SPACEWIRE RMAP IP CORE

Session: SpaceWire Components

Long Paper

Chris McClements

STAR-Dundee, c/o School of Computing, University of Dundee, Dundee, Scotland, UK

Steve Parkes

University of Dundee, School of Computing, Dundee, Scotland, UK

E-mail: chris@star-dundee.com, sparkes@computing.dundee.ac.uk

ABSTRACT

The SpaceWire RMAP core is a VHDL IP core implementing the Remote Memory Access Protocol (RMAP). RMAP has been defined by the SpaceWire Working Group and standardised by the European Cooperation for Space Standarization (ECSS) as standard number ECCS-E-ST-50-52C. RMAP allows devices to read and write from memory spaces in a standard way, increasing device interoperability and reducing development time. The SpaceWire RMAP IP core provides a well tested, easy to use core for systems that need RMAP capability.

1 INTRODUCTION

The RMAP IP core was developed by the University of Dundee under contract from the European Space Agency (ESA). It is available from ESA for use on European space missions or projects and available from STAR-Dundee for other applications. The core is highly configurable and can be used as an RMAP target or initiator. The core can be implemented in a number of technologies, including various radiation tolerant FPGAs.

2 BACKGROUND

The Remote Memory Access Protocol (RMAP) standard is now available from the ECSS website as ECCS-E-ST-50-52C [1]. The RMAP standard was written by the University of Dundee with support from members of the SpaceWire Working Group. RMAP is a SpaceWire protocol that provides a standard mechanism for reading from, and writing, to memory in a remote SpaceWire node. The RMAP protocol has already been designed into the ESA SpW-10X router ASIC [2] and into missions like Bepi-Colombo [3][4], MMS [5], and ExoMars rover [6].

3 RMAP IP CORE OVERVIEW

3.1 FUNCTION

There are two main functions which can be selected by configuration at synthesis time. The first type is referred to as the "initiator" RMAP Interface which sends out RMAP commands and receives replies. The second type is the "target" RMAP Interface which receives RMAP commands, executes them and sends out any required replies. The RMAP core has a wrapper which connects to the ESA/Dundee SpaceWire-b core [7] [8] which handles the SpaceWire point-to-point link protocol. The RMAP core target and initiator functions are illustrated in Figure 1. The core is shown in three configurations; target only, initiator only or both.

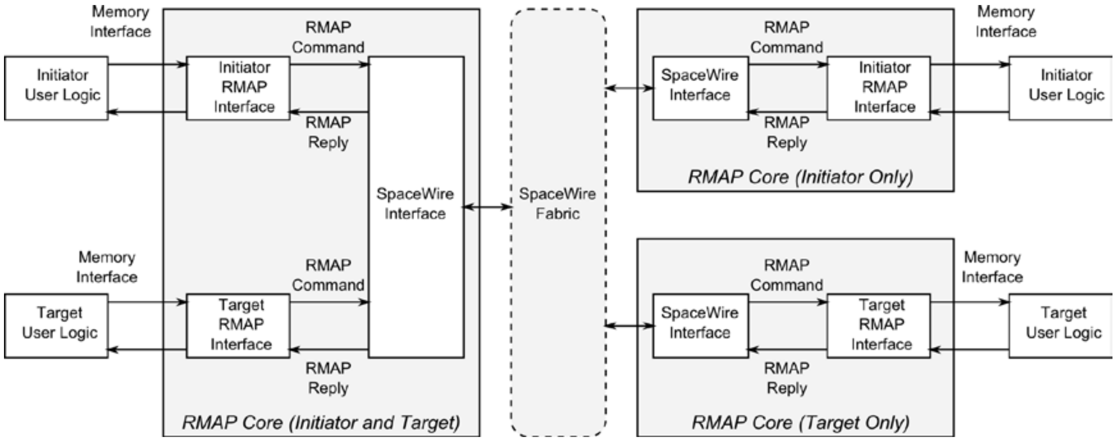

Figure 1: RMAP IP core Data Flow

RMAP commands are initiated in the initiator user logic, encoded as RMAP packets in the "Initiator RMAP Interface", sent over the SpaceWire link as an RMAP packet, decoded by the "Target RMAP Interface" and data or information is passed to the target user logic after authorisation of the command. The "Target RMAP Interface" formats an RMAP reply packet which is sent over the SpaceWire interface, decoded by the "Initiator RMAP Interface" and the reply data/status information is passed to the initiator user logic.

