plevin

SIGA SPEC

spec.qms

*** Request Q00000.14191 for plevin@cheese.bbn.com Tue Jul 26 15:48:57 1988

Account.	2128022
Submitted.	Tue Jul 26 15:49:58 1988
Printed	Tue Jul 26 15:48:57 1988
Font	74
Orientation:	Portrait
Line-spacing:	600
Character-spacing:	1100
Page-break:	After 60 lines
Top-margin:	550
Left-margin:	750

Table of Contents

1 Introduction

The SIGA is a gate array device which serves as the bidirectional interface between a Computational Node and the Switch network of the Butterfly II Parallel Processor. As such, the SIGA provides devices on each Computational Node with virtually transparent read and write data access to similar devices on physically remote nodes. The SIGA accomplishes this by accepting/presenting data via the standard interface that these devices support - namely the T-Bus - and then presenting/accepting this same data to the Butterfly Switch interface for transport.

This document will present both a detailed functional and operational description of the SIGA. It is intended to be used as a design guide for both hardware and software system engineers. This specification is necessarily limited in its scope and thus will touch upon other Butterfly II-related subjects only when it is necessary for completeness. Therefore, it is assumed that the reader of this document has a general knowledge of the concepts of the Butterfly II architecture and operation. The reference documents are as follows:

T-Bus Specification (Ward Harriman)

Switch Gate Array Design Specification (Ward Harriman)

Butterfly II Level Converter Array Specification (Mike Sollins)

Switch Protocol Specification (Ward Harriman)

Reference Documents
Figure 1

2 Terminology

The following terms will be used throughout this document:

Byte - Refers to an 8-bit quantity.

Anticipation - A feature of the SIGA design that allows the SIGA to take advantage of certain parallel optimizations.

Downstream Node - The node which services a switch transaction.

Drop-Lock - When the Requestor negates Frame during a locked sequence, causing the Server to issue a FREE-LOCK.

Function Response - A generic term for the various incarnations of a response to a function request from some downstream T-Bus slave to an upstream T-Bus slave. This includes the transformations that the response undergoes as it travels from the downstream T-Bus, downstream SIGA, Switch, upstream SIGA, and finally the upstream T-Bus. (see Function Request)

Function Request - A generic term for the various incarnations of a request from some upstream T-Bus master to a downstream T-Bus slave. This includes the transformations that the request undergoes as it travels through the upstream T-Bus, upstream SIGA, Switch, downstream SIGA, and finally the downstream T-Bus. (see Function Response)

Final Locked message - The same as a Locked message except that the Switch path is released by letting Frame=0 for at least two Switch Intervals after the operation has been acknowledged.

Half-Word - Refers to a 16-bit quantity. (see Word)

Initial Locked message - Occurs under the same circumstances as the Unlocked message except that the Switch path is held open once the operation has been acknowledged without errors.

Local Errors - Errors which originate in the Requestor.

Logical Route Address - A 9-bit Switch node address generated from either the Interleaver or the T-Bus. This address is then transformed, possibly by randomizing some of the bits, into the Physical Route Address.

Locked message - A message which occurs when the Switch path was already locked and causes it continue to be locked after the operation has been acknowledged.

Message - With the exception of Reject, a Message is the assertion of Frame (downstream message) or Reverse (upstream message) possibly with associated data on the data lines.

Message Acknowledgment - Also known as M_ACK. This refers to the assertion of Reverse for at least two Switch Intervals during a function response. It indicates that the downstream Server has Acknowledged the receipt of a Function Request.

Message Header - The part of a downstream Switch message that carries routing information. That part is stripped-off by the Switch and thus never reaches the downstream Server The message header for an upstream Switch message is null.

Message Body - The part of the downstream Switch message that carries the command, address, data, and checksum bytes.

Multi-Word Transfer - Refers to a read or write function request that involves more than one word (32 bit) of data

Operational State - A SIGA initialization state which allows full operation of the SIGA.

Pad - A special class of downstream message which contains all zero's for data. It is used by the Requestor to hold the Switch path open while it awaits a message acknowledgement.

Physical Route Address - The transformation of the Logical Route Address after some of its bits have been randomized. The Physical Route Address is placed into the downstream Message Header.

Quick-Drop - This is an optimization in the Requestor where the R_FRAME signal is negated as soon as possible after an R_REVERSE is received.

Quiescent State - A SIGA initialization state which allows partial operation of the SIGA.

Remote Errors - Errors which originate in the Server.

Reject - An assertion of Reverse for one Switch Interval. Indicates that a message was rejected at either a Server or an SGA.

Sequence - A function request followed by a function response.

Split-Cycle - A T-Bus Read transaction where the Master releases the bus while the Slave is completing the transaction.

Switch Interval - The 25 ns period in which Switch data is propogated.

Switch Modulus - The number of ports that a basic switching element can handle. That number is currently eight.

Transaction - Another word for a Sequence.

Unlocked Message - Occurs when the Switch path had previously been "torn-down". This occurs whenever Frame was "0" for at least two Switch Intervals. Once the operation has been acknowledged, the path is torn-down again.

Upstream Node - The node which initiates a switch transaction.

Valid Message - A downstream message which carries a read or write request and does not violate switch protocol.

Word - Refers to a 32-bit quantity. (see Half-Word)

3 Document Standards

The following describes some of the standard syntax and expressions used in this document.

3.1 Register Definition Syntax

A typical register definition is shown in Figure 2. Referring to Figure 2, the "-" in any bit indicates that this bit is a "don't care" on a write and indeterminate on a read. If "-" totally fills a field of eight bits, that field should NEVER be written to but of course, can be read from. The entire register may be referred to in any one of the following ways: The sub-fields, shown in Figure 2 within "[]", can be referred to in various ways. For instance, the "Cnt" subfield could be referred to as:

```
Register: Protocol_Timer_Config<15..0>

15                    0
|                     |
3..0   3..0   7......0
CCCC   PPPP   --NNNNNN
[Cnt]  [Pre]  [Con]
```

Register Syntax Definition
Figure 2

(1) Protocol_Timer_Config<15..0>
(2) Protocol_Timer_Config
(3) PTC<15..0>, or
(4) PTC

(1) Protocol_Timer_Config<15..12>
(2) Protocol_Timer_Config.Cnt<3..0>
(3) Protocol_Timer_Config.Cnt
(4) PTC.Con

3.2 Logical Operators

Figure 3 shows the standard operators used in this document.

3.3 Timing Diagram Symbols

Timing diagrams use ASCII characters to reprsent signal states. Figure 4 illustrates some of those symbols and their associated meanings. In addition, if no clock signal is present in a timing

OPERATOR	FUNCTION
&	logical "and"
#	logical "or"
$	logical "exor"
!	logical "not"
!$	logical "exnor"
|	concatenate

Example - Logical Operators
Figure 3

SYMBOL	MEANING
H	logical "1"
_	logical "0"
.....	continue previous state
?????	state unknown and unimportant

Example - Logical Operators
Figure 4

diagram, it is assumed that each character column represents an active transition of the appropriate clock.

4 Functional Overview

The following describes the basic functionality of the SIGA at a conceptual level.

4.1 Functional Unit Description

The SIGA is composed of four basic elements, the Requestor, Server, Control Net Interface and the Config/Status Unit. Although these are physically co-located and share some common logic and control, they are functionally independent units and will be described separately.

4.1.1 Requestor

The Requestor is a T-Bus slave device which transparently couples physically remote T-Bus slave devices to the local T-Bus by interacting with both the Switch and the downstream Server. The Requestor appears to the current T-Bus master as a segment of memory which is out of the range of physical memory at the local node. Signals on the T-Bus alert the Requestor that the current access is for a remote location and the Requestor then initiates the switch transaction to comply with the master's read or write request.

Since the transaction is not completed immediately (indicated by the Requestor with a PROMISE response) , the requesting T-Bus master follows the T-Bus protocol and releases the bus so that other devices may use it. The Requestor eventually regains control of the T-Bus, alerts the requesting master that the read or write operation has been completed, and returns data or an error indication. If the current sequence is locked, as requested by the T-Bus master, and no errors are encountered, the Requestor holds open the Switch path for the next transaction rather than rearbitrating for a new Switch path. Any errors that may have occured during this operation are signalled to the T-Bus Master through the ERROR response.

4.1.2 Server

The Server acts as a master on the local T-Bus of the downstream node and services requests from the upstream node's Requestor. When a new valid message enters the Server from the Switch, the Server obtains the local T-Bus; locks the T-Bus slave, if desired; performs the read or write operation; and then returns the data and/or error byte to the Upstream Node's Requestor. The Server can also initiate other special operations independently of receiving a new Switch message. This operation, known as drop-locks, is described elsewhere in this document.

4.1.3 TCS Control Unit

The basic purpose of the TCS Control Unit (TCU) is to give the serial interface of the TCS Control Slave Processor access to the T-Bus interface - in essence, to act as a protocol converter. A secondary function is to allow the TCS Slave Processor DIRECT access to some of the internal functions of the SIGA, rather than forcing it to access via the T-Bus interface. This is useful for fault-tolerance and "out-of-band" functions such as bootstrapping.

4.1.4 Configuration/Status Unit

The Config/Status Unit (CSU), acting as a T-Bus slave, allows read/write access to all programmable parameters of the Requestor, Server and TCS Control Unit. The CSU also provides convenient access to the internal state of certain nodes for testability.

4.2 System Operation

From a high-level view, the SIGA is one link in the chain of devices that allows a T-Bus device to fulfill a function request with a function response. The role of the SIGA in fulfilling both function requests and responses is now described.

4.2.1 Function Requests

A local T-Bus master in the upstream node, usually the CPU, initiates the sequence by placing an address on the T-Bus, which is detected by the SIGA Requestor as a remote access request. During the T-Bus request phase, the SIGA stores the address, produces and stores the bid, and command bytes. It then initiates the downstream message at the Switch interface by asserting Frame and placing the bid symbols on the Switch data lines. Normally, this message tramsmission is initiated by the SIGA immediately upon receiving the address from the T-Bus, but it can be programmed to start later. On a write, the SIGA loads its data registers during the response phase of the T-Bus cycle. All operations are split-cycle and thus the Server will release the bus while it processes the transaction request.

If there is no Switch contention, the assembled message continues to be transmitted and is ultimately appended with a checksum derived from the message data bytes. If there is Switch contention, a Reject is generated by the Switch and eventually makes its way upstream to the Requestor via the Reverse line When this happens, the Requestor negates Frame, waits for a predetermined amount of time and then retries the message by asserting Frame and sending the message components stored from the first attempt.

Sometime after the beginning of the message reaches the Server at the downstream node (i.e. it is not Rejected by the Switch), that Server begins arbitration for its local bus to complete the transaction. If the device on the downstream node is locked to a remote bus master other than the Server, the Server issues a Reject which propogates upstream and is eventually detected at the upstream Requestor. This Reject is treated exactly the same by the Requestor as a Reject from the Switch. Note that this is the ONLY instance in which the Server will issue a Switch Reject - an Initial Message.

Assuming that the Requestor receives neither a Switch Reject nor a Server Reject, it deasserts Frame for one switch interval while it sends the checksum byte. Within the checksum byte, the "forward" bit is reset. This event would normally cause the forward drivers of the SGA's to turn off after they send the checksum byte. However, the current implementation of the SGA ignores this bit and turns-on its foward drivers in response to the Frame profile. The Requestor then sends the Pad message (all 0's) and awaits a response from the Server. Note that the forward bit is not used by the current SGA's.

In the meanwhile, the downstream Server begins processing the request by arbitrating for the local T-Bus. Assuming that the target downstream bus slave was not locked to a downstream master other than the Server, the Server obtains the local bus and possibly opens the local memory lock. The Server will open the lock only if this action was requested in the downstream message. This would occur if the master on the upstream node's local bus requested an OPEN lock when it initiated a transaction through it's associated Requestor.

Once the downstream Server obtains the local bus, it makes a function request to perform the appropriate read or write operation. The only exception to this is when the Server detects a checksum error in the downstream message. If this occurs, instead of making a request, the Server releases control of the T-Bus, creating a "dead" bus cycle and thereby aborting the transaction. This action on an aborted transaction should eliminate any unwanted side-effects if the switch message is corrupted.

4.2.2 Function Responses

Assuming that a read transaction was requested, the downstream Server completes the read as a normal local T-Bus master. As soon as the read data is obtained by the Server, a message is returned to the upstream Requestor. This happens (over the same data wires which the downstream message was sent) by asserting Reverse and applying data to the Switch data lines. The upstream message contains the read data, and possible error data; a checksum; and a message acknowledgement, or M_ACK which is implicit in the assertion of Reverse for at least two Switch intervals. If a write transaction was requested, the Server writes the data to the address specified in the downstream message and sends back an M_ACK with an error byte data and checksum after the data has been accepted by the slave on the local T-Bus. In short, a read returns data/errors and an acknowledgement whereas a write only returns possible errors and an acknowledgement.

In the case of a read transaction, the upstream Requestor detects the M_ACK and alerts the local split-cycle master which initiated the request that the requested data has been returned. That master then completes the operation by retrieving the data. In the case of a write transaction, the Requestor also alerts the initiating local bus master that the write was completed but returns only error information.

In the absence of errors, the Requestor will continue to hold the Switch path open by asserting Frame only if the sequence was initiated with an OPEN. If that master decides to release the lock, the Requestor will tear-down the switch connection by negating Frame and will enter its unlocked idle state. This is the state that it was in at the beginning of this discussion of function requests. If the upstream bus master does not release the lock, it may initiate another read or write transaction. This and subsequent transactions are referred to as locked transactions. Outside of errors, locked transactions end only when the upstream T-Bus master which OPENed, MAINTAINed or BYPASSed the SIGA Requestor lock decides to release that lock with a FREE-LOCKS command.

Subsequent message transactions in a locked sequence differ from the initital transactions described above in three major ways. First, locked messages do not contain any bids because the path has already been established. Second, the Switch will never issue a Reject because the path has already been established and is being reserved for the Requestor. And third, the downstream Server will never issue a Reject because it will already have exclusive use of the local memory lock. Aside from these exceptions, subsequent locked transactions occur in exactly the same manner as unlocked transactions. As mentioned previously, the upstream T-Bus master owning the SIGA Requestor MUST release that lock explicitly with a FREE-LOCKS.

4.3 Basic Message Formats

Message formats differ mainly with the type of function request; read or write. Within a read or write message, the downstream and upstream messages corresponding to a function request and response also differ.

4.3.1 Read Messages

Read message formats differ mainly depending on whether or not they are downstream or upstream messages.

4.3.1.1 Downstream

Downstream Read messages are differentiated partly because of their data format and partly because of the state of Frame at the beginning and end of the message. The formats for three possible SIGA Requestor read operations are considered:

1) An Unlocked Read occurs when the Switch path had previously been "torn-down". This occurs whenever Frame was "0" for at least two Switch Intervals. Once the operation has been acknowledged, the path is torn-down again.

2) An Initial Locked Read occurs under the same circumstances as the Unlocked Read except that the Switch path is held open once the operation has been acknowledged.

3) A Locked Read is a read which occurs when the Switch path was already locked and it continues to be locked after the operation has been acknowledged.

In all cases, the Requestor waits for a Message Acknowledgement (M_ACK) from the downstream Server before completing the message. Figure 5 illustrates the three read message types for a two column switch. In this figure, the "d" field indicates the direction of the LCON drivers which interface data with the LCON. When d = "P" (Output), the Requestor is sourcing data to the Requestor/LCON interface. When d = "I" (Input), the LCON drivers are sourcing data to the Requestor/LCON interface. The "f" field is the state of the Frame bit. Data is MSB at the left of the field.

4.3.1.2 Upstream

When a downstream read message has been received and processed by a Server, an upstream message is returned to the initiating Requestor based on the operation requested. Under normal conditions, the Upstream Message is composed of two parts: the returned data (with checksum) and the M_ACK (Message Acknowledge). The returned data is the contents of the remote memory location read, which can be 1 ,2 or 4 words in length. With the exception of rare error conditions, the actual message data field is almost always a multiple of four.

```
Unlocked            Initial             Locked
Read                Locked Read         Read
============        ============        ============

d f   data          d f   data          d f   data
- - --------        - - --------        - - --------
P 0 xxxxxxxx        P 0 xxxxxxxx        P 1 xxxxxxxx
P 0 xxxxxxxx        P 0 xxxxxxxx        P 0 xxxxxxxx
P 1 -bid1---        P 1 -bid1---        P 1 -cmd----
P 1 -bid2---        P 1 -bid2---        P 1 -addr1--
P 1 -cmd----        P 1 -cmd----        P 1 -addr2--
P 1 -addr1--        P 1 -addr1--        P 1 -addr3--
P 1 -addr2--        P 1 -addr2--        P 0 -check--
P 1 -addr3--        P 1 -addr3--        I 1 00000000
P 0 -check--        P 0 -check--        I 1 00000000
I 1 xxxxxxxx        I 1 xxxxxxxx               "
I 1 xxxxxxxx        I 1 xxxxxxxx            M_ACK
       "                   "            and read data
                                               "
    M_ACK               M_ACK
and read data       and read data       I 1 xxxxxxxx
       "                   "            P 1 xxxxxxxx
I 0 xxxxxxxx        I 1 xxxxxxxx
P 0 xxxxxxxx        P 1 xxxxxxxx
```

Read Switch Message Format - Downstream
Figure 5

Figure 6 illustrates the upstream message. The "r" field is the Reverse signal. Data is MSB at left of the field.

4.3.2 Write Messages

Write message formats differ mainly depending on whether or not they are downstream or upstream messages.

```
1-word,
4-byte
Read
==========

r   data
- --------
0 xxxxxxxx
1 -data a-
1 -data b-
1 -data c-
1 -data d-
1 -check--
0 xxxxxxxx
```

Read Switch Message Format - Upstream
Figure 6

4.3.2.1 Downstream

Downstream Write messages are differentiated partly because of their data format and partly because of the state of Frame at the beginning and end of the message. The formats for three possible SIGA Requestor write operations are considered: In all cases, the Requestor waits for a Message Acknowledgement (M_ACK) from the downstream Server before completing the message. Figure 7 illustrates the three write message types for a two column switch. In the figure, The "d" field is the direction of the LCON drivers which interface data with the SGA. When d = I, the Requestor is sourcing data to the Requestor/LCON interface. When d = P, the LCON drivers are sourcing data to the Requestor/LCON interface. The "f" field is the state of the Frame bit. Data is MSB at left of the field.

1) An Unlocked Write occurs when the Switch path had previously been "torn-down" by the fact that Frame was "0" for at least two Switch Intervals. Once the operation has been acknowledged, the path is torn-down again.

2) An Initial Locked Write occurs under the same circumstances as the Unlocked Write except that the Switch path is held open once the operation has been acknowledged.

3) A Locked Write is a write which occurs when the Switch path was already locked and it continues to be locked after the operation has been acknowledged.

4.3.2.2 Upstream

When a downstream write message has been received and processed by a Server, an upstream message is returned to the initiating Requestor based on the operation requested. Under some conditions, the Server will not act on the downstream message and will instead send a Reject back to the Requestor. Under normal conditions however, upstream messages contain an M_ACK, an error byte (normally all 0's) and a checksum.

The following illustrates the only possible return message for a write. The "r" field is the Reverse signal. Data is MSB at left of field.

5 Detailed Functional Description

The Requestor, Server, TCU and Configuration/Status Unit are now described in detail.

```
Unlocked            Initial             Locked
Write               Locked.Write        Write
===========         ===========         ===========

d f   data          d f   data          d f   data
- -  --------       - -  --------       - -  --------
I 0 xxxxxxxx        I 0 xxxxxxxx        P 1 xxxxxxxx
I 0 xxxxxxxx        I 0 xxxxxxxx        P 0 xxxxxxxx
P 1 -bid1---        P 1 -bid1---        P 1 -cmd----
P 1 -bid2---        P 1 -bid2---        P 1 -addr1--
P 1 -cmd----        P 1 -cmd----        P 1 -addr2--
P 1 -addr1--        P 1 -addr1--        P 1 -addr3--
P 1 -addr2--        P 1 -addr2--        P 1 -data a-
P 1 -addr3--        P 1 -addr3--        P 1 -data b-
P 1 -data a--       P 1 -data a-        P 1 -data c-
P 1 -data b-        P 1 -data b-        P 1 -data d-
P 1 -data c-        P 1 -data c-        P 0 -check--
P 1 -data d-        P 1 -data d-        I 1 xxxxxxxx
P 0 -check--        P 0 -check--        I 1 xxxxxxxx
I 1 xxxxxxxx        I 1 xxxxxxxx              "
I 1 xxxxxxxx        I 1 xxxxxxxx            M_ACK
      "                   "                   "
    M_ACK               M_ACK           I 1 xxxxxxxx
      "                   "             P 1 xxxxxxxx
I 0 xxxxxxxx        I 1 xxxxxxxx
P 0 xxxxxxxx        P 1 xxxxxxxx
```

Write Switch Message Format - Downstream
Figure 7

5.1 Requestor

The Requestor is described from the point of view of its overall operation and its two major interfaces: the T-Bus interface and the Switch Interface.

Any Write

```
=========

r   data
-  --------
0 xxxxxxxx
1 -error--
1 -check--
0 xxxxxxxx
0 xxxxxxxx
```

Write Switch Message Format - Upstream
Figure 8

5.1.1 Operation

The operation of the Requestor is described by discussing its major functions.

