OVERVIEW OF THE INTAμSAT'S MASS MEMORY UNIT BASED ON SPACEWIRE[1]

Session: SpaceWire Missions and Applications

Short Paper

J. Ignacio García, J. Antonio Martín, Daniel Meziat, Manuel Prieto

Space Research Group. Dpto. Automática. Universidad de Alcalá. Ctra. Madrid Barcelona Km 33,600. 28871, Alcalá de Henares, Spain

Laura Seoane, Manuel Angulo

INTA. Space Programmes Department., Ctra. de Ajalvir, km. 4. 28850, Torrejón de Ardoz, Spain

E-mail: ignacio.garcia@srg.aut.uah.es, jamartin@srg.aut.uah.es, daniel.meziat@srg.aut.uah.es, manuel.prieto@srg.aut.uah.es, angulom@inta.es, seoanepl@inta.es

ABSTRACT

This paper presents the solutions adopted for the INTAμSAT-1 Mass Memory Unit, which makes use of several high-speed interfaces based on SpaceWire. This MMU handles the information coming from 3 medium resolution Earth observation cameras and other high data rate experiments in parallel by using 13 point to point SpaceWire channels with up to 110 Mbps data rate. The information will be properly formatted, stored and queued for later transmission to ground through additional point to point SpaceWire nodes connected to the spacecraft X-Band and the S-Band modems, and a PTM modulated Laser downlink high speed transmitter. The MMU architecture and preliminary tests over a representative prototype to validate the concept are presented.

1 INTRODUCTION

INTAμSAT-1 will be the first mission of a new small satellite programme with a mass ranging from 80 to 150 kg, and compatible with VEGA, Soyuz-ST and Dnepr launchers. The first INTAμSAT-1 will be a nadir pointing satellite with body fixed solar panels devoted to R&D remote sensing, with a launch tentative date during 2012. This enlarged IμSAT class is a further step after the NANOSAT programme success with a launch onboard Ariane-V-165 in Dec. 2004 (Nanosat-01 is still working in orbit), and the Nanosat-1B, a follow-on UHF store&forward communications mission with new experiments, launched from Baikonur by a DNEPR in July 2009. Inside the INTAμSAT-1 spacecraft two types of data buses are used [1]. The OBDH housekeeping TM/TC data bus, to which all units inside the spacecraft will be connected, will use the CAN standard. The Mass Memory Unit (MMU) implements several SpaceWire high-speed channels for information exchange

[1] This work has been partially supported by the Spanish Ministry of Science and Innovation MICINN under the grant AYA2009-13478-C02-02.

with the payload equipments and the communications subsystem. Its objective is to provide a temporary memory resource in order to store the huge amount of data provided by the cameras until it is downloaded to the ground stations.

2 MMU REQUIREMENTS

2.1 COMMUNICATION INTERFACES AND BANDWIDTH ASSESSMENT

The most defining factor in the design of the MMU is the amount of high-speed channels to be used simultaneously. The satellite features 3 cameras [2] with several CCD outputs, providing many data links even after processing and multiplexing:

- The CINCLUS instrument provides 8 data signals with a bitrate around 18 Mbps. Multiplexing results in 2 links producing 72 Mbps each.
- Each of the two MS-WAC cameras provides 10 data signals with a bitrate around 55 Mbps. Multiplexing results in 10 links producing 110 Mbps each.
- Finally, the PAU experiment requires a single 8Mbps link.

This results in a total of 13 SpaceWire links with a peak data rate of 1,252 Mbps. Data must also be sent to ground, for which the satellite features 3 communications systems: A 100 Mbps laser downlink, a 72 Mbps X-Band modem (including 7/8 Viterbi and 187/204 Reed-Solomon codifications) and an 8 Mbps S-Band modem (including 7/8 Viterbi), resulting in a peak bandwidth of 172 Mbps.

2.2 CAPACITY AND OTHER MEMORY CONSTRAINTS

It's been determined that a total of 80 Gbits of net storage is necessary for the mission. Due to its combination of higher radiation tolerance and reasonably high density, SDRAM technology has been selected for mass storage.

2.3 OTHER CONSIDERATIONS

Apart from the aforementioned constraints, the MMU must feature a CAN bus interface both for command and control, and to provide low-speed storage services for other spacecraft subsystems such as the OBDH. Additionally, the need for achieving low power consumption and the size constraints (160x200mm double-Europe board size), dictate a system with few components and low operating frequency. A single FPGA system with a LEON2 synthesized processor and embedded RAM seems like a good enough target; specifically an Actel RTAX2000 FPGA has been chosen.

3 MMU DESIGN

3.1 PRELIMINARY DESIGN CONSIDERATIONS AND APPROACH

The fact that the MMU needs to address many SpaceWire channels concurrently makes a classical single-CPU data processor approach to the MMU design very difficult. The interrupt rate generated and a data stream well above the 100 MB/s mark would be difficult to handle by a synthesized CPU with a targeted speed of a few tens of Mhz. Direct memory access by the hardware handling the SpaceWire

links is clearly a must, and on-the-fly post-processing and formatting of the stored data seems infeasible. Another matter of concern is that the imposed 80 Gbits storage requirement far surpasses the 4 GB physical addressable memory directly supported by typical 32-bit processors such as LEON2.

