SCI Europe number:	D211-10-310398
Deliverable number:	D2.1.1
Contractual date:	31st March, 1998
Work package:	2.1
Document version:	1.0
Document status:	complete
Confidentiality:	Consortium
Document date:	March 1998

Deliverable

SCI Test Equipment Requirement Specification

Partner: Trinity College Dublin
Author(s): B.A.Coghlan, B.O.Bakka, I.Birekli, A.Berrone, M.Gueguen, N.Laborde, K.Lochsen
Editor: B.A.Coghlan

Keywords: SCI, test, requirement, specification

Abstract:

In this document we provide both a review of the test requirements for SCI, and an informal specification for the prototype test equipment to be developed within the project. The review is provided because this is the first complete discussion of this subject. The requirement specification is thus a nominal target for further deliverables.

esprit project
25257

Table of Contents

ESPRIT Project P25257 SCIEurope

Deliverable D 2.1.1

SCI Test Equipment Requirement Specification

March 1998

Dr.B.A.Coghlan
Department of Computer Science
Trinity College Dublin
coghlan@cs.tcd.ie

B.O.Bakka
Department of Electronics and Cybernetics
SINTEF
bjorn.o.bakka@ecy.sintef.no

A.Berrone
Bull HN Information Systems Italia S.p.A.
a.berrone@it12p.bull.it

M.Gueguen and N.Laborde
Department Calculateurs et Avionique
Dassault Electronique
michel.gueguen@dassault-elec.fr

K.Lochsen and I.Birekli
Dolphin Interconnect Solutions
kl@dolphinics.no

1. Introduction

The definition of Task 2.1 is as follows :

Task	**2.1**	**Test Equipment Requirement Specification**		
Market and User Need	In order to develop SCI systems, one needs to have an infrastructure of support equipment for debugging and performance evaluation. There are needs for everything from protocol analyzers and impedance measurements to traffic generators.			
Objectives	To establish a requirement specification for a full suite of usable and needed tools.			
Approach	Collect all the different requirements from the partners into a report which specifies the main requirements for test equipment in the development, debugging, performance evaluation and support phases of a product.			
Lead Partner	Trinity	2 person months		
Other Partners	D.E. Dolphin SINTEF Bull	2 person months 2 person months 2 person months 1 person months		
Major deliverables	D 2.1.1	Q2	Trinity	Test Equipment Requirement Specification

In broad terms, the objective of this document is to outline the test requirements for the SCI fraternity. The lack of test equipment, particularly for research and development, is felt by some to be one of the restraints on market penetration of SCI. Here a wider view is taken that also encompasses manufacturing and maintenance requirements, and we also distinguish between testing at physical and logical levels.

2. Testing at a Physical Level

High bandwidth links such as in SCI require tight control of physical parameters that determine their performance and stability; here we will concentrate on electrical performance. Existing test equipment suppliers have developed three kinds of techniques for this problem:

(a) impedance measurement

(b) error rate analysis

(c) signal validation

2.1 Impedance Measurement

The integrity of any transmission line cable, such as the SCI LVDS link cables, can be verified by measuring its characteristic impedance Z_0, which is given by the familiar equation:

$$Z_0 = \sqrt{\frac{L}{C}}$$

where L and C are the inductance and capacitance per unit length. For SCI, the standard LVDS links should have a characteristic impedance of 100ohms, within a $\pm$10% tolerance.

If the impedance of a cable is not correct, then probably there is a cable fault, such as an electrical short circuit, or an impedance change, that may indicate physical damage. The location of the cable fault, and the cable and fault impedances, can be found using Time Domain Reflectometry (TDR). A pulse is transmitted onto the cable, and the time-of-flight to its return is measured. The propagation delay d per unit length is given by:

$$d = \frac{g}{jw} = \sqrt{LC}$$

where g is the phase delay per unit length. For SCI, the maximum length of a LVDS link is 10metres, yielding a two-way propagation delay of 100nSec.

For a given impedance change along the cable, the reflection coefficient C_r is:

$$C_r = \frac{Z_T - Z_0}{Z_T + Z_0}$$

The voltage magnitude $\|v_r\|$ of the reflected pulse is:

$$\|v_r\| = C_r \|v_i\|$$

where $\|v_i\|$ is the incident voltage magnitude. Obviously a small impedance change may not be sufficient to allow accurate measurements to be made. The impedance tolerance for the SCI LVDS link cables is $\pm$10%, therefore the voltage magnitude of the reflected pulse will be greater than 5% of that of the incident pulse for a cable that does not meet the SCI specification.

Commercially available Time Domain Reflectometers for copper cables include the Tektronix 1502C, which generates 200pS step pulses and can locate multiple faults as close together as 2cm, the Tektronix 1503C, which generates 2nS half-sine pulses and can locate multiple faults as close together as 30cm, and the Tektronix TS100, which generates 5nS half-sine pulses and can locate faults to within few tens of centimetres. Similar capabilities are available using plug-in TDR modules for high performance oscilloscopes, such as the Tektronix SD24 for their 11800 series, or the Hewlett Packard HP54753 for their HP54750A. For those with plenty of money, there is the Tektronix CSA803A Communications Signal Analyzer, or the similar Hewlett Packard HP83480A Communications Analyzer, each of which can serve multiple purposes.

For optical cables there are the Tektronix TFP2A and TFS3031, which can locate faults to within a few centimetres and a few tens of centimetres respectively, or the corresponding Hewlett Packard HP8147 and HPE6000A. Hewlett Packard also offers a comprehensive fibre optic test system, the HP81700. For gross faults in optical cables, visible-light fault finders, such as the Tektronix TOP300, use short-wavelength lasers to inject visible light into the cable, so that any loss of light from the cable is visible.

