■在 Linux 下搭建 51 单片机的开发烧写环境

在 Linux 下没有像 keli 那样好用的 IDE 来开发 51 单片机，开发环境只能自己搭建了。

第一步：安装交叉编译工具

a) 安装 SDCC

sudo apt-get install sdcc

b)测试 SDCC 是否可用，这是个网上找的简单的流水灯代码 test.c， 用来测试

```
#include "8051.h"
#define uint unsigned int
#define uchar unsigned char
uchar tab[8] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};
void Delay(uint xms)
{
uint i,j;
for(i=xms;i>0;i--)
for(j=110;j>0;j--);
}
void main()
{
uchar i;
while(1)
{
for(i=0;i<8;i++)
{
P1 = tab[i];
Delay(100);
}
}
}12345678910111213141516171819202122232425
```

编译它: sdcc test.c

会生成这么多的文件：

test.lk test.map test.rel test.sym test.asm test.ihx test.lst test.mem test.rst

我们只需要其中的 test.ihx

packihx file.ihx >file.hex 转换为 hex 文件

接着下载 hex2bin 文件，网址(http://sourceforge.net/projects/hex2bin/files/latest/download)。命令：hex2bin sourcefile.hex。之后就会生成 sourcefile.bin 文件。

hextobin file.hex 生成 bin 文件

注意：为了方便以后调用 hex2bin，可以将路径加入到 .bashrc 文件

在～/.bashrc 最后一行加上 Hex2bin 所在的文件夹位置

PATH=$PATH:/home/leo/workspace/c51/Hex2bin-2.31

可以写个 makefile 文件，编译方便些

这是我写的 makefile：

test.hex : test.c

sdcc test.c

packihx test.ihx > test.hex

hex2bin test.hex

clean:

rm -rf *.asm *.lst *.mem *.rst *.lnk *.rel *.sym *.ihx *.hex *.map

~ 1234567

第二步：安装烧写工具

a)下载 stcflash：github.com/laborer/stcflash，这是个用 python 写的向单片机烧写 bin 文件的软件

b)安装环境：sudo apt-get install python-serial

c)烧写： sudo python ./stcflash.py test.bin

■单片机片内存储器烧写(ROM 编程)

单片机应用系统由硬件和软件组成，软件的载体是硬件的程序存储器，程序存储器采用只读存储器，这种存储器在电源关闭后，仍能保存程序，在系统上电后，CPU 可取出这些指令重新执行。只读存储器(Read Only Memory,ROM)中的信息一旦写入，就不能随意更改，特别是不能在程序运行过程中写入新的内容，

故称只读存储器。向 ROM 中写入信息称为 ROM 编程。根据编程方式不同，

掩模 ROM.

在制造过程中编程，是以掩模工艺实现的，因此称为掩模 ROM。这种芯片存储结构简单，集成度高，但是由于掩模工艺成本较高，只适合于大批量生产。

可编程 ROM(PROM).

芯片出厂时没有任何程序信息，用独立的编程器写入。但是 PROM 只能写一次，写入内容后，就不能再修改。

EPROM.

用紫外线擦除，用电信号编程。在芯片外壳的中间位置有一个圆形窗口，对该窗口照射紫外线就可擦除原有的信息，使用编程器可将调试完毕的程序写入。

E2PROM(EEPROM).

用电信号擦除，用电信号编程。对 E2PROM 的读写操作与 RAM 存储器几乎没什么差别，只是写入速度慢一些，但断电后仍能保存信息。

Flash ROM.

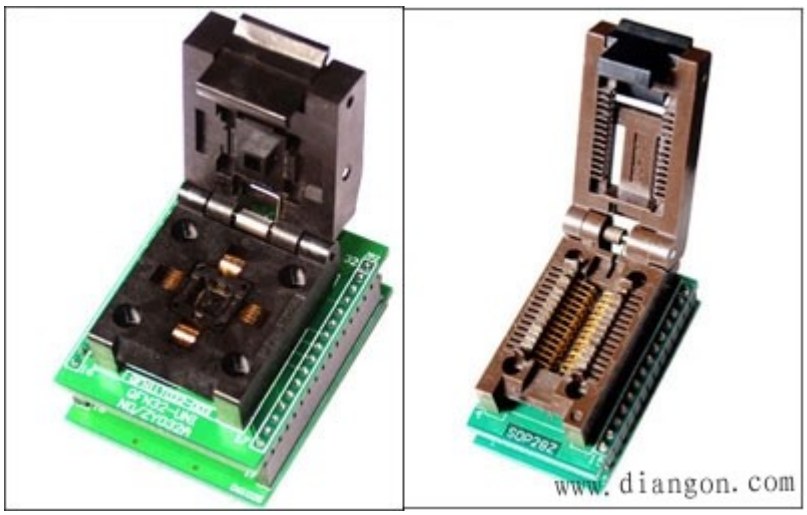
闪速存储器(简称闪存)，是在 EPROM 和 E2PROM 的基础上发展起来的一种电擦除型只读存储器。特点是可快速在线修改其存储单元中的数据，改写次数达一万次(ROM 都有改写次数)，读写速度快，存取时间可达 70ns，而成本比 E2PROM 低得多，因此正逐步取代 E2PROM。

注意：更多存储器内容请参考，《电子技术基础》数字部分(第五版) 主编 康华光. 第七章，或者 电工学(第七版)(下册) 主编 秦曾煌 第 22 章.

烧写器、烧录器、编程器、下载器、仿真器、调试器

单片机编程器(烧写器、烧录器)是用来将程序代码写入存储器芯片或者单片机内部的工具。编程器主要修改只读存储器中的程序，编程器通常与计算机连接，再配合编程软件使用。如下图所示是一个典型的编程器外形。图中黑色的是集成电路插座(也有下载编程器，不用取下芯片，涉及 ISP、ICP，后面有介绍)，通过拨动手柄可以将置于其中的集成电路芯片锁紧或松开。编程时锁紧以保证接触好，编程完毕松开，可以更换下一片芯片。

当芯片是双列直插形式时，直接插在座子上就可以烧写了，但是如果遇到下图所示的各种封装形式的芯片，又如何烧写呢?一种方法是使用适配器，下图是各种封装芯片所使用的适配器。



下载器(下载线)是将编译好的机器码写进单片机的工具。程序经过编译生成目标代码，然后通过下载软件和下载器下载到单片机的 ROM 或者 Flash 中。下载器主要来控制编程时序等。

