

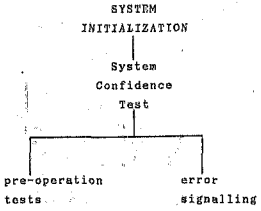
https://dspace.org/

Fault Tolerance In Computer Systems

Bury, Michael John

2015-01-12
http://hdl.handle.net/10539/16479

Downloaded from a DSpace repository. https://dspace.org/



PRE-OPERATION TESTS

All test procedures are used, starting with full testing of the cpu.

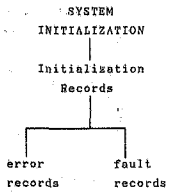
Next, the memory is tested, according to the procedure outlined previously. Error and fault records are generated when necessary.

All servos are exercised to their fullest extent, ensuring that they may be controlled over their full range. Sensors are checked for reasonableness and consistency. If the values they measure have a known power-up value, then the reading is checked for correctness.

The bus controller is tested according to the standard procedure.

ERROR SIGNALLING

Error and faults are signalled by the test routines when they are detected.



ERROR RECORDS

Error records conform with the standard format.

FAULT RECORDS

Fault records conform with the standard format.

This appendix contains descriptions of the software modules that make up the software system. These descriptions were derived from the software functional specifications, and led to the design and coding of the actual modules. The listings are contained in a separate volume.

1. Check_Clocks

PURPOSE:

Checks that all clocks in the system are sufficiently close to one another. If they are not, then the module calls "set_up_stagger", supplying an average clock value to reset by. It then rechecks, and if there is still too much difference, an error record is generated. An error record is also generated if one clock drifts unacceptably far from the others.

EXECUTION SEQUENCE:

1. Obtains all clock values

2. Checks variation in clock times

3. If sufficiently close, then terminates with status=OK

4. If a clock shows excessive inaccuracy, then an error record is generated

5. If the difference in times is above a limit, then resynchronization is done by calling "set_up_stagger"

6. Obtains all clock values again, and checks the variation again

7. If still erroneous, then an error record is generated, and a status value indicating the fault is returned

2. Check_Consistency

PURPOSE:

Checks that an input value is sufficiently close to the previous input value, produces an error record if it is not, and returns an appropriate status value. The previous value and previous time are updated.

EXECUTION SEQUENCE:

1. Calculates the rate of change of the variable, using the present and previous values and times

2. Compares actual rate of change with the maximum positive and negative possible rates of change

3. If unacceptable, an error record is generated, and an appropriate status is returned.

4. The previous value and time are updated, and the module terminates

3. Check_Feedback

PURPOSE:

Checks that the feedback value resulting from an output is within the expected bounds. If the value is unacceptable, then an error record is generated, and an appropriate status value is returned.

EXECUTION SEQUENCE:

1. Calculates upper and lower bounds for the feedback value, using the supplied expected value and the I/O channel tolerance

2. Compares the feedback value with these bounds

3. If unacceptable, then an error record is generated, and an appropriate status value is returned

4. Check_Readback

PURPOSE:

Checks that the output value requested has been correctly produced at the I/O channel. If the value read back is not sufficiently close to the requested value, then an error record is generated, and an appropriate status value is returned.

EXECUTION SEQUENCE:

1. Calculates upper and lower bounds for the readback value, using the requested value and the device tolerances

2. Checks the readback value to see if it is within the bounds

3. If unacceptable, then an error record is generated, and an appropriate status value is returned

5. Check_Reason

PURPOSE:

Checks that an input value is possible for an input channel. If the value is not possible under normal operation, then an error record is generated, and an appropriate status value is returned.

EXECUTION SEQUENCE:

1. Compares the input value with the upper and lower bounds of possibility for the input channel

2. If unacceptable, then an error record is generated, and an appropriate status value is returned

~~6. Task~~ Completion

PURPOSE:

Checks that all executions of a task have ended within an acceptable time period. Ensures that the correct tasks were executed. If either test fails, then an error record is generated, and an appropriate status value is returned.

EXECUTION SEQUENCE:

1. Obtains all task complete records

2. Compares task IDs

3. If not the same, generates an error record and returns an appropriate status

4. Compares the task-complete times

5. If unacceptable, generates an error record and returns an appropriate status value

7. *Deschedule_Task*

PURPOSE:

Removes a task from the execution list and cancels the task schedule. If the task is not on the execution list,

then an error record is generated, and an appropriate status is returned.

