XS1 System Specification

(VERSION 1.3)

XMOS™

2009/1/27

Authors:

DAVID MAY
ALI DIXON
AYEWIN OUNG
HENK MULLER

1 Introduction

This document specifies the XS1 boot protocol, the XMOS link specification, the security module comprising a one-time-programmable memory, and a description of all configuration registers.

2 Booting the XS1-G4

The normal boot procedure is to first boot Core 0 from an external ROM or Flash memory that is connected via an SPI interface. Each of the other cores is then booted over the core's channel-end 0 from core 0. This boot mode, called SPI-boot, is enabled by setting pin SS_XC0_BS0 to 0 (ground).

Alternatively, the XS1-G4 can be booted via the JTAG interface. This is useful during program development, and can be enabled by setting pin SS_XC0_BS0 to logical 1 (IO_VDD).

A further option is to use *secure boot* for one or more of the cores. Each core can be configured to boot from a program held in its security module. This is enabled by setting a bit in the core's security module and will cause the core to always use secure boot.

2.1 Boot format

When a core is booting over the SPI interface, from a channel, or from the security module, the boot monitor built-in to the XS1-G4 reads in a program and stores it in on-chip RAM starting at the lowest memory location. The program is then started by transferring control to the lowest location in RAM.

The boot format used for the program to boot from an SPI interface, channel or security module is represented as follows.

1. the program size s in words - a 32-bit value, least significant byte first.
2. program consisting of $s \times 4$ bytes.
3. a 32-bit CRC, least significant byte first.

The CRC is calculated over the byte stream represented by the program size and the program itself. The polynomial used is is 0xEDB88320 (IEEE 802.3); the CRC register is initialised with 0xFFFFFFFF and the residue is inverted to produce the CRC.

The CRC check can be disabled by setting the CRC to 0x0D15AB1E.

2.2 Boot from SPI interface

To boot from an SPI interface, an SPI slave device must be connected as follows.

port	**use**
P1A0	MISO
P1B0	SS
P1C0	SCLK
P1D0	MOSI

A READ command is issued with a 24 bit address 0x000000. Based on the 100MHz reference clock of the Xcore, an SPI clock rate of 2.5 MHz is used. The clock polarity / phase is of 0 / 0.

The Xcore expects each byte to be transferred with the least-significant bit first. Many programmers will write bytes into an SPI interface using the most significant bit first, and hence you may have to reverse the bits in each byte of the image stored in the SPI device.

If a large boot image is to be read in, it is faster to first load a small boot-loader that will read the large image using a faster SPI clock, for example 50 MHz, or as fast as the flash device supports.

If field-upgradeable firmware is required, a small boot-loader should be stored in the first sector of flash memory, followed by two boot-images starting on sector boundaries. The boot-loader should be written to first read the first image, and on CRC failure boot from the second image. On upgrade, the first image is upgraded first, followed by the second image. If the upgrade process is interrupted at any point, there will always be a working boot image.

The SPI device will boot core 0 of the XS1-G4 only, the other cores must be booted via channels.

2.3 Boot from channel-end 0

When boot from SPI is selected, the boot monitor on cores 1, 2, and 3 will be waiting on communication via channel end 0. If required, those cores can be booted by opening a channel to channel-end 0 of each of the cores, and transferring the code to those cores.

In order to boot a core the following procedure must be followed:

- Allocate a channel-end and connect it to the channel end of the core you want to boot (one of identifiers 0x00010002, 0x00020002, or 0x00030002), using a SETD instruction. We call the allocated channel-end *c*.
- Output the 32-bit identifier *c* of the channel over *c*, this allows the other side to open a back-channel.
- Send the sequence of bytes representing size, code, and CRC as specified earlier.
- Send an END control token.
- Receive an END control token.
- Free *c* using FREER

2.4 Boot from Security Module

If a core is set to use secure boot, the program in boot format is taken from address 0 of the OTP (one-time-programmable) memory in the core's security module. Each core has its own individual OTP memory, and hence some cores can be booted from OTP while others are booted from SPI or the channel interface. This enables an XS1-G4 to be partially programmed, dedicating one or more cores to perform a particular function, leaving the other cores user-programmable.

3 XMOS link specification

The interconnect provides communication between all tiles on the system. A system can comprises one or more nodes, that may be physically separated. In conjunction with simple programs, the interconnect can also be used to support access to the memory on any tile from any other tile, and to allow any tile to initiate programs on any other tile.

