CCU25

Communication and Control Unit
ASIC
for Embedded Slow Control

A. Marchioro C. Ljuslin C. Paillard

D R A F T

1. DOCUMENT HISTORY

27/3/97- A.M. - Rev 0.10
First draft version.
30/4/97 – A.M. Rev 0.127
Clean-up and improvements
24/7/97 – A.M. Rev 0.143
Clarifications on broadcast, clean-up, packet length.
8/6/00 – A.M. Rev. 2.0
Major Revision for 0.25 um version
19/2/02
- A.M., C.P. cleanup after submission and first tests

2. GENERAL

The Communication and Control Unit ASIC (CCU-25) is a special purpose integrated circuit built in a radiation hard technology. It is used to implement a dedicated control link system in the CMS tracker for the control and monitoring of the embedded front-end electronics. It is normally used in conjunction with another ASIC, a special purpose PLL (Phase-Locked-Loop), which is necessary for the distribution of the time critical trigger and of the low jitter clock to the front end ASICs.

This document describes the CCU, its logical and electrical interfaces, programming features and operating modes. It also includes descriptions of the chip pin-out and electrical characteristics. In its basic architecture, the CCU supports a ring-type network topology, although a point-to-point type of network can also be configured.

To put this ASIC in the context where it will be used, a brief review of the tracker control system is provided in the next section. On the other side, the CCU is not limited to applications in the CMS tracker, as no special features and/or constraints of the tracker have determined its architecture.

2.1. Overview of CMS tracker Slow Control

The CMS tracker control system uses a ring topology configured as a local area network. A module called FEC (Front-End-Controller) is the master of the network and uses two fibers for sending the timing and data signals to the slave modules which in turn use two to fibers to transmit a return clock and the return data back. Due to the relatively long distance between the control room and the detector, the communication between the embedded electronics on the CMS tracker and the external electronics is expected to use a ribbon of four optical fibers. An equivalent network can obviously be built using a copper link in locations where the distance is limited.

This data link is synchronized to the LHC clock frequency and has a raw capacity of 40 Mbit/sec. The synchronization of the FEC and the embedded electronics is provided by the clock line. CCUs are typically mounted on Control Modules (CCUM), housing the necessary ancillary electronics, such as line drivers, receivers and level translators.

To minimize costs of a slow control system, it is expected to connect a certain number of Control Modules serially on local part of the detector resulting in a ring-like arrangement as the one shown in the figure below.

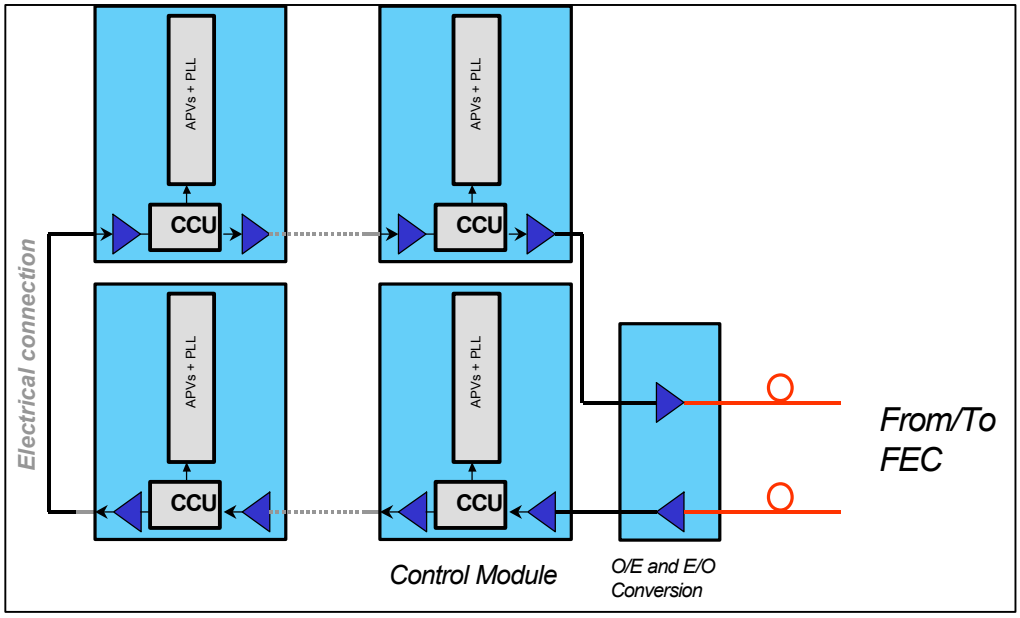


Figure 2 Control ring, simplified view

The arrangement shown in the figure assumes that the connection between the FEC and the first Control Module is done via optical fibers, the connections between embedded Control modules is done electrically using differential lines (LVDS) and again the connection back to the FEC is optical.

The CCU does not support directly connections to optical elements, and therefore separate optical to electrical interface modules including laser drivers and p-i-n diode receivers are required.

The length of a single electrical connections between CCUs is expected not to exceed 50-60 cm, with about 8-10 control modules per ring for a total length of the electrical portion of the ring not exceeding 2-3 meters.

A block diagram of the CCU is shown in the figure below:

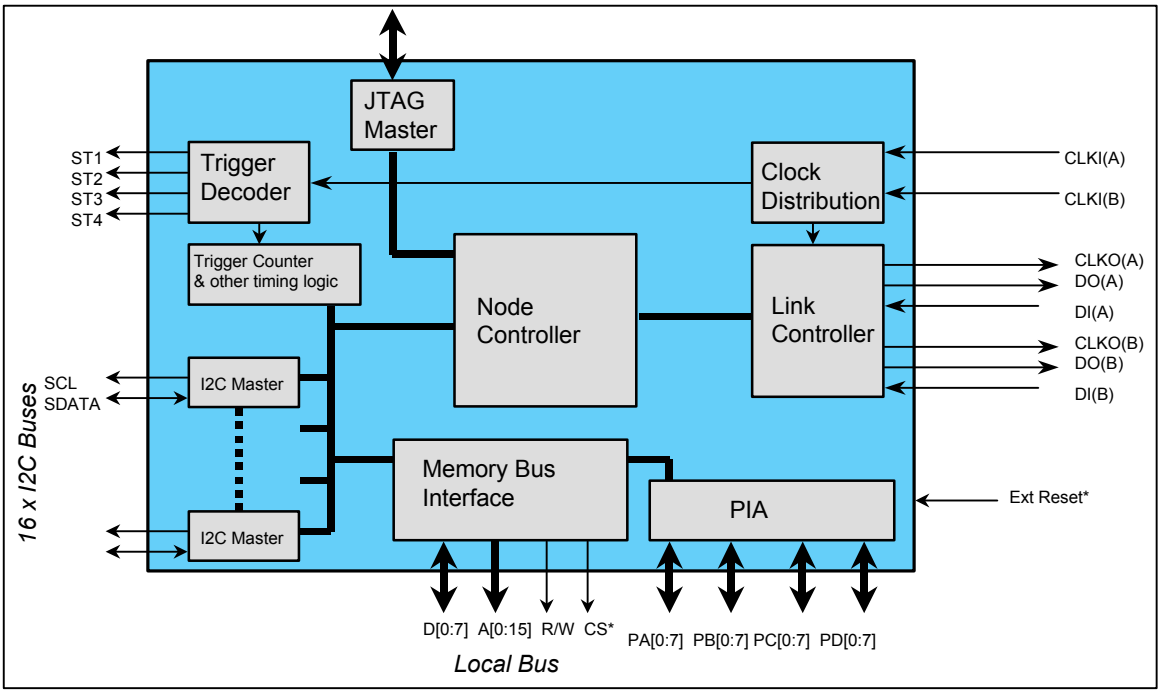


Figure 3 CCU Block Diagram

2.2. Overview of Communication Architecture

The communication architecture used by the CCU is based on two layers:

- The first layer (called the **Ring**) connects the FEC to CCUs and the CCUs between themselves; the protocol on this layer is message based and is implemented in a way similar to standard computer LAN networks. The protocol used on this layer will be called the **Ring Protocol**.
- The second layer connects the CCU itself to other chips via so called **Channels**. The protocols used here are called the **Channel Protocols**.

The first layer is unified and common to all CCUs, and is based on a LAN architecture transporting data packets to and from the FEC and Channel controllers. The second layer is specific to the channel, and different kind of physical implementations of the channels are foreseen.

The CCU25 contains the following channel blocks:

- One node controller (the CCU control itself is seen as a special channel capable for instance to report the status of the other CCU channels)
- Sixteen I2C master controllers
- One memory-like bus controller to access devices such as static memories, A/D converters etc.
- Four I/O like parallel bus controllers such as the ones used in the Motorola PIA etc.
- One trigger distribution controller
- One JTAG master controller

The dual network layer architecture introduced above is necessary to support applications where long cables/fibers are used between the FEC and the CCUs (therefore generating long delays) and to support the relatively slow buses chosen to interface to the front end chips, such as the I2C bus. This architecture assumes that the control is done by sending data packets (messages) to the respective channels, which interpret the messages as commands, execute them on their external interfaces (for example just a read or write operation to a memory bus) and conditionally return a status reply to the FEC via another message.

This protocol assumes that the remote devices controlled by the CCUs are seen from the FEC as remote independent channels, each one with a particular set of control registers and/or allocated memory locations. The channels operate independently from each other to allow concurrent transactions. The channels can perform transfers to their end-devices concurrently.

The high level ring network layer, being a local area network-like protocol, is controlled by software running on an appropriate microprocessor through the FEC.

To decouple the operation of the channels with respect to the one of the ring, the architecture assumes that all operations on the channels are asynchronous and do not demand an immediate acknowledgement. Basically this means that all commands carried by the ring under the form of network messages are posted to the channel interfaces. This is easy to implement for write operations, where practically one works by posting write operations to the channels. For read operations one instead sends a read request to the channel using a request packet; the channel performs the operation on its interface and returns a message to the requester using a separate packet.

Broadcast operations are supported in a similar manner. Only write broadcasts are supported. For example, a broadcast operation to several I2C ports proceeds as follows:

1. a broadcast message is sent to all I2C channels in a CCU

2. the I2C channels execute the command concurrently but do not complete it necessarily at the same time
3. if no error occurs, no acknowledgment is sent back
4. I2C channels with errors report their status conditions back by sending different error report messages back to the command originator.
5. A FEC can always examine the status of the I2C channels, or any other channel, by interrogating the CCU node controller in the CCU at channel #0.

