AMC13 v1 Operational Specification

Charlie Hill

21 December 2012

Revision History:

02 Mar 2013, ch: overall format adjustments and writing cleanup

28 Feb 2013, ch: edits to the “AMC13 Links” section

27 Jan 2013, ch: edits in response to requests from esh on 2 Jan

2 Jan 2013, esh: edits with change tracking based on 21 Dec version

21 Dec 2012, ch: sent to esh for comments

Table of Contents

1 AMC13 Overview

The AMC13 module is the micro telecommunications architecture (μTCA) data concentration and clock distribution card designed specifically for the Compact Muon Solenoid (CMS) Hadronic Calorimeter (HCAL) Upgrade back-end electronics.

The AMC13 is responsible for processing data fragments received from the μTCA crate's advanced mezzanine cards (AMCs) and constructing formatted events for readout. The module is also responsible for acquiring trigger and timing information and distributing it to the rest of the μTCA crate.

The following documentation discusses the layout and functionalities of the AMC13, as well as important details regarding the external hardware with which it interacts.

1.1 μTCA Overview

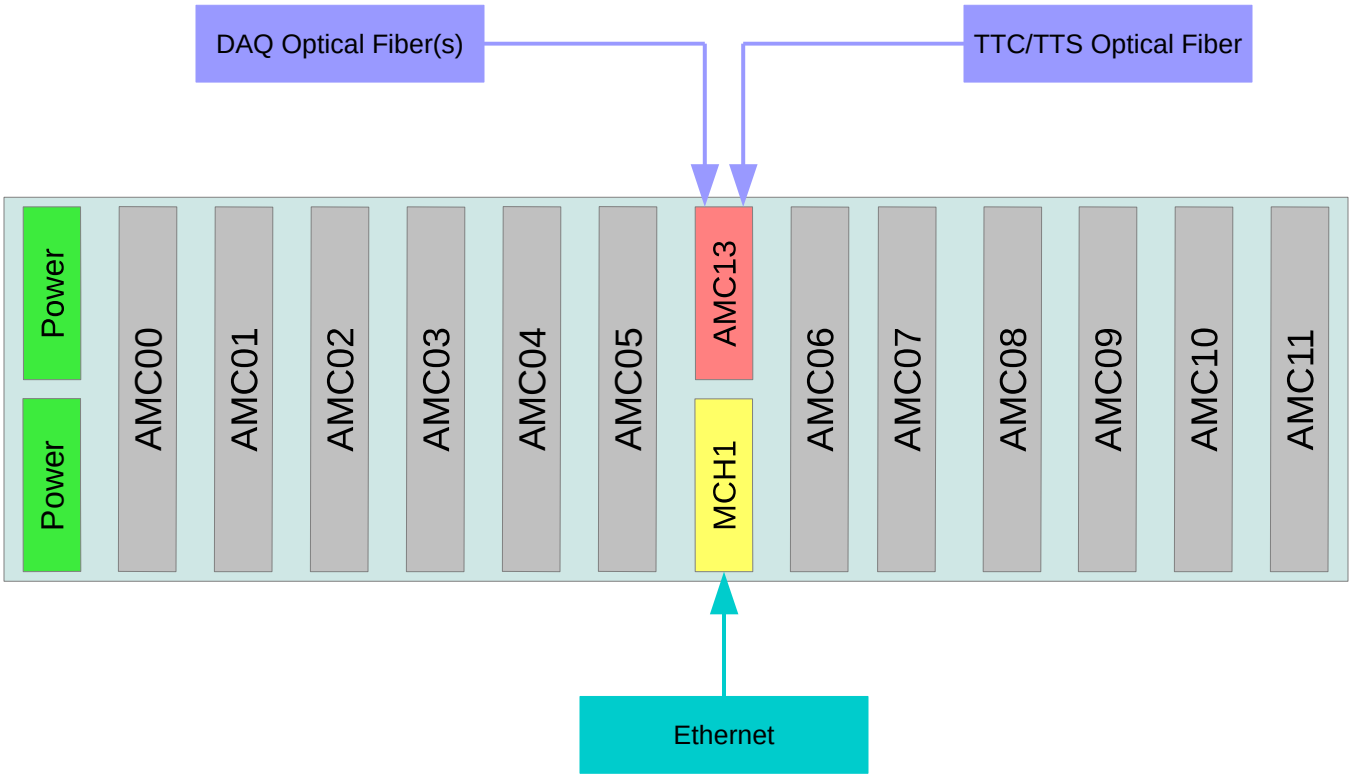


Illustration 1.1: μTCA Crate

Illustration 1.1 shows the basic layout of a dual-star μTCA crate with an AMC13 in the redundant microTCA carrier hub (MCH) site. There are twelve AMCs which are connected to the commercial MCH card (MCH1) and the AMC13 via backplane fabric connections. MCH1 supports an external Ethernet Interface (see section 5.1) which allows the user to communicate with all installed AMC modules as well as the AMC13. The AMC13 (MCH2) has four small form-factor pluggable optical transceivers (SFPs) on the front panel which provide bidirectional links to the Trigger, Timing, and

Control system (TTC) and CMS central data acquisition (cDAQ), both of which are external to the crate.

1.1.1 CMS Specifics

The AMC13 and its role within the μTCA system[A] have been developed by the HCAL group for future use as the back-end data concentrator card on the CMS experiment; therefore, there are some CMS-specific features of the μTCA system which need to be mentioned.

First, the AMCs are, in the context of the HCAL Upgrade, micro HCAL Trigger Readout (μHTR) boards[B]. These μHTRs each take twelve optical fiber connections from the front-end detector electronics and create event fragments from the raw ADC and TDC channel data. These data are then sent to the AMC13 for processing.

Second, the current μTCA layout for the HCAL Upgrade uses one of two MCH1 modules: the Vadatech U202 or the NAT MCH. Note that these are *not the only* commercial MCH modules available for the μTCA crate but *are the only* two that have been tested with the AMC13.

Third, since the AMC13 has a very specific function within HCAL, the collaboration often refers to it as the Data and Timing Readout card (DTC). This jargon appears throughout HCAL Upgrade documentation.

Fourth, all events prepared by the μHTRs and AMC13 follow a specific HCAL Upgrade data format[C]. Even though AMC13 development has been exclusively for HCAL thus far, the module has potential for application outside of CMS; therefore, the following document will take a very general approach to the AMC13's operational functionality.

1.2 AMC13 Hardware Overview

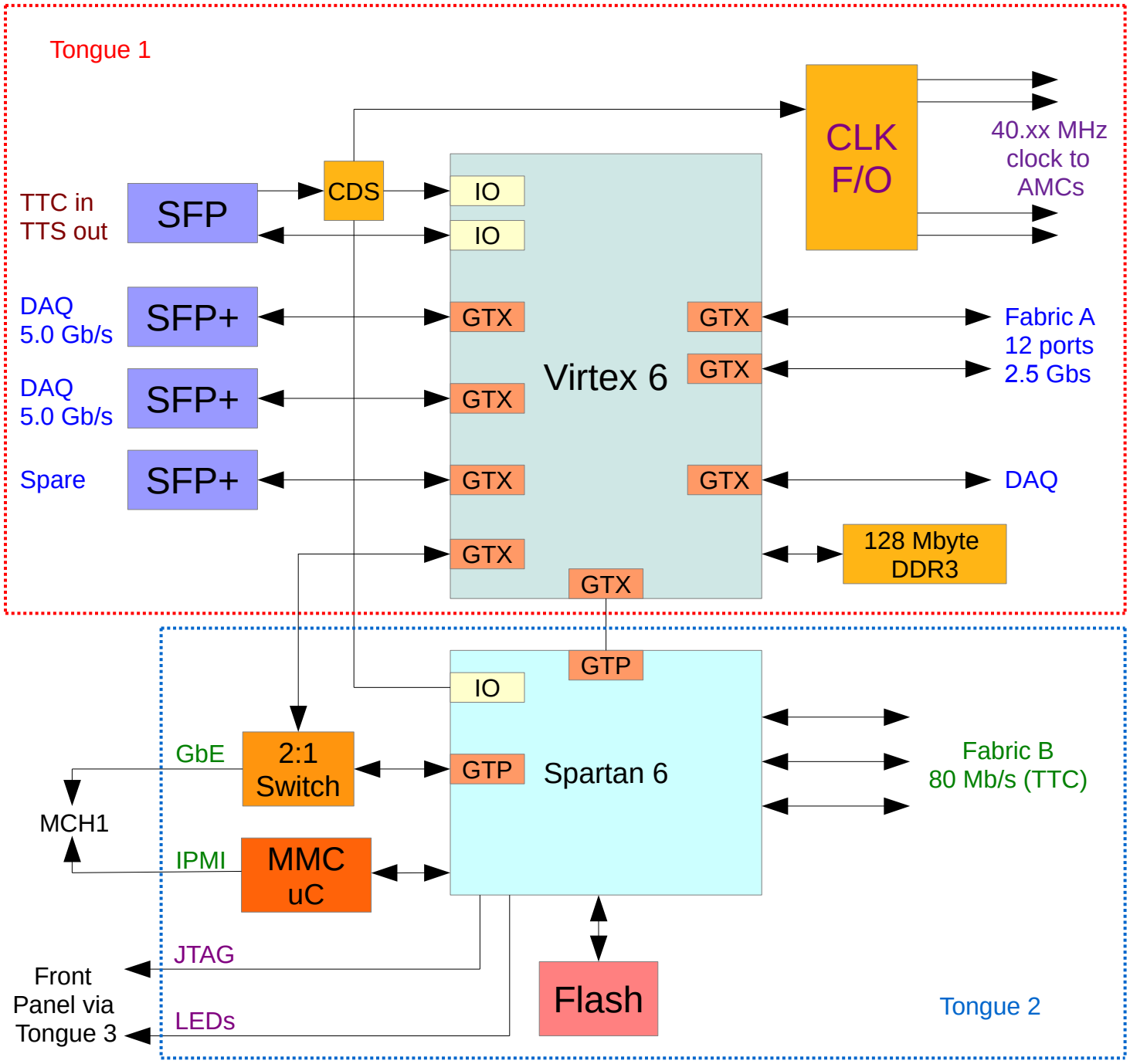


Illustration 1.2: AMC13 Hardware Layout

The AMC13[D] is comprised of three printed circuit board (PCB) tongues, two of which (T1 and T2) have the module's key features. T1 carries a Virtex-6 XC6VLX130T FPGA[1] which contains most of the board's functionality, including TTC clock recovery, the acquisition of received event fragments

1 AMC13 Revision 2 will have a Kintex-7 FPGA instead of a Virtex-6

from the AMCs, building events, general monitoring, handling the backplane links, sending out DAQ data, and saving built events in local memory. T2 contains a Spartan-6 XC6SLX25T FPGA which is responsible for distributing TTC information to the μTCA system as well as managing the AMC13's flash memory and Module Management Controller (MMC) interface[E].

Illustration 1.2 shows the basic layout of the AMC13 tongues and FPGAs. Additional elements include, on T1, a clock-data separator (CDS) which deciphers the TTC input, a Gigabit Ethernet (GbE) switch between the Spartan and Virtex chips, and double data rate synchronous dynamic random-access memory (DDR SDRAM or just SDRAM) for storing built events. All connections marked “GTX” and “GTP” are high-speed serializer/deserializer (SERDES) links, while all connections marked “IO” are basic input-output ports.

The AMC13 front panel contains four SFP sites, Joint Test Action Group (JTAG) headers for MMC and FPGA programming, a USB connection to the MMC console, and LEDs for easy monitoring of the MMC. T3 is an optional PCB used for initial programming.

