

Acute Data Generator – SPI/SIPI Protocol Software development kit (SDK) Programming guide

For Data Generator 3000 and TravelData 3000

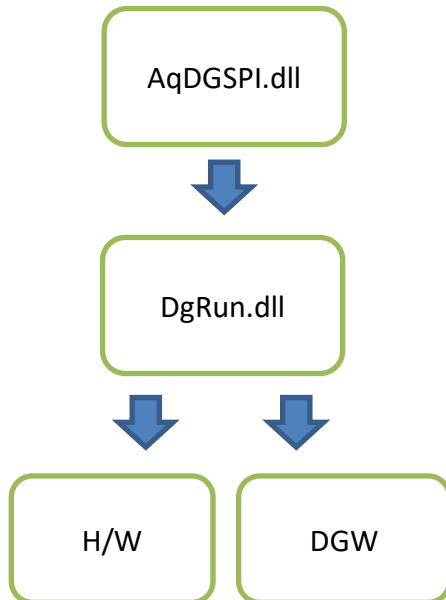
Version: 1.0

Publish: 2020/4/14

内容

SDK 架构介绍	3
SDK 函式说明	3
bool InitProtocol(int iDGModel, bool fConnect).....	3
HDGPTL SPI_Init(UINT32 iProtocolClockFreqInKHz,UINT32 iDGInitState, bool fSingleStep, int* piChBuf, int iFormat, int iProtocolType, int iWordSize, bool fRepeat).....	4
bool CloseProtocol(HDGPTL hDGPTl).....	5
bool ClearProtocolPacket(HDGPTL hDGPTl).....	6
int SaveProtocolList(HDGPTL hDGPTl, bool fFile, char* pPtr).....	6
bool AppendDGInstruction(HDGPTL hDGPTl, int iInst, int iParam, DGADDR iDGAddr).....	7
int GetLastDGError().....	8
int GetPodNum().....	8
bool SetOutputVolt(int imV, int iPodIndex).....	8
bool OutputProtocol(HDGPTL hDGPTl).....	9
int GetDGStatus().....	9
bool StopDG().....	9
bool ShutdownDG().....	10
int GetProtocolName(HDGPTL hDGPTl, char* pBuf, int iBufSize).....	10
DGADDR SPI_AppendIdle(HDGPTL hDGPTl, UINT32 nsecs).....	10
DGADDR 4WireSPI_AppendPacket (HDGPTL hDGPTl, UINT64* pi64DatBuf, int iDatSize, int iSlaveRespSet).....	11
DGADDR 3WireSPI_AppendPacket(HDGPTL hDGPTl, UINT64* pi64DatBuf, int iDatSize, bool fUseWrLatencyRd, int* piBitLengBuf, int iFrameGuardTime, int iSlaveRespSet).....	11
DGADDR 2WireSPI_AppendPacket(HDGPTL hDGPTl, UINT64* pi64DatBuf, int iDatSize, bool fUseWrLatencyRd, int* piBitLengBuf, int iFrameGuardTime, int iSlaveRespSet).....	12
DGADDR SIPI_AppendPacket(HDGPTL hDGPTl, inti Type, int iClockNum, UINT64 i64Dat).....	13
DGADDR SPI_InputFile(HDGPTL hDGPTl, char* pStrFile, UINT32 nsecs).....	14
DGADDR SIPI_InputFile(HDGPTL hDGPTl, char* pStrFile).....	14

SDK 架构介绍



此 SDK 提供一个开放接口让用户可以 2 种方式来发送 SPI/SIPI 波形。

1. 以一个一个 SPI/SIPI 封包的形式发送。
2. 一次发送所有 SPI/SIPI 封包的形式。

SDK 函式说明

```

typedef unsigned int UINT32;
typedef unsigned int HDGPTL;
typedef unsigned int DGADDR;
typedef __int64 I64;
typedef unsigned __int64 UI64;
  
```

bool InitProtocol(int iDGModel, bool fConnect)

