# LEON Linux

## LEON Linux User's Manual

USER MANUAL

RELEASED JUNE 2025

# Table of Contents

1. Introduction ..... 4
1.1. Conventions in this document ..... 4
2. The LEON Linux 6.13 Beta release ..... 5
3. Configuring the Linux Kernel ..... 6
4. Pin multiplexing ..... 7
4.1. Mklinuximg configuration ..... 7
4.2. GR740 ..... 7
5. Subsystems and drivers ..... 8
5.1. CAN subsystem ..... 8
5.1.1. Kernel configuration for CAN ..... 9
5.1.2. GRCAN and GRHCAN driver ..... 9
5.1.3. CAN_OC driver ..... 9
5.1.4. CAN kernel space interface ..... 10
5.1.5. CAN user space interface ..... 10
5.1.6. CAN in Buildroot ..... 10
5.2. Clock subsystem ..... 11
5.2.1. Kernel configuration ..... 11
5.2.2. Debugfs support ..... 11
5.2.3. Frontgrade Gaisler Clock Gating Unit (GRCG) Driver ..... 11
5.3. Debugging ..... 14
5.3.1. ftrace ..... 14
5.3.2. Lock proving ..... 15
5.4. Ethernet subsystem ..... 15
5.4.1. GRETH 10/100 and GBIT driver ..... 15
5.5. Framebuffer subsystem ..... 16
5.5.1. Kernel configuration for framebuffer ..... 16
5.5.2. SVGACTRL driver ..... 16
5.6. GPIO subsystem ..... 16
5.6.1. Kernel configuration for GPIO ..... 16
5.6.2. GRGPIO driver ..... 16
5.6.3. GPIO kernel space interface ..... 17
5.6.4. GPIO user space interface ..... 17
5.6.5. GPIO in Buildroot ..... 17
5.7. $I^2C$ subsystem ..... 18
5.7.1. Kernel configuration for $I^2C$ ..... 18
5.7.2. I2CMST driver ..... 18
5.7.3. $I^2C$ user space interface ..... 19
5.7.4. $I^2C$ in Buildroot ..... 19
5.8. Memory Management ..... 19
5.8.1. Memory Models ..... 19
5.8.2. Defer memory initialisation ..... 19
5.8.3. Wide address space support ..... 19
5.9. PCI subsystem ..... 20
5.9.1. Kernel configuration for PCI ..... 20
5.9.2. GRPCI driver ..... 21
5.9.3. GRPCI2 driver ..... 21
5.10. Perf subsystem ..... 21
5.10.1. Kernel configuration for `perf` ..... 21
5.10.2. `Perf` in Buildroot ..... 21
5.10.3. L4STAT driver ..... 21
5.11. PS/2 subsystem ..... 23
5.11.1. Kernel configuration for PS/2 ..... 23
5.11.2. APBPS2 driver ..... 23
5.12. SPI subsystem ..... 23
5.12.1. Kernel configuration for SPI ..... 23
5.12.2. SPICTRL driver ..... 23
5.12.3. Important changes between Linux 4.9 and Linux 6.13 ..... 25

5.12.4. SPI user space interface ..... 25
5.12.5. SPI in Buildroot ..... 26
5.13. Timers subsystem ..... 26
5.13.1. Clock sources ..... 26
5.13.2. Clock event devices ..... 26
5.13.3. High Resolution Timers ..... 26
5.13.4. LEON Linux default system timer ..... 26
5.13.5. Clocksource using the up-counter in LEON ..... 26
5.13.6. Frontgrade Gaisler GPTIMER timer driver ..... 27
5.13.7. User space interface ..... 28
5.14. TTY subsystem ..... 29
5.14.1. Kernel configuration for TTY ..... 29
5.14.2. APBUART driver ..... 29
5.14.3. Mklinuximg configuration ..... 29
5.15. USB device subsystem ..... 29
5.15.1. Kernel configuration for USB device ..... 29
5.15.2. GRUSBDC driver ..... 30
5.16. USB host subsystem ..... 30
5.16.1. Kernel configuration for USB host ..... 30
5.16.2. GRUSBHC EHCI driver ..... 30
5.16.3. GRUSBHC UHCI driver ..... 31
5.17. Watchdog subsystem ..... 31
5.17.1. Kernel configuration for Watchdog ..... 31
5.17.2. GPTIMER Watchdog Driver ..... 32
5.17.3. Early Watchdog Support ..... 32
5.17.4. GRWATCHDOG Driver ..... 33
5.17.5. Watchdogs with short maximum hw-timeout ..... 34
5.17.6. Watchdog in Buildroot ..... 34
6. Reboot ..... 36
7. Real-time Linux (PREEMPT_RT) ..... 37
7.1. Latencies ..... 37
7.2. Switching to a Fully Preemptible Kernel: Key Considerations ..... 37
7.3. Kernel configuration ..... 38
7.3.1. Mandatory options ..... 38
7.3.2. Optional options ..... 38
7.3.3. Debugging options ..... 38
7.4. MKLINUXIMG configuration ..... 39
7.5. Real-time Linux in Buildroot ..... 39
7.5.1. cyclictest ..... 39
7.6. Tuning ..... 39
8. Frequently Asked Questions ..... 41
8.1. What causes DMA-API warnings on addresses 0xf0005000-0xf0007fff? ..... 41
8.2. Initialisation of unavailable ranges when using Sparse memory ..... 41
9. Support ..... 42

# 1. Introduction

This document contains information on the LEON Linux releases and documentation on configuring the Linux kernel, and configuring and using device drivers for GRLIB IP cores.

Additional information can be found in the LEON Linux overview document, in the documentation for the LINUXBUILD linux build environment, the mklinuximg RAM loader tool and the external driver package documentation.

## 1.1. Conventions in this document

When describing configuration options, both the descriptive name of the option and the variable name of the option is mentioned, the first in quotation marks and the latter within parenthesis. E.g., when configuring the kernel for LEON, the option described as “Sparc Leon processor family” should be chosen. This configuration option has the varible name “SPARC_LEON”. This is mentioned as “Sparc Leon processor family” (SPARC_LEON). Searches in the configurators (xconfig, gconfig, menuconfig) are matching against the variable names.

Descriptions on where to find an configuration option to be chosen are often in relation to earlier selections. E.g., in the description on how to select a driver for a specific core, the description on where to find it is often described in relation to where the subsystem was configured.

Chapter 5 contains descriptions for setting up extra nodes and/or properties in the devicetree for several cores using **mklinuximg**. These xml files are used with the `-xml` option of **mklinuximg**. See the Mklinuximg manual for details.

# 2. The LEON Linux 6.13 Beta release

The 6.13 release brings a lot of new features compared to 5.10 which is the current stable LEON Linux release such as:

- Real-time Linux (PREEMPT_RT)
- Ftrace support (including dynamic ftrace and stacktrace support)
- Lock proving / debugging support
- High-resolution timer support
- CPU up-counter based clocksource

---

The LEON Linux 6.13 beta release is based on a kernel version without LTS support which means that no new offical releases from kernel.org is to be expected. It can be assumed that future releases of LEON Linux will move to a suitable version with LTS support.

---

# 3. Configuring the Linux Kernel

This chapters gives the bare minimum for configuring the Linux kernel for LEON. The configurations from the leon-linux kernel packages contains suitable default configurations to use as a starting point. See Chapter 5 for configuration of device drivers for GRLIB IP cores.

When configuring the kernel make sure to specify ARCH=sparc when starting the configurator (e.g. **make ARCH=sparc xconfig**).

Make sure that

- “64-bit kernel” (64BIT)

is not selected at top level. Then select

- “Sparc Leon processor family” (SPARC_LEON)

under “Processor type and features”.

To build an SMP kernel, also select

- “Symmetric multi-processing support” (SMP)

under “Processor type and features”.

# 4. Pin multiplexing

This section describes how pin multiplexing affects the availability of cores in the system.

Mklinuximg have support for static pin multiplexing for some components, listed in their own subsections. For components not supported by Mklinuximg, the removal of cores can be achieved by using the “del-core” tag in a xml.

## 4.1. Mklinuximg configuration

The static pin multiplexing function is enabled by using the `-pinmux` flag.

## 4.2. GR740

Mklinuximg since 2.0.17 can filter devices and skip adding them to the device tree if the device can not be routed to a pin due to the pin multiplexing setting detected on boot. The pin multiplexing is based on the state of the bootstrap signals and is enabled if the **-pinmux** option is provided to Mklinuximg.

Mklinuximg does not support to re-configure alternative functions to function as general-purpose I/Os.

The bootstrap signals involved in pin multiplexing are described in the GR740 Datasheet & User manual, “Pin multiplexing”.

# 5. Subsystems and drivers

This Chapter contains documentation about drivers included in the Linux mainline kernel. The chapter is organized per kernel subsystem and contains information on configuring the subsystem, configuring and drivers in that subsystem, information on kernel and user space interfaces and notes on using them in the Buildroot environment.

See separate document for drivers provided in the GRLIB Driver Package available from https://gaisler.com.

*Table 5.1. Overview of Cores and Drivers*

| Core | Section | Linux Source Path | Comments |
|---|---|---|---|
| APBPS2 | Section 5.11.2 | `drivers/input/serio/apbps2.c` | In mainline since v3.10 |
| APBUART | Section 5.14.2 | `drivers/tty/serial/apbuart.c` | In mainline since v2.6.33 |
| CAN_OC | Section 5.1.3 | `drivers/net/can/sja1000/sja1000*` | In mainline since v3.8 |
| GPTIMER | Section 5.17 | `drivers/watchdog/gptimer_wdt.c` | Only in Gaisler LEON Linux release |
| GPTIMER | Section 5.13 | `drivers/clocksource/timer-gptimer.c` | Only in Gaisler LEON Linux release |
| GRCAN | Section 5.1.2 | `drivers/net/can/grcan.c` | In mainline since v3.8 |
| GRCG | Section 5.2.3 | `drivers/clk/gaisler/clk-grcg.c` | Only in Gaisler LEON Linux release |
| GRETH/ GRETH_GBIT | Section 5.4.1 | `drivers/net/ethernet/aeroflex/greth.c` | In mainline since v2.6.34 |
| GRGPIO | Section 5.6.2 | `drivers/gpio/gpio-grgpio.c` | In mainline since v3.10 |
| GRHCAN | Section 5.1.2 | `drivers/net/can/grcan.c` | In mainline since v3.8 |
| GRPCI | Section 5.9.2 | `arch/sparc/kernel/leon_pci_grpci1.c` | In mainline since v3.10 |
| GRPCI2 | Section 5.9.3 | `arch/sparc/kernel/leon_pci_grpci2.c` | In mainline since v3.0 |
| GRSPW2 | Separate document | `drivers/grlib/spw/grspw.c` | In GRLIB Driver Package |
| GRSPW2_ ROUTER | Separate document | `drivers/grlib/spw/grspw_router.c` | In GRLIB Driver Package |
| GRUSBDC | Section 5.15.2 | `drivers/usb/gadget/udc/gr_udc.c` | In mainline since v3.14 |
| GRUSBHC | Section 5.16.2 & Section 5.16.3 | `drivers/usb/host/{e,u}hci-grlib.c` | In mainline since v3.0 |
| GRWATCH-DOG | Section 5.17 | `drivers/watchdog/grwatchdog_wdt.c` | Only in Gaisler LEON Linux release |
| I2CMST | Section 5.7.2 | `drivers/i2c/busses/i2c-ocores.c` | In mainline since v3.8 |
| L4STAT | Section 5.10.3 | `drivers/perf/l4stat_pmu.c` | Only in Gaisler LEON Linux release |
| SVGACTRL | Section 5.5.2 | `drivers/video/fbdev/grvga.c` | In mainline since v3.2 |
| SPICTRL | Section 5.12.2 | `drivers/spi/spi-fsl-spi.c` | In mainline since v3.10 |

## 5.1. CAN subsystem

General documentation about CAN in Linux can be found in `Documentation/networking/can.txt` in the Linux kernel source tree.

### 5.1.1. Kernel configuration for CAN

Enable CAN support in the Linux kernel by selecting:

- “CAN bus subsystem support” (CAN)

under “Networking support”

Under this section select the following options:

- “Raw CAN Protocol” (CAN_RAW)
- “Platform CAN drivers with Netlink support” (CAN_DEV)
- “CAN bit-timing calculation” (CAN_CALC_BITTIMING)

### 5.1.2. GRCAN and GRHCAN driver

The GRCAN and GRHCAN driver supports listen-only mode, triple samling mode (when available in the core), and one-shot mode. The driver follows the standard Linux CAN interface.

