

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

For Data Generator 3000 and TravelData 3000

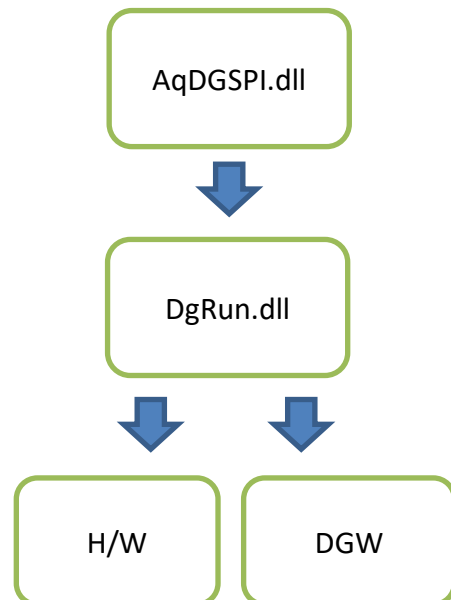
Version: 1.0

Publish: 2019/12/04

Contents

SDK Control Flow and simple introduction.....	3
SDK Function Definitions.....	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().....	9
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, int iType, int iClockNum, UINT64 i64Dat).....	13
DGADDR SPI_InputFile(HDGPTL hDGPTl, char* pStrFile, UINT32 nsecs).....	14
DGADDR SIPI_InputFile(HDGPTL hDGPTl, char* pStrFile).....	14

SDK Control Flow and simple introduction



This SDK provides an open interface for users to generate the SPI/SIPI pattern; there are 2 ways to output the SPI/SIPI pattern:

1. Output SPI/SIPI pattern directly in a packet by packet way.
2. Output the SPI/SIPI pattern directly in all the packets way.

SDK Function Definitions

```
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)

Search and initialize the DG device connected on the computer.

Parameters

iDGModel[in]:

Type: **int**

Select the DG hardware model, enumerating these models as the following

enum DG_HW_MODEL

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

fConnect[in]:
    Type: bool
    Set the connected mode, false is demo mode.
```

Return value

Return true if the function succeeded; false if the function failed.

Remarks

```
InitProtocol(TD3216B, false);
// Select the TD3216B model and demo mode
```

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

Initialize SPI/SIPI protocol parameters.

Parameters

iProtocolClockFreqInKHz[in]:
Type: **UINT32**
Set the SPI/SIPI frequency, unit: KHz.

iDGInitState[in]:
Type: **UINT32**
Set the beginning state of SPI/SIPI pattern, there are 3 states can be select:
DGINIT_STATE_LOW = 0,
DGINIT_STATE_HIGH = 1,
DGINIT_STATE_HI_Z = 2,

fSingleStep[in]:
Type: **bool**
Select **true**, DG will generate the SPI/SIPI pattern in a packet by packet way; select **false**, it will generate the SPI/SIPI pattern in all the packets way.

piChBuf [in]:

Type: **int***

Set DG channel number for SPI/SIPI pin.

-1 if unused.

piChBuf[0]: CS

piChBuf[1]: SCK; SIPI-CLK

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

piChBuf[3]: SDO

iFormat[in]:

Type: **int**

#define DGFMT_DGW 0x0001

Select *.DGW file format.

iProtocolType[in]:

Type: **int**

0 = SPI

1 = SIPI

iWordSize[in]:

Type: **int**

4~40 bit

fRepeat[in]:

Type: **bool**

True: output protocol repeatedly

Return value

Return the handle if the initial succeeded or -1 when failed.

Remarks

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

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

SPI_PROTOCOL, 8, **false**);

// Set 1 MHz frequency for SPI pattern, low state for the beginning of SPI,

// selecting the single step way to output SPI pattern packet by packet,

// set the DG CH-00 / CH-01 / CH-02 / CH-03 as SPI CS / SCK / SDI / SDO

// DGW file format,

// not output repeatedly

bool CloseProtocol(HDGPTL hDGPTl)

Release the resource.

Parameters

hDGPtr[in]:

Type: **HDGPTL**

The handle for SPI/SIPI packet list.

Return value

Return true if the function succeeded; false if the function failed.

bool ClearProtocolPacket(HDGPTL** hDGPtr)**

Clear the protocol packet.

Parameters

hDGPtr[in]:

Type: **HDGPTL**

The handle for SPI/SIPI packet list.

Return value

Return true if the function succeeded; false if the function failed.

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

Save the protocol to the file.

