Kiwi
Middleware

Date: Feb. 23, 2022 Revision 1.0.1

Table of Contents

Revision History

Revision Number	Description	Data	Editor
V1.0.0	Initial Release	Dec 2021	Edward
V1.0.1	Modify GPIO direction	Feb 2022	Edward

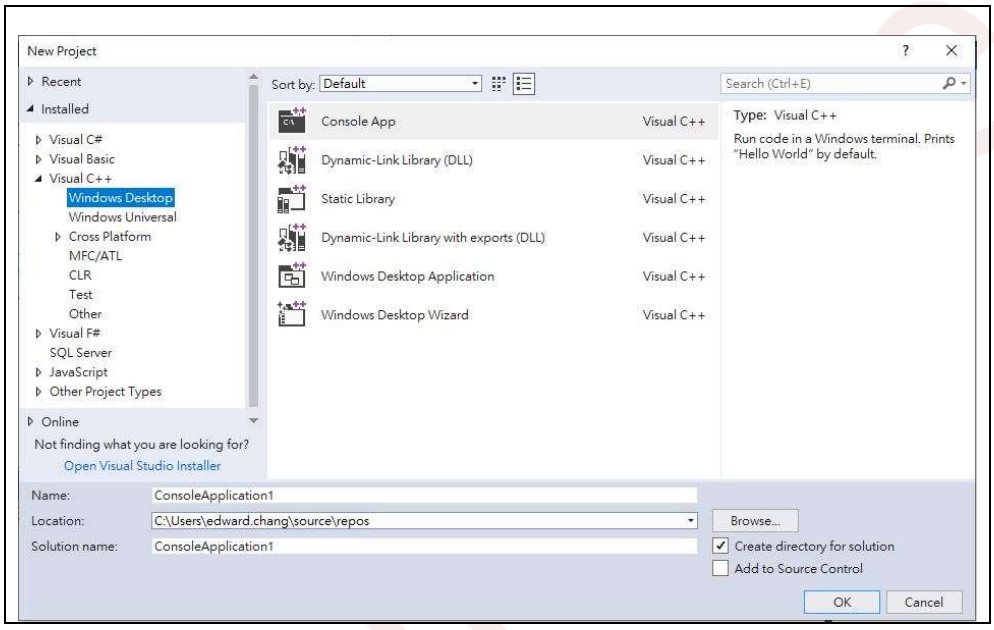
CHAPTER 1
Introduction

This chapter is written to introduce the information of the SDK.

1.1 Creating a Windows Application Using Microsoft Visual C++

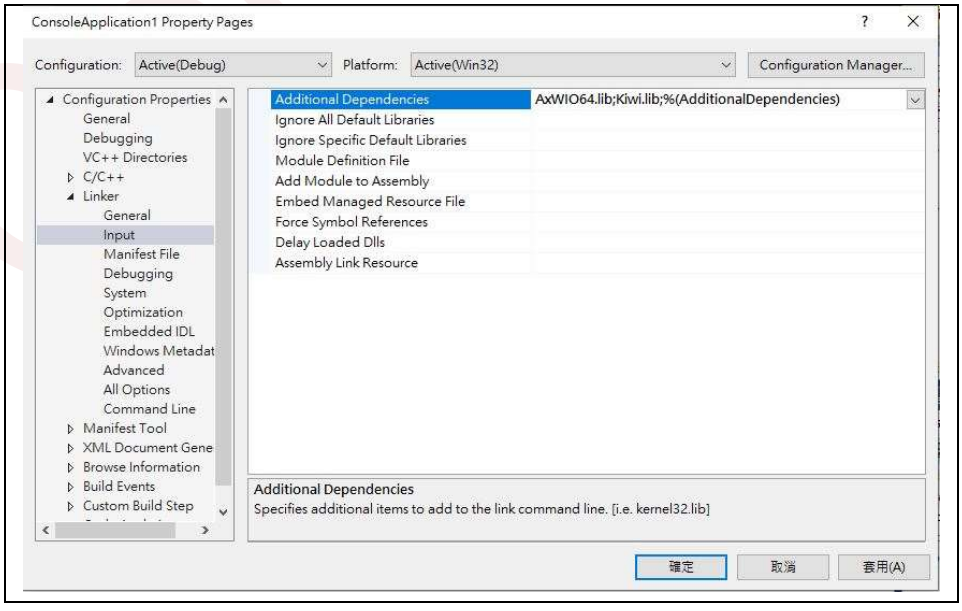
To create an application with SDK and Microsoft Visual C++, please follow the steps below.

