OptoHybrid v3 Firmware

Thomas Lenzi ([thomas.lenzi@cern.ch)](mailto:thomas.lenzi@cern.ch))  
Andrew Peck ([andrew.peck@cern.ch)](mailto:andrew.peck@cern.ch))  
Evaldas Juska ([evaldas.juska@cern.ch](mailto:evaldas.juska@cern.ch))

August 21, 2017

This document describes how to interact with the OptoHybrid (OH) modules and how to parameterize and use the various functionalities integrated in the firmware.

# Contents

[Contents 1](#_Toc491102360)

[Introduction 4](#_Toc491102361)

[Hardware Description 5](#_Toc491102362)

[Clocking 6](#_Toc491102363)

[Fabric Clocks 6](#_Toc491102364)

[GBT Clock 6](#_Toc491102365)

[Synchronization 8](#_Toc491102366)

[Trigger Link Frame Sequencing 8](#_Toc491102367)

[TTC Monitoring 8](#_Toc491102368)

[GBT E-Link Deserialization 9](#_Toc491102369)

[In “16 bit” full-bandwidth mode 9](#_Toc491102370)

[In “10 bit” low-bandwidth mode 9](#_Toc491102371)

[GBT Packets (Full Bandwdith Mode) 10](#_Toc491102372)

[Transmit (FPGA to GBT) 10](#_Toc491102373)

[Receive (GBT to FPGA) 12](#_Toc491102374)

[GBT Packets in Low Bandwidth Mode 13](#_Toc491102375)

[Transmit (FPGA to GBT) 13](#_Toc491102376)

[Receive (GBT to FPGA) 14](#_Toc491102377)

[S-bit Deserialization 15](#_Toc491102378)

[Trigger Algorithm 16](#_Toc491102379)

[Clock 1 16](#_Toc491102380)

[Clock 2 16](#_Toc491102381)

[Clock 3 17](#_Toc491102382)

[Clock 4 17](#_Toc491102383)

[Clock 5 18](#_Toc491102384)

[Clock 6 18](#_Toc491102385)

[Clock 7 18](#_Toc491102386)

[Clock 8 19](#_Toc491102387)

[Clock 9 19](#_Toc491102388)

[Clock 10 19](#_Toc491102389)

[Clock 11 19](#_Toc491102390)

[Clock 12 20](#_Toc491102391)

[Clock 13 20](#_Toc491102392)

[Clock 14 20](#_Toc491102393)

[Clock 16 20](#_Toc491102394)

[Cluster Packer Miscellany 22](#_Toc491102395)

[Synchronization Frame 22](#_Toc491102396)

[Cluster Counter 22](#_Toc491102397)

[LED Indicators 23](#_Toc491102398)

[Registers 24](#_Toc491102399)

[Loopback 25](#_Toc491102400)

[Addressing 25](#_Toc491102401)

[Errors 25](#_Toc491102402)

[Errors to avoid 25](#_Toc491102403)

[Counters 26](#_Toc491102404)

[Addressing 26](#_Toc491102405)

[System Registers 28](#_Toc491102406)

[Addressing 28](#_Toc491102407)

[Status Registers 29](#_Toc491102408)

[Addressing 29](#_Toc491102409)

[ADC 30](#_Toc491102410)

[Addressing 30](#_Toc491102411)

[Errors 30](#_Toc491102412)

[Errors to avoid 30](#_Toc491102413)

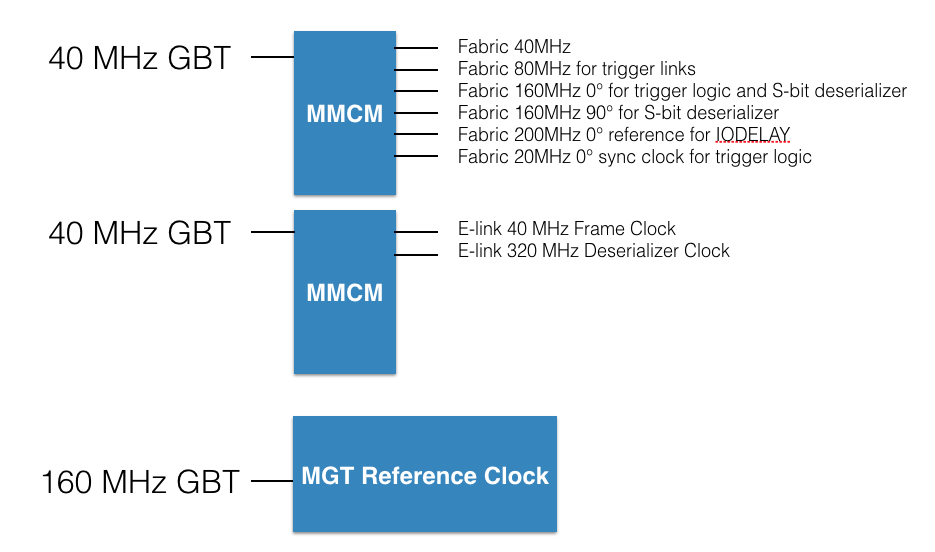
[History 31](#_Toc491102414)

# Introduction

# Hardware Description

# Clocking

Basic clocking structure of the Optohybid is as shown in the following figure.



