

# [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

**Source:** <http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-11/2252.html>

---

*dhowells\_at\_redhat.com*

**Date:** 11/08/04

Date: Mon, 8 Nov 2004 14:34:17 GMT  
To: torvalds@osdl.org, akpm@osdl.org, davidm@snapgear.com

The attached patches provides part 3 of an architecture implementation for the Fujitsu FR-V CPU series, configurably as Linux or uClinux.

Signed-Off-By: dhowells@redhat.com

---

```
diffstat frv-arch_3-2610rc1mm3.diff
 frv_ksyms.c      | 124 +++
 gdb-io.c        | 216 +++++
 gdb-io.h        | 55 +
 gdb-stub.c      | 2082 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 head-mmu-fr451.S | 374 ++++++++
 head.S          | 638 ++++++++
 head.inc        | 50 +
 7 files changed, 3539 insertions(+)
diff -uNrp /warthog/kernels/linux-2.6.10-rc1-mm3/arch/frv/kernel/frv_ksyms.c linux-2.6.10-rc1-mm3
--- /warthog/kernels/linux-2.6.10-rc1-mm3/arch/frv/kernel/frv_ksyms.c 1970-01-01 01:00:00.00000
+++ linux-2.6.10-rc1-mm3-frv/arch/frv/kernel/frv_ksyms.c 2004-11-05 14:13:03.088564605 +00
@@ -0,0 +1,124 @@
+#include <linux/module.h>
+#include <linux/linkage.h>
+#include <linux/sched.h>
+#include <linux/string.h>
+#include <linux/mm.h>
+#include <linux/user.h>
+#include <linux/elfcore.h>
+#include <linux/in6.h>
+#include <linux/interrupt.h>
+#include <linux/config.h>
+
+#include <asm/setup.h>
+#include <asm/pgalloc.h>
+#include <asm/irq.h>
+#include <asm/io.h>
+#include <asm/semaphore.h>
+#include <asm/checksum.h>
+#include <asm/hardirq.h>
+#include <asm/current.h>
+
+extern void dump_thread(struct pt_regs *, struct user *);
+extern long __memcpy_user(void *dst, const void *src, size_t count);
+
+/* platform dependent support */
```

```

+
+EXPORT_SYMBOL(__ioremap);
+EXPORT_SYMBOL(iounmap);
+
+EXPORT_SYMBOL(dump_thread);
+EXPORT_SYMBOL(strnlen);
+EXPORT_SYMBOL(strrchr);
+EXPORT_SYMBOL(strstr);
+EXPORT_SYMBOL(strchr);
+EXPORT_SYMBOL(strcat);
+EXPORT_SYMBOL(strlen);
+EXPORT_SYMBOL(strcmp);
+EXPORT_SYMBOL(strncmp);
+EXPORT_SYMBOL(strncpy);
+
+EXPORT_SYMBOL(ip_fast_csum);
+
+#if 0
+EXPORT_SYMBOL(local_irq_count);
+EXPORT_SYMBOL(local_bh_count);
+#endif
+EXPORT_SYMBOL(kernel_thread);
+
+EXPORT_SYMBOL(enable_irq);
+EXPORT_SYMBOL(disable_irq);
+EXPORT_SYMBOL(__res_bus_clock_speed_HZ);
+EXPORT_SYMBOL(__page_offset);
+EXPORT_SYMBOL(__memcpy_user);
+EXPORT_SYMBOL(flush_dcache_page);
+
+#ifndef CONFIG_MMU
+EXPORT_SYMBOL(memory_start);
+EXPORT_SYMBOL(memory_end);
+#endif
+
+EXPORT_SYMBOL_NOVERS(__debug_bug_trap);
+
+/* Networking helper routines. */
+EXPORT_SYMBOL(csum_partial_copy);
+
+/* The following are special because they're not called
+   explicitly (the C compiler generates them). Fortunately,
+   their interface isn't gonna change any time soon now, so
+   it's OK to leave it out of version control. */
+EXPORT_SYMBOL_NOVERS(memcpy);
+EXPORT_SYMBOL_NOVERS(memset);
+EXPORT_SYMBOL_NOVERS(memcmp);
+EXPORT_SYMBOL_NOVERS(memscan);
+EXPORT_SYMBOL_NOVERS(memmove);
+EXPORT_SYMBOL_NOVERS(strtok);
+
+EXPORT_SYMBOL(get_wchan);
+
+#ifdef CONFIG_FRV_OUTOFLINE_ATOMIC_OPS
+EXPORT_SYMBOL_NOVERS(atomic_test_and_ANDNOT_mask);
+EXPORT_SYMBOL_NOVERS(atomic_test_and_OR_mask);
+EXPORT_SYMBOL_NOVERS(atomic_test_and_XOR_mask);
+EXPORT_SYMBOL_NOVERS(atomic_add_return);
+EXPORT_SYMBOL_NOVERS(atomic_sub_return);
+EXPORT_SYMBOL_NOVERS(__xchg_8);
+EXPORT_SYMBOL_NOVERS(__xchg_16);
+EXPORT_SYMBOL_NOVERS(__xchg_32);

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+EXPORT_SYMBOL_NOVERS(__cmpxchg_8);
+EXPORT_SYMBOL_NOVERS(__cmpxchg_16);
+EXPORT_SYMBOL_NOVERS(__cmpxchg_32);
+#endif
+
+/*
+ * libgcc functions - functions that are used internally by the
+ * compiler... (prototypes are not correct though, but that
+ * doesn't really matter since they're not versioned).
+ */
+extern void __gcc_bcmp(void);
+extern void __ashldi3(void);
+extern void __ashrdi3(void);
+extern void __cmpdi2(void);
+extern void __divdi3(void);
+extern void __lshrdi3(void);
+extern void __moddi3(void);
+extern void __muldi3(void);
+extern void __negdi2(void);
+extern void __ucmpdi2(void);
+extern void __udivdi3(void);
+extern void __udivmoddi4(void);
+extern void __umoddi3(void);
+
+ /* gcc lib functions */
+//EXPORT_SYMBOL_NOVERS(__gcc_bcmp);
+EXPORT_SYMBOL_NOVERS(__ashldi3);
+EXPORT_SYMBOL_NOVERS(__ashrdi3);
+//EXPORT_SYMBOL_NOVERS(__cmpdi2);
+//EXPORT_SYMBOL_NOVERS(__divdi3);
+EXPORT_SYMBOL_NOVERS(__lshrdi3);
+//EXPORT_SYMBOL_NOVERS(__moddi3);
+EXPORT_SYMBOL_NOVERS(__muldi3);
+EXPORT_SYMBOL_NOVERS(__negdi2);
+//EXPORT_SYMBOL_NOVERS(__ucmpdi2);
+//EXPORT_SYMBOL_NOVERS(__udivdi3);
+//EXPORT_SYMBOL_NOVERS(__udivmoddi4);
+//EXPORT_SYMBOL_NOVERS(__umoddi3);
diff -uNrp /warthog/kernels/linux-2.6.10-rc1-mm3/arch/frv/kernel/gdb-io.c linux-2.6.10-rc1-mm3-fr
--- /warthog/kernels/linux-2.6.10-rc1-mm3/arch/frv/kernel/gdb-io.c      1970-01-01 01:00:00.00000
+++ linux-2.6.10-rc1-mm3-frv/arch/frv/kernel/gdb-io.c    2004-11-05 14:13:03.093564183 +0000
@@ -0,0 +1,216 @@
+/* gdb-io.c: FR403 GDB stub I/O
+ *
+ * Copyright (C) 2003 Red Hat, Inc. All Rights Reserved.
+ * Written by David Howells (dhowells@redhat.com)
+ *
+ * This program is free software; you can redistribute it and/or
+ * modify it under the terms of the GNU General Public License
+ * as published by the Free Software Foundation; either version
+ * 2 of the License, or (at your option) any later version.
+ */
+
+#include <linux/string.h>
+#include <linux/kernel.h>
+#include <linux/signal.h>
+#include <linux/sched.h>
+#include <linux/mm.h>
+#include <linux/console.h>
+#include <linux/init.h>
+#include <linux/serial_reg.h>
+

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

#include <asm/pgtable.h>
#include <asm/system.h>
#include <asm/irc-regs.h>
#include <asm/timer-regs.h>
#include <asm/gdb-stub.h>
#include "gdb-io.h"
+
#ifdef CONFIG_GDBSTUB_UART0
#define __UART(X) (*(volatile uint8_t *) (UART0_BASE + (UART_##X)))
#define __UART_IRR_NMI 0xff0f0000
#else /* CONFIG_GDBSTUB_UART1 */
#define __UART(X) (*(volatile uint8_t *) (UART1_BASE + (UART_##X)))
#define __UART_IRR_NMI 0xffff0000
#endif
+
#define LSR_WAIT_FOR(STATE)          \
do {                                  \
+   gdbstub_do_rx();                 \
} while (!(__UART(LSR) & UART_LSR_##STATE))
+
#define FLOWCTL_QUERY(LINE)          ({ __UART(MSR) & UART_MSR_##LINE; })
#define FLOWCTL_CLEAR(LINE)          do { __UART(MCR) &= ~UART_MCR_##LINE; mb(); } while (0)
#define FLOWCTL_SET(LINE)            do { __UART(MCR) |= UART_MCR_##LINE; mb(); } while (0)
+
#define FLOWCTL_WAIT_FOR(LINE)       \
do {                                  \
+   gdbstub_do_rx();                 \
} while(!FLOWCTL_QUERY(LINE))
+
+ /*****
+ /
+ * initialise the GDB stub
+ * - called with PSR.ET=0, so can't incur external interrupts
+ */
void gdbstub_io_init(void)
+{
+   /* set up the serial port */
+   __UART(LCR) = UART_LCR_WLEN8; /* 1N8 */
+   __UART(FCR) =
+       UART_FCR_ENABLE_FIFO |
+       UART_FCR_CLEAR_RCVR |
+       UART_FCR_CLEAR_XMIT |
+       UART_FCR_TRIGGER_1;
+
+   FLOWCTL_CLEAR(DTR);
+   FLOWCTL_SET(RTS);
+
+ //   gdbstub_set_baud(115200);
+
+   /* we want to get serial receive interrupts */
+   __UART(IER) = UART_IER_RDI | UART_IER_RLSI;
+   mb();
+
+   __set_IRR(6, __UART_IRR_NMI); /* map ERRs and UARTx to NMI */
+
+ } /* end gdbstub_io_init() */
+
+ /*****
+ /
+ * set up the GDB stub serial port baud rate timers
+ */
void gdbstub_set_baud(unsigned baud)

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+{
+    unsigned value, high, low;
+    u8 lcr;
+
+    /* work out the divisor to give us the nearest higher baud rate */
+    value = __serial_clock_speed_HZ / 16 / baud;
+
+    /* determine the baud rate range */
+    high = __serial_clock_speed_HZ / 16 / value;
+    low = __serial_clock_speed_HZ / 16 / (value + 1);
+
+    /* pick the nearest bound */
+    if (low + (high - low) / 2 > baud)
+        value++;
+
+    lcr = __UART(LCR);
+    __UART(LCR) |= UART_LCR_DLAB;
+    mb();
+    __UART(DLL) = value & 0xff;
+    __UART(DLM) = (value >> 8) & 0xff;
+    mb();
+    __UART(LCR) = lcr;
+    mb();
+
+} /* end gdbstub_set_baud() */
+
+/*
+ * receive characters into the receive FIFO
+ */
+void gdbstub_do_rx(void)
+{
+    unsigned ix, nix;
+
+    ix = gdbstub_rx_inp;
+
+    while (__UART(LSR) & UART_LSR_DR) {
+        nix = (ix + 2) & 0xfff;
+        if (nix == gdbstub_rx_outp)
+            break;
+
+        gdbstub_rx_buffer[ix++] = __UART(LSR);
+        gdbstub_rx_buffer[ix++] = __UART(RX);
+        ix = nix;
+    }
+
+    gdbstub_rx_inp = ix;
+
+    __clr_RC(15);
+    __clr_IRL();
+} /* end gdbstub_do_rx() */
+
+/*
+ * wait for a character to come from the debugger
+ */
+int gdbstub_rx_char(unsigned char *_ch, int nonblock)
+{
+    unsigned ix;
+    u8 ch, st;
+
+

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+     *_ch = 0xff;
+
+     if (gdbstub_rx_unget) {
+         *_ch = gdbstub_rx_unget;
+         gdbstub_rx_unget = 0;
+         return 0;
+     }
+
+ try_again:
+     gdbstub_do_rx();
+
+     /* pull chars out of the buffer */
+     ix = gdbstub_rx_outp;
+     if (ix == gdbstub_rx_inp) {
+         if (nonblock)
+             return -EAGAIN;
+         //watchdog_alert_counter = 0;
+         goto try_again;
+     }
+
+     st = gdbstub_rx_buffer[ix++];
+     ch = gdbstub_rx_buffer[ix++];
+     gdbstub_rx_outp = ix & 0x00000fff;
+
+     if (st & UART_LSR_BI) {
+         gdbstub_proto("### GDB Rx Break Detected ###\n");
+         return -EINTR;
+     }
+     else if (st & (UART_LSR_FE|UART_LSR_OE|UART_LSR_PE)) {
+         gdbstub_proto("### GDB Rx Error (st=%02x) ###\n",st);
+         return -EIO;
+     }
+     else {
+         gdbstub_proto("### GDB Rx %02x (st=%02x) ###\n",ch,st);
+         *_ch = ch & 0x7f;
+         return 0;
+     }
+
+ } /* end gdbstub_rx_char() */
+
+ /*****
+ */
+ * send a character to the debugger
+ */
+ void gdbstub_tx_char(unsigned char ch)
+ {
+     FLOWCTL_SET(DTR);
+     LSR_WAIT_FOR(THRE);
+     // FLOWCTL_WAIT_FOR(CTS);
+
+     if (ch == 0x0a) {
+         __UART(TX) = 0x0d;
+         mb();
+         LSR_WAIT_FOR(THRE);
+         // FLOWCTL_WAIT_FOR(CTS);
+     }
+     __UART(TX) = ch;
+     mb();
+
+     FLOWCTL_CLEAR(DTR);
+ } /* end gdbstub_tx_char() */
+

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+ /*****
+ */
+ * send a character to the debugger
+ */
+void gdbstub_tx_flush(void)
+{
+    LSR_WAIT_FOR(TEMT);
+    LSR_WAIT_FOR(THRE);
+    FLOWCTL_CLEAR(DTR);
+} /* end gdbstub_tx_flush() */
diff -uNrp /warthog/kernels/linux-2.6.10-rc1-mm3/arch/frv/kernel/gdb-io.h linux-2.6.10-rc1-mm3-fr
--- /warthog/kernels/linux-2.6.10-rc1-mm3/arch/frv/kernel/gdb-io.h      1970-01-01 01:00:00.00000
+++ linux-2.6.10-rc1-mm3-frv/arch/frv/kernel/gdb-io.h      2004-11-05 14:13:03.096563929 +0000
@@ -0,0 +1,55 @@
+/* gdb-io.h: FR403 GDB I/O port defs
+ *
+ * Copyright (C) 2003 Red Hat, Inc. All Rights Reserved.
+ * Written by David Howells (dhowells@redhat.com)
+ *
+ * This program is free software; you can redistribute it and/or
+ * modify it under the terms of the GNU General Public License
+ * as published by the Free Software Foundation; either version
+ * 2 of the License, or (at your option) any later version.
+ */
+
+#ifndef _GDB_IO_H
+#define _GDB_IO_H
+
+#include <asm/serial-regs.h>
+
+#undef UART_RX
+#undef UART_TX
+#undef UART_DLL
+#undef UART_DLM
+#undef UART_IER
+#undef UART_IIR
+#undef UART_FCR
+#undef UART_LCR
+#undef UART_MCR
+#undef UART_LSR
+#undef UART_MSR
+#undef UART_SCR
+
+#define UART_RX          0*8      /* In:  Receive buffer (DLAB=0) */
+#define UART_TX          0*8      /* Out: Transmit buffer (DLAB=0) */
+#define UART_DLL         0*8      /* Out: Divisor Latch Low (DLAB=1) */
+#define UART_DLM         1*8      /* Out: Divisor Latch High (DLAB=1) */
+#define UART_IER         1*8      /* Out: Interrupt Enable Register */
+#define UART_IIR         2*8      /* In:  Interrupt ID Register */
+#define UART_FCR         2*8      /* Out: FIFO Control Register */
+#define UART_LCR         3*8      /* Out: Line Control Register */
+#define UART_MCR         4*8      /* Out: Modem Control Register */
+#define UART_LSR         5*8      /* In:  Line Status Register */
+#define UART_MSR         6*8      /* In:  Modem Status Register */
+#define UART_SCR         7*8      /* I/O: Scratch Register */
+
+#define UART_LCR_DLAB    0x80     /* Divisor latch access bit */
+#define UART_LCR_SBC     0x40     /* Set break control */
+#define UART_LCR_SPAR    0x20     /* Stick parity (?) */
+#define UART_LCR_EPAR    0x10     /* Even parity select */
+#define UART_LCR_PARITY   0x08     /* Parity Enable */
+#define UART_LCR_STOP    0x04     /* Stop bits: 0=1 stop bit, 1= 2 stop bits */

