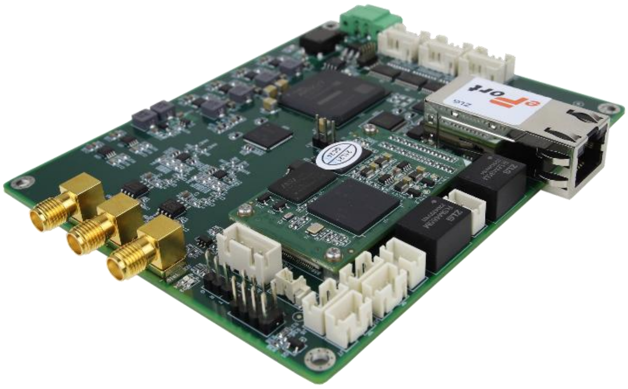
Dual-channel DTS high-speed data acquisition card

P/N:GY-DTS-x



Specification

- 12bits dual-channel simultaneous real-time sampling
- 100/200/250MSps sampling rate
- DC coupling, 50Ω input impedance
- ±2V input voltage range
- Support trigger input/output
- 100Mbps Ethernet port, TCP transmission protocol
- Support 32768 point acquisition per channel
- Built-in data averaging engine, maximum averaging times 65536

Overview

GY-DTS is a DTS data acquisition card with 100/200/250MSps sampling rate. It adopts 100Mbps network port for data transmission and uses TCP transmission protocol for stable and reliable data transmission. Built-in averaging function with high signal-to-noise ratio. Each channel supports 32768 point acquisition, and the maximum number of averaging can be up to 65536 times.

Power supply and consumption

Supply voltage: 5V
Power consumption: 4W (Max)

Temperature range

Operating temperature: -40~70℃

Storage temperature: -40~85℃

❖ Mechanical dimensions

120mm(L) x 100mm(W)

❖ Ordering Guide

PN	Description
GY-DTS-100	2-channel input, 100MSps
GY-DTS-200	2-channel input, 200MSps
GY-DTS-250	2-channel input, 250MSps

❖ Board Network Parameters

The capture card acts as a TCP Server with the default IP address of 192.168.1.100 and port number of 5000.

❖ Communication protocols

Data communication consists of 3 protocols, writing registers, reading registers and receiving data.

- Write register

It needs to send 8 consecutive bytes, the instruction structure is as follows. The example program has encapsulated the write instruction, the following table is for reference only.

BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5	BYTE6	BYTE7
Frame header		**Write Address，16BITs**		**Write data，32BITs**			
0x5A	**0xA5**	**ADDR_H**	**ADDR_L**	**DATA31_24**	**DAT23_16**	**DATA15_8**	**DATA7_0**

- Read register

1) Consecutive 8 bytes need to be sent, and the instruction structure is in the following table.

BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5	BYTE6	BYTE7
Frame header		**Write Address，16BITs**		**Invalid fields**			
0x69	**0x96**	**ADDR_H**	**ADDR_L**	**0x00**	**0x00**	**0x00**	**0x00**

2) Then the board will return 8 consecutive bytes of data, and the return frame instruction structure is in the following table.

BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5	BYTE6	BYTE7
Frame header		**Invalid fields**		**The data to be read**			
0x69	**0x96**	**0x00**	**0x00**	**DATA31_24**	**DAT23_16**	**DATA15_8**	**DATA7_0**

- Receive data command

This includes the frame header, the actual number of data bytes, and the actual data transmitted, as shown in the table below.

BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5	BYTE6	BYTE7
Frame header				Actual number of received data bytes			
0x5A	0xA5	0x69	0x96	ByteNum 31_24	ByteNum 23_16	ByteNum 15_8	ByteNum 7_0

Note: The actual number of received data bytes = ByteNum31_24* 16777216+ ByteNum23_16* 65536+ ByteNum15_8*256+ ByteNum7_0；

BYTE8	BYTE9	BYTE10	BYTE11	BYTE12	BYTE13	BYTE14	BYTE15……
Actual data received							
rx_byte0	rx_byte1	rx_byte2	rx_byte3	rx_byte4	rx_byte5	rx_byte6	rx_byte7……

Parsing of the received data by.

```
short *ch0_data, *ch1_data;

ch0_data [0]= rx_byte0*256+ rx_byte1;
ch1_data [0]= rx_byte2*256+ rx_byte3;
ch0_data [1]= rx_byte4*256+ rx_byte5;
ch1_data [1]= rx_byte6*256+ rx_byte7;
……
```

Received binary bit values up to 2047 and down to -2048.

