

# SCT/Pixel DAQ Workshop

## ROD DSP Software

### Overview of the DSP Software Architecture

- The List Manager
  - 1) From VME Host
  - 2) From DSP
- Text Buffer Handling
- The Task Manager
- The Event Manager (slaves)

### Overview of DSP Primitives & Tasks

- Common to Both DSP Types
- Slave Primitives & Tasks
- Master Primitives & Tasks

### Detailed Discussions

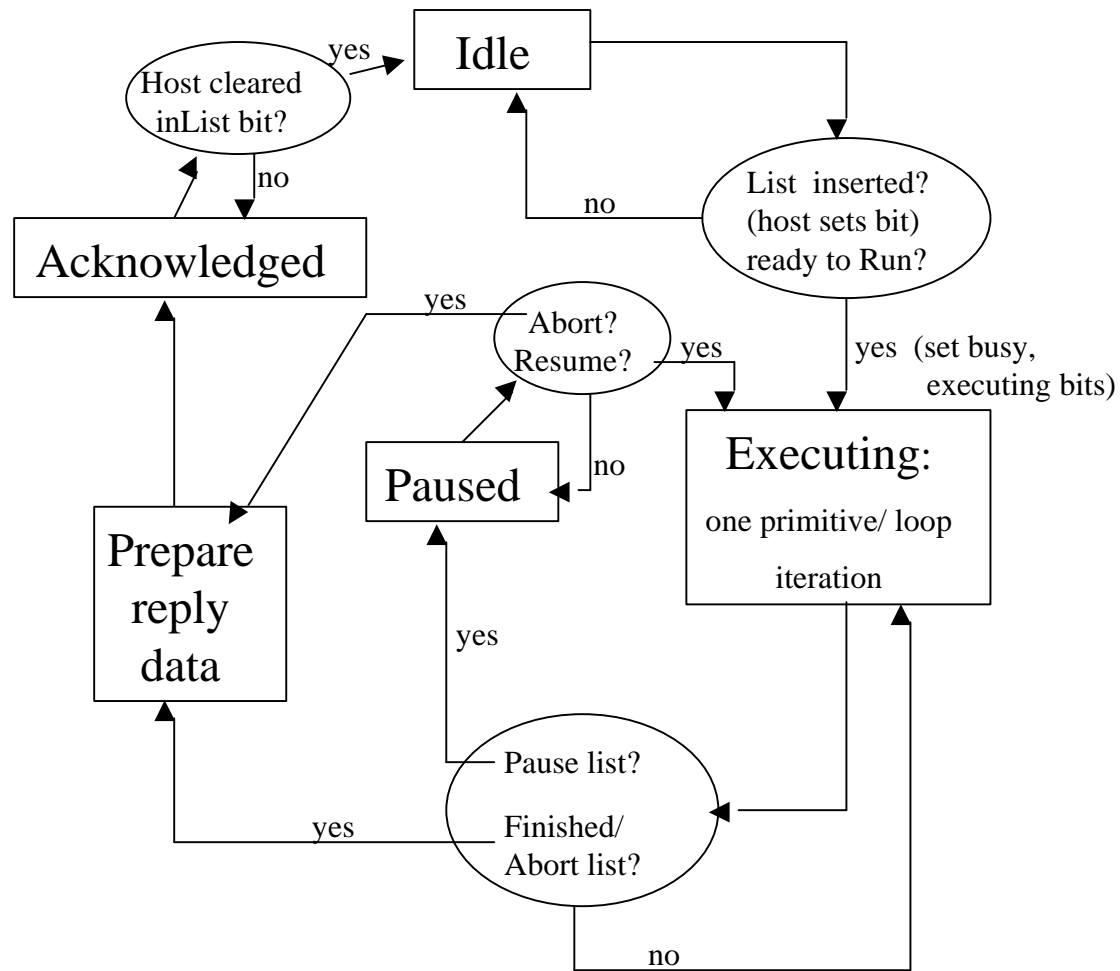
- Module Configuration
- Event trap Setup
- Histogramming