### Fabric Clocks

### GBT Clock

Note that there is an interesting caveat to the GBT 40 and 320 MHz clocks.

The GBT IO suffer an irreducible insertion delay as they enter the FPGA. This is described in the Xilinx datasheet by TIOPI, the “delay from IOB pad through the input buffer to the I-pin of an IOB pad. The delay varies depending on the capability of the SelectIO input buffer.”

The clock inputs suffer this same delay. In a normal system where the 320 MHz clock was in the same region as the GBT I/Os, and the GBT I/Os were in the same regions as each other, we would use regional clock buffers (subject to this same delay) that would directly drive the logic and this would be accounted with an OFFSET\_IN / OUT constraint applied to the pins.

But instead we deserialize and frame align using MMCM outputs. The MMCM feedback is specifically designed to eliminate this delay so that the MMCM clock is synchronous with the input pin. Generally this is desirable, but in our case the insertion delay incurred on the data inputs but NOT on the clock means that the data will be phase shifted relative to the clock. This can be correctly in several ways:

1. Delay the incoming data through an IDELAY element to phase align it back to the clock
2. Phase shift the incoming 40MHz clock (using the GBT) to align the clock correctly to account for this
3. Phase shift the MMCM outputs to provide a negative delay to the GBT inputs

In the Optohybrid version 3 firmware, as of this writing, **choice #3** was made. Thus in the MMCM configuration you will see a phase shift applied to both the 40MHz and 320 MHz clocks, of 15 and 125 degrees, respectively, to account for the approximate 1.08 ns insertion delay of the IOBs.

This should allow the Optohybrid to work with a 40MHz GBT clock which is phase aligned to the center of the data eye (as the GBT sends by default).

# Synchronization

The primary purpose of the Optohybrid firmware is its role in the firmware. Integral to this role is the ability to synchronize each Optohybrid to a known phase.

Two provisions are implemented in the Optohybrid firmware to aid this.

### Trigger Link Frame Sequencing

In addition to the 56 bits of trigger data that are sent every bunch crossing, each fiber link also transmits an 8 bit frame delimiter which is used to align the frames into packets.

There are several frame markers which can be chosen, and we exploit this to provide information about the Optohybrid which will aid in synchronization.

During normal operation, the frame will follow a predictable cycle of 0xBC, 0xF7, 0xFB, 0xFD.

In this simplest form we can ensure crude synchronization between links and different chambers as well as monitoring the data integrity, by comparing the sequence to a local state machine in the backend electronics.

Two additional separators are used to mark special conditions:

1. **BC0:** the bunch crossing zero flag received from the GBTx is connected to the trigger link logic. When a BC0 (or equivalently BX0) is received the frame marker will be changed to 0x50.
2. **Overflow:** the trigger links are only capable of transmitting 8 clusters per bunch crossing. Hence there is a possibility for overflows to occur, wherein clusters are found by not transmitted. When this happens, the trigger link will change its frame separator to 0xFC.

### TTC Monitoring

The Optohybrid has a local accumulator which increments bunch crossing number (bxn) and produces a locally generated bx0 flag.

In order for the two flags, (1) locally generated and (2) received from GBTx, to be in sync a programmable parameter (bxn offset) is provided which allows you to set an “offset” which is the value which will be assumed by the bxn counter after ttc resync.

The synchronization of these two flags can be monitored by checking the status of bxn\_sync\_error in the OH status register (*n.b. that bx0\_sync\_error is NOT suitable for this, as it is a 1bx wide pulse useful for internal counters when the bx0 is received, while bxn\_sync\_error will persist for the entire orbit*).

When the Optohybrid is correctly timed in to the TTC system, bxn\_sync\_error should be 0. A software routine in the CTP-7 to find the correct bxn offset should be easily designed.

A counter in the counters module counts the number of bx0 sync errors since the last resync or hard-reset.

# GBT E-Link Deserialization

With the existence of two different communication modes corresponding yielding 16 bits/bx and 10 bits/bx (duplex) there will be two alternative assignments of which bits will be carried on which e-links.

In “16 bit” full-bandwidth mode:

We specify that one eLink (labelled in firmware as eLink0) be run at 320 MHz and carry the 8 least-significant bits of the packet. The other eLink (labelled in firmware as eLink1) will run at 320 MHz and carry the 8 most-significant bits.

i.e.

|  |  |  |
| --- | --- | --- |
|  | **E-link** | **Payload** |
| RX  (GBT to OH) | eLink1 | TTC[3:0],  data[11:8] |
| eLink0 | data[7:0] |
| TX  (OH to GBT) | eLink1 | data[15:8] |
| eLink0 | data[ 7:0] |

In “10 bit” low-bandwidth mode:

We specify that one eLink (labelled in firmware as eLink0) be run at 80 MHz and carry the 2 least-significant bits of the packet. The other eLink (labelled in firmware as eLink1) will run at 320 MHz and carry the 8 most-significant bits.

i.e.

|  |  |  |
| --- | --- | --- |
|  | **E-link** | **Payload** |
| RX  (GBT to OH) | eLink1 | TTC[3:0],  data[5:2] |
| eLink0 | data[1:0] |
| TX  (OH to GBT) | eLink1 | data[9:2] |
| eLink0 | data[1:0] |

Within each e-link, all bits are transmitted and most-significant bit first.