EXECUTION SEQUENCE:

1. Searches execution list for task ID

2. If not found, then an error record is generated, an appropriate status value is returned, and the module terminates

3. If found, the task is removed from the scheduling list

8. End_Off_Task

PURPOSE:

Ends the execution of every task by producing and checking task-complete records. When there has been a task scheduling error, and task voting is in progress, the routine will vote on the task to execute. If there is a voting error, then an error record is generated. The task checks to see how much time there is before the next task is due, and schedules a test task if there is enough time.

EXECUTION SEQUENCE:

1. Produces a task complete record, with task ID and completion time

2. Checks task completion

3. Votes on the next task when necessary

4. If there is a voting error, then an error record is produced

5. Calculates the free time before the next application task

6. Finds a test task to run in the free time

7. If a test task is found then it is scheduled

9. Exchange_Data

PURPOSE:

To supply all versions of a piece of data to each node. The routine sets up the command sequence needed by the 1553 controller board, to communicate with the other nodes and exchange the data. The data is placed in a buffer. 1553 errors are reported via the status word, and error records are generated.

EXECUTION SEQUENCE:

1. Sets up the command sequence for the 1553 board

2. Places data in the transmit buffer

3. Initiates the exchange

4. Receives the data from the receive buffer

5. If there is a 1553 error, a status value appropriate to the error is returned, and an error record is generated

10. Initialize_Devices

PURPOSE:

Initializes all parts of the system. The module checks that the initialization of each device has succeeded. If

there are initialization errors, then error records are generated, and appropriate status values are returned.

EXECUTION SEQUENCE:

1. Initializes a device

2. Checks the initialization

3. If the initialization has failed, then an error record is generated

4. The sequence is repeated for all devices

11. Isolate_Node

PURPOSE:

Instructs an erring node to stop normal operation and to perform self-testing until it has validated itself as much as possible. The routine votes before performing the isolation, ensuring that at least two nodes identify the node as erring. The module waits for a reply from the erring node, indicating its status. If the status indicates errors, then the module generates appropriate error reports, and returns an appropriate status value.

EXECUTION SEQUENCE:

1. Votes on the isolation of the node

2. Sends a message to the node, telling it to perform the tests

3. Waits for the results of the tests

4. If errors occur, then error reports are generated, and a status value is returned

12. Local_Exception_Sequence

PURPOSE:

Responds to an instruction from the other nodes when the node has made too many errors, by pinpointing the fault as far as possible, and communicating the results of the tests with the other nodes. If no faults are found, then a status of OK is returned. Otherwise an appropriate status is given.

EXECUTION SEQUENCE:

1. Receives a "begin" message from the other nodes

2. Pinpoints the fault if possible

3. Receives a "send" message from the other nodes (this prevents untimely transmissions)

4. Transmits the results of the test

5. Returns a status value

13. Pinpoint_Fault

PURPOSE:

Performs testing of a node to isolate a fault as far as possible. A table, of data IDs and possible faults which will cause errors in the data, is consulted to decide on the tests to perform. When the fault is found, an appropriate status value is returned.

EXECUTION SEQUENCE:

1. Obtains fault possibilities from the fault table

2. Tests for each fault in the order of probability, until the root cause is found

3. Returns a status value and terminates

14. Produce_Error_Record

PURPOSE:

Makes a record of an error by increasing a count for the appropriate error type. The routine checks the error count against a maximum number of tolerable errors and invokes "produce_fault_record" when necessary.

EXECUTION SEQUENCE:

1. Increases error count for the error type

2. Checks error count against a maximum error count

3. If there are too many errors, then "produce_fault_record" is called

4. Otherwise the module terminates

15. Produce_Fault_Record

PURPOSE:

Makes a record of a fault when too many errors have been detected. When the error type is insufficient to identify the fault, fault pinpointing is done.

EXECUTION SEQUENCE:

1. Calls "Pinpoint_Fault" if the error type is *insufficient to identify* the fault

2. Produces a fault record by incrementing a fault-type count

16. Produce_Random_Number

PURPOSE:

Produces pseudo-random numbers for test purposes.