The voltage value and binary bit value conversion relationship is: Voltage=BITS/2048.0.

Communication instruction set

```
int SetTrigDir(unsigned int trig_dir);
int SetTrigFreq(unsigned int trig_freq);
int SetTrigPusleWidth(unsigned int pusle_width_ns) ;
int SetPointNumPerScan(unsigned int point_num_per_scan);
int SetAverageTimes(unsigned int average_times);
int SetDOBit(unsigned short bit_en,unsigned short bit_status2set);
int GetDIBit(unsigned char *p_di_status);
int Start();
```

■ int SetTrigDir(unsigned int trig_dir)

/**/

Function Description:

Set the direction of the trigger signal

Function Parameters：

trig_dir： 0---Receive trigger signal；

1---Output trigger signal；

Function Return Value：

Success，0

Failure，-1

Function Code：

```
int SetTrigDir(unsigned int trig_dir)
{
        tx_buf[0]=0x5A;
        tx_buf[1]=0xA5;
        tx_buf[2]=0x00;
        tx_buf[3]=0x2C;
        tx_buf[4]=(trig_dir>>24)&0xFF;
        tx_buf[5]=(trig_dir>>16)&0xFF;
        tx_buf[6]=(trig_dir>>8)&0xFF;
        tx_buf[7]=(trig_dir>>0)&0xFF;
        //Send write instruction, 8 bytes, use the corresponding send function according to the user development environment
        if (ClientTCPWrite (tcp_server_handle, tx_buf,8, 1000) < 0) return -1;

        return 0;
}
```

/***/

- **int SetTrigFreq(unsigned int trig_freq)**

/***/

Function Description：

Set the frequency of the output trigger signal. If the board receives the trigger signal from the optical path, this function can be used without。

Function Parameters：

trig_freq ：Trigger frequency in Hz；

Function Return Value：

Success，0

Failure，-1

Function Code：

```
int SetTrigFreq(unsigned int trig_freq)
{
        tx_buf[0]=0x5A;
        tx_buf[1]=0xA5;
        tx_buf[2]=0x00;
        tx_buf[3]=0x24;
        tx_buf[4]=(trig_freq>>24)&0xFF;
        tx_buf[5]=(trig_freq>>16)&0xFF;
        tx_buf[6]=(trig_freq>>8)&0xFF;
        tx_buf[7]=(trig_freq>>0)&0xFF;

        if (ClientTCPWrite (tcp_server_handle, tx_buf,8, 1000) < 0) return -1;

        return 0;
}
/****************************************************************/
```

■ int SetTrigPusleWidth(unsigned int pusle_width_ns)

/**/

Function Description:

Set the high level pulse width of the output trigger signal. If the board receives the trigger signal from the optical path, this function can be used without。

Function Parameters:

pusle_width_ns ： Trigger signal high level pulse width in ns；

Function Return Value:

Success，0

Failure，-1

Function Code:

```
int SetTrigPusleWidth(unsigned int pusle_width_ns)
{
        tx_buf[0]=0x5A;
        tx_buf[1]=0xA5;
        tx_buf[2]=0x00;
        tx_buf[3]=0x28;
        tx_buf[4]=(pusle_width_ns>>24)&0xFF;
        tx_buf[5]=(pusle_width_ns>>16)&0xFF;
        tx_buf[6]=(pusle_width_ns>>8)&0xFF;
        tx_buf[7]=(pusle_width_ns>>0)&0xFF;

        if (ClientTCPWrite (tcp_server_handle, tx_buf,8, 1000) < 0) return -1;

        return 0;
}
/****************************************************************/
```

■ int SetPointNumPerScan(unsigned int point_num_per_scan)

/**/

Function Description:

Set the number of points per pulse acquisition。

Function Parameters:

point_num_per_scan：Number of points per pulse acquisition, max. 32768；

Function Return Value：

Success，0

Failure，-1

Function Code：

```
int SetPointNumPerScan(unsigned int point_num_per_scan)
{
    tx_buf[0]=0x5A;
    tx_buf[1]=0xA5;
    tx_buf[2]=0x00;
    tx_buf[3]=0x30;
    tx_buf[4]=(point_num_per_scan>>24)&0xFF;
    tx_buf[5]=(point_num_per_scan>>16)&0xFF;
    tx_buf[6]=(point_num_per_scan>>8)&0xFF;
    tx_buf[7]=(point_num_per_scan>>0)&0xFF;

    if (ClientTCPWrite (tcp_server_handle, tx_buf,8, 1000) < 0) return -1;

    return 0;
}
/**************************************************************/
```

■ **int SetAverageTimes(unsigned int average_times)**

```
/**************************************************************/
```

Function Description：

Set the average number of times

Function Parameters：

average_times：Average number of times, range 1~65536；

Function Return Value：

Success，0

Failure，-1

Function Code：

```
int SetAverageTimes(unsigned int average_times)
{
    if(average_times==0) return -1;

    tx_buf[0]=0x5A;
    tx_buf[1]=0xA5;
    tx_buf[2]=0x00;
    tx_buf[3]=0x34;
    tx_buf[4]=(average_times>>24)&0xFF;
    tx_buf[5]=(average_times>>16)&0xFF;
    tx_buf[6]=(average_times>>8)&0xFF;
    tx_buf[7]=(average_times>>0)&0xFF;

    if (ClientTCPWrite (tcp_server_handle, tx_buf,8, 1000) < 0) return -1;

    return 0;
}
/*****************************************************************/
```

- **int SetDOBit(unsigned short bit_en,unsigned short bit_status2set)**

```
/*****************************************************************/
```

Function Description：

Set DO output

Function Parameters：

bit_en： If the corresponding bit is 1, the bit can be updated, if it is 0, it will not be updated;

bit_status2set： High and low levels of CH15~CH0;

For example, if bit_en=0x8005, bit_status2set=0x3BD8, then DO0 will be updated to low, DO3 to high, and DO15 to low; other DO states will not be changed;

Function Return Value：

Success，0

Failure，-1

Function Code：

```
int SetDOBit(unsigned short bit_en,unsigned short bit_status2set)
{
    tx_buf[0]=0x5A;
    tx_buf[1]=0xA5;
    tx_buf[2]=0x00;
    tx_buf[3]=0x70;
    tx_buf[4]=0x00;
    tx_buf[5]=0x00;
    tx_buf[6]=(bit_en>>8)&0xFF;
    tx_buf[7]=(bit_en>>0)&0xFF;

    if (ClientTCPWrite (tcp_server_handle, tx_buf,8, 1000) < 0) return -1;

    tx_buf[0]=0x5A;
    tx_buf[1]=0xA5;
    tx_buf[2]=0x00;
    tx_buf[3]=0x71;
    tx_buf[4]=0x00;
    tx_buf[5]=0x00;
    tx_buf[6]=(bit_status2set>>8)&0xFF;
    tx_buf[7]=(bit_status2set>>0)&0xFF;
```

```
        if (ClientTCPWrite (tcp_server_handle, tx_buf,8, 1000) < 0) return -1;

        return 0;
}
/*****************************************************************/
```

- **int GetDIBit(unsigned char *p_di_status)**

/***/

Function Description：

Read DI status

Function Parameters：

p_di_status： Each bit corresponds to the low/high state of DI;

Function Return Value：

Success，0

Failure，-1

Function Code：

```
int GetDIBit(unsigned char *p_di_status)
{
        unsigned char rx_buf[8];

        tx_buf[0]=0x69;
        tx_buf[1]=0x96;
        tx_buf[2]=0x00;
        tx_buf[3]=0x80;
        tx_buf[4]=0x00;
        tx_buf[5]=0x00;
        tx_buf[6]=0x00;
        tx_buf[7]=0x00;

        if (ClientTCPWrite (tcp_server_handle, tx_buf,8, 1000) < 0) return -1;

        if (ClientTCPRead (tcp_server_handle, rx_buf,8, 1000) < 0) return -1;
        if((rx_buf[0]!=0x69)||(rx_buf[1]!=0x96)) return -1;

        *p_di_status=rx_buf[7];

        return 0;
}
/*****************************************************************/
```

