xCONNECT Architecture

IN THIS DOCUMENT

- Channel Communication
- The Switch
- Link layer
- Physical layer: XMOS links

XMOS devices provide a scalable architecture, where multiple xCORE devices can be connected together to form one system. Each xCORE device has an xCONNECT interconnect that provides a communication infrastructure for all tasks that run on the various xCORE tiles on the system.

The interconnect allows *packets* and *streams* of data to be communicated with low latency. Both are *circuit switched*: a packet is sent by opening a circuit, transmitting the data, and closing the circuit; a stream never closes the circuit. As opening and closing the circuit are low-overhead operations that are fully pipelined, sending packets is a low latency operation.

The interconnect relies on a collection of *switches* and *XMOS links*. Each xCORE device has an on-chip switch that can set up circuits or route data. The switches are connected by XMOS links. An XMOS link provides a physical connection between two switches. The switch has a routing algorithm that supports many different topologies, including lines, meshes, trees, and hypercubes.

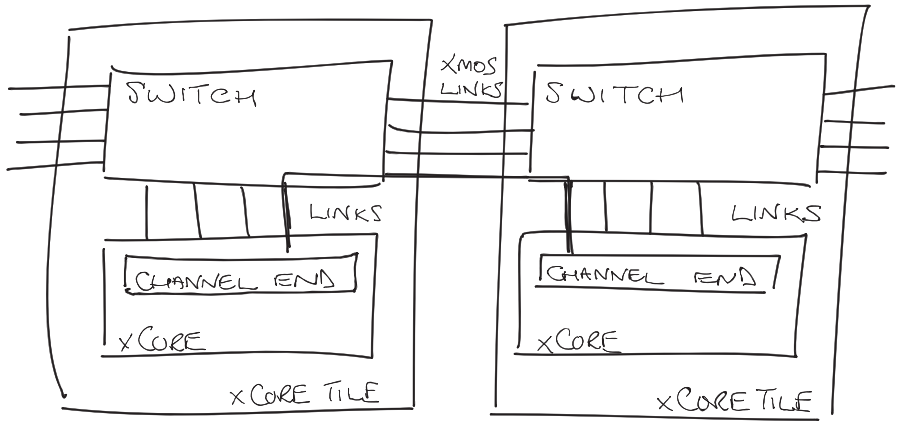

Figure 1: xCONNECT architecture

At the programming level, communication always takes place between two *channel-ends*. A channel-end is a resource on an xCORE tile, that is allocated by the

program. Their identifier comprises a unique number and their tile identifier, creating a system-wide unique identifier. Data is transmitted to a channel-end from the program by means of a sequence of output-instructions; and the other side will execute a series of input-instructions on the remote channel-end in order to receive data. The xCONNECT architecture will automatically open a communication circuit through the appropriate switches. The architecture guarantees that all data output over this stream is delivered in order. If required, the programmer can *partition* the network, supporting separation between, for example, data intensive streams and control streams. Partitioning provides real-time guarantees for parts of the network that need guarantees.

This document describes how streams and packets are communicated over channels (the transport layer), the switching fabric (the packet layer), the point-to-point protocol (the link layer) and the physical layer in turn. This document refers to configuration registers that are documented in the datasheet of the part that you use (see the *Tile configuration* and *Node configuration* sections).

1 Channel Communication

Programs running on xCORE tiles communicate by sending data from one channel-end to another channel-end. One channel-end is used to *output* data on, this channel-end has its *destination* set to the other channel-end, that can be used to *input* data from. The two channel-ends can be on the same tile, or they can be on different tiles; it is transparent to the program. If the destination channel-end is local, data is exchanged between programs running on two logical cores directly. If the destination channel-end is on a remote xCORE tile, a circuit is opened to the other tile that is used for data transmission.

1.1 Streams and tokens

The basic entity that can be output onto a channel-end is a *token*. The architecture distinguishes *data* and *control* tokens. Data tokens are eight bits of data, there are 256 control tokens that are used to send control information, such as “END of message”. Of these 256 control-tokens, 124 can be used by the application software in any way the programmer wishes.

