Monitoring and Analysis of SpaceWire Links

James P Lux[1], Frank Loya[1], Barry M Cook[2], Paul Walker[3]

[1] *Jet Propulsion Laboratory*
4800 Oak Grove Avenue,
Mailstop 161-213
Pasadena CA 91109
USA
Email:james.p.lux@jpl.nasa.gov

[2,3] *4Links Limited*
The Mansion,
Bletchley Park,
Milton Keynes,
MK3 6ZP, UK
Email:[2] barry@4Links.co.uk
Email:[3] paul@4Links.co.uk

INTRODUCTION

A breadboard scatterometer (a type of orbiting radar used for measuring vector ocean winds) was constructed at Jet Propulsion Laboratory to investigate the feasibility of using general purpose programmable Digital Signal Processors (DSP) to replace the special purpose hardware used in the previous successful instruments[1]. The architecture used multiple processors to reach the required computational performance with the available space qualified processors (Analog Devices 21020 family). Three Astrium MCMDSP modules were used in the breadboard, each of which integrates the CPU, memory, peripheral controls, and a SMCS332 high speed interface. The latter was used to implement SpaceWire links between the processors. The breadboard successfully demonstrated a scalable multiprocessor approach for this type of instrument.

Early in the development effort it was realized that monitoring of the messages being passed over the SpaceWire links would be essential, particularly for development of the low level message passing drivers, as well as the higher level algorithms. In particular, there was a need to match individual messages with specific Radio Frequency (RF) pulses transmitted or received by the breadboard radar, requiring a real-time capability that could not be conveniently met with software based solutions in the limited resources of the DSP. The need to make real-time performance and timing measurements required that the monitoring approach be entirely passive (as opposed to a decode and retransmit approach) Figure 1 shows a block diagram of the breadboard.

