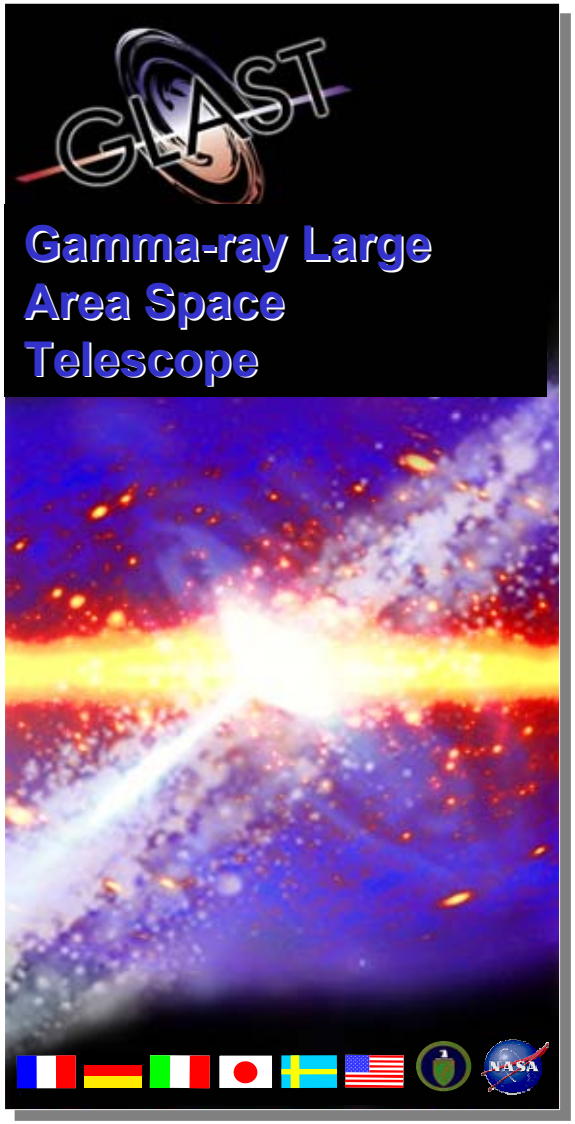


GLAST Large Area Telescope:

Electronics, Data Acquisition & Instrument Flight Software

Flight Software III

A.P.Waite
Stanford Linear Accelerator Center
Engineering Physicist

apw@slac.stanford.edu
(650) 926-2075

Outline

- LAT Startup
- Instrument Configuration
  - Instrument configuration for EM1
  - Software configuration for EM1
- Front End Simulators
- Event Filtering
- Software Development Approach
- Software Safety
- Software Test Approach
- Software Test Executive

LAT Startup (0)

- **The following details the cold startup procedure in a step-by-step fashion. The outline follows the consistent procedure**
  - **Check if conditions are OK to proceed by examining SC and/or LAT telemetry.**
  - **If OK, then by ground command to either the SC or LAT, execute the next step.**
- **The steps that involve strictly SC telecommands or decisions based on SC telemetry are indicated as the bold, underlined numbered steps. The steps that involve strictly LAT telecommands or decisions based on LAT telemetry are indicated as normally numbered steps.**

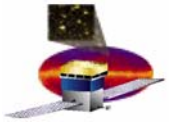
Step	Description	Means/Action
1	Ground checks temperatures and voltages on the SIUs, PDUs and GASUs.	Information is in SC to ground housekeeping telemetry. If OK, proceed to Step 2.
2	Select and power on of the two SIUs	Ground to SC ground command
3	Ground checks temperatures and voltages on the powered SIU	Information is in SC to ground housekeeping telemetry. If OK, proceed to Step 4
4	SIU notifies S/C that it has configured its 1553 bus by raising an output discrete line.	The SIU boot process can now use a primitive 1553 driver to transmit a telemetry containing boot status and a limited set of LAT housekeeping data.
5	Ground may optionally send telecommands to the SIU during the boot sequence.	The SIU boot process has a pause loop allowing the ground to interrupt redirect the boot sequence by sending telecommands This feature is generally (and rarely) used to reconfigure the secondary boot.

LAT Startup (1)

Step	Description	Means/Action
6	LAT obtains the following information a. Will the LAT be powered by the Primary or Redundant main DAQ power feed b. Should PDU Primary be selected c. Should PDU Redundant be selected d. Should GASU Primary be selected e. Should GASU Redundant be selected f. Should LAT communicate with primary or redundant Spacecraft C&DH (SSR interface, 1PPS time hack and Discretes. Note that these selections are controlled as a group by a single bit). g. Should LAT listen to GBM primary or redundant GRB alert interrupt. As a matter of note, the LAT expects the choice will always be PRIMARY until a failure occurs. SC obtains information which Science Data Interface the LAT will be using	Information is stored in the SIUs EEPROM. The selection of the Primary or Redundant PDUs is not mutually exclusive. Either one or both PDUs can be selected (powered). The selection of the Primary or Redundant GASUs is not mutually exclusive. Either one or both GASUs can be selected (powered). The SC can learn which science interface is being used by the LAT either through explicit ground command or by storing this information in nonvolatile memory.
7	The selected LAT main DAQ power feed is switched on	Ground to SC telecommand. Note that no power is drawn on this feed until Step 8.

LAT Startup (2)

