

MK PROM2 User's Manual

Table of Contents

1. MKPROM2	3
1.1. Introduction	3
1.2. Source code	3
1.3. Usage	3
1.4. Creating ROM resident applications	4
1.5. Internals	4
1.6. MKPROM2 general options	5
1.7. LEON2/3/4 memory controllers options	7
1.7.1. General options	7
1.7.2. Wait states options	8
1.7.3. Explicit memory controller parameters	8
1.7.4. EDAC output file options	9
1.8. LEON3 options	9
1.9. DDR/DDR2 controller options	10
1.10. SDCTRL64/FTSDCTRL64 controller options	11
1.11. FTAHBRAM controller options	11
1.12. SDCTRL controller options	11
1.13. SPI memory controller options	11
1.14. Custom controllers	11
1.15. Timer initialization	12
2. Examples	13
2.1. Quick start	13
2.1.1. Extracting mkprom2 parameters from GRMON	13
2.1.2. Loading boot image to ROM	13
2.1.3. Application start on system reset	14
2.2. Customizing with <code>bdinit.o</code>	14
3. Support	16

1. MKPROM2

This document describes MKPROM2 boot image generator.

1.1. Introduction

MKPROM2 is a utility program which converts a LEON RAM application image into a bootable ROM image. The resulting bootable ROM image contains system initialization code, an application loader and the RAM application itself. The RAM application is compressed with a modified LZSS algorithm, typically achieving a compression factor of 2.

The system initialization code and the application loader operates in the following steps:

- The register file of IU and FPU (if present) is initialized.
- The memory controller, UARTs and timer unit are initialized according to the specified options.
- The RAM application image stored in ROM is decompressed into RAM.
- Finally, the application is started, setting the stack pointer to the top of RAM.

MKPROM2 can create ROM images for ERC32 and LEON2/3/4 systems.

The words ROM and PROM are used in this document to denote normally non-volatile memories such as ROM, PROM, EPROM, EEPROM, Flash PROM, MRAM etc.

The word RAM is used in this document to denote normally volatile memory such as SRAM, DRAM, SDRAM, and sometimes DDR and DDR2 SDRAM.

1.2. Source code

MKPROM2 comes with full source code included. The source code is located in the `<mkprom-dir>/src` directory.

Requirements for building MKPROM is a host C compiler and the GCC 3.4.4 version of BCC 1.0.x.

To recompile mkprom issue a **make** command inside the source directory. This will compile MKPROM2 into the default location, which is `/opt/mkprom2` on Linux and `c:/opt/mkprom` on Windows. On Windows you should use the MINGW/Msys compile system.

1.3. Usage

mkprom2 is a command line utility that takes a number of options and files to encapsulate:

```
mkprom2 -freq <mhz> [options] files
```

To generate a boot-prom for a typical system with 50 MHz system clock, use:

```
$ mkprom2 -freq 50 -v -rmw -ramsize 1024 hello
```

which generates terminal output similar to

```
MKPROM v2.0.62 - boot image generator for LEON applications
Copyright Cobham Gaisler AB 2004-2017, all rights reserved.

phead0: type: 1, off: 65536, vaddr: 40000000, paddr: 40000000, fsize: 27584, msize: 28008
section: .text at 0x40000000, size 26272 bytes
Uncoded stream length: 26272 bytes
Coded stream length: 14091 bytes
Compression Ratio: 1.864
section: .rodata at 0x400066a0, size 128 bytes
Uncoded stream length: 128 bytes
Coded stream length: 38 bytes
Compression Ratio: 3.368
section: .data at 0x40006720, size 1184 bytes
Uncoded stream length: 1184 bytes
Coded stream length: 572 bytes
Compression Ratio: 2.070
Creating LEON3 boot prom: prom.out
[...]
Success!
```

When executed, the PROM loader prints a configuration message at start-up:

```
tsim> run

MKPROM2 boot loader v2.0.62
Copyright Cobham Gaisler AB - all right reserved

system clock : 50.0 MHz
baud rate : 19171 baud
prom : 512 K, (2/2) ws (r/w)
sram : 1024 K, 1 bank(s), 0/0 ws (r/w)

decompressing .text
decompressing .data
decompressing .jcr

starting hello

Hello world!
```

It is essential that the same `-mflat`, `-qsvt` and `-msoft-float` parameters are given to `mkprom2`, as was used when the binary was compiled.

