NAME

trloii-intro – Start-up guide for TRLO II usage.

DESCRIPTION

This guide walks through compiling the *TRLO II* companion programs, flashing the firmware onto a VU-LOM/TRIDI module and some first tests.

DOWNLOAD

The code is available with git. Currently at a GSI filesystem. (If you have no access, the author may provide a tar-ball.)

git clone /u/johansso/trloii.git

or from outside:

git clone *USERNAME*@**lx-pool.gsi.de:/u/johansso/trloii.git**

cd trloii

Find out the current tar-ball firmware file name by visiting:

www-browser http://fy.chalmers.se/~f96hajo/trloii/firmwares.html

Download (**wget**(1) may be useful), and then unpack:

tar -zxvf trloii_firmwares_*XXX*.tar.gz

This creates a directory *fw* with all bitstream files (**.rbt*), version notes / comments (*trlo_compile.txt*) and interface definitions (**_defs.h*).

NFS-HOST COMPILATION

There are not (yet) any programs that run on the host. The directories for the control programs support having executables for multiple architectures present at the same time. Doing a build of the control library on the (presumably faster) server/workstation/desktop does cut down compile time a bit as some common generated files are produced.

cd trloctrl

Find all firmwares that we have downloaded and create build directories for those:

./find_firmwares.pl

Do all the builds:

make

This will take some time. Tested on GSI and also more recent Debian.

EMBEDDED SYSTEM COMPILATION

Be in the same directory on your embedded system:

cd trloii
cd trloctrl

The above, again:

make

This will take even more time. Have patience. (Read e.g. the *vulomflash*(1) manual page.)

Then compile the *TRIMI* control program:

cd ../trimictrl
make

And the flash utility:

cd ../flash
make
cd ..

ALIASES

It simplifies use to set up aliases for the control programs. (When having the executables for several architectures it also prevents accidental use of the wrong executable - which even is fatal (hang) on some systems.)

If you are using *tsch*, then add to *~/.tcshrc*, with ~ representing the home directory on the embedded system (here with hostname **VMECPU**, and **ARCH** the architecture string):

test $HOST = *VMECPU* **&& **
alias vulomflash */trloii-path*/**flash/bin_***ARCH*/**vulomflash**
test $HOST = *VMECPU* **&& **
alias trimictrl */trloii-path*/**trimictrl/bin_***ARCH*/**trimictrl --addr=***VA*

If you are using *bash*, then add to *~/.bashrc*:

test $HOSTNAME == *VMECPU* **&& **
alias vulomflash=/*trloii-path*/**flash/bin_***ARCH*/**vulomflash**
test $HOSTNAME == *VMECPU* **&& **
alias trimictrl="/*trloii-path*/**trimictrl/bin_***ARCH*/**trimictrl --addr=***VA***"**

The *trimictrl* program like *vulomflash* is module and firmware independent. If the TRIMI is actually used, the module would for best MBS compatibility generally be placed at address *VA***=2**. As one would only have one TRIMI per processor, it makes sense to include the VME address in the alias. Please do **NOT** use module address 0x02 for this tutorial.

A helper script exist which produces such lines. To be run on the embedded system:

scripts/makealiases.sh

The shell has to be restarted to get the aliases. Easiest is probably to log out and in again.

FLASHING

For the following, let us assume that your VULOM/TRIDI module is at address 0x0*7*. This is set by the rotary switches on the module. On a VULOM with a display this should be shown in the lower right corner of the display.

First just read two registers from the module (we use the alias):

vulomflash --addr=7 --read

This should give something like:

VULOM base address: 0x**07**000000
LOG: Virtual address for VULOM/TRIDI @ VME 0x**07**000000 is 0x3002a000.
VOLUM+0 => 0x**785b9f22**
VOLUM+RANGE_REG(0x800000) => 0x**00000067**

where 0x*785b9f22* has been read from the FPGA and 0x*00000067* from the CPLD. (The CPLD handles the flash memory.) If the module runs the default priority encoder and deadtime locker, then VULOM+0 probably reads as 0x*0*. In this case it is a TRLO II, and *785b* is part of the firmware identification md5sum. The CPLD can also differ, but only in the low byte (0x67 here). If the program crashes (trap / SIGBUS), the VME address is likely wrong. Alternatively, the FPGA has not loaded any (good) firmware.

The next step is to list the contents of the flash memory of the module. The memory is divided into 8 independent ranges, which can be used to store different firmwares. After power-on, the firmware in range 0 is automatically loaded. One should therefore *only* put firmwares for production use in range 0, after having tested them from another range. If the firmware interacts badly with the VME interface, it could prevent communication with the CPLD, and thus simple repair. After each firmware image in flash is space for a text comment, which is printed thus:

vulomflash --addr=7 --readprogs

Giving the comment field for each range. If it recognizes a TRLO II comment, only one abbreviated line is output per range, e.g.:

Rng 3: TRLO II ver/vulom4b_trlo 2013-12-23 18:39:03 (9.491ns) **785b15c6** Rng 4: TRLO II vulom4b_trlo 2013-12-11 10:21:58 (9.496ns) **119bae55**

where the identification md5sum is at the end. (Note: the checksum is over the source code, not the generated .rbt bitstream file.)