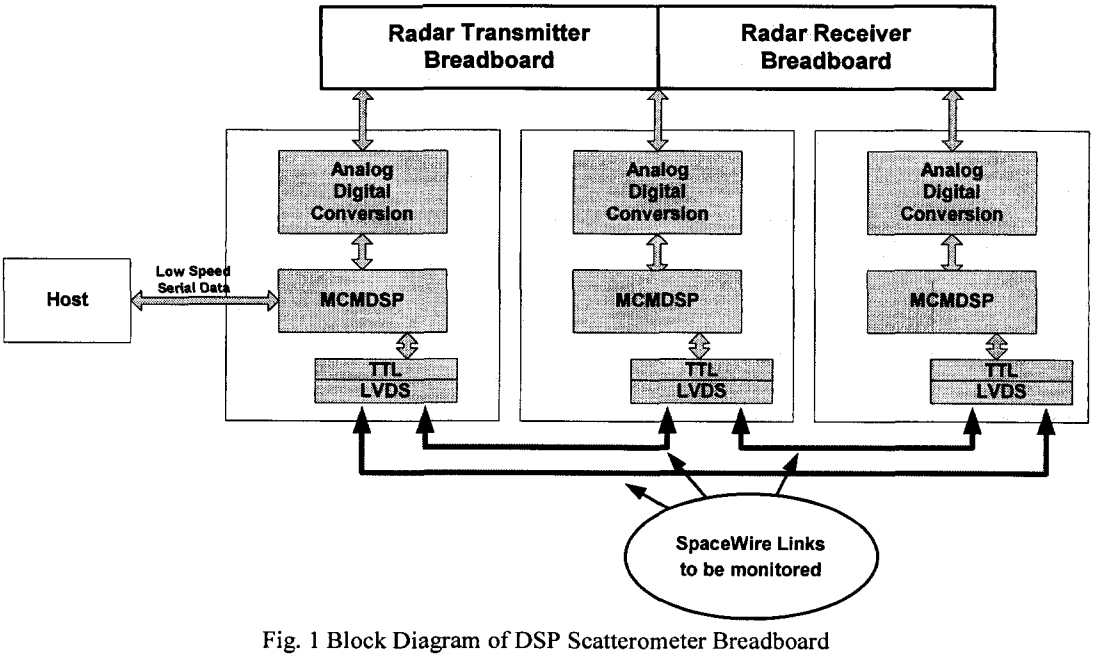


Fig. 1 Block Diagram of DSP Scatterometer Breadboard

Monitoring with Standard Lab Instruments

The initial efforts to monitor the links used differential high impedance probes and a digital storage oscilloscope/logic analyzer, which was satisfactory at first (it is easy to recognize NUL and FCT tokens after a bit of practice). When the monitoring need moved to needing higher level information beyond "is the link running?", the limited capture memory, the high speed signals, and the lack of a high level interface prompted further development. One of the authors (Loya) identified 5 characteristics of the SpaceWire protocol design that made possible a simple hardware implementation using a small Finite State Machine to decode the messages. Table 1 lists the properties and their implications.

Table 1 – SpaceWire properties enabling passive monitoring and decoding

SpaceWire continually transmits bits at a constant rate (this is accomplished by filling in gaps with Null characters).	allows us to derive formulas that can determine when a character occurred. For example, given the character sequence $\langle C_i \rangle$, $i = 0 \cdots n$, and the time that C_0 occurred (call it T_0), then the time that C_i occurred is given by: $T_i = T_0 + \tau \sum_{j=1}^{i} \#_b(C_j)$ where τ is the bit time, and $\#_b$ is a function that yields the number of bits in C_j
If either the D or S lines on either end (receive, transmit) of the link fails to make a transition within 850 ns., the link will be broken, and both sides of the link will reset (i.e. stop transmitting for a preset time interval).	assures us that property 1 will be enforced.
The SpaceWire bit stream contains no framing or error-correcting bits	imposes a "zero tolerance" condition on bit error. Since no characters are used for framing or error correction, it is assumed that each bit will contribute either to the content or attributes of a specific character.
The SpaceWire bit stream always contains enough error detection bits to detect any single bit error.	assures us that if a single bit error does occur, it will be detected. The SpaceWire data stream always contains between 10-25% odd parity bits (as attributes of a character). When a parity error is detected link is reset. This assures us that no corrupt data will gain acceptance by the receiving node (i.e. a detected reset implies that the incoming data packet should be discarded).
The separation of the SpaceWire bit stream into characters can be done in a single pass	allows the separation of the bit stream into characters with the application of "straight forward" hardware.

Using the SpaceWire-PCI as a Monitor Receiver

The SpaceWire-PCI card, in a suitable realtime programming environment, can provide a significant fraction of the monitoring functionality that would be provided by the fully idealized monitor embodied in the state machine described above. This approach had the significant advantage that it required almost no hardware development and there was a suitable set of API calls available to integrate the card into a realtime computing environment.

Figure 3 shows two ends of a SpaceWire link, N1 and N2, where Tx and Rx represent the DS signaling pair transmitter and receiver, respectively; each arrow represents 4 physical wires. Two additional SpaceWire nodes, N3 and N4 are set up with their receivers fed by the Tx side of N1 and N2 respectively. The Tx side of these monitoring nodes are terminated into dummy loads (*i.e.* 100 ohm terminations). Since a SpaceWire-PCI card has 3 link ports, it is convenient to use two of them as N3 and N4, leaving the third for other uses. The SMCS332 used in the SpaceWire-PCI treats each channel independently, so it is possible to program 2 links for monitoring, and the third for conventional SpaceWire traffic.

