Table of Contents

U3/U4 Firmware Design Overview

The U3/U4 firmware collects and assembles fragments that have been tagged for spy buffer readout by U1/U2. U1/U2 checks each received fragment's Level 1 (L1) ID against the bit pattern set by the user. There are separate pattern checks for determining whether the packet is sent to U3 or U4. If the bit pattern matches, then the fragment data is duplicated with the copy sent to U3 and/or U4 for assembly. Both U3 and U4 have 8 SerDes connections with U1/U2, one allocated for each SSB.

U3/U4 implements independent receiver pipelines for each fragment transfer lane from U1/U2. The start of this pipeline is the Tagged Event Receiver (TER). The TER performs format checking and stores fragment information required for fragment assembly. The TER transfers the unmodified fragments data to a holding FIFO. These FIFOs are then processed by the Fragment Assemble Logic (FAL).

The FAL continuously monitors the input FIFOs for fragment data, and uses the extracted L1 ID to sort them into DDR memory. The DDR is divided into 16 working buffers. During assembly, fragments with new L1's are allocated to the next available buffer. Additional fragments with the same L1 ID are then stored sequentially in these buffers. There is no ordering of fragments within an assembly buffer. If a new L1 ID is encountered and no assembly buffers are available, the oldest working buffer will be forced to read out. Otherwise, buffers are only transferred once the expected SSB channels (as set by the user) have reported fragments.

The other function of the FAL is that is appends fragment headers to the fragment as it is sorted into the buffers. It also stores information in a separate RAM that will be later be used by the Data Packet Generator (DPG) to create the overall fragment assembly, UDP, and IP headers.

Once all fragments have been prepended with a fragment header and stored in the assembly buffer, the DPG is responsible for generating the assembly header and footer and transferring the entire assembly to the Ethernet interface logic for transmission to blades over the ATCA backplanes. Both U3 and U4 have two 10G Ethernet interface, for a total of four per FLIC. Each interface has its own DPG. The currently supported fragment transmission logic supports sending all data to either Ethernet interface, or dividing it such that even buffers are sent to one and odd buffers sent to the other. The DPG also handles UDP fragmentation protocol.

Currently no slow control occurs on the backplane interfaces.

Firmware Components Overview

GTX Transceiver

FIFOs

The U3/U4 FPGA design uses a slightly modified GTX transceiver when compared to the U1/U2 implementation. The primary difference is that the RX FIFO, uses an "Event" FIFO. However, this is only done to provide, a FIFO of configurable depth, so that both the TX and RX FIFO sizes are independently configurable via a generic value passed in at the top level of the design. This allows the unused "GTX112" to use minimal RAM resources, without requiring a unique component declaration. Also the RX FIFOs in GTX 113 and 114 are only 16 words deep as they only serve as a clock domain crossing between the individual link's receive clocks at 125MHz and the faster 200MHz clock used for all Tagged Event Receivers.

Logic

Logically, the U3/U4 FPGA is identical to the generic GTX instantiated in U1/U2. Each GTX runs four independent links, and each supports the same diagnostic modes as U1/U2. GTX113 connects to U2 and GTX114 connects to U1. GTX112 cross connects U3 and U4, but is unused. Receive ILAs are provided on GTX113 and GTX114. However, only GTX114 has a transmit ILA.

Tagged Event Reception Logic

A tagged event or fragment is one that was selected by U1/U2 for readout. They are sent unmodified from U1/U2 to U3/U4, with the exception that an over length is tagged onto the end for data checking purposes. U3/U4 implements a similar data reception state machine as U1/U2's "Core Crate Receiver." However, there are many simplifications, as well as some added facilities related to the processing of fragments for assembly.

The tagged event receiver logic is structured into four main components:

1. The Tagged Event Receiver state machine
2. The Event Description FIFO
3. The Fragment Data Width Adapter
4. The Fragment Data FIFO

Tagged Event Receiver Logic

The Tagged Event Receiver verifies the format of incoming fragments in the same way as the Core Crate Receiver does for U1/U2. However, it also pulls out the L1 ID and the total fragment length for input into the Event Description FIFO. Another difference form the Core Crate Receiver, is that it will never assert flow control to U1/U2. Fragments are always received from U1/U2, and the GTX FIFO is constantly read. Fragments are initially stored in the Fragment Data FIFO. If this FIFO fills, new event fragment are simply dropped, on the effected channel, until space becomes available. This FIFO is capable of storing several fragments of the maximal length as set by U1/U2. Fragments are dropped if the available FIFO space drops below the space required to store a maximum length fragment. Once a

complete event has been received, this machine notifies the Fragment Assembly logic via the "Event" FIFO controller of the Fragment Data FIFO.

Event Description FIFO

The Event Description FIFO holds fragment information required for assembly. This includes the L1 ID, the total fragment length, and any error status reported by the Tagged Event Receiver.

Fragment Data Width Adapter

The width of the data is changed from 16-bits to 64-bit prior to being written to the Fragment Data FIFO. This is done to prevent throughput bottlenecks in the fragment assembly logic (into which all data must be funneled). 64-bits is also the required width for interfacing to both the DDR Interface and 10G Ethernet Interface IP Cores.

