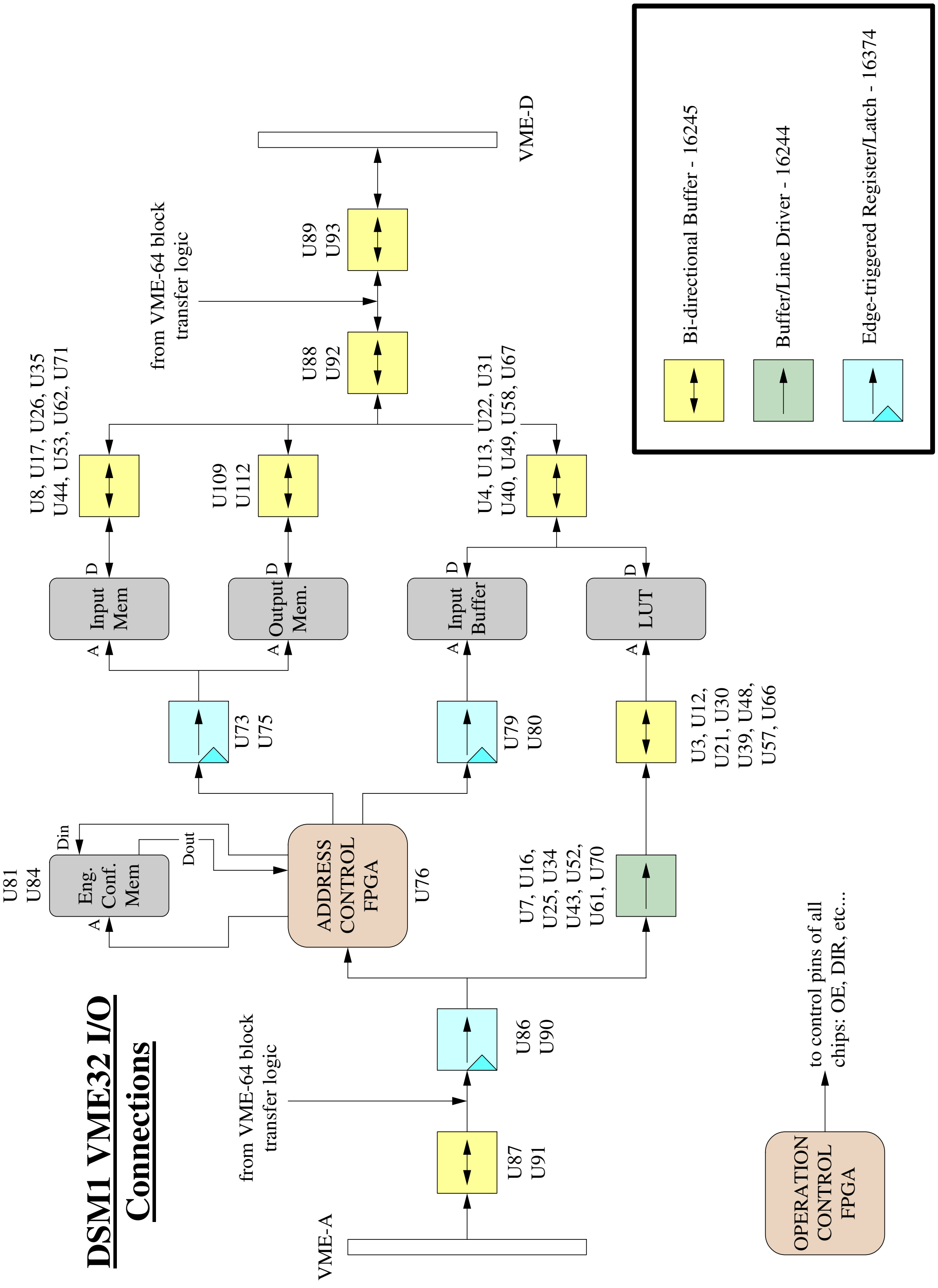
DSM1 Testing

[THIS FILE HAS BEEN EDITED FOR USE IN THE 510 STSG LAB]



