UPort 1100 Series User Manual

Version 10.0, June 2024

www.moxa.com/products

UPort 1100 Series User Manual

The software described in this manual is furnished under a license agreement and may be used only in accordance with the terms of that agreement.

Copyright Notice

Trademarks

Disclaimer

- Information in this document is subject to change without notice and does not represent a commitment on the part of Moxa.
- Moxa provides this document as is, without warranty of any kind, either expressed or implied, including, but not limited to, its particular purpose. Moxa reserves the right to make improvements and/or changes to this manual, or to the products and/or the programs described in this manual, at any time.
- Information provided in this manual is intended to be accurate and reliable. However, Moxa assumes no responsibility for its use, or for any infringements on the rights of third parties that may result from its use.
- This product might include unintentional technical or typographical errors. Changes are periodically made to the information herein to correct such errors, and these changes are incorporated into new editions of the publication.

Technical Support Contact Information

www.moxa.com/support

Table of Contents

1. Introduction

The UPort 1100 Series of USB-to-serial converters provides an easy way to add COM ports to your computer. The UPort 1110 provides one RS-232 port; the UPort 1130 and UPort 1130I each provide one RS-422/RS-485 port; the UPort 1150 and UPort 1150I each provide one RS-232/422/485 port. As a plug-and-play USB device, the converters are perfect for mobile, instrumentation, and point-of-sale applications.

The UPort 1100 Series includes the following models:

- **UPort 1110:** 1-port RS-232 USB-to-serial converter.
- **UPort 1130:** 1-port RS-422/485 USB-to-serial converter.
- **UPort 1130I:** 1-port RS-422/485 USB-to-serial converter with isolation protection.
- **UPort 1150:** 1-port RS-232/422/485 USB-to-serial converter.
- **UPort 1150I:** 1-port RS-232/422/485 USB-to-serial converter with isolation protection.

Overview

The UPort 1100 Series is part of Moxa's UPort line of USB-to-serial converters. The UPort line provides a range of easy-to-use solutions for adding COM ports through a PC's USB port. Simply install the drivers, connect the UPort to your computer, plug in your serial devices, and you're ready to go. Programming is not required, and you do not need to worry about IRQs, configuring a board, power requirements, or connection schemes.

The UPort 1100 series provides single COM port expansion. The UPort 1110 adds one RS-232 port to your computer, which is the same type of COM port that is built into most PC motherboards. The UPort 1130/1130I adds one port that is configurable for RS-422, 2-wire RS-485, and 4-wire RS-485 modes. The UPort 1150/1150I adds one RS-232, RS-422, 2 wire RS-485, or 4 wire RS-485 port to your computer. All of these models can operate at speeds of up to 12 Mbps, which are much faster than the current maximum serial transfer rate. The converter is bus-powered, so no external power supply is required.

ARDC (Automatic Recovery Data Communication)

ARDC (Automatic Recovery Data Communication) makes it easier to recover data transmission, even if the cable is accidentally unplugged. This means that you do not need to worry about reconfiguring complex settings to ensure smooth data transmission. If the USB cable is accidentally unplugged, simply plug it back into the same port, and the converter will automatically reconnect with the host and continue to transfer data. This feature not only eliminates the need for reconfiguration, but it also reduces the probability of data loss.

Package Checklist

The following items are included in your UPort 1110/1130/1130I/1150 package:

- UPort 1100 Series USB-to-serial converter
- Quick installation guide (printed)
- Warranty card

The UPort 1130/1130I/1150 also come with the following item:

- 1 serial adapter: Mini DB9F-to-TB

The following items are included in your UPort 1150I package:

- 1 UPort 1150I USB-to-serial converter
- 1 serial adapter: Mini DB9F-to-TB
- USB cable: CBL-USBA/B-100
- Velcro lockdown strap for the USB cable
- Quick installation guide (printed)
- Warranty card

NOTE

Please notify your sales representative if any of the above items are missing or damaged.

Product Features

The UPort converter has the following features:

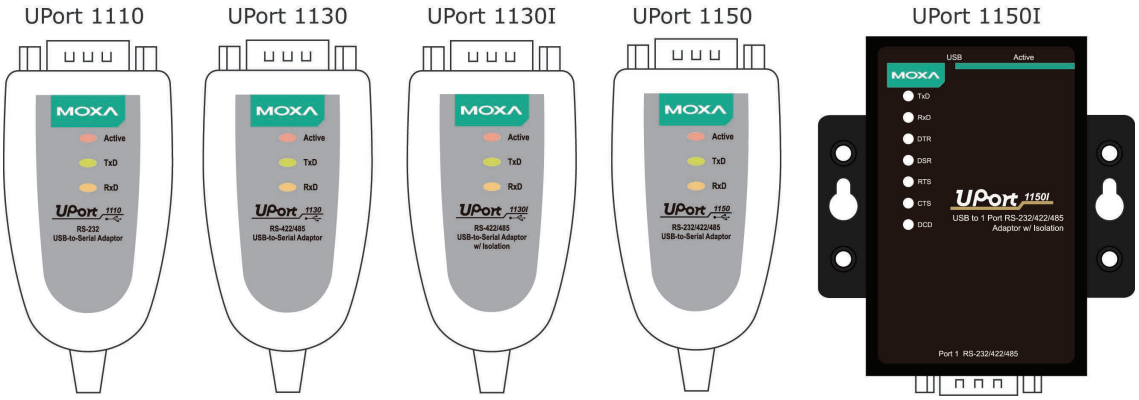
- Full-speed USB operation at up to 12 Mbps
- Additional I/O or IRQ not required
- Serial transmission speed up to 921.6 kbps
- 64-byte FIFO and built-in hardware and software flow control
- Support for RS-422, 2-wire RS-485, and 4-wire RS-485 (UPort 1130/1130I/1150/1150I)
- Terminal block adapter (UPort 1130/1130I/1150/1150I)
- 2 kV Optical isolation protection (UPort 1130I/1150I only)

Product Specifications

NOTE

The latest specifications for Moxa's products can be found at https://moxa.com.

LED Indicators

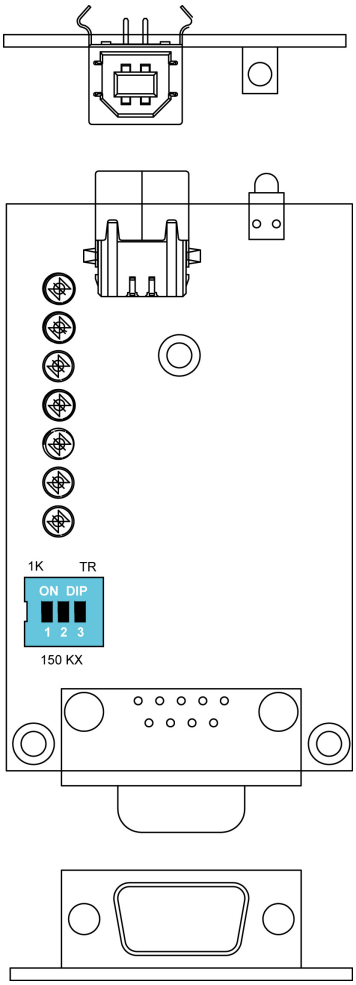


UPort 1110/1130/1130I/1150 LED description:

Name	Color	Description
Active	Red	This LED indicates normal operation. Steady: The UPort is operational. Off: A fault condition exists; there may be a problem with the converter, the driver installation, or PC configuration.
TxD	Green	Blinking: Serial device is transmitting data.
RxD	Yellow	Blinking: Serial device is receiving data.

UPort 1150I LED description:

Name	Color	Description
Active	Red	Steady: The UPort is operational. Off: A fault condition exists; there may be a problem with the converter, the driver installation, or PC configuration.
TxD	Green	Blinking: Serial device is transmitting data.
RxD	Yellow	Blinking: Serial device is receiving data.
DTR, DSR, RTS, CTS, DCD	Red	Steady: Require these signals to operate. Off: Disable these signals to operate.



Adjustable Pull High/Low, Terminator Resistors for the RS-485 Port

In some critical environments, you may need to add termination resistors to prevent the reflection of serial signals. When using termination resistors, it is important to set the pull high/low resistors correctly so that the electrical signal is not corrupted. The UPort uses DIP switches to set the pull high/low resistor values for each serial port.

To set the pull high/low resistors to 150 kΩ, make sure both assigned DIP switches are in the OFF position.

To set the pull high/low resistors to 1 kΩ, make sure both assigned DIP switches are in the ON position. This is the default setting.

SW	1	2	3
	Pull High	**Pull Low**	**Terminator**
ON	1 kΩ	1 kΩ	120 Ω
OFF	150 kΩ	150 kΩ	Disable

2. Installation and Configuration

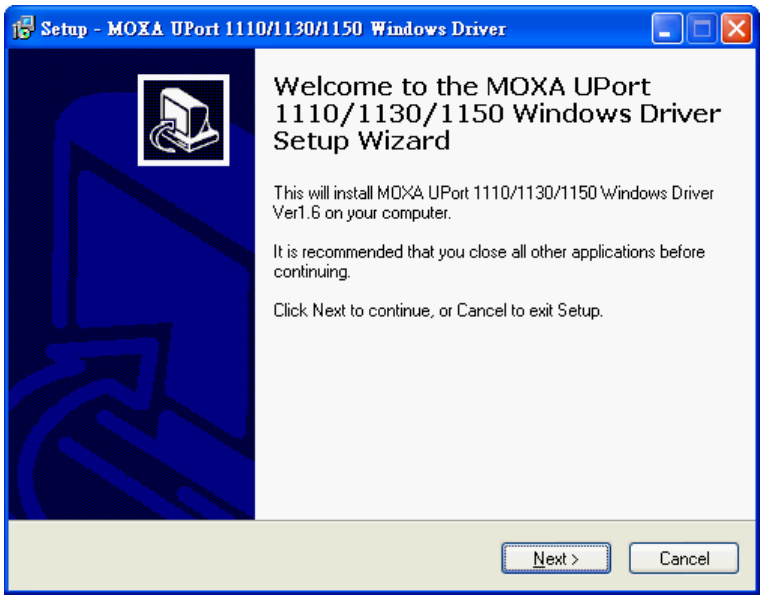
In this chapter, the product and driver installation and configuration procedures are explained.

Initial Driver Installation

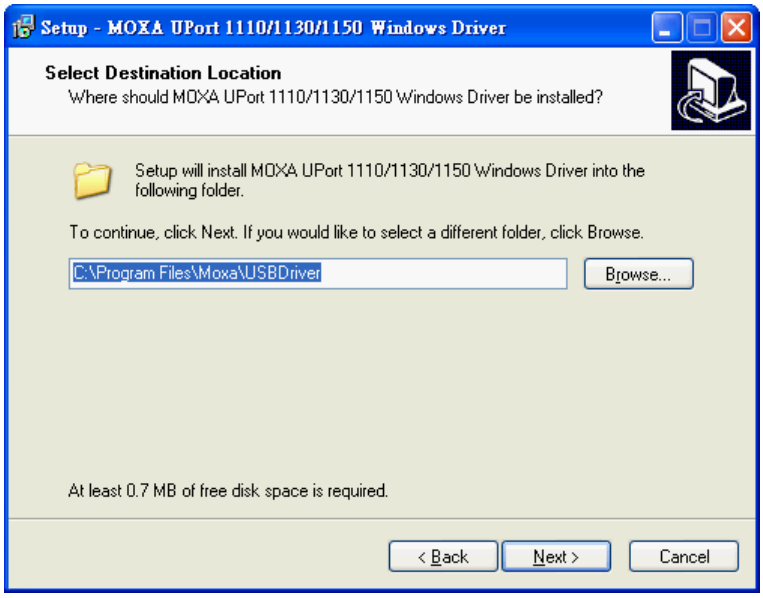
You need to download the drivers at www.moxa.com. The installation procedure is the same for all Windows operating systems.

After downloading the driver from Moxa website, extract files from the zip file and double-click the **Setup** or **Install** file.

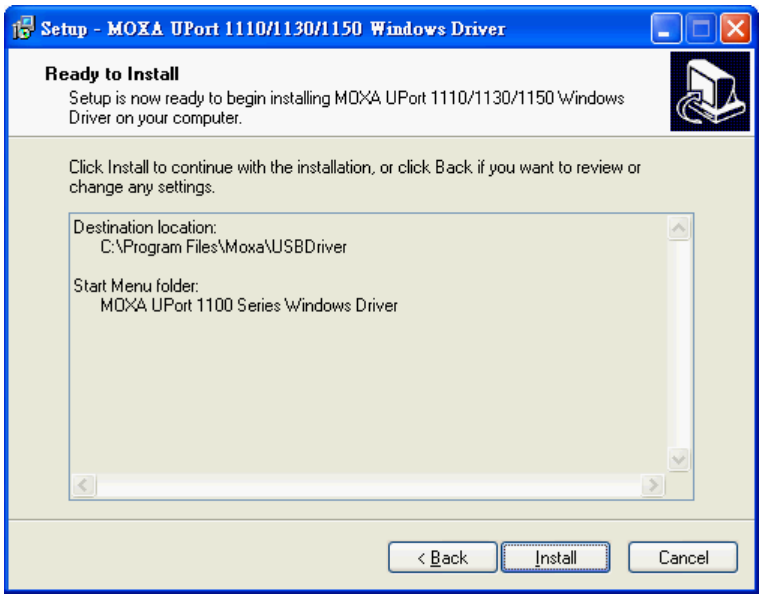
1. The Setup Wizard will open. Click **Next** to begin installing the driver.



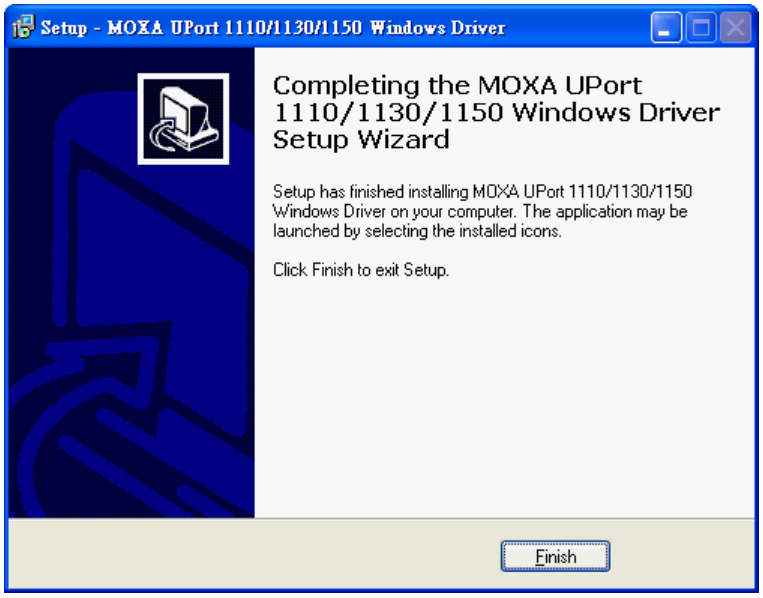
2. Click **Next** to install the driver in the indicated folder or use the drop-down folder list to locate a different folder.



3. Click **Install** to proceed with the installation.



4. Click **Finish** to complete the installation of the driver.



Hardware Installation

After installing the driver, plug the converter into the USB port on your computer. Windows will automatically detect the converter and begin installing the driver. When Windows finishes installing the driver for the converter, it will detect a new COM port and will then install another driver for the new COM port.

ATTENTION

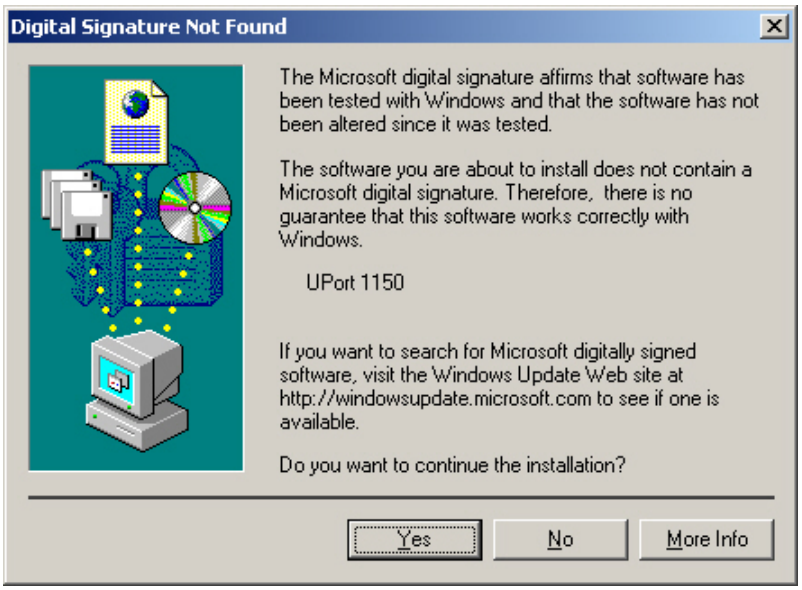
For the best results, we recommend you install the driver before plugging the converter into the USB port. Please refer to the previous section on Initial Driver Installation for instructions.

Windows 98/ME and Windows 2000

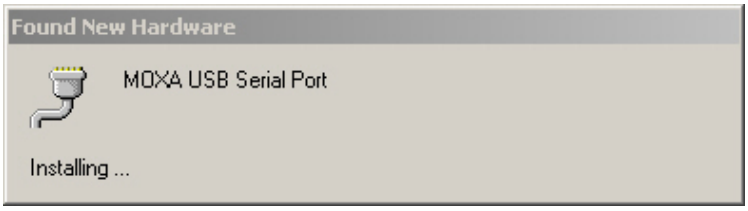
The following instructions are for Windows 98 and Windows 2000 systems:

1. After plugging the converter into a USB port, the **Found New Hardware** window should open automatically. The USB icon indicates that the USB port is being installed. No action is required.

2. Windows 98 users may skip to the next step. On Windows 2000 systems, a window will pop up cautioning you that this software has not passed Windows logo testing. This is a standard warning, and Moxa has thoroughly tested the driver for safe Windows operation. Please click Yes to proceed.



3. Windows will automatically detect and install the new serial port. No further action is required.



Windows XP, Windows 2003/Vista, and Windows 2008 and 2012 (32-bit and 64-bit)

Case 1: The UPort Driver is Certified

If the UPort driver that you installed has been certified by Microsoft, the UPort and the UPort's serial ports will be installed automatically when you plug the UPort into your computer's USB port.