Fragment Data FIFO

Each receive pipeline implements a reception buffer of 65,536kB. This is also serves a domain crossing FIFO in the final stage of throughput expansion. While data is written in 64-bit words at 200Mhz, it is read in 64-bit words @ 333MHz. 333MHz is the processing clock used for fragment assembly. This frequency allows for full utilization of the DDR bandwidth after accounting for the required processing delays. There is also a large FIFO between the Fragment Assembly Logic and the DDR for additional burst handling capability. The net fragment assembly throughput approaches 21Gbps for large fragments, but falls off to just below 16Gbps for zero track fragments.

There are only two reasons these Fragment Data FIFOs will contain more than 2 events. The first is if the continuous through put exceeds the DDR bandwidth, and the second is if there is a continuous steam of single or zero track fragments on 7 or more channels, in which case the assembly logic will not keep pace with the incoming data. However, in this state it is still operating faster than the DDR can sustain.

Fragment Assembly

The Fragment Assembly Logic consists of four main components:

1. The Fragment Data Multiplexer
2. The Fragment Data Table
3. The Fragment Assembly State Machine
4. The Fragment Data Table Management State Machine

The fragment assembly is managed by the two state machines. The Fragment Assembly State Machine controls the Fragment Data Multiplexer and the Fragment Data Table Management State Machine. It does not directly interact with the Fragment Data Table. Instead it routs all interactions, as commands, to the Fragment Data Table Management State Machine.

Fragment Data Multiplexer

The Fragment Data Multiplexer is responsible for switching the input to the Fragment Assembly Logic between the 8 tagged event channels. This Multiplexer switches the 64-bit fragment data buses,

the event description FIFO outputs, and all FIFO status signals of the 8 independent Fragment Receivers. This occurs at 333MHz, and requires a multi stage pipelined fan-in process. This also incorporates a pipelined fan-out of the FIFO control signals. While simple in design its operation is fundamental to the operation of the assembly logic. Due to pipelining of the data, status, and control signal, careful consideration must be made to latencies in the design of the Fragment Assembly State Machine. For example, when a transfer of data occurs from one of the Fragment Data FIFO to the DDR Write FIFO, the operation is not interruptible. This is achieved by verifying that sufficient space is available in the DDR Write FIFO prior to starting transfer, similar to the way the Fragment Receiver State Machine verifies available space in the Fragment Data FIFO prior to transfer.

Fragment Data Table

The Fragment Data Table is the heart of the design. All fragment information required for assembly and transfer status information is stored in this segmented internal RAM. The table is constructed as a RAM with 1 write port and 4 read ports. The write port and two read ports are used by the Fragment Data Table Management State Machine to manage the table. The other two read ports are used by the two Data Packet Generators.

Fragment Data Table Management State Machine

The Fragment Data Table Management State Machine (FDTMSM) handles all interactions with the Fragment Data Table. It also calculates DDR memory address offsets for placement of new fragments into the assembly regions. It defines several command or operations that the Fragment Data Assembly State Machine uses to interact with the Fragment Data Table.

For each operation the FDTMSM responds with an operation complete and operation success status. If success signal is set the operation completed normally, otherwise the operation was unsuccessful. The definition of success is dependent on the operation being performed. The operations are:

1. New – Attempt to allocate a new assembly region with the current L1 ID. If the operation is successful it will automatically record the first fragments information into the table. This will fail if no assembly regions are available. If the table is full, the New operation will automatically invoke a clean operation to free any assembles regions that have been completely processed, including transmission to the blade.
2. Scan – Search for an incomplete assembly matching the current L1 ID. This will only return the success status if an existing assembly region with less than 8 fragments is found. A successful Scan does not update the table, but it does change to the current working region for future (append) operations.
3. Append – Appends the current fragment to currently selected assembly. This will always be successful as long as the assembly has not been marked as complete. After appending the fragment to the assembly, it checks whether the assembly is complete by comparing the enabled channels to the channels that have been recorded in this region.
4. Force – Forces the oldest assembly to be marked for readout. This operation is requested by the Fragment Data Assembly State Machine in the event that a New operation fails. This

forces incomplete fragments to read out in the event that one or more channels never report a fragment.

5. Clean – Performed automatically when a New operation is executed and no free regions are available. Due to pipelining, it is more efficient to free assembly regions in bursts, when space is required rather than one at a time when assemblies are sent. The Clean operation is never commanded by the Fragment Data Assembly State Machine.

Once an assembly has been marked complete, due to either all eight fragment slots being used or the all expected fragments being processed, the Data Packet Generator will automatically transfer the data to the Ethernet Interfaces. Assemblies are sent as they are completed, not in order of L1 ID. Once sent, the Data Packet Generator will signal the FDTMSM, which then marks the region as sent so that they may be freed during a Clean operation. This may occur even, while other operations are in progress.

Fragment Data Assembly State Machine

The purpose of the Fragment Data Assembly State Machine (FDASM) is to control the transfer of event fragment data from the Fragment Data FIFOs (via the Fragment Data Multiplexer) into the appropriate assembly regions within the DDR address space.

The state machine continuously scans the Event Description FIFOs. When a channel receives an event fragment, the FDASM proceeds through a series of interactions with FDTMSM.