仿真器(调试器)指以调试单片机软件为目的而专门设计制作的一套专用的硬件装置。单片机在体系结构上与 PC 机是完全相同的，也包括中央处理器，输入输出接口，存储器等基本单元，因而与 PC 机等设备的软件结构也是类似的。因为单片机在软件开发的过程中需要对软件进行调试，观察其中间结果，排除软件中存在的问题。但是由于单片机的应用场合问题，其不具备标准的输入输出装置，受存储空间限制，也难以容纳用于调试程序的专用软件，因此要对单片机软件进行调试，就必须使用单片机仿真器。单片机仿真器具有基本的输入输出装置，具备支持程序调试的软件，使得单片机开发人员可以通过单片机仿真器输入和修改程序，观察程序运行结果与中间值，同时对与单片机配套的硬件进行检测与观察，可以大大提高单片机的编程效率和效果。(许多仿真器具有下载器的功能)

片内 Flash 存储器的编程器

Flash 存储器是一种电擦除型只读存储器。它的主要特点是在不加电的情况下能长期保持存储的信息，又可快速在线修改其存储单元中的数据，其在线改写功能使单片机的程序改写和固化过程更加简便。目前很多单片机都用 Flash 作为片内程序存储器。当片内的 Flash 存储器满足要求时，不必扩展外部程序存储器。

下面讨论如何把已经调试完毕的程序写入 AT89S51 单片机中的片内 Flash 存储器。

AT89S51 单片机片内 4KB 的 Flash 存储器的特点：

可擦写寿命 10，000 次

数据保存时间 10 年

最大读取时间 150ns，页编程时间 10ns

具有 3 级加密算法，使得 AT89S 系列单片机的解密变得不可能，程序的加密性大大加强

AT89S51 出厂时，Flash 存储器处于全部空白状态(各单元均为 FFH)，可直接进行编程。若不全为空白状态(单元中有不是 FFH 的)，应首先将芯片擦除后，方可写入程序。(某些单片机出厂前会固化在 ROM 中一段 Bootloader 代码，其作用就是在启动操作系统之前进行初始化并引导系统，并能把我们下载在存储器中的程序加载到 RAM 中运行然后把控制权交还给系统。就好比 PC 上的 BIOS。ROM 编程跟 Bootloader 没有直接关系)

片内 Flash 存储器有低电压编程(Vpp=5V)和高电压编程(Vpp=12V)两类芯片。低电压编程可用于在线编程，高电压编程与一般常用的 EPROM 编程器兼容。在 AT89S51 芯片的封装面上标有低电压编程还是高电压编程的编程标志。不同的单片机的烧写电压，要视数据手册来定。

应用程序在 PC 中与在线仿真器以及用户目标板一起调试通过后，PC 中调试完毕的程序代码文件(.HEX 目标文件)，必须写入到 AT89S51 片内的 Flash 存储器中。目前常用的编程方法主要有两种：一种是使用通用编程器编程，另一种是使用下载型编程器进行编程。

通用编程器

通用编程器即不同的器件(比如不同的公司生产，不同架构)能普遍使用的编程器，它是编程器的功能分类下的一个子类，与之相对的是专用编程器。在硬件上,通用编程器的管脚驱动采用具有各种功能的通用结构，只需更新软件就能支持新的器件，具有很好的灵活性，满足目前和未来的器件需要。通常通用编程器锁紧座的每一个管脚可以用作时钟信号、数据总线、地址总线、控制总线、地线、电源线、VCCQ 电压、VPP 编程电压、管脚接触不良测量专用总线等等。驱动电压可以从 1.8V-25.0V，起码分 256 级可调。

下载型编程器 (自我理解)

常见的下载型编程器

ISPro 下载型编程器、ISP 编程器、ICP 编程器、ISP-ICP 编程器…

几种烧写方式

ICP(In Circuit Programing):在电路编程(单片机通电即可写入程序，包括 bootloader)

ISP(In System Programing):在系统编程(单片机通电加上晶振，有 bootloader 引导的情况下，即可写入应用程序)

IAP(In Application Programing):在应用编程(单片机在正常运行的程序中，内部程序对单片机局部重写数据)

ICP、ISP、IAP 详细介绍

ISP 是指可以在板级上进行编程，而不用把芯片拆下来放到烧写器中，即不脱离系统，所以称作“在系统编程”，它是对整个程序的擦除和写入，通过单片机专用的串行编程接口对单片机内部的 Flash 存储器进行编程。即使芯片焊接在电路板上，只要留出和上位机接口的串行口就能进行烧写。(芯片一般固化了用来 ISP 升级的 boot 程序)

IAP 同样是在板级上进行编程，MCU 获取新代码并对程序的某部分重新编程，即可用程序来改变程序，修改程序的一部分达到升级、消除 bug 的目的，而不影响系统的其它部分，烧写过程中程序可以继续运行，另外接口程序是自已写的，这样可以进行远程升级而不影响应用。IAP 的实现更加灵活,通常可利用单片机的串行口接到计算机的 RS232 口，通过专门设计的固件程序来编程内部存储器。(通常自己编写一个 bootloader 程序) (注: 有时候, ISP/IAP 的分别并不是那么清楚!)

ICP 在电路编程，ICP (In-circuit programmer)，在电路编程方法允许使用商业编程器来实现编程和擦除功能，而无需将微控制器从系统中移出，可完全由微控器硬件完成，不需要外部引导器。(百度)

ISP、IAP 应用场合

1、ISP 程序升级需要到现场解决,不过好一点的是不必拆机器了。

2、IAP 如果有网管系统的话,用网管下载一切搞定,人不用跑来跑去。

在线编程目前有两种实现方法：在系统编程(ISP)和在应用编程(IAP)。ISP 一般是通过单片机专用的串行编程接口对单片机内部的 Flash 存储器进行编程,而 IAP 技术是从结构上将 Flash 存储器映射为两个存储体,当运行一个存储体上的用户程序时,可对另一个存储体重新编程,之后将控制从一个存储体转向另一个。ISP 的实现一般需要很少的外部电路辅助实现,而 IAP 的实现更加灵活,通常可利用单片机的串行口接到计算机的 RS232 口,通过专门设计的固件程序来编程内部存储器。 ISP 和 IAP 很相似，都是不需要把芯片从板子上拔出来，就达到了用 PC-MCU 的编程接口(JTAG、串口、双绞线、SPI 等)搞定新版本的升级的目的。MCU 内部都是首先执行一段独立的 Boot 代码(这段 Boot 代码一般是出厂预置，或使用编程器烧录的，通常只有 1k 或 4k，SST 通常是占用一块独立的 Block，Philips 通常是让 BootROM 地址与其他 Flash 重叠，以达到隐藏的效果)，Boot 负责控制擦除程序存储器及给程序存储器编程的代码(或是处理器外部提供的执行代码)，然后通过某种与 PC 计算机的通信方式(如，ether 网口)，将用户指定的某个在 PC 上编译完成的 MCU 可运行的二进制代码文件编程入 MCU 内的程序存储器。