Case 2: The UPort Driver is NOT Certified

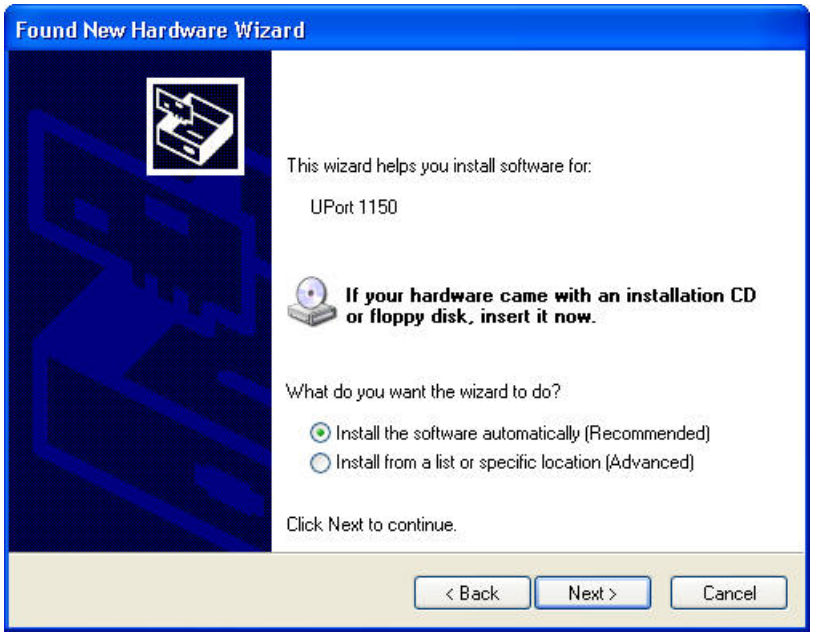
If the UPort driver that you installed has not been certified by Microsoft, then plugging the driver into your computer's USB port will activate the UPort installation program. The first part of the installation procedure installs the software for the UPort itself; the second part of the procedure installs the serial ports.

The following instructions are for Windows XP, Windows 2003/Vista, Windows 2008, and Windows 2012 systems.

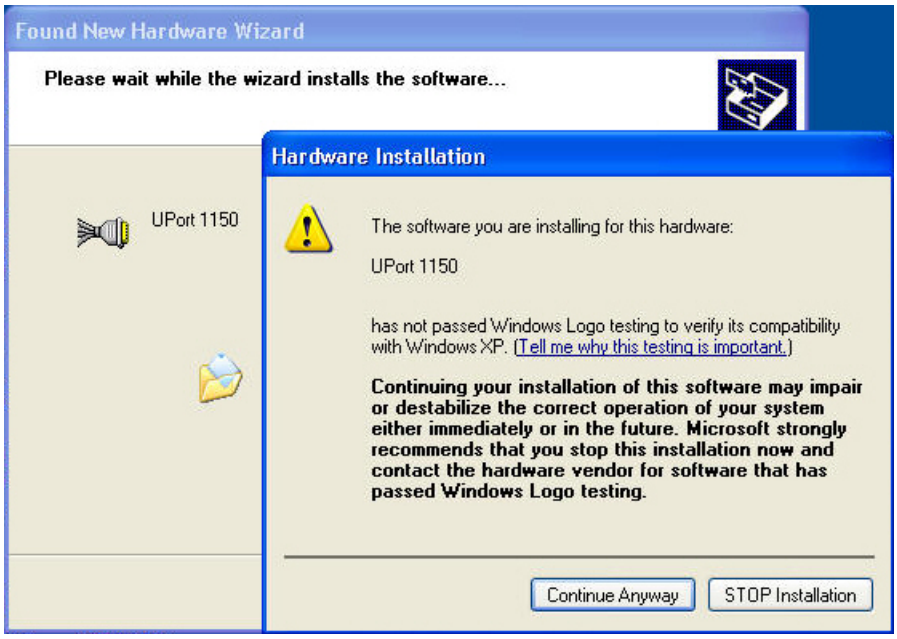
1. After plugging the converter into a USB port, Windows will automatically detect the new device. The **Found New Hardware** speech balloon will appear in the bottom right corner of the Windows desktop. No action is required yet.

2. After a moment, the Found New Hardware Wizard will open. If you see the following screen, select **No, not this time**, then click **Next**.

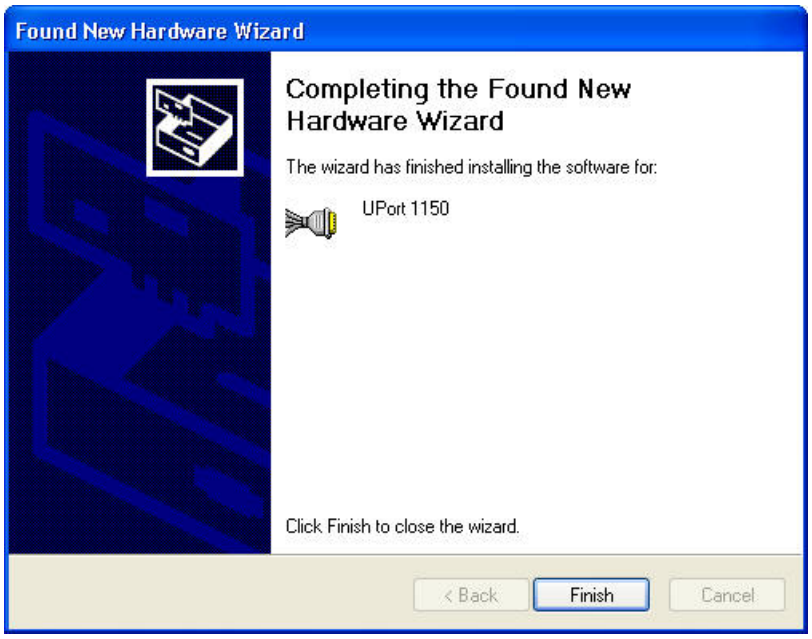
3. On the next window that appears, select **Install the software automatically (Recommended)**; then click **Next**.



4. The installation wizard will search for the correct drivers. After a moment, a window will pop up, cautioning you that this software has not passed Windows logo testing. This is a standard warning, and Moxa has thoroughly tested the driver for safe Windows operation. Please click **Continue Anyway** to proceed.



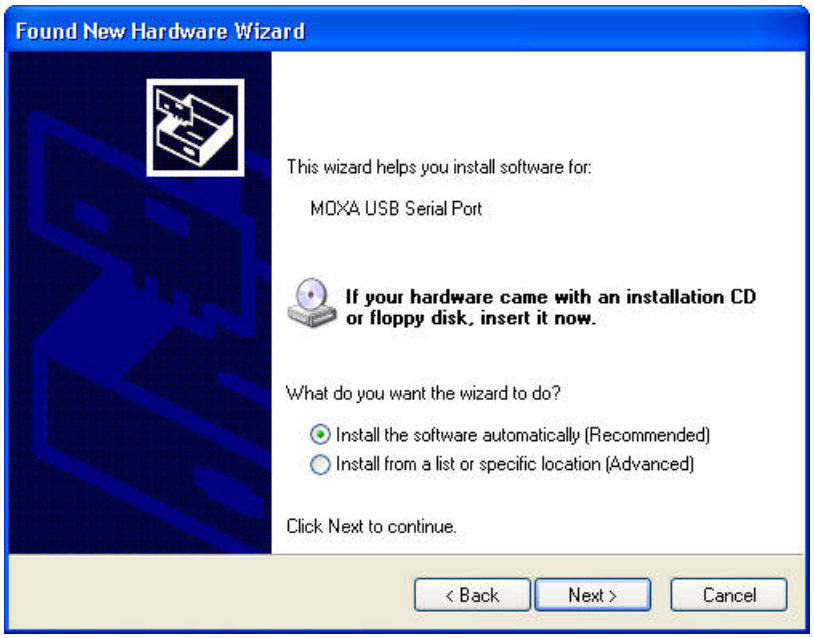
5. Windows will take a few moments installing the UPort driver.
6. The next window indicates Windows has completed the installation. Click **Finish** to continue with installation procedure.



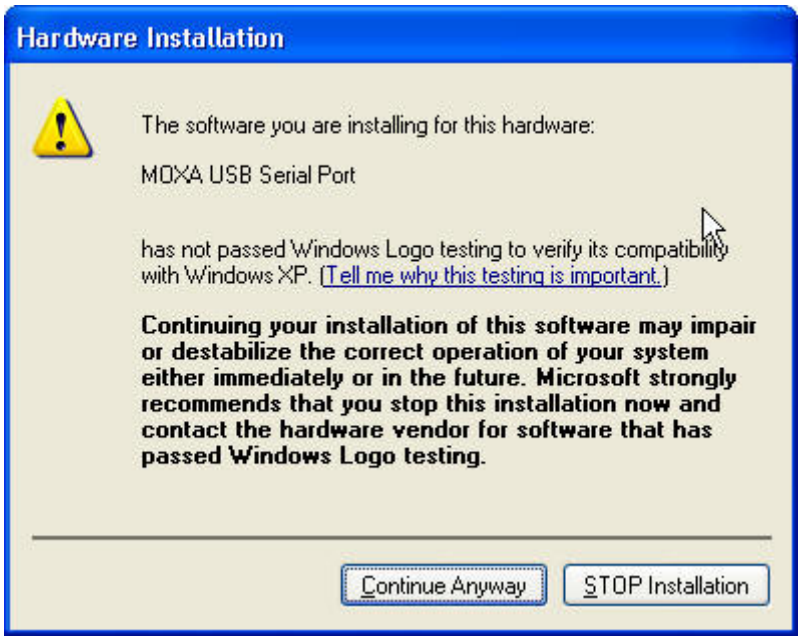
7. After Windows has completed installing the converter, it will automatically detect the new COM port. The **Found New Hardware** speech balloon will appear in the bottom right corner of the Windows desktop.

8. The Found New Hardware Wizard will open. If you see the following screen, select **No, not this time**, then click **Next**.

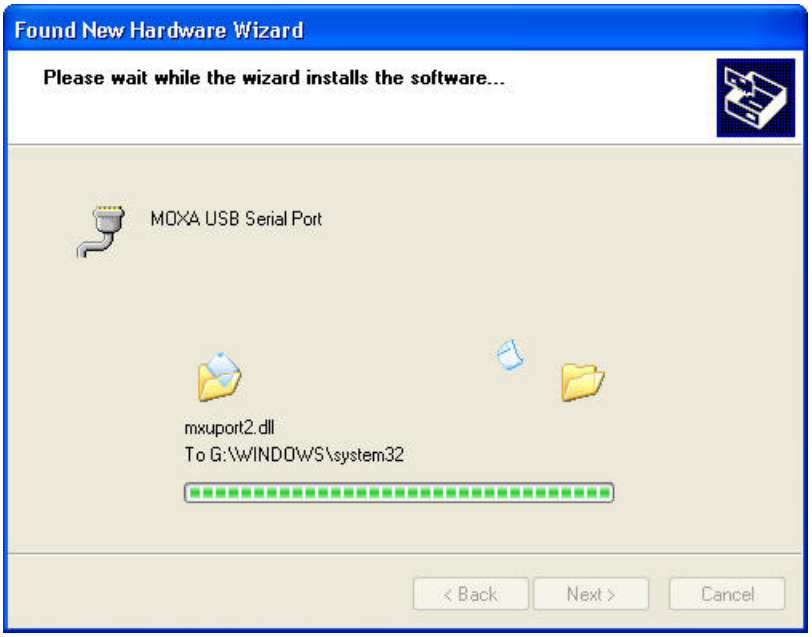
9. After a moment, the Found New Hardware Wizard will open. Select **Install the software automatically (Recommended)**, then click **Next**.



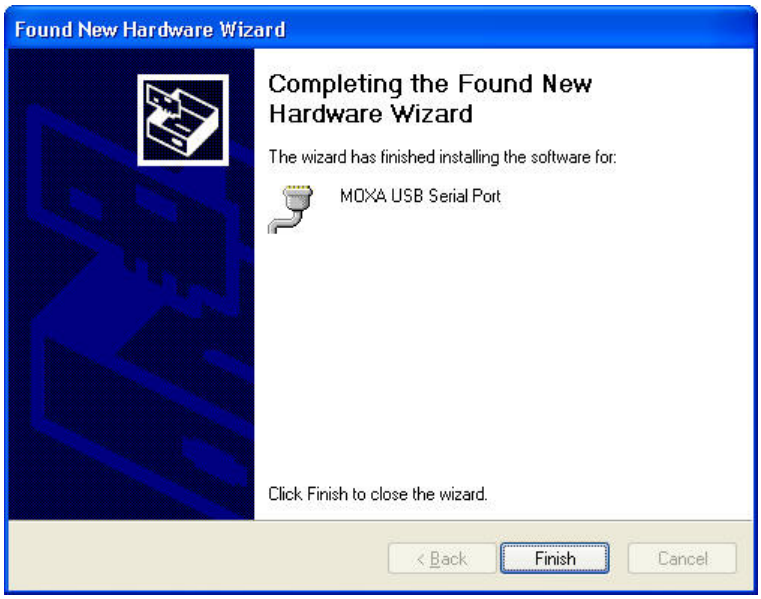
10. The installation wizard will search for the correct drivers. After a moment, a window will pop up, cautioning you that this software has not passed Windows logo testing. This is a standard warning, and Moxa has thoroughly tested the driver for safe Windows operation. Please click **Continue Anyway** to proceed.



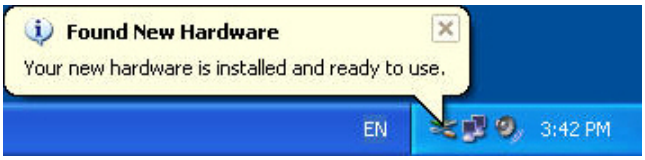
11. Windows will take a few moments to install the driver.



12. The next window indicates Windows has completed the installation. Click **Finish** to proceed with installation procedure.

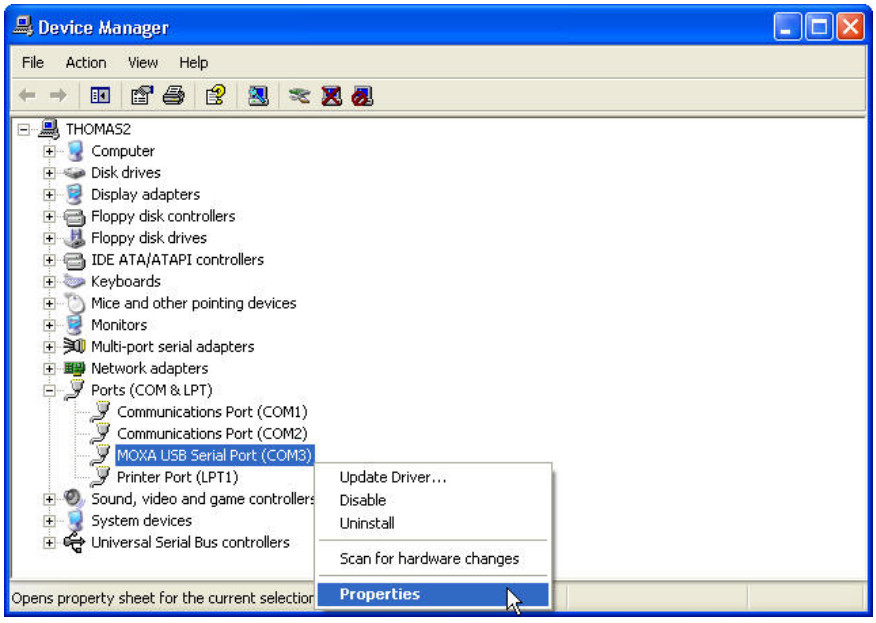


13. The **Found New Hardware** speech balloon will reappear, indicating that the installation was successful.



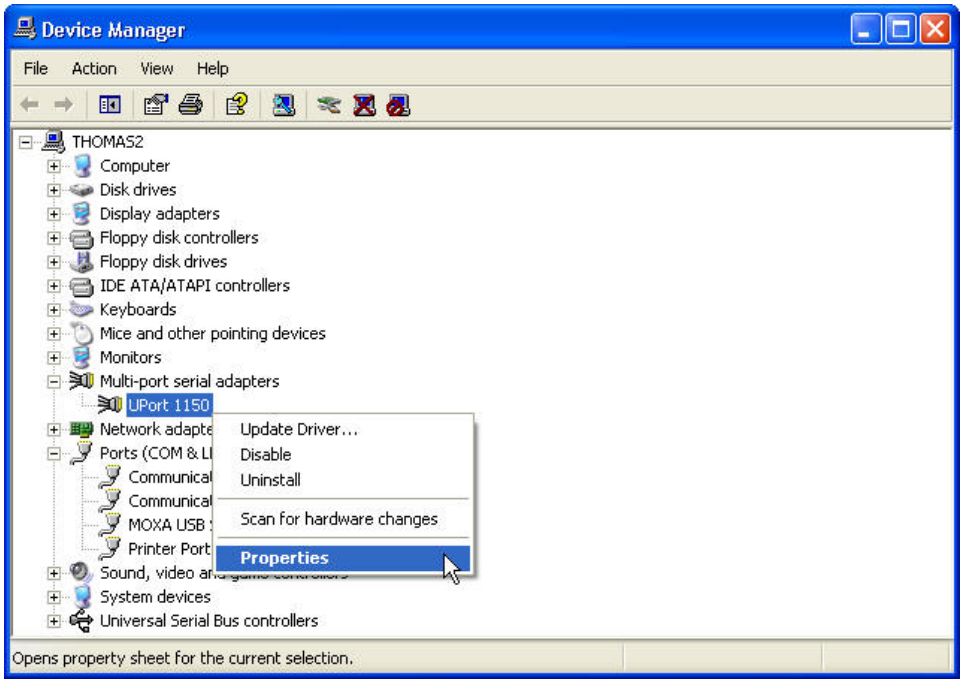
Configuring the COM Port

After the driver and hardware have been successfully installed, the new COM port will have a COM number and can be accessed and controlled just like your PC's built-in COM ports. If you need to change the baudrate, parity, or other COM port settings, you may use your serial communication application to make those changes. You may also go to Device Manager and right-click the **MOXA USB Serial Port**, which will be listed under **Ports** along with your PC's built-in COM ports. In the context menu that pops up, you may select **Properties** in order to modify the COM port settings.

