Application note

Debugging uClinux kernel and applications with GNU gdb/insight

Version 1.0

RONETIX

embedded development tools

1 Introduction

This is an Application Note for debugging uClinux kernel and applications with GNU gdb/insight using the JTAG Emulator PEEDI.
The increasing popularity of high-speed 32-bit ARM based microcontrollers allowed Linux to enter the world of embedded devices. That is why the need of debugging its kernel and applications became essential. Since Linux is a true multi-process operating system, it utilizes a Memory Management Unit (MMU) to give each process a separate memory space. But the low end ARM cores (ARM7) don't have a MMU, so the conventional Linux is not applicable. Here comes the uClinux – a port of Linux targeted to the MMU-less CPU's. Since no MMU is utilized, no virtual memory space is available. This means all processes share the same memory space, therefore each process, each time is loaded to a different base address not known during compilation.

Why debugging via JTAG?

The debugging via JTAG interface does not need a monitor like program running on the target so it is non-intrusive and it uses no target resources.

Platform and tools:

For the purpose of this application note I will use:

- uClinux kernel v2.4
- GNU toolchain – gcc v2.95, gdb/insight v6.5.50
- target CPU – Atmel AT91C140

JTAG emulator – PEEDI (www.ronetix.at)

Note: Don't use GNU gdb/Insight earlier than v6.3 – it has broken task support.

2 Setting PEEDI

Apart all the common things that must be set in the target configuration file, there is another parameter that needs to be set – the base address of the **init_task_union** structure in the uClinux kernel. This structure contains the list of all the processes currently running on the target. So PEEDI will be able to show them. This address is known during compilation of the uClinux kernel, because the kernel is loaded on the same address every time. It may be obtained using the **nm** gcc tool like this:

```
[ilko@ibox spar]$ nm linux |grep init_task_union
0010c000 D init_task_union
```

Where “linux” is the uClinux elf image. Now we can set the address in the target configuration file:

```
CORE_OS = ucLinux, 0x10c000          ; init_task_union is at addr 0x10c000
```

Another thing to set is the init section of the target, which will initialize the FLASH, RAM and the system clock:

```
CORE_INIT=INIT_FLASH_PROGRAM
```

3 Setting the target

Three different images have to be programmed into the target:

- uboot – this is the bootloader
- the linux kernel
- a ROM file system image

For easy programming each image is described in different FLASH section in the configuration file, this way the **flash set** PEEDI command can be used to switch among them.

```
[ilko@ibox tftpboot]$ telnet 192.168.3.66
Trying 192.168.3.66...
Connected to 192.168.3.66.
Escape character is '^]'.

PEEDI - Powerful Embedded Ethernet Debug Interface
Copyright (c) 2005-2006 www.ronetix.at - All rights reserved
Hw:1.2, L:JTAG v1.1 Fw:1.11, SN: PD-0000-0B69-3811
------------------------------------------------------------

at91c140> flash set

*Flash #0 -> FLASH_UBOOT  (current)
 Flash #1 -> FLASH_KERNEL
 Flash #2 -> FLASH_ROMFS

at91c140> flash set 1
Selected FLASH section: FLASH_KERNEL
```

After we have selected the kernel we can program it, and then select and program the other two images:

```
at91c140> flash program
at91c140> flash set 2
Selected FLASH section: FLASH_ROMFS

at91c140> flash program
at91c140> flash set 0
Selected FLASH section: FLASH_UBOOT

at91c140> flash program
```

We have finished programming the target and now we can set the other init section, which does not initialize the target because this is done by the bootloader, but only sets the EmbeddedIce registers to break on software breakpoint pattern:

```
CORE_INIT=INIT_RUN
```

4 Debugging the kernel

To debug the kernel you need to compile it with debug information:

- run “make config” or “make menuconfig”
- enable “Customize kernel Settings"
- enable "Customize Vendor/User Settings"
- in the kernel settings, turn on "Full Symbolic/Source debugging support"
- in the vendor/user settings, turn on "build debugable libraries" and "build debugable applications".
- make clean; make dep; make

Note: If you want to debug only uClinux applications and not the kernel, it is not necessary to compile the kernel with debug information.