IAP 的实现

通常在用户需要实现 IAP 功能时，即用户程序运行中作自身的更新操作，需要在设计固件程序时编写两个项目代码，第一个项目程序不执行正常的功能操作，而只是通过某种通信管道(如 USB、USART)接收

程序或数据，执行对第二部分代码的更新;第二个项目代码才是真正的功能代码。这两部分项目代码都同时烧录在 User Flash 中，当芯片上电后，首先是第一个项目代码开始运行，它作如下操作：

1)检查是否需要对第二部分代码进行更新

2)如果不需要更新则转到 4)

3)执行更新操作

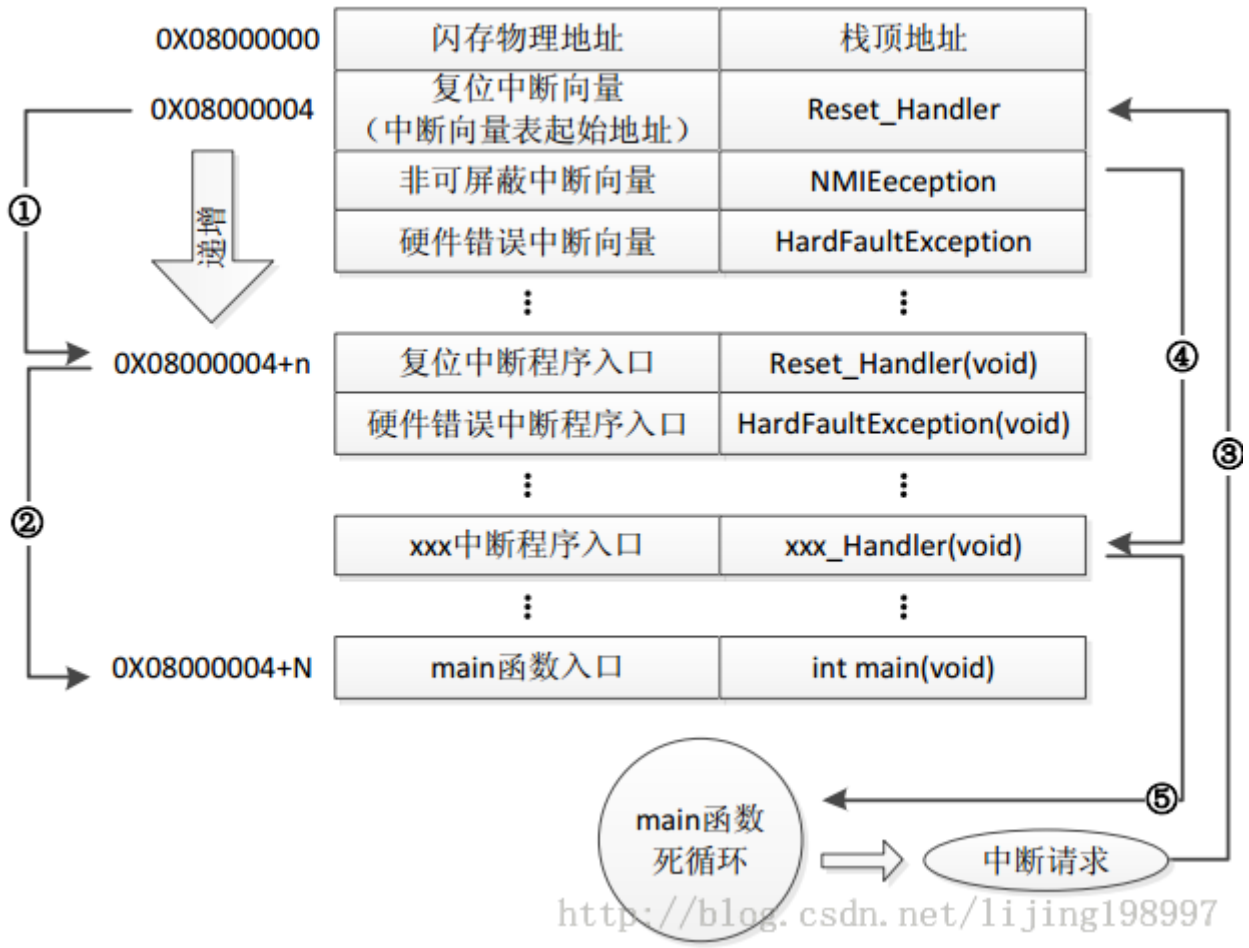
4)跳转到第二部分代码执行

第一部分代码必须通过其它手段，如 JTAG 或 ISP 烧入;第二部分代码可以使用第一部分代码 IAP 功能烧入，也可以和第一部分代码一道烧入，以后需要程序更新是再通过第一部分 IAP 代码更新。

