

# [PATCH 2.6.17-rc1] [SERIAL] DCC(JTAG) serial and the console emulation support(revised#3)

---

*Source:* <http://linux.derkeiler.com/Mailing-Lists/Kernel/2006-04/msg00047.html>

---

- *From:* "Hyok S. Choi" <[hyok.choi@xxxxxxxxxxx](mailto:hyok.choi@xxxxxxxxxxx)>
  - *Date:* Tue, 04 Apr 2006 11:19:56 +0900
- 

From: Hyok S. Choi <[hyok.choi@xxxxxxxxxxx](mailto:hyok.choi@xxxxxxxxxxx)>

Many of JTAG debuggers for ARM support DCC protocol over JTAG connection, which is very useful to debug hardwares which has no serial port. This patch adds DCC serial emulation and the console support. System timer based polling method is used for the emulation of serial input interrupt handling.

Signed-off-by: Hyok S. Choi <[hyok.choi@xxxxxxxxxxx](mailto:hyok.choi@xxxxxxxxxxx)>

---

```
drivers/serial/Kconfig | 31 +++
drivers/serial/Makefile | 1
drivers/serial/dcc.c | 467 ++++++++++++++++++++++++++++++++++++++
include/linux/serial_core.h | 3
4 files changed, 502 insertions(+), 0 deletions(-)
```

```
diff --git a/drivers/serial/Kconfig b/drivers/serial/Kconfig
index 7d22dc0..3c3a3f1 100644
--- a/drivers/serial/Kconfig
+++ b/drivers/serial/Kconfig
@@ -352,6 +352,37 @@ config SERIAL_CLPS711X_CONSOLE
your boot loader (lilo or loadlin) about how to pass options to the
kernel at boot time.)
```

```
+config SERIAL_DCC
+ bool "JTAG ICE/ICD DCC serial port emulation support"
+ depends on ARM
+ select SERIAL_CORE
+ help
+ This selects serial port emulation driver for ICE/ICD JTAG debugger
+ (e.g. Trace32) for ARM architecture. You should make an terminal with
+ DCC(JTAG1) protocol.
+
+ if unsure, say N.
+
+config SERIAL_DCC_CONSOLE
+ bool "Support for console on JTAG ICE/ICD DCC"
```