# GBT Packets (Full Bandwdith Mode)

In “Full Bandwidth Mode” the GBT to FPGA link consists of two 320 Mbps duplex links. In version 3a this was not implemented, and a separate functionality of 1x 320 Mbps duplex and 1x 80Mbps duplex was adopted. This alternative, “Low Bandwidth Mode” is covered in the next section.

### Transmit (FPGA to GBT)

Communication with the optohybrid in the direction of FPGA to GBT is accomplished with a simple 16 bit packet format.

A valid packet is three frames long and will consist of the 16 bit frame marker BCBC followed by two 16 bit data frames:

|  |  |
| --- | --- |
| **Frame** | **frame[15:0]** |
| Start | 0xBCBC |
| Data0 | data[31:16] |
| Data1 | data[15: 0] |

For example, this waveform shows the transmission of the 32-bit packet 0x12345678:



Consecutive packets will simply repeat this sequence, e.g:

|  |  |
| --- | --- |
| **Frame** | **frame[15:0]** |
| Start | 0xBCBC |
| Data0 | data[31:16] |
| Data1 | data[15: 0] |
| Start | 0xBCBC |
| Data0 | data[31:16] |
| Data1 | data[15: 0] |

When the transmit FIFO is empty and there is no data to send, the state machine will send an idle frame marker, 0xFCFC:

|  |  |
| --- | --- |
| **Frame** | **frame[15:0]** |
| Start | 0xBCBC |
| Data0 | data[31:16] |
| Data1 | data[15: 0] |
| IDLE | 0xFCFC |
| IDLE | 0xFCFC |
| IDLE | 0xFCFC |
| IDLE | 0xFCFC |
| Start | 0xBCBC |
| Data0 | data[31:16] |
| Data1 | data[15: 0] |

For example, this waveform shows the transmission of the 32-bit packet 0x12345678, followed by the idle transmission of the 0xFCFC frame marker.



The receiving state machine should monitor for 0xBCBC for alignment and record the next two frames as the 32 bit data packet.

If the following frame is 0xBCBC, then the sequence should repeat and record the following two frames, and so on. If the frame is 0xFCFC then the state machine should wait until it sees a start-of-packet marker (0xBCBC).

### Receive (GBT to FPGA)

Communication in the direction of GBT to FPGA is similarly accomplished with a packet format, whose payload includes 16 bits of address and 32 bits of data. Transmit and receive bandwidths are asymmetric, due to the requirement of transmitting synchronous TTC commands on the GBT to FPGA link, so the format on the receive side is somewhat more complicated.

In this direction, the highest 4 bits of the GBT link are reserved for TTC commands:

* Bit15 = L1A
* Bit14 = VFAT Reset
* Bit13 = TTC Resync
* Bit12 = BC0

The remaining 12 bits are used as the GBT data frame used to construct packets.

|  |  |  |
| --- | --- | --- |
| **Frame** | **Highest 4 bits** | **Lowest 12 bits** |
| Start | L1A + reset + Resync + BC0 | 0xABC |
| Header | L1A + reset + Resync + BC0 | req\_valid + wr\_enable + reserved[5:0] + address[15:12] |
| Address 0 | L1A + reset + Resync + BC0 | address[11:0] |
| Data 0 | L1A + reset + Resync + BC0 | data[31:24] |
| Data 1 | L1A + reset + Resync + BC0 | data[23:12] |
| Data 2 | L1A + reset + Resync + BC0 | data[11: 0] |

For continuous transmission, this sequence should just be repeated cyclically. i.e. the frame after Data 2 should be “Start” (0xABC). E.g. as follows:

**<START> <HEADER> <ADR0><DATA0> <DATA1> <DATA2>, <START> <HEADER> <ADR0> ….**

When idle, there are two possible ways of sending data to the OH:

Either cycle through the 7 frame packet format continuously, making sure to set request\_valid to 0 for any null packets. The Optohybrid will process and discard the packet.

Alternatively (preferably) when can send an idle frame marker, for example 0x5FC. The exact choice does not matter as long as it is not 0xABC. This marker can be sent continuously as long as the transmission is idle and the Optohybrid will remain in an idle SYNC state. The state machine will monitor for 0xABC and if received, will continue to process the rest of the frame normally.

# GBT Packets in Low Bandwidth Mode

In “Low Bandwidth Mode” the GBT to FPGA link consists of 1x 320 Mbps duplex and 1x 80Mbps duplex, leaving 10 bits per bunch crossing in each direction.