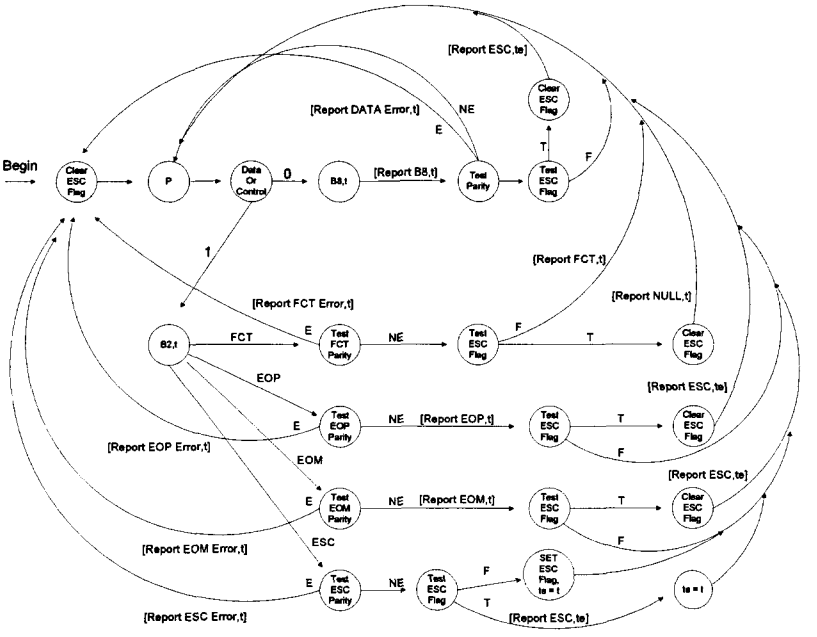


Fig 2 – Mealy State Machine for Decoding Spacewire Packets

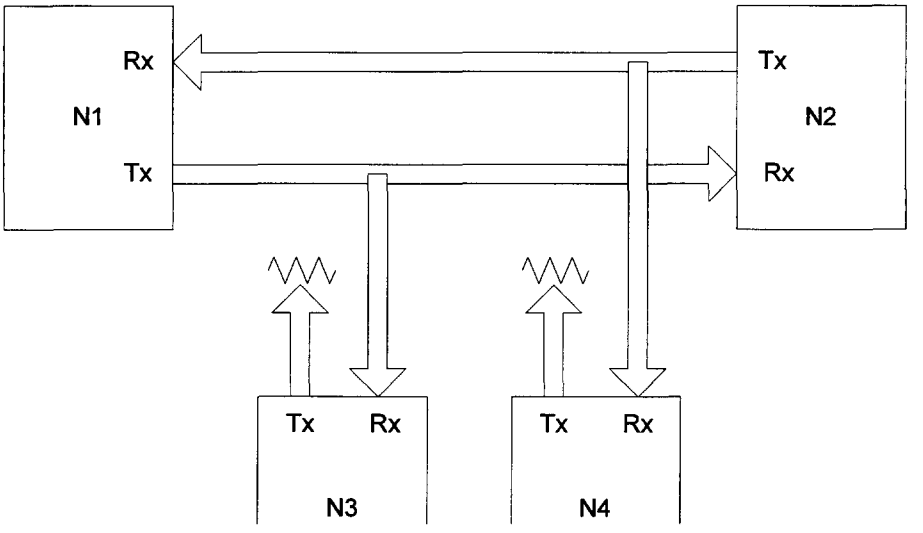


Fig 3 – Using Spacewire Links for Monitoring

When the link N1:N2 is started, both monitor nodes will see the NUL tokens and establish pseudo-links N1:N3 and N2:N4. If an error is detected by either of the link nodes (N1,N2), the reset protocol will be performed. The monitor nodes will also reset, and the pseudo link will re-establish when the N1:N2 link is established.

If an error occurs on a monitor link (*e.g.* N1:N3), it will also attempt to reset, but the resynchronization is a bit more complex. If the node can resynchronize with the running stream, the monitor node will sync up when it sees the parity bit of a NUL token sequence. If the primary link is generating a pure NUL stream (i.e. the link is "idle") the probability of landing on the correct bit (by chance) is 1 in 8. The SpaceWire State diagram (referred to in [2]) suggests that the

link reset-error detect-reset sequence takes about 25 microseconds. A monitor node can therefore attempt about 4000 of these sequences in a second. Even if the running stream is not a pure NUL sequence, periodic NULs are almost certain to occur, and eventually, the monitor node will sync up.