EXECUTION SEQUENCE:

1. Calculates a pseudo-random number, using a seed to start with, and the previous random number afterwards

2. Sets previous random number to the current random number

3. Returns the random number, and terminates

17. Produce_Task_Complete_Record

PURPOSE:

Makes a record of the task just completed, with its completion time.

EXECUTION SEQUENCE:

1. Establishes completion time

2. Produces record and terminates

18. Receive_From_Bus

PURPOSE:

Sets up the commands to the 1553 controller board to enable the reception of messages from the bus. The module extracts the message from the reception and places it in a message buffer. The status value returned reflects the status indicated by the 1553 controller board.

EXECUTION SEQUENCE:

1. Sets up 1553 control commands

2. Waits for message

3. Extracts message and places it in a buffer

4. If an error is found, an error is generated

5. Returns a status, and terminates

19. Remote_Exception_Sequence

PURPOSE:

Isolates an erring node and resets it when it has indicated that is ready for operation.

EXECUTION SEQUENCE:

1. Isolates the node

2. Waits for the response

3. If the node is ready, then the module resets the node by passing it the current execution list

4. If the node is not ready, then the status returned indicates that the node may not be used, or may be *used* in a diminished capacity.

20. Remote_Test

PURPOSE:

Instructs *another node to perform* a test. The decision to activate a remote test is voted upon. The module waits for the results of the test. If the local node is to be tested, then it calls "respond_to_remote_test" and terminates itself.

EXECUTION SEQUENCE:

1. Votes on the node to test, and the test type

2. If self then terminates after initiating response module

3. Sends "begin" message to test node

4. Waits to receive the results of the test

5. *If no errors then terminates, returning status=OK*

6. Otherwise generates an error record and returns appropriate status

21. Reset_Node

PURPOSE:

Informs a recently validated node of the current execution list. If the node does not acknowledge

reception of the list, then an error record is generated.

EXECUTION SEQUENCE:

1. The execution list is sent to the node

2. The module waits for a limited time for a reply

3. If no reply is received, then the module generates an error record

4. The module terminates, returning an appropriate status value

22. Respond_to_Remote_Test

PURPOSE:

Performs a test as instructed by the other nodes. When elementary tests are performed, the results of the tests are sent to the other nodes. If full tests are performed, then the status is returned to the other nodes.

EXECUTION SEQUENCE:

1. Receives instruction from other nodes to proceed.

2. Performs the required test

3. Transmits the results of the test to the other nodes

4. An appropriate status is returned, and the module is terminated

23. Schedule_Task

PURPOSE:

Sets up the execution of a task. The module checks for scheduling clashes, schedules the task as requested, and returns an appropriate status.

EXECUTION SEQUENCE:

1. Checks the execution list for clashes

2. If there is a clash, then the module returns a status value, produces an error record, and terminates

3. The module schedules the task to run as requested

4. The module terminates, returning the appropriate status

24. *Set_Up_Stagger*

PURPOSE:

Sets the clock in each node to a specified time, with a time stagger between the nodes.

EXECUTION SEQUENCE:

1. All node times are obtained

2. All setting times are calculated

3. Setting times are checked with the other nodes' calculations

4. If there is disagreement, then an error record is generated, and an appropriate status is returned

The module waits until it is time to set

6. The module sets the node-time to the required value

25. Start_Up_Sequence

PURPOSE:

Sets up the system for operation by initializing all devices, testing each node, setting up the zero time, and checking the clocks. If any of these operations fails, then a warning is given.

EXECUTION SEQUENCE:

1. Initializes devices, warns if any critical initialization fails

2. Tests the system, warns if any test fails

3. Sets up the clocks

4. Checks the clocks, warns if clock set-up has failed

5. Schedules the first application task

26. Test_Ana_Input

PURPOSE:

Tests the input channels of the analog I/O board by calling "validate_input" for each channel. All board

registers are read to ensure that they are valid. If channels are found to be faulty, then they are marked as such.

EXECUTION SEQUENCE:

1. All registers are read and checked

2. Validate_input is invoked for all input channels

3. Faulty channels are marked as such

4. An appropriate status value is returned, and the module terminates

27. Test_Ana_Output

PURPOSE:

Tests the output channels of the analog I/O board by setting each output to a suitable value, and calling "validate_output" for each channel. All board registers are read to ensure that they are valid. If channels are found to be faulty, then they are marked as such.

