AST 2000

编程手册

（Rev 1.06）

Advanced Semiconductor Test Inc.

目　　录

1. CParam

1.1. CParam()

CParam()

Remarks

定义一个被测参数。

Example

CParam param;

1.2. SetDispName()

void SetDispName(const char *psz)

Parameters

psz

指向被测参数的显示名称的字符串指针。

Remarks

设置被测参数的显示名称，如果用户在定义了一个被测参数后但没有调用此函数设置显示名称，则该参数会被显示为“Nonnamed”。

Example

param.SetDispName(“Vio”);/*此被测参数显示为” Vio” */

1.3. SetUnitName()

void SetUnitName(const char *psz)

Parameters

psz

指向被测参数单位名的字符串指针。

Remarks

设置被测参数的单位。

Example

param.SetUnitName(“mV”);/*参数单位为” mV” */

1.4. SetMinLimit()

void SetMinLimit(double *minLimit*)

Parameters

minLimit

被测参数下限值。

Remarks

设置被测参数下限值。

Example

param.SetMinLimit(-0.5); /*参数下限为” -0.5mV” */

1.5. SetMaxLimit()

void SetMaxLimit(double *maxLimit*)

Parameters

maxLimit

被测参数上限值。

Remarks

设置被测参数上限值。

Example

param.SetMaxLimit(0.5);/*参数上限为” 0.5mV” */

1.6. SetDispFormat()

void SetDispFormat(const char **psz*)

Parameters

psz

指向被测参数显示格式的字符串指针。

Remarks

设置被测参数显示格式。

Example

```
param.SetDispFormat(“0.00”);/*显示到小数点后 2 位有效数字*/
param.SetDispFormat(“0.0”);/*显示到小数点后 1 位有效数字*/
```

1.7. SetSubUnitsCount()

void SetSubUnitsCount(int *count*)

Parameters

count

被测参数的子单元数。

Remarks

设置被测参数子单元数，当不调用此函数时，默认被被测参数子单元数为 1。

Example

```
/*假设被测器件为 LM324，其参数 Vio 分别对应四个运放单元*/
param.SetSubUnitsCount(4); /*设置此参数子单元数为 4*/
```

1.8. SetTestFunc()

void SetTestFunc(void* *testFunc*)

Parameters

testFunc

指向实现被测参数测试的函数指针。

Remarks

设置被测参数的测试函数。

Example

```
/* void test_Vio(); */
param.SetTestFunc(test_Vio); /*在调用此函数之前先声明 test_Vio 函数*/
...
/*
void test_Vio()
{
```

```
    //...
}
*/
```

1.9. SetTestCondition()

int SetTestCondition(int *nIndex*,
char **pzName*,
char **pzUnit*,
double *fValue*,
int *attribute*)

Parameters

nIndex

测试条件编号，最大值为 40。

pzName

测试条件显示名称。

pzUnit

测试条件单位。

fValue

测试条件初始值。

attribute

未用。

Return Values

调用正常返回 0，出错返回-1。

Remarks

设置被测参数的测试条件，此条件可以在主测试程序中进行修改，详见《AST2000 软件使用手册》。

Example

```
/*设置第 1 个测试条件 Vccp 初始值为 5V*/
    param.SetTestCondition(0, "Vccp", "V", 5);
```

```
/*设置第 2 个测试条件 Vccn  初始值为 0V*/
param.SetTestCondition(1, "Vccn", "V", 0);
```

1.10. GetTestConditionValue()

double GetTestConditionValue(BYTE *nIndex*)

Parameters

nIndex

测试条件编号。

Return Values

返回编号为 *nIndex* 的测试条件值。

Remarks

在实现被测参数测试的测试函数中调用此函数得到用户设置的测试条件值或默认值。

1.11. SetTestResult()

int SetTestResult(BYTE *SiteID*, BYTE *SubunitID*, double *fValue*)

Parameters

SiteID

工位号。

0：第 1 工位

1：第 2 工位

2：第 3 工位

3：第 4 工位

SubunitID

子单元号。

0：第 1 子单元

1：第 2 子单元

2：第 3 子单元

3：第 4 子单元

…　…

n：第 n+1 子单元

fValue

测试结果值。

Return Values

调用正常返回 0，出错返回-1，测试结果超限返回 1。

Remarks

当获得被测参数测试结果时，将测试结果赋给此参数。

Example

```
for(int i=0; i<4; i++ )
{
  /*第 1 工位 4 个测试子单元测试结果为 5mV*/
  param.SetTestResult(0, i, 5);

  /*第 2 工位 4 个测试子单元测试结果为-5mV*/
  param.SetTestResult(1, i, -5);

  /*第 3 工位 4 个测试子单元测试结果为 5mV*/
  param.SetTestResult(2, i, 5);

  /*第 4 工位 4 个测试子单元测试结果为 5mV*/
  param.SetTestResult(3, i, 5);
}
```

1.12. GetTestResult()

double GetTestResult(BYTE *SiteID*, BYTE *SubunitID*)

Parameters

SiteID

工位号。

0：第 1 工位

1：第 2 工位

2：第 3 工位

3：第 4 工位

SubunitID

子单元号。

0：第 1 子单元

1：第 2 子单元

2：第 3 子单元

3：第 4 子单元

… …

n：第 n+1 子单元

Return Values

返回指定工位指定子单元的测试结果。

Remarks

读取参数的测试结果数据。

Example

```
/*得到第 1 工位第 2 个测试子单元测试结果*/
double val = param.GetTestResult(0, 1);
```

2. DVI

2.1. DVI()

DVI(BYTE *channel*)

Parameters

channel

DVI 逻辑通道号

DVI_CH0：逻辑通道 0

DVI_CH1：逻辑通道 1

DVI_CH2：逻辑通道 2

DVI_CH3：逻辑通道 3

… …

DVI_CH31：逻辑通道 31

Remarks

定义一个 DVI，并指定使用的逻辑通道号。当系统配置为 8 块 DVI 板时单工位工作时最大逻辑通道号为 DVI_CH15，双工位并行工作时最大逻辑通道号为 DVI_CH7，四工位并行工作时最大逻辑通道号为 DVI_CH3。

Example

```
/*假设系统配置为 8 块 DVI，四工位并行工作，定义可使用的 DVI*/
DVI dvi0(DVI_CH0);      //使用逻辑通道 0
DVI dvi1(DVI_CH1) ;     //使用逻辑通道 1
DVI dvi2(DVI_CH2) ;     //使用逻辑通道 2
DVI dvi3(DVI_CH3) ;     //使用逻辑通道 3
```

2.2. Init()

void Init()

Remarks

初始化 DVI 通道。在使用 DVI 之前应当对每一个通道进行初始化，在一个器件测试结

束之后也应当初始化 DVI。

Example

```
dvi0.Init();        //初始化逻辑通道 0
dvi1.Init();        //初始化逻辑通道 1
dvi2.Init();        //初始化逻辑通道 2
dvi3.Init();        //初始化逻辑通道 3
```

2.3. Connect()

void Connect();

Remarks

接通 DVI 通道的输出继电器。

Example

```
dvi0.Connect ();
```

2.4. Disconnect()

void Disconnect();

Remarks

断开 DVI 通道的输出继电器。

Example

```
dvi0.Disconnect ();
```

2.5. SetModeFVMV()

int SetModeFVMV(BYTE *vForceRange*,
double *forceVolt*,
BYTE *iMeasRange*,
double *iClampUp*,
double *iClampDown*);

Parameters

vForceRange

DVI 电压量程，同时也为测压量程，取值为以下：

DVI_VRNG_1V：1V 量程档

DVI_VRNG_2V：2V 量程档

DVI_VRNG_5V：5V 量程档

DVI_VRNG_10V：10V 量程档

DVI_VRNG_20V：20V 量程档

DVI_VRNG_50V：50V 量程档

forceVolt

输出电压值，单位为 V。

iMeasRange

钳位电流量程。

DVI_IRNG_4UA： 4uA 量程档

DVI_IRNG_40UA： 40uA 量程档

DVI_IRNG_400UA：400uA 量程档

DVI_IRNG_4MA： 4mA 量程档

DVI_IRNG_40MA： 40mA 量程档

DVI_IRNG_400MA：400mA 量程档

iClampUp

输出电流钳位上限值，单位为 A。

iClampDown

输出电流钳位下限值，单位为 A。

Return Values

调用成功返回 0，出错返回-1。

Remarks

设置 DVI 为恒压测压模式。

Example

```
/*
设置 dvi2 为恒压测压模式，使用 50V 电压量程档，输出电压值为 30V，钳位电流 0-20mA
*/
```

dvi2.SetModeFVMV(DVI_VRNG_50V, 30, DVI_IRNG_40MA, 20e-3, 0);

2.6. SetModeFVMI()

int SetModeFVMI(BYTE *vForceRange*,
double *forceVoltage*,
BYTE *iMeasRange*,
double *iClampUp*,
double *iClampDown*)

Parameters

vForceRange

DVI 电压量程。

DVI_VRNG_1V：1V 量程档

DVI_VRNG_2V：2V 量程档

DVI_VRNG_5V：5V 量程档

DVI_VRNG_10V：10V 量程档

DVI_VRNG_20V：20V 量程档

DVI_VRNG_50V：50V 量程档

forceVolt

输出电压值，单位为 V。

iMeasRange

测流量程。

DVI_IRNG_4UA： 4uA 量程档

DVI_IRNG_40UA： 40uA 量程档

DVI_IRNG_400UA：400uA 量程档

DVI_IRNG_4MA： 4mA 量程档

DVI_IRNG_40MA： 40mA 量程档

DVI_IRNG_400MA：400mA 量程档

iClampUp

输出电流钳位上限值，单位为 A。

iClampDown

输出电流钳位下限值，单位为 A。

Return Values

调用成功返回 0，出错返回-1。

Remarks

设置 DVI 为恒压测流模式。

Example

```
/*
设置 dvi1 为恒压测流模式，使用 50V 电压量程档，输出电压值为 30V，测流量程为 40mA 档，钳位电流为 0-40mA
*/
    dvi1.SetModeFVMI(DVI_VRNG_50V, 30, DVI_IRNG_40MA, 40e-3, 0);
```

2.7. SetModeFIMI()

int SetModeFIMI(BYTE *iForceRange*,
double *forceCurrent*,
BYTE *vMeasRange*,
double *vClampUp*,
double *vClampDown*)

Parameters

iForceRange

DVI 输出电流量程。

DVI_IRNG_4UA： 4uA 量程档

DVI_IRNG_40UA： 40uA 量程档

DVI_IRNG_400UA：400uA 量程档

DVI_IRNG_4MA： 4mA 量程档

DVI_IRNG_40MA： 40mA 量程档

DVI_IRNG_400MA：400mA 量程档

forceCurrent

输出电流值，单位为 A。

vMeasRange

钳位电压量程。

DVI_VRNG_1V：1V 量程档

DVI_VRNG_2V：2V 量程档

DVI_VRNG_5V：5V 量程档

DVI_VRNG_10V：10V 量程档

DVI_VRNG_20V：20V 量程档

DVI_VRNG_50V：50V 量程档

vClampUp

输出电压钳位上限值，单位为 V。

vClampDown

输出电压钳位下限值，单位为 V。

Return Values

调用成功返回 0，出错返回-1。

Remarks

设置 DVI 为恒流测流模式。

Example

```
/*
设置 dvi1 为恒流测流模式，使用 40mA 电流量程档，输出电流值为 10mA，钳位电压 0-5V
*/
    dvi1.SetModeFIMI(DVI_IRNG_40MA, 10e-3, DVI_VRNG_10V, 5, 0);
/*
设置 dvi2 为恒流测流模式，使用 40mA 电流量程档，灌入电流值为 5mA，钳位电压-5-0V
*/
    dvi2.SetModeFIMI(DVI_IRNG_40MA,-5e-3, DVI_VRNG_10V, 0, -5);
```

2.8. SetModeFIMV()

int SetModeFIMV(BYTE *iForceRange*,

float *forceCurrent*,

BYTE *vMeasRange*,

float *vClampUp*,

float *vClampDown*)

Parameters

iForceRange

DVI 电流量程。

DVI_IRNG_4UA： 4uA 量程档

DVI_IRNG_40UA： 40uA 量程档

DVI_IRNG_400UA：400uA 量程档

DVI_IRNG_4MA： 4mA 量程档

DVI_IRNG_40MA： 40mA 量程档

DVI_IRNG_400MA：400mA 量程档

forceCurrent

输出电流值，单位为 A。

vMeasRange

DVI 电压量程。

DVI_VRNG_1V：1V 量程档

DVI_VRNG_2V：2V 量程档

DVI_VRNG_5V：5V 量程档

DVI_VRNG_10V：10V 量程档

DVI_VRNG_20V：20V 量程档

DVI_VRNG_50V：50V 量程档

vClampUp

输出电压钳位上限值，单位为 V。

vClampDown

输出电压钳位下限值，单位为 V。

Return Values

调用成功返回 0，出错返回-1。

Remarks

设置 DVI 为恒流测压模式。

Example

```
/*
设置 dvi1 为恒流测压模式，使用 40mA 电流量程档，输出电流值为 10mA，测压量程为 10V 档，钳位电压为 0-5V
*/
    dvi1.SetModeFIMV(DVI_IRNG_40MA, 10e-3f, DVI_VRNG_10V, 5, 0);
```

2.9. Enable()

void Enable();

Remarks

使设置有效，当用户修改 DVI 的参数后，调用此函数后 DVI 的输出设置才会有效。

Example

```
    dvi0.Enable();
```

2.10. Disable()

void Disable();

Remarks

调用此函数后 DVI 的恒压恒流输出归 0，建议在每个参数的测试子函数结束前将所用到的 VI 源 Disable，防止测试上一个参数时的 VI 源状态，在下个参数测试线路接过程中，对器件造成损坏，建议在切换电压和电流量程是调用此函数，避免带电切换。

Example

```
    dvi0.Disable();
```

2.11. Measure()

void Measure(float* *pAdcResult*, int *sampleTimes*)

Parameters

pAdcResult

指向保存测试结果的 4 维数组，分别对应四个工位的测试结果。

forceCurrent

输出电流值，单位为 A。

sampleTimes

AD 采样次数，默认为 10 次。

Remarks

读取 DVI 测压或测流值，结果保存到 *pAdcResult* 指向的数组中，默认单位为 V 或 A。

Example

```
/*
设置 dvi1 为恒流测压模式，使用 40mA 电流量程档，输出电流值为 10mA，测压量程为 10V 档，钳位电压为 0-5V
*/
    float val[4];
    dvi1.Connect();                    //接通输出继电器
    delay_ms(2)                        //延时 2ms
    dvi1.SetModeFIMV(DVI_IRNG_40MA, 10e-3f, DVI_VRNG_10V, 5, 0);
    dvi1.Enable();                     //输出有效
    dvi1.Measure(val);                 //采样 10 次
    //dvi1.Measure(val, 100);          //采样 100 次
    for(int i=0; i<4; i++ )
    {
    /*假设为四个工位第一子单元的测试结果，并将测试结果换算为 mV，赋给被测参数 param*/
    param.SetTestResult(i, 0, val[i] * 1000);
    }
    dvi1.Disable();
    dvi1.Disconnect();                 //断开 DVI 输出继电器，一次测试完成
```

2.12. SetRiseTime()

Int SetRiseTime(BYTE byRiseTime)

byRiseTime

DVI 的积分时间。

DVI_T_ 50US：50US 档

DVI_T_ 100US：100US 档

DVI_T_ 200US：200US 档

DVI_T_ 500US V：500US 档

Remarks

调用此函数设置 DVI 的积分时间，默认的积分时间为 200US。

3. PVI

3.1. PVI()

PVI(BYTE *channel*)

Parameters

channel

PVI 逻辑通道号

PVI_CH0：逻辑通道 0

PVI_CH1：逻辑通道 1

PVI_CH2：逻辑通道 2

PVI_CH3：逻辑通道 3

... ...