3.2 ARCHITECTURE

The architecture of the RMAP core is illustrated in Figure 2. The SpaceWire CODEC implements the SpaceWire serial point to point protocol, ECSS-E-ST-50-12C [9], and provides FIFO ports to the Protocol Input and Output blocks. The Protocol Input and Output blocks determine the destination of packets dependent on the packet header. The protocol handlers can bypass data from the RMAP core if the protocol identifier does not identify the packet as an RMAP packet.

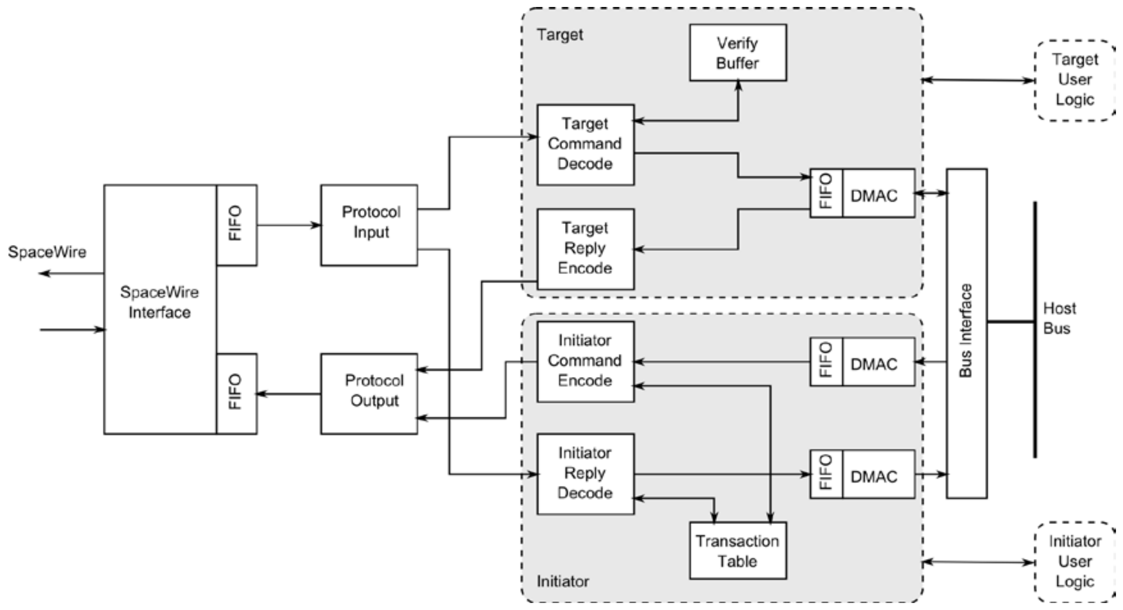

Figure 2: RMAP IP core Architecture

The Target RMAP Interface decodes RMAP command packets, reads or writes data from the host bus and returns RMAP reply packets. If the RMAP command is a verified write command the target writes the command data to the verify buffer before it is transferred to host memory. The Initiator RMAP Interface accepts commands into the initiator transaction table, encodes RMAP command packets, decodes reply packets and outputs status information. The target and initiator interact with the user memory space using DMA controllers.

3.3 MEMORY INTERFACE

The RMAP controller interface to memory is modelled on the AMBA AHB bus standard which provides a pipelined control/data bus transfer model. Data is transferred to and from the bus in bursts using internal burst FIFOs in the RMAP core. The bus can be configured for different bus size widths, byte order and bit swapping operations.

3.4 CONFIGURATION

The RMAP core is configured using VHDL generics. The configuration options of the core include the ability to implement target only logic, initiator only logic, or both target and initiator; configuration of the host bus width, the burst transfer depth and the byte/bit ordering of the RMAP packet data; watchdog timer on bus transfers; maximum number of outstanding initiator commands and therefore the initiator transaction table size; configuration of the internal FIFO sizes and the target verify buffer size.

4 USING THE RMAP CORE

4.1 TARGET

The target command logic is responsible for decoding RMAP command packets and executing the specified command, e.g. write. RMAP command headers are checked for validity and the set of RMAP command authorisation parameters are passed to the host for authorisation. The host can check the memory address, command and other parameters, and decide to authorise or discard the command. When the RMAP command is a write command, and authorisation has been given by the host, data is placed in user memory by the target DMA controller.