First the FDASM commands a Scan operation. If it fails, a New operation is then commanded from the FDTMSM. Otherwise, if the Scan operation was successful, an Append operation is requested. As long as the Scan/New is successful, it proceeds to transfer the fragment to the DDR after writing the Fragment Header. However if the New operation is unsuccessful, it will command a Force of the last unsent assembly. It then waits for a successful New operation before continuing. Fragment data is initially transferred into the Write Burst Master's data FIFO.

After transferring the fragment, assuming the assembly is still incomplete, it proceeds to scan the other channels for the same L1 ID. This continues until the either a match is found or, after checking all other channels at least once, the Write Burst Master's data FIFO reaches the almost empty data level. If a match is found the new fragment is simply appended. Otherwise, it reverts to looking for any event, as it is more important to maximize the DDR bandwidth than to minimize the number of Fragment Data Table operations. The FDASM will immediately being looking for and available fragment if the last assembly was marked as complete by the FDTMSM.

AXI Bus interface

All access to the DDR is performed over a Xilinx AXI Crossbar IP. The cross bar implements 2 write channels and 2 read channels. The Write and Read channels interface to Xilinx AXI Burst Masters cores via AXI to AXIS converters. Refer to Xilinx documentation for more detail.

UDP Packet Generator

Once a fragment assembly has been marked for readout, the UDP packet generation logic will Read the data from the DDR and generate the UDP packets. UDP Fragmentation protocol is used send assemblies that are greater than 1500 bytes. Packets are transferred to the XAUI core. Once all assembly data has been transferred, the assembly is marked for cleanup by the FDTMSM.

Ethernet XAUI and 10G MAC

U3/U4 implements standard Xilinx XAUI cores, for interfacing to the 10G Ethernet interfaces. The XAUI interface to the 10G MAC core, which is also implemented as standard Xilinx Core. Refer to Xilinx documentation for more detail.

U3/U4 Control Registers

DDR_CLOCK_CONTROL_REG – Address: 0x0000

Bit 15:3 – Reserved.

Bit 2:0 – Sets the ddr clock speed. Should, not be changed. Defaults to 200MHz.(0x0004).

GENERAL_CTL_REG – Address 0x0002

Bit 15 – Selects between static and pulsed reset modes for frame gen and frame check for GTX112, GTX113, and GTX114.

Bit 14:9 – Reserved.

Bit 8 – Frame Generator reset for GTX114.

1 = Reset

0 = Run

Bit 7– Frame Checker reset for GTX114.

1 = Reset

0 = Run

Bit 6 – Frame Generator reset for GTX113.

1 = Reset

0 = Run

Bit 5 – Frame Checker reset for GTX113.

1 = Reset

0 = Run

Bit 4 – Reserved.

Bit 3 – Frame Generator reset for GTX112.

1 = Reset

0 = Run

Bit 2 – Frame Checker reset for GTX112.

1 = Reset

0 = Run

Bit 1:0 – Reserved.

GTX112_CTL_REG – Address 0x0003

Bit 15:8 – Reserved

Bit 7:4 - Debug use only. For each link, if set to '1', timing tags may be issued.

Bit 3:0 - Debug use only. For each link, if set to '0', that link transmits PRBS data. .

GTX113_CTL_REG – Address 0x0004

Bit 15:8 – Reserved

Bit 7:4 - Debug use only. For each link, if set to '1', timing tags may be issued.

Bit 3:0 - Debug use only. For each link, if set to '0', that link transmits PRBS data. .

GTX114_CTL_REG – Address 0x0005

Bit 15:8 – Reserved

Bit 7:4 - Debug use only. For each link, if set to '1', timing tags may be issued.

Bit 3:0 - Debug use only. For each link, if set to '0', that link transmits PRBS data.

PRBS_CONTROL_0 – Address 0x0007

Bit 15:0 – Debug use only. For all links in PRBS mode, sets how many commas to send at reset before pattern starts.

PRBS_CONTROL_1 – Address 0x0008

Bit 15:0 – Debug use only. For all links in PRBS mode, how many words to send before inserting a comma.

PRBS_CONTROL_2 – Address 0x0009

Bit 15:4 – Reserved

Bit 3:0 – Debug use only. For all links in PRBS mode, how many commas to insert during each comma insertion time.

PRBS_CONTROL_3 – Address 0x000A

Bit 15:0 – Debug use only. For all links in PRBS mode, how many PRBS words to send before pattern restarts.

ILA_MUX_CTL_REG – Address 0x000B

Bit 15:0 – Debug use only.

MONITOR_FIFO_CTL_REG – Address 0x000C

Bit 15:0 – Debug use only.

ETH_XAUI_CTL_REG_1/2 – Address 0x0010/0x0030

Bit 15 – Ethernet XAUI Reset.

Bit 14:7 – Reserved.

Bit 6:5 – Test Pattern Selection:

"00" = High frequency test pattern

"01" = Low frequency test pattern

"10" = Mixed frequency test pattern

"11" = Reserved

Bit 4 - '1' enables Test Pattern

Bit 3 - '1' resets RX Link Status

Bit 2 - '1' resets TX and RX Local Fault bits

Bit 1 - '1' sets the transceivers into power down mode.

