	Document # LAT-TD-00000-00	Date Effective May 9, 2003
	Prepared by Curt Brune	Supercedes None
GLAST LAT Electronics System	Subsystem/Office Electronics System	
Document Title GEM Trigger Signal Testing – Pattern Description Language and Tools		

GEM Trigger Signal Testing

Pattern Description Language and Tools

GEM Trigger Signal Testing: Pattern Description Language and Tools

by Curt Brune

Published May 9, 2003

This document proposes a hardware and software system for sending trigger signal patterns into the GEM for testing. The name of the system is PGEM, which is a combination of "Pattern Generator" and GEM.

This document is also available in the following formats:

- PDF
- One page html
- Text

Revision History

Version	Date	Comment
0.1	May 9, 2003	Initial draft.

Table of Contents

Introduction

PGEM is a system of hardware and software for providing trigger input signals to the GLT Electronics Module (GEM) [huffer1]. PGEM will produce trigger input signals corresponding to four Tower Electronics Modules (TEMs) [huffer2] and two ACD FREE boards [acd_icd]. Four VME COMM boards [sapozhnikov1] will be used to produce the trigger input signals.

1. Trigger Input Signals

Each TEM produces four trigger input signals:

- Tracker 3-in-a-row (TKR)
- Calorimeter Low Energy (CAL_{LE})
- Calorimeter High Energy (CAL_{HE})
- Busy Signal

Each FREE produces 19 trigger input signals:

- CNO Signal
- 18 Veto Signals ($VETO_0 \ldots VETO_{17}$)

This is a total of 54 signals.

Each signal can be specified as asserted (on) or de-asserted (off) for varying amounts of time in units of sysclks (50 ns for a 20 MHz clock).

Chapter 1. Theory of Operation

This chapter describes how the user would interact with the system.

1.1. Specify the Signal Patterns

Using a text editor the user creates a text file that specifies the time sequencing for all the input trigger signals. The format of the text file is called the "Pattern Description Language" (PDL) and is described in Chapter 2.

1.2. Pattern Playback

Using the tools described in Chapter 3 the user first converts the text file into a binary file, suitable for playback.

On the target system the user initiates playback of the binary file, specifying whether or not they want the patterns to loop forever.

It may also be worthwhile to provide a tool that converts the PDL file into a graphic bitmap to help the user visualize the patterns.

Chapter 2. Pattern Description Language

This chapter describes how a user will specify the trigger signal patterns.

2.1. Named Patterns

Patterns have names, much like variables in computer languages. This allows building complex patterns from simpler base patterns. The syntax of a named pattern definition follows:

```
META_PATTERN:
  PATTERN_1(n1)   - PATTERN_1 repeated n1 times
  PATTERN_2(n2)   - PATTERN_2 repeated n2 times
  PATTERN_3(n3)   - PATTERN_3 repeated n3 times
  PATTERN_4(n4)   - PATTERN_4 repeated n4 times
  ...
```

Here `META_PATTERN` is the pattern name being defined, `PATTERN_1` ... `PATTERN_4` are previously defined pattern names and `n1` ... `n4` are repeat counts (integers[1]), specifying how many times to repeat the pattern.

Two pre-defined pattern names exist, with the following definitions:

- **1** – Assert for 1 clock cycle
- **0** – De-assert for 1 clock cycle

All named patterns are built from these two fundamental patterns (obviously the "0" and "1" pattern names would be reserved language keywords).

2.1.1. Example: A Clock-like Pattern

For example to create a clock-like pattern named CLOCK_4 that has a frequency of sys_frequency / 4 with a duty cycle of 50% and lasts for 100 sys_clock cycles, one could write:

```
DUTY_50:        - Start definition of pattern DUTY_50
  0(2)          - Signal de-asserted (off) for two clock cycles
  1(2)          - Signal    asserted  (on) for two clock cycles

CLOCK_4:        - Start definition of pattern CLOCK_4
  DUTY_50(25)   - Repeat pattern DUTY_50 25 times
```

The `DUTY_50` pattern is four sys_clocks long, de-asserted for two clocks and asserted for two clocks. The `CLOCK_4` pattern is 25 copies of the `DUTY_50` pattern, for a total of 100 sys_clock cycles.

1. It may also be useful for the repeat count to take a wild card value, *, meaning repeat pattern until FIFO full. It remains to be seen if that will be a useful concept.

Note: If you were to create a pattern of alternating 1's and 0's this clock-like signal would have a frequency equal to one half of the FIFO clocking frequency, sys_frequency. The normal sys_frequency for the VME LAT COMM I/O Board is 20 MHz, but an external clock source could also be used.

Using named patterns the user can create a palette of pattern fragments.