```



## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```
+ * <checksum>      :: < two hex digits computed as modulo 256 sum of <packetinfo>>
+ *
+ * When a packet is received, it is first acknowledged with either '+' or '-'.
+ * '+' indicates a successful transfer. '-' indicates a failed transfer.
+ *
+ * Example:
+ *
+ * Host:              Reply:
+ * $m0,10#2a         +$00010203040506070809101112131415#42
+ *
+ * =====
+ * MORE EXAMPLES:
+ * =====
+ *
+ * For reference -- the following are the steps that one
+ * company took (RidgeRun Inc) to get remote gdb debugging
+ * going. In this scenario the host machine was a PC and the
+ * target platform was a Galileo EVB64120A MIPS evaluation
+ * board.
+ *
+ * Step 1:
+ * First download gdb-5.0.tar.gz from the internet.
+ * and then build/install the package.
+ *
+ * Example:
+ *   $ tar xzf gdb-5.0.tar.gz
+ *   $ cd gdb-5.0
+ *   $ ./configure --target=frv-elf-gdb
+ *   $ make
+ *   $ frv-elf-gdb
+ *
+ * Step 2:
+ * Configure linux for remote debugging and build it.
+ *
+ * Example:
+ *   $ cd ~/linux
+ *   $ make menuconfig <go to "Kernel Hacking" and turn on remote debugging>
+ *   $ make dep; make vmlinux
+ *
+ * Step 3:
+ * Download the kernel to the remote target and start
+ * the kernel running. It will promptly halt and wait
+ * for the host gdb session to connect. It does this
+ * since the "Kernel Hacking" option has defined
+ * CONFIG_REMOTE_DEBUG which in turn enables your calls
+ * to:
+ *   set_debug_traps();
+ *   breakpoint();
+ *
+ * Step 4:
+ * Start the gdb session on the host.
+ *
+ * Example:
+ *   $ frv-elf-gdb vmlinux
+ *   (gdb) set remotebaud 115200
+ *   (gdb) target remote /dev/ttyS1
+ *   ...at this point you are connected to
+ *   the remote target and can use gdb
+ *   in the normal fasion. Setting
+ *   breakpoints, single stepping,
+ *   printing variables, etc.
```