The interconnect allows *streams* of data to be communicated with low latency. A stream comprises *data* tokens and *control* tokens, where data tokens contain 8 bits of data, and control tokens specify operations. Streams are circuit switched, but they can be set-up and terminated at low cost. This enables the network to be used as a packet switching network, where short packets are carried through the interconnect in a pipelined manner.

Each tile has four links that connect the tile to an on-chip switch that provides non-blocking communication between the tiles on a node. The on-chip switch also provides a number off-chip links that can be connected to links of other nodes. The structure and performance of the link connections can be varied to meet the needs of applications. The topology of the interconnect is not fixed, a topology appropriate to the application can be used.

The network supports partitioning. Partitioning provides separation between, for example, data intensive streams and control streams. Partitioning provides real time guarantees for parts of the network that need the guarantees.

As far as a program is concerned, communication always takes place between two *channel ends*. A channel end is a physical resource that is allocated on the Xcore. Channels-ends reside on a tile and are identified by means of an identifier on the tile, a tile-identifier, and a node-identifier. Data is transmitted to a channel end by using a sequence of OUT instructions, and OUTCT instructions; when a communication is complete, an END token is transmitted by the program, which will free up any resources allocated in the network. The architecture guarantees that all data- and control-tokens sent over this stream are delivered in order. Multiple streams can be set up, and no guarantee is given about the ordering of data and control tokens between streams.

This document describes what stream (the transport layer), the switching method (the packet layer), the point-to-point protocol (the link layer) and the physical layer.

There are four groups of control tokens:

- Tokens 0x00-0x7f: (Application tokens). These are intended for use by compilers or applications software to implement streamed, packetised and synchronised communications, to encode data-structures and to provide run-time type-checking of channel communications.

- Tokens 0x80-0xbf: (Special tokens) are architecturally defined and may be interpreted by hardware or software. They are used to give standard encodings of common data types and structures.
- Tokens 0xc0-0xdf: (Privileged tokens) 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 0xe0-0xff: (Hardware tokens) are only used by hardware; they control the physical operation of the link. An attempt to transfer one of these tokens using an output instruction will cause an exception.

The sections below define the protocol layers bottom up: physical layer (Section 3.1), link layer (Section 3.2), switch layer (Section 3.3), physical layer (Section 3.1), processor communication (Section 3.4), and channel communication (Section 3.5).

3.1 Physical layer

External link communication uses a transition-based non return-to-zero signalling scheme. Bits are sent at a rate derived from the XS1 clock; this rate can be programmed to meet applications requirements.

The links can be switched between between a slow serial mode that uses just four wires, and a fast, wide mode that needs 10 wires. These two modes use different encoding schemes.

3.1.1 Serial external link

The serial link uses two data wires, “0” and “1” in each direction (four wires in total). A transition on wire “0” represents a zero bit and a transition on wire “1” represents a one bit. Note that it is the transition that signals the bit; the level of the wire is irrelevant.

For each token 10 transitions will be made, transmitting 10 bits. The first 8 bits are the token value. transmitted most significant bit first. The next bit signals whether the transmission is for a *control* or a *data* token. A 1-bit signals a control-token, a zero-bit signals a data-token.The final bit is an even parity bit is transmitted that will cause both wires to go back to low state. The two signal wires are both at rest between tokens.

For example, in order to send control token 0x09 one would transmit the following:

1. Set wire "0" to high (signals a zero in bit 7)
2. Set wire "0" to low (signals a zero in bit 6)
3. Set wire "0" to high (signals a zero in bit 5)
4. Set wire "0" to low (signals a zero in bit 4)
5. Set wire "1" to high (signals a one in bit 3)
6. Set wire "0" to high (signals a zero in bit 2)
7. Set wire "0" to low (signals a zero in bit 1)
8. Set wire "1" to low (signals a one in bit 0)
9. Set wire "1" to high (signals control token)
10. Set wire "1" to low (terminate transmission - both wires are in rest state)

3.1.2 Fast external link

The fast link uses five wires in each direction to transmit data; 10 wires in total. Those wires are called "0", "1", "2", "3", and "4". 1-of-5 codes are used to transmit data;; changing the state of one of the five wires transmits a *symbol*. A transition on each of the wires has the following meaning:

transition on	**symbol**	**meaning**
"0"	*v*	value 00
"1"	*v*	value 01
"2"	*v*	value 10
"3"	*v*	value 11
"4"	*e*	escape

A sequence of four symbols are used to encode the data and control tokens. If all four symbols are data *v* symbols, a total of 8 bits of data are transferred (a data-token). If one of the four symbols is an *e* symbol, and the other three are *v* symbols, then a control-token is transmitted.

