SpaceWire Remote Memory Access Protocol

Steve Parkes and Chris McClements

University of Dundee, Applied Computing, Dundee, DD1 4HN, Scotland, UK.
sparkes@computing.dundee.ac.uk, , cmclements@computing.dundee.ac.uk.

Abstract

SpaceWire is a spacecraft onboard communications network used to connect together electronic sub-systems like sensors, mass-memory, processors, control and telemetry/telecommand units. SpaceWire provides high bandwidth communication using point-to-point links between sub-systems or networked interconnection using routing switches to forward packets of data across the network. SpaceWire has a simple interface which can be readily implemented in a range of different chip technologies including Field Programmable Gate Arrays (FPGAs). The SpaceWire standard defines “links, nodes and routers.” It does not define any higher level communications standards for operating over the SpaceWire network other than straightforward packet encapsulation. This paper describes the Remote Memory Access Protocol for SpaceWire which provides a standard method of reading and writing to registers and memory within a SpaceWire unit by sending a command and where appropriate receiving a reply across the network.

1. Introduction

SpaceWire [1][2] is a communications network for use onboard spacecraft. It is designed to connect high data-rate sensors, large solid-state memories, processing units and the downlink telemetry subsystem providing an integrated onboard, data-handling network. SpaceWire links are serial, high-speed (2 Mbits/sec to 200 Mbits/sec or higher), bi-directional, full-duplex, point-to-point data links which connect together SpaceWire equipment. Application information is sent along a SpaceWire link in discrete packets. Control and time information can also be sent along SpaceWire links. SpaceWire is defined in the European Cooperation for Space Standardization ECSS-E50-12A standard [1]. It is being widely used on current space missions.

SpaceWire has been adopted for use on many space missions partly because of the ready availability of intellectual property (IP) cores, components, software drivers, and development support and test equipment. A SpaceWire CODEC designed by the University of Dundee and implemented in VHDL is available as an IP core from ESA for European space projects [3]. This CODEC is being designed into several SpaceWire chips. A SpaceWire router chip has been designed by University of Dundee and is currently being implemented in an Atmel radiation tolerant ASIC [4]. A wide range of development support and test equipment is available from STAR-Dundee Ltd [5] and other organizations.

SpaceWire has met the need for standardization of a high-speed communications network for spacecraft at the physical and data link levels, the SpaceWire working group is now focusing on higher-level protocols. The aim of these protocols is to support software and hardware reuse further up the communications protocol stack. The first protocol being defined by the SpaceWire working group is a relatively low-level service called the remote memory access protocol (RMAP). The remote memory access protocol (RMAP) is used to write to and read from memory or registers in a destination node on a SpaceWire network. Input/output registers and control/status registers are assumed to be memory mapped so are accessed as memory. This paper describes RMAP.

2. Overview of Remote Memory Access Protocol

The remote memory access protocol (RMAP) provides a means for one SpaceWire node to write to and read from memory inside another SpaceWire node. The aim of the RMAP protocol is to standardize the way in which SpaceWire units are configured and to provide a low-level mechanism for the transfer of data between two SpaceWire nodes. For example RMAP may be used to configure a camera or a mass memory device. The camera device may then write image data to allocated areas of memory in the mass memory, or the mass memory may read image data from the camera.

All read and write operations defined in the RMAP protocol are posted operations i.e. the source does not wait for an acknowledgement or reply to be received. This means that many reads and writes can be outstanding at any time.

Write command

Write commands can be acknowledged or not acknowledged by the destination node when they have been received correctly. If the write is to be acknowledged and there is an error with the write request, the destination will send an error code to the source that sent the command. The error can only be sent to the source if the write command header was received intact, so that the destination that detected the error knows where to send the error message.

Write commands can perform the write operation after verifying that the data has been transferred to the destination without error, or it can write the data without verification. To perform verification on the data requires buffering in the destination node to store the data while it is being verified, before it is written. The amount of buffering is likely to be limited so verified writes ought only to be performed for relatively short sets of data that will fit in the available buffer at the destination. Longer writes can be performed but without verification prior to writing. Verification in this case is done after the data has been written. Verified writes should always be used when writing to configuration or control registers. The acknowledged/non-acknowledged and verified/non-verified options to the write command result in four different write operations.