For easier debugging we will use this .gdbinit file:

```
echo Setting up the environment for debugging gdb with PEEDI.\n

set complaints 1
set remotetimeout 10

dir .
set prompt (peedi)


define conn
    target remote 192.168.3.66:2000
    info target
end


define fast
    set remote memory-write-packet-size fixed
    set remote memory-write-packet-size 16384
    set download-write-size 16128
end

# start connect command
#----------------------
conn
fast
load
```

After the kernel is built you can start gdb/insight loading it:

```
[ilko@ibox spar]$ arm-elf-gdb linux
GNU gdb 6.5.50.20060528-cvs
Copyright (C) 2006 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=arm-elf"...
Setting up the environment for debugging gdb with PEEDI.
Remote serial target in gdb-specific protocol:
Debugging a target over a serial line.
        While running this, GDB does not access memory from...
Local exec file:
        `/home/ilko/work/spar/linux', file type elf32-littlearm.
        Entry point: 0x8000
        0x00008000 - 0x00014000 is .init
        0x00014000 - 0x0010bba0 is .text
        0x0010c000 - 0x00116d0c is .data
        0x00116d20 - 0x0013a2f0 is .bss

(peedi)
```

Following the .gdbinit file, gdb will automatically connect to PEEDI and next we can load the image

```
(peedi) load
Loading section .init, size 0xc000 lma 0x8000
Loading section .text, size 0xf7ba0 lma 0x14000
Loading section .data, size 0xad0c lma 0x10c000
Start address 0x8000, load size 1108140
Transfer rate: 1754159 bits/sec, 15830 bytes/write.
```

To start the uClinux kernel just type:

```
(peedi) continue
Continuing.
```

At this point you have kicked the kernel into life and it will boot. You can now break into the debugger and add some break points to debug your code.

5 Debugging an Application

To debug an uClinux application you need first to prepare the application – at the beginning of the application put a software breakpoint pattern – this will cause the program to stop here when started:

```
int main(void)
{
        // set software breakpoint pattern, must be the same as in config file for PEEDI
        // this will cause the program to stop here
        //================================================================================
        asm(".long 0xDFFFDFFF");

        printf("Hello world\r\n");
```

Compile the application again, make the ROMFS and program it into the target.
Boot the uClinux kernel from the flash or from RAM.
You can start your application from the serial console or from a telnet session.
The target will stop at the line with **asm(".long 0xDFFFDFFF");**

The gdb command “info threads” gives you the information about the start addresses of the .text, .data and .bss sections, which is necessary for the loading the symbolic information:

```
(peedi) c
Continuing.
During symbol reading, misplaced first line number at 0x77810 for 'romfs_checksum'.

Program received signal SIGTRAP, Trace/breakpoint trap.
0x00f6918c in ?? ()
(peedi) info threads
  11 thread 18 (spar: 0xF64040 -s .data 0xF4A004 -s .bss 0xF53104)  0x00f6918c in ?? ()
  10 thread 17 (sh: 0xF89040 -s .data 0xFA1004 -s .bss 0xFA40B4)  0x00f6918c in ?? ()
  9 thread 16 (telnetd: 0xFAC040 -s .data 0xFA8004 -s .bss 0xFA9984)  0x00f6918c in ?? ()
  8 thread 15 (inetd: 0xFB8040 -s .data 0xFD9004 -s .bss 0xFDA544)  0x00f6918c in ?? ()
  7 thread 7 (mtdblockd)  0x00f6918c in ?? ()
  6 thread 6 (kupdated)  0x00f6918c in ?? ()
  5 thread 5 (bdflush)  0x00f6918c in ?? ()
  4 thread 4 (kswapd)  0x00f6918c in ?? ()
  3 thread 3 (ksoftirqd_CPU0)  0x00f6918c in ?? ()
  2 thread 2 (keventd)  0x00f6918c in ?? ()
  1 thread 1 (init: 0xFF0040 -s .data 0xFC4004 -s .bss 0xFC4F44)  0x00f6918c in ?? ()
```

Spar is the the thread we are interested in. Now we know the addresses of the sections that we need and we can load the executable file with corrected addresses of symbolic information:

```
 (peedi) add-symbol-file spar.gdb 0xF64040 -s .data 0xF4A004 -s .bss 0xF53104
add symbol table from file "spar.gdb" at
        .text_addr = 0xf64040
        .data_addr = 0xf4a004
        .bss_addr = 0xf53104
(y or n) y
Reading symbols from /home/ilko/spar/spar.gdb...done.
```

From this point on you can set breakpoints where you need and debug the application normally.

Keep in mind that the target executes several threads switching among them, so if you forcibly stop it there is guarantee that it will stop when executing the debugged thread. Because of this reason, it is recommended to use breakpoints to stop the target at desired point of the debugged thread.