```

+ *
+ */
+
+#include <linux/string.h>
+#include <linux/kernel.h>
+#include <linux/signal.h>
+#include <linux/sched.h>
+#include <linux/mm.h>
+#include <linux/console.h>
+#include <linux/init.h>
+#include <linux/slab.h>
+#include <linux/nmi.h>
+
+#include <asm/pgtable.h>
+#include <asm/system.h>
+#include <asm/gdb-stub.h>
+
+#define LEDS(x) do { /* *(u32*)0xel200004 = ~(x); mb(); */ } while(0)
+
+#undef GDBSTUB_DEBUG_PROTOCOL
+
+extern void debug_to_serial(const char *p, int n);
+extern void gdbstub_console_write(struct console *co, const char *p, unsigned n);
+
+extern volatile uint32_t __break_error_detect[3]; /* ESR1, ESR15, EAR15 */
+extern struct user_context __break_user_context;
+
+struct __debug_amr {
+    unsigned long L, P;
+} __attribute__((aligned(8)));
+
+struct __debug_mmu {
+    struct {
+        unsigned long hsr0, pcsr, esr0, ear0, epcr0;
+#ifdef CONFIG_MMU
+        unsigned long tplr, tppr, tpxr, cxnr;
+#endif
+    } regs;
+
+    struct __debug_amr iamr[16];
+    struct __debug_amr damr[16];
+
+#ifdef CONFIG_MMU
+    struct __debug_amr tlb[64*2];
+#endif
+};
+
+static struct __debug_mmu __debug_mmu;
+
+/*
+ * BUFMAX defines the maximum number of characters in inbound/outbound buffers
+ * at least NUMREGBYTES*2 are needed for register packets
+ */
+#define BUFMAX 2048
+
+#define BREAK_INSN 0x801000c0 /* use "break" as bkpt */
+
+static const char gdbstub_banner[] = "Linux/FR-V GDB Stub (c) RedHat 2003\n";
+
+volatile u8 gdbstub_rx_buffer[PAGE_SIZE] __attribute__((aligned(PAGE_SIZE)));
+volatile u32 gdbstub_rx_inp = 0;
+volatile u32 gdbstub_rx_outp = 0;

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+volatile u8      gdbstub_rx_overflow = 0;
+u8              gdbstub_rx_unget = 0;
+
+/* set with GDB whilst running to permit step through exceptions */
+extern volatile u32 __attribute__((section(".bss"))) gdbstub_trace_through_exceptions;
+
+static char      input_buffer[BUFMAX];
+static char      output_buffer[BUFMAX];
+
+static const char hexchars[] = "0123456789abcdef";
+
+static const char *regnames[] = {
+    "PSR ", "ISR ", "CCR ", "CCCR ",
+    "LR  ", "LCR ", "PC  ", "_stt",
+    "sys ", "GR8*", "GNE0", "GNE1",
+    "IACH", "IACL",
+    "TBR ", "SP  ", "FP  ", "GR3  ",
+    "GR4  ", "GR5  ", "GR6  ", "GR7  ",
+    "GR8  ", "GR9  ", "GR10", "GR11",
+    "GR12", "GR13", "GR14", "GR15",
+    "GR16", "GR17", "GR18", "GR19",
+    "GR20", "GR21", "GR22", "GR23",
+    "GR24", "GR25", "GR26", "GR27",
+    "EFRM", "CURR", "GR30", "BFRM"
+};
+
+struct gdbstub_bkpt {
+    unsigned long   addr;           /* address of breakpoint */
+    unsigned        len;           /* size of breakpoint */
+    uint32_t        originsns[7]; /* original instructions */
+};
+
+static struct gdbstub_bkpt gdbstub_bkpts[256];
+
+/*
+ * local prototypes
+ */
+
+static void gdbstub_recv_packet(char *buffer);
+static int  gdbstub_send_packet(char *buffer);
+static int  gdbstub_compute_signal(unsigned long tbr);
+static int  hex(unsigned char ch);
+static int  hexToInt(char **ptr, unsigned long *intValue);
+static unsigned char *mem2hex(const void *mem, char *buf, int count, int may_fault);
+static char *hex2mem(const char *buf, void *_mem, int count);
+
+/*
+ * Convert ch from a hex digit to an int
+ */
+static int hex(unsigned char ch)
+{
+    if (ch >= 'a' && ch <= 'f')
+        return ch-'a'+10;
+    if (ch >= '0' && ch <= '9')
+        return ch-'0';
+    if (ch >= 'A' && ch <= 'F')
+        return ch-'A'+10;
+    return -1;
+}
+
+void gdbstub_printk(const char *fmt, ...)
+{

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+     static char buf[1024];
+     va_list args;
+     int len;
+
+     /* Emit the output into the temporary buffer */
+     va_start(args, fmt);
+     len = vsnprintf(buf, sizeof(buf), fmt, args);
+     va_end(args);
+     debug_to_serial(buf, len);
+ }
+
+static inline char *gdbstub_strcpy(char *dst, const char *src)
+{
+     int loop = 0;
+     while ((dst[loop] = src[loop]))
+         loop++;
+     return dst;
+}
+
+static void gdbstub_purge_cache(void)
+{
+     asm volatile(" dcef    @(gr0,gr0),#1    \n"
+                 " icei    @(gr0,gr0),#1    \n"
+                 " membar                               \n"
+                 " bar                               \n"
+                 );
+}
+
+/*
+ * scan for the sequence $<data>#<checksum>
+ */
+static void gdbstub_recv_packet(char *buffer)
+{
+     unsigned char checksum;
+     unsigned char xmitcsum;
+     unsigned char ch;
+     int count, i, ret, error;
+
+     for (;;) {
+         /* wait around for the start character, ignore all other characters */
+         do {
+             gdbstub_rx_char(&ch, 0);
+         } while (ch != '$');
+
+         checksum = 0;
+         xmitcsum = -1;
+         count = 0;
+         error = 0;
+
+         /* now, read until a # or end of buffer is found */
+         while (count < BUFMAX) {
+             ret = gdbstub_rx_char(&ch, 0);
+             if (ret < 0)
+                 error = ret;
+
+             if (ch == '#')
+                 break;
+             checksum += ch;
+             buffer[count] = ch;
+             count++;
+         }
+     }

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+
+         if (error == -EIO) {
+             gdbstub_proto("### GDB Rx Error - Skipping packet ###\n");
+             gdbstub_proto("### GDB Tx NAK\n");
+             gdbstub_tx_char('-');
+             continue;
+         }
+
+         if (count >= BUFMAX || error)
+             continue;
+
+         buffer[count] = 0;
+
+         /* read the checksum */
+         ret = gdbstub_rx_char(&ch, 0);
+         if (ret < 0)
+             error = ret;
+         xmitcsum = hex(ch) << 4;
+
+         ret = gdbstub_rx_char(&ch, 0);
+         if (ret < 0)
+             error = ret;
+         xmitcsum |= hex(ch);
+
+         if (error) {
+             if (error == -EIO)
+                 gdbstub_proto("### GDB Rx Error - Skipping packet\n");
+             gdbstub_proto("### GDB Tx NAK\n");
+             gdbstub_tx_char('-');
+             continue;
+         }
+
+         /* check the checksum */
+         if (checksum != xmitcsum) {
+             gdbstub_proto("### GDB Tx NAK\n");
+             gdbstub_tx_char('-'); /* failed checksum */
+             continue;
+         }
+
+         gdbstub_proto("### GDB Rx '$s#%02x' ###\n", buffer, checksum);
+         gdbstub_proto("### GDB Tx ACK\n");
+         gdbstub_tx_char('+'); /* successful transfer */
+
+         /* if a sequence char is present, reply the sequence ID */
+         if (buffer[2] == ':') {
+             gdbstub_tx_char(buffer[0]);
+             gdbstub_tx_char(buffer[1]);
+
+             /* remove sequence chars from buffer */
+             count = 0;
+             while (buffer[count]) count++;
+             for (i=3; i <= count; i++)
+                 buffer[i - 3] = buffer[i];
+         }
+
+         break;
+     }
+} /* end gdbstub_recv_packet() */
+
+/*
+*****
+*/
+ * send the packet in buffer.

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+ * - return 0 if successfully ACK'd
+ * - return 1 if abandoned due to new incoming packet
+ */
+static int gdbstub_send_packet(char *buffer)
+{
+    unsigned char checksum;
+    int count;
+    unsigned char ch;
+
+    /* $<packet info>#<checksum> */
+    gdbstub_proto("### GDB Tx '%s' ###\n", buffer);
+
+    do {
+        gdbstub_tx_char('$');
+        checksum = 0;
+        count = 0;
+
+        while ((ch = buffer[count]) != 0) {
+            gdbstub_tx_char(ch);
+            checksum += ch;
+            count += 1;
+        }
+
+        gdbstub_tx_char('#');
+        gdbstub_tx_char(hexchars[checksum >> 4]);
+        gdbstub_tx_char(hexchars[checksum & 0xf]);
+
+    } while (gdbstub_rx_char(&ch,0),
+#ifdef GDBSTUB_DEBUG_PROTOCOL
+        ch=='-' && (gdbstub_proto("### GDB Rx NAK\n"),0),
+        ch!='-' && ch!='+' && (gdbstub_proto("### GDB Rx ??? %02x\n",ch),0),
+#endif
+        ch!='+' && ch!='$');
+
+    if (ch=='+') {
+        gdbstub_proto("### GDB Rx ACK\n");
+        return 0;
+    }
+
+    gdbstub_proto("### GDB Tx Abandoned\n");
+    gdbstub_rx_unget = ch;
+    return 1;
+} /* end gdbstub_send_packet() */
+
+/*
+ * While we find nice hex chars, build an int.
+ * Return number of chars processed.
+ */
+static int hexToInt(char **ptr, unsigned long *_value)
+{
+    int count = 0, ch;
+
+    *_value = 0;
+    while (**ptr) {
+        ch = hex(**ptr);
+        if (ch < 0)
+            break;
+
+        *_value = (*_value << 4) | ((uint8_t) ch & 0xf);
+        count++;
+
+        (*ptr)++;
+    }