Transitions				**use**
first	second	third	fourth	
v	*v*	*v*	*v*	256 data tokens
e	*v*	*v*	*v*	64 control tokens 192-255
v	*e*	*v*	*v*	64 control tokens 128-191
v	*v*	*e*	*v*	64 control tokens 64-127
v	*v*	*v*	*e*	64 control tokens 0-63

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 *e* symbol.

For example, in order to send control token 0x09 one would transmit the following:

1. Set wire “0” to high (signals 00 bits in bits 5 and 4)
2. Set wire “2” to high (signals 10 bits in bits 3 and 2)
3. Set wire “1” to high (signals 01 bits in bits 1 and 0)
4. Set wire “4” to high (signals an escape, bits control token bits 6 and 7 are 0)

After transmitting one token, none, two, or four wires are high. Wires are returned to zero only when a message is completed (when data is streamed wires do not return to zero). In order to return to zero, a sequence of an END token and an optional return-to-zero-NOP are transmitted. They are chosen so that after them all wires are low.

Transitions				**use**
first	second	third	fourth	
e	*e*	*v*	*v*	END tokens
e	0	*e*	0	CREDIT8 token
e	1	*e*	1	CREDIT64 token
e	2	*e*	2	RESET token
e	3	*e*	3	CREDIT32 token
v	*v*	*e*	*e*	PAUSE tokens
e	*v*	*v*	*e*	NOPD tokens
e	3	3	*v*	NOPE tokens (control tokens 252...255)

There are sixteen possible sequences to transmit an END token on the 5-wire link. All of them signal END; but by choosing the appropriate sequence, it can be guaranteed that none, one, or two of wires “0” to “3” are left high. After the END token, one of the NOP tokens may have to be transmitted in order to return the final wires to zero.

- If wire “4” is high after the END token, then the NOPE token is transmitted, and the final transition is chosen so to return the last high wire low (note that if wire “4” is high exactly one of wires “0”... “3” must be high).
- If wire “4” is low after the END token, then the 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 so to return the final two wires to low.

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

1. Set wire “4” to low (signals an escape)
2. Set wire “4” to high (signals a second escape, this is an END)
3. Set wire “0” to low
4. Set wire “1” to low. This ends the END token, now only wires “4” and “2” are left high; hence, we need to transmit an NOPE token.
5. Set wire “4” to low (signals an escape)
6. Set wire “3” to high (transmits token value 11)

7. Set wire “3” to low (transmits token value 11)
8. Set wire “2” to low (transmits token value 10). This has transmitted token 254, which is a NOP token that will be ignored by the receiver. All wires are now low.

A link can be *paused* by transmitting one of the PAUSE tokens, followed by a NOP token in order to bring all five wires to a low state.

Note that the physical layer transmits tokens 0x1 and 0x2 using two escapes; they are not transmitted using the conventional single escape for control tokens less than 64. It is also the task of the physical layer to transmit a NOP after either a PAUSE or END token. Finally, on reception of a double escape END or PAUSE token, the physical layer must report this as a 0x1 or 0x2 control token, and the physical layer shall discard any NOP tokens that are received.

The encoding of the four hardware tokens operated by the physical layer is:

Name	**Value**	**Description**
RTNZ1	0xfc	NOP (return “0” to zero).
RTNZ2	0xfd	NOP (return “1” to zero).
RTNZ3	0xfe	NOP (return “2” to zero).
RTNZ4	0xff	NOP (return “3” to zero).

The PAUSE and END tokens are application level control tokens, and their encodings for higher levels are discussed in Section 3.5.1.

3.1.3 XS1-G4 Physical layer configuration

Bits are transmitted at a speed that is set under software control. Both speed and width are set by writing to the link’s speed register. Each of the speed registers specifies the width of the link, the gap between bits, and the gap between tokens. The addresses and contents of the speed registers are summarised in Section 3.6.2.

Normally the number of system clock cycles between tokens can be set to the number of system clock cycles between bits; this bit spacing must be at least 2 clock cycles.

For a 400MHz system clock and bit spacing $s \geq 2$, the data rate achievable using 2 signal wires is $(320/s)$ Mbits/second; the data rate using 5 signal wires is (800/s) Mbits/second. The actual speed that can be achieved depends on the electrical characteristics of the physical connection.

