Aricent
.ALTRAN GROUP

UART Driver

(Universal Asynchronous Receiver/Transmitter)

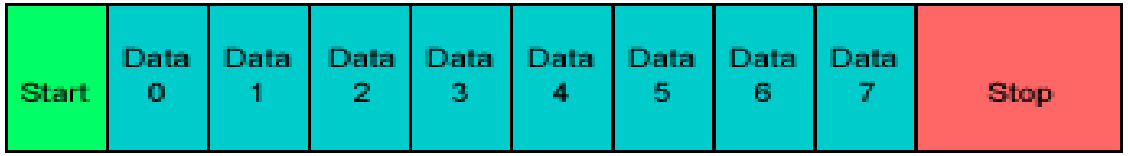
Outline

- What is UART?
- Protocol, Variations
- Signals
- DB9 Stuff
- RS 232 Transmission examples
- TTY Driver use case
- TTY Serial Subsystem
- Low Level Driver API
- UART Data Structure

What is UART?

- Universal Asynchronous Receiver/Transmitter
- Hardware that translates between parallel and serial forms
- Commonly used in conjunction with communication standards such as EIA, RS-232, RS-422 or RS-485
- The universal designation indicates that the data format and transmission speeds are configurable and that the actual electric signaling levels and methods (such as differential signaling etc.) typically are handled by a special driver circuit external to the UART.

Protocol



- Each character is sent as
  - a logic *low* **start** bit
  - a configurable number of data bits (usually 7 or 8, sometimes 5)
  - an optional parity bit
  - *one or more logic high* **stop** bits
  - with a particular bit timing (“baud”)
- Examples
  - “9600-N-8-1” → <baudrate><parity><databits><stopbits>
  - “9600-8-N-1” → <baudrate><databits><parity><stopbits>

Variations and fun times

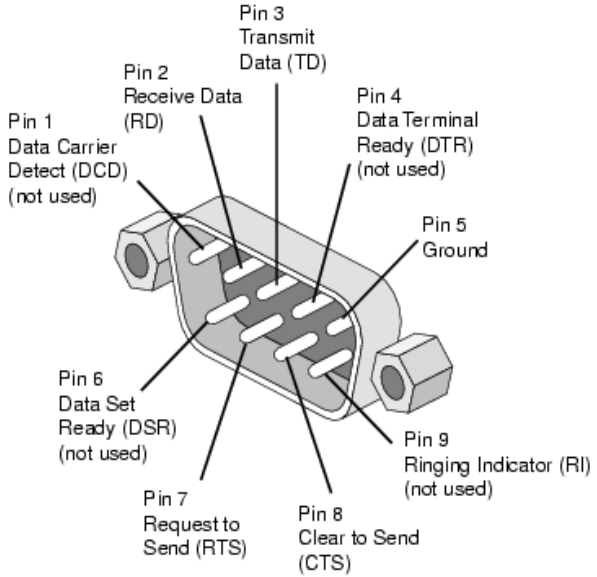
- UART is actually a generic term that includes a large number of different devices/standards.
  - RS-232 is a standard that specifies
    - “electrical characteristics and timing of signals, the meaning of signals, and the physical size and pin out of connectors

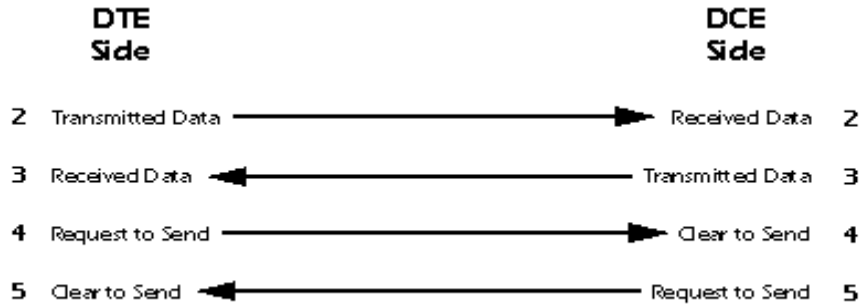
Signals (only most common)