In practice, the error rate on SpaceWire links is so low that errors don't occur. Resets and Resynchronization almost always occur as the result of an overt act (such as disconnecting a cable) and the monitor is usually manually restarted anyway.

Software Implementation

A wrapper was created in LabView to expose the Spacewire-PCI API provided by the Windows NT driver supplied with the SpaceWire-PCI interface. Direct access to SMCS registers is provided by the driver and is used for most of the .vi operations. The SMCS 332 initialization is performed in the usual way with the Windows NT drivers, with the exception is that the CHx_COMICFG register is set for one packet/message. In addition, the monitor nodes are configured as follows

- Transparent mode
- Explicitly matched for bit-rate, endian mode, etc. to the link under observation
- Never used for transmit

The LabView virtual instrument (.vi) controls all aspects of the channel for the monitor links, including servicing the interrupts, emptying the buffers, and so forth. On a high rate link with a lot of messages, this actually proves to be the limiting resource. The performance limitations of the Labview/SpaceWire-PCI monitor approach has not been quantified through testing, however, it easily keeps up with the usual breadboard traffic, consisting of messages every few milliseconds consisting of about a thousand bytes. Unfortunately, the Labview implementation does not allow accurate unambiguous time-stamping (with an accuracy of milliseconds), mostly because of the limitations inherent in the NT driver model.

MONITORING SPACEWIRE WITH AN FPGA IMPLEMENTATION OF SPACEWIRE

JPL have achieved a remarkable degree of monitoring SpaceWire links by using the SpaceWire-PCI boards. There is, however, information inside the SMCS332 on the board that would be useful for the monitoring purpose, and an FPGA implementation would provide easier access to this information.

4Links have developed a series of FPGA implementations of SpaceWire, and use their latest implementation for monitoring SpaceWire links. Useful information that is made available for monitoring includes:

- The initialization state machine
- All the error signals
- All the different types of character
- The precise timing of when characters arrive
- The actual received Data and Strobe signals

This information is recorded and reported as the user requests, in the form of time-stamps, activity logs, and waveform traces.

Time stamps

With access to the data when it arrives, it is possible to give very precise time-of arrival information. A resolution of 100ns is to within a bit at 10Mbits/s and to within a data character at 100Mbits/s. The 100ns resolution is also easily generated by the clock that is used for 10Mbits/s data, and it is also a standard period provided by external signal generators with 10MHz outputs.

Time stamps can be useful at the start of packet and at the end of packet, and sometimes both. So these are provided. In special cases, it may also be useful to stamp the arrival time of each character, and so this capability is provided — with the proviso that it generates rather more data than the packet itself.

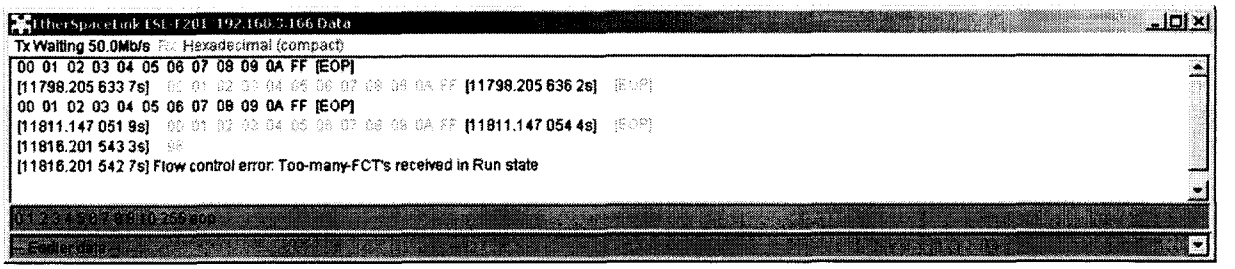


Fig. 4: Data log showing time-stamps, such as `[11798.205 633 7s]`*, for received data, together with an error report showing the time stamp, the type of error, and the SpaceWire state machine state when the error occurred.*

Fig. 4 shows the time stamp reports , such as `[11798.205 633 7s]`, as shown by the 4Links EtherSpaceLink ESL-F201 with the TimeStamp option. The stamp is a free-running 32-bit count with the resolution of 100ns. More precise resolution could be offered if required but, as explained, the 100ns provides useful resolution.

