快速掌握一款新型 MCU 的方法(推荐)

转自 蠢蠢欲动的单片机

http://blog.21ic.org/more.asp?name=zhangdage&id=5662

任何一款 MCU，其基本原理和功能都是大同小异，所不同的只是其外围功能模块的配置及数量、指令系统等。对于指令系统，虽然形式上看似千差万别，但实际上只是符号的不同，其所代表的含义、所要完成的功能和寻址方式基本上是类似的。因此，对于任何一款 MCU，主要应从如下的几个方面来理解和掌握：

* MCU 的特点：要了解一款 MCU，首先需要知道就是其 ROM 空间、RAM 空间、IO 口数量、定时器数量和定时方式、所提供的外围功能模块（Peripheral Circuit）、中断源、工作电压及功耗等等。

* 了解这些 MCU Features 后，接下来第一步就是将所选 MCU 的功能与实际项目开发的要求的功能进行对比，明确那些资源是目前所需要的，那些是本项目所用不到的。对于项目中需要用到的而所选 MCU 不提供的功能，则需要认真理解 MCU 的相关资料，以求用间接的方法来实现，例如，所开发的项目需要与 PC 机 COM 口进行通讯，而所选的 MCU 不提供 UART 口，则可以考虑用外部中断的方式来实现；

* 对于项目开发需要用到的资源，则需要对其 Manua*进行认真的理解和阅读，而对于不需要的功能模块则可以忽略或浏览即可。对于 MCU 学习来讲，应用才是关键，也是最主要的目的。

* 明确了 MCU 的相关功能后，接下来就可以开始编程了。对于初学者或初次使用此款 MCU 的设计者来说，可能会遇到很多对 MCU 的功能描述不明确的地方，对于此类问题，可以通过两种方法来解决，一种是编写特别的验证程序来理解资料所述的功能；另一种则可以暂时忽略，程序设计中则按照自己目前的理解来编写，留到调试时去修改和完善。前一种方法适用于时间较宽松的项目和初学者，而后一种方法则适合于具有一定 MCU 开发经验的人或项目进度较紧迫的情况；

* 指令系统千万不要特别花时间去理解。指令系统只是一种逻辑描述的符号，只有在编程时根据自己的逻辑和程序的逻辑要求来查看相关的指令即可，而且随着编程的进行，对指令系统也会越来越熟练，甚至可以不自觉地记忆下来；

MCU 的基本功能：

对于绝大多数 MCU，下列功能是最普遍也是最基本的，针对不同的 MCU，其描述的方式可能会有区别，但本质上是基本相同的：

* Timer（定时器）：Timer 的种类虽然比较多，但可归纳为两大类：一类是固定时间间隔的 Timer，即其定时的时间是由系统设定的，用户程序不可控制，系统只提供几种固定的时间间隔给用户程序进行选择，如 32Hz，16Hz，8Hz 等，此类 Timer 在 4 位 MCU 中比较常见，因此可以用来实现时钟、计时等相关的功能；另一类则是 Programmable Timer（可编程定时器），顾名思义，该类 Timer 的定时时间是可以由用户的程序来控制的，控制的方式包括：时钟源的选择、分频数（Prescale）选择及预制数的设定等，有的 MCU 三者都同时具备，而有的则可能是其中的一种或两种。此类 Timer 应用非常灵活，实际的使用也千变万化，其中最常见的一种应用就是用其实现 PWM 输出（具体的应用，后续会有特别的介绍）。由于时钟源可以自由选择，因此，此类 Timer 一般均与 Event Counter（事件计数器）合在一起；