#### 5.1.2.1. Kernel configuration

To use GRCAN and/or GRHCAN, select:

- “Aeroflex Gaisler GRCAN and GRHCAN CAN devices” (CAN_GRCAN)

under “CAN Device drivers”.

The default values of the enable0, enable1 and select bits of the configuration register and the tx and rx buffer sizes can be configured via kernel paramenters listed in Table 5.2.

*Table 5.2. Kernel Parameters*

| Parameter | Default | Description |
|---|---|---|
| grcan.enable0 | 0 | Default configuration of physical interface 0. Configures the “Enable 0” configuration register bit |
| grcan.enable1 | 0 | Default configuration of physical interface 1. Configures the “Enable 1” configuration register bit |
| grcan.select | 0 | Default configuration of physical interface selection. Configures the “Select” bit of the configuration register. |
| grcan.txsize | 1024 | Configures the size of the tx buffer in bytes. |
| grcan.rxsize | 1024 | Configures the size of the rx buffer in bytes. |

For example, adding `grcan.enable0=1` to the kernel command line will set the default for enable0 to 1, and keep 0 as the the default for enable1. The settings for enable0, enable1 and select settings can be overriden at runtime, whereas txsize and rxsize can not. The kernel parameters values can be read at runtime from `/sys/module/grcan/parameters/`. See also `Documentation/kernel-parameters.txt` in the Linux kernel source tree.

##### 5.1.2.1.1. Runtime Configuration

The enable0, enable1 and select bits of the configuration register can be set at runtime via the sysfs file system under `/sys/class/net/canX/grcan/`, where `canX` is the interface name of the device in Linux. For example, **echo 1 > /sys/class/net/can0/grcan/enable0** will set the enable0 bit of interface can0 to 1. See also `Documentation/ABI/testing/sysfs-class-net-grcan` in the Linux kernel source tree.

### 5.1.3. CAN_OC driver

CAN_OC cores use an SJA1000 driver in Linux. Only PeliCAN mode is supported. The driver supports listen-only mode, triple samling mode, one-shot mode and bus-error reporting. The driver follows the standard Linux CAN interface.

#### 5.1.3.1. Kernel configuration

To use CAN_OC, select:

- “Philips/NXP SJA1000 devices” (CAN_SJA1000)
- “Generic Platform Bus based SJA1000 driver” (CAN_SJA1000_PLATFORM)

#### 5.1.3.2. Open Firmware device tree details for CAN_OC

One core can contain several CAN_OC instantiations. The “version” property plus one in the AMBA Plug&Play indicates the number of CAN_OC instantiations. However, mklinuximg creates separate virtual devices for the Open Firmware device tree. So from the Linux kernel point of view every instantiation is a separate device. The nodes are called “GAISLER_CANAHB” in the device tree.

### 5.1.4. CAN kernel space interface

Not applicable

### 5.1.5. CAN user space interface

CAN devices is accessed in Linux as network interfaces, via sockets. The interfaces are named can0, can1, etc. The available CAN interfaces can be seen by running **/sbin/ip link**. Bitrate or specific bittiming parameters are set up from within Linux using the **/sbin/ip** command. Run **/sbin/ip link set canX type can help** (with X substituted by appropriate number) to see the configuration parameters available. Some listed parameters might not be available for particular drivers/hardware. Bringing a CAN interface canX up and down can be done by **ifconfig canX up** and **ifconfig canX up** respectively. Bittiming parameters needs to be set before bringing a CAN interface up the first time.

See `Documentation/networking/can.txt` in the Linux kernel source tree for more details on configuration and the CAN specific configuration parameters of **/sbin/ip**. Note: as of Linux 3.8, hardware filtering is not supported in Linux.

#### 5.1.5.1. The can-utils toolset

The can-utils toolset can be used as quick testing and diagnostics tools and the source code also serves as examples on socket programming for CAN interfaces.

The following example sets up interface can0 at 125 kbps, enables the interface, starts **candump** with parameters set to receive any can frames including error frames and sends a frame using on the CAN bus that is received by **candump**:

```
# /sbin/ip link set can0 type can bitrate 125000
# /sbin/ifconfig can0 up
# candump -e any,0:0,#FFFFFFFF &
# cansend can0 123#abcdef
  can0  123  [3] AB CD EF
```

Note that if there are no other active devices on the bus, the transmit fails even if a local candump is listening to the socket. In the case above, where one-shot mode has not been specified, the controller would retry until it ends up in an error-passive state and error frame(s) would instead be delivered to candump.

### 5.1.6. CAN in Buildroot

The built-in **ip** command of BusyBox can not currently handle all CAN configuration options. The iproute2 package provides a working **/sbin/ip**.

To make sure that the iproute2 package is selectable, under “Target packages” select:

- “Show packages that are also provided by busybox” (BR2_PACKAGE_BUSYBOX_SHOW_OTHERS)

Then under “Network applications” select:

- “iproute2” (BR2_PACKAGE_IPROUTE2)

The “can-utils” package contains the can-utils toolset (see Section 5.1.5.1). Under “Network applications” select:

- “can-utils” (BR2_PACKAGE_CAN_UTILS)

## 5.2. Clock subsystem

The common clock framework is an interface to control the clock nodes.

The common clock framework has two classes, called clock producers and clock consumers. The clock producer is responsible for maintaining a set of clocks. The features of a clock producer varies but for a producer that handles gates it is responsible for enabling/disabling/status of these gates. The clock consumer on the other hand (typically a device driver) utilize functions provided by the clock producer. The relationship between a clock consumer and a clock producer is set up through device tree properties.

General documentation about the common clock framework in Linux can be found in `Documentation/driver-api/clk.rst` in the Linux kernel source tree.

### 5.2.1. Kernel configuration

Enable Common Clock Framework support in the Linux kernel by selecting:

- "Common Clock Framework" (COMMON_CLK)

under "Device Drivers"

*Table 5.3. Kernel cmdline parameters*

| Parameter | Description |
|---|---|
| clk_ignore_unused | On boot, the "Common Clock Framework" automatically disables unused/unreferenced clocks. When this argument is specified, then the unused clocks are ignored and left in whatever state it had on boot. |

### 5.2.2. Debugfs support

Information on clocks are available in debugfs:

```
# mount -t debugfs none /sys/kernel/debug
# cat /sys/kernel/debug/clk/clk_summary
enable  prepare  protect                                duty
clock                          count    count    count        rate   accuracy phase  cycle
---------------------------------------------------------------------------------------------
clkname0                           0        0        0           0          0     0  50000
clkname1                           1        1        0           0          0     0  50000
clkname2                           1        1        0           0          0     0  50000
clkname ...
clknameN                           0        0        0           0          0     0  50000
```

The example above shows the format of the clock summary file with made up clock names, the clock names are set by the clock producers when the clocks are registered.

### 5.2.3. Frontgrade Gaisler Clock Gating Unit (GRCG) Driver

The clock gating driver follows the standard Linux common clock framework interface, including setting up clock consumers from other devices in the device tree. Device drivers (clock consumers) provided in Gaisler LEON Linux releases since 5.10-1.7 and the LEON Linux driver package since 1.2.1 have clock gating support.

When configuring the kernel, select:

- "Frontgrade Gaisler Clock Gating support" (CLK_GRCG)

under "Common Clock Framework".

#### 5.2.3.1. Protected clocks

During the late boot stage, Linux will disable unused clocks unless the kernel argument "clk_ignore_unused" has been specified, then all unused clocks are ignored and left in whatever state they had on boot. This might be useful in some situations when the clock state should remain after boot or during development. But it could be that just one or a few clocks should be ignored. For example when using a driver that does not implement clock handling but depend on a managed clock or if the system is part of an AMP configuration. The "Frontgrade Gaisler Clock Gating" driver can be configured to protect individual clocks that should not be handled by the driver by specifying

the clocks in a device tree property called “protected-clocks”. The property is described in Section 5.2.3.2 (under “Optional device tree properties”) and there is an example of usage in Section 5.2.3.4 (see Example 3).

#### 5.2.3.2. Mklinuximg configuration

The “Frontgrade Gaisler Clock Gating” driver needs device tree additions.

Mklinuximg from version 2.0.17 have support for generating them automatically for some components, listed in their own subsections. The `-clockgate` option enables the clock gate function and it is expected that the “Frontgrade Gaisler Clock Gating” driver is enabled in Linux when the option is set. The device tree additions can also be configured by a custom xml (see Section 5.2.3.4).

For the clock producer (clock gating core) a number of required properties are expected but the driver also support a few optional properties:

Required device tree properties:

- “compatible” should be set to “gaisler,grcg”.
- “clock-cells” should be set to 1.
- “clock-output-names” property should be set to an array of strings that describes the clocks that the driver handles where each index in the array corresponds to the bit index of the gate it represent. The max number of items is 32.

Optional device tree properties:

- “clock-indices”, if specified, should be set to an array of integers (bit index) of the gate it represent. The array should have the same size as specified for the “clock-output-names” property. (see example 2 in Section 5.2.3.4).
- “protected-clocks”, if specified, should be set to an array of integers (bit index) of the gates it represent. The clocks specified will not be registered and not handled by the driver. (see example 3 in Section 5.2.3.4).

The clock consumers (i.e peripherals connected to the clock gating core) should specify the following properties:

- “clocks” specifies a reference (phandle) to the device tree node of the clock producer and the index of the clock it depends on.
- “clock-names” is an array with string id's of the clocks it reference. It should be set to “gate” as all peripheral drivers that supports clock gating will look for it when they check if the device is connected to a clock gating unit.

More information about the device tree properties can be found here:

- https://github.com/devicetree-org/dt-schema/blob/main/dtschema/schemas/clock/clock.yaml

The peripheral drivers (that supports clock handling) are designed to optionally support clock handling. The clock handling in the drivers will be enabled when:

- All the clock related device tree properties are available in the device node.
- The “Common Clock Framework” is enabled.

If the “Frontgrade Gaisler Clock Gating” driver is not enabled, the peripheral drivers will fail to obtain a valid clock and not be able to load.

The clock handling in the peripheral drivers is ignored if either:

- The clock related device tree properties are missing in the device node.
- The “Common Clock Framework” is disabled.

For a system that do not use the clock gate function but requires the “Common Clock Framework” (i.e for another driver), then the `-clockgate` option should not be set to avoid issues with loading drivers.

#### 5.2.3.3. Mklinuximg configuration for GR740

The device tree additions for the clock gating unit and all connected peripherals are automatically handled for GR740 by Mklinuximg since 2.0.17 (when the `-clockgate` option is specified).

Please refer to the GR740 User manual, “25. Clock gating unit” in version: 2.6, for detailed information about the peripherals that are connected to the clock gating unit.

#### 5.2.3.4. Mklinuximg configuration for a custom system

For components without built-in support in Mklinuximg it is possible to do a custom configuration using xml additions. Example 1-3 describes how to set up the clock producer (GAISLER_CLKGATE). Example 4 describes how to set up clock consumers. A complete xml should include a combination of both clock producer(s) and clock consumers as the producer needs to declare a “corelabel” and the consumers need to reference them using a “corehandle” to get a valid link through the phandle.

