High performance SpaceWire RMAP/DMA engine for the CASTOR microprocessor

SpaceWire Components, Short Paper

Chris McClements, Steve Parkes, Albert Ferrer,
Alberto Gonzalez-Villafranca
STAR-Dundee Ltd
Dundee, UK
chris.mcclements@star-dundee.com, steve.parkes@star-dundee.com

***Abstract*— CASTOR is a new radiation tolerant SPARC V8 processor chip which is currently being developed by Atmel in partnership with STAR-Dundee. The chip is implemented on a 90 nm radiation tolerant process which will deliver an expected processor clock speed of 200 MHz. The CASTOR chip is targeted at data processing and instrument control applications, and will deliver functional improvements over previous SPARC processors. The chip has eight SpaceWire interfaces running at 200 MBits/s, a CAN bus interface and IEEE 1553 bus interface.**

At the core of the CASTOR chip is a number of dedicated high performance SpaceWire Remote Memory Access Protocol (RMAP) and Direct Memory Access (DMA) engine's connected to the SpaceWire interfaces through a SpaceWire router. Each SpaceWire engine is capable of acting as an RMAP target, RMAP initiator or as a general purpose SpaceWire packet transmitter and receiver between the SpaceWire network and packet data defined in internal memory. Dedicated SpaceWire DMA channels are used to ensure software involvement in SpaceWire packet generation and reception is kept to a minimum. The SpaceWire interfaces support the SpaceWire-D protocol used for guaranteed latency and deterministic packet delivery. In conjunction with the RMAP initiator the chip can rapidly be configured as a highly capable SpaceWire-D initiator.

The chip can act as an RMAP target, initiator or both. The RMAP target provides a mechanism to allow remote access to the internal memory space. Two modes of operation are supported to allow direct access to a pre-defined area of memory or controlled access using authorisation by software. The RMAP initiator uses information stored in internal memory by the application software to access remote memory in equipment connected to the SpaceWire network. The engine is capable of initiating a number of RMAP transfers from remote memory, either writing data from internal memory to a remote memory location or receiving data from a remote memory location and writing it to internal memory, then interrupting the host when all transactions are complete.

The DMA channels allow the application software to send and receive data packets using data structures defined in internal memory. Each SpaceWire engine has a number of DMA channels which can operate independently of each other.

***Index Terms*—SpaceWire, CASTOR, RMAP, Sparc V8**

I. Introduction

An onboard SpaceWire [1] system comprises a number of SpaceWire nodes and routers connected together through high speed serial links. The nodes on the SpaceWire network can be sensors, mass memories and processing units. CASTOR is a new radiation tolerant SPARC V8 processor chip which is currently being developed by Atmel in partnership with STAR-Dundee.

The CASTOR chip has eight SpaceWire interfaces to facilitate communication over the SpaceWire network. The application software running on the processor has access to a number of dedicated high performance SpaceWire Remote Memory Access Protocol (RMAP) [2] and Direct Memory Access (DMA) engines to provide RMAP and application specific packet generation and reception without excessive processor workload.

II. Features

The CASTOR chip has dedicated RMAP target and initiator hardware which offloads RMAP packet generation and checking from the processor. The target can be configured to allow a remote unit to read and write memory locations inside the processor memory space without interrupting the host software. The initiator facilitates access to remote memory spaces through RMAP protocol commands and offloads multiple transaction generation and reply packet checking from the processor.

A multi-channel DMA packet transmission and reception controller is available to the processor to send and receive data through a SpaceWire router. The DMA channels are optimised to support high throughput of SpaceWire packets with minimal interruption of the processor. Generation and checking of CRC-8 and CRC-16 checksums are supported by the DMA channels.

Packets are routed to the SpaceWire network through an eight port SpaceWire router. This allows the CASTOR chip to connect too many peripherals and also act as a routing device.

Protocol support is provided for the SpaceWire-D deterministic data delivery protocol [3], the SpaceWire plug and play protocol [4], multiple time-code counters and distributed interrupt time-codes [5].

III. System Architecture

The system architecture is defined in Fig. 1.

