How to Use the MSP430 with Linux

Brian A. Carter

March 3, 2008

1 Introduction

The MSP430-F2013 ez430 USB sticks use a JTAG interface to debug the MSP430 MCU. They have a USB-to-serial adapter built in that will let the devices show up as a serial port in Linux.

2 Kernel Support

Luckily, newish kernels have a driver built in that lets us do exactly this: `ti_usb_3410_5052`. To get this guy on, run `make menuconfig` and navigate to "Device Drivers," "USB Support," and "USB Serial Converter Support," then turn on "USB TI 3410/5052 Serial Driver." But before you run `make`, we have to make a two-line fix to our kernel source.

Switch to `drivers/usb/serial` and edit `ti_usb_3410_5052.c`. Jump to around line 454 where we get some code similar to the following:

```
...
/* 3410 must be reset, 5052 resets itself */
if (tdev->td_is_3410) {
    msleep_interruptible(100);
    usb_reset_device(dev);
}

status = -ENODEV;
goto free_tdev;
...
```

Since the kernel driver doesn't support this version of the MSP430, we need to switch the bad status code with a good one. Replace `status = -ENODEV;` with the following:

```
status = 0x01e;
```

Next, scroll down a few lines to find something like the following:

```
...
/* the second configuration must be set (in sysfs by hotplug script) */
if (dev->actconfig->desc.bConfigurationValue == TI_BOOT_CONFIG) {
    status = -ENODEV;
    goto free_tdev;
}
...
```

Again, we need to replace `status = -ENODEV;` with the following:

```
(void) usb_driver_set_configuration(dev, TI_ACTIVE_CONFIG);
status = 0xa1b;
```

Now save the file and go ahead and build the kernel. The changes we made should not have any effect on whether or not the kernel compiles.

After compiling is finished, reboot into the new kernel and, if necessary, load the `ti_usb_3410_5052` module. Get to a terminal and pop in the USB device. We need to make sure the driver mapped it to a device, so check the output of `dmesg | tail`. We want something like the following:

```
...
[320714.045050] ti_usb_3410_5052: probe of 4-1:1.0 failed with error -5
[320714.158916] usb 4-1: new full speed USB device using uhci_hcd and address 3
[320714.385958] usb 4-1: configuration #1 chosen from 2 choices
[320714.388989] ti_usb_3410_5052 4-1:1.0: TI USB 3410 1 port adapter converter d
etected
[320714.389002] ti_usb_3410_5052: probe of 4-1:1.0 failed with error -5
[320714.389013] usbcore: registered new interface driver ti_usb_3410_5052
[320714.389017] /build/buildd/linux-source-2.6.22-2.6.22/drivers/usb/serial/ti_u
sb_3410_5052.c: TI USB 3410/5052 Serial Driver v0.9
[320714.538859] ti_usb_3410_5052 4-1:2.0: TI USB 3410 1 port adapter converter d
etected
[320714.539699] usb 4-1: TI USB 3410 1 port adapter converter now attached to tt
yUSB0
```

Then, go ahead and verify (if you like) that the device really exists with `ls /dev/ttyUSB0`.

3 Running `msp430-gdbproxy`

The next step is to run `msp430-gdbproxy`. We need to do this so `msp430-gdb` can communicate with the MSP430 over the JTAG interface. This is how we load stuff to the MSP430 and how we debug programs running on it. `msp430-gdbproxy` runs a small server, so we'll be connecting to `localhost` with `msp430-gdb`. Let's get `msp430-gdbproxy` running on port 2000:

```
$ gdbproxy --port=2000 --debug msp430 /dev/ttyUSB0
```

Note a couple of things, here. First, this command won't detach `msp430-gdbproxy` from the terminal, so make sure you have X or ssh or something that gives you access to more than one terminal. Second, you might have to do this as root. If so, you'll get some output like the following:

```
Remote proxy for GDB, v0.7.1, Copyright (C) 1999 Quality Quorum Inc.
MSP430 adaption Copyright (C) 2002 Chris Liechti and Steve Underwood

GDBproxy comes with ABSOLUTELY NO WARRANTY; for details
use ‘--warranty’ option. This is Open Source software. You are
welcome to redistribute it under certain conditions. Use the
’--copying’ option for details.

debug:     msp430: msp430_open()
debug: MSP430_Initialize()
error:     msp430: Could not initialize device interface (1)
```

Note that `msp430-gdbproxy` just hangs here and you have to kill it with Ctrl-C.

