嵌入式系统简介

信息工程学院 蔡亮

为什么要学习嵌入式技术？

从计算发展的趋势看

Size

One computer for many people

Number

One computer for each person

Many computers for each person

软件工程师Vs嵌入式软件工程师

软件工程师

嵌入式

软件工程师

提　要

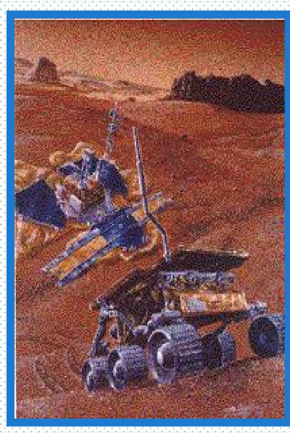
1. 嵌入式系统的发展及应用领域
2. 嵌入式系统的定义与体系结构
3. 嵌入式系统的应用案例

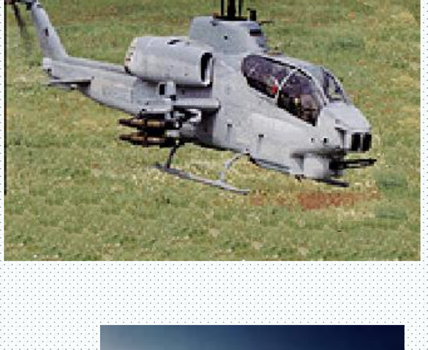
一些典型的嵌入式系统应用实例

goReader
Internet
eBook

Tektronix
TDS7000 Digital
Oscilloscopes

Samsung AnyWeb
Internet Screen
Phone





Nixvue Digital Album
Digital Photo Album

eRemote
Intelligent Home
Controller

嵌入式系统的应用

家用方面：数字电视、信息家电、智能玩具、手持通讯、存储设备的核心。

现代化家庭

因特网

电话网络

小区网络

烟感

煤气探头

红外探头

门磁

紧急按钮

光线探测器

智能家居控制器

吊灯

壁灯

电动窗帘

电视

音响

洗衣机

电冰箱

微波炉

电饭煲

空调

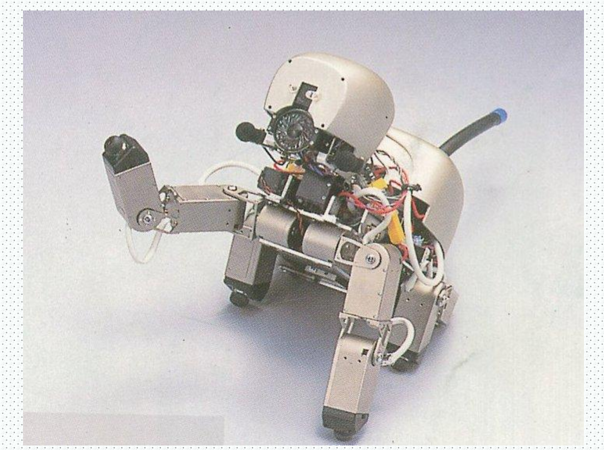
水表

电表

气表

热表

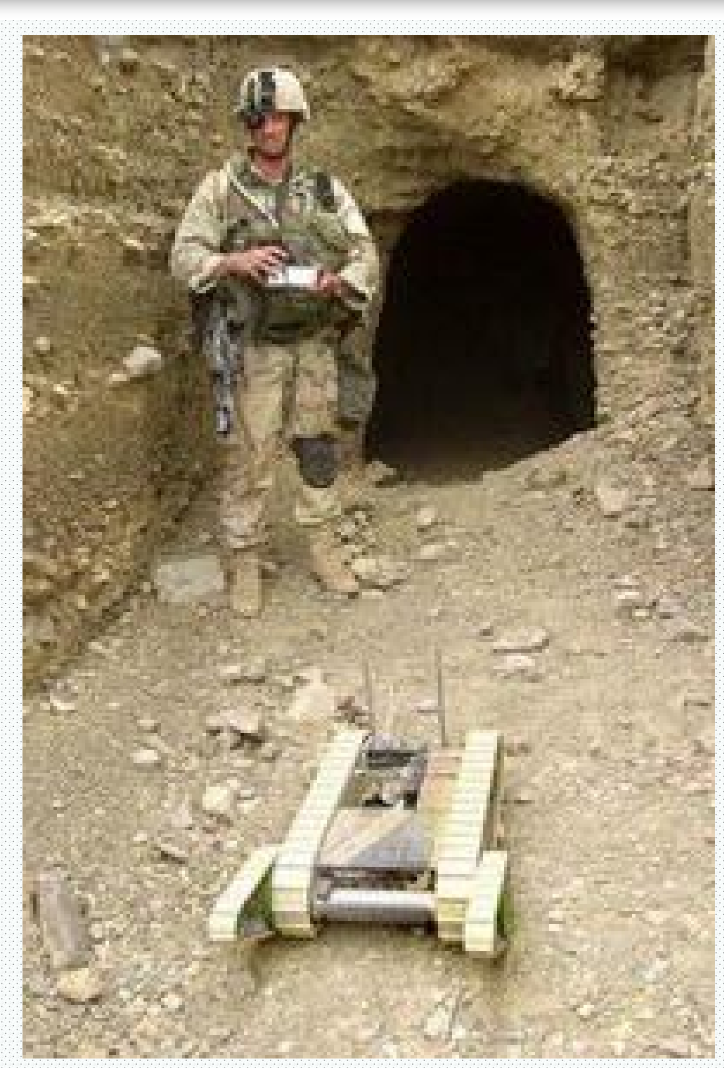
智能玩具与机器人



嵌入式技术应用——工业控制

工业方面：机床、冶金、电子、交通、航空航天等行业业技术升级的重要基础；

军事侦察

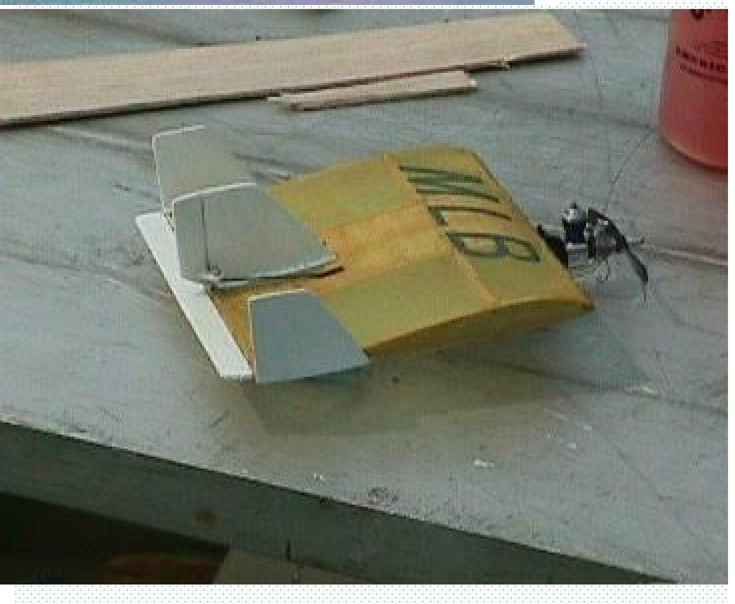


阿富汗参加反恐作战的“赫耳墨斯”价值4万美元，可携带2架摄像机，发挥了很好作用。

反恐防暴

以色列一选举投票点，发生枪击事件，造成至少7人死亡，数十人受伤。以警方用机器人在检查一具巴勒斯坦枪手的尸体。

空中飞行器



嵌入式系统与机器人技术

图：卡耐基梅隆大学和瑞士EPFL研制的机器人控制器
（采用卡西欧PDA和Windows CE）

基于VXworks的火星探路者

嵌入式系统的市场分析

产业发展趋势：嵌入式系统外包业务的高速发展

市场发展趋势：通信与汽车电子市场对嵌入式系统产业的整体推动作用

技术发展趋势：

- 嵌入式软件工程设计方法：基于组件的设计方法、模型驱动的设计方法
- 基于FPGA的嵌入式系统设计
- 嵌入式多核处理器的设计与应用

嵌入式系统的发展—Moor定律的影响

Moor定律的影响：IC每两年晶体管数量增加一倍（微处理器性能增加一倍），而价格下降一半；

由制造工艺与散热的限制，多核处理器是未来的发展趋势；

多核处理器带来的设计挑战：操作系统技术、设计工具、软件开发方法……

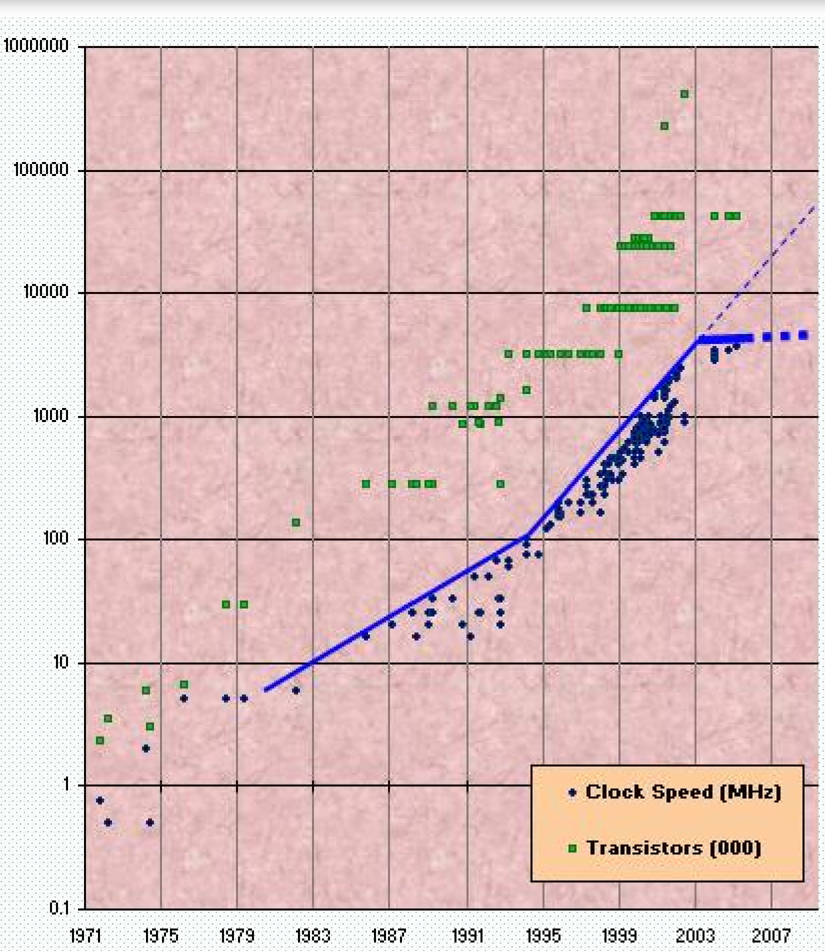


Figure 1: Intel CPU Introductions
(sources: Intel, Wikipedia)

提　要

1. 嵌入式系统的发展及应用领域
2. 嵌入式系统的定义与体系结构
3. 嵌入式系统的应用案例

IEEE定义

根据IEEE（国际电气和电子工程师协会）的定义：

嵌入式系统是"用于控制、监视或者辅助操作机器和设备的装置"（原文为devices used to control, monitor, or assist the operation of equipment, machinery or plants）。

可以看出此定义是从应用上考虑的，嵌入式系统是软件和硬件的综合体，还可以涵盖机电等附属装置。

一般定义

"以应用为中心、以计算机技术为基础、软件硬件可裁剪、功能、可靠性、成本、体积、功耗严格要求的专用计算机系统。"

嵌入式系统

广义上讲，凡是带有微处理器的专用软硬件系统都可称为嵌入式系统。如各类单片机和DSP系统。这些系统在完成较为单一的专业功能时具有简洁高效的特点。但由于他们没有操作系统，管理系统硬件和软件的能力有限，在实现复杂多任务功能时，往往困难重重，甚至无法实现。

从狭义上讲，我们更加强调那些使用嵌入式微处理器构成独立系统，具有自己操作系统，具有特定功能，用于特定场合的嵌入式系统。这里所谓的嵌入式系统是指狭义上的嵌入式系统。

实时系统

实时系统

- 实时系统指系统的计算正确性不仅取决于计算的逻辑正确性，还取决于产生结果的时间。如果未满足系统的时间约束，则认为系统失效。
- 所谓“实时”，是表示“及时”，而实时系统是指系统能及时响应外部事件的请求，在规定的时间内完成对该事件的处理，并控制所有实时任务协调一致的运行。

实时系统的特征（1）

时间约束性

- 实时系统的任务具有一定的时间约束（截止时间）。根据截止时间，实时系统的实时性分为“硬实时”和“软实时”。
- 硬实时是指应用的时间需求能够得到完全满足，否则就造成重大安全事故，甚至造成重大的生命财产损失和生态破坏，如在航空航天、军事、核工业等一些关键领域中的应用。
- 软实时是指某些应用虽然提出时间需求，但实时任务偶尔违反这种需求对系统运行及环境不会造成严重影响，如监控系统等和信息采集系统等。

实时系统的特征（2）

可预测性

- 可预测性是指系统能够对实时任务的执行时间进行判断，确定是否能够满足任务的时限要求。
- 由于实时系统对时间约束要求的严格性，使可预测性称为实时系统的一项重要性能要求。
- 除了要求硬件延迟的可预测性以外，还要求软件系统的可预测性，包括应用程序的响应时间是可预测的，即在有限的时间内完成必须的工作；以及操作系统的可预测性，即实时原语、调度函数等运行开销应是有界的，以保证应用程序执行时间的有界性。

实时系统的特征（3）

可靠性

- 大多数实时系统要求有较高的可靠性。在一些重要的实时应用中，任何不可靠因素和计算机的一个微小故障，或某些特定强实时任务（又叫关键任务）超过时限，都可能引起难以预测的严重后果。

- 为此，系统需要采用静态分析和保留资源的方法及冗余配置，使系统在最坏情况下都能正常工作或避免损失。可靠性已成为衡量实时系统性能不可缺少的重要指标。

实时系统的特征（4）

与外部环境的交互作用性

- 实时系统通常运行在一定的环境下，外部环境是实时系统不可缺少的一个组成部分。
- 计算机子系统一般是控制系统，它必须在规定的时间内对外部请求做出反应。外部物理环境往往是被控子系统，两者互相作用构成完整的实时系统。

实时系统与非实时系统

非实时操作系统时 - 代表产品 Windows

- Windows不是一个实时操作系统，并不是Windows不够快或效率不够高，而是因为它不能提供确定性，所以，Windows不是一个实时操作系统。

硬实时 - 代表产品 VxWorks

- 硬实时系统指系统要有确保的最坏情况下的服务时间，即对于事件的响应时间的截止期限是无论如何都必须得到满足。

软实时 - 代表产品 软实时Linux

- 软实时系统就是那些从统计的角度来说，一个任务能够得到有确保的处理时间，到达系统的事件也能够在截止期限到来之前得到处理，但违反截止期限并不会带来致命的错误。

嵌入式系统的几个重要特征

（1）系统内核小

- 由于嵌入式系统一般是应用于小型电子装置的，系统资源相对有限，所以内核较之传统的操作系统要小得多。
- 比如ENEA公司的OSE分布式系统，内核只有5K，而Windows的内核则要大得多。

嵌入式系统的几个重要特征

（2）专用性强

- 嵌入式系统的个性化很强，其中的软件系统和硬件的结合非常紧密，一般要针对硬件进行系统的移植。
- 即使在同一品牌、同一系列的产品中也需要根据系统硬件的变化和增减不断进行修改。
- 同时针对不同的任务，往往需要对系统进行较大更改，程序的编译下载要和系统相结合，这种修改和通用软件的“升级”是完全不同的概念。

嵌入式系统的几个重要特征

（3）系统精简

- 嵌入式系统一般没有系统软件和应用软件的明显区分，不要求其功能设计及实现上过于复杂，这样一方面利于控制系统成本，同时也利于实现系统安全。

嵌入式系统的几个重要特征

（4）高实时性OS

- 这是嵌入式软件的基本要求，而且软件要求固态存储，以提高速度。软件代码要求高质量和高可靠性、实时性。

嵌入式系统的几个重要特征

（5）嵌入式软件开发走向标准化

- 嵌入式系统的应用程序可以没有操作系统直接在芯片上运行。
- 为了合理地调度多任务、利用系统资源、系统函数以及和专家库函数接口，用户必须自行选配RTOS（Real—Time Operating System）开发平台，这样才能保证程序执行的实时性、可靠性，并减少开发时间，保障软件质量。

嵌入式系统的几个重要特征

（6）嵌入式系统需要开发工具和环境

- 由于其本身不具备自主开发能力，即使设计完成以后，用户通常也是不能对其中的程序功能进行修改，必须有一套开发工具和环境才能进行开发。
- 这些工具和环境一般是基于通用计算机上的软硬件设备以及各种逻辑分析仪、混合信号示波器等。
- 开发时往往有主机和目标机的概念，主机用于程序的开发，目标机作为最后的执行机，开发时需要交替结合进行。

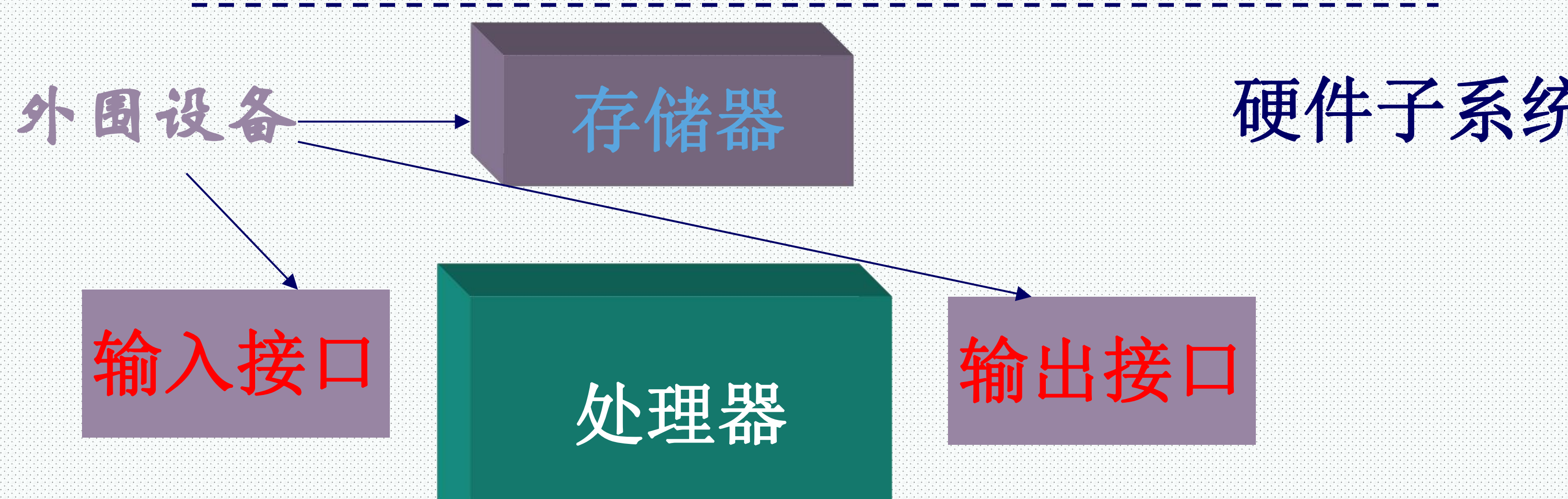
嵌入式系统简介

嵌入式系统组成

应用程序

操作系统(驱动程序)

软件子系统



嵌入式处理器

嵌入式系统的核心是嵌入式微处理器。嵌入式微处理器一般就具备以下4个特点

- 对实时多任务有很强的支持能力。能完成多任务并且有较短的中断响应时间，从而使内部的代码和实时内核的执行时间减少到最低限度。
- 具有功能很强的存储区保护功能。这是由于嵌入式系统的软件结构已模块化，而为了避免在软件模块之间出现错误的交叉作用，需要设计强大的存储区保护功能，同时也有利于软件诊断。
- 可扩展的处理器结构。以能最迅速地开发出满足应用的最高性能的嵌入式微处理器。
- 嵌入式微处理器必须功耗很低。尤其是用于便携式的无线及移动的计算和通信设备中靠电池供电的嵌入式系统更是如此，如需要功耗只有mW甚至μW级。

嵌入式微处理器分类

嵌入式处理器

- 嵌入式微控制器（MCU）
- 嵌入式DSP处理器(DSP)
- 嵌入式微处理器(MPU)
- 嵌入式片上系统(System On Chip)

RTOS是32位嵌入式CPU的软件基础

- RTOS内核 提供CPU的管理
- RTOS内核提供任务，内存管理
- RTOS提供设备管理，文件和网络的支持
- RTOS提供C/C++，JAVA，图形模块等编程接口

常见的嵌入式操作系统

实时嵌入式操作系统的种类繁多，大体上可分为两种，商用型和免费型。

- 商用型的实操作系统功能稳定、可靠，有完善的技术支持和售后服务，但往往价格昂贵--VxWorks 。
- 免费型的实时操作系统在价格方面具有优势，目前主要有Linux，但稳定性与服务性存在挑战。

VxWorks

- VxWorks操作系统是美国WindRiver公司于1983年设计开发的一种嵌入式实时操作系统（RTOS），具有良好的持续发展能力、高性能的内核以及友好的用户开发环境，在嵌入式实时操作系统领域牢牢占据着一席之地。

- VxWorks所具有的显著特点是:
  - 可靠性、实时性和可裁减性。
  - 它支持多种处理器，如x86、i960、Sun Sparc、Motorola MC68xxx、MIPS 、POWER PC等等。

- 大多数的VxWorks API是专有的，火星机器人。

Windows Embedded

- Windows CE 3.0：一种针对小容量、移动式、智能化、32位、连接设备的模块化实时嵌入式操作系统。
- 针对掌上设备、无线设备的动态应用程序和服务提供了一种功能丰富的操作系统平台， WindowsCE嵌入但不够实时，属于软实时操作系统，
- 由于其Windows背景，界面比较统一认可。
- 操作系统的基本内核需要至少200K的ROM。

Palm OS

- Palm OS是著名的网络设备制造商3COM旗下的Palm Computing掌上电脑公司的产品。
- 3COM、CISCO竞争
- Palm OS在PDA市场上占有很大的市场份额， Palm OS的市场份额占到将近90%，最近下降70％，目前主要与WIN CE进行激烈竞争。

嵌入式Linux

- 嵌入式系统越来越追求数字化、网络化和智能化。因此原来在某些设备或领域中占主导地位的软件系统越来越难以为继，整个系统必须是开放的、提供标准的API，并且能够方便地与众多第三方的软硬件沟通。

- Linux是开放源码的，不存在黑箱技术，遍布全球的众多Linux爱好者又是Linux开发的强大技术后盾。

- Linux的内核小、功能强大、运行稳定、系统健壮、效率高，易于定制剪裁，在价格上极具竞争力。

- Linux不仅支持x86 CPU，还可以支持其他数十种CPU芯片。

嵌入式Linux及应用

- 嵌入式Linux(Embedded Linux)是指对Linux经过小型化裁剪后，能够固化在容量只有几百K字节或几兆字节的存储器芯片或单片机中，应用于特定嵌入式场合的专用Linux操作系统。嵌入式Linux的开发和研究是目前操作系统领域的一个热点。主要有RTLinux和μCLinux

- Linux在嵌入式领域异军突起不过是近两年的事情，过去的一年中有13%的用户已经开始使用嵌入式Linux系统进行开发工作；有52%的用户决定在未来24个月内开始使用Linux作为嵌入式操作系统的开发原型。

μC/OS及μC/OS-II

1、μC/OS——Micro Controller O S，微控制器操作系统

2、 μC/OS简介

- 美国人Jean Labrosse 1992年完成
- 应用面覆盖了诸多领域，如照相机、医疗器械、音响设备、发动机控制、高速公路电话系统、自动提款机等
- 1998年μC/OS-II，目前的版本μC/OS -II V2.61，2.72
- 2000年，得到美国航空管理局（FAA）的认证，可以用于飞行器中
- 网站www.ucos-II.com（www.micrium.com)

Android

- Android一词的本义指“机器人”，同时也是Google于2007年11月5日宣布的基于Linux平台的开源手机操作系统的名称，该平台由操作系统、中间件、用户界面和应用软件组成，号称是首个为移动终端打造的真正开放和完整的移动软件。

嵌入式系统的软/硬件框架

<table>
<tr><td>功能层</td><td colspan="3">应用程序</td></tr>
<tr><td rowspan="2">软件层</td><td>文件系统</td><td>图形用户接口</td><td>任务管理</td></tr>
<tr><td colspan="3">实时操作系统(RTOS)</td></tr>
<tr><td>中间层</td><td colspan="3">BSP/HAL 硬件抽象层/板极支持包</td></tr>
<tr><td rowspan="4">硬件层</td><td>D/A</td><td rowspan="3">嵌入式微处理器</td><td>通用接口</td></tr>
<tr><td>A/D</td><td>ROM</td></tr>
<tr><td>I/O</td><td>SDRAM</td></tr>
<tr><td colspan="3">人机交互接口</td></tr>
</table>

嵌入式计算机系统

嵌入式系统简介

❖ 最小硬件系统

时钟系统

调试测试接口

可选，但是在样品阶段通常都会设计这部分电路

嵌入式控制器

供电系统（电源）

复位及其配置系统

存储器系统

可选，因为许多面向嵌入式领域的微控制器内部集成了程序和数据存储器

嵌入式系统软件体系

Hardware Independent Software

Applications

I/O System

RTOS libraries

TCP/IP Stack

File System

RTOS Kernel

Hardware Abstraction Layer: BSP & Device Driver

SCSI Driver

Flash Driver

MMU Driver

Cache Driver

Serial Driver

Ethernet Driver

Device Drivers

Hardware

嵌入式操作系统

特点：

- 可装卸性。开放性、可伸缩性的体系结构。
- 强实时性。EOS实时性一般较强，可用于各种设备控制当中。
- 统一的接口。提供各种设备驱动接口。
- 操作方便、简单、提供友好的图形GUI，图形界面，追求易学易用。
- 提供强大的网络功能，支持TCP/IP协议及其它协议，提供TCP/UDP/IP/PPP协议支持及统一的MAC访问层接口，为各种移动计算设备预留接口。
- 强稳定性，弱交互性。嵌入式系统一旦开始运行就不需要用户过多的干预，这就要负责系统管理的EOS具有较强的稳定性。嵌入式操作系统的用户接口一般不提供操作命令，它通过系统的调用命令向用户程序提供服务。
- 固化代码。在嵌入式系统中，嵌入式操作系统和应用软件被固化在嵌入式系统计算机的ROM中。辅助存储器在嵌入式系统中很少使用，因此，嵌入式操作系统的文件管理功能应该能够很容易地拆卸，而用各种内存文件系统。
- 更好的硬件适应性，也就是良好的移植性。

