Ser8UART

Serial NT / W2000 Device Driver for SC28L198 Based ISA Cards

Title	**:** Ser8UART
Subject	**:** Serial NT / W2000 Device Driver for SC28L198 Based ISA Cards
Author	**:** Ghijselinck Christiaan
Reference	**:** Ser8UART.doc
Version	**:** SW.DRIVER.SER8UART.01.01.0002.0001
Date	**:** 4/1/2002 5:26 PM
Pages	**: 39**

Table of contents

1. Introduction

The Ser8UART Serial IO Card Device Driver has been developed to support the Philips SC28L198 eight-port UART chip for both the Windows NT40 and Windows 2000 operating systems. Although, it is only functionally tested on a dedicated "MultiPort-IO-Card", it should support all generic designs for the SC28L198 using the ISA Bus interface.

This driver maintains full compatibility with the existing 16C550 UART family Windows drivers concerning behavior and published interfaces. Win32 Applications that use serial communication through the existing Win32 API layer, will fully profit from its 16C550 simulation. Applications will be able to access the SC28L198 UART without any changes, as if it was a 16C550 UART.

This documentation is restricted to the description of the particular behavior that may deviate from the core "Serial" driver provided and documented within the WinNT40 and Win2000 OS's. If not mentioned explicitly, existing interfaces and techniques as described by the Win32 API Microsoft description, are to be used to access the driver from within applications. To save redundancy, a description of these 16C550 compatible interfaces to the existing Serial Drivers will only be provided if they are different from the common behavior. These interface descriptions and Serial Communication treatment in general, can be retrieved from the "Microsoft Developer Network",
"Microsoft Platform SDK" and "Microsoft DDK for NT40 and Win2000". See References [4] Chapter "\Platform SDK Documentation\Base Services\Hardware\Communication Resources" and Reference [5] Chapter "\Kernel Mode Drivers".

Programming guidelines and details about the PHILIPS SC28L198 UART can be retrieved from the SC28L198 Datasheet. See Reference [2].

This device driver also supports the 16C550 UART in both single and multi port configuration, configurable through particular registry entries. The description of the 16C550 support is however not treated here.

2. Feature Description (Summary)

Summarized Overview:

- Support for 8 simultaneously driven SC28L198 UART ports
- Separate and independent programmable control of the RTS/CTS/DSR/RTS signal lines as input or output. If no HW handshaking is used, the Serial Communication port can be used simultaneously and independent together with these control input/output pins.
- Works both on NT40 and W2000
- Runs in parallel with the core Serial Communication Driver (“Serial.sys”) on NT40 and W2000.
- Support for the new WIN2000 "IOCTL_SERIAL_SET_FIFO_CONTROL" call

Restrictions:

- The SC28L198 UART does not support the Carrier Detect (CD) and Ring Indicator (RI) signals. The driver supports however the interfaces to these control signals (set and reset) for compatibility reasons, and does not reject calls that would manipulate these handshake lines (*). Both software and hardware act on these signal lines as "not wired" through. Query calls will thus always return a "low" state for these signal lines.

  (*) Blocking the treatment to these lines would disable to run some applications, even if they don't make usage of these lines.

- PNP / WMI /HID and "Power Management" are not supported (Windows 2000 features only)

- There are no specialized installation software and configuration control applications (control panel applet) provided. Entering a few uncomplicated registry entries with one PC reboot performs the installation. Configuring the driver is a three step action :
  - ✓ "stopping" the driver,
  - ✓ changing registry entries,
  - ✓ "restarting" the driver again.

- Some 16C550 baud rates are not supported by the SC28L198. See the detailed description.

- Some RxFIFO Trigger levels are forced to a lower (sharper) trigger level to be compatible with programs that would rely on 16C550 programmable trigger levels. A sharper trigger level will occasionally cause more ISR calls to be executed, but will prevent the RxFIFO to run over. This behavior is generally transparent to the application. See further.

- The “always zero” and “always one” parity bit modes provided by the 16C550 family UARTs are not supported by the SC28L198

- The Ser8UART driver supports only ONE (1) Ser8UART ISA card to be plugged on a particular system. If more cards would be used, it should be possible to install renamed device driver files with own registry entries. Resource checks are however performed, which will prevent redundant usage.

- The Ser8UART supports maximally 8 UART channels at the same time (accordingly the amount of channels provided on the SC28L198), also when configured for 16C550 compliant devices. The core “Serial” device driver provided by the OS supports theoretically an infinite number of 16C550 compliant UARTS.

Integration:

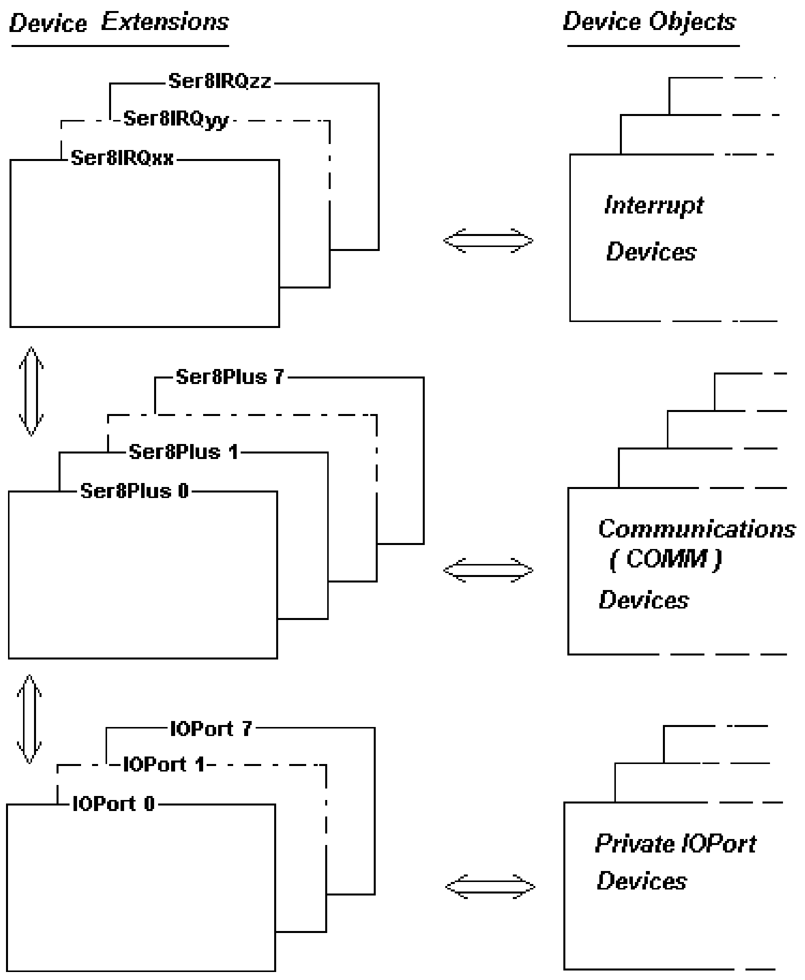
The driver will run and service the SC28L198 in parallel with the standard Serial Port driver provided as base component of the OS. Since the "Ser8UART" driver will be started automatically within Start Group 3 of the WINNT40/WIN2000 driver hives, symbolic names for devices that are serviced by this core Serial Driver will already exist. This implies that the Ser8UART driver can enumerate through this device names and check for redundant Port Names (called "Symbolic Names" further on) provided in the Ser8UART typical Parameters registry entries. The Ser8UART will not install devices that were already created by the regular Serial Driver [1], even if this device with the particular name has been disabled.

