使用DJGPP编写DOS程序

freevanx

作者简介

作者freevanx,您可以通过Mail： freevanx@gmail.com 或者MSN：freevanx@hotmail.com联系。

2009/6/12	初稿 V1.0

参考文档

DJGPP: http://www.delorie.com/djgpp/
GCC: http://gcc.gnu.org/
Glibc: http://www.gnu.org/software/libc/
DOS Extender: http://en.wikipedia.org/wiki/DOS_extender
GNU Make: http://www.gnu.org/software/make/

前言

写这篇文章之前，看到网络上很多同行都寻找编写DOS程序的工具，借此机会向大家介绍这样一款既免费无责任，又非常好用的工具。由于BIOS开发的特殊性，DOS作为古老的OS之一，其非常简单的特点很适合一些测试工具的运行，大部分情况下，可以简化测试流程，缩短测试时间，所以在量产测试中，DOS下测试是一个非常重要且简单的测试过程。当然在EFI BIOS下，可以将测试程序写成EFI Shell APP。

现在开发传统DOS下的程序，常用的是Borland的TC或者BCC，或者是MS的VC1.52，TC只能写16位程序，如果要用到32位数据的话，就非常麻烦，要用不但要用嵌汇编，还要用机器码来实现32位的操作。其他的软件，即使可以使用32位的数据，当面临下面一种情况时，同样也需要很麻烦操作：

由于DOS系统运行在real mode下，所以一般的程序不能访问大于1M的地址，但是在开发一些程序的时候需要访问1M以上直至4GB的地址，这个时候怎么办？传统的DOS程序会另外使用一个叫做DOS Extender的程序，用来提供protect mode的服务，其中最有名的当属DOS4GW这个程序，但是这个软件是商业软件，开发早停，收费不止，而且这个程序笨重难用，开源爱好者们开发了一款open source的软件来代替他，名叫DOS/32 Advanced DOS Extender，借助DOS Extender，DOS下的程序可以访问高达4GB的地址空间，解决了不能访问高地址的问题，不过解决的同时带来了另一个问题，就是DOS Extender提供的是一系列library形式的function，程序中需要调用这些function才能达到目的。

今天要介绍的DJGPP，同时解决了以上两个问题，首先DJGPP中可以使用32位数据，所以不用自己写机器码，其次DJGPP编译的DOS程序，默认运行在protect mode下，所以可以访问4GB的地址空间，当然DJGPP也需要一个DOS Extender，不过DJGPP不要你额外写代码，因为编译器把Extender绑定在一起，这样你只需要关注你自己的代码就可以了，当然这样也有一个坏处，就是不管你写什么样的程序，你总是需要这样一个Extender，也就是说，你分发的程序至少有两个文件，其中一个是Extender的可执行文件。