Bit 0 - '1' sets the transceivers into loopback mode

ETH_10GEMAC_TX_CONFIG_REG_1/2 – Address 0x0011/0x0031

Bit 15:2 – Reserved.
Bit 1 - '1' enables the transmitter.
Bit 0 - '1' resets the transmitter.

ETH_10GEMAC_RX_CONFIG_REG_1/2 – Address 0x0013/0x0033

Bit 15:2 – Reserved.
Bit 1 - '1' enables the receiver.
Bit 0 - '1' resets the receiver.

ETH_10GEMAC_PAUSE_REG_1/2 – Address 0x0015/0x0035

Bit 15:0 – Debug use only.

ETH_10GEMAC_CTL_REG_1/2 – Address 0x0016/0x0036

Bit 15:0 – Debug use only.

ETH_FLIC_MAC_ADDRESS_0_REG_1/2 – Address 0x0017/0x0037

Bit 15:8 – FLIC MAC Address Byte 2 (__-__-__-__-XX-__)
Bit 7:0 – FLIC MAC Address Byte 1. (__-__-__-__-__-XX)

ETH_FLIC_MAC_ADDRESS_1_REG_1/2 – Address 0x0018/0x0038

Bit 15:8 – FLIC MAC Address Byte 4 (__-__-XX-__-__-__)
Bit 7:0 – FLIC MAC Address Byte 3. (__-__-__-XX-__-__)

ETH_FLIC_MAC_ADDRESS_2_REG_1/2 – Address 0x0019/0x0039

Bit 15:8 – FLIC MAC Address Byte 6 (XX-__-__-__-__-__)
Bit 7:0 – FLIC MAC Address Byte 5. (__-XX-__-__-__-__)

ETH_FLIC_IP_ADDRESS_0_REG_1/2 – Address 0x001A/0x003A

Bit 15:8 – FLIC IP Address Byte 2 (__.__.XX.__)
Bit 7:0 – FLIC IP Address Byte 1. (__.__.__.XX)

ETH_FLIC_IP_ADDRESS_1_REG_1/2 – Address 0x001B/0x003B

Bit 15:8 – FLIC IP Address Byte 4 (XX.__.__.__)
Bit 7:0 – FLIC IP Address Byte 3 (__.XX.__.__)

ETH_FLIC_UDP_PORT_REG_1/2 – Address 0x001C/0x003C

Bit 15:0 – Sets the FLIC's UDP Port.

ETH_HOST_MAC_ADDRESS_0_REG_1/2 – Address 0x001D/0x003D

Bit 15:8 – Host MAC Address Byte 2 (__-__-__-__-XX-__)

Bit 7:0 – Host MAC Address Byte 1. (__-__-__-__-__-XX)

ETH_HOST_MAC_ADDRESS_1_REG_1/2 – Address 0x001E/0x003E

Bit 15:8 – Host MAC Address Byte 4 (__-__-XX-__-__-__)

Bit 7:0 – Host MAC Address Byte 3. (__-__-__-XX-__-__)

ETH_HOST_MAC_ADDRESS_2_REG_1/2 – Address 0x001F/0x003F

Bit 15:8 – Host MAC Address Byte 6 (XX-__-__-__-__-__)

Bit 7:0 – Host MAC Address Byte 5. (__-XX-__-__-__-__)

ETH_HOST_IP_ADDRESS_0_REG_1/2 – Address 0x0020/0x0040

Bit 15:8 – Host IP Address Byte 2 (__.__.XX.__)

Bit 7:0 – Host IP Address Byte 1. (__.__.__.XX)

ETH_HOST_IP_ADDRESS_1_REG_1/2 – Address 0x0021/0x0041

Bit 15:8 – Host IP Address Byte 4 (XX.__.__.__)

Bit 7:0 – Host IP Address Byte 3 (__.XX.__.__)

ETH_HOST_UDP_PORT_REG_1/2 – Address 0x0022/0x0042

Bit 15:0 – Sets the expected hosts UDP port. All data will be sent to this port.

TX_SEED_0 – Address 0x0050

Bit 15:0 – Debug use only.

TX_SEED_1 – Address 0x0051

Bit 15:0 – Debug use only.

TX_SEED_2 – Address 0x0052

Bit 15:0 – Debug use only.

TX_SEED_3 – Address 0x0053

Bit 15:0 – Debug use only.

DDR_CONTOL_REG – Address 0x0068

Bit 15:0 – Debug use only.

ENABLE_PROCESSING_REG – Address 0x006A

Bit 15:14 – Assembly routing control.
Bit 13 – Data Packet Sender 2 enable. (1 = enabled)
Bit 12 – Data Packet Sender 1 enable. (1 = enabled)
Bit 11:9 – Reserved.
Bit 8 – Fragment Data Assembly enable. (1 = enabled)
Bit 7 – GTX_115 Link 3 receiver enable. (1 = enabled)
Bit 6 – GTX_115 Link 2 receiver enable. (1 = enabled)
Bit 5 – GTX_115 Link 1 receiver enable. (1 = enabled)
Bit 4 – GTX_115 Link 0 receiver enable. (1 = enabled)
Bit 3 – GTX_114 Link 3 receiver enable. (1 = enabled)
Bit 2 – GTX_114 Link 2 receiver enable. (1 = enabled)
Bit 1 – GTX_114 Link 1 receiver enable. (1 = enabled)
Bit 0 – GTX_114 Link 0 receiver enable. (1 = enabled)