[1]

Within WIN2000, the names are generated automatically by the PnP Serial Enumeration Driver after "Serial" has revealed its existence, not by the NT-Kernel Hardware detection software as on NT40. The devices are always named as "COMx" where "x" stands for a port number. Due to a flaw within this software, particular "COMx" port names are hard to be used again, once this port has been removed. The "new hardware" detection software will create a new "COMy" device name where (y > x) leaving a hole in the numbering sequence. One should be aware of this fact when defining symbolic device names. The Ser8UART driver does not rely on this PnP enumeration scheme, which allows the user to define names i.e. "SERx" that will never get in conflict with the "COMx" automatic naming. See also Chapter " Registry & Configuration " for more details.

3. Detailed Description

3.1 Schematic Overview



At driver initialization, the OS allocates for each "Device Object" that the driver creates, a "Driver Extension" memory block. This is a work area where the driver stores dynamic and static software and hardware information. The Ser8UART driver creates an amount of "Device Objects" for the Communication (COMM) Devices" (see picture) accordingly the amount of channels stored at the "Parameters" hive of the drivers registry entry.

These Device Objects and corresponding Extension are created only on following conditions:

- the device is defined in the registry within a particular "Parameters" registry key hive **(1)**
- the device name does not collate with existing serial device names
- the device has not been disabled by means of the "DisablePort" key value within the drivers "Parameters" key hive **(1).**

The "Private IOPort Device" is created on following conditions:

- the creation of its associated "Communication Device" succeeded.
- the "Parameters" key hive of the associated "Communication Device" contains valid values for the key entries "PrivateIOPort" and "PrivateIOPortName"

To keep track of the card interrupts, the Ser8UART creates an "Interrupt Device" for each different hardware interrupt that is used by the serviced UARTS. This will likely be only one when servicing the SC28L198 chip, but could be more, even one for each UART, when servicing the 16C550 UARTS.

The device extensions are mutually linked as follows:

- All "Device Objects" are forward and backward linked to their respective "Extension".
- Each "Private Port Extension" is forward and backward linked to its respective "Communication Device" extension.
- An "Interrupt Device" is forwarding linked to each "Communication Device" extension if it has to service the IRQ for this particular device.
- From the "Communication Device" extension, their exists a link back to the "Interrupt Device" extension that services the associated UART channel.
- Their exist NO direct forward and backward links from the "Interrupt Device" extension to the "Private Port Extension".

These linkages are built during the driver initialization (driver startup) and remain unchanged during operation.

The main goal is to assure fast data access within each
"Communication Device" - "Interrupt Device" - "IOPort Device" trilogy. Hereby, the "Communication Device" extension is used centralized within every action performed with the UART. Even in case of servicing a "Private IOPort", the "Communication Device" extension will be addressed, since this the only extension that contains the actual hardware and software states.

Note that the OS links all "Device Objects" with each other. This order is used when unloading the driver, since the driver itself does not link the "Communication Device" extensions within each other.

(1) See chapter on "Registry and Configuration"

3.2 SC28L198 specific behavior

3.2.1 FIFO Treatment

3.2.1.1 RxFIFO Trigger Level

To guarantee that the existing Windows applications will be able to work with the SC28L198, the interfaces to the Serial Communication related Win32 API calls are made fully compatible with the 16C550 as far as the SC28L198 can provide these interfaces. Since the 16C550 and SC28L198 Tx/Rx FIFO trigger levels are different, the driver converts the 16C550 specific values into SC28L198 trigger levels in a way to ensure that the applications will not notice any difference. In general, the 16C550 trigger levels will be converted into the next lower value (more sensitive). This will generally cause a slight more dynamic ISR overhead with exception for the trigger levels 1 and 8. These levels are provided by both UART's. See next table.

16C550 Programmable trigger level	Resulting SC28L198 trigger level (*)
1 byte	1 byte
4 byte	Not available
8 byte	8 byte
14 byte	12 byte
(*) SC28L198 - 16 byte fifo trigger level not used	

The RxFIFO trigger level is dynamically changed during the receive of characters depending on the amount of characters the user asks to read. The RxFIFO trigger level will however always been set to one (thus generating an interrupt when one character enters the FIFO) in following cases :

1. Upon initialization [equivalent to the "CreateFile (\\.\ComportName ...)" call] of a particular port.

2. If the device has been "Opened" by an application, but there exists no outstanding read (equivalent to "ReadFile"). Since the driver may not know how many characters have to come in before the application starts its read, all characters are collected one by one by the driver into its own ISR buffer (8 Kbyte) until the application starts a read.

3. When the application asks for fewer characters than the current RxFIFO level.
   In this case, the RxFIFO trigger level is decreased to one character trigger level dynamically. All characters that may already have been stored in the RxFIFO remain available in the FIFO after changing the RxFIFO trigger level. However, changing the interrupt level will not initiate an interrupt to be generated until a new character arrives. To overcome this problem, the SC28L198 is programmed to run its "Watch dog" timer who is especially conceived for this purpose. The watch-dog timer is inhibited again once this timer expires (= 64 receiver clock counts). This timer is automatically reset when a character arrives or the RxFIFO is read. If the timer expires, a RECEIVER interrupt is generated by the SC28L198. Upon this event, the driver explicitly disables the timer before the first character is read from the FIFO. Every time the application initiates a read of more characters than the initial RxFIFO trigger level stored in the registry, or the actual level configured by the application, the RxFIFO trigger level will dynamically be restored to this initial value.

4. In case XON/XOFF has been programmed.

3.2.1.2 TxFIFO Trigger Level

The TxFIFO trigger level treatment by this driver is identical to the existing UART 16C550 compatible Serial NT40/Win2000 drivers treatment. The driver programs the TxFIFO trigger level always to "1", thus generating an interrupt when the TxFIFO becomes empty.

The SC28L198 requires however a specialized treatment that deviates from the typical 16C550 UART control sequence due to its nature of treating and arbitrating interrupts.

The SC28L198 prioritizes the TxFIFO interrupt against the RxFIFO interrupts within its arbitration logic. As long as transmissions are in progress (i.e. characters are leaving the FIFO causing the FIFO to empty) Tx interrupts will override pending RxFIFO interrupts. This will likely cause Rx overruns, since these Rx interrupts get not serviced. From experience, it was noticed furthermore that, when the Rx-FIFO trigger level is reached, an Rx-interrupt is not initiated during an enabled TxFIFO transmit. To avoid Rx overflow, the TxFIFO is only enabled during the transfer of the characters into this FIFO, and disabled immediately again when all characters were written, even if the TxFIFO has not been flushed yet. The SC28L198 supports this behavior (see Reference [2]). If more characters are to be sent out, the TxFIFO interrupt is enabled again after a particular time calculated by the driver:

```
Time_Interval = Amount_of_Written_Characters * Character_Time * 80/100
```

If the TxFIFO interrupt is enabled again exactly after this timeout, the interrupt itself will occur later on when the TxFIFO gets empty. If however, due to other time consuming activities (i.e. treatment of received characters), all characters have been sent out during this time interval, the TxFIFO empty interrupt will occur immediately after it has been enabled again. In both cases, remaining characters will be written into the TxFIFO which will restart a new cycle. Working this way, guarantees that the SC28L198 remains blocked only during small time-slices, giving the opportunity to the RxFIFO interrupts to come in. At the same time, this guarantees continuous non-bursty output streaming.

Note that the "amount of characters" adapts according the "TxFIFO" registry key entry from the "Parameters" hive, and the amount of presented characters to write out, i.e. the amount of characters that are sent within one cycle, is always less or equal the value stored in the registry at "TxFIFO". If fewer characters are asked to write, this particular amount of characters represents one cycle. In any case, the maximum amount of characters that ever will be sent out within one cycle will never exceed 16 (= FIFO depth). This behavior corresponds with the behavior of the core 16C550 Serial Driver provided by the OS.

