

## **\*\*day01 计算机硬件基础**

### **\*\*D1-01 底层课程导学**

计算机 底层课程 觉得底层课程主要就是讲设备驱动，将cpu对设备的控制封装成函数，给上层软件使用；封装具有联合性、套路性，这就要学装系统（系统移植），驱动了（驱动初级、高级）；控制通过改寄存器数值实现，这就要学看芯片手册和电路原理图，看懂设备功能的连接和寄存器的功能用法

（Arm体系结构与接口技术 CPU组成原理与接口技术 觉得cpu组成原理是铺垫，理解记住有印象就行，重点还是接口技术 需要看连接、学寄存器使用）。

（操作系统就是把对硬件的操作，做成函数，然后给应用层用，分为进程的，内存的，文件系统的，网络的，设备驱动的。）

### **D1-02 ARM体系结构与接口技术课程导学**

Arm体系结构与接口技术，其实就是CPU组成原理与接口技术。CPU组成原理就是讲CPU组成和它们是如何配合着执行程序（处理数据）的，（重点在后者 明白后者前者自然明白了）；组成包括控制器 运算器（指令集）、寄存器（存储模型）等；配合执行程序包括流水线（工作模式），异常机制等

（学好了可以知道怎么优化代码，让其在底层执行的更快；老板知道工人怎么干活才能进行指导呀）。接口技术，之前讲了，就是学看连接，学寄存器使用的（觉得讲的就是CPU怎么控制硬件的，主要就是通过修改硬件控制器中的寄存器来控制）。（再看课程改，或者ai问问）（和驱动的区别是，驱动是配合操作系统去控制硬件，这里是直接控制硬件？）（计算机有了这样的接口，各种牌子的相关设备就不用再自己配接口了，节省资源？）

### **D1-03 计算机基础知识**

### **D1-04 多级存储结构与地址空间**

计算机 存储器体系结构与原理 存储存储器体系包括内存外存高速缓存，外存就是长期存放程序和数据的，内存就是运行时存放程序和数据的，高速缓存也是运行时存放程序和数据的，就是常用的。CPU可以控制程序和数据。从外存转移到内存，然后转移到高速缓存。这样建立体系结构的话是平衡了价钱，速度和容量，速度快往往价钱高，容量低（觉得和寻址空间大小相关大一些）。（CPU控制的话，它通过控制线能够传输指令和地址，对吗）（高速缓存是在哪儿呢？在CPU上还是内存上？）

（觉得重要性不大，暂不主动记；上面讲课程主要讲了什么更重要）

## D1-05 CPU工作原理概述

（加上03，觉得应该是计算机组成原理）

计算机 计算机组成原理，主要讲计算机组成和它们是如何配合着工作的。组成包括cpu，内存，硬盘，输入设备，输出设备。中央处理器就是取指，译指，执行的。内存就是临时放指令和数据的。硬盘就是长期放质量和数据的。输入设备就是把其他物理信号转化成01信号。输出就是把01信号转换成其他物理信号。（总起来就是根据读取的程序、数据、状态来执行，然后输出内容，同时可能改变程序、数据、状态。觉得还有些模糊，以后在找资料研究再看吧）（觉得重要性不大，暂不主动记；上面讲课程主要讲了什么更重要）

## \*\*day02 ARM处理器概论

### \*\*D2-01 ARM处理器概述

a系列r系列m系列，早期经典产品，应用高级应用手机，平板，实时性刹车系统，微控制器 洗衣机，手环，手表。精简指令集，减去80%不常用的复杂指令，需要时用简单指令来替代。

### D2-02 ARM指令集概述

计算机 Arm指令集概述 指令就是处理器能识别执行的操作。这些操作的结合就是指令集。M指令积是32位的，比thumb16位的好解码，灵活。代码编译原理。机器码汇编c语言，机器识别的码，机器码的自然语言别名，那几条指令对应成一条指令。先预处理在编译汇编链接，删掉注释自动化的改代码，转成汇编语言，转成机器码，链接各种库。（比如乘法对应成几个加法）

## **\*\*D2-03 ARM存储模型**

**arm存储模型** arm支持哪些数据类型、怎么存取、支持哪些指令、怎么存取。字节 字 半个字，整存整取分大小段。见下面指令，整存整取分大小段。

## **\*\*D2-04 ARM工作模式**

计算机 arm工作模式

**用户模式就是在执行应用程序时的模式，嗯，不管Linux操作系统是用户态还是内核态，因为这就是CPU的状态，不是操作系统的状态（区分觉得还有些模糊）。（？）**

**觉得中断应该就是嗯正常做点事儿呢，突然就来了一个。必须做的，必须当前做的一个急事儿。比如打印机卡了，同事有急事儿找你停下来。**

中断模式就是CPU正常执行程序时，遇到了当前不得不做的一个急事儿（不做会有重大损失），然后去做这个急事儿，做完了再回来。这个急事儿一般就是硬件方面的节事儿，比如嗯打印机卡了，网卡拆包，等等。因为优先级低的，优先级高的。

**SVC模式，CPU复位（觉得就是开关机重启？）或软中断的时候进入的状态。**

终止模式，存取异常时进入的状态。

未定义模式，执行未定义指令时进入的模式。

系统模式，和用户模式相同寄存器集的特权模式（什么时候进入的？）。

监控模式，安全监控代码的模式。

不同模式拥有不同的权限，执行不同的代码，比如用户模式为非特权模式，一些情况下改内存什么的会并不是那么自由；用户模式之前是各种人写的代码，其他的可能是系统代码。（中断服务终止未定义，这几个都是CPU遇到异常后进入的）

（讲的A9系列芯片）

（这部分关于各种模式及其特点的大概了解，大概知道，模糊的知道就行，之后应该还会详细的学习。见到知道，想到知道就行。了解熟悉的程度。）

## **\*\*day03 ARM寄存器组织**

### **\*\*D3-01 ARM寄存器组织（一）**

**觉得寄存器就是配合控制器、运算器存储的机器，比如配合控制器存指令，配合运算器存结果等。分为通用寄存器和专用寄存器。专用寄存器又分为pc lr sp cpsr spsr。cpsr 后五位表示模式，第六位表示状态，第七八位分别表示快速中断、中断的禁止与否，第一位表示负数位，第二位表示零位，第三位表示进位借位拓展（无符号），第四位表示溢出（有符号）。**

## D3-02 ARM寄存器组织（二）

## D3-03 ARM寄存器组织（三）

# \*\*day04 ARM异常处理

## \*\*D4-01 ARM异常处理（一）

觉得异常处理就是，正常执行程序，遇到必须停下来做的不正常事儿（不做有重大损失），处理完之后再回来。不正常事儿有快速中断、中断等七种，分别对应快速中断、中断五种模式。停下来就是保存当前的程序和状态，涉及cpsr, spsr, PC, Lr, Spsr lr都是下一个模式的。处理就是修改成异常的程序和状态，Cpsr修改要涉及状态、模式、中断禁止；PC修改要涉及到向量表（向量表上指定要跳转的地方，应该有程序根据异常能自动写）。回来，将程序和状态恢复回来，涉及spsr, cpsr, lr, pc, Spsr lr都是下一个模式的。（助记见课件）

## D4-02 ARM异常处理（二）

## D4-03 ARM异常处理（三）

## \*\*D4-04 ARM微架构

Arm微架构就是指令流水线 控制器运算器、寄存器的配合，取指，译指，执指，分别对应下下条、下条、本条。（多核处理器 一个soc上放了多个CPU，不同县城可以在不同的CPU上运行；可以实现真正的并发）

# \*\*day05 ARM指令集仿真环境搭建

## **\*\*D5-01 ARM指令集导学**

(以下改成概念的形式觉得更好)

**是什么。指令集就是命令集合。命令在处理器内有电路对应（也可以说是处理器向外提供的虚拟接口，进行什么样的输入了，就会有怎样的输出）。分为汇编式的，机器码式的；又分arm、thumb式的。**

**为什么。学汇编式的，有利于读写汇编（当然写的会很少），理解CPU怎么执行程序，写出高效代码（因为c语言就是翻译成这些指令集。然后才去执行的。知道工人怎么干活，你才也可能更好的指挥他干的更好，更高效（比如用哪个、先后顺序等）；比如c中寄存器的使用，栈的分配与使用，程序调用函数、参数传递的。）**

**怎么学。学汇编指令，而不是机器码指令（一一对应，但更简单）**

## **\*D5-02 Keil仿真环境搭建**

(以下改成概念的形式觉得更好)

keil集成开发环境是什么？有什么用（为什么）？怎么安装使用？gCc交叉编译链是什么？有什么用（为什么）？怎么安装使用？keil集成开发环境就是集成了编辑，编译，调试，仿真器（？）等的一个工具。编辑编译调试程序。详见课程课件（创建工程文件；编写保存；编译；仿真界面及其上面的关键要素 单步 复位 退出）。Cc交叉编译器就是针对arm体系结构的gc编译器。对c代码等进行编译。详见课程课件(应该上面编译就是)。

## **\*D5-03 ARM指令集概述（ARM汇编概述）**

汇编就是和机器码一一对应的文字指令，分为指令、伪指令、伪操作。指令就是确实一一对应，在处理器中有对应电路的指令，分为数据处理、加载保存、暂停、跳转、协处理器、状态六种。伪指令就是假指令，要转变下才有对应电路的指令。伪操作就是假操作，不执行，只用于让编译器识别改代码改完就删（比如.if.else）。