1.3 AMC13 Network Settings

The AMC13 is accessible via an Ethernet connection to the commercial MCH module, and therefore is accessible via the network. Each AMC13 has two IP addresses, one for the Spartan FPGA and another for the Virtex FPGA, and their network assignments are based on the AMC13 serial number (SN):

Virtex IP Address: 192.168.1.(255-2*SN)

Spartan IP Address: 192.168.1.(254-2*SN)

If powered up and fully operational, the AMC13 should respond to “pings” at these addresses:

```
$ ping 192.168.1.238
```

This jumper-based address setting scheme is incompatible with other μTCA hardware in CMS and will be changed in the future.

1.4 AMC13 Block Diagram

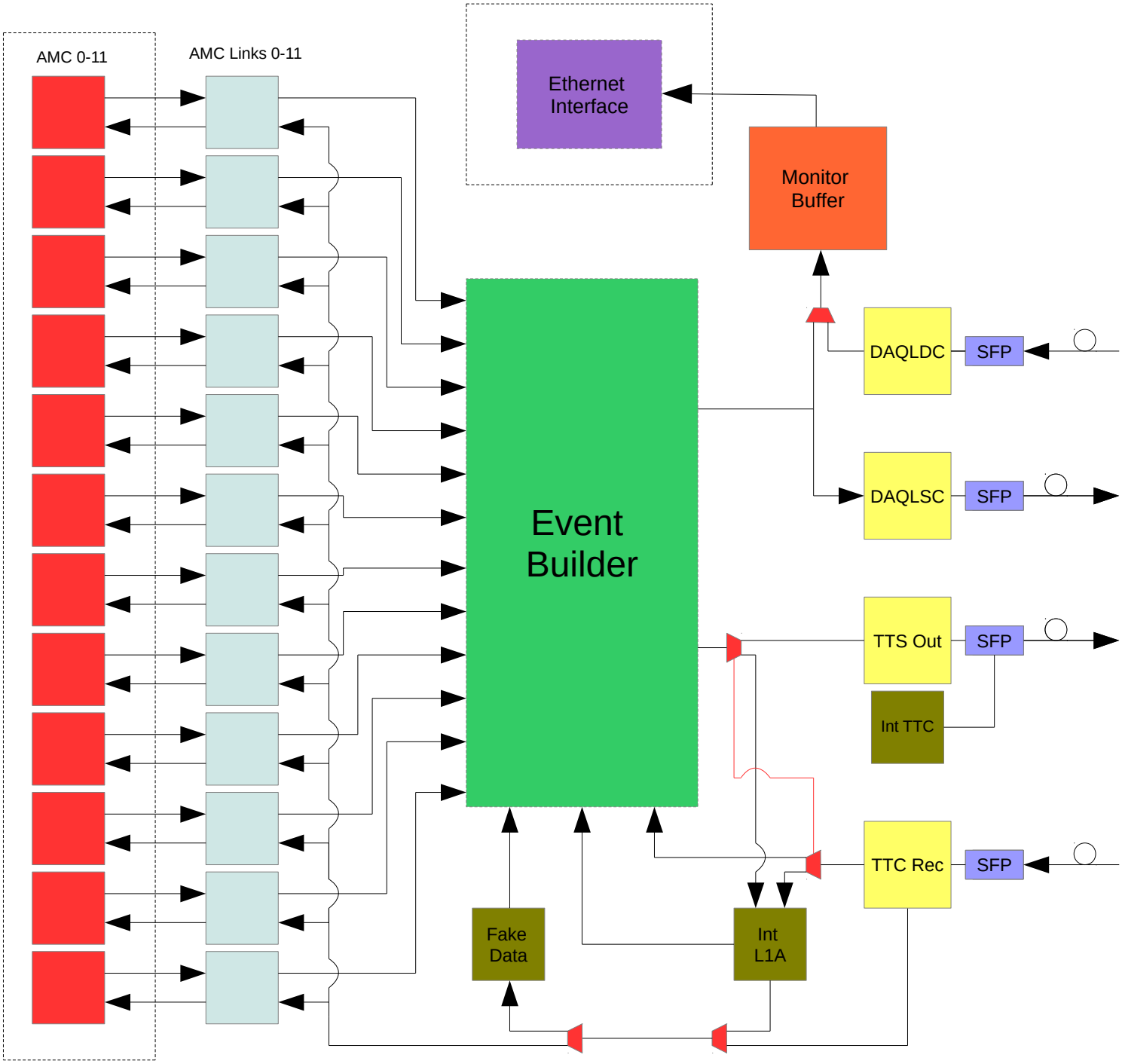


Illustration 1.3: AMC13 Block Diagram

1.5 Control and Status Registers

Table 1.1 presents overarching registers for the AMC13, namely for reset, AMC13 identification, and general hardware status.

Registers	***Bit(s)***	***R/W***	***Description***
0x0	0	W	General Reset
0x0	1	W	Zero all counter registers
0x0	24-31	R	AMC13 Serial Number
0x1	16-31	R	AMC13 Virtex Firmware Version
0x7	0-23	RW	Source ID (FEDid)
0x30	NA	R	Virtex chip DIE temperature (units of 0.1 degree Celcius)
0x31	NA	R	1.0V analog power voltage (in mV) (For SN >= 0x10 only)
0x32	NA	R	1.2V analog power voltage (in mV) (For SN >= 0x10 only)
0x33	NA	R	1.0V power voltage (in mV)
0x34	NA	R	1.5V power voltage (in mV) (For SN >= 0x10 only)
0x35	NA	R	2.5V power voltage (in mV)
0x36	NA	R	3.3V power voltage (in mV) (For SN >= 0x10 only)
0x37	NA	R	3.6V power voltage (in mV) (For SN >= 0x10 only)
0x38	NA	R	12V power voltage (in mV) (For SN >= 0x10 only)

Table 1.1: General AMC13 Registers

1.5.1 General Reset

The General Reset command resets the AMC13 logic, including a reset of all counters, error bits, and status registers.

Table 1.2 presents registers that ***will not*** be zeroed upon a general reset. Note that all of these are read/write, are user-defined, and are used to configure the AMC13 for a run.

Registers	***Bit(s)***	***R/W***	***Description***
0x1	0-15	RW	Run-configuration bits for the AMC13
0x2	0-31	RW	Configure the SDRAM mode of operation
0x3	0-31	RW	Enable corresponding AMC links
0x6	0-4	RW	Bc0 AMC link offset configuration
0x7	0-31	RW	AMC13 FEDid configuration
0x8	0-31	RW	TTC OrN and BcN offset configuration
0x1c	0-31	RW	Local L1A configuration register

Table 1.2: Registers Not Affected by an AMC13 General Reset

2 Descriptions of Functional Elements

Chapter 2 is organized to follow the flow of data and control through the AMC13. Each block in Illustration 1.3 is explained in the following sections (except for “Int L1A”, “TTC Rec”, “Fake Data”, and “DAQLDC”, which are described in Chapter 3), with special attention to functionality, buffering, and register footprint.

2.1 TTC Receiver

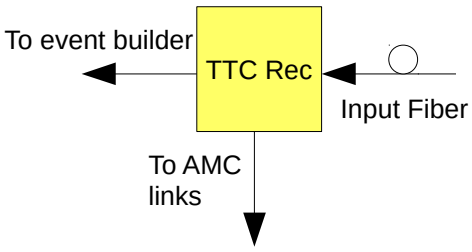


Illustration 2.1: TTC Receiver

2.1.1 About the TTC system

Trigger, Timing and Control[F] (TTC) is responsible for the distribution of synchronous clock signals, Level 1 Accepts (L1As or triggers), and command broadcasts to electronics throughout the LHC. With respect to the µTCA system, the TTC system is primarily responsible for sending L1As and clock signals to the AMC13 TTC receiver, which in turn relays the trigger and clock information to the rest of the crate. The following paragraphs discuss the TTC signal received by the AMC13.

TTC Input Signal

The TTC input signal is divided into two channels, 'A' and 'B', and is encoded using 160.32 Mbaud biphase mark encoding. The 'A Channel' is dedicated to sending Level-1 accepts and rejects, where the 'B Channel' is dedicated to sending broadcast commands.

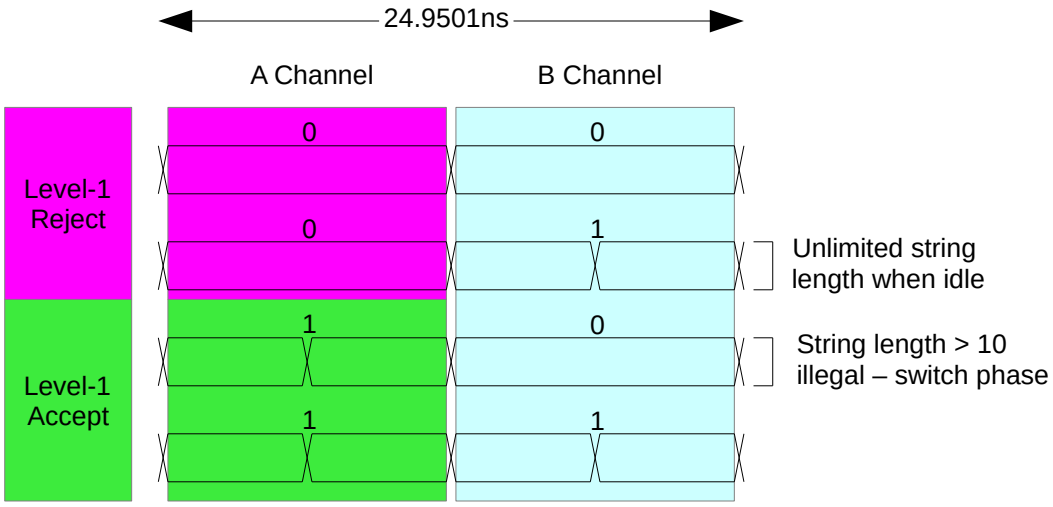


Illustration 2.2: TTC Input Signal

TTC Broadcast Command

The TTC system supports global broadcast commands, the format and contents of which are shown in Illustration 2.3.

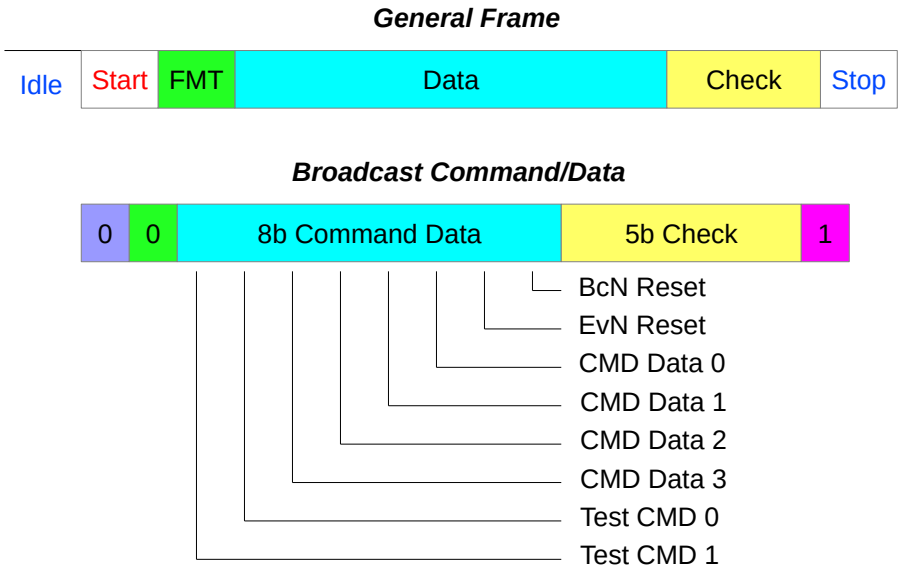


Illustration 2.3: TTC Broadcast Command Format

The bit contents of the TTC broadcast commands, as well as whether they are recognized by the AMC13, are presented in Table 2.1. Those commands which the AMC13 does not recognize are not yet well defined and may be implemented at a later time.