The target reply logic is responsible for sending RMAP reply packets with the status of commands and additional data when a read command is performed. The status is dependent on the validity of the RMAP command packet and the authorisation response of the host. Reply data is read from the host user memory by the DMA controller and sent in the RMAP reply packet.

4.2 INITIATOR

The RMAP Initiator Handler uses several memory structures inside the RMAP core and inside initiator user memory. The structures are used to control the passing of commands from initiator user memory to the RMAP core and the passing of replies from the RMAP core to initiator user memory. The RMAP initiator data structures and data flow is depicted in Figure 3.

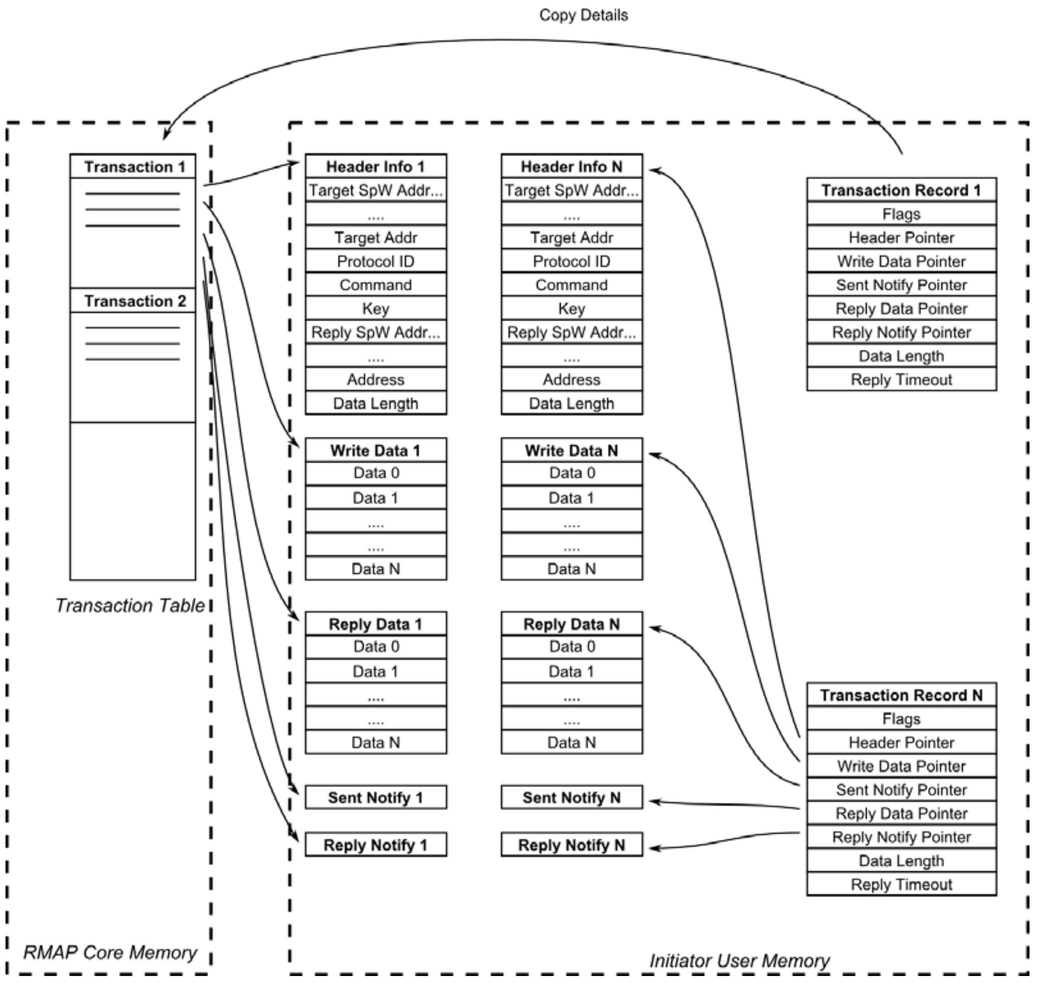

Figure 3: Initiator Data Structures

In the initiator user memory there are four possible memory areas or buffers associated with each RMAP command: transaction details record, header information, write data, and reply data.

