

MA111

Application Note

USB Vendor ID : 0x0E6A

USB Product ID : 0x0124

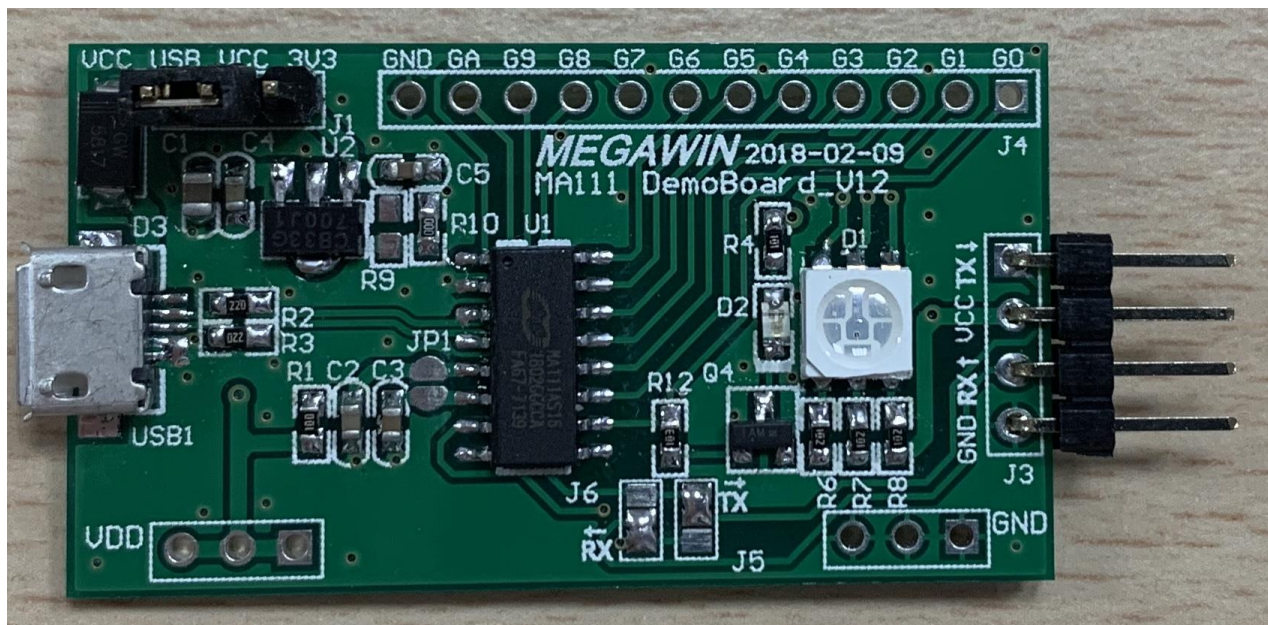
Index

1	Demo Board	4
1.1	Whole Picture	4
1.2	Demo Board Hardware Instruction	5
1.3	Demo Board Circuit	6
2	DLL Functions Description	7
2.1	Introduction	7
2.2	Files Needed	7
2.3	Description of MA111 DLL Function	8
2.3.1	MWBridge structure	8
2.3.2	ConnectBridge	9
2.3.3	DisconnectBridge	10
2.3.4	SetStreamReadTimeOut	11
2.3.5	SetStreamWriteTimeOut	12
2.3.6	ResetBridge	13
2.3.7	ClearBuffer	14
2.3.8	GetDLLVersion	15
2.3.9	GetStatus	15
2.3.10	SetStreamMode	16
2.3.11	GetStreamMode	17
2.3.12	StreamWrite	18
2.3.13	StreamRead	19
2.3.14	SetGPIOAlternateFunction	20
2.3.15	GetGPIOAlternateFunction	21
2.3.16	SetGPIOMode	22
2.3.17	GetGPIOMode	23
2.3.18	SetGPIOData	24
2.3.19	GetGPIOData	25
2.3.20	SetGPIOInterrupt	26
2.3.21	GetGPIOInterrupt	27
2.3.22	SetGPIOPulse	28
2.3.23	GetGPIOPinStatus	29
2.3.24	SetPWMDutyValue	30
2.3.25	GetPWMDutyValue	31
2.3.26	Error Code Reference	32
3	Demo Application Description	33