These measurement techniques are most commonly used for manufacturing, installation and maintenance purposes, and this is also expected to be true for SCI. The most pressing problem is that there are no commercially available adapters to connect these instruments to the standard SCI LVDS link cable, but this at least is easily remedied.

2.2 Error Rate Analysis

Ideally there are no errors at any time, but the fewer there are the harder they are to quantify. Accumulating more errors takes more time. SCI links are required to exhibit on average fewer than 1 error per 10^{12} bits transmitted, i.e. a bit-error rate (BER) of less than 10^{-12}. At 1GByte/sec, on average we will have to wait 2 minutes between errors.

As a first approximation, it is generally assumed that communications errors are caused by a Poisson process, i.e. the errors are independent and the actual error rate r does not vary. In reality bursts are common, causing the error rate to vary widely, and therefore many errors may need to be counted to average out the bursty behaviour. However, assuming this is not the case, then for N measurements of the number of errors n_i during equal time intervals T_i, the measured number of errors n_i will vary from their mean value $\tilde{n}$ with a standard deviation s :

$$s = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(n_i - \tilde{n})^2}$$

About 68% of the measured values will lie within $\tilde{n} \pm s$, about 95.5% will lie within $\tilde{n} \pm 2s$, and about 99.7% will lie within $\tilde{n} \pm 3s$. But for a Poisson process $s = \sqrt{\tilde{n}}$, thus 68% of the time the inaccuracy is within $\frac{s}{\tilde{n}} = \frac{1}{\sqrt{\tilde{n}}}$ and 95.5% of the time it is within $\frac{2\,s}{\tilde{n}} = \frac{2}{\sqrt{\tilde{n}}}$, i.e. if the mean is 10 errors, then 68% of the time the inaccuracy is within 31.6% and 95.5% of the time it is within 63.2%. Clearly for better accuracy we need to count more errors, which takes more time.

The time may be reduced by :

(a) accepting statistical upper bounds rather than actual error rates, and

(b) stressing the system.

2.2.1 Upper Bounds for Error Rates

We can only 100% accurately estimate the value of the error rate by measuring it over all time. Instead of trying to do this, why not just try to confidently establish that it is less than some upper limit ? If for a given error rate r, the probability $p(0)$ of no errors in time T is low, and yet no errors are measured, then we can be $100(1-p(0))$% confident that the actual error rate is less than r.

For a Poisson process, the probability of n errors over time T is:

$$p(n) = \frac{(\tilde{n})^n e^{-\tilde{n}}}{n!} = \frac{(rT)^n e^{-rT}}{n!}$$

Thus the probability of no errors over time T is $p(0) = e^{-rT}$, or vice versa the time T for which there is a probability $p(0)$ of no errors is:

$$T = \frac{-ln(p(0))}{r}$$

If despite this, there are no errors in time T, then we can be $100(1-p(0))$% confident that the error rate is less than $\frac{-ln(p(0))}{T}$. For example, if there are no errors in time T, then we can be 95% confident that the error rate is less than $\frac{-ln(0.05)}{T} \approx \frac{3}{T}$. Estimation of the upper bound of the error rate then reduces to determining whether the system survives for time T with no errors.

There are a number of commercially available instruments that can be used to do this. Examples include the Tektronix GB1400 and the Hewlett Packard HP71603B, both of which do a bit-by-bit comparison of the received data stream with a reference data stream that is generated either as a pseudo-random or a user-programmed sequence - the latter can be used for simulating worst-case bit patterns. In either case the output signal levels are not LVDS, so an adapter would be needed. More importantly, only a single channel of the link could be tested at a time, but given the low LVDS voltage swings (typically +/-250mV around 1.2V) and the relatively constant currents in the drivers and links, crosstalk between the channels could be expected to be quite small, and therefore testing a single channel at a time might be sufficient.

2.2.2 Stressing the System

The time T can be reduced dramatically by stressing the system in such a way as to increase the error rate. If the relationship between error rate and stress is known, then the results can be extrapolated back to normal stress levels.

The error rate for the SCI LVDS links will be a function of the signal to noise ratio. An error will occur if the superposition of noise voltage v_N and signal voltage v_S yields a result that is on the wrong side of the decision threshold voltage v_T. In these terms, the bit error rate is the probability of v_N exceeding $(v_S - v_T)$. For Gaussian distributed noise:

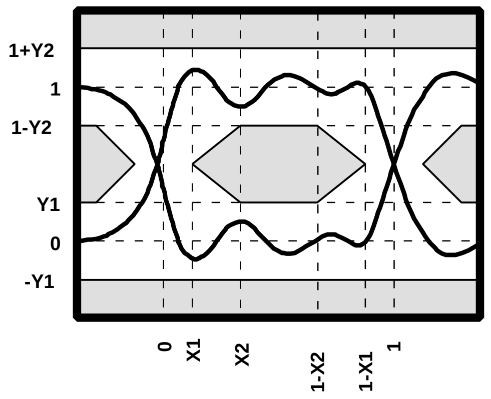
$$r = 1 - c_{norm}\left[\frac{v_S - v_T}{v_{N_{rms}}}\right]$$

where $c_{norm}[L]$ is the cumulative normal distribution $\frac{1}{2p}\int_0^L e^{-y^2/2}dy$.

As long as the noise distribution is known, attenuation is obviously a predictable form of stress, results from which can be extrapolated back to normal signal to noise ratios. A wide variety of fixed and programmable attenuators are commercially available, both electrical (Tektronix 011 series, Hewlett Packard HP84900 series) and optical (Tektronix OA5000 series, Hewlett Packard HP33300 series). Alternately, external noise with a known distribution can be deliberately introduced - the Hewlett Packard HP71603B mentioned in the last section specifically allows for this.

