Using the C8051F120/020 boards and EC3 debug module in a Linux Environment

Jeremy Pedersen

September 7, 2011

Notes

These instructions assume that we'll be using Ubuntu 11.04 (but the instructions SHOULD also work in newer versions of Ubuntu as well as Debian and could work under Gentoo/Fedora/OpenSUSE with some modification to account for the different package managers and package names).

This guide was written for users who prefer a Linux environment to a Windows one, but need to use the C8051F120/020 boards for either LITEC or MPS (the undergraduate embedded control courses offered at RPI). Using the C8051 in Linux has the advantage that you can use ddd (a GUI interface to the popular gdb debugger) to see where you are in your program's assembly code as you step through your C code. This is a feature missing from the SiLabs IDE.

Installing SDCC

Installing sdcc is very easy in Ubuntu (and presumably Debian also):

```
$> sudo apt-get install sdcc sdcc-lib sdcc-dev
```

And that's it! Now we can delve into the much more difficult business of getting the EC3 debug module to talk to our Linux machine.

Installing ec2drv from precompiled binaries

There is a freely available piece of software called ec2drv which allows people to load Intel hex files onto SiLabs C8051 boards from within a native Linux environment. However, development on this project has been slow and most of the code we'll need has to be pulled from the project's SVN repository, and some of it has to be modified by hand. To save time, I've put two precompiled versions of this code up online. If you are in a 32-bit Linux environment, use the 32-bit version. In a 64-bit environment, opt for the 64-bit version.

Get the 64-bit version:

`http://dl.dropbox.com/u/9131033/LITEC-MPS/ec2drv-bin64.tgz`

Get the 32-bit version:

`http://dl.dropbox.com/u/9131033/LITEC-MPS/ec2drv-bin32.tgz`

For 64-bit users:

```
$> tar -xvzf ec2drv-bin64.tgz
$> cd ec2drv64
$> sudo make install
$> sudo ldconfig
```

For 32-bit users:

```
$> tar -xvzf ec2drv-bin32.tgz
$> cd ec2drv32
$> sudo make install
$> sudo ldconfig
```

If the "make" command does not work for you, you may need to install automake:

```
$> sudo apt-get install automake
```

Installing ec2drv from source

If for some reason the above does not work for you, you can try and build the source yourself. Skip this section and move on to "Re-flashing the EC3 firmware" if you managed to install from the binaries.

The first way to install from source is via a source tarball of the version I pulled from the SVN, available here:

`http://dl.dropbox.com/u/9131033/LITEC-MPS/ec2drv.tgz`

You'll need some tools before you can successfully compile this code:

```
$> sudo apt-get install build-essential automake sdcc libreadline5-dev
   libboost-regex-dev libtool libusb-dev
```

Now, to build and install:

```
$> tar -xvzf ec2drv.tgz
$> cd ec2drv
$> autoreconf
$> make -f Makefile.cvs
$> ./configure
$> make
$> sudo make install
$> sudo ldconfig
```

If you have difficulty with that, you can install subversion and fetch the latest version of the trunk yourself:

```
$> sudo apt-get install subversion
$> svn co https://ec2drv.svn.sourceforge.net/svnroot/ec2drv/ec2drv/trunk
   ec2drv
$> cd ec2drv
$> autoreconf
$> make -f Makefile.cvs
$> ./configure
$> make
$> sudo make install
$> sudo ldconfig
```

The version I pulled from the SVN would not compile because several files were missing stdio.h in their #include statements. Here's a list of the files I needed to modify, in case the "make" step above fails with errors about printf or similar stdio functions:

```
src/debug-core/symbol.cpp
src/debug-core/symtab.cpp
src/debug-core/symtypetree.cpp
src/debug-core/target.cpp
src/debug-core/targets51.cpp
src/debug-core/targetsilabs.cpp
src/debug-core/contextmgr.cpp
src/debug-core/breakpointmgr.cpp
src/ec2tools/ec2test-any.cpp
src/newcdb/cmdcommon.cpp
src/newcdb/cmddisassemble.cpp
```

As a final note, if after the make or configure steps you think you are getting errors due to a mistake made earlier in the process, and you want to start over, you can issue the following while in the ec2drv directory:

```
$> make maintainer-clean
```

Re-flashing the EC3 firmware

Since the ec2drv project is built around a very careful reverse-engineering of the firmware on the EC3 debug module (the little USB dongle you plug in between your computer and the C8051 board), only very particular versions of this firmware work well. Since most newer dongles have an unsupported version, you'll need to reflash the dongle. I used firmware version 0x0C, which you can get via the EC2DRV sourceforge page:

```
http://ec2drv.sourceforge.net/ec3-fw-0x0c.raw
```