4	How to use	36
4.1.1	Flow Chart	36
4.1.2	UART	37
4.1.3	SPI	42
4.1.4	I2C	47
5	Revision history.....	51

1 Demo Board

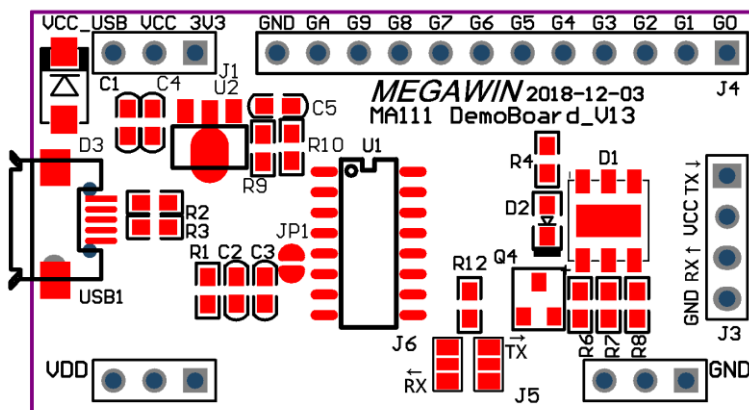
1.1 Whole Picture



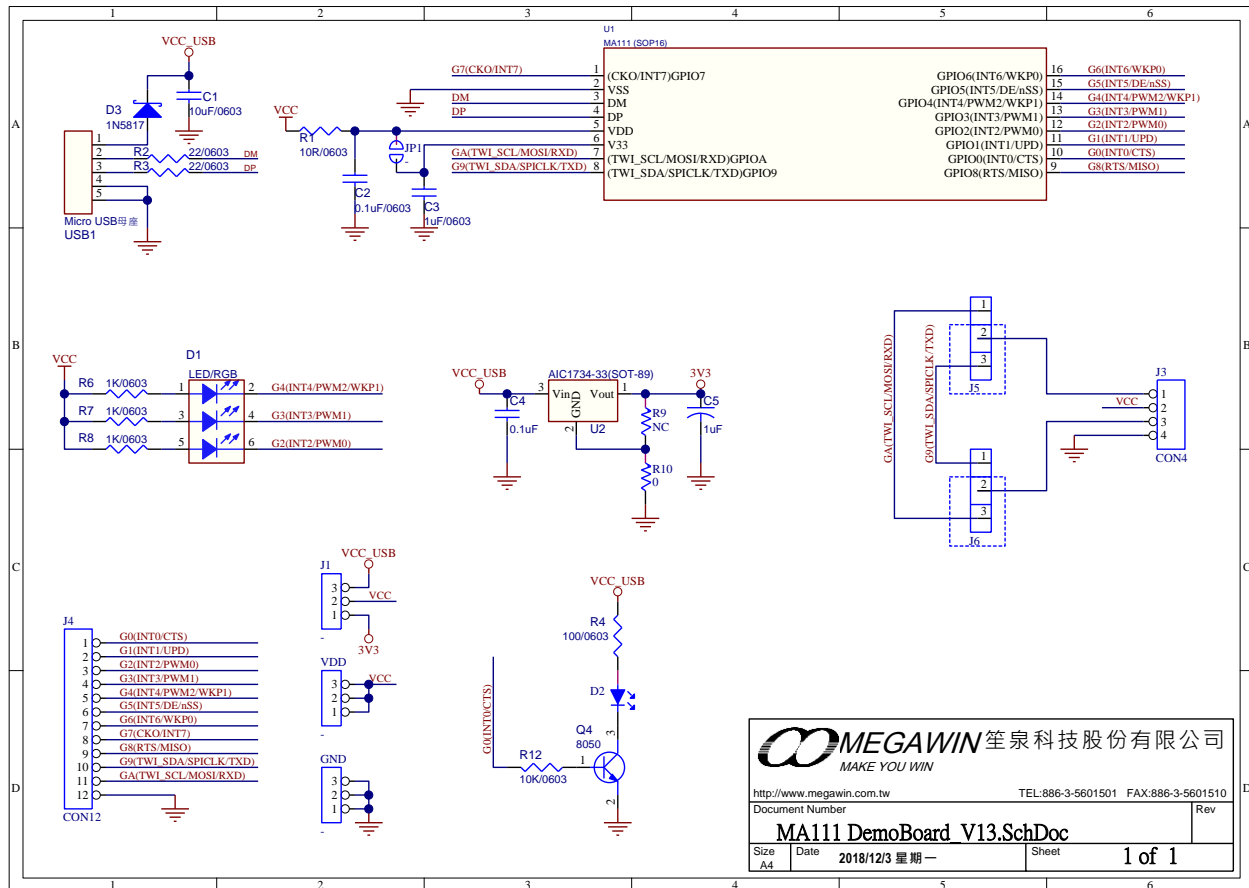
MA111 demo board

1.2 Demo Board Hardware Instruction

1. USB1 : Micro USB Connector
2. J1 :
 - a. 5V --USB 5V Output
 - b. VCC -- Select 5V or 3V3(Default 5V)
 - c. 3V3 --Chip 3.3V LDO Output
3. U1 : MA111
4. J4:
 - a. GND
 - b. GA(I2C_SCL/MOSI/RXD) --GPIOA
 - c. G9(I2C_SDA/SPICLK/TXD)--GPIO9
 - d. G8(RTS/MISO) --GPIO8
 - e. G7(CKO/INT7) --GPIO7
 - f. G6(INT6/WKP0) --GPIO6
 - g. G5(INT5/DE/nSS) --GPIO5
 - h. G4(INT4/PWM2/WKP1) --GPIO4
 - i. G3(INT3/PWM1) --GPIO3
 - j. G2(INT2/PWM0) --GPIO2
 - k. G1(INT1/UPD) --GPIO1
 - l. G0(INT0/nCTS) --GPIO0
5. J3 : UART
 - a. TXD : MA111 UART TXD
 - b. VCC : 5V Output
 - c. RXD : MA111 UART RXD
 - d. GND
6. LED :
 - a. D1 : 3 channel PWM output at R/G/B Tricolor LED.
 - b. D2 : user defined that is connected to GPIO0



1.3 Demo Board Circuit



2 DLL Functions Description

2.1 Introduction

This section explains how to use MWBridge DLL for the MA111 device.

2.2 Files Needed

Add following files, MWBridge (DLL, Lib, and H), to your program path, and make sure they are in the searching path.

MWBridge.DLL

MWBridge.lib

MWBridgeDLL.H

Then, add “#pragma comment(lib, “MWBridge.lib”)” in the program header file.

2.3 Description of MA111 DLL Function

2.3.1 MWBridge structure

```
struct sMWBridge
{
    DWORD    VID;
    DWORD    PID;
    DWORD    ReadTimeOut;
    DWORD    WriteTimeOut;
    HANDLE    Handle;
};
```

Parameter:

VID:

Megawin's VID number is **0x0E6A**.

PID:

MA111's product ID is **0x0124**.

ReadTimeOut:

Specifies the time-out interval, in milliseconds. Within this given time interval, read activity should be finished; otherwise, it will be considered as an error.

WriteTimeOut:

Specifies the time-out interval, in milliseconds. Within this given time interval, write activity should be finished; otherwise, it will be considered as an error.

Handle:

Do not modify this value, reserved for DLL.

2.3.2 ConnectBridge

[ConnectBridge\(\)](#) sets up the software relation between user application and MA111 device. It pairs up with [DisconnectBridge\(\)](#).

```
MWBridgeAPI DWORD __stdcall  
ConnectBridge(sMWBridge *pMWBridge ,  
             UINT Index = 1,  
             DWORD BufferSize = 64 * 1024);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

Index

Device index of a MA111, and it starts from one. (Default is 1)

BufferSize

Device buffer size, to keep polling data. (Default is 64k)

Sample Code:

```
DWORD      dwResult = 0;  
sMWBridge  BridgeObj;  
BridgeObj.VID = 0x0E6A;  
BridgeObj.PID = 0x0124;  
BridgeObj.ReadTimeOut = 0xffffffff; // demo purpose  
BridgeObj.WriteTimeOut = 0xffffffff;  
dwResult = ConnectBridge( &BridgeObj );  
if ( dwResult == ERROR\_SUCCESS ){  
    // open success  
} else{  
    // open failure  
}
```

2.3.3 DisconnectBridge

[DisconnectBridge\(\)](#) closes the relation between user application and MA111 device. It pairs up with [ConnectBridge\(\)](#).

```
BridgeMiniAPI DWORD __stdcall  
DisconnectBridge(sMWBridge *pMWBridge);
```

Parameter:

pMWBridge
Caller-supplied pointer to a [sMWBridge](#) structure.

Remark:

It is necessary to call disconnect after each call of connect.

Sample Code:

```
DWORD   dwResult = 0;  
  
...  
dwResult = ConnectBridge( &BridgeObj , 1 );  
  
...  
dwResult = DisconnectBridge(&BridgeObj);  
if ( dwResult == ERROR\_SUCCESS ){  
    // close success  
}
```

2.3.4 SetStreamReadTimeOut

Modify ReadTimeOut for sMWBridge struct.

```
BridgeMiniAPI DWORD __stdcall  
SetStreamReadTimeOut (sMWBridge *pMWBridge,  
                      DWORD Times);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

Times

This variable is going to set the time out value (in millisecond) for the [sMWBridge](#).

Sample Code:

```
DWORD dwResult = 0;  
dwResult = SetStreamReadTimeOut(&BridgeObj, 5000);  
if ( dwResult == ERROR\_SUCCESS ){  
    // SetStreamReadTimeout success  
}
```

2.3.5 SetStreamWriteTimeOut

Modify WriteTimeOut for sMWBridge struct.

```
BridgeMiniAPI DWORD __stdcall  
SetStreamWriteTimeOut (sMWBridge *pMWBridge,  
                        DWORD Times);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

Times

This variable is going to set the time out value (in millisecond) for the [sMWBridge](#).

Sample Code:

```
DWORD dwResult = 0;  
dwResult = SetStreamWriteTimeOut(&BridgeObj, 5000);  
if ( dwResult == ERROR\_SUCCESS ){  
    // SetStreamWriteTimeOut success  
}
```

2.3.6 ResetBridge

The [ResetBridge\(\)](#) does two operations. First, it will send the reset command down to MA111, and MA111 will reset itself. Secondly, this function will call [DisconnectBridge\(\)](#) API to close the relation with MA111. Because when the MA111 device resets itself, the previous device handle is invalid; therefore, it is necessary to disconnect and connect again before we do any further operations.

```
BridgeMiniAPI DWORD __stdcall  
ResetBridge (sMWBridge *pMWBridge);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

Remark:

Within [ResetBridge\(\)](#), after device has been reset, the DLL internally call [DisconnectBridge\(\)](#) itself. So, user may call [DisconnectBridge\(\)](#) again to pair up the Connect/Disconnect logic or simply [ConnectBridge\(\)](#) to open up the device.

Sample Code:

```
DWORD dwResult = 0;  
  
dwResult = ResetBridge(&BridgeObj);  
dwResult = ConnectBridge( &BridgeObj );
```

2.3.7 ClearBuffer

The [ClearBuffer\(\)](#) cleans up DLL internal buffer 、 Device buffer.

```
BridgeMiniAPI DWORD __stdcall  
ClearBuffer(sMWBridge *pMWBridge,  
           eClear ClearType);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

ClearType:

Caller-supplied pointer to a eClear Enum,
Select clear DLL or Device buffer.

Sample Code:

```
DWORD dwResult = 0;  
  
dwResult = ClearBuffer(&BridgeObj, ClearDLL);  
if ( dwResult == ERROR\_SUCCESS ){  
    // clear success  
}
```

2.3.8 GetDLLVersion

Return the version of this DLL.

```
BridgeMiniAPI DWORD __stdcall  
GetDLLVersion( DWORD *pDLLVersion );
```

Parameter:

pDLLVersion:

The version number will be retrieved from DLL.

Sample Code:

```
DWORD   dwResult = 0;  
DWORD   dwVersion = 0;  
dwResult = GetDLLVersion( &dwVersion );
```

2.3.9 GetStatus

The [GetStatus\(\)](#) retrieves the UART mode 、I2C mode and GPIO status.

```
BridgeMiniAPI DWORD __stdcall  
GetStatus(sMWBridge *pMWBridge,  
          sStatus *pStatus);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

sStatus:

Caller-supplied pointer to a sStatus structure.