2.3 Signal Validation

In the above we interpreted the error rate as a function of the signal to noise ratio. This can be clearly verified on any suitable storage oscilloscope. For a given error rate, the voltage accumulated on the oscilloscope over time T will fall within an "eye mask" template that defines the relevant limits of $(v_S + v_N)$. The SCI standards document specifies the appropriate parameters $X1$, $X2$, $Y1$, $Y2$ for an example single-bit serial link:



Two sets of eye mask parameters are given, one for $r = 10^{-4}$ (i.e. an error every 10µSec for a 1Gbit/sec channel) for visual estimations, and the other for $r = 10^{-12}$ (i.e. an error every 16 minutes for a 1Gbit/sec channel) for automated measurements. The standard specifies that the eye mask for $r = 10^{-4}$ should be checked with a 10GHz storage oscilloscope, triggered by the transmit clock, over 2.10^{7} samples of random data (the document specifies a $(2^{23} - 1)$ bit pseudo-random binary sequence). No equivalent specification is given for the 18bit-wide links specified by the standard, nor does one exist for the standard LVDS links. This is a serious omission.

In principle, automated measurements could be performed with suitable single-channel digital oscilloscopes. For the more sophisticated, there is the Tektronix CSA803A Communications Signal Analyzer, which has a bandwidth of 50GHz and allows the user to define up to 10 eye masks, each with 50 vertices. The Hewlett Packard HP83480A Communications Analyzer has similar capabilities, but neither include a $(2^{23} - 1)$ bit pseudo-random binary sequence generator. They may be used in conjunction with bit error rate testers, such as the Tektronix GB1400 and the Hewlett Packard HP71603 mentioned above, to calibrate the relationship between the error rate and the eye mask.

If all 18 channels must be tested simultaneously, then this may be achieved at further expense by using an expansion chassis for a high performance oscilloscope, such as the 32-channel SM11 for the Tektronix 11800 series (on which their CSA803A is based).

It can be seen from all of the above that an adequate range of instruments already exists for testing at the physical level, and that all that is absent is a range of adapters or test jigs for interfacing to the SCI LVDS links.

3. Testing at a Logical Level

Most testing at a logical level assumes behaviour can be modelled by a finite state automaton that maps a set of discrete values A to another set of discrete values B at a discrete points in time, i.e. $F: B \leftarrow A$ such that if $X_t \in A$ and $X_{t+1} \in B$, then $X_{t+1} \leftarrow F[X_t]$. If the discrete times are defined by a regular clock interval, then behaviour is synchronous with the clock; otherwise it is event-driven.

There are a range of commercial instruments that satisfy the need for testing of the "run-of-the-mill" synchronous system at a logical level, such as the Tektronix TLA700 series and the Hewlett-Packard HP16500 series, but it is difficult to find examples that assume event-driven behaviour. Fortunately SCI and its related interfaces are entirely synchronous with defined clocks. Even so, there appear to be very few commercially available instruments that are adequate for checking its logical behaviour. Ideally there should be :

(a) timing analyzers
(b) protocol analyzers
(c) traffic generators
(d) traffic tracers
(e) traffic analyzers

3.1 Timing Analyzers

In the 'bottom-up' world of debugging a totally unresponsive system, one of the first thoughts is to check the timing of critical logic signals. Hence this is one of the most established tool categories, with a range of commercially available instruments, such as the aforementioned Tektronix TLA700 series and the Hewlett-Packard HP16500 series. While each of these are capable of far more than basic timing analysis, it remains their core facility.

SCI doesn't present anything out of the ordinary in this sense.

3.2 Protocol Analyzers

The word protocol originally meant the first leaf of a manuscript, which was glued to the corpus around which the manuscript was rolled, from *protos*, first, and *kolla*, glue. Over time it has acquired derivations in diplomatic and legal circles, but since the last century has also been taken to mean a convention. A bus protocol defines the way bus transactions are done, in a logical rather than physical sense. A protocol analyzer checks for violations of the protocol, which can lead to anomalies. It continuously monitors, or "snoops", the logical sequence that represents bus actions, and reports any deviation from the standards specification. An example might be the deactivation of the SCI F signal at an incorrect point within a packet. Many protocol analyzers also check for violations of the timing of the signals that delineate these sequences, as although this function is more properly called timing analysis, it is often a fine distinction.

Generally protocol analyzers are attachments to traffic tracers, where the protocol violations are saved into trace memory in some distinct way, since in this case the protocol analyzer only needs to recognise the violations, which it can do via a set of rules, and trigger the tracer. The rule set can be directly derived from the standards document, so apart from allowing for users to mask out rules, the functionality can be fixed. Two natural rule sets arise, that mirror the natural subsets for traffic tracers, as we will see later:

(a) those for protocol analyzers that snoop on the SCI electrical or optical cable.

(b) those for protocol analyzers that snoop on the LinkController local bus.

The first of these presents a formidable challenge. For SCI, the protocol and format definitions are intertwined and account for nearly 60% of the standards document, so the rule set could be very large. The second is really a local bus protocol analyzer, operating within a protected environment, but is equally necessary for the SCI community. Neither is commercially available.

3.3 Traffic Generators

We have already discussed a number of test procedures at the physical level, such as error analysis, where there is a need to generate a fixed amount of deterministic data. The same requirement extends to the logical layer, except that the information will be of a more structured form.

Systems usually get larger, and then larger again. Unfortunately, with logic analyzers, traffic tracers and traffic analyzers, the network can only be analyzed at one given point. For large systems, complex topologies, and protocols like SCI that do not broadcast packets to all nodes, even just within a ringlet, comprehensive analysis becomes very difficult. As a result it is useful to employ equipment that can independently generate and receive large amounts of data at high speed. Such a device is called a traffic generator.