EXECUTION SEQUENCE:

1. All registers are read and checked

2. A channel is set an appropriate value

3. Validate_output is invoked for the channels

4. The process is repeated for all channels

5. Faulty channels are marked as such

6. An appropriate status value is returned, and the module terminates

28. Test_Inter_Node_Comms

PURPOSE:

The module tests an inter-node communication link by first voting on the link to test, and then passing known and random messages across the link in order to verify its operation. If errors occur, then error reports are generated, and an appropriate status value is returned.

EXECUTION SEQUENCE:

1. All versions of the proposed test link are obtained

2. The agreement is checked, and if there is an error, an error record is generated

3. The link decided upon is used to send a standard message from one node to another

4. On reception, the message is checked, and if incorrect, and error record is generated

5. The message is passed in the other direction, and checked

6. The process is repeated for a random message

7. At the end of the test, the results are exchanged, so that each node can update its records

29. Test_I/O_Equipment

PURPOSE:

The module invokes the necessary input and output testing routines (which may vary from system to system). At the end of the test, the results are sent to the other nodes, and a status value is returned.

EXECUTION SEQUENCE:

1. Analog inputs are tested

2. Analog outputs are tested

3. The test results are exchanged with the other nodes

30. Test_Memory

PURPOSE:

Tests regions of memory. The module works out how much of the region it can test within the available time. The region to be tested is copied and checked. Then a sequence of sliding ones is written into the block, and verified. Pseudo-random numbers are written into the block, and into a temporary location. A comparison is made. Finally, the original data is written back to the original block, and checked. The results of the checks are communicated to the other nodes, and error records are generated when necessary.

EXECUTION SEQUENCE:

1. The module works out which region it will test

2. The region is copied, and the copy verified

3. If the copy is incorrect, it is recopied, and reverified

4. If the copy is still incorrect, then an error record is generated

5. Sliding ones are written into the block, and verified

6. If there is an error, an error record is generated

7. Random numbers are written into the block, and another block, and verified

8. If there is an error, an error record is generated

9. The results of the tests are sent to the other nodes

10. A appropriate status value is returned, and the module terminates

31. Test_Processor

PURPOSE:

Tests all instructions of the processor to ensure that it functions correctly. Core instructions are executed first because they are essential to further testing. Then a block of memory is tested for use in the processor tests. All other instructions are then methodically tested. Any errors lead to the creation of error records.

EXECUTION SEQUENCE:

1. Core instructions are tested

2. If there is an error, the instructions are re-tried

3. If there is still an error, an error record is generated

4. A memory block is tested

5. All instructions are tested

6. All peripherals components on the processor board are tested

7. If there are errors, then error records are generated

8. A suitable status value is returned, and the module terminates

32. Test_System

PURPOSE:

The module validates the entire system by performing complete tests of all parts of the system. A status value is returned.

EXECUTION SEQUENCE:

1. Processor testing is done

2. Memory testing is done

3. I/O equipment testing is done

4. Inter-node communications testing is done

5. A status value is returned, and the module terminates

33. Transmit_on_Bus

PURPOSE:

Sets up the commands to the 1553 controller board to enable the transmission of messages on the bus. The module puts the message into the transmission buffer, in the correct format. A status value is returned, which indicates the 1553 controller board status.

EXECUTION SEQUENCE:

1. Sets up 1553 control commands

2. *Formats the message* into the transmission buffer

3. Activates the transmission

4. If an error is found, an error record is generated

5. A status value is returned, and the module terminates

34. Validate_Input

PURPOSE:

Checks that an input value is valid. If the input is at a *remote node, then the node performs* transmission and reception on the bus to obtain the data. When each node has a version of the inputs, the data is exchanged, and voted on. When errors are found, error records are generated, and an appropriate status value is returned.

EXECUTION SEQUENCE:

1. If the data is at a remote node, then an instruction to send the data is transmitted to the remote node

2. The module then waits for the data

3. If the data is only local, then it is sent to the other nodes on request

4. The data is checked for reason

5. The data is checked for consistency

6. If there is more than one version of the data, then these are compared with a tolerance

7. Errors are collected in an error record and the returned status value

35. Validate_Output

PURPOSE:

Checks that an output is executed correctly. If the output and/or associated inputs (if they exist) are at a remote node, then the node performs transmission and reception in order to perform the output and to read the inputs. If each node has a version of the output or inputs, then the data is exchanged and checked against a tolerance. When errors are found, error records are generated.