Command	*Broadcast <7:0>*	*Recog by AMC13?*	*Meaning*
Orbit-count Reset	001x1xxx	Yes	Reset OrN
Resync	010x1xxx	No	Reset DAQ path
Hard Reset	011x1xxx	No	Reset DAQ path
Start	100x1xxx	No	Start accepting L1As
Stop	101x1xxx	No	Stop accepting L1As
Calib Trig	110x1xxx	No	Generate Calibration Trigger
Event-count Reset	xxxxxx1x	Yes	Reset EvN
Bunch-count Reset	xxxxxxx1	Yes	Reset BcN

Table 2.1: TTC Broadcast Commands

2.1.2 General Functionality

The AMC13 TTC Receiver's primary responsibility is to relay L1A information (in packet form) to the AMC13 Event Builder as well as each AMC Link (see section 2.2.2). Table 2.2 describes the most common information within an L1A signal.

Name	*Description*
L1A	Level 1 Trigger
Bc0	Bunch-count zero. Synonymous with bunch-count reset (BcR)
OcR	Orbit-count reset
EcR	Event-count reset

Table 2.2: TTC L1A Information

The TTC Receiver also distributes the raw TTC bitstream throughout the µTCA crate and relays TTC broadcast commands to the AMC13 Event Builder for evaluation. Illustration 2.4 shows how the TTC input is acquired, sorted, decoded, and distributed throughout the AMC13 and the µTCA system.

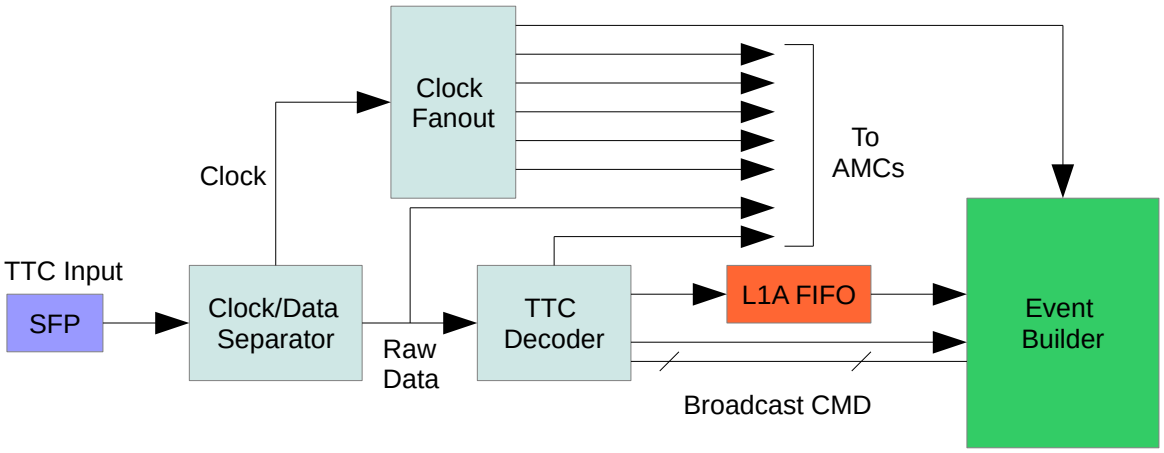


Illustration 2.4: TTC Distribution

2.1.3 Buffering

L1As are buffered in the 'L1A FIFO' block, as shown in Illustration 2.4, before being sent to the Event Builder for processing. This FIFO can hold up to 256 L1As and is used to determine the TTS state of the AMC13. For more information on the TTS and its interaction with the TTC L1A buffer, see section 2.5.

The L1A information is also sent to and buffered at the AMC Links; see section 2.2.3 for more information on these FIFOs.

2.1.4 Status Registers

Table 2.3 gives the status bits designated to monitor the received TTC input for bit errors, hardware malfunctions, and bad synchronization.

If bits 6-8 of register 0x0 are non-zero, the number of errors detected can be found in the counter registers 0x40-45. If bit 9 of register 0x0 is non-zero, the AMC13 TTS state (see section 2.5) will be 'SYN', and counter registers 0x4e-4f will be incrementing.

Register	*Bit(s)*	*R/W*	*Description*
0x0	5	R	TTC not ready (fiber disconnected)
0x0	6	R	TTC bunch-count error (incorrect Bc0 spacing)
0x0	7	R	TTC single-bit error (corrected error in TTC bit stream)
0x0	8	R	TTC multi-bit error (uncorrected error in TTC bit stream)
0x0	9	R	TTC sync lost (L1A FIFO overflow)
0x0	13	R	L1A FIFO overflow warning
0x40-41	NA	R	TTC single-bit errors
0x42-43	NA	R	TTC multi-bit errors
0x44-45	NA	R	TTC Bc0 errors
0x4e-4f	NA	R	L1A sync lost time (150MHz)

Table 2.3: TTC Status Registers

Further details on various TTC error conditions:

The encoded TTC bit stream includes Hamming code check bits which can correct all single-bit errors in a single transmission and detect all double-bit errors as well as many other errors.

The single-bit error flag indicates that there was a data error in the TTC but that the error was corrected. The double-bit error flag indicates that the data was corrupted.

The bunch-count error flag indicates that successive Bc0 broadcast commands were not separated by 3563 clock cycles (the length of one LHC orbit).

2.1.5 Control Registers

Table 2.4 presents bits which control the TTC receiver. These bits enable the transmission of broadcast commands to the Event Builder and set a local OrN and/or BcN offset. The BcN and OrN offsets are added to the local bunch count number and orbit number, respectively, before the values are stored in the to-be-built event's header.

Register	*Bits(s)*	*R/W*	*Description*
0x1	5	RW	Enable TTC broadcast commands
0x1	13	RW	If '1', saved SDRAM data is sent downstream to the DAQ Link upon a TTC resync. If '0', then the data is flushed upon a TTC resync. This functionality is not yet implemented.
0x8	0-11	RW	Local, unsigned bunch-count offset (default to 0x0, no offset)
0x8	12	RW	If '1', only one bunch-count reset is allowed following a system reset
0x8	16-19	RW	Local, unsigned orbit number offset (default to 0x0, no offset)

Table 2.4: TTC Control Bits

2.2 AMC Links

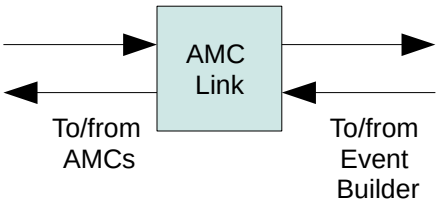


Illustration 2.5: AMC Link

2.2.1 General Functionality

The AMC Links are fabric connections from the AMC13 to the rest of the µTCA crate via the backplane.

The firmware which manages the AMC links is partitioned into two parts, one on either side of the link. This format allows for the buffering of DAQ and L1A data on both the AMC and AMC13 sides of the link while also checking for errors and mismatches on each side.

Illustration 2.6 shows the link between the AMC13 and an AMC card in the µTCA crate. **Fabric A**, which runs between T1 of the AMC13 (specifically the Virtex 6 FPGA) and the AMC card, is a 2.5Gb/s[2] high-speed bidirectional link that carries outgoing information—such as initialization requests—as well as incoming information—such as AMC event fragments. **Fabric B**, which runs between T2 of the AMC13 (specifically the Spartan 6 FPGA) and the AMC card, is an 80Mb/s unidirectional link which carries raw TTC data (see section 2.1.1) to the AMCs.

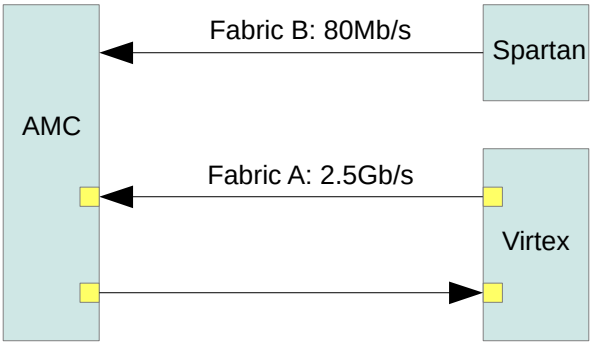


Illustration 2.6: uTCA Fabric Connections

2.2.2 Protocol

Data is sent over Fabric A in 8b/10b[G] encoded packets of maximum length 4kB (1023 16-bit words). Each type of packet is presented in Table 2.5, and the contents of each type are shown in Illustrations 2.7, 2.8, and 2.9. Note that the packets are organized little endian.

2 It is expected that in the final AMC13 firmware this link will operate at 5.0Gb/s

Packet Type	***Direction***	***Description***
Init Request	Outgoing	Packet carrying a request to activate the AMC link
Init Ack	Incoming	Acknowledgement by the AMC side of the link of a successfully received “Init Request” packet
L1A Info	Outgoing	Packet carrying trigger info to the AMC side of the link for error-checking AMC event fragments
L1A Info Ack	Incoming	Acknowledgement by the AMC side of the link of a successfully received “L1A Info” packet
Counter Info	Incoming	Packet carrying counter values from the AMC side of the link to the AMC13 for storage on the Virtex chip
Counter Info Ack	Outgoing	Acknowledgement by the AMC13 of a successfully received “Counter Info” packet
Event Data	Incoming	Packet carrying AMC event fragment data to the AMC13 for event building
Event Data Ack	Outgoing	Acknowledgement by the AMC13 of a successfully received “Event Data” packet

Table 2.5: AMC Link Packet Definitions

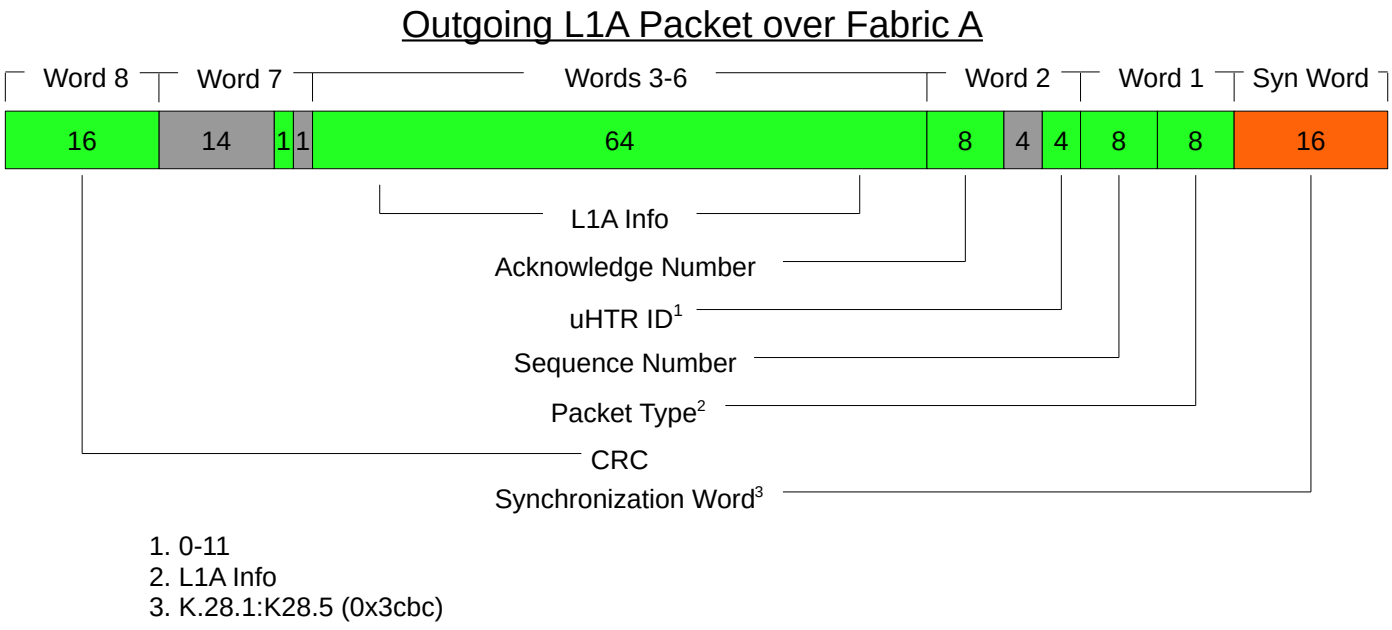
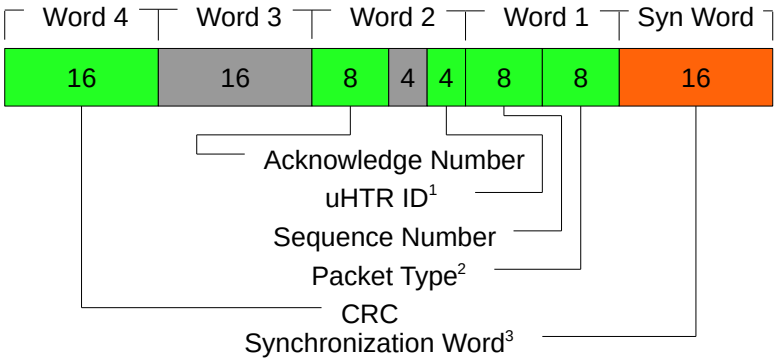