硬件抽象层

- 硬件层和软件层之间为中间层，也称为硬件抽象层（Hardware Abstract Layer，HAL）或板级支持包(Board Support Package，BSP)。它将系统上层软件与底层硬件分离开来，使得系统的底层驱动程序与硬件无关，上层软件开发人员无须关心底层硬件的具体情况，根据BSP层提供的接口即可进行开发。该层一般包含相关底层硬件的初始化、数据的输入/输出操作和硬件设备的配置等功能。BSP具有以下两个特点：
  - 硬件相关性：因为嵌入式实时系统的硬件环境具有应用相关性，而作为上层软件与硬件平台之间的接口，BSP需要为操作系统提供操作和控制具体硬件的方法。
  - 操作系统相关性：不同的操作系统具有各自的软件层次结构，因此，不同的操作系统具有特定的硬件接口形式。

设备驱动程序

- 所谓的设备驱动程序，就是一组库函数，用来对硬件进行初始化和管理，并向上层软件提供良好的访问接口。
- 大多数的设备驱动程序都会具备以下的一些基本功能。
  - 硬件启动：在开机上电或系统重启的时候，对硬件进行初始化；
  - 硬件关闭：将硬件设置为关机状态；
  - 硬件停用：暂停使用这个硬件；
  - 硬件启用：重新启用这个硬件；
  - 读操作：从硬件中读取数据；
  - 写操作：往硬件中写入数据。

嵌入式文件系统

嵌入式文件系统具有以下特点：

- 兼容性：嵌入式文件系统通常支持几种标准的文件物理结构，如FAT32、JFFS2、YAFFS等。
- 实时文件系统：除支持标准的文件物理结构外，为提高实时性，有些嵌入式文件系统还支持自定义的实时文件系统，这些文件系统一般采用连续文件的方式存储文件。
- 可裁减、可配置：可根据嵌入式系统的要求选择所需的文件物理结构，可选择所需的存储介质，配置可同时打开的最大文件数等。
- 支持多种存储设备：嵌入式系统的外存形式多样，嵌入式文件系统需方便的挂接不同存储设备的驱动程序，具有灵活的设备管理能力。同时根据不同外部存储器的特点，嵌入式文件系统还需考虑其性能、寿命等因素，发挥不同外存的优势，提高存储设备的可靠性和使用寿命。

嵌入式图形用户界面

嵌入式系统中的图形界面的几种解决方案：

- 针对特定的图形设备输出接口，自行开发相应的功能函数；
- 购买针对特定嵌入式系统的图形中间软件包；
- 采用源码开放的嵌入式GUI支持系统；
- 使用独立软件开发商提供的嵌入式GUI产品。

ARM处理器的体系结构

ARM（Advanced RISC Machines）是一个32位元精简指令集(RISC)处理器架构，ARM处理器包含以下几个系列的处理器产品以及其他厂商实现的基于ARM体系结构的处理器。

ARM7系列、ARM9系列、ARM9E系列、ARM10E系列、SecurCore系列、Intel的Xscale、Intel的StrongARM。

这些处理器广泛应用于以下应用领域：

- 开放应用平台。包括无线平台、消费产品以及成像设备等
- 实时嵌入式应用。包括存储设备、汽车、工业和网络设备
- 安全系统。包括信用卡和SIM卡等

ARM处理器的体系结构

- 体系结构-CISC与RISC

  CISC（Complex Instruction Set Computer，复杂指令集计算机）

在CISC指令集的各种指令中，大约有20%的指令会被反复使用，占整个程序代码的80%。而余下的80%的指令却不经常使用，在程序设计中只占20%。

  RISC（Reduced Instruction Set Computer，精简指令集计算机）

RISC结构优先选取使用频最高的简单指令，避免复杂指令；将指令长度固定，指令格式和寻地方式种类减少；以控制逻辑为主，不用或少用微码控制等

ARM处理器的体系结构

RISC体系结构特点：

※采用固定长度的指令格式，指令归整、简单、基本寻址方式有2～3种。

※使用单周期指令，便于流水线操作执行。

※大量使用寄存器，数据处理指令只对寄存器进行操作，只有加载/　存储指令可以访问存储器，以提高指令的执行效率。除此以外，ARM体系结构还采用了一些特别的技术，在保证高性能的前提下尽量缩小芯片的面积，并降低功耗：

※所有的指令都可根据前面的执行结果决定是否被执行，从而提高指令的执行效率。

※可用加载/存储指令批量传输数据，以提高数据的传输效率。

※可在一条数据处理指令中同时完成逻辑处理和移位处理。

ARM处理器的体系结构

- ARM处理器工作模式

  除用户模式外，其余6种模式称为非用户模式或特权模式，在这些模式下，程序可以访问所有的系统资源，也可以任意地进行处理器模式的切换。其中，除系统模式外，其他5种特权模式又称为异常模式。

  处理器模式可以通过软件控制进行切换，也可以通过外部中断或异常处理过程进行切换。大多数的用户程序运行在用户模式下。这时，应用程序不能够访问一些受操作系统保护的系统资源。应用程序也不能直接进行处理器模式的切换。当需要进行处理器模式切换时，应用系统可以产生异常处理，在异常处理过程中进行处理器模式的切换。这种体系结构可以使操作系统控制整个系统的资源。

ARM处理器的体系结构

❖ ARM处理器工作模式

处理器模式	描述
用户模式（User，usr）	正常程序执行的模式
快速中断模式（FIQ，fiq）	用于高速数据传输和通道处理
外部中断模式（IRQ，irq）	用于通常的中断处理
特权模式（Supervisor，sve）	供操作系统使用的一种保护模式
数据访问中止模式（Abort，abt）	用于虚拟存储及存储保护
未定义指令中止模式（Undefined，und）	用于支持通过软件仿真硬件的协处理器
系统模式（System，sys）	用于运行特权级的操作系统任务

ARM处理器的体系结构

❖ 寄存器结构

ARM处理器共有37个寄存器，被分为若干个组（BANK），这些寄存器包括：

- 31个通用寄存器，包括程序计数器（PC指针），这些寄存器均为32位的寄存器。
- 6个状态寄存器，用以标识CPU的工作状态及程序的运行状态，这些寄存器都是32位寄存器，但目前只使用了其中12位。
- ARM处理器共有7种不同的处理器模式，每一种处理器模式中都有一组响应的寄存器组。在任意时刻（任意的处理器模式下），可见的寄存器组包括16个通用寄存器（R0～R14、程序计数器PC）、一个或两个状态寄存器CPSR。

ARM处理器的体系结构

ARM各种模式的寄存器

在所有寄存器中，有些是各种模式下共用的同一个物理寄存器，有些是各种模式自己独立拥有的寄存器。

用户模式	系统模式	特权模式	中止模式	未定义指令模式	外部中断模式	快速中断模式
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

ARM处理器的体系结构

通用寄存器

通用寄存器可以分为以下3类：

- 未备份寄存器，包括R0~R7；
- 备份寄存器，包括R8~R14；
- 程序计数器，R15。

ARM处理器的体系结构

❖ 未备份寄存器

未备份寄存器包括R0~R7，在所有的处理器模式下都使用同一个物理寄存器；在异常中断造成处理器模式切换时，由于不同的处理器模式使用相同的物理寄存器，可能造成寄存器中数据被破坏；未备份寄存器没有被系统用于特别的用途，任何可采用通用寄存器的应用场合都可以使用未备份寄存器。

ARM处理器的体系结构

备份寄存器

对于备份寄存器R8~R12来说，每个寄存器对应两个不同的物理寄存器。系统未将备份寄存器用于任何的特殊用途，但当中断处理非常简单，仅仅使用R8～R14寄存器时，FIQ处理程序可以不必执行保存和恢复中断现场的指令，从而可以使中断处理过程很迅速。

对于备份寄存器R13和R14来说，每个寄存器对应于6个不同的物理寄存器，其中的一个是用户模式和系统模式共同的，另外的5个异常模式对应于其他5种处理处理器模式。

ARM处理器的体系结构

程序计数器R15

程序计数器R15又被称为PC，它虽然可以作为一般的通用寄存器使用，但是有一些指令在使用R15时有一些特殊限制。当违反了这些限制，该指令执行的结果将是不可预料的。当顺序执行程序时，PC指针指向下一条指令地址；成功向PC指针写入一个地址数值后，程序将跳转到该地址执行。

ARM处理器的体系结构

程序状态寄存器

程序状态寄存器包含：

- 1个CPSR（当前程序状态寄存器）
- 5个SPSR（备份程序状态寄存器）

ARM处理器的体系结构

指令结构

ARM微处理器的在较新的体系结构中支持两种指令集：ARM指令集和Thumb指令集。其中，ARM指令为32位的长度，Thumb指令为16位长度。Thumb指令集为ARM指令集的功能子集，但与等价的ARM代码相比较，可节省30%～40%以上的存储空间，同时具备32位代码的所有优点。

ARM处理器的体系结构

- **ARM体系程序的执行流程**

  在ARM体系中，通常有以下3种方式控制程序的执行流程。

  - 正常程序执行，每执行一条ARM指令，程序计数寄存器（PC）的值加4个字节；每执行一条Thumb指令，程序计数寄存器（PC）的值加两个字节。整个过程按顺序执行。
  - 跳转指令执行，通过跳转指令，程序可以跳转到特定的地址标号处执行，或者跳转到特定的子程序处执行。其中，B指令用于执行跳转操作；BL指令在执行跳转指令的同时，保存子程序的返回地址；BX指令在执行跳转指令的同时，根据目标地址的最低位将程序状态切换到Thumb状态；BLX指令执行3个操作，跳转到目标地址处执行，保存子程序的返回地址，根据目标地址的最低位可以将程序状态切换到Thumb状态。
  - 异常中断发生，当异常中断发生时，系统执行完当前指令后，将跳转到相应的异常中断处理程序处执行。在异常中断处理程序执行完成后，程序返回到发生中断的指令的下一条指令处执行。在进入异常中断处理程序时，要保存被中断的程序的执行现场，在从异常中断处理程序退出时，要恢复被中断的程序的执行现场。

ARM处理器的体系结构

ARM体系程序的异常中断种类

异常中断名称	含 义
复位（Reset）	当处理器复位引脚有效时，系统产生复位，程序跳转到复位异常中断处理程序处执行，复位异常中断的优先级是最高优先级的中断。通常复位产生有下面几种情况：系统加电时、系统复位时、各种不同的ARM处理器的复位有一些区别的，具体的参见后面的实例中的描述
未定义的指令 Undefined instruction	当ARM处理器或者系统中的协处理器认为当前指令未定义时，产生该中断，可以通过该异常中断仿真浮点向量运算
软件中断 Software Interrupt SWI	这是由用户定义的中断指令，可用于用户模式下的程序调用特权操作指令
数据访问中止 Data Abort	数据访问指令的目标地址不存在，或者该地址不允许当前指令访问，处理器产生数据访问中止异常中断
外部中断请求 IRQ	当处理器的外部中断请求引脚有效，或者CPSR寄存器的I控制位被清除时，处理器产生外部中断请求,应用中对于IRQ的中断处理是比较关键的技术
快速中断请求 FIQ	当处理器的外部中断请求引脚有效，或者CPSR寄存器的F控制位被清楚时，处理器产生外部中断请求

ARM处理器的体系结构

ARM体系的存储空间

ARM体系使用单一的地址空间。

该地址空间的大小为2^32个8位字节，即为4GB大小。这些字节单元的地址都是无符号的32位数值。

ARM处理器的体系结构

ARM处理器命名

架构	处理器家族
ARMv1	ARM1
ARMv2	ARM2、ARM3
ARMv3	ARM6、ARM7
ARMv4	StrongARM、ARM7TDMI、ARM9TDMI
ARMv5	ARM7EJ、ARM9E、ARM10E、XScale
ARMv6	ARM11、ARM Cortex-M
ARMv7	ARM Cortex-A、ARM Cortex-M、ARM Cortex-R
ARMv8	Cortex-A50

ARM处理器的体系结构

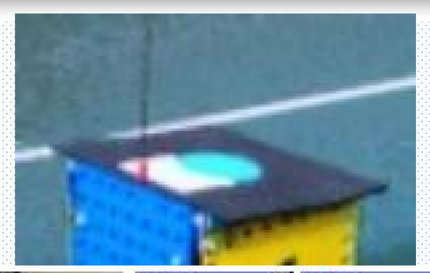
ARM处理器命名

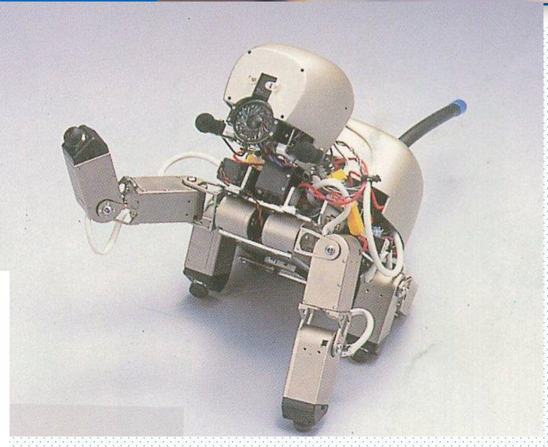
<table>
<tr><th>在 ARMv3 ~ ARMv6 时期
ARM{x}{y}{z}{T}{D}{M}{I}{E}{J}{F}{-S}</th><th>在 ARMv7 以后时期</th></tr>
<tr><td>·x -- 处理器系列
·y -- 存储管理/保护单元
·z -- cache
·T -- 支持Thumb指令集
·D -- 支持片上调试
·M -- 支持快速乘法器
·I -- 支持Embedded ICE，支持嵌入式跟踪调试
·E -- 支持增强型DSP指令
·J -- 支持Jazelle
·F -- 具备向量浮点单元VFP
·-S -- 可综合版本</td><td>公司改革了以前的冗长的命名方法，用看起来比较整齐的办法，统一用 Cortex 作为主名。</td></tr>
</table>

提　　要

1. 嵌入式系统的发展及应用领域
2. 嵌入式系统的定义与体系结构
3. 嵌入式系统的应用案例

嵌入式控制是智能机器人系统的核心





日本SONY机器人

AIBO机器狗

SDR-4X

机器人智能水平的进化历程

Processing Power (MIPS)

Prediction by H. Moravec

Human

Monkey

Mouse

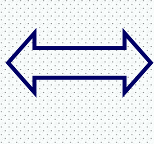
Lizard

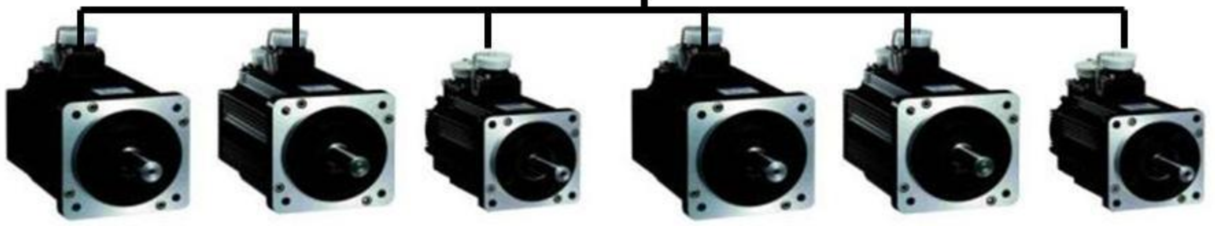
Target of RoboCup

10^{10} 10^{8} 10^{6} 10^{4} 10^{2}

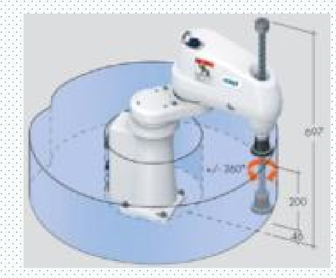
1990 2000 2010 2020 2030 2040 2050

工业机器人控制系统整体解决方案

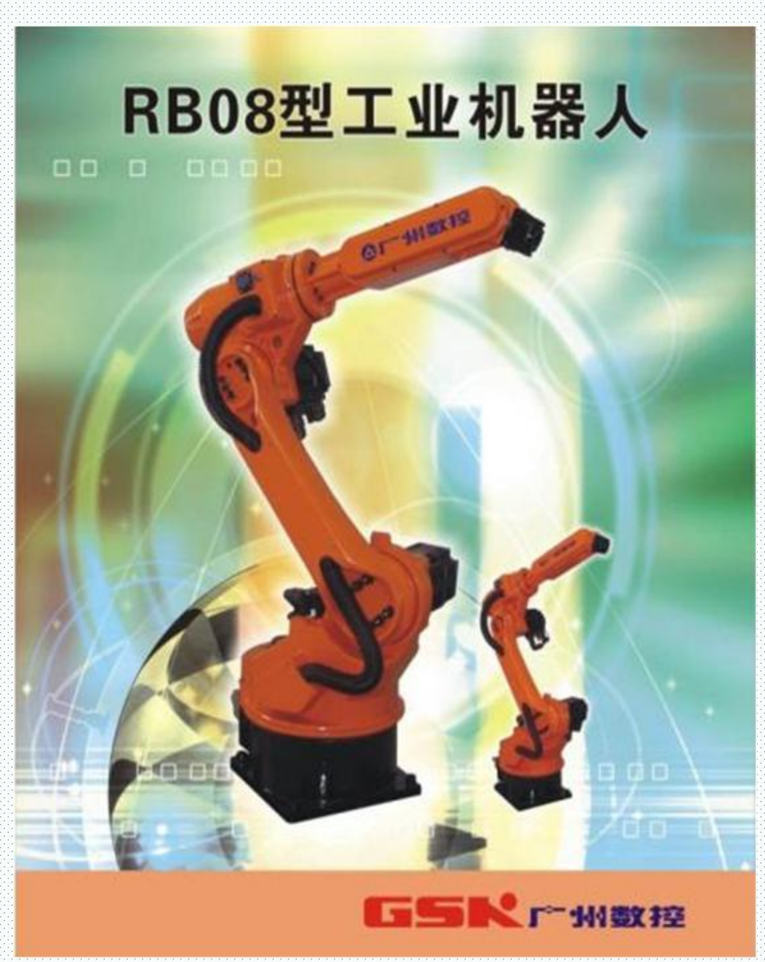




采用PXA270+uCOS操作系统，自主开发了工业机器人（4-8轴）控制器、示教器软硬件系统，产品已在奇瑞工业机器人、广数工业机器人推广50台套以上。



工业机器人控制系统应用情况



广数6自由度上下料机器人

奇瑞165Kg点焊机器人及成线装备

模块化机器人平台

软件仿真

示教与再现

关节模块

模块化机械臂

组装

ROS

RGMP

Linux

Nuttx

运行

组装

X86双核控制器+CAN卡

CAN总线

作业：

1. 现阶段的市场占有率高的嵌入式处理器有哪些？请列举一二并简要说明其性能特点。

2.嵌入式处理器和通用处理器有哪些区别?你觉得嵌入式处理器最终会取代通用处理器么？

注意：以上题目要用自己的话简略回答即可，不要大段复制网上的内容。

■ HAL库入门

适用平台

- STM32F4xx 开发板（正点原子）

目 录

1 固件库和寄存器区别

2 STM32CubeF4/HAL库包介绍

固件库和寄存器区别和联系

1 固件库和寄存器区别

固件库和寄存器区别

◆ 固件库和寄存器的区别?

```
GPIOF->BSRR=0x00000001;
```

```
void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin,
                                      GPIO_PinState PinState)
{
  if(PinState != GPIO_PIN_RESET)
  {
      GPIOx->BSRR = GPIO_Pin;
  }
  else
  {
      GPIOx->BSRR = (uint32_t)GPIO_Pin << 16;
  }
}
```

固件库和寄存器区别

固件库是什么?

固件库就是函数的集合（API），把寄存器操作封装起来。

固件库和寄存器区别

固件库作用是什么？为什么需要固件库？

STM32寄存器成百上千，一一操作非常不便。通过API把寄存器操作封装起来，这样大家不需要在记寄存器的每个位，而是直接操作固件库函数。简单方便很多。

固件库和寄存器区别

是不是就不需要对寄存器有任何了解?

固件库不是万能的。要想全面的掌握STM32，必须对寄存器有一定的了解，尤其是入门学习的时候。只有通过对寄存器有一个基本的了解，才能全面掌握了STM32各个功能外设的工作原理，才能更好的使用固件库。

对于寄存器，大家不需要去死记硬背寄存器名称以及每个位作用，大家只需要大致的了解大致的配置过程，这样在开发中遇到问题，就可以通过调试直接查看寄存器配置，从而找出问题所在。

固件库和寄存器区别

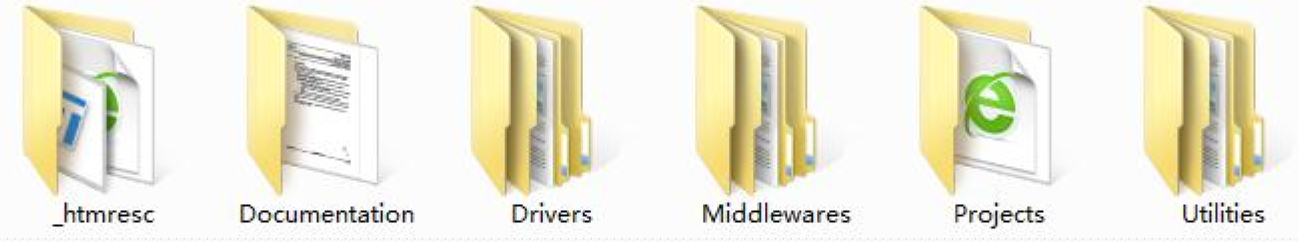
在使用固件库开发的工程可以操作寄存器吗?

当然可以。固件库函数的本质就是直接封装的寄存器操作。所以你单独操作寄存器肯定是可以的了。

HAL库包介绍

2 HAL库包和关键文件介绍

HAL库包介绍

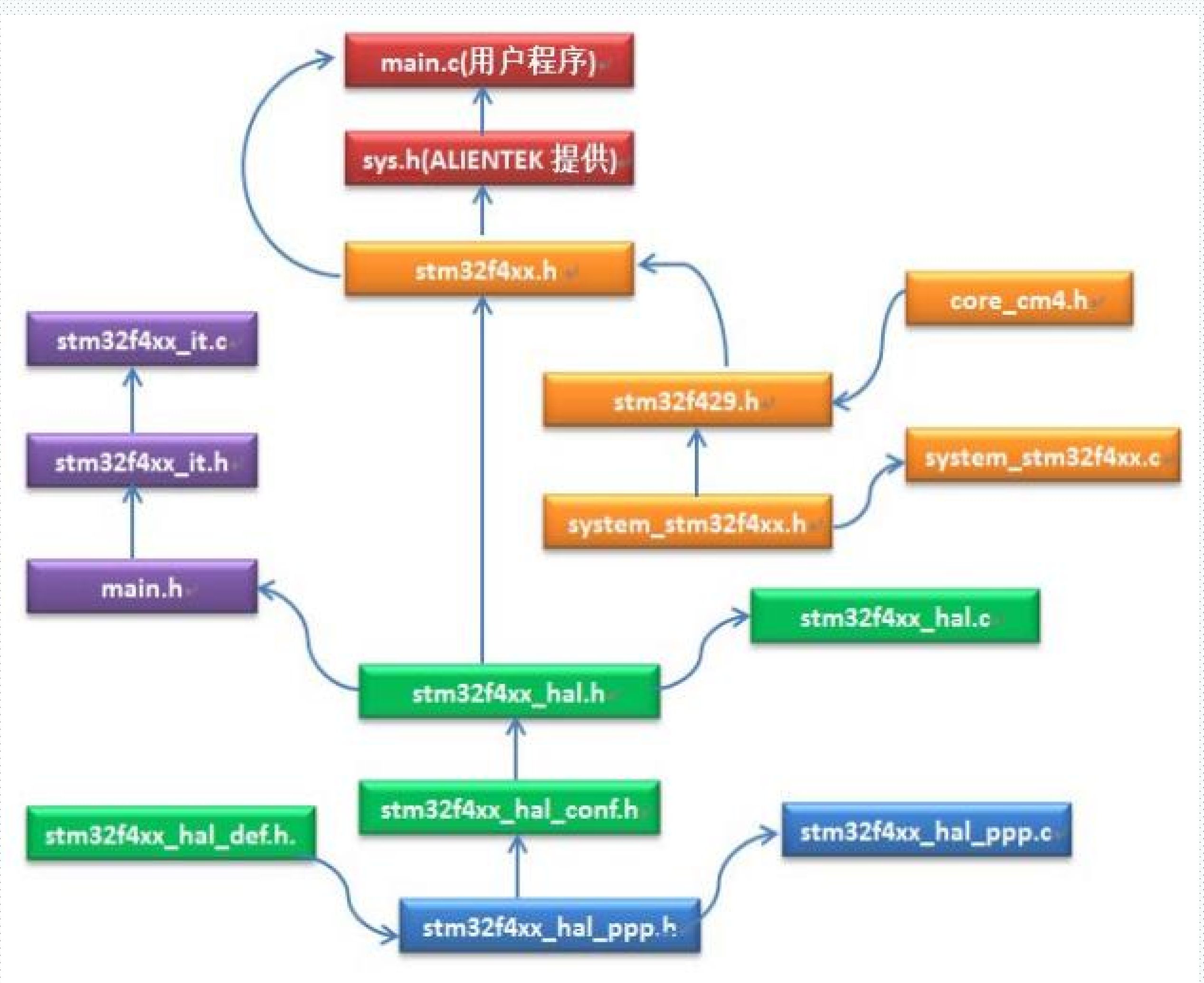


HAL库包介绍

STM32F4xx HAL库文件描述			
类别	文件名	描述	是否必须
启动文件	stm32f429xx.s	启动文件,引导进入systemInit和main函数	是
外设和HAL库相关文件	stm32f4xx.h	顶层头文件，根据芯片型号包含具体芯片头文件	是
	stm32f429xx.h	真正的顶层头文件，外设寄存器定义	是
	system_stm32f4xx.h	主要是存放系统初始化函数SystemInit	是
	system_stm32f4xx.c		是
	sm32f4xx_hal.h	HAL库通用API（比如HAL_Init,HAL_DeInit,HAL_Delay等）	是
	sm32f4xx_hal.c		是
	stm32f4xx_ppp.h	外设HAL库操作API头文件和源文件，每个外设对应一个源文件和头文件。	是
	stm32f4xx_ppp.c		是
	stm32f4xx_hal_ppp_ex.h	拓展的外设API头文件和源文件	是
	stm32f4xx_hal_ppp_ex.c		是
	stm32f4xx_II_ppp.h	在一些复杂外设中实现底层功能，它们在stm32f4xx_hal_ppp.c中被调用	是
	stm32f4xx_II_ppp.c		是
	stm32f4xx_hal_conf.h	外设头文件引入，以及一些以太网和时钟相关常量定义	是
内核头文件	core_cm4.h	主要是内核寄存器定义，例如Systick，SCB	是
	core_cmFunc.h	CMSIS定义内核操作，一般不需要我们去了解	是
	core_cmInstr.h		
	core_cmSimd.h		
	cmsis_armcc.h		
用户程序文件	main.c	存放main函数，不一定要放这个文件	否
	stm32f4xx_it.h	用户中断服务函数存放位置（不一定放这里）	否
	stm32f4xx_it.c		
	stm32f4xx_hal_msp.c	回调函数存放文件	否

