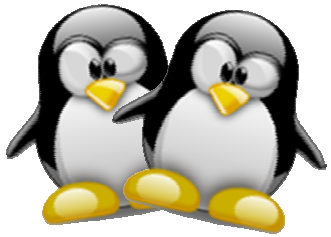


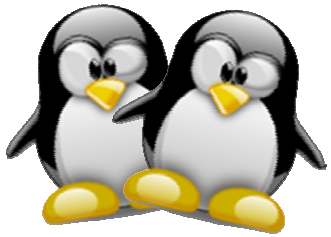
The latest UART driver model for porting serial devices as standard Linux serial ports

Jim Chen

(jim.chen@hytec-electronics.co.uk)

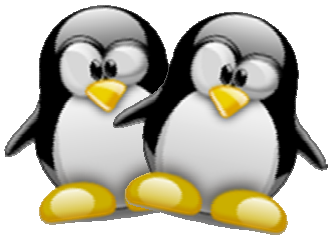
14/06/2011

Hytec Electronics Ltd



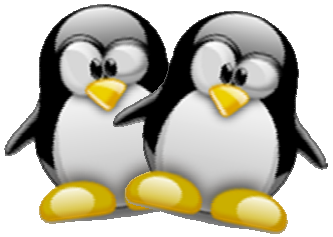
Contents

- Overview of the hardware and software
- TTY device driver
- New UART device driver
- An example in brief
- Summary