1.4. Creating ROM resident applications

`mkprom2` can also create applications that execute from ROM, and have data and stack in RAM. This is only supported for applications developed with BCC version 1.

A ROM resident application is created in two steps:

- Compile the application into one or more object files, but do not link:

```
sparc-elf-gcc -msoft-float -c -g -O2 hello.c
```

- Create the ROM image with `mkprom2` and the `-romres` option, listing all object files on the command line:

```
mkprom2 -romres -freq 40 -rmw hello.o -msoft-float
```

A ROM resident application has its code (`.text` segment) in ROM, and data (`.data` and `.bss`) in RAM. At startup, the `.data` segment is copied from the ROM to the RAM, and the `.bss` segment is cleared. A ROM resident application is linked to start from address `0x0`. The data segment is by default linked to `0x40000000`, but can be changed by giving the `-Tdata=<address>` option of `gcc` to `mkprom2`. Note that if no FPU is present, the `-msoft-float` option must also be given to `mkprom2` in this case since it is needed during the final linking.

When debugging ROM resident applications with GRMON or `gdb`, hardware breakpoints (`hbreak` command) have to be used. ROM resident applications cannot be compressed, and thus the `-nocomp` option is implied.

When generating a ROM resident image, a symbol image with name `<ofile>.sym` is created that can be used for debugging. The actual ROM output image `<ofile>` does not contain symbol information.

As mentioned earlier, ROM resident MKPROM2 images are supported only for application developed with BCC version 1.0.x. The reason for this is because MKPROM2 utilises knowledge about the application run-time and patches itself into the initializations. Such shared knowledge is not available for other run-times.

1.5. Internals

`mkprom2` is delivered with source code. `mkprom2` is compiled from source file `mkprom.c`. `mkprom2` creates a PROM image through the following steps:

- Parse option switches
- Calculate the register initialization values from the switches.
- Read in elf-format object files and extract load location and section data from it.
- Dump register values and sections data into a file called `dump.s`. You can preserve and read this file using the `-dump` option.

- Use the crosscompile toolchain to compile `dump.s` and link this file against the boot-loader object files. You can see the command that is issued by adding the `-v (-V)` switch to `mkprom2`.

The MKPROM2 binary distribution includes precompiled object code which is used when linking the boot loader. It is available in the installation subdirectory named `lib/`. The object code has been compiled with workarounds enabled for UT699. Compiling for UT699 represents a conservative approach which includes workarounds for the other LEON components. In addition, workarounds for the following technical notes have been taken into account for the included object code:

- GRLIB-TN-0009
- GRLIB-TN-0010 (MMU is not enabled when MKPROM boot loader executes)
- GRLIB-TN-0011 (MMU is not enabled when MKPROM boot loader executes)
- GRLIB-TN-0012 (FPU operations are not performed by MKPROM boot loader)
- GRLIB-TN-0013 (FPU operations are not performed by MKPROM boot loader)

There is in general no need for the user to recompile the MKPROM boot loader target object code. Advanced who want to compile the target libraries can specify compiler options in the file `src/multilibs_gcc-3.4.4`.

The *MKPROM boot loader run-time* is self-containing and does not bring in startup-code provided by the tool chain used to link the boot image.

1.6. MKPROM2 general options

In GRMON, the command **info mkprom2** is available for extracting `mkprom2` parameters for memory controller, timer, uart and interrupt controller. The extracted parameters can be used as a starting point for `mkprom2` and should be verified against the target system by the user.

```
grmon2> info mkprom2
Mkprom2 switches:
-leon3 -freq 48 -rmw -ramsize 8192 -sdram 128 -sdrambanks 2 -trfc 83
```

For hardware without an FPU, the `-msoft-float` has to be given to `mkprom2`. Note the FPU registers will be cleared regardless of the `-msoft-float` flag if an FPU is present, however the FPU will be turned off when entering the application if `-msoft-float` has been given.

