RIO Application Kit for Windows Embedded CE 6.0

User's Manual

Digi document reference number: 90001030_A

Digi International Inc.

11001 Bren Road East

Minnetonka, MN 55343 (USA)

+1 877 912-3444 or +1 952 912-3444

http://www.digiembedded.com

Contents

Conventions used in this manual

Here is a list of the typographical conventions used in this manual:

Style	New terms and variables in commands, code, and other input.
`Style`	In examples, to show the contents of files, the output from commands. In, text the C code. Variables to be replaced with actual values are shown in italics.
Style	For menu items, dialogs, tabs, buttons, and other controls. In examples, to show the text that you enter
Menu name > option	A menu followed by one or more options; for example, File > New.

This manual also uses these frames and symbols:

This is a warning. It helps solve or avoid common mistakes or problems.

This is a hint. It contains useful information about a topic.

```
> This is a host computer session
> Bold text indicates what must be entered.
```

```
> This is a target session
> Bold text indicates what must be entered.
```

Abbreviations

BSP	Board Support Package
CPU	Central Processing Unit
FTP	File Transfer Protocol
HS SPI	High Speed SPI
I/O	Input/Output
LSB	Less Significant Byte
MSB	Most Significant Byte
OS	Operating System
PC	Personal Computer
PWM	Pulse Width Modulator
PPM	Pulse Position Modulator
QD	Quadrature Decoder
SPI	Serial Peripheral Interface
us	Microsecond

1 Introduction

1.1 Overview

The Rabbit RIO Programmable I/O Application Kit is a peripheral product designed to be incorporated into systems requiring versatile timing controls and a broader range of functionality.

The RIO Application Kit hardware board connects to the Development or JumpStart board via the Serial Peripheral Interface.

The design of the RIO Application Kit's ports allows any of the eight identical ports, each with four bits or I/O pins, to be programmed to perform any number of different functions, including a pulse-width modulator, a pulse-position modulator, quadrature decoders, pulse measurement, and I/O, including pin-pair protection for applications such a H-bridge drivers.

The RIO Application Kit is powered by the SPI port on the Development or JumpStart board.

The RIO Application Kit contains a driver for Microsoft Windows® Embedded CE 6.0 OS Design.

This User's Manual assumes that the reader is able to create, compile and download a kernel image.

1.2 Features of the RIO Application Kit

The following are the major features of the RIO Application Kit:

- 5V tolerant
- 8 independent functional ports with 4 pins each
- Any pin of each port is capable of:
    - Generating PWM outputs and variable-phase PWM outputs
    - Input capture (pulse length or frequency)
    - Decoding quadrature signals
    - Provide extended I/O pins to the microprocessor
    - Pin-pair protection for driving H bridges
- Up to 32 digital I/O lines, up to 4 general-purpose inputs
- 4 User's button
- SPI Mode communication, LSB first

2 Requirements

Minimum and recommended development system requirements:

- x86 PC with 500 MHz Pentium III or faster processor; 2 GHz Pentium 4 or equivalent recommended
- Microsoft Windows® 2000 Professional with Service Pack 4 or Windows XP Professional with Service Pack 1.
- Microsoft Windows® CE 6.0 with Service Pack 1 installed and updates
- Serial port
- Ethernet network card

2.1 BSP requirements

Display interface support recommended for development purposes, but not required.

2.2 Microsoft Windows® CE OS Configuration

2.2.1 Required components

Go to the Catalog and expand **Core OS > CEBASE**. Then include this element:

Core OS Services:

- Serial Port Support

If you are using the ConnectCore 9C/Wi-9C or ConnectCore 9P 9360 go to the Catalog and expand **Third Party > BSP > *YourPlatform* > Device Drivers > Serial Port**. Then include this element:

Serial B:

- NS9XXX SPI

If you are using the ConnectCore 9M 2443 go to the Catalog and expand **Third Party > BSP > ConnectCore 9M Wi-9M 2443 > Device Drivers > SPI**. Then include these elements:

- S3C2443 HS SPI Driver
- S3C2443 SPI Driver

2.2.2 Recommended catalog components

Other recommended networking utilities and services are:

Core OS > CEBASE > Communication Services and Networking:

- Networking General
  - Network Utilities
- Servers
  - FTP Server
  - Telnet Server

3 Installation

This software package is installed by executing downloadable setup.exe from *http://www.digi.com/support/* , from the "Select a Product" drop down box, select "Application Kits". The installer wizard will guide you through the rest of the installation.

The installer adds the following components to your system:

%ProgramFiles%\Digi\AppKits\RIO_AppKit:

- **Uninstaller**: Executable to uninstall the RIO Application Kit.
- **Release Notes** and **License Agreements.**

%_WINCEROOT%\OTHERS\Digi\AppKits\RIO_AppKit\src\driver:

- The driver's source code.

