GLAST VME
Software User’s Guide
Startup:
source the environmental variables in the file \texttt{.tornado}

When the launcher window appears, select the target server for the machine you’re controlling, ptem1, ptem2, or ptem3.

If the target server for you’re machine isn’t displayed
On the host machine type \texttt{tgtsvr <machine>} &
tgtsvr ptem1 &
tgtsvr ptem2 &
tgtsv ptem3 &
With the cursor select the **Windsh** button.

Refer to the Wind River Tornado User’s Guide for more info on using the shell.
In the shell enter:
cd "/disk1/wind/sw/src"
ld < glastdaq.o
Memory operations:
Create a variable with the memory address to be accessed such as addr = 0x55550000
To read from that address type *addr, the result will show the address at that location.
To write to a location type *addr=<number>
Note the number is in C style radix
None = decimal
0x = hex
etc.
downloading code to the L1Trigger FPGA:
type loadtrig("filename")
note: include the full path in the filename such as :
"/disk1/wind/sw/include/CONFIG_TEST_L1T.TTF"
downloading code to the Tracker readout FPGA:
type loadtrack("filename")
note: include the full path in the filename such as :
"/disk1/wind/sw/include/CONFIG_TEST_TKRRDOUT.TTF"
Memory Map
The first 16-bits of the address must match the settings on the two dip switches on the board.

<table>
<thead>
<tr>
<th>Dip</th>
<th>HEX</th>
<th>Binary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssss</td>
<td>0000</td>
<td>0000000000</td>
<td>Tracker data FIFO Read Only 32-bits</td>
</tr>
<tr>
<td>ssss</td>
<td>0010</td>
<td>0000010000</td>
<td>Level One Trigger FIFO Read Only 32-bits</td>
</tr>
<tr>
<td>ssss</td>
<td>0020</td>
<td>0000100000</td>
<td>Command Register Read / Write 32-bits</td>
</tr>
<tr>
<td>ssss</td>
<td>0030</td>
<td>0000110000</td>
<td>Status Register Read Only 16-bits</td>
</tr>
<tr>
<td>ssss</td>
<td>0040</td>
<td>0001000000</td>
<td>Interrupter #1 Read / Write 8-bits</td>
</tr>
<tr>
<td>ssss</td>
<td>0050</td>
<td>0001010000</td>
<td>Interrupter #2 Read / Write 8-bits</td>
</tr>
<tr>
<td>ssss</td>
<td>0100</td>
<td>0100000000</td>
<td>Tracker FPGA Commands</td>
</tr>
<tr>
<td>ssss</td>
<td>0200</td>
<td>1000000000</td>
<td>L1T and Tracker Register Access</td>
</tr>
</tbody>
</table>

Command Register
Bit 0 Reset – A low resets the FPGA’s and the FIFOs
Bit 1 Trigger Counter Clear
Bit 2 Trigger Timer Clear
Bit 3 Reset Tracker
Bit 4 Sparef
Bit 5 Spare
Bit 6 Spare
Bit 7 CPU Busy
Bit 8 Tracker Data 0 – Serial data for programming the Tracker FPGA
Bit 9 Tracker N_Config – A low pulse starts the programming of the tracker FPGA
Bit 10 Tracker D-Clock – The data clock for programming the tracker FIFO
Bit 11 L1T Data 0 – Serial data for programming the L1T FPGA
Bit 12 L1T N_Config – A low pulse starts the programming of the L1T FPGA
Bit 13 L1T D-Clock – The data clock for programming the L1T FPGA
Bit 14 Spare
Bit 15 Spare
Status Register

Bit 0 – Tracker Command Busy
Bit 1 – Tracker Read Timeout
Bit 2 – CAL (L1T) readout Busy
Bit 3 – Tracker FIFO empty flag – MSB empty flag
Bit 4 – Level One Trigger Configuration Done Flag – Goes high when the L1T FPGA is programmed
Bit 5 – Level One N_Status – Goes low if there’s an error loading the FPGA program.
Bit 6 – Tracker Configuration Done Flag – Goes high when the Tracker FPGA is programmed.
Bit 7 – Tracker N_Status – Goes low if there’s an error loading the Tracker FPGA Program.
Bit 8 – Tracker FIFO LSB Empty flag
Bit 9 – Tracker FIFO LSB Half Full flag
Bit 10 – Tracker FIFO LSB Full flag
Bit 11 – Tracker FIFO BYTE 1 Empty flag
Bit 12 – Tracker FIFO BYTE 1 Half Full flag
Bit 13 – Tracker FIFO BYTE 1 Full flag
Bit 14 – Tracker FIFO BYTE 2 Empty flag
Bit 15 – Tracker FIFO BYTE 2 Half Full flag
Bit 16 – Tracker FIFO MSB Half Full Flag
Bit 17 – Tracker FIFO MSB Full Flag
Bit 18 – Tracker FIFO Byte 2 Full Flag
Bit 19 – Tracker FIFO Full From Tracker FPGA
Bit 20 – Level One Trigger FIFO LSB Empty Flag
Bit 21 – Level One Trigger FIFO LSB Half Full flag
Bit 22 – Level One Trigger FIFO LSB Full Flag
Bit 23 – Level One Trigger FIFO Byte 1 Empty Flag
Bit 24 – Level One Trigger FIFO Byte 1 Half Full flag
Bit 25 – Level One Trigger FIFO Byte 1 Full Flag
Bit 26 – Level One Trigger FIFO Byte 2 Empty Flag
Bit 27 – Level One Trigger FIFO Byte 2 Half Full Flag
Bit 28 – Level One Trigger FIFO MSB Empty Flag
Bit 29 – Level One Trigger FIFO MSB Half Full Flag
Bit 30 – Level One Trigger FIFO Byte 2 Full Flag
Bit 31 – Level One Trigger FIFO MSB Full Flag
GLAST VME Board Interrupts

