



RoBoard RB-100 Software Development Introduction for RoBoIO Library V1.5

DMP Electronics Inc
Robotics Division
May 13, 2009



Overview

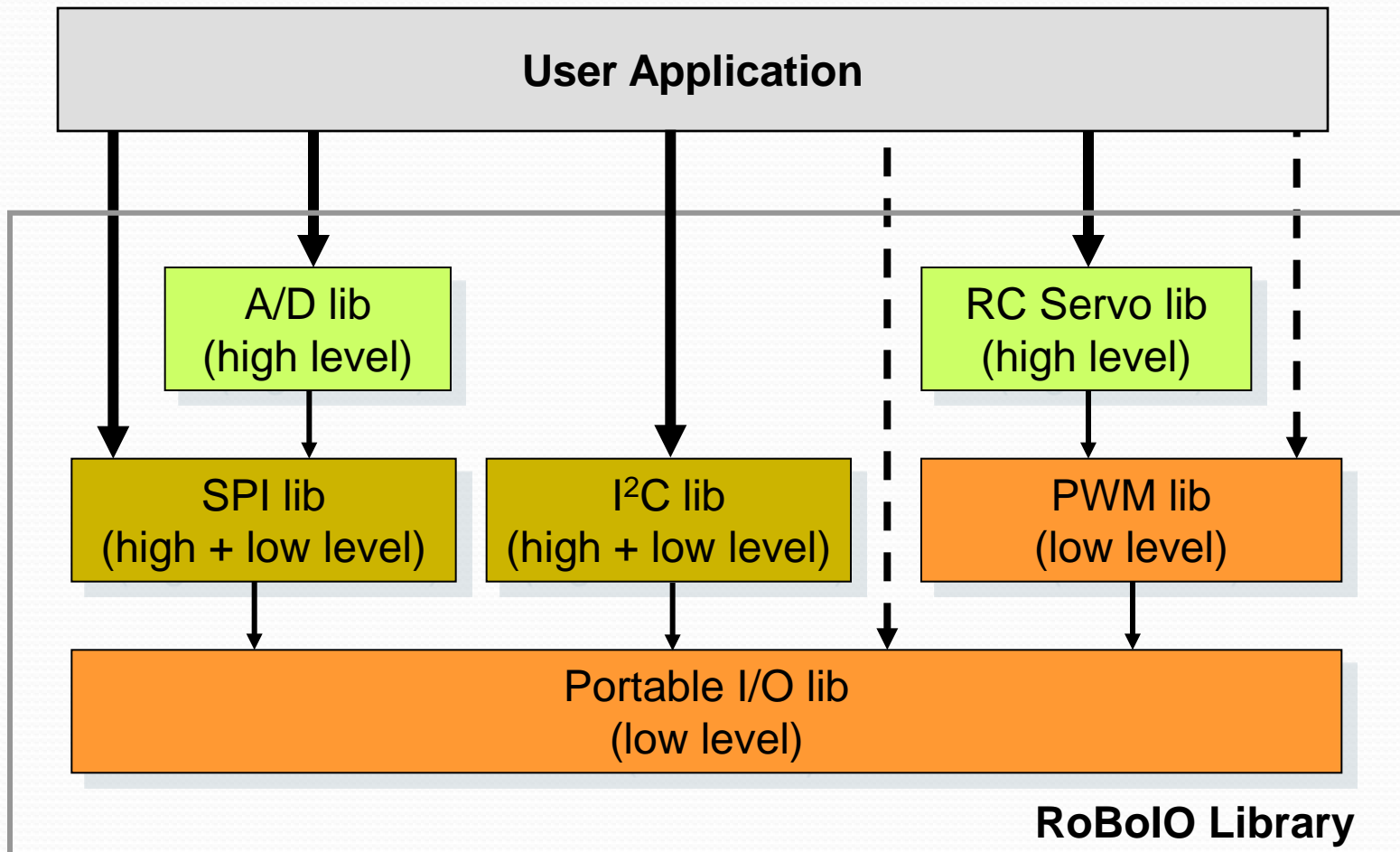
RoBoIO Library

- A **open-source** library for RoBoard's unique I/O functions
 - Free for academic & commercial use
- Supported functions
 - SPI
 - A/D
 - PWM
 - I²C