Read command

The read command reads one or more bytes of data from a specified area of memory in a destination node. The data read is returned in a reply packet.

Read-modify-write command

The read-modify-write command reads a register (or memory) returning its value and then writes a new value, specified in the command, to the register. A mask can be included, in the command, so that only certain bits of the register are written. This provides an atomic operation that can be used for semaphores and other handshaking operations.

3. Write Command

The write command provides a means for one node, the source node, to write one or more bytes of data into memory of another node, the destination node on a SpaceWire network. The operation of the write command is illustrated in the sequence diagram of Figure 1.

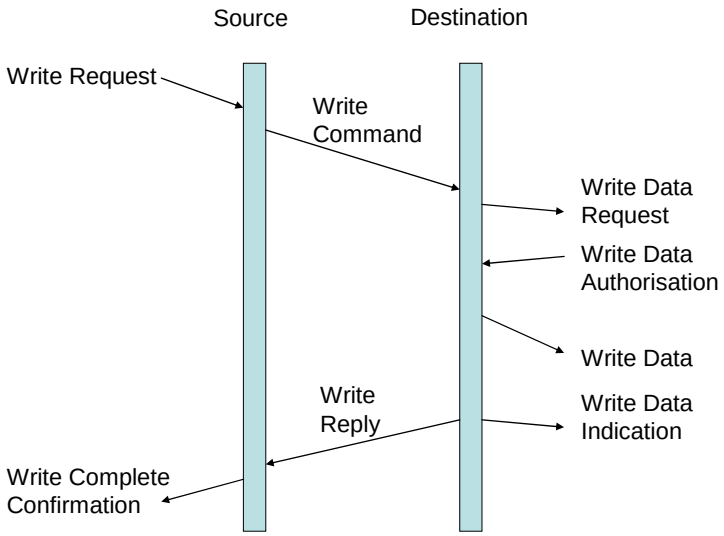

Figure 1: Write Command/Acknowledge Sequence

The write command sequence begins when an application requests to perform a write operation (Write Request). In response to this the source node builds the write command and sends it across the SpaceWire network to the destination node (Write Command). When the Write Command arrives at the destination, the header is first checked for errors and if there are no errors the user application at the destination node is asked if it will accept the write operation (Write Data Request). Assuming that authorization is given by the destination user application (Write Data Authorization) the data contained in the write command is written into the specified memory location of the destination node (Write Data). A bit in the command field of the write packet (Validate Data Before Write bit) may be set to cause the data in the write command to be buffered and checked using a data checksum before it is written to memory.

Once data has been written to memory the user application running on the destination node is informed that a write operation has taken place (Write Data Indication). If an acknowledgement has been requested, by setting an Ack/No_Ack bit in the command field, then the destination node will wait until the data has been written to memory in the destination node. It will then send a write reply packet back to the source of the write

command (Write Reply). When the write reply is received, the source node indicates successful completion of the write request (Write Complete Confirmation). If no acknowledgement is requested then the destination node waits for the data to be written into destination memory, but does not send an acknowledgement write reply to the source.

Note that the speed with which the destination user application responds to the Write Data Request with a Write Data Authorization will limit the rate at which RMAP commands can be processed by the destination node. The SpaceWire interface will block during this period. In some cases, for example writing to control or configuration registers, the Write Data Request and Write Data Indication are implicit in the actual write operation so there is no appreciable delay and one command can immediately follow the previous one.

The format of the write command is shown in Figure 2.

