A Lightweight Operating System for the SSDP

A. Luntzer[a] , F. Kerschbaum[a] , R. Ottensamer[a] , C. Reimers[a]

[a]Department of Astrophysics, University of Vienna, 1010 Vienna, Austria

armin.luntzer@univie.ac.at

Abstract

The Department of Astrophysics at the University of Vienna is a provider of payload instrument flight software. Among the projects under development is a custom, lightweight operating system for the upcoming Scalable Sensor Data Processor (SSDP) based on prior experience with its predecessor, the Massively Parallel Processor Breadboard (MPPB). The objective of this project is to create easy to use software that is capable of efficiently driving the SSDP's Xentium DSP cores. Through its unique concept of driving the DSPs, it allows the user to make full use of the resources of this specific platform.

I. Introduction

A common problem of space missions is the limited processing power of available space-qualified hardware, as Payload data processors of on-board spacecraft and satellites are subject to high levels of radiation. While there is the LEON to fill the role of a general purpose processor (GPP), the only radiation hardened digital signal processor (DSP) available in Europe is the already dated ADSP-21020, if ITAR/EAR regulations are taken into account.

The need for a new processor or System-on-Chip (SoC) computer design for on-board payload data processing is high. This is mainly due to the ever increasing quantity of sensor data, as modern instruments produce ever larger volumes of measurements. Available down-link bandwidth however, is limited by available power, antenna sizes and in the end, physics.

In recent years, ESA has been pursuing the development of a next generation payload processor. One of the outputs of this effort is a prototype SoC called the MPPB (Massively Parallel Processor Breadboard) developed by *Recore Systems* under ESA contract 21986 [1]. The MPPB is built around a Very Long Instruction Word DSP architecture named *Xentium*. In this platform, a LEON processor is acting as a supervisor, controlling a Network-on-Chip (NoC) with multiple DSPs, memory and I/O devices attached to it.

II. Motivation

In the course of the NGAPP (Next Generation Astronomy Processing Platform) activities, an evaluation of the MPPB was performed in a joint effort of RUAG Space Austria (RSA) and the Department of Astrophysics at the University of Vienna (UVIE). While the original intent of the work of UVIE was to quantify the performance of the Xentium DSPs and the MPPB as a whole with regard to on-board data treatment and reduction in an astronomical mission setting, it was found that, given the highly innovative nature of this new processing platform, a novel approach was needed concerning the management of system resources, DMA mechanics and DSP program design for best efficiency and turnover rates.

Consequently, the University of Vienna developed an experimental operating system to stably drive the DSP cores and the MPPB close to its performance limit. This was achieved by splitting processing tasks into a pipeline of small units (kernels) that are dynamically scheduled to run on the Xentium DSPs, as required by the amount of data in the pipeline stages, thereby overcoming bottlenecks resulting from memory transfer overheads and cache sizes that would inevitably emerge when using large, monolithic programs with the particular characteristics of the MPPB.

At present, activities are carried out by Thales Alenia Space España and Recore Systems in an effort to create the Scalable Sensor Data Processor (SSDP) hardware, where an ASIC is being developed based on the MPPB *2.0*, which is an update of the original MPPB with adapted specification [2]. This new implementation was made available to UVIE in Q1 2016 as a firmware update to the existing MPPB hardware box.

In order to support this new hardware, a more refined version of the experimental operating system is under development at the University of Vienna under a nationally funded ASAP 11 project, which also aims to become space-qualifiable, supporting applicable documentation and S/W standards.

The software is tailored to the NoC concept present in the SSDP and is optimised for best performance in key areas of system and resource management. These include fast and efficient interrupt handling to ensure low response times and high memory throughput for DMA transfers that service the Xentium data caches and fast I/O interfaces like SpaceWire or ADC/DAC.

Supporting functionality, for example device drivers, threads and schedulers, timing and a system configuration/information interface will be provided. Great effort is made to keep CPU and memory footprints at a minimum, so the LEON processor is available for duties other than DSP and data processing control, such as handling of tele-commands or instrument-related control tasks.

A major aim is to make the operating system as easy to use as possible, by providing appropriate, well designed interfaces in order to keep the need for configuration and extra programming effort at a minimum.

To encourage use, modification and redistribution of the operating system, it will be made available under an open-source license, including all drivers, modules and example DSP program kernels, as well as the documentation.

