

[PATCH,RFC 2.6.14 09/15] KGDB: SuperH-specific changes

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-11/3770.html>

From: Tom Rini (trini_at_kernel.crashing.org)
Date: 11/10/05

To: Andrew Morton <akpm@osdl.org>
Date: Thu, 10 Nov 2005 11:42:18 -0500

This adds basic support for KGDB on SuperH as well as adding some architecture specific notes to the DocBook file and converting the 7751 to use this. I have tested all combinations of 8250 and SCI(F) ports being used as KGDB and console that I could (one of each usable to me).

Documentation/DocBook/kgdb.tmpl | 16
arch/sh/Kconfig.debug | 92 ---
arch/sh/Makefile | 1
arch/sh/boards/se/7751/setup.c | 139 ----
arch/sh/kernel/Makefile | 2
arch/sh/kernel/cpu/sh3/ex.S | 2
arch/sh/kernel/cpu/sh4/ex.S | 2
arch/sh/kernel/entry.S | 30
arch/sh/kernel/kgdb-jmp.S | 32
arch/sh/kernel/kgdb.c | 363 ++++++++
arch/sh/kernel/kgdb_jmp.S | 33
arch/sh/kernel/kgdb_stub.c | 1491 -----
arch/sh/kernel/setup.c | 94 ---
arch/sh/kernel/time.c | 11
arch/sh/kernel/traps.c | 20
arch/sh/mm/extable.c | 7
arch/sh/mm/fault-nommu.c | 14
arch/sh/mm/fault.c | 12
drivers/serial/sh-sci.c | 298 ++++----
include/asm-sh/kgdb.h | 116 ----
include/asm-sh/system.h | 40 +
lib/Kconfig.debug | 6
22 files changed, 676 insertions(+), 2145 deletions(-)

Index: linux-2.6.14/arch/sh/boards/se/7751/setup.c

```
=====
--- linux-2.6.14.orig/arch/sh/boards/se/7751/setup.c
+++ linux-2.6.14/arch/sh/boards/se/7751/setup.c
@@ -18,10 +18,6 @@
#include <asm/io.h>
```

```

#include <asm/se7751/se7751.h>

#ifndef CONFIG_SH_KGDB
#include <asm/kgdb.h>
#endif
-
/*
 * Configure the Super I/O chip
 */
@@ -83,12 +79,6 @@ const char *get_system_type(void)
    return "7751 SolutionEngine";
}

#ifndef CONFIG_SH_KGDB
-static int kgdb_uart_setup(void);
-static struct kgdb_sermap kgdb_uart_sermap =
-{ "ttyS", 0, kgdb_uart_setup, NULL };
#endif
-
/*
 * Initialize the board
 */
@@ -96,133 +86,4 @@ void __init platform_setup(void)
{
    /* Call init_smsc() replacement to set up SuperIO. */
    /* XXX: RTC setting comes here */
#ifndef CONFIG_SH_KGDB
    kgdb_register_sermap(&kgdb_uart_sermap);
#endif
-}
-
-/******
- * Currently a hack (e.g. does not interact well w/serial.c, lots of *
- * hardcoded stuff) but may be useful if SCI/F needs debugging. *
- * Mostly copied from x86 code (see files asm-i386/kgdb_local.h and *
- * arch/i386/lib/kgdb_serial.c). *
- *****/
-
#ifndef CONFIG_SH_KGDB
#include <linux/types.h>
#include <linux/serial.h>
#include <linux/serialP.h>
#include <linux/serial_reg.h>
-
#define COM1_PORT 0x3f8 /* Base I/O address */
#define COM1_IRQ 4 /* IRQ not used yet */
#define COM2_PORT 0x2f8 /* Base I/O address */
#define COM2_IRQ 3 /* IRQ not used yet */
-
#define SB_CLOCK 1843200 /* Serial baud clock */
#define SB_BASE (SB_CLOCK/16)

```

```

-#define SB_MCR UART_MCR_OUT2 | UART_MCR_DTR | UART_MCR_RTS
-
-struct uart_port {
- int base;
-};
-#define UART_NPORTS 2
-struct uart_port uart_ports[] = {
- { COM1_PORT },
- { COM2_PORT },
-};
-struct uart_port *kgdb_uart_port;
-
-#define UART_IN(reg) inb_p(kgdb_uart_port->base + reg)
-#define UART_OUT(reg,v) outb_p((v), kgdb_uart_port->base + reg)
-
-/* Basic read/write functions for the UART */
-#define UART_LSR_RXCERR (UART_LSR_BI | UART_LSR_FE | UART_LSR_PE)
-static int kgdb_uart_getchar(void)
-{
- int lsr;
- int c = -1;
-
- while (c == -1) {
- lsr = UART_IN(UART_LSR);
- if (lsr & UART_LSR_DR)
- c = UART_IN(UART_RX);
- if ((lsr & UART_LSR_RXCERR))
- c = -1;
- }
- return c;
-}
-
-static void kgdb_uart_putchar(int c)
-{
- while ((UART_IN(UART_LSR) & UART_LSR_THRE) == 0)
- ;
- UART_OUT(UART_TX, c);
-}
-
-/*
- * Initialize UART to configured/requested values.
- * (But we don't interrupts yet, or interact w/serial.c)
- */
-static int kgdb_uart_setup(void)
-{
- int port;
- int lcr = 0;
- int bdiv = 0;
-
- if (kgdb_portnum >= UART_NPORTS) {
- KGDB_PRINTK("uart port %d invalid.\n", kgdb_portnum);

```

```

- return -1;
- }
-
- kgdb_uart_port = &uart_ports[kgdb_portnum];
-
- /* Init sequence from gdb_hook_interrupt */
- UART_IN(UART_RX);
- UART_OUT(UART_IER, 0);
-
- UART_IN(UART_RX); /* Serial driver comments say */
- UART_IN(UART_IIR); /* this clears interrupt regs */
- UART_IN(UART_MSR);
-
- /* Figure basic LCR values */
- switch (kgdb_bits) {
- case '7':
- lcr |= UART_LCR_WLEN7;
- break;
- default: case '8':
- lcr |= UART_LCR_WLEN8;
- break;
- }
- switch (kgdb_parity) {
- case 'O':
- lcr |= UART_LCR_PARITY;
- break;
- case 'E':
- lcr |= (UART_LCR_PARITY | UART_LCR_EPAR);
- break;
- default: break;
- }
-
- /* Figure the baud rate divisor */
- bdiv = (SB_BASE/kgdb_baud);
-
- /* Set the baud rate and LCR values */
- UART_OUT(UART_LCR, (lcr | UART_LCR_DLAB));
- UART_OUT(UART_DLL, (bdiv & 0xff));
- UART_OUT(UART_DLM, ((bdiv >> 8) & 0xff));
- UART_OUT(UART_LCR, lcr);
-
- /* Set the MCR */
- UART_OUT(UART_MCR, SB_MCR);
-
- /* Turn off FIFOs for now */
- UART_OUT(UART_FCR, 0);
-
- /* Setup complete: initialize function pointers */
- kgdb_getchar = kgdb_uart_getchar;
- kgdb_putchar = kgdb_uart_putchar;
-