介绍

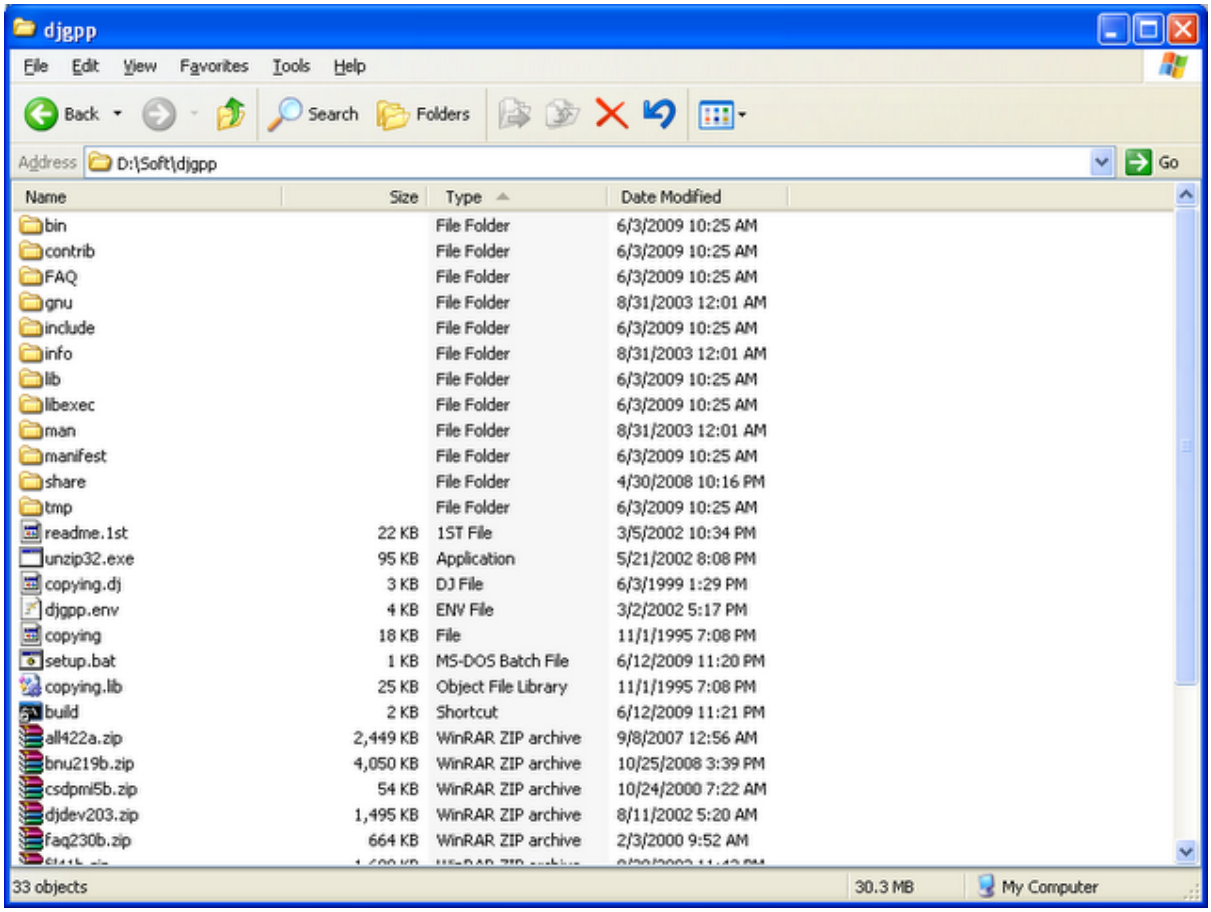
DJGPP是基于GCC的一个DOS porting，所以接受GCC语法，GCC设定，如果你熟悉Linux GCC的使用或者使用过Windows下的Mingw，那么你可以很容易的使用DJGPP来编译你的程序，即使你以前对GCC一无所知，你也可以相信，follow freevanx的介绍，你可以轻易的学会如何使用DJGPP并提高你的开发效率。

关于DOS Extender的携带。所有使用DJGPP编译的DOS程序，都需要配合支持DPMI接口的DOS Extender使用，DJGPP默认编译过的程序使用CSDPMI这个程序，所以你在DOS下使用的时候需要同时携带cwsdpmi.exe使用。你可以从DJGPP站点中选择一个ftp下载csdpmi的软件包，放在ftp v2misc目录下的csdpmi5b.zip。也可以直接从这里下载http://djgpp.linux-mirror.org/v2misc/ 。除了CSDPMI之外，还有另外一个DPMI Service程序可供选择，即pmode，如果想是程序运行更加高效，可以使用pmode，使用方法请参考pmode13b.zip中的文档。

关于Ring0。在与BIOS有关的DOS程序中，很多软件都需要运行在Ring0下，也就是说可能很多软件都或多或少需要依赖于特权指令，DJGPP的DOS Extender csdpmi中，如果使用默认的CWSDPMI.exe的话，默认是运行在protect mode的user mode，即Ring3，如果你需要运行在Ring0下，请使用csdpmi包中的CWSDPR0.EXE，将这个软件重命名成CWSDPMI.EXE随你的软件分发就可以了。

安装

DJGPP的主站：http://www.delorie.com/djgpp/
使用DJGPP提供的**Zip Picker** 选择你要下载的包,下载后请选择解压到当然文件夹，然后就可以看到像下面这样的内容。（见下图 ）