DSM boards to be tested should be placed in a 9u VME crate with a 160 pin connectors on its P3 backplane. The crate should also contain an MVME 2306 processor and an RCC2 clock board with a RCF2 back-of-crate fan-out board. Just such a crate, outfitted with the necessary boards and network connections, is currently setup in the southeast corner of the STSG lab. To communicate with the DSM board to be tested you must access the processor (currently trgfe2) that is in that DSM test crate. To gain access to the trgfe2 processor log into presley2 linux machine either from the console in bldg 510 rm 235 (southeast corner) or remotely by accessing presley3.star.bnl.gov (typically using the daqlab account). Once on presley2, you can access the "trgfe2" mvme2306 processor either by connecting to it's debug/console port from the "daqserv" computone server:

```
telnet daqserv 9001
```

(where 9001 expects the silver satin cable in the mvme 2306 debug port is connected to channel 1 of the computone server, daqserv), or by telneting directly to the mvme2306 processor:

```
telnet trgfe2
```

(note: to exit either type of telnet session hold down the "control key" and hit close square bracket key, "]", then at the telnet> prompt enter “quit”. Next take the processor down to its boot mode (enter cntrl-x then hit any key to stop the processor from rebooting). Then type “p” at the boot prompt to display the current startup configuration of processor:

```
[VxWorks Boot]: p

boot device          : dc
processor number     : 0
host name            : presley2
file name            :
/home/daqlab/vxworks/wind_ppc/target/config/kern2306/vxWorks.sb
inet on ethernet (e) : 192.168.140.12:ffffff00
host inet (h)        : 192.168.140.5
user (u)             : daqlab
ftp password (pw)    : ********
flags (f)            : 0x8
target name (tn)     : trgfe2
startup script (s)   :
/home/daqlab/old_presley/Trigger/trgfe2.startup.DSMtest

[VxWorks Boot]:
```

If necessary change the startup file to

```
/home/daqlab/old_presley/Trigger/trgfe2.startup.DSMtest
```

then reboot the processor. this is a minimal startup script that loads some of the necessary object files (testdma.o but not dsm.o) needed for running the scripts that test the DSM boards.

STANDALONE TESTS:

For the standard DSM standalone tests you need to load the dsm.o library:

```
ld < dsm.o
```

(note: the alternative is the dsm_egj.o library which will be discussed later in this docuement).

the following standalone test scripts for testing the various memories and different functions of the FPGA are available (see DSM flow diagram on previous page)

```
DSM_FPGA_Config_Test(0x14000000,0x8,"config_board_14.dat",0x1)
DSM_Mem_Test(0x14000000,0x1,"mem_board_14.dat",0x0,0x0,0x1)
DSM_Addr_Test(0x14000000,0x200,"addr_board_14.dat",0x1)
DSM_FIFO_Test(0x14000000,0x200,"fifo_board_14.dat",0x1)
DSM_InOut_FPGA_Test(0x14000000,0x8,"fpga_board_14.dat",0x1)
```

in the above the first entry in each of these tests designates the VME address of the board being tested. This VME address is set by two rotary switches, SW2 & SW4, located in the upper right hand portion of the DSM board. The entry in quotes is the name given to the output log file that stores the results of the test. The unofficial standard is to include the serial number of the dsm board being tested (e.g. board_14 in the example above). there are example test files in the ~daqlab/old_presley/Trigger directory which are already setup to test either single or multiple DSM (e.g. test_dsm_board_17, test_151617_crate, etc.).

note: if an error occurs you will see it printed out to the screen. However, should you miss the error notice as it scrolls past, you can retrieve the information by simply "grep"-ing the output files:

```
grep -i err *.dat
```

in many cases is it is informative to keep the results of the tests. if this is true for your work, please create a new subdirectory that includes the name of the detector system tested and the date of the test (e.g.

```
mkdir board_14_dsm_test_sep03)
```

and move your test results there

```
mv *board_14*.dat /board_14_dsm_tests_sep03/.
```

this will keep the ~daqlab/old_presley/Trigger directory from becoming junked up with log files.