3.2 Link layer

The link layer protocol operates a point-to-point connection over a full-duplex external link. The link layer governs when data is transmitted, and how links start communicating. Three control tokens are used by the link layer: CREDIT8, CREDIT64, and LRESET.

A link can be disabled or enabled. When disabled, no outside signals are coming through to the link state machine. When enabled, signals come through and are assembled into tokens. On enable, a LRESET is issued that clears both credit ounters.

3.2.1 Credits

The normal mode of operation is that a switch can *issue credits* on a link - when it does so, the switch allows the transmitter on the remote end of the link to transmit data to this switch.

The switch specifies how much credit is issued (8 to 64 bytes) using the reserved control tokens CREDIT8 and CREDIT64. The transmitter will not transmit more tokens then there are credits. When multiple credit messages are issued, credits are summed together at the transmitting side; a transmitter must have a credit counter of at least 7 bits. Hence, it is illegal to send two subsequent CREDIT64 tokens, but legal to send a second CREDIT64 when one token has been received.

An LRESET token can be sent. On reception of an LRESET the the credit counter will be set to 0. The credit counter on the transmitting side is also set to 0 when issuing an LRESET. On reception of an LRESET, the receiver must issue credit.

A transmitter should issue credits to the receiver, if it knows that the receiver is running low on credits, and if there is space in its input buffer. In order to save bandwidth, the transmitter should try and issue the largest possible credit token.

All data tokens require and consume credits. Most control tokens require and consume credits when transmitted, the exceptions are CREDIT*n*, LRESET, and RTNZ*n*; these tokens can be transmitted when there are no credits present because the link layer will interpret them and not insert them into the buffer. The application level tokens END and PAUSE consume credits as usual since they do end up in the buffer.

3.2.2 Start-up

On start-up, the credit counters are always zero. When a link is enabled, it will first issue a RESET and then issue initial credits immediately following the reset. The other side of the link will perform the same operation, one of the resets will arrive last, and will cause credits to be issued in both directions, setting up the network.

3.2.3 Network numbers

A link can be assigned to be part of one of four “networks”. That is, this link will only carry traffic belonging to that network. For this to work, both channel ends must also be made part of this network. The intended use of this is to assign specific links to, for example, carry small control messages.

When setting up networks, no traffic should flow over the target network; for 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, then the link should be disabled before the assignment is changed.

3.2.4 Link Layer configuration

Before a link can be used it must be enabled. These actions are performed by writing a ’1’ to the appropriate bit in the speed registers (the details are shown in Section 3.6.2).

3.3 Switch layer

The switch layer forwards messages from one link to another. Before data is transmitted on a stream, the switch sends a header to the destination tile. The header establishes a route through the interconnect, and subsequent tokens will follow the same route until the end-of-message (END) or pause (PAUSE) token are encountered. The header contains the identifier of the destination processor This is encoded using either an 8 bit node id, 8-bit core id, and 8-bit channel id or a 1-bit node id and a 2-bit core-id, and 5-bit channel id.

The header mode can be set by software, by changing the lowest bit of configuration register 0x4 (Section 3.6.2). By default 3-byte mode is used, if the 1-byte header is used it should be used on all nodes in the system.

Each node has a switch with a configurable identifier and routing table. The identifier is a bit pattern that (uniquely) identifies this node in the system. When a stream enters the switch, the destination node identifier is compared bit-by-bit with the switch-identifier. If all bits match then the message is destined for this node, and the message is routed to one of the local tiles using the tile-identifier.

If the switch-identifier is not equal to the stream's destination-node-identifier, then the number of the first bit that differs specify the dimension (direction) in which the message needs to be routed; this results in eight possible routing dimensions. The routing table associates each outgoing link with exactly one dimension, and the switch picks an available outgoing link for this dimension before forwarding the stream. This mechanism enables system designers to construct the routing tables so to build meshes, pipelines or hypercubes.

The node identifier of the G4 is initialised by writing its value in 7 ... 0 of the node identifier register. The most significant 24 bits are ignored.

Each link can be associated with one of four logical networks by writing the network number to bits 5 ... 4 of the link's configuration register. These network numbers correspond to the network numbers used when initialising channels using the SETN instruction.

On the G4 the mismatching bit in the node address (a number between 0 and 7 inclusive) is used to lookup an outgoing link. Each link can be associated with one of the mismatching bits by writing the direction to bits 10 ... 8 of the link's configuration register. Four bits are sufficient for up to 16 directions. On the G4 only 3 bits are used.