2.2. Reserved System Pattern Names

Reserved system pattern names exist for the 54 trigger input signals. These pattern names can only be defined (assigned to). They cannot be used to form other patterns – use regular named patterns for that purpose. The reserved system pattern names are:

```
TEM[i].TKR
TEM[i].CAL_LE
TEM[i].CAL_HE
TEM[i].BUSY

where i = 0, 1, 2, 3

FREE[j].CNO
FREE[j].VETO[k]

where j = 0, 1
      k = 0..17
```

The values assigned to these patterns will be used to program the VME LAT COMM I/O Board FIFOs for playback.

2.2.1. Example: Programming a System Pattern Name

For example, to program the $\texttt{VETO}_{17}$ signal of FREE board 1 with 100 copies of the CLOCK_4 pattern defined in Section 2.1.1 one could write:

```
FREE[1].VETO[17]:   - Assign system pattern
  CLOCK_4(100)      - Repeat pattern CLOCK_4 100 times
```

Chapter 3. Software Tools

This chapter describes the software tools used to create, load and view PDL files.

3.1. pgem_bin

pgem_bin is a program, run on the host system, that converts a PDL file to a binary representation of the signal description. In addition to converting the PDL file pgem_bin also

- Checks for valid syntax.
- Checks for FIFO depth over runs.

The output of pgem_bin is a binary file containing all the trigger signal patterns in a machine readable form.

3.2. pgem_play

pgem_play is shared library loaded into VxWorks, which provides a single callable function:

```
int  PGEM_play (   const char        *file_name,
                   unsigned int         rep_cnt );
```

file_name is a pointer to a string containing the path name of the binary file produced previously by pgem_bin. It is assumed that the target and host system will share a common NFS volume or that some means exists to put the output of pgem_bin in a file system where the target system can see it.

rep_cnt is the repeat count (loop count) indicating how many times to loop through the FIFO's contents. The special value of 0xFFFFFFFF means loop forever.

3.3. pgem_bmp

pgem_bmp is a program, run on the host system, that converts the binary output of pgem_bin to a graphic image file (PNG for example). The image (or maybe the sequence of images) would graphically depict the time evolution of the 54 signals. The idea is to provide a visual picture to assist the pattern author.

Chapter 4. Hardware

This chapter describes the hardware of the system and potential challenges. Here I am only discussing the VME LAT COMM I/O Board, not the cabling, nor the break out box.

4.1. COMM Board Consideration

PGEM proposes to use a group of four COMM boards in a manner theoretically possible, but not yet tested. As such a little R&D is required and potential problems may arise.

4.1.1. Chaining COMM Boards

Since we have more signals than a single VME LAT COMM I/O Board can provide we must daisy-chain 4 COMM boards together and synchronize their playback. While documented as possible this has never been tested.

The first step will be to verify that a master COMM board can initiate playback in the slave COMM boards. This should be easy to verify.

Next the COMM boards must be configured to remove any clock skew introduced by the chaining. The skew could be removed in one of two ways: 1) Each COMM board has a playback delay register, 2) The FIFOs of each COMM board could be padded with a suitable number of zero words.

4.1.2. FIFO Depth

The FIFOs of the VME LAT COMM I/O Board are 32K entries deep, which limits the length of the patterns. The COMM board has a couple of features to work around this limitation however.

The COMM board can be configured to fire an interrupt when playback is complete. Also once the FIFO is programmed it can be played back over and over without reloading the FIFO. Remember loading the FIFO is somewhat costly, using programmed I/O to write one word at a time to the FIFO. So we could program the FIFO once, start playback once and re-start playback in an ISR.

Obviously the interrupt handling introduces some delay before restarting playback. Also the last play back word "sticks" on the COMM outputs and would remain until playback restarted. Therefore it seems wise to insure that the last word written to the FIFO is a zero, leaving all the outputs de-asserted until playback restarts.

As a wild guess say it takes 40 microseconds (800 clocks @ 20 MHz) to restart playback once the interrupt is raised. Then the time to play back a full FIFO plus restart would be 32767 + 800 clocks, about 33500 cycles. Of the total cycles for one loop the 800 clocks represents about 2.5% of the total time. When running continuously all signals would be de-asserted for 2.5% of the time for the same 800 cycle interval. If the 40 microsecond guess is too high then the percentage will be smaller.

This does not seem like a big deal to me, but something to be aware of.

References

[huffer1] Michael Huffer, *The GLT Electronics Module (GEM): Programming ICD Specification*, LAT-TD-01545.

[huffer2] Michael Huffer, *The Tower Electronics Module (TEM): A Primer*, LAT-DS-00605-D1.

[acd_icd] ACD GSFC, *ACD-AEM Interface Control Document*, LAT-SS-00363-D9.

[sapozhnikov1] Leonid Sapozhnikov, *LAT VME Front-End Communications (Com) Board*, LAT-DS-00226-D1.