asynDebugDriver –

EPICS Software for FPGA programming ,VME access, and Crate Reboot

DGS EPICS software includes a driver that allows the update of FPGA firmware and direct access to VME memory space through EPICS PVs. The driver is called asynDebugDriver and its source code resides in

gretTop/9-22/dgsDrivers/dgsDriverApp/src/asynDebugDriver.cpp , asynDebugDriver.h

The Driver requires an EPICS database that defines PVs for VME access and FPGA update. This database is found in

gretTop/9-22/dgsDrivers/dgsDriverApp/Db/asynDebug.template

Any IOC that allows VME access, say with DGS Gammaware running on a PC, or FPGA update must include these lines in its startup command file.

dbLoadRecords("db/asynDebug.template","P=VME01:,R=DBG:,PORT=DBG,ADDR=0,TIMEOUT=1")

asynDebugConfig("DBG",0)

Only one EPICS database and one call to asynDebugConfig() is sufficient, regardless of how many boards exist in the system. The software relies on asynDrivers, an EPICS driver created by Mark Rivers and can be found at http://www.aps.anl.gov/epics/modules/soft/asyn

It does not matter if the crate is a trigger or digitizer crate. The asynDebugDriver will work with each type of card.

VME Access

VME Buss access is on a per-board basis. That is , a board must be initialized successfully such that vxWorks assigns a memory space to the board. All VME registers are then mapped to memory space in vxWorks. When accessing a board, the user specifies which card, then accesses registers by memory offsets relative to the first address in that board. Cards are numbered from 0,1,2, etc. regardless of

which slot they are in. When a board is initialized, the logical card number, 0,1,2,… is associated with the physical slot number 1,2,3… where the IOC is in the leftmost slot 1. In DGS dig boards are typically in slots 3,5 and are numbered cards 0,1. In DFMA the dig boards are in slots 3,4,5,6 and are numbered cards 0,1,2,3.

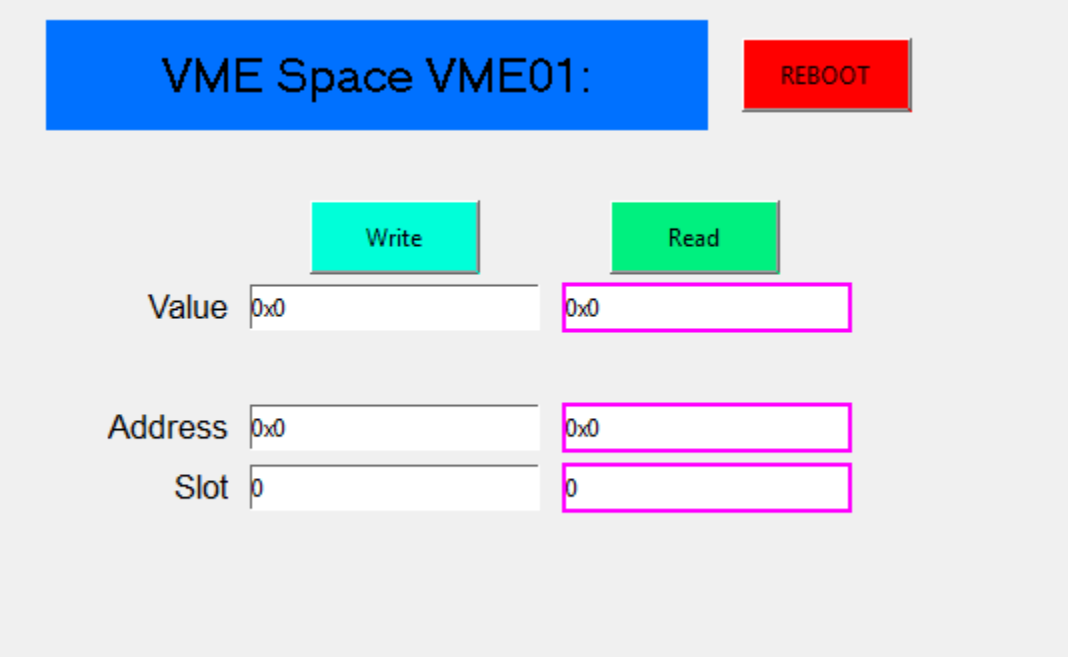
The PVs on a crate will be named based on the line in the st.cmd file:

dbLoadRecords("db/asynDebug.template","P=VME01:,R=DBG:,PORT=DBG,ADDR=0,TIMEOUT=1")

In this example the PVs will be named VME01:DBG:XXXX where XXX is the specific PV. The PORT=DBG associates the PVs with the actual C++ driver configuration call, asynDebugConfig("DBG",0). Thus by having PORTs of different names, more than one asynDebugConfig can be called, though there is no point to it.

