NAME

uart - driver for Universal Asynchronous Receiver/Transmitter (UART) devices

SYNOPSIS

device uart

device puc
device uart

device scc
device uart

In */boot/device.hints*:
hint.uart.0.disabled="1"
hint.uart.0.baud="38400"
hint.uart.0.port="0x3f8"
hint.uart.0.flags="0x10"

With *flags* encoded as:

0x00010	device is potential system console
0x00080	use this port for remote kernel debugging
0x00100	set RX FIFO trigger level to ‘‘low’’ (NS8250 only)
0x00200	set RX FIFO trigger level to ‘‘medium low’’ (NS8250 only)
0x00400	set RX FIFO trigger level to ‘‘medium high’’ (default, NS8250 only)
0x00800	set RX FIFO trigger level to ‘‘high’’ (NS8250 only)

DESCRIPTION

The **uart** device driver provides support for various classes of UARTs implementing the EIA RS-232C (CCITT V.24) serial communications interface. Each such interface is controlled by a separate and independent instance of the **uart** driver. The primary support for devices that contain multiple serial interfaces or that contain other functionality besides one or more serial interfaces is provided by the puc(4), or scc(4) device drivers. However, the serial interfaces of those devices that are managed by the puc(4), or scc(4) driver are each independently controlled by the **uart** driver. As such, the puc(4), or scc(4) driver provides umbrella functionality for the **uart** driver and hides the complexities that are inherent when elementary components are packaged together.

The **uart** driver has a modular design to allow it to be used on differing hardware and for various purposes. In the following sections the components are discussed in detail. Options are described in the section that covers the component to which each option applies.

CORE COMPONENT

At the heart of the **uart** driver is the core component. It contains the bus attachments and the low-level interrupt handler.

HARDWARE DRIVERS

The core component and the kernel interfaces talk to the hardware through the hardware interface. This interface serves as an abstraction of the hardware and allows varying UARTs to be used for serial communications.

SYSTEM DEVICES

System devices are UARTs that have a special purpose by way of hardware design or software setup. For example, Sun UltraSparc machines use UARTs as their keyboard interface. Such an UART cannot be used for general purpose communications. Likewise, when the kernel is configured for a serial console, the corresponding UART will in turn be a system device so that the kernel can output boot messages early on in the boot process.

KERNEL INTERFACES

The last but not least of the components is the kernel interface. This component ultimately determines how the UART is made visible to the kernel in particular and to users in general. The default kernel interface is the TTY interface. This allows the UART to be used for terminals, modems and serial line IP applications. System devices, with the notable exception of serial consoles, generally have specialized kernel interfaces.

HARDWARE

The **uart** driver supports the following classes of UARTs:

- NS8250: standard hardware based on the 8250, 16450, 16550, 16650, 16750 or the 16950 UARTs.
- SCC: serial communications controllers supported by the scc(4) device driver.

Pulse Per Second (PPS) Timing Interface

The **uart** driver can capture PPS timing information as defined in RFC 2783. The API, accessed via ioctl(2), is available on the tty device. To use the PPS capture feature with ntpd(8), symlink the tty callout device */dev/cuau?* to */dev/pps0.*

The *hw.uart.pps_mode* tunable configures the PPS capture mode for all uart devices; it can be set in loader.conf(5). The *dev.uart.0.pps_mode* sysctl configures the PPS capture mode for a specific uart device; it can be set in loader.conf(5) or sysctl.conf(5).

The following capture modes are available:

0x00	Capture disabled.

0x01	Capture pulses on the CTS line.
0x02	Capture pulses on the DCD line.

The following values may be ORed with the capture mode to configure capture processing options:

0x10	Invert the pulse (RS-232 logic low = ASSERT, high = CLEAR).
0x20	Attempt to capture narrow pulses.

Add the narrow pulse option when the incoming PPS pulse width is small enough to prevent reliable capture in normal mode. In narrow mode the driver uses the hardware's ability to latch a line state change; not all hardware has this capability. The hardware latch provides a reliable indication that a pulse occurred, but prevents distinguishing between the CLEAR and ASSERT edges of the pulse. For each detected pulse, the driver synthesizes both an ASSERT and a CLEAR event, using the same timestamp for each. To prevent spurious events when the hardware is intermittently able to see both edges of a pulse, the driver will not generate a new pair of events within a half second of the prior pair. Both normal and narrow pulse modes work with ntpd(8).

Add the invert option when the connection to the uart device uses TTL level signals, or when the PPS source emits inverted pulses. RFC 2783 defines an ASSERT event as a higher-voltage line level, and a CLEAR event as a lower-voltage line level, in the context of the RS-232 protocol. The modem control signals on a TTL-level connection are typically inverted from the RS-232 levels. For example, carrier presence is indicated by a high signal on an RS-232 DCD line, and by a low signal on a TTL DCD line. This is due to the use of inverting line driver buffers to convert between TTL and RS-232 line levels in most hardware designs. Generally speaking, a connection to a DB-9 style connector is an RS-232 level signal at up to 12 volts. A connection to header pins or an edge-connector on an embedded board is typically a TTL signal at 3.3 or 5 volts.

Special Devices

The **uart** driver also supports an initial-state and a lock-state control device for each of the callin and the callout "data" devices. The termios settings of a data device are copied from those of the corresponding initial-state device on first opens and are not inherited from previous opens. Use stty(1) in the normal way on the initial-state devices to program initial termios states suitable for your setup.

The lock termios state acts as flags to disable changing the termios state. E.g., to lock a flag variable such as CRTSCTS, use *stty crtscts* on the lock-state device. Speeds and special characters may be locked by setting the corresponding value in the lock-state device to any nonzero value. E.g., to lock a speed to 115200, use "stty 115200" on the initial-state device and "stty 1" on the lock-state device.

Correct programs talking to correctly wired external devices work with almost arbitrary initial states and almost no locking, but other setups may benefit from changing some of the default initial state and locking the state. In particular, the initial states for non (POSIX) standard flags should be set to suit the devices attached and may need to be locked to prevent buggy programs from changing them. E.g., CRTSCTS should be locked on for devices that support RTS/CTS handshaking at all times and off for devices that do not support it at all. CLOCAL should be locked on for devices that do not support carrier. HUPCL may be locked off if you do not want to hang up for some reason. In general, very bad things happen if something is locked to the wrong state, and things should not be locked for devices that support more than one setting. The CLOCAL flag on callin ports should be locked off for logins to avoid certain security holes, but this needs to be done by getty if the callin port is used for anything else.

FILES

/dev/ttyu?	for callin ports
/dev/ttyu?.init	
/dev/ttyu?.lock	corresponding callin initial-state and lock-state devices
/dev/cuau?	for callout ports
/dev/cuau?.init	
/dev/cuau?.lock	corresponding callout initial-state and lock-state devices

SEE ALSO

cu(1), puc(4), scc(4), ttys(5)

HISTORY

The **uart** device driver first appeared in FreeBSD 5.2.

AUTHORS

The **uart** device driver and this manual page were written by Marcel Moolenaar <*marcel@xcllnt.net*>.