RoBoIo Library

- Supported functions (cont.)
 - GPIO
 - RC servo control (KONDO, HiTEC, ...)
- Supported platforms
 - **Windows XP**: Visual C++ 2005/2008
 - require **WinIo** or **PciDebug**
 - **Windows CE**: Visual C++ 2005/2008
 - **Linux**: gcc
 - **DOS**: DJGPP, Borland C++ 3.0

Architecture



Usage Overview

- Include **roboard.h** to use
 - SPI lib
 - A/D lib
 - I²C lib
 - RC Servo lib
- Note: cannot use PWM lib when using RC Servo lib
 - because RC Servo lib will manage PWM lib

```
#include <roboard.h>
int main() {
    .....
    // use API of RoBoIO
    // library here
    .....
    return 0;
}
```

Usage Overview

- Include `roboard_dll.h` instead if you use RoBoIO DLL version
- The DLL version is only available on
 - Windows XP
 - Windows CE

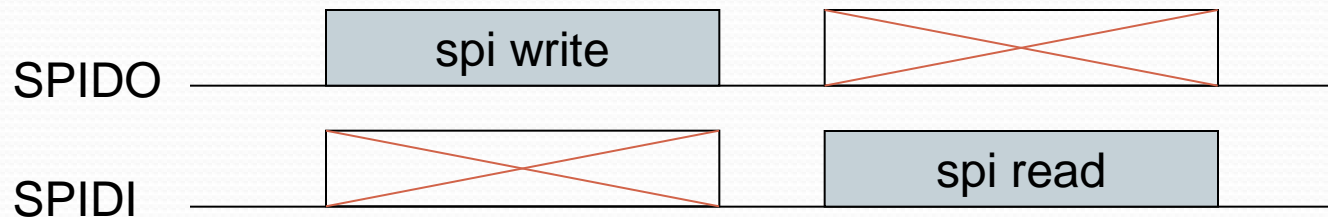
```
#include <roboard_dll.h>
int main() {
    .....
    // use API of RoBoIO
    // library here
    .....
    return 0;
}
```



SPI lib

RoBoard SPI Features & Limits

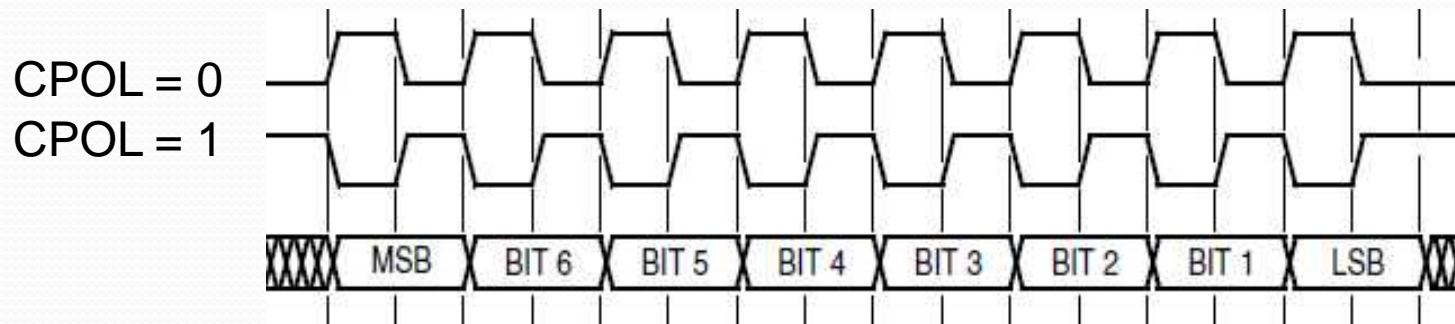
- Dedicated to SPI flash
- Half-Duplex



- Support only high-speed devices
 - Max: 150 Mbps
 - Min: 10 Mbps

RoBoard SPI Features & Limits

- Support only two clock modes
 - CPOL = 0, CPHA = 1 Mode
 - CPOL = 1, CPHA = 1 Mode



- See http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus for more information about SPI modes.