## **\*\*day06 数据处理指令**

### **\*\*D6-01 数据处理指令（一）**

看法 直接找要操作的数操作就行；主动记，就记加减乘除与或非异或几种，其他多看几遍见到知道或来查就行

赋值（搬移），mov r1, r2, #2

加，add r1, r2, r3, #3

减，sub r1, r2, r3, #3

乘，mul r1, r2, r3, #3（去除#3）（可能妨碍编码解码了吧）

除，无（精简指令集，复杂点的都不要；低功耗、高速、容易设计等）

与，and r1, r2, r3, #3

或，orr r1, r2, r3, #3

非，无（同上）

异或，eor r1, r2, r3, #3

左移，lsl r1, r2, r3, #3

右移，lsr r1, r2, r3, #3

清零，bic r1, r2, r3, #3（哪位是1就清谁）

格式，操作 结果 算数寄 算数寄（+算数）

格式拓展 有2个操作，也按上面去凑

取反赋值（取反搬移），mvn r1 r2, #2

逆向减，rsb r1, r2, r3, #3（3-2）

64位加，分高低位分别存放、分别加，第一个加S，第二个加C（去个D）

64位减，分高低位分别存放、分别减，第一个加S，第二个加C（去个U）

立即数，能放到32位指令里面的数就是立即数，0~255都是，之后大点儿了，有的是，有的不是。觉得有的可能是对应到寄存器了（当然可用ldr）。

## D6-02 数据处理指令（二）

## D6-03 数据处理指令（三）

## D6-04 数据处理指令（四）

## \*\*day07 跳转与存储器访问指令

### \*\*D7-01 跳转指令

跳转指令就是修改PC值，让CPU别按顺序执行了。一是自己直接去改（还要去算，比较麻烦），2是想跳到哪儿就写个标号，然后指示跳到标号

### \*D7-02 ARM指令的条件码

条件码就是cmp之后确定什么条件执行什么操作，cmp本质就是相减（subs），根据nzcv判断两数关系，条件一般有带符号大于、小于、等于、大于等于、小于等于、不等于等（具体可再去看 [📖随记-通用学习笔记3 D7](#)），基本上大部分操作都行（等着见到再学）。

### \*\*D7-03 内存访问指令（一）

内存访问指令就是去读写内存,读就是ldr r1 [r2]（操作地址（加【】）），写就是str r1 [r2]（操作寄存器（加【】））（单字节加b，半个字加h）

### \*D7-04 ARM指令的寻址方式

寻址方式就是寻找地址上数据的方式。根据地址可以分为立即寻址，寄存器寻址。寄存器寻址又分为立即（mov r1 r2），移位（mov r1 r2 lsl #1），间接（ldr r1 [r2]），基址加变指（ldr r1 [r2 r3]）。基加变又分前索引（加！之后r2要改），后索引（r3挪出之后r2要改），自动索引（同前索引）。（详见 [📖随记-通用学习笔记3 D7](#)；大概记，见到知道或者来查就行）

## \*\*day08 栈的种类与应用

（觉得可以叫多寄存器寻址）

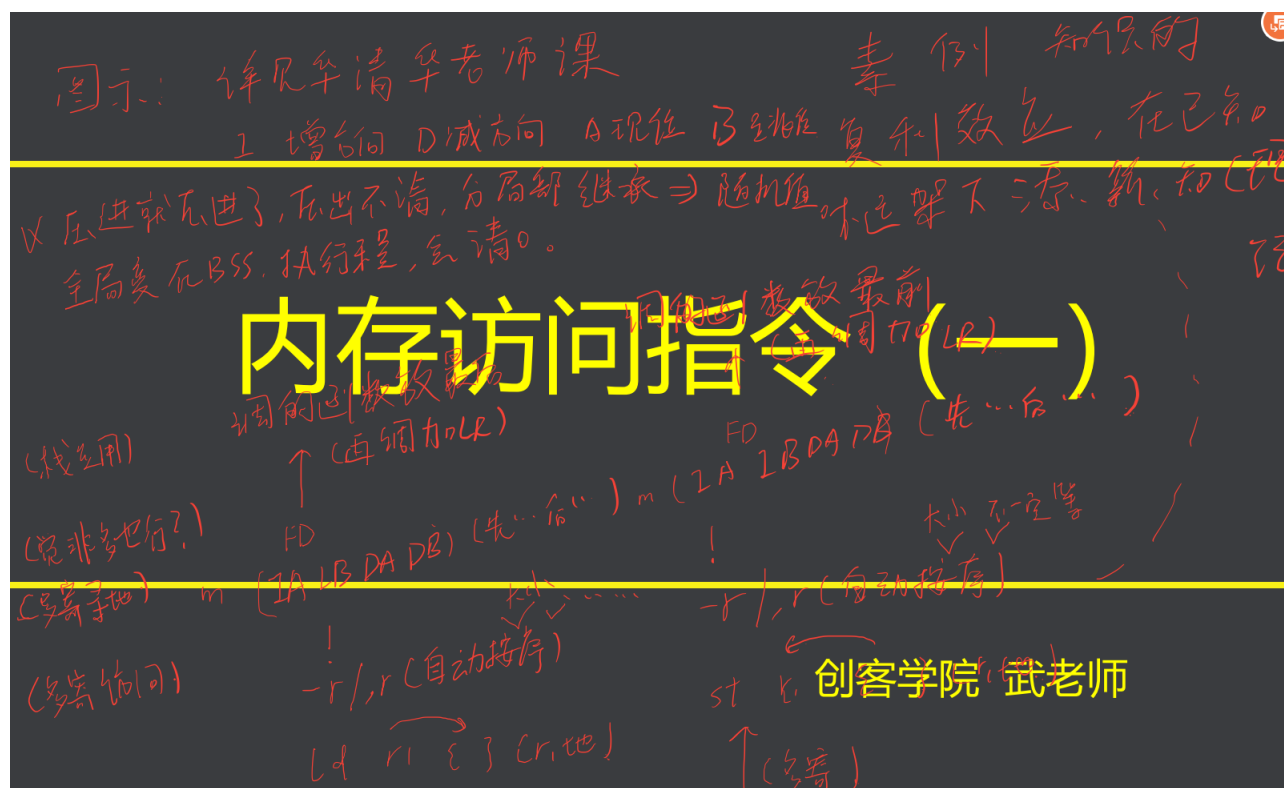
### \*\*D8-01 多寄存器内存访问指令

多寄存器内存访问就是，ldm, stm；内存r0放前面；寄存器r123放{}，放后面（操作地址（加【】））（操作寄存器（加【】））。

(觉得这里可以不用显示r0, r1 上同)

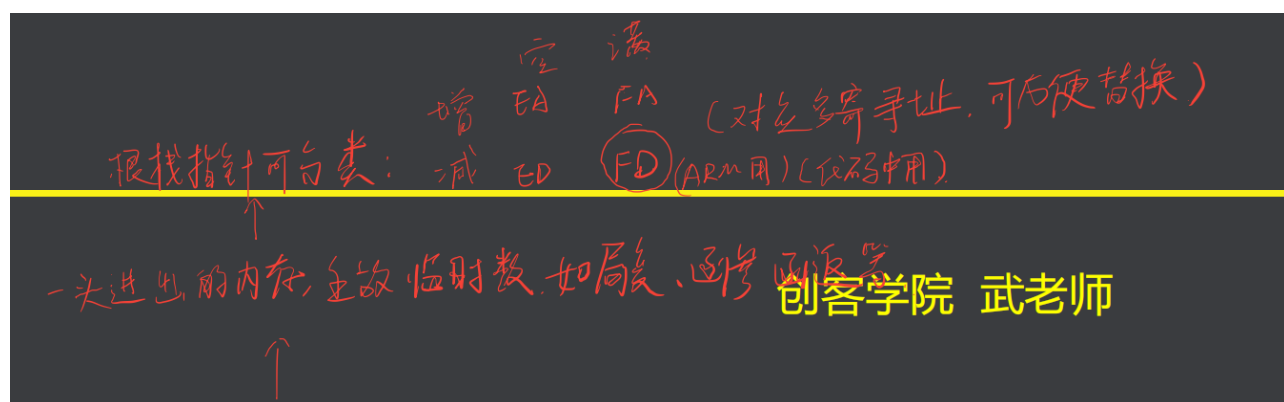
## \*\*D8-02 多寄存器内存访问指令的寻址方式

多寄存器寻址方式，有现位能拿到的，也有跳位才能拿到的；有增方向去拿的，也有减方向去拿的；自动化的话，在地址后加！



## \*\*D8-03 栈的种类与使用

栈分类，可以根据上面多寄存器寻址方式分，arm是按照**现位减方向**拿设计的，所以一般用FD





(根据内存上指针)

#### **\*D8-04 栈的应用举例**

如果一个函数内调了其他函数

那这个函数内部，要先压栈保护可能被破坏的寄存器值，如 r1、r2 和 lr。

执行完毕后，通过出栈恢复现场，包括恢复寄存器值和栈指针位置。

#### **\*\*day09 专用指令**

##### **\*\*D9-01 状态寄存器传送指令**

状态寄存器传送指令就是MRS（Move to Register from Status Register）和MSR（Move to Status Register from Register），操作对象r1、cpsr也是对应的；一般用在系统刚上电的时候，系统调用的时候。