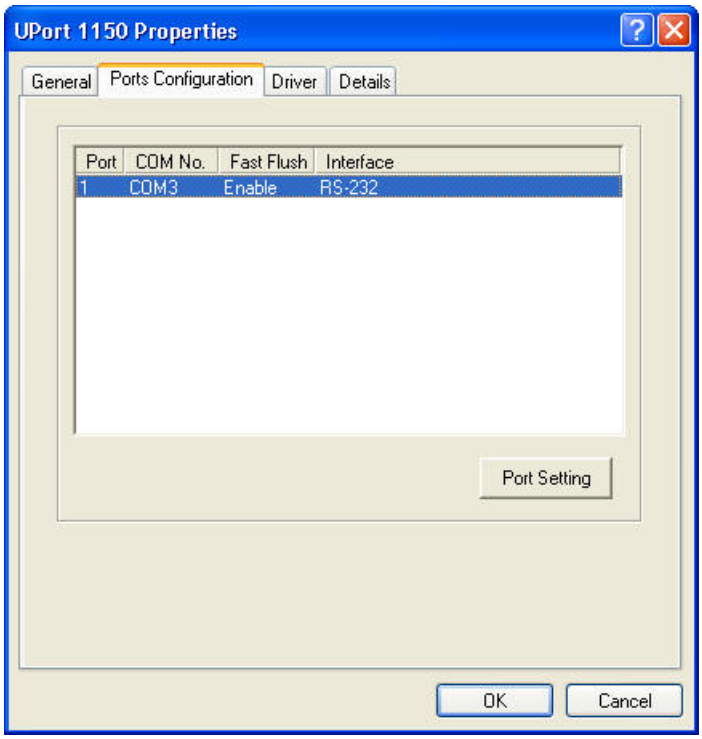


Configuring the Converter

If you need to change the COM number the converter assigns to the COM port, or adjust other advanced settings, you may go to Device Manager and right-click the UPort converter, which will be listed under **Multi-port serial adapters**. In the context menu that pops up, you may select **Properties** in order to modify the COM port settings.



In the **Port Configuration** tab, you will see the new COM port listed. Click **Port Setting** to change the COM number and other parameters.



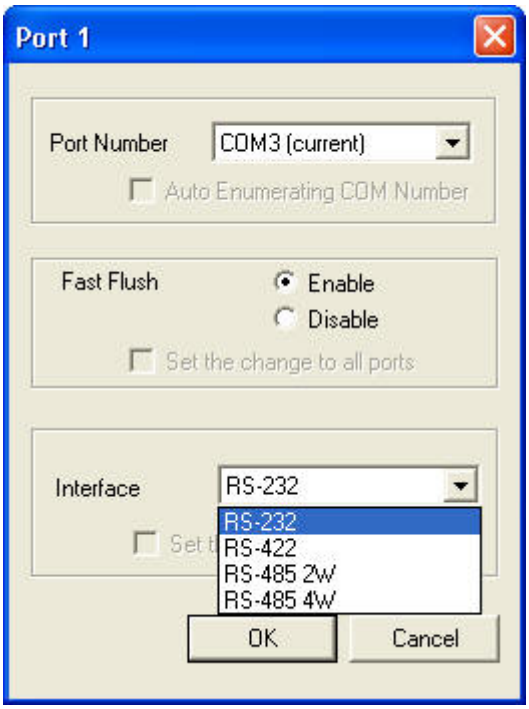
ATTENTION

Before modifying these settings, make sure that you have closed any applications that may be accessing the COM port, such as HyperTerminal.

You may change the Port Number and enable or disable the Fast Flush function. For the UPort 1130/1130I, you may also select between RS-422, 2-wire RS-485 and 4-wire RS-485 modes.

NOTE

Auto Enumerating COM Number and **Set the change to all ports** are not available for this model of the UPort.



* The UPort 1100 only supports RS-232. For this reason, when using the UPort 1100, the Interface drop-down box will be inactive.

*The UPort 1150/1150I support RS-232, RS-422, 2-wire RS-485, and 4-wire RS-485. The default setting is RS-232.

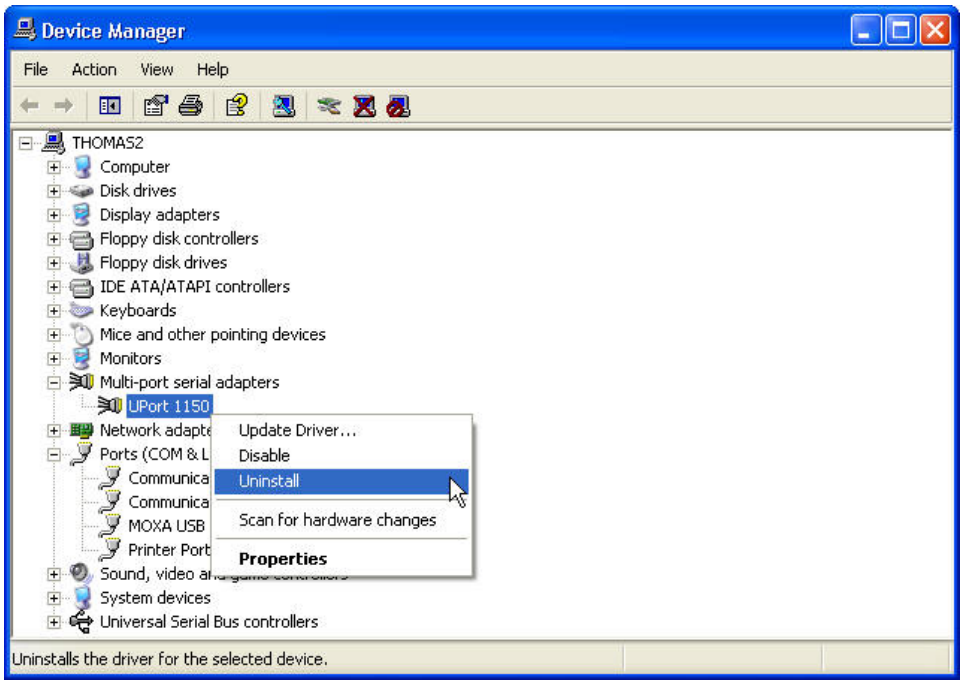
The Fast Flush function is specifically designed to handle Win32 PurgeComm() function calls and is enabled by default. When Fast Flush is enabled, the driver will automatically clear the local buffer when it receives a PurgeComm() command. When Fast Flush is disabled, the driver will repeatedly query the converter until it verifies that there is no more data in the buffer. Disabling this function can cause significantly lower throughput for applications that use PurgeComm() extensively.

Removing the Converter

The converter is a plug-and play-device. No special procedures are required to remove the converter; you may simply pull the converter out of the USB port. You should verify that no data is being transmitted before removing the converter.

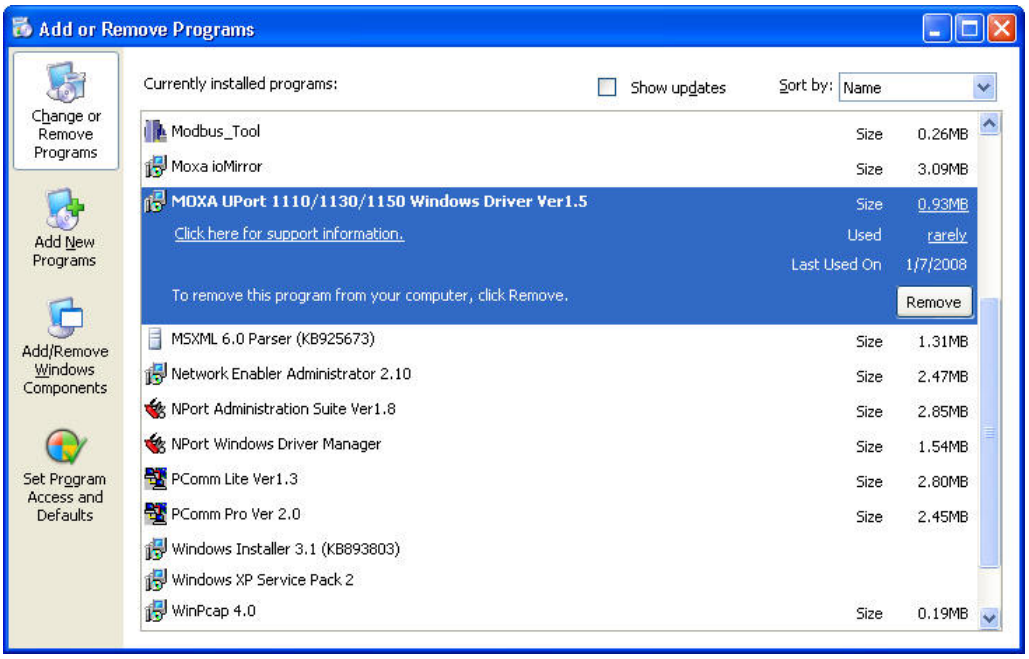
Removing the converter does not remove the drivers. The drivers remain in place so that the converter can be automatically detected and installed if it is plugged back into the USB port.

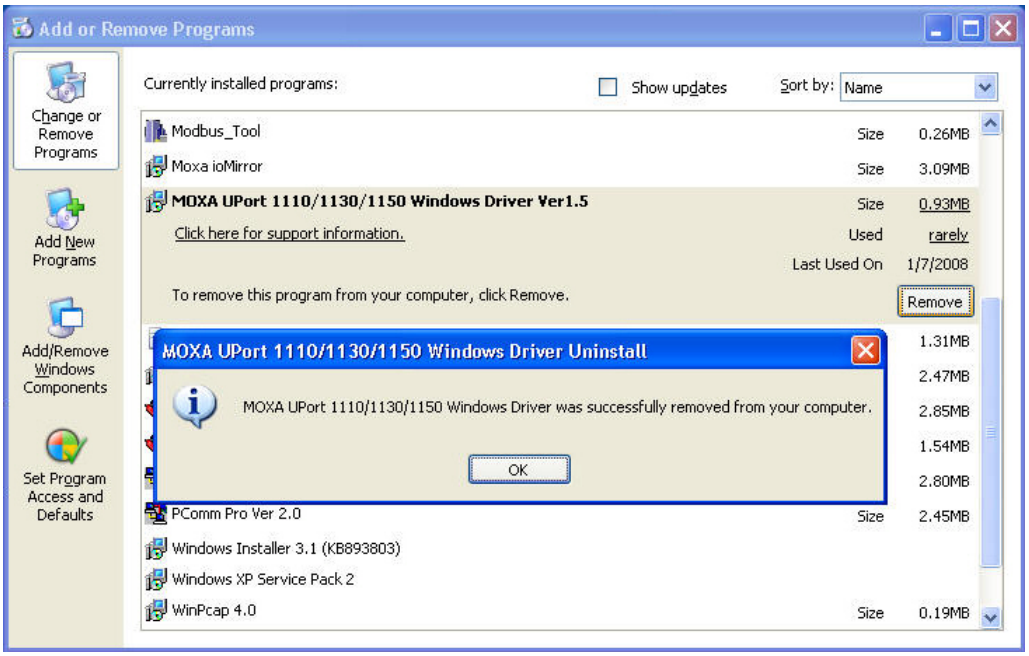
The converter can also be uninstalled through the Device Manager, the same as other Windows devices. Right-click the converter, which will be found under **Multi-port serial adapters**; then, select **Uninstall** from the context menu.



Uninstalling the Driver

The UPort driver may be removed through Add/Remove Programs in the Windows Control Panel. Click **Remove** next to **MOXA UPort 1110/1130/1150 Windows Driver Verx.x**.





Advanced UPort 1100 Driver Functions

The utilities of Moxa's UPort 1100 Series give users a convenient tool for configuring and maintaining the UPort Series products. In this section, we introduce the "Fixed-base COM Mode" function that enables users to set COM names on the host PC.

NOTE

Fixed-base COM is for Windows only.

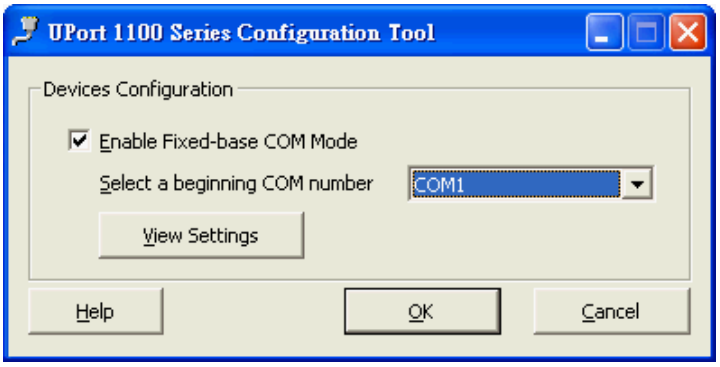
Enable Fixed-base COM Mode

Moxa's UPort 1100 Series provides the unique Fixed-base COM function that allows users to set a specific beginning COM port number. Regardless of which UPort is plugged into the host, the COM port numbers for the UPort's serial ports will be numbered sequentially—starting with the initial COM port number.

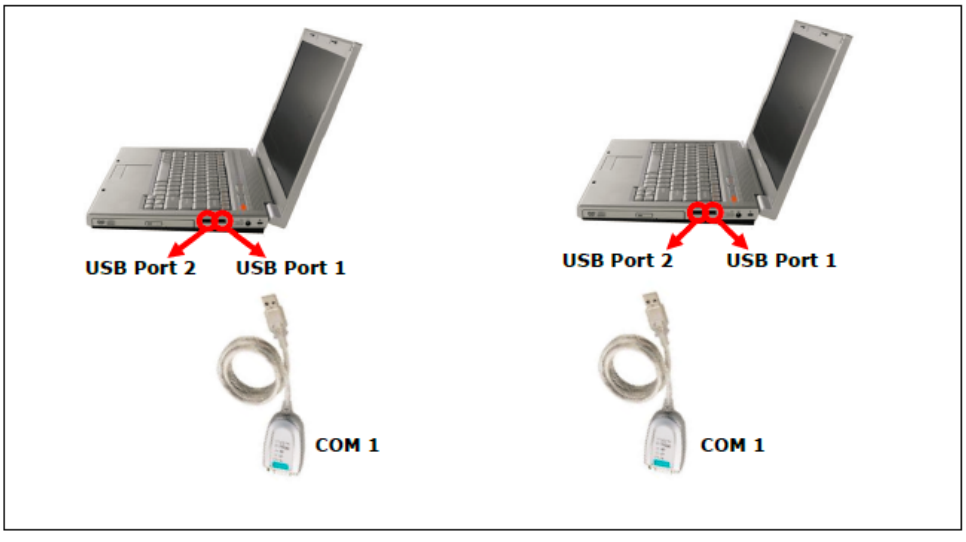
To use "Fixed-base COM Mode," be sure to check the "Enable Fixed-base COM Mode" check box.

NOTE

The default setting for "Fixed-base COM Mode" is set at disable.



The COM numbers are assigned sequentially and are not tied to a specific UPort device or USB port number. For example, let's assume that you've set COM1 as the first COM number that will be assigned. If a UPort 1150 is plugged into your computer's USB port 1, your computer assigns COM1 to the UPort's serial port. When the UPort 1150 is plugged into USB port 2, the computer still assigns COM1 to the UPort's serial port.



When “Fixed-base COM Mode” is enabled for the first time, all the COM port numbers and serial port parameters will reset to their default values. You can then set the COM numbers and configuration parameters to the values needed for your application(s).

NOTE

If you later disable “Fixed-base COM Mode,” all the parameters will be restored to their original settings.

Install Linux Driver

Driver Installation

1. Driver files
2. Device naming convention
3. Module driver configuration
4. Static driver configuration
5. Verify driver installation
6. Uninstall

1. Driver files

The driver file may be obtained from website, under the product page.

The first step is to copy driver file "driv_linux_uport1p_[VERSION]_[BUILD].tgz" into a user directory. e.g. /moxa. Please execute commands as below.

```
# cd /moxa
# tar xvfz driv_linux_uport1p_[VERSION]_[BUILD].tgz
```

2. Device naming convention

You may find all the driver files in /<driver directory>/mxu11x0/.

The following installation procedure depends on the model you'd like to run the driver.

```
Dialing and callout port
------------------------
This driver remains traditional serial device properties. Because the
limitation of the usb build-in modules that are usbcore and usbserial,
There is only one special file name for each serial port. This one is
dial-in port which is named "ttyUSBxx".
```

3. Module driver configuration

To simplify the processes behind, we provide a single step to build, install and load the MOXA driver. You may execute the ./mxinstall to use the MOXA product. Once you successfully execute ./mxinstall, you can skip step 3.3.1 and 3.3.2.

Build the MOXA driver

Before using the MOXA driver, you need to compile all the source code. This step only needs to be executed once. But you still re-compile the source code if you modify the source code.

Find "Makefile" in /moxa/mxu11x0/driver, then run

```
# make clean; make install
```

The driver files "mxu11x0.ko" will be properly compiled and copied to system directories respectively.

Load the MOXA driver

The driver will be loaded automatically while plugging the UPort 1110/1130/1130I/1150/1150I or USB console cable into your PC. Besides, you can load the driver manually.

modprobe mxu11x0

It will activate the module driver. You may run "lsmod" to check if "mxu11x0" is activated. Before you load this module driver, you have to run "lsmod" to check if "usbcore" and "usbserial" are activated.

4. Static driver configuration

NOTE

To use a static driver, you must install the linux kernel source package.

Create link

```
# cd /usr/src/<kernel-source directory>/drivers/usb/serial/
# ln -s /moxa/mxu11x0/driver/mxu11x0.c mxu11x0.c
# ln -s /moxa/mxu11x0/driver/mxu11x0.h mxu11x0.h
# ln -s /moxa/mxu11x0/driver/mxu1110_fw.h mxu1110_fw.h
# ln -s /moxa/mxu11x0/driver/mxu1130_fw.h mxu1130_fw.h
# ln -s /moxa/mxu11x0/driver/mxu1131_fw.h mxu1131_fw.h
# ln -s /moxa/mxu11x0/driver/mxu1150_fw.h mxu1150_fw.h
# ln -s /moxa/mxu11x0/driver/mxu1151_fw.h mxu1151_fw.h
# ln -s /moxa/mxu11x0/driver/mxu3001_fw.h mxu3001_fw.h
# ln -s /moxa/mxu11x0/driver/mxu7001_fw.h mxu7001_fw.h
```

Modify kernel configuration file.