Select a range where the new firmware can be stored (in the following example it is *6*). Move to the directory with the appropriate firmware:

VOLUM4: **cd fw/vulom4_trlo**

VULOM4B: **cd fw/vulom4b_trlo**

TRIDI1: **cd fw/tridi1_trlo**

Find the name of the bitstream file (**.rbt*):

ls

Beam it up:

vulomflash --addr=7 *X***logic***Y***.rbt --prog=***6***

Which should progress like this:

VULOM base address: 0x**07**000000
LOG: Virtual address for VULOM/TRIDI @ VME 0x**07**000000 is 0x3002a000.
Firmware: **vlogic_4b.rbt**
Comments: **trlo_compile.txt**
From file: firmware: 7819904 bits, comments: 323 bytes.
Performing command 'prog'...
Read 977488 bytes from range 6.
Range: 6 Erased blocks 00 - 07.
Firmware loaded (7819904 bits, 977488 bytes). 323 bytes comment.
Read 977488 bytes from range 6.
Range 6 verified **OK** (7819904 bits, 977488 bytes). 323 bytes comment.

If this went fine, one may now read all ranges again:

vulomflash --addr=7 --readprogs

The newly stored firmware should appear:

Rng 6: TRLO II ver/vulom4b_trlo 2014-01-08 16:56:47 (9.491ns) *ab66c5ad*

Tell the CPLD to load this firmware into the FPGA and restart it:

vulomflash --addr=7 --restart=6

Check to see if it restarted with the new firmware:

vulomflash --addr=7 --read

VOLUM+0 => 0x**ab661f20**
VOLUM+RANGE_REG(0x800000) => 0x**0000006e**

The new identification md5sum should appear at VOLUM+0. If it has the same value as before, but firmware flashing has succeeded, then the module most likely has a known bug in its CPLD firmware that prevents restarts from non-0 ranges. The CPLD firmware can be reprogrammed via JTAG. (A workaround is to load the FPGA firmware into range 0. This is however for the moment disabled in *vulomflash*, pending addition of a few more safety checks... Contact the author.)

cd ../..

TRLOCTRL ALIASES

Knowing the used firmware image identification, an alias for the *trloctrl* program can be set up.

Like above, either in *~/.tcshrc*:

test $HOST = *VMECPU* **&& **
alias trloctrl */trloii-path*/**trloctrl/fw_***FWID_ZZ***/bin_***ARCH***/trlo_ctrl **
--addr=*VA*

or in *~/.bashrc*:

test $HOSTNAME == *VMECPU* **&& **
alias trloctrl="*/trloii-path***/trloctrl/fw_***FWID_ZZ***/bin_***ARCH***/trlo_ctrl **
--addr=*VA"*

Such lines can be produced by giving the module address and firmware identifier to the helper script:

scripts/makealiases.sh *VA FWID*

TRLOCTRL USE

First test is to see *trloctrl* talk to the module:

trloctrl

which would look like:

LOG: Virtual address for TRLO II @ VME 0x**03000000** is 0x3002a000.
LOG: TRLO: MD5SUM: 0x**ab66c5ad** (CT: 52cd753f = 2014-01-08 15:56:47 UTC)

Next thing is to clear the setup registers (they are unfortunately not initialised to suitable defaults - being set to 0, basically all multiplexers source the first front-panel input; better is to be tied to a hardwired zero):

trloctrl --clear-setup

And printing the current setup:

trloctrl --print-config

will only show one item, as all setup registers that have clear-default values (0 for all but the multiplexer inputs) are suppressed:

DEADTIME_IN(1) = WIRED_ONE

(This is the deadtime input of the trigger state machine which is initialised differently by --clear-setup.)

Start a 1 kHz pulser, and route the output to the front-panel via a pulse stretcher:

**trloctrl "period(1)=1000us" "stretch(1)=50ns" **
**"GATE_DELAY(1)=PULSER(1)" **
"LEMO_OUT(1)=GATE_DELAY(1)" "ECL_OUT(1)=GATE_DELAY(1)"

Monitor the values of the scaler channels connected to each multiplexer source:

trloctrl --mux-src-scalers

One could connect a cable from the used ECL or LEMO output to an input an see the input channel counting. Abort with Ctrl-C.

Have fun!

AUTHOR

Håkan T. Johansson <f96hajo@chalmers.se>

SEE ALSO

trimictrl(1), **trloctrl**(1), **vulomflash**(1),