HAL库包介绍

关键文件关系图

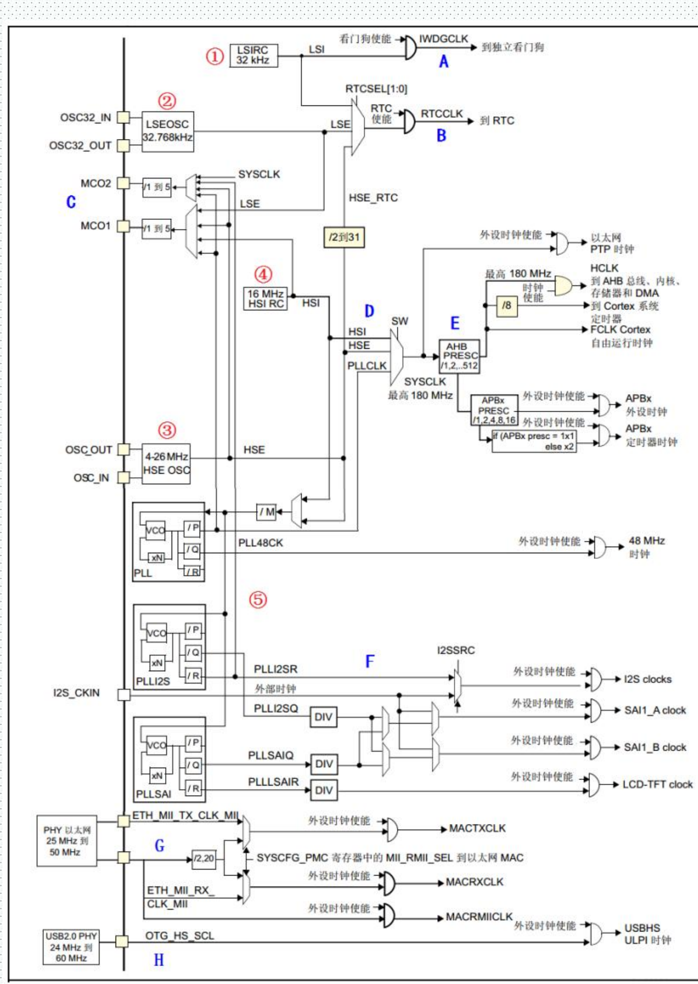


■ 时钟系统讲解

适用平台

✓ STM32F429开发板
（正点原子）

时钟系统框图



- ◆时钟树比较复杂

时钟系统框图

①
LSIRC
32 kHz
LSI
看门狗使能
IWDGCLK
到独立看门狗
A
RTCSEL[1:0]
②
OSC32_IN
OSC32_OUT
LSEOSC
32.768kHz
LSE
RTC
使能
RTCCLK
到 RTC
B
MCO2
C
MCO1
/1 到 5
/1 到 5
SYSCLK
LSE
HSE_RTC
/2到31
④
16 MHz
HSI RC
HSI
外设时钟使能
以太网
PTP 时钟
最高 180 MHz
HCLK
到 AHB 总线、内核、
存储器和 DMA
时钟
使能
/8
到 Cortex 系统
定时器
FCLK Cortex
自由运行时钟
D
SW
E
HSI
HSE
PLLCLK
AHB
PRESC
/1,2,..512
SYSCLK
最高 180 MHz
APBx
PRESC
/1,2,4,8,16
外设时钟使能
APBx
外设时钟
外设时钟使能
if (APBx presc = 1x1
else x2
APBx
定时器时钟
③
OSC_OUT
OSC_IN
4-26 MHz
HSE OSC
HSE
/ M
VCO
xN
/ P
/ Q
/ R
PLL
PLL48CK
外设时钟使能
48 MHz
时钟

时钟系统框图

时钟系统讲解

1. STM32 有5个时钟源:HSI、HSE、LSI、LSE、PLL。

①、HSI是高速内部时钟，RC振荡器，频率为16MHz，精度不高。可以直接作为系统时钟或者用作PLL时钟输入。

②、HSE是高速外部时钟，可接石英/陶瓷谐振器，或者接外部时钟源，频率范围为4MHz~26MHz。

③、LSI是低速内部时钟，RC振荡器，频率为32kHz，提供低功耗时钟。主要供独立看　门狗和自动唤醒单元使用。

④、LSE是低速外部时钟，接频率为32.768kHz的石英晶体。RTC

⑤、PLL为锁相环倍频输出。

时钟系统讲解

- **PLL为锁相环倍频输出。STM32F4有三个PLL:**
  - 主PLL(PLL)由HSE或者HSI提供时钟信号，并具有两个不同的输出时钟。
    ①第一个输出PLLP用于生成高速的系统时钟（最高180MHz）
    ②第二个输出PLLQ为48M时钟，用于USB OTG FS时钟，随机数发生器的时钟和SDIO时钟。
  - 第一个专用PLL(PLLI2S)生成精确时钟，在I2S和SAI1上实现高品质音频
    N是用于PLLI2S vco的倍频系数，其取值范围是：192~432；
    R是I2S时钟的分频系数，其取值范围是：2~7；
    Q是SAI时钟分频系数，其取值范围是：2~15；P没用到。
  - 第二个专用PLL(PLLSAI)同样用于生成精确时钟，用于SAI1输入时钟，同时还为LCD_TFT接口提供精确时钟。
    N是用于PLLSAI vco的倍频系数，其取值范围是：192~432；
    Q是SAI时钟分频系数，其取值范围是：2~15；
    R是LTDC时钟的分频系数，其取值范围是：2~7；P没用到。

时钟系统讲解

■ **主PLL时钟计算**

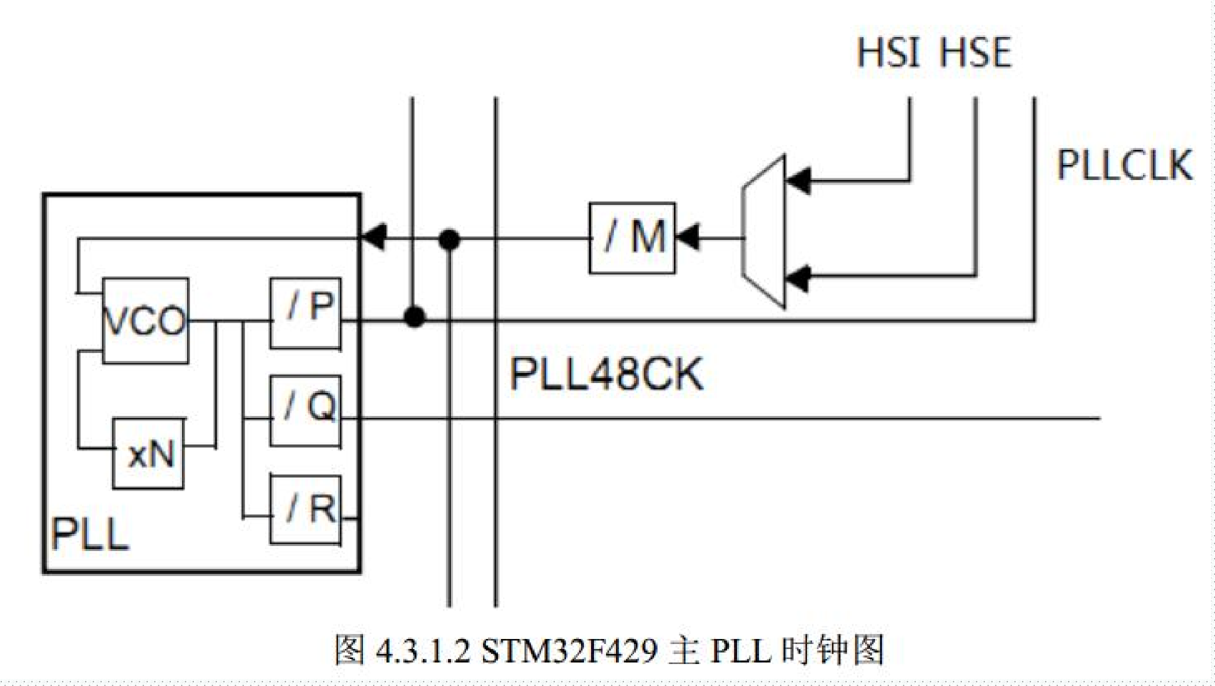


图 4.3.1.2 STM32F429 主 PLL 时钟图

PLLCLK= HSE * N / (M*P)

时钟系统讲解

2. 系统时钟SYSCLK可来源于三个时钟源：

①、HSI振荡器时钟

②、HSE振荡器时钟

③、PLL时钟

时钟系统讲解

3.STM32F4时钟信号输出MCO1（PA8）和MCO2(PC9)。

- MCO1:用户可以配置预分频器（1~5）向MCO1引脚PA8输出4个不同的时钟源：
  - ✓ HIS
  - ✓ LSE
  - ✓ HSE
  - ✓ PLL
- MCO2:用户可以配置预分频器（1~5）向MCO2引脚PC9输出4个不同的时钟源：
  - ✓ HSE
  - ✓ PLL
  - ✓ SYSCLK
  - ✓ PLLI2S

MCO最大输出时钟不超过100MHz

4.任何一个外设在使用之前，必须首先使能其相应的时钟。

时钟系统讲解

4.任何一个外设在使用之前，必须首先使能其相应的时钟。

时钟系统讲解

◆ **RCC时钟控制相关寄存器定义在 stm32f429xx.h中。**

结构体： RCC_TypeDef;

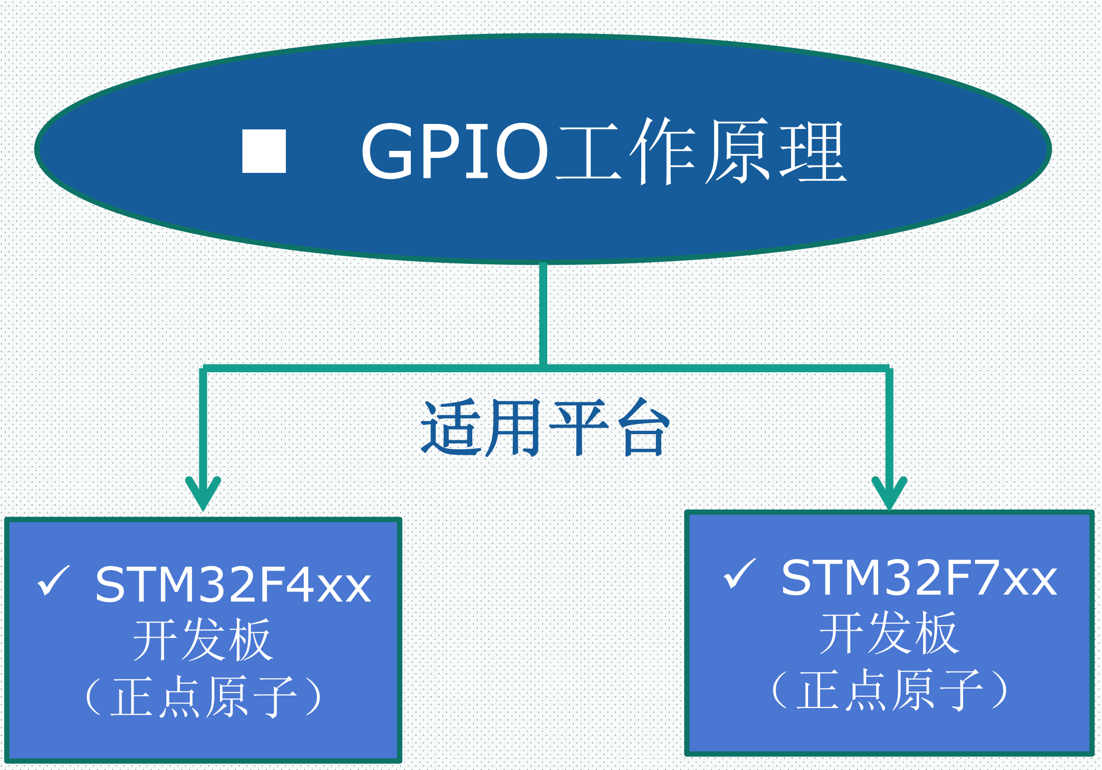
RCC时钟相关定义和函数在文件
stm32f4xx_hal_rcc.h
stm32f4xx_hal_rcc.c

```
typedef struct
{
  __IO uint32_t CR;
  __IO uint32_t PLLCFGR;
  __IO uint32_t CFGR;
  __IO uint32_t CIR;
  __IO uint32_t AHB1RSTR;
  __IO uint32_t AHB2RSTR;
  __IO uint32_t AHB3RSTR;
  uint32_t      RESERVED0;
  __IO uint32_t APB1RSTR;
  __IO uint32_t APB2RSTR;
  uint32_t      RESERVED1[2];
  __IO uint32_t AHB1ENR;
  __IO uint32_t AHB2ENR;
  __IO uint32_t AHB3ENR;
  uint32_t      RESERVED2;
  __IO uint32_t APB1ENR;
  __IO uint32_t APB2ENR;
  uint32_t      RESERVED3[2];
  __IO uint32_t AHB1LPENR;
  __IO uint32_t AHB2LPENR;
  __IO uint32_t AHB3LPENR;
  uint32_t      RESERVED4;
  __IO uint32_t APB1LPENR;
  __IO uint32_t APB2LPENR;
  uint32_t      RESERVED5[2];
  __IO uint32_t BDCR;
  __IO uint32_t CSR;
  uint32_t      RESERVED6[2];
  __IO uint32_t SSCGR;
  __IO uint32_t PLLI2SCFGR;
  __IO uint32_t PLLSAICFGR;
  __IO uint32_t DCKCFGR;

} RCC_TypeDef;x
```

时钟系统讲解

- 对照《STM32F4中文参考手册》6.3节 P114~170 了解这些寄存器的作用和配置方法。



■ GPIO工作原理
适用平台
✓ STM32F4xx
开发板
（正点原子）
✓ STM32F7xx
开发板
（正点原子）

目录

1 GPIO入门知识

2 GPIO的8种工作模式

3 GPIO寄存器

STM32 GPIO入门知识

1 STM32 GPIO入门知识

STM32 GPIO入门知识

GPIO是什么?

全称： general purpose intput output

通用输入输出端口。

可以做输入也可以做输出。

GPIO端口可通过程序配置成输入或者输出。

STM32 GPIO入门知识

引脚和GPIO的区别和联系?

STM32的引脚中，有部分是做GPIO使用，部分是电源引脚/复位引脚/启动模式引脚/晶振引脚/调试下载引脚。

STM32 GPIO入门知识

STM32 GPIO入门知识

LQFP176

◆STM32FXXXIGT6

①一共有9组IO: PA~PI

②其中PA~PH 每组16个IO

PI只有PI0~PI11

③一共有140个IO口:

16*8+12=140

STM32 GPIO入门知识

◆**绝多数引脚都是GPIO，有限的引脚怎么实现更多的功能?**

STM32的大部分引脚除了当GPIO使用外，还可以复用为外设功能引脚（比如串口）。一个引脚，可以作为IO口，同时也可以作为复用功能外设引脚。这部分知识我们会在后面讲解。本讲主要讲解引脚做IO使用方面的知识。

STM32 GPIO入门知识

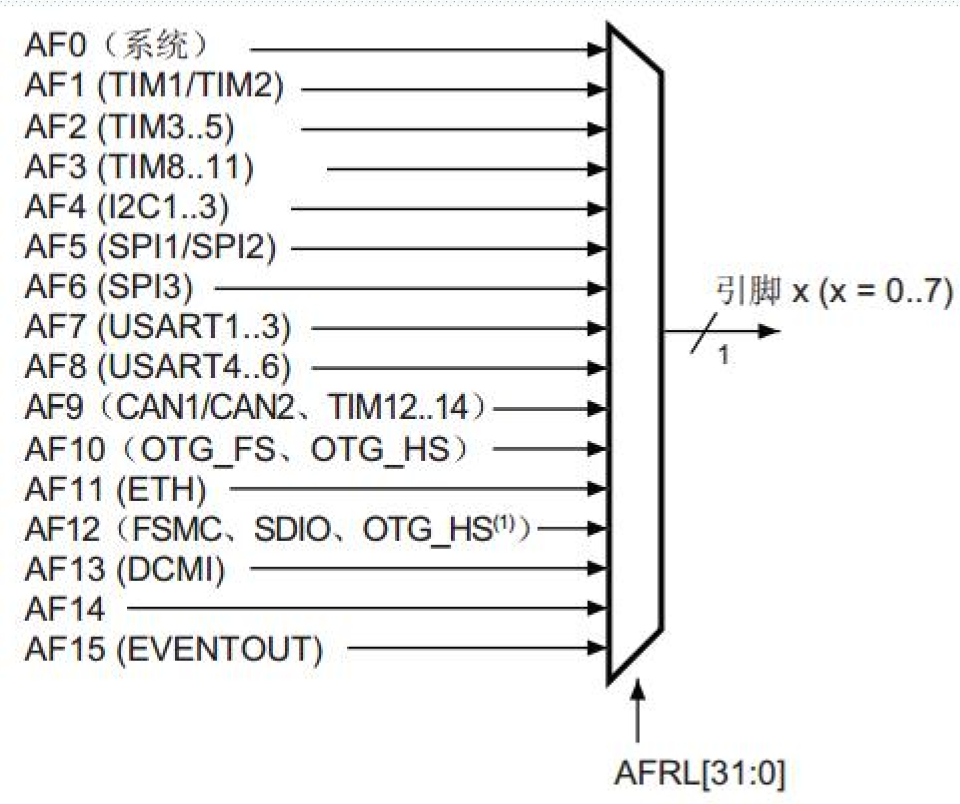
复用原理

没有复用

复用后

STM32 GPIO入门知识

复用原理



STM32 GPIO入门知识

怎么知道具体的芯片外设资源?

STM32 GPIO入门知识

◆怎么查看GPIO引脚功能?

每个STM32芯片的芯片数据手册都会提供引脚功能描述

\7，硬件资料\2，芯片资料\STM32F429IGT6.pdf

\7，硬件资料\2，芯片资料\STM32F767IGT6.pdf

STM32 GPIO入门知识

Pin number							Pin name (function after reset)[1]	Pin type	I / O structure	Notes	Alternate functions	Additional functions
LQFP100	LQFP144	UFBGA176	LQFP176	WLCSP143	LQFP208	TFBGA216						
67	100	F15	119	F1	142	F15	PA8	I/O	FT		MCO1, TIM1_CH1, I2C3_SCL, USART1_CK, OTG_FS_SOF, LCD_R6, EVENTOUT	
68	101	E15	120	E2	143	E15	PA9	I/O	FT		TIM1_CH2, I2C3_SMBA, USART1_TX, DCMI_D0, EVENTOUT	OTG_FS_VBUS
69	102	D15	121	D5	144	D15	PA10	I/O	FT		TIM1_CH3, USART1_RX, OTG_FS_ID, DCMI_D1, EVENTOUT	

✓ 1.3 STM32引脚说明

◆IO口怎么做到可以做输入，输出，也可以复用?

目录

2 GPIO的8种工作模式

GPIO的8种工作模式

- **4种输入模式：**

  输入浮空

  输入上拉

  输入下拉

  模拟输入

- **4种输出模式(带上下拉：**

  开漏输出(带上拉或者下拉)

  开漏复用功能(带上拉或者下拉)

  推挽式输出(带上拉或者下拉)

  推挽式复用功能(带上拉或者下拉)

GPIO的8种工作模式

- **4种最大输出速度：**

  -2MHZ 低速

  -25MHz 中速

  -50MHz 快速

  -100MHz 高速

  在F429参考手册描述可以直接找到速度值，F767的参考手册直接描述为低/中/快/高速，实际上F767的高速是可以达到108MHz。F429和F767输出速度差别不大。

8种工作模式的区别：STM32八种IO口模式区别.pdf

GPIO的8种工作模式

IO口基本结构（F4XX和F7XX一模一样）

到片上外设
模拟
复用功能输入
读
开/关
TTL 施密特触发器
输入数据寄存器
输入驱动器
开/关
写
置位/复位寄存器
输出数据寄存器
输出驱动器
V_{DD}
P-MOS
输出控制
N-MOS
V_{SS}
推挽
开漏或
禁止
开/关
读/写
自片上外设
复用功能输出
模拟
V_{DD}
V_{DD_FT} (1)
拉上
拉下
V_{SS}
保护二极管
I/O 引脚
保护二极管
V_{SS}
FT:5V容忍
ai15939b

✓ 1.1 GPIO基本结构

M3的IO口基本结构

至片上外设
模拟输入
复用功能输入
读出
写入
读/写
来自片上外设
复用功能输出
位设置/清除寄存器
输出数据寄存器
输入数据寄存器
开/关
TTL肖特基触发器
输入驱动器
输出驱动器
V_{DD}
V_{SS}
开/关
开/关
输出控制
P-MOS
N-MOS
推挽、开漏或关闭
保护二极管
保护二极管
I/O 引脚
ai14781

GPIO的8种工作模式

◆ GPIO的输入工作模式1—输入浮空模式

GPIO的8种工作模式

◆ GPIO的输入工作模式2—输入上拉模式

到片上
外设
模拟
复用功能输入
开/关
读
输入数据寄存器
TTL 施密特
触发器
开/关
输入驱动器
V_{DD}
V_{DD_FT} (1)
保护
二极管
I/O
引脚
写
置位/复位寄存器
输出数据寄存器
输出驱动器
V_{DD}
开/关
拉
下
保护
二极管
V_{SS}
V_{SS}
P-MOS
输出控制
N-MOS
读/写
V_{SS}
推挽
开漏或
禁止
自片上
外设
复用功能输出
模拟
ai15939b

GPIO的8种工作模式

◆GPIO的输入工作模式3—输入下拉模式

GPIO的8种工作模式

◆ GPIO的输入工作模式4—模拟模式

到片上
外设
模拟
复用功能输入
读
开/关
输入数据寄存器
TTL 施密特
触发器
V_{DD}
V_{DD_FT} (1)
保护
二极管
写
置位/复位寄存器
输出数据寄存器
开/关
拉上
输入驱动器
I/O
引脚
输出驱动器
V_{DD}
开/关
拉下
保护
二极管
P-MOS
输出控制
V_{SS}
V_{SS}
读/写
N-MOS
V_{SS}
推挽
开漏或
禁止
自片上
外设
复用功能输出
模拟
ai15939b

GPIO的8种工作模式

◆GPIO的输出工作模式1—开漏输出模式

到片上外设
模拟
复用功能输入
开/关
读
输入数据寄存器
TTL 施密特触发器
开/关
输入驱动器
写
置位/复位寄存器
输出数据寄存器
输出驱动器
V_{DD}
开/关
输出控制
P-MOS
N-MOS
V_{SS}
推挽
开漏或
禁止
读/写
自片上外设
复用功能输出
模拟
V_{DD}
V_{DD_FT} (1)
拉上
保护二极管
I/O 引脚
拉下
保护二极管
V_{SS}
V_{SS}
ai15939b

GPIO的8种工作模式

◆GPIO的输出工作模式1—开漏输出模式

开启状态

模拟输入

至片上
外设模块

复用功能输入

V_{DD}

开/关

⑦ 读

⑥

输入数据寄存器

⑤ 开/关

TTL施密特
触发器

开/关

V_{SS}

输入驱动器

V_{DD}

保护
二极管

④

I/O端口

保护
二极管

V_{SS}

写

位设置/清除寄存器

输出数据寄存器

①

输出驱动器 ②

V_{DD}

P-MOS

输出控
制电路

N-MOS

V_{SS}

③

读/写

来自片上
外设模块

复用功能输出

GPIO的8种工作模式

◆GPIO的输出工作模式2—开漏复用输出模式

到片上
外设
模拟
复用功能输入
读
开/关
输入数据寄存器
TTL 施密特
触发器
写
置位/复位寄存器
输出数据寄存器
开/关
输入驱动器
输出驱动器
开/关
V_{DD}
P-MOS
输出控制
N-MOS
V_{SS}
推挽
开漏或
禁止
读/写
自片上
外设
复用功能输出
模拟
V_{DD}
V_{DD_FT} (1)
保护
二极管
拉
上
I/O
引脚
拉
下
保护
二极管
V_{SS}
V_{SS}
ai15939b

GPIO的8种工作模式

◆GPIO的输出工作模式2—开漏复用输出模式

开启状态

至片上外设模块

模拟输入

复用功能输入

输入数据寄存器

开/关

TTL施密特触发器

V_{DD}

开/关

开/关

V_{SS}

输入驱动器

保护二极管

保护二极管

V_{DD}

V_{SS}

I/O端口

读

写

位设置/清除寄存器

输出数据寄存器

读/写

来自片上外设模块

复用功能输出

输出驱动器

输出控制电路

P-MOS

N-MOS

V_{DD}

V_{SS}

①②③④⑤⑥⑦

GPIO的8种工作模式

◆ GPIO的输出工作模式3—推挽输出模式

到片上
外设
模拟
复用功能输入
读
开/关
TTL 施密特
触发器
输入数据寄存器
输入驱动器
写
置位/复位寄存器
输出数据寄存器
读/写
自片上
外设
复用功能输出
输出驱动器
输出控制
V_{DD}
P-MOS
N-MOS
V_{SS}
推挽
开漏或
禁止
开/关
V_{DD}
V_{DD_FT} (1)
拉上
拉下
保护
二极管
I/O
引脚
V_{SS}
模拟
ai15939b

GPIO的8种工作模式

◆ GPIO的输出工作模式3——推挽输出模式

开启状态

至片上外设模块

模拟输入

复用功能输入

⑦ 读

⑥

输入数据寄存器

⑤ 开/关

TTL施密特触发器

V_{DD}

开/关

开/关

V_{SS}

输入驱动器

V_{DD}

保护二极管

④

I/O端口

保护二极管

V_{SS}

写

位设置/清除寄存器

输出数据寄存器

①

读/写

来自片上外设模块

复用功能输出

输出驱动器

②

输出控制电路

V_{DD}

P-MOS

N-MOS

V_{SS}

③

GPIO的8种工作模式

◆ GPIO的输出工作模式4——推挽复用输出模式

到片上
外设
模拟
复用功能输入
开/关
读
输入数据寄存器
TTL 施密特
触发器
开/关
输入驱动器
写
置位/复位寄存器
输出数据寄存器
输出驱动器
开/关
V_{DD}
P-MOS
输出控制
N-MOS
V_{SS}
推挽
开漏或
禁止
读/写
自片上
外设
复用功能输出
模拟
V_{DD}
V_{DD_FT} (1)
拉
上
保护
二极管
I/O
引脚
拉
下
保护
二极管
V_{SS}
V_{SS}
ai15939b

GPIO的8种工作模式

◆ GPIO的输出工作模式4—推挽复用输出模式

开启状态

模拟输入
至片上
外设模块
复用功能输入
⑦ 读
⑥
输入数据寄存器
开/关
⑤
TTL施密特
触发器
V_{DD}
开/关
开/关
V_{SS}
输入驱动器
位设置/清除寄存器
写
输出数据寄存器
读/写
来自片上
外设模块
① 复用功能输出
输出驱动器
②
输出控制电路
V_{DD}
P-MOS
N-MOS
V_{SS}
③
V_{DD}
保护二极管
④
I/O端口
保护二极管
V_{SS}

GPIO的8种工作模式

推挽输出：

可以输出强高低电平，连接数字器件

开漏输出：

只可以输出强低电平，高电平得靠外部电阻拉高。输出端相当于三极管的集电极. 要得到高电平状态需要上拉电阻才行. 适合于做电流型的驱动,其吸收电流的能力相对强(一般20ma以内)

GPIO的8种工作模式

■ 上电复位后IO口状态?