Illustration 2.7: AMC13 Fabric A L1A Info Packet Format

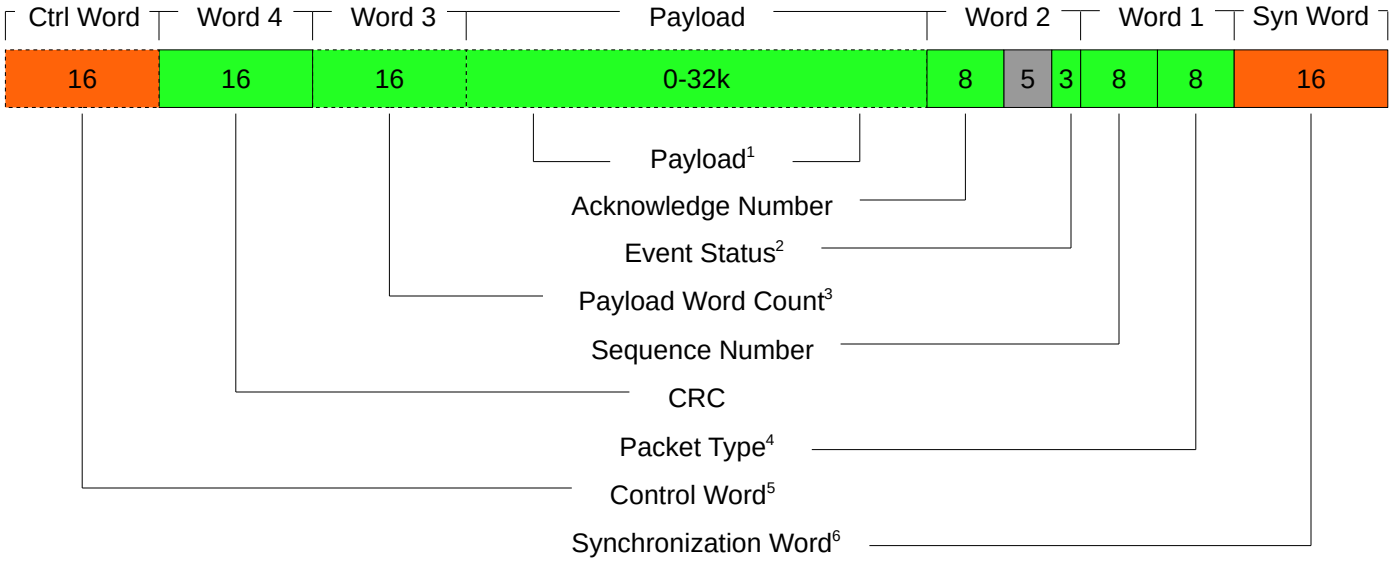
Outgoing Packet over Fabric A



1. 0-11
2. Init Request, Counter Info Ack, or Event Data Ack
3. K.28.1:K.28.5 (0x3cbc)

Illustration 2.8: AMC13 Fabric A Outgoing Packet Format

Incoming Packet over Fabric A



1. Variably sized. Exists only for the 'Counter Info' and 'Event Data' packet types. Otherwise, this section will be of size zero.
2. Flags the presence of packet mismatches.
3. Exists only for the 'Counter Info' and 'Event Data' packet types. Otherwise, these 16 bits are all zero.
4. Initialization Ack, Counter Info, L1A Info Ack, or Event Data
5. If this packet is also the end of an event, it is trailed by an a control word K.23.7:K.28.2 (0x5cf7)
6. K.28.1:K.28.5 (0x3cbc)

Illustration 2.9: AMC13 Fabric A Incoming Packet Format

NOTE: If the link is idle, the “idle” word K.28.6:K.27.7 (0xdcfb) and “sync” word K.28.1:K.28.5 (0x3cbc) are sent in an alternating pattern across the link. When a packet is ready to be sent, it grapples onto a the next available synchronization word.

Raw TTC data is sent over Fabric B in a two-channel bitstream. See section 2.1.1 for further details on the TTC protocol.

Notice that the AMC13 sends L1A packets over Fabric A *as well as* raw TTC data over Fabric B. This is for error-checking purposes on the AMC side of the link. After the AMC builds an event, the L1A info from that fragment can be compared against the corresponding L1A packet from the AMC13 and checked for discrepancies.

2.2.3 Buffering

“L1A Info” packets being sent from the AMC13 to the AMCs via Fabric A are buffered in a 128-packet FIFO on the AMC13 side of the link *as well as* another 128-packet FIFO on the AMC side of the link, as shown in Illustration 2.10. These packets are sent from the AMC13 to the AMC non-periodically: as soon as there is room in the AMC L1A FIFO, the next packet is sent over.

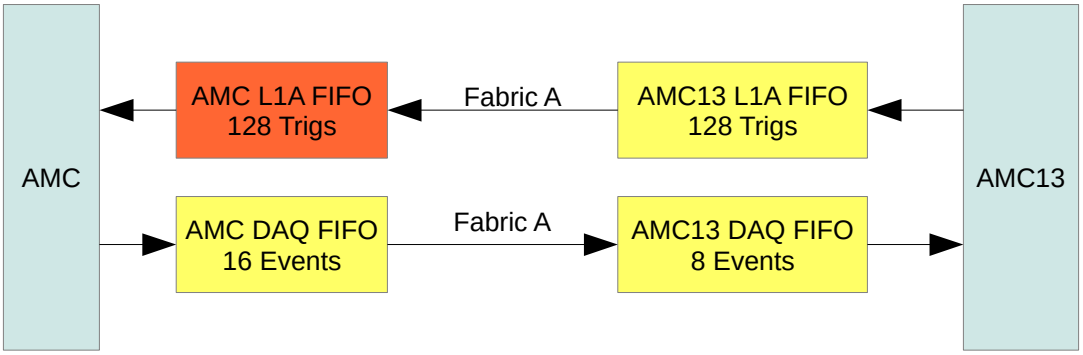


Illustration 2.10: AMC Link Buffering

Incoming DAQ data from the AMCs are held in an 8-event FIFO on the AMC13 as well as a 16-event FIFO on the AMC side of the link, as shown in Illustration 2.10. An event fragment is read out of the FIFO when the Event Builder is ready to build the next event. If the AMC13 senses an imminent overflow, registers 0x16-17 corresponding to this AMC start counting (see Table 2.7).

2.2.4 Status Registers

Table 2.6 shows registers which monitor the state of the links between the AMCs and the AMC13. These status registers come in groups of 12 bits, one bit for each port. For instance, bit 0/16 corresponds to AMC00, and bit 11/27 corresponds to AMC11.

Register	*R/W*	*Bits*	*Description*
0x3	RW	0-11	Corresponding bit indicates whether AMC Link is enabled
0x3	R	16-27	Corresponding bit indicates whether AMC Link is ready
0x5	R	0-11	Corresponding bit indicates whether AMC Link firmware is out of date
0x5	R	16-27	Corresponding bit indicates whether AMC Link has lost bit synchronization
0x6	R	16-27	Corresponding bit indicates whether AMC Link sees consistent orbit-signal timing

Table 2.6: AMC Link Status Bits

2.2.5 Counter Registers

Since the AMC13 link firmware lives on both the AMC side and AMC13 side of the link, there exist counters on either side which check the integrity of the data before and after transit. Illustration 2.11 depicts the link layout.

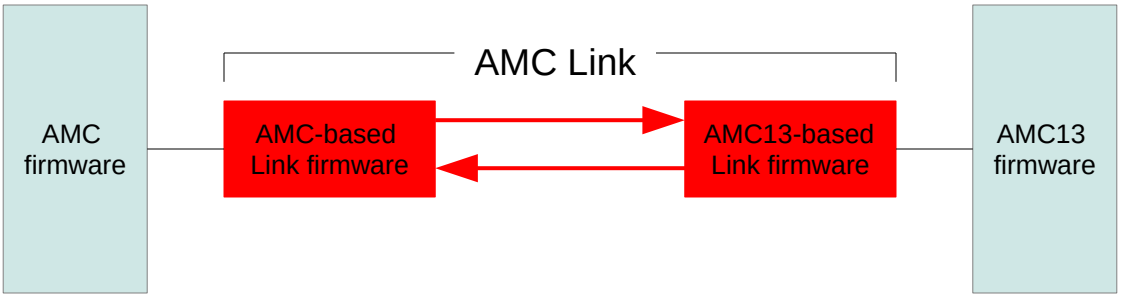


Illustration 2.11: AMC Link Firmware Layout

Note that because counter values from *both* sides of the link are stored on the AMC13 (on the Virtex chip), the values tallied on the AMC side must be sent over periodically and stored for readout on the AMC13.

AMC Side of the Link Counters

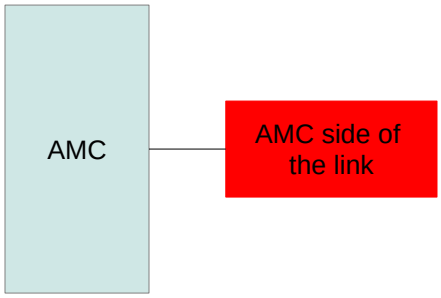


Illustration 2.12: AMC - AMC Link

Table 2.7 shows counter registers which track data flowing through the AMC side of the link. These counters are tallied on the AMC side of the link, and their values are sent across Fabric A every ~262ns

for storage on the Virtex 6 FPGA. These AMC counter values are compared against counters on the AMC13 side to verify the link's integrity and to check for data corruption that may have occurred in transit.

All counters are 64 bits and can be read at Virtex registers {0x800 + (0x80*AMC#) + “Register”} for each (enabled) AMC (0-11).

