ENHANCED DYNAMIC RECONFIGURABLE PROCESSING MODULE FOR FUTURE SPACE APPLICATIONS

Session: Missions and Applications

Long Paper

Frank Bubenhagen, Björn Fiethe, Harald Michalik, Björn Osterloh

IDA TU Braunschweig, Hans-Sommer-Str.66, D-38106 Braunschweig, Germany

Paul Norridge, Wayne Sullivan, Chris Topping

Astrium Ltd, Gunnels Wood Road, Stevenage, Herts, UK SG1 2AS3

Jørgen Ilstad

European Space Agency, ESTEC, Keplerlaan 1, Noordwijk ZH, Netherlands

E-mail: bubenhagen@ida.ing.tu-bs.de, fiethe@ida.ing.tu-bs.de, michalik@ida.ing.tu-bs.de, b.osterloh@tu-bs.de, paul.norridge@astrium.eads.net, wayne.sullivan@astrium.eads.net, christopher.topping@astrium.eads.net, jorgen.ilstad@esa.int

ABSTRACT

Future space missions require high-performance on-board processing capabilities and a high degree of flexibility. State of the art radiation tolerant SRAM-based FPGAs with large gate count provide an attractive solution for in-flight dynamic reconfigurability. With these devices an advanced System-on-Chip (SoC) can be implemented, but also the system reliability and qualification has to be guaranteed for the harsh space environment. Therefore single modules have to be isolated from the system physically and logically by qualified communication architecture, presented in this paper: The SpaceWire based System-on-Chip Wire (SoCWire) communication network. SoCWire provides a safe way to dynamically reconfigure parts of the FPGA during flight. First verification results of a dynamic reconfigurable SoC based on SoCWire are presented. Developed around SoCWire, the basic architecture for an advanced Dynamic Reconfigurable Processing Module (DRPM) is proposed.

1 INTRODUCTION

For data processing of payload instruments on scientific spacecrafts specific processing modules are commonly used. With increased data rates and the requirement to control multiple sensors, the need for increased on-board processing capabilities and a higher degree of instrument autonomy grow. While there are higher requirements for a data processing on the one hand, on the other hand some basic conditions remain still the same, i.e. limited downlink capacity, limited resources of power and mass. Also the need for shorter development times and the demand by scientists to adapt the instrument to mission specific requirements, even after launch, require an advanced architecture. This has to be adaptable in flight and has to guarantee the once on-ground achieved qualification even after change of modules.

Today the SRAM-based Virtex FPGAs from Xilinx provides high logic capacity and thus offer a highly flexible platform to implement a reconfigurable System-on-Chip (SoC) in a single device. These devices are available in radiation tolerant versions and already have proven reliable flight heritage in many space missions, e.g. ESA Venus Express (VEX) or NASA Dawn. However, the full flexibility of these devices to perform complete or partial reconfiguration even during operation was only used throughout the development phase on ground so far.

For an enhanced reconfigurable system the system qualification has to be guaranteed. Effects during the reconfiguration process, space radiation induced errors and interference of updated modules on the system have to be prevented. Therefore updated modules have to be isolated physically and logically by qualified communication architecture from the system.

This paper presents the key element for such an enhanced architecture, the SpaceWire based System-on-Chip Wire (SoCWire) communication network. SoCWire provides a safe way to dynamically reconfigure parts of the FPGA during flight. First verification results of a dynamic reconfigurable SoC based on SoCWire are presented. At last the basic architecture for an advanced processing module is proposed.

2 EFFECTS WITHIN A RECONFIGURABLE FPGA

The use of Xilinx SRAM-based FPGAs for a dynamic reconfigurable system requires considering of two effects: (i) glitch effects, which occur during the dynamic partial reconfiguration process while the FPGA is in operation and (ii) SEUs (Single-Event-Upsets) within the space environment.