* IO 口：任何 MCU 都具有一定数量的 IO 口，没有 IO 口，MCU 就失去了与外部沟通的渠道。根据 IO 口的可配置情况，可以分为如下几种类型：

** 纯输入或纯输出口：此类 IO 口有 MCU 硬件设计决定，只能是输入或输出，不可用软件来进行实时的设定；

** 直接读写 IO 口：如 MCS-51 的 IO 口就属于此类 IO 口。当执行读 IO 口指令时，就是输入口；当执行写 IO 口指令则自动为输出口；

** 程序编程设定输入输出方向的：此类 IO 口的输入或输出由程序根据实际的需要来进行设定，应用比较灵活，可以实现一些总线级的应用，如 I2C 总线，各种 LCD、LED Driver 的控制总线等；

** 对于 IO 口的使用，重要的一点必须牢记的是：对于输入口，必须有明确的电平信号，确保不能浮空（可以通过增加上拉或下拉电阻来实现）；而对于输出口，其输出的状态电平必须考虑其外部的连接情况，应保证在 Standby 或静态状态下不

存在拉电流或灌电流。

* 外部中断：外部中断也是绝大多数 MCU 所具有的基本功能，一般用于信号的实时触发，数据采样和状态的检测，中断的方式由上升沿、下降沿触发和电平触发几种。外部中断一般通过输入口来实现，若为 IO 口，则只有设为输入时其中断功能才会开启；若为输出口，则外部中断功能将自动关闭（ATMEL 的 ATiny 系列存在一些例外，输出口时也能触发中断功能）。外部中断的应用如下：

** 外部触发信号的检测：一种是基于实时性的要求，比如可控硅的控制，突发性信号的检测等；而另一种情况则是省电的需要；

** 信号频率的测量；为了保证信号不被遗漏，外部中断是最理想的选择；

** 数据的解码：在遥控应用领域，为了降低设计的成本，经常需要采用软件的方式来对各种编码数据进行解码，如 Manchester 和 PWM 编码的解码；

** 按键的检测和系统的唤醒：对于进入 Sleep 状态的 MCU，一般需要通过外部中断来进行唤醒，最基本的形式则是按键，通过按键的动作来产生电平的变化；

* 通讯接口：MCU 所提供的通讯接口一般包括 SPI 接口，UART，I2C 接口等，其分别描述如下：

** SPI 接口：此类接口是绝大多数 MCU 都提供的一种最基本通讯方式，其数据传输采用同步时钟来控制，信号包括：SDI（串行数据输入）、SDO（串行数据输出）、SCLK（串行时钟）及 Ready 信号；有些情况下则可能没有 Ready 信号；此类接口可以工作在 Master 方式或 Slave 方式下，通俗说法就是看谁提供时钟信号，提供时钟的一方为 Master，相反的一方则为 Slaver；

** UART（Universal Asynchronous Receive Transmit）：属于最基本的一种异步传输接口，其信号线只有 Rx 和 Tx 两条，基本的数据格式为：
Start Bit + Data Bit(7-bits/8-
bits) + Parity Bit(Even, Odd or None) + Stop Bit(1~2Bit)。一位数据所占的时间称为 Baud Rate（波特率）。对于大多数的 MCU 来讲，数据为的长度、数据校

验方式（奇校验、偶校验或无校验）、停止位（Stop Bit）的长度及 Baud Rate 是可以通过程序编程进行灵活设定。此类接口最常用的方式就是与 PC 机的串口进行数据通讯。

** I2C 接口：I2C 是由 Philips 开发的一种数据传输协议，同样采用 2 根信号来实现：SDAT（串行数据输入输出）和 SCLK（串行时钟）。其最大的好处是可以在此总线上挂接多个设备，通过地址来进行识别和访问；I2C 总线的一个最大的好处就是非常方便用软件通过 IO 口来实现，其传输的数据速率完全由 SCLK 来控制，可快可慢，不像 UART 接口，有严格的速率要求。

* Watchdog（看门狗定时器）：Watchdog 也是绝大多数 MCU 的一种基本配置（一些 4 位 MCU 可能没有此功能），大多数的 MCU 的 Watchdog 只能允许程序对其进行复位而不能对其关闭（有的是在程序烧入时来设定的，如 Microchip PIC 系列 MCU），而有的 MCU 则是通过特定的方式来决定其是否打开，如 Samsung 的 KS57 系列，只要程序访问了 Watchdog 寄存器，就自动开启且不能再被关闭。一般而言 watchdog 的复位时间是可以程序来设定的。Watchdog 的最基本的应用是为 MCU 因为意外的故障而导致死机提供了一种自我恢复的能力。