The GLAST VME board has four separate interrupters, implemented as four identical MACH Programmable Logic Arrays from AMD. Figure 4 shows the block diagram of this device. There are two registers in this chip, the vector and level registers. The vector register stores the interrupt vector that is set by the processor. This is an eight-bit read-write register and is cleared on reset. The second is a three-bit write-only register that holds the interrupt level. It is also cleared to zero, and since there is no interrupt level zero, the chip is disabled on startup. Before it can be used the processor must set the appropriate level and vector.

When the interrupt strobe line goes low the level decoder places the interrupt level out on one of the IRQ lines. An external open-collector driver is required to drive the VME IRQ. This causes the processor to initiate an interrupt acknowledge cycle. The control state machine checks if it is in an interrupt cycle and that the interrupt level is the same as asserted, when the IACKIN signal goes low. If not it passes the IACKIN signal through to IACKOUT, where it is connected to the IACKIN of the next interrupter in the daisy-chain. If all of the signals match, then the chip asserts its vector onto the data lines and sets DTACK low.
### Table 4 MACHINT Addresses

<table>
<thead>
<tr>
<th>address</th>
<th>register</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Interrupt Vector</td>
</tr>
<tr>
<td>3</td>
<td>Interrupt Level</td>
</tr>
<tr>
<td>5</td>
<td>Count-down timer</td>
</tr>
</tbody>
</table>

**Figure 5 MACHINT Block Diagram**
The Internet communications functions are contained in the CEnet class, and consist of only one function:

```c
udpSend (char *databuf, int nbytes, int portnum, char *host);
```

This sends the data buffer pointed to by databuf to the host and UDP port number. If the host is left blank, then the data is broadcasted to all machines on the network.
The Level One Trigger FPGA is serviced by the CL1t class and has the following functions:

load(char *fileName);
    This function loads the Alera program into the Level one trigger FPGA.

getFIFO(int count,long *buffer);
    This is an overloaded C++ function that reads count number of 32-bit words from
    the Level One trigger FIFO and stores them in the memory buffer pointed to by buffer. It
    returns a zero if it executed OK, or 1 if the FIFO has insufficient data.

getFIFO(int count,FILE *fp);
    This is an overloaded C++ function that reads count number of 32-bit words from
    the Level One trigger FIFO and stores them in the file pointed to by the FILE pointer fp.
    The file must be opened for writing before calling this function. It returns a zero if it
    executed OK, or 1 if the FIFO has insufficient data.

getFIFO(int count,char *fileName);
    This function opens the file fileName and calls getFIFO(int count,FILE *fp);

getFIFO(FILE *fp);
    This is an overloaded C++ function that reads ALL of the 32-bit words from the
    Level One trigger FIFO and stores them in the file pointed to by the FILE pointer fp. The
    file must be opened for writing before calling this function.

getFIFO(char *filename);
    This function opens the file fileName and calls getFIFO(FILE *fp);

reset(int level);
    Resets the level one trigger registers and if level equals zero, it resets the FIFO's
    and the L1T FPGA.
stat(void);

Decodes the status register for the Level one trigger and displays it on the default console.
The Tracker FPGA is serviced by the CTkr class and has the following functions:

load(char *fileName);
   This function loads the Alera program into the tracker FPGA.

getFIFO(int count, long *buffer);
   This is an overloaded C++ function that reads count number of 32-bit words from
   the Tracker FIFO and stores them in the memory buffer pointed to by buffer. It returns a
   zero if it executed OK, or 1 if the FIFO has insufficient data.

getFIFO(int count, FILE *fp);
   This is an overloaded C++ function that reads count number of 32-bit words from
   the Tracker FIFO and stores them in the file pointed to by the FILE pointer fp. The file
   must be opened for writing before calling this function. It returns a zero if it executed
   OK, or 1 if the FIFO has insufficient data.

getFIFO(int count, char *fileName);
   This function opens the file fileName and calls getFIFO(int count, FILE *fp);

getFIFO(FILE *fp);
   This is an overloaded C++ function that reads ALL of the 32-bit words from the
   Tracker and stores them in the file pointed to by the FILE pointer fp. The file must be
   opened for writing before calling this function.

getFIFO(char *filename);
   This function opens the file fileName and calls getFIFO(FILE *fp);

reset(int level);
   Resets the tracker and if level equals zero, it resets the FIFO's and the L1T and
   trackerFPGA.

stat(void);
   Decodes the status register for the tracker and displays it on the default console.
Class: CServe  This class services the interrupts on the target.

serverStart(void);
    Server Start initializes the semiphores, and VME Interruptor; spawns the service task; and enables the interrupts.

serveInt(void);
    This is the function called when the VME interrupt fires. It sets the intSource flag to let the service task know it was a VME interrupt, and sets the semiphore.

cserviceTask(void);
    The service task waits for the semId semiphore then assembles the telemetery packet and ships the packet out via the ethernet to the host machine.