Add the following line into configuration file.

```
/usr/src/<kernel-source directory>/drivers/usb/serial/Kconfig
...
config USB_SERIAL_CONSOLE
...
config USB_SERIAL_GENERIC
...
config MOXA_UPORT_11X0    <-- Add the lines.
tristate "USB Moxa UPort 11x0 Driver" <--
depends on USB_SERIAL     <--
...
```

Modify the kernel Makefile

Add the following line to the last line of Makefile.

```
/usr/src/<kernel-source directory>/drviers/usb/serial/Makefile
...
...
...
obj-$(CONFIG_MOXA_UPORT_11X0) += mxu11x0.o <-- Add the line.
```

Setup kernel configuration

Configure the kernel:

```
# cd /usr/src/<kernel-source directory>
# make menuconfig
```

You will go into a menu-driven system. Please select [Device Drivers] [USB Support], [USB Serial Converter support]. Enable both the [USB Serial Converter support] and the [USB MOXA UPORT 11x0 Driver] drivers with "[*]" by pressing the space bar for built-in (not "[M]"), then select [Exit] to exit this program and save the kernel configurations.

Rebuild kernel

The following are for Linux kernel rebuilding, for your reference only.

For details, please refer to the Linux document.

a. cd /usr/src/<kernel-source directory>
b. make
c. make modules
d. make modules_install
e. make install

5. Verify driver installation

You may refer to /var/log/syslog to check the latest status log reported by this driver whenever it's activated or type command "dmesg" to get driver information.

The Following shows the messages when installing UPort 1150.

```
usbcore: registered new interface driver mxu11x0
usbserial: USB Serial support registered for MOXA UPort 1110
usbserial: USB Serial support registered for MOXA UPort 1130
usbserial: USB Serial support registered for MOXA UPort 1150
usbserial: USB Serial support registered for MOXA UPort 1150I
usbserial: USB Serial support registered for MOXA UPort 1130I
usbserial: USB Serial support registered for MOXA USB Console
usbserial: USB Serial support registered for MOXA USB-to-Serial Port Driver
mxu11x0 1-2.2:1.0: MOXA UPort 1150 converter detected
usb 1-2.2: MOXA UPort 1150 converter now attached to ttyUSB0
mxu11x0: Ver6.0: MOXA UPort 11x0 USB to Serial Hub Driver
```

Above message indicates /dev/ttyUSB0 are installed successfully.

6. Uninstall

```
# cd /moxa/mxu11x0
# make remove
```

For Linux Kernel 4.x

Introduction

This document details porting the Moxa UPort driver to a specified Arm-based platform. The following knowledge is recommended before reading the instructions in this guide.

- Linux Kernel Programming
- ARM Platform Compiler
- Yocto Project Document
- MOXA UC-Series Manual
- Raspberry Pi Manual

Instructions in this guide use examples of porting on the Moxa UC-Series Arm platform and Raspberry Pi. You can apply the experience of porting UPort driver to other platforms.

The UPort driver fully supports all modern-day Linux distributions running on x86 environments, and the driver core is also compatible with the Arm platform. This document will guide you on how to port the UPort driver core.

However, some platform-dependent services, such as installer, are not available. You may refer to the platform's documentation to fulfill the requirements.

Porting to Moxa UC-Series - Arm-based Computer

Build binaries on a general ARM platform

If your platform is powerful and consists of the necessary development tools, the driver can be built on the platform directly. You can refer to readme.txt of UPort driver to understand the requirement.

The step of building this driver in an ARM environment is the same as in x86 and x64 environments.

UPort 1100 Series driver

```
# tar zxvf [DRIVER NAME].tgz
# cd mxu11x0
# ./mxinstall
```

Cross-compiler and the UPort driver

NOTE

To cross-compiler on a x86 or x64 Linux host, the target ARM environment's kernel source package and cross compiler toolchain must be installed first.

After installing and configuring the kernel source package and toolchain, you need to compile the source code with the kernel source package and toolchain.

In this example, we install the cross-compiler for the Moxa UC-Series ARM-based computer. You can refer to the product's manual for further detail.

1. Download the kernel source package webpage under the product page.

   ```
   $ git clone https://github.com/Moxa-Linux/am335x-linux-4.4
   ```

   You can use the following commands to show the git tag list and check out the tag of the specific UC device and firmware version.

   Show the tag list:

   ```
   $ git tag
   ```

   Check out the specific tag:

   ```
   $ git checkout UC-2100_V1.7 ← Replace the tag name, UC-2100_V1.7,
                                  with the UC Series that is being used.
   ```

2. Download the toolchain from the product's webpage. The toolchain, which is used by the UC Series, is arm-linux-gnueabihf. It is a script that will install the related packages. Execute the script and follow the steps to install the Linux cross-compiler tools. You will need the root privilege to install the toolchain and the kernel source.

   ```
   # sh arm-linux-gnueabihf_6.3_Build_amd64_<build_date>.sh
   ```

   If the script shows the notification message: "Please export these environment variables before using the toolchain", enter the following script command:

   ```
   # export PATH=$PATH:/usr/local/arm-linux-gnueabihf-6.3/usr/bin
   ```

3. The kernel source, which is used by the UC Series, is am335x-linux-4.4. You need to configure these files before starting to cross-compile.

   Move the kernel source to /moxa/kernel and configure the kernel source.

   ```
   # mv am335x-linux-4.4 /moxa/kernel
   # cd /moxa/kernel
   # make uc2100_defconfig ← Replace the uc2100 with the UC Series
                             that is being used.
   # make modules_prepare
   ```

   After the abovementioned steps, please follow the processes as set out in Section 2.3, "Moxa cross-compiling interactive script," and Section 2.4, "Manually build the UPort driver with a cross-compiler," to cross-compile Moxa's driver for the UC-Series platforms.

   The UPort driver, which includes the driver modules, needs to be compiled. The file of each UPort Series is listed as follow:

   UPort 1100 Series driver

   - mxu11x0.ko

   If it is preferred to build these binaries with automatic script, please refer to Section 2.3, "Moxa cross-compiling interactive script." If you find the build script troublesome, or you prefer to build these binaries manually, please refer to section 2.4 "Manually build the UPort driver with a cross-compiler."

   If you have generated the necessary binaries, please refer to section 2.5 to deploy to the target platform.

Moxa cross-compiling interactive script

To simplify the processes above, Moxa has provided an interactive script, "mxcc", to cross-compile these drivers. You may execute ./mxcc in the UPort driver source directory to cross-compile the MOXA driver.

For the UPort 1100 Series

The steps are as follows:

```
#./mxcc
Enter target device architecture (ARCH) [arm]:
Enter cross-compile (CROSS_COMPILE) [arm-linux-gnueabihf-]:
Enter target device kernel source directory [/moxa/kernel]:
****************************************
mxu11x0 cross-compile is success.
****************************************

**************************************************************************
MOXA UPort 1100 Series driver cross-compile finished.
If cross compile is success, driver is outputted to the output folder.
**************************************************************************
```

The binaries will now be generated and placed in the output directory under the /moxa/mxu11x0 folder.

Manually build the UPort driver with a cross-compiler

For the UPort 1100 Series

To cross-compile the UPort 1100 Series driver, you can find "Makefile" in the driver folder, and then run it.

```
# make ARCH=<ARCH> CROSS_COMPILE=<CROSS_COMPILE> KDIR=<KERNEL_SOURCE>
KVER_MAJOR=<KERNEL_MAJOR> KVER_MINOR=<KERNEL_MINOR>
```

<ARCH>: The target ARM environment device's CPU architecture. For example, arm, arm64.

<CROSS_COMPILE>: The cross-compile toolchain path. If the toolchain is arm-linux-gnueabihf, and you have already added the path of the toolchain to the environment variable, you should enter "arm-linux-gnueabihf-" here.

<KERNEL_SOURCE>: The directory of the target kernel source.

<KERNEL_MAJOR>: The major version of the target ARM system's kernel source. You can use the command "make kernelversion" to get the kernel source's major version.

e.g.,

```
# make kernelversion
4.4.0
|
+--- kernel major version
```

<KERNEL_MINOR>: The minor version of the target ARM system's kernel source. You can use the command "make kernelversion" to get the kernel source's minor version.

e.g.,

```
# make kernelversion
4.4.0
  |
  +--- kernel minor version
```

The "make" command would be similar to the following example:

```
#make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- KDIR=/moxa/kernel KVER_MAJOR=4
KVER_MINOR=4
```

After using the "make" command to cross-compile the drivers, the driver file "mxu11x0.ko" can be found in the /moxa/ mxu11x0/driver directory.

Deploy cross-compiled binary to target

For the UPort 1100 Series

You should find the kernel module, mxu11x0.ko, under the output or driver source code directory.

Follow the steps below to deploy to the target Arm platform.

1. Copy the mxu11x0.ko to the path
   /lib/modules/`uname -r`/kernel/drivers/char on the ARM platform.
2. Boot into the ARM platform and load the driver.
   ```
   # modprobe mxu11x0
   ```

Porting to Raspberry Pi OS

Raspberry Pi OS images are prebuilt by www.raspberrypi.org. You can install the image and start up the system. The process to build the UPort Series driver is the same as with x86 Linux. Please refer to readme.txt to check the system requirements.

You may use the rpi-source to install the kernel source packages for a more convenient option. Please refer to the official website https://github.com/notro/rpi-source/wiki for more information.

rpi-source is a third-party package offering an integrated kernel resource for building a driver. The UPort is tested with this package to see if it works well. However, the requirements may vary for different Raspberry Pi OS versions. Please read the manual of rpi-source to understand the know-how and the limitations.

Porting to the Yocto Project on Raspberry Pi

Prerequisite

You are expected to be familiar with the Yocto Project. Please refer to https://docs.yoctoproject.org for the Yocto Project documentation for further understanding. Also, it is encouraged to follow the procedures in this guide unless you have sufficient knowledge about the UPort driver, the Yocto Project, and Raspberry Pi.

The branch zeus is referred to throughout in this section. Please base it on this version before reading the instructions in the Yocto Project documentation. You are required to build the Yocto image successfully with the "Yocto Project Quick Build" document.

In Yocto Project, you can select the platform you want to build. This guide installs Raspberry Pi BSP Layer as a demonstration in the following steps:

1. Suppose the Yocto is installed in /home/user/poky folder. Check out the source code of the Raspberry Pi BSP Layer.

   ```
   # cd /home/user/poky
   # git clone https://git.yoctoproject.org/cgit/cgit.cgi/meta-raspberrypi -b
   zeus
   ```

2. A meta-raspberrypi folder will be checked out now. Use the following instructions to set up Raspberry Pi BSP:

   ```
   # source oe-init-build-env
   ```

3. Use a text editor to add the following content to the configuration file './conf/local.conf'

   Add the type 'rpi-sdimg' optionally if an SD card is preferred
   IMAGE_FSTYPES="tar.bz2 ext3 rpi-sdimg"

   Change the machine name of your target
   Use raspberrypi2 for Pi 2 board
   Use raspberrypi3 for Pi 3 board
   Use raspberrypi3-64 for 64-bit Pi 3 board

   MACHINE ?= "raspberrypi2"

4. Use the text editor to add the following content to the configuration file './conf/bblayers.conf'

   Add this line '/home/user/poky/meta-raspberrypi' to BBLAYERS
   BBLAYERS ?= " \
   /home/user/poky/meta \
   /home/user/poky/meta-poky \
   /home/user/poky/meta-yocto-bsp \
   /home/user/poky/meta-raspberrypi \
   "

5. Build the target core-image-base by following this command and the Raspberry Pi image will be generated.

   ```
   # bitbake core-image-base
   ```

Once the above image runs on Raspberry Pi, go to the next section.

Create a Moxa layer for the Yocto Project

Introduction

Moxa UPort driver is packaged as a layer for Yocto. You can add or remove the driver by modifying BBLAYERS attribute in the bblayers.conf file.

The following sections describe how to create a meta-moxa layer for the zeus branch. Note that the process may vary if your target uses a different branch. Please refer to Yocto's manual for complete information.

An example is also available in the examples folder in the UPort Series driver.

You may follow the subsequent procedures to create the same meta-moxa layer.

Create an empty Moxa layer

Use the following commands to create an empty layer, named meta-moxa.

1. Initiate the environment first. Suppose the project is installed in /home/user/poky.

```
$ cd /home/user/poky
$ source oe-init-build-env
```

2. The above commands changed the directory to the built directory. Now, we change the directory back to the Yocto root directory.

```
$ cd /home/user/poky
```

3. Create meta-moxa:

   A message appears reminding you to add the layer later.

```
$ bitbake-layers create-layer meta-moxa
NOTE: Starting bitbake server...
Add your new layer with 'bitbake-layers add-layer meta-moxa'
```

   The meta-moxa directory will be created in /home/user/poky.

```
$ tree meta-moxa
```

```
meta-moxa
├── conf
│   └── layer.conf
├── COPYING.MIT
├── README
└── recipes-example
    └── example
        └── example_0.1.b
```

   The "recipes-example" folder is not necessary; it may be deleted at any time.

Create a recipe for the UPort driver

Use the following commands to create a recipe for installing the UPort driver kernel to the target platform.

1. Create a directory recipes-kernel in meta-moxa.

```
$ cd /home/user/poky
$ mkdir meta-moxa/recipes-kernel
```

2. The simplest way is to copy and modify from a hello example, which is available in the Yocto source code.

```
$ cp -r ./meta-skeleton/recipes-kernel/hello-mod ./meta-moxa/recipes-kernel
```

   The content of meta-moxa is now listed below:

```
$ tree meta-moxa
```

```
meta-moxa/
├── conf
│   └── layer.conf
├── COPYING.MIT
├── README
└── recipes-kernel
    └── hello-mod
        ├── files
        │   ├── COPYING
        │   ├── hello.c
        │   └── Makefile
        └── hello-mod_0.1.bb
```

3. Delete the unnecessary files in hello-mod and rename the hello-mod.

   For the UPort 1100 Series

```
$ cd ./meta-moxa/recipes-kernel
$ rm ./hello-mod/files/COPYING
$ rm ./hello-mod/files/hello.c
$ mv ./hello-mod/hello-mod_0.1.bb ./hello-mod/uport1100_0.1.bb
$ mv ./hello-mod uport1100
```

4. Extract the UPort driver source code. Copy the following files into uport1100:

   For the UPort 1100 Series

```
$ cp /moxa/ mxu11x0/COPYING-GPLV2.TXT ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu11x0.c ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu11x0.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu1110_fw.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu1130_fw.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu1131_fw.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu1150_fw.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu1151_fw.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu3001_fw.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mx_dist.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mx_ver.h ./uport1100/files/
```

5. The content of the recipes-kernel is listed below:

   For the UPort 1100 Series

```
$ tree ./
./
└── uport1100
    ├── files
    │   ├── COPYING-GPLV2.TXT
    │   ├── Makefile
    │   ├── mx_dist.h
    │   ├── mxu1110_fw.h
    │   ├── mxu1130_fw.h
    │   ├── mxu1131_fw.h
    │   ├── mxu1150_fw.h
    │   ├── mxu1151_fw.h
    │   ├── mxu11x0.c
    │   ├── mxu11x0.h
    │   ├── mxu3001_fw.h
    │   └── mx_ver.h
    └── uport1100_0.1.bb
```

6. Modify the content of 'Makefile' in the files folder as follows:

   For the UPort 1100 Series

```
##### Makefile start #####
obj-m := mxu11x0.o
SRC := $(shell pwd)
all:
   $(MAKE) -C $(KERNEL_SRC) M=$(SRC)
modules_install:
   $(MAKE) -C $(KERNEL_SRC) M=$(SRC) modules_install
clean:
   rm -f *.o *~ core .depend .*.cmd *.ko *.mod.c
   rm -f Module.markers Module.symvers modules.order
   rm -rf .tmp_versions Modules.symvers
##### Makefile end #####
```

7. Modify the content of the file '*_0.1.bb' as follows:

   For the UPort 1100 Series

   We have to modify the content of the file 'uport1100_0.1.bb':

   ```
   ##### uport1100_0.1.bb start #####
   DESCRIPTION = "Linux kernel module for Moxa UPort 11x0 Series"
   LICENSE = "GPLv2"
   LIC_FILES_CHKSUM = "file://COPYING-GPLV2.TXT;md5=5205bcd21ef6900c98e19cf948c26b41"
   inherit module
   SRC_URI = "file://Makefile \
            file://mx_dist.h \
            file://mxu1110_fw.h \
            file://mxu1130_fw.h \
            file://mxu1131_fw.h \
            file://mxu1150_fw.h \
            file://mxu1151_fw.h \
            file://mxu11x0.h \
            file://mxu3001_fw.h \
            file://mx_ver.h \
            file://mxu11x0.c \
            file://COPYING-GPLV2.TXT \
           "
   S = "${WORKDIR}"
   # The inherit of module.bbclass will automatically name module packages with
   # "kernel-module-" prefix as required by the oe-core build environment.
   RPROVIDES_${PN} += "kernel-module-mxu11x0"
   ##### uport1100_0.1.bb end #####
   ```

8. If you are using the UPort 1200/1400/1600 Series driver, you have to modify the content of the file 'mx-uport.c'.

   Replace the following line

   ```
   #include "../mxusbserial/mxusb-serial.h"
   ```

   by

   ```
   #include "mxusb-serial.h"
   ```

Install a Moxa layer into the Yocto Project

1. Install the Moxa layer and UPort driver recipes into the Yocto Project.

   ```
   $ cd /home/user/poky
   $ source oe-init-build-env
   ```

   Use a text editor to add the following content to the configuration file: './conf/bblayers.conf'

   Add this line '/home/user/poky/meta-moxa' to BBLAYERS
   BBLAYERS ?= " \
   /home/user/poky/meta \
   /home/user/poky/meta-poky \
   /home/user/poky/meta-yocto-bsp \
   /home/user/poky/meta-raspberrypi \
   /home/user/poky/meta-moxa \

2. Use a text editor to add the following content to the configuration file: './conf/local.conf'.

   For the UPort 1100 Series
   IMAGE_INSTALL_append += " uport1100"

Deploy the Yocto image in Raspberry Pi

Build the image with the UPort driver.

```
$ cd /home/user/poky
$ source oe-init-build-env
$ bitbake core-image-base
```

A SD-card format image (.rpi-sdimg) is generated under /home/user/poky/build/tmp/deploy/images/raspberrypi2. It is suggested to use the Raspberry Pi official tool 'rpi-imager' to burn the image into the SD-card and then boot it into the Linux kernel in Raspberry Pi.