- The **RXD** signal of a UART is the signal receiving the data. This will be an input and is usually connected to the TXD line of the downstream device.
- The **TXD** signal of a UART is the signal transmitting the data. This will be an output and is usually connected to the RXD line of the downstream device.
- The **RTS#** (Ready to Send) signal of a UART is used to indicate to the downstream device that the device is ready to receive data. This will be an output and is usually connected to the CTS# line of the downstream device.
- The **CTS#** (Clear to Send) signal of a UART is used by the downstream device to identify that it is OK to transmit data to the upsteam device. This will be an input and is usually connected to the RTS# line of the upstream device

DB9 stuff

- DTE vs DCE
- Pinout of a DCE?
- Common ground?
- Noise effects?



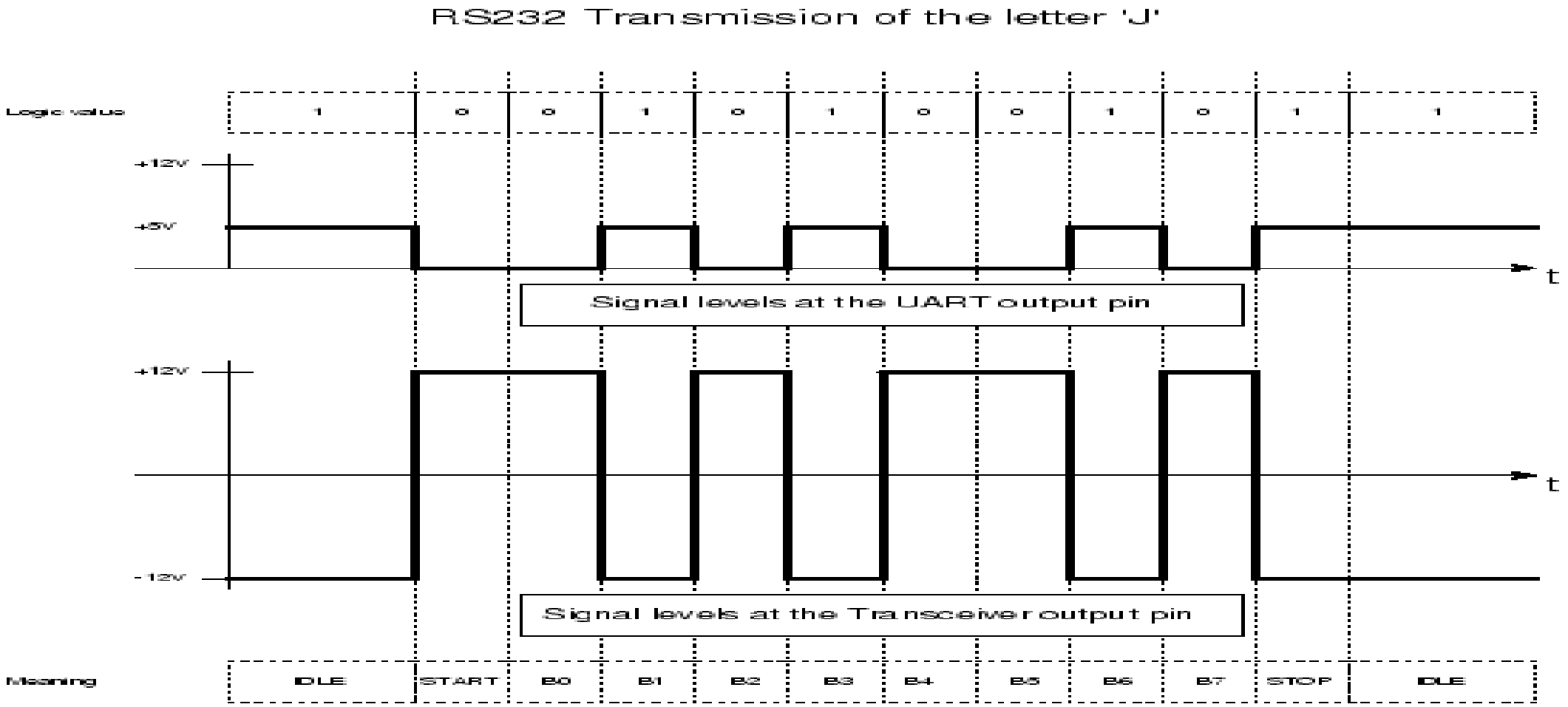


Pin Number	Signal	Description
1	DCD	Data carrier detect
2	RxD	Receive Data
3	TxD	Transmit Data
4	DTR	Data terminal ready
5	GND	Signal ground
6	DSR	Data set ready
7	RTS	Ready to send
8	CTS	Clear to send
9	RI	Ring Indicator

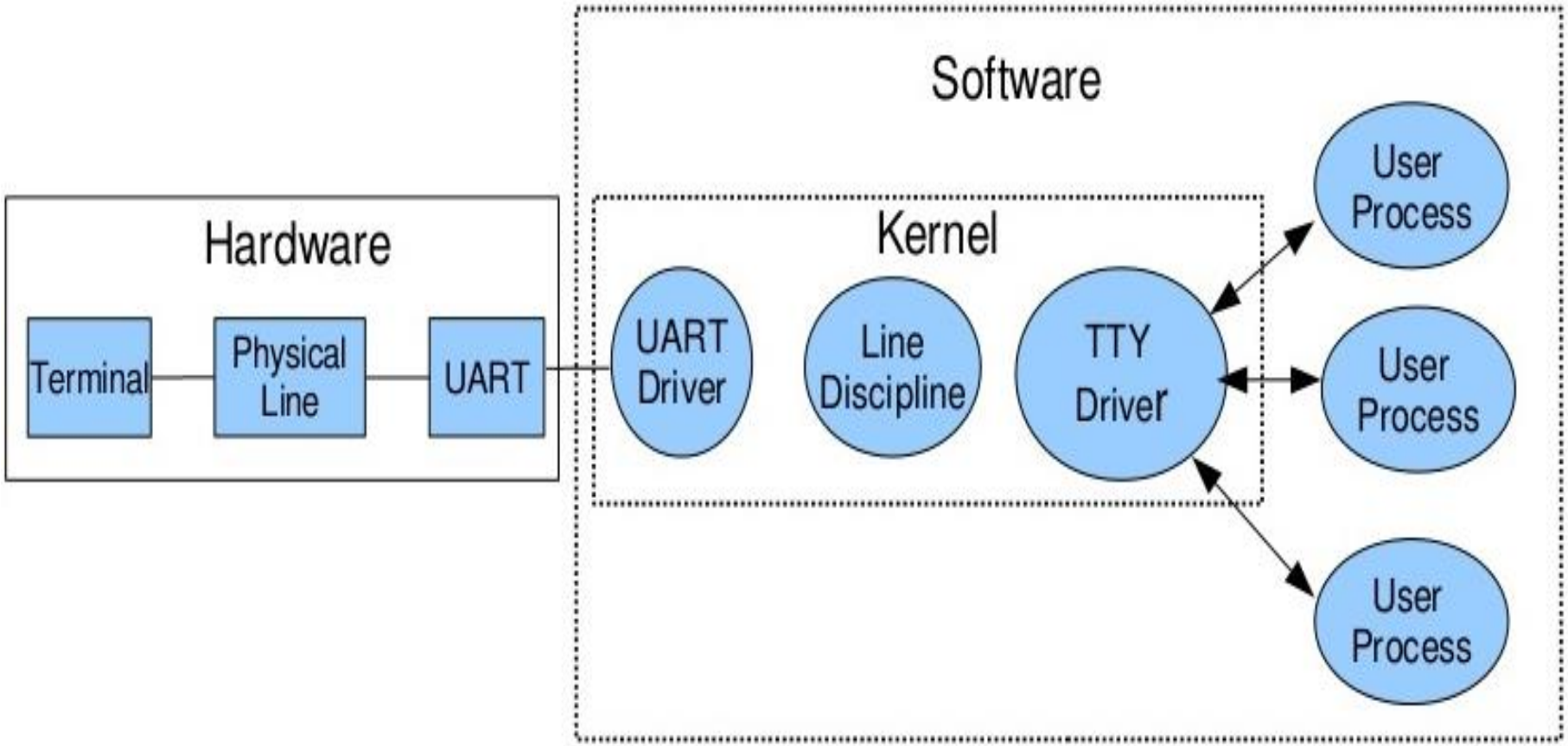
Wiring a DTE device to a DCE device for communication is easy.
The pins are a one-to-one connection, meaning all wires go from pin x to pin x.
A straight through cable is commonly used for this application.
In contrast, wiring two DTE devices together requires crossing the transmit and receive wires.
This cable is known as a null modem or crossover cable.