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+     }
+
+     return count;
+}
+
+/*
+ * probe an address to see whether it maps to anything
+ */
+static inline int gdbstub_addr_probe(const void *vaddr)
+{
+#ifdef CONFIG_MMU
+     unsigned long paddr;
+
+     asm("lrad %1,%0,#1,#0,#0" : "=r"(paddr) : "r"(vaddr));
+     if (!(paddr & xAMPRx_V))
+         return 0;
+#endif
+
+     return 1;
+} /* end gdbstub_addr_probe() */
+
+#ifdef CONFIG_MMU
+static unsigned long __saved_dampr, __saved_damlr;
+
+static inline unsigned long gdbstub_virt_to_pte(unsigned long vaddr)
+{
+     pgd_t *pgd;
+     pmd_t *pmd;
+     pte_t *pte;
+     unsigned long val, dampr5;
+
+     pgd = (pgd_t *) __get_DAMLR(3) + pgd_index(vaddr);
+     pmd = pmd_offset(pgd, vaddr);
+
+     if (pmd_bad(*pmd) || !pmd_present(*pmd))
+         return 0;
+
+     /* make sure dampr5 maps to the correct pmd */
+     dampr5 = __get_DAMPR(5);
+     val = pmd_val(*pmd);
+     __set_DAMPR(5, val | xAMPRx_L | xAMPRx_SS_16Kb | xAMPRx_S | xAMPRx_C | xAMPRx_V);
+
+     /* now its safe to access pmd */
+     pte = (pte_t *) __get_DAMLR(5) + __pte_index(vaddr);
+     if (pte_present(*pte))
+         val = pte_val(*pte);
+     else
+         val = 0;
+
+     /* restore original dampr5 */
+     __set_DAMPR(5, dampr5);
+
+     return val;
+}
+#endif
+
+static inline int gdbstub_addr_map(const void *vaddr)
+{
+#ifdef CONFIG_MMU
+     unsigned long pte;
+

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+     __saved_dampr = __get_DAMPR(2);
+     __saved_damlr = __get_DAMLR(2);
+ #endif
+     if (gdbstub_addr_probe(vaddr))
+         return 1;
+ #ifdef CONFIG_MMU
+     pte = gdbstub_virt_to_pte((unsigned long) vaddr);
+     if (pte) {
+         __set_DAMPR(2, pte);
+         __set_DAMLR(2, (unsigned long) vaddr & PAGE_MASK);
+         return 1;
+     }
+ #endif
+     return 0;
+ }
+
+ static inline void gdbstub_addr_unmap(void)
+ {
+ #ifdef CONFIG_MMU
+     __set_DAMPR(2, __saved_dampr);
+     __set_DAMLR(2, __saved_damlr);
+ #endif
+ }
+
+ /*
+  * access potentially dodgy memory through a potentially dodgy pointer
+  */
+ static inline int gdbstub_read_dword(const void *addr, uint32_t *_res)
+ {
+     unsigned long brr;
+     uint32_t res;
+
+     if (!gdbstub_addr_map(addr))
+         return 0;
+
+     asm volatile(" movgs   gr0,brr \n"
+                 " ld%I2   %M2,%0 \n"
+                 " movsg   brr,%1 \n"
+                 : "=r"(res), "=r"(brr)
+                 : "m"(*(uint32_t *) addr));
+
+     *_res = res;
+     gdbstub_addr_unmap();
+     return likely(!brr);
+ }
+
+ static inline int gdbstub_write_dword(void *addr, uint32_t val)
+ {
+     unsigned long brr;
+
+     if (!gdbstub_addr_map(addr))
+         return 0;
+
+     asm volatile(" movgs   gr0,brr \n"
+                 " st%I2   %1,%M2 \n"
+                 " movsg   brr,%0 \n"
+                 : "=r"(brr)
+                 : "r"(val), "m"(*(uint32_t *) addr));
+
+     gdbstub_addr_unmap();
+     return likely(!brr);
+ }
+
+ static inline int gdbstub_read_word(const void *addr, uint16_t *_res)

```

```

+{
+   unsigned long brr;
+   uint16_t res;
+
+   if (!gdbstub_addr_map(addr))
+       return 0;
+
+   asm volatile("  movgs   gr0,brr \n"
+                "  lduh%I2 %M2,%0 \n"
+                "  movsg   brr,%1 \n"
+                : "=r"(res), "=r"(brr)
+                : "m"(*(uint16_t *) addr));
+   *_res = res;
+   gdbstub_addr_unmap();
+   return likely(!brr);
+}
+
+static inline int gdbstub_write_word(void *addr, uint16_t val)
+{
+   unsigned long brr;
+
+   if (!gdbstub_addr_map(addr))
+       return 0;
+
+   asm volatile("  movgs   gr0,brr \n"
+                "  sth%I2  %1,%M2 \n"
+                "  movsg   brr,%0 \n"
+                : "=r"(brr)
+                : "r"(val), "m"(*(uint16_t *) addr));
+   gdbstub_addr_unmap();
+   return likely(!brr);
+}
+
+static inline int gdbstub_read_byte(const void *addr, uint8_t *_res)
+{
+   unsigned long brr;
+   uint8_t res;
+
+   if (!gdbstub_addr_map(addr))
+       return 0;
+
+   asm volatile("  movgs   gr0,brr \n"
+                "  ldub%I2 %M2,%0 \n"
+                "  movsg   brr,%1 \n"
+                : "=r"(res), "=r"(brr)
+                : "m"(*(uint8_t *) addr));
+   *_res = res;
+   gdbstub_addr_unmap();
+   return likely(!brr);
+}
+
+static inline int gdbstub_write_byte(void *addr, uint8_t val)
+{
+   unsigned long brr;
+
+   if (!gdbstub_addr_map(addr))
+       return 0;
+
+   asm volatile("  movgs   gr0,brr \n"
+                "  stb%I2  %1,%M2 \n"
+                "  movsg   brr,%0 \n"
+                : "=r"(brr)

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+         : "r"(val), "m"(*(uint8_t *) addr));
+     gdbstub_addr_unmap();
+     return likely(!brr);
+ }
+
+static void __gdbstub_console_write(struct console *co, const char *p, unsigned n)
+{
+     char outbuf[26];
+     int qty;
+
+     outbuf[0] = '0';
+
+     while (n > 0) {
+         qty = 1;
+
+         while (n > 0 && qty < 20) {
+             mem2hex(p, outbuf + qty, 2, 0);
+             qty += 2;
+             if (*p == 0x0a) {
+                 outbuf[qty++] = '0';
+                 outbuf[qty++] = 'd';
+             }
+             p++;
+             n--;
+         }
+
+         outbuf[qty] = 0;
+         gdbstub_send_packet(outbuf);
+     }
+ }
+
+#if 0
+void debug_to_serial(const char *p, int n)
+{
+     gdbstub_console_write(NULL,p,n);
+ }
+#endif
+
+#ifdef CONFIG_GDBSTUB_CONSOLE
+
+static kdev_t gdbstub_console_dev(struct console *con)
+{
+     return MKDEV(1,3); /* /dev/null */
+ }
+
+static struct console gdbstub_console = {
+     .name     = "gdb",
+     .write    = gdbstub_console_write,          /* in break.S */
+     .device   = gdbstub_console_dev,
+     .flags    = CON_PRINTBUFFER,
+     .index    = -1,
+ };
+
+#endif
+
+/*
+ * Convert the memory pointed to by mem into hex, placing result in buf.
+ * - if successful, return a pointer to the last char put in buf (NUL)
+ * - in case of mem fault, return NULL
+ * may_fault is non-zero if we are reading from arbitrary memory, but is currently
+ * not used.

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+ */
+static unsigned char *mem2hex(const void *_mem, char *buf, int count, int may_fault)
+{
+    const uint8_t *mem = _mem;
+    uint8_t ch[4] __attribute__((aligned(4)));
+
+    if ((uint32_t)mem&1 && count>=1) {
+        if (!gdbstub_read_byte(mem,ch))
+            return NULL;
+        *buf++ = hexchars[ch[0] >> 4];
+        *buf++ = hexchars[ch[0] & 0xf];
+        mem++;
+        count--;
+    }
+
+    if ((uint32_t)mem&3 && count>=2) {
+        if (!gdbstub_read_word(mem,(uint16_t *)ch))
+            return NULL;
+        *buf++ = hexchars[ch[0] >> 4];
+        *buf++ = hexchars[ch[0] & 0xf];
+        *buf++ = hexchars[ch[1] >> 4];
+        *buf++ = hexchars[ch[1] & 0xf];
+        mem += 2;
+        count -= 2;
+    }
+
+    while (count>=4) {
+        if (!gdbstub_read_dword(mem,(uint32_t *)ch))
+            return NULL;
+        *buf++ = hexchars[ch[0] >> 4];
+        *buf++ = hexchars[ch[0] & 0xf];
+        *buf++ = hexchars[ch[1] >> 4];
+        *buf++ = hexchars[ch[1] & 0xf];
+        *buf++ = hexchars[ch[2] >> 4];
+        *buf++ = hexchars[ch[2] & 0xf];
+        *buf++ = hexchars[ch[3] >> 4];
+        *buf++ = hexchars[ch[3] & 0xf];
+        mem += 4;
+        count -= 4;
+    }
+
+    if (count>=2) {
+        if (!gdbstub_read_word(mem,(uint16_t *)ch))
+            return NULL;
+        *buf++ = hexchars[ch[0] >> 4];
+        *buf++ = hexchars[ch[0] & 0xf];
+        *buf++ = hexchars[ch[1] >> 4];
+        *buf++ = hexchars[ch[1] & 0xf];
+        mem += 2;
+        count -= 2;
+    }
+
+    if (count>=1) {
+        if (!gdbstub_read_byte(mem,ch))
+            return NULL;
+        *buf++ = hexchars[ch[0] >> 4];
+        *buf++ = hexchars[ch[0] & 0xf];
+    }
+
+    *buf = 0;
+
+    return buf;