Partial reconfiguration denotes the modification of a limited, predefined portion of a FPGA. A minimal reconfigurable system consists of a static area, which remains unchanged and a Partial Reconfigurable Area (PRA), which is shared by at least two Partial Reconfigurable Modules (PRMs) with different functionality. Xilinx FPGAs have no explicit activation technique for a PRA. Therefore the configuration frames become active as they were written. Configuration bits remaining unchanged will not glitch during reconfiguration, but bits with a change of its logical state could momentarily glitch when the frame write is processed. Experiments with reconfiguration of a PRA from PRM1 to PRM2 and vice versa have shown unpredictable behaviour for both, the duration of glitches and their influence on the interface between the PRM and the static area.

A SEU is caused by charged particles losing energy by ionizing the medium which they pass and leaving behind electron-hole pairs. Within a memory cell or flip-flop this can cause a change of state and consequently corrupt the stored data. The configuration for the programmable elements and routing resources of a Xilinx FPGA is stored within static memory cells. With scrubbing falsified memory cells can be corrected by reloading of configuration memory with the initial design, but this does not prevent a propagation of an error through the system. Techniques like Triple Modular Redundancy (TMR) can mitigate error propagation. The drawbacks of TMR are higher resource utilization, a decrease of speed due to longer paths and an increase of current because of more logic. Typically processing units for scientific instruments are not mission critical. As result a trade-off between limited resources and instrument

availability is partly applied TMR. Anyhow, a SEU in a non-TMR PRM interface logic could block the communication architecture and stop the system.

With glitch effects and SEU induced errors during dynamic partial reconfiguration the system qualification in a classical bus-based architecture within a FPGA cannot be guaranteed. An enhanced architecture is required, which isolates PRMs from the TMR protected host system to guarantee uninterruptable operation of the system.

3 SYSTEM-ON-CHIP WIRE (SOCWIRE)

SoCWire has been developed to provide a Network-on-Chip (NoC) architecture which is able to connect several PRMs with a host system and concurrently isolate the PRMs logically and physically. SEU induced error, glitch effects or an intended replacement of a module does not affect the operation of the remaining system.

3.1 SOCWIRE BASICS

Available spacecraft communication standards, e.g. MIL-STD-1553B, CAN bus, SpaceWire were analyzed and compared for their suitability for a NoC. The outcome of this analysis was that SpaceWire as an asynchronous, point-to-point, bi-directional, serial link interface with a credit-based flow control, error detection, hot-plug ability and automatic reconnection after a link disconnection [1] is currently the only available switch topology and most suitable for a fault-tolerant and robust NoC approach. As mentioned before SpaceWire is an asynchronous interface and performance depends on skew and jitter. Processing modules are implemented within a complete on-chip environment (NoC approach). Therefore, the Spacewire interface has been modified to a synchronous, 10bit parallel data interface (8bit data, control flag, parity bit), which results in significantly higher data rates compared to the SpaceWire standard, e.g. 800Mbit/s at clock frequency of 100MHz. Additionally, the data word width is scalable from 8bit to 128bit, which further improves the throughput. Furthermore, the advantageous and reliable features from this standard, such as flow-control, error detection and automatic link recovery in case of an error, were preserved. Since SoCWire operates in a complete synchronous environment, the timeouts during initialization and detection and recovery after a link disconnection could be significantly decreased.

3.2 SOCWIRE NETWORK

To build up a network, a switch and a packet oriented protocol is needed. A SoCWire network as shown in Figure 1 comprises: SoCWire coder/decoder (CODEC) as network interface and a SoCWire switch to route the data packets through the network [2]. The SoCWire switch is again based on the SpaceWire standard. A SoCWire CODEC connects a node or the host system typically via a SoCWire switch to a SoCWire network. The nodes are similar to SpaceWire nodes source and destination of a link. The SoCWire switch is scalable from 8bit to 128bit data word width and provides a configurable number of up to 32 ports. In contrast to a SpaceWire router the configuration port was discarded and logical addressing is not supported to safe resources. A simple path addressing scheme is implemented instead, which is suitable for small on-chip networks. The SoCWire switch comprises wormhole routing and the simple time slot based round robin scheduling algorithm.

