Multi-Core Software Considerations

Application note 2015-10-28

Doc. No GRLIB-AN-0005

Issue 1.0

CHANGE RECORD

Issue	Date	Section / Page	Description
1.0	2015-10-28		First issue.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Scope of the Document

This document contains general notes of on handling uncorrectable errors in LEON/GRLIB system-on-chip devices.

1.2 Reference Documents

[RD1] GRLIB IP Core User's Manual, latest version: http://gaisler.com/products/grlib/grip.pdf

2 MULTI-CORE SYSTEMS

2.1 Background

Traditionally, LEON/GRLIB systems have consisted of a LEON processor, a memory controller and peripheral units connected to one shared bus. The LEON3 and LEON4 processor models also support operation in multi-core systems. Several commercial multi-core systems exist and devices for harsh environments such as the GR712RC and GR740 are also available.

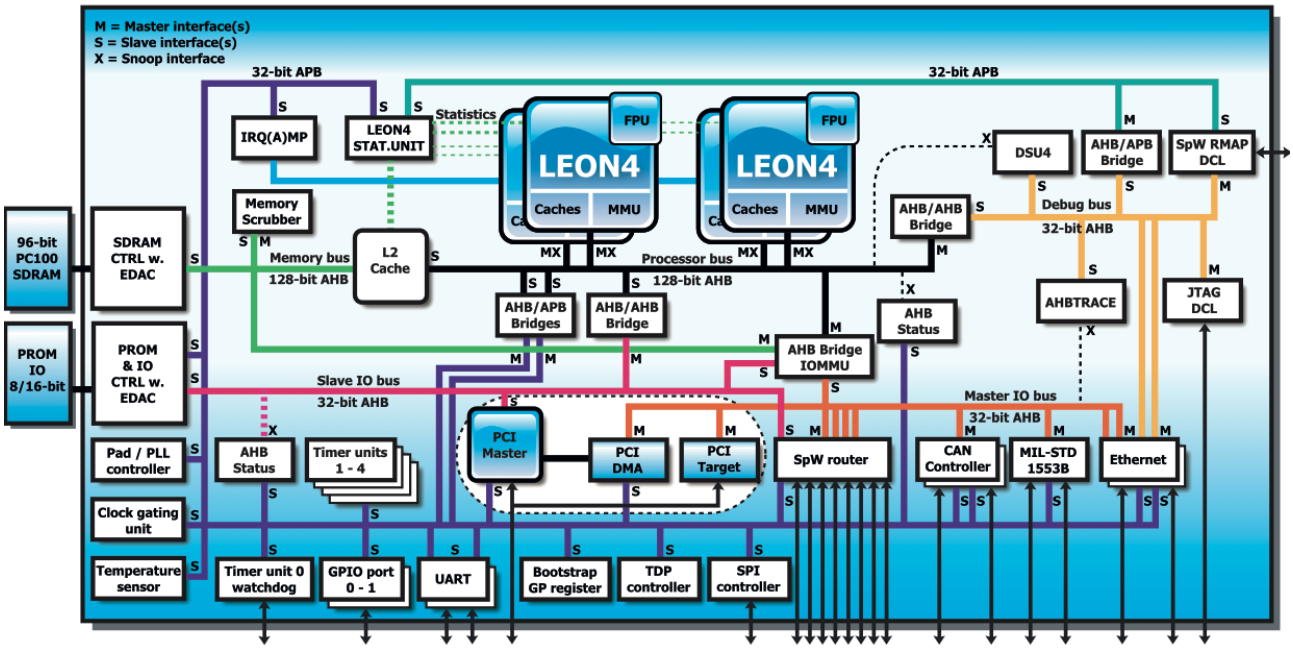

Figure 1: GR740 system-on-chip block diagram

A multi-core LEON system means that the design contains several individual LEON processors. These processors are typically connected to a shared bus and share the same memory. The LEON processor documentation in [RD1] contains additional information about the LEON processor memory model. The term multi-core system is synonymous to multi-processor system in the context of LEON3 and LEON4 devices.

2.2 Making use of LEON Multi-Core

In practice it is not possible to use technologies such as parallelizing compilers in order to convert any software program to make use of multiple cores. To make use of multiple cores (processors) it is required that the software is designed for this purpose with clearly divided entities that can be run in parallel on different processors. A run time environment or operating system can allow to run several instances of software designed for uni-processor systems in parallel but a single invocation

of the software application will not make use of more than one processor. However, the operating system may select to schedule its internal tasks on one processor while dedicating another processor to the single-core application, giving a performance boost for the application compared to running the same application on the same operating system on a uni-processor system.

Software can make use of a multi-core system in several different ways:

- Use as a **single-core system (uni-processor system)**. In the typical LEON/GRLIB system, all processors except processor 0 are idle after reset. Unless software enables the remaining processors then they will remain idle in power-down state.