### Transmit (FPGA to GBT)

Communication with the optohybrid in the direction of FPGA to GBT is accomplished with a simple 10 bit packet format.

A valid packet is three five long and will consist of the a bit frame marker BC followed by three 10 bit data frames:

|  |  |
| --- | --- |
| **Frame** | **frame[9:0]** |
| Start | 2’b00 & 0xBC |
| Data0 | 2’b00 & data[31:24] |
| Data1 | 2’b01 & data[23:16] |
| Data2 | 2’b10 & data[15: 8] |
| Data3 | 2’b11 & data[ 7: 0] |

For example, this waveform shows the transmission of the 32-bit packet 0x12345678:



Note that the highest two bits of the data frames indicate the numbering of the frame in the packet and can be used for error checking.

Consecutive packets will simply repeat this sequence.

When the transmit FIFO is empty and there is no data to send, the state machine will send an idle frame marker, 0xFC. For example, this waveform shows the transmission of the 32-bit packet 0x12345678, followed by the idle transmission of the 0xFC frame marker, followed by the beginning of a new valid packet.



The receiving state machine should monitor for 0xBC for alignment and record the next four frames as the 32 bit data packet.

If the frame following the last data frame is 0xBC, then the sequence should repeat and record the following two frames, and so on. If the frame is 0xFC (or anything else) then the state machine should wait until it sees a start-of-packet marker (0xBC).

### Receive (GBT to FPGA)

Communication in the direction of GBT to FPGA is similarly accomplished with a packet format, whose payload includes 32 bits of address and 32 bits of data. Transmit and receive bandwidths are asymmetric, due to the requirement of transmitting synchronous TTC commands on the GBT to FPGA link, so the format on the receive side is somewhat more complicated.

In this direction, the highest 4 bits of the GBT link are reserved for TTC commands:

* Bit10 = L1A
* Bit9 = VFAT Reset
* Bit8 = TTC Resync
* Bit7 = BC0

The remaining 6 bits are used as the GBT data frame used to construct packets.

|  |  |  |
| --- | --- | --- |
| **Frame** | **Highest 4 bits** | **Lowest 6 bits** |
| Start | L1A + reset + Resync + BC0 | 0x2A |
| Header | L1A + reset + Resync + BC0 | request\_valid + wr\_enable + reserved[3:0] |
| Address 0 | L1A + reset + Resync + BC0 | address[15:10] |
| Address 1 | L1A + reset + Resync + BC0 | address[ 9: 4] |
| Address 2 | L1A + reset + Resync + BC0 | address[3:0] + data[31:30] |
| Data 0 | L1A + reset + Resync + BC0 | data[29:24] |
| Data 1 | L1A + reset + Resync + BC0 | data[23:18] |
| Data 2 | L1A + reset + Resync + BC0 | data[17:12] |
| Data 3 | L1A + reset + Resync + BC0 | data[11: 6] |
| Data 4 | L1A + reset + Resync + BC0 | data[ 5: 0] |

For continuous transmission, this sequence should just be repeated cyclically. i.e. the frame after Data 2 should be “Start” (0xABC). E.g. as follows:

**<START><HEADER><ADR0><ADR1><ADR2><DATA0><DATA1><DATA2><DATA3><DATA4>, <START>**

When idle, there are two possible ways of sending data to the OH:

Either cycle through the 7 frame packet format continuously, making sure to set request\_valid to 0 for any null packets. The Optohybrid will process and discard the packet.

Alternatively (preferably) when can send an idle frame marker, for example 0x1C. The exact choice does not matter as long as it is not 0x2A. This marker can be sent continuously as long as the transmission is idle and the Optohybrid will remain in an idle SYNC state. The state machine will monitor for 0x2A and if received, will continue to process the rest of the frame normally.

# S-bit Deserialization

# Trigger Algorithm

Trigger information in the Optohybrid comes from the VFAT’s S-bits (Sector bits, or Sum bits, depending on who you ask). In the GE1/1 VFAT 3 these take the form of 8 320 MHz trigger links plus a 320 MHz synchronization pulse used to align the S-bits to the 40 MHz clock.

From 24 VFATs, the S-bits links are deserialized into 1536 S-bits divided in 8 physical partitions.

The S-bits are passed into a module called the cluster packer, which uses a priority encoding scheme to find the coordinates of 8 S-bit clusters inside of the GEM chamber.

All of the priority encoding logic operates at 160 MHz, so the number of bx consumed by the algorithm is 1/4 of the number of 160 MHz clocks.

A sequential description of the algorithm follows:

### Clock 1

Oneshots trim the tails of S-bit pulses in order to prevent retriggering on the same S-bit in subsequent clock cycles. The S-bit will become active again for triggering after it goes low for 1 clock cycle. This is especially important if the monostable length of the VFAT is set to anything longer than one.

An optional deadtime parameter can be controlled through the sys module to provide a 4-bit delay (0-15 bunch crossings) during which the S-bit cannot retrigger at all. This may be useful to supress afterpulsing which has been observed in the VFATs.