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+} /* end mem2hex() */
+
+/*
+*****
+
+ * convert the hex array pointed to by buf into binary to be placed in mem
+ * return a pointer to the character AFTER the last byte of buffer consumed
+ */
+static char *hex2mem(const char *buf, void *_mem, int count)
+{
+    uint8_t *mem = _mem;
+    union {
+        uint32_t l;
+        uint16_t w;
+        uint8_t b[4];
+    } ch;
+
+    if ((u32)mem&1 && count>=1) {
+        ch.b[0] = hex(*buf++) << 4;
+        ch.b[0] |= hex(*buf++);
+        if (!gdbstub_write_byte(mem,ch.b[0]))
+            return NULL;
+        mem++;
+        count--;
+    }
+
+    if ((u32)mem&3 && count>=2) {
+        ch.b[0] = hex(*buf++) << 4;
+        ch.b[0] |= hex(*buf++);
+        ch.b[1] = hex(*buf++) << 4;
+        ch.b[1] |= hex(*buf++);
+        if (!gdbstub_write_word(mem,ch.w))
+            return NULL;
+        mem += 2;
+        count -= 2;
+    }
+
+    while (count>=4) {
+        ch.b[0] = hex(*buf++) << 4;
+        ch.b[0] |= hex(*buf++);
+        ch.b[1] = hex(*buf++) << 4;
+        ch.b[1] |= hex(*buf++);
+        ch.b[2] = hex(*buf++) << 4;
+        ch.b[2] |= hex(*buf++);
+        ch.b[3] = hex(*buf++) << 4;
+        ch.b[3] |= hex(*buf++);
+        if (!gdbstub_write_dword(mem,ch.l))
+            return NULL;
+        mem += 4;
+        count -= 4;
+    }
+
+    if (count>=2) {
+        ch.b[0] = hex(*buf++) << 4;
+        ch.b[0] |= hex(*buf++);
+        ch.b[1] = hex(*buf++) << 4;
+        ch.b[1] |= hex(*buf++);
+        if (!gdbstub_write_word(mem,ch.w))
+            return NULL;
+        mem += 2;
+        count -= 2;
+    }
+
+}

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+     if (count>=1) {
+         ch.b[0] = hex(*buf++) << 4;
+         ch.b[0] |= hex(*buf++);
+         if (!gdbstub_write_byte(mem,ch.b[0]))
+             return NULL;
+     }
+
+     return (char *) buf;
+} /* end hex2mem() */
+
+/*
+ * This table contains the mapping between FRV TBR.TT exception codes,
+ * and signals, which are primarily what GDB understands. It also
+ * indicates which hardware traps we need to commandeer when
+ * initializing the stub.
+ */
+static const struct brr_to_sig_map {
+    unsigned long   brr_mask;      /* BRR bitmask */
+    unsigned long   tbr_tt;        /* TBR.TT code (in BRR.EBTT) */
+    unsigned int    signo;         /* Signal that we map this into */
+} brr_to_sig_map[] = {
+    { BRR_EB,      TBR_TT_INSTR_ACC_ERROR, SIGSEGV      },
+    { BRR_EB,      TBR_TT_ILLEGAL_INSTR,  SIGILL        },
+    { BRR_EB,      TBR_TT_PRIV_INSTR,     SIGILL        },
+    { BRR_EB,      TBR_TT_MP_EXCEPTION,   SIGFPE       },
+    { BRR_EB,      TBR_TT_DATA_ACC_ERROR, SIGSEGV      },
+    { BRR_EB,      TBR_TT_DATA_STR_ERROR, SIGSEGV      },
+    { BRR_EB,      TBR_TT_DIVISION_EXCEP, SIGFPE       },
+    { BRR_EB,      TBR_TT_COMPOUND_EXCEP, SIGSEGV      },
+    { BRR_EB,      TBR_TT_INTERRUPT_13,   SIGALRM      }, /* watchdog */
+    { BRR_EB,      TBR_TT_INTERRUPT_14,   SIGINT       }, /* GDB serial */
+    { BRR_EB,      TBR_TT_INTERRUPT_15,   SIGQUIT      }, /* NMI */
+    { BRR_CB,      0,                      SIGUSR1      },
+    { BRR_TB,      0,                      SIGUSR2      },
+    { BRR_DBNEx, 0,                      SIGTRAP      },
+    { BRR_DBx,   0,                      SIGTRAP      }, /* h/w watchpoint */
+    { BRR_IBx,   0,                      SIGTRAP      }, /* h/w breakpoint */
+    { BRR_CBB,     0,                      SIGTRAP      },
+    { BRR_SB,      0,                      SIGTRAP      },
+    { BRR_ST,      0,                      SIGTRAP      }, /* single step */
+    { 0,           0,                      SIGHUP       }, /* default */
+};
+
+/*
+ * convert the FRV BRR register contents into a UNIX signal number
+ */
+static inline int gdbstub_compute_signal(unsigned long brr)
+{
+    const struct brr_to_sig_map *map;
+    unsigned long tbr = (brr & BRR_EBTT) >> 12;
+
+    for (map = brr_to_sig_map; map->brr_mask; map++)
+        if (map->brr_mask & brr)
+            if (!map->tbr_tt || map->tbr_tt == tbr)
+                break;
+
+    return map->signo;
+} /* end gdbstub_compute_signal() */
+
+/*

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+/*
+ * set a software breakpoint or a hardware breakpoint or watchpoint
+ */
+static int gdbstub_set_breakpoint(unsigned long type, unsigned long addr, unsigned long len)
+{
+    unsigned long tmp;
+    int bkpt, loop, xloop;
+
+    union {
+        struct {
+            unsigned long mask0, mask1;
+        };
+        uint8_t bytes[8];
+    } dbmr;
+
+    //gdbstub_printk("setbkpt(%ld,%08lx,%ld)\n", type, addr, len);
+
+    switch (type) {
+        /* set software breakpoint */
+        case 0:
+            if (addr & 3 || len > 7*4)
+                return -EINVAL;
+
+            for (bkpt = 255; bkpt >= 0; bkpt--)
+                if (!gdbstub_bkpts[bkpt].addr)
+                    break;
+
+            if (bkpt < 0)
+                return -ENOSPC;
+
+            for (loop = 0; loop < len/4; loop++)
+                if (!gdbstub_read_dword(&((uint32_t *) addr)[loop],
+                                        &gdbstub_bkpts[bkpt].originsns[loop]))
+                    return -EFAULT;
+
+            for (loop = 0; loop < len/4; loop++)
+                if (!gdbstub_write_dword(&((uint32_t *) addr)[loop],
+                                        BREAK_INSN)
+                    ) {
+                /* need to undo the changes if possible */
+                for (xloop = 0; xloop < loop; xloop++)
+                    gdbstub_write_dword(&((uint32_t *) addr)[xloop],
+                                        gdbstub_bkpts[bkpt].originsns[xloop]);
+                return -EFAULT;
+            }
+
+            gdbstub_bkpts[bkpt].addr = addr;
+            gdbstub_bkpts[bkpt].len = len;
+
+        #if 0
+            gdbstub_printk("Set BKPT[%02x]: %08lx #%d {%04x, %04x} -> { %04x, %04x }\n",
+                bkpt,
+                gdbstub_bkpts[bkpt].addr,
+                gdbstub_bkpts[bkpt].len,
+                gdbstub_bkpts[bkpt].originsns[0],
+                gdbstub_bkpts[bkpt].originsns[1],
+                ((uint32_t *) addr)[0],
+                ((uint32_t *) addr)[1]
+            );
+        #endif
+
+        return 0;
+
+        /* set hardware breakpoint */

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+ case 1:
+     if (addr & 3 || len != 4)
+         return -EINVAL;
+
+     if (!(__debug_regs->dcr & DCR_IBE0)) {
+         //gdbstub_printk("set h/w break 0: %08lx\n", addr);
+         __debug_regs->dcr |= DCR_IBE0;
+         asm volatile("movgs %0,ibar0" : : "r"(addr));
+         return 0;
+     }
+
+     if (!(__debug_regs->dcr & DCR_IBE1)) {
+         //gdbstub_printk("set h/w break 1: %08lx\n", addr);
+         __debug_regs->dcr |= DCR_IBE1;
+         asm volatile("movgs %0,ibar1" : : "r"(addr));
+         return 0;
+     }
+
+     if (!(__debug_regs->dcr & DCR_IBE2)) {
+         //gdbstub_printk("set h/w break 2: %08lx\n", addr);
+         __debug_regs->dcr |= DCR_IBE2;
+         asm volatile("movgs %0,ibar2" : : "r"(addr));
+         return 0;
+     }
+
+     if (!(__debug_regs->dcr & DCR_IBE3)) {
+         //gdbstub_printk("set h/w break 3: %08lx\n", addr);
+         __debug_regs->dcr |= DCR_IBE3;
+         asm volatile("movgs %0,ibar3" : : "r"(addr));
+         return 0;
+     }
+
+     return -ENOSPC;
+
+     /* set data read/write/access watchpoint */
+ case 2:
+ case 3:
+ case 4:
+     if ((addr & ~7) != ((addr + len - 1) & ~7))
+         return -EINVAL;
+
+     tmp = addr & 7;
+
+     memset(dbmr.bytes, 0xff, sizeof(dbmr.bytes));
+     for (loop = 0; loop < len; loop++)
+         dbmr.bytes[tmp + loop] = 0;
+
+     addr &= ~7;
+
+     if (!(__debug_regs->dcr & (DCR_DRBE0|DCR_DWBE0))) {
+         //gdbstub_printk("set h/w watchpoint 0 type %ld: %08lx\n", type, addr);
+         tmp = type==2 ? DCR_DWBE0 : type==3 ? DCR_DRBE0 : DCR_DRBE0|DCR_DWBE0;
+         __debug_regs->dcr |= tmp;
+         asm volatile(" movgs  %0,dbar0      \n"
+                    " movgs  %1,dbmr00    \n"
+                    " movgs  %2,dbmr01    \n"
+                    " movgs  gr0,dbdr00   \n"
+                    " movgs  gr0,dbdr01   \n"
+                    : : "r"(addr), "r"(dbmr.mask0), "r"(dbmr.mask1));
+         return 0;
+     }
+
+
+

