

# I2C 产品二次开发函数与命令 说明手册

手册版本：V1.0

更新时间：2008-08-01

# 目 录

目 录 .....	2
一、 Easy I2C 与 Timing I2C .....	3
1.1 Esay I2C 模式: .....	3
1.2 Timing I2C 模式: .....	3
二、 VCI_GYI2C 库文件接口函数 .....	4
2.1 设备类型定义.....	4
2.2 数据结构体定义.....	4
2.3 接口函数定义与描述.....	4
2.3.1 设备打开.....	5
2.3.2 设备关闭.....	5
2.3.3 设置工作模式.....	6
2.3.4 获取当前工作模式.....	6
2.3.5 设置 I2C 时钟频率.....	6
2.3.6 获取当前 I2C 时钟频率.....	6
2.3.7 设置 I2C 通道号.....	6
2.3.8 获取当前 I2C 通道号.....	7
2.3.9 I2C 读操作(Easy I2C 模式) .....	7
2.3.10 I2C 写操作(Easy I2C 模式) .....	8
2.3.11 I/O 端口输出控制.....	9
2.3.12 获取 I/O 端口状态.....	9
2.3.13 产生 I2C 启动时序 (Timing I2C 模式) .....	9
2.3.14 产生写入一个字节的时序 (Timing I2C 模式) .....	10
2.3.15 产生读出一个字节的时序 (Timing I2C 模式) .....	10
2.3.16 产生 I2C 停止时序 (Timing I2C 模式) .....	10
三、利用串口控制指令.....	10
3.1 命令字汇总如下: .....	10
3.2.1 选择 I2C 工作模式 .....	11
3.2.2 获取 I2C 工作模式 .....	11
3.2.3 选择 I2C 当前通道号 .....	11
3.2.4 获取 I2C 当前通道号 .....	12
3.2.5 设置当前 I2C 通道的 I2C 速率.....	12
3.2.6 获取当前 I2C 通道的 I2C 速率.....	12
3.2.7 Easy I2C 写操作 .....	12
3.2.8 Easy I2C 读操作.....	12
3.2.9 产生 I2C 启动时序 (Timing I2C 模式) .....	13
3.2.10 产生 I2C 字节写时序 (Timing I2C 模式) .....	13
3.2.11 产生 I2C 字节读时序 (Timing I2C 模式) .....	13
3.2.12 产生 I2C 停止时序 (Timing I2C 模式) .....	14

本手册适用型号：GY7501A/GY7512/GY7505/GY7506/GY760X

用户利用我们的产品进行二次开发，有两种方式：

- 1) 利用 VCL\_GYI2C 动态链接库文件，调用我们已封装好的各功能函数，直接开发基于 Windows 的用户软件或界面。
- 2) 直接通过串口指令进行二次开发。可以自行编写应用软件或者使用串口调试助手之类的通用软件。注意：此方式支持串口转 I2C 系列的产品。

I2C 转换器上电后的默认参数：

工作模式：EasyI2C 模式

I2C 通道号：0 号通道

I2C 时钟频率：200khz

## 一、Easy I2C 与 Timing I2C

所有 GY7XXX 系列的 I2C 转换器/模块支持两种读写操作模式。

### 1.1 Esay I2C 模式：

可直接通过命令读写数据，无须考虑去产生 I2C 的时序。

工作过程：转换器/模块得到该命令以后，进行解析，然后启动内部的 I2C 读写控制时序，将上位机要求的操作完成以后，再将结果返回给上位机

优点：该方式简单方便，快速，推荐使用。用户不需要了解 I2C 时序协议。

I2C 时钟频率从 1k—800khz 可设置。

局限性：受内部缓冲区的限制，一次命令最多读出来的数据为 512 个，一次最多写入的数据为 520 个（包含命令字）。

### 1.2 Timing I2C 模式：

I2C 时序由上位机软件或命令来控制，分如下 4 种命令。

- 1)产生 I2C 启动时序状态。
- 2)写入 8 个 bit，即一个字节，之后获取并返回 ACK 状态
- 3)读出 8 个 bit，即一个字节，之后给出 ACK 或 NACK
- 4)产生 I2C 停止时序状态