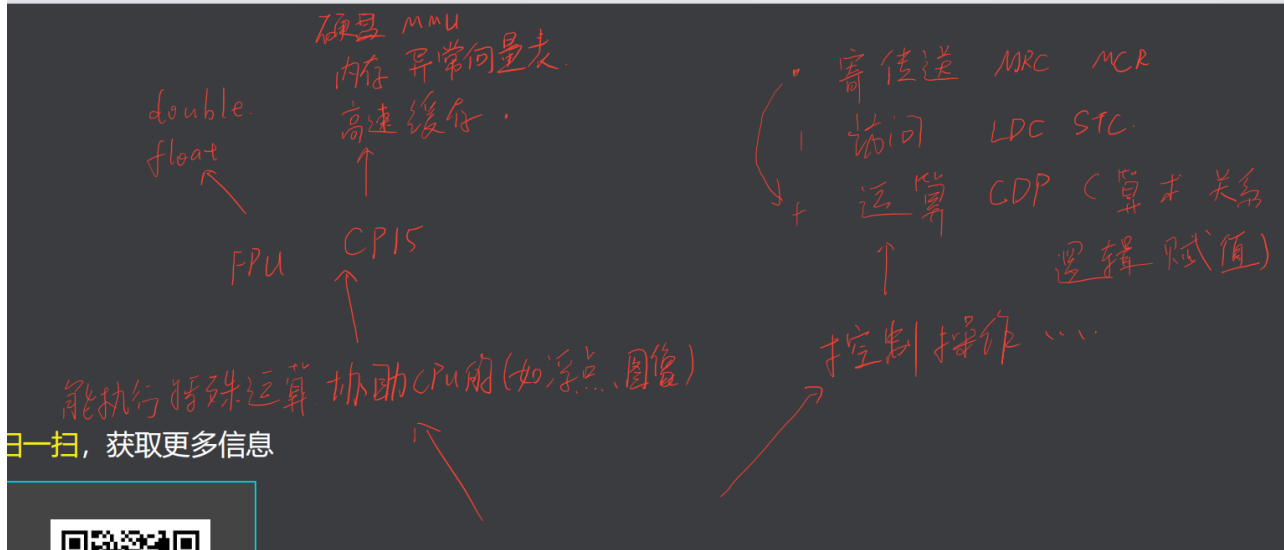
##### **\*\*D9-02 软中断指令**

在异常处理机制上面，跳过去处理部分，还要加上压栈保护现场（r123 lr spsr等）出栈还原（根据栈应用）

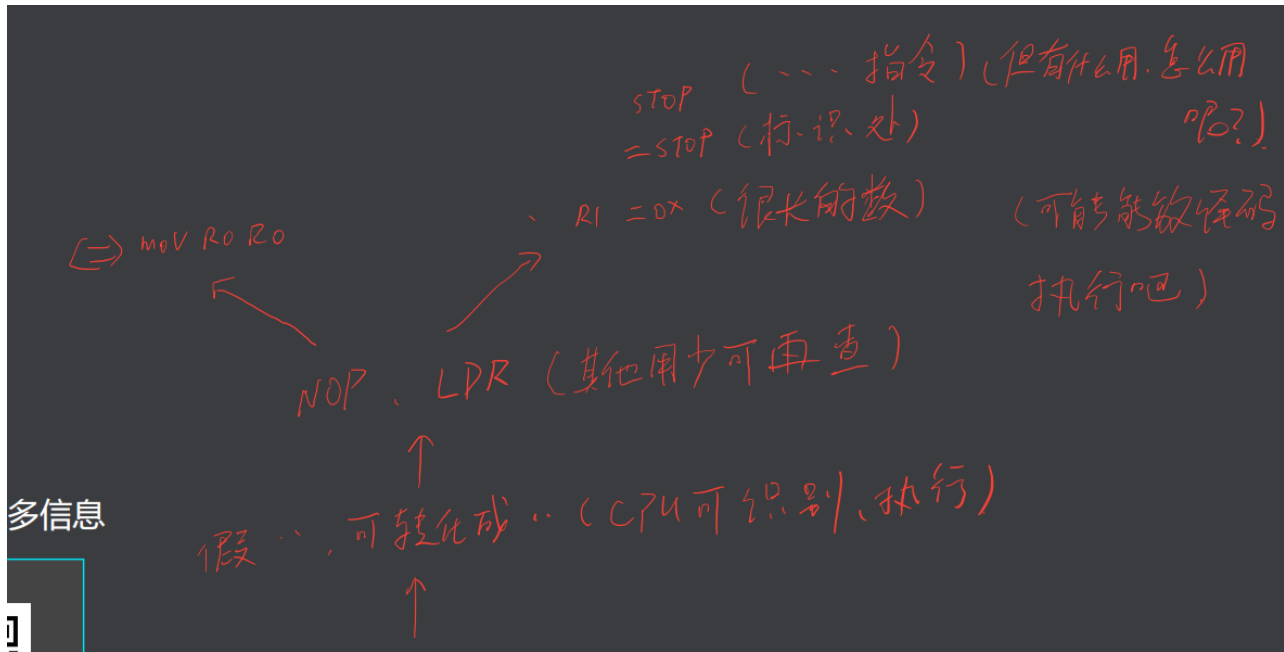
##### **\*\*D9-03 协处理器指令**

(觉可能就映射、读写)

(格式涉协商, 还需时研究; 也访真不)



## \*\*D9-04 伪指令



## \*\*day10 伪操作与混合编程

## \*\*D10-01 伪操作

伪操作就是假操作，不执行，只用于让编译器识别改代码 改完就删；类似于c预处理

创客学院 武老师

`.weak (symbol)`  
`.space`  
`.rept |.endr` `.align` `(2?)`


`(.endif`  
`.if(0)`  
`.equ(DATA, 0xFF)`  
`#if` `define` `include` 其他

`(.local (但默认就是吧?)`  
`.global` `(记得无变量, 不指对吧?)`  
`(.macro .endm (Func Func)`  
`.text .end`

word, byte  
↑  
填充, 有意义的, 无意义的  
(再其他见到可查, 也简单)

扫一扫, 获取更多信息

假操作 生成新指令的操作: 类似C预处理



## \*D10-02 C和汇编的混合编程

`(之前.global)` `(无须.global)`

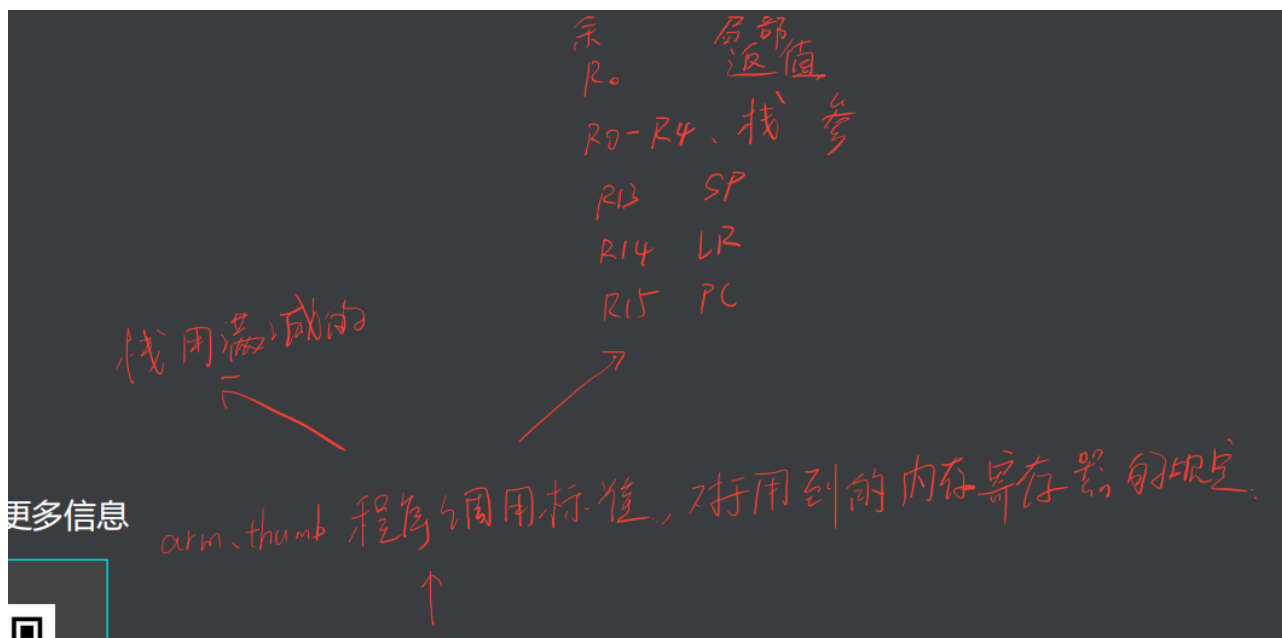
`标头(); LC规` `BL 函数(记规)` `asm();`  
`... \n`

↑

C调汇, 汇调C, C放汇

取更多信息

## \*D10-03 ATPCS协议



## \*D10-04 ARM体系结构总结

无，以后处理问题。

计算机 D10-04 ARM体系结构总结.mp4

(嗯，操作系统就是封装对底层硬件的操作变成接口给上层应用有)

(地址总线和寄存器的位数应该是有一定对应关系的吧?)

(哦，嵌入式一般用的都是soc? 那就是河吗?)

(CPU整存整取数据的话，怎么取数据呢?)

(哦，知道了c语言中定义结构体一个char类型，一个int类型。中间可能要有dummy。)

(设计模式是为了上层的操作系统和应用的需要，比如不能让应用程序，然后是乱搞的。操作系统，应用程序就要用用户权。)

(异常不一定是错误，我可能就是处理一些紧急的更重要的事儿。时间管理类似。)

(对，觉得异常就是非当前程序，嗯，但是需要当前嗯，要去处理的程序。可能是紧急可能重要。)

(真的就是异常这种机制。)

(觉得内存访问和寻址是关联起来的，一般就是先去的地址在访问再可能改编地址，不管是多计算器还是单个寄存器都那都是这样的。)

(低级语言用起来困难，但能说明原理：高级语言用起来容易，但说明原理不容易。)

(shell原来比c更高级，为什么呢? 各种语言分类，优缺点?)