EXECUTION SEQUENCE:

1. All versions of the required output are obtained

2. The versions are checked, and if there is an error, an error record is generated

3. The output is performed

4. Readback and feedback values are obtained, if available, and checked

5. If there are errors, then error records are generated, and they are reflected in the status value.

This appendix describes a reduced user interface to the communication controller board. This reduced definition was used to implement inter-node communication in the system. The communication standard on which this is based is the *MIL-STD-1553B standard.*

A10.1 The System Configuration Pointer (SCP), and the Intermediate System Configuration Pointer (ISCP)

The system configuration pointer is situated at a fixed memory location in ROM, and applies to all peripherals which require an address from which they receive their commands. The SCP is loaded with an address for a RAM location (the ISCP), into which individual control block addresses can be loaded. After the appropriate address for each peripheral is loaded into this address in RAM, the *peripheral's select line* is activated, and it can read the value at the ISCP.

A10.2 System Control Block (SCB)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	INT		X	X	STATUS					ERR					
X	INT-ACK		X	X	RE		X	X		COMMAND					
CBL OFFSET															
BEQ OFFSET															
X	X	X	X	X	X	X	X	X	X	X	X	X	X	RTU ADDRESS	

STATUS WORD -

Bit 14: BEV - When set, indicates that a bus event has occurred

Bit 13: CER - When set, indicates that a controller error has occurred

Bits 10 to 8: STATUS - 0 - Idle
1 - Busy executing a command
2 - Suspended

Bits 7 to 0: ERR - 0 - No error
1 - BEQ is full
2 - Invalid control command from subsystem
3 - Memory error in interface's internal memory
4 - Undefined action command
5 - Invalid action command (interface not in the correct mode)
6 - Invalid parameters in action command

COMMAND WORD -

Bit 14: ACK-BEV - Acknowledges the bus event

Bit 13: ACK-CER - Acknowledges the interface error

Bit 10: RE - Resets the interface (same as hardware reset)

Bits 7 to 0: COMMAND - 0 - NOP - used to acknowledge interrupt conditions
1 - START - Start execution of the CBL (command block list)
2 - SUSPEND - Suspend execution of the CBL after the current command is complete
3 - RESUME - Resume execution

CBL OFFSET WORD -

Specifies the offset portion of the address of the first CB on the CBL. It is accessed only when the command is START.

BEQ OFFSET WORD -

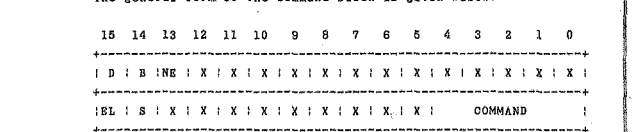
Specifies the offset portion of the address of the first *BEDB* on the BEQ. It is accessed only on initialization, as the BEQ is cyclic.

RTU ADDRESS -

Specifies the RTU address of the interface. It is only accessed on initialization, as it is not changed.

A10.3 Command Block (CB)

The general form of the command block is given below:



STATUS WORD -

Bit 15: D - DONE - Set to 0 by subsystem when the CB is placed on the CBL, set to 1 by the interface when it has finished executing the CB.

Bit 14: B - BUSY - Set to 0 by subsystem when the CB is placed on the CBL, set to 1 by the inteface while it is executing the command, and to 0 when it has finished.

Bit 13: NE - NOERROR - Set to 0 by the subsystem when the CB is placed on the CBL, set to 1 by the interface if no error occurred during the execution of the command.

COMMAND WORD -

Bit 15: EL - ENDLIST - Set to one if this CB is the last in the CBL.

Bit 14: S - SUSPEND - Suspend execution after this command is complete

Bits 4 to 0: COMMAND - Specifies the action opcode (See later for action opcode definitions).

LINK FIELD -

Contains the 16-bit offset from SCB(0) to the next CB in the CBL.

COMMAND-SPECIFIC PARAMETERS -

Variable length field, dependent on the action opcode (See later for definitions of command specific parameters).