- **int Start()**

/**/

Function Description：

Start collecting

Function Parameters：

无

Function Return Value：

Success，0

Failure，-1

Function Code：

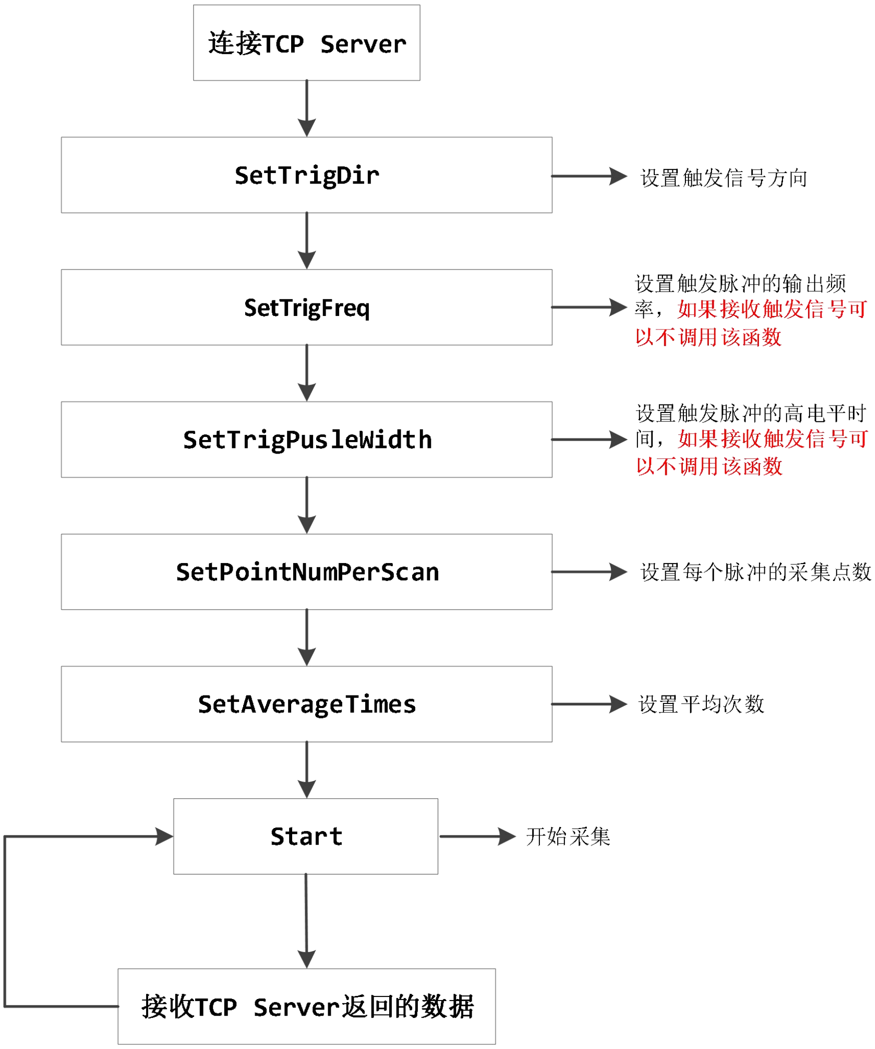
```
int Start()
{
        tx_buf[0]=0x5A;
        tx_buf[1]=0xA5;
        tx_buf[2]=0x00;
        tx_buf[3]=0x20;
        tx_buf[4]=0x00;
        tx_buf[5]=0x00;
        tx_buf[6]=0x00;
        tx_buf[7]=0x01;

        if (ClientTCPWrite (tcp_server_handle, tx_buf,8, 1000) < 0) return -1;

        return 0;
}
```

/**/

❖ Communication flow chart


连接TCP Server
SetTrigDir
设置触发信号方向
SetTrigFreq
设置触发脉冲的输出频率，如果接收触发信号可以不调用该函数
SetTrigPusleWidth
设置触发脉冲的高电平时间，如果接收触发信号可以不调用该函数
SetPointNumPerScan
设置每个脉冲的采集点数
SetAverageTimes
设置平均次数
Start
开始采集
接收TCP Server返回的数据