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+         if (!(__debug_regs->dcr & (DCR_DRBE1|DCR_DWBE1))) {
+             //gdbstub_printk("set h/w watchpoint 1 type %ld: %08lx\n", type, addr);
+             tmp = type==2 ? DCR_DWBE1 : type==3 ? DCR_DRBE1 : DCR_DRBE1|DCR_DWBE1;
+             __debug_regs->dcr |= tmp;
+             asm volatile(" movgs    %0,dbar1        \n"
+                 " movgs    %1,dbmr10        \n"
+                 " movgs    %2,dbmr11        \n"
+                 " movgs    gr0,dbdr10        \n"
+                 " movgs    gr0,dbdr11        \n"
+                 : : "r"(addr), "r"(dbmr.mask0), "r"(dbmr.mask1));
+             return 0;
+         }
+
+         return -ENOSPC;
+
+     default:
+         return -EINVAL;
+     }
+} /* end gdbstub_set_breakpoint() */
+
+/*
+ * clear a breakpoint or watchpoint
+ */
+int gdbstub_clear_breakpoint(unsigned long type, unsigned long addr, unsigned long len)
+{
+     unsigned long tmp;
+     int bkpt, loop;
+
+     union {
+         struct {
+             unsigned long mask0, mask1;
+         };
+         uint8_t bytes[8];
+     } dbmr;
+
+     //gdbstub_printk("clearbkpt(%ld,%08lx,%ld)\n", type, addr, len);
+
+     switch (type) {
+         /* clear software breakpoint */
+     case 0:
+         for (bkpt = 255; bkpt >= 0; bkpt--)
+             if (gdbstub_bkpts[bkpt].addr == addr && gdbstub_bkpts[bkpt].len == len)
+                 break;
+
+         if (bkpt < 0)
+             return -ENOENT;
+
+         gdbstub_bkpts[bkpt].addr = 0;
+
+         for (loop = 0; loop < len/4; loop++)
+             if (!gdbstub_write_dword(&((uint32_t *) addr)[loop],
+                                     gdbstub_bkpts[bkpt].originsns[loop]))
+                 return -EFAULT;
+
+         return 0;
+
+         /* clear hardware breakpoint */
+     case 1:
+         if (addr & 3 || len != 4)
+             return -EINVAL;
+
+         return 0;
+
+     }
+}
+
+#define __get_ibar(X) ({ unsigned long x; asm volatile("movsg ibar"#X,"%0" : "=r"(x)); x; })

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+
+     if (__debug_regs->dcr & DCR_IBE0 && __get_ibar(0) == addr) {
+         //gdbstub_printk("clear h/w break 0: %08lx\n", addr);
+         __debug_regs->dcr &= ~DCR_IBE0;
+         asm volatile("movgs gr0,ibar0");
+         return 0;
+     }
+
+     if (__debug_regs->dcr & DCR_IBE1 && __get_ibar(1) == addr) {
+         //gdbstub_printk("clear h/w break 1: %08lx\n", addr);
+         __debug_regs->dcr &= ~DCR_IBE1;
+         asm volatile("movgs gr0,ibar1");
+         return 0;
+     }
+
+     if (__debug_regs->dcr & DCR_IBE2 && __get_ibar(2) == addr) {
+         //gdbstub_printk("clear h/w break 2: %08lx\n", addr);
+         __debug_regs->dcr &= ~DCR_IBE2;
+         asm volatile("movgs gr0,ibar2");
+         return 0;
+     }
+
+     if (__debug_regs->dcr & DCR_IBE3 && __get_ibar(3) == addr) {
+         //gdbstub_printk("clear h/w break 3: %08lx\n", addr);
+         __debug_regs->dcr &= ~DCR_IBE3;
+         asm volatile("movgs gr0,ibar3");
+         return 0;
+     }
+
+     return -EINVAL;
+
+     /* clear data read/write/access watchpoint */
+ case 2:
+ case 3:
+ case 4:
+     if ((addr & ~7) != ((addr + len - 1) & ~7))
+         return -EINVAL;
+
+     tmp = addr & 7;
+
+     memset(dbmr.bytes, 0xff, sizeof(dbmr.bytes));
+     for (loop = 0; loop < len; loop++)
+         dbmr.bytes[tmp + loop] = 0;
+
+     addr &= ~7;
+
+ #define __get_dbar(X) ({ unsigned long x; asm volatile("movsg dbar"#X",%0" : "=r"(x)); x; })
+ #define __get_dbmr0(X) ({ unsigned long x; asm volatile("movsg dbmr"#X"0,%0" : "=r"(x)); x; })
+ #define __get_dbmr1(X) ({ unsigned long x; asm volatile("movsg dbmr"#X"1,%0" : "=r"(x)); x; })
+
+     /* consider DBAR 0 */
+     tmp = type==2 ? DCR_DWBE0 : type==3 ? DCR_DRBE0 : DCR_DRBE0|DCR_DWBE0;
+
+     if ((__debug_regs->dcr & (DCR_DRBE0|DCR_DWBE0)) != tmp ||
+         __get_dbar(0) != addr ||
+         __get_dbmr0(0) != dbmr.mask0 ||
+         __get_dbmr1(0) != dbmr.mask1)
+         goto skip_dbar0;
+
+     //gdbstub_printk("clear h/w watchpoint 0 type %ld: %08lx\n", type, addr);
+     __debug_regs->dcr &= ~(DCR_DRBE0|DCR_DWBE0);
+     asm volatile(" movgs    gr0,dbar0        \n"

```



## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+
+   gdbstub_printk("Frame: @%p [%s]\n",
+                 __debug_frame,
+                 __debug_frame->psr & PSR_S ? "kernel" : "user");
+
+   reg = (uint32_t *) __debug_frame;
+   for (loop = 0; loop < REG__END; loop++) {
+       printk("%s %08x", regnames[loop + 0], reg[loop + 0]);
+
+       if (loop == REG__END - 1 || loop % 5 == 4)
+           printk("\n");
+       else
+           printk(" | ");
+   }
+
+   gdbstub_printk("Process %s (pid: %d)\n", current->comm, current->pid);
+} /* end gdbstub_show_regs() */
+
+/*
+*****
+*/
+ * dump debugging regs
+ */
+static void __attribute__((unused)) gdbstub_dump_debugregs(void)
+{
+   unsigned long x;
+
+   x = __debug_regs->dcr;
+   gdbstub_printk("DCR   %08lx  ", x);
+
+   x = __debug_regs->brr;
+   gdbstub_printk("BRR %08lx\n", x);
+
+   gdbstub_printk("IBAR0 %08lx  ", __get_ibar(0));
+   gdbstub_printk("IBAR1 %08lx  ", __get_ibar(1));
+   gdbstub_printk("IBAR2 %08lx  ", __get_ibar(2));
+   gdbstub_printk("IBAR3 %08lx\n", __get_ibar(3));
+
+   gdbstub_printk("DBAR0 %08lx  ", __get_dbar(0));
+   gdbstub_printk("DBMR00 %08lx  ", __get_dbmr0(0));
+   gdbstub_printk("DBMR01 %08lx\n", __get_dbmr1(0));
+
+   gdbstub_printk("DBAR1 %08lx  ", __get_dbar(1));
+   gdbstub_printk("DBMR10 %08lx  ", __get_dbmr0(1));
+   gdbstub_printk("DBMR11 %08lx\n", __get_dbmr1(1));
+
+   gdbstub_printk("\n");
+} /* end gdbstub_dump_debugregs() */
+
+/*
+*****
+*/
+ * dump the MMU state into a structure so that it can be accessed with GDB
+ */
+void gdbstub_get_mmu_state(void)
+{
+   asm volatile("movsg hsr0,%0" : "=r"(__debug_mmu.regs.hsr0));
+   asm volatile("movsg pcsr,%0" : "=r"(__debug_mmu.regs.pcsr));
+   asm volatile("movsg esr0,%0" : "=r"(__debug_mmu.regs.esr0));
+   asm volatile("movsg ear0,%0" : "=r"(__debug_mmu.regs.ear0));
+   asm volatile("movsg epcr0,%0" : "=r"(__debug_mmu.regs.epcr0));
+
+   /* read the protection / SAT registers */
+   __debug_mmu.iamr[0].L = __get_IAMLR(0);

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```
+ __debug_mmu.iamr[0].P = __get_IAMPR(0);
+ __debug_mmu.iamr[1].L = __get_IAMLR(1);
+ __debug_mmu.iamr[1].P = __get_IAMPR(1);
+ __debug_mmu.iamr[2].L = __get_IAMLR(2);
+ __debug_mmu.iamr[2].P = __get_IAMPR(2);
+ __debug_mmu.iamr[3].L = __get_IAMLR(3);
+ __debug_mmu.iamr[3].P = __get_IAMPR(3);
+ __debug_mmu.iamr[4].L = __get_IAMLR(4);
+ __debug_mmu.iamr[4].P = __get_IAMPR(4);
+ __debug_mmu.iamr[5].L = __get_IAMLR(5);
+ __debug_mmu.iamr[5].P = __get_IAMPR(5);
+ __debug_mmu.iamr[6].L = __get_IAMLR(6);
+ __debug_mmu.iamr[6].P = __get_IAMPR(6);
+ __debug_mmu.iamr[7].L = __get_IAMLR(7);
+ __debug_mmu.iamr[7].P = __get_IAMPR(7);
+ __debug_mmu.iamr[8].L = __get_IAMLR(8);
+ __debug_mmu.iamr[8].P = __get_IAMPR(8);
+ __debug_mmu.iamr[9].L = __get_IAMLR(9);
+ __debug_mmu.iamr[9].P = __get_IAMPR(9);
+ __debug_mmu.iamr[10].L = __get_IAMLR(10);
+ __debug_mmu.iamr[10].P = __get_IAMPR(10);
+ __debug_mmu.iamr[11].L = __get_IAMLR(11);
+ __debug_mmu.iamr[11].P = __get_IAMPR(11);
+ __debug_mmu.iamr[12].L = __get_IAMLR(12);
+ __debug_mmu.iamr[12].P = __get_IAMPR(12);
+ __debug_mmu.iamr[13].L = __get_IAMLR(13);
+ __debug_mmu.iamr[13].P = __get_IAMPR(13);
+ __debug_mmu.iamr[14].L = __get_IAMLR(14);
+ __debug_mmu.iamr[14].P = __get_IAMPR(14);
+ __debug_mmu.iamr[15].L = __get_IAMLR(15);
+ __debug_mmu.iamr[15].P = __get_IAMPR(15);
+
+ __debug_mmu.damr[0].L = __get_DAMLR(0);
+ __debug_mmu.damr[0].P = __get_DAMPR(0);
+ __debug_mmu.damr[1].L = __get_DAMLR(1);
+ __debug_mmu.damr[1].P = __get_DAMPR(1);
+ __debug_mmu.damr[2].L = __get_DAMLR(2);
+ __debug_mmu.damr[2].P = __get_DAMPR(2);
+ __debug_mmu.damr[3].L = __get_DAMLR(3);
+ __debug_mmu.damr[3].P = __get_DAMPR(3);
+ __debug_mmu.damr[4].L = __get_DAMLR(4);
+ __debug_mmu.damr[4].P = __get_DAMPR(4);
+ __debug_mmu.damr[5].L = __get_DAMLR(5);
+ __debug_mmu.damr[5].P = __get_DAMPR(5);
+ __debug_mmu.damr[6].L = __get_DAMLR(6);
+ __debug_mmu.damr[6].P = __get_DAMPR(6);
+ __debug_mmu.damr[7].L = __get_DAMLR(7);
+ __debug_mmu.damr[7].P = __get_DAMPR(7);
+ __debug_mmu.damr[8].L = __get_DAMLR(8);
+ __debug_mmu.damr[8].P = __get_DAMPR(8);
+ __debug_mmu.damr[9].L = __get_DAMLR(9);
+ __debug_mmu.damr[9].P = __get_DAMPR(9);
+ __debug_mmu.damr[10].L = __get_DAMLR(10);
+ __debug_mmu.damr[10].P = __get_DAMPR(10);
+ __debug_mmu.damr[11].L = __get_DAMLR(11);
+ __debug_mmu.damr[11].P = __get_DAMPR(11);
+ __debug_mmu.damr[12].L = __get_DAMLR(12);
+ __debug_mmu.damr[12].P = __get_DAMPR(12);
+ __debug_mmu.damr[13].L = __get_DAMLR(13);
+ __debug_mmu.damr[13].P = __get_DAMPR(13);
+ __debug_mmu.damr[14].L = __get_DAMLR(14);
+ __debug_mmu.damr[14].P = __get_DAMPR(14);
```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+   __debug_mmu.damr[15].L = __get_DAMLR(15);
+   __debug_mmu.damr[15].P = __get_DAMPR(15);
+
+#ifdef CONFIG_MMU
+   do {
+       /* read the DAT entries from the TLB */
+       struct __debug_amr *p;
+       int loop;
+
+       asm volatile("movsg tplr,%0" : "=r"(__debug_mmu.regs.tplr));
+       asm volatile("movsg tppr,%0" : "=r"(__debug_mmu.regs.tppr));
+       asm volatile("movsg tpxr,%0" : "=r"(__debug_mmu.regs.tpxr));
+       asm volatile("movsg cxnr,%0" : "=r"(__debug_mmu.regs.cxnr));
+
+       p = __debug_mmu.tlb;
+
+       /* way 0 */
+       asm volatile("movgs %0,tpxr" :: "r"(0 << TPXR_WAY_SHIFT));
+       for (loop = 0; loop < 64; loop++) {
+           asm volatile("tlbpr %0,gr0,#1,#0" :: "r"(loop << PAGE_SHIFT));
+           asm volatile("movsg tplr,%0" : "=r"(p->L));
+           asm volatile("movsg tppr,%0" : "=r"(p->P));
+           p++;
+       }
+
+       /* way 1 */
+       asm volatile("movgs %0,tpxr" :: "r"(1 << TPXR_WAY_SHIFT));
+       for (loop = 0; loop < 64; loop++) {
+           asm volatile("tlbpr %0,gr0,#1,#0" :: "r"(loop << PAGE_SHIFT));
+           asm volatile("movsg tplr,%0" : "=r"(p->L));
+           asm volatile("movsg tppr,%0" : "=r"(p->P));
+           p++;
+       }
+
+       asm volatile("movgs %0,tplr" :: "r"(__debug_mmu.regs.tplr));
+       asm volatile("movgs %0,tppr" :: "r"(__debug_mmu.regs.tppr));
+       asm volatile("movgs %0,tpxr" :: "r"(__debug_mmu.regs.tpxr));
+   } while(0);
+#endif
+} /* end gdbstub_get_mmu_state() */
+
+/*
+ * handle event interception and GDB remote protocol processing
+ * - on entry:
+ *   PSR.ET==0, PSR.S==1 and the CPU is in debug mode
+ *   __debug_frame points to the saved registers
+ *   __frame points to the kernel mode exception frame, if it was in kernel
+ *   mode when the break happened
+ */
+void gdbstub(int signal)
+{
+   unsigned long addr, length, loop, dbar, temp, temp2, temp3;
+   uint32_t zero;
+   char *ptr;
+   int flush_cache = 0;
+
+   LEDS(0x5000);
+
+   if (signal < 0) {
+#ifndef CONFIG_GDBSTUB_IMMEDIATE