Parameters

hDGPtr[in]:

Type: **HDGPTL**

The handle for SPI/SIPI packet list.

fFile[in]:

Type: **bool**

Save the protocol to the file(true) or to the buffer(false).

pPtr[in]:

Type: **char***

The file path when fFile = true or pointer of the buffer when fFile = false.

The function will return the buffer size when pPtr == NULL.

Return value

Return file/buffer size if the function succeeded; -1 if the function failed.

Remarks

```
SaveProtocolList(hDG, true, "SPI.dgw");  
// Save SPI protocol to the file "SPI.dgw"
```

bool AppendDGInstruction(HDGPTL** hDGPTl, **int** iInst, **int** iParam, **DGADDR** iDGAddr)**

Append the DG instruction to the SPI packet link list.

Parameters

hDGPTl[in]:

Type: **HDGPTL**

The handle for SPI packet list.

iInst[in]:

Type: **int**

The DG instructions, e.g. JP, LC, LP ... and so on.

#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]:

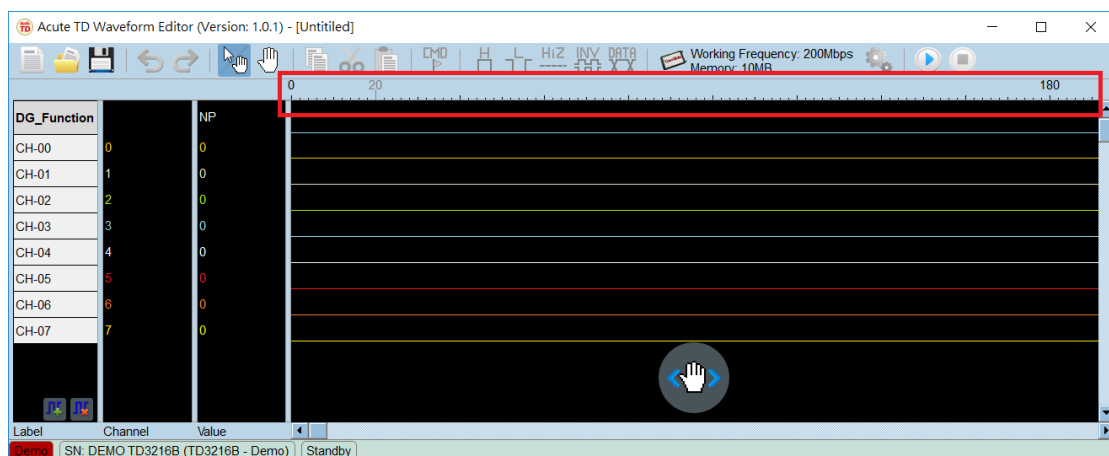
Type: **int**

The DG instruction parameter, e.g. JP 10, iParam = 10.

iDGAddr[in]:

Type: **DGADDR**

The address of DG, referring to the red area of the following snapshot.



Return value

Return true if the function succeeded; false if the function failed.

Remarks

AppendDGInstruction(hDG, CMD_JP, 0, 1000);

```
// Set DG instruction: Jump to New Address 0 at the DG address = 1000
```

int GetLastDGError()

Get Last Error.

Return value

Return the error code if the function failed, 0 if the function succeeded.

Error code

#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()

Get the numbers of pod.

Return value

Return 2 for model TD3216B, 6 for DG3064B.

bool SetOutputVolt(int imV, int iPodIndex)

Set the voltage for each pod.

There is a voltage range as following:

DG3000 series: 0.8V ~ 5V

TD3000 series: 0.8V ~ 4.5V

Parameters

imV[in]:

Type: **int**

The voltage for each pod, unit: mV.

iPodIndex[in]:

Type: **int**

Zero-based index for pod.

Return value

Return true if the function succeeded; false if the function failed.

Remarks

```
SetOutputVolt(3000, 0);
```

```
// Set the Pod 0, 3V, Pod 0: CH0 ~ CH7
```

```
SetOutputVolt(3000, 1);
```

```
// Set the Pod 1, 3V, Pod 1: CH8 ~ CH15
```

bool OutputProtocol(HDGPTL** hDGPTl)**

Output the protocol.

Parameters

hDGPTl[in]:

Type: **HDGPTL**

The handle for SPI/SIPI packet list.

Return value

Return true if the function succeeded; false if the function failed.

int GetDGStatus()

Get the status of DG hardware.

Return value

Return DG_WAVEFORM_SENDING(0x80000000) status or other value for ready.

bool StopDG()

Stop the DG.

Return value

Return true if the function succeeded; false if the function failed.

bool ShutdownDG()

Shutdown the DG and disconnect DG with the computer.

Return value

Return true if the function succeeded; false if the function failed.

