NIOS Mini-Script Language

V1.0 - 2/23/06

(jml@bnl.gov)

I. User Level Language Specification

The Mini-Script Language is a simple list processor designed primarily to set and read ALTRO parameters. It has the following commands:

set:	Used to set ALTRO registers or Mini-script registers
read:	Read ALTRO/Mini-script registers
flush:	Flush's the NIOS read buffer to DDL
etrg:	Issue emulated triggers
trg:	Place RDO in running mode, to wait for real events to be issued
peek:	Peek a memory location in the NIOS address space
poke:	Poke a memory location in the NIOS address space

In addition, the language can easily extended to parse the existing DDL commands

The Mini-Script language understands the following registers. Each register has a series of arguments. Individual registers and even individual arguments within registers may be read only or write only, but the Mini-Script does not distinguish between this. If one attempts to write a RO register the command will fail silently.

Mini-script Registers -

Register	Parameters	Meaning
run_number	value	run number (Used when issuing triggers)

Altro Registers –

Register	Parameters	Meaning
k1...k3, l1...l3	value	Tail suppression filter params
vfped	vpd	first baseline: variable ped value (RO)
	fpd	first baseline: fixed pedestal offset value (RW)
zsthr	offset	zero suppression: offset added before zs
	zs_thr	zs threshold
bcthr	thr_hi	second baseline: window for active filter
	thr_lo	
trcfg	acq_start	Adjust T0 and number of timebins
	acq_end	
dpcfg	zs_on	enable zero suppression
	zs_pre	number of presamples not suppressed
	zs_post	number of postsamples not suppressed
	zs_glitch	zerosuppression mode (table 2.7 in user manual)

	bc2_on	enable baseline 2
	bc2_post	postsamples excluded from baseline 2
	bc2_pre	presamples excluded from baseline 2
	bc1_pol	polarity of acd bits
	bc1_mode	baseline 1 mode (table 2.6 in user manual)
dpcf2	pwsv	power save mode
	flt_en	enable tail suppression filter
	nbut	number of buffers in data memory
	ptrg	number of pretrigger samples
pmadd	pma	pedestal memory address
erstr	err	error status register (RO)
adevl	adevl	
trcnt	trgcnt	

SET commands:

set *register arg1 arg2 arg3 ...*

The argument list depends upon the specific register being set. Each argument is an integer which may be expressed either as a decimal number or as a hex number using the standard "c" format (0xnn). The valid register / argument lists follow.

Currently, all set commands are implemented as broadcast's to all ALTRO's and all channels. One can not set a register value for a specific ALTRO/channel.

READ commands:

read *register*

No argument list is currently accepted. (I will modify this later to allow specifying the altro/channel to read). The current behavior is to read every ALTRO. For registers that are set by channel, each channel is read.

FLUSH command:

flush

This command flushes the output buffer. Output from the mini-script programs are written to a memory buffer on the NIOS. Data is shipped via DDL to the LINUX process as a LOG event under three conditions. First, if the NIOS buffer is filled. Second, if this command is received. Third, some error conditions may cause an implicit read buffer flush.

ETRG/TRG commands:

etrg *n*	(emulated triggers)
trg *n*	(real triggers)

This command issues n events to the ALTRO's. And waits for the events to (either) finish processing or for processing to stop through some other channel. n is a 20 bit number, so limit the number of triggers to 1 million. Issuing these commands with n=0 results in a run that continues until being stopped through some other channel.

At the NIOS level, these commands start a run tagged by the run number, so typically, from the Mini-script you would issue two commands, for example:

```
set run_number 5000
trg 0
```

which would have the effect of starting run #5000 to run until stopped on some other channel. This paradigm could easily be extended to add other parameters, such as timeout's etc...

PEEK command:

peek *addr*

Peek a memory location in the NIOS address space

POKE command:

poke *addr value*

Poke a memory location in the NIOS address space.

II. Extensibility for DDL commands

The Mini-Script language is designed to easily parse & build additional commands. These commands must be defined in mscript.C in the following format

```
struct CommandSpec {
     char *spec;
     int id;
};

CommandSpec ddlCmds[] =
  {
```

```
    { "dummy1",  1 },
    { "dummy2 a:5 b:5 c:9", 2 },
    { NULL, 0 },
};
```

Each line defines a command. The first word in the *spec* string defines the command name, while the additional lines encode the commands parameters, and the number of bits allocated to each parameter. The *id* field, encodes the DDL opcode.

The construction of the parameter bit field handled by assuming that the total number of data bits is 20 + 32*n, where n is an integer. To construct the payload, first the size is calculated in bits, and this is used to calculate the number of words used for the command. The command values are then packed into the data words, with the rightmost parameter appearing in the least significant bits of the final data word.
If the number of bits is larger than 20, one should total the number of bits in all commands to be 20 + 32*n bits long. To do this, the parameter name "uu" can be used as a spacer. It is ignored while parsing commands. For example, the poke command is specified:

poke uu:20 addr:32 val:32

but it is used:

poke *addr val*

III. Mini-Script Opcode specification

Each command is represented by 1 or more 32 bit data words. The length of the command is determined by the command specification. For a given command, the length is always fixed. If a command takes up multiple words, every word after the first is made up from payload, which is specified by the "spec" definition of the command. The opcode can have two formats depending on value of bit 31.

If Bit 31 is set, the command is treated as a Mini-script command

31	30	29				25	24				20	19																			0
1	B	Mini-Opcode					Register					Payload																				

Here, B is the "broadcast" flag, which is currently unused. The register value is used only by the set & read commands. If the mini-opcode is SET/READ then the register is interpreted as the ALTRO register. If the mini-opcode is SETL/READL, then the register is interpreted as a Mini-Script register.

If Bit 31 is not set, the command is treated as a DDL-script command

31	30					25	24					19	18																	0
0	DDL Opcode						unused						Payload																	

The opcode & register values are defined in /RTS/include/DAQ1000/mscript.h

IV. Possible Future Extensions

The Mini-Script language will probably need to be extended in the future. Features that make sense are:

- Address specific altro's / channels for set commands, in particular for dealing with error conditions.
- Address specific altro/channels for read commands for more efficiency.
- Parse sector/RDO level addressing for the LINUX DDL control program to use to route the Mini-script data.
- Accept commands in the format: set register PARAM=value
    - other parameters might be set according to defaults
    - other parameters might be set by an implicit read before the set.