Start the UPort driver in Raspberry Pi

After logging into the system, start the UPort driver

For the UPort 1100 Series

```
root@raspberrypi2:~# modprobe mxu11x0
```

Troubleshooting

If the following error is encountered during the building of the image,

ERROR: Task (/home/user/poky/meta/recipes-devtools/binutils/binutils_2.34.bb:do_compile) failed with exit code '1'

It is suggested to compile binutils first, then compile the entire image:

```
$ bitbake binutils -c do_compile
$ bitbake core-image-base
```

For Linux Kernel 5.x

Introduction

This document details porting the Moxa UPort driver to a specified Arm-based platform. The following knowledge is recommended before reading the instructions in this guide.

- Linux Kernel Programming
- ARM Platform Compiler
- Yocto Project Document
- MOXA UC-Series Manual
- Raspberry Pi Manual

Instructions in this guide use examples of porting on the Moxa UC-Series Arm platform and Raspberry Pi. You can apply the experience of porting UPort driver to other platforms.

The UPort driver fully supports all modern-day Linux distributions running on x86 environments, and the driver core is also compatible with the Arm platform. This document will guide you on how to port the UPort driver core.

However, some platform-dependent services, such as installer, are not available. You may refer to the platform's documentation to fulfill the requirements.

Porting to Moxa UC-Series - Arm-based Computer

Build binaries on a general ARM platform

If your platform is powerful and consists of the necessary development tools, the driver can be built on the platform directly. You can refer to readme.txt of the UPort driver to understand the requirement.

The step of building this driver in an ARM environment is the same as in x86 and x64 environments.

UPort 1100 Series driver

```
# tar zxvf [DRIVER NAME].tgz
# cd mxu11x0
# ./mxinstall
```

Cross-compiler and the UPort driver

NOTE

To cross-compile on a x86 or x64 Linux host, the target ARM environment's kernel source package and cross compiler toolchain must be installed first.

After installing and configuring the kernel source package and toolchain, you need to compile the source code with the kernel source package and toolchain.

V5.1 driver is for the Linux kernel source 5. In the following example, we demonstrate the cross-compiler on the Moxa UC-Series ARM-based computer, which runs kernel source 4. However, the steps are identical. You can refer to the product's manual for further detail.

1. Download the kernel source package webpage under the product page.

   ```
   $ git clone https://github.com/Moxa-Linux/am335x-linux-4.4
   ```

   You can use the following commands to show the git tag list and check out the tag of the specific UC device and firmware version.

   Show the tag list:

   ```
   $ git tag
   ```

   Check out to the specific tag:

   ```
   $ git checkout UC-2100_V1.7 ← Replace the tag name, UC-2100_V1.7,
                                 with the UC Series that is being used.
   ```

2. Download the toolchain from the product's webpage. The toolchain, which is used by the UC Series, is arm-linux-gnueabihf. It is a script that will install the related packages. Execute the script and follow the steps to install the Linux cross-compiler tools. You will need the root privilege to install the toolchain and the kernel source.

   ```
   # sh arm-linux-gnueabihf_6.3_Build_amd64_<build_date>.sh
   ```

   If the script shows the notification message: "Please export these environment variables before using the toolchain", enter the following script command:

   ```
   # export PATH=$PATH:/usr/local/arm-linux-gnueabihf-6.3/usr/bin
   ```

3. The kernel source, which is used by the UC Series, is am335x-linux-4.4. You need to configure these files before starting to cross-compile.

   Move the kernel source to /moxa/kernel and configure the kernel source.

   ```
   # mv am335x-linux-4.4 /moxa/kernel
   # cd /moxa/kernel
   # make uc2100_defconfig ← Replace the uc2100 with the UC Series
                             that is being used.
   # make modules_prepare
   ```

   After the abovementioned steps, please follow the processes as set out in Section 2.3, "Moxa cross-compiling interactive script," and Section 2.4, "Manually build the UPort driver with a cross-compiler," to cross-compile Moxa's driver for the UC-Series platforms.

   The UPort driver, which includes the driver modules, needs to be compiled. The file of each UPort Series is listed as follow:

   UPort 1100 Series driver

   - mxu11x0.ko

   If it is preferred to build these binaries with automatic script, please refer to Section 2.3, "Moxa cross-compiling interactive script." If you find the build script troublesome, or you prefer to build these binaries manually, please refer to Section 2.4 "Manually build the UPort driver with a cross-compiler."

   If you have generated the necessary binaries, please refer to section 2.5 to deploy to the target platform.

Moxa cross-compiling interactive script

To simplify the processes above, Moxa has provided an interactive script, "mxcc", to cross-compile these drivers. You may execute ./mxcc in the UPort driver source directory to cross-compile the MOXA driver.

For the UPort 1100 Series

The steps are as follows:

```
#./mxcc
Enter target device architecture (ARCH) [arm]:
Enter cross-compile (CROSS_COMPILE) [arm-linux-gnueabihf-]:
Enter target device kernel source directory [/moxa/kernel]:
****************************************
mxu11x0 cross-compile is success.
****************************************

************************************************************************
MOXA UPort 1100 Series driver cross-compile finished.
If cross compile is success, driver is outputted to the output folder.
************************************************************************
```

The binaries will now be generated and placed in the output directory under the /moxa/mxu11x0 folder.

Manually build the UPort driver with a cross-compiler

For the UPort 1100 Series

To cross-compile the UPort 1100 Series driver, you can find "Makefile" in the driver folder, and then run it.

```
# make ARCH=<ARCH> CROSS_COMPILE=<CROSS_COMPILE> KDIR=<KERNEL_SOURCE>
KVER_MAJOR=<KERNEL_MAJOR> KVER_MINOR=<KERNEL_MINOR>
```

<ARCH>: The target ARM environment device's CPU architecture. For example, arm, arm64.

<CROSS_COMPILE>: The cross-compile the toolchain path. If the toolchain is arm-linux-gnueabihf, and you have already added the path of the toolchain to the environment variable, you should enter "arm-linux-gnueabihf-" here.

<KERNEL_SOURCE>: The directory of the target kernel source.

<KERNEL_MAJOR>: The major version of the target ARM system's kernel source. You can use the command "make kernelversion" to get the kernel source's major version.

e.g.,

```
# make kernelversion
4.4.0
|
+--- kernel major version
```

<KERNEL_MINOR>: The minor version of the target ARM system's kernel source. You can use the command "make kernelversion" to get the kernel source's minor version.

e.g.,

```
# make kernelversion
4.4.0
  |
  +--- kernel minor version
```

The "make" command would be similar to the following example:

```
# make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- KDIR=/moxa/kernel KVER_MAJOR=4
KVER_MINOR=4
```

After using the "make" command to cross-compile the drivers, the driver files "mxu11x0.ko" can be found in the /moxa/mxu11x0/driver directory.

Deploy cross-compiled binary to target

For the UPort 1100 Series

You should find the kernel module, mxu11x0.ko, under the output or driver source code directory.

Follow the steps below to deploy to the target Arm platform.

1. Copy the mxu11x0.ko to the path
   /lib/modules/`uname -r`/kernel/drivers/char on the ARM platform.
2. Boot into the ARM platform and load the driver.
   ```
   # modprobe mxu11x0
   ```

Porting to Raspberry Pi OS

Raspberry Pi OS images are prebuilt by www.raspberrypi.org. You can install the image and start up the system. The process to build the UPort Series driver is the same as with x86 Linux. Please refer to readme.txt to check the system requirements.

You may use the rpi-source to install the kernel source packages for a more convenient option. Please refer to the official website https://github.com/notro/rpi-source/wiki for more information.

rpi-source is a third-party package offering an integrated kernel resource for building a driver. The UPort is tested with this package to see if it works well. However, the requirements may vary for different Raspberry Pi OS versions. Please read the manual of rpi-source to understand the know-how and the limitations.

Porting to the Yocto Project on Raspberry Pi

Prerequisite

You are expected to be familiar with the Yocto Project. Please refer to https://docs.yoctoproject.org for the Yocto Project documentation for further understanding. Also, it is encouraged to follow the procedures in this guide unless you have sufficient knowledge about the UPort driver, the Yocto Project, and Raspberry Pi.

The dunfell branch (3.1.9) is referred to throughout in this section. Please base it on this version before reading the instructions in the Yocto Project documentation. You are required to build the Yocto image successfully with the "Yocto Project Quick Build" document.

In Yocto Project, you can select the platform you want to build. This guide installs Raspberry Pi BSP Layer as a demonstration in the following steps:

1. Suppose the Yocto is installed in /home/user/poky folder. Check out the source code of the Raspberry Pi BSP Layer.
   ```
   # cd /home/user/poky
   # git clone https://git.yoctoproject.org/cgit/cgit.cgi/meta-raspberrypi -b dunfell
   ```
2. A meta-raspberrypi folder will be checked out now. Use the following instructions to set up Raspberry Pi BSP:
   ```
   # source oe-init-build-env
   ```
3. Use a text editor to add the following content to the configuration file './conf/local.conf'

   Add the type 'rpi-sdimg' optionally if an SD card is preferred
   IMAGE_FSTYPES="tar.bz2 ext3 rpi-sdimg"

   Change the machine name of your target
   Use raspberrypi2 for Pi 2 board
   Use raspberrypi3 for Pi 3 board
   Use raspberrypi3-64 for 64-bit Pi 3 board

   MACHINE ?= "raspberrypi2"

4. Use the text editor to add the following content to the configuration file './conf/bblayers.conf'

   Add this line '/home/user/poky/meta-raspberrypi' to BBLAYERS
   BBLAYERS ?= " \
   /home/user/poky/meta \
   /home/user/poky/meta-poky \
   /home/user/poky/meta-yocto-bsp \
   /home/user/poky/meta-raspberrypi \
   "

5. Build the target core-image-base by following this command and the Raspberry Pi image will be generated:

   ```
   # bitbake core-image-base
   ```

Once the above image runs on Raspberry Pi, go to the next section.

Create a Moxa layer for the Yocto Project

Introduction

The Moxa UPort driver is packaged as a layer for Yocto. You can add or remove the driver by modifying BBLAYERS attribute in the bblayers.conf file.

The following sections describe how to create the meta-moxa layer for the dunfell branch (3.1.9). Note that the process may vary if your target uses a different branch. Please refer to Yocto's manual for complete information.

An example is also available in the examples folder in the UPort Series driver.

You may follow the subsequent procedures to create the same meta-moxa layer.

Create an empty Moxa layer

Use the following commands to create an empty layer, named meta-moxa.

1. Initiate the environment first. Suppose the project is installed in /home/user/poky:

   ```
   $ cd /home/user/poky
   $ source oe-init-build-env
   ```

2. The above commands changed the directory to the built directory. Now, we change the directory back to the Yocto root directory.

   ```
   $ cd /home/user/poky
   ```

3. Create meta-moxa:

   A message appears reminding you to add the layer later.

   ```
   $ bitbake-layers create-layer meta-moxa
   NOTE: Starting bitbake server...
   Add your new layer with 'bitbake-layers add-layer meta-moxa'
   ```

   The meta-moxa directory will be created in /home/user/poky.

   ```
   $ tree meta-moxa
   ```

   ```
   meta-moxa
   ├── conf
   │   └── layer.conf
   ├── COPYING.MIT
   ├── README
   └── recipes-example
       └── example
           └── example_0.1.bb
   ```

   The “recipes-example” folder is not necessary; it may be deleted at any time.

Create a recipe for the UPort driver

Use the following commands to create a recipe for installing the UPort driver kernel to the target platform.

1. Create a directory recipes-kernel in meta-moxa:

```
$ cd /home/user/poky
$ mkdir meta-moxa/recipes-kernel
```

2. The simplest way is to copy and modify from a hello example, which is available in the Yocto source code.

```
$ cp -r ./meta-skeleton/recipes-kernel/hello-mod ./meta-moxa/recipes-kernel
```

The content of meta-moxa now is listed below:

```
$ tree meta-moxa
```

```
meta-moxa/
├── conf
│   └── layer.conf
├── COPYING.MIT
├── README
└── recipes-kernel
    └── hello-mod
        ├── files
        │   ├── COPYING
        │   ├── hello.c
        │   └── Makefile
        └── hello-mod_0.1.bb
```

3. Delete the unnecessary files in hello-mod and rename the hello-mod.

For the UPort 1100 Series

```
$ cd ./meta-moxa/recipes-kernel
$ rm ./hello-mod/files/COPYING
$ rm ./hello-mod/files/hello.c
$ mv ./hello-mod/hello-mod_0.1.bb ./hello-mod/uport1100_0.1.bb
$ mv ./hello-mod uport1100
```

4. Extract the UPort driver source code. Copy the following files into uport1100:

For the UPort 1100 Series

```
$ cp /moxa/ mxu11x0/COPYING-GPLV2.TXT ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu11x0.c ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu11x0.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu1110_fw.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu1130_fw.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu1131_fw.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu1150_fw.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu1151_fw.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mxu3001_fw.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/usb-serial.h ./uport1100/files/
$ cp /moxa/ mxu11x0/driver/mx_ver.h ./uport1100/files/
```

5. The content of the recipes-kernel is listed below:

   For the UPort 1100 Series:

```
$ tree ./
./
└── uport1100
    ├── files
    │   ├── COPYING-GPLV2.TXT
    │   ├── Makefile
    │   ├── mxu1110_fw.h
    │   ├── mxu1130_fw.h
    │   ├── mxu1131_fw.h
    │   ├── mxu1150_fw.h
    │   ├── mxu1151_fw.h
    │   ├── mxu11x0.c
    │   ├── mxu11x0.h
    │   ├── mxu3001_fw.h
    │   ├── mx_ver.h
    │   └── usb-serial.h
    └── uport1100_0.1.bb
```

6. Modify the content of 'Makefile' in the files folder as follows:

   For the UPort 1100 Series

```
##### Makefile start #####
obj-m := mxu11x0.o
SRC := $(shell pwd)
all:
   $(MAKE) -C $(KERNEL_SRC) M=$(SRC)
modules_install:
   $(MAKE) -C $(KERNEL_SRC) M=$(SRC) modules_install
clean:
   rm -f *.o *~ core .depend .*.cmd *.ko *.mod.c
   rm -f Module.markers Module.symvers modules.order
   rm -rf .tmp_versions Modules.symvers
##### Makefile end ####
```

7. Modify the content of the file '*_0.1.bb' as follows:

   For the UPort 1100 Series

   We have to modify the content of the file 'uport1100_0.1.bb':

```
##### uport1100_0.1.bb start #####
DESCRIPTION = "Linux kernel module for Moxa UPort 11x0 Series"
LICENSE = "GPLv2"
LIC_FILES_CHKSUM = "file://COPYING-GPLV2.TXT;md5=5205bcd21ef6900c98e19cf948c26b41"
inherit module
SRC_URI = "file://Makefile \
          file://mxu1110_fw.h \
          file://mxu1130_fw.h \
          file://mxu1131_fw.h \
          file://mxu1150_fw.h \
          file://mxu1151_fw.h \
          file://mxu11x0.h \
          file://mxu3001_fw.h \
          file://mx_ver.h \
          file://usb-serial.h \
          file://mxu11x0.c \
          file://COPYING-GPLV2.TXT \
          "
S = "${WORKDIR}"
# The inherit of module.bbclass will automatically name module packages with
# "kernel-module-" prefix as required by the oe-core build environment.
RPROVIDES_${PN} += "kernel-module-mxu11x0"
##### uport1100_0.1.bb end #####
```

Install a Moxa layer into the Yocto Project

1. Install the Moxa layer and UPort driver recipes into the Yocto Project.

   ```
   $ cd /home/user/poky
   $ source oe-init-build-env
   ```

   Use a text editor to add the following content to the configuration file: './conf/bblayers.conf'

   Add this line '/home/user/poky/meta-moxa' to BBLAYERS
   BBLAYERS ?= " \
   /home/user/poky/meta \
   /home/user/poky/meta-poky \
   /home/user/poky/meta-yocto-bsp \
   /home/user/poky/meta-raspberrypi \
   /home/user/poky/meta-moxa \
   "

2. Use a text editor to add the following content to the configuration file: './conf/local.conf'.

   For the UPort 1100 Series
   IMAGE_INSTALL_append += " uport1100"

Deploy the Yocto image in Raspberry Pi

Build the image with UPort driver.

```
$ cd /home/user/poky
$ source oe-init-build-env
$ bitbake core-image-base
```

A SD-card format image (.rpi-sdimg) is generated under /home/user/poky/build/tmp/deploy/images/raspberrypi2. It is suggested to use the Raspberry Pi official tool 'rpi-imager' to burn the image into the SD-card and then boot it into the Linux kernel in Raspberry Pi.

Start the UPort driver in Raspberry Pi

After logging into the system, start the UPort driver

For the UPort 1100 Series

```
root@raspberrypi2:~# modprobe mxu11x0
```

Troubleshooting

If the following error is encountered during the building of the image,

ERROR: Task (/home/user/poky/meta/recipes-devtools/binutils/binutils_2.34.bb:do_compile) failed with exit code '1'

It is suggested to compile binutils first, then compile the entire image:

```
$ bitbake binutils -c do_compile
$ bitbake core-image-base
```

Installing the macOS Driver

For macOS 10.1x

Follow the steps below to install driver

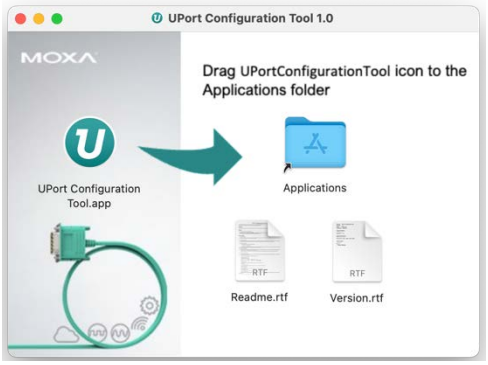
1. Download the driver "Driver for UPort 1100 Series (macOS 10.1x to 11)" from Moxa product website.
2. Enter Recovery Mode.
3. Disable System Integrity Protection (SIP) by typing 'csrutil disable' in the terminal.
4. Return to Normal Mode.
5. Launch the UPort driver installer.
6. Enter Recovery Mode again.
7. Enable System Integrity Protection (SIP) by typing 'csrutil enable' in the terminal.
8. Return to Normal Mode For detailed instructions, please refer to 'readme.txt' in the driver installation package.