A logical view of this multi-channel architecture is shown in the next figure:

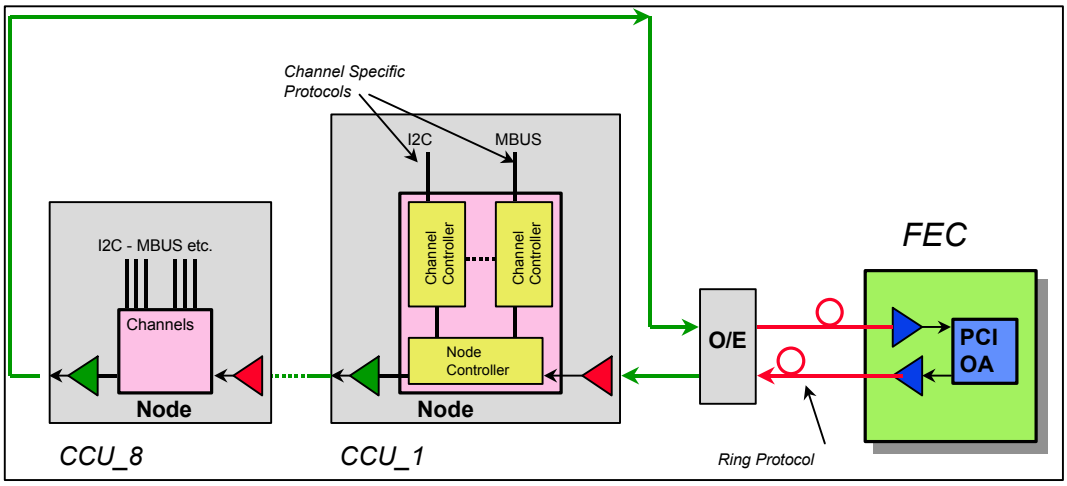


Figure 5 Logical view of control ring

2.3. Radiation tolerance features

To be able to operate in the CMS tracker environment, the CCU25 is designed with rad-tolerance features. These features include:

- rad-tolerant library cells, for total dose tolerance
- redundant circuitry on critical logic blocks for SEU robustness.

Redundant circuitry using three identical copies of the same logic and voting logic is used for the node controller. This block is critical for the correct functioning of the CCU25, it is included three times in the chip and all its output signals are selected after a voting circuit (purely combinatorial) which takes as valid output the signal for which at least 2 out of the three blocks agree.

To make the rest of the logic blocks more robust against single event upsets, the following circuit features are also implemented:

- all data paths are protected by parity
- all finite state machines are encoded as one-hot circuits, majority logic with voting added to node controller

These two features allow the logic in the CCU to discover nodes which have been flipped due to an SEU event. Whenever such an error occurs, the corresponding logic block aborts execution of the current operation and signals this event to the node controller. These features are sufficient to guarantee that the CCU will not perform a wrong operation on one of the slave chips attached to one of its channels.

3. RING ARCHITECTURE

3.1. Token-ring like protocol

The architecture of a ring of CCUs is functionally very similar to (and inspired from) the one used by commercial token ring networks (similar to the IBM's Token Ring or FDDI).

The ring consists basically of a number of node devices (the FEC and the embedded CCUs), that are all capable of accepting and inserting packets in the ring. In the simplest implementation the ring consists of only two devices, the FEC and one embedded CCU, thus resembling a point to point network.

The basic token-ring message transmission protocol for a the ring is based on the following mechanism:

1) at start up the FEC starts circulating idle patterns in the ring
2) following initialization, the FEC inserts a token in the ring which starts circulating from node to node
3) all nodes not wanting to transmit will just forward the token, a node which wants to transmit information to another node waits for arrival of the network token
4) this node replaces the token with a data frame and transmits it in the ring
5) all the nodes reached by the frame which are not the destination of the current frame just forward the frame
6) the destination node copies the passing frame, modifies just one symbol([1]) at the tail of the frame and forwards the rest of the frame to the network
7) the emitting node receives back its original frame with one symbol modified, removes it from the ring and regenerates an empty token.

To guarantee the synchronization and proper operation of the ring, a token packet is generated automatically by the FEC whenever the ring is initialized and travels around the ring.

The above protocol assumes that an entire ring has only one circulating token or data packet at any one time.

In this network implementation, to simplify the design of the CCUs, the FEC node will contain more complexity, such as the capability of generating empty tokens and of initializing and diagnosing the network.

Handling of communication errors in the ring is a complex issue and is discussed elsewhere in this manual.

3.2. Token ring packet Format

Two basic packet types are foreseen, one for signaling availability of the ring and the second to actually carry data.

The network token format is defined as:

SOF [1 B]	**EOF** [2 B([2])]

Figure 6 Token packet format

[1] "Symbol" refers to a set of four bits making up the full message packet. For more details see the following chapters. One byte contains two symbols.

[2] "B" refers to one byte

the network data packet format is defined below:

SOF [1B]	**Destination** DAddress [1 B]	**Source** SAddress [1 B]	**Length** [1 or 2 B]	**Data** [<128B or <32K B]	**CRC-16** [2 B]	**EOF** [2 B]

Figure 7 Data packet format

The Start of Frame (SOF), End of Frame (EOF), Source, Destination, Length and CRC fields are mandatory for all circulating data packets.

The SOF field is defined as the unique "J-H/K" sequence, using the two special characters defined in Figure 56 Control characters in network".

The SOF field is defined as follows:

Symbol	Name	Comment
0	J	Special symbol used for synchronisation
1	H/K	K used to mark token packet H used to mark normal data packet

Figure 8 Frame Header format

The End of Frame (EOF) field consists of two bytes and is defined as follows:

End Delimiter	**Frame Status**
[0.5 B]	[1.5 B]

Figure 9 EOF definition

The End Delimiter consists of a single "T" character as defined in the Figure 56 Control characters in network". The Frame status is generated by the transmitter as three "R" characters and contains the three following sub-fields:

Symbol	Value	Comment
ER	R/S	Error symbol
AR	R/S	Address recognised
DC	R/S	Data copied

Figure 10 The EOF frame

These symbols are set by the receiving node; they are generated as "R" symbols from the source and modified to "S" by the receiver.

The Length field can be one or two bytes long and gives the length of the data payload, excluding the two length bytes themselves and the CRC field.

When the high bit [bit 7] of the first byte is '0' the length of the field is one byte only and the maximum payload can be 0-127 bytes long. When the high bit is a '1', the length field is two bytes long and the data payload can be 0-32K bytes long.

The Data field is not interpreted by the ring protocol and is used exclusively by the channel adapter for internal addressing and data. This fields are defined explicitly by the functionality of each channel later in this document.

The data portion of the packet is shown below:

CH# [1 B]	**TR#** [1 B]	**Channel Specific Command** [(Length-2) B]

Figure 11 Data portion of packet

Two data bytes are mandatory as payload at the beginning of each data packet:

- a channel number (single byte field), used to identify a device channel within a node
- a transaction number (single byte field with wrap-around) used to assure correct identification of operation within a given channel. This field is always generated by the initiator of a transaction.

For transactions initiated by the FEC, the transaction number should always be in the range 1-255, as the special transaction number 0 is used for Alarms generated by the CCUs.

The CRC-16 field covers the packet content from the Destination address to the end of the Data field. The polynomial used for the CRC-16 calculations is:

$$X^{16} + X^{15} + X^2 + 1$$

3.2.1. Spacing between packets

All devices on the ring make sure that the minimum spacing between two packets is at least two bytes (four characters) of idle characters.

3.2.2. Broadcast mechanism on ring

Broadcast packets are used to send control information to all CCUs (typically from the FEC) and are defined in the special paragraph “Format of Broadcast packets“ on page 11.

The Destination and Source Address fields specify the addresses of the devices involved in a transaction, for broadcast packets the destination field should be set in the range 128-255.

Broadcast packets are never acknowledged by CCUs. Such packets always come back to the generator (FEC) with the three last characters in the EOF field set to “R”.

This broadcast mechanism does not guarantee that broadcast packets are seen by all CCUs and it should not be used for critical applications

Therefore, critical messages should never be sent using broadcast packets.

3.3. Node addressing

The following table specifies the node addressing scheme for a ring:

Address	Unit
0	FEC’s address
1-127	Address of CCUs
128-255	Broadcast classes 0-127

Figure 12 Node addressing on ring

This scheme assumes that only one FEC per ring will be used and that other FEC modules eventually on the ring (for example for debugging purposes) should use an address in the range used for CCUs.

Address assignment of the CCUs on the ring is done via dedicated pins on each CCU.

3.3.1. Format of Broadcast packets

The broadcast packets are marked by using the special address range 128-255 in the destination field and are otherwise identical to normal packets. Only the sender software in the FEC has to make sure that the broadcast operation is meaningful for all nodes.

Each CCU is assigned a unique broadcast class number in its internal broadcast register. Upon reception of a broadcast packet a CCU accepts it only if the broadcast address in the destination field is equal or less than its internal broadcast class.

3.4. Redundancy specifications

As each ring could control a sizable number of front-end channels it is important to be able to guarantee a very high reliability for the system. One malfunctioning element in a control ring will mean the loss of control of too many detector elements and it would clearly be unacceptable.

A redundancy scheme based on doubling signal paths and bypassing of interconnection lines between CCUs and between the CCU and the FEC is supported.

3.4.1. Skip Fault Architecture

A simplified schematic of the redundancy architecture used in the ring is shown in the figure below.

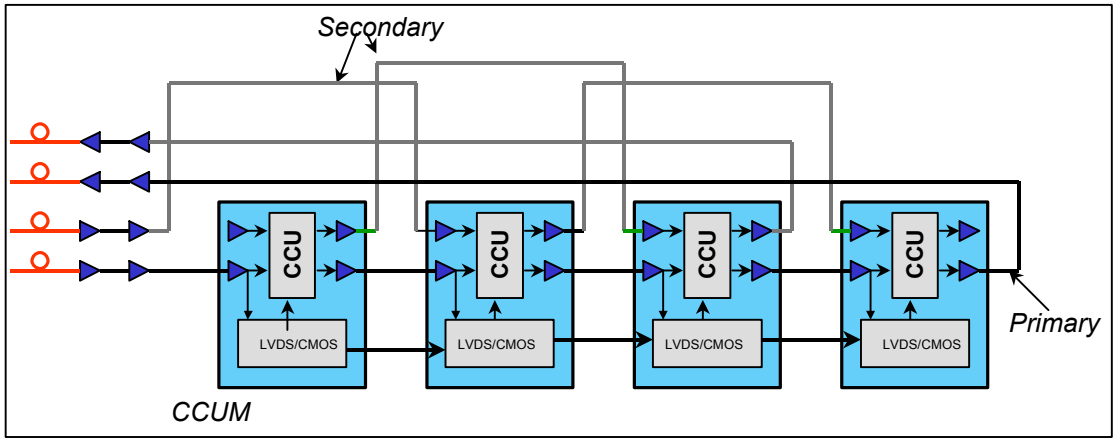


Figure 13 Redundant Skip-Fault Architecture

Basically the ring will consist of two independent data paths, one connecting the CCU modules serially and a second redundant path which alternatively skips one CCU module in the chain. To implement this architecture all modules, including the FEC, have two sets of input ports and two sets of output ports. These ports are simply called A and B ports. During normal operation, communication occurs among all adjacent modules using the A ports. Whenever one module fails, the ring can be configured to skip the faulty module. This is achieved by programming the module preceding the faulty one to select the B port as its output port. By using the section of bypassing ring the faulty module can be excluded from the communication chain and full ring protocol can be reestablished. This scheme works as long as one does not have two broken adjacent modules. Should this occur, the ring can not be configured to work any longer and it must be repaired.