上电复位后，GPIO默认为输入浮空状态，部分特殊功能引脚为特定状态。

复位后，调试引脚处于复用功能上拉/下拉状态：

- PA15：JTDI处于上拉状态
- PA14:JTCK/SWCLK处于下拉状态
- PA13:JTMS/SWDAT处于下拉状态
- PB4:NJTRST处于上拉状态
- PB3:JTDO处于浮空状态

GPIO的8种工作模式

- 怎么配置IO口的8种模式?

目录

3 GPIO寄存器

GPIO寄存器

每组GPIO端口的寄存器包括：

一个端口模式寄存器（GPIOx_MODER）
一个端口输出类型寄存器(GPIOx_OTYPER)
一个端口输出速度寄存器（GPIOx_OSPEEDR)
一个端口上拉下拉寄存器（GPIOx_PUPDR)

4个32位配置寄存器

一个端口输入数据寄存器（GPIOx_IDR)
一个端口输出数据寄存器（GPIOx_ODR)

2个32位数据寄存器

一个端口置位/复位寄存器（GPIOx_BSRR)

1个32位置位/复位寄存器

一个端口配置锁存寄存器（GPIOx_LCKR)

1个32位锁存寄存器

两个复用功能寄存器（低位GPIOx_AFRL & GPIOx_AFRH)

2个32复用功能共寄存器

GPIO寄存器

重点说明：

① 每组IO口由10个寄存器组成，如果芯片有GPIOA~GPIOI 9个组，那么一共有对应9X10=90个寄存器。

② 如果配置一个IO口需要2个位，那么刚好32位寄存器配置一组IO口16个IO口

③ 如果配置一个IO口只需要1个位，一般高16位保留

④ BSRR寄存器32位分为低16位BSRRL和高16位BSRRH，BSRRL配置一组IO口的16个IO口的置位状态（1），BSRRH配置复位状态（0）。

GPIO寄存器

◆1 端口模式寄存器(GPIOx_MODER)

GPIO 端口模式寄存器 (GPIOx_MODER) (x =A..K)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 2y+1:2y **MODERy[1:0]**：端口 x 配置位 (Port x configuration bits) (y = 0..15)

这些位通过软件写入，用于配置 I/O 模式。

00：输入模式（复位状态）

01：通用输出模式

10：复用功能模式

11：模拟模式

GPIO寄存器

◆2端口输出类型寄存器(GPIOx_OTYPER)

GPIO 端口输出类型寄存器 (GPIOx_OTYPER) (x = A..K)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:16 保留，必须保持复位值。

位 15:0 **OTy**：端口 x 配置位 (Port x configuration bits) (y = 0..15)
这些位通过软件写入，用于配置 I/O 输出类型。
0：推挽输出（复位状态）
1：开漏输出

GPIO寄存器

◆ 3 端口输出速度寄存器(GPIOx_OSPEEDR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7 [1:0]		OSPEEDR6 [1:0]		OSPEEDR5 [1:0]		OSPEEDR4 [1:0]		OSPEEDR3 [1:0]		OSPEEDR2 [1:0]		OSPEEDR1 [1:0]		OSPEEDR0 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 2y+1:2y **OSPEEDRy[1:0]**：端口 x 配置位 (Port x configuration bits) (y = 0..15)

这些位通过软件写入，用于配置 I/O 输出速度。

00：低速

01：中速

10：快速

11：高速

GPIO寄存器

◆4 端口上拉/下拉寄存器(GPIOx_PUPDR)

GPIO 端口上拉/下拉寄存器 (GPIOx_PUPDR) (x = A..K)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 2y+1:2y **PUPDRy[1:0]**：端口 x 配置位 (Port x configuration bits) (y = 0..15)

这些位通过软件写入，用于配置 I/O 上拉或下拉。

00：无上拉或下拉
01：上拉
10：下拉
11：保留

GPIO寄存器

◆ 5 端口输入数据寄存器(GPIOx_IDR)

GPIO 端口输入数据寄存器 (GPIOx_IDR) (x = A..K)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

位 31:16 保留，必须保持复位值。

位 15:0 **IDRy：**端口输入数据 (Port input data) (y = 0..15)
这些位为只读。它们包含相应 I/O 端口的输入值。

GPIO寄存器

◆ 6 端口输出数据寄存器(GPIOx_ODR)

GPIO 端口输出数据寄存器 (GPIOx_ODR) (x = A..K)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:16 保留，必须保持复位值。

位 15:0 **ODRy：**端口输出数据 (Port output data) (y = 0..15)
这些位可通过软件读取和写入。
注： 对于原子置位/复位，通过写入 GPIOx_BSRR 或 GPIOx_BRR 寄存器，可分别置位和/或复位 ODR 位 (x = A..F)。

GPIO寄存器

◆7 端口置位/复位寄存器(GPIOx_BSRR)

GPIO 端口置位/复位寄存器 (GPIOx_BSRR) (x = A..K)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

位 31:16 **BRy**：端口 x 复位位 y (Port x reset bit y) (y = 0..15)

这些位为只写。读取这些位可返回值 0x0000。

0：不会对相应的 ODRx 位执行任何操作

1：复位相应的 ODRx 位

注：　如果同时对 BSx 和 BRx 置位，则 BSx 的优先级更高。

位 15:0 **BSy**：端口 x 置位位 y (Port x set bit y) (y= 0..15)

这些位为只写。读取这些位可返回值 0x0000。

0：不会对相应的 ODRx 位执行任何操作

1：置位相应的 ODRx 位

GPIO寄存器

◆8 端口配置锁定寄存器(GPIOx_LCKR)

GPIO 端口配置锁定寄存器 (GPIOx_LCKR) (x = A..K)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	LCKK
															rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LCK15	LCK14	LCK13	LCK12	LCK11	LCK10	LCK9	LCK8	LCK7	LCK6	LCK5	LCK4	LCK3	LCK2	LCK1	LCK0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:17 保留，必须保持复位值。

位 16 **LCKK：**锁定键 (Lock key)

可随时读取此位。可使用锁定键写序列对其进行修改。

0：端口配置锁定键未激活

1：端口配置锁定键已激活。在下一次 MCU 复位或外设复位之前，GPIOx_LCKR 寄存器始终处于锁定状态。

位 15:0 **LCKy：**端口 x 锁定位 y (Port x lock bit y) (y= 0..15)

这些位都是读/写位，但只能在 LCKK 位等于“0”时执行写操作。

0：端口配置未锁定

1：端口配置已锁定

GPIO寄存器

◆ 9 复用功能寄存器(GPIOx_AFRL、AFRH)

分高位AFRH和低位AFRL，分别控制8个IO口

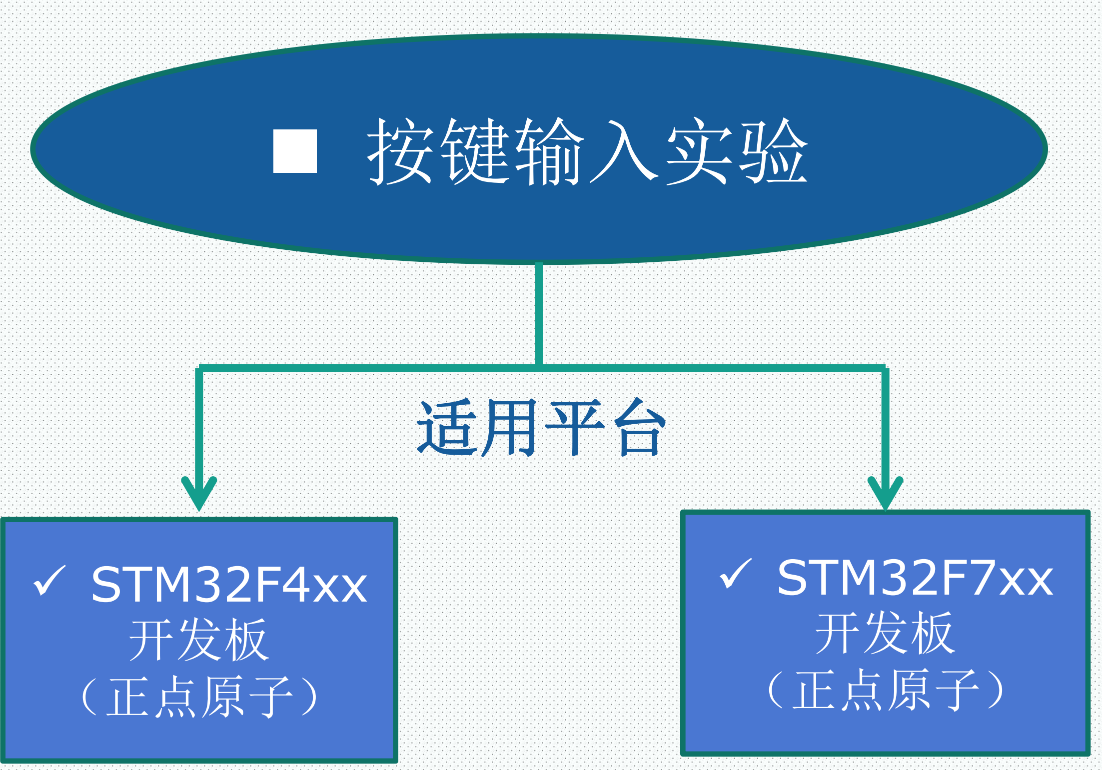
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFR7[3:0]				AFR6[3:0]				AFR5[3:0]				AFR4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFR3[3:0]				AFR2[3:0]				AFR1[3:0]				AFR0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:0 **AFRy[3:0]**：端口 x 引脚 y 的复用功能选择 (Alternate function selection for port x pin y) (y = 0..7)

这些位通过软件写入，用于配置复用功能 I/O。

AFSELy 选择：

0000: AF0	1000: AF8
0001: AF1	1001: AF9
0010: AF2	1010: AF10
0011: AF3	1011: AF11
0100: AF4	1100: AF12
0101: AF5	1101: AF13
0110: AF6	1110: AF14
0111: AF7	1111: AF15


■ 按键输入实验
适用平台
✓ STM32F4xx
开发板
（正点原子）
✓ STM32F7xx
开发板
（正点原子）

目录

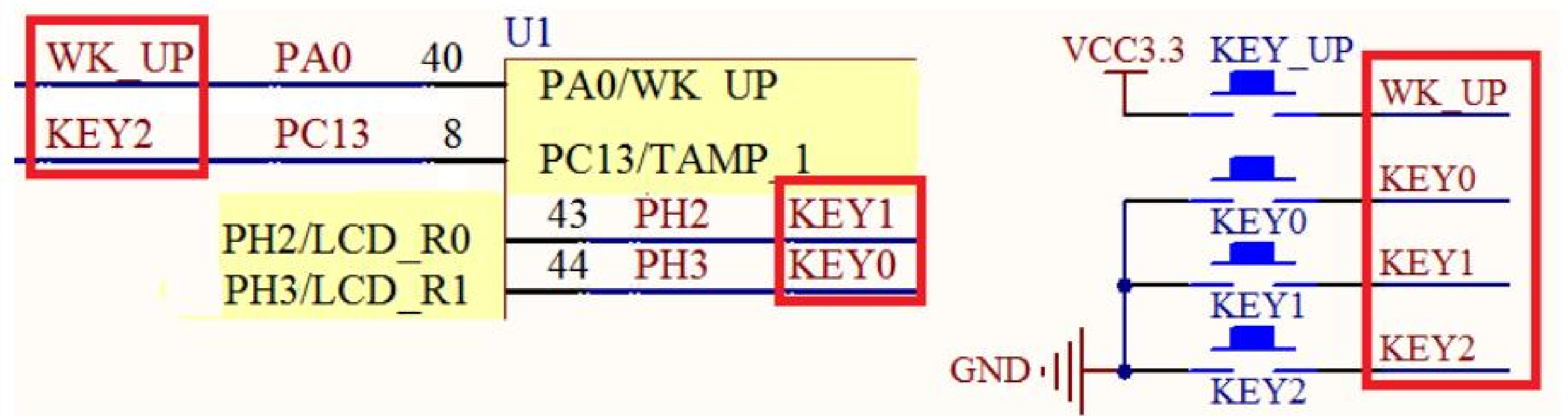
1 按键实验硬件连接

2 GPIO输入操作说明

3 按键实验

按键实验硬件连接

硬件连接



KEY0->PH3 上拉输入

KEY1->PH2 上拉输入

KEY2->PC13 上拉输入

WK_UP->PA0 下拉输入

按键输入实验

- **读取IO口输入电平调用库函数为：**

GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef GPIOx, uint16_t GPIO_Pin);*

- **读取IO口输入电平操作寄存器为：**

GPIOx_IDR:端口输入寄存器

- **使用位带操作读取IO口输入电平方法：**

PEin(4)　　-读取GPIOE.4口电平
PEin(n)　　-读取GPIOE.n口电平

按键输入实验

◆ 手把手写按键输入实验。

① 使能按键对应IO口时钟：

__HAL_RCC_GPIOx_CLK_ENABLE;

② 初始化IO模式：上拉/下拉输入。

void HAL_GPIO_Init();

③ 扫描IO口电平（库函数/寄存器/位操作）：

HAL库函数：*GPIO_PinState HAL_GPIO_ReadPin();*

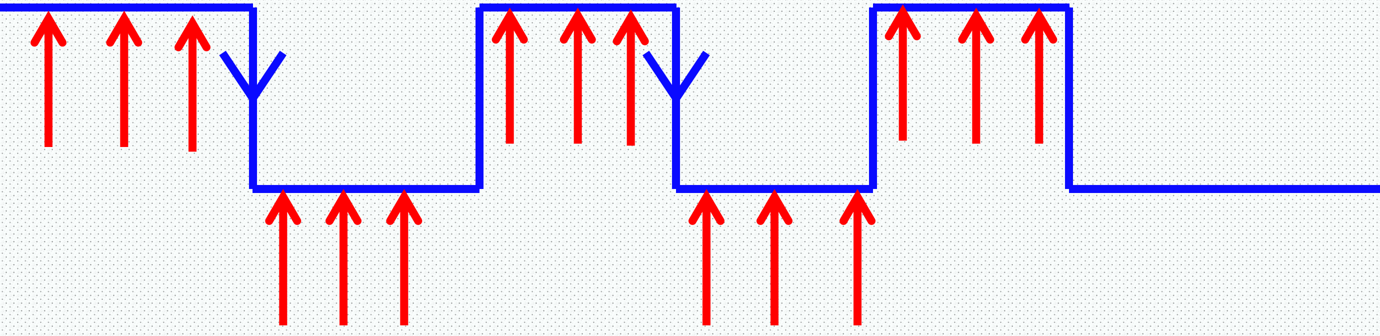
寄存器： *GPIOx_IDR*

位操作： *PHin(1);*

④ 编写按键扫描逻辑

按键输入实验

- 按键扫描思路



按键输入实验

◆ 按键扫描（支持连续按）的一般思路

```
u8 KEY_Scan(void)
 {
  if(KEY按下）
  {
     delay_ms(10)；//延时10-20ms，防抖。
     if(KEY确实按下)
      {
        return KEY_Value；
      }
     return 无效值；
  }
 }
```

如果我要实现：按键按下，没有松开，只能算按下一次，这个函数无法实现。

按键输入实验

◆每次调用getValue函数之后，返回值是多少?

```
int getValue(void)
{
  int  flag=0;
  flag++;
  return flag;
}
```

```
int getValue(void)
{
  static int  flag=0;
  flag++;
  return flag;
}
```

按键输入实验

- 按键扫描（不支持连续按）的一般思路

```
u8 KEY_Scan(void)
  {
   static u8 key_up=1;
    if（key_up && KEY按下）
    {
      delay_ms(10);//延时，防抖
      key_up=0;//标记这次key已经按下
      if(KEY确实按下)
        {
         return KEY_VALUE;
        }
      }else if(KEY没有按下)  key_up=1;
  }
```

不支持连续按：就是说，按键按下了，没有松开，只能算一次。

按键输入实验

◆ 按键扫描（两种模式合二为一）的一般思路

```
u8 KEY_Scan(u8 mode)
 {
  static u8 key_up=1;
  if(mode==1) key_up=1;//支持连续按
   if（key_up &&  KEY按下）
   {
     delay_ms(10);//延时，防抖
     key_up=0;//标记这次key已经按下
     if(KEY确实按下)
       {
        return KEY_VALUE;
       }
    }else if(KEY没有按下)  key_up=1;
   return 没有按下
 }
```

- 中断优先级管理 NVIC

目录

1 NVIC中断优先级分组

2 NVIC中断优先级设置

3 NVIC总结

目录

1 NVIC中断优先级分组

NVIC中断优先级分组

- CM4/CM7 内核支持256个中断，其中包含了16个内核中断和240个外部中断，并且具有256级的可编程中断设置。
- STM32F4/F7并没有使用CM4内核的全部东西，而是只用了它的一部分。
  ① STM32F40xx/STM32F41xx总共有92个中断。10个内核中断，82个可屏蔽中断。
  ② STM32F42xx/STM32F43xx则总共有97个中断。10个内核中断，87个可屏蔽中断。
  ③ STM32F76x总共118个中断，10个内核中断，108个可屏蔽中断。

NVIC中断优先级分组

- STM32具有16级可编程的中断优先级，而我们常用的就是这些可屏蔽中断。

NVIC中断优先级分组

《STM32F中文参考手册》中搜索向量表可以找到相应的中断说明。

10个内核中断

位置	优先级	优先级类型	名称	说明	地址
	-	-	-	保留	0x0000 0000
	-3	固定	Reset	复位	0x0000 0004
	-2	固定	NMI	不可屏蔽中断。RCC 时钟安全系统 (CSS) 连接到 NMI 向量。	0x0000 0008
	-1	固定	HardFault	所有类型的错误	0x0000 000C
	0	可设置	MemManage	存储器管理	0x0000 0010
	1	可设置	BusFault	预取指失败，存储器访问失败	0x0000 0014
	2	可设置	UsageFault	未定义的指令或非法状态	0x0000 0018
	-	-	-	保留	0x0000 001C - 0x0000 002B
	3	可设置	SVCall	通过 SWI 指令调用的系统服务	0x0000 002C
	4	可设置	Debug Monitor	调试监控器	0x0000 0030
	-	-	-	保留	0x0000 0034
	5	可设置	PendSV	可挂起的系统服务	0x0000 0038
	6	可设置	SysTick	系统嘀嗒定时器	0x0000 003C

可屏蔽中断

0	7	可设置	WWDG	窗口看门狗中断	0x0000 0040
1	8	可设置	PVD	连接到 EXTI 线的可编程电压检测 (PVD) 中断	0x0000 0044
2	9	可设置	TAMP_STAMP	连接到 EXTI 线的入侵和时间戳中断	0x0000 0048
3	10	可设置	RTC_WKUP	连接到 EXTI 线的 RTC 唤醒中断	0x0000 004C
4	11	可设置	FLASH	Flash 全局中断	0x0000 0050
5	12	可设置	RCC	RCC 全局中断	0x0000 0054
6	13	可设置	EXTI0	EXTI 线 0 中断	0x0000 0058
7	14	可设置	EXTI1	EXTI 线 1 中断	0x0000 005C
8	15	可设置	EXTI2	EXTI 线 2 中断	0x0000 0060
9	16	可设置	EXTI3	EXTI 线 3 中断	0x0000 0064
10	17	可设置	EXTI4	EXTI 线 4 中断	0x0000 0068
11	18	可设置	DMA1_Stream0	DMA1 流 0 全局中断	0x0000 006C

......

75	82	可设置	OTG_HS_EP1_IN	USB On The Go HS 端点 1 输入全局中断	0x0000 016C
76	83	可设置	OTG_HS_WKUP	连接到 EXTI 线的 USB On The Go HS 唤醒中断	0x0000 0170
77	84	可设置	OTG_HS	USB On The Go HS 全局中断	0x0000 0174
78	85	可设置	DCMI	DCMI 全局中断	0x0000 0178
79	86	可设置	CRYP	CRYP 加密全局中断	0x0000 017C
80	87	可设置	HASH_RNG	哈希和随机数发生器全局中断	0x0000 0180
81	88	可设置	FPU	FPU 全局中断	0x0000 0184

NVIC中断优先级分组

几十个中断，怎么管理?

NVIC中断优先级分组

◆中断管理方法：

首先，对STM32中断进行分组，组0~4。同时，对每个中断设置一个抢占优先级和一个响应优先级值。

分组配置是在寄存器SCB->AIRCR中配置：

组	AIRCR[10：8]	IP bit[7：4]分配情况	分配结果
0	111	0：4	0位抢占优先级，4位响应优先级
1	110	1：3	1位抢占优先级，3位响应优先级
2	101	2：2	2位抢占优先级，2位响应优先级
3	100	3：1	3位抢占优先级，1位响应优先级
4	011	4：0	4位抢占优先级，0位响应优先级

NVIC中断优先级分组

◆抢占优先级 & 响应优先级区别：

- 高优先级的抢占优先级是可以打断正在进行的低抢占优先级中断的。
- 抢占优先级相同的中断，高响应优先级不可以打断低响应优先级的中断。
- 抢占优先级相同的中断，当两个中断同时发生的情况下，哪个响应优先级高，哪个先执行。
- 如果两个中断的抢占优先级和响应优先级都是一样的话，则看哪个中断先发生就先执行；

NVIC中断优先级分组

◆ 举例:

假定设置中断优先级组为2，然后设置

中断3(RTC中断)的抢占优先级为2，响应优先级为1。

中断6（外部中断0）的抢占优先级为3，响应优先级为0

中断7（外部中断1）的抢占优先级为2，响应优先级为0。

那么这3个中断的优先级顺序为：中断7>中断3>中断6。

NVIC中断优先级分组

◆ **特别说明：**

一般情况下，系统代码执行过程中，只设置一次中断优先级分组，比如分组2，设置好分组之后一般不会再改变分组。随意改变分组会导致中断管理混乱，程序出现意想不到的执行结果。

NVIC中断优先级分组

◆ 中断优先级分组函数：

void HAL_NVIC_SetPriorityGrouping(uint32_t PriorityGroup);

```
void HAL_NVIC_SetPriorityGrouping(uint32_t PriorityGroup)
{
  /* Check the parameters */
  assert_param(IS_NVIC_PRIORITY_GROUP(PriorityGroup));

  NVIC_SetPriorityGrouping(PriorityGroup);
}
```

```
HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_2);
```

中断优先级设置

分组设置好之后，怎么设置单个中断的抢占优先级和响应优先级？

目录

2 NVIC中断优先级设置

中断优先级设置

◆ 中断设置相关寄存器

```
__IO uint8_t  IP[240]; //中断优先级控制的寄存器组

__IO uint32_t ISER[8]; //中断使能寄存器组
__IO uint32_t ICER[8]; //中断失能寄存器组
__IO uint32_t ISPR[8]; //中断挂起寄存器组
__IO uint32_t ICPR[8]; //中断解挂寄存器组
__IO uint32_t IABR[8]; //中断激活标志位寄存器组
```

中断优先级设置

◆ MDK中NVIC寄存器结构体

```
typedef struct
{
  __IO uint32_t ISER[8];
       uint32_t RESERVED0[24];
  __IO uint32_t ICER[8];
       uint32_t RSERVED1[24];
  __IO uint32_t ISPR[8];
       uint32_t RESERVED2[24];
  __IO uint32_t ICPR[8];
       uint32_t RESERVED3[24];
  __IO uint32_t IABR[8];
       uint32_t RESERVED4[56];
  __IO uint8_t  IP[240];
       uint32_t RESERVED5[644];
  __O  uint32_t STIR;
}  NVIC_Type;
```

中断优先级设置

◆ 对于每个中断怎么设置优先级?