Aricent
.ALTRAN GROUP

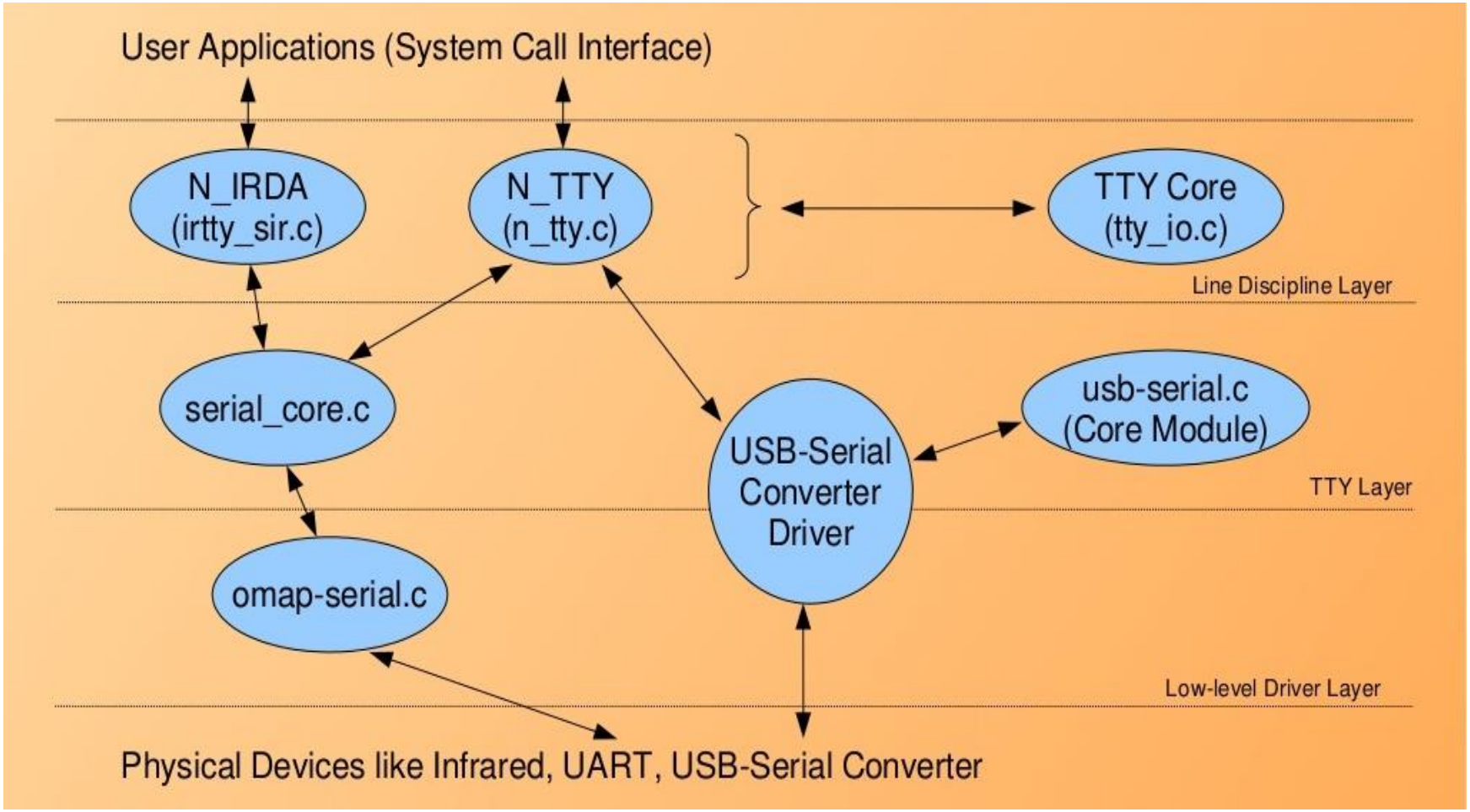
RS-232 transmission example



TTY Driver Use Case



TTY Serial Subsystem



Low level (UART) Driver Related Data Structures &APIs

- Data Structures
  - struct uart_driver
  - struct uart_port
  - struct uart_ops
- APIs
  - int uart_register_driver(struct uart_driver *);
  - int uart_unregister_driver(struct uart_driver *);
  - int uart_add_one_port(struct uart_driver *, struct uart_port);
  - int uart_remove_one_port(struct uart_driver *, struct uart_port);
- Typically implemented as a platform driver
- Header: <linux/serial_core.h>

UART Driver structure

- struct uart_driver
  - struct module *owner;
  - const char *driver_name; /* Name */
  - const char *dev_name; /* dev node names */
  - int major; /* Major number */
  - int minor; /* Minor number */
  - struct tty_driver *tty_driver; /* tty driver */

UART Port structure

- struct uart_port
  - spinlock_t lock; /* lock */
  - unsigned int iobase; /* in/out[bwl] */
  - unsigned char __iomem *membase; /* I/O membase */
  - unsigned int irq; /* irq number */
  - unsigned int uartclk; /* base uart clock */
  - unsigned char fifosize; /* tx fifo size */
  - unsigned char x_char; /* flow control */