3.4.2. Fault repair reconfiguration

The mechanism used to reconfigure a ring with a faulty module is explained in this paragraph with reference to the Figure 14.

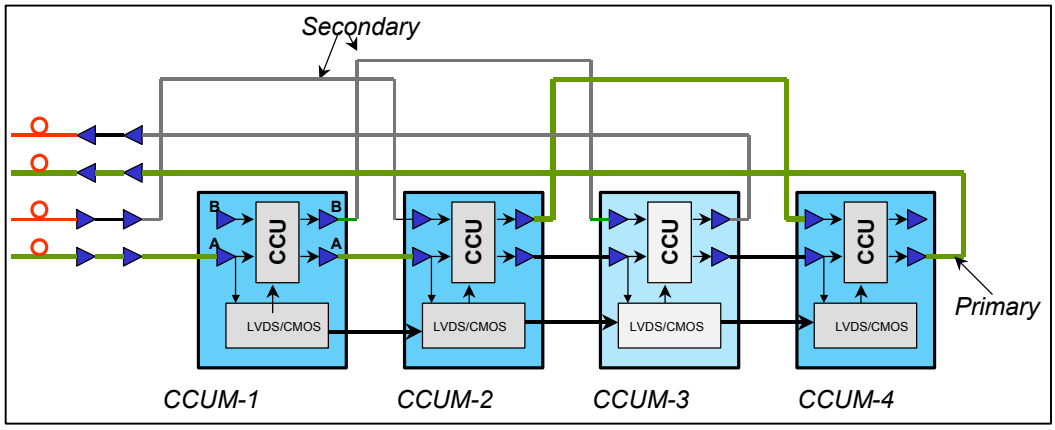


Figure 14 Fault repair reconfiguration example

A fault can occur anywhere in a CCU module in a control ring. Any of the components on the module, i.e. the CCU itself, the LVDS line driver, the local power supply, the connector etc. could become defective. In this example, one will assume that the CCUM-3 is somehow defective. This could be recognized in a variety of ways at the FEC level, for instance the ring may become silent, or a number of malformed packets could be received at the FEC or finally packet could occasionally become lost in the ring. Should this happen the software in the FEC has to start a fault scan procedure.

To support the search in software of the fault in the ring, the following protocol is implemented in the CCU:

- after hardware reset the CCU assumes that the good input and output ports are always the ports A.
- consider the CCU which has to be programmed to change its output port (in this example CCUM-2): the CCU can change the output port it uses only upon reception of a well formed packet addressed to the CCU itself from the active input port (A or B)
- consider the CCU which has to be programmed to change its input port (in this example CCUM-4): to instruct the CCU to switch to the alternate input port B (or back to A), a well formed packet with the command to perform this reconfiguration has to be fully received from the port B (or A) itself.
- packet requesting the switch of input and output ports may not be returned from the receiving CCU integrally to the FEC which should not expect to receive them back well formed. These packets will necessarily be cut by the CCU which performs the switch. During this reconfiguration, the FEC should not expect to receive well formed returning packets.

Notice that the FEC does not know where the fault occurred and has to search for it. The procedure is based on searching the fault from the end of the ring back to the first CCU module and can work as follows:

- the ring is reset, all CCUs are configured to use ports A for input and output
- sending packet or tokens to the faulty ring will result in no packet or malformed packets to be returned
- the FEC assumes that CCUM-4 is faulty. It will send a message to CCU-3 to use its alternative output port and the FEC itself is programmed to listen to input port B (this has to be so because the CCU-4 can not use the alternative output port, as this is not connected, see figure)

- If the fault existed at the level of the LVDS driver between CCU-3 and CCU-4, the communication should be reestablished and the ring should be back working. If the fault was instead in the CCU-3 itself, the search has to continue
- If the CCU-3 was faulty, the FEC would still see no returning packets from the ring. It will then assume that CCUM-3 is faulty and reconfigure itself to listen to input port A.
- The FEC will then send a packet to CCU-2 instructing it to use the alternate output port B, therefore skipping module 3. Still, as module 4 is configured to listen to input port A the ring will appear to be faulty. The FEC ignores the problem and sends now a packet to CCU-4 (which will be received at port B) instructing it to use B as input port. This packet will also not get back to the FEC, but the ring is now configured properly and operation can restart normally. Subsequent packets will circulate normally in the ring.

3.4.2.1. Timing synchronization

Reconfiguration of the ring required recalibration of the timing synchronization paths, as the normal and the redundant paths are necessarily of different physical lengths.

3.5. Error Handing

Error handling can become quite complex on LAN networks. The following paragraphs attempt to collect a list of error possibilities with the relative handling procedure.

In the following discussion one has to take into account that each ring packet has always a forward and a return path, as all packets make always a complete tour of the ring and are always removed from the ring by the source of the packet itself.

The handling of error conditions is largely a responsibility of the software driver running on the FEC. For simplicity, the CCU hardware has to be able to recognize error conditions, but all correction actions are always taken in software by the driver running on the FEC.

3.5.1. Ring timeouts

Under normal conditions, the ring always contains one and only one token or data packet at any one time. Hardware failures could generate the condition that the token gets lost during a circulation of the ring, and the ring becomes not accessible. This situation must be recognized by the FEC which monitors the time between two successive tokens or packets. Should this time exceed a maximum timeout of 100 ms, the FEC generates a new token and inserts it in the ring.

When the FEC enters the mode used for recognizing and bypassing a faulty CCU module, all timeout recognition features must be disabled.

3.5.2. Packet lost

Occasionally a data packet could be lost in the ring (for instance due to a sync loss in the header). Under this condition, the emitting node will never receive the packet back and in principle would hang on an infinite wait loop and would also never re-emit a token. To avoid this situation, the FEC emits a new token after a maximum silence time-out as explained in the previous paragraph. When a CCU waiting for the return of its data packet receives instead a token, it should assume that the packet has gone lost. If this was an Alarm packet, the CCU is programmed with a retry counter and will try again. If this was a normal packet, the CCU may or may not have actually performed the associated action and the software driver will have the responsibility of handling this situation.

Another error mechanism is in the case of a read operation on a busy channel. In that case the originating packet is returned with the Data not Copied flag set. This means that the channel certainly skipped the command and that it has to be repeated.

3.5.3. Error present but packet length correct

First we will deal with all simple hardware generated errors, i.e. we assume that all packets are generated correctly at the source and just get corrupted by the transmission hardware on one or more bit, but the total bit length of the ring packet remains correct.

3.5.3.1.Traffic originated by the FEC

3.5.3.1.1.Command packet corrupted in the path between FEC and CCU

In this situation the destination (assuming that the destination field is not corrupted) receives a corrupted packet as seen by the CRC-16 calculation. The node controller does not dispatch the packet to any channel. The ER symbol in the EOF frame is set, CRC is not recalculated and the packet is returned back to the FEC. All other CCUs ignore the error and just forward the packet as normal. The FEC should assume a transient error and re-transmit the packet.

In the special case when the error has occurred exactly in the destination field, no CCU may see the packet, which will then return to the FEC without the ER symbol set in the EOF frame. The FEC will see the CRC-16 error, assume a transient error and re-transmit the packet.

3.5.3.1.2.Command packet corrupted in the return path between CCU and FEC

In this case the CCU has received a packet correctly, but this gets corrupted while traveling back to the FEC. In this case the EOF field was already set with the AR and DC symbols. The FEC recognizes a wrong CRC.

In the meantime the CCU may have already executed the command and, depending on the command, it might be ready to send back a reply packet.

At this point the FEC has to re-synchronize by software to the CCU.

3.5.3.2.Traffic originated by the CCU

3.5.3.2.1.Reply packet corrupted between CCU and FEC

A command packet was correctly sent to the CCU and the CCU sends back a reply packet, but this gets corrupted on its way to the FEC. The FEC recognizes the wrong CRC-16 and takes no action, it forwards the packet back to the CCU as normal. The CCU will eventually get the corrupted packet back; it will find out that the CRC-16 is wrong and re-transmit the packet.

3.5.3.2.2.Reply packet corrupted in return path between FEC and CCU

A command packet was correctly sent to the CCU and the CCU sends back a reply packet. The reply packet arrives properly at the FEC (which has set the AR and DC symbols in the EOF) but is gets corrupted on its way back to the CCU but this gets corrupted on its way to the FEC. The CCU finds out about the error with the CRC-16 check but it does not know whether the FEC has correctly received it (as the EOF field is unreliable).

The CCU re-transmits the same packet again. The CCU has to discard packets which are received twice.

3.5.3.3.Retransmission maximum count

To avoid dangerous situations in which one of the above conditions results in an infinite retransmission loop, both CCUs and FECs have a maximum retransmission count of four.

3.5.3.4.Error in Interrupt packets

The interrupt packets are originated in the CCUs. As for the other packets they can get corrupted on two sections of the ring tour: on the forward path to the FEC and on the return path to the source CCU. For both cases the CCU will find a wrong CRC-16 and will retry the transmission.

3.5.4. Complex error conditions

By allowing the malicious modifications of two or more bits in two correlated packets, one can build extremely complex error conditions in the protocol between the FEC and the CCUs. These conditions should be handled by software on the FEC and no special hardware support is foreseen in the CCU.

A careful examination of these complex error conditions is left to the implementation of a software driver for the FEC.

One relatively simple error condition occurs when a packet is completely corrupted, because of noise, new connections in the ring etc. All CCUs ignore anyway packets where the destination address does not correspond to their ring address, and check CRC-16 for all the ones with the correct destination address.