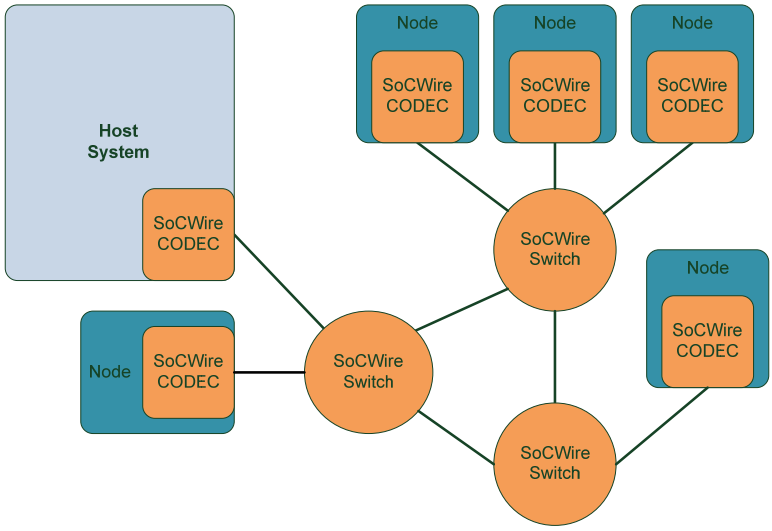

Figure 1 SoCWire architecture network example

4 SoCWire: Architecture Verification

The objective of SoCWire is to provide a robust communication architecture for dynamic partial reconfiguration systems. Since the major requirement for SoCWire is the isolation of a PRM, this feature has to be validated in an architecture verification.

4.1 Functional Verification

SoCWire has to be validated on the advantageous features of SpaceWire with link initialization, error detection/recovery and unidirectional and bidirectional data rates. The main difference between SpaceWire and SoCWire is that SoCWire provides a parallel data interface and operates in a completely synchronous environment. The advantage of this point is that SoCWire is more deterministic, because any change of state is related to clock cycles. SoCWire is a fully pipelined implementation and two clock cycles are required to perform an action. One advantage of the synchronous environment is the much faster initialization of a link in comparison to SpaceWire. A disconnection is detected after three clock cycles, the exchange of silence lasts six clock cycles and the timeout twelve clock cycles. Overall, 26 clock cycles minimum are necessary for building up a link on the condition that both SoCWire CODECs receive the reset at the same time. Tests with adding delays of different length to one of these reset signals always resulted in a proper initialization of the link. Both the unidirectional and the bidirectional data transfer have been tested with a Pseudo Random Bit Sequence (PRBS) generator stimulus to validate data integrity. Furthermore, data packets of different length (1 to 1048576 bytes) have been tested. In all performed tests no transmission errors have been detected and the data rates from the simulations could be verified. Furthermore, SoCWire has been tested and validated on the error detection/recovery features of the SpaceWire standard, e.g. parity errors, escape errors, character sequence errors, credit errors and disconnect errors. Figure 2 depicts the fault injection mechanism for this verification. All errors have been successfully injected and error detection and recovery could be validated to be SpaceWire conform. The error recovery time of SoCWire is at minimum the initialization time for a link plus synchronization overhead.

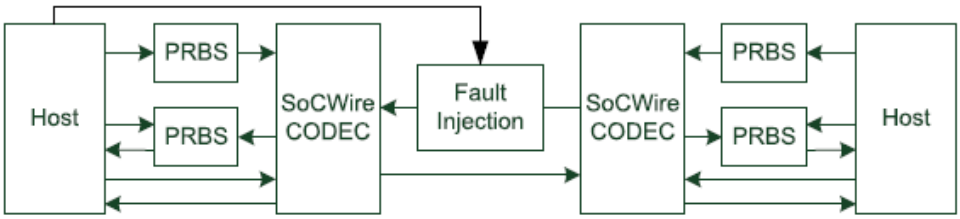

Figure 2 SoCWire verification architecture

4.2 FAULT TOLERANCE

One mandatory requirement in a space environment is fault tolerance. Single-Event-Upsets on a SoCWire node within the FPGA can be modelled as stuck bit either at logical ‘0’ or ’1’. This error can occur during a link initialization or during run-time. With the programmable fault injection mechanism a certain bit in the link has been fixed to one of the logical states. In all performed test a stuck bit either in the link initialization phase or during run-time does not affect the functionality of the host system and an error is reported. Appropriate error recovery schemes can be applied by the user, e.g. scrubbing.