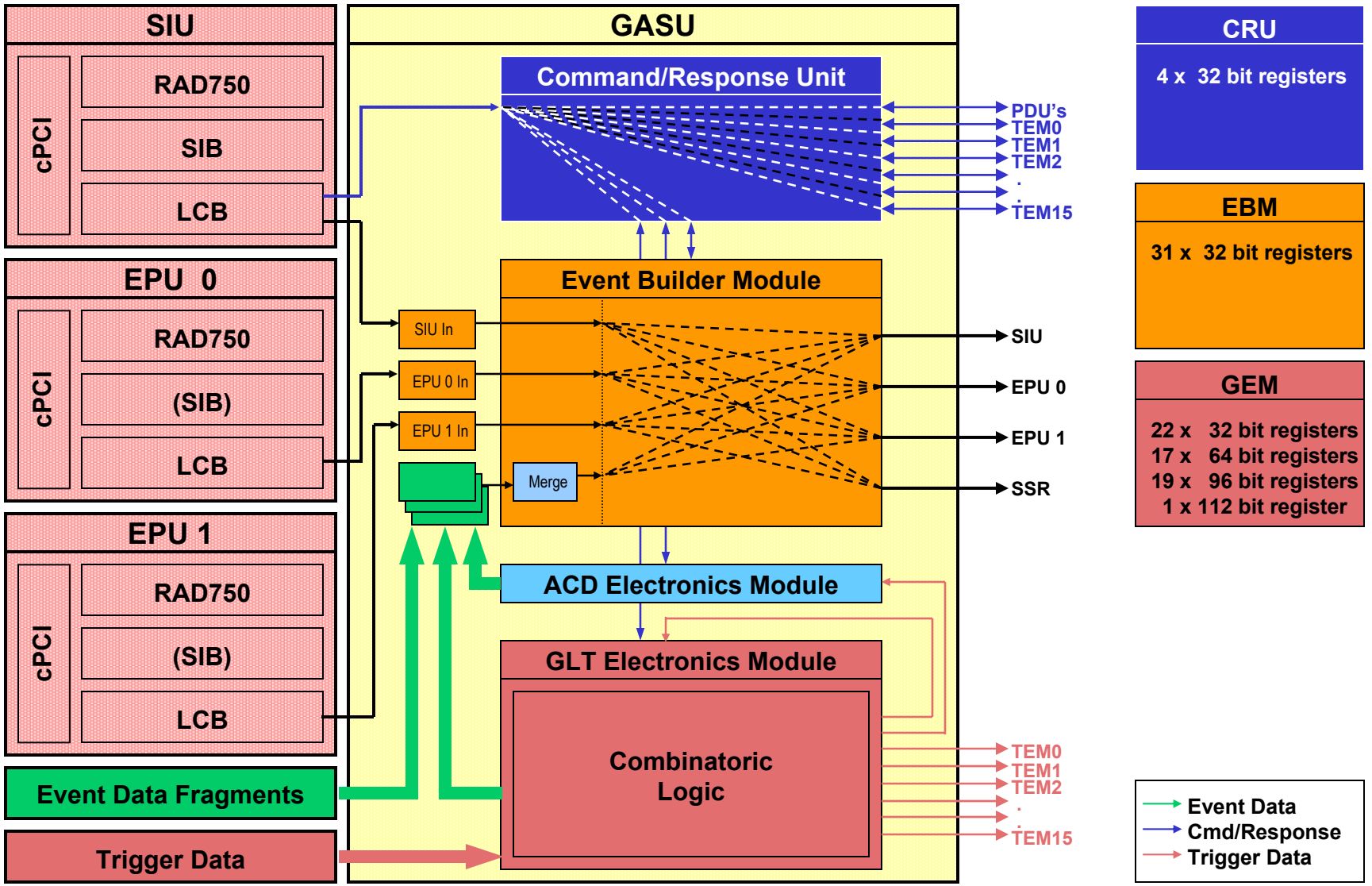
Step	Description	Means/Action
8	LAT closes the input switches of the selected PDU in accordance with 4a, 4b, 4c	Ground command from ground to LAT. The PDUs must have come up such that their input and output switches are all in the open position, ensuring no power is applied to the LAT until instructed.
9	Ground checks PDU voltages and temperatures	Information is in SC housekeeping telemetry. If OK, proceed to step 9
10	SIU powers GASU(s), consistent with 4d and 4e	Ground to LAT telecommand
11	Ground checks GASU voltages and temperatures	Information is in SC housekeeping telemetry. If OK, proceed to step 12
12	Ground instructs the SIU to start the Thermal Control System	Ground to LAT telecommand
13	Ground monitors TEM/AEM housekeeping	Information is in LAT housekeeping telemetry. The LAT housekeeping telemetry was automatically started after the GASU was configured. If OK, proceed to step 14. The housekeeping is still the limited set available when only the PDU and GASU are powered.
14	LAT internal configuration	The remaining startup is a LAT internal procedure and does not involve the SC.



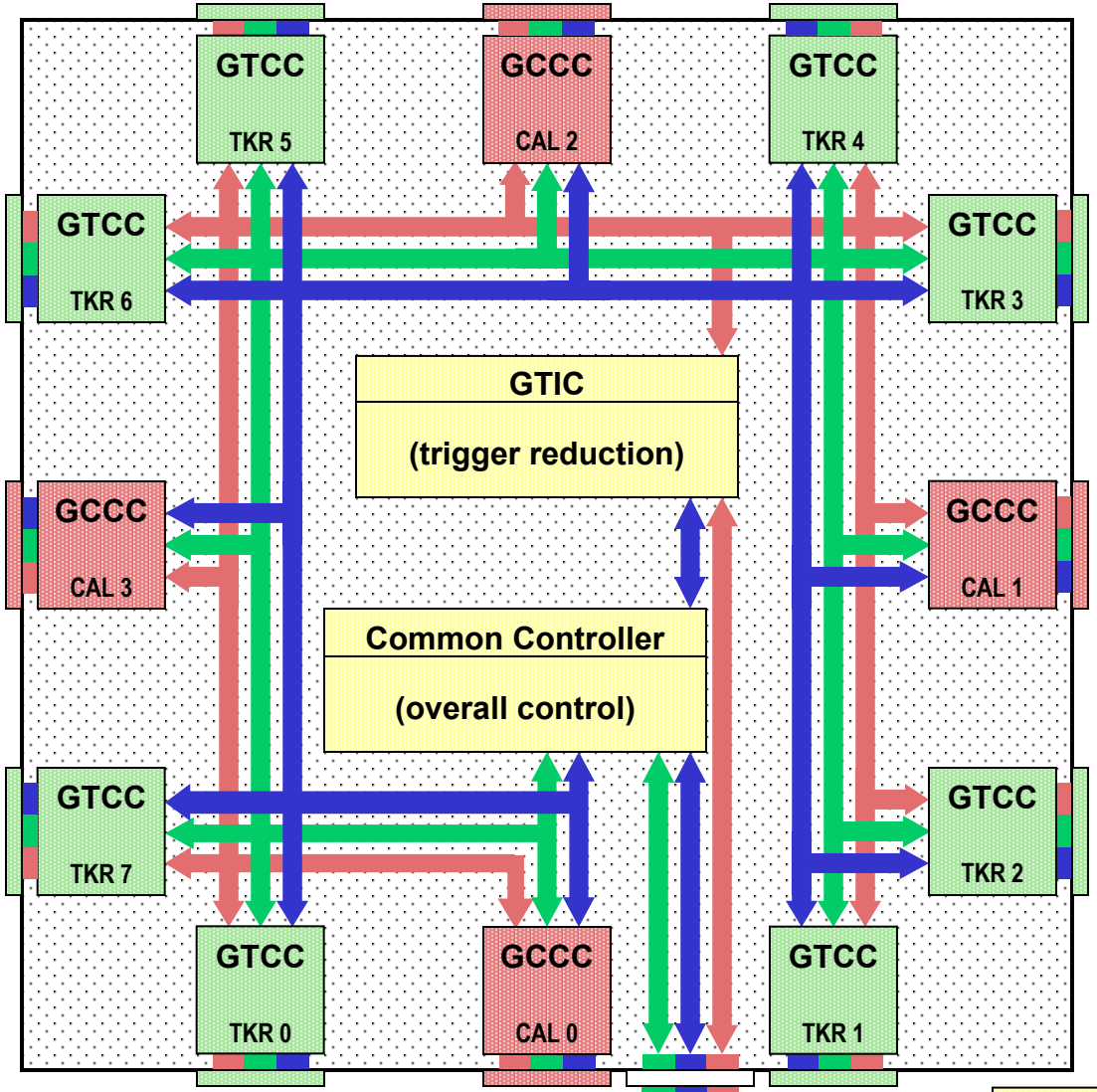
LAT Configuration

- Configuration controlled by setting registers
  - Messages sent from SIU via LCB using command / response fabric
    - Message protocol is LATp
    - Routed through CRU on GASU to destination modules
  - Message data contains routing information for forwarding to final hardware destination
    - CRU
    - GEM
    - EBM
    - TEM(s)
      - Common
      - Common → GTIC
      - Common → GTCC
      - Common → GTCC → GTRC
      - Common → GTCC → GTRC → GTFE
      - Common → GCCC
      - Common → GCCC → GCRC
      - Common → GCCC → GCRC → GCFE
    - AEM
      - GARC
      - GARC → GAFE
    - PDU(s)