[PATCH 2.6.17-rc1] [SERIAL] DCC(JTAG) serial and the console emulation support(revised#3)

```
+ depends on SERIAL_DCC
+ select SERIAL_CORE_CONSOLE
+ help
+ Say Y here if you wish to use ICE/ICD JTAG DCC serial port emulation
+ as the system console.
+
+ if unsure, say N.
+
+config SERIAL_DCC_STDSERIAL
+ bool "Install JTAG ICE/ICD DCC as standard serial"
+ default y
+ depends on !SERIAL_8250 && SERIAL_DCC
+ help
+ Say Y here if you want to install DCC driver as a normal serial port
+ /dev/ttyS0 (major 4, minor 64). Otherwise, it appears as /dev/ttyJ0
+ (major 204, minor 186) and can co-exist with other UARTs, such as
+ 8250/16C550 compatibles.
+
config SERIAL_S3C2410
tristate "Samsung S3C2410 Serial port support"
depends on ARM && ARCH_S3C2410
diff --git a/drivers/serial/Makefile b/drivers/serial/Makefile
index 0a71bf6..63f2462 100644
--- a/drivers/serial/Makefile
+++ b/drivers/serial/Makefile
@@ -23,6 +23,7 @@ obj-$(CONFIG_SERIAL_8250_AU1X00) += 8250
obj-$(CONFIG_SERIAL_AMBA_PL010) += amba-pl010.o
obj-$(CONFIG_SERIAL_AMBA_PL011) += amba-pl011.o
obj-$(CONFIG_SERIAL_CLPS711X) += clps711x.o
+obj-$(CONFIG_SERIAL_DCC) += dcc.o
obj-$(CONFIG_SERIAL_PXA) += pxa.o
obj-$(CONFIG_SERIAL_SA1100) += sa1100.o
obj-$(CONFIG_SERIAL_S3C2410) += s3c2410.o
diff --git a/drivers/serial/dcc.c b/drivers/serial/dcc.c
new file mode 100644
index 0000000..ff7419b
--- /dev/null
+++ b/drivers/serial/dcc.c
@@ -0,0 +1,467 @@
+/*
+ * linux/drivers/serial/dcc.c
+ *
+ * serial port emulation driver for the JTAG DCC Terminal.
+ *
+ * DCC(JTAG1) protocol version for JTAG ICE/ICD Debuggers:
+ * Copyright (C) 2003, 2004, 2005 Hyok S. Choi (hyok.choi@xxxxxxxxxxxxx)
+ * SAMSUNG ELECTRONICS Co.,Ltd.
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License version 2 as
+ * published by the Free Software Foundation.
```

```
+ *
+ * Changelog:
+ * Oct-2003 Hyok S. Choi Created
+ * Feb-2004 Hyok S. Choi Updated for serial_core.c and 2.6 kernel
+ * Mar-2005 Hyok S. Choi renamed from T32 to DCC
+ * Apr-2006 Hyok S. Choi revised including the MAJOR number
+ *
+ */
+
+#include <linux/config.h>
+#include <linux/module.h>
+#include <linux/tty.h>
+#include <linux/ioport.h>
+#include <linux/init.h>
+#include <linux/serial.h>
+#include <linux/console.h>
+#include <linux/sysrq.h>
+#include <linux/tty_flip.h>
+#include <linux/major.h>
+
+#include <asm/io.h>
+#include <asm/irq.h>
+
+#include <linux/serial_core.h>
+
+/*
+ * if real irq interrupt used for receiving character,
+ * uncomment below line and modify the DCC_IRQ to correct one.
+ * Unless polling method id used.
+ */
+//#define DCC_IRQ_USED
+//#define DCC_IRQ (INT_N_EXT0)
+
+#ifndef DCC_IRQ_USED
+static struct work_struct dcc_poll_task;
+static void dcc_poll(void *data);
+/* need to schedule itself? (>=1:yes, 0:no) */
+static unsigned int dcc_poll_run = 0;
+#endif
+
+#define UART_NR 1 /* we have only one JTAG port */
+
+#ifdef CONFIG_SERIAL_DCC_STD SERIAL
+/* use ttyS for emulation of standard serial driver */
+#define SERIAL_DCC_NAME "ttyS"
+#define SERIAL_DCC_MAJOR TTY_MAJOR
+#define SERIAL_DCC_MINOR 64
+#else
+/* use ttyJ0(128) */
+#define SERIAL_DCC_NAME "ttyJ"
+#define SERIAL_DCC_MAJOR 204
```

```

+#define SERIAL_DCC_MINOR 186
+#endif
+
+/* DCC Status Bits */
+#define DCC_STATUS_RX (1 << 0)
+#define DCC_STATUS_TX (1 << 1)
+
+/* primitive JTAG1 protocol utilities */
+static inline u32 __dcc_getstatus(void)
+{
+ u32 ret;
+
+ asm("mrc p14, 0, %0, c0, c0 @ read comms ctrl reg"
+ : "=r" (ret));
+
+ return ret;
+}
+
+static inline char __dcc_getchar(void)
+{
+ char c;
+
+ asm("mrc p14, 0, %0, c1, c0 @ read comms data reg"
+ : "=r" (c));
+
+ return c;
+}
+
+static inline void __dcc_putchar(char c)
+{
+ asm("mcr p14, 0, %0, c1, c0 @ write a char"
+ : /* no output register */
+ : "r" (c));
+}
+/* end of JTAG1 dependencies */
+
+static void dcc_putchar(struct uart_port *port, int ch)
+{
+ while (__dcc_getstatus() & DCC_STATUS_TX)
+ cpu_relax();
+ __dcc_putchar((char)(ch & 0xFF));
+}
+
+static inline void xmit_string(struct uart_port *port, char *p, int len)
+{
+ for ( ; len; len--, p++)
+ dcc_putchar(port, *p);
+}
+
+static inline void dcc_transmit_buffer(struct uart_port *port)
+{

```

```

+ struct circ_buf *xmit = &port->info->xmit;
+ int pendings = uart_circ_chars_pending(xmit);
+
+ if (pendings + xmit->tail > UART_XMIT_SIZE) {
+ xmit_string(port, &(xmit->buf[xmit->tail]),
+ UART_XMIT_SIZE - xmit->tail);
+ xmit_string(port, &(xmit->buf[0]), xmit->head);
+ } else
+ xmit_string(port, &(xmit->buf[xmit->tail]), pendings);
+
+ xmit->tail = (xmit->tail + pendings) & (UART_XMIT_SIZE-1);
+ port->icount.tx += pendings;
+}
+
+static inline void dcc_transmit_x_char(struct uart_port *port)
+{
+ dcc_putchar(port, port->x_char);
+ port->icount.tx++;
+ port->x_char = 0;
+}
+
+static void dcc_start_tx(struct uart_port *port)
+{
+ dcc_transmit_buffer(port);
+}
+
+static inline void dcc_rx_chars(struct uart_port *port)
+{
+ unsigned char ch;
+ struct tty_struct *tty = port->info->tty;
+
+ /*
+ * check input.
+ * checking JTAG flag is better to resolve the status test.
+ * incout is NOT used for JTAG1 protocol.
+ */
+
+ if (__dcc_getstatus() & DCC_STATUS_RX) {
+ /* for JTAG 1 protocol, incout is always 1. */
+ ch = __dcc_getchar();
+
+ if (tty) {
+ tty_insert_flip_char(tty, ch, TTY_NORMAL);
+ port->icount.rx++;
+ tty_flip_buffer_push(tty);
+ }
+ }
+}
+
+static inline void dcc_tx_chars(struct uart_port *port)
+{