Example 1. The following example set up the “Frontgrade Gaisler Clock Gating” driver (clock provider) to handle three clocks which corresponds to index 0-2 of the clock enable register of the clock gating unit.

```
<?xml version="1.0"?>
<matches>
  <match-core vendor="VENDOR_GAISLER" device="GAISLER_CLKGATE" index="0">
    <corelabel name="clkc0"/>
    <add-prop name="compatible">
      <string>gaisler,grcg</string>
    </add-prop>
    <add-prop name="#clock-cells">
      <int>1</int>
    </add-prop>
    <add-prop name="clock-output-names">
      <string>greth0</string>    <!-- bit 0 of the clock enable register -->
      <string>greth1</string>    <!-- bit 1 of the clock enable register -->
      <string>spwrouter</string> <!-- bit 2 of the clock enable register -->
    </add-prop>
  </match-core>
</matches>
```

Example 2. The following example set up the “Frontgrade Gaisler Clock Gating” driver by explicitly specifying clock-indices.

```
<?xml version="1.0"?>
<matches>
  <match-core vendor="VENDOR_GAISLER" device="GAISLER_CLKGATE" index="0">
    <corelabel name="clkc0"/>
    <add-prop name="compatible">
      <string>gaisler,grcg</string>
    </add-prop>
    <add-prop name="#clock-cells">
      <int>1</int>
    </add-prop>
    <add-prop name="clock-output-names">
      <string>greth0</string>
      <string>greth1</string>
      <string>spwrouter</string>
    </add-prop>
    <!-- Explicitly specify the indexes using clock-indices -->
    <!-- Useful for indexes which is not linear from zero -->
    <add-prop name="clock-indices">
      <int>0</int>  <!-- bit 0 of the clock enable register -->
      <int>10</int> <!-- bit 10 of the clock enable register -->
      <int>20</int> <!-- bit 20 of the clock enable register -->
    </add-prop>
  </match-core>
</matches>
```

Example 3. The following example set up the “Frontgrade Gaisler Clock Gating” driver but set one clock as protected. This clock will be skipped and will not be handled by the common clock framework.

```
<?xml version="1.0"?>
<matches>
  <match-core vendor="VENDOR_GAISLER" device="GAISLER_CLKGATE" index="0">
    <corelabel name="clkc0"/>
    <add-prop name="compatible">
      <string>gaisler,grcg</string>
    </add-prop>
    <add-prop name="#clock-cells">
      <int>1</int>
    </add-prop>
    <add-prop name="clock-output-names">
      <string>greth0</string>
      <string>greth1</string>
      <string>spwrouter</string>
    </add-prop>
```

```
    <!-- Skip a clock by adding the index to the protected-clocks property -->
    <add-prop name="protected-clocks">
      <int>1</int>  <!-- bit 1 (greth1) will be skipped -->
    </add-prop>
  </match-core>
</matches>
```

Example 4: The following example set up a GAISLER_ETHMAC core as a clock consumer and then two GAISLER_GRCAN cores (0 and 1) are set up as clock consumers of the same gate (1). The CAN driver instances (in Linux) will keep a reference each to the clock which will ensure that the clock is remained enabled if an instance is unloaded.

```
<?xml version="1.0"?>
<matches>
  <!-- Clock producer -->
  <match-core vendor="VENDOR_GAISLER" device="GAISLER_CLKGATE" index="0">
    <corelabel name="clkc0"/>
    <add-prop name="compatible">
      <string>gaisler,grcg</string>
    </add-prop>
    <add-prop name="#clock-cells">
      <int>1</int>
    </add-prop>
    <add-prop name="clock-output-names">
      <string>greth0</string>  <!-- bit 0 of the clock enable register -->
      <string>can</string>     <!-- bit 1 of the clock enable register -->
    </add-prop>
  </match-core>
  <!-- Clock consumers -->
  <match-core vendor="VENDOR_GAISLER" device="GAISLER_ETHMAC" index="0">
    <add-prop name="clocks">
      <corehandle ref="clkc0"/> <!-- Corehandle ref to the GRCG core. -->
      <int>0</int> <!-- Index of the clock in (clock-output-names) -->
    </add-prop>
    <add-prop name="clock-names">
      <string>gate</string> <!-- The string id of the clock -->
    </add-prop>
  </match-core>
  <match-core vendor="VENDOR_GAISLER" device="GAISLER_GRCAN" index="0">
    <add-prop name="clocks">
      <corehandle ref="clkc0"/> <!-- Corehandle ref to the GRCG core. -->
      <int>1</int> <!-- Index of the can clock in (clock-output-names) -->
    </add-prop>
    <add-prop name="clock-names">
      <string>gate</string> <!-- The string id of the clock -->
    </add-prop>
  </match-core>
  <match-core vendor="VENDOR_GAISLER" device="GAISLER_GRCAN" index="1">
    <add-prop name="clocks">
      <corehandle ref="clkc0"/> <!-- Corehandle ref to the GRCG core. -->
      <int>1</int> <!-- Index of the can clock in (clock-output-names) -->
    </add-prop>
    <add-prop name="clock-names">
      <string>gate</string> <!-- The string id of the clock -->
    </add-prop>
  </match-core>
</matches>
```

## 5.3. Debugging

### 5.3.1. ftrace

Ftrace is Linux's built-in kernel tracer for debugging and performance analysis. It traces function calls and key kernel events (scheduler, memory, network) via static tracepoints. Latency issues can be investigated with a variaty of tracers like “function” and “wakeup_rt”. Ftrace is controlled via the “tracefs” filesystem where it’s possible to select tracers and apply filters.

The kernel documentation about ftrace is very detailed and show plenty of examples on how to use different tracers and options can be found in `Documentation/trace/ftrace.rst`

Enable ftrace by selecting:

- “Tracers” (CONFIG_FTRACE)

Under “Kernel hacking”

Enable tracers of interest, for example:

- “Kernel Function Tracer” (FUNCTION_TRACER)
- “Scheduling Latency Tracer” (SCHED_TRACER)
- “Trace max stack” (STACK_TRACER)

Under “Kernel hacking” under “Tracers”

#### 5.3.1.1. Important notes on using Debuggers with Dynamic FTRACE

When FTRACE_DYNAMIC is enabled, it enables the ftrace framework to dynamically modify ftrace instrumentation points at runtime by replacing “nop” instructions with tracing calls and vice versa. This allows for efficient, on-the-fly debugging and performance analysis of specific kernel functions without requiring a kernel rebuild.

But if a debugger has inserted a software breakpoint at this location then ftrace will fail to update the instrumentation point. If that happens, then ftrace will print splat and disable the function tracer and it will not be possible to use it until after reboot.

### 5.3.2. Lock proving

Lock proving is a Linux kernel debugging feature that helps detect potential deadlocks and improper lock usage by validating lock dependencies at runtime. It builds a graph of lock relationships and checks for circular dependencies that could cause system hangs. When issues are found, it logs detailed warnings to assist with debugging. This option is mainly intended for kernel developers and testers, as it introduces performance overhead and is not suitable for production systems.

Enable lock proving by selecting:

- “Lock debugging: prove locking correctness” (PROVE_LOCKING)

Under “Kernel hacking” under “Lock Debugging (spinlocks, mutexes, etc, ...)”

## 5.4. Ethernet subsystem

No attempt is made here to try to cover Ethernet and networking for Linux in general.

### 5.4.1. GRETH 10/100 and GBIT driver

#### 5.4.1.1. Kernel configuration

When configuring the kernel, select:

- “Aeroflex Gaisler GRETH Ethernet MAC support” (GRETH)

under “Ethernet driver support”.

The MAC address of the first GRETH core can be set up by a kernel module parameter (described below) or in the ID prom (desbribed under mklinuximg configuration further down), where the module parameter takes precedence. Th last byte of the MAC address will get increase for each following GRETH core.

*Table 5.4. Kernel Parameters*

| Parameter | Default | Description |
|---|---|---|
| greth.debug | -1 | Bitmask deciding which types of messages the GRETH driver should emit. See the NETIF_MSG_* enums in `include/linux/netdevice.h` in the Linux kernel source tree for the different bits. (It can later be set from userspace using the ethtool utility. It is there called msglvl.) |
| greth.macaddr | 0,0,0,0,0,0 | Default MAC address (on the form 0x08,0x00,0x20,0x30,0x40,0x50) of the first GRETH. If this is all zeroes the ID prom is instead used. See on mklinuximg below for how to set it there. |
| greth.edcl | 1 | Whether EDCL is used. Is used to determine if PHY autonegotiation is to be done right away or not. |

#### 5.4.1.2. Mklinuximg configuration

The default MAC address for the first GRETH core can be set in the ID prom from mklinuximg by using the -ethmac flag. The kernel module parameter of the driver descruibed above takes precedence over this setting.

## 5.5. Framebuffer subsystem

See `Documentation/fb` in the Linux kernel source tree for general documentation about the framebuffer subsystem and its kernel and user space interfaces in Linux.

### 5.5.1. Kernel configuration for framebuffer

Enable framebuffer support in the Linux kernel by selecting:

- "Support for frame buffer devices" (FB)

under "Grahics support" under "Device drivers".

### 5.5.2. SVGACTRL driver

The SVGACTRL driver follows the standard Linux kernel framebuffer interface.

When configuring the kernel, select:

- "Aeroflex Gaisler framebuffer support" (FB_GRVGA)

under "Support for frame buffer devices".

## 5.6. GPIO subsystem

See `Documentation/gpio.txt` in the Linux kernel source tree for general documentation about the GPIO subsystem and its kernel and user space interfaces in Linux. See also `devicetree/bindings/gpio/gpio.txt` for documentation on devicetree representations.

### 5.6.1. Kernel configuration for GPIO

Enable GPIO support in the Linux kernel by selecting:

- "GPIO Support" (GPIOLIB)

under "Device Drivers".

### 5.6.2. GRGPIO driver

The GRGPIO driver follows the standard Linux GPIO interface, including setting up GPIO usage from other devices in the devicetree.

#### 5.6.2.1. Kernel configuration

When configuring the kernel, select:

- "Aeroflex Gaisler GRGPIO support" (GPIO_GRGPIO)

under "GPIO Support".

#### 5.6.2.2. Mklinuximg configuration

The GRGPIO driver needs special devicetree properties. Mklinuximg from version 2.0.6 have support for generating them from explicitely specified information or from autoprobing the core (when possible).

The following options all take a comma-separated list of values with one value for each GRGPIO core in the system (in scan order). The `-gpio-irqgen` option specifies the `irqgen` generic for each core. Autoprobing can not distinguish between 0 and 1. The `-gpio-nbits` option specifies the `nbits` generic for each core — i.e. the number of GPIO lines. The `-gpio-imask` option specifies the `imask` generic for each core — i.e. which GPIO lines have interrupt support. The `-gpio-noprobe` option makes sure that no autoprobing is being done. Otherwise probing is done for information not specified by the above options.

#### 5.6.2.3. Mklinuximg configuration for GPIO users

It is possible for some drivers that uses GPIO to specify in the devicetree which GPIO core and GPIO line that should be used by setting up an array property containing GPIO information for the GPIO lines to use.

Each GPIO line used needs three entries in the array. The first entry is a handle to the GPIO core. This handle to the GPIO core is realized in mklinuximg by using the corelabel and corehandle tags. The second entry is the

offset within the GPIO core (i.e. which GPIO line of the core to use). The third entry is a bitmask. Set bit 0 in the bitmask to 1 if the GPIO line is active-low.

The following example sets up some LED:s connected to the first GRGPIO to be used by the “default on” and “heartbeat” LED trigger drivers.

```
<?xml version="1.0"?>
<matches>
  <match-core vendor="VENDOR_GAISLER" device="GAISLER_GPIO" index="0">
    <corelabel name="gpio0"/> <!-- For user to refer to this core -->

    <!-- Set up user that could be placed elsewhere -->
    <add-node name="exampleleds"> <!-- irrelevant name -->
      <add-prop name="compatible">
        <string>gpio-leds</string> <!-- Matches leds-gpio driver -->
      </add-prop>
      <add-node name="led0">
        <add-prop name="default-state">
          <string>on</string>
        </add-prop>
        <add-prop name="linux,default-trigger">
          <string>default-on</string> <!-- use the default-on trigger driver -->
        </add-prop>
        <add-prop name="gpios">
          <corehandle ref="gpio0"/> <!-- Use "gpio0" -->
          <int>27</int> <!-- Use GPIO line 27 -->
          <int>0</int>
        </add-prop>
      </add-node>
      <add-node name="led0">
        <add-prop name="default-state">
          <string>keep</string>
        </add-prop>
        <add-prop name="linux,default-trigger">
          <string>heartbeat</string>  <!-- use the heartbeat trigger driver -->
        </add-prop>
        <add-prop name="gpios">
          <corehandle ref="gpio0"/> <!-- Use "gpio0" -->
          <int>28</int> <!-- Use GPIO line 28 -->
          <int>0</int>
        </add-prop>
      </add-node>
  </match-core>
</matches>
```

The example sets the “exampleleds” up as a child node of the GRGPIO core, but that is not necessary. Here two different LED trigger drivers uses the leds-gpio driver that in turn uses GRGPIO. The example depends on “LED support”, “LED Class support”, “LED Trigger support”, “LED Support for GPIO connected LEDs”, “LED Heartbeat trigger”, “LED Heartbeat Trigger” and “LED Default ON Trigger” being enabled for the kernel.