优点：用户通过上位机软件自行控制 I2C 的时序，时序完全透明开放。可读写的长度不受限制，由用户控制。

局限性：步骤繁琐，用户需要熟悉 I2C 时序才能使用该方式。

来回通信握手判断，对速度有影响。

I2C 时钟频率可设置的范围：1k—235khz 可设置

## 二、 VCI\_GYI2C 库文件接口函数

共有 3 个文件 VCI\_GYI2C.DLL, VCI\_GYI2C.LIB, SiUSBXP.DLL。

VCI\_GYI2C.H 的函数定义文件可直接用于 Visual c++。

VCI\_GYI2C.BAS 的函数定义文件可直接用于 Visual Basic。

### 2.1 设备类型定义

```
#define DEV_GY7501A  1    //1ch-I2C
#define DEV_GY7512  2    //2ch-I2C
#define DEV_GY7514  3    //4ch-I2C
#define DEV_GY7518  4    //8ch-I2C
#define DEV_GY7503  5    //1ch-I2C
#define DEV_GY7506  6    //1ch-I2C,module/
#define DEV_GY7601  7    //1ch-I2C
#define DEV_GY7602  8    //2ch-I2C
#define DEV_GY7604  9    //4ch-I2C
#define DEV_GY7608 10   //8ch-I2C
```

### 2.2 数据结构体定义

```
typedef struct GYI2C_DATA_INFO{
    BYTE SlaveAddr;    //设备物理地址，bit7-1 有效
    BYTE Databuffer[520]; //Data 报文的数据；
    UINT WriteNum;    //地址和数据的总个数
    UINT ReadNum;    //需要读的数据的个数
    BYTE IoSel;    //1 表示被选择，将被读/写，bit3-0 分别表示 4 个 IO 口
    BYTE IoData;    //IO 口状态，bit3-0 分别表示 4 个 IO 口
                    //只有与 IoSel 中为 1 的位相同的位值有效
    UINT DlyMsRead; //I2C 读操作时，PC 发出读命令后，延时多少 ms 请求读到的数据。
    BYTE Reserved[4]; //Reserved 系统保留。
}GYI2C_DATA_INFO,*pGYI2C_DATA_INFO;
```

注：UINT 的数据类型 4 字节宽度

### 2.3 接口函数定义与描述

```
extern "C"
{
    DWORD __stdcall GYI2C_Open(DWORD DeviceType, DWORD DeviceInd, DWORD Reserved);
    DWORD __stdcall GYI2C_Close(DWORD DeviceType, DWORD DeviceInd);
```

```
DWORD __stdcall GYI2C_SetMode(DWORD DeviceType ,DWORD DeviceInd, BYTE ModeValue);
DWORD __stdcall GYI2C_GetMode(DWORD DeviceType, DWORD DeviceInd);
DWORD __stdcall GYI2C_SetClk(DWORD DeviceType, DWORD DeviceInd, DWORD ClkValue);
DWORD __stdcall GYI2C_GetClk(DWORD DeviceType, DWORD DeviceInd);
DWORD __stdcall GYI2C_SetChannel(DWORD DeviceType, DWORD DeviceInd, BYTE ChValue);
DWORD __stdcall GYI2C_GetChannel(DWORD DeviceType,DWORD DeviceInd);
DWORD __stdcall GYI2C_Read(DWORD DeviceType, DWORD DeviceInd, pGYI2C_DATA_INFO
pDataInfo);
DWORD __stdcall GYI2C_Write(DWORD DeviceType ,DWORD DeviceInd, pGYI2C_DATA_INFO
pDataInfo);
DWORD __stdcall GYI2C_SetIO(DWORD DeviceType, DWORD DeviceInd, pGYI2C_DATA_INFO
pDataInfo);
DWORD __stdcall GYI2C_GetIO(DWORD DeviceType, DWORD DeviceInd, pGYI2C_DATA_INFO
pDataInfo);
DWORD __stdcall GYI2C_Start(DWORD DeviceType, DWORD DeviceInd);
DWORD __stdcall GYI2C_WriteByte(DWORD DeviceType, DWORD DeviceInd, BYTE DataValue);
DWORD __stdcall GYI2C_ReadByte(DWORD DeviceType, DWORD DeviceInd, BYTE AckValue);
DWORD __stdcall GYI2C_Stop(DWORD DeviceType, DWORD DeviceInd);

}
```