Sample Code:

```
DWORD   dwResult = 0;  
sStatus  status;  
dwResult = GetStatus(&BridgeObj, &status);  
if ( dwResult == ERROR\_SUCCESS ){  
    // GetStatus OK  
}
```

2.3.10 SetStreamMode

The [SetStreamMode\(\)](#) function has four modes can be configured. And they are Idle, UART, SPI and I2C mode.

```
BridgeMiniAPI DWORD __stdcall  
SetStreamMode(sMWBridge *pMWBridge,  
             sStreamSetting StreamSetting);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

StreamSetting:

Caller-supplied pointer to a sStreamSetting structure.

Sample Code:

```
DWORD      dwResult  = 0;  
sStreamSetting setting;  
Setting.Mode = Idle;  
dwResult = SetStreamMode(&BridgeObj, setting);  
if ( dwResult == ERROR\_SUCCESS){  
    // SetStreamMode success  
}
```

2.3.11 GetStreamMode

The [GetStreamMode\(\)](#) function retrieves stream mode settings from device. The [GetStreamMode\(\)](#) and [SetStreamMode\(\)](#) are pair functions.

```
BridgeMiniAPI DWORD __stdcall  
GetStreamMode(sMWBridge *pMWBridge,  
             sStreamSetting *pStreamSetting);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

pStreamSetting;

Caller-supplied pointer to a sStreamSetting structure.

Sample Code:

```
DWORD      dwResult = 0;  
sStreamSetting setting;  
dwResult = GetStreamMode(&BridgeObj, &setting);  
if ( dwResult == ERROR\_SUCCESS){  
    // GetStreamMode success  
}
```

2.3.12 StreamWrite

The [StreamWrite\(\)](#) function writes data to MA111 device.

BridgeMiniAPI DWORD __stdcall

```
StreamWrite ( sMWBridge *pMWBridge,  
             DWORD      NumberOfBytesToWrite,  
             BYTE        *pBuffer,  
             DWORD        *pNumberOfBytesWritten,  
             sOptions     Options);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

NumberOfBytesToWrite:

It is the length of pBuffer.

pBuffer:

Pointer to the buffer containing the data to be written to the MA111 device.

pNumberOfBytesWritten:

It is the actually written length.

Options:

Caller-supplied pointer to a sOptions structure.

Sample Code:

```
DWORD  dwResult  = 0;  
BYTE    buffer[100] = { 0 };  
DWORD   written   = 0;  
sOptions options;  
// UART write data  
options.Mode = UART  
options.UART.TADDRActive = Disable;  
options.UART.SADDRActive = Disable;  
dwResult = StreamWrite(&BridgeObj, 100, buffer, &written, options);  
if ( dwResult == ERROR\_SUCCESS ){  
    // StreamWrite ok  
}
```

2.3.13 StreamRead

The [StreamRead\(\)](#) function reads data from the MA111 device.

BridgeMiniAPI DWORD __stdcall

```
StreamRead ( sMWBridge *pMWBridge,  
            DWORD      NumberOfBytesToRead,  
            BYTE        *pBuffer,  
            DWORD      *pNumberOfBytesToRead,  
            sOptions    Options);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

NumberOfBytesToRead:

It is the length to be read.

pBuffer:

Pointer to the buffer that receives the data read from the MA111 device.

pNuberOfBytesRead:

It is the actually read length.

Options:

Caller-supplied pointer to a sOptions structure.

Sample Code:

```
DWORD  dwResult  = 0;  
BYTE    buffer[100] = {0};  
DWORD  retReturn  = 0;  
sOptions options  
// UART  
options.Mode = UART  
options.UART.TADDRActive = Disable;  
options.UART.SADDRActive = Disable  
dwResult = StreamReadData(&BridgeObj, 100, buffer, &retReturn, options);  
if ( dwResult == ERROR\_SUCCESS ){  
    // StreamRead success  
}
```

2.3.14 SetGPIOAlternateFunction

The [SetGPIOAlternateFunction\(\)](#) function sets GPIO alternate function setting on MA111.

BridgeMiniAPI DWORD __stdcall

```
SetGPIOAlternateFunction(sMWBridge *pMWBridge,  
sGPIOAlternateSetting GPIOAlternateSetting);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

GPIOAlternateSetting:

Caller-supplied a sGPIOAlternateSetting structure,

Sample Code:

```
DWORD dwResult = 0;  
sGPIOAlternateSetting setting;  
setting.USBRemoteWakeup = Disable;  
setting.USBResetEventOutput = Enable;  
setting.USBSuspendEventOutput = Disable;  
setting.USBSuspendPolarity = Low;  
setting.ICKOConfigured = OutputDisable  
setting.PWM0Output = Disable;  
setting.PWM1Output = Disable;  
setting.PWM2Output = Disable;  
setting.WKP0Wakeup = Disable;  
setting.WKP0ActiveLevel = Low;  
setting.WKP1Wakeup = Enable;  
setting.WKP1ActiveLevel = High;  
dwResult = SetGPIOAlternateFunction(&BridgeObj, setting);  
if ( dwResult == ERROR\_SUCCESS ){  
    // SetGPIOAlternateFunction OK  
}
```

2.3.15 GetGPIOAlternateFunction

The [GetGPIOAlternateFunction\(\)](#) function retrieves GPIO alternate function setting from MA111. The [GetGPIOAlternateFunction\(\)](#) and [SetGPIOAlternateFunction\(\)](#) are pair functions.

BridgeMiniAPI DWORD __stdcall

```
GetGPIOAlternateFunction(sMWBridge *pMWBridge,  
                        sGPIOAlternateSetting *pGPIOAlternateSetting);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

pGPIOAlternateSetting

Caller-supplied pointer to a sGPIOAlternateSetting structure.

Sample Code:

```
DWORD          dwResult = 0;  
sGPIOAlternateSetting  setting;  
  
dwResult = GetGPIOAlternateFunction(&BridgeObj, &setting);  
if ( dwResult == ERROR\_SUCCESS ){  
    // GetGPIOAlternateFunction OK  
}
```

2.3.16 SetGPIOMode

The [SetGPIOMode\(\)](#) function sets GPIO Mode on each pin.

BridgeMiniAPI DWORD __stdcall

```
SetGPIOMode( sMWBridge\_ *pMWBridge,  
             sGPIOModeSetting ModeSetting);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge_](#) structure.

ModeSetting:

Caller-supplied a sGPIOModeSetting structure,

ModeSetting is 11 bytes of array, and each byte is corresponding with each pin.

Sample Code:

```
DWORD          dwResult = 0;  
sGPIOModeSetting setting;  
setting.GPIO[0].Setting = Enable;  
setting.GPIO[0].Mode = OpenDrainOutput;  
setting.GPIO[1].Setting = Enable;  
setting.GPIO[1].Mode = PushPullOutput;  
setting.GPIO[2].Setting = Enable;  
setting.GPIO[2].Mode = PushPullOutput;  
setting.GPIO[3].Setting = Disable;  
setting.GPIO[4].Setting = Disable;  
setting.GPIO[5].Setting = Disable;  
setting.GPIO[6].Setting = Disable;  
setting.GPIO[7].Setting = Disable;  
setting.GPIO[8].Setting = Disable;  
setting.GPIO[9].Setting = Disable;  
setting.GPIO[10].Setting = Disable;  
dwResult = SetGPIOMode(&BridgeObj, setting);  
if ( dwResult == ERROR\_SUCCESS ){  
    // SetGPIOMode OK  
}
```

2.3.17 GetGPIOMode

The [GetGPIOMode\(\)](#) function retrieves GPIO Mode from MA111. The [GetGPIOMode\(\)](#) and [SetGPIOMode\(\)](#) are pair functions.

```
BridgeMiniAPI DWORD __stdcall  
GetGPIOMode( sMWBridge *pMWBridge,  
             sGPIOModeSetting *pModeSetting);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

pModeSetting

Caller-supplied pointer to a sGPIOModeSetting structure.

pModeSetting is 11 bytes of array, and each byte is corresponding with each GPIO pin.

Sample Code:

```
DWORD          dwResult = 0;  
sGPIOModeSetting setting;  
  
dwResult = GetGPIOMode(&BridgeObj, &setting);  
if ( dwResult == ERROR\_SUCCESS ){  
    // GetGPIOMode OK  
}
```

2.3.18 SetGPIOData

The [SetGPIOData\(\)](#) function sets GPIO Data on each pin.