功能

寻找并启动目前接在此计算机上的数据产生器。

参数

iDGModel[in]:

Type: **int**

选择数据产生器机种，列举如下：

```
enum DG_HW_MODEL
```

```
{
    DG3064B = 0x33064,
    DG3096B = 0x33096,
    DG3128B = 0x33128,
    TD3008E = 0x23008,
    TD3116B = 0x23116,
    TD3216B = 0x23216,
};
```

fConnect[in]:

Type: **bool**

设置连接模式，false 表示 demo 模式。

回传值

如果回传值为 True，代表模式设置成功。如果回传 False 值则代表设置失败。

备注

InitProtocol(TD3216B, false);

// 选择 TD3216B 机种并设置 demo 模式。

HDGPTL SPI_Init(UINT32 iProtocolClockFreqInKHz, UINT32 iDGInitState, bool fSingleStep, int* piChBuf, int iFormat, int iProtocolType, int iWordSize, bool fRepeat)

功能

初始化 SPI/SIPI 总线设置。

参数

iProtocolClockFreqInKHz[in]:

Type: **UINT32**

设置 SPI/SIPI 频率，单位: KHz。

iDGInitState[in]:

Type: **UINT32**

设置 SPI/SIPI 总线起始状态，共有下列 3 种:

DGINIT_STATE_LOW = 0,

DGINIT_STATE_HIGH = 1,

DGINIT_STATE_HI_Z = 2,

fSingleStep[in]:

Type: **bool**

设置发送模式，选择 true 时，数据产生器将以一个接一个封包的方式来发送

SPI/SIPI 封包，反之则是一次全部发送。

piChBuf [in]:

Type: **int***

设置 SPI/SIPI 发送通道编号。

该通道没有使用时，设为 -1。

piChBuf[0]: CS

piChBuf[1]: SCK; SIPI-CLK

piChBuf[2]: SDI(SDA); SIPI-DATA

piChBuf[3]: SDO

iFormat[in]:

Type: **int**

选择发送的文件格式。

#define DGFMT_DGW 0x0001

iProtocolType[in]:

Type: **int**

0 = SPI

1 = SIPI

iWordSize[in]:

Type: **int**

4~40 bit

fRepeat[in]:

Type: **bool**

设置是否重复发送。

回传值

回传 handle 数值，代表模式设置成功。如果回传 -1 则代表设置失败。

备注

int iChNoBuf[] = {0, 1, 2, 3};

HDGPTL hDG = m_pSPIInit(1000, DGINIT_STATE_LOW, **true**, iChNoBuf, DGFMT_DGW,

SPI_PROTOCOL, 8, **false**);

// 设置 1 MHz 频率 SPI，波形初始阶段为low，

// 选择一个封包接着一个封包发送形式，

// 设置 CH-00 / CH-01 / CH-02 / CH-03为SPI CS / SCK / SDI / SDO通道，， DGW 格式，

// 非重复发送。

bool CloseProtocol(HDGPTL** hDGptl)**

功能

释放 SDK 所占用的资源。

参数

hDGPtr[in]:

Type: **HDGPTL**

SPI/SIPI 封包序列 handle。

回传值

如果回传值为 **True**，代表模式设置成功。如果回传 **False** 值则代表设置失败。

bool ClearProtocolPacket(HDGPTL** hDGPtr)**

功能

清除 SPI/SIPI 封包序列。

参数

hDGPtr[in]:

Type: **HDGPTL**

SPI/SIPI 封包序列 handle。

回传值

如果回传值为 **True**，代表模式设置成功。如果回传 **False** 值则代表设置失败。

int SaveProtocolList(HDGPTL** hDGPtr, **bool** fFile, **char*** pPtr)**

功能

将 SPI/SIPI 封包序列存档。

参数

hDGPtr[in]:

Type: **HDGPTL**

SPI/SIPI 封包序列 handle。

fFile[in]:

Type: **bool**

存入档案(true) 或存入缓冲(false)。

pPtr[in]:

Type: **char***

当 **fFile = true** 表示档案路径反之则为缓冲区指标。

回传值

回传档案或缓冲大小，代表模式设置成功。如果回传 **-1** 则代表设置失败。

备注

SaveProtocolList(hDG, true, "SPI.dgw");

// 储存 “SPI.dgw” 档案

**bool AppendDGInstruction(HDGPtL hDGPtL, int iInst, int iParam, DGADDR
iDGAddr)**

功能

加入数据产生器指令。

参数

hDGPtL[in]:

Type: **HDGPtL**

SPI/SIPI 封包序列 handle。

iInst[in]:

Type: **int**

设置 DG 指令, e.g. JP, LC, LP。

#define CMD_NP	0	// No Operation
#define CMD_LC	1	// Loop Count
#define CMD_LP	2	// Loop to New Address
#define CMD_JP	3	// Jump to New Address
#define CMD_WE	5	// Wait Event
#define CMD_HD	7	// Hold Count

iParam[in]:

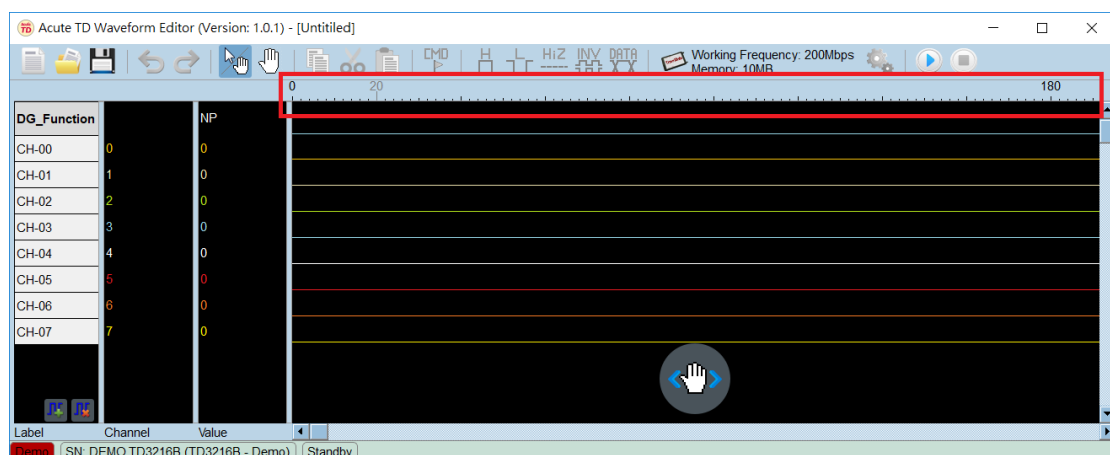
Type: **int**

设置 DG 指令参数, e.g. JP 10, iParam = 10.

iDGAddr[in]:

Type: **DGADDR**

设置欲插入指令位置, 请参考下方撷图。



回传值

如果回传值为 **True**，代表模式设置成功。如果回传 **False** 值则代表设置失败。

备注

```
AppendDGInstruction(hDG, CMD_JP, 0, 1000);  
// 设定 DG 指令:在 DG address = 1000 插入 JP 0 指令。
```

int GetLastDGError()

功能

取得错误代码。

回传错误码或是 0 表示无错误。

错误码

#define ERR_MSG_FILE_NOT_FOUND	0x0001
#define ERR_MSG_CANT_FIND_DLL	0x1001
#define ERR_MSG_EMPTY_SLOT	0x1002
#define ERR_MSG_NO_HARDWARE	0x1004
#define ERR_MSG_INVALID_WORK_FREQ	0x1005
#define ERR_MSG_DUPLICATED_CH_NO	0x1006
#define ERR_MSG_CONFLICTED_HIZ_CH_NO	0x1007
#define ERR_MSG_INVALID_STATUS	0x1008
#define ERR_MSG_NOT_UNDER_CAPTURE	0x1009
#define ERR_MSG_NONEXISTENT_HANDLE	0x100A
#define ERR_MSG_INVALID_IDLE_TIME	0x100B
#define ERR_MSG_OVER_DATA_BUFF_SIZE	0x100C

int GetPodNum()