Douglas Ferguson  
UW-Madison & LBL



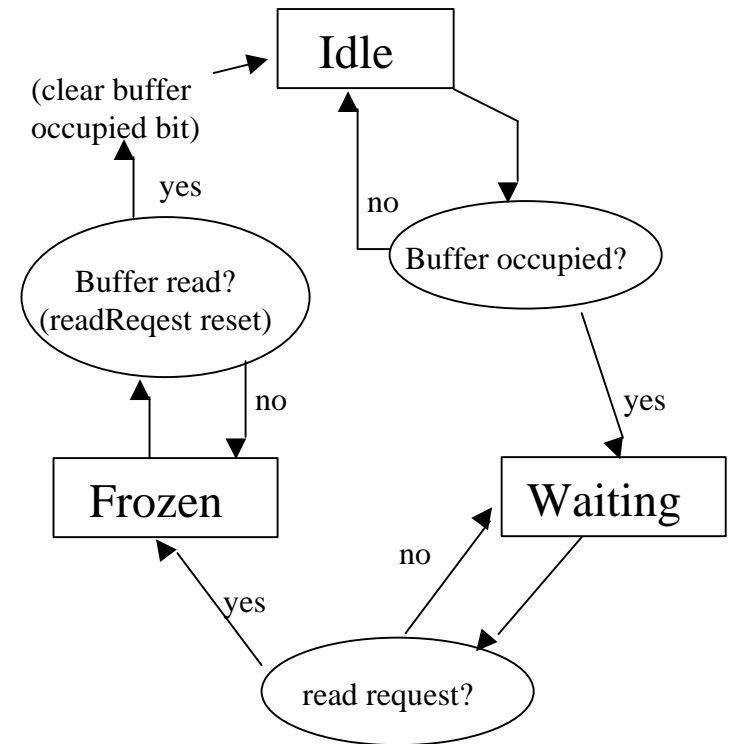
# State Machines

## Primitive List Execution



Two types: Host List & Inter-DSP List

## Text Buffer Sending



## Primitives vs. Tasks

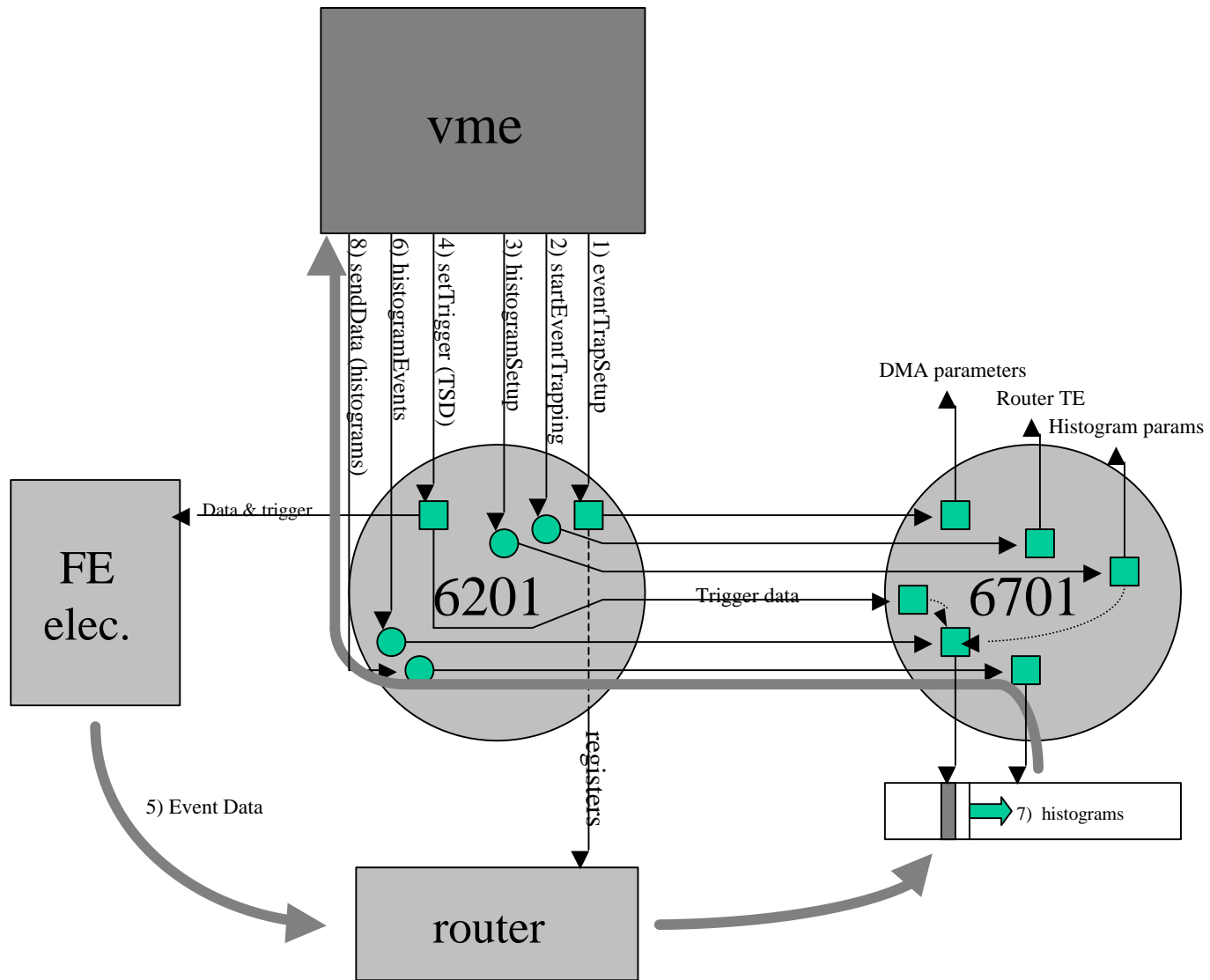
### Primitives:

- o Generally are executed once. Primitives can return a value that requests that the primitive be repeated (say, for polling), but only one primitive executes at a time.
- o Return data via the reply list. Each primitive in the input list contributes a header block to the reply, and optionally adds data, which can either be structured or unstructured.
- o Generally are simple. The list is a “program”, and primitives are its functions.

### Tasks:

- o Execute continuously, and together. Several tasks will happily co-exist on a DSP. Tasks can be paused & aborted, just like primitive lists. Tasks tend to be independent of each other, though there is no need for this.
- o Return data asynchronously. When a task completes, it can send an info message to the host. Any output data can be transferred via a primitive. sendData will handle large chunks of data processed by tasks & primitives (events, histograms, fits etc.); taskOperation:stop or query return the task’s internal variables. Some tasks (histogramming) also report via registers.
- o Typically are more complex. Tasks can have several different internal states; the task manager will report the states of any running or completed/halted tasks in the running task register (idata).

# Phase 2



# Event Management

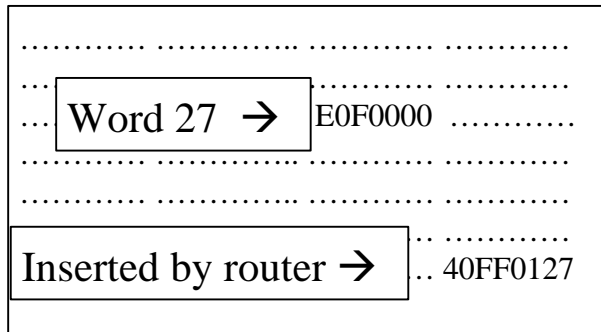
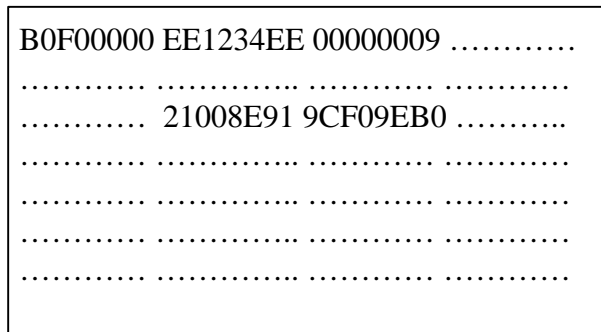
Events in memory:

0x80008000 – 0x80010000 “burst buffer”,

0x02092000 – 0x020d2000 “external burst buffers”

} 8 frame sets, + 1 reserve (in SDRAM).  
 Frame set 0 → IDRAM

## Event Based Data:



1 frame= 0x100 words

Event based data allows the DMA ISR to complete its work quickly, since the event’s length is always found at the end of a frame. Marker 0x40ff should never occur in data. The expense is that small events take up more space.