```

```
- return 0;
}
```

```
---#endif /* CONFIG_SH_KGDB */
```

```
Index: linux-2.6.14/arch/sh/Kconfig.debug
```

```
----- linux-2.6.14.orig/arch/sh/Kconfig.debug
```

```
+++ linux-2.6.14/arch/sh/Kconfig.debug
```

```
@@ -29,96 +29,4 @@ config EARLY_PRINTK
```

```
    This option is only useful porting the kernel to a new machine,
    when the kernel may crash or hang before the serial console is
    initialised. If unsure, say N.
```

```
-
-configure KGDB
```

```
- bool "Include KGDB kernel debugger"
```

```
- help
```

```
- Include in-kernel hooks for kgdb, the Linux kernel source level
```

```
- debugger. See <http://kgdb.sourceforge.net/> for more information.
```

```
- Unless you are intending to debug the kernel, say N here.
```

```
-menu "KGDB configuration options"
```

```
- depends on KGDB
```

```
-configure MORE_COMPILE_OPTIONS
```

```
- bool "Add any additional compile options"
```

```
- help
```

```
- If you want to add additional CFLAGS to the kernel build, enable this
```

```
- option and then enter what you would like to add in the next question.
```

```
- Note however that -g is already appended with the selection of KGDB.
```

```
-configure COMPILE_OPTIONS
```

```
- string "Additional compile arguments"
```

```
- depends on MORE_COMPILE_OPTIONS
```

```
-configure KGDB_NMI
```

```
- bool "Enter KGDB on NMI"
```

```
- default n
```

```
-configure KGDB_THREAD
```

```
- bool "Include KGDB thread support"
```

```
- default y
```

```
-configure SH_KGDB_CONSOLE
```

```
- bool "Console messages through GDB"
```

```
- default n
```

```
-configure KGDB_SYSRQ
```

```
- bool "Allow SysRq 'G' to enter KGDB"
```

```
- default y
```

```
-configure KGDB_KERNEL_ASSERTS
```

```
- bool "Include KGDB kernel assertions"
```

```
- default n
-
-comment "Serial port setup"
-
-config KGDB_DEFPORT
-int "Port number (ttySCn)"
-default "1"
-
-config KGDB_DEFBAUD
-int "Baud rate"
-default "115200"
-
-choice
-prompt "Parity"
-depends on KGDB
-default KGDB_DEFPARITY_N
-
-config KGDB_DEFPARITY_N
-bool "None"
-
-config KGDB_DEFPARITY_E
-bool "Even"
-
-config KGDB_DEFPARITY_O
-bool "Odd"
-
-endchoice
-
-choice
-prompt "Data bits"
-depends on KGDB
-default KGDB_DEFBITS_8
-
-config KGDB_DEFBITS_8
-bool "8"
-
-config KGDB_DEFBITS_7
-bool "7"
-
-endchoice
-
-endmenu
-
-config FRAME_POINTER
-bool "Compile the kernel with frame pointers"
-default y if KGDB
-help
- If you say Y here the resulting kernel image will be slightly larger
- and slower, but it will give very useful debugging information.
- If you don't debug the kernel, you can say N, but we may not be able
- to solve problems without frame pointers.
```

```

-
endmenu
Index: linux-2.6.14/arch/sh/kernel/cpu/sh3/ex.S
=====
--- linux-2.6.14.orig/arch/sh/kernel/cpu/sh3/ex.S
+++ linux-2.6.14/arch/sh/kernel/cpu/sh3/ex.S
@@ -43,7 +43,7 @@ ENTRY(exception_handling_table)
     .long exception_error ! reserved_instruction (filled by trap_init) /* 180 */
     .long exception_error ! illegal_slot_instruction (filled by trap_init) /* 1A0 */
ENTRY(nmi_slot)
-#if defined (CONFIG_KGDB_NMI)
+#if defined (CONFIG_KGDB)
     .long debug_enter /* 1C0 */ ! Allow trap to debugger
#else
     .long exception_none /* 1C0 */ ! Not implemented yet
Index: linux-2.6.14/arch/sh/kernel/cpu/sh4/ex.S
=====
--- linux-2.6.14.orig/arch/sh/kernel/cpu/sh4/ex.S
+++ linux-2.6.14/arch/sh/kernel/cpu/sh4/ex.S
@@ -47,7 +47,7 @@ ENTRY(exception_handling_table)
     .long exception_error ! reserved_instruction (filled by trap_init) /* 180 */
     .long exception_error ! illegal_slot_instruction (filled by trap_init) /* 1A0 */
ENTRY(nmi_slot)
-#if defined (CONFIG_KGDB_NMI)
+#if defined (CONFIG_KGDB)
     .long debug_enter /* 1C0 */ ! Allow trap to debugger
#else
     .long exception_none /* 1C0 */ ! Not implemented yet
Index: linux-2.6.14/arch/sh/kernel/entry.S
=====
--- linux-2.6.14.orig/arch/sh/kernel/entry.S
+++ linux-2.6.14/arch/sh/kernel/entry.S
@@ -92,7 +92,7 @@ @@ INTEVT = 0xff000028
MMU_TEA = 0xff00000c ! TLB Exception Address Register
#endif

-#if defined(CONFIG_KGDB_NMI)
+#if defined(CONFIG_KGDB)
NMI_VEC = 0x1c0 ! Must catch early for debounce
#endif

@@ -244,31 +244,33 @@ call_dae:
2: .long do_address_error
#endif /* CONFIG_MMU */

-#if defined(CONFIG_SH_STANDARD_BIOS) || defined(CONFIG_SH_KGDB)
+#if defined(CONFIG_SH_STANDARD_BIOS) || defined(CONFIG_KGDB)
! Handle kernel debug if either kgdb (SW) or gdb-stub (FW) is present.
! If both are configured, handle the debug traps (breakpoints) in SW,
! but still allow BIOS traps to FW.

```

```

.align 2
debug_kernel:
-#if defined(CONFIG_SH_STANDARD_BIOS) && defined(CONFIG_SH_KGDB)
+#if defined(CONFIG_SH_STANDARD_BIOS) && defined(CONFIG_KGDB)
    /* Force BIOS call to FW (debug_trap put TRA in r8) */
    mov r8,r0
    shlr2 r0
    cmp/eq #0x3f,r0
    bt debug_kernel_fw
-#endif /* CONFIG_SH_STANDARD_BIOS && CONFIG_SH_KGDB */
+#endif /* CONFIG_SH_STANDARD_BIOS && CONFIG_KGDB */

-#debug_enter:
-#if defined(CONFIG_SH_KGDB)
+ .align 2
+ .globl debug_enter
+debug_enter:
+#if defined(CONFIG_KGDB)
    /* Jump to kgdb, pass stacked regs as arg */
debug_kernel_sw:
    mov.l 3f, r0
    jmp @r0
    mov r15, r4
    .align 2
-3: .long kgdb_handle_exception
-#endif /* CONFIG_SH_KGDB */
+3: .long kgdb_exception_handler
+#endif /* CONFIG_KGDB */

#if defined(CONFIG_SH_STANDARD_BIOS)
    /* Unwind the stack and jmp to the debug entry */
@@ -310,12 +312,12 @@ debug_kernel_fw:
2: .long gdb_vbr_vector
#endif /* CONFIG_SH_STANDARD_BIOS */

-#endif /* CONFIG_SH_STANDARD_BIOS || CONFIG_SH_KGDB */
+#endif /* CONFIG_SH_STANDARD_BIOS || CONFIG_KGDB */

    .align 2
-#debug_trap:
-#if defined(CONFIG_SH_STANDARD_BIOS) || defined(CONFIG_SH_KGDB)
+debug_trap:
+#if defined(CONFIG_SH_STANDARD_BIOS) || defined(CONFIG_KGDB)
    mov #OFF_SR, r0
    mov.l @(r0,r15), r0 ! get status register
    shll r0
@@ -659,7 +661,7 @@ skip_restore:
6: or k0, k2 ! Set the IMASK-bits
    ldc k2, ssr
    !

```

```

-#if defined(CONFIG_KGDB_NMI)
+#if defined(CONFIG_KGDB)
    ! Clear in_nmi
    mov.l 4f, k0
    mov #0, k1
@@ -711,7 +713,7 @@ tlb_miss:
interrupt:
    mov.l 2f, k2
    mov.l 3f, k3
-#if defined(CONFIG_KGDB_NMI)
+#if defined(CONFIG_KGDB)
    ! Debounce (filter nested NMI)
    mov.l @k2, k0
    mov.l 5f, k1
@@ -726,7 +728,7 @@ interrupt:
5: .long NMI_VEC
6: .long in_nmi
0:
-#endif /* defined(CONFIG_KGDB_NMI) */
+#endif /* defined(CONFIG_KGDB) */
    bra handle_exception
    mov.l @k2, k2

```

Index: linux-2.6.14/arch/sh/kernel/kgdb.c

```

=====
--- /dev/null
+++ linux-2.6.14/arch/sh/kernel/kgdb.c
@@ -0,0 +1,363 @@
+/*
+ * arch/sh/kernel/kgdb.c
+ *
+ * Contains SH-specific low-level support for KGDB.
+ *
+ * Contains extracts from code by Glenn Engel, Jim Kingdon,
+ * David Grothe <dave@gcom.com>, Tigran Aivazian <tigran@sco.com>,
+ * Amit S. Kale <akale@veritas.com>, William Gatliff <bgat@open-widgets.com>,
+ * Ben Lee, Steve Chamberlain and Benoit Miller <fulg@iname.com>,
+ * Henry Bell <henry.bell@st.com> and Jeremy Siegel <jsiegel@mvista.com>
+ *
+ * Maintainer: Tom Rini <trini@kernel.crashing.org>
+ *
+ * 2004 (c) MontaVista Software, Inc. This file is licensed under
+ * the terms of the GNU General Public License version 2. This program
+ * is licensed "as is" without any warranty of any kind, whether express
+ * or implied.
+ */
+
+#include <linux/string.h>
+#include <linux/kernel.h>
+#include <linux/sched.h>
+#include <linux/smp.h>