BridgeMiniAPI DWORD __stdcall

```
SetGPIOData( sMWBridge *pMWBridge,  
             sGPIODataSetting DataSetting);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

DataSetting:

Caller-supplied a sGPIODataSetting structure,

DataSetting is 11 bytes of array, and each byte is corresponding with each pin.

Sample Code:

```
DWORD          dwResult = 0;  
sGPIOInterruptSetting setting;  
setting.GPIO[0].Setting = Enable;  
setting.GPIO[0].Data = OutputHigh;  
setting.GPIO[1].Setting = Enable;  
setting.GPIO[1].Data = OutputHigh;  
setting.GPIO[2].Setting = Enable;  
setting.GPIO[2].Data = OutputLow;  
setting.GPIO[3].Setting = Disable;  
setting.GPIO[4].Setting = Disable;  
setting.GPIO[5].Setting = Disable;  
setting.GPIO[6].Setting = Disable;  
setting.GPIO[7].Setting = Disable;  
setting.GPIO[8].Setting = Disable;  
setting.GPIO[9].Setting = Disable;  
setting.GPIO[10].Setting = Disable;  
  
dwResult = SetGPIOData(&BridgeObj, setting);  
if ( dwResult == ERROR\_SUCCESS ){  
    // SetGPIOData OK  
}
```

2.3.19 GetGPIOData

The [GetGPIOData\(\)](#) function retrieves GPIO Data from MA111. The [GetGPIOData\(\)](#) and [SetGPIOData\(\)](#) are pair functions.

```
BridgeMiniAPI DWORD __stdcall  
GetGPIOData( sMWBridge *pMWBridge,  
             sGPIODataSetting *pDataSetting);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

pDataSetting

Caller-supplied pointer to a sGPIODataSetting structure.

pDataSetting is 11 bytes of array, and each byte is corresponding with each GPIO pin.

Sample Code:

```
DWORD          dwResult = 0;  
sGPIODataSetting setting;  
  
dwResult = GetGPIOData(&BridgeObj, &setting);  
if ( dwResult == ERROR\_SUCCESS ){  
    // GetGPIOData OK  
}
```

2.3.20 SetGPIOInterrupt

The [SetGPIOInterrupt\(\)](#) function sets GPIO interrupt on each pin.

BridgeMiniAPI DWORD __stdcall

```
SetGPIOInterrupt( sMWBridge *pMWBridge,  
                  sGPIOInterruptSetting InterruptSetting);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

InterruptSetting:

Caller-supplied a sGPIOInterruptSetting structure,

InterruptSetting is 8 bytes of array, and each byte is corresponding with each pin.

Sample Code:

```
DWORD          dwResult = 0;  
sGPIOInterruptSetting setting;  
setting.GPIO[0] = Enable;  
setting.GPIO[1] = Enable;  
setting.GPIO[2] = Enable;  
setting.GPIO[3] = Enable;  
setting.GPIO[4] = Disable;  
setting.GPIO[5] = Disable;  
setting.GPIO[6] = Disable;  
setting.GPIO[7] = Disable;  
  
dwResult = SetGPIOInterrupt(&BridgeObj, setting);  
if ( dwResult == ERROR\_SUCCESS ){  
    // SetGPIOInterrupt OK  
}
```

2.3.21 GetGPIOInterrupt

The [GetGPIOInterrupt\(\)](#) function retrieves GPIO interrupt setting from MA111. The [GetGPIOInterrupt\(\)](#) and [SetGPIOInterrupt\(\)](#) are pair functions.

BridgeMiniAPI DWORD __stdcall

```
GetGPIOInterrupt ( sMWBridge *pMWBridge,  
                  sGPIOInterruptSetting *pInterruptSetting);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

pInterruptSetting:

Caller-supplied pointer to a sGPIOInterruptSetting structure,

pInterruptSetting is 8 bytes of array, and each byte is corresponding with each GPIO pin.

Sample Code:

```
DWORD          dwResult = 0;  
sGPIOInterruptSetting setting;  
  
dwResult = GetGPIOInterrupt(&BridgeObj, &setting);  
if ( dwResult == ERROR\_SUCCESS ){  
    // GetGPIOInterrupt OK  
}
```

2.3.22 SetGPiOPulse

The [SetGPiOPulse\(\)](#) function sets pulse on each GPIO pin.

BridgeMiniAPI DWORD __stdcall

```
SetGPiOPulse( sMWBridge *pMWBridge,  
              sGPiOPulseSetting PulseSetting);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

PulseSetting:

Caller-supplied a sGPiOPulseSetting structure,

PulseSetting is 11 bytes of array, and each byte is corresponding with each GPIO pin.

Sample Code:

```
DWORD          dwResult = 0;  
sGPiOPulseSetting setting;  
setting.GPIO[0] = OutputPulse;  
setting.GPIO[1] = OutputPulse;  
setting.GPIO[2] = OutputPulse;  
setting.GPIO[3] = OutputPulse;  
setting.GPIO[4] = OutputPulse;  
setting.GPIO[5] = None;  
setting.GPIO[6] = None;  
setting.GPIO[7] = None;  
setting.GPIO[8] = None;  
setting.GPIO[9] = None;  
setting.GPIO[10] = None;  
dwResult = SetGPiOPulse(&BridgeObj, setting);  
if ( dwResult == ERROR\_SUCCESS ){  
    // SetGPiOPulse OK  
}
```

2.3.23 GetGPIOPinStatus

The [GetGPIOPinStatus\(\)](#) function retrieves GPIO pin status from MA111. This function constantly updates its value.

```
BridgeMiniAPI DWORD __stdcall  
GetGPIOPinStatus( sMWBridge *pMWBridge,  
                  sGPIOPinStatus *pPinStatus);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

pPinStatus:

Caller-supplied pointer to a sGPIOPinStatus structure,

pPinStatus is 11 bytes of array, and each byte is corresponding with each pin.

Sample Code:

```
DWORD    dwResult = 0;  
sGPIOPinStatus status;  
  
dwResult = GetGPIOPinStatus(&BridgeObj, &status);  
if ( dwResult == ERROR\_SUCCESS ){  
    // GetGPIOPinStatus OK  
}
```

2.3.24 SetPWMDutyValue

The [SetPWMDutyValue\(\)](#) function sets PWM duty value on each channel.

BridgeMiniAPI DWORD __stdcall

```
SetPWMDutyValue ( sMWBridge *pMWBridge,  
                  sPWMDutySetting PWMDutySetting);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

PWMDutySetting:

Caller-supplied a sPWMDutySetting structure,

PWMDutySetting is 3 bytes of array, and each byte is corresponding with each channel.

Sample Code:

```
DWORD          dwResult = 0;  
sPWMDutySetting setting;  
setting.PWM[0].Setting = Enable;  
setting.PWM[0].Value = 0x01;  
setting.PWM[1].Setting = Enable;  
setting.PWM[1].Value = 0x02;  
setting.PWM[2].Setting = Disable;  
dwResult = SetPWMDutySetting(&BridgeObj, setting);  
if ( dwResult == ERROR\_SUCCESS ){  
    // SetPWMDutyValue OK  
}
```

2.3.25 GetPWMDutyValue

The [GetPWMDutyValue\(\)](#) function retrieves PWM duty value from MA111. The [GetPWMDutyValue\(\)](#) and [SetPWMDutyValue\(\)](#) are pair functions.

