Fermilab

Engineering Note

To: Muon Electronics group

From: Boris Baldin

CC: Fritz Bartlett, Dmitri Denisov, Tom Diehl, Darien Wood

Date: 02/19/01

Re: Requirements for multi-block transfers using 1553 bus (Updated 02/16/01)

Purpose

This update supersedes all previous versions of this document and contains current requirements for multi-block data transfers via 1553 bus for muon front-end (FE) electronics systems. The expectation is that all the limitations of the hardware design of different sub-systems are taken into account in this update. D0 experiment uses the 1553 bus to provide remote controls and monitoring of the detector electronics.

Hardware

The existing hardware from run I includes 1553 bus cables in the detector area, 1553 bus fan-outs and 1553 control nodes, which are 6U VME crates with VME processor and 1553 bus driver cards. All muon FEs are equipped with 1553 remote terminal (RT) interface.

Software

D0 on-line group uses EPICS software to support communications between on-line hosts and remote terminals. RT functions in muon FE systems are supported either by the DSP software or by programmed array logic hardware.

Multi-block data transfers

The main requirement for multi-block transfers comes from a need for muon FEs to store and retrieve relatively large arrays of data used for setting up hardware and software parameters. The size of the data arrays varies from a few kilobytes to several megabytes. The multi-block transfers are *always initiated by the hos*t and consisted of a sequence of transmit/receive commands sent to the same sub-address of the dedicated remote terminal. The RT receives or transmits data in 32 16-bit word units (maximum allowed by the MIL-1553B standard). The last data transfer in the sequence may have less than 32 words of data. The RT stores or retrieves data according to the contents of the internal address and byte count registers, which are previously loaded by a separate 1553 command. Currently, the EPICS software allows transferring up to 16 KBytes of the data to/from host in one sequence. Unless newer versions of the EPICS software with larger data buffer size are available, an end user is responsible for implementation necessary routines in host and RT software to transfer larger data arrays.

Remote Terminal requirements

1. Each RT must have two dedicated sub-addresses: one for data upload and another for data download. After the data transfer has been initialized, any multi-block data transfer commands of the opposite direction must be ignored at the software level until the original transfer is complete (i.e. byte counter is exhausted). The EPICS software has the ability to lock up multi-block transfer to serve the same purpose. This feature may only be used with the series of data transfer commands and will improve reliability of the transmission to some extent.

2. Each RT must have one dedicated sub-address for setting up data transfer parameters and reporting multi-block status. Depending on the value of the T/R bit in the command, there are six or seven 16-bit words defined with this sub-address. Writing six words to it will initialize RT's multi-block logic and set up logical destination address for the data, data length in bytes and 16-bit checksum word. Reading seven words from this sub-address provides current multi-block status. The sixth 16-bit word represents multi-block control word, which describes type of the data transfer (upload or download, see Table I below). The seventh word contains error bits representing status. After host initialized the data transfer parameters, the multi-block data transfer commands received by the RT must correspond to the direction of the data transfer determined by upload/download bits. If a data transfer of the opposite direction is detected, "multi-block error" bit must be set in the multi-block status word (see Table I). This bit also must be set if RT detects another attempt to set up parameters before current multi-block transfer is complete.

3. Each RT must have one dedicated mode code "reset multi-block mode" in the mode command to reset RT's multi-block logic. This command is sent before data transfer to reset the status of RT's multi-block logic to a default state (see Table I below) and may be used as an emergency reset..

4. Two data direction bits and two error bits required to be implemented in two separate 16-bit registers. The remaining six bits of the lower byte in both 16-bit registers are reserved and the upper byte bits are user defined. Multi-block transfer status can be checked by accessing multi-block control word and analyzing data direction bits (non-zero, if in progress) during the data transfer. Optionally, a RT may have a "multi-block transfer in progress" bit, which is an OR of two data direction bits. If exists, this bit has to be implemented in the FE's status register. If PROM programming follows data downloading, the RT has to reset download bit only after the programming process is finished. A "multi-block error" bit will be set to one by the RT logic when an attempt to interfere with the current transfer occurs or RT detects any unusual status (user defined). The RT will set a "checksum error" bit if checksum calculation fails at the end of data download. All control and status bits and data pointers (logical address and byte count) are reset by the "reset multi-block mode" mode command described earlier. All four control/status bits have default value of zero.