Action Opcodes and Definitions.

00H: NOP - no action is taken
- no command-specific parameters are used

02H: CONFIGURE - the interface is set up as a BC or RTU
- command-specific parameter:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	MODE

MODE - 00 - RTU
01 - BC

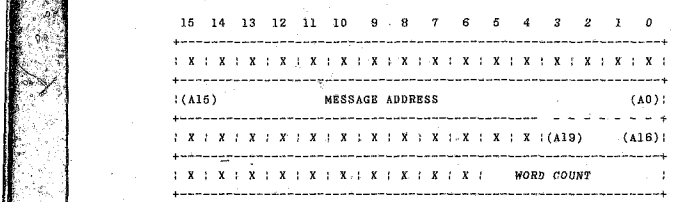
0CH: BUS-CONTROL ACCEPTANCE ENABLE - enable interface to accept control if offered
- no command-specific parameter

0DH: BUS-CONTROL ACCEPTANCE DISABLE - disable control acceptance
- no command-specific parameter

0FH: SET SUBSYSTEM ERROR - sets 'subsystem error' bit in the 1553 status word
- no command-specific parameter

0FH: CLEAR SUBSYSTEM ERROR - clears 'subsystem error' bit in the 1553 status word
- no command-specific parameter

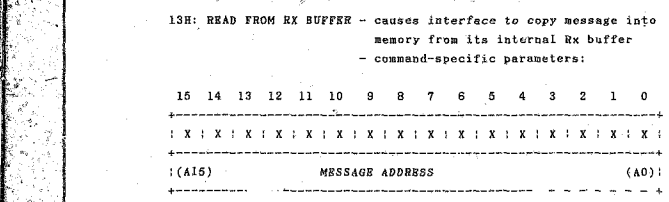
12H: WRITE TO TX BUFFER - causes interface to copy message from memory into its internal Tx buffer
- command-specific parameters:



MESSAGE ADDRESS - *Points* to the message anywhere in the 1 Mbyte of addressable memory

WORD COUNT - The number of words in the message. A value less than 1 or greater than 32 is invalid

13H: READ FROM RX BUFFER - causes *interface to copy* message into memory from its internal Rx buffer - command-specific parameters:

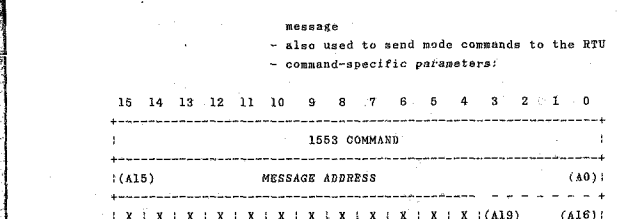


MESSAGE ADDRESS - Points to the message anywhere in the 1 Mbyte of addressable memory

WORD COUNT - The number of words in the message. A value less than 1 or greater than 32 is invalid

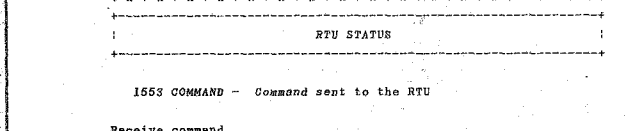
14H: TRANSMIT TO RTU - Causes the interface to send a 'receive' *1553 command* to the RTU, followed by the

message
- also used to send mode commands to the RTU
- command-specific *parameters;*



1553 COMMAND - *Command sent* to the RTU

Receive command



RTU ADDRESS - 0, 1, or 2, depending on which node is the RTU

T/R - 1 (RTU must receive)
SUBADDRESS/MODE - Anything but 00000 or 11111
DATA CNT/MODE CODE - Data count

Dynamic bus control mode command

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTU ADDRESS					*T/R*	*SUBADDRESS/MODE*					*DATA CNT/MODE CODE*				
X	X	X	X	X	1	0	0	0	0	0	0	0	0	0	0

RTU ADDRESS - 0, 1, or 2, depending on which node is offered control

Transmit status word command

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTU ADDRESS					T/R	SUBADDRESS/MODE					DATA CNT/MODE CODE				
X	X	X	X	X	1	0	0	0	0	0	0	0	0	1	0