(各种语言都是对底层语言的封装，然后向用户提供结果。)

(底层知识涉及的面儿是很多的，比如说内核化涉及到而且进程内存文件系统驱动网络这些嗯，咱们现在是为了驱动而学其他的还是其实还有很多没学呢? 对于底层这么多的话，应对的话就是嗯你需要什么就去学什么都行。)

## **\*day11 FS4412开发环境搭建**

(接口部分，学法参考，之前笔记，ai课程总结，课件，课程；人总结，机器总结，原件 课件 课程 知识提问)

### **\*D11-01 开发板硬件资源介绍**

开发板硬件资源介绍，觉得和计算机组成差不多，cpu，内存，硬盘，网络，设备；cpu包含了gpio等控制器；内存有四个拓展的；硬盘用的emmc；网络包括网卡、串口、总线；设备包括传感器、音频、视频、灯光等。其他就大概了解知道就行。（cpu和网络、设备之间应该还有控制器做个中介，比如和led中间有个gpio控制器。为什么cpu还设个分管的呢？模块化，避免改一个全改吧；还有什么好处呢？之后复习再说吧。）

### **\*D11-02 初识电路原理图**

#### **见下12-1**

电路原理图会**查找元器件**，能看懂元器件实现功能涉及哪些接口，然后顺着找到控制器接口就行。前者一般**看目录或者搜索丝印**，后者一般借助网络标号。（更小的元器件 见到知道就行，电阻（限流分压），电容（电源稳定性？），二极管（单向导电性），三极管（开关），芯片（引脚的名字就代表功能，VDD，VCC？））

### **\*D11-03 交叉开发环境搭建**

交叉开发环境就是交叉编译工具链，USB转串口驱动，SecureCRT；因为x86架构和arm架构指令不同，电脑和开发版通信，开发版显示交互。具体安装使用见说明或者网上搜索、ai问答。

### **\*D11-04 地址映射表**

觉得地址映射表就是，Flash、rom、ram、专用寄存器会映射成整个内存。cpu控制硬件，通过st、ld专用寄存器映射的。(好像视频上还有内容，有时间再看)

## **\*\*day12 GPIO实验**

(觉得介绍控制器，一般就是介绍**基本原理**，然后按照**流程去实现**)

### **\*\*D12-01 GPIO简介**

觉得gpio就是输入输出高低电平，实现控制、信号采集、（通讯）什么的；具有通用性，谁想输入输出高低电平实现功能，都能拿来用。

### **\*\*D12-02 GPIO寄存器分析（一）**

寄存器分析一般就是根据设备功能；找相关线路（及控制）逻辑、引脚、寄存器逻辑（了解寄存器功能、用法）；编写程序实现（读写寄存器）。

设备功能、线路逻辑、引脚到时候联想；寄存器逻辑，可以根据某引脚（gpx2），浏览找有用的寄存器（con、data），及其用法（31-28设置引脚功能，0-7设置引脚数值）。

（哦，原来一个寄存器是可以控制多个引脚的，比如这里gpx2控制了有8个引脚）

### **D12-03 GPIO寄存器分析（二）**

### **\*\*D12-04 GPIO编程**

编写程序实现就是，编写、编译、执行。编写就是根据设备功能、寄存器逻辑联想、设位。编译包括汇编、连接、转化，到时候参考模版或者ai（模版 通用笔记3 [📖随记-通用学习笔记3](#) day12（借助ai也能看懂））。执行就是go transfer enter loadb。

## **\*\*D12-05 LED实验**

LED实验就是不仅实现开灯，还要实现开一会儿灭一会儿：让引脚有，设上位，暂停会儿，设上位，暂停会儿，重复前面四步。

(汇编代码编写方法 觉得只要涉及到跳转，都要考虑pc pc pc 保存恢复吧，保pc L，改pc B，改pc MOV，保存恢复 考虑R LR (一般都弄 之后pc^)(觉得如果是异常处理，cpsr应该是自动改了，之前看swi示例看着是没涉及)

## **\*day13 C工程与寄存器封装**

### **\*D13-01 C语言工程简介**

(觉得c语言工程，就是一种工程模版)

详见 [📖随记-通用学习笔记3](#) day13部分

计算机 D13-01 C语言工程简介.mp4

详见 [随记-通用学习笔记2](#)

### **\*D13-02 启动代码分析**

详见 [📖随记-通用学习笔记3](#) day13部分

## **D13-03 C语言实现LED实验**

### **\*D13-04 寄存器的封装方式**

(觉得寄存器封装，就是对数据和操作的封装，本质也是一种抽象)

# 寄存器的封装方式

对地址做性质，用，#define 封装

↑ 创客学院 武老师

(寄存器地址和上面的标签，都是生的没有类型的，所以) 改地址性质，获得标签，然后封装

## volatile关键字

: 提到了volatile关键字，但未详细解释，通常用于告诉编译器一个变量可能会在程序的控制之外改变。



```

1  #define      CLK_GATE_IP_ISP0      __REG(0x10048800)
2  #define      CLK_GATE_IP_ISP1      __REG(0x10048804)
3  #define      CLKOUT_CMU_ISP        __REG(0x10048A00)
4  #define      CLKOUT_CMU_ISP_DIV    __REG(0x10048A04)
5  #define      CMU_ISP_SPARE0        __REG(0x10048B00)
6  #define      CMU_ISP_SPARE1        __REG(0x10048B04)
7  #define      CMU_ISP_SPARE2        __REG(0x10048B08)
8  #define      CMU_ISP_SPARE3        __REG(0x10048B0C)
9
10 /***** PWM *****/
11
12 typedef struct {
13     unsigned int    TCFG0;
14     unsigned int    TCFG1;
15     unsigned int    TCON;
16     unsigned int    TCNTB0;
17     unsigned int    TCMPB0;
18     unsigned int    TCNT00;
19     unsigned int    TCNTB1;
20     unsigned int    TCMPB1;
21     unsigned int    TCNT01;
22     unsigned int    TCNTB2;
23     unsigned int    TCMPB2;
24     unsigned int    TCNT02;
25     unsigned int    TCNTB3;
26     unsigned int    TCMPB3;
27     unsigned int    TCNT03;
28     unsigned int    TCNTB4;
29     unsigned int    TCNT04;
30     unsigned int    TINT_CSTAT;
31 }pwm;
32 #define      PWM (* (volatile pwm *)0x139D0000)

```

## \*D13-05 寄存器操作的标准化

# 寄存器操作的标准化

$(1 \ll ?)$   $(\sim(1 \ll ?))$

1 1 & 0

可写  
1 1 (改几位)  
& 0 (改几位)

改单个性 改几位 创客学院 武老师

(清零、置一，具有单个性，改这几位是1，另几位就改不了了，如果不是想要的，那就不是想要的了；所以干脆先全清零，该清零的清零了，再说置一的)

```
1 #include "exynos_4412.h"
2
3 int main()
4 {
5     GPX2.CON = GPX2.CON & (~0xF << 28) | (0x1 << 28);
6
7     while(1)
8     {
9         /*点亮LED2*/
10        GPX2.DAT = GPX2.DAT | (1 << 7);
11        /*延时*/
12        Delay(1000000);
13        /*熄灭LED2*/
14        GPX2.DAT = GPX2.DAT & (~(1 << 7));
15        /*延时*/
16        Delay(1000000);
17    }
18    return 0;
19 }
```

## **\*\*day14 UART实验**

### **\*\*D14-01 UART帧格式详解**

uart是个串口（全双工异步）收发器，（嵌入式系统中常用于主机、辅助设备通信）。帧格式分为，开始，结束，数据，校验，空闲（意义很显然，怎么设是关键；下面是电平表现）。开始，一位（区别于空闲的）低电位。结束，1个或1.5个或2个（和空闲相同的）高电平。数据，1就高，0就低，5-8位。校验，偶校验设1，奇校验设零，无校验就无位。空闲，高电平，发送完一个数据必须空闲一段时间，避免误差累计（十中可能一个走的快，一个走的慢；设备是嗯通过掐时间，来确定连续的0或者一）。

### **\*\*D14-02 Exynos4412下的UART控制器**

串口控制器工作原理：

Figure 28-1 illustrates the block diagram of UART.