5.1.1.1 Overview

The Requestor is a local T-Bus slave which creates a logical coupling to a physically remote T-Bus slave via the Switch. The Requestor acts as the "initiator" of this coupling on the Switch and thus can be thought of as a "slave" on the T-Bus but a "master" to the Switch. Referring to Figure 9, the Requestor contains three major functional units: Bus Interface Unit (BIU), Switch Tx Unit (STU), and the Switch Rx Unit (SRU). The BIU is clocked by the T-Bus clock and both the STU and SRU are clocked by the Requestor Switch clock (R_CLK). Interfacing of control signals between these units is accomplished with handshake synchronizers, as shown. The BIU handles all of the T-Bus transactions of the Requestor. The STU translates function requests that it receives from the BIU into Switch transactions. The SRU receives reply messages from the Switch and passes their status, in the form of a status code, back to the STU and their data back to the BIU. The STU serves as the

```
|~|
| |          +----------+                    +------+
| |          |          |       +----+       |      |          |
| |          |          |<-----|sync|------->|Switch|          |
|T|          |          |       +----+       |Tx    |--------->|
|B|          | Bus      |===================>|Unit  |=========>|
|U|          | Interface|                    |      |          |
|S|<=======>| Unit     |               +-->|      |          |S
| |          |          |               |    +------+          |W
| |          |          |               |                      |I
| |          |          |               |    +------+          |T
| |          |          |               +-->|      |          |C
| |          |          |                    |      |          |H
| |          |          |                    |Switch|          |
| |          |          |<===================|Rx    |<---------|
| |          |          |                    |Unit  |<=========|
| |          |          |                    |      |          |
| |          +----------+                    +------+          |
|~|
```

Requestor Block Digram
Figure 9

single interface for control information between the T-Bus side and Switch side of the Requestor and therefore control information in either direction must pass through the STU. This is done to reduce the number of control interfaces that the BIU must deal with.

The BIU/STU interface is a streamlined request/response type interface where for each BIU request there is an STU response. The BIU presents an encoded function request to the STU and sets an "execute" flag. When the STU is done operating on that request, it sets a "done" flag and returns a status code and data to the BIU. Both the BIU and STU are responsible for handling their own functions independently and they have very little real-time knowledge of each other's state. This approach simplifies the Requestor design and carries the request/response philosophy throughout the system.

The BIU has four major responsibilities: (1) screen T-Bus requests for correctness; (2) transfer screened T-Bus requests to the STU if a Switch transaction is indicated by that T-Bus request; (3) receive replies from the STU; and (4) pass replies, including any errors, as responses to the T-Bus. The BIU acts as a T-Bus slave which is always in split-cycle mode. In other words, it NEVER responds immediately to a function request from a T-Bus master except when a request error is detected. Outside of those exceptions, the BIU always responds with a PROMISE to T-Bus requests.

The BIU screens T-Bus requests for both T-Bus protocol violations and illegal function requests. Without exception, these conditions will prevent the BIU from ever activating the STU to complete an initial function request. The BIU can also initiate certain function requests to the STU independently of T-Bus requests. An example of this is the drop-lock function which may under certain conditions be initiated by the BIU rather than the T-Bus.

The STU acts on a function request from the BIU and initiates the Switch transaction to carry out that request. The STU also is responsible for assembling and transmitting the data in an outgoing message. It also handles things such as the message start/retry and priority promotion algorithms and deals with various protocol timeout violations.

The SRU is fairly simple in function. It detects the return message of a function request inititated by the STU, verifies the checksum and alerts the STU of the incoming message and the checksum status. The SRU also detects Switch Rejects.

5.1.1.2 RTC and related functions

The Real Time Clock, besides being useful as a system timekeeper, is central to the operation of much of the Requestor. It is used to directly control the functions of the Time_Of_Next_Interrupt and the Priority_Time_Slot mechanisms. These mechanisms are described in this section. The RTC is also used, in a less direct manner, to control the Protocol Timers. Protocol timers are discussed elswhere in this document.

5.1.1.2.1 Real Time Clock and Prescaler

The RTC is basically a large (32 bits) counter which is updated every one microsecond from a divided-down version of the Switch clock. Since the frequency of the Switch may vary in different applications, the Real Time Clock uses a programmable prescaler to divide the Switch frequency down to a one microsecond time base. A functional diagram of the Real Time Clock is shown in Figure 10.

```
|----------Real Time Clock------------|       |-Prescaler-|

|-----RTC.Hi-----|   |-----RTC.Lo-----|       |----RTP----|

+----------------+   +----------------+       +-+   +-----+
|       16       |   |       16       |<---o|1|<-h|  5  |
+-------------i--+   +-------------c--+       +c+   +---c-+
              ^                    ^           ^        ^
              |                    |           |        |
              |                    |           |        |
              +--------------------+-----------+--------+
                                   |
                                   |
                                +-----+
                                |delay|<--M_SIXTY_FIVE
                                +-----+
```

...where,

h = increment pulse (period = 0.5 us)
o = increment pulse (period = 1.0 us)
s = increment pulse (period = 65536 us)
c = clear input
i = increment input

Functional Diagram - Real Time Clock
Figure 10

Figure 10 shows that prescaler is actually composed of two parts. The first part is a count-up prescale counter that has a programmable terminal count value. This 5-bit terminal value is supplied by the Real_Time_Prescale subfield of the ConfigA register (REQ_ConfigA.Real_Time_Prescale). The 5-bit counter drives the second part of the prescaler: a divide-by-two. The divide-by-two then generates the one-microsecond time base used by the Real Time Clock. Figure 10 also shows the presence of the M_SIXTY_FIVE signal. This signal is a system-wide pulse which occurs every 65 milliseconds and lasts for one Switch Interval It is used to keep all the Real Time Clocks on all nodes in synchronization.

The M_SIXTY_FIVE resets the entire prescaler and the the lower-half of the Real Time Clock. In addition, it increments the upper-half of the Real Time Clock. Figure 10 also shows a "pipeline" delay for the M_SIXTY_FIVE signal. The Configuration bits, REQ_ConfigA.Sixty_Five_Delay<1..0>, allow the adjustment of this delay The adjustment values and their effects are shown in Figure 11.

WARNING: The setting DD = 00 is for test purposes only and must NOT be used in normal operation.

DD	Delay
00	none
01	1 Switch interval
10	2 Switch intervals
11	3 Switch intervals

...where.

D..D = ConfigB.Sixty_Five_Delay<1..0>

Sixty_Five_Delay Settings
Figure 11

In actual operation, the prescaler RTP<4..0> counts-up at the Switch frequency until it reaches the count stored in REQ_ConfigA.Real_Time_Prescale, where it generates an increment pulse lasting one Switch Interval. In the next Switch clock interval, the prescaler rolls-over to zero. Thus, the ConfigB.Real_Time_Prescale must always be programmed to make RTP<5> have a period of 0.5 microseconds.

WARNING: Because of hardware speed considerations, the OMSP generated by the RTP is actually pipelined by one Switch Interval. Therefore, the RTP appears to be running "ahead" of the RTC by one Switch interval. This fact only becomes signifcant for the Slotted Start/Retry criteron. See that section for further details.

The Real Time Clock is basically, as mentioned previously, a large counter. The register definition of the Real Time Counter is shown in Figure 12.

```
Register: Real_Time_Clock<31..0>

31                              0
|                               |
15.............0 15.............0
HHHHHHHHHHHHHHHH LLLLLLLLLLLLLLLL
[Hi]             [Lo]

...where,

H..H  = high-order value (in 65,536 us)
L..L  = low-order value (in 1 us)
```

Register Definition - Real_Time_Clock
Figure 12

Referring to Figure 12, both the upper and lower-halves of the Real Time Clock (RTC.Hi) can be both written to and read from during actual operation.

WARNING: Any reads of the RTC must be taken as needed. This means that if the entire 32 bits must be read, it should be done in a single word-mode operation. Performing this same function with two serial half-word operations will yield incorrect results. In addition, any reads of the Real Time

Clock have an uncertainty of approximately one microsecond. For writes, ONLY the half-word mode is acceptable for loading a value into the RTC.Hi or RTC.Lo register. This operation should only be attempted after reading the half-register of the RTC and determining that it will not overflow when the write is being performed.

When performing reads of the Real Time Clock, the Configuration/Status Unit must take some special action to ensure that the read data is valid (stable). This is required because the Switch and T-Bus clocks are not always ensured to be synchronous and thus the Real Time Clock may be advancing as it is being read. The CSU accomplishes this goal in the following manner:

When a read request for the Real Time Clock is detected by the CSU, the CSU immediately asserts the external SIGA pin, T_NSPAUSE_SIGA, and sends a request across a handshake synchronizer to the RTC controller logic. The RTC controller logic then waits for the next occurence of the one microsecond increment pulse from: Real_Time_Prescaler<4>. When this occurs, the CSU is ensured of having a stable reading from the Real Time clock for at least one microsecond. The RTC controller logic then sends an acknowledgement back across the handshake synchronizer where the CSU, upon detecting this event, negates T_NSPAUSE_SIGA and allows the data to be read. This is what contributes to the one microsecond uncertainty mentioned above.

WARNING: The CSU relies on the fact that the requesting T-Bus master will ensure that the total time - from the next occurence of the one-microsecond increment pulse to the reading of data - will take no more than 1 us. This time includes the synchronizer delay from the RTC controller, the response time of the CSU, and time for any pauses that the T-Bus master may assert. Excluding the assertion of those pauses (T_NMPAUSE_xxxx) from the T-Bus master, the delay in the SIGA will be: 2*p(R_CLK) + 6*p(T_CLK) nanoseconds. The "p" represents the period of the indicated clock in nanoseconds. Therefore, the T-Bus master should use EXTREME caution when causing the assertion of T_NMPAUSE_xxxx. Beyond that, the CSU cannot guarantee the accuracy of the read data!

5.1.1.2.2 Time Of Next Interrupt

The Time Of Next Interrupt or TONI registers, are two 32-bit registers (A and B) which in combination with the Real Time Clock, are used to schedule an interrupt to occur at some moment in the future. Both registers, and their associated control logic, are completely independent from each other although they both interact with the Real Time Clock.

The TONI control logic performs a 32-bit subtraction between the current TONI_A (TONI_B) register values and the value of the entire Real Time Clock each time the OMSP is valid. Whenever this subtraction yields a negative (two's-complement form) number, the SIGA sets (=1) the bit: TONIA_Config.Status (TONIB_Config.Status).

Normally, whenever time the Status bit is asserted, an external pin, M_TONIA_INT (M_TONIB_INT), is also asserted (=1). This can be enabled/disabled - asynchronously to the OMSP - by setting the TONIA_Config.Enable (TONIB_Config.Enable) bit to a 1/0. Disabling will force ONLY the pin to a "0." The associated status bit will still reflect the result of the current subtraction. Figure 13 illustrates the TONI register definition.

```
Register: Time_Of_Next_Interrupt

31..............................0
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT

...where,

T..T = interrupt value
```

Register Definition - Time_Of_Next_Interrupt
Figure 13

Figure 14 illustrates the TONIA(B) configuration register definition.

```
Register: TONIA(B)_Config"

31.............................0
-------------------------------E (write)
------------------------------SE (read)

...where,

E = asynchronously enable external pin
  = 0 disable M_TONIA(B)_INT external pin
  = 1 enable M_TONIA(B)_INT external pin

S = status
  = 0 TONIA(B) interrupt is not active
  = 1 TONIA(B) interrupt is active
```

Register Definition - TONIA(B)_Config
Figure 14

The actual subtraction that is performed to initiate the interrupt is shown in Figure 15.

```
TONIA(B)_Config<1> = 1 IFF,

   (TONIA(B)<31..0> - RTC<31..0>) < 0

...where TONIA(B) and RTC are treated as unsigned
   32-bit numbers and the difference is treated
   as a two's-complement number.
```

Rule - Time of Next Interrupt Calculation
Figure 15

When performing writes to the TONI register, the Configuration/Status Unit must take some special action to ensure that the TONI register is not updated in the middle of the difference operation. The CSU accomplishes this goal in the following manner:

When a write request for the TONI register is detected by the CSU, the CSU immediately asserts the external SIGA pin: T_NSPAUSE_SIGA and sends a request across a fixed-delay handshake synchronizer to the TONIA(B) controller logic. The TONIA(B) controller logic then waits for the next occurance of the OMSP before it actually loads the TONIA(B) register. Because of pipelining, the TONIA(B) Subtraction Unit is ensured of having exactly one microsecond in which to complete the subtraction. The TONIA(B) controller then sends an acknowledgement back across the handshake synchronizer where the CSU, upon detecting this, negates T_NSPAUSE_SIGA, thus freeing-up the T-Bus master. This means, of course, that the SIGA will assert T_NSPAUSE_SIGA for approximately one microsecond.

5.1.1.2.3 Priority Time Slot

The Switch protocol provides a mechanism by which initial messages may be transmitted at various levels of priority in order to place an upper bound on remote access time. Normally, this priority is set by the T-Bus bits, T_PRIORITY<1..0>, during the request phase of the T-Bus transaction. In this case, the initial message is transmitted/retransmitted with the priority set during the T-Bus transaction which initiated the message. However, the Requestor can also force these bits to their EXPRESS value independently of the T-Bus transaction via the Priority Time Slot mechanism.

This mechanism works by assigning each Requestor a particular active time slot which is based on the value of the Real Time Clock. When that time slot "arrives," any pending Intital Switch message in the Requestor will have its priority raised to the EXPRESS level (=00). The priority is "sticky" in that once raised to EXPRESS, it remains there until the T-Bus initiates a new Initial Switch message. This new Intital message updates the priority with the value of T_PRIORITY<1..0>, as normal.

The equation for determining the active Priority Time Slot is shown in Figure 16. This equation takes a slot value

Priority Time Slot is active IFF the equation,

(RTC.Lo<15..0> !$ PTC.Slot<15..0>) # PTC.Mask<15..0>

...is all 1's

Rule - Priority Time Slot Promotion
Figure 16

(PTC.Slot), compares it on a bit-by-bit basis with a portion of the Real Time Clock (RTC.Lo) and then logically "or's" the result with the priority slot mask (PTC.Mask). It then detects the result for the presence of all "1's." Essentially, the RTC.Lo and the PTC.Slot are compared for equality on a word basis with some of the bits excluded, or "don't cared," in the comparison. A given bit position is excluded by setting the corresponding bit position in the Mask subfield to a "1". The Mask and Slot subfields, which are defined in Figure 17, are programmable via the Configuration/Status Unit. The Priority Time Slot function can be disabled so that it NEVER promotes the priority of any message by negating (=0) the ConnfigB.Ena_Priority_Promotion bit. The fully programmable capability of the Priority Time Slot allows the slot to be valid at different nodes in almost any order. It also allows the period of occurence of the slot at a given node to be adjusted from constant up to 65 ms. Of course, the minimum time that a "slot" can be active at a given Requestor is one microsecond. Note that it is possible for the "slot" to arrive while the Requestor is sending out bids. This could result in one Bid being sent at lower priority and the remaining bid(s) sent at EXPRESS priority. However, logic in the Requestor ensures that no updating of priority occurs DURING Bid transmission. In addition, no updating will occur while the Requestor is either "idle" or "waiting." The "waiting" state is where the Requestor STU is waiting for a slotted/random start criterion to become valid.

Note that the purpose of the Priority Slot Value is NOT to ensure that a single high priority message be present in the Switch at any given time. Rather, the goal is to define the

```
Register: Priority_Time_Config<31..0>

31                               0
|                                |
15.............0 15..............0
SSSSSSSSSSSSSSSS MMMMMMMMMMMMMMMM
[Slot]           [Mask]

where,

S..S  = slot value
M..M  = mask value
```

Register Definition - Priority_Time_Config
Figure 17

maximum bandwidth of priority messages to make the servicing of these messages as predictable as possible. In addition, the Priority Time Slot mechanism only applies to Initial Switch Messages (locked or not), which are always attempting to make a connection with some downstream node. Subsequent messages do not send Bids and thus are not affected by the Priority Time Slot mechanism.

5.1.1.3 Function Request Types

The Requestor handles various types of function requests from a T-Bus master. Those functions include read and writes of either bytes, words, or multiple words. Byte reads/writes may be of one to four bytes but must NOT wrap across word boundaries.

WARNING: It is important not to violate word wrapping because the Requestor does NOT check for this condition. Word reads/writes MUST be word-aligned and multiple read/writes are limited to a maximum of four words.

5.1.1.4 T-Bus Request Screening

T-Bus requests to the BIU of the Requestor are screened for both context errors and T-Bus protocol errors before any action is taken on them. Protocol errors include such things as a T-Bus master requesting an illegal (=00) T_PRIORITY field or illegally wrapping across word boundaries. Currently, protocol errors are NOT detected. Context errors, mostly relating to errors in handling locking, are listed in Figure 18.

1) Requestor was asked to access a node within a locked sequence which is different than the node which opened that sequence. (Lock Address Error)

2) Requestor was asked to MAINTAIN a remote lock when it was never opened. (Maintain Present Error)

3) Requestor was not asked to MAINTAIN, BYPASS or OPEN a lock that was not yet explicitly released with FREE-LOCK. In other words, a NORMAL was issued while the Requestor was locked. (Maintain Absent Error)

Requestor T-Bus Screening Errors
Figure 18

Any of these errors will cause the Requestor to return an ERROR response with the appropriate error code on the T-Bus (See: "Error Detection and Reporting"). In addition, no Switch message will leave the STU. If the Switch path happens to be locked, any of these errors will also cause the BIU to initiate a sequence which will tear-down the Switch path (drop-lock) providing certain conditions are met. See "Locked Sequences" for more details.

NOTE: The Requestor, if unlocked, will treat a BYPASS in the same manner as a NORMAL Function Request; that is, it will NOT open a lock.

5.1.1.5 Initial Message Start/Retry Criterion

The Requestor can use one of several different methods to decide when to first begin transmission of an Initial Message and when to retry that transmission if the Switch rejects it. These methods are refered to as: slotted, random and immediate. The start transmission time can be programmed to correspond to either one of two fixed time slots, one of two random numbers, or immediate transmission. The retry can correspond to either one of two fixed time slots or one of two random numbers. Only some combinations of these start and retry criterion are available for a given initial message.

The operation of random and slotted start and retry are described first. The process of selecting the various random/slotted start and retry criterion for a given message is then explained.

5.1.1.5.1 Random Start/Retry

There is a random number generator associated with the start/retry criterion. The generator is 12 bits long and is continuously updated at the Switch frequency. Each time an initial message start/retry occurs and the random backoff is selected, a new random number is transfered from the generator to a 12-bit count-down counter. This counter, known as the backoff counter, also runs at the Switch frequency. When the backoff counter reaches -1, the Requestor is released to start/retry the initial message transmission.

Before the backoff counter is actually loaded with the random number, that number is logically "anded" with a 12-bit backoff mask. When the Requestor first attempts the start/retry of an initial message, the backoff mask is initialized, forcing some number of most significant contiguous bits of the random number to zero as they are loaded into the backoff counter. After a certain number of Switch rejects for the same initial message, the mask is "shifted left" to allow an increase in the maximum allowable value of the next 12-bit random number loaded into the backoff counter. Thus, the random backoff limit, in terms of Switch intervals, is a binary number of length 12, or 4096. Each time a Switch reject is encountered, the Requestor makes a decision about whether or not to shift the backoff mask.

That decision is made by adding a constant number to an accumulator after each Switch reject. Each time the accumulator overflows, the mask is shifted. Therefore, the mask may not change for several rejects.

In implementation, a register specifies randomization characteristics for the random start/retry criterion. This register is duplicated to allow for two sets of characteristics to be stored simultaneously. The mechanism for choosing one set or the other is described in a subsequent section. Each register is 8 bits long and specifies the initial mask setting, the constant value for accumulator addition and whether or not immediate start transmission is requested. These registers, and the random specifications which they describe, are subfields of the Transmit_Time_Config Register known as "Random0" and "Random1". Figure 19 illustrates the structure of the random registers.

```
Register: Transmit_Time_Config.Random0<7..0>,
          Transmit_Time_Config.Random1<7..0>

7......0
IMMMMMEE

where,

I     = immediate
EE    = accumulator addition constant
MMMMM = initial comparison mask
```

Register Definition - Transmit_Time_Config.Random0,1
Figure 19

Referring to Figure 19, the immediate field, "I", when "1", forces an initial random start to be immediate, ignoring any randomization parameters. For initial retries, the "I" field is ignored and the randomization parameters are always used. The constant value for accumulator addition is specified by the "EE" field. This number is added to a 3-bit accumulator, which is then tested for overflow. The initital backoff mask is specified by the 5-bit identifier, "MMMMM", which is loaded directly into a Johnson Counter. The output of the Johnson Counter is decoded to derive a 12-bit backoff mask as shown in Figure 20.

	mask identifier<5..0>	backoff mask<11..0>
increasing	000000	000000000001
count	000001	000000000011
|	000011	000000000111
|	000111	000000001111
|	001111	000000011111
|	011111	000000111111
|	111111	000001111111
|	111110	000011111111
|	111100	000111111111
|	111000	001111111111
|	110000	011111111111
V	100000	111111111111

Random Start/Retry Bit Mask Encoding
Figure 20

Figure 21 also shows how the counter advances once loaded with an initial value. This advancment, of course, is governed by the overflow of the 3-bit accumulator. Also note that the LSB of the backoff mask can never be cleared.