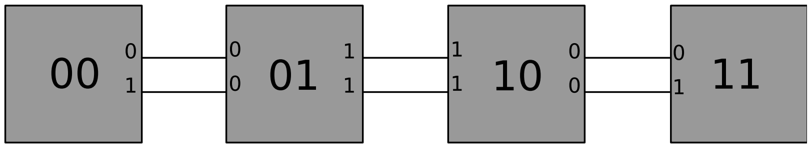

Figure 1 *Example: configuring a pipeline of four G4s*

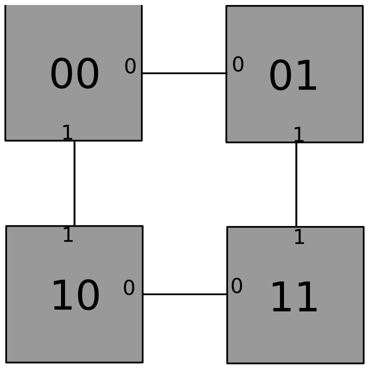

Figure 2 *Example: configuring a hypercube of dimension 2 (a square)*

Three example topologies, a regular pipeline, a small hypercube and a big hypercube are shown in Figures 1, 2, and 3.

3.3.1 XS1-G4 Switch Layer configuration

Each core in the G4 is connected to the switch by four internal links (PA/PB/PC/PD), and the switch also allows connection to other chips via sixteen external links. The switch fully connects its 32 links (16 internal links and 16 external links) and can support 32 simultaneous message transfers.

The switch is configured by sending it *configuration messages*. These messages request the switch to write data to or read data from a bank of 32-bit configuration registers internal to the switch. These messages are used when booting to set the *node identifier* of the switch, associate *specific links with logical networks* and set the *speed and width of the links*, and set the *routing strategy*. Section 3.6.2 summarises the registers (and the fields within the registers) that must be initialised in order to use the switch. The addresses used in the config-

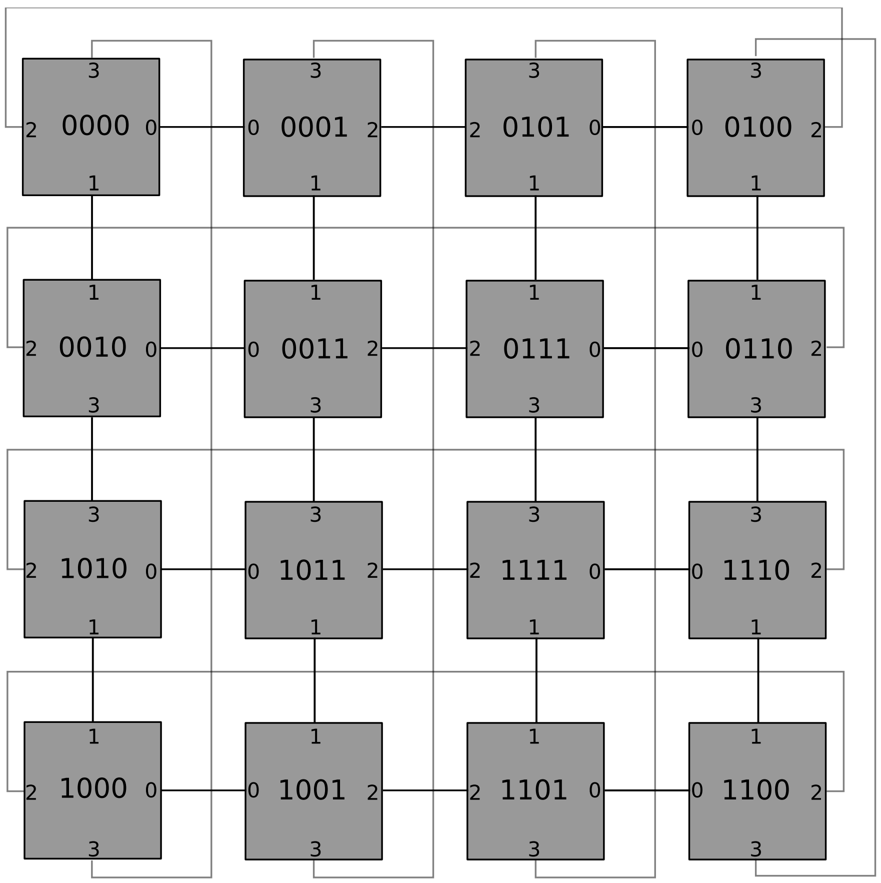

Figure 3 *Example: configuring a hypercube of dimension 4*

uration messages are the register numbers of the 32-bit registers in the switch.

