gdbstub.txt

DEBUGGING FR-V LINUX

The kernel contains a GDB stub that talks GDB remote protocol across a serial port. This permits GDB to single step through the kernel, set breakpoints and trap exceptions that happen in kernel space and interrupt execution. It also permits the NMI interrupt button or serial port events to jump the kernel into the debugger.

On the CPUs that have on-chip UARTs (FR400, FR403, FR405, FR555), the GDB stub hijacks a serial port for its own purposes, and makes it generate level 15 interrupts (NMI). The kernel proper cannot see the serial port in question under these conditions.

On the MB93091-VDK CPU boards, the GDB stub uses UART1, which would otherwise be /dev/ttyS1. On the MB93093-PDK, the GDB stub uses UART0. Therefore, on the PDK there is no externally accessible serial port and the serial port to which the touch screen is attached becomes /dev/ttyS0.

Note that the GDB stub runs entirely within CPU debug mode, and so should not incur any exceptions or interrupts whilst it is active. In particular, note that the clock will lose time since it is implemented in software.

KERNEL PREPARATION

Firstly, a debuggable kernel must be built. To do this, unpack the kernel tree and copy the configuration that you wish to use to .config. Then reconfigure the following things on the "Kernel Hacking" tab:

(*) "Include debugging information"

    Set this to "Y". This causes all C and Assembly files to be compiled to include debugging information.

(*) "In-kernel GDB stub"

    Set this to "Y". This causes the GDB stub to be compiled into the kernel.

(*) "Immediate activation"

    Set this to "Y" if you want the GDB stub to activate as soon as possible and wait for GDB to connect. This allows you to start tracing right from the beginning of start_kernel() in init/main.c.

(*) "Console through GDB stub"

    Set this to "Y" if you wish to be able to use "console=gdb0" on the command line. That tells the kernel to pass system console messages to GDB (which then prints them on its standard output). This is useful when debugging the serial drivers that'd otherwise be used to pass console

messages to the outside world.

Then build as usual, download to the board and execute. Note that if "Immediate activation" was selected, then the kernel will wait for GDB to attach. If not, then the kernel will boot immediately and GDB will have to interrupt it or wait for an exception to occur before doing anything with the kernel.

========================
KERNEL DEBUGGING WITH GDB
========================

Set the serial port on the computer that's going to run GDB to the appropriate baud rate. Assuming the board's debug port is connected to ttyS0/COM1 on the computer doing the debugging:

```
stty -F /dev/ttyS0 115200
```

Then start GDB in the base of the kernel tree:

```
frv-uclinux-gdb linux           [uClinux]
```

Or:

```
frv-uclinux-gdb vmlinux         [MMU linux]
```

When the prompt appears:

```
GNU gdb frv-031024
Copyright 2003 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you
are
welcome to change it and/or distribute copies of it under certain
conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for
details.
This GDB was configured as "--host=i686-pc-linux-gnu
--target=frv-uclinux"...
(gdb)
```

Attach to the board like this:

```
(gdb) target remote /dev/ttyS0
Remote debugging using /dev/ttyS0
start_kernel () at init/main.c:395
(gdb)
```

This should show the appropriate lines from the source too. The kernel can then be debugged almost as if it's any other program.

==============================
INTERRUPTING THE RUNNING KERNEL
==============================

The kernel can be interrupted whilst it is running, causing a jump back to the GDB stub and the debugger:

(*) Pressing Ctrl-C in GDB. This will cause GDB to try and interrupt the kernel by sending an RS232 BREAK over the serial line to the GDB stub. This will (mostly) immediately interrupt the kernel and return it to the debugger.

(*) Pressing the NMI button on the board will also cause a jump into the debugger.

(*) Setting a software breakpoint. This sets a break instruction at the desired location which the GDB stub then traps the exception for.

(*) Setting a hardware breakpoint. The GDB stub is capable of using the IBAR and DBAR registers to assist debugging.

Furthermore, the GDB stub will intercept a number of exceptions automatically if they are caused by kernel execution. It will also intercept BUG() macro invocation.