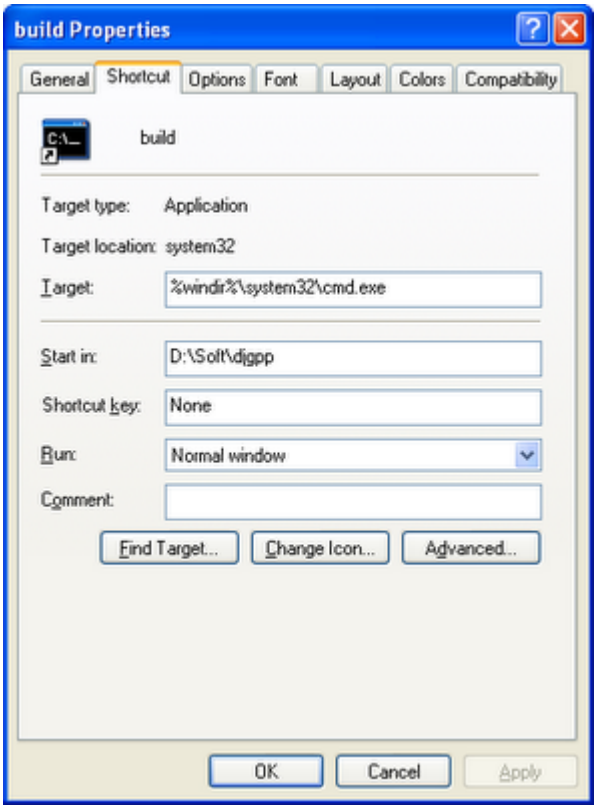


注意： **Zip Picker**选择给出的软件并不包含**CSDPMI**，所以要单独下载这个包

设置环境。DJGPP需要一些简单的设置，需要将编译器的路径添加到PATH里面，并且建立一个名叫DJGPP的环境变量指向djgpp.env文件。以下是我使用的一个batch file的内容

```
@REM ================================================
@REM    file setup.bat
@REM   seting build environment for DJGPP
@REM   create by freevanx
@REM ================================================
@set ROOT=%CD%
@set PATH=%PATH%;%ROOT%\bin;
@set DJGPP=%ROOT%\djgpp.env
@echo "DJGPP build environment is ready!"
```

另外，为了方便，我还在DJGPP目录下建立一个快捷方式，（如下图 ）



然后，我们可以测试一下我们的环境是否OK，双击刚才建立的快捷方式，然后输入下面命令：

```
D:\Soft\djgpp>setup
"DJGPP build environment is ready!"

D:\Soft\djgpp>cd bin

D:\Soft\djgpp\bin>gcc
gcc.exe: no input files

D:\Soft\djgpp\bin>gcc --version
gcc.exe (GCC) 4.3.2
Copyright (C) 2008 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.


D:\Soft\djgpp\bin>
```

说明环境建立完毕。可以进行编译了。。

第一个程序： **Hello World**

DJGPP是GCC的一个porting，所以编译器是gcc，不必对gcc感到畏惧，使用你常用的C语法就可以了，建立一个C文件输入以下内容

```
#include <stdio.h>

int main(void)
{
  printf("Hello, world!\n");
```

```
  return 0;
}
```

编译链接：

```
D:\Soft\djgpp\bin>gcc hello.c -o hello.exe
```

在DOS下运行：

```
C:\ > hello
Hello, World!
```

当然，如果程序比较大，有很多个源文件，可以使用makefile, DJGPP中包含的make是属于GNU Make，功能非常强大，你可以写一个非常简单的makefile完成很复杂的任务。

进阶

在稍微熟悉了gcc以后，我们继续下面的程序，下面将提供一些在BIOS开发中常用的函数，例如IO Memory==来辅助大家开发。

必要的头文件：以下是一个头文件示例，要包含哪些头文件，取决于你要写的程序

```
/*++
************************************************
*            Copyright (c) 2008~2009 freevanx.  All rights reserved.
*
*                  freevanx@gmail.com
*
*   This file is distributed under BSD liscense
*   File:
*   Description:    remember to use cwsdpr0.exe instead of cwsdpmi.exe
*
--***********************************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys/nearptr.h>
#include <unistd.h>

#define    QIO_PCI_ADDR_PORT    0xCF8
#define    QIO_PCI_DATA_PORT    0xCFC

typedef    unsigned char    UCHAR, UINT8;
typedef    unsigned short    USHORT, UINT16;
typedef    unsigned long    ULONG, UINT32;
typedef    unsigned int    UINT;
```