%_WINCEROOT%\OTHERS\Digi\AppKits\RIO_AppKit\src\lib:

- The library's source code.

%_WINCEROOT%\OTHERS\Digi\AppKits\RIO_AppKit\src\test_application:

- The test application source code.

%ProgramFiles% is the environment variable that points to your Program Files directory (usually C:\Program Files)

%_WINCEROOT% is the environment variable that points to your Microsoft Windows® CE root directory (usually C:\WINCE600).

4 Integration

4.1 Integration overview

Before beginning the integration process you must have the following applications installed.

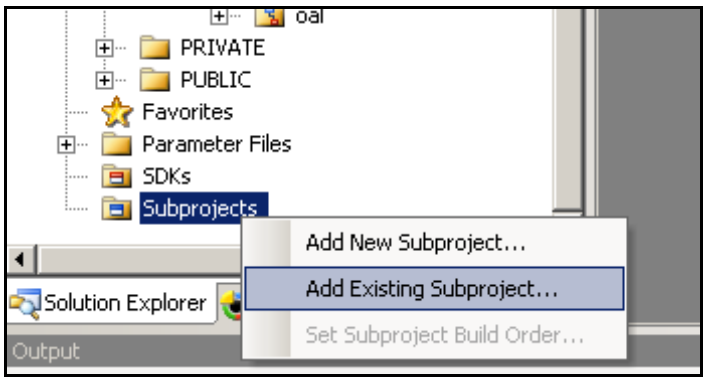
- Microsoft Windows® CE 6.0 as described in Chapter 2, and
- a BSP corresponding with the hardware that will be used.

The following sections describe the steps needed to integrate the RIO Application Kit drivers and software into your project.

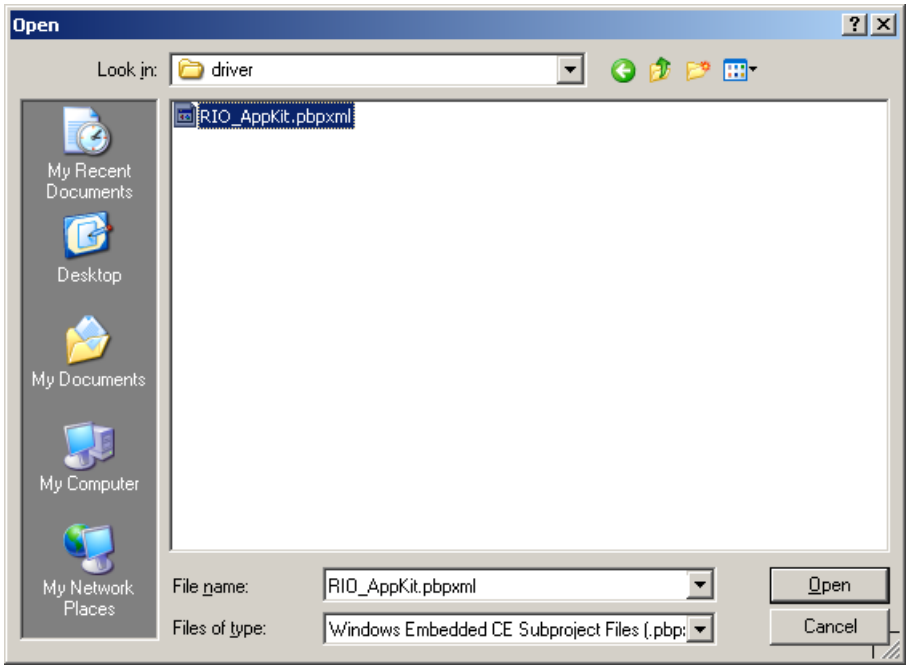
4.2 Integration process

First, you need to complete a build of your project, when that is complete you can then add the RIO Application Kit driver to your project as a new subproject.