Usage Overview

```
if (spi_initialize(clock_mode)) {  
    .....  
    unsigned val = spi_Read(); //read a byte from SPI bus  
    spi_Write(0x55); //write a byte (0x55) to SPI bus  
    .....  
    spi_Close(); //close SPI lib  
}
```

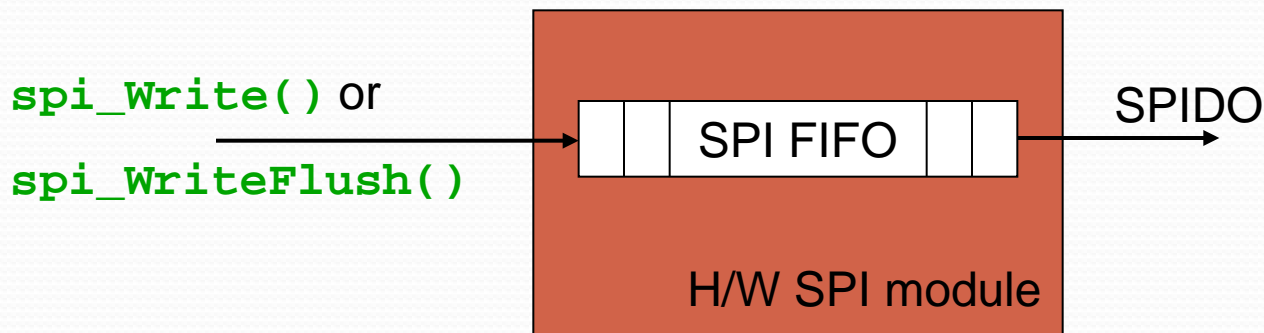
- `clock_mode` can be, e.g.,
 - `SPICLK_10000KHZ` (10 Mbps)
 - `SPICLK_12500KHZ` (12.5 Mbps)
 - `SPICLK_21400KHZ` (21.4 Mbps)
 - `SPICLK_150000KHZ` (150 Mbps)
- See `spi.h` for all available clock modes.

SPI-Write Functions

- Two different SPI-write functions:

`spi_Write()` vs. `spi_WriteFlush()`

- All data are written to SPI FIFO, and then transferred by Hardware.



SPI-Write Functions

- Two different SPI-write functions: (cont.)
 - `spi_Write()` does not wait transfer completion.
 - Faster
 - But must be careful about timing issue
 - Can call `spi_FIFOFlush()` to flush SPI FIFO
 - `spi_WriteFlush()` waits that SPI FIFO becomes empty.

SPISS Pin

- Control the **SPISS** pin
 - Set SPISS to 0: `spi_EnableSS()`
 - Set SPISS to 1: `spi_DisableSS()`
- Usually used for turning on/off SPI devices
 - If need more than one SPISS pin, simulate them using RoBoard's GPIO



A/D lib

RoBoard A/D Features

- 10-bit resolution & 1M samples per second
- Share RoBoard's SPI bus
 - When accessing A/D, signals appear on **SPICLK**, **SPIDO**, **SPIDI** pins.
 - So be careful about **bus conflict** if you have devices attached to SPI bus
 - Disable your SPI devices (using, e.g., **SPISS** pin) when accessing A/D

Usage Overview

```
if (spi_initialize(SPICLK_21400KHZ)) {  
    .....  
    int val = ad7918_ReadChannel(channel, //channel = 0 ~ 7  
        AD7918MODE_RANGE_2VREF, AD7918MODE_CODING_1023);  
    .....  
    spi_close();  
}
```

- To use the 8-channel A/D, we must initialize SPI lib first.
 - SPI clock must < **21.4** Mbps
- Only provides usual AD7918 functions
 - Refer to AD7918 datasheet if you want to extend A/D lib.