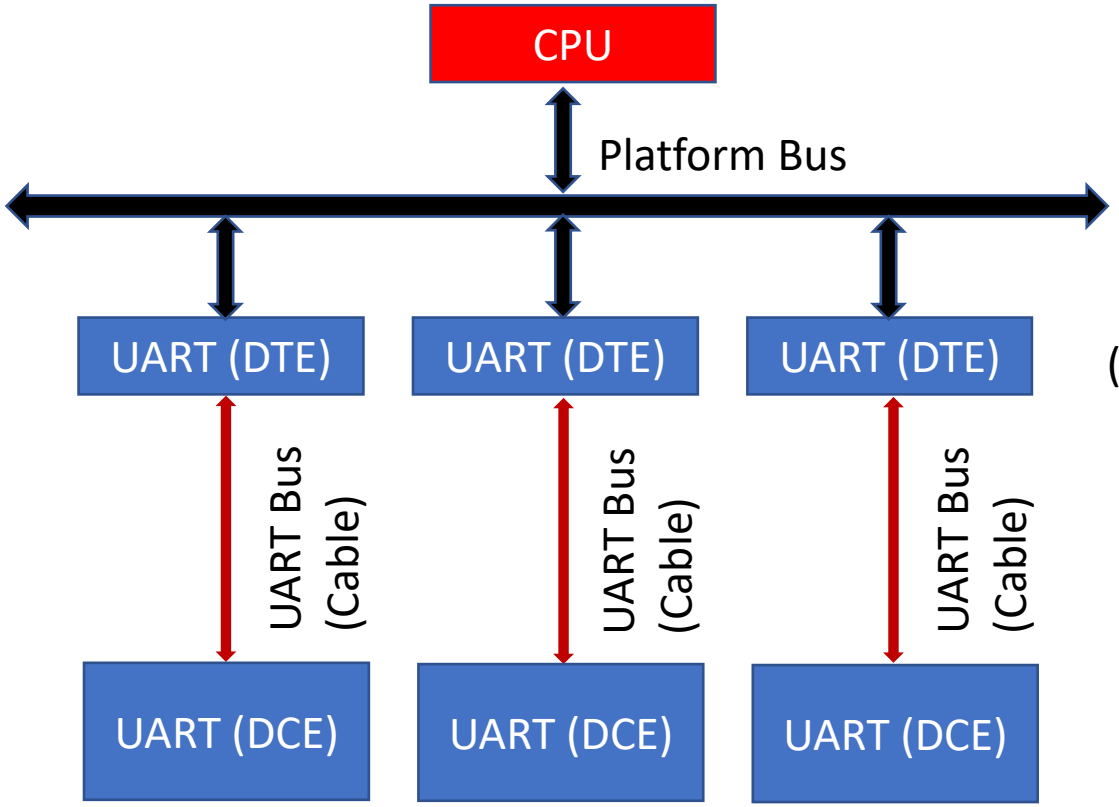
UART Operations structure

- struct uart_ops
  - uint (*tx_empty)(struct uart_port *); /* Is TX FIFO empty? */
  - void (*set_mctrl)(struct uart_port *, unsigned int mctrl); /* Set modem control params */
  - uint (*get_mctrl)(struct uart_port *); /* Get modem control params */
  - void (*stop_tx)(struct uart_port *); /* Stop xmission */
  - void (*start_tx)(struct uart_port *); /* Start xmission */
  - void (*shutdown)(struct uart_port *); /* Disable the port */
  - void (*set_termios)(struct uart_port *, struct termios *new, struct termios *old); /* Terminal interface parameters */
  - void (*config_port) (struct uart_port *, int);

TTY Driver related Data Structures &APIs

- Header: <linux/tty.h>
- Data Structures
  - struct tty_struct
  - struct tty_flip_buffer
  - struct tty_driver
- APIs
  - int tty_register_driver(struct tty_driver *);
  - int tty_unregister_driver(struct tty_driver *);
  - struct device *tty_register_device(struct tty_driver *, unsigned, struct device);
  - void tty_unregister_device(struct tty_driver *, unsigned);