Table 1.1. Linking options

Option	Description
<code>-msoft-float</code>	Link for application not using FPU.
<code>-mflat</code>	Link for application using the GCC 3.4.4 flat register window model.
<code>-qsvt</code>	Link application using the single vector trapping model.

Table 1.2. General options

Option	Description
<code>-leon2</code>	Generate a LEON2 executable.
<code>-leon3</code>	Generate a LEON3 executable. This is the default.
<code>-erc32</code>	Generate a ERC32 executable.
<code>-agga4</code>	Generate an AGGA4 executable.
<code>-baud baudrate</code>	Set rate of UART A to baudrate. Default value is 19200.
<code>-bdinit</code>	The user can optionally define three hook routines, <code>bdinit0()</code> , <code>bdinit1()</code> and <code>bdinit2()</code> , which are called by CPU0 during the boot process. <code>bdinit0()</code> is called before and <code>bdinit1()</code> is called after the LEON peripheral registers have been initialized but before the memory has been cleared.

Option	Description
	<p>bdinit2() is called after the memory has been initialized but before the application is loaded. Note that when bdinit0() and bdinit1() is called, the stack has not been setup meaning that bdinit0() and bdinit1() must be leaf routines and not use RAM. bdinit0() is called before FPU registers are initialized.</p> <p>When the option -bdinit is used, a file named bdinit.o must exist in the current directory, containing the three routines bdinit0(), bdinit1() and bdinit2().</p> <p>One additional function, named bdcpuinit0(), can also be defined. This function will be called by all processors at the same boot stage as bdinit0() is called by CPU0. The execution environment is the same as for bdinit0(). However, resources configured by CPU0 may now be available for the other processors. CPU0 calls bdcpuinit0() just before bdinit0().</p>
-ccprefix <prefix>	On startup mkprom2 will search for sparc-elf-gcc, sparc-gaisler-elf-gcc, sparc-rtems-gcc and sparc-linux-gcc. The first found will be used to create the PROM image. The -ccprefix option lets you override the compiler prefix. Example: -ccprefix sparc-elf
-dump	The intermediate assembly code with the compressed application and the LEON register values is put in dump.s (only for debugging of mkprom2). This switch is very useful to verify the calculated initialization values of the registers.
-ecos	Use eCOS realtime library options
-edac	Clear all memory specified by the memory parameters and enable EDAC.
-edac-clean [bank0-addr] [bank0-size] [bank1-addr] [bank1-size]	Explicitly specify the 2 banks [[bank0-addr],[bank0-size]] and [[bank1-addr],[bank1-size]] that should be cleared at bootup to prepare for EDAC enable. If only one bank should be cleared specify 0 as size. The switch -edac has to be given also.
-entry addr	Sets the application.s start address (after decompression). Default: the ELF start address
-freq mhz	<p>Defines the system clock frequency in MHz. This value is used to calculate the divider value for the baud rate generator and timers. This option is mandatory.</p> <p>The parameter mhz can be a floating point number, for example -freq 33.333.</p>
-noinit	Suppress all code which initializes on-chip peripherals such as UARTs, timers and memory controllers. This option requires -bdinit to add custom initialisation code, or the boot process will fail.
-nomsg	Suppress the boot message.
-romres	Create a ROM resident image. See Section 1.4.
-nocomp	Do not compress application. Decreases loading time on the expense of ROM size.
-o outfile	Put the resulting image in outfile, rather than prom.out (default).
-rstaddr addr	Sets the PROM start address. In case of an execute-in-prom configuration addr is limited to 0x0-0x20000000. Default: 0x0
-stack addr	Sets the initial stack pointer to addr. If not specified, the stack starts at top-of-ram.

Option	Description
-sparcleon0	Normally objects with load address 0 will force MKPROM2 into execute-from-rom mode. To avoid this the option -sparcleon0 can be specified. This option can be used if the application was linked with -msparcleon0.
-sparcleon0rom	Use this switch to force creation of a execute-from-rom image for applications with ram-load address 0.
-v	Be verbose; reports compression statistics and compile commands
-V	Very verbose output (as opposed to -v, which is just verbose)
input_files	The input files must be in aout or elf32 format. If more than one file is specified, all files are loaded by the loader and control is transferred to the first segment of the first file.

