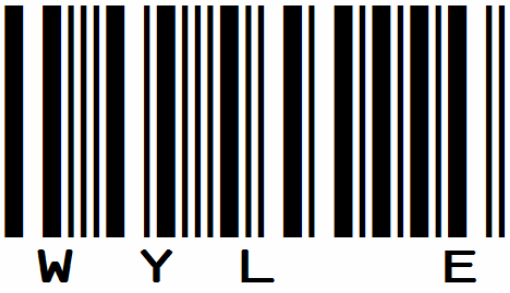
C51 精确延时函数的编写及调试方法

Author : wyl_e

QQ : 404530302

E-mail : wyl-e@163.com

Blog : http://blog.ednchina.com/wyl_e/



前 言

本文详细分析了函数及循环语句的执行过程，介绍了用 C51 编写和调试精确延时程序的方法。

在单片机程序设计过程中，实现精确的时间控制通常可以采用定时或延时的方法。传统的定时器定时的方法可实现 1 个机器周期的精确的时间控制；神圣的汇编语言同样可以实现 1 个机器周期的精确延时；而 C51，不能直观的计算出执行周期，精确延时似乎难以实现。以至于，执着于汇编程序设计的前辈们经常将 C51 的这些缺点否定得一无是处（曾在某公司，唯我使用 C51。当有一次牵涉到时间控制的时候，我的领导：“C 语言延时是不精确的，我们某产品上的 DS18B20，程序是用汇编写的，用 C 就没法控制”——荒唐）；而使用 C51 的新手也因这些问题而烦恼（论坛、QQ 群，咨询 C51 延时的相关问题的真是太多了）。当 C51 新手提到此类问题的时候，习惯了计算汇编代码的执行周期的高手当然会毫不吝啬的将自己的方法告诉他们。这方法似乎太麻烦了吧？那么，采用什么样的方法可以方便的计算出 C51 代码的执行周期而实现精确定时呢？请看下文……

时间匆促，疏漏难免，敬请指正，3KS！

一、C 函数的执行周期的计算方法

1、预备知识：

要编写一个精确延时函数，我们首先要了解一般延时函数的执行周期的计算方法。

要点：函数入口、返回过程；循环语句的循环过程。

函数调用过程与图2.1所示：

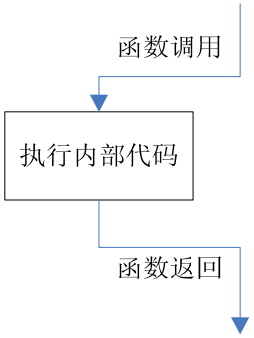


图2.1

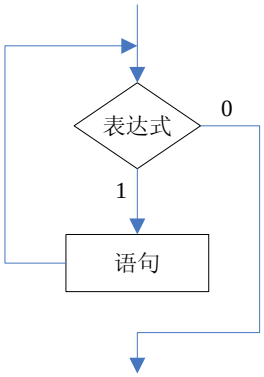


图2.2

可见，函数调用包括了入口（调用）、内部代码执行、返回三个过程。那么一个被调用函数的的执行周期就是该函数的入口周期、内部代码的执行周期、函数返回的周期之和。

例如：某一函数F，入口周期为T_E，返回周期为T_R，内部代码执行周期为T_I，该函数的执行周期：$T_F = T_E + T_R + T_I$。

通常我们编写的延时函数是一个有限带参循环。那么，要求出该函数的执行周期，除了要知道入口周期、返回周期，我们还要知道内部代码的执行周期，即带参循环的执行周期，才可以求出该函数在带实参调用后的执行周期了。

我们拟在延时函数中使用while()语句来实现循环，while()语句的执行过程如图2.2所示。它包括条件判断、语句执行两个过程。请注意，while语句中，条件判断在整个循环过程中的执行次数，要比内部语句多一次。

假如有：while(n--){_nop_();}；n=10；那么在该while(){}执行过程中，对n的判断次数为11次，语句执行次数为10次。因为当执行完第10次循环后，n=0，但

while语句还得进行一次表达式的判断才会跳出循环。所以，当我们编写以下函数：

```
Void Func(unsigned int n)
    {
        while(n--){……};
    }
```