### Clock 2

The second clock cycle is responsible for the creation of cluster flags and sizes which will be inputs to the priority encoder. A cluster is defined as a continuous sequence of S-bits, which can be encoded together to save bandwidth.

In the chosen data format, the maximum size of a cluster is 7, meaning that the primary S-bit was active along with 7 additional neighbours, for a total cluster size of 7. A cluster size of 0 indicates that only 1 S-bit was active.

Due to limitations of the logic, if a cluster is longer than 8 S-bits, its tail will simply be truncated.

There is an optional, compile-time parameter that can change this behaviour so that the tail of a cluster could be split into a 2nd cluster, with the limitation that after 16 S-bits it would be truncated again. By default this is disabled, due to the resource savings that are achieved.

In the firmware, two processes occur in parallel:

1st is that a “valid primary flag” is created for each of the 1536 S-bits, which identifies the particular S-bit as being the primary (first) S-bit in a cluster. This is done by looking for a preceding 0 (S-bit OFF).

2nd is that a count is derived for all pads, which is a 3-bit number following the scheme above.

For each of the 1536 S-bits, these 4 bits (vpf and count) are registered and passed to the priority encoder.

### Clock 3

This clock is used to register the vpfs and counts of the 1536 trigger pads (vpfs are latched inside the truncator, counts are latched inside the priority encoder), providing a 6.25 ns routing slack which allows the signals to be routed through the FPGA and aligned at the priority encoder.

Nota bene: This clock cycle is critical in the choice of the phase of the frame clock. The frame clock should be adjusted such that the rising edge of the 20 MHz frame clock is coincident with the rising edge of the clock at the end of this cycle.

Each 160 MHz clock cycle represents a 45 degree phase shift of the frame clock. Thus, because this step is in Clock 3, the phase shift applied to the frame clock should be 45 \* 3 = 135 degrees. This needs to be correctly configured in the MMCM

### Clock 4

Multi-bit priority encoding presents a significant challenge for a latency-constrained system. Finding a single cluster can be done efficiently through a multi-stage pipelined priority encoding tree. But finding subsequent clusters is very slow, because the entirety of the process must be pipelined:

e.g. 3 clock cycles to find the first cluster, mask it off, 3 clock cycles to find the second cluster, mask it off, 3 clock cycles to find the 3rd, etc.

It should be apparent that in such a scheme the latency would be very large.

Fortunately a clever scheme was devised which allows for masking and encoding to occur as separate, parallel processes.

This exploits the fact that the twos complement of a number has an interesting relation to its least significant set bit (note that the twos\_complement of a = -a = ~a+1).

At each clock cycle, the least-significant 1 becomes 0, using a simple property of integers: subtracting 1 from a number will always affect the least-significant set 1-bit. Using just arithmetic, with this trick we can take some starting number, and generate a copy of it that has the least-significant 1 changed to a zero.

e.g.

let a = 101100**1**00 // our starting number   
 ~a = 010011**0**11 // bitwise inversion  
 b = ~a+1 = 010011**1**00 // b is the twos complement of a  
 ~b = 101100**0**11 // bitwise inversion  
 a & b = 000000**1**00 // one hot of first one set  
 a &~b = 101100**0**00 // copy of a with the first set bit changed to 0

or as a one line expression in Verilog,

c = a & ~(~a+1), or equivalently  
 c = a & ~( -a), or equivalently  
 c = a & ~({1536{1'b1}}-a), etc., I'm sure there are more.

But alas, the point: we can Zero out bits without knowing the position of the bit, So this so-called cluster-truncator can run independently of a priority encoder that is finding the position of the bit. This allows the cluster truncation to be the timing critical step (running at 160 MHz) while the larger amount of logic in the priority encoder can be pipelined, to run over 2 or 3 clock cycles, which adds an overall latency but still allows the priority encoding to be done quickly without the necessity of a pipelined feedback.

This reduces the overall encoding time for 8 clusters in the example above from 3\*8 = 24 clock cycles to 2+8 = 10 clock cycles (2bx latency + 8 bx of data).

Additionally, operating on such large amounts of data (1536 bits) is a significant processing hurdle. To simplify this, split the chamber into two parts, each of which is finding 8 clusters from that half of the chamber (768). The findings of the two encoders will be merged at the end.

**Having said all this**--- in this clock cycle:

1. The “cluster truncator” applies this twos-complement math to the S-bit valid-primary flags and registers a modified version of the logic with the least-significant S-bit set to 0
   1. Cluster n-1
2. In parallel, the data from the cluster truncator flip-flops is routed to the priority encoder, where it will be registered in order to find the 1st valid cluster.

### Clock 5

1. The “cluster truncator” applies its twos-complement math to the S-bit valid-primary flags and registers a modified version of the logic with the least-significant S-bit set to 0
   1. Cluster n-2
2. In parallel, the data the first 4 pipeline stages of the priority encoder are completed, reducing the search set from 768 to 48 clusters.