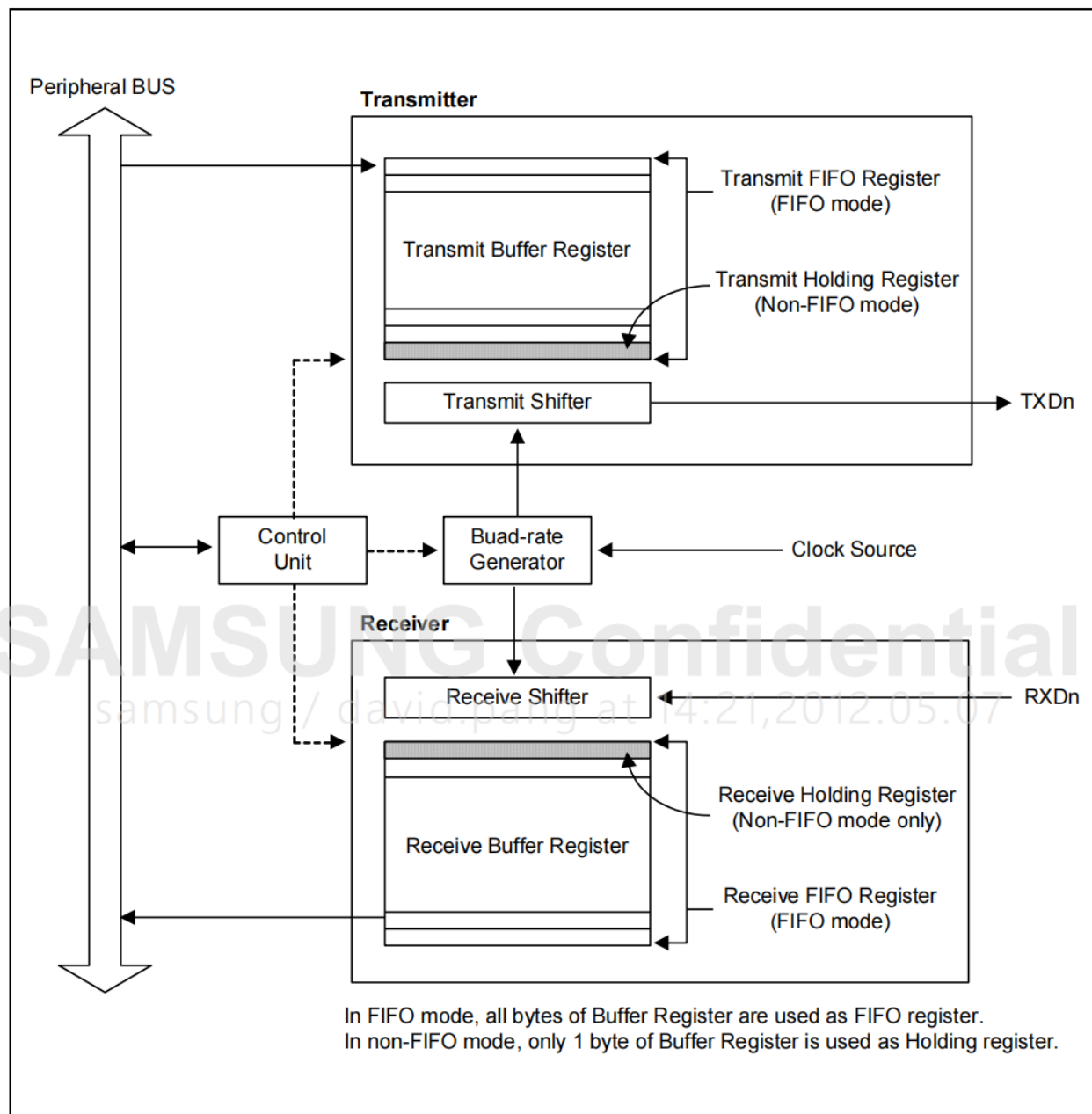


Figure 28-1 Block Diagram of UART

(觉得主要根据这图背就行，清楚，也容易背)

## \*\*D14-03 UART寄存器详解

详解主要讲了让引脚有，控制，波特率，发送接收数据及其模式、状态这几个寄存器，这里控制主要控制的是帧；重点是自己找到这些有用的寄存器及有用的位以及怎么设置。

(哦，原来是波特率分频器及其小数部分)

UART Baud-Rate Configuration

You can use the value stored in the Baud-rate divisor (UBRDIVn) and divisor fractional value (UFRACVALn) to determine the serial Tx/Rx clock rate (Baud rate) as:

$$DIV\_VAL = UBRDIVn + UFRACVALn/16$$

or

$$DIV\_VAL = (SCLK\_UART/(bps \times 16)) - 1$$

Where, the divisor should be from 1 to (216 - 1).

By using UFRACVALn, you can generate the Baud rate more accurately.

For example, if the Baud rate is 115200 bps and SCLK\_UART is 40 MHz, UBRDIVn and UFRACVALn are:

$$\begin{aligned} DIV\_VAL &= (40000000/(115200 \times 16)) - 1 \\ &= 21.7 - 1 \\ &= 20.7 \end{aligned}$$

$$UBRDIVn = 20 \text{ (integer part of } DIV\_VAL \text{)}$$

$$UFRACVALn/16 = 0.7$$

$$\text{Therefore, } UFRACVALn = 11$$

(为什么公式是这样设计的呢？除以16，减1)

(哦，觉得应该是好几个控制器，对应一个引脚，在一个控制器寄存器设置谁连接引脚就行了吧

GPA1CON[1]	[7:4]	RW	0x0 = Input 0x1 = Output 0x2 = UART_2_TXD 0x3 = Reserved 0x4 = UART_AUDIO_TXD 0x5 to 0xE = Reserved 0xF = EXT_INT2[1]	0x00
GPA1CON[0]	[3:0]	RW	0x0 = Input 0x1 = Output 0x2 = UART_2_RXD 0x3 = Reserved 0x4 = UART_AUDIO_RXD 0x5 to 0xE = Reserved 0xF = EXT_INT2[0]	0x00

\*\*D14-04 UART编程

初始化，然后不断地 接收数据，处理数据（加一）， 发送数据，发送字符串（hello world）。

初始化，一般就除了接收发送寄存器。

接收数据，先检查状态再接受，（再返回）。

发送数据，先检查状态再发送。

发送字符串，非\0就发送。

(详见 [📖随记-通用学习笔记3](#) day14部分，相应程序部分)

## \*D14-05 输入输出重定向

开发板上，标准输入输出 是串口，有了操作系统（应用层） 则是键盘和屏幕（显卡）；移植系统，要注意相关设置。

(详见 [📖随记-通用学习笔记3 day14部分](#))

## \*\*day15 WDT实验

### \*\*D15-01 WDT简介

看门狗就是个计数器，接收CPU每隔一段时间给的数，由此（慢慢）减一；接收不到，减到0，让CPU复位（含soc）。（主要为防止系统死机或软件故障）（助记见课件）

#### 计算机 看门狗简介

看门狗就是一个计数器，接收CPU每隔一段时间给的数，然后由此每隔一段时间减一，如果CPU死机了，那他就会降到0，然后让CPU复位（含soc）（助记见课件）。

（嗯，手机或电脑死机是什么情况呢？一般怎么着？为什么有效？在CPU层面是什么情况？在软件层面是什么情况呢？觉得可能类似人头脑蒙了，处理的太多，或者太难不会，就变卡顿了 甚至停止了）

（系统死机或软件故障，更本质上是是什么意思？就是发生错误，cpu不知道怎么运转了吗）

### \*\*D15-02 Exynos4412下的WDT控制器

看门狗控制器工作原理：

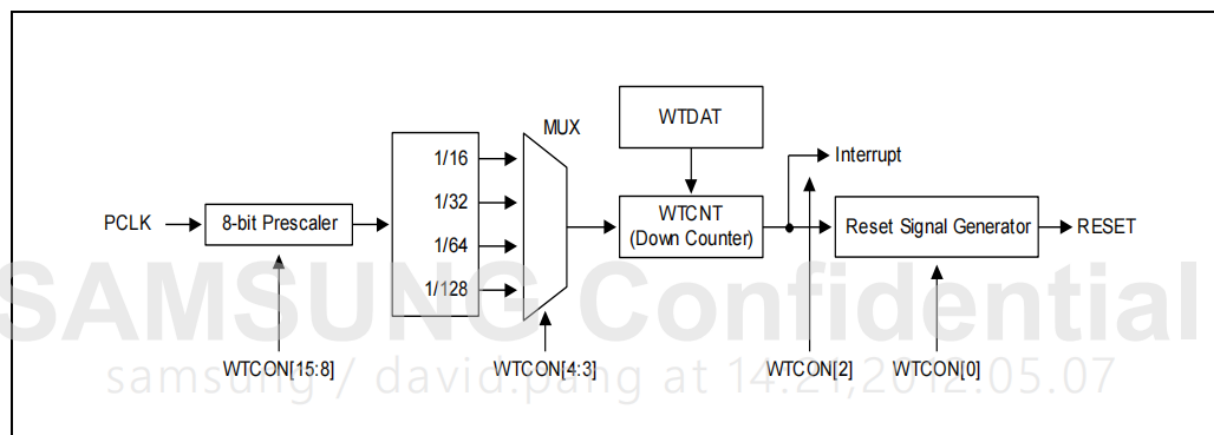


Figure 26-1 Watchdog Timer Block Diagram

(预分频器, 分频器, 计数器, 中断, 重置发生器)

[Figure 26-1](#) illustrates the functional block diagram of WDT.

The Watchdog Timer Control (WTCN) specifies the prescaler value and frequency division factor. Valid prescaler values range from 0 to  $(2^8-1)$ . You can select the frequency division factor as: 16, 32, 64, or 128.

Use this equation to calculate the WDT clock frequency and the duration of each timer clock cycle:

$$t\_watchdog = 1/(PCLK/(Prescaler\ value + 1)/Division\_factor)$$

(哦, 知道为什么加一了, 让0也有去处, 哈哈)

(1, pclk, 预分加一, 分)

## **\*\*D15-03 WDT寄存器详解**

详解主要讲了看门狗控制、计数这两个寄存器; 重点是自己找到这些有用的寄存器、有用的位以及怎么设置。

(wdt不需要外部设备, 所以没有对应的引脚吧)

## **\*\*D15-04 WDT编程**

初始化, 不断打印cnt值, 延时, 喂狗 (让消耗下, 再喂)

初始化, 就是上面寄存器位设上位

延时, while (time--)

## **\*\*day16 轮询与中断**

### **\*\*D16-01 CPU与硬件的交互方式**

CPU与硬件的交互方式主要有轮训、中断、dma三种, 轮训就是每隔一段时间就去问下硬件 效率较低, 中断就是等着硬件发中断信号 效率较高, dma就是硬件直接把数据放内存稍后cpu再取 效率最高但用的少 (额外的软硬件支持、资源占用, 代价可能高)。

计算机 D16-01 CPU与硬件的交互方式.mp4

详见 随记-通用学习笔记2

(为什么DMA：硬件直接操作内存，无需CPU干预，效率最高但使用较少。)

## **\*\*D16-02 轮训方式实现按键实验**

详解主要讲了GPX1.CON、GPX1.DAT这两个寄存器；重点是自己找到这些有用的寄存器、有用的位以及怎么设置。

编程就是初始化，不断判断 按下的话 打印按下了同时等待松手。初始化，就是除了dat寄存器的初始化。等待松手是为了按了一次就打印一次。

(详见 [📖随记-通用学习笔记3](#) day16部分，相应代码部分)

## **\*\*D16-03 GPIO中断相关寄存器详解**

详解主要讲了让有、让能、方式 (GPX1.CON、EXT\_INT41\_CON、~~EXT\_INT41\_MASK~~)；总开关、单开关、cpu、接口；几个控制器；中断号、清楚挂起 (EXT\_INT41\_PEND)、告知做完；重点是自己找到这些有用的寄存器、有用的位以及怎么设置 (中断控制器的可适当参考)。

(设置GPIO引脚为中断功能，当按键按下时，GPIO给CPU发送中断信号；这就是不经过中端处理器呗；那我想经过怎么办呢？哦，本来就是经过的呀；看代码。)

(觉得代码里清除中断挂起标志位，是不是没有正确清除？)

中断产生时，EXTERN\_INTERRUPT\_41\_PEND寄存器会自动挂起中断，待CPU处理后需要手动清除挂起状态。(???)