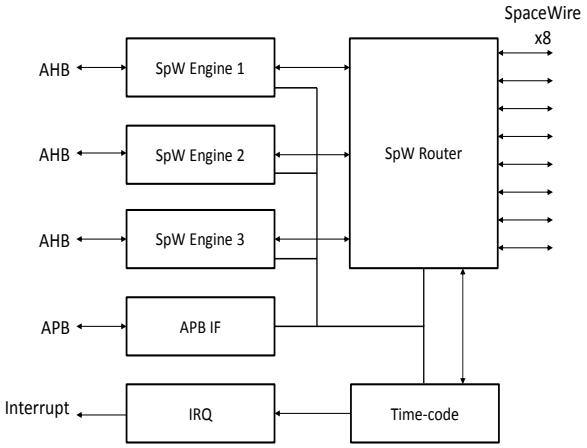

Fig. 1. System architecture

The SpaceWire engines contain an RMAP target [6], an RMAP initiator and a multi-channel DMA controller. Each engine facilitates packet generation and checking of RMAP and DMA transfers between the processor and the SpaceWire router, offloading the processor for other tasks. The SpaceWire router [7] has 8 SpaceWire ports running at 200 MBps and 3 internal FIFO ports for connection to the engines. The routers internal configuration port, port 0, facilitates configuration of the internal registers through RMAP or Plug and Play. The APB interface is used to configure and read status registers from SpaceWire engines, time-code controller and SpaceWire router. The interrupt controller provides event notification to the host processor for packet, time-code and error events which occur. The time-code controller implements time-code forwarding and distributed interrupt forwarding.

IV. SpaceWire Engine

The CASTOR chip has three SpaceWire engines which can act as an RMAP target, an RMAP initiator and to transmit and receive data from internal memory through a multi-channel DMA controller. The engine performs memory accesses through an AHB master interface and is configured through an APB interface.

The SpaceWire engine architecture is shown in Fig 2. The engine is comprised of a protocol multiplexer which connects to the SpaceWire router, an RMAP target, an RMAP initiator, a multi-channel DMA controller, an AHB interface and an APB interface.

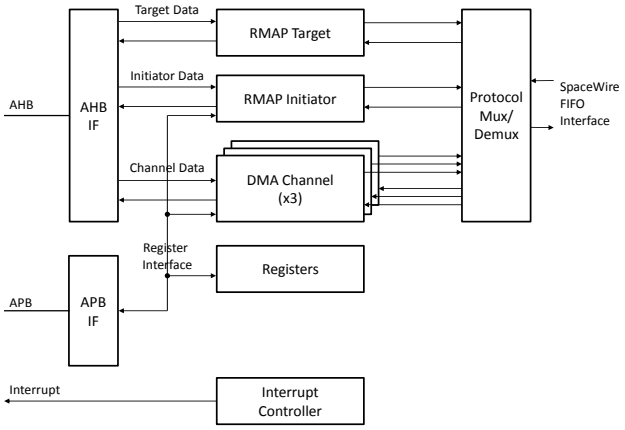

Fig. 2. SpaceWire engine

A. Protocol Multiplexer

When sending, packets to the SpaceWire Router, the multiplexer selects the next packet to be sent and waits for the end of packet before selecting the next packet to be transmitted.

When receiving, packets from the SpaceWire Router, the protocol de-multiplexer checks the first four packet bytes against a configurable pattern and mask to determine the destination of the packet, either RMAP target, RMAP initiator or a specific DMA channel. The pattern and mask are programmable by the host processor through the APB registers.

The protocol multiplexer allows multiple destination nodes or multiple protocols to be handled by the DMA channels. A packet received at a node which conforms to the ECSS-E-ST-50-51C [8] standard will have a leading logical address byte and a protocol identifier byte, followed by the packet cargo bytes and an end of packet. The protocol multiplexer transfers data packets from the RMAP target, initiator and the DMA channels into the SpaceWire FIFO. Arbitration is performed between the channels using a fair arbitration scheme where each packet source takes it in turn to transmit packets.

B. RMAP target

The RMAP target accepts RMAP commands from a remote system, performs read and write memory access commands over the AHB bus to system memory and returns an optional reply packet to the remote system. The target supports all RMAP commands with the option of limiting the commands which can be performed by configuration from software. A 16 byte verified write buffer is provided to support verified write commands.

An RMAP command received by the target is required to be authorised before it can access system memory. The processor can configure the RMAP target to act in two modes of operation.