A traffic generator can fulfil different needs:

(a) stress testing

(b) performance evaluation

(c) surveillance of latency and hot-spots

Let us look at each of these in turn. Firstly, in SCI, as well as in other interconnect technologies, different mechanisms are used, depending on the sequence and frequency of events. System components react differently to a packet when there is no traffic in the system, as opposed to when several other packets compete for the same node and/or all the queue buffers are filled to their limits. Errors are more likely to occur in these circumstances, or as a result of rare sequences; identifying and correcting these kinds of errors are tough tasks. Furthermore, in a system like SCI it is extremely difficult, if not impossible, to exactly repeat a given complex traffic pattern. Traffic generators are specifically intended to stress the system so that these rare sequence errors show up regularly.

Secondly, performance evaluation implies that the system under test is driven towards saturation, and then the correlation between throughput and latency is measured. This method is regularly used in performance simulations, where the models are loaded with traffic from some statistical source. Therefore, in order to perform realistic performance evaluations, the traffic generator must be able to generate so much traffic that some part of the system (e.g. a ring, or a switch) is saturated, and it should support the associated latency and throughput measurements, either directly or indirectly. Support for different traffic profiles would be a distinct advantage. Once this is in place it becomes feasible to characterise systems by throughput and latency functions for a range of topologies and traffic profiles, as well as to calibrate simulators.

Finally, when a system is put to work, it is possible to configure it in different ways. Usually there is a choice between a ring and/or a switch, a choice of switch topology, and so on. In order to qualify the configuration, it is usual to construct an application benchmark that will indicate the best choice. In real life, however, the workload may change significantly from time to time, and may create hot spots in certain parts of the network that might not be detected by such benchmarks or static tests. A traffic generator can be used to highlight these conditions by periodically sending packets through the network and measuring latency, whereupon proper control and management schemes can be invoked to reconfigure either the network or the applications to avoid the congestion.

A traffic generator device can be constructed in different ways, either:

(a) as a standard computer running standard applications, or

(b) as a low cost PC running a special traffic generator application, or

(c) as a dedicated traffic generator device.

Standard computers running standard benchmark applications will be efficient for small configurations, but will be relatively expensive for a large system test, and will also heavily involve application and system software overheads, so that the impact of the network relative to deviations in application performance will be difficult to measure. A dedicated traffic generator device is the preferred tool for large system topologies, always assuming that the unit cost is relatively low compared with the other alternatives. The penalty is that a separate control and management system must then be designed in order to handle all the data.

3.4 Traffic Tracers

Two subsets arise:

(a) tracers that snoop on the SCI electrical or optical cable.

(b) tracers that snoop on the LinkController local bus.

Each should:

(a) work at the full data rate.

(b) accommodate a wide variety of SCI configurations.

(c) be cost-effective.

Their modus operandus is to selectively respond to some user-defined circumstance by collecting user-designated traffic from the cable/bus, then to either display the traffic online or store it for later analysis.

3.4.1 Tracers that snoop on the SCI electrical or optical cable

SCILab have recently taken over the LinkProbe [Birekli 98] originally developed in 1992 by Inge Birekli at Dolphin, and intend to market it as a commercial product. This instrument snoops on the SCI LVDS electrical cable and converts the traffic to an approximately 80bit wide signal stream for attachment to a Hewlett Packard HP16500 Logic Analyzer. It is structured as two units: a small probe unit that inserts into the SCI cable, and the larger LinkProbe proper that connects to the probe via a cable and to the HP16500 via several other cables.

The probe monitors the 18bit LVDS signals at 250MHz[1] (500MB/s), converts them to PECL levels, and outputs them to the LinkProbe proper. PECL is ECL shifted to +2V above ground.

The LinkProbe samples the symbol data at a precise offset from the F and C signals. The offset is either derived automatically from the transitions of C, or is manually set in fixed steps as an offset from the transitions.

The LinkProbe then filters out IDLE symbols and assembles groups of four symbols into left justified quad symbol groups (QSGs), for which it generates QSG attribute codes that distinguish between:

(a) IDLE or non-IDLE symbols

(b) packets without IDLEs

(c) packets with 4 IDLEs

(d) SYNC packets

(e) ECHO packets

(f) the first QSG of a packet

(g) the last QSG of a packet

(h) eight intermediate QSGs of a packet

[1] The SCI clock rate is twice the frequency of the C signal waveform (i.e. the discrete clock interval is half the period of the waveform), since information is clocked on both transitions of the waveform.

(i) framing errors

The LinkProbe outputs the 64bit QSGs plus approximately 16 other signals, including the QSG clocks and attribute codes, to the HP16500 at a quarter of the SCI clock rate (62.5MHz for a 250MHz SCI clock). The left justification with respect to the first symbol of a packet ensures that the various bit fields of a packet always appear in fixed positions at the logic analyzer, so that its triggering can be set up accordingly.

These tracers will see the request packets upstream of their destination nodes, and the response packets downstream of their source nodes. Thus at any one time a tracer will only see a proportion of the traffic of interest.

3.4.2 Tracers that snoop on the LinkController local bus

The motivation for this approach is a need to know what is happening at the system level. The best that can be done at present is to use a standard logic analyzer, such as a HP16500 with several HP16550A 100MHz 102-channel modules, each with 4ksamples per channel. There is a lack of more substantial trace memory, and of SCI-specific triggering mechanisms.

The most widely used LinkController local bus is the Dolphin B-Link for their LC-1 and LC-2 LinkControllers. Next most widely used would be their somewhat similar C-Bus for their NodeChips. Both of these are 64bit packetised busses, with the 16bit SCI symbols automatically left justified within their 64bit packets. The B-Link currently operates at 50MHz, but this will be increased to 100MHz in the next generation of LinkControllers.

One might suspect that the behaviour of the LinkController local bus would be very dependent upon the overall system architecture, and in conflict with the desire for tracers to accommodate a wide variety of SCI configurations. To evaluate this, let us consider two computers that are at the opposites poles in an architectural sense:

(a) a cache coherent SCI system, and

(b) a message-passing SCI system.