Error reports

SpaceWire defines a wide variety of error conditions, which can be reported by a monitor function. Fig. 4 shows an error being reported (a flow-control credit error as a result of unplugging a cable). The error is time-stamped, and the SpaceWire state machine state when the error occurred is also reported.

Time Codes

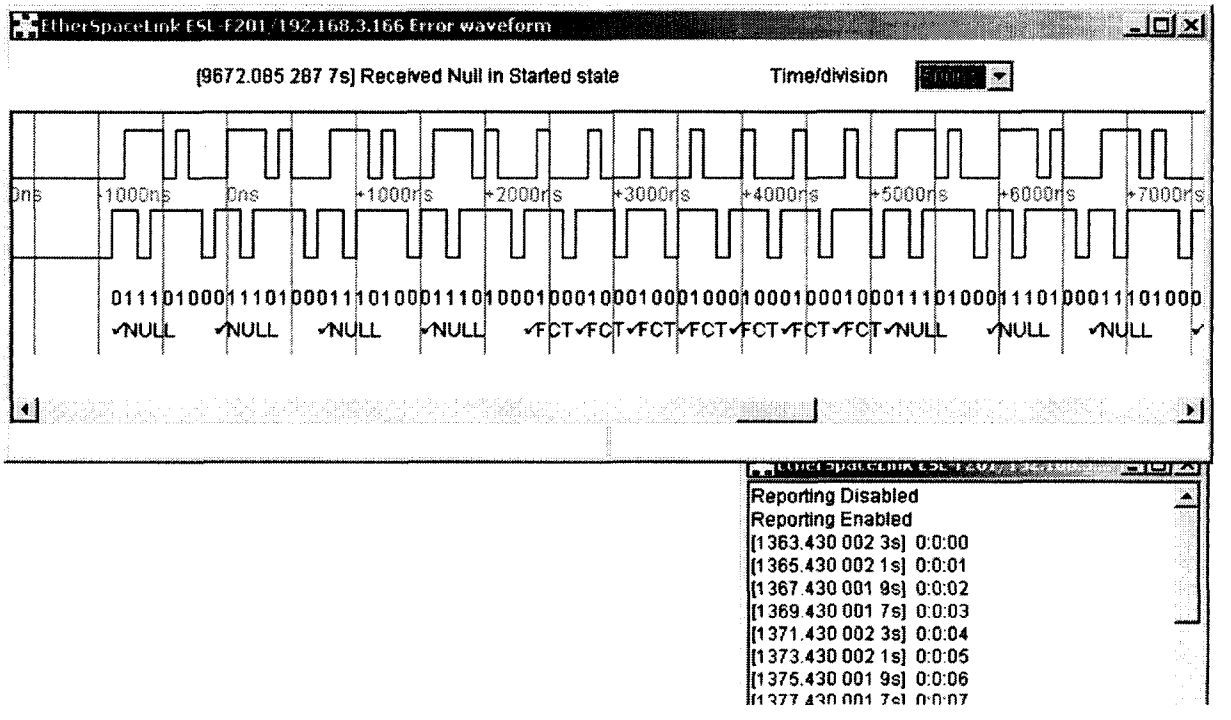
Individual time-codes could be reported on the normal Data log window, but they could be rather frequent and they are orthogonal to most of the other activity on the link. So it is useful to provide a separate window for displaying reports of received time codes. This window is shown in Fig. 5. The report includes a time stamp of when the time code arrived, and the time code in the format specified in the SpaceWire standard,
with two reserved bits and then a six-bit value. The six-bit values
should increment as is shown in Fig. 5.