The FEC will drop all packets from the ring which are evidently corrupted, wait for a nominal delay time, and insert a clean token in the ring. Packets evidently corrupted are defined as those with:

?? Wrong SOF field: the FEC blocks forwarding of this packet until it recognizes idle characters, it then waits a 1 milliseconds delay and inserts a new token

?? Invalid source or destination field: The FEC keeps a list of the valid sources and destinations in the ring. Assuming a packet is being forwarded by the FEC with one of these two fields invalid, the FEC will stop the forwarding and abort transmission, recovering with a delay and a clean token.

?? Invalid length: The FEC is forwarding a packet but it realizes that the length of the packet is wrong. It will abort forwarding, wait for a nominal delay and insert a clean token.

A particular nasty error could occur when the FEC recognizes that the input line does never stabilize with idle characters. At this point a hard reset of the ring will be necessary.

4. CHANNELS IN CCU

4.1. General

Channels receive commands from the node controller and control the actions on the bus to which they connect as masters. These commands are contained in the data packet and therefore the data content in a given packet is interpreted by each channel differently.

The command sub-field (typically the third byte in a message to a channel) contains the code for the requested operation. Each channel has a set of valid commands as explained in this chapter. Channels receiving an invalid command do not execute any action and report the error condition to the node controller.

Upon reception of a command a channel performs the required operation on its interface and then, depending on the specifics of the command, can return a reply to the node controller as a data block which is then transmitted to the ring destination in a ring message packet.

The distinction between channels at the level of the FEC is performed essentially by software.

The following paragraphs specify the content of the packets for each type of channel.

In this implementation, the CCU does not support command queueing for the I2C channels. Only one command can be in execution at any one time.

4.2. Allocations of channels in the CCU

Each channel in a ring data packet (message) is identified by the first data byte in the data payload. The following table gives the internal address allocation for channels in the CCU:

Channel Number [Hex]	Function
0	CCU Node Controller
0x01?0x0F	Reserved
0x10?0x1F	I2C channels (16 identical)
0x20	Broadcast channel for I2C
0x21?0x2F	Reserved
0x30?0x33	PIO channels (4 identical)
0x34?0x3F	Reserved
0x40	Memory channel
0x41?0x4F	Reserved
0x50	Trigger distribution channel
0x51?0x5F	Reserved
0x60	JTAG Master channel
0x61?0xfc	Reserved
0xfe?0xff	Special Interrupt channels

Figure 15 Channel number allocation

All I2C channels behave in a similar way, by accepting the command and executing an operation on their ports. Only the broadcast I2C channel number 0x20 is special, as an operation sent to this channel is actually copied to all I2C channels in the CCU. Broadcast operations to I2C assume that write “posting” is used, i.e. no acknowledgement is required.

4.3. CCU controller

The CCU controller is a dedicated logic block inside each CCU which is needed mainly for network and internal channels supervision. The CCU controller is reachable with the same protocol used to transfer data to the other port channels.

The following eight bit registers are defined in the controller:

Name	Function
CRA	Control register A
CRB	Control register B
CRC	Control register C
CRD	Control register D
CRE	Control register E
SRA	Status register A
SRB	Status register B
SRC	Status register C
SRD	Status register D
SRE	Status register E
SRF	Status register F

Figure 16 Control and Status registers in CCU Controller

Control registers are all read/write registers. They can be read back after a write operation to verify their content. Status registers are read-only registers, as they are set typically by hardware inside the CCU.

4.3.1.1. Control Register A

Control register A is a general control register for the CCU. It contains control bits which are relevant for the operation of all channels in the CCU.

The following bits are defined:

Bit	Name	Function	Comment
0-4	Reserved		
5	EXTRES	Generates external reset	Writing a "1" to this bit generates a 10 microseconds reset pulse on the ResetOutZ pin. This bit is always read back as "0". Read as a "1" during the reset time.
6	CLRE	Clear error	Writing a "1" to this bit clears all internal error flags. The bit is always read back as "0"
7	RES	Reset All Channels	Cold reset for all channels, but not for the node controller, writing a "1" to this bit generates a general reset. This bit is always read back as "0".

Figure 17 Node controller control register A

4.3.1.2. Control Register B

This register controls the operation of the special ALARM1-4* lines and of the retry counters as defined below:

Bit	Name	Function	Comment
0	ENAL1	Enable ALARM1* interrupt	This bit is initialised to "0" after reset.
1	ENAL2	Enable ALARM2* interrupt	This bit is initialised to "0" after reset.
2	ENAL3	Enable ALARM3* interrupt	This bit is initialised to "0" after reset.
3	ENAL4	Enable ALARM4* interrupt	This bit is initialised to "0" after reset.
5-4	RTRY	Retry count for Alarm interrupts. The retry is activated if a clear has not been received after a 1 millisecond timeout interval.	These two bits determine how many times a CCU will try to retransmit a packet, if this came back with a wrong CRC-16 after a ring circulation. 00 – none 01 – once 10 – twice 11 – four times These bits are initialised to "11" after reset.
7-6		Reserved	

Figure 18 Control register B in node controller

The Enable Alarm bits are cleared automatically. They cannot be cleared by writing a “0”.

When one of the external ALARM lines is activated, the node controller sends a special interrupt message packet to the FEC. The ALARM lines are level sensitive, as is detailed in the paragraph 4.10 Handling of Alarms.

The interrupt packet is a special packet, as it is not solicited by the FEC, and therefore can not contain a meaningful transaction number (a “00” is sent in the transaction number field).

The format of the packet is shown below:

Action	CMD [hex]	Command Packet Format
ALARM Interrupt	none	**R:** CH#+ “00” + Alarm_Number

Figure 19 Interrupt packet format

In this special case, the channel number is 0xfe.

4.3.1.3. Control Register C

Control register C is used to configure the redundancy features of the CCU.

Bit	Name	Function	Comment
0	ALTIN	Writing a “1” in this bit selects Port B as the current Input Port	This bit is initialised to “0” after reset. Writing a “1” into this bit also generates a “1” in the PLLSEL output pin.
1	SSP	Selects the alternate Output Port (B).	This bit is initialised to “0” after reset. The signal DoutA/B on the non-selected port produces a continuous stream of idle characters.
7-2	Unused	Reserved	

Figure 20 Definition of control register C (redundancy control)

4.3.1.4. Control Register D

Control register D contains the broadcast class for the CCU. The broadcast class is a 7 bit number determining which CCUs respond to a broadcast operation

Bit	Name	Function	Comment
6-0	BCLS	Broadcast Class	
7	Unused	Reserved	

Figure 21 Control register D bit allocation

4.3.1.5. Control Register E

Control register E is a 24 bit wide register and contains channel enable bits

Bit	Name	Function	Comment
23		Unused	
22	ENJTAG	Enable JTAG controller	
21	ENTRG	Enable trigger controller	
20	ENMEM	Enable memory controller	
19:16	ENPIO	Enable PIO controllers	
15:0	ENI2C	Enable I2V controllers	

Figure 22 Control register D bit allocation

CAREFUL:

Due to a small bug in the logic, writing the control register E requires care. The following sequence MUST be followed:

WR Control Reg A

WR Control Reg B

WR Control Reg C

WR Control Reg D

WR Control Reg E

RD Control Reg A

RD Control Reg B

RD Control Reg C

RD Control Reg D

RD Control Reg E

4.3.1.6. Status Register A

The node controller status register A is defined as follows:

Bit	Name	Function
0	PED	Error bit, set if any CRC error was detected in a passing packet. This error is cleared with a general node controller reset or clear error.
1	IE	Internal error, set if any error inside the node controller state machines of the CCU was detected. This error is cleared with a general node controller reset or clear error.
2	ALSET	This bit is set if one or more of the ALARM* inputs to the CCU are currently active (low). The bit is cleared by de-activating all external alarm lines.
3	CCUPERR	This bit is set if a parity error is detected in the internal CCU registers. This error is cleared with a general node controller reset or clear error.
4	CHAPERR	This bit is set if a parity error is detected in any of the channels. This error is cleared with a general node controller reset or clear error.
5	ILLSEQ	This bit is set if an illegal sequence of tokens is detected on the serial line. This error is cleared with a general node controller reset or clear error.
6	INVCMD	Invalid command. This bit is set when the node controller receives a correct ring message but the command is invalid. This error is cleared with a general node controller reset or clear error.
7	GE	This bit is the global OR of all error bits generated in any channel in the CCU.

Figure 23 Node controller status register A

4.3.1.7. Status Register B

The status register B is an 8 bit register containing the transaction number (TR#) of the last correctly received command for any of the CCU channels. This is necessary to support the case when a packet traveling through the ring gets corrupted after having reached the destination and the FEC has to find out whether the packet had already reached its destination or not.

4.3.1.8. Status Register C

This register reports the status of the redundancy configuration register.

Bit	Name	Function	Comment
0	ACTPRT	This bit reports the currently active input port. (This is the status pf the PLLSEL output pin)	This bit is initialised to "0" after reset. When the current active input port is A this bit contains a "0", when the current active input port is B this bit contains a "1".
7-1	Unused	Reserved	

Figure 24 Definition of status register C (redundancy control)

4.3.1.9. Status Register D

The status register D contains the 8 bit address of the source field for the last ring message addressed to this CCU.

4.3.1.10.Status Register E

The status register E is a 24 bit register and contains the following:

High Byte	Middle Byte	Low Byte
23-18 Unused 17 Trigger Busy 16 Memory Busy	15-8 I2C Busy	7-0 I2C Busy

Figure 25 Data portion of packet

4.3.1.11.Status Register F

The status register F is a 16 bit wide register and contains the parity error counter bits[15:0]. This counter is cleared by Clear Error from control register A

4.3.1.12.Command codes ([3])

The following table summarizes the commands accepted by the node controller for operations on its registers.