### 2.3.1 设备打开

```
DWORD __stdcall GYI2C_Open(DWORD DeviceType, DWORD DeviceInd, DWORD
Reserved);
```

参数 DeviceType: 设备类型, 例如 DEV\_GY7501A

参数 DeviceInd: 设备索引号。指示第几个设备。如果为串口转 I2C 模块, 则输入 0 表示串口 1, 输入 1 表示串口 2

参数 Reserved: 只有串口转 I2C 适配器才会用到该参数。表示串口波特率, 值为 9600, 19200, 57600, 115200 等。

返回值: 1 表示成功, 0 表示失败

### 2.3.2 设备关闭

```
DWORD __stdcall GYI2C_Close(DWORD DeviceType, DWORD DeviceInd);
```

参数 DeviceType: 设备类型, 例如 DEV\_GY7501A

参数 DeviceInd: 设备索引号。指示第几个设备。如果为串口转 I2C 模块, 则输入 0 表示串口 1, 输入 1 表示串口 2

返回值: 1 表示成功, 0 表示失败

### 2.3.3 设置工作模式

DWORD \_\_stdcall GYI2C\_SetMode(DWORD DeviceType ,DWORD DeviceInd, BYTE ModeValue);

参数 DeviceType: 设备类型, 例如 DEV\_GY7501A

参数 DeviceInd: 设备索引号。指示第几个设备。如果为串口转 I2C 模块, 则输入 0 表示串口 1, 输入 1 表示串口 2

参数 ModeValue: 设置当前通道的 I2C 工作模式。0 表示 Easy I2C 模式, 1 表示 Timing I2C 模式。

返回值: 1 表示成功, 0 表示失败, -1 表示设备未打开

### 2.3.4 获取当前工作模式

DWORD \_\_stdcall GYI2C\_GetMode(DWORD DeviceType, DWORD DeviceInd);

参数 DeviceType: 设备类型, 例如 DEV\_GY7501A

参数 DeviceInd: 设备索引号。指示第几个设备。如果为串口转 I2C 模块, 则输入 0 表示串口 1, 输入 1 表示串口 2

返回值: 返回当前通道的 I2C 工作模式。0 表示 Easy I2C 模式, 1 表示 Timing I2C 模式。

### 2.3.5 设置 I2C 时钟频率

DWORD \_\_stdcall GYI2C\_SetClk(DWORD DeviceType, DWORD DeviceInd, DWORD ClkValue);

参数 DeviceType: 设备类型, 例如 DEV\_GY7501A

参数 DeviceInd: 设备索引号。指示第几个设备。如果为串口转 I2C 模块, 则输入 0 表示串口 1, 输入 1 表示串口 2

参数 ClkValue: 设置当前通道的 I2C 时钟频率, 单位 khz

返回值: 1 表示成功, 0 表示失败, -1 表示设备未打开

### 2.3.6 获取当前 I2C 时钟频率

DWORD \_\_stdcall GYI2C\_GetClk(DWORD DeviceType, DWORD DeviceInd);

参数 DeviceType: 设备类型, 例如 DEV\_GY7501A

参数 DeviceInd: 设备索引号。指示第几个设备。如果为串口转 I2C 模块, 则输入 0 表示串口 1, 输入 1 表示串口 2

返回值: 返回当前通道的 I2C 时钟频率, 单位 khz。

### 2.3.7 设置 I2C 通道号

DWORD \_\_stdcall GYI2C\_SetChannel(DWORD DeviceType, DWORD DeviceInd, BYTE ChValue);

参数 DeviceType: 设备类型, 例如 DEV\_GY7501A

参数 DeviceInd: 设备索引号。指示第几个设备。如果为串口转 I2C 模块, 则输入 0 表示串口 1, 输入 1 表示串口 2

参数 ChValue: 选择 I2C 通道号, 作为当前工作的通道。

返回值: 1 表示成功, 0 表示失败, -1 表示设备未打开

### 2.3.8 获取当前 I2C 通道号

```
DWORD __stdcall GYI2C_GetChannel(DWORD DeviceType, DWORD DeviceInd);
```

参数 DeviceType: 设备类型, 例如 DEV\_GY7501A

参数 DeviceInd: 设备索引号。指示第几个设备。如果为串口转 I2C 模块, 则输入 0 表示串口 1, 输入 1 表示串口 2

返回值: 返回当前通道的 I2C 工作通道。

### 2.3.9 I2C 读操作(Easy I2C 模式)

```
DWORD __stdcall GYI2C_Read(DWORD DeviceType, DWORD DeviceInd, pGYI2C_DATA_INFO pDataInfo);
```