A system consisting of several independent SpaceWire controllers with direct access to the mass memory managed by a simple CPU-based controller subsystem seems like a more feasible approach. The controller is unloaded of the burden of moving data back and forth and can be kept as simple as possible. A single-channel 64-bit SDRAM memory seems adequate for the application while working at a reasonable 25Mhz operating frequency (peaking at 200 MB/s data transfer rate).

3.2 DETAILED FUNCTIONAL DESCRIPTION

Given the MMU requisites and design limitations, the only service provided by the MMU is raw storage. Data received from each channel will be directly stored in memory with no further post-processing or formatting, and will be transmitted to ground also in this form. Error checking, data correlation, analysis, etc. will be done in the ground station and the packet format (if any) used by the payloads will be transparent to the MMU. The OBDH is be responsible of instructing the MMU about the memory areas where the data coming from every specific channel is to be stored, or what data is to be sent to the communications subsystem. Thus, the MMU is treated as a raw storage device and it is the OBDH who accounts for free and used space and decides data retransmission or other management decisions.

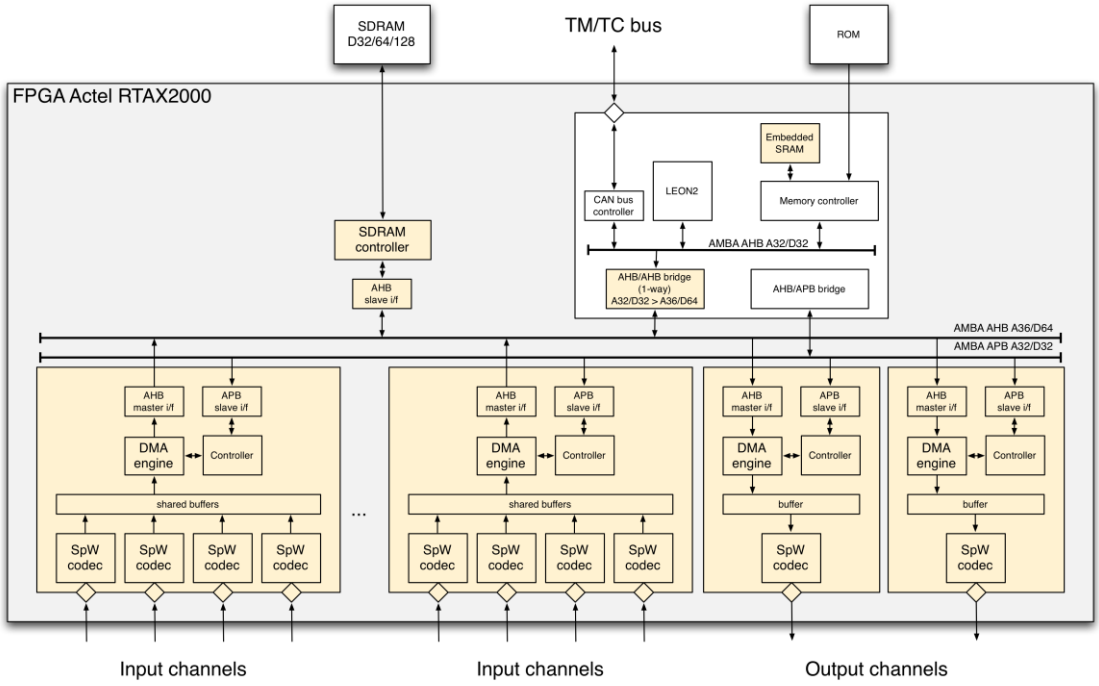

Figure 1: MMU block diagram

3.3 BLOCK DIAGRAM

A SoC design (further explained in the following paragraphs) has been produced, represented in the high-level block diagram on Figure 1. The shaded blocks are in-house developed IP cores, while the light ones are freely available IP cores, slightly

modified versions of these or just discrete components. The design is built around a slightly customized AMBA AHB bus with modifications to the address space range to fit the required addressable mass memory, and wide enough to cope with the required data rate while maintaining a relatively low operating frequency. A 36-bit address, 64-bit data bus has been proposed, potentially leading to a 128-bit data bus implementation. The mass memory chips are connected to an in-house developed SDRAM controller with slave AHB interface, allowing us to make it adaptable to various address and data widths or even different memory technology. This approach allows for easy addition of multiple mass memory banks if necessary, by connecting more slave memory controllers to the AHB bus.

The SpaceWire receiver and transmitter elements are kept as simple as possible, focusing mainly in high-performance and low FPGA footprint. Transfer parameters can be programmed through registers, and once a transfer is activated the IP core can receive/send data from/to the SpaceWire link autonomously, much like a programmable DMA controller. These elements employ an in-house developed SpaceWire IP core codec [3] and have a master AHB interface, which allows them to perform read/write operations independently and provides great flexibility since it allows adding and removing SpaceWire channels in a very straightforward manner. 2 or 4 SpaceWire channels are grouped in the same IP core to keep the number of AHB masters in the bus below the maximum supported.