```

```

+#include <linux/spinlock.h>
+#include <linux/delay.h>
+#include <linux/linkage.h>
+#include <linux/init.h>
+#include <linux/kgdb.h>
+
+#include <asm/system.h>
+#include <asm/current.h>
+#include <asm/signal.h>
+#include <asm/pgtable.h>
+#include <asm/ptrace.h>
+
+extern void per_cpu_trap_init(void);
+extern atomic_t cpu_doing_single_step;
+
+/* Function pointers for linkage */
+static struct kgdb_regs trap_registers;
+
+/* Globals. */
+char in_nmi; /* Set during NMI to prevent reentry */
+
+/* TRA differs sh3/4 */
+#if defined(CONFIG_CPU_SH3)
+#define TRA 0xfffffd0
+#elif defined(CONFIG_CPU_SH4)
+#define TRA 0xff000020
+#endif
+
+/* Macros for single step instruction identification */
+#define OPCODE_BT(op) (((op) & 0xff00) == 0x8900)
+#define OPCODE_BF(op) (((op) & 0xff00) == 0x8b00)
+#define OPCODE_BTf_DISP(op) (((op) & 0x80) ? (((op) | 0xfffff80) << 1) : \
+ (((op) & 0x7f) << 1))
+#define OPCODE_BFS(op) (((op) & 0xff00) == 0x8f00)
+#define OPCODE_BTS(op) (((op) & 0xff00) == 0x8d00)
+#define OPCODE_BRA(op) (((op) & 0xf000) == 0xa000)
+#define OPCODE_BRAf_DISP(op) (((op) & 0x800) ? (((op) | 0xffff800) << 1) : \
+ (((op) & 0x7ff) << 1))
+#define OPCODE_BRAF(op) (((op) & 0xf0ff) == 0x0023)
+#define OPCODE_BRAF_REG(op) (((op) & 0x0f00) >> 8)
+#define OPCODE_BSR(op) (((op) & 0xf000) == 0xb000)
+#define OPCODE_BSRf_DISP(op) (((op) & 0x800) ? (((op) | 0xffff800) << 1) : \
+ (((op) & 0x7ff) << 1))
+#define OPCODE_BSRF(op) (((op) & 0xf0ff) == 0x0003)
+#define OPCODE_BSRF_REG(op) (((op) >> 8) & 0xf)
+#define OPCODE_JMP(op) (((op) & 0xf0ff) == 0x402b)
+#define OPCODE_JMP_REG(op) (((op) >> 8) & 0xf)
+#define OPCODE_JSR(op) (((op) & 0xf0ff) == 0x400b)
+#define OPCODE_JSR_REG(op) (((op) >> 8) & 0xf)
+#define OPCODE_RTS(op) ((op) == 0xb)
+#define OPCODE_RTE(op) ((op) == 0x2b)

```

```

+
+ #define SR_T_BIT_MASK 0x1
+ #define STEP_OPCODE 0xc320
+ #define BIOS_CALL_TRAP 0x3f
+
+ /* Exception codes as per SH-4 core manual */
+ #define ADDRESS_ERROR_LOAD_VEC 7
+ #define ADDRESS_ERROR_STORE_VEC 8
+ #define TRAP_VEC 11
+ #define INVALID_INSN_VEC 12
+ #define INVALID_SLOT_VEC 13
+ #define NMI_VEC 14
+ #define SERIAL_BREAK_VEC 58
+
+ /* Misc static */
+ static int stepped_address;
+ static short stepped_opcode;
+
+ /* Translate SH-3/4 exception numbers to unix-like signal values */
+ static int compute_signal(const int excep_code)
+ {
+     switch (excep_code) {
+     case INVALID_INSN_VEC:
+     case INVALID_SLOT_VEC:
+     return SIGILL;
+     case ADDRESS_ERROR_LOAD_VEC:
+     case ADDRESS_ERROR_STORE_VEC:
+     return SIGSEGV;
+     case SERIAL_BREAK_VEC:
+     case NMI_VEC:
+     return SIGINT;
+     default:
+     /* Act like it was a break/trap. */
+     return SIGTRAP;
+     }
+ }
+
+ /* Translate the registers of the system into the format that GDB wants. Since
+ * we use a local structure to store things, instead of getting them out
+ * of pt_regs, we can just do a memcpy.
+ */
+ void regs_to_gdb_regs(unsigned long *gdb_regs, struct pt_regs *ign)
+ {
+     memcpy(gdb_regs, &trap_registers, sizeof(trap_registers));
+ }
+
+ /*
+ * On SH we save: r1 (prev->thread.sp) r2 (prev->thread.pc) r4 (prev) r5 (next)
+ * r6 (next->thread.sp) r7 (next->thread.pc)
+ */

```

```

+void sleeping_thread_to_gdb_regs(unsigned long *gdb_regs, struct task_struct *p)
+{
+ int count;
+
+ for (count = 0; count < 16; count++)
+ *(gdb_regs++) = 0;
+ *(gdb_regs++) = p->thread.pc;
+ *(gdb_regs++) = 0;
+ *(gdb_regs++) = 0;
+ *(gdb_regs++) = 0;
+ *(gdb_regs++) = 0;
+ *(gdb_regs++) = 0;
+ *(gdb_regs++) = 0;
+ *(gdb_regs++) = 0;
+}
+
+/*
+ * Translate the registers values that GDB has given us back into the
+ * format of the system. See the comment above about memcpy.
+ */
+void gdb_regs_to_regs(unsigned long *gdb_regs, struct pt_regs *ign)
+{
+ memcpy(&trap_registers, gdb_regs, sizeof(trap_registers));
+}
+
+/* Calculate the new address for after a step */
+static short *get_step_address(void)
+{
+ short op = *(short *)trap_registers.pc;
+ long addr;
+
+ /* BT */
+ if (OPCODE_BT(op)) {
+ if (trap_registers.sr & SR_T_BIT_MASK)
+ addr = trap_registers.pc + 4 + OPCODE_BT_DISP(op);
+ else
+ addr = trap_registers.pc + 2;
+ }
+
+ /* BTS */
+ else if (OPCODE_BTS(op)) {
+ if (trap_registers.sr & SR_T_BIT_MASK)
+ addr = trap_registers.pc + 4 + OPCODE_BT_DISP(op);
+ else
+ addr = trap_registers.pc + 4; /* Not in delay slot */
+ }
+
+ /* BF */
+ else if (OPCODE_BF(op)) {
+ if (!(trap_registers.sr & SR_T_BIT_MASK))
+ addr = trap_registers.pc + 4 + OPCODE_BT_DISP(op);
+ else

```

```

+ addr = trap_registers.pc + 2;
+ }
+
+ /* BFS */
+ else if (OPCODE_BFS(op)) {
+ if (!(trap_registers.sr & SR_T_BIT_MASK))
+ addr = trap_registers.pc + 4 + OPCODE_BTF_DISP(op);
+ else
+ addr = trap_registers.pc + 4; /* Not in delay slot */
+ }
+
+ /* BRA */
+ else if (OPCODE_BRA(op))
+ addr = trap_registers.pc + 4 + OPCODE_BRA_DISP(op);
+
+ /* BRAF */
+ else if (OPCODE_BRAF(op))
+ addr = trap_registers.pc + 4
+ + trap_registers.regs[OPCODE_BRAF_REG(op)];
+
+ /* BSR */
+ else if (OPCODE_BSR(op))
+ addr = trap_registers.pc + 4 + OPCODE_BSR_DISP(op);
+
+ /* BSRF */
+ else if (OPCODE_BSRF(op))
+ addr = trap_registers.pc + 4
+ + trap_registers.regs[OPCODE_BSRF_REG(op)];
+
+ /* JMP */
+ else if (OPCODE_JMP(op))
+ addr = trap_registers.regs[OPCODE_JMP_REG(op)];
+
+ /* JSR */
+ else if (OPCODE_JSR(op))
+ addr = trap_registers.regs[OPCODE_JSR_REG(op)];
+
+ /* RTS */
+ else if (OPCODE_RTS(op))
+ addr = trap_registers.pr;
+
+ /* RTE */
+ else if (OPCODE_RTE(op))
+ addr = trap_registers.regs[15];
+
+ /* Other */
+ else
+ addr = trap_registers.pc + 2;
+
+ kgdb_flush_icache_range(addr, addr + 2);
+ return (short *)addr;