(觉得外中断mask有点保险的意思)

## **\*\*D16-04 GPIO中断编程**

编程就是初始化，led不断亮灭，中断处理函数

初始化就是上面寄存器设位



led，之前led函数

中断处理函数，获取中断号，switch自己的号，打印按下，清除中断挂起标志，写回中断控制器  
(怎么把这里的中断处理函数放到中断向量表呢？估计后面可能有吧)

(中断设计这么多寄存器设置，说难果然还是难些的；不过还是觉得，多学多接触，自然熟悉简单。)

(果然，中断处理程序，在启动代码异常向量表跳转里有；之前学汇编学异常处理、函数调用，看这代码还是简单的)

## **\*\*day17 中断控制器**

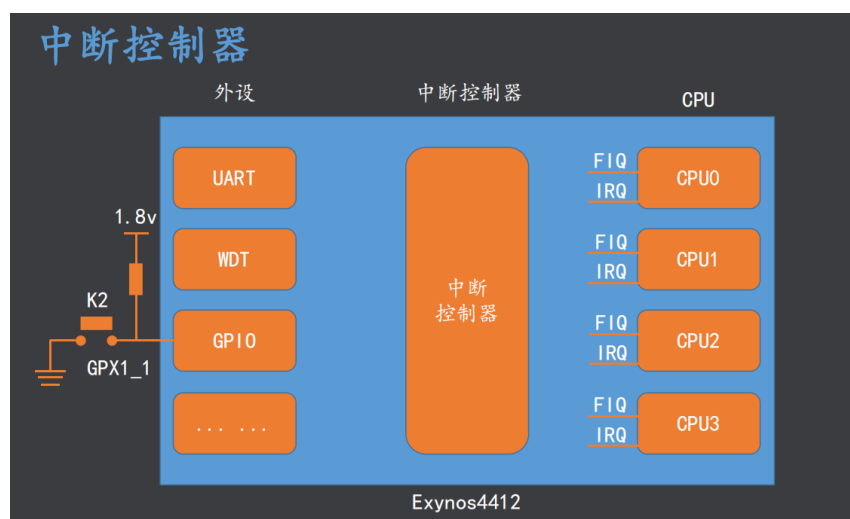
### **\*\*D17-01 中断控制器**

觉得中断控制器就是对外设中断编号、排序、分配 cpu等管理中断信号的，详见 [📖随记-通用学习笔记3 day17部分](#)。

计算机 D17-01 中断控制器.mp41718785261686.docx

详见 随记-通用学习笔记2（助记在课件）

(可以设置再中断)



### **\*\*D17-02 Exynos4412下的中断控制器**

无，详见 [📖随记-通用学习笔记3](#) day17部分。

## **\*\*D17-03 中断控制器寄存器详解（一）**

见day16 详解

## **D17-04 中断控制器寄存器详解（二）**

计算机 D17-04 中断控制器寄存器详解（二）.mp4

（觉得寄存器就是有地址的盒子）

（如果有操作系统，中断控制器，就一点都不用配置了吗）

## **\*\*D17-05 中断控制器编程**

见day16 编程

## **\*day18 中断处理**

### **D18-01 ARM的异常处理机制**

详见之前笔记， [📖随记-通用学习笔记3](#)

计算机 异常机制

（异常机制应该是一种功能性质的机制，不一定是一种错误。）

（中断比轮巡效率更高高是高，但是要付出代价，你在玩游戏的多了明显的）

### **D18-02 工程模板代码结构分析**

详见之前笔记， [📖随记-通用学习笔记3](#)

## **\*D18-03 中断处理框架搭建（一）**

详见 [📖 随记-通用学习笔记3](#)

## **\*D18-04 中断处理框架搭建（二）**

详见 [📖 随记-通用学习笔记3](#)

## **D18-05 中断处理程序编程**

详见day16编程

## **\*D18-06 中断编程补充**

详见 [📖 随记-通用学习笔记3](#)

## **\*\*day19 ADC实验**

### **\*\*D19-01 ADC简介**

adc就是模数转换，就是模拟信号转换成数字信号，等分电压电流值（模拟物理信号来的），转化来的电压电流除以这个等分就行。重要参数有最大电压电流、精度、采样频率（5倍频率下达到）。（详见 [📖 随记-通用学习笔记3 day19](#)）

计算机 D19-01 ADC简介.mp4

adc就是一个将模拟信号转化成数字信号的原件，模拟信号就是将物理量模拟成电压，电流等电的形式（？），转化就是把一段电压或电流范围（取决于最大收入）等分，到时候转化出的数值，乘这个精度就行。（助记详见课件 重点在后者）

（觉得adc是不是一个将浮点数转化成嗯，二进制的控制器？原来是把一段电压范围等分到时候转化出来的数值，乘这个精度就行。）

（精度也可以叫做分辨率。）

（最大输入电压觉得就可能取决于元气价的承载能力吧，和想要达到的精度）

(模拟信号，声光热等转换成电压电流，觉得之前初中高中学过有点印象)

## **\*\*D19-02 Exynos4412下的ADC控制器**

见上，详见 [📖随记-通用学习笔记3](#) day19部分

## **\*\*D19-03 ADC寄存器详解**

详解主要讲了控制（精度设置什么的）、数据、通道这几个寄存器；重点是自己找到这些有用的寄存器、有用的位以及怎么设置。（详见 [📖随记-通用学习笔记3](#) day19 和代码）

## **\*\*D19-04 ADC编程**

初始化，然后不断地 开始转换 如果完成了 就读取结果 转换 打印

初始化就是主要就是设置控制、通道那些值

## **\*\*day20 RTC实验**

### **\*\*D20-01 RTC简介**

RTC就是个实时时钟，需要时可以去读时间，也可以改时间，原理就是有个高精度晶振作为时钟源（让从s开始加，支持闰年判断，支持闹钟），可以外加纽扣电池或电容供电。

计算机 D20-01 RTC简介.mp4

RTC就是嵌入设备中的一个时钟，嗯，需要时可以读取时间。嗯，也可以去改时间，嗯，原理就是有一个精度很高的晶振作为时钟源。可以外加纽扣电池或者电容供电。（助记详见课件）

（Adc为什么要用到时钟源呢？我觉得输输出的话，都需要时钟源。）

（晶振是什么东西有什么用呢？）

（电容供电是什么原理，怎么避免电子马上中和）

## \*\*D20-02 Exynos4412下的RTC控制器

rtc具体工作原理：

### 27.2.1 Block Diagram

[Figure 27-1](#) illustrates block diagram of **RTC**.

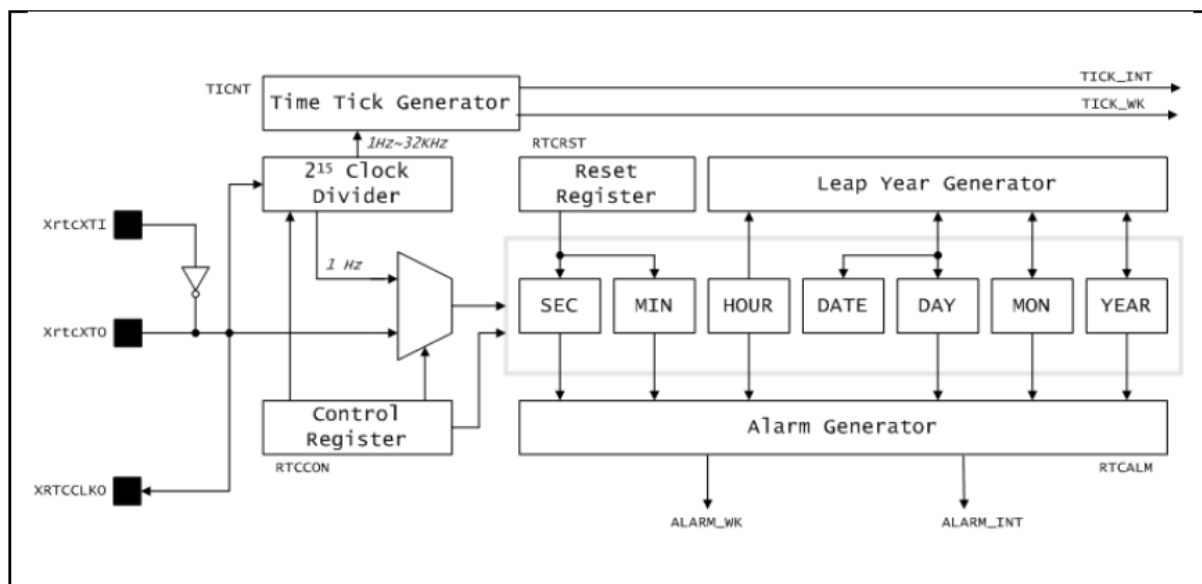


Figure 27-1 **RTC** Block Diagram

(详见 [📖随记-通用学习笔记3 day20](#))

## \*\*D20-03 RTC寄存器详解

详解主要讲了BCD时间存储、控制这几个寄存器；重点是自己找到这些有用的寄存器、有用的位以及怎么设置。（详见 [📖随记-通用学习笔记3 day20](#) 和代码）