III. SSDP/MPPB 2.0 Hardware Overview

The MPPB 2.0 (hereafter referred to as just MPPB) platform is a representative "preview" of the future SSDP hardware. It consists of two VLIW DSPs, called Xentiums, which are connected to a high-speed Network-on-Chip (NoC) along with distributed SDRAM memories and external high-speed interfaces, such as SpaceWire, to satisfy requirements for space-based platforms. Attached to the NoC is a conventional AMBA bus, which serves as an inter-connect for a LEON GPP. The LEON is intended to control, manage and serve the nodes of the NoC and other payload oriented interfaces (e.g. the real time clock). It can also be used to run legacy software for satellite control operations beside its NoC servicing tasks. The system is clocked at 50 MHz.

A. *Network-on-Chip*

In high-performance multi-core computing, input/output bandwidth and data transport capability are most critical issues. In the MPPB, this is addressed by a Network-on-Chip (NoC), which is a packet-switched network based on an XY routing scheme. XY routing is a simple method of routing packets through a network, where first the horizontal (X) direction is taken, followed by a turn to the vertical (Y) path at the targets X location. For this reason, the forward and return paths are different most of the time and are guaranteed to be safe from deadlocking.

The 3x3 NoC mesh connects the following devices:

- 2 Xentiums,
- a bridge to the ADC/DAC
- an 8-channel DMA controller
- 2 SpaceWire connections
- DDR (SDRAM) controller
- SRAM memory tile
- AMBA Subsystem

Every mesh routing node has 5 ports and serves 4 channels per port, each of them with different priorities. A channel offers a bandwidth of 1.6 Gbit/s at a system clock of 50 MHz. Two high-priority channels are dedicated to DMA transfers, while the low-priority channels serve single read/write operations and interrupts. The high-bandwidth design is important to the NoC concept, which intends to contain all high-volume data flows to the network, never crossing the slow AMBA bridge.

B. *Xentium DSP*

The Xentium is a little-endian Very Long Instruction Word (VLIW) type digital signal processor IP core developed by Recore Systems, The Netherlands.

A Xentium DSP consists of three main parts: the Xentium local bus, the data path (processing core) and a tightly-coupled memory (TCM) bank composed of 4 sub-banks of 8 kiB each. The Xentium local bus is an AHB-like bus that allows the attachment to already existing compatible hardware if needed.

Most instructions work on 32 bit or pairs of 16 bit complements of data-words. The different units offer different functionality:

A0, A1 32 bit and 2x16 bit arithmetic with 40 bit wide add registers

S0, S1 32 bit and 2x16 bit arithmetic with 40 bit wide add registers, shift operations

M0, M1 multipliers for 32-bit operands or 2x16-bit operands

E0, E1 load/store functionality

C0 32 bit and 2x16 bit arithmetic, loop and branch control instructions

P0 32 bit and 2x16 bit arithmetic, compare and packing instructions

The TCM provides access to 4 different memory banks at the same time. As the data path can load and/or store 4x32 bit values simultaneously using these banks, enough bandwidth is available to all different parallel execution units in the Xentium.

IV. Fundamental Requirements of the OS

A set of core prerequisites that are crucial to the usability of an operating system has been identified and are described in short below. These are not unusual for an operating system of this category, the features that are particular or less common are presented in more detail in the next sections.

A. *Interrupts and Traps*

CPU traps are a central element in the run-time environment of the SPARC architecture, they provide means to treat hardware exceptions and interrupt requests. Interfaces to manage and install higher level trap handlers are available and default handlers for different traps typed are provided. Effort is made to reduce interrupt entry and exit times as much as possible, as the SSDP tends to have higher rates than comparable systems under load. This is a consequence of necessary signalling with the Xentium DSPs and other platform-specific devices, so reducing even small amounts of systematic overhead can have great effects in the long run.

Interrupt call-back support for both hardware and software interrupts are provided. These will not only allow fast and easy (de-)registration of an arbitrary number of call-backs per interrupt with associated user-data, but also deferred low-priority interrupt handling in a dedicated thread or OS idle loop.

B. Multi-Core Support

In order to make it future-proof and interesting for use with other LEON-based platforms, the OS is written with multi-core support in mind, including dedicated per-CPU interrupt stacks, cross-CPU synchronisation and locking, as well as task migration with according support in the threading library.

C. Timers

In addition to the usual facilities, emphasis is put on tick-less (i.e. non-periodic) timing functionality, so unnecessary wake-ups of the GPP and inherent waste of CPU cycles can be avoided.