5. The logical addressing of the data is user defined, however user must define the data arrays in such a way that they always consist of the even number of bytes. This requirement eliminates the need of adding pad byte in multi-block data upload.

6. There is a minimum time interval (inter-message gap, 4 μS) specified for the RT between beginning and end of sequential transmit/receive commands. For the software-based implementation of the 1553 interface, it is important to process data in parallel to avoid additional dead time in 1553 command response.

7. The RT must be capable to respond to the other non-multi-block 1553 commands (except the ones described above) interleaved with multi-block data transfer commands.

Host software requirements

1. Before starting actual data transfer, host software is required to verify that there is no current data transfer in progress with the selected RT. This has to be done by testing upload/download status bits in the RT multi-block status word. After the multi-block parameters are set, host requests the multi-block status word from the RT. If multi-block error bit is set, the host software must cease communications with the RT and generate an error message.

2. An additional data byte has to be added to the data array that has odd number of bytes to achieve proper 16-bit word data representation required by the EPICS software.

3. Host software calculates 16-bit checksum value and sends it with "receive multi-block status" command to the RT before data download starts (see Table I below). Host software receives checksum using "transmit multi-block status" command at the end of the data upload and compares it to the calculated checksum.

4. Host software may request RT to send multi-block status by accessing RT's muti-block status sub-address (see above) during the data transfer. The only acceptable multi-block status command during this time is "transmit multi-block status".

5. Host software may include some means of preventing simultaneous access to the same RT from multiple hosts.

Table I. Multi-block data transfer parameters format.

Parameter	***Data words (16 bit)***	***Default value***	***Receive value***	***Transmit value***	***Comment***
Destination address	2	0	Destination address (logical)	Current destination address	Big-Endian byte order, logical address: 0 – illegal, 1..FFFFFFFF - valid
Data byte count (always even)	2	0	Data byte count	Current byte count	Big-Endian byte order, set to zero when finished, updated during transfer
Multi-block checksum word	1	0	Checksum if download, zero otherwise	Current checksum	16-bit checksum, updated during the data transfer, see paragraph below
Multi-block control word	1	0	Data transfer direction	Current transaction status	Bit 0 – download, bit 1 – upload, 2..7 – reserved, 8..15 – user defined
Multi-block status word	1	0	N/A	Current transaction status	Bit 0 – checksum error, bit 1 – multi-block error, 2..7 – reserved, 8..15 – user defined

Multi-block data download sequence

1. The multi-block data download consists of the following 1553 commands:

a. Transmit multi-block status (verify that there is no data transfer in progress)

b. Reset multi-block logic (set parameters to default)

c. Receive multi-block status (setup data transfer parameters)

d. Transmit multi-block status (verify that there is no collision)

e. Receive data words (download data array)

f. Transmit multi-block status (send checksum and status to the host)

Notes: During the sequence, the RT must respond to any other 1553 commands that the 1553 driver may send to it. Some of the multi-block commands may be ignored at the software level. EPICS software may "lock" the sequence of commands during phase (e).

Multi-block data upload sequence

2. The multi-block data upload consists of the following 1553 commands:

a. Transmit multi-block status (verify that there is no data transfer in progress)

b. Reset multi-block logic (set parameters to default)

c. Receive multi-block status (setup data transfer parameters)

d. Transmit multi-block status (verify that there is no collision)

e. Transmit data words (upload data array)

f. Transmit multi-block status (send checksum and status to the host)

Notes: The host does not provide a checksum (sets to zero) when setting up data upload from the RT during phase (c). During the sequence, the RT must respond to any other 1553 commands that the 1553 driver may send to it. Some of the multi-block commands may be ignored at the software level. The RT sends calculated during phase (f) checksum at the end of the data upload. EPICS software may "lock" the sequence of commands during phase (e).

Checksum calculation

One of the following algorithms must be used to calculate 16-bit checksum value:

1. Addition:

a. Add sequentially 16-bit words with no carry

b. Invert all bits in the result

2. Subtraction:

a. Set initial value to –1

b. Subtract sequentially 16-bit words with no carry