The first token that is output will automatically set up a circuit to the remote channel-end. Packets are terminated by an END control-token, which will fold up the circuit between the two channel-ends. If no END token is sent, the circuit is kept open. A program can temporarily suspend a stream by issuing a PAUSE token at any time, which frees up the circuit and returns the communication wires to a low power state. Unlike the END token, the PAUSE token is invisible to the receiver, and is discarded once the final switch has freed its resources.

Since setting up and folding up of streams is a low-cost process, the network can be used as a *packet switching network*, by rapidly opening and creating streams. The shortest stream comprises just a single END control token, and can be used to, for example, synchronize two tasks. Much of the transmission of data and control tokens is overlapped, reducing latency.

Circuit switched streams can be used to send data that is highly sensitive to jitter. The number of streams that can be opened simultaneously is limited by the number of physical links available between switches; each uni-directional stream occupies one physical link in the direction of the stream. An XS1-L has four bi-directional links between the tile and switch, so no more than four streams can be open in one direction simultaneously. If switches are connected by fewer than four Links, the number of simultaneous streams that can be used is limited to the number of Links connecting the switches.

1.2 Channel-ends

Each channel-end has an input-side and an output-side. These can be used completely independently, but normally channel-ends are allocated in pairs and each channel-end in the pair has its destination set to point to the other one. This creates a pair of uni-directional channels that can be used for bi-directional communication and synchronization between two tasks. Indeed, channels in XC comprise two channel-ends that point to each other.

Channel-ends can be allocated and freed dynamically, and their destination can be set dynamically. However, once a channel is in use as a stream, its destination cannot be changed until the stream has finished. Services can be implemented using many-to-one channels, where multiple channel-ends all point to a single one. Each of the channel-ends can transmit tokens to the shared channel-end, but if two circuits are created simultaneously to a single channel-end, then one circuit will take precedence and be connected, and the other one is blocked until the first circuit is terminated by an END.

When setting the destination of a channel-end, the destination address comprises a 32-bit resource-id. The most significant 16 bits are the tile and processor identifier. Bits 8 to 15 are the channel number on the tile, and the lowest 8 bits are the resource type, which is 0x02 (meaning a channel-end). Channel-end 0xff on each tile is a "/dev/null" channel and can be used as to sink data if required.

1.3 Control tokens

There are 256 control tokens separated into three classes: those that can be used by application programs to transmit out-of-band data, those that are used to communicate with the architecture, and those that are used by the architecture internally.

Application control-tokens are defined by the compiler or application program, these are control tokens in the range 0x00 - 0x7f. Application tokens 0x01 and 0x02 are architecturally defined and should not be used for any other purpose; tokens 0x03 and 0x04 have a predefined conventional meaning but can be used for other purposes too:

Name	Value	Description
END	0x01	End - free up interconnect and inform target
PAUSE	0x02	Pause - free up interconnect but do not inform target
ACK	0x03	Acknowledge operation completed successfully
NACK	0x04	Acknowledge that there was an error

Token values 0x00 and 0x03-0x7F can be used by the application software in any way that it sees fit. For example, an application control token can be used in the middle of a message to ensure that the inputting and outputting side are synchronized. If, at any time, the receiver were to try and input data when a control token is available or vice versa, the task is trapped, and the program can flag or maybe recover from software errors. Application control tokens can, for example, also be used to identify alternatives of a complex data type

In addition to the application tokens, there is a block of tokens that are reserved for specific operations. These tokens have predefined meanings, and any implementation should respect those meanings. Below 11 of those tokens are defined, all other 53 token values between 0x8B and 0xBF are reserved for future use.