访问IO 端口函数的封装

```
#define __BUILDIO(bwl, bw, type)              \
static inline void out##bwl(unsigned type value, int port)    \
{                          \
   out##bwl##_local(value, port);             \
}                          \
                          \
static inline unsigned type in##bwl(int port)          \
{                          \
   return in##bwl##_local(port);              \
}

#define BUILDIO(bwl, bw, type)                    \
static inline void out##bwl##_local(unsigned type value, int port)    \
```

```
{                               \
   asm volatile("out" #bwl " %" #bw "0, %w1"       \
         : : "a"(value), "Nd"(port));          \
}                               \
                               \
static inline unsigned type in##bwl##_local(int port)          \
{                               \
   unsigned type value;                    \
   asm volatile("in" #bwl " %w1, %" #bw "0"       \
         : "=a"(value) : "Nd"(port));          \
   return value;                           \
}                               \
                               \
                               \
__BUILDIO(bwl, bw, type)                    \
                               \
static inline void outs##bwl(int port, const void *addr, unsigned long count) \
{                               \
   asm volatile("rep; outs" #bwl                 \
         : "+S"(addr), "+c"(count) : "d"(port));       \
}                               \
                               \
static inline void ins##bwl(int port, void *addr, unsigned long count)    \
{                               \
   asm volatile("rep; ins" #bwl                  \
         : "+D"(addr), "+c"(count) : "d"(port));       \
}

BUILDIO(b, b, char)
BUILDIO(w, w, short)
BUILDIO(l, , int)
```

以上一些宏定义提供下面几个函数：

读IO Port

```
static inline unsigned char inb(int port);
static inline unsigned short inw(int port);
static inline unsigned int inl(int port);
```

写IO Port

```
static inline void outb(unsigned char value, int port);
static inline void outw(unsigned short value, int port);
static inline void outl(unsigned long int, int port);
```

访问Memory函数的封装：

```
static inline unsigned char readb( unsigned long addr)
{
   unsigned char    ret=0;
    if (__djgpp_nearptr_enable())
    {
      unsigned char *phy_addr = (unsigned char *)
                                 (__djgpp_conventional_base + addr);
       ret = *phy_addr;
      __djgpp_nearptr_disable();
      return ret;
    }
    return 0xFF;
}
```

```
static inline unsigned short readw( unsigned long addr)
{
   unsigned short    ret=0;
    if (__djgpp_nearptr_enable())
    {
      unsigned short *phy_addr = (unsigned short *)
                                 (__djgpp_conventional_base + addr);
       ret = *phy_addr;
      __djgpp_nearptr_disable();
      return ret;
    }
    return 0xFF;
}

static inline unsigned long readl( unsigned long addr)
{
   unsigned long    ret=0;
    if (__djgpp_nearptr_enable())
    {
      unsigned long *phy_addr = (unsigned long *)
                                 (__djgpp_conventional_base + addr);
       ret = *phy_addr;
      __djgpp_nearptr_disable();
      return ret;
    }
    return 0xFF;
}

inline void writeb(unsigned char value, unsigned long addr)
{
    if (__djgpp_nearptr_enable())
    {
      unsigned char *phy_addr = (unsigned char *)
                                 (__djgpp_conventional_base + addr);
       *phy_addr = value;
      __djgpp_nearptr_disable();
    }
}

inline void writew(unsigned short value, unsigned long addr)
{
    if (__djgpp_nearptr_enable())
    {
      unsigned short *phy_addr = (unsigned short *)
                                 (__djgpp_conventional_base + addr);
       *phy_addr = value;
      __djgpp_nearptr_disable();
    }
}

inline void writel(unsigned long value, unsigned long addr)
{
    if (__djgpp_nearptr_enable())
    {
      unsigned long *phy_addr = (unsigned long *)
                                 (__djgpp_conventional_base + addr);
       *phy_addr = value;
```

```
        __djgpp_nearptr_disable();
      }
    }
```