```

```

+ struct circ_buf *xmit = &port->info->xmit;
+
+ if (port->x_char) {
+   dcc_transmit_x_char(port);
+   return;
+ }
+
+ if (uart_circ_empty(xmit) || uart_tx_stopped(port))
+   return;
+
+   dcc_transmit_buffer(port);
+
+   if (uart_circ_chars_pending(xmit) < WAKEUP_CHARS)
+     uart_write_wakeup(port);
+ }
+
+ #ifdef DCC_IRQ_USED /* real IRQ used */
+ static irqreturn_t dcc_int(int irq, void *dev_id, struct pt_regs *regs)
+ {
+   struct uart_port *port = dev_id;
+   int handled = 0;
+
+   spin_lock(&port->lock);
+
+   dcc_rx_chars(port);
+   dcc_tx_chars(port);
+
+   handled = 1;
+   spin_unlock(&port->lock);
+
+   return IRQ_RETVAL(handled);
+ }
+ #else /* emulation by scheduled work */
+ static void dcc_poll(void *data)
+ {
+   struct uart_port *port = data;
+
+   spin_lock(&port->lock);
+
+   /*
+    * The dcc polling task rescheduling is controlled by
+    * dcc_startup and dcc_shutdown. To avoid the race condition,
+    * which comes from rescheduling by the task itself,
+    * dcc_poll_run is increased or decreased by the functions,
+    * and the rescheduling is stopped when comes to '0'.
+    */
+   if (dcc_poll_run == 0)
+     goto dcc_poll_out;
+
+   dcc_rx_chars(port);
+   dcc_tx_chars(port);

```



```

+static void dcc_set_termios(struct uart_port *port, struct termios *termios,
+ struct termios *old)
+{
+#ifdef DCC_IRQ_USED
+ unsigned long flags;
+#endif
+ unsigned int baud, quot;
+
+ /*
+ * We don't support parity, stop bits, or anything other
+ * than 8 bits, so clear these termios flags.
+ */
+ termios->c_cflag &= ~(CSIZE | CSTOPB | PARENB | PARODD | CREAD);
+ termios->c_cflag |= CS8;
+
+ /*
+ * We don't appear to support any error conditions either.
+ */
+ termios->c_iflag &= ~(INPCK | IGNPAR | IGNBRK | BRKINT);
+
+ /*
+ * Ask the core to calculate the divisor for us.
+ */
+ baud = uart_get_baud_rate(port, termios, old, 0, port->uartclk/16);
+ quot = uart_get_divisor(port, baud);
+
+#ifdef DCC_IRQ_USED
+ spin_lock_irqsave(&port->lock, flags);
+#endif
+
+ uart_update_timeout(port, termios->c_cflag, baud);
+
+#ifdef DCC_IRQ_USED
+ spin_unlock_irqrestore(&port->lock, flags);
+#endif
+}
+
+static const char * dcc_type(struct uart_port *port)
+{
+ return port->type == PORT_DCC_JTAG1 ? "DCC" : NULL;
+}
+
+static int dcc_request_port(struct uart_port *port)
+{
+ return 0;
+}
+
+ /*
+ * Configure/autoconfigure the port.
+ */
+static void dcc_config_port(struct uart_port *port, int flags)

```

```

+{
+ if (flags & UART_CONFIG_TYPE) {
+ port->type = PORT_DCC_JTAG1;
+ dcc_request_port(port);
+ }
+}
+
+/*
+ * verify the new serial_struct (for TIOCSSERIAL).
+ */
+static int dcc_verify_port(struct uart_port *port, struct serial_struct *ser)
+{
+ int ret = 0;
+ if (ser->type != PORT_UNKNOWN && ser->type != PORT_DCC_JTAG1)
+ ret = -EINVAL;
+ if (ser->irq < 0 || ser->irq >= NR_IRQS)
+ ret = -EINVAL;
+ if (ser->baud_base < 9600)
+ ret = -EINVAL;
+ return ret;
+}
+
+/* dummy operation handlers for uart_ops */
+static void dcc_dummy_ops(struct uart_port *port)
+{
+}
+static void dcc_dummy_ops_ui(struct uart_port *port, unsigned int ui)
+{
+}
+static void dcc_dummy_ops_i(struct uart_port *port, int i)
+{
+}
+
+static struct uart_ops dcc_pops = {
+ .tx_empty = dcc_tx_empty,
+ .set_mctrl = dcc_dummy_ops_ui,
+ .get_mctrl = dcc_get_mctrl,
+ .stop_tx = dcc_dummy_ops,
+ .start_tx = dcc_start_tx,
+ .stop_rx = dcc_dummy_ops,
+ .enable_ms = dcc_dummy_ops,
+ .break_ctl = dcc_dummy_ops_i,
+ .startup = dcc_startup,
+ .shutdown = dcc_shutdown,
+ .set_termios = dcc_set_termios,
+ .type = dcc_type,
+ .release_port = dcc_dummy_ops,
+ .request_port = dcc_request_port,
+ .config_port = dcc_config_port,
+ .verify_port = dcc_verify_port,
+};

