SPACEWIRE DEVICE DRIVER FOR THE REMOTE TERMINAL CONTROLLER

Session: SpaceWire Components

Short Paper

Albert Ferrer Florit, Wahida Gasti

European Space Agency, Postbus 299, NL-2200 AG Noordwijk, The Netherlands

E-mail: albert.ferrer.florit@esa.int, wahida.gasti@esa.int

ABSTRACT

The SpW Remote Terminal Controller (RTC) ASIC is a single chip embedded system designed to effectively perform data handling at platform level and powerful data processing at payload level. One of its key features is the implementation of two highly configurable SpaceWire interfaces.

This paper presents an implementation of a SpW software device driver for the RTC. The driver provides the necessary interrupt service routines, functions and procedures to handle the SpW interface. It includes the definition of an Application Programming Interface (API) for SpW compliant embedded systems. Reception scheme is based on early packet identification so that a decision can be made whether a packet is immediately processed, discarded or saved in the system for further processing. Transmission scheme supports multicasting, send queues, and priority schemes. The driver implementation overcomes the problematic of arbitrary packet lengths and network blocking, providing sustained high data rates transfers.

1 HARDWARE DESCRIPTION

The SpaceWire-RTC device is a single chip that includes an embedded LEON2-FT SPARC V8 processor with a Floating Point Unit, two SpaceWire interfaces, a Controller Area Network (CAN) bus controller, ADC/DAC interfaces for analogue acquisition/conversion, and standard interfaces and resources (UARTs, timers, general purpose input and output). Software can be executed from the on-chip 64Kbyte memory or from external SRAM. [1]

The actual implementation of the SpaceWire interface is interrupt driven and performs DMA transfers for the reception and transmission of SpaceWire packets. The interface must be programmed with the memory location and the size of one receive and one transmit buffer, i.e it implements a single virtual channel for each link direction. Multiple packets can be stored in the same receive buffer but the information about their lengths is not preserved.

As additional features, the SpaceWire interfaces provide an extra virtual channel reserved for VCTP packets [1], and a hardware implementation of the RMAP protocol [2]. This last characteristic allows to read/write all memory space of the RTC from a SpW remote node without the intervention of the embedded processor.

2 DESIGN CONSIDERATIONS

After reviewing some on-board recurring applications SpW-based, the following design considerations for the SpW Driver have been identified:

Variable packet length: The SpaceWire standard [3, 4] defines a data link protocol layer with arbitrary packet sizes. In a robust SpW network all nodes should be able to handle or tolerate SpW packets of arbitrary size, despite the limited memory available in some embedded systems.

Network congestion: One of the important points for any SpaceWire node design is its networking issues and accordingly, its capability to avoid network congestion. When a destination node sends more data than the receiver one can store or process, the whole data path gets blocked through multiple routers due to the use of wormhole routing. Therefore, a SpW driver should be able to discard incoming data automatically when the receive buffer is full.

Limited resources: Spacecraft data handling is based on embedded systems with limited resources in terms of memory and processing power. A SpW driver implementation should avoid memory transfers when possible. For example, basic packet identification, i.e. size and protocol used, should always be obtained without requiring the application to read and store the complete packet. Considering that the link speed can be a multiple of the processor speed, in case of high network traffic non DMA memory transfers can be very demanding for the system.

Multi-threaded applications: For complex applications, the driver should also allow multiple threads to send packets with different priorities, optionally, in the context of an operating system. Blocking and non blocking functions should be provided.

Extended functionalities: Link error reporting capability and time-code functionality has to be considered, as the possibility to provide time stamp information on packet reception.

3 SPECIFICATIONS

Based on the above design considerations, the following main SpW driver features were implemented.

Performance and efficiency:

- Support for sustained bidirectional high data rate transfers (up to 160Mbit/s, for low demanding data processing applications)
- User application can obtain the length of a packet and may read a complete packet without performing any memory copy.

Memory requirements

- Driver implementation has a small code and data footprint, and does not require any external library.
- Receive buffers can have arbitrary sizes and can be dynamically adjusted by the user application. A receive buffer may contain multiple SpW packets.