During the INITIAL start/retry, five of the mask identifier bits related to the initial message are specified by the "MMMMM" field in the random register. The sixth, most significant bit is ALWAYS initialized to "0". So, if MMMMM = "11111", the initial backoff identifier would be: "011111". In this case, the maximum possible random backoff is "1111110", or 128 Switch intervals (recalling that the backoff counter overflows at -1). Once the maximum identifer of "100000" has been reached, the counter "wraps around" and thus the next backoff mask will be "000000". The "multiply by two" effect of the left-shifting backoff mask is intended to implement an exponentially increasing random backoff. An equation summarizing the preceeding discussion is shown in Figure 21.

WARNING: The initial mask identifier MUST be a value which would result in a legal Johnson Counter value as shown in Figure 20. Legal Values would be: "00011" or "01111" for example. Illegal values would be: "00100" or "10110", for example.

$$\text{Maximum backoff (Switch intervals)} = 2^{[M + int(R*E/8)]}$$

...where,

M = initialized value of MMM bits
R = number of rejects
E = value of the EE bits

Equation - Maximum Exponential Random Backoff
Figure 21

5.1.1.5.2 Slotted Start/Retry

Slotted start and retry involves holding-off transmission based on the "arrival" of a pre-specified time slot. Once a slot has "arrived," a message assigned to that slot for starting can start transmission, and a message assigned to that slot for retry can retry transmission. The time slots are derived from the the comparison of the Real Time Clock and a register used to specify the slot characteristics. This register is duplicated to allow for two sets of characteristics to be stored simultaneously. The mechanism for choosing one set over the other is described in a subsequent section. Each register is 8 bits long and specifies the comparison mask, the comparison value, and whether or not immediate start transmission is requested. These registers, and the slot specifications which they describe, are subfields of the Transmit_Time_Config Register known as "Slot0" and "Slot1".

Figure 22 shows the structure of the slot registers.

```
Register: Transmit_Time_Config.Slot0<7..0>,
          Transmit_Time_Config.Slot1<7..0>

7......0
IMMDDDDD

.. where,

I     = immediate
MM    = mask specification
         00  4.0 us slot period
         01  2.0 us slot period
         10  1.0 us slot period
         11  0.5 us slot period
DDDDD = phase specification (restricted, see text)
```

Register Definition - Transmit_Time_Config.Slot0,1
Figure 22

Referring to Figure 22, the slot register contains three sub-fields: the compare mask field, specified by the two bit number, "MM"; the compare data field, specified by the five bit number, "DDDDD"; and immediate field, "I". The immediate field, when "1", forces an initial slotted start to be immediate, ignoring any slot parameters. For initial retries, the "I" field is ignored and the slot parameters are always used. The comparison for an active slot is made partially by comparing bits of the "D" sub-field with bits of the of the Real Time Clock and Real Time Prescaler. The "M" sub-field is used to either compare some of those bits with zeros or to ignore them in the comparison. This operation is shown in Figure 23. Referring to Figure 23, the D field can only take on values that are less than or equal to the setting of the Real_Time_Prescaler<4..0>.

WARNING: Values outside this range may cause the message to never be transmitted, and are therefore illegal.

Figure 23 also demonstrates the two properties of the slots:

given, nnnnnnnn = RTC.Lo<1..0> | RTP<5..0>

mm	compare	with	cycle period
00	000DDDDD	nnnnnnnn	4 us
01	X00DDDDD	nnnnnnnn	2 us
10	XX0DDDDD	nnnnnnnn	1 us
11	XXXDDDDD	nnnnnnnn	.5 us

Rule - Start/Retry Valid Slot Comparison
Figure 23

frequency and phase. the D field allows setting a number of phases equal to the setting of RTP<4..0> plus one. the M field allows the comparison to occur at varying time intervals.

Because of hardware limitations, the concatenated quantity, (RTC.Lo<1..0> | RTP<5..0>), does not act exactly like an eight bit counter. the RTP portion is actually running one switch interval "ahead" of the RTC.Lo<1..0> portion. This means that the RTC actually increments on the 000000-to-000001 transtion of the RTP portion, rather then on the 111111-to-000000 portion. A sample transition would look like that in figure 24.

5.1.1.5.3 Start/Retry Criterion Selection

A function request from a master on the T-Bus is transformed into a Switch message by the Requestor. Depending on certain parameters of that function request, the Requestor catagorizes the message into one of four Message Classes. Each of these classes will have a different start and retry criterion. The correspondence of start/retry criterion based on message classes is shown in Figure 25. A class is selected for each Switch message based on the state of three bits of T-Bus function request that initiated the message. Those bits are the T-Bus signals T_LOCKOP<1> and T_RR<1..0>. The Requestor uses the encoded state of those three bits to "look

RTC.Lo<1..0>	RTP<5..0>
10	11111100
10	11111101
10	11111110
10	11111111
10	00000000
11	00000001
11	00000010
11	00000011

Start/Retry Slot Comparison Count Sequence
Figure 24

Class	Start	Retry
00	Slot0/Immediate	Slot0
01	Slot1/Immediate	Slot1
10	Random0/Immediate	Random0
11	Random1/Immediate	Random1

Start/Retry Criterion based on Message Classes
Figure 25

up" the class of the message. The lookup table itself is a 16-bit register known as the Message_Classification Register. This register is defined in Figure 26. To illustrate the Message Start/Retry Criterion selection with an example, suppose that a function request to the Requestor may have set, (T_LOCKOP<1> | T_RR<2..0>) = 100. From Figure 26, this would cause the Requestor to look in the Message Classification register "D" subfield (for Locked Writes). In this subfield, the Requestor would find the "class of message" corresponding

```
Register: Message_Classification<15..0>

15                                              0
|                                               |
10     10     10     10     10     10     10    10
CC     CC     CC     CC     CC     CC     CC    CC
[A]    [B]    [C]    [D]    [E]    [F]    [G]   [H]


...where given that nnn = T_LOCKOP<1> | T_RR<1..0>,
   the subfields selected and the type of function
   request that selects them are,
```

nnn	Subfield	Function Request
000	MC.H	Unlocked Writes
001	MC.G	Unlocked Reads
010	MC.F	Auxilliary Unlocked Writes
011	MC.E	Auxilliary Unlocked Reads
100	MC.D	Locked Writes
101	MC.C	Locked Reads
110	MC.B	Auxilliary Locked Writes
111	MC.A	Auxilliary Locked Reads

Register Definition - Message_Classification
Figure 26

to the particular function request. If the "D" subfield were a "10", that particular message would have use the parameters in Random0 register for both message start and retry.

Both the Start/Retry Random and Start/Retry Slot registers are actually subfields of the Transmit_Time_Config Register. The bit definition for this register is illustrated in Figure 27. NOTE: Function requests can be forced to completely ignore the Message Classification register on a request-by-request basis. This occurs whenever a request is made and the T-Bus signal:

```
Register: Transmit_Time_Config<31..0>

31                                        0
|                                         |
7......0   7......0   7......0   7......0
IMMMMMEE   IMMMMMEE   IMMDDDDD   IMMDDDDD
[Random1]  [Random0]  [Slot1]    [Slot0]

...where, Random0, Random1, Slot0 and Slot1
   are previously defined
```

Register Definition - Transmit_Time_Config
Figure 27

T_SYNC is asserted (=1). In this case, the message is automatically classed as "00" and both initial transmission and retry criterion is taken from the Transmit_Time_Config.Slot0 register.

5.1.1.6 Switch Tx Protocol Timers

The Requestor contains timers which monitor the progress of the transmitted message and alert the Requestor if they detect an error condition. Specifically, there are two timers, the Reject Timer and Connection Timer. The Reject Timer determines how long the Requestor will attempt to open a Switch path in the face of Switch rejects. The Connection Timer monitors how long the Requestor will keep a Switch path open once the rejection period is finished. Parameters for both the Reject Timer and the Connection Timer are contained in the Protcol_Timer_Config Register.

5.1.1.6.1 Reject Timer

The Reject Timer is enabled at the beginning of the first attempt to transmit an initial message. Each time the Requestor receives a reject, it first examines the Reject Timer. If the timer has underflowed (the underflow is latched), the Requestor halts the transmission attempt and returns the Rej_TO Error code to the T-Bus master. The Requestor also tears-down the Switch path whether or not it was locked. Parameters for the Reject Timer are located in the Protocol_Timer_Config Register.

The Reject Timer is structured as a 4-bit down-counter clocked by a selectable prescaled time base. The reload value for the counter is contained in Protocol_Timer_Config.Cnt<3..0>. A 4-bit prescale parameter, located in Protocol_Timer_Config.Pre<3..0>, is used to select the desired prescale time base from one of sixteen possible frequencies. Those frequencies are derived from the low-to-high transition of bits of the real time clock, Real_Time_Clock.Lo<15..0>, as illustrated in Figure 28.

PRE	Q	PRE	Q
0000	0	1000	8
0001	1	1001	9
0010	2	1010	10
0011	3	1011	11
0100	4	1100	12
0101	5	1101	13
0110	6	1110	14
0111	7	1111	15

...where,

PRE = Protocol_Timer_Config.Cnt<3..0>
Q = selection from Real_Time_Clock.Lo, bit Q

Reject Timer Prescale Selection
Figure 28

The Reject Timer is continually loaded with TPC.Cnt<3..0> until it begins transmitting Bid #1. An equation for the maximum Reject timout is shown in Figure 29.

given,

CNT = Protocol_Timer_Config.Cnt<3..0>
PRE = Protocol_Timer_Config.Pre<3..0>

...then,

$$\text{Timeout} = \text{CNT} * 2^{(\text{PRE} + 1)} \text{ microseconds}$$

$$\text{...with an uncertainty of } 2^{(\text{PRE} + 1)} \text{ microseconds}$$

Equation - Reject Timeout
Figure 29

5.1.1.6.2 Connection Timer

The Connection Timer is loaded each time the Requestor sends Bid 1. This means that it is reloaded both juse before transmitting an initial message and after the Requestor receives each Switch reject. Like the Reject timer, its underflow condition is latched.

The Connection Timer's timeout has two different effects depending on when it occurs. If the timeout occurs while the Requestor is waiting for a message acknowledgement (M_ACK), the Switch path is torn-down (whether locked or not) and a Conn_TO Error is returned to the T-Bus master. If the timeout occurs while a Switch path is locked, but after the M_ACK was received, the Requestor will teardown the Switch path but cannot return an error to the T-Bus master immediately. Rather, it waits until the next T-Bus master makes a request to return a Wait_TO Error. In the "race condition" case where the M_ACK and connection timer underflow occur on the same clock edge, a Conn_TO Error is detected.

The Connection Timer is structured as an 8-bit down-counter clocked at 1 Mhz by a bit from the Real Time Prescaler, Real_Time_Prescale<5>. The counter underflows at -1. The reload value for the counter is contained in, Protocol_Timer_Config.Con<7..0>. The equation for the maximum connection timeout is shown in Figure 30.

given,

CON = Protocol_Timer_Config.Con<7..0>

then,

Timeout = CON + 1 microseconds

...with an uncertainty of 1 microsecond

Equation - Connection Timeout
Figure 30

5.1.1.6.3 Protocol Timer Programming

As previously mentioned, the parameters for the Protocol Timers are contained in subfields of the the Protocol_Timer_Config Register as shown in Figure 31.

5.1.1.7 Anticipation Support

The operation of the Requestor has two main goals: (1) to pass a T-Bus function request to the Switch as quickly and efficiently as possible, and (2) to return the corresponding function response from the upstream Switch message to the T-Bus master as quickly and efficiently as possible. Certain techniques can be used to take advantage of the expected operation of the logic in the function request and response path. These techniques are known collectively as "anticipation". The use of anticipation in achieving the two main goals of the Requestor are now discussed.

```
Register: Protocol_Timer_Config<15..0>

15                   0
|                    |
3..0   3..0   7......0
CCCC   PPPP   NNNNNNNN
[Cnt]  [Pre]  [Con]

...where, Cnt, Pre and Con have
   been previously defined.
```

Register Definition - Protocol_Timer_Config
Figure 31

5.1.1.7.1 Function Requests

Maximizing downstream function request efficiency in the Requestor involves balancing the desire for speed with the desire to maintain a streamlined Switch protocol. These tradeoffs become apparent when considering a multi-word write sequence. Here, the Requestor could signal its Switch Transmit Unit to begin transmitting as soon as possible after receiving the T-Bus request. This would always work if the T-Bus were guaranteed to supply all words of a multi-word transfer at a bandwidth equivalent to the bandwidth of the Switch. However, this will not always be the case as the variations between the clock frequency of the T-Bus and the Switch, combined with the ability of the current T-Bus master to assert PAUSE, create the possiblility of the STU "running out of data" in some circumstances.

To circumvent this problem, two immediate options are available. First, change the Switch protocol to allow the insertion of "null data word" fields when data is not available. Second, the Requestor could be programmed to signal the STU to start only after a specified number of words have been written during the data portion of the T-Bus transfer. The first alternative is unattractive because it increases Switch bandwidth and unnecessarily introduces complexity into the Switch message protocol. The second option is therefore implemented in the Requestor. The programmed parameter is known as, FQ_Anticipation, and can be set to any of the thresholds listed in Figure 32.

Register: Requestor_ConfigA.FQ_Anticipation<2..0>

210	Anticipation
000	after first data word transfered
001	after second data word transfered
010	after third data word transfered
011	after fourth data word transfered
1XX	immediately after T-Bus request

Register Definition - Requestor_ConfigA.FQ_Anticipation<2..0>
Figure 32

Since it is possible for the FQ_Anticipation to be set greater than the last word of a particular write, the Requestor will commit to transmission when either the last word has been written OR the Requestor FQ_Anticipation threshold has been reached - whichever occurs first. For example, if FQ_Anticipation were a "011" and a three word write occured, anticiaption would take place after the third word were written. In addition, an Interleaved request (I_INTERLEAVED=1) will cause a "1XX" setting to signal the STU in the cycle AFTER the T-Bus request. The threshold should be set based on the T-Bus and Switch clock frequencies, the maximum number of PAUSE assertions expected during a write, and the handshake synchronizer delay setting.

NOTE: For MOST applications, where no T-Bus master accessing the Switch will assert its T_NMPAUSE_xxxx, use the FQ_Anticipation=1XX setting.

5.1.1.7.2 Function Responses

Anticipation during function responses would allow the Requestor to take advantage of the synchronizer settling time by beginning the T-Bus request BEFORE the message checksum has been verified. Unfortunately, the Requestor is limited in the amount of anticipation that it can provide. Whatever anticipation the Requestor can extract from an upstream message, that anticipation has to be constant over all messages. This is because the Requestor STU-to-BIU handshake synchronizer has to compensate for message anticipation and cannot have its setting varied according to the expected upstream message type. And of course, even if the anticipator could vary its setting, the return message profile is not always known.

In fact, the Requestor SRU must assume a minimum expected upstream message length before starting anticipation. That minimum message length is two bytes. And since the SRU cannot tell if the assertion of Reverse is a Reject until the second byte, the minimum anticipation of the Checksum byte is one Switch Interval (for a function response to a write request). This then limits anticipation of all messages to one byte. By comparison, the Server has a minimum message length of 5 bytes and can thus take greater advantage of anticipation techniques.

As previously mentioned, Switch to T-Bus anticipation usually requires some minimum setting on the receiving T-Bus synchronizer. However, it turns out that no MINIMUM setting of Req_ConfigA.BIU_Synch<3..0> is required to compensate for the small amount of Requestor SRU anticipation. This is because pipeline overhead already accounts for this anticipation. However, a minimum setting IS required to meet the minimum settling time for the synchronizer. Fore more details on this subject, see: "Special Topics/Synchronization."

5.1.1.8 Locked Sequences

Sometimes an upstream T-Bus master wishes to perform several consecutive function requests to a locked remote T-Bus slave without the overhead of opening the Switch connection before each request. A mechanism known as Switch locking allows such multiple accesses by keeping the Switch path open between function requests. All transactions that take place during locking are known as locked sequences. A locked sequence has three distinct events: opening, maintaining and closing. Each of these events has different characteristics and restrictions for the Requestor.

5.1.1.8.1 Opening and Maintaining Locks

Opening a Switch lock begins with an otherwise normal function request from a T-Bus master that carries with it a request for "opening a lock" to a remote T-Bus slave. The upstream Requestor transfers the OPEN lock request to the downstream Server via a bit in the message protocol. Since the Switch path has not yet been established, either the Switch or the downstream Server may reject the message. A Switch reject will occur because of normal Switch contention and the Server reject will occur if the downstream target was locked. The Requestor, not knowing the source of the Switch reject, will simply retry the message transmission within the constraints of the Protocol Timers.

Assuming that the message finally does "get through" to the downstream Server, that Server "opens a lock" to the target T-Bus slave in accordance to the T-Bus protocol. Meanwhile the upstream Requestor, recognizing that it has established the beginning of a locked sequence, does not normally tear-down the Switch connection upon receiving an M_ACK unless an error was detected. This is discussed in detail in the "Auto Drop" section.

Once a locked Switch path is established with OPEN lock, it must be explicitly instructed to remain open by the upstream T-Bus master. This is accomplished by following the OPEN function request with either: another OPEN, a MAINTAIN, or BYPASS function request. Essentially, the Requestor takes no special action on either of these requests but does demand their presence. If the OPEN/MAINTAIN/BYPASS protocol is violated by subsequently initiating a NORMAL function request, the Requestor will respond to the offending T_Bus master with an ERROR and tear-down the Switch path. This mechanism is described in the "T-Bus Request Screening" section.

5.1.1.8.2 Dropping Locks

The Requestor has a flag, known as the "drop-lock request" flag, which causes the Requestor to negate Frame and return to its unlocked Idle state. Although the flag does not cause this action until the Requestor BIU is in its Locked Idle state, it can be set at any time. Once set, a drop-lock condition is said to be active. There are three distinct scenarios under which a drop-lock condition may occur: (1) A T-Bus master which is locked to the Requestor may issue a FREE-LOCK, (2) The Requestor issues an ERROR response (under certain conditions), and (3) a Connection Timer timeout.

Whatever the cause of the drop-lock condition, the Requestor BIU waits until it returns naturally to its Locked Idle state before taking action. Once there, the Requestor BIU will then enter the "unlock" state in which it will fulfill the drop-lock request flag by commanding the Requestor STU to negate Frame. During this state, the Requestor BIU will issue a REFUSED response to ANY T-Bus Master that accesses it. Once the Requestor BIU has been signalled by the STU that Frame was negated, the BIU returns to its Unlocked Idle state. Of course, the drop-lock request flag is then also negated. The downstream Server, knowing that it was previously locked, interprets the subsequent loss of its incoming Frame to be a FREE-LOCKS. The Server, sensing an unexpected loss of Frame, then issues a FREE-LOCKS to the local T-Bus.

The first drop-lock scenario - a FREE-LOCKS issued by a T-Bus master - is the most conventional. The FREE-LOCKS request is the only function request that is NOT explicitly transmitted to the downstream Server in the form of a message. Instead, the Requestor responds to a FREE-LOCKS by negating Frame to the Switch interface. Because the drop-lock condition can be entered at any time, a T-Bus master can issue a FREE-LOCKS at any time - whether the Requestor is idle or acting on a current split-cycle. However, the Requestor must be already locked to the T-Bus master which made the request. If not, the BIU will ignore the FREE-LOCK request.

In the ERROR response scenario, the Requestor will NEVER enter the drop-lock condition when the ERROR response is due to a Remote Class Error. However, it MAY enter the drop-lock condition when the ERROR response is due to an FQ or Switch

Class Error. This conditional action is described in the "Auto Drop" section. Error classes are discussed in the "Error Detection and Reporting"section. However, if those conditions ARE valid for a drop-lock, the Requestor processes the drop-lock in the same manner as the FREE-LOCKS scenario. Unlike the FREE-LOCKS however, drop-lock processing takes place almost immediately after the event which caused the drop-lock condition (responding with an ERROR). This is because the Requestor BIU always enters its Locked Idle state immediately after issuing an ERROR response.

The Connection Timer timeout scenario is slightly different from the previous two. When the Connection Timer times-out, it indirectly causes the drop-lock condition by eventually causing an ERROR response (Wait_TO or Idle_TO) by the Requestor BIU. This normally would be sufficient because the BIU would then enter the drop-lock condition, which would then signal the Requstor STU to negate Frame. However, one of the reasons that the Connection Timer may have timed-out was because the Requestor BIU had lost its T-Bus clock (T_CLK). In this case, Frame would never get negated. Therefore, the Requestor STU takes the initiative to negate Frame immediately after a Connection Timer timeout. For consistency, the drop-lock mechanism continues as normal. When the Requestor STU finally gets the request from the BIU to negate Frame, the STU simply ignores that request.

5.1.1.8.3 Auto Drop

Auto drop is a parameter set by the Req_ConfigA.Ena_Auto_Drop bit. When asserted (=1) the Requestor will be permitted to enter the drop-lock condition whenever an ERROR response is generated because of an FQ or Switch Class error. Otherwise, the Requestor will NEVER enter the drop-lock condition due to an ERROR response. This is because the only other class of Requestor error - Remote Error - will NEVER cause the drop-lock condition.