中断优先级控制的寄存器组：IP[240]
全称是：Interrupt Priority Registers

240个8位寄存器，每个中断使用一个寄存器来确定优先级。
STM32F40x系列一共82个可屏蔽中断，使用IP[81]~IP[0]。

每个IP寄存器的高4位用来设置抢占和响应优先级（根据分组），低4位没有用到。

```
void HAL_NVIC_SetPriority(IRQn_Type IRQn, uint32_t
                          PreemptPriority, uint32_t SubPriority);
```

中断优先级设置

◆ 中断使能寄存器组：ISER[8]

作用：用来使能中断

32位寄存器，每个位控制一个中断的使能。STM32F40x只有82个可屏蔽中断，所以只使用了其中的ISER[0]~ISER[2]。

ISER[0]的bit0~bit31分别对应中断0~31。ISER[1]的bit0~31对应中断32~63；ISER[2]的bit0~31对应中断64~96；

void HAL_NVIC_EnableIRQ(IRQn_Type IRQn)；

中断优先级设置

◆ 中断失能寄存器组：ICER[8]

作用：用来失能中断

32位寄存器，每个位控制一个中断的失能。STM32F40x只有82个可屏蔽中断，所以只使用了其中的ICER[0]和ICER[1]。

ICER[0]的bit0~bit31分别对应中断0~31。ICER[1]的bit0~31对应中断32~63； ICER[3]的bit0~31对应中断64~95；

配置方法跟ISER一样。

```
void HAL_NVIC_DisableIRQ(IRQn_Type IRQn);
```

中断优先级设置

- 中断挂起控制寄存器组：ISPR[8]

  作用：用来挂起中断

- 中断解挂控制寄存器组：ICPR[8]

  作用：用来解挂中断

- 中断激活标志位寄存器组：IABR [8]

  作用：只读，通过它可以知道当前在执行的中断是哪一个

目录

3 NVIC总结

NVIC总结

◆ 中断优先级设置步骤

①系统运行后在**HAL_Init**函数中设置中断优先级分组。

调用函数：

HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_2);

//中断优先级分组2 整个系统执行过程中，只设置一次中断分组。

②针对每个中断，设置对应的抢占优先级和响应优先级：

void HAL_NVIC_SetPriority(IRQn_Type IRQn, uint32_t PreemptPriority, uint32_t SubPriority);

③ 使能中断通道：

void HAL_NVIC_EnableIRQ(IRQn_Type IRQn);

- 外部中断原理与配置

目录

1 IO口外部中断原理概述

2 IO口外部中断HAL库配置方法

3 手把手写IO口外部中断实验

目录

1 IO口外部中断原理概述

外部中断概述

◆STM32控制器支持的外部中断/事件请求：

中断线	M3	M4	M7
EXTI线0~15:对应外部IO口的输入中断。	✓	✓	✓
EXTI线16：连接到PVD输出。	✓	✓	✓
EXTI线17：连接到RTC闹钟事件。	✓	✓	✓
EXTI线18：连接到USB OTG FS唤醒事件。	✓	✓	✓
EXTI线19：连接到以太网唤醒事件。		✓	✓
EXTI线20：连接到USB OTG HS(在FS中配置)唤醒事件		✓	✓
EXTI线21：连接到RTC入侵和时间戳事件。		✓	✓
EXTI线22：连接到RTC唤醒事件。		✓	✓
EXSTI线23：连接到LPTIM1异步事件			✓

外部中断概述

■IO口外部中断：

① STM32的每个IO都可以作为外部中断输入。

② 每个外部中断线可以独立的配置触发方式（上升沿，下降沿或者双边沿触发），触发/屏蔽，专用的状态位。

③ STM32供IO使用的中断线只有16个，但是STM32F系列的IO口多达上百个，STM32F103ZGT6(112),那么中断线怎么跟io口对应呢？

外部中断概述

■ GPIO和中断线映射关系

GPIOx.0映射到EXTI0

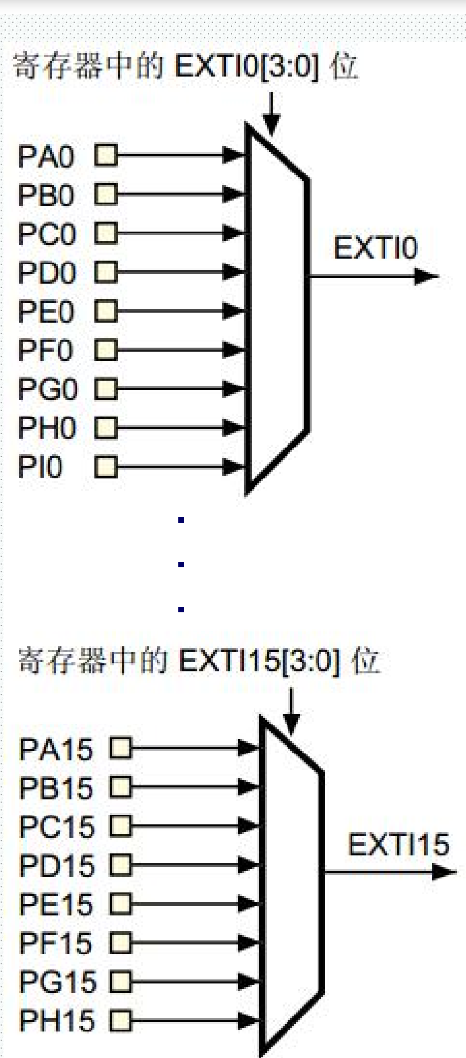
GPIOx.1映射到EXTI1

…

…

GPIOx.14映射到EXTI14

GPIOx.15映射到EXTI15



对于M4/M7，配置寄存器为SYSCFG_EXTIRx
对于M3，配置寄存器为AFIO_EXTICRx

外部中断概述

◆16个中断线就分配16个中断服务函数?

外部中断概述

◆ IO口外部中断在中断向量表中只分配了7个中断向量，也就是只能使用7个中断服务函数。

13	可设置	EXTI0	EXTI 线 0 中断
14	可设置	EXTI1	EXTI 线 1 中断
15	可设置	EXTI2	EXTI 线 2 中断
16	可设置	EXTI3	EXTI 线 3 中断
17	可设置	EXTI4	EXTI 线 4 中断
30	可设置	EXTI9_5	EXTI 线 [9:5] 中断
47	可设置	EXTI15_10	EXTI 线 [15:10] 中断

◆ 从表中可以看出，外部中断线5~9分配一个中断向量，共用一个服务函数外部中断线10~15分配一个中断向量，共用一个中断服务函数。

外部中断概述

- **中断服务函数列表：**

```
EXTI0_IRQHandler
EXTI1_IRQHandler
EXTI2_IRQHandler
EXTI3_IRQHandler
EXTI4_IRQHandler
EXTI9_5_IRQHandler
EXTI15_10_IRQHandler
```

目录

2 IO口外部中断HAL库配置方法

外部中断配置

- **外部中断操作使用到的函数分布文件：**

stm32fxxx_hal_gpio.h

stm32fxxx_hal_gpio.c

外部中断配置

◆ 外部中断配置：

外部中断的中断线映射配置和触发方式都是在GPIO初始化函数中完成：

```
GPIO_InitTypeDef GPIO_Initure;
GPIO_Initure.Pin=GPIO_PIN_0;                //PA0
GPIO_Initure.Mode=GPIO_MODE_IT_RISING;      //上升沿触发
GPIO_Initure.Pull=GPIO_PULLDOWN;
HAL_GPIO_Init(GPIOA,&GPIO_Initure);
```

外部中断配置

- 和串口中断一样，HAL库同样提供了外部中断通用处理函数HAL_GPIO_EXTI_IRQHandler，我们在外部中断服务函数中会调用该函数处理中断。

```
//中断服务函数
void EXTI0_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);//调用中断处理公用函数
}

void EXTI2_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_2);//调用中断处理公用函数
}
```

外部中断配置

- HAL_GPIO_EXTI_IRQHandler函数内部通过判断中断来源引脚，最终调用外部中断回调函数HAL_GPIO_EXTI_Callback来处理中断。

```
void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
{
  if(__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) != RESET)
  {
     __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
    HAL_GPIO_EXTI_Callback(GPIO_Pin);
  }
}
```

外部中断配置

- 用户最终编写中断处理回调函数来编写中断处理逻辑。

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    switch(GPIO_Pin)
    {
        case GPIO_PIN_0:
            //控制逻辑
            break;
        case GPIO_PIN_2:
            //控制逻辑
            break;
    }
}
```

外部中断一般配置过程

◆ 外部中断的一般配置步骤：

① 使能IO口时钟。

② 初始化IO口，设置触发方式：HAL_GPIO_Init();

③ 设置中断优先级，并使能中断通道。

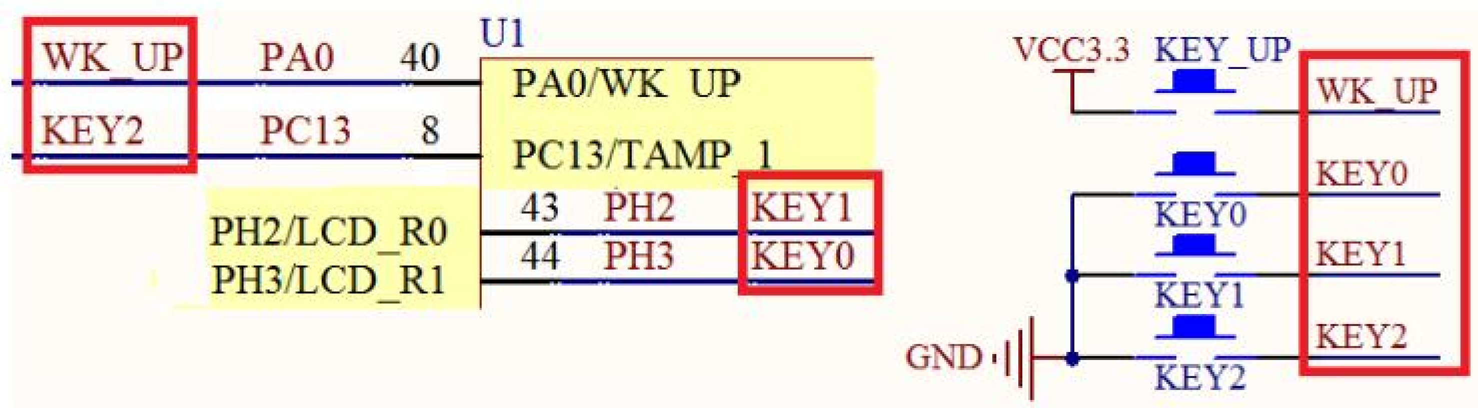
④ 编写中断服务函数：

函数中调用外部中断通用处理函数HAL_GPIO_EXTI_IRQHandler。

⑥ 编写外部中断回调函数：HAL_GPIO_EXTI_Callback;

外部中断实验硬件连接

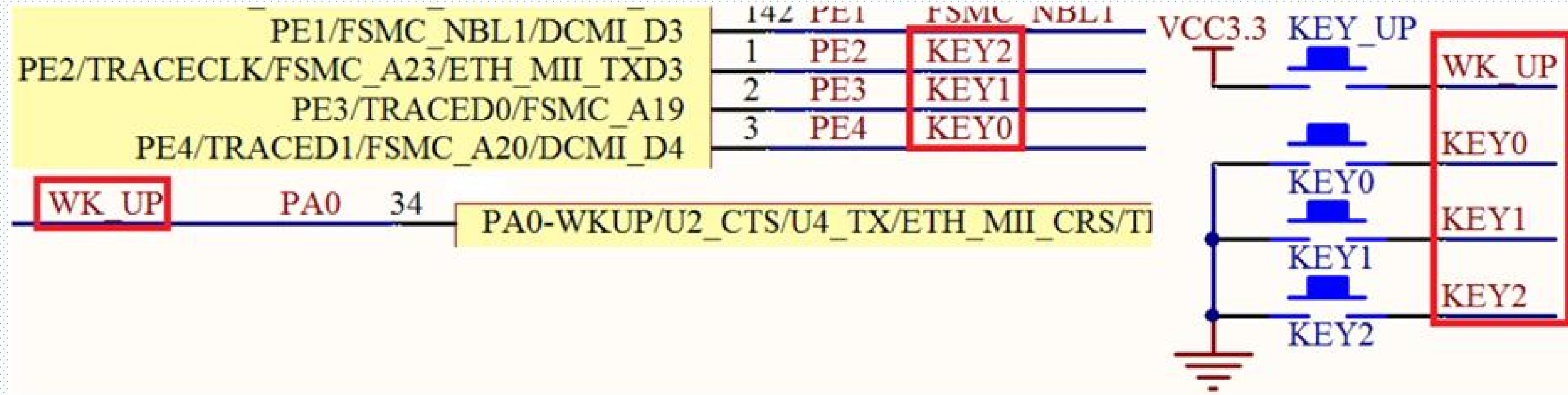
■ 阿波罗开发板按键硬件连接：



KEY0->PH3　　上拉输入，下降沿触发
KEY1->PH2　　上拉输入，下降沿触发
KEY2->PC13　上拉输入，下降沿触发
WK_UP->PA0　下拉输入，上升沿触发

外部中断实验硬件连接

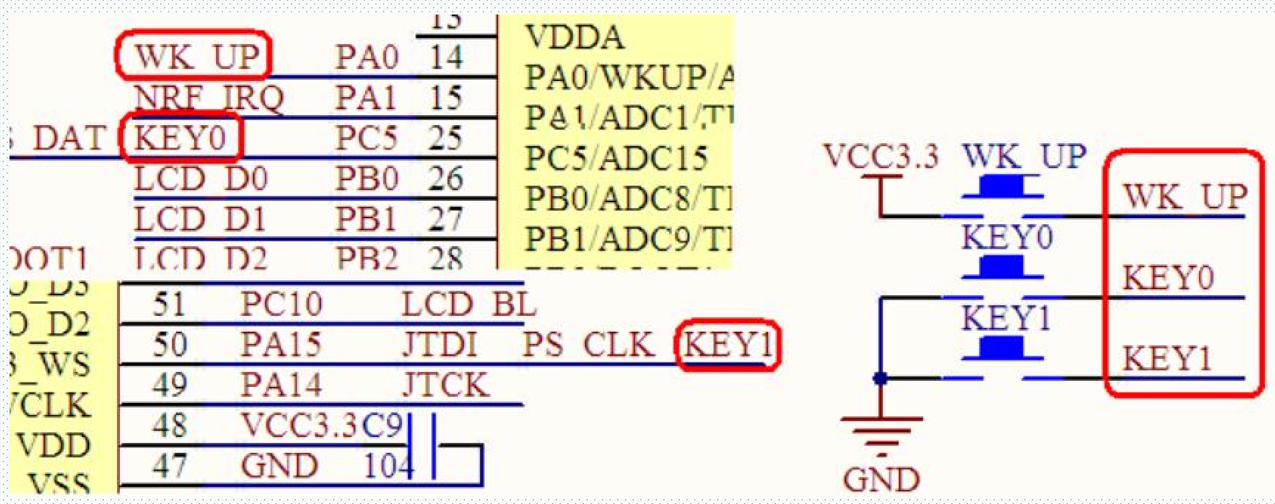
■ **探索者/战舰/精英 开发板按键硬件连接：**



KEY0->PE4　上拉输入，下降沿触发
KEY1->PE3　上拉输入，下降沿触发
KEY2->PE2　上拉输入，下降沿触发（精英板没有KEY2）
WK_UP->PA0　下拉输入，上升沿触发

外部中断实验硬件连接

- mini开发板按键硬件连接：



KEY0->PC5 上拉输入，下降沿触发

KEY1->PA15 上拉输入，下降沿触发

WK_UP->PA0 下拉输入，上升沿触发

目录

3 手把手写IO口外部中断实验

■ 要求外部中断实验实现功能：

按键KEY0按下：　同时控制LED0和LED1翻转。

按键KEY1按下：　LED1状态翻转。

按键KEY2按下：　LED0翻转。

按键WK_UP按下：控制LED0和LED1互斥点亮。

- IO引脚复用和映射

端口复用

◆ 什么是端口复用?

STM32有很多的内置外设，这些外设的外部引脚都是与GPIO复用的。也就是说，一个GPIO如果可以复用为内置外设的功能引脚，那么当这个GPIO作为内置外设使用的时候，就叫做复用。

端口复用

PE2 1
PE3 2
PE4 3
PE5 4
PE6 5
VBAT 6
PI8 7
PC13 8
PC14 9
PC15 10
PI9 11
PI10 12
PI11 13
VSS 14
VDD 15
PF0 16
PF1 17
PF2 18
PF3 19
PF4 20
PF5 21
VSS 22
VDD 23
PF6 24
PF7 25
PF8 26
PF9 27
PF10 28
PH0 29
PH1 30
NRST 31
PC0 32
PC1 33
PC2 34
PC3 35
VDD 36
VSSA 37
VREF+ 38
VDDA 39
PA0 40
PA1 41
PA2 42
PH2 43
PH3 44

132 PI1
131 PI0
130 PH15
129 PH14
128 PH13
127 V_{DD}
126 V_{SS}
125 V_{CAP_2}
124 PA13
123 PA12
122 PA11
121 PA10
120 PA9
119 PA8
118 PC9
117 PC8
116 PC7
115 PC6
114 V_{DD}
113 V_{SS}
112 PG8
111 PG7
110 PG6
109 PG5
108 PG4
107 PG3
106 PG2
105 PD15
104 PD14
103 V_{DD}
102 V_{SS}
101 PD13
100 PD12
99 PD11
98 PD10
97 PD9
96 PD8
95 PB15
94 PB14
93 PB13
92 PB12
91 V_{DD}
90 V_{SS}
89 PH12

端口复用

◆ 例如串口1 的发送接收引脚是PA9,PA10，当我们把PA9,PA10不用作GPIO，而用做复用功能串口1的发送接收引脚的时候，叫端口复用。

Pin number					Pin name (function after reset)[1]	Pin type	I / O structure	Notes	Alternate functions	Additional functions
LQFP64	LQFP100	LQFP144	UFBGA176	LQFP176						
41	67	100	F15	119	PA8	I/O	FT		MCO1 / USART1_CK/ TIM1_CH1/ I2C3_SCL/ OTG_FS_SOF/ EVENTOUT	
42	68	101	E15	120	PA9	I/O	FT		USART1_TX/ TIM1_CH2 / I2C3_SMBA / DCMI_D0/ EVENTOUT	OTG_FS_VBUS
43	69	102	D15	121	PA10	I/O	FT		USART1_RX/ TIM1_CH3/ OTG_FS_ID/DCMI_D1/ EVENTOUT	

端口复用

◆ STM32(M4内核以上)的端口复用映射原理

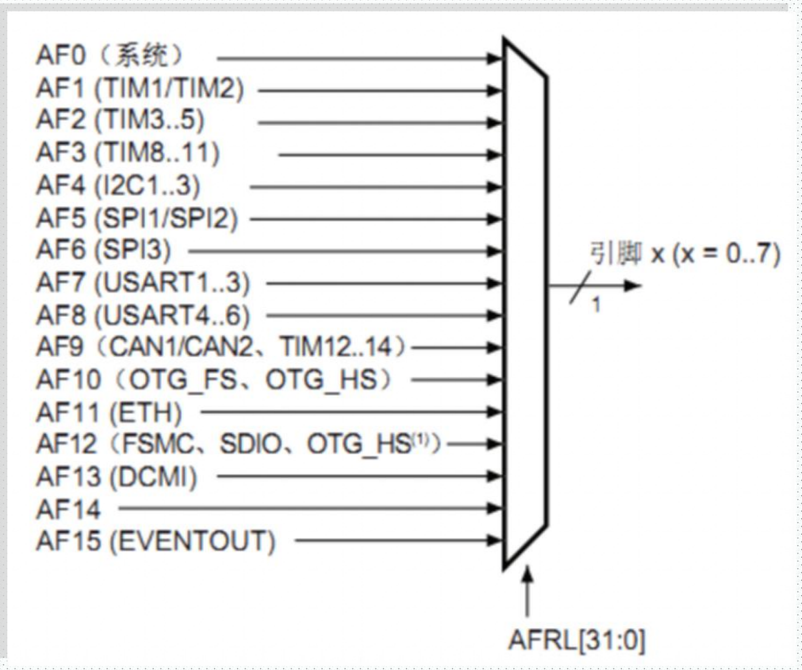
- ✓ STM32系列微控制器IO引脚通过一个复用器连接到内置外设或模块。该复用器一次只允许一个外设的复用功能（AF）连接到对应的IO口。这样可以确保共用同一个IO引脚的外设之间不会发生冲突。

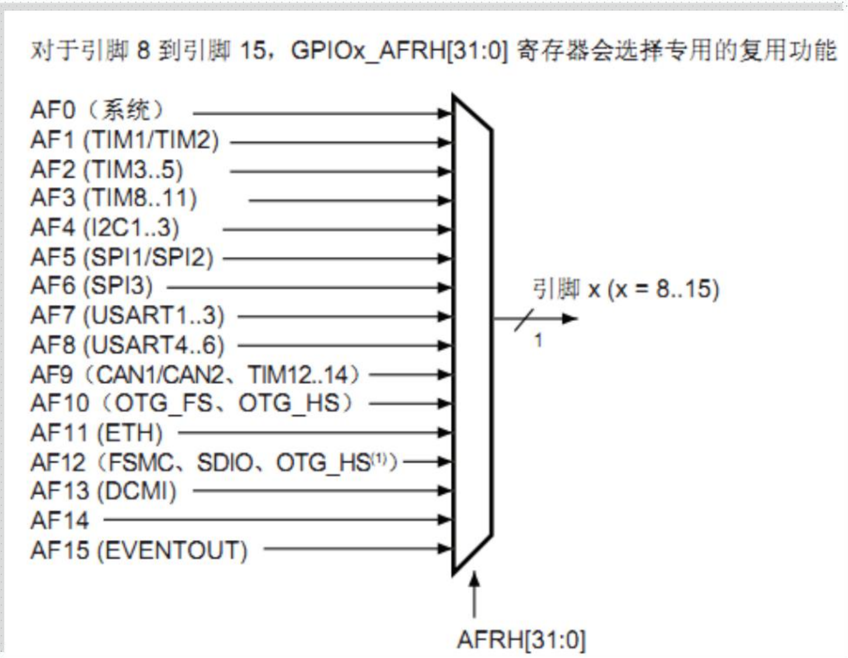
端口复用

- ✓ 每个IO引脚都有一个复用器，该复用器采用16路复用功能输入（AF0到AF15），可通过GPIOx_AFRL(针对引脚0-7)和GPIOx_AFRH（针对引脚8-15）寄存器对这些输入进行配置，每四位控制一路复用。

端口复用

■ 端口复用映射示意图





端口复用

■AFRL复用功能低位寄存器

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:0 **AFRLy**：端口 x 位 y 的复用功能选择 (Alternate function selection for port x bit y) (y = 0..7)
这些位通过软件写入，用于配置复用功能 I/O。

AFRLy 选择：

0000：AF0	1000：AF8
0001：AF1	1001：AF9
0010：AF2	1010：AF10
0011：AF3	1011：AF11
0100：AF4	1100：AF12
0101：AF5	1101：AF13
0110：AF6	1110：AF14
0111：AF7	1111：AF15

端口复用

■AFRH复用功能高位寄存器

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRH15[3:0]				AFRH14[3:0]				AFRH13[3:0]				AFRH12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRH11[3:0]				AFRH10[3:0]				AFRH9[3:0]				AFRH8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位 31:0 **AFRHy**：端口 x 位 y 的复用功能选择 (Alternate function selection for port x bit y) (y = 8.0.15)

这些位通过软件写入，用于配置复用功能 I/O。

AFRHy 选择：

0000：AF0	1000：AF8
0001：AF1	1001：AF9
0010：AF2	1010：AF10
0011：AF3	1011：AF11
0100：AF4	1100：AF12
0101：AF5	1101：AF13
0110：AF6	1110：AF14
0111：AF7	1111：AF15

端口复用

■ 复用功能映射配置

1. 系统功能

将 I/O 连接到 AF0，然后根据所用功能进行配置：

– JTAG/SWD：在各器件复位后，会将这些引脚指定为专用引脚，可供片上调试模块立即使用（不受 GPIO 控制器控制）。

– RTC_REFIN：此引脚应配置为输入浮空模式。

– MCO1 和 MCO2：这些引脚必须配置为复用功能模式。

2. GPIO

在 GPIOx_MODER 寄存器中将所需 I/O 配置为输出或输入。

端口复用

3. 外设复用功能

对于 ADC 和 DAC，在 GPIOx_MODER 寄存器中将所需 I/O 配置为模拟通道。

对于其它外设：

– 在 GPIOx_MODER 寄存器中将所需 I/O 配置为复用功能

– 通过 GPIOx_OTYPER、GPIOx_PUPDR 和 GPIOx_OSPEEDER 寄存器，分别选择类型、上拉/下拉以及输出速度

– 在 GPIOx_AFRL 或 GPIOx_AFRH 寄存器中，将 I/O 连接到所需 AFx

端口复用配置过程

◆端口复用为复用功能配置过程

-以PA9,PA10配置为串口1为例

①GPIO端口时钟使能。

__HAL_RCC_GPIOA_CLK_ENABLE(); //使能GPIO时钟

②复用外设时钟使能。

比如你要将端口PA9,PA10复用为串口，所以要使能串口时钟。

__HAL_RCC_USART1_CLK_ENABLE(); //使能GPIO时钟

端口复用配置过程

③端口模式配置为复用功能。 HAL_GPIO_Init函数。

GPIO_Initure.Mode=GPIO_MODE_AF_PP; //复用推挽输出

④配置GPIOx_AFRL或者GPIOx_AFRH寄存器，将IO连接到所需的AFx。HAL_GPIO_Init函数。

GPIO_Initure.Alternate=GPIO_AF7_USART1;
//复用为USART1

端口复用配置过程

◆PA9,PA10复用为串口1的配置过程

```
__HAL_RCC_GPIOA_CLK_ENABLE();                  //使能GPIOA时钟
__HAL_RCC_USART1_CLK_ENABLE();                 //使能USART1时钟

GPIO_Initure.Pin=GPIO_PIN_9;                   //PA9
GPIO_Initure.Mode=GPIO_MODE_AF_PP;             //复用推挽输出
GPIO_Initure.Pull=GPIO_PULLUP;                 //上拉
GPIO_Initure.Speed=GPIO_SPEED_FAST;            //高速
GPIO_Initure.Alternate=GPIO_AF7_USART1;        //复用为USART1
HAL_GPIO_Init(GPIOA,&GPIO_Initure);            //初始化PA9