D. Threads and Schedulers

Along fixed priority based scheduling, a modified earliest deadline first scheduler with priority execution in overload conditions is implemented. This, along with dynamic ticking, gives an option to optimise thread CPU utilisation with the added benefit of predictable execution for certain high-priority threads in conditions, where the total load unexpectedly exceeds 100%.

E. DMA Driver

The 8 channels of the DMA 2D-transfer feature in the MPPB/SSDP are essential to its computational performance. Low overhead and ease of use are desirable for this driver. Special care must be taken to avoid access conflicts to channels since the Xentiums have gained the faculties to receive transfer-completion signalling with version 2 of the MPPB and can now be used to initiate transfers themselves, thereby reducing the interrupt load of the GPP significantly. As there are (by the nature of the NoC) no atomic loads/stores possible, the usage state of a DMA channel might change unexpectedly during programming, if channels are dynamically used, rather than being statically assigned to either a DSP or the GPP. The former is clearly more desirable, as there is less downtime when more transfers need to be started than channels are assigned to a node.

F. I/O Interface Drivers

The major I/O devices, i.e. ADC/DAC and SpaceWire that are common to both the MPPB and the SSDP are supported, others (FLASH, GPIO, LEDs, LCD, ...) as they are present or needed for OS operations or development support.

G. FDIR and Error Reporting

Fault detection and recovery with regard to hardware devices is part of the drivers themselves. EDAC handling and memory scrubbing is present as part of the OS. A central error reporting facility is in place that is being used by drivers or other software components.

H. Miscellaneous

Additional functionality to support application software development is available. This includes an interface to the debug support unit (DSU) of the LEON, generation of stack traces and register dumps on hardware traps, along with any NoC/Xentium focused debugging facilities.

V. Xentium Kernel Scheduler

Within the NoC of the MPPB, functional components may be viewed to behave similarly to hosts in a computer network. Any data transferred between nodes of the NoC, even dedicated memories, are sent via datagrams. This means, for example, that a data load from an SDRAM bank executed on a Xentium node is executed via its Xentium Network Interface (XNI), which effectively constructs a request packet that is sent to the SDRAM node. The receiving node then reads the requested memory locations and sends a number of packets holding the data back to the DSP. The communication overhead and subsequent packet creation time generated for every single request of a program instruction or data word read from a larger memory store inevitably inserts significant latency into every operation of the Xentium that requires external interaction, as the possible throughput is 4x32 bit words per clock cycle, if the DSP program is properly written. A way to avoid these delays is to restrict Xentium memory access to the local TCM banks and, in order to forgo stalls in the instruction pipeline, restrict program sizes to be at most the size of the local instruction cache (16 kiB).

The contents of the TCM can be exchanged with bulk memory via the DMA feature of the MPPB, as of version 2.0, transfers can also be locally controlled by the Xentium. The DMA function is essentially the same feature that is used for data transfer in the opaque XNI, but may be used to initiate larger, more complex (2D) data block transfers, so network overhead is minimized and transfers can happen at much higher rates, limited only by the mass memory throughput and, to a lesser extent, NoC bandwidth.

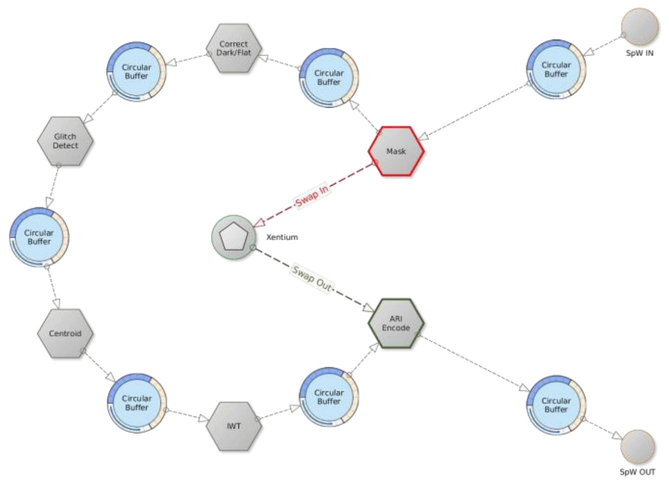

Figure 1 Chaining concept of individual, pipelined program kernels. Data arriving via a SpaceWire link are processed by Xentium DSPs as needed by dynamically changing the running kernel. The data progresses through the pipeline and are sent to their destination via outgoing SpaceWire link.