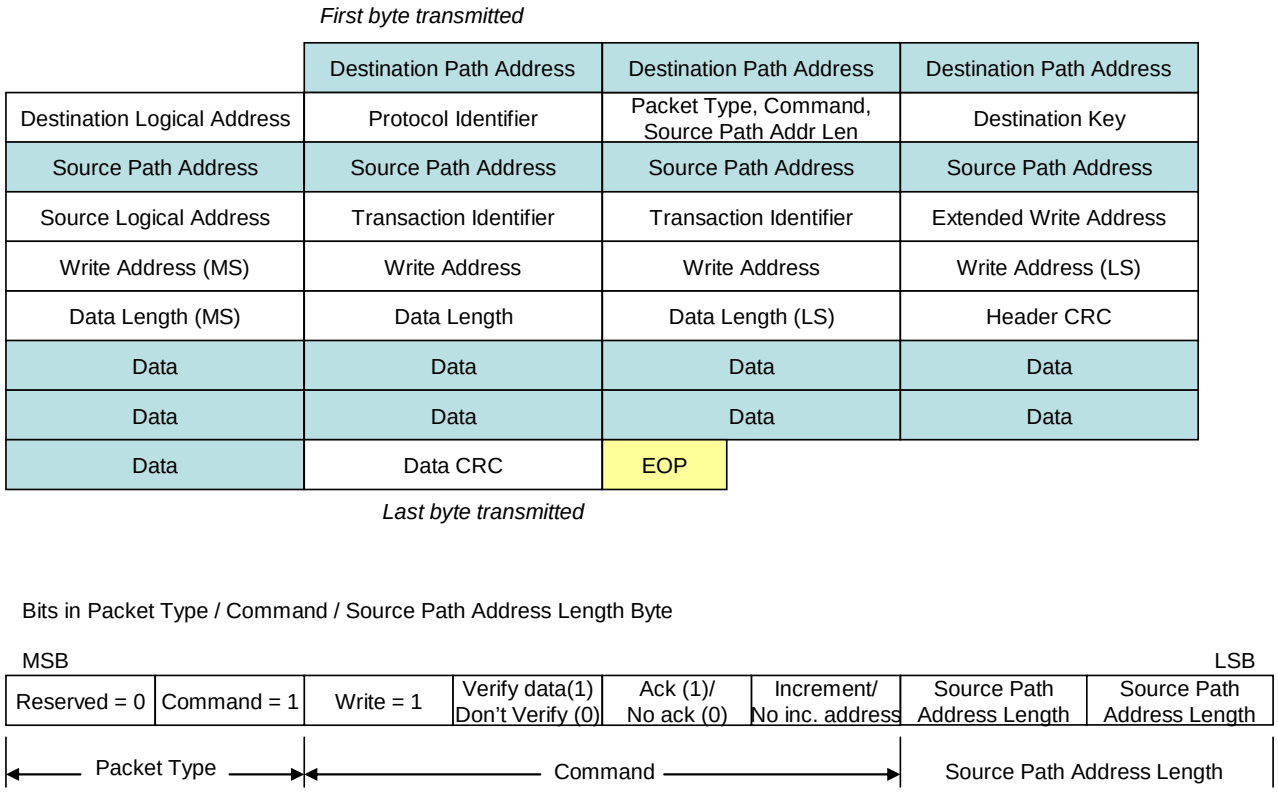

Figure 2: Write Command Format

Optional fields are shown shaded.

The destination path address is the SpaceWire path address to the node containing the register or memory to be written (destination node).

The destination logical address is the SpaceWire logical address of the destination node. This may be set to 254 as a default if only path addressing is being used to address the destination node.

The protocol identifier, set to 01h, indicates that this SpaceWire packet is an RMAP protocol data unit.

The packet type, command and source address length byte indicates the type of packet (command or reply), the command (read, write, read-modify write) and the number of 32-bit words used to contain a possible return path address to the source node.

The destination key is a value used to help ensure that the packet is intended for the node that it arrives at. The source and destination nodes agree in advance what the destination key should be. If the destination key received is not the expected value the command is rejected.

The source path address is an optional path address indicating the route back to the source node that sent the RMAP command.

The source logical address is the logical address of the source. This may be set to the default value of 254 if only source path addressing is being used.

The transaction identifier is used by the source to associate a response (reply or acknowledgement) with the particular command that caused the response. This allows posted read and write operations.

The extended write address is an extra 8-bits of address which may be used for memory bank selection etc.

The write address is a 32-bit address of the register or memory to be written to.

The data length is the amount of data to be written in bytes.

The header CRC is a check code used to confirm that the header is correct before any action is taken at the destination node.

The data contains the data to be written.