RTU ADDRESS - 0, 1, or 2, depending on which node is to send its status word

MESSAGE ADDRESS - Points to the message anywhere in the 1 Mbyte of addressable memory
- not valid when mode commands are sent

RTU STATUS - Word received from the RTU after completion of the command

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTU ADDRESS					ME	0	TXR	X	X	X	BCR	B	SF	BCA	TF

RTU ADDRESS - 0, 1, or 2, depending on which node is sending status word

ME - *MESSAGE ERROR* - set if the message received by the RTU was in error (invalid sync field, or invalid Manchester code, or incorrect number of bits, or even parity, or improperly timed syncs, or unsupported command, or incorrect data count)

TXR - TRANSMIT READY - Transmit buffer ready for transmission

BCR - BROADCAST COMMAND RECEIVED - set if the command received by the RTU was a broadcast command

B - BUSY - set if BEQ is full or if the RTU controller is busy executing a CB on the CBL

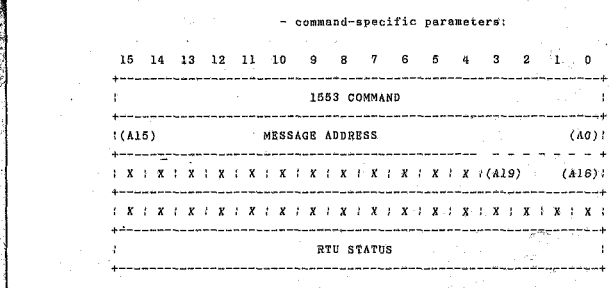
SF - SUBSYSTEM FLAG - set if the subsystem is faulty

BCA - BUS CONTROL ACCEPT - set to acknowledge acceptance of control

TF - TERMINAL FLAG - set if the interface is faulty

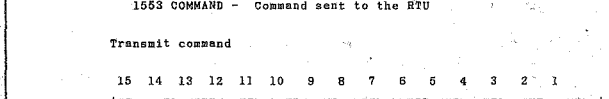
15H: RECEIVE FROM RTU - Causes the interface to send a 'transmit' 1553 command to the RTU

- command-specific parameters:



1553 COMMAND - Command sent to the RTU

Transmit command



RTU ADDRESS - 0, 1, or 2, depending on which node is the RTU

T/R - 0 (RTU must transmit)
SUBADDRESS/MODE - Don't care
DATA CNT/MODE CODE - Data count

MESSAGE ADDRESS - Points to the message anywhere in the 1 Mbyte of addressable memory

RTU STATUS - Word received from the RTU after completion of the transfer

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTU ADDRESS						ME	0	X	X	X	X	BCR	B	SF	BCA	TF	

RTU ADDRESS - 0, 1 or 2, depending on which node is sending the status word

ME - MESSAGE ERROR - set if the message received by the RTU was in error (invalid sync field, or invalid Manchester code, or incorrect number of bits, or even parity, or improperly timed syncs, or unsupported command, or incorrect data count)

TXR - TRANSMIT READY - Transmit buffer ready for transmission

BCR - BROADCAST COMMAND RECEIVED - set if the command received by the RTU was a broadcast command

B - BUSY - set if BEQ is full or if the RTU controller is busy executing a CB in the CBL

SF - SUBSYSTEM FLAG - set if the subsystem is faulty

BCA - BUS CONTROL ACCEPT - set to acknowledge acceptance of control

TF - TERMINAL FLAG - set if the interface is faulty

16H: TRANSFER FROM RTU TO RTU - Causes the interface to send a 'receive' 1553 command to one RTU and a 'transmit' 1553 command to another RTU

- *command*-specific parameters:

```
 15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          1553 COMMAND TO THE RECEIVING RTU                    |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          1553 COMMAND TO THE TRANSMITTING RTU                 |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| X | X | X | X | X | X | X | X | X | X | X | X | X | X | X | X |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          STATUS WORD OF RECEIVING RTU                         |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          STATUS WORD OF TRANSMITTING RTU                      |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

1553 COMMAND TO THE RECEIVING RTU

```
 15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   RTU ADDRESS     |T/R| SUBADDRESS/MODE   | DATA CNT/MODE CODE|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| X | X | X | X | X | 1 | X | X | X | X | X | X | X | X | X | X |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