If, instead, all goes as planned, you should get some output like this:

```
Remote proxy for GDB, v0.7.1, Copyright (C) 1999 Quality Quorum Inc.
MSP430 adaption Copyright (C) 2002 Chris Liechti and Steve Underwood

GDBproxy comes with ABSOLUTELY NO WARRANTY; for details
use ‘--warranty’ option. This is Open Source software. You are
welcome to redistribute it under certain conditions. Use the
’--copying’ option for details.
```

```
debug:     msp430: msp430_open()
debug: MSP430_Initialize()
debug: MSP430_Configure()
debug: MSP430_VCC(3000)
debug: MSP430_Identify()
info:      msp430: Target device is a 'MSP430F20x3' (type 52)
debug: MSP430_Configure()
notice:    msp430-gdbproxy: waiting on TCP port 2000
```

I have no idea how stable `msp430-gdbproxy` is, so you probably want to leave it running in the terminal until you decide it's stable enough to run "for real." If you want it to detach, just use the `--daemon` option.

4 Compiling

To get started, compile a test program to make sure the MSP430 is working right:

```
#include <msp430x20x3.h>

int main (void) {
        /* We want P1.0 output */
        P1DIR = 1;

        /* Turn on the LED */
        P1OUT = 1;

        return 0;
}
```

Compile using some command like this:

```
$ msp430-gcc -g test.c -o test.elf
```

If you like, you can make sure that worked fine using something like `msp430-objdump`.

5 Using `msp430-gdb`

Now we're ready to communicate with the MSP430 using `msp430-gdb`. Make sure `msp430-gdbproxy` is still running, then run `msp430-gdb`. You should get some nice output like the following:

```
GNU gdb 5.1.1
Copyright 2002 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=msp430".
(gdb)
```

We need to run a couple of commands before we can use `msp430-gdb` to communicate with the MSP430:

```
(gdb) setremoteaddresssize 64
(gdb) setremotetimeout 999999
(gdb) target remote localhost:2000
Remote debugging using localhost:2000
0x0000fc00 in ?? ()
(gdb) monitor erase all
Erasing target flash - all... Erased OK
(gdb)
```

The first three commands set up `msp430-gdb` for communicating with `msp430-gdbproxy` (and thus the MSP430 via USB/serial/JTAG). The final command erases the MSP430's flash memory (which, of course, must be done before we can flash a new program).[1]

Now let's exit `msp430-gdb` so we can make a `.gdbinit` to automate things:

```
(gdb) q
The program is running.  Exit anyway? (y or n) y
$
```

[1] To investigate: should this be minimized to hopefully extend the life of these USB sticks?

Of course, there's no reason to retype those commands every time you use `msp430-gdb`, so you can create a `.gdbinit` file to automatically execute these commands every time you use `msp430-gdb`. You have two options here. First, you can create the file in your source directory (from where you'll be executing `msp430-gdb`), or second, you can create it in your home directory. `gdb` (and by extension, `msp430-gdb`) will execute the commands in your home directory's `.gdbinit` (if it exists) and then the commands in the current directory's `.gdbinit` (if it exists).

After deciding where you'd like to make your `.gdbinit`, simply put the following lines in it:

```
set remoteaddresssize 64
set remotetimeout 999999
target remote localhost:2000
monitor erase all
```

Now, start `msp430-gdb` again, this time specifying an MSP430 ELF file as the argument:

```
$ msp430-gdb test.elf
GNU gdb 5.1.1
Copyright 2002 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=msp430"...
0x0000ffff in InterruptVectors ()
Erasing target flash - all... Erased OK
(gdb)
```

`msp430-gdb` should automatically prepare the MSP430 for running your program and load the symbol table from your program (if you compiled with `-g`). Next, we have to load the program to the MSP430:

```
(gdb) lo
Loading section .text, size 0x58 lma 0xfc00
Loading section .vectors, size 0x20 lma 0xffe0
Start address 0xfc00, load size 120
Transfer rate: 960 bits in <1 sec, 30 bytes/write.
(gdb)
```