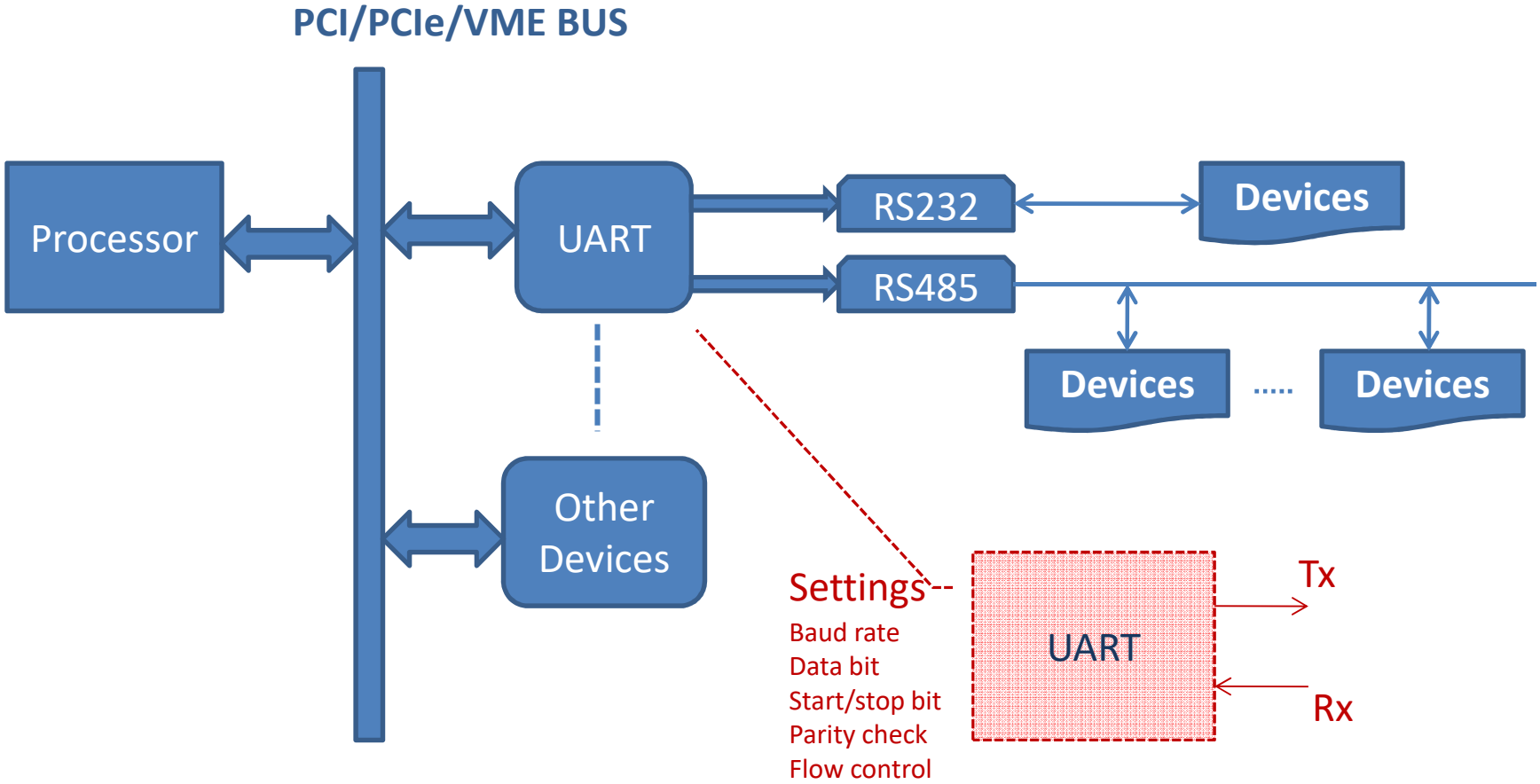


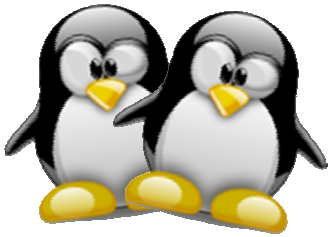
We all familiar with

```
drvAsynSerialPortConfigure("port1","/dev/ttyS0",0,0,0)
asynSetOption(" port1 ", -1, "baud", "9600")
asynSetOption(" port1 ", -1, "bits", "8")
asynSetOption(" port1 ", -1, "parity", "none")
asynSetOption(" port1 ", -1, "stop", "1")
asynSetOption(" port1 ", -1, "clocal", "Y")
asynSetOption(" port1 ", -1, "rtsts", "N")
```