Usage Overview

- Input-voltage range:
 - **AD7918MODE_RANGE_2VREF**: 0V ~ 5V
 - allow higher voltage
 - **AD7918MODE_RANGE_VREF**: 0V ~ 2.5V
 - allow higher resolution
- A/D value range:
 - **AD7918MODE_CODING_1023**: 0 ~ 1023
 - **AD7918MODE_CODING_511**: -512 ~ 511
 - min value \Rightarrow lowest voltage, max value \Rightarrow highest voltage

Batch Mode

- `ad7918_ReadChannel()` is slower due to channel-addressing overhead.
- In batch mode, multiple channels are read without channel-addressing ⇒ **better performance**
 - `ad7918_InitializeMCH()`: open batch mode
 - `ad7918_ReadMCH()`: read user-assigned channels
 - `ad7918_CloseMCH()`: close batch mode

Batch Mode

```
int* ad_data;
if (ad7918_InitializeMCH(AD7918_USECHANNEL0 +
                        AD7918_USECHANNEL1 + ...,
                        AD7918MODE_RANGE_2VREF,
                        AD7918MODE_CODING_1023)) {

    .....
    ad_data = ad7918_ReadMCH();
    for (i=0; i<8; i++)
        printf("A/D channel %d = %d", i, ad_data[i]);

    .....
    ad_data = ad7918_ReadMCH();

    .....
    ad7918_CloseMCH();
}
```



I²C lib

RoBoard I²C Features

- Support both master & slave modes
- Support 10-bit address (master only)
- Three speed modes
 - **standard mode** (~100 Kbps)
 - **fast mode** (~400 Kbps)
 - must pull-up **I2C0_SCL**, **I2C0_SDA** pins
 - **high-speed mode** (~3.3 Mbps)
 - To achieve 3.3 Mbps, pull-up resistors should $\leq 1\text{K ohm}$

Usage Overview: Master Mode

```
if (i2c_Initialize(I2CIRQ_DISABLE)) {  
    i2c0_SetSpeed(speed_mode, bps);  
  
    .....  
    //use master API of I2C lib here  
  
    .....  
    i2c_Close(); //close I2C lib  
}
```

- **speed_mode** can be
 - I2CMODE_STANDARD: standard mode
 - I2CMODE_FAST: fast mode
 - I2CMODE_HIGHSPEED: high-speed mode
 - I2CMODE_AUTO: set speed mode according to **bps**
- **bps** can be any integer ≤ 33000000 (3.3 Mbps)

Master API

- The most simple API
 - `I2C0master_StartN()`: send START signal to slave devices
 - `I2C0master_WriteN()`: write a byte to slave devices
 - `I2C0master_ReadN()`: read a byte from slave devices
 - Automatically send **STOP** signal after reading/writing the last byte

```
i2c0master_StartN(0x30, //to slave with address 0x30;  
                  I2C_WRITE, //use I2C_READ instead, for read  
                  3); //3 bytes to write  
  
i2c0master_WriteN(0x11);  
i2c0master_WriteN(0x22);  
i2c0master_WriteN(0x33);
```

Master API

- Send **RESTART** instead of **STOP**
 - Call `i2c0master_SetRestart()` before the first reading/writing
 - Will send RESTART signal instead after reading/writing the last byte

```
i2c0master_StartN(0x30, I2C_WRITE, 2);  
  
//set to RESTART (after writing) for reading 1 bytes  
i2c0master_SetRestart(I2C_READ, 1);  
i2c0master_WriteN(0x44);  
i2c0master_WriteN(0x55); //auto. send RESTART after this  
  
data = i2c0master_ReadN(); //auto. send STOP after this
```

Usage Overview: Slave Mode

```
if (i2c_Initialize(I2CIRQ_DISABLE)) {  
    i2c0_SetSpeed(speed_mode, bps);  
  
    //set slave address as, e.g., 0x30  
    i2c0slave_SetAddr(0x30);  
  
    .....  
    //slave event loop here  
  
    .....  
    i2c_Close(); //close I2C lib  
}
```

- In the event loop, use slave API to listen and handle I²C bus events.