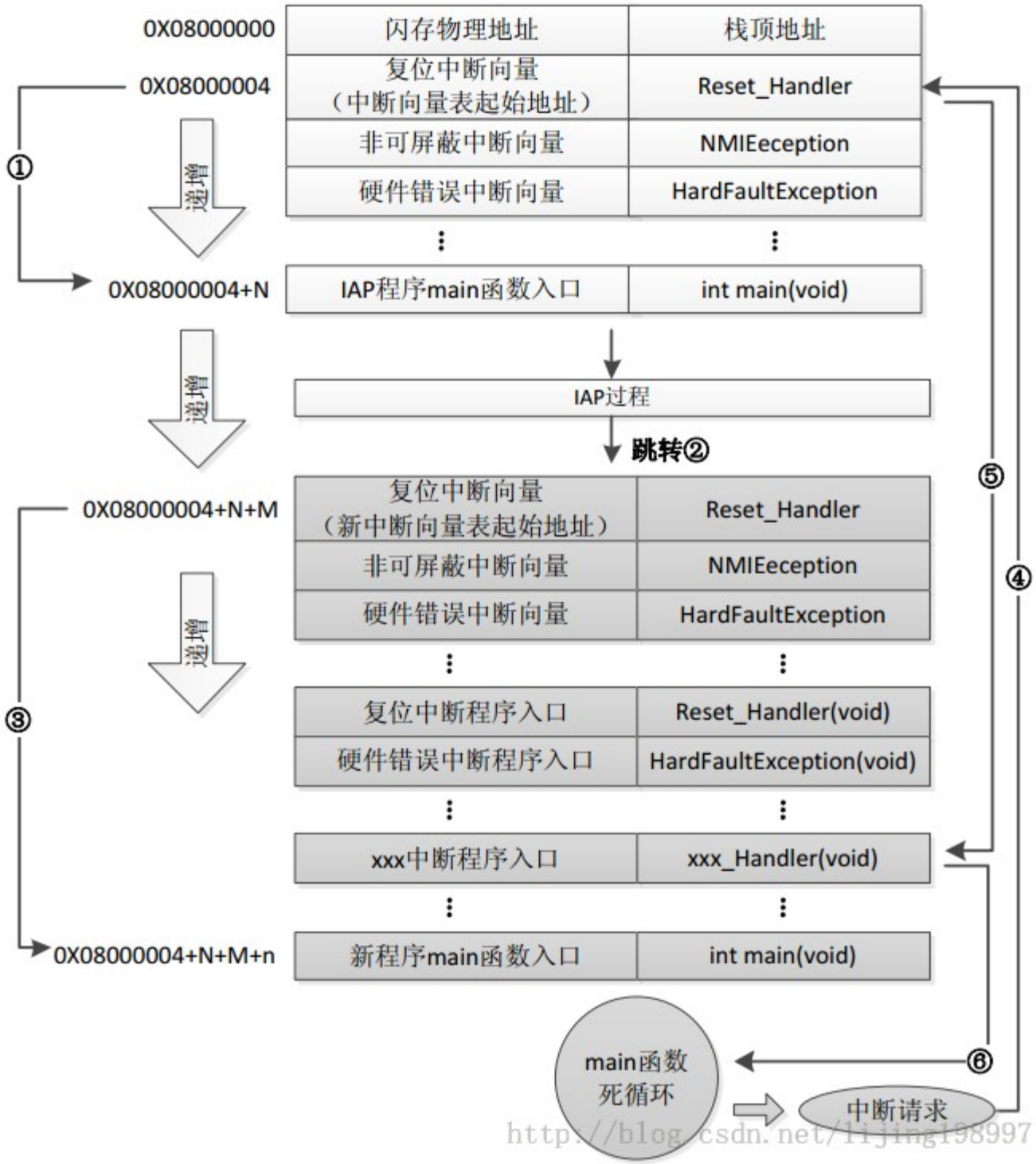
对于 STM32 来说，因为它的中断向量表位于程序存储器的最低地址区，为了使第一部分代码能够正确地响应中断，通常会安排第一部分代码处于 Flash 的开始区域，而第二部分代码紧随其后。

在第二部分代码开始执行时，首先需要把 CPU 的中断向量表映像到自己的向量表，然后再执行其他的操作。

STM32 正常的程序运行流程：



当加入 IAP 程序之后，程序运行流程：



从第二个图，STM32 复位后，还是从 0X08000004 地址取出复位中断向量的地址，并跳转到复位中断服务程序，在运行完复位中断服务程序之后跳转到 IAP 的 main 函数，如图标号①所示，此部分同图 53.1.1 一样;在执行完 IAP 以后(即将新的 APP 代码写入 STM32 的 FLASH，灰底部分。新程序的复位中断向量起始地址为 0X08000004+N+M)，跳转至新写入程序的复位向量表，取出新程序的复位中断向量的地址，并跳转执行新程序的复位中断服务程序，随后跳转至新程序的 main 函数，如图标号②和③所示，同样 main 函数为一个死循环，并且注意到此时 STM32 的 FLASH，在不同位置上，共有两个中断向量表。

在 main 函数执行过程中，如果 CPU 得到一个中断请求，PC 指针仍强制跳转到地址

0X08000004 中断向量表处，而不是新程序的中断向量表，如图标号④所示;程序再根据我们设置的中断向量表偏移量，跳转到对应中断源新的中断服务程序中，如图标号⑤所示;在执行完中断服务程序后，程序返回 main 函数继续运行，如图标号⑥所示。通过以上两个过程的分析，我们知道 IAP 程序必须满足两个要求：

1) 新程序必须在 IAP 程序之后的某个偏移量为 x 的地址开始;

2) 必须将新程序的中断向量表相应的移动，移动的偏移量为 x;

如果 IAP 程序被破坏，产品必须返厂才能重新烧写程序，这是很麻烦并且非常耗费时间和金钱的。针对这样的需求，STM32 在对 Flash 区域实行读保护的同时，自动地对用户 Flash 区的开始 4 页设置为写保护，这样可以有效地保证 IAP 程序(第一部分代码)区域不会被意外地破坏。

ISP 与 ICP 的差别

For ISP

(1) MCU 必须处于可执行程序的状态 (除了上电, 还要接 XTAL), 且必须预烧 ISP-code 在 LDROM 里面

(2) 烧录范围只限于 APROM, DataFlash 或 CONFIG (但对使用者来说, 应经够了!)

(3) chip 在 LOCK 的状态下, 仍然可以只更新某一区块 (APROM, DataFlash 或 CONFIG)

(4) 因为烧录的动作取决于 ISP-code 的写法, 所以给系统设计者的弹性较大

For ICP

(1) MCU 只要处于上电状态即可, 不必预烧任何 code 在 MCU 里面

(2) 烧录范围涵盖整颗 MCU, 包括 APROM, DataFlash, CONFIG, LDROM 和 ROMMAP

(3) chip 在 LOCK 的状态下, 无法只更新某一区块, 只能在 erase-ALL 之后, 更新某一区块, 再逐一烧回其它区块(因为 ICP 的本质就是走串行接口的 Writer Mode, chip 被 LOCK 之后, 除了 erase-ALL, 所有烧录动作皆会被禁止)

(4) 因为烧录纯粹是 ICP 硬件的行为, MCU 无法自己更新自己, 所以给系统设计者的弹性较小(例如: 无法藉由 ICP 去实现 IAP 的功能)

ISP 与 ICP 的使用场合

依这两者的特性, 配合客户的系统需求, 而后才建议客户使用 ISP 或 ICP.