### Clock 6

1. The “cluster truncator” applies its twos-complement math to the S-bit valid-primary flags and registers a modified version of the logic with the least-significant S-bit set to 0
   1. Cluster n-3
2. In parallel, the data the next 4 pipeline stages of the priority encoder are completed, reducing the search set from 48 to a single cluster, and returns the address, cnt, and vpf of that cluster
   1. *1st valid cluster*

### Clock 7

1. The “cluster truncator” applies its twos-complement math to the S-bit valid-primary flags and registers a modified version of the logic with the least-significant S-bit set to 0
   1. Cluster n-4
2. In parallel, the data the next 4 pipeline stages of the priority encoder are completed, reducing the search set from 48 to a single cluster, and returns the address, cnt, and vpf of that cluster
   1. 2nd valid cluster

### Clock 8

1. The “cluster truncator” applies its twos-complement math to the S-bit valid-primary flags and registers a modified version of the logic with the least-significant S-bit set to 0
   1. Cluster n-5
2. In parallel, the data the next 4 pipeline stages of the priority encoder are completed, reducing the search set from 48 to a single cluster, and returns the address, cnt, and vpf of that cluster (cluster #3)
   1. *3rd valid cluster*

### Clock 9

1. The “cluster truncator” applies its twos-complement math to the S-bit valid-primary flags and registers a modified version of the logic with the least-significant S-bit set to 0
   1. Cluster n-6
2. In parallel, the data the next 4 pipeline stages of the priority encoder are completed, reducing the search set from 48 to a single cluster, and returns the address, cnt, and vpf of that cluster (cluster #4)
   1. *4th valid cluster*

### Clock 10

1. The “cluster truncator” applies its twos-complement math to the S-bit valid-primary flags and registers a modified version of the logic with the least-significant S-bit set to 0
   1. Cluster n-7
2. In parallel, the data the next 4 pipeline stages of the priority encoder are completed, reducing the search set from 48 to a single cluster, and returns the address, cnt, and vpf of that cluster (cluster #5)
   1. *5th valid cluster*

### Clock 11

1. The truncation process for this set of S-bits has completed and we are only awaiting the latency of the priority encoder. Because the data is a continuous input stream, we have a pipelined logic design which utilizes this logic again for new data before the previous set of S-bits has finished processing.   
     
   So in this clock cycle we load new data into the register from bunch crossing N+2 (where N was the bx of the data which we just completed) and repeat the same process:   
     
   The “cluster truncator” applies this twos-complement math to the S-bit valid-primary flags and registers a modified version of the logic with the least-significant S-bit set to 0 (cluster-1)  
     
   For clarity, in subsequent clock cycles, the descriptions of processing on the data from BX = N+2 is removed.
2. In parallel, the data the next 4 pipeline stages of the priority encoder are completed, reducing the search set from 48 to a single cluster, and returns the address, cnt, and vpf of that cluster (cluster #6)
   1. *6th valid cluster*

### Clock 12

In this clock:

1. The data the next 4 pipeline stages of the priority encoder are completed, reducing the search set from 48 to a single cluster, and returns the address, cnt, and vpf of that cluster (cluster #7)
   1. *7th valid cluster*

### Clock 13

1. The data the next 4 pipeline stages of the priority encoder are completed, reducing the search set from 48 to a single cluster, and returns the address, cnt, and vpf of that cluster (cluster #8)
   1. *8th valid cluster*
2. The 8 clusters from each of the two priority encoders are collated and aligned for input in into the merge16 module, which uses a merge-sort algorithm to pick out 8 clusters from the 16 inputs.

### Clock 14

1. The merge 16 module completes the first sets of sorting, swapping pairs from different halves of the chamber depending on whether which has the lowest address (invalid patterns are automatically set to the maximum address 0x7FF).   
     
   Stage 0: sort eights (0,8), (1,9), (2,10), (3,11), (4,12), (5,13), (6,14), (7,15)
2. The merge 16 module completes the second set of sorting, swapping pairs of clusters depending on which has the lowest address.   
     
   Stage 1: sort fours (4,8), (5,9), (6,10), (7,11)

### Clock 16

1. The merge 16 module completes the third set of sorting, swapping pairs of clusters depending on which has the lowest address.   
     
   Stage 2: sort twos (2,4), (3,5), (6,8), (7,9)
2. The merge 16 module completes the fourth and last set of sorting, swapping pairs of clusters depending on which has the lowest address.   
     
   Stage 3: swap odd pairs (1,2), (3,4), (5,6), (7,8), (9,10), (11,12), (13,14)
3. The 8 sorted clusters are multiplex into the 40MHz clock domain.   
     
   The 8 sorted clusters have a mask applied to them, so that during reset they are set to 0x7FE and if the Optohybrid trigger control is idle (waiting for bc0) they will be set to 0x7FD.   
     
   The masked clusters are moved onto the fabric 40MHz clock and will be sent out on the optical links.

A total of 16 clocks at 160Mhz means that the entire cluster encoding process is 4bx (100ns), exactly.

Retiming the module to consume an additional BX while adding additional pipeline stages would greatly ease the burden of place-and-route, but at this point does not seem necessary.