See "Configuring the Ser8UART" driver for details about the registry entries concerning TxFIFO levels for the Ser8UART device driver.

3.2.2 Baud Rate Treatment

As with the FIFO treatment, the SC28L198 defines deviant Baud Rate values from the 16C550 family of UART's. To be compatible with the existing Win32 API Interfaces for Serial Communications, this driver will only allow programming Baud Rates that are supported by both the Operating System AND the SC28L198. See next table. The supported Baud Rates are marked as **BOLD**.

Baudrate	**Supported by 16C550 / W2000 / NT**	**Supported by SC28L198**
BAUD_050	No	Yes
BAUD_075	Yes	Yes
BAUD_110	Yes	No
BAUD_134_5	Yes	No
BAUD_150	Yes	Yes
BAUD_200	No	Yes
BAUD_300	Yes	Yes
BAUD_450	No	Yes
BAUD_600	Yes	Yes
BAUD_900	No	Yes
BAUD_1200	Yes	Yes
BAUD_1800	Yes	Yes
BAUD_2400	Yes	Yes
BAUD_3600	No	Yes
BAUD_4800	Yes	Yes
BAUD_7200	Yes	Yes
BAUD_9600	Yes	Yes
BAUD_14400	Yes	Yes
BAUD_19200	Yes	Yes
BAUD_28800	No	Yes
BAUD_38400	Yes	Yes
BAUD_56K	Yes	No
BAUD_57600	Yes	Yes
BAUD_115200	Yes	Yes
BAUD_128000	Yes	No
BAUD_230400	No	Yes

3.2.3 Parity Mode and Stop Bits

The 16C550 Family UARTS allow to use an “always zero” or “always one” parity flag (respectively called MARK_PARITY and SPACE_PARITY). These types of parity treatment are not supported by the SC28L198. Applications will get an error code returned when trying to program these parity modes since no SC28L198 substitution can be provided. On the contrary, the SC28L198 supports to select the polarity of the parity bit. This extension is unknown to the Win32 API and through this, not supported.

The SC28L198 supports a “9/16 character time” Stop Bit Length. The driver does support this neither for compatibility reasons.

3.3 "Private IOPort" treatment

For special dedicated usage, the driver supports to create named "Devices" that enable control of the handshaking lines (CTS, DSR, RTS and DTR) as general purpose input and/or output pins. Creating one symbolic device for each SC28L198 channel can access these devices on per UART controller channel base. Windows Applications may access and control these handshaking lines through specialized IOCTL control calls (API call "DeviceIoControl") to program [*], read and write the lines.

These Symbolic Devices can be used in conjunction with the SC28L198 "Serial Communication Device" at the same time and for the same UART channel. Management and control of these pins run independent from the remaining Rx and Tx pins of the same UART channel and does not interfere. This implies that "Private IOPorts" can be opened/closed not dependably from opening/closing the "Serial Device" on the same UART channel. Note that the usage of "Private IOPorts" for a particular channel is determined at Driver Initialization by means of registry settings and cannot be changed dynamically (see also [*]).

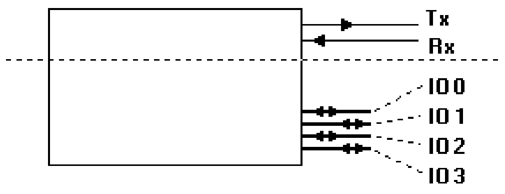
In case the IOPort device has been defined to access the handshaking lines of the particular channel, all access to these lines from the Serial Communication Device that services the same UART channel, will be rejected. Serial Communication still remains full functional when not using these handshaking signals.

[*] Reprogramming specialized handshaking lines will be irrelevant if the usage of these ports is "hard wired" not bi-directional. In general, the configuration of these lines as either input or output pin though registry settings, fully depends on how they are accessed by the hardware.

HW configuration example:

A dedicated SC28L198 ISA card could hard-wire the IOPorts of the first four controllers (channels) for dedicated usage (Picture 1). This corresponds to the "UART-A" through "UART-D" as specified in "Reference [2]". The IO ports of the last four channels, mentioned as "UART-E" through "UART-H", are wired could be wired on the card as typical RTS/CTS/DTR/RTS handshaking lines. This corresponds with the normal usage of these lines as specified (Picture 2) (*)

IO0 = CTS (input line)
IO1 = DSR (input line)
IO2 = DTR (output line)
IO3 = RTS (output line)



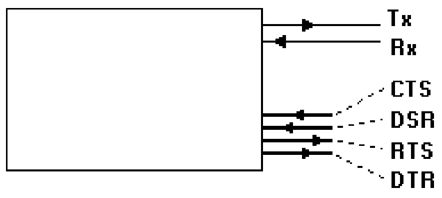


Figure 1 (UART A-D) Figure 2. (UART E-H)

(*) DTR and RTS are not wired conform SC28L198 specification

3.3.1 Private IOPort access by the Win 32 API.

Before the driver can be accessed, its "Device" has to be "created". The device open call is common for almost all Windows device drivers and described in detail in the MSDN "Microsoft Developer Network" (See reference [4]). Each "Private IOPort" device must have its own "Symbolic Device Name" defined in the registry. The code is listed here for clarification. The returned handle has to be provided within the calls to "DeviceIOControl".

```
char device[128];
char szDeviceUnicodePart [] = "\\\\.\\";

strcpy ( &device[0] , szDeviceUnicodePart ) ;

// IOPORT  = "Symbolic Device Name" defined in the registry
strcat  ( &device[0] , "IOPORT"            ) ;

hDevice = CreateFile ( &device[0]      ,
                        GENERIC_READ    | GENERIC_WRITE,
                        FILE_SHARE_READ | FILE_SHARE_WRITE,
                        NULL,
                        OPEN_EXISTING,
                        FILE_ATTRIBUTE_NORMAL,
                        NULL ) ;
```

"Device" access is closed by means of the " CloseHandle (hDevice) " function

Similar to making "Open" calls to the UART's COM channels, an IOPort "Open" call is exclusive i.e. sharing the same "IOPort" device among multiple applications or making multiple "Open" calls within the same application, is not supported.

Reading and writing from/to the IOPorts is to be performed through the Win32 API call "DeviceIOControl" after the Device has been opened by the application. The driver "exports" three dedicated functions that can be accessed using next sample code :

```
#ifndef CTL_CODE
#define CTL_CODE(DeviceType, Function, Method, Access) \
        (((DeviceType) << 16) | \
         ((Access)     << 14) | \
         ((Function)    << 2) | \
         ((Method)          ))
#endif

#define IO_BUFFER_SIZE 0x0100

#define FILE_DEVICE_SERIAL_PORT  0x0000001b

#ifndef FILE_ANY_ACCESS
#define FILE_ANY_ACCESS   0
#define FILE_READ_ACCESS  1
#define FILE_WRITE_ACCESS 2
#endif

#define METHOD_BUFFERED   0
#define METHOD_IN_DIRECT  1
#define METHOD_OUT_DIRECT 2
#define METHOD_NEITHER    3


#define IOCTL_SC28L198_CONFIG_IOPORTS \
CTL_CODE(FILE_DEVICE_SERIAL_PORT,100,METHOD_BUFFERED,FILE_ANY_ACCESS)
#define IOCTL_SC28L198_READ_IOPORTS \
CTL_CODE(FILE_DEVICE_SERIAL_PORT,101,METHOD_BUFFERED,FILE_ANY_ACCESS)
#define IOCTL_SC28L198_WRITE_IOPORT \
CTL_CODE(FILE_DEVICE_SERIAL_PORT,102,METHOD_BUFFERED,FILE_ANY_ACCESS)
```