PVI_CH15：逻辑通道 15

Remarks

定义一个 PVI，并指定使用的逻辑通道号。当系统配置为 4 块 PVI 板时单工位工作时最大逻辑通道号为 PVI_CH7，双工位并行工作时最大逻辑通道号为 PVI_CH3，四工位并行工作时最大逻辑通道号为 PVI_CH1。

Example

```
/*假设系统配置为 4 块 PVI，四工位并行工作，定义可使用的 PVI*/
PVI pvi0(PVI_CH0);      //使用逻辑通道 0
PVI pvi1(PVI_CH1) ;     //使用逻辑通道 1
```

3.2. Init()

void Init()

Remarks

初始化 PVI 通道。在使用 PVI 之前应当对每一个通道进行初始化，在一个器件测试结束之后也应当初始化 PVI。

Example

pvi0.Init(); //初始化逻辑通道 0

pvi1.Init(); //初始化逻辑通道 1

3.3. Connect()

void Connect();

Remarks

接通 PVI 通道的输出继电器。

Example

pvi0.Connect ();

3.4. Disconnect()

void Disconnect();

Remarks

断开 PVI 通道的输出继电器。

Example

pvi0.Disconnect ();

3.5. SetModeFVMV()

int SetModeFVMV(BYTE *vForceRange*,
double *forceVolt*,
BYTE *iMeasRange*,
double *iClampUp*,
double *iClampDown*);

Parameters

vForceRange

PVI 输出电压量程。

PVI_VRNG_1V：1V 量程档

PVI_VRNG_2V：2V 量程档

PVI_VRNG_5V：5V 量程档

PVI_VRNG_10V：10V 量程档

PVI_VRNG_20V：20V 量程档

PVI_VRNG_50V：50V 量程档

forceVolt

输出电压值，单位为 V。

iMeasRange

钳位电流量程。

PVI_IRNG_100UA：100uA 量程档

PVI_IRNG_1mA： 1mA 量程档

PVI_IRNG_10mA：10mA 量程档

PVI_IRNG_100mA：100mA 量程档

PVI_IRNG_1A：1A 量程档

PVI_IRNG_10A：10A 量程档

iClampUp

钳位电流上限值，单位为 A。

iClampDown

钳位电流下限值，单位为 A。

Return Values

调用成功返回 0，出错返回-1。

Remarks

设置 PVI 为恒压测压模式。

Example

```
/*
设置 pvi2 为恒压测压模式，使用 50V 电压量程档，输出电压值为 30V，钳位电流 0-30mA
*/
    pvi2.SetModeFVMV(PVI_VRNG_50V, 30, PVI_IRNG_100V, 30e-3, 0);
```

3.6. SetModeFVMI()

int SetModeFVMI(BYTE *vForceRange*,

double *forceVoltage*,

BYTE *iMeasRange*,

double *iClampUp*,

double *iClampDown*)

Parameters

vForceRange

PVI 电压量程。

PVI_VRNG_1V：1V 量程档

PVI_VRNG_2V：2V 量程档

PVI_VRNG_5V：5V 量程档

PVI_VRNG_10V：10V 量程档

PVI_VRNG_20V：20V 量程档

PVI_VRNG_50V：50V 量程档

forceVolt

输出电压值，单位为 V。

iMeasRange

测流量程。

PVI_IRNG_100UA：100uA 量程档

PVI_IRNG_1mA：　1mA 量程档

PVI_IRNG_10mA：10mA 量程档

PVI_IRNG_100mA：100mA 量程档

PVI_IRNG_1A：1A 量程档

PVI_IRNG_10A：10A 量程档

iClampUp

输出电流钳位上限值，单位为 A。

i*ClampDown*

输出电流钳位下限值，单位为 A。

Return Values

调用成功返回 0，出错返回-1。

Remarks

设置 PVI 为恒压测流模式。

Example

/*

设置 pvi1 为恒压测流模式，使用 50V 电压量程档，输出电压值为 30V，测流量程为 100mA 档，钳位电流为 0-40mA

*/

pvi1.SetModeFVMI(PVI_VRNG_50V, 30, PVI_IRNG_100MA, 40e-3, 0);

3.7. SetModeFIMI()

int SetModeFIMI(BYTE *iForceRange*,
double *forceCurrent*,
BYTE *vMeasRange*,
double *vClampUp*,
double *vClampDown*)

Parameters

iForceRange

PVI 电流量程。

PVI_IRNG_100UA：100uA 量程档

PVI_IRNG_1mA： 1mA 量程档

PVI_IRNG_10mA：10mA 量程档

PVI_IRNG_100mA：100mA 量程档

PVI_IRNG_1A：1A 量程档

PVI_IRNG_10A：10A 量程档

forceCurrent

输出电流值，单位为 A。

vMeasRange

钳位电压量程。

PVI_VRNG_1V：1V 量程档

PVI_VRNG_2V：2V 量程档

PVI_VRNG_5V：5V 量程档

PVI_VRNG_10V：10V 量程档

PVI_VRNG_20V：20V 量程档

PVI_VRNG_50V：50V 量程档

vClampUp

输出电流钳位上限值，单位为 V。

vClampDown

输出电流钳位下限值，单位为 V。

Return Values

调用成功返回 0，出错返回-1。

Remarks

设置 PVI 为恒流测流模式。

Example

```
/*
设置 pvi1 为恒流测流模式，使用 100mA 电流量程档，输出电流值为 5mA，钳位电压 0-4V
*/
    pvi1.SetModeFIMI(PVI_IRNG_100MA, 5e-3, PVI_VRNG_10V, 4, 0);
```

3.8. SetModeFIMV()

int SetModeFIMI(BYTE *iForceRange*,
double *forceCurrent*,
BYTE *vMeasRange*,
double *vClampUp*,
double *vClampDown*)

Parameters

iForceRange

PVI 电流量程。

PVI_IRNG_100UA：100uA 量程档

PVI_IRNG_1mA： 1mA 量程档

PVI_IRNG_10mA：10mA 量程档

PVI_IRNG_100mA：100mA 量程档

PVI_IRNG_1A：1A 量程档

PVI_IRNG_10A：10A 量程档

forceCurrent

输出电流值，单位为 A。

vMeasRange

PVI 电压量程。

PVI_VRNG_1V：1V 量程档

PVI_VRNG_2V：2V 量程档

PVI_VRNG_5V：5V 量程档

PVI_VRNG_10V：10V 量程档

PVI_VRNG_20V：20V 量程档

PVI_VRNG_50V：50V 量程档

vClampUp

输出电压钳位上限值，单位为 V。

vClampDown

输出电压钳位下限值，单位为 V。

Return Values

调用成功返回 0，出错返回-1。

Remarks

设置 PVI 为恒流测压模式。

Example

```
/*
设置 pvi1 为恒流测压模式，使用 10mA 电流量程档，输出电流值为 5mA，测压量程为 10V
档，钳位电压为 0-5V
*/
```

```
pvi1.SetModeFIMV(PVI_IRNG_10MA, 5e-3, PVI_VRNG_10V, 5, 0);
```

3.9. Enable()

```
void Enable();
```

Remarks

使设置有效，当用户修改 PVI 的参数后，调用此函数后 PVI 的输出设置才会有效。

当 PVI 的输出电流量程为 10A 档时，Enable 后 PVI 会自动启动保护功能，当输出时间超过 10mS，PVI 会自动关闭输出。PVI 的电压/电流及钳位全部归零，再次调用 Enable 后恢复原先状态。

Example

```
float val[4];
/*如 PVI 输出 5A，首先设置输出为 0*/
pvi0.SetModeFIMV(PVI_IRNG_10A, 0, PVI_VRNG_10V, 5, 0);
/*此次 Enable 中会有量程继电器搭接时间，由于输出为 0，将不会启动保护功能*/
pvi0.Enable();
/*量程不变设置输出为 5A */
pvi0.SetModeFIMV(PVI_IRNG_10A, 5, PVI_VRNG_10V, 5, 0);
/*如果量程不变，此次 Enable 中不包括量程继电器搭接时间，开始启动保护功能*/
pvi0.Enable();
delay_ms(3);   /*延时时间或其它硬件动作时间，小于 10ms*/
pvi0.Measure(val,50); /*读取 PVI 测压结果*/
```

3.10. Disable()

```
void Disable();
```

Remarks

调用此函数后 PVI 的恒压恒流输出归 0，建议在每个参数的测试子函数结束前将所用到的 VI 源 Disable，防止测试上一个参数时的 VI 源状态，在下个参数测试线路搭接过程中，对器件造成损坏，尤其当前 PVI 前一个状态为较大电流/电压状态。

Example

```
pvi0.Disable();
```

3.11. Measure()

void Measure(float* *pAdcResult*, int *sampleTimes*)

Parameters

pAdcResult

指向保存测试结果的 4 维数组，分别对应四个工位的测试结果。

forceCurrent

输出电流值，单位为 A。

sampleTimes

AD 采样次数，默认为 10 次。

Remarks

读取 PVI 测压或测流值，结果保存到 *pAdcResult* 指向的数组中，默认单位为 V 或 A。

Example

```
/*
设置 pvi1 为恒流测压模式，使用 10mA 电流量程档，输出电流值为 5mA，测压量程为 10V
档，钳位电压为 0-5V
*/
    float val[4];
    pvi1.Connect();                    //接通输出继电器
    delay_ms(10)                       //延时 10ms
    pvi1.SetModeFIMV(PVI_IRNG_10MA, 50e-3, PVI_VRNG_10V, 5, 0);
    pvi1.Enable();                     //输出有效
    pvi1.Measure(val);                 //采样 10 次
    //pvi1.Measure(val, 100);          //采样 100 次
    for(int i=0; i<4; i++ )
    {
    /*假设为四个工位第一子单元的测试结果，并将测试结果换算为 mV，赋给被测参数
    param*/
```

```
param.SetTestResult(i, 0, val[i] * 1000);
}
pvi1.Disable();
pvi1.Disconnect();                    //断开继电器，一次测试完成
```

3.12. SetRiseTime()

Int SetRiseTime(BYTE byRiseTime)

byRiseTime

PVI 的积分时间。

PVI_T_ 200US：200US 档

PVI_T_ 500US V：500US 档

Remarks

调用此函数设置 PVI 的积分时间，默认的积分时间为 200US。

4. MVS

4.1. MVS()

MVS(BYTE *channel*)

Parameters

channel

MVS 逻辑通道号

MVS_CH0：逻辑通道 0

MVS_CH1：逻辑通道 1

MVS_CH2：逻辑通道 2

MVS_CH3：逻辑通道 3

... ...