ISR fills in a set of “frame registers”, and an auxiliary set of “frame summary registers.”

→ These are located in slave IPRAM at an (~) fixed location, and can be monitored to keep track of the DSP’s activity.

Event queuing sequence:

- Header is detected: new event, begin filling in its data.
- Any frames until trailer is found are marked in the event’s frame mask.
- Trailer is detected: event goes onto event queue.

## Event Management (cont)

In the “idata” memory section (0x80000040- 0840, for important program variables):

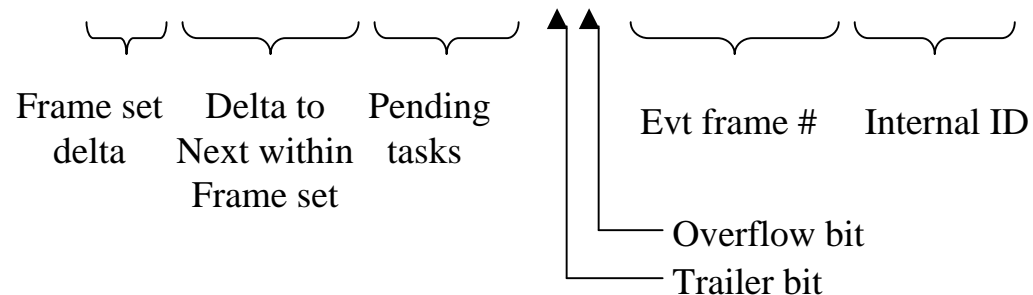
Event Manager Control structure (small set of control & ISR variables).

Frame summary registers:

- One per frame set, used by the DMA ISR to find where to put the next incoming data frame. Show at a glance the DSP’s burst buffer usage (one bit per frame).

Frame set registers:

- 32 per frame set, used by the event manager to jump from one frame to the next when processing multi-frame events, and determine which tasks to give them to.
- defined as 0xSSSD DDDD PPPP PxTO FFFF FFFF IIII IIII



Overflows: if the IDRAM burst buffer fills up (with a giant event, or many events arriving more quickly than they can be processed), a buffer overflow occurs.

All IPRAM frames are copied into SDRAM, and re-organization must be done on the frame set registers, and event frame masks. → **EXPENSIVE.**

## Common Primitives:

Set Memory	Set Err Msg Mask	Pause List	Event Trap Setup	Set LED
Copy Memory	Echo	Memory Test	<u>Send Data</u>	Flash LED
Start Task	<u>Task Operation</u>	<u>Module Mask</u>	<u>Set Trigger</u>	Write Buffer

## Slave Primitives & Tasks:

Start Event Trapping      Stop Event Trapping      Histogram Setup

---

Tasks:

Histogramming	<u>Event Trapping</u>	Occupancy Counting
Error Counting	Link Re-synchronization	

---

## Master Primitives & Tasks:

<u>Transmit Serial Data</u>	R/W Register Field	Start Slave Executing	Start Slave List
R/W Module Data	Poll Register Field	Configure Slave	Send Slave List
Send Configuration	R/W FIFO	R/W Slave Memory	Slave List Operation
<u>Build Stream</u>	<u>Send Stream</u>		

---

Tasks:

Histogram Control	<u>Memory Mirroring</u>	Trap Request Monitoring
-------------------	-------------------------	-------------------------

! = use with caution

## Primitive Summaries:

**Echo:** a primitive “ping”. Hands back as a reply whatever you give it. Useful for testing DSP response time, DSP buffers.

**Set Err Msg Mask:** sets the default mask for printing messages in the error buffer.

**Pause List:** Pauses a primitive list when encountered during its execution. The list can be resumed by setting the Resume List bit in the command register.

**Set Memory, Copy Memory:** Utility primitives for organizing DSP memory. They operate on DSP memory and do not return any data. (!)

**Memory Test:** use some basic memory tests to check memory integrity. (!)

**Set Led, Flash Led:** happy blinking little lights. People viewing the ROD crates could use these to tell what they’re doing?