Register	***R/W***	***Description***
0x0-1	R	“L1A Info” packets accepted by the AMC side of the link from the AMC13
0x2-3	R	“Event Data Ack” packets received by the AMC side of the link from the AMC13. In other words, “Event Data” packets sent from the AMC side of the link which have been acknowledged by the AMC13.
0x4-5	R	“L1A Info” packets from the AMC13 which have been aborted by the AMC side of the link (due to data corruption)
0x6-7	R	EvN mismatches found on the AMC side of the link between trigger info from an “Event Data” packet and the corresponding “L1A Info” packet from the AMC13
0x8-9	R	OrN mismatches found on the AMC side of the link between trigger info from an “Event Data” packet and the corresponding “L1A Info” packet from the AMC13
0xa-b	R	BcN mismatches found on the AMC side of the link between trigger info from an “Event Data” packet and the corresponding “L1A Info” packet from the AMC13
0xc-d	R	Events received by the AMC side of the link from the AMC. Note that since events can span more than one packet, this number is not in general equal to the number of “Event Data” packets sent over the link (0x10-11)
0xe-f	R	“Counter Info Acknowledge” packets received by the AMC side of the link from the AMC13. In other words, “Counter Info” packets sent from the AMC side of the link which have been acknowledged by the AMC13.
0x10-11	R	“Event Data” packets sent from the AMC side of the link which were not acknowledged by the AMC13 and therefore were resent
0x12-13	R	Events with CRC errors, as determined by the AMC side of the link
0x14-15	R	Events whose trailer's EvN does not match its header's EvN
0x16-17	R	Time spent with the event buffer on the AMC side of the link almost full (150 MHz)
0x18-19	R	Total DAQ words sent from the AMC to the AMC13, as determined by the AMC side of the link
0x1a-1b	R	DAQ header words sent from the AMC side of the link to the AMC13
0x1c-1d	R	DAQ trailer words sent from the AMC side of the link to the AMC13
0x1e-1f	R	Events sent from the AMC side of the link to the AMC13 whose EvN is NOT one greater than the previously-sent event (all EvNs should be sequential during normal operation)

Table 2.7: AMC Side of the Link Counter Registers

AMC13 Side of the Link Counters

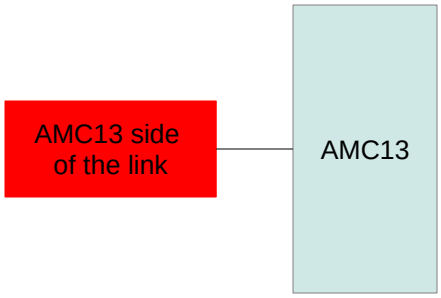


Illustration 2.13: AMC Link - AMC13

Table 2.8 shows counter registers which track the data flowing through the AMC13 side of the link. All counters are 64 bits and hence span two registers.

These counters are kept on the AMC13 side of the link, and their values are stored in registers on the Virtex 6 FPGA. These counter values are also compared against counter values sent from the AMC side of the link to verify the integrity of the link and to check for data corruption that may have occurred in transit.

All counters are 64 bits and can be read at Virtex registers {0x800 + (0x80*AMC#) + “Register”} for each (enabled) AMC (0-11).

Register	*R/W*	*Description*
0x40-41	R	Total DAQ words received by the AMC13 from the link
0x42-43	R	Single-bit errors found in TTC information from the link (not yet well defined)
0x44-45	R	Multi-bit errors found in TTC information from the link (not yet well defined)
0x46-47	R	OrN mismatches found in TTC information from the link (not yet well defined)
0x48-49	R	BcN mismatches found in TTC information from the link (not yet well defined)
0x4a-4b	R	“L1A Info” packets sent by the AMC13 which were not acknowledged by the AMC side of the link and therefore were resent
0x4c-4d	R	“Event Data” packets accepted by the AMC13 from the link
0x4e-4f	R	“Counter Info” packets received by the AMC13 from the link. This counter increments periodically, since AMC-side counter values are sent over every ~262ns.
0x50-51	R	“L1A Info Acknowledge” packets received by the AMC13 from the link. In other words, “L1A Info” packets sent from the AMC13 which have been acknowledged by the AMC side of the link.
0x52-53	R	Events received by the AMC13 from the link
0x54-55	R	Events read out from the AMC13 side of the link's event FIFO (see figure 2.10) by the Event Builder
0x56-57	R	“Event Data” packets aborted by the AMC13 side of the link due to bad data
0x58-59	R	“Counter Info” packets aborted by the AMC13 side of the link due to bad data

Table 2.8: AMC13 Side of the Link Counter Registers

General Link Counters

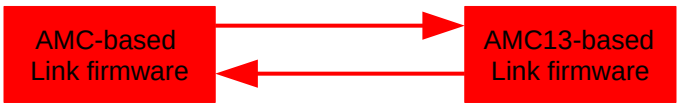


Illustration 2.14: AMC Link Connection

Table 2.9 presents counters which keep track of errors in the AMC Link, number of events aborted due to these errors, and discrepancies between event information received and its corresponding TTC information. These counters are tallied and stored on the AMC13 side of the link.

The counters listed have values for every (enabled) AMC (0-11) at registers {0x800 + (0x80*AMC#) + “Register”}.

Register	*R/W*	*Description*
0x56-57	R	“Event Info” packets aborted by the AMC13 side of the link due to bad data
0x58-59	R	“Counter Info” packets aborted by the AMC13 side of the link due to bad data
0x5a-5b	R	“Event Data” packets aborted due to the link's packet-acknowledgements buffers being full
0x5c-5d	R	“Event Data” packets aborted due to the link's event buffers being full
0x5e-5f	R	“Event Data” packets aborted due to the link's event-info buffers being full
0x60-61	R	“Event Data” packets aborted due to bad sequence number
0x62-63	R	“Event Data” packets aborted due to CRC error
0x64-65	R	“Event Data” packets aborted due to bad frame
0x66-67	R	“Event Data” packets aborted due to bad K character
0x68-69	R	Link busy time (not used)
0x6a-6b	R	EvN mismatches found between “Event Data” packets from the AMC and L1A info in the Event Builder's L1A FIFO
0x6c-6d	R	BcN mismatches found between “Event Data” packets from the AMC and L1A info in the Event Builder's L1A FIFO
0x6e-6f	R	OrN mismatches found between “Event Data” packets from the AMC and L1A info in the Event Builder's L1A FIFO

Table 2.9: General Link Counter Registers

2.2.6 Control Registers

Table 2.10 presents registers which enable the AMC Links and offset the outgoing Bc0 information. Data can only be received from the AMCs if the corresponding link is active!

Register	*Bit(s)*	*R/W*	*Description*
0x3	0-11	RW	Enable corresponding AMC Link (AMC# corresponds to bit #) data and clock
0xd SPA	0-11	RW	On the Spartan chip! Enable corresponding AMC Link (AMC# corresponds to bit #) *only* clock, no data
0x6	0-4	RW	Bc0 compensation to synchronize the orbit signals of the AMCs and AMC13 (default to 0x18)

Table 2.10: AMC Link Control Bits

2.3 Event Builder

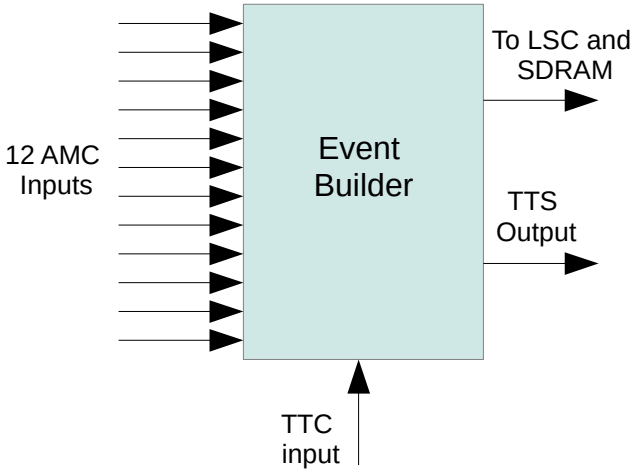


Illustration 2.15: Event Builder

2.3.1 General Functionality

The Event Builder receives event fragments from the AMC Links as well as L1A info from the TTC Receiver and builds events which meet certain format requirements (currently set by CMS HCAL, see section 1.1.1). It then sends the formatted events to the SDRAM for storage and/or the DAQLSC high-speed link for readout.

2.3.2 Buffering

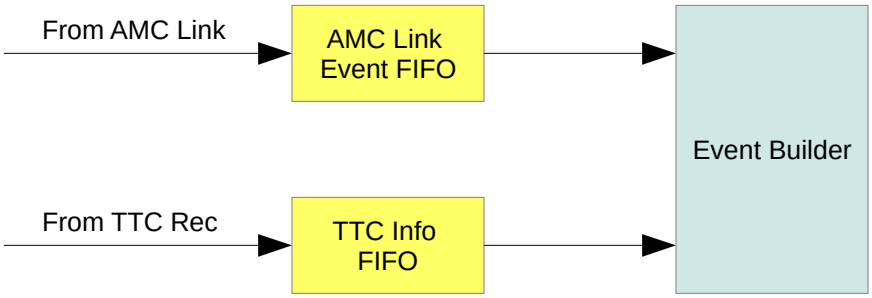


Illustration 2.16: Event Builder Buffer Readout

The Event builder output has no buffering. After building an event and sending it off, it either reads out the next trigger from the TTC Info FIFO (max 128 triggers, see section 2.2.3) and the next event fragment from each enabled AMC Link Event FIFO (max 8 event fragments per enabled AMC, see section 2.2.3), or, if these are not available, it waits for them to become so.

When the an event is ready to be built, the event builder reads one L1A from the TTC FIFO, then reads one event fragment from each enabled AMC link input (or the fake data generator, see section 3.4). If the event header passes integrity checks (CRC and basic payload verification), a header is written to the

output stream, followed by the event fragment read from each AMC, followed by an event trailer. The detailed data format is HCAL specific (see section 1.1.1).

Note that other sensible error-checking—such as EvN, BcN, and OrN matching—are carried out on the AMCs and are therefore not repeated by the Event Builder.

2.3.3 Output Counter Registers

Table 2.11 presents counter registers that monitor data flowing out of the Event Builder and into the DAQLSC and/or SDRAM monitor buffer.

Register	***R/W***	***Description***
0x52-53	R	DAQ words sent to DAQLSC
0x54-55	R	Events sent to DAQLSC
0x56-57	R	Events sent to Monitor Buffer

Table 2.11: Event Builder Output Counter Registers

2.4 SDRAM Monitor Buffer

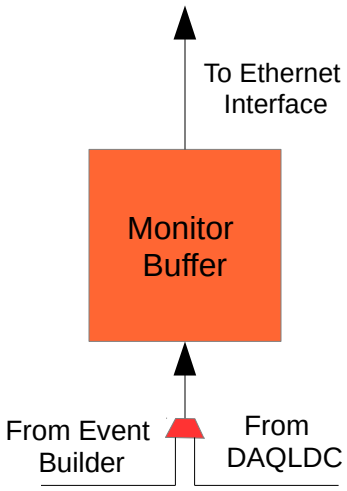


Illustration 2.17: Monitor Buffer

2.4.1 General Description

The monitor buffer is a synchronous dynamic random-access memory (SDRAM) block which receives fully formatted events from the Event Builder or DAQLDC and stores them for readout via the Ethernet Interface (see section 5.1). The SDRAM can operate in several different modes and respond to several different AMC13 conditions.

2.4.2 Buffering

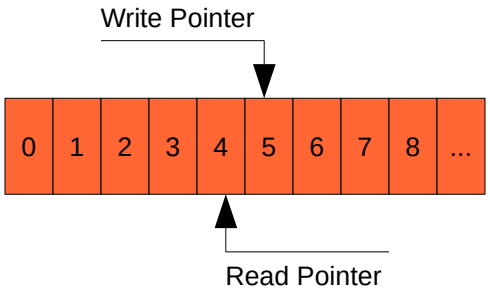


Illustration 2.18: SDRAM Structure

The RAM can hold up to 2048 (0x800) events with maximum size 64kBytes (0x3e80 32-bit words). The memory is divided into pages of this 64kB length, one event per page (no matter the event size), as shown in Illustration 2.6. The Virtex firmware allows the control of the read pointer position when the module is not in run mode, allowing the user to access any page of the buffer for manual readout. However, when the module *is* in run mode and the Event Builder is sending events for storage, the read pointer and write pointer advance sequentially, and the RAM acts like a FIFO.

2.4.3 Modes of Operation

The SDRAM can operate in one of several modes listed in Table 2.12. More detailed descriptions follow the table.