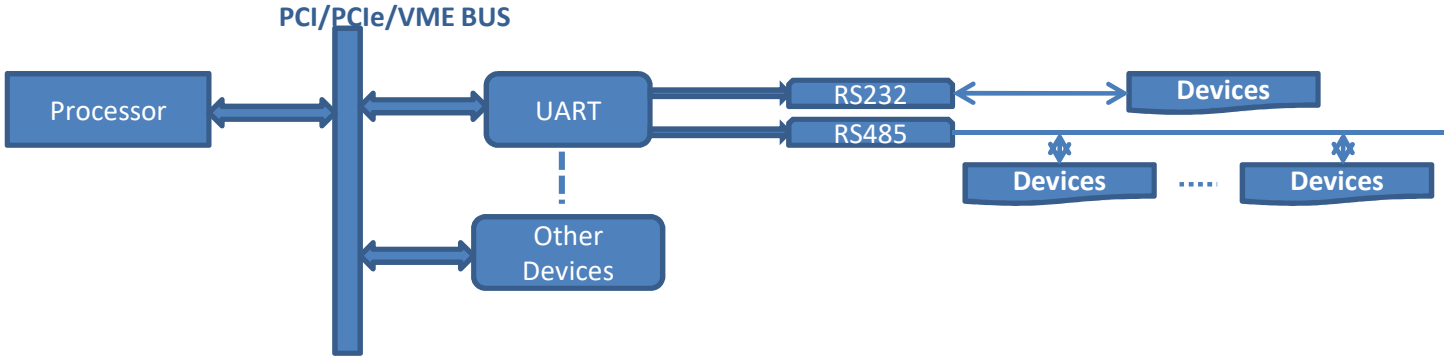


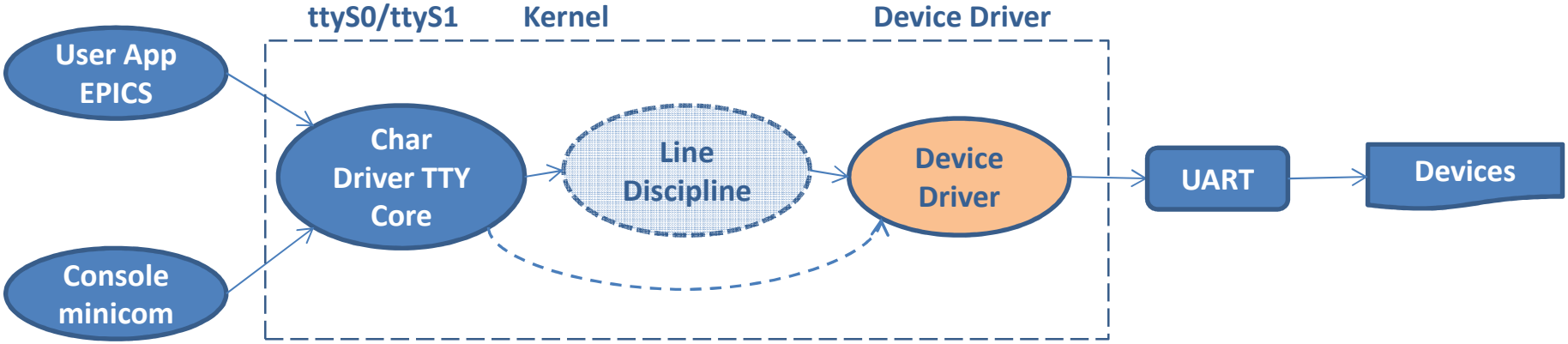
Serial Communication Hardware

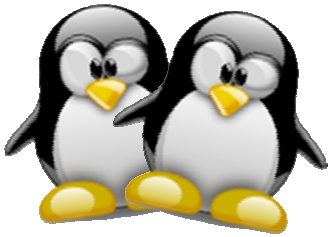




Software Overview







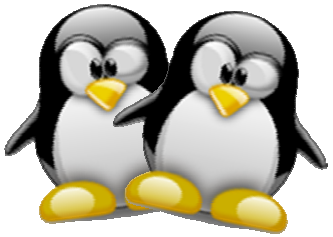
What is our goal

Majority of serial users – sending/receiving data over 3/2/4 wires

- Set up serial port in the simplest way – baud rate, data bit, stop bit
- No parity check, no flow control, no echo etc.
- Sending data via Tx.
- Receiving data via Rx.

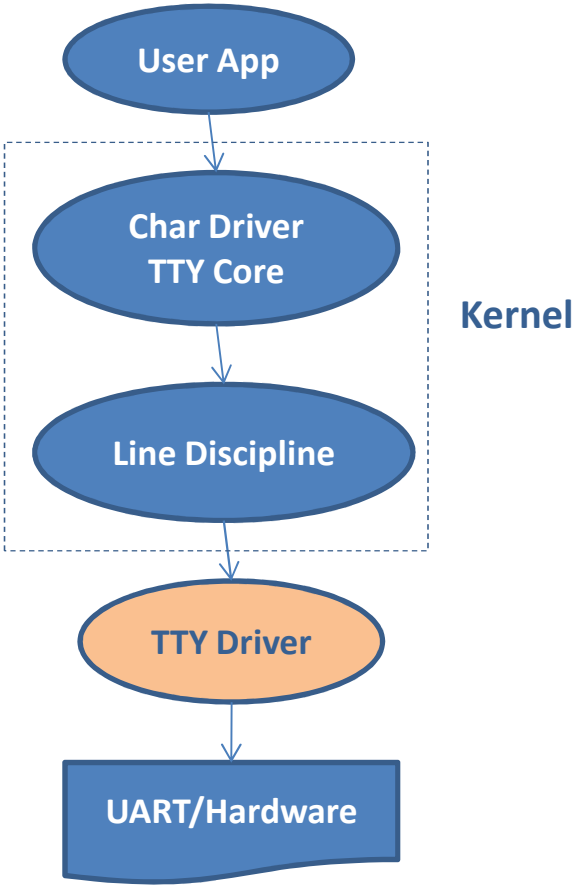
Options (most apps don't)

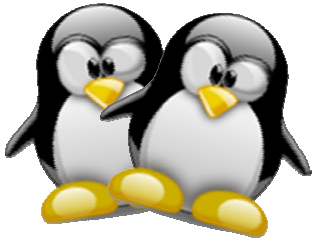
- Manage modem – CTS, DSR, DTR, RTS, CD, RI etc, etc.
- Manage flow control and handshake.
- Manage special characters etc.



What the driver used to be – TTY device driver

- Before kernel 2.6 (2.5.28)
- Serial ports must be visible as TTY devices
- Must be part of the kernel TTY core
- Direct implementation by TTY driver



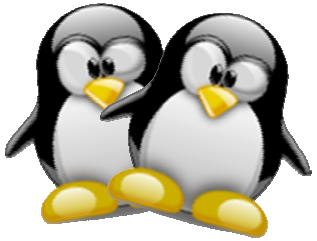


A glance at TTY device driver

- Detecting devices
- In init routine:
  - Allocate tty_driver
  - Register tty_driver: name, driver name, major, minor, subtype, termios etc
  - Register devices
- Implement operations
- Use ‘write’ operation to send
- Use interrupt (receiving chars) or timer (polling) for receiving.