For macOS 11 and After

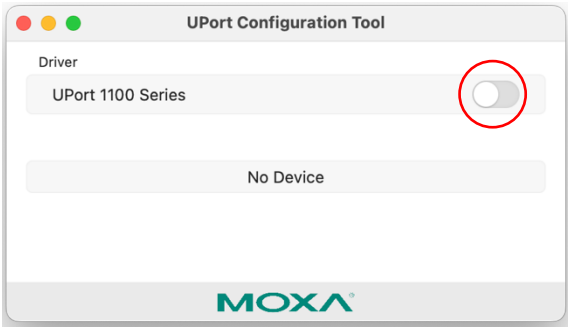
We have provided GUI utility along with the driver in this release. Please download the driver "Driver for the UPort 1100 Series (macOS 11 to 14)" from Moxa product website.

1. While opening the DMG file, there will be
   - UPortConfigurationTool.APP.
   - Readme.rtf
   - Version.rtf.

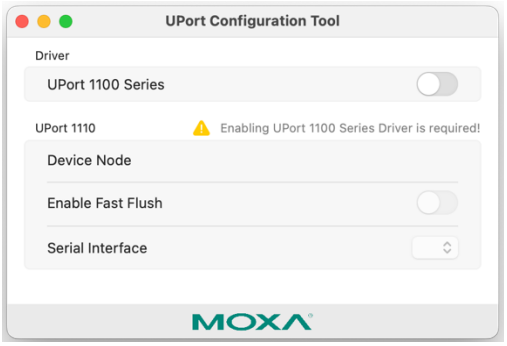
   Drag the UPortConfigurationTool.APP to the Application folder to finish the installation.



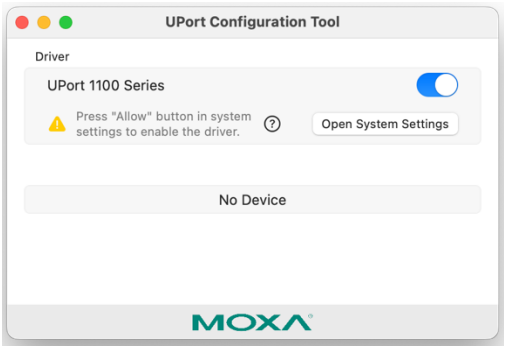
2. Launch **UPort Configuration Tool** from **Applications** folder.
3. After the tool is launched, enabling the driver to use the UPort.

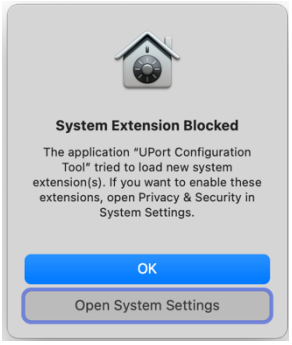


If there is already a UPort 1100 inserted in a USB slot, the system will display the device information and show a different message to remind you to enable the driver first.

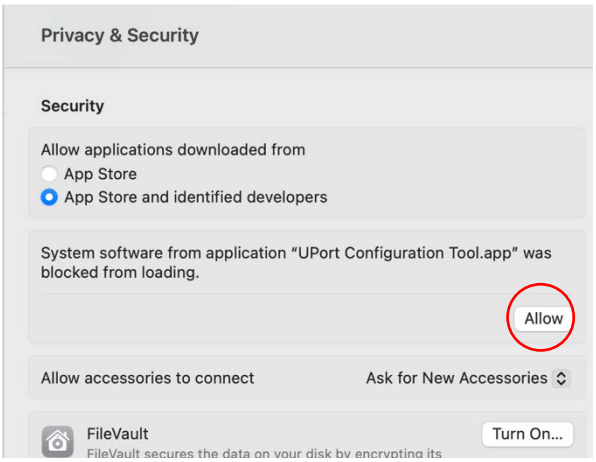


Since the driver is loaded as system software, it asks for permission every time. Click **Open System Settings**.

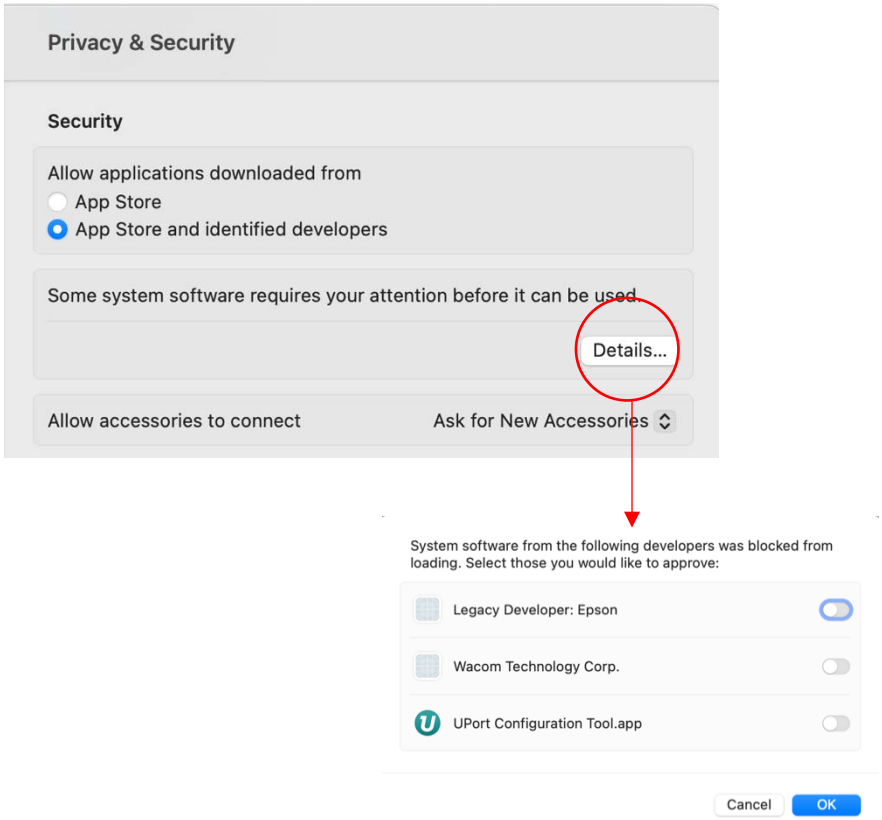




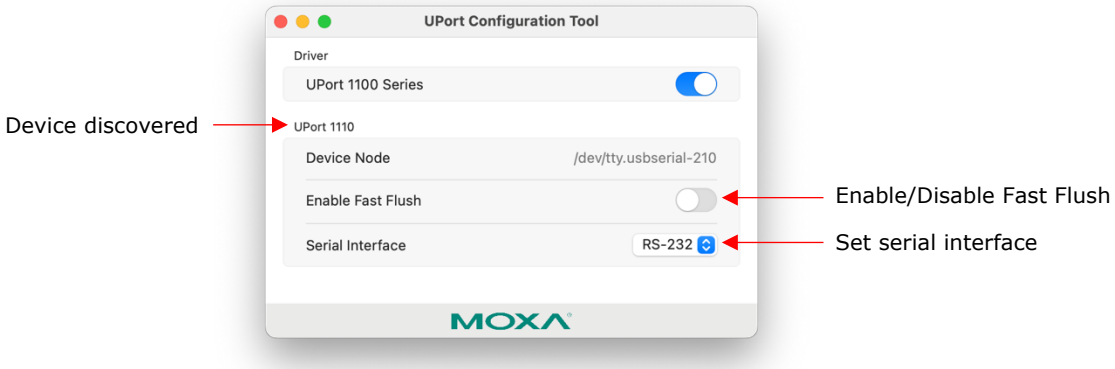
In **Privacy & Security**, click **Allow**.



Or if you have more than one item that requires permission, you will see the format is slightly different:



4. Once you enable the driver, the tool will automatically scan for UPort, and when it finds any, it will display them in the window.



Fast Flash

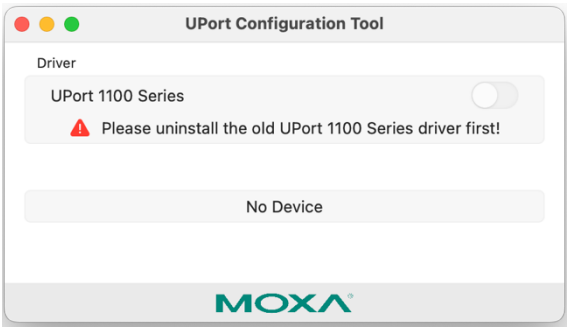
When Fast Flush is enabled, the driver automatically clears the local buffer upon receiving a purge command. When Fast Flush is disabled, the driver will continue to query the converter until it establishes that there is no more data in the buffer. Disabling this function can cause lower throughput for applications that use purge command extensively.

Disabling the Driver

When disabling the driver is unloading software from the system, the system will ask for permission again for the action.

Troubleshooting

If your Macbook has installed an older version of driver, you will need to uninstall it before the new driver can get into action.



Do the following command to remove previous version of driver:

```
#sudo bash /usr/local/share/uport11x0/uninstall.sh
Or
By removing /Library/Extensions/mxu11x0.kext
```

Installing the Windows CE Driver

In this section, the driver installation procedures for installing Windows CE 5.0 and Windows CE 6.0 driver are described. Both WinCE 5.0 and WinCE 6.0 require the installation of WinCE 6.0 platform builder. In addition, you will also need to install Visual Studio 2005 if you are using WinCE 6.0. WinCE 6.0 platform builder has all the necessary tools to help you design, create, build, test, and debug a Windows CE. WinCE 6.0 platform Builder, together with Visual Studio 2005, provides a workspace where you can work on OS designs and projects to build your own embedded system.

With WinCE 6.0, you need to install Visual Studio 2005 & WinCE 6.0 Builder first. When installing Visual Studio 2005, you can select the language with .NET Framework SDK or use the "default."

After you've installed VS 2005; run WinCE 6.0 Builder "setup.exe" to install and download the files. You need to select "x86" for the WinCE 6.0 operating system.

NOTE

This process will take a couple of hours, as the folder size is around 3 GB.

The Win CE drivers support the following models:

- **UPort 1110:** 1-port RS-232 USB-to-Serial Converter.
- **UPort 1130:** 1-port RS-422/485 USB-to-Serial Converter.
- **UPort 1150:** 1-port RS-232/422/485 USB-to-Serial Converter.
- **UPort 1150I:** 1-port RS-232/422/485 USB-to-Serial Converter with isolation protection.
- **UPort 1130I:** 1-port RS-422/485 USB-to-serial converter with isolation protection.

Follow "Installing Visual Studio 2005" to install the UPort 1100 Series WinCE 6.0 driver.

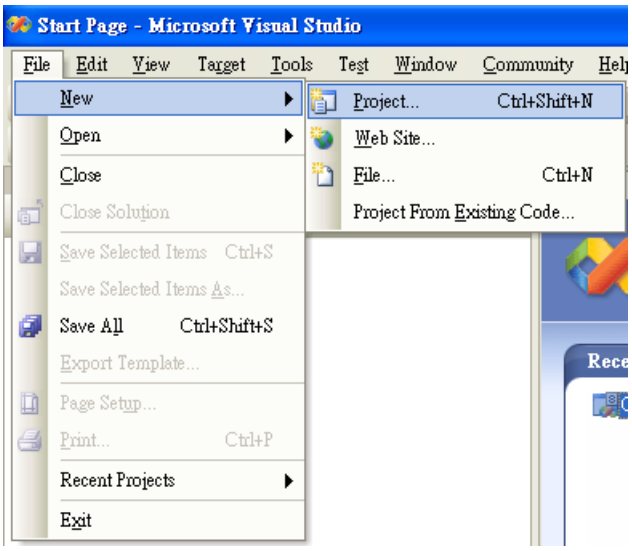
Installation With an Installation Package for Win CE 6.0

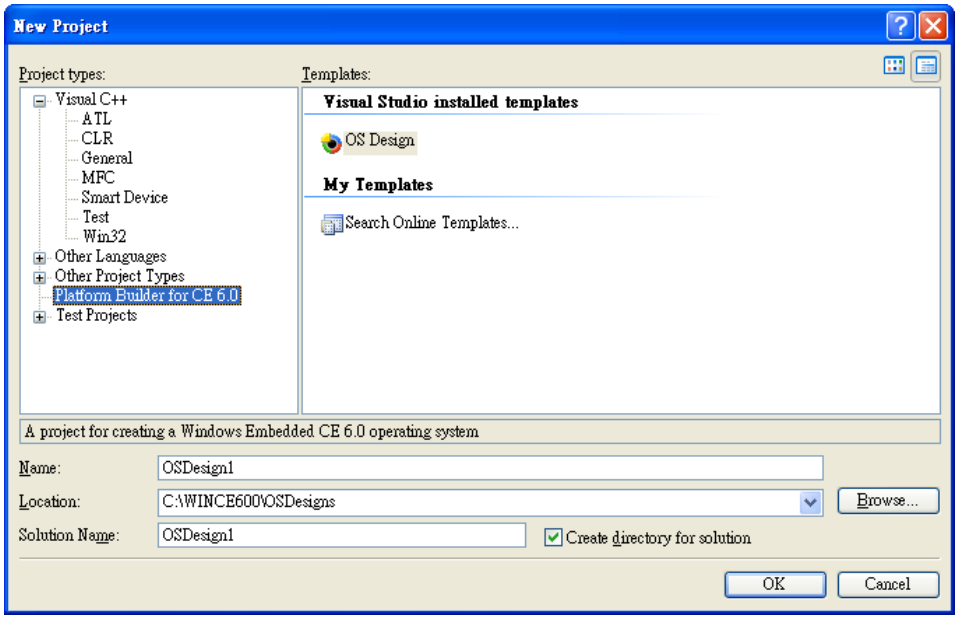
Installation procedure

Copy the UPort 1100 WinCE 6.0 driver package onto your computer and extract. Double-click on the installation package, and it will automatically copy the **Mxser** folder to C:\ MOXAUP1100\wince600\U1100.

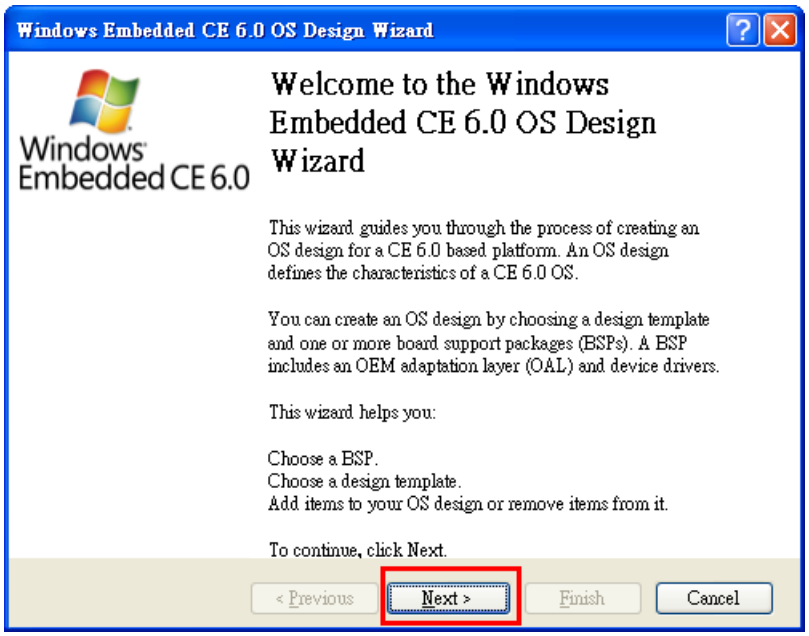
1. Launch Visual Studio 2005 with Platform Builder WinCE6.0. Open the OSDesign that you want to install.