Name	Value	Description
READN	0x80	Read data
READ1	0x81	Read one byte
READ2	0x82	Read two bytes
READ4	0x83	Read four bytes
READ8	0x84	Read eight bytes
WRITEN	0x85	Write data
WRITE1	0x86	Write one byte
WRITE2	0x87	Write two bytes
WRITE4	0x88	Write four bytes
WRITE8	0x89	Write eight bytes
CALL	0x8a	Call code at the specified address

Control tokens in the range 0xC0 to 0xFF are reserved for use by the architecture. Tokens in the range 0xC0-0xDF are privileged tokens that are architecturally defined and may be interpreted by hardware or privileged software. They are used to perform system functions including hardware resource sharing, control, monitoring and debugging. An attempt to transfer one of these tokens to or from unprivileged software will cause an exception. Tokens in the range 0xE0-0xFF are hardware tokens that control the physical operation of the link. An attempt to transfer one of these tokens using an output instruction will cause an exception.

1.4 Virtual networks

By default, all communication happens over a single network. The network can be partitioned into four virtual networks, each of which uses its own subset of links.

The “SETN” instruction should be used to set the virtual network of a channel-end. Any data output on this channel will only be routed over links that share the same network number.

2 The Switch

The xCORE tile in the XS1-L is connected to the switch by four internal links, and the switch also allows connection to other chips via up to eight bi-directional XMOS links; not all of which may be available depending on the package. The switch implements a full crossbar between these 12 links (four internal links and eight external links) and can support 12 simultaneous circuits in each direction.

2.1 Headers

When a circuit is opened to a channel-end on a remote tile, the xCORE will create a header and transmit this header to the switch prior to transmitting the first token. The header comprises either a single byte or three bytes. The latter is the default mode, the former is an optimized mode for small systems. In order to select 1-byte mode, *all* nodes in the system should be set to generate and expect 1-byte headers by writing to the Switch configuration register of the node on all nodes.

The 3-byte header comprises a 16-bit tile identifier followed by an 8-bit channel identifier. The 1-byte header comprises a 3-bit tile identifier and a 5-bit channel identifier. As such, 3-byte mode scales up to a system of 65536 tiles with 256 channel-ends per tile, whereas the 1-byte header scales up to a system of eight tiles with 32 channel-ends per tile.

The most significant *16-n* bits of a tile identifier specify the node on which the tile resides. The least significant *n* bits of a tile identifier specify the tile number on the node. Each node has its own value for *n* depending on the number of tiles per node. For example, on an XS1-L node, *n* equals 0, and there is only one tile per node, so all 16 bits are used for the node identifier. In 1-byte mode only the lowest three bits of the node-identifier are matched. This will comprise *3-n* bits of the node-identifier, and *n* bits of the tile-identifier.

2.2 Routed links

When a header is received by the switch, it is used to build a circuit. Each bit in the tile identifier is compared with the local node identifier to decide:

- whether the local node is the final destination of the circuit;
- and if not, in which direction to extend the circuit.

The routing algorithm is as follows:

1. Establish the index of the most significant mismatching bit between the tile-identifier in the header and the local node-identifier, this is a number between 15 and 0; if the two are identical this number is defined to be -1.

2. If this mismatching bit index is less than *n* (defined in the previous section), then the header is destined to a tile on this node. If a link is available to the local tile, then this link is opened as the final part of the circuit. If no link is available, then the header is blocked until a link to the local tile becomes available.

3. If the mismatching bit index is greater than or equal to *n*, then the direction associated with this mismatching bit is looked up in the *Direction lookup table*. This table is stored Direction registers of the node configuration. Each entry is four bits; the first 32-bit register holds the lookup for mismatching bits 0..7, the second 32-bit register holds the direction for mismatching bits 8..15.

4. Each link is associated with a direction, and when the mismatching bit has been looked up, the switch finds an enabled link that has the correct direction and that is not part of any active circuit. The header is forwarded to that link, and the link is marked as being part of this circuit. If no enabled link has the required direction, then the header and subsequent tokens are discarded until the circuit is rolled up. Otherwise, if there are links with the required direction, but they are all in use, then the header is blocked until a link becomes available. This will block one of the input links.

5. If the header is blocked, then no more incoming traffic behind the header is processed. This can deadlock the network in cases where large messages are sent without knowing whether the inputting side is ready to accept the message. To create a deadlock-free environment, sender and receiver should agree that they are ready to exchange a large message by synchronizing using small, empty, messages; these can always be buffered in the channel-ends concerned and will hence not block the network. Channel ends can hold at least a word and one token enabling, for example, an identifier and an end token to be output unsolicited.