The IOCTL_SC28L198_READ_IOPORTS call returns in a DWORD sized variable, the statutes of ALL four handshaking lines. The statuses of both the "input" ports as well as "output" ports are returned in the lowest nibble (4 bit) of the returned DWORD. The position of the bit corresponds to the IO port as described in the SC28L198 specification [2] :

Bit 2exp0 = IO0
Bit 2exp1 = IO1
Bit 2exp2 = IO2
Bit 2exp3 = IO3

A "one" corresponds to a "high" output (positive voltage) ; a "zero" corresponds to a "low" voltage (negative voltage).

```
// -----------------------------------------------------------------
BOOL ReadPorts  (    HANDLE      hDevice,
                     DWORD      *pStatuses )
{
    DWORD dBytesReturned;
    return DeviceIoControl ( hDevice                            ,
                             (DWORD)IOCTL_SC28L198_READ_IOPORTS ,
                             NULL               ,
                             0                  ,
                             pStatuses          ,
                             sizeof (DWORD)     ,
                             &dBytesReturned    ,
                             NULL               ) ;
}
```

The IOCTL_SC28L198_WRITE_IOPORT enables to control the IO ports that are programmed as "output". The DWORD value that has to be provided to the call, must be composed out of the "IOPort number" (i.e. IO0, IO1, IO2 or IO3) and the value that corresponds to a "high" (positive voltage) or "low" (negative voltage) thus a "one" or a "zero". The IOPort number must be stored in the lowest byte of the DWORD as a decimal value from 0 to 3 :

IO0 = 0
IO1 = 1
IO2 = 2
IO3 = 3

The "value" to set (0 or 1) must be stored in the second lowest byte of the DWORD.

```
// -------------------------------------------------------------
BOOL WritePort     ( HANDLE hDevice  ,
                     DWORD  dwPort   ,
                     DWORD  dwValue  )
{
    DWORD dBytesReturned  ;
    DWORD dwData          ;

    if ( dwValue ) dwValue = 1 ;

    dwData = dwPort | ( dwValue << 8  );

    return DeviceIoControl ( hDevice               ,
                             (DWORD)IOCTL_SC28L198_WRITE_IOPORT  ,
                             &dwData               ,
                             sizeof(DWORD)         ,
                             NULL                  ,
                             0                     ,
                             &dBytesReturned       ,
                             NULL                  ) ;
}
```

This IOCTL_SC28L198_CONFIG_IOPORTS function is provided "as is", it should not be used if the Ser8UART hardware card does not support bi-directional ports.
To reprogram the IO Ports, the user has to "set" and "clear" the bits within the lowest nibble of the DWORD input value. A "one" will configure the port as "Output"; a "zero" will configure the port as "Input".
Every new CONFIG_IOPORTS call overrides a previous configuration.

```
// -------------------------------------------------------------

BOOL ProgramPorts ( HANDLE hDevice        ,
                    DWORD  dwOutputPorts  )
{
   DWORD dBytesReturned;

   return DeviceIoControl ( hDevice                             ,
                           (DWORD)IOCTL_SC28L198_CONFIG_IOPORTS ,
                           &dwOutputPorts                       ,
                           sizeof(DWORD)                        ,
                           NULL                                 ,
                           0                                    ,
                           &dBytesReturned                      ,
                           NULL                                 );
}
```

User programs may wait on IOPort input changes within a dedicated thread. For this purpose, the driver creates an "Event" with a particular name stored in the registry for the corresponding device, or if no name was provided, composes the event name from the "IOPort Symbolic Name" suffixed with the word "Event" (see next sample). If the event is fired from the driver, the user thread has to reset manually the event and eventually, execute an IOCTL_SC28L198_READ_IOPORTS call to get the updated statuses of the input ports.

```
/*this sample assumes that the event name has been automatically created from
the IOPort name appending the "Event" string. When acting like this, the event
name must not be stored in the drivers registry parameters hive for a
particular "Ser8UART*", thus "PrivateIOPortEvent" = "A Name"  may NOT be
provided, or must be equal to “IOPORTEvent” when using this sample */

char   device[128]      ;
ULONG  dwResult         ;
char   eventname[128]   ;

// IOPORT  = "Symbolic Device Name" defined in the registry
strcpy  ( &eventname[0] , "IOPORT"         ) ;
strcat  ( &eventname[0] , "Event"          ) ;

/* try to open the event with the given name ( if exists ) */
hEvent = OpenEvent ( SYNCHRONIZE , FALSE , &eventname[0] ) ;

while ( TRUE )
{
   DWORD dwPortStatuses ;

   __try { dwResult = WaitForSingleObject ( hEvent , 5000 ) ; }
   __except ( dwResult = WAIT_FAILED ) { dwResult = WAIT_FAILED ; }

   if ( dwResult == WAIT_TIMEOUT  ) continue ;
   if ( dwResult == WAIT_OBJECT_0 )
   {
      ReadPorts ( hDevice , &dwPortStatuses ) ;
      printf ( "IOPort Statuses for %s = %02x\n" ,
               &device[0] , dwPortStatuses ) ;
   }
   else
   {
      printf ( "WaitForSingleObject failed "
               " with result %d ( LastError = %d )\n" ,
               dwResult , GetLastError ( ) ) ;
      break ;
   }
}
CloseHandle ( hEvent  ) ;
CloseHandle ( hDevice ) ;
```

4. Registry & Configuration

Drivers in general are solely propagated to the OS through the registry. Further on, it is a common usage that driver specific configuration settings are also stored in the registry and mostly within the same hive as the propagating entries. On may distinguish in general, three data collections:

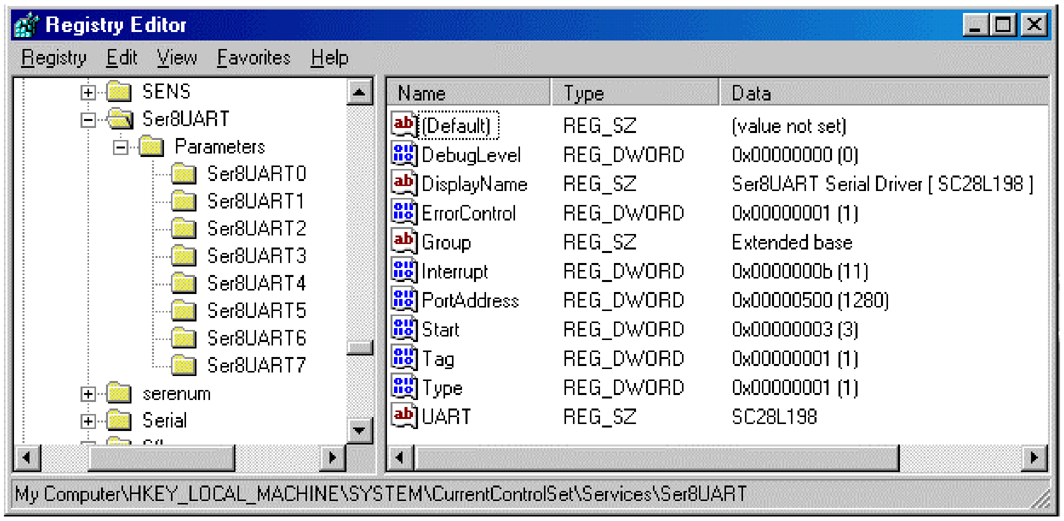
- ✓ The System Hive: this hive propagates the driver to the OS on reboot. This hive must contain particular entries through which the OS can classify, load and start the device driver. This hive will mostly also contain Driver Specific entries that are not interpreted by the OS.
- ✓ The Parameters hive. This hive is exclusively Device Driver specific. There exist only some commonly accepted rules when defining their content.
- ✓ The "Event Logging" registration entry. Device Drivers that create logging events have to register themselves to the Event Logging System. This is simply performed by specifying some additional entries in the "Parameters" hive of the "EventLog" driver.