GASU


SIU
cPCI
RAD750
SIB
LCB
EPU 0
cPCI
RAD750
(SIB)
LCB
EPU 1
cPCI
RAD750
(SIB)
LCB
Event Data Fragments
Trigger Data
GASU
Command/Response Unit
PDU's
TEM0
TEM1
TEM2
TEM15
Event Builder Module
SIU In
EPU 0 In
EPU 1 In
Merge
SIU
EPU 0
EPU 1
SSR
ACD Electronics Module
GLT Electronics Module
Combinatoric Logic
TEM0
TEM1
TEM2
TEM15
CRU
4 x 32 bit registers
EBM
31 x 32 bit registers
GEM
22 x 32 bit registers
17 x 64 bit registers
19 x 96 bit registers
1 x 112 bit register
Event Data
Cmd/Response
Trigger Data

Tower Electronics Module (TEM)



GTCC
6 x 32 bit registers

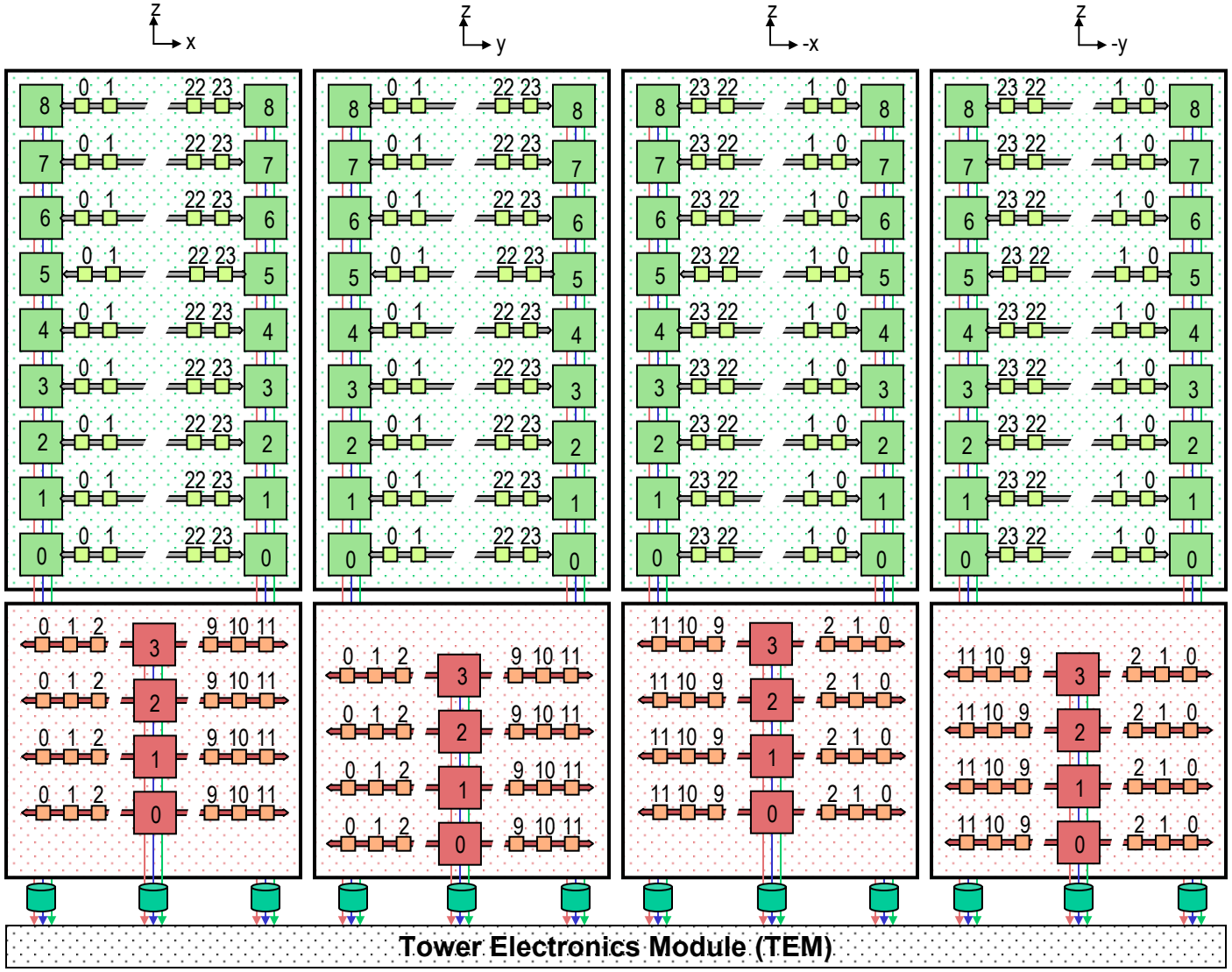
GCCC
7 x 32 bit registers

GTIC
11 x 32 bit registers
2 x 18 bit registers
2 x 16 bit registers
1 x 3 bit register
1 x 112 bit register