MCU 程序的编写：

MCU 的程序的编写与 PC 下的程序的编写存在很大的区别，虽然现在基于 C 的 MCU 开发工具越来越流行，但对于一个高效的程序代码和喜欢使用汇编的设计者来讲，汇编语言仍然是最简洁、最有效的编程语言。对于 MCU 的程序编写，其基本的框架可以说是大体一致的，一般分为初始化部分（这是 MCU 程序设计与 PC 最大的不同），主程序循环体和中断处理程序三大部分（见图 1 a 和 b），其分别说明如下：

* 初始化：对于所有的 MCU 程序的设计来讲，出世化是最基本也是最重要的一步，一般包括如

下内容：

** 屏蔽所有中断并初始化堆栈指针：初始化部分一般不希望有任何中断发生；

** 清除系统的 RAM 区域和显示 Memory：虽然有时可能没有完全的必要，但从可靠性及一致性的角度出发，特别是对于防止意外的错误，还是建议养成良好的编程习惯；

** IO 口的初始化：根据项目的应用的要求，设定相关 IO 口的输入输出方式，对与输入口，需要设定其上拉或下拉电阻；对于输出口，则必须设定其出世的电平输出，以防出现不必要的错误；

** 中断的设置：对于所有项目需要用到的中断源，应该给予开启并设定中断的触发条件，而对于不使用的多余的中断，则必须给予关闭；

** 其他功能模块的初始化：对于所有需要用到的 MCU 的外围功能模块，必须按项目的应用的要求进行相应的设置，如 UART 的通讯，需要设定 Baud Rate，数据长度，校验方式和 Stop Bit 的长度等，而对于 Programmer Timer，则必须设置其时钟源，分频数及 Reload Data 等；

** 参数的出世化：完成了 MCU 的硬件和资源的出世化后，接下来就是对程序中使用到的一些变量和数据的初始化设置，这一部分的初始化需要根据具体的项目及程序的总体安排来设计。对于一些用 EEPROM 来保存项目预制数的应用来讲，建议在初始化时将相关的数据拷贝到 MCU 的 RAM，以提高程序对数据的访问速度，同时降低系统的功耗（原则上，访问外部 EEPROM 都会增加电源的功耗）。

* 主程序循环体：大多数 MCU 是属于长时间不间断运行的，因此其主程序体基本上都是以循环的方式来设计，对于存在多种工作模式的应用来讲，则可能存在多个循环体，相互之间通过状态标志来进行转换。对于主程序体，一般情况下主要安排如下的模块：

** 计算程序：计算程序一般比较耗时，因此坚决反对放在任何中断中处理，特别是乘除法运算；

** 实时性要求不高或没有实时性要求的处理程序；

** 显示传输程序：主要针对存在外部 LED、LCD Driver 的应用；

* 中断处理程序：中断程序主要用于处理实时性要求较高的任务和事件，如，外部突发性信号的检测，按键的检测和处理，定时计数，LED 显示扫描等。一般情况下，中断程序应尽可能保证代码的简洁和短小，对于不需要实时去处理的功能，可以在中断中设置触发的标志，然后由主程序来执行具体的事务－－这一点非常重要，特别是对于低功耗、低速的 MCU 来讲，必须保证所有中断的及时响应。

* 对于不同任务体的安排，不同的 MCU 其处理的方法也有所不同。例如，对于低速、低功耗的 MCU（Fosc＝32768Hz）应用，考虑到此类项目均为手持式设备和采用普通的 LCD 显示，对按键的反应和显示的反应要求实时性较高，应此一般采用定时中断的方式来处理按键的动作和数据的显示；而对于高速的 MCU，如 Fosc>1MHz 的应用，由于此时 MCU 有足够的时间来执行主程序循环体，因此可以只在相应的中断中设置各种触发标志，并将所有的任务放在主程序体中来执行；

* 在 MCU 的程序设计中，还需要特别注意的一点就是：要防止在中断和主程序体中同时访问或设置同一个变量或数据的情况。有效的预防方法是，将此类数据的处理安排在一个模块中，通过判断触发标志来决定是否执行该数据的相关操作；而在其他的程序体中（主要是中断），对需要进行该数据的处理的地方只设置触发的标志。－－这可以保证数据的执行是可预知和唯一的。

总之，对于 MCU 开发来讲，必须记住一点：“条条大路通罗马”，没有做不到的事，关键是看方法是否正确！再就是多做多动手和多想。