以上的函数提供下列函数的封装

读Memory函数

```
static inline unsigned char readb( unsigned long addr) ;
static inline unsigned short readw( unsigned long addr) ;
static inline unsigned long readl( unsigned long addr) ;
```

写Memory函数

```
inline void writeb(unsigned char value, unsigned long addr) ;
inline void writew(unsigned short value, unsigned long addr) ;
inline void writel(unsigned long value, unsigned long addr) ;
```

访问PCI设备函数的封装

```
UINT pci_bus_read_config_dword(UCHAR bus, UCHAR devfn, UCHAR offset,
ULONG * pvalue)
{
   ULONG    addr = 0x80000000;

   if (pvalue == NULL)
      return 1;       // Fix Here
   offset = offset/4 * 4;
   addr = addr + (bus << 16) + (devfn << 8) + offset;
   outl(addr, QIO_PCI_ADDR_PORT);
   *pvalue = inl(QIO_PCI_DATA_PORT);
   return 0;
}

UINT pci_bus_read_config_byte(UCHAR bus, UCHAR devfn, UCHAR offset, UCHAR
* pvalue)
{
   ULONG    value =0;
   UINT ret=0;
   UCHAR    longoffset= offset/4 * 4;
   UCHAR    index = offset - longoffset;

   if (pvalue == NULL)
      return 1;       // Fix Here

   ret = pci_bus_read_config_dword(bus, devfn, longoffset, &value);
   *pvalue = (UCHAR) ((value >> (index*8)) & 0xFF);
   return 0;
}
UINT pci_bus_read_config_word(UCHAR bus, UCHAR devfn, UCHAR offset,
USHORT * pvalue)
{
   ULONG    value =0;
   UINT ret=0;
   UCHAR    longoffset= offset/4 * 4;
   UCHAR    index = offset - longoffset;

   if (pvalue == NULL)
      return 1;       // Fix Here
```

```
   ret = pci_bus_read_config_dword(bus, devfn, longoffset, &value);
   *pvalue = (USHORT) ((value >> (index*8)) & 0xFFFF);
   if (index >= 3)
   {
      ret = pci_bus_read_config_dword(bus, devfn, longoffset+4, &value);
      *pvalue += (USHORT) ((value  & 0xFF) <<8);
   }

   return 0;
}

UINT pci_bus_write_config_dword(UCHAR bus, UCHAR devfn, UCHAR offset,
ULONG  value)
{
   ULONG    addr = 0x80000000;

   offset = offset/4 * 4;
   addr = addr + (bus << 16) + (devfn << 8) + offset;
   outl(addr, QIO_PCI_ADDR_PORT);
   outl(value, QIO_PCI_DATA_PORT);
   return 0;
}

UINT pci_bus_write_config_byte(UCHAR bus, UCHAR devfn, UCHAR offset, UCHAR
value)
{
   ULONG    temp_value =0;
   UINT ret=0;
   UCHAR    longoffset= offset/4 * 4;
   UCHAR    index = offset - longoffset;
   UCHAR* pvalue = (UCHAR*)&temp_value;

   ret = pci_bus_read_config_dword(bus, devfn, longoffset, &temp_value);
   *(pvalue + index ) = value;
   ret = pci_bus_write_config_dword(bus, devfn, longoffset, temp_value);

   return 0;
}
UINT pci_bus_write_config_word(UCHAR bus, UCHAR devfn, UCHAR offset,
USHORT  value)
{
   ULONG    temp_value =0;
   UINT ret=0;
   UCHAR    longoffset= offset/4 * 4;
   UCHAR    index = offset - longoffset;
   UCHAR* pvalue = (UCHAR*)&temp_value;
   if (index <3)
   {
      ret = pci_bus_read_config_dword(bus, devfn, longoffset, &temp_value);
      *(USHORT *)(pvalue + index ) = value;
      ret = pci_bus_write_config_dword(bus, devfn, longoffset, temp_value);
   }
   else
   {
      ret = pci_bus_read_config_dword(bus, devfn, longoffset, &temp_value);
      *(pvalue + index ) =(UCHAR) (value & 0xFF);
      ret = pci_bus_write_config_dword(bus, devfn, longoffset, temp_value);
      ret = pci_bus_read_config_dword(bus, devfn, longoffset+4, &temp_value);
```

```
            *pvalue=(UCHAR) (value >> 8);
            ret = pci_bus_write_config_dword(bus, devfn, longoffset+4, temp_value);
        }

        return 0;
    }
```