During the reconfiguration process of a PRM glitch effects occur and affect nearly every part of the interface logic for a given time in the range of microseconds. These effects impact the link initialization phase when an empty PRA is configured for the first time or during an established link connection when a PRM is replaced by another one. To verify the impact of glitch effects on a SoCWire CODEC interface, a random pattern generator with random delay in the range of nanoseconds to several microseconds was implemented in hardware. This generator emulates the behaviour on the interface signals which could occur with different PRM configuration patterns. Even though this generator does not simulate the real FPGA technology and effects during dynamic partial reconfiguration, during the test the SoCWire host system was not disturbed in its operation.

4.3 PARTIAL DYNAMIC RECONFIGURATION

A dynamic partial reconfigurable SoCWire architecture with host system including SoCWire CODEC and a PRM with SoCWire CODEC as well as an additional module for control and data generation e.g. PRBS has been implemented. Moreover, a static PRM with all outputs ones and a PRM with pure counter functionality have been created. The following tests have been performed with the JTAG interface to reconfigure the system dynamically: (i) SoCWire counter module to SoCWire PRBS module, and (ii) Static module to SoCWire PRBS module. Since dynamic partial reconfiguration has the same behaviour as scrubbing on all elements within a module which does not change, test (i) was performed to prevent the SoCWire CODEC from not being affected by the dynamic reconfiguration process.

Two behaviours have been observed during the tests, which are shown in Figure 3. The figure represents the “active signals” or link connected from the SoCWire CODEC core on host system side and on PRM side. (I) shows a smooth dynamic reconfiguration of the PRM. Glitches occurred on all PRM outputs during the reconfiguration process. (II) shows the glitch effects as well as a repeatedly establishing link connection stabilising at the end. There is not much known about the

dynamic partial reconfiguration process in Xilinx FPGAs to explain this effect. Configuration frames become active when they are written; it is most likely that parts of the design operate before the reconfiguration process is finished.

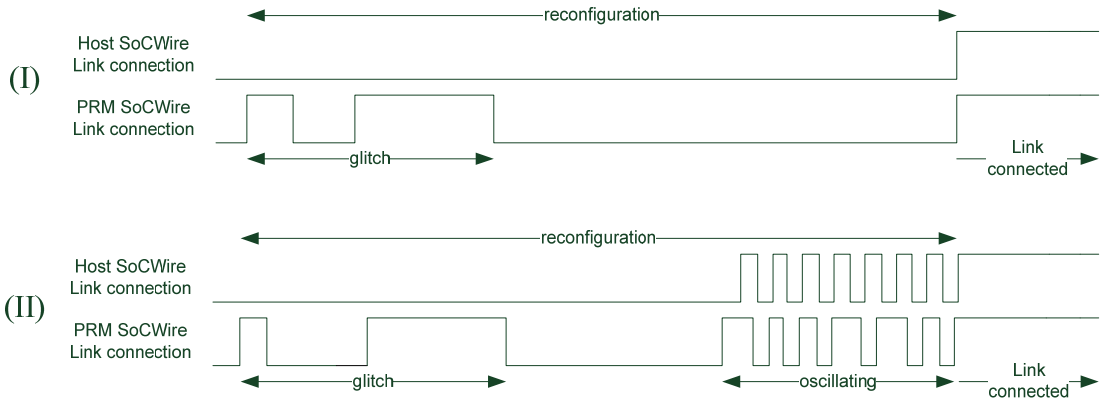

Figure 3 SoCWire "active signals" during dynamic partial reconfiguration

Since the configuration memory within a Xilinx FPGA is written from the left to right side [3], also the influence of bus macro placement, which establishes communication between static area and PRAs, has been analysed. During the initial tests the bus macros were placed on right side of the PRM as depicted in Figure 4 on the left hand side. Tests with placement of the bus macros on the right hand side of the PRM showed a smooth stable link connection avoiding the oscillation effect. Even with oscillating behaviour during the dynamic partial reconfiguration process, the PRMs were isolated from the host system and do not have any effect on its operation.