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+         /* return immediately if GDB immediate activation option not set */
+         return;
+     }
+     sigval = SIGINT;
+ }
+
+     save_user_regs(&__break_user_context);
+
+ #if 0
+     gdbstub_printk("--> gdbstub() %08x %p %08x %08x\n",
+         __debug_frame->pc,
+         __debug_frame,
+         __debug_regs->brr,
+         __debug_regs->bpsr);
+ //     gdbstub_show_regs();
+ #endif
+
+     LEDS(0x5001);
+
+     /* if we were interrupted by input on the serial gdbstub serial port,
+      * restore the context prior to the interrupt so that we return to that
+      * directly
+      */
+     temp = (unsigned long) __entry_kerneltrap_table;
+     temp2 = (unsigned long) __entry_usertrap_table;
+     temp3 = __debug_frame->pc & ~15;
+
+     if (temp3 == temp + TBR_TT_INTERRUPT_15 ||
+         temp3 == temp2 + TBR_TT_INTERRUPT_15
+         ) {
+         asm volatile("movsg pcsr,%0" : "=r"(__debug_frame->pc));
+         __debug_frame->psr |= PSR_ET;
+         __debug_frame->psr &= ~PSR_S;
+         if (__debug_frame->psr & PSR_PS)
+             __debug_frame->psr |= PSR_S;
+         __debug_regs->brr = (__debug_frame->tbr & TBR_TT) << 12;
+         __debug_regs->brr |= BRR_EB;
+         sigval = SIGINT;
+     }
+
+     /* handle the decrement timer going off (FR451 only) */
+     if (temp3 == temp + TBR_TT_DECREMENT_TIMER ||
+         temp3 == temp2 + TBR_TT_DECREMENT_TIMER
+         ) {
+         asm volatile("movgs %0,timerd" :: "r"(1000000));
+         asm volatile("movsg pcsr,%0" : "=r"(__debug_frame->pc));
+         __debug_frame->psr |= PSR_ET;
+         __debug_frame->psr &= ~PSR_S;
+         if (__debug_frame->psr & PSR_PS)
+             __debug_frame->psr |= PSR_S;
+         __debug_regs->brr = (__debug_frame->tbr & TBR_TT) << 12;
+         __debug_regs->brr |= BRR_EB;
+         sigval = SIGXCPU;
+     }
+
+     LEDS(0x5002);
+
+     /* after a BREAK insn, the PC lands on the far side of it */
+     if (__debug_regs->brr & BRR_SB)
+         gdbstub_check_breakpoint();
+
+

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+ LEDS(0x5003);
+
+ /* handle attempts to write console data via GDB "O" commands */
+ if (__debug_frame->pc == (unsigned long) gdbstub_console_write + 4) {
+     __gdbstub_console_write((struct console *) __debug_frame->gr8,
+                             (const char *) __debug_frame->gr9,
+                             (unsigned) __debug_frame->gr10);
+     goto done;
+ }
+
+ if (gdbstub_rx_unget) {
+     sigval = SIGINT;
+     goto packet_waiting;
+ }
+
+ if (!sigval)
+     sigval = gdbstub_compute_signal(__debug_regs->brr);
+
+ LEDS(0x5004);
+
+ /* send a message to the debugger's user saying what happened if it may
+  * not be clear cut (we can't map exceptions onto signals properly)
+  */
+ if (sigval != SIGINT && sigval != SIGTRAP && sigval != SIGILL) {
+     static const char title[] = "Break ";
+     static const char crlf[] = "\r\n";
+     unsigned long brr = __debug_regs->brr;
+     char hx;
+
+     ptr = output_buffer;
+     *ptr++ = 'O';
+     ptr = mem2hex(title, ptr, sizeof(title) - 1, 0);
+
+     hx = hexchars[(brr & 0xf0000000) >> 28];
+     *ptr++ = hexchars[hx >> 4];     *ptr++ = hexchars[hx & 0xf];
+     hx = hexchars[(brr & 0xf0000000) >> 24];
+     *ptr++ = hexchars[hx >> 4];     *ptr++ = hexchars[hx & 0xf];
+     hx = hexchars[(brr & 0xf0000000) >> 20];
+     *ptr++ = hexchars[hx >> 4];     *ptr++ = hexchars[hx & 0xf];
+     hx = hexchars[(brr & 0xf0000000) >> 16];
+     *ptr++ = hexchars[hx >> 4];     *ptr++ = hexchars[hx & 0xf];
+     hx = hexchars[(brr & 0xf0000000) >> 12];
+     *ptr++ = hexchars[hx >> 4];     *ptr++ = hexchars[hx & 0xf];
+     hx = hexchars[(brr & 0xf0000000) >> 8];
+     *ptr++ = hexchars[hx >> 4];     *ptr++ = hexchars[hx & 0xf];
+     hx = hexchars[(brr & 0xf0000000) >> 4];
+     *ptr++ = hexchars[hx >> 4];     *ptr++ = hexchars[hx & 0xf];
+     hx = hexchars[(brr & 0xf0000000)];
+     *ptr++ = hexchars[hx >> 4];     *ptr++ = hexchars[hx & 0xf];
+
+     ptr = mem2hex(crlf, ptr, sizeof(crlf) - 1, 0);
+     *ptr = 0;
+     gdbstub_send_packet(output_buffer);     /* send it off... */
+ }
+
+ LEDS(0x5005);
+
+ /* tell the debugger that an exception has occurred */
+ ptr = output_buffer;
+
+ /* Send trap type (converted to signal) */
+ *ptr++ = 'T';