Mode	***Description***
Backpressure	If the Monitor Buffer is full, the Event Builder will stop constructing events
Memory Self Test	Write a pattern to memory for easy debugging via Ethernet readout
Mega Scaling	Save only events whose event number is a multiple of a user-set scaling factor
Error Catching	Catch an event with a CRC, EvN, OrN, or BcN error and stop writing

Table 2.12: SDRAM Modes of Operation

Default: Fill the Monitor Buffer normally, and when it becomes full, let incoming events from the Event Builder fall on the floor.

Monitor Buffer Backpressure: The Event Builder stops building events when the SDRAM becomes full. Once room becomes available in the buffer again, event building resumes. This backpressure ensures that no built events are discarded and that all events are read out sequentially.

Memory Self Test: If bit 6 of register 0x1 is set to '1', then a pseudorandom binary sequence of 64-bit words (PRBS) is written to memory; if this bit is set to '0', then sequential 32-bit words are written. After the memory has been written to (in either mode), it can be read out via the Ethernet Interface and verified. This capability is useful to check whether the SDRAM is causing data corruption.

Mega Monitor Scaling: Save only those events whose EvN is a multiple of a user-defined scale factor.

All other events are discarded. This feature is useful when dealing with a high data rate.

Error Catching: When in an error-catching mode, the RAM gets divided into four memory blocks of 128 pages (1 page contains 1 event) each, as shown in Illustration 2.19.

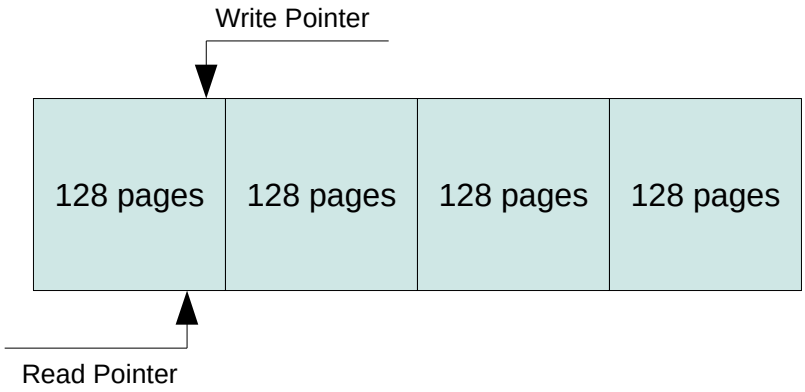


Illustration 2.19: SDRAM Error-catching Partitioning

The write pointer loops circularly through the first block, overwriting old events; the read pointer is not used when writing is in process. This cyclical writing continues indefinitely until an error "caught". Following this caught error, 64 additional events are written to the current block before the write pointer is advanced to the next 128-page section. The "defective" block now contains 64 events before the initial error and 64 events after it, and the user can investigate the circumstances that led up to and followed the initial error.

Illustration 2.20 shows a possible state of an error-catching block after completing the previously described process.

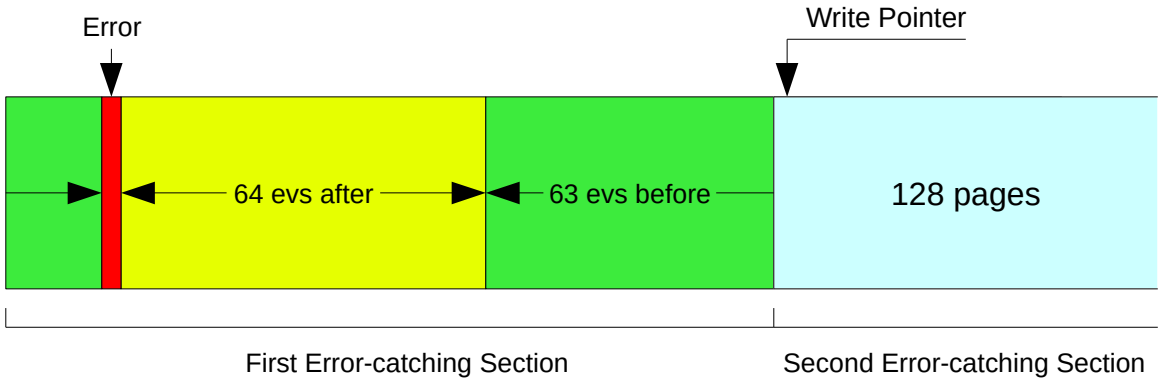


Illustration 2.20: SDRAM Memory Block Following a Caught Error

The user can catch up to four errors, one caught error for each block; after 64 events following the caught error are written, the write pointer advances to the next empty block and repeats the process. If there are no more empty blocks to write to (I.e. if all four blocks have caught an error), the SDRAM

will be marked as full and events will stop being saved to it.

If an error occurs before 64 events have been written to a block, 64 events are *still* written following the caught error, and the block's extra space is simply left empty. For example, if the 32nd event of our current block has an error, then 64 events are written following it before the write pointer advances to the next block (or stops writing), leaving 32 pages blank.

If another error occurs less than 64 events following the initial caught error, the write pointer pays no mind; it will write 64 events following the caught error, whether they are correct or not.

Mode Control Registers

Table 2.13 presents registers relevant to the various SDRAM modes.

Register	***Bit(s)***	***R/W***	***Description***
Monitor Buffer Backpressure			
0x1	14	RW	Stop building events when the Monitor Buffer becomes full
Flush/Send Data Downstream			
0x1	13	RW	If '1', flush data upon a TTC resync; if '0', send data downstream
Self Memory Test			
0x1	4	RW	Enable Memory Self Test
Mega Monitor Scaling			
0x2	23	RW	Enable Mega Monitor Scaling
0x2	19-22	RW	Set number of zeros in prescale factor, min of 5 zeros, max of 20. 0x0: 20 zeros (prescale factor 0x100000). 0xf: 5 zeros (prescale factor 0x20)
Error Catching			
0x2	17	RW	If '1', SDRAM catches events with a CRC error
0x2	16	RW	If '1', SDRAM catches events with an EvN, OrN, and/or BcN mismatch
0x2	15	RW	If '1', there are at least 128 events in the Monitor Buffer
0x2	0-7	RW	Events captured after an error event was caught. Should be 0x40 (64)
0xc	1	W	Write '1' to advance to the next 128-page block (run mode must be enabled)

Table 2.13: SDRAM Mode Control Registers

2.4.4 Status and Control Registers

Table 2.14 presents registers that monitor the state of the SDRAM, allow the user to maneuver and read the memory block, and give summary information about the RAM's contents.

Register	*Bit(s)*	*R/W*	*Description*
0x2	8-14	RW	SDRAM write pointer position
0xc	0-11	RW	SDRAM page number (read pointer position)—each page is 64kbytes/1 event. Writable only when not in run mode
0xc	0	W	Write '1' to advance one page (Run Mode must be enabled)
0xd	0-13	R	Event size on current page in 32-bit words
0xe	0-11	R	Number of unread events captured by buffer (max 2048)
0xf	NA	R	Number of HTR CRC errors in built events

Table 2.14: SDRAM Status and Control Registers

2.4.5 Readout Registers

The contents of the current page—which is determined by the position of the read pointer—can be read out of the SDRAM from registers 0x4000-7fff. When the read pointer is advanced to the next page, then this register window will contain the contents of that next event.

2.5 TTS Out

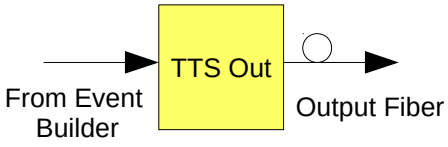


Illustration 2.21: TTS Output

2.5.1 General Description

The TTS signal is an 8-bit broadcast signal used to relay the state of the AMC13 to the CMS global trigger system. The signal's low-level protocol is exactly like that of the TTC Broadcast command (see section 2.1.1), as it is biphase 8b/10b encoded. The lower four bits of the signal are (as of right now) undefined and not used, while the upper four bits carry the TTS state command.

As shown in Illustration 2.22, the TTS signal is sent from the AMC13 through a fiber optic cable to the TTC Transceiver, which is either external to the board or the internal TTC generator (see section 3.3). This signal relays the state of the AMC13 to the TTC system, which in turn slows or stops sending triggers when appropriate.

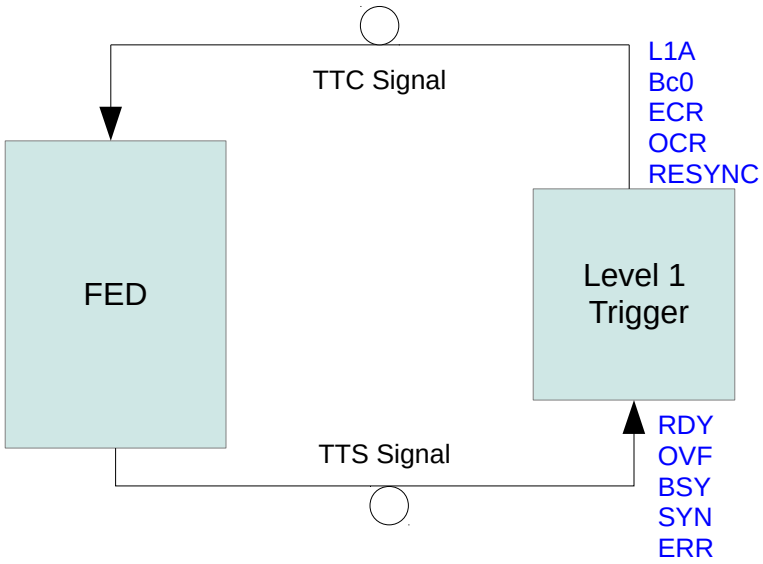


Illustration 2.22: TTS Layout

The signal is composed of four bits which are used to broadcast certain messages to the TTC system. Table 2.15 shows each signal and the effect it has on the TTC system (and therefore, the incoming triggers).

Signal	***Description***
0000	Hardware failure or broken cable
0001	Overflow Warning: causes triggers to slow down (OVF)
0010	Synchronization lost: causes the triggers to stop (SYN)
0100	AMC13 can no longer accept triggers: causes the triggers to stop (BSY)
1000	AMC13 ready to receive triggers (RDY)
1100	Any other state that prevents correct functioning (ERR)
1111	Hardware failure or broken cable
All others	Disallowed states. The AMC13 TTS will not transmit them

Table 2.15: TTS Signal Definitions

Table 2.16 shows what causes each of the five dynamic TTS states. When a new state is achieved, the signal is changed. Otherwise, the TTS state is resent every ~6 microseconds (every 256 bunch crossings) over the outgoing fiber.

Signal	*Cause*
OVF	Overflow warning. Adjacent to the RDY and SYN states. There must be more than 95 L1As are in the Event Builder's input L1A buffer to reach OVF from RDY. Once in OVF, the buffer must contain less than 64 L1As to return to RDY.
SYN	Synchronization lost. Once the Event Builder's input L1A buffer drops a trigger while full (256 triggers), the TTS state changes from OVF to SYN and stays there until a general AMC13 reset is carried out.
BSY	Busy. The Event Builder is busy and cannot accept any more L1As.
RDY	Ready. The Event Builder is ready to accept triggers. Less than 96 triggers in the Event Builder L1A input buffer.
ERR	Error. Some error other than the four listed above

Table 2.16: TTS State Descriptions

2.5.2 Status Registers

Table 2.17 presents counters that display the amount of time (counted at 150MHz) spent in each of the TTS states. You can learn the current TTS state of the AMC13 by noting which counter is currently incrementing.

Register	*R/W*	*Description*
0x4a-4b	R	Time spent in Ready Mode (RDY)
0x4c-4d	R	Time spent in Busy Mode (BSY)
0x4e-4f	R	Time spent in Synchronization Lost Mode (SYN)
0x50-51	R	Time spent in Overflow Warning Mode (OVF)

Table 2.17: TTS Status Registers

2.5.3 Test Registers

The TTS signal can be hijacked for testing. Table 2.18 presents registers that allow the AMC13 to ignore its TTS state and instead enter “TTS test mode”. When in this mode, the user can control the TTS state explicitly in a way completely transparent to the TTC system.