```

```

+}
+
+/* The command loop, read and act on requests */
+int kgdb_arch_handle_exception(int e_vector, int signo, int err_code,
+ char *remcom_in_buffer, char *remcom_out_buffer,
+ struct pt_regs *ign)
+{
+ unsigned long addr;
+ char *ptr = &remcom_in_buffer[1];
+
+ /* Examine first char of buffer to see what we need to do */
+ switch (remcom_in_buffer[0]) {
+ case 'c': /* Continue at address AA..AA (optional) */
+ case 's': /* Step one instruction from AA..AA */
+ /* Try to read optional parameter, PC unchanged if none */
+ if (kgdb_hex2long(&ptr, &addr))
+ trap_registers.pc = addr;
+
+ atomic_set(&cpu_doing_single_step, -1);
+ if (remcom_in_buffer[0] == 's') {
+ /* Replace the instruction immediately after the
+ * current instruction (i.e. next in the expected
+ * flow of control) with a trap instruction, so that
+ * returning will cause only a single instruction to
+ * be executed. Note that this model is slightly
+ * broken for instructions with delay slots
+ * (e.g. B[TF]S, BSR, BRA etc), where both the branch
+ * and the instruction in the delay slot will be
+ * executed.
+ */
+ /* Determine where the target instruction will send
+ * us to */
+ unsigned short *next_addr = get_step_address();
+ stepped_address = (int)next_addr;
+
+ /* Replace it */
+ stepped_opcode = *(short *)next_addr;
+ *next_addr = STEP_OPCODE;
+
+ /* Flush and return */
+ kgdb_flush_icache_range((long)next_addr,
+ (long)next_addr + 2);
+ if (kgdb_contthread)
+ atomic_set(&cpu_doing_single_step,
+ smp_processor_id());
+ }
+ return 0;
+ }
+ return -1;
+}
+

```

```

+/*
+ * When an exception has occurred, we are called. We need to set things
+ * up so that we can call kgdb_handle_exception to handle requests from
+ * the remote GDB.
+ */
+void kgdb_exception_handler(struct pt_regs *regs)
+{
+ int excep_code, vbr_val;
+ int count;
+
+ /* Copy kernel regs (from stack) */
+ for (count = 0; count < 16; count++)
+ trap_registers.regs[count] = regs->regs[count];
+ trap_registers.pc = regs->pc;
+ trap_registers.pr = regs->pr;
+ trap_registers.sr = regs->sr;
+ trap_registers.gbr = regs->gbr;
+ trap_registers.mach = regs->mach;
+ trap_registers.macl = regs->macl;
+
+ __asm__ __volatile__("stc vbr, %0" : "=r"(vbr_val));
+ trap_registers.vbr = vbr_val;
+
+ /* Get the exception code. */
+ __asm__ __volatile__("stc r2_bank, %0" : "=r"(excep_code));
+
+ excep_code >>= 5;
+
+ /* If we got an NMI, and KGDB is not yet initialized, call
+ * breakpoint() to try and initialize everything for us. */
+ if (excep_code == NMI_VEC && !kgdb_initialized) {
+ breakpoint();
+ return;
+ }
+
+ /* TRAP_VEC exception indicates a software trap inserted in place of
+ * code by GDB so back up PC by one instruction, as this instruction
+ * will later be replaced by its original one. Do NOT do this for
+ * trap 0xff, since that indicates a compiled-in breakpoint which
+ * will not be replaced (and we would retake the trap forever) */
+ if (excep_code == TRAP_VEC &&
+ (* (volatile unsigned long *)TRA != (0xff << 2)))
+ trap_registers.pc -= 2;
+
+ /* If we have been single-stepping, put back the old instruction.
+ * We use stepped_address in case we have stopped more than one
+ * instruction away. */
+ if (stepped_opcode != 0) {
+ *(short *)stepped_address = stepped_opcode;
+ kgdb_flush_icache_range(stepped_address, stepped_address + 2);
+ }

```

```

+ stepped_opcode = 0;
+
+ /* Call the stub to do the processing. Note that not everything we
+ * need to send back and forth lives in pt_regs. */
+ kgdb_handle_exception(excep_code, compute_signal(excep_code), 0, regs);
+
+ /* Copy back the (maybe modified) registers */
+ for (count = 0; count < 16; count++)
+ regs->regs[count] = trap_registers.regs[count];
+ regs->pc = trap_registers.pc;
+ regs->pr = trap_registers.pr;
+ regs->sr = trap_registers.sr;
+ regs->gbr = trap_registers.gbr;
+ regs->mach = trap_registers.mach;
+ regs->macl = trap_registers.macl;
+
+ vbr_val = trap_registers.vbr;
+ __asm__ __volatile__ ("ldc %0, vbr": : "r"(vbr_val));
+}
+
+int __init kgdb_arch_init(void)
+{
+ per_cpu_trap_init();
+
+ return 0;
+}
+
+struct kgdb_arch arch_kgdb_ops = {
+#ifdef CONFIG_CPU_LITTLE_ENDIAN
+ .gdb_bpt_instr = {0xff, 0xc3},
+#else
+ .gdb_bpt_instr = {0xc3, 0xff},
+#endif
+};

```

Index: linux-2.6.14/arch/sh/kernel/kgdb_jump.S

```

-----
--- linux-2.6.14.orig/arch/sh/kernel/kgdb_jump.S
+++ /dev/null
@@ -1,33 +0,0 @@
-#include <linux/linkage.h>
-
-ENTRY(setjmp)
- add #(9*4), r4
- sts.l pr, @-r4
- mov.l r15, @-r4
- mov.l r14, @-r4
- mov.l r13, @-r4
- mov.l r12, @-r4
- mov.l r11, @-r4
- mov.l r10, @-r4
- mov.l r9, @-r4

```

```

- mov.l r8, @-r4
- rts
- mov #0, r0
-
-ENTRY(longjmp)
- mov.l @r4+, r8
- mov.l @r4+, r9
- mov.l @r4+, r10
- mov.l @r4+, r11
- mov.l @r4+, r12
- mov.l @r4+, r13
- mov.l @r4+, r14
- mov.l @r4+, r15
- lds.l @r4+, pr
- mov r5, r0
- tst r0, r0
- bf 1f
- mov #1, r0 ! in case val==0
-1: rts
- nop
-

```

Index: linux-2.6.14/arch/sh/kernel/kgdb-jmp.S

```

=====
--- /dev/null
+++ linux-2.6.14/arch/sh/kernel/kgdb-jmp.S
@@ -0,0 +1,32 @@
+#include <linux/linkage.h>
+
+ENTRY(kgdb_fault_setjmp)
+ add #(9*4), r4
+ sts.l pr, @-r4
+ mov.l r15, @-r4
+ mov.l r14, @-r4
+ mov.l r13, @-r4
+ mov.l r12, @-r4
+ mov.l r11, @-r4
+ mov.l r10, @-r4
+ mov.l r9, @-r4
+ mov.l r8, @-r4
+ rts
+ mov #0, r0
+
+ENTRY(kgdb_fault_longjmp)
+ mov.l @r4+, r8
+ mov.l @r4+, r9
+ mov.l @r4+, r10
+ mov.l @r4+, r11
+ mov.l @r4+, r12
+ mov.l @r4+, r13
+ mov.l @r4+, r14
+ mov.l @r4+, r15

```

```
+ lds.l @r4+, pr
+ mov r5, r0
+ tst r0, r0
+ bf 1f
+ mov #1, r0
+1: rts
+ nop
```

Index: linux-2.6.14/arch/sh/kernel/kgdb_stub.c

```
=====
--- linux-2.6.14.orig/arch/sh/kernel/kgdb_stub.c
+++ /dev/null
@@ -1,1491 +0,0 @@
-/*
- * May be copied or modified under the terms of the GNU General Public
- * License. See linux/COPYING for more information.
- *
- * Contains extracts from code by Glenn Engel, Jim Kingdon,
- * David Grothe <dave@gcom.com>, Tigran Aivazian <tigran@sco.com>,
- * Amit S. Kale <akale@veritas.com>, William Gatliff <bgat@open-widgets.com>,
- * Ben Lee, Steve Chamberlain and Benoit Miller <fulg@iname.com>.
- *
- * This version by Henry Bell <henry.bell@st.com>
- * Minor modifications by Jeremy Siegel <jsiegel@mvista.com>
- *
- * Contains low-level support for remote debug using GDB.
- *
- * To enable debugger support, two things need to happen. A call to
- * set_debug_traps() is necessary in order to allow any breakpoints
- * or error conditions to be properly intercepted and reported to gdb.
- * A breakpoint also needs to be generated to begin communication. This
- * is most easily accomplished by a call to breakpoint() which does
- * a trapa if the initialisation phase has been successfully completed.
- *
- * In this case, set_debug_traps() is not used to "take over" exceptions;
- * other kernel code is modified instead to enter the kgdb functions here
- * when appropriate (see entry.S for breakpoint traps and NMI interrupts,
- * see traps.c for kernel error exceptions).
- *
- * The following gdb commands are supported:
- *
- * Command Function Return value
- *
- * g return the value of the CPU registers hex data or ENN
- * G set the value of the CPU registers OK or ENN
- *
- * mAA..AA,LLLL Read LLLL bytes at address AA..AA hex data or ENN
- * MAA..AA,LLLL: Write LLLL bytes at address AA..AA OK or ENN
- * XAA..AA,LLLL: Same, but data is binary (not hex) OK or ENN
- *
- * c Resume at current address SNN ( signal NN)
- * cAA..AA Continue at address AA..AA SNN
```