## D20-04 RTC编程 (1)

## \*\*D20-05 RTC编程 (2)

初始化，然后不断地 获取新秒数 如果和老秒数不一样 就打印时间 更新老时间

初始化就是主要就是设置BCD、控制那些值，控制先使能再禁止 bcd用16禁止（输出也是）

(输出用10进制可能出错吗)

## **\*\*day21 PWM实验**

### **\*\*D21-01 PWM简介**

**pwm就是宽度可调制的脉冲，整体宽度周期、局部宽度占空比都可以调制；可以驱动无源蜂鸣器，解放cpu。**

计算机 D21-01 PWM简介.mp4

宽度可调制的脉冲，整体宽度就是周期短啊，局部宽度就是占红笔。（再看一下课程视频，补充一下。）

（可以用它来控制无源蜂鸣器，然后他想起来。可以用gl，但是比较消耗GPU资源。）。。

（嗯，有缘蜂鸣器的话需要可以是直流电，然后无烟风鸣器必须是那种。嗯，嗯，脉冲一小时的。）

（还有幅值相位的反转，暂时用不到就不说了。）

（有源什么原理，直流电就可以响）

### **D21-02 Exynos4412下的PWM控制器（一）**

### **\*\*D21-03 Exynos4412下的PWM控制器（二）**

pwm工作原理：

Figure 24-2 illustrates the clock generation scheme for individual PWM channels.

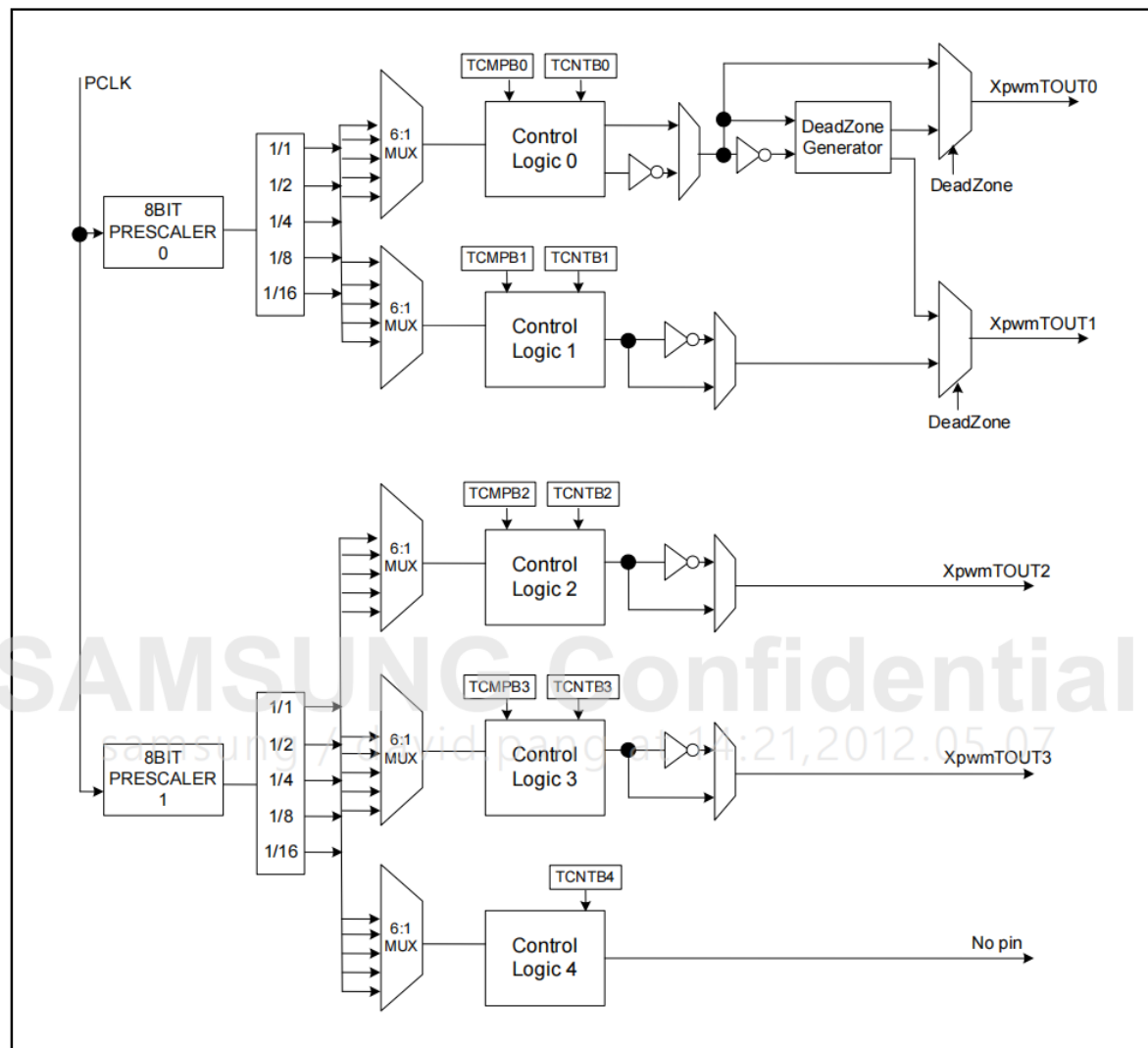


Figure 24-2 PWM TIMER Clock Tree Diagram

Each timer can generate level interrupts.

(右上角有些看不懂，但觉得暂时也不影响，暂时不看 看2/3这种容易理解的) (详见 [📖随记-通用学习笔记3 day21](#))

## \*\*D21-04 PWM寄存器详解

详解主要讲了分频、周期、占空比、控制、让引脚有这几个寄存器；重点是自己找到这些有用的寄存器、有用的位以及怎么设置。 (详见 [📖随记-通用学习笔记3 day21](#) 和代码)

## \*\*D21-05 PWM编程

初始化，然后不断地 开启 延迟 关闭 延迟

初始化就是主要就是设置上面那些寄存器，周期、占空比里面装的应该是分频后的频率数量，手动更新打开后马上关闭 (觉得最后使能没必要呢，while1里面上来就有了)

(手动更新马上就能更新上吗?)

## **\*\*day22 IIC总线原理**

### **\*\*D22-01 IIC总线概述**

iic是个串口同步总线，帧和通信过程如下（类似uart）。

#### 1. 信号的具体实现（一帧包括开始、停止、定位、传输（注意要应答），具体实现见下）

∴

- 起始信号：在I<sup>2</sup>C总线空闲（SDA和SCL均为高电平）时，主机通过使SDA线从高电平变为低电平（SCL仍为高电平）来产生起始信号。
- 停止信号：在SCL为高电平时，由低到高的变化在SDA线上产生，即SDA线由低电平变为高电平，表示通信结束。
- 字节数据传输：每个字节包含8位数据，数据传输时先传送最高位。I<sup>2</sup>C总线使用高电平表示1，低电平表示0。数据传输过程中，SCL线为低电平时，发送器将数据位写入SDA线；SCL线为高电平时，接收器从SDA线读取数据位。
- 应答信号：接收器在接收到一个字节数据后，通过在SCL为低电平期间将SDA线置为低电平来发送应答信号，表示数据已被接收。如果接收器不置SDA为低电平，则表示非应答。

#### 1. 通信过程

∴

- 主机向从机发送数据：主机发送起始信号，随后发送包含从机地址和写方向（读写位为0）的数据字节，从机应答。之后，主机发送数据字节，从机应答，直至通信结束，主机发送停止信号。
- 从机向主机发送数据：主机发送起始信号，随后发送包含从机地址和读方向（读写位为1）的数据字节，从机应答。然后，从机发送数据字节，主机应答，直至通信结束，主机发送停止信号或不进行应答以结束通信。
- 方向变更特殊模式：主机先向从机发送数据，然后在不发送停止信号的情况下直接发送另一个起始信号，随后发送包含从机地址和读方向的数据字节，从机应答并开始向主机发送数据，直至通信结束。

(详见 [📖随记-通用学习笔记3](#) day22部分)



属性，串行，双工半双工同步。过程，发送开始信号（占用线路）。发送从机（地址）和（数据）方向（7位 1位 0主从1从主）接收回应。发送数据 接收回应。发送结束信号（释放线路）。（助记，文件助记）

（主机从机的概念就想到就行，主机可以呼叫接收，从机只能被呼叫。一个线路可以有好几个手机，好几个从机。）。。

（线路连接，sda scl 数据线，时钟线，接到线路上去就行）。。

（不同于uart，开始后发送任意多个字节）

（主要用于一个板子上几个芯片的通信）

（为什么i2c长距离通讯可能会出现错误？）

（i2c为什么是低速的呢？怎么能才能高速呢？）

（主机从机是不是应该可以在控制器寄存器设置？这个应该是的。）

（起始星耀是怎样的呢？）

（觉得老师讲得确实细致，一天的课程，ai两大段就总结出来了 详见 [📖随记-通用学习笔记3](#) day22部分）

## D22-02 IIC总线信号实现

见上

## D22-03 IIC典型时序

见上

## \*\*day23 IIC控制器与MPU6050

### \*\*D23-01 Exynos4412下的IIC控制器

发送过程：觉得差不多就是，3个填充弹药，扣动扳机，发射的过程；中间被一个等待确认、中断、判断隔开；当然填充弹药、扣动扳机可能略有不同。

Figure 29-6 illustrates the operations for Master/Transmitter mode.