5.1.1.9 Stolen Bit Support

Because of the structure of the Switch message format, only one bit of Stolen information can be transferred between upstream and downstream nodes during a given message. The Requestor records the state of the Stolen bit during the word transfered in a byte write operation. It is this state that is relected in the Switch message. Normally, the Requestor expects the Stolen bit to be asserted only during a BYTE write operation. In fact, it is illegal to assert the Stolen bit to the Requestor during a multi-word operation.

NOTE: If the Stolen bit IS asserted during a multi-word write, the state of the the first word written is recorded.

The Requestor provides a mechanism to verify that the Stolen bits of all words in a multi-word write are zero, and prevent the message from being transmitted if this is not the case. The Ena_Stolen_Verify bit in the Req_ConfigB register, when asserted, will enable this verification of Stolen bits in a multi-word write. There is however, a small price to pay for this feature: the FQ_Anticipation register must be set to its MAXIMUM value (=011). This is because the Requestor must load all words of a multi-word write and verify the Stolen bits before commiting to transmission. The Requestor cannot "call back" the outgoing message. Figure 33 summarizes the rules for verifying the Stolen bit.

To enable the verification of Stolen bits on a multi-word writes,

1) Set FQ_Anticipation = 011, AND...

2)Assert (=1) the Req_ConfigB.Ena_Stolen_Verify bit

Rules - Stolen Bit Verification - Multi-Word Write
Figure 33

If the rules of Figure 33 are adhered to and a particular multi-word write has some of the Stolen bits asserted, the Requestor will respond with an ERROR ("Stolen_Verify" error code) to the T-Bus master. The Requestor, of course, will NOT transmit the message in this case.

For single-word reads, the Requestor presents to the T-Bus a Stolen bit (T_AD<32>) which is the same state as the Stolen bit in the upstream Checksum byte. For multi-word reads, the Requestor always assumes that the words of the transfer are NOT Stolen until it encounters an asserted Stolen bit in the Checksum byte. When this occurs, only the last word received by the Requestor is assumed to be Stolen. This fact is transmitted to the T-Bus by asserting T_AD<32> during the transfer of the last word on the T-Bus.

5.1.1.10 Quick Drop

The Requestor STU has an option which enables it to negate Frame during an Initial Message as soon as the STU detects an asserted Reverse. This can be done without the STU actually waiting to see if Reverse is going to be a Reject or an actual message. This action is allowed only when the STU is transmitting an Initial Message (NOT an Initial Locked Message) because in this situation, the only possible responses are: Reject or an upstream Switch message. In either case, the Requestor will negate Frame immediately if the bit: Requestor_ConfigB.Ena_Quick_Drop is asserted (=1). Essentially, Quick Drop is an optimization which will free up the Switch earlier - although only by one Switch Interval - than if Quick Drop were not enabled.

5.1.1.11 Reverse Profile Monitoring

The Requestor is enabled to monitor the profile of Reverse for errors asserting (=1) the Req_ConfigB.Ena_Rev_Err bit. Once enabled, the Requestor will report a Switch Class Error (Reverse_Error) whenever it observes an incorrect state for Reverse during an upstream message. Since there is more than one possible Reverse profile for a given Function Request, not every Switch Interval of Reverse can be checked for a given state (0/1) because either state may be valid. However, when the Reverse profile is incorrect in ANY place that is checked, a Reverse_Error is reported.

Figure 34 illustrates how the Requestor checks the Reverse profile. The "x's" represent where either state is valid and is therefore not checked by the Requestor.

```
TYPE      #WORDS               RETURN MSG FORMAT
=====     =========            ================

                               +--- first received
                               |
                               V
write     any                  xxL

read      non-multi            xxHH,HL

          two-words            xxHH,HL
             "                 xxHH,HxHH,HL

          three-words          xxHH,HL
             "                 xxHH,HxHH,HL
             "                 xxHH,HxHH,HxHH,HL

          three-words          xxHH,HL
             "                 xxHH,HxHH,HL
             "                 xxHH,HxHH,HxHH,HL
             "                 xxHH,HxHH,HxHH,HxHH,HL

...where,

    x = don't care
    H = check for Reverse = 1
    L = check for Reverse = 0
```

Requestor Reverse Profile Monitoring
Figure 34

NOTE: The Requestor will NOT specifically check that Reverse was negated (=0) when the Function Request was initiated. However, it DOES begin looking for a 0-to-1 transition of Reverse in order to recognize the beginning of the upstream message. Therefore, if Reverse were to be "hung high" when the Requestor began its Function Request, the Requestor would eventually timeout the Connection Timer.

5.1.1.12 Error Detection and Reporting

Errors delivered by the Requestor to an initiating T-Bus master can be divided into three classes depending on which part of the SIGA detects them. The classes are: 1) FQ Errors - which are detected by the BIU from the original Function Request; 2) Switch Errors - which are detected by the STU and SRU because of Switch interactions and 3) Remote Errors - which are detected by the downstream Server and are "reflected" up to the initiating T-Bus Master.

For a given Function Request/Response sequence, errors from different classes can occur simultaneously. Since only one error can be reported at a time, a sense of "priority" exists between error classes. If there is a FQ Error, it always be reported, regardless of the presence of Switch or Remote Errors. If there is no Local Error, than any Switch Errors will be reported, regardless of the presence of Remote Errors. If there is neither a Local nor a Switch Error, then and only then will any Remote Errors are reported.

Figure 35 shows the Error Codes for the Requestor which include the FQ and Switch type errors. Note that WITHIN a given Error Class, the errors are again not all mutually exclusive, and are therefore given "within-class" priorities. A more detailed description of the three Error Classes follows.

5.1.1.12.1 FQ Errors

FQ Errors are detected by the BIU during the original Function Request. Their detection, when enabled, will ALWAYS prevent the Function Request from initiating a Switch access. If the Requestor is unlocked, it will NOT assert Frame after detecting an FQ Error. If the Requestor is locked, it MAY immdiately tear-down the lock if certain conditions are met. See "Auto Drop" for more details.

FQ Error types and their definitions are illustrated in Figure 36.

Requestor Error Codes:

```
7      0
|      |
PPPPdcba
```

d	c	b	a	Requestor Error	Class
0	0	0	0	Maintain_Absent-(1a)	FQ
0	0	0	1	Maintain_Present-(1b)	FQ
0	0	1	0	Stolen_Verify(2)	FQ
0	0	1	1	Lock_Address-(3)	FQ
0	1	0	0	Wait_TO-(4a)	Switch
0	1	0	1	Idle_TO-(4b)	Switch
0	1	1	0	Rej_Abort(5)	Switch
0	1	1	1	Rej_TO-(6)	Switch
1	0	0	0	Reverse-(7)	Switch
1	0	0	1	Check-(8)	Switch

...where,

P..P = Requestor_ConfigA.Error_Prefix<3..0>.
Priority is from highest (1) to lowest (8).
Within a given priority, errors are mutually exclusive (i.e.,4a,b...).

Requestor Error Codes
Figure 35

Lock Address Violation - Requestor was asked to access a node within a locked sequence which is different than the node which opened that sequence. (only detected if configured to do so).

Maintain Present - Requestor was asked to MAINTAIN a remote lock when it was never OPENed. (only detected if configured to do so).

Maintain Absent - Requestor was not asked to MAINTAIN, BYPASS or OPEN a lock that was not yet explicitly released with FREE-LOCK. (only detected if configured to do so).

FQ Error Definitions
Figure 36

5.1.1.12.2 Switch Errors

Switch Errors are caused by a variety of conditions that are detected by the logic which monitors the progress of the Switch message as it enters and returns from the Switch interface. Unlike FQ Errors, Switch Errors are detected once the Switch transaction is already underway. They are reported to the T-Bus Master only when the the transaction is "finished", either normally or due to some timeout. Therefore, Switch Errors can only have a special affect on Frame during a locked sequence. In this case, the Requestor MAY immdiately tear-down the lock if certain conditions are met. See "Auto Drop" for more details.

Switch Error types and their definitions are illustrated in Figure 37.

Wait_TO - The Switch Transmit Connection Timer overflowed while the Requestor was waiting for a Function Response. (See: "Connection Timer")

Idle_TO - The Switch Transmit Connection Timer overflowed while the Requestor was in its idle state. (See: "Connection Timer")

Rej_Abort - The Switch Transmit Reject Timer was forced into overflow by the the REJ_ABORT input pin. (See: "Reject Timer")

Rej_TO - The Switch Transmit Reject Timer overflowed while the Requestor was attempting to open a connection. (See: "Reject Abort")

Reverse - The Requestor detected an incorrect polarity of the Reverse signal during a Function Response. (See: "Reverse Profile Monitoring")

Check - The Requestor detected an incorrect Checksum during a Function Response. (See: "Checksum Support")

Switch Error Definitions
Figure 37

5.1.1.12.3 Remote Errors

Remote Errors include: 1) errors which are detected within the Server logic itself, and 2) errors generated as T-Bus errors responses by a downstream T-Bus slave device. Both types errors are simply passed-through "as is" to the upstream Requestor. This Requestor simply "hands" them - without differentiation - to the initiating T-Bus Master. Remote Errors, unlike FQ and Switch Errors, can NEVER cause the Requestor to "drop" a lock.

A summary of the "Server-sourced" Remote errors, see: "Server/Operation/Error Reporting."

5.1.1.13 Disabled Operation

The Requestor can be disabled via a number of bits in the Requestor_ConfigB register. These include: Ena_REQ_BIU, Ena_REQ_STU, Ena_REQ_SRU, and Ena_REQ_CNT. These bits reset the four major blocks of the Requestor.

WARNING: In normal operation, these bits SHOULD ALWAYS BE ASSERTED/NEGATED AT THE SAME TIME. Otherwise, erratic Requestor operation may result.

When these bits are disabled (=0), the Requestor T-Bus interface will respond "REFUSED" to any T-Bus master that tries to access it. The Requestor will also ignore any assertions of REVERSE from the Switch interface.

5.1.1.14 Configuration Registers

The Requestor has two general Configuration Registers. They are: Requestor_ConfigA and Requestor_ConfigB. In general, both Configuration Registers are used to set miscellaneous parameters and enable/disable certain functions. Figure 38 shows the structure of Requestor_ConfigA.

Register: Requestor_ConfigA<31..0>

BIT/FIELD	FUNCTION (read/write)
<31..29>	REQ_Slave_Num[3]
<28>	Modulo_8
<27>	Columns_2
<26>	Ena_Auto_Drop
<25..23>	FQ_Anticipation[3]
<22..19>	STU_Synch[4]
<18..15>	BIU_Synch[4]
<14..11>	Error_Prefix[4]
<10..9>	Sixty_Five_Delay[2]
<8..6>	CSU_Slave_Number[3]
<5..1>	Real_Time_Prescale[5]
<0>	Columns_1

Register Definition - Requestor_ConfigA
Figure 38

The bit definition of Requestor_ConfigA is shown in Figure 39. This register contains mostly configuration bits that affect the run-time parameters of the Requestor. All bits are "high-true" and are reset (low) upon system reset. The structure of Requestor_ConfigB is shown in Figure 40. The bit definition of Requestor_ConfigB is shown in Figure 41. This register contains mostly configuration bits that enable/disable different functions and error reports of the Requestor. All bits are "high-true" and are reset (low) upon system reset.

5.1.1.15 Test Registers

The Requestor also contains a test register, Requestor_TestA. Its structure is shown in Figure 42. This register contains bits that are related to production testing of the SIGA, and unlike all other configuration registers, a read of Requestor_TestA does not yield the data last written. The write bits are initialized in their negated state and are related to

REQ_Slave_Num[3] - Configures the T-Bus slave number that the Requestor will respond with (on the T_SOURCE<2..0> pins) when making a Function Response..

Modulo_8 - Configures the Requestor to expect either a modulo-8 element (=1) or a modulo-16 (=0) Switch element.

Columns_2 - Configures the Requestor to expect either a 2-column (=0) or a 3-column Switch.

Ena_Auto_Drop - Enables the Requestor to tear-down a connection when a Function_Request or Switch class of error is detected (=1). Otherwise, these types of error will only be reported by the Requestor and no special action will be taken (=0).

FQ_Anticipation[3] - Configures the Requestor for the desired Function Request Anticipation. (See: "Anticipation Support")

STU_Sync[4] - Configures the settling time of the Switch Transmit Unit's (STU) handshake synchronizer which receives an "execute" signal from the Bus Interface Unit (BIU). This signal is used to initiate a Function Request on the Switch. (See: "Synchronization")

BIU_Sync[4] - Configures the settling time of the Bus Interface Unit's (BIU) handshake synchronizer which receives a "completed" signal from the switch transmit unit (STU). This signal is used to inicate that a function response has been received by the SRU. (See: "Synchronization")

Error_Prefix[4] - Configures the Prefix (T-Bus bits: D7-D4) of the Error code response for Requestor errors. (See: "Error Handling")

Sixty_Five_Delay[2] - Configures the pipeline delay of

M_SIXTY_FIVE pulse. Millisecond pulse as seen by the Requestor. WARNING DO NOT USE THE "00" SETTING. (See: "Real Time Clock" for further details)

CSU_Slave_Number[3] - Configures the Slave number that the CSU will respond with (on the T_SOURCE<2..0> pins) when making a Function Response.

Real_Time_Prescale[5] - Configures the terminal count of the Real Time Prescaler. (See: "Real Time Clock" for further details)

Columns_1 - Configures the Siga for a 1-column switch. (See: "Real Time Clock" for further details)

Bit Definition - Requestor_ConfigA
Figure 39

production testing of the SIGA. Their functional description is not within the scope of this document and therefore is not listed here.

WARNING: Bits of Req_TestA SHOULD NEVER BE ASSERTED DURING NORMAL OPERATION.

The read bits are used to observe the internal state of the Requestor. They will yield no useful information during normal operation.

Register: Requestor_ConfigB<31..0>

BIT/FIELD	FUNCTION (read/write)
<31..23>	Route_Address_Mask[9]
<22>	Ena_Stolen_Verify_Err
<21>	Ena_Maintain_Absent_Err
<20>	Ena_Maintain_Present_Err
<19>	Ena_Lock_Addr_Err
<18>	Ena_Wait_TO_Err
<17>	Ena_Idle_TO_Err
<16>	Ena_Rej_Abort_Err
<15>	Ena_Rej_TO_Err
<14>	Ena_Check_Err
<13>	Ena_Reverse_Err
<12>	Ena_Remote_Err
<11>	Ena_Quick_Drop
<10>	Ena_Priority_Promotion
<9>	Ena_Interleaver
<8>	Ena_Reject_Abort
<7>	Ena_Reject_Timer
<6>	Ena_Conn_Timer
<5>	Ena_Switch_Frame
<4>	Ena_REQ_BIU
<3>	Ena_REQ_STU
<2>	Ena_REQ_SRU
<1>	Ena_REQ_CNT
<0>	SPARE

Register Definition - Requestor_ConfigB
Figure 40

Route_Address_Mask[9] - Configures the randomization mask for the Bus Interface Unit's translation of the Logical Route Address to the Physical Route Address. (See: "Route Address Generation")

The Enable Error bits allow the indicated errors to be REPORTED (=1), or to be unreported (=0). Note that they DO NOT prevent the errors from occuring. The error functions that these bits enable/disable are described in the "Error Handling" section. The bits are as follows:

Error Bit
Ena_Stolen_Verify_Err
Ena_Maintain_Absent_Err
Ena_Maintain_Present_Err
Ena_Lock_Addr_Err
Ena_Wait_TO_Err
Ena_Idle_TO_Err
Ena_Rej_Abort_Err
Ena_Rej_TO_Err
Ena_Check_Err
Ena_Reverse_Err
Ena_Remote_Err

Ena_Quick_Drop - Enables (=1) or disables (=0) the Requestor Switch Transmitter to neagte Frame as early as possible on an Unlocked operation. (See: "Quick Drop")

Ena_Priority_Promotion - Enables (=1) or disables (=0) the Priority Promotion mechanism. (See: "Priority Promotion")

Ena_Interleaver - Enables (=1) or disables (=0) the Requestor's detection of the INTERLEAVED pin. (See: "Interleaver Support")

Ena_Reject_Abort - Enables (=1) or disables (=0) the Requestor's responding to the REJ_ABORT pin. (See:

"Reject Timer")

Ena_Reject_Timer - Enables (=1) or disables (=0) the operation of the Reject Timer. This bit will override the Ena_Reject_Abort bit.

Ena_Conn_Timer - Enables (=1) or disables (=0) the operation of the Connection Timer.

Ena_Switch_Frame - Enables (=1) or disables (=0) the assertion of the REQ_SW_FRAME pin. This function overrides any other function which effects the assertion of the REQ_SW_FRAME pin.

Ena_REQ_BIU - Enables (=1) or resets (=0) the Requestor Bus Interface Unit. WARNING: MUST ALWAYS HAVE THE SAME STATE AS: Ena_REQ_STU, Ena_REQ_SRU, Ena_REQ_CNT. (See: "Disabled Operation")

Ena_REQ_STU - Enables (=1) or resets (=0) Requestor Switch Transmit Unit. WARNING: MUST ALWAYS HAVE THE SAME STATE AS: Ena_REQ_BIU, Ena_REQ_SRU, Ena_REQ_CNT.(See: "Disabled Operation")

Ena_REQ_SRU - Enables (=1) or resets (=0) Requestor Switch Receive Unit. WARNING: MUST ALWAYS HAVE THE SAME STATE AS: Ena_REQ_BIU, Ena_REQ_STU, Ena_REQ_CNT. (See: "Disabled Operation")

Ena_REQ_CNT - Enables (=1) or resets (=0) Requestor Counter (Timer) Module. WARNING. MUST ALWAYS HAVE THE SAME STATE AS: Ena_REQ_BIU, Ena_REQ_STU, Ena_REQ_SRU. (See: "Disabled Operation")

Columns_1 - Configures the Requestor to expect a 1-column Switch (=1). In this case, the Requestor still uses Columns_2 to determine the Bid construction. When negated (=0), the Requestor uses Columns_2 for both number of bids to be sent AND bid construction. (See: "Downstream Message Components")

Bit Definition - Requestor_ConfigB
Figure 41

Register: Requestor_TestA<31..0>

BIT/FIELD	FUNCTION (write)
<31>	TST_BIU_GEN
<30>	TST_CNT_RTP
<29..27>	TST_CNT_RJT[3]
<26>	TST_CNT_COT
<25..22>	TST_CNT_RSR[4]
<21>	TST_TIO_RND
<20..0>	SPARE[21]

BIT/FIELD	FUNCTION (read)
<31>	TM_SSR
<30..27>	TM_RSR[4]
<26>	TM_COT
<25..24>	TM_RJT[2]
<23>	TM_RTP
<22>	SR_REJ_DET
<21..15>	SR_FSM[7]
<14>	ST_LOCKED
<13..1>	ST_FSM[13]
<0>	ST_RND_ROUTE

Register Definition - Requestor_TestA
Figure 42

5.1.2 Switch Message Protocol

The Requestor fully generates and supports the Butterfly Switch protocol. That support is described below.

5.1.2.1 Physical Route Address Generation

The Switch route address from the T-Bus field, T_AD<33..25>, is actually a logical address. This Logical Route Address, which has two possible sources, undergoes a transformation to derive the Physical Route Address. It is the Physical Route Address which is assembled into the bid symbols of the downstream Switch message. The Logical Route address is used in the calculation of the Header Partial Sum (see the Requestor/Checksum Calculation section). During a given function request, the two possible sources of Logical Route Address for the Requestor are the T-Bus (T_AD<33..25>) and the interleaver port (I_MOD<8..0>). The interleaver port is chosen if (1) the I_INTERLEAVED pin is asserted on the SIGA during the T-Bus request cycle AND (2) the Enable_Interleave bit in the Requestor_ConfigB register is asserted.

NOTE: It is assumed that both the T-Bus Master making the request and the Interleaver will force any unused bits in Logical Route Address to "0" as it is presented to the pins of the SIGA.

Whichever routing address is actually chosen, that 9-bit quantity undergoes a transformation. It is modified to allow the randomization of a selectable number of the routing bits. The random bits that potentially replace routing bits are obtained from a 9-bit random number generator, the Random Route Generator, which runs at the T-Bus clock rate. A bit in the route address can be specified as random by setting a corresponding bit in the Route Address Mask register to a "1". The transformation for the Physical Route Address generation can be expressed by an equation as shown in Figure 43. The first equation in Figure 43 represents the selection of either the Interleaver port or the T-Bus port for the Logical Route Address. The second equation randomizes selected bits in the Logical Route Address. The Route Address Mask is located in the Req_ConfigB configuration register.

5.1.2.2 Downstream Message Components

Some of the relavent aspects of the downstream Switch message components are now discussed. For a more detailed explanation of Switch message definition and protocol, see the reference

```
temp<8..0> = MOD & INT & INT_EN
             # [T_SNN & (!INT # !INT_EN)]

 PRA<8..0> = (RAND & RAM) # (temp & !RAM)

    ...where,

    T_SNN     = T_AD<33..25>
    MOD       = I_MOD<8..0>
    INT       = T_INTERLEAVED
    INT_EN    = Req_ConfigB.Ena_Interleaver
    RAND      = RAND<8..0>, random # generator
    RAM       = Route_Address_Mask<8..0>
    PRA       = Physical Route Address
```

Equation - Physical Route Address Generation
Figure 43

documents.