- * CNN; Resume at current address with signal SNN
- * CNN;AA..AA Resume at address AA..AA with signal SNN
- *
- * s Step one instruction SNN
- * sAA..AA Step one instruction from AA..AA SNN
- * SNN; Step one instruction with signal SNN
- * SNNAA..AA Step one instruction from AA..AA w/NN SNN
- *
- * k kill (Detach GDB)
- *
- * d Toggle debug flag
- * D Detach GDB
- *
- * Hct Set thread t for operations, OK or ENN
- * c = 'c' (step, cont), c = 'g' (other
- * operations)
- *
- * qC Query current thread ID QCpid
- * qfThreadInfo Get list of current threads (first) m<id>
- * qsThreadInfo " " " " " (subsequent)
- * qOffsets Get section offsets Text=x;Data=y;Bss=z
- *
- * TXX Find if thread XX is alive OK or ENN
- * ? What was the last signal ? SNN (signal NN)
- * O Output to GDB console
- *
- * Remote communication protocol.
- *
- * A debug packet whose contents are <data> is encapsulated for
- * transmission in the form:
- *
- * \$ <data> # CSUM1 CSUM2
- *
- * <data> must be ASCII alphanumeric and cannot include characters
- * '\$' or '#'. If <data> starts with two characters followed by
- * ':', then the existing stubs interpret this as a sequence number.
- *
- * CSUM1 and CSUM2 are ascii hex representation of an 8-bit
- * checksum of <data>, the most significant nibble is sent first.
- * the hex digits 0-9,a-f are used.
- *
- * Receiver responds with:
- *
- * + - if CSUM is correct and ready for next packet
- * - - if CSUM is incorrect
- *
- * Responses can be run-length encoded to save space. A '*' means that
- * the next character is an ASCII encoding giving a repeat count which
- * stands for that many repetitions of the character preceding the '*'.
- * The encoding is n+29, yielding a printable character where n >=3
- * (which is where RLE starts to win). Don't use an n > 126.

```

- *
- * So "0* " means the same as "0000".
- */
-
-#include <linux/string.h>
-#include <linux/kernel.h>
-#include <linux/sched.h>
-#include <linux/smp.h>
-#include <linux/spinlock.h>
-#include <linux/delay.h>
-#include <linux/linkage.h>
-#include <linux/init.h>
-
-#include <asm/system.h>
-#include <asm/current.h>
-#include <asm/signal.h>
-#include <asm/pgtable.h>
-#include <asm/ptrace.h>
-#include <asm/kgdb.h>
-
-#ifdef CONFIG_SH_KGDB_CONSOLE
-#include <linux/console.h>
-#endif
-
-/* Function pointers for linkage */
-kgdb_debug_hook_t *kgdb_debug_hook;
-kgdb_bus_error_hook_t *kgdb_bus_err_hook;
-
-int (*kgdb_getchar)(void);
-void (*kgdb_putchar)(int);
-
-static void put_debug_char(int c)
-{
- if (!kgdb_putchar)
- return;
- (*kgdb_putchar)(c);
-}
-static int get_debug_char(void)
-{
- if (!kgdb_getchar)
- return -1;
- return (*kgdb_getchar)();
-}
-
-/* Num chars in in/out bound buffers, register packets need NUMREGBYTES * 2 */
-#define BUFMAX 1024
-#define NUMREGBYTES (MAXREG*4)
-#define OUTBUFMAX (NUMREGBYTES*2+512)
-
-enum regs {
- R0 = 0, R1, R2, R3, R4, R5, R6, R7,

```

```

- R8, R9, R10, R11, R12, R13, R14, R15,
- PC, PR, GBR, VBR, MACH, MACL, SR,
- /* */
- MAXREG
- };
-
-static unsigned int registers[MAXREG];
-struct kgdb_regs trap_registers;
-
-char kgdb_in_gdb_mode;
-char in_nmi; /* Set during NMI to prevent reentry */
-int kgdb_nofault; /* Boolean to ignore bus errs (i.e. in GDB) */
-int kgdb_enabled = 1; /* Default to enabled, cmdline can disable */
-int kgdb_halt;
-
-/* Exposed for user access */
-struct task_struct *kgdb_current;
-unsigned int kgdb_g_imask;
-int kgdb_trapa_val;
-int kgdb_excode;
-
-/* Default values for SCI (can override via kernel args in setup.c) */
-#ifndef CONFIG_KGDB_DEFPORT
-#define CONFIG_KGDB_DEFPORT 1
-#endif
-
-#ifndef CONFIG_KGDB_DEFBAUD
-#define CONFIG_KGDB_DEFBAUD 115200
-#endif
-
-#if defined(CONFIG_KGDB_DEFPARITY_E)
-#define CONFIG_KGDB_DEFPARITY 'E'
-#elif defined(CONFIG_KGDB_DEFPARITY_O)
-#define CONFIG_KGDB_DEFPARITY 'O'
-#else /* CONFIG_KGDB_DEFPARITY_N */
-#define CONFIG_KGDB_DEFPARITY 'N'
-#endif
-
-#ifdef CONFIG_KGDB_DEFBITS_7
-#define CONFIG_KGDB_DEFBITS '7'
-#else /* CONFIG_KGDB_DEFBITS_8 */
-#define CONFIG_KGDB_DEFBITS '8'
-#endif
-
-/* SCI/UART settings, used in kgdb_console_setup() */
-int kgdb_portnum = CONFIG_KGDB_DEFPORT;
-int kgdb_baud = CONFIG_KGDB_DEFBAUD;
-char kgdb_parity = CONFIG_KGDB_DEFPARITY;
-char kgdb_bits = CONFIG_KGDB_DEFBITS;
-
-/* Jump buffer for setjmp/longjmp */

```

```

-static jmp_buf rem_com_env;
-
-/* TRA differs sh3/4 */
-#if defined(CONFIG_CPU_SH3)
-#define TRA 0xfffffd0
-#elif defined(CONFIG_CPU_SH4)
-#define TRA 0xff000020
-#endif
-
-/* Macros for single step instruction identification */
-#define OPCODE_BT(op) (((op) & 0xff00) == 0x8900)
-#define OPCODE_BF(op) (((op) & 0xff00) == 0x8b00)
-#define OPCODE_BTF_DISP(op) (((op) & 0x80) ? (((op) | 0xfffff80) << 1) : \
- ((op) & 0x7f) << 1)
-#define OPCODE_BFS(op) (((op) & 0xff00) == 0x8f00)
-#define OPCODE_BTS(op) (((op) & 0xff00) == 0x8d00)
-#define OPCODE_BRA(op) (((op) & 0xf000) == 0xa000)
-#define OPCODE_BRA_DISP(op) (((op) & 0x800) ? (((op) | 0xfffff800) << 1) : \
- ((op) & 0x7ff) << 1)
-#define OPCODE_BRAF(op) (((op) & 0xf0ff) == 0x0023)
-#define OPCODE_BRAF_REG(op) (((op) & 0x0f00) >> 8)
-#define OPCODE_BSR(op) (((op) & 0xf000) == 0xb000)
-#define OPCODE_BSR_DISP(op) (((op) & 0x800) ? (((op) | 0xfffff800) << 1) : \
- ((op) & 0x7ff) << 1)
-#define OPCODE_BSRF(op) (((op) & 0xf0ff) == 0x0003)
-#define OPCODE_BSRF_REG(op) (((op) >> 8) & 0xf)
-#define OPCODE_JMP(op) (((op) & 0xf0ff) == 0x402b)
-#define OPCODE_JMP_REG(op) (((op) >> 8) & 0xf)
-#define OPCODE_JSR(op) (((op) & 0xf0ff) == 0x400b)
-#define OPCODE_JSR_REG(op) (((op) >> 8) & 0xf)
-#define OPCODE_RTS(op) ((op) == 0xb)
-#define OPCODE_RTE(op) ((op) == 0x2b)
-
-#define SR_T_BIT_MASK 0x1
-#define STEP_OPCODE 0xc320
-#define BIOS_CALL_TRAP 0x3f
-
-/* Exception codes as per SH-4 core manual */
-#define ADDRESS_ERROR_LOAD_VEC 7
-#define ADDRESS_ERROR_STORE_VEC 8
-#define TRAP_VEC 11
-#define INVALID_INSN_VEC 12
-#define INVALID_SLOT_VEC 13
-#define NMI_VEC 14
-#define USER_BREAK_VEC 15
-#define SERIAL_BREAK_VEC 58
-
-/* Misc static */
-static int stepped_address;
-static short stepped_opcode;
-static const char hexchars[] = "0123456789abcdef";