GPIO_Initure.Pin=GPIO_PIN_10;                  //PA10
HAL_GPIO_Init(GPIOA,&GPIO_Initure);            //初始化PA10
```

- 串口通信原理

目录

1 串行通信接口背景知识

2 异步串口通信UART基础知识

3 STM32串口数据格式和通信过程

目录

1 串行通信接口背景知识

通信接口背景知识

◆处理器与外部设备通信的两种方式：

- **并行通信**

  -传输原理：数据各个位同时传输。

  -优点：速度快

  -缺点：占用引脚资源多

- **串行通信**

  -传输原理：数据按位顺序传输。

  -优点：占用引脚资源少

  -缺点：速度相对较慢

通信接口背景知识

◆ 串行通信：

按照数据传送方向，分为：

◆**单工**：

数据传输只支持数据在一个方向上传输

◆**半双工**：

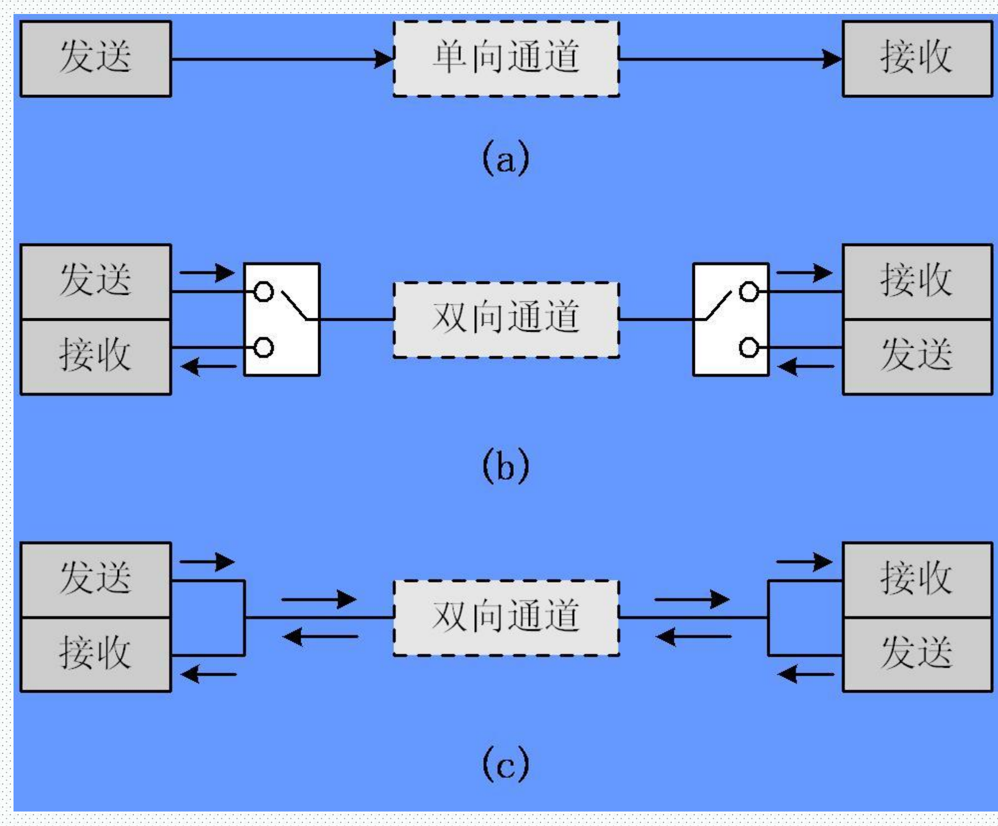
允许数据在两个方向上传输，但是，在某一时刻，只允许数据在一个方向上传输，它实际上是一种切换方向的单工通信；

◆**全双工**：

允许数据同时在两个方向上传输，因此，全双工通信是两个单工通信方式的结合，它要求发送设备和接收设备都有独立的接收和发送能力。

通信接口背景知识

◆ 串行通信三种传送方式：



通信接口背景知识

◆串行通信的通信方式：

- **同步通信**：带时钟同步信号传输。

  -SPI，IIC通信接口

- **异步通信**：不带时钟同步信号。

  -UART(通用异步收发器),单总线

通信接口背景知识

◆常见的串行通信接口：

通信标准	引脚说明	通信方式	通信方向
UART (通用异步收发器)	TXD:发送端 RXD:接受端 GND:公共地	异步通信	全双工
单总线 （1-wire)	DQ:发送/接受端	异步通信	半双工
SPI	SCK:同步时钟 MISO:主机输入，从机输出 MOSI:主机输出，从机输入	同步通信	全双工
I2C	SCL:同步时钟 SDA:数据输入/输出端	同步通信	半双工

目录

2 异步串口通信UART基础知识

串口通信基础

◆异步通信UART包含三点知识：

① 物理层(电气层：接口决定）：

通信接口（RS232,RS485,RS422,TTL)

② 数据格式（数据层：芯片决定）

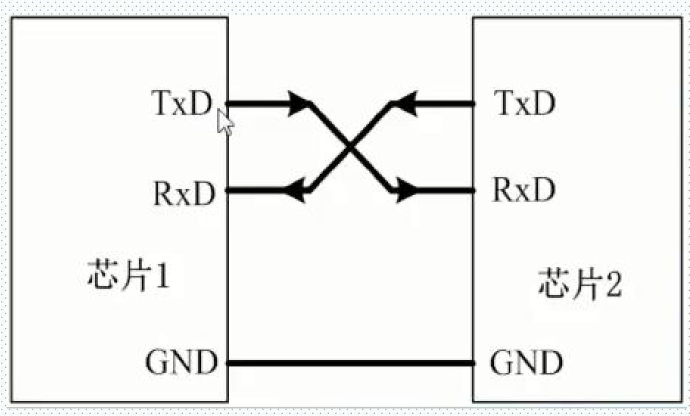
③ 通信协议（协议层：程序决定）

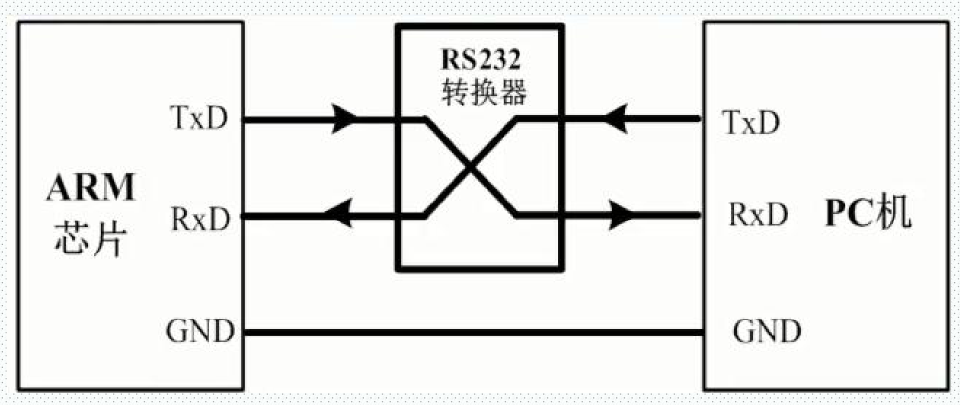
串口通信基础

◆UART异步通信方式引脚连接方法:

-RXD:数据输入引脚。数据接受。

-TXD:数据发送引脚。数据发送。





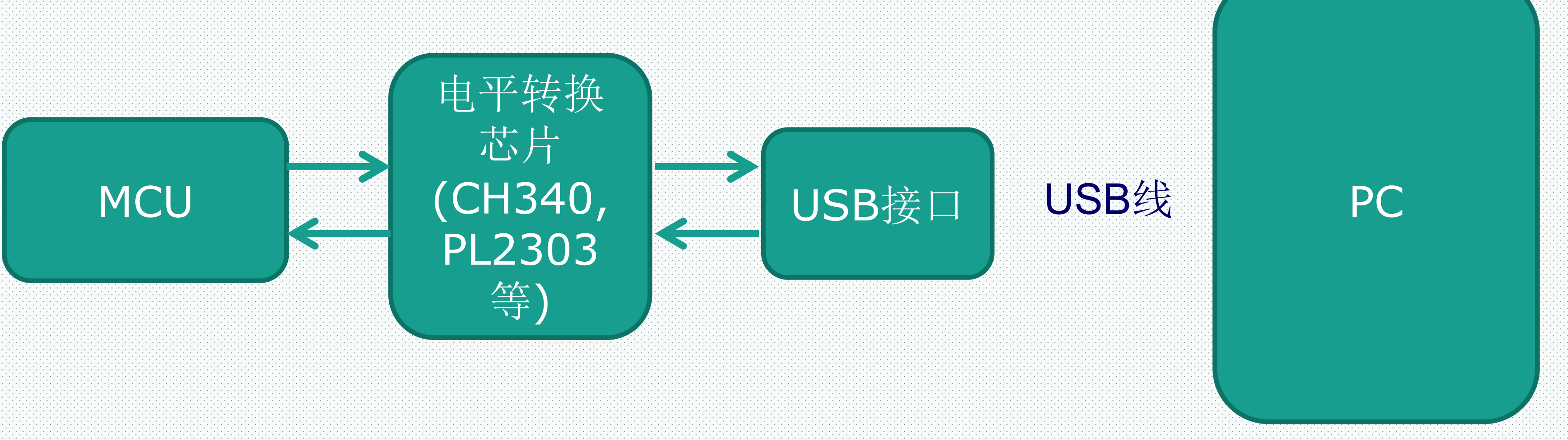
串口通信基础

◆TTL串口 & RS232 & RS485 & RS422

接口类型	逻辑1 典型值	逻辑0 典型值	说明	优缺点
TTL	+15/3.3	0	一般MCU串口引脚都支持TTL	
RS232	+15V	-15V	3线全双工，点对点	接口电平高，传输速度相对较低，传输距离近
RS485	压差+（2~6）V	压差-（2~6）V	2线半双工，点对多，主从通信。使用压差传递信号。	传输速度高可达10M，抗干扰能力强，距离远。
RS422	相对比较少用。			

串口通信基础

■ USB串口



串口通信基础

◆STM32 UART异步通信方式引脚：

串口号	RXD	TXD
1	PA10(PB7)	PA9（PB6)
2	PA3(PD6)	PA2(PD5)
3	PB11(PC11/PD9)	PB10(PC10/PD8)
4	PC11(PA1)	PC10(PA0)
5	PD2	PC12
6	PC7(PG9)	PC6(PG14)

■查看相应的芯片数据手册即可

串口通信基础

查看芯片数据手册引脚功能表：

68	101	E15	120	E2	143	E15	PA9	I/O	FT		TIM1_CH2, I2C3_SMBA, USART1_TX, DCMI_D0, EVENTOUT	OTG_FS_VBUS
69	102	D15	121	D5	144	D15	PA10	I/O	FT		TIM1_CH3, USART1_RX, OTG_FS_ID, DCMI_D1, EVENTOUT	

47	69	R12	79	M3	90	P12	PB10	I/O	FT		TIM2_CH3, I2C2_SCL, SPI2_SCK/I2S2_CK, USART3_TX, OTG_HS_ULPI_D3, ETH_MII_RX_ER, LCD_G4, EVENTOUT	
48	70	R13	80	N3	91	R13	PB11	I/O	FT		TIM2_CH4, I2C2_SDA, USART3_RX, OTG_HS_ULPI_D4, ETH_MII_TX_EN/ETH_RMII_TX_EN, LCD_G5, EVENTOUT	

目录

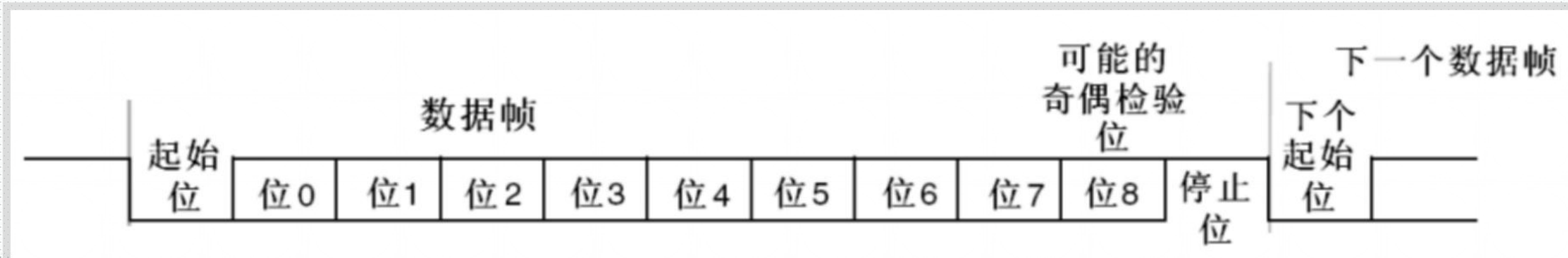
3 STM32串口数据格式和通信过程

STM32串口通信数据格式

◆STM32串口异步通信需要定义的参数：

① 起始位：1个逻辑0数据位开始
② 数据位（8位或者9位）
③ 奇偶校验位（第9位）
④ 停止位（1,1.5,2位）
⑤ 波特率设置

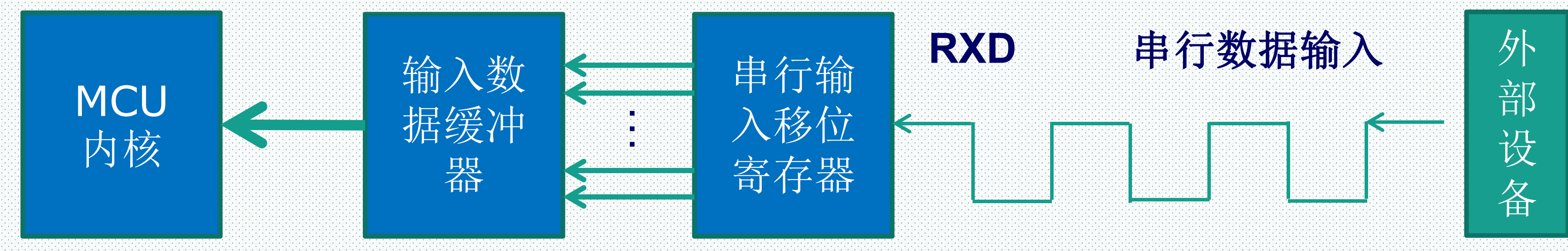
- 范例：



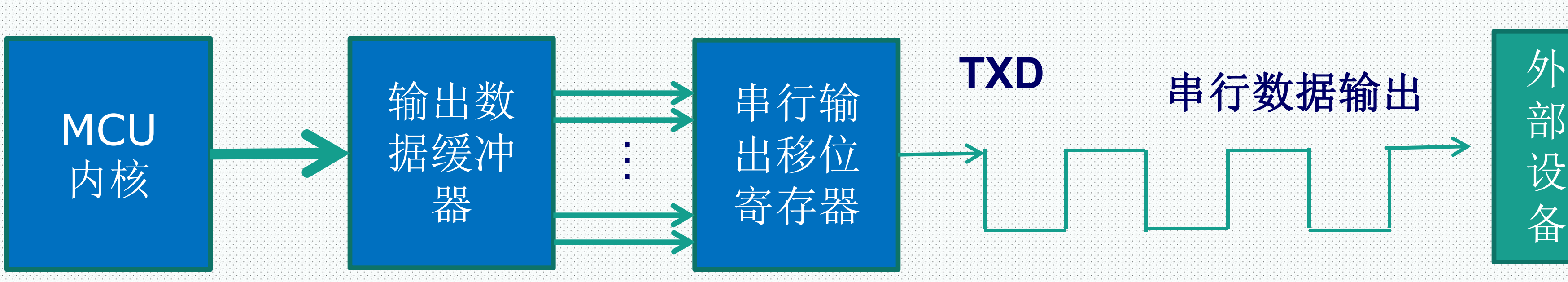
STM32串口通信过程

◆STM32串口通信过程

数据接收过程：



数据发送过程：



- 串口发送接收配置流程概述

目录

1 串口发送过程配置流程概述

2 手把手写串口发送小实验

3 串口接收过程配置流程概述

4 串口接收中断流程

寄存器描述

- **HAL库中串口寄存器定义文件：**

stm32f429xx.h	F429芯片
stm32f767xx.h	F767芯片
stm32f103xx.h	F103芯片
stm32fnnnx.x.h	其他芯片

寄存器描述

```c
typedef struct
{
  __IO uint32_t SR;
__IO uint32_t BRR;
__IO uint32_t CR1;
__IO uint32_t CR2;
__IO uint32_t CR3;
__IO uint32_t GTPR;
} USART_TypeDef;
```

M3/M4

```c
typedef struct
{
  __IO uint32_t CR1;
  __IO uint32_t CR2;
  __IO uint32_t CR3;
  __IO uint32_t BRR;
  __IO uint32_t GTPR;
  __IO uint32_t RTOR;
  __IO uint32_t RQR;
  __IO uint32_t ISR;
  __IO uint32_t ICR;
  __IO uint32_t RDR;
  __IO uint32_t TDR;
} USART_TypeDef;
```

M3/M4

寄存器描述

- **HAL库中串口函数声明定义文件：**

stm32f7xx_hal_uart.c / stm32f7xx_hal_uart.c
stm32f7xx_hal_usart.c / stm32f7xx_hal_usart.c

stm32f4xx_hal_uart.c / stm32f4xx_hal_uart.c
stm32f4xx_hal_usart.c / stm32f4xx_hal_usart.c

stm32f1xx_hal_uart.c / stm32f1xx_hal_uart.c
stm32f1xx_hal_usart.c / stm32f1xx_hal_usart.c

目录

1 串口发送过程配置流程概述

串口发送流程

■串口字节发送流程

① 编程USARTx_CR1的M位来定义字长。
② 编程USARTx_CR2的STOP位来定义停止位位数。
③ 编程USARTx_BRR寄存器确定波特率。
④ 使能USARTx_CR1的UE位使能USARTx。
⑤ 如果进行多缓冲通信，配置USARTx_CR3的DMA使能（DMAT)。具体请参考后面DMA实验。
⑥ 使能USARTx_CR1的TE位使能发送器。
⑦ 向发送数据寄存器TDR写入要发送的数据（对于M3，发送和接收共用DR寄存器）。
⑧ 向TRD寄存器写入最后一个数据后，等待状态寄存器USARTx_SR(ISR)的TC位置1，传输完成。

配置步骤（①–⑥）

发送数据（⑦–⑧）

串口发送流程

■串口字节发送流程（HAL库函数）

- 配置步骤①~⑥：配置字长，停止位，奇偶校验位，波特率等：

*HAL_StatusTypeDef HAL_UART_Init(UART_HandleTypeDef *huart);*

该函数内部会引用标识符__HAL_USART_ENABLE使能相应串口。

- 步骤⑦~⑧发送数据和等待发送完成：

*HAL_StatusTypeDef HAL_USART_Transmit(USART_HandleTypeDef *husart, uint8_t *pTxData, uint16_t Size, uint32_t Timeout);*

目录

2 手把手编写串口发送小实验

串口发送流程

■ __Weak关键字

函数前面加__weak修饰符，我们称之为弱函数。对于弱函数，用户可以在用户文件中重新定义一个同名函数，最终编译器编译的时候会选择用户定义的函数。如果用户没有定义，那么函数内容就是弱函数定义的内容。

串口发送流程

- 函数声明：

```
void HAL_UART_MspInit(UART_HandleTypeDef *huart);
```

- 函数定义（弱函数）：

```
__weak void HAL_UART_MspInit(UART_HandleTypeDef *huart)
{
}
```

- 弱函数重新被定义：

```
void HAL_UART_MspInit(UART_HandleTypeDef *huart)
{
  …//内容
}
```

串口发送流程

■ _Weak关键字的好处

① 对于事先已经定义好的一个流程，我们只希望修改流程中的某部分与用户相关的代码，这个时候我们可以采用弱函数定义一个空函数，然后让用户自行定义该函数。这样做的好处是我们不会对既有程序流程做任何修改。

② HAL库中大量使用__weak关键字修饰外设回调函数。

③ 外设回调函数供用户编写MCU相关程序，大大提高程序的通用性移植性。

串口发送流程

■ 串口发送程序配置过程(HAL库）：

① 初始化串口相关参数，使能串口：HAL_UART_Init();

② 串口相关IO口配置，复用配置：

在HAL_UART_MspInit中调用HAL_GPIO_Init函数。

③ 发送数据，并等待数据发送完成：

HAL_UART_Transmit()函数;

串口发送流程

■ 串口句柄

```
typedef struct
{
  USART_TypeDef                 *Instance;
  UART_InitTypeDef              Init;
  uint8_t                   *pTxBuffPtr;
  uint16_t                   TxXferSize;
  uint16_t                   TxXferCount;
  uint8_t                   *pRxBuffPtr;
  uint16_t                   RxXferSize;
  uint16_t                   RxXferCount;
  DMA_HandleTypeDef              *hdmatx;
  DMA_HandleTypeDef              *hdmarx;
  HAL_LockTypeDef                Lock;
  __IO HAL_UART_StateTypeDef    State;
  __IO uint32_t                 ErrorCode;
}UART_HandleTypeDef;
```

串口发送流程

■ 串口参数初始化结构体

```
typedef struct
{
  uint32_t BaudRate;
  uint32_t WordLength;
  uint32_t StopBits;
  uint32_t Parity;
  uint32_t Mode;
  uint32_t HwFlowCtl;
  uint32_t OverSampling;
}UART_InitTypeDef;
```

目录

3 串口接收过程配置流程概述

串口接收流程

■串口接收流程

① 编程USARTx_CR1的M位来定义字长。

② 编程USARTx_CR2的STOP位来定义停止位位数。

③ 编程USARTx_BRR寄存器确定波特率。

④ 使能USARTx_CR1的UE位使能USARTx。

⑤ 如果进行多缓冲通信，配置USARTx_CR3的DMA使能（DMAT)。具体请参考后面DMA实验。

⑥ 使能USARTx_CR1的RE位为1使能接收器。

⑦ 如果要使能接收中断（接收到数据后产生中断），使能USARTx_CR1的RXNEIE位为1。

配置步骤

串口接收流程

■当串口接收到数据时：

① USARTx_SR（ISR)的RXNE位置1。表明移位寄存器内容已经传输到RDR（DR）寄存器。已经接收到数据并且等待读取。

② 如果开启了接收数据中断（USARTx_CR1寄存器的RXNEIE位为1），则会产生中断。（程序上会执行中断服务函数)

③ 如果开启了其他中断（帧错误等），相应标志位会置1。

④ 读取USARTx_TDR（DR）寄存器的值，该操作会自动将RXNE位清零，等待下次接收后置位。

接收数据

串口接收流程

■串口接收流程（HAL库）

□ 配置过程

➢ 接收配置步骤①~⑥和发送流程一样，调用HAL_UART_Init函数

*HAL_StatusTypeDef HAL_UART_Init(UART_HandleTypeDef *huart);*

➢ 步骤⑦开启接收中断：

*HAL_StatusTypeDef HAL_UART_Receive_IT(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size);*

串口接收流程

接收数据过程

- 步骤①获取状态标志位通过标识符实现：

  __HAL_UART_GET_FLAG　　　　//判断状态标志位

  __HAL_UART_GET_IT_SOURCE　//判断中断标志位

- 步骤②~③中断服务函数：

void USARTx_IRQHandler(void) ;//(x=1~3,6)

void USARTx_IRQHandler(void) ;//(x=4,5,7,8)

在启动文件startup_stm32fxxx.s中查找。

- 步骤④读取接收数据：

*HAL_StatusTypeDef HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout);*

目录

4 串口接收中断流程

串口接收中断流程

■ **串口接收中断程序配置过程(HAL库)：**

① 初始化串口相关参数，使能串口：HAL_UART_Init();

② 串口相关IO口配置，复用配置：

在HAL_UART_MspInit中调用HAL_GPIO_Init函数。

③ 串口接收中断优先级配置和使能：

HAL_NVIC_EnableIRQ()；

HAL_NVIC_SetPriority();

④ 使能串口接收中断：HAL_UART_Receive_IT();

⑤ 编写中断服务函数：USARTx_IRQHandler

串口接收中断流程

经过上面6个步骤，我们就可以写完整的串口接收实验。我们就可以在中断服务函数中编写中断处理过程。

HAL库提供了详细的中断处理函数

HAL_UART_IRQHandler

我们在中断服务函数中会调用此函数处理中断。

串口接收中断流程

发生中断

USART1_IRQHandler

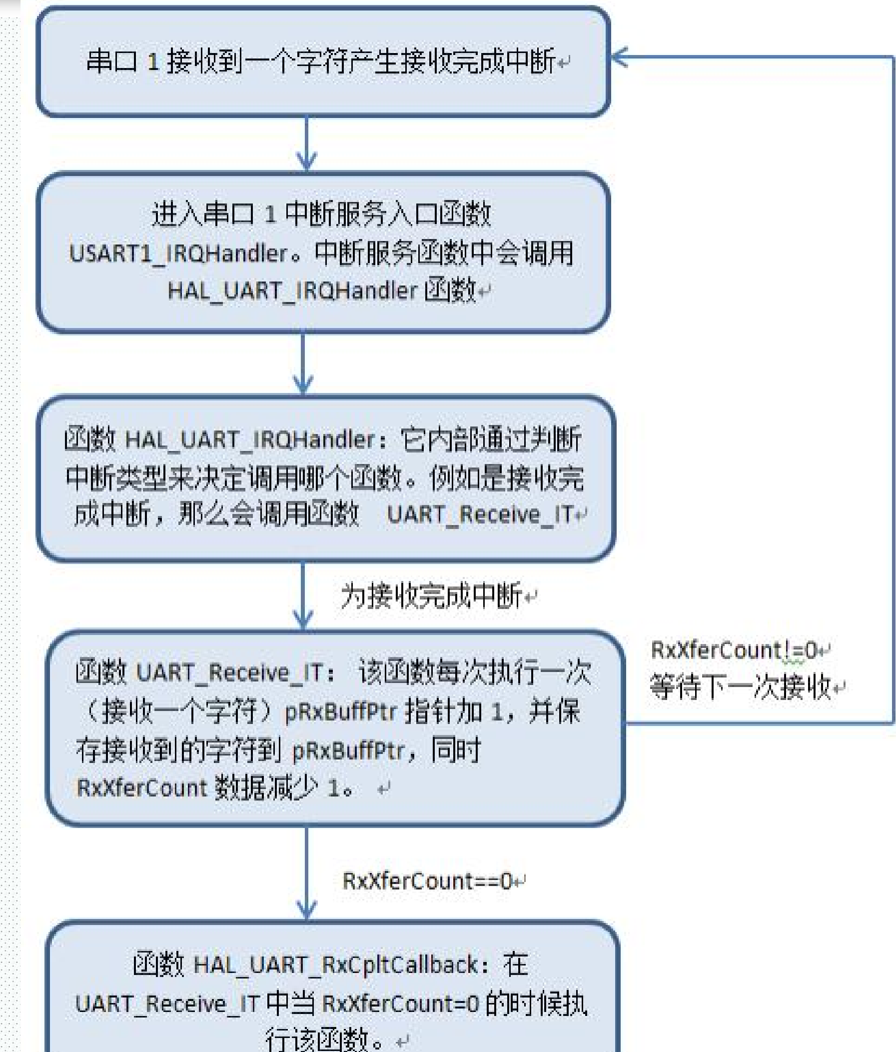
HAL_UART_IRQHandler

判断中断类型调用不同处理函数

接收中断处理 UART_Receive_IT	发送中断处理 UART_Transmit_IT	发送完成中断处理: UART_EndTransmit_IT	错误处理 HAL_UART_ErrorCallback
HAL_UART_RxCpltCallback	HAL_UART_TxCpltCallback		HAL_UART_ErrorCallback

串口接收中断流程

■ 串口接收中断流程



串口接收中断流程

■ 串口中断服务函数执行流程：

① 串口中断服务函数中调用HAL库串口中断通用处理函数：

HAL_UART_IRQHandler();

该函数会对中断来源进行分析，调用相应函数。

② 对于不同的中断类型，我们只需要编写最终的中断处理函数：