**Write Buffer:** Writes the input data in the selected buffer.

## Primitive Summaries (cont):

**Start Task:** Starts a task, using the input data. The task must not be currently running on the DSP. The order in which tasks run is determined by their priority (input).

**Task Operation:** Perform an operation on a currently running task. Currently these are stop, pause, resume, query & reset. Querying a task returns its current data (internal variables). Stopping & resetting it also return data. **Data return not fully implemented.**

**Send Data:** Returns a block of data which has been processed by a primitive or task. The data is well defined, but with a variable length. If the data is large (histograms), returns a pointer to it; **the host will then ask the master DSP to shut down communications to the slave DSPs so that it can retrieve it** (currently this is manual). Data types include: **master DSP- mirror, router & stream data; slave DSPs- event, histogram, occupancy & fit data.**

**Module Mask:** Perform all the steps necessary to set the front-end module masks. Would it be useful for slaves?

## Primitive Summaries (cont):

**Set Trigger:** Being re-vamped. Primitive version of the histogram control task. Differences between them are that set trigger only does a single bin, and can be set to send triggers at an input rate. Also can be run locally on slave DSPs.

---

**Start & Stop Event Trapping:** Once Event Trap Setup has been run, these can be run to load & unload the DMA engine, & enable or disable the router's Trap Enable flag (controlled by slave DSPs).

**Histogram Setup:** Sets up histogramming variables, and initializes the histograms. The inputs are the histogram memory base (default= 0x020D2000), the ROD type, the # of bins, **the counter set**, any padding between them (I use this to view the histograms more easily using a hex editor). For fitting purposes **x0 & delta-x** between histogram bins have also been included.

---

**Read & Write Register Field:** Used to set any of the of the ROD's FPGA RRIF (Rod Register InterFace) registers. The ROD Operations Manual explains these.

**Poll Register Field:** Some RRIF registers are self-clearing. This primitive monitors them to see when they have done so.

## Primitive Summaries (cont):

**Read & Write FIFO:** Places data in any of the ROD's on-board FIFOs; this is used in conjunction with R&W Reg. Field to trigger the FIFOs to play their data into the desired ROD FPGA (formatters, EFB, or Router).

**Start Slave Executing:** After loading the slave's IPRAM, IDRAM and external memories, SSE sets them running. Once running, board must be reset to stop them! (Standard list resets board). Note that the input parameters are unimportant now.

**Configure Slave:** Now obsolete.

**R/W Slave Memory:** Reads from or writes to specific locations in the slave DSPs' memory. The data length must be  $< \sim 150\text{KB}$ , the size of the primitive list buffers. (!)

**Send/Start Slave List:** When slave DSPs execute host lists, the master DSP serves as the intermediary, doing all the actual list coordination using these primitives

**Transmit Serial Data:** Transmit the input data out of serial port 0, 1 or both, and optionally capture the data in the ROD input memories. To work properly, the RRIF registers must be set correctly. Example lists are included in the RODTS setup. (!)

## Primitive Summaries (cont):

**Build Stream:** Build up a command stream from input command/data lists, which can contain up to 4 commands at a time; also used to reset a stream buffer.

Command lists contain up to 4 commands, and successive calls will concatenate them.

Either of 2 stream buffers can be used, or both if data is to be transmitted from both serial ports. Commands can be any of the delay, fast, slow, data or mask commands.

The stream buffers are located at: 0: 0x80008000, 1:0x8000B400.

**Send Stream:** Sends any data contained in the stream buffer(s) out of the requested serial port(s), with optional capture. **Needs testing.**

## Task Summaries:

**Event Trapping:** Currently this is a stand-alone slave task which simply traps (copies) the desired # of events into the (input) trapping buffer. (A default buffer in high memory is suggested). The router must be set properly beforehand; soon will be modified to cooperate with the master DSP in setting the router's registers on the fly.

**Occupancy counting:** Creates occupancy maps for incoming events.

**Error counting:** Counts & categorizes events with errors.

**Link Re-synchronization:** Performs whatever corrective actions are necessary to bring links/events back into synchronization when a trigger is dropped.

---

**Memory Mirroring:** Mirror the slave DSP's memory (for example, their communication registers & idata) This task is special in that it can be called multiple times & accumulate up to 6 different slave memory regions to mirror (mirrors are stacked). Needs testing.