其入口周期为T_E，返回周期为T_R，while的条件判断周期为T_J，while的语句执行周期为T_P。如果n被赋值为10，那么，该函数的执行周期：$T_F = T_E + T_R + [10*(T_J + T_P) + T_J]$——公式1。

在利用公式1求解函数的执行周期前，我们要得到函数每一个执行过程的执行周期。要得到每个执行过程的执行周期，我们不采用查看汇编代码的方法，因为那样确实太麻烦了。在Keil 软件中，通过Simulator的Debug功能，可以让我们非常直观方便的得到每个过程甚至每条C语句的执行周期。我们可以Debug出每条语句、每个过程的执行时间，然后累加，就是整个函数的执行时间了。但这似乎还是有些麻烦，我们继续简化Debug这个过程。

通常，我们将上面的函数的执行周期分成调用周期与循环周期两部分。调用周期T_{Call}是包括 入口周期T_C+返回周期T_R+ while的条件为假时的最后一次判断的周期T_J，且其为常数；循环周期T_{Cycle}是整个while的n次条件判断周期T_J与执行周期为T_P的和。

这样，我们可将一个带参的循环函数的执行时间看成$T=T_{Cycle}x+T_{Call}$的形式。呵呵，这下好了，要求出T_{Cycle}与T_{Call}，就成了求解线性方程了。下一章，我们就借助Keil 软件中的Simulator，将这个线性方程的解Debug出来。

二、C51 精确延时函数的调试方法

1、编写精确延时函数

参考程序如附录中所示，该段代码已通过调试，每行代码注释的执行周期都是在 Keil 调试结果。

请在新建的工程中，将该参考代码复制到 C 文件中，并包含以下头文件：

```
#include<INTRINS.H>              //for _nop_()
```

新建 main 函数：

```
void main(void)
    {
        _nop_();          //DEBUG RESULT                    //COST
        Delay_20T(0);     //390
        Delay_20T(1);     //411     Delay_20T(0) cost   411 - 390     = 21T
        Delay_20T(2);     //451     Delay_20T(1) cost   451 - 411     = 40T     (20*1+20)
        Delay_E3T(0);     //511     Delay_20T(2) cost   511 - 451     = 60T     (20*2+20)
        Delay_E3T(1);     //532     Delay_E3T(0) cost   532 - 511     = 21T
        Delay_E3T(2);     //1552    Delay_E3T(1) cost   1552- 532     = 1020T (1000*1+20)
        _nop_();          //3572    Delay_E3T(0) cost   3572- 1552    = 2020T (1000*2+20)
        while(1);
    }
```

2、设置 simulator

我们的目标是要获取函数的执行周期而不是执行时间，因为计算周期更加方便以后的代码移植。而 Keil 中反映的是执行时间，我们在 Debug 时将晶振设置为特殊值，如 12MHz，那么 1us 也就是一个机器周期了，然后根据 Keil 中的执行时间，轻松的得出它的执行周期。