5.1.2.2.1 Header

The construction of the message header; which contains the bid symbols; varies depending on the modulus of the Switch, which can be either 8 or 16. The SIGA design will support both options, although the modulo-8 Switch is the most likely to be encountered. In addition, the Requestor can support a one, two or three column Switch. Figure 44 shows the format of the bid symbols in both modulus configurations. As seen from Figure 44, certain bid symbols may never be sent if the Switch is small enough. Note that a modulo-8 switch is always expected to have at least two switch columns and a modulo-16 can have as few as one. The random bits mentioned in Figure 44 are obtained from a separate random number generator known as the Random Route Generator.

```
7                               0
|                               |
0   0   P1   P0   Rd   Rc   Rb   Ra   (BID 1)   (first sent)
0   0   P1   P0   Rd   Rc   Rb   Ra   (BID 2)         V
0   0   P1   P0   Rd   Rc   Rb   Ra   (BID 3)   (last sent)
```

...where,

P1..P0 = priority from T-Bus: PRIORITY<1..0>
Ra..Rd = Physical Route Address (see below...)

COL1	COL2	MOD8	BID1 Rd Rc Rb Ra	BID2 Rd Rc Rb Ra	BID3 Rd Rc Rb Ra
0	0	0	n2 n1 n0 R8	R7 R6 R5 R4	R3 R2 R1 R0
0	0	1	0 R8 R7 R6	0 R5 R4 R3	0 R2 R1 R0
0	1	0	R7 R6 R5 R4	R3 R2 R1 R0	----------
0	1	1	0 R5 R4 R3	0 R2 R1 R0	----------
1	0	0	n2 n1 n0 R8	----------	----------
1	0	1	0 R8 R7 R6	----------	----------
1	1	0	R7 R6 R5 R4	----------	----------
1	1	1	0 R5 R4 R3	----------	----------

...where,

COL2	= Requestor_ConfigA.Columns_2
COL1	= Requestor_ConfigA.Columns_1
MOD8	= Requestor_ConfigA.Modulo_8
n1,n2,n3	= random bits
----------	= Bid is NOT transmitted

Bit Definition - Downstream Message Header
Figure 44

5.1.2.2.2 Body

The message body; which contains the command, address, data and checksum bytes; varies based on the type of message being sent downstream. The general format is shown in Figure 45. Figure 45, of course, shows a single word write message. For multi-word write transfers there would be correspondingly more data bytes. For a read message, the difference would be that all data fields would be missing and bit S would be forced to a zero.

NOTE: The current SIGA design ALWAYS forces the "F" bit to be a "0".

5.1.2.3 Checksum Support

The Requestor and Server each have two separate units of checksum logic. The first, known as the Transmit Checksum Unit, calculates the message checksum during its transmission. The second, known as the Receive Checksum Unit, calculates and verifies the checksum for the incoming message.

The elements included in the calculation of the checksum of a downstream message vary depending on the type of message being transmitted. For any initial message (locked or unlocked), the Requestor always initializes its Transmit Checksum Unit with the "flash" sum of the Logical Route Address. The Logical Route Address can, of course, come from either the MOD pins (interleaved access) or from the T-Bus (non-interleaved). For any locked messages, the Requestor always initializes its Transmit Checksum Unit to zero.

In the same way, the downstream Server must initialize its Receive Checksum Unit to ITS node checksum whenever it expects an initial message. This initialization value will, of course, match that calculated by a Requestor about to transmit to that Server's node. For locked messages, the Server will initialize its Receive Checksum Unit to zero, just as the Requestor does with its Transmit Checksum Unit.

In an upstream message, there are NEVER any routing bits to contend with. Therfore, the downstream Server always initializes its Transmit Checksum Unit to zero, as does the

```
7                                                   0
|                                                   |
L1    L0    R1    R0    S2    S1    S0    A24   (first sent)
A23   A22   A21   A20   A19   A18   A17   A16        |
A15   A14   A13   A12   A11   A10   A9    A8         |
A7    A6    A5    A4    A3    A2    A1    A0         |
D31   D30   D29   D28   D27   D26   D25   D24        |
D23   D22   D21   D20   D19   D18   D17   D16        |
D15   D14   D13   D12   D11   D10   D9    D8         |
D7    D6    D5    D4    D3    D2    D1    D0         |
                  .                                  |
   <possible addtional write words>                  |
                  .                                  |
                                                     V
F     0     0     S     CS3   CS2   CS1   CS0   (last sent)

 ...where,

L1..L0    = lock operation from T-Bus: T_LOCKOP<1..0>
R1..R0    = portion of field from T-Bus: T_RR<1..0>
            R1  R0
            ==  ==
            0   0    write
            0   1    read
            1   0    <unused>
            1   1    <unused>
S2..S0    = size information from T-Bus: T_SIZE<2..0>
A24..A0   = address information from T-Bus: T_AD<24..0>
D31..D0   = data information from T-Bus: T_AD<31..0>
F         = enable forward drivers
            F
            =
            0   disable forward drivers next clock
            1   enable forward drivers next clock
       S  = Stolen Bit
CS3..CS0 = message checksum
```

Bit Definition – Downstream Message Body (write)
Figure 45

Requestor's Receive Checksum Unit.

5.1.2.4 Checksum Calculation

The checksum for a downstream message is actually calculated in two parts. If the message is an initial (locked or unlocked) one, a partial sum of the message header is calculated (by separate logic) and stored in the Transmit Checksum Unit. Then, the Transmit Checksum Unit adds the initial value, if any, to the bytes of the body of the message as it is transmitted.

5.1.2.4.1 Header Partial Sum

The header partial sum is derived by considering only the Logical Route Address bits. This means that the priority and random bits are not included in the calculation. This approach eases the design of the checksum logic and makes it independent of the Switch modulus. The equation for this calculation is shown in Figure 46.

```
HPS<3> = R8 $ R7 $ R3
HPS<2> = R6 $ R2
HPS<1> = R5 $ R1
HPS<0> = R4 $ R0

...where,

HPS<3..0>  = Header Partial Sum
R8..R0     = Logical Route Address
```

Equation - Requestor Header Partial Sum Calculation
Figure 46

5.1.2.4.2 Message Checksum

As previously mentioned, the header partial sum is added to the body of a downstream message if and only if that message is an initial message. The message checksum calculation is shown in figure 47.

```
CS<3> = HPS<3> $ exor(L1,S2,A23,A19,A15,A11,A7,A3,
                      D31,D27,D23,D19,D15,D10,D7,D3,F)

CS<2> = HPS<2> $ exor(L0.S1,A22,A18,A14,A10,A6,A2,
                      D30,D26,D22,D18,D14,D9,D6,D2,0)

CS<1> = HPS<1> $ exor(R1,S0,A21,A17,A13,A9,A5,A1,
                      D29,D25,D21,D17,D13,D8,D5,D1,0)

CS<0> = HPS<0> $ exor(R0,A24,A20,A16,A12,A8,A4.A0
                      D28,D24,D20,D16,D12,D7,D4,D0,S)

...where,

exor'ed components from: "Bit Definition - Message Body"
CS<3..0>  = message checksum
HPS<3..0> = Header Partial Sum
```

Equation - Message Checksum (see text)
Figure 47

Figure 47 shows the calculation for a single word write message. For write messages with more words, those bytes would be included in the same manner as the data bytes in the figure. For read messages, the data field would be missing entirely from the calculation.

NOTE: The "F" field is always "0".

5.1.2.5 T-Bus Interface

The Requestor supports the standard T-Bus protocol with some small limitations. For one, the Requestor does NOT support unaligned transfers which fall accross word (32-bits) boundaries. In addition, when it is locked to a T-Bus Master and in its "WAIT" state, the Requestor will always issue a REFUSED LOCKED to ANY T-Bus query while it is busy processing a split-cycle request. This means that it will even REFUSED LOCKED to it's own T-Bus master! This is a hardware optimization which should cause no problems. The locking T-Bus master normally has no reason to query the Requestor until the Requestor finishes its current operation.

Figure 48 shows the Requestor's state-dependent T-Bus responses while it is in some of its more interesting states.

```
 NEXT RESPONSE                     CONDITION
 =============  ============================================

 State = IDLE  (satisfied a function request, waiting for new one):

  PROMISE          !LOCKED & !DROP_LOCK &  read
  PROMISE          !LOCKED & !DROP_LOCK &  write & !multi
  MORE             !LOCKED & !DROP_LOCK &  write &  multi
  REFUSED          !LOCKED &  DROP_LOCK
  REFUSED LOCKED    LOCKED & !DROP_LOCK & !my_master
  PROMISE           LOCKED & !DROP_LOCK &  my_master &  read
  PROMISE           LOCKED & !DROP_LOCK &  my_master &  write & !multi
  MORE              LOCKED & !DROP_LOCK &  my_master &  write &  multi
  REFUSED           LOCKED &  DROP_LOCK


 State = WAIT  (waiting for function request to traverse Switch)
     -or-
 State = BREQ  (making T-Bus request for T-Bus with split response):

  REFUSED          !LOCKED
  REFUSED LOCKED    LOCKED
```

Requestor T-Bus Responses (partial list)
Figure 48

5.1.2.6 LCON Interface

The LCON is a the physical and logical link between the SIGA-Requestor and the "input" port of the Switch Gate Array (SGA). In other words, for the SIGA, the LCON interface is the logical Switch interface. The LCON provides the Requestor with: 1) level conversion to and from the ECL levels of the SGA and 2) reclocking of data, Frame, Reverse and the 65 ms pulse to and from the SGA.

Figure 49 shows the Requestor's LCON (Switch) Interface Pins.

PIN NAME	TYPE	FUNCTION
R_DATA<7..0>	bidirectional	Requestor-LCON data bus
R_FRAME	output	Frame output to Switch
R_REVERSE	input	Reverse input from Switch
R_NENA_BACK	output	LCON TTL driver enable
M_SIXTY_FIVE	input	65 ms timer input

Requestor LCON (Switch) Interface Pins
Figure 49

5.1.2.6.1 Data Bus Enable Control

The Requestor controls the enables of both its own output drivers and the LCON's output drivers to the SIGA-LCON data interface - R_DATA<7..0>. To control its own output drivers, the Requestor generates an internal signal called, nena_out. When asserted (=0), nena_out enables the Requestors R_DATA<7..0> drivers. To control the LCON, the Requestor provides the R_NENA_BACK signal to directly enable(=0)/disable(=1) the LCON's output drivers to R_DATA<7..0>. In addition, R_NENA_BACK, after a flip-flop delay, is used to enable/disable the LCON's Switch data ECL interface bus. When the Requstor is driving R_DATA<7..0>, it is in "Talk" Mode. When the LCON is driving that bus, the Requestor is in "Listen" Mode.

There are two major reasons why the Requestor separately provides the R_NENA_BACK signal First, the Requestor already "knows" which direction the bus should be driving, and therefore this logic need not be repeated in the LCON. Second, this configuration gives the Requestor the ability to prevent bus contention.

Bus contention can occur when the direction of data changes on the LCON interface. If R_NENA_BACK changed on the same clock edge as nena_out, there would be contention on R_DATA<7..0> each time both of those signals changed. However, because of timing skew and minimum delays, contention is actually only a problem when the Requestor tries to enable its own drivers as it disables the LCON's backward drivers. This occurs during the transition from Listen to Talk Mode. But since the Requestor has separate control of its own output drivers and the LCON's, it can prevent this case of contention. It does this by inserting a "dead" state for one Switch Interval where neither the Requestor nor the LCON is driving R_DATA<7..0>.

The Requestor is considered "quiescent" when it is not transmitting messages and not waiting for any replies. When quiescent, the Requestor is in Talk Mode. The Requestor tries to stay in Talk Mode whenever possible, making the transition to Listen only for the absolute minimum time necessary. This situation is the mirror image to the Server. It is always in Listen Mode when quiescent and tries to stay in Talk mode for as little time as possible.

When the Requestor finishes transmitting the checksum of an Initial or Locked message, it transitions directly into Listen Mode. Once there, it waits for either a Reject (which could have been detected and latched during the message transmission) or a return message. When either of those two events are complete, the Requestor transitions back to the Talk Mode, via the dead state. Figure 50 shows this sequence for both a replied and a rejected Switch message. Note from Figure 50 that there is a dead state only when making a transition from Listen to Talk Mode. Although not show in the Figure, subsequent Locked messages act in the exact same manner.

```
Transmit Mode    tttttttttttttllllll ... llllllllllldttttttttt
Frame            ______HHHHHH_xxxxx
Reverse          __________________  ... ___HHHHH______________
R_DATA<7..0>     xxxxxmmmmmmmcxxxxx  ... xxxmmmmcxxxxxxxxxxxxx
nena_out         _____________HHHHH  ... HHHHHHHHHHHH__________
R_NENA_BACK      HHHHHHHHHHHHH_____  ... _____________HHHHHHHHHH

                 (a) Message Returned, No Reject


Transmit Mode    tttttttttttttlldtttttttt
Frame            ______HHHHHH_xxxxxxxxxxx
Reverse          _________H_______________
R_DATA<7..0>     xxxxxmmmmmmmcx-xxxxxxxxx
nena_out         _____________HHH_________
R_ENA_BACK       HHHHHHHHHHHHH__HHHHHHHHH

                 (b) Reject Latched during Tx
```

...where,

```
    m..m   is a message
       c   is the checksum
       t   is Talk Mode
       l   is Listen Mode
       d   is the dead state
       -   floating bus
```

Timing - Requestor Switch Data Bus Enable
Figure 50

5.2 Server

The Server is described from the point of view of its overall operation and its two major interfaces: the T-Bus interface and the Switch Interface.

5.2.1 Operation

The operation of the Server is described by discussing its major functions.

5.2.1.1 Overview

The Server is a local T-Bus master which creates a logical coupling to a physically remote T-Bus slave via the Switch. The Server acts as the "responder" of this coupling on the Switch and thus can be thought of as a "master" on the T-Bus but a "slave" to the Switch. Referring to Figure 51, the Server contains three major functional units: Bus Interface Unit (BIU), Switch Tx Unit (STU), and the Switch Rx Unit (SRU). The BIU is clocked by the T-Bus clock and both the STU and SRU are clocked by the Switch clock. Interfacing of control signals between these units is accomplished with handshake synchronizers, as shown. The SRU receives function requests from the Switch and translates those requests into commands for the BIU. The BIU handles all of the T-Bus transactions of the Server to comply with a given function request. When a T-Bus slave device responds to a function request, the BIU picks-up that response and passes it as a command to the STU. The STU then initiates an upstream Switch message to return the function response.

The SRU detects the downstream message of a function request, verifies the checksum and alerts the BIU of the incoming message and the checksum status. The SRU also causes Switch rejects when either the BIU has explicitly commanded this action or when the SRU decides to on its own. The BIU will command a Switch reject when a function request is trying to access a T-Bus device which is locked to a T-Bus device other than the Server. The SRU will NOT initiate a reject without a command from the BIU and thus CANNOT correctly handle a nonsequitur downstream message. A nonsequitur would occur, for instance, when the SRU receives a function request (in the form of a downstream message) and knows that the STU has not even begun to send an upstream Switch message in response to the last function request.

The SRU has the additional responsibility of initiating a FREE-LOCKS command to the BIU when the Switch path is locked and

```
                                                          |~|
          +-------+              +----------+             | |
 |        |       |   +----+     |          |             | |
 |        |Switch |<--|sync|---->|          |             | |
 |------->|Rx     |   +----+     |          |             |T|
 |=======>|Unit   |=============>| Bus      |             |B|
S|        |       |              | Interface|             |U|
W|        |       |<--+          | Unit     |<=========>  |S|
I|        +-------+   |          |          |             | |
T|                    |          |          |             | |
C|        +-------+   |          |          |             | |
H|        |       |<--+          |          |             | |
 |        |       |              |          |             | |
 |<-------|Switch |<=============|          |             | |
 |<=======|Tx     |   +----+     |          |             | |
 |        |Unit   |<--|sync|---->|          |             | |
 |        |       |   +----+     |          |             | |
 |        +-------+              +----------+             | |
                                                          |~|
```

Server Block Digram
Figure 51

the incoming Frame signal negates unexpectedly. This situation is known as "dropping a lock" and is the ONLY time when the Server does not create a Function Response as a result of an explicit function request.

The SRU/BIU interface is a streamlined request/response type interface where for each SRU request there is an BIU response. The SRU presents an encoded function request to the BIU and sets an "execute" flag. When the BIU is done operating on that request, it sets a "done" flag and returns a status code and data to the SRU. The SRU also has the ability to "interrupt" the pending BIU operation. This is accomplished with a "terminate" handshake signal from the SRU. The "terminate" handshake receives a "terminate-done" from the BIU when the BIU finishes. This "interrupt" path is used for situations where the BIU may be indefinitely "hung" because a failed T-Bus slave is continuously asserting Slave pause.

Both the SRU and BIU are responsible for handling their own functions independently and they have very little real-time knowledge of each other's state. This approach simplifies the Server design and carries the request/response philosophy throughout the system.

The BIU has three major responsibilities: (1) initiate T-Bus requests to comply with a command from the SRU; (2) receive responses from the T-Bus; (3) transfer those responses, along with any error indications, to the STU. To accomplish the T-Bus request/response transfer, the BIU supports most of the T-Bus protocol.

The STU is a fairly simple device. It acts on a function response from the BIU and initiates the upstream Switch message to carry out that response. The STU also is responsible for assembling and transmitting the data in an outgoing message.

5.2.1.2 Anticipation Support

The operation of the Server has two main goals: (1) to pass a downstream Switch function request to a T-Bus slave as quickly and efficiently as possible, and (2) to return the corresponding function response from that T-Bus slave as quickly and efficiently as possible. Certain techniques can be used to take advantage of the expected operation of the logic in the function request and response path. These techniques are known collectively as "anticipation". The use of anticipation in achieving the two main goals of the Server are now discussed.

5.2.1.2.1 Function Requests

Maximizing downstream function request efficiency in the Server involves balancing the desire for speed with the desire for eliminating unwanted side-effects. The speed issue relates to the desire to transfer data from an incoming Switch message to the T-Bus as soon as it is available. Unwanted side-effects involve taking any action on the T-Bus that would cause a change in stored data in a T-Bus slave device given that the downstream message was corrupted. Two extreme approaches could be taken in the design of the Server. First, the Server could wait until the entire downstream message had been received, including the checksum; verify the checksum; and then begin access to the T-Bus. Second, the Server could begin access to the T-Bus immediately upon receiving a downstream message.

The first approach would cause the Server to waste valuable time in accessing the T-Bus, and the second could possibly cause unwanted side effects. Since one of the design goals of the Butterfly II is that data integrity should take precidence over speed, a compromise between the first and second approaches is implemented in the Server.

The Server "anticipates" the verification of the downstream checksum and begins it's request for T-Bus drivership. The timing is set up such that the Server BIU is commanded by the SRU to make a bus request at a specific moment in time. In fact, the SRU commands the BIU (input to the BIU synchronizer) to begin the T-Bus request EXACTLY five Switch intervals before the "Checksum_is_OK" signal is valid. This is true for both reads and writes. Therefore, the synchronizer setting, Server_ConfigA.BIU_Xfer_Sync<3..0> should be set accordingly. See "Synchronizer Settings" for more details.

5.2.1.2.2 Function Responses

The Server uses a similar technique as the Requestor for anticipating T-Bus transactions. Of course, in the case of the Server, the anticipation is for Function Responses rather than Function Requests. The Server_ConfigA.Multiv_Head_Start<1..0> register is used to set the anticipation for multi-word writes. Figure 52 illustrates its settings. In addition, the Server_ConfigA.Ena_Byte_Head_Start bit, when asserted (=1), begins anticipation whenever the T-Bus Slave responds with EARLY-ACK.

Normally, the Server will take anticipate for reads only. However, in some hardware configurations it is possible to anticipate on writes. When Server_ConfigB.Ena_Wr_Head_Start is asserted (=1), the Server treats writes exactly the same way as reads for all purposes.

Register: Server_ConfigA.Multi_Head_Start<1..0>

10	Wait until...
00	all words are transfered
01	three words have been transfered
10	two words have been transfered
11	one word has been transfered

Register Definition - Server_ConfigA.Multi_Head_Start<1..0>
Figure 52

WARNING: Using anticipation in multi-word writes can cause unusual side-effects if the multi-word write does not complete in time. This is because the Server SRU may mistakenly believe that the write data buffers are actually stable until the upstream Requestor has seen the Function Response and taken some action. As seen by the Server, this response takes quite long, at least 4-6 Switch Intervals. Thus, if the multi-word write takes only this long to complete, there is no problem.

WARNING: Using read anticipation requires that the T-Bus Slave issue an ERROR before transfering any data.

NOTE: The EARLY-ACK response has no meaning for multi-word reads or writes, and this response is ignored by the Server. Also, the Server must examine the T_RR field even though T_SPAUSE may be asserted.

5.2.1.3 Locked Sequences

The Server's handling of locked sequences parallels that of the Requestor and is described in the "Requestor/Operation/Locked Sequences" section. Like the Requestor, the Server's locked sequence has three distinct events: opening, maintaining and dropping.