# Cluster Packer Miscellany

This section contains various other notes on the configuration, inputs, and outputs of the cluster packer aside from its algorithm.

### Synchronization Frame

Performance of the cluster packer is achieved by splitting the chamber into two time-multiplexed logic blocks. Every bunch crossing, either the “odd” or “even” encoders receives the entire chamber’s worth of data and produces a result.

The results from the odd or even encoders are multiplexed back together and output from the cluster packer module.

Rather than a counter, which I worried would be too fickle and subject to SEU, the synchronization and multiplexing of these two modules is achieved with a MMCM derived 20MHz clock.

The odd and even modules are given either 0 or 180 degree versions of this 20MHz clock and they synchronize their logic and multiplexing to this clock.

This scheme was chosen because on every clock cycle of the slow clock, each of the priority encoder modules will reset itself on the rising edge of this clock, irrespective of any previous state.

### Cluster Counter

A submodule of the cluster packer produces a full 12 bit count of the number of clusters found in the chamber. This is accomplished with a multi-step pipelined adder tree that produces its result faster than the cluster packer. An SRL delay must be correctly programmed to align the adder results with the output of the cluster packer.

In another module of the Optohybrid firmware, the cluster count output is used in a rate counter. This module averages the cluster count over a compile-time programmable time window and produces an output in Hertz.

# LED Indicators

# Registers

A summary table of the wishbone slaves in the Optohybrid v3 is given here. Detailed descriptions of individual registers follow on subsequent pages.

|  |  |  |  |
| --- | --- | --- | --- |
| Slave | Name | Addresss | Function |
| Wishbone Slaves | | | |
| 0 | Loopback | 0x0000 | Loopback register receives data and responds with the same data, for testing communication and error rates. Data in the loopback register has no effects in the FPGA. |
| 1 | Counters | 0x10YY | Counters register allows readback of a number of counters |
| 2 | System | 0x20YY | System register is responsible for controlling various settings in the Optohybrid |
| 3 | Status | 0x30YY | Status register is responsible for readback of various status |
| 4 | ADC | 0x40YY | Controls the System Monitor of the Virtex-6 FPGA |

In an address such as 0x10YY, the lowest 11 bits denoted by YY are used to select between individual end-points in the wishbone bus. Splitting the request is handled by the individual wishbone slaves.

The highest 5 bits are used to select the wishbone slave. Presently only the highest 4-bits are used, so that addresses count as 0x0000, 0x1000, 0x2000, 0x3000… for future expansion the 5th bit is reserved such that if additional addresses are needed we can occupy the 0x0800, 0x1800, 0x2800, 0x3800… up to 0xF800. In total this provides space for 32 wishbone slaves, from 0x0000 to 0xF800, each of which can have 2048 endpoints.

## Loopback

This module is designed for testing GBT communication. It will echo as a response whatever data is written to it as a request.

### Addressing

Module ID 0

Address 0x0000

|  |  |  |
| --- | --- | --- |
| Y register | Mode | Function |
| Loopback Registers | | |
| 0 | Read/write | Echoes back the last written value as a wishbone response |

### Errors

None

### Errors to avoid

* None

## Counters

This module holds all the counters of the OptoHybrid. Writing to a given register will reset its value.

All counters are 32 bit

### Addressing

Module ID 1

Address 0x10YY

|  |  |  |
| --- | --- | --- |
| Y register | Mode | Function |
| Control | | |
| 0 | Read | *Snap Enable*  Write (anything) to this register to take a snapshot of the counters, freezing a shadow copy for synchronous readout. This MUST be flagged before reading out a register or it will not be updated.  The register will readback a counter of the number of snapshots taken since last reset. |
| 1 | Read | *Reset All*  Write anything to this register to reset all of the counters  The register will readback a counter of the number of reset\_alls done. This register is reset by writing to the register. A non-writing request will readback the contents. |
| Wishbone |  |  |
| 2 | Read | *Wishbone master strobes*  Counts the number of strobes received on the wishbone master (GBT) |
| 3 | Read | *Wishbone master acknowledgments*  Counts the number of acknowledgements returned by the wishbone master. |
| 4—8 | Read | *Wishbone slaves strobe*  Order: Loopback, counters, system, status, adc |
| 9—13 | Read | *Wishbone slaves acknowledgments*  Order: Loopback, counters, system, status, adc |
| TTC |  |  |
| 14 | Read | *L1A Counter* |
| 15 | Read | *Resync Counter* |
| 16 | Read | *BC0 Counter* |
| 17 | Read | *BC0 Sync Error Counter* |
| MMCM | | |
| 18 | Read | *GBT Frame Clock MMCM Unlocked Counter* |
| 19 | Read | *Fabric Clock MMCM Unlocked Counter* |
| 20 | Read | *Either MMCM Unlocked Counter* |
| GBT | | |
| 21 | Read | *GBT Link Errors* |
| SEM |  |  |
| 22 | Read | *SEM Corrections* |
| S-Bits |  |  |
| 23 | Read | *SBit Overflows* |
| 24—47 | Read | *Active VFAT Flags count the number of the number of bx with at least one S-bit in a VFAT* |
| Latency Timers | | |
| 48—71 | Read | *Latency timer: Number of BX between the SBits and the L1A, for each VFAT* |