The data CRC is a check code used to confirm that the data is correct before writing or, for large amounts of data, that it was correct after being written.

EOP is the SpaceWire end of packet marker.

The reply to a write command is sent by the destination back to source of the write command. The reply is used to indicate the success or failure of the write command. The format of the write reply is shown in Figure 3.

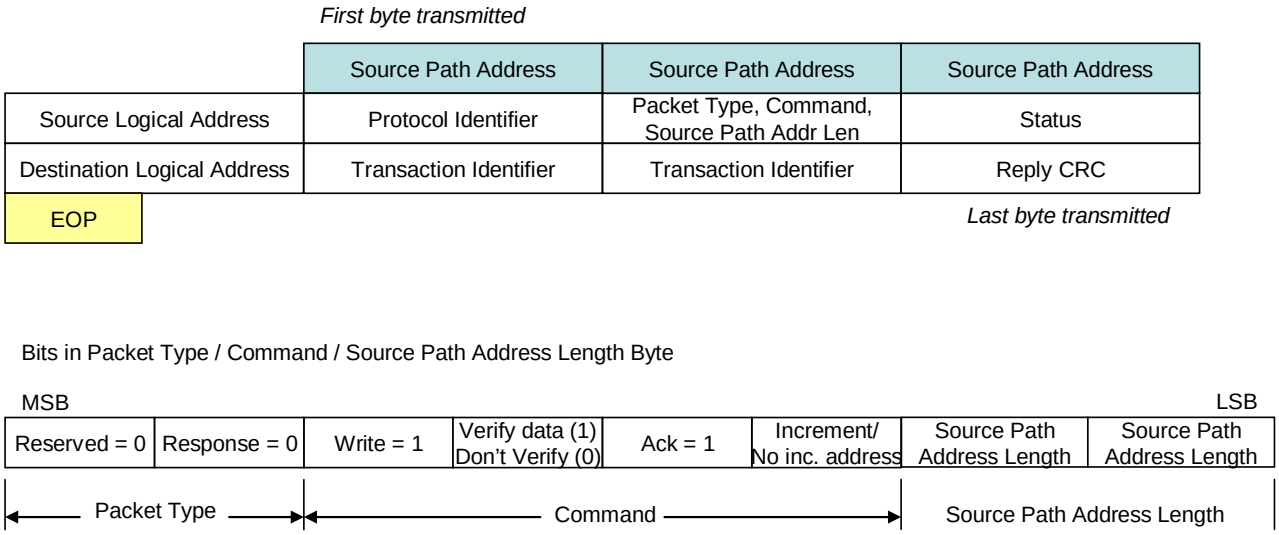

Figure 3: Write Reply Format

The fields in the reply have the same or similar meanings to the write command.

The status indicates any errors that have occurred and are able to be reported in the reply.

4. Read Command

The read command provides a means for one node, the source node, to read one or more bytes of data from the memory of a destination node. The operation of the read command is illustrated in the sequence diagram of Figure 4.

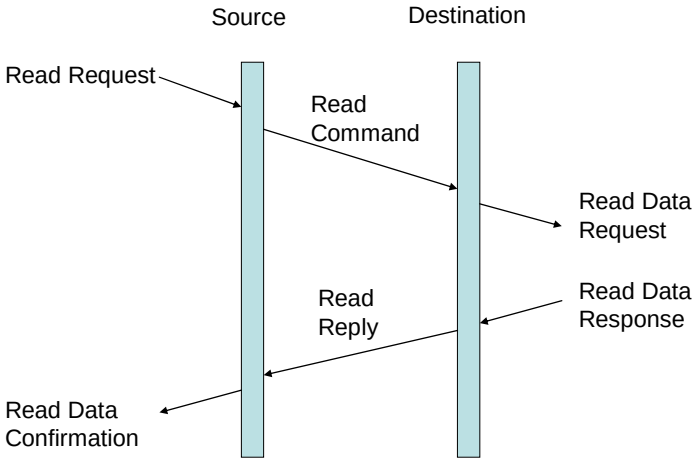

Figure 4: Read Command/Reply Sequence

The read command sequence starts when an application requests to perform a read operation (Read Request). The read command is constructed and sent to the destination node (Read Command). When the read command arrives at the destination it is flagged to the user application on the destination node (Read Data Request). The header of the read

reply packet is formed and the requested data appended to it. The read reply containing the data is then sent back to the source of the read command. When it arrives there the user application that requested the data is informed (Read Data Confirmation).