MVS_CH15：逻辑通道 15

Remarks

定义一个 MVS，并指定使用的逻辑通道号。当系统配置为 1 块 MVS 板时单工位工作时最大逻辑通道号为 MVS_CH15，双工位并行工作时最大逻辑通道号为 MVS_CH7，四工位并行工作时最大逻辑通道号为 MVS_CH3。

Example

```
/*假设系统配置为 1 块 MVS，四工位并行工作，定义可使用的 MVS*/
MVS mvs0(MVS_CH0);        //使用逻辑通道 0
MVS mvs1(MVS_CH1) ;       //使用逻辑通道 1
MVS mvs2(MVS_CH2) ;       //使用逻辑通道 2
MVS mvs3(MVS_CH3) ;       //使用逻辑通道 3
```

4.2. Init()

void Init()

Remarks

初始化 MVS 通道。所有输入输出继电器断开，输出电压归 0。该函数对所有通道都有

效。

Example

mvs0.Init(); //初始化所有通道

4.3. Connect()

void Connect();

Remarks

接通 MVS 通道的输出继电器。

Example

mvs0.Connect ();

4.4. Disconnect()

void Disconnect();

Remarks

断开 MVS 通道的输出继电器。

Example

mvs0.Disconnect ();

4.5. SetVoltage()

int SetVoltage(double *fVoltage*)

Parameters

fVoltage

输出电压值，单位为 V。

Return Values

调用成功返回 0，出错返回非 0。

Remarks

设置 MVS 输出电压值为 *fVoltage*。

Example

mvs0. SetVoltage(0.7); /*0 通道输出 0.7V*/

```
mvs1. SetVoltage(0.7);      /*1 通道输出 0.7V*/
mvs2. SetVoltage(0.7);      /*2 通道输出 0.7V*/
mvs3. SetVoltage(0.7);      /*3 通道输出 0.7V*/
```

4.6. SetDriveMode()

int SetDriveMode(double *dig_h*, double *dig_l*)

Parameters

dig_h

高电平值，单位为 V。

dig_l

低电平值，单位为 V。

Return Values

调用成功返回 0，出错返回非 0。

Remarks

设置 MVS 为高低电压工作模式，其中输出高低电平分别为 *dig_h*，*dig_l*。此功能一般应用在运放压率测试中。注意，此功能函数只能用于偶数通道中。

Example

```
mvs0.SetDriveMode(5, 0);/*设置高低电平分别为 5V 0V*/
```

4.7. SetDriveOut()

int SetDriveOut(unsigned char driveLevel)

Parameters

driveLevel

输出平值。

LEVEL_H： 高电平

LEVEL_L： 低电平

Return Values

调用成功返回 0，出错返回非 0。

Remarks

设置 MVS 在高低电压工作模式的输出，其中 LEVEL_H、LEVEL_L 输出电平分别为 *dig_h*，*dig_l*。此功能一般应用在运放压率测试中。注意，此功能函数只能用于偶数通道中。

Example

```
mvs0.SetDriveOut(LEVEL_H);/*设置 MVS 输出高电平*/
```

5. CBIT

5.1. CBIT()

CBIT()

Remark

创建一个 CBIT。

Example

CBIT cbit;

5.2. Init()

void Init()

Remarks

初始化 CBIT，所有 CBIT 输出状态为 OFF，CBIT 所有输出位为高电平状态。

Example

cbit.Init();

5.3. SetCBITOff()

BYTE SetCBITOff(BYTE *nIndex*)

Parameters

nIndex

要设置的 CBIT 的位。

Return Values

调用成功返回非 0，失败返回 0。

Remarks

使 *nIndex* 指向的位输出高电平。

当系统配置为单工位时 *nIndex* 取值范围为 0-127，分别对应 CBIT 的 128 位；当配置为双工位并行工位时 CBIT 被分为两个单元，*nIndex* 取值范围为 0-63，分别对应 CBIT 两个单元的 64 位；当配置为四工位并行工作时，CBIT 被分为四个单元，*nIndex* 取值范

围为 0-31，分别对应 CBIT 四个单元的 32 位。

Example

```
/*设 CBIT 的第 3 位控制继电器 k1*/
BYTE    k1 = 2;
cbit.SetCBITOff(k1);                //断开 k1 继电器
```

5.4. SetCBITOn()

BYTE SetCBITOn(BYTE *nIndex*)

Parameters

nIndex

要设置的 CBIT 的位。

Return Values

调用成功返回非 0，失败返回 0。

Remarks

使 *nIndex* 指向的位输出低电平。

Example

```
/*设 CBIT 的第 3 位控制继电器 k1*/
BYTE    k1 = 2;
cbit.SetCBITOn(k1);                //接通 k1 继电器
```

5.5. SetCBIT()

void SetCBIT(BYTE* *pCtrlByte*)

Parameters

pCtrlByte

指向保存 CBIT 状态位的数组。

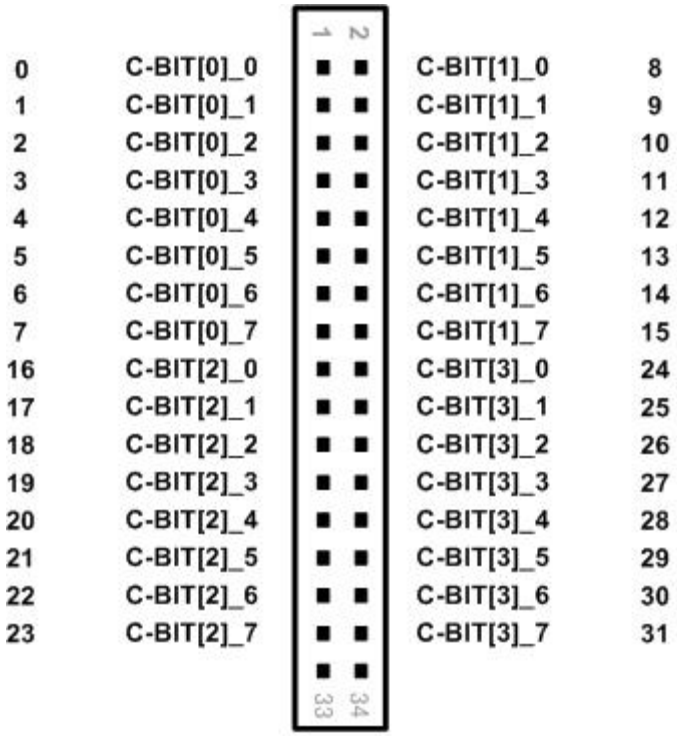
Remarks

pCtrlByte 指向的数组中每个字节的 8 位分别对应 CBIT 的 8 位，为 1 则对应的位输出低电平，继电器接通，为 0 则对应的位输出高电平，继电器断开。

Example

/*四工位配置*/

下图为典型四工位配置中，每一工位输出插座对应的 CBIT 位：



C-BIT插座引脚图

```
BYTE cbitdata[4];
/*                                site1     site2   site3   site4*/
cbitdata[0] = 0x00;//对应 CBIT 的 0-7，     32-39，64-71，71-103
cbitdata[1] = 0x00;//对应 CBIT 的 8-15，    40-47，72-79，80-111
cbitdata[2] = 0x00;//对应 CBIT 的 16-23，   48-55，80-87，88-119
cbitdata[3] = 0x00;//对应 CBIT 的 24-31，   56-63，88-95，96-127
cbit.SetCBIT(cbitdata);//四个工位对应的继电器全部断开
for(int i=0; i<4; i++ )
{
    cbitdata[i] = 0xFF;          //字节中所有位为 1
}
cbit.SetCBIT(cbitdata);//四个工位对应的继电器全部接通
```

/*单工位配置*/

```
BYTE cbitdata[16];
for(int i=0; i<16; i++ )
{
    cbitdata[i] = 0xFF;
}
cbit.SetCBIT(cbitdata);//CBIT 所有位输出低电平，对应继电器全部接通
```

5.6. SetOn()

BYTE SetOn(int *k1*, ...)

Parameters

k1

对应的 CBIT 位。

Return Values

调用成功返回 0，出错返回非 0。

Remarks

当函数执行时，*k1*, ...指向的 CBIT 的位输出低电平，接通对应的继电器，参数中没有指定的 CBIT 位输出高电平，对应的继电器断开。

参数个数为任意多，但最后一个参数为必须为-1，作为结束参数标志。

Example

/*四工位配置*/

假设用户 DUT 板上有 4 个继电器 K1、K2、K3、K4，分别由 CBIT 插座上的 C-BIT[0]_0，C-BIT[1]_0，C-BIT[2]_0，C-BIT[3]_1 控制，即对应的 CBIT 位分别为 0、 8、 16、25，参看上图。

```
CBIT cbit;
BYTE K1 = 0;
BYTE K2 = 8;
BYTE K3 = 16;
BYTE K4 = 25;
cbit.SetOn(K1, K2, -1);      /*K1、K2 闭合 K3、K4 断开*/
```

```
cbit.SetOn(K1, K2, K3, K4, -1);      /*K1、K2、K3、K4 闭合*/
cbit.SetOn(-1);      /*K1、K2、K3、K4 全断开*/
```

6. TMU

6.1. TMU()

TMU()

Remark

创建一个 TMU。

Example

TMU tmu;

6.2. Init()

void Init()

Remarks

初始化 TMU，断开所有输入输出继电器，计数器清零，比较器参考电压设为零晶体振荡器停振。

Example

tmu.Init();

6.3. MeasDutyCycle()

int MeasDutyCycle(unsigned int *nGateTime*, BYTE *channel*, float *fValue*[4])

Parameters

nGateTime

测试占空比的闸门时间，单位为 us，最小值为 1000，最大为 6000，闸门时间越长测试精度越高，但测试时间也会越长。

channel

输入通道。

TMU_CHANNEL_A：通道 A

TMU_CHANNEL_B：通道 B

fValue

指向保存测试结果的 4 维数组。

Return Values

调用成功返回 0，出错返回非 0。

Remarks

占空比测试，实际测试值为低电平占空比值，如果被测信号为高电平占空比，则要用 100 减去测试值。

Example

```
float val[4];
tmu.MeasDutyCycle(TMU_CHANNEL_A, val);
```

6.4. MeasFreq()

int MeasFreq(unsigned int *nGateTime*, BYTE *channel*, float *fValue*[4])

Parameters

nGateTime

测试频率的闸门时间，单位为 us，最小值为 1000，最大为 6000，闸门时间越长测试精度越高，但测试时间也会越长。

channel

信号输入通道。

TMU_CHANNEL_A：通道 A

TMU_CHANNEL_B：通道 B

fValue

指向保存测试结果的 4 维数组指针，输出结果单位为 KHz。

Return Values

调用成功返回 0，出错返回非 0。

Remarks

完成周期信号频率或周期的测量。

Example

```
float val[4];
tmu.MeasFreq(2000, TMU_CHANNEL_A, val);
```

6.5. MeasSlewRate()

int MeasSlewRate(

float *startLevel*,

float *stopLevel*,

BYTE *vRange*,

BYTE *tRange*,

BYTE *tInMode*,

double *srValue*[4]

)

Parameters

startLevel

起始电压。

stopLevel

终止电压。

vRange

测压量程。

RANGE_5V：5V 量程

RANGE_15V：15V 量程

tRange

测时间量程。

TIME_4US：4us 量程

TIME_40US：40us 量程

tInMode

输入信号模式。

POS_SLOPE：正向（压摆率）

NEG_SLOPE：负向（压摆率）

NEG_PULSE：负脉冲

POS_PULSE：正脉冲

srValue

指向保存测试结果的 4 维数组。

Return Values

调用成功返回 0，出错返回非 0。。

Remarks

测量运放的压摆率。

Example

```
float sr[4];
/*正向压摆率*/
tmu.MeasSlewRate(1.5, 4, RANGE_5V, TIME_40US ,POS_SLOPE, sr);
/*负向压摆率*/
tmu.MeasSlewRate(4, 1, RANGE_5V, TIME_40US ,NEG_SLOPE, sr);
```

6.6. SetInChannel()

int SetInChannel(BYTE channel)

Parameters

channel

信号输入通道。

TMU_CHANNEL_A：通道 A

TMU_CHANNEL_B：通道 B

Return Values

调用成功返回 0，出错返回非 0。

Remarks

完成周期信号频率或周期的测量的测量输入通道的选择，时间测量时，两个输入通道短接，故不需要选择通道。

6.7. SetFreqRange()

int SetFreqRange(BYTE fRange , unsigned int gateCount)

Parameters

fRange

频率信号量程，取值为以下：

RANGE_10MHz：10MHzV 量程档

RANGE_1MHz： 1MHz 量程档

RANGE_100KHz：100KHz 量程档

gateCount

闸门个数，范围为 1000 到 60000，如果给值小于 1000，则设置为 1000；给值大于 60000，则设置为 60000。

Return Values

调用成功返回 0，出错返回非 0。

Remarks

设置测量频率和占空比的量程选择，可以不写参数，默认 fRange 为，gateCount 为 5000。

6.8. SetTimeRange()

int SetTimeRange(BYTE timeRange, BYTE voltRange)

Parameters

timeRange

时间测量量程，取值为以下：

TIME_4US：4us 量程档

TIME_40US：40us 量程档

voltRange

时间测量的设置的电压量程，取值为以下：

RANGE_5V：5V 量程档

RANGE_15V：15V 量程档

Return Values

调用成功返回 0，出错返回非 0。

Remarks

设置测量时间时的时间测量量程和测量电压量程。

6.9. SetTimeSlope()

int SetTimeSlope(BYTE slopeType, float startLevel, float stopLevel)

Parameters

slopeType

时间测量触发电平模式，取值为以下：

NEG_SLOPE：下降沿

POS_SLOPE：上升沿

NEG_PULSE：负脉冲

POS_PULSE：正脉冲

startLevel

开始触发电平值，范围为-5V 到+5V

stopLevel

结束触发电平值，范围为-5V 到+5V

Return Values

调用成功返回 0，出错返回非 0。

Remarks

设置测量时间的触发电平模式和触发电平值。

6.10. ReadyForTest()

int ReadyForTest(BYTE measureType)

Parameters

measureType

信号测量类型，可以取以下几种类型：

FREQUENCY：频率测量

DUTYFACTOR：占空比测量

TIME：时间测量

Return Values

调用成功返回 0，出错返回非 0。

Remarks

在进行相应类型的测试前调用此函数，选择测试类型，进行测试准备，需要注意的是，在调用测试准备函数之前，需要先调用选择通道的函数和设置量程的函数，其中测量时间的时候还需要选择触发电平模式。

6.11. MeasDuty()

int MeasDuty(BYTE dutyType, float ratioValue[4])

Parameters

dutyType

占空比测量类型，可以取以下几种类型：

DUTY_ON_MIN：测量小占空比信号（被测信号反相）

DUTY_ON_MAX：测量大占空比信号

ratioValue

指向保存测试结果的 4 维数组。

Return Values

调用成功返回 0，出错返回非 0。

Remarks

测量占空比信号的测试函数。

Example

下面给出一个完整的测量占空比的 TMU 的使用例子，请注意各个函数的调用顺序：

```
TMU tmu;
tmu.Init();//TMU 初始化
tmu.SetInChannel(TMU_CHANNEL_A);//用通道 A 测占空比
tmu.SetFreqRange(RANGE_10MHz);//选择 10MHzV 量程档
tmu.ReadyForTest(DUTYFACTOR);//选择测占空比模式，准备测试
float result[4];
tmu. MeasDuty(result);//测量占空比，并保存结果到 result 数组中
```

6.12. MeasFrequence()

int MeasFrequence(float frequencyValue[4])

Parameters

frequencyValue

指向保存测试结果的 4 维数组，默认单位为 KHz。

Return Values

调用成功返回 0，出错返回非 0。

Remarks

测量频率信号的测试函数。

Example

下面给出一个完整的测量频率的 TMU 的使用例子，请注意各个函数的调用顺序：

```
TMU tmu;
```

tmu.Init();//TMU 初始化

tmu.SetInChannel(TMU_CHANNEL_A);//用通道 A 测频

tmu.SetFreqRange(RANGE_10MHz);//选择 10MHzV 量程档

tmu.ReadyForTest(FREQUENCY);//选择测频模式，准备测试

float result[4];

tmu.MeasFrequence(result);//测量频率，并保存结果到 result 数组中

6.13. MeasTime()

int MeasTime(float timeValue[4])