This routing algorithm enables the construction of meshes, hypercubes, trees, embeddings of each of these inside each other (for example a mesh of trees), and other networks.

To set up a network, each switch has to be configured by:

1. Setting the node identifier of the switch

2. Setting the direction table of the switch

3. Setting the direction for each enabled link

Two example topologies are shown below; a regular pipeline, and a mesh with missing wires (or pipe of pipelines). Each node shows the node-identifier, the direction associated with each link, and the direction associated with each mismatching bit in the tile identifier. Note that the direction numbers are only relevant within a node, and we have chosen random numbers (2, 3, 1, 7) for the directions to reflect this.

If required, up to four separate virtual networks can be created by setting the network number of a set of links. A header that arrives on a link that belongs to virtual network *X* will only be forwarded to a link that also belongs to network *X*;

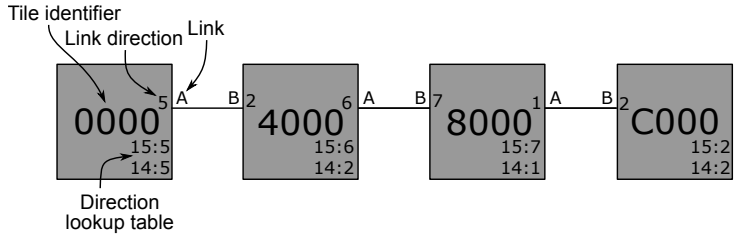

Figure 2: Example: Configuring a pipeline of four XS1-L8

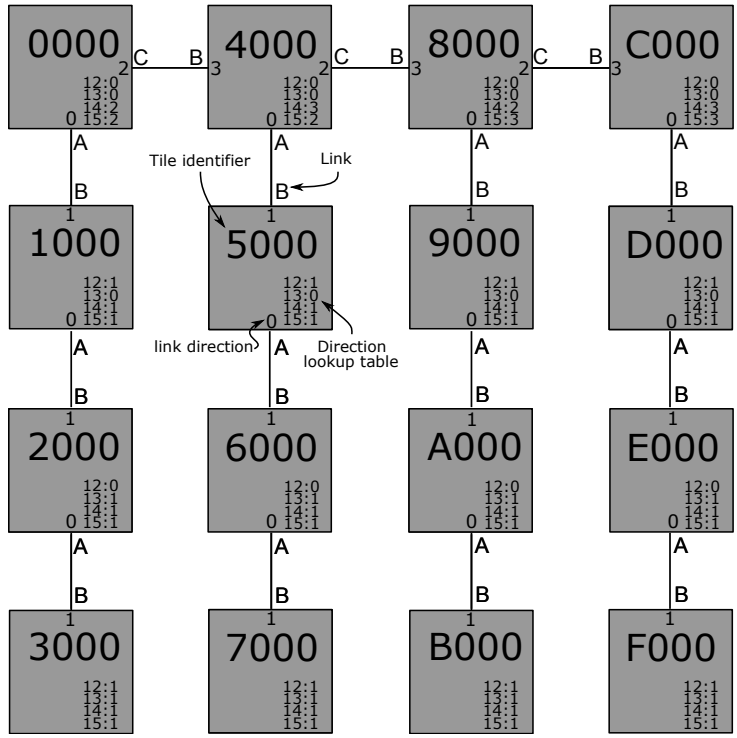

Figure 3: Example: Configuring a pipeline of four pipelines of four XS1-L8s

otherwise it will be blocked or discarded. The channel-ends should also be set to belong to network *X*. When setting up networks, no traffic should flow over the target network, as routing would be ambiguous. Network assignments are designed to be static, but if a link needs to be reassigned to, for example, the default network, the link should be disabled before the assignment is changed.

2.3 Non-routed links

Links can be set to deliver data to a statically determined channel-end instead of using the routing table. In non-routed mode no header is sent, and the message is sent to a specified tile and channel-end. One register for each link (the Static Link Configuration) is used to enable the link, and set the tile/channel identifier.