RTU ADDRESS - 0, 1 or 2, depending on which *node* is the RTU
T/R - 0 (RTU must transmit)
SUBADDRESS/MODE - Anything but 00000 or 11111
DATA CNT/MODE CODE - Data count

1553 *COMMAND TO THE TRANSMITTING* RTU

```
 15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   RTU ADDRESS     |T/R| SUBADDRESS/MODE   | DATA CNT/MODE CODE|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| X | X | X | X | X | 0 | X | X | X | X | X | X | X | X | X | X |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

RTU ADDRESS - 0, 1 or 2, depending on which node is the RTU
T/R - 0 (RTU must transmit)
SUBADDRESS/MODE - Don't care
DATA CNT/MODE CODE - Data count

RTU STATUS - Word received from each RTU after completion of the transfer

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RTU ADDRESS						ME	0	X	X	X	BCR	B	SF	BCA	TF

RTU ADDRESS - 0, 1 or 2, depending on which node is sending the status word

ME - MESSAGE ERROR - set if the message received by the RTU was in error (invalid sync field, or invalid Manchester code, or incorrect number of bits, or even parity, or improperly timed syncs, or unsupported command, or incorrect data count)

TXR - TRANSMIT READY - Transmit buffer ready for transmission

BCR - BROADCAST COMMAND RECEIVED - set if the command received by the RTU was a broadcast command

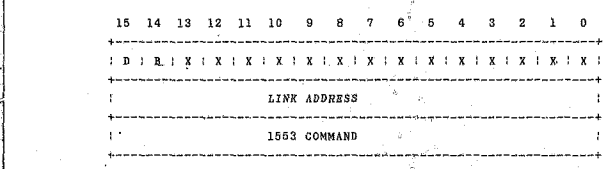
B - BUSY - set if BEQ is full or if the RTU controller is busy executing a CB in the CBL

SF - SUBSYSTEM FLAG - set if the subsystem is faulty

BCA - BUS CONTROL ACCEPT - set to acknowledge acceptance of control

TF - TERMINAL FLAG - set if the interface is faulty

A10.4 Bus Event Descriptor Block (BEDB)



STATUS WORD

Bit 15 - D - DONE - Set to 1 by the interface when it has finished writing into the *BEDB*

Bit 14 - B - BUSY - Set to 1 by the interface while it is writing *into the BEDB*; set 0 when the interface is done

LINK ADDRESS - The 16-bit offset of the next BEDB. The last link address should point to the first BEDB

1553 COMMAND - The 16-bit command word received via the bus

ERROR BITS FROM THE RTU HYBRID -

Bit 15 - ANYERR - Set to 1 if GBNR, RTADER, *HSFAIL*, *ERROR*, RTO, TXTO, PARER or MANER is set

Bit 14 - *GBNR* - Set if a block *was received*, but it was not good

Bit 13 - RTADER - RTU address parity error

Bit 12 - HSFAIL - Handshaking error

Bit 11 - ERROR - Encoding error

Bit 10 - RTO - Reply time exceeded

Bit 9 - TXTO - Transmitter timeout exceeded

Bit 8 - PARER - Parity error in a received word

Bit 7 - MANER - Manchester coding error in a received word

Bit 6 - LTFAIL - Loop test failure

Bit 5 - RBUS - Bus: B/notA

Bit 4 - BCST - Broadcast command received

Bit 3 - MODE - Mode command received

Bit 2 - SYNC - Synchronize command received

Bit 1 - DBCREQ - The RTU has accepted dynamic bus control

Bit 0 - ANYSTAT - Set if SYNC or DBCREQ is set

Author Bury Michael John
Name of thesis Fault Tolerance In Computer Systems. 1986

PUBLISHER:
University of the Witwatersrand, Johannesburg
©2013

LEGAL NOTICES:

Copyright Notice: All materials on the University of the Witwatersrand, Johannesburg Library website are protected by South African copyright law and may not be distributed, transmitted, displayed, or otherwise published in any format, without the prior written permission of the copyright owner.

Disclaimer and Terms of Use: Provided that you maintain all copyright and other notices contained therein, you may download material (one machine readable copy and one print copy per page) for your personal and/or educational non-commercial use only.

The University of the Witwatersrand, Johannesburg, is not responsible for any errors or omissions and excludes any and all liability for any errors in or omissions from the information on the Library website.