3.4 Processor communication

When processors communicate with each other, they transmit messages over the switches that, in addition to the 16- or 3-bit switch header include an 8- or 5-bit channel-end identifier. When a message is transmitted to the switch, it has a 24-bit (16 bits core-id + 8 bits channel-end) or 8-bit (3 bits core-id + 5 bits channel-end) header. When a message is transmitted to the processor over an internal link, the message always has an 8-bit header which indicates the channel-end. If 1-byte switch headers is used, then the first three bits of this byte will always be 0.

Processors can also communicate with switches. In this case the switch must be set to 3-byte header mode. When a message is transmitted to the switch, it contains a 2-byte header, and then a control token PSCTRL or SSCTRL. This indicates that the message is destined for the processor-control or switch-control associated with the processor addressed by the first two bytes. Messages that are transmitted to the PSCTRL or SSCTRL follow the following format:

- Two byte header identifying the destination processor/switch
- PSCTRL or SSCTRL token.
- WRITEC control token
- Two bytes identifying core that reply should go to.
- One byte identifying Channel-end for reply.
- Two bytes identifying address within switch (*address*[15 ... 8], *address*[7 ... 0])
- Four bytes data to be written (*data*[31 ... 24], *data*[23 ... 16], *data*[15 ... 8], *data*[7 ... 0])
- EOM control token (value (0x01)

This will result in the following reply message.

- three bytes header (two bytes core identifier, one byte channel)
- ACK control token
- EOM control token

A read message is sent as follows:

- Two byte header identifying the destination processor/switch
- PSCTRL or SSCTRL token.
- READC control token

- Two bytes identifying core that reply should go to.
- One byte identifying Channel-end for reply.
- Two bytes identifying address within switch (*address*[15 ... 8], *address*[7 ... 0])
- EOM control token

This will result in the following reply message.

- three bytes header (two bytes core identifier, one byte channel)
- ACK control token
- Four bytes data read (*data*[31 ... 24], *data*[23 ... 16], *data*[15 ... 8], *data*[7 ... 0])
- EOM control token

The four privileged tokens used to control the switch are defined as follows:

Name	**Value**	**Description**
WRITEC	0xc0	Write control register.
READC	0xc1	Read control register.
PSCTRL	0xc2	PSwitch configuration message.
SSCTRL	0xc3	SSwitch configuration message.

3.5 Channel Communication

At application level, the basic communication entity is a stream of data. A stream does not need to be limited in length, but it can be terminated after a short number of tokens has been transmitted, and can hence act as a ”packet” in a packet switched network. A stream is circuit switched and needs to be set up and terminated. If the destination channel end is local, data will be exchanged directly. If the destination channel end is on a remote tile, then the switch will first transmit a header to the other remote tile. This header sets up a circuit for the stream. After the header is transmitted, the data- and control-tokens of the stream are transmitted. When the END token is transmitted, the switches will free any resources, folding up the the circuit that was used for streaming the data. The END token will also return all communication wires to a low-power state. If a thread wants to temporarily suspend a stream it may issue a PAUSE token at any time, which will free up the circuit and return the communication wires to a low power state. Unlike the END token the PAUSE token is invisible to the receiver, it is discarded once the final switch has freed its resources (analogous to the final switch discarding the header that was sent when the stream started).

Streams can be used to stream data such as audio or video just by opening the stream and sending volumes of data. Complex data types can be transmitted over a stream by opening a stream, and serialising the data, interspersed with user defined control tokens. This allows software to be constructed defensively by using control tokens to mark known synchronisation points in the data stream. If, at any time, the receiver were to try and input data when a control token is available or vice versa, the thread will be trapped, and the program can flag or maybe recover from software errors.

By keeping streams short and synchronising often streams can also be used to exchange packets of data. The cost of setting up a stream and terminating a stream is small, and unlike traditional packet-oriented networks, the packet is transmitted while it is being constructed; this overlaps packet creation and packet reception, reducing latency.

3.5.1 Application Tokens

Application tokens are defined by the compiler or application program. Four Application Control Tokens have been predefined, and these shall not be used for any other purpose:

Name	**Value**	**Description**
END	0x01	End - free up interconnect and tell target.
PAUSE	0x02	Pause - free up interconnect but don’t tell target.
ACK	0x03	Acknowledge operation completed successfully.
NACK	0x04	Acknowledge that there was an error.