功能

取得 POD 数量。

回传值

TD3216B = 2, DG3064B = 6。

bool SetOutputVolt(int imV, int iPodIndex)

功能

设置输出电压。

参数

imV[in]:

Type: **int**

设置单一 pod 输出电压，单位: mV。

iPodIndex[in]:

Type: **int**

设置 pod 索引，从 0 开始。

回传值

如果回传值为 **True**，代表模式设置成功。如果回传 **False** 值则代表设置失败。

备注

```
SetOutputVolt(2500, 0);
```

```
// 设置 Pod 0, 2.5V, Pod 0: CH0 ~ CH7
```

```
SetOutputVolt(2500, 1);
```

```
// 设置 Pod 1, 2.5V, Pod 1: CH8 ~ CH15
```

bool OutputProtocol(HDGPTL** hDGPTl)**

功能

输出 protocol。

参数

hDGPTl[in]:

Type: **HDGPTL**

SPI/SIPI 封包序列 handle。

回传值

如果回传值为 **True**，代表模式设置成功。如果回传 **False** 值则代表设置失败。

int GetDGStatus()

功能

取得资料产生器目前状态。

回传值

回传 **DG_WAVEFORM_SENDING(0x80000000)** 表示 DG 在发送状态，其他数值则为准备状态。

bool StopDG()

功能

停止发送。

回传值

如果回传值为 **True**，代表模式设置成功。如果回传 **False** 值则代表设置失败。

bool ShutdownDG()

功能

关闭数据产生器定中断与计算机的 USB 连接。

回传值

如果回传值为 **True**，代表模式设置成功。如果回传 **False** 值则代表设置失败。

int GetProtocolName(HDGPTL hDGPTl, char* pBuf, int iBufSize)

功能

取得目前总线名称与 ID。

参数

hDGPTl[in]:

Type: **HDGPTL**

SPI/SIPI 封包序列 handle。

pBuf[in]:

Type: **char***

设置总线名称缓冲区。

iBufSize[in]:

Type: **int**

设置缓冲区大小。

回传值

回传总线 ID。

DGADDR SPI_AppendIdle(HDGPTL hDGPTl, UINT32 nsecs)

功能

加入 SPI/SIPI Idle 。

参数

hDGPTl[in]:

Type: **HDGPTL**

SPI/SIPI 封包序列 handle。

nsecs[in]:

Type: **UINT32**

设置 Idle 时间，单位: ns。

回传值

回传 DG address，代表模式设置成功。如果回传 -1 则代表设置失败。

备注

```
SPI_AppendIdle(hDGPTl, 1000000);
```

```
// Set the idle time 1 ms
```

DGADDR 4WireSPI_AppendPacket (HDGPTL hDGPTl, UINT64* pi64DatBuf, int iDatSize, int iSlaveRespSet)

功能

加入 4-Wire SPI 封包。

参数

hDGPTl[in]:

Type: **HDGPTL**

SPI 封包序列 handle。

pi64DatBuf[in]:

Type: **UINT64***

SPI 数据 buffer。

iDatSize[in]:

Type: **int**

设置 SPI 数据宽度: 8 ~ 40 bit。

iSlaveResp[in]:

Type: **int**

设置 slave 回应状态 Hi-Z (SET_HIZ) 或 High (NOT_SET_HIZ)。

回传值

回传 DG address，代表模式设置成功。如果回传 -1 则代表设置失败。

备注

```
I64 i64DatBuf[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};
```

```
4WireSPI_AppendPacket(hDGPTl, i64DatBuf, 8, SET_HIZ);
```

```
// 加入 4-Wire SPI 封包
```

```
// 设置 SDO 为 Hi-Z 状态。
```