When an xCORE wants to send data over a non-routed link it sets the channel-end destination register to address an xCORE tile that differs in place *x* from the local tile-id. Destination *x* is mapped in the lookup table to a direction that is associated with the required link. The destination channel has no relevance and is set to zero.

3 Link layer

The link layer protocol operates a point-to-point connection over a bi-directional XMOS link. The link layer governs when data is transmitted, and how links start communicating.

An XS1-L has eight bi-directional links, denoted Link A to H; typically the datasheet abbreviates them as "X0LA" which means Link A of xCORE tile 0. Only a subset of links is typically bonded out, and some links may be used inside the device. For example, an XS1-L8A-64-TQ128 has four links bonded out: X0LA, X0LB, X0LC, and X0LD; they are all connected to tile 0. An XS1-L16A-128-FB324 also has four links bonded out: X0LA, X0LB, X1LA, and X1LB, two on each of the tiles; internally links E, F, G, and H of tile 0 are connected to links H, G, F, and E of tile 1.

3.1 Credits

Each link has two counters associated with it: a *credit-counter* and a *credits-issued-counter*. The credit-counter governs whether tokens can be transmitted over the link: tokens can only be transmitted when the credit counter is not zero, and the credit counter is decremented for every token that is transmitted. This guarantees that each token can be stored on the receiving side. The receiving side will issue credits when it has space available. The receiver uses its *credits-issued-counter* and the available space in its input buffers to work out how many credits it can issue. To save bandwidth, the transmitter issues the largest possible credit token. The tokens used to issue credits are:

Name	Value	Description
CREDIT8	0xE0	Give additional 8 tokens of credit
CREDIT64	0xE1	Give additional 64 tokens of credit
CREDIT16	0xE4	Give additional 16 tokens of credit
HELLO	0xE6	Say hello and solicit credits (discussed later)

A transmitter must have a credit counter of at least 7 bits. Hence, it is illegal to send two subsequent CREDIT64 tokens as this would overflow the credit counter. It is legal to send a second CREDIT64 when one token has been received.

The only tokens that bypass the credit mechanism are the link-level tokens in the range 0xE0..0xFF, since these are not stored in the buffers. These tokens can be transmitted even if no credits are available, and they do not affect the credit counters. This includes the HELLO and CREDIT tokens.

3.2 Initializing a link

Initially links are *disabled*. When disabled, no outside signals are coming in. When a link is enabled, signals come through and are assembled into tokens. Links should only be enabled when the input and output signals are all low, otherwise spurious transitions will be observed. A link can be RESET which clears all state, clearing out any tokens that may have been received.

A link is initialized by triggering the transmission of a HELLO control token. This operation clears the *credit counter* of the XMOS link, and transmits a HELLO control token. On reception of a HELLO, the receiving XMOS link clears its *credits-issued counter*, and issues credits by transmitting one or more CREDIT tokens, setting the *credits-issued counter*. On reception of a CREDIT token, the receiving XMOS link increases its *credit-counter*. At this stage, both transmitting and receiving sides have an agreed number of credits, and data can be transmitted in one direction.

The HELLO token is triggered by a write to the control register of the link. This write can be triggered locally, or by a remote node. In the latter case, the link has to be enabled and have credits. So one way to set up a bi-directional link is for the local node to first trigger a HELLO locally, and then send a message to the remote switch forcing it to say HELLO back. This initializes the credit counters on both transmitters and both receivers in order to establish a bi-directional link.

The above scheme assumes that the two nodes are reset simultaneously and that one node is the master. Different schemes can be used for hot-plugging or for master-slave relationships:

- If hot-plugging is required, links should be set to non-routed mode in software, a software layer should first enable the link, then repeatedly issue HELLO until it establishes a link by reading the "credits issued" bit. The software waits between resetting the link by disabling and enabling it, and issuing HELLOs.
- In some designs there is a master-slave relationship between nodes; for example a "master"-node that is always on controlling the power-supply of one or more "slave"-nodes. In this particular case, the master has knowledge that the slave is in a known state when booted. The master will hence wait for the slave to be booted, and then the master will enable the link and issue a HELLO.