Parameters

timeValue

指向保存测试结果的 4 维数组，默认输出单位为 us。

Return Values

调用成功返回 0，出错返回非 0。

Remarks

测量时间信号的测试函数。

Example

下面给出一个完整的测量时间的 TMU 的使用例子，请注意各个函数的调用顺序：

TMU tmu;

tmu.Init();//TMU 初始化

tmu.SetTimeRange(TIME_4US, RANGE_15V);//选择 4us 时间量程档，15V 电压量程档

tmu. SetTimeSlope (POS_SLOPE, 0.2, 4.5);//选择测量上升沿，开始电平值为 0.2V，结束电平值为 4.5V

tmu.ReadyForTest(TIME);//选择测时间模式，准备测试

float result[4];

tmu.MeasTime (result);//测量频率，并保存结果到 result 数组中

7. PVM

7.1. PVM()

PVM()

Remark

创建一个 PVM。

Example

PVM pvm;

7.2. Init()

void Init()

Remarks

初始化 PVM 所有控制字设置为默认值，所有 DA 输出 0V。

Example

pvm.Init();

7.3. SetInputMode()

int SetInputMode(BYTE *byInMode*)

Parameters

byInMode

IN_MODE_DIFF：差分模式，用于测量背板总线上的信号。

IN_MODE_SE：单端模式，用于直接测量 DUT 板上的信号。

Return Values

当前输入模式，错误则返回-1。

Remarks

切换输入模式。

Example

```
pvm.SetInputMode(IN_MODE_SE);
```

7.4. Measure()

int Measure(float * *pBuf*, WORD *samples*);

Parameters

pBuf

指向保存测试结果的 4 维数组指针，分别对应四个工位的数据。

samples

AD 采样次数，默认为 10 次。

Return Values

正确返回 0，如果超时返回非 0。

Remarks

直接用 PVM 进行测试时，调用此函数将测试结果保存在 *pBuf* 指向的数组中。在第一次调用 PVM 的测试函数前，应首先调用 SetInputMode()函数将输入模式设置为单端模式，即 SetInputMode(IN_MODE_SE)。

Example

```
float adresult[4];
pvm.SetInputMode(IN_MODE_SE);
pvm.Measure(adresult);
```

7.5. EnableCalibration()

BOOL EnableCalibration(BOOL *bCal*)

Parameters

bCal

TRUE：使用校正后的数据进行测试。

FALSE：不使用校正后的数据进行测试。

Return Values

正常调用返回 1，如果不存在校正后的数据，或校正后的数据文件格式被破坏则返回 0。此时测试不采用校正后的数据。

Remarks

此调用一次，对所有 PVM 通道都有效，默认状态为不使用校正后的数据。

Example

```
pvm. EnableCalibration(TRUE);       //使用校正后的数据进行测试
...
pvm. EnableCalibration(FALSE);      //不使用校正后的数据
```

8. OVI

8.1. OVI()

OVI(BYTE *channel*)

Parameters

channel

OVI 逻辑通道号

OVI_CH0：逻辑通道 0

OVI_CH1：逻辑通道 1

OVI_CH2：逻辑通道 2

OVI_CH3：逻辑通道 3

… …

OVI_CH63：逻辑通道 63

Remarks

定义一个 OVI，并指定使用的逻辑通道号。当系统配置为 8 块（最多）OVI 板时单工位工作时最大逻辑通道号为 OVI_CH63，双工位并行工作时最大逻辑通道号为 OVI_CH31，四工位并行工作时最大逻辑通道号为 OVI_CH15。

Example

```
/*假设系统配置为 2 块 OVI（16 个物理通道），四工位并行工作，定义可使用的 OVI*/
OVI ovi0(OVI_CH0);      //使用逻辑通道 0
OVI ovi1(OVI_CH1) ;     //使用逻辑通道 1
OVI ovi2(OVI_CH2) ;     //使用逻辑通道 2
OVI ovi3(OVI_CH3) ;     //使用逻辑通道 3
```

8.2. Init()

void Init()

Remarks

初始化 OVI 通道。在使用 OVI 之前应当对每一个通道进行初始化，在一个器件测试结束之后也应当初始化 OVI。

Example

```
ovi0.Init();        //初始化逻辑通道 0
ovi1.Init();        //初始化逻辑通道 1
ovi2.Init();        //初始化逻辑通道 2
ovi3.Init();        //初始化逻辑通道 3
```

8.3. Connect()

void Connect();

Remarks

接通 OVI 通道的输出继电器。

Example

```
ovi0.Connect ();
```

8.4. Disconnect()

void Disconnect();

Remarks

断开 OVI 通道的输出继电器。

Example

```
ovi0.Disconnect ();
```

8.5. SetModeFVMV()

int SetModeFVMV(BYTE *vRange*,
double *vValue*,
BYTE *iClampRange*,
double *iClampValue*);

Parameters

vRange

OVI 电压量程，同时也为测压量程，取值为以下：

OVI_VRNG_2V：2V 量程档

OVI_VRNG_5V：5V 量程档

OVI_VRNG_10V：10V 量程档

OVI_VRNG_20V：20V 量程档

vValue

输出电压值，单位为 V。

iClampRange

钳位电流量程。

OVI_IRNG_4UA： 4uA 量程档

OVI_IRNG_40UA： 40uA 量程档

OVI_IRNG_400UA：400uA 量程档

OVI_IRNG_4MA： 4mA 量程档

OVI_IRNG_40MA： 40mA 量程档

iClampValue

输出电流钳位值，单位为 A（安）。与 DVI 相对应的钳位上限值为 iClampValue 的绝对值，下限值为 iClampValue 的绝对值加负号。如果不写设置值，默认为当前量程的最大值。

Return Values

调用成功返回 0，出错返回-1。

Remarks

设置 OVI 为恒压测压模式。

Example

```
/*
设置 ovi2 为恒压测压模式，使用 20V 电压量程档，输出电压值为 15V，钳位电流 20mA
*/
    ovi2.SetModeFVMV(OVI_VRNG_20V, 15, OVI_IRNG_40MA, 20e-3);
```

8.6. SetModeFVMI()

int SetModeFVMI(BYTE *vRange*,

double *vValue*,

BYTE *iClampRange*,

double *iClampValue*);

Parameters

vRange

OVI 电压量程，同时也为测压量程，取值为以下：

OVI_VRNG_2V：2V 量程档

OVI_VRNG_5V：5V 量程档

OVI_VRNG_10V：10V 量程档

OVI_VRNG_20V：20V 量程档

vValue

输出电压值，单位为 V。

iClampRange

钳位电流量程。

OVI_IRNG_4UA： 4uA 量程档

OVI_IRNG_40UA： 40uA 量程档

OVI_IRNG_400UA：400uA 量程档

OVI_IRNG_4MA： 4mA 量程档

OVI_IRNG_40MA： 40mA 量程档

iClampValue

输出电流钳位值，单位为 A。与 DVI 相对应的嵌位上限值为 iClampValue 的绝对值，下限值为 iClampValue 的绝对值取负号。如果不写设置值，默认为当前量程的最大值。

Return Values

调用成功返回 0，出错返回-1。

Remarks

设置 OVI 为恒压测流模式。

Example

```
/*
设置 ovi2 为恒压测流模式，使用 20V 电压量程档，输出电压值为 15V，钳位电流 20mA(代表嵌位上限值为 20MA，下限值为-20MA)
*/
    ovi2.SetModeFVMV(OVI_VRNG_20V, 15, OVI_IRNG_40MA, 20e-3);
```

8.7. SetModeFIMI()

int SetModeFIMI(BYTE *iRange*,
double *iValue*,
BYTE *vClampRange*,
double *vClampValue*)

Parameters

iRange

OVI 输出电流量程。

OVI_IRNG_4UA： 4uA 量程档

OVI_IRNG_40UA： 40uA 量程档

OVI_IRNG_400UA：400uA 量程档

OVI_IRNG_4MA： 4mA 量程档

OVI_IRNG_40MA： 40mA 量程档

iValue

输出电流值，单位为 A。

vClampRange

钳位电压量程。

OVI_VRNG_1V：1V 量程档

OVI_VRNG_2V：2V 量程档

OVI_VRNG_5V：5V 量程档

OVI_VRNG_10V：10V 量程档

OVI_VRNG_20V：20V 量程档

vClampValue

输出电压钳位值，单位为 V。OVI 的恒流模式是靠嵌位电压来设置的，根据负载的大小，将嵌位电压设置为大于负载大小与电流乘积的值时，才会出现恒流模式。如果不写设置值，默认为当前电压量程的最大值。

Return Values

调用成功返回 0，出错返回-1。

Remarks

设置 OVI 为恒流测流模式。

Example

```
/*
设置 ovi1 为恒流测流模式，使用 40mA 电流量程档，输出电流值为 4mA，如果负载在 1k 左右，此时设置钳位电压 5V 可以实现恒流模式
*/
    ovi1.SetModeFIMI(OVI_IRNG_40MA, 4e-3, OVI_VRNG_10V, 5);
/*
```

8.8. SetModeFIMV()

int SetModeFIMV(BYTE *iRange*,
double *iValue*,
BYTE *vClampRange*,
double *vClampValue*)

Parameters

iRange

OVI 输出电流量程。

OVI_IRNG_4UA： 4uA 量程档

OVI_IRNG_40UA： 40uA 量程档

OVI_IRNG_400UA：400uA 量程档

OVI_IRNG_4MA： 4mA 量程档

OVI_IRNG_40MA： 40mA 量程档

iValue

输出电流值，单位为 A。

vClampRange

钳位电压量程。

OVI_VRNG_1V：1V 量程档

OVI_VRNG_2V：2V 量程档

OVI_VRNG_5V：5V 量程档

OVI_VRNG_10V：10V 量程档

OVI_VRNG_20V：20V 量程档

vClampValue

输出电压钳位值，单位为 V。OVI 的恒流模式是靠嵌位电压来设置的，根据负载的大小，将嵌位电压设置为大于负载电阻大小与电流乘积的值时，才会出现恒流模式。如果不写设置值，默认为当前电压量程的最大值。

Return Values

调用成功返回 0，出错返回-1。

Remarks

设置 OVI 为恒流测流模式。

Example

```
/*
设置 ovi1 为恒流测压模式，使用 40mA 电流量程档，输出电流值为 4mA，如果负载在 1k 左右，此时设置钳位电压 5V 可以实现恒流模式
*/
    ovi1.SetModeFIMI(OVI_IRNG_40MA, 4e-3, OVI_VRNG_10V, 5);
```

8.9. Enable()

```
void Enable();
```

Remarks

使设置有效，当用户修改 OVI 的参数后，调用此函数后 OVI 的输出设置才会有效。

Example

```
    ovi0.Enable();
```

8.10. Measure()

void Measure(float* *pAdcResult*, int *sampleTimes*)

Parameters

pAdcResult

指向保存测试结果的 4 维数组，分别对应四个工位的测试结果。

sampleTimes

AD 采样次数，默认为 10 次。

Remarks

读取 OVI 测压或测流值，结果保存到 *pAdcResult* 指向的数组中，默认单位为 V 或 A。

Example

```
/*
设置 ovi1 为恒流测压模式，使用 40mA 电流量程档，输出电流值为 10mA，测压量程为
10V 档，钳位电压为 5V
*/
    float val[4];
    ovi1.Connect();                    //接通输出继电器
    ovi1.SetModeFIMV(OVI_IRNG_40MA, 10e-3f, OVI_VRNG_10V, 5);
    ovi1.Enable();                     //输出有效
    delay_ms(2);                       //延时 2ms
    ovi1.Measure(val);                 //采样 10 次
    //ovi1.Measure(val, 100);          //采样 100 次
    for(int i=0; i<4; i++ )
    {
    /*假设为四个工位第一子单元的测试结果，并将测试结果换算为 mV，赋给被测参数
    param*/
    param.SetTestResult(i, 0, val[i] * 1000);
    }
    ovi1.Disconnect();                 //断开 OVI 输出继电器，一次测试完成
```

9. QTMU

9.1. QTMU()

QTMU(BYTE *channel*)

Parameters

channel

QTMU 逻辑通道号。

QTMU_CH0：逻辑通道 0。

QTMU_CH1：逻辑通道 1。

QTMU_CH2：逻辑通道 2。

QTMU_CH3：逻辑通道 3。

... ...