Common
7 x 32 bit registers

Event Data
Cmd/Response
Trigger Data

TKR & CAL



GTFE
5 x 64 bit registers

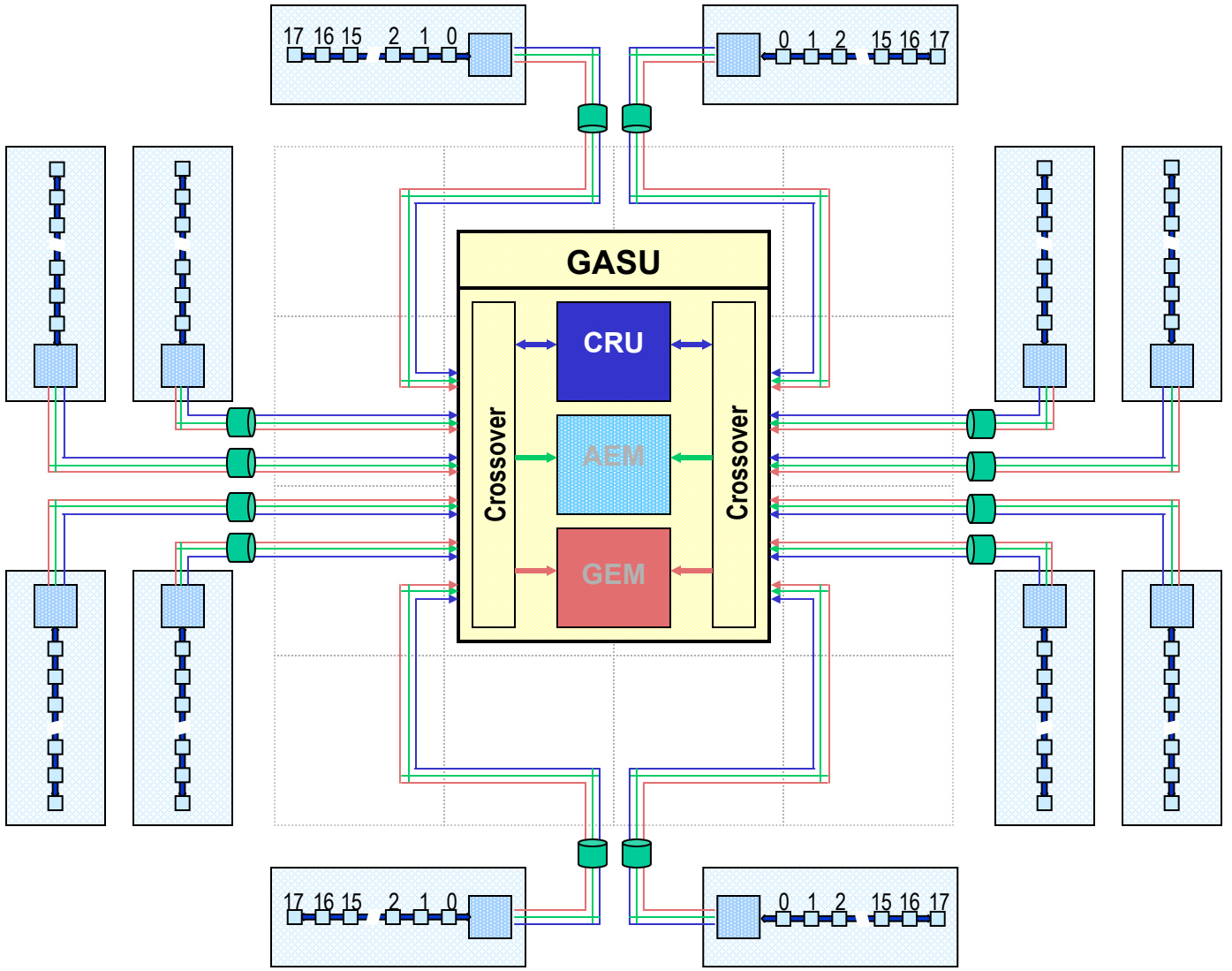
GTRC
2 x 64 bit registers

GCFE
7 x 16 bit registers

GCRC
8 x 16 bit registers

Event Data
Cmd/Response
Trigger Data

ACD



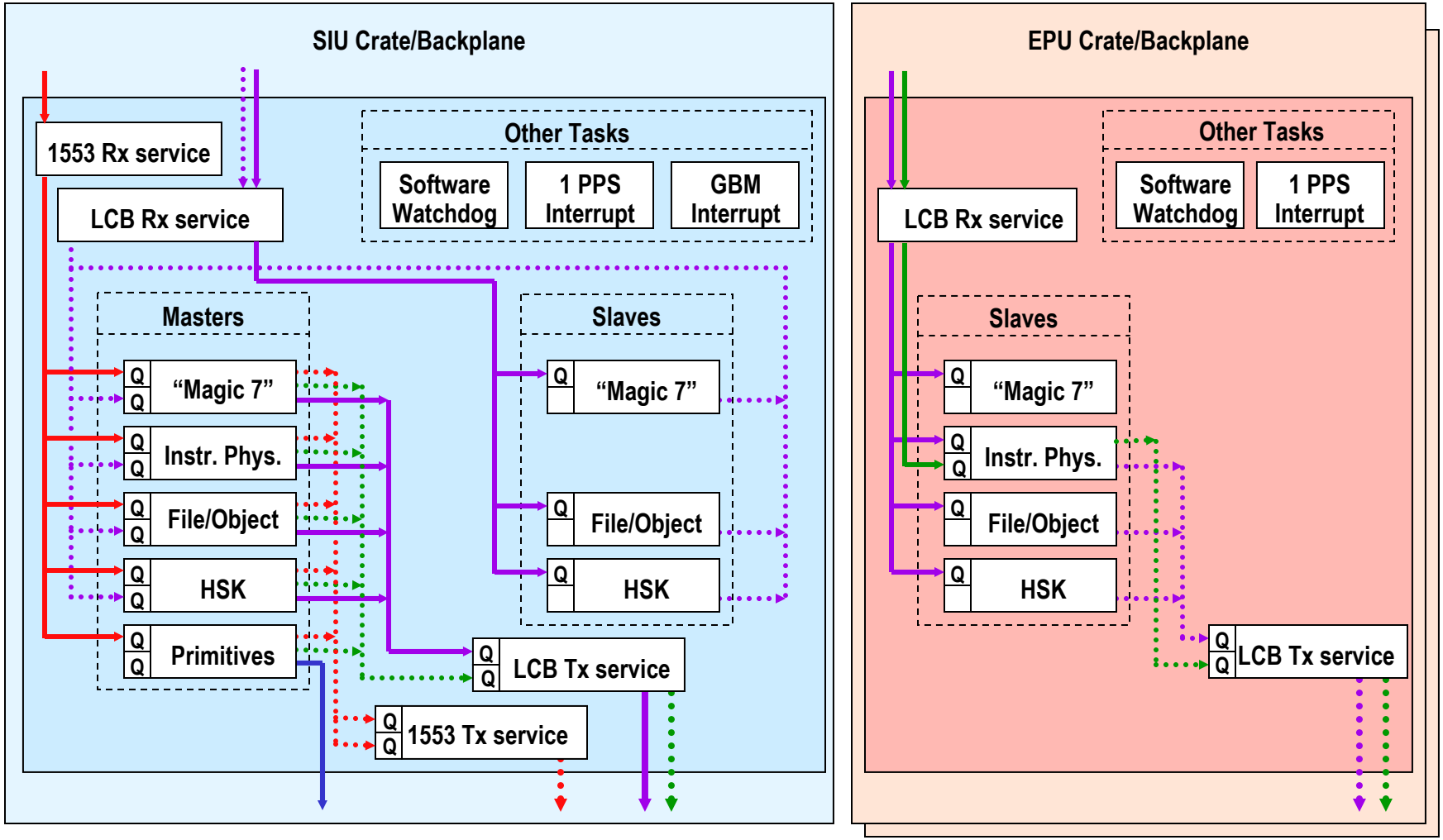
GAFE
11 x 16 bit registers

GARC
44 x 16 bit registers

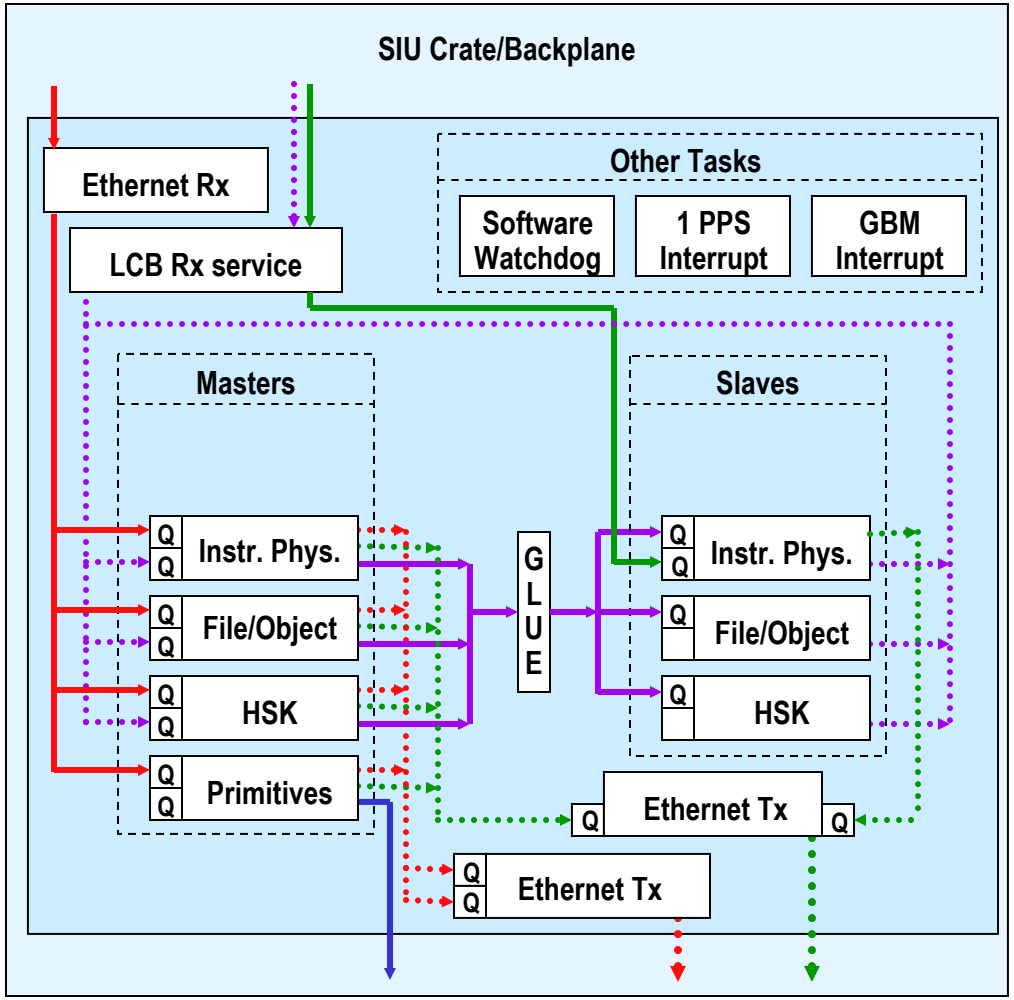
AEM
8 x 32 bit registers
12 x 64 bit registers

Event Data
Cmd/Response
Trigger Data

FSW Architecture with Internal Framework



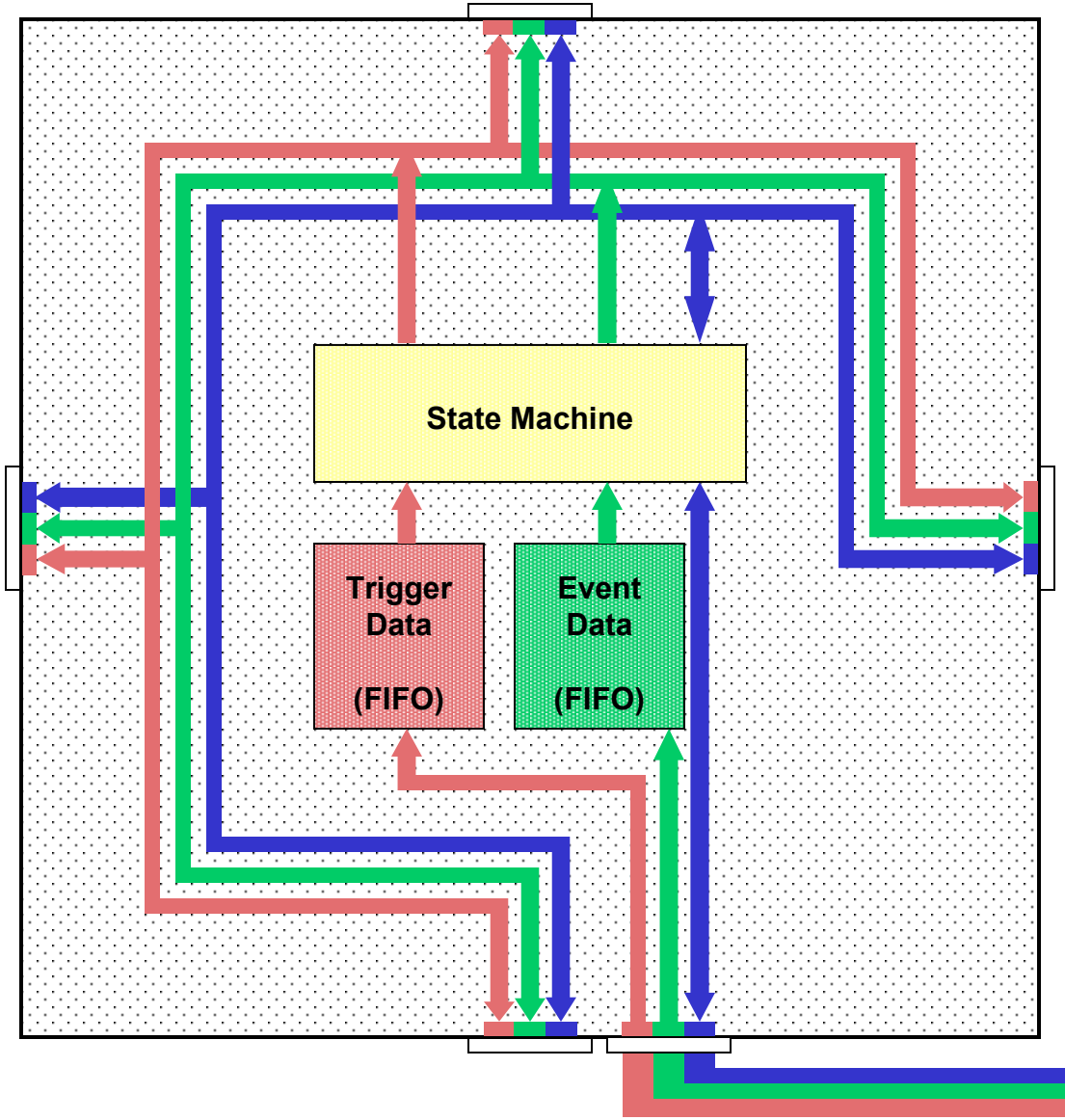
FSW Architecture for EM1



Front End Simulator (FES)

- Document: LAT-TD-1825
- Requirement
  - Provide realistic simulation of TKR, CAL and ACD detectors and front-end electronics including:
    - Events, noise, pile-up, faults and commands
  - Do this for extended periods (full orbit's worth)
    - ~90 minutes at event rates from 2 kHz to 10 kHz
- Implementation
  - Prepare simulated datasets in near-electronics format
  - Drive datasets from PC storage into TEM / AEM cable inputs
    - Data goes through custom transition board
      - Buffers the data
      - Keeps strict timing relationships
      - Same board used for TKR, CAL and ACD … only state machine firmware differs
    - This is a *cable* level simulation