Step 1: Create the project.



Step 2: Copy library files (.dll .lib, .h) to your project.

Step 3: Add library name (.lib) to Linker->Input->Additional Dependencies.



Step 4: Include header file in your project.

```
#include <Windows.h>
#include "kiwi310.h"
```

Step 5: Get Gpio config

```
bool status = false;
UINT8 Config = 0;
UINT8 HATNum = 0;
status = GetGpioConfig( HATNum, &Config);
if (status)
{
  printf("Gpio Config = %d\n", Config);
}
Else
{
printf("Failed to get GPIO config\n" );
}
```

Step 6: Build application.

Step 7: Before running the application, the .dll file must be located at the same folder with the application.

Step 8: Run the application with the Microsoft Visual C++ 2017 Redistributable (x86 for 32-bit process or x64 for 64-bit process) are installed

1.2 Creating a Linux Application Using Ubuntu

Step 1: Copy library files (.so, .h) to your project.

Step 2: Install kernel library by Internet

Step 3: Include header file in your project.

```
#include "kiwi310.h"
```

Step 4: gcc -g -o Test Test.cpp kiwi.so

CHAPTER 2
Function Introduction

This chapter describes the detail information of the SDK functions.

2.1 GetGpioConfig

- **Description**

  Get the GPIO configuration.

- **Definition**

```
bool GetGpioConfig( uint8_t HATNum, uint8_t *Config );
```

- **Parameters**

HATNum	The number of GPIOx. (x = 4, 5, 6, 14, 15, 16, 17,18, 19, 20, 21, 22, 23, 24, 25, 26, 27)
Config	The config of GPIOx. 0 : Disable 1 : Enable

- **Return value**

  True if success, Otherwise fail.

- **Example**

```
bool status = false;
// Get Gpio Config
UINT8 Config = 0;
UINT8 HATNum = 4;
status = GetGpioConfig( HATNum, &Config);
if (status)
{
printf("Gpio4 Config = %d\n", Config);
}
else
{
printf("Failed to get GPIO4 config\n" );
}
```

2.2 GetGpioMode

- **Description**

  Get each GPIO direction.

- **Definition**

```
bool GetGpioMode( uint8_t HATNum, uint8_t *Mode );
```

- **Parameters**

  HATNum The number of GPIOx. (x = 4, 5, 6, 14, 15, 16, 17,18, 19, 20, 21, 22, 23, 24, 25, 26, 27)

  Mode The mode of GPIOx.

  0 : Output

  1 : input

- **Return value**

  True if success, Otherwise fail.

- **Example**

```
bool status = false;
// Get Gpio Mode
UINT8 Mode = 0;
UINT8 HATNum = 4;
status = GetGpioMode ( HATNum, &Mode);
if (status)
{
printf("Gpio4 Mode = %d\n", Mode);
}
else
{
printf("Failed to get GPIO4 Mode \n" );
}
```

2.3 SetGpioMode

- **Description**

  Set each GPIO direction.

- **Definition**

```
bool SetGpioMode(uint8_t HATNum, uint8_t Mode);
```

- **Parameters**

  HATNum The number of GPIOx. (x = 4, 5, 6, 14, 15, 16, 17,18, 19, 20, 21, 22, 23, 24, 25, 26, 27)

  Mode The mode of GPIOx.

  0 : Output

  1 : Input

- **Return value**

  True if success, Otherwise fail.

- **Example**

```
bool status = false;
// Set Gpio Mode
UINT8 Mode = 0;
UINT8 HATNum = 4;
status = GetGpioMode ( HATNum, Mode);
if (status)
{
printf("Gpio4 Mode = %d\n", Mode);
}
else
{
printf("Failed to Set GPIO4 Mode \n" );
}
```

2.4 GetGpioStatus

- **Description**

  Get each GPIO status.

- **Definition**

```
bool GetGpioStatus (uint8_t HATNum, uint8_t *Status);
```

- **Parameters**

HATNum	The number of GPIOx. (x = 4, 5, 6, 14, 15, 16, 17,18, 19, 20, 21, 22, 23, 24, 25, 26, 27)
Status	The status of GPIOx. 0 : Low 1 : High

- **Return value**

  True if success, Otherwise fail.