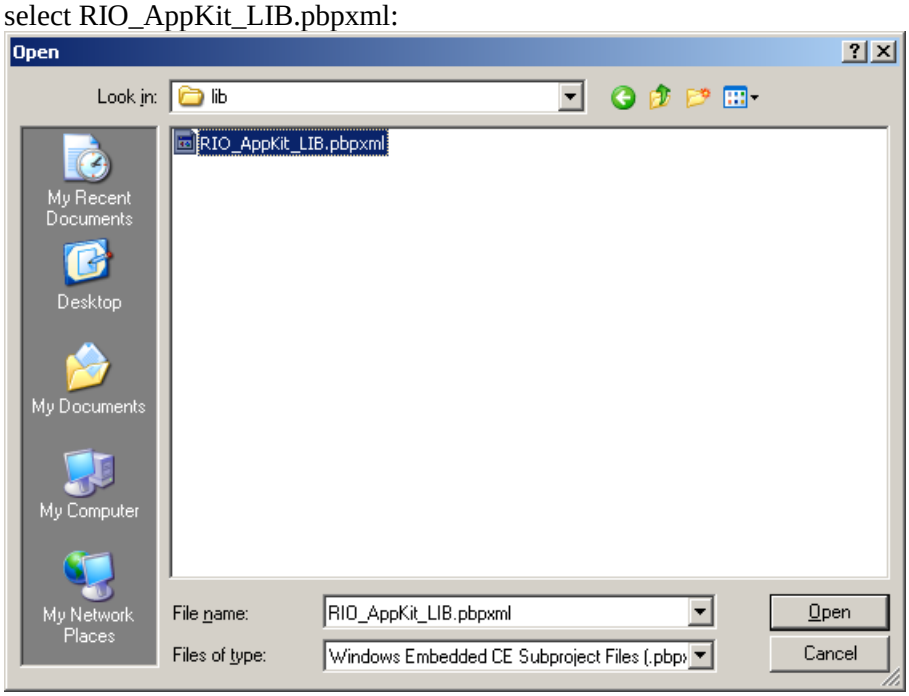
Open the Solution Explorer and right-click on Subprojects. Choose **Add Existing Subproject**:



Navigate to *%_WINCEROOT%\OTHERS\Digi\AppKits\RIO_AppKit\src\driver* directory and select RIO_AppKit.pbpxml:



Then, navigate to *%_WINCEROOT%\OTHERS\Digi\AppKits\RIO_AppKit\src\lib* directory and select RIO_AppKit_LIB.pbpxml:



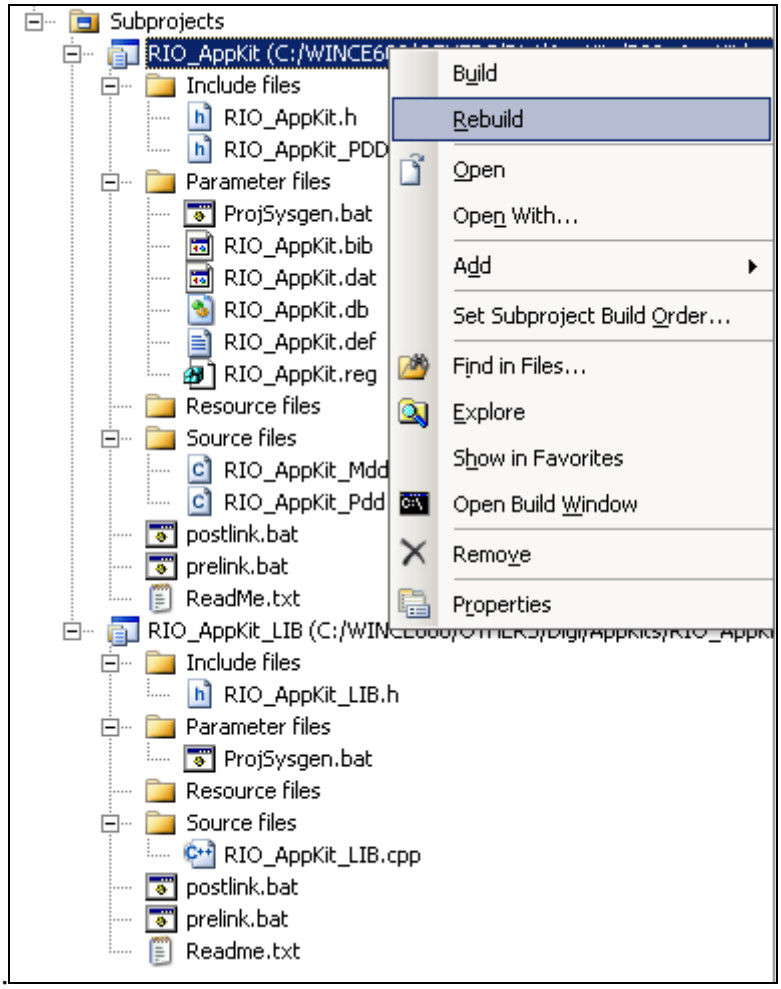
At this point you should see the parameter files and the source code.

Before you rebuild the RIO Application Kit driver you must select the registry entry that corresponds to the SPI port the RIO Application Kit is connected to.