Slave Event Loop

```
while (.....) {  
    switch (i2c0slave_Listen()) {  
        case I2CSLAVE_START: //receives START signal  
            .....; break;  
        case I2CSLAVE_WRITEREQUEST: //request slave to write  
            //handle write request  
        case I2CSLAVE_READREQUEST: //request slave to read  
            //handle read request  
        case I2CSLAVE_END: //receives STOP signal  
            .....; break;  
    }  
    ..... //can do something here when listening  
}
```

Slave Read/Write API

- Call `i2c0slave_Write()` for sending a byte to master

```
.....  
case I2CSLAVE_WRITEREQUEST:  
    i2c0slave_Write(byte_value);  
    break;  
.....
```

- Call `i2c0slave_Read()` for reading a byte from master

```
.....  
case I2CSLAVE_READREQUEST:  
    data = i2c0slave_Read();  
    break;  
.....
```



RC Servo lib

(with GPIO functions)

RC Servo lib Usage

```
if (rcservo_initialize(RCSERVO_USECHANNEL0 +
                    RCSERVO_USECHANNEL1 + .....)) {
    //use API of RC Servo lib here
    rcservo_close();
}
```

- **RCSERVO_USECHANNEL0 ~ RCSERVO_USECHANNEL23** are available
- Three operation modes
 - **Capture mode** (for reading RC servo's position feedback)
 - **Action playing mode** (for playing user-defined motions)
 - **PWM mode** (send PWM pulses for individual channels)

RC Servo lib Usage

- Capture mode
 - Call `rcservo_EnterCaptureMode()` to enter this mode
 - This is the initial mode after `rcservo_Initialize()`.
- Available API in Capture mode
 - `rcservo_ReadPosition()`: read position feedback of an RC servo motor
 - `rcservo_ReadPositions()`: read feedback of multiple RC servo motors

RC Servo lib Usage

```
rcservo_EnterCaptureMode();  
.....  
unsigned long pos = rcservo_ReadPosition(channel, 0);  
.....  
unsigned long motion[32];  
rcservo_ReadPositions(RCSERVO_USECHANNEL0 +  
                      RCSERVO_USECHANNEL1 + .....,  
                      0,  
                      motion);
```

- The second argument is servo feedback command, and is usually 0.
- if fail to read, return **0xffffffffL**

RC Servo lib Usage

- Action playing mode
 - Call `rcservo_EnterPlayMode()` to enter this mode
 - Used to replay motions captured by `rcservo_ReadPositions()`
- Available API in Action playing mode
 - `rcservo_SetAction()`: set the motion that is ready to play
 - `rcservo_PlayAction()`: rotate all servo motors to the motion set by `rcservo_SetAction()`
 - Must call `rcservo_PlayAction()` repeatedly until it returns `RCSERVO_PLAYEND`

RC Servo lib Usage

```
unsigned long motion[32];
.....
//set motion for playing
.....
rcservo_EnterPlayMode();
.....
rcservo_SetAction(motion, 500); //play motion in 500ms
while (rcservo_PlayAction() != RCSE servo_PLAYEND) {
    //
    //can do something here when playing motion
    //
}
```

RC Servo lib Usage

- PWM mode
 - Call `rcservo_EnterPWMMode()` to enter this mode
 - In this mode, all used PWM channels remain to be low if no pulse is sent.
- Available API in PWM mode
 - `rcservo_SendPWMPulses()`: send a given number of pulses with specific duty and period
 - `rcservo_IsPWMCompleted()`: return true when all PWM pulses have been sent out

RC Servo lib Usage

```
rcservo_EnterPWMMode( );  
  
.....  
unsigned long PWM_period = 10000; //10000us  
unsigned long PWM_duty   = 1500;  //1500us  
unsigned long count      = 100;  
rcservo_SendPWMPulses(channel,  
                        PWM_period, PWM_duty, count);  
while (!rcservo_IsPWMCompleted(channel)) {  
    //  
    //can do something here when sending PWM pulses  
    //  
}
```

RC Servo lib Usage

- Unused PWM channels can be treated as GPIO.
 - Used channels are decided when calling `rcservo_Initialize()`.
 - Available GPIO API:

```
rcservo_Outp(channel, out_value); //out_value = 0 or 1  
int in_value = rcservo_Inp(channel);
```

- Do nothing if `channel` is a used channel



PWM lib

PWM lib Usage

- Allow users to employ complete RoBoard's PWM features
 - Control PWM output waveform
 - Control PWM resolution (minimum PWM duty = **20ns**)
 - Enable PWM interrupt
 -
- Do not use PWM lib when RC Servo lib is in use
- See **pwm.h** and **pwmdx.h** for available API