1.7. LEON2/3/4 memory controllers options

The LEON2/3/4 memory controller options apply to MCTRL, FTMCTRL, SRCTRL and FTSRCTRL.

1.7.1. General options

The options in Table 1.3 are used by the MKPROM2 tool to configure the memory controller. MKPROM2 uses these options to calculate suitable values which are used by the boot code to initialize the memory controller. The options in Table 1.3 are interpreted by MKPROM2 at the time when the boot image is generated.

Table 1.3. Memory controller configuration register options

Option	Description
-cas delay	Set the SDRAM CAS delay. Allowed values are 2 and 3 (default is 2).
-col bits	Set the number of SDRAM column address bits. Allowed values are 8 - 11 (default is 9).
-nosram	Disables the static SRAM and maps SDRAM at address 0x40000000.
-ramcs chip_selects	Set the number of SRAM banks to chip_selects. Default is 1.
-ramsize size	Defines the total available RAM in kBytes. Used to initialize the in the memory configuration register(s). The default value is 2048 (2 MByte).
-ramwidth width	Sets the SRAM bit width to 8, 16, 32, or 39 bits. Default: 32 bits
-refresh period	Set the SDRAM refresh period (in us). This parameter is calculated using the system clock frequency (-freq) and assumes that the system clock frequency and SDRAM clock frequency are equal. Default value is 7.8 us.
-romcs chip_selects	Set the number of ROM banks to chip_selects. Default is 1, possible values are 1, 2, 4 and 8. This options is used by -bch8q where it becomes mcfg1.ebsz.
-romsize kb	Sets the total size of the PROM in kByte. Default: 0x80000
-rmw	Perform read-modify-write cycles during byte and halfword writes.
-romwidth width	Sets the PROM bit width to 8, 16, 32, or 39 bits. Default: 8 bits
-sdram size	The total amount of attached SDRAM in MByte. To use -sdram in the calculation of the stack also specify -nosram. 0 by default
-sdrambanks num_banks	Set the number of populated SDRAM banks (default is 1).
-trfc delay	Set the SDRAM tRFC parameter (in ns). Default is 66 ns.
-trp delay	Set the SDRAM tRP parameter (in ns). Delay defaults to 20 ns. If two system clock periods is shorter than the given tRP value the MCFG2.TRP bit is set to increase to 3 system clocks.

Option	Description
	<p>The formula used is:</p> <pre>IF (2*1E9/FREQ_HZ) < delay THEN set MCFG2.TRP=1 ELSE set MCFG2.TRP=0 END IF</pre> <p>Note that the system clock is used in the calculation, if the SDRAM controller is clocked on a different clock frequency the <code>-mcfg{1 2 3}</code> parameters should be used.</p>
<code>-iowidth width</code>	Sets the IO bit width to 8, 16, or 32 bits. Default: 32 bits

1.7.2. Wait states options

The wait states parameters sets the corresponding wait states field in the memory controller. It shall be noted that the effective number of wait states depends on the target system. In particular, some memory controllers have the synthesis parameter `wsshift` which affects how the wait states field maps to effective number of wait states.

For information on how the wait state fields are translated into actual wait states, consult the component documentation. For custom designs, see the *GRLIB IP Core User's Manual*.

Table 1.4. Memory controller configuration direct options

Option	Description
<code>-ramws ws</code>	Set the SRAM read and write wait states parameter. Default is 3.
<code>-ramrws ws</code>	Set the SRAM read wait states parameter. Default is 3.
<code>-ramwws ws</code>	Set the SRAM write wait states parameter. Default is 3.
<code>-romws ws</code>	Set the PROM read and write wait states parameter. Default is 15.
<code>-iows ws</code>	Set the IO wait states parameter. Default is 7.

1.7.3. Explicit memory controller parameters

An optional way to specify the initialization values for the memory controller configuration registers is to use the options in Table 1.5. The `-mcfg{1|2|3}` options have priority over the parameters in Table 1.3 for memory controller initialization during boot. However, the options in Table 1.3 are still used for calculating the initial stack pointer, RAM wash size and EDAC allocation when using 8-bit SRAM bus.