QTMU_CH15：逻辑通道 15。

Remarks

定义一个 QTMU，并指定使用的逻辑通道号。当系统配置为 4 块 QTMU 板时单工位工作时最大逻辑通道号为 QTMU_CH15，双工位并行工作时最大逻辑通道号为 QTMU_CH7，四工位并行工作时最大逻辑通道号为 QTMU_CH3。

Example

```
/*假设系统配置为 4 块 QTMU，四工位并行工作，定义可使用的 QTMU*/
QTMU qtmu0(QTMU_CH0);        //使用逻辑通道 0
QTMU qtmu1(QTMU _CH1) ;      //使用逻辑通道 1
QTMU qtmu2(QTMU_CH2) ;       //使用逻辑通道 2
QTMU qtmu3(QTMU_CH3) ;       //使用逻辑通道 3
```

9.2. Init()

void Init()

Remarks

初始化 QTMU 通道。在使用 QTMU 之前应当对每一个通道进行初始化，在一个器件测

试结束之后也应当初始化 QTMU。

Example

```
qtmu0.Init();      //初始化逻辑通道 0
qtmu1.Init();      //初始化逻辑通道 1
qtmu2.Init();      //初始化逻辑通道 2
qtmu3.Init();      //初始化逻辑通道 3
```

9.3. Connect()

void Connect();

Remarks

接通 QTMU 通道的输入继电器。

Example

```
qtmu0.Connect();
```

9.4. Disconnect()

void Disconnect();

Remarks

断开 QTMU 通道的输入继电器。

Example

```
qtmu0.Disconnect();
```

9.5. SetInSource()

int SetInSourse(BYTE sourceSel)

Parameters

sourceSel

待测信号源选择，默认值为 QTMU_SINGLE_SOURCE：

QTMU_SINGLE_SOURCE： 单信号源触发开始和结束信号进行测量。

QTMU_DUAL_SOURCE： 双信号源分别触发开始和结束信号进行测量。

Remarks

选择 QTMU 信号源。

Example

```
/*
qtmu0 选择单信号源触发开始和结束信号。
*/
    qtmu0.SetInSource(QTMU_SINGLE_SOURCE);
```

9.6. SetStartInput()

```
int SetStartInput(BYTE InputZ,
                  BYTE VRange,
                  BYTE filterMode);
```

Parameters

InputZ

设置 QTMU 输入阻抗，默认值为 1Mohm 输入阻抗

QTMU_IMPEDANCE_50： 50ohm 输入阻抗。

QTMU_IMPEDANCE_1M： 1Mohm 输入阻抗。

VRange

电压量程，默认值为 25V 量程

QTMU_VRNG_5V： 5V 量程。

QTMU_VRNG_25V： 25V 量程。

filterMode

设置输入滤波器，默认值为直通通路，不使用滤波器：

QTMU_FILTER_PASS： 直通通路，不使用滤波器。

QTMU_FILTER_100KHz： 100KHz 低通滤波器。

QTMU_FILTER_1MHz： 1MHz 低通滤波器。

QTMU_FILTER_10MHz： 10MHz 低通滤波器。

Remarks

设置 QTMU Start 信号输入通道模式。

Example

```
/*
设置 qtmu0 Start 信号输入阻抗为 1Mohm，电压量程为 25V，不使用滤波器。
*/
    qtmu0.SetStartInput(QTMU_IMPEDANCE_1M,
                        QTMU_VRNG_25V,
                        QTMU_FILTER_PASS);
```

9.7. SetStopInput()

```
int SetStopInput(BYTE InputZ,
                 BYTE VRange,
                 BYTE filterMode);
```

Parameters

InputZ

QTMU Start 信号输入阻抗，默认值为 1Mohm 输入阻抗。

QTMU_IMPEDANCE_50： 50ohm 输入阻抗。

QTMU_IMPEDANCE_1M： 1Mohm 输入阻抗。

VRange

QTMU Start 信号电压量程，默认值为 25V 量程。

QTMU_VRNG_5V： 5V 量程。

QTMU_VRNG_25V： 25V 量程。

filterMode

QTMU Start 信号输入滤波器，默认值为直通通路。

QTMU_FILTER_PASS： 直通通路，不使用滤波器。

QTMU_FILTER_100KHz： 100KHz 低通滤波器。

QTMU_FILTER_1MHz： 1MHz 低通滤波器。

QTMU_FILTER_10MHz： 10MHz 低通滤波器。

Remarks

设置 QTMU Stop 信号输入通道模式。

Example

```
/*
设置 qtmu0 Stop 信号输入阻抗为 1Mohm，电压量程为 25V，不使用滤波器。
*/
    qtmu0.SetStopInput(QTMU_IMPEDANCE_1M,
                       QTMU_VRNG_25V,
                       QTMU_FILTER_PASS);
```

9.8. SetStartTrigger()

```
int SetStartTrigger(double Level,
                    BYTE Slope);
```

Parameters

Level

QTMU Start 信号触发电平值，单位为 V。

Slope

QTMU Start 信号触发极性，默认值为 QTMU_POS_SLOPE。

QTMU_POS_SLOPE： QTMU Start 信号上升沿有效。

QTMU_NEG_SLOPE： QTMU Start 信号下降沿有效。

Remarks

设置 QTMU Start 信号触发电平。

Example

```
/*
设置 qtmu0 Stop 信号触发电平为 3V，上升沿有效。
*/
    qtmu0.SetStartTrigger(3, QTMU_POS_SLOPE);
```

9.9. SetStopTrigger()

```
int SetStopTrigger(double Level,
                   BYTE Slope);
```

Parameters

Level

QTMU Stop 信号触发电平值，单位为 V。

Slope

QTMU Stop 信号触发极性，默认值为 QTMU_POS_SLOPE。

QTMU_POS_SLOPE: QTMU Stop 信号上升沿有效。

QTMU_NEG_SLOPE: QTMU Stop 信号下降沿有效。

Remarks

设置 QTMU Stop 信号触发电平。

Example

```
/*
设置 qtmu0 Stop 信号触发电平为 5V，上升沿有效。
*/
    qtmu0.SetStopTrigger(5, QTMU_POS_SLOPE);
```

9.10. SelectArm()

```
int SelectArm(BYTE armSource,
              UNIT armStart,
              UNIT armStop,
              double extLevel,
              BYTE Polarity);
```

Parameters

armSource

ARM 信号源。

QTMU_NO_ARM: ARM 信号无效。

QTMU_COUNT_ARM: 被测信号周期数作为 ARM 信号。

QTMU_TIME_ARM: 内部定时器作为 ARM 信号。

QTMU_EXT_ARM: ARM 信号由外部输入。

armStart

ARM 信号对开始信号的作用时间。当选择被测信号周期数作为 ARM 信号时，输入值

表示 armStart 个被测信号周期后开始信号有效；当选择内部定时器作为 ARM 信号时，输入值表示 armStart us 后开始信号有效，选择 ARM 信号无效和 ARM 信号由外部输入时，此参数无效。

armStop

ARM 信号对开始信号的作用时间。当选择被测信号周期数作为 ARM 信号时，输入值表示在开始信号有效后，armStop 个被测信号周期之后的结束信号有效；当选择内部定时器作为 ARM 信号时，输入值表示在开始信号有效后，armStop us 之后的结束信号有效，选择 ARM 信号无效和 ARM 信号由外部输入时，此参数无效。

extLevel

选择 ARM 信号由外部输入时，外部 ARM 信号的触发电平，选择其他 ARM 信号时此参数无效。

Polarity

选择 ARM 信号由外部输入时，外部 ARM 信号的触发极性，选择其他 ARM 信号时此参数无效，默认值为高电平有效。

QTMU_POS：　　高电平有效。

QTMU_NEG：　　低电平有效。

Remarks

设置 QTMU ARM 信号。

Example

```
/*
设置 qtmu0 不使用 ARM 信号。
*/
    qtmu0.SelectArm(QTMU_NO_ARM,0,0,QTMU_POS);
```

9.11. SetTimeOut()

```
int SetTimeOut(UNIT mstimeOut);
```

Parameters

mstimeOut

测量等待时间，单位为 ms。

Remarks

设置 QTMU 测量最大的等待时间。

Example

```
/*
qtmu0 测量等待时间为 100ms。
*/
    qtmu0. SetTimeOut (100);
```

9.12. Measure()

```
int Measure(double *value,
            BYTE measMode,
            BYTE tRange,
            UNIT mstimeOut);
```

Parameters

value

时间测量结果，单位为 us。

measMode

测量模式，默认值为低分辨率时间测量。

QTMU_COARSE：　　低分辨率时间测量。

QTMU_FINE：　　高分辨率时间测量。

tRange

时间量程，当选择低分辨率测量模式时，此参数无效，默认值为 QTMU_TIME_US。

QTMU_TIME_US：　　被测时间>1us。

QTMU_TIME_NS：　　被测时间<1us。

mstimeOut

测量等待时间，单位为 ms，默认值为 100ms。

Remarks

对时间量进行测量。

Example

```
/*
qtmu0 对单信号源，低电平-4V，高电平+4V， 周期性 10us 正脉冲信号进行测量，开始
和结束信号均为输入阻抗 1Mohm，电压量程 5V，不使用滤波器，开始信号触发电平 0V，
上升沿有效，结束信号触发电平 0V，下降沿有效。选用低分辨率进行测量，等待时间 1ms。
*/
    double val[4]={0};
    qtmu0.SetStartInput(QTMU_IMPEDANCE_1M,   //开始信号输入阻抗 1Mohm，
                        QTMU_VRNG_5V,        //电压量程 5V，不使用滤波器。
                        QTMU_FILTER_PASS);
    qtmu0.SetStopInput(QTMU_IMPEDANCE_1M,    //结束信号输入阻抗 1Mohm，
                        QTMU_VRNG_5V,        //电压量程 5V，不使用滤波器。
                        QTMU_FILTER_PASS);
    qtmu0.SetStartTrigger(0,QTMU_POS_SLOPE); //开始信号触发电平 0V，
                                             //上升沿有效。
    qtmu0.SetStopTrigger(0,QTMU_NEG_SLOPE);  //结束信号触发电平 0V，
                                             //下降沿有效。
    qtmu0.SetInSource(QTMU_SINGLE_SOURCE); //单信号源。
    qtmu0.Connect();                         //接通输入继电器。
    qtmu0.Measure(val,QTMU_COARSE,QTMU_TRNG_US,1);//低分辨率测量
                                             //等待时间 1ms。
    for(int i=0; i<4; i++ )
    {
    /*假设为四个工位第一子单元的测试结果，赋给被测参数 param*/
    param.SetTestResult(i, 0, val[i]);
    }
    qtmu0.Disconnect();                      //断开输入继电器。
```

9.13. MeasureFreq()

```
int MeasureFreq(double *value,
```

BYTE measMode,

BYTE tRange,

UNIT count,

UNIT mstimeOut);

Parameters

value

频率测量结果，单位为 KHz。

measMode

测量模式，默认值为低分辨率频率测量。

QTMU_COARSE：　　低分辨率频率测量。

QTMU_FINE：　　高分辨率频率测量。

tRange

时间量程，当选择低分辨率测量模式时，此参数无效，默认值为 QTMU_TIME_US。

QTMU_TIME_US：　　被测时间>1us。

QTMU_TIME_NS：　　被测时间<1us。

count

选择用来测量频率的被测信号周期数，默认值为 10。

mstimeOut

测量等待时间，单位为 ms，默认值为 100ms。

Remarks

对周期信号的频率进行测量。

Example

```
/*
qtmu0 对低电平 0V，高电平+15V，50KHz 周期信号进行测量，输入阻抗 1Mohm，电压量程 25V，不使用滤波器，触发电平 10V，上升沿有效，选用低分辨率，选取被测信号的 10 个周期进行测量，等待时间 10ms。
*/
    double val[4]={0};
    qtmu0.SetStartInput(QTMU_IMPEDANCE_1M,  //开始信号输入阻抗 1Mohm，
```

```
                                QTMU_VRNG_25V,        //电压量程 25V，
                                QTMU_FILTER_PASS);    //不使用滤波器。
    qtmu0.SetStartTrigger(10,QTMU_POS_SLOPE); //开始信号触发电平 10V，
                                                  //上升沿有效。
    qtmu0.SetInSource(QTMU_DUAL_SOURCE);      //双信号源。
    qtmu0.Connect();                          //接通输入继电器。
    qtmu0.MeasureFreq(val,QTMU_COARSE,QTMU_TRNG_US,10,10);
                                          //低分辨率测量,等待时间 10ms。
    for(int i=0; i<4; i++ )
    {
    /*假设为四个工位第一子单元的测试结果，赋给被测参数 param*/
    param.SetTestResult(i, 0, val[i]);
    }
    qtmu0.Disconnect();                       //断开输入继电器。
```

9.14. MeasureDutyCycle()

```
int MeasureDutyCycle(double *value,
                     double Level,
                     BYTE dutyType,
                     UNIT mstimeOut);
```

Parameters

value

占空比测量结果。

Level

触发电平，单位为 V。

dutyType

占空比类型，默认值为高占空比。

QTMU_HIGH_DUTY：　　高占空比。

QTMU_LOW_DUTY：　　低占空比。

mstimeOut

测量等待时间，单位为 ms，默认值为 100ms。

Remarks

占空比测量。

```
/*
qtmu0 对低电平 0V，高电平+20V，周期信号占空比测量，输入阻抗 1Mohm，电压量程
25V，不使用滤波器，触发电平 15V，等待时间 1ms。
*/
    double val[4]={0};
    qtmu0.SetStartInput(QTMU_IMPEDANCE_1M,   //开始信号输入阻抗 1Mohm，
                        QTMU_VRNG_25V,       //电压量程 25V，
                        QTMU_FILTER_PASS);   //不使用滤波器。
    qtmu0.SetStopInput(QTMU_IMPEDANCE_1M,    //结束信号输入阻抗 1Mohm，
                        QTMU_VRNG_25V,       //电压量程 25V，
                        QTMU_FILTER_PASS);   //不使用滤波器。
    qtmu0.SetInSource(QTMU_ SINGLE _SOURCE);     //单信号源。
    qtmu0.Connect();                             //接通输入继电器。
    qtmu0. MeasureDutyCycle (val,15, QTMU_HIGH_DUTY,1); //高占空比测量
                                                 //等待时间 1ms。
    for(int i=0; i<4; i++ )
    {
    /*假设为四个工位第一子单元的测试结果，赋给被测参数 param*/
    param.SetTestResult(i, 0, val[i]);
    }
    qtmu0.Disconnect();                          //断开输入继电器。
```

9.15. SetSinglePulseMeas()

```
int SetSinglePulseMeas(BYTE measMode,
                       BYTE tRange,
```

BYTE site);

Parameters

measMode

测量模式，默认值为低分辨率测量。

QTMU_COARSE：　　低分辨率测量。

QTMU_FINE：　　高分辨率测量。

tRange

时间量程，当选择低分辨率测量模式时，此参数无效，默认值为 QTMU_TIME_US。

QTMU_TIME_US：　　被测时间>1us。

QTMU_TIME_NS：　　被测时间<1us。

site

工位号，当选择低分辨率测量模式时，此参数无效。

Remarks

用于单个非重复性信号测量，对测量进行设置。

/*

设置 qtmu0 为低分辨率单个非重复性信号测量。

*/

qtmu0. SetSinglePulseMeas(QTMU_COARSE,QTMU_TIME_US,0);

9.16. SinglePulseMeas()

int SinglePulseMeas(double *value,

BYTE site);