```
BridgeMiniAPI DWORD __stdcall  
GetPWMDutyValue( sMWBridge *pMWBridge,  
                 sPWMDutySetting *pPWMDutySetting);
```

Parameter:

pMWBridge

Caller-supplied pointer to a [sMWBridge](#) structure.

pPWMDutySetting:

Caller-supplied pointer to a sPWMDutySetting structure,

pPWMDutySetting is 3 bytes of array, and each byte is corresponding with each channel.

Sample Code:

```
DWORD          dwResult = 0;  
sPWMDutySetting setting;  
  
dwResult = GetPWMDutyValue(&BridgeObj, &setting);  
if ( dwResult == ERROR\_SUCCESS ){  
    // GetPWMDutyValue OK  
}
```

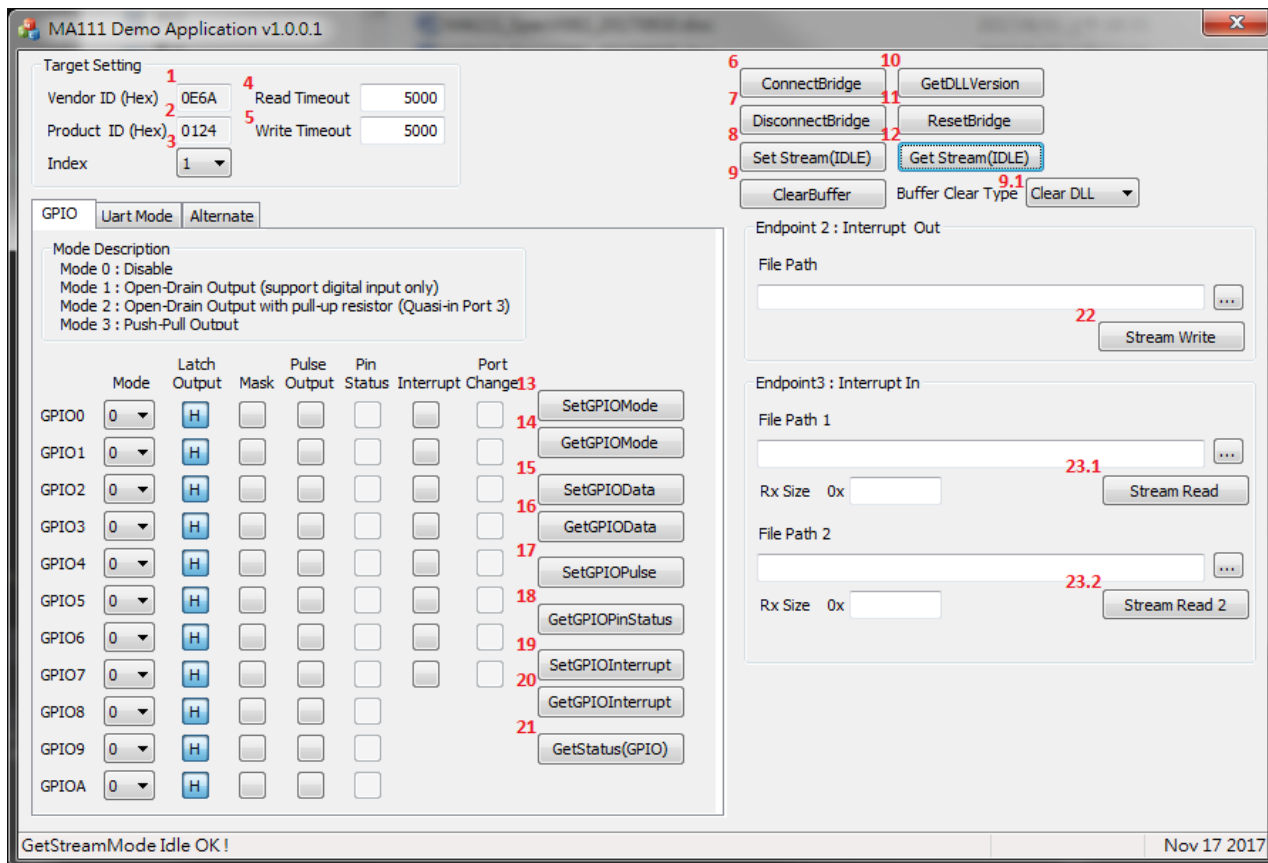
2.3.26 Error Code Reference

// If the function succeeds, the return value is zero.

// It means ERROR_SUCCESS.

<code>#define CANNOT_FINDE_THE_DEVICE_SPECIFIED</code>	0xE0000001L
<code>#define INSUFFICIENT_MOMORY_AVAILABLE</code>	0xE0000002L
<code>#define GET_INPUT_REPORT_SIZE_FAIL</code>	0xE0000003L
<code>#define DELETE_INPUT_QUEUE_FAIL</code>	0xE0000004L
<code>#define SET_INPUT_REPORT_SIZE_FAIL</code>	0xE0000005L
<code>#define HIDD_GETATTRIBUTES_FAIL</code>	0xE0000006L
<code>#define DEVICE_NOT_CONNECT</code>	0xE0000007L
<code>#define INVALID_PARAMETER</code>	0xE0000008L
<code>#define DLLOBJECT_IS_CONNECT</code>	0xE0000009L
<code>#define OPEN_EXISTING_RETURNED</code>	0xE000000AL
<code>#define SET_CMD_FAIL</code>	0xE000000BL
<code>#define GET_CFG_FAIL</code>	0xE000000CL
<code>#define WRONG_MODE_SETTING</code>	0xE000000DL
<code>#define MODE_IS_IDLE</code>	0xE000000EL
<code>#define WRONG_OPTIONS_SETTING</code>	0xE000000FL
<code>#define NOT_FIND_ADDR_TO_READ</code>	0xE0000010L
<code>#define READ_DATA_TIMEOUT</code>	0xE0000011L
<code>#define WRITE_DATA_TIMEOUT</code>	0xE0000012L
<code>#define I2C_SCL_TIMEOUT</code>	0xE0000013L
<code>#define I2C_TX_TERMINATED</code>	0xE0000014L

3 Demo Application Description

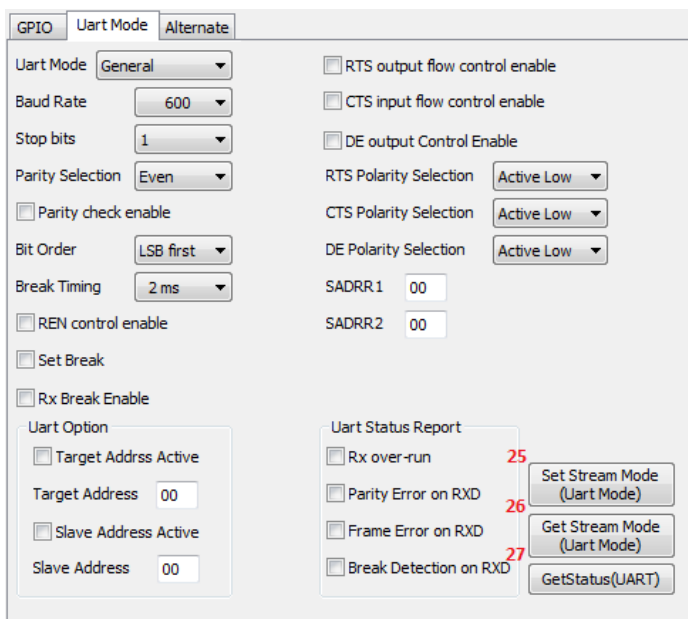


1. Vendor ID: The vendor id is used in function “**ConnectBridge**” to connect MA111.
2. Product ID: The product id is used in function “**ConnectBridge**” to connect MA111.
3. Index: The index is used in function “**ConnectBridge**” to connect to the specify MA111.
4. Read Timeout: The time starts to count when function “**StreamRead**” is called.
5. Write Timeout: The time starts to count when function “**StreamWrite**” is called.
6. ConnectBridge: The code shows how to call the function “**ConnectBridge**”
7. DisconnectBridge: The code shows how to call the function “**DisconnectBridge**”.
8. SetStream(IDLE): The code shows how to call the function “**SetStreamMode**” to set mode to “IDLE”.
9. ClearBuffer: The code shows how to call the function “**ClearBuffer**”.
- 9.1. Clear Type : The parameter of “**ClearBuffer**”.
10. GetDLLVersion: The code shows how to call the function “**GetDLLVersion**”.
11. ResetBridge: The code shows how to call the function “**ResetBridge**”.
12. GetStream(IDLE) : The sample code gets the stream mode and check if the MA111 is in IDLE mode.

From 13 to 21, these buttons with functions need the left-side user interface of the application as input or

output.

13. SetGPIOMode: The code shows how to call the function “SetGPIOMode”.
14. GetGPIOMode: The code shows how to call the function “GetGPIOMode”.
15. SetGPIOData: The code shows how to call the function “SetGPIOData”.
16. GetGPIOData: The code shows how to call the function “GetGPIOData”.
17. SetGPIOPulse: The code shows how to call the function “SetGPIOPulse”.
18. GetGPIOPinStatus: The code shows how to call the function “GetGPIOPinStatus”.
19. SetGPIOInterrupt: The code shows how to call the function “SetGPIOInterrupt”.
20. GetGPIOInterrupt: The code shows how to call the function “GetGPIOInterrupt”.
21. GetStatus(GPIO): The code shows how to call the function “GetStatus”, the code only shows the “Port Change” of the status report.
22. StreamWrite: The code shows how to call the function “StreamWrite”. The parameters of the function are in the tab of “Uart Mode”.
23. StreamRead: The two “Stream Read” buttons do the same thing but use different “Slave Address”. The code shows how to call the function “StreamRead”.



24. Set Stream Mode (Uart Mode): The code shows how to call the function “SetStreamMode” with all parameters.
25. Get Stream Mode (Uart Mode): The code shows how to call the function “GetStreamMode” and display the return value.
26. GetStatus(Uart) : The code shows how to call the function “GetStatus” and display only the status of UART.

GPIO

Uart Mode

Alternate

☐ USB Remote Wakeup Enable
☐ USB Reset Event Output Enable
☐ USB Suspend Event Output Enable
 USB Suspend Polarity ☒ Active Low ☐ Active High
 ICKO Configure

Output Disable

☐ PWM0 Output Enable
☐ PWM1 Output Enable
☐ PWM2 Output Enable
☐ WKP0 Wakeup Enable
 WKP0 Active Level ☒ Active Low ☐ Active High
☐ WKP1 Wakeup Enable
 WKP1 Active Level ☒ Active Low ☐ Active High

Mask

PWM Value

☐ 00
☐ 00
☐ 00

28

SetGPIOAlternateFunction

29

GetGPIOAlternateFunction

30

SetPWMDutyValue

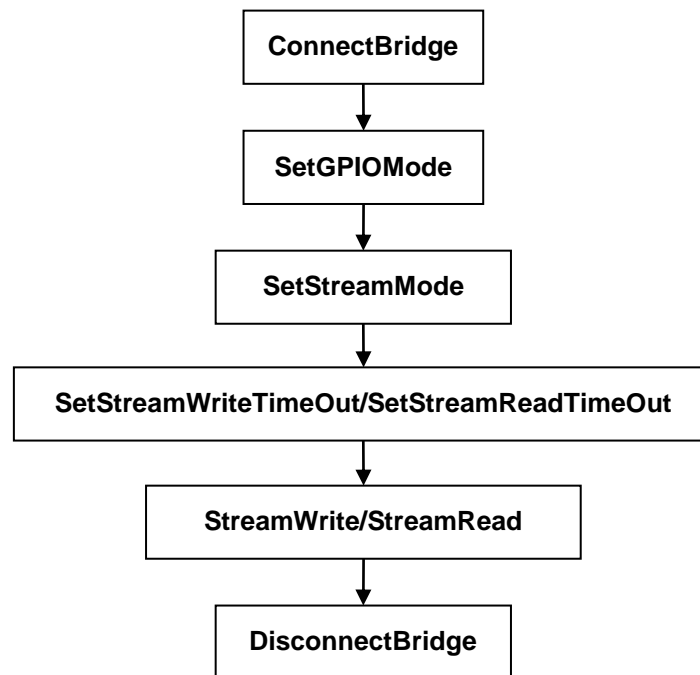
31

GetPWMDutyValue

27. SetGPIOAlternateFunction: The code shows how to call the function “**SetGPIOAlternateFunction**”.
28. GetGPIOAlternateFunction: The code shows how to call the function “**GetGPIOAlternateFunction**”.
29. SetPWMDutyValue: The code shows how to call the function “**SetPWMDutyValue**”.
30. GetPWMDutyValue: The code shows how to call the function “**GetPWMDutyValue**”.

4 How to use

4.1.1 Flow Chart



4.1.2 UART

1. ConnectBridge

To use this function create the connection between user application and MA111. To create structure variable sMWBridge which include product information and read/write timeout value, send it to ConnectBridge and check the return value ([2.3.26 Error Code Reference](#))