EPICS PVs for VME Access

EPICS PVs for VME Memory space access appear in the following CSS window.



The procedure for writing to a board:

1. Set card number in PV (P)(R)dbg_card_number. “Slot” on the screen above from 0,1,2,3. This is not the physical slot in the crate from 3,4,5,6 eventhough the control is mislabeled.
2. Set Address in Hex, say to 0x900 in PV (P)(R)dbg_address.
3. Set Value to what you want to write to the address in PV (P)(R)dbg_value.
4. Hit Write button, which sets the write PV (P)(R)dbg_write_addr to 1.

To read back a register on a board.

1. Set card number from 0,1,2,3 as above.
2. Set Address in Hex, say to 0x900 as above.
3. Hit Read button, which sets read PV (P)(R)dbg_read_addr to 1.
4. The value of the register address will appear in PV (P)(R)dbg_value_RBV.

The Pvs are named the following:

- (P)(R)dbg_address
- (P)(R)dbg_address_RBV
- (P)(R)dbg_value
- (P)(R)dbg_value_RBV
- (P)(R)dbg_write_addr
- (P)(R)dbg_read_addr
- (P)(R)dbg_card_number
- (P)(R)dbg_card_number_RBV

The RBV PVs are read back values, for reading back the value in the address, or reading back what the address is set to in the driver. The RBV PV for dbg_value is really the only import read back PV, as it is needed for reading the register values from a board. The macros P and R are typically set to VME01 (or whatever crate) and DBG, respectively.

Rebooting a Crate through EPICS

Setting the PV (P)(R)rebootcrate to an integer N from 1 to infinity will reboot cause the crate to reboot in N seconds. Example

caput VME01:DBG:rebootcrate 120

Will cause crate VME01 to reboot in 2 minutes time.

Updating FPGA Flash in DGS hardware through EPICS

To update a flash in EPICS asynDebugDriver, the flash program is uploaded as an array into an EPICS waveform record. The waveform record is in byte format, as opposed to shorts or longs, and the endianness must be considered when uploading the file. A simple caput will not work properly as the endaness will likely be wrong. To improve reliability, the fpga program, often several MB long, is uploaded in blocks with successive CA puts. Typically, one sets a PV to a block size of 1024 bytes, tells the driver how long the file is, then performs many CA puts of 1k size byte arrays.

Once the FPGA program is put into the EPICS array, three PVs are provided to erase, program, and verify the flash. When the flash is verified, the flash contents are read into a second EPICS waveform PV, to enable the user to transfer flash contents to a file on a remote computer for analysis.

Currently there exists a java program that allows sending an FPGA program from a file on a disk to the IOC for FPGA update. Once the program is uploaded and programmed into the Flash, the FPGA must be reconfigured either by a PV put to a configure PV or a power cycle of the crate.

It is possible to scramble the flash making it unwritable if the programming process is interrupted. Care must be taken not to update the flash while the DGS system is running or being used by other users.

Java Software for FPGA Flash Update

To allow for command line (linux) sending of FPGA binary files to the VME boards a Java program exists that connects to PVs, sends the FPGA file, and triggers the IOC to erase, program and verify the flash. The contents of the flash after programming are then retrieved from the IOC and saved to a local file.

The Java source code is developed in Eclipse and resides on dgs1 in the path:

/home/dgs/tmadden/swWork/workspace

Two major Java classes are need and reside in subdirectories epicsClient, and plotter. The svn locatinos are in

https://svn.inside.anl.gov/repos/dgs/DGS_SW/Plotter

https://svn.inside.anl.gov/repos/dgs/DGS_SW/epicsJavaClient/trunk/epicsClient

The Plotter software has some plots to be used for the DGS oscilloscope, which is not working yet. Plotter class supplies a Javascript console to allow interactive running of Java classes. The epicsClient has code for EPICS channel access from Java. This is same Java code used in Gammaware for allowing the LabWindows PC to access EPICS. For editing and compiling the classes on DGS1, run Eclipse as

On DGS1:

Cd

Cd tmadden/eclipse

./eclipse

Choose the workspace

~dgs/tmadden/swWork/workspace

The classes should all appear in the Eclipse development system.

