Connect Tech Inc.
Industrial Strength Communications

VxWorks
Driver Installation

Document: CTI_SIO_Install
Revision: 1.01
Date: January 4, 2010

VxWorks Driver Installation

1. History

Rev	Author	Date	Changes
0.00	CJD	Aug. 24, 2005	• Original draft.
1.00	CJD	Sept. 8, 2005	• Added installation checklist. • Added 3rd sample BSP. • Fixed minor mistakes.
1.01	RAC	Jan. 4, 2010	• Adapted for both Tornado and Workbench builds.

2. Hardware Requirements

This driver supports the following products:

- Xtreme/104 switchable
- Xtreme/104 232
- BlueStorm/LP
- Xtreme/104-***Plus***

3. Driver Files

Copy the following files into your BSP directory (<WIND_INST>\target\config\<BSP_NAME>):

- 00cti.cdf
- ctiSio.c
- ctiSio.h
- sysCTISerial.c
- usrCTISerial.c

4. Configuration

4.1. BSP Files

The following files should be modified to support the CTI SIO driver. These modifications can be done in any other way as appropriate, but the following are suggestions made based on the Pentium III BSP. For sample modified files, refer to the files (of the same name) located within the driver distribution package. Sample files are from the Pentium III BSP. **Note;** file names and locations of information may differ in your BSP.

- <WIND_INST>\target\config\<BSP_NAME>\config.h
- <WIND_INST>\target\config\<BSP_NAME>\sysLib.c

The sample files included with the driver package should not be copied over any files in your BSP. These sample files are included only so that you can see the required changes in the context of complete files.

4.1.1. config.h

Add a section to your config.h file so that there is a place for the driver options. If your config.h has an "if-def" block used to determine when to define INCLUDE_PCI, it is advised that you place this new section above that "if-def" block. This suggestion is made so that the "if-def" section can be modified to define INCLUDE_PCI when needed by any included CTI boards. The section for CTI driver configuration should surrounded by a check for INCLUDE_CTI_SIO being defined.

For example:

```
/* Connect Tech Inc. serial driver options */
#if defined(INCLUDE_CTI_SIO)
#   define INCLUDE_CTI_U550           /* support for 550 UARTs */
#endif  /* defined(INCLUDE_CTI_SIO) */
```

A complete list of the available configuration options follows:

4.1.1.1. INCLUDE_CTI_U550

Define INCLUDE_CTI_U550 to include support for Xtreme/104 products that utilize 550 UARTs. This can be determined by examining the UART chips on the Xtreme/104 board itself, or by contacting CTI support staff.

Usage:

```
#define INCLUDE_CTI_U550
```

4.1.1.2. INCLUDE_CTI_U650

Define INCLUDE_CTI_U650 to include support for Xtreme/104 products that utilize 650 UARTs. This can be determined by examining the UART chips on the Xtreme/104 board itself, or by contacting CTI support staff.

Usage:

```
#define INCLUDE_CTI_U650
```

4.1.1.3. INCLUDE_CTI_U850

Define INCLUDE_CTI_U850 to include support for Xtreme/104 products that utilize 850 UARTs. This can be determined by examining the UART chips on the Xtreme/104 board itself, or by contacting CTI support staff.

Usage:

```
#define INCLUDE_CTI_U850
```

4.1.1.4. INCLUDE_CTI_EXAR17XX5XPCI

Define INCLUDE_CTI_EXAR17XX5XPCI to include support for BlueStorm/LP and Xtreme/104-***Plus*** products. You should also include you BSP's PCI support if these products are to be used.

Usage:

```
#define INCLUDE_CTI_EXAR17XX5XPCI
```

4.1.1.5. CTI_PCI_MAXBOARDS

CTI_PCI_MAXBOARDS is used to define the maximum number of CTI PCI boards that can be enumerated. If not defined the default is 1.

Usage:

```
#define CTI_PCI_MAXBOARDS  X  /* where X is the maximum number of boards */
```

4.1.1.6. CTI_PCI_MAXCHANS

CTI_PCI_MAXCHANS is used to define the maximum number of serial channels that can be used by enumerated CTI PCI devices. This number is not the maximum per CTI PCI device, it is the maximum total number of channels across all CTI PCI devices. If not defined the default is 8.

Usage:

```
#define CTI_PCI_MAXCHANS  X  /* where X is the maximum number of channels */
```

4.1.1.7. INCLUDE_CTI_AUTO485POLLER

When using half-duplex RS485, the receiver and transmitter need to be toggled on and off depending on whether data is being sent or received. On CTI products that do not utilize UARTs with the capability to do this automatically, the driver must perform the toggling. This can be done in one of two ways; Using the interrupt service routine (ISR) of the channel; Using a watchdog timer as a poller. In order to use the watchdog timer INCLUDE_CTI_AUTO485POLLER must be defined.