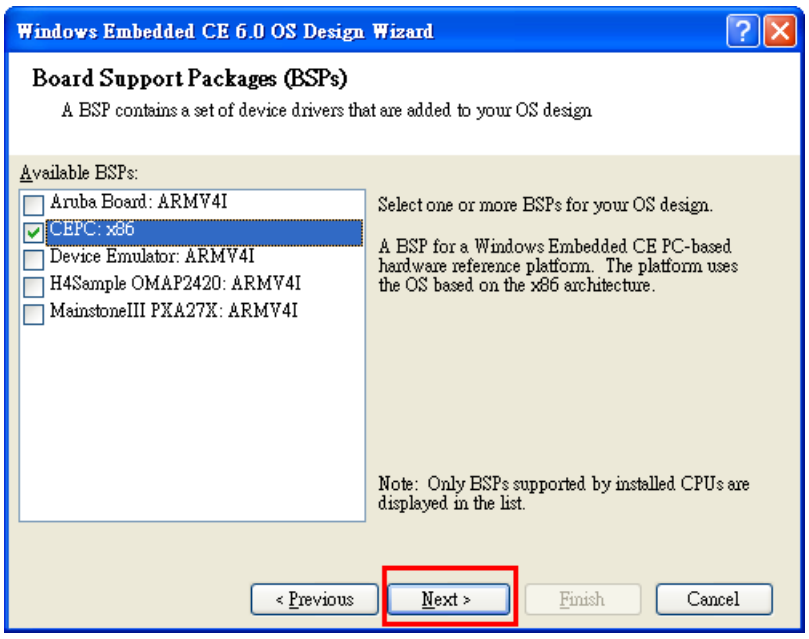
   In Visual Studio 2005, click **File > New > Project** and select "Platform Builder for CE 6.0". Select "OS Design" for the template then click **OK.**



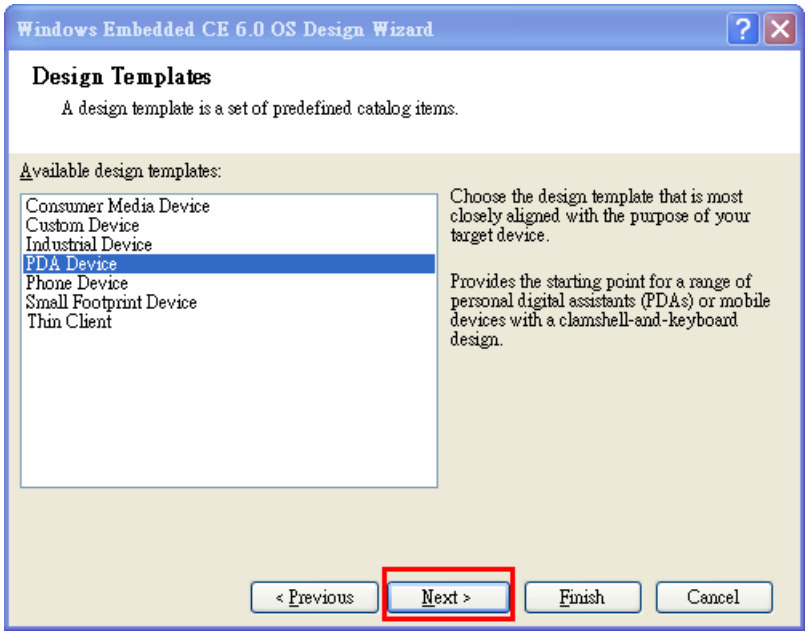


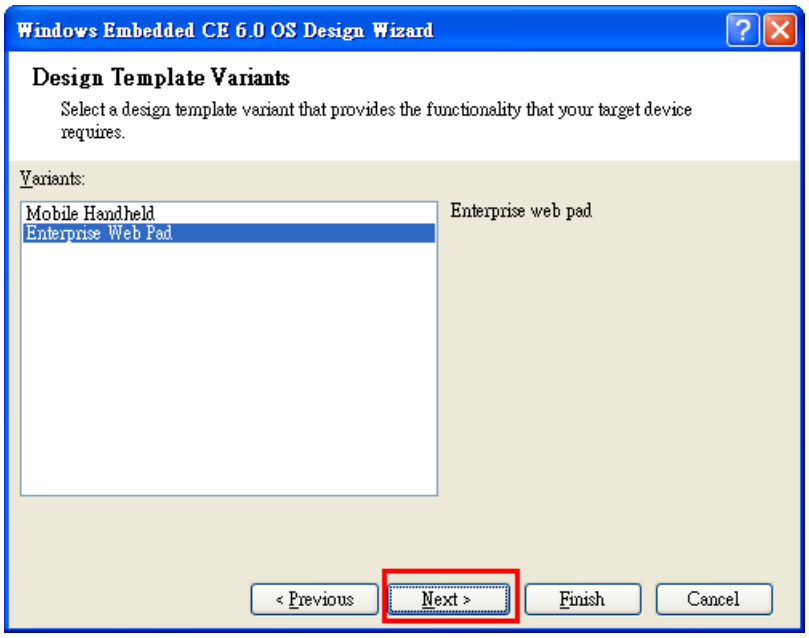
2. The WinCE 6.0 OS Design Wizard will start, click "**Next**" to continue. In the "Board Support Packages" page, select the "CEPC: x86." Click "**Next**."

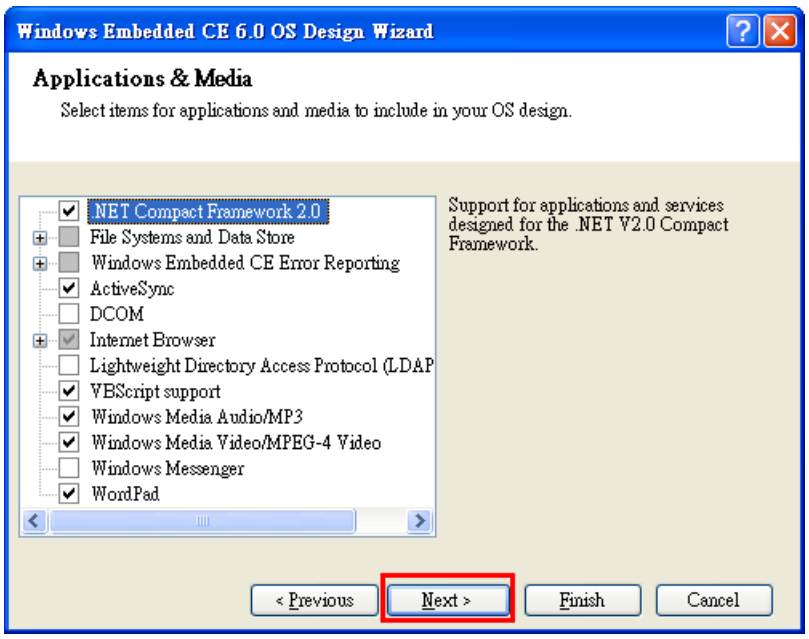


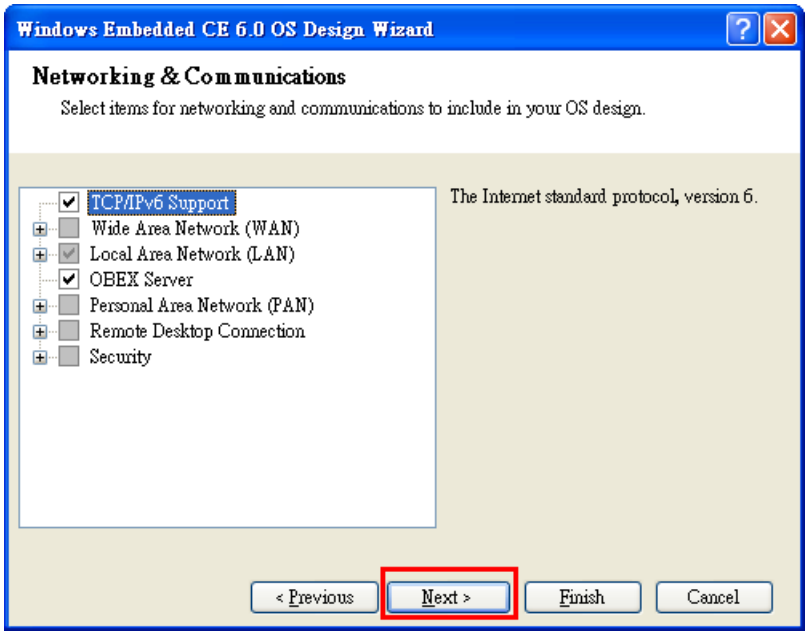


3. Under Design Templates and Design Template Variants, select your environment, such as PDA Device or Mobile Handheld, and then click **Next**. In Application & Media and Networking & Communication, also select your environment, such as .NET Compact Framework 2.0, ActiveSync, Quarter VGA Resources-Portrait Mode, or TCP/IPv6 Support.

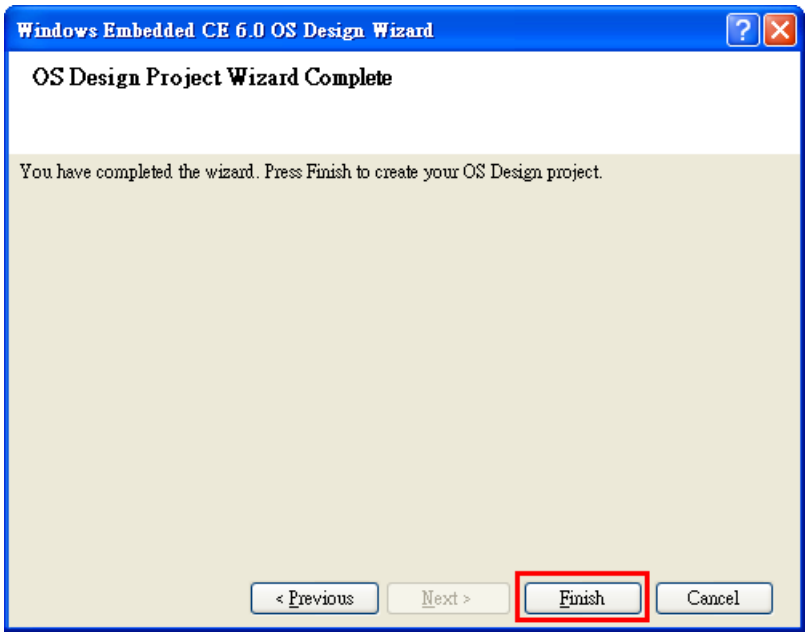




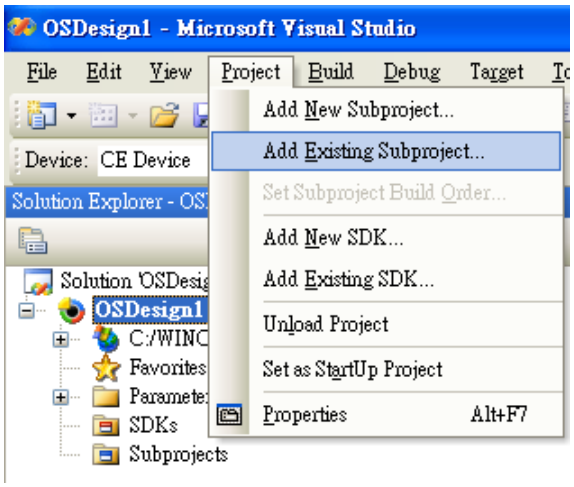

Windows Embedded CE 6.0 OS Design Wizard
Applications & Media
Select items for applications and media to include in your OS design.
.NET Compact Framework 2.0
File Systems and Data Store
Windows Embedded CE Error Reporting
ActiveSync
DCOM
Internet Browser
Lightweight Directory Access Protocol (LDAP
VBScript support
Windows Media Audio/MP3
Windows Media Video/MPEG-4 Video
Windows Messenger
WordPad
Support for applications and services designed for the .NET V2.0 Compact Framework.
< Previous
Next >
Finish
Cancel


Windows Embedded CE 6.0 OS Design Wizard
Networking & Communications
Select items for networking and communications to include in your OS design.
TCP/IPv6 Support
Wide Area Network (WAN)
Local Area Network (LAN)
OBEX Server
Personal Area Network (PAN)
Remote Desktop Connection
Security
The Internet standard protocol, version 6.
< Previous
Next >
Finish
Cancel

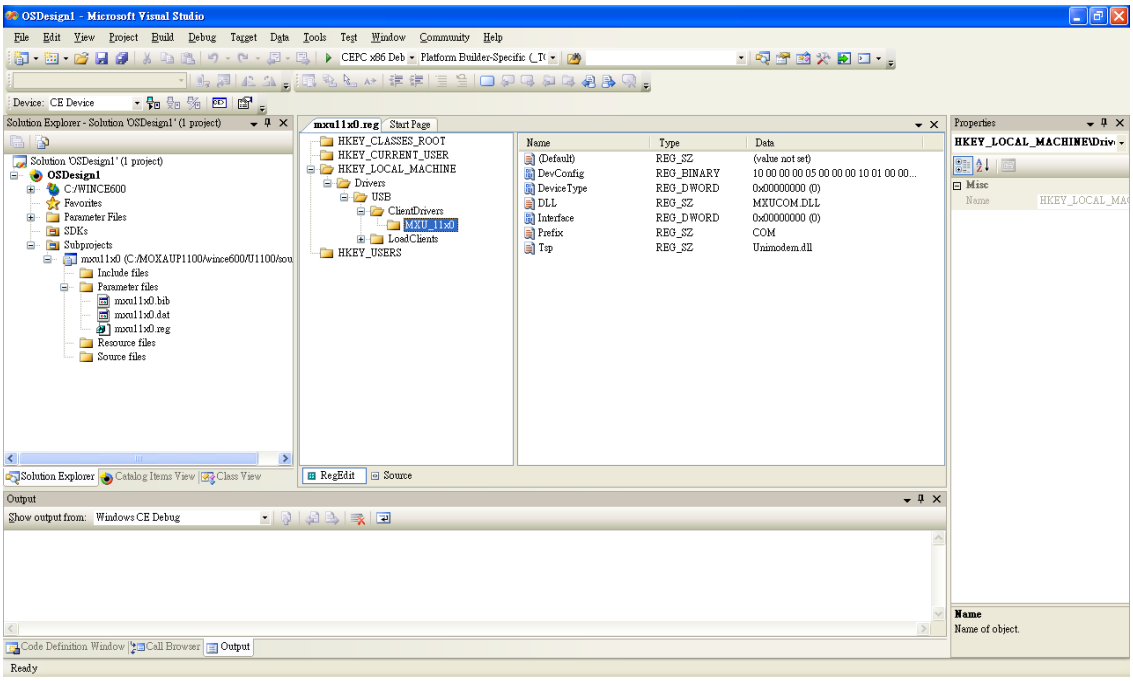
4. When the **OS Design Project Wizard Complete** screen appears, click **Finish**. A notification will pop up. Click "**Acknowledge**" to finish the project.



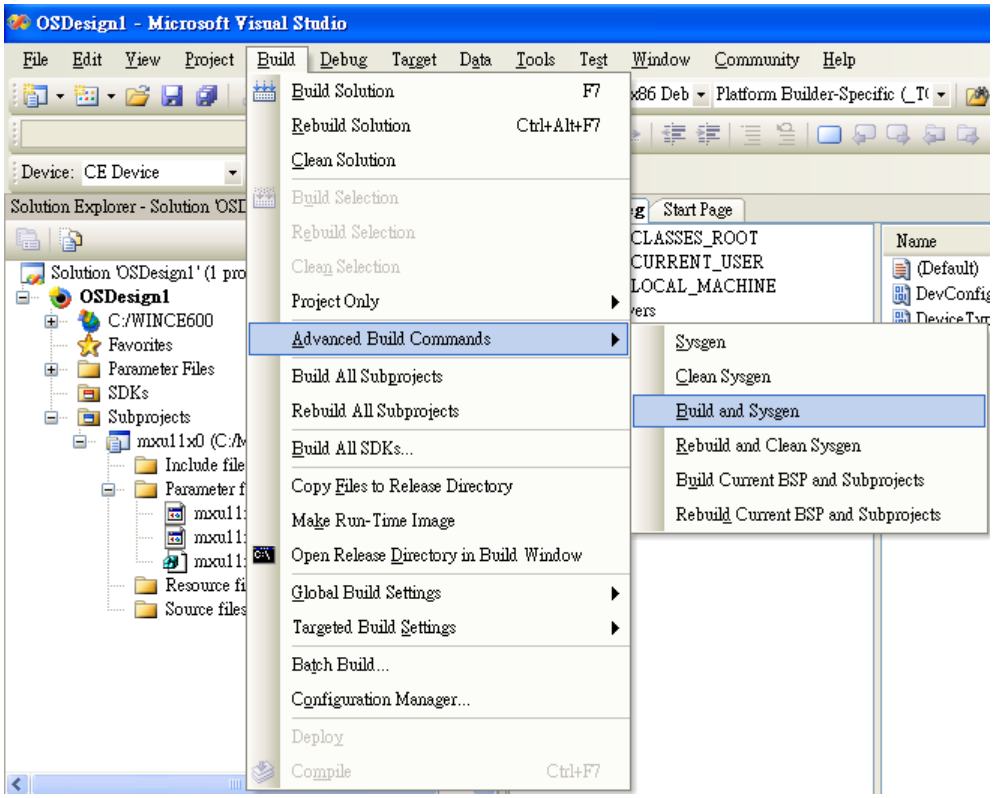
5. Open the project you created. Click **Project** on top of the screen; then, select **Add Existing Subproject**. Switch the folder to "C:\ MOXAUP1100\wince600\U1100" and add subprojects into your OS Design. Assign the pbpxml file to a different folder (i.e. mxu11x0.pbpxml).



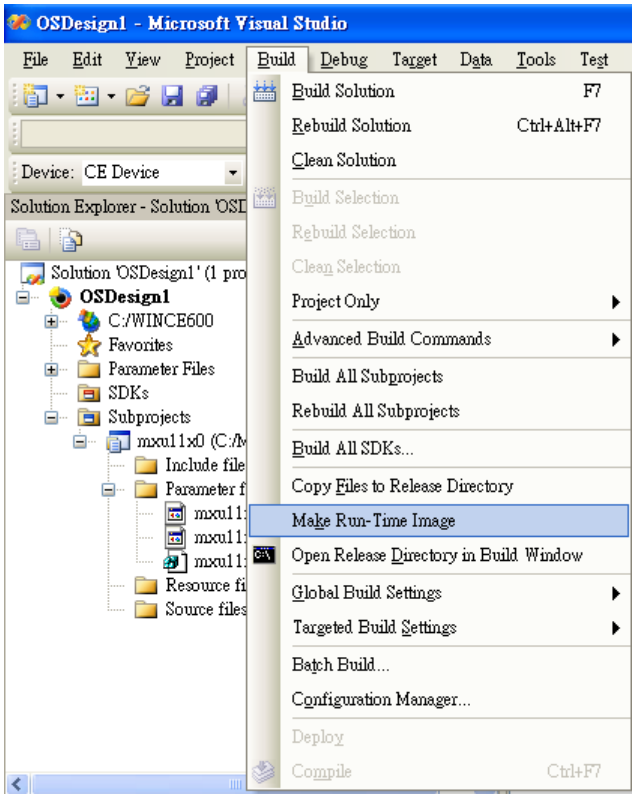
6. After the subproject is added, you may configure the "mxupce6.reg" registry file with the location [HKEY_LOCAL_MACHINE\Drivers\USB\ClientDrivers\MXU_11x0].



7. Open "**Build**" and select "**Advanced Build Commands**" and "**Build Sysgen**". This operation will take you a few minutes.



8. After building sysgen, select **"Build"** and **"Make Run-Time Image"** to create the WinCE OS image. Finally, copy your image file to the target host.