Note that you can use either the command `load`, or the shorter form, `lo`.[2]

Now, let's run the program:

```
(gdb) c
Continuing.
```

You should see the LED turn on on the MSP430 stick and you should also see lots of debug text scrolling past in your `msp430-gdbproxy` window. Of course, `gdb` doesn't understand that a program can terminate on the MSP430[3], so you need to press Ctrl-C to stop execution:

```
Program received signal SIGINT, Interrupt.
0x0000fc56 in __stop_progExec__ ()
(gdb)
```

Then, you can do whatever you want in `msp430-gdb`, like set breakpoints, print stuff, continue, step, and so on. Assuming you're ready to quit, go ahead and give `msp430-gdb` the `q` command:

```
(gdb) q
The program is running.  Exit anyway? (y or n) y
```

In its infinite wisdom, `gdb` makes sure you meant to do that. After you answer yes, you'll see the LED turn off on the USB stick. Note that these tools are fancy enough to automatically put the stick in some low-power mode after you're done running stuff on it.

6 Conclusion

Since the MSP430 is such a nicely designed device, it has a lovely removable unit at the end. The side with the USB port is the JTAG controller with a USB-to-serial adapter built in. The four pins connecting it to the other side lead to the actual MSP430 F2013 microcontroller unit, with embedded LED. Part of the MSP430 F2013 is flash memory—thus, you can actually

[2] You'll find that `gdb` lets you use any unambiguous left-first substring of any command. Run `help` in `msp430-gdb` for details. But don't expect too much help.

[3] At least I don't know how to do this. Perhaps there's a C macro or something that does this.

keep these things lying around with programs flashed to them. You'll see that you can buy three packs of T2012s (which are the same thing, but with an 8-channel 10-bit ADC added on) for only ten bucks, so they're quite a bargain. As far as I know, you get 32 kilobytes of flash memory—plenty more than enough.

7 Notes

- **N.B.: Before you remove the MSP430 device from the system, kill `msp430-gdbproxy`!** You just need to Ctrl-C `msp430-gdbproxy` or otherwise send it a `SIGINT` and it knows how to do the rest. The kernel driver does not respond nicely to having the device removed while it thinks it's still there. This looks like a sysfs bug—for now, just be careful.[4]

- Don't touch the MSP430 boards too much. They're pretty fragile. You might be better off never taking the things out of the plastic cases if you can avoid it.

- The flash memory is pretty cool. If you plug in the MSP430 stick, it automatically starts executing the last thing you flashed, even if a driver doesn't pick up the device—plain old five volts is enough to do it.[5]

- The `ti_usb_3410_5052` driver is sort of flaky sometimes. If you don't get a `/dev/ttyUSB`*n* device when you plug the device in, unplug, wait a few seconds, and replug. Monitoring `dmesg`'s output can be helpful here.

- Those pins (P1 through P14) at the end of the USB device correspond directly to the MCU's pins (that is, the F2013's pins). They are mapped as follows:

MCU Pin Number	Board Pin Number
VCC	P1
P1.0	P2

[4] If you're the kernel hacker type, go ahead and do this in a controlled experiment and take a look at `dmesg`'s output afterwards. But be prepared to reboot your kernel afterwards to get processes to die properly.

[5] I have no idea if the device debounces the input power. Plan for the possibility that the program executes hundreds of times very quickly when you plug the MSP430 in.

P1.1	P3
P1.2	P4
P1.3	P5
P1.4	P6
P1.5	P7
P1.6	P8
P1.7	P9
RST/	P10
TST	P11
XOUT	P12
XIN	P13
GND	P14

8 See Also

You'll want to see the following documents from TI:

- "MSP430x2xx Family User's Guide (Rev. D," [*sic*]
  `http://www.ti.com/litv/pdf/slau144d`
- "eZ430-F2013 Development Tool User's Guide (Rev. B),"
  `http://www.ti.com/litv/pdf/slau176b`
- "MSP430x20x1, MSP430x20x2, MSP430x20x3 Mixed Signal Microcontroller (Rev. D),"
  `http://www.ti.com/litv/pdf/slas491d`