For another example of setting up GPIO usage, see Section 5.12.2.1.

### 5.6.3. GPIO kernel space interface

See `Documentation/gpio.txt` in the Linux kernel source for documentation on the general GPIO interface. See `include/linux/of_gpio.h` for details on how to get from the devicetree the gpio numbers used in the general kernel interface.

### 5.6.4. GPIO user space interface

#### 5.6.4.1. The libgpiod toolset

The libgpiod toolset is useful for quick testing and diagnostics of a GPIO device in Linux and the source code also serves as examples on GPIO programming from user space. Run the **gpiodetect**, **gpioinfo**, **gpioget**, **gpioset**, **gpiomon** and **gpionotify** commands without parameters to get rudimentary documentation.

### 5.6.5. GPIO in Buildroot

The libgpiod toolset can be installed by enabling the following packages: Under “Hardware handling” select:

- “libgpiod” (BR2_PACKAGE_LIBGPIOD)
- “install tools” (BR2_PACKAGE_LIBGPIOD_TOOLS)

Man pages for the tools in the toolset can be found under the tools directory of the libgpiod build directory (e.g. `build/libgpiod-1.6.4/man/gpiodetect.man`) and can be used by using man -l.

## 5.7. I2C subsystem

See `Documentation/i2c` in the Linux kernel source tree for general documentation about the I2C subsystem and its kernel and user space interfaces in Linux.

### 5.7.1. Kernel configuration for I2C

Enable I2C support in the Linux kernel by selecting:

- "I2C support" (I2C)

under "Device Drivers"

### 5.7.2. I2CMST driver

I2CMST cores use the "I2C Open Cores" driver in Linux. The driver follows the standard Linux I2C interface, including registering devices through the device tree.

When configuring the kernel, select:

- "OpenCores I2C Controller" (I2C_OCORES)

under "I2C Hardware Bus support".

#### 5.7.2.1. Mklinuximg configuration

The I2CMST driver needs special devicetree properties. Mklinuximg from version 2.0.4 and onward adds these automatically.

To connect a I2C slave device to a I2C master through the device tree, a node for the slave device, containing some properties, should be put as a child of the node for the master device. A "register" property should contain the address of the slave on the I2C bus and a "compatile" property should contain the name in the `struct i2c_device_id` of the slave device driver.

The following example xml file for mklinuximg adds a Dallas DS1672 real time clock as a I2C slave device with chip address 0x68 to the device tree under the first I2CMST core:

```
<?xml version="1.0"?>
<matches>
  <match-core  vendor="VENDOR_GAISLER" device="GAISLER_I2CMST" index="0">
    <add-node name="rtc"> <!-- irrelevant name -->
      <add-prop name="reg">
        <int>0x68</int> <!-- chip address -->
      </add-prop>
      <add-prop name="compatible">
        <string>ds1672</string> <!-- match with driver -->
      </add-prop>
    </add-node>
  </match-core>
</matches>
```

From Linux 4.0 it is recommended to use Mklinuximg version 2.0.18 or newer as the meaning of the "clock-frequency" property changed from describing the input clock frequency of the I2C controller to describe the I2C bus clock frequency. The driver will issue the following warning when using a Mklinuximg version prior to 2.0.18:

```
ocores-i2c ffd1125c: Deprecated usage of the 'clock-frequency' property, please update to 'opencores,ip-clock-frequency'
```

Even though the intention was to keep backwards compatibility in the driver, due to a regression, the driver might fail to load when using Mklinuximg < 2.0.18 and Linux >= 4.0.

Mklinuximg since 2.0.18 sets the clock frequency of the I2C bus clock to 100 KHz. The following xml example show how the I2C bus clock frequency can be overridden:

```
<?xml version="1.0"?>
<matches>
  <match-core  vendor="VENDOR_GAISLER" device="GAISLER_I2CMST" index="0">
    <add-prop name="clock-frequency">
      <int>400000</int> <!-- 400 KHz I2C bus clock frequency -->
    </add-prop>
  </match-core>
```

```
</matches>
```

### 5.7.3. I²C user space interface

#### 5.7.3.1. The i2c-tools toolset

The i2c-tools toolset is useful for quick testing and diagnostics of I²C in Linux and the source code also serves as examples on I²C programming from user space. Run the **i2cdetect**, **i2cget**, **i2cset** and **i2cdump** commands without parameters to get rudimentary documentation.

Warning: These commands can, quoting from the man page, “be extremely dangerous if used improperly”.

### 5.7.4. I²C in Buildroot

The “i2c-tools” package contains the i2c-tools toolset (see Section 5.7.3.1). Under “Hardware handling” select:

- “i2c-tools” (BR2_PACKAGE_I2C_TOOLS)

Man pages for the tools in the toolset can be found under the tools directory of the i2c-tools build directory (e.g. `build/i2c-tools-3.0.3/tools/i2cdetect.8`) and can be used by using man -l.

## 5.8. Memory Management

### 5.8.1. Memory Models

General documentation about memory models in Linux can be found in `Documentation/vm/memory-model.rst` in the Linux kernel source tree.

The following memory models are supported:

- Flat Memory
- Sparse Memory

The Flat memory model is the most efficient system in terms of performance and resource consumption and it is the best option for smaller systems with contiguous, or mostly contiguous, physical memory.

The Sparse memory model is the most versatile memory model available in Linux and is more advanced compared to Flat memory as it supports systems with holes in their physical address spaces and features like deferred initialization of the memory map for larger systems (see Section 5.8.2). Choosing the Sparse memory model is only recommended if the deferred memory initialisation feature is intended to be used.

Choose Memory model by selection one of the available options:

- “Flat Memory” (FLATMEM_MANUAL)
- “Sparse Memory” (SPARSEMEM_MANUAL)

under “Memory Management options” under “Memory model”

### 5.8.2. Defer memory initialisation

Ordinarily all struct pages are initialised during early boot in a single thread. For large systems, this might take considerable amount of time. If this option is set, a small portion of the memory is initialised on early boot and later when the kernel has started some of its core functions the remaining memory gets initialized. On SMP systems, all available CPUs will be used for the deferred initialisation and can lead to improved boot time for systems with large memory (> 256 MiB).

This option depends on “Sparse Memory” (see Section 5.8.1)

Enable deferred memory initialisation support in the Linux kernel by selecting:

- “Defer initialisation of struct pages to kthreads” (DEFERRED_STRUCT_PAGE_INIT)

under “Memory Management options”.

### 5.8.3. Wide address space support

For components with support for 36-bit address space it is possible to access more than 4 GiB of memory. This requires enabling the wide address space support in the kernel. Enable it by selecting:

- “36-bit physical address support” (LEON_PHYS_36BIT)

under “Processor type and features”.

An application can only directly access less than 4 GiB of memory at the same time due to the virtual memory space still being 32-bit and the kernel using more than 256 MiB for its own mappings. To make full use of the memory one can run multiple applications that uses different parts of the physical memory, use the memory to hold a large file system, or manually map in and out memory to access the full address space.

For a memory layout starting at different memory ranges there might be a hole between the ranges. Using the default flat memory model this means that the gap will waste some memory in the page bookkeeping data structures. This can be avoided by switching to the sparse memory model instead. See Section 5.8.2.

#### 5.8.3.1. Mklinuximg configuration

Linux learns about the size of the regular memory in the lower 32-bit address space from Mklinuximg via the OPENPROM API. By default Mklinuximg in turn receives this information from a bootloader or from e.g. GRMON or TSIM acting as a bootloader.

For a block of memory placed at address 0x100000000 or above, information about its base and size needs to be passed in the devicetree information. The Mklinuximg xml configuration interface will be used to set up this information, just as for other devicetree information.

The following example xml file for Mklinuximg 2.0.18 or newer sets up an extra memory block at address 0x180000000 and size 0x80000000 (2 GiB).

```
<?xml version="1.0"?>
<matches>
  <match-root>
    <add-node name="memory">
      <add-prop name="device_type">
        <string>extra</string>
      </add-prop>
      <!-- base is expected to be a 64-bit value -->
      <add-prop name="base">
        <int>0x01</int>       <!-- Upper 32 bits -->
        <int>0x80000000</int> <!-- Lower 32 bits -->
      </add-prop>
      <!-- size is expected to be a 64-bit value -->
      <add-prop name="size">
        <int>0</int>          <!-- Upper 32 bits -->
        <int>0x80000000</int> <!-- Lower 32 bits -->
      </add-prop>
    </add-node>
  </match-root>
</matches>
```

## 5.9. PCI subsystem

See `Documentation/PCI` in the Linux kernel source tree for general documentation about the PCI subsystem and its kernel and user space interfaces in Linux.

The Linux PCI Host layer on a LEON system is used to perform device discovery using PCI Plug & Play, enumerating PCI buses found, allocating the PCI address space and basic device initialization. Its up to the PCI host bridge driver to set up and start the initialization. The LEON host bridge drivers assume that PCI have not been initialized by the bootloader since before.

---

This section only describes the LEON acting as a PCI Host. There is no specific support for PCI peripheral mode. In peripheral mode, the host typically has a driver for the LEON peripheral which communicates over the LEON peripheral main memory.

---

### 5.9.1. Kernel configuration for PCI

Enable PCI support in the Linux kernel by selecting:

- “Support for PCI and PS/2 keyboard/mouse” (PCI)

under “Bus options (PCI etc.)”

### 5.9.2. GRPCI driver

When configuring the kernel, select:

- "GRPCI Host Bridge Support" (SPARC_GRPCI1)

under "Bus options (PCI etc.)"

### 5.9.3. GRPCI2 driver

When configuring the kernel, select:

- "GRPCI2 Host Bridge Support" (SPARC_GRPCI2)

under "Bus options (PCI etc.)"

#### 5.9.3.1. Mklinuximg configuration

The GRPCI2 PCI Host bridge driver take configuration options controlled from the device tree. The configuration options are described in the table below.

*Table 5.5. GRPCI2 device tree properties*

| Name | Type | Default | Description |
|---|---|---|---|
| barcfg | 12 words | -1=Auto | Optional custom Target BAR configuration. The configuration property is an array of length 12 32-bit words, each pair of words describe one PCI target BAR set up. The first word in a pair describes the PCI address of BAR_N and the second the AMBA AHB base address translated into. The Target BAR size is calculated from the PCI BAR alignment. |
| irq_mask | word | 0=All | Limit which PCI interrupts are enabled. By default all are enabled. This property is typically used when one or more PCI interrupt pins are not used, or must be used when they are floating on the PCB. The property is a 4-bit mask where bit N controls Interrupt N. 0=Disable, 1=Enable.<br>• bit0 = PCI INTA#<br>• bit1 = PCI INTB#<br>• bit2 = PCI INTC#<br>• bit3 = PCI INTD# |
| reset | word, boolean | 0=no-rst | Force PCI reset on startup. If the property value is set to non-zero the GRPCI2 host driver will use the GRPCI2 register interface to reset the PCI bus on boot. |

## 5.10. Perf subsystem

`Perf` is a performance analysis tool in Linux. In LEON Linux, it can be used to gather performance data from statistics units, such as L4STAT, with the `perf stat` command. For general information about perf and its usage, see `man perf`. For examples on using `perf` with L4STAT, refer to Section 5.10.3.3.

### 5.10.1. Kernel configuration for `perf`

Enable `perf` support in the Linux kernel by selecting:

- "Kernel performance events and counters" (PERF_EVENTS)

under "General setup"

### 5.10.2. `Perf` in Buildroot

Under "Linux Kernel Tools" select:

- "`perf`" (BR2_PACKAGE_LINUX_TOOLS_PERF)

### 5.10.3. L4STAT driver

The L4STAT driver makes use of `perf` to provide a command line interface to the user. Performance counter statistics can be gathered with the `perf stat` command. This section covers the kernel and runtime configura-

tion, as well as usage examples. Additional details can be found in `Documentation/admin-guide/perf/l4stat_pmu.rst` in the Linux kernel source tree.

#### 5.10.3.1. Kernel configuration

To use L4STAT, select:

- "Gaisler L4STAT statistics unit support" (L4STAT_PMU)

under "Performance monitor support".