```
sMWBridge MWBridge;           //Create structure variable
MWBridge.VID = 0x0E6A;         //Set VID of the structure variable, Megawin's VID is 0x0E6A
MWBridge.PID = 0x0124;         // Set PID of the structure variable, the PID of MA111 is 0x0124
MWBridge.ReadTimeOut = 1000;   //Set ReadTimeOut value of the structure variable, it can be modified. The unit is ms.
MWBridge.WriteTimeOut = 1000;  //Set WriteTimeOut value of the structure variable, it can be modified. The unit is ms.
DWORD Result = ConnectBridge(&MWBridge); //Send the sMWBridge to ConnectBridge, and check the return value
```

2. SetGPIOMode

UART are using GPIO 9 and GPIO for transfer. It needs to be set as OpenDrainOutputWithPullUpResistor

```
sGPIOModeSetting ModeSetting; //Create structure variable
for (int i = 0; i < 9; i++) {
    ModeSetting.GPIO[i].Setting = Disable //Initial GPIO 0~8, keep it as default
}
ModeSetting.GPIO[9].Setting = Enable; //To set GPIO[9] (TXD) as enable
ModeSetting.GPIO[9].Mode = OpenDrainOutputWithPullUpResistor; //To set GPIO[9] as Open Drain Output with Pullup Resistor
ModeSetting.GPIO[10].Setting = Enable; //To set GPIO[10] (RXD) as enable
ModeSetting.GPIO[10].Mode = OpenDrainOutputWithPullUpResistor //To set GPIO[10] as Open Drain Output with Pullup Resistor
DWORD Result = SetGPIOMode(&MWBridge, ModeSetting); //To use SetGPIOMode set the I/O configuration & check the
return value
```

3. SetStreamMode

To set the MA111 transfer mode as UART, and set the UART control configurations.

```
sStreamSetting StreamSetting;           //Create structure variable
StreamSetting.Mode = UART;               //Assign UART Mode as the transfer mode for MA111
StreamSetting.UART.BaudRate = BaudRate_600; // UART Baud rate setting
StreamSetting.UART.BitOrder = LSBfirst;   //To set UART as LSB first ordering
StreamSetting.UART.BreakTiming = Timing_2ms; //To set UART Break time
StreamSetting.UART.RENControl = Enable;    //To set UART REN Control
StreamSetting.UART.ParityCheck = Disable;  //Disable UART Parity Check
StreamSetting.UART.ParitySelection = Even; //To set UART Parity selection as even
StreamSetting.UART.StopBit = Bit_1;        //To UART stop bit as 1 bit
StreamSetting.UART.BreakControl = ClearBreak; //To set UART Break control
StreamSetting.UART.RXBreak = Disable;      //To set UART RX Break
StreamSetting.UART.RTSOutputFlowControl = Disable; //To set UART RTS (Request to Send) output flow control
StreamSetting.UART.RTSPolaritySelection = Low; //To set UART RTS Polarity
StreamSetting.UART.CTSInputFlowControl = Disable; //To set UART CTS (Clear to Send) Input flow control
StreamSetting.UART.CTSPolaritySelection = Low; //To set UART CTS Polarity
StreamSetting.UART.DEOutputControl = Disable; //To set UART to RS485 DE output control
StreamSetting.UART.DEPolaritySelection = Low; //To set UART to RS485 DE Polarity
StreamSetting.UART.Mode = UART_General;    //To set UART as general mode
StreamSetting.UART.SADDR1 = 0x01;          //If device is Slave, it has two address can be used
StreamSetting.UART.SADDR2 = 0x02;          //If the address of 2 slave, it will be traded as the same device
DWORD Result=SetStreamMode(&MWBridge , StreamSetting); //To use SetStreamMode to set the configurations and check the return value
```

ps. There are four UART Modes, when using StreamWrite/StreamRead the parameters is needed to set.

General : No need to set the parameters

Master :

```
Options.Mode = UART;
Options.TADDRActive = Enable; //Enable TADDR
Options.TADDR = 0x00; //To set target address when the device run at Master Mode
```

Slave :

```
Options.Mode = UART;
Options.SADDRActive = Enable; //Enable SADDR
Options.SADDR = 0x02; //To choose Read StreamSetting Address
```

MasterAndSlave :

```
Options.Mode = UART;
Options.TADDRActive = Enable; //Enable TADDR
Options.TADD = 0x00; //To set target address when the device run at Master Mode.
Options.SADDRActive = Enable; //Enable SADDR
Options.SADDR = 0x01; // To choose Read StreamSetting Address
```

4. *SetStreamWriteTimeOut / SetStreamReadTimeOut*

To set the Timeout value before StreamWrite and StreamRead.