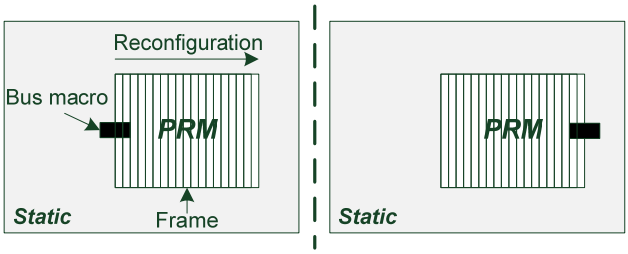

Figure 4 PRM bus macro placement

5 DRPM ARCHITECTURE

Around the SoCWire communication architecture we are currently developing under ESA contract a flexible processing system with full support for in-flight dynamic partial reconfiguration of application firmware, the Dynamic Reconfigurable Processing Module (DRPM). The basic DRPM architecture is shown in Figure 5 [4]. The major subunits are (i) dynamically reconfigurable FPGAs (within each DFPGA), (ii) SpaceWire router for hosting and managing the networking between various subunits, (iii) system controller for overall configuration control of the module and execution of application software, (iv) interfaces to spacecraft using standards like SpaceWire, MIL-1553B and CAN bus, and finally (v) interfaces to the instrument electronics, e.g. sensors or cameras.

The DRPM comprise a highly modular architecture. Consequently, the SpaceWire router can provide expandability not only to additional DFPGAs, but also to

additional DRPMs. With this concept it is possible to simply extend the processing capacity by attaching additional modules or adding modules for hardware redundancy. Then one system controller would be the master and the other ones slaves.

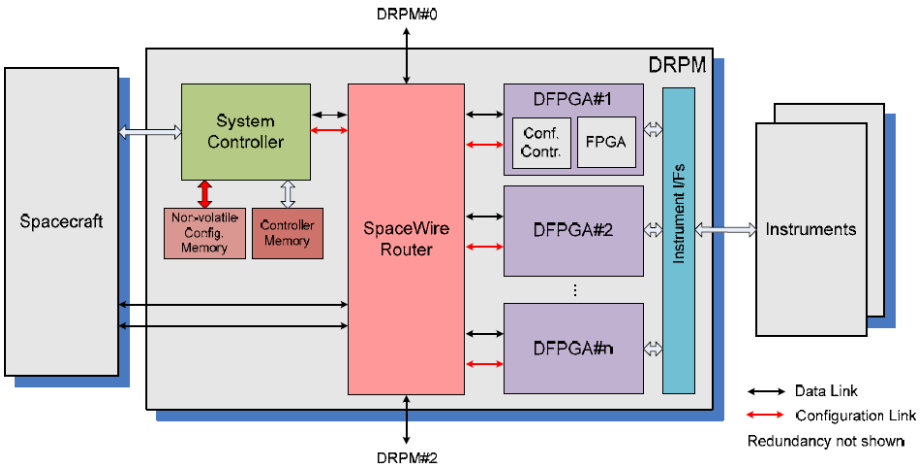

Figure 5 DRPM architecture

Since the system controllers' main task is controlling and supervising the overall DRPM, a fault-tolerant processor implementation should be used for this subunit. For instance, the LEON-based SpaceWire RTC ASIC (AT7913E) already incorporates the required interfaces like RMAP compatible SpaceWire and CAN bus controller. Of major importance is a safe and flexible implementation of the high capacity non-volatile memory for secure storage of all partial configuration bit files needed for the DFPGAs. Each reconfigurable DFPGA consists of configuration controller containing the static area with common interfaces and one or several reconfigurable FPGA(s), mainly comprising the dynamic area. This basic architecture is depicted in Figure 6.