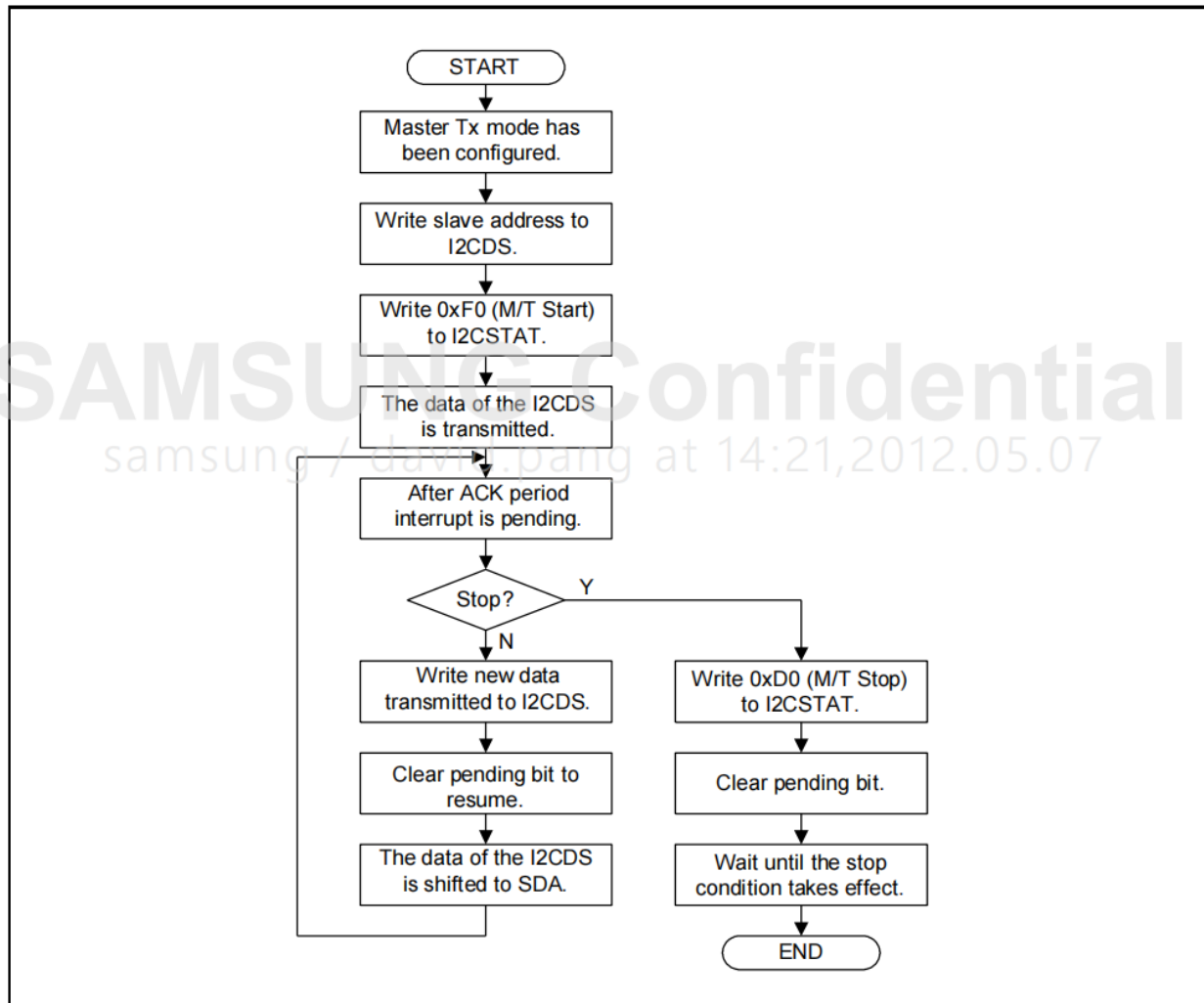


Figure 29-6 Operations for Master/Transmitter Mode

Figure 29-7 illustrates the operations for Master/Receiver Mode.

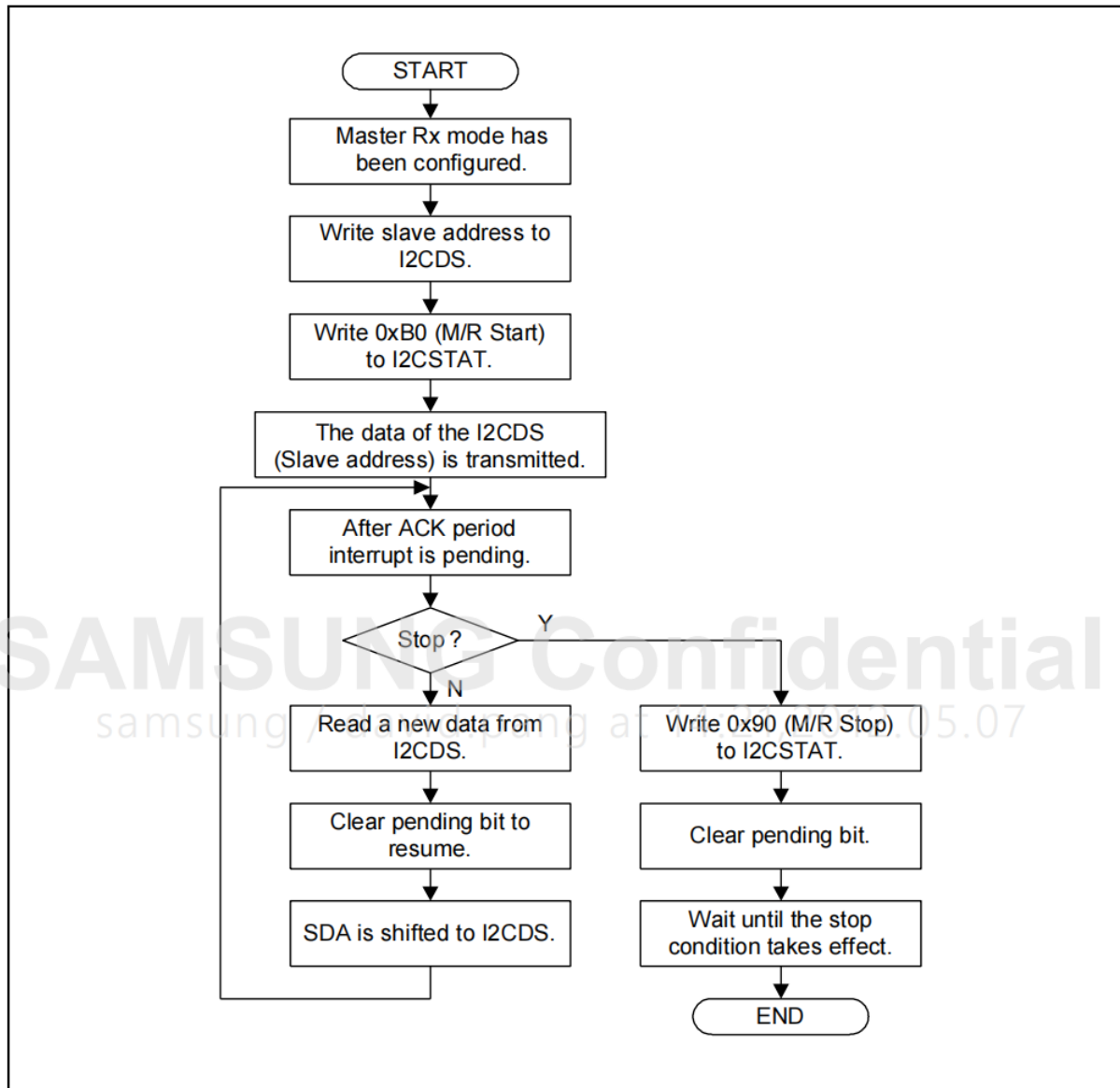


Figure 29-7 Operations for Master/Receiver Mode

(详见 [📖随记-通用学习笔记3 day23](#) 和芯片手册)

(哦，**再改方向**，是不填充数据，清除中断挂起；然后正常填充、扣动就行)

(哦，**主机接收数据**，不应答（因为只接受一次类似填充数据），清除中断挂起，之后等待确认，完了获取数据。觉得芯片手册主机接收模式流程图这部分可能有点问题)

(**注意最后延迟下**，稳定信号，但为什么呢？哦，可能是为了连续读取的情况)

(为什么应答后，中断挂起后，判断停不停？为什么设置中断挂起)

## \*\*D23-02 IIC寄存器详解

详解主要讲了控制、状态、数据、让引脚有这几个寄存器；重点是自己找到这些有用的寄存器、有用的位以及怎么设置。（详见[📖随记-通用学习笔记3](#) day23 和代码）

编程就是初始化，读取数据。初始化，主要就是让引脚有。读取数据，主要根据iic流程来。

## **\*\*D23-03 MPU6050原理**

MPU6050就是个测量三轴加速度、角速度的，传感器将测量值放到特定寄存器，cpu再到这个寄存器取；cpu也可以给传感器传参按照工作，比如加速度量程、角速度量程、低功耗模式等；取得、传参详见上面iic的过程。

详解主要讲了设置量程模式（等）、存放结果这几个寄存器；重点是自己找到这些有用的寄存器、有用的位以及怎么设置。（地址应该单独用个引脚设置）（详见[📖随记-通用学习笔记3](#) day23 和代码）编程就是初始化，读取数据。初始化，就是设置量程模式什么的。读取数据，主要根据iic流程来，用到了存放结果的那个寄存器。

计算机 D23-03 MPU6050原理.mp4

详见 [随记-通用学习笔记2](#)

## **D23-04 MPU6050寄存器读写时序**

详见 [📖随记-通用学习笔记3](#) day23，但融合到上面iic的过程里面吧

## **\*\*day24 IIC编程**

（觉得是不是还得学点spi）

## **\*\*D24-01 陀螺仪实验代码分析**

除了上面的初始化、初始化、读取数据，还有，不断地 读z轴角度高字节 读 低字节 合并 打印 延迟