The Server becomes locked if and only if it receives an Initial Locked message (OPEN , by definition is the command). It remains locked as long as it returns any function response except Reject. When a lock is dropped at the upstream Requestor, Frame is negated. As mentioned in the "Reqestor/Operation/Locked Sequences" section, a Requestor drop-lock function request can occur as the result of a T-Bus master issuing a FREE-LOCK or possibly a Requestor Switch Class error. The Server NEVER knows the reason for the drop-lock request, it simply issues the perfunctory FREE-LOCK to a T-Bus slave.

5.2.1.4 Stolen Bit Support

Because of the structure of the Switch message format, only one bit of Stolen information can be transferred between upstream and downstream nodes during a given message. Therefore, during byte reads, the Stolen bit from the Server's T-Bus is transported to the upstream Requestor exactly as it is read from T_AD<32> during the data transfer cycle of the T-Bus. For multi-word reads, the Server continues the T-Bus transaction, reading and storing all of the intended words even when it encounters a Stolen bit BEFORE the last word of the transfer.

However, when the Server finally transmits that data to the upstream Requestor, it acts differently depending on whether or not the data contains a Stolen bit. If it does not, all of the multi-word data is included in the upstream message and the Stolen bit in the Checksum byte is sent negated. If it does, the Server ends transmission of the data AFTER it sends the Stolen word, and it asserts the Stolen bit in the Checksum byte. The upstream Requestor always assumes that the words of a multi-word transfer are NOT Stolen until it encounters an asserted Stolen bit in the Checksum byte. When this occurs, the LAST word and only the last word received by the Requestor is assumed to be Stolen.

For byte write transfers, the Server presents the state of the Stolen bit in the downstream Checksum byte to the downstream T-Bus bit, T_AD<32>. For multi-word writes however, the state of ALL Stolen bits transported downstream is assumed by the Server to be "0". In this case, the Server will ignore the state of the Stolen bit in the downstream Checksum byte.

5.2.1.5 Error Reporting

Errors delivered by the Server (Requestor "Remote Error" Class) are transported by the Server to the upstream Requestor via the function response Switch message. Those errors may have one of two sources: they could originate from the Server itself, or they could be errors passed to the Server from a downstream Slave. The error codes due to the Server are shown in Figure 53.

```
Server Error Codes:

7      0
|      |
PPPPPPba

b a  Server Error
= =  ================
0 0  Downstream_Refused
0 1  Downstream_Write
1 0  Downstream_Late
1 1  Downstream_OTL
```

...where,

P..P = Server_ConfigA.Error_Prefix<5..0>

Server Remote Error Codes and Definitions
Figure 53

Their definitions are shown in Figure 54. Other remote slave errors are described in other system documents.

5.2.1.6 Disabled Operation

The Server can be disabled via a number of bits in the Server_ConfigB register. These include: Ena_BIU and Ena_SRU. These bits reset the two major blocks of the Server.

WARNING: In normal operation, these bits SHOULD ALWAYS BE ASSERTED/NEGATED AT THE SAME TIME. Otherwise, erratic Server

Downstream_Write - A downstream write error was detected from a T-Bus Slave while the downstream Server was sourcing data. Because of the direction of the data bus, the Server cannot return the actual error code.

Downstream_OTL - A downstream T-Bus Slave did not respond to the Server's request. Specifically, the Slave did not assert T_DRIVEN in the T-Bus cycle following the Servers' T-Bus request.

Downstream_Late - A downstream T-Bus slave responded with a LATE ERROR.

Downstream_Refused - A downstream T-Bus slave responded with REFUSED-LOCKED when the Server thought itself to be locked.

Server Remote Error Definitions
Figure 54

operation may result.

5.2.1.7 Configuration Registers

The Server has two general Configuration Registers, known as Server_ConfigA and Server_ConfigB, which are used to set miscellaneous parameters and enable/disable certain functions. The structure of Server_ConfigA is shown in Figure 55. The bit definition of Server_ConfigA is shown in Figure 56. This register contains mostly configuration bits that affect the run-time parameters of the Server. All bits are "high-true" and are reset (low) upon system reset. The structure of Server_ConfigB is shown in Figure 57. The bit definition of Server_ConfigB is shown in Figure 58. This register contains mostly configuration bits that affect the run-time parameters of the Server. All bits are "high-true" and are reset (low) upon system reset.

Register. Server_ConfigA<31..0>

BIT/FIELD	FUNCTION (read/write)
<31>	Ena_Wr_Head_Start
<30>	Ena_Byte_Head_Start
<29..28>	Multi_Head_Start[2]
<27..24>	RX_Init_CS[4]
<23..18>	Error_Prefix[6]
<17>	Ena_BIU
<16>	Ena_SRU
<15..12>	STU_Freed_Sync[4]
<11..8>	STU_Done_Sync[4]
<7..4>	BIU_Free_Sync[4]
<3..0>	BIU_Xfer_Sync[4]

Register Definition - Server_ConfigA
Figure 55

Dis_Frame - Disables the SRU by forcing it to see the incoming Frame negated, regardless of its actual state (=1). Otherwise, the SRU will see the actual incoming Frame. (=0). (See: "Disabled Operation")

Ena_SOC - Enables the SRU to recognize the start of a new connection (=1). Otherwise, the SRU will ignore this event (=0). (See: "Disabled Operation")

Dis_Check_Err - Disables the detection of checksum errors (=1). Otherwise, the detection is enabled (=0). (See: "Checksum Calculation)

SER_Slave_Num[3] - Configures the Slave number that the Server will place on the T_SOURCE<2..0> pins when it is making a T-Bus Function Request.

Ena_Wr_Head_Start - Enables the Server to anticipate during write-type Function Responses (=1). Otherwise, anticipation will only occur for read-type Function Responses. (See: "Anticipation Support")

Ena_Byte_Head_Start - Enables the Server to anticipate during byte-type Function Responses (=1). Otherwise, anticipation will not occur for byte-type Function Responses (=0). (See: "Anticipation Support")

Multi_Head_Start[2] - Configures the Server for the desired Function Response Anticipation for all multi-word operations. (See: "Anticipation Support")

RX_Init_CS[4] - Configures the initial checksum for Initial Messages. NOTE: This register must contain the logical INVERSE of the initial checksum. (See: "Checksum Calculation")

Error_Prefix[6] - Configures the Prefix (T-Bus bits: D7-D2) of the Error code response for Server error. (See: "Error Handling")

Ena_BIU - Enables the by releasing its reset signal (=1). Otherwise, the BIU will be held in reset. (=0). (See: "Disabled Operation")

Ena_SRU - Enables the SRU by releasing its reset signal (=1). Otherwise, the SRU will be held in reset. (=0). (See: "Disabled Operation")

STU_Freed_Sync[4] - Configures the settling time of the Switch Transmit Unit's (STU) handshake synchronizer which receives a "freed" signal from the Bus Interface Unit (BIU). This signal indicates that the BIU has acted on a previous "free" command from the SRU. (See: "Synchronization")

STU_Done_Sync[4] - Configures the settling time of the Switch Transmit Unit's (STU) handshake synchronizer which receives a "done" signal from the the Bus Interface Unit (BIU). This is used to indicate completion of a Fuction Request. (See: "Synchronization")

BIU_Free_Sync[4] - Configures the settling time of the Bus Interface Unit's (BIU) handshake synchronizer which receives a "free" signal from the Switch Receive Unit (SRU). This is used to issue a FREE-LOCK. (See: "Synchronization")

BIU_Xfer_Sync[4] - Configures the settling time of the Bus Interface Unit's (BIU) handshake synchronizer which receives a "xfer" from the Switch Receive Unit (SRU). This is used to initiate a Function Request. (See: "Synchronization")

Bit Definition - Server_ConfigA
Figure 56

Register: Server_ConfigB<31..0>

BIT/FIELD	FUNCTION (read/write)
<31..8>	not used
<7..6>	spare
<5>	Dis_Frame
<4>	Ena_SOC
<3>	Dis_Check_Err
<2..0>	SER_Slave_Num[3]

Register Definition - Server_ConfigB
Figure 57

5.2.1.8 Test Registers

The Server contains a read-only test register which should NEVER be accessed during normal operation. Figure 58 shows the structure of that register which is used mostly for observing internal states.

Register: Server_TestA<31..0>

BIT/FIELD	FUNCTION (read-only)
<31>	<unused>
<30>	SRU believes it is locked
<29>	SRU refusing new connections
<28>	Synchronized "Enable New SOC's"
<27>	SRU "Should be Checksum"
<26>	SRU Checksum OK signal
<25>	SRU Anticipation Signal
<24>	Checksum errors occured
<23..20>	<unused>
<19..16>	Running Version of Rx Checksum
<15..8>	Internal State of SRU FSM
<15>	SRU has seen Reverse come and go and has seen Frame go away. Transition to 9,10, or 13 will occur
<14>	SRU has seen first Reverse and is waiting for the end of the Reverse transmission
<13>	SRU is waiting for lock to be FREE-LOCKEed
<12>	SRU is waiting for first Reverse
<11>	SRU receiving Checksum byte
<10>	SRU receiving a command
<9>	SRU is idle
<8>	Bad SOC seen (low true)

Register Definition - Server_TestA
Figure 58

Figure 59 shows the bit definition of SOME of the bits in the Server_TestA register.

SRU believes it is locked - The BIU will issue a FREE-LOCKS request if Frame is negated for more than one Switch Interval.

SRU refusing new connections - Indicates that there is no active connection and that new connections will be refused (with Reject). The SRU IS currently and WILL be idle until re-enabled. (See: "Disabled Operation")

Synchronized Enable New SOC's - The synchronized version of Server_ConfigB.4. The programmer should check this bit before assuming that the SRU will Reject or accept new connections. (See: "Disabled Operation")

SRU "Should be Checksum" - Indicates that the Checksum should have arrived. This is used in conjunction with the "SRU Anticipation Signal" to determine if the SRU is properly anticipating the reception of the Checksum byte.

SRU "Checksum OK" - Indicates to the BIU that the TBus operation should, in fact, take place.

SRU Anticipation Signal - Indicates to the BIU that it should begin the TBus request. See SRU "Should be Checksum" above.

Checksum errors occured - Indicates that a checksum error did occur sometime in the past. This bit is negated whenever Server_ConfigB.4 is negated.

Bit Definition - Server_TestA
Figure 59

5.2.2 Switch Message Protocol

The Server fully generates and supports the Butterfly Switch protocol. That support is described below.

5.2.2.1 Upstream Message Components

Unlike the Requestor, the Server never has to create a message header with routing information because the return path to the upstream Requestor has already been established. The Server need only return a checksum with data and/or error code information. Figure 60 shows a typical upstream Server message as a response to a word-read function request. The significance of the "E" and "S" bits are described in: "Stolen and Error Messages."

```
7                                          0
|                                          |
D31   D30   D29   D28   D27   D26   D25   D24   (first sent)
D23   D22   D21   D20   D19   D18   D17   D16         |
D15   D14   D13   D12   D11   D10   D9    D8          |
D7    D6    D5    D4    D3    D2    D1    D0          |
                        .                             |
   <possible additional read words>                   |
                        .                             V
0     0     E     S     CS3   CS2   CS1   CS0   (last sent)


 ...where,

D31..D8   = data information from T-Bus: T_AD<31..8>
D7..D0    = error code (E=1), T_AD<7..0> (E=0)
E         = Error bit
S         = Stolen bit
CS3..CS0  = message checksum
```

Bit Definition - Upstream Message Body (read)
Figure 60

The upstream message body for a write is always of the same format whether the function request was multi-word or non-multi word. Figure 61 shows a typical upstream Server message as a response to a word write Function Request. The significance of the "E" and "S" bits are described in: "Stolen and Error Messages."

```
7                                   0
|                                   |
D7   D6   D5   D4   D3   D2   D1   D0   (first sent)
0    0    E    S    CS3  CS2  CS1  CS0  (last sent)
```

...where,

D7..D0 = error code (E=1), unknown (E=0)
E = Error bit
CS3..CS0 = message checksum

Bit Definition - Upstream Message Body (write)
Figure 61

5.2.2.2 Stolen and Error Messages

When the Upstream Read message has Stolen and/or Error bits asserted in the checksum, their presence modify the meaning of the message byte (or bytes) PRECEDING the checksum byte. In the case of an asserted (=1) Stolen bit, the Server is indicating that ONLY the previous four bytes are stolen. This is consistent with what can happen on the T-Bus side of the Server. There, a T-Bus Slave may happen to return a Stolen data word which is not necessarily the last word of the read opertion. The Server's BIU will continue to read any data "past" the Stolen word, but its STU will always END transmission of the Upstream Switch Message on the Stolen word - ignoring the rest. The consequence for the Upstream Requestor is that the "S" bit always modifies the LAST word received. The "S" bit has no meaning for Upstream write messages and is ignored.

When the Error bit is asserted (=1) during an Upstream Read message, the Server is indicating that the byte immediately PRECEDING the Checksum contains the Error Code and that any other bytes in the message are "garbage" data. The T-Bus protocol demands that all Slaves respond with "ERROR" during the FIRST word transfer and that an "ERROR" response ends the

T-Bus transfer. Therefore, an Upstream Read Message with E=1 will only contain one word of data. Assertion of the "E" bit has higher priority than assertion of the "S" bit, so they will never be asserted simultaneously in a given Upstream message.

Figure 62 shows a summary of the effect of the "E" and "S" bits on an Upstream Message.

E	S	previous byte is...
0	0	Data byte, previous word is NOT stolen (reads only)
0	1	Data byte, previous word is stolen (reads only)
1	0	Error Code (reads or writes)

Note: the value ES = 11 will never occur

Interpretation of Checksum E and S Bits
Figure 62

5.2.2.3 Upstream Message Types

The previous discussions about message formats can be brought together to produce an enumeration of the possible Upstream Message types. This summary is shown in Figure 63.

5.2.2.4 Checksum Calculation

Checksum support for the Server is described in the "Requestor/Operation/Checksum Calculation" section. The actual calculation performed by the Server is shown in Figure 64. Figure 64 shown the calculation for a single word read message. For read messages with more words, those bytes would be included in the same manner as the data bytes in the figure. For write messages, the data field would be missing entirely

TYPE	#WORDS	STOLEN or ERRORS	RETURN MSG FORMAT
write	any	none	XC
	"	any error	ZC
read	non-multi	none	DDDDC
	"	either on word1	DDDEC
	two-words	none	DDDDDDDDC
	"	either on word1	DDDEC
	"	stolen on word2	DDDDDDDDC
	three-words	none	DDDDDDDDDDDDC
	"	either on word1	DDDEC
	"	stolen on word2	DDDDDDDDC
	"	stolen on word3	DDDDDDDDDDDDC
	four-words	none	DDDDDDDDDDDDDDDDC
	"	either on word1	DDDEC
	"	stolen on word2	DDDDDDDDC
	"	stolen on word3	DDDDDDDDDDDDC
	"	stolen on word4	DDDDDDDDDDDDDDDDC

NOTE:

Frame is high for entire return message.

X = don't care
Z = always an Error Code
E = Error Code (Checksum bit 5 = 1)
  = Data Byte (Checksum bit 5 = 0)
C = Checksum Byte

Upstream Message Types
Figure 63

from the calculation and only the error byte would be included.

$$CS\langle 3\rangle = \mathrm{exor}(D31,D27,D23,D19,D15,D10,D7,D3,0)$$

$$CS\langle 2\rangle = \mathrm{exor}(D30,D26,D22,D18,D14,D9,D6,D2,0)$$

$$CS\langle 1\rangle = \mathrm{exor}(D29,D25,D21,D17,D13,D8,D5,D1,E)$$

$$CS\langle 0\rangle = \mathrm{exor}(D28,D24,D20,D16,D12,D7,D4,D0,S)$$

...where,

$CS\langle 3..0\rangle$ = message checksum

Equation - Message Checksum (single-word read, see text)
Figure 64

5.2.2.5 Rejects

A Reject is the assertion of Reverse for exactly one Switch Interval. Rejects are not, strictly speaking, messages; because the Switch data pins do not carry any known data. The Server produces a Reject (assertion of Reverse for only one Switch Interval) in either of three conditions: 1) An addressed downstream T-Bus slave is found to be locked during an Intitial Switch Message, 2) The Server has been configured to reject all Downstream messages, or 3) The Server's SRU state machine is busy while trying to return to its "idle" state.

During the Initial Switch message, the targeted Downstream device may, in fact, be locked to a device other than the Server. The Server issues a Reject to indicate this fact to the Upstream Requestor. Once the Server has sucessfully locked some device, it is still possible for a Locked Message to attempt an access to device other than one to which the Server is currently locked. In this situation however, the Server does NOT issue a Reject. Instead, it sends an error response to the upstream Requestor (see: "Error Reporting")

The Server can also be configured - via the

Requestor_ConfigA.Ena_SOC bit -to issue a reject on any new incoming message. This is a synchronized enable such that it can be asserted/negated at any time. The Server will continue to process any pending transactions but will prevent any new ones. Thus, the Server can be "gracefully" removed from the Switch interface.

Whenever the Server is in any state other than its "idle" state (locked or unlocked), it will refuse new attempts at a connection (Frame high preceded by Frame low for for at least two Switch Intervals) by issuing a Reject. There are many instances when a new connection attempt would indicate an Switch protocol violation, and thus a Reject issued by the Server would make little difference. However, there are some situations where the Server would correctly issue a Reject while it is off processing some event. For instance, a drop-lock would cause the Server to begin issuing a FREE-LOCK on the T-Bus. If new downstream Switch message attempted to access the Server before it finished the transaction, the Server would issue a Reject.

5.2.3 T-Bus Interface

The Server supports the standard T-Bus protocol with some small limitations. For one, the Server does NOT support unaligned transfers which fall accross word (32-bits) boundaries. The Server also expects to see an ERROR response as the FIRST response from a T-Bus Slave if that slave is goning to issue any ERROR's. If the Slave cannot issue an ERROR in the cycle immediatly following the T-Bus request (i.e., the first response cycle), it must assert T_NSPAUSE_xxx until it decides if the request is an error or not.

5.2.4 LCON Interface

The LCON is a the physical and logical link between the SIGA-Server and the "input" port of the Switch Gate Array (SGA). In other words, for the SIGA, the LCON interface is the logical Switch interface. The LCON provides the Server with: 1) level conversion to and from the ECL levels of the SGA and 2) reclocking of data, Frame, Reverse to and from the SGA.

Figure 65 shows the Server's LCON (Switch) Interface Pins.

PIN NAME	TYPE	FUNCTION
S_DATA<7..0>	bidirectional	Server-LCON data bus
S_FRAME	input	Frame input from Switch
S_REVERSE	output	Reverse output to Switch
S_NENA_BACK	input	LCON TTL driver enable

Server LCON (Switch) Interface Pins
Figure 65

5.2.4.1 Data Bus Enable Control

The Server controls the enables of both its own output drivers and the LCON's output drivers to the SIGA-LCON data interface - S_DATA<7..0>. It does so in a manner complementary to the Requestor's method (see "Requestor/Operation/LCON Interface/Data Bus Enable Control). The Server uses the same concept of "Talk" and "Listen" mode as the Requestor.

The Server is considered "quiescent" when it is not transmitting messages and not waiting for any replies. When quiescent, the Server is in Listen Mode. The Server tries to stay in Listen Mode whenever possible, making the transition to Talk only for the absolute minimum time necessary. This situation is the mirror image to the Requestor. It is always in Talk Mode when quiescent and tries to stay in Listen mode for as little time as possible.

When the Server receives the checksum of a downstream message, it transitions to Talk mode - via the "dead" state. It remains in Talk mode until the T-Bus transaction is complete and the upstream return message has been sent. Once the upstream checksum has been sent, the Server transitions immediately into Listen mode (no contention is possible - as with the Requstor).

5.3 TCS Control Unit (TCU)

The basic purpose of the TCS Unit (TCU) is to allow the Test and

Control System (TCS) Slave Processor access to the T-Bus interface - in esscence, to act as a protocol converter. Normally, this involves the TCU acting like a T-Bus Master - performing reads and writes. However, the TCU is flexible enough so that it can also generate or "spoof" responses for any T-Bus Master or Slave. A "spoofed" response essentially involves issuing a response on the T-Bus in the absence of a request. This can used, for instance, to free-up an observing T-Bus Master who's locked Slave has failed. In this case, the TCU can "make believe" that IT is the "failed" slave.

A secondary function of the TCU is to allow the TCS Slave Processor DIRECT access to the CSU Map, rather than forcing it to make an access via the T-Bus interface. This is useful for fault-tolerance and bootstrapping.

5.3.1 I/O Description

The TCU interface is composed of four pins on the SIGA. The pins and their basic functions are shown in Figure 65.

C_CLK - The data shift clock. Data is shifted into the SIGA on each rising edge of C_CLK. Data is shifted out of the SIGA on each falling edge of C_CLK.

C_IN - TCS data into the SIGA.

C_OUT - TCS data out of the SIGA. This is a tri-state signal which is driven when C_NEXECUTE is asserted (=0).

C_NEXECUTE - Asynchronously initiates execution of a command (=0) and enables C_OUT. In addition, neagting C_NEXECUTE (=1) resets the TCU interface.

