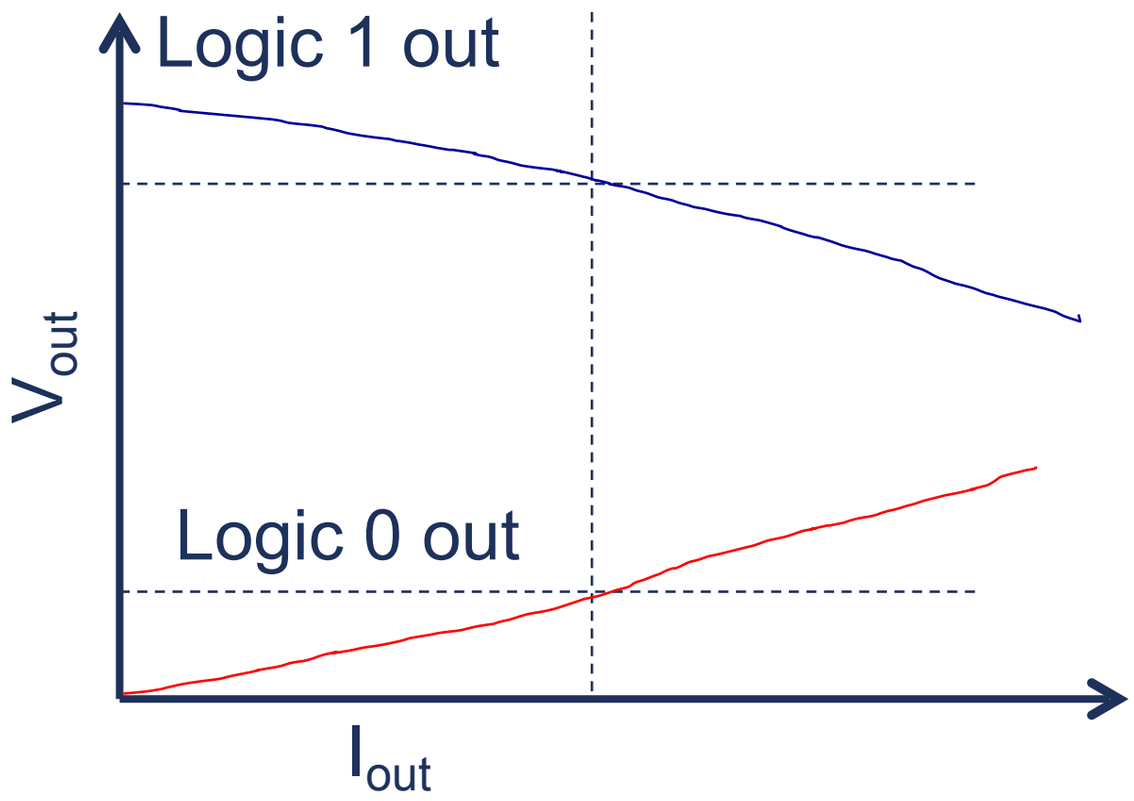


输出：什么是1？0呢?

- **正常输出电压**
  - 1： V_{DD}-0.5 V 到 V_{DD}
  - 0： 0 to 0.5 V



- **注意：输出电压受引脚上的负载所汲取的电流的影响**
  - 需要考虑晶体管里的源极到漏极的电阻
  - 上述值只是当电流<5mA（对于高驱动能力引脚是18mA）并且V_{DD} > 2.7V时才是有意义的



GPIO

- 对于一个GPIO引脚，最基本要做的事情是和它是用作输入还是输出有关。如果是输入，要看它所连接的电信号是否有效，而如果是输出，要设置它为有效或无效。

- 更高级的使用方式还可以硬件监视一个输入引脚的状态，当状态变化的时候中断计算机

软件中的GPIO

- 允许或禁止一个GPIO引脚；
- 设置何种信号电平为“有效”；
- 决定引脚是输入还是输出；
- 向一个引脚写一个值；
- 从引脚读一个值；
- 设置哪种边沿会产生输入中断；
- 等待中断发生。

性能问题

- 信号变化的速度或者说频率。在理想世界里，一个引脚可以达到的最高频率只是由电特性所决定的，就是由GPIO电路的特性和它所驱动的负载或感应的谐振能力所决定的。不过，很可能在信号处理过程中的软件也会实际上造成频率的限制。软件所花的时间会限制每秒能变化的次数，或每秒能检测到的变化。