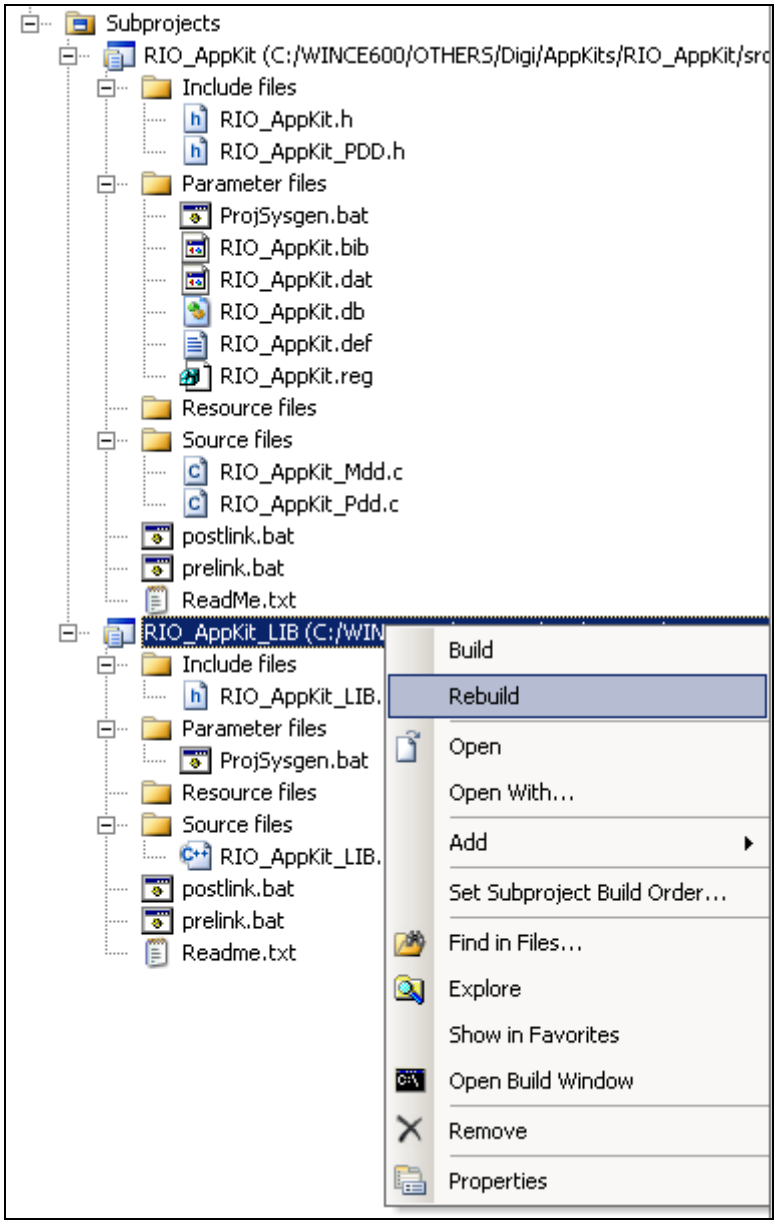
The RIO Application Kit driver needs to know which SPI port the development board is connected to. That can be configured in RIO_AppKit.reg. Simply comment out, or comment in the SPI port you would like to use.

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\RIO_AppKit]
    "Dll"="RIO_AppKit.dll"
    "Prefix"="RIO"
    "Priority256"=dword:64
    ;"RIOControllerInterface"="SPI1:" ;CCX9M2443 SPI
    "RIOControllerInterface"="SPI2:"  ;CC9C/CCWi-9C/CC9P9360 CCX9M2443 HSSPI
    "RIOPrescaleFreq"=dword:F4240     ;1000000Hz = 1MHz -> Period = 1us
```

Right click on RIO_AppKit and select **Rebuild**



Then right click on RIO_AppKit_LIB and select **Rebuild.**



And then right click on the project and select **Make Run-Time Image**.

This will build the library and driver and include them in the final nk.bin image.

4.3 Driver starts

When the build is complete you should verify that **RIO_AppKit.dll** and your registry changes are included in **reginit.ini**. Both files are located in your **Flat Release Directory.**

Once verified, download the image to the target device and start it up. You should see a console message during boot-up similar to the following:

```
[RIO_Init]: Loading RIO Application Kit… OK.
```

Depending on the debug level, more information may be shown.

5 Using the RIO Application Kit

This chapter describes the exported functions available to user applications.

For a deeper understanding of the RIO Application Kit you should read the Rabbit RIO User's Manual, which can be downloaded from its website http://www.rabbit.com/

5.1 Work with the RIO Application Kit Library

The RIO Application Kit comes with a library to simplify the management of the RIO driver via the following functions:

- RIO_OpenDevice
  - This function opens the RIO Application Kit driver
  - Needs the RIO driver name (RIOx:)
  - Returns a handle to the RIO

```
hRIO = RIO_OpenDevice(TEXT("RIO1:"));
if (hRIO == INVALID_HANDLE_VALUE)
{
    printf(("[TestRIO_LIB]Unable to open port\r\n"));
    return FALSE;
}
```

- RIO_SetOutput
  - This function sets a specific pin as output
  - Requires a handle, port number, pin number, and output pin value
  - Returns TRUE if successful, FALSE otherwise

```
if(!RIO_SetOutput(hRio, Port0, Pin0,TRUE))
{
    printf(("RIO_SetOutput, error %d\r\n"),GetLastError());
    return 0;
}
```

- RIO_SetInput
  - This function sets a specific pin as input
  - Requires a handle, port number, and pin number
  - Returns TRUE if successful, FALSE otherwise