DGADDR 3WireSPI_AppendPacket(HDGPTL hDGPTl, UINT64* pi64DatBuf, int iDatSize, bool fUseWrLatencyRd, int* piBitLengBuf, int iFrameGuardTime, int iSlaveRespSet)

功能

加入 3-Wire SPI 封包。

参数

hDGPI[in]:

Type: **HDGPTL**

SPI 封包序列 handle。

pi64DatBuf[in]:

Type: **UINT64***

SPI 数据 buffer。

iDatSize[in]:

Type: **int**

设置 SPI 数据宽度: 8 ~ 40 bit。

fUseWrLatencyRd[in]:

Type: **bool**

设置 SDI(Write)-Latency-SDO(Read)。

piBitLengBuf[in]:

Type: **int**

Write/Read/Latency 长度 (Bits)

piBitLengBuf[0]: Write Length Bits

piBitLengBuf[1]: Read Length Bits

piBitLengBuf[2]: Latency Length Bits

iFrameGuardTime[in]:

Type: **int**

Frame guard time, 单位: ns

iSlaveResp[in]:

Type: **int**

设置 slave 回应状态 Hi-Z (SET_HIZ) 或 High (NOT_SET_HIZ)。

回传值

回传 DG address, 代表模式设置成功。如果回传 -1 则代表设置失败。

DGADDR 2WireSPI_AppendPacket (HDGPTL hDGPI, UINT64* pi64DatBuf, int iDatSize, bool fUseWrLatencyRd, int* piBitLengBuf, int iFrameGuardTime, int iSlaveRespSet)

功能

加入 2-Wire SPI 封包。

参数

hDGPI[in]:

Type: **HDGPTL**

SPI 封包序列 handle。

pi64DatBuf[in]:

Type: **UINT64***

SPI 数据 buffer。

iDatSize[in]:

Type: **int**

设置 SPI 数据宽度: 8 ~ 40 bit。

fUseWrLatencyRd[in]:

Type: **bool**

设置 SDI(Write)-Latency-SDO(Read)。

piBitLengBuf[in]:

Type: **int**

Write/Read/Latency 长度 (Bits)

piBitLengBuf[0]: Write Length Bits

piBitLengBuf[1]: Read Length Bits

piBitLengBuf[2]: Latency Length Bits

iFrameGuardTime[in]:

Type: **int**

Frame guard time, 单位: ns

iSlaveResp[in]:

Type: **int**

设置slave 回应状态Hi-Z (SET_HIZ) 或 High (NOT_SET_HIZ)。

回传值

回传 DG address, 代表模式设置成功。如果回传 -1 则代表设置失败。

DGADDR SIPI_AppendPacket (HDGPTL hDGPTl, int iType , int iClockNum, UINT64 i64Dat)

功能

加入 SIPI 封包。

参数

hDGPTl[in]:

Type: **HDGPTL**

SIPI 封包序列 handle。

iClockNum[in]:

Type: **int**

设置 clock 数量。

i64Dat[in]:

Type: **UINT64**

设置 SIPI 数据。

回传值

回传 DG address，代表模式设置成功。如果回传 -1 则代表设置失败。

DGADDR SPI_InputFile(HDGPTL hDGPTl, char* pStrFile, UINT32 nsecs)

功能

载入 *.txt 或 *.bin 档案。

参数

hDGPTl[in]:

Type: **HDGPTL**

SPI 封包序列 handle。

pStrFile[in]:

Type: **char***

设置 *.txt or *.bin 档案路径。

nsecs[in]:

Type: **UINT32**

设置 Idle, 单位: ns

回传值

回传 DG address，代表模式设置成功。如果回传 -1 则代表设置失败。

DGADDR SIPI_InputFile(HDGPTL hDGPTl, char* pStrFile)

功能

载入 SIPI 文本文件。

Parameters

hDGPTl[in]:

Type: **HDGPTL**

SIPI 封包序列 handle。

pStrFile[in]:

Type: **char***

设置 SIPI 文本文件路径。

回传值

回传 DG address，代表模式设置成功。如果回传 -1 则代表设置失败。