Booting a LEON system over SpaceWire RMAP

Application note

2017-05-23

Doc. No GRLIB-AN-0002

Issue 2.1

CHANGE RECORD

Issue	Date	Section / Page	Description
1.0	2015-10-30		Conversion of GR-AN-102 (Feb. 18 2014) to this template. Improve Introduction, add Prerequisites.
2.0	2017-04-12		Added examples, added GR740
2.1	2017-05-03		GR740 BREAK signal description Clarified GR740 example flow Fixed spwboot_rmap_GR740.h MUXCFG address (SW bundle) Added troubleshooting section Corrected bundle URL

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Scope of the Document

The SpaceWire RMAP (Remote Memory Access Protocol) provides read and write access to a target node directly by the SpaceWire interface, without any further active involvement of the target hardware. On a LEON system which includes a SpaceWire interface supporting RMAP, the whole AMBA address space visible from that SpaceWire core is accessible, including RAM, ROM, AHB and APB configuration registers.

A target platform can be booted over SpaceWire RMAP, provided it features a RMAP enabled SpaceWire link, and either a Debug Support Unit (DSU) or an interrupt controller supporting processor execution control.

This application note describes the SpWBoot bundle, a collection of scripts, tools and example code for the various steps needed to boot a system over SpaceWire RMAP.

2 ABBREVIATIONS

RMAP	Remote Memory Access Protocol
AMBA	Advanced Microcontroller Bus Architecture

2.1 Methodology

This application note and the associated software provide a simple way to demonstrate one LEON processor development board booting another. The platform to be booted through RMAP is referred to as "target", while the platform that sends the initialization and program data over the SpaceWire link is referred to as "master". The scripts are designed to connect through the GRMON2 debugger to both platforms and to automatically discover the relevant initialization data for the target platform, as well as the hardware configuration of the master platform to generate a master boot application. The master boot application "spwboot" encapsulates the initialization and program data for the target, and composes and sends suitable RMAP packets over the specified SpaceWire link.

In an actual application, the initiator of the RMAP packets might be implemented in various forms. In a lab environment, a PC workstation might be used with a suitable bridge to the SpaceWire network, such as the GRESB Ethernet to SpaceWire bridge. For users in such kind of situations where a second LEON system is not available or desired, the SpWBoot software bundle contains an `examples` directory with the generated include files for different LEON master and target platforms. The relevant files can be used to avoid having to connect to a master platform, and still use the `Makefile` with the "`target`" and "`spwboot`" targets to generate appropriate data for further use with the system. The `README` file contains the log of an exemplary run of a build of the spwboot application. Example files with GR712RC as master were created with UT699 as target, and vice versa.

GRMON2 is used to discover and initialize the master and target platforms, and to automatically generate appropriate C header files for the master boot application. This includes initialization values for the target, as well as information on the SpaceWire interface of the master.

The target application is encapsulated in a standard MKPROM2 boot image. Using a standard boot loader relies on a standard way to initialize the target, and provides features such as support for compression. The target boot image binary is converted to static data and included in the master boot application. The master boot application uploads the target boot image to the target platform RAM over RMAP, into an area that is not allocated by the target application. The target system is started by RMAP writes to the target DSU or interrupt controller, which disables the "break" status of the target processor. The target boot image is executed and the target application is extracted to the target RAM. Once control is transferred to the target application, the contents of the memory area occupied by the boot image becomes irrelevant, and the whole memory is available for the target application.

2.2 Prerequisites

The example Makefile and scripts rely on the GRMON2 tcl feature to discover all system properties necessary to configure the master and target system. To create the data records necessary to initialize the target, and to subsequently create the master application to send the data over SpaceWire RMAP, two makefile targets are executed one by one while being connected via a debug link to the master and target platform respectively.

This implies that both platforms must be in the state they are intended to be when the RMAP boot takes place. The SpaceWire interfaces used must be connected and in run state, with the correct startup clock divisor value set to provide for a 10 Mbps link during startup. Different devices use different bootstrap signals to initialize the CLKDIVSTART register value. The run state clock divisor should be set to result in the desired transfer rate.

GRMON2 probes and initializes a system automatically when connecting to a board. The initialization that the tool performs is equivalent to the initialization done by a boot loader. Some user dependent features are enabled by command line switches, for example to enable EDAC in the memory controller by the `-edac` switch. Since the SpWBoot scripts rely on a properly configured system to generate the initialization data transmitted over RMAP, GRMON2 must be invoked in a way that results in the desired memory configuration. The proper command line options must be set in the `Makefile` as explained below.

The GRMON2, MKPROM2 and sparc-elf-gcc executables must be in the search path.