**Trap Request Monitoring:** Monitors the slave DSPs trap request registers; when a request is made sets up that DSP's router registers appropriately.

## Module Configuration

### Read & Write Module Configuration:

- Input parameters:

**readNotWrite:** If set, indicates that data is to be read; if not set, write the data into the appropriate module.

**configuration set, module #:** 5 configuration sets of 48 modules each, beginning at memory location 0x02800000 (length ~3MB).

0	·
1	·
·	46
·	47

**ATLAS**

0	·
1	·
·	46
·	47

**CALIB**

0	·
1	·
·	46
·	47

**EXPERimental**

+ **EXPER1,  
EXPER2**

**pointer to ABCDModule data:** Null if read, beginning of data if write.

- Output parameters (for read): **ABCDModule**.
- Module data need not be stored separately in sets of 48. Since the FE command masks allow fanout of the serial streams, a configuration set could contain as many as 48 different sub-sets.

## Module Configuration (cont)

Before sending module configuration, the front-end command masks must be set appropriately (moduleMask or rwRegField). The appropriate sequence is:

- 1) **RRIF Command 1**: set bits 0 (enable command streams) & 20 (mask load enable). If bit 18 (trigger signal decoder enable) is set, setting bit 2 will load the mask; if not set, the next L1 trigger will.
- 2) **FE Command Mask Registers, LO & HI**: set appropriately to mask data streams from serial port 0 & 1, respectively.
- 3) **RRIF Command 1**: Set bit 2; will either load immediately or wait for the next trigger.

### **Send Configuration:**

- Input parameters:

**cmdBuff**: The command stream buffer to use: 0,1 or both.

**captureSerOn**: Flag indicating whether or not (part of) the serial data sent out will be captured in the input memories .

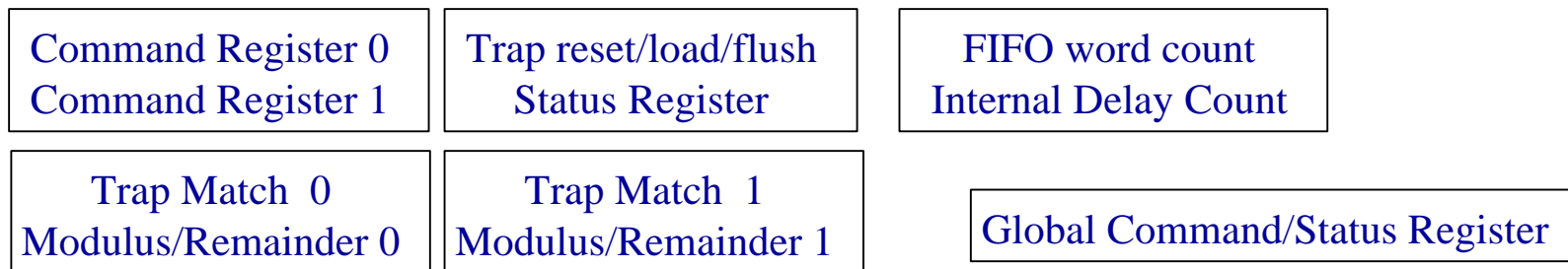
**type**: The type of module data to send out from the set: basic, trim DACs, or all.

**struct ID**: The ID of the configuration set to use.

**module #[2]**: The module #(s) from the config set which will send data. If more than one is specified, cmdBuff must be set to both.

## Event Trap Setup

- Event trap setup is an important primitive central to the event trapping; it is used to coordinate the actions of the slave DSPs and Router.
- Event trap setup should be sent from the host to the master DSP; the master will then set the Router's registers, and pass a copy of itself onto the slave using the inter-DSP primitive list.
- The router has a set of registers for each slave DSP, and a global register set. Each DSP register set has two (mostly) independent traps. Trap 0 contains some Extra flags which are common to both traps:



- The Trap reset/load/flush & internal delay count are set by the DSP and generally should not be touched. The status & FIFO word count are read-only.