## System Registers

System registers are writable registers used to control programmable settings of the Optohybrid

### Addressing

Module ID 2

Address 0x20YY

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Y register | Mode | Bits | Function | |
| VFAT | | | |
| 1 | Read / write | 4:0 | *SBit magic select for HDMI* | |
| 2 | Write only | 0 | *VFAT reset*  Write any data to this register and it will reset the VFATs (for 1 bx.. is this sufficient??). This register automatically returns to a 0 state | |
|  |  |  |  | |
| 2 | Read / write | 23:0 | *VFAT trigger data mask*  Allows to mask individual VFATs for trigger data | |
| 3 | Read / write | 29:0 | *TDC SBits selects* | |
| 4 | Read / write | 1:0 | *TDC SBits mode*  0 : single VFAT2  1 : Eta row (OR of 3 VFAT2s)  2 : Sector (OR of 4 VFAT2s)  3 : no output | |
| 5 | Read / write | 0 | FMM dont\_wait  If this is asserted, the Optohybrid will not wait for resync 1st BC0 before sending trigger data. This is useful for test-stands and debugging where a reliable TTC startup sequence may not be expected. | |
| 6 | Read / write | 11:0 | TTC BXN Offset  This 12 bit configuration will set the “zero” value of the local BC0 counter which will be assumed after resync. Adjust this to account for different latencies to chambers. When this register is correctly set the BC0 error counters in the status register should be zero.  Note that since the OH does not have a startup configuration this must be programmed by the CTP-7 after every hard reset. | |
| 7 | Read / write | 3:0 | *Trigger Deadtime*  Adds a deadtime, 0-15 bunch crossings, on each S-bit which will prevent retriggering on that same S-bit during the time window specified by the deadtime. This is meant to supress afterpulsing in the trigger data path. | |

## Status Registers

List of status registers

### Addressing

Module ID 3

Address 0x30YY

|  |  |  |  |
| --- | --- | --- | --- |
| Y register | Mode | Bit(s) | Function |
| Version | | | |
| 0 | Read only | 31:0 | *OH firmware date*  *Hex encoded: YYYYMMDD* |
| 1 | Read only | 31:0 | *Firmware version Hex Encoded*  Major 8 bits  Minor 8 bits  Version 8 bits  HW 8 bits = Hardware ID A = v3A, etc. |
| Clocking | | | |
| 2 | Read only | 0  1  2 | *Both MMCMs Locked*  *GBT SerDes Frame Clock MMCM Locked*  *Fabric clock MMCM Locked* |
| Soft Error Mitigation | | | |
| 3 | Read only | 0 | *Critical error induced by SEU detected* |
| GBT |  |  |  |
| 4 | Read only | 0  1  2 | *GBT TX Ready*  *GBT RX Valid*  *GBT RX Ready* |
| Trigger |  |  |  |
| 5 | Read only | 31:0 | *Averaged cluster rate in Hertz* |
| TTC |  |  |  |
| 6 | Read only | 31:0 | *Local BC0 Counts* |
| 7 | Read only | 31:0 | *Received BC0 Counts from GBT* |
| 8 | Read only | 31:0 | *TTC Orbit Counter* |
| 9 | Read only | 11:0 | *TTC BXN Count* |
| 10 | Read only | 0 | *TTC BXN Sync Error* |

## ADC

This module is directly connected to the xADC of the Virtex-6 FPGA. Refer to the following user guide for a full list of register: <http://www.xilinx.com/support/documentation/user_guides/ug370.pdf>

### Addressing

Module ID 4

Address 0x40YY

|  |  |  |
| --- | --- | --- |
| Y register | Mode | Function |
| ADC registers | | |
| 0 - 79 | Read/write | Returns the conversion value of a given channel  0 = Temperature  1 = VCCINT  2 = VCCAUX  3 = VP/VN  16-31 = VAUX[0:15] (not connected)  32 = Temperature max  33 = VCCINT max  34 = VCCAUX max  36 = Temperature min  37 = VCCINT min  38 = VCCAUX max  64-66 = Control registers[0:2]  67-71 = Test registers[0:4]  72-79 = Sequencer registers[0:7] |

### 

### Errors

The module returns an error if the parameters are not in spec.

### Errors to avoid

* The register ID must be in the range 0 to 79.

# History

|  |  |  |
| --- | --- | --- |
| Date | Author | Modification |
| 8/7/2017 | AP | Initial conversion to OHv3 from Thomas’ v2 document |
| 8/10/2017 | AP | Description of GBT link formats |
| 8/14/2017 | AP | Updates register definitions according to latest firmware |
| 8/14/2017 | AP | Add description of synchronization and algorithmic description of the cluster packer |
| 8/17/2017 | AP | Update trigger description to match new firmware (reduced latency) |
| 8/18/2017 | AP | Update register description and GBT decoding for 16 bit addresses |