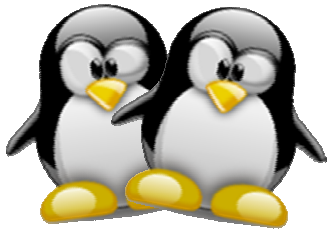
Operations

```
static struct tty_operations serial_ops = {
   .open              = my_open,
   .close             = my_close,
   .write             = my_write,
   .write_room        = my_write_room,
   .set_termios       = my_set_termios,
   .ioctl             = my_ioctl,
   .put_char          = my_put_char,
   .flush_chars       = my_flush_chars,
   .wait_until_sent   = my_wait_until_sent,
   .chars_in_buffer   = my_chars_in_buffer,
   .throttle          = my_throttle,
   .unthrottle        = my_unthrottle,
   .stop              = my_stop,
   .start             = my_start,
   .hangup            = my_hangup,
   .break_ctl         = my_break_ctl,
   .set_ldisc         = my_set_ldisc,
   .send_xchar        = my_send_xcahr,
};
```



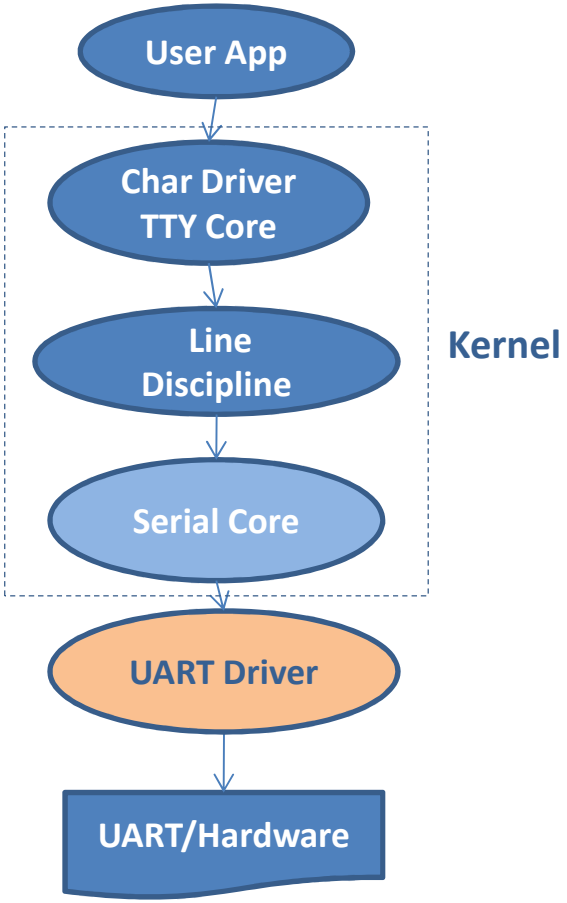
Problems

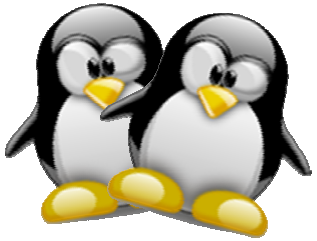
- We need to deal with TTY stuff all the time
- A bit messy with numerous #ifdef lines
- Very complicated to implement
- Not UART oriented
- Flip Buffer issue. A device driver receive handler can run in parallel with the flush routine in SMP system
- Multiple instances of the flush routine can run in parallel
- Since the tty buffers are attached directly to the tty. It is causing a lot of the problems related to tty layer locking, also problems at high speed and also with bursty data etc
- ……



What it is now – UART driver

- After kernel 2.6, TTY driver split into two
- kernel programmer has implemented serial_core which is equivalent to a tty wrapper
- Driver developer only need to deal with UART operations



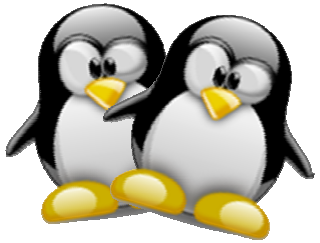


The UART driver Implementation

- Detecting devices via PCI enumeration
- Register UART driver structure
- Add UART ports
- Implement the operations
- Implement the console operations (optional)
- Use interrupt (receiving chars) or timer (polling) for receiving.
- Sending upon TX holding register empty interrupt