```
SetStreamWriteTimeOut(&MWBridge, 2000); //To set StreamWrite timeout value, the unit is ms
```

```
SetStreamReadTimeOut(&MWBridge, 2000); // To set StreamRead timeout value, the unit is ms
```

5. *StreamWrite*

UART Write can use address or no need to use address. It is set in StreamSetting before Write.

UART_General

```
DWORD Size = The output data length; //To set the output data length
BYTE* pBuffer = new BYTE[Size]; //Set the Buffer for output data
DWORD Written = 0; //When Function end and return the actual data length have been sent
sOptions Options; //Create structure variable
Options.Mode = UART; //Set the transfer mode to UART
Options.UART.TADDRActive = Disable; //UART_General no needs TADDR
Options.UART.SADDRActive = Disable; //UART_General no needs SADDR
DWORD Result = StreamWrite(&MWBridge, Size, pBuffer, &Written, Options); //set the configuration, and check the return value
```

UART_Master/UART_MasterAndSlave : Multi-Processor

```
DWORD Size = The output data length; //To set the output data length
BYTE* pBuffer = new BYTE[Size]; //Set the Buffer for output data
DWORD Written = 0; //When Function end and return the actual data length have been sent
sOptions Options; //Create structure variable
Options.Mode = UART; //Set the transfer mode to UART
Options.UART.TADDRActive = Enable; //Enable TADDR
Options.UART.TADDR = 0x00; //Set the Address
DWORD Result = StreamWrite(&MWBridge, Size, pBuffer, &Written, Options); //set the configuration, and check the return value
```

6. StreamRead

UART Read needs to be set according to the set Address (or not set).

UART_Genral : When Address is not set

```
DWORD Size = The length to read;           //To set the input data length
BYTE* pBuffer = new BYTE[Size];           // Set the Buffer for input data
DWORD Read = 0;                             // When Function end and return the actual data length of the input data
sOptions Options;                           //Create structure variable
Options.Mode = UART;                        //Set the transfer mode to UART
Options.UART.TADDRActive = Disable;         // UART_General is no need the TADDR
Options.UART.SADDRActive = Disable;         // UART_General is no need the SADDR
DWORD Result = StreamRead(&MWBridge, Size, pBuffer, &Read, Options); //set the configuration, and check the return value
```

UART_Master/UART_Slave/UART_MasterAndSlave : When Address is needed to set

```
DWORD Size = The length to read;           //To set the input data length
BYTE* pBuffer = new BYTE[Size];           //Set the Buffer for input data
DWORD Read = 0;                             // When Function end and return the actual data length of the input data
sOptions Options;                           //Create structure variable
Options.Mode = UART;                        //Set the transfer mode to UART
Options.UART.TADDRActive = Enable;          // UART_Master/UART_MasterAndSlave needs TADDR
Options.UART.TADD = 0x00;                   // UART_Master/UART_MasterAndSlave needs TADDR
Options.UART.SADDRActive = Enable;          // UART_Slave/UART_MasterAndSlave needs SADDR
Options.UART.TADD = 0x01;                   // UART_Slave/UART_MasterAndSlave needs SADDR
DWORD Result = StreamRead(&MWBridge, Size, pBuffer, &Read, Options); //set the configuration, and check the return value
```


7. *DisconnectBridge*

DisconnectBridge before the end of the program, the DLL will release the memory space

```
DWORD Result = DisconnectBridge(&MWBridge);           //Disconnect the bridge and check the return value
```

4.1.3 SPI

1. *ConnectBridge*

To use this function create the connection between user application and MA111. To create structure variable sMWBridge which include product information and read/write timeout value, send it to ConnectBridge and check the return value ([2.3.26 Error Code Reference](#))

```
sMWBridge MWBridge;           //Create structure variable
MWBridge.VID = 0x0E6A;         //Set VID of the structure variable, Megawin's VID is 0x0E6A
MWBridge.PID = 0x0124;         // Set PID of the structure variable, the PID of MA111 is 0x0124
MWBridge.ReadTimeOut = 1000;   //Set ReadTimeOut value of the structure variable, it can be modified. The unit is ms.
MWBridge.WriteTimeOut = 1000;  //Set WriteTimeOut value of the structure variable, it can be modified. The unit is ms.
DWORD Result = ConnectBridge(&MWBridge); //Send the sMWBridge to ConnectBridge, and check the return value
```

2. *SetGPIOMode*

SPI are using GPIO5, GPIO8, GPIO9 and GPIOA. Receiver GPIO8 set to "OpenDrainOutputWithPullUpResistor" I/O mode, and the other three pins set to "PushPullOutput" I/O mode.

```
sGPIOModeSetting ModeSetting; //Create structure variable
for (int i = 0; i < 11; i++) {
    ModeSetting.GPIO[i].Setting = Disable //Initial GPIO, keep it as default
}
ModeSetting.GPIO[8].Setting = Enable;     //To enable GPIO[8]
ModeSetting.GPIO[8].Mode = OpenDrainOutputWithPullUpResistor; //To set GPIO[8] as OpenDrainOutputWithPullUpResistor
ModeSetting.GPIO[5].Setting = Enable;     //To enable GPIO[5]
ModeSetting.GPIO[5].Mode = PushPullOutput; //To set GPIO[5] as PushPullOutput
ModeSetting.GPIO[9].Setting = Enable;     //To enable GPIO[9]
ModeSetting.GPIO[9].Mode = PushPullOutput; //To set GPIO[9] as PushPullOutput
ModeSetting.GPIO[10].Setting = Enable;    //To enable GPIO[10]
ModeSetting.GPIO[10].Mode = PushPullOutput; // To set GPIO[10] as PushPullOutput
DWORD Result = SetGPIOMode(&MWBridge, ModeSetting); //To Set GPIO Mode and check the return value
```

3. *SetStreamMode*

To set the MA111 transfer mode as SPI, and set the SPI control configurations.

```
sStreamSetting StreamSetting;           //Create structure variable
StreamSetting.Mode = SPI;                //Assign SPI Mode as the transfer mode for MA111
StreamSetting.SPI.ModeSelection = Mode0; //Use SPI Mode0
StreamSetting.SPI.BitOrder = LSBfirst;   //To set SPI as LSB first ordering
StreamSetting.SPI.nSSOutputControl = Enable; //Enable SPI nSS output control
StreamSetting.SPI.ClockRate = ClockRate_2MHz; //To set SPI clock rate
DWORD Result = SetStreamMode(&MWBridge, StreamSetting); //To use SetStreamMode to set the configurations and check the return value
```

4. *SetStreamWriteTimeOut / SetStreamReadTimeOut*

To set the Timeout value before StramWrite and StreamRead.

```
SetStreamWriteTimeOut(&MWBridge, 2000); //To set StreamWrite timeout value, the unit is ms
SetStreamReadTimeOut(&MWBridge, 2000);  // To set StreamRead timeout value, the unit is ms
```

5. StreamWrite

There are two modes of SPI Write: SequentialWrite and StandardWriteWithRead

SequentialWrite: Corresponds to SequentialRead, the usage is similar to SequentialRead

```
DWORD Size = The output data length;           //To set the output data length
BYTE* pBuffer = new BYTE[Size];               //Set the Buffer for output data
DWORD Written = 0;                             //When Function end and return the actual data length have been sent
sOptions Options;                             //Create structure variable
Options.Mode = SPI;                           //Set the transfer mode to SPI
Options.SPI.Mode = SPI_SequentialWrite;       //To set SPI write mode as SPI_SequentialWrite
Options.SPI.WriteDummy = 0x00;                //To set SPI WriteDummy
DWORD Result = StreamWrite(&MWBridge, Size, pBuffer, &Written, Options); //set the configuration, and check the return value
```

When the pBuffer is transmitted, it also starts reading data from slave to Options.pAUXBuffer