- 全志A10使用AHB总线，所以GPIO上的切换速度只有MHz级别

Linux内核驱动程序

- 驱动程序实现了一套C的函数，内核的其他部分可以用它们来建立使用GPIO的某种具体的硬件设备的驱动程序

- 这个级别的程序在读写一个引脚的时候具有非常低的延迟，但是也受制于系统中断延迟的程度。服务于硬件其他部分的中断响应程序会造成可能的延迟

- http://www.kernel.org/doc/Documentation/gpio.txt

应用级别的支持

- 可以通过sysfs目录中特殊的文件来控制GPIO引脚。这些文件不是真实的和磁盘上的数据有关的文件，当读写这些文件的时候，内核中的程序会被调用来实现用户所需的功能。
- 启用、停用、读写、设置何种信号电平为有效以及决定何种边沿会触发中断，是由对这些文件的读和写这样的普通文件操作来实现的。等待一个中断，是由标准的Unix类操作系统一般都支持的“poll”或“select”这样的函数，来通知用户在某个已经打开的文件上发生了任何异常状况的。

/sys的局限

- 这个级别的程序会受到两种额外的延时和延迟的影响。一种是调度延迟。这实际上是Linux在别的应用程序正在使用处理器时，让另一个应用程序获得处理器所需的时间。另一种是系统负载，当其他更重要的应用程序需要使用处理器时，系统会不让你的程序使用处理器。在这点上还有一些控制可以做，因为你的程序的重要性是可以用“nice”命令来改变的。不过，通常这两种延迟主要都是在等待一个中断处理完毕时发生的。

/sys/class/gpio

- export：写入引脚编号表明启用
- unexport：写入引脚编号来停用
- 编号和实际CPU引脚的关系：Linux移植者决定

启用后

- active_low：0/1；是否低电平有效
- direction：in/out
- edge：none、rising、falling或both；中断触发类型
- value：读当前值；select来等待中断

直接访问寄存器

- AHB上的寄存器是挂在总线上的，e.g.每个寄存器有一个32位的地址
- 在用户模式下，MMU的设置使得应用程序无法访问这些寄存器
- 内核程序，如设备驱动程序可以访问
- 裸机程序可以直接访问
- 应用程序经过特殊设置也可以让某块内存映射到AHB上

Arduino-ish

- An Arduino wiring-like library written in C and released under the GNU LGPLv3 license which is usable from C and C++ and many other languages with suitable wrappers

blink LED

```
pinMode (0, OUTPUT) ;

for (;;)
{
    digitalWrite (0, 1) ;          // On
    delay (500) ;                  // mS
    digitalWrite (0, 0) ;          // Off
    delay (500) ;
  }
```

APIs

- wiringPiSetup(void) ;
- void pinMode (int pin, int mode) ;
- void digitalWrite (int pin, int value) ;
- void digitalWriteByte (int value) ;
- void pwmWrite (int pin, int value) ;
- int digitalRead (int pin) ;
- void pullUpDnControl (int pin, int pud) ;
- unsigned int millis (void);
- void delay (unsigned int howLong);
- void delayMicroseconds (unsigned int howLong);

安装Arduino-ish

- git clone https://github.com/pcduino/c_enviroment
- cd c_enviroment
- make
- #include <core.h>

物理计算

- 计算：输入-->输出
- 物理计算：输入或输出是物理世界
  - 输入：传感器sensor
  - 输出：动作器actor

电压

- 电压：电路上两点之间的电位差异，单位V
  - 通常以与地GND之间的电位差异来表达成一个绝对值
- 工作电压表示器件正常的工作电压
- 额定（最高）电压表示器件最高能承受的电压
- 5V vs 3.3V
  - 常见的数字器件的工作电压
  - 很多芯片工作在3.3V，但是引脚可以承受5V

电流

- 单位时间流过器件的电子，单位A，常用mA
- 电源决定电压，负载决定电流
- 器件的工作电流表示正常工作时消耗的电流
- 电源的输出电流表示正常工作时能输出的最大电流
  - 超过该电流时工作不正常，如电压下降
- 选择电源的原则：输出电压等于工作电压、输出电流不小于工作电流

电阻

- 电阻 = 电压 / 电流，单位Ω
- 电路总是形成一个回路，从电源出发，经过负载，回到电源。负载在这个回路中就表现为一个电阻
- 这个电阻越大，电流就越小
- 电源本身也具有一定的电阻（内阻），它决定了电源的输出能力（恒定的是功率）
    - 稳压电源：具有动态调节内阻的能力以保证输出的电压恒定