GTX112_PULSED_CTL_REG_A – Address 0x0200

Bit 15 – GTX_112 Link 3 RX Logic Reset. Set to '1' to reset, self clears after write.
Bit 14 – GTX_112 Link 3 RX PLL Reset. Set to '1' to reset, self clears after write.
Bit 13 – GTX_112 Link 3 TX Logic Reset. Set to '1' to reset, self clears after write.
Bit 12 – GTX_112 Link 3 TX PLL Reset. Set to '1' to reset, self clears after write.
Bit 11 – GTX_112 Link 2 RX Logic Reset. Set to '1' to reset, self clears after write.
Bit 10 – GTX_112 Link 2 RX PLL Reset. Set to '1' to reset, self clears after write.
Bit 9 – GTX_112 Link 2 TX Logic Reset. Set to '1' to reset, self clears after write.
Bit 8 – GTX_112 Link 2 TX PLL Reset. Set to '1' to reset, self clears after write.
Bit 7 – GTX_112 Link 1 RX Logic Reset. Set to '1' to reset, self clears after write.
Bit 6 – GTX_112 Link 1 RX PLL Reset. Set to '1' to reset, self clears after write.
Bit 5 – GTX_112 Link 1 TX Logic Reset. Set to '1' to reset, self clears after write.
Bit 4 – GTX_112 Link 1 TX PLL Reset. Set to '1' to reset, self clears after write.
Bit 3 – GTX_112 Link 0 RX Logic Reset. Set to '1' to reset, self clears after write.
Bit 2 – GTX_112 Link 0 RX PLL Reset. Set to '1' to reset, self clears after write.
Bit 1 – GTX_112 Link 0 TX Logic Reset. Set to '1' to reset, self clears after write.
Bit 0 – GTX_112 Link 0 TX PLL Reset. Set to '1' to reset, self clears after write.

GTX113_PULSED_CTL_REG_A – Address 0x0202

Bit 15 – GTX_113 Link 3 RX Logic Reset. Set to '1' to reset, self clears after write.
Bit 14 – GTX_113 Link 3 RX PLL Reset. Set to '1' to reset, self clears after write.
Bit 13 – GTX_113 Link 3 TX Logic Reset. Set to '1' to reset, self clears after write.
Bit 12 – GTX_113 Link 3 TX PLL Reset. Set to '1' to reset, self clears after write.
Bit 11 – GTX_113 Link 2 RX Logic Reset. Set to '1' to reset, self clears after write.
Bit 10 – GTX_113 Link 2 RX PLL Reset. Set to '1' to reset, self clears after write.
Bit 9 – GTX_113 Link 2 TX Logic Reset. Set to '1' to reset, self clears after write.
Bit 8 – GTX_113 Link 2 TX PLL Reset. Set to '1' to reset, self clears after write.
Bit 7 – GTX_113 Link 1 RX Logic Reset. Set to '1' to reset, self clears after write.
Bit 6 – GTX_113 Link 1 RX PLL Reset. Set to '1' to reset, self clears after write.
Bit 5 – GTX_113 Link 1 TX Logic Reset. Set to '1' to reset, self clears after write.
Bit 4 – GTX_113 Link 1 TX PLL Reset. Set to '1' to reset, self clears after write.
Bit 3 – GTX_113 Link 0 RX Logic Reset. Set to '1' to reset, self clears after write.
Bit 2 – GTX_113 Link 0 RX PLL Reset. Set to '1' to reset, self clears after write.
Bit 1 – GTX_113 Link 0 TX Logic Reset. Set to '1' to reset, self clears after write.
Bit 0 – GTX_113 Link 0 TX PLL Reset. Set to '1' to reset, self clears after write.

GTX114_PULSED_CTL_REG_A – Address 0x0204

Bit 15 – GTX_114 Link 3 RX Logic Reset. Set to '1' to reset, self clears after write.
Bit 14 – GTX_114 Link 3 RX PLL Reset. Set to '1' to reset, self clears after write.
Bit 13 – GTX_114 Link 3 TX Logic Reset. Set to '1' to reset, self clears after write.
Bit 12 – GTX_114 Link 3 TX PLL Reset. Set to '1' to reset, self clears after write.
Bit 11 – GTX_114 Link 2 RX Logic Reset. Set to '1' to reset, self clears after write.
Bit 10 – GTX_114 Link 2 RX PLL Reset. Set to '1' to reset, self clears after write.
Bit 9 – GTX_114 Link 2 TX Logic Reset. Set to '1' to reset, self clears after write.
Bit 8 – GTX_114 Link 2 TX PLL Reset. Set to '1' to reset, self clears after write.
Bit 7 – GTX_114 Link 1 RX Logic Reset. Set to '1' to reset, self clears after write.
Bit 6 – GTX_114 Link 1 RX PLL Reset. Set to '1' to reset, self clears after write.
Bit 5 – GTX_114 Link 1 TX Logic Reset. Set to '1' to reset, self clears after write.
Bit 4 – GTX_114 Link 1 TX PLL Reset. Set to '1' to reset, self clears after write.
Bit 3 – GTX_114 Link 0 RX Logic Reset. Set to '1' to reset, self clears after write.
Bit 2 – GTX_114 Link 0 RX PLL Reset. Set to '1' to reset, self clears after write.
Bit 1 – GTX_114 Link 0 TX Logic Reset. Set to '1' to reset, self clears after write.
Bit 0 – GTX_114 Link 0 TX PLL Reset. Set to '1' to reset, self clears after write.