Next pictures and tables bring an overview of the SC28L198 typical settings. Registry entries for usage with 16C550 compatible UART's are not treated.

Next two pictures show a typical example of the "Ser8UART" System Hive and parameters hive.

System Hive:

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART]

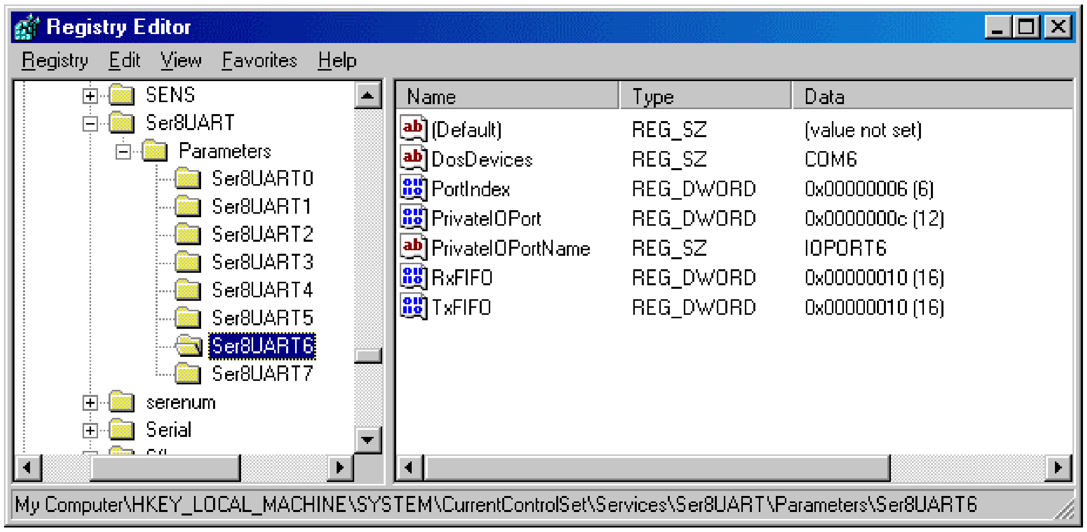


Name	Type	Data
(Default)	REG_SZ	(value not set)
DebugLevel	REG_DWORD	0x00000000 (0)
DisplayName	REG_SZ	Ser8UART Serial Driver [SC28L198]
ErrorControl	REG_DWORD	0x00000001 (1)
Group	REG_SZ	Extended base
Interrupt	REG_DWORD	0x0000000b (11)
PortAddress	REG_DWORD	0x00000500 (1280)
Start	REG_DWORD	0x00000003 (3)
Tag	REG_DWORD	0x00000001 (1)
Type	REG_DWORD	0x00000001 (1)
UART	REG_SZ	SC28L198

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART

Parameters Hive:

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters]
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters\Ser8UART6]



Name	Type	Data
(Default)	REG_SZ	(value not set)
DosDevices	REG_SZ	COM6
PortIndex	REG_DWORD	0x00000006 (6)
PrivateIOPort	REG_DWORD	0x0000000c (12)
PrivateIOPortName	REG_SZ	IOPORT6
RxFIFO	REG_DWORD	0x00000010 (16)
TxFIFO	REG_DWORD	0x00000010 (16)

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters\Ser8UART6

Following table lists all possible and relevant registry entries specific for SC28L198 usage. A detailed description follows thereafter.

Driver Event Logging Registration

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog\System\Ser8UART]	
Name	**Value Required by OS**
EventMessageFile	%SystemRoot%\\System32\\IoLogMsg.dll;%SystemRoot%\\System32\\Drivers\\Ser8UART.sys
TypesSupported	dword:00000007

Driver Registry System Hive

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART]		
Name	**Value [default]**	**Obligatory / Required by OS**
Type	dword:00000001	Yes / Yes
Start	dword:00000002	Yes / Yes
Group	"Extended base"	Yes / Yes
Tag	dword:00000001	Yes / Yes
ErrorControl	dword:00000001	Yes / Yes
DebugLevel	dword:00000000	No
DisplayName	"Ser8UART Serial Driver [SC28L198]"	Yes / Yes
PortAddress	dword:00000500	Yes / No
Interrupt	dword:0000000b	Yes / No
UART	"SC28L198"	No

Driver Registry Parameters Hive

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters]		
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters\Ser8UART0]		
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters\Ser8UART1]		
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters\Ser8UART2]		
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters\Ser8UART3]		
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters\Ser8UART4]		
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters\Ser8UART5]		
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters\Ser8UART6]		
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters\Ser8UART7]		
Name	**Value [default]**	**Obligatory**
DosDevices	"????"	Yes
RxFIFO	RxFIFO=dword:00000008	No
TxFIFO	TxFIFO=dword:00000001	No
PortIndex	dword:0000000x [x >= 0 ; x <=7]	Yes
PrivateIOPort	dword:0000000F	No
PrivateIOPortName	"??????"	Yes [*]
PrivateIOPortEvent	"??????"	No
DisablePort	dword:00000000	No
	[*] only required if "PrivateIOPort" is provided	

Driver Event Logging Registration Hive:

The Driver Event Logging Registry are fixed as far as the driver resides into the "%SystemRoot%\System32\Drivers" directory

EventMessageFile	%SystemRoot%\\System32\\IoLogMsg.dll;%SystemRoot%\\System32\\Drivers\\Ser8UART.sys
TypesSupported	dword:00000007

Driver Registry System Hive:

Type	dword:00000001	This value is required by the OS, it should never be changed
Start	dword:00000003	This value determines the start group of the driver : Value = 2 : The driver is started automatically within the group of non-system drivers. This is the normal start up value used during operation. The driver should never been started within the "System" start group, since the legacy "Serial" driver from this system group must always be started first. Value = 3 : This value must be entered during installation of the driver. It means that the driver will be started manually. Note : changing the value from 3 to 2 can be performed by means of the "Device Manager"
Group	"Extended base"	This value is required by the OS, it should never be changed
Tag	dword:00000001	This value is required by the OS, it should never be changed
ErrorControl	dword:00000001	This value is required by the OS, it should never be changed
DebugLevel	dword:00000000	This value can be used within the DEBUG enabled version of the driver. Each bit in the DWORD value can be used to switch on/off certain debug information
DisplayName	"Ser8UART Serial Driver [SC28L198]"	This string value is required by the OS, it may be changed before the driver is installed

PortAddress	dword:00000500	The Ser8UART card determines this value. Since the IO mapping of the Ser8UART card can not be changed, this value should never change neither
Interrupt	dword:0000000b	The Ser8UART card determines this value by means of its jumper settings. Possible values are 10 (0x0A) 11 (0x0B) 12 (0x0C) and 15 (0x0F). It is recommended to use value 11 (0x0B).
UART	"SC28L198"	This entry and this string value can be omitted. It determines if the driver will service 16C550 compliant UARTs or the SC28L198 card. The "SC28L198" is the default string.