EXPANDED MEMORY TESTING:

A common failure on the aging DSM1 boards are the memory chips. If the DSM_Mem_Test above reports and error, the next step in identifying the problem chip is to use an expanded memory test that Eleanor developed. To employ this expanded memory test, you must reboot the processor (or unload the various object files by hand). Once the reboot has finished, load the dsm_egj.o library NOT dsm.o

```
ld < dsm_egj.o
```

the new version of the DSM_Mem_Test contains additional arguments to specify which memory banks are to be tested and if the engine registers are loaded:

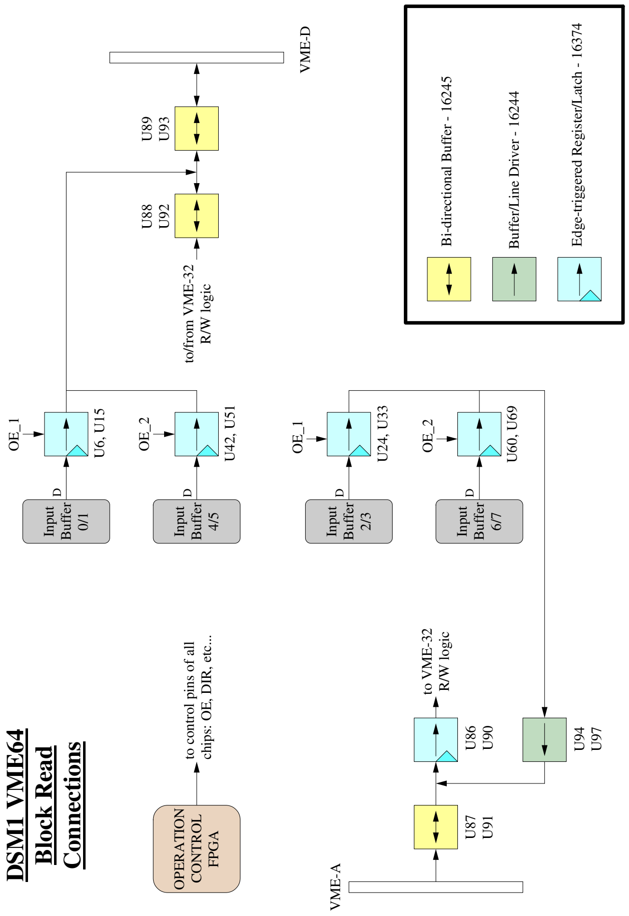
```
DSM_Mem_Test(base_address,#loops,"outfile_name.dat",onoff_mask,#e
ng_reg,eng_reg_mask,finish_flag)
```

where onoff_mask is the piece added and everything else is as it was in the old version.

onoff_mask is a bitmask, one bit for each memory in the VME memory map. The bit map is:
bit 0 = Input Buffer 1
bit 1 = input Buffer 2
bit 2 = Input Buffer 3
bit 3 = Input Buffer 4
bit 4 = LUT 1
bit 5 = LUT 2
bit 6 = LUT 3
bit 7 = LUT 4
bit 8 = Input Memory (aka Simulation Memory) 1
bit 9 = Input Memory 2
bit 10 = Input Memory 3
bit 11 = Input Memory 4
bit 12 = Output Memory
bit 13 = Engine Registers
bit 14 = Engine Configuration Memory

Useful values:
onoff_mask = 0x5f77 is used when the Engine FPA is not configured. This turns on everything EXCEPT IB4 and LUT4 (contention with FPGA) and the engine registers.
onoff_mask = 0x5fff turns on everything except the engine registers. This is useful if the big FPGA is configured with an algorithm that has no registers
onoff_mask = 0x0070 turns on ONLY the LUTS that have no problems when the FPGA is not configured. This is useful because the LUT are the only memories that can be accessed without the Address Control FPGA. If you cannot access any of these 3 memories correctly then the problem is either the Op Control FPGA (unlikely) or the VME interface chips (most likely).

CHAINBLOCK TRANSFER (DMA) TEST:



once you have at least two DSMs that pass all the standalone tests above, you will want to check that the boards will pass a full on chainblock transfer test. To do so, at the prompt on the processor, load the following object file:

```
ld < DSM_cblt_test.o
```

This file operates a 10 ms delay between every check. There is another file than can be used:

```
ld < DSM_cblt_test2.o
```

which has two changes from the previous option. (1) the DMA routines are exactly those used in trigger software.   (The previous option uses much older routines).   (2) There is an additional argument which specifies the time delay between every check – the delay is given in microsecs.   If this delay is 0, then the data checks are not made; there is no delay; and the DMA readout is executed in a tight loop. The intention is to discover if there are any rate dependencies in the DMA loop.