Command	CMD [hex]	Command and Reply Formats	Operation
Write control register A	0x00	**C:** CH#+TR#+CMD+DW **R:** none	The control register A is written with a byte
Write control register B	0x01	**C:** CH#+TR#+CMD+DW **R:** none	The control register B is written with a byte
Write control register C	0x02	**C:** CH#+TR#+CMD+DW **R:** none	The control register C is written with a byte

[3] The symbols used in the command tables are explained in Appendix 1 – Command symbols

Write control register D	0x03	**C:** CH#+TR#+CMD+DW **R:** none	The control register D is written with a byte
Write control register E	0x04	**C:** CH#+TR#+CMD+DW24 **R:** none	The control register E is written with a byte
Read Control register A	0x10	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR	Read back control register A
Read Control register B	0x11	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR	Read back control register B
Read Control register C	0x12	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR	Read back control register C
Read Control register D	0x13	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR	Read back control register D
Read Control register E	0x14	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR24	Read back control register E
Read Status register A	0x20	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR	Send back a packet with content of status register A
Read Status register B	0x21	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR	Send back a packet with content of status register B
Read Status register C	0x22	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR	Send back a packet with content of status register C
Read Status register D	0x23	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR	Send back a packet with content of status register D
Read Status register E	0x24	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR24	Send back a packet with content of status register E
Read Status register F	0x25	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR16	Send back a packet with content of status register F

Figure 26 Commands for node controller

4.4. Redundancy control

The design of the redundancy scheme for the CCU must take into account that the same clock used by the CCU to synchronize to the incoming data must also be distributed to the electronics on the front end modules again for clocking and trigger distribution. It must also be remembered that the clock signal used in the ring is coded with a missing pulse to indicate a trigger signal.

In addition, as the clock and data signals used in the ring protocol are transmitted through AC coupled lines, they must be running on both ports at all time. Under normal operation the CCU uses the input port A (ClkInA and DinA signals) to receive clock and data.

4.4.1. LVDSMUX

To provide a proper clock to the front end electronics, a special ASIC is used together with the CCU. This ASIC is called the LVDSMUX. This ASIC has two functions:

- it selects one of the two incoming clock lines and distributes it to the front end components
- it converts signals from LVDS levels from/to CMOS levels as necessary.

A block diagram of the LVDSMUX is shown in the following figure.

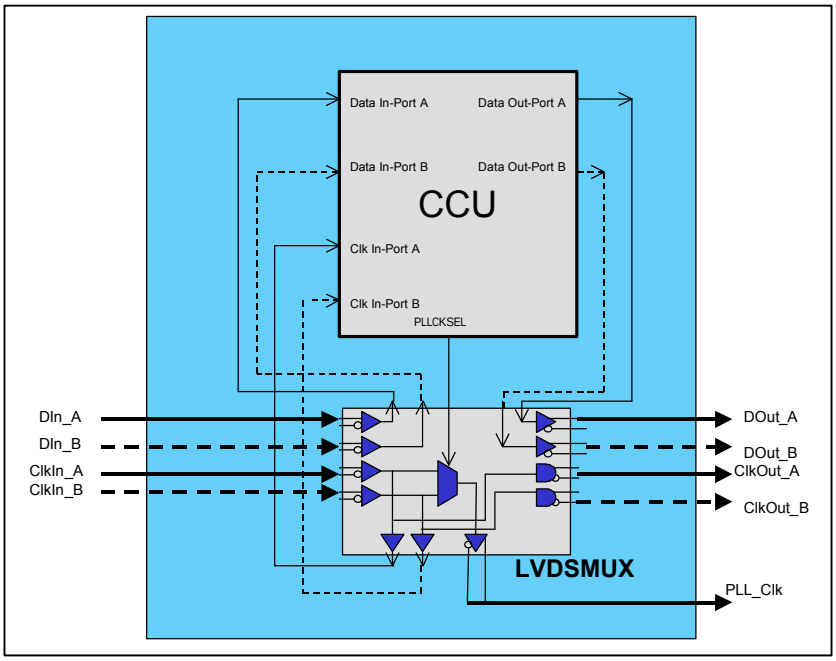


Figure 27 CCU Redundancy cabling scheme

The CCU receives two continuous clock streams on ports A and B. The inactive port always receives and sends idle patterns without any token.

When the alternate port B is used by the CCU ring, the timing of the clock signal clearly changes, as the normal and alternate ring are of different length. This requires a readjustment of the clock phase to the front end module. To control which clock has to be used by the front end electronics, the CCU can use the PLLSEL output to force the LVDSMUX to select the CLKB signal.

When switching between the two different clock sources, the CCU should pay attention not to generate spurious glitches which could generate problems in the front end electronics.

Notice that the CCU uses as its main clock a signal which has sometimes missing pulses, therefore the serial data are not allowed to change when a clock pulse is missing. To be able to recover the trigger pulse, a PLL must be used in parallel to the CCU to detect the trigger signals.

4.4.2. Input port control

After a power up or an external reset the CCU operates normally with A as active input port. Port B receives idle patterns. To switch to the alternate port B the CCU must receive a proper and well formed command through its B port. This command consist of writing a “0x01” into the Register C in the Node Controller. At his point, any activity coming from the A port is ignored. To switch back to the A port, the CCU must receive a proper and well formed command from the A port, i.e. write a “0” into the LSB of the Register C in the Node Controller.

Notice that during port switching the CCU must also change its input clock source. This can only be done after a complete packet is received. To be able to perform this operation the CCU has two completely and fully independent clock sources reaching the node controller. The CCU contain internal logic capable if switching its own input clock source without generating glitches in its operation. During port switching, the packet sent by the FEC is not returned.

During the time an input port is inactive, any other packet received which is not the port switch command itself is simply ignored.

4.4.3. Output port control

After a power up or an external reset the CCU operates normally with port A as active output port. The operation of switching output port to B is achieved by writing a “1” into bit 1 of the Control Register C in the Node Controller. The command must come from the currently active input port. To switch back to Output Port A, the bit 1 in register C has to written with a “0”. Notice that it is perfectly possible for a CCU to work with both input and output ports B.

4.5. I2C Channel

The I2C interface implements normal 7 bit addressing, long 10 bit addressing I2C transactions and extended RAL mode transfers. The operations performed by the I2C interface are:

- Single byte read-write with normal 7 bit I2C addressing
- single byte read-write with 7+ 8 bit address (indirect address in RAL([4]) mode)
- extended I2C with 10 bits addressing
- four bytes read-write mode with normal 7 bit I2C addressing

4.5.1.1. I2C interface registers

Several registers control the operation of the I2C interface and are implemented in each of the 16 I2C channels.

Name	Comment
CRA	Control register A
MSK	Mask register for logical operations
SRA	Status register A
SRB	Status register B
SRC	Status register C
SRD	Status register D

Figure 28 Control and Status registers in I2C channel

4.5.2. I2C Control registers

The Control register A in the I2C interface is defined as follows:

[4] Special I2C mode defined to access registers in APV chips.

Bit	Name	Function
1-0	SPEED	Denotes the speed of operation of the I2C interface (SCL clock rate): 00 – 100 kHz 01 – 200 kHz 10 – 400 kHz 11 – 1 MHz
2	Reserved	0
3	Reserved	0
4	Reserved	0
5	EBRDCST	Enable acceptance of broadcast operations. Once set to "1" this bit enables the channel to accept I2C broadcast operations. This bit is initialised to "0" after reset.
6	FACKW	Force acknowledge for write or RMW operation Write operations and RMW operations which do not generate errors are not acknowledged. Forcing this bit to "1" generates an acknowledgement packet. This bit is cleared at reset.
7	Unused	

Figure 29 Control register A in I2C interface

4.5.3. Logical mask register.

This register can be written with an 8-bit value which is used during logical operations on the I2C bus. These operations are of the type read-modify-write and can only be executed in single-byte mode.

Three basic operations are allowed:

?? Logical AND

?? Logical OR

?? Logical XOR

And are performed in the following way:

1. the I2C interface reads a byte from the specified address
2. a logical operation is performed with the mask register value
3. the result is written back into the I2C address
4. the original value is returned to the FEC (if CRA[6] is set).

4.5.4. I2C Status Registers

Several registers are used to report the status of the I2C channel

4.5.4.1. Status Register A

This register contains the following information:

Bit	Name	Function
1-0	Reserved	
2	SUCC	This bit is set when the last I2C transaction was successfully executed.
3	I2CLOW	This bit is set to '1' is the I2C master port finds that the SDA line is pulled low ("0") before initiating a transaction. If this happens the I2C bus is probably broken. The bit represents the status of the SDA line and cannot be reset.
4	Reserved	
5	INVCOM	This bit is set if an invalid command was sent to the I2C channel. The bit is cleared by a channel reset.
6	NOACK	This bit is set if the last operation has not been acknowledged by the I2C slave acknowledge. This bit is set/reset at the end of each I2C transaction
7	GE	This bit is set if any error has occurred on the I2C channel and is cleared only by a channel reset command

Figure 30 Status register A in I2C interface

4.5.4.2. Status Register B

Status register B contains the number of the last correctly executed transaction (TR#). This register is overwritten after every I2C transaction except read and write operations to/from the control and status register.

This register is cleared at reset.

4.5.4.3. Status Register C

Status register C contains the number of the last incorrectly executed transaction (TR#). The register is cleared at reset.

4.5.4.4. Status Register D

This register contains the code of the last command sent to this I2C channel, including control and status commands.

4.5.5. I2C Commands

The following table summarizes all the commands accepted by the I2C channels.

Command	CMD [hex]	Command and Reply Format
Single byte write normal mode	0x00	**C:** CH#+TR#+CMD+A7+DW **R:** none or CH#+TR#+ACK
Single byte read normal mode	0x01	**C:** CH#+TR#+CMD+A7 **R:** CH#+TR#+DR+ACK
Single byte write extended mode	0x02	**C:** CH#+TR#+CMD+A[9:8]+A[7:0]+DW

		R: none or CH#+TR#+ACK
Single byte read extended mode	0x03	**C:** CH#+TR#+CMD+A[9:8]+A[7:0] **R:** CH#+TR#+DR+ACK
Single byte write RAL mode (same CMD code as extended mode)	0x02	**C:** CH#+TR#+CMD+A7+ACR+DW **R:** none or CH#+TR#+ACK
Single byte read RAL mode	0x11	**C:** CH#+TR#+CMD+A7+ACR **R:** CH#+TR#+DR+ACK
RMW-AND in normal mode	0x80	**C:** CH#+TR#+CMD+A7 **R:** none or CH#+TR#+DR+ACK
RMW-OR in normal mode	0x81	**C:** CH#+TR#+CMD+A7 **R:** none or CH#+TR#+DR+ACK
RMW-XOR in normal mode	0x82	**C:** CH#+TR#+CMD+A7 **R:** none or CH#+TR#+DR+ACK
RMW-AND in extended mode	0x83	**C:** CH#+TR#+CMD+A[9:8]+A[7:0] **R:** none or CH#+TR#+DR+ACK
RMW-OR in extended mode	0x84	**C:** CH#+TR#+CMD+ A[9:8]+A[7:0] **R:** none or CH#+TR#+DR+ACK
RMW-XOR in extended mode	0x85	**C:** CH#+TR#+CMD+ A[9:8]+A[7:0] **R:** none or CH#+TR#+DR+ACK
Write Control register A	0xf0	**C:** CH#+TR#+CMD+DW **R:** none
Read Control register A	0xf1	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR
Read Status register A	0xf2	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR
Read Status register B	0xf3	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR
Read Status register C	0xf4	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR
Read Status register D	0xf5	**C:** CH#+TR#+CMD

		R: CH#+TR#+DR
Write Mask register	0xf6	**C:** CH#+TR#+CMD+DW **R:** none
Read Mask register	0xf7	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR
I2C channel reset	0xff	**C:** CH#+TR#+CMD **R:** none This command bypasses all pending commands in the channel and is executed as a hardware reset on the specified channel only.

