PREPARING THE RASTA SOFTWARE FOR SPACEWIRE BACKPLANES

Session: SpaceWire Test and Verification
Short Paper

Daniel Hellström, Kristoffer Glembo, Sandi Habinc

Aeroflex Gaisler, Kungsgatan 12, SE-411 19 Göteborg, Sweden

daniel@gaisler.com, kristoffer@gaisler.com, sandi@gaisler.com

ABSTRACT

Replacing a Peripheral Component Interconnect (PCI) [3] backplane with a SpaceWire [1] backplane would normally require major changes to hardware and software. Today most PCI boards [6] used in the Reference Avionics System Test-bench Activity (RASTA) already include SpaceWire interfaces and connectors on the front panel. Aeroflex Gaisler has developed new abstract bus models for the *Driver Manager* (DM) used previously in RASTA software [5] solutions, in order to simplify porting of existing and newly developed RTEMS [4] software and improve code reuse. This paper focuses on the software aspects of this development.

BACKGROUND

The RASTA facility is a test-bedding element of the avionics laboratory at ESA [6]. It provides a representative environment for demonstration, evaluation and testing of hardware, software and communication protocols that comprise on-board data systems. RASTA has since grown in to a standardized platform [10] for on-board data system developments and is made mandatory in many ESA activities. RASTA consists of on-board computers, boards on-board I/O boards, TM/TC boards, etc.

Most of the RASTA hardware elements are implemented as system-on-chip designs based on the AMBA [9] on-chip bus. The processor element is the LEON2 or LEON3 SPARC processors, residing on the AMBA on-chip bus. The processor is connected via AMBA to one or more cores implementing various interfaces to buses such as PCI and SpaceWire. Other elements, such as on-board I/O, are also based on the AMBA on-chip bus, featuring interfaces such as PCI, SpaceWire, CAN, Mil-Std-1553B, CCSDS/ECSS Telemetry (TM) &Telecommand (TC) etc. Thus each board comprises an ASIC/FPGA that implements a system-on-chip based on the AMBA on-chip bus.

The PCI bus is currently used as the main interface between these boards. This paper discusses the next step, where PCI is replaced with SpaceWire links for implementing communication between boards. Communication between the processor and any other resource can be done in several steps; firstly the processor accesses local resources (i.e. cores) over the AMBA bus; secondly it accesses other units via PCI (current baseline); thirdly it accesses other units via a SpaceWire link (discussed in the paper).

RMAP INITIATOR STACK

Throughout this paper all SpaceWire accesses involve Remote Memory Access Protocol (RMAP) [2] commands sent to and responses received from RMAP targets. The RMAP header and CRC are generated by an *RMAP initiator stack* developed by

Aeroflex Gaisler (CRC is generated by hardware where supported), received RMAP responses are also processed by the stack. All RMAP commands are generated according to the ECSS standard [2]. The initiator stack supports path addressing, logical addressing and all defined transfer types. The initiator stack does not implement the RMAP command/response transportation itself, but relies on an *RMAP stack driver*, which in turn relies on SpaceWire packet transportation implemented by an underlying *SpaceWire driver*. Layering the software makes it possible to support SpaceWire cores from different vendors. The initiator stack does not require any particular RMAP support in hardware since the RMAP commands are generated in software. An *RMAP stack driver* has been implemented supporting the RTEMS SpaceWire drivers for the GRLIB GRSPW/GRSPW2 IP cores [7][8].

DRIVER MANAGER

1.1 OVERVIEW

Device drivers rely on services such as Plug&Play scanning and IRQ (Interrupt Request) management. The services are often implemented differently for different bus types (e.g. AMBA or PCI), and as a result different Application Programming Interfaces (API) are utilized when accessing these services. The *Driver Manager* (DM), developed by Aeroflex Gaisler, in RTEMS provides services such as driver loading (finding devices and appropriate drivers for them), IRQ handling etc. In an attempt to provide a single API for the above mentioned services regardless of bus type, the DM was created and abstract models for buses, devices, bus drivers and device drivers were introduced. These concepts have been successfully implemented and used in RASTA for on-chip AMBA buses, PCI buses and *AMBA-over-PCI* (i.e. communication from local-AMBA bus over PCI to remote-AMBA bus).