Front End Simulator Transition Board



FES Computer

Intel Pentium @ 2.4 GHz
4 120 GByte disks
High speed bridges

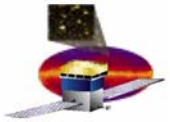
VxWorks Operating System

One computer drives two towers (both TKR and CAL)

Event Data
Cmd/Response
Trigger Data

Front End Simulator (FES)

- Status
  - Trade studies performed to select hardware
    - Driven by
      - Data storage requirement
      - Bandwidth into transition boards
  - Hardware selected
    - Intel Pentium @ 2.4 GHz
    - 4 x 120 GByte disks
    - Moselle split bridge from PC to transition board (PCI standard)
    - This configuration sufficient to drive two towers
    - Two copies purchased
  - PC development environment selected (VxWorks)
  - Skeleton of PC processing coded
  - Transition boards in layout



Event Filtering - Introduction

- Requirements
  - On & off board science
    - Sets photon capture efficiency
- Constraints
  - Processing rate
    - Orbit max trigger rate is ~10kHz: ~100 μsec per event system-wide
  - Downlink bandwidth
    - Orbit average of ~300kbps: ~ 20 Hz background / 10Hz physics
      - Send few events. Filter must reject ~99.8% of background events
      - Send small events. Design dense event formats.
- Processing throughput is a complex function of
  - Event size (longer events result in more memory accesses)
  - Event layout (memory accesses can be saved with good layout)
  - CPU architecture (CPU speed, memory speed, execution units, …)
  - Filtering code
    - Algorithms
    - Implementation

Event Filtering – Development Path

- Using Monte Carlo events generated in GLASTsim
  - Event size of 1 kB includes noise at prescribed rates
  - Event layout is finalized
- Algorithmic development
  - Designing and debugging on SUN/LINUX boxes
  - Measuring performance on Motorola MV2303 and RAD750
- Event features used in current round of analysis
  - TKR layer hit bits (very fast access)
  - ACD tile hit bits (disordered but access still fast)
  - CAL energy sums (slowest access … needs coarse calibration constants)
  - Minimal track finding

Event Filtering - Results

Cut	Events				<Time> μsec	
	Analyzed (%)		Rejected (%)		603	750
No CAL LO + Veto Tile	15420	(100.0)	9923	(64.4)		
ACD Splash Veto (pass 0)	5497	(35.6)	1566	(10.2)	4.5	9.2
CAL < 350Mev + Veto Tile	3931	(25.5)	224	(1.5)		
CAL < 10 Mev + Any Tile	3707	(24.0)	464	(3.0)		
ACD Splash Veto (pass 1)	3243	(21.0)	69	(0.4)	0.3	0.4
TKR tower match with ACD top tile	3174	(20.6)	424	(2.7)		
TKR tower match with ACD side tile	2750	(17.8)	304	(2.0)		
No connection between CAL energy & TKR	2446	(15.9)	1152	(7.8)	5.6	6.7
CAL Energy Layer 0/Total Energy < .01	1294	(8.4)	156	(1.0)		
CAL Energy Layer 0/Total Energy > .90	1138	(7.4)	94	(0.6)	0.1	0.2
Before track finding	**1044**	**(6.8)**	**14376**	**(93.2)**	**5.8**	**10.6**
TKR/ACD matching	1044	(6.8)	262	(1.7)		
Projects into skirt region	782	(5.1)	83	(0.5)		
E < 350 Mev, Number of Tracks < 2	699	(4.5)	461	(3.0)	29.9	40.5
Final	**238**	**(1.5)**	**15182**	**(98.5)**	**7.7**	**13.3**

Event Filtering - Summary

- Current status
  - Rejection rate 98.4%
  - Time (RAD750) 14 μsec
- Still need to go from 98.4% → 99.8%
  - > 95% rejection in 14 μsec/event (RAD750) leaves an interval of 1.4 msec/event for additional processing to go the final 1.4%
  - To preserve 100% margin in one CPU, still have 700 μsec/event
  - This is 50 times the event processing time used so far
- Status
  - Filter development proceeding
  - 1 BAE 750 will be sufficient to do filtering with 100% margin

Software Development Approach

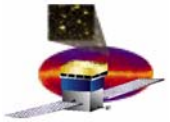
- Software lifecycle model
  - Iterative / incremental development model
  - Multiple builds with increased capability with each build
  - Regression testing on each build
- Requirements flowdown, analysis, review
  - Flowdown from program and system specs
  - Peer reviews
- Design and code inspections / review
  - Top-level design review
  - Detailed design reviews and code inspections on per release basis
- Continuous cycle of development and test
- Code management
  - Formal control through the CMX / CMT / CVS toolchain
- Configuration management
  - Formal control through project management tools
    - Cyberdocs
    - Non conformance reporting system
- Quality assurance and test oversight
  - Independent review of test plans, procedures, scenarios, data
  - Reports directly to LAT QA, systems engineering

Software Design for Safety

- The software safety environment
  - Software cannot damage hardware (hardware protects itself)
  - Reprogrammable on orbit (except for primary boot code)
- The software safety philosophy during development
  - Leverage the fact that software cannot damage hardware
  - Make unexplained conditions "fatal but not serious" and reboot
    - Decreases complexity
    - Increases reliability / robustness
    - Immediate and graceful exit quickly identifies code weaknesses
      - Improves efficiency for producing reliable / robust final code
  - On a case by case basis, develop recovery strategies
    - Fully recoverable: Perform recovery action, continue operation
    - Not recoverable but CPU integrity good: Report to ground and await intervention
    - Not recoverable and CPU compromised: Stay with reboot strategy

Software Fault Detection Methods

- State of health (SOH)
  - Bridge chip built-in test
- Software watchdog
  - All registered tasks must regularly report progress in order for SWD to reset hardware watchdog
- Data validity
  - Checksums
  - Parity bits
  - LCB timeouts, error records
- Data reasonableness checks
  - HSK queries software/hardware regularly
    - Memory correction reports from bridge chip
    - Thermal conditions
    - Power conditions
    - …
  - All instrument configurations read out at beginning and end of all data collection runs (must agree)
  - Event monitoring code on EPUs checks event data for correct format, completeness
  - Event filtering code checks event data for physics consistency



Software Fault Handling

- Fault Recovery
  - On basis of fault identified
    - Execute recovery procedure and continue operation
    - Notify ground and await intervention
    - Reboot
      - Always attempt to save a block of information describing the fault condition in a known fixed memory location so that it can be picked up and sent to ground after the reboot

FSW Package Development

- FSW partitioned into functional blocks based on the SRS
  - Functional blocks are then mapped into packages, the fundamental unit of the code management system
- Package Development
  - Detailed design elements (algorithms, finite state diagrams, logic flows, etc.) and development notes are generated on a per package basis
  - Design information is stored in a Software Development Folder (SDF) which accompanies each package
  - Contents of SDF are version controlled alongside the package's code using the code management system
  - As the software matures, design descriptions from the SDFs evolve along with the code to provide a complete set of detailed design documentation