#### 5.10.3.2. Runtime Configuration

Some of the L4STAT counter control register (CCTRL) fields can be set at runtime via the `perf` interface. See Table 5.6

*Table 5.6. CCTRL fields in `perf`*

| L4STAT CCTRL field | `perf` config parameter alias | example parameter values |
|---|---|---|
| EVENT ID | event | 0x11, proc_total_instructions |
| AHBM | ahbm | 0, 1, 2, 3, 4, 5 |
| SU | su | 0, 2, 3 |

#### 5.10.3.3. Usage examples

List all available events

```
perf list
```

Some command formatting examples for counting the total number of instructions

```
perf stat -e proc_total_instructions sleep 1
perf stat -e l4stat/proc_total_instructions/ sleep 1
perf stat -e l4stat/event=0x11/ sleep 1
perf stat -e l4stat/config=0x11/ sleep 1
```

Some command formatting examples for specifying user and/or kernel space (u/k/uk)

```
perf stat -e proc_total_instructions:u sleep 1
perf stat -e l4stat/proc_total_instructions/u sleep 1
perf stat -e l4stat/event=0x11,su=2/ sleep 1
```

Count L2 cache misses for CPU3 (AHB master 3)

```
perf stat -e l4stat/ext_l2cache_miss,ahbm=3/ sleep 1
```

Count AHB BUSY cycles for all AHB masters in total (SU is 0 by default)

```
perf stat -e l4stat/ahb_busy_cycles/ sleep 1
```

Count AHB BUSY cycles for AHB master 4 (IO Memory Management Unit)

```
perf stat -e l4stat/ahb_busy_cycles,ahbm=4,su=1/ sleep 1
```

Count events where master 1 on the processor AHB has REQ asserted and master 2 on the processor AHB has GNT asserted

```
perf stat -e l4stat/reqgnt_ahbm1_proc,ahbm=2/ sleep 1
perf stat -e l4stat/event=0x81,ahbm=2/ sleep 1
```

Count events where master 1 on the processor AHB has REQ asserted and master 2 on the processor AHB has GNT deasserted

```
perf stat -e l4stat/req_ahbm1_proc,ahbm=2/ sleep 1
perf stat -e l4stat/event=0x91,ahbm=2/ sleep 1
```

## 5.11. PS/2 subsystem

### 5.11.1. Kernel configuration for PS/2

Enable PS/2 support in the Linux kernel by selecting:

- “Serial I/O support” (SERIO)

under “Hardware I/O ports” under “Input device support” under “Device drivers”.

### 5.11.2. APBPS2 driver

The APBPS2 driver follows the standard Linux kernel PS/2 interface;

When configuring the kernel, select:

- “GRLIB APBPS2 PS/2 keyboard/mouse controller” (SERIO_APBPS2)

under “Hardware I/O ports”.

## 5.12. SPI subsystem

The SPI subsystem in Linux only has support for SPI masters. See `Documentation/spi` in the Linux kernel source tree for general documentation about the SPI subsystem and its kernel and user space interfaces in Linux.

### 5.12.1. Kernel configuration for SPI

Enable SPI support in the Linux kernel by selecting:

- “SPI support” (SPI)

under “Device Drivers”.

See also Section 5.6 on how to set up GPIO to be able to use GPIO for chipselects.

### 5.12.2. SPICTRL driver

The driver for SPICTRL is integrated with the driver for the Freescale SPI controller. The driver follows the standard Linux SPI interface, including registering devices through the device tree.

When configuring the kernel, select:

- “Freescale SPI controller and Aeroflex Gaisler GRLIB SPI controller” (SPI_FSL_SPI)

under “SPI support”.

For the chipselect signal, the SPICTRL driver can use a combination of the slave select register of SPICTRL core (if available) and external GPIO cores (if configured). Each on the SPI bus where SPICTRL is the master has a chipselect number. If the slave select register is available, the slave select register will be used by default for chipselect numbers [0, slvselsz-1], where slvselsz is the number of slave select signals of the SPICTRL core. For other chipselect numbers, the default behavior of the driver is to do nothing (i.e. an always driven chipselect signal would be needed). The driver can be configured to use the GPIO subsystem to drive chipselect signals for some chipselect numbers. See `Documentation/devicetree/bindings/spi/spi-bus.txt` on needed devicetree nodes and properties for chipselects and slave devices and see below on how to realize this using mklinuximg.

#### 5.12.2.1. Mklinuximg configuration

The SPICTRL driver needs special devicetree properties. Mklinuximg from version 2.0.4 and onward adds these automatically.

To make the SPICTRL driver use a GPIO core to handle some or all of the chipselect signals an array property named “cs-gpios”. A chipselect that uses the default behavior (i.e. slave select register or always selected) should have one entry in the array containing 0. A chipselect that uses a GPIO line should have three entries in the array. The first entry is a handle to the GPIO core and is guaranteed to not be 0. This handle to the GPIO core is realized in mklinuximg by using the corelabel and corehandle tags. The second entry is the offset within the GPIO core (i.e. which GPIO line of the core to use). The third entry is a bitmask (see `include/dt-bindings/gpio/`

`gpio.h` for details). Typically (e.g. for the GRGPIO core), set bit 0 in the bitmask to 1 if the GPIO line is active-low.

Note that there is no way to know directly from chipselect number where in the array the corresponding entry or entries are. They are put one after another and are distinguished by if the first entry is 0 or not. The array is optional. If not present all chipselect numbers use the default behavior. The array can also be short. Any chipselect number with no entry in the array will use the default behavior.

To connect a SPI slave device to a SPI master through the device tree, a nodefor the slave device, containing some properties, should be put as a child of the node for the master device. A “register” property should contain the chipselect number address of the slave, a “compatible” property should contain a name that is matched by the driver of the slave, and a “spi-max-frequency” property should contain maximum clocking frequency of the slave. The drivers also supports the optional “spi-cpha”, “spi-cpol”, “spi-lsb-first” and “spi-cs-high” properties that are added as empty properties. See `Documentation/devicetree/bindings/spi/spi-bus.txt` for details.

Here follows a clarification about what the active pin state for chipselect will be for the different configurations.

For a SPI slave, if active-high is desired, then the device tree property “spi-cs-high” should be present in the device node. If the property is missing, then the chipselect will be handled as active-low.

For a SPI controller, when defining GPIO chipselects in the “cs-gpios” array, it should have its bitmask value set accordingly (i.e set bit 0 in the bitmask to 1 if the GPIO line is active-low or set the bit to 0 if it is active-high).

*Table 5.7. Chipselect pin state*

| Device node (SPI slave) | cs-gpio (bitmask value) | Chipselect pin state (active) | Note |
|---|---|---|---|
| spi-cs-high | - (Internal chipselect) | High | |
| - | - (Internal chipselect) | Low | |
| spi-cs-high | 0 (active-high) | High | |
| - | 0 (active-high) | Low | [1] |
| spi-cs-high | 1 (active-low) | High | [2] |
| - | 1 (active-low) | Low | |

[1]: The polarity inversion will be detected by Linux and an info text will be printed. The cs-gpio bitmask will be ignored and the cs pin state will be enforced active-low. The cs-gpio bitmask should be updated to 1 (active-low) to avoid the info text.

[2]: The polarity inversion will be detected by Linux and a warning will be printed. Since the “spi-cs-high” is specified it will override the bitmask in the cs-gpio. The cs-gpio bitmask should be updated to 0 (active-high) to avoid the warning.

The following example xml file for mklinuximg sets up the first SPICTRL core to use default chipselect behavior for chipselect numbers 0 and 2, and to use GPIO line 27 of the first GRGPIO core for chipselect number 1. It also adds a AD7814 temperature sensor SPI slave device with chipselect number 0 to the device tree under this SPICTRL core (that would need a driver matching “adi,ad7814”).

```
<?xml version="1.0"?>
<matches>
  <match-core vendor="VENDOR_GAISLER" device="GAISLER_SPICTRL" index="0">
    <!-- chipselect setup -->
    <add-prop name="cs-gpios">
      <!-- chipselect 0: default -->
      <int>0</int>
      <!-- chipselect 1: line 27 of "gpio0" -->
      <corehandle ref="gpio0"/>
      <int>27</int>
      <int>0</int>
      <!-- chipselect 2: default -->
      <int>0</int>
    </add-prop>

    <!-- SPI slave -->
    <add-node name="ad7814">
      <add-prop name="reg">
        <int>0</int> <!-- chipselect number -->
```

```
      </add-prop>
      <add-prop name="spi-max-frequency">
        <int>10000000</int>
      </add-prop>
      <add-prop name="spi-cpha"/> <!-- Shifted clock phase mode -->
      <add-prop name="spi-cpol"/> <!-- Inverse clock polarity mode -->
      <add-prop name="compatible">
        <string>adi,ad7814</string>
      </add-prop>
    </add-node>
  </match-core>

  <match-core vendor="VENDOR_GAISLER" device="GAISLER_GPIO" index="0">
    <corelabel name="gpio0"/> <!-- Used by cs-gpios to refer to this core -->
  </match-core>
</matches>
```

### 5.12.3. Important changes between Linux 4.9 and Linux 6.13

The driver for SPICTRL was converted to support the GPIO descriptor feature in Linux 5.4. In this effort, a lot of the chipselect GPIO related logic was removed from the driver and instead handled by common functions in the SPI core. With this change, one of the differences is that the bitmask value of the entries in “cs-gpios” (i.e bit 0: GPIO_ACTIVE_HIGH(0)/GPIO_ACTIVE_LOW(1)) is used and should match the presence/absence of “spi-cs-high” in the SPI slave node. If Linux detect polarity inversion in the setup of “cs-gpios” between a SPI slave and the SPI master it will enforce the active chipselect pin state according to Table 5.7.

### 5.12.4. SPI user space interface

In Linux it is possible to enable a user mode SPI device driver called spidev. The spidev driver export some functions of a SPI device so a user space application can access the device, for example to send and receive data. See `Documentation/spi/spidev.rst` for details.

Enable spidev in the Linux kernel by selecting:

- “User mode SPI device driver support” (SPI_SPIDEV)

under “SPI support”.

The spidev driver can be configured in a few different ways. One example is to add support for a SPI device in the spidev driver so it automatically can be configured in the device tree. Another example is to explicitly bind the SPI device to the spidev driver through sysfs. These examples will be described in detail below. Please refer to the official spidev documentation for a complete list of alternatives.

It is good to know that it used to be supported to define an SPI device by specifing “spidev” in the compatible string of the device. But this is no longer supported by the Linux kernel.

Example 1: Enable spidev for a SPI slave device from the device tree

This example is based on the xml from Section 5.12.2.1 and shows how the AD7814 SPI slave device automatically can bound to a spidev device by adding the compatible string for AD7814 to the list of supported devices in the spidev driver.

- Open drivers/spi/spidev.c in the Linux source tree
- Locate the “spidev_dt_ids” array
- Add the compatible string (“adi,ad7814”) to the array
- Re-compile

The AD7814 can now be accessed from user space using `/dev/spidev0.0`

Example 2: Enable spidev for a SPI slave device through sysfs

This example is based on the xml from Section 5.12.2.1 and shows how the AD7814 SPI slave device can bound to a spidev device through sysfs.

```
# Bind spi0.0 (ad7814) to spidev
echo spidev > /sys/bus/spi/devices/spi0.0/driver_override
echo spi0.0 > /sys/bus/spi/drivers/spidev/bind
# /dev/spidev0.0 is now available

# Unbind
echo spi0.0 > /sys/bus/spi/drivers/spidev/unbind
# /dev/spidev0.0 is no longer available
```

#### 5.12.4.1. The spidev_test test utility

The spidev_test application, available within the Linux kernel, is a test tool to perform tests on a spidev device.

#### 5.12.4.2. The spi-tools toolset

The spi-tools toolset is useful for quick testing and diagnostics of spidev devices in Linux and the source code also serves as examples on spidev programming from user space. **spi-config** can be used for querying (or setting) the configuration of a SPI device. **spi-pipe** is a tool that send and receive data simultaneously to and from a SPI device.

### 5.12.5. SPI in Buildroot

The “spi-tools” package contains the spi-tools toolset (see Section 5.12.4.2). Under “Hardware handling” select:

- “spi-tools” (BR2_PACKAGE_SPI_TOOLS)

The “spidev_test” package contains the spidev_test tool (see Section 5.12.4.1). Under “Debugging, profiling and benchmark” select:

- “spidev_test” (BR2_PACKAGE_SPIDEV_TEST)

## 5.13. Timers subsystem

See `Documentation/timers` in the Linux kernel source tree for general documentation about timekeeping, high resolution timers and other timer related functions.