PULSED_RESETS_REG – Address 0x020F

Bit 15:14 – Reserved
Bit 13 – GTX_114, GTX_113, and GTX112 TX Logic reset. Set to '1' to reset, self clears after write.
Bit 12:10 – Reserved
Bit 9 – GTX_114, GTX_113, and GTX112 TX FIFO reset. Set to '1' to reset, self clears after write.
Bit 8:1 – Reserved
Bit 0 – Fragment Assembly master logic reset. Set to '1' to reset, self clears after write.

U3/U4 General Setup Guidelines

Phase 1: Setup MAC and IP Address

The user should setup the MAC and IP address after a power cycle, and prior to other configuration.

Step 1.1: Configure FLIC MAC Address

Configure the FLICs Eth1 and Eth2 MAC address for both U3 and U4. The following example configures the MAC address as follows:

U3 Eth1: 00-11-22-33-44-55
U3 Eth2: AA-BB-CC-DD-EE-FF
U4 Eth1: 01-23-45-67-78-9A
U4 Eth2: 45-67-89-AB-CD-EF

Line	Command	Value	Target FPGA	Target Register	Target Address
1	Write	0x4455	U3	ETH_FLIC_MAC_ADDRESS_0_REG_1	0x0017
2	Write	0x2233	U3	ETH_FLIC_MAC_ADDRESS_1_REG_1	0x0018
3	Write	0x0011	U3	ETH_FLIC_MAC_ADDRESS_2_REG_1	0x0019
4	Write	0xEEFF	U3	ETH_FLIC_MAC_ADDRESS_0_REG_2	0x0037
5	Write	0xCCDD	U3	ETH_FLIC_MAC_ADDRESS_1_REG_2	0x0038
6	Write	0xAABB	U3	ETH_FLIC_MAC_ADDRESS_2_REG_2	0x0039
7	Write	0x789A	U4	ETH_FLIC_MAC_ADDRESS_0_REG_1	0x0017
8	Write	0x4567	U4	ETH_FLIC_MAC_ADDRESS_1_REG_1	0x0018
9	Write	0x0123	U4	ETH_FLIC_MAC_ADDRESS_2_REG_1	0x0019
10	Write	0xCDEF	U4	ETH_FLIC_MAC_ADDRESS_0_REG_2	0x0037
11	Write	0x89AB	U4	ETH_FLIC_MAC_ADDRESS_1_REG_2	0x0038
12	Write	0x4567	U4	ETH_FLIC_MAC_ADDRESS_2_REG_2	0x0039

Step 1.2: Configure FLIC IP Addresses

Configure the FLICs Eth1 and Eth2 IP address for both U3 and U4. The following example configures the MAC address as follows:

U3 Eth1: 10.10.10.10
U3 Eth2: 10.10.10.11
U4 Eth1: 10.10.10.12
U4 Eth2: 10.10.10.13

Line	Command	Value	Target FPGA	Target Register	Target Address
13	Write	0x0A0A	U3	ETH_FLIC_IP_ADDRESS_0_REG_1	0x001A
14	Write	0x0A0A	U3	ETH_FLIC_IP_ADDRESS_1_REG_1	0x001B
15	Write	0x0A0B	U3	ETH_FLIC_IP_ADDRESS_0_REG_2	0x003A
16	Write	0x0A0A	U3	ETH_FLIC_IP_ADDRESS_1_REG_2	0x003B
17	Write	0x0A0C	U4	ETH_FLIC_IP_ADDRESS_0_REG_1	0x001A
18	Write	0x0A0A	U4	ETH_FLIC_IP_ADDRESS_1_REG_1	0x001B
19	Write	0x0A0D	U4	ETH_FLIC_IP_ADDRESS_0_REG_2	0x003A
20	Write	0x0A0A	U4	ETH_FLIC_IP_ADDRESS_1_REG_2	0x003B

Step 1.3: Configure FLIC Port Numbers

Configure the FLICs Eth1 and Eth2 UDP port from which fragment assemblies will be sent.
U3 Eth1: 50000
U3 Eth2: 50001
U4 Eth1: 50002
U4 Eth2: 50003

Line	Command	Value	Target FPGA	Target Register	Target Address
21	Write	0xC350	U3	ETH_FLIC_UDP_PORT_REG_1	0x001C
22	Write	0XC351	U3	ETH_FLIC_UDP_PORT_REG_2	0x003C
23	Write	0XC352	U4	ETH_FLIC_UDP_PORT_REG_1	0x001C
24	Write	0xC353	U4	ETH_FLIC_UDP_PORT_REG_2	0x003C

Step 1.4: Configure Destination/Receiver MAC, IP, and Port

Set the destination MAC address, IP address, and port number for Eth1 and Eth2 for both U3 and U4. This set where fragment assemblies will be sent. This example configures the following destination MAC/IP and ports:

U3 Eth1: MAC: 11-11-11-11-11-11, IP: 10.10.10.20, Port: 51000
U3 Eth2: MAC: 22-22-22-22-22-22, IP: 10.10.10.21, Port: 51001
U4 Eth1: MAC: 33-33-33-33-33-33, IP: 10.10.10.22, Port: 51002
U4 Eth2: MAC: 44-44-44-44-44-44, IP: 10.10.10.23, Port: 51003

Line	Command	Value	Target FPGA	Target Register	Target Address
25	Write	0x1111	U3	ETH_HOST_MAC_ADDRESS_0_REG_1	0x001D
26	Write	0x1111	U3	ETH_HOST_MAC_ADDRESS_1_REG_1	0x001E
27	Write	0x1111	U3	ETH_HOST_MAC_ADDRESS_2_REG_1	0x001F
28	Write	0x2222	U3	ETH_HOST_MAC_ADDRESS_0_REG_2	0x003D
29	Write	0x2222	U3	ETH_HOST_MAC_ADDRESS_1_REG_2	0x003E
30	Write	0x2222	U3	ETH_HOST_MAC_ADDRESS_2_REG_2	0x003F
31	Write	0x3333	U4	ETH_HOST_MAC_ADDRESS_0_REG_1	0x001D
32	Write	0x3333	U4	ETH_HOST_MAC_ADDRESS_1_REG_1	0x001E
33	Write	0x3333	U4	ETH_HOST_MAC_ADDRESS_2_REG_1	0x001F
34	Write	0x4444	U4	ETH_HOST_MAC_ADDRESS_0_REG_2	0x003D
35	Write	0x4444	U4	ETH_HOST_MAC_ADDRESS_1_REG_2	0x003E
36	Write	0x4444	U4	ETH_HOST_MAC_ADDRESS_2_REG_2	0x003F
37	Write	0x0A14	U3	ETH_HOST_IP_ADDRESS_0_REG_1	0x0020
38	Write	0x0A0A	U3	ETH_HOST_IP_ADDRESS_1_REG_1	0x0021
39	Write	0x0A15	U3	ETH_HOST_IP_ADDRESS_0_REG_2	0x0040
40	Write	0x0A0A	U3	ETH_HOST_IP_ADDRESS_1_REG_2	0x0041
41	Write	0x0A16	U4	ETH_HOST_IP_ADDRESS_0_REG_1	0x0020
42	Write	0x0A0A	U4	ETH_HOST_IP_ADDRESS_1_REG_1	0x0021
43	Write	0x0A17	U4	ETH_HOST_IP_ADDRESS_0_REG_2	0x0040
44	Write	0x0A0A	U4	ETH_HOST_IP_ADDRESS_1_REG_2	0x0041
45	Write	0XC738	U3	ETH_HOST_UDP_PORT_REG_1	0x0022
46	Write	0XC739	U3	ETH_HOST_UDP_PORT_REG_2	0x0042
47	Write	0XC740	U4	ETH_HOST_UDP_PORT_REG_1	0x0022
48	Write	0XC741	U4	ETH_HOST_UDP_PORT_REG_2	0x0042

Phase 2: FPGA to FPGA (intra-board) Link Configuration

The FPGA-to-FPGA links should be configured prior to configuration of the SSB and RTM links. Even though all internal links are not used, there is no drawback to configuring and locking all inter-FPGA links. The Following table provides a mapping of the inter-FPGA links.

From / To	U1	U2	U3	U4
U1	---	GTX113	GTX114	GTX115
U2	GTX114	---	GTX113	GTX115
U3	GTX114	GTX113	---	GTX112
U4	GTX114	GTX113	GTX112	---

The following example configures all inter-FPGA links:

Step 2.1: Reset RX/TX

First put all inter-FPGA frame checker, generator, and FIFO logic into reset. The following four commands put U1/U2's GTX113, GTX114 and GTX115; and U3/U4's GTX112, GTX113 and GTX114, frame checking logic into reset.

Line	Command	Value	Target FPGA	Target Register	Target Address
49	Set	0x078C	U1	GENERAL_CTL_REG	0x0002
50	Set	0x078C	U2	GENERAL_CTL_REG	0x0002
51	Set	0x01EC	U3	GENERAL_CTL_REG	0x0002
52	Set	0x01EC	U4	GENERAL_CTL_REG	0x0002

Step 2.2: Switch to Real Data Mode

Next, switch all inter-FPGA links to real data mode. This is achieved by the following four commands. Note that different register addresses are used for U1/U2 and U3/U4.

Line	Command	Value	Target FPGA	Target Register	Target Address
53	Write	0x000F	U1	GTX113_CTL_REG	0x0012
54	Write	0x000F	U1	GTX114_CTL_REG	0x0013
55	Write	0x000F	U1	GTX115_CTL_REG	0x0014
56	Write	0x000F	U2	GTX113_CTL_REG	0x0012
57	Write	0x000F	U2	GTX114_CTL_REG	0x0013
58	Write	0x000F	U2	GTX115_CTL_REG	0x0014
59	Write	0x000F	U3	GTX112_CTL_REG	0x0004
60	Write	0x000F	U3	GTX113_CTL_REG	0x0005
61	Write	0x000F	U3	GTX114_CTL_REG	0x0006
62	Write	0x000F	U4	GTX112_CTL_REG	0x0004
63	Write	0x000F	U4	GTX113_CTL_REG	0x0005
64	Write	0x000F	U4	GTX114_CTL_REG	0x0006

Step 2.3: Reset the RX/TX GTX control logic.