On-board processing pipelines, at least in the astronomical use cases that were explored in the NGAPP performance study, typically require many steps in data treatment, resulting in binary sizes that easily exceed the capacity of a Xentium's instruction cache. Instead, the monolithic program code can be broken down into arbitrarily small functional fragments (kernels) that are executed on the Xentium DSP as they are needed (see Figure 1). Such a science data processing chain is briefly described in [3]. Each step in there would be implemented in the SSPD as a processing kernel. These kernels require a generic data exchange interface for input and output, so data can be passed between arbitrarily chained processing nodes. This is done via dynamically defined metadata containers, which hold information about data type, references to location and size, previously applied processing steps and other configuration parameters, thus allowing the receiving kernel to act on the input data accordingly and to record its own data operations to the container when passing it on. In between operations, the metadata containers are held in circular buffers, which act as both a connecting intermediate and a measure of the state of the pipeline.

A. Scheduling

Since Xentium kernels act upon their input only as a link in a chain and do no further processing than what is their purpose, they must occasionally be exchanged, or the pipeline would stall eventually, because either the output of the kernel would run full, or the input would run empty. This is a task that is supervised by the MPPB's LEON GPP. A very simple, yet effective metric is used to determine whether the DSP should be assigned another kernel.

During pipeline construction, each kernel is assigned an input and an output circular buffer, which is configured with two parameters: total size and critical fill state. The latter is used as a threshold trip point that results in a scheduling event signal when it is exceeded. The signal is emitted by the circular buffer itself, hence no periodic polling overhead is generated on the GPP and as long as the critical level is sensibly defined, it provides enough hysteresis for the pipeline not to stall. This applies to all but the last buffers in the processing chain, which is ignored, or rather, has no critical fill state, since its contents are typically sent to a bulk storage device or via a network interface.

On a buffer criticality signal, the kernel scheduler selects the most critical buffer based on its location in the pipeline, with later buffers having less priority. It then selects a Xentium based on their kernel input buffers fill state and position in the pipeline and switches the running program. This is done so that data are buffered towards the end of the pipeline, rather than the beginning, allowing input to be accepted as long as possible, even if there are issues with output network interface or mass storage device.

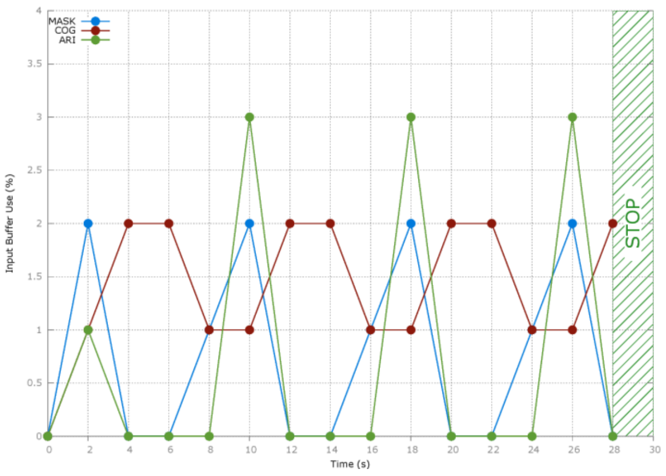

Figure 2: Successful test of a processing chain. Only buffers that show usage > 0 during any sampling period are included in the diagram.

Figure 2 shows a test of the self-balancing nature of this approach. The processing pipeline of a fine guidance sensor and photometer instrument was implemented and fed 512x512 pixel-sized input frames with simulated stars via two SpaceWire links running at 100 Mbits at maximum data rate (~34 frames per second). In the initial processing step, a region of interest of 100x100 pixels was masked, which was then examined by a center-of-gravity (COG) algorithm to determine the precise position of the guide star on the frame. The output of the COG step consisted of the object shift relative to the center of the input frame and photometric flux data for a 40x40 region of interest. This region was deglitched and calibrated in the next nodes of the processing chain, followed by de-correlation via integer wavelet transform and finally compressed by arithmetic coding (ARI).

The resulting load curves, represented by the fill states of the circular buffers, demonstrate the quick emergence of a periodic pattern that clearly demonstrates the effectiveness of this approach (see Figure 2).

VI. Run-Time Configuration Interface

A core necessity of any type of on-board software is the ability to generate housekeeping data to be sent to ground, in order to provide information about the prevailing run-time parameters of both hardware and software.