Usage:

```
#define INCLUDE_CTI_AUTO485POLLER
```

4.1.1.8. CTI_DEFAULT_BAUD

CTI_DEFAULT_BAUD is used to define the default baud rate when the serial channels are initialized. If not defined the default is 9600.

Usage:

```
#define CTI_DEFAULT_BAUD  X  /* where X is the default baud rate in bps */
```

4.1.1.9. CTI_AUTO485SWITCHBAUD

When using the CTIAUTO485SWITCHED auto 485 direction control method, CTI_AUTO485SWITCHBAUD defines the baud rate used to switch between ISR and poller methods. When the baud rate is less than CTI_AUTO485SWITCHBAUD the ISR method is used, else the poller method is used. If not defined the default is 9600.

Usage:

```
#define CTI_AUTO485SWITCHBAUD  X  /* where X is the baud rate in bps */
```

4.1.1.10. CTI_AUTO485POLLER_DELAY

When the poller auto 485 direction control method is being used a watchdog timer performs a polling operation to look for the end of a transmit. This watchdog timer is set to expire every CTI_AUTO485POLLER_DELAY ticks. If not defined the default is 1.

Usage:

```
#define CTI_AUTO485POLLER_DELAY  X  /* where X is the delay in ticks */
```

4.1.1.11. CTI_SYS_PCI_INTCONNECT()

The CTI_SYS_PCI_INTCONNECT() macro is used to connect a PCI device interrupt to an interrupt service routine. The default definition of this macro uses pciIntConnect().

Usage:

```
/*
* _intLevel_ - interrupt level to connect _isr_ to
* _isr_      - interrupt service routine to connect to _intLevel_
* _arg_      - argument to be passed to _isr_
* _result_   - STATUS result of the macro
*/
#define CTI_SYS_PCI_INTCONNECT(_intLevel_, _isr_, _arg_, _result_) \
{ \
      /* place desired code here */ \
}
```

4.1.1.12. CTI_SYS_INTCONNECT()

The CTI_SYS_INTCONNECT() macro is used to connect a device interrupt to an interrupt service routine. The default definition of this macro uses intConnect().

Usage:

```
/*
* _intLevel_ - interrupt level to connect _isr_ to
* _isr_      - interrupt service routine to connect to _intLevel_
* _arg_      - argument to be passed to _isr_
* _result_   - STATUS result of the macro
*/
#define CTI_SYS_INTCONNECT(_intLevel_, _isr_, _arg_, _result_) \
{ \
       /* place desired code here */ \
}
```

4.1.1.13. CTI_SYS_INTENABLE()

The CTI_SYS_INTENABLE() macro is used to enable an interrupt level. The default definition of this macro uses sysIntEnablePIC().

Usage:

```
/*
* _intLevel_ - interrupt level to enable
* _result_   - STATUS result of the macro
*/
#define CTI_SYS_INTENABLE(_intLevel_, _result_) \
{ \
       /* place desired code here */ \
}
```

4.1.1.14. CTI_SYS_IN_BYTE()

The CTI_SYS_IN_BYTE() macro is used to read an 8 bit value. The default definition of this macro uses sysInByte() for port I/O and does a direct assignment for memory I/O.

Usage:

```
/*
* _ioMode_  - I/O mode (CTIIOPORT or CTIIOMEM)
* _address_ - address to read from
* _value_   - receives 8 bit value read from _address_
*/
#define CTI_SYS_IN_BYTE(_ioMode_, _address_, _value_) \
{ \
       if((_ioMode_) == CTIIOPORT) { \
              /* place port I/O code here */ \
       } \
       else { \
              /* place memory I/O code here */ \
       } \
}
```

4.1.1.15. CTI_SYS_OUT_BYTE()

The CTI_SYS_OUT_BYTE() macro is used to write an 8 bit value. The default definition of this macro uses sysOutByte() for port I/O and does a direct assignment for memory I/O.

Usage:

```
/*
* _ioMode_   - I/O mode (CTIIOPORT or CTIIOMEM)
* _address_  - address to write to
* _value_    - 8 bit value to write to _address_
*/
#define CTI_SYS_OUT_BYTE(_ioMode_, _address_, _value_) \
{ \
       if((_ioMode_) == CTIIOPORT) { \
              /* place port I/O code here */ \
       } \
       else { \
              /* place memory I/O code here */ \
       } \
}
```

4.1.1.16. CTI_SYS_IN_WORD()

The CTI_SYS_IN_WORD() macro is used to read a 16 bit value. The default definition of this macro uses sysInWord() for port I/O and does a direct assignment for memory I/O.