Figure 31 Commands for I2C channel

4.6. Memory Bus Channel

The Memory Data bus implemented on the CCU can address a 64KB memory through a 16 bit address and 1 byte wide data interface. It can perform single and multiple byte read-write operations as on a normal byte-wide memory device. The operations foreseen are:

- single byte read-write to address
- multiple (up to 2K) bytes read-write to address with address auto-increment
- read-modify-write single byte to address with mask

Several registers control the operation of the memory bus channel.

Name	Comment
CRA	Control register A
MSK	Mask register for logical operations
SRA	Status register A
WIN1L	Window 1 base address
WIN1H	Window 1 high address
WIN2L	Window 2 base address
WIN1H	Window 2 base address

Figure 32 Registers in memory bus channel

To simplify the design of simple peripheral devices connected to the CCU memory channel, the CCU provides pre-decoding of up to two memory ranges defined in the window registers.

4.6.1. Memory Bus Control registers

The Control register is defined as follows:

Bit	Name	Function
1-0	SPEED	Denotes the speed of operation of the memory interface (CS* is low for half of the bus clock cycle time): 00 – 1 MHz 01 – 4 MHz WARNING: This is the only valid mode for block transfer operations 10 – 10 MHz 11 – 20 MHz. Initialised to "00" after power up and reset.
2	ENW1	Enable operation of decoder for window 1. Initialised to "0" after power up and reset.
3	ENW2	Enable operation of decoder for window 2. Initialised to "0" after power up and reset.
4-7		Reserved

Figure 33 Control register A in memory channel

4.6.2. Memory bus Window registers

The window registers 1 and 2 provide a base address and a high address respectively for two memory ranges that external devices might be using without the need of an external decoder.

Register	Function
WIN1L	This 16 bit register contains the base address for window 1. Reset writes a 0 to this register. This value is the lowest accessible address for window 1.
WIN1H	This 16 bit register contains the high address for window 1. Reset writes a 0 to this register. This value is the highest accessible address for window 1.
WIN2L	This 16 bit register contains the base address for window 2. Reset writes a 0 to this register. This value is the lowest accessible address for window 2.
WIN2H	This 16 bit register contains the high address for window 2. Reset writes a 0 to this register. This value is the highest accessible address for window 2.

Figure 34 Memory window registers in memory channel

To logic in the channel activates the external CSj* pin (j=1,2) whenever a memory operation has an addressed in the range covered by the corresponding window registers; i.e. the CS* signals are active when:

CS1* active when : WIN1L <= address =< WIN1H & ENW1* & ENCH

CS2* active when: WIN2L <= address =< WIN2H & ENW2* & ENCH

4.6.3. Logical mask register.

This register can be written with an 8-bit value which is used during logical operations on the memory bus. These operations are of the type read-modify-write. Three basic operations are allowed:

?? Logical AND

?? Logical OR

?? Logical XOR

And are performed in the following way:

1. the memory interface reads a byte from the specified address
2. a logical operation is performed with the mask register value
3. the result is written back into the memory address while the original value is returned to the FEC.

4.6.4. Memory Bus Status Registers

Bit	Name	Function
0-4		Reserved
5	INVCOM	Invalid command received. Cleared by channel reset.
6	INVADD	Invalid address. A command with a memory write operation was received with an address outside both window ranges. Cleared by channel reset.
7	GE	Global error. Logical OR of all error conditions in the interface

Figure 35 Status register in memory channel

4.6.5. Memory Bus Commands

The commands used for operating the Memory Bus interface are defined in this paragraph.

Action	CMD [hex]	Command Packet Format
MBUS Reset channel	0xFF	**C:** CH#+TR#+CMD **R:** none
Write control register A	0x01	**C:** CH#+TR#+CMD+DW **R:** none
Read Control Register A	0x02	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR
Write WIN1L register	0x03	**C:** CH#+TR#+CMD+DW16 **R:** none
Read WIN1L register	0x04	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR16 CAREFUL: Due to a bug in the CCU25-1 logic, reading this register generates an unwanted parity error
Write WIN1H register	0x05	**C:** CH#+TR#+CMD+DW16 **R:** none
Read WIN1H register	0x06	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR16 CAREFUL: Due to a bug in the CCU25-1 logic, reading this register generates an unwanted parity error
Write WIN2L register	0x07	**C:** CH#+TR#+CMD+DW16 **R:** none
Read WIN2L register	0x08	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR16 CAREFUL: Due to a bug in the CCU25-1 logic, reading this register generates an unwanted parity

		error
Write WIN2H register	0x09	**C:** CH#+TR#+CMD+DW16 **R:** none
Read WIN2H register	0x0a	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR16 CAREFUL: Due to a bug in the CCU25-1 logic, reading this register generates an unwanted parity error
Write Mask Register	0x0b	**C:** CH#+TR#+CMD+DW **R:** none
Read Mask Register	0x0c	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR
Read Status Register	0x0f	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR
Single Byte Write to memory	0x10	**C:** CH#+TR#+CMD+AH+AL+DW **R:** none
Single Byte Read from memory	0x11	**C:** CH#+TR#+CMD +AH+AL **R:** CH#+TR#+DR
Multiple Byte Read from Memory	0x14	**C:** CH#+TR#+ CMD+ AH+AL +LENH+LENL [5] **R:** CH#+TR#+LENH+LENL+DR[n]
Multiple Byte Write to Memory	0x15	**C:** CH#+TR#+ CMD+AH+AL+D[n] [6] **R:** none
Single Byte RMW-AND	0x20	**C:** CH#+TR#+CMD+AH+AL **R:** CH#+TR#+DR
Single Byte RMW-OR	0x21	**C:** CH#+TR#+CMD+AH+AL **R:** CH#+TR#+DR
Single Byte RMW-XOR	0x22	**C:** CH#+TR#+CMD+AH+AL **R:** CH#+TR#+DR

Figure 36 Memory Bus channel Commands

The Memory address is divided into the two bytes AH and AL, for the high and low part of the address respectively. The block length (max 2 K) is also divided into two bytes, LENH and LENL.

When operated in the 2K block transfer mode, the CCU can check the CRC on the incoming packet only at the end of the packet itself. As no internal buffering is foreseen, it may be possible that due to a bit error the packet data are corrupted and therefore wrong data are written to the memory. In this case the CCU still signals the error condition through the bit 0 in Status Register A in the Node Controller.

[5] Careful, the value of LENH-LENL has to be augmented by 2.
[6] Same as above

4.7. Parallel I/O bus

The parallel I/O bus channel is an adapter similar functionally to a Motorola PIA interface, allowing parallel connections with programmable direction in groups of 8 bits. Four independent byte PIO adapter channels are available in the CCU.

The following registers control the operation of each PIO channel:

Name	Function
GCR	General control register
SR	Status Register
DDRA	Data direction register for Port
DREG	Data register

Figure 37 Registers in Parallel IO bus

4.7.1. Registers in PIO channel

The functions of the registers are detailed in the following paragraphs.

4.7.1.1. General Control Register in PIO

The PIO Control register is defined as follows:

Bit	Name	Function
1-0	STRW	Denotes the width of the strobe signals on the port 00 – 1000ns 01 – 500ns 10 – 200ns 11 – 100ns. Initialised to "00" after power up and reset.
2	CLRINT	Writing a "1" to this bit generates a clear of the pending interrupt
3	STRB	Strobed operation. When written with a "0" this bit determines that the input bits are non-strobed, i.e. the lines read on the port are read directly. When written with a "1" a strobe signal on PIOISTR is necessary to transfer the status of the lines to an internal register.
4	STROUTP	Determines polarity of output strobe on port A. When written with "0" the polarity produced for a write operation to the PIO port is positive, a "1" determines a negative polarity.
5	ENINTA	Enables generation of Interrupt message to FEC on reception of STRINP
6	STRINP	Determines polarity of input strobe on port. When written with "0" the polarity expected for a write operation to the PIO port from external logic is positive, a "1" determines a negative polarity.
7	Reserved	

Figure 38 General Control Register in PIO channel

4.7.1.2. Status Register in PIO

Bit	Name	Function
0	INT	An interrupt was generated by a strobe on Port. Cleared by writing a "1" to the CLR bit in GCR.
4-1	Reserved	
5	INVCOM	Invalid command received. Cleared by channel reset. Does not generate a GE bit (see below)
6	Reserved	
7	GE	Global error. Logical OR of all error conditions in the PIO interface

Figure 39 Status register in PIO channel

4.7.1.3. Data Direction Register Port

The data direction register defined the direction of the data transfer in the corresponding port. Writing a "0" in bit 'j' in this register determines the direction of the corresponding data bit in the port as input. Writing a "1" determines the bit on the port as output. The register is initialized after power on and reset as all zeros, i.e. with all I/O pins configured as inputs.

4.7.2. Commands for PIO channel

The following commands are defined for the PIO channel:

Action	CMD [hex]	Command Packet Format
PIO Reset channel	0xFF	**C:** CH#+TR#+CMD **R:** none
Write General Control Register	0x01	**C:** CH#+TR#+CMD+DW **R:** none
Read General Control Register	0x02	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR
Read Status Register	0x0F	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR
Write DDR	0x05	**C:** CH#+TR#+CMD+DW **R:** none
Read DDR	0x22	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR
Write Data Register	0x10	**C:** CH#+TR#+CMD+DW **R:** none
Read Data Register	0x11	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR

Figure 42 Commands for PIO channel

4.7.3. Interrupt generation

Activating the four input strobe signals on the ports A, B, C and D can generate an interrupt message packet to be sent to the FEC (at address 0). To control the generation of the interrupt packets, four control bits (ENINTA, ENINTB, ENINTC and ENINTD) in the control registers enable this feature on each port separately. The interrupts are generated on the leading edge of the input strobe. As soon as an interrupt packet is generated, further packets are disabled until the Clear Interrupt bit is written in the General Control Register.