注:上述差别的第(3)点和第(4)点, 可能是 ICP 带给使用者的最大限制. 站在使用者的观点, 若 ISP 与 ICP 只能择一的话, ISP 会是必要的选择。

ISP、IAP、ICP 与传统烧写方式的区别

1)在开发阶段

改 code 时, 不再需要将 MCU 从板子上拔起来, 拿到烧录器上烧, 然后再装回去. 可以直接利用 ISP/ICP Programmer 做板上烧录, 为开发者提供了极大的便利性.

2)在量产阶段

客户可以采用”先焊到板子上再烧 code”的方式, 将烧 code 的动作安排在生产线的某一站.

那么传统的方式 (先将 code 烧好再焊到板子上)有什么缺点?

传统的方式是这样的: 拆封–>从 tray 盘取出 chip–>烧录–>把 chip 放回 tray 盘.

这样的流程比起上面建议的方式: 增加了烧录时间, 容易造成 QFP 包装的 chip 弯脚, 或忘了烧 code 即放回 tray 盘.

3)在成品阶段

已组装好的成品若要改 code, 可以透过预留的接口, 利用 ISP 或 ICP, 更新 MCU, 不需要拆机.

■烧写程序(下载程序)基础知识

简单整理下固件、烧写的定义以及烧写过程的几种方式。

一、什么是烧写(下载)和固件

嵌入式系统需要预先将程序写入 ROM/FLASH 中,在启动时载入运行这些程序,将已经生成的程序写入 ROM/FLASH 的过程叫做烧写。这样固化在存储介质上的文件叫固件(firmware)。那么烧写即为更新固件的过程。

二、常用的几种烧写方式

把系统当做一个存储区域进行烧写;

采用专门接口及专用的下载线将程序烧写到固件中。这时 CPU 除了执行单片机本身的指令之外，还能执行对存储器进行操作的特殊指令，擦除、烧写和校验指令。先通过传输相应的指令给 CPU 执行(擦除、读入、烧写、校验等)，这样完成烧写。现在普遍使用的 JTAG 仿真器也是这样，CPU 能执行 JTAG 的特殊指令，完成对固件的烧写操作。

引导程序，即单片机中已经存在了一个烧写程序。启动单片机时首先运行这程序，程序判断端口状态，识别为烧写的特殊状态，就从某个端口读取数据，然后写入到存储单元中。如果是正常的状态，就转到用户的程序开始执行。需要触发硬件选择用户程序模式还是烧写模式。

三、关于 ISP、ICP、IAP 三种编程方式区别

ISP：In System Programing，在系统编程

ICP：In Circuit Programing，在电路编程

IAP：In Applicating Programing，在应用编程

ICP 工作方式：只需要上电，处理器无需工作，更新整个存储内容的方式，不需要处理器中间操作，不需要存在引导程序，属于硬件单独处理行为，芯片编成方式??

ISP 工作方式：可以编程写入最终用户代码，而不需要从电路板上取下器件，已经编程的器件也可以用 ISP 方式擦除或再编程，理解上需要一些必要的硬件逻辑或引导代码，如使用 JTAG 下载;

IAP 工作方式:IAP 从结构上将 Flash 存储器映射为多个存储体区域(引导区、运行区、下载存储区等等)，第一个项目程序存放在应用引导区，不执行正常的功能操作，而只是通过某种通信管道接收程序或数据，执行对其他部分代码的更新;其他代码才是真正的功能代码，存放在程序存储区。当芯片上电后，首先运行应用引导区程序，它作如下操作，检查是否需要对第二部分代码进行更新，存在更新则进行更新后执行，否则直接执行。所以引导区程序应该不可以 IAP 方式更新。

个人理解：

IAP 的方式最为灵活，但是由于使用方式的受到限制，无法对存储区域的分配进行重新的划分，引导区也应该无法自修改。

ICP 硬件基础型=>ISP 借助外力型=>IAP 自由独立型

Linux 系统下烧录单片机

有些人说 **Linux** 下怎么开发单片机程序? 我们学得都是在 Windows 下的开发工具和烧录工具。其实这只是中国的情况，中国大都是用 Windows 系统的。所以我们也只接触到 Windows 下的开发工具。在国外，许多大牛是不用 Windows 的。那么，他们如果要做单片机怎么办?肯定没问题的，Linux 也有许多的单片机和**嵌入式**的开发工具。下面，我以 51 系列单片机为例，介绍一下怎么使用 Linux 系统玩单片机。

首先，说一下我的硬件和软件情况。

硬件：

一台笔记本、 一个 51 单片机开发板(这个在网上有很多卖，也就 150 左右的价格)。单片机芯片有两块：一块是 Atmel 的 AT89S52， 另一块是 STC 的 STC89C52RC 。USB 转串口线一根(串口通信和 STC 下载用)。USBISP 下载线一根(AT89S52 下载用)。

软件：

操作系统：Ubuntu 11.04 (内核 2.6.38-10-generic)、 sdcc(单片机 C 语言编译器)、 avrdude(AT 单片机下载程序)、gSTCISP(STC 单片机下载程序) 、objcopy(GUN 开发工具里的 ，应该默认 Linux 都会安装)

操作系统的安装就不讲了。

SDCC 可以在网上下到源代码然后编译安装。/*SDCC 也有 Windows 版本，不过我这只讲 Linux 下的东西，如果有兴趣请自己百度 Google 一下*/

不过这样很麻烦，而且不一定安装的上。我们的 ubuntu 有超级牛力(不知道这个?在终端下输入 apt-get 就知道啦。)直接在终端下输入以下命令：

```
sudoapt-get install sdcc sdcc-doc
```

先讲 AT89S52 的下载

首先是安装 AT89S52 的下载软件，还是用 ubuntu 的超级牛力：

```
sudo apt-get install gcc-avr binutils-avr avrdude avr-libc
```