The format of the read command is shown in Figure 5.

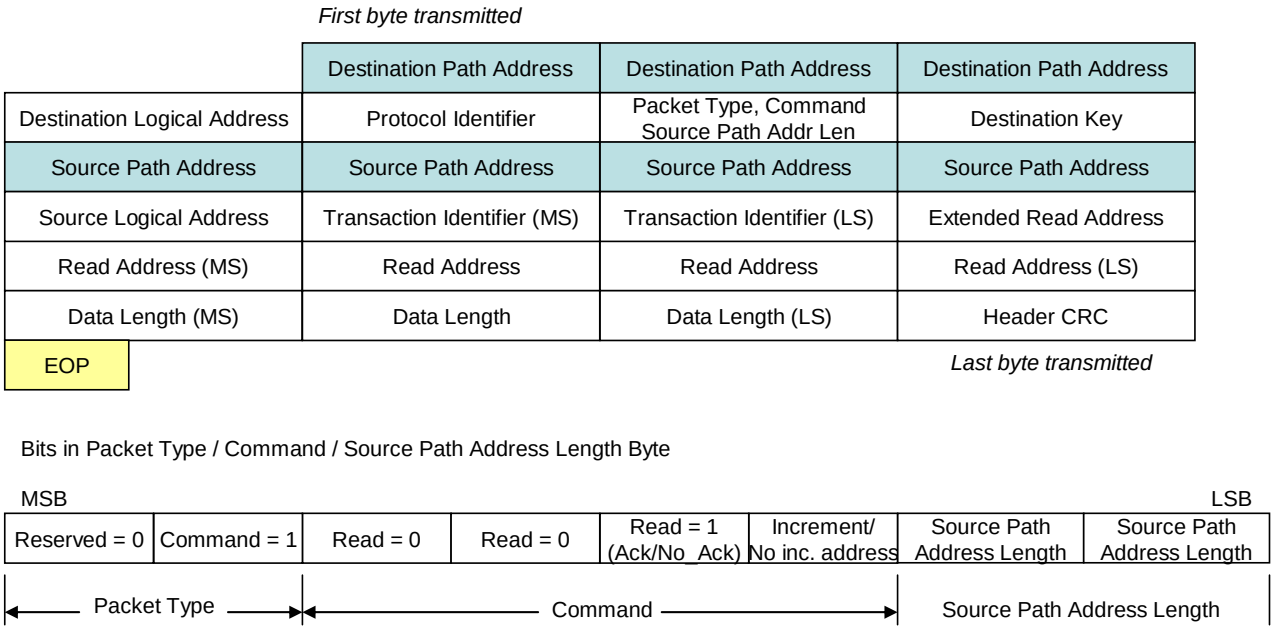

Figure 5: Read Command Format

The read reply contains either the data that was read from the destination node, or an error code indicating why data could not be read. The reply to a read command is sent by the destination node back to the source of the read command. The format of the read reply is illustrated in Figure 6.

The fields in the read command are similar or the same as those in the write command.

The read address is the register or memory address that reading is to start from.

The data length indicates the amount of data to be read in bytes.