Task to Package Mapping

		Package																																								
		FMP	HUT	EDP	EOP	EMP	ECL	ECP	GNAT	GCFG	SOP	SMP	SCL	GBM	GRB	HSK	CHP	THP	TCS	MCP	EFP	GPS	LAP	LCP	SDF	SWD	SSR	LIO	CO1553	PBS	Exc	MemTest	BootSh	EEPRO	PCI	CCSDS	Upload	ZLIB	BootUt
Task	1553 service																							●					●	●										
	LCB service																							●				●		●										
	Software 'dog																								●	●			●	●										
	Primitive								●	●												●	●	●	●	●	●	●	●	●										
	ACQ master		●						●	●	●	●	●	●	●					●		●	●	●	●	●	●	●	●	●										
	7 master																					●	●	●	●	●	●	●	●	●										
	HSK master															●	●	●	●					●	●	●	●	●	●	●										
	File master	●																						●	●	●	●	●	●	●										
	ACQ slave		●	●	●	●	●	●													●	●	●		●	●	●	●		●										
	7 slave							●														●	●		●	●	●	●		●										
	HSK slave							●								●	●	●							●	●	●	●		●										
	File slave	●						●																	●	●	●	●		●										
	Boot																											●	●	●	●	●	●	●	●	●	●	●	●	●

Development Environment

- Embedded System
  - Processor / operating system: BAE RAD750 / VxWorks
  - Toolset (Wind River Systems):
    - Language: C
    - Development platform: Sun / Solaris
    - Compiler / linker / binutils: GNU cross compiler suite
    - Debugger: Crosswind
- Host System
  - Processor / operating system: Sun / Solaris or Intel / Linux
  - Toolset (host simulation or cooperating processes):
    - Language: C
    - Development platform: Sun / Solaris or Intel / Linux
    - Compiler / linker / binutils: GNU compiler suite
    - Debugger: GDB / DDD
  - Toolset (test executive and scripting):
    - Python / XML / MySQL / Qt / Perl
- Other Tools
  - Requirements management: DOORS
  - Code / configuration management: CMX / CMT / CVS
  - Autogeneration of documentation: Doxygen
  - Documentation: Microsoft office suite (also Adobe / Framemaker, etc.)

Software Testing Approach

- Software code and package level test
  - Performed on developer's platform or captive embedded system
  - Verify algorithm development, debug software logic
- Software composite test
  - Higher level functionality tests – combine many packages
  - Verify functionality and interfaces
- System build tests
  - Highest level tests – verify / validate against requirements
- Test environment
  - Software / hardware integration and test
    - Performed on FSW test bed with breadboard / brassboard hardware (COTS and then RAD750)
    - Verify software executing on target processors with real-time operating system (VxWorks).
    - Verify software interfaces with input/output hardware in loop
  - Software / system integration and test
    - Performed on flight spacecraft hardware in EGSE environment
    - Verify FSW with flight spacecraft hardware

Component, System, FQT Level Testing

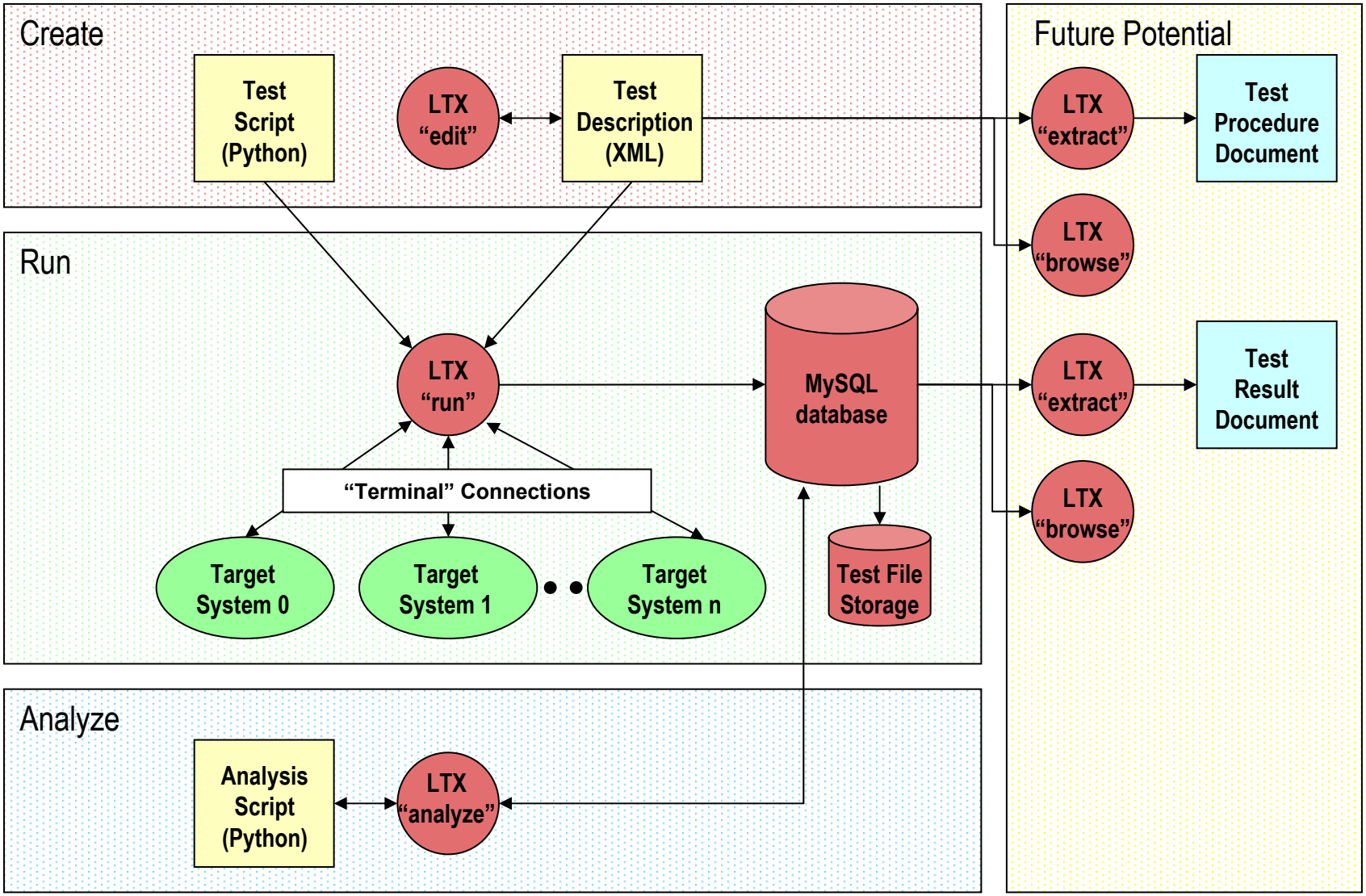
TEST NUMBER		001	002a	002b	003	004	005	006a	006b	007a	007b	007c	007d	008a	008b	009a	009b	010	011	012a	012b	013a	013b	014	015		100	101	102	103		100	101	102	103
LAT FSW RELEASES AND TEST PHASES / **LAT FSW TESTS**	**COMPONENT-LEVEL**	1553 INTERFACE TEST	COMMAND FUNCTIONAL	MEMORY UPLOAD VERIFICATION	NARROWBAND TELEMETRY VERIFICATION	WIDEBAND TELEMETRY VERIFICATION	VEHICLE SIGNALS INTERFACE VERIFICATION	SIU/EPU CPU-TO-CPU COMM FNCTL TEST	EPU INTERNAL CONFIGURATION TEST	T&DF INTERNAL CONFIGURATION TEST	TKR INTERNAL CONFIGURATION TEST	CAL INTERNAL CONFIGURATION TEST	ACD INTERNAL CONFIGURATION TEST	EVENT FILTERING - ALGORITHM VERIFICATION	ELECTRONIC CALIBRATION	FSW AND LAT INITIALIZATION	FSW AND LAT RE-INITIALIZATION AND RECOVERY	LAT POINTING CALCULATIONS	GRB HANDLING/REPOINTING REQUESTS	ADDL SCI PROC - GRB DETECTION AND LOCATION	ADDL SCI PROC - REPOINTING LOGIC	OPERATIONAL MODES - SAFE MODES	OPERATIONAL MODES - SCIENCE MODES	LAT DIAGNOSTIC AND CALIBRATION MODES	LAT THERMAL CONTROL SYSTEM	**SYSTEM-LEVEL**	LAT ON ORBIT INITIALIZATIONS	NORMAL OPERATIONS – SKY SURVEY MODE	NORMAL OPERATIONS – POINTED OBS MODE	LAT PERFORMANCE VERIFICATIONS	**FORMAL QUALIFICATION TESTS**	LAT ON ORBIT INITIALIZATIONS	NORMAL OPERATIONS – SKY SURVEY MODE	NORMAL OPERATIONS – POINTED OBS MODE	LAT PERFORMANCE VERIFICATIONS
ENGINEERING MODEL RELEASES																																			
EM1					P			P	P	X	X	X		T	P	P																			
EM2		P	P	X	P	P		X	X	X	X	X		T	P	P			P			P	P	P											
FULL LAT RELEASE																																			
AMBIENT		X	X	X	X	X	X	X	X	X	X	X	X	T	X	X	X	X	X	T	X	X	X	X	P		X	X	X			X	X	X	
LAT BEAM TEST			X		X	X										X														X					X
THERMAL VACUUM HOT			X		X	X										X									X		X	X	X			X	X	X	
THERMAL VACUUM COLD			X		X	X										X									X		X	X	X			X	X	X	