Both cases assume that there is a clean electrical environment at the local bus that will not contribute any uncertainty.

*3.4.2.1 Tracers that snoop on a **cache-coherent** LinkController local bus*

In the first case a 4-way SMP is bridged from the SMP bus to a Dolphin LC-2 (eventually a LC-3) LinkController on the B-Link, with cache coherence control within the SCI bridge. There is a separate bridge from the SMP bus to a PCI bus; the SCI bridge plugs into this, but only for DC power. Up to eight such SMP nodes can be interconnected by SCI in this way.

Each 64byte cache line has a home node. Each node contributes 3GBytes to the global address space, i.e. 48million cache lines. The cache directory appears to take a similar form to that in Stanford DASH [Lenoski 92], with an 8bit field per cache line, where each bit i indicates whether $node_i$ is involved with the cache line. The bridge contains a 10MByte partial cache directory (a full map would require 48MBytes). The ownership is fixed: the home always owns its cache lines.

Instead of using a subset of the SCI cache coherency mechanisms, a proprietary cache coherency policy (presumably based upon the Stanford DASH protocols and optimizations) is enforced by the bridge using non-coherent SCI transactions. Any cache transaction can involve up to three parties: the requester, the home node (the owner), and the node currently using the cache line. Even if only these three nodes were ever users of the cache lines of interest, three synchronized tracers would be needed, one per node.

The perceived need in this case is to be able to trigger on an event, principally a sequence of cacheing actions, around which B-Link traffic is collected, with reasonable control over the position of the event within the collected data. The primary focus is on cacheing activity, so the items of most interest are the event itself and its descendants, and their effect on their peers. Consequently the traffic can simply be collected until the trace memory is full, without filtering. In a similar vein, the trace data will require careful picking over to understand the cache behaviour, so it is unlikely that online analysis is necessary. It would be more likely that the trace data would be transferred elsewhere, over a network, for filtering and analysis.

The toughest cache coherency problems manifest themselves as infrequent anomalies in policy enforcement. Often the sequence that leads to such an anomaly can be quite tortuous and unexpected. This manufacturer envisages that each trigger should encompass up to sixteen 64bit B-Link symbols, i.e. 1024bits, and that up to four of these should be able to be composed into a 4-level triggering sequence, with a sequence counter per level, and moreover, that the sequence should be able to backtrack over up to four 64bit B-Link symbols, i.e. 256bits. The tracer triggering has to be flexible enough to allow definition of this - in the limit it has to be totally programmable. The manufacturer has proposed that the triggering should be done by an uncommitted block of programmable logic in a FPGA[2], and that the sequence should be defined as a VHDL[3] program that is loaded into the FPGA at the time of measurement. This is a very attractive approach.

Given the lack of real-time filtering of the trace data and the possibly tortuous triggering sequence, a large trace memory is required; the manufacturer would like to be able to analyze up to 8million 64bit B-Link symbols, which requires 64MBytes of trace memory.

In addition, since there are up to eight nodes, up to eight tracers must be able to co-ordinate their triggering, so that a system-wide trace can be collected that is correlatable in time across the whole system.

[2] FPGA: Field Programmable Gate Array
[3] VHDL: Very-high-speed Hardware Description Language

*3.4.2.2 Tracers that snoop on a **message-passing** LinkController local bus*

At the opposite architectural extreme, a uniprocessor is bridged to a PCI bus, and thence via a Dolphin PSB[4] to a Dolphin LC-2 (eventually a LC-3) LinkController on the B-Link. This is for a safety-critical application, therefore sharing is excluded in order to maximize fault isolation. Thus there is no cache coherency - instead an "exchange memory" in PCI space is used for message-passing communications. The processor may tell the PSB to transfer data to a remote exchange memory, and it will attempt to do remote writes. If the remote PSB is configured to allow this, then it will become the PCI master to do the writes.

The major problem in this system is to verify that the behaviour of the software always remains safe, despite these complex interactions. This requires symbolic tracing of message-passing actions. The problem is eased somewhat if, as in this system, the environment is statically assigned at compile time, so that the mappings from logical to physical to PCI to B-Link space are fixed.

These tracers will see the request and response transactions relevant to the local node, but not for any other node. Nor will they see any of the raw packets or subaction packets generated by the LinkController. Thus in a N node system, at any one time the tracer will only see traffic for $\frac{1}{N}$th of the system. Since system behaviour is the particular subject of interest, again a tracer will be needed for each node, and they must be able to co-ordinate their triggering in a correlatable way.

Also as above, there is no need for online analysis, since it is expected that the trace data would be transferred elsewhere, over a network, for analysis on a workstation or PC. In this case, however, the need is to trace behaviour at a higher level, and so there is a potential for even larger data sets unless the data is filtered at acquisition time. The manufacturers perceive a need to collect filtered data around a user-definable event, where the position of the event within the trace should also be user-definable. Pragmatically, they feel that both the filter and trigger should be able to be composed from:

(a) SCI command and control fields (mandatory),

(b) SCI address fields (very desirable), and

(c) SCI data fields (optional but desirable).

Again there is a need to "develop" an event as a complex sequence, and a need to be able to backtrack from a failed predicate within the sequence. Apart from the requirement to filter at acquisition time, the requirements are remarkably similar to those above. One difficulty is that filtering leads to a loss of temporal reference, but this can be resolved by timestamping the data. It is also difficult to say how deep the resulting trace will be, since that will depend upon how effective the filtering is.