The interrupt packet is a special packet, as it is not solicited by the FEC, and therefore can not contain a meaningful transaction number (a “00” is sent in the transaction number field).

The format of the packet is shown below:

Action	CMD [hex]	Command Packet Format
PIO Interrupt	none	**R:** CH#+ “00” + INT

Figure 43 Interrupt packet format

In this special case, the channel number is 0xff”.

4.8. Trigger distribution

The trigger distribution logic block is a special channel dedicated to the distribution of the three special trigger (ST1-ST4) signals. The special trigger signals are generated form the input clock and T1 to the CCU and are distributed to the trigger destination front end ASICs. The ST1-ST4 signals can be delayed a number of cycles by programming a delay value in Control register A.

The following registers are allocated to the trigger channel:

Name	Comment
CRA	Control register A
CRB	Control register B
SRA	Status register A
4 x TCNT	32 bit trigger counters

Figure 44 Trigger distribution control and status registers

4.8.1.1. Control register A

The control register is a read-write register and is defined as follows:

Bit	Name	Function
0-3	Unused	
7-4	EC	This bit enables counting in the trigger counting register. It is initialised to "0" after a power on or reset.

Figure 45 Control register A in trigger distribution logic

4.8.1.2. Control register B

The control register is a read-write register and is defined as follows:

Bit	Name	Function
3:0	STD<3:0>	ST delay 3:0
7-4	Unused	

Figure 46 Control register B in trigger distribution logic

4.8.1.3. Status register

The status register contains the following bits:

Bit	Name	Function
0		Reserved
1-4		Reserved
5		Reserved
6	INVC	Invalid command. Set when the trigger distribution channel receives and invalid command. Cleared by a reset to the channel.
7	ERR	Global error. Cleared by a reset to the channel.

Figure 47 Status register A in trigger distribution logic

4.8.1.4. Special trigger signals

The special trigger signals are generated from the T1 input with a default delay of three periods and a length of one period. The bits 3-6 in Control register control the delay of the ST signals from their default position.

:

Name	Comment
ST1	Generated by T1 sequence 100
ST2	Generated by T1 sequence 110
ST3	Generated by T1 sequence 101
ST4	Generated by T1 sequence 111

Figure 48 Generation of the ST pulses

4.8.1.5. Trigger counter register

The trigger counter register is a 32-bit register counting the number of level 1 (L1) triggers received by the CCU. The counter is reset when the T1 sequence 101 is encountered. This is the same sequence that resets the APVs.

4.8.1.6. Commands on the Trigger Distribution Channel

The following commands are allowed for the trigger distribution channel:

Command	CMD [hex]	Command and Reply Format
Write control register A	0x00	**C:** CH#+TR#+CMD+DW **R:** none
Read Control register A	0x01	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR
Write control register B	0x03	**C:** CH#+TR#+CMD+DW **R:** none
Read Control register B	0x04	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR

Read Status register	0x02	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR
Read Trigger counter (i)	0x0(5-8)	**C:** CH#+TR#+CMD **R:** CH#+TR#+DR32
TRIG Reset channel	0xFF	**C:** CH#+TR#+CMD **R:** none

Figure 49 Commands for trigger distribution logic channel

4.9. JTAG Channel

A simplified JTAG master channel is implemented in the CCU. The JTAG master generates three signals i.e. TCK, TMS and TDO, where TCK is the scan chain clock, TMS the mode control bit for the JTAG slave state machines and TDO the serial data sent to the scan chain. Data is returned on the TDI line. The transitions on TMS and TDO takes place upon the negative edge of TCK. TDI is sampled on the positive edge of TCK. There is no autonomous JTAG controller as such with a command structure, the protocol is simply implemented in software. A JTAG packet consists of the normal header information like Destination, Source, Length, ChannelNumber and TransactionNumber then follows the Data and CRC16. The channel number for the JTAG is x60, the data part contains the TMS and TDO information. Each byte of data contains four bits of TMS and four bits of TDO. This permits sending JTAG sequences of up to a length of ~8K bits. This is accomplished by packing 3 bits each of TMS and TDO per data byte.

TMS[11:8]	TDO[11:8]	TMS[7:4]	TDO[7:4]	TMS[3:0]	TDO[3:0]

Figure 50 Packing of data bytes

As the serial data is shifted around the ring it is at the same time shifted out onto the TMS and TDO (MSB first out) lines at half the frequency. For each byte four clockpulses (TCK) are generated which are used to shift out the contents of the TMS and TDO registers, MSB first. The incoming serial data from the JTAG chain (TDI) is collected in an other four bit register and fed back into the passing data packet on the token ring. This is not a standard procedure on the token ring, but it avoids having large storage buffers on chip. To be able to do this trick the last two byte of the ring data packet should be empty. The returned data is in fact delayed by two bytes, so here the first data bytes are empty and the last one is filled by the last TDI nibble. The return data bytes only contains four bits of TDI per byte in bits [3:0]. The CRC16 is recalculated on the fly so it corresponds to the data of the outgoing packet.

Byte 0	Byte 1	Byte 2	Byte 3	Empty	Empty

Figure 51 Incoming data stream

Empty	Empty	Empty TDI 0	Empty TDI 1	Empty TDI 2	Empty TDI 3

Figure 52 Outgoing data stream

4.9.1. JTAG command

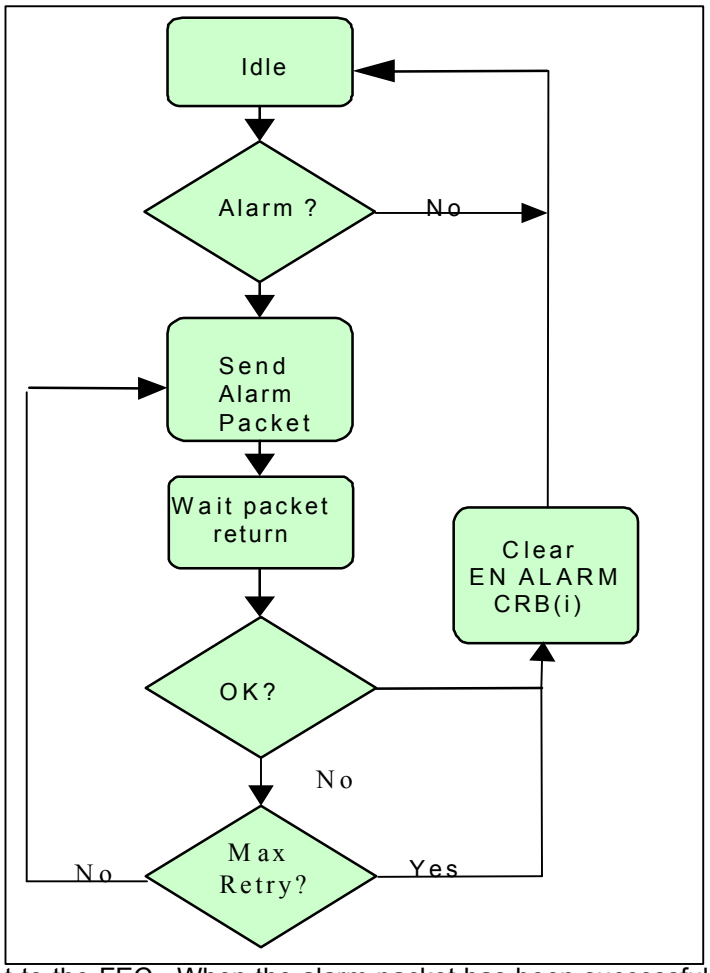
The following table summarizes all the commands accepted by the JTAG channel.

Command	Command and Reply Format
Write JTAG string	**C:** (Length –4) x D8 + x0 +x0 **R:** x0 + x0 + (Length –4) x D8

CAREFUL: due to s bug in the design, if bit (7:3) in the packet are equal to zero, a CRC error is generated even if in reality there was no error. The data are still accepted by the CCU.

4.10. Handling of Alarms

The CCU provides the basic hardware necessary to handle unsolicited alarm conditions from chips in the front end, such as the APV etc. As described in paragraph 4.3.1.2 (Control Register B), a control register in the CCU node controller enables this function. Enabled alarms generate a special packet from the CCU to the FEC. It is then a function of the software running on the FEC to find the source of the alarm conditions and reset it. The ALARM lines are level sensitive, the CCU node controller will probe the lines and send out an



alarm packet to the FEC.. When the alarm packet has been successfully transferred, or max retry condition reached, the corresponding enable bit in CRB is cleared. The following flow chart explains how the alarm mechanism works.

Figure 53 Alarm Handling

This mechanism insures that even if the first alarm packet got lost or corrupted due to network problems, the FEC will eventually always receive a notification of the alarm.

5. TOKEN RING INTERFACE

This chapter defines the operation of the token ring interface, including functional and electrical characteristics.

The token ring interface supports the protocol defined in paragraph 3.1, Token-ring like protocol.

The circulating messages consist of variable length data packets. To allow proper transmission of these packets on high speed data lines a coding technique commonly used in communication networks is used as defined in the next section.

5.1. Coding

The optical (and also the embedded electrical) network will be implemented with two wires:

- one for clock (and T1)
- one for data

A full duplex transmission (like the one between the FEC and the CCU will then require four wires. Each CCU will have two incoming wires and two outgoing ones (neglecting redundancy).

In order to allow easy implementation of the support hardware, data on the network will be explicitly synchronized using the clock line. No data and clock recovery is necessary from the data line.

5.1.1. Clock encoding

The FEC generates a general 40 MHz clock distributed to all CCUs in a ring. The FEC is responsible to properly align clock and data at the source of the ring. Each CCU will also regenerate both signals with the appropriate timing.

The scheme used to transmit the clock is shown in the figure below.

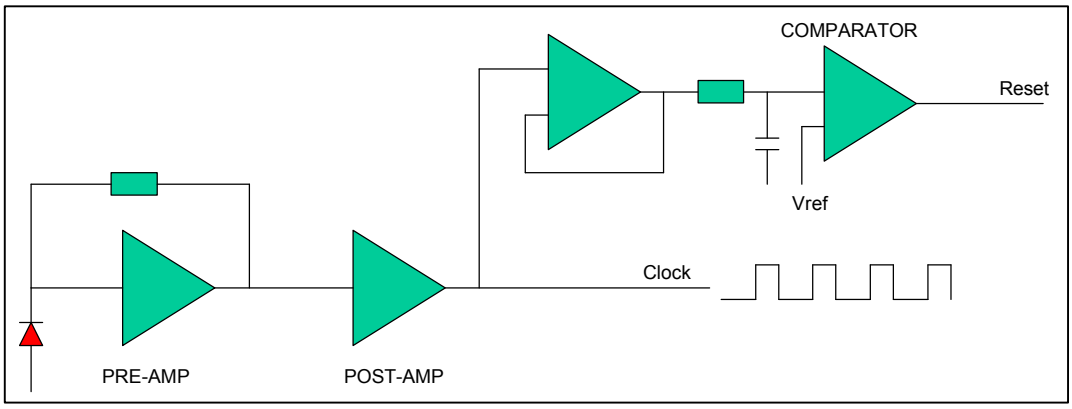


Figure 54 Reset generator from optical link

(What are the problems if the link is AC coupled ???)