```
if(!RIO_SetInput(hRio, Port0, Pin1))
{
    printf(("RIO_SetInput, error %d\r\n"),GetLastError());
    return 0;
}
```

- RIO_ReadOutput
  - This function reads a specific pin configured as output
  - Requires a handle, port number, and pin number
  - Returns the pin value

```
printf(("Read Output: Value= %x\r\n"),RIO_ReadOutput(Port0,Pin1));
```

- RIO_ReadInput
  - This function reads a specific pin configured as input
  - Requires a handle, port number, and pin number
  - Returns the pin value

```
printf(("Read Input: Value= %x\r\n"),RIO_ReadInput(Port0,Pin1));
```

- RIO_SetPWM
  - This function sets a PWM on a specific pin
  - Requires a handle, port number, pin number, period (in time units), and duty cycle.
  - Returns TRUE if successful, FALSE otherwise

```
if(!(RIO_SetPWM(hRio, Port1, Pin0, 1000, 50)))
{
    printf(("RIO_SetPWM, error %d\r\n"),GetLastError());
    return 0;
}
```

- RIO_SetPPM
  - This function sets a PPM on a specific pin
  - Requires a handle, port number, pin number, period (in time units), duty cycle, and phase (in degrees)
  - Returns TRUE if successful, FALSE otherwise

```
if(!(RIO_SetPPM(hRio, Port2, Pin0, 1000, 50, 90)))
{
    printf(("RIO_SetPPM, error %d\r\n"),GetLastError());
    return 0;
}
```

- RIO_SetCapture
  - This function configures a pin for input capture
  - Needs handler, port number, pin to the capture, pin to start the capture and value, pin to finish the capture and value.
  - Returns TRUE if successful, FALSE otherwise

```
if(!RIO_SetCapture(hRio, Port3, Pin0, Pin0, RISING, Pin1, FALLING))
{
    printf(("RIO_SetCapture, error %d\r\n"),GetLastError());
    return 0;
}
```

- RIO_GetCapture
    - This function gets the value of the Counter End Register, see the Rabbit RIO User's Manual for more information
    - Requires a handle and port number
    - Returns the Counter End value

```
printf(("Value= %d\r\n"),RIO_GetCapture(hRio, Port3));
```

- RIO_SetPinPairProtection
    - This function sets pin pair protection
    - Requires a handle, port number, two pins to protect, and the value of each pin.
    - Returns TRUE if successful, FALSE otherwise

```
if(!RIO_SetPinPairProtection(hRio, Port4, Pin0, Pin1, 0, 0))
{
    printf(("RIO_SetPinPariProtection, error %d\r\n"),GetLastError());
    return 0;
}
```

- RIO_GetPinPairProtection
    - This function gets the pin pair protection status of a specific port
    - Requires a handle and a port number
    - Returns TRUE if the port is protected, FALSE if not

```
if(RIO_GetPinPairProtection(hRio, Port4)
{
    printf(("Port has protection\r\n"),Port4);
}
```

- RIO_GetPinPairProtectionValue
    - This function gets the pin pair protection value on the requested pins
    - Requires a handle and the two pin numbers
    - Returns the protection value, 0xFF if it fails, or 0xFE if there is no protection

```
PP1 = RIO_GetPinPairProtectionValue(hRio, Port4, 0, 1);
if (PP1==0xFF)
    printf("  RIO_GetPinPairProtectionValue Failed!\r\n");
else if (PP1==0xFE)
    printf("  Pin1,Pin0 do not have protection\r\n");
else
    printf("  Pin1,Pin0 have protection=0x%x\r\n",PP1);
PP2 = RIO_GetPinPairProtectionValue(hRio, Port4, 2, 3);
```

```
if (PP2==0xFF)
    printf("  RIO_GetPinPairProtectionValue Failed!\r\n");
else if (PP2==0xFE)
    printf("  Pin2,Pin3 do not have protection\r\n");
else
    printf("  Pin2,Pin3 have protection=0x%x\r\n",PP1);
```

- RIO_SetQuadratureDecoder
    - This function sets up a quadrature decoder on three pins
    - Requires a handle, port number, pin for the I signal, pin for the Q signal, pin to reset the counter, and the reset value
    - Returns TRUE if successful, FALSE otherwise

```
if(!(RIO_SetQuadratureDecoder(hRio, Port5, Pin0, Pin1, Pin2,FALLING)))
{
    printf(("RIO_SetQuadratureDecoder, error %d\r\n"),GetLastError());
    return 0;
}
```

- RIO_ReadButton
    - This function reads the state of the user button.
    - Requires a handle and button number
    - Returns the current state