Triggering and Waveform Capture

As JPL comment, standard equipment such as oscilloscopes is useful for initial work with SpaceWire links but there soon becomes a need to examine more detail, and particularly to trigger on events such as initialization and errors. Building waveform capture into the monitoring system allows the data to be presented already decoded, and co-ordinated with a report of the triggering event. Fig. 6 shows a waveform captured for SpaceWire initialization.

Fig. 6: Waveform capture of link start-up

Fig. 5: Time Codes reported with time-stamps



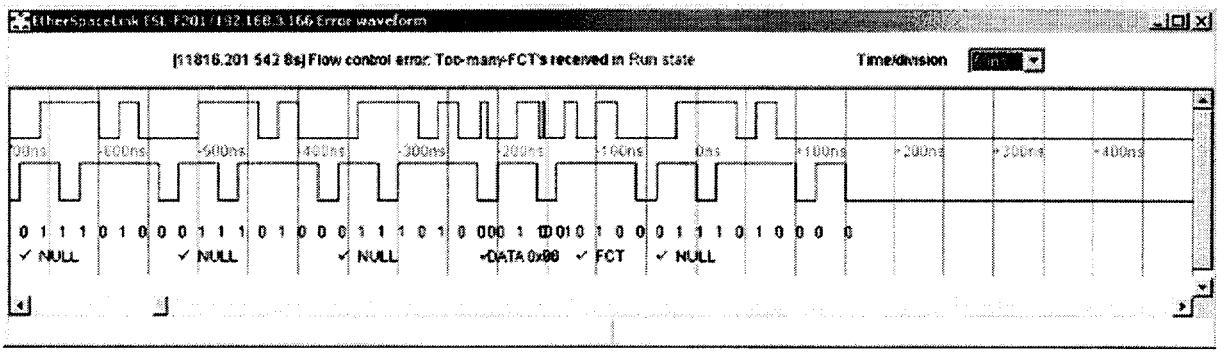


Fig. 7: Waveform capture of the error reported in Fig. 4

Waveform capture is also useful for diagnosing errors, as shown in Fig. 7, which is the same event as was reported in Fig. 4. Note the time stamp of when the error occurred, the type of error, the SpaceWire State Machine state when the error occurred, the decode of individual bits, and the further decode of those bits into characters. At the time this error occurred, there was a glitch on the Data signal which caused first of all a data character in error, and then an extra Flow-control token (FCT) which exceeded the permissible credit count.

The reports shown in Figs. 4 to 7 were generated by the EtherSpaceLink ESL-F201, with options for time codes, time stamps, error reporting and error waveforms, and were displayed by the SpaceWireUI program [3].

The EtherSpaceMon ESM F200 will include the two link receive circuits shown in JPL's Fig.3 and will bring these capabilities to monitoring links within systems. Fig.8 shows a block diagram of how the EtherSpaceMon would be used in a system.

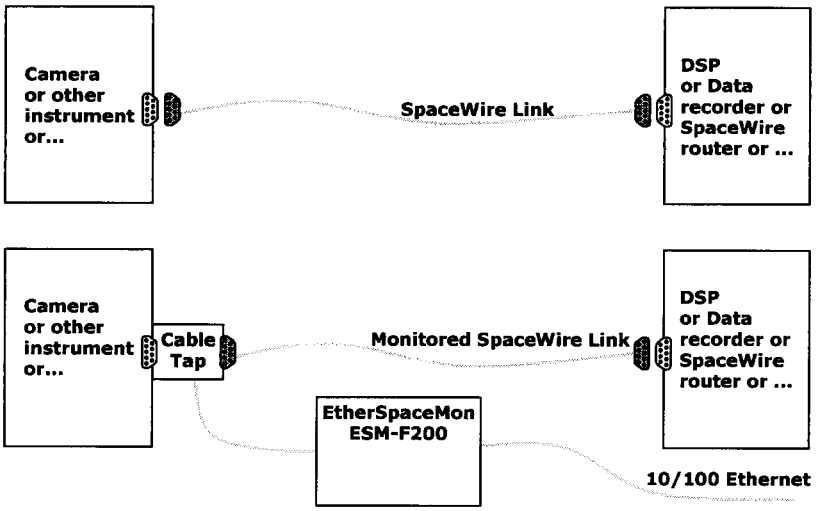


Fig. 8: Use of EtherSpaceMon in SpaceWire system

Acknowledgement:

This research was partially carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References:

1. Clark, D.J.; Lux,J.P; Shirbacheh,M.; "Testbed for development of a DSP-based signal processing subsystem for an earth-orbiting radar scatterometer", *IEEE Aerospace Conference Proceedings*, 2002, Vol 4, pp 4-1881 - 4-1890

2. Spacewire standard ECSS-E-50-12A (24January2003)

3. EtherSpaceLink and SpaceWireUI User Manuals, 4Links, 2003

APPENDIX: DESCRIPTION OF STATE MACHINE

The diagram begins with the resetting of an ESC character detect flag. This flag is set whenever an ESC character is detected (and the flag has not yet been set). A flag is required to coordinate the proper reporting of NULL and FCT characters, as an FCT preceded by an ESC is the SpaceWire definition of a NULL .The next state, called P, takes the first bit of the stream and stores it in the Boolean parity variable P. Following P is a state called "Data or Collect", that takes the next bit and uses it to decide which way to branch. If it is a '1', then the character is a control character, otherwise it is a data character. The '0' path leads to a state called "B8,t". This state will shift the next eight bits into the octet variable B8, after which it will update the variable t with the current time. The following path, called [Report B8,t], "emits" the content of B8 and t to a *monitor events list* (essentially a cache). Every path with a statement in square brackets contributes likewise to this list when that path is traversed. After [Report B8,t], a "Test Parity" state is entered. This compares the Odd parity of B8 with the contents of P. The results will determine whether the E (Error) path or NE (No Error) path is taken. All E paths return to beginning state, as a parity error will result in link re-initialization. Note that the error is reported just before the links reset. The next state will test if the previous character was an ESC. If so, then the ESC flag is cleared and a [Report ESC,te] is emitted. The 'te' variable is the time when the ESC character was detected, and is updated along the ESC case part of the diagram. Since the SpaceWire specification allows for transmission of 'free' ESC characters (even during a data packet transfer), a provision must be made to accurately report their occurrence. The scheme employed here is one of "deferred reporting", i.e. reporting of the ESC is deferred by one character until it can be determined whether it is 'free' or part of a NULL character.

If the character is a control character, The state "b2,t" is entered. This will shift the next two bits into the doublet variable b2, after which it will update the variable t with the current time. State b2,t has four possible output paths that depend entirely upon the value of b2. Each of these paths first encounters a parity check state that will return the machine to its' beginning state in the event of an error. (just like in the Data case). Following is a description of each case. Note that all of the control cases return to the P state if no error is detected.

Case 0: FCT detected
A Test ESC Flag state is entered. If it is true (an ESC character was received previously), then the ESC flag will be cleared, and a NULL character will be reported. Otherwise an FCT character will be reported.

Case 1: EOP detected
Since an EOP does not combine with any other character so as to change it's meaning, it is reported immediately. If the ESC flag was set, it is cleared, and the ESC is then reported. (Note that the deferred ESC report causes no temporal confusion, as the time-of-event variable 'te' is reported with the ESC instead of 't'.

Case 2: EOM detected
The case with EOM is entirely analogous to the EOP case.

Case 3: ESC detected
If the prior character was also an ESC, then that previous ESC will be reported (The ESC flag state is not cleared, as an ESC has just shown up). The 'te' variable will then be set to the value if 't', (The new arrival time for an ESC). If the ESC flag is not set, then it will be set, along with the value of 'te'. This sequencing insures that the te variable will contain the time when the reported ESC occurred.