COM ports

COM Ports

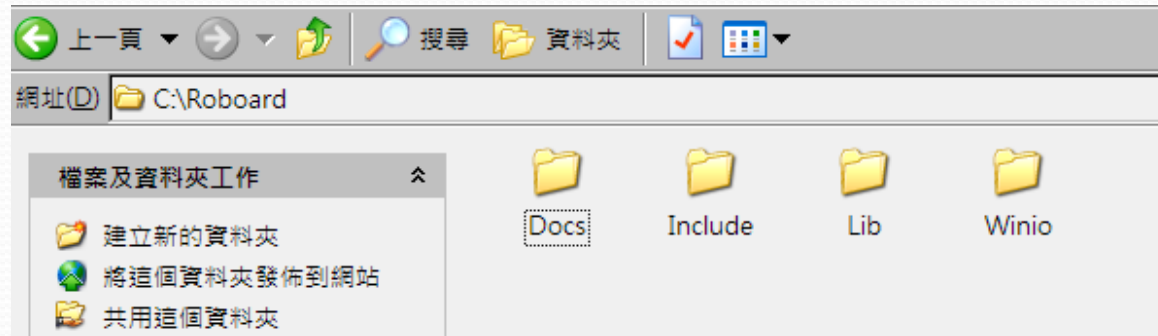
- Can be used as standard COM ports in WinXP, Linux, and DOS
- Max speed is **115200** bps.
- Can customize each COM port in BIOS
 - **IRQ**
 - **I/O base address**
 - **Default speed**



Installation

Setup in VC2005/2008

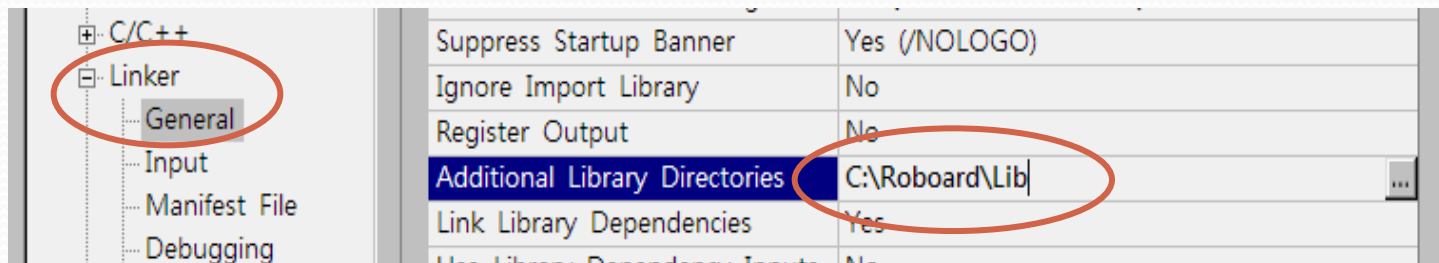
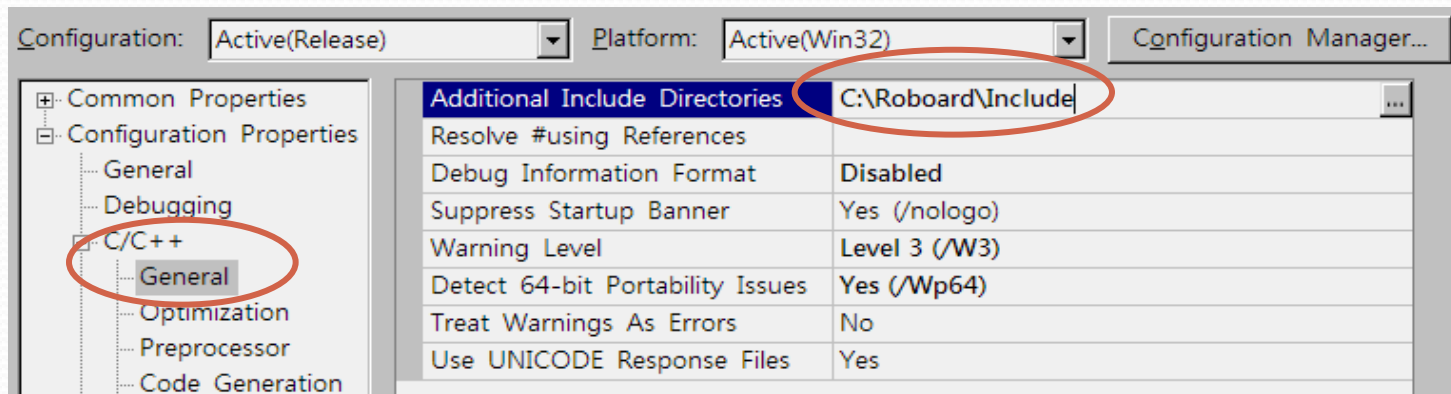
- Decompose RoBoIO bin zip-file to, e.g., **C:\Roboard**