You need to make sure tonko's logging system is turned off the verbose mode

```
tonkoLogLevel= 1
```

(this may not be necessary for the startup file being used in the STSG lab). you will now need to initialize the boards and specify the DSM that is at the begining(first) of the chain, the end(last) of the chain, and those in the middle:

```
DSMinit("DSM_CBLT_First.dat",0x10000000)
DSMinit("DSM_CBLT_Middle.dat",0x11000000)
.
.
.
DSMinit("DSM_CBLT_Last.dat",0x1c000000)
```

again, there are files already setup that contain these calls so in most cases you can simply load one of those files (e.g.

```
<cblt_test_151617_setup
```

YOU MUST PUT THE SYSTEM INTO RUN MODE BEFORE RUNNING THIS TEST!

the DSM test crate contains an RCC2 clock modules at VME address 0x25000000. to put the DSMs in run mode, put a "1" into the run/stop register at 0x20. (note: putting a "0" in that location take the RCC2 board (and hence the DSMs) in that crate out of run mode). So to put the system in run mode simply type

```
m 0x25000020,4
```

then input a "1"

go back to the processor controlling the DSMs for the detector system that you are testing and execute the following command

```
DSM_CBLT_Crate_Test(0x20000000,0x1,3,"B14B26B96_cblt_test.dat",1)
```

the first entry for this command is the chainblock transfer address (we use 0x20000000 which must be set on SW1 and SW3 on the DSM boards), the second entry is the number of times to read out (i.e. loop thru) the system, the third entry specifies the number of DSM boards in the detector system (i.e. the crate), the fourth entry is the name of the logfile to be created, the final entry specifies whether the test is to stop at a failure (1) or to continue until the number of loops requested has been accomplished (0x0). again, errors will be printed on the screen and can be found in the logfile. For example:

```
trgfe2_DSM>
DSM_CBLT_Crate_Test(0x20000000,0x1,3,"B18B31B66_cblt_test.dat",0x
1)
DSM_Set_ERR: INFO, WARNINGs and ERRs will be printed
Address of buffer: 0x1eff200
Size of buffer is 48 bytes
Size of DMA Transfer will be 48 bytes
DMA initialised
DSM_CBLT_Crate_Test: DSM Data will be at 0x1eff200
DSM_CBLT_Crate_Test: Starting Loop 0
DSM_CBLT_Crate_Test: ERROR - Loop = 0 i = 0x18
DSM_CBLT_Crate_Test: Latched data 0x18
DSM_CBLT_Crate_Test: Read data 0x0 in DSM 1 Ch 8
value = 0 = 0x0
```

note: the board that has shown an error, "DSM 1", and the channel, "Ch 8", are reported. Board numbers start at "0" and channel numbers run from "0" to "15". so in this example board with serial number 26 is the one showing a problem. Note: there are times when the chainblock transfer fails to complete a single loop, no logfile is generated, and the error on the screen will show something like:

```
trgfe2_DSM>
DSM_CBLT_Crate_Test(0x20000000,0x1,4,"B18B31B66BXX_cblt_test.dat"
,0x1)
DSM_Set_ERR: INFO, WARNINGs and ERRs will be printed
Address of buffer: 0x1eff248
Size of buffer is 64 bytes
Size of DMA Transfer will be 64 bytes
DMA initialised
DSM_CBLT_Crate_Test: DSM Data will be at 0x1eff248
DSM_CBLT_Crate_Test: Starting Loop 0
interrupt: VME DMA Bus Error: status 0x0040026F.
0x1cd9648 (tShell): WARNING: uniDmaLib.c [line 115]: Universe DMA
semaphore timed out.
Problem with DMA. err=-1
DSM_CBLT_Crate_Test: ERROR - dma transaction failed
DSM_CBLT_Crate_Test: Loop = 0 i = 0x2
value = 0 = 0x0
```

in these cases, to find the troubled board you need to look into the MVME2306's memory buffer to determine the board that is failing to post its data. to do this use the "m" command and the memory buffer address reported on the screen to query the data that was sent. in this example:

```
trgfe2_DSM> m 0x1eff248
```

```
01eff248:  02020202-
01eff24c:  02020202-
01eff250:  02020202-
01eff254:  02020202-
01eff258:  02020202-
01eff25c:  02020202-
01eff260:  02020202-
01eff264:  02020202-
01eff268:  02020202-
01eff26c:  02020202-
01eff270:  02020202-
01eff274:  02020202-
01eff278:  00000000-
01eff27c:  00000000-
01eff280:  00000000-
```

each DSM should present 128 bits (or four 32 bit words). In this example there are twelve non-zero 32 bit words. This means that the first 3 DSMs reported their data correctly. So the trouble is either with the fourth DSM or possibly the 3rd DSM failed to relay the chainblock command to the fourth DSM.