[4] PSB: PCI-to-SCI Bridge

Yet another difficulty is that the actions on the B-Link are decoupled by buffers from those on the PCI bus. Even with the assumption that both busses are finite state machines that are synchronous to designated clocks, their mutual asynchronicity makes their interactions difficult to track. This could be accommodated by either tracing an external "event identifier" from a PCI bus analyzer, or generating one for a PCI bus analyzer to trace. The latter is probably the easier approach, since at least one commercially available PCI bus analyzer[5] allows for this. Allied to timestamping, this becomes an attractive principle, since it can be generalised to enable correlation with any asynchronous higher-order events.

Thus it seems that a common specification can be formulated, even with the profound architectural differences above, for tracers that snoop on a LinkController local bus. Whether they can be cost-effective remains a moot point.

3.5 Traffic Analyzers

When applied in the context of a communications network, the term "traffic analysis" usually denotes a statistical analysis of the kinds of payloads the network is carrying, where they are going, how much data is involved and how efficiently it is transported, utilization, queue lengths, quality of service, and etc.. But traffic analysis is an elastic band that stretches to encompass the user's needs. In a message passing computing system it might include some understanding of the messaging layer; in a symmetric multiprocessor it may encompass the cacheing behaviour. SCI presents a large increment in the complexity of the interactions being handled directly by the hardware, since it includes all three of the above.

The big question is, just what do people want to see ? There is a general desire to know just what the system is doing, and how well, for optimization, debugging, manufacturing qualification, online health checks and maintenance, that can best be satisfied by direct measurement. Each of these requires information at a number of levels.

At the network level, analysis might allow determination of:

(a) the distributions for request packets of each type, the number rejected,

(b) the distributions for each type of response to each type of request packet,

(c) the distributions for sources and destinations.

At the cache-coherency level, analysis might allow determination of:

(a) distributions per node for invalidations, updates, purges, synchronizations,

(b) coherence tree navigations and maintenance,

(c) cache model parameters (state probabilities, transition probabilities, etc.)

[5] The VMetro PBT-315 allows an 8bit external input to be traced. The external input is connected via a header.

(e) efficiency of policy enforcement, management of ownership,

(d) livelocks, deadlocks.

At the memory reference level, analysis might allow determination of:

(a) address distributions,

(b) per-processor address and data traces,

(c) system-wide symbolic traces.

While statistical results may be most useful for optimization, they can be surprisingly good indicators of misbehaviour, and therefore useful also for manufacturing qualification, online health checks and maintenance. Debugging, however, is different: it requires determination of why the system is not behaving as it should, rather than whether it is, and really does require memory reference level information, preferably symbolic and system-wide. For most systems this is probably a long way off, but for the statically assigned environment described above it seems ambitious but possible.

From all of the above we may conclude that there are many commercially available timing analyzers, and that an instrument is being developed by commercial interests for traffic tracing from the SCI link, but that no instruments are available or planned for any of the other categories.

4. Test Equipment Requirement Specifications

Here we will attempt to define a specification for the following instruments, which are not readily available as commercial products:

(a) a protocol analyzer

(b) a traffic generator

(c) a traffic tracer for the B-Link

(d) a traffic anlayzer

4.1 Protocol Analyzer Specification

The SCI standards document specifies a number of protocol violations for which the LinkController is required to respond by indicating an error condition. Given that the LinkController filters those violations, and that the B-Link is designed as a realization of the SCI “transfer cloud”, one may concentrate on detecting further violations at the B-Link. As noted before, protocol analyzers are usually attachments to traffic tracers, where they only need to recognize the violations via a set of rules, and trigger the tracer. In the context of the B-Link traffic tracers discussed above, where it is suggested that the triggering is defined as a VHDL program that is downloaded into a FPGA at the time of measurement, the protocol violation rule set could be defined as part of the triggering program. This is the approach that will be adopted here.

4.2 Traffic Generator Specification

There is a need for relative inexpensive modules capable of sending and receiving large amounts of data and at the same time collect vital statistics, in order to test and qualify large networks.

SINTEF are developing a traffic generator that can send a specified number of types of packets to the SCI network. The implementation is expected to be a small standalone instrument that features realtime generation at 500MB/s. Each type of packet will be able to be sent a given number of times, at a given interval. The procedure is likely to be:

(a) the user defines J packets to be generated, AND

(b) the user also specifies when packet generation is to start and end, THEN

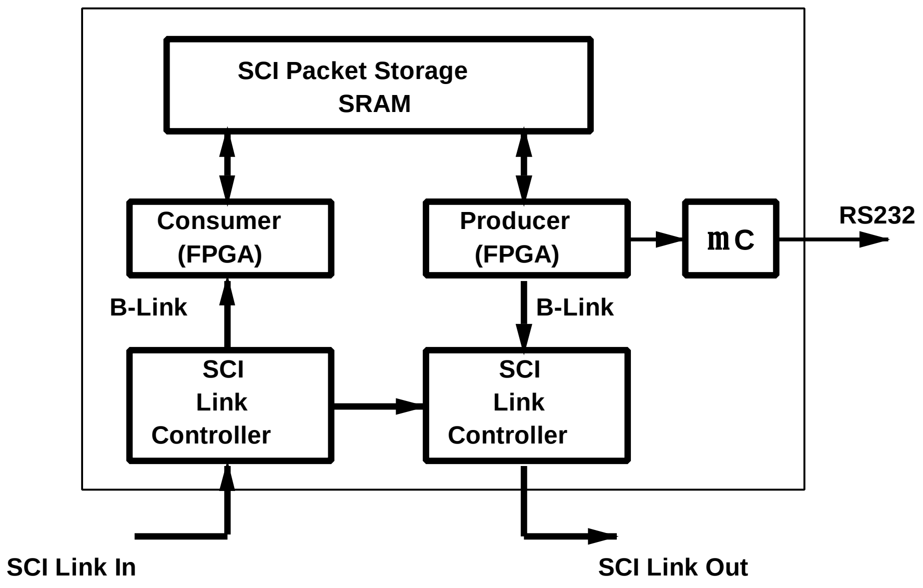
(c) $packet_i$ is sent N_i times at T_i intervals, for $1 \le i \le J$.