It is recommended *NOT* to use the `-mcfg{1|2|3}` options at all. Instead go through the memory related options in Table 1.3 and match them with the target system. This also provides a straight forward way to enable/disable individual options by only changing the human readable command line switch.

If however `-mcfg{1|2|3}` options must be used, make sure that to use all of them and do not include any other memory controller switches that influence that same registers.

To get a human readable printout of the current memory controller configuration from GRMON, use:

```
grmon2> info sys mctrl0
[...]
grmon2> info reg -v mctrl0
[...]
```

Table 1.5. Memory controller configuration direct options

Option	Description
<code>-mcfg1 <hex></code>	Specify the <code>mcfg1</code> register directly.

Option	Description
-mcfg2 <hex>	Specify the mcfg2 register directly.
-mcfg3 <hex>	Specify the mcfg3 register directly.

1.7.4. EDAC output file options

The options in Table 1.6 allows generation of additional output files with PROM EDAC check bits.

Table 1.6. EDAC output file options

Option	Description
-bch8	<p>Generate an additional output file <output>.bch8 with a .bch section that contains the EDAC BCH checksums used with 8-bit wide PROM memories. 4/5 of the PROM size is for user data and 1/5 for EDAC BCH checksums. The .bch section is positioned at the end of the PROM (growing in reverse address order). The total PROM size is specified with the -romsize option. The -romcs option must be 1 (default). The -romwidth option must be 8.</p> <p>The 4/5 EDAC scheme is supported by FTMCTRL (e.g. UT699, LEON3FT-RTAX CID-3 through CID-8) and LEON2FT MCTRL (e.g. AT697F, AT9713E/F). Note that only one PROM bank is supported.</p>
-bch8q	<p>Generate an additional output file <output>.bch8q with a .bch section that contains the EDAC BCH checksums used with 8-bit wide PROM memories. 3/4 of the PROM size is for user data and 1/4 for EDAC BCH checksums. The .bch section is positioned at 3/4 of the total PROM (growing in forward address order). The total PROM size is specified with the -romsize option. The -romcs option must be 1, 2, 4 or 8. The -romwidth option must be 8.</p> <p>The 3/4 EDAC scheme is supported by FTSRCTRL (e.g. LEON3FT-RTAX CID-1 through CID-2) for multiple PROM banks, with the EDAC size matching the total PROM size specified with the -romsize option. The 3/4 EDAC scheme is also supported by the old FTMCTRL and the old LEON2FT MCTRL (e.g. AT697E), but only for one PROM bank, i.e. -romcs option must be 1.</p>

1.8. LEON3 options

Currently the following IP cores are detected and initialized using plug and play: DDR2SPA, DDRSPA, SDCTR, IRQMP, APBUART, GPTIMER, GRTIMER, MCTRL, FTMCTRL, FTSRCTRL, FTAHBRAM.

Table 1.7. MKPROM2 options for LEON3

Option	Description
-gpt addr	Sets the address of the timer unit regs. Default: 0x80000300
-irqmp addr	Sets the address of the IRQMP controller regs. Default: 0x80000200. This option is only useful when -nopnp is specified.
-memc addr	Sets the address of the memory controller regs. Default:0x80000000
-mp	Enable multi CPU support. Mutliple stacks, entry points, UARTs etc.
-mpentry ncpu entry1 entry2 .. entryN	Defines the entry points of N CPUs in a multiprocessor system where different entry points are needed, this is typically the case for RTEMS.
-mpirqsel cpu val	In a multiprocessor system specify the value of the TCSELn field of the IRQAMP irq controller's Interrupt Controller Select Register for <cpu>. -mpirqsel can be called several times for each CPU.