The transaction details array holds the following information: a pointer to the command header in user memory, a pointer to any data to be sent with a write or read-modify-write command, a pointer to the memory location for sent notification, a pointer to space for a reply to a read or read-modify-write command, a pointer to the memory location for reply notification, the length of data to be read or written and the reply time-out value.

The header information buffer holds the RMAP command header information including the target SpaceWire address and the reply address. The write data buffer holds any data to be sent with a write or read-modify-write command. The reply data buffer is reserved space into which any data associated with a read or read-modify-write command will be written. The length of the buffer is given by the data length field and will not be overwritten by the core, even when a read reply packet is received with more data than is in the buffer or the RMAP header record data length field is greater than the transaction record data length.

4.3 SENDING A COMMAND

To send an RMAP command the host sets up the header of the command in a header information buffer, any data to be sent with the command in a write data buffer and space for any reply in a reply data buffer. The user then creates a transaction record with pointers to the header information buffer, write data buffer and reply data buffer along with information about the amount of data in these buffers. It also provides pointers to memory locations (or registers) where sent and reply notifications are to be made. Finally it adds into the transaction record a reply time-out value which is set in micro seconds or can also be infinite. Once the transaction record is complete the initiator user application informs the RMAP core that is has an RMAP command to send and passes the RMAP core a pointer to the corresponding transaction record.

If the transaction details record flags field indicates that the command is expecting a reply the command is not started (sent) until there is room for another transaction in its outstanding transaction array. The RMAP core will then send the command by copying the header information from user memory to the SpaceWire interface, adding any detail necessary (e.g. header CRC). The header information checked for errors before sending begins. Any errors which are detected in the header are recorded and output on the status interface and to the notify sent register, if used.

If there is any write data to be sent this will be copied from the write data buffer in user memory to the SpaceWire interface and appending the data CRC. Finally an EOP marker will be added to complete the packet. The initiator user application will be informed that the command has been sent by the RMAP core writing the transaction ID and status to the memory location specified by the sent notify pointer in the transaction details array element.

4.4 RECEIVING A REPLY

When an RMAP reply is received the core searches the outstanding transaction array for an entry with a transaction identifier that matches the transaction identifier of the reply. Assuming there is a match the core then writes any data from a read or read-modify-write reply to the user memory location specified by the reply data pointer for the corresponding entry in the transaction details array. The RMAP core writes the transaction identifier and status to the memory location specified by the reply notification pointer in the transaction details array entry. When this has been done the relevant entry in the outstanding transaction array is cleared freeing it for use by another RMAP transaction. The core can generate transaction Ids automatically to avoid the change that the user logic may use duplicate identifiers.

4.5 TRANSACTION DETAILS RECORD

The transaction details record is initialised in user memory by the host application when it wishes to send an RMAP command. The format of the transaction details record is illustrated in Table 1. The flags field is a bit mask which holds properties of the transaction record such as notify on send, wait forever, etc.

	31	23	15	7 0
0	Unused		Flags	
1	Header Pointer			
2	Write Data Pointer			
3	Sent Notify Pointer			
4	Reply Data Pointer			
5	Reply Notify Pointer			
6	Unused	Data Length		
7	Reply Timeout			

Table 1: Transaction Details Record Memory Setup

4.6 HEADER INFORMATION RECORD

The header information record holds information on the RMAP command parameters to be sent. An example header information record is stored in memory as defined by an example header in Table 2. In the example there are four target SpaceWire addresses and one block of reply SpaceWire addresses.

	31	23	15	7 0
0	Target Path Address 1	Target Path Address 2	Target Path Address 3	Target Path Address 4
1	Target Address	Protocol ID	Instruction	Key
2	Reply Path Address 1	Reply Path Address 2	Reply Path Address 3	Reply Path Address 4
3	Initiator Address	Transaction ID 1	Transaction TID 0	Extended Address
4	Address 3	Address 2	Address 1	Address 0
5	Data Length 2	Data Length 1	Data Length 0	Unused

Table 2: Header Information Record Setup

5 IP VALIDATION

Verification of the RMAP core is performed using an automated VHDL test-bench which runs a series of test command scripts to check the function of the RMAP core.

The command scripts run test-cases which detect correct and incorrect behaviour of the configured RMAP core relative to the functional specification.

VHDL code coverage using the Modelsim simulator code coverage option in Modelsim SE is used to check for coverage of the complete design by the test cases. The purpose of the test cases is to show the function of the UUT is equivalent to the specification defined in the functional specification document.