```

```

+
+static struct uart_port dcc_port = {
+ .membase = (char*)0x12345678, /* we need these garbages */
+ .mapbase = 0x12345678, /* for serial_core.c */
+ .iotype = UPIO_MEM,
+ #ifdef DCC_IRQ_USED
+ .irq = DCC_IRQ,
+ #else
+ .irq = 0,
+ #endif
+ .uartclk = 14745600,
+ .fifosize = 0,
+ .ops = &dcc_pops,
+ .flags = UPF_BOOT_AUTOCONF,
+ .line = 0,
+ };
+
+
+ #ifdef CONFIG_SERIAL_DCC_CONSOLE
+ static void dcc_console_write(struct console *co, const char *s, unsigned int count)
+ {
+ uart_console_write(&dcc_port, s, count, dcc_putchar);
+ }
+
+ static int __init dcc_console_setup(struct console *co, char *options)
+ {
+ struct uart_port *port = &dcc_port;
+ int baud = 9600;
+ int bits = 8;
+ int parity = 'n';
+ int flow = 'n';
+
+ if (options)
+ uart_parse_options(options, &baud, &parity, &bits, &flow);
+
+ return uart_set_options(port, co, baud, parity, bits, flow);
+ }
+
+ extern struct uart_driver dcc_reg;
+ static struct console dcc_console = {
+ .name = SERIAL_DCC_NAME,
+ .write = dcc_console_write,
+ .device = uart_console_device,
+ .setup = dcc_console_setup,
+ .flags = CON_PRINTBUFFER,
+ .index = -1,
+ .data = &dcc_reg,
+ };
+
+ static int __init dcc_console_init(void)
+ {

```

```

+ register_console(&dcc_console);
+ return 0;
+}
+console_initcall(dcc_console_init);
+
+#define DCC_CONSOLE &dcc_console
+#else
+#define DCC_CONSOLE NULL
+#endif
+
+static struct uart_driver dcc_reg = {
+ .owner = THIS_MODULE,
+ .driver_name = SERIAL_DCC_NAME,
+ .dev_name = SERIAL_DCC_NAME,
+ .major = SERIAL_DCC_MAJOR,
+ .minor = SERIAL_DCC_MINOR,
+ .nr = UART_NR,
+ .cons = DCC_CONSOLE,
+};
+
+static int __init dcc_init(void)
+{
+ int ret;
+
+ printk(KERN_INFO "DCC: JTAG1 Serial emulation driver driver $Revision: 1.1 $\n");
+
+ ret = uart_register_driver(&dcc_reg);
+
+ if (ret)
+ return ret;
+
+#ifndef DCC_IRQ_USED
+ /* initialize the poll work */
+ INIT_WORK(&dcc_poll_task, dcc_poll, &dcc_port);
+#endif
+
+ uart_add_one_port(&dcc_reg, &dcc_port);
+
+ return 0;
+}
+
+__initcall(dcc_init);
+
+MODULE_DESCRIPTION("DCC(JTAG1) JTAG debugger console emulation driver");
+MODULE_AUTHOR("Hyok S. Choi <hyok.choi@xxxxxxxxxxxx>");
+MODULE_SUPPORTED_DEVICE(SERIAL_DCC_NAME);
+MODULE_LICENSE("GPL");
diff --git a/include/linux/serial_core.h b/include/linux/serial_core.h
index c32e60e..561bc6a 100644
--- a/include/linux/serial_core.h
+++ b/include/linux/serial_core.h

```

[PATCH 2.6.17-rc1] [SERIAL] DCC(JTAG) serial and the console emulation support(revised#3)

```
@@ -130,6 +130,9 @@
/* SUN4V Hypervisor Console */
#define PORT_SUNHV 72

+/* DCC(JTAG) emulation port types */
+#define PORT_DCC_JTAG1 73
+
#ifdef __KERNEL__

#include <linux/config.h>
```

—  
To unsubscribe from this list: send the line "unsubscribe linux-kernel" in  
the body of a message to majordomo@xxxxxxxxxxxxxxxxxxx  
More majordomo info at <http://vger.kernel.org/majordomo-info.html>  
Please read the FAQ at <http://www.tux.org/lkml/>