```

```

-static char in_buffer[BUFMAX];
-static char out_buffer[OUTBUFMAX];
-
-static void kgdb_to_gdb(const char *s);
-
-#ifdef CONFIG_KGDB_THREAD
-static struct task_struct *trapped_thread;
-static struct task_struct *current_thread;
-typedef unsigned char threadref[8];
-#define BUF_THREAD_ID_SIZE 16
-#endif
-
-/* Return addr as a real volatile address */
-static inline unsigned int ctrl_inl(const unsigned long addr)
-{
- return *(volatile unsigned long *) addr;
-}
-
-/* Correctly set *addr using volatile */
-static inline void ctrl_outl(const unsigned int b, unsigned long addr)
-{
- *(volatile unsigned long *) addr = b;
-}
-
-/* Get high hex bits */
-static char highhex(const int x)
-{
- return hexchars[(x >> 4) & 0xf];
-}
-
-/* Get low hex bits */
-static char lowhex(const int x)
-{
- return hexchars[x & 0xf];
-}
-
-/* Convert ch to hex */
-static int hex(const 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);
-}
-
-/* Convert the memory pointed to by mem into hex, placing result in buf.
- Returns a pointer to the last char put in buf (null) */
-static char *mem_to_hex(const char *mem, char *buf, const int count)

```

```

- {
- int i;
- int ch;
- unsigned short s_val;
- unsigned long l_val;
-
- /* Check for 16 or 32 */
- if (count == 2 && ((long) mem & 1) == 0) {
- s_val = *(unsigned short *) mem;
- mem = (char *) &s_val;
- } else if (count == 4 && ((long) mem & 3) == 0) {
- l_val = *(unsigned long *) mem;
- mem = (char *) &l_val;
- }
- for (i = 0; i < count; i++) {
- ch = *mem++;
- *buf++ = highhex(ch);
- *buf++ = lowhex(ch);
- }
- *buf = 0;
- return (buf);
- }
-
- /* 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 written */
- static char *hex_to_mem(const char *buf, char *mem, const int count)
- {
- int i;
- unsigned char ch;
-
- for (i = 0; i < count; i++) {
- ch = hex(*buf++) << 4;
- ch = ch + hex(*buf++);
- *mem++ = ch;
- }
- return (mem);
- }
-
- /* While finding valid hex chars, convert to an integer, then return it */
- static int hex_to_int(char **ptr, int *int_value)
- {
- int num_chars = 0;
- int hex_value;
-
- *int_value = 0;
-
- while (**ptr) {
- hex_value = hex(**ptr);
- if (hex_value >= 0) {
- *int_value = (*int_value << 4) | hex_value;
- num_chars++;

```

```

- } else
- break;
- (*ptr)++;
- }
- return num_chars;
-}
-
-/* Copy the binary array pointed to by buf into mem. Fix $, #,
- and 0x7d escaped with 0x7d. Return a pointer to the character
- after the last byte written. */
-static char *ebin_to_mem(const char *buf, char *mem, int count)
-{
- for (; count > 0; count--, buf++) {
- if (*buf == 0x7d)
- *mem++ = *(++buf) ^ 0x20;
- else
- *mem++ = *buf;
- }
- return mem;
-}
-
-/* Pack a hex byte */
-static char *pack_hex_byte(char *pkt, int byte)
-{
- *pkt++ = hexchars[(byte >> 4) & 0xf];
- *pkt++ = hexchars[(byte & 0xf)];
- return pkt;
-}
-
-#ifdef CONFIG_KGDB_THREAD
-
-/* Pack a thread ID */
-static char *pack_threadid(char *pkt, threadref * id)
-{
- char *limit;
- unsigned char *altid;
-
- altid = (unsigned char *) id;
-
- limit = pkt + BUF_THREAD_ID_SIZE;
- while (pkt < limit)
- pkt = pack_hex_byte(pkt, *altid++);
- return pkt;
-}
-
-/* Convert an integer into our threadref */
-static void int_to_threadref(threadref * id, const int value)
-{
- unsigned char *scan = (unsigned char *) id;
- int i = 4;
-
-

```

```

- while (i--)
- *scan++ = 0;
-
- *scan++ = (value >> 24) & 0xff;
- *scan++ = (value >> 16) & 0xff;
- *scan++ = (value >> 8) & 0xff;
- *scan++ = (value & 0xff);
-}
-
-/* Return a task structure ptr for a particular pid */
-static struct task_struct *get_thread(int pid)
-{
- struct task_struct *thread;
-
- /* Use PID_MAX w/gdb for pid 0 */
- if (pid == PID_MAX) pid = 0;
-
- /* First check via PID */
- thread = find_task_by_pid(pid);
-
- if (thread)
- return thread;
-
- /* Start at the start */
- thread = init_tasks[0];
-
- /* Walk along the linked list of tasks */
- do {
- if (thread->pid == pid)
- return thread;
- thread = thread->next_task;
- } while (thread != init_tasks[0]);
-
- return NULL;
-}
-
-#endif /* CONFIG_KGDB_THREAD */
-
-/* Scan for the start char '$', read the packet and check the checksum */
-static void get_packet(char *buffer, int buflen)
-{
- unsigned char checksum;
- unsigned char xmitsum;
- int i;
- int count;
- char ch;
-
- do {
- /* Ignore everything until the start character */
- while ((ch = get_debug_char()) != '$');
-

```

```

- checksum = 0;
- xmitcsum = -1;
- count = 0;
-
- /* Now, read until a # or end of buffer is found */
- while (count < (buflen - 1)) {
- ch = get_debug_char();
-
- if (ch == '#')
- break;
-
- checksum = checksum + ch;
- buffer[count] = ch;
- count = count + 1;
- }
-
- buffer[count] = 0;
-
- /* Continue to read checksum following # */
- if (ch == '#') {
- xmitcsum = hex(get_debug_char()) << 4;
- xmitcsum += hex(get_debug_char());
-
- /* Checksum */
- if (checksum != xmitcsum)
- put_debug_char('-'); /* Failed checksum */
- else {
- /* Ack successful transfer */
- put_debug_char('+');
-
- /* If a sequence char is present, reply
- the sequence ID */
- if (buffer[2] == ':') {
- put_debug_char(buffer[0]);
- put_debug_char(buffer[1]);
-
- /* Remove sequence chars from buffer */
- count = strlen(buffer);
- for (i = 3; i <= count; i++)
- buffer[i - 3] = buffer[i];
- }
- }
- }
- }
- while (checksum != xmitcsum); /* Keep trying while we fail */
- }
-
- /* Send the packet in the buffer with run-length encoding */
- static void put_packet(char *buffer)
- {
- int checksum;

```

```

- char *src;
- int runlen;
- int encode;
-
- do {
- src = buffer;
- put_debug_char('$');
- checksum = 0;
-
- /* Continue while we still have chars left */
- while (*src) {
- /* Check for runs up to 99 chars long */
- for (runlen = 1; runlen < 99; runlen++) {
- if (src[0] != src[runlen])
- break;
- }
-
- if (runlen > 3) {
- /* Got a useful amount, send encoding */
- encode = runlen + ' ' - 4;
- put_debug_char(*src); checksum += *src;
- put_debug_char('*'); checksum += '*';
- put_debug_char(encode); checksum += encode;
- src += runlen;
- } else {
- /* Otherwise just send the current char */
- put_debug_char(*src); checksum += *src;
- src += 1;
- }
- }
-
- /* '#' Separator, put high and low components of checksum */
- put_debug_char('#');
- put_debug_char(highhex(checksum));
- put_debug_char(lowhex(checksum));
- }
- while ((get_debug_char()) != '+'); /* While no ack */
- }
-
- /* A bus error has occurred - perform a longjmp to return execution and
- allow handling of the error */
- static void kgdb_handle_bus_error(void)
- {
- longjmp(rem_com_env, 1);
- }
-
- /* Translate SH-3/4 exception numbers to unix-like signal values */
- static int compute_signal(const int excep_code)
- {
- int signal;
-