Driver Registry Parameters Hive:

Each of the eight controllers (= UART channels) may contain a key entry into the parameters key hive. Each of these key entries is per definition named from "Ser8UART0" through "Ser8UART7". Other names are allowed. Note that not all 8 channels must be defined, if a particular controller on the UART (= channel) will not be used, its key entry may be ommitted. The association to the particular SC28L198 channel is purely made on the key value "PortIndex" within a "Ser8UART$_x$" key entry (see further).

DosDevices	"????"	This string value represents the "Symbolic Link Name" through which an application accesses the device **[1]**. This string value **must** be provided.
RxFIFO	RxFIFO=dword:00000008	This key value may be omitted, in which case a default value of eight will be programmed, conform the existing default values used by the standard "Serial" device. See also **[2]**
TxFIFO	TxFIFO=dword:00000001	See "RxFIFO". See also **[2]**
PortIndex	dword:0000000x	**Obligatory** : see **[3]**
PrivateIOPort	dword:0000000F	This entry may only be provided if the handshaking lines are used as dedicated IOPorts. **[4]**.
PrivateIOPortName	"??????"	Obligatory if the entry "PrivateIOPort" is provided, otherwise, this parameter is meaningless **[4].**
PrivateIOPortEvent	"??????"	See **[4]**
DisablePort	dword:00000000	If set to "1", the according device will not be created and will thus not be accessible. The device can be made active again after the value has been reset to "0" AND the device driver has been "stopped" and "restarted"

[1]

This string value represents the name through which the application opens the device with the "CreateFile" Win32 API call (see MSDN description on accessing devices and Serial ports in particular). Names provided here will usually have the form "COMx" (i.e. "COM5" , "COM6", etc..). It is important that the names are unique through the system. In this particular case, the names provided here must also be different from the symbolic names that have been selected by the regular standard "Serial" device driver in conjunction with the "Serenum" driver (Serenum is only relevant on Win2000 OS), even if these devices are disabled. If this not the case, the device will not be created by the "Ser8UART" driver. Other names can however be used, for example "SER1". It is commonly accepted that symbolic names restrict to eight signs (alphabetic characters and numbers).

[2]

Although, the standard values can be used, it is recommend to programm full Tx and Rx FIFO support:

TxFIFO = 16
RxFIFO = 14

See also " SC28L198 specific behavior- RxFIFO trigger level " for details. Note that programming RxFIFO = 14 will revert to programming the SC28L198 trigger level to the 3/4 full water mark.

[3]

The "PortIndex" reflects the used hardware channel on the SC28L198 Philips UART :

PortIndex	UART channel
PortIndex = 0	UARTA
PortIndex = 1	UARTB
PortIndex = 2	UARTC
PortIndex = 3	UARTD
PortIndex = 4	UARTE

PortIndex = 5
PortIndex = 6
	UARTF
	UARTG

PortIndex = 7	UARTH

The relationship is "hard wired" i.e. it can not been changed.

[4]

The " PrivateIOPort " key value is to be provided, if the IO pins of the UART channel will be used as private ports accessible through another device name than the one provided in "DosDevices". The provided value reflects bitwise the IO ports that will be used as OUTPUT pins.

Examples :

"PrivateIOPort" = dword:00000000	→ all IO pins (IO0 through IO3) will be used as input pins
"PrivateIOPort" = dword:0000000F	→ all IO pins (IO0 through IO3) will be used as output pins
"PrivateIOPort" = dword:0000000A	→ IO0 and IO2 are input pins; IO1 and IO3 are output pins

See also chapter "Private IOPort treatment" and the sample code

If this " PrivateIOPort " entry is provided, the " PrivateIOPortName " string parameter MUST be provided too. The name stored at this value will be used within the "CreateFile (....)" win32 API call to access these IO ports.

The " PrivateIOPortEvent " string value will be used by the driver to create a Named Event through which it fires an event when IO input port status changes occur.
If the parameter is omitted, the driver creates its own name out of the " PrivateIOPortName " by adding suffix "Event" ;

Example: "PrivateIOPortName"="IOPORT6" → "PrivateIOPortName"="IOPORT6Event"

5. User Guidelines & Installation

Once installed, the "Ser8UART" driver can be accessed in an identical way as the classic "Serial" devices enumerated by the OS and controlled by the core "Serial" driver. Because the driver is non-PnP compatible, it will not be visible on Win2000 within the ports hive of the listed devices. To display the properties of the Ser8UART, the view has to be extended with the "Hidden Devices". Within this hive, the drivers can be controlled :

- Start / Stop
- Changing Startup type : "Demand" or "Automatic".

Driver installation

Preparation

Before installing the driver, it is recommended to prepare the registry entries necessary to let it function properly. See Chapter " Registry & Configuration ".

One may use next default values, copy them into a text file with extension "reg" (for example "Ser8UART.reg") and adapt the settings accordingly.

```
REGEDIT4

; Register the Event Log for the Driver

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog\System\Ser8UART]
"EventMessageFile"="%SystemRoot% \\System32\\IoLogMsg.dll;%SystemRoot%\\System32\\Drivers\\Ser8UART.sys"
"TypesSupported"=dword:00000007

; Detailed settings ....

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART]
"Type"=dword:00000001
"Start"=dword:00000003
"Group"="Extended base"
"Tag"=dword:00000001
"ErrorControl"=dword:00000001
"DebugLevel"=dword:00000000
"DisplayName"="Ser8UART Serial Driver [ SC28L198 ]"
"PortAddress"=dword:00000500
"Interrupt"=dword:0000000b
"UART"="SC28L198"

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters]

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters\Ser8UART0]
"DosDevices"="PORT0"
"RxFIFO"=dword:00000008
"TxFIFO"=dword:00000001
"PortIndex"=dword:00000000

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters\Ser8UART1]
"DosDevices"="PORT1"
"RxFIFO"=dword:00000008
"TxFIFO"=dword:00000001
```

```
"PortIndex"=dword:00000001

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters\Ser8UART2]
"DosDevices"="PORT2"
"RxFIFO"=dword:00000008
"TxFIFO"=dword:00000001
"PortIndex"=dword:00000002

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters\Ser8UART3]
"DosDevices"="PORT3"
"RxFIFO"=dword:00000008
"TxFIFO"=dword:00000001
"PortIndex"=dword:00000003

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters\Ser8UART4]
"DosDevices"="COM4"
"RxFIFO"=dword:00000008
"TxFIFO"=dword:00000001
"PortIndex"=dword:00000004

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters\Ser8UART5]
"DosDevices"="COM5"
"RxFIFO"=dword:00000008
"TxFIFO"=dword:00000001
"PortIndex"=dword:00000005

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters\Ser8UART6]
"DosDevices"="COM6"
"RxFIFO"=dword:00000010
"TxFIFO"=dword:00000010
"PortIndex"=dword:00000006
"#PrivateIOPort"=dword:0000000c
"PrivateIOPortName"="IOPORT6"

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Ser8UART\Parameters\Ser8UART7]
"DosDevices"="COM7"
"RxFIFO"=dword:00000008
"TxFIFO"=dword:00000001
"PortIndex"=dword:00000007
```

Installation

Installation is performed within next well defined steps ;

1. Copy the "**Ser8UART.sys**" into the "**%Systemroot%\Sytem32\Drivers**" directory.

2. Register the driver by double-clicking on the "**Ser8UART.reg**". Assure that the driver may not be started automatically (Registry entry : "Start"=dword:00000003)

3. Reboot the PC.
   The reboot will allow the OS to enumerate the driver and register it to the "Event Logging" system.

4. Start now the driver :
   On Win2000, this is performed through the "Device Manager". :
   - Select "View" → "Show Hidden Devices"
   - Expand the hive "Non-Plug and Play Drivers" and search for the "Ser8UART Serial Driver [SC28L198]"
   - Select "Properties" of this driver
   - Select the "Driver" map within this "Properties" window
   - Select the value "Automatic" as "Type" within the "Startup" selection box
   - Push the "Start" button within the "Current Status" selection box

The driver should now be ready for use.