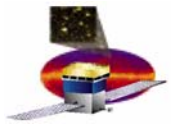
X = Test is performed P = Partial test is performed T = Test performed on separate testbed with simulated data

Test Executive (LTX)

- Goal is to provide a uniform and efficient method to
  - Create and track a test description/procedure (in a computer format)
  - Run a test and capture the results into persistent storage (database)
  - Attach an analysis suite to a test and capture the analysis results
- Potential
  - A GUI editor for the test descriptions
  - Browsing of test descriptions through a dedicated GUI or the web
  - Browsing of test results through a dedicated GUI or the web
  - Semi-automatic generation of test descriptions
  - Semi-automatic generation of test results

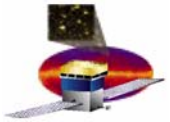
Test Executive (LTX)





Test Executive … Create Test

- Create and track a test description/procedure (in a computer format)
  - Code developer or external organization prepares two files
    - Test description
      - A structured XML file
      - Editor provided by the test executive (enforces structural rules)
      - Top level includes
        - » A test description block (free text)
        - » A test procedure block (free text)
        - » Tie back to unit / composite / system test identity or requirement
      - Describes one or more target systems and for each system
        - » The system type (unix or embedded)
        - » A list of capabilities the system must provide (ethernet, 1553, LCB, …)
        - » A list of software to run (package/constituent, package/constituent, …)
      - Input vectors, list of produced files, …
    - Test script
  - These files reside in standard CMX packages
    - Code managed and versioned using CMX / CMT / CVS
    - Can be either
      - Part of a main line package (unit level / white box testing)
      - In a dedicated test package (composite or system level / black box testing)



Test Executive … Run Test (0)

- Run a test and capture the results into persistent storage (database)
  - Tests are run by the test executive following the recipe
    - Instantiate the system(s) defined in the test description
      - Description says “type=unix” “capability=VxWorks”
      - Test executive instantiates this as “tersk” at SLAC, “goose” at NRL
    - Connect to the instantiated system(s) with “terminals”
      - Wind River Systems Tornado/WindSh interface to embedded systems
      - Simple terminal interface to unix systems
    - Identify the versions of software to use
      - Description says use software “package=CCSDS” “constituent=ccsds”
      - Parameter to test says “use EM1 release”
      - Test executive look-up finds CCSDS/ccsds was at version V1-2-3 in EM1
    - Set up the software environment on the target system(s)
      - Embedded systems: Load the identified package/constituent/version
      - Unix systems: Set the CMX branches correctly
    - Run the user supplied test script
      - “Library” functions in the text executive provide
        - » Write functions to send commands to subsystems individually on “terminals”
        - » Read functions to capture output from subsystem “terminals”
        - » Boilerplate functions to do run time identification of software/hardware
    - Clean up

Test Executive … Run Test (1)

- Run a test and capture the results into persistent storage (database)
  - Captured into a random access database (MySQL)
    - All tests uniquely identified
    - Test associated files (e.g. terminal logs) saved in protected area and referenced by entries in the database
  - Information captured
    - Date/time, where, who, …
    - Pointer to test description (including version)
    - Information instantiated by test executive for this run of the test
      - System selection
        - » Unix: Host type (Sun, Linux), host name, …
        - » Embedded: Real board type (mv2304, mcp750, rad750), …
      - Software version selection
      - …
    - Where available, information gathered at run time
      - Confirmation of software versions loaded
      - LCB hardware revision level
      - …
    - All traffic between the test executive and the target system(s) "terminals"
    - Files generated by test (as specified in the description)

Test Executive … Analyze Test

- Test results are highly variable
  - Simple ASCII files, unformatted binary files, CCSDS formatted files, histograms, …
  - Difficult (impossible ?) for test executive to satisfy all analysis demands
- What the test executive *will* provide
  - The ability for the test designer to define an analysis stage
    - Define an analysis script
    - Define the innies and outies to the analysis stage
  - Capture of analysis products indexed to original test

Test Executive Status

- Tools
  - Tools chosen
    - Python for scripting
    - XML to capture test descriptions
    - MySQL for database functions
  - Tool choice drivers
    - Python comes with many add-ons to make life easier
      - PyXML to manage XML files
      - PyMySQL to talk to MySQL databases
      - PyQt to design and implement GUI interfaces
    - The same toolset is in use with I&T
- Implementation
  - XML file structure definition well advanced
  - XML file editing from command line demonstrated
  - Opening terminals to both host and embedded systems demonstrated
  - Plan an “alpha” release to developers in next few weeks
    - Move beyond simple “Hello World” test exercises
    - Start training developers on the system and get feedback for improvements

Summary

- FSW requirements and design understood
  - Architecture
  - Interfaces
  - Functionality
- Design for EM1 complete
  - Development and testing in progress
- Development path through EM1, EM2, FU phases identified
- On parallel paths, work continues on
  - Boot code
  - 1553 and spacecraft interfaces
  - Event filtering software