然后编辑 avrdude.conf 以使得 avrdude 可以下载 51 的单片机

```
sudo gedit/etc/avrdude.conf
```

在最后面加入 AT89s52 的相关信息(这是 JoyShukla 写的)

```
#------------------------------------------------------------
# Below chips by added by me .. Joy Shukla(joy_shukla@yahoo.in)
# these chips can be programmed with my usbasp programmer(changedatmega8 program)
# after adding avrdude support in this file ..
#------------------------------------------------------------
#------------------------------------------------------------
# AT89S52
#------------------------------------------------------------
part
id = "8052";
desc = "AT89S52";
signature = 0x1E 0x52 0x06;
chip_erase_delay = 20000;
pgm_enable = "1 0 1 0 1 1 0 0 0 1 0 1 0 0 1 1",
"x x x x x x x x x x x x x x x x";
chip_erase = "1 0 1 0 1 1 0 0 1 0 0 x x x x x",
"x x x x x x x x x x x x x x x x";
timeout = 200;
stabdelay = 100;
cmdexedelay = 25;
```

```
synchloops = 32;
bytedelay = 0;
pollindex = 3;
pollvalue = 0x53;
predelay = 1;
postdelay = 1;
pollmethod = 0;
memory "flash"
size = 8192;
paged = no;
min_write_delay = 4000;
max_write_delay = 9000;
readback_p1 = 0xff;
readback_p2 = 0xff;
read = " 0 0 1 0 0 0 0 0",
" x x x a12 a11 a10 a9 a8",
" a7 a6 a5 a4 a3 a2 a1 a0",
" o o o o o o o o";
write = " 0 1 0 0 0 0 0 0",
" x x x a12 a11 a10 a9 a8",
" a7 a6 a5 a4 a3 a2 a1 a0",
" i i i i i i i i";
mode = 0x21;
delay = 12;
;
memory "signature"
size = 3;
read = "0 0 1 0 1 0 0 0 x x x 0 0 0 a1 a0",
"0 0 0 0 0 0 0 0 o o o o o o o o";
;
;
```

#----------

然后保存

现在就可以烧录 AT89S52 的单片机了，

用命令

sudo avrdude -p 8052 -c usbasp -e -U flash:w:'跑马灯.hex'

就可以烧录了 你可以把命令中的”跑马灯.hex”换成你自己的程序。

如果觉得 命令太长 太难记， 那么推荐你使用 makefile 来自动完成。

我是这样做的

在工作目录下建立一个 makefile，内容如下：

###########内容开始##############

main：

sdcc liushui.c #注意这一行以及一下每行开头都是一个 Tab

packihxliushui.ihx>liushui.hex #SDCC 默认的输出格式是 ihx 我们把它转换成 hex

sudo avrdude -p 8052 -cusbasp -e -U flash:w:liushui.hex #这就是烧录的语句，其实最后的格式是 ihx 也可以烧录的

##########内容结束###############

保存之后 在工作目录下只要输入一句 make 就可以完成 编译 烧录 一步到位了。

/*当然你得保障程序能正常编译，下载器和单片机都连接好了*/

接下来讲讲如何烧录 STC 单片机

STC 的程序编写 不用我说了。它的下载是用 gSTCISP 貌似是一个中国人写的，还是图形界面的。

gSTCISP 下载地址如下

http://forum.ubuntu.org.cn/download/file.PHP?id=104628&sid=767b624faf73a36c71fcd4f8111773e

gSTCISP 下载要求的格式是 bin 我在网上也没找到所谓的 Hex2bin 程序 ，不过 发现用 objcopy 也可以把 hex 文件转化为 bin'文件，姑且我就先这么用了。

生成 hex 文件后，用如下命令将 hex 转化为 bin

objcopy -I ihex -Obinary liushui.hex liushui.bin # I 和 O 都是大写

然后 打开 gSTCISP (我直接把它 cp 到了/usr/bin，就可以直接在提示符下输入 gSTCISP 运行了)

选择 port(端口) 我的是/dev/ttyUSB0 然后选择文件 选择刚刚生成的 bin 文件 点击 DownLoad

好像 STC 的是要先断电再上电的时候 就能把程序写入单片机了

当你看到 Program OK Havealready encrypt.就说明你烧录成功啦。

写在最后：

我其实也是一个小菜鸟，这其实只是我的一次笔记而已。而且有些地方还写得很罗嗦。高手可以直接略过啦。使用 Linux 做单片机可以不用学怎么去用鼠标操纵不同图形界面的 IDE(集成开发环境)，只需考虑自己要做的程序，然后记住需要的编译选项，而且 SDCC 是 windows Linux 通用的，在 linux 写的代码，makefile 在 windows 下甚至可以不用改 或者改动少许就能使用。而且 Linux 不用去装什么下载线驱动，我的 usbisp 下载线 和 usb 转串口下载线都是直接插上就可以下载了。

本文参考资料：

蝶晓梦的百度空间 /* 看了这篇日志才使我成功的在 Linux 下烧录成功 */

http://hi.baidu.com/%B5%FB%CF%FE%C3%CE/blog/item/1aa1050958898c2e6a60fbd0.html

在 linux 系统下去烧录单片机 /* 这篇介绍了很多在 Linux 下的单片机的工具，不仅仅是 51 的 */

##

我写的: 尼玛, 自从做单片机后, 好久没回 linux 系统!

■单片机 ISP、IAP 和 ICP 几种烧录方式的区别

单片机 ISP、IAP 和 ICP 几种烧录方式的区别

玩单片机的都应该听说过这几个词。一直搞不太清楚他们之间的区别。今天查了资料后总结整理如下。

ISP：In System Programing，在系统编程

IAP：In applicating Programing，在应用编程

ICP：In Circuit Programing，在电路编程

ISP 是指可以在板级上进行编程，也就是不用拆芯片下来，写的是整个程序，一般是通过 ISP 接口线来写。

IAP 虽然同样也是在板级上进行编程，但是是自已对自已进行编程，在应用中进行编程，也即可以只是更改某一部分而不影响系统的其它部分，另外接口程序是自已写的，这样可以进行远程升级而不影响应用。

打个比喻吧：

1、ISP 是把房子拆了再重造一间，那么在造好之前当然是不能住人的啦!

2、IAP 是在造好的房子里边进行一些装修，当然人可以继续住啦!