Troubleshooting

Error messages from the driver can be shown via the "Event Viewer" (System Log)

6. Test Description

The table below represents a “quick check list” flow for hardware and software tests of both the Ser8UART card and driver. This list is only provided to test the SC28C198 configuration of the Ser8UART driver. Executing this test list assumes basic knowledge of using the WIN2000 OS and especially on how to “Start” and “Stop” device drivers and on how to manipulate the registry.
Base knowledge of the Ser8UART card (its hardware features and possibilities) and some basic knowledge about serial communication in general, may also be required.

This table is also available as separate document for reproduction (See Reference [3]). These tests make usage of two test executables:

- **Ser8IOTst.exe**
- **Ser8COMM.exe**

and a "**Ser8COMM.INI**' file. The content of a master copy of the INI file is provided herein after. This "master copy" can be used to create the necessary INI files, needed during the test.

No	***Title***	***Description***	***Remarks***
1	Preparations		
		Prepare and configure two PC's with at least one Ser8UART card on one side	
		Get familiar with configuring the Serial Ports (SC28L198 and 16C550 if used) through the registry. Get familiar with the "Ser8COMM.INI" from the test package (executables and files)	
		Connect both PC's through a full DTE-DTE serial cable	
		Configure the ports (registry settings) with TxFIFO = 16 , RxFIFO = = 1	
		Select the port that will be used among the four (4) last ports of the Ser8UART card	
		Copy the INI file into the %SystemRoot% directory (i.e. C:\WINNT) of both PC's.	
		Provide a readable text file (for example the accompagning "Whatsnew.txt") containing about 10000 characters and copy this file into a directory of your choice on both PC's.	
		Provide the full path name of this text file in the INI file field entry "FILE"	
		Copy the help executables "Ser8IOTst.exe" and "Ser8COMM.exe" into a directory of your choice	
		Open a "DOS Box" on both PC's and change the directory to the directory where these help executables reside	
No	***Title***	***Description***	***Remarks***
2	Normal Flow		
21	Configuration		
		Configure this Ser8UART port to NOT using a "PrivateIOPort" in the registry (rename or delete the entry)	
		Set "TESTRUN=1" in the INI files on both PC's	

		Configure one PC as "Server" (*) by means of INI setting MODE="Server". This PC should contain the "Ser8UART" card	
		Configure the other PC as "Requester" by means of INI setting MODE="Requester"	
22	Test Execution		
		Configure INI on "server" side : baudrate = 9600 , 8 , n , 1 , n	
		Configure INI on "requester" side : baudrate = 9600 , 8 , n , 1 , n	
		Start first the "server" Ser8COMM.exe, start then "requester.exe"	
		Press "N" for the question "continue reading" if all bytes where received on the "Server" side from the requester side	
		Check the amount of bytes sent back from the "Server" side to "Requester" side with the original amount of sent bytes.	
		Eventually, read the content	
23	Test Execution (continued)		
		Repeat step 2.2 for the following port settings by adapting the INI file on both sides :	
		baudrate = 9600 , 8 , n , 1 , x	
		baudrate = 9600 , 8 , n , 1 , d	
		baudrate = 19200 , 8 , n , 1 , n	

		baudrate = 19200 , 8 , n , 1 , x
		baudrate = 19200 , 8 , n , 1 , d	

		baudrate = 57600 , 8 , n , 1 , n	
		baudrate = 57600 , 8 , n , 1 , x	
		baudrate = 57600 , 8 , n , 1 , d	
		baudrate = 115200 , 8 , n , 1 , n	
		baudrate = 115200 , 8 , n , 1 , x	
		baudrate = 115200 , 8 , n , 1 , d	
No	***Title***	***Description***	***Remarks***
3	Special Test Flow : XOFF/XON case		
31	Configuration		
		Configure this Ser8UART port to NOT using a "PrivateIOPort" in the registry (rename or delete the entry) if necessary	
		Set "TESTRUN=2" in the INI files on both PC's	
		Configure one PC as "Server" by means of INI setting MODE="Server"	
		Configure the other PC conatining the "Ser8UART" card as "Requester" by means of the INI setting MODE="Requester"	
		Repeat test steps 22 and 23 above for this TESTRUN case	
4	Special Test Flow : RTS/CTS case		
41	Configuration		
		Configure this Ser8UART port to NOT using a "PrivateIOPort" in the registry (rename or delete the entry) if necessary	

		Configure the PC containing the "Ser8UART" card as "Server" by means of the INI setting MODE="Server"	
42	Test Execution		
		Repeat test steps 22 and 23 above for this TESTRUN case	
No	***Title***	***Description***	***Remarks***
5 **51**	Break Detection Configuration		
		Configure this Ser8UART port to NOT using a "PrivateIOPort" in the registry (rename or delete the entry) if necessary	
		Set "TESTRUN=1" in the INI files on both PC's	
		Configure the PC containing "Ser8UART" card as "Server" by means of INI setting MODE="Server"	
		Configure the INI file on the server side : baudrate = 9600 , 8 , n , 1 , n	
52	Test Break detection	Configure TERM95.exe on the other PC to use the same port settings : baudrate = 9600 , 8 , n , 1 , n	
		Send a break to the SC28L198 by means of this "Term95.exe"	
		Expected response : --> error [80720002] (break detected) on "server" side	
		Execute the test in the opposite direction : configure the INI file on the opposite site as “server” and start “Term95.exe” on the PC containing the SC28L198. Send a break.	
		Expected response : --> error [80720002] (break detected) on the opposite "server" side	
No	***Title***	***Description***	***Remarks***
6	Communication Error Cases		
61	Configuration		
		Configure this Ser8UART port to NOT using a "PrivateIOPort" in the registry (rename or delete the entry) if necessary	
		Set "TESTRUN=1" in the INI files on both PC's	
		Configure the PC containing the "Ser8UART" card as "Server" by means of the INI setting MODE="Server"	
		Configure the other PC as "Requester" by means of the INI setting MODE="Requester"	
62	Test framing error		
		Configure the INI on server side : baudrate = 9600 , 8 , n , 1 , n	
		Configure the INI on requeter side : baudrate = 115200 , 8 , o , 1 , n	
		Start first the "server" Ser8COMM.exe, start then "requester" .exe	
		Expected response : --> error [80720020] (framing error) on the "server" side Configure the INI on server side : baudrate = 115200 , 8 , n , 1 , n	
		Configure the INI on requester side : baudrate = 115200 , 8 , o , 1 , n	

		Start first the "server" Ser8COMM.exe, start then "requester" .exe	
		Expected response : --> error [80720020] (framing error) on "server" side	
63	Test parity error	Configure the INI on server side : baudrate = 115200 , 8 , e , 1 , n	
		Configure the INI on requester side : baudrate = 115200 , 8 , o , 1 , n	
		Start first the "server" Ser8COMM.exe, start then "requester" .exe	
		Expected response : --> error [80720008] (parity error) on the "server" side	
64	Test overrun error		
		Configure the INI on server side : baudrate = 115200 , 8 , n , 1 , n	
		Configure the INI on requester side : baudrate = 115200 , 8 , n , 1 , n	
		Start first the "server" Ser8COMM.exe, start then "requester" .exe	
		Start the "Server"...and let time out (30 seconds) ; do not press "Y" or "N" !	
		Start the "Requester" ... and let transmit characters Press now "Y" on the "Server" running side	
		Expected response : --> error [80720010] (overrun error) on "server" side	