Option	Description
-mpstack ncpu stack1 stack2 .. stackN	In a multiprocessor system it may be required to use different stack areas for the different CPUs. This option enables the user to set the stack for each CPU.
-mpstart val	In a multiprocessor system specify a value to write into the MPIRQ status register.
-mpuart nuart UART[1] UART[2] .. UART[N]	Defines the base register address of the first N UARTs. This option is only possible with -nopnp. All uarts defined are initialized with the baudrate given by the -baud option.
-uart addr	Sets the address of the UART base used to output boot messages. Default: 0x80000100
-dsustart addr	Set the DSU start address used by -dsutrace. Default: 0x90000000
-dsutrace	Switches on instruction trace buffer on startup by writing the DSU registers. Default: disabled
-dsubreak	Switches on DSU control register.s BZ bit. Default value written into DSU control register: 0xcf
-nopnp	Switches off plug and play initialization. In this case only mctrl, uart and timer are initialized. Addresses can be specified with -memc, -gpt and -uart or left default. If -ddr2spa_cfg[1 3 4] is supplied, instead of (FT)MCtrl, DDR2Ctrl initialization is performed. Default: pnp enabled
-pnp addr	Define the AMBA plug and play configuration area address where the AHB slave members are located. Default: 0xfffff800

To create a multiprocessor AMP image the options -mp, -mpstack, -mpentry, -mpstart and -mpirqsel can be given. First the user would create different images linked to different RAM addresses. Using the -mpentry option the entry address of each processor can be specified. Processor 0 will handle the setup and decompression, thereafter starting the other processors. The -mpstart option specifies which processors to start. The -mpstack will specify the end-of-stack for each processor. The convention in software is that [bss-end,end-of-stack] defines the available memory region for each processor. Finally the IRQAMP controller can be configured using the -mpirqsel option. Below is an example of a AMP system with 2 processors. One RTEMS image running at 0x0, the other at 0x40000000.

```
$ mkprom2 \
  -freq 50 \
  -mp \
  -mpstart 0x3 \
  -mpirqsel 0 0 \
  -mpirqsel 1 1 \
  -mpuart 2 0xF0000000 0xf0001000 \
  -mpstack 2 0x3fffff00 0x400fff00 \
  -mpentry 2 0x0 0x40000000 \
  rtems-tasks-0x00000000 rtems-tasks-0x40000000 -o amp.prom
```

1.9. DDR/DDR2 controller options

Table 1.8. MKPROM2 options for DDR/DDR2 controller

Option	Description
-ddrram size	Set memory bank size in MByte. Supported values are: 8-1024. Default: 64
-ddrbanks count	Set number of banks. Default: 1
-ddrfreq freq	Set DDR frequency in MHz. Default: 90.
-ddrrefresh num	Set the DDR refresh period in us. Default is 7.8 us.
-ddrcol size	Set columns size. Supported values are: 512, 1024, 2048, 4096. Default: 1024
-ddr2spa_cfg1 hex	Alternatively specify cfg1 of the DDR2 controller as hex.

Option	Description
-ddr2spa_cfg3 hex	Alternatively specify cfg3 of the DDR2 controller as hex.
-ddr2spa_cfg4 hex	Optionally specify cfg4 of the DDR2 controller as hex.
-ddrspa_cfg1 hex	Alternatively specify cfg1 of the DDR controller as hex.

1.10. SDCTRL64/FTSDCTRL64 controller options

Table 1.9. MKPROM2 options for SDCTRL64/FTSDCTRL64 controller

Option	Description
-ftsctrl64_cfg1 [val]	Specify the cfg1 register of the SDCTRL64/FTSDCTRL64 controller (SDRAM Configuration register) .
-ftsctrl64_cfg2 [val]	Specify the cfg2 register of the SDCTRL64/FTSDCTRL64 controller (SDRAM Power-Saving configuration register).

1.11. FTAHBRAM controller options

Table 1.10. MKPROM2 options for FTAHBRAM controller

Option	Description
-ftahbram_edac	If specified the first FTAHBRAM controller's EDAC is enabled. If not specified the first FTAHBRAM controller's configuration register will be written a zero, disabling on-chip memory EDAC. Note that memory is not washed, that can either be done manually from a <code>bdinit</code> function or using on of the optional <code>-edac-clean</code> regions.