2.3 Creation of the master boot application

Before the boot application can be created, the `Makefile` has to be configured to reflect target and master platform properties. At least the following adjustments are necessary:

- `GRMONOPT`: Options to connect with GRMON2 to the master and target platforms. The method relies on the system discovery and initialization performed by GRMON2 on the target platform, and the memory configuration register values found on the hardware will be used in the boot process. Therefore, relevant switches should be included to obtain the intended operation, e.g. `-nosram` if SDRAM is exclusively used, at address 0x40000000 (for LEON3 systems)

- `SPWBOOT_TARGET_APP`: The application binary to be uploaded by SpaceWire and started on the target platform
- `MKPROM_GEN_ARGS`: Generic arguments to MKPROM2 for the boot image on the target platform. In particular, `-freq` and `-ramsize` are essential, the latter to achieve a correct stack setup. Further options concerning memory are irrelevant, since the memory controller has already been initialized over RMAP at this stage, and MCTRL configuration registers will not be rewritten.
- `RAM_START, RAM_END`: Information about the system RAM is necessary in order to provide for a correct initialization of the whole EDAC memory. The `-edac-clean` option to MKPROM2 is used to initialize the whole RAM area before and after the boot image.
- `GRSPW_MASTER, GRSPW_TARGET`: SpaceWire links through which the master platform is connected to the target, and vice versa.
- `TARGET_GR740`: Set to 1 if GR740 is the target platform, 0 otherwise
- `GR740_SPWPORT`: Set to the number of the SpaceWire port used to connect to the master. Ignore if the target is not GR740.

The master boot application is created by the following steps:

1. Connect the master platform connected via the debug link
2. `make master`
   - Connects to the master platform and discovers address and CLKDIV settings of the selected interface. Creates `spwboot_master.h`. If no master platform is available or required, `spwboot_master.h` can be copied from the `examples` folder
3. Connect the target platform connected via the debug link
4. `make target`
   - Connects to the target platform and discovers address and CLKDIV settings of the selected interface. Creates `spwboot_target.h`
   - Discovers the address of the DSU and stores values for triggering a series of actions and initializations on the target platform: Break CPU, clear error mode, disable caches and MMU, memory controller registers, UART, IRQ controller and timer addresses, PC, nPC, clear break CPU. Creates `spwboot_rmap.h` and `spwboot.mk`.
   - For the GR740, as a specific target, the corresponding target configuration `spwboot_target_GR740.h` is not generated, but part of the source files. Since

for a specific target generation of files is hardly necessary, the `target-GR740` target can be invoked instead of `target`, which copies the remaining needed files from the `examples` folder.

5. `make spwboot`
   - Invokes `mkprom2` twice to create the target boot image `spw_bootprom`. Creates the master boot application binary `spwboot`. The `obj2array` tool, part of SpWBoot, is invoked several times to convert the target application binary to C structures (`spwboot_rmap_img.h`), and to extract various size and address information from the target application.

The `spwboot` application can be loaded and run on the master platform. The MKPROM2 boot message and the `Target says hello!` message should be printed on the serial console connected to the first UART interface of the target platform. Initialize the PC serial console with 38400 baud, 8 data bit, no parity, 1 stop bit, no flow control.

Note: In order to debug the boot process, it can be helpful to connect with GRMON to the target platform before running the master boot application. GRMON should be started with the `-ni` switch and the board should have been reset or power cycled previously, in order to avoid initialization prior to the boot process. Any PROM memory should not contain code at the default start address. Before starting the boot process on the master, the `detach` command should be issued to release the target processor from GRMON. After the boot process is initiated by the master, the `verify` commands can be used with the `spw_bootprom` and `hello` binaries on the target to check whether the data transfer from the master occurred correctly, and the target application was correctly extracted to its entry point.

Note: Below is an example walk-through of the SpWBundle flow without physically connecting to master (GR712RC) or target (UT699) platform. The example files in `UT699-target` were created with the GR712RC as master:

```
~$  cp examples/UT699-target/Makefile .

~$  cp examples/UT699-target/spwboot_master.h .

~$  cp examples/UT699-target/spwboot_target.h examples/UT699-
target/spwboot_rmap.h examples/UT699-target/spwboot.mk .

~$  make spwboot
```

2.4 Boot process

The transfer of data over RMAP to launch the boot process consists of three stages:

1. Halt the processor and initialize the memory controller and some peripherals
2. transfer the boot image to target RAM
3. set the entry point and resume execution

Stage one and three are defined as C-arrays of 32-bit words as address and data in the file `spwboot_rmap.h`, which is created as a result of the `target` Makefile target. Stage two, the boot image data derived from the `spw_bootprom` MKPROM2 image, is defined as C-arrays of address, data and size in the file `spwboot_rmap_img.h`. Both the elf and header file of the boot image are a result of the `spwboot` Makefile target. The `spwboot` application is built as an example how to send the three stages over RMAP from a LEON based master platform.