4 Physical layer: XMOS links

XMOS links use a transition-based non-return-to-zero signalling scheme, where the signal transitions between ground and VDDIO. Bits are sent at a rate programmed to meet applications requirements. XMOS links can be switched between a slow serial mode that uses two wires in each direction (four wires in total), and a fast, wide mode that needs five wires in each direction (10 wires in total).

4.1 Configuring link width and speed

Bits are transmitted at a speed and width that is set under software control by writing to the Link Configuration register of the link. The mapping between the register and the link is specified in the datasheet. The number of clock cycles between transitions, the number of clock cycles between tokens and the width of the link are programmed in these registers. On a system-reset the link is set to a serial XMOS link using two pairs of wires with 400 clock cycles between tokens and 400 clock cycles between symbols. This link is slow enough to support a wide variety of devices on the receiving end.

The token spacing field is encoded with an offset of 2, i.e. 0x000 represents 2 cycles delay, 0x001 represents 3 cycles delay, up to 0x7ff representing 2049

cycles delay. The symbol spacing field is encoded with an offset of 1, i.e. 0x000 represents a single cycle delay, 0x001 represents a two-cycle delay, etc and 0x7ff represents a 2048 cycle delay. All clock cycles are relative to the *switch clock*, which is normally the clock generated by the PLL, but it can be slowed down by writing to the System Switch Clock Divider register of the node configuration.

The XS1-L cannot receive data if the transmitter does not space the symbols by at least two clock cycles. So, when two XS1-Ls are running at the same clock, they should set their symbol/token delay to at least 2. If one of the XS1-Ls has a lower switch-clock-speed, the other one should adjust its token/symbol delay accordingly. Assuming a 400 MHz switch clock, the link can be set as fast as 200 Mtransitions/sec, or as slow as 100 Ktransitions per second.

4.2 The serial XMOS link

The serial XMOS link uses two data wires in each direction labelled “0” and “1”. A transition on wire “0” represents a zero bit and a transition on wire “1” represents a one bit. It is the transition that signals the bit; the level of the wire is irrelevant.

For each token 10 transitions are made. The first eight transitions encode the 8 bits of the token value, transmitted most significant bit first. The ninth transition signals whether this is a *control* or *data* token. A transition on “1” signals a control-token, a transition on “0” signals a data-token. The final transition causes both wires to go back to low state. The two signal wires are both at rest between tokens.

For example, to send control token 0x09, the following transitions are made:

- Wire “0” to high (signals a zero in bit 7)
- Wire “0” to low (signals a zero in bit 6)
- Wire “0” to high (signals a zero in bit 5)
- Wire “0” to low (signals a zero in bit 4)
- Wire “1” to high (signals a one in bit 3)
- Wire “0” to high (signals a zero in bit 2)
- Wire “0” to low (signals a zero in bit 1)
- Wire “1” to low (signals a one in bit 0)
- Wire “1” to high (signals control token)
- Wire “1” to low (terminate transmission - both wires are in rest state)

4.3 The fast XMOS link

The fast XMOS link uses five wires in each direction to transmit data; 10 wires in total. The wires are labelled “0”, “1”, “2”, “3”, and “4”. A transition on one of the five wires transmits a *symbol*:

Transition on	Symbol	Bit value
“0”	value	00
“1”	value	01
“2”	value	10
“3”	value	11
“4”	escape	

A sequence of four symbols is used to encode tokens. If all four are *value* symbols, a total of eight bits of data are transferred (a data-token). A control token is transmitted using one *escape* symbol, and three value symbols. The bits of data and control tokens are always transmitted starting with the two most significant bits. In the case of control tokens, the first two bits of the control token are determined by the position of the escape symbol in the four transitions:

First	Second	Third	Fourth	Encodes
value	value	value	value	Data tokens 0x00 - 0xFF
escape	value	value	value	Control tokens 0xC0 - 0xFF
value	escape	value	value	Control tokens 0x80 - 0xCF
value	value	escape	value	Control tokens 0x40 - 0x7F
value	value	value	escape	Control tokens 0x00 - 0x3F

For example, to send control token 0x09, the following transitions are made:

- Wire “0” to high (signals 00 bits, bits 5 and 4)
- Wire “2” to high (signals 10 bits, bits 3 and 2)
- Wire “1” to high (signals 01 bits, bits 1 and 0)
- Wire “4” to high (escape, signals a control token in range 0x00-0x3F, hence bits 6 and 7 are 00: 00 00 10 01 = control token 0x09).