This paper introduces two new bus models; the *SpaceWire Network* and *AMBA-over-RMAP* (i.e. communication from local-AMBA bus over SpaceWire to remote-AMBA bus). In order to support the new models the DM API has been extended with READ and WRITE operations that can be used for communication over all bus types.

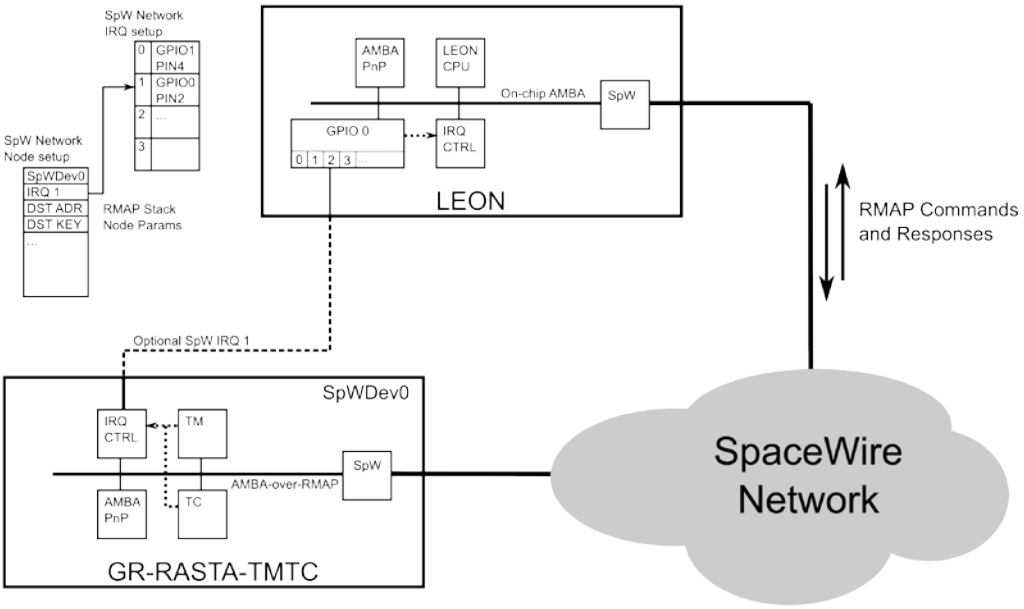

Figure 1: System overview

1.2 SPACEWIRE NETWORK BUS MODEL

The *SpaceWire Network* model implements IRQ management and read and write access to SpaceWire RMAP targets. The services are used by *SpaceWire Node drivers*. The SpaceWire standard currently does not define Plug&Play, instead the SpaceWire network configuration including all nodes are provided by the user. The most important node configuration parameters are the SpaceWire destination address, destination key and a virtual SpaceWire network IRQ number.

In each service request initiated by a *SpaceWire Node driver*, a pointer identifying the particular node is given to the *SpaceWire Network* model. From the node configuration the parameters used in lower layers can be determined. Read and write access requests are for example implemented by adding SpaceWire destination address/key and passing on the request to the RMAP initiator stack.

SpaceWire currently does not transport IRQ over the link. IRQ is optional and when enabled are communicated as discrete level sensitive signals (requiring some sort of GPIO hardware). The *SpaceWire Network* bus model works with virtual SpaceWire IRQ numbers that can be resolved into a GPIO pin accessible by the processor. In order to determine what caused the IRQ and to acknowledge the source, the RMAP target must be accessed over the SpaceWire bus.

There are a number of reasons why an RMAP request cannot be performed in an interrupt context, for example the time for completing an RMAP request may be too long or the initiator stack itself may rely on interrupts. Instead of acknowledging the IRQ source directly the GPIO interrupt is masked, stopping further IRQs from the source, and a high-priority Interrupt Service Routine (ISR) task is awoken. The task is responsible for calling all registered ISRs, the IRQ source will finally be acknowledged and before suspending the ISR task will unmask the GPIO interrupt.