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+ *ptr++ = hexchars[sigval >> 4];
+ *ptr++ = hexchars[sigval & 0xf];
+
+ /* Send Error PC */
+ *ptr++ = hexchars[GDB_REG_PC >> 4];
+ *ptr++ = hexchars[GDB_REG_PC & 0xf];
+ *ptr++ = ':';
+ ptr = mem2hex(&__debug_frame->pc, ptr, 4, 0);
+ *ptr++ = ';';
+
+ /*
+  * Send frame pointer
+  */
+ *ptr++ = hexchars[GDB_REG_FP >> 4];
+ *ptr++ = hexchars[GDB_REG_FP & 0xf];
+ *ptr++ = ':';
+ ptr = mem2hex(&__debug_frame->fp, ptr, 4, 0);
+ *ptr++ = ';';
+
+ /*
+  * Send stack pointer
+  */
+ *ptr++ = hexchars[GDB_REG_SP >> 4];
+ *ptr++ = hexchars[GDB_REG_SP & 0xf];
+ *ptr++ = ':';
+ ptr = mem2hex(&__debug_frame->sp, ptr, 4, 0);
+ *ptr++ = ';';
+
+ *ptr++ = 0;
+ gdbstub_send_packet(output_buffer);      /* send it off... */
+
+ LEDS(0x5006);
+
+ packet_waiting:
+ gdbstub_get_mmu_state();
+
+ /* wait for input from remote GDB */
+ while (1) {
+     output_buffer[0] = 0;
+
+     LEDS(0x5007);
+     gdbstub_recv_packet(input_buffer);
+     LEDS(0x5600 | input_buffer[0]);
+
+     switch (input_buffer[0]) {
+         /* request repeat of last signal number */
+     case '?':
+         output_buffer[0] = 'S';
+         output_buffer[1] = hexchars[sigval >> 4];
+         output_buffer[2] = hexchars[sigval & 0xf];
+         output_buffer[3] = 0;
+         break;
+
+     case 'd':
+         /* toggle debug flag */
+         break;
+
+         /* return the value of the CPU registers
+          * - GR0, GR1, GR2, GR3, GR4, GR5, GR6, GR7,
+          * - GR8, GR9, GR10, GR11, GR12, GR13, GR14, GR15,
+          * - GR16, GR17, GR18, GR19, GR20, GR21, GR22, GR23,
+          * - GR24, GR25, GR26, GR27, GR28, GR29, GR30, GR31,

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+         * - GR32, GR33, GR34, GR35, GR36, GR37, GR38, GR39,
+         * - GR40, GR41, GR42, GR43, GR44, GR45, GR46, GR47,
+         * - GR48, GR49, GR50, GR51, GR52, GR53, GR54, GR55,
+         * - GR56, GR57, GR58, GR59, GR60, GR61, GR62, GR63,
+         * - FP0, FP1, FP2, FP3, FP4, FP5, FP6, FP7,
+         * - FP8, FP9, FP10, FP11, FP12, FP13, FP14, FP15,
+         * - FP16, FP17, FP18, FP19, FP20, FP21, FP22, FP23,
+         * - FP24, FP25, FP26, FP27, FP28, FP29, FP30, FP31,
+         * - FP32, FP33, FP34, FP35, FP36, FP37, FP38, FP39,
+         * - FP40, FP41, FP42, FP43, FP44, FP45, FP46, FP47,
+         * - FP48, FP49, FP50, FP51, FP52, FP53, FP54, FP55,
+         * - FP56, FP57, FP58, FP59, FP60, FP61, FP62, FP63,
+         * - PC, PSR, CCR, CCCR,
+         * - _X132, _X133, _X134
+         * - TBR, BRR, DBAR0, DBAR1, DBAR2, DBAR3,
+         * - _X141, _X142, _X143, _X144,
+         * - LR, LCR
+         */
+     case 'g':
+         zero = 0;
+         ptr = output_buffer;
+
+         /* deal with GR0, GR1-GR27, GR28-GR31, GR32-GR63 */
+         ptr = mem2hex(&zero, ptr, 4, 0);
+
+         for (loop = 1; loop <= 27; loop++)
+             ptr = mem2hex((unsigned long *)__debug_frame + REG_GR(loop),
+                             ptr, 4, 0);
+         temp = (unsigned long) __frame;
+         ptr = mem2hex(&temp, ptr, 4, 0);
+         ptr = mem2hex((unsigned long *)__debug_frame + REG_GR(29), ptr, 4, 0);
+         ptr = mem2hex((unsigned long *)__debug_frame + REG_GR(30), ptr, 4, 0);
+ #ifdef CONFIG_MMU
+         ptr = mem2hex((unsigned long *)__debug_frame + REG_GR(31), ptr, 4, 0);
+ #else
+         temp = (unsigned long) __debug_frame;
+         ptr = mem2hex(&temp, ptr, 4, 0);
+ #endif
+
+         for (loop = 32; loop <= 63; loop++)
+             ptr = mem2hex((unsigned long *)__debug_frame + REG_GR(loop),
+                             ptr, 4, 0);
+
+         /* deal with FR0-FR63 */
+         for (loop = 0; loop <= 63; loop++)
+             ptr = mem2hex((unsigned long *)&__break_user_context +
+                             __FPMEDIA_FR(loop),
+                             ptr, 4, 0);
+
+         /* deal with special registers */
+         ptr = mem2hex(&__debug_frame->pc, ptr, 4, 0);
+         ptr = mem2hex(&__debug_frame->psr, ptr, 4, 0);
+         ptr = mem2hex(&__debug_frame->ccr, ptr, 4, 0);
+         ptr = mem2hex(&__debug_frame->cccr, ptr, 4, 0);
+         ptr = mem2hex(&zero, ptr, 4, 0);
+         ptr = mem2hex(&zero, ptr, 4, 0);
+         ptr = mem2hex(&zero, ptr, 4, 0);
+         ptr = mem2hex(&__debug_frame->tbr, ptr, 4, 0);
+         ptr = mem2hex(&__debug_regs->brr, ptr, 4, 0);
+
+         asm volatile("movsg dbar0,%0" : "=r"(dbar));
+         ptr = mem2hex(&dbar, ptr, 4, 0);

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```

+         asm volatile("movsg dbar1,%0" : "=r"(dbar));
+         ptr = mem2hex(&dbar, ptr, 4, 0);
+         asm volatile("movsg dbar2,%0" : "=r"(dbar));
+         ptr = mem2hex(&dbar, ptr, 4, 0);
+         asm volatile("movsg dbar3,%0" : "=r"(dbar));
+         ptr = mem2hex(&dbar, ptr, 4, 0);
+
+         asm volatile("movsg scr0,%0" : "=r"(dbar));
+         ptr = mem2hex(&dbar, ptr, 4, 0);
+         asm volatile("movsg scr1,%0" : "=r"(dbar));
+         ptr = mem2hex(&dbar, ptr, 4, 0);
+         asm volatile("movsg scr2,%0" : "=r"(dbar));
+         ptr = mem2hex(&dbar, ptr, 4, 0);
+         asm volatile("movsg scr3,%0" : "=r"(dbar));
+         ptr = mem2hex(&dbar, ptr, 4, 0);
+
+         ptr = mem2hex(&__debug_frame->lr, ptr, 4, 0);
+         ptr = mem2hex(&__debug_frame->lcr, ptr, 4, 0);
+
+         ptr = mem2hex(&__debug_frame->iacc0, ptr, 8, 0);
+
+         ptr = mem2hex(&__break_user_context.f.fsr[0], ptr, 4, 0);
+
+         for (loop = 0; loop <= 7; loop++)
+             ptr = mem2hex(&__break_user_context.f.acc[loop], ptr, 4, 0);
+
+         ptr = mem2hex(&__break_user_context.f.accg, ptr, 8, 0);
+
+         for (loop = 0; loop <= 1; loop++)
+             ptr = mem2hex(&__break_user_context.f.msr[loop], ptr, 4, 0);
+
+         ptr = mem2hex(&__debug_frame->gner0, ptr, 4, 0);
+         ptr = mem2hex(&__debug_frame->gner1, ptr, 4, 0);
+
+         ptr = mem2hex(&__break_user_context.f.fner[0], ptr, 4, 0);
+         ptr = mem2hex(&__break_user_context.f.fner[1], ptr, 4, 0);
+
+         break;
+
+         /* set the values of the CPU registers */
+     case 'G':
+         ptr = &input_buffer[1];
+
+         /* deal with GR0, GR1-GR27, GR28-GR31, GR32-GR63 */
+         ptr = hex2mem(ptr, &temp, 4);
+
+         for (loop = 1; loop <= 27; loop++)
+             ptr = hex2mem(ptr, (unsigned long *)__debug_frame + REG_GR(loop),
+                             4);
+
+         ptr = hex2mem(ptr, &temp, 4);
+         __frame = (struct pt_regs *) temp;
+         ptr = hex2mem(ptr, &__debug_frame->gr29, 4);
+         ptr = hex2mem(ptr, &__debug_frame->gr30, 4);
+ #ifdef CONFIG_MMU
+         ptr = hex2mem(ptr, &__debug_frame->gr31, 4);
+ #else
+         ptr = hex2mem(ptr, &temp, 4);
+ #endif
+
+         for (loop = 32; loop <= 63; loop++)
+             ptr = hex2mem(ptr, (unsigned long *)__debug_frame + REG_GR(loop),

```

## Linux-Kernel: [PATCH 5/20] FRV: Fujitsu FR-V CPU arch implementation part 3

```
+         4);
+
+         /* deal with FR0-FR63 */
+         for (loop = 0; loop <= 63; loop++)
+             ptr = mem2hex((unsigned long *)&__break_user_context +
+                __FPMEDIA_FR(loop),
+                ptr, 4, 0);
+
+         /* deal with special registers */
+         ptr = hex2mem(ptr, &__debug_frame->pc, 4);
+         ptr = hex2mem(ptr, &__debug_frame->psr, 4);
+         ptr = hex2mem(ptr, &__debug_frame->ccr, 4);
+         ptr = hex2mem(ptr, &__debug_
```