参数 DeviceType: 设备类型, 例如 DEV\_GY7501A

参数 DeviceInd: 设备索引号。指示第几个设备。如果为串口转 I2C 模块, 则输入 0 表示串口 1, 输入 1 表示串口 2

参数 pDataInfo 是指向 pGYI2C\_DATA\_INFO 结构体的指针, 所需要填入的参数如下:

```
BYTE SlaveAddr; //设备物理地址, bit7-1 有效
```

```
BYTE Databuffer[520]; //Data 报文的数据; (如果需要先写地址, 则依次填入到里面)
```

```
UINT WriteNum; //待写入的 ROM 地址的总个数 (如果为立即地址读, 则该参数设为 0, 如果先写地址再读, 则该值为需要先写入的地址个数)
```

```
UINT ReadNum; //需要读的字节个数
```

```
UINT DlyMsRead; //I2C 读操作时, PC 发出读命令后, 适配器得到这个命令, 然后开始读 I2C, 完成后, PC 请求适配器将数据传过来。这个参数就是 PC 发出读命令后到再发请求命令之间的延时时间。DLL 中需要用到这个值。大概的计算方法举例: I2C 速率 5khz, 则一个字节大约需要 2ms, 则读 256 字节时仅 I2C 操作就需要 256*2=512ms, 以及为其他传输留有余量, 则应设置为 600~800ms 为宜。
```

返回值: 1 表示成功, 0 表示失败, -1 表示设备未打开

如果返回为 1, 则可以从 Databuffer[520]数组中取出已读到的数据。

*注: 对于客户如果使用 labview 或者 VB2005 之类的开发环境, 对 GYI2C\_Read 函数有问题, 请使用 GYI2C\_Read2 函数。该函数与 GYI2C\_Read 函数功能一样, 仅是入口参数类型差别。*

```
DWORD __stdcall DWORD GYI2C_Read2(DWORD DeviceType, DWORD DeviceInd, BYTE *buf, DWORD buflen)
```

```

buf[0]=slaveaddr; //I2C 从设备地址
buf[1]=WriteData0; //如果是当前地址读，则不需要
                    //指定地址读的时候，需要在 I2C 读之前先写入的内部地址或其他命
                    令。比如读 AT24C02, 则为 24c02 的内部地址
buf[2]=WriteData1; //如果是单字节内部地址，或是对当前地址读，则不需要 WriteData1
...
buf[buflen-2]=ReadNum>>8;//高 8 位，需要读的数据字节个数
buf[buflen-1]=ReadNum;
数组的最后 2 个字节表示需要读取的字节个数。

```

举例：AT24C02 指定地址读：A1, 00, 01, 00 //读数据长度 0100=256

当前地址读：A1, 01, 00 //直接读，不指定内部地址。

AT24C256 指定地址读：A1, 00, 00, 01, 00

### 2.3.10 I2C 写操作(Easy I2C 模式)

```

DWORD __stdcall GYI2C_Write(DWORD DeviceType ,DWORD DeviceInd,
pGYI2C_DATA_INFO
pDataInfo);

```

参数 DeviceType: 设备类型，例如 DEV\_GY7501A

参数 DeviceInd: 设备索引号。指示第几个设备。如果为串口转 I2C 模块，则输入 0 表示串口 1，输入 1 表示串口 2

参数 pDataInfo 是指向 pGYI2C\_DATA\_INFO 结构体的指针，所需要填入的参数如下：

```

BYTE SlaveAddr; //设备物理地址，bit7-1 有效

```

```

BYTE Databuffer[520]; //Data 报文的数据;(将待写数据内容依次填入到数组
里面)

```

```

UINT WriteNum; //待写入的数据的总个数

```

```

UINT ReadNum; //需要读的字节个数，填入 0 即可。

```

```

UINT DlyMsRead; //I2C 读操作时，PC 发出写命令后，适配器得到这个命令，然后
开始写 I2C，完成后，PC 询问适配器是否完成。这个参数就是 PC 发出写命令后到再发询问
结果命令之间的延时时间。DLL 中需要用到这个值。大概的计算方法举例：I2C 速率 5khz，
则一个字节大约需要 2ms，则写 256 字节时仅 I2C 操作就需要 256*2=512ms，以及为其他
传输留有余量，则应设置为 600~800ms 为宜。