1.12. SDCTRL controller options

Table 1.11. MKPROM2 options for SDCTRL controller

Option	Description
-sdcfg1 [val]	Specify the cfg1 register of the SDCTRL controller (SDRAM Configuration register) .

1.13. SPI memory controller options

Table 1.12. MKPROM2 options for SPI memory controller

Option	Description
-spimeas	Enables the SPI memory controller alternate scaler early in the boot process.

1.14. Custom controllers

If the target LEON3 system contains a custom controller, the initialization of the controller must be made through the `bdinit1` function. Below is an example of a suitable `bdinit.c` file. The file should be compiled with `.sparc-elf-gcc -O2 -c -msoft-float.`, and `mkprom2` should be run with the `-bdinit` option.

```
void bdinit1() {
<.. your init code here ..>
}

void bdinit2 () {}
```

1.15. Timer initialization

This section describes the default initialization of GPTIMER and GRTIMER cores on LEON3 systems with AM-BA plug and play.

- Only timer cores (GPTIMER/GRTIMER) on the first APB bus are initialized by `mkprom2`.
- GPTIMER cores are initialized before GRTIMER cores.
- The timer core prescaler reload value is set such that it underflows once every microsecond. The `-freq` parameter is used to calculate the prescaler value.
- The first subtimer of each timer core is configured with reload value `0xffffffff`. Its control register is then initialized such that the subtimer is loaded, enabled and set in restart mode.
- The last subtimer of the first timer core (watchdog) is configured with reload value `300000000` (5 minutes). Its control register is not initialized (reset value remains).
- All other subtimers are initialized with 0 in their counter value registers, reload value registers and control registers.

This default timer initialization can be overridden by `bdinit1()` as described in this document.

2. Examples

This chapter contains examples on how MKPROM2 can be used.

2.1. Quick start

This is a quick start tutorial on how to come up and running with MKPROM2 on a typical LEON system. UT700 LEAP board is used for the purpose of this example.

2.1.1. Extracting mkprom2 parameters from GRMON

This example shows how GRMON can be used to extract the parameters to use with the **mkprom2** frontend.

```
$ grmon -uart /dev/ttyUSB0

GRMON2 LEON debug monitor v2.0.87 pro version

[...]

using port /dev/ttyUSB0 @ 115200 baud
Device ID:          0x699
GRLIB build version: 4110
Detected system:   UT699E/UT700
Detected frequency: 67 MHz

[...]

grmon2> info mkprom2
Mkprom2 switches:
-leon3 -freq 67 -nosram -sdram 32 -trfc 75 -trp 30 -baud 38417
grmon2>
```

The switches presented by the **info mkprom2** command corresponds to system properties that GRMON has probed as part of its initialization. This includes, for example:

- System clock frequency
- Memory controller configuration
- Properties of installed memory
- UART baud, as specified by the GRMON command line or the default.

These extracted parameters can be used directly with the **mkprom2** command line tool:

```
$ sparc-rtems-gcc rtems-hello.c -O2 -o rtems-hello.elf
$ mkprom2 -leon3 -freq 67 -nosram -sdram 32 -trfc 75 -trp 30 -baud 38417 rtems-hello.elf -o rtems-hello.rom
MKPROM v2.0.65 - boot image generator for LEON applications
Copyright Cobham Gaisler AB 2004-2017, all rights reserved.

Creating LEON3 boot prom: rtems-hello.rom
Success!
```

Note that GRMON has limited knowledge of the installed external memory: density is often correct but timing parameters estimated. This means that the memory controller parameters must be verified against the relevant data sheet by the user.

2.1.2. Loading boot image to ROM

Loading the boot image to the ROM is board dependent. For the UT700 LEAP board, it can be loaded to the on-board MRAM using the GRMON2 **load** command:

```
grmon2> load -wprot rtems-hello.rom
00000000 .text          88.8kB / 88.8kB  [=====] 100%
Total size: 88.8kB (95.23kbit/s)
Entry point 0x0
Image rtems-hello.rom loaded
```

The option **-wprot** temporarily disables the memory controller write protection during the load operation.