When a token has been transmitted, some wires may be left high (in the previous example four wires were left high). Wires are only returned to zero when a circuit is PAUSED or ENDED. For this purpose a sequence of an END or PAUSE token and an optional return-to-zero token are transmitted. They are chosen so that after them all wires are low.

This means that control tokens 1 (END) and 2 (PAUSE) are not transmitted using the conventional single escape for control tokens less than 64. Instead there are sixteen possible sequences to transmit an END or PAUSE token on the 5-wire XMOS link:

First	Second	Third	Fourth	Encodes
escape	escape	value	value	16 different END tokens
value	value	escape	escape	16 different PAUSE tokens

All of them signal END/PAUSE; but by choosing the appropriate sequence value symbols, at least two data lines can be taken to zero. This guarantees that after an END or PAUSE token either zero or two wires are left high.

- If wire “4” is still high, one of the RTNZ tokens is transmitted; chosen to return both the “4” wire and the last data wire to zero. (note that if wire “4” is high, exactly one of wires “0”... “3” must be high).

First	Second	Third	Fourth	Encodes
escape	value11	value11	value00	RTNZ0 (control token 0xFC)
escape	value11	value11	value01	RTNZ1 (control token 0xFD)
escape	value11	value11	value10	RTNZ2 (control token 0xFE)
escape	value11	value11	value11	RTNZ3 (control token 0xFF)

- If two of the wires “0”, “1”, “2” and “3” are still high, a NOPD token is transmitted. The NOPD token has two transitions on wire “4” and hence leaves wire “4” low. The two “v” transitions are chosen to return the final two wires to low:

First	Second	Third	Fourth	Encodes
escape	value	value	escape	16 NOPD tokens

For example, to send an end-of-message after the control token sent earlier (wires “0”, “1”, “2”, and “4” are high), transmit the following:

- Wire “4” to low (signals an escape).
- Wire “4” to high (signals a second escape, this is an END token).
- Wire “0” to low.
- Wire “1” to low. Sends the END token, and only wires “4” and “2” are left high; hence, a RTNZ2 token must be transmitted.
- Wire “4” to low (signals an escape).
- Wire “3” to high (transmits token value 11).
- Wire “3” to low (transmits token value 11).
- Wire “2” to low (transmits token value 10). This has transmitted token 0xFE, which is a RTNZ2 token that is ignored by the receiver. All wires are now low.

In 5-wire mode, the link-level tokens use special encodings that guarantee that the wire state is not affected:

First	Second	Third	Fourth	Encodes
escape	value00	escape	value00	CREDIT8
escape	value01	escape	value01	CREDIT64
escape	value10	escape	value10	HELLO
escape	value11	escape	value11	CREDIT16

4.4 Physical considerations

As the link protocol is transition based, care should be taken to avoid spurious transitions. In particular, the links assume that they start low. If a noisy power supply leaves a charge on the links then an external pull down should be placed on the links.

Over short distances, links can be wired up directly. A small series resistor (33 ohm), can be inserted near the transmitter to terminate the signal for medium length PCB traces. Over longer distances, the links may have to be transmitted as LVDS pairs to guarantee correct operation.

4.5 Speed considerations

For a 400 MHz system clock and symbol/token spacing of 2 the transition rate achievable is 200 Mtransitions/second. The actual speed that can be achieved depends on the electrical characteristics of the physical connection.

On a two wire system eight bits are transmitted every 10 transitions, leading to a speed of 160 Mbits/s. On a five wire system, eight bits are transmitted every four transitions, leading to a speed of 400 Mbits/s. The actual data rate of the connection is slightly lower since there is an overhead in transmitting credit tokens.