1.3 AMBA-OVER-RMAP BUS MODEL

The *AMBA-over-RMAP* bus model provides AMBA device drivers with IRQ management and read and write access to AMBA cores over SpaceWire. The model can be seen as a generic *SpaceWire Node* driver laying on top of the *SpaceWire Network* model. By reusing existing AMBA bus layers and Plug&Play scanning, the footprint is reduced, and the *AMBA-over-RMAP* bus model interface is similar to a standard on-chip AMBA bus model, with a few additions.

The IRQ service is implemented on top of the *SpaceWire Network* IRQ service, the constraints are thus propagated to *AMBA-over-RMAP* device drivers. ISRs are executed in a high-priority task context in order to read and write the AMBA bus, this differs from the on-chip AMBA bus model.

Critical areas, such as accessing registers, in combination with changing the state of the software, are normally protected from an ISR by changing the Processor Interrupt Level (PIL) to disable interrupts during the execution of the critical section. *AMBA-over-RMAP* drivers cannot change the PIL to protect themselves, since a register access over SpaceWire may require IRQ or may take substantial time to complete. The same applies to the ISR itself that now executes in task context, it may be pre-empted while accessing an AMBA register over SpaceWire. This problem is solved by protecting critical areas with semaphores and introducing semaphores in the ISRs.

Even though register and memory accesses differ radically from the on-chip AMBA bus model, the simple API provided minimizes the required changes. The driver author needs not to worry about RMAP parameters as they are added in lower layers. The major differences are that accesses can be pre-empted by other tasks while waiting for response (depending on initiator stack, SpaceWire driver and scheduling policy) and that accesses may fail due to SpaceWire communication errors.

FUTURE IMPROVEMENTS

The current implementation of the *RMAP initiator stack* and the *SpaceWire driver* may create lock congestion when multiple tasks access the SpaceWire link simultaneously. Setting up task scheduling may reduce the problem significantly, however it cannot be avoided completely. Current software could be improved and support for multiple DMA channels could be added for the GRSPW2 core to avoid lock congestion.

Error handling could be improved in the bus models. RMAP read and write accesses may fail, layers beneath the AMBA device driver involved could be made to handle failures or re-execute requests automatically. With the bus model, error handling can be isolated per SpaceWire network, per SpaceWire node or per AMBA device.

CONCLUSIONS

The SpaceWire backplane software is already used in a separate ECSS TM/TC FPGA [12] project to set up, control and transport frames [11] over SpaceWire using RMAP.

The abstract bus models of the Driver Manager (DM) makes is possible to reuse and maintain most of the interface towards an AMBA driver, regardless of bus location (on-chip or over SpaceWire). Even though this paper describes a number of changes

that AMBA drivers must undergo for conversion to an *AMBA-over-RMAP* bus model, they are relatively few and easy and the driver structure can be maintained. Nevertheless, most importantly the user interface remains the same.

Currently SpaceWire Plug&Play is the only missing link in order to go completely Plug&Play, all the way from the controlling processor to a resource on the other side of a SpaceWire network.

REFERENCES

1. SpaceWire - Links, Nodes, Routers and Networks, ECSS-E-ST-50-12C
2. SpaceWire - Remote memory access protocol, ECSS-E-ST-50-52C
3. PCI Local Bus Specification, Revision 2.3, 2002, www.pcisig.com
4. RTEMS, Real-Time Executive for Multiprocessor Systems, www.rtems.org
5. RASTA Interface Control Document (ICD) - Software, TEC-EDD/2007.32/GF
6. RASTA Interface Control Document (ICD) - Hardware, TEC-EDD/2007.31/GF
7. GRLIB IP Library User's Manual, Aeroflex Gaisler, www.gaisler.com
8. GRLIB IP Core User's Manual, Aeroflex Gaisler, www.gaisler.com
9. AMBA Specification, Rev 2.0, ARM IHI 0011A, Issue A, ARM Limited
10. S. Habinc, *Crucial SpaceWire Elements in RASTA*, ISC 2008
11. S. Habinc, *CCSDS/ECSS Telemetry and Telecommand for RASTA*, DASIA 2009
12. S. Habinc, *ECSS TM-TC Component with SpaceWire RMAP Interfaces*, ISC 2010