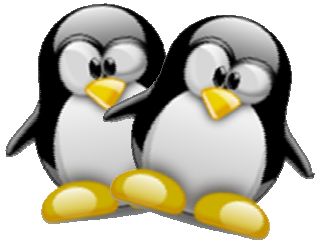
Operations

```
static struct uart_ops ioc9010_uart_pops = {
    .tx_empty       = ioc9010_tx_empty,
    .set_mctrl      = ioc9010_set_mctrl,
    .get_mctrl      = ioc9010_get_mctrl,
    .stop_tx        = ioc9010_stop_tx,
    .start_tx       = ioc9010_start_tx,
    .stop_rx        = ioc9010_stop_rx,
    .enable_ms      = ioc9010_enable_ms,
    .break_ctl      = ioc9010_break_ctl,
    .startup        = ioc9010_startup,
    .shutdown       = ioc9010_shutdown,
    .set_termios    = ioc9010_set_termios,
    .type           = ioc9010_type,
    .release_port   = ioc9010_release_port,
    .request_port = ioc9010_request_port,
    .config_port    = ioc9010_config_port,
    .verify_port    = ioc9010_verify_port,
};
```



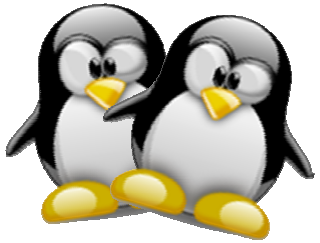
Implementation Details

```
static struct uart_ops ioc9010_uart_pops = {
    .tx_empty      = ioc9010_tx_empty,        check transmt FIFO empty
    .set_mctrl     = ioc9010_set_mctrl,       set modem control lines, RTS, DTR etc
    .get_mctrl     = ioc9010_get_mctrl,       get modem control status, CTS, DCD, DSR RI etc
    .stop_tx       = ioc9010_stop_tx,         stop transmitting
    .start_tx      = ioc9010_start_tx,        enable transmitting
    .stop_rx       = ioc9010_stop_rx,         stop receiving, stop receive interrupt
    .enable_ms     = ioc9010_enable_ms,       set modem timer
    .break_ctl     = ioc9010_break_ctl,       control the transmission of a break signal
    .startup       = ioc9010_startup,         initialise port, register and enable interrupt
    .shutdown      = ioc9010_shutdown,        stop using port. Disable and free interrupt resources
    .set_termios   = ioc9010_set_termios,     set line pararmeters: baud rate, data bit, stop bit etc
    .type          = ioc9010_type,            return the string of the port device
    .release_port  = ioc9010_release_port,    release memory and IO region used by the port
    .request_port = ioc9010_request_port,     request memory and IO region
    .config_port   = ioc9010_config_port,     perform any auto configuration steps
    .verify_port   = ioc9010_verify_port,     verify the port info is suitable for this port type
};
```

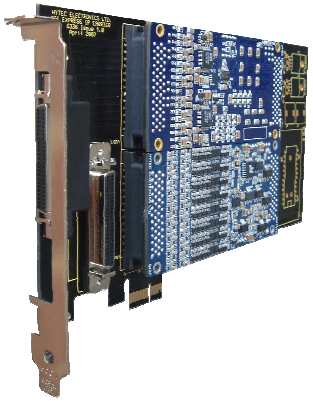


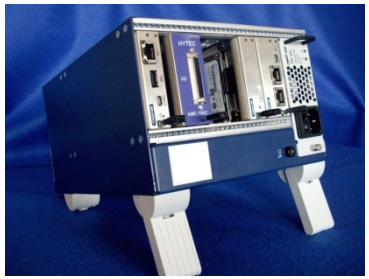
What can we tell from the new model

- This is much closer to the nitty-gritty of it -- the UART business. We are talking the same language.
- Very neat in design.
- No need for complicated TTY stuff contemplation. The UART structure replaces the TTY structure
- Let's look at an example…



Example: Steps for building Hytec 8515/8516 8 channel serial IP UART driver



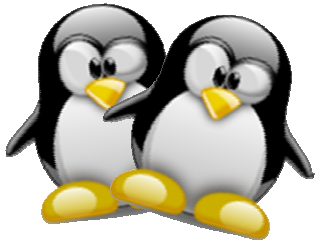


6335 PCIe Carrier
2 IPs

7002 uTCA Carrier
1 IP

IOC9010 Blade
6 IPs

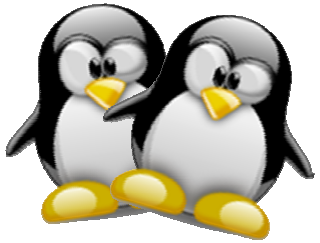
PCI/PCIe arch



Example: Hytec 8515/8516 (Continued 1)