Procedure for FPGA Update on DGS1

To update the FPGAs, one must retrieve the latest FPGA binaries from the SVN repo and save as a file on DGS1. Currently there are working copies of FPGA binaries in

/home/dgs/tmadden/DGSDigFirmware

/home/dgs/tmadden/DGSTrigFirmware

Choose which subdir and svn update. For a specific version svn update –r 1880 for revision 1880 in the repo.

To run the Java code in interactive mode

1. cd /home/dgs/tmadden/swWork/workspace/epicsClient/src
2. java -classpath jca-2.3.5.jar:caj-1.1.9.jar:fpgasender.jar plotControl
    a. Now the javascript console is running.
3. Copy/paste these lines into the javascript console. These lines are also in the file, flashdgs.js.

```
epics.epics_init();

var fn0=new String("/home/dgs/tmadden/swWork/workspace/epicsClient/src/asynRecords.txt");

var fn1=new String("/home/dgs/tmadden/retfile.bin");

epics.connectPVs(fn0);

var digware=new String("/home/dgs/tmadden/DGSDigFirmware/Work11DGS/digitizer.bin");

var mastware=new
String("/home/dgs/tmadden/DGSTrigFirmware/MastTrigDist20121130DGS/trigger_top.bin");

var routware=new
String("/home/dgs/tmadden/DGSTrigFirmware/RoutDGSVerMainWork8/router_top.bin");
```

4. The lines above define Javascript variables digware, mastware, routware which are Java strings pointing to the firmware. The String fn0 is a filename for asynRecords.txt, which is a list of PV names. Fn1 is a filename where flash contents on board is returned and saved as a local file.

5. To send flash data to a board copy this line into the javascript console

   epics.sendFpga(digware, fn1, 1, 0,1, 1, 1);

   This line sends the firmware file defined in digware to the board. The args of the function are defined as

   Firmwarename, ret file name, Crate num, Board num, erase, program, verify

   IN the above function, we send digware to crate 1, board 0 and erase, program and verify the fpga.

6. Update the FPGA so it runs the new firmware. ON the dig boards there is a PV called GLBL:DIG:config_main_fpga that will reconfigure ALL digitizer FPGAs in DGS. This is convenient. To update a single FPGA, say in VME1, we use VME01:DIG1:config_main_fpga. Simple write a 1 to these PVs to reconfig the FPGAs. For the Trigger crate, a power cycle is needed. CSS screens have controls for these PVs. The easiest is on the Global Control screen accessible from the main DGSCommander screen. Simply hit the button for config main fpga to reconfigure ALL digitizer FPGAs. Trigger FPGAs are not affected.

To update the entire DGS system, which can be risky, with one line type at the linux command prompt

./flashdgs

To see the Javascipts for updating DGS, type at the linux command prompt

less flashdgs.js

You can then copy/paste contents of flashdgs.js into the javascript terminal.

PV Definition File for Java software

The java code defines PVs in a text file, then associates a crate can card number, address, or pv type with each PV. The DGS pvs are stored in

/home/dgs/tmadden/swWork/workspace/epicsClient/src/asynRecords.txt

Clover PVs are in asynRecordsCLO.txt.

An example line in the asynRecords.txt file is

0x000 asynAdrOut VME32:DBG:dbg_address 0 16

The fields represent

1. Unique hex address. Just make one up that is uniqie. This used to be used for assigning a VME address to a PV, but it is deprecated.
2. PV type, that is, what the PV is used for. asynAddrOut means Java will use this PV to store an address.
3. PV name.
4. Crate number, an integer. We use crate 0 for the trigger crate. Creates 1,2,3..11 for crates 1 to 11.
5. Card Number. Deprecated… Just put a 16 here.16 tells Java to use the asynAdrOut for something. Other numbers will cause Java to read the file differently.

It is best to just copy/past blocks of lines associated with the crate. The entire file must NOT end with an empty line or Java will throw an exception. When editing the file, remove all trailing empty lines at the end, and remove all empty lines in the middle of the file.