Note that these are the token values seen by the application software. When transmitted over a 5-wire link the END and PAUSE tokens are represented by special token values. All other token values can be used by the application software in any way that it sees fit.

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 use those meanings. Below 11 of those tokens are defined, all other 53 token values between 0x8b and 0xc0 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.

3.6 Sending configuration messages

The destination of a configuration message is specified by a a configuration resource identifier; this must be used to initialise a channel end using a SETD instruction in the normal way. The configuration resource identifier is a 32-bit word consisting of the following bytes:

byte	value
3	The Node Identifier of the switch to be configured
2	The SSCTRL control token (value 0xC3)
1	0
0	12

Note that the Node Identifier in a configuration message need not match the Node Identifier in the destination switch, allowing a configuration message to be used to initialise the switch Node Identifier.

The configuration messages can then be sent via the channel. A write configuration message is constructed as follows.

- WRITEC control token (value 0xC0)
- Return channel end identifier (Node, Processor, Channel-end)
- Address within switch (*address*[15 ... 8], *address*[7 ... 0])

- Data to be written (*data*[31 ... 24], *data*[23 ... 16], *data*[15 ... 8], *data*[7 ... 0])
- EOM control token (value (0x01)

This will result in the following reply message.

- ACK control token (value 0x03)
- EOM control token (value 0x01)

3.6.1 Configuring channel ends

Channel ends are configured using the SETD instruction. The SETD instruction takes a 32-bit resource-id. This resource-id must be either another channel end (type 2) or a configuration channel (type 12). The least significant 8 bits are the resource type, the following 8 bits the number of the channel-end (or in the case of a configuration special values 0xc2 or 0xc3 to indicate whether to control PSCTRL or SSCTRL), and the most significant 16 bits are the core and processor identifier.

3.6.2 Initialising the switch

The switch is configured by sending it *configuration messages*. These messages request the switch to write data to or read data from a bank of 32-bit configuration registers internal to the switch. These messages are used when booting to set the *node identifier* of the switch, associate *specific links with logical networks* and set the *speed and width of the links*, and set the *routing strategy*. Table 1 shows the registers (and the fields within the registers) that must be initialised in order to use the switch.

The addresses used in the configuration messages are the register numbers of the 32-bit registers in the switch.

The node identifier of the XS1-G4 is initialised by writing its value in 7 ... 0 of the node identifier register. The most significant 24 bits are ignored.

Each link can be associated with one of four logical networks by writing the network number to bits 5 ... 4 of the link's configuration register. These network

Purpose	Core 0	Core 1	Core 2	Core 3
Node identifier		0x5		
Link A configuration - direction & network	0x22	0x26	0x2A	0x2E
Link B configuration - direction & network	0x23	0x27	0x2B	0x2F
Link C configuration - direction & network	0x20	0x24	0x28	0x2C
Link D configuration - direction & network	0x21	0x25	0x29	0x2D
Link PA configuration - network	0x40	0x44	0x48	0x4C
Link PB configuration - network	0x41	0x45	0x49	0x4D
Link PC configuration - network	0x42	0x46	0x4A	0x4E
Link PD configuration - network	0x43	0x47	0x4B	0x4F
Link A speed - timing and width	0x82	0x86	0x8A	0x8E
Link B speed - timing and width	0x83	0x87	0x8B	0x8F
Link C speed - timing and width	0x80	0x84	0x88	0x8C
Link D speed - timing and width	0x81	0x85	0x89	0x8D

Table 1 *Register numbers for configuration registers*

numbers correspond to the network numbers used when initialising channels using the SETN instruction.

Each external link can be associated with one of eight *directions* by writing the direction to bits 10 ... 8 of the link's configuration register. When comparing the node address in an incoming message header with the XS1-G4 node address, the number of the most-significant non-matching pair of bits selects the direction to be used to forward the message. Note that the node address will is received most significant bit first, so direction 7 will be selected if the first bit received does not match bit 7 of the node address.

Before a link can be used, its speed and width must be set and it must be enabled. This is done by writing to the link's speed register. Each of the speed registers is arranged as follows:

bits	**use**
3 ... 0	minimum number of system clock cycles between tokens
11 ... 8	minimum number of system clock cycles between bits
30	width - 0: 2 signal wires; 1: 5 signal wires
31	enable link

Normally the number of system clock cycles between tokens can be set to the number of system clock cycles between bits; this bit spacing must be at least 2 clock cycles.