- Run different software instances on each processor. For example, one processor may run an bare-metal image while another processor is running an operating system such as RTEMS.

  In this configuration, care must be taken so that the different software images do not act as they have exclusive access to shared resources. One example of this is that the different software images likely need to be assigned dedicated timer units and UART peripherals. In systems such as the GR712RC, the interrupt controller is a shared resource and software will need to be adapted so that one software image does not affect interrupt handling of the other software image.

  Configurations with one separate software image per processor are referred to as **Asymmetric Multiprocessing (AMP or ASMP)** configurations.

- Run one software instance on several processors. In this case the operating system or run time environment is able to handle multiple processors and make use of more than one processor for executing tasks.

  When one software image is running on several processors the configuration is referred to as **Symmetric Multiprocessing (SMP)**. Operating systems that support this include RTEMS, VxWorks and Linux.

Is is also possible to mix the configurations above. For example, in a quad-core system the user may select to run an SMP operating systems on the two first processors and have standalone software images of another type running on the third and fourth processor. Please note that such configurations are limited by the available peripherals in a design. Devices such as the GR712RC requires that software shares the interrupt controller between software images while devices such as GR740 allows the user to allocate one interrupt controller per software image.

2.3 Constraints of LEON Multi-Core Systems

In a multi-core LEON3/LEON4 system, each core is a full processor with its own instruction pipeline and register file. The processors share access to the on-chip bus, memory and I/O peripherals. They also may or may not share the interrupt controller and timers, this depends on how the LEON system has been designed.

There is no way for one CPU to directly access the other CPUs internal registers. Therefore it's not possible to "slice up" code so the CPUs run every other instruction in the same code sequence, since each instruction can depend on registers that were written by previous instructions they all must run on the same CPU.

One way to take advantage of multi-core is to parallelize programs, ie make them run as multiple independent tasks or threads that run asynchronously to each other. These different tasks can then be put by the OS to run on different CPUs. They tasks can share data by using the same memory buffer (that may need to be guarded by data structures to prevent coherency problems).

Use of an SMP operating system like Linux will automatically provide multi-core support, all the processes run will get distributed over the different cores automatically.

SMP operating systems use the fact that on a task switch you dump the whole state of the CPU and then reload the state belonging to the new task. Since this is done to/from memory it doesn't have to be on the same CPU. Therefore it can rebalance the processes so that there is a running thread on each CPU (unless there are more CPUs than runnable threads, then the additional CPUs are put to sleep until the next time slice).

Cobham Gaisler's Bare-C runtime (BCC) does at the time of writing not support multi-core operation, it was designed for single-core use and assumes that it is the only CPU on the system and has alone full access to the system and all peripherals, and it may crash if someone else also accesses a shared resource at the same time. Multi-core support requires extra design work since software has to manage possibility of concurrent access to all shared resources (interrupts, timers, peripherals, memory).

The closest to bare metal is to run RTEMS in AMP mode. The quickest way to be up and running with multi-core is to use Linux.

2.4 Considerations When Using Multi-Core

Integration of multiple processors into one device and connecting them to a shared memory controller has several positive effects:

- Increased maximum processing performance over a uni-processor system due to the addition of execution units.
- Possibility of moving more software functions into one device, reducing device count.
- In addition to mass and power savings from reduced device count, the software instances running on the same device will share the same memory, possibly further reducing device count and power requirements.

Introducing additional processors in the same system may also have negative effects:

- In traditional uni-processor systems, DMA traffic from communication controllers such as Ethernet or SpaceWire may contend for bus access with the processor. Causing jitter in software execution. When introducing more processors in a system, the load on the bus is typically increased since a CPU may put a high load on the bus (access the bus frequently). This leads to additional jitter in the system.
- Processors may contend for shared resources, such as memory, and this reduces the maximum performance. It may also complicate execution time analysis of the system. It is possible to run a multi-processor system and never have two processors trying to access the bus at the same time. During software's lifetime it may also be possible that several processors reach a part of their programs that lead to heavy bus load and each access the processors tries to perform collides with accesses generated by the other processor.
- Several analysis techniques have been proposed for multi-core systems. One way is to count the number of memory accesses performed by each application/processor and assume that each access will contend for the bus with other processors, up to the total number of accesses performed by the other contender(s).

Several aspects need to be considered when designing for a multi-core system:

- The effects listed above should drive software design. It is necessary to analyse software images in the context of a multi-core system and not just analyse the software when it is running without contenders.
- In systems with shared Level-2 cache, like the GR740 device, the effects on one software instance from cache line evictions caused by another software instance are considered difficult to analyse. In this case it should be considered to partition the Level-2 cache and allocate one way per software instance.
- When parallelizing applications, scheduling is key to achieving good performance in multi-core system. Tasks need to be divided and scheduled so that all processors are utilized as much as possible.