No	***Title***	***Description***	***Remarks***
7	Test Of "Private IOPort"		
71	Configuration		
		Insert a serial "LED box" into the serial connection	
		Configure the used "Ser8UART" port to explicitely usage of the "PrivateIOPort" in the registry and provide it with the value 0x0000000C (decimal = 12) : "PrivateIOPort"=dword:0000000C	
		Provide a name for this private IOPort : "PrivateIOPortName"="IOTEST"	
		"Stop" the Ser8UART driver and "Start" the driver again to detect the registry changes	
		Configure the INI files on both PC's for executing the test step 22 above	
72	Test Execution		
		Open an additional DOS box and start the Test program "Ser8IOTst.exe" as follows : " \>Ser8IOTst.exe IOTEST /run "	
		Check if the DTR and RTS lines toggle alternately on an off	
		Execute test step 22 Watch if the data transfer executes properly and if the "leds" keep toggling during and after the data transfer	

No	***Title***	***Description***	***Remarks***
8	Extended stress test		
81	Configuration		

		Connect the first four (4) serial ports of the Ser8UART through a NULL modem cable : port 1 to port 2 ; port 3 to port 4	
		Connect the last four (4) serial ports of the Ser8UART through a full DTE-DTE NULL cable with each other : port 5 to port 6 ; port 7 to port 8	
		Configure the ports (registry settings) with TxFIFO = 16 , RxFIFO = 1	
		Assure that all ports have a specific "COMx" name and are accessible	
		Copy the help executable "Ser8COMM.exe" into a directory of your choice and open four (4) DOS boxes. Change within each DOS box to the directory where the "Ser8COMM.exe" resides	
		Copy the INI file to eight (8) different named INI files into a directory of your choice	
		Provide a readable text file (for example the accompagning "Whatsnew.txt") containing about 10000 characters and copy this file into a directory of your choice on both PC's	
		Configure all Ser8UART ports to NOT using a "PrivateIOPort" in the registry (rename or delete the entry)	
82	Test execution for 9600 Baud		
		Configure each INI file of the eight ini files separately for each port	
		port 1 : COM="*" MODE="S" ; TESTRUN=1 ; baudrate = 9600 , 8 , n , 1 , x	
		port 2 : COM="*" MODE="R" ; TESTRUN=1 ; baudrate = 9600 , 8 , n , 1 , x	
		port 3 : COM="*" MODE="S" ; TESTRUN=1 ; baudrate = 9600 , 8 , n , 1 , x	
		port 4 : COM="*" MODE="R" ; TESTRUN=1 ; baudrate = 9600 , 8 , n , 1 , x	
		port 5 : COM="*" MODE="S" ; TESTRUN=1 ; baudrate = 9600 , 8 , n , 1 , d	
		port 6 : COM="*" MODE="R" ; TESTRUN=1 ; baudrate = 9600 , 8 , n , 1 , d	
		port 7 : COM="*" MODE="S" ; TESTRUN=1 ; baudrate = 9600 , 8 , n , 1 , d	
		"*" = available "COMx:" name	
		port 8 : COM="*" MODE="R" ; TESTRUN=1 ; baudrate = 9600 , 8 , n , 1 , d	
		Open 8 separate DOS boxes	
		Start in each DOS box the "Ser8COMM.exe" with the INI file as line command parameter ((between """s) : \>Ser8UART.exe "Full Path and Name to the INI file" and proceed as described in 22 , assuring that during a certain time interval, all connections are active at the same time i.e. start the "Servers" on ports 1, 3, 5, 7 before the "Requesters" on ports 2, 4, 6, 8	
83	Test execution for 115200 Baud		
		Repeat Test run 82 using baudrate = 115200	

"Ser8COMM.INI" Master INI file

```
; Generic Communication Test
; __________________________


; This INI file is meant to be used with the generic Ser8COMM test program.
; The test program must simultaneously be executed on 2 different test PC's
; connected with each through the COM ports that will be tested
; ( the D.U.T. 's ) with means of a "FULL ( DTE-DTE )" NULL modem cable.
; On one side, the program must be started as "Server" , on the other side,
; the program must be started as "Requester". The "Server" should always be
; started first! The program recognizes this operation mode through the entry
; from the "MODE" key ( see further ) out of this INI file.

; To work properly, both requester and server should have identical values
; for the test case to be executed : see "TESTRUN" key values further on.
; To execute a particular test, the commented "TESTRUN" case should be
; uncommented.


; IMPORTANT : THIS .INI FILE MUST RESIDE INTO THE %SystemRoot% DIRECTORY IF
; "ser8COMM.exe" is started without a INI file name as command line parameter !!


[Port]

; 1. COM port setting
; -------------------
; Use only the format "COMx" entries when using the Ser8COMM.exe test
; program. The ports to be tested must also reside on the OS as "COMx"
; ( and not a private name such as "SER1" ). This corresponds to the driver
; registry entry "DosDevices" within the drivers registry parameter list.

COM="COM6:"


; 2. MODE setting
; ---------------
; MODE = "Requester" or "Server" [ may be abbreviated as "R" and "S" ]
; The "Requester" initiates the test case, the "Server" responds to the
; "Requester" with appropriate actions :

MODE="R"
; MODE="S"


; 3. Baudrate, data & stopbits, ...
; ---------------------------------
; Adapt these settings according the test list or your own inventivity
; SET=" baud , databits, parity (y,n) , stopbits ( 0,1,2) ,
; n = none | d = RTS/CTS | x = XON/XOFF "

SET=" 9600 , 8 , n , 1 , x "
; SET=" 9600 , 8 , n , 1 , n "
; SET=" 9600 , 8 , n , 1 , d "
; SET=" 19200 , 8 , n , 1 , x "
; SET=" 19200 , 8 , n , 1 , n "
; SET=" 19200 , 8 , n , 1 , d "
; SET=" 115200 , 8 , n , 1 , x "
; SET=" 115200 , 8 , n , 1 , n "
; SET=" 115200 , 8 , n , 1 , d "


; 4. Test Text file
; -----------------
; A text file containing characters ( maximally 100000 characters )
; Adapt this according your own ideas.

FILE="D:\Ddk\SRC\COMM\Ser8UART\Ser8COMM\WHATSNEW.TXT"


; 5. Test Configuration Cases
; ---------------------------

; 5.1 Case 1
; ----------
; Write and read back an amount of characters from a text file,
; eventually display the read back content

TESTRUN=1

; 5.2 Case 2
; ----------
; Write and read back an amount of characters from a text file with explicit
; XOFF/XON holding periods, eventually displays the read back content
; These runs require that the "SET" key entry contains the "x" setting for
; XON/OFF

; TESTRUN=2

; 5.3 Case 3
; ----------
; Write and read back an amount of characters from a text file with explicit
; CTS/RTS holding periods, eventually displays the read back content
; These runs require that the "SET" key entry contains the "d" setting for
; CTS/RTS

; TESTRUN=3
```

7. References

[1]	"Ser8UART" ISA Card -	
[2]	DATA SHEET SC28L198 "Octal UART for 3.3V and 5V supply Voltage"	Philips 1999 Jan 14 [853-2047-20654]
[3]	"Ser8UART" Driver Test List in XLS format " Ser8UARTTestList.xls"	
[4]	MSDN "Microsoft Developer Network" -	October 2001 or more recent
[5]	Windows Device Driver Kit [DDK] Documentation for Windows NT40 and Windows2000	October 2001 or more recent