ICP 是在电路编程，是 LPC 芯片的编程方式，相当于 ATMEL 的 ISP，而 LPC 的 ISP 就相当于 ATMEL 的 IAP 了。

ISP 于 IAP 的区别

ISP：in system programming, 在系统编程

IAP: in applicatin programming 在应用编程

但两者的操作方式,结果和应用场合有什么区别

什么是 ISP:

用写入器将 code 烧入,不过,芯片可以在目标板上,不用取出来,在设计目标板的时候就将接口设计在上面,所以叫"在系统编程",即不用脱离系统;

什么是 IAP:

在应用编程,有芯片本身(或通过外围的芯片)可以通过一系列操作将 code 写入,比如一款支持 Iap 的单片机,内分 3 个程序区,1 作引导程序区,2 作运行程序区,3 作下载区,芯片通过串口接收到下载命令,进入引导区运行引导程序,在引导程序下将 new code 内容下载到下载区,下载完毕并校验通过后再将下载区内容复制到 2 区,运行复位程序,则 Iap 完成;

应用场合:

1,ISP 程序升级需要到现场解决,不过好一点的是不必拆机器了;

2,IAP 如果有网管系统的话,用网管下载一切搞定,人不用跑来跑去,

这可能是他们的优点或应用吧

在线编程目前有两种实现方法：在系统编程(ISP)和在应用编程(IAP)。ISP 一般是通过单片机专用的串行编程接口对单片机内部的 Flash 存储器进行编程,而 IAP 技术是从结构上将 Flash 存储器映射为两个存储体,当运行一个存储体上的用户程序时,可对另一个存储体重新编程,之后将控制从一个存储体转向另一个。ISP 的实现一般需要很少的外部电路辅助实现,而 IAP 的实现更加灵活,通常可利用单片机的串行口接到计算机的 RS232 口,通过专门设计的固件程序来编程内部存储器。 ISP 和 IAP 很相似，都是不需要把芯片从板子上拔出来，就达到了用 PC-MCU 的编程接口(JTAG、串口、双绞线、SPI 等)搞定新版本的升级的目的。MCU 内部都是首先执行一段独立的 Boot 代码(这段 Boot 代码一般是出厂预置，或使用编程器烧录的，通常只有 1k 或 4k，SST 通常是占用一块独立的 Block，Philips 通常是让 BootROM 地址与其他 Flash 重叠，以达到隐藏的效果)，Boot 负责控制擦除程序存储器及给程序存储器编程的代码(或是处理器外部提供的执行代码)，然后通过某种与 PC 计算机的通信方式(如，ether 网口)，将用户指定的某个在 PC 上编译完成的 MCU 可运行的二进制代码文件编程入 MCU 内的程序存储器。

ISP 和 IAP 最大的不同是：由谁来触发。

ISP 有 4 种触发方式：

1.由外部硬件电路：如 VDD 保持高电平，给 RST 连续 3 个脉冲;

2.检测状态位：如 ISPEN，为 0 时 PC 指针从 0000H 开始执行;为 1 时，通过“引导向量”计算出“ISP 代码”的位置。每次复位后都会检测该状态位;

3.中止控制符信号触发芯片复位：中止控制符信号就是指在异步串行口的接收脚上出现长 达一帧长度的低电平，这里一帧的长度与异步串行口的工作模式有关。

4.直接调用 ISP：用户程序也可以调用，但是很危险。

4 种方式的目的是相同的——进入 ISP 子程序，比如 Philips 出厂的 ISP 子程序在 1E00H-1FFFH，只要能引导 PC 指针指向 1E00H 就可以了。

而进入 ISP 代码的目的是：进入 BootROM。

IAP 的触发比较简单一些，没有外部触发。通过一些指示位(SST 为 SC0/SC1、SFCF;Philips 为一段 IAP 子程序，保存在 FF00H～FFFFH 地址空间中)，达到引导到 BootROM 的目的。

殊途同归，ISP、IAP 所进入的 BootROM 里面驻留的 Boot 代码，才是最终目标。

最后一种：并行编程模式。不需要 BootROM，直接设置芯片的多个引脚来让芯片识别命令(如：擦除、写入、验证等)，从 P 口传地址、数据，就可以写入 Flash

用 LPC2000 的 IAP，你自己分配好 FLASH 空间，指定一个 BLOCK 用来存放你的数据，然后通过 IAP 进行写操作。每次开机后，从这个 BLOCK 读你的数据。

IAP 的实现

通常在用户需要实现 IAP 功能时，即用户程序运行中作自身的更新操作，需要在设计固件程序时编写两个项目代码，第一个项目程序不执行正常的功能操作，而只是通过某种通信管道(如 USB、USART)接收程序或数据，执行对第二部分代码的更新;第二个项目代码才是真正的功能代码。这两部分项目代码都同时烧录在 User Flash 中，当芯片上电后，首先是第一个项目代码开始运行，它作如下操作：

1)检查是否需要对第二部分代码进行更新

2)如果不需要更新则转到 4)

3)执行更新操作

4)跳转到第二部分代码执行

第一部分代码必须通过其它手段，如 JTAG 或 ISP 烧入;第二部分代码可以使用第一部分代码 IAP 功能烧入，也可以和第一部分代码一道烧入，以后需要程序更新是再通过第一部分 IAP 代码更新。

对于 STM32 来说，因为它的中断向量表位于程序存储器的最低地址区，为了使第一部分代码能够正确地响应中断，通常会安排第一部分代码处于 Flash 的开始区域，而第二部分代码紧随其后。

在第二部分代码开始执行时，首先需要把 CPU 的中断向量表映像到自己的向量表，然后再执行其他的操作。

如果 IAP 程序被破坏，产品必须返厂才能重新烧写程序，这是很麻烦并且非常耗费时间和金钱的。针对这样的需求，STM32 在对 Flash 区域实行读保护的同时，自动地对用户 Flash 区的开始 4 页设置为写保护，这样可以有效地保证 IAP 程序(第一部分代码)区域不会被意外地破坏。

ISP(In-System Programming)在系统可编程，指电路板上的空白器件可以编程写入最终用户代码，而不需要从电路板上取下器件，已经编程的器件也可以用 ISP 方式擦除或再编程。IAP(In-Application Programming)指 MCU 可以在系统中获取新代码并对自己重新编程，即可用程序来改变程序。ISP 和 IAP 技术是未来仪器仪表的发展方向。

1 ISP 和 IAP 的工作原理

ISP 的实现相对要简单一些，一般通用做法是内部的存储器可以由上位机的软件通过串口来进行改写。对于单片机来讲可以通过 SPI 或其它的串行接口接收上位机传来的数据并写入存储器中。所以即使我们将芯片焊接在电路板上，只要留出和上位机接口的这个串口，就可以实现芯片内部存储器的改写，而无须再取下芯片。

IAP 的实现相对要复杂一些，在实现 IAP 功能时，单片机内部一定要有两块存储区，一般一块被称为 BOOT 区，另外一块被称为存储区。单片机上电运行在 BOOT 区，如果有外部改写程序的条件满足，则对存储区的程序进行改写操作。如果外部改写程序的条件不满足，程序指针跳到存储区，开始执行放在存储区的程序，这样便实现了 IAP 功能。