The process of flashing the firmware onto the debugger module is a little delicate. Note that the debug module does not have to be plugged into the C8051 board for the programming to take place. I recommend plugging the debugger in, waiting 2 or 3 seconds, and then executing the following command. If it doesn't work the first time, simply try again. The flashing seems

to take place best immediately after the debug module has been attached to the host computer. Also, it helps to be in the same directory as the .raw file when executing the command below:

```
$> sudo ec2-update-fw --port=USB --image=ec3-fw-0x0c.raw --xor
```

Note that after flashing the module may hang. As long as you make it to the point shown below, you should be in OK shape. Once polling the EC3 times out, the ec2-update-fw program will terminate on its own, so give it a minute:

```
$> sudo ec2-update-fw --port USB --image ec3-fw-0x0c.raw --xor
9728 bytes read
0x067b, 0x2303
0x1d6b, 0x0001
ec2_GetDbgInfo(0x10c4,0x8044)  1
Found EC3 debugger
Updating EC3 Firmware
Firmware update 0
5
10
15
21
26
31
36
42
47
52
57
63
68
73
78
84
89
94
100
```

```
PASSED

**********************************************************************
* WARNING: Auto detection of mode may cause initialisation sequence *
* to differ significantly from the SiLabs IDE.                       *
* In the case of problems specify --mode=C2 or --mode=JTAG           *
**********************************************************************

0x067b, 0x2303
0x1d6b, 0x0001
ec2_GetDbgInfo(0x10c4,0x8044)  1
Found EC3 debugger
ec2_reset C2
ec2-update-fw:ec2drv.c:1962:
usb_interrupt_read in read_usb returned -110 : Connection timed out
Exiting now
ec2-update-fw:ec2drv.c:1962:
usb_interrupt_read in read_usb returned -110 : Connection timed out
```

Note that it is normal during/after programming for the adapter's lights to remain off. To make sure the flashing proceeded normally, execute the following ec2device command and see if you get similar results to those shown below. The important bit is that the "Debug adapter ver" should show as 0x0c, and you should notice the lights on the EC3 come on. If this part also reports "Connection timed out" the flashing probably did NOT work and you'll have to try it over again (but before you do, just unplug the EC3 and plug it back in, because sometimes it is really tetchy). If you don't have any success trying it over again, you'll need to hook the module up to a Windows machine and use the SiLabs IDE (USB Recover Program) to de-brick the adapter and try to flash it again.

```
$> sudo ec2device --port USB
**********************************************************************
* WARNING: Auto detection of mode may cause initialisation sequence *
* to differ significantly from the SiLabs IDE.                       *
* In the case of problems specify --mode=C2 or --mode=JTAG           *
**********************************************************************
```

```
0x067b, 0x2303
0x1d6b, 0x0001
ec2_GetDbgInfo(0x10c4,0x8044)  1
Found EC3 debugger
ec2_reset C2
EC3 debugger firmware version = 0x0c
NOT C2, Trying JTAG
0x067b, 0x2303
0x1d6b, 0x0001
ec2_GetDbgInfo(0x10c4,0x8044)  1
Found EC3 debugger
ec2_reset C2
Debug adaptor ver = 0x0c
FOUND:

device : C8051F020
mode : JTAG
Flash size = 65536 bytes
Internal xram size = 4096 bytes
External bus = Yes
Read lock addr = 0xfdff
Write lock addr = 0xfdfe
Flash sector size = 512 bytes
Flash reserved top = 0xffff
Flash reserved bottom = 0xfe00
Has Scratchpad = Yes
Scratchpad start addr = 0x0000
Scratchpad length = 0x0080
Scratchpad sector size = 128 bytes
Has paged SFR = No
USB FIFO size = 0 bytes

Tested = Yes

If the above does not match your processor exactly, please contact us at :
http://sourceforge.net/tracker/?atid=775284&group_id=149627&func=browse
We need your help to discover if any sub device id's exsist in the silicon
```

```
exiting now
disconnect done
```

Now, we can move on to building our code with SDCC, then running it.

Header files and such

Both users of the C8051F120 and the C8051F020 boards will need to slightly modify the provided header c8051_SDCC.h. This header file has an include statement which calls for a "c8051f020.h" or "c8051f120.h" near the top. Note that because in Unix environments file name capitalization is important, the letters in these includes must be changed to "C8051F120.h" or "C8051F020.h" instead. This include is right at the top of the file and should be easy to find.

Also note that out of convenience I haven't bothered to add c8051_SDCC.h to sdcc's path, and I think you probably shouldn't bother to either. As such, assignments in MPS or LITEC in which .c files have an #include<c8051_SDCC.h> should be changed to #include "c8051_SDCC.h" instead, and c8051_SDCC.h should be placed locally with the .c file you want to compile.