功率

- 功率 = 电压 * 电流，单位W
- 电路中消耗的功率，一部分用于"计算"，一部分产生热量
- 电源的功率表达它的输出能力
- 负载的功率表达它的消耗能力
- 元件的功率表达它的承受能力

安时

- 电流乘以时间，单位AH
- 表达电源的储存能力
- 1AH的电池表示如果持续供应1A的电流，1小时内电池耗尽（电压低于额定输出电压）
- 电池不是稳压电源，所以电压是持续下降的
- 单节锂电的电压是3.6V，充满后的电压是4.2V
- 移动电池内部一般具有DC-DC电路，兼具升压和稳压

表示倍数的符号

- M：10^{6}
- k：10^{3}
- m：10^{-3}
- μ：10^{-6}
- n：10^{-9}
- p：10^{-12}

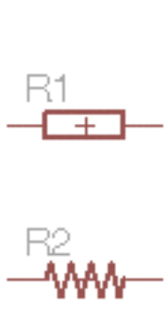
USB供电

- 5V
- 标准规定，主机向枚举的设备提供最大500mA电流
- 对未枚举的设备只能提供100mA电流

直流vs交流

- 直流：电流方向永远不变
- 交流：电流方向周期性变化
  - 变化的速度：频率，以Hz为单位

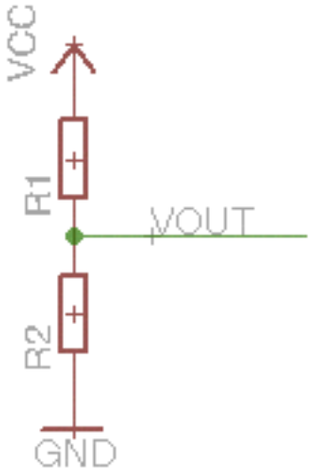
电阻



- 电阻具有两端，无极性
- 电流流经电阻时在电阻的两端产生压降，进入端电压高，压降＝电流*电阻， 无论直流还是交流均相同
- 电阻在电路图中标为R，常见的阻值以kΩ为单位
- 商品电阻的阻值是在一个固定的序列中的
- 电阻的精度5%指±5%范围，常见的5%，高精度1%
- 电阻的功率指它能承受的最大功率

电阻的用途

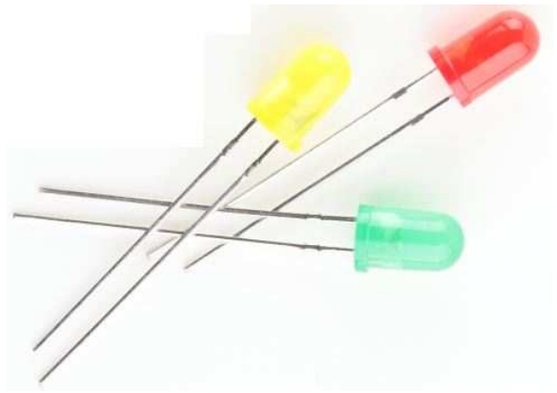
- 限流：如LED限流电阻
- 分压：从一个电压得到另一个电压
- 拉动：上拉/下拉
- 把电流转换为电压：为了测量电流

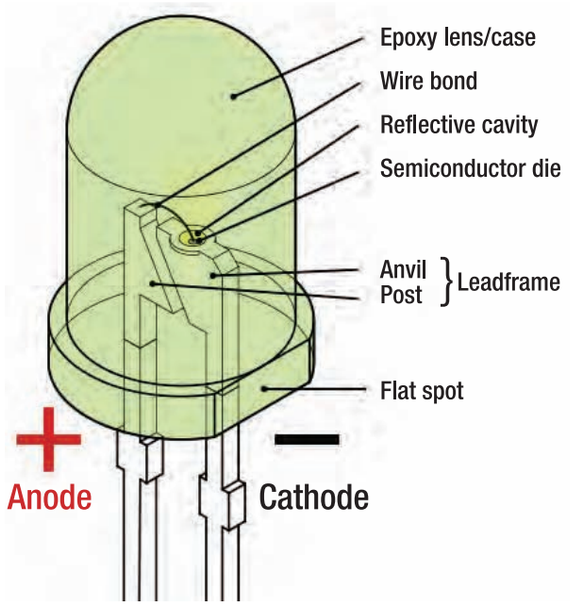


分压电路

- VCC：电源。本意表示所有的晶体管的集电极连接的地方，所以也有VDD，表示所有MOS管的漏极
- GND：地
- 流经两个串联电阻的电流i = U/(R1+R2)
- 在R2上产生的压降VOUT=i*R2
- 所以VOUT = UR2/(R1+R2)
- 如果VOUT右边接了负载，负载的电阻得要与R2并联计算
    - 分压电路不能用来提供电源！！