For a 400MHz system clock and bit spacing $x \geq 2$, the data rate achievable using 2 signal wires is $(320/x)$ Mbits/second; the data rate using 5 signal wires is $(800/x)$ Mbits/second. The actual speed that can be achieved depends on the electrical characteristics of the physical connection.

4 General configuration registers

The G4 has three types of control registers: registers in the processor itself that are accessed using GETPS/SETPS instructions, registers in the processor switch, and registers in the interconnect switch. The first set controls information private to the processor, the second group controls information specific to a processor that can also be accessed by other processors, the third one controls information related to the interconnect and the logic shared between the four processors.

4.1 Processor status registers

The following are processor status registers that are used using GETPS and SETPS. The register number must be translated to a resource ID by shifting the register number left 8 bits, and oring 0x0B in (the resource ID that identifies a processor control register).

Address			Contents
0x00	PS_RAM_BASE	RW	Address of RAM. Keep at 0x00010000.
0x01	PS_VECTOR_BASE	RW	Base of all resource vectors. Used for both events and interrupts. Bits 31-16 should be set, bits 15-0 should be kept 0.

4.2 Processor switch registers (per core)

The following registers are in the processor switch - they can be accessed over JTAG or by sending a message to the processor switch. The message to be sent is specified in Section 3.6. These registers are specific to a processor.

Address	Contents
0x00	Device ID register bits 7..0: Xcore version bits 15..8: Xcore revision bits 23..16: Node number (from SSwitch???) bits 25..24: Core number
0x01	Number of resources - I bits 7..0: Number of threads bits 15..8: Number of Synchronisers bits 23..16: Number of Locks bits 31..24: Number of Channel Ends
0x02	Number of resources - II bits 7..0: Number of Timers bits 15..8: Number of Clock Blocks
0x04	Control PSwitch permissions to debug registers. bit 0: Disable write access to processor registers. bit 8: Disable remote access, can only be cleared locally. bit 31: Disable further updates to any PSwitch register
0x05	Debug interrupts bit 0: Writing a one will generate a debug interrupt.
0x06	Processor clock speed bit 7: 0 indicates full speed; 1 indicates slow mode
0x07	Copy of the security configuration reg.
0x10-0x13	Link status PA/PB/PC/PD, see Section 3.6.2
0x20-0x27	Scratch register for debug software protocols 0-7
0x40-0x47	Copy of the PC of threads 0-7
0x60-0x67	Copy of the SR of threads 0-7
0x80-0x9F	LLink status of LLINK 0-31, see Section 3.6.2

4.3 Interconnect registers (per node)

The following registers are in the interconnect - they can be accessed over JTAG or by sending a message to the system switch. The message to be sent is specified in Section 3.6. Changing these registers has a global effect on the chip.

Address	Contents
0x00	The device ID register 0. bits 7..0: SSwitch version bits 15..8: SSwitch revision bits 23..16: Boot CTRL bits 25..24: XCore Config
0x01	Number of resources - I bits 7..0: Number of internal links per core (PA/PB/PC/PD) bits 15..8: Number of Cores bits 23..16: Number of external links
0x04	Node configuration. bit 0: Short headers (use 1-byte headers if set) bit 8: Disable PLL modifications. bit 31: Disable further updates to any SSwitch control register.
0x05	Node ID, lower 8 bits only
0x06	PLL control register. bit 4..0: INPUT_DIVISOR. Oscillator input divider value range from 1 (0x00) to 32 (0x1F). N value. bit 15..8: FEEDBACK_MUL. Feedback multiplication ratio, range from 1 (0x0) to 255 (0xFE). M value. bit 23..16: POST_DIVISOR. Output divider value range from 1 (0x0) to 250 (0xF9). P value. bit 24 VCO_RANGE. VCO operating range, 0 (250MHz $\leq$ frequency $\leq$ 500MHz), 1 (500MHz $\leq$ frequency $\leq$ 1GHz) $F_{out} = \frac{F_{in}}{N+1}\frac{M+1}{P+1}$
0x07	Lowest 8 bits set the switch clock frequency, $clk = pll/(2n+2)$. Keep at 0 for 400 MHz.
0x08	Lowest 8 bits define the relation between the reference clock and the PLL clock, $ref = pll/(2n+2)$. Keep at 2 for 100 MHz.
0x20-0x2F	External Link 0-15 direction and network, see Section 3.6.2
0x40-0x4F	Internal Link 0-15 network, see Section 3.6.2
0x80-0x8F	External Link 0-15 speed, timing, and width, see Section 3.6.2