Parameters

value

测量结果，单位为 us。

site

工位号，当选择低分辨率测量模式时，此参数无效。

Remarks

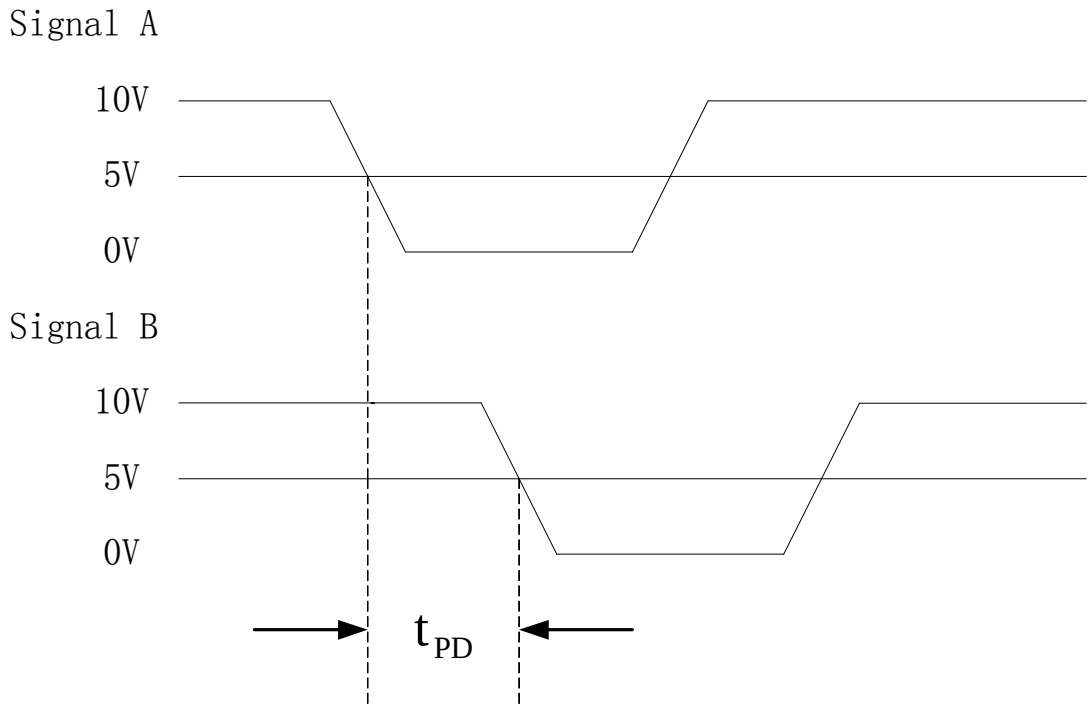
用于单个非重复性信号测量。

```
/*
设置 qtmu0 对单信号源，低电平 0V，高电平 15V，上升沿信号从 1.5V 到 13.5V 的上升时间进行测量，开始信号输入阻抗 1Mohm，电压量程 25V，不使用滤波器，触发电平 1.5V，上升沿有效，结束信号输入阻抗 1Mohm，电压量程 25V，不使用滤波器，触发电平 13.5V，上升沿有效，选用低分辨率进行测量，等待时间 10ms。
*/
    //用户在此处施加激励，使被测信号为 0V 低电平。
    double val[4]={0};
    qtmu0.SetStartInput(QTMU_IMPEDANCE_1M,   //开始信号输入阻抗 1Mohm，
                        QTMU_VRNG_25V,       //电压量程 25V
                        QTMU_FILTER_PASS);   //不使用滤波器。
    qtmu0.SetStopInput(QTMU_IMPEDANCE_1M,    //结束信号输入阻抗 1Mohm，
                        QTMU_VRNG_25V,       //电压量程 25V，
                        QTMU_FILTER_PASS);   //不使用滤波器。
    qtmu0.SetStartTrigger(1.5,QTMU_POS_SLOPE);//开始信号触发电平 0V，
                                              //上升沿有效。
    qtmu0.SetStopTrigger(13.5,QTMU_POS_SLOPE);//结束信号触发电平 0V，
                                              //上升沿有效。
    qtmu0.SetInSource(QTMU_SINGLE_SOURCE);   //单信号源。
    qtmu0.Connect();                          //接通输入继电器。
    qtmu0. SetSinglePulseMeas(QTMU_COARSE,QTMU_TIME_US,0);
                                              //低分辨率测量
    qtmu0. SetTimeOut (10);                   //等待时间 10ms。
    //用户在此处施加激励，使被测信号由 0V 低电平上升至 15V 高电平。
    qtmu0.SinglePulseMeas(val, 0);            //读取测量结果
    for(int i=0; i<4; i++ )
    {
    /*假设为四个工位第一子单元的测试结果，并将结果转换为压摆率，单位为 V/us，赋给被测参数 param*/
```

```
param.SetTestResult(i,0,(13.5-1.5)/(val[i]+1e-12));
}
qtmu0.Disconnect();                                    //断开输入继电器。
```

9.17. QTMU 编程范例

测量两不同信号的时间间隔



如图，要测量两不同信号 A 和 B 下降沿之间的时间间隔t_{PD}，可将信号 A 和 B 分别接入 QTMU 任一通道的 CHA 和 CHB 接口。如将信号 A 接入通道 0 的 CHA_STA，信号 B 接入 CHB_STA（若使用 StationB，则接入 CHA_STB 和 CHB_STB）。

例 1：周期信号的测量

```
double val[4]={0};
qtmu0.SetStartInput(QTMU_IMPEDANCE_1M,  //开始信号输入阻抗 1Mohm，
QTMU_VRNG_25V,                          //电压量程 25V，不使用滤波器。
QTMU_FILTER_PASS);
qtmu0.SetStopInput(QTMU_IMPEDANCE_1M,  //结束信号输入阻抗 1Mohm，
QTMU_VRNG_25V,                          //电压量程 25V，不使用滤波器。
```

```
QTMU_FILTER_PASS);
qtmu0.SetStartTrigger(5,QTMU_NEG_SLOPE);  //开始信号触发电平 5V，
                                          //下降沿有效。
qtmu0.SetStopTrigger(5,QTMU_NEG_SLOPE);   //结束信号触发电平 5V，
                                          //下降沿有效。
qtmu0.SeInSource(QTMU_DUAL_SOURCE);       //双信号源。
qtmu0.Connect();                          //接通输入继电器。
qtmu0.Measure(val,QTMU_COARSE,QTMU_TRNG_US,1);   //低分辨率测量
                                          //等待时间 1ms。
for(int i=0; i<4; i++ )
{
/*假设为四个工位第一子单元的测试结果，赋给被测参数 param*/
param.SetTestResult(i, 0, val[i]);
}
qtmu0.Disconnect();                       //断开输入继电器。
```

例 2：非周期信号的测量

```
double val[4]={0};
qtmu0.SetStartInput(QTMU_IMPEDANCE_1M,  //开始信号输入阻抗 1Mohm，
QTMU_VRNG_25V,                          //电压量程 25V，不使用滤波器。
QTMU_FILTER_PASS);
qtmu0.SetStopInput(QTMU_IMPEDANCE_1M,   //结束信号输入阻抗 1Mohm，
QTMU_VRNG_25V,                          //电压量程 25V，不使用滤波器。
QTMU_FILTER_PASS);
qtmu0.SetStartTrigger(5,QTMU_NEG_SLOPE);  //开始信号触发电平 5V，
                                          //下降沿有效。
qtmu0.SetStopTrigger(5,QTMU_NEG_SLOPE);   //结束信号触发电平 5V，
                                          //下降沿有效。
qtmu0.SeInSource(QTMU_DUAL_SOURCE);       //双信号源。
qtmu0.Connect();                          //接通输入继电器。
```

```
qtmu0. SetSinglePulseMeas(QTMU_COARSE,QTMU_TIME_US,0);
                                                    //低分辨率测量
qtmu0. SetTimeOut (10);                             //等待时间 10ms。
//用户在此处施加激励，使信号 A 由 10V 高电平下降至 0V 低电平。
qtmu0.SinglePulseMeas(val, 0);                      //读取测量结果
for(int i=0; i<4; i++ )
{
/*假设为四个工位第一子单元的测试结果，赋给被测参数 param*/
param.SetTestResult(i, 0, val[i]);
}
qtmu0.Disconnect();                                 //断开输入继电器。
```

10. ACSM

10.1. ACSM()

ACSM(BYTE *channel*)

Parameters

channel　　ACSM 逻辑通道号

ACSM _CH0：　逻辑通道 0

ACSM _CH1：　逻辑通道 1

ACSM _CH2：　逻辑通道 2

ACSM _CH3：　逻辑通道 3

Remarks

定义一个 ACSM，并指定使用的逻辑通道号。系统目前只能配置 1 块 ACSM，单工位工作时最大逻辑通道号为 ACSM_CH3，双工位并行工作时最大逻辑通道号为 ACSM_CH1，四工位并行工作时最大逻辑通道号为 ACSM_CH0。

Example

```
/* 假设系统配置为 1 块 ACSM，单工位工作，定义可使用的 ACSM */
ACSM acsm0(ACSM_CH0) ; //使用逻辑通道 0
ACSM acsm1(ACSM_CH1) ; //使用逻辑通道 1
ACSM acsm2(ACSM_CH2) ; //使用逻辑通道 2
ACSM acsm3(ACSM_CH3) ; //使用逻辑通道 3
/* 假设系统配置为 1 块 ACSM，双工位并行工作，定义可使用的 ACSM */
ACSM acsm0(ACSM_CH0) ; //使用逻辑通道 0
ACSM acsm1(ACSM_CH1) ; //使用逻辑通道 1
/* 假设系统配置为 1 块 ACSM，四工位并行工作，定义可使用的 ACSM */
ACSM acsm(ACSM_CH0);        //使用逻辑通道 0
```

10.2. Init()

void Init(void)

Remarks

初始化 ACSM 模块。所有输入输出继电器断开，输出电压归 0。该函数对所有 ACSM 通道都有效。

Example

```
acsm.Init();              //初始化 ACSM 模块
```

10.3. InitACM()

void InitACM(void)

Remarks

初始化 ACM 模块。所有输入继电器断开，各控制位恢复初始状态。该函数对所有 ACM 通道都有效。

Example

```
acsm.InitACM();           //初始化 ACM 模块
```

10.4. InitACS()

void InitACS(void)

Remarks

初始化 ACS 模块。所有输出继电器断开，各控制位恢复初始状态。该函数对所有 ACS 通道都有效。

Example

```
acsm.InitACS();           //初始化 ACS 模块
```

10.5. DisableACM()

void DisableACM(void)

Remarks

关闭 ACM 模块与所有通道。与 InitACM 功能一样。该函数对所有 ACM 通道都有效。

Example

```
acsm.DisableACM();        //关闭 ACM 模块与所有 ACM 通道
```

10.6. EnableACS()

void EnableACS(void)

Remarks

启动 ACS 模块输出。

Example

acsm.EnableACS ();

10.7. DisableACS()

void DisableACS(void)

Remarks

停止 ACS 模块输出，恢复初始状态。

Example

acsm.DisableACS ();

10.8. ACSDutConnect()

void ACSDutConnect(void)

Remarks

闭合 ACS 模块通道的 DUT 输出继电器，这是输出至用户 DUT 板的继电器。

Example

acsm.ACSDutConnect ();

10.9. ACSDutDisConnect()

void ACSDutDisConnect(void)

Remarks

断开 ACS 模块通道的 DUT 输出继电器，这是输出至用户 DUT 板的继电器。

Example

acsm.ACSDutDisConnect ();

10.10. ACSBusConnect()

void ACSBusConnect(int *busout*)

Parameters

busout BUS 输出继电器选择。

ACS_BUSOUT_OFF： 关闭 BUS 输出继电器（缺省）

ACS_BUSOUT_CH1： 输出至背板总线 CH1

ACS_BUSOUT_CH2： 输出至背板总线 CH2

ACS_BUSOUT_ALL： 输出至背板总线 CH1、CH2

Remarks

闭合 ACS 模块通道的 BUS 输出继电器，这是输出至背板总线的继电器。

Example

```
acsm.ACSBusConnect (ACS_BUSOUT_OFF);
acsm.ACSBusConnect (ACS_BUSOUT_CH1);
acsm.ACSBusConnect (ACS_BUSOUT_CH2);
acsm.ACSBusConnect (ACS_BUSOUT_ALL);
```

10.11. ACSBusDisConnect()

void ACSBusDisConnect(void)

Remarks

断开 ACS 模块通道的 BUS 输出继电器，这是输出至背板总线的继电器。

Example

```
acsm.ACSBusDisConnect ();
```

10.12. ACSConfig()

int ACSConfig(int *wavetype*,double *wavefreq*,double *wavevpp*,double *waveoffsetv*,int *wavefilter*)

Parameters

return ACS 模块配置返回值

0： 配置正确

1： 输出波形配置不正确，缺省 ACS_SINE_WAVE

2： 输出波形频率配置不正确，缺省 0.0 KHz

3： 输出波形峰峰值配置不正确，缺省 0.0 V

4： 输出波形偏置电压配置不正确，缺省 0.0 V

5： 输出波形最大值超限，缺省峰峰值 0.0 V，缺省偏置电压 0.0 V

6： 输出波形滤波器选择不正确，缺省 ACS_FILTER_OFF

wavetype 输出波形选择

ACS_SINE_WAVE 正弦波（缺省）

ACS_SQUARE_WAVE 方波

ACS_TRI_WAVE 三角波

ACS_DC_WAVE 直流电平

wavefreq 输出波形频率，单位：KHz ，范围：0.1 KHz ~ 200 KHz

wavevpp 输出波形峰峰值，单位：V ，范围：0.0 V ~ +10.0 V

waveoffsetv 输出波形偏置电压，单位：V ，范围：-10.0 V ~ +10.0 V

wavefilter 输出波形低通滤波器

ACS_FILTER_OFF 关闭输出低通滤波器

ACS_FILTER_50K 50KHz 低通滤波器

ACS_FILTER_100K 100KHz 低通滤波器

ACS_FILTER_200K 200KHz 低通滤波器

Remarks

配置 ACS 模块输出参数。

当选择 DC_WAVE 直流电平输出时，只有偏置电压 waveoffsetv 有效。

Example

```
int status;
/* 正弦波 频率 10.0 KHz 峰峰值 8.0V 偏置电压 2.0V 输出低通滤波器关闭 */
status=acsm.ACSConfig(ACS_SINE_WAVE, 10.0, 8.0, 2.0,
ACS_FILTER_OFF);
/* 方波 频率 100.0 KHz 峰峰值 10.0V 偏置电压 0.0V 输出低通滤波器 50KHz */
```

status=acsm.ACSConfig(ACS_SQUARE_WAVE, 100.0, 10.0, 0.0, ACS_FILTER_50K);