- **Example**

```
bool status = false;
// Get Gpio status
UINT8 GpioStatus = 0;
UINT8 HATNum = 4;
status = GetGpioStatus ( HATNum, &GpioStatus);
if (status)
{
printf("Gpio4 status = %d\n", GpioStatus);
}
else
{
printf("Failed to get GPIO4 status \n" );
}
```

2.5 SetGpioStatus

- **Description**

  Set each GPIO status.

- **Definition**

```
bool SetGpioStatus (uint8_t HATNum, uint8_t Status);
```

- **Parameters**

HATNum	The number of GPIOx. (x = 4, 5, 6, 14, 15, 16, 17,18, 19, 20, 21, 22, 23, 24, 25, 26, 27)
Status	The status of GPIOx. 0 : Low 1 : High

- **Return value**

  True if success, Otherwise fail.

- **Example**

```
bool status = false;
// Get Gpio status
UINT8 GpioStatus = 0;
UINT8 HATNum = 4;
status = SetGpioStatus ( HATNum, GpioStatus);
if (status)
{
printf("Gpio4 status = %d\n", GpioStatus);
}
else
{
printf("Failed to get GPIO4 status \n" );
}
```

2.6 SetPwmStatus

- **Description**

  Set each PWM status.

- **Definition**

```
bool SetPwmStatus(uint8_t PwmNum, uint8_t PwmStatus);
```

- **Parameters**

  PwmNum PWM port number (Pwm0 = 0, Pwm1=1).

  PwmStatus The status of PWM.

  0: Disable.

  1: Enable.

- **Return value**

  True if success, Otherwise fail.

- **Example**

```
bool status = false;
// Get Pwm status
UINT8 PwmNum = 0;
UINT8 PwmStatus = 0;
status = SetPwmStatus ( PwmNum, PwmStatus );
if (status)
{
printf("Pwm status = %d\n", Pwmstatus );
}
else
{
printf("Failed to set pwm status \n" );
}
```

2.7 SetPwmFrequency

- **Description**

  Set each PWM frequency.

- **Definition**

```
bool SetPwmFrequency (uint8_t PwmNum, uint32_t PwmFrequency);
```

- **Parameters**

  PwmNum PWM port number. (Pwm 0 = 0, Pwm 1=1)

  PwmFrequency The frequency of PWM.

  Range 1000 ~ 200000 (1KHz ~ 200KHz)

- **Return value**

  True if success, Otherwise fail.

- **Example**

```
bool status = false;
// Set Pwm frequency
UINT8 PwmNum = 0;
UINT32 PwmFrequency = 0x2710; //10000(Hz)
status = SetPwmFrequency ( PwmNum, PwmFrequency);
if (status)
{
printf("Pwm frequency = %d\n", PwmFrequency);
}
else
{
printf("Failed to set pwm frequency \n" );
}
```

2.8 SetPwmDutyCycle

- **Description**

  Set each PWM duty cycle.

- **Definition**

```
bool SetPwmDutyCycle (uint8_t PwmNum, uint8_t PwmDutyCycle);
```

- **Parameters**

PwmNum	Pwm port number. (Pwm 0 = 0, Pwm 1=1)
PwmDutyCycle	The duty cycle of Pwm. Range 0 ~ 100 (%)

- **Return value**

  True if success, Otherwise fail.

- **Example**

```
bool status = false;
// Set Pwm frequency
UINT8 PwmNum = 0;
UINT32 PwmDutyCycle = 50;
status = SetPwmDutyCycle ( PwmNum, PwmDutyCycle);
if (status)
{
printf("Pwm duty cycle = %d\n", PwmDutyCycle);
}
else
{
printf("Failed to set pwm duty cycle \n" );
}
```

2.9 GetPwmStatus

- **Description**

  Get each PWM status.

- **Definition**

```
bool GetPwmStatus (uint8_t PwmNum, uint8_t *PwmStatus);
```

- **Parameters**

  PwmNum PWM port number. (Pwm 0 = 0, Pwm 1=1)

  PwmStatus The status of PWM.

  0: Disable.

  1: Enable.

- **Return value**

  True if success, Otherwise fail.

- **Example**

```
bool status = false;
// Get Pwm status
UINT8 PwmNum = 0;
UINT8 PwmStatus = 0;
status = GetPwmStatus ( PwmNum, &PwmStatus );
if (status)
{
printf("Pwm status = %d\n", Pwmstatus );
}
else
{
printf("Failed to set pwm status \n" );
}
```

2.10 GetPwmFrequency

Description

Get each PWM frequency.