1. Register driver structure

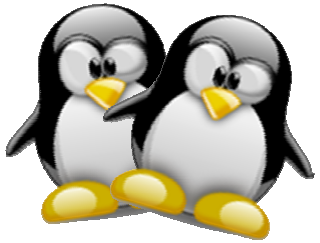
```
static int __init ioc9010_serial_init(void)
{
        int status;
        status = uart_register_driver(&ioc9010_uart_driver);
        if (status == 0)
                ioc9010_init_ports();
        return status;
}
```



Example: Hytec 8515/8516 (Continued 2)

2. Identify PCI/PCIe and uart devices

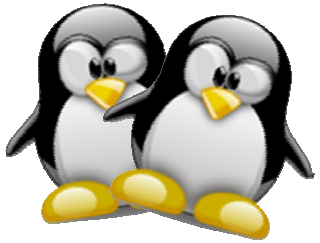
```
int buildHytecPciDeviceList(void)
{
      .....
       //Enumerate PCI devices
       pPciDev = pci_get_device(PCI_ANY_ID, PCI_ANY_ID, NULL);

       while(pPciDev){
              if(checkHytecPciDevice(pPciDev)){
                  status = addPciDevice(pPciDev);                    // Add the device to the device list
                  if (status != 0) serialCount += status;            // Increment device counter
              }
             pPciDev = pci_get_device(PCI_ANY_ID, PCI_ANY_ID, pPciDev);   // check next device
         }
        return serialCount;
}
```



Example: Hytec 8515/8516 (Continued 3)

3. Register UART port

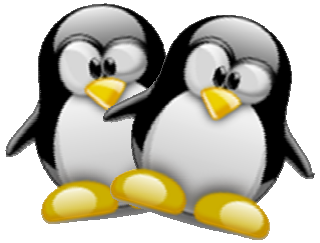
```
static void __init ioc9010_init_ports(void)
{
        ......
        serialCardNos = buildHytecPciDeviceList();
        for(i=0; i<serialCardNos; i++){
                ......
                /* initialise uart default */
                ioc9010_ports[k].baud = 9600;
                ioc9010_ports[k].parity = 'N';
                ioc9010_ports[k].stopbits = 1;
                ioc9010_ports[k].databits = 8;
                ioc9010_ports[k].flowControl = 0;

                ioc9010_ports[k].port.ops = &ioc9010_uart_pops;
                ioc9010_ports[k].port.iobase = (unsigned long)(ioc9010dev->iocMemoryAddr +
                        IP_IO_BASE_ADDR + IP_A_IO_BASE_ADDR + (IP_B_ID_BASE_ADDR - IP_A_ID_BASE_ADDR) *
                        ioc9010_ports[k].ipSlot);
```



Example: Hytec 8515/8516 (Continued 4)

4. Implement startup

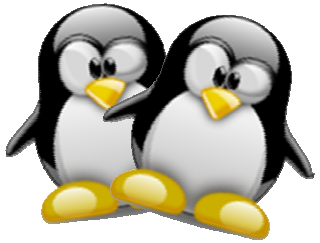
```
static int ioc9010_startup(struct uart_port *port)
{
        struct ioc9010_uart_port *ioc9010 = (struct ioc9010_uart_port *)port;
        int retval = 0;
        spin_lock_irq(&ioc9010->port.lock);
        if(ioc9010->moduleType == SIGNATURE_8516)                    /* for 8516 */
                Hy8516_ComsSetup(ioc9010, portNo, baud, parity, stopbits, databits, flowControl, duplex,
                delay485);
        else                                                         /* for 8515 */
                Hy85156_ComsSetup(ioc9010, portNo, baud, parity, stopbits, databits, flowControl);
        retval = request_irq(ioc9010->port.irq, ioc9010_isr, SA_INTERRUPT | SA_SHIRQ, "Hytec Serial INT
        Handler", ioc9010);                                  /* Request the IRQ and set ISR */
        retval = Hy85156_EnableInterrupt(ioc9010, ioc9010->portNo, ioc9010->mask,
                ioc9010->rcvTrigLevel, ioc9010->transTrigLevel);     /* Enable interrupts */
        spin_unlock_irq(&ioc9010->port.lock);
        return retval;
}
```



Example: Hytec 8515/8516 (Continued 5)