Register	*Bit(s)*	*R/W*	*Description*
0x1	12	RW	If '1', TTS output corresponds to register 0x19 instead of TTS state (for testing)
0x19	0-3	RW	TTS test output. Only effective when register 0x1 bit 12 is set to '1'

Table 2.18: TTS Test Registers

2.6 DAQLSC

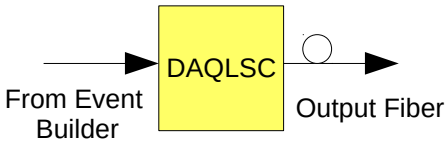


Illustration 2.23: DAQLSC

2.6.1 About the DAQ Link

After the AMC13 has built an event and packaged it for readout, the packet is sent to the SDRAM for storage and/or to the DAQ Link for readout over an optical fiber (or both). When sent over the DAQ Link, the fully-formatted event is ultimately accepted by the DAQ receiver, which is responsible for delivering the event to the CMS high-level trigger.

The following paragraph describes some facts important to DAQ Link latency and backpressure. The DAQ Link speed is 5Gb/s using 8b/10b encoding, so effectively 4Gb/s of real data, or 500MB/s; therefore, the maximum sustained write clock to the link is 62.5 MHz (16ns per 64-bit word). The DAQ Link input buffer has four 4kB pages, one page per event. If we assume a 6ns transmission time per meter of optical fiber, we have a 12ns/m round-trip latency due to fiber travel (we can assume that the acknowledge send and acknowledge receiver logic time is negligible). Hence, we can write only 3 words to the DAQ Link buffer (24 Bytes, only half the size of an event header!) per every 4 meters of fiber. If we were using 40 meters of fiber (a more realistic scenario), we could then write 240Bytes of data to the DAQ Link buffer while waiting for an event to transmit over the fiber, still very far from an event size of any consequence (still much smaller than a typical event)! Therefore, for typically-sized AMC13 events, the link latency will not be responsible for LSC backpressure. The only exceptionally special case would arise if while a full-sized packet were being transmitted, three very small packets were written to the DAQ Link buffer before the receiver's acknowledge signal arrives. In all practical circumstances, however, this type of behavior will not be a factor.

The CMS Central DAQ Group developed the AMC13 DAQ-Link firmware; for any questions regarding it, contact Christoph Schwick at christoph.schwick@cern.ch.

2.6.2 General Description

The DAQLSC is the transmitter end of the high-speed DAQ Link. It receives fully formatted events from the Event Builder and sends them via an SFP connection to an external DAQ receiver.

The following is a brief overview of the LSC's workings. As soon as the DAQLSC receives an event from the AMC13 logic, it is sent over the link's optical fiber. Upon receiving the transmitted event, the DAQ receiver sends an acknowledge signal if there is no transmission error. Once this acknowledge signal is received by the DAQLSC, the next packet is sent over. If an acknowledgement is not received by the LSC from the DAQ receiver (due to a transmission error), the packet is sent again after a configurable timeout value (set by the DAQ Link firmware).

2.6.3 Buffering

Because of the low latency of the DAQ Link (see section 2.6.1), the LSC has a small buffer of 16kB

(four pages, one packet per page) which is used only to to keep packets long enough for the AMC13 to receive an acknowledge signal from the DAQ receiver for the previously-sent packet. Once that acknowledgment is received, the LSC sends the next packet over the fiber connection.

2.6.4 Initialization

The initialization of the DAQ Link is done almost entirely from the receiver-end of the connection. That said, the DAQLSC *does* need to be enabled using the register shown in Table 2.19.

Register	***Bit***	***R/W***	***Description***
0x1	1	RW	'1' indicates the AMC13 is ready to send DAQ data through the DAQLSC link

Table 2.19: DAQLSC Initialization Bits

2.6.5 Status Registers

Table 2.20 presents registers that are responsible for monitoring the condition of the DAQLSC output link.

Registers	***Bit(s)***	***R/W***	***Description***
0x0	2	W	Write '1' to reset the LSC, except not the multi-gigabit transceiver
0x0	3	W	Write '1' to reset the LSC plus the multi-gigabit transceiver
0x0	0	R	LSC link not connected or not active
0x0	1	R	Input FIFO to LSC is almost full
0x16-17	NA	R	LSC output status

Table 2.20: DAQLSC Status Registers

2.6.6 Counter Registers

Table 2.21 presents counter registers that are responsible for monitoring the activity of the LSC link while checking for errors.

Register	***R/W***	***Description***
0x10	R	Packet acknowledgements received from the receiver end
0x11	R	Packets sent out
0x12	R	Retransmission requests received from the receiver
0x13	R	Events sent out
0x14	R	Words sent out
0x15	R	Sync lost due to FIFO overflow time (150MHz)
0x16-17	R	Status output

Table 2.21: DAQLSC Counter Registers

3 Additional Features

The AMC13 has some additional features which are useful for testing purposes, as well as for non-CMS users. The following section discuss these features and their potential applications.

3.1 DAQLDC

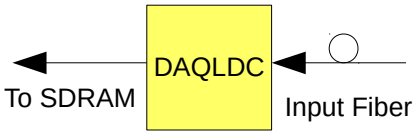


Illustration 3.1: DAQLDC

3.1.1 General Description

The DAQLDC is the receiver end of the DAQ Link, receiving formatted events from the AMC13 and sending them to the Central DAQ for unpacking. As an optional feature, the AMC13 can act as a DAQLDC receiver, taking data sent from another AMC13 (or from the same AMC13's DAQLSC link) and storing the data in its SDRAM for readout via the Ethernet Interface.

This feature is particularly useful when no DAQ receiver is available and the AMC13's DAQLSC functionality needs to be tested.

3.1.2 Initialization

The following procedure enables the DAQ Link, and can *only be carried out from the receiver end*.

1. Write 0x80010006 to register 0x80
2. Write 0x00000000 to register 0x81
3. Read back register 0x80. If bit 0 is set to '1', the DAQ sender has successfully responded to the initialization.

NOTE: to complete the DAQ Link initialization, you must enable the DAQLSC by writing '1' to bit 1 of register 0x1 on the sending AMC13 (see section 2.6.4).

3.1.3 Status and Control Registers

Registers shown in Table 3.1 are used to reset the LDC link, direct received DAQLDC events, and monitor the LDC's status.

Register	*Bit(s)*	*R/W*	*Description*
0x0	4	W	Reset LDC, except not the multi-gigabit transceiver
0x0	5	W	Reset LDC and transceiver
0x80-82	NA	RW	Used to initialize the DAQLDC link
0x83	3	RW	Send events received via the DAQLDC to the SDRAM
0x83	2	R	Sync lost due to FIFO overflow time (150MHz)
0x83	1	R	Link enable complete
0x83	0	R	Link initialization complete
0x84-85	NA	R	Status port register
0x86	NA	R	Packet number (increments sequentially)
0x87	NA	R	Command number (increments sequentially)

Table 3.1: DAQ LDC Status and Control Registers

3.1.4 Counter Registers

Table 3.2 presents counters that keep track of the AMC13 event data which flows into the DAQ Receiver through the DAQLDC. Note that all DAQ words are 64 bits!

Register	*R/W*	*Description*
0x90	R	Words Received
0x91	R	Event tags received (should be two per event)
0x92	R	Events Received
0x93	R	Words sent to memory
0x94	R	Event tags sent to memory (should be two per event)
0x95	R	Events sent to memory
0x96	R	Events written to the FIFO when full (FIFO write errors)
0x97	R	CRC Errors
0x98	R	Sync Lost Time due to FIFO full
0x99	R	Packets Acknowledged
0x9a	R	Valid Packets Seen
0x9b	R	Total Packets Seen
0x9c	R	Bad Packets Seen
0x9d	R	Time since the FIFO could no longer accept events (due to backpressure)
0x9e	R	Events with bad tags
0x9f	R	Data rate in bytes per second

Table 3.2: DAQ LDC Counter Registers

3.2 Internal L1A

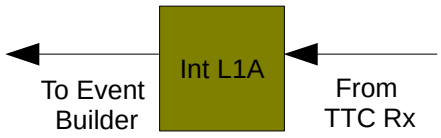


Illustration 3.2: Internal L1A Feature

3.2.1 General Description

Given an input TTC clock signal, the AMC13 has an optional feature to generate its own triggers. The AMC13 can generate many different types of triggers, including periodic triggers (based on Bc0 or BcN), random triggers, and individual bursts of triggers. When this feature is enabled, the TTS output (ready, busy, etc) from the Event Builder controls the internal L1A generator, and L1A functionality becomes completely internal.

This feature is useful when there is no external trigger generator available or when performing simple tests. Note that *either* an externally-generated TTC clock *or* an internally-generated TTC clock (see

section 3.3) can be used with internally-generated L1As.

3.2.2 Setup

Table 3.3 presents registers that configure the local L1A generator with customizable settings and features. All of the following options are based on the received TTC clock signal.

Register	*Bit(s)*	*R/W*	*Description*
0x1c	30-31	RW	**L1A Type** '00': L1A per orbit '10': L1A per bunch crossing '11': random L1A
0x1c	28-29	RW	**Trigger Rules Enforced** '00': Rules 1-4 '01': Rules 1-3 '10': Rules 1-2 '11': Rule 1
0x1c	16-27	RW	**Number of L1As generated in a burst**
0x1c	0-15	RW	**L1A Rate** If in orbit mode, L1A every N+1 orbits at BCN 0x1f4 If in bunch crossing mode, L1A every N+1 bunch crossing If in random mode, L1A every (2*N)/s

Table 3.3: Local Trigger Configuration Bits

The standard CMS trigger rules are as follows:

1. No more than 1 Level 1 Accept per 75 ns (minimum 2 BX between L1A)
2. No more than 2 Level 1 Accepts per 625 ns (25 BX)
3. No more than 3 Level 1 Accepts per 2.5 μs (100 BX)
4. No more than 4 Level 1 Accepts per 6 μs (240 BX)

NOTE: The internally-generated triggers do *not* respect an orbit gap.

3.2.3 Control Registers

Once register 0x1c is set, the bits presented in Table 3.4 are used to send the local triggers.

Register	*Bit(s)*	*R/W*	*Description*
0x0	10	W	**Burst of Local L1As** If bit reads '0', writing '1' sends a burst of local L1As specified by 0x1c If bit reads '1', writing '1' resets bit to '0', disabling continuous local L1As
0x0	26	W	'1' enables continuous local L1As

Table 3.4: Local Trigger Control Bits

3.2.4 Status Registers

Table 3.5 presents a register that monitors whether local triggers are enabled.

Register	*Bit(s)*	*R/W*	*Description*
0x0	10	R	'1': continuous local L1As are enabled (being sent) at rate specified by 0x1c

Table 3.5: Local Trigger Status Bits

3.3 Internal TTC Transceiver

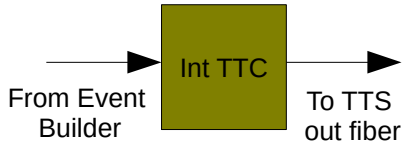


Illustration 3.3: TTC Tx Feature

3.3.1 General Description

If the internal L1A feature is being used, the AMC13 has an additional feature that transforms the TTS SFP output into an internally generated, 40MHz (error tolerance 50 parts per million) pseudo TTC clock output which can be looped back via an optical fiber to the TTC Receiver SFP input and used as the effective TTC clock for internally generated L1As. This signal simply uses the Virtex chip's crystal oscillator to build an orbit signal with a bunch structure.

This feature is useful if there is no external clock signal available for the AMC13, or if the user wants the module to work without any external dependencies.