## Event Trap Setup (cont)

### • Input Parameters:

**slvBits:** A bitfield indicating which slaves we are setting up. Bit 0 → slave 0, etc.

**# of events:** The total # of events to trap. Currently should be set to `COLLECT_FOREVER`.

**timeout:** A timeout for the reply, when the master DSP sends the primitive to a slave.

**extRouterSetup:** Flag indicating that the router is set up independently; if set the master DSP will not write to the router's registers. Use with care!

**distribute:** If set, the master DSP will distribute check that the input modulus (described later) is correct, and will distribute remainders among the input slaves.

**permitBackPressure:** Bit in the global CSR. Should only be applied in calibration mode (!!)

Allows the router to apply back pressure to the EFB if large events are coming through and the DSP needs time to move data into SDRAM. (The slave DSP sets the interrupt mask from DSP inside the SR, if so).

**format:** Bit inside CR0 which controls the format in which events are sent to the DSP. This is common to trap 0 & 1. High= error format, Low= normal or "S-Link" format.

## Event Trap Setup (cont)

### • Input Parameters (cont):

**dataMode:** Bit inside CR0 which controls the whether the Router (for that DSP) outputs data in event mode (skipping to the end of a frame with a trailer, and writing 0x40ff +length, or in data mode, in which it waits until it has enough data to fill a frame. Common to both traps.

**sLink:** Bit in SR0; if set the trap match is overridden and only the modulus & remainder of trap 0 are used to distribute events among DSPs. Common to both traps. **Probably should use for now. The trap 1 VHDL has not been tested as thoroughly; likewise trapping specific event types needs more testing.**

### • Trap Input Parameters:

**config:** The type of events to trap. If IDLE, trap is unused; otherwise ATLAS, ROD or TIM. If sLink is set, Trap 1 should be IDLE, and trap 0 set to SLINK.

**exclusionFlag:** If set will trap all events \*except\* those in the match register. **NOTE: for now this is common to both traps.**

**match:** The type of event for the router to match.

## Event Trap Setup (cont)

### • Trap Input Parameters (cont):

**function:** Determines what tasks will process the events coming from the trap. Functions are additive: **HISTO**= 1, **TRAP**= 2, **OCCUPANCY**= 4, etc. Used only by the slave DSPs.

**modulus/remainder:** Determines which matched events are trapped. Events are trapped if  $\text{mod}(\text{trap match counter}, \text{modulus}) = \text{remainder}$ .

### • DSP Input Parameters:

**releaseFrames:** If set, the last task to operate on an event will release its frames as it processes them, freeing up space. **This is experimental and should be left OFF for now.**

**trapStray/iterLimit:** Stray events are events which shouldn't be there, but were trapped anyway, either through an electronics glitch or operator error. They gum up the works if not disposed of. If **trapStray** is set, they are put into a special "observation buffer" (after the 7<sup>th</sup> SDRAM frame buffer). **iterLimit** determines how long they should be kept around before being removed/sent for observation. Suggestion: trap strays, and set **iterLimit** to a low number, say 2 or so.

# Histogramming

First a dumb quote: “Just a shell...”

## Histogram Control Parameters:

**cmdList[2]:** Two 4 fxn command lists; list 0 if for configuration, list 1 is a trigger.

**slvBits:** similar to ETS slvBits, except for the histogram control task it indicates which slaves have had the histogramming task set up on them.

**cmdBuff:** Which command buffer to use. If both, will use buffer [0] for the config and buffer [1] for the trigger sequence.

**preBuilt:** Flag indicating that pre-built, custom sequences are in the command buffers. If so, the cmdLists will be ignored, and cmdBuff must be “both”.

**preBuiltDataOffset:** If pre-built buffers are being used, offsets from the beginning of each buffer to the word containing the data which will be incremented.

USEFUL?

**chipAddress:** the chip address for slow & data commands. Normally should be “all chips”?

## Histogramming (cont)

### Histogram Control Parameters (cont):

**repetitions, nPoints:** The # of triggers per bin, and the # of bins to do.

**bin0, set:** The starting bin, and the set counter to use.

**incCmd[2]:** The position of the command on the list which is incremented between bins. (non-pre-built commands only).

**incData[2]:** The amount to increment the command's data by. If no increment is wanted, choose a command without data or use 0. (pre-built & non-pre-built).