2.5 SpaceWire link

The clock divisor register of the target platform's SpaceWire port must be set to the proper value with respect to the core frequency for a functional SpaceWire link. This register is typically reset to a value defined by external pull-ups on power-up. In our scenario, it may have been also set by software running prior to the boot process over RMAP. Please refer to the target component or IP core data sheet for details.

2.6 Target memory layout

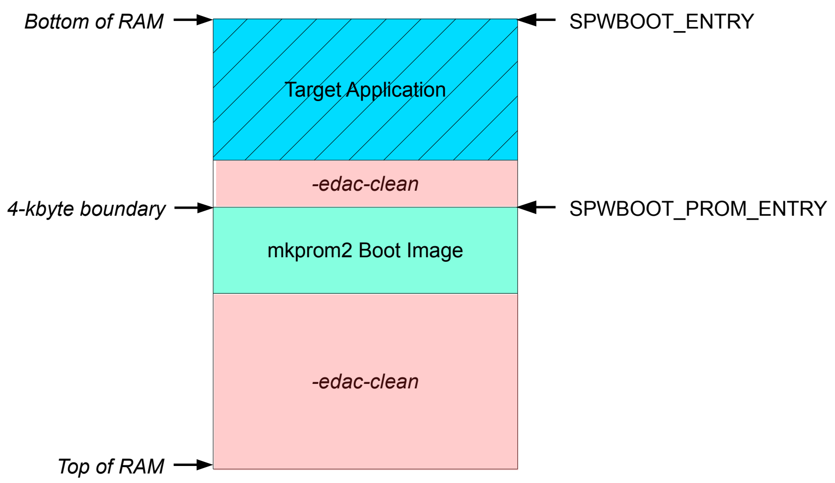

Figure 1: Memory layout of the target RAM

The target application is transferred to the target memory encapsulated in a MKPROM2 boot image. The actual target application will be extracted from the boot image and copied to the beginning of RAM, typically at 0x40000000 for LEON3 systems.

The MKPROM2 image uploaded over SpaceWire is linked to the 4 kbyte boundary following the last allocated address of the target application. In order to safely initialize (“wash”) the whole memory, the RAM areas below and above the boot image are cleared by the boot image using the MKPROM2 `-edac-clean` switch providing the start addresses and sizes of the two areas. The MKPROM2 tool is invoked twice during the compilation process, since the size of the boot image itself and thus the start address of the second area to initialize is only known after the first MKPROM2 invocation.

Reconfiguration of the memory controller during execution of the boot image from RAM must be avoided. The memory controller location is hence set to an arbitrary address by the MKPROM2 `-mctrl` switch in the boot image. The target application entry point is chosen to that end as a defined memory address, unused at this stage of the boot process.

3 SPWBOOT BUNDLE

The SpWBoot bundle is a source file archive (SpWBoot-v2.1.tar.gz) that includes the following items:

- `spwboot.tcl`: A set of Tcl procedures to inspect the master and target platform in order to create C header files for the master boot application. The header files contain appropriate values to set over SpaceWire on the target platform during the boot process, as well as initialization values for the GRSPW driver
- `master_gen.tcl, target_gen.tcl`: Entry points for GRMON2 to the above Tcl procedures
- `spwboot.c`: The master boot application, which includes the boot image to download on the target platform
- `spwapi.c, spwapi.h`: Driver for the GRSPW core
- `obj2array.c`: Tool to convert the boot image to a C-Array for inclusion in the master boot application, and to extract various address information from ELF object files
- `bdinit.c`: Initialization routine to be included to the MKPROM2 boot process. For our purposes, it serves only to initialize the first word in the target memory following the boot image, in case the size of the boot image is not 8-byte aligned. This alignment is required for the routine to initialize EDAC memory (MKPROM2 `-edac-clean` option)
- `Makefile`: Contains Makefile targets to inspect the master and target platforms, generate platform specific configuration header files, create a target MKPROM2 image, and create the master boot application.
- The remaining files are specific to a target or generated. Examples of those are available in the `examples` folder.

4 GR740 TARGET

For the GR740 quad-core LEON4FT device, several items in the boot process are different from most LEON3FT devices, which typically share the properties addressed by the boot flow described in the previous chapters:

- The IRQ(A)MP interrupt controller core implements processor execution control. In the first stage of the boot process, the DSU does not need to be used to stop and start execution and to set the entry point.

- The memory controller requires a disable-enable initialization sequence to enable 2T-signalling. This is done in the first stage of the boot process.
- The level-2 cache must be turned off and invalidated before transferring the boot image. This is done in the first stage of the boot process.
- The level-2 cache needs to be enabled and the I/O pin multiplexing must be initialized for the first UART. This is implemented in the `bdinit` functions of MKPROM2.
- The GRSPW2 (AMBA) ports with the RMAP target are located behind the SpaceWire router. An additional path address and return address must be added to the RMAP packets in the `spwboot` application.