- **Definition**

```
bool GetPwmFrequency (uint8_t PwmNum, uint32_t *PwmFrequency);
```

- **Parameters**

PwmNum PWM port number. (Pwm 0 = 0, Pwm 1=1)

PwmFrequency The frequency of PWM.

Range 1000 ~ 200000 (1KHz ~ 200KHz)

- **Return value**

True if success, Otherwise fail.

- **Example**

```
bool status = false;
// Set Pwm frequency
UINT8 PwmNum = 0;
UINT32 PwmFrequency = 0x0;
status = GetPwmFrequency ( PwmNum, &PwmFrequency);
if (status)
{
printf("Pwm frequency = %d\n", PwmFrequency);
}
else
{
printf("Failed to set pwm frequency \n" );
}
```

2.11 GetPwmDutyCycle

- **Description**

  Get each PWM duty cycle.

- **Definition**

```
bool GetPwmDutyCycle (uint8_t PwmNum, uint8_t PwmDutyCycle);
```

- **Parameters**

  PwmNum Pwm port number. (Pwm 0 = 0, Pwm 1=1)

  PwmDutyCycle The duty cycle of Pwm.

  Range 0 ~ 100 (%)

- **Return value**

  True if success, Otherwise fail.

- **Example**

```
bool status = false;
// Get Pwm frequency
UINT8 PwmNum = 0;
UINT32 PwmDutyCycle = 0;
status = GetPwmDutyCycle ( PwmNum, &PwmDutyCycle);
if (status)
{
printf("Pwm duty cycle = %d\n", PwmDutyCycle);
}
else
{
printf("Failed to set pwm duty cycle \n" );
}
```

2.12 GetI2cConfig

- **Description**

  Get I2c configuration.

- **Definition**

```
bool GetI2cConfig(uint8_t *Enable, uint8_t *Speed, uint8_t *Rs);
```

- **Parameters**

Enable	Switch control I2c interface. 0: Disable 1: Enable
Speed	The speed of I2c. 0: 100KHz 1: 400KHz
Rs	Repeat start of I2c. 0: Disable 1: Enable

- **Return value**

  True if success, Otherwise fail.

- **Example**

```
bool status = false;
// Get Pwm frequency
UINT8 Enable = 0;
UINT8 Speed = 0;
UINT8 Rs = 0;
status = GetI2cConfig ( &Enable, &Speed, &Rs );
if (status)
{
printf("I2c Enable = %d speed = %d Rs = %d \n", Enable, Speed, Rs);
}
else
{
printf("Failed to get I2c configuration.\n" );
}
```

2.13 SetI2cConfig

- **Description**

  Set I2c configuration.

- **Definition**

```
bool SetI2cConfig(uint8_t Enable, uint8_t Speed, uint8_t Rs);
```

- **Parameters**

Enable	Switch control I2c interface. 0: Disable 1: Enable
Speed	The speed of I2c. 0: 100KHz 1: 400KHz
Rs	Repeat start of I2c. 0: Disable 1: Enable

- **Return value**

  True if success, Otherwise fail.

- **Example**

```
bool status = false;
// Set I2c state
UINT8 Enable = 0;
UINT8 Speed = 0;
UINT8 Rs = 0;
status = SetI2cConfig ( Enable, Speed, Rs );
if (status)
{
printf("I2c Enable = %d speed = %d Rs = %d \n", Enable, Speed, Rs);
}
else
{
printf("Failed to get I2c configuration.\n" );
}
```

2.14 GetSPIConfig

- **Description**

  Get SPI configuration.

- **Definition**

```
bool GetSPIConfig (uint8_t *Enable, uint8_t *Mode, uint8_t *DataOrder, uint8_t
*Speed);
```

- **Parameters**

Enable	Switch control SPI interface. 0: Disable 1: Enable
Mode	The mode of SPI. 0: CPOL=0, CPHA=0 1: CPOL=0, CPHA=1 2: CPOL=1, CPHA=0 3: CPOL=1, CPHA=1
DataOrder	Data Order of SPI. 0: MSB 1: LSB
Speed	Speed of SPI. 0: 1MHz 1: 2MHz 2: 4MHz 3: 8MHz 4: 16MHz

- **Return value**

  True if success, Otherwise fail.