SpaceWire functionality

- Three different packet transmission functions, including multicast packet function. Information about time transmission is provided upon completion.
- Multiple send requests can be queued and an identifier is provided to supervise their status. They can have two levels of priority and be cancelled before being executed.
- Information provided on packet reception: packet length, protocol ID, EOP marker, CRC and other errors.
- Multiple receive buffers can be queued or added dynamically. Receive buffers are actually implemented as optimised receive FIFOs.
- Capability for hardware packet rejection and software packet filtering on packet reception. Statistical information about packets rejected and filtered is provided.
- There is no limitation in packet sizes. Packets that are bigger than the size of receives buffers available can be read in multiple chunks of bytes.
- User application can be notified when a packet or Time Code is received and/or a send request is completed.
- The driver can be configured to automatically discard incoming data in case there is no more memory available for packet reception, and/or notify the user application of this event.
- The driver can configure the hardware to act as a time-master (sending Time Codes periodically) or time-receiver (retrieving the last received Time-Code).
- The driver provides complete link configuration and error notification and recovering. It also provides configuration functions for the RMAP and VCTP hardware support.

4 OPERATION

The SpW device driver provides the data link layer services defined in the SpW standard, plus configuration functions for the RTC specific hardware support of VCTP and RMAP protocols. The user application access to these services using an API (Application Programming Interface) specifically designed to simplify the operation of all SpW driver features [5].

Upon a successful initialization of the driver and a SpW Link, a buffer previously allocated in the memory by the user must be provided to the driver to be used as a receive buffer for the incoming packets.

The application requests information about each packet received, and obtains a pointer to the memory location where each packet is stored. The driver implements a circular FiFO, trying to avoid splitting a packet in multiple buffers. However, in some situations with large packets, this may not be possible. In that case, a specific function must be called that copies the packet to another user buffer.

The memory space used by the received packets in the receive buffer should be explicitly freed by the user after the packet has been copied or processed.

It is also possible to keep the packet in the driver's receive buffer. But in that case when the buffer gets full the driver will not reuse the memory space used. Therefore, the application must provide a new receive buffer (it may be queued, see figure 1)

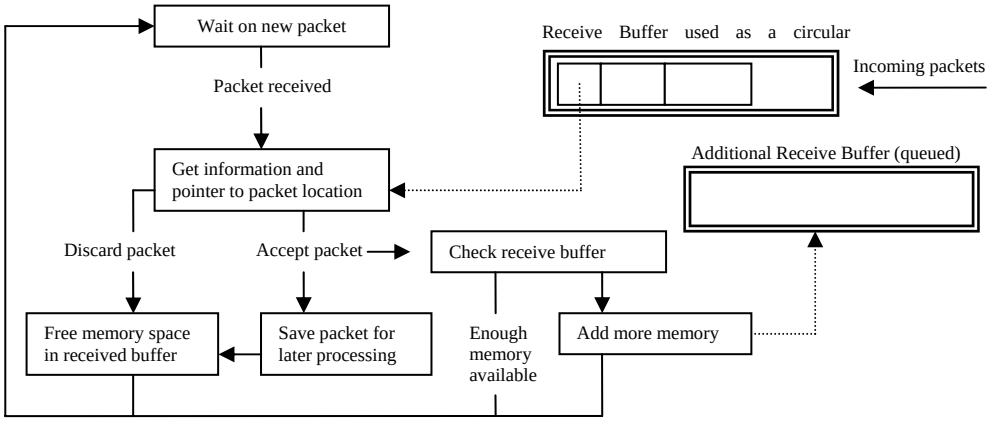

Figure 1: Simplified program flow for packets reception. Each box represents an API function call and an abstract representation of the receive buffers is included. Note that depending on the application, packets can be kept in the circular RxFIFO until they are processed.

Complex applications dealing with multiple protocols may have a single thread, which monitor a link and identifies the packets, and then forwards them to other threads that handle specific protocols. For packet transmission, multiple threads may execute and supervise send requests independently.

An application using some of the Driver's features can be found in [6].

5 CONCLUSIONS

An implementation of a SpW device driver has been presented targeting the Remote Terminal Controller. The Driver provides a full set of data link layer services, including time stamp information, priorities, packet filtering, link error reporting, and time-codes. The resulting design has a small memory footprint and low processing power needs, while providing enough flexibility to support a wide range of applications.

6 REFERENCES

1. SpaceWire Remote Terminal Controller, User Manual v1.7 (issued Feb 2007)
2. Remote Memory Access Protocol. ECSS-E-50-11 Draft F
3. SpaceWire - Links, Nodes Routers and Networks, ECSS-50-12A
4. ECSS-E-50-12 Part 2 Draft B January 2005 (also in ECSS-E50-11 chapter 5)
5. SpaceWire driver for the Remote Terminal Controller, User Manual.
6. TopNet Demonstration, R.Vitulli, A.Ferrer Florit, J.Ilstad

   International SpaceWire Conference 17-19 September, 2007 Dundee (Scotland)