```
printf(("Value= %d\r\n"),RIO_ReadButton (hRio, 1));
```

- RIO_GetCounter
    - This function gets the value of the Count Value Register, see Rabbit RIO User's Manual for more information
    - Requires a handle and a port number
    - Returns the counter value

```
printf(("Value= %d\r\n"),RIO_GetCounter(hRio, Port5));
```

- RIO_ResetDevice
    - This function forces a soft-reset of the RIO chip
    - Requires a handle
    - Returns TRUE if successful, FALSE otherwise

```
if(!(RIO_ResetDevice(hRio)))
{
    printf(("RIO_ResetDevice, error %d\r\n"),GetLastError());
    return 0;
}
```

- RIO_CloseDevice
  - This function closes the RIO Application Kit driver
  - Requires a handle
  - Returns TRUE if successful, FALSE otherwise

```
if(!(RIO_CloseDevice(hRio)))
{
    printf(("RIO_CloseDevice, error %d\r\n"),GetLastError());
    return 0;
}
```

5.2 Limitations

The design of the RIO Application Kit's ports allows any of the eight identical ports, each with four bits or I/O pins, to be programmed to perform any number of different functions. However, there are some limitations.

In the RIO_AppKit.reg file you can define the Master Prescale Register of the RIO. By default this value is set to 0xF4240 (1,000,000) to obtain a frequency of 1MHz (period of 1us). If you change the Master Prescale Register you will also change the period for all of the functions.

To calculate the Master Prescale Register value, you can use the following formula:

Master Prescale Register Value = (Clock frequency/Desired Frequency) – 1

The maximum value is: 0x1312D00 (20,000,000) for a frequency of 20MHz (period of 0.05us).
The minimum value is: 0x1312D (78,125) for a frequency of 78,125Hz (period of 12.8us).

PWM:

- All PWMs on the same port must have the same period.
- The maximum period for each PWM is 65536 units of time.
- Any pin not in use for PWM can be used for general-purpose I/O

PPM:

- Only pins 0 and 2 of each port can be configured as PPM
- All PPMs on the same port must have the same period.
- The maximum period for each PWM is 65536 units of time.
- Each port can only support two PPM
- Any pin not in use for PPM can be used for general-purpose I/O

Be aware when configuring PWM or PPM, as the period gets closer to the main clock frequency (i.e. shorter in duration), the resolution of the duty cycle decreases.

Input Capture

- Only one pin of the port can be configured as input capture.
- Any pin not in use for input capture can be used for general-purpose I/O

Pin Pair Protection

- Pin Pair Protection must be set before any configuration of the pin.
- Pin pair protection must be pin0 and pin1 or pin2 and pin3 of each port.
- A hardware reset is required to remove pin pair protection.

Quadrature Decoder

- Pin to the I signal, pin to the Q signal and reset pin, must be different

When a Port is configured with a specific profile, to reconfigure this port you must do a reset to reconfigure the desired port. A reset in the RIO Application Kit resets all the ports.

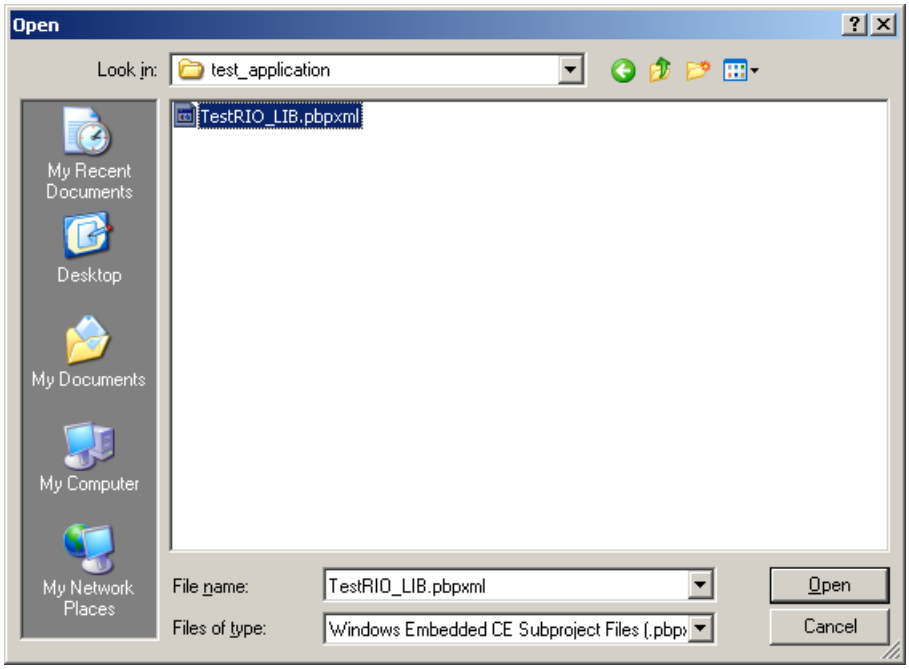
6 Test Application

The source code of the test application can be integrated into a Windows Embedded CE 6.0 project by importing it as a subproject.

Once your project has been built with the RIO Application Kit driver and library you can add the test application.

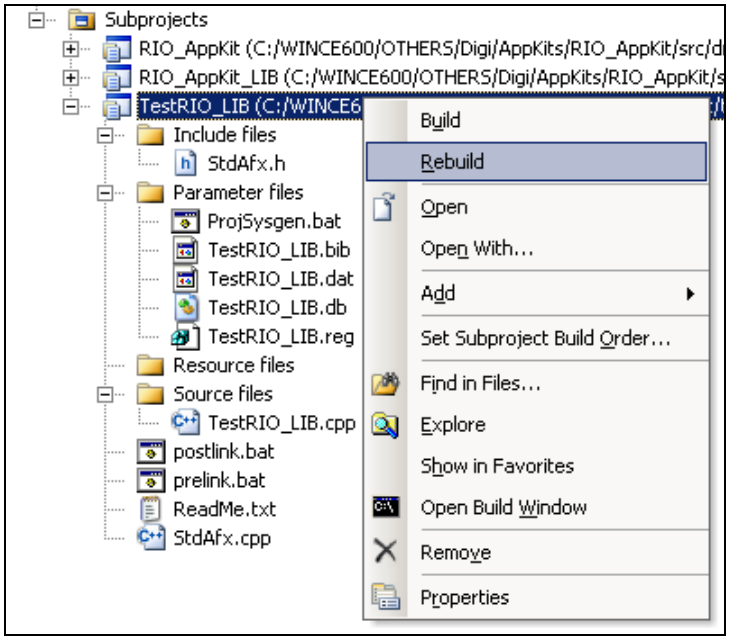
In the Solution Explorer right-click on Subprojects. Choose **Add Existing Subproject:**

Navigate to *%_WINCEROOT%\OTHERS\Digi\AppKits\RIO_AppKit\src\test_application* directory and select TestRIO_LIB.pbpxml:



Now you can see the parameter files and the source files.

Right click on TestRIO_LIB and select **Rebuild**.



Right click on the project and select **Make Run-Time Image.**

The test application build should then be rebuilt and included in the final **nk.bin** image.

TestRIO_LIB is a command line application that is up to test all the profiles of the RIO Application Kit. Given below the help of the application to manage the different profiles:

```
\> testRIO_LIB HELP
TestRIO Application (LIB)!
TesRIO_LIB Help!

Application to test the different profiles of the RIO Application Kit
Revision 1.0 Copyright (c) 2008 by Digi International Inc.

  Usage: TestRIO_LIB <options>

  Where options are:
    OUTPUT <Port> <Pin> <Value>                                    Set a pin as
                                                                   output
    INPUT <Port> <Pin>                                             Set a pin as
                                                                   input
    READOUTPUT <Port> <Pin>                                        Read output
                                                                   value
    READINPUT <Port> <Pin>                                         Read input
                                                                   value
    PWM <Port> <Pin> <Period(units time)> <Duty cycle(%)>          Set a PWM

    PPM <Port> <Pin> <Period(units time)> <Duty cycle(%)> <Phase(degree)> Set a PPM

    CAPTURE <Port> <Pin> <PinStart> <Value> <PinEnd> <Value>       Set an input
                                                                   capture
    COUNTEREND <Port>                                              Get the
                                                                   Count End
    QD <Port> <PinI> <PinQ> <PinReset> <ResetValue>                Set a
                                                                   quadrature decoder
    COUNTERVAL <Port>                                              Get the
                                                                   Count value
    PINPAIRPROTECTION <Port> <Pin> <Pin> <Value> <Value>           Set a pin
                                                                   pair protection
    GETPINPAIRPROTECTION <Port>                                    Get a pin
                                                                   pair protection
    BUTTON <number>                                                Read Button
                                                                   value
    RESET                                                          Reset the RIO
                                                                   chip
    HELP                                                           Show this
                                                                   help
  Examples:
    TestRIO_LIB OUTPUT 0 0 1
    TestRIO_LIB INPUT 0 1
    TestRIO_LIB READOUTPUT 0 0
    TestRIO_LIB READINPUT 0 1
    TestRIO_LIB PWM 1 0 1000 50
    TestRIO_LIB PPM 2 2 20000 10 180
    TestRIO_LIB CAPTURE 3 0 0 RISING 0 FALLING
    TestRIO_LIB COUNTEREND 3
    TestRIO_LIB QD 4 0 1 2 FALLING
    TestRIO_LIB COUNTERVAL 4
    TestRIO_LIB PINPAIRPROTECTION 5 0 1 0 0
    TestRIO_LIB GETPINPAIRPROTECTION 5
    TestRIO_LIB BUTTON 1
    TestRIO_LIB RESET
    TestRIO_LIB HELP

  To obtain specific information about one profile write only the first parameter, for
example: TestRIO_LIB PWM
\>
```

6.1 Examples

In this chapter you can see different examples to the different configurations of the RIO Application Kit.

6.1.1 Configure an Output

Command line to configure Port 0 Pin 0 as Output to high value.