- **Docs**: datasheets of RoBoard's unique I/O
- **Include**: include files of RoBoIO library
- **Lib**: binary of RoBoIO library
- **Winio**: needed when using RoBoIO under WinXP

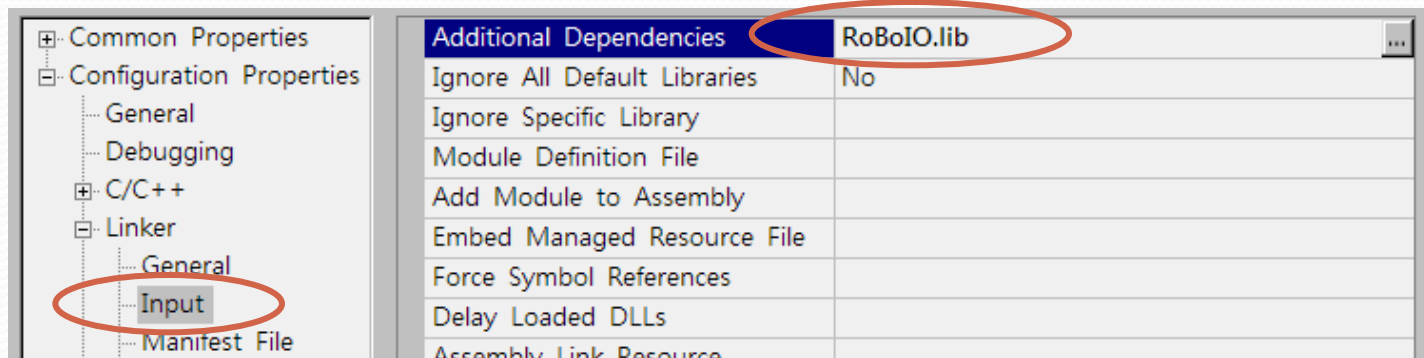
Setup in VC2005/2008

- Setting RoBoIO in your VC2005/2008 project

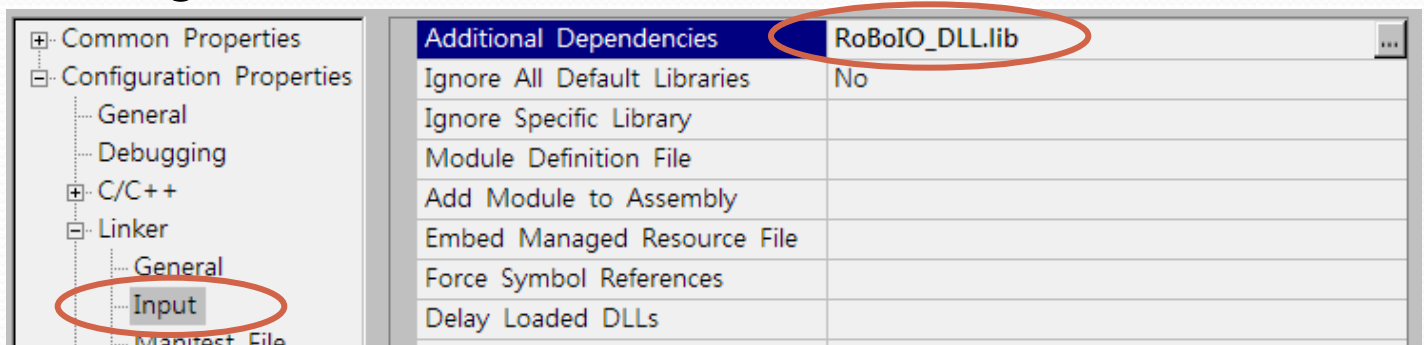


Setup in VC2005/2008

- Setting RoBoIO in your VC2005/2008 project (cont.)
 - If using the static version



- If using the DLL version



Setup in VC2005/2008

- If you use .NET

The screenshot shows the Visual Studio project properties dialog box. The left pane shows the 'Configuration Properties' tree with 'General' selected and circled in red. The right pane shows the 'General' tab settings, with the 'Common Language Runtime Support' setting circled in red.