4. Implement shut down

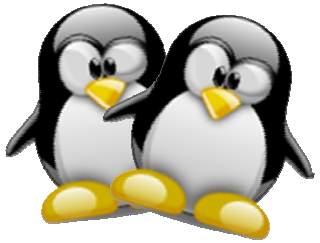
```
static int ioc9010_shutdown(struct uart_port *port)
{
        struct ioc9010_uart_port *ioc9010 = (struct ioc9010_uart_port *)port;
        Hy85156_Flush_Single(ioc9010, ioc9010->portNo, 1, 1);
        Hy85156_ResetInterrupt(ioc9010, ioc9010->portNo, reason);
        Hy85156_DisableInterrupt(ioc9010, ioc9010->portNo, 0); here we cannot disable the whole thing.

        free_irq(ioc9010->port.irq, ioc9010);
}
```



Example: Hytec 8515/8516 (Continued 6)

5. Implement start_tx

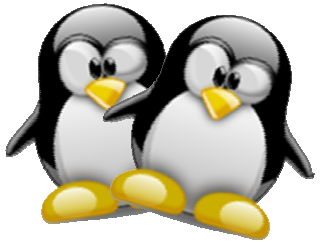
```
static void ioc9010_start_tx(struct uart_port *port)
{
        struct ioc9010_uart_port *ioc9010 = (struct ioc9010_uart_port *)port;

        ioc9010->mask |= ENABLE_THR_INT_85156;
        Hy85156_EnableInterrupt(ioc9010, ioc9010->portNo, ioc9010->mask,
                                                ioc9010->rcvTrigLevel, ioc9010->transTrigLevel);
}
```



Example: Hytec 8515/8516 (Continue 7)

6. Implement stop_tx

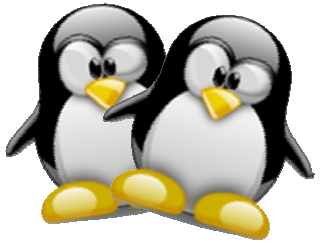
```
static void ioc9010_stop_tx(struct uart_port *port)
{
        struct ioc9010_uart_port *ioc9010 = (struct ioc9010_uart_port *)port;
        int mask;

        mask = ioc9010->mask & (~ENABLE_RHR_INT_85156);
        Hy85156_DisableInterrupt(ioc9010, ioc9010->portNo, mask);
}
```



Example: Hytec 8515/8516 (Continue 8)

7. Implement stop_rx

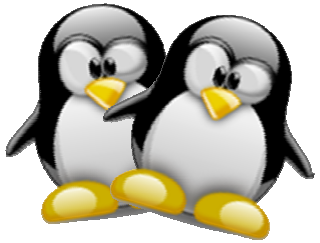
```
static void ioc9010_stop_rx(struct uart_port *port)
{
        struct ioc9010_uart_port *ioc9010 = (struct ioc9010_uart_port *)port;
        int mask;

        mask = ioc9010->mask & (~ENABLE_RHR_INT_85156);
        Hy85156_DisableInterrupt(ioc9010, ioc9010->portNo, mask);
}
```



Example: Hytec 8515/8516 (Continue 9)

8. Implement interrupt service routine

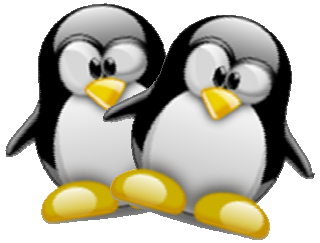
```
irqreturn_t ioc9010_isr(int irq, void *dev_id, struct pt_regs *regs)
{
        struct ioc9010_uart_port *ioc9010 = (struct ioc9010_uart_port *)dev_id;
        spin_lock(&ioc9010->port.lock);
        ......
        Hy85156_INTReason(ioc9010, &port, &reason);

        if ((reason & INTERRUPT_REASON_RXRDY) || ((reason & INTERRUPT_REASON_LSR) && (reason &
                INTERRUPT_REASON_LSR_RX_BREAK)))
            ioc9010_rx_chars(ioc9010); /* Byte or break signal received */

        if (reason & INTERRUPT_REASON_TXRDY)
            ioc9010_tx_chars(ioc9010); /* TX holding register empty - transmit a byte if there is */

        Hy85156_ResetInterrupt(ioc9010, ioc9010->portNo, reason);
        spin_unlock(&ioc9010->port.lock);
        return IRQ_HANDLED;
}
```



Example: Hytec 8515/8516 (Continue 10)