```
\> TestRIO_LIB OUTPUT 0 0 1
TestRIO_LIB Application!
Output:
  Port number=0
  Pin number=0
  Value=1
\>
```

6.1.2 Configure an Input

Command line to configure Port 0 Pin 1 as Input.

```
\> TestRIO LIB INPUT 0 1
TestRIO_LIB Application!
Input:
  Port number=0
  Pin number=1
  Read Input=1
\>
```

6.1.3 Read an Output

Command line to read Port 0 Pin 0 as Input.

```
\> TestRIO_LIB INPUT 0 0
TestRIO_LIB Application!
Input:
  Port number=0
  Pin number=0
  Read Output=1
\>
```

6.1.4 Read an Input

Command line to read Port 0 Pin 1 as Input

```
\> TestRIO_LIB INPUT 0 0
TestRIO_LIB Application!
Input:
  Port number=0
  Pin number=1
  Read Input=1
\>
```

6.1.5 Set a PWM

Command line to set Port1 Pin2 as PWM, period of 15000 units time and duty cycle 20%.

```
\> TestRIO_LIB PWM 1 2 15000 20
TestRIO_LIB Application!
Pulse-Width Modulation:
  Port number=1
  Pin number=2
  Period in units time=15000
  Duty cycle (%)=20
\>
```

6.1.6 Set a PPM

Command line to set Port2 Pin2 as PPM, period of 1000 units time, duty cycle 50% and phase 90º.

```
\> TestRIO_LIB PPM 2 2 1000 50 90
TestRIO_LIB Application!
Pulse-Position Modulation:
  Port number=2
  Pin number=2
  Period in units time=1000
  Duty cycle (%)=50
  Phase (degree)=90
\>
```

6.1.7 Configure an Input Capture/Get Capture

Command line to configure Port3 Pin0 as input capture, the start signal is Pin 1 in rising edge, and the end signal is pin 2 in falling edge. Then read the capture value. We have connected the PWM of Section 6.1.5 to the Port3 Pin0

```
\> TestRIO_LIB CAPTURE 3 0 1 RISING 2 FALLING
TestRIO_LIB Application!
Input Capture:
  Port number=3
  Pin input number=0
  Pin start number=1
  Start value=RISING
  Pin end number =2
  End value=FALLING
\> TestRIO_LIB COUNTEREND 3
TestRIO_LIB Application!
Counter End:
  Port number=3
  Value= 2999
\>
```

6.1.8 Configure a Quadrature Decoder/Get Counter

Command line to configure Port5 Pin0 as I input signal, Pin1 as Q input signal, and Pin2 as reset signal when falling. We have connected a the PPM 50% phase 0 to pin0, PPM 50% phase 270º to pin1, and pin 2 is connected to ground, in this case the counter will be incremented.

```
\> testRIO_LIB QD 5 0 1 2 FALLING
TestRIO_LIB Application!
Quadrature Decoder:
  Port number=5
  Pin I number=0
  Pin Q number=1
```

```
  Pin reset number=2
  Reset value =FALLING
\> TestRIO_LIB COUNTERVAL 5
TestRIO_LIB Application!
Counter Value:
  Port number=5
  Value= 368
\>
```

6.1.9 Set a Pin Pair Protection

Command line to set a Pin Pair Protection to Port5 Pins 0 (value0) and 1 (value0).

```
\> testRIO_LIB PINPAIRPROTECTION 5 0 1 0 0
TestRIO_LIB Application!
Pin Pair Protection:
  Port number=5
  Pin 1 number=0
  Pin 2 number=1
  Pin 1 value=0
  Pin 2 value=0
\>
```

6.1.10 Get a Pin Pair Protection

Command line to get the current Pin Pair Protection of Port5.

```
\> testRIO_LIB GETPINPAIRPROTECTION 5
TestRIO_LIB Application!
Pin Pair Protection:
  Port number=5 has protection
  Pin1,Pin0 have protection=0x0
  Pin3,Pin2 do not have protection
\>
```

6.1.11 Read a Button state

Command line to read User Button 1.

```
\> testRIO_LIB BUTTON 1
TestRIO_LIB Application!
Button read:
  Button number=1
  Read Button= 0
\>
```

6.1.12 Force a Reset

Command line to make a reset to the RIO chip.

```
\> testRIO_LIB RESET
TestRIO_LIB Application!
Reset Done!
\>
```

7 Uninstalling

The RIO Application Kit can be uninstalled separately using Windows Control Panel.

Add or Remove Programs:

- **RIO Application Kit for Windows Embedded CE 6.0**