- **Example**

```
bool status = false;
// Get Pwm frequency
UINT8 SPIEnable = 0;
UINT8 SPIMode = 0;
UINT8 DataOrder = 0;
UINT8 SPISpeed = 0;
status = GetSPIConfig(&SPIEnable, &SPIMode, &DataOrder, &SPISpeed);
if (status)
{
printf("SPI Enable = %d, Mode = %d, Data Order = %d, speed: %d\n", SPIEnable, SPIMode,
DataOrder, SPISpeed);
```

```
}
else
{
printf("Failed to get spi configuration\n" );
}
```

2.15 SetSPIConfig

- **Description**

  Get SPI configuration.

- **Definition**

```
bool SetSPIConfig (uint8_t Enable, uint8_t Mode, uint8_t DataOrder, uint8_t Speed);
```

- **Parameters**

  Enable Switch control SPI interface.

  0: Disable

  1: Enable

  Mode The mode of SPI.

  0: CPOL=0, CPHA=0

  1: CPOL=0, CPHA=1

  2: CPOL=1, CPHA=0

  3: CPOL=1, CPHA=1

  DataOrder Data Order of SPI.

  0: MSB

  1: LSB

  Speed Speed of SPI.

  0: 1MHz

  1: 2MHz

  2: 4MHz

  3: 8MHz

  4: 16MHz

- **Return value**

  True if success, Otherwise fail.

- **Example**

```
bool status = false;
// Get Pwm frequency
UINT8 SPIEnable = 0;
UINT8 SPIMode = 0;
UINT8 DataOrder = 0;
UINT8 SPISpeed = 0;
status = SetSPIConfig (SPIEnable, SPIMode, DataOrder, SPISpeed);
if (status)
{
printf("SPI Enable = %d, Mode = %d, Data Order = %d, speed: %d\n", SPIEnable, SPIMode,
DataOrder, SPISpeed);
```

```
}
else
{
printf("Failed to get spi configuration\n" );
}
```

2.16 AccessI2c

- **Description**

  Access I2c function.

- **Definition**

```
bool AccessI2c(uint8_t Address, uint8_t wSize, uint8_t *wData, uint8_t rSize,
               uint8_t *rData);
```

- **Parameters**

Address	Address of slave device.(8bit)
wSize	Size of write data.
wData	Write Data.
rSize	Size of read data.
rData	Read Data.

- **Return value**

  True if success, Otherwise fail.

- **Example**

```
bool status = false;
uint8_t Address = 0;
uint8_t wSize = 0;
uint8_t *wData = 0;
uint8_t rSize = 0;
uint8_t *rData = 0;
wData = (uint8_t*)malloc(1 , sizeof(uint8_t));
rData = (uint8_t*) malloc (3 , sizeof(uint8_t));
wSize = 1;
rSize = 3;
wData[0] = 0xE3;
Address = 0x80;
status  = AccessI2c(Address, wSize, wData, rSize, rData);
if (status)
{
    for(int i = 0 ; I < 3 ; i++ )
        printf("Read Data%d = %x", i, rData[i] );
}
else
{
printf("Failed to i2c access fail\n" );
}
```

2.17 AccessSPI

- **Description**

  Access SPI function.

- **Definition**

```
bool AccessSPI (uint8_t Chipselect, uint8_t wSize, uint8_t *wData, uint8_t rSize,
                uint8_t *rData);
```

- **Parameters**

Chipselect	Pin Number of chip select.(0/1)
wSize	Size of write data
WData	Write data
rSize	Size of read data
rData	Read Data

- **Return value**

  True if success, Otherwise fail.

- **Example**

```
bool status = false;
uint8_t Chipsel= 0;
uint8_t wSize = 0;
uint8_t *wData = 0;
uint8_t rSize = 0;
uint8_t *rData = 0;
wData = (uint8_t*)malloc(1 , sizeof(uint8_t));
rData = (uint8_t*) malloc (3 , sizeof(uint8_t));
wSize = 1;
rSize = 3;
wData[0] = 0x90;
Chipsel = 0x0;
status  = AccessSPI (Address, wSize, wData, rSize, rData);
if (status)
{
    for(int i = 0 ; I < 5 ; i++ )
        printf("Read Data%d = %x", i, rData[i] );
}
else
{
printf("Failed to spi access fail\n" );
}
```