```

返回值：1 表示成功，0 表示失败，-1 表示设备未打开

*注：对于客户如果使用 labview 或者 VB2005 之类的开发环境，对 GYI2C\_Write 函数有问题，请使用 GYI2C\_Write2 函数。该函数与 GYI2C\_Write 函数功能一样，仅是入口参数类型差别。*

```

DWORD __stdcall GYI2C_Write2(DWORD DeviceType, DWORD DeviceInd, BYTE *buf, int
buflen)

```

```
Example: WRITE buf[]= A0, 00, 11, 22, 33, 44
          Buflen=6
```

### 2.3.11 I/O 端口输出控制

```
DWORD __stdcall GYI2C_SetIO(DWORD DeviceType, DWORD DeviceInd,
pGYI2C_DATA_INFO pDataInfo);
```

有 IO 口的适配器才有此功能。如 GY7501A, GY7512。另外: GY7512 中, 只有 I2C-1 通道没有被选择的时候才具备 4 个 IO 口, 否则只有 2 个 IO 口 (IO-0, IO-1)

参数 DeviceType: 设备类型, 例如 DEV\_GY7501A

参数 DeviceInd: 设备索引号。指示第几个设备。如果为串口转 I2C 模块, 则输入 0 表示串口 1, 输入 1 表示串口 2

参数 pDataInfo 是指向 pGYI2C\_DATA\_INFO 结构体的指针, 所需要填入的参数如下:

```
BYTE IoSel; //bit 位为 1 表示被选择, 将被读/写。Bit3-0 分别对应 4 个 IO
口
```

```
BYTE IoData; //IO 口状态, bit3-0 分别表示 4 个 IO 口
//只有与 IoSel 中为 1 的位相同的位值有效
```

返回值: 1 表示成功, 0 表示失败, -1 表示设备未打开

### 2.3.12 获取 I/O 端口状态

```
DWORD __stdcall GYI2C_GetIO(DWORD DeviceType, DWORD DeviceInd,
pGYI2C_DATA_INFO pDataInfo);
```

有 IO 口的适配器才有此功能。如 GY7501A, GY7512。另外: GY7512 中, 只有 I2C-1 通道没有被选择的时候才具备 4 个 IO 口, 否则只有 2 个 IO 口 (IO-0, IO-1)

参数 DeviceType: 设备类型, 例如 DEV\_GY7501A

参数 DeviceInd: 设备索引号。指示第几个设备。如果为串口转 I2C 模块, 则输入 0 表示串口 1, 输入 1 表示串口 2

参数 pDataInfo 是指向 pGYI2C\_DATA\_INFO 结构体的指针, 得到的 IO 状态将存在该结构体中:

```
BYTE IoSel; //bit 位为 1 表示被选择, 将被读/写。Bit3-0 分别对应 4 个 IO
口
BYTE IoData; //IO 口状态, bit3-0 分别表示 4 个 IO 口的状态
```

返回值: 1 表示成功, 0 表示失败, -1 表示设备未打开

### 2.3.13 产生 I2C 启动时序 (Timing I2C 模式)

```
DWORD __stdcall GYI2C_Start(DWORD DeviceType, DWORD DeviceInd);
```

参数 DeviceType: 设备类型, 例如 DEV\_GY7501A

参数 DeviceInd: 设备索引号。指示第几个设备。如果为串口转 I2C 模块, 则输入 0 表示串口 1, 输入 1 表示串口 2

返回值: 1 表示成功, 0 表示失败, -1 表示设备未打开

### 2.3.14 产生写入一个字节的时序 (Timing I2C 模式)

```
DWORD __stdcall GYI2C_WriteByte(DWORD DeviceType, DWORD DeviceInd, BYTE  
DataValue);
```

参数 DeviceType: 设备类型, 例如 DEV\_GY7501A

参数 DeviceInd: 设备索引号。指示第几个设备。如果为串口转 I2C 模块, 则输入 0 表示串口 1, 输入 1 表示串口 2

参数 DataValue: 待写入的字节数据。

返回值: 1 表示得到了从设备的 ACK, 0 表示得到了 NACK。

### 2.3.15 产生读出一个字节的时序 (Timing I2C 模式)

```
DWORD __stdcall GYI2C_ReadByte(DWORD DeviceType, DWORD DeviceInd, BYTE  
AckValue);
```

参数 DeviceType: 设备类型, 例如 DEV\_GY7501A

参数 DeviceInd: 设备索引号。指示第几个设备。如果为串口转 I2C 模块, 则输入 0 表示串口 1, 输入 1 表示串口 2

参数 AckValue: 1 表示读完一个字节后, 给出 ACK; 0 表示读完一个字节后给出 NACK.