In order to process TC from the OBDH and produce the required TM, the MMU features a LEON2 SoC with CAN bus interface as controller subsystem, which is able to program the SpaceWire controllers through the AMBA APB bus and perform various other operations. The AHB bus of the LEON2 controller subsystem is independent from the main AHB bus in order not to cripple performance and to override the address and data bus width limitations of the LEON2 CPU. An AHB/AHB unidirectional, programmable bridge has been developed which maps a programmable window of the controller AHB space into the main AHB bus, thus allowing access to the full storage space from the LEON2 controller. Also, since all the IP cores connected to the modified version of the AHB bus are developed in-house, any customization regarding address or data width is possible.

4 IMPLEMENTATION AND PROTOTYPE

To help with IP core testing and to prove the validity of the design, a simpler MMU demonstration prototype has been produced using a Pender GR-CPCI-XC4V Virtex4 FPGA development board with the accessory GR-CPCI-SER2-SPW2 mezzanine, which provides two SpaceWire ports. The prototype features the same design as the final system, but there are only two SpaceWire IP cores, one for input and one for output. Also, the main AMBA AHB bus is only 32 bits wide and supports 32-bit addresses, and all the AMBA AHB masters and slaves are modified accordingly. The two AHB bus approach using the AHB/AHB bridge is used, nonetheless. Finally, the LEON2 controller subsystem does not use an external ROM; the firmware is directly loaded and run from embedded SRAM using the GRMON tool and the on-board SDRAM of the XC4V is used exclusively as mass storage. The firmware running in the MMU prototype has been custom built and does not use any operating system, and despite reduced functionality is basically the same than the firmware that is expected to run in the final system.

In order to emulate the payloads, a Gaisler SpaceWire-RTC development suite has been used. This system features two SpaceWire links which are used to emulate the data stream coming from a payload and a to downlink interface, allowing us to building a simplified scenario which is representative of the final solution. It also features a CAN bus controller which is used to command the emulator, to configure and run the tests. The software running in the RTC uses RTEMS as underlying OS.

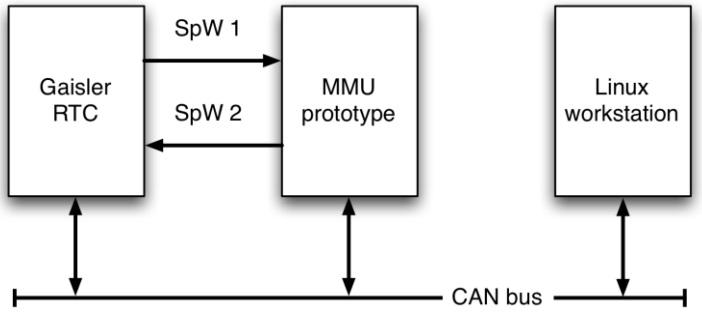

Figure 2: Prototype testing scenario

Finally, a Linux PC workstation with a CAN bus adapter is used to configure and control both the MMU prototype and the payload/downlink emulator in the RTC through a simple to use visual application. The scenario is shown in Figure 2, and has been proved to work as expected with both SpaceWire channels working concurrently and the LEON2 controller handling the configuration and control as instructed.

5 CONCLUSIONS AND FUTURE WORK

A simple MMU design valid for high data rate, high interface count applications has been produced, providing basic raw storage services adequate for certain applications. The design is focused in high bandwidth and simple implementation, sacrificing the provision of more complex storage services in exchange of the ability to use fewer components and lower power. At the same time, the design provides high flexibility to provide an easy upgrade path in case a different channel configuration, more memory banks or higher fault tolerance is needed, and in the presented scenario it requires only the use of a single, medium capacity FPGA. Work is already underway to support DDR and DDR2 memory and to improve the developed IP cores to enhance performance and reduce footprint. Also, more advanced firmware capabilities are being considered to support features such as scheduled data downloading.

6 REFERENCES

1. D. Guzman, M. A. Angulo, L. Seoane, S. Sánchez, M. Prieto and D. Meziat, "Overview of the INTAμSAT's Data Architecture Based on SpaceWire", International SpaceWire Conference, Dundee, Scotland, 2007.

2. M. A. Angulo., L. Seoane, E. Molina, M. Prieto, O. Rodríguez, S. Esteban, J. Palau and E. Cornara , "INTAμSat-1 First Earth Observation Mission", 7th IAA Symposium on Small Satellites for Earth Observation. Berlín, Germany, 2009

3. P. Aguilar, M. Prieto, D. Guzmán, D. García and R. Castillo, "Implementación en SoC reconfigurable de un códec Spacewire basado en el estándar ESA ECSS-E-ST-50-12C", Jornadas de Computación Reconfigurable, Alcalá de H., Spain, 2009