以上函数提供以下PCI函数的封装

读PCI设备配置空间：

```
    UINT pci_bus_read_config_dword(UCHAR bus, UCHAR devfn, UCHAR offset,
    ULONG * pvalue);
    UINT pci_bus_read_config_byte(UCHAR bus, UCHAR devfn, UCHAR offset, UCHAR
    * pvalue);
    UINT pci_bus_read_config_word(UCHAR bus, UCHAR devfn, UCHAR offset,
    USHORT * pvalue);
```

写PCI设备的配置空间：

```
    UINT pci_bus_write_config_dword(UCHAR bus, UCHAR devfn, UCHAR offset,
    ULONG  value);
    UINT pci_bus_write_config_byte(UCHAR bus, UCHAR devfn, UCHAR offset, UCHAR
    value);
    UINT pci_bus_write_config_word(UCHAR bus, UCHAR devfn, UCHAR offset,
    USHORT  value);
```

注意：**dword**的读写函数都只接受与**4**字节对齐**offset**值，与**PCI**设备配置空间的访问方式相对应，参见**PCI specification**

接下来，是一组访问MSR的函数，这组函数封装了读写CPU MSR寄存器的操作，使用时只需要调用这些函数即可

```
    static inline void readmsr(unsigned long ecx, unsigned long* eax, unsigned long*
    edx)
    {
        asm volatile("rdmsr"
                : "=a"(*eax), "=d"(*edx)
                : "c"(ecx) );
    }

    static inline void writemsr(unsigned long ecx, unsigned long eax, unsigned long
    edx)
    {
        asm volatile("wrmsr"
                :
                : "c"(ecx), "a"(eax), "d"(edx));
    }
```

以上两个函数提供下列的API

读取CPU MSR的值

```
    static inline void readmsr(unsigned long ecx, unsigned long* eax, unsigned long*
    edx)；
```

写CPU MSR的值

```
    static inline void writemsr(unsigned long ecx, unsigned long eax, unsigned long
    edx) ；
```

好了，以上提供了4组函数，通过这4组函数，你可以读写IO Port，读写Memory/MMIO, 读写PCI 配置空间，读写MSR，那么有了这几组函数，还有多少你做不到的事情呢？Let's go！

应用

接下来我们将使用上面提供的PCI函数和IO函数，来写一个DOS下的延时程序，这个延时程序使用32位的计时器，可以精确到毫秒级别，最长可以延时19.99分钟。

这里使用的计时器是ACPI PM1 Timer,这是一个24位的计时器，使用一个3.579545MHz的时钟，刚才我们说过我们的时钟是32bit的，如何使用24位时钟实现32位时钟呢？看了以下代码相比就明白了吧：

```
ULONG    current_ticket=0;
ULONG    previous_ticket = 0;
ULONG    count=0;

current_ticket = previous_ticket = inl(pmbase +0x08);      // PM1 timer register

while(count < delay_ms)
{
   current_ticket = inl(pmbase +0x08);                     // PM1 timer register
   if (current_ticket >= previous_ticket)
   {
      count += (current_ticket - previous_ticket);
   }
   else
   {
      count += current_ticket;
   }
   previous_ticket = current_ticket;
}
```

count 是计数器的累计，是一个32bit的值，用来表示一个32bit的计数器，current_ticket是当前读到PM1 Timer的数值，previous_ticket是上次读到PM1 Timer的数值，如果当前的数值比上次独到的数值大，说明PM1 Timer在增长并且没有溢出24bit这个范围，我们只需要在count上加上(current_ticket - previous_ticket)这样一个经过时间的差量就可以了，当前读到的数值较小，说明PM1 Timer计数已经溢出24bit并且重新开始，那么我们要给count加上当前读到的数值。