6 SYNTHESIS

The RMAP core has one system clock input which clocks all flip-flops in the design except the receive clock domain of the SpaceWire link. The SpaceWire transmitter can also be clocked from a separate clock input on the core dependent on the SpaceWire link configuration settings.

A typical way to implement this RMAP core design is to run the system clock at the byte rate of the system and use a separate transmit clock to transmit the bytes at the required bit rate. For example a system which processes RMAP data at 20 Mbytes/s requires a 100 MHz transmit clock to transmit the byte data at 200 Mbps DDR, taking into account the SpaceWire data character length of 10 bits.

6.1 SYNTHESIS RESULTS

The results of synthesis runs on the Mentor Graphics Precision synthesiser are given below.

Model	AX2000			Spartan3E 1600	ProASIC3E1500
	FF	Comb	Modules	Slices	Tiles
Target Only with SpW	1425	2962	4464 (13.84%)	1134 (7.69%)	4576 (11.92%)
Initiator Only with SpW	2029	4434	6463 (20.04%)	2213 (15.00%)	7987 (20.80%)
Target and Initiator no SpW	2599	5634	8233 (26.20%)	2584 (17.52%)	10206 (26.58%)
Target and Initiator with SpW	2957	6249	9206 (29.06%)	3095 (20.97%)	11261 (29.33%)

Table 3: Area usage of RMAP core

6.2 SEU PROTECTION

It is expected that the fabric of the FPGA or ASIC technology will provide SEU protection for synchronous elements in the design (flip-flops). Typically memory blocks are not protected in silicon, therefore they should either be implemented as flip-flops, or a drop in replacement for the single and dual clocked memory blocks should be used in the final synthesised model. For example, memory blocks with error detection and correction (EDAC) using error correcting codes (ECC) are provided with the Actel Libero and designer tool chain. Critical memory blocks for SEU protection in the design are the verify buffer, transaction table and DMA controller FIFOs.

The SpaceWire interface transmit and receive FIFOs are also critical but as the RMAP protocol is used, the packet data is protected by header and data CRCs. In this case SEU protection may not be required.

7 CONCLUSION

The RMAP IP core was developed in the frame of the SpaceNet activity. This resulted in a SpaceWire interface VHDL core that includes the RMAP protocol extension to SpaceWire, which will enable users to readily implement the RMAP protocols in FPGAs or ASICs.

It is available from ESA for use on European space missions or projects and available from STAR-Dundee for other applications. The core is designed to be a highly configurable VHDL IP core which can be used as an RMAP target or initiator. The core can be implemented in a number of technologies, including the radiation tolerant Actel RTAX which is widely used in the Space industry.

8 REFERENCES

1. ECSS, “SpaceWire - Remote memory access protocol”, ECSS-E-ST-50-52C, Feb 2010, available from http://www.ecss.nl

2. S. Parkes, C. McClements, G. Kempf, S. Fishcher, P. Fabry, A. Leon, “SpaceWire Router ASIC”, International SpaceWire Conference, Dundee, 2007.

3. T. Takashima, H. Hayakawa, H. Ogawa, Y. Kasaba, M. Koyama, K. Masukawa, M. Kawasaki, S. Ishii, Y. Kuroda, BepiColombo MMO Project Data-Handling Team, “Introduction of SpaceWire Applications for the MMO Spacecraft in BepiColombo Mission”, International SpaceWire Conference, Dundee, 2007.

4. T. Yuasa, K. Nakazawa, K. Makishima, H. Odaka, M. Kokubun, T. Takashima, T. Takahashi, M. Nomachi, I. Fujishiro, F. Hodoshima, “Development of a SpW/RMAP-based Data Acquisition Framework for Scientific Detector Applications, International SpaceWire Conference, Dundee, 2007.

5. ESA, “ESA – Aurora Programme – ExoMars”, http://www.esa.int/SPECIALS/Aurora/SEM1NVZKQAD_0.html

6. Soutwest Research Institute, “MMS-SMART Home Page”, http://mms.space.swri.edu/

7. ESA, “Micro-electronics Website “, http://www.esa.int/TEC/Microelectronics/SEMLOU8L6VE_0.html”, ESA, 2010.

8. STAR-Dundee website: www.star-dundee.com

9. ECSS, “SpaceWire: Links, nodes, routers and networks”, ECSS-E-ST-50-12C, July 2008.