### Histogram Parameters:

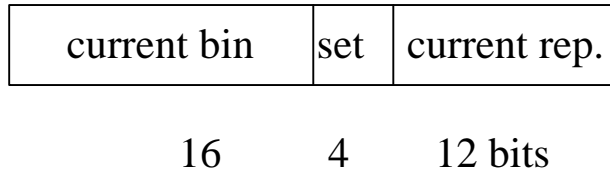
**nEvents:** The total # of events to histogram; can be set to COLLECT\_FOREVER (like ETS).

**controlFlag:** A flag indicating who is controlling the histogramming. Three settings are currently implemented: **MASTER\_HREG** means that either set trigger or the histogram control task running on the master DSP will control through the histogram command register; **LOCAL\_INC** means that the bin will increment between each event (useful for time scans), and **LOCAL\_SET\_TRIG** means that a set trigger primitive running on the slave will set the bin.

# Histogramming Registers

## Master DSP

### Histogram CMD (really status)



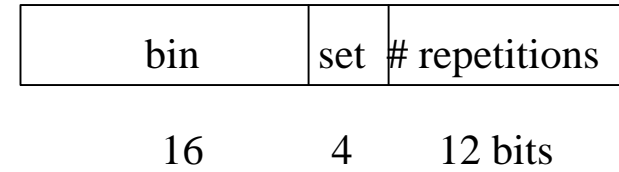
### Histogram STAT 0: events received



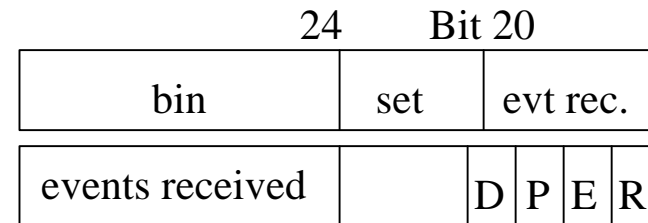
### Histogram STAT 1: Processing time & event length

## Slave DSPs

### Histogram CMD



### Histogram STAT 0



↑  
Bit 8

R: Ready  
E: Expecting  
P: Processing  
D: Done

# Histogramming Task States

## Histogram Control Task

### INIT

- Task variable setup.
- Check HRDY in slaves.

### Slave Polling (all states)

- Copy slave HSTAT 0,1 to local variables. Update own HSTAT 0.

### NEWBIN

- Zero HSTAT 0,1. Place current bin & set in HCMD.
- Zero out current repetitions
- Place current bin set, # rep in all slave HCMD.
- Create & send configuration command stream. Prepare trigger command stream.

### WAITEXP

- Wait for all slave HSTAT EXP HI.

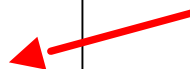
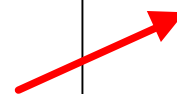
## Histogram Task

### INIT

- Task variable setup.
- HSTAT 0 RDY HI

### READY

- Zero out events received variable.
- Wait for #repetitions in HCMD != 0. Then:
  - Set bin, histogram set, # rep. this bin; copy bin & set to own HSTAT 0.
  - Clear HSTAT 0 events received.
  - HSTAT 0 DONE LO, EXP HI.



# Histogramming Task States (cont)

## Histogram Control Task

### PULSING

- Wait for the # events received by all slaves == current rep per slave. (= curr. Rep./# slaves). Check for timeout. Then:
- Send out one trigger per slave, update current rep., HCMD. # rep
- If Current repetitions < # rep. per bin, repeat.

### wAITING

- Wait for all slave HSTAT 0 DONE HI

### PREP

- Increment # bins done.
- If bins done == # points:  
state → DONE. Else:
  - Increment the data variables of the configuration & trigger lists
  - state → NEWBIN

## Histogram Task

### EXPECTING

- Wait for events to arrive on queue. Then:
  - HSTAT 0 PROC HI.
  - Histogram the event
  - HSTAT 0 PROC LO.
  - Increment events received, HSTAT 0 events received. HSTAT 0 PROC LO.
- If events received < #rep. this bin, repeat. Else:
  - HSTAT 0 EXP LO, DONE HI
  - Clear HCMD #repetitions.
  - state → READY