```

```

- switch (excep_code) {
-
- case INVALID_INSN_VEC:
- case INVALID_SLOT_VEC:
- signal = SIGILL;
- break;
- case ADDRESS_ERROR_LOAD_VEC:
- case ADDRESS_ERROR_STORE_VEC:
- signal = SIGSEGV;
- break;
-
- case SERIAL_BREAK_VEC:
- case NMI_VEC:
- signal = SIGINT;
- break;
-
- case USER_BREAK_VEC:
- case TRAP_VEC:
- signal = SIGTRAP;
- break;
-
- default:
- signal = SIGBUS; /* "software generated" */
- break;
- }
-
- return (sigval);
-}
-
-/* Make a local copy of the registers passed into the handler (bletch) */
-static void kgdb_regs_to_gdb_regs(const struct kgdb_regs *regs,
- int *gdb_regs)
-{
- gdb_regs[R0] = regs->regs[R0];
- gdb_regs[R1] = regs->regs[R1];
- gdb_regs[R2] = regs->regs[R2];
- gdb_regs[R3] = regs->regs[R3];
- gdb_regs[R4] = regs->regs[R4];
- gdb_regs[R5] = regs->regs[R5];
- gdb_regs[R6] = regs->regs[R6];
- gdb_regs[R7] = regs->regs[R7];
- gdb_regs[R8] = regs->regs[R8];
- gdb_regs[R9] = regs->regs[R9];
- gdb_regs[R10] = regs->regs[R10];
- gdb_regs[R11] = regs->regs[R11];
- gdb_regs[R12] = regs->regs[R12];
- gdb_regs[R13] = regs->regs[R13];
- gdb_regs[R14] = regs->regs[R14];
- gdb_regs[R15] = regs->regs[R15];
- gdb_regs[PC] = regs->pc;
- gdb_regs[PR] = regs->pr;

```

```

- gdb_regs[GBR] = regs->gbr;
- gdb_regs[MACH] = regs->mach;
- gdb_regs[MACL] = regs->macl;
- gdb_regs[SR] = regs->sr;
- gdb_regs[VBR] = regs->vbr;
-}
-
-/* Copy local gdb registers back to kgdb regs, for later copy to kernel */
-static void gdb_regs_to_kgdb_regs(const int *gdb_regs,
- struct kgdb_regs *regs)
-{
- regs->regs[R0] = gdb_regs[R0];
- regs->regs[R1] = gdb_regs[R1];
- regs->regs[R2] = gdb_regs[R2];
- regs->regs[R3] = gdb_regs[R3];
- regs->regs[R4] = gdb_regs[R4];
- regs->regs[R5] = gdb_regs[R5];
- regs->regs[R6] = gdb_regs[R6];
- regs->regs[R7] = gdb_regs[R7];
- regs->regs[R8] = gdb_regs[R8];
- regs->regs[R9] = gdb_regs[R9];
- regs->regs[R10] = gdb_regs[R10];
- regs->regs[R11] = gdb_regs[R11];
- regs->regs[R12] = gdb_regs[R12];
- regs->regs[R13] = gdb_regs[R13];
- regs->regs[R14] = gdb_regs[R14];
- regs->regs[R15] = gdb_regs[R15];
- regs->pc = gdb_regs[PC];
- regs->pr = gdb_regs[PR];
- regs->gbr = gdb_regs[GBR];
- regs->mach = gdb_regs[MACH];
- regs->macl = gdb_regs[MACL];
- regs->sr = gdb_regs[SR];
- regs->vbr = gdb_regs[VBR];
-}
-
-#ifdef CONFIG_KGDB_THREAD
-/* Make a local copy of registers from the specified thread */
-asm linkage void ret_from_fork(void);
- static void thread_regs_to_gdb_regs(const struct task_struct *thread,
- int *gdb_regs)
- {
- int regno;
- int *tregs;
-
- /* Initialize to zero */
- for (regno = 0; regno < MAXREG; regno++)
- gdb_regs[regno] = 0;
-
- /* Just making sure... */
- if (thread == NULL)

```

```

- return;
-
- /* A new fork has pt_regs on the stack from a fork() call */
- if (thread->thread.pc == (unsigned long)ret_from_fork) {
-
- int vbr_val;
- struct pt_regs *kregs;
- kregs = (struct pt_regs*)thread->thread.sp;
-
- gdb_regs[R0] = kregs->regs[R0];
- gdb_regs[R1] = kregs->regs[R1];
- gdb_regs[R2] = kregs->regs[R2];
- gdb_regs[R3] = kregs->regs[R3];
- gdb_regs[R4] = kregs->regs[R4];
- gdb_regs[R5] = kregs->regs[R5];
- gdb_regs[R6] = kregs->regs[R6];
- gdb_regs[R7] = kregs->regs[R7];
- gdb_regs[R8] = kregs->regs[R8];
- gdb_regs[R9] = kregs->regs[R9];
- gdb_regs[R10] = kregs->regs[R10];
- gdb_regs[R11] = kregs->regs[R11];
- gdb_regs[R12] = kregs->regs[R12];
- gdb_regs[R13] = kregs->regs[R13];
- gdb_regs[R14] = kregs->regs[R14];
- gdb_regs[R15] = kregs->regs[R15];
- gdb_regs[PC] = kregs->pc;
- gdb_regs[PR] = kregs->pr;
- gdb_regs[GBR] = kregs->gbr;
- gdb_regs[MACH] = kregs->mach;
- gdb_regs[MACL] = kregs->macl;
- gdb_regs[SR] = kregs->sr;
-
- asm("stc vbr, %0":"=r"(vbr_val));
- gdb_regs[VBR] = vbr_val;
- return;
- }
-
- /* Otherwise, we have only some registers from switch_to() */
- tregs = (int *)thread->thread.sp;
- gdb_regs[R15] = (int)tregs;
- gdb_regs[R14] = *tregs++;
- gdb_regs[R13] = *tregs++;
- gdb_regs[R12] = *tregs++;
- gdb_regs[R11] = *tregs++;
- gdb_regs[R10] = *tregs++;
- gdb_regs[R9] = *tregs++;
- gdb_regs[R8] = *tregs++;
- gdb_regs[PR] = *tregs++;
- gdb_regs[GBR] = *tregs++;
- gdb_regs[PC] = thread->thread.pc;
- }

```

```

-#endif /* CONFIG_KGDB_THREAD */
-
-/* Calculate the new address for after a step */
-static short *get_step_address(void)
-{
- short op = *(short *) trap_registers.pc;
- long addr;
-
- /* BT */
- if (OPCODE_BT(op)) {
- if (trap_registers.sr & SR_T_BIT_MASK)
- addr = trap_registers.pc + 4 + OPCODE_BT_DISP(op);
- else
- addr = trap_registers.pc + 2;
- }
-
- /* BTS */
- else if (OPCODE_BTS(op)) {
- if (trap_registers.sr & SR_T_BIT_MASK)
- addr = trap_registers.pc + 4 + OPCODE_BT_DISP(op);
- else
- addr = trap_registers.pc + 4; /* Not in delay slot */
- }
-
- /* BF */
- else if (OPCODE_BF(op)) {
- if (!(trap_registers.sr & SR_T_BIT_MASK))
- addr = trap_registers.pc + 4 + OPCODE_BT_DISP(op);
- else
- addr = trap_registers.pc + 2;
- }
-
- /* BFS */
- else if (OPCODE_BFS(op)) {
- if (!(trap_registers.sr & SR_T_BIT_MASK))
- addr = trap_registers.pc + 4 + OPCODE_BT_DISP(op);
- else
- addr = trap_registers.pc + 4; /* Not in delay slot */
- }
-
- /* BRA */
- else if (OPCODE_BRA(op))
- addr = trap_registers.pc + 4 + OPCODE_BRA_DISP(op);
-
- /* BRAF */
- else if (OPCODE_BRAF(op))
- addr = trap_registers.pc + 4
- + trap_registers.regs[OPCODE_BRAF_REG(op)];
-
- /* BSR */
- else if (OPCODE_BSR(op))

```

```

- addr = trap_registers.pc + 4 + OPCODE_BSR_DISP(op);
-
- /* BSRF */
- else if (OPCODE_BSRF(op))
- addr = trap_registers.pc + 4
- + trap_registers.regs[OPCODE_BSRF_REG(op)];
-
- /* JMP */
- else if (OPCODE_JMP(op))
- addr = trap_registers.regs[OPCODE_JMP_REG(op)];
-
- /* JSR */
- else if (OPCODE_JSR(op))
- addr = trap_registers.regs[OPCODE_JSR_REG(op)];
-
- /* RTS */
- else if (OPCODE_RTS(op))
- addr = trap_registers.pr;
-
- /* RTE */
- else if (OPCODE_RTE(op))
- addr = trap_registers.regs[15];
-
- /* Other */
- else
- addr = trap_registers.pc + 2;
-
- kgdb_flush_icache_range(addr, addr + 2);
- return (short *) addr;
-}
-
-/* Set up a single-step. Replace the instruction immediately after the
- current instruction (i.e. next in the expected flow of control) with a
- trap instruction, so that returning will cause only a single instruction
- to be executed. Note that this model is slightly broken for instructions
- with delay slots (e.g. B[TF]S, BSR, BRA etc), where both the branch
- and the instruction in the delay slot will be executed. */
-static void do_single_step(void)
-{
- unsigned short *addr = 0;
-
- /* Determine where the target instruction will send us to */
- addr = get_step_address();
- stepped_address = (int)addr;
-
- /* Replace it */
- stepped_opcode = *(short *)addr;
- *addr = STEP_OPCODE;
-
- /* Flush and return */
- kgdb_flush_icache_range((long) addr, (long) addr + 2);