The first mode requires the host processor to authorise commands through the APB register interface. Authorisation is requested using the Interrupt output of the core. The host software should read all the authorisation fields and then decide if the command is valid by authorising the command

through the RMAP target command register. When the target has completed the RMAP command it will interrupt the host processor again with the notification status.

In the second operation mode the host processor sets which RMAP operations are authorised and the address range in which RMAP commands can operate. Any command which is performed outside of the address range or other authorisation fields is not authorised and recorded as an error.

C. RMAP initiator

The initiator uses the RMAP protocol to write data from system memory to a remote system, or read data from a remote system and place it in a pre-defined area of memory. The initiator can be used by the processor to collect data from remote targets into system memory and check the data received. The initiator uses RMAP transaction specific data structures in memory to control the command type and command fields which will be used to generate the RMAP packet. A transaction table is stored in memory to facilitate the transmission of multiple command packets before the replies for those commands have been received. The initiator validates all reply packet fields against the expected fields stored in the transaction table. If an error occurs the error is recorded and the reply packet is not acted upon.

Before the initiator can be used to send RMAP commands it must be given space in system memory to store outstanding transactions. An outstanding transaction is required to tell the initiator where in memory it should store reply data and notification status

The initiator is split into three separate entities: the encoder, decoder and timeout checker. Each of the initiator entities can operate in a different mode. The encoder and decoder have three modes of operation: notification mode, list mode and watchdog mode (modes 1, 2 and 3). The timeout checker has two modes of operation: notification mode and passive mode.

Encoder/Decoder modes:

In mode 1, notification mode, the initiator waits for the host software to respond to each initiator command sent and reply received before continuing. This mode is suitable for hosts which wish to know when commands are sent or received and process the command data and status immediately.

In mode 2, command list mode, the initiator can send a number of commands or receive a number of replies before the host software is notified. The status for each command and reply is stored in a transaction defined notification area of memory. The host can check the command/reply status after the command list has been completed.

In mode 3, watchdog mode, the initiator can send a number of commands and receive a number of replies while the host is waiting for a timer to expire or another interrupt/event to occur. The host uses the timer, or other interrupt/event, to check if the commands have completed and the status of each command. This mode is useful when the host needs to know if the commands have been sent within a defined time period but does not need to check the operation status until the time period has expired.

The initiator implements an optional timeout counter for each outstanding transaction. When a reply is not received within the timeout period the transaction will be discarded and an error recorded.

Timeout checker modes:

In mode 1, notification mode, a transaction which times out, reply not received within the selected timeout period, will cause the notification bit in the status register to be set. The notification bit is acknowledged by the host software before the initiator can perform any further operations.

In mode 2, passive mode, a transaction which times out will be deleted from the initiator table and no notification will be generated. The timeout status will be recorded in the transaction defined notification area of memory.

D. Transmitting packets using the DMA channel transmitter

The DMA controller supports multiple concurrent TX channels which can be programmed to send one or multiple SpaceWire packets continuously. Channels can be disabled and enabled at any time, affecting the data rate of the corresponding channel without producing data loss. This allows a simpler implementation of MAC algorithms by software.

A packet consists of one or multiple data chunks stored in different memory locations. This allows the packet header to be stored in a different location that the packet data content. Sending of PUS [9] packets is supported by providing the hardware computation of its CRC-16. Continuous transmission of packets is provided using circular buffer architecture with data and packet descriptor pointers. Interrupts can be set to monitor the progress of transmission of packets without halting the actual operation. This makes it possible to achieve the maximum SpaceWire data rate with minimum CPU utilization. Errors in one channel do not affect the operation of other channels.

E. Receiving packets using the DMA channel receivers

Each channel can be associated to a different packet type or protocol using a packet filter based on the first four bytes of the header. Packets which are received on the same DMA channel are stored contiguously in memory and their packet length is stored in packet descriptors. Reception of RMAP packets is supported by providing the hardware computation of its CRC-8. Reception of PUS packets is supported by providing the hardware computation of its CRC-16. Continuous reception of packets is provided using circular buffer architecture with data and packet descriptor pointers. It is possible to enforce that a packet is not split at the end of the memory region. Interrupts can be set to monitor the progress of packets received without halting the actual operation. The user application or the SW driver should free the space used by packets already processed. This procedure allows data to be received at the maximum SpaceWire data rate with minimum CPU utilization. When an error occurs the reception is halted and the system is interrupted.