If you use the second object file (DSM_cblt_test2.o) described above:

```
DSM_CBLT_Crate_Test(0x20000000,0x1,13,"FPW_cblt_crate_test.dat",0x1,100)
```

then the extra argument at the end (100 in this example) is the delay in microseconds between each check.

NOTE: if the chainblock transfer test fails. the DSM test crate containing the DSMs being tested MUST BE POWER CYCLED, simply rebooting the processor will not be sufficient to clear the DSMs. further, to reconfigure correctly upon reboot, the system must be placed in LOAD mode (i.e. the RCC2 run register, 0x25000020, must be set to “0”). For the DSM systems at BNL:

```
m 0x25000020,4
0
.
```

(it is best to do this before power cycling the crate that contained the DSMs which failed the chainblock transfer test)

[NOTE: if you are testing at the STAR hall and had previously removed the processor from run control, then when you are finished testing, remember to reset the processor to use its original startup script e.g.

```
/home/startrg/trg/cfg/STARTUP/dsm.startup.full
```

once the processor has been rebooted into this running configuration, re-establish the processor's communciation with run control by clicking it back using the "Show Components" GUI. Finally, take a pedAsPhys run for at least 100 events and make sure the system is able to take data. be aware of any error messages that might appear at the bottom of the DAQ monitor screen].

OUT IN LOOP TESTS

Plug a 34 conductor cable between Output (0-15) and INPUT Ch0 on the DSMI behind the DSM being tested. next place another cable between Output (16-31) and Input Ch2. run the command

```
DSM_OutIn_Loop_Test(void *base_address, int nloop, char *errname,
int finish)
```

for example:

```
DSM_OutIn_Loop_Test(0x17000000,8,"DSM21_OutIn01_loop.dat",0x1)
```

this will generate a ramp on the output that will be read and compared on the inputs. if successful, then move the cables to the next set of inputs (ch2 and ch3) and repeat the procedure using the same command. continue to 4&5 and 6&7.

TEST DATA TRANSFER FROM ONE LAYER DSM TO THE NEXT LAYER

Reconfigure the two (or more) DSMs to be tested using the following:

```
DSMinit("DSM_Ramp_Test.dat",0xYY000000)
```

where "YY" is the board address for both the source DSM(s) and the DSM that will be receiving the data. The DSM_Ramp_Test.dat file loads a ramp into the output memories and sets them to play while seting the input memories to record. once the boards are configured put the system into run mode (using the run control gui or) the RCC2 master in the L1 crate (make sure the RCC2s in the crates that house the DSMs you are using are set to "slave" mode i.e. the register at 0x08 of the respective RCC2s is set to "0").

```
0x25000020,4
1
.
```

then immediately set the system back to the ready mode

```
0x25000020,4
0
.
```

now scroll thru the relevant memories on the receiving DSM (e.g.

```
m 0x17000000,4
or
m 0x17040000,4
or
m 0x17080000,4
or
m 0x170c0000,4
```

you should see a 16 bit ramp being sent from the source boards that were configured (e.g.

```
> > BC1> m 0x15000000,4
> > 15000000:  fffdfffd-
> > 15000004:  fffefffe-
> > 15000008:  ffffffff-
> > 1500000c:  00000000-
> > 15000010:  00010001-
> > 15000014:  00020002-
> > 15000018:  00030003-
> > 1500001c:  00040004-
> > 15000020:  00050005-
> > 15000024:  00060006-
> > 15000028:  00070007-
> > 1500002c:  00080008-
> > 15000030:  00090009-
> > 15000034:  000a000a-
> > 15000038:  000b000b-
> > 1500003c:  000c000c-
> > 15000040:  000d000d-
> > 15000044:  000e000e-
> > 15000048:  000f000f-
> > 1500004c:  00100010-
> > 15000050:  00110011-.
```