Usage:

```
/*
* _ioMode_   - I/O mode (CTIIOPORT or CTIIOMEM)
* _address_  - address to read from
* _value_    - receives 16 bit value read from _address_
*/
#define CTI_SYS_IN_WORD(_ioMode_, _address_, _value_) \
{ \
       if((_ioMode_) == CTIIOPORT) { \
              /* place port I/O code here */ \
       } \
       else { \
              /* place memory I/O code here */ \
       } \
}
```

4.1.1.17. CTI_SYS_OUT_WORD()

The CTI_SYS_OUT_WORD() macro is used to write a 16 bit value. The default definition of this macro uses sysOutWord() for port I/O and does a direct assignment for memory I/O.

Usage:

```
/*
* _ioMode_   - I/O mode (CTIIOPORT or CTIIOMEM)
* _address_  - address to write to
* _value_    - 16 bit value to write to _address_
*/
#define CTI_SYS_OUT_WORD(_ioMode_, _address_, _value_) \
{ \
       if((_ioMode_) == CTIIOPORT) { \
              /* place port I/O code here */ \
       } \
       else { \
              /* place memory I/O code here */ \
       } \
}
```

4.1.1.18. CTI_SYS_IN_LONG()

The CTI_SYS_IN_LONG() macro is used to read a 32 bit value. The default definition of this macro uses sysInLong() for port I/O and does a direct assignment for memory I/O.

Usage:

```
/*
* _ioMode_  - I/O mode (CTIIOPORT or CTIIOMEM)
* _address_ - address to read from
* _value_   - receives 32 bit value read from _address_
*/
#define CTI_SYS_IN_BYTE(_ioMode_, _address_, _value_) \
{ \
       if((_ioMode_) == CTIIOPORT) { \
              /* place port I/O code here */ \
       } \
       else { \
              /* place memory I/O code here */ \
       } \
}
```

4.1.1.19. CTI_SYS_OUT_LONG()

The CTI_SYS_OUT_LONG() macro is used to write a 32 bit value. The default definition of this macro uses sysOutLong() for port I/O and does a direct assignment for memory I/O.

Usage:

```
/*
* _ioMode_  - I/O mode (CTIIOPORT or CTIIOMEM)
* _address_ - address to write to
* _value_   - 32 bit value to write to _address_
*/
#define CTI_SYS_OUT_LONG(_ioMode_, _address_, _value_) \
{ \
       if((_ioMode_) == CTIIOPORT) { \
              /* place port I/O code here */ \
       } \
       else { \
              /* place memory I/O code here */ \
       } \
}
```

4.1.2. sysLib.c

Add a section to your sysLib.c file that includes the file sysCTISerial.c. This file includes the CTI SIO driver and defines routines used to configure the CTI serial channels. This section should come after the PCI configuration files have been included and should be surrounded by a check for INCLUDE_CTI_SIO being defined.

For example:

```
#if defined(INCLUDE_CTI_SIO)
#   include "sysCTISerial.c"
#endif  /* defined(INCLUDE_CTI_SIO) */
```

In addition to including the driver and driver configuration routines, the routines sysHwInit() and sysHwInit2() need to be modified to call the serial configuration routines defined in sysCTISerial.c.

4.1.2.1. sysHwInit()

During sysHwInit(), sysCTISerialHwInit() should be called to initially configure the CTI serial channels. This call should be made after any PCI initialization in order for CTI PCI adapters to be configured.

For example:

```
#if defined(INCLUDE_CTI_SIO)
    sysCTISerialHwInit();
#endif  /* defined(INCLUDE_CTI_SIO) */
```

4.1.2.2. sysHwInit2()

During sysHwInit2(), sysCTISerialHwInit2() should be called to connect and enable interrupts used by the CTI serial channels.

For example:

```
#if defined(INCLUDE_CTI_SIO)
    sysCTISerialHwInit2();
#endif  /* defined(INCLUDE_CTI_SIO) */
```

4.2. Driver Files

The file sysCTISerial.c, which is included into sysLib.c, needs to be configured to properly match the setup of your system.

4.2.1. sysCTISerial.c

sysCTISerial.c contains rountines used to configure any CTI serial adapters you have. The configuration includes:

- storage for board and channel structures
- board/channel configuration and initialization
- interrupt connecting and enabling
- channel enumeration

The code in this section is only meant to serve as a short example. For a more complete sample refer to the sample BSP's provided with the driver files. For a description of routine parameters refer to the documentation for that routine.