/* 三角波 频率 50.0 KHz 峰峰值 8.0V 偏置电压 4.0V 输出低通滤波器关闭 */

status=acsm.ACSConfig(ACS_TRI_WAVE, 50.0, 8.0, 4.0, ACS_FILTER_OFF);

10.13. ACMLMeaDutDC()

int ACMLMeaDutDC(int *testvrange*,int *testfilter*,double * *buf*)

Parameters

return ACM 模块 LADC 测量直流配置参数状态返回值

0： 测量配置参数正确，测量结果有效

1： 测量幅度量程参数配置不正确，缺省 ACM_LADC_VR_10V

2： 测量低通滤波器参数配置不正确，缺省 ACM_LADC_LPF_OFF

3： 测量过程中出现超限报警信号，需选择合适的幅度量程

testvrange ACM 模块 LADC 测量幅度量程选择

ACM_LADC_VR_10V： 10 V 档幅度量程（缺省）

ACM_LADC_VR_5V： 5 V 档幅度量程

ACM_LADC_VR_2V： 2 V 档幅度量程

ACM_LADC_VR_1V： 1 V 档幅度量程

ACM_LADC_VR_20V： 20 V 档幅度量程

ACM_LADC_VR_50V： 50 V 档幅度量程

ACM_LADC_VR_100V： 100 V 档幅度量程

testfilter ACM 模块 LADC 测量低通滤波器选择

ACM_LADC_LPF_OFF： 关闭低通滤波器（缺省）

ACM_LADC_LPF_10K： 打开 10KHz 低通滤波器

* *buf* ACM 模块 LADC 测量结果存放首地址

buf 是 double 类型数组首地址，buf[4]。

Remarks

ACM 模块低速 LADC 测量用户 DUT 板直流信号，测量结果为直流电平，低速 LADC

是利用 16 位 200 KHz 的 ADC 芯片进行采样，采样点数固定为 200 点。

Example

```
int status;
double result[4];
status=ACMLMeaDutDC(ACM_LADC_VR_10V,ACM_LADC_LPF_OFF,result);
status=ACMLMeaDutDC(ACM_LADC_VR_5V,ACM_LADC_LPF_OFF,result);
status=ACMLMeaDutDC(ACM_LADC_VR_10V,ACM_LADC_LPF_10K,result);
```

10.14. ACMLMeaDutAC()

int ACMLMeaDutAC(int *testvrange*,int *samplenum*,int *testfilter*,double * *buf*)

Parameters

return ACM 模块 LADC 测量交流配置参数状态返回值

0： 测量配置参数正确，测量结果有效

1： 测量幅度量程参数配置不正确，缺省 ACM_LADC_VR_10V

2： 测量采样点数配置不正确，缺省 200 点

3： 测量低通滤波器参数配置不正确，缺省 ACM_LADC_LPF_OFF

4： 测量过程中出现超限报警信号，需选择合适的幅度量程

testvrange ACM 模块 LADC 测量幅度量程选择

ACM_LADC_VR_10V： 10 V 档幅度量程（缺省）

ACM_LADC_VR_5V： 5 V 档幅度量程

ACM_LADC_VR_2V： 2 V 档幅度量程

ACM_LADC_VR_1V： 1 V 档幅度量程

ACM_LADC_VR_20V： 20 V 档幅度量程

ACM_LADC_VR_50V： 50 V 档幅度量程

ACM_LADC_VR_100V： 100 V 档幅度量程

samplenum ACM 模块 LADC 测量采样点数，范围：10 ~ 60000

testfilter ACM 模块 LADC 测量低通滤波器选择

ACM_LADC_LPF_OFF： 关闭低通滤波器（缺省）

ACM_LADC_LPF_10K： 打开 10KHz 低通滤波器

** buf*　　ACM 模块 LADC 测量结果存放首地址

buf 是 double 类型数组首地址，buf[4]。

Remarks

ACM 模块低速 LADC 测量用户 DUT 板交流信号，测量结果为有效值，低速 LADC 是利用 16 位 200 KHz 的 ADC 芯片进行采样。

Example

```
int status;
double result[4];
status=ACMLMeaDutAC(ACM_LADC_VR_10V,10000,
ACM_LADC_LPF_OFF,result);
status=ACMLMeaDutAC(ACM_LADC_VR_5V,8000,
ACM_LADC_LPF_OFF,result);
status=ACMLMeaDutAC(ACM_LADC_VR_10V,20000,
ACM_LADC_LPF_10K,result);
```

10.15. ACMLMeaBusDC()

int ACMLMeaBusDC(int *nch*,int *testvrange*,int *testfilter*,double ** buf*)

Parameters

return　　ACM 模块 LADC 测量直流配置参数状态返回值

0：　测量配置参数正确，测量结果有效

1：　测量幅度量程参数配置不正确，缺省 ACM_LADC_VR_10V

2：　测量低通滤波器参数配置不正确，缺省 ACM_LADC_LPF_OFF

3：　测量过程中出现超限报警信号，需选择合适的幅度量程

nch　　ACM 模块 LADC 测量背板总线通道

ACSM_BUS_CH0：　背板总线通道 CH0

ACSM_BUS_CH1：　背板总线通道 CH1

ACSM_BUS_CH2：　背板总线通道 CH2

ACSM_BUS_CH3：　背板总线通道 CH3

testvrange　ACM 模块 LADC 测量幅度量程选择

ACM_LADC_VR_10V： 10 V 档幅度量程（缺省）

ACM_LADC_VR_5V： 5 V 档幅度量程

ACM_LADC_VR_2V： 2 V 档幅度量程

ACM_LADC_VR_1V： 1 V 档幅度量程

ACM_LADC_VR_20V： 20 V 档幅度量程

ACM_LADC_VR_50V： 50 V 档幅度量程

ACM_LADC_VR_100V： 100 V 档幅度量程

testfilter ACM 模块 LADC 测量低通滤波器选择

ACM_LADC_LPF_OFF： 关闭低通滤波器（缺省）

ACM_LADC_LPF_10K： 打开 10KHz 低通滤波器

* *buf* ACM 模块 LADC 测量结果存放首地址

buf 是 double 类型数组首地址，buf[4]。

Remarks

ACM 模块低速 LADC 测量背板总线 BUS 直流信号，测量结果为直流电平，低速 LADC 是利用 16 位 200 KHz 的 ADC 芯片进行采样，采样点数固定为 200 点。

Example

```
int status;
double result[4];
status=ACMLMeaBusDC(ACSM_BUS_CH0,ACM_LADC_VR_10V,
ACM_LADC_LPF_OFF,result);
status=ACMLMeaBusDC(ACSM_BUS_CH1,ACM_LADC_VR_5V,
ACM_LADC_LPF_OFF,result);
status=ACMLMeaBusDC(ACSM_BUS_CH2,ACM_LADC_VR_10V,
ACM_LADC_LPF_10K,result);
```

10.16. ACMLMeaBusAC()

int ACMLMeaBusAC(int *nch*,int *testvrange*,int *samplenum*,int *testfilter*,double * *buf*)

Parameters

return ACM 模块 LADC 测量交流配置参数状态返回值

0： 测量配置参数正确，测量结果有效

1： 测量幅度量程参数配置不正确，缺省 ACM_LADC_VR_10V

2： 测量采样点数配置不正确，缺省 200 点

3： 测量低通滤波器参数配置不正确，缺省 ACM_LADC_LPF_OFF

4： 测量过程中出现超限报警信号，需选择合适的幅度量程

nch ACM 模块 LADC 测量背板总线通道

ACSM_BUS_CH0： 背板总线通道 CH0

ACSM_BUS_CH1： 背板总线通道 CH1

ACSM_BUS_CH2： 背板总线通道 CH2

ACSM_BUS_CH3： 背板总线通道 CH3

testvrange ACM 模块 LADC 测量幅度量程选择

ACM_LADC_VR_10V： 10 V 档幅度量程（缺省）

ACM_LADC_VR_5V： 5 V 档幅度量程

ACM_LADC_VR_2V： 2 V 档幅度量程

ACM_LADC_VR_1V： 1 V 档幅度量程

ACM_LADC_VR_20V： 20 V 档幅度量程

ACM_LADC_VR_50V： 50 V 档幅度量程

ACM_LADC_VR_100V： 100 V 档幅度量程

samplenum ACM 模块 LADC 测量采样点数，范围：10 ~ 60000

testfilter ACM 模块 LADC 测量低通滤波器选择

ACM_LADC_LPF_OFF： 关闭低通滤波器（缺省）

ACM_LADC_LPF_10K： 打开 10KHz 低通滤波器

** buf* ACM 模块 LADC 测量结果存放首地址

buf 是 double 类型数组首地址，buf[4]。

Remarks

ACM 模块低速 LADC 测量背板总线 BUS 交流信号，测量结果为有效值，低速 LADC 是利用 16 位 200 KHz 的 ADC 芯片进行采样。

Example

```
int status;
double result[4];
status=ACMLMeaBusAC(ACSM_BUS_CH0,ACM_LADC_VR_10V,10000,
ACM_LADC_LPF_OFF,result);
status=ACMLMeaBusAC(ACSM_BUS_CH1,ACM_LADC_VR_10V,30000,
ACM_LADC_LPF_OFF,result);
status=ACMLMeaBusAC(ACSM_BUS_CH2,ACM_LADC_VR_10V,20000,
ACM_LADC_LPF_10K,result);
```

10.17. ACMLMeaDutMAC()

int ACMLMeaDutMAC(double *testoffsetv*,int *testvgain*,int *samplenum*,int *testfilter*,double * *buf*)

Parameters

return ACM 模块 LADC 测量微小交流配置参数状态返回值

0： 测量配置参数正确，测量结果有效

1： 测量偏置电压参数配置不正确，缺省 0.0 V

2： 测量偏置电压后级增益不正确，缺省 ACM_LADC_VG_X1

3： 测量采样点数配置不正确，缺省 200 点

4： 测量低通滤波器参数配置不正确，缺省 ACM_LADC_LPF_OFF

5： 测量过程中出现超限报警信号，需选择合适的幅度量程

testoffsetv ACM 模块 LADC 测量测量偏置电压，单位：V ，范围：-10.0 V ~ +10.0 V

testvgain ACM 模块 LADC 测量偏置电压后级增益选择

ACM_LADC_VG_X1： 偏置电压后级增益 X1（缺省）

ACM_LADC_VG_X10： 偏置电压后级增益 X10

ACM_LADC_VG_X100：偏置电压后级增益 X100

samplenum ACM 模块 LADC 测量采样点数，范围：10 ~ 60000

testfilter ACM 模块 LADC 测量低通滤波器选择

ACM_LADC_LPF_OFF： 关闭低通滤波器（缺省）

ACM_LADC_LPF_10K： 打开 10KHz 低通滤波器

** buf* ACM 模块 LADC 测量结果存放首地址

buf 是 double 类型数组首地址，buf[4]。

Remarks

ACM 模块低速 LADC 测量用户 DUT 板微小交流信号，测量结果为有效值，低速 LADC 是利用 16 位 200 KHz 的 ADC 芯片进行采样。主要用于测量叠加在直流电压上微小交流信号，测量的原理是利用偏置电压消除测量信号中较大的直流电压，然后对微小交流信号进行放大，测量其交流有效值。

Example

```
int status;
double result[4];
status=ACMLMeaDutMAC(-5.0,ACM_LADC_VG_X100,10000,
ACM_LADC_LPF_OFF,result);
status=ACMLMeaDutMAC(-2.5,ACM_LADC_VG_X10,50000,
ACM_LADC_LPF_OFF,result);
```

10.18. ACMLMeaBusMAC()

int ACMLMeaBusMAC(int *nch*,double *testoffsetv*,int *testvgain*,int *samplenum*,int *testfilter*,double * *buf*)

Parameters

return ACM 模块 LADC 测量微小交流配置参数状态返回值

0： 测量配置参数正确，测量结果有效

1： 测量偏置电压参数配置不正确，缺省 0.0 V

2： 测量偏置电压后级增益不正确，缺省 ACM_LADC_VG_X1

3： 测量采样点数配置不正确，缺省 200 点

4： 测量低通滤波器参数配置不正确，缺省 ACM_LADC_LPF_OFF

5： 测量过程中出现超限报警信号，需选择合适的幅度量程

testoffsetv ACM 模块 LADC 测量测量偏置电压，单位：V ，范围：-10.0 V ~ +10.0 V

testvgain ACM 模块 LADC 测量偏置电压后级增益选择

ACM_LADC_VG_X1： 偏置电压后级增益 X1（缺省）

ACM_LADC_VG_X10： 偏置电压后级增益 X10

ACM_LADC_VG_X100：偏置电压后级增益 X100

samplenum ACM 模块 LADC 测量采样点数，范围：10 ~ 60000

testfilter ACM 模块 LADC 测量低通滤波器选择

ACM_LADC_FILTER_OFF： 关闭低通滤波器（缺省）

ACM_LADC_FILTER_10K： 打开 10KHz 低通滤波器

** buf* ACM 模块 LADC 测量结果存放首地址

buf 是 double 类型数组首地址，buf[4]。

Remarks

ACM 模块低速 LADC 测量背板总线 BUS 板微小交流信号，测量结果为有效值，低速 LADC 是利用 16 位 200 KHz 的 ADC 芯片进行采样。主要用于测量叠加在直流电压上微小交流信号，测量的原理是利用偏置电压消除测量信号中较大的直流电压，然后对微小交流信号进行放大，测量其交流有效值。

Example

```
int status;
double result[4];
status=ACMLMeaBusMAC(ACSM_BUS_CH0,-5.0,ACM_LADC_VG_X100,10000, ACM_LADC_LPF_OFF,result);
status=ACMLMeaBusMAC(ACSM_BUS_CH2,-6.5,ACM_LADC_VG_X10,20000, ACM_LADC_LPF_OFF,result);
```

10.19. ACMHMeaDutDC()

int ACMHMeaDutDC(int *testvrange*,double * *buf*)