A pure traffic generator would do no more than this. It would be the task of some other instrument to receive the data, sift it, and report the results to the user. Most such instruments, however, are designed to closely co-operate with the traffic generator, synchronizing to it and receiving reference data streams from it. Since there is likely to be very little added cost, SINTEF's instrument is likely to include some raw data analysis functions, such as:

(a) counting the received packets, both in total and by type, and

(b) reporting the counter values to the user at regular intervals.

A block diagram of the proposed traffic generator is shown below. It will insert into the SCI network with two LinkControllers, one for the transmitter and one for the receiver. The transmitting LinkController's local bus (the B-Link) will connect to a Packet Store SRAM via a FPGA. Another FPGA will provide the necessary support for the receiving LinkController, so that packets will be counted and the values stored in the SRAM. Reporting will be done by including the counter values in a SCI packet in the Packet Store; this will then be transmitted to a special node. The user will exercise control over the instrument, and receive results, either via the SCI network or via a RS-232 serial link. The actual specification is the subject of Task 2.2, and the result will be Deliverable 2.2.2.



4.3 B-Link Traffic Tracer Specification

As indicated above, SCILab have already declared an intention to develop an instrument to trace SCI traffic by snooping on the SCI LVDS electrical cable. In due course it can be expected that it will be extended to the optical cable by some form of optical-to-electrical conversion. In contrast, we have no knowledge of any plans outside the SCIEurope consortium to develop a tracer to snoop on the LinkController local bus, such as the B-Link, so here we will concentrate on this.

From the previous discussion of B-Link traffic tracing, we may compose the following general specification:

(a) Trace up to 8million 64bit B-Link symbols, using 64MBytes of trace memory.

(b) Collect filtered data.

(c) Timestamp the filtered data.

(d) Trigger on an event, principally a sequence, around which B-Link traffic is collected, with user-definition of the position of the event within the trace.

(e) Compose both the filter and trigger from:

(1) SCI command and control fields (mandatory),

(2) SCI address fields (very desirable), and

(3) SCI data fields (optional but desirable).

(f) Trigger on up to sixteen 64bit B-Link symbols, i.e. 1024bits, and allow up to four of these to be composed into a 4-level triggering sequence, with a sequence counter per level, and allow the sequence to backtrack over up to four 64bit B-Link symbols, i.e. 256bits.

(g) Implement the triggering in an uncommitted block of programmable logic in a FPGA, and define the sequence as a VHDL program that is loaded into the FPGA at the time of measurement.

(h) Allow up to eight tracers to co-ordinate their triggering, so that a system-wide trace can be collected that is correlatable in time across the whole system.

(i) Generate an "event identifier" for an external analyzer to trace.

(j) Transfer the trace data elsewhere, over a network, for filtering and analysis.

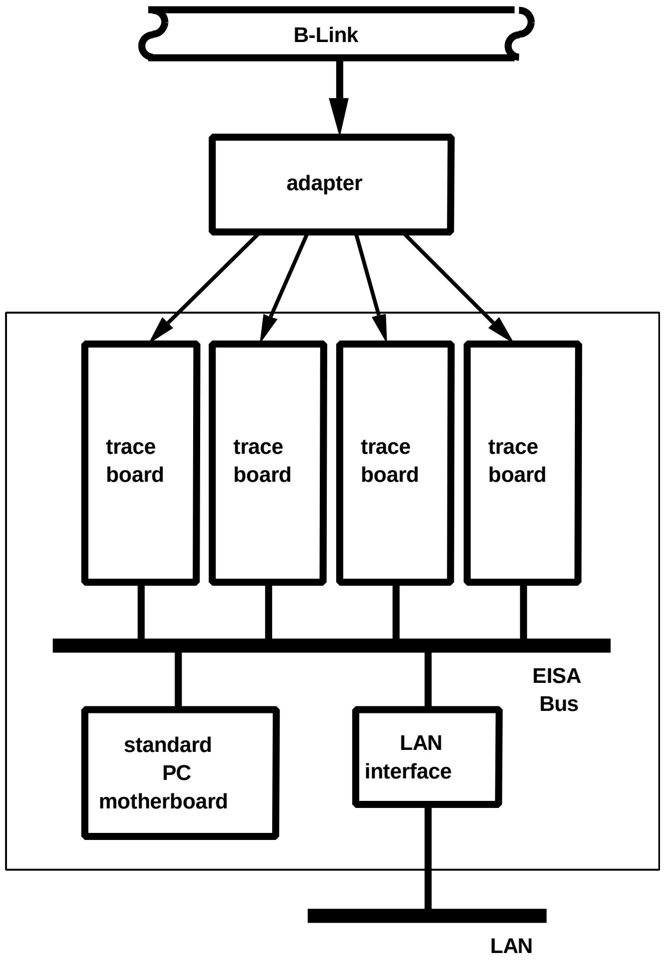
To this end, Trinity College Dublin are developing a B-Link tracer as part of their Traffic Tracer/Analyzer. The implementation is expected to be a "luggable" instrument that features realtime tracing at 500MB/s. It will contain a substantial amount of trace memory, and will be web-enabled for remote management. The instrument will contain an embedded PC that will execute either Windows95[6], Windows-NT[7], Mach3.0[8] or Linux[9], although at this stage it appears Windows95 might be the best candidate. The actual specification is the subject of Task 2.2, and the result will be a prototype that will represent Deliverable 2.2.1 and a Mk.II version that will represent Deliverable 2.2.3. A block diagram is shown below.