```

```

- return;
- }
-
- /* Undo a single step */
- static void undo_single_step(void)
- {
- /* If we have stepped, put back the old instruction */
- /* Use stepped_address in case we stopped elsewhere */
- if (stepped_opcode != 0) {
- *(short*)stepped_address = stepped_opcode;
- kgdb_flush_icache_range(stepped_address, stepped_address + 2);
- }
- stepped_opcode = 0;
- }
-
- /* Send a signal message */
- static void send_signal_msg(const int signum)
- {
- #ifndef CONFIG_KGDB_THREAD
- out_buffer[0] = 'S';
- out_buffer[1] = highhex(signum);
- out_buffer[2] = lowhex(signum);
- out_buffer[3] = 0;
- put_packet(out_buffer);
- #else /* CONFIG_KGDB_THREAD */
- int threadid;
- threadref thref;
- char *out = out_buffer;
- const char *tstring = "thread";
-
- *out++ = 'T';
- *out++ = highhex(signum);
- *out++ = lowhex(signum);
-
- while (*tstring) {
- *out++ = *tstring++;
- }
- *out++ = ':';
-
- threadid = trapped_thread->pid;
- if (threadid == 0) threadid = PID_MAX;
- int_to_threadref(&thref, threadid);
- pack_threadid(out, &thref);
- out += BUF_THREAD_ID_SIZE;
- *out++ = ':';
-
- *out = 0;
- put_packet(out_buffer);
- #endif /* CONFIG_KGDB_THREAD */
- }
-

```

```

-/* Reply that all was well */
-static void send_ok_msg(void)
-{
- strcpy(out_buffer, "OK");
- put_packet(out_buffer);
-}
-
-/* Reply that an error occurred */
-static void send_err_msg(void)
-{
- strcpy(out_buffer, "E01");
- put_packet(out_buffer);
-}
-
-/* Empty message indicates unrecognised command */
-static void send_empty_msg(void)
-{
- put_packet("");
-}
-
-/* Read memory due to 'm' message */
-static void read_mem_msg(void)
-{
- char *ptr;
- int addr;
- int length;
-
- /* Jmp, disable bus error handler */
- if (setjmp(rem_com_env) == 0) {
-
- kgdb_nofault = 1;
-
- /* Walk through, have m<addr>,<length> */
- ptr = &in_buffer[1];
- if (hex_to_int(&ptr, &addr) && (*ptr++ == ','))
- if (hex_to_int(&ptr, &length)) {
- ptr = 0;
- if (length * 2 > OUTBUFMAX)
- length = OUTBUFMAX / 2;
- mem_to_hex((char *) addr, out_buffer, length);
- }
- if (ptr)
- send_err_msg();
- else
- put_packet(out_buffer);
- } else
- send_err_msg();
-
- /* Restore bus error handler */
- kgdb_nofault = 0;
-}

```

```

-
-/* Write memory due to 'M' or 'X' message */
-static void write_mem_msg(int binary)
-{
- char *ptr;
- int addr;
- int length;
-
- if (setjmp(rem_com_env) == 0) {
-
- kgdb_nofault = 1;
-
- /* Walk through, have M<addr>,<length>:<data> */
- ptr = &in_buffer[1];
- if (hex_to_int(&ptr, &addr) && (*ptr++ == ','))
- if (hex_to_int(&ptr, &length) && (*ptr++ == ':')) {
- if (binary)
- ebin_to_mem(ptr, (char*)addr, length);
- else
- hex_to_mem(ptr, (char*)addr, length);
- kgdb_flush_icache_range(addr, addr + length);
- ptr = 0;
- send_ok_msg();
- }
- if (ptr)
- send_err_msg();
- } else
- send_err_msg();
-
- /* Restore bus error handler */
- kgdb_nofault = 0;
- }
-
-/* Continue message */
-static void continue_msg(void)
-{
- /* Try to read optional parameter, PC unchanged if none */
- char *ptr = &in_buffer[1];
- int addr;
-
- if (hex_to_int(&ptr, &addr))
- trap_registers.pc = addr;
- }
-
-/* Continue message with signal */
-static void continue_with_sig_msg(void)
-{
- int signal;
- char *ptr = &in_buffer[1];
- int addr;
-

```

```

- /* Report limitation */
- kgdb_to_gdb("Cannot force signal in kgdb, continuing anyway.\n");
-
- /* Signal */
- hex_to_int(&ptr, &signal);
- if (*ptr == ';')
- ptr++;
-
- /* Optional address */
- if (hex_to_int(&ptr, &addr))
- trap_registers.pc = addr;
- }
-
- /* Step message */
- static void step_msg(void)
- {
- continue_msg();
- do_single_step();
- }
-
- /* Step message with signal */
- static void step_with_sig_msg(void)
- {
- continue_with_sig_msg();
- do_single_step();
- }
-
- /* Send register contents */
- static void send_regs_msg(void)
- {
- #ifdef CONFIG_KGDB_THREAD
- if (!current_thread)
- kgdb_regs_to_gdb_regs(&trap_registers, registers);
- else
- thread_regs_to_gdb_regs(current_thread, registers);
- #else
- kgdb_regs_to_gdb_regs(&trap_registers, registers);
- #endif
-
- mem_to_hex((char *) registers, out_buffer, NUMREGBYTES);
- put_packet(out_buffer);
- }
-
- /* Set register contents – currently can't set other thread's registers */
- static void set_regs_msg(void)
- {
- #ifdef CONFIG_KGDB_THREAD
- if (!current_thread) {
- #endif
- kgdb_regs_to_gdb_regs(&trap_registers, registers);
- hex_to_mem(&in_buffer[1], (char *) registers, NUMREGBYTES);

```

```

- gdb_regs_to_kgdb_regs(registers, &trap_registers);
- send_ok_msg();
-#ifdef CONFIG_KGDB_THREAD
- } else
- send_err_msg();
-#endif
-}
-
-#ifdef CONFIG_KGDB_THREAD
-
-/* Set the status for a thread */
-void set_thread_msg(void)
-{
- int threadid;
- struct task_struct *thread = NULL;
- char *ptr;
-
- switch (in_buffer[1]) {
-
- /* To select which thread for gG etc messages, i.e. supported */
- case 'g':
-
- ptr = &in_buffer[2];
- hex_to_int(&ptr, &threadid);
- thread = get_thread(threadid);
-
- /* If we haven't found it */
- if (!thread) {
- send_err_msg();
- break;
- }
-
- /* Set current_thread (or not) */
- if (thread == trapped_thread)
- current_thread = NULL;
- else
- current_thread = thread;
- send_ok_msg();
- break;
-
- /* To select which thread for cCsS messages, i.e. unsupported */
- case 'c':
- send_ok_msg();
- break;
-
- default:
- send_empty_msg();
- break;
- }
-}

```

```

-
-/* Is a thread alive? */
-static void thread_status_msg(void)
-{
- char *ptr;
- int threadid;
- struct task_struct *thread = NULL;
-
- ptr = &in_buffer[1];
- hex_to_int(&ptr, &threadid);
- thread = get_thread(threadid);
- if (thread)
- send_ok_msg();
- else
- send_err_msg();
-}
-/* Send the current thread ID */
-static void thread_id_msg(void)
-{
- int threadid;
- threadref thref;
-
- out_buffer[0] = 'Q';
- out_buffer[1] = 'C';
-
- if (current_thread)
- threadid = current_thread->pid;
- else if (trapped_thread)
- threadid = trapped_thread->pid;
- else /* Impossible, but just in case! */
- {
- send_err_msg();
- return;
- }
-
- /* Translate pid 0 to PID_MAX for gdb */
- if (threadid == 0) threadid = PID_MAX;
-
- int_to_threadref(&thref, threadid);
- pack_threadid(out_buffer + 2, &thref);
- out_buffer[2 + BUF_THREAD_ID_SIZE] = '\0';
- put_packet(out_buffer);
-}
-
-/* Send thread info */
-static void thread_info_msg(void)
-{
- struct task_struct *thread = NULL;
- int threadid;
- char *pos;
- threadref thref;

```

```
-  
- /* Start with 'm' */  
- out_buffer[0] = 'm';  
- pos = &out_buffer[1];  
-  
- /* For all possible thread IDs - this will overrun if > 44 threads! */  
- /* Start at 1 and include PID_MAX (since GDB won't use pid 0...) */  
- for (threadid = 1; threadid <= PID_MAX; threadid++) {  
-  
- read_lock(&tasklist_lock);  
- thread = get_thread(threadid);  
- read_unlock(&tasklist_lock);  
-  
- /* If it's a valid thread */  
- if
```