General	
Output Directory	\$(SolutionDir)\$(ConfigurationName)
Intermediate Directory	\$(ConfigurationName)
Extensions to Delete on Clean	*.obj;*.ilk;*.tlb;*.tli;*.tlh;*.tmp;*.rsp;*.pgc;*.pgd;\$(TargetDir)*
Build Log File	\$(IntDir)\BuildLog.htm
Inherited Project Property Sheets	
Project Defaults	
Configuration Type	Application (.exe)
Use of MFC	Use Standard Windows Libraries
Use of ATL	Not Using ATL
Minimize CRT Use in ATL	No
Character Set	Use Unicode Character Set
Common Language Runtime Support	Common Language Runtime Support (/clr)
Whole Program Optimization	Use Link Time Code Generation

Setup in VC2005/2008

- To run your RoBoIO application on WinXP:
 1. First install **VC2005/2008 runtime redistribution** in RoBoard
 2. Copy your application to RoBoard's storage (the MicroSD or USB storage)
 3. Copy all files in **Roboard\Winio** to your application directory or Window's **System32** directory
- RoBoIO recognizes RoBoard's CPU, and doesn't run on other PC.



Applications

Introduction

- **x86-based** ⇒ Almost all resources on PC can be employed as development tools of RoBoard.
 - **Languages:** C/C++, Visual Basic, Java, Python, Matlab, ...
 - **Libraries:** OpenCV, SDL, Allegro, ...
 - **IDE:** Visual Studio, Dev-C++, ...
 - **GUI (if needed):** MFC, .Net Framework, wxWidgets, GTK, ...

Introduction

- **Rich I/O interfaces** \Rightarrow Various sensors & devices can be employed as RoBoard's senses.
 - **A/D, SPI, I²C, COM**: accelerometer, gyro, ...
 - **GPIO**: bumper, infrared sensors, on/off switches, ...
 - **PWM**: RC servos, DC motors, ...
 - **USB**: webcam, ...
 - **Audio in/out** : speech interface

Introduction

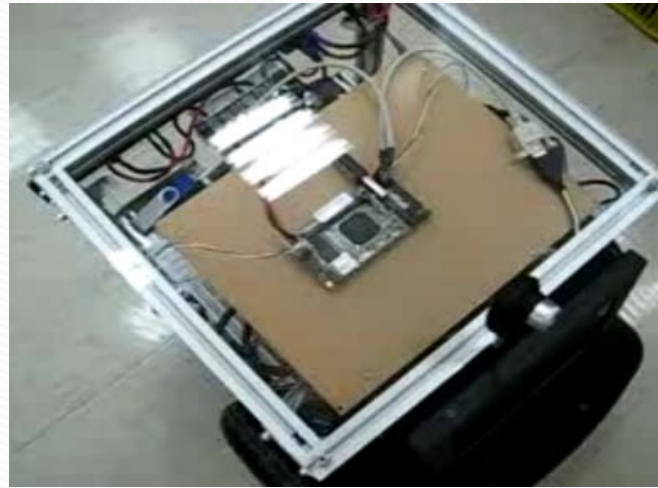
**Rich I/O (using RoBoIO) + Rich
resources on PC**



**Can develop robots more easily and
rapidly**

Experiences

- Mobile robot controlled by wireless joystick



- RoBoIO library + Allegro game library
- Take < 20 minutes to complete the control program

Experiences

- Manipulator with object tracking



- RoBoIO library + OpenCV library
- Take < 3 hours to complete the program

Experiences

- RC-servo humanoid robots (motion capture + replay, script control, MP3 sound, zipped data files)



- RoBoIO library + irrKlang library + zziplib library
- Take < 5 days to complete the program



The heart of Robotics

THANK YOU

www.roboard.com
info@roboard.com