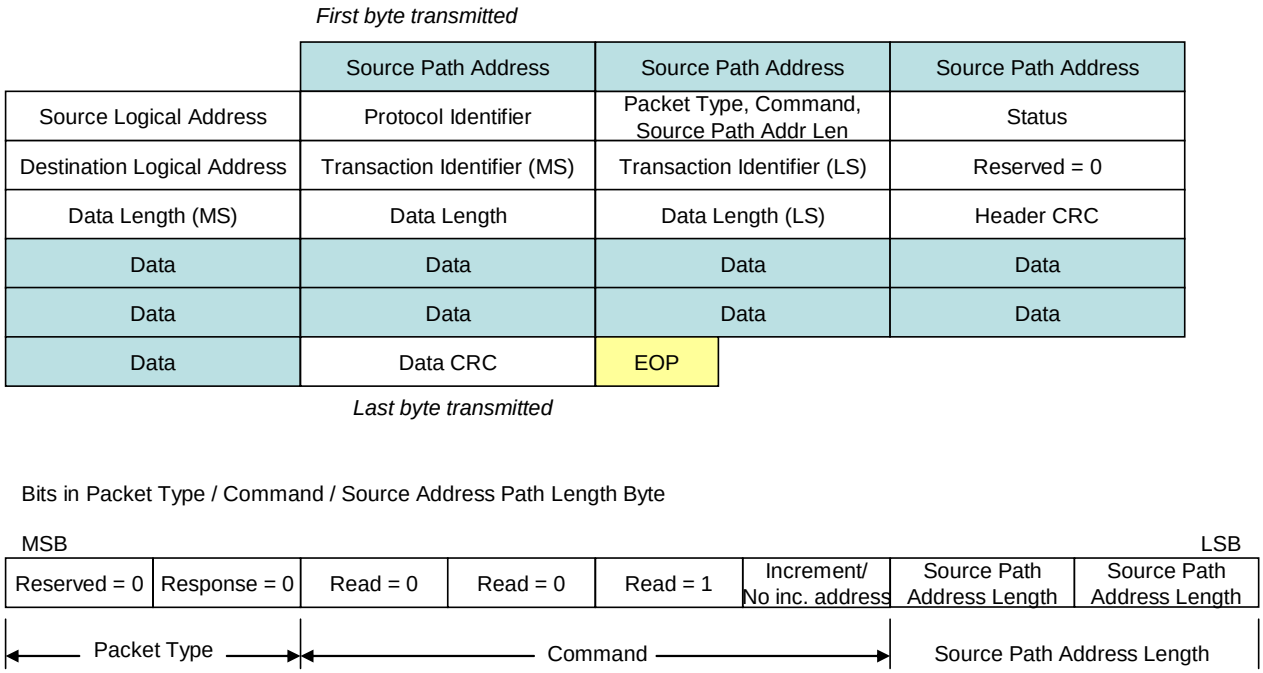

Figure 6: Read Reply Format

The fields in the read reply are again similar to the previous command and reply fields.

The data field contains the data that was read from the destination node.

5. Conclusions and Future Work

The Remote Memory Access Protocol provides a valuable addition to the SpaceWire standard defining a standard method of reading and writing to registers and memory across a SpaceWire network. Standardisation of these functions allows software and hardware implementations of RMAP to be used on many missions reducing the cost of development. RMAP is currently being implemented in the configuration port of the ESA SpaceWire router chip. It is to be included in the ESA Remote Terminal Computer chip and will be developed as a standard VHDL core available from ESA for ESA space missions.

At the moment of writing the RMAP standard is in the last stages of technical definition. It will then be drafted and issued as an ECSS standard document.

The authors would like to acknowledge the support of ESA for the work done at University of Dundee on the RMAP protocol and the valuable input from the members of the SpaceWire working group in defining the RMAP standard.

6. References

1. European Cooperation for Space Standardization, Standard ECSS-E-50-12A, “SpaceWire, Links, Nodes, Routers and Networks", Issue 1, European Cooperation for Space Data Standardization, February 2003.

2. Rosello, J., “SpaceWire Web Page”, European Space Agency, http://www.estec.esa.nl/tech/spacewire/.

3. C. McClements, S.M. Parkes, and A. Leon, “The SpaceWire CODEC,” International SpaceWire Seminar, ESTEC Noordwijk, The Netherlands, November 2003.

4. S.M. Parkes, C. McClements, G. Kempf, S. Fischer and A. Leon, “SpaceWire Router,” International SpaceWire Seminar, ESTEC Noordwijk, The Netherlands, November 2003.

5. S.M. Parkes, C. McClements, I. Martin, S. Mills, R. Manston, “SpaceWire Development and Test Equipment,” International SpaceWire Seminar, ESTEC Noordwijk, The Netherlands, November 2003.