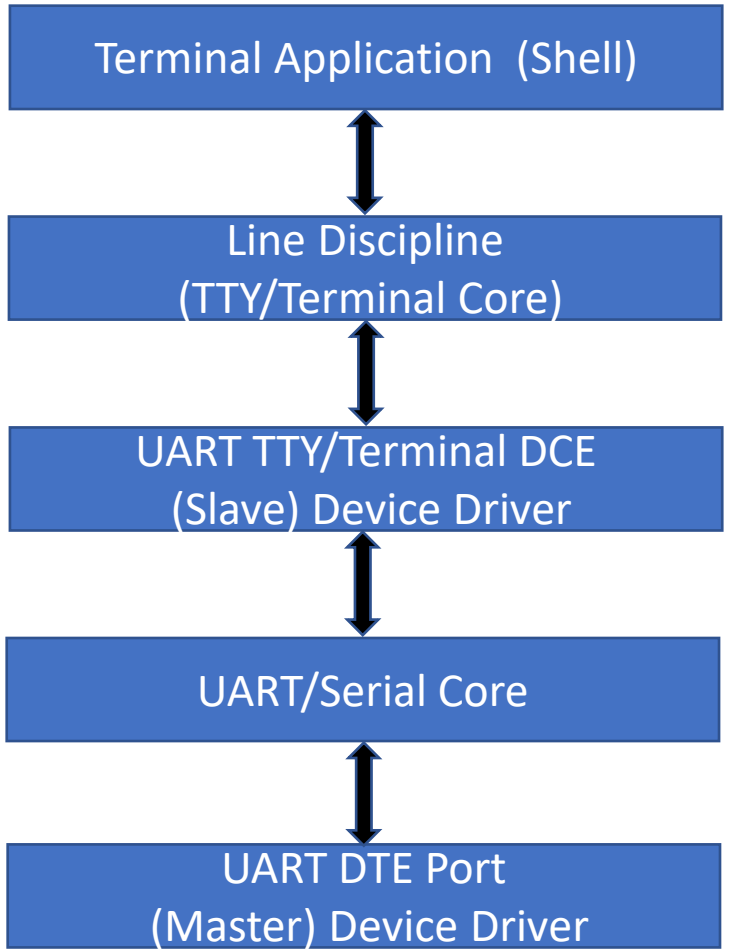
UART – Platform Bus H/W Architecture



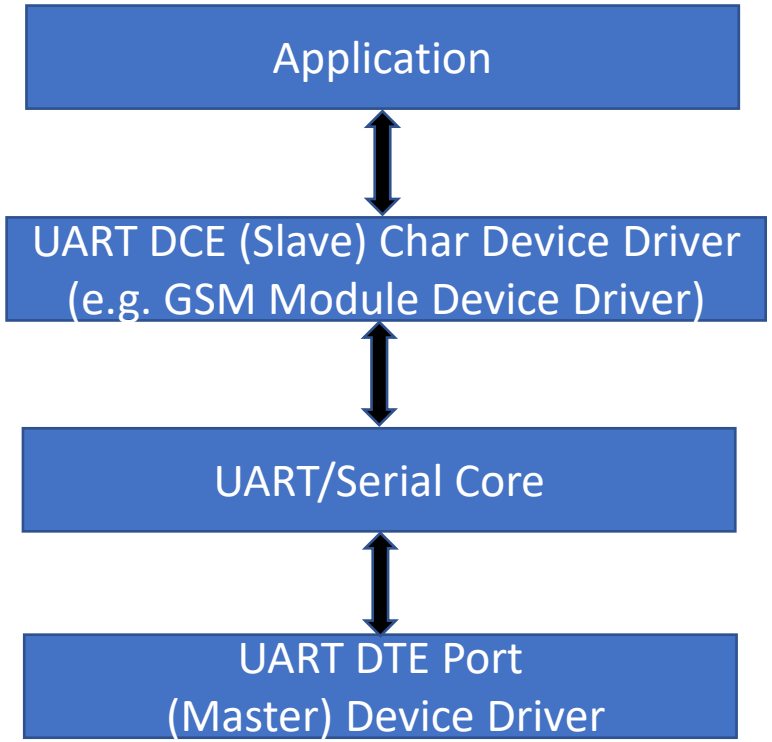
(UART Master/Controller Devices)

(UART Slave/Client Devices)

UART-Terminal Driver Stack/Architecture



Normal UART Device Driver Stack/Architecture



What all we learnt?

- What is UART?
- Protocol, Variations
- Signals
- DB9 Stuff
- RS 232 Transmission examples
- TTY Driver use case
- TTY Serial Subsystem
- Low Level Driver API
- UART Data Structure

Discussion Questions

- How fast can we run a UART?
- What are the limitations?
- Why do we need start/stop bits?
- How many data bits can be sent?
    - 9600-8-N-1 is ok. Is 9600-8192-N-1 ok too?

Aricent
.ALTRAN GROUP

Any Queries?