```
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart);
void HAL_UART_TxHalfCpltCallback(UART_HandleTypeDef *huart);
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart);
void HAL_UART_RxHalfCpltCallback(UART_HandleTypeDef *huart);
void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart);
```

- 串口通信实验源码讲解

串口通信实验源码讲解

Printf支持：

```
//加入以下代码,支持printf函数,而不需要选择use MicroLIB
#if 1
#pragma import(__use_no_semihosting)
//标准库需要的支持函数
struct __FILE
{ int handle;
};

FILE __stdout;
//定义_sys_exit()以避免使用半主机模式
_sys_exit(int x)
{ x = x; }

//重定义fputc函数
int fputc(int ch, FILE *f)
{
    while((USART1->SR&0X40)==0);//循环发送,直到发送完毕
    USART1->DR = (u8) ch;
   return ch;
}
#endif
```

串口通信实验源码讲解

实验现象：

从电脑串口助手发送长度为200以内任意长度的字符串给STM32串口1（字符串以回车换行标识结束），STM32接收到字符串之后，一次性通过串口1把所有数据返回给电脑。

串口通信实验源码讲解

实现过程：

把每个接收到的数据保存在一个程序定义的Buffer数组中（数组长度为200），同时把接收到的数据个数保存在定义的变量中。程序通过对接收到的每个数据进行结束判断（接收到回车0x0d之后再接收到换行0x0a），程序接收结束之后，设置相应的标记位，标记结束。。。外部 循环通过判断标志位来判断程序结束，然后一次性通过串口1发送出来。发送完成之后，所有标志位和数据量都清零。

串口通信实验源码讲解

```
#define USART_REC_LEN   200          //定义最大接收字节数 200

 u8  USART_RX_BUF[USART_REC_LEN];
          //接收缓冲,最大USART_REC_LEN个字节.末字节为换行符

 u16 USART_RX_STA;                          //接收状态标记
```

USART_RX_STA		
bit15	**bit14**	**bit13~0**
接收完成标志	接收到**0X0D**标志	接收到的有效数据个数

程序要求，发送的字符是以回车换行结束（**0x0D,0x0A)**

ABCDEFGHI…….M(0x0D),(0x0A)

- 通用定时器基本原理

通用定时器概述

◆ STM32定时器

产品型号	主频 (MHz)	内核	FLASH (KB)	RAM (KB)	EEPROM (B)	封装	通用IO	最低工作电压	最高工作电压	16位定时器	32位定时器	电机控制定时器	低功耗定时器
STM32F767IGT6	216	ARM Cortex-M7	1024	512	0	LQFP176	132	1.7	3.6	12	2	2	1
STM32F429IGT6	180	ARM Cortex-M4	1024	256	0	UFBGA176/ LQFP176	140	1.7	3.6	12	2	2	0
STM32F407ZGT6	168	ARM Cortex-M4	1024	192	0	LQFP144	114	1.73	3.6	12	2	2	0
STM32F103ZET6	72	ARM Cortex-M3	512	64	0	LFBGA144/ LQFP144	112	2	3.6	8	0	2	0
STM32F103RCT6	72	ARM Cortex-M3	256	48	0	LQFP64/ WLCSP64	51	2	3.6	8	0	2	0

通用定时器概述

◆三种（4）STM32定时器区别

定时器种类	位数	计数器模式	产生DMA请求	捕获/比较通道	互补输出	特殊应用场景
高级定时器（TIM1,TIM8)	16	向上，向下，向上/下	可以	4	有	带可编程死区的互补输出
通用定时器（TIM2,TIM5）	32	向上，向下，向上/下	可以	4	无	通用。定时计数，PWM输出，输入捕获，输出比较
通用定时器（TIM3,TIM4）	16	向上，向下，向上/下	可以	4	无	通用。定时计数，PWM输出，输入捕获，输出比较
通用定时器（TIM9~TIM14）	16	向上	没有	2	无	通用。定时计数，PWM输出，输入捕获，输出比较
基本定时器(TIM6,TIM7)	16	向上，向下，向上/下	可以	0	无	主要应用于驱动DAC

通用定时器概述

◆ 通用定时器功能特点描述

STM32的通用 TIMx (TIM2、TIM3、TIM4 和 TIM5等)定时器功能特点包括：

- 16 /32 位向上、向下、向上/向下(中心对齐)计数模式，自动装载计数器（TIMx_CNT）。
- 16 位可编程(可以实时修改)预分频器(TIMx_PSC)，计数器时钟频率的分频系数为 1～65535 之间的任意数值。
- 4 个独立通道（TIMx_CH1~4），这些通道可以用来作为：
  ① 输入捕获
  ② 输出比较
  ③ PWM 生成(边缘或中间对齐模式)
  ④ 单脉冲模式输出
- **可使用外部信号（TIMx_ETR）控制定时器和定时器互连（可以用 1 个定时器控制另外一个定时器）的同步电路。**

通用定时器概述

■**如下事件发生时产生中断/DMA（6个独立的IRQ/DMA请求生成器）：**

① 更新：计数器向上溢出/向下溢出，计数器初始化(通过软件或者内部/外部触发)

② 触发事件(计数器启动、停止、初始化或者由内部/外部触发计数)

③ 输入捕获

④ 输出比较

⑤ 支持针对定位的增量(正交)编码器和霍尔传感器电路

⑥ 触发输入作为外部时钟或者按周期的电流管理

●STM32 的通用定时器可以被用于：测量输入信号的脉冲长度(输入捕获)或者产生输出波形(输出比较和 PWM)等。

●使用定时器预分频器和 RCC 时钟控制器预分频器，脉冲长度和波形周期可以在几个微秒到几个毫秒间调整。STM32 的每个通用定时器都是完全独立的，没有互相共享的任何资源。

通用定时器概述

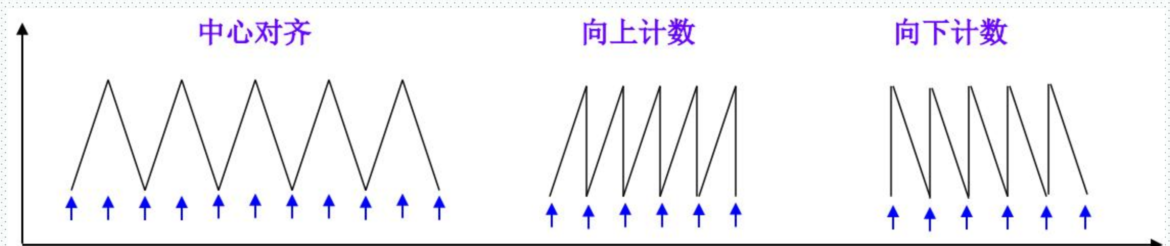
◆计数器模式

通用定时器可以向上计数、向下计数、向上向下双向计数模式。

①向上计数模式：计数器从0计数到自动加载值(TIMx_ARR)，然后重新从0开始计数并且产生一个计数器溢出事件。

②向下计数模式：计数器从自动装入的值(TIMx_ARR)开始向下计数到0，然后从自动装入的值重新开始，并产生一个计数器向下溢出事件。

③中央对齐模式（向上/向下计数）：计数器从0开始计数到自动装入的值-1，产生一个计数器溢出事件，然后向下计数到1并且产生一个计数器溢出事件；然后再从0开始重新计数。



通用定时器概述

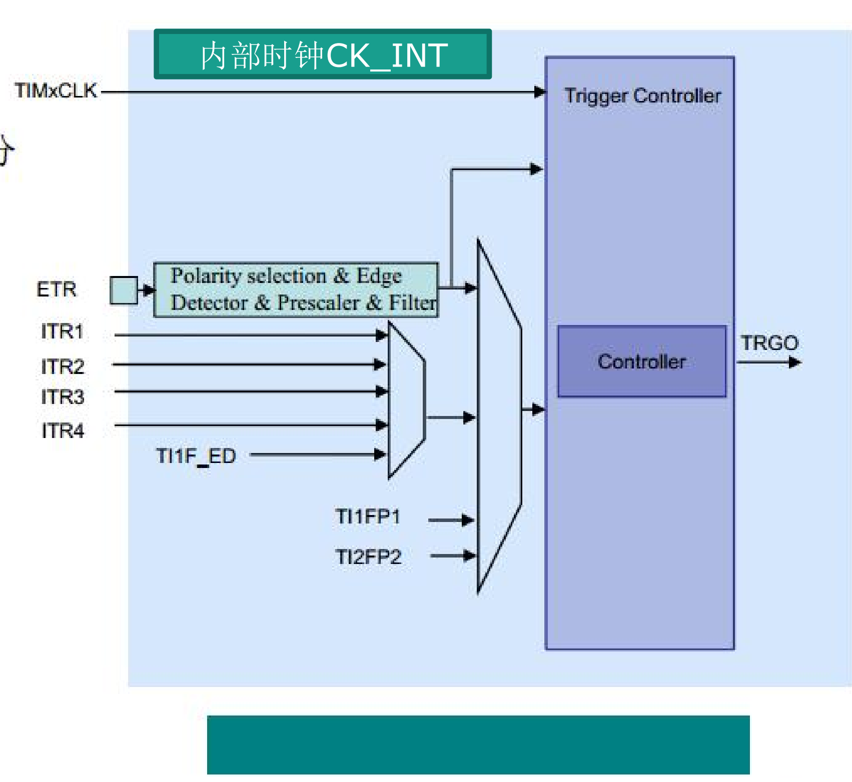
◆ 通用定时器(TIM2,3,4,5)工作过程：

内部时钟 (CK_INT)
来自RCC的 TIMxCLK
TIMx_ETR
ETR
ETRP
ETRF
极性选择、边沿检测和预分频器
输入滤波
触发控制器
TRGO
至其它定时器
至 DAC/ADC
ITR0
ITR1
ITR2
ITR3
ITR
TRC
TGI
TRGI
TI1F_ED
从模式控制器
复位、使能、向上/向下、计数
TI1FP1
TI2FP2
编码器接口
U
自动重装载寄存器
UI
停止、清除或向上/向下
U
CK_PSC
PSC 预分频器
CK_CNT
+/-
CNT 计数器
异或
TI1
输入滤波器和边沿检测器
TI1FP1
TI1FP2
IC1
预分频器
IC1PS
CC1I
U
捕获/比较1寄存器
CC1I
OC1REF
输出控制
OC1
TIMx_CH1
TRC
TIMx_CH1
TIMx_CH2
TI2
输入滤波器和边沿检测器
TI2FP1
TI2FP2
IC2
预分频器
IC2PS
CC2I
U
捕获/比较2寄存器
CC2I
OC2REF
输出控制
OC2
TIMx_CH2
TRC

通用定时器概述

计数时钟的选择

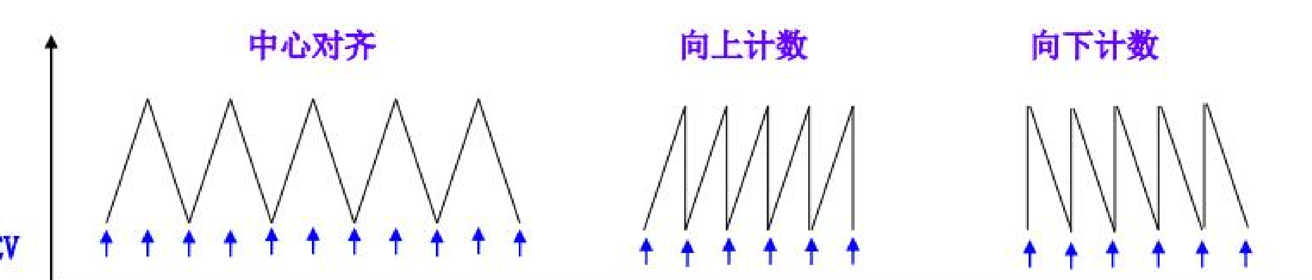
- 计数器的时钟有8种选择：
  - 内部RCC提供的时钟：TIMxCLK
  - 内部触发输入口1～4：
    - ITR1 / ITR2 / ITR3 / ITR4
    - 用一个定时器作为另一定时器的分频器
  - 外部捕捉比较引脚
    - 引脚1：TI1FP1或TI1F_ED
    - 引脚2：TI2FP2
  - 外部引脚：ETR
    - 使能/禁止位
    - 可编程设定极性
    - 4位外部触发过滤器
    - 外部触发分频器：
      - 分频器关闭
      - 二分频
      - 四分频
      - 八分频



通用定时器概述

时基单元

- 计数器寄存器(TIMx_CNT)
  - 向上计数、向下计数或者中心对齐向计数；
- 预分频器寄存器 (TIMx_PSC)
  - 可将时钟频率按1到65536之间的任意值进行分频，可在运行时改变其设置值；
- 自动装载寄存器 (TIMx_ARR)
  - 如果TIM1_CR1寄存器中的ARPE位为0，ARR寄存器的内容将直接写入影子寄存器；如果ARPE为1， ARR寄存器的内容将在每次的更新事件UEV发生时，传送到影子寄存器；
  - 如果TIM1_CR1中的UDIS位为0，当计数器产生溢出条件时，产生更新事件；



通用定时器概述

更新事件

- 将预载寄存器的内容写入影子寄存器(通过自动重载位是否被使能来决定):
  - 立即
  - 在每次更新事件发生时
- 产生更新事件的条件:
  - 当计数器上溢或下溢时,
  - 当循环计数器计数值为0时(仅适用于TIM1),
  - 通过软件设置UG(Update Generation)位。
- 更新事件的请求源可以从下面选择:
  - URS = 1 --- 仅当计数器到达上溢/下溢时,将发出更新请求;
  - URS = 0 --- 计数器的上溢/下溢、更新位的设置或从模式控制器产生的更新,将发出更新请求。

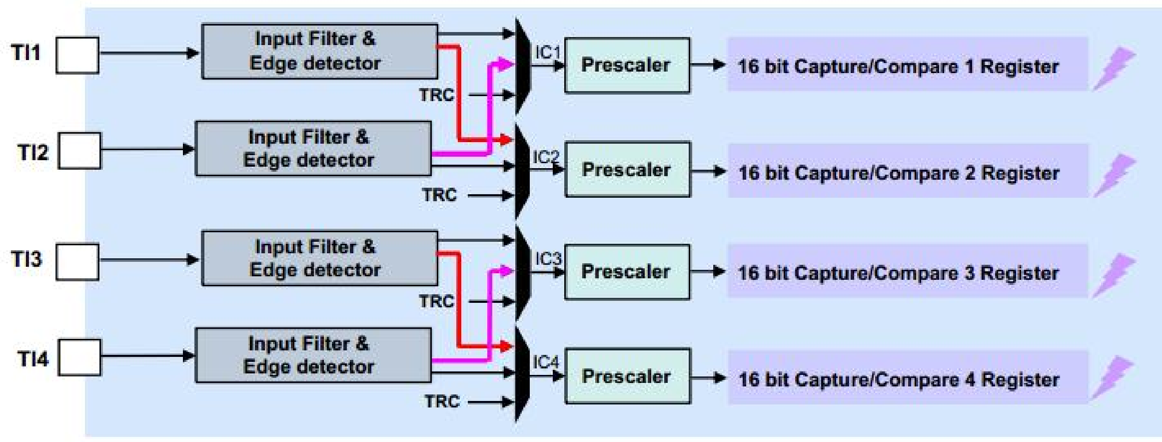
通用定时器概述

捕捉比较阵列介绍

- 捕捉比较阵列包括：
  - 每个定时器拥有4个同样的捕捉比较通道；
- 可编程设定通道的方向：输入/输出
- 每个通道由以下部分组成：
  - 捕捉/比较寄存器
  - 针对捕捉的输入部分：
    - 4位数字滤波器
    - 输入捕捉分频器：
      - 检测到每个边沿完成捕捉
      - 每产生2个事件完成捕捉
      - 每产生4个事件完成捕捉
      - 每产生8个事件完成捕捉
  - 针对比较的输出部分：
    - 比较器
    - 输出控制

通用定时器概述

输入捕捉模式（1）

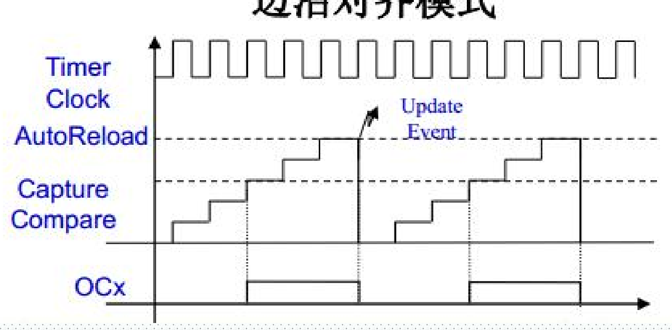


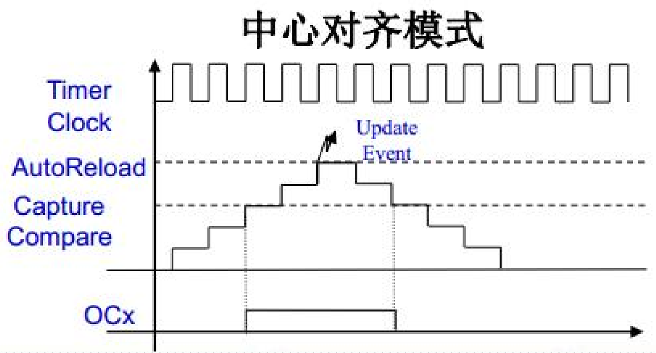
- IC1、2和IC3、4可以分别通过软件设置将其映射到TI1、TI2和TI3、TI4；
- 4个16位捕捉比较寄存器可以编程用于存放检测到对应的每一次输入捕捉时计数器的值；
- 当产生一次捕捉，相应的CCxIF标志位被置1；同时如果中断或DMA请求使能，则产生中断或DMA请求。
- 如果当CCxIF标志位已经为1，当又产生一个捕捉，则捕捉溢出标志位CCxOF将被置1。

通用定时器概述

PWM模式

- PWM模式运行产生：
  - 定时器2、3和4可以产生4独立的信号
  - 频率和占空比可以进行如下设定：
    - 一个自动重载寄存器用于设定PWM的周期；
    - 每个PWM通道有一个捕捉比较寄存器用于设定占空时间。
  - ➔ 例如：产生一个40KHz的PWM信号：在定时器2的时钟为72MHz下，占空比为50% ：
    - 预分频寄存器设置为0（计数器的时钟为TIM1CLK/(0+1))，自动重载寄存器设为1799，CCRx寄存器设为899。
- 两种可设置PWM模式：
  - 边沿对齐模式
  - 中心对齐模式





- 定时器中断原理与配置

目录

1 通用定时器知识回顾

2 常用寄存器和库函数配置

3 手把手写定时器中断实验

目录

1 通用定时器知识回顾

通用定时器概述

◆ 通用定时器工作过程：

内部时钟 (CK_INT)
来自RCC的TIMxCLK
TIMx_ETR
ETR
极性选择和边沿检测器和预分频器
ETRP
输入滤波器
ETRF
ITR0
ITR1
ITR2
ITR3
ITR
TRC
TI1F_ED
TGI
TRGI
触发控制器
TRGO
到其它定时器
到 DAC/ADC
从模式控制器
复位、使能、递增 / 递减、计数
TI1FP1
TI2FP2
编码器接口
U
自动重载寄存器
UI
停止、清零或递增 / 递减
CK_PSC
PSC
预分频器
CK_CNT
+/-
CNT
计数器
异或
TI1
输入滤波器和边沿检测器
TI1FP1
TI1FP2
TRC
IC1
预分频器
IC1PS
CC1I
捕获 / 比较 1 寄存器
OC1REF
输出控制
OC1
TIMx_CH1
TIMx_CH1
TIMx_CH2
TI2
输入滤波器和边沿检测器
TI2FP1
TI2FP2
TRC
IC2
预分频器
IC2PS
CC2I
捕获 / 比较 2 寄存器
OC2REF
输出控制
OC2
TIMx_CH2

✓ 定时器中断实验

◆时钟选择

计数器时钟可以由下列时钟源提供：

① 内部时钟(CK_INT)

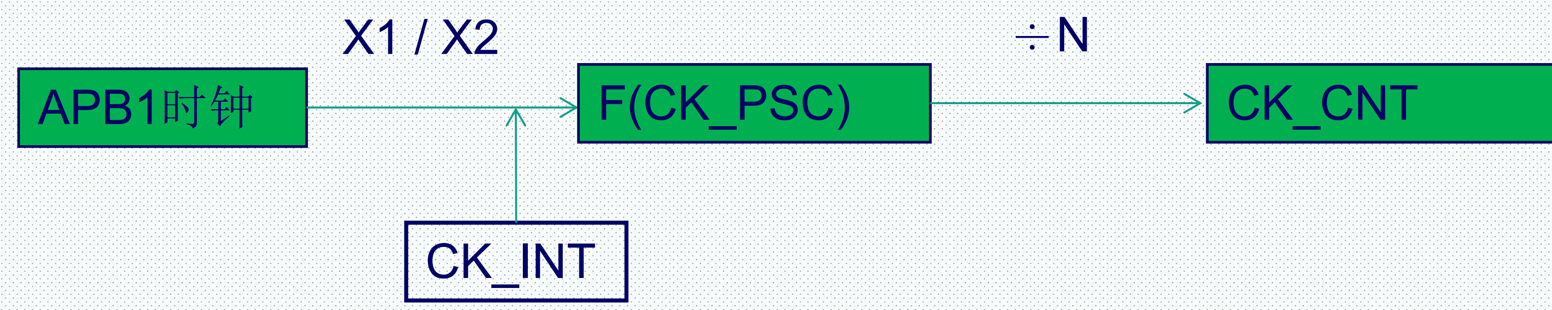
② 外部时钟模式1：外部输入脚(TIx)

③ 外部时钟模式2：外部触发输入(ETR)(仅适用TIM2,3,4)

④ 内部触发输入(ITRx)：使用一个定时器作为另一个定时器的预分频器，如可以配置一个定时器Timer1而作为另一个定时器Timer2的预分频器。

定时器中断

◆内部时钟选择



除非APB1的分频系数是1，否则通用定时器的时钟等于APB1时钟的2倍。

以F407为例，默认调用时钟初始化函数之后：
SYSCLK=168M
AHB时钟=168M
APB1时钟=42M
所以APB1的分频系数=AHB/APB1时钟=4
所以，通用定时器时钟CK_INT=2*42M=84M

定时器中断

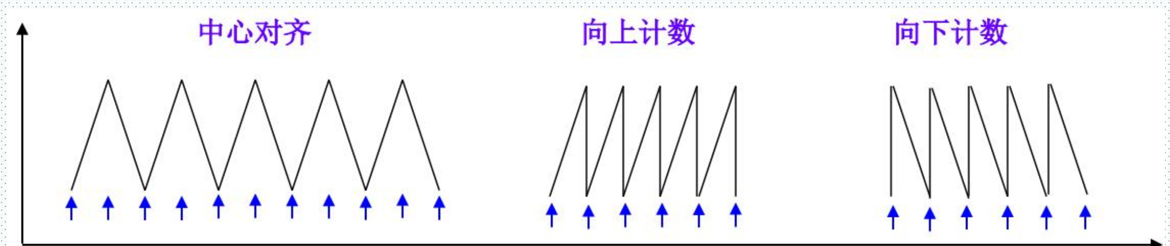
◆计数器模式

通用定时器可以向上计数、向下计数、向上向下双向计数模式。

①向上计数模式：计数器从0计数到自动加载值(TIMx_ARR)，然后重新从0开始计数并且产生一个计数器溢出事件。

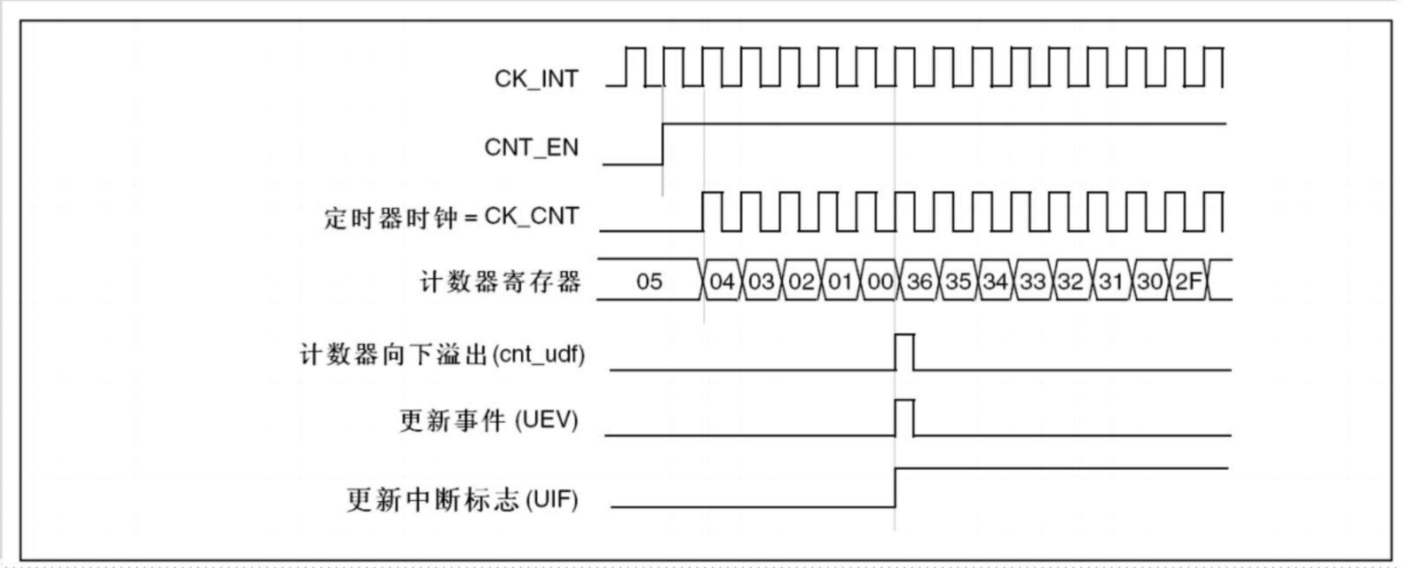
②向下计数模式：计数器从自动装入的值(TIMx_ARR)开始向下计数到0，然后从自动装入的值重新开始，并产生一个计数器向下溢出事件。

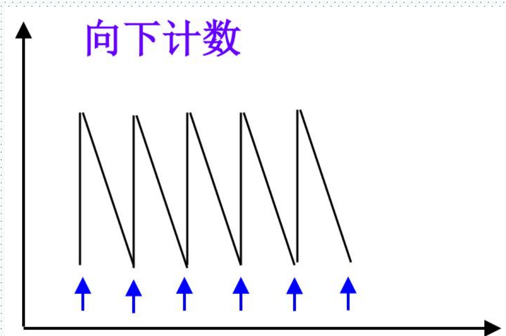
③中央对齐模式（向上/向下计数）：计数器从0开始计数到自动装入的值-1，产生一个计数器溢出事件，然后向下计数到1并且产生一个计数器溢出事件；然后再从0开始重新计数。



定时器中断

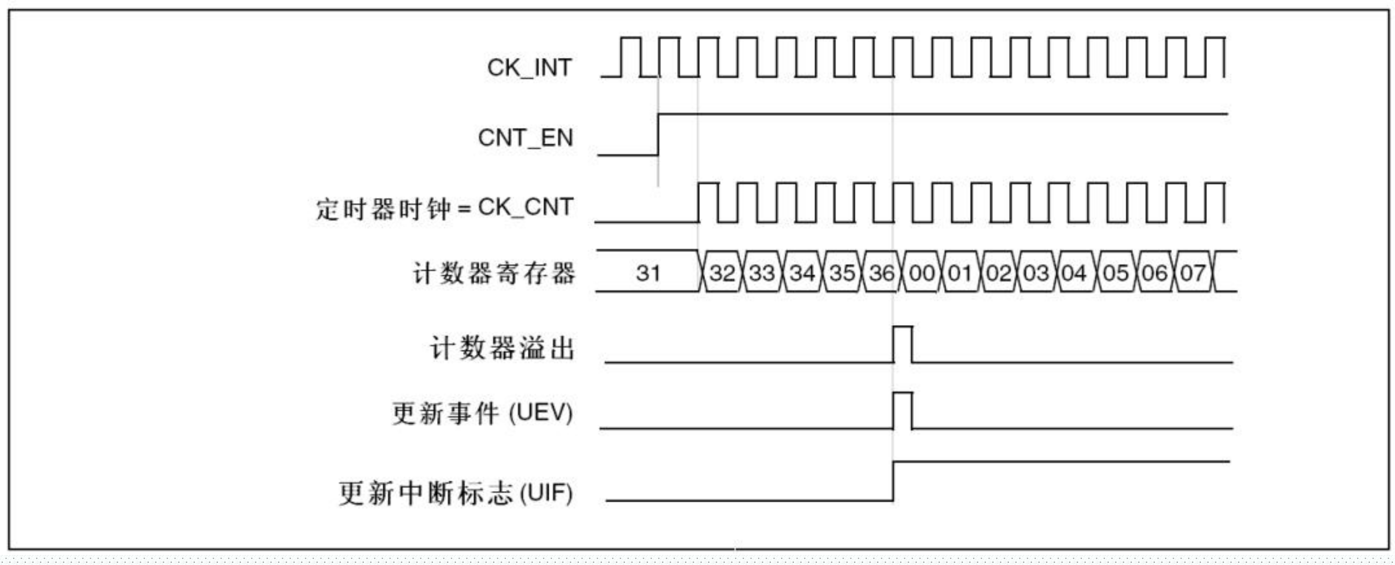
◆ 向下计数模式（时钟分频因子=1）

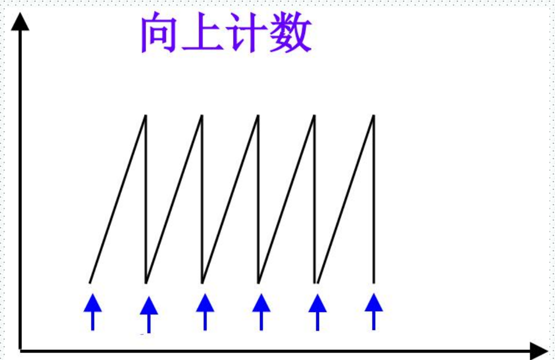




定时器中断

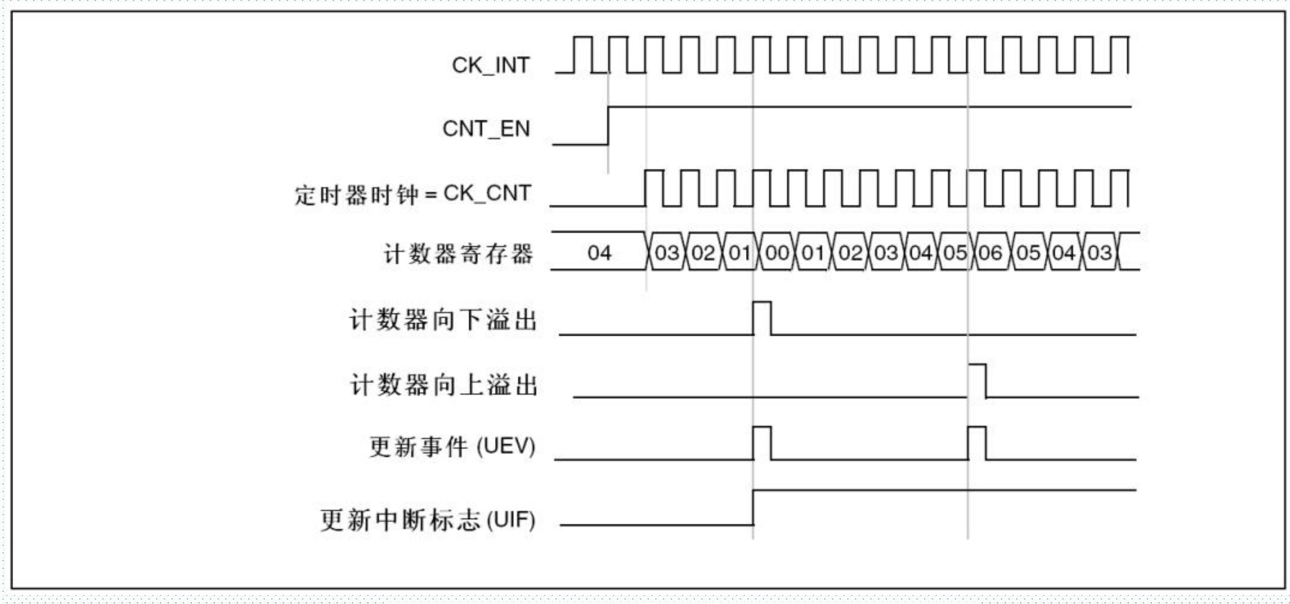
- 向上计数模式（时钟分频因子=1）

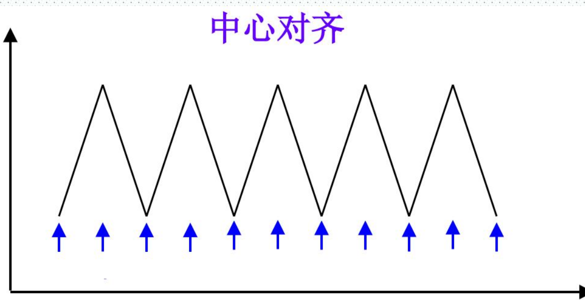




定时器中断

◆ 中央对齐计数模式（时钟分频因子=1 ARR=6）



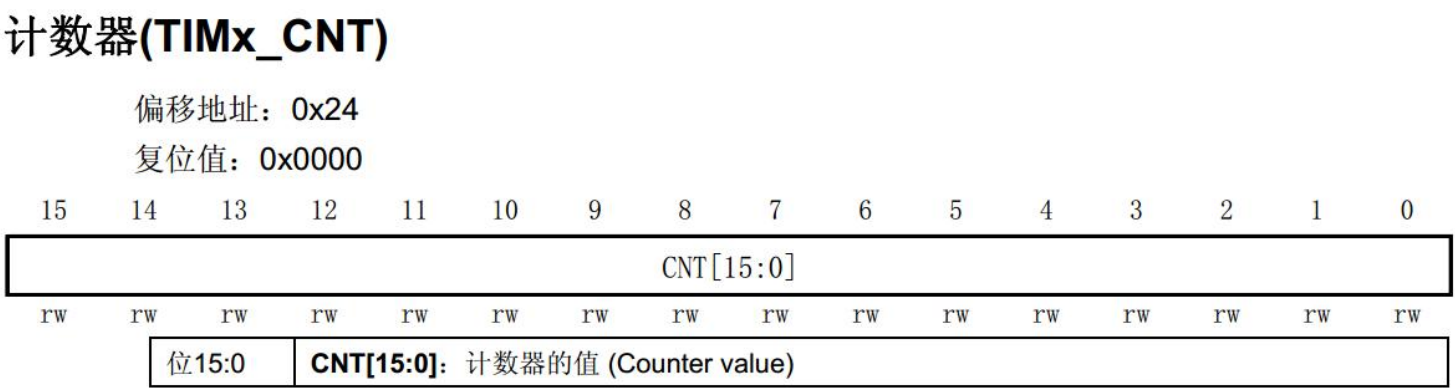


目录

2 常用寄存器和库函数配置

通用定时器常用寄存器和库函数

◆计数器当前值寄存器CNT



计数器(TIMx_CNT)

偏移地址：0x24

复位值：0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位15:0	**CNT[15:0]**：计数器的值 (Counter value)

通用定时器常用寄存器和库函数

◆预分频寄存器TIMx_PSC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位15:0	**PSC[15:0]**：预分频器的值 (Prescaler value) 计数器的时钟频率CK_CNT等于f_{CK_PSC}/(PSC[15:0]+1)。 PSC包含了当更新事件产生时装入当前预分频器寄存器的值。

通用定时器常用寄存器和库函数

◆自动重装载寄存器（TIMx_ARR)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位15:0	**ARR[15:0]**: 自动重装载的值 (Auto reload value) ARR包含了将要传送至实际的自动重装载寄存器的数值。 详细参考14.3.1节：有关ARR的更新和动作。 当自动重装载的值为空时，计数器不工作。

通用定时器常用寄存器和库函数

◆控制寄存器1（TIMx_CR1）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留						CKD[1:0]		ARPE	CMS[1:0]		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位	说明
位4	**DIR**：方向 (Direction) 0：计数器向上计数； 1：计数器向下计数。 注：当计数器配置为中央对齐模式或编码器模式时，该位为只读。
位0	**CEN**：使能计数器 0：禁止计数器； 1：使能计数器。 注：在软件设置了CEN位后，外部时钟、门控模式和编码器模式才能工作。触发模式可以自动地通过硬件设置CEN位。 在单脉冲模式下，当发生更新事件时，CEN被自动清除。

通用定时器常用寄存器和库函数

◆DMA中断使能寄存器（TIMx_DIER）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
保留	TDE	保留	CC4DE	CC3DE	CC2DE	CC1DE	UDE	保留	TIE	保留	CC4IE	CC3IE	CC2IE	CC1IE	UIE
	rw		rw	rw	rw	rw	rw		rw		rw	rw	rw	rw	rw

位	说明
位5	保留，始终读为0。
位4	**CC4IE**：允许捕获/比较4中断 (Capture/Compare 4 interrupt enable) 0：禁止捕获/比较4中断； 1：允许捕获/比较4中断。
位3	**CC3IE**：允许捕获/比较3中断 (Capture/Compare 3 interrupt enable) 0：禁止捕获/比较3中断； 1：允许捕获/比较3中断。
位2	**CC2IE**：允许捕获/比较2中断 (Capture/Compare 2 interrupt enable) 0：禁止捕获/比较2中断； 1：允许捕获/比较2中断。
位1	**CC1IE**：允许捕获/比较1中断 (Capture/Compare 1 interrupt enable) 0：禁止捕获/比较1中断； 1：允许捕获/比较1中断。
位0	**UIE**：允许更新中断 (Update interrupt enable) 0：禁止更新中断； 1：允许更新中断。

目录

3 手把手写定时器中断实验

通用定时器HAL库函数

- 通用定时器基本函数和定义所在文件：

  Stm32fxxx_hal_tim.c

  Stm32fxxx_hal_tim.h

通用定时器HAL库函数

◆定时器时基部分常用函数：

①定时器时基参数初始化函数：

HAL_StatusTypeDef HAL_TIM_Base_Init(TIM_HandleTypeDef *htim);