4.2.1.1. Storage

Each board and channel must have a corresponding structure allocated for it. This is typically done by declaring local variables in the file sysCTISerial.c.

Typically one board structure (CTI_BOARD) is defined for each non-PCI CTI adapter.

For example:

```
LOCAL CTI_BOARD  ctiXt1Board;
LOCAL CTI_BOARD  ctiXt2Board;
```

For each of the non-PCI adapters, an array of channel structures should be defined as well. The size of these arrays depends on the number of channels supported by the adapters.

For example:

```
LOCAL CTI_CHAN  ctiXt1Chans[8];
LOCAL CTI_CHAN  ctiXt2Chans[4];
```

For PCI adapters, only one instance of CTI_PCI_DEVICESET need be defined. The maximum number of supported adapters and channels should be defined with CTI_PCI_MAXBOARDS and CTI_PCI_MAXCHANS (see section 4.1.1 - config.h for more information on CTI_PCI_MAXBOARDS and CTI_PCI_MAXCHANS).

For example:

```
#if defined(INCLUDE_PCI)
LOCAL CTI_PCI_DEVICESET  ctiPciDevset;
#endif  /* defined(INCLUDE_PCI) */
```

4.2.1.2. Configuration

During sysHwInit() sysCTISerialHwInit() should be called. This is where the board and channels structures declared in the previous section are configured and initialized.

Each non-PCI board structure should be configured using ctiBoardConfig() and initialized using ctiBoardHrdInit().

PCI adapters should be enumerated and initialized using ctiPciConfig() and ctiPciHrdInit().

For example:

```
void sysCTISerialHwInit(
       void
       )
{
       if(OK == ctiBoardConfig(
              &ctiXt1Board,
              CTIU650,
              CTIIOPORT,
              0x300,
              0x340,  /* 0 if status port not being used */
              0x05,
              7372800,
              4,  /* 1 if using extended baud rates */
              ctiXt1Chans,
              8,
              NULL,
              NULL,
              0xFF))
       {
              if(sysBp) {
                     ctiBoardHrdInit(&ctiXt1Board);
              }
       }

       if(OK == ctiBoardConfig(
              &ctiXt2Board,
              CTIU850,
              CTIIOPORT,
              0x200,
              0x240,  /* 0 if status port not being used */
              0x0A,
              7372800,
              4,  /* 1 if using extended baud rates */
              ctiXt2Chans,
              4,
              NULL,
              NULL,
              0xF))
       {
              if(sysBp) {
                     ctiBoardHrdInit(&ctiXt2Board);
              }
       }

#if defined(INCLUDE_PCI)
       if(OK == ctiPciConfig(&ctiPciDevset, NULL, NULL)) {
              if(sysBp) {
                     ctiPciHrdInit(&ctiPciDevset);
              }
       }
#endif  /* defined(INCLUDE_PCI) */
}
```

4.2.1.3. Interrupts

sysCTISerialHwInit2() should be called during sysHwInit2() and is responsible for connecting and enabling interrupts used by the CTI adapters.

The operations of connecting an interrupt to an interrupt service routine as well as enabling the interrupt have been encapsulated into driver rountines ctiBoardIntConnect() and ctiPciIntConnect(). The behaviour of these routines is controlled by the macros CTI_SYS_INTCONNECT(), CTI_SYS_PCI_INTCONNECT(), and CTI_SYS_INTENABLE(). These macros can be redefined from their default behaviour. See section 4.1.1 - config.h for

more information on CTI_SYS_INTCONNECT(), CTI_SYS_PCI_INTCONNECT(), and CTI_SYS_INTENABLE(). These routines do not have to be used and in some cases may not be sufficient. The specifics of their operation can be seen in ctiSio.c.

For example:

```
void sysCTISerialHwInit2(
       void
       )
{
       ctiBoardIntConnect(&ctiXt1Board);

       ctiBoardIntConnect(&ctiXt2Board);

#if defined(INCLUDE_PCI)
       ctiPciIntConnect(&ctiPciDevset);
#endif  /* defined(INCLUDE_PCI) */
}
```

4.2.1.4. Enumeration

Each serial channel must be associated with a device. The CTI serial channels are enumerated using sysCTISerialChanGet(). This routine takes an index value specifying which channel structure to retrieve, and effectively acts as a look-up table for the channels. The order in which the channels are returned determines the order in which the channels are associated with device names.