3.3.2 Control Registers

Table 3.6 shows the bits for enabling the internal TTC signal as well as the bits for sending internally-generated TTC broadcast commands.

Though the internally generated clock signal does not have all of the bells and whistles of an LHC-based TTC generator, it does carry Orbit and Bunch-crossing information; therefore, the internal TTC feature allows for the sending of two TTC broadcast commands (see section 2.1.1 for more information on TTC broadcast commands): Orbit reset and Bunch-count reset.

Register	*Bit(s)*	*R/W*	*Description*
0x0	11	W	Write '1' to send an event-number reset through TTC
0x0	12	W	Write '1' to send an orbit-number reset through TTC
0x1	8	RW	'1' transforms the TTS output into an internally-generated TTC output signal

Table 3.6: Internal TTC Control Bits

3.4 Fake Data

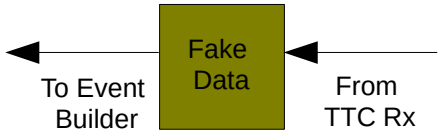


Illustration 3.4: Fake Data Generator

3.4.1 General Description

The AMC13 has the ability to ignore the AMC Links (and hence all AMCs in the crate) and independently generate data fragments which follow the µHTR data format (see section 1.1.1). The fake data generator acts just like the AMC Links and should be treated in the same way (see section *2.2* for more detail regarding the AMC Links).

This feature is useful when no AMCs are available or when you want to test more AMC Links than the available number of AMCs can occupy.

3.4.2 Control Registers

Table 3.7 presents registers that enable fake data generation, control the size of the fake-event payloads, and enable the fake ports.

Register	***Bit(s)***	***R/W***	***Description***
0x1	7	RW	Generate fake data upon receiving an L1A
0x18	0-10	RW	Payload size (in 16-bit words) of fake µHTR data (0-2047)
0x3	0-11	RW	When bit 7 of 0x1 set to '1', write '1' to enable corresponding ***fake*** AMC Link

Table 3.7: Fake Data Control Registers

4 Flash

Firmware management for the AMC13 is handled by a Numonyx M25P128 Serial Flash Memory Chip[H] on Tongue 2, specifically the Spartan FPGA. The AMC13 firmware is compiled into Map Cycle Set (MCS) files[I] which can be programmed to the Flash Memory chip by the "AMC13Tool" software executable.

The following sections will discuss the organization of the flash memory as well as the procedure for updating the AMC13 firmware.

4.1 Layout

The flash memory is divided into four sectors, as shown in Illustration 4.1. Any of these sections can be erased and reprogrammed.

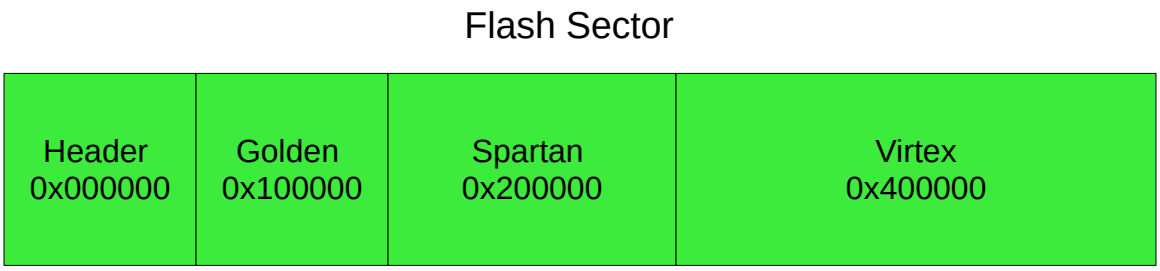


Illustration 4.1: Flash Layout

Header: An 80byte header for the flash section which should never be overwritten. Below is the Flash Header format as documented on the Xilinx website[J].

```
FFFFFFFF   // DUMMYWORD, DUMMYWORD
AA995566   // SYNCWORD
31E1FFFF
32610000   // 32A1=GENERAL1 0x0000=address[15:0] MultiBoot image location
32810340   // 3281=GENERAL2 0x03=SPIx1 read command, 0x40=address[23:16]
           // MultiBoot image address
32A10000   // 32A1=GENERAL3 0x0000=address[15:0] Golden image address
           // location
32C10301   // 32C1=GENERAL4 0x03=SPIx1 read command, 0x01=address[23:16] of
           // Golden image address
32E10000
30A10000
33012100
3201001F
30A1000E
20002000   // NOOP, NOOP
20002000   // NOOP, NOOP
```

Golden: This is the backup flash firmware for the Spartan FPGA, marked as version 0xffff. This flash

sector will be uploaded to T2 upon booting the module if, for some reason, the Spartan chip does not have functional firmware at power-up. This “safeboot” functionality makes the module more robust against flash corruptions which cannot be fixed by the AMC13 software; as long as the Ethernet Interface can talk to the Spartan chip, the firmware can be reprogrammed without JTAG.

Spartan: This section contains the current version of the T2 Spartan firmware.

Virtex: This section contains the current version of the T1 Virtex firmware.

4.2 Programming the Flash

The flash sector can be programmed using the AMC13Tool C++ command line tool (see section 5.2) using a specific software recipe[K].

4.3 Flash Access and Control

Table 4.1 shows the three registers on the Spartan chip which are used to interact with the flash memory. It is helpful to notice that though these events are written to (via the Ethernet Interface) Little Endian, the registers themselves are written to the flash Big Endian. Therefore, for the programming of the flash, bits 31-24 is Byte 0, 23-16 is Byte 1, etc.

Register	***R/W***	***Name***	***Description***
0x1	RW	FLASH_CMD	Sends a specified flash command to the flash module
0x1000-107f	RW	FLASH_WBUF	Flash write buffer. The flash command is to be written to address 0x1000. For a page write command, attach the write data starting at address 0x1001.
0x1080-10ff	RW	FLASH_RBUF	Flash read buffer. The flash status is stored in address 0x180. For a page-read command, read data is stored starting at address 0x1081.

Table 4.1: Flash Registers

Tables 4.2 and 4.3 show the commands that interact with the flash. To carry out each action, the correct value must be written to 0x1000 (the first address of the Write Buffer), and then the correct value must be written to 0x1.

Flash Commands for bits 24-31 of 'FLASH_WBUF' at 0x1000		
Value	**Action**	**Bits 0-23 of 0x1000**
0x06	Enable Flash Write	0x000000
0x05	Wait for Flash Write to finish	0x000000
0xd8	Erase Flash Sector	Starting address of the sector to be erased
0x9f	Get Flash Identification	0x000000
0x02	Write from 'FLASH_WBUF' to Flash Memory	Starting address of the sector to be written to
0x0b	Read from Flash Memory to 'FLASH_RBUF'	Starting address of the sector to be read from

Table 4.2: FLASH_WBUF Commands

Flash Commands for 'FLASH_CMD' at 0x1	
Value	**Action**
0	Enable Flash Write
1	Wait for Flash Write to finish
3	Erase Flash Sector
3	Get Flash Identification
3+n	Write n bytes from 'FLASH_WBUF' to Flash Memory
4+n	Read n bytes from 'FLASH_RBUF' to Flash Memory

Table 4.3: FLASH_CMD Commands

For a more comprehensive look at how the flash memory is erased, programmed, and verified, refer to the source code for the AMC13 Python Software's flash-programming module[L].

4.4 Reconfigure FPGAs from Flash

Table 4.4 presents Spartan registers which will reconfigure T1 and T2 from flash memory.

Register	***Bit***	***R/W***	***Description***
0x0	4	RW	Reconfigure Virtex FPGA from the Virtex flash sector
0x0	8	RW	Reconfigure both the Virtex FPGA from the Virtex flash sector and the Spartan FPGA from the Spartan flash sector

Table 4.4: FPGA Reconfiguration Bits

5 The Ethernet Interface and Software

5.1 Ethernet Interface

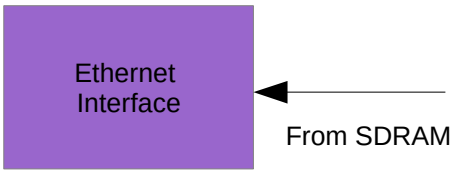


Illustration 5.1: Ethernet Interface

The ethernet connection to the μTCA crate is through the commercial MCH module, which gives the user access to the AMC13 and AMCs via the backplane fabric connections. The current communication interface between the AMC13 and an outside computer is the IPBus protocol, developed by the Code Archive for CERN Trigger Upgrades (CACTUS) group[M].

IPBus in particular has a set of basic operational capabilities, and those used by the AMC13 are shown in Table 5.1. All AMC13 software functions are based off of these low-level functions.

Function	***Description***
read( )	Read a register
readBlock( )	Read a block of registers in a single Ethernet transaction
write( )	Write to a register
writeBlock( )	Write to a block of registers in a single Ethernet transaction

Table 5.1: AMC13 CPP Class Functions

Though IPBus does offer other functionalities, they are not used within the HCAL μTCA framework.

5.2 AMC13 Software

The AMC13 has two main software packages, one written in C++ and one written in Python.

The C++ software was developed within the HCAL xDAQ framework and contains both a command-line tool and online software support for CMS DAQ. However, the C++ software has recently been migrated to the μHAL IPBus base software support package which allows it to operate outside of HCAL. Releases of these stand-alone C++ packages can be found at http://bucms.bu.edu/twiki/bin/view/BUCMSPublic/AMC13CppSoftware.

The Python software is much simpler and more limited in functionality than the C++ software but is less prone to bugs and dependency issues, making it an ideal backup to the C++ releases. AMC13 Python tarballs can be found at http://bucms.bu.edu/twiki/bin/view/BUCMSPublic/AMC13PythonSoftware.

Detailed instructions regarding software functionality, command descriptions, and common procedures can be found at http://amc13.info under the AMC13CppSoftware subsection.

A ***µTCA System within CMS HCAL:***

http://ohm.bu.edu/~hazen/CMS/AMC13/The%20CMS%20uTCA%20Crate%20v0.9_esh.pdf

B ***HCAL Upgrade Technical Design Report. Back-end specifications found at pages 115-135:***

http://cds.cern.ch/record/1481837/files/CMS-TDR-010.pdf

C ***HCAL Upgrade Data Format:***

http://ohm.bu.edu/~hazen/CMS/SLHC/HcalUpgradeDataFormat_v1_2_2.pdf

D ***AMC13 Schematics:***

http://ohm.bu.edu/~hazen/CMS/AMC13/Files/2011-06-01/

AMC13 Short Specification:

http://ohm.bu.edu/~hazen/CMS/AMC13/AMC13_Short_Spec_05Oct2012.pdf

E ***MMC – AMC13 Interface:***

http://ohm.bu.edu/~chill90/op_specs/UW_MMC_FPGA_Config_Interface_2013-01-31.pdf

F ***TTC website:***

http://ttc.web.cern.ch/TTC/intro.html

G ***8B/10b Encoding Wikipedia Page:***

http://en.wikipedia.org/wiki/8b/10b_encoding

H ***Data Sheet for Flash Memory Chip:***

http://ohm.bu.edu/~chill90/amc13_flash/M25P128.pdf

I ***Xilinx Description of MCS File Format:***

http://www.xilinx.com/support/answers/476.htm

AMC13 MCS Firmware Files:

http://ohm.bu.edu/~hazen/test/firmware.cgi

J ***Spartan Chip MultiBoot Design:***

http://www.xilinx.com/support/documentation/boards_and_kits/xtp059.pdf

K ***Procedure to update AMC13 firmware:***

http://bucms.bu.edu/twiki/bin/view/BUCMSPublic/AMC13ToolRecipes#UpdateFirmware

L ***Python Functions for AMC13 Flash Actions:***

http://ohm.bu.edu/~chill90/amc13_flash/flashcommands.py

M ***CACTUS Website:***

http://cactus.hepforge.org/index.php