The necessary settings for correct generation of the master boot application and other files are derived from the GR740 relevant settings to be edited in the `Makefile`. It is recommended to use the `target-GR740` target instead the `target` target to simply copy the relevant files from the `examples` folder. Unlike with LEON3FT systems, where many different hardware configurations are already available, the existence of only one or few LEON4FT devices today makes generation of files unattractive.

Note: Below is an example walk-through of the SpWBundle flow without physically connecting to master (GR712RC) or target (GR740) platform. In the listing, Makefile and the master configuration are copied from the examples. The example files in the `UT699-target` folder were created with the GR712RC as master. The `target-GR740` target copies the target configuration from examples and must not be omitted before creating the master boot application:

```
~$  cp examples/GR740-target/Makefile .
~$  cp examples/UT699-target/spwboot_master.h .
~$  make target-GR740
~$  make spwboot
```

The `spwboot` application can be loaded and run on the master platform. The MKPROM2 boot message and the `Target says hello!` message should be printed on the serial console connected to the first UART interface of the target platform. Initialize the PC serial console with 38400 baud, 8 data bit, no parity, 1 stop bit, no flow control.

Note: The processor must not be in debug mode when the remote boot takes place. Debug mode is enabled either by GRMON or by an asserted BREAK signal during reset when the DSU_EN signal is asserted (high).

During reset or power up with DSU_EN high, the BREAK signal of the GR740 must not be asserted (the BREAK switch on the GR-CPCI-GR740 board must be set to the right position). When DSU_EN is low during reset or power up, an asserted BREAK signal causes the first processor to halt after reset, and disables the watchdog timer (but not the PLL watchdog). This is a suitable configuration in a flight environment.

GRMON should not be connected to the target, or if connected detach from the CPUs with the `detach` command. In that case, GRMON should have been started with the `-ni` switch to avoid a target in pre-initialized state. The processor remains in debug mode when GRMON is quit, unless previously detached.

The clock gating unit setting for the SpaceWire router core is initialized by the GPIO[11] input at reset. GPIO[11] must be bootstrapped low to enable the clock after reset.

No boot code should be in the PROM area to avoid a target in pre-initialized state.

MKPROM2 version 2.0.61 and earlier include the 0x40000000 offset of the classic LEON3 memory layout into the stack calculation from memory size, even if the `-sparcleon0` option for applications starting at address 0x0 is given. This stack pointer setting does not necessarily fail with a BCC application because the memory area can wrap around. However, until the release of later MKPROM2 versions with this bug corrected, the `-stack` option should be added to `MKPROM_GEN_ARGS` in the `Makefile`. For example with 128MiB of SDRAM, use `-stack 0x7ffffe0`. The `-ramsize` setting is overridden by `-stack` and becomes irrelevant.

5 TROUBLESHOOTING

Typical sources of problems on the SpaceWire link include the following:

- Clock divisor settings at master (initialized by master application) and target (bootstrapped with external signals)
- Target SpaceWire links support RMAP and are enabled at reset
- Path to target, target destination key

If data transfer to the target occurs but the boot process does not succeed, useful approaches to debug include

- Connect to the target platform after a failed boot process, without initialization by GRMON: `$~ grmon -ni`
- Verify registers initialized in stage 1, in particular memory configuration registers
- Check whether the target bootprom was transferred correctly: `grmon2> verify spw_bootprom`
- Check whether the target application was extracted correctly: `grmon2> verify hello`. If the application was executed, verification errors in the initialized data section are expected.
- The target bootprom `spw_bootprom` must run when loaded directly to the target. Since the target must be initialized by GRMON for that test, this is a necessary but not a sufficient requirement. Typical errors in the bootprom include faulty memory controller configuration register settings, wrong target frequency, and a wrong target memory size which leads to a bad stack pointer.

If further debugging of the boot process with GRMON is required, the following approach can be suitable. All data that is normally transferred by RMAP to the target platform is written over the debug link by GRMON:

- Connect to the uninitialized target, without initialization by GRMON: `$~ grmon -ni`
- Perform all writes of stage 1, as in the corresponding address and data arrays defined in the `spwboot_rmap` header file, with the `wmem` command
- Load the target bootprom with avoiding memory controller initialization by GRMON: `grmon2> load -nmcr spw_bootprom`
- Set a hardware breakpoint to the target application entry point: `grmon2> bp hard 0`
- When the breakpoint hits. load the application symbols with `grmon2> symbols hello`. No errors should occur with `grmon2> verify hello`. Application debugging can begin with setting breakpoints etc.
- Resume target application with `grmon2> cont`