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

Get the protocol name & ID.

Parameters

hDGPTl[in]:

Type: **HDGPTL**

The handle for SPI packet list.

pBuf[in]:

Type: **char***

The buffer to storage the protocol name.

iBufSize[in]:

Type: **int**

The buffer size.

Return value

Return the protocol ID.

DGADDR SPI_AppendIdle(HDGPTL hDGPTl, UINT32 nsecs)

Append the idle time to the SPI/SIPI packet link list.

Parameters

hDGPTl[in]:

Type: **HDGPTL**

The handle for SPI/SIPI packet list.

nsecs[in]:

Type: **UINT32**

Idle time between the SPI/SIPI packets, unit:ns

Return value

Return the address of DG if the function succeeded; -1 if the function failed.

Remarks

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

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

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

Append a 4-Wire SPI Packet.

Parameters

hDGPTl[in]:

Type: **HDGPTL**

The handle for SPI packet list.

pi64DatBuf[in]:

Type: **UINT64***

SPI data buffer.

iDatSize[in]:

Type: **int**

Data size: 8 ~ 40 bit.

iSlaveResp[in]:

Type: **int**

Set the slave response as Hi-Z (SET_HIZ) or High (NOT_SET_HIZ).

Return value

Return the address of DG if the function succeeded; -1 if the function failed.

Remarks

i64 i64DatBuf[] = {0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08};

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

// Append an 4-Wire SPI packet

// Set SDO at Hi-Z state

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

Append a 3-Wire SPI packet.

Parameters

hDGPTl[in]:

Type: **HDGPTL**

The handle for SPI packet list.

pi64DatBuf[in]:

Type: **UINT64***

SPI data buffer.

iDatSize[in]:

Type: **int**

Data size: 8 ~ 40 bit.

fUseWrLatencyRd[in]:

Type: **bool**

Use SDI(Write)-Latency-SDO(Read).

piBitLengBuf[in]:

Type: **int**

Write/Read/Latency length (Bits)

piBitLengBuf[0]: Write Length Bits

piBitLengBuf[1]: Read Length Bits

piBitLengBuf[2]: Latency Length Bits

iFrameGuardTime[in]:

Type: **int**

Frame guard time, unit: ns

iSlaveResp[in]:

Type: **int**

Set the slave response as Hi-Z (SET_HIZ) or High (NOT_SET_HIZ).

Return value

Return the address of DG if the function succeeded; -1 if the function failed.

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

Append a 2-Wire SPI packet.

Parameters

hDGPTl[in]:

Type: **HDGPTL**

The handle for SPI packet list.

pi64DatBuf[in]:

Type: **UINT64***

SPI data buffer.

iDatSize[in]:

Type: **int**

Data size: 8 ~ 40 bit.

fUseWrLatencyRd[in]:

Type: **bool**

Use SDI(Write)-Latency-SDO(Read).

piBitLengBuf[in]:

Type: **int**

Write/Read/Latency length (Bits)

piBitLengBuf[0]: Write Length Bits

piBitLengBuf[1]: Read Length Bits

piBitLengBuf[2]: Latency Length Bits

iFrameGuardTime[in]:

Type: **int**

Frame guard time, unit: ns

iSlaveResp[in]:

Type: **int**

Set the slave response as Hi-Z (SET_HIZ) or High (NOT_SET_HIZ).

Return value

Return the address of DG if the function succeeded; -1 if the function failed.

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

Append a SIPI packet.

Parameters

hDGPTl[in]:

Type: **HDGPTL**

The handle for SIPI packet list.

iType[in]:

Type: **int**

SEL_SIPI = 4 // 3Wire

SEL_2WIRE_SIPI = 5

iClockNum[in]:

Type: **int**

Set the clock number.

i64Dat[in]:

Type: **UINT64**

Set the SIPI data.

Return value

Return the address of DG if the function succeeded; -1 if the function failed.

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

Input a SPI data file.

Parameters

hDGPTl[in]:

Type: **HDGPTL**

The handle for SPI packet list.

pStrFile[in]:

Type: **char***

Set *.txt or *.bin file path.

nsecs[in]:

Type: **UINT32**

Idle time between the SPI packets, unit: ns

Return value

Return the address of DG if the function succeeded; -1 if the function failed.

DGADDR SIPI_InputFile(HDGPTL hDGPTl, char* pStrFile)

Input a SIPI data file.

Parameters

hDGPTl[in]:

Type: **HDGPTL**

The handle for SIPI packet list.

pStrFile[in]:

Type: **char***

Set *.txt file path.

Return value

Return the address of DG if the function succeeded; -1 if the function failed.