在这之前，还要解决一个问题，PM1 Timer的IO Port是在PMBASE的基础上加8（参见南桥的spec），而PMBASE是一个不定值，是在BIOS POST的时候设置的，常见的值是PMBASE=0x800或者PMBASE=0x400，那如何获取当前系统PMBASE的设定呢？我们需要使用上面提供的PCI 函数读取LPC Bridge上PMBASE的设置，参见以下代码：

```
inline static USHORT GetPMBase(void)
{
   UINT    ret = 0;
   USHORT    base =0;
   ret = pci_bus_read_config_word(0, 0xF8, 0x40, &base);
   base &= 0xFF80;
   return base;
}
```

好，大部分问题都解决了，那么我现在贴出源程序的代码，加上上面的头文件和函数定义，就可以完成我们需要的功能了。

```
inline static USHORT GetPMBase(void)
{
```

```
    UINT    ret = 0;
    USHORT    base =0;
    ret = pci_bus_read_config_word(0, 0xF8, 0x40, &base);
    base &= 0xFF80;
    return base;
}


char  gMsg[] =  "lychee Project (c) freevanx All Rights Reserved!\n" \
            "QuikIO hardware access library!\n" \
           "ms delay utility for DOS, max delay input value is 1199879 = 19.99 min\n" \
            "    Build by DJGPP";

int verbose =0;


inline static UINT    mdelay(USHORT pmbase, ULONG ms)
{
    ULONG    current_ticket=0;
    ULONG    previous_ticket = 0;
    ULONG    count=0;

    // PM1 timer add 7159.09 every 2ms
    ULONG    delay_ms = (ms/2 *7159) + ((ms %2)? 7159/2 : 0 );
    static    int    start_flag =0;

    current_ticket = previous_ticket = inl(pmbase +0x08);  // PM1 timer register

    if (verbose)
    {
        printf("Delay start at ticket %u\n", current_ticket);
    }


    while(count < delay_ms)
    {
        current_ticket = inl(pmbase +0x08);         // PM1 timer register
        if (current_ticket >= previous_ticket)
        {
            count += (current_ticket - previous_ticket);
        }
        else
        {
            count += current_ticket;
        }
        previous_ticket = current_ticket;
    }

    if (verbose)
    {
        printf("Delay end at ticket %u\n", current_ticket);
        printf("Delay should end at count %u\n", delay_ms);
        printf("Delay end at count %u\n", count);
    }

    return 0;
}
```

```
int main(int argc, char** argv)
{
   int c =0;
   ULONG    time=0;
   USHORT    pmbase =0;
   int go_flag=0;

   while ((c = getopt(argc, argv, "t:hv")) != -1 )
   {
      switch (c)
      {
      case    't'    :
         time = strtoul(optarg, NULL, 0);
         go_flag=1;
         break;

      case 'v'    :
         verbose =1;
         break;

      case 'h'    :
      case '?'    :
         printf(gMsg);
         break;

      default:
         printf(gMsg);
      }
   }

   pmbase = GetPMBase();
   if (verbose)
   {
      printf("\n Delay %u ms\n", time);
      printf(" Found PMBASE at %04x\n", pmbase);
   }

   if (go_flag)
   {
      mdelay(pmbase, time);
   }
   return 0;
}
```

再解释一下，其中用到的getopt函数，是glibc特有的函数，MSVC的libc库并没有这个函数，详细的用法可以查看glibc的帮助。

事实上，这个延时Timer的程序经过了HW示波器的检验，在1ms的精度上可以看到非常的精确，用肉眼无法发现误差。

本文中用到的所有代码都可以通过：http://docs.google.com/View?id=d97vr8z_53c7k6bdfs 观看。

结束

OK，到此为止，我们整个学习过程就结束了，你可以使用给出的几组函数，任意达成你的目标，希望学习完这个简单的教程之后，会帮助你提高开发DOS程序的效率。 Enjoy the programming!

这些代码是我本人开发中的lychee project的一部分，现在将其中DOS平台的部分代码以BSD License发布，希望对大家的开发有所帮助。