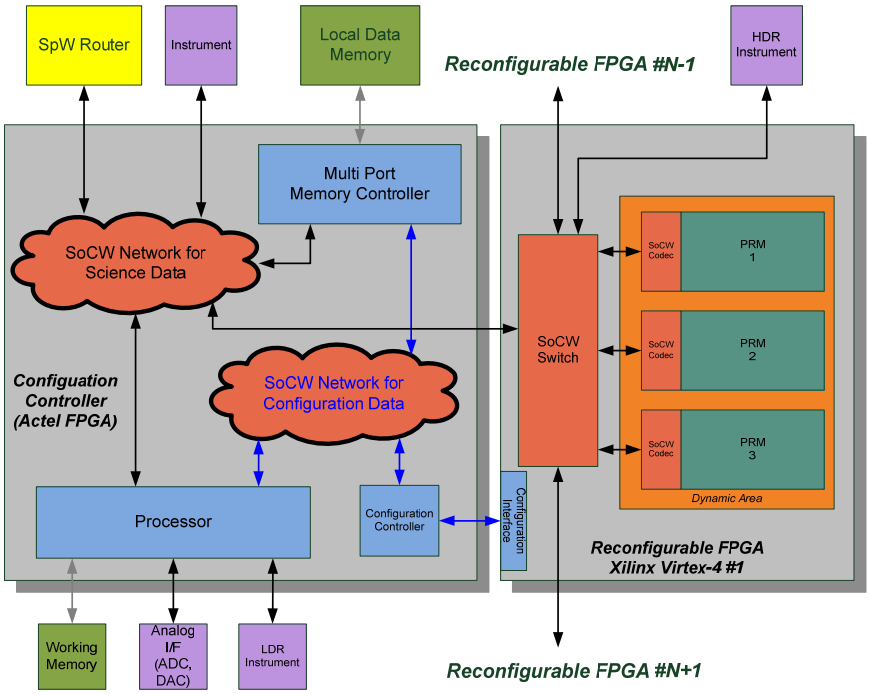

Figure 6 DFPGA architecture

The configuration controller is responsible for configuration, verification and supervision of PRMs within the dynamic area. It is implemented within a TMR by design one-time programmable RTAX FPGA. Two independent SoCWire communication networks form the backbone for controller functionality. One network is responsible for secure dynamic configuration and scrubbing of reconfigurable FPGA(s). The other one connects the dynamic area with the processor of the controller, instrument interfaces and a large local data memory. To achieve high reliability this memory is implemented in the static area with advanced symbol error correction capabilities for secure temporal storage of local configuration files. The independency of the two SoCWire networks provides increased reliability. The processor is required to provide data-flow control functions for the allocation and access of various interfaces to the commonly used data memory and configuration management of the attached reconfigurable FPGA(s). The dynamic area is based on Xilinx Virtex-4 FPGAs which are available on reliable packaging and certified for radiation performance and reliability. The SoCWire switch within the small static area of the reconfigurable FPGA(s) connects to the different PRMs and optionally directly to external high-speed interfaces, e.g. Channel Link. To achieve a modular architecture, the switch provides also direct data exchange between different reconfigurable FPGAs or even DFPGA subunits.

6 CONCLUSION

The DRPM provides an architecture being suitable to satisfy the demand of future space missions for high performance on-board processing with the requirement to update processing modules in-flight. One issue within such an enhanced architecture is the guarantee of system qualification, even after an update of a processing module. SpaceWire is widely used as a proved reliable interface standard on-board spacecrafts. Modifying this standard to the fault-tolerant, high-speed on-chip communication architecture SoCWire for FPGAs offers the possibility to built-up systems where processing modules can be exchanged without affecting the operation of the host system. SoCWire is published as an open source project provided by IDA. Source code, documentation and testbenches can be accessed at www.socwire.org.

7 REFERENCES

1. ESA ESTEC, “Space Engineering: SpaceWire-Links, nodes, routers, and networks”, ECSS-E-50-12A, Noordwijk Netherlands, January 2003.

2. B. Osterloh, “SoCWire User Manual“, www.socwire.org, 2009

3. Xilinx Inc., “Virtex-4 FPGA Configuration User Guide. UG071(v1.11)”, www.xilinx.com, USA, 2009.

4. ESA, “FPGA bases generic module and dynamic reconfigurator”, TEC-EDP/2008.30/JI, Issue: 1 Rev.1, Noordwijk, Netherlands, 2008