V. SpaceWire Router

The SpaceWire router has eight SpaceWire interfaces, three external port interfaces and an internal configuration port which supports the RMAP protocol. The internal configuration and status registers are also accessible through an APB interface. A control register is used to determine if the router is controlled through the configuration port or through the APB interface. Configuration by both masters at the same time is not supported although reading the status information from both masters at the same time is supported.

The SpaceWire router architecture is illustrated in Fig 3.

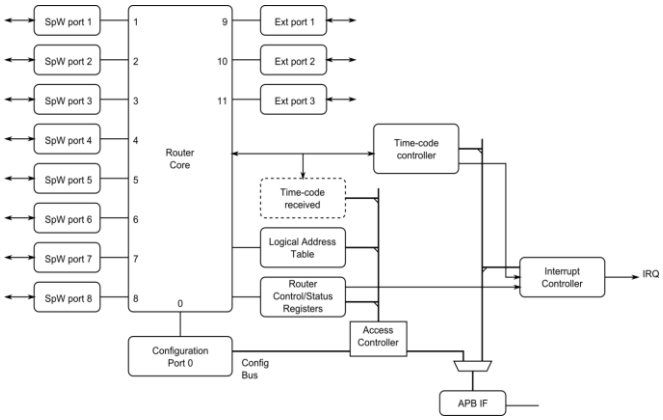

Fig. 3. SpaceWire router architecutre

VI. Time-code Controller

The SpaceWire time-code controller has functions to forward time-codes dependent on the time-code flags or to generate time-codes from software, processor timer interrupt or an internal dedicated time-code master count. The time-code controller has a time-code register for each of the four time-code flags, therefore allowing independent time-code forwarding for each flag code.

The time-code controller stores the last time-code received for each type of control flag and can indicate to the host that a time-code has been received through the status/interrupt interface.

The time-code forwarding mechanism checks that received time-codes are one more than the last time-code received then the time-code will be forwarded through all ports except the port the time-code arrived on. If the time-code is a distributed interrupt code then the interrupt vector is checked and the controller will forward the time-code if the interrupt vector bit is 0. If the interrupt vector bit is 1 the time-code is discarded as the interrupt has already been set. The time-code will be forwarded through all ports except the port the time-code was received on.

The controller can act as a time-code master either by software insertion of a time-code, sending time-code on a processor timer interrupt or by setting up an internal time-code master counter. The time-code frequency can be controlled by the host software with up to 1 micro-second precision.

Status bits and processor interrupts are provided for received time-codes for each time-code flag value, time-codes transmitted for each time-code flag value and distributed time-code interrupt occurred.

VII. Conclusion

The CASTOR chip is a capable SpaceWire processing unit which comprises a SPARC V8 process with an enhanced floating point unit and memory management unit running at 200 MHz on a radiation tolerant process. The SpaceWire engines inside the CASTOR chip provide high performance SpaceWire RMAP and DMA functions including dedicated RMAP target and initiator hardware to reduce the processor workload.

References

[1] ECSS Standard ECSS-E-ST-50-12C, “SpaceWire, Links, Nodes, Routers and Networks”, Issue 1, European Cooperation for Space Data Standardization, July 2008.

[2] ECSS, “SpaceWire - Remote memory access protocol”, ECSS-E-ST-50-52C, Feb 2010.

[3] S. Parkes, Albert. Ferrer, S Mills, A Mason, “SpaceWire-D: Deterministic data delivery with SpaceWire”, International SpaceWire conference, Russia, 2010.

[4] P. Mendham, S. Parkes, “SpaceWire Plug-and-play: a Roadmap”, International SpaceWire conference, Nara, Japan, 2008.

[5] Yuriy Sheynin, Sergey Gorbatchev, Liudmila Onishchenko, “Real-Time Signalling in SpaceWire Networks”, International SpaceWire conference, Nara, Japan, 2008.

[6] Chris McClements, Steve Parkes, “SpaceWire RMAP IP Core”, International SpaceWire conference, Russia, 2010.

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

[8] ECSS standard ECSS-E-ST-50-51C, “SpaceWire engineering: SpaceWire protocol identification”, European Cooperation for Space Data Standardization, February 2010.

[9] ECSS standard ECSS-E-70-41A, “Ground systems and operations – Telemetry and telecommand packet utilization”, European Cooperation for Space Data Standardization, January 2003.