Line	Command	Value	Target FPGA	Target Register	Target Address
65	Write	0xFFFF	U1	GTX113_PULSED_CTL_REG_A	0x0202
66	Write	0xFFFF	U1	GTX114_PULSED_CTL_REG_A	0x0204
67	Write	0xFFFF	U1	GTX115_PULSED_CTL_REG_A	0x0206
68	Write	0xFFFF	U2	GTX113_PULSED_CTL_REG_A	0x0202
69	Write	0xFFFF	U2	GTX114_PULSED_CTL_REG_A	0x0204
70	Write	0xFFFF	U2	GTX115_PULSED_CTL_REG_A	0x0206
71	Write	0x5555	U3	GTX112_PULSED_CTL_REG_A	0x0200
72	Write	0xAAAA	U3	GTX112_PULSED_CTL_REG_A	0x0200
73	Write	0x5555	U3	GTX112_PULSED_CTL_REG_A	0x0200
74	Write	0x5555	U3	GTX113_PULSED_CTL_REG_A	0x0202
75	Write	0xAAAA	U3	GTX113_PULSED_CTL_REG_A	0x0202
76	Write	0x5555	U3	GTX113_PULSED_CTL_REG_A	0x0202
77	Write	0x5555	U3	GTX114_PULSED_CTL_REG_A	0x0204
78	Write	0xAAAA	U3	GTX114_PULSED_CTL_REG_A	0x0204
79	Write	0x5555	U3	GTX114_PULSED_CTL_REG_A	0x0204
80	Write	0x5555	U4	GTX112_PULSED_CTL_REG_A	0x0200
81	Write	0xAAAA	U4	GTX112_PULSED_CTL_REG_A	0x0200
82	Write	0x5555	U4	GTX112_PULSED_CTL_REG_A	0x0200
83	Write	0x5555	U4	GTX113_PULSED_CTL_REG_A	0x0202
84	Write	0xAAAA	U4	GTX113_PULSED_CTL_REG_A	0x0202
85	Write	0x5555	U4	GTX113_PULSED_CTL_REG_A	0x0202
86	Write	0x5555	U4	GTX114_PULSED_CTL_REG_A	0x0204
87	Write	0xAAAA	U4	GTX114_PULSED_CTL_REG_A	0x0204
88	Write	0x5555	U4	GTX114_PULSED_CTL_REG_A	0x0204

Step 2.4: Enable GTX RX/TX Logic

Now take the inter-FPGA frame checker, generator, and FIFO logic out of reset.

Line	Command	Value	Target FPGA	Target Register	Target Address
89	Clear	0x078C	U1	GENERAL_CTL_REG	0x0002
90	Clear	0x078C	U2	GENERAL_CTL_REG	0x0002
91	Clear	0x01EC	U3	GENERAL_CTL_REG	0x0002
92	Clear	0x01EC	U4	GENERAL_CTL_REG	0x0002

Step 2.5: Configure L1 Routing Registers

Set the desired L1 ID mask values for the U1 and U2 Core Crate Links. This control the routing of events to either U3 or U4 by the L1 ID.

Line	Command	Value	Target FPGA	Target Register	Target Address
93	Write	User val	U1	U3_L1_ID_MATCH_REG_1	0x002A
94	Write	User val	U1	U3_L1_ID_MATCH_REG_2	0x002B
95	Write	User val	U1	U4_L1_ID_MATCH_REG_1	0x002C
96	Write	User val	U1	U4_L1_ID_MATCH_REG_2	0x002D
97	Write	User val	U2	U3_L1_ID_MATCH_REG_1	0x002A
98	Write	User val	U2	U3_L1_ID_MATCH_REG_2	0x002B
99	Write	User val	U2	U4_L1_ID_MATCH_REG_1	0x002C
100	Write	User val	U2	U4_L1_ID_MATCH_REG_2	0x002D

Continue to configure the external links, prior to proceeding to Phase 3.

Phase 3: Logic Initialization

The phase performs final initialization of the fragment assembly logic prior to the start of a run.

Step 3.1: Reset Fragment Assembly Logic

Reset all fragment assembly logic.

Line	Command	Value	Target FPGA	Target Register	Target Address
101	Set	0x0001	U3	PULSED_RESETS_REG	0x020F
102	Set	0x0001	U4	PULSED_RESETS_REG	0x020F

Step 3.2: Configure Ethernet Interfaces, and Active Assembly Channels

Set which SSB channels will participate in fragment assembly and which Ethernet interfaces will be active.

Example values:

All assembly's to Eth1: 0x11XX

All assembly's to Eth2: 0xE1XX

Send half to Eth1 and half to Eth2: 0xB1XX

The lower 8 bits ("XX") set which SSB are monitored for purposes of fragment assembly.

Line	Command	Value	Target FPGA	Target Register	Target Address
103	Write	0xB1FF	U3	ENABLE_PROCESSING_REG	0x006A
104	Write	0xB1FF	U4	ENABLE_PROCESSING_REG	0x006A