9. Implement transmitting: tx_char

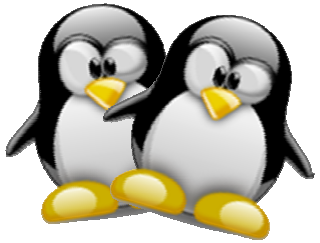
```
static void ioc9010_tx_chars(struct ioc9010_uart_port *ioc9010)
{
        ......
        /* Check available chars for transmitting */
        if (uart_circ_empty(xmit) || uart_tx_stopped(&ioc9010->port)) {
                ioc9010_stop_tx(&ioc9010->port);
                return;
        }

        /* check transmitting FIFO availability */
        status = IOC9010IPMemoryRead(ioc9010->virtureMemBaseAddr, ioc9010->ipSlot, (ULONG) ioc9010->
                portNo * PORT_OFFSET + TXCNT_85156*2, 1, &data);            //read TXCNT register
        while (data < 40)    {
                value[0] = xmit->buf[xmit->tail];
                Hy85156_Write(ioc9010, ioc9010->portNo, value, 1);
                xmit->tail = (xmit->tail + 1) & (UART_XMIT_SIZE - 1);
                ioc9010->port.icount.tx++;
                if (uart_circ_empty(xmit))
                        break;
```



Example: Hytec 8515/8516 (Continue 11)

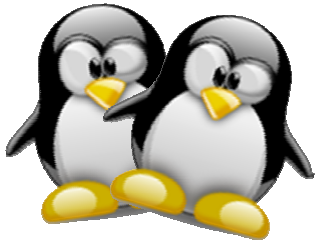
10. Implement receiving: rx_char

```
static void ioc9010_rx_chars(struct ioc9010_uart_port *ioc9010)
{
        ......
       status = IOC9010IPMemoryRead(ioc9010->virtureMemBaseAddr, ioc9010->ipSlot, (ULONG) ioc9010->
                portNo * PORT_OFFSET + RHR_85156*2, 1, &data);       //read data from buffer to value pointer
        flag = TTY_NORMAL;
        ch = (unsigned int) data & 0xFF;
        ioc9010->port.icount.rx++;

        // check status LSR and errors ......

        if (uart_handle_sysrq_char(&ioc9010->port, ch, NULL))
                                        continue;
        uart_insert_char(&ioc9010->port, data, LSR_OVERRUN_85156, ch, flag);

        tty_flip_buffer_push(ioc9010->port.info->tty);
/*      tty_flip_buffer_push(ioc9010->port.state->port.tty);kernel2.6.37*/
}
```



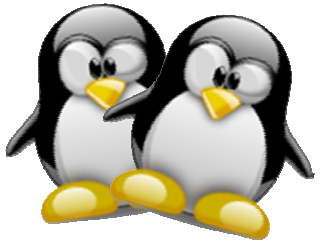
Recommendations

- Download the latest Linux source, say 2.6.37
- Look at the source examples in linux-2.6.37/drivers/serial
  A number of devices have been ported as the uart driver.
- Good examples: bcm63xx_uart.c, pnx8xxx_uart.c
- It is worth reading the serial_core.c which wraps the tty driver stuff.



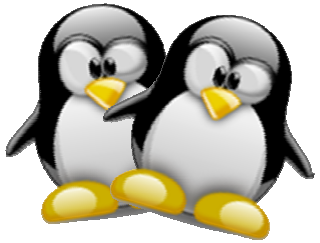
Things to know and Caveats

- Continuous changing (improvement) from version to version
- port.info->tty ⇔ port.state->port.tty
- uart_event(port, EVT_WRITE_WAKEUP);
  => uart_write_wakeup(&ioc9010->port);
- new tty_port structure. 2.6.27 (October 9, 2008)
- tty buffering layer revamp 2.6.16 (Aug 4, 2006)
- We still end up with some #if … #endif
- Console implementation
- Platform implementation



References

- Linux Device Driver, 3rd edition 2005 (a bit obsolete)
- Low Level Serial API
  http://rhkernel.org/RHEL6+2.6.32-19.el6/Documentation/serial/driver
- The Serial Driver Layer http://www.linuxjournal.com/article/6331
- TTY flip buffer SMP changes
  http://lwn.net/Articles/105770/
- Linux: tty Cleanup http://kerneltrap.org/node/5473
- API changes in the 2.6 kernel series
  http://lwn.net/Articles/2.6-kernel-api/, http://lwn.net/Articles/183225/
- The TTY demystified
  http://www.linusakesson.net/programming/tty/index.php



Thanks for your patience!

Well Jim, it looks as clear as mud to me!