Each trace board will store 2 million 48bit samples in 12MB of dual-ported VRAM. The maximum sample rate will be 50MHz on the VRAM serial port, i.e. a maximum data rate of 300MB/s. A 64k x 12bit trigger SRAM will give a minimum of four arbitrary map-based triggering ranges, and an inter-board connector will allow multiple boards to be cascaded to increase the sample width in multiples of 48bits, up to 192bits for the four boards shown. It will also be possible to

[6] Trademark of Microsoft Ltd.
[7] Trademark of Microsoft Ltd.
[8] Courtesy of Carnegie Mellon University.
[9] Shareware version of UNIX.

interleave boards to increase buffer depth in multiples of 2 million samples, with the by-product that this will also increase the maximum sample rate. A 48bit 50MHz counter will allow timestamping of samples.

Four boards will be able to store 2 million 192bit samples in 48MB of trace memory. The trace memory will be locatable anywhere within the PC memory address space. The EISA system bus port will operate independently as a 33MB/s bursting slave. If the aggregate acquisition rate is less than this, then data could be read out transparently and asynchronously without interrupting the tracing. Otherwise the 48MB of data will contain 32MB of data, 4MB of attributes and 12MB of timestamps, where the 32MB limit on data represents just 80mS of B-Link traffic at 50MHz.



The major issues here are:

(a) is the trace storage real-time ?

(b) are there too many boards ?

(c) can the prototype be finished by Q12 ?

(d) should the software be a priority ?

(e) should elegance be sacrificed until Mk.2 ?

It is expected that a Mk.2 tracer/analyzer would give:

(a) 100MHz tracing on the B-Link.

(b) more depth.

(c) better triggering.

(d) fewer boards.

(e) a PCI form factor.

Bull Italy are developing, for internal use only, a B-Link traffic tracer that encompasses much of the general specification given above. It is being designed to trace up to 8million 64bit B-Link symbols using 64MBytes of trace memory, triggering on an event, around which B-Link traffic will be collected, with user-definition of the position of the event within the trace. It will trigger on up to sixteen 64bit B-Link symbols, will allow up to four of these to be composed into a 4-level triggering sequence, with a sequence counter per level, and will allow the sequence to backtrack over up to four 64bit B-Link symbols. It will implement the triggering in an uncommitted block of programmable logic in a FPGA, and will define the sequence as a VHDL program that is loaded into the FPGA at the time of measurement. It will allow up to eight tracers to co-ordinate their triggering, and will transfer the trace data elsewhere, over a network, for filtering and analysis. The composition of the triggering field is not yet well defined, and it is not being designed to filter or timestamp data.

Between these two efforts there can be some confidence of a good result. The current proposal is that they will use common triggering mechanisms, trace file formats and trace analysis software.

4.4 Traffic Analyzer Specification

As noted above, Trinity College Dublin are developing a tracer as part of their Traffic Tracer/Analyzer - the other half of this is a traffic analyzer. This will be composed of several software tools, targeted to execute on either Windows95 or Windows-NT platforms. It will include both the management and acquisition clients for the web-enabled traffic tracer, as well as post-acquisition analysis tools for optimization, debugging, manufacturing qualification, online health checks and maintenance.

The basic analysis tools will interpret the B-Link traces to extract SCI packet information. It will automatically accommodate the timing of the B-Link, for example, it should correctly interpret the various arbitration cycles, and it should correctly associate the address cycles with the 3-cycle-delayed *here* and *busy* responses. It should also intelligently interpret the information, for example, it should allow recognition of retried B-Link actions as single logical operations. It will allow for manual navigation of the trace data, and statistical evaluation of it.

Trinity intend, however, to explore the development of more ambitious tools, based on the use of analytical models to aid the understanding of the trace data. The effects of cache coherence on a network can be examined analytically. The intended analytical approach is that taken by Isi Mitrani

[Mitrani 97, Coghlan et al 98], who has shown that if the transitions of cache states from one to another are described using a Markov matrix, their equilibrium distribution can be solved via a fixed point equation. The mean bus service time can be determined using this, which enables various metrics of the system to be evaluated. This has been expanded for a simple model of SCI in [Fields et al 95], and then elaborated on in [Bennet et al 96] to deal with more complicated systems by increasing the number of classes of bus operations which can occur as well as simply having a greater number of cache states. These models will be further extended, with a view to incorporation with neural nets, evolutionary algorithms and predicate logic into a framework for guided analysis of the trace data. This is a speculative agenda, and no expectations are in order.

The actual specification of the Traffic Analyzer software is the subject of Task 2.2. The result will be integrated in preliminary form into the prototype that will represent Deliverable 2.2.1 and in final form into the Mk.II version that will represent Deliverable 2.2.3. This is intended to provide the post-acquisition analysis for both the tracers from Trinity and Bull.

5. References

[Birekli 98] **Birekli, I.**, SCI LinkProbe User's Manual, *Dolphin Interconnect Solutions*, January, 1998.

[Bennet et al 96] **Bennet, A.J., Fields, A.J., and Harrison, P.G.**, Modelling and validation of shared memory consistency protocols, Vol.27 and 28, *Performance Evaluation*, 1996.

[Coghlan et al 98] **Coghlan, B.A., and Fabregat, G., Ed.**, *FASST: A Fault Tolerant Architecture using Stable Storage Technology,* in preparation.

[Fields et al 95] **Fields, A.J., and Harrison, P.G.**, A Fixed-Point Model of a Distributed Memory Consistency Protocol, In: *Quantitative Methods in Parallel Systems*, Baccelli, F., Jean-Marie, A., and Mitrani, I., Ed., pp.237-247, Springer Verlag, 1995.

[Lenoski 92] **Lenoski, et al**, The Stanford Dash Multiprocessor, *IEEE Computer*, pp.63-79, Vol.25, No.3, March 1992.

[Mitrani 97] **Mitrani, I.**, *Probabilistic Modelling*, Second Edition, Cambridge University Press, 1997.