6 Appendix

This is the complete PEEDI target configuration file used for this article:

```
;------------------------------------------------------------------------
;
;                               peedi.cfg
;                               ---------
;
;           PEEDI target configuration file for AT91C140 microcontroller
;
;
;   Ronetix
;
;   Supported devices   : Atmel AT91C140
;
;   Revision            : 1.0
;
;   Date                : Mai 28, 2006
;
;------------------------------------------------------------------------

;------------------------------------------------------------------------
; The following section contains licenses that are required for PEEDI
; to operate.
;
; These licenses must be filled before using this file.
;
;   Example:
;       [LICENSE]
;       KEY         = UPDATE_24MAY2007, 1111-1111-1111-1
;       KEY         = ARM7,             2222-2222-2222-2
;
; The minimum required licenses are provided when PEEDI is purchased
; and are printed on the bottom side of PEEDI.
;

[LICENSE]
KEY = UPDATE_13SEP2006, 1111-1111-1111-1
KEY = ARM7_ARM9,        2222-2222-2222-2
KEY = GDB_REMOTE,       3333-3333-3333-3

[DEBUGGER]
PROTOCOL = gdb_remote               ; gdb remote
REMOTE_PORT = 2000                  ; TCP/IP port
GDB_READ_INGNORE_TIME = 3000        ; time in ms

[TARGET]
PLATFORM = ARM

[PLATFORM_ARM]
JTAG_CHAIN = 4                       ; list of IR lengths of all TAP in the JTAG chain
JTAG_CLOCK = 10, 16000               ; JTAG Clock in [kHz] - 10kHz init; 16MHz normal work
TRST_TYPE = PUSHPULL                 ; type of TRST output: OPENDRAIN or PUSHPULL
RESET_TIME = 5                       ; length of RESET pulse in ms
CORE_STARTUP_MODE=RESET              ; if RESET than no code is executed after reset
CORE = ARM7TDMI                      ; TAP 0 is an ARM7TDMI
CORE_BREAK_PATTERN = 0xDFFFDFFF      ; software breakpoint pattern
CORE_BREAKMODE = soft                ; breakpoint mode:
                                     ; soft - software breakpiont
                                     ; hard - use hardware breakpoints instead of software

CORE_INIT=INIT_RUN                   ; use this section to debug kernel/allpication
; CORE_INIT=INIT_FLASH_PROGRAM       ; use this init section to initialize the chip select
CORE_FLASH0=FLASH_UBOOT
CORE_FLASH1=FLASH_KERNEL
CORE_FLASH2=FLASH_ROMFS

CORE_ENDIAN=little
CORE_WORKSPACE_ADDR=0xFC000000       ; PEEDI: start address of workspace for flash programmer
CORE0_WORKSPACE_LEN=0x2600           ; length of workspace in bytes

CORE_OS = ucLinux, 0x10c000          ; init_task_union is at addr 0x10c000

[INIT_RUN]
break add 0xFFFFFFFF
go
;wait 6000
;halt

[INIT_FLASH_PROGRAM]
memory write 0xff004000 0x4000207d
memory write 0xff004004 0x404020bd
memory write 0xff004008 0x80000001
memory write 0xff00400C 0x80000001
memory write 0xff000000 0x000000            ; CPU_CLOCK=240MHz/6
```

```
[FLASH_UBOOT]
CHIP=AT49BV6416
ACCESS_METHOD=AGENT
CHECK_ID=YES
CHIP_WIDTH=16
CHIP_COUNT=1
BASE_ADDR=0x40000000
FILE_NAME="tftp://192.168.3.1/uboot.bin"
FILE_ADDR=0x40000000
FILE_FORMAT=BIN
AUTO_ERASE=YES

[FLASH_KERNEL]
CHIP=AT49BV6416
ACCESS_METHOD=AGENT
CHECK_ID=YES
CHIP_WIDTH=16
CHIP_COUNT=1
BASE_ADDR=0x40000000
FILE_NAME="tftp://192.168.3.1/linux.bin"
FILE_ADDR=0x40010000
FILE_FORMAT=BIN
AUTO_ERASE=YES

[FLASH_ROMFS]
CHIP=AT49BV6416
ACCESS_METHOD=AGENT
CHECK_ID=YES
CHIP_WIDTH=16
CHIP_COUNT=1
BASE_ADDR=0x40000000
FILE_NAME="tftp://192.168.3.1/romdisk.bin"
FILE_ADDR=0x401F0000
FILE_FORMAT=BIN
AUTO_ERASE=YES
```