While requirements of update rates and number of variables – especially regarding software – may vary greatly for different mission profiles, there are generally hundreds of these data that are available for selection to form a housekeeping telemetry message. Usually, these are not solely read-only variables, but may also be patched by an appropriate tele-command in order to induce a mode change or adjust parameters to modify the behaviour of the software.

These variables are often stored in large, monolithic, globally accessible "data pools". Such simplistic structures may at first glance be the logical choice, suggesting ease of both use and implementation, but are however very susceptible to breakage, particularly in top-down designs, where the data type of the implemented variables is not uniform and interaction with the data structure is only intended to occur via opaque accessor functions. If adjustments are made during development, memory violations may occur during runtime, and those can result in erratic, unpredictable, opaque bugs that are very difficult to track down. Another objection to this type of design is its re-usability, as there may exist multiple points of adaption, especially in circumstances where a great number of internally used variables, which are elemental to a software module or function, are stored in an externally administered data structure.

Highly modular, encapsulated software modules with an as minimalistic as possible external interface are very preferable for re-use. Ideally, for example, a SpaceWire driver would only provide an interface to send or receive packets and handle all configuration of the underlying hardware internally. This however poses a problem to a user that would, for example, configure a particular link speed or continuously monitor data transfer rates.

For such purposes, an interaction point is needed that exposes certain internal attributes via a generic interface and acts as a conduit between operating system elements and user-space. There are essentially four fundamental requirements for such functionality. First, internal interfaces or variables must not be slower to use than when not exposed. Second, all exposed functionality is defined by the module and exported to the generic interface when initialised. Third, the exposed functionality must not result in unpredictable behaviour, i.e. the software module must be insensitive to sudden changes in states or variables, or care must be taken by the module designer, so that interactions are properly handled. In any case, this must never be a concern for the user. Finally, any access must be on the user's processing time, not on that of the module.

Given that the interaction point has to be completely generic to accommodate any kind of mapping defined by a module without restrictions, it must consequently be very simple. This is most easily achieved by implementing a character-buffer based interface that interacts with a module via functions provided by the latter to the generic interface structure. The necessary parsing or value conversion of text buffers on the user side is obviously slow compared to raw variable access, but given the underlying assumption that this system control interface is to be accessed in the order of no more than a few hundred or at most several thousand times per second, the overhead is effectively negligible.

The concept is very similar to the *sysfs* and *sysctl* interfaces found in Linux and BSD operating systems, with the former being file-system driven, while the latter is implemented as a system call. Since a file-system in the classic sense is not foreseen to be implemented in the OS, the actual implementation can be seen as a hybrid of the two, which represents nodes in the configuration in the same fashion as a virtual file system tree, while all access is performed via a call interface.

To create a *system object* for exporting items, a software module must define at least one attribute structure that configures the name and the appropriate *show* and *store* methods of that attribute. The object is then registered to an existing logical *set* of objects. For instance, a SpaceWire driver would register its attributes under a */sys/drivers* tree, while an interrupt manager would register under */sys/irq*, provided that these sets were already defined. Optionally, a new sub-set to hold the system objects of particular attributes may be created before attaching an object. If the SpaceWire driver was to manage multiple interfaces, it could create a logical sub-set */sys/drivers/spw* and group interfaces *SpW0*, SpW1, ... under that set.

Since there are no formal restrictions on what qualifies to this system configuration tree, application software running on top of the operating system can (and should) make use of it as well. The aforementioned housekeeping data generation makes a good example for an application that both uses the the data provided by the registered software modules to generate housekeeping packets and is itself configured via this interface, e.g. its polling rate and the definition of housekeeping data to collect.

VII. Summary

Given the unique nature of the SSDP/MPPB hardware concept, a custom approach is needed to efficiently run computational operations in an (astronomical) on-board data processing and compression setup of instrument payloads. The operating system currently under development at the Department of Astrophysics of the University of Vienna addresses this challenge. To encourage its use, modification and redistribution, it will be published under an open source license in all of its parts.

VIII. References

[1] Massively Parallel Processor Breadboarding Study, ESA Contract 21986, Final presentation, ESA DSP Day, (2012) Available: http://www.spacewire.esa.int/edp-page/events/DSP Day - RECORE MPPB presentation - part 1.pdf [Online; accessed 13-May-2016].

[2] Berrojo, L. et al. (2015, 09). Scalable Sensor Data Processor: A Multi-Core Payload Data Processor ASIC. DASIA 2015

[3] Ottensamer, R. et al. Open-Source Instrument Flight Software for CHEOPS. AMICSA & DSP Day (2016)