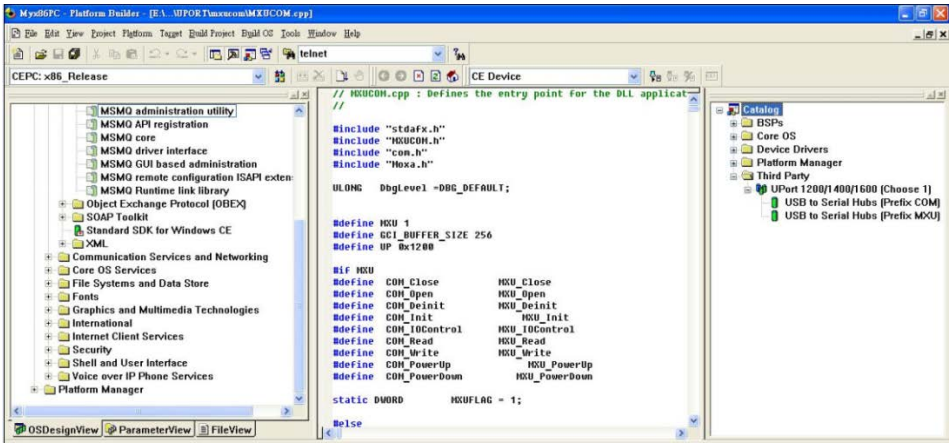


9. Configure the interface on the target device, using the configuration tool found in **"Start > Programs > MOXA UP Configuration Panel\UPort Configuration Utility".**

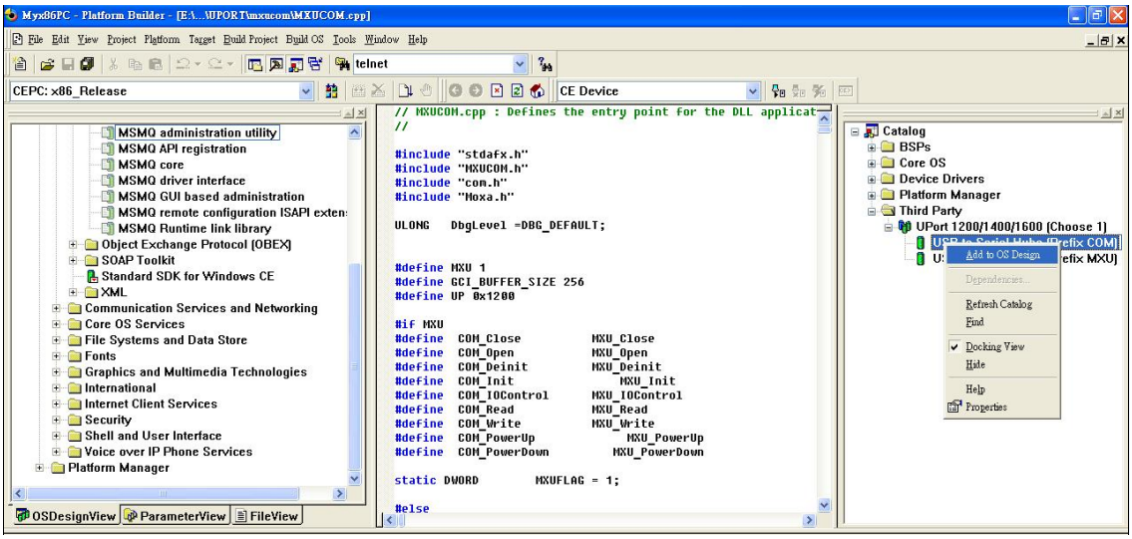
Installation With an Installation Package for Win CE 5.0

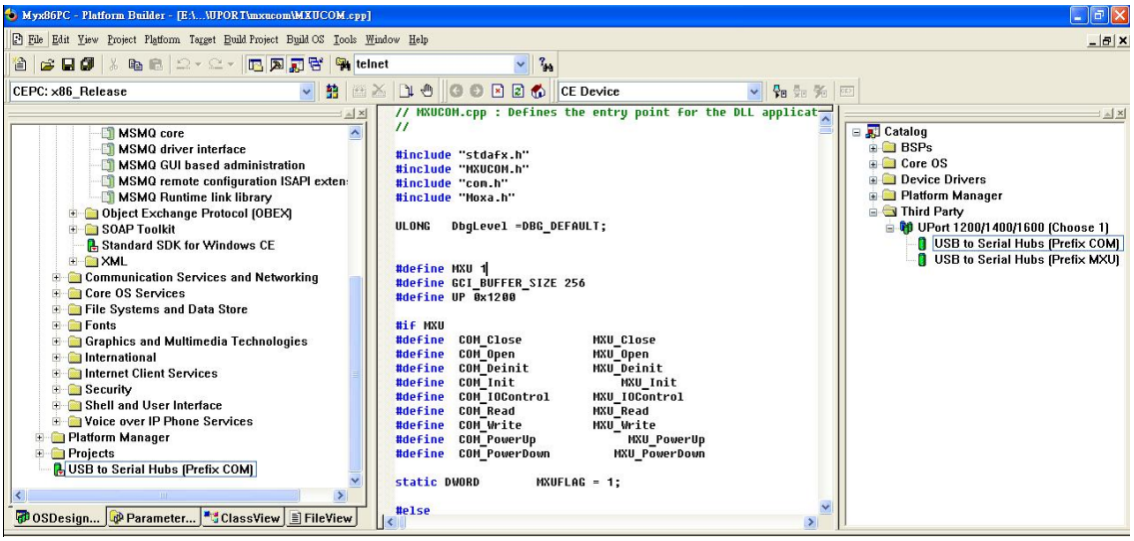
In this section, we show how to install the Moxa WinCE 5.0 driver on a Windows CE 5.0 operating system directly.

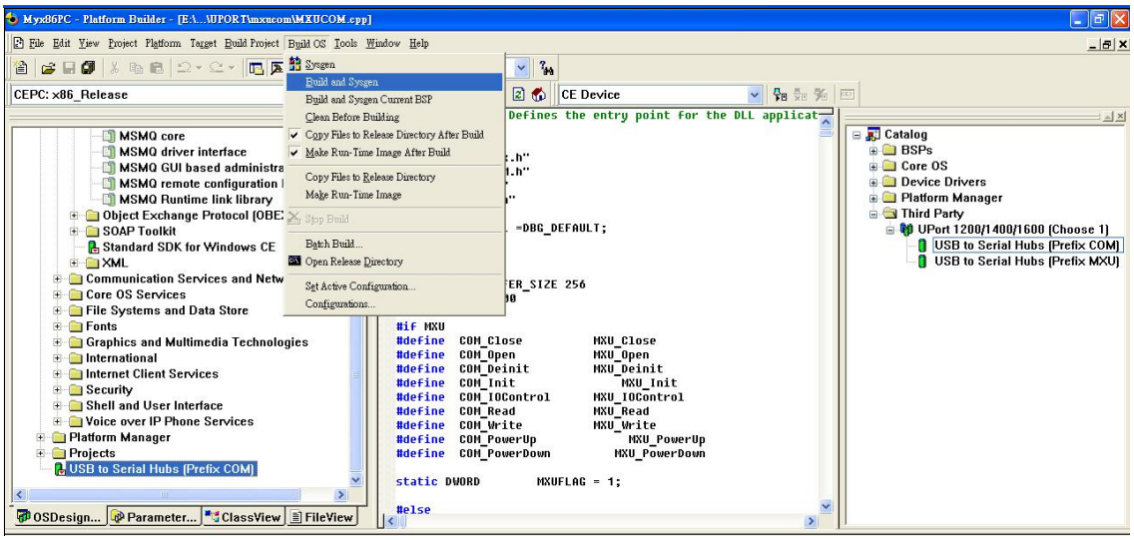
1. Obtain a copy of the UPort 1110/1130/1150 WinCE 5.0 driver package and extract it to your computer. Double-click the installation package to copy the files automatically to the UPort folder **\WINCE500\PLATFORM** (e.g., C:\WINCE500\PLATFORM\U11x0) and import the UPort 1110/1130/1150 WinCE 5.0 driver into the Catalog.
2. Open your workspace in Platform Builder and then open "Manage Catalog Items" under **View (Catalog**. The UPort 1110/1130/1150 WinCE 5.0 driver is in the "Third Party" folder.



3. Right-click on the driver "USB to Serial Adapters (Prefix COM)" or "USB to Serial Adapters (Prefix MXU)" and then choose "Add to OS Design." After adding the driver, you should be able to find it in your workspace. You can start building your operating system and download it to a target.



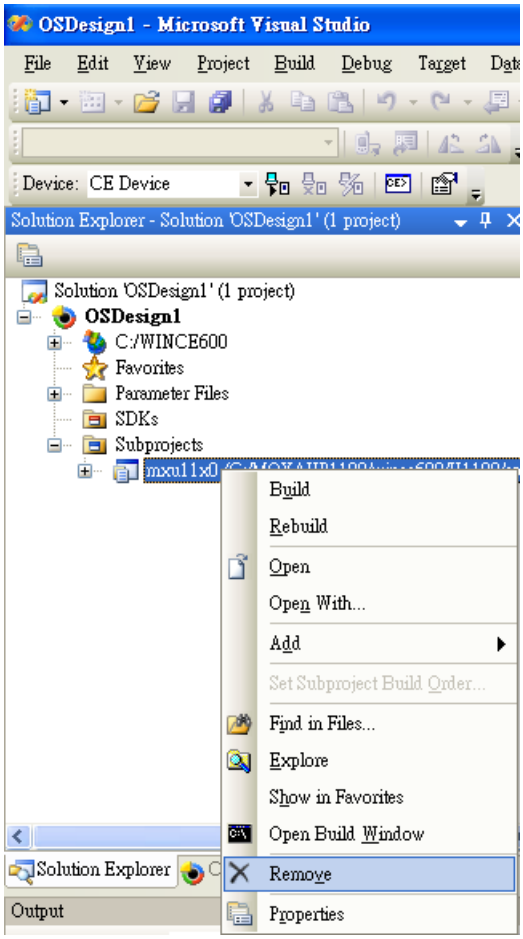

Myx86PC - Platform Builder - [E:\...\UPORT\mxucom\MXUCOM.cpp]
File Edit View Project Platform Target Build Project Build OS Tools Window Help
telnet
CEPC: x86_Release
CE Device
MSMQ core
MSMQ driver interface
MSMQ GUI based administration
MSMQ remote configuration ISAPI exten
MSMQ Runtime link library
Object Exchange Protocol (OBEX)
SOAP Toolkit
Standard SDK for Windows CE
XML
Communication Services and Networking
Core OS Services
File Systems and Data Store
Fonts
Graphics and Multimedia Technologies
International
Internet Client Services
Security
Shell and User Interface
Voice over IP Phone Services
Platform Manager
Projects
USB to Serial Hubs [Prefix COM]
// MXUCOM.cpp : Defines the entry point for the DLL applicat
//
#include "stdafx.h"
#include "MXUCOM.h"
#include "com.h"
#include "Moxa.h"
ULONG DbgLevel =DBG_DEFAULT;
#define MXU 1
#define GCI_BUFFER_SIZE 256
#define UP 0x1200
#if MXU
#define COM_Close MXU_Close
#define COM_Open MXU_Open
#define COM_Deinit MXU_Deinit
#define COM_Init MXU_Init
#define COM_IOControl MXU_IOControl
#define COM_Read MXU_Read
#define COM_Write MXU_Write
#define COM_PowerUp MXU_PowerUp
#define COM_PowerDown MXU_PowerDown
static DWORD MXUFLAG = 1;
#else
Catalog
BSPs
Core OS
Device Drivers
Platform Manager
Third Party
UPort 1200/1400/1600 (Choose 1)
USB to Serial Hubs [Prefix COM]
USB to Serial Hubs [Prefix MXU]
OSDesign...
Parameter...
ClassView
FileView


Myx86PC - Platform Builder - [E:\...\UPORT\mxucom\MXUCOM.cpp]
File Edit View Project Platform Target Build Project Build OS Tools Window Help
Sysgen
Build and Sysgen
Build and Sysgen Current BSP
Clean Before Building
Copy Files to Release Directory After Build
Make Run-Time Image After Build
Copy Files to Release Directory
Make Run-Time Image
Stop Build
Batch Build...
Open Release Directory
Set Active Configuration...
Configurations...
CEPC: x86_Release
CE Device
Defines the entry point for the DLL applicat
=DBG_DEFAULT;
ER_SIZE 256
#if MXU
#define COM_Close MXU_Close
#define COM_Open MXU_Open
#define COM_Deinit MXU_Deinit
#define COM_Init MXU_Init
#define COM_IOControl MXU_IOControl
#define COM_Read MXU_Read
#define COM_Write MXU_Write
#define COM_PowerUp MXU_PowerUp
#define COM_PowerDown MXU_PowerDown
static DWORD MXUFLAG = 1;
#else
Catalog
BSPs
Core OS
Device Drivers
Platform Manager
Third Party
UPort 1200/1400/1600 (Choose 1)
USB to Serial Hubs [Prefix COM]
USB to Serial Hubs [Prefix MXU]
OSDesign...
Parameter...
ClassView
FileView

Removing the Moxa WinCE 5.0/CE 6.0 Driver

For WinCE 6.0 Driver:

1. Select driver to **Remove**.



2. Switch the folder to C:\MOXAUP1100\ and double-click "**unins000.exe**" to remove installation folder.

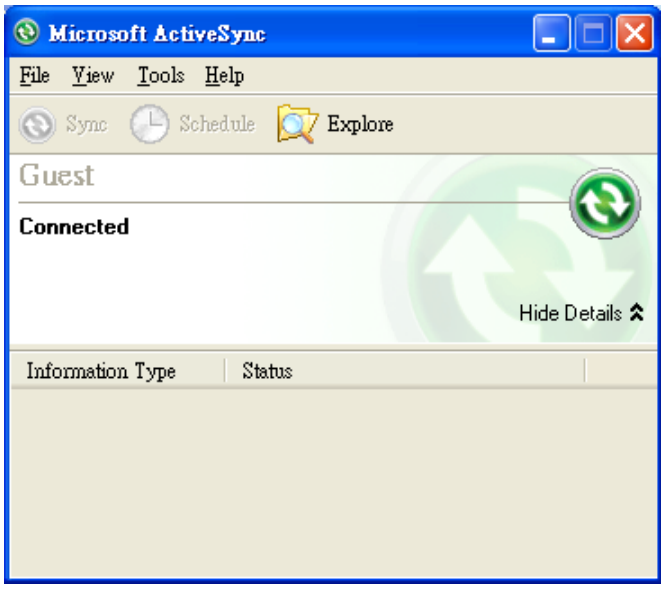
For WinCE 5.0 Driver:

1. In the "OSDesign_View", delete "USB to Serial Adapters (Prefix XXX)".
2. Remove driver from control panel.
3. Check "clean before building"

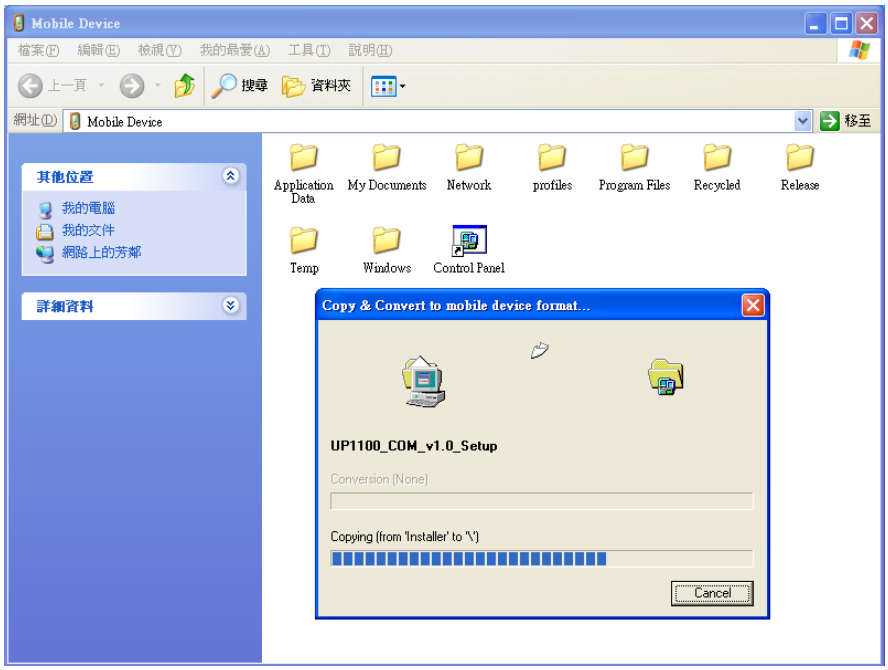
Installation With a CAB File

Install the UPort 1100 series WinCE driver to target host.

1. First, install Microsoft ActiveSync on the source PC. Before running ActiveSync on the source PC, use an RS-232 null modem cable to connect the WinCE device (CEPC) to the PC.
2. Go to the "Command Prompt" on the WinCE device and execute the **repllog** command. A new partnership window will appear. Select "No" and then click on "Next." You will see "**Connected**" in ActiveSync.



3. Click on the "Explore" button in the ActiveSync window and then copy the UP1100_COM_v1.0_Setup.CAB (For WinCE 6.0) or UP1100_COM_v1.1_Setup.CAB (For WinCE 5.0) file to the Mobile/Target Device.

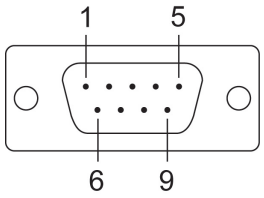


4. Double-click on the CAB file to start installing the UPort 1100 series WinCE driver. After installing the driver, plug the UPort 1100 series device into the USB port of the WinCE Device. The driver will be loaded automatically. At this point, the UPort 1100 series is ready to use.

3. Pin Assignment

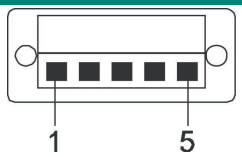
UPort DB9 Pin Assignments

The UPort 1100 series uses male DB9 connectors. Pin assignments are shown in the following diagram:

DB9 (male)	Pin	RS-232	RS-422 4-wire RS-485	2-wire RS-485
	1	DCD (in)	TxD-(A)	–
	2	RxD (in)	TxD+(B)	–
	3	TxD (out)	RxD+(B)	Data+(B)
	4	DTR (out)	RxD-(A)	Data-(A)
	5	GND	GND	GND
	6	DSR (in)	–	–
	7	RTS (out)	–	–
	8	CTS (in)	–	–

Terminal Block Pin Assignments

The UPort 1130/1130I/1150/1150I comes with a DB9 to terminal block converter, with pin assignments as shown below:

Terminal Block	Pin	RS-422 4-wire RS-485	2-wire RS-485
	1	TxD+(B)	–
	2	TxD-(A)	–
	3	RxD+(B)	Data+(B)
	4	RxD-(A)	Data-(A)
	5	GND	GND

NOTE

The converter maps pin 1 on the DB9 connector to pin 2 on the terminal block and pin 2 on the DB9 connector to pin 1 on the terminal block.