We'll be needing just two more things to proceed: a copy of ddd (a GUI for gdb, the popular Linux/Unix debugger), and screen (a program we can use to see a raw serial terminal):

```
$> sudo apt-get install ddd screen
```

Running hw1.c from LITEC

I've provided all the files you'll need for this here:

```
http://dl.dropbox.com/u/9131033/LITEC-MPS/hw1.tgz
```

And the steps to compile and flash are pretty straightforward:

```
$> tar -xvzf hw1.tgz
$> cd hw1
$> sdcc --debug -V --nogcse hw1.c
$> sudo ec2writeflash --port USB --mode JTAG --hex hw1.ihx --eraseall
```

After flashing, you can see if your program is working by executing

```
$> screen /dev/ttyUSB0 38400
```

Changing the name of your USB to serial dongle as appropriate (or just use /dev/ttyS0 or similar for a real serial port). Unplug the power to the C8051 board and plug it back in, and some program text should pop up in your terminal, and you should now be able to enter numbers to interact with the hw1 program!

Running Hello.c from MPS

Here again are all the files you'll need:

```
http://dl.dropbox.com/u/9131033/LITEC-MPS/Hello.tgz
```

And the steps!:

```
$> tar -xvzf Hello.tgz
$> cd Hello
$> sdcc --debug -V --nogcse Hello.c
$> sudo ec2writeflash --port USB --mode JTAG --hex Hello.ihx --eraseall
```

After flashing, you can see if your program is working by executing

```
$> screen /dev/ttyUSB0 38400
```

Note that I actually modified Hello.c from the one provided in class, first to take care of the problem with #include<C8051F120.h> and then to change the baud rate when initializing the UART, because normally it would be 28,800 but the Linux USB serial drivers don't support "non-standard" baud rates. They support things like 19200 and 38400 but NOT 28800 or 14400, so avoid these when developing in Linux. This will save you the trouble of wondering why only junk prints out in screen because you are trying to use a funny baud rate and can't sync up.

In case you have concerns that the flashing did not proceed normally, here's what a "normal" flash procedure returns (i.e. one that worked):

```
port = USB
0x067b, 0x2303
0x1d6b, 0x0001
ec2_GetDbgInfo(0x10c4,0x8044)  1
Found EC3 debugger
ec2_reset C2
Debug adaptor ver = 0x0c
FOUND:
device : C8051F020
mode : JTAG

Erasing entire flash
   Loaded 2032 bytes between: 0000 to 07EF
Writing to flash
start=0x00000, end=0x007ef
done
Exiting now
Disconnect done
```

Getting Fancy

For those who really want "the cool shit", recent work on the ec2drv project has included a debugger interface for gdb which lets you directly control (and step through) your C program on the 8051 from a nice little GUI. In fact, this GUI will let you see where you are in your program in C and assembly, as you step through it, a feature which appears to be missing from the SiLabs IDE itself.

Once you've flashed the C8051F120 or C8051F020 with your program, do the following (note, you should give sdcc the –debug flag for this to work well):

```
$> sudo ddd --debugger newcdb
```

Be sure to do this from inside the directory where your program files are. Once ddd fires up, put your cursor in the command window at the bottom of the ddd window, and execute the following (type enter after each line)

```
$> set target SL51
```

```
$>set target port USB
$>set target connect
$>file test
```

Where test is replaced with the name of the program you want to run (do NOT include an extension, just give the name of the program minus any extension). After a minute of slogging through the debug stuff associated with your code, your program should show up in DDD and you can now step through line by line, set breakpoints, examine memory, or just stop and run your program. To run the program without breakpoints, hit "Run" and then "Continue" in the toolbar pane. To stop the program hit "Kill". And there you have it! Fully functional C8051F120/020 programming in Linux!

As just one parting comment, I feel I should mention that for those not already acquainted with a good text editor for editing C/Perl/C++/whatever files, emacs or vim are both great options and you should consider one of those. You'll need a decent editor if you plan on doing some of your LITEC or MPS work in Linux.

References

The webpage of an RPI student using the C8051 under Linux (tutorial may be a smidge outdated):

`http://www.hortont.com/blog/linux-and-the-silabs-c8051f02x-ec2/`

The webpage of another student (this is the first guide I found, and is the basis of my updated tutorial)

`http://homepages.rpi.edu/~langdg/litec-linux.html`

This page was handy in figuring out why the stuff I pulled from subversion wouldn't build (it was configured for an older version of libtool and had to be reconfigured):

`http://ubuntuforums.org/showthread.php?t=1505777`

Page discussing various options for C8051 users who are not in Windows (and explaining why Wine isn't a good option right now):

http://www.cygnal.org/ubb/Forum4/HTML/001038.html

Page about bricked EC3 modules, and how finicky they can be (and what to do if you brick one):

http://sourceforge.net/projects/ec2drv/forums/forum/499982/topic/1843323

More info about misbehaving debug adapters and un-bricking them:

http://www.cygnal.org/ubb/Forum4/HTML/000562.html

The ec2drv homepage, where you can find compatible firmware versions and hints about using the ec2drv tools:

http://ec2drv.sourceforge.net/quickstart.html