### 5.13.1. Clock sources

A clock source in Linux refers to a monotonic, atomic counter that wraps around and never stops ticking, and can thus maintain a timeline. There can be, and often are, multiple clock sources available. Only one clock source is actively in use as the system clock. It can be selected automatically, e.g. by using clock source rating, manually by overwriting the current clock source (through sysfs) or by specifing the “clocksource=<clocksourcename>” as a kernel parameter.

### 5.13.2. Clock event devices

A clock event device in Linux refers to a device that can be setup to trigger an event (using interrupts) on the timeline, either once (oneshot mode) or periodically.

### 5.13.3. High Resolution Timers

The accuracy of various system calls that set timeouts, and measure CPU time is limited by the resolution of the software clock, a clock maintained by the kernel which measures time in jiffies. The size of a jiffy is determined by the value of the kernel constant HZ. For example, when HZ is set to 100, giving a jiffy value of 0.01 seconds. On a system that supports high resolution timers, the accuracy of sleep and timer system calls is no longer constrained by the jiffy, but instead can be as accurate as the hardware allows. To qualify as a high resolution timer, a clock event device must support one-shot mode (i.e. scheduling events at arbitrary future times). Devices restricted to periodic mode remain low-resolution. On an SMP system, it is ideal (and customary) to have a clock event device per CPU core, so that each core can trigger events independently of any other core. See `Documentation/timers/hrtimers.rst` for more information about high resolution timers.

### 5.13.4. LEON Linux default system timer

In LEON Linux, the default timer serves as both the clock source and the clock event device. The clock source is a free-running software counter that is periodically updated by a clock event device, typically connected to the boot CPU. The clock event driver consumes one dedicated GPTIMER core, runs at a 1 MHz tick rate, and supports only periodic mode so it cannot provide high-resolution timing. In a SMP system, when a clock event triggers on the GPTIMER, the interrupt is broadcasted to the other CPUs (setup through IRQMP) allowing each clock event device (per-CPU) to maintain synchronized timekeeping.

### 5.13.5. Clocksource using the up-counter in LEON

The “Clocksource using the up-counter in LEON” driver utilize the up-counter in LEON as clock source, when available. Not all LEON CPUs support the counter but components like GR740 have support for it.

When configuring the kernel, select:

- “Common Clock Framework” (COMMON_CLK)

under “Device drivers”.

Now the clock source driver can be selected:

- “Clocksource using the up-counter in LEON” (CLKSRC_LEON)

under “Device drivers” under “Clock Source drivers”.

#### 5.13.5.1. Mklinuximg configuration

The “Clocksource using the up-counter in LEON” driver needs special devicetree properties. Mklinuximg from version 2.0.19 and onward can add these when the option `-hrtimers` is set. If the running system does not have an up-counter then the device tree bindings for the driver will not be generated. Additionally, if the full requirements mentioned in Section 5.13.6.1 for clock event devices are not fulfilled, the `-hrtimers` option will not generate any device tree bindings for the “Clocksource using the up-counter in LEON” driver either.

### 5.13.6. Frontgrade Gaisler GPTIMER timer driver

The timer driver for GPTIMER follows the standard Linux interface for registering clock event and clock source devices. By supporting periodic and one-shot modes, it can operate as a high-resolution timer. The driver setup a clock event device per CPU and requires a unique interrupt (can't be shared with another device) to work as expected.

The Frontgrade Gaisler GPTIMER timer driver and the LEON Linux default system timer both potentially use the same GPTIMERs and cannot operate simultaneously. Therefore, when Frontgrade Gaisler GPTIMER timer driver is enabled, the default timer is disabled during boot. It is recommended that the Frontgrade Gaisler GPTIMER timer driver be enabled only on systems configured to use high-resolution timers.

Enable “High resolution timer” support in the Linux kernel by selecting:

- “High Resolution Timer Support” (HIGH_RES_TIMERS)

under “General setup” under “Timers subsystem”.

Enable clock support:

- “Common Clock Framework” (COMMON_CLK)

under “Device drivers”.

Now the timer driver can be selected:

- “Frontgrade Gaisler GPTIMER timer driver” (GAISLER_GPTIMER_TIMER)

under “Device drivers” under “Clock Source drivers”.

#### 5.13.6.1. Mklinuximg configuration

To be able to use high resolution timers using the “Frontgrade Gaisler GPTIMER timer driver” devicetree properties are needed. Mklinuximg from version 2.0.19 can generate the necessary bindings when the `-hrtimers` option is provided. Mklinuximg will then setup which GPTIMER subtimers that will be allocated as clock event devices and which clock source to use.

For the clock event devices, Mklinuximg will try to setup as many subtimers with unique timer interrupts as there are CPUs in the system. Mklinuximg will use as many subtimers it can from each GPTIMER until it has setup all the necessary subtimers.

When Mklinuximg selects subtimers for clock event devices it takes the following two cases into consideration:

- The last subtimer of GPTIMER0 is reserved by the GPTIMER Watchdog and will not be included, i.e. no bindings for the timer drivers will be setup for this subtimer. This is applicable to both clock source and clock event devices.
- If a GPTIMER core does not support separate interrupts, only the first subtimer will be setup for clock event devices. However, the clock source device does not need an interrupt and can still use the next subtimer (as long as it's not restricted by the first case above).

After the clock event devices have been setup, Mklinuximg will choose from two clock sources:

- If Mklinuximg detects during boot, that the up-counter in LEON is available and the “Clocksource using the up-counter in LEON” driver has been enabled (see Section 5.13.5) in Linux then Mklinuximg will prefer it as clock source and only generate bindings for this driver.
- If the up-counter is not available (or hasn't been enabled in the kernel) then Mklinuximg will try to find an available subtimer of a GPTIMER core and add the necessary device tree bindings for the clock source driver implemented by the Frontgrade Gaisler GPTIMER timer driver. Mklinuximg will choose the first available subtimer after all subtimers for the clock event devices have been allocated.

Depending on the availability of GPTIMER cores, number of implemented subtimers of the GPTIMER cores and the number of CPUs it might not be possible to setup the required amount of resources. In that case, Mklinuximg will skip adding any devicetree bindings and Linux will use the default system timer. As mentioned in Section 5.13.6, an interrupt used by one of the clock event devices must be unique and cannot be shared with another device. For some systems the selected subtimers share interrupts with other cores. Such a situation currently must be resolved by not including any kernel drivers for colliding cores, removing cores using the XML file support (see the Mklinuximg User`s manual [https://download.gaisler.com/products/leon-linux/doc/mklinuximg.pdf] ) or do changes to the Mklinuximg source code to select a suitable set of subtimers.

The `-d` option, when given to Mklinuximg, will print some extra information related to how Mklinuximg allocate the subtimers for the timer driver.

In case the `-hrtimers` is set, Mklinuximg will, at RAM-image creation, make sure that the “Frontgrade Gaisler GPTIMER timer driver” has been enabled in the Linux kernel.

```
$ make

ERROR: -hrtimers specified but no timer driver enabled in Linux.
Did you enable "Frontgrade Gaisler GPTIMER Timer Driver"?

make: *** [Makefile:23: _all] Error 2
```

### 5.13.7. User space interface

In Linux it is possible to read information about the clock source and clock event devices through sysfs. On a system using the default system timer the clock source and clock event devices are listed as follows:

```
# The default clock source:
$ cat /sys/devices/system/clocksource/clocksource0/current_clocksource
timer_cs

# Clock event devices (as listed on GR740):
$ ls /sys/devices/system/clockevents
clockevent0  clockevent1  clockevent2  clockevent3  uevent

# Check current device for the clockevents:
$ cat /sys/devices/system/clockevents/clockevent*/current_device
percpu_ce
percpu_ce
percpu_ce
percpu_ce
```

When “Clocksource using the up-counter in LEON” is enabled and used as clock source:

```
$ cat /sys/devices/system/clocksource/clocksource0/current_clocksource
leon_cs
```

Here follows an example of how it will look when the “Frontgrade Gaisler GPTIMER timer driver” has registered both a clock source and clock event devices:

```
# When a GPTIMER sub-timer is used as clock source:
$ cat /sys/devices/system/clocksource/clocksource0/current_clocksource
gptimer_cs

# Clock event devices (as listed on GR740):
$ ls /sys/devices/system/clockevents
clockevent0  clockevent1  clockevent2  clockevent3  uevent

# Check current device for the clockevents:
$ cat /sys/devices/system/clockevents/clockevent*/current_device
gptimer_ce
gptimer_ce
```

```
gptimer_ce
gptimer_ce
```

In addition to the above examples there is a file called `/proc/timer_list` which exposes a list of all currently pending (high-resolution) timers, all clock-event sources, and their parameters in a human-readable form.

## 5.14. TTY subsystem

### 5.14.1. Kernel configuration for TTY

Enable TTY support in the Linux kernel by selecting:

- "Enable TTY" (TTY)

under "Character devices".

### 5.14.2. APBUART driver

The APBUART driver follows the standard Linux TTY kernel interface.

#### 5.14.2.1. Kernel configuration

When configuring the kernel, select:

- "GRLIB APBUART serial support" (SERIAL_GRLIB_GAISLER_APBUART)

under "Serial drivers" under "Character devices". To use APBUART driver for system console also select:

- "Console on GRLIB APBUART serial port" (SERIAL_GRLIB_GAISLER_APBUART_CONSOLE)

under the former.

#### 5.14.2.2. Mklinuximg configuration

The APBUART driver ignores an APBUART core if it has an "amptopts" property with a value of 0. This is useful when running an AMP where certain uarts are used by another operating system. The ampopts property can be set in two different ways:

One way to add amptopts properties is to add it using the `-amp` flag with mklinuximg. The flag takes as an argument a string on the form "idx0=val0:idx1=val1:..." that adds an "amptopts" property with value val0 for the core with scan index idx0, an "amptopts" property with value val1 for the core with scan index idx1, etc.

The other way to add an ampopts property is to add it via xml. For the APBUART case where the only function of amptopts is for the core to be ignored, even easier is to delete the node of the core altogether. The following example adds an amptopts property with value 0 to the first APBUART and outright deletes the third from the devicetree:

```
<?xml version="1.0"?>
<matches>
  <match-core vendor="VENDOR_GAISLER" device="GAISLER_APBUART" index="0">
    <add-prop name="ampopts">
      <int>0</int>
    </add-prop>
  </match-core>

  <del-core vendor="VENDOR_GAISLER" device="GAISLER_APBUART" index="2"/>
</matches>
```

### 5.14.3. Mklinuximg configuration

To set up an UART as system console, use the **console** kernel parameter to choose UART and baud rate. Commands line options can be set using the `-cmdline` option of mklinuximg. For example, using mklinuximg with **-cmdline "console=ttyS0,38400"** sets up console on ttyS0 (which is the first UART in the system) with baud rate of 38400.

## 5.15. USB device subsystem

See `Documentation/usb` in the Linux kernel source tree for general documentation about the USB device subsystem and its kernel and user space interfaces in Linux.

### 5.15.1. Kernel configuration for USB device

Enable USB device support in the Linux kernel by selecting:

- “USB support” (USB_SUPPORT), and under there
- “USB Gadget Support” (USB_GADGET)

under “Device drivers”.

Under “USB Gadget Support” select the preferred gadget driver.

### 5.15.2. GRUSBDC driver

The GRUSBDC driver follows the standard Linux device periferal controller interface and will be matched with the selected gadget driver. The driver only supports AHB master mode.

#### 5.15.2.1. Kernel configuration

When configuring the kernel, select:

- “Aeroflex Gaisler GRUSBDC USB Peripheral Controller Driver” (USB_GR_UDC)

under “USB Peripheral Controller” under “USB Gadget Support”.

The periferal controller driver will automatically bind to the selected gadget. No further configuration is necessary.

To get enable debug printouts and to enable detailed information in debugfs, select:

- “Debugging messages (DEVELOPMENT)” (USB_GADGET_DEBUG)
- “Debugging information files in debugfs (DEVELOPMENT)” (USB_GADGET_DEBUG_FS)

under “USB Gadget Support”.

#### 5.15.2.2. Mklinuximg configuration

If the GRUSBDC has non-default (1024) buffer sizes for the endpoints, this needs to be specified in the device-tree using the property “epobufsizes” for OUT endpoints and “epibufsizes” for IN endpoins. Fewer entries than endpoints overrides the default sizes only for as many endpoints as the array contains. See also `Documentation/devicetree/bindings/usb/gr-udc.txt` in the Linux kernel source tree.