LED

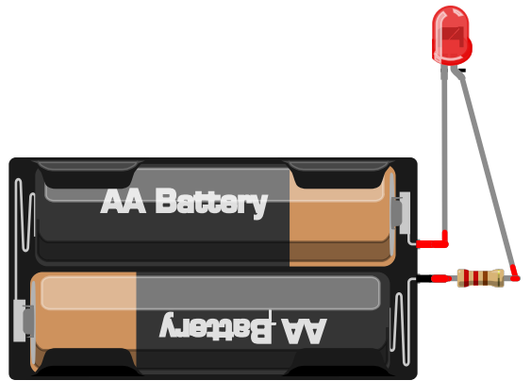




直接将LED接在引脚和VCC或GND之间并不总是安全的

LED数学

- 导通后，二极管在正向上有固定的压降，与电流无关
- 红色LED的导通压降大约2.0V，消耗大约20mA的电流。
- 其他颜色LED，越在光谱的高端的一般压降越大，或电流越大



- 限流电阻R＝(5-2)/20=0.15k
  - 如果没有限流电阻?
- 一般选择更高的电阻（200-470）以保护LED
  - 别把人逼急了:)

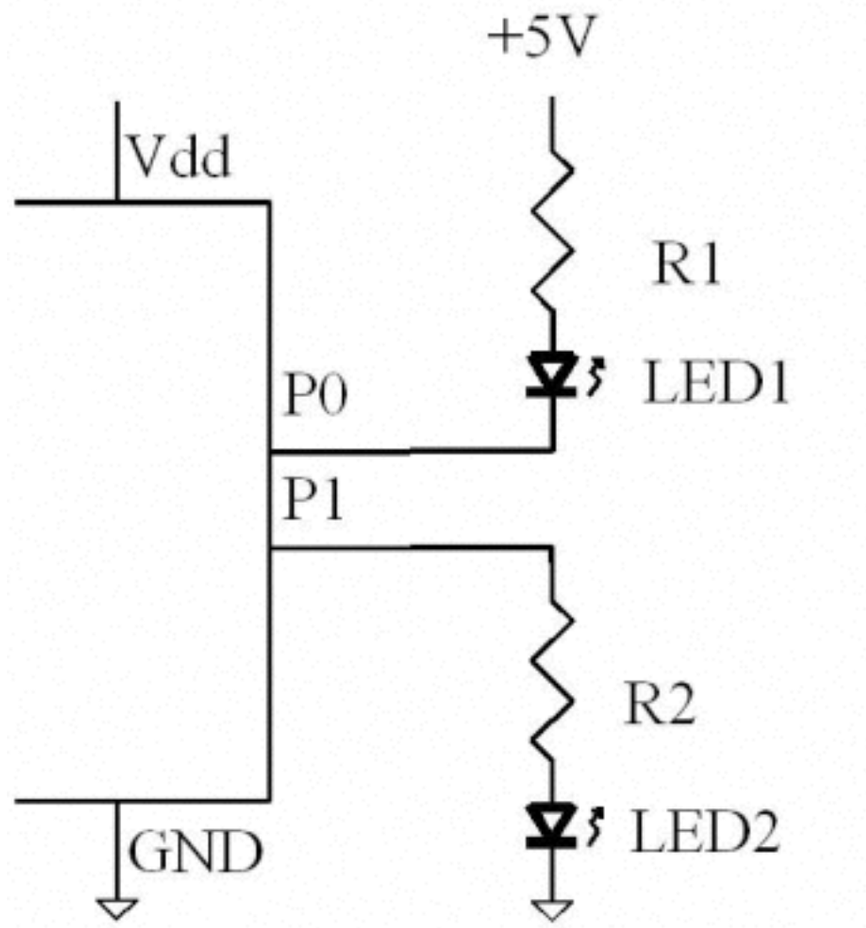
输出HIGH和LOW

- 扇出：输出HIGH时，电流从CPU的Vcc流入，经过引脚输出给外部

- 灌入：输出LOW时，电流从外部的Vcc流入，经过引脚后，直接到地

- 对于某些CPU，使用灌入方式，整体能驱动更大的设备

- 工业界一般采用灌入



安全提示

- pcd没有在GPIO上做任何保护
- 断电之后再接线或拆线
- 上电之前先检查接线
  - 尤其注意有没有直接将电源和地连起来的
- 不要直接接5V的器件