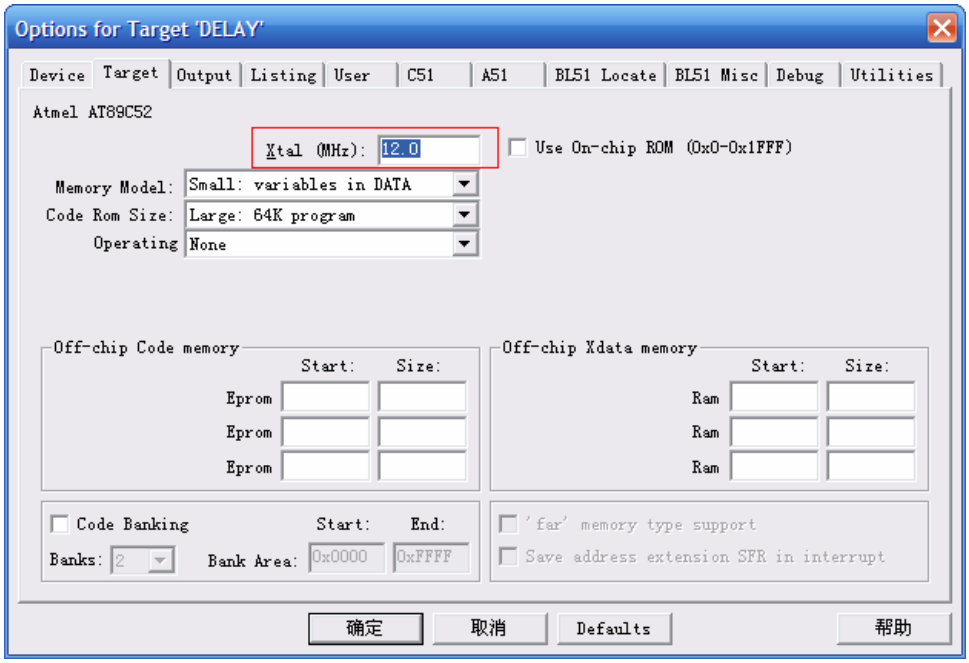


图 3.1

3、编译，并开始 Debug

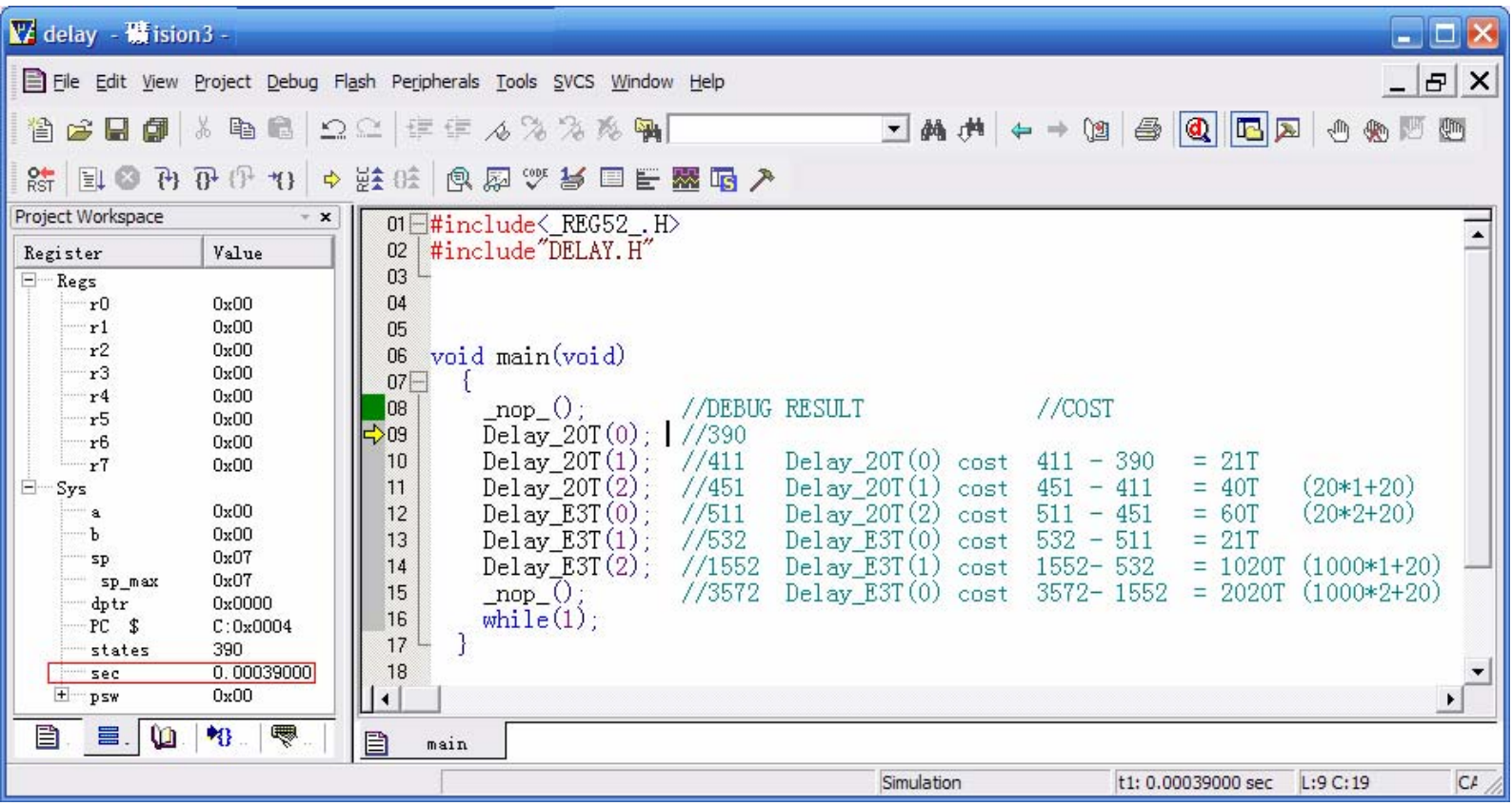


图 3.2

如图 3.2 所示，使用右键中的 Run to cursor line 命令，可以执行到鼠标指定的某一行，同时 sec 栏显示执行到此处所花的时间（下文中我们用周期表示，因为在 12MHz 晶振下，1T=1us），单位为秒。

当得到了 CPU 执行到每一语句所花周期时，通过计算就可以知道某一语句的执行周期了。图 3.2 中已经列出了执行到某一句语句所花的执行周期，以及计算所得的该语句的执行周期。

例如，执行到 Delay_20T(1)时，花了 411 个时钟周期；

执行到 Delay_20T(2)时，花了 451 个时钟周期；

执行到 Delay_E3T(0)时，花了 511 个时钟周期；

这样，我们就知道，Delay_20T(1)的执行周期为 451-411=40T；

Delay_20T(2)的执行周期为 511-451=60T；

据上一章$T=T_{Cycle}x+T_{Call}$，可以得到：

$40 = T_{Cycle}*1+T_{Call}$ <方程一>

$60 = T_{Cycle}*2+T_{Call}$ <方程二>

$=> T_{Cycle}=20T$； $T_{Call}=20T$

也就是说：Delay_20T（）这个函数的调用周期为 20T；每一循环周期为 20T；如有 Delay_20T(n)的调用 ，则它的执行周期为 20* (n+1) T。

同样的方法，我们可以获取不同类似函数的循环周期与调用周期，而求出它的执行周期。

Keil 的 Simulator 功能实在强大，你可以借助它方便的将每个函数的执行周期及调用过程的每一步的执行周期 Debug 出来。

时间匆促，此处不再赘述。

本文另附 Delay 调试工程，包括附录的代码在内，详细注释了每一语句及函数的执行周期。仅供参考。

例中仅包含了 20 和 1000 个时钟周期的延时函数。你可以参照例子，编写出周期更长或其它量级的延时函数。这些延时函数的最小延时为 21T，如需要更短的延时，请用多条_nop_()语句吧。

附一、精确延时参考程序代码：

```
/*****************************************************
* Function    : delay for 20Ts
*        Note : cost 20*cnt+20 (T), but Delay_20T(0),cost 21 T
*****************************************************/
void Delay_20T(unsigned int cnt)
    {                           //--------------Call,if(cnt==0)cost 5T else 4T
        while(cnt--)            //---------------while{},if(cnt==0)cost 10T else 11T
            {
                _nop_();  _nop_();
                _nop_();  _nop_();
                _nop_();  _nop_();
                _nop_();  _nop_();
                _nop_();     //---------------9 nops,cost 9 T
            };                //---------------all,cost cnt * 20 T
    _nop_();_nop_();
    _nop_();_nop_();    //-----------4 nops,cost 4 T
    }                   //-----------return,cost 2 T
/*****************************************************
* Function : delay for E3Ts
*       Note : cost 1000*cnt+20 (T), but Delay_E3T(0),cost 21 T
*****************************************************/
void Delay_E3T(unsigned int cnt)
    {                           //---------------Call,if(cnt==0)cost 9T else 8T
        while(cnt--)            //---------------while{},if(cnt==0)cost 10T else 11T
            {
                Delay_20T(48);//---------cost (48+1)*20T=980 T
                _nop_();  _nop_();
                _nop_();  _nop_();
                _nop_();  _nop_();
                _nop_();  _nop_();
                _nop_();     //---------------9 nops,cost 9 T
            }           //-----------------------all,cost cnt * 1000 T
    }                   //-----------------------return,cost 2 T
```