```
typedef struct
{
  uint32_t Prescaler;       //预分频系数
  uint32_t CounterMode;   //计数模式：向上/下
  uint32_t Period;      //自动装载值
  uint32_t ClockDivision;  //时钟分频因子：定时器时钟与数字滤波器分频比
  uint32_t RepetitionCounter; //重复计数次数：高级定时器使用
} TIM_Base_InitTypeDef;
```

通用定时器HAL库函数

②定时器时基参数初始化回调函数：

```
void HAL_TIM_Base_MspInit(TIM_HandleTypeDef *htim);
```

主要用来编写定时器时钟使能，以及中断优先级。

```
__HAL_RCC_TIM3_CLK_ENABLE();//定时器3时钟使能
```

③使能定时器：

```
HAL_StatusTypeDef HAL_TIM_Base_Start(TIM_HandleTypeDef *htim)

HAL_StatusTypeDef HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim)
```

此函数同时开启了定时器更新中断

通用定时器HAL库函数

④定时器中断通用处理函数：

```
void HAL_TIM_IRQHandler(TIM_HandleTypeDef *htim);
```

该函数被中断服务函数调用。是定时器中断处理通用入口函数，通过对中断类型进行分析判断，调用对应的回调函数。

通用定时器HAL库函数

⑤定时器中断处理回调函数：

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim);
void HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef *htim)
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
void HAL_TIM_PWM_PulseFinishedCallback(TIM_HandleTypeDef *htim)
void HAL_TIM_TriggerCallback(TIM_HandleTypeDef *htim)
void HAL_TIM_ErrorCallback(TIM_HandleTypeDef *htim)
```

定时器中断实现步骤

◆ 定时器中断实现步骤

① 使能定时器时钟。

__HAL_RCC_TIM3_CLK_ENABLE();

② 初始化定时器，配置**ARR,PSC**。

HAL_TIM_Base_Init();

③ 开启定时器/中断。

HAL_TIM_Base_Start();

HAL_TIM_Base_Start_IT();

④ 设置中断优先级。

HAL_NVIC_SetPriority();   *HAL_NVIC_EnableIRQ();*

⑤ 编写中断服务函数。

TIMx_IRQHandler();//中断服务函数

HAL_TIM_IRQHandler();//中断处理入口函数

HAL_TIM_PeriodElapsedCallback();//定时器更新中断回调函数

手把手写定时器中断实验

◆程序要求

通过定时器中断配置，每500ms中断一次，然后中断服务函数中控制LED1实现LED1状态取反（闪烁）。

Tout（溢出时间）=（ARR+1)(PSC+1)/Ftclk

GO!!

LCD显示实验

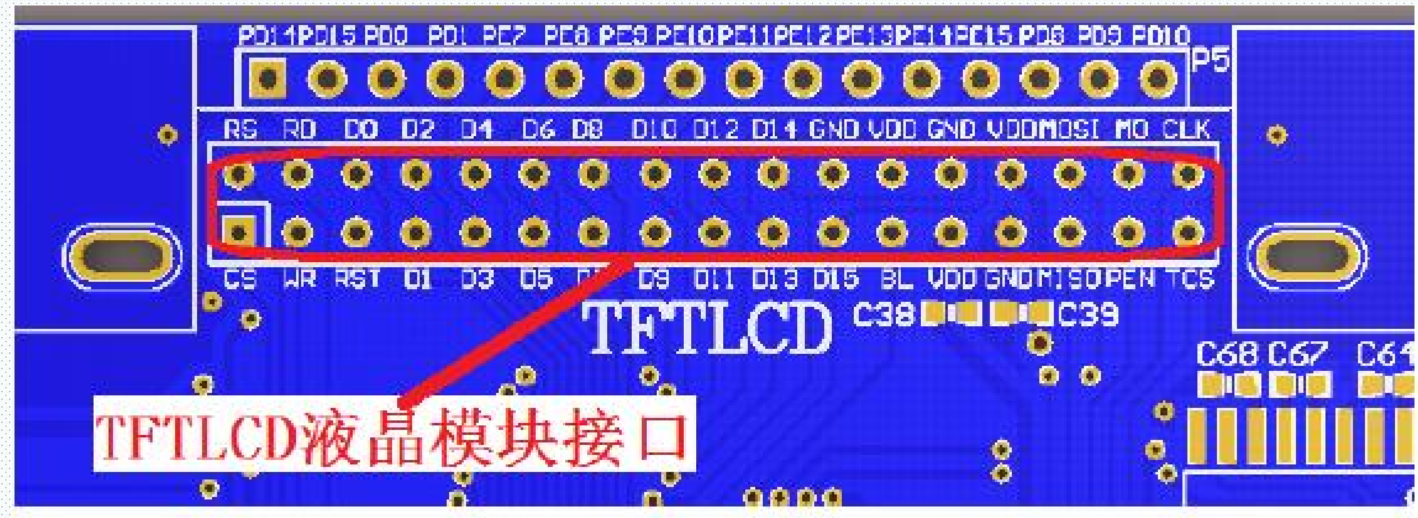
实验目的

1、了解LCD显示原理。

2、掌握LCD控制器NT35510的驱动方法。

实验原理

LCD硬件连接图



管脚对应关系：

LCD_ BL(背光控制)对应 PB0;

LCD_CS 对应 PG12 即 FSMC_NE4 ;

LCD_RS 对应 PF12 即 FSMC_A6;

LCD_WR 对应 PD5 即 FSMC_NWE;

LCD_RD 对应 PD4 即 FSMC_NOE E;

LCD_D[15:0]则直接连在 FSMC_D15~D0;

实验原理

LCD驱动流程

硬复位
LCD_RST=0;
delay_ms(100);
LCD_RST=1;

初始化序列

设置坐标

写GRAM指令

写入颜色数据

LCD显示

读GRAM指令

读出颜色数据

单片机处理

图 18.1.1.5 TFTLCD 使用流程

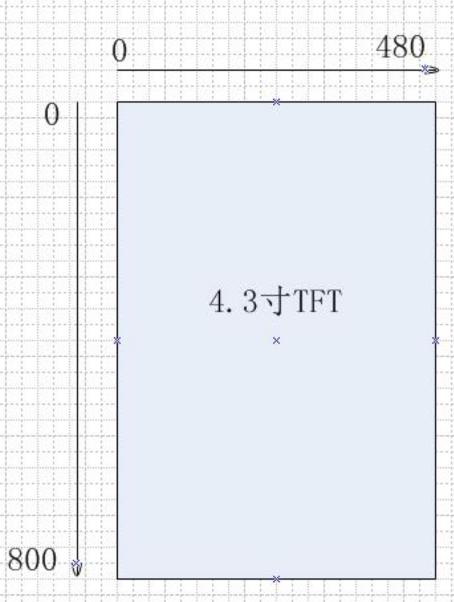
实验原理

LCD接口设置方法

1）设置STM32F4 与TFTLCD 模块相连接的IO。

2）初始化TFTLCD 模块。

3）通过函数将字符和数字显示到TFTLCD 模块上。



实验内容

1.编写程序在LCD上画点。

2.编写程序在LCD上画直线。

3.编写程序在LCD上画圆。

4.编写程序在LCD上画奥运五环。

5.编写程序在LCD上画一条直线，并且让直线绕中心进行360度旋转。

提示：需要用到SIN,COS数学函数，形参为弧度。

```
extern _ARMABI float sinf(float /*x*/);
extern _ARMABI float cosf(float /*x*/);
```

奥运五环效果图

触摸屏实验

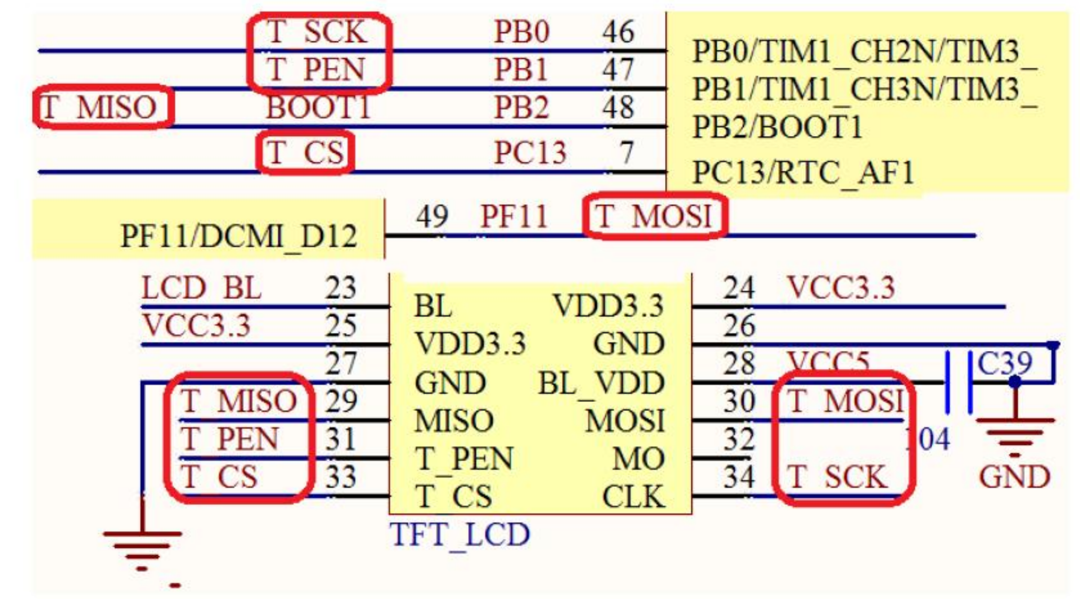
实验目的

1、了解触摸屏原理。

2、掌握触摸屏控制器GT9147的驱动方法。

实验原理

触摸屏硬件连接图



管脚对应关系：

T_PEN(CT_INT):中断输出信号；

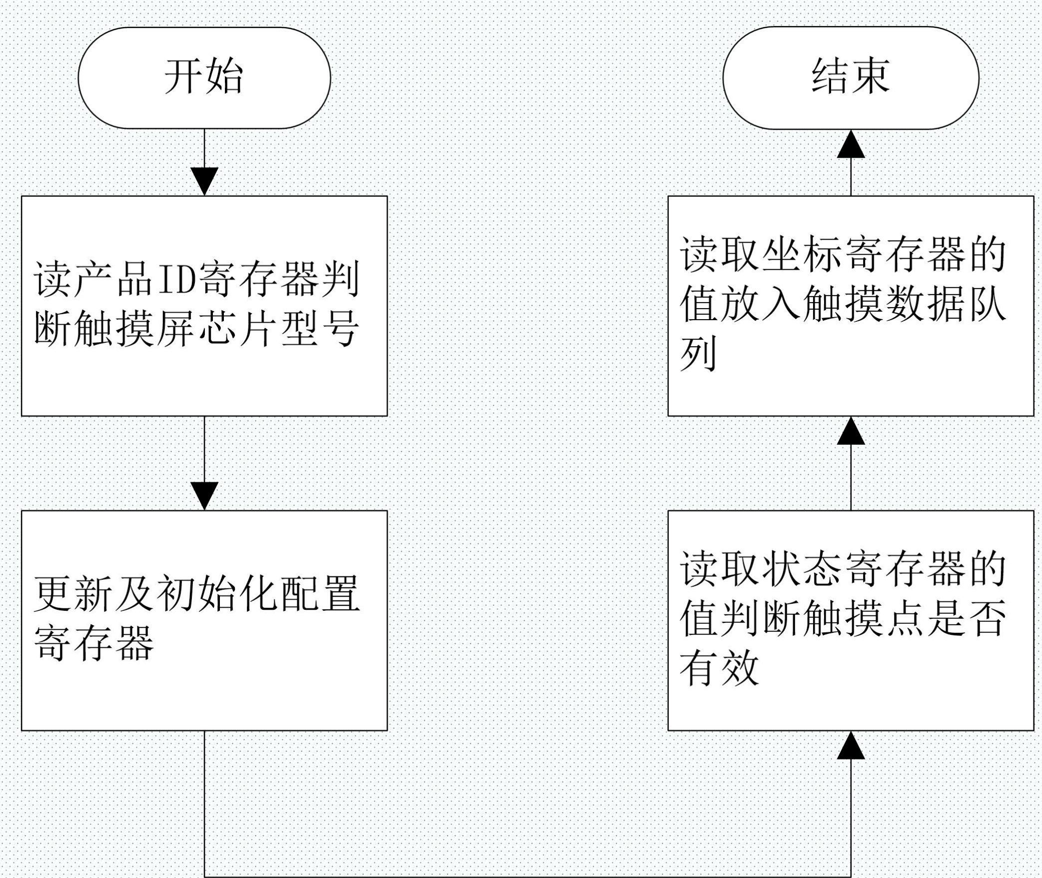
T_CS(CT_RST):复位信号；

T_CLK(CT_SCL):IIC接口的SCL信号；

T_MOSI(CT_SDA):IIC接口的SDA信号；

实验原理

触摸屏驱动流程



实验原理

触摸屏接口设置方法

1）设置STM32F4 与触摸屏驱动模块GT9147相连接的IO。

2）初始化GT9147模块。

3）通过查询方式读取GT9147的寄存器（0X814E）判断是否有有效触点。

tp_dev数据结构

```
31  //触摸屏控制器
32  typedef struct
33  {
34      u8  (*init)(void);          //初始化触摸屏控制器
35      u8  (*scan)(u8);            //扫描触摸屏.0,屏幕扫描;1,物理坐标;
36      void (*adjust)(void);       //触摸屏校准
37      u16 x[CT_MAX_TOUCH];        //当前坐标
38      u16 y[CT_MAX_TOUCH];        //电容屏有最多5组坐标,电阻屏则用x[0],y[0]代表:此次扫描时,触屏的坐标,用
39                                  //x[4],y[4]存储第一次按下时的坐标.
40      u8  sta;                //笔的状态
41                          //b7:按下1/松开0;
42                          //b6:0,没有按键按下;1,有按键按下.
43                          //b5:保留
44                          //b4~b0:电容触摸屏按下的点数(0,表示未按下,1表示按下)
45  /////////////////////触摸屏校准参数(电容屏不需要校准)//////////////////////
46      float xfac;
47      float yfac;
48      short xoff;
49      short yoff;
50  //新增的参数,当触摸屏的左右上下完全颠倒时需要用到.
51  //b0:0,竖屏(适合左右为X坐标,上下为Y坐标的TP)
52  //   1,横屏(适合左右为Y坐标,上下为X坐标的TP)
53  //b1~6:保留.
54  //b7:0,电阻屏
55  //   1,电容屏
56      u8 touchtype;
57  }_m_tp_dev;
58
```

实验内容

1. 编写程序利用触摸屏在LCD上画线。
2. 编写程序点击LCD上显示的按钮，以触发LED灯和蜂鸣器事件。