The following example xml file specifies for the first GRUSBDC core that OUT endpoint 1 has a non-default buffer size of 2048:

```
<?xml version="1.0"?>
<matches>
  <match-core vendor="VENDOR_GAISLER" device="GAISLER_USBDC" index="0">
    <add-prop name="epobufsizes">
      <int>1024</int> <!-- OUT endpoint 0 has default buffer size 1024 -->
      <int>2048</int> <!-- OUT endpoint 1 has non-default buffer size 2048 -->
      <!-- OUT endpoint 2 has default buffer size 1024, but need no entry -->
    </add-prop>
  </match-core>
</matches>
```

## 5.16. USB host subsystem

See `Documentation/usb` in the Linux kernel source tree for general documentation about the USB host subsystem and its kernel and user space interfaces in Linux.

### 5.16.1. Kernel configuration for USB host

Enable USB host support in the Linux kernel by selecting:

- “USB support” (USB_SUPPORT)
- “Support for Host-side USB” (USB)

under “Device drivers” and under “Device drivers/USB support” respectively.

Under there are a myriad of options and drivers for various USB devices.

### 5.16.2. GRUSBHC EHCI driver

The GRUSBHC core suports both EHCI and UHCI standard interfaces. To use the EHCI driver interface, select:

- "EHCI HCD (USB 2.0) support" (USB_EHCI_HCD)

under "USB support" when configuring the kernel.

### 5.16.3. GRUSBHC UHCI driver

The GRUSBHC core suports both EHCI and UHCI standard interfaces. To use the UHCI driver interface, select:

- "UHCI HCD (most Intel and VIA) support" (USB_UHCI_HCD)

under "USB support" when configuring the kernel.

## 5.17. Watchdog subsystem

See `Documentation/watchdog` in the Linux kernel source tree for general documentation about the watchdog subsystem and its kernel and user space interfaces in Linux.

Before reading the watchdog section its good to read the following descriptions about the different timeouts that are referred to in this section.

*Table 5.8. Watchdog timeout nomenclature for this section*

| Name | Description |
|---|---|
| hw-timeout | The hw-timeout is the maximum time between pings to the watchdog hardware by the responsible driver. |
| user-timeout | The user-timeout is the maximum time between pings to the kernel watchdog framework by userspace. |

The "hw-timeout" is provided to the driver and can not be changed after boot. If a ping is not made within "hw-timeout" then the system will restart. On some components, this timeout is static due to hardware requirements but on others it might be configurable (by a bootloader).

After user space has opened the watchdog device, the watchdog framework (in kernel) will expect that user space ping the watchdog within "user-timeout", outside the "user-timeout" the watchdog is left unattended and will ultimately, when exceeding "hw-timeout" lead to a restart of the system. The user space application use the "WDIOC_SETTIMEOUT"/"WDIOC_GETTIMEOUT" ioctl to set/get "user-timeout" in the watchdog framework (in kernel). The default "user-timeout" is set by the watchdog driver.

It is possible to set a larger "user-timeout" than "hw-timeout". In this scenario, the kernel will ensure to ping the watchdog up until the "user-timeout" and then it expect user space to ping. When the "user-timeout" is smaller than the "hw-timeout" the kernel will not schedule any pings in between and leave the responsibility to user space.

### 5.17.1. Kernel configuration for Watchdog

When configuring the kernel, first select:

- "Watchdog Timer Support" (WATCHDOG)

under "Device Drivers".

Optionally, the following option can be enabled to configure the kernel to ping the watchdog on boot until user space is ready to take over:

- "Update boot-enabled watchdog until user space ..." (WATCHDOG_HANDLE_BOOT_ENABLED)

under "Watchdog Timer Support".

If option (WATCHDOG_HANDLE_BOOT_ENABLED) is enabled, then a timeout value for opening the watchdog needs to be specified.

The default value is **infinite** (0), a suitable timeout for the system should be specified that covers the time loading user space including a margin of safety.

Set the maximum timeout the kernel will ping the watchdog on boot by specifying:

- "Timeout value for opening watchdog device" (WATCHDOG_OPEN_TIMEOUT)

under “Watchdog Timer Support”.

---

If the default value is used for (WATCHDOG_OPEN_TIMEOUT) after enabling (WATCHDOG_HANDLE_BOOT_ENABLED) then the watchdog will not restart the system if user space fails to load as the kernel will continue to ping the watchdog.

---

### 5.17.2. GPTIMER Watchdog Driver

The driver supports the GPTIMER watchdog timer present on many SPARC LEON systems (including the PLL hardware watchdog on GR740) and it follows the standard Linux Watchdog driver interface.

Enable “Frontgrade Gaisler GPTIMER Watchdog” driver by selecting:

- “Frontgrade Gaisler GPTIMER Watchdog” (GPTIMER_WATCHDOG)

under “Watchdog Timer Support”.

#### 5.17.2.1. Mklinuximg configuration

The “Frontgrade Gaisler GPTIMER Watchdog” driver needs special devicetree properties. Mklinuximg from version 2.0.16 and onward has support for generating them.

The `-watchdog` option enables the watchdog function. It adds the necessary device tree additions that the Linux driver requires. Also, when the watchdog function is enabled, Mklinuximg pings the watchdog during boot:

- Before starting the kernel.
- When building the device tree.
- On UART console printouts (i.e earlyconsole prints)

The number of pings allowed for each item is limited.

The `-watchdog-user-timeout` option can be set to override the default “user-timeout” (30 seconds) set by the GPTIMER watchdog driver (see Table 5.8). The flag expects a timeout value in seconds: “-watchdog-user-timeout timeout_value”

Mklinuximg expects that the watchdog timer (the last implemented timer of the first timer unit) has been initialized prior to application start by the bootloader.

The following registers are expected to be set:

*Timer control register (TCTRL) of the watchdog timer*

- Enable (EN)
- Interrupt Enable (IE)

The timer should be enabled (EN) and interrupt enabled (IE) as Mklinuximg nor the GPTIMER watchdog driver writes these settings.

*Timer counter reload value register (TRLDVAL) of the watchdog timer*

- “hw-timeout” in µs

Mklinuximg reads the reload value and use it when setting up the device tree properties required by the GPTIMER watchdog driver. On some components though, a static timeout value will be used.

---

If the reload register is not set, then Mklinuximg will not be able to set up the required device tree properties the GPTIMER watchdog driver will not register on boot.

---

### 5.17.3. Early Watchdog Support

The Linux watchdog subsystem is not available until after the system is done with init tasks like setting up the CPUs, memory and other fundamental kernel functions. This means that its not possible to register a watchdog driver through the watchdog subsystem at this point, leaving the watchdog unattended until the system reach the point where the subsystems are ready to be initialised.

When the watchdog subsystem is ready it can be configured to continue to ping the watchdog until userspace takes over. Please see the description regarding (WATCHDOG_HANDLE_BOOT_ENABLED) in Section 5.17.1.

The Early Watchdog Driver is registered when the system reaches the early initcall level and pings the watchdog until the watchdog subsystem is ready to take over the responsibility (or until it timeout and will then stop.).

Define a maximum timeout (in seconds) for the Early Watchdog driver (default 15 seconds):

- “Timeout value ...” (SPARC_LEON_EARLY_WDT_TIMEOUT)

under “Processor type and features” under “Sparc Leon processor family”

#### 5.17.3.1. Early GPTIMER Watchdog Driver

The Early GPTIMER Watchdog driver supports the watchdog timers mentioned in Section 5.17.2

Enable the “Early GPTIMER Watchdog Driver” driver by selecting:

- “Early GPTIMER Watchdog Driver” (SPARC_LEON_EARLY_GPTIMER_WDT)

under “Processor type and features” under “Sparc Leon processor family”.

#### 5.17.3.2. Mklinuximg configuration

Please refer to Section 5.17.2.

### 5.17.4. GRWATCHDOG Driver

The “Frontgrade Gaisler GRWATCHDOG” driver follows the standard Linux Watchdog driver interface.

Enable “Frontgrade Gaisler GRWATCHDOG” driver by selecting:

- “Frontgrade Gaisler GRWATCHDOG” (GRWATCHDOG_WDT)

under “Watchdog Timer Support”.

#### 5.17.4.1. Pretimeout governor

The watchdog framework has a feature called the pretimeout governor framework. When the pretimeout governor framework is enabled, it allows for a custom action to be carried out before the watchdog restart the system. The GRWATCHDOG core supports asserting a pretimeout interrupt some time before the system is restarted. The “Frontgrade Gaisler GRWATCHDOG” driver can be configured to handle the interrupt which then notifies the pretimout governor framework in case the interrupt is triggered. The pretimeout handling in the driver requires an additional device tree property: “interrupts” (see Section 5.17.4.2).

Enable “watchdog pretimeout governors” by selecting:

- “Enable watchdog pretimeout governors” (WATCHDOG_PRETIMEOUT_GOV)

under “Watchdog Timer Support”.
The default governor puts the kernel into panic.

Change the selected governor by changing option under:

- “Default Watchdog Pretimeout Governor”

under “Watchdog Timer Support” under “Enable watchdog pretimeout governors”

Note that in Linux, interrupt 15 (NMI) is used by the kernel so the GRWATCHDOG core should use a different interrupt for the pretimeout feature to work.

#### 5.17.4.2. Mklinuximg configuration

Mklinuximg from 2.0.18 and onward will, when the `-watchdog` option is set, ping (ensure that the challenge/response protocol of the GRWATCHDOG core is maintained) during early boot up until the Linux driver is ready to take over the responsibility.

Mklinuximg will ping the GRWATCHDOG watchdog at the same places as mentioned for the GPTIMER watchdog (see Section 5.17.2.1).

The “Frontgrade Gaisler GRWATCHDOG” driver also needs special devicetree properties. Mklinuximg does not add these properties automatically for the GRWATCHDOG core, instead the xml configuration interface should be used.

Required device tree properties:

- “compatible” should be set to “gaisler,grwatchdog-wdt”.
- “gaisler,max-hw-timeout-ms” should be set to the maximum timeout of the GRWATCHDOG core until system is restarted.

Optional device tree properties:

- “timeout-sec”, if specified, should be set to the number of seconds of the user-timerout. If not specified then the driver will use the default 30 second user-timeout.
- “interrupts”, if specified, should be set to the interrupt number of the pretimeout interrupt generated by the GRWATCHDOG core. If not specified then the pretimeout handling in the driver is omitted.

The following example xml set up the device tree information.

```
<?xml version="1.0"?>
<matches>
  <match-core vendor="VENDOR_GAISLER" device="GAISLER_GRWATCHDOG" index="0">
    <!-- Required properties -->
    <add-prop name="gaisler,max-hw-timeout-ms">
      <int>5000</int>
    </add-prop>
    <add-prop name="compatible">
      <string>gaisler,grwatchdog-wdt</string>
    </add-prop>

    <!-- Optional properties -->
    <!-- Use a 45 second user timeout -->
    <!--
        <add-prop name="timeout-sec">
          <int>45</int>
        </add-prop>
    -->
    <!-- Enable the pretimeout handling be specifing an interrupt -->
    <!--
        <add-prop name="interrupts">
          <int>IRQ_NUMBER</int>
        </add-prop>
    -->
  </match-core>
</matches>
```

### 5.17.5. Watchdogs with short maximum hw-timeout

On some components there might be watchdogs with a short maximum “hw-timeout” (for example the PLL watchdog on GR740). Depending on the “hw-timeout” and the system specification, further configuration of the kernel might be required.

The time spent in the early phase of the boot can be shorten by enabling the option described in Section 5.8.2 called “Defer initialisation of struct pages until kthreads”. When this option is enabled, a small portion of the memory is initialised on early boot and later when the kernel has started some of its core functions the remaining memory gets initialized.

It is recommended to enable a “Early Watchdog Driver”. (Please see Section 5.17.3) to ensure that the watchdog is pinged until the watchdog subsystem is ready.

When the watchdog subsystem is ready it can be configured to continue to ping the watchdog until user space takes over. Please see how to configure (WATCHDOG_HANDLE_BOOT_ENABLED) in Section 5.17.1.

### 5.17.6. Watchdog in Buildroot

There are a few available watchdog applications in Buildroot. Before use they should be configured to fit the system.