For example:

```
SIO_CHAN* sysCTISerialChanGet(
       int  channel  /* serial channel */
       )
{
       if((channel >= 0) && (channel < 8)) {
              return((SIO_CHAN*)&ctiXt1Chans[channel]);
       }
       else if((channel >= 8) && (channel < (8 + 4))) {
              return((SIO_CHAN*)&ctiXt2Chans[channel - 8]);
       }
#if defined(INCLUDE_PCI)
       else if((channel >= (8 + 4)) &&
              (channel < (8 + 4 + ctiPciDevset.cChans)))
       {
              return((SIO_CHAN*)
                     &(ctiPciDevset.pChans[channel - 8 - 4]));
       }
#endif  /* defined(INCLUDE_PCI) */

       return((SIO_CHAN*)ERROR);
}
```

The total number of channels needs to be reported using sysCTISerialGetNumChans(). This routine simply needs to return the total number of channels being used across all CTI adapters.

For example:

```
UINT16 sysCTISerialGetNumChans(
       void
       )
{
       UINT16  retVal = 0;

       retVal += 8;

       retVal += 4;

#if defined(INCLUDE_PCI)
       retVal += ctiPciDevset.cChans;
#endif  /* defined(INCLUDE_PCI) */

       return(retVal);
}
```

4.3. Tornado/WorkBench Projects

Bootable VxWorks image projects based on a BSP containing the CTI driver files will have a new folder available under **hardware\peripherals\serial** called "Connect Tech Inc.". This folder contains components for controlling the CTI serial driver. A description of the components follows:

4.3.1. Connect Tech Inc. SIO

This component controls INCLUDE_CTI_SIO (used earlier when including the CTI serial driver in the BSP).

4.3.2. Use alternate base name

If this component is not included, the CTI serial channels will be named "/tyCo/*X*" where *X* is the channel index value and is based on the defined NUM_TTY value so that the names do not collide with other serial devices. However, it may be desireable to use a different base name. By including the **Use alternate base name** component, the default behaviour is to name the CTI serial channels "/tyCTI/*X*" where *X* is the channel index value starting at 0. The **Use alternate base name** component has a parameter called CTI_TY_NAME_BASE which can be used to set a base name other than "/tyCTI/".

5. Examples

Included with the driver files are three example BSP's. These example BSP's are based on the Pentium III BSP from WindRiver. A description of each of the example BSP's follows:

5.1. pcP3CTI_1

This example is based on the following CTI serial adapters:

- Xtreme/104 RS-232
  - 8 channels.
  - Base I/O address is 0x300.
  - IRQ is 10.
  - Status port is in use.
  - Extended baud rates not in use.
- BlueStorm/LP RS-232
  - 4 channels.

5.2. pcP3CTI_2

This example is based on the following CTI serial adapters:

- Xtreme/104 1
    - 4 RS-232 channels.
    - Base I/O address is 0x300.
    - IRQ is 10.
    - Status port is in use.
    - Extended baud rates not in use.
- Xtreme/104 2
    - 8 channels.
    - Base I/O address is 0x200.
    - Channels 1, 3, 5, and 7 are on IRQ 5, channels 2, 4, 6, and 8 are on IRQ 9. See Mode 2 in the *Interrupt Selection* section of the Xtreme/104 manual.
    - Status port is not in use.
    - Extended baud rates in use.
    - Channels 1 and 2 are RS-232, channels 3, 4, and 5 are RS-485 full-duplex, channels 6, 7, and 8 are RS-485 half-duplex.
- Xtreme/104-***Plus*** Switchable
    - 4 channels.
    - Channels 1 and 2 are RS-485 full-duplex, channels 3 and 4 are RS-232.

5.3. pcP3CTI_3

This example is based on the following CTI serial adapters:

- BlueStorm/LP
    - 8 RS-232 channels.

6. Installation Checklist

☑	Task
	Copy driver files into BSP directory
	Modify **config.h** to include desired support
	Modify **sysLib.c** to include **sysCTISerial.c**
	Modify *sysHwInit()* to call *sysCTISerialHwInit()*
	Modify *sysHwInit2()* to call *sysCTISerialHwInit2()*
	Modify **sysCTISerial.c** to include required storage
	Modify *sysCTISerialHwInit()* to configure adapters
	Modify *sysCTISerialHwInit2()* to connect interrupts
	Modify *sysCTISerialChanGet()* to return requested channel structures
	Modify *sysCTISerialGetNumChans()* to return total number of channels
	Include the **Connect Tech Inc. SIO** component into Tornado/Workbench projects