返回值: 1 表示成功, 0 表示失败, -1 表示设备未打开

### 2.3.16 产生 I2C 停止时序 (Timing I2C 模式)

```
DWORD __stdcall GYI2C_Stop(DWORD DeviceType, DWORD DeviceInd);
```

参数 DeviceType: 设备类型, 例如 DEV\_GY7501A

参数 DeviceInd: 设备索引号。指示第几个设备。如果为串口转 I2C 模块, 则输入 0 表示串口 1, 输入 1 表示串口 2

返回值: 1 表示成功, 0 表示失败, -1 表示设备未打开

## 三、利用串口控制指令

对模块的参数设置和读写 I2C 从设备, 均通过输入串口命令数据来进行。(电脑上可用串口调试助手等串口软件测试, 8 个数据位, 1 个停止位, 无奇偶校验, 串口波特率选被设置的值, 如果 COMSET0,COMSET1 悬空, 则是 115200bps)

### 3.1 命令字汇总如下:

```
#define CMD_SET_MODE      0x10    //0~EasyI2C 模式, 1~TimingI2C 模式  
#define CMD_GET_MODE     0x11    //0~EasyI2C 模式, 1~TimingI2C 模式
```

```

#define CMD_SET_CHANNEL 0x40 //选择当前 I2C 通道号
#define CMD_GET_CHANNEL 0x41 //查询当前 I2C 通道号
#define CMD_SET_CLKVALUE 0x42 //设置 I2C 时钟频率, 单位 KHZ
#define CMD_GET_CLKVALUE 0x43 //查询 I2C 时钟频率, 单位 KHZ
#define CMD_SEND_DATA 0x44 //EasyI2C 模式的读写命令字
#define CMD_SET_STA 0x60 //TimingI2C 模式, 产生 I2C 启动时序
#define CMD_WRITE_BYTE 0x61 //TimingI2C 模式, 产生 I2C 写一个字节时序
#define CMD_READ_BYTE 0x62 //TimingI2C 模式, 产生 I2C 读一个字节时序
#define CMD_SET_STO 0x63 //TimingI2C 模式, 产生 I2C 停止时序

```

(以下命令字和数据都为 16 进制表示)

### 3.2.1 选择 I2C 工作模式

```
#define CMD_SET_MODE 0x10 //0~Easy I2C 读写模式, 1~Timing I2C 读写模式.
```

(执行命令后, 内部 I2C 引脚重新配置)

格式: 命令字 40 + 需要选择的 I2C 通道号

举例:

10 00            设置为 EasyI2C 一体化读写工作模式

40 01            设置为 TimingI2C 分步控制读写工作模式

返回值: AA

### 3.2.2 获取 I2C 工作模式

```
#define CMD_GET_MODE 0x11//查询当前 I2C 工作模式
```

格式: 命令字 11

返回值: 当前 I2C 工作模式

举例:

11            返回值 00            当前为 Easy I2C 读写模式

11            返回值 01            当前为 Timing I2C 模式

### 3.2.3 选择 I2C 当前通道号

```
#define CMD_SET_CHANNEL 0x40 //选择当前 I2C 通道号
```

(执行命令后, 内部 I2C 引脚重新配置)

格式: 命令字 40 + 需要选择的 I2C 通道号

举例:

40 00            选择 0 号 I2C 通道作为当前通道

40 03            选择 3 号 I2C 通道作为当前通道

返回值: AA

默认设置: 如果不进行此设置, 则默认为 00

### 3.2.4 获取 I2C 当前通道号

```
#define CMD_GET_CHANNEL 0x41 //查询当前 I2C 通道号
```

格式：命令字 41

返回值：当前工作的 I2C 通道号

举例：

41                    返回值 01                    当前工作的通道索引号为 01 ， 即第 01 路 I2C 接口

### 3.2.5 设置当前 I2C 通道的 I2C 速率

```
#define CMD_SET_CLKVALUE 0x42 //设置 I2C 时钟频率， 单位 KHZ
```

格式：命令字 42 + 速率的高字节+速率的低字节

举例：

42 00 64            将当前 I2C 通道的速率设置为 0x0064 即 100khz

42 01 90            将当前 I2C 通道的速率设置为 0x0190 即 400khz

返回值：AA

默认设置：如果不进行此设置，则默认为 00 64， 即 100khz

### 3.2.6 获取当前 I2C 通道的 I2C 速率

```
#define CMD_GET_CLKVALUE 0x43 //查询 I2C 时钟频率， 单位 KHZ
```

格式：命令字 43 举例：

43    返回值 00 64            当前 I2C 通道的速率为 0x0064 即 100khz

43    返回值 01 90            当前 I2C 通道的速率为 0x0190 即 400khz

### 3.2.7 Easy I2C 写操作

```
#define CMD_SEND_DATA 0x44 //一体化模式的读写命令字
```

格式：

命令字	设备地址+R/W	ROM 地址， 数据
44	7 位设备地址+读写位为 0	依次写入内部 ROM 或寄存器的 地址和数据

注：一个命令帧的总长度最大为 260 个字节

举例：（slaveaddress+W =0xA0 ）

44 A0 00 33 44    返回值 0xAA            依次写入地址 00， 数据 33,44。

44 A0 00            返回值 0xAA            只写入地址 00

                    返回值 0xBB            错误

### 3.2.8 Easy I2C 读操作

```
#define CMD_SEND_DATA 0x44 //EasyI2C 模式的读写命令字
```

格式：

命令字	设备地址+R/W	ROM 地址	长度（该命令帧的最后
-----	----------	--------	------------

			一个字节)
44	7 位设备地址+读写位为 1	一般有 1-2 个字 节	希望读的个数减 1

举例：(slaveaddress+R =0xA1 )

44 A1 FF 直接启动读，正常会返回值 256 个数据 可读出 256 个 (0xFF+1) 字节。

44 A1 00 FF 随机读 (random read)

I2C 接口会先写地址 00，然后从该地址读，要求读数据个数 256。  
正常会返回值 256 个所读到的数据。

44 A1 00 00 07 随机读 (random read)

I2C 接口先写地址 00 00，然后从该地址读，要求读数据个数 8，  
正常会返回值 8 个所读到的数据。

返回值 0xBB 错误

### 3.2.9 产生 I2C 启动时序 (Timing I2C 模式)

```
#define CMD_SET_STA 0x60 //TimingI2C 模式，产生 I2C 启动时序
```

格式：命令字 60

举例：

60 执行命令后，会在当前 I2C 通道上产生 I2C 启动时序波形

返回值：AA

### 3.2.10 产生 I2C 字节写时序 (Timing I2C 模式)

```
#define CMD_WRITE_BYTE 0x61 //TimingI2C 模式，产生 I2C 写一个字节的时序
```

格式：命令字 61 + 待写字节 (8 个 bit)

返回值：该字节写入后，获得的 ACK 状态。1 表示得到了 ACK，0 表示得到 NACK。

举例：

61 A0 执行命令后，会在当前 I2C 通道产生 I2C 时序将内容 A0 串行移出。

返回值：1 表示得到了 ACK。说明上位机可继续发写命令

### 3.2.11 产生 I2C 字节读时序 (Timing I2C 模式)

```
#define CMD_READ_BYTE 0x62 //TimingI2C 模式，产生 I2C 读写一个字节的时序
```

格式：命令字 62 +(读字节后的需要给出的 ACK 命令)

1 表示读字节完成后给 ACK，0 表示读字节完成后给 NACK。

返回值：读出的一个字节数据。

举例：

62 01 读完一个字节后，给 ACK 允许继续传输。

返回值：1 表示得到了 ACK。上位机可继续发写命令

### 3.2.12 产生 I2C 停止时序 (Timing I2C 模式)

#define CMD\_SET\_STO            0x63     //TimingI2C 模式，产生 I2C 停止时序

格式：命令字 63

举例：

63            执行命令后，会在当前 I2C 通道上产生 I2C 停止时序

返回值：AA