*Table 5.9. Watchdog packages under “System Tools”*

| Package | Description |
|---|---|
| BR2_PACKAGE_WATCHDOGD | watchdogd is an advanced system and process supervisor daemon. |
| BR2_PACKAGE_BUSYBOX_WATCHDOG | This package enables the busybox watchdog daemon on boot. The functionality of this service is less advanced compared to watchdogd. |

| Package | Description |
| --- | --- |
| BR2_PACKAGE_BUSYBOX_WATCHDOG_PERIOD | If the busybox watchdog daemon is enabled then this config specifies the “user-timeout” which control the time between pings from user space. (see Table 5.8). |

When configuring the watchdog application on a multi-processor system care should be made to ensure that all CPUs are monitored to find potential hard (and soft) lockups on the CPUs.

# 6. Reboot

Reboot of the system is handled through standard Linux interfaces.

For example using the **reboot** command:

```
# reboot
The system is going down NOW!
Sent SIGTERM to all processes
Sent SIGKILL to all processes
reboot: Restarting system
```

The control is then handed over to Mklinuximg. When Mklinuximg has been configured with the `-watchdog` it will try to perform an instant reboot using the watchdog.

This requires that the active watchdog supports changing timeout. Mklinuximg supports both the GPTIMER and GRWATCHDOG watchdogs but only the timeout of the GPTIMER watchdog can be changed. This means that for a GRWATCHDOG, the system might have to wait a full timeout period until the reboot occurs.

For a system without a watchdog. The reset handling needs to be implemented in the function `leon_reboot` in `mklinuximg/src/prom.c`.

# 7. Real-time Linux (PREEMPT_RT)

Fully-preemptible kernel support (PREEMPT_RT) was finally merged into mainline Linux 6.12. Enabling real-time capabilities in Linux makes the system more responsive and predictable, which is essential for applications that require precise timing and low-latency performance. There is no mainline support for SPARC32 yet, but it has been added to the LEON Linux 6.13 Beta release. This section will cover how to configure a PREEMPT_RT enabled kernel and how to run some tools and scripts to measure latencies.

## 7.1. Latencies

These values represent typical latency measurements observed during testing. Actual results may vary based on test input, system configuration, network conditions, and other factors.

*Table 7.1. Measurements*

| System load | Max latencies (Non-preempt) | Max latencies (PREEMPT_RT) |
|---|---|---|
| Idle | ~150-350 µs | ~150-350 µs |
| Heavy load | ~3000-15000 µs | ~400-500 µs |

The mesurements were collected by the “cyclictest” application (see Section 7.5.1 for more information about how the latencies are measured) and the heavy load test used a combination of running “hackbench” togheter with network traffic to introduce system load.

## 7.2. Switching to a Fully Preemptible Kernel: Key Considerations

When switching from a non-preempt kernel to a fully-preemptible kernel one needs to be aware about some key features that comes with it.

- Trade-offs

It is important to know that even though PREEMPT_RT is great for real-time systems, it isn't always the best choice for every situation. For example it may lower overall performance for background work (and/or network throughput). Often additional tuning is needed, tailored to the application, to achive a high-perfomant system.

- Locks are handled differently

For example spinlocks/rwlocks are converted to rt_mutexs (preemtible, can sleep). More information about the lock types can be found in the kernel documentation `Documentation/locking/locktypes.rst`

- Interrupts are handled in kernel threads

Which means that they are preemptible, can sleep and have scheduling priority. There are still a few hard IRQ handlers which run in true IRQ context.

- Priority inheritance

When a task blocks on an rt_mutex, priority inheritance will make sure that a low priority task temporarily inherits the priority of a higher-priority task if it holds a resource that the higher-priority task is currently blocked on or needs to acquire. raw_spinlocks are not converted to rt_mutex (i.e does not support priority inheritance) - important to keep in mind to avoid priority inversion

- Preemption can happen “anywhere”

In a non-preemptible kernel, tasks can only be preempted at specific points (when a system call returns or when an interrupt occurs.) In contrast, with a fully-preemptible kernel, most parts of the kernel become preemptible. This means that a running task in the kernel can be preempted almost anywhere, not just at system call exits or interrupt points. Only a few carefully selected critical sections remain non-preemptible.

- High-resolution timers

PREEMPT_RT relies on high-resolution timers to achieve precise and deterministic task scheduling required for real-time performance. They allow the kernel to wake up tasks at exact times, minimizing latency and jitter. Without high-resolution timers, the kernel's timing granularity would be limited to the system tick, which is too coarse for real-time guarantees. See Section 5.13.3 for more information on high-resolution timers.

## 7.3. Kernel configuration

This section describes how to enable real-time support with PREEMPT_RT on a LEON based system including some useful options.

### 7.3.1. Mandatory options

Start by enabling kernel configuration for expert users by selecting:

- “Expert users” (EXPERT)

under “General setup”.

Enable PREEMPT_RT by selecting:

- “Fully Preemptible Kernel (Real-Time)” (PREEMPT_RT)

under “General setup”.

Enable high resolution timers by selecting:

- “High Resolution Timer Support” (HIGH_RES_TIMERS)

under “General setup” under “Timers subsystem”.

Enable common clock framework support:

- “Common Clock Framework” (COMMON_CLK)

under “Device Drivers”.

Enable the Frontgrade Gaisler GPTIMER timer driver:

- “Frontgrade Gaisler GPTIMER timer driver”(GAISLER_GPTIMER_TIMER)

under “Device Drivers” under “Clock Source Drivers”.

### 7.3.2. Optional options

The “Frontgrade Gaisler GPTIMER timer” includes a clock source but even better is to use the up-counter in LEON as clock source, when available.

Enable the LEON clock source driver by selecting:

- “Frontgrade Gaisler GPTIMER timer driver”(GAISLER_GPTIMER_TIMER)
- “Clocksource using the up-counter in LEON” (CLKSRC_LEON)

under “Device Drivers” under “Clock Source Drivers”.

The default timer frequency is set to 100HZ which may be increased to improve responsiveness, for example by selecting:

- “300 HZ” (HZ_300)

under “Processor type and features” under “Timer frequency”.

The default preemption model after enabling the fully preemptible kernel is “Preemptible Kernel (Low-Latency Desktop)”. In Linux 6.13, a new preemption model was introduced called “Scheduler controlled preemption” (PREEMPT_LAZY). The main difference is that lazy preemption is less aggressive about preempting SCHED_NORMAL tasks, allowing them to continue running if the preemption is not urgent.

To change preemption model to lazy preemption, choose:

- “Scheduler controlled preemption model” (PREEMPT_LAZY)

under “General setup”.

### 7.3.3. Debugging options

The kernel includes a valuable debugging feature for PREEMPT_RT systems that enables runtime checks on RT-mutexes to detect misuse, deadlocks, and priority inheritance issues. It’s useful for validating real-time locking behavior during development but introduces overhead, making it unsuitable for production environments. To enable it, select:

- “RT Mutex debugging, deadlock detection” (DEBUG_RT_MUTEXES)

under “Kernel hacking” under “Lock Debugging (spinlocks, mutexes, etc...)”.

## 7.4. MKLINUXIMG configuration

The clock source and clock event drivers needs special devicetree properties. Mklinuximg from version 2.0.19 and onward adds these when the option `-hrtimers` is set and the requirements mentioned in Section 5.13.6.1 are fulfilled.

## 7.5. Real-time Linux in Buildroot

The “rt-tests” package contains the rt-tests testsuite. The testsuite contains programs to test various real time Linux features. Under “Debugging, profiling and benchmark” select:

- “rt-tests” (BR2_PACKAGE_RT_TESTS)

Another useful package is “schedutils” from the “util-linux” package which provides tools like chrt and taskset for managing CPU affinities and real-time scheduling policies, which are important for tuning a real-time kernel. Under “Target packages, System tools” select:

- “util-linux” (BR2_PACKAGE_UTIL_LINUX)

now its possible to select “schedutils”:

- “schedutils” (BR2_PACKAGE_LINUX_SCHEDUTILS)

under “Target packages, System tools, util-linux”.

### 7.5.1. cyclictest

A commonly used tool for measuring latencies is called cyclictest, which measures latency by setting up a number of measuring threads. These threads use a periodic (cyclic) timer which they set at a specified wakeup time. The latency is then calculated as the difference between the actual wakeup time and the programmed wakeup time.

```
Example:
        $ cyclictest -S -p80 -i1000 -l10000 -q
        WARN: stat /dev/cpu_dma_latency failed: No such file or directory
        T: 0 (136) P:80 I:1000 C: 10000 Min: 59 Act: 75 Avg: 74 Max: 272
        T: 1 (137) P:80 I:1500 C: 6679 Min: 59 Act: 64 Avg: 73 Max: 164
        T: 2 (138) P:80 I:2000 C: 5009 Min: 60 Act: 83 Avg: 70 Max: 256
        T: 3 (139) P:80 I:2500 C: 4007 Min: 60 Act: 92 Avg: 86 Max: 202

        Breakdown of the most important numbers:
        - T: Thread ID (Typically one measuring thread per CPU).
        - P: Priority of the real-time thread.
        - I: Programmed wake-up interval in microseconds (µs).
        - Min: Minimum latency that was measured (in µs).
        - Act: Latency measured during the last completed iteration (in µs).
        - Max: Maximum latency that was measured (in µs).
```

**About the warning when starting cyclictest**

```
WARN: stat /dev/cpu_dma_latency failed: No such file or directory
```

The warning is printed because the cyclictest program fails to probe a capability in the system that is not supported on SPARC Linux.

## 7.6. Tuning

Tuning a real-time kernel is often required to optimize latency and performance based on the system and application requirements. An out-of-box real-time kernel may exhibit higher-than-desired latency or suboptimal performance under load, but tuning can improve results. Here follows a couple of techniques that can be used.

**CPU isolation**
Use the isolcpus kernel parameter to reserve CPUs for specific tasks, reducing interference from the general scheduler.

**IRQ affinity**
Adjust IRQ affinity to move interrupt handling to non-critical CPUs, minimizing context switches on real-time CPUs.

**Changing task priority and affinity**

Assign real-time priority, for example, using the chrt command:

```
chrt -f 5 program
```

The program will get the SCHED_FIFO policy which have a higher priority compared to the default priority (SCHED_OTHER) and the priority level range is 0-99. In a similar way its possible to pin a task to one or several CPUs, for example, using the taskset program:

```
taskset -c 1-3 chrt -f 5 program
```

The program will get real-time priority (SCHED_FIFO) and be pinned to CPU1-3 leaving CPU0 free.

Both **chrt** and **taskset** are available from the “schedutils” package in Buildroot, see Section 7.5 for instructions on how to enable it.

**NOHZ_FULL**
Many guides (available online) on tuning Real-time Linux recommend using the NOHZ_FULL feature, which allows CPUs to run without the periodic system tick which reduce jitter introduced by timer interrupts. This feature is not supported on SPARC32.

# 8. Frequently Asked Questions

## 8.1. What causes DMA-API warnings on addresses 0xf0005000-0xf0007fff?

This memory region is part of the kernel text section and is used to store the trap tables for CPU1-CPU3. If less than four CPUs are used, the memory containing the trap tables for the unused CPUs is freed and put in the general memory pool. The DMA-API debug functionality (enabled by compiling the kernel with the DMA_API_DEBUG option) issues a warning when memory from the kernel text section is used as part of a DMA operation. This coarse check causes it to issue false warnings whenever the memory that formerly held the trap tables is used.

Here is an example of such a warning message:

```
grlib-greth ffd0f610: DMA-API: device driver maps memory from kernel text or rodata [addr=f0006000] [len=1352]
```

## 8.2. Initialisation of unavailable ranges when using Sparse memory

The Sparse memory model presents the memory with sections and if the available memory does not align with the number of sections then Linux will identify a hole at the end of the memory range and make sure to initialise its internal representation of this range in the memory map. The size of the available memory depends on how much memory Mklinuximg retain after boot as it keeps memory for OPENPROM operations.

Here is an example of such a info message:

```
On node 0, zone HighMem: 9 pages in unavailable ranges
```

# 9. Support

For support contact the 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.

Frontgrade Gaisler AB
Kungsgatan 12
411 19 Göteborg
Sweden
frontgrade.com/gaisler
sales@gaisler.com
T: +46 31 7758650
F: +46 31 421407