TCU I/O Signal Description
Figure 66

5.3.2 Read/Write Operation

The TCU contains 16 addressable registers - each 8-bits wide. The TCS Slave can read any register by clocking-in the required address (4-bits), a Read/Write bit (=1), and assert C_NEXECUTE (=0). A read operation is illustrated in Figure 67.

```
                 inactive | addr in  |     data out

C_CLK        _____________H_H_H_H_H__H_H_H_H_H_H_H_H____
C_IN         ............a3a2a1a0pp.....................
C_NEXECUTE   HHHHHHHHHHHHHHHHHHHHHH_____________________
C_OUT        ------------------------d7d6d5d4d3d2d1d0...
```

...where,

a3..a0 = address of register to be read
d7..d0 = data from read register
pp = Read/Write bit (=1)

Timing - TCU Read Operation
Figure 67

Some additional details for Read operations - not apparent from Figure 67 - are now discussed.

1) C_IN data is clocked-in on the positive edge of C_CLK and C_OUT data is clocked-out on the negative edge of C_CLK.

2) Data can be clocked in or out at any desired rate, provided that the AC specifications of the C_CLK pin are not violated. The duty cycle of C_CLK is variable within the AC specifications. There is no MAXIMUM high (=1) or low (=0) time for C_CLK.

3) Reads are non-destructive and can be aborted at any time.

4) C_NEXECUTE is not synchronized with C_CLK and can be asserted at any time after the address and Read/Write bit has been clocked-in.

5) The C_OUT pin may be used to monitor, in real time, the value of a particular bit. This is done by reading the appropriate register, shifting-out the desired bit using C_CLK, and then holding C_CLK steady. C_CLK can be held in either state (1 or 0) as long as it does not make another positive transition.

6) Extra data bits preceding the negative transition of C_NEXECUTE, are ignored.

A write operation is performed by clocking-in four bits of data, 4-bits of address, a Read/Write bit (=0), and then asserting C_NEXECUTE (=0). This is illustrated in Figure 68.

```
                inactive |         command in          | exec

C_CLK       ____________H_H_H_H_H_H_H_H___H_H_H_H_H__________
C_IN        ..........d7d6d5d4d3d2d1d0..a3a2a1a0pp.........
C_NEXECUTE  HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH___HHHHH
C_OUT       -----------------------------------------d0..----
```

...where,

a3..a0 = address of register to be written to
d7..d0 = data to be written
pp = Read/Write bit (=0)

Timing - TCU Write Operation
Figure 68

Some additional details for Write operations - not apparent from Figure 68- are now discussed.

1) C_IN data is clocked-in on the positive edge of C_CLK and C_OUT data is clocked-out on the negative edge of C_CLK.

2) Data can be clocked in or out at any desired rate, provided that the AC specifications of the C_CLK pin are not violated. The duty cycle of C_CLK is variable within the AC specifications. There is no MAXIMUM high (=1) or low (=0) time for C_CLK.

3) Reads are non-destructive and can be aborted at any time. Reads of the TBUS_Response register can be aborted as well. However if this is done AFTER C_NEXECTUTE has been asserted, the TBUS operation may be aborted.

4) C_NEXECUTE is not synchronized with C_CLK and can be asserted at any time after the address and Read/Write bit has been clocked-in.

5) C_NEXECUTE need only be asserted for a short moment to begin execution of the command. The minimum low time is described in "AC Specifications."

6) Extra data bits preceding the negative transition of C_NEXECUTE, are ignored.

5.3.3 Register Map

The register map for the 16 TCU registers are shown in figure 69. Referring to Figure 69, registers 0 through 3 are special registers. For write operations, their contents are loaded, via the TCU interface, with the data to be written TO some T-Bus slave. For read operations, their contents are replaced with the data read FROM some T-Bus slave. Registers 4 through 7 are loaded ONLY by the TCU interface. The contents of these registers are placed on the T-Bus during the address phase of a T-Bus request.

The registers at address "C" and "D" are used to initialize CSU_Map<8..0>. Register "D" - bit "0", corresponds to CSU_Map<8>. Bits 7 through 1 of register "D" are unused. Figure 70 shows the definition of the TBUS Response and

a3..a0	DESCRIPTION
0	T_AD<7..0> (data)
1	T_AD<15..8> (data)
2	T_AD<23..16> (data)
3	T_AD<31..24> (data)
4	T_AD<7..0> (addr)
5	T_AD<15..8> (addr)
6	T_AD<23..16> (addr)
7	T_AD<31..24> (addr)
8	TBUS_Response
9	TBUS_Command
A	TBUS_Command_Modifier_0
B	TBUS_Command_Modifier_1
C	CSU Map<7..0>
D	CSU Map<8>
E	unused
F	unused

TCU Register Map
Figure 69

Command Registers. Referring to Figure 70, the TBUS_Response register is a read-only register which is valid after a T-Bus operation has been executed. The "Done" bit is monitored after a T-Bus command is initiated by the TCU. When asserted (=1), it indicates that the operation is complete. See the "T-Bus Operations" section for more detail. The "Drive_AD" bit indicates that the T_AD Bus was driven during a T-Bus access (=1). The remaining bits in the TBUS_Response register are the "responses" received from the T-Bus operation.

The TBUS_Command and BUS_Command_Modifier_1 registers contains the indicated fields to be placed on the T-Bus during the address phase of any operation. The TBUS_Command_Modifier_0

Register: TBUS_Response<7..0> (read only)

BIT/FIELD	FUNCTION (read only)
<7>	Done
<6>	Drive_AD
<5>	T_DRIVEN
<4>	M_PARITY
<3>	T_AD<32>
<2..0>	T_RR<2..0>

Register: TBUS_Command<7..0>

BIT/FIELD	FUNCTION
<7..6>	output T_AD<33..32> (addr)
<5..3>	output T_SIZE<2..0>
<2..0>	output T_RR<2..0>

Register: TBUS_Command_Modifier_0<7..0>

BIT/FIELD	FUNCTION
<7..0>	unused
<3>	Response
<2>	output T_AD<32> (data)
<1..0>	output T_PATH<1..0>

Register: TBUS_Command_Modifier_1<7..0>

BIT/FIELD	FUNCTION
<7>	output T_SYNC
<6..5>	output T_PRIORITY<1..0>
<4..3>	output T_LOCKOP<1..0>
<2..0>	output T_SOURCE<2..0>

Register Definitions - TBUS Response and Command Registers
Figure 70

register outputs the "T_PATH" field during the address phase of any operation and the T_AD<32> bit during the data phase of a write operation.

The "Response" field of the TBUS_Command_Modifier_0 register, has a special function. When asserted (=1), the TCU will place a "0" on the T_REQUEST and drive the T-Bus FOR A SINGLE CYCLE with the register settings intended for the address phase of a T-Bus cycle. This is used for "spoofing" a T-Bus response. When the "Response" field is a "1", the TCU makes a normal T-Bus Request with T_REQUEST asserted (=1).

5.3.4 Normal T-Bus Operations

The TCU can be used to read and write, one to four bytes. Multi-word transfers are not allowed. The TCU can also OPEN and FREE locks although this is not recommended because the TCS Slave interface is relatively slow.

A read or write operation is setup by loading the desired data into the registers. The operation is actually initiated by a read of the TBUS_Response register. Since the MSB of this register is the "Done" bit, C_CLK should be disabled just after C_NEXECUTE is asserted (=0). This allows asynchronous monitoring of the "Done" bit. Terminating the read by negating (=1) C_NEXECUTE will abort the T-Bus request.

The TCU will retry after becoming REFUSED but will ignore a REFUSED LOCKED. In other words, the TCU will not become an "observing master."

5.3.5 Special T-Bus Operations

The TCU can FREE-LOCKS for any T-Bus master by specifying the correct T_SOURCE field value and performing a write operation. The TCU can also spoof any one-cycle response of a Slave by asserting the "Response" bit in the TBUS_Command_Modifier_0 register. For instance it can issue a COMPLETED or ERROR for some Slave that is known to be faulty.

5.3.6 CSU Map Initialization

The CSU_Map is a 9-bit quantity which maps the SIGA CSU into a desired 8k page. This quantity is initialized by the TCU and is one of the first things that must be done to the SIGA upon power-up. If the CSU_Map is not initialized, it defaults to the setting of all 1's.

5.4 Configuration/Status Unit

The Configuration Status Unit (CSU) is the T-Bus Slave interface which allows any T-Bus master read and write access to the SIGA's configuration and status registers.

5.4.1 Normal Register Accesses

The CSU is limited in its support of the T-Bus protocol and is NOT optimized for minimum wait states (Slave pause cycles). The CSU will respond to a T-Bus query ONLY when T-Bus bits T_AD<24..16> match CSU_Map<8..0>. The CSU_Map is initialized by the TCU (See: TCS Control Unit/CSU Map Initialization).

In the cycle following a request to the CSU, the CSU will either respond with an ERROR or go on to complete the requested function. Figure 71 shows the TCU responding with an ERROR. Note from Figure 71, that T_NSPAUSE_SIGA is asserted for only one cycle. The ERROR response is triggered by exactly two conditions: 1) T_SIZE<2> = 1 or 2) T_LOCKOP<1> = 1. This means that the CSU will not support multi-word writes or locking. A normal read and write operation are shown in Figure 72. Note from Figure 72 that T_AD<32> is always a "0" on a read and a "don't care" on a write. In addition, during write operations, data is setup to the configuration latches during cycle #1, written to them during cycle #2, and held at the configuration latches during cycle #3.

5.4.2 Synchronized Accesses

Certain accesses to the CSU must be synchronized to the One Microsecond Pulse (OMSP). These include: 1) read/writes of the Real Time Clock, and 2) writes to the TONI_A or TONI_B

```
T-Bus cycle #  |  0  |  1   |  2  |
T-Bus cycle    | req | resp | end |

T_NSPAUSE_SIGA HHHHHHHHHH________HHHHH
T_RR<3..0>               xxxxxxxxeeeee
```

...where,

x..x = invalid response
e..e = ERROR response

Timing - CSU ERROR Access
Figure 71

registers. This mechanism is described in: "Requestor/Operation/RTC and Related Functions". Essentially, all this means to the CSU timing diagram in Figure 72, is that cycle #2 is repeated until the synchronization pulse is received from the RTC or TONI_A/B controller.

5.4.3 Interleaver Loader

The CSU provides support for loading and reading the Interleaver Modulus Ram through the use of two special registers: Interleave_Address and Interleave_Data; and an external pin to the SIGA: I_NACCESS. Reads and writes to both the Interleave_Address and Interleave_Data registers are different than accesses to other configuration/status registers in the SIGA. The structure of the Interleaver_Address register is shown in Fgure 73. The structure of the Interleaver_Data register is shown in figure 74. As seen in Figure 74, read/write access to the I_D register does not involve any data transfer within the SIGA.

```
T-Bus cycle #           |  0   |   1   |   2   |   3   |
T-Bus cycle             | req  | resp  | resp  |  end  |

T_NSPAUSE_SIGA          HHHHHHHHHH_______________HHHH
T_RR<3..0>              ??????????xxxxxxxxxxxxxxcccc

T_AD<32>     (read)     ???????????XXXXXXXXXXXX_____
T_AD<31..0>  (read)     ???????????XXXXXXXXXXXXRRRR

T_AD<32>     (write)    XXXXXXXXXXXXXXXXXXXXXXXXXXX
T_AD<31..0>  (write)    ?????????WWWWWWWWWWWWWWWWWWWW
```

...where,

x..x = invalid response
c..c = COMPLETED response
X..X = invalid data
W..W = valid write data

Timing - Normal CSU Read/Write
Figure 72

5.4.3.1 Address Register Access

When a T-Bus master reads the Address_Register, the CSU immediately responds with a Slave Pause cycle by asserting (=0) the T_NSPAUSE_SIGA pin, as it does with all other accesses. However, in the following cycle, the CSU also asserts the I_NACCESS pin and places the contents of the Interleave_Address register on the T-Bus. The CSU then waits for exactly seven (7) T-Bus cycles in this state. The mapping of the I_A register to the T-Bus during this "wait" state is shown in Figure 75, part (a). In the cycle following the wait period, the CSU then negates (=1) both T_NSPAUSE_SIGA and I_NACCESS, and maps the I_A to the T-Bus as shown in Figure 75, part (b). The timing for writes to the I_A register is exactly the same as for reads. The actual timing for

```
Register: Interleave_Address

31..............................0
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

...where,

A..A = interleaver address
```

Register Definition - Interleave_Address
Figure 73

```
Register: Interleave_Data

31..............................0
--------------------------------  (read)
--------------------------------  (write)
```

Register Definition - Interleave_Data
Figure 74

Interleave_Address register read/write access is shown in Figure 76.

5.4.3.2 Data Register Access

The Interleave_Data access is EXACTLY the same as the Interleave_Address access EXCEPT for two key features: (1) during writes, no data is actually stored in the SIGA, and (2) during reads, the SIGA does NOT drive the T_AD<33..0> field. During this time, logic external to the SIGA will manipulate the Modulus Ram, and the SIGA is basically being used as an address decoder and T-Bus control signal driver. The actual timing for

T_AD<33> = Interleave_Address<1>
T_AD<32> = Interleave_Address<0>
T_AD<31..0> = Interleaver_Address<31..0>

(a) wait (cycle 3 - 9)

T_AD<33> = 0
T_AD<32> = 0
T_AD<31..0> = Interleaver_Address<31..0>

(b) end (cycle 10)

Interleave_Address Register to T-Bus Mapping
Figure 75

Interleave_Data register read/write access is shown in Figure 77. Note from Figure 77 that the CSU temporarily drives the T-Bus during cycle #1. The data is unknown.

5.4.4 Debug Support

The CSU supports "freezing" a CSU read or write for debugging purposes. This is accomplished by initiating a normal T-Bus access (see the "Timing - Normal CSU Read/Write" figure) and asserting (=0) and holding the pin, M_NDEBUG, during cycle #1 and #2. This will cause the CSU to repeat cycle #2 indefinitely until M_NDEBUG is negated (=1). When this occurs, the CSU will continue with cycle #3 as normal.

For read cycles this means that T_AD<31..0> will have the real-time state of any register being read. By reading a test register, for example, the state machine of the STU an be observed while it sends a message.

For write cycles, the use is somewhat limited. It simply means that T_AD<31..0> can be manipulated in real-time from the master (or logic analyzer). Since during cycle #2 the

```
T-Bus cycle # |  0   |  1   |  2   |...|  10  |  11  |
T-Bus cycle   | req  | resp | wait |...|  end |  ?   |

T_NSPAUSE_SIGA HHHHHHHHH______________..._____HHHHHHHHHHH
T_RR<3..0>                            ...  cccccc

I_NACCESS    (read) HHHHHHHHHHHH______..._____HHHHHHHHHHH
T_AD<33..0> (read)     ------???aaaaa...aaabbbb

I_NACCESS   (write) HHHHHHHHHHHH______..._____HHHHHHHHHHH
T_AD<33..0> (write)     ddddddddddd...dddd???
```

...where,

c..c = COMPLETED response
a..a = "wait" type read of I_A (bit swapping)
b..b = "end" type read of I_A (bit masking)
d..d = data written TO the I_A register

Timing - CSU Interleave_Address Register Read/Write Access
Figure 76

configuration latches are transparent, so that any external manipulation will be seen internally in real-time.

5.4.5 Restriction Summary

The following restictions apply to CSU operation:

6 Programming Model

This section provides a memory map of the previously defined SIGA registers, as well as a compilation of all SIGA Error Codes.

```
T-Bus cycle # |  0   |  1   |  2   |...|  10  |  11  |
T-Bus cycle   | req  | resp | wait |...| end  |  ?   |

T_NSPAUSE_SIGA HHHHHHHHH______________..._____HHHHHHHHHHH
T_RR<3..0>                            ...  ccccc

I_NACCESS (read) HHHHHHHHHHHHHH______..._____HHHHHHHHHHH
T_AD<33..0> (read) xxxxxaaaaaaa----bb...bbbbbbb

I_NACCESS  (write) HHHHHHHHHHHH______..._____HHHHHHHHHHH
T_AD<33..0> (write)    ???????????xxx...xxxxxxxxx
```

...where,

c..c = COMPLETED response
a..a = unknown data driven by CSU (only for one cycle)
b..b = data from/to Interleaver (not driven by SIGA)

Timing - CSU Interleave_Data Read/Write Access
Figure 77

6.1 Memory Map

Figure 78 shows the memory map of the various registers. Note from Figure 78 that the "M" field is programmable via the CNU_Config.CSU_Map bits.

6.2 Error Code Summary

Figure 79 presents an Error Code summary for the SIGA. Figure 80 summarizes the Error Code definitions.

1) The CSU will flag as an ERROR any multi-word access or an OPEN or MAINTAIN. Therefore, the CSU does not support these operations. However, byte masking on writes IS supported.

2) The CSU will NOT check for unaligned transfers. It is illegal to request an operation with an unaligned address.

3) Synchronized Accesses rely on the presence of R_CLK to complete. If R_CLK is non-exisitent, the CSU will pause the T-Bus Master indefinitely. The only way to release the pause would be to assert the M_NRESET pin.

4) The Stolen bit (T_AD<32>) is not supported on either reads or writes.

7 Special Topics

This section describes some of the special topics relating to SIGA operation.

7.1 Initialization States

The external Reset signal is resynchronized by the SIGA for use by all synchronous logic clocked by all three major clocks (R_CLK, S_CLK and T_CLK). When Reset is applied and then released, all internal storage logic that needs to be initialized, will be so initialized. The SIGA will now be in its first initialization state, known as the Quiescent State.

In this state, the SIGA Switch and T-Bus interfaces are partially disabled. The Server's Switch interface responds to any assertions of downstream Frame with Rejects. The Requestor's Switch interface ignores any assertions of the upstream Reverse. The Server's T-Bus interface makes no T-Bus requests and the Requestor's T-Bus interface responds to any remote function requests with a REFUSED. The Configuration/Status Unit and the TCU, however, are

```
T_AD<24..0>                 REGISTER
===================         ===================================
  15  12          2 10
   |   |          | ||
M 000 XXXXXXXX000 bb        TONIA_Config
              001 bb        Time_Of_Next_InterruptA

M 001 XXXXXXXX000 bb        TONIB_Config
              001 bb        Time_Of_Next_InterruptB

M 100 0XXXXXXX000 bb        Protocol_Timer_Config | Message_Class
              001 bb        Transmit_Time_Config
              010 bb        Priority_Time_Config
              011 bb        Requestor_ConfigA
              100 bb        Requestor_ConfigB
              101 bb        Requestor_TestA
              110 bb        Real_Time_Clock (hi/lo)
              111 bb        <reserved>

      1XXXXXXX000 bb        Server_ConfigA
              001 bb        Server_ConfigB
              010 bb        Server_TestA

M 101 0XXXXXXXXXX xx        Interleave_Address_Reg
      1XXXXXXXXXX xx        Interleave_Data_Reg

        ...where,

          M  = (T_AD<24..16> = CNU_Config.CSU_Map<8..0>)

          bb =  00  byte 0 <31..24>
                01  byte 1 <23..16>
                10  byte 2 <15..8>
                11  byte 3 <7..0>

          xx =  no byte addressing capability
```

SIGA Memory Map
Figure 78

Requestor/CSU Error Codes:

```
7      0
|      |
PPPPdcba
```

d	c	b	a	Requestor/ CSU Error
0	0	0	0	Maintain_Absent-(2a)
0	0	0	1	Maintain_Present-(2b)
0	0	1	0	Stolen_Verify-(1)
0	0	1	1	Lock_Address-(2)
0	1	0	0	Wait_TO-(3a)
0	1	0	1	Idle_TO-(3b)
0	1	1	0	Rej_Abort(4)
0	1	1	1	Rej_TO-(5)
1	0	0	0	Reverse-(6)
1	0	0	1	Check-(7)
1	0	1	0	Misc. CSU Error

...where,

P..P = Requestor_ConfigA.Error_Prefix<3..0>
Priority is from highest (1) to lowest (8).
Within a given priority, errors are mutually exclusive (i.e.,4a,b...).

Server Error Codes:

```
7      0
|      |
PPPPPPba
```

b	a	Server Error
0	0	Downstream_Refused
0	1	Downstream_Write
1	0	Downstream_Late
1	1	Downstream_OTL

...where,

P..P = Server_ConfigA.Error_Prefix<5..0>

Error Code Summary
Figure 79

operational. Normally, in the Quiescent state, the TCU will intialize the CSU's mapping logic via the CNU_Config.CSU_Map<8..0> register. Once the Control Net initializes the CSU_Map, any T-Bus master can then initialize the SIGA registers via the CSU.

Once this is accomplished the SIGA is in the Operational State. The Operational State is the normal operational mode of the SIGA.

7.2 Synchronization

Because of the use of multiple clocks, the SIGA design inherently requires the use of synchronizers to implement handshaking across clock boundaries. Some of these synchronizers are in non-critical paths and are thus implemented in the most cost-effective manner. In particular, these synchronizers are of the "large uncertainty, fixed-delay" variety. This means that there delay is not programmable and that "input-to-output" delay is not constant over changes in input. These are used in areas such as: 1) Between the external reset pin, M_NRESET, and the internal reset destinations, 2) Between the TCU negation of C_NEXECUTE and the T_Bus access. These synchonizers are designed to provide a MINIMUM of 100 ns settling time (T_CLK <= 22 MHz, R_CLK,S_CLK <= 45 Mhz).