2 ISP 和 IAP 在单片机中的应用

2.1 传统编程方法的不足

在一般的单片机的实验或开发时，编程器是必不可少的装置。仿真、调试完的程序需要借助编程器烧到单片机内部或外接的程序存储器中。普通的编程器价格从几百元到几千元不等，对于一般的单片机用户来说还是一笔不小的开支。另外，在开发过程中，程序每改动一次就要拔下电路板上的芯片编程后再插上，这样不但麻烦也很容易对芯片和电路板造成损伤，另外在程序需要升级做改动时，必须将设备返厂或是技术人员到现场操作，既不方便也造成成本浪费。

2.2 ISP 和 IAP 的优点

ISP 技术的优势是不需要编程器就可以进行单片机的实验和开发，单片机芯片可以直接焊接到电路板上，调试结束即成成品，免去了调试时由于频繁地插入取出芯片对芯片和电路板带来的不便。

IAP 技术是从结构上将 Flash 存储器映射为两个存储体，当运行一个存储体上的用户程序时，可对另一个存储体重新编程，之后将程序从一个存储体转向另一个。

ISP 的实现一般需要很少的外部电路辅助实现，而 IAP 的实现更加灵活，通常可利用单片机的串行口接到计算机的 RS232 口，通过专门设计的固件程序来编程内部存储器，可以通过现有的 INTERNET 或其它通讯方式很方便地实现远程升级和维护。

2.3 产品分析

目前市场上不少的单片机具有 ISP 和 IAP 功能。

ST 公司的 μPSD32××系列单片机片内带 128KB/256KB 的 FLASH 存储器及 32KB Boot ROM,通过 JTAG 串行口能很容易地实现 ISP 功能。IAP 功能则可由用户在应用中实现。

ATMEL 公司的单片机 AT89S8252，提供了一个 SPI 串行接口对内部程序存储器编程(ISP)。SST 公司的单片机 SST89C54，其最大的特点内部有两块独立的 FLASH 存储器,具有 IAP(应用中在线编程)功能，对于 8052 系列的其它公司的单片机(ATMEL、 WINBOND、ISSI)可以直接代换，软硬件无须做任何改动。

PHILIPS 公司的 P89C51RX2xx 系列是带 ISP/IAP 的 8 位 Flash 单片机。PHILIPS 公司为了使 ISP 技术和 IAP 技术得以推广，在芯片上免费提供了 Boot ROM 固件，并且巧妙地解决了固件和 FLASH 的地址覆盖问题及一些具体实现细节问题，使它们的实现变得简单。

此外在外围器件中 ST 公司的 PSD 系列产品片内带大容量存储器，支持 ISP 及 IAP 功能。

ISP，In System Programing

主要是指代芯片的烧写方式，以往写片子需要把片子拿下来，离开电路，用编程器烧，换句话说，芯片不能不脱离应用系统进行写入。ISP 主要针对这个问题，使用 JTAG 或者串行口(MCU 内部有 Boot Loader ，通过指定的方式激活之，它可以和 PC 或其它上位机通过串口联系，不用使单片机离开应用系统而更新内部的程序/设置)进行程序的烧写操作。

因此，具有 ISP 功能的 MCU 可以不使用编程器进行编程。当然，实现 ISP 可能需要一些硬件电路支持，具体的在数据手册中有说明。IAP，In Application Programing 单片机内部具有一些可擦写的非易失存储器，如 Flash。在单片机独立运行时，不具备 IAP 功能的单片机并不能对 Flash 的数据进行修改，比如，对自身的某一个模块的代码，数据进行修改。具备了 IAP 功能的 MCU ，能够通过使用各自公司开发的技术，对于自身进行修改。

简言之：ISP=>修改 MCU 内部数据需要有外部介入;IAP=>修改 MCU 内部数据可以不用外部介入。二者可以说是数据更新的一种实现机制。一般具备 ISP 功能后，就不要编程器了，而是使用下载线进行编程工作。但是不是说他们就一定不支持编程器了，具体型号具体分析。是否需要仿真器进行仿真和是否具备 ISP 和 IAP 没有必然的联系。只不过具备了 IAP 功能，可以在 MCU 内写入监控程序，模拟一个仿真器，当然，这个监控程序是要消耗资源的，和使用硬件的仿真器还有一定的差异。IAP 的应用，在 51 上可能比较多，就是 SST 出的那个 SoftICE 功能。他的 MCU 其实不具备 ISP 功能，需要使用编程器烧一个 Boot Loader 进去，然后，利用 IAP 功能，就可以通过串行口下载数据了。SST 的那个，使用的是自己开发的一个 SoftICE 软件，和 Keil C 的 Mon51 接口。看 DataSheet 上说，先要用编程器烧 Boot Loader 进去，然后，用串口烧 SoftICE 进去，然后就可以用了。

■单片机的烧写原理

我要问的不是如何制作烧写接口。 而是，电脑在通过接口向单片机烧写时(此时单片机 内没有程序)，电脑是如何通过接口访问内部 rom 的??对应 p 口怎么就会指向单片机内部 rom(就像单片机是一个 rom?

有三种方式：

1.把单片机当做一个 ROM 芯片，早期的单片机都是如此。将单片机放在通用编程上编程时，就像给 28C256 这样的 ROM 中写程序的过程一样。只是不同的单片机使用的端口，编程用的时序不一样。

2.像 AT89S52 或 AVR 单片机一样，在单片机上有 SPI 接口，这时用专用的下载线将程序烧写到单片机中。这时不同的是，单片机的 CPU 除了执行单片机本身的指令之外，还能执行对 ROM 进行操作的特殊指令，如 ROM 擦除、烧写和校验指令。在编程 ROM 时，下载线先通过传输这些指令给 CPU 执行(擦除 ROM、读入数据、烧写 ROM、和校验 ROM)，这样完成对单片机的 ROM 的烧写。此外，现在普遍使用的 JTAG 仿真器也是这样，单片机的 CPU 能执行 JTAG 的特殊指令，完成对 ROM 的烧写操作。

3.引导程序，即单片机中已经存在了一个烧写程序。启动单片机时首先运行这程序，程序判断端口状态，如果符合“要烧写 ROM”的状态存在，就从某个端口(串口、SPI 等等)读取数据，然后写入到单片机的 ROM 中。如果没有“要烧写 ROM”的状态，就转到用户的程序开始执行。像 AVR 单片机的 bootloader 方式、STC 的串口下载方式，还有其他单片机的串口编程等等都是这样。