5.1.2. Data encoding

Data on the data line are qualified by the rising edge of the clock signal. In addition, the data bits on the data line will be transmitted using 4bit to 5bit encoding, using a NRZI (Non Return to Zero with Invert 1 on change) signaling scheme as follows:

4 bit Binary	Hex Value	5 bit Symbol
0000	0	11110
0001	1	01001
0010	2	10100
0011	3	10101
0100	4	01010
0101	5	01011
0110	6	01110
0111	7	01111
1000	8	10010
1001	9	10011
1010	A	10110
1011	B	10111
1100	C	11010
1101	D	11011
1110	E	11100
1111	F	11101

Figure 55 4B/5B encoding

This scheme has been chosen because it requires very limited hardware resources an can be easily implemented in hardware. The worst case DC unbalance deriving from the usage of this coding is estimated to be about 10%.

The control symbols are defined as follows:

Control Symbol	Code	Comment
Idle	11111	Idle
J	11000	In SOF field
K	10001	in SOF field
H	00100	Special
R	00111	Reset
S	11001	Set
T	01101	Termination

Figure 56 Control characters in network

The NRZI signaling scheme is shown below:

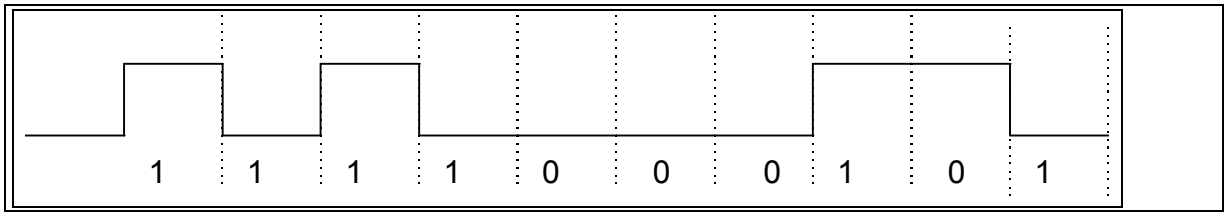


Figure 57 NRZI encoding format

Basically, this scheme uses a line transition to represent a “1” and no transition to represent a “0”. The 4B/5B symbols used above are made such that there are never more than 3 consecutive zeros in the line, assuring an adequate number of line transitions, to avoid large DC shifts.

5.2. Reset Condition

To allow the distribution of a hardware reset to the embedded CCUs, a special extension of the transmission protocol allows the generation of a reset at the level of the optical receiver in the embedded electronics. The optical network normally carries a balanced set of ones and zeros (assured by the NRZI coding), but occasionally the transmitting laser can be forced to transmit a “1” for a long time. The optical receiver shall include a comparator, capable of discriminating this condition and to generate an electrical signal generating a reset on the embedded ring.

5.3. Token Ring Interface Signal Description

5.3.1. CLK(A) and CLK(B)

5.3.2. DI(A) and DI(B)

5.3.3. CLKO(A) and CLKO(B)

5.3.4. DO(A) and DO(B)

6. CHANNEL EXTERNAL SIGNALS AND PROTOCOLS

This chapter defines the pin operation and electrical interfaces on the external ports in the CCU.

6.1. I2C port

I2C uses a synchronous two wires protocol to transmit serial data between a master up to 128 slaves. Full detailed specifications on the I2C bus can be found in http://www.philips.com looking for I2C components in the semiconductor product pages. An excellent I2C FAQ paper is also available from the SCI.ELECTRONICS newsgroup. For the special application environment of the CCU several extensions are made to the standard protocol and also some restrictions apply. Extensions and restrictions are detailed in this section.

6.1.1. Waiting for ACKOWLEDGE

The CCU implementation of the I2C interface differs from the standard I2C protocol in the handling of the acknowledge signal from the slave. The CCU assumes that all I2C devices attached to it (if working properly) will answer within a normal clock cycle from the assertion of the last data or address cycle. Should a slave not produce an acknowledge condition within this time, the CCU will assume an error and signal the condition to the FEC.

6.1.2. I2C signals

Two signals are used on each external I2C port. The **SCL** signal is a unidirectional clock/strobe signal used by the I2C master to indicate that valid data are available (or are expected) on the data line. The CCU is expected to be configured in single master I2C buses where it is expected to be the only master. The **SDA** line is a bi-directional open-drain data line. During bit write operation it is driven by the CCU, during bit read operations it is driven by the I2C slaves on the bus.

6.1.3. I2C Port protocol

Three types of addressing modes are supported by the I2C master. Single word read-write with normal 7 bit addressing, extended addressing with 10 bits special addressing (RAL mode) with 7 + 8 bit address.

6.1.3.1. Normal I2C addressing

The following figure shows the normal addressing I2C transfer format:

Start	Slave Address [6:0]	R/W [0]	Ack [0]	Data [7:0]	Ack [0]	Stop

Figure 58 I2C Normal mode addressing format

6.1.3.2. Extended I2C addressing

Start	Slave Address [6:0] 1 1 1 1 0 X X	R/W [0]	A1	Slave Address2 [7:0]	A2 [0]	Data [7:0]	Ack [0]	Stop

Figure 59 Extended I2C addressing format

6.1.3.3. Special RAL addressing

A packet using the extended RAL format is shown below:

Start	Slave Address [6:0]	R/W [0]	Ack [0]	Sub Address [7:0]	Ack [0]	Data [7:0]	Ack [0]	Stop

Figure 60 I2C Extended RAL format

This mode was defined to extend the addressing capability of I2C. The first part of the transaction is performed as a normal I2C read-write operation which transfers 7 bit of address, the second byte is used to transfer an 8 bit sub-address used to select a particular register inside an APV ASIC and the third byte carries the data information.

6.2. Parallel Interface Adapter port

t.b.d.

6.3. Memory Port

The CCU provides a simple memory-like interface to a parallel device. This is done using an interface consisting of an 8 bit data path, an 8 bit address bus, a write line and a strobe signal CS*. The figures below show a typical write and read cycles.

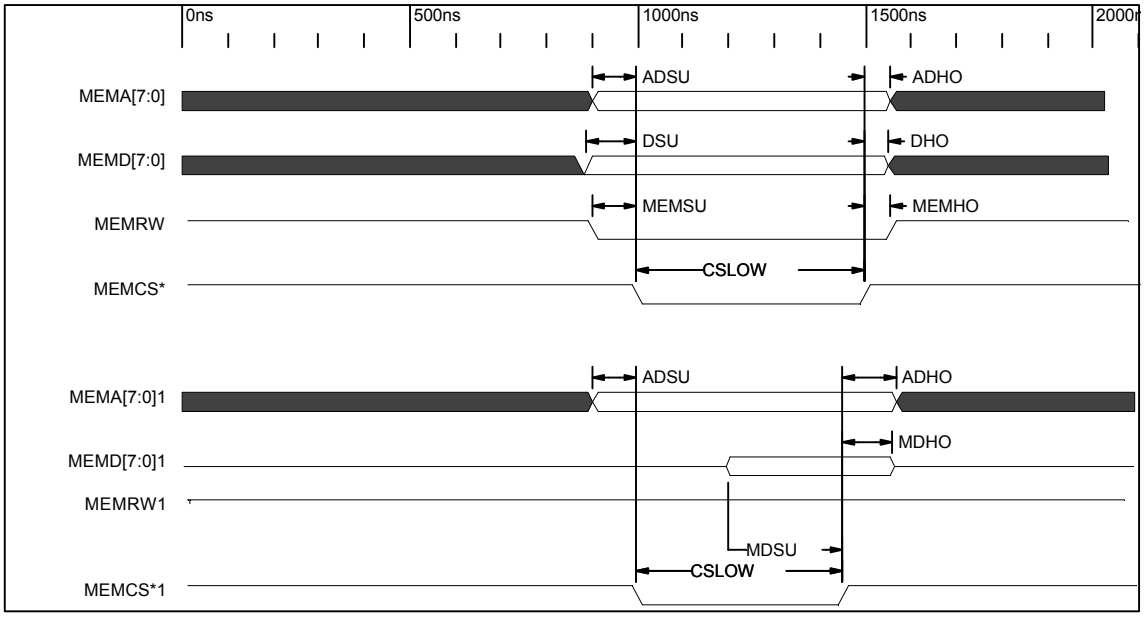


Figure 61 Memory timing for 1 MHz operation

7. PROGRAMMING MODEL

From the point of view of the user programming the POA interface on the FEC resembles a typical network adapter. A programming style as the one shown in the following pseudo-code example below is envisaged (it is assumed that the channel has already been appropriately configured):

```
I2C_write:
     prepare the I2C ch. control block in a message buffer
     transmit the message through the ring
     wait for a reply message from the destination
```

Figure 62 Single word write to I2C

The CPU generating the transfer is free to perform other operations instead of just waiting as shown in the example above.

A certain level of speed-up can be achieved by parallelizing operations on the different channels as shown in the example below.

Here is an example where the read-out of two different addresses on two channels are simultaneously done:

```
I2C_read_multiple:
     prepare the I2C channel control block_1 in a buffer
     prepare the I2C channel control block_2 in a buffer
     transmit the buffers through the ring
     wait for a reply message from destination 1
     wait for a reply message from destination 2
```

Figure 63 Concurrent single word read from I2C

7.1.1. Control link protocol

8. ELECTRICAL CHARACTERISTICS

9. PINOUT

SEE FILE: CC25FPBGA.XLS

10. APPENDIX 1 – COMMAND SYMBOLS

Symbol	Width [bytes]	Function
CH#	1	The channel number to whioch the command is directed
TR#	1	The transaction number used in the command
CMD	1	The command code
DW	1	The 8 bit data part of the command
DW16	2	The 16 bit data part of the command
DW32	4	The 32 bit data part of the command
DR	1	The 8 bit data response for the command

Figure 64 Command symbols

11. LIST OF FIGURES

Index