Parameters

return ACM 模块 HADC 测量直流配置参数状态返回值

0： 测量配置参数正确，测量结果有效

1： 测量幅度量程参数配置不正确，缺省 ACM_HADC_VR_10V

2： 测量过程中出现超限报警信号，需选择合适的幅度量程

testvrange ACM 模块 HADC 测量幅度量程选择

ACM_HADC_VR_10V： 10 V 档幅度量程（缺省）

ACM_HADC_VR_5V： 5 V 档幅度量程

ACM_HADC_VR_2V： 2 V 档幅度量程

ACM_HADC_VR_1V： 1 V 档幅度量程

ACM_HADC_VR_20V： 20 V 档幅度量程

ACM_HADC_VR_50V： 50 V 档幅度量程

ACM_HADC_VR_100V： 100 V 档幅度量程

* *buf* ACM 模块 HADC 测量结果存放首地址

buf 是 double 类型数组首地址，buf[4]。

Remarks

ACM 模块高速 HADC 测量用户 DUT 板直流信号，测量结果为直流电平，高速 HADC 是利用 12 位 10 MHz 的 ADC 芯片进行采样，采样点数固定为 4000 点。

Example

```
int status;
double result[4];
status=ACMHMeaDutDC(ACM_HADC_VR_10V,result);
status=ACMHMeaDutDC(ACM_HADC_VR_5V,result);
status=ACMHMeaDutDC(ACM_HADC_VR_10V,result);
```

10.20. ACMHMeaDutAC()

int ACMHMeaDutAC(int *testvrange*,int *samplenum*,double * *buf*)

Parameters

return ACM 模块 HADC 测量交流配置参数状态返回值

0： 测量配置参数正确，测量结果有效

1： 测量幅度量程参数配置不正确，缺省 ACM_HADC_VR_10V

2： 测量采样点数配置不正确，缺省 4000 点

3： 测量过程中出现超限报警信号，需选择合适的幅度量程

testvrange ACM 模块 HADC 测量幅度量程选择

ACM_HADC_VR_10V： 10 V 档幅度量程（缺省）

ACM_HADC_VR_5V： 5 V 档幅度量程

ACM_HADC_VR_2V： 2 V 档幅度量程

ACM_HADC_VR_1V： 1 V 档幅度量程

ACM_HADC_VR_20V： 20 V 档幅度量程

ACM_HADC_VR_50V： 50 V 档幅度量程

ACM_HADC_VR_100V： 100 V 档幅度量程

samplenum ACM 模块 HADC 测量采样点数，范围：10 ~ 60000

** buf* ACM 模块 HADC 测量结果存放首地址

buf 是 double 类型数组首地址，buf[4]。

Remarks

ACM 模块高速 HADC 测量用户 DUT 板交流信号，测量结果为有效值，高速 HADC 是利用 12 位 10 MHz 的 ADC 芯片进行采样。

Example

```
int status;
double result[4];
status=ACMHMeaDutAC(ACM_HADC_VR_10V,20000,result);
status=ACMHMeaDutAC(ACM_HADC_VR_5V,40000,result);
status=ACMHMeaDutAC(ACM_HADC_VR_10V,50000,result);
```

10.21. ACMHMeaBusDC()

int ACMHMeaBusDC(int *nch*,int *testvrange*,double ** buf*)

Parameters

return ACM 模块 HADC 测量直流配置参数状态返回值

0： 测量配置参数正确，测量结果有效

1： 测量幅度量程参数配置不正确，缺省 ACM_HADC_VR_10V

2： 测量过程中出现超限报警信号，需选择合适的幅度量程

nch ACM 模块 HADC 测量背板总线通道

ACSM_BUS_CH0： 背板总线通道 CH0

ACSM_BUS_CH1： 背板总线通道 CH1

ACSM_BUS_CH2： 背板总线通道 CH2

ACSM_BUS_CH3： 背板总线通道 CH3

testvrange ACM 模块 HADC 测量幅度量程选择

ACM_HADC_VR_10V： 10 V 档幅度量程（缺省）

ACM_HADC_VR_5V： 5 V 档幅度量程

ACM_HADC_VR_2V： 2 V 档幅度量程

ACM_HADC_VR_1V： 1 V 档幅度量程

ACM_HADC_VR_20V： 20 V 档幅度量程

ACM_HADC_VR_50V： 50 V 档幅度量程

ACM_HADC_VR_100V： 100 V 档幅度量程

* *buf* ACM 模块 HADC 测量结果存放首地址

buf 是 double 类型数组首地址，buf[4]。

Remarks

ACM 模块高速 HADC 测量背板总线 BUS 直流信号，测量结果为直流电平，高速 HADC 是利用 12 位 10 MHz 的 ADC 芯片进行采样，采样点数固定为 4000 点。

Example

```
int status;
double result[4];
status=ACMHMeaBusDC(ACSM_BUS_CH0, ACM_HADC_VR_10V,result);
status=ACMHMeaBusDC(ACSM_BUS_CH1, ACM_HADC_VR_5V,result);
status=ACMHMeaBusDC(ACSM_BUS_CH2, ACM_HADC_VR_10V,result);
```

10.22. ACMHMeaBusAC()

int ACMHMeaBusAC(int *nch*,int *testvrange*,int *samplenum*,double * *buf*)

Parameters

return ACM 模块 HADC 测量交流配置参数状态返回值

0： 测量配置参数正确，测量结果有效

1： 测量幅度量程参数配置不正确，缺省 ACM_HADC_VR_10V

2： 测量采样点数配置不正确，缺省 4000 点

3： 测量过程中出现超限报警信号，需选择合适的幅度量程

nch ACM 模块 HADC 测量背板总线通道

ACSM_BUS_CH0： 背板总线通道 CH0

ACSM_BUS_CH1： 背板总线通道 CH1

ACSM_BUS_CH2： 背板总线通道 CH2

ACSM_BUS_CH3： 背板总线通道 CH3

testvrange ACM 模块 HADC 测量幅度量程选择

ACM_HADC_VR_10V： 10 V 档幅度量程（缺省）

ACM_HADC_VR_5V： 5 V 档幅度量程

ACM_HADC_VR_2V： 2 V 档幅度量程

ACM_HADC_VR_1V： 1 V 档幅度量程

ACM_HADC_VR_20V： 20 V 档幅度量程

ACM_HADC_VR_50V： 50 V 档幅度量程

ACM_HADC_VR_100V： 100 V 档幅度量程

samplenum ACM 模块 HADC 测量采样点数，范围：10 ~ 60000

** buf* ACM 模块 HADC 测量结果存放首地址

buf 是 double 类型数组首地址，buf[4]。

Remarks

ACM 模块高速 HADC 测量背板总线 BUS 交流信号，测量结果为有效值，高速 HADC 是利用 12 位 10 MHz 的 ADC 芯片进行采样。

Example

```
int status;
double result[4];
status=ACMHMeaBusAC(ACSM_BUS_CH0,
ACM_HADC_VR_10V,20000,result);
status=ACMHMeaBusAC(ACSM_BUS_CH1,
ACM_HADC_VR_5V,40000,result);
```

status=ACMHMeaBusAC(ACSM_BUS_CH2,
ACM_HADC_VR_10V,50000,result);

11. 系统函数

11.1. AstSetMultiSite()

BOOL AstSetMultiSite(int *nSiteNum*)

Parameters

nSiteNum

工位数。

1：单工位

2：双工位

4：四工位

其它：单工位

Return Values

调用正确返回 0，错误返回非 0。

Remarks

在初始化函数中，调用一次此函数将系统配置为 *nSiteNum* 指向的工位数。

Example

AstSetMultiSite(4); /*系统配置为 4 工位并行方式*/

11.2. AddToParamTemplate()

int AddToParamTemplate(CParam& *dutParam*)

Parameters

dutParam

用户定义的测试参数。

Return Values

调用正确返回 0，错误返回非 0。

Remarks

在用户初始化被测参数后调用此函数将该参数添加到测试模板中，在新版本的程序中，用户在定义一个被测参数时会自动添加到测试模板中，而不再用调用此函数。

Example

```
CParam param1, param2;
...              /* param1 param2 初始化*/
AddToParamTemplate(param1);
AddToParamTemplate(param2);
```

11.3. AstIgnoreAlarm ()

void AstIgnoreAlarm(BYTE *iAlarmBit*)

Parameters

iAlarmBit

Alarm 类型。

Return Values

无。

Remarks

当 VI 源输出钳位时，产生的原因可能是 VI 源恒流开路或器件出现损坏或程序中 clamp 参数设置不合理等，此时，测到的数据可能会在合格判据范围内，但不是真实值，会造成系统误判。为了防止这种现象产生，系统在每次测试时会自动读取 VI 源的 clamp 报警状态，将器件分到默认的 Alarm Bin 中。

为了防止正常的 Clamp Alarm 被处理，如参数的测试值即为钳位值，可以在对应参数的测试函数中增加函数 AstIgnoreAlarm()，　　这样主控程序便不会处理在此测试函数中出现的 Alarm 状态。

11.4. delay_ms()

void　delay_ms(DWORD *ms*)

Parameters

ms

延时的毫秒数。

Remarks

延时函数，精度为毫秒。

11.5. delay_us()

void delay_ms(DWORD *us*)

Parameters

us

延时的微秒数。

Remarks

延时函数，精度为微秒。

11.6. AstInitAllDVI()

int AstInitAllDVI()

Return Values

操作成功返回 0，不成功返回非 0。

Remarks

初始化所有 DVI，根据配置文件里面的 DVI 的总单元数一起进行初始化，能节约初始化的时间，减少器件的总的测试时间。

11.7. AstInitAllPVI()

int AstInitAllPVI()

Return Values

操作成功返回 0，不成功返回非 0。

Remarks

初始化所有 PVI，根据配置文件里面的 PVI 的总单元数一起进行初始化，能节约初始化的时间，减少器件的总的测试时间。

12. 用户测试程序

12.1. Initialize()

Remarks

此函数为用户测试 DLL 中的初始化函数，在用户的 DLL 被调用的时自动执行函数中的内容。

12.2. InitBeforeTest()

Remarks

此函数为用户测试 DLL 中的函数，在每只器件的测试前主测试程序会调用此函数，主要做一些测试前的初始化工作，如资源的初始化，内容可以为空。

12.3. InitAfterTest()

Remarks

此函数为用户测试 DLL 中的函数，在每只器件的测试后主测试程序会调用此函数，主要做一些测试后的初始化工作，如资源的初始化，内容可以为空。

13. 串行测试相关函数

13.1. 工位串行操作宏

BEGIN_SINGLE_SITE(SiteID)

END_SINGLE_SITE()

Parameters

SiteID

串行操作工位号，建议使用 BYTE 类型。

Remarks

在硬件绑定为多工位（双工位或者四工位）时，利用这两个宏（注意：这两个宏一定要成对使用，否则会出现不可预料的错误）可以对某一个工位单独动作，其他工位的硬件不进行动作。

Example

```
for(BYTE i = 0; i < 4; i++)//四个工位依次动作
{
    BEGIN_SINGLE_SITE(i)//串行工位操作开始
    mvs2.Connect();
    dvi2.Connect();
    dvi3.Connect();
    mvs2.SetVoltage(1);
    dvi2.SetModeFVMI(OVI_VRNG_50V, 2, OVI_IRNG_40MA, 40e-3, 0);
    dvi3.SetModeFVMI(OVI_VRNG_50V, 1, OVI_IRNG_40MA, 0, -40e-3);
    dvi2.Enable();
    dvi3.Enable();
    END_SINGLE_SITE()//串行工位操作结束
}
```

13.2. AstSetPVItoSite()

BOOL AstSetPVItoSite(BYTE BoardID, BYTE SiteID, BYTE LogicChannelID[2])

Parameters

BoardID[in]

需要进行串行动作的 PVI 的物理板号,注意板号从 0 开始。

SiteID[in]

需要此 PVI 板进行串行动作的工位号。

LogicChannelID[2] [out]

返回此 PVI 板两个物理通道的逻辑通道号，用户可以利用返回的两个逻辑通道号定义 PVI 对象，利用生成的 PVI 对象进行硬件动作，具体将后面的例子程序。

Remarks

如果系统中只配置有一块比较昂贵的 PVI 源，但是系统其他硬件配置成四工位形式，可以利用此函数将此 PVI 串行动作，以节约测试成本。

Example

此示例程序主要是在系统其他硬件为四工位配置的情况下，利用一块 PVI 串行在四个工位进行测试。

```
for(BYTE i = 0; i < 4; i++)
{
    BEGIN_SINGLE_SITE(i)
    cbit.SetCBIT(cbitdata);
    mvs2.Connect();
    dvi2.Connect();
    dvi3.Connect();
    delay_ms(1);
    mvs2.SetVoltage( float(-Vo / 2.0) );
dvi2.SetModeFVMI(OVI_VRNG_50V,5,OVI_IRNG_40MA,40e-3f,40e-3f);
dvi3.SetModeFVMI(OVI_VRNG_50V, 5, OVI_IRNG_40MA, 40e-3f, -40e-3f);
    dvi2.Enable();
    dvi3.Enable();
```

```
        delay_ms(1);
        delay_ms(RELAY_TIME);
        dvi2.Measure(adresult, AD_SAMPLE);
        dvi2.Disable();
        dvi3.Disable();
        mvs2.Disconnect();
        mvs2.SetVoltage(0);
        BYTE logicalsite[2];
        AstSetPVItoSite(0, i, logicalsite);//将 PVI 的第 0 块板配置到工位 i
        PVI pvi0(logicalsite[0]);
        pvi0.Init();
        pvi0.Connect();
        pvi0.SetModeFVMV(PVI_VRNG_20V, 2, PVI_IRNG_1MA, 1e-3, -1e-3);
        pvi0.Enable();
        pvi0.Measure(adresult);
        PVI pvi1(logicalsite[1]);
        pvi1.Init();
        pvi1.Connect();
        pvi1.SetModeFVMV(PVI_VRNG_20V, 2, PVI_IRNG_1MA, 1e-3, -1e-3);
        pvi1.Enable();
        pvi1.Measure(adresult);
        pvi0.Disable();
        pvi1. Disable();
        END_SINGLE_SITE()
    }
```