The other variety of synchonizers - used in critical path applications - are the "variable delay, zero uncertainty" synchronizers. These are used beween the T-Bus and Switch interfaces along the Function request/response paths. These are the synchronizers which have 4-bits of configuration to control the settling time. Figure 81 shows the various settings for ALL variable-delay synchronizers. Figure 81 should be used in combination with the clock period of the

Maintain_Absent - An NORMAL was issued to the Requestor during its idle state and it was locked.

Maintain_Present - A MAINTAIN was issued to the Requestor during its idle state and it was NOT locked.

Lock_Address - A Function Request was made to a locked Requestor during its idle state with a node address was different than that which opened the locked sequence.

Wait_TO - The Switch Transmit Connection Timer overflowed while the Requestor was waiting for a Function Response.

Idle_TO - The Switch Transmit Connection Timer overflowed while the Requestor was in its idle state.

Rej_Abort - The Switch Transmit Reject Timer was forced into overflow by the the REJ_ABORT input pin.

Rej_TO - The Switch Transmit Reject Timer overflowed while the Requestor was attempting to open a connection.

Reverse - The Requestor detected an incorrect polarity of the Reverse signal during a Function Response.

Check - The Requestor detected an incorrect Checksum during a Function Response.

CSU Error - An error was made accessing the CSU. It could be one or both of the of the following: 1) An OPEN lock was requested or 2) A Multi-word transfer was requested.

Downstream_Write - A downstream write error was detected while the downstream Server was sourcing data.

Downstream_OTL - A downstream T-Bus slave did not respond to the Server's request.

Downstream_Late - A downstream T-Bus slave responded with a LATE ERROR.

Downstream_Refused - A downstream T-Bus slave responded with REFUSED-LOCKED when the Server thought itself locked.

Error Code Definition Summary
Figure 80

3210	# CLOCK DELAYS	TRANSFER EDGE
0000	1	Positive
0001	1	Negative
0010	2	Positive
0011	2	Negative
0100	3	Positive
0101	3	Negative
0110	4	Positive
0111	4	Negative
1000	5	Positive
1001	5	Negative
1010	ILLEGAL	-
1011	ILLEGAL	-
1100	ILLEGAL	-
1101	ILLEGAL	-
1110	ILLEGAL	-
1111	ILLEGAL	-

Variable-Delay Synchronizer Settings
Figure 81

logic RECEIVING the synchronizer data to determine the actual settling time. For instance, if a 100 ns settling time on the positive edge is desired for the STU Synchronizer, the register: Requestor_ConfigA.STU_Sync<3..0>, should be set to a "0110." This is because assuming R_CLK = 40 MHz (25 ns period), the synchronizer will require four clock periods - at 25 ns a piece - to obtain the total of 100 ns.

On the other hand, the BIU Synchronizer control, set by Requestor_ConfigA.BIU_Sync<3..0>, would need a setting of "0010" to obtain the same settling time. Here, of course, the clock period is twice as long as the STU Synchronizer so the number of synchronizer clock delays is half.

NOTE: Currently, it is recommended that only the POSITIVE transfer edge be used for any setting.

NOTE: It has been determined that a settling time of 100 ns is a reasonable goal for the variable-delay synchronizers.

8 Pin Description and Pinout

The next page begins a pin description of the SIGA:

PIN NAME	TYPE	DESCRIPTION
C_CLK	IN	TCU input clock
C_IN	IN	TCU data input
C_NEXECUTE	IN	TCU execute handshake input
C_OUT	OUT	TCU data output
F_AD<24..16>	IN	T-Bus input for T_AD<24..16>
F_PATH<1..0>	IN	T-Bus input for T_PATH<1..0>
F_REQUEST	IN	T-Bus input for T_REQUEST
F_RR<2..0>	IN	T-Bus input for T_RR<2..0>
F_SIZE_2	IN	T-Bus input for T_SIZE_2
F_SOURCE<2..0>	IN	T-Bus input for T_SOURCE<2..0>
I_INTERLEAVED	IN	=0. do NOT use I_MOD<8..0> for route address =1: use I_MOD<8..0> for route address
I_MOD<8..0>	IN	Interleaver data input
I_NACCESS	OUT	=0: CSU Interleaver loader is active =1: CSU Interleaver loader is NOT active
M_NDEBUG	IN	=0: Debug mode during CSU access (TEST ONLY) =1: Do NOT enter debug mode (NORMAL MODE)
M_NFLOAT	IN	=0: Tri-state all ouputs (TEST ONLY) =1: Normal output operation (NORMAL MODE)
M_NRESET	IN	=0: Hardware reset to SIGA =1: Normal operational mode
M_NSELECT	IN	=0: Select CSU, attach to T_PATH<1/0> =1: Do NOT select CSU
M_PARA	OUT	Parametric nand tree output (TEST ONLY)
M_PARITY	IN	=0: No parity error during T-Bus respnse =1: Parity error during T-Bus response
M_REJ_ABORT	IN	=0: Do NOT abort Switch retries =1: Abort Switch retries
M_SIXTY_FIVE	IN	=0: 65 ms pulse NOT active =1: 65 ms pulse active (one R_CLK period)
M_TONIA_INT	OUT	=0: TONIA interrupt is active =1: TONIA interrupt is NOT active
M_TONIB_INT	OUT	=0: TONIB interrupt is active =1: TONIB interrupt is NOT active
R_CLK	IN	Requestor clock input
R_DATA<7..0>	BID	Requestor Switch data interface
R_FRAME	OUT	Requestor Switch Frame output
R_NENA_BACK	OUT	=0: Enable LCON to drive R_DATA<7..0> =1: Disable LCON from driving R_DATA<7..0>
R_REVERSE	IN	Requestor Switch Reverse Input
S_CLK	IN	Server clock input
S_DATA<7..0>	BID	Server Switch data interface

S_FRAME	IN	Server Switch Frame input
S_NENA_BACK	OUT	=0: Disable LCON from driving S_DATA<7..0> =1: Enable LCON to drive S_DATA<7..0>
S_REVERSE	OUT	Server Switch Reverse Input
T_AD<33..25>	BID	T-Bus input/output for T_AD<33..25>
T_AD<24..16>	OUT	T-Bus output for T_AD<24..16>
T_AD<15..0>	BID	T-Bus input/output for T_AD<15..0>
T_CLK	IN	T-Bus input clock
T_DRIVEN	OUT	T-Bus output for T_DRIVEN
T_ENA_HOLD	IN	=0: Disable T-Bus input latches =1: Enable T-Bus input latches
T_ENA_TDAT.2	OUT	=0: Enable T_AD<33..0> drivers =1: Disable T_AD<33..0> drivers
T_ENA_TDAT<1..0>	OUT	=0: Disable T_AD<33..0> drivers =1: Enable T_AD<33..0> drivers
T_ENA_TRANS.1	OUT	=0: Enable transaction T-Bus field =1: Disable transaction T-Bus field
T_ENA_TRANS.0	OUT	=0: Disable transaction T-Bus field =1: Enable transaction T-Bus field
T_LOCKOP<1..0>	BID	T-Bus input/output for T_LOCKOP<1..0>
T_MPAUSE	OUT	T-Bus output for T_MPAUSE
T_NBGRANT_SIGM	IN	=0: SIGA Master granted next T-Bus =1: SIGA Master NOT granted next T-Bus
T_NBGRANT_SIGS	IN	=0: SIGA Slave granted next T-Bus =1: SIGA Slave NOT granted next T-Bus
T_NBREQ_SIGM	OUT	=0: SIGA Master is requesting T-Bus =1: SIGA Master is NOT requesting T-Bus
T_NBREQ_SIGS	OUT	=0: SIGA Slave is requesting T-Bus =1: SIGA Slave is NOT requesting T-Bus
T_NDRIVEN_SIGA	OUT	=0: SIGA is driving T-Bus next cycle =1: SIGA is NOT driving T-Bus next cycle
T_NSPAUSE_SIGA	OUT	=0: SIGA is pausing T-Bus next cycle =1: SIGA is NOT pausing T-Bus next cycle
T_PATH<1..0>	OUT	T-Bus output for T_PATH<1..0>
T_PRIORITY<1..0>	BID	T-Bus input/output for T_PRIORITY<1..0>
T_REQUEST	OUT	T-Bus output for T_REQUEST
T_RR<2..0>	OUT	T-Bus output for T_RR<2..0>
T_SIZE.2	OUT	T-Bus output for T_SIZE.2
T_SIZE<1..0>	BID	T-Bus input/output for T_SIZE<1..0>
T_SOURCE<2..0>	OUT	T-Bus output for T_SOURCE<2..0>
T_SPAUSE	OUT	T-Bus output for T_SPAUSE
T_SYNC	BID	T-Bus input/output for T_SYNC

The following page shows the SIGA pinout sorted by pin function.

SIGA PINOUT SORTED BY PIN FUNCTION

R15	C_CLK	R06	R_DATA.6	B12	T_DRIVEN
T14	C_IN	P06	R_DATA.7	C12	T_ENA_HOLD
R14	C_NEXECUTE	R05	R_FRAME	C03	T_ENA_TDAT.0
P13	C_OUT	T05	R_NENA_BACK	B03	T_ENA_TDAT.1
B09	F_AD.16	T04	R_REVERSE	A03	T_ENA_TDAT.2
C09	F_AD.17	T13	S_CLK	C14	T_ENA_TRANS.0
A10	F_AD.18	T12	S_DATA.0	C15	T_ENA_TRANS.1
B10	F_AD.19	P11	S_DATA.1	D01	T_LOCKOP.0
C10	F_AD.20	R11	S_DATA.2	D02	T_LOCKOP.1
A11	F_AD.21	T11	S_DATA.3	E01	T_MPAUSE
B11	F_AD.22	P10	S_DATA.4	A06	T_NBGRANT_SIGM
C11	F_AD.23	R10	S_DATA.5	C07	T_NBGRANT_SIGS
A12	F_AD.24	T10	S_DATA.6	C05	T_NBREQ_SIGM
A05	F_CLK	P09	S_DATA.7	B05	T_NBREQ_SIGS
A07	F_PATH.0	R13	S_FRAME	C06	T_NDRIVEN_SIGA
C08	F_PATH.1	R12	S_NENA_BACK	B06	T_NSPAUSE_SIGA
B14	F_REQUEST	P12	S_REVERSE	C13	T_PATH.0
C04	F_RR.0	P02	T_AD.0	A14	T_PATH.1
B04	F_RR.1	N03	T_AD.1	E02	T_PRIORITY.0
A04	F_RR.2	F14	T_AD.10	E03	T_PRIORITY.1
F03	F_SIZE_2	F15	T_AD.11	A13	T_REQUEST
G03	F_SOURCE.0	F16	T_AD.12	D14	T_RR.0
F01	F_SOURCE.1	G14	T_AD.13	D15	T_RR.1
F02	F_SOURCE.2	G15	T_AD.14	D16	T_RR.2
B02	I_INTERLEAVED	G16	T_AD.15	E14	T_SIZE.0
M02	I_MOD.0	H14	T_AD.16	E15	T_SIZE.1
M01	I_MOD.1	H15	T_AD.17	E16	T_SIZE.2
L03	I_MOD.2	J15	T_AD.18	D03	T_SOURCE.0
L02	I_MOD.3	J14	T_AD.19	C01	T_SOURCE.1
L01	I_MOD.4	P01	T_AD.2	C02	T_SOURCE.2
K03	I_MOD.5	K16	T_AD.20	B15	T_SPAUSE
K02	I_MOD.6	K15	T_AD.21	B13	T_SYNC
K01	I_MOD.7	K14	T_AD.22	A09	VDD
J03	I_MOD.8	L16	T_AD.23	A15	VDD
P03	I_NACCESS	L15	T_AD.24	B01	VDD
R02	M_NDEBUG	L14	T_AD.25	B16	VDD
P14	M_NFLOAT	M16	T_AD.26	J01	VDD
T15	M_NRESET	M15	T_AD.27	J16	VDD
B07	M_NSELECT	M14	T_AD.28	T01	VDD
R03	M_PARA	N16	T_AD.29	T08	VDD
C16	M_PARITY	N02	T_AD.3	T16	VDD

R04	M_REJ_ABORT	N15	T_AD.30	A02	VSS
J02	M_SIXTY_FIVE	N14	T_AD.31	A08	VSS
P04	M_TONIA_INT	P16	T_AD.32	A16	VSS
T03	M_TONIB_INT	P15	T_AD.33	H01	VSS
P05	R_CLK	N01	T_AD.4	H16	VSS
R09	R_DATA.0	M03	T_AD.5	R01	VSS
R08	R_DATA.1	H02	T_AD.6	R16	VSS
P08	R_DATA.2	H03	T_AD.7	T02	VSS
R07	R_DATA.3	G01	T_AD.8	T07	VSS
P07	R_DATA.4	G02	T_AD.9	T09	VSS
T06	R_DATA.5	B08	T_CLK		

The following page shows the SIGA pinout sorted by pin number

SIGA PINOUT SORTED BY PIN NUMBER

A02	VSS	D16	T_RR.2	N14	T_AD.31
A03	T_ENA_TDAT.2	E01	T_MPAUSE	N15	T_AD.30
A04	F_RR.2	E02	T_PRIORITY.0	N16	T_AD.29
A05	F_CLK	E03	T_PRIORITY.1	P01	T_AD.2
A06	T_NBGRANT_SIGM	E14	T_SIZE.0	P02	T_AD.0
A07	F_PATH.0	E15	T_SIZE.1	P03	I_NACCESS
A08	VSS	E16	T_SIZE.2	P04	M_TONIA_INT
A09	VDD	F01	F_SOURCE.1	P05	R_CLK
A10	F_AD.18	F02	F_SOURCE.2	P06	R_DATA.7
A11	F_AD.21	F03	F_SIZE_2	P07	R_DATA.4
A12	F_AD.24	F14	T_AD.10	P08	R_DATA.2
A13	T_REQUEST	F15	T_AD.11	P09	S_DATA.7
A14	T_PATH.1	F16	T_AD.12	P10	S_DATA.4
A15	VDD	G01	T_AD.8	P11	S_DATA.1
A16	VSS	G02	T_AD.9	P12	S_REVERSE
B01	VDD	G03	F_SOURCE.0	P13	C_OUT
B02	I_INTERLEAVED	G14	T_AD.13	P14	M_NFLOAT
B03	T_ENA_TDAT.1	G15	T_AD.14	P15	T_AD.33
B04	F_RR.1	G16	T_AD.15	P16	T_AD.32
B05	T_NBREQ_SIGS	H01	VSS	R01	VSS
B06	T_NSPAUSE_SIGA	H02	T_AD.6	R02	M_NDEBUG
B07	M_NSELECT	H03	T_AD.7	R03	M_PARA
B08	T_CLK	H14	T_AD.16	R04	M_REJ_ABORT
B09	F_AD.16	H15	T_AD.17	R05	R_FRAME
B10	F_AD.19	H16	VSS	R06	R_DATA.6
B11	F_AD.22	J01	VDD	R07	R_DATA.3
B12	T_DRIVEN	J02	M_SIXTY_FIVE	R08	R_DATA.1
B13	T_SYNC	J03	I_MOD.8	R09	R_DATA.0
B14	F_REQUEST	J14	T_AD.19	R10	S_DATA.5
B15	T_SPAUSE	J15	T_AD.18	R11	S_DATA.2
B16	VDD	J16	VDD	R12	S_NENA_BACK
C01	T_SOURCE.1	K01	I_MOD.7	R13	S_FRAME
C02	T_SOURCE.2	K02	I_MOD.6	R14	C_NEXECUTE
C03	T_ENA_TDAT.0	K03	I_MOD.5	R15	C_CLK
C04	F_RR.0	K14	T_AD.22	R16	VSS
C05	T_NBREQ_SIGM	K15	T_AD.21	T01	VDD
C06	T_NDRIVEN_SIGA	K16	T_AD.20	T02	VSS
C07	T_NBGRANT_SIGS	L01	I_MOD.4	T03	M_TONIB_INT
C08	F_PATH.1	L02	I_MOD.3	T04	R_REVERSE
C09	F_AD.17	L03	I_MOD.2	T05	R_NENA_BACK
C10	F_AD.20	L14	T_AD.25	T06	R_DATA.5

C11	F_AD.23	L15	T_AD.24	T07	VSS
C12	T_ENA_HOLD	L16	T_AD.23	T08	VDD
C13	T_PATH.0	M01	I_MOD.1	T09	VSS
C14	T_ENA_TRANS.0	M02	I_MOD.0	T10	S_DATA.6
C15	T_ENA_TRANS.1	M03	T_AD.5	T11	S_DATA.3
C16	M_PARITY	M14	T_AD.28	T12	S_DATA.0
D01	T_LOCKOP.0	M15	T_AD.27	T13	S_CLK
D02	T_LOCKOP.1	M16	T_AD.26	T14	C_IN
D03	T_SOURCE.0	N01	T_AD.4	T15	M_NRESET
D14	T_RR.0	N02	T_AD.3	T16	VDD
D15	T_RR.1	N03	T_AD.1		

9 A.C./D.C. Parameters

All SIGA input and bidirectional pins have a light pullup resistor, a diode protection network (max = 2000V) and latch-up (max = 200 ma). All inputs and output have standard TTL VIL/VIH and VOL/VOH characteristics. All outputs and bidirectional pins have 4ma drive capability - except T_ENA_TDAT<2..0> and T_ENA_TRANS<1..0>, which have 8 ma drive capability. The SIGA will dissipate less than 3 watts.

The following page shows the A.C. timing parameters.

Note: for the B2/VME, the following A.C. parameters override the normal ones:

PIN/CLASS	Tsu	Thld	Tpd (min/max)	LOAD
T_NDRIVEN_SIGA	-	-	2.0/11.0	20.0
F_SOURCE<2..0>	21.0	0.0	-	-

SIGA A.C. CHARACTERISTICS

PIN/CLASS	Tsu	Thld	Tpd (min/max)	LOAD
TBUS:				
T_DRIVEN	25.0	0.0	-	-
T_MPAUSE	25.0	0.0	-	-
T_SPAUSE	25.0	0.0	-	-
T_NBGRANT_SIGM	25.0	0.0	-	-
T_NBGRANT_SIGS	25.0	0.0	-	-
T_REQUEST	(a)	(a)	2.0/18.0	30.0
T_RR<2..0>	(a)	(a)	2.0/18.0	30.0
T_PATH<1..0>	(a)	(a)	2.0/18.0	30.0
T_SOURCE<2..0>	(a)	(a)	2.0/18.0	30.0
T_SIZE.2	(a)	(a)	2.0/18.0	30.0
T_SIZE<1..0>	20.0	0.0	2.0/18.0	30.0
T_SYNC	20.0	0.0	2.0/18.0	30.0
T_LOCKOP<1..0>	20.0	0.0	2.0/18.0	30.0
T_PRIORITY<1..0>	20.0	0.0	2.0/18.0	30.0
T_AD<33..0>	20.0	0.0	2.0/18.0	30.0
T_NBREQ_SIGM	-	-	2.0/13.0	20.0
T_NBREQ_SIGS	-	-	2.0/13.0	20.0
T_NDRIVEN_SIGA	-	-	2.0/13.0	20.0
T_NSPAUSE_SIGA	-	-	2.0/13.0	20.0
T_ENA_TDAT<2..0>	-	-	2.0/16.0	30.0
T_ENA_TRANS<1..0>	-	-	2.0/16.0	30.0
FAST:				
F_REQUEST	25.0	0.0	-	-
F_RR<2..0>	24.0	0.0	-	-
F_SOURCE<2..0>	25.0	0.0	-	-
F_PATH<1..0>	25.0	0.0	-	-
F_SIZE_2	25.0	0.0	-	-
F_AD<24..16>	25.0	0.0	-	-
SWITCH - REQ:				

R_DATA<7..0>	7.8	14.2	2.0/14.0	20.0
R_REVERSE	7.8	14.2	–	–
R_FRAME	–	–	2.0/14.0	20.0
R_NENA_BACK	–	–	2.0/14.0	20.0
SWITCH - SER:				
S_DATA<7..0>	7.8	14.2	2.0/14.0	20.0
S_FRAME	7.8	14.2	–	–
S_REVERSE	–	–	2.0/14.0	20.0
S_NENA_BACK	–	–	2.0/14.0	20.0
TCS:				
C_IN	50.0	50.0	–	–
C_OUT	–	–	2.0/50.0	20.0
C_NEXECUTE	50.0	50.0	–	–
INTERLEAVER:				
I_MOD<8..0>	17.0	0.0	–	–
I_INTERLEAVED	24.0	0.0	–	–
I_NACCESS	–	–	2.0/30.0	20.0
MISCELLANEOUS:				
M_TONIA_INT	–	–	2.0/30.0	20.0
M_TONIB_INT	–	–	2.0/30.0	20.0
M_PARITY	21.0	0.0	–	–
M_NSELECT	25.0	0.0	–	–
M_NDEBUG	25.0	24.0	–	–
M_SIXTY_FIVE	7.9	14.9	–	–
M_NRESET	(b)	(b)	–	–
M_REJ_ABORT	(b)	(b)	–	–

NOTES:

specific:

(a) No internal connection to SIGA - timing is unimportant

(b) Synchronized within SIGA - timing is unimportant general:

1. All times in nanoseconds
2. All loads in picofarads
3. TBUS, FAST and INTERLEAVER timing are relative to rising T_CLK
4. SWITCH - REQ timing is relative to rising R_CLK
5. SWITCH - SER timing is relative to rising S_CLK
6. TCS timing is relative to falling C_CLK