```
DWORD Size = The output data length;           //To set the output data length
BYTE* pBuffer = new BYTE[Size];               //Set the Buffer for output data
DWORD Written = 0;                             //When Function end and return the actual data length have been sent
sOptions Options;                             //Create structure variable
Options.Mode = SPI;                           //Set the transfer mode to SPI
Options.SPI.Mode = SPI_StandardWriteWithRead; //To set SPI write mode as SPI_StandardWriteWithRead
Options.SPI.WriteDummy = 0x00;                //To set SPI WriteDummy
Options.SPI.AUXBufferSize = The data length to read; // To set the input data length
Options.SPI.pAUXBuffer = new BYTE[Options.SPI.AUXBufferSize]; //To set the Buffer for input data
DWORD Result = StreamWrite(&MWBridge, Size, pBuffer, &Written, Options); //set the configuration, and check the return value
```

6. StreamRead

There are two modes of SPI Read: SequentialRead and RandomRead

SequentialRead : Corresponds to SequentialWrite, the usage is similar to SequentialWrite

```
DWORD Size = The data length to read;           // To set the input data length
BYTE* pBuffer = new BYTE[Size];                 //To set the Buffer for input data
DWORD Read = 0;                                 //When Function end and return the actual data length have been read
sOptions Options;                               //Create structure variable
Options.Mode = SPI;                             //Set the transfer mode to SPI
Options.SPI.Mode = SPI_SequentialRead;          //To set SPI Read mode as SPI_SequentialRead
Options.SPI.WriteDummy = 0x00;                  // To set SPI WriteDummy
DWORD Result = StreamRead(&MWBridge, Size, pBuffer, &Read, Options); //set the configuration, and check the return value
```

RandomRead : The Options.AUXBuffer will be transmitted first, then Read will start; the application can first send out the address of the target data area wants to read, and then read the data of the location.

```
DWORD Size = The data length to read;           // To set the input data length
BYTE* pBuffer = new BYTE[Size];                 // To set the Buffer for input data
DWORD Read = 0;                                 // When Function end and return the actual data length have been read
sOptions Options;                               //Create structure variable
Options.Mode = SPI;                             //Set the transfer mode to SPI
Options.SPI.Mode = SPI_RandomRead;              //To set SPI Read mode as SPI_RandomRead
Options.SPI.WriteDummy = 0x00;                  //To set SPI WriteDummy
Options.SPI.pAUXBuffer = The buffer address;    //To set the Buffer for output data
Options.SPI.AUXBufferSize = Read data length;  //To set the output data length
DWORD Result = StreamRead(&MWBridge, Size, pBuffer, &Read, Options); //set the configuration, and check the return value
```

7. *DisconnectBridge*

DisconnectBridge before the end of the program, the DLL will release the memory space

```
DWORD Result = DisconnectBridge(&MWBridge);           //Disconnect the bridge and check the return value
```

4.1.4 I2C

1. ConnectBridge

To use this function create the connection between user application and MA111. To create structure variable sMWBridge which include product information and read/write timeout value, send it to ConnectBridge and check the return value ([2.3.26 Error Code Reference](#))

```
sMWBridge MWBridge;           //Create structure variable
MWBridge.VID = 0x0E6A;         //Set VID of the structure variable, Megawin's VID is 0x0E6A
MWBridge.PID = 0x0124;         // Set PID of the structure variable, the PID of MA111 is 0x0124
MWBridge.ReadTimeOut = 1000;   //Set ReadTimeOut value of the structure variable, it can be modified. The unit is ms.
MWBridge.WriteTimeOut = 1000;  //Set WriteTimeOut value of the structure variable, it can be modified. The unit is ms.
DWORD Result = ConnectBridge(&MWBridge); //Send the sMWBridge to ConnectBridge, and check the return value
```

2. SetGPIOMode

UART are using GPIO9 and GPIOA for transfer. It needs to be set as OpenDrainWithPullUpResistor

```
sGPIOModeSetting ModeSetting; //Create structure variable
for (int i = 0; i < 9; i++) {
    ModeSetting.GPIO[i].Setting = Disable //Initial GPIO 0~8, keep it as default
}
ModeSetting.GPIO[9].Setting = Enable; //To set GPIO[9] (TXD) as enable
ModeSetting.GPIO[9].Mode = OpenDrainOutputWithPullUpResistor; //To set GPIO[9] as Open Drain Output with Pullup Resistor
ModeSetting.GPIO[10].Setting = Enable; //To set GPIO[10] (RXD) as enable
ModeSetting.GPIO[10].Mode = OpenDrainOutputWithPullUpResistor //To set GPIO[10] as Open Drain Output with Pullup Resistor
DWORD Result = SetGPIOMode(&MWBridge, ModeSetting); //To use SetGPIOMode set the I/O configuration & check the
return value
```

3. *SetStreamMode*

To set the MA111 transfer mode as I2C, and set the I2C control configurations.

```
sStreamSetting StreamSetting; //Create structure variable
StreamSetting.Mode = I2C; //Assign I2C Mode as the transfer mode for MA111
StreamSetting.I2C.ResetControl = NoAction; // I2C Reset Control
StreamSetting.I2C.SCLTimeoutDetectionControl = Disable; //To set I2C SCL time out detection control
StreamSetting.I2C.SpeedSelection = Speed_25k; //To set I2C speed
StreamSetting.I2C.SCLTimeoutSelection = MoreThan_20ms; //To set I2C time out period
DWORD Result = SetStreamMode(&MWBridge, StreamSetting); //To use SetStreamMode to set the configurations and check the return value
```

4. *SetStreamWriteTimeOut / SetStreamReadTimeOut*

To set the Timeout value before StreamWrite and StreamRead.

```
SetStreamWriteTimeOut(&MWBridge, 2000); //To set StreamWrite timeout value, the unit is ms
SetStreamReadTimeOut(&MWBridge, 2000); // To set StreamRead timeout value, the unit is ms
```

5. *StreamWrite*

I2C Write has only one SequentialWrite mode, which is the most basic mode. It is used as follows.

```
DWORD Size = The output data length; //To set the output data length
BYTE* pBuffer = new BYTE[Size]; //Set the Buffer for output data
DWORD Written = 0; //When Function end and return the actual data length have been sent
sOptions Options; //Create structure variable
Options.Mode = I2C; //Set the transfer mode to I2C
Options.I2C.Mode = I2C_SequentialWrite; //Set the I2C as I2C_SequentialWrite
Options.I2C.Address = 0x01; // Set the address of I2C
DWORD Result = StreamWrite(&MWBridge, Size, pBuffer, &Written, Options); //set the configuration, and check the return value
```


6. StreamRead

I2C Read has two modes SequentialRead and RandomRead.

SequentialRead : Corresponds to SequentialWrite, the usage is similar to SequentialWrite

```
DWORD Size = The data length to read;           // To set the input data length
BYTE* pBuffer = new BYTE[Size];                 //To set the Buffer for input data
DWORD Read = 0;                                 //When Function end and return the actual data length have been read
sOptions Options;                               //Create structure variable
Options.Mode = I2C;                             //Set the transfer mode to I2C
Options.I2C.Mode = I2C_SequentialRead;          //To set I2C as I2C_SequentialRead
Options.I2C.Address = 0x01;                     // To set the address of I2C
DWORD Result = StreamRead(&MWBridge, Size, pBuffer, &Read, Options); //set the configuration, and check the return value
```

RandomRead : The Options.AUXBuffer will be transmitted first, then Read will start; the application can first send out the address of the target data area wants to read, and then read the data of the location.

```
DWORD Size = The data length to read;           // To set the input data length
BYTE* pBuffer = new BYTE[Size];                 // To set the Buffer for input data
DWORD Read = 0;                                 // When Function end and return the actual data length have been read
sOptions Options;                               //Create structure variable
Options.Mode = I2C;                             //Set the transfer mode to I2C
Options.I2C.Mode = I2C_RandomRead;              //To set I2C Read mode as I2C_RandomRead
Options.I2C.Address = 0x01;                     //To set the address of I2C
Options.I2C.pAUXBuffer = The buffer address;    //To set the Buffer for output data
Options.I2C.AUXBufferSize = Read data length;   //To set the output data length
DWORD Result = StreamRead(&MWBridge, Size, pBuffer, &Read, Options); //set the configuration, and check the return value
```

7. *DisconnectBridge*

DisconnectBridge before the end of the program, the DLL will release the memory space

```
DWORD Result = DisconnectBridge(&MWBridge);           //Disconnect the bridge and check the return value
```

5 Revision history

Version	Date	Description
v1.00	2017/11	- Initial version
v1.01	2017/12	- Update figure
v1.02	2018/05	- Update figure
v1.03	2018/11	Add "How to use" and modify "Functions Description"
v1.04	2022/10/18	Modify "TWI" string to "I2C"