As a sanity check, the GRMON 2 command **verify** can be used to verify that the image was loaded correctly.

```
grmon2> verify rtems-hello.rom
00000000 .text                88.8kB / 88.8kB  [=====] 100%
Total size: 88.83kB (51.03kbit/s)
Entry point 0x0
Image of rtems-hello.rom verified without errors

grmon2> dis 0 4
0x00000000: 88100000  clr  %g4
0x00000004: 09000000  sethi %hi(0x0), %g4
0x00000008: 81c120b0  jmp  %g4 + 0xB0
0x0000000c: 01000000  nop
```

The first couple of instructions is also disassembled in the example above.

2.1.3. Application start on system reset

On system reset, the following message is displayed on the first UART (APBUART0):

```
MKPPROM2 boot loader v2.0.65
Copyright Cobham Gaisler AB - all rights reserved

system clock   : 67.0 MHz
baud rate      : 38417 baud
prom           : 512 K, (2/2) ws (r/w)
sdram          : 32768 M, 1 bank(s), 9-bit column
sdram          : cas: 2, trp: 45 ns, trfc: 90 ns, refresh 7.8 us

decompressing .text to 0x40000000
decompressing .data to 0x40023530
decompressing .jcr to 0x400246f0

starting rtems-hello.elf

[...]
```

2.2. Customizing with `bdinit.o`

MKPPROM2 boot code initializes the most essential and common peripherals. Custom initializations can be defined by the user with the `mkpprom2` option `-bdinit`.

The reason behind the `-bdinit` concept is to limit the complexity of the boot loader standard code. Another reason is that custom initialization is highly board and application specific so it would be hard to provide a slim and useful interface for all possible kinds of peripheral initializations. The primary area for MKPPROM2 built-in initialization is CPU internal registers, memory controller registers, APBUART and optionally memory initialization when EDAC is used.

Usage of `-bdinit` is described in Table 1.2. A file named `bdinit.o` is picked up from the current directory and is assumed to contain entry points for the functions:

- `bdinit0()`
- `bdinit1()`
- `bdinit2()`

The `bdinit.o` file can be compiled for example with:

```
sparc-rtems-gcc -Os -Wall -Wextra -pedantic -c bdinit.c -o bdinit.o
```

Any LEON compatible compiler can be used to compile the `bdinit.o`. It is important to provide the `-c` option to only do the compile step and not linking. It is also recommended to use optimization level `-Os`.

Care must be taken when implementing `bdinit0()` and `bdinit1()` since they are called by the boot code before memory is available in the run-time. It is safe to keep them as empty void-functions:

```
/*
 * bdinit0 is called before peripheral has been initialized and before RAM is
 * available.
 */
```

```
void bdinit0(void) { }

/*
 * bdinit1 is called after peripherals registers have been initialized, but
 * before RAM is available.
 */
void bdinit1(void) { }

/* bdinit2 is called after MKPROM boot loader has initialized memory. */
void bdinit2(void)
{
  /* Do some custom initialization, for example clock gating. */
}
```

3. Support

For support contact the Cobham Gaisler support team at support@gaisler.com.

When contacting support, please identify yourself in full, including company affiliation and site name and address. Please identify exactly what product that is used, specifying if it is an IP core (with full name of the library distribution archive file), component, software version, compiler version, operating system version, debug tool version, simulator tool version, board version, etc.

The support service is only for paying customers with a support contract.

Cobham Gaisler AB
Kungsgatan 12
411 19 Gothenburg
Sweden
www.cobhamaes.com/gaisler
sales@gaisler.com
T: +46 31 7758650
F: +46 31 421407

Cobham Gaisler AB, reserves the right to make changes to any products and services described herein at any time without notice. Consult Cobham or an authorized sales representative to verify that the information in this document is current before using this product. Cobham does not assume any responsibility or liability arising out of the application or use of any product or service described herein, except as expressly agreed to in writing by Cobham; nor does the purchase, lease, or use of a product or service from Cobham convey a license under any patent rights, copyrights, trademark rights, or any other of the intellectual rights of Cobham or of third parties. All information is provided